

ESTTA Tracking number: **ESTTA404691**

Filing date: **04/20/2011**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE TRADEMARK TRIAL AND APPEAL BOARD

Proceeding	91193335
Party	Defendant RStudio, Inc.
Correspondence Address	CHARLES E. WEINSTEIN FOLEY HOAG LLP 155 SEAPORT BLVD, STE 1600 BOSTON, MA 02210-2600 UNITED STATES ARufo@foleyhoag.com, JHuston@foleyhoag.com, USTRademark@foleyhoag.com, cweinstein@foleyhoag.com, jjarvis@foleyhoag.com
Submission	Defendant's Notice of Reliance
Filer's Name	Anthony E. Rufo
Filer's e-mail	arufo@foleyhoag.com
Signature	/Anthony E. Rufo/
Date	04/20/2011
Attachments	Applicant's Notice of Reliance.pdf (7 pages)(24558 bytes) Exhibit B.pdf (8 pages)(36688 bytes) Exhibit C.pdf (75 pages)(7211579 bytes) Exhibit D.pdf (52 pages)(1855702 bytes)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE TRADEMARK TRIAL AND APPEAL BOARD

EMBARCADERO TECHNOLOGIES, INC.,

Opposer,

v.

RSTUDIO, INC.

Applicant.

Opposition No. 91193335

Applications S.N.

77/691980

77/691984

77/697987

APPLICANT'S NOTICE OF RELIANCE

Pursuant to 37 CFR §§ 2.120(j)(1); 2.120(j)(5); and 2.122(e), Applicant RStudio, Inc. ("Applicant"), by its attorneys, hereby submits through its Notice of Reliance that it will or may rely upon and make of record with this opposition proceeding the items set forth below.

- A. Specified portions of the 30(b)(6) discovery deposition of Opposer Embarcadero Technologies, Inc. ("Opposer"), dated November 4, 2010, as well as any applicable exhibit discussed in the specified portions. Applicant intends to rely on pp. 1-4, 12-27, 40-50, 53-54, 60-67, 70-80, and 93-98 and deposition Exhibit 6. Pursuant to the Board's Standardized Protective Agreement, certain portions of the above referenced excerpts have been designated by Opposer as "Trade Secret/Commercially Sensitive" and, accordingly, have been redacted from the transcript for public filing. A Notice of Reliance without redaction shall be filed concurrently under seal. A true and correct copy of these excerpts from the certified transcript and a copy of the deposition exhibit are attached hereto as Exhibit A.
- B. Opposer's Amended Responses to Applicant's Amended First Set of Interrogatories, specifically, Opposer's Response to Interrogatory No. 14. A true and correct copy is

attached hereto as Exhibit B.

- C. Chapters 19 (pp. 369-372), 20 (pp. 373-433), and 23 (pp. 463-468) of R In a Nutshell: A Desktop Quick Reference by Joseph Adler, released in December 2009 by O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, upon which applicant intends to rely to demonstrated statistical computing functions inherent in the R computing language. A true and correct copy is attached hereto as Exhibit C.
- D. Applicant's current website (www.rstudio.org) as of April 18, 2011, upon which Applicant intends to rely to demonstrate: (1) Applicant's current use of its RSTUDIO trademark; and (2) Applicant's direct offering of its RSTUDIO statistical computing software. A true and correct copy is attached hereto as Exhibit D.
- E. A collection of fifty (50) web pages which demonstrate that the term "studio" is commonly used in the names of software products which are comparable to products offered by the Opposer or the product offered by the Applicant. For the convenience of the Board, a summary presenting the contents of these voluminous web pages has been included. True and correct copies are attached hereto as Exhibit E.

1. <http://www.activestate.com/activeperl-pro-studio>
2. <http://www.activestate.com/activetcl-pro-studio>
3. <http://www.apтана.com/>
4. <http://www.aquafold.com/>
5. http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725
6. <http://www.atstechnologies.co.uk/>
7. http://www.yessoftware.com/products/product_detail.php?product_id=1
8. <http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/>
9. <http://www.ivrsoft.com/ct-developer-studio.htm>
10. <http://datafeedstudio.com/>
11. <http://www.devart.com/dbforge/mysql/studio/>

12. <http://www.softwarefx.com/devstudio/>
13. http://www.gadwin.com/cad_programs.htm
14. <http://www.eiffel.com/products/studio/>
15. <http://libre.adacore.com/libre/tools/gps/>
16. <http://usa.autodesk.com/adsk/servlet/pc/index?id=11179508&siteID=123112>
17. <http://www.ohloh.net/p/gtkstudio>
18. <http://www-01.ibm.com/software/data/optim/data-studio/>
19. <http://software.intel.com/en-us/intel-parallel-studio-home/>
20. <http://ironpythonstudio.codeplex.com/>
21. http://www.redhat.com/developer_studio/
22. <http://confluence.atlassian.com/display/JIRASTUDIO>
23. <http://www.thekompany.com/projects/kdestudio/>
24. <http://www.liquid-technologies.com/>
25. <http://www.ufasoft.com/lisp/>
26. <http://lua-studio.luaforge.net/>
27. http://www.enterprisedb.com/products/postgres_plus_as/overview.do#TabOverview
28. <http://www.omnis.net/products/studio/index.html?detail=overview>
29. <http://www-01.ibm.com/software/data/optim/development-studio/>
30. <http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html>
31. <http://www.devert.com/dbforge/oracle/studio/>
32. <http://www.cayoren.com/Perl-Studio/>
33. <http://www.cayoren.com/PHP-Studio/>
34. <http://www.qppstudio.net/documentation.htm>
35. <http://www.realsoftware.com/realstudio/>
36. https://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=B2425
37. <http://www.microsoft.com/robotics/>
<http://www.abb.com/product/seitp327/78fb236cae7e605dc1256f1e002a892c.aspx>
38. <http://www.abb.com/product/seitp327/78fb236cae7e605dc1256f1e002a892c.aspx>
39. <http://support.sas.com/rnd/app/studio/studio.html>
40. <http://www-01.ibm.com/software/data/informix/serverstudio/>
41. <http://www.synsys.com/Products/stStudio/index.html>

42. <http://www.codesegment.com/products.htm>
43. <http://www.sqlstudio.com/>
44. <http://www.sqlmanager.net/products/studio/oracle>
45. <http://susestudio.com/>
46. <http://kakunin.chat.ru/tcldev/>
47. <http://www.pragsoft.com/>
48. http://www.microsoft.com/visualstudio/en-us/visual-studio-2010-launch?WT.mc_id=SEARCH&WT.srch=1
49. <http://www-01.ibm.com/software/integration/wsadie/>
50. <http://shop.zend.com/en/zend-studio-for-eclipse.html?gclid=CLaPucz6aMCFd9n5QodISw92Q>

F. A collection of fifty (50) web pages which demonstrate that the term “ER” as applied to relational databases and database software means “Entity Relationship.” For the convenience of the Board, a summary presenting the contents of these voluminous web pages has been included. True and correct copies are attached hereto as Exhibit F.

1. <http://www.pera.net/Methodologies/ARIS/ARIS.html>
2. http://www.umsl.edu/~sauterv/analysis/er/er_intro.html
3. <http://wofford-ecs.org/DataAndVisualization/ermodel/material.htm>
4. http://www.conceptdraw.com/en/products/cd5/ap_er_diagram.php
5. http://media.visual-paradigm.com/media/documents/dbva40dg/pdf/dbva_designer_guide_chapter4.pdf
6. <http://databases.about.com/cs/specificproducts/g/er.htm>
7. <http://searchsqlserver.techtarget.com/definition/entity-relationship-model>
8. http://www.worldlingo.com/ma/enwiki/en/Entity-relationship_model
9. http://www.pcmag.com/encyclopedia_term/0,2542,t=entity+relationship+model&i=42662,00.asp
10. <http://www.scribd.com/doc/3053988/ER-Diagram-convention>
11. <http://www.dulcian.com/FAQ/Designer%20FAQ%20page.htm>
12. <http://cisnet.baruch.cuny.edu/holowczak/classes/9440/entityrelationship/>
13. <http://www.downloadatoz.com/toplight/entity+relationship+diagram.html>
14. <http://msdn.microsoft.com/en-us/library/aa224825%28SQL.80%29.aspx>
15. http://www.sethi.org/classes/cet415/lab_notes/lab_03.html

16. <http://www.utexas.edu/its/archive/windows/database/datamodeling/index.html>
17. <http://bit.csc.lsu.edu/~chen/chen.html>
18. <http://www.aquafold.com/er-modeler.html>
19. <http://www.tmssoftware.com/site/tmsdm.asp>
20. <http://www.casestudio.com/enu/products.aspx>
21. http://www.sqlmaestro.com/products/mysql/maestro/tour/database_designer/
22. http://www.sparxsystems.com/enterprise_architect_user_guide/modeling_languages/entity_relationship_diagrams_e.html
23. <http://www.visual-paradigm.com/product/vpuml/provides/dbmodeling.jsp>
24. <http://www.orafaq.com/tools/heraut/dezign.htm>
25. <http://www.information-management.com/infodirect/20030123/6268-1.html>
26. <http://www.devarticles.com/c/a/Development-Cycles/Entity-Relationship-Modeling/>
27. <http://en.allexperts.com/q/Oracle-1451/Entity-Relationship-Diagrammer.htm>
28. <http://bit.csc.lsu.edu/~chen/pdf/english.pdf>
29. http://en.wikipedia.org/wiki/Entity-relationship_model#ER_diagramming_tools
30. <http://www.codewalkers.com/c/a/Database-Code/Relationships-Entities-and-Database-Design/2/>
31. <http://www.amazon.com/dp/3540582177>
32. <http://www.crcpress.com/product/isbn/9780849315480>
33. <http://www.smartdraw.com/resources/tutorials/entity-relationship-diagrams/>
34. <http://www.ncgia.ucsb.edu/giscc/units/u045/u045.html>
35. http://www.techdictionary.com/search_action.lasso
36. http://it.toolbox.com/wiki/index.php/Entity_Relationship_Diagram
37. http://it.toolbox.com/wiki/index.php/Entity_Relationship_Diagram
38. <http://www.computeruser.com/dictionary/>
39. <http://dictionary.reference.com/browse/ER>
40. <http://www.visual-paradigm.com/VPGallery/datamodeling/EntityRelationshipDiagram.html>
41. <http://www.oppapers.com/essays/Journal-Entry-Reversal-Entity-Relationship-Diagram/96308>
42. http://www.computingstudents.com/notes/database_systems/entities_entity_relationship_er_modelling.php
43. <http://www.edrawsoft.com/chen-erd.php>
44. <http://www.datanamic.com/dezign/index.html>
45. <http://searchcrm.techtarget.com/answer/Data-modeling-Dimensional-vs-E-R>
46. <http://dret.net/glossary/er>

47. http://www.ibm.com/developerworks/rational/library/content/03July/2500/2785/2785_uml.pdf
48. http://bit.csc.lsu.edu/~chen/pdf/Chen_Pioneers.pdf
49. http://www.ischool.drexel.edu/faculty/song/publications/p_Jcse-erd.PDF
50. <http://docs.aquafold.com/docs-er-diagram.html>

- G. A portion of Opposer's website (www.embarcadero.com) which lists software products sold by Opposer, including those branded with the ER/STUDIO mark at issue in this proceeding, which demonstrates how Embarcadero categorizes its various products. A true and correct copy is attached hereto as Exhibit G.
- H. A portion of Opposer's website (www.embarcadero.com) which lists the current prices charged as of April 19, 2011 for various ER/STUDIO products. A true and correct copy is attached hereto as Exhibit H.
- I. A Wikipedia article titled "Comparison of Statistical Packages" that lists a number of statistical software packages and their various features and which is demonstrative of the statistical computing software category as it exists in commerce. A true and correct copy is attached hereto as Exhibit I.

Respectfully submitted,

RSTUDIO, INC.,

/Anthony E. Rufo/

Julia Huston

Charles E. Weinstein

Joshua S. Jarvis

Anthony E. Rufo

Foley Hoag LLP

155 Seaport Boulevard

Boston, MA 02210

Tel. 617/832-1000

jhuston@foleyhoag.com

cweinstein@foleyhoag.com

jjarvis@foleyhoag.com

arufo@foleyhoag.com

Attorneys for Applicant

Dated: April 20, 2011

CERTIFICATE OF SERVICE

I hereby certify that I have this day served a true copy of the above-identified Notice of Reliance upon Opposer's attorneys of record:

Martin R. Greenstein
Mariela P. Vidolova
TechMark A Law Corporation
4820 Harwood Road, 2nd Floor
San Jose, CA 95124-5273

via First-Class Mail and e-mail to MRG@TechMark.com and MPV@TechMark.com.

/Anthony E. Rufo/

Anthony E. Rufo

DATED: April 20, 2011

Exhibit B

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE TRADEMARK TRIAL AND APPEAL BOARD**

**EMBARCADERO TECHNOLOGIES,
INC.**

Opposer,

v.

RSTUDIO, INC.

Applicant.

Opposition No. 91-193,335

Trademarks: RSTUDIO

**Serial Nos.: 77/691,980
77/691,984
77/691,987**

**OPPOSER’S OBJECTIONS AND AMENDED RESPONSES TO
APPLICANT’S AMENDED FIRST SET OF INTERROGATORIES TO OPPOSER**

Pursuant to Federal Rules of Civil Procedure 26 and 33 and Trademark Rule 2.120, Opposer, EMBARCADERO TECHNOLOGIES, INC. ("Embarcadero" or "Opposer") makes the following objections and responses to Applicant's Amended First Set of Interrogatories to Opposer by Applicant, RSTUDIO, INC. ("RStudio" or "Applicant").

GENERAL OBJECTIONS

The following General Objections are applicable to, and are hereby incorporated by reference into each of Opposer’s Responses and Objections to the specific discovery requests propounded herein. The provision of any response, in whole or in part, is not intended to and does not waive any of these General Objections, or any specific objection with respect to any discovery request.

1. Opposer objects to each and every discovery request propounded herein by Applicant to the extent that it is overly broad, vague, ambiguous, cumbersome, unduly burdensome, and/or

seeks information that is neither relevant nor reasonably calculated to lead to the discovery of admissible evidence.

2. Opposer objects to each and every discovery request propounded herein to the extent it imposes or attempts to impose greater obligations on Opposer than those authorized by the Federal Rules of Civil Procedure and the Trademark Rules of Practice, or which are inconsistent therewith

3. Opposer objects to each and every discovery request propounded herein that seeks legal opinions or conclusions, or that is more properly propounded as a request for admission.

4. Opposer objects to each and every discovery request propounded herein to the extent that they mis-characterize Opposer, Opposer's products, services, or any fact alleged by Opposer. By responding to these discovery requests, Opposer does not admit that Applicant's characterizations are accurate or correct.

5. Opposer objects to each and every discovery request propounded herein to the extent that it requests information or responses relating to information to which Opposer has objected.

6. Opposer objects to each and every discovery request propounded herein to the extent it purports to demand information not in the possession, custody or control of Opposer, or to require Opposer to undertake efforts to respond thereto that are not reasonably calculated to ascertain information responsive to one or more specific discovery requests propounded herein.

7. Opposer objects to each and every discovery request propounded herein on the basis that it is not stated simply or directly, and/or uses compound terms.

8. Opposer objects to each and every discovery request propounded herein to the extent that it contains interdependent, compound issues, and/or is premised on a fact that is denied, and/or contains words, terms or phrases that are vague and ambiguous, and therefore not subject to succinct

response.

9. Opposer objects to each and every discovery request propounded herein to the extent that and on the basis that it goes beyond the permissible scope of discovery, including but not limited to the fact that it goes beyond the marks and goods/services at issue in this proceeding, and/or is not limited to commerce which may be lawfully regulated by Congress. To the extent any response to any such discovery request is provided, it is provided with respect to the United States, use in the United States, or use in commerce which may be lawfully regulated by Congress.

10. Opposer objects to each and every overly expansive and vague “Definition” or other term or phrase which Applicant purports to impose herein and which renders the discovery sought irrelevant, not likely to lead to relevant information, overly burdensome and beyond the scope of permissible discovery, including, but not limited to, the overly broad and overly expansive definitions of “Opposer” and “Embarcadero Technologies, Inc.”, of “Applicant” and “Rstudio, Inc.”, of “Person”, of the “Opposer’s Mark”, and each and every other overly broad, expansive and improper definition or term.

11. Opposer objects to each and every discovery request propounded herein on the basis that it is vague, uncertain, ill-defined and not capable of response because of the vagueness and uncertainty created by overly broad definitions and instructions.

12. Opposer objects to each and every discovery request propounded herein to the extent that it asks a question that is not reasonably understood or subject to interpretation, and thus is not capable of response.

13. Opposer objects to each and every discovery request propounded herein to the extent that it asks for information about future plans, non-public information, confidential information,

confidential business information, trade secrets or otherwise protected information unless such production is pursuant to an appropriate protective order for the information at question.

14. Opposer objects to each and every discovery request propounded herein to the extent that it asks for information that is protected from discovery by the attorney-client privilege, the attorney work-product doctrine or any other applicable privilege. Opposer objects to divulging any such information in response to discovery requests propounded herein. To the extent any such information is or may be divulged in response to discovery requests propounded herein, the divulging of such information is inadvertent and is not deemed to be a waiver of the privilege in question, or of any other applicable privilege, with respect to the divulged information or any other information.

15. Opposer objects to each and every discovery request propounded herein to the extent that it asks for confidential information. Pursuant to Rule 2.116(g) of the Trademark Rules of Practice, all produced documents requiring confidential treatment shall be subject to the TTAB Standard Protective Order and marked in accordance with its provisions.

16. Opposer's internal inquiries and discovery are ongoing. Opposer therefore objects to Applicant's discovery requests propounded herein to the extent that they cut off or purport or may have the effect of cutting off Opposer's right to supplement its responses. Opposer reserves its right to supplement its responses up to and including time of trial.

17. These general objections are applicable to and are incorporated into each specific response herein, with or without further reference. Insertion of specific objections in the response to any particular discovery request shall not be construed as a waiver of such objection in any other response.

RESPONSE:

Subject to the foregoing objections, the term “Studio” is simply a metaphor sometimes used in the software industry to suggest a collection of tools or programs in a way somewhat analogous to what one may find in an artist’s “Studio” or a music “Studio”.

INTERROGATORY NO. 14

Explain whether STUDIO is or is not descriptive as applied to any of Opposer’s products associated with Opposer’s Mark.

RESPONSE:

Subject to the foregoing objections, the term “Studio” is simply a metaphor sometimes used in the software industry to suggest a collection of tools or programs in a way somewhat analogous to what one may find in an artist’s “Studio” or a music “Studio”.

INTERROGATORY NO. 15

Identify each product offered under Opposer’s Mark that is used in connection with entity relationship modeling.

RESPONSE:

Subject to the foregoing general objections and to the vague and unclear meaning of the question, Opposer response that when first adopted the products under its ER/STUDIO mark included features and functionality in the field of entity relationship modeling, *inter alia*. Since then

Dated: November 1, 2010

AS TO RESPONSES

Embarcadero Technologies, Inc.,

By: _____

Name:

JASON TIVET

Title:

DIRECTOR - MODELING &
DESIGN SOLUTIONS

Dated: September 30, 2010

Re-signed November 1, 2010

AS TO OBJECTIONS:

By: /Martin R Greenstein/

Martin R. Greenstein

Mariela P. Vidolova

TechMark a Law Corporation

4820 Harwood Road, 2nd Floor

San Jose, CA 95124-5237

Tel: 408-266-4700 Fax: 408-850-1955

E-mail: MRG@TechMark.com

Attorneys for Opposer

CERTIFICATE OF SERVICE

I hereby certify that a copy of the foregoing **OPPOSER'S OBJECTIONS AND AMENDED RESPONSES TO APPLICANT'S AMENDED FIRST SET OF INTERROGATORIES TO OPPOSER** is being served by first class mail postage prepaid on this 1st day of November, 2010, on Applicant's attorneys:

Julia Huston
Charles E. Weinstein
Anthony E. Rufo
FOLEY HOAG LLP
155 Seaport Blvd, Ste 1600
Boston, MA 02210-2600
Attorneys for Applicant

/Martin R Greenstein/
Martin R. Greenstein

Exhibit C



R

IN A NUTSHELL

A Desktop Quick Reference

O'REILLY®

Joseph Adler

R

IN A NUTSHELL

R

IN A NUTSHELL

Joseph Adler

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

R in a Nutshell

by Joseph Adler

Copyright © 2010 Joseph Adler. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://my.safaribooksonline.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor: Mike Loukides
Production Editor: Sumita Mukherji
Production Services: Newgen North
America, Inc.

Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Printing History:

December 2009: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *R in a Nutshell*, the image of a harpy eagle, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.



This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 978-0-596-80170-0

[M]

[3/10]

1267050233



When designing an experiment, it's often helpful to know how much data you need to collect to get a statistically significant sample (or, alternatively, the maximum significance of results that can be calculated from a given amount of data). R provides a set of functions to help you calculate these amounts.

Experimental Design Example

Suppose that you want to test the efficacy of a new drug for treating depression. A common score used to measure depression is the Hamilton Rating Scale for Depression (HAMD). This measure varies from 0 to 48, where higher values indicate increased depression. Let's consider two different experimental design questions. First, suppose that you had collected 50 subjects for the study and split them into two groups of 25 people each. What difference in HAMD scores would you need to observe in order for the results to be considered statistically significant?

We assume a standard deviation of 8.9 for this experiment.* We'll also assume that we want a power of .95 for the experiment (meaning that the probability of a Type II error is less than .05). To calculate the minimum statistically significant difference in R, we could use the following expression:

* Number from http://www.fda.gov/OHRMS/DOCKETS/ac/07/slides/2007-4273s1_05.pdf.

```
> power.t.test(power=.95, sig.level=.05, sd=8.9, n=25)
```

```
Two-sample t test power calculation
```

```
      n = 25
  delta = 9.26214
      sd = 8.9
sig.level = 0.05
  power = 0.95
alternative = two.sided
```

NOTE: n is number in *each* group

According to the output, the difference in means between the two groups would need to be at least 9.26214 to be significant at this level. Suppose that we doubled the number of subjects. What difference would be considered significant?

```
> power.t.test(power=.95, sig.level=.05, sd=8.9, n=50)
```

```
Two-sample t test power calculation
```

```
      n = 50
  delta = 6.480487
      sd = 8.9
sig.level = 0.05
  power = 0.95
alternative = two.sided
```

NOTE: n is number in *each* group

As you can see, the power functions can be very useful for designing an experiment. They can help you to estimate, in advance, how large a difference you need to see between groups to get statistically significant results.

t-Test Design

If you are designing an experiment where you will use a *t*-test to check the significance of the results (typically, an experiment where you calculate the mean value of a random variable for a “test” population and a “control” population), then you can use the `power.t.test` function to help design the experiment:

```
power.t.test(n = NULL, delta = NULL, sd = 1, sig.level = 0.05,
             power = NULL,
             type = c("two.sample", "one.sample", "paired"),
             alternative = c("two.sided", "one.sided"),
             strict = FALSE)
```

For this function, `n` specifies the number of observations (per group); `delta` is the true difference in means between the groups; `sd` is the true standard deviation of the underlying distribution; `sig.level` is the significance level (Type I error probability); `power` is the power of the test (1 - Type II error probability); `type` specifies whether the test is one sample, two sample, or paired; `alternative` specifies whether the test is one or two sided; and `strict` specifies whether to use a strict interpretation in the two-sided case. This function will calculate either `n`, `delta`, `sig.level`, `sd`, or `power`,

That's right, the estimate is over 2,724 at bats. So, let's ask the opposite question: what is the confidence we can have in the results? Let's fix sig.level=0.05 and power=0.95:

```
> power.prop.test(n=10, p1=.260, p2=.300, power=.95,  
+ sig.level=NULL, alternative="one.sided")
```

Two-sample comparison of proportions power calculation

```
n = 10  
p1 = 0.26  
p2 = 0.3  
sig.level = 0.9256439  
power = 0.95  
alternative = one.sided
```

NOTE: n is number in *each* group

```
> power.prop.test(n=10, p1=.260, p2=.300, power=NULL,  
+ sig.level=.05, alternative="one.sided")
```

Two-sample comparison of proportions power calculation

```
n = 10  
p1 = 0.26  
p2 = 0.3  
sig.level = 0.05  
power = 0.07393654  
alternative = one.sided
```

NOTE: n is number in *each* group

With significance levels that low, I think it's safe to say that most of these situational statistics are nonsense.

ANOVA Test Design

If you are designing an experiment where you will be using ANOVA, you can use the `power.anova.test` function:

```
power.anova.test(groups = NULL, n = NULL,  
                 between.var = NULL, within.var = NULL,  
                 sig.level = 0.05, power = NULL)
```

For this function, `groups` specifies the number of groups, `n` specifies the number of observations (per group), `between.var` is the variance between groups, `within.var` is the variance within groups, `sig.level` is the significance level (Type I error probability), and `power` is the power of the test (1 - Type II error probability). This function will calculate either `groups`, `n`, `sig.level`, `between.var`, `power`, `within.var`, or `sig.level`, depending on the input. You must specify exactly six of these parameters, and the remaining argument must be null; this is the value that the function calculates.



A *regression model* shows how a continuous value (called the *response variable*, or the *dependent variable*) is related to a set of other values (called the *predictors*, *stimulus variables*, or *independent variables*). Often, a regression model is used to predict values where they are unknown. For example, warfarin is a drug commonly used as a blood thinner or anticoagulant. A doctor might use a regression model to predict the correct dose of warfarin to give a patient based on several known variables about the patient (such as the patient's weight). Another example of a regression model might be for marketing financial products. An analyst might estimate the average balance of a credit card customer (which, in turn, affects the expected revenue from that customer).

Sometimes, a regression model is simply used to explain a phenomenon, but not to actually predict values. For example, a scientist might suspect that weight is correlated to consumption of certain types of foods, but wants to adjust for a variety of factors, including age, exercise, genetics (and, hopefully, other factors). The scientist could use a regression model to help show the relationship between weight and food consumed by including other variables in the regression. Models can be used for many other purposes, including visualizing trends, analysis of variance tests, and testing variable significance.

This chapter looks at regression models in R; classification models are covered in Chapter 21. To show how to use statistical models in R, I will start with the simplest type of model: linear regression models. (Specifically, I'll use the least squares method to estimate coefficients.) I'll show how to build, evaluate, and refine a model in R. Then I'll describe functions in R for building more sophisticated types of models.

Example: A Simple Linear Model

A linear regression assumes that there is a linear relationship between the response variable and the predictors. Specifically, a linear regression assumes that a response variable y is a linear function of a set of predictor variables x_1, x_2, \dots, x_n .

As an example, we're going to look at how different metrics predict the runs scored by a baseball team.² Let's start by loading the data for every team between 2000 and 2008. We'll use the SQLite database that we used in Chapter 14 and extract the fields we want using an SQL query:

```
> library(RSQLite)
> drv <- dbDriver("SQLite")
> con <- dbConnect(drv,
+ dbname=paste(.Library, "/nutshell/data/bb.db", sep=""))
> team.batting.00to08 <- dbGetQuery(con,
+ paste(
+ 'SELECT teamID, yearID, R as runs, ',
+ ' H-"2B"-"3B"-HR as singles, ',
+ ' "2B" as doubles, "3B" as triples, HR as homeruns, ',
+ ' BB as walks, SB as stolenbases, CS as caughtstealing, ',
+ ' HBP as hitbypitch, SF as sacrificeflies, ',
+ ' AB as atbats ',
+ ' FROM Teams ',
+ ' WHERE yearID between 2000 and 2008'
+ )
+ )
```

² Or, if you'd like, you can just load the file from the nutshell package:

```
> library(nutshell)
> data(team.batting.00to08)
```

Because this is a book about R and not a book about baseball, I renamed the common abbreviations to more intuitive names for plays. Let's look at scatter plots of runs versus each other variable, so that we can see which variables are likely to be most important.

We'll create a data frame for plotting, using the `make.groups` function:

```
> attach(team.batting.00to08);
> forplot <- make.groups(
+ singles = data.frame(value=singles, teamID, yearID, runs),
+ doubles = data.frame(value=doubles, teamID, yearID, runs),
+ triples = data.frame(value=triples, teamID, yearID, runs),
+ homeruns = data.frame(value=homeruns, teamID, yearID, runs),
+ walks = data.frame(value=walks, teamID, yearID, runs),
+ stolenbases = data.frame(value=stolenbases, teamID, yearID, runs),
+ caughtstealing = data.frame(value=caughtstealing, teamID, yearID, runs),
+ hitbypitch = data.frame(value=hitbypitch, teamID, yearID, runs),
+ sacrificeflies = data.frame(value=sacrificeflies, teamID, yearID, runs)
+ );
> detach(team.batting.00to08);
```

Now, we'll generate the scatter plots using the `xyplot` function:

```
> xyplot(runs~value|which, data=forplot,
+ scales=list(relation="free"),
```

* This example is closely related to the batter runs formula, which was popularized by Pete Palmer and Jim Thorne in the 1984 book *The Hidden Game of Baseball*. The original batter runs formula worked slightly differently: it predicted the number of runs above or below the mean, and it had no intercept. For more about this problem, see [Adler2006].

erent metrics predict the runs scored
ata for every team between 2000 and
used in Chapter 14 and extract the

a/bb.db", sep="")

```
, HR as homeruns, '  
CS as caughtstealing, '  
ceflies, '
```

08'

the nutshell package:

ut baseball, I renamed the common
Let's look at scatter plots of runs
ich variables are likely to be most

ake.groups function:

```
s, teamID, yearID, runs),  
s, teamID, yearID, runs),  
s, teamID, yearID, runs),  
ns, teamID, yearID, runs),  
teamID, yearID, runs),  
ases, teamID, yearID, runs),  
stealing, teamID, yearID, runs),  
tch, teamID, yearID, runs),  
ceflies, teamID, yearID, runs)
```

lot function:

hich was popularized by Pete Palmer
ball. The original batter runs formula
above or below the mean, and it had

```
+ pch=19, cex=.2,  
+ strip=strip.custom(strip.levels=TRUE,  
+ horizontal=TRUE,  
+ par.strip.text=list(cex=.8))  
+ )
```

The results are shown in Figure 20-1. Intuitively, teams that hit a lot of home runs score a lot of runs. Interestingly, teams that walk a lot score a lot of runs as well (maybe even more than teams that score a lot of singles).

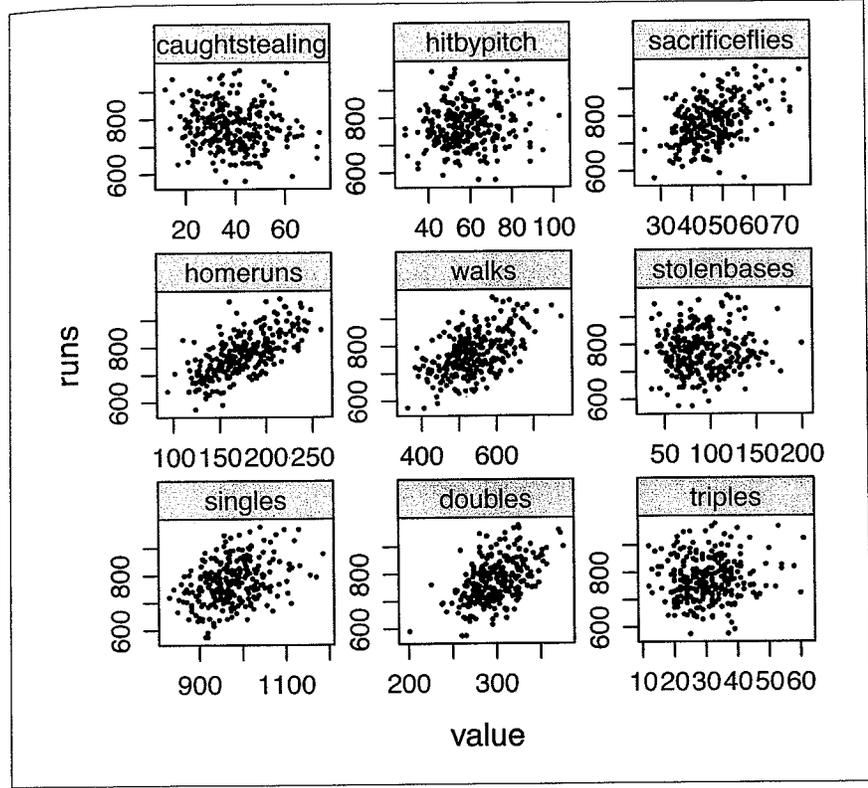


Figure 20-1. Scatter plots: runs as a function of different batter statistics

Fitting a Model

Let's fit a linear model to the data and assign it to the variable runs.mdl. We'll use the lm function, which fits a linear model using ordinary least squares:

```
> runs.mdl <- lm(  
+ formula=runs~singles+doubles+triples+homeruns+  
+ walks+hitbypitch+sacrificeflies+  
+ stolenbases+caughtstealing,  
+ data=team.batting.00to08)
```

R doesn't show much information when you fit a model. (If you don't print the returned object, most modeling functions will not show any information, unless

there is an error.) To get information about a model, you have to use helper functions.

Helper Functions for Specifying the Model

In a formula object, some symbols have special interpretations. Specifically, “+”, “*”, “-”, and “^” are interpreted specially by R. This means that you need to use some helper functions to represent simple addition, multiplication, subtraction, and exponentiation in a model formula. To interpret an expression literally, and not as a formula, use the identity function `I()`. For example, suppose that you want to include only the product of variables `a` and `b` in a formula specification, but not just `a` or `b`. If you specify `a*b`, this is interpreted as `a`, `b`, or `a*b`. To include only `a*b`, use the identity function `I()` to protect the expression `a*b`:

```
lm(y~I(a*b))
```

Sometimes, you would like to fit a polynomial function. Writing out all the terms individually can be tedious, but R provides a short way to specify all the terms at once. To do this, you use the `poly` function to add all terms up to a specified degree:

```
poly(x, ..., degree = 1, coefs = NULL, raw = FALSE)
```

As arguments, the `poly` function takes a vector `x` (or a set of vectors), `degree` to specify a maximum degree to generate, `coefs` to specify coefficients from a previous fit (when using `poly` to generate predicted values), and `raw` to specify whether to use raw and not orthogonal polynomials. For more information on how to specify formulas, see “Formulas” on page 88.

Getting Information About a Model

In R, statistical models are represented by objects; statistical modeling functions return statistical model objects. When you fit a statistical model with most statistical software packages (such as SAS or SPSS) they print a lot of diagnostic information. In R, most statistical modeling functions do not print any information.

If you simply call a model function in R, but don’t assign the model to a variable, the R console will print the object. (Specifically, it will call the generic method `print` with the object generated by the modeling function.) R doesn’t clutter your screen with lots of information you might not want. Instead, R includes a large set of functions for printing information about model objects. This section describes the functions for getting information about `lm` objects. Many of these functions may also be used with other types of models; see the help files for more information.

Viewing the model

For most model functions (including `lm`), the best place to start is with the `print` method. If you are using the R console, you can simply enter the name of the returned object on the console to see the results of `print`:

```
> runs.mdl
```

Call:

ation about a model, you have to use help

the Model

have special interpretations. Specifically, "+" especially by R. This means that you need to use simple addition, multiplication, subtraction, and so on to interpret an expression literally, and not as a formula. For example, suppose that you want to include a and b in a formula specification, but not just as a, b, or a*b. To include only a*b, use the expression a*b:

ynomial function. Writing out all the terms provides a short way to specify all the terms at once. To add all terms up to a specified degree: `ALL, raw = FALSE`

ector x (or a set of vectors), degree to specify coefficients from a previous fit (when `raw` and `raw` to specify whether to use `raw` and `raw` information on how to specify formulas, see

r objects; statistical modeling functions fit a statistical model with most statistical functions print a lot of diagnostic information. `print` not print any information.

it don't assign the model to a variable, `summary` will call the generic `summary` method (the `summary` function.) R doesn't clutter your console if you don't want. Instead, R includes a large set of model objects. This section describes these objects. Many of these functions may have help files for more information.

best place to start is with the `print` function. `print` imply enter the name of the returned object:

```
lm(formula = runs ~ singles + doubles + triples + homeruns +
    walks + hitbypitch + sacrificeflies + stolenbases + caughtstealing,
    data = team.batting.00to08)
```

Coefficients:

(Intercept)	singles	doubles	triples
-507.16020	0.56705	0.69110	1.15836
homeruns	walks	hitbypitch	sacrificeflies
1.47439	0.30118	0.37750	0.87218
stolenbases	caughtstealing		
0.04369	-0.01533		

To show the formula used to fit the model, use the `formula` function:
`formula(x, ...)`

Here is the formula on which the model function was called:

```
> formula(runs.mdl)
runs ~ singles + doubles + triples + homeruns + walks + hitbypitch +
sacrificeflies + stolenbases + caughtstealing
```

To get the list of coefficients for a model object, use the `coef` function:
`coef(object, ...)`

Here are the coefficients for the model fitted above:

```
> coef(runs.mdl)
(Intercept)      singles      doubles      triples
-507.16019759  0.56704867  0.69110420  1.15836091
homeruns        walks        hitbypitch  sacrificeflies
 1.47438916    0.30117665  0.37749717  0.87218094
stolenbases    caughtstealing
 0.04369407   -0.01533245
```

Alternatively, you can use the alias `coefficients` to access the `coef` function.

