Chapter 109

Exporting Data to R

Introduction

The free **R** programming language gives access to a core set of statistical and graphical tools, as well as many thousands of user-created packages. These capabilities grow in number by the hundreds or even the thousands each year. While the capabilities are immense, there is a learning curve for its use. One step that is difficult for many new users of **R** is importing the data and appropriately referencing the columns.

The 'Exporting Data to R' procedure is used to

1. Generate a data file from an **NCSS** data table that can be read by **R**.
2. Provide the **R** code to import the file and ready the columns for immediate use.

Once the data is read into **R**, the user can employ the desired **R** tools from the various **R** libraries.

List of Examples

The following examples of exporting data from **NCSS** and analyzing it in **R** are given at the end of this chapter:

1. Exporting and Reading in Data
2. Running Some Basic Functions in R
3. Cochran-Armitage Trend Test (Exact)
4. Quade Test for Unreplicated Complete Block Designs
5. Interrupted Time Series Analysis
6. Generalized Estimating Equations
7. Little's Missing Completely at Random Test
8. Viewing Data on Maps
9. Multivariate Adaptive Regression Splines
10. Factor Analysis with Oblique Rotation
11. Hierarchical Clustering using Squared Euclidean Distance
12. Rolling Correlation
13. Classification and Regression Trees
14. Meta-Analysis of One Proportion
15. Exporting Data from R

Procedure Details

The tools and output from running the 'Exporting Data to R' procedure are described in the following sections.

Generating a CSV File

The **NCSS** 'Exporting Data to R' tool first creates a .CSV file from your **NCSS** data table. The filename is chosen by the user, as well as the file path. The file can consist of all the **NCSS** column names and data, or a subset of the columns and rows. When selecting a subset of rows, the user can either enter a list or range of rows to export (e.g., "1-10, 15, 20-25" or "1-20"), or, if there is an active filter, the user can choose to export only those rows that pass the filter.

R Code to Read in the CSV File

Two of the output options of the **NCSS** 'Exporting Data to R' tool give the **R** code that is needed to read the CSV data table into **R**. After the 'Exporting Data to R' procedure is run to create the CSV file, the code from the **NCSS** output can be copied and pasted into **R**. When the code is pasted into **R**, a 'data frame' is generated in **R** and the columns/variables are ready to be used.

R Code Functions for Examining the Imported Data Frame

In addition to generating the CSV file and providing the code to import it into **R**, the 'Exporting Data to R' procedure also gives a variety of ready-to-use commands to describe or summarize the imported data frame. These lines of code can be used independently of each other. One purpose of these functions is to allow you to verify that the data has been imported as expected. Another purpose is to allow you to view the available columns and their contents.

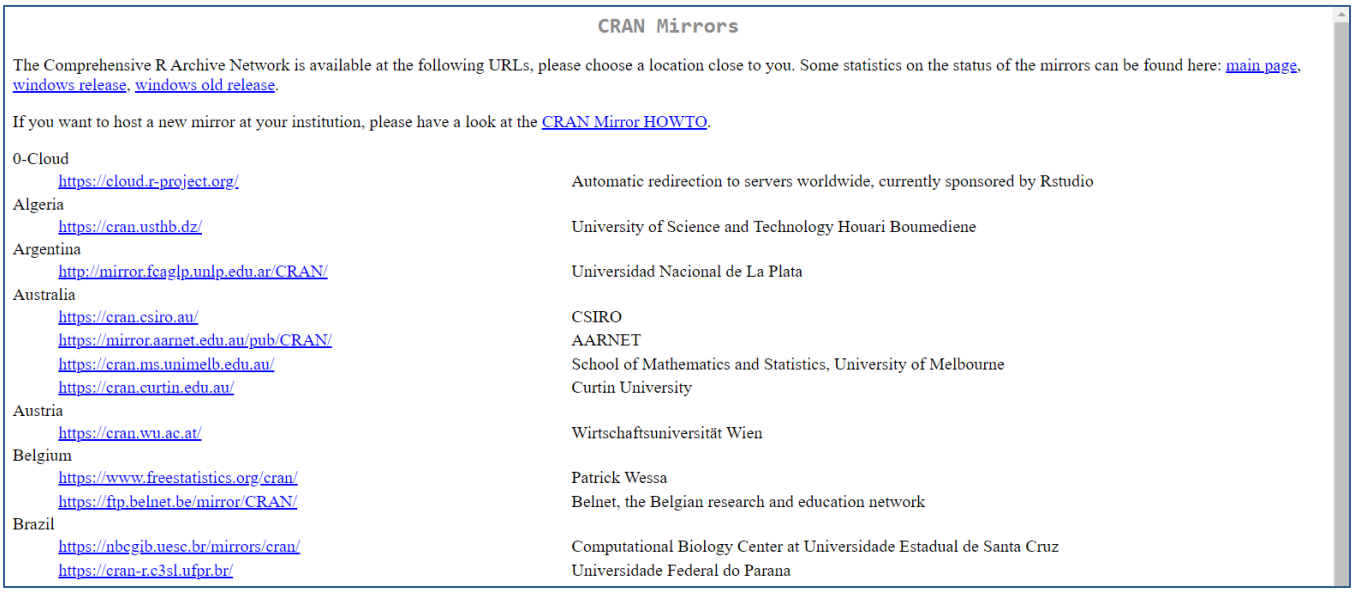
R Startup Basics

Many books, articles, training videos, and classes are available for learning to use **R**. The following sections are intended only to provide a brief background of how to initially run **R**.

Obtaining the R Program

The **R** program can be downloaded from one of the Comprehensive R Archive Network (CRAN) Mirrors at:

https://cran.r-project.org/mirrors.html



CRAN Mirrors

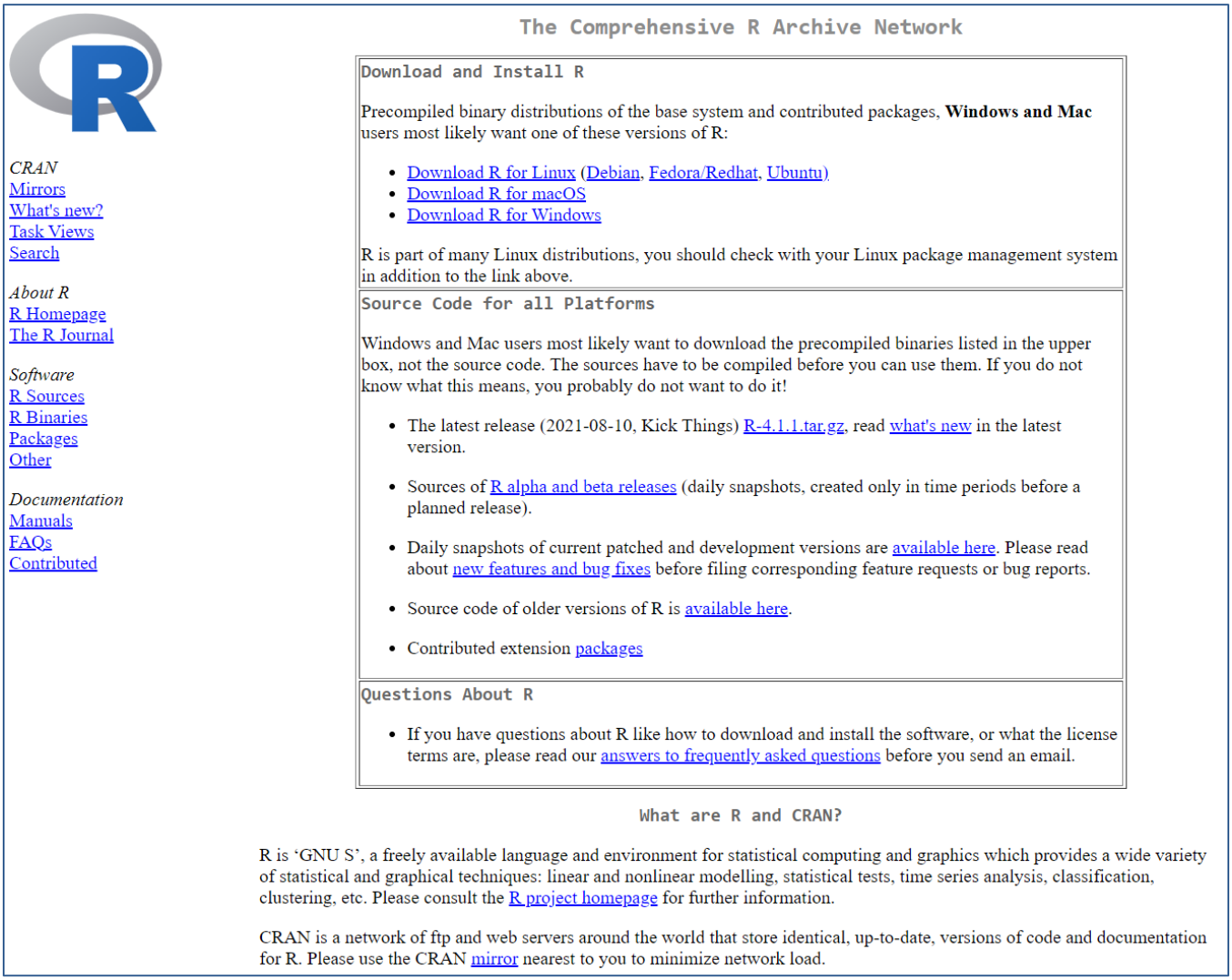
The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: main page, windows release, windows old release.

If you want to host a new mirror at your institution, please have a look at the CRAN Mirror HOWTO.

0-Cloud	
https://cloud.r-project.org/	Automatic redirection to servers worldwide, currently sponsored by Rstudio
Algeria	
https://cran.usthb.dz/	University of Science and Technology Houari Boumediene
Argentina	
http://mirror.fcaglp.unlp.edu.ar/CRAN/	Universidad Nacional de La Plata
Australia	
https://cran.csiro.au/	CSIRO
https://mirror.aarnet.edu.au/pub/CRAN/	AARNET
https://cran.ms.unimelb.edu.au/	School of Mathematics and Statistics, University of Melbourne
https://cran.curtin.edu.au/	Curtin University
Austria	
https://cran.wu.ac.at/	Wirtschaftsuniversität Wien
Belgium	
https://www.freestatistics.org/cran/	Patrick Wessa
https://ftp.belnet.be/mirror/CRAN/	Belnet, the Belgian research and education network
Brazil	
https://nbcgib.uesc.br/mirrors/cran/	Computational Biology Center at Universidade Estadual de Santa Cruz
https://cran-r.c3sl.ufpr.br/	Universidade Federal do Parana

You can select any of the country mirror sites, but it may be preferrable to choose one that is nearer to your location.

Clicking on one of the mirror site links gives:



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-08-10, Kick Things) R-4.1.1.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

What are R and CRAN?

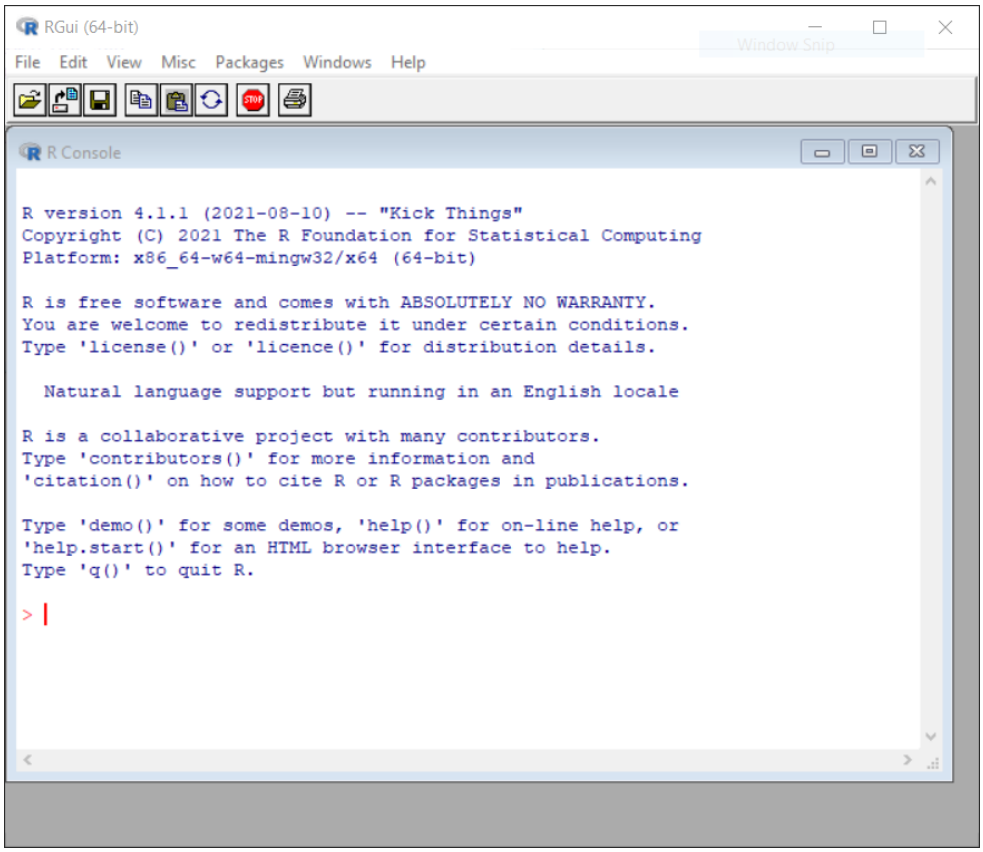
R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.

In the top box, you can select the download corresponding to your operating system. Then follow the installation instructions.

The R Interface (Console)

Although there are several interface programs that can be used as a front end for **R**, the typical use is through the command-line interpreter. When the **R** program is opened, it will appear as:



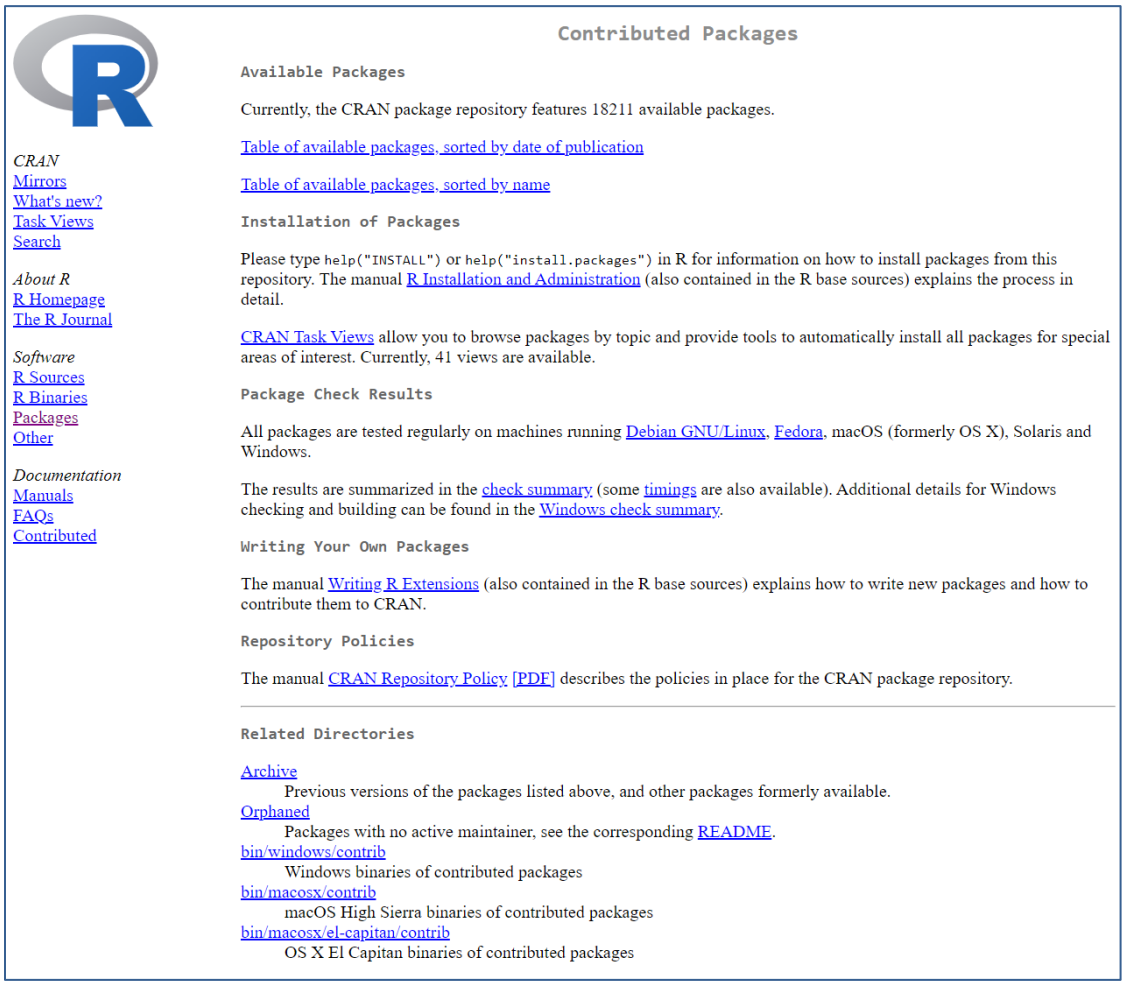
R Packages

Packages are collections of data, code, functions, tests, and documentation. **R** comes with a standard set of packages. Thousands of additional packages, developed by the **R** community, can be downloaded, installed, and used in **R**.

To view a current list of available packages (with brief descriptions), you can go to the same site that was used to initially download **R**, namely,

https://cran.r-project.org/mirrors.html

When a mirror is selected, the same screen appears (shown previously in the Obtaining the R Program section) as was used to download **R**. In this case, to see the list of all packages, choose 'Packages' on the left, or at the bottom of the middle box. The following will be shown:



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Contributed Packages

Available Packages

Currently, the CRAN package repository features 18211 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type help("INSTALL") or help("install.packages") in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 41 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, macOS (formerly OS X), Solaris and Windows.

The results are summarized in the check summary (some timings are also available). Additional details for Windows checking and building can be found in the Windows check summary.

Writing Your Own Packages

The manual Writing R Extensions (also contained in the R base sources) explains how to write new packages and how to contribute them to CRAN.

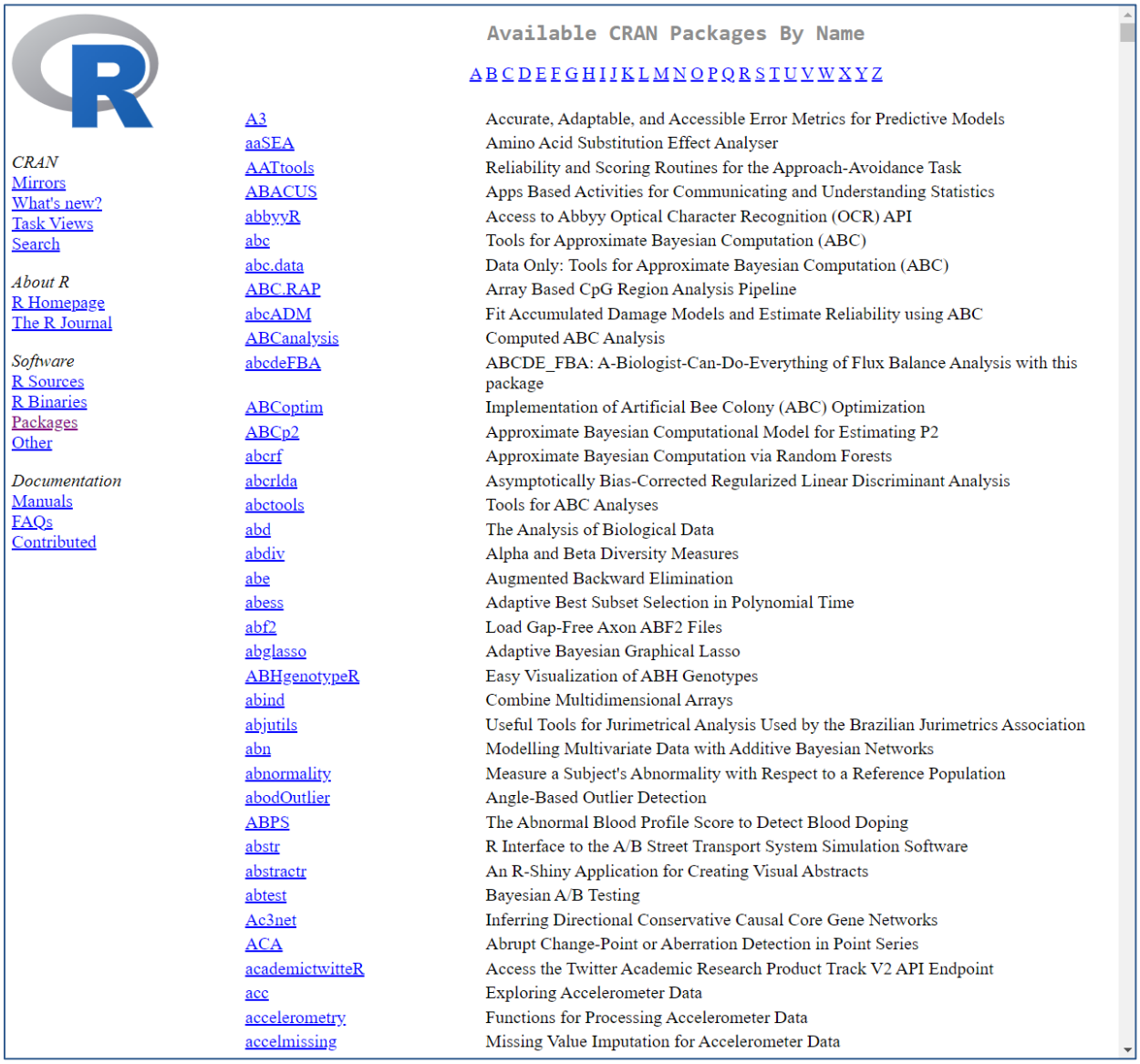
Repository Policies

The manual CRAN Repository Policy [PDF] describes the policies in place for the CRAN package repository.

Related Directories

Archive
Previous versions of the packages listed above, and other packages formerly available.
Orphaned
Packages with no active maintainer, see the corresponding README.
bin/windows/contrib
Windows binaries of contributed packages
bin/macosx/contrib
macOS High Sierra binaries of contributed packages
bin/macosx/el-capitan/contrib
OS X El Capitan binaries of contributed packages

The packages can be viewed by clicking on one of the two links at the top, sorting by date or by name. The scrollable table of packages sorted by name is shown here:



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

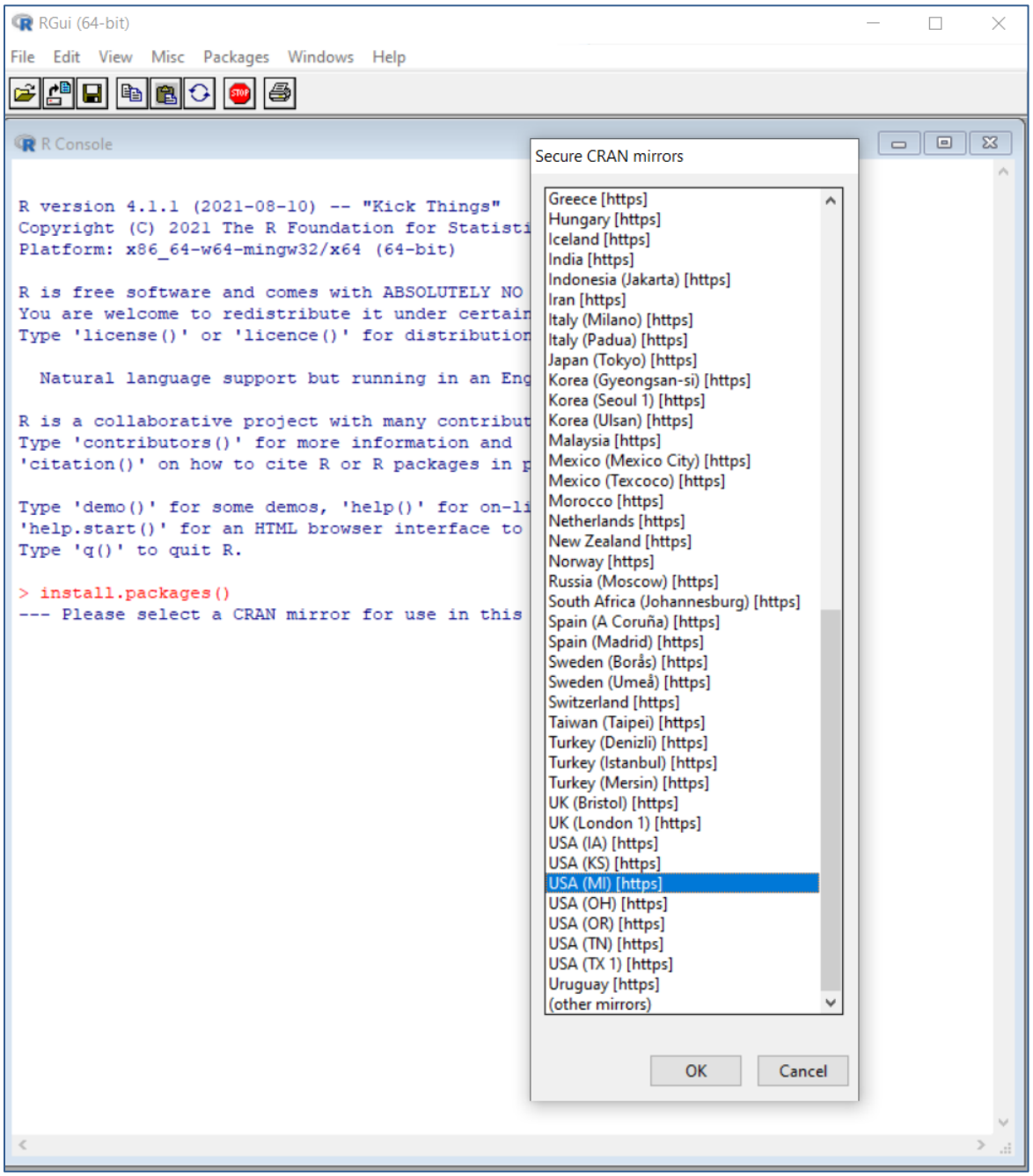
Available CRAN Packages By Name

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

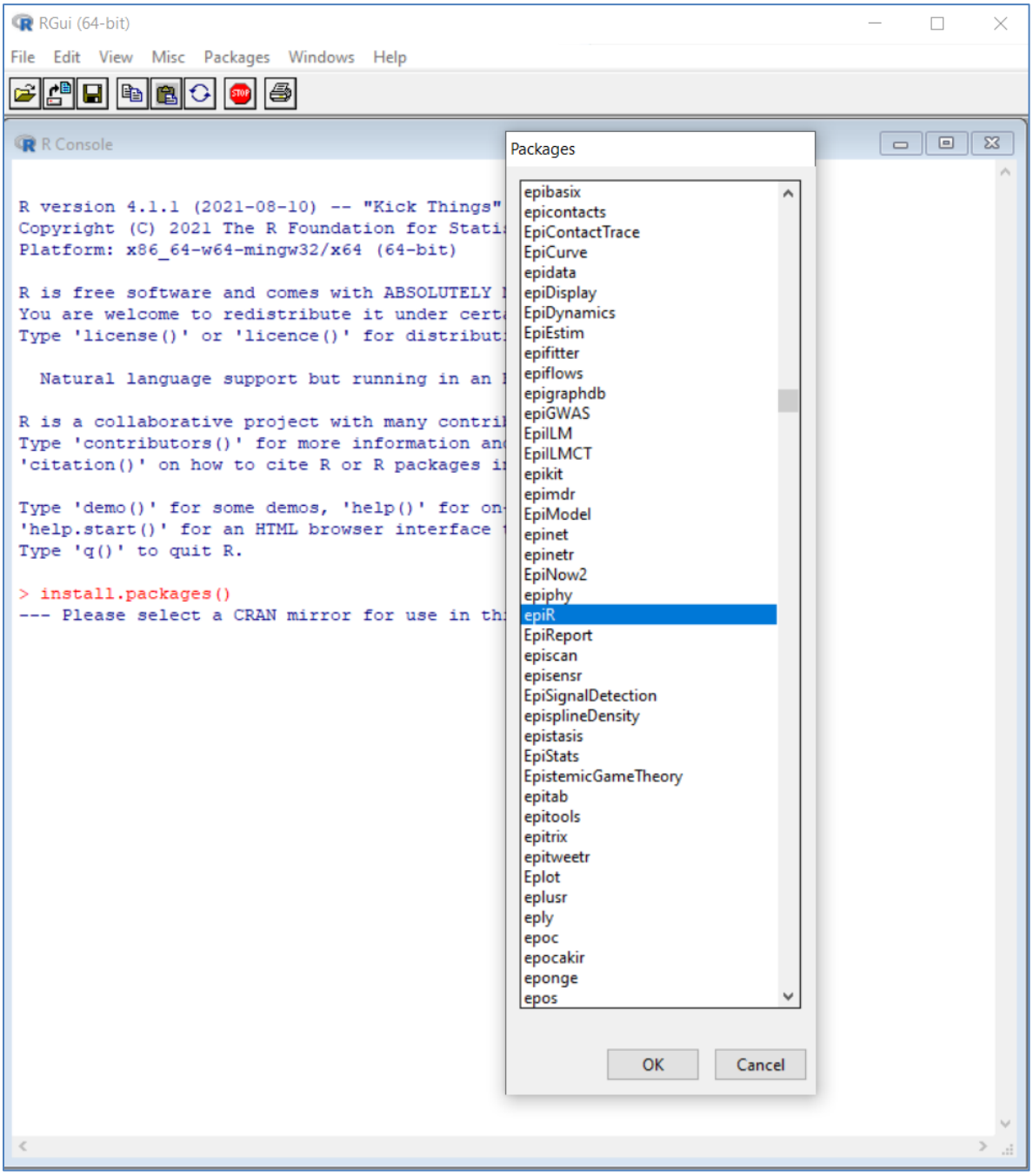
A3	Accurate, Adaptable, and Accessible Error Metrics for Predictive Models
aaSEA	Amino Acid Substitution Effect Analyser
AATtools	Reliability and Scoring Routines for the Approach-Avoidance Task
ABACUS	Apps Based Activities for Communicating and Understanding Statistics
abbyyR	Access to Abbyy Optical Character Recognition (OCR) API
abc	Tools for Approximate Bayesian Computation (ABC)
abc.data	Data Only: Tools for Approximate Bayesian Computation (ABC)
ABC.RAP	Array Based CpG Region Analysis Pipeline
abcADM	Fit Accumulated Damage Models and Estimate Reliability using ABC
ABCanalysis	Computed ABC Analysis
abcdeFBA	ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package
ABCoptim	Implementation of Artificial Bee Colony (ABC) Optimization
ABCp2	Approximate Bayesian Computational Model for Estimating P2
abcrf	Approximate Bayesian Computation via Random Forests
abcrlda	Asymptotically Bias-Corrected Regularized Linear Discriminant Analysis
abctools	Tools for ABC Analyses
abd	The Analysis of Biological Data
abdiv	Alpha and Beta Diversity Measures
abe	Augmented Backward Elimination
abess	Adaptive Best Subset Selection in Polynomial Time
abf2	Load Gap-Free Axon ABF2 Files
abglasso	Adaptive Bayesian Graphical Lasso
ABHgenotypeR	Easy Visualization of ABH Genotypes
abind	Combine Multidimensional Arrays
abjutils	Useful Tools for Jurimetrical Analysis Used by the Brazilian Jurimetrics Association
abn	Modelling Multivariate Data with Additive Bayesian Networks
abnormality	Measure a Subject's Abnormality with Respect to a Reference Population
abodOutlier	Angle-Based Outlier Detection
ABPS	The Abnormal Blood Profile Score to Detect Blood Doping
abstr	R Interface to the A/B Street Transport System Simulation Software
abstractr	An R-Shiny Application for Creating Visual Abstracts
abtest	Bayesian A/B Testing
Ac3net	Inferring Directional Conservative Causal Core Gene Networks
ACA	Abrupt Change-Point or Aberration Detection in Point Series
academictwitteR	Access the Twitter Academic Research Product Track V2 API Endpoint
acc	Exploring Accelerometer Data
accelerometry	Functions for Processing Accelerometer Data
accelmissing	Missing Value Imputation for Accelerometer Data