To get a summary of a linear model object, you can use the `summary` function. The method used for linear model objects is:

```
summary(object, correlation = FALSE, symbolic.cor = FALSE, ...)
```

For the example above, here is the output of the `summary` function:

```
> summary(runs.mdl)

Call:
lm(formula = runs ~ singles + doubles + triples + homeruns +
    walks + hitbypitch + sacrificeflies + stolenbases + caughtstealing,
    data = team.batting.00to08)

Residuals:
    Min       1Q   Median       3Q      Max
-71.9019 -11.8282  -0.4193  14.6576  61.8743

Coefficients:
(Intercept)      singles      doubles      triples
-507.16020    0.56705    0.69110    1.15836
homeruns        walks        hitbypitch  sacrificeflies
 1.47439    0.30118    0.37750    0.87218
stolenbases    caughtstealing
 0.04369   -0.01533

Estimate Std. Error t value Pr(>|t|)
(Intercept) -507.16020    32.34834  -15.678 < 2e-16 ***
singles      0.56705     0.02601   21.801 < 2e-16 ***
```

doubles	0.69110	0.05922	11.670	< 2e-16	***
triples	1.15836	0.17309	6.692	1.34e-10	***
homeruns	1.47439	0.05081	29.015	< 2e-16	***
walks	0.30118	0.02309	13.041	< 2e-16	***
hitbypitch	0.37750	0.11006	3.430	0.000702	***
sacrificeflies	0.87218	0.19179	4.548	8.33e-06	***
stolenbases	0.04369	0.05951	0.734	0.463487	
caughtstealing	-0.01533	0.15550	-0.099	0.921530	

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.21 on 260 degrees of freedom
 Multiple R-squared: 0.9144, Adjusted R-squared: 0.9114
 F-statistic: 308.6 on 9 and 260 DF, p-value: < 2.2e-16

When you print a summary object, the following method is used:

```
print(x, digits = max(3, getOption("digits") - 3),
      symbolic.cor = x$symbolic.cor,
      signif.stars = getOption("show.signif.stars"), ...)
```

Predicting values using a model

To get the vector of residuals from a linear model fit, use the `residuals` function:

```
residuals(object, ...)
```

To get a vector of fitted values, use the `fitted` function:

```
fitted(object, ...)
```

Suppose that you wanted to use the model object to predict values in another data set. You can use the `predict` function to calculate predicted values using the model object and another data frame:

```
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
        interval = c("none", "confidence", "prediction"),
        level = 0.95, type = c("response", "terms"),
        terms = NULL, na.action = na.pass,
        pred.var = res.var/weights, weights = 1, ...)
```

The argument `object` specifies the model returned by the fitting function, `newdata` specifies a new data source for predictions, and `na.action` specifies how to deal with missing values in `newdata`. (By default, `predict` ignores missing values. You can choose `na.omit` to simply return NA for observations in `newdata` with missing values.) The `predict` function can also return confidence intervals for predictions, in addition to exact values; see the help file for more information.

Analyzing the fit

To get the list of coefficients for a model object, use the `coef` function:

```
coef(object, ...)
```

```

22 11.670 < 2e-16 ***
29 6.692 1.34e-10 ***
11 29.015 < 2e-16 ***
9 13.041 < 2e-16 ***
6 3.430 0.000702 ***
9 4.548 8.33e-06 ***
1 0.734 0.463487
) -0.099 0.921530
.01 '*' 0.05 '.' 0.1 ' ' 1

```

0 degrees of freedom
 ted R-squared: 0.9114
 p-value: < 2.2e-16

lowing method is used:

```

igits") - 3),
signif.stars"), ...

```

model fit, use the residuals function:

ed function:

bject to predict values in another data
 late predicted values using the model

```

scale = NULL, df = Inf,
", "prediction"),
", "terms"),
s,
nts = 1, ...)

```

med by the fitting function, newdata
 na.action specifies how to deal with
 et ignores missing values. You can
 ions in newdata with missing values.)
 intervals for predictions, in addition
 nation.

use the coef function:

Here are the coefficients for the model fitted above:

```

> coef(runs.mdl)
      (Intercept)      singles      doubles      triples
      -507.16019759    0.56704867    0.69110420    1.15836091
      homeruns       walks       hitbypitch sacrificeflies
      1.47438916     0.30117665    0.37749717    0.87218094
      stolenbases caughtstealing
      0.04369407     -0.01533245

```

Alternatively, you can use the alias coefficients to access the coef function.

To compute confidence intervals for the coefficients in the fitted model, use the confint function:

```
confint(object, parm, level = 0.95, ...)
```

The argument object specifies the model returned by the fitting function, parm specifies the variables for which to show confidence levels, and level specifies the confidence level. Here are the confidence intervals for the coefficients of the model fitted above:

```

> confint(runs.mdl)
              2.5 %      97.5 %
(Intercept) -570.85828008 -443.4621151
singles      0.51583022    0.6182671
doubles      0.57449582    0.8077126
triples      0.81752968    1.4991921
homeruns     1.37432941    1.5744489
walks        0.25570041    0.3466529
hitbypitch   0.16077399    0.5942203
sacrificeflies 0.49451857    1.2498433
stolenbases  -0.07349342    0.1608816
caughtstealing -0.32152716    0.2908623

```

To compute the influence of different parameters, you can use the influence function:

```
influence(model, do.coef = TRUE, ...)
```

For more friendly output, try influence.measures:

```
influence.measures(model)
```

To get analysis of variance statistics, use the anova function. For linear models, the method used is anova.lm, which has the following form:

```
anova.lm(object, ..., scale = 0, test = "F")
```

By default, *F*-test statistics are included in the results table. You can specify test="F" for *F*-test statistics, test="Chisq" for chi-squared test statistics, test="Cp" for Mallows' C_p statistic, or test=NULL for no test statistics. You can also specify an estimate of the noise variance σ^2 through the scale argument. If you set scale=0 (the default), the anova function will calculate an estimate from the test data. The test statistic and *p*-values compare the mean square for each row to the residual mean square.

Here are the ANOVA statistics for the model fitted above:

```
> anova(runs.mdl)
Analysis of Variance Table

Response: runs

   Df Sum Sq Mean Sq  F value    Pr(>F)
singles  1 215755  215755  400.4655 < 2.2e-16 ***
doubles  1 356588  356588  661.8680 < 2.2e-16 ***
triples  1    237    237    0.4403 0.5075647
homeruns 1 790051  790051 1466.4256 < 2.2e-16 ***
walks    1 114377  114377  212.2971 < 2.2e-16 ***
hitbypitch 1  7396   7396   13.7286 0.0002580 ***
sacrificeflies 1 11726  11726  21.7643 4.938e-06 ***
stolenbases 1   357   357    0.6632 0.4161654
caughtstealing 1    5    5    0.0097 0.9215298
Residuals 260 140078    539
---
```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Interestingly, it appears that triples, stolen bases, and times caught stealing are not statistically significant.

You can also view the effects from a fitted model. The effects are the uncorrelated single degree of freedom values obtained by projecting the data onto the successive orthogonal subspaces generated by the QR-decomposition during the fitting process. To obtain a vector of orthogonal effects from the model, use the effects function:

```
effects(object, set.sign = FALSE, ...)
```

To calculate the variance-covariance matrix from the linear model object, use the vcov function:

```
vcov(object, ...)
```

Here is the variance-covariance matrix for the model fitted above:

```
> vcov(runs.mdl)
      (Intercept)  singles  doubles  triples
(Intercept) 1046.4149572 -6.275356e-01 -6.908905e-01 -0.8115627984
singles      -0.6275356  6.765565e-04 -1.475026e-04  0.0001538296
doubles      -0.6908905 -1.475026e-04  3.506798e-03 -0.0013459187
triples      -0.8115628  1.538296e-04 -1.345919e-03  0.0299591843
homeruns     -0.3190194  2.314669e-04 -3.940172e-04  0.0011510663
walks        -0.2515630  7.950878e-05 -9.902388e-05  0.0004174548
hitbypitch   -0.9002974  3.385518e-04 -4.090707e-04  0.0018360831
sacrificeflies 1.6870020 -1.723732e-03 -2.253712e-03 -0.0051709718
stolenbases  0.2153275 -3.041450e-04  2.871078e-04 -0.0009794480
caughtstealing -1.4370890  3.126387e-04  1.466032e-03 -0.0016038175
      homeruns  walks  hitbypitch  sacrificeflies
(Intercept) -3.190194e-01 -2.515630e-01 -0.9002974059  1.6870019518
singles      2.314669e-04  7.950878e-05  0.0003385518  -0.0017237324
doubles     -3.940172e-04 -9.902388e-05 -0.0004090707  -0.0022537124
triples      1.151066e-03  4.174548e-04  0.0018360831  -0.0051709718
homeruns     2.582082e-03 -4.007590e-04 -0.0008183475  -0.0005078943
walks        -4.007590e-04  5.333599e-04  0.0002219440  -0.0010962381
```

model fitted above:

```
F value Pr(>F)
400.4655 < 2.2e-16 ***
661.8680 < 2.2e-16 ***
0.4403 0.5075647
466.4256 < 2.2e-16 ***
212.2971 < 2.2e-16 ***
13.7286 0.0002580 ***
21.7643 4.938e-06 ***
0.6632 0.4161654
0.0097 0.9215298
```

01 '*' 0.05 '.' 0.1 ' ' 1

bases, and times caught stealing are not

model. The effects are the uncorrelated projecting the data onto the successive QR-decomposition during the fitting effects from the model, use the effects

from the linear model object, use the

model fitted above:

```
doubles triples
-6.908905e-01 -0.8115627984
-1.475026e-04 0.0001538296
3.506798e-03 -0.0013459187
1.345919e-03 0.0299591843
3.940172e-04 0.0011510663
3.902388e-05 0.0004174548
1.090707e-04 0.0018360831
.253712e-03 -0.0051709718
.871078e-04 -0.0009794480
.466032e-03 -0.0016038175
hitbypitch sacrificeflies
.9002974059 1.6870019518
.0003385518 -0.0017237324
.0004090707 -0.0022537124
0018360831 -0.0051709718
0008183475 -0.0005078943
3002219440 -0.0010962381
```

```
hitbypitch -8.183475e-04 2.219440e-04 0.0121132852 -0.0011315622
sacrificeflies -5.078943e-04 -1.096238e-03 -0.0011315622 0.0367839752
stolenbases -2.041656e-06 -1.400052e-04 -0.0001197102 -0.0004636454
caughtstealing 3.469784e-04 6.008766e-04 0.0001742039 -0.0024880710
stolenbases caughtstealing
(Intercept) 2.153275e-01 -1.4370889812
singles -3.041450e-04 0.0003126387
doubles 2.871078e-04 0.0014660316
triples -9.794480e-04 -0.0016038175
homeruns -2.041656e-06 0.0003469784
walks -1.400052e-04 0.0006008766
hitbypitch -1.197102e-04 0.0001742039
sacrificeflies -4.636454e-04 -0.0024880710
stolenbases 3.541716e-03 -0.0050935339
caughtstealing -5.093534e-03 0.0241794596
```

To return the deviance of the fitted model, use the deviance function:

```
deviance(object, ...)
```

Here is the deviance for the model fitted above (though this value is just the residual sum of squares in this case because runs.mdl is a linear model):

```
> deviance(runs.mdl)
[1] 140077.6
```

Finally, to plot a set of useful diagnostic diagrams, use the plot function:

```
plot(x, which = c(1:3,5),
     caption = list("Residuals vs Fitted", "Normal Q-Q",
                   "Scale-Location", "Cook's distance",
                   "Residuals vs Leverage",
                   expression("Cook's dist vs Leverage " * h[ii] / (1 - h[ii]))),
     panel = if(add.smooth) panel.smooth else points,
     sub.caption = NULL, main = "",
     ask = prod(par("mfcol")) < length(which) && dev.interactive(),
     ...,
     id.n = 3, labels.id = names(residuals(x)), cex.id = 0.75,
     qqline = TRUE, cook.levels = c(0.5, 1.0),
     add.smooth = getOption("add.smooth"), label.pos = c(4,2),
     cex.caption = 1)
```

This function shows the following plots:

- Residuals against fitted values
- A scale-location plot of $\sqrt{| \text{residuals} |}$ against fitted values
- A normal Q-Q plot
- (Not plotted by default) A plot of Cook's distances versus row labels
- A plot of residuals against leverages
- (Not plotted by default) A plot of Cook's distances against leverage/(1 - leverage)

There are many more functions available in R for regression diagnostics; see the help file for influence.measures for more information on many of these.

Refining the Model

Often, it is better to use the `update` function to refit a model. This can save you some typing if you are using R interactively. Additionally, this can save on computation time (for large data sets). You can run `update` after changing the formula (perhaps adding or subtracting a term) or even after changing the data frame.

For example, let's fit a slightly different model to the data above. We'll omit the variable `sacrificeflies` and add `0` as a variable (which means to fit the model with no intercept):

```
> runs.mdl2 <- update(runs.mdl, formula=runs ~ singles + doubles +
+ triples + homeruns + walks + hitbypitch +
+ stolenbases + caughtstealing + 0)
> runs.mdl2
```

Call:

```
lm(formula = runs ~ singles + doubles + triples + homeruns +
walks + hitbypitch + stolenbases + caughtstealing - 1,
data = team.batting.00to08)
```

Coefficients:

singles	doubles	triples	homeruns
0.29823	0.41280	0.95664	1.31945
walks	hitbypitch	stolenbases	caughtstealing
0.21352	-0.07471	0.18828	-0.70334

Details About the `lm` Function

Now that we've seen a simple example of how models work in R, let's describe in detail what `lm` does and how you can control it. A linear regression model is appropriate when the response variable (the thing that you want to predict) can be estimated from a linear function of the predictor variables (the information that you know). Technically, we assume that:

$$y = c_0 + c_1 x_1 + c_2 x_2 + \dots + c_n x_n + \varepsilon$$

where y is the response variable, x_1, x_2, \dots, x_n are the predictor variables (or predictors), c_1, c_2, \dots, c_n are the *coefficients* for the predictor variables, c_0 is the *intercept*, and ε is the *error term*. (For more details on the assumptions of the least squares model, see "Assumptions of Least Squares Regression" on page 384.) The predictors can be simple variables or even nonlinear functions of variables.

Suppose that you have a matrix of observed predictor variables X and a vector of response variables Y . (In this sentence, I'm using the terms "matrix" and "vector" in the mathematical sense.) We have assumed a linear model, so given a set of coefficients c , we can calculate a set of estimates \hat{y} for the input data X by calculating $\hat{y} = cX$. The differences between the estimates \hat{y} and the actual values Y are called the *residuals*. You can think of the residuals as a measure of the prediction error; small residuals mean that the predicted values are close to the actual values. We assume that the expected difference between the actual response values and the

to refit a model. This can save you some computation, this can save on computation after changing the formula (perhaps changing the data frame).
 model to the data above. We'll omit the variable (which means to fit the model with

```
runs ~ singles + doubles +
  pitch +
```

```
triples + homeruns +
  caughtstealing - 1,
```

```
triples homeruns
15664 1.31945
bases caughtstealing
8828 -0.70334
```

models work in R, let's describe in a linear regression model is approximately what you want to predict) can be estimated (the information that you

predictor variables (or predictor variables, c_0 is the *intercept*, assumptions of the least squares method" on page 384.) The predictors of variables.

or variables X and a vector of coefficients terms "matrix" and "vector" model, so given a set of coefficient input data X by calculating the actual values Y are called the prediction error; measure of the prediction error; use to the actual values. We use the actual response values and the

residual values (the error term in the model) is 0. This is important to remember: at best, a model is probabilistic.†

Our goal is to find the set of coefficients c that does the best job of estimating Y given X ; we'd like the estimates \hat{y} to be as close as possible to Y . In a classical linear regression model, we find coefficients c that minimize the sum of squared differences between the estimates \hat{y} and the observed values Y . Specifically, we want to find values for c that minimize:

$$RSS(c) = \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

This is called the least squares method for regression. You can use the `lm` function in R to estimate the coefficients in a linear model:‡

```
lm(formula, data, subset, weights, na.action,
  method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
  singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments to `lm` include the following.

Argument	Description	Default
<code>formula</code>	A formula object that specifies the form of the model to fit.	
<code>data</code>	A data frame, list, or environment (or an object that can be coerced to a data frame) in which the variables in <code>formula</code> can be evaluated.	
<code>subset</code>	A vector specifying the observations in <code>data</code> to include in the model.	
<code>weights</code>	A numeric vector containing weights for each observation in <code>data</code> .	NULL
<code>na.action</code>	A function that specifies what <code>lm</code> should do if there are NA values in the data. If NULL, <code>lm</code> uses <code>na.omit</code> .	<code>getOption("na.action")</code> , which defaults to <code>na.fail</code>
<code>method</code>	The method to use for fitting. Only <code>method="qr"</code> fits a model, though you can specify <code>method="model.frame"</code> to return a model frame.	"qr"
<code>model</code>	A logical value specifying whether the "model frame" should be returned.	TRUE
<code>x</code>	Logical values specifying whether the "model matrix" should be returned.	FALSE
<code>y</code>	A logical value specifying whether the response vector should be returned.	FALSE
<code>qr</code>	A logical value specifying whether the QR-decomposition should be returned.	TRUE
<code>singular.ok</code>	A logical value that specifies whether a singular fit results is an error.	TRUE

† By the way, the estimate returned by a model is not an exact prediction. It is, instead, the expected value of the response variable given the predictor variables. To be precise, the estimate \hat{y} means: $\hat{y} = E[y | x_1, x_2, \dots, x_n]$

This observation is important when we talk about generalized linear models later.

‡ To efficiently calculate the coefficients, R uses several matrix calculations. R uses a method called QR-decomposition to transform X into an orthogonal matrix Q and an upper triangular matrix R , where $X = QR$, and then calculates the coefficients as $c = R^{-1}Q^T Y$.

Argument	Description	Default
contrasts	A list of contrasts for factors in the model, specifying one contrast for each factor in the model. For example, for formula $y \sim a + b$, to specify a Helmert contrast for a and a treatment contrast for b , you would use the argument <code>contrasts=(a="contr.helmert", b="contr.treatment")</code> . Some options in R are "contr.helmert" for Helmert contrasts, "contr.sum" for sum-to-zero contrasts, "contr.treatment" to contrast each level with the baseline level, and "contr.poly" for contrasts based on orthogonal polynomials. See [Venables2002] for an explanation of why contrasts are important and how they are used.	When <code>contrasts=NULL</code> (the default), <code>lm</code> uses the value from <code>options("contrasts")</code>
offset	A vector of offsets to use when building the model. (An offset is a linear term that is included in the model without fitting.)	
...	Additional arguments passed to lower-level functions such as <code>lm.fit</code> (for unweighted models) or <code>lm.wfit</code> (for weighted models).	

Model-fitting functions in R return model objects. A model object contains a lot of information about the fitted model (and the fitting operation). Different model objects contain slightly different information.

You may notice that most modeling functions share a few common variables: `mula`, `data`, `na.action`, `subset`, `weights`. These arguments mean the same thing for most modeling functions.

If you are working with a very large data set, you may want to consider using the `biglm` function instead of `lm`. This function uses only p^2 memory for p variables, which is much less than the memory required for `lm`.

Assumptions of Least Squares Regression

Linear models fit with the least squares method are one of the oldest statistical methods, dating back to the age of slide rules. Even today, when computers are ubiquitous, high-quality statistical software is free, and statisticians have developed thousands of new estimation methods, they are still popular. One reason why linear regression is still popular is because linear models are easy to understand. Another reason is that the least squares method has the smallest variance among all unbiased linear estimates (proven by the Gauss-Markov theorem).

Technically, linear regression is not always appropriate. Ordinary least squares (OLS) regression (implemented through `lm`) is only guaranteed to work when certain properties of the training data are true. Here are the key assumptions:

1. **Linearity.** We assume that the response variable y is a linear function of the predictor variables x_1, x_2, \dots, x_n .
2. **Full rank.** There is no linear relationship between any pair of predictor variables. (Equivalently, the predictor matrix is not singular.) Technically, $\forall x_i, x_j, \nexists c$ such that $x_i = cx_j$.
3. **Exogeneity of the predictor variables.** The expected value of the error term ϵ is 0 for all possible values of the predictor variables.

	Default
one contrast for each	When con-
specify a Helmert contrast	trasts=NULL (the
the argument	default), lm uses
tr. treatment").	the value from
ert contrasts,	options("contrasts")
treatment" to contrast	
y" for contrasts based on	
ianation of why contrasts	

4. Homoscedasticity. The variance of the error term ϵ is constant and is not correlated with the predictor variables.
5. Nonautocorrelation. In a sequence of observations, the values of y are not correlated with each other.
6. Exogenously generated data. The predictor variables x_1, x_2, \dots, x_n are generated independently of the process that generates the error term ϵ .
7. The error term ϵ is normally distributed with standard deviation σ and mean 0.

In practice, OLS models often make accurate predictions even when one (or more) of these assumptions are violated.

By the way, it's perfectly OK for there to be a *nonlinear* relationship between some of the predictor variables. Suppose that one of the variables is age. You could add $age^2, \log(age)$, or other nonlinear mathematical expressions using age to the model and not violate the assumptions above. You are effectively defining a set of new predictor variables: $w_1 = age, w_2 = age^2, w_3 = \log(age)$. This doesn't violate the linearity assumption (because the model is still a linear function of the predictor variables) or the full rank assumption (as long as the relationship between the new variables is not linear).

If you want to be careful, you can use test functions to check if the OLS assumptions apply:

- You can test for heteroscedasticity using the function `ncv.test` in the `car` (Companion to Applied Regression) package, which implements the Breusch-Pagan test. (Alternatively, you could use the `bptest` function in the `lmtest` library, which implements the same test. The `lmtest` library includes a number of other functions for testing for heteroscedasticity; see the documentation for more details.)
- You can test for autocorrelation in a model using the function `durbin.watson` in the `car` package, which implements the Durbin-Watson test. You can also use the function `dwtest` in the library `lmtest` by specifying a formula and a data set. (Alternatively, you could use the function `bgtest` in the `lmtest` package, which implements the Breusch-Godfrey test. This functions also tests for higher-order disturbances.)
- You can check that the predictor matrix is not singular by using the `singular.ok=FALSE` argument in `lm`.

Incidentally, the example used in "Example: A Simple Linear Model" on page 373 is not heteroscedastic:

```
> ncv.test(runs.mdl)
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 1.411893   Df = 1   p = 0.2347424
```

Nor is there a problem with autocorrelation:

```
> durbin.watson(runs.mdl)
lag Autocorrelation D-W Statistic p-value
1 0.003318923 1.983938 0.884
Alternative hypothesis: rho != 0
```

Or with singularity:

```
> runs.mdl <- lm(
+ formula=runs~singles+doubles+triples+homeruns+
+ walks+hitbypitch+sacrificeflies+
+ stolenbases+caughtstealing,
+ data=team.batting.00to08,singular.ok=FALSE)
```

If the model has problems with heteroscedasticity or outliers, consider using a resistant or robust regression function, as described in “Robust and Resistant Regression” on page 386. If the data is homoscedastic and not autocorrelated, but the error form is not normal, a good choice is ridge regression, which is described in “Ridge Regression” on page 389. If the predictors are closely correlated (and nearly collinear), a good choice is principal components regression, as described in “Principal Components Regression and Partial Least Squares Regression” on page 391.

Robust and Resistant Regression

Often, ordinary least squares regression works well even with imperfect data. However, it's better in many situations to use regression techniques that are less sensitive to outliers and heteroscedasticity. With R, there are alternative options for fitting linear models.

Resistant regression

If you would like to fit a linear regression model to data with outliers, consider using resistant regression. Using the least median squares (LMS) and least trimmed squares (LTS) estimators:

```
library(MASS)
## S3 method for class 'formula':
lqs(formula, data, ...,
    method = c("lts", "lqs", "lms", "s", "model.frame"),
    subset, na.action, model = TRUE,
    x.ret = FALSE, y.ret = FALSE, contrasts = NULL)

## Default S3 method:
lqs(x, y, intercept = TRUE, method = c("lts", "lqs", "lms", "s"),
    quantile, control = lqs.control(...), ko = 1.548, seed, ...)
```

Robust regression

Robust regression methods can be useful when there are problems with heteroscedasticity and outliers in the data. The function `rlm` in the MASS package fits a model using MM-estimation:

```
## S3 method for class 'formula':
rlm(formula, data, weights, ..., subset, na.action,
```

ation:

p-value
0.884

riples+homeruns+
sacrificeflies+
stealing,
ar.ok=FALSE)

lasticity or outliers, consider using a re-
scribed in "Robust and Resistant Regres-
edastic and not autocorrelated, but the
ridge regression, which is described in
dictors are closely correlated (and nearly
onents regression, as described in "Prin-
east Squares Regression" on page 391.

ks well even with imperfect data. How-
ession techniques that are less sensitive
here are alternative options for fitting

l to data with outliers, consider using
squares (LMS) and least trimmed

'model.frame"),
s = NULL)

;", "lqs", "lms", "s"),
0 = 1.548, seed, ...)

when there are problems with
nction rlm in the MASS package fits

action,

```
method = c("M", "MM", "model.frame"),  
wt.method = c("inv.var", "case"),  
model = TRUE, x.ret = TRUE, y.ret = FALSE, contrasts = NULL)
```

```
## Default S3 method:  
rlm(x, y, weights, ..., w = rep(1, nrow(x)),  
init = "ls", psi = psi.huber,  
scale.est = c("MAD", "Huber", "proposal 2"), k2 = 1.345,  
method = c("M", "MM"), wt.method = c("inv.var", "case"),  
maxit = 20, acc = 1e-4, test.vec = "resid", lqs.control = NULL)
```

You may also want to try the function `lmRob` in the robust package, which fits a model using MS- and S-estimation:

```
library(robust)  
lmRob(formula, data, weights, subset, na.action, model = TRUE, x = FALSE,  
y = FALSE, contrasts = NULL, nrep = NULL,  
control = lmRob.control(...), genetic.control = NULL, ...)
```

Comparing lm, lqs, and rlm

As a quick exercise, we'll look at how `lm`, `lqs`, and `rlm` perform on some particularly ugly data: U.S. housing prices. We'll use Robert Schiller's home price index as an example, looking at home prices between 1890 and 2009.[§] First, we'll load the data and fit the data using an ordinary linear regression model, a robust regression model, and a resistant regression model:

```
> library(nutshell)  
> data(schiller.index)  
> hpi.lm <- lm(Index~Year, data=schiller.index)  
> hpi.rlm <- rlm(Index~Year, data=schiller.index)  
> hpi.lqs <- lqs(Index~Year, data=schiller.index)
```

Now, we'll plot the data to compare how each method worked. We'll plot the models using the `abline` function because it allows you to specify a model as an argument (as long as the model function has a coefficient function):

```
> plot(hpi, pch=19, cex=0.3)  
> abline(reg=hpi.lm, lty=1)  
> abline(reg=hpi.rlm, lty=2)  
> abline(reg=hpi.lqs, lty=3)  
> legend(x=1900, y=200, legend=c("lm", "rlm", "lqs"), lty=c(1,2,3))
```

As you can see from Figure 20-2, the standard linear model is influenced by big peaks (such as the growth between 2001 and 2006) and big valleys (such as the dip between 1920 and 1940). The robust regression method is less sensitive to peaks and valleys in this data, and the resistant regression method is the least sensitive.

Subset Selection and Shrinkage Methods

Modeling functions like `lm` will include every variable specified in the formula, calculating a coefficient for each one. Unfortunately, this means that `lm` may calculate

[§] The data is available from <http://www.irrationalexuberance.com/>.

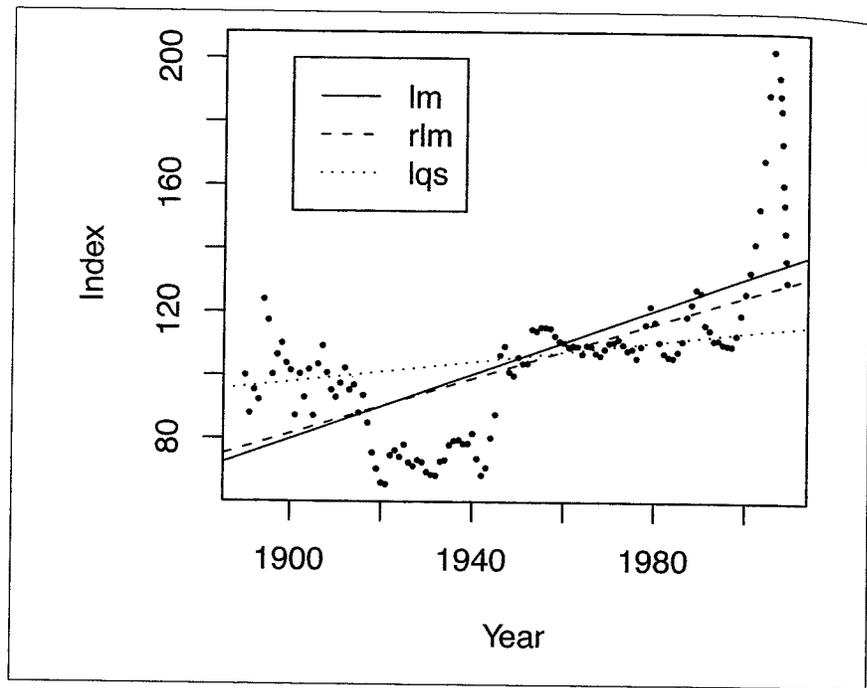


Figure 20-2. Home prices and *lm*, *rlm*, and *lqs* models

coefficients for variables that aren't needed. You can manually tune a model using diagnostics like `summary` and `lm.influence`. However, you can also use some other statistical techniques to reduce the effect of insignificant variables or remove them from a model altogether.

Stepwise Variable Selection

A simple technique for selecting the most important variables is stepwise variable selection. The stepwise algorithm works by repeatedly adding or removing variables from the model, trying to “improve” the model at each step. When the algorithm can no longer improve the model by adding or subtracting variables, it stops and returns the new (and usually smaller) model.

Note that “improvement” does not just mean reducing the residual sum of squares (RSS) for the fitted model. Adding an additional variable to a model will not increase the RSS (see a statistics book for an explanation of why), but it does increase model complexity. Typically, AIC (Akaike's information criterion) is used to measure the value of each additional variable. The AIC is defined as $AIC = -2 * \log(L) + k * edf$, where L is the likelihood and edf is the equivalent degrees of freedom.

In R, you perform stepwise selection through the `step` function:

```
step(object, scope, scale = 0,
      direction = c("both", "backward", "forward"),
      trace = 1, keep = NULL, steps = 1000, k = 2, ...)
```

Here is a description of the arguments to step.

Argument	Description	Default
object	An object representing a model, such as the objects returned by <code>lm</code> , <code>glm</code> , or <code>aov</code> .	
scope	An argument specifying a set of variables that you want in the final model and a list of all variables that you want to consider including in the model. The first set is called the <i>lower bound</i> , and the second is called the <i>upper bound</i> . If a single formula is specified, it is interpreted as the upper bound. To specify both an upper and a lower bound, pass a list with two formulas labeled as upper and lower.	
scale	A value used in the definition of AIC for <code>lm</code> and <code>aov</code> models. See the help file for <code>extractAIC</code> for more information.	0
direction	Specifies whether variables should be only added to the model (<code>direction="forward"</code>), removed from the model (<code>direction="backward"</code>), or both (<code>direction="both"</code>).	<code>c("both", "backward", "forward")</code>
trace	A numeric value that specifies whether to print out details of the fitting process. Specify <code>trace=0</code> (or a negative number) to suppress printing, <code>trace=1</code> for normal detail, and higher numbers for even more detail.	1
keep	A function used to select a subset of arguments to keep from an object. The function accepts a fitted model object and an AIC statistic.	NULL
steps	A numeric value that specifies the maximum number of steps to take before the function halts.	1000
k	The multiple of the number of degrees of freedom to be used in the penalty calculation (<code>extractAIC</code>).	2
...	Additional arguments for <code>extractAIC</code> .	

There is an alternative implementation of stepwise selection in the MASS library: the `stepAIC` function. This function works similarly to `step`, but operates on a wider range of model objects.

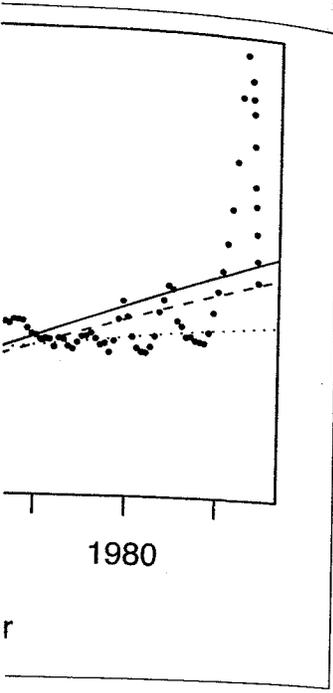
Ridge Regression

Stepwise variable selection simply fits a model using `lm`, but limits the number of variables in the model. In contrast, ridge regression places constraints on the size of the coefficients and fits a model using different computations.

Ridge regression can be used to mitigate problems when there are several highly correlated variables in the underlying data. This condition (called *multicollinearity*) causes high variance in the results. Reducing the number, or impact, of regressors in the data can help reduce these problems.¹¹

In "Details About the `lm` Function" on page 382, we described how ordinary linear regression finds the coefficients that minimize the residual sum of squares. Ridge regression does something similar. Ridge regression attempts to minimize the sum of squared residuals plus a penalty for the coefficient sizes. The penalty is a constant

¹¹ For example, see [Greene2007].



in manually tune a model using `stepAIC`, you can also use some other significant variables or remove them

it variables is stepwise variable selection by adding or removing variables at each step. When the algorithm is selecting variables, it stops and

ing the residual sum of squares. Ridge regression does something similar. Ridge regression attempts to minimize the sum of squared residuals plus a penalty for the coefficient sizes. The penalty is a constant

`stepAIC` function:

```
stepAIC(
  object,
  scope,
  scale,
  direction,
  trace,
  keep,
  steps,
  k,
  ...)
```

λ times the sum of squared coefficients. Specifically, ridge regression tries to minimize the following quantity:

$$RSS_{\text{ridge}}(c) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^m c_j^2$$

To estimate a model using ridge regression, you can use the `lm.ridge` function from the MASS package:

```
library(MASS)
lm.ridge(formula, data, subset, na.action, lambda = 0, model = FALSE,
         x = FALSE, y = FALSE, contrasts = NULL, ...)
```

Arguments to `lm.ridge` are the following.

Argument	Description	Default
<code>formula</code>	A formula object that specifies the form of the model to fit.	
<code>data</code>	A data frame, list, or environment (or an object that can be coerced to a data frame) in which the variables in <code>formula</code> can be evaluated.	
<code>subset</code>	A vector specifying the observations in <code>data</code> to include in the model.	
<code>na.action</code>	A function that specifies what <code>lm</code> should do if there are NA values in the data. If NULL, <code>lm</code> uses <code>na.omit</code> .	
<code>lambda</code>	A scalar or vector of ridge constants.	0
<code>model</code>	A logical value specifying whether the "model frame" should be returned.	FALSE
<code>x</code>	Logical values specifying whether the "model matrix" should be returned.	FALSE
<code>y</code>	A logical value specifying whether the response vector should be returned.	FALSE
<code>contrasts</code>	A list of contrasts for factors in the model.	NULL
...	Additional arguments to <code>lm.fit</code> .	

Lasso and Least Angle Regression

Another technique for reducing the size of the coefficients (and thus reducing their impact on the final model) is the lasso. Like ridge regression, lasso regression puts a penalty on the size of the coefficients. However, the lasso algorithm uses a different penalty: instead of a sum of squared coefficients, the lasso sums the absolute value of the coefficients. (In math terms, ridge uses L^2 -norms, while lasso uses L^1 -norms.) Specifically, the lasso algorithm tries to minimize the following value:

$$RSS_{\text{lasso}}(c) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^m |c_j|$$

The best way to compute lasso regression in R is through the `lars` function:

```
library(lars)
lars(x, y, type = c("lasso", "lar", "forward.stagewise", "stepwise"),
```

ally, ridge regression tries to mini-

can use the `lm.ridge` function from

```
lambda = 0, model = FALSE,
NULL, ...)
```

	Default
it.	
coerced to a data frame) in which	
i the model.	
A values in the data. If NULL, <code>lm</code>	0
uld be returned.	FALSE
uld be returned.	FALSE
uld be returned.	FALSE
	NULL

coefficients (and thus reducing their regression, lasso regression puts a lasso algorithm uses a different the lasso sums the absolute value ms, while lasso uses L^1 -norms.) the following value:

ough the `lars` function:

```
agewise", "stepwise"),
```

```
trace = FALSE, normalize = TRUE, intercept = TRUE, Gram,
eps = .Machine$double.eps, max.steps, use.Gram = TRUE)
```

The `lars` function computes the entire lasso path at once. Specifically, it begins with a model with no variables. It then computes the lambda values for which each variable enters the model and shows the resulting coefficients. Finally, the `lars` algorithm computes a model with all the coefficients present, which is the same as an ordinary linear regression fit.

This function actually implements a more general algorithm called *least angle regression*; you have the option to choose least angle regression, forward stagewise regression, or stepwise regression instead of lasso. Here are the arguments to the `lars` function.

Argument	Description	Default
<code>x</code>	A matrix of predictor variables.	
<code>y</code>	A numeric vector containing the response variable.	
<code>type</code>	The type of model to fit. Use <code>type="lasso"</code> for lasso, <code>type="lar"</code> for least angle regression, <code>type="forward.stage wise"</code> for infinitesimal forward stagewise, and <code>type="stepwise"</code> for stepwise.	<code>c("lasso", "lar", "forward.stagewise", "stepwise")</code>
<code>trace</code>	A logical value specifying whether to print details as the function is running.	FALSE
<code>normalize</code>	A logical value specifying whether each variable will be standardized to have an L^2 -norm of 1.	TRUE
<code>intercept</code>	A logical value indicating whether an intercept should be included in the model.	TRUE
<code>Gram</code>	The $X'X$ matrix used in the calculations. To rerun <code>lars</code> with slightly different parameters, but the same underlying data, you may reuse the Gram matrix from a prior run to increase efficiency.	
<code>eps</code>	An effective 0.	<code>.Machine\$double.eps</code>
<code>max.steps</code>	A limit on the number of steps taken by the <code>lars</code> function.	
<code>use.Gram</code>	A logical value specifying whether <code>lars</code> should precompute the Gram matrix. (For large N , this can be time consuming.)	TRUE

Principal Components Regression and Partial Least Squares Regression

Ordinary least squares regression doesn't always work well with closely correlated variables. A useful technique for modeling effects in this form of data is principal components regression. Principal components regression works by first transforming the predictor variables using principal components analysis. Next, a linear regression is performed on the transformed variables.

A closely related technique is partial least squares regression. In partial least squares regression, both the predictor and the response variables are transformed before fitting a linear regression. In R, principal components regression is available through the function `pcr` in the `pls` package:

```
library(pls)
plr(..., method = pls.options()$pcralg)
```

Partial least squares is available through the function `plsr` in the same package:

```
plsr(..., method = pls.options()$plsralg)
```

Both functions are actually aliases to the function `mvr`:

```
mvr(formula, ncomp, data, subset, na.action,
    method = pls.options()$mvralg,
    scale = FALSE, validation = c("none", "CV", "LOO"),
    model = TRUE, x = FALSE, y = FALSE, ...)
```

Nonlinear Models

The regression models shown above all produced linear models. In this section, we'll look at some algorithms for fitting nonlinear models when you know the general form of the model.

Generalized Linear Models

Generalized linear modeling is a technique developed by John Nelder and Robert Wedderburn to compute many common types of models using a single framework. You can use generalized linear models (GLMs) to fit linear regression models, logistic regression models, Poisson regression models, and other types of models.

As the name implies, GLMs are a generalization of linear models. Like linear models, there is a response variable y and a set of predictor variables x_1, x_2, \dots, x_n . GLMs introduce a new quantity called the *linear predictor*. The linear predictor takes the following form:

$$\eta = c_1 x_1 + c_2 x_2 + \dots + c_n x_n$$

In a general linear model, the predicted value is a function of the linear predictor. The relationship between the response and predictor variables does not have to be linear. However, the relationship between the predictor variables and the linear predictor must be linear. Additionally, the only way that the predictor variables influence the predicted value is through the linear predictor.

In "Example: A Simple Linear Model" on page 373, we noted that a good way to interpret the predicted value of a model is as the expected value (or mean) of the response variable, given a set of predictor variables. This is also true in GLMs, and the relationships between that mean and the linear predictor is what makes GLMs so flexible. To be precise, there must be a smooth, invertible function m such that:

$$\mu = m(\eta), \eta = m^{-1}(\mu) = l(\mu)$$

The inverse of m (denoted by l above) is called the *link function*. You can use many different function families with a GLM, each of which lets you predict a different form of model. For GLMs, the underlying probability distribution needs to be part

of the exponential family of probability distributions. More precisely, distributions that can be modeled by GLMs have the following form:

$$f_y(y; \mu; \varphi) = \exp\left(\frac{A}{\varphi}(y\lambda(\mu) - \eta\lambda(\mu)) + \tau(y, \varphi)\right)$$

As a simple example, if you use the identity function for m and assume a normal distribution for the error term, then $\eta = \mu$ and we just have an ordinary linear regression model. However, you can specify some much more interesting forms of models with GLMs. You can model functions with Gaussian, binomial, Poisson, gamma, and other distributions, and use a variety of link functions, including identity, logit, probit, inverse, log, and other functions.

In R, you can model all of these different types of models using the `glm` function:

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart,
    offset, control = glm.control(...), model = TRUE,
    method = "glm.fit", x = FALSE, y = TRUE, contrasts = NULL,
    ...)
```

Here are the arguments to `glm`.