Obtaining a Non-Core Package

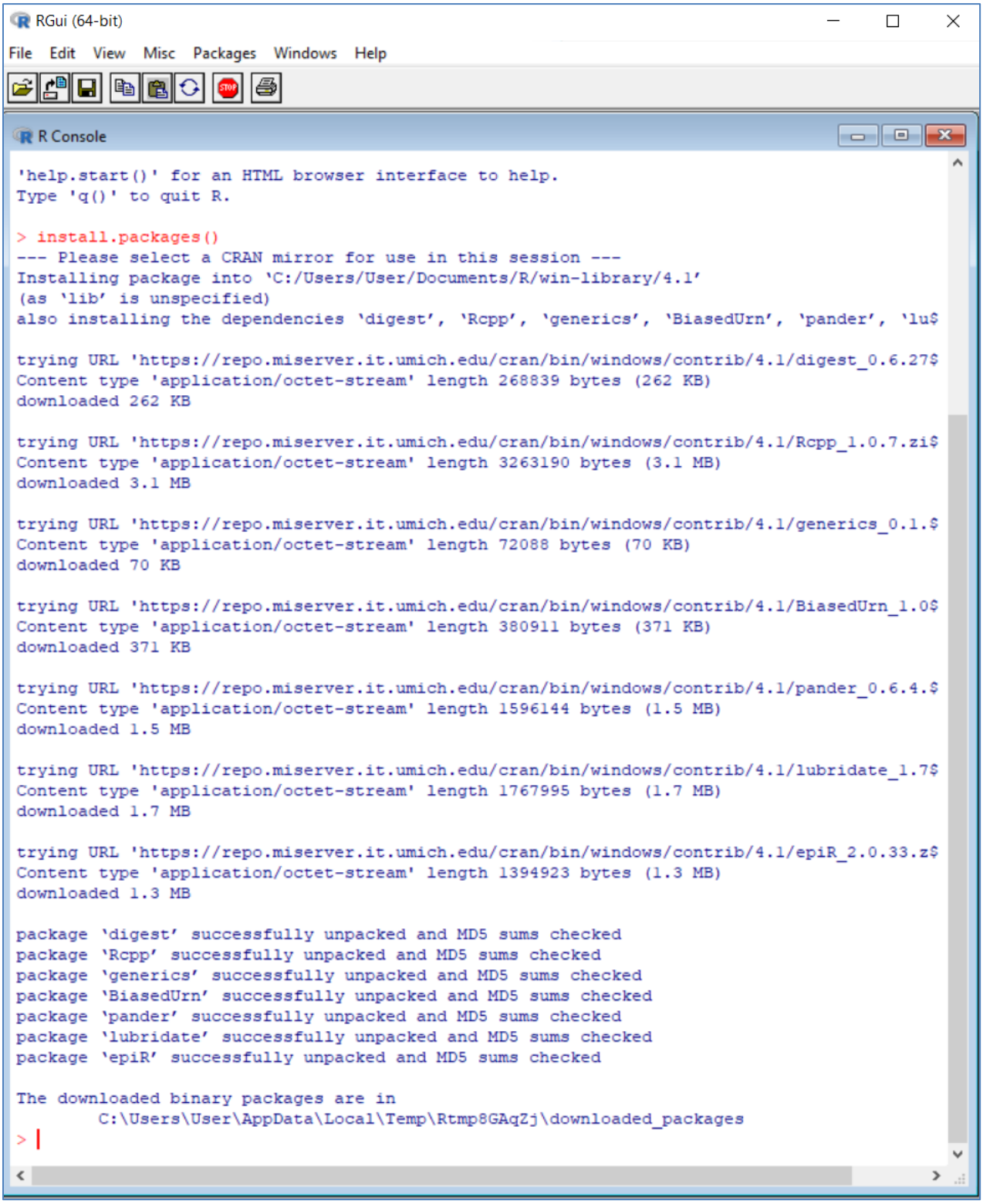
There are various ways to install packages in **R**. Perhaps the most straight-forward is using the 'install.packages()' function. When this function is entered, the user is first asked to choose a mirror (repository location). It is generally recommended that one chooses a nearby location.



After pressing OK, a list of all available packages (many thousands) will be shown.



Scroll to the desired package, click on the package name, and press OK.

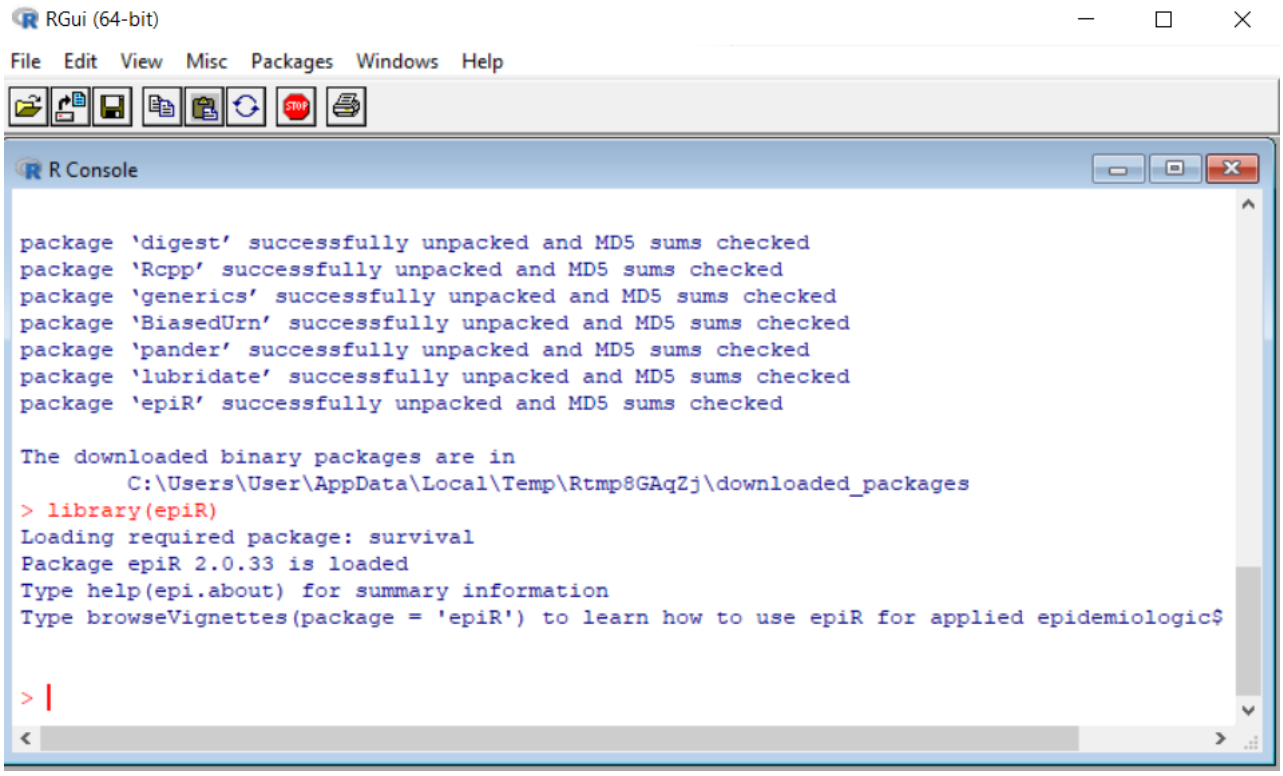


The package will proceed to download and unpack (install). Once the package has been downloaded, it will stay on the computer forever [unless it is removed using remove.packages("package name")].

If the package name is already known, an alternative is to use the function install.packages("package name"). However, the double quotes in this specification must be the double quotes that **R** recognizes.

Using a Non-Core Package

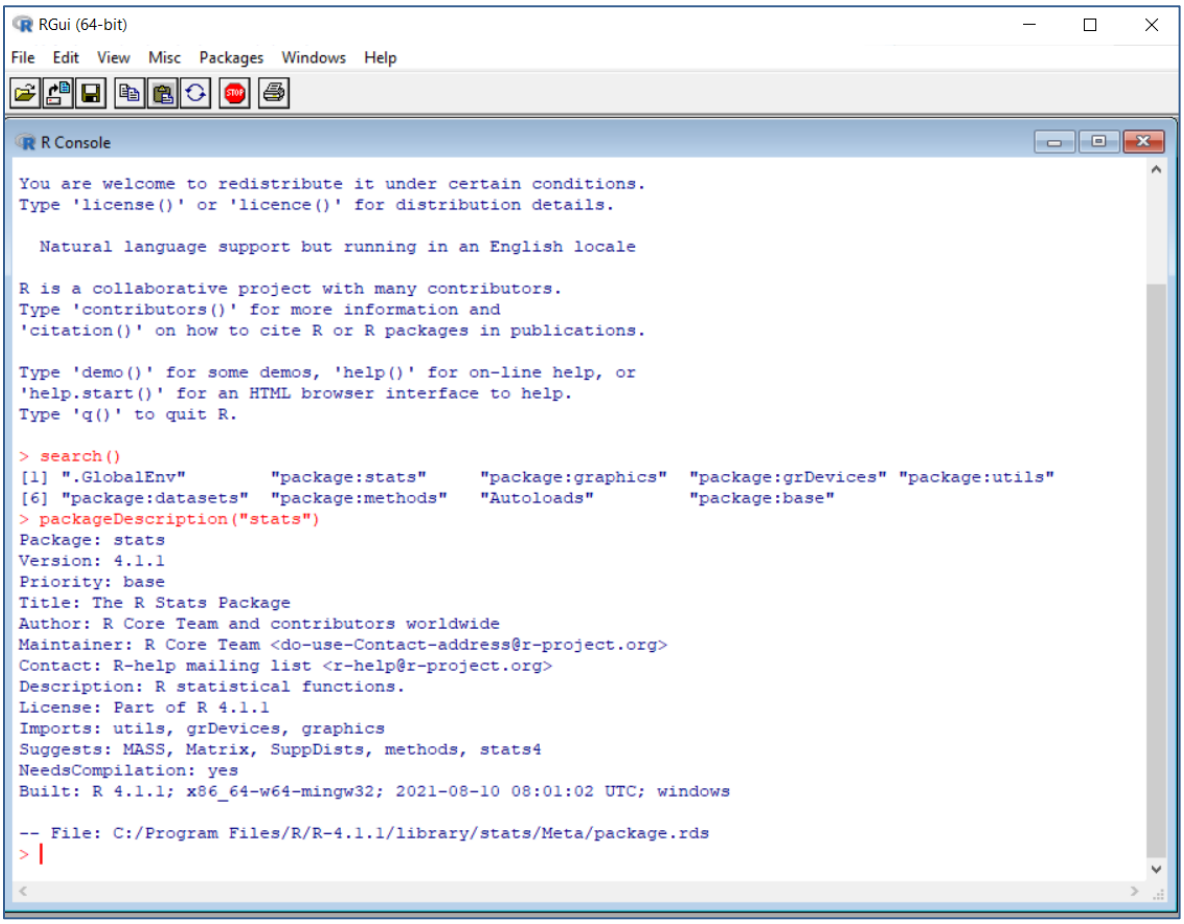
To use the package in a session of **R**, use the library(package name) command.



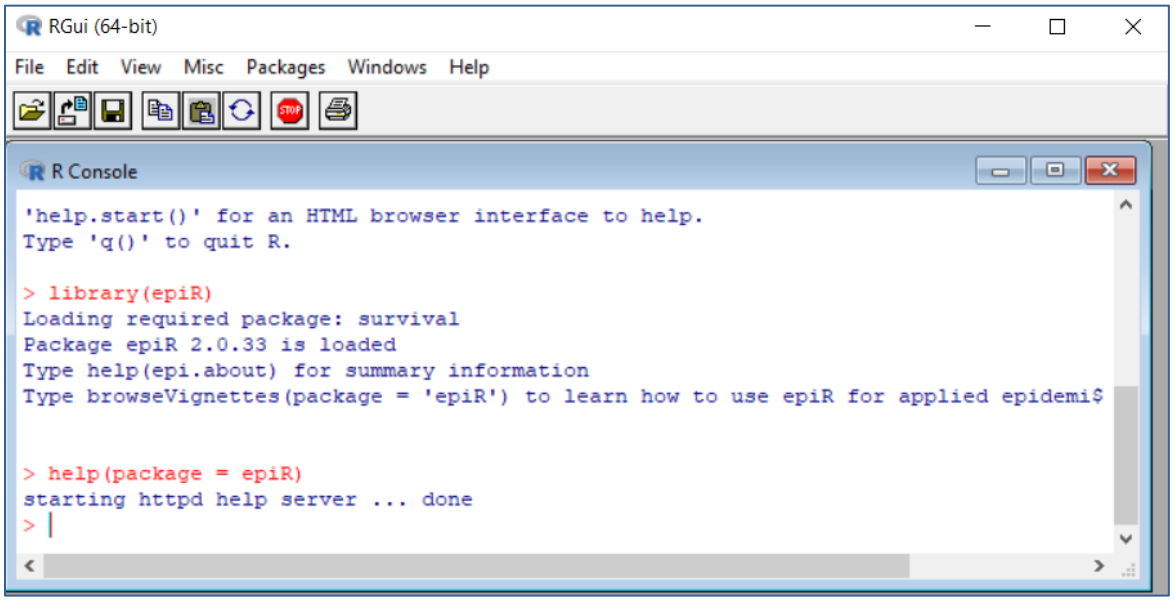
All the libraries will disappear each time an **R** session is ended. The library(package name) function will need to be called each time a new session of **R** is started, if the functions of the package are to be used in that session.

Package Descriptions and Help

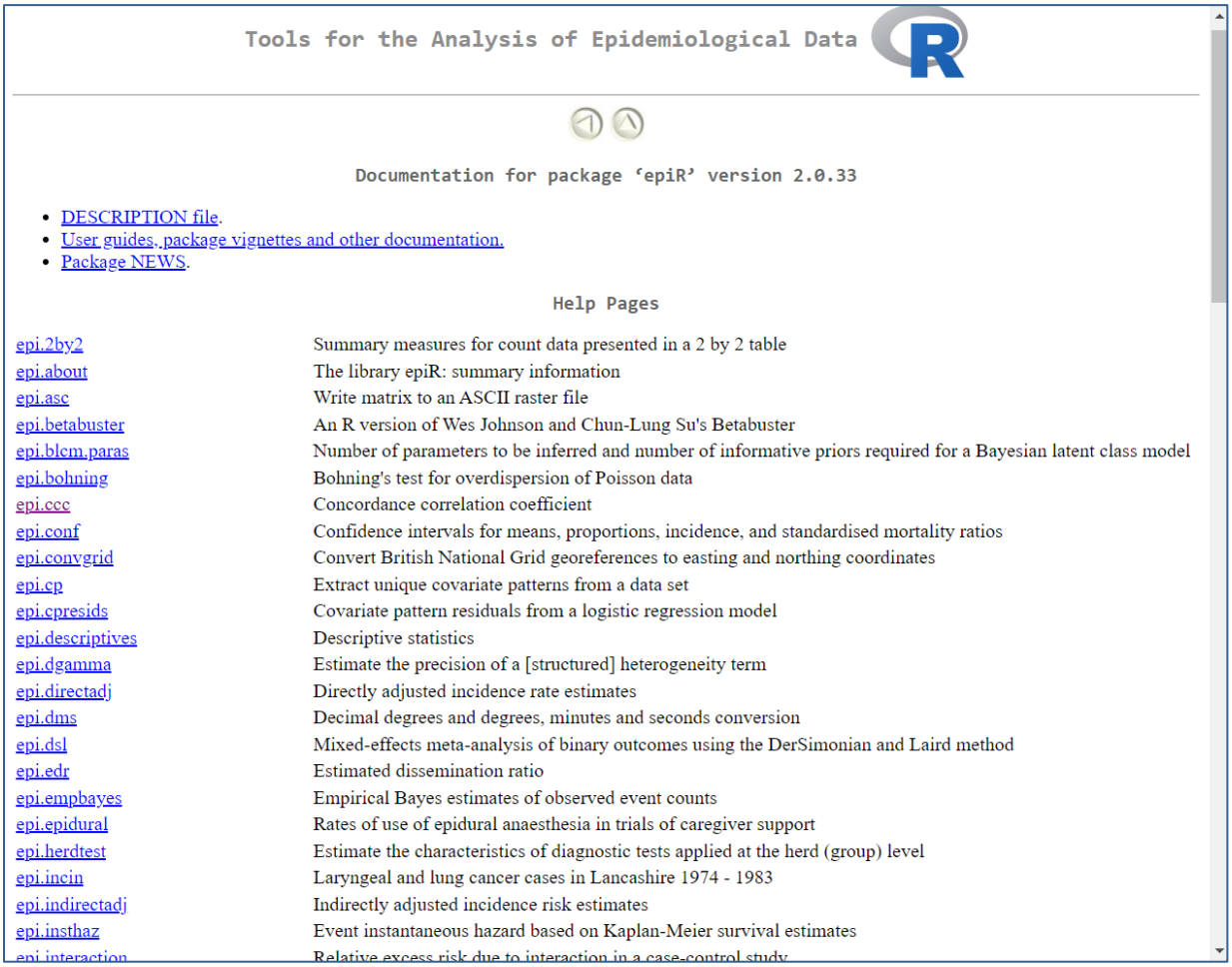
In the **R** program, the 'search()' command can be used to view the currently loaded packages in the current session of **R**. The 'packageDescription("package")' function can be used to get a description of any of the currently-loaded packages.



The help documentation for each package is accessed using the help(package = package name) command.



The documentation information is opened in your web browser:



Tools for the Analysis of Epidemiological Data

Documentation for package 'epiR' version 2.0.33

- DESCRIPTION file.
- User guides, package vignettes and other documentation.
- Package NEWS.

Help Pages

epi.2by2	Summary measures for count data presented in a 2 by 2 table
epi.about	The library epiR: summary information
epi.asc	Write matrix to an ASCII raster file
epi.betabuster	An R version of Wes Johnson and Chun-Lung Su's Betabuster
epi.blcm.paras	Number of parameters to be inferred and number of informative priors required for a Bayesian latent class model
epi.bohning	Bohning's test for overdispersion of Poisson data
epi.ccc	Concordance correlation coefficient
epi.conf	Confidence intervals for means, proportions, incidence, and standardised mortality ratios
epi.convgrid	Convert British National Grid georeferences to easting and northing coordinates
epi.cp	Extract unique covariate patterns from a data set
epi.cpresids	Covariate pattern residuals from a logistic regression model
epi.descriptives	Descriptive statistics
epi.dgamma	Estimate the precision of a [structured] heterogeneity term
epi.directadj	Directly adjusted incidence rate estimates
epi.dms	Decimal degrees and degrees, minutes and seconds conversion
epi.dsl	Mixed-effects meta-analysis of binary outcomes using the DerSimonian and Laird method
epi.edr	Estimated dissemination ratio
epi.empbayes	Empirical Bayes estimates of observed event counts
epi.epidural	Rates of use of epidural anaesthesia in trials of caregiver support
epi.herdtest	Estimate the characteristics of diagnostic tests applied at the herd (group) level
epi.incin	Laryngeal and lung cancer cases in Lancashire 1974 - 1983
epi.indirectadj	Indirectly adjusted incidence risk estimates
epi.insthaz	Event instantaneous hazard based on Kaplan-Meier survival estimates
epi.interaction	Relative excess risk due to interaction in a case-control study

Clicking on an individual function gives the **R** documentation for that function:

epi.ccc {epiR} R Documentation

Concordance correlation coefficient

Description

Calculates Lin's (1989, 2000) concordance correlation coefficient for agreement on a continuous measure.

Usage

```
epi.ccc(x, y, ci = "z-transform", conf.level = 0.95, rep.measure = FALSE,
   subjectid)
```

Arguments

`x`
: a vector, representing the first set of measurements.

`y`
: a vector, representing the second set of measurements.

`ci`
: a character string, indicating the method to be used. Options are `z-transform` or `asymptotic`.

`conf.level`
: magnitude of the returned confidence interval. Must be a single number between 0 and 1.

`rep.measure`
: logical. If `TRUE` there are repeated observations across `subject`.

`subjectid`
: a factor providing details of the observer identifier if `rep.measure == TRUE`.

Details

Computes Lin's (1989, 2000) concordance correlation coefficient for agreement on a continuous measure obtained by two methods. The concordance correlation coefficient combines measures of both precision and accuracy to determine how far the observed data deviate from the line of perfect concordance (that is, the line at 45 degrees on a square scatter plot). Lin's coefficient increases in value as a function of the nearness of the data's reduced major axis to the line of perfect concordance (the accuracy of the data) and of the tightness of the data about its reduced major axis (the precision of the data).

Both `x` and `y` values need to be present for a measurement pair to be included in the analysis. If either or both values are missing (i.e. coded `NA`) then the measurement pair is deleted before analysis.

Value

A list containing the following:

`rho.c`
: the concordance correlation coefficient.

`s.shift`
: the scale shift.

`l.shift`
: the location shift.

`C.b`
: a bias correction factor that measures how far the best-fit line deviates from a line at 45 degrees. No deviation from the 45 degree line occurs when C.b = 1. See Lin (1989, page 258).

Example 1 – Exporting and Reading in Data

This section presents an example of exporting an **NCSS** dataset to a .CSV file, and then reading the .CSV file into **R**. The newly imported data table is examined using some **R** functions that are output in **NCSS**.

Setup

To run this example, complete the following steps:

1 Open the Resale example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **Resale** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 1** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\Resale.CSV** (The 'R Data' folder will need to be created, if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

- Click the **Run** button to perform the calculations and generate the output.

Summary

This section lets the user know the name and path of the file that was created.

Summary
A CSV file was created from the NCSS data sheet and was stored at C:\R Data\Resale.CSV

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
Resale_DataFrame<-read.csv("C:\\R Data\\Resale.CSV")
attach(Resale_DataFrame)
```

These two lines of **R** code are sufficient to read in the newly created Resale.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.

R Code for Reading in the CSV Data File (With Notes)

R Code for Reading in the CSV Data File (With Notes)

```
############## Code to Read in the CSV Data File and Make it Ready for Use ###############
### Note: Any line that begins with # is ignored by R.

### Read in the Resale.CSV file as a new data frame named Resale_DataFrame
Resale_DataFrame<-read.csv("C:\\R Data\\Resale.CSV")

### Tell R to allow the user to reference the column names without having to
### reference the data frame with every column name
### Note: Make sure that the column names are not already in use
attach(Resale_DataFrame)

#################################################################################
```

These lines of **R** code produce the same exact result as the two lines of code of the previous section. The entire section may be copied and pasted into the **R** Console. All lines that begin with '#' are ignored by **R**.

R Code for Examining the Data Frame

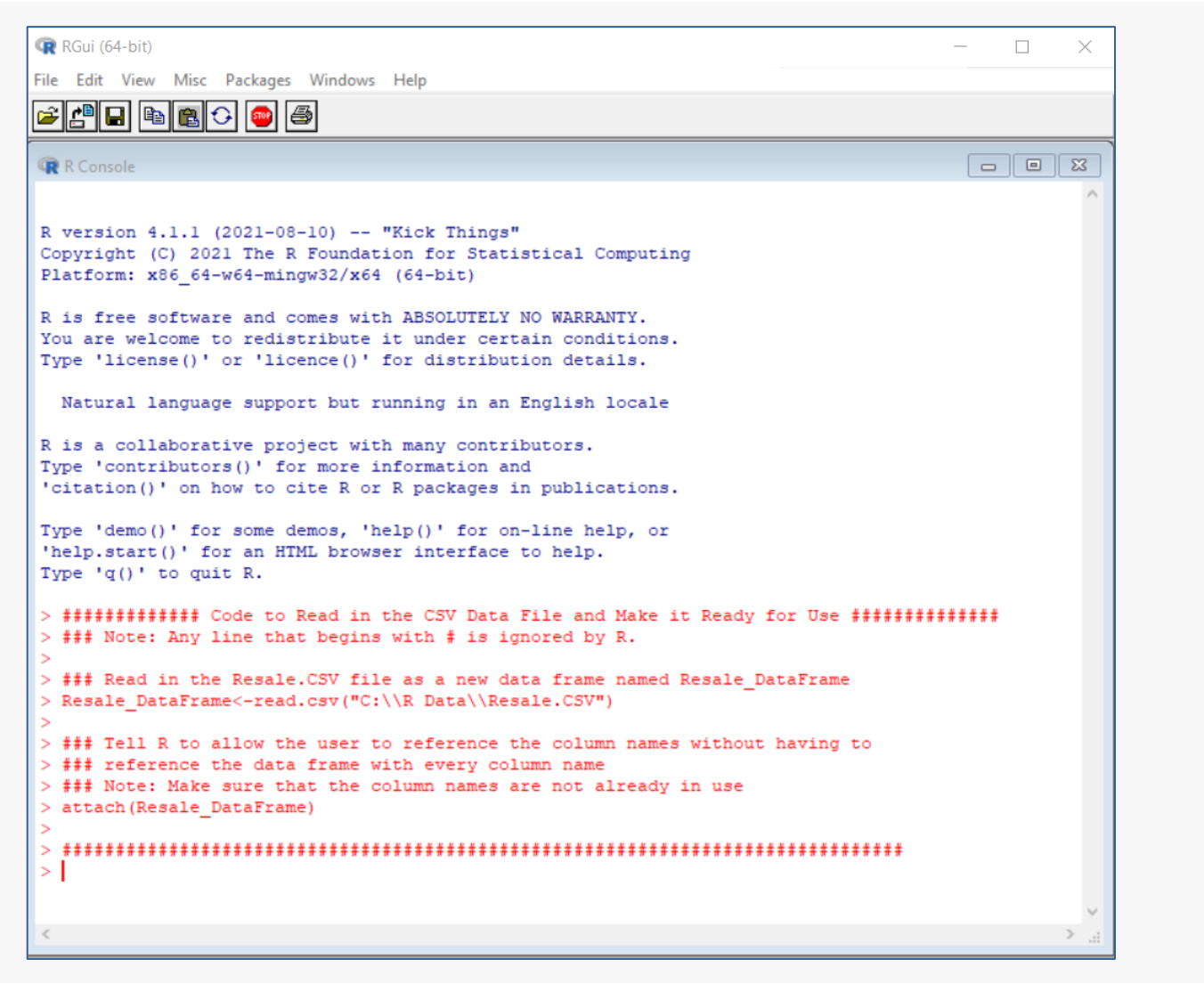
R Code for Examining the Data Frame

```
##################### Functions to Review the Data Frame ########################
### These functions are not required. They are for information purposes only.
### Each of these functions can be used independently.