Argument	Description	Default
<code>formula</code>	A formula object that specifies the form of the model to fit.	
<code>family</code>	Describes the probability distribution of the disturbance term and the link function for the model. (See below for information on different families.)	gaussian
<code>data</code>	A data frame, list, or environment (or an object that can be coerced to a data frame) in which the variables in <code>formula</code> can be evaluated.	
<code>weights</code>	A numeric vector containing weights for each observation in data.	
<code>subset</code>	A vector specifying the observations in data to include in the model.	
<code>na.action</code>	A function that specifies what <code>lm</code> should do if there are NA values in the data. If NULL, <code>lm</code> uses <code>na.omit</code> .	<code>getOption("na.action")</code> , which defaults to <code>na.fail</code>
<code>start</code>	A numeric vector containing starting values for parameters in the linear predictor.	NULL
<code>etastart</code>	A numeric vector containing starting values for the linear predictor.	
<code>mustart</code>	A numeric vector containing starting values for the vector of means.	
<code>offset</code>	A set of terms that are added to the linear term with a constant coefficient of 1. (You can use an offset to force a variable, or a set of variables, into the model.)	
<code>control</code>	A list of parameters for controlling the fitting process. Parameters include <code>epsilon</code> (which specifies the convergence tolerance), <code>maxit</code> (which specifies the maximum number of iterations), and <code>trace</code> (which specifies whether to output information on each iteration). See <code>glm.control</code> for more information.	<code>glm.control(...)</code> , which, in turn, has defaults <code>epsilon=1e-8</code> , <code>maxit=25</code> , <code>trace=FALSE</code>
<code>model</code>	A logical value specifying whether the "model frame" should be returned.	TRUE

Argument	Description	Default
method	The method to use for fitting. Only <code>method="glm.fit"</code> fits a model, though you can specify <code>method="model.frame"</code> to return a model frame.	"glm.fit"
x	Logical values specifying whether the "model matrix" should be returned.	FALSE
y	A logical value specifying whether the "response vector" should be returned.	TRUE
contrasts	A list of contrasts for factors in the model.	NULL
...	Additional arguments passed to <code>glm.control</code> .	

GLM fits a model using iteratively reweighted least squares (IWLS).

As noted above, you can model many different types of functions using GLM. The following function families are available in R:

```
binomial(link = "logit")
gaussian(link = "identity")
Gamma(link = "inverse")
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")
```

You may specify an alternative link function for most of these function families. Here is a list of the possible link functions for each family.

Family function	Allowed link functions	Default link function
binomial	"logit", "probit", "cauchit", "log", and "cloglog"	"logit"
gaussian	"identity", "log", and "inverse"	"identity"
Gamma	"inverse", "identity", and "log"	"inverse"
inverse.gaussian	"1/mu^2", "inverse", "identity", and "log"	"1/mu^2"
poisson	"log", "identity", and "sqrt"	"log"
quasi	"logit", "probit", "cloglog", "identity", "inverse", "log", "1/mu^2", and "sqrt", or use the power function to create a power link function	"identity"
quasibinomial		"logit"
quasipoisson		"log"

The `quasi` function also takes a variance argument (with default constant); see the help file for `quasi` for more information.

If you are working with a large data set and have limited memory, you may want to consider using the `biglm` function in the `biglm` package.

As an example, let's use the `glm` function to fit the same model that we used for `lm`. By default, `glm` assumes a Gaussian error distribution, so we expect the fitted model to be identical to the one fitted above:

```
> runs.glm <- glm(
+   formula=runs~singles+doubles+triples+homeruns+
```

Default

lm.fit" fits a model, though o return a model frame.	"glm.fit"
matrix" should be returned.	FALSE
vector" should be returned.	TRUE
	NULL

least squares (IWLS).
t types of functions using GLM. The

tant")
most of these function families. Here
amily.

Default link function

	"logit"
	"identity"
	"inverse"
	"1/mu^2"
	"log"
"log", "1/mu^2", and er link function	"identity"
	"logit"
	"log"

nt (with default constant); see the
limited memory, you may want to
ackage.
e same model that we used for lm.
ion, so we expect the fitted model
meruns+

```
+ walks+hitbypitch+sacrificeflies+
+ stolenbases+caughtstealing,
+ data=team.batting.00to08)
> runs.glm
```

```
Call: glm(formula = runs ~ singles + doubles + triples + homeruns +
walks + hitbypitch + sacrificeflies + stolenbases + caughtstealing,
data = team.batting.00to08)
```

Coefficients:

(Intercept)	singles	doubles	triples
-507.16020	0.56705	0.69110	1.15836
homeruns	walks	hitbypitch	sacrificeflies
1.47439	0.30118	0.37750	0.87218
stolenbases	caughtstealing		
0.04369	-0.01533		

Degrees of Freedom: 269 Total (i.e. Null); 260 Residual
Null Deviance: 1637000
Residual Deviance: 140100 AIC: 2476

As expected, the fitted model is identical to the model from lm. (Typically, it's better to use lm rather than glm when fitting an ordinary linear regression model because lm is more efficient.) Notice that glm provides slightly different information through the print statement, such as the degrees of freedom, null deviance, residual deviance, and AIC. We'll revisit glm when talking about logistic regression models for classification; see "Logistic Regression" on page 435.

Nonlinear Least Squares

Sometimes, you know the form of a model, even if the model is extremely nonlinear. To fit nonlinear models (minimizing least squares error), you can use the nls function:

```
nls(formula, data, start, control, algorithm,
trace, subset, weights, na.action, model,
lower, upper, ...)
```

Here is a description of the arguments to the nls function.

Argument	Description
formula	A formula object that specifies the form of the model to fit.
data	A data frame in which formula can be evaluated.
start	A named list or named vector with starting estimates for the fit.
control	A list of arguments to pass to control the fitting process (see the help file for nls.control for more information).
algorithm	The algorithm to use for fitting the model. Use algorithm="plinear" for the Golub-Pereyra algorithm for partially linear least squares models and algorithm="port" for the 'nl2sol' algorithm from the Port library.
trace	A logical value specifying whether to print the progress of the algorithm while nls is running.

Argument	Description
subset	An optional vector specifying the set of rows to include.
weights	An optional vector specifying weights for observations.
na.action	A function that specifies how to treat NA values in the data.
model	A logical value specifying whether to include the model frame as part of the model object.
lower	An optional vector specifying lower bounds for the parameters of the model.
upper	An optional vector specifying upper bounds for the parameters of the model.
...	Additional arguments (not currently used).

The `nls` function is actually a wrapper for the `nlm` function. The `nlm` function is similar to `nls`, but takes an R function (not a formula) and list of starting parameters as arguments. It's usually easier to use `nls` because `nls` allows you to specify models using formulas and data frames, like other R modeling functions. For more information about `nlm`, see the help file.

By the way, you can actually use `nlm` to fit a linear model. It will work, but it will be slow and inefficient.

Survival Models

Survival analysis is concerned with looking at the amount of time that elapses before an event occurs. An obvious application is to look at mortality statistics (predicting how long people live), but it can also be applied to mechanical systems (the time before a failure occurs), marketing (the amount of time before a consumer cancels an account), or other areas.

In R, there are a variety of functions in the survival library for modeling survival data.

To estimate a survival curve for censored data, you can use the `survfit` function:

```
survfit(formula, data, weights, subset, na.action,
        etype, id, ...)
```

This function accepts the following arguments.

Argument	Description
formula	Describes the relationship between the response value and the predictors. The response value should be a <code>Surv</code> object.
data	The data frame in which to evaluate formula.
weights	Weights for observations.
subset	Subset of observation to use in fitting the model.
na.action	Function to deal with missing values.
etype	The variable giving the type of event.
id	The variable that identifies individual subjects.

Argument	Description
<code>type</code>	Specifies the type of survival curve. Options include "kaplan-meier", "fleming-harrington", and "fh2".
<code>error</code>	Specifies the type of error. Possible values are "greenwood" for the Greenwood formula or "tsiatis" for the Tsiatis formula.
<code>conf.type</code>	Confidence interval type. One of "none", "plain", "log" (the default), or "log-log".
<code>conf.lower</code>	A character string to specify modified lower limits to the curve, the upper limit remains unchanged. Possible values are "usual" (unmodified), "peto", and "modified".
<code>start.time</code>	Numeric value specifying a time to start calculating survival information.
<code>conf.int</code>	The level for a two-sided confidence interval on the survival curve(s).
<code>se.fit</code>	A logical value indicating whether standard errors should be computed.
...	Additional variables passed to internal functions.

As an example, let's fit a survival curve for the GSE2034 data set. This data comes from the Gene Expression Omnibus of the National Center for Biotechnology Information (NCBI), which is accessible from <http://www.ncbi.nlm.nih.gov/geo/>. The experiment examined how the expression of certain genes affected breast cancer relapse-free survival time. In particular, it tested estrogen receptor binding sites. (We'll revisit this example in Chapter 24.)

First, we need to create a `Surv` object within the data frame. A `Surv` object is an R object for representing survival information, in particular, censored data. Censored data occurs when the outcome of the experiment is not known for all observations. In this case, the data is censored. There are three possible outcomes for each observation: the subject had a recurrence of the disease, the subject died without having a recurrence of the disease, or the subject was still alive without a recurrence at the time the data was reported. The last outcome—the subject was still alive without a recurrence—results in the censored values:

```
> library(survival)
> GSE2034.Surv <- transform(GSE2034,
+   surv=Surv(
+     time=GSE2034$months.to.relapse.or.last.followup,
+     event=GSE2034$relapse,
+     type="right"
+   )
+ )
# show the first 26 observations:
> GSE2034.Surv$surv[1:26,]
[1] 101+ 118+ 9 106+ 37 125+ 109+ 14 99+ 137+ 34 32 128+
[14] 14 130+ 30 155+ 25 30 84+ 7 100+ 30 7 133+ 43
```

Now, let's calculate the survival model. We'll just make it a function of the `ER.status` flag (which stands for "estrogen receptor"):

```
> GSE2034.survfit <- survfit(
+   formula=surv~ER.Status,
```

```
+ data=GSE2034.Surv,
+ )
```

The easiest way to view a `survfit` object is graphically. Let's plot the model:

```
> plot(GSE2034.survfit,lty=1:2,log=T)
> legend(135,1,c("ER+", "ER-"),lty=1:2,cex=0.5)
```

The plot is shown in Figure 20-3. Note the different curve shape for each cohort.

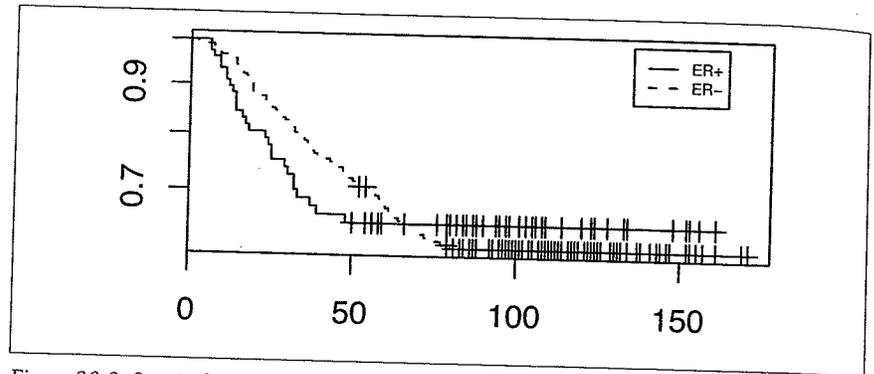


Figure 20-3. Survival curves for the GSE2034 data

To fit a parametric survival model, you can use the `survreg` function in the `survival` package:

```
survreg(formula, data, weights, subset,
         na.action, dist="weibull", init=NULL, scale=0,
         control, parms=NULL, model=FALSE, x=FALSE,
         y=TRUE, robust=FALSE, score=FALSE, ...)
```

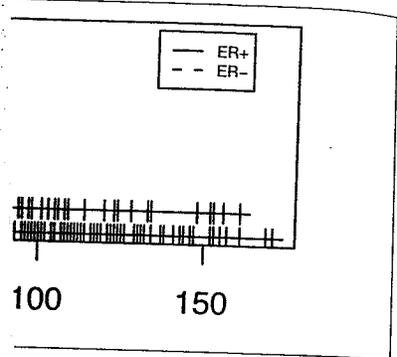
Here is a description of the arguments to `survreg`.

Argument	Description	Default
<code>formula</code>	A formula that describes the form of the model; the response is usually a <code>Surv</code> object (created by the <code>Surv</code> function).	
<code>data</code>	A data frame containing the training data for the model.	
<code>weights</code>	A vector of weights for observations in <code>data</code> .	
<code>subset</code>	An expression describing a subset of observations in <code>data</code> to use for fitting the model.	
<code>na.action</code>	A function that describes how to treat NA values.	<code>options()\$na.action</code>
<code>dist</code>	A character value describing the form of the y variable (either "weibull", "exponential", "gaussian", "logistic", "lognormal", or "loglogistic") or a distribution like the ones in <code>survreg.distributions</code> .	"weibull"
<code>init</code>	Optional vector of initial parameters.	NULL
<code>scale</code>	Value specifying the scale of the estimates. Estimated if <code>scale <= 0</code> .	0
<code>control</code>	A list of control values, usually produced by <code>survreg.control</code> .	
<code>parms</code>	A list of fixed parameters for the distribution function.	NULL

graphically. Let's plot the model:

ex=0.5)

ifferent curve shape for each cohort.



use the survreg function in the

LL, scale=0,
FALSE,
...)

'g.

onse is usually a Surv

l.

ta to use for fitting the

ither "weibull",
gnormal", or "loglo
distributions.

cale <= 0.

ontrol.

Argument	Default
options()\$na.action	"weibull"
NULL	NULL
scale <= 0.	0
control.	NULL

Argument	Description	Default
model, x, y	Logical values indicating whether to return the model frame, X matrix, or Y vector (respectively) with the results.	FALSE
robust	A logical value indicating whether to use "robust sandwich standard methods."	FALSE
score	A logical value indicating whether to return the score vector.	FALSE
...	Other arguments passed to survreg.control.	

You can compute the expected survival for a set of subjects (or individual expectations for each subject) with the function survexp:

```
library(survival)
survexp(formula, data, weights, subset, na.action, times, cohort=TRUE,
         conditional=FALSE, ratetable=survexp.us, scale=1, npoints,
         se.fit, model=FALSE, x=FALSE, y=FALSE)
```

Here is a description of the arguments to survexp.

Argument	Description	Default
formula	A formula object describing the form of the model. The (optional) response should contain a vector of follow-up times, and the predictors should contain grouping variables separated by + operators.	
data	A data frame containing source data on which to predict values.	
weights	A vector of weights for the cases.	
subset	An expression indicating which observations in data should be included in the prediction.	
na.action	A function specifying how to deal with missing (NA) values in the data.	options()\$na.action
times	A vector of follow-up times at which the resulting survival curve is evaluated. (This may also be included in the formula; see above.)	
cohort	A logical value indicating whether to calculate the survival of the whole cohort (cohort=TRUE) or individual observations (cohort=FALSE).	TRUE
conditional	A logical value indicating whether to calculate conditional expected survival. Specify conditional=TRUE if the follow-up times are times of death, and conditional=FALSE if the follow-up times are potential censoring times.	FALSE
ratetable	A fitted Cox model (from coxph) or a table of survival times.	survexp.us
scale	A numeric value specifying how to scale the results.	1
npoints	A numeric value indicating the number of points at which to calculate individual results.	
se.fit	A logical value indicating whether to include the standard error of the predicted survival.	
model, x, y	Specifies whether to return the model frame, the X matrix, or the Y vector (respectively) in the results.	FALSE for all three

The Cox proportional hazard model is a nonparametric method for fitting survival models. It is available in R through the coxph function in the survival library:

```
coxph(formula, data=, weights, subset,
       na.action, init, control,
       method=c("efron", "breslow", "exact"),
```

```
singular.ok=TRUE, robust=FALSE,
model=FALSE, x=FALSE, y=TRUE, ...)
```

Here is a description of the arguments to `coxph`.

Argument	Description	Default
<code>formula</code>	A formula that describes the form of the model; the response must be a <code>Surv</code> object (created by the <code>Surv</code> function).	
<code>data</code>	A data frame containing source data on which to predict values.	
<code>weights</code>	A vector of weights for the cases.	
<code>subset</code>	An expression indicating which observations in data should be fit.	
<code>na.action</code>	A function specifying how to deal with missing (NA) values in the data.	
<code>init</code>	A vector of initial parameter values for the fitting process.	0 for all variables
<code>control</code>	Object of class <code>coxph.control</code> specifying the iteration limit and other control options.	<code>coxph.control(...)</code>
<code>method</code>	A character value specifying the method for handling ties. Choices include "efron", "breslow", and "exact".	"efron"
<code>singular.ok</code>	A logical value indicating whether to stop with an error if the X matrix is singular or to simply skip variables that are linear combinations of other variables.	TRUE
<code>robust</code>	A logical value indicating whether to return a robust variance estimate.	FALSE
<code>model</code>	A logical value specifying whether to return the model frame.	FALSE
<code>x</code>	A logical value specifying whether to return the X matrix.	FALSE
<code>y</code>	A logical value specifying whether to return the Y vector.	TRUE
...	Additional arguments passed to <code>coxph.control</code> .	

As an example, let's fit a Cox proportional hazard model to the GSE2034 data:

```
> GSE2034.coxph <- coxph(
+   formula=surv~ER.Status,
+   data=GSE2034.Surv,
+ )
> GSE2034.coxph
Call:
coxph(formula = surv ~ ER.Status, data = GSE2034.Surv)
```

```
              coef exp(coef) se(coef)      z    p
ER.StatusER+ -0.00378    0.996    0.223 -0.0170 0.99
```

```
Likelihood ratio test=0 on 1 df, p=0.986 n= 286
```

The summary method for `coxph` objects provides additional information about the fit:

```
> summary(GSE2034.coxph)
Call:
coxph(formula = surv ~ ER.Status, data = GSE2034.Surv)

n= 286
```

```

              coef exp(coef) se(coef)      z Pr(>|z|)
ER.StatusER+ -0.00378  0.99623  0.22260 -0.017  0.986

```

```

              exp(coef) exp(-coef) lower .95 upper .95
ER.StatusER+  0.9962    1.004    0.644    1.541

```

```

Rsquare= 0 (max possible= 0.983 )
Likelihood ratio test= 0 on 1 df,  p=0.9865
Wald test              = 0 on 1 df,  p=0.9865
Score (logrank) test = 0 on 1 df,  p=0.9865

```

Another useful function is `cox.zph`, which tests the proportional hazards assumption for a Cox regression model fit:

```

> cox.zph(GSE2034.coxph)
              rho chisq      p
ER.StatusER+ 0.33  11.6 0.000655

```

There are additional methods available for viewing information about `coxph` fits, including `residuals`, `predict`, and `survfit`; see the help file for `coxph` object for more information.

There are other functions in the survival package for fitting survival models, such as `cch` which fits proportional hazard models to case-cohort data. See the help files for more information.

Smoothing

This section describes a number of functions for fitting piecewise smooth curves to data. Functions in this section are particularly useful for plotting charts; there are even convenience functions for using these functions to show fitted values in some graphics packages.

Splines

One method for fitting a function to source data is with splines. With a linear model, a single line is fitted to all the data. With spline methods, a set of different polynomials is fitted to different sections of the data.

You can compute simple cubic splines with the `spline` function in the `stats` package:

```

spline(x, y = NULL, n = 3*length(x), method = "fmm",
       xmin = min(x), xmax = max(x), xout, ties = mean)

```

Here is a description of the arguments to `smooth.spline`.

Argument	Description	Default
<code>x</code>	A vector specifying the predictor variable, or a two-column matrix specifying both the predictor and the response variables.	
<code>y</code>	If <code>x</code> is a vector, then <code>y</code> is a vector containing the response variable.	NULL
<code>n</code>	If <code>xout</code> is not specified, interpolation is done at <code>n</code> equally spaced points between <code>xmin</code> and <code>xmax</code> .	<code>3*length(x)</code>

Argument	Description	Default
method	Specifies the type of spline. Allowed values include "fmm", "natural", "periodic", and "monoH.FC".	"fmm"
xmin	Lowest x value for interpolations.	
xmax	Highest x value for interpolations.	min(x)
xout	An optional vector of values specifying where interpolation should be done.	max(x)
ties	A method for handling ties. Either the string "ordered" or a function that returns a single numeric value.	mean

To return a function instead of a list of parameters, use the function `splinefun`:

```
splinefun(x, y = NULL, method = c("fmm", "periodic", "natural", "monoH.FC"),
          ties = mean)
```

To fit a cubic smoothing spline model to supplied data, use the `smooth.spline` function:

```
smooth.spline(x, y = NULL, w = NULL, df, spar = NULL,
              cv = FALSE, all.knots = FALSE, nknots = NULL,
              keep.data = TRUE, df.offset = 0, penalty = 1,
              control.spar = list())
```

Here is a description of the arguments to `smooth.spline`.

Argument	Description	Default
x	A vector specifying the predictor variable, or a two-column matrix specifying both the predictor and the response variables.	
y	If x is a vector, then y is a vector containing the response variable.	
w	An (optional) numeric vector containing weights for the input data.	NULL
df	Degrees of freedom.	NULL
spar	Numeric value specifying the smoothing parameter.	
cv	A logical value specifying whether to use ordinary cross-validation (cv=TRUE) or generalized cross-validation (cv=FALSE).	FALSE
all.knots	A logical value specifying whether to use all values in x as knots.	FALSE
nknots	An integer value specifying the number of knots to use when all.knots=FALSE.	NULL
keep.data	A logical value indicating whether the input data should be kept in the result.	TRUE
df.offset	A numeric value specifying how much to allow the df to be increased in cross-validation.	0
penalty	The penalty for degrees of freedom during cross-validation.	1
control.spar	A list of parameters describing how to compute spar (when not explicitly specified). See the help file for more information.	list()

For example, we can calculate a smoothing spline on the Schiller home price index. This data set contains one annual measurement through 2006, but then has fractional measurements after 2006, making it slightly difficult to align with other data:

	Default
"fmm", "natural", "peri	"fmm"
	min(x)
	max(x)
relation should be done.	
red" or a function that returns a	mean

eters, use the function `splinefun`:
`"periodic", "natural", "monoH.FC")`,

plied data, use the `smooth.spline`

```
spars = NULL,
SE, nknots = NULL,
= 0, penalty = 1,
```

h.spline.

	Default
matrix specifying both the predictor	
onse variable.	NULL
the input data.	NULL
	NULL
-validation (cv=TRUE) or generalized	FALSE
x as knots.	FALSE
e when all.knots=FALSE.	NULL
ld be kept in the result.	TRUE
to be increased in cross-validation.	0
ation.	1
(when not explicitly specified). See	list()

e on the Schiller home price index.
through 2006, but then has frac-
y difficult to align with other data:

```
> schiller.index[schiller.index$Year>2006,]
      Year Real.Home.Price.Index
118 2007.125          194.6713
119 2007.375          188.9270
120 2007.625          184.1683
121 2007.875          173.8622
122 2008.125          160.7639
123 2008.375          154.4993
124 2008.625          145.6642
125 2008.875          137.0083
126 2009.125          130.0611
```

We can use smoothing splines to find values for 2007 and 2008:

```
> library(nuts�hell)
> data(schiller.index)
> schiller.index.spl <- smooth.spline(schiller.index$Year,
+ schiller.index$Real.Home.Price.Index)
> predict(schiller.index.spl,x=c(2007,2008))
$x
[1] 2007 2008

$y
[1] 195.6682 168.8219
```

Fitting Polynomial Surfaces

You can fit a polynomial surface to data (by local fitting) using the `loess` function. (This function is used in many graphics functions; for example, `panel.loess` uses `loess` to fit a curve to data and plot the curve.)

```
loess(formula, data, weights, subset, na.action, model = FALSE,
span = 0.75, enp.target, degree = 2,
parametric = FALSE, drop.square = FALSE, normalize = TRUE,
family = c("gaussian", "symmetric"),
method = c("loess", "model.frame"),
control = loess.control(...), ...)
```

Here is a description of the arguments to `loess`.

Argument	Description	Default
formula	A formula specifying the relationship between the response and the predictor variables.	
data	A data frame, list, or environment specifying the training data for the model fit. (If none is specified, formula is evaluated in the calling environment.)	
weights	A vector of weights for the cases in the training data.	
subset	An optional expression specifying a subset of cases to include in the model.	
na.action	A function specifying how to treat missing values.	getOption("na.action")
model	A logical value indicating whether to return the model frame.	FALSE
span	A numeric value specifying the parameter α , which controls the degree of smoothing.	0.75

Argument	Description	Default
enp.target	A numeric value specifying the equivalent number of parameters to be used (replaced span).	
degree	The degree of polynomials used.	2
parametric	A vector specifying any terms that should be fit globally rather than locally. (May be specified by name, number, or as a logical vector.)	FALSE
drop.square	Specifies whether to drop the quadratic term for some predictors.	FALSE
normalize	A logical value specifying whether to normalize predictors to a common scale.	TRUE
family	Specifies how fitting is done. Specify family="gaussian" to fit by least squares, and family="symmetric" to fit with Tukey's biweight function.	"gaussian"
method	Specifies whether to fit the model or just return the model frame.	"loess"
control	Control parameters for loess, typically generated by a call to loess.control.	loess.control(...)
...	Additional arguments are passed to loess.control.	

Using the same example as above:

```
> schiller.index.loess <- loess(Real.Home.Price.Index~Year, data=schiller.index)
> predict(schiller.index.loess, newdata=data.frame(Year=c(2007,2008)))
[1] 156.5490 158.8857
```

Kernel Smoothing

To estimate a probability density function, regression function, or their derivatives using polynomials, try the function `locpoly` in the library `KernSmooth`:

```
library(KernSmooth)
locpoly(x, y, drv = 0L, degree, kernel = "normal",
        bandwidth, gridsize = 401L, bwdisc = 25,
        range.x, binned = FALSE, truncate = TRUE)
```

Here is a description of the arguments to `locpoly`.

Argument	Description	Default
x	A vector of x values (with no missing values).	
y	A vector of y values (with no missing values).	
drv	Order of derivative to estimate.	0L
degree	Degree of local polynomials.	drv + 1
kernel	Kernel function to use. Currently ignored ("normal" is used).	"normal"
bandwidth	A single value or an array of length <code>gridsize</code> that specifies the kernel bandwidth smoothing parameter.	
gridsize	Specifies the number of equally spaced points over which the function is estimated.	401L
bwdisc	Number of (logarithmically equally spaced) values on which <code>bandwidth</code> is discretized.	25
range.x	A vector containing the minimum and maximum values of x on which to compute the estimate.	

	Default
nt number of parameters to be used	2
d be fit globally rather than locally. (s is a logical vector.)	FALSE
ern for some predictors.	FALSE
normalize predictors to a common scale.	TRUE
amily="gaussian" to fit by least squares with Tukey's biweight function.	"gaussian"
return the model frame.	"loess"
generated by a call to	loess.control(...)
loess.control.	

```
l.Home.Price.Index~Year,data=schiller.index)
data=data.frame(Year=c(2007,2008))
```

on, regression function, or their derivatives
only in the library KernSmooth:

```
el = "normal",
bwdisc = 25,
ncate = TRUE)
```

locpoly.

	Default
	0L
	drv + 1
family" is used).	"normal"
what specifies the kernel bandwidth smoothing	
over which the function is estimated.	401L
values on which bandwidth is discretized.	25
number of values of x on which to compute the	

Argument	Description	Default
binned	A logical value specifying whether to interpret x and y as grid counts (as opposed to raw data).	FALSE
truncate	A logical value specifying whether to ignore x values outside range . x.	TRUE

R also includes an implementation of local regression through the locfit function in the locfit library:

```
library(locfit)
locfit(formula, data=sys.frame(sys.parent()), weights=1, cens=0, base=0,
subset, geth=FALSE, ..., lfproc=locfit.raw)
```

Machine Learning Algorithms for Regression

Most of the models above assumed that you knew the basic form of the model equation and error function. In each of these cases, our goal was to find the coefficients of variables in a known function. However, sometimes you are presented with data where there are many predictive variables, and the relationships between the predictors and response are very complicated.

Statisticians have developed a variety of different techniques to help model more complex relationships in data sets and to predict values for large, complicated data sets. This section describes a variety of techniques for finding not only the coefficients of a model function but also the function itself.

In this section, I use the San Francisco home sales data set described in "More About the San Francisco Real Estate Prices Data Set" on page 290. This is a pretty ugly data set, with lots of nonlinear relationships. Real estate is all about location, and we have several different variables in the data set that represent location. (The relationships between these variables is not linear, in case you were worried.)

Before modeling, we'll split the data set into training and testing data sets. Splitting data into training and testing data sets (and, often, validation data sets as well) is a standard practice when fitting models. Statistical models have a tendency to "overfit" the training data; they do a better job predicting trends in the training data than in other data.

I chose this approach because it works with all of the modeling functions in this section. There are other statistical techniques available for making sure that a model doesn't overfit the data, including cross-validation and bootstrapping. Functions for cross-validation are available for some models (for example, `xpred.rpart` for `rpart` trees); look at the detailed help files for a package (in this case, with the command `help(package="rpart")`) to see if these functions are available for a specific modeling tool. Bootstrap resampling is available through the `boot` library.

Because this section presents many different types of models, I decided to use a simple, standard approach for evaluating model fits. For each model, I estimated the root mean square (RMS) error for the training and validation data sets. Don't interpret the results as authoritative: I didn't try too hard to tune each model's parameters and know that the models that worked best for this data set do not work

best for all data sets. However, I thought I'd include the results because I was interested in them (in good fun) and thought readers would be as well.

Anyway, I wrote the following function to evaluate the performance of each function:

```
calculate_rms_error <- function mdl, train, test, yval {  
  train.yhat <- predict(object=mdl, newdata=train)  
  test.yhat <- predict(object=mdl, newdata=test)  
  train.y <- with(train, get(yval))  
  test.y <- with(test, get(yval))  
  train.err <- sqrt(mean((train.yhat - train.y)^2))  
  test.err <- sqrt(mean((test.yhat - test.y)^2))  
  c(train.err=train.err, test.err=test.err)  
}
```

To create a random sample, I used the `sample` function to pick 70% of values for the training data. I saved the sample indices to a vector for later reuse (so that I could derive the same sample later, and allow you to use the same sample as well). I also saved the sample indices to make it easy to define the testing data set.

```
> nrow(sanfrancisco.home.sales) * .7  
[1] 2296.7  
> sanfrancisco.home.sales.training.indices <-  
+ sample(1:nrow(sanfrancisco.home.sales), 2296)  
> sanfrancisco.home.sales.testing.indices <-  
+ setdiff(rownames(sanfrancisco.home.sales),  
+         sanfrancisco.home.sales.training.indices)  
> sanfrancisco.home.sales.training <-  
+ sanfrancisco.home.sales[sanfrancisco.home.sales.training.indices,]  
> sanfrancisco.home.sales.testing <-  
+ sanfrancisco.home.sales[sanfrancisco.home.sales.testing.indices,]  
> save(sanfrancisco.home.sales.training.indices,  
+      sanfrancisco.home.sales.testing.indices,  
+      sanfrancisco.home.sales,  
+      file="~/Documents/book/current/data/sanfrancisco.home.sales.RData")
```

Note that the sampling is random, so you will get a different subset each time you run this code. The vectors `sanfrancisco.home.sales.training.indices` and `sanfrancisco.home.sales.testing.indices` that I used in this section are included in the `nutshell` package. (Use the command `data(sanfrancisco.home.sales)` to access them. The data sets `sanfrancisco.home.sales.training` and `sanfrancisco.home.sales.testing` are not included.) You can use the same training and testing sets to re-create the results in this section, or you can pick your own subsets.

Regression Tree Models

Most of the models that we have seen in this chapter are in the form of a single equation. You can use the model to predict values by plugging new data values into a single equation.

Tree models have a slightly different form. Instead of a single, compact equation, tree models represent data by a set of binary decision rules. Instead of plugging

numbers into an equation, you follow the rules in a tree to determine the predicted value. Tree models are very easy to interpret, but don't usually predict values as accurately as other types of models. Tree models are particularly popular in medicine and biology, perhaps because they resemble the process that doctors use to make decisions. In this section, we'll show how to use some popular tree methods for regression in R.

Recursive partitioning trees

One of the most popular algorithms for building tree models is classification and regression trees, or CART. CART uses a greedy algorithm to build a tree from the training data. Here's an explanation of how CART works:

1. Grow the tree using the following (recursive) method:
 - A. Start with a single set containing all the training data.
 - B. If the number of observations is less than the minimum required for a split, stop splitting the tree. Output the average of all the y -values in the training data as the predicted value for the terminal node.
 - C. Find a variable x_j and value s that minimizes the RMS error when you split the data into two sets.
 - D. Repeat the splitting process (starting at step B) on each of the two sets.
2. Prune the tree using the following (iterative) method:
 - A. Stop if there is only one node in the tree.
 - B. Measure the cost/complexity of the overall tree. (The cost/complexity measurement is a measurement that takes into account the number of observations in each node, the RMS prediction error, and the number of nodes in the tree.)
 - C. Try collapsing each internal node on the tree and measure which subtree has the best cost/complexity.
 - D. Repeat the process (starting at step A) on the subtree with the best cost/complexity.
3. Output the tree with the lowest cost/complexity.

R includes an implementation of classification and regression trees in the `rpart` package. To fit a model, use the `rpart` function:

```
library(rpart)
rpart(formula, data, weights, subset, na.action = na.rpart, method,
      model = FALSE, x = FALSE, y = TRUE, parms, control, cost, ...)
```

Here are the arguments to `rpart`.

Argument	Description	Default
<code>formula</code>	A formula describing the relationship between the response and the predictor variables.	
<code>data</code>	A data frame to use for fitting the model.	
<code>weights</code>	An optional vector of weights to use for the training data.	

Argument	Description	Default
subset	An optional expression specifying which observations to use in fitting the model.	
na.action	The function to call for missing values.	na.rpart
method	A character value that specifies the fitting method. Must be one of "exp", "poisson", "class", or "anova".	If y is a survival object, then method="exp", if y has two columns then method="poisson", or y if a factor then method="class", otherwise method="anova"
model	A logical value specifying whether to keep the model frame in the results.	FALSE
x	A logical value specifying whether to return the x matrix in the results.	FALSE
y	A logical value specifying whether to return the y matrix in the results.	TRUE
parms	A list of parameters passed to the fitting function.	
control	Options that control details of the rpart algorithm; see rpart.control for more information.	
cost	A numeric vector of costs, one for each variable in the model.	1 for all variables
...	Additional argument passed to rpart.control.	

The CART algorithm handles missing values differently from many other modeling algorithms. With an algorithm like linear regression, missing values need to be filtered out in order for the math to work. However, CART takes advantage of the rule-based model structure to handle missing values differently. When a value is missing for an observation at a split, CART can instead split values using a *surrogate* variable. See the help files for `rpart` for more information on how to control the process of finding and using surrogates.