### Give a high-level summary of each column
summary(Resale_DataFrame)

### List the column names
colnames(Resale_DataFrame)

### List the first 5 rows of each column
head(Resale_DataFrame, n = 5)

### List the last 5 rows of each column
tail(Resale_DataFrame, n = 5)

### List all the data in a separate viewing table (View must have a capital V)
View(Resale_DataFrame)

### List all the data, using the R Console
Resale_DataFrame

### Examine the dimensions of the data frame
dim(Resale_DataFrame)

### Examine the column structure
str(Resale_DataFrame)

################################################################################
```

Any or all of these lines of **R** code may be copied and pasted into the **R** Console to generate the corresponding summary or description. Any lines that begin with '#' are ignored by **R**.

Creating the Data Frame in R

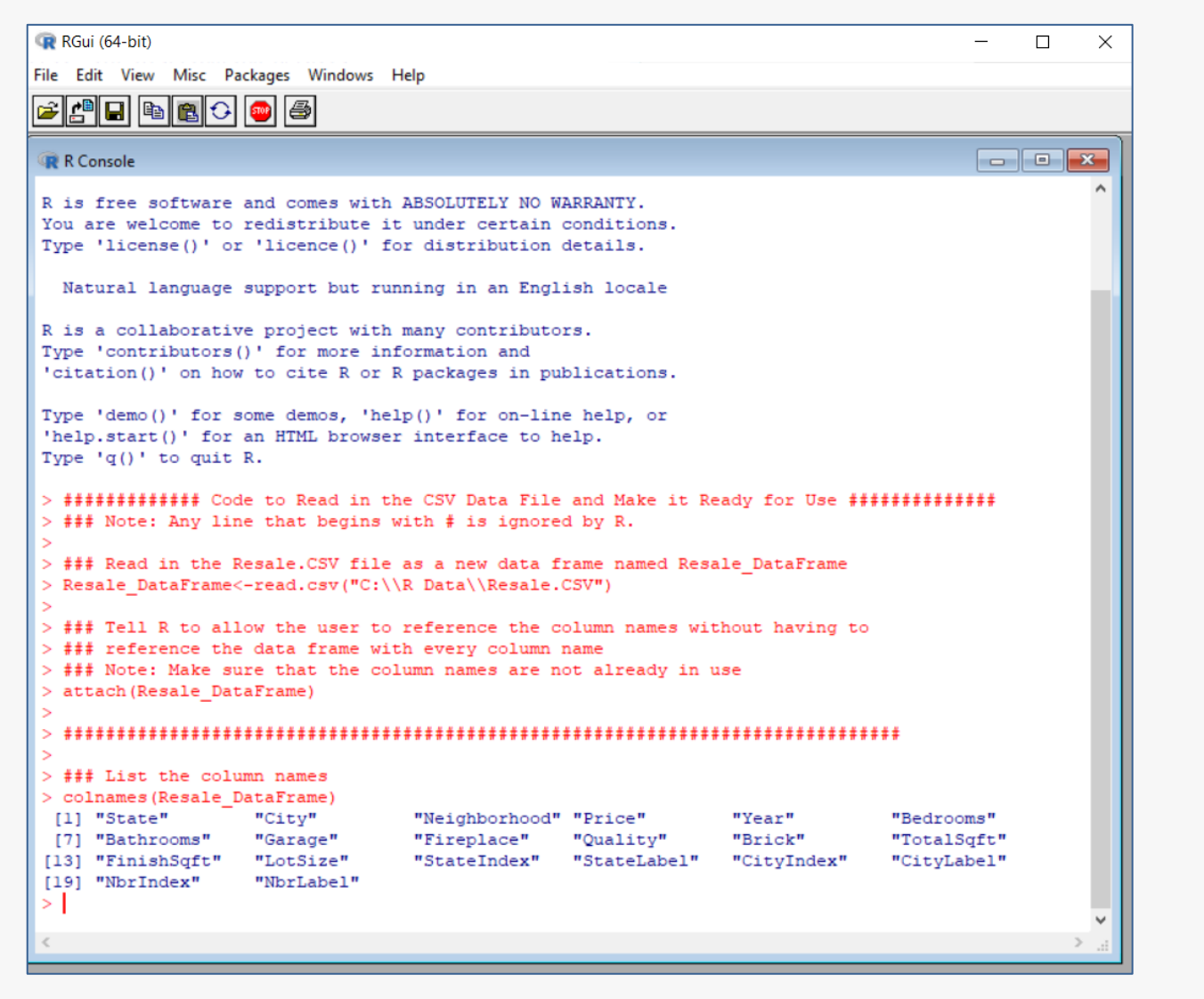
The dataset is read into **R** by copying and pasting either of the first two code sections of the **NCSS** output. The second section is copied and pasted in this example.



The Resale_DataFrame data frame has now been created and contains the Resale.CSV data. The 'attach' command allows the use of the columns by their previous column names.

Listing the Column Names

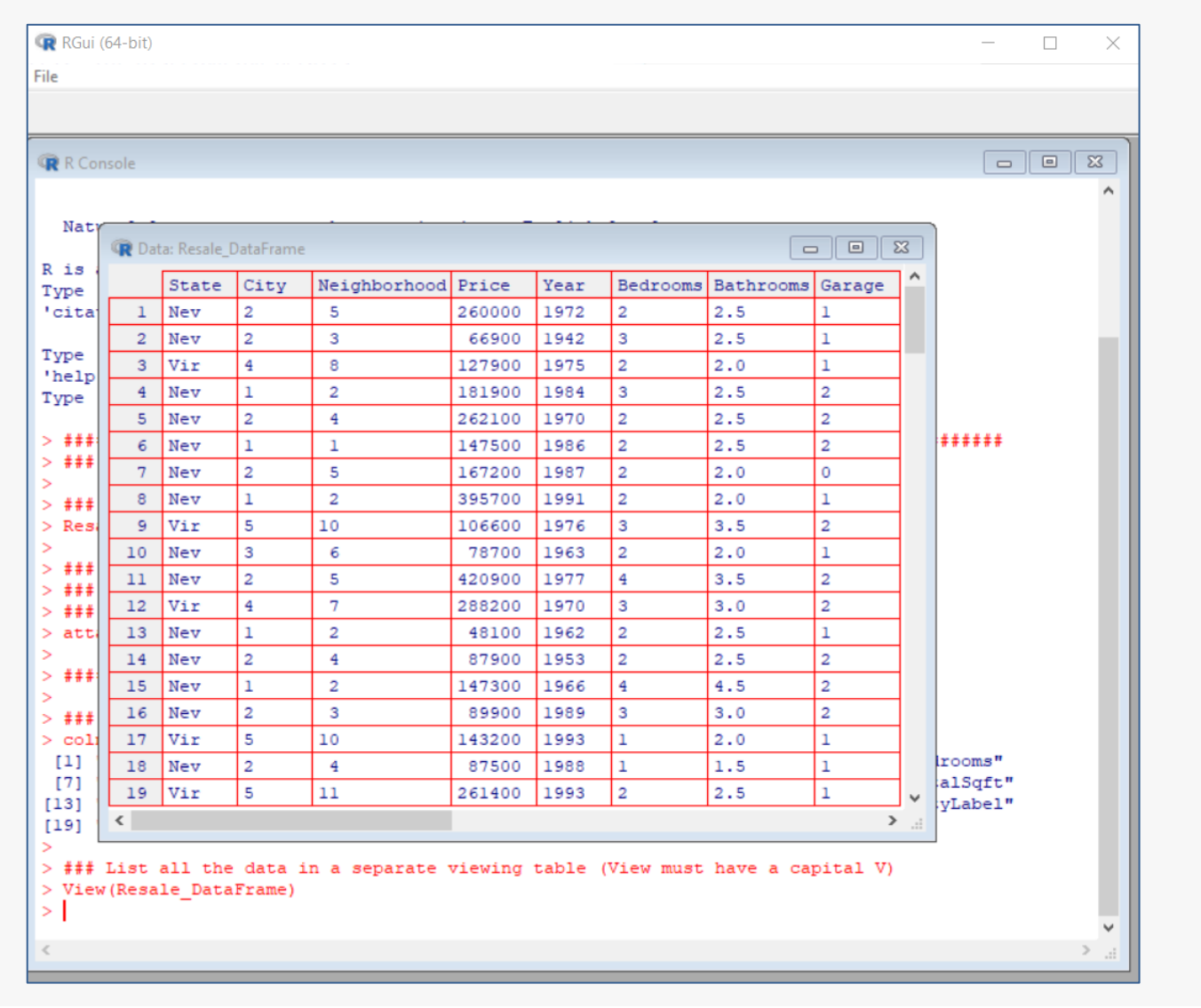
The 'colnames' function can be copied and pasted into **R** to view the column names.



Several other functions can be used to examine the details of the Resale_DataFrame data frame.

Viewing the Data Table in an R Table

The 'View' function can be copied and pasted into **R** to give a table for viewing the data.



	State	City	Neighborhood	Price	Year	Bedrooms	Bathrooms	Garage
1	Nev	2	5	260000	1972	2	2.5	1
2	Nev	2	3	66900	1942	3	2.5	1
3	Vir	4	8	127900	1975	2	2.0	1
4	Nev	1	2	181900	1984	3	2.5	2
5	Nev	2	4	262100	1970	2	2.5	2
6	Nev	1	1	147500	1986	2	2.5	2
7	Nev	2	5	167200	1987	2	2.0	0
8	Nev	1	2	395700	1991	2	2.0	1
9	Vir	5	10	106600	1976	3	3.5	2
10	Nev	3	6	78700	1963	2	2.0	1
11	Nev	2	5	420900	1977	4	3.5	2
12	Vir	4	7	288200	1970	3	3.0	2
13	Nev	1	2	48100	1962	2	2.5	1
14	Nev	2	4	87900	1953	2	2.5	2
15	Nev	1	2	147300	1966	4	4.5	2
16	Nev	2	3	89900	1989	3	3.0	2
17	Vir	5	10	143200	1993	1	2.0	1
18	Nev	2	4	87500	1988	1	1.5	1
19	Vir	5	11	261400	1993	2	2.5	1

This table can be used only for viewing the data. It cannot be used to edit the data or perform other functions.

Example 2 – Running Some Basic Functions in R

Once the Resale data has been imported into **R** and set up as a data frame, individual **R** functions may be used to analyze or visualize the data.

Setup

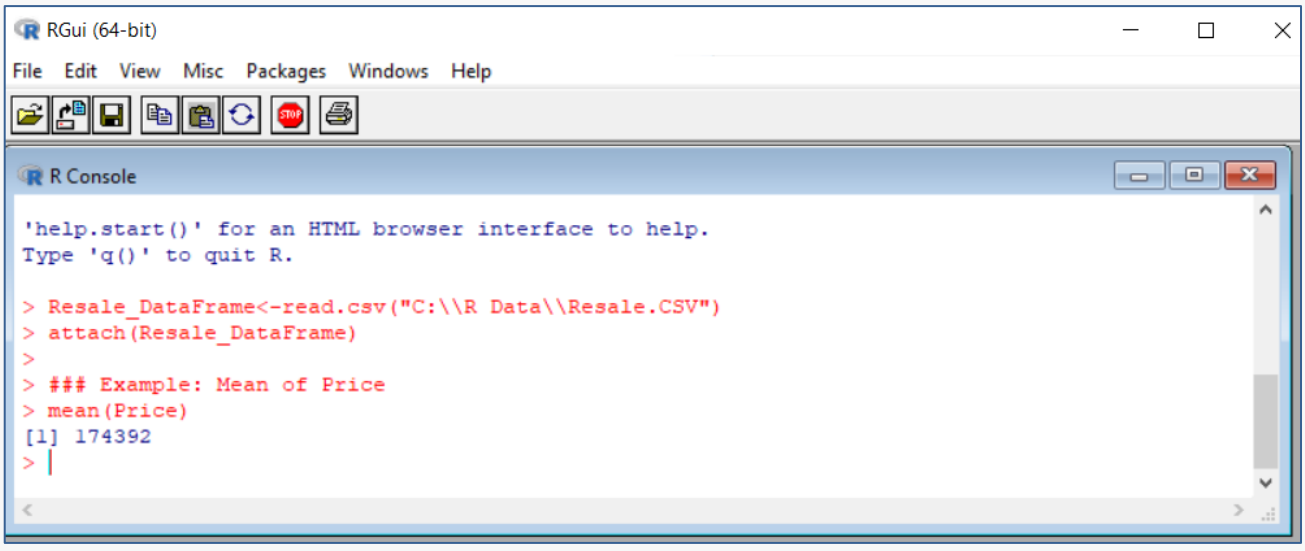
To run this example, complete the following steps:

1 **Export and Read in the data using the steps of Example 1.**

2 **Type, or copy and paste, one section at a time, the following R commands, that are included with the base functions of R.**

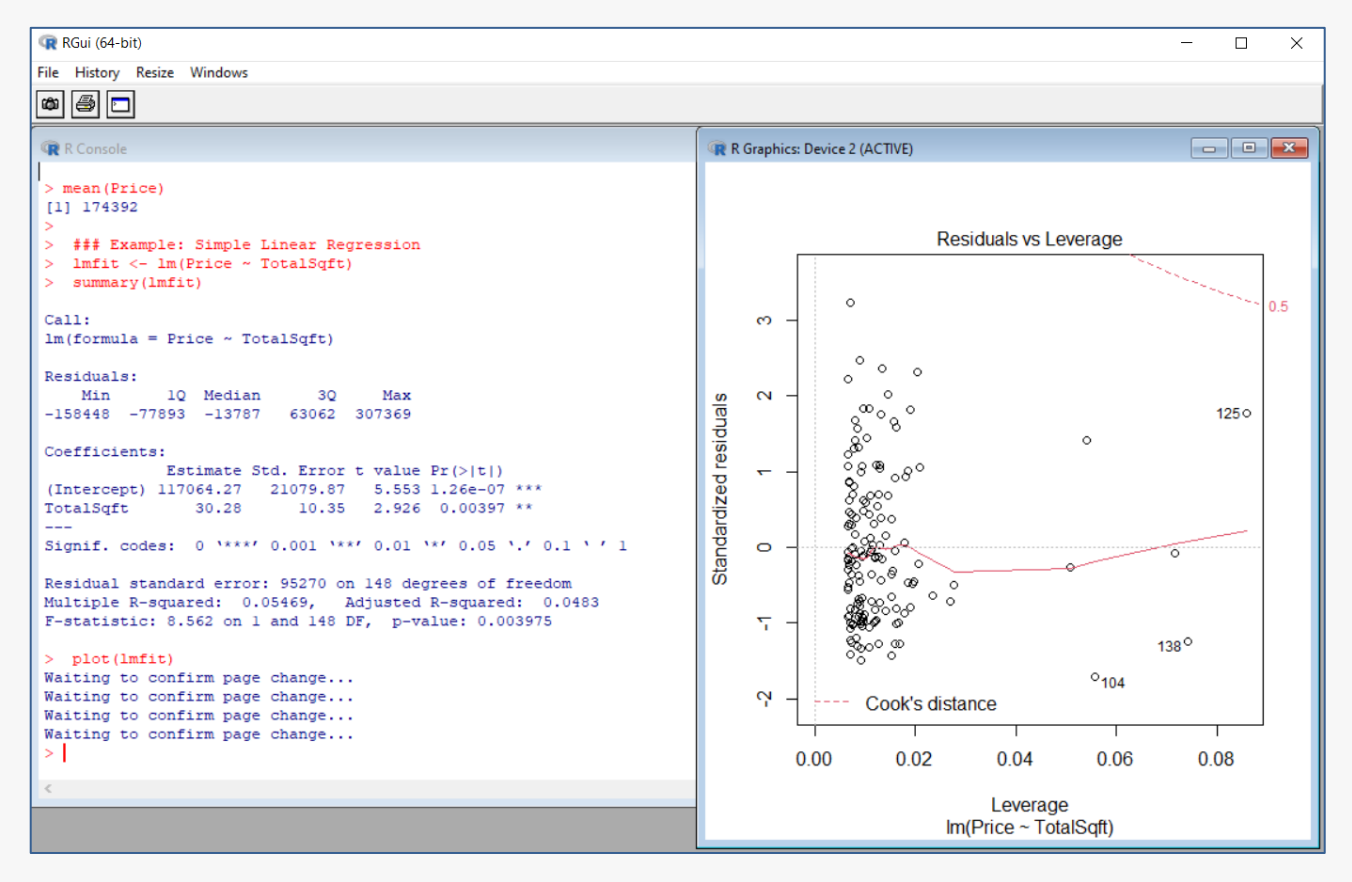
```
### Example: Mean of Price
mean(Price)

 ### Example: Simple Linear Regression
 lmfit <- lm(Price ~ TotalSqft)
 summary(lmfit)
 plot(lmfit)

  ### Example: Simple Scatter Plot
  plot(TotalSqft, Price)
```

R Output – Mean of Price



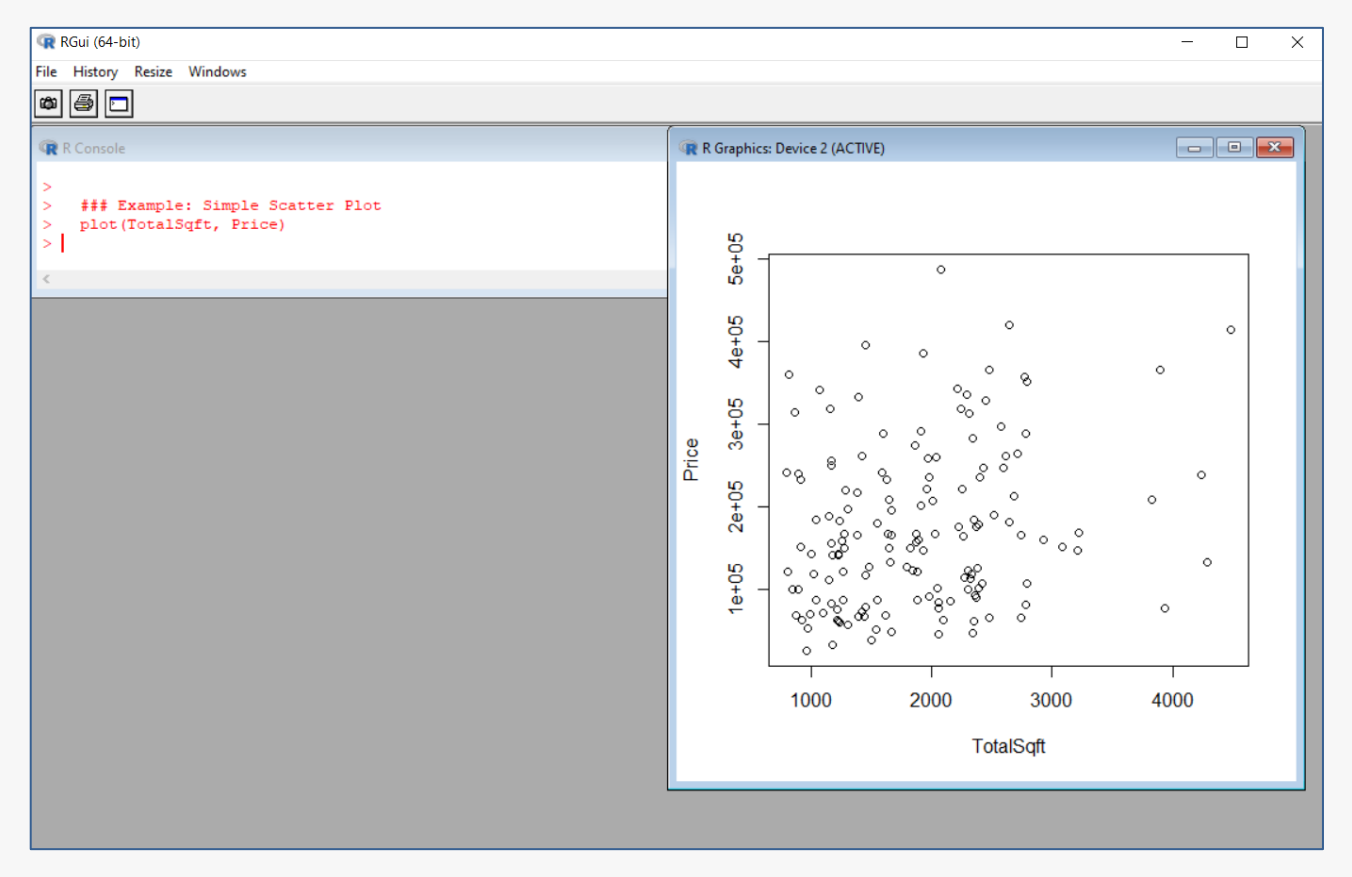
The mean of the Price column is given.

R Output – Simple Linear Regression



The plots are viewed one by one by pressing the Enter key.

R Output – Simple Scatter Plot



There are many, many options that could be used to adjust the appearance of the plot in **R**, but that is outside the scope of this example.

Example 3 – Cochran-Armitage Trend Test (Exact)

The Contingency Tables procedure in **NCSS** has three available Cochran-Armitage Trend tests, but it does not offer the exact Cochran-Armitage Trend test. This example shows the process by which an **R** package can be used to obtain the p-value for the exact test.

The format of the data needed for running the trend test in **NCSS** is (see Armitage example data):

Rating	Category	Count
1	Carriers	19
2	Carriers	29
3	Carriers	24
1	Non-carriers	497
2	Non-carriers	560
3	Non-carriers	269

The format needed (for the same values) in the R package is (see Armitage Total example data):

Rating	Cases	Total
1	19	516
2	29	589
3	24	293

Setup

To run this example, complete the following steps:

1 Open the Armitage Total example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **Armitage Total** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 3** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab

Rows to Export ..... **Export All Rows**
Columns to Export ..... **Export All Columns**
Path and Name..... **C:\R Data\ArmitageTotal.CSV** (The “R Data” folder will need to be created if it is not already.)
R Code - Without Notes..... **Checked**
R Code - With Notes..... **Checked**
R Code - Summary Functions..... **Checked**

3 Run the procedure

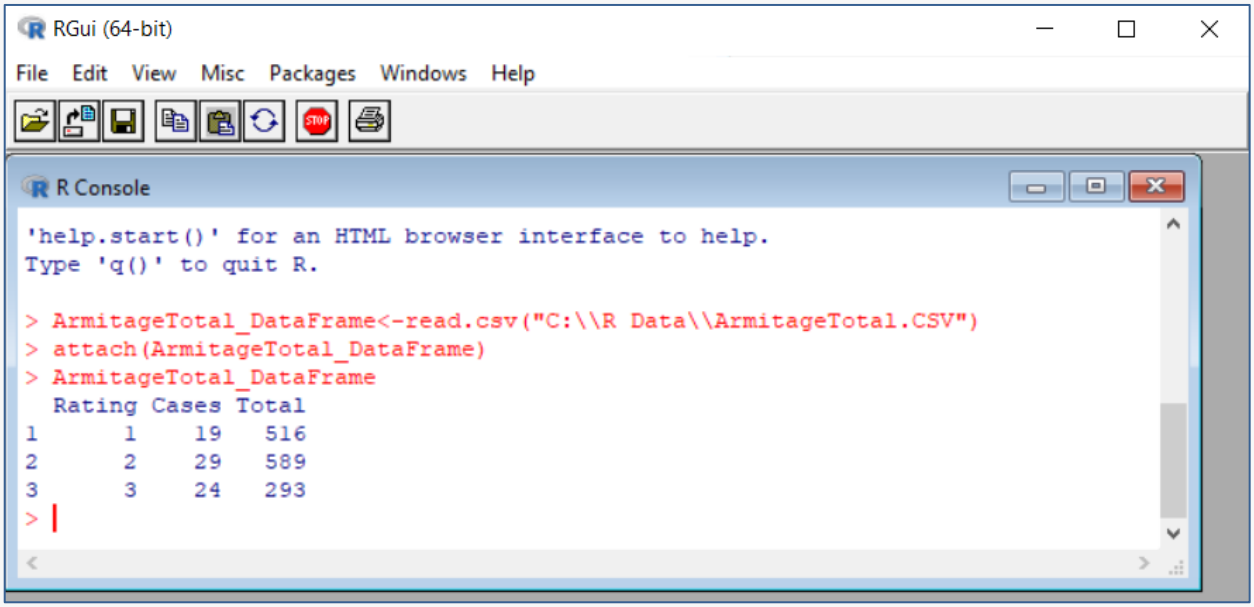
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
ArmitageTotal_DataFrame<-read.csv("C:\\R Data\\ArmitageTotal.CSV")
attach(ArmitageTotal_DataFrame)
```

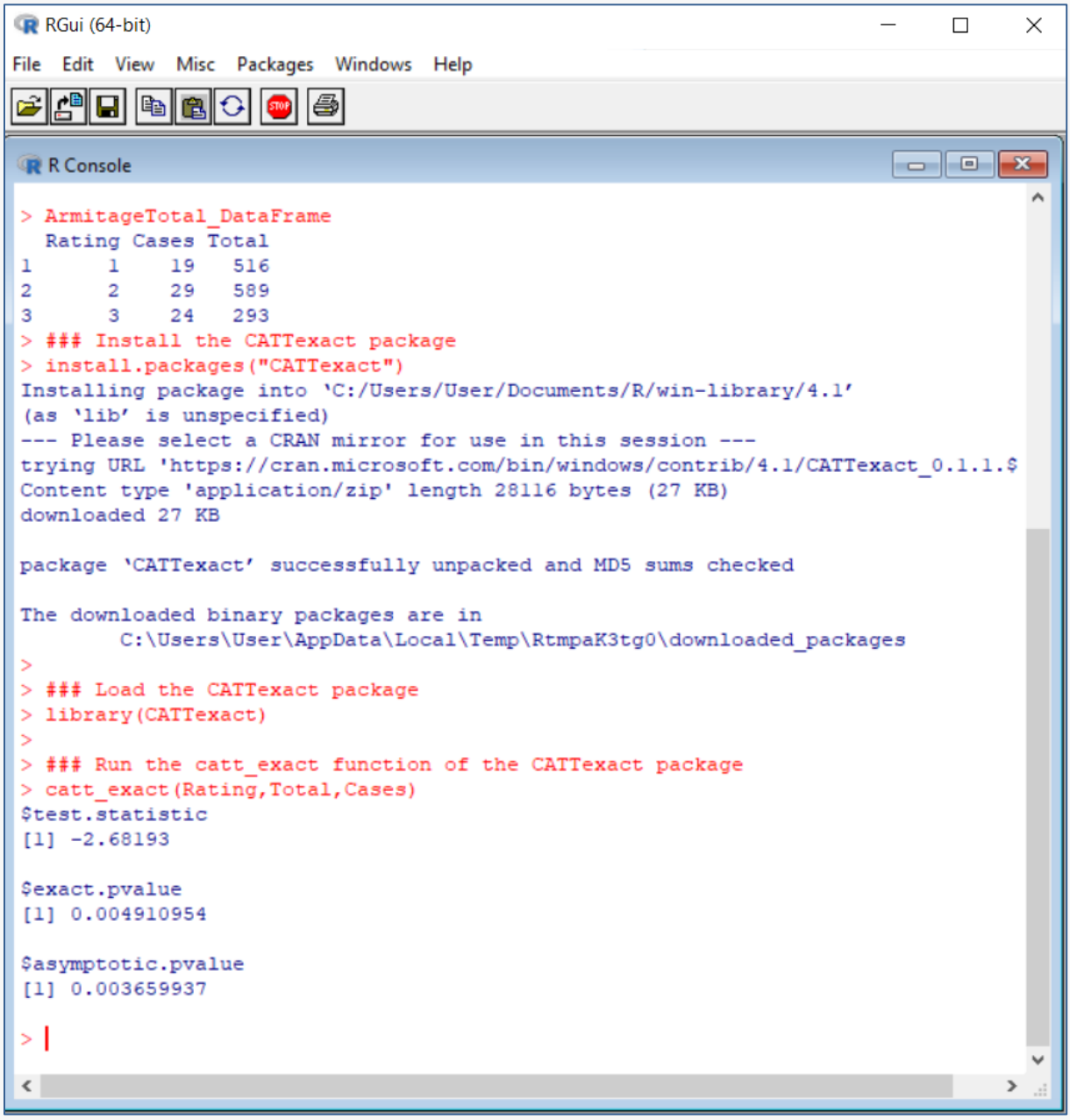
These two lines of **R** code are sufficient to read in the newly created ArmitageTotal.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for performing the Exact Cochran-Armitage Trend Test

```
### Install the CATTexact package
### If the CATTexact package has been installed previously, this line may be skipped
install.packages("CATTexact")

### Load the CATTexact package
library(CATTexact)

### Run the catt_exact function of the CATTexact package
catt_exact(Rating,Total,Cases)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



The test statistic value of -2.68193 matches that given by the **NCSS** Contingency Tables procedure example exactly. The asymptotic p-value also matches the **NCSS** example value. The exact p-value, which is not available in NCSS, is shown to be 0.004910954.

Example 4 – Quade Test for Unreplicated Complete Block Designs

The Quade test is a non-parametric rank test for analyzing unreplicated complete block designs where the response is not normal. The Quade test is not currently available in the **NCSS** procedures. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and performing the Quade test in **R**.

Suppose four different X-ray machines are compared for easy-of-use. Thirty-eight (38) technicians are asked to rate each of the four machines on a 10-point Likert scale. The result of the ratings is given in the XRay dataset.

Setup

To run this example, complete the following steps:

1 Open the XRay example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **XRay** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 4** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\XRay.CSV** (The “R Data” folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

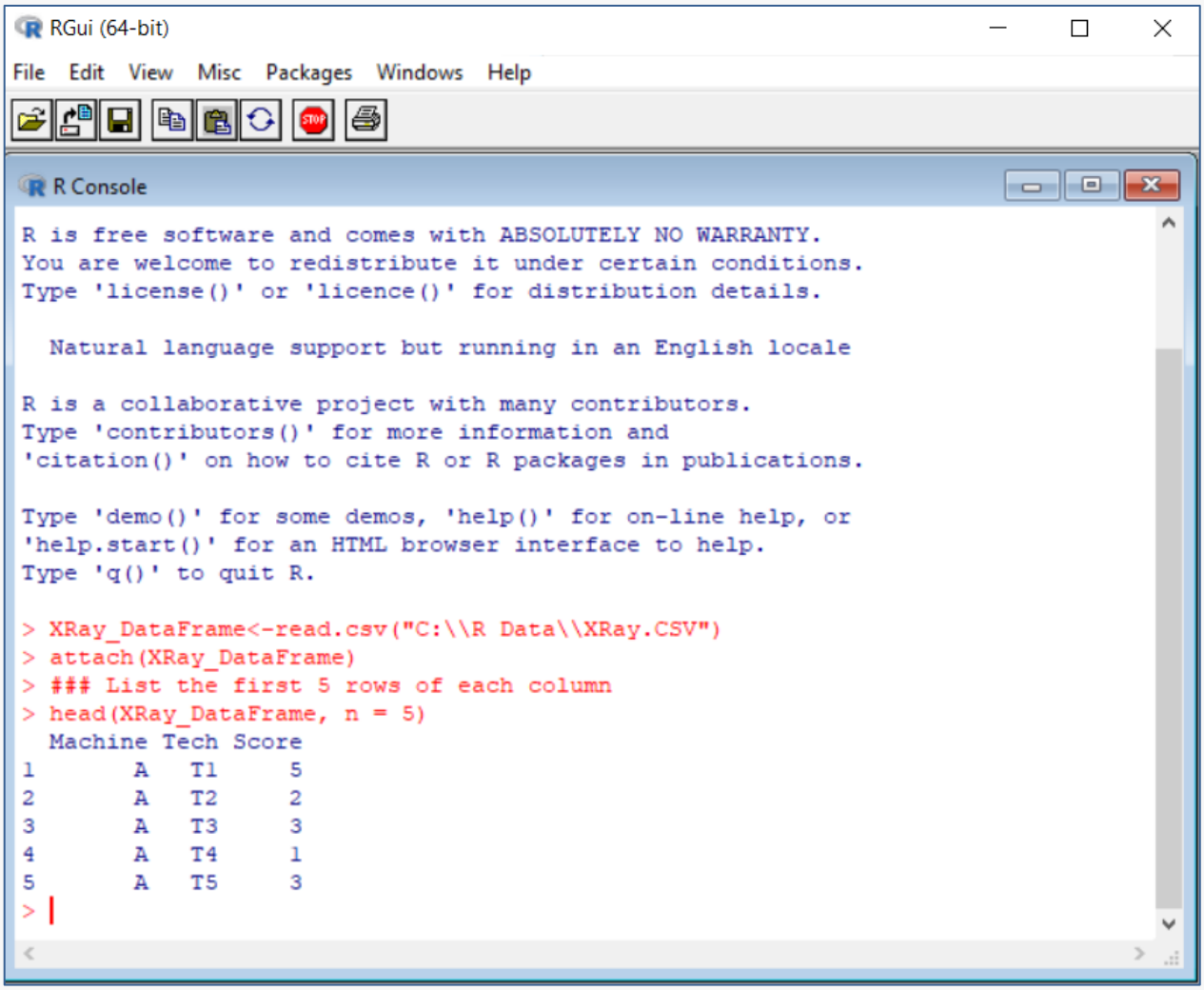
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
XRay_DataFrame<-read.csv("C:\\R Data\\XRay.CSV")
attach(XRay_DataFrame)
```