As an example, let's build a regression tree on the San Francisco home sales data set. We'll start off naively, adding some redundant information and fields that could lead to a model that overfits the data:

```
> library(rpart)
> sf.price.model.rpart <- rpart(
+   price~bedrooms+squarefeet+lotsize+latitude+
+   longitude+neighborhood+month,
+   data=sanfrancisco.home.sales.training)
```

Let's take a look at the model returned by this call to `rpart`. The simplest way to examine the object is to use `print.rpart` to print it on the console. The output below has been modified slightly to fit in this book:

```
> sf.price.model.rpart
n= 2296

node), split, n, deviance, yval
* denotes terminal node

1) root 2296 8.058726e+14 902088.0
```

Default

observations to use in fitting the	na.rpart
method. Must be one of "exp",	If y is a survival object, then method="exp", if y has two columns then method="poisson", or y if a factor then method="class", otherwise method="anova"
the model frame in the results.	FALSE
the x matrix in the results.	FALSE
the y matrix in the results.	TRUE
function.	
algorithm; see rpart.con	
variable in the model.	1 for all variables
control.	

values differently from many other modeling or regression, missing values need to be filled. However, CART takes advantage of the missing values differently. When a value is missing, CART can instead split values using a surrogate for more information on how to control the

tree on the San Francisco home sales data set. Redundant information and fields that could

longitude+latitude+

training)

by this call to rpart. The simplest way to print it on the console. The output below looks like:

- 2) neighborhood=Bayview,Bernal Heights,Chinatown,Crocker Amazon, Diamond Heights,Downtown,Excelsior,Inner Sunset,Lakeshore, Mission,Nob Hill,Ocean View,Outer Mission,Outer Richmond, Outer Sunset,Parkside,Potrero Hill,South Of Market, Visitacion Valley,Western Addition 1524 1.850806e+14 723301.8
- 4) squarefeet< 1772 1282 1.124418e+14 675471.1
- 8) neighborhood=Bayview,Chinatown,Crocker Amazon, Diamond Heights,Downtown,Excelsior,Lakeshore,Ocean View, Outer Mission,Visitacion Valley 444 1.408221e+13 539813.1 *
- 9) neighborhood=Bernal Heights,Inner Sunset,Mission,Nob Hill, Outer Richmond,Outer Sunset,Parkside,Potrero Hill, South Of Market,Western Addition 838 8.585934e+13 747347.3 *
- 5) squarefeet>=1772 242 5.416861e+13 976686.0 *
- 3) neighborhood=Castro-Upper Market,Financial District,Glen Park, Haight-Ashbury,Inner Richmond,Marina,Noe Valley,North Beach, Pacific Heights,Presidio Heights,Russian Hill,Seacliff, Twin Peaks,West Of Twin Peaks 772 4.759124e+14 1255028.0
- 6) squarefeet< 2119 591 1.962903e+14 1103036.0
- 12) neighborhood=Castro-Upper Market,Glen Park,Haight-Ashbury, Inner Richmond,Noe Valley,North Beach,Pacific Heights, Russian Hill,Twin Peaks, West Of Twin Peaks 479 1.185669e+14 1032675.0
- 24) month=2008-02-01,2008-03-01,2008-06-01,2008-07-01, 2008-08-01,2008-09-01,2008-10-01,2008-11-01,2008-12-01, 2009-01-01,2009-02-01,2009-03-01,2009-04-01,2009-05-01, 2009-06-01,2009-07-01 389 5.941085e+13 980348.3 *
- 25) month=2008-04-01,2008-05-01 90 5.348720e+13 1258844.0
- 50) longitude< -122.4142 81 1.550328e+13 1136562.0 *
- 51) longitude>=-122.4142 9 2.587193e+13 2359389.0 *
- 13) neighborhood=Financial District,Marina,Presidio Heights, Seacliff 112 6.521045e+13 1403951.0 *
- 7) squarefeet>=2119 181 2.213886e+14 1751315.0
- 14) neighborhood=Castro-Upper Market,Glen Park,Haight-Ashbury, Inner Richmond,Marina,Noe Valley,North Beach,Russian Hill, Twin Peaks,West Of Twin Peaks 159 1.032114e+14 1574642.0
- 28) month=2008-04-01,2008-06-01,2008-07-01,2008-10-01, 2009-02-01,2009-03-01,2009-04-01,2009-05-01, 2009-06-01,2009-07-01 77 2.070744e+13 1310922.0 *
- 29) month=2008-02-01,2008-03-01,2008-05-01,2008-08-01, 2008-09-01,2008-11-01,2008-12-01, 2009-01-01 82 7.212013e+13 1822280.0
- 58) lotsize< 3305.5 62 3.077240e+13 1598774.0 *
- 59) lotsize>=3305.5 20 2.864915e+13 2515150.0
- 118) neighborhood=Glen Park,Inner Richmond,Twin Peaks, West Of Twin Peaks 13 1.254738e+13 1962769.0 *
- 119) neighborhood=Castro-Upper Market,Marina, Russian Hill 7 4.768574e+12 3541000.0 *
- 15) neighborhood=Financial District,Pacific Heights, Presidio Heights,Seacliff 22 7.734568e+13 3028182.0
- 30) lotsize< 3473 12 7.263123e+12 2299500.0 *
- 31) lotsize>=3473 10 5.606476e+13 3902600.0 *

Notice the key on the second line of the output. (Each line contains the node number, description of the split, number of observations under that node in the tree, deviance, and predicted value.) This tree model tells us some obvious things, like location and

size are good predictors of price. Reading a textual description of an `rpart` object is somewhat confusing. The method `plot.rpart` will draw the tree structure in an `rpart` object:

```
plot(x, uniform=FALSE, branch=1, compress=FALSE, nspace,
     margin=0, minbranch=.3, ...)
```

You can label the tree using `text.rpart`:

```
text(x, splits=TRUE, label, FUN=text, all=FALSE,
     pretty=NULL, digits=getOption("digits") - 3, use.n=FALSE,
     fancy=FALSE, fwidth=0.8, fheight=0.8, ...)
```

For both functions, the argument `x` specifies the `rpart` object; the other options control the way the output looks. See the help file for more information about these parameters. As an example, let's plot the tree we just created above:

```
> plot(sf.price.model.rpart, uniform=TRUE, compress=TRUE, lty=3, branch=0.7)
> text(sf.price.model.rpart, all=TRUE, digits=7, use.n=TRUE, cex=0.4, xpd=TRUE,)
```

As you can see from Figure 20-4, it's difficult to read a small picture of a big tree. To keep the tree somewhat readable, we have abbreviated neighborhood names to single letters (corresponding to their order in the factor). Sometimes, the function `draw.tree` in the package `maptree` can produce prettier diagrams. See "Classification Tree Models" on page 446 for more details.

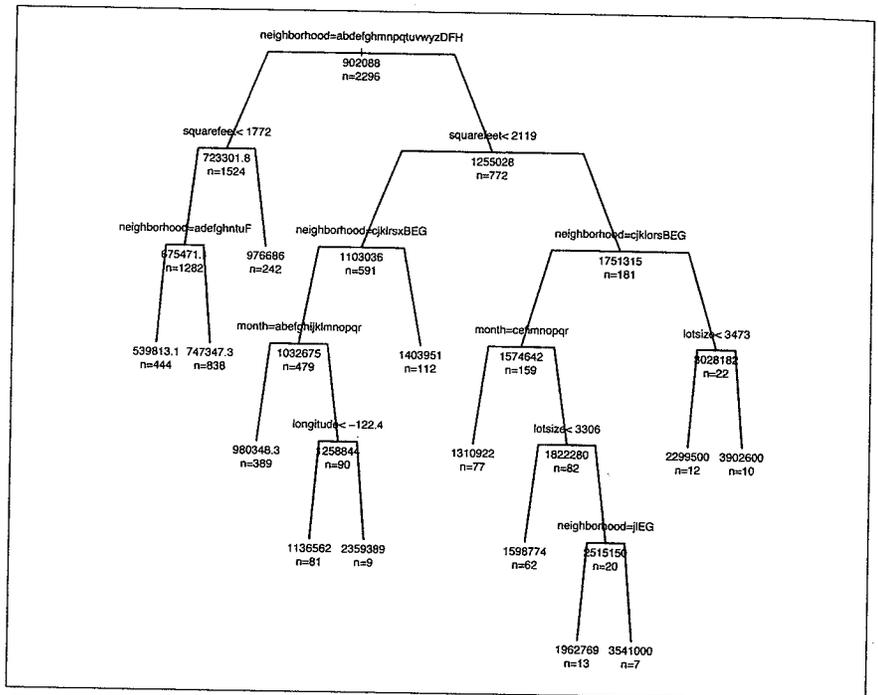


Figure 20-4. `rpart` tree for the San Francisco home sales model

g a textual description of an rpart object is .rpart will draw the tree structure in an

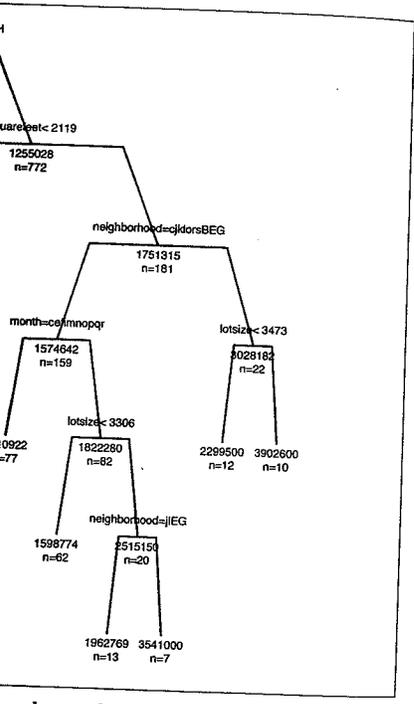
```
compress=FALSE, nspace,
```

```
, all=FALSE,
'digits') - 3, use.n=FALSE,
t=0.8, ...)
```

ifies the rpart object; the other options help file for more information about these tree we just created above:

```
TRUE,compress=TRUE,lty=3,branch=0.7)
,digits=7,use.n=TRUE,cex=0.4,xpd=TRUE,)
```

cult to read a small picture of a big tree. have abbreviated neighborhood names to r in the factor). Sometimes, the function uce prettier diagrams. See "Classification S.



the sales model

To predict a value with a tree model, you would start at the top of the tree and follow the tree down, depending on the rules for a specific observation. For example, suppose that we had a property in Pacific Heights with 2,500 square feet of living space and a lot size of 5,000 square feet. We would traverse the tree starting at node 1, then go to node 3, then node 7, then node 15, and, finally, land on node 31. The estimated price of this property would be \$3,902,600.

There are a number of other functions available in the rpart package for viewing (or manipulating) tree objects. To view the approximate r-square and relative error at each split, use the function rsq.rpart. The graphical output is shown in Figure 20-5; here is the output on the R console:

```
> rsq.rpart(sf.price.model.rpart)

Regression tree:
rpart(formula = price ~ bedrooms + squarefeet + lotsize + latitude +
longitude + neighborhood + month, data = sanfrancisco.home.sales.training)

Variables actually used in tree construction:
[1] longitude lotsize month neighborhood squarefeet

Root node error: 8.0587e+14/2296 = 3.5099e+11

n= 2296
```

	CP	nsplit	rel error	xerror	xstd
1	0.179780	0	1.00000	1.00038	0.117779
2	0.072261	1	0.82022	0.83652	0.105103
3	0.050667	2	0.74796	0.83211	0.096150
4	0.022919	3	0.69729	0.80729	0.094461
5	0.017395	4	0.67437	0.80907	0.096560
6	0.015527	5	0.65698	0.82365	0.097687
7	0.015511	6	0.64145	0.81720	0.097579
8	0.014321	7	0.62594	0.81461	0.097575
9	0.014063	9	0.59730	0.81204	0.097598
10	0.011032	10	0.58323	0.81559	0.097691
11	0.010000	12	0.56117	0.80271	0.096216

As you can probably tell, the initial tree was a bit complicated. You can remove nodes where the cost/complexity trade-off isn't great by using the prune function:

```
prune(tree, cp, ...)
```

The argument cp is a complexity parameter that controls how much to trim the tree. To help choose a complexity parameter, try the function plotcp:

```
plotcp(x, minline = TRUE, lty = 3, col = 1,
upper = c("size", "splits", "none"), ...)
```

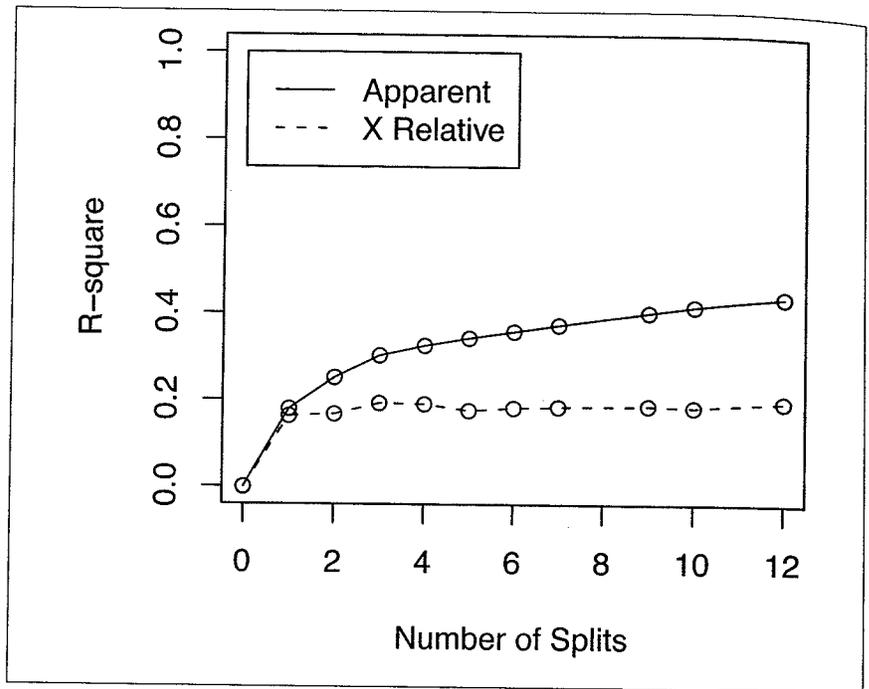


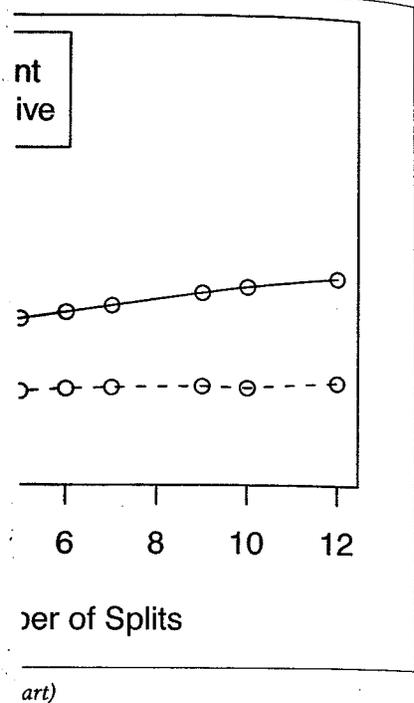
Figure 20-5. Plot from `rsq.rpart(sf.price.model.rpart)`

The `plotcp` function plots tree sizes and relative errors for different parameters of the complexity parameter. For the example above, it looks like a value of .011 is a good balance between complexity and performance. Here is the pruned model (see also Figure 20-6):

```
> prune(sf.price.model.rpart, cp=0.11)
n= 2296
```

```
node), split, n, deviance, yval
* denotes terminal node
```

- 1) root 2296 8.058726e+14 902088.0
- 2) neighborhood=Bayview,Bernal Heights,Chinatown,Crocker Amazon,
Diamond Heights,Downtown,Excelsior,Inner Sunset,Lakeshore,Mission,
Nob Hill,Ocean View,Outer Mission,Outer Richmond,Outer Sunset,
Parkside,Potrero Hill,South Of Market,Visitacion Valley,
Western Addition 1524 1.850806e+14 723301.8 *
- 3) neighborhood=Castro-Upper Market,Financial District,Glen Park,
Haight-Ashbury, Inner Richmond,Marina,Noe Valley,North Beach,
Pacific Heights,Presidio Heights,Russian Hill,Seacliff,Twin Peaks,
West Of Twin Peaks 772 4.759124e+14 1255028.0 *



relative errors for different parameters of above, it looks like a value of .011 is a good choice. Here is the pruned model (see

Marina, Crocker Amazon, Inner Sunset, Lakeshore, Mission, Outer Richmond, Outer Sunset, Visitacion Valley, 723301.8 *
 Financial District, Glen Park, Ina, Noe Valley, North Beach, Russian Hill, Seacliff, Twin Peaks, 4 1255028.0 *

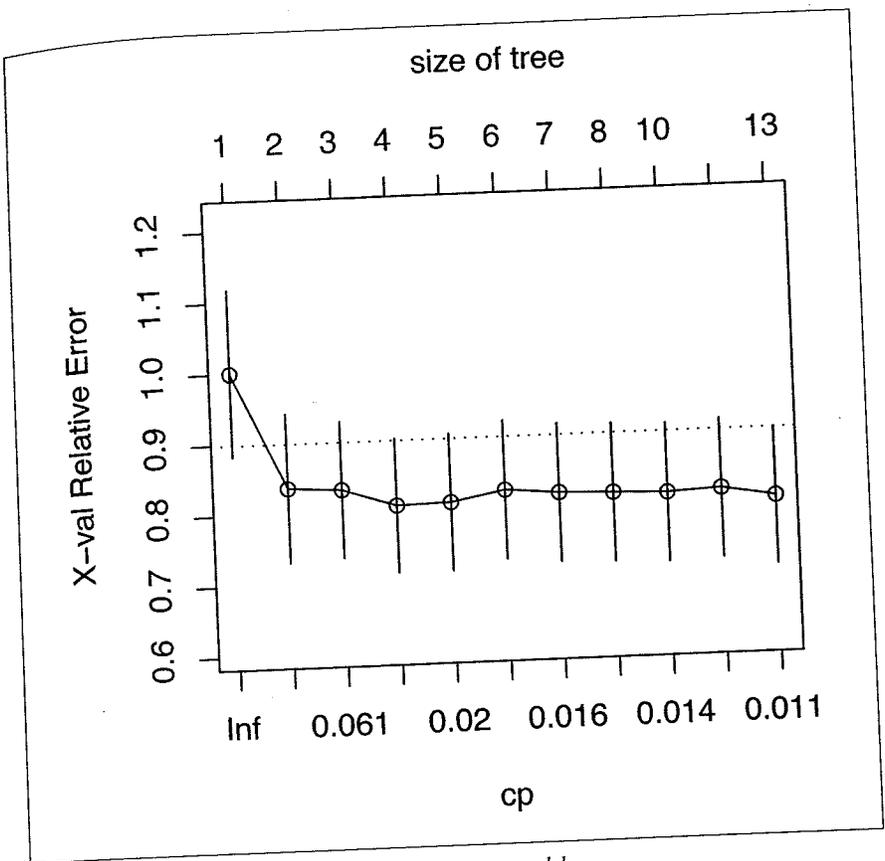


Figure 20-6. Output of plotcp for the sf.prices.rpart model

And if you're curious, here is the error of this model on the training and test populations:

```
> calculate_rms_error(sf.price.model.rpart,
+ sanfrancisco.home.sales.training,
+ sanfrancisco.home.sales.testing,
+ "price")
train.err test.err
443806.8 564986.8
```

The units, incidentally, are dollars.

There is an alternative implementation of CART trees available with R through the tree package. It was written by W. N. Venables, one of the authors of [Venables2002]. He notes that tree can give more explicit output while running, but recommends rpart for most users.

Patient rule induction method

Another technique for building rule-based models is the patient rule induction method (PRIM) algorithm. PRIM doesn't actually build trees. Instead, it partitions the data into a set of "boxes" (in p dimensions). The algorithm starts with a box containing all the data and then shrinks the box one side at a time, trying to maximize the average value in the box. After reaching a minimum number of observations in the box, the algorithm tries expanding the box again, as long as it can increase the average value in the box. When the algorithm finds the best initial box, it then repeats the process on the remaining observations, until there are no observations left. The algorithm leads to a set of rules that can be used to predict values.

To try out PRIM in R, there are functions in the library `prim`:

```
prim.box(x, y, box.init=NULL, peel.alpha=0.05, paste.alpha=0.01,
        mass.min=0.05, threshold, pasting=TRUE, verbose=FALSE,
        threshold.type=0)

prim.hdr(prim, threshold, threshold.type)
prim.combine(prim1, prim2)
```

Bagging for regression

Bagging (or bootstrap aggregation) is a technique for building predictive models based on other models (most commonly trees). The idea of bagging is to use bootstrapping to build a number of different models and then average the results. The weaker models essentially form a committee to vote for a result, which leads to more accurate predictions.

To build regression bagging models in R, you can use the function `bagging` in the `ipred` library:

```
library(ipred)
bagging(formula, data, subset, na.action=na.rpart, ...)
```

The `formula`, `data`, `subset`, and `na.action` arguments work the same way as in most modeling functions. The additional arguments are passed on to the function `ipredbag`, which does all the work (but doesn't have a method for formulas):

```
ipredbag(y, X=NULL, nbagg=25, control=rpart.control(xval=0),
        comb=NULL, coob=FALSE, ns=length(y), keepX = TRUE, ...)
```

You can specify the number of trees to build by `nbagg`, control parameters for `rpart` through `control`, a list of models to use for double-bagging through `comb`, `coob` to indicate if an out-of-bag error rate should be computed, and `ns` to specify the number of observations to draw from the learning sample.

Let's try building a model on the pricing data using bagging. We'll pick 100 `rpart` trees (for fun):

```
> sf.price.model.bagging <- bagging(
+ price~bedrooms+squarefeet+lotsize+latitude+
+ longitude+neighborhood+month,
+ data=sanfrancisco.home.sales.training, nbagg=100)
> summary(sf.price.model.bagging)
```

sed models is the patient rule induction
t actually build trees. Instead, it partitions
ensions). The algorithm starts with a box
e box one side at a time, trying to maximize
ng a minimum number of observations in
e box again, as long as it can increase the
m finds the best initial box, it then repeats
s, until there are no observations left. The
e used to predict values.

in the library `prim`:

```
alpha=0.05, paste.alpha=0.01,  
sig=TRUE, verbose=FALSE,
```

ype)

chnique for building predictive models
ees). The idea of bagging is to use boot-
odels and then average the results. The
to vote for a result, which leads to more

ou can use the function `bagging` in the

```
on=na.rpart, ...)
```

arguments work the same way as in
guments are passed on to the function
sn't have a method for formulas):

```
rpart.control(xval=0),  
, ns=length(y), keepX = TRUE, ...)
```

ild by `nbagg`, control parameters for
use for double-bagging through `comb`,
ould be computed, and `ns` to specify
learning sample.

a using bagging. We'll pick 100 `rpart`

```
latitude+
```

```
g, nbagg=100)
```

	Length	Class	Mode
y	1034	-none-	numeric
X	7	data.frame	list
mtrees	100	-none-	list
OOB	1	-none-	logical
comb	1	-none-	logical
call	4	-none-	call

Let's take a quick look at how bagging worked on this data set:

```
> calculate_rms_error(sf.price.model.bagging,  
+ sanfrancisco.home.sales.training,  
+ sanfrancisco.home.sales.testing,  
+ "price")  
train.err test.err  
491003.8 582056.5
```

Boosting for regression

Boosting is a technique that's closely related to bagging. Unlike bagging, the individual models don't all have equal votes. Better models are given stronger votes.

You can find a variety of tools for computing boosting models in R in the package `mboost`. The function `blackboost` builds boosting models from regression trees, `glmboost` from general linear models, and `gamboost` for boosting based on additive models. Here, we'll just build a model using regression trees:

```
> library(mboost)  
Loading required package: modeltools  
Loading required package: stats4  
Loading required package: party  
Loading required package: grid  
Loading required package: coin  
Loading required package: mvtnorm  
Loading required package: zoo  
> sf.price.model.blackboost <- blackboost(  
+ price~bedrooms+squarefeet+lotsize+latitude+  
+ longitude+neighborhood+month,  
+ data=sanfrancisco.home.sales.training)
```

Here is a summary of the model object:

```
> summary(sf.price.model.blackboost)
```

	Length	Class	Mode
ensemble	100	-none-	list
fit	2296	-none-	numeric
offset	1	-none-	numeric
ustart	2296	-none-	numeric
risk	100	-none-	numeric
control	8	boost_control	list
family	1	boost_family	S4
response	2296	-none-	numeric
weights	2296	-none-	numeric
update	1	-none-	function
tree_controls	1	TreeControl	S4
data	1	LearningSampleFormula	S4

predict	1	-none-	function
call	3	-none-	call

And here is a quick evaluation of the performance of this model:

```
> calculate_rms_error(sf.price.model.blackboost,
+   sanfrancisco.home.sales.training,
+   sanfrancisco.home.sales.testing,
+   "price")
train.err test.err
1080520 1075810
```

Random forests for regression

Random forests are another technique for building predictive models using trees. Like boosting and bagging, random forests work by combining a set of other tree models. Unlike boosting and bagging, which use an existing algorithm like CART to build a series of trees from a random sample of the observations in the test data, random forests build trees from a random sample of the columns in the test data.

Here's a description of how the random forest algorithm creates the underlying trees (using variable names from the R implementation):

1. Take a sample of size `sampsiz` from the training data.
2. Begin with a single node.
3. Run the following algorithm, starting with the starting node:
 - A. Stop if the number of observations is less than `nodesize`.
 - B. Select `mtry` variables (at random).
 - C. Find the variable and value that does the "best" job splitting the observations. (Specifically, the algorithm uses MSE [mean square error] to measure regression error, and Gini to measure classification error.)
 - D. Split the observations into two nodes.
 - E. Call step A on each of these nodes.

Unlike trees generated by CART, trees generated by random forest aren't pruned; they're just grown to a very deep level.

For regression problems, the estimated value is calculated by averaging the prediction of all the trees in the forest. For classification problems, the prediction is made by predicting the class using each tree in the forest and then outputting the choice that received the most votes.

To build random forest models in R, use the `randomForest` function in the random forest package:

```
library(randomForest)
## S3 method for class 'formula':
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
## Default S3 method:
randomForest(x, y=NULL, xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
                 max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
```

function
call

performance of this model:

el.blackboost,
ing,
ing,

or building predictive models using trees.
sts work by combining a set of other tree
high use an existing algorithm like CART
ample of the observations in the test data,
a sample of the columns in the test data.

rest algorithm creates the underlying trees
(mentation):

the training data.

with the starting node:
ns is less than nodesize.

does the "best" job splitting the observa-
uses MSE [mean square error] to measure
sure classification error.)

odes.

s.

erated by random forest aren't pruned;

re is calculated by averaging the predic-
cation problems, the prediction is made
e forest and then outputting the choice

re RandomForest function in the random

ubset, na.action=na.fail)

est=NULL, ntree=500,
s.factor(y))
lse floor(sqrt(ncol(x))),

```
replace=TRUE, classwt=NULL, cutoff, strata,  
sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),  
nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,  
importance=FALSE, localImp=FALSE, nPerm=1,  
proximity, oob.prox=proximity,  
norm.votes=TRUE, do.trace=FALSE,  
keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,  
keep.inbag=FALSE, ...)
```

Unlike some other functions we've seen so far, `randomForest` will fail if called on data with missing observations. So, we'll set `na.action=na.omit` to omit NA values. Additionally, `randomForest` cannot handle categorical predictors with more than 32 levels, so we will cut out the neighborhood variable:

```
> sf.price.model.randomforest <- randomForest(  
+ price~bedrooms+squarefeet+lotsize+latitude+  
+ longitude+month,  
+ data=sanfrancisco.home.sales.training,  
+ na.action=na.omit)
```

The print method for `randomForest` objects returns some useful information about the fit:

```
> sf.price.model.randomforest
```

Call:

```
randomForest(formula = price ~ bedrooms + squarefeet + lotsize +  
latitude + longitude + month,  
data = sanfrancisco.home.sales.training,  
na.action = na.omit)
```

```
Type of random forest: regression  
Number of trees: 500  
No. of variables tried at each split: 2
```

```
Mean of squared residuals: 258521431697  
% Var explained: 39.78
```

Here is how the model performed:

```
> calculate_rms_error(sf.price.model.randomforest,  
+ na.omit(sanfrancisco.home.sales.training),  
+ na.omit(sanfrancisco.home.sales.testing),  
+ "price")  
train.err test.err  
241885.2 559461.0
```

As a point of comparison, here are the results of the `rpart` model, also with NA values omitted:

```
> calculate_rms_error(sf.price.model.rpart,  
+ na.omit(sanfrancisco.home.sales.training),  
+ na.omit(sanfrancisco.home.sales.testing),  
+ "price")  
train.err test.err  
442839.6 589583.1
```

MARS

Another popular algorithm for machine learning is multivariate adaptive regression splines, or MARS. MARS works by splitting input variables into multiple *basis functions* and then fitting a linear regression model to those basis functions. The basis functions used by MARS come in pairs: $f(x) = \{x - t \text{ if } x > t, 0 \text{ otherwise}\}$ and $g(x) = \{t - x \text{ if } x < t, 0 \text{ otherwise}\}$. These functions are *piecewise linear* functions. The value t is called a *knot*.

MARS is closely related to CART. Like CART, it begins by building a large model and then prunes back unneeded terms until the best model is found. The MARS algorithm works by gradually building up a model out of basis functions (or products of basis functions) until it reaches a predetermined depth. This results in an overfitted, overly complex model. Then the algorithm deletes terms from the model, one by one, until it has pared back everything but a constant term. At each stage, the algorithm uses generalized cross-validation (GCV) to measure how well each model fits. Finally, the algorithm returns the model with the best cost/benefit ratio.

To fit a model using MARS in R, use the function `earth` in the package `earth`:

```
library(earth)
earth(formula = stop("no 'formula' arg"),
      data, weights = NULL, wp = NULL, scale.y = (NCOL(y)==1), subset = NULL,
      na.action = na.fail, glm = NULL, trace = 0,
      keepxy = FALSE, nfold=0, stratify=TRUE, ...)
```

Arguments to `earth` include the following.

Argument	Description	Default
<code>formula</code>	A formula describing the relationship between the response and the predictor variables.	<code>stop("no 'formula' arg")</code>
<code>data</code>	A data frame containing the training data.	
<code>weights</code>	An optional vector of weights to use for the fitting data. (It is especially optional, because it is not supported as of <code>earth</code> version 2.3-2.)	NULL
<code>wp</code>	A numeric vector of response weights. Must include a value for each column of <code>y</code> .	NULL
<code>scale.y</code>	A numeric value specifying whether to scale <code>y</code> in the forward pass. (See the help file for more information.)	<code>(NCOL(y)==1)</code>
<code>subset</code>	A logical vector specifying which observations from <code>data</code> to include.	NULL
<code>na.action</code>	A function specifying how to treat missing values. Only <code>na.fail</code> is currently supported.	<code>na.fail</code>
<code>glm</code>	A list of arguments to <code>glm</code> .	NULL
<code>trace</code>	A numeric value specifying whether to print a "trace" of the algorithm execution.	0
<code>keepxy</code>	A logical value specifying whether to keep <code>x</code> and <code>y</code> (or <code>data</code>), <code>subset</code> , and <code>weights</code> in the model object. (Useful if you plan to use <code>update</code> to modify the model at a later time.)	FALSE
<code>nfold</code>	A numeric value specifying the number of cross-validation folds.	0
<code>stratify</code>	A logical value specifying whether to stratify the cross-validation folds.	TRUE

Learning is multivariate adaptive regression splines (MARS). It takes input variables into multiple basis functions and combines them into a model. The basis functions are piecewise linear functions. The model is built by adding terms until the best model is found. The MARS algorithm deletes terms from the model, one by one, until a constant term. At each stage, the model is evaluated using cross-validation (CV) to measure how well each model performs with the best cost/benefit ratio.

The `earth` function in the package `earth` starts by building a large model and then prunes it to find the best model. The MARS algorithm starts with a model of depth 1 and increases the depth until the model is overfitted. This results in an overfitted model. The `earth` function uses cross-validation (CV) to measure how well each model performs with the best cost/benefit ratio.

```

earth(
  y = (NCOL(y)==1), subset = NULL,
  n = 0,
  ...
)

```

	Default
response and the predictor	stop("no 'formula' arg")
data. (It is especially optional, see ?earth)	NULL
value for each column of y.	NULL
forward pass. (See the help file)	(NCOL(y)==1)
data to include.	NULL
if na.fail is currently TRUE	na.fail
if the algorithm execution.	NULL
lambda, subset, and weights to modify the model at a	0
number of cross-validation folds.	FALSE
number of bootstrap folds.	0
validation folds.	TRUE

Argument	Description	Default
	Additional options are passed to <code>earth.fit</code> . There are many, many options available to tune the fitting process. See the help file for <code>earth</code> for more information.	

The `earth` function is very flexible. By default, `lm` is used to fit models. Note that `glm` can be used instead to allow finer control of the model. The function `earth` can't cope directly with missing values in the data set. To deal with NA values, you need to explicitly deal with them in the input data. You could, for example, impute median values or model imputed values. In the example below, I picked the easy solution and just used the `na.omit` function to filter them out.

Let's build an `earth` model on the San Francisco home sales data set. We'll add the `trace=1` option to show some details of the computation:

```

> sf.price.model.earth <- earth(
+   price~bedrooms+squarefeet+latitude+
+   longitude+neighborhood+month,
+   data=na.omit(sanfrancisco.home.sales.training), trace=1)
x is a 957 by 54 matrix: 1=bedrooms, 2=squarefeet, 3=latitude,
4=longitude, 5=neighborhoodBernalHeights, 6=neighborhoodCastro-UpperMarket,
7=neighborhoodChinatown, 8=neighborhoodCrockerAmazon,
9=neighborhoodDiamondHeights, 10=neighborhoodDowntown,
11=neighborhoodExcelsior, 12=neighborhoodFinancialDistrict,
13=neighborhoodGlenPark, 14=neighborhoodHaight-Ashbury,
15=neighborhoodInnerRichmond, 16=neighborhoodInnerSunset,
17=neighborhoodLakeshore, 18=neighborhoodMarina,
19=neighborhoodMission, 20=neighborhoodNobHill,
21=neighborhoodNoeValley, 22=neighborhoodNorthBeach,
23=neighborhoodOceanView, 24=neighborhoodOuterMission,
25=neighborhoodOuterRichmond, 26=neighborhoodOuterSunset,
27=neighborhoodPacificHeights, 28=neighborhoodParkside,
29=neighborhoodPotreroHill, 30=neighborhoodPresidioHeights,
31=neighborhoodRussianHill, 32=neighborhoodSeaCliff,
33=neighborhoodSouthOfMarket, 34=neighborhoodTwinPeaks,
35=neighborhoodVisitacionValley, 36=neighborhoodWestOfTwinPeaks,
37=neighborhoodWesternAddition, 38=month2008-03-01, 39=month2008-04-01,
40=month2008-05-01, 41=month2008-06-01, 42=month2008-07-01,
43=month2008-08-01, 44=month2008-09-01, 45=month2008-10-01,
46=month2008-11-01, 47=month2008-12-01, 48=month2009-01-01,
49=month2009-02-01, 50=month2009-03-01, 51=month2009-04-01,
52=month2009-05-01, 53=month2009-06-01, 54=month2009-07-01
y is a 957 by 1 matrix: 1=price
Forward pass term 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,
30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58,
60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80
Reached delta RSq threshold (DeltaRSq 0.000861741 < 0.001)
After forward pass GRSq 0.4918 RSq 0.581
Prune method "backward" penalty 2 nprune 44: selected 36 of 44 terms, and 26
of 54 predictors
After backward pass GRSq 0.5021 RSq 0.5724

```

The earth object has an informative print method, showing the function call and statistics about the model fit:

```
> sf.price.model.earth
Selected 31 of 41 terms, and 22 of 55 predictors
Importance: squarefeet, neighborhoodPresidioHeights,
latitude, neighborhoodSeacliff, neighborhoodNoeValley,
neighborhoodCastro-UpperMarket, neighborhoodNobHill,
lotsize, month2008-07-01, neighborhoodWesternAddition, ...
Number of terms at each degree of interaction: 1 30 (additive model)
GCV 216647913449   RSS 1.817434e+14   GRSq 0.5162424   RSq 0.5750596
```

The summary method will show the basis functions for the fitted model in addition to information about the fit:

```
> summary(sf.price.model.earth)
Call: earth(formula=price~bedrooms+squarefeet+lotsize+latitude+
longitude+neighborhood+month,
data=na.omit(sanfrancisco.home.sales.training))
```

	coefficients
(Intercept)	1452882
h(bedrooms-3)	130018
h(bedrooms-5)	-186130
h(squarefeet-2690)	81
h(2690-squarefeet)	-178
h(lotsize-2495)	183
h(lotsize-3672)	-141
h(latitude-37.7775)	-112301793
h(37.7775-latitude)	-7931270
h(latitude-37.7827)	420380414
h(latitude-37.7888)	-188726623
h(latitude-37.8015)	-356738902
h(longitude- -122.464)	-6056771
h(-122.438-longitude)	-6536227
neighborhoodCastro-UpperMarket	338549
neighborhoodChinatown	-1121365
neighborhoodInnerSunset	-188192
neighborhoodMarina	-2000574
neighborhoodNobHill	-2176350
neighborhoodNoeValley	368772
neighborhoodNorthBeach	-2395955
neighborhoodPacificHeights	-1108284
neighborhoodPresidioHeights	1146964
neighborhoodRussianHill	-1857710
neighborhoodSeacliff	2422127
neighborhoodWesternAddition	-442262
month2008-03-01	181640
month2008-04-01	297754
month2008-05-01	187684
month2008-07-01	-322801
month2008-10-01	115435

```
Selected 31 of 41 terms, and 22 of 55 predictors
Importance: squarefeet, neighborhoodPresidioHeights, latitude,
neighborhoodSeacliff, neighborhoodNoeValley,
```

t method, showing the function call and

predictors
residioHeights,
ghborhoodNoeValley,
ghborhoodNobHill,
oodWesternAddition, ...
raction: 1 30 (additive model)
GRSq 0.5162424 RSq 0.5750596
unctions for the fitted model in addition

arefeet+lotsize+latitude+

.sales.training))

ients
52882
30018
36130
81
-178
183
-141
1793
1270
0414
6623
3902
5771
5227
3549
365
192
574
350
772
955
284
964
710
127
162
140
54
84
01
35

dictors
dioHeights, latitude,
lley,

neighborhoodCastro-UpperMarket, neighborhoodNobHill,
lotsize, month2008-07-01, neighborhoodWesternAddition, ...
Number of terms at each degree of interaction: 1 30 (additive model)
GCV 216647913449 RSS 1.817434e+14 GRSq 0.5162424 RSq 0.5750596

The output of summary includes a short synopsis of variable importance in the model.
You can use the function evimp to return a matrix showing the relative importance
of variables in the model:

```
evimp(obj, trim=TRUE, sqrt.=FALSE)
```

The argument obj specifies an earth object, trim specifies whether to delete rows in
the matrix for variables that don't appear in the fitted model, and sqrt. specifies
whether to take the square root of the GCV and RSS importances before normalizing
them. For the example above, here is the output:

```
> evimp(sf.price.model.earth)
```

	col	used	nsubsets	gcv
squarefeet	2	1	30	100.0000000 1
neighborhoodPresidioHeights	31	1	29	62.71464260 1
latitude	4	1	28	45.85760472 1
neighborhoodSeacliff	33	1	27	33.94468291 1
neighborhoodNoeValley	22	1	25	22.55538880 1
neighborhoodCastro-UpperMarket	7	1	24	18.84206296 1
neighborhoodNobHill	21	1	23	14.79044745 1
lotsize	3	1	21	10.94876414 1
month2008-07-01	43	1	20	9.54292889 1
neighborhoodWesternAddition	38	1	19	7.47060804 1
longitude	5	1	18	6.37068263 1
neighborhoodNorthBeach	23	1	16	4.64098864 1
neighborhoodPacificHeights	28	1	14	3.21207679 1
neighborhoodMarina	19	1	13	3.25260354 0
neighborhoodRussianHill	32	1	12	3.02881439 1
month2008-04-01	40	1	10	2.22407575 1
bedrooms	1	1	8	1.20894174 1
neighborhoodInnerSunset	17	1	5	0.54773450 1
month2008-03-01	39	1	4	0.38402626 1
neighborhoodChinatown	8	1	3	0.24940165 1
month2008-10-01	46	1	2	0.15317304 1
month2008-05-01	41	1	1	0.09138073 1

	rss
squarefeet	100.0000000 1
neighborhoodPresidioHeights	65.9412651 1
latitude	50.3490370 1
neighborhoodSeacliff	39.2669043 1
neighborhoodNoeValley	28.3043535 1
neighborhoodCastro-UpperMarket	24.6223129 1
neighborhoodNobHill	20.6738425 1
lotsize	16.5523065 1
month2008-07-01	14.9572215 1
neighborhoodWesternAddition	12.8021914 1
longitude	11.4928253 1
neighborhoodNorthBeach	9.2983004 1
neighborhoodPacificHeights	7.3843377 1
neighborhoodMarina	7.0666997 1
neighborhoodRussianHill	6.5297824 1

Regression

month2008-04-01	5.1687163	1
bedrooms	3.6503604	1
neighborhoodInnerSunset	2.1002700	1
month2008-03-01	1.6337090	1
neighborhoodChinatown	1.1922930	1
month2008-10-01	0.7831185	1
month2008-05-01	0.4026390	1

The function `plot.earth` will plot model selection, cumulative distribution of residuals, residuals versus fitted values, and the residual Q-Q plot for an earth object:

```
> plot(sf.price.model.earth)
```

The output of this call is shown in Figure 20-7. There are many options for this function that control the output; see the help file for more information. Another useful function for looking at earth objects is `plotmo`:

```
> plotmo(sf.price.model.earth)
```

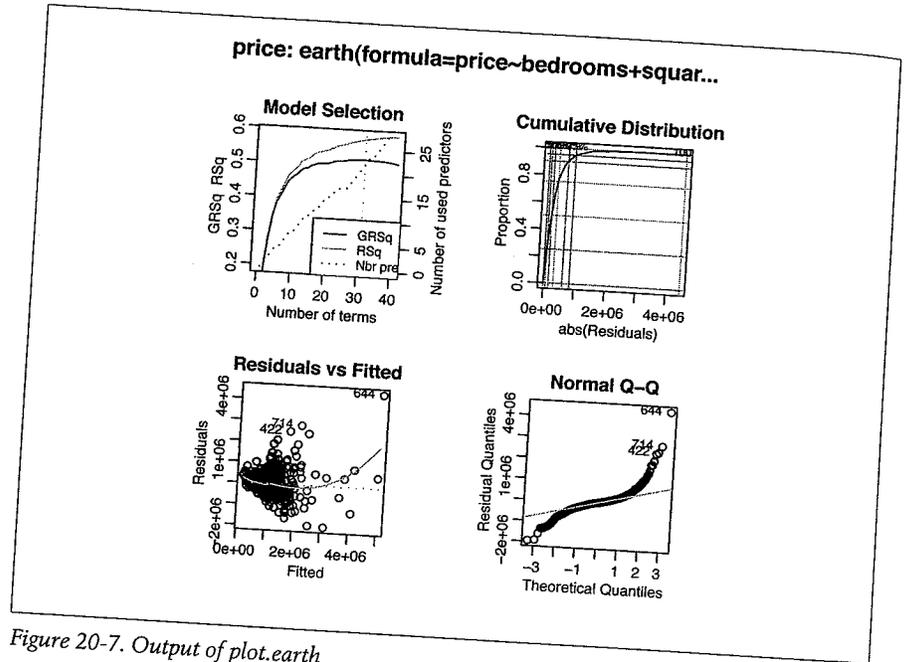


Figure 20-7. Output of `plot.earth`

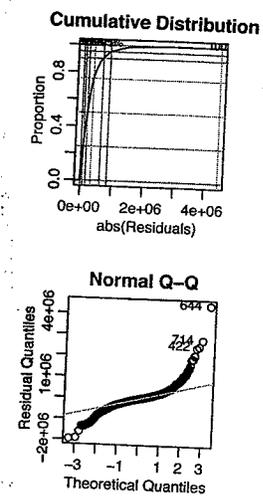
The `plotmo` function plots the predicted model response when varying one or two predictors while holding other predictors constant. The output of `plotmo` for the San Francisco home sales data set is shown in Figure 20-8.