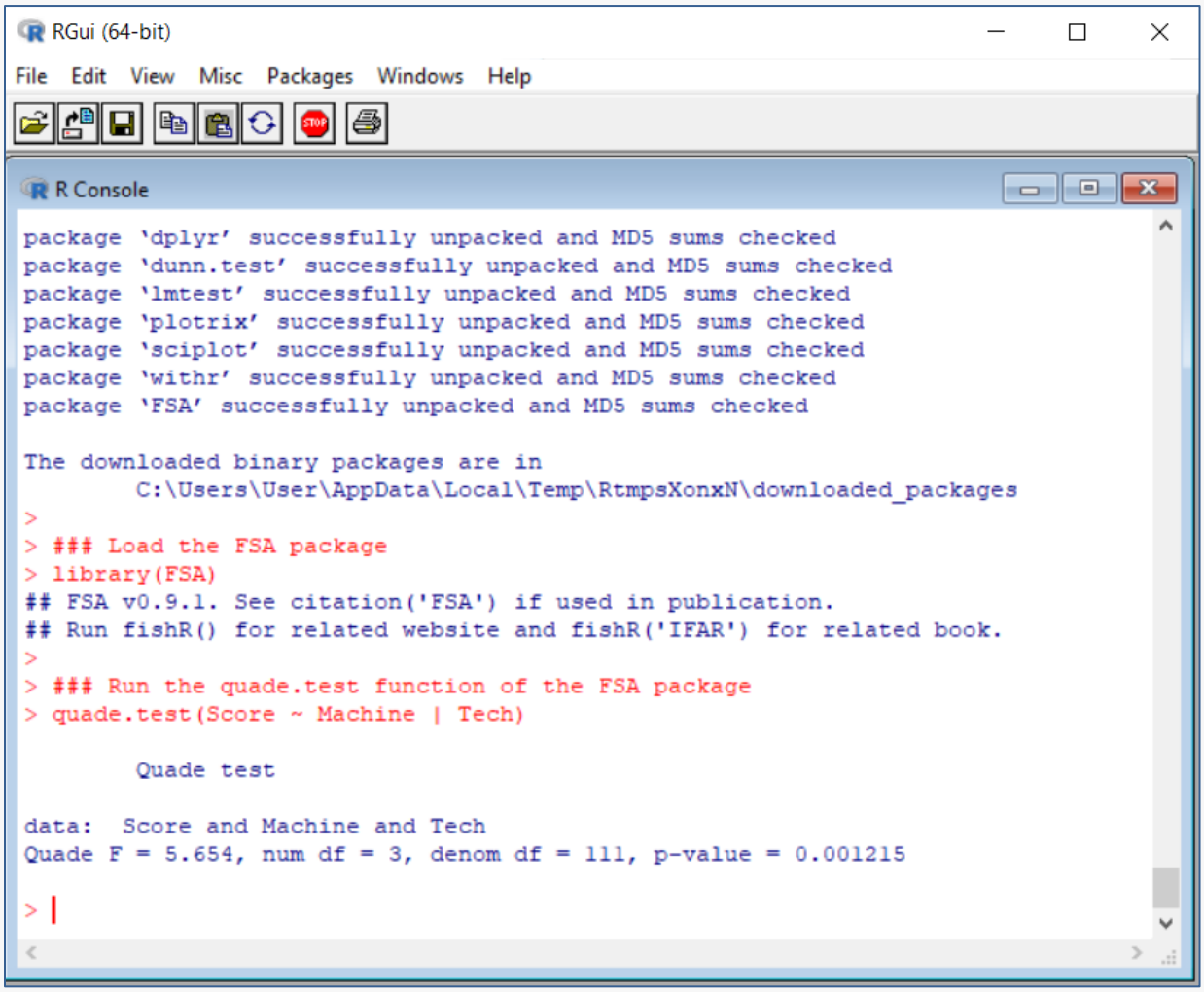
These two lines of **R** code are sufficient to read in the newly created XRay.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for Performing the Quade Test

```
### Install the FSA package
### If the FSA package has been installed previously, this line may be skipped
install.packages("FSA")

### Load the FSA package
library(FSA)

### Run the quade.test function of the FSA package
quade.test(Score ~ Machine | Tech)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



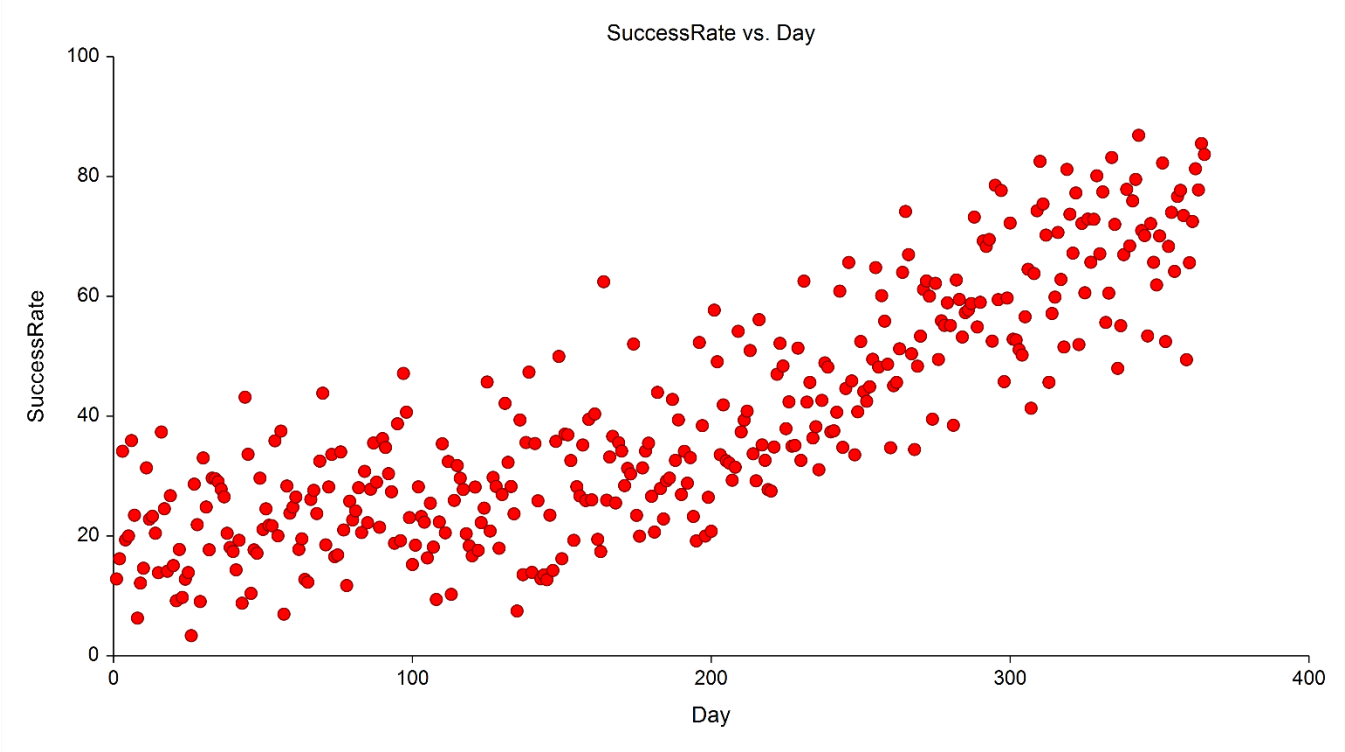
The p-value of 0.001215 indicates evidence of a difference in ease-of-use among the four machines. Additional functions, such as pairwise.wilcox.test() or posthoc.quade.test() (PMCMR package) might be used to make follow-up pairwise comparisons.

Example 5 – Interrupted Time Series Analysis

Interrupted time series analysis is a method for modeling time series data when an event or intervention occurs during the course of the time series. Interrupted time series analysis is not currently available in the **NCSS** procedures. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and performing the interrupted time series analysis in **R**.

Suppose researchers wish to determine the effect of adding a specific drug to the treatment regimen for an epidemic disease. The daily success rate of the treatment was followed for one year. The drug was added to the treatment regimen on day 198.

A simple scatter plot of the success rate versus the day (generated using the **NCSS** Scatter Plots procedure) is shown here:



There appears to be a transition point just before day 200 at which there is an upward change in the trajectory of the success rate of the treatment.

Setup

To run this example, complete the following steps:

1 Open the Treatment Success example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **Treatment Success** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 5** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\TreatmentSuccess.CSV** (The "R Data" folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

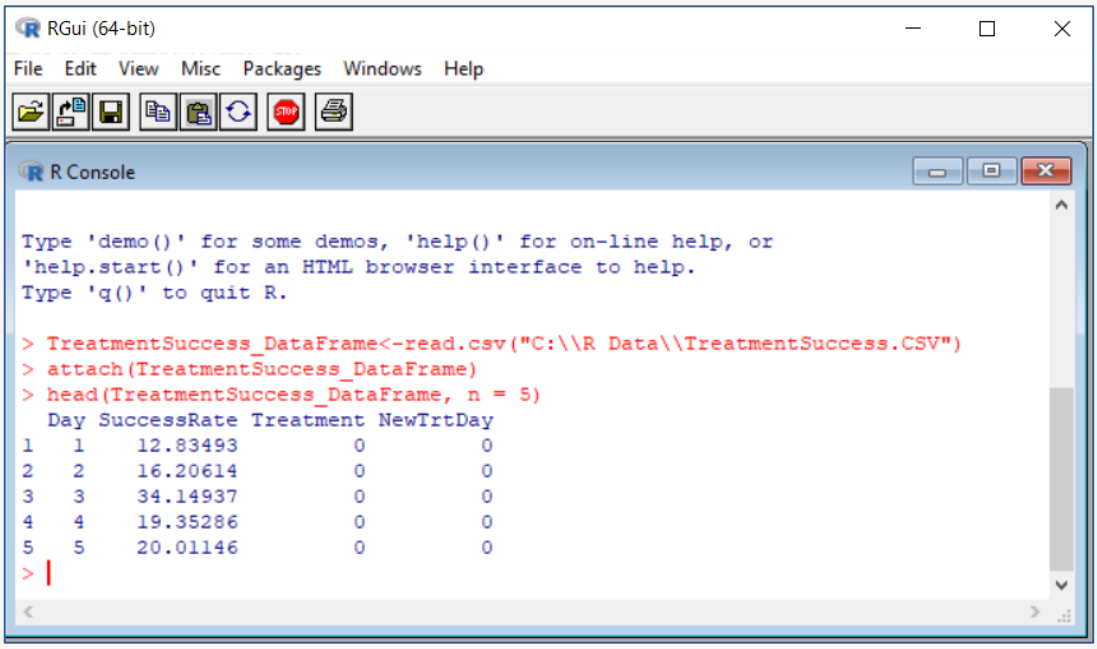
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
TreatmentSuccess_DataFrame<-read.csv("C:\\R Data\\TreatmentSuccess.CSV")
attach(TreatmentSuccess_DataFrame)
```