.1687163 1
 .6503604 1
 .1002700 1
 .6337090 1
 .1922930 1
 .7831185 1
 .4026390 1

selection, cumulative distribution of residual
 the residual Q-Q plot for an earth object:

re 20-7. There are many options for this
 : help file for more information. Another
 ts is plotmo:

ice~bedrooms+squar...



el response when varying one or two
 stant. The output of plotmo for the San
 ure 20-8.

earth(formula=price~bedroom...

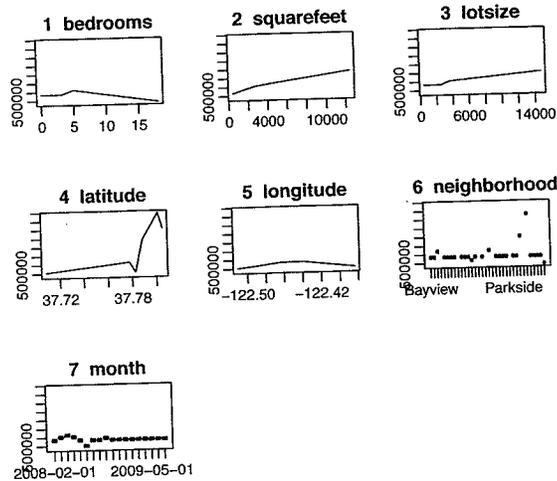


Figure 20-8. Output of plotmo

For the fun of it, let's look at the predictions from earth:

```
> calculate_rms_error(sf.price.model.earth,
+   na.omit(sanfrancisco.home.sales.training),
+   na.omit(sanfrancisco.home.sales.testing),
+   "price")
train.err test.err
435786.1 535941.5
```

Neural Networks

Neural networks are a very popular type of statistical model. Neural networks were originally designed to approximate how neurons work in the human brain; much of the original research on neural networks came from artificial intelligence researchers. Neural networks are very flexible and can be used to model a large number of different problems. By changing the structure of neural networks, it's possible to model some very complicated nonlinear relationships. Neural networks are so popular that there are entire academic journals devoted to them (such as *Neural Networks*, published by Elsevier).

The base distribution of R includes an implementation of one of the simplest types of neural networks: single-hidden-layer neural networks. Even this simple form of neural network can be used to model some very complicated relationships in data sets. Figure 20-9 is a graphical representation of what these neural networks look like. As you can see, each input value feeds into each "hidden layer" node. The output of each hidden-layer node feeds into each output node. What the modeling

function actually does is to estimate the weights for each input into each hidden node and output node.

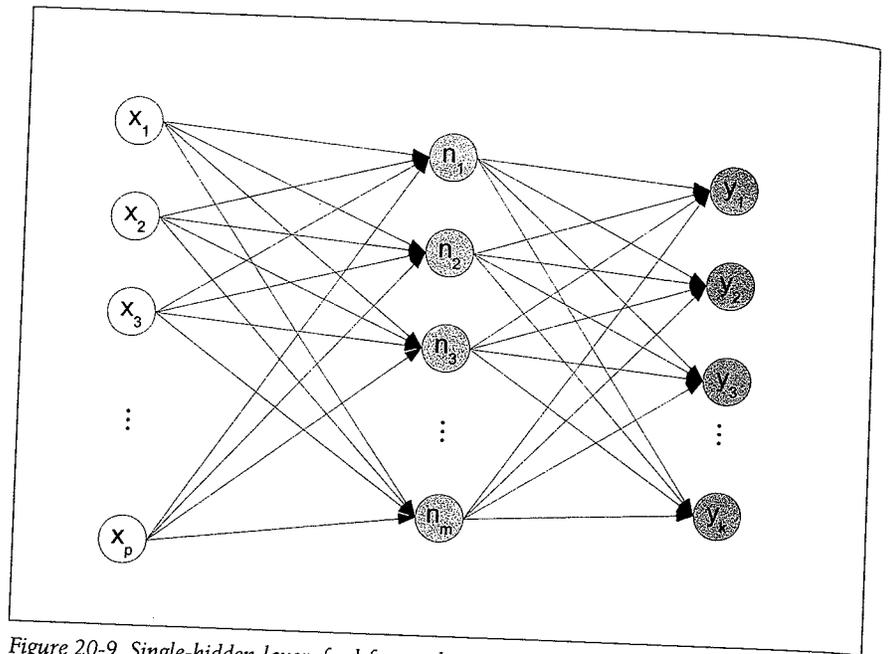


Figure 20-9. Single-hidden-layer, feed-forward neural network

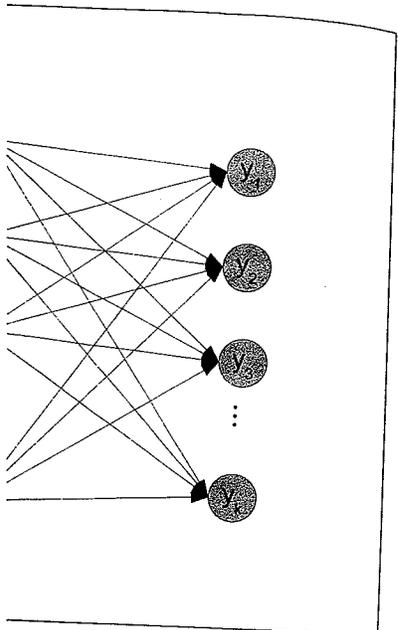
The diagram omits two things: bias units and skip layer connectors. A *bias unit* is just a constant input term; it lets a constant term be mixed into each unit. *Skip layer connections* allow values from the inputs to be mixed into the outputs, skipping over the hidden layer. Both of these additions are included in the R implementation.

In equation form, here is the formula for neural network models:

$$y_k = g_o \left(\alpha_k + \sum_{i=1}^{i=m} w_{i,k} g_i \left(\alpha_i + \sum_{j=1}^{j=p} w_{j,i} x_j \right) \right)$$

The function g_i used for the hidden nodes is the sigmoid function: $\sigma(x) = e^x / (1 + e^x)$. The function used for the output nodes is usually the identity function for regression, and the softmax function for classification. (We'll discuss the softmax function in "Neural Networks" on page 450.) For classification models, there are k outputs corresponding to the different levels. For regression models, there is only one output node.

weights for each input into each hidden



al network

skip layer connectors. A bias unit is mixed into each unit. Skip layer mixed into the outputs, skipping over included in the R implementation.

network models:

the sigmoid function: $\sigma(x) = e^x / (1 + e^x)$ usually the identity function for regression. (We'll discuss the softmax or classification models, there are or regression models, there is only

To fit neural network models, use the function nnet in the package nnet:

```
library(nnet)
## S3 method for class 'formula':
nnet(formula, data, weights, ...,
      subset, na.action, contrasts = NULL)

## Default S3 method:
nnet(x, y, weights, size, Wts, mask,
      linout = FALSE, entropy = FALSE, softmax = FALSE,
      censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
      maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
      abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Arguments to nnet include the following.

Argument	Description	Default
formula	A formula describing the relationship between the response and the predictor variables.	
data	A data frame containing the training data.	
weights	An optional vector of weights to use for the training data.	
...	Additional arguments passed to other functions (such as the nnet.default if using the nnet method, or optim).	
subset	An optional vector specifying the subset of observations to use in fitting the model.	
na.action	A function specifying how to treat missing values.	
contrasts	A list of factors to use for factors that appear in the model.	NULL
size	Number of units in the hidden layer.	Randomly chosen, if not specified
Wts	Initial parameter vector.	
mask	A logical vector indicating which parameters should be optimized.	All parameters
linout	Use linout=FALSE for logistic output units, linout=TRUE for linear units.	FALSE
entropy	A logical value specifying whether to use entropy/maximum conditional likelihood fitting.	FALSE
softmax	A logical value specifying whether to use a softmax/log-linear model and maximum conditional likelihood fitting.	FALSE
censored	A logical value specifying whether to treat the input data as censored data. (By default, a response variable value of c(1, 0, 1) means "both classes 1 and 3." If we treat the data as censored, then c(1, 0, 1) is interpreted to mean "not 2, but possibly 1 or 3.")	FALSE
skip	A logical value specifying whether to add skip-layer connections from input to output.	FALSE
rang	A numeric value specifying the range for initial random weights. Weights are chosen between -rang and rang.	0.7
decay	A numeric parameter for weight decay.	0

Argument	Description	Default
maxit	Maximum number of iterations.	100
Hess	A logical value specifying whether to return the Hessian of fit.	FALSE
trace	A logical value specifying whether to print out a "trace" as nnet is running.	TRUE
maxNWts	A numeric value specifying the maximum number of weights.	1000
abstol	A numeric value specifying absolute tolerance. (Fitting process halts if the fit criterion falls below abstol.)	1.0e-4
reltol	A numeric value specifying relative tolerance. (Fitting process halts if the algorithm can't reduce the error by reltol in each step.)	1.0e-8

There is no simple, closed-form solution for finding the optimal weights for a neural network model. So, the `nnet` function uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization method of the `optim` function to fit the model.

Let's try `nnet` on the San Francisco home sales data set. I had to play with the parameters a little bit to get a decent fit. I settled on 12 hidden units, linear outputs (which is appropriate for regression), skip connections, and a decay of 0.025:

```
> sf.price.model.nnet <- nnet(
+ price~bedrooms+squarefeet+lotsize+latitude+
+ longitude+neighborhood+month,
+ data=sanfrancisco.home.sales.training, size=12,
+ skip=TRUE, linout=TRUE, decay=0.025, na.action=na.omit)
# weights: 740
initial value 1387941951981143.500000
iter 10 value 292963198488371.437500
iter 20 value 235738652534232.968750
iter 30 value 215547308140618.656250
iter 40 value 212019186628667.375000
iter 50 value 210632523063203.562500
iter 60 value 208381505485842.656250
iter 70 value 207265136422489.750000
iter 80 value 207023188781434.906250
iter 90 value 206897724524820.937500
iter 100 value 206849625163830.156250
final value 206849625163830.156250
stopped after 100 iterations
```

To view the model, you can use the `print` or `summary` methods. Neither is particularly informative, though the `summary` method will show weights for all the units. Here is a small portion of the output for `summary` (the omitted portion is replaced with an ellipsis):

```
> summary(sf.price.model.nnet)
a 55-12-1 network with 740 weights
options were - skip-layer connections linear output units decay=0.025
  b->h1   i1->h1   i2->h1   i3->h1   i4->h1   i5->h1
  12.59    9.83   21398.35  29597.88  478.93  -1553.28
  i6->h1   i7->h1   i8->h1   i9->h1  i10->h1  i11->h1
  -0.15   -0.27    0.34   -0.05   -0.31    0.16
  ...
```

	Default
er to return the Hessian of fit.	FALSE
er to print out a "trace" as nnet is	TRUE
Maximum number of weights.	1000
ite tolerance. (Fitting process halts if >1.)	1.0e-4
ite tolerance. (Fitting process halts if or by reltol in each step.)	1.0e-8

er finding the optimal weights for a neural network using the Broyden-Fletcher-Goldfarb-Shanno algorithm to fit the model.

sales data set. I had to play with the parameters of the 12 hidden units, linear outputs, and connections, and a decay of 0.025:

```
+latitude+
nnet, size=12,
na.action=na.omit)
```

Summary methods. Neither is particularly informative. Here is the omitted portion is replaced with an

```
linear output units decay=0.025
>h1 i4->h1 i5->h1
.88 478.93 -1553.28
>h1 i10->h1 i11->h1
.05 -0.31 0.16
```

Here's how this model performed:

```
> calculate_rms_error(sf.price.model.nnet,
+ na.omit(sanfrancisco.home.sales.training),
+ na.omit(sanfrancisco.home.sales.testing),
+ "price")
train.err test.err
447567.2 566056.4
```

For more complex neural networks (such as networks with multiple hidden layers), see the packages AMORE, neural, and neuralnet.

Project Pursuit Regression

Projection pursuit regression is another very general model for representing non-linear relationships. Projection pursuit models have the form:

$$f(X) = \sum_{m=1}^M g_m(\omega_m^T X)$$

The functions g_m are called *ridge functions*. The project pursuit algorithm tries to optimize parameters for the parameters ω_m by trying to minimize the sum of the residuals. In equation form:

$$\min_{g_m, \omega_m} \left(\sum_{i=1}^N \left[y_i - \sum_{m=1}^M g_m(\omega_m^T x_i) \right]^2 \right)$$

Project pursuit regression is closely related to the neural network models that we saw above. (Note the similar form of the equations.) If we were to use the sigmoid function for the ridge functions g_m , projection pursuit would be identical to a neural network. In practice, projection pursuit regression is usually used with some type of smoothing method for the ridge functions. The default in R is to use Friedman's supersmoother function. (This function is actually pretty complicated and chooses the best of three relationships to pick the best smoothing function. See the help file for `supsmu` for more details. Note that this function finds the best smoother for the input data, not the smoother that leads to the best model.)

To use projection pursuit regression in R, use the function `ppr`:

```
## S3 method for class 'formula':
ppr(formula, data, weights, subset, na.action,
     contrasts = NULL, ..., model = FALSE)

## Default S3 method:
ppr(x, y, weights = rep(1,n),
     ww = rep(1,q), nterms, max.terms = nterms, optlevel = 2,
     sm.method = c("supsmu", "spline", "gcv spline"),
     bass = 0, span = 0, df = 5, gcvpen = 1, ...)
```

Arguments to `ppr` include the following.

Argument	Description	Default
<code>formula/data/subset/na.action, x/y</code>	Specifies the data to use for modeling, depending on the form of the function.	
<code>weights</code>	A vector of weights for each case.	
<code>contrasts</code>	A list specifying the contrasts to use for factors.	
<code>model</code>	A logical value indicating whether to return the model frame.	NULL
<code>ww</code>	A vector of weights for each response.	FALSE
<code>nterms</code>	Number of terms to include in the final model.	<code>rep(1, q)</code>
<code>max.terms</code>	Maximum number of terms to choose from when building the model.	<code>nterms</code>
<code>optlevel</code>	An integer value between 0 and 3, which determines how optimization is done. See the help file for more information.	2
<code>sm.method</code>	A character value specifying the method used for smoothing the ridge functions. Specify <code>sm.method="supsmu"</code> for Friedman's supersmoother, <code>sm.method="spline"</code> to use the code from <code>smooth.spline</code> , or <code>sm.method="gcv spline"</code> to choose the smoothing method with <code>gcv</code> .	"supsmu"
<code>bass</code>	When <code>sm.method="supsmu"</code> , a numeric value specifying the "bass" tone control for the supersmoother algorithm.	0
<code>span</code>	When <code>sm.method="supsmu"</code> , a numeric value specifying the "span" control for the supersmoother.	0
<code>df</code>	When <code>sm.method="spline"</code> , specifies the degrees of freedom for the spline function.	5
<code>gcvpen</code>	When <code>sm.method="gcv spline"</code> , a numeric value specifying the penalty for each degree of freedom.	1

Let's try projection pursuit regression on the home sales data:

```
> sf.price.model.ppr <- ppr(
+ price~bedrooms+squarefeet+lotsize+latitude+
+ longitude+neighborhood+month,
+ data=sanfrancisco.home.sales.training, nterms=20)
> sf.price.model.ppr
Call:
ppr(formula = price ~ bedrooms + squarefeet + lotsize + latitude +
longitude + neighborhood + month,
data = sanfrancisco.home.sales.training,
nterms = 20)
```

```
Goodness of fit:
20 terms
1.532615e+13
```

The summary function for `ppr` models prints out an enormous amount of information, including the function call, goodness-of-fit measurement, projection pursuit vectors,

and coefficients of ridge terms; I have omitted the output from the book to save space.

You can plot the ridge functions from a ppr model using the plot function. To plot them all at the same time, I used the graphical parameter mfcol=c(4, 4) to plot them on a 4 x 4 grid. (I also narrowed the margins to make them easier to read.)

```
par(mfcol=c(4,4), mar=c(2.5,2.5,1.5,1.5))
plot(sf.price.model.ppr)
```

The ridge functions are shown in Figure 20-10. I picked 12 explanatory variables, which seemed to do best on the validation data (though not on the training data):

```
> calculate_rms_error(sf.price.model.ppr,
+   na.omit(sanfrancisco.home.sales.training),
+   na.omit(sanfrancisco.home.sales.testing),
+   "price")
train.err test.err
194884.8  585613.9
```

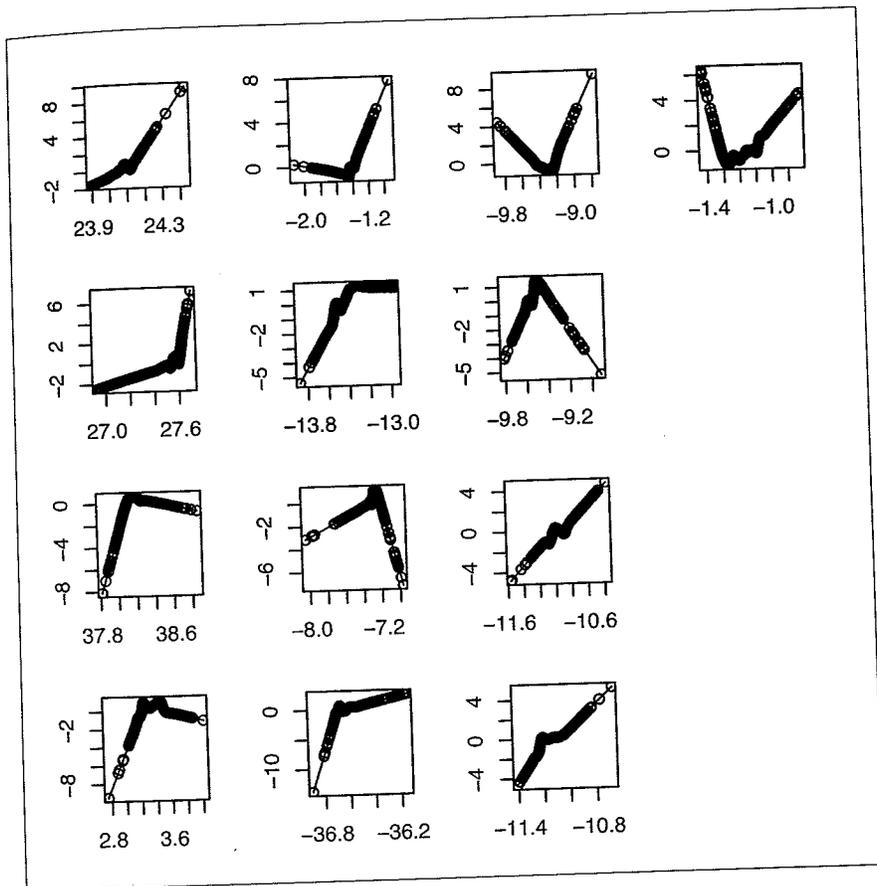


Figure 20-10. Ridge functions from projection pursuit model

Generalized Additive Models

Generalized additive models are another regression model technique for modeling complicated relationships in high-dimensionality data sets. Generalized additive models have the following form:

$$Y = \alpha + \sum_{j=1}^p f_j(X_j) + \varepsilon$$

Notice that each predictor variable x_j is first processed by a function f_j and is then used in a linear model. The generalized additive model algorithm finds the form of the functions f . These functions are often called *basis functions*.

The simplest way to fit generalized additive models in R is through the function `gam` in the library `gam`:

```
gam(formula, family = gaussian, data, weights, subset, na.action,
     start, etastart, mustart, control = gam.control(...),
     model=FALSE, method, x=FALSE, y=TRUE, ...)
```

This implementation is similar to the version from S and includes support for both local linear regression and smoothing spline basis functions. The `gam` package currently includes two different types of basis functions: smoothing splines and local regression. The `gam` function uses a back-fitting method to estimate parameters for the basis functions, and also estimates weights for the different terms in the model using penalized residual sum of squares.

When using the `gam` function to specify a model, you need to specify which type of basis function to use for which term. For example, suppose that you wanted to fit a model where the response variable was y , and the predictors were u , v , w , and x . To specify a model with smoothing functions for u and v , a local regression term for w , and an identity basis functions for x , you would specify the formula as $y \sim s(u) + s(v) + lo(w) + x$.

Here is a detailed description of the arguments to `gam`.

Argument	Description	Default
<code>formula</code>	A GAM formula specifying the form of the model. (See the help files for <code>s</code> and <code>lo</code> for more information on how to specify options for the basis functions.)	
<code>family</code>	A family object specifying the distribution and link function. See "Generalized Linear Models" on page 392 for a list of families.	<code>gaussian()</code>
<code>data</code>	A data frame containing the data to use for fitting.	<code>list</code>
<code>weights</code>	An (optional) numeric vector of weights for the input data.	<code>NULL</code>
<code>subset</code>	An optional vector specifying the subset of observations to use in fitting the model.	<code>NULL</code>
<code>na.action</code>	A function that indicates how to deal with missing values.	<code>options("na.action")</code> , which is <code>na.omit</code> by default

regression model technique for modeling dimensionality data sets. Generalized additive

first processed by a function f_j and is then additive model algorithm finds the form of called *basis functions*.

ive models in R is through the function

```
weights, subset, na.action,
control = gam.control(...),
y=TRUE, ...)
```

ion from S and includes support for both ne basis functions. The gam package cur- s functions: smoothing splines and local fitting method to estimate parameters for ghts for the different terms in the model

odel, you need to specify which type of ample, suppose that you wanted to fit a nd the predictors were $u, v, w,$ and x . To for u and v , a local regression term for w , ould specify the formula as $y \sim s(u) + s(v)$

nts to gam.

	Default
el. (See the help files specify options for the	
ink function. See list of families.	gaussian()
ng.	list
input data.	NULL
rvations to use in	NULL
ng values.	options("na.action"), which is na.omit by default

Argument	Description	Default
offset (through gam.fit)	A numeric value specifying an a priori known component to include in the additive predictor during fitting.	NULL
start	Starting values for the parameters in the additive predictors.	
etastart	Starting values for the additive predictors.	
mustart	Starting values for the vector of means.	
control	A list of parameters for controlling the fitting process. Use the function gam.control to generate a suitable list (and see the help file for that function to get the tuning parameters).	gam.control()
model	A logical value indicating whether the model frame should be included in the returned object.	FALSE
method	A character value specifying the method that should be used to fit the parametric part of the model. The only allowed values are method="glm.fit" (which uses iteratively reweighted least squares) or method="model.frame" (which does nothing except return the model frame).	NULL
x	A logical value specifying whether to return the X matrix (the predictors) with the model frame.	FALSE
y	A logical value specifying whether to return the Y vector (the response) with the model frame.	TRUE
...	Additional parameters passed to other methods (particularly, gam.fit).	

In R, there is an alternative implementation of generalized additive models available through the function gam in the package mgcv:

```
library(mgcv)
gam(formula, family=gaussian(), data=list(), weights=NULL, subset=NULL,
na.action, offset=NULL, method="GCV.Cp",
optimizer=c("outer", "newton"), control=gam.control(), scale=0,
select=FALSE, knots=NULL, sp=NULL, min.sp=NULL, H=NULL, gamma=1,
fit=TRUE, paraPen=NULL, G=NULL, in.out, ...)
```

This function allows a variety of different basis functions to be used: thin-plate regression splines (the default), cubic regression splines, and p-splines. The alternative gam function will estimate parameters for the basis functions as part of the fitting process using penalized likelihood maximization. The gam function in the mgcv package has many more options than the gam function in the gam package, but it is also a lot more complicated. See the help files in the mgcv package for more on the technical differences between the two packages.

Support Vector Machines

Support vector machines (SVMs) are a fairly recent algorithm for nonlinear models. They are a lot more difficult to explain to nonmathematicians than most statistical modeling algorithms. Explaining how SVMs work in detail is beyond the scope of this book, but here's a quick synopsis:

- SVMs don't rely on all of the underlying data to train the model. Only some observations (called the *support vectors*) are used. This makes SVMs somewhat resistant to outliers (like robust regression techniques) when used for regression. (It's also possible to use SVMs in the opposite way: to detect anomalies in the data.) You can control the range of values considered through the insensitive-loss function parameter `epsilon`.
- SVMs use a nonlinear transformation of the input data (like the basis functions in additive models or kernels in kernel methods). You can control the type of kernel used in SVMs through the parameter `kernel`.
- The final SVM model is fitted using a standard regression, with maximum likelihood estimates.

SVMs are black-box models; it's difficult to learn anything about a problem by looking at the parameters from a fitted SVM model. However, SVMs have become very popular, and many people have found that SVMs perform well in real-world situations. (An interesting side note is that SVMs are included as part of the Oracle Data Mining software, while many other algorithms are not.)

In R, SVMs are available in the library `e1071`,[#] through the function `svm`:

```
library(e1071)
## S3 method for class 'formula':
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Other implementations are available through the `ksvm` and `lssvm` functions in the `kernlab` library, `svmlight` in the `klaR` library, and `svmpath` in the `svmpath` library.

Let's try building an `svm` model for the home sales data:

```
> sf.price.model.svm <- svm(
+ price~bedrooms+squarefeet+lotsize+latitude+
+ longitude+neighborhood+month,
+ data=sanfrancisco.home.sales.training)
```

[#]Incidentally, this is, by far, the worst named package available on CRAN. It's named for a class given by the Department of Statistics, TU Wien. The package contains a number of very useful functions: SVM classifiers, algorithms for tuning other modeling functions, naive Bayes classifiers, and some other useful functions. It really should be called something like "veryusefulstatisticalfunctions."

g data to train the model. Only some are used. This makes SVMs somewhat on techniques) when used for regression the opposite way: to detect anomalies of values considered through the lon.

he input data (like the basis functions methods). You can control the type of er kernel.

andard regression, with maximum like-

learn anything about a problem by model. However, SVMs have become at SVMs perform well in real-world Ms are included as part of the Oracle ithms are not.)

through the function `svm`:

```
action =
, kernel =
r(x) 1 else 1 / ncol(x),
rance = 0.001, epsilon = 0.1,
FALSE, fitted = TRUE,
```

he `ksvm` and `lssvm` functions in the `lsvmpath` in the `svmpath` library.

es data:

ude+

ilable on CRAN. It's named for a class kage contains a number of very useful er modeling functions, naive Bayes ly should be called something like

Here is how the model performed:

```
> calculate_rms_error(sf.price.model.svm,
+   na.omit(sanfrancisco.home.sales.training),
+   na.omit(sanfrancisco.home.sales.testing),
+   "price")
train.err test.err
518647.9 641039.5
```

able in R:
le through the agnes function in the
ilable through the diana function in the
nly binary variables are used).
ne fanny function in the cluster package.
ough the batchSOM and SOM functions in



23

Time Series Analysis

Time series are a little different from other types of data. Time series data often has long-term trends or periodic patterns that traditional summary statistics don't capture. To find these patterns, you need to use different types of analyses. As an example of a time series, we will revisit the turkey price data that we first saw in "Time Series" on page 89.

Autocorrelation Functions

One important property of a time series is the autocorrelation function. You can estimate the autocorrelation function for time series using R's `acf` function:

```
acf(x, lag.max = NULL,  
    type = c("correlation", "covariance", "partial"),  
    plot = TRUE, na.action = na.fail, demean = TRUE, ...)
```

The function `pacf` is an alias for `acf`, except with the default type of "partial":

```
pacf(x, lag.max, plot, na.action, ...)
```

By default, this function plots the results. (An example plot is shown in "Plotting Time Series" on page 218.) As an example, let's show the autocorrelation function of the turkey price data:

```
> library(nutshell)  
> data(turkey.price.ts)  
> acf(turkey.price.ts, plot=FALSE)
```

Autocorrelations of series 'turkey.price.ts', by lag

```
0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500  
1.000 0.465 -0.019 -0.165 -0.145 -0.219 -0.215 -0.122 -0.136 -0.200  
0.8333 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833  
-0.016 0.368 0.723 0.403 -0.013 -0.187 -0.141 -0.180 -0.226 -0.130
```

```
> pacf(turkey.price.ts, plot=FALSE)
```

Partial autocorrelations of series 'turkey.price.ts', by lag

```
0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500 0.8333
0.465 -0.300 -0.020 -0.060 -0.218 -0.054 -0.061 -0.211 -0.180 0.098
0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833
0.299 0.571 -0.122 -0.077 -0.075 0.119 0.064 -0.149 -0.061
```

The function `ccf` plots the cross-correlation function for two time series:

```
ccf(x, y, lag.max = NULL, type = c("correlation", "covariance"),
    plot = TRUE, na.action = na.fail, ...)
```

By default, this function will plot the results. You can suppress the plot (to just view the function) with the argument `plot=FALSE`.

As an example of cross-correlations, we can use average ham prices in the United States. These are included in the `nutshell` package as `ham.price.ts`:

```
> library(nutshell)
> data(ham.price.ts)
> ccf(turkey.price.ts, ham.price.ts, plot=FALSE)
```

Autocorrelations of series 'X', by lag

```
-1.0833 -1.0000 -0.9167 -0.8333 -0.7500 -0.6667 -0.5833 -0.5000 -0.4167
0.147 0.168 -0.188 -0.259 -0.234 -0.098 -0.004 0.010 0.231
-0.3333 -0.2500 -0.1667 -0.0833 0.0000 0.0833 0.1667 0.2500 0.3333
0.228 0.059 -0.038 0.379 0.124 -0.207 -0.315 -0.160 -0.084
0.4167 0.5000 0.5833 0.6667 0.7500 0.8333 0.9167 1.0000 1.0833
-0.047 -0.005 0.229 0.223 -0.056 -0.099 0.189 0.039 -0.108
```

You can apply filters to a time series with the `filter` function or convolutions (using fast Fourier transforms [FFTs]) with the `convolve` function.

Time Series Models

Time series models are a little different from other models that we've seen in R. With most other models, the goal is to predict a value (the response variable) from a set of other variables (the predictor variables). Usually, we explicitly assume that there is no autocorrelation: that the sequence of observations does not matter.

With time series, we assume the opposite: we assume that previous observations help predict future observations (see Figure 23-1).

To fit an autoregressive model to a time series, use the function `ar`:

```
ar(x, aic = TRUE, order.max = NULL,
    method=c("yule-walker", "burg", "ols", "mle", "yw"),
    na.action, series, ...)
```

```

s 'turkey.price.ts', by lag
7 0.5000 0.5833 0.6667 0.7500 0.8333
3 -0.054 -0.061 -0.211 -0.180 0.098
) 1.3333 1.4167 1.5000 1.5833
) 0.119 0.064 -0.149 -0.061

```

tion function for two time series:
("correlation", "covariance"),
il, ...)

ults. You can suppress the plot (to just view
LSE.

can use average ham prices in the United
1 package as ham.price.ts:

```

, plot=FALSE)

```

```

lag
500 -0.6667 -0.5833 -0.5000 -0.4167
234 -0.098 -0.004 0.010 0.231
000 0.0833 0.1667 0.2500 0.3333
124 -0.207 -0.315 -0.160 -0.084
500 0.8333 0.9167 1.0000 1.0833
56 -0.099 0.189 0.039 -0.108

```

the filter function or convolutions (using
involve function.

other models that we've seen in R. With
value (the response variable) from a set
Usually, we explicitly assume that there
observations does not matter.

we assume that previous observations
23-1).

es, use the function ar:

", "mle", "yw"),

MY HOBBY: EXTRAPOLATING

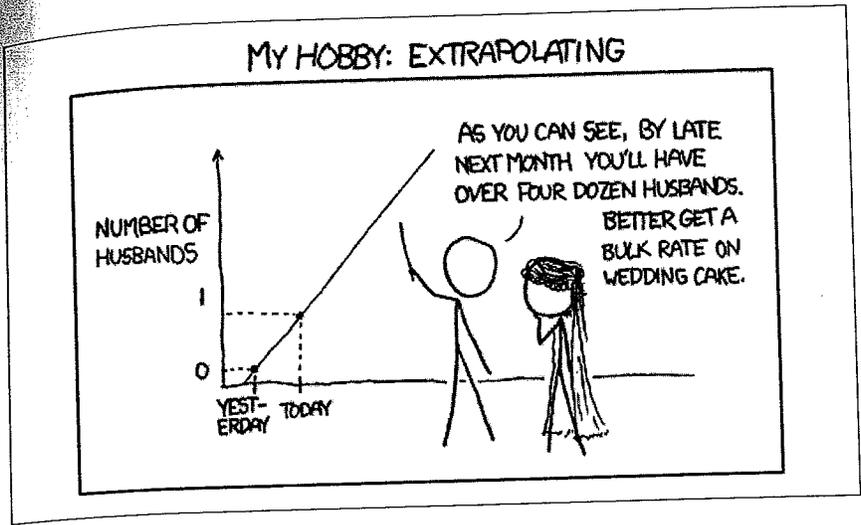


Figure 23-1. Extrapolating times series

Here is a description of the arguments to ar.

Argument	Description	
x	A time series.	
aic	A logical value that specifies whether the Akaike information criterion is used to choose the order of the model.	TRUE
order.max	A numeric value specifying the maximum order of the model to fit.	NULL
method	A character value that specifies the method to use for fitting the model. Specify method="yw" (or method="yule-walker") for the Yule-Walker method, method="burg" for the Burg method, method="ols" for ordinary least squares, or method="mle" for maximum likelihood estimation.	c("yule-walker", "burg", "ols", "mle", "yw")
na.action	A function that specifies how to handle missing values.	
series	A character vector of names for the series.	
demean	A logical value specifying if a mean should be estimated during fitting.	
var.method	Specifies the method used to estimate the innovations variance when method="ar.burg".	
...	Additional arguments, depending on method.	

The ar function actually calls one of four other functions, depending on the fit method chosen: ar.yw, ar.burg, ar.ols, or ar.mle. As an example, let's fit an autoregressive model to the turkey price data:

```

> library(nutshell)
> data(turkey.price.ts)
> turkey.price.ts.ar <- ar(turkey.price.ts)
> turkey.price.ts.ar

```

```

Call:
ar(x = turkey.price.ts)

```

Coefficients:

1	2	3	4	5	6	7
0.3353	-0.1868	-0.0024	0.0571	-0.1554	-0.0208	0.0914
8	9	10	11	12		
-0.0658	-0.0952	0.0649	0.0099	0.5714		

Order selected 12 σ^2 estimated as 0.05182

You can use the model to predict future values. To do this, use the `predict` function. Here is the method for `ar` objects:

```
predict(object, newdata, n.ahead = 1, se.fit = TRUE, ...)
```

The argument `object` specifies the model object to use for prediction. You can use `newdata` to specify new data to use for prediction, or `n.ahead` to specify a number of periods ahead to predict. The argument `se.fit` specifies whether to return standard errors of the prediction error.

Here is a forecast for the next 12 months for turkey prices:

```
> predict(turkey.price.ts.ar,n.ahead=12)
$pred
      Jan      Feb      Mar      Apr      May      Jun
2008                1.8827277 1.7209182
2009 1.5439290 1.6971933 1.5849406 1.7800358
      Jul      Aug      Sep      Oct      Nov      Dec
2008 1.7715016 1.9416776 1.7791961 1.4822070 0.9894343 1.1588863
2009
$se
      Jan      Feb      Mar      Apr      May      Jun
2008                0.2276439 0.2400967
2009 0.2450732 0.2470678 0.2470864 0.2480176
      Jul      Aug      Sep      Oct      Nov      Dec
2008 0.2406938 0.2415644 0.2417360 0.2429339 0.2444610 0.2449850
2009
```

To take a look at a forecast from an autoregressive model, you can use the function `ts.plot`. This function plots multiple time series on a single chart, even if the times are not overlapping. You can specify colors, line types, or other characteristics of each series as vectors; the i th place in the vector determines the property for the i th series.

Here is how to plot the turkey price time series as a solid line, and a projection 24 months into the future as a dashed line:

```
ts.plot(turkey.price.ts,
        predict(turkey.price.ts.ar,n.ahead=24)$pred,
        lty=c(1:2))
```

The plot is shown in Figure 23-2. You can also fit autoregressive integrated moving average (ARIMA) models in R using the `arima` function:

```
arima(x, order = c(0, 0, 0),
      seasonal = list(order = c(0, 0, 0), period = NA),
      xreg = NULL, include.mean = TRUE,
```

```

5          6          7
-0.1554 -0.0208  0.0914
12
0.5714

```

d as 0.05182
 values. To do this, use the predict function.

l, se.fit = TRUE, ...)
 object to use for prediction. You can use
 diction, or n.ahead to specify a number of
 a. fit specifies whether to return standard

for turkey prices:
 =12)

```

Apr      May      Jun
1.8827277 1.7209182
7800358
Oct      Nov      Dec
4822070 0.9894343 1.1588863

```

```

Apr      May      Jun
0.2276439 0.2400967
2480176
Oct      Nov      Dec
2429339 0.2444610 0.2449850

```

ressive model, you can use the function
 series on a single chart, even if the times
 s, line types, or other characteristics of
 ctor determines the property for the *i*th
 ries as a solid line, and a projection 24

lead=24)\$pred,

so fit autoregressive integrated moving
 na function:

), period = NA),

```

transform.pars = TRUE,
fixed = NULL, init = NULL,
method = c("CSS-ML", "ML", "CSS"),
n.cond, optim.method = "BFGS",
optim.control = list(), kappa = 1e6)

```

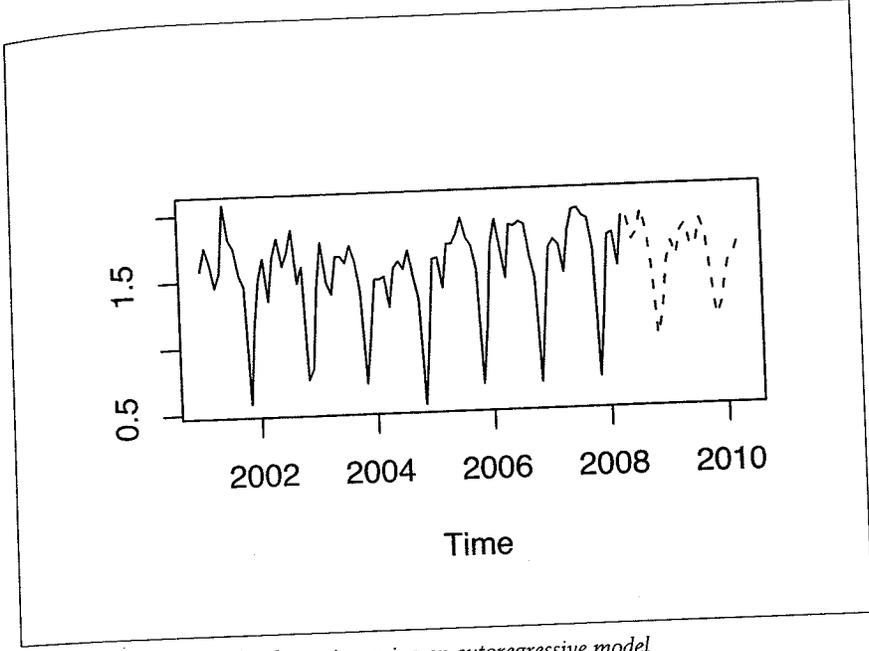


Figure 23-2. Forecast of turkey prices using an autoregressive model

Here is a description of the arguments to arima.

Argument	Description	Default
x	A time series.	
order	A numeric vector (p, d, q), where p is the AR order, d is the degree of differencing, and q is the MA order.	c(0, 0, 0)
seasonal	A list specifying the seasonal part of the model. The list contains two parts: the order and the period.	list(order = c(0, 0, 0), period = NA)
xreg	An (optional) vector or matrix of external regressors (with the same number of rows as x).	NULL
include.mean	A logical value specifying whether the model should include a mean/intercept term.	TRUE
transform.pars	A logical value specifying whether the AR parameters should be transformed to ensure that they remain in the region of stationarity.	TRUE
fixed	An optional numeric vector specifying fixed values for parameters. (Only NA values are varied.)	NULL
init	A numeric vector of initial parameter values.	NULL

Argument	Description	Default
method	A character value specifying the fitting method to use. The default setting, <code>method="CSS-ML"</code> , uses conditional sum of squares to find starting values, then maximum likelihood. Specify <code>method="ML"</code> for maximum likelihood only, or <code>method="CSS"</code> for conditional sum of squares only.	<code>c("CSS-ML", "ML", "CSS")</code>
n.cond	A numeric value indicating the number of initial values to ignore (only used for conditional sum of squares).	
optim.method	A character value that is passed to <code>optim</code> as <code>method</code> .	<code>"BFGS"</code>
optim.control	A list of values that is passed to <code>optim</code> as <code>control</code> .	<code>list()</code>
kappa	The prior variance for the past observations in a differenced model. See the help file for more information.	<code>1e-6</code>

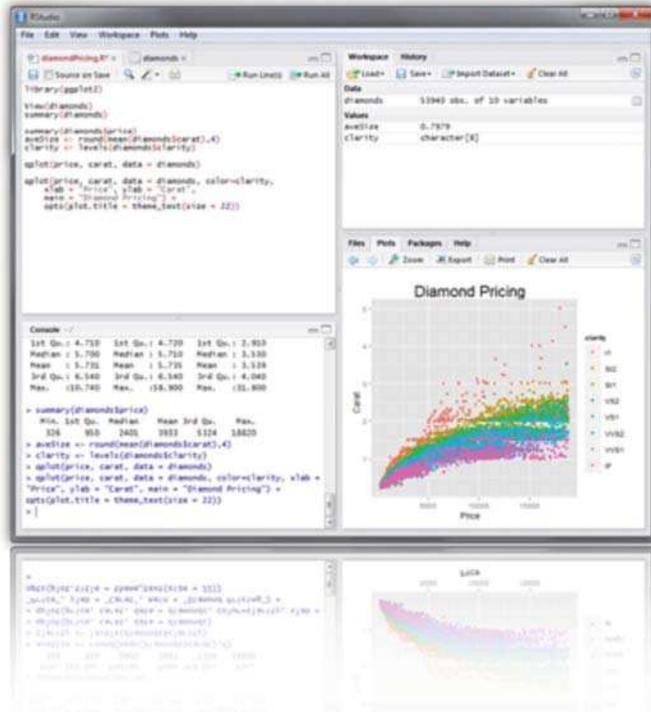
The `arima` function uses the `optim` function to fit models. You can use the result of an ARIMA model to smooth a time series with the `tsSmooth` function. For more information, see the help file for `tsSmooth`.

Exhibit D



Introducing RStudio

RStudio™ is a new integrated development environment (IDE) for R. RStudio combines an intuitive user interface with powerful coding tools to help you get the most out of R.



Productive

RStudio brings together everything you need to be productive with R in a single, customizable environment. Its intuitive interface and [powerful coding tools](#) help you get work done faster.

Runs Everywhere

RStudio is available for [all major platforms](#) including Windows, Mac OS X, and Linux. It can even run alongside R on a server, enabling multiple users to access the RStudio IDE using a web browser.

Free & Open

Like R, RStudio is available under a [free software license](#) that guarantees the freedom to share and change the software, and to make sure it remains free software for all its users.



News

RStudio v0.93 Available (4/11/2011)

RStudio v0.93 is now available. This release includes source editor enhancements, options for customizing pane layout and appearance, an interactive plotting package, improved handling of Unicode characters, and many more small enhancements and bug fixes. All of the details on the new release can be found in our [release notes](#)

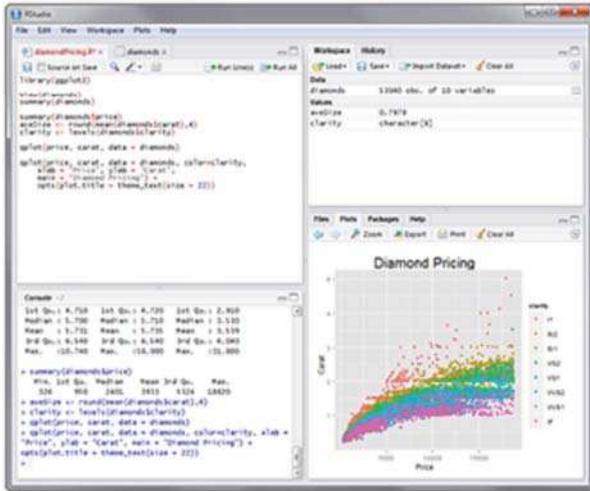
Announcing RStudio (2/28/2011)

We are pleased to announce availability of the beta version of RStudio! RStudio is a new open-source IDE for R that runs either on your desktop or on a server (where it is accessed using a browser). We invite everyone to check out the [screenshots](#) for more details; [download](#) the product to try it out, and follow its ongoing [development](#) on github.

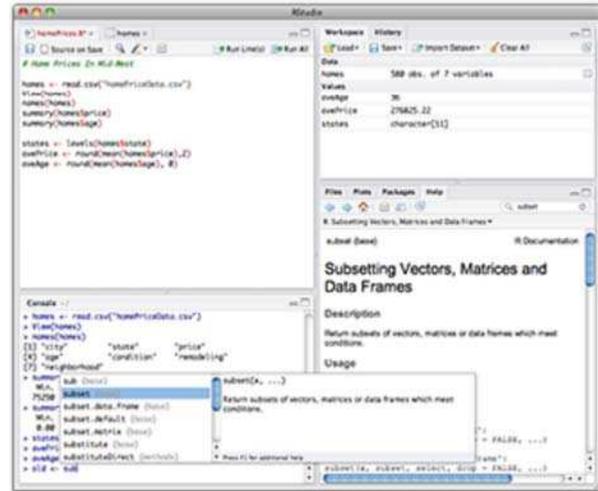


RStudio Screenshots

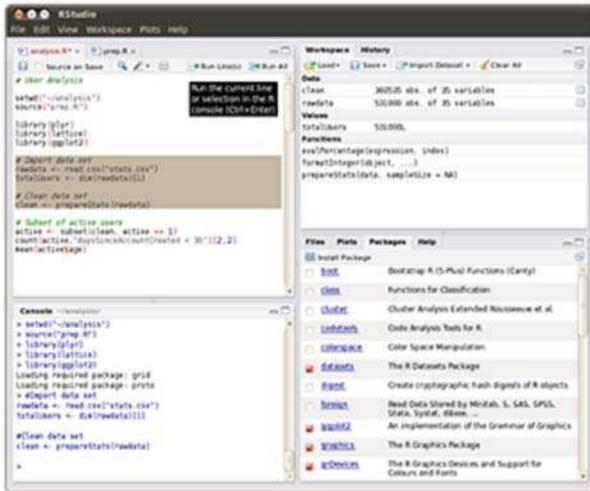
RStudio runs on all major platforms, and can even be run on a server and accessed remotely using a web browser:



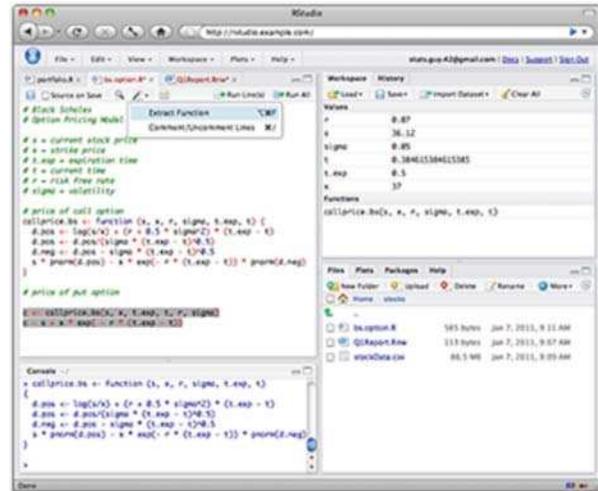
Windows



Mac OS X

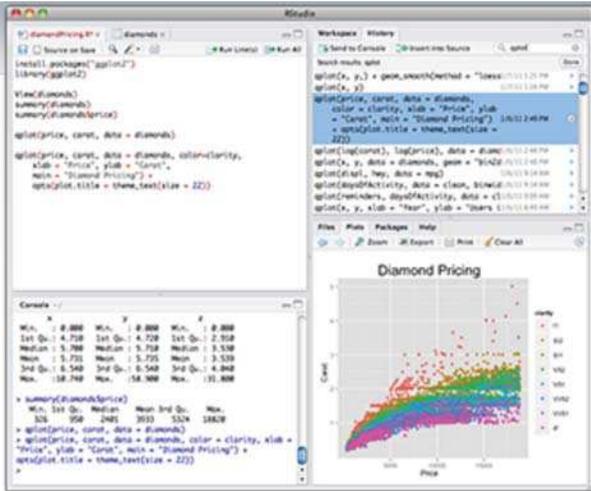


Ubuntu

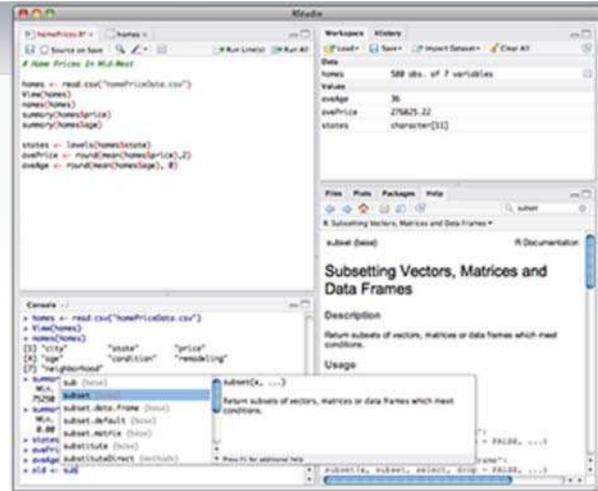


Over the Web

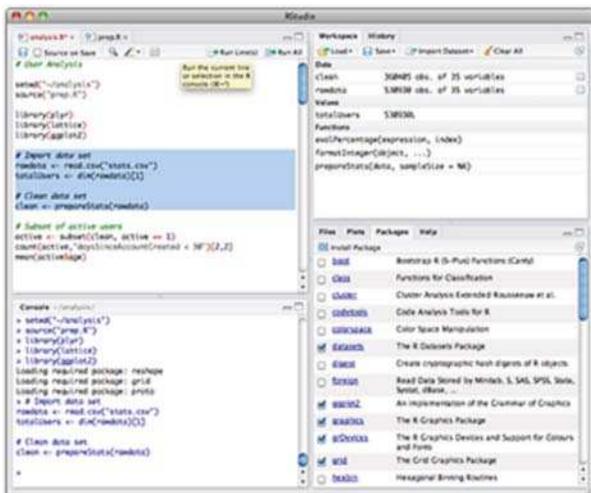
RStudio integrates all of the tools you use while working with R into a single customizable environment:



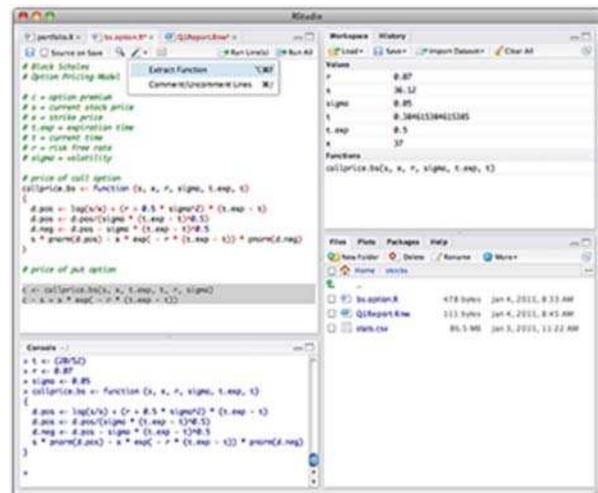
Searchable History



Code Completion

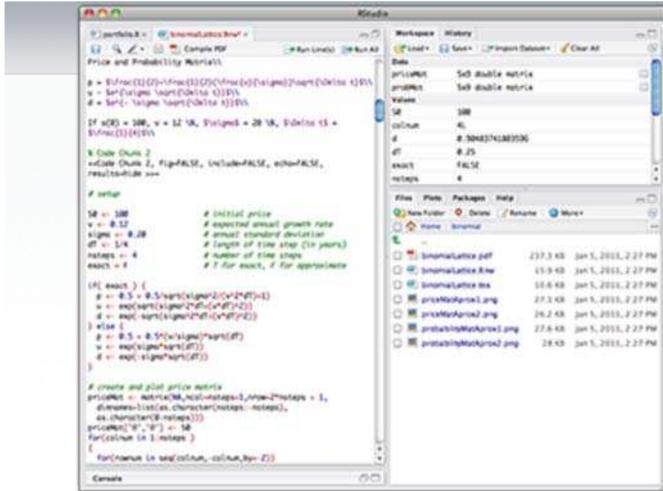


Execute From Source

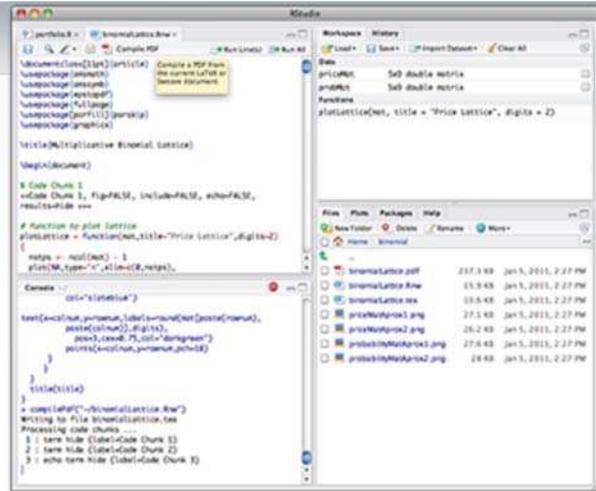


Code Transformations

RStudio supports authoring TeX and Sweave documents:



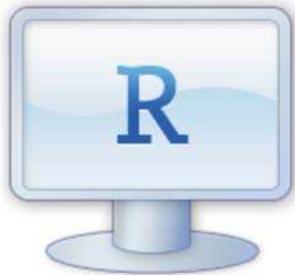
Syntax Highlighting Editor



One-Click PDF



Download RStudio v0.93



If you run R on your desktop:



[Download RStudio Desktop](#)

OR



If you run R on a Linux server and want to enable users to remotely access RStudio using a web browser:



[Download RStudio Server](#)

RStudio Documentation

Using RStudio

-  [Working in the Console](#)
-  [Editing and Executing Code](#)
-  [Using Command History](#)
-  [Working Directories and Workspaces](#)
-  [Customizing RStudio](#)
-  [Keyboard Shortcuts](#)

RStudio Server

-  [Getting Started](#)
-  [Configuring the Server](#)
-  [Managing the Server](#)
-  [Running with a Proxy](#)

Advanced Topics

-  [Interactive Plotting with Manipulate](#)
-  [Using Different Versions of R](#)
-  [Character Encoding](#)
-  [Optimizing your Browser for RStudio](#)
-  [Uploading and Downloading Files](#)

More

-  [About RStudio](#)
-  [Release Notes](#)
-  [Frequently Asked Questions](#)
-  [Getting Help with R](#)

© 2011 RStudio, Inc.



Working in the Console

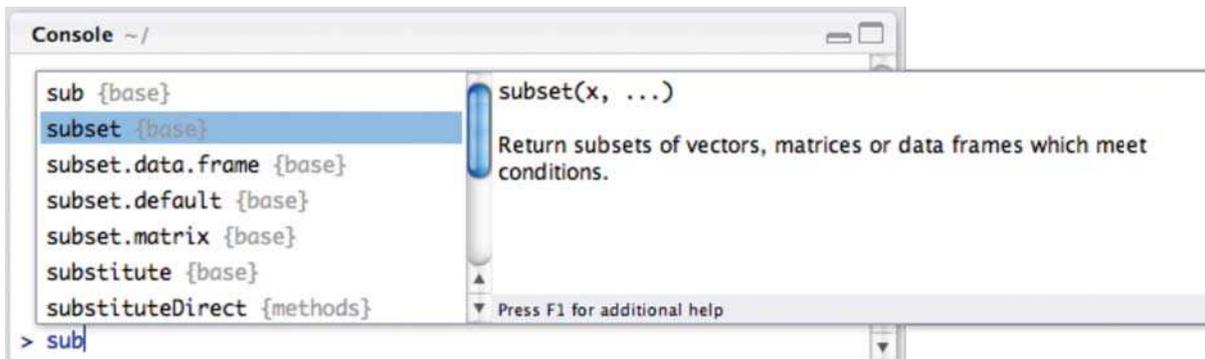
Overview

The RStudio console includes a variety of features intended to make working with R more productive and straightforward. This article reviews these features. Learning to use these features along with the related features available in the [Source](#) and [History](#) panes can have a substantial payoff in your overall productivity with R.

Code Completion

RStudio supports the automatic completion of code using the **Tab** key. For example, if you have an object named `pollResults` in your workspace you can type `poll` and then **Tab** and RStudio will automatically complete the full name of the object.

The code completion feature also provides inline help for functions whenever possible. For example, if you typed `sub` then pressed **Tab** you would see:



Code completion also works for function arguments, so if you typed `subset(` and then pressed **Tab** you'd see the following:

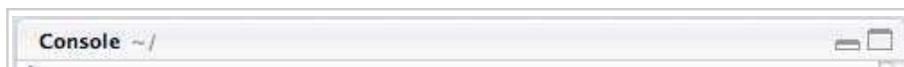


Retrieving Previous Commands

As you work with R you'll often want to re-execute a command which you previously entered. As with the standard R console, the RStudio console supports the ability to recall previous commands using the arrow keys:

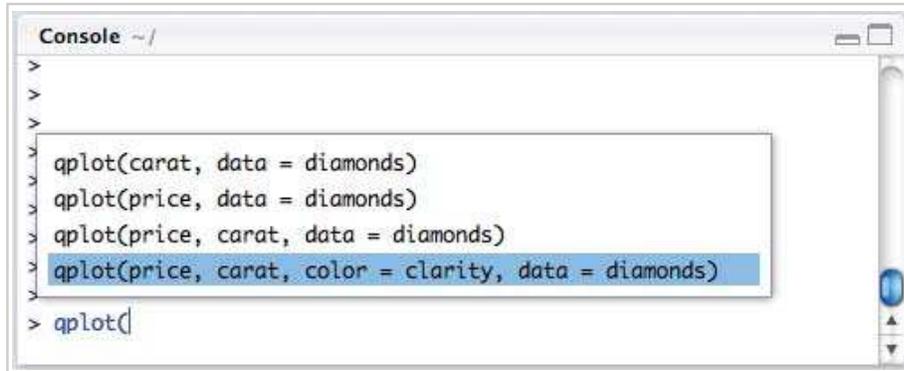
- **Up** — Recall previous command(s)
- **Down** — Reverse of Up

If you wish to review a list of your recent commands and then select a command from this list you can use **Ctrl+Up** to review the list (note that on the Mac you can also use **Command+Up**):





You can also use this same keyboard shortcut to quickly search for commands that match a given prefix. For example, to search for previous instances of the `plot` function simply type `plot` and then **Ctrl+Up**:



Console Title Bar

This screenshot illustrates a few additional capabilities provided by the Console title bar:

- Display of the current working directory.
- The ability to interrupt R during a long computation.
- Minimizing and maximizing the Console in relation to the Source pane (using the buttons at the top-right or by double-clicking the title bar).



Keyboard Shortcuts

Beyond the history and code-completion oriented keyboard shortcuts described above, there are a wide variety of other shortcuts available. Some of the more useful shortcuts include:

- **Ctrl+1** — Move focus to the Source Editor
- **Ctrl+2** — Move focus to the Console
- **Ctrl+L** — Clear the Console
- **Esc** — Interrupt R

You can find a list of all shortcuts in the [Keyboard Shortcuts](#) article.

Related Topics

- [Editing and Executing Code](#)
- [Using Command History](#)



Editing and Executing Code

Overview

RStudio's source editor includes a variety of productivity enhancing features including syntax highlighting, code completion, multiple-file editing, and find/replace.

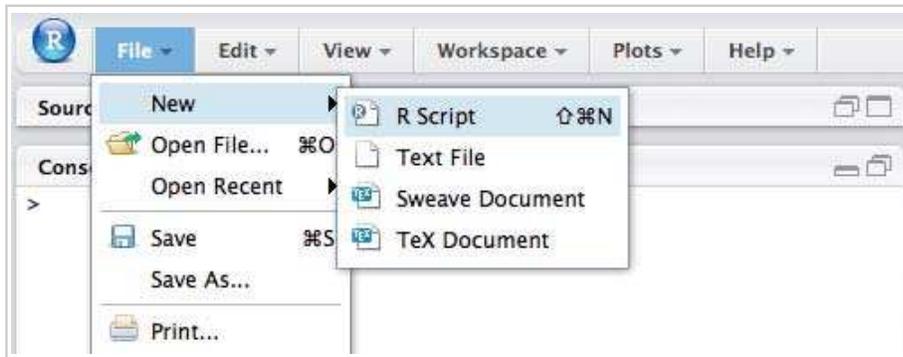
RStudio also enables you to flexibly execute R code directly from the source editor. For many R developers this represents their preferred way of working with R. By executing commands from within the source editor rather than the console it is much easier to reproduce sequences of commands as well as package them for re-use as a function. These features are described in the [Executing Code](#) section below.

Managing Files

RStudio supports syntax highlighting and other specialized code-editing features for the following types of files:

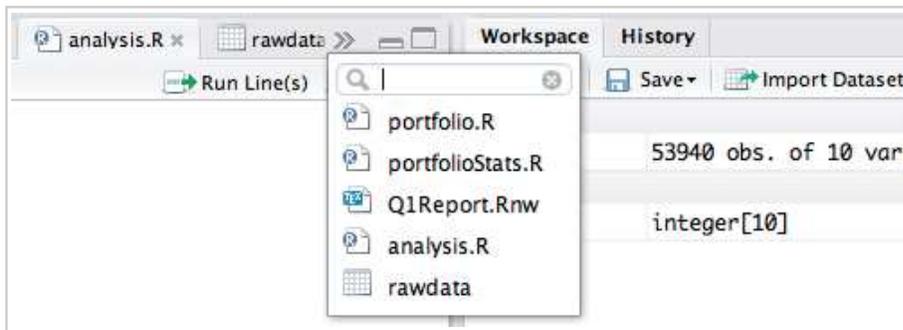
- R scripts
- Sweave documents
- TeX documents

To create a new file you use the **File -> New** menu:



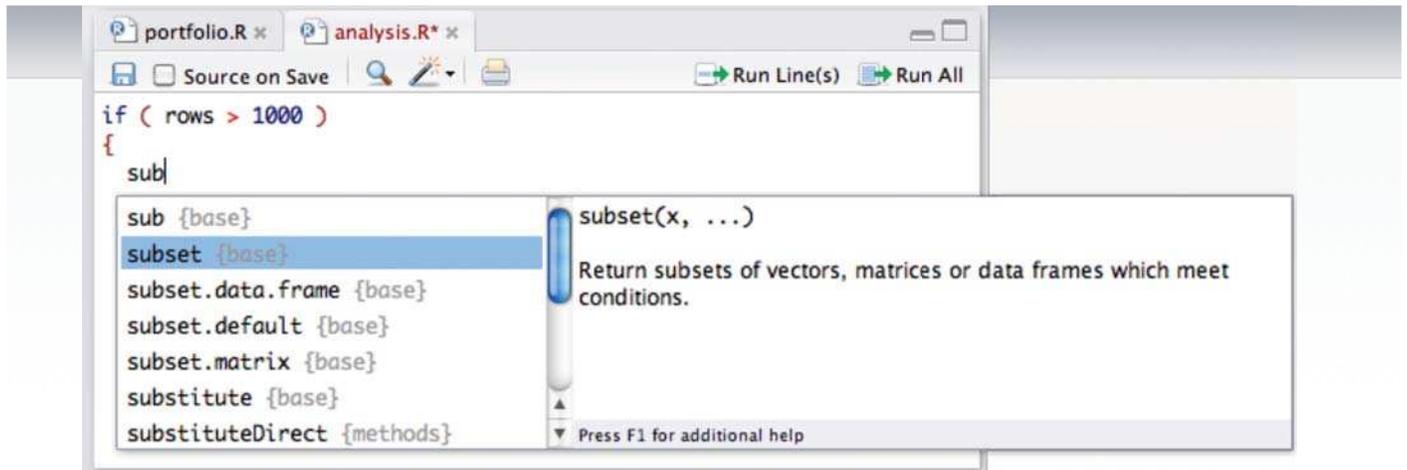
To open an existing file you use either the **File -> Open** menu or the **Open Recent** menu to select from recently opened files.

If you open several files within RStudio they are all available as tabs to facilitate quick switching between open documents. If you have a large number of open documents you can also navigate between them using the >> icon on the tab bar or the **View -> Switch to Tab** menu item:



Code Completion

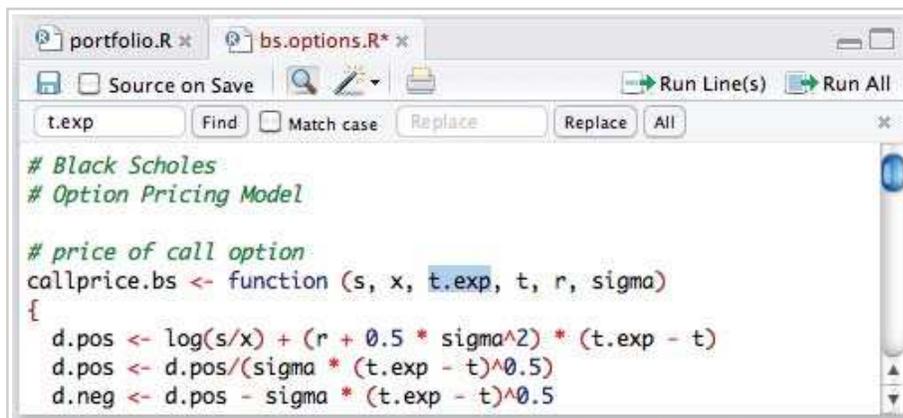
RStudio supports the automatic completion of code using the **Tab** key. For example, if you have an object named `pollResults` in your workspace you can type `poll` and then **Tab** and RStudio will automatically complete the full name of the object.



Code completion also works in the console, and more details on using it can be found the console [Code Completion](#) documentation.

Find and Replace

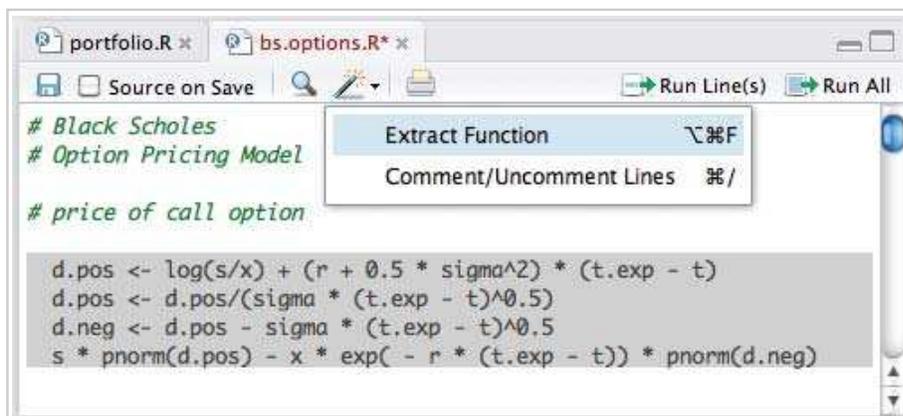
RStudio supports finding and replacing text within source documents:



Find and replace can be opened using the **Ctrl+F** shortcut key, or from the **Edit -> Find and Replace** menu item.

Extract Function

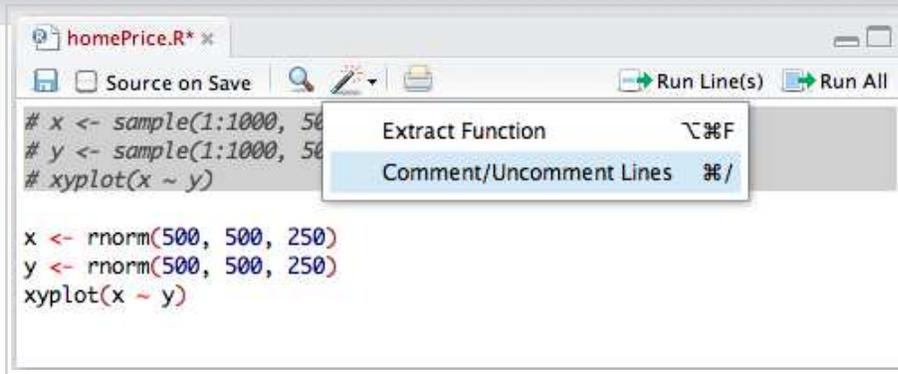
RStudio can analyze a selection of code from within the source editor and automatically convert it into a re-usable function. Any "free" variables within the selection (objects that are referenced but not created within the selection) are converted into function arguments:



Comment/Uncomment

You can comment and uncomment entire selections of code using the **Edit -> Comment/Uncomment Lines** menu item

(you can also do this using the **Ctrl+/**** keyboard shortcut):

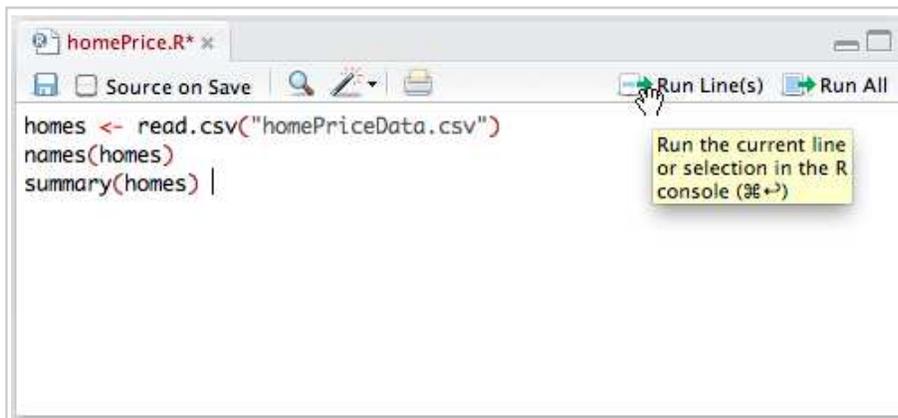


Executing Code

RStudio supports the direct execution of code from within the source editor (the executed commands are inserted into the console where their output also appears).

Executing a Single Line

To execute the line of source code where the cursor currently resides you press the **Ctrl+Enter** key (or use the **Run Line(s)** toolbar button):

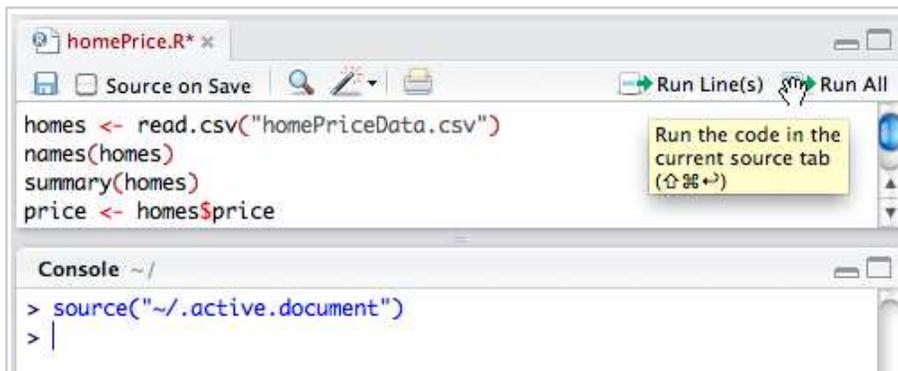


After executing the line of code, RStudio automatically advances the cursor to the next line. This enables you to single-step through a sequence of lines.

Executing Multiple Lines

There are two ways to execute multiple lines from within the editor:

- Select the lines and press the **Ctrl+Enter** key (or use the **Run Line(s)** toolbar button); or
- To run the entire document press the **Ctrl+Shift+Enter** key (or use the **Run All** toolbar button).

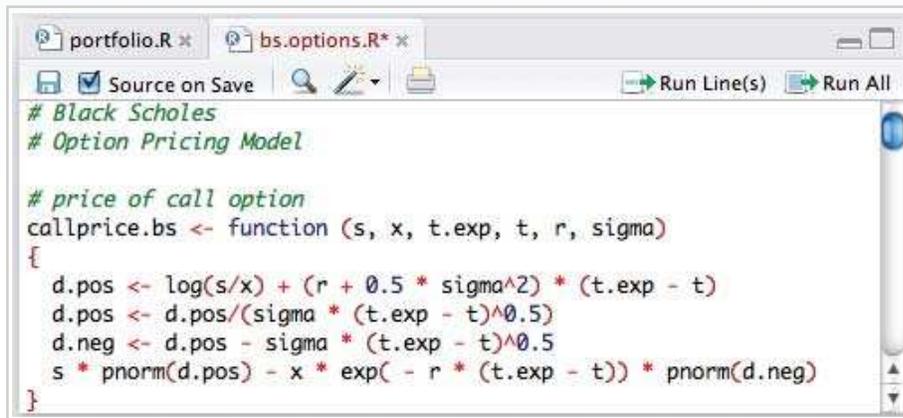


The difference between running lines from a selection and invoking **Run All** is that when running a selection all lines are inserted directly into the console whereas for **Run All** the file is saved to a temporary location and then sourced into

the console from there (thereby creating less clutter in the console).

Source on Save

When editing re-usable functions (as opposed to freestanding lines of R) you may wish to set the **Source on Save** option for the document (available on the toolbar next to the Save icon). Enabling this option will cause the file to automatically be sourced into the global environment every time it is saved:



```
portfolio.R * bs.options.R *
Source on Save Run Line(s) Run All
# Black Scholes
# Option Pricing Model

# price of call option
callprice.bs <- function (s, x, t.exp, t, r, sigma)
{
  d.pos <- log(s/x) + (r + 0.5 * sigma^2) * (t.exp - t)
  d.pos <- d.pos / (sigma * (t.exp - t)^0.5)
  d.neg <- d.pos - sigma * (t.exp - t)^0.5
  s * pnorm(d.pos) - x * exp(- r * (t.exp - t)) * pnorm(d.neg)
}
```

Setting **Source on Save** ensures that the copy of a function within the global environment is always in sync with its source, and also provides a good way to arrange for frequent syntax checking as you develop a function.

Keyboard Shortcuts

Beyond the keyboard shortcuts described above, there are a wide variety of other shortcuts available. Some of the more useful ones include:

- **Ctrl+Shift+N** — New document
- **Ctrl+O** — Open document
- **Ctrl+S** — Save active document
- **Ctrl+I** — Move focus to the Source Editor
- **Ctrl+2** — Move focus to the Console

You can find a list of all shortcuts in the [Keyboard Shortcuts](#) article.

Related Topics

- [Working in the Console](#)
- [Using Command History](#)

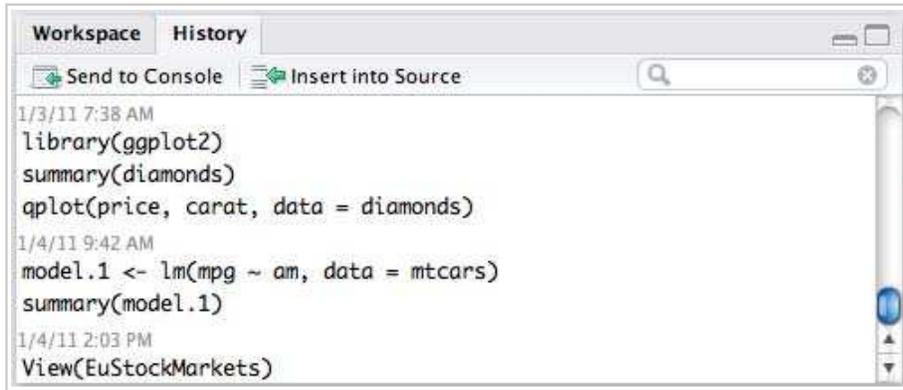


Using Command History

RStudio maintains a database of all commands which you have ever entered into the Console. You can browse and search this database using the History pane.

Browsing History

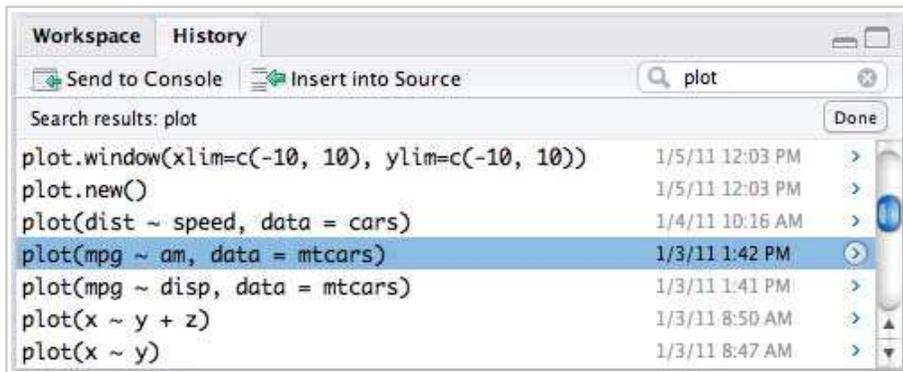
Commands you have previously entered in the RStudio console can be browsed from the History tab. The commands are displayed in order (most recent at the bottom) and grouped by block of time:



Searching History

Executing a Search

You can use the search box at the top right of the history tab to search for all instances of a previous command (e.g. plot). The search can be further refined by adding additional words separated by spaces (e.g. the name of particular dataset):



Showing Command Context

After searching for a command within your history you may wish to view the other commands that were executed in proximity to it. By clicking the arrow in the right margin of the search results you can view the command within its context:



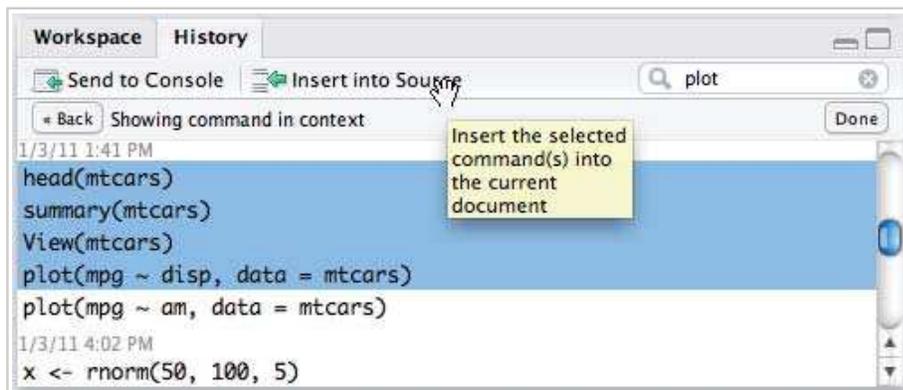
```
view(mtcars)
plot(mpg ~ disp, data = mtcars)
plot(mpg ~ am, data = mtcars)
1/3/11 4:02 PM
x <- rnorm(50, 100, 5)
```

Using Commands

Commands selected within the History pane can be used in two fashions (corresponding to the two buttons on the left side of the History toolbar):

- **Send to Console**— Sends the selected command(s) to the Console. Note that the commands are inserted into the Console however they are not executed until you press **Enter**.
- **Insert into Source**— Inserts the selected command(s) into the currently active Source document. If there isn't currently a Source document available then a new untitled one will be created.

Within the history list you can select a single command or multiple commands:



Related Topics

- Working in the Console
- Editing and Executing Code



Working Directories and Workspaces

The default behavior of R for the handling of .RData files and workspaces encourages and facilitates a model of breaking work contexts into distinct working directories. This article describes the various features of RStudio which support this workflow.

Default Working Directory

As with the standard R GUI, RStudio employs the notion of a global default working directory. Normally this is the user home directory (typically referenced using ~ in R). When RStudio starts up it does the following:

- Executes the .Rprofile (if any) from the default working directory.
- Loads the .RData file (if any) from the default working directory into the workspace.
- Performs the other actions described in [R Startup](#).

When RStudio exits and there are changes to the workspace, a prompt asks whether these changes should be saved to the .RData file in the current working directory.

This default behavior can be customized in the following ways using the [RStudio Options](#) dialog:

- Change the default working directory
- Enable/disable the loading of .RData from the default working directory at startup
- Specify whether .RData is always saved, never saved, or prompted for save at exit.

Changing the Working Directory

The current working directory is displayed by RStudio within the title region of the Console. There are a number of ways to change the current working directory:

- Use the `setwd` R function
- Use the **Tools | Change Working Dir...** menu. This will also change directory location of the Files pane.
- From within the Files pane, use the **More | Set As Working Directory** menu. (Navigation within the Files pane alone will not change the working directory.)

Be careful to consider the side effects of changing your working directory:

- Relative file references in your code (for datasets, source files, etc) will become invalid when you change working directories.
- The location where .RData is saved at exit will be changed to the new directory.

Because these side effects can cause confusion and errors, it's usually best to start within the working directory associated with your project and remain there for the duration of your session. The section below describes how to set RStudio's initial working directory.

Starting in Other Working Directories

If all of the files related to a project are contained within a single directory then you'll likely want to start RStudio within that directory. There are a number of ways (which vary by platform) to do this.

File Associations

On all platforms RStudio registers itself as a handler for .RData, .R, and other R related file types. This means that the system file browser's context-menu will show RStudio as an **Open With** choice for these files.

You can also optionally create a default association between RStudio and the .RData and/or .R file types.

When launched through a file association, RStudio automatically sets the working directory to the directory of the opened file. Note that RStudio can also open files via associations when it is already running—in this case RStudio

simply opens the file and does not change the working directory.

Shortcuts (Windows)

On Windows, you can create a shortcut to RStudio and customize the "Start in" field. When launched through this shortcut RStudio will startup within the specified working directory.

Drag and Drop (Mac)

On Mac, dragging and dropping a folder from the Finder on the RStudio Dock icon will cause RStudio to startup with the dropped folder as the current working directory.

Run from Terminal (Mac and Linux)

On Mac and Linux systems you can run RStudio from a terminal and specify which working directory to startup within. Additionally, on Linux systems if you run RStudio from a terminal and specify no command line argument then RStudio will startup using the current working directory of the terminal.

For example, on the Mac you could use the following commands to open RStudio (respectively) in the '~/projects/foo' directory or the current working directory:

```
$ open -a RStudio ~/projects/foo
$ open -a RStudio .
```

On Linux you would use the following commands (note that no '.' is necessary in the second invocation):

```
$ rstudio ~/projects/foo
$ rstudio
```

Handling of .Rprofile

When starting RStudio in an alternate working directory the .Rprofile file located within that directory is sourced. If (and only if) there is not an .Rprofile file in the alternate directory then the global default profile (e.g. ~/.Rprofile) is sourced instead.

Loading and Saving Workspaces

If you want to save or load a workspace during an RStudio session you can use the following commands to save to or load from the .RData file in the current working directory:

```
> save.image()
> load(".RData")
```

Note that the load function appends (and overwrites) objects within the current workspace rather than replacing it entirely. Prior to loading you may therefore wish to clear all objects currently within the workspace. You can do so using the following command:

```
> rm(list=ls())
```

Note that since loading is handled at startup and saving is handled at exit, in many cases you won't require these commands. If however you change working directories within a session you may need them in order to sync your workspace with the directory you have changed to.

The RStudio **Workspace** menu also includes items that execute the above described commands, as well as enables you to load or save specific .RData files.

Handling of .Rhistory

The .Rhistory file determines which commands are available by pressing the up arrow key within the console. RStudio handles the .Rhistory file differently than the standard R GUI. This is because in addition to the .Rhistory file RStudio also includes a searchable history database (accessible via the History tab). For the sake of simplicity RStudio attempts to keep these two history contexts in sync.

The conventional handling of .Rhistory files is as follows:

- Load and save .Rhistory within the current working directory
- Only save the .Rhistory file when the user chooses to save the .RData file

Whereas the RStudio handling of .Rhistory files is:

- Load and save a single global .Rhistory file (located in the default working directory)
- Always save the .Rhistory file (even if the .RData file is not saved)

As a result, the contents of the History pane always match the up arrow history within the console.



Customizing RStudio

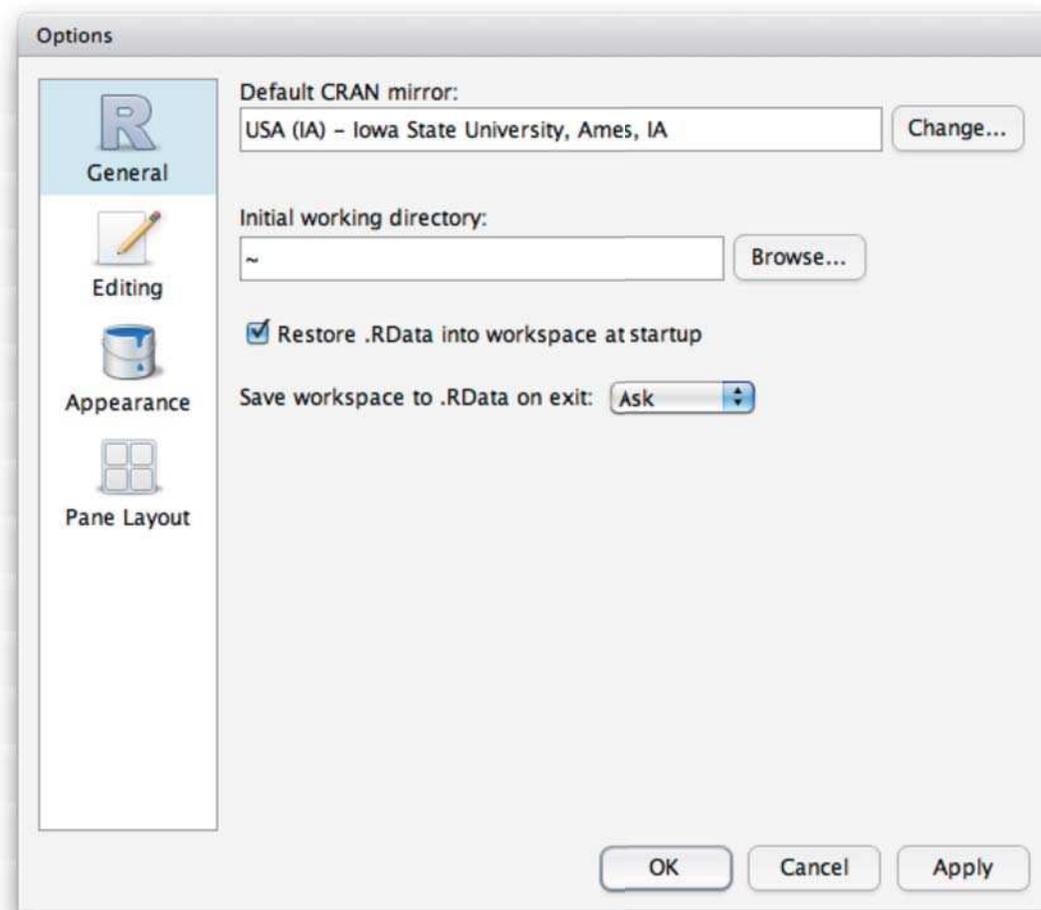
Overview

RStudio options are accessible from the Options dialog **Tools : Options** menu and include the following categories:

- [General R Options](#) — Default CRAN mirror, initial working directory, workspace save and restore behavior.
- [Source Code Editing](#) — Enable/disable line numbers, line highlighting, soft-wrapping for R files, and right margin display; configure tab spacing; set default text encoding.
- [Appearance and Themes](#) — Specify font size and visual theme for the console and source editor.
- [Pane Layout](#) — Locations of console, source editor, and tab panes; set which tabs are included in each pane.

Details on the various settings are provided in the sections below.

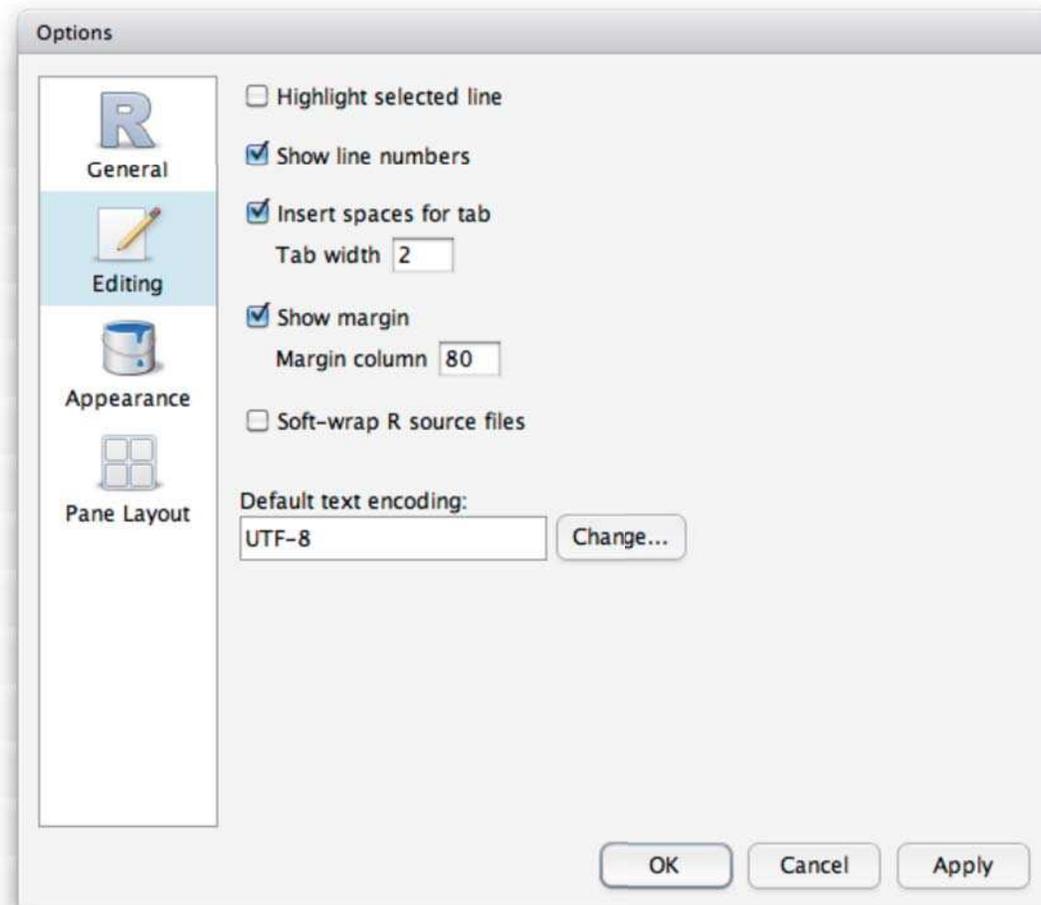
General R Options



- **Default CRAN mirror** — Set the CRAN mirror used for installing packages (can be overridden using the `repos` argument to `install.packages`).
- **Initial working directory** — Startup directory for RStudio. The initial `.RData` and `.Rprofile` files (if any) will be read from this directory. The current working directory and Files pane will also be set to this directory. Note that this setting can be overridden when launching RStudio using a file association or a terminal with a command line parameter indicating the initial working directory.

- **Restore .RData into workspace at startup** — Load the .RData file (if any) found in the initial working directory into the R workspace (global environment) at startup. If you have a very large .RData file then unchecking this option will improve startup time considerably.
- **Save workspace to .RData on exit** — Ask whether to save .RData on exit, always save it, or never save it. Note that if the workspace is not dirty (no changes made) at the end of a session then no prompt to save occurs even if Ask is specified.

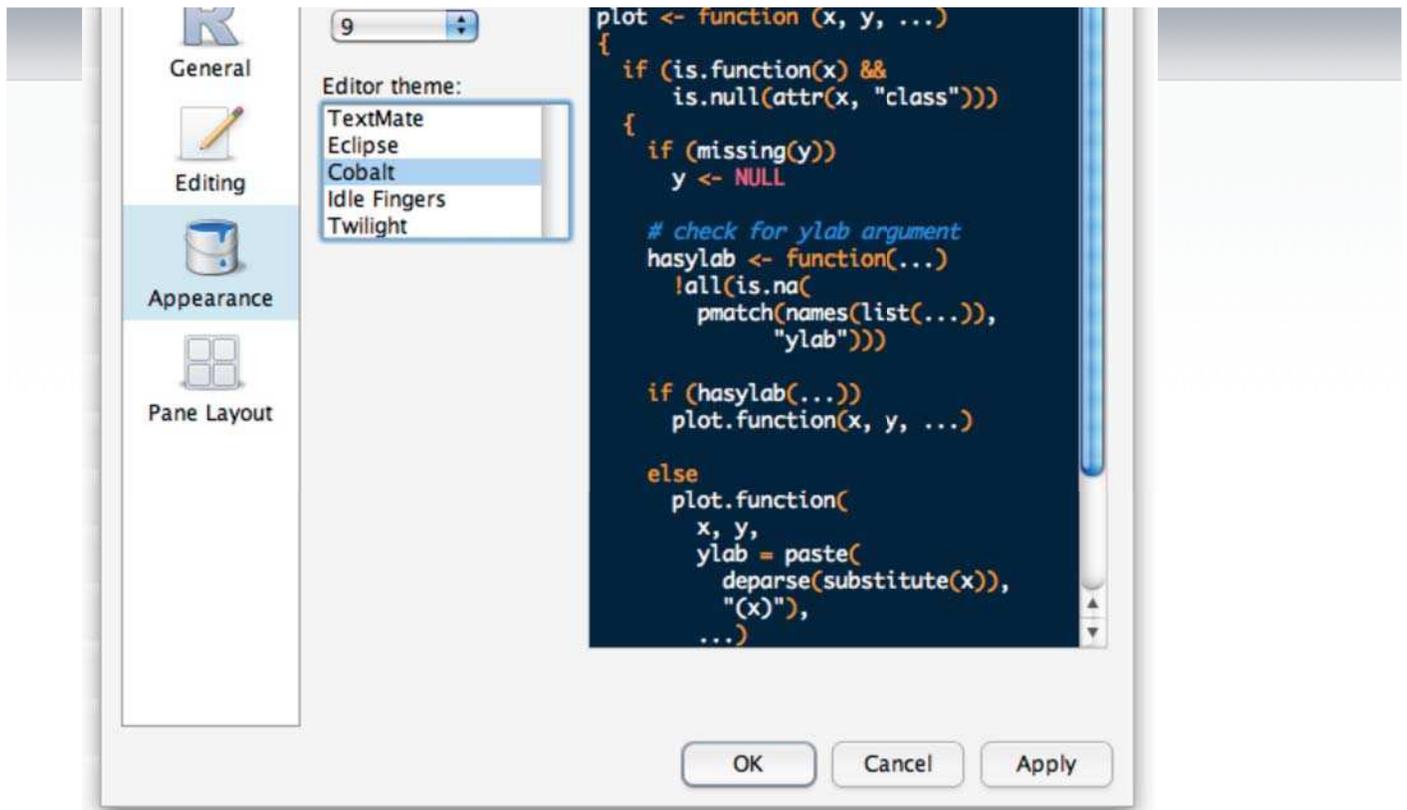
Source Code Editing



- **Highlight selected line** — Add a background highlight effect to the currently selected line of code.
- **Show line numbers** — Show or hide line numbers within the left margin.
- **Insert spaces for tab** — Determine whether the tab key inserts multiple spaces rather than a tab character (soft tabs). Configure the number of spaces per soft-tab.
- **Show margin** — Display a margin guide on the right-hand side of the source editor at the specified column.
- **Soft-wrap R source files** — Wrap lines of R source code which exceed the width of the editor onto the next line. Note that this does not insert a line-break at the point of wrapping, it simply displays the code on multiple lines in the editor.
- **Default text encoding** — Specify the default text encoding for source files. Note that source files which don't match the default encoding can still be opened correctly using the **File : Reopen with Encoding** menu command.

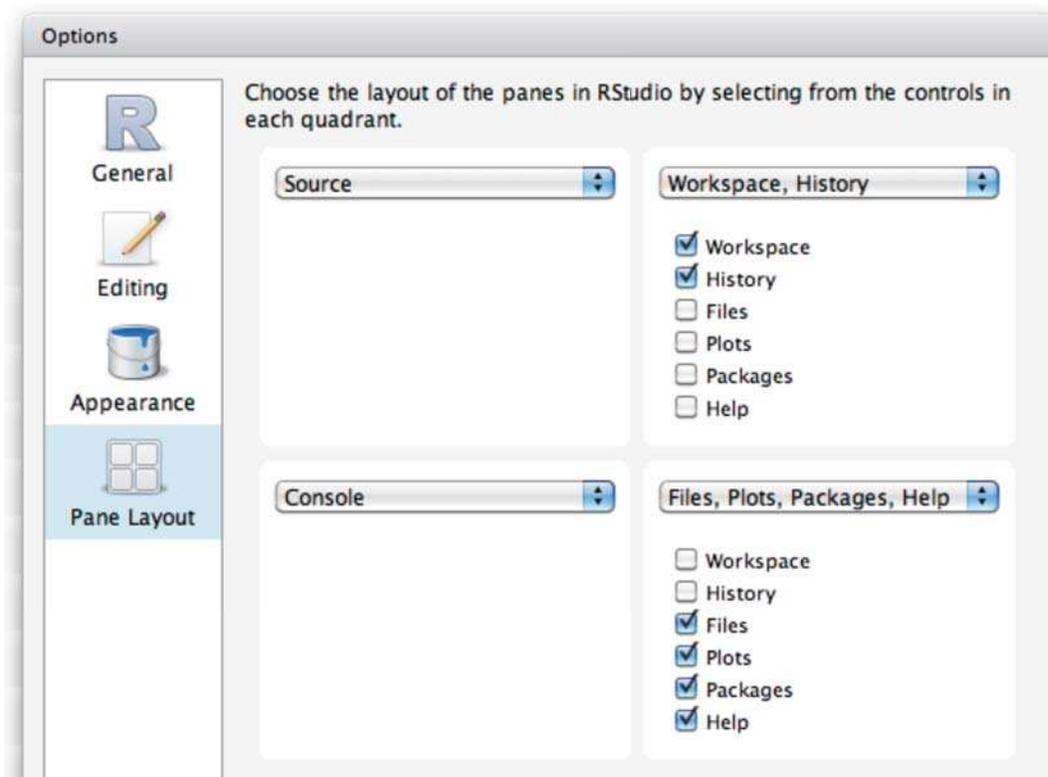
Appearance and Themes





- **Font size** — Set the font size (in points) for panes which display code (Console, Source, History, and Workspace).
- **Editor theme** — Specify the visual theme for the Console and Source panes. You can preview the theme using the inline preview or by pressing the Apply button.

Pane Layout





- Specify the location and tab sets of panes within RStudio.
- Each of the 4 panes is always displayed (it isn't currently possible to hide a pane).



Keyboard Shortcuts

Console

Description	Windows & Linux	Mac
Move cursor to Console	Ctrl+2	Ctrl+2
Clear console	Ctrl+L	Command+L
Move cursor to beginning of line	Home	Command+Left
Move cursor to end of line	End	Command+Right
Navigate command history	Up/Down	Up/Down
Popup command history	Ctrl+Up	Command+Up
Interrupt currently executing command	Esc	Esc
Yank line up to cursor	Ctrl+U	Command+U
Yank line after cursor	Ctrl+K	Command+K
Insert currently yanked text	Ctrl+Y	Command+Y
Insert assignment operator	Alt+-	Option+-

Source

Description	Windows & Linux	Mac
Move cursor to Source Editor	Ctrl+l	Ctrl+l
New document (except on Chrome/Windows)	Ctrl+Shift+N	Command+Shift+N
Open document	Ctrl+O	Command+O
Save active document	Ctrl+S	Command+S
Close active document	Ctrl+Shift+L	Command+Shift+L
Compile PDF (TeX and Sweave)	Ctrl+Shift+P	Command+Shift+P
Run current line/selection	Ctrl+Enter	Command+Enter
Run current document	Ctrl+Shift+Enter	Command+Shift+Enter
Switch to tab	Ctrl+Alt+Down	Ctrl+Option+Down
Previous tab	Ctrl+Alt+Left	Ctrl+Option+Left
Next tab	Ctrl+Alt+Right	Ctrl+Option+Right
First tab	Ctrl+Shift+Alt+Left	Ctrl+Shift+Option+Left
Last tab	Ctrl+Shift+Alt+Right	Ctrl+Shift+Option+Right
Extract function from selection	Ctrl+Shift+F	Command+Shift+F
Comment/uncomment current line/selection	Ctrl+/ /	Command+/ /
Insert assignment operator	Alt+-	Option+-
Transpose Letters		Ctrl+T
Jump to Word	Ctrl+Left/Right	Option+Left/Right

Jump to Start/End	Ctrl+Home/End or Ctrl+Up/Down	Command+Home/End or Command+Up/Down
Delete Line	Ctrl+D	Command+D
Move Lines Up/Down	Alt+Up/Down	Option+Up/Down
Copy Lines Up/Down	Ctrl+Alt+Up/Down	Command+Option+Up/Down
Select	Shift+[Arrow]	Shift+[Arrow]
Select Word	Ctrl+Shift+Left/Right	Option+Shift+Left/Right
Select to Line Start	Shift+Home	Command+Shift+Left or Shift+Home
Select to Line End	Shift+End	Command+Shift+Right or Shift+End
Select Page Up/Down	Shift+PageUp/PageDown	Shift+PageUp/Down
Select to Start/End	Ctrl+Shift+Home/End or Shift+Alt+Up/Down	Command+Shift+Up/Down
Delete Word Left	Ctrl+Backspace	Option+Backspace or Ctrl+Option+Backspace
Delete Word Right		Option+Delete
Delete to Line End		Ctrl+K
Delete to Line Start		Option+Backspace
Indent	Tab (at beginning of line)	Tab (at beginning of line)
Outdent	Shift+Tab	Shift+Tab

Editing (Console and Source)

Description	Windows & Linux	Mac
Undo	Ctrl+Z	Command+Z
Redo	Ctrl+Shift+Z	Command+Shift+Z
Cut	Ctrl+X	Command+X
Copy	Ctrl+C	Command+C
Paste	Ctrl+V	Command+V
Select All	Ctrl+A	Command+A

Completions (Console and Source)

Description	Windows & Linux	Mac
Attempt completion	Tab or Ctrl+Space	Tab or Command+Space
Navigate candidates	Up/Down	Up/Down
Accept selected candidate	Enter, Tab, or Right	Enter, Tab, or Right
Show help for selected candidate	F1	F1
Dismiss completion popup	Esc	Esc

Views

Description	Windows & Linux	Mac
Move cursor to Source Editor	Ctrl+1	Ctrl+1
Move cursor to Console	Ctrl+2	Ctrl+2
Show workspace	Ctrl+3	Ctrl+3
Show data	Ctrl+4	Ctrl+4

Show history	Ctrl+5	Ctrl+5
Show files	Ctrl+6	Ctrl+6
Show plots	Ctrl+7	Ctrl+7
Show packages	Ctrl+8	Ctrl+8
Show help	Ctrl+9	Ctrl+9
Plots		
Description	Windows & Linux	Mac
Previous plot	Ctrl+PageUp	Command+PageUp
Next plot	Ctrl+PageDown	Command+PageDown

© 2011 RStudio, Inc. [About](#) | [Contact](#) | [Follow on Twitter](#) | [FAQ](#) | [License \(AGPL\)](#) | [Trademark](#)



Interactive Plotting with Manipulate

RStudio includes a `manipulate` package that enables the addition of interactive capabilities to standard R plots. This is accomplished by binding plot inputs to custom controls rather than static hard-coded values.

Basic Usage

The `manipulate` function accepts a plotting expression and a set of controls (e.g. slider, picker, or checkbox) which are used to dynamically change values within the expression. When a value is changed using its corresponding control the expression is automatically re-executed and the plot is redrawn.

For example, to create a plot that enables manipulation of a parameter using a slider control you could use syntax like this:

```
library(manipulate)
manipulate(plot(1:x), x = slider(1, 100))
```

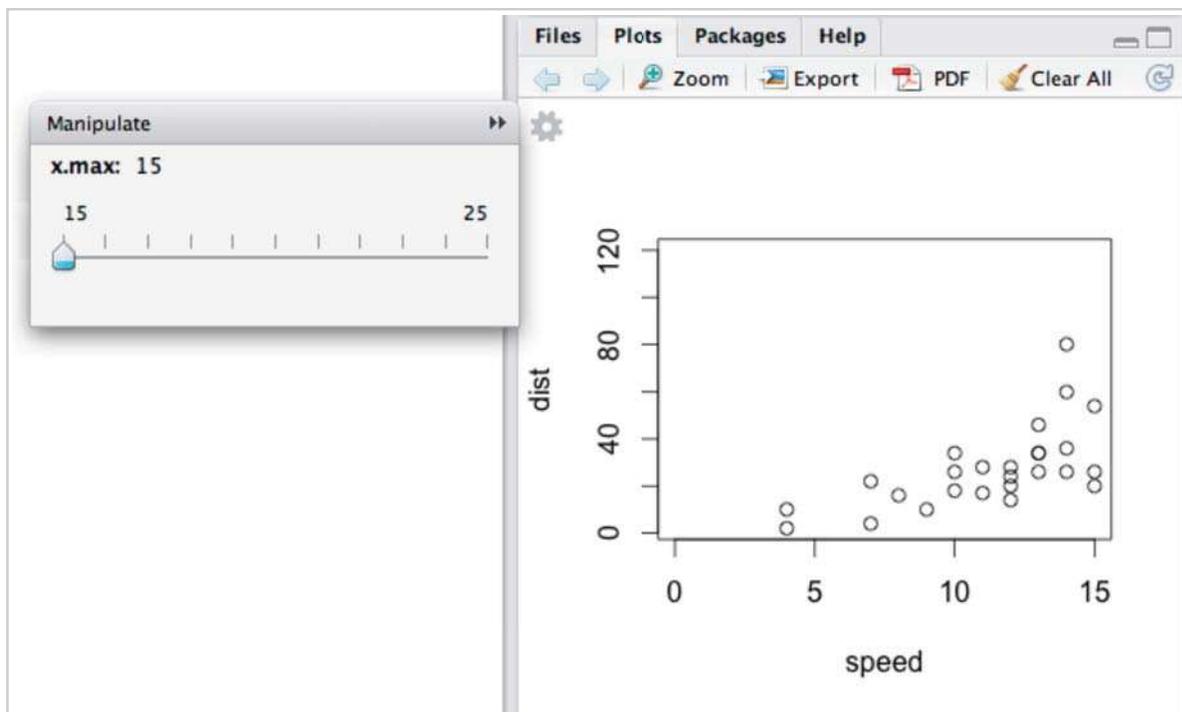
After this code is executed the plot is drawn using an initial value of 1 for `x`. A manipulator panel is also opened adjacent to the plot which contains a slider control used to change the value of `x` from 1 to 100.

Slider Control

The `slider` control enables manipulation of plot variables along a numeric range. For example:

```
manipulate(
  plot(cars, xlim=c(0,x.max)),
  x.max=slider(15,25))
```

Results in this plot and manipulator:



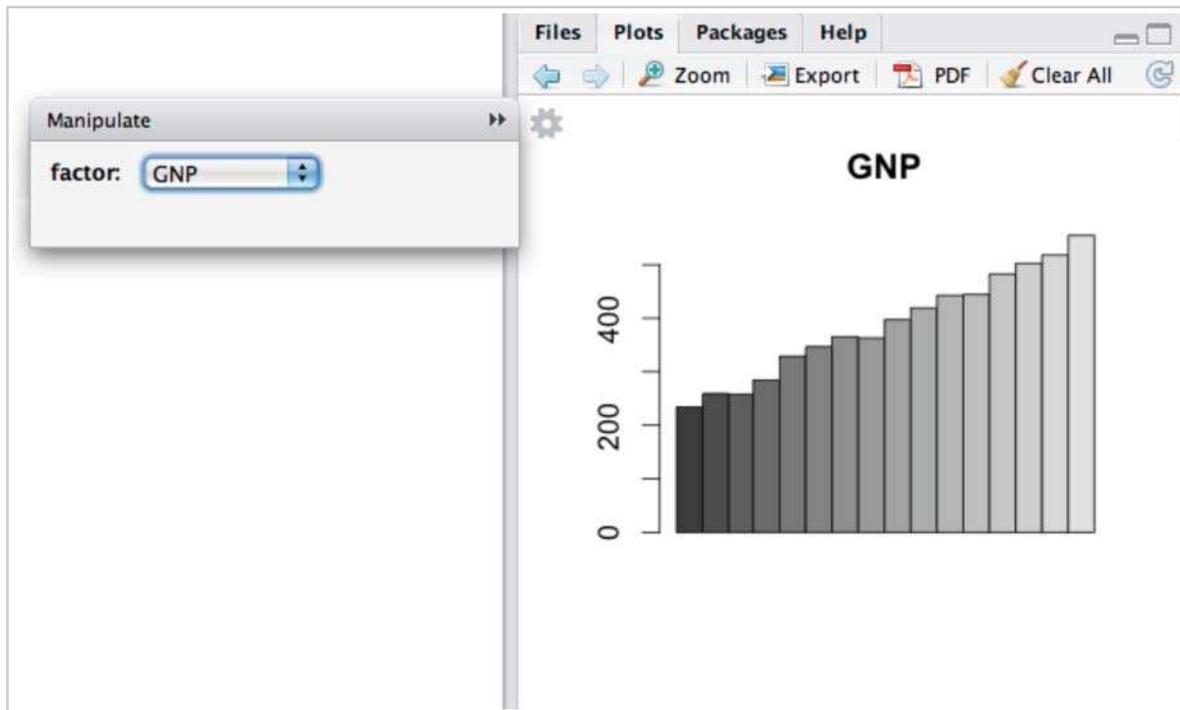
Slider controls also support custom labels and step increments.

Picker Control

The picker control enables manipulation of plot variables based on a set of fixed choices. For example:

```
manipulate(  
  barplot(as.matrix(longley[, factor]),  
    beside = TRUE, main = factor),  
  factor = picker("GNP", "Unemployed", "Employed"))
```

Results in this plot and manipulator:



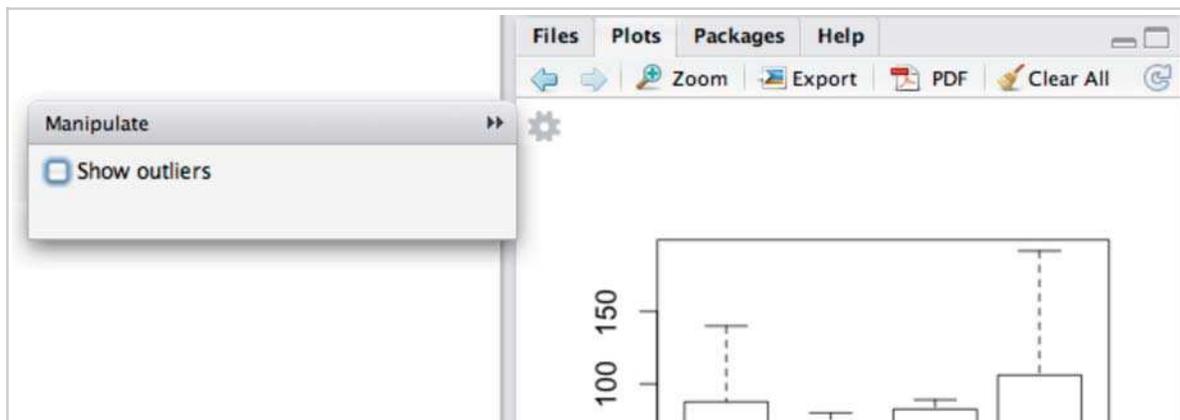
Picker controls support arbitrary value types, and can also include custom user-readable labels for each choice.

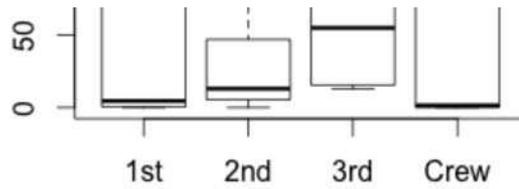
Checkbox Control

The checkbox control enables manipulation of logical plot variables. For example:

```
manipulate(  
  boxplot(Freq ~ Class, data = Titanic, outline = outline),  
  outline = checkbox(FALSE, "Show outliers"))
```

Results in this plot and manipulator:





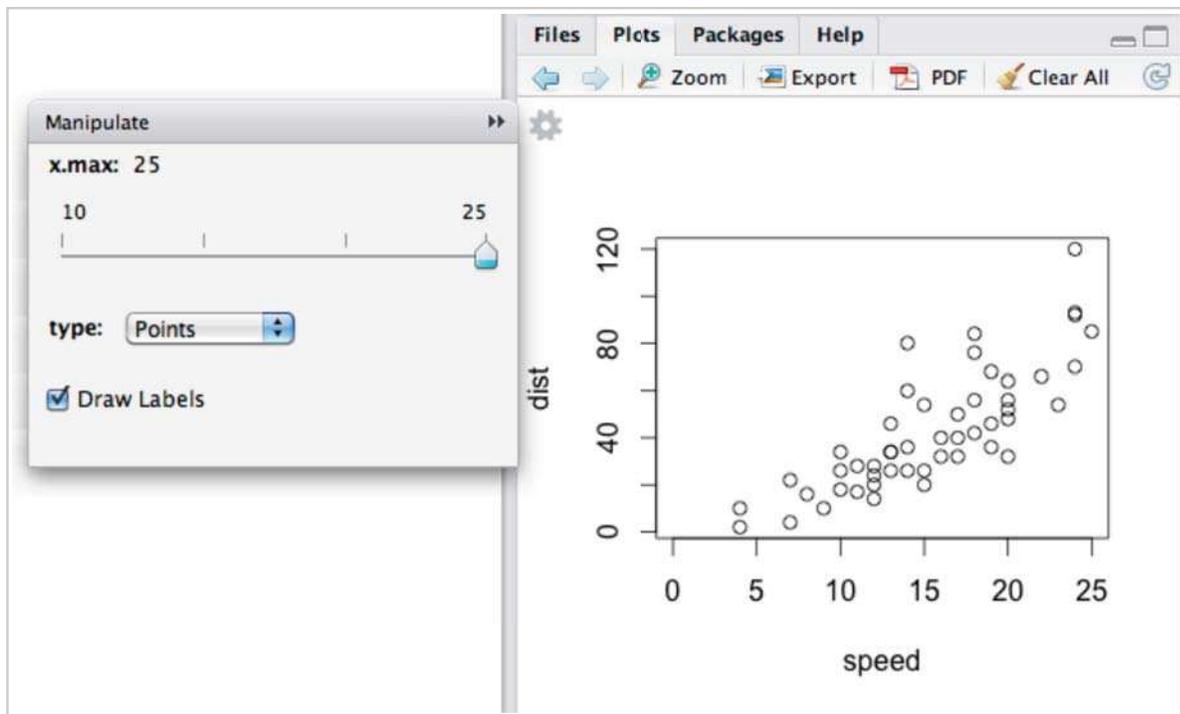
The `manipulate` package documentation contains additional details on all of the options available for the various control types.

Combining Controls

Multiple controls can be combined within a single manipulator. For example:

```
manipulate(
  plot(cars, xlim = c(0, x.max), type = type, ann = label),
  x.max = slider(10, 25, step=5, initial = 25),
  type = picker("Points" = "p", "Line" = "l", "Step" = "s"),
  label = checkbox(TRUE, "Draw Labels"))
```

Results in this plot and manipulator:





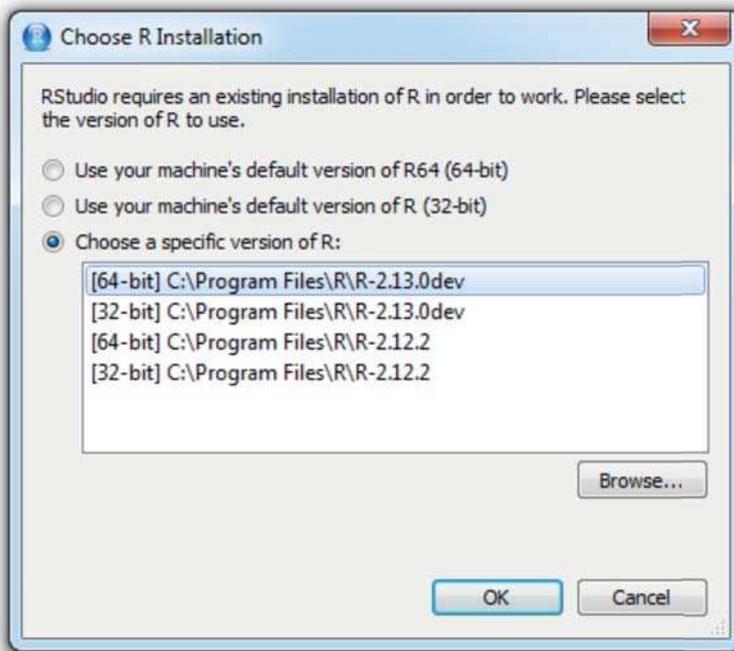
Using Different Versions of R

RStudio requires R version 2.11.1 or higher. Since R versions can be installed side-by-side on a system, RStudio needs to select which version of R to run against. The way this occurs varies between platforms—this article covers how version selection is handled on each platform.

Windows

On Windows, RStudio uses the system's current version of R by default. When R is installed on Windows it writes the version being installed to the Registry as the "current" version of R (the specific registry keys written are described [here](#)). This is the version of R which RStudio runs against by default.

You can override which version of R is used via General panel of the RStudio Options dialog. This dialog allows you to specify that RStudio should always bind to the default 32 or 64-bit version of R, or to specify a different version altogether:



Note that by holding down the Control key during the launch of RStudio you can cause the R version selection dialog to display at startup.

Mac OS X

R from CRAN

On Mac OS X if the only version of R you have installed is the standard R distribution from CRAN then RStudio will by default run against the current version of R.Framework. You can list all of the versions of R.Framework on your system and determine which one is considered the current one by executing the following command:

```
ls -l /Library/Frameworks/R.framework/Versions/
```

To change the current version of R.Framework you can either:

- Run the installer from CRAN for the R version you want to be current

- Use the RSwitch utility available at: <http://r.research.att.com/>
- Update the R.framework/Versions/Current directory alias directly using `ln -s`

R from source (including MacPorts and Homebrew)

When R is installed from CRAN on OS X the R executable is installed at `/usr/bin/R`. However, if R is installed directly from source or via a package manager like MacPorts or Homebrew, then the R executable is installed to either `/usr/local/bin/R` (Homebrew) or `/opt/local/bin/R` (MacPorts). In order to support these variations, RStudio scans for the R executable in the following sequence:

1. `/opt/local/bin/R`
2. `/usr/bin/R`
3. `/usr/local/bin/R`

This order is based on the conventional ordering of the OS X PATH environment variable, and therefore should normally yield the same version that is run when R is executed from a terminal.

If you want to override the version of R selected by RStudio's default behavior then you can set the `RSTUDIO_WHICH_R` environment variable to the R executable that you want to run against. For example, to force RStudio to use the R executable located at `/usr/local/bin`:

```
export RSTUDIO_WHICH_R=/usr/local/bin/R
```

Note that in order for RStudio to see this environment variable it needs to be added to the OS X `environment.plist` file. Instructions for editing this file are [available here](#).

Linux

On Linux, RStudio uses the version of R pointed to by the output of the following command:

```
which R
```

The `which` command performs a search for the R executable using the system PATH. RStudio will therefore by default bind to the same version that is run when R is executed from a terminal.

For versions of R installed by system package managers (e.g. `r-base` on Ubuntu) this will be `/usr/bin/R`. For versions of R installed from source this will typically (but not always) be `/usr/local/bin/R`.

If you want to override which version of R is used then you can set the `RSTUDIO_WHICH_R` environment variable to the R executable that you want to run against. For example:

```
export RSTUDIO_WHICH_R=/usr/local/bin/R
```

Not that in order for RStudio to see this environment variable when launched from the Ubuntu desktop Applications menu (as opposed to from a terminal) it must be defined in the `~/.profile` file.

Web

If you are running RStudio within a web browser then the version of R is determined by whatever version of R is running alongside RStudio Server. The version currently in use on the server can be printed using the following command:

```
> R.version.string
```



Character Encoding

Starting with version 0.93, RStudio supports non-ASCII characters for input and output.

Console

Unicode characters can be used for both input and output in the console.

Source Editor

The RStudio source editor natively supports Unicode characters. It will allow you to type or paste characters from any language, even ones that are not part of the document's character set. RStudio will allow you to save such documents, but will print a warning to the R console that not all characters could be encoded. If you close the document without re-saving in a more suitable encoding, those characters will be lost.

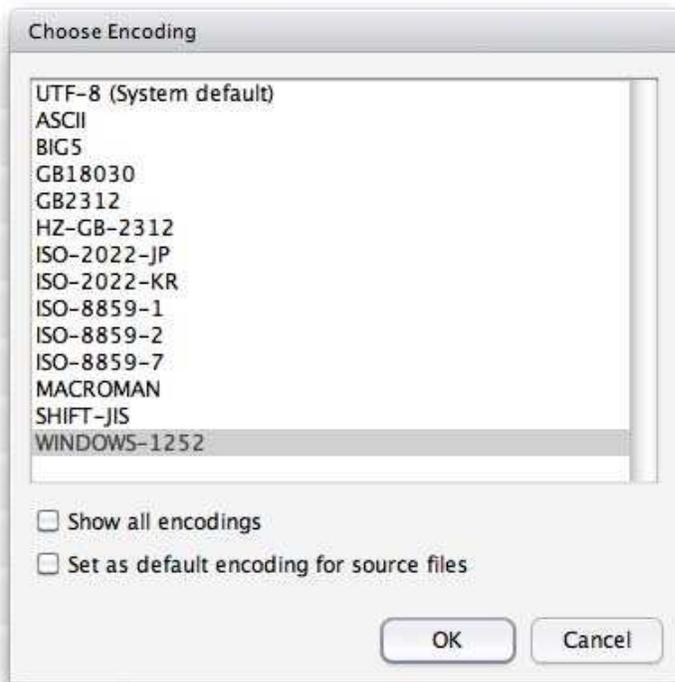
If in doubt about which encoding to use, use UTF-8, as it can encode any Unicode character.

Reading and Writing Files

The RStudio source editor can read and write files using any character encoding that is available on your system:

- You can choose the encoding for reading with **File : Reopen with Encoding**, which will re-read the current file from disk with the new encoding.
- You can also save an open file using a different encoding with **File : Save with Encoding**.

The Reopen and Save with Encoding commands both display the following dialog:



Setting the Default Encoding

If you frequently use the character set, check "Set as default encoding for source files". You can view or change this default in the **Tools : Options** (for Windows & Linux) or **Preferences** (for Mac) dialog, in the **Editing** section.

If you don't set a default encoding, files will be opened using UTF-8 (on Mac desktop, Linux desktop, and server) or the system's default encoding (on Windows). When saving a previously unsaved file, RStudio will ask you to choose an encoding if non-ASCII characters are present.

Known Issues

- If you call `Sys.setlocale` with "LC_CTYPE" or "LC_ALL" to change the system locale while RStudio is running, you may run into some minor issues as RStudio assumes the system encoding doesn't change. If you are on Windows, we recommend you only call `Sys.setlocale` in `.Rprofile`. If you are on Mac or Linux and want to change the system locale, please visit the [support forum](#) and let us know your scenario.
- On Windows, R's `source` function does not work with files that include characters that aren't part of the current system encoding. You may have trouble with RStudio's **Run All** and **Source on Save** commands, as they rely on `source`.



Optimizing your Browser for RStudio

NOTE: This article is only applicable if you are using the RStudio IDE within a web browser (as opposed to using RStudio as a standalone desktop application).

Run a Recent Browser Version

RStudio makes use of a number of advanced web browser features and as a result benefits from running within more up-to-date browser versions. The following are the recommend minimum versions for various browsers:

- [Firefox 3.5](#)
- [Safari 4.0](#)
- [Google Chrome 5.0](#)

Note that RStudio can also be run from within Internet Explorer using the [Google Chrome Frame](#) browser plugin.

Disable Pop-Up Blockers

There are a number of instances where RStudio needs to show a new external popup window (e.g to display a PDF file). We therefore recommend that you disable pop-up blocking for the RStudio domain. Most browsers will prompt you regarding whether you want to enable popups for RStudio the first time one is blocked. Some browsers (such as Safari) may require you to globally enable and disable popups.

Safari and Chrome: Disable Browser Spell Checking

If you are using Safari or Chrome, you may find it desirable to disable "Check Spelling While Typing" (available from the Edit menu) since many of the words you enter in the Source and Console will not be in the built-in dictionary and thus will show up with a red underline when entered.

Firefox for Mac: Install Inline PDF Extension

Firefox for the Mac does not display PDFs inline by default (rather they are downloaded like any other file). RStudio uses PDFs for both printing plots as well as for Sweave/LaTeX documents and having them display inline is much preferred. To enable this you should install the following Firefox extension: <http://code.google.com/p/firefox-mac-pdf/>



Uploading and Downloading Files

NOTE: This article is only applicable if you are using the RStudio IDE within a web browser (as opposed to using RStudio as a standalone desktop application).

Uploading Files

To upload datasets, scripts, or other files to RStudio Server you should take the following steps:

1. Switch to the **Files** pane
2. Navigate to the directory you wish to upload files into
3. Click the **Upload** toolbar button
4. Choose the file you wish to upload and press OK

Note that if you wish to upload several files or even an entire folder, you should first compress your files or folder into a zip file and then upload the zip file (when RStudio receives an uploaded zip file it automatically uncompresses it).

Downloading Files

To download files from RStudio Server you should take the following steps:

1. Switch to directory you want to download files from within the **Files** pane
2. Select the file(s) and/or folder(s) you want to download
3. Click **More** -> **Export** on the toolbar
4. You'll then be prompted with a default file name for the download. Either accept the default or specify a custom name then press OK.

Note that if you select multiple files or folders for download then RStudio compresses all of the files into a single zip archive for downloading.



RStudio Server: Getting Started

Overview

RStudio Server enables you to provide a browser based interface (the RStudio IDE) to a version of R running on a remote Linux server. Deploying R and RStudio on a server has a number of benefits, including:

- The ability to access your R workspace from any computer in any location;
- Easy sharing of code, data, and other files with colleagues;
- Allowing multiple users to share access to the more powerful compute resources (memory, processors, etc.) available on a well equipped server; and
- Centralized installation and configuration of R, R packages, TeX, and other supporting libraries.

RStudio Server works with recent versions of popular Linux distributions including Red Hat and Ubuntu. RStudio Server can also be built and installed from source on other platforms (see notes on this below).

Download and Install

RStudio Server binary packages are available for recent versions of popular Linux distributions including Ubuntu (version 10.04 or higher) and RedHat/CentOS (version 5.4 or higher). For other platforms it is also possible to build and install from source.

Instructions for downloading and installing RStudio Server can be found on the [server downloads](#) page.

Accessing the Server

By default RStudio Server runs on port 8787 and accepts connections from all remote clients. After installation you should therefore be able to navigate a web browser to the following address to access the server:

```
http://<server-ip>:8787
```

RStudio will prompt for a username and password, and will authenticate the user by checking the server's username and password database. Note that user credentials are encrypted using RSA as they travel over the network.

Configuration and Management

RStudio Server has a variety of configuration options (including the ability to change what port the server listens on) as well as a utility for managing the lifetime of the server and remote user sessions. You can find out more about these capabilities in the following articles:

- [Configuring the Server](#)
- [Managing the Server](#)

If you are running RStudio on a public network you may wish deploy RStudio behind another server (e.g. [Nginx](#) or [Apache](#)) which acts as a reverse proxy to it. You can find out more about doing this in the following article:

- [Running with a Proxy](#)



RStudio Server: Configuring the Server

Overview

RStudio is configured by adding entries to two configuration files (note that these files do not exist by default so you will need to create them if you wish to specify custom settings):

```
/etc/rstudio/rserver.conf  
/etc/rstudio/rsession.conf
```

After editing configuration files you should perform a check to ensure that the entries you specified are valid. This can be accomplished by executing the following command:

```
$ sudo rstudio-server test-config
```

Note that this command is also automatically executed when starting or restarting the server (those commands will fail if the configuration is not valid).

Network Port and Address

After initial installation RStudio accepts connections on port 8787. If you wish to change to another port you should create an `/etc/rstudio/rserver.conf` file (if one doesn't already exist) and add a `www-port` entry corresponding to the port you want RStudio to listen on. For example:

```
www-port=80
```

By default RStudio binds to address 0.0.0.0 (accepting connections from any remote IP). You can modify this behavior using the `www-address` entry. For example:

```
www-address=127.0.0.1
```

Note that after editing the `/etc/rstudio/rserver.conf` file you should always restart the server to apply your changes (and validate that your configuration entries were valid). You can do this by entering the following command:

```
$ sudo rstudio-server restart
```

Specifying R Version

By default RStudio Server runs against the version of R which is found on the system PATH (using `which R`). You can override which version of R is used via the `rsession-which-r` setting. For example, if you have two versions of R installed on the server and want to make sure the one at `/usr/local/bin/R` is used by RStudio then you would use:

```
rsession-which-r=/usr/local/bin/R
```

Note again that the server must be restarted for this setting to take effect.

Setting User Limits

There are a number of settings which place limits on which users can access RStudio and the amount of resources they can consume. This file does not exist by default so if you wish to specify any of the settings below you should create the file.

To limit the users who can login to RStudio to the members of a specific group, you use the `auth-required-user-group` setting. For example:

```
auth-required-user-group=rstudio_users
```

You can also limit the total memory, stack size, and number of simultaneous child processes for users using settings like the following:

```
rsession-memory-limit-mb=4000  
rsession-stack-limit-mb=10  
rsession-process-limit=100
```

Additional Settings

There is a separate `/etc/rstudio/rsession.conf` configuration file that enables you to control various aspects of R sessions (note that as with `rserver.conf` this file does not exist by default). These settings are especially useful if you have a large number of potential users and want to make sure that resources are balanced appropriately.

By default if a user hasn't issued a command for 2 hours RStudio will suspend that user's R session to disk so they are no longer consuming server resources (the next time the user attempts to access the server their session will be restored). You can change the timeout (including disabling it by specifying a value of 0) using the `session-timeout-minutes` setting. For example:

```
session-timeout-minutes=30
```

Note that a user's session will never be suspended while it is running code (only sessions which are idle will be suspended).

You can limit the size of file uploads using the `limit-file-upload-size-mb` setting. For example:

```
limit-file-upload-size-mb=100
```

If you are using the XFS filesystem and you have disk quotas enabled you can have RStudio notify the user when they are close to their soft and/or hard quota by specifying the `limit-xfs-disk-quota` setting. For example:

```
limit-xfs-disk-quota=1
```

Finally, you can set the default CRAN repository for the server using the `r-cran-repos` setting. For example:

```
r-cran-repos=http://cran.case.edu/
```

Note again that the above settings should be specified in the `/etc/rstudio/rsession.conf` file (rather than the aforementioned `rserver.conf` file).

Related Topics

- [Getting Started](#)
- [Managing the Server](#)
- [Running with a Proxy](#)



RStudio Server: Managing the Server

Overview

RStudio server management tasks are performed using the `rstudio-server` utility (installed under `/usr/sbin` in binary distributions). This utility enables the stopping, starting, and restarting of the server, enumeration and suspension of user sessions, taking the server offline, as well as the ability to hot upgrade a running version of the server.

Stopping and Starting

If you installed RStudio using a package manager binary (e.g. a Debian package or RPM) then RStudio is automatically registered as a daemon which starts along with the rest of the system. On Ubuntu this registration is performed using an Upstart script at `/etc/init/rstudio-server.conf`. On other systems an `init.d` script is installed at `/etc/init.d/rstudio-server`.

To manually stop, start, and restart the server you use the following commands:

```
$ sudo rstudio-server stop
$ sudo rstudio-server start
$ sudo rstudio-server restart
```

Managing Active Sessions

There are a number of administrative commands which allow you to see what sessions are active and request suspension of running sessions (note that session data is not lost during a suspend).

To list all currently active sessions:

```
$ sudo rstudio-server active-sessions
```

To suspend an individual session:

```
$ sudo rstudio-server suspend-session <pid>
```

To suspend all running sessions:

```
$ sudo rstudio-server suspend-all
```

The suspend commands also have a "force" variation which will send an interrupt to to the session to request the termination of any running R command:

```
$ sudo rstudio-server force-suspend-session <pid>
$ sudo rstudio-server force-suspend-all
```

The `force-suspend-all` command should be issued immediately prior to any reboot so as to preserve the data and state of active R sessions across the restart.

Taking the Server Offline

If you need to perform system maintenance and want users to receive a friendly message indicating the server is offline you can issue the following command:

```
$ sudo rstudio-server offline
```

When the server is once again available you should issue this command:

Upgrading to a New Version

If you install RStudio using a package manager binary (e.g. a Debian package or RPM) and a version of RStudio Server is currently running on the system, then the current running version is automatically upgraded. This includes the following behavior:

- Running R sessions are suspended so that future interactions with the server automatically launch the updated R session binary
- Currently connected browser clients are notified that a new version is available and automatically refresh themselves.
- The core server binary is restarted

Related Topics

- [Getting Started](#)
- [Configuring the Server](#)
- [Running with a Proxy](#)



RStudio Server: Running with a Proxy

Overview

If you are running RStudio on a public network it is strongly recommended that you deploy RStudio behind another web server (e.g. Nginx or Apache) which acts as a reverse proxy to it. Doing this allows you to benefit from the the robust HTTP protocol handling built into the web server. This has both performance (e.g. keep-alive) and security (e.g. rejection of maliciously malformed requests) benefits.

Nginx Configuration

On Ubuntu a version of Nginx that supports reverse-proxying can be installed using the following command:

```
sudo apt-get install nginx
```

To enable an instance of Nginx running on the same server to act as a front-end proxy to RStudio you would add commands like the following to your `nginx.conf` file:

```
location / {
    proxy_pass http://localhost:8787;
    proxy_redirect http://localhost:8787/ $scheme://$host/;
}
```

After adding this entry you'll then need to restart Nginx so that the proxy settings take effect:

```
sudo /etc/init.d/nginx restart
```

Apache Configuration

To enable an instance of Apache running on the same server to act as a front-end proxy to RStudio you need to use the `mod_proxy`. The steps for enabling this module vary across operating systems so you should consult your distribution's Apache documentation for details.

On Ubuntu systems Apache can be installed with `mod_proxy` using the following commands:

```
sudo apt-get install apache2
sudo apt-get install libapache2-mod-proxy-html
sudo apt-get install libxml2-dev
```

Then, to update the Apache configuration files to activate `mod_proxy` you execute the following commands:

```
sudo a2enmod proxy
sudo a2enmod proxy_http
```

Once you have enabled `mod_proxy` in your Apache installation you need to add the required proxy commands to your `VirtualHost` definition. For example:

```
<VirtualHost *:80>

    <Proxy *>
        Allow from localhost
    </Proxy>

    ProxyPass          / http://localhost:8787/
```

```
ProxyPassReverse / http://localhost:8787/
```

```
</VirtualHost>
```

Note that if you want to serve RStudio from a custom path (e.g. /rstudio) you would replace the ProxyPass directives described above to:

```
ProxyPass /rstudio/ http://localhost:8787/
ProxyPassReverse /rstudio/ http://localhost:8787/
RedirectMatch permanent ^/rstudio$ /rstudio/
```

Finally, after you've completed all of the above steps you'll then need to restart Apache so that the proxy settings take effect:

```
sudo /etc/init.d/apache2 restart
```

RStudio Configuration

Once you are successfully proxying requests from Nginx or Apache to RStudio you should change the port RStudio listens on from 0.0.0.0 (all remote clients) to 127.0.0.1 (only the localhost). This ensures that the only way to connect to RStudio is through the proxy server. You can do this by adding the `www-address` entry to the `/etc/rstudio/rsserver.conf` file as follows:

```
www-address=127.0.0.1
```

Note that this config file does not exist by default so you may need to create it if it doesn't already exist.

Related Topics

- [Getting Started](#)
- [Configuring the Server](#)
- [Managing the Server](#)



About RStudio

The RStudio Project

We started the RStudio project because we were excited and inspired by R. The [creators of R](#) provided a flexible and powerful foundation for statistical computing; then made it free and open so that it could be improved collaboratively and its benefits could be shared by the widest possible audience.

It's better for everyone if the tools used for research and science are free and open. Reproducibility, widespread sharing of knowledge and techniques, and the leveling of the playing field by eliminating cost barriers are but a few of the shared benefits of free software in science.

RStudio is an integrated development environment (IDE) for R which works with the standard version of R available from CRAN. Like R, RStudio is available under a free software license. Our goal is to develop a powerful tool that supports the practices and techniques required for creating trustworthy, high quality analysis. At the same time, we want RStudio to be as straightforward and intuitive as possible to provide a friendly environment for new and experienced R users alike. RStudio is also a company, and we plan to sell services (support, training, consulting, hosting) related to the open-source software we distribute.

We're looking forward to joining the R community, learning from users, growing the product, and hopefully making a meaningful contribution to the practice of research and science.

Our Team

**JJ Allaire**

JJ Allaire is a software engineer and entrepreneur who has created a wide variety of products including [ColdFusion](#), [Windows Live Writer](#), [Lose It!](#), and [RStudio](#).

**Joe Cheng**

Joe Cheng is a software engineer who has worked at a number of startups including Allaire, Upromise, and Onfolio. Most recently, he worked at Microsoft as the development lead for [Windows Live Writer](#).

**Josh Paulson**

Josh Paulson is a product manager who has been working with R for over 4 years, focusing principally on data visualization and financial modeling applications.

**Paul DiCristina**

Paul DiCristina is a designer with broad experience including consumer, enterprise, mobile, and web. Paul's design portfolio includes [RStudio](#), [Lose It!](#) and many other apps and sites.

Contact Us

Support: <http://support.rstudio.org>

Feedback: feedback@rstudio.org

Inquiries: info@rstudio.org



RStudio v0.93 — Release Notes

April 11th, 2011

New Features

Source editor enhancements

We've added some new features & options to the source editor. We've also received lots of feedback on more advanced capabilities users want in the editor and we will definitely address this in upcoming releases. New stuff in the editor includes:

- Highlight all instances of selected text
- Insert spaces for tabs (soft-tabs)
- Customizable print margin line
- Selected line highlight
- Toggle line numbers on/off
- Optional soft-wrapping for R source files

The docs on [source code editing options](#) include more details.

Customizable layout and appearance

One of the most frequently requested features we've have is the ability to put the Console and Source views side-by-side. This configuration (and others) are now possible. New appearance and layout options include:

- Customize locations of panes and tabs
- Change default font size for code views
- Select from various editor themes including TextMate, Eclipse, and others.

For more details see the docs on [appearance and layout options](#).

Interactive plotting (manipulate)

This release includes a package called `manipulate` that can be use to create interactive plots within RStudio. `Manipulate` is very flexible and includes the following capabilities:

- Generate plots with inputs bound to custom controls (rather than being hard-coded to a single value)
- Variety of control types including slider, picker, and checkbox.
- Controls appear next to the plot and can be easily shown and hidden

More details as well as screenshots with examples can be found in the [manipulate documentation](#).

Works with R installed from source

The first beta of RStudio was compatible with the binary version of R distributed from CRAN. The current release works with any version of R, including:

- R built and installed from source using `make install`
- MacPorts or Homebrew versions of R on MacOS X

The docs on [using different versions of R](#) describe how RStudio determines which R to run against on each platform.

Character encoding

In this release we've significantly improved handling of non-ASCII characters, this includes:

- Unicode characters can be used for input and output in the console.

- The source editor supports Unicode characters and can open/save files using any character encoding.
- A product-wide default encoding can be set, and can be overridden on a per-document basis.

See the [character encoding](#) documentation for more details.

Improved management of working directories

We've added a number of new features to make it easier to switch between working contexts located in different directories. These include:

- Option to specify a default initial working directory
- Tools | Change Working Dir menu command to change both the working directory and Files pane.
- Optional file associations for .RData and .R that initialize RStudio within the opened file's directory
- Windows: Startup in working directory specified for shortcuts
- Mac: Startup in folder dragged and dropped on RStudio Dock icon
- Linux: Startup in terminal working directory when run from the command line

The docs on [working directories](#) and [workspaces](#) go into more depth on these features.

Other Enhancements

Console

- Recognize `\r` character in console so that `txtProgressBar` works as expected
- Lift restrictions on size of console input which can be sent to R (was 4K total, is now 4K per line)
- Jump to next non-blank line in source after executing via `Ctrl-Enter`

Source Editing

- Improved size and legibility of default fonts on Windows & Linux
- New keyboard shortcuts:
 - `Alt-` for inserting assignment ("`<-`") operator.
 - `Ctrl+Shift+Home/End` for select to start/end
 - `Ctrl+Shift+P` for Compile PDF
- Add "return" to list of symbols syntax highlighted as a keyword

Compatibility

- Compatible with R 2.11.1 on Mac (previously required R 2.12)
- Added `CFBundleSignature` to Mac version
- Correctly initialize `memory.limit` to available physical memory on 64-bit Windows
- Ensure that R uses Internet2 on Windows for interoperability with proxy servers.
- Compatibility with changes to the R 2.13 internal web server (pass headers to custom handlers).
- Allow RStudio desktop to run under root account
- Improved support for openSUSE (still requires install from source):
 - Added `install-dependencies-zypper` script
 - Added `init.d` script for daemon management

Packaging and Installation

- Added `RStudio.version` function to show current version of RStudio
- Changed name of RStudio binary from `rdesktop` to `rstudio` (avoid conflict with existing `rdesktop` binary)
- Added `/usr/bin/rstudio` soft-link
- Change DEB and RPM dependency on base R package to "recommends" rather than "depends"
- Made it more straightforward to install from source:
 - Eliminated git pull requirement (can now build directly from tarball)

- Optionally use system package manager installed versions of Qt4 & Boost

Miscellaneous

- Added Tools menu with Interrupt R, Change Working Dir, and Options commands.
- Add support for loading .rda files into Workspace.
- Improved file icons including new custom icons for Rnw and Rd files.
- Respect both R_USER and HOME environment variables for determining location of R home directory
- Render plot changes on calls to Sys.sleep (enables animated plots)
- Workaround Ubuntu TeX ~ substitution bug by using pdflatex rather than texi2dvi

Bug Fixes

Console

- Workspace restored message prints at startup even if no workspace was restored
- Numeric keypad Enter and navigation keys not correctly interpreted by console
- Esc key not always correctly interpreted when attempting to exit from incomplete command.

Source Editing

- Source pane can become fully selected and impossible to unselect.
- Control-Enter to execute sometimes results in selection not updating properly
- Jump to Word (Ctrl+Right) doesn't navigate past '[' character.
- Ctrl+Backspace doesn't delete previous word on Windows
- Characters illegible when Lucida Grande is installed on Mac systems
- Active tab in source mode sometimes hidden or partially obscured
- Control+S repeats last Undo/Redo on Firefox 3.6
- Print from source view not working in Firefox 4
- Eliminate key binding conflicts for international keyboard layouts
- Incorrect shortcut key displayed in tooltip for Run from source commands
- Variables with dots (".") in their names not highlighting on double-click.

Plotting

- Graphics device not reselected after closing other device (e.g. pdf or png device)
- X11 device and rgl package not working properly on on Mac.
- Save as PNG command not working on Linux
- Re-entrant plot rendering routine causes crash for plots which take a long time to be rendered.
- plotmath expressions not rendered correctly
- Plot history can grow arbitrarily large and cause disk/memory problems.

Compatibility

- rjava package not working on Linux due to incomplete LD_LIBRARY_PATH
- Not always correctly detecting whether TeX is installed
- Dependency on psmisc package not specified for RStudio Server
- Incorrect file association for download of RPM on Fedora

Miscellaneous

- Failed to start when running behind some proxy server configurations.
- Unable to initialize from .Rhistory file that is owned by root
- Exit delay of 2-3 seconds in Mac version
- Crash when custom gtk theme contains missing or invalid images for standard icons

- Unresponsiveness when viewing extremely large data frames
- Conflicting RStudio instances running in parallel if launched from different executable paths

© 2011 RStudio, Inc. [About](#) | [Contact](#) | [Follow on Twitter](#) | [FAQ](#) | [License \(AGPL\)](#) | [Trademark](#)



Frequently Asked Questions

What is RStudio?

RStudio is an integrated development environment (IDE) for R which works with the standard version of R available from CRAN. RStudio includes a wide range of productivity enhancing features and runs on all major platforms. RStudio can optionally also be run as server which enables you to provide a browser based interface to a version of R running on a remote system.

What versions of R is RStudio compatible with?

In order to run RStudio you need to have already installed R 2.11.1 or higher. You can download the most recent version of R for your environment from [CRAN](#).

What is the best way to get started with R and RStudio?

To get started with R there are a wide variety of learning and reference materials available online. We recommend you check out the [Getting Help with R](#) article to find the appropriate resources.

To get started using RStudio check out the [Product Overview](#) as well as the other articles linked to from the [documentation](#).

What is the difference between RStudio Desktop and RStudio Server?

RStudio Desktop is an R IDE that works with the version of R you have installed on your local Windows, Mac OS X, or Linux workstation. RStudio Desktop is a standalone desktop application that in no way requires or connects to RStudio Server.

RStudio Server is a Linux server application that provides a web browser based interface to the version of R running on the server. For more on why you might want to deploy an RStudio Server see the [server documentation](#).

Will RStudio be providing a hosted version of RStudio Server?

Yes, we do plan to provide a hosted version of RStudio Server so that customers can use RStudio over the web without having to deploy their own servers. We haven't announced a specific timeframe for this service yet, however if you are interested in testing it when the beta version becomes available please contact us at info@rstudio.org.

What license is RStudio available under?

RStudio is available under the [GNU Affero General Public License v3](#). The AGPL v3 is an open-source license that guarantees the freedom to share and change the software, and to make sure it remains free software for all its users.

Where can I find the latest updates and news about RStudio?

To keep up with the latest RStudio developments you can subscribe to our [blog](#) or [follow us](#) on Twitter.

Where can I get support for using RStudio?

Our [support web site](#) includes a knowledge base and discussion forum for reporting problems, asking questions, and discussing new ideas and features. Our team and others in the community actively participates in the forum so it is a great place to go for help or answers.



Getting Help with R

There are a number of good resources available on the web for both learning R and seeking answers to questions about how to accomplish various tasks. This article summarizes a few of the more helpful ones.

Learning R

If you are just learning R there are a number of good places to start:

- This [basic R tutorial](#) takes you step-by-step through the core functions of R.
- The CRAN [Introduction to R](#) provides a more complete and detailed overview of the entire R language.
- This article provides a nice [introduction to R for those coming from other languages](#).
- [The R Reference Card](#) provides a useful quick reference for how to perform common tasks in R.
- The Carnegie Mellon Open Learning Initiative has a free online [Introduction to Statistics](#) course has an option to do the exercises and labs using R.

If you have some familiarity with R and want to learn about the system or particular features in more depth these resources might be helpful.

- This Stack Overflow question provides some pointers to good [books for learning the R language](#).
- The CRAN [Contributed Documentation](#) page lists other manuals, tutorials, etc. provided by users of R.
- Once you've gained some familiarity with R, [The R Inferno](#) provides an entertaining roadmap to some of the deeper subtleties of the language and how to work with it most effectively.
- The [Google R Style Guide](#) provides some guidelines for writing readable and maintainable R code.

Asking Questions

A great place to start for any question about R is the [RSeek meta search engine](#), which provides a unified interface for searching the various sources of online R information. If there is an answer to question already available there is a good chance that RSeek can locate it.

If you aren't able to locate an answer using RSeek, the following are good places to find an answer or ask a question:

- The [R-help mailing list](#) is a very active list with questions and answers about problems and solutions using R. Before posting to the list you can also [search the list archives](#) to see if an answer already exists.
- Stack Overflow is also becoming an increasingly important resource for seeking [answers to questions about R](#).
- If you have a question that is more about statistical methodology there are also plenty of R users active on the [CrossValidated Q&A community](#).

Finding Packages

There are thousands of R packages [available from CRAN](#) but navigating them all can be a challenge. The following resources can help find the packages most appropriate for your tasks:

- [CRAN Task Views](#) provides comprehensive summaries of the packages most commonly used in various disciplines.
- [crantastic](#) is a community site for R packages where you can search for, review, and tag CRAN packages.

News and Information

The R community is growing rapidly and there are lots of new things happening all the time. If you want to stay on top of what's happening we recommend keeping up with the following sites:

- [R-bloggers](#) is a news site that combines posts from over 140 R bloggers. Almost everything that happens in the R community is mentioned and discussed on R-bloggers.
- Users in the R community also frequently record videos of presentations, seminars, and user-groups. The [R Videos channel](#) run by Drew Conway is a great way to keep up with all of the available videos.



RStudio™ Trademark

The RStudio Integrated Development Environment (IDE), under the terms of the Affero General Public License, version 3, may be redistributed or modified. However, the name RStudio™ is a trademark of RStudio, Inc. and all rights in the RStudio™ trademark are exclusive to RStudio, Inc.

The RStudio™ trademark may be displayed on any deployment of the RStudio IDE for its use in the unmodified form supplied by RStudio, Inc. For example, individuals, businesses or organizations may deploy and use the unmodified RStudio IDE bearing the RStudio™ trademark on individual workstations or on public or private networks. In addition, the RStudio™ trademark may be used to indicate or refer to the unmodified RStudio IDE, provided that any such use is accompanied by the following notice: "RStudio is a trademark of RStudio, Inc."

Anyone wishing to use the RStudio™ trademark for any reason other than those listed above, including but not limited to advertising, on hardware, or on software derivative of the RStudio IDE, must obtain the express, written permission of RStudio, Inc. in advance. To request permission to use the RStudio™ trademark or report suspected, unauthorized use please contact us at info@rstudio.org.