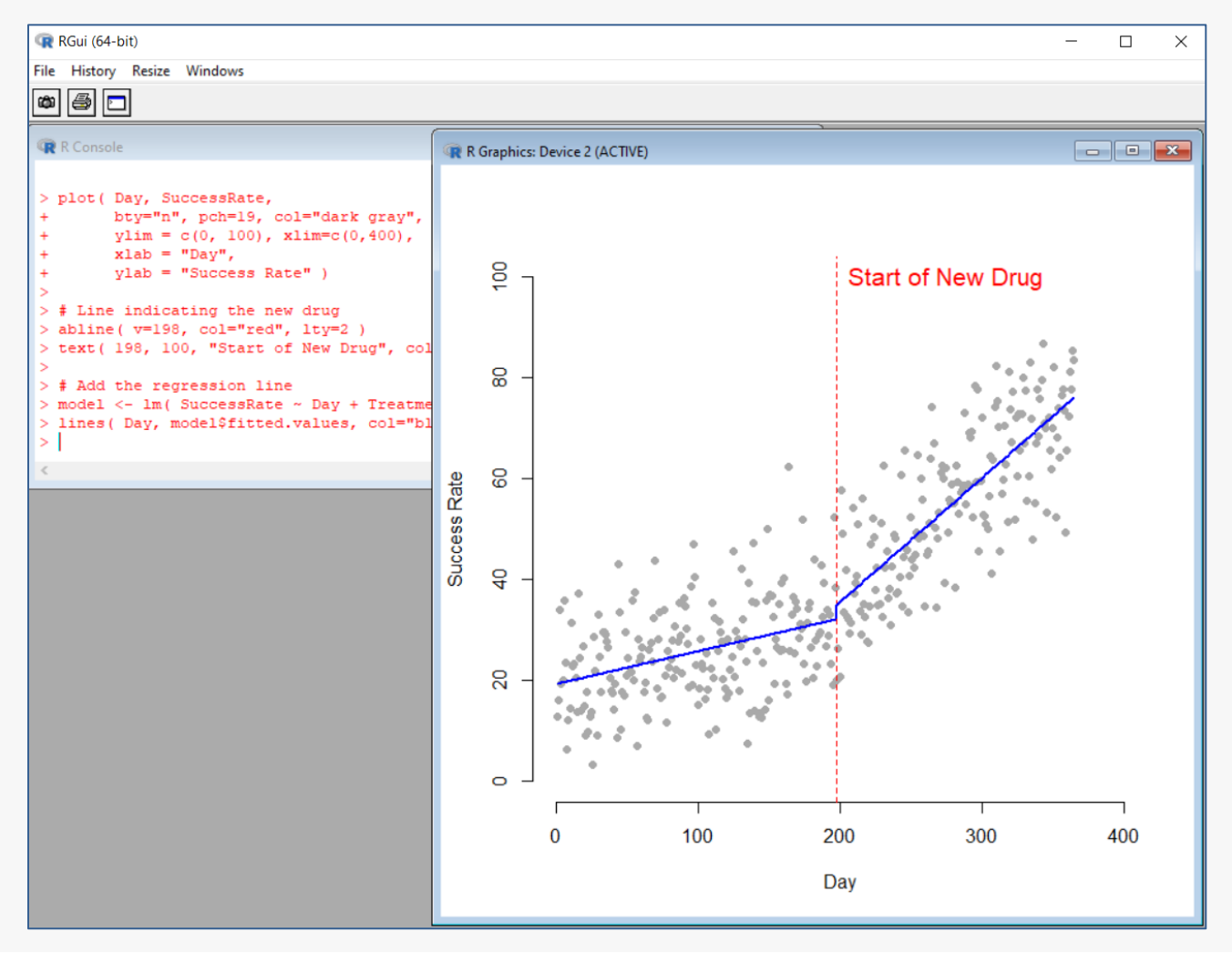
These two lines of **R** code are sufficient to read in the newly created TreatmentSuccess.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for Visualizing the Interrupted Time Series

```
### Begin the plotting
plot( Day, SuccessRate,
  bty="n", pch=19, col="dark gray",
  ylim = c(0, 100), xlim=c(0,400),
  xlab = "Day",
  ylab = "Success Rate" )

### Line indicating the new drug
abline( v=198, col="red", lty=2 )
text( 198, 100, "Start of New Drug", col="red", cex=1.3, pos=4 )

### Add the regression line
model <- lm( SuccessRate ~ Day + Treatment + NewTrtDay)
lines( Day, model$fitted.values, col="blue", lwd=2 )
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:

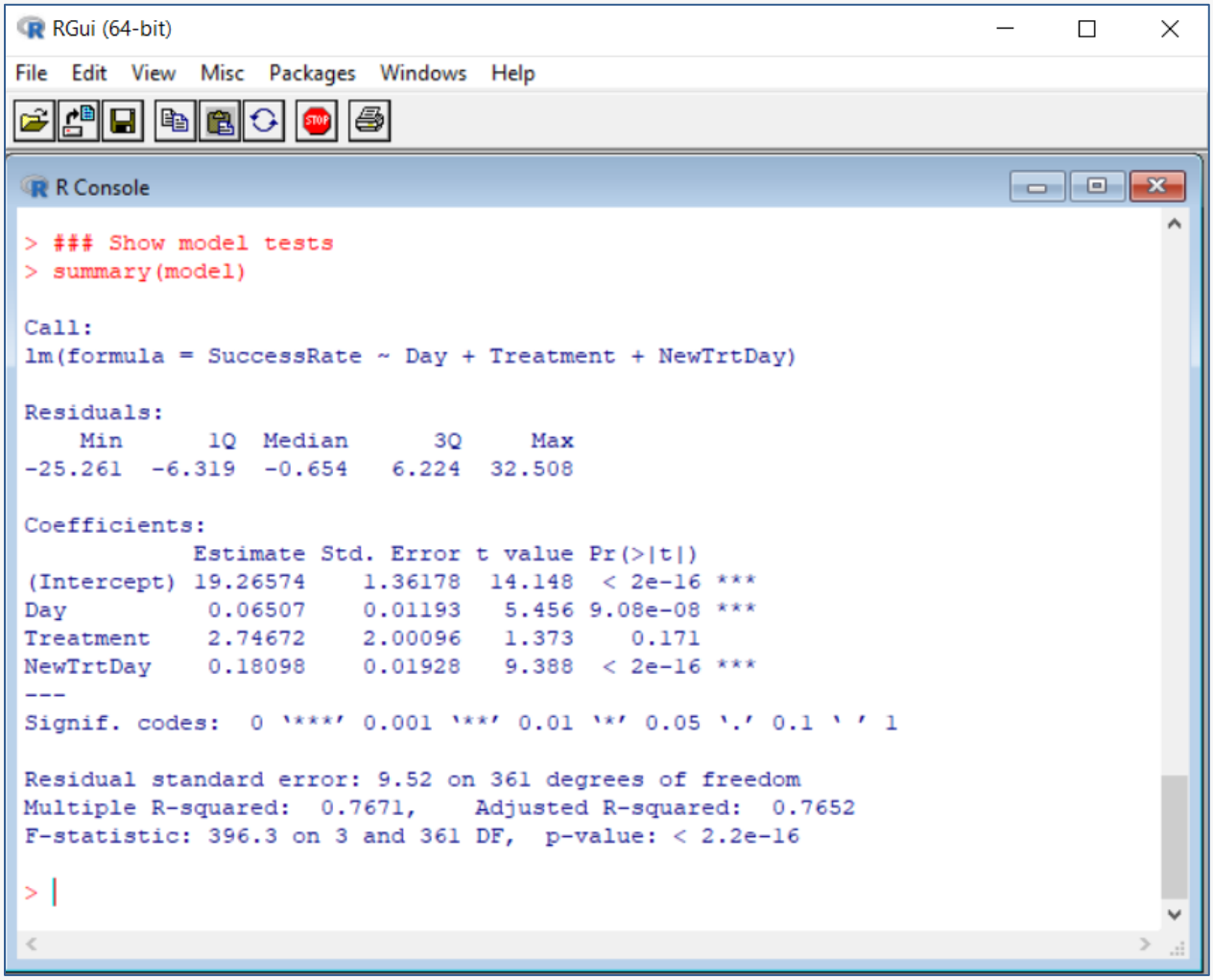


This code generates a highly specific plot of the interruption of the series.

R Code for Performing the Interrupted Time Series Analysis

```
### Show model tests
summary(model)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:

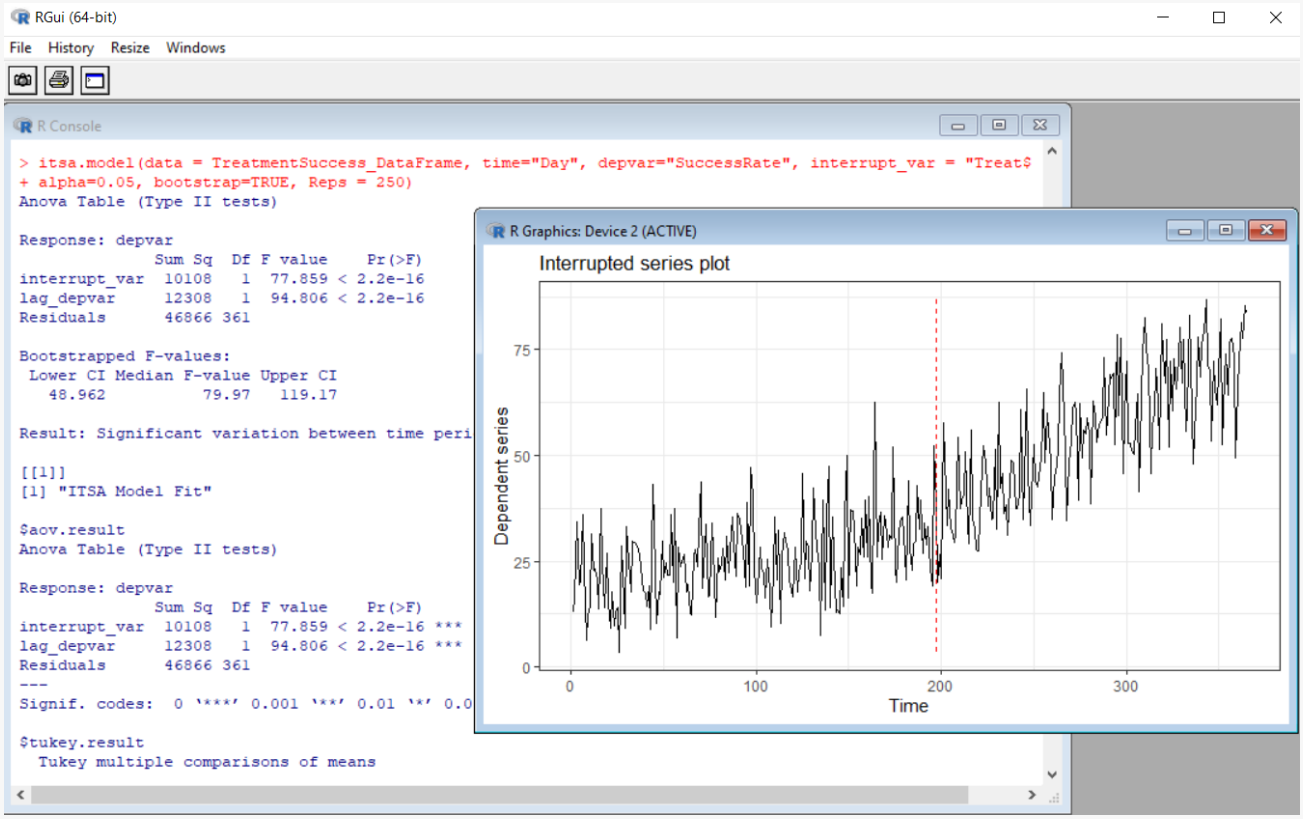


This code generates the same analysis that could be given in multiple regression in **NCSS**, but the plot with the overlaid regression lines could not be produced in **NCSS**.

R Code for Performing the Interrupted Time Series Analysis – its.analysis

```
### Install the its.analysis package
### If the its.analysis package has been installed previously, this line may be skipped
install.packages("its.analysis")

### Load the its.analysis package
library(its.analysis)

itsa.model(data = TreatmentSuccess_DataFrame, time="Day", depvar="SuccessRate", interrupt_var = "Treatment",
alpha=0.05, bootstrap=TRUE, Reps = 250)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



This analysis is not available in the **NCSS** procedures.

Example 6 – Generalized Estimating Equations

Generalized estimating equations (GEE) are used to estimate the parameters of a generalized linear model using semiparametric methods. Tools for analysis of GEE scenarios are not currently available in the **NCSS** procedures. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and performing a GEE analysis in **R**.

We will examine the Mixed Models procedure example for repeated measures data.

Setup

To run this example, complete the following steps:

1 Open the Pain example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **Pain** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 6** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\Pain.CSV** (The "R Data" folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

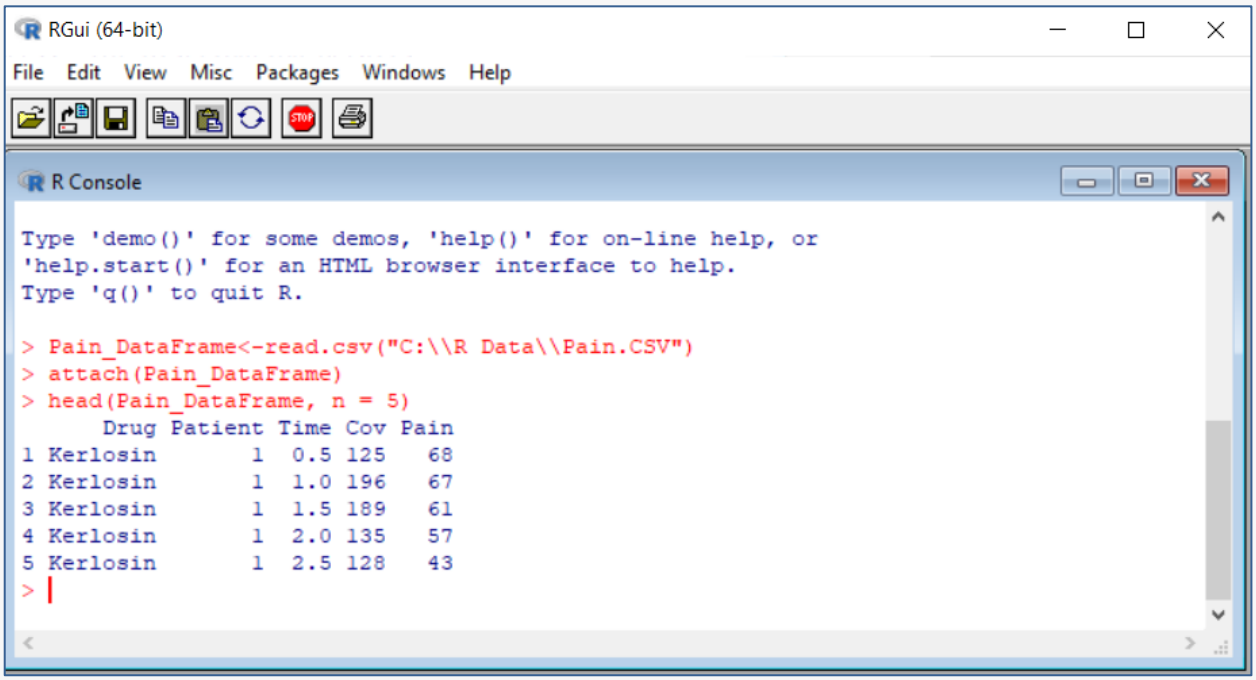
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
Pain_DataFrame<-read.csv("C:\\R Data\\Pain.CSV")
attach(Pain_DataFrame)
```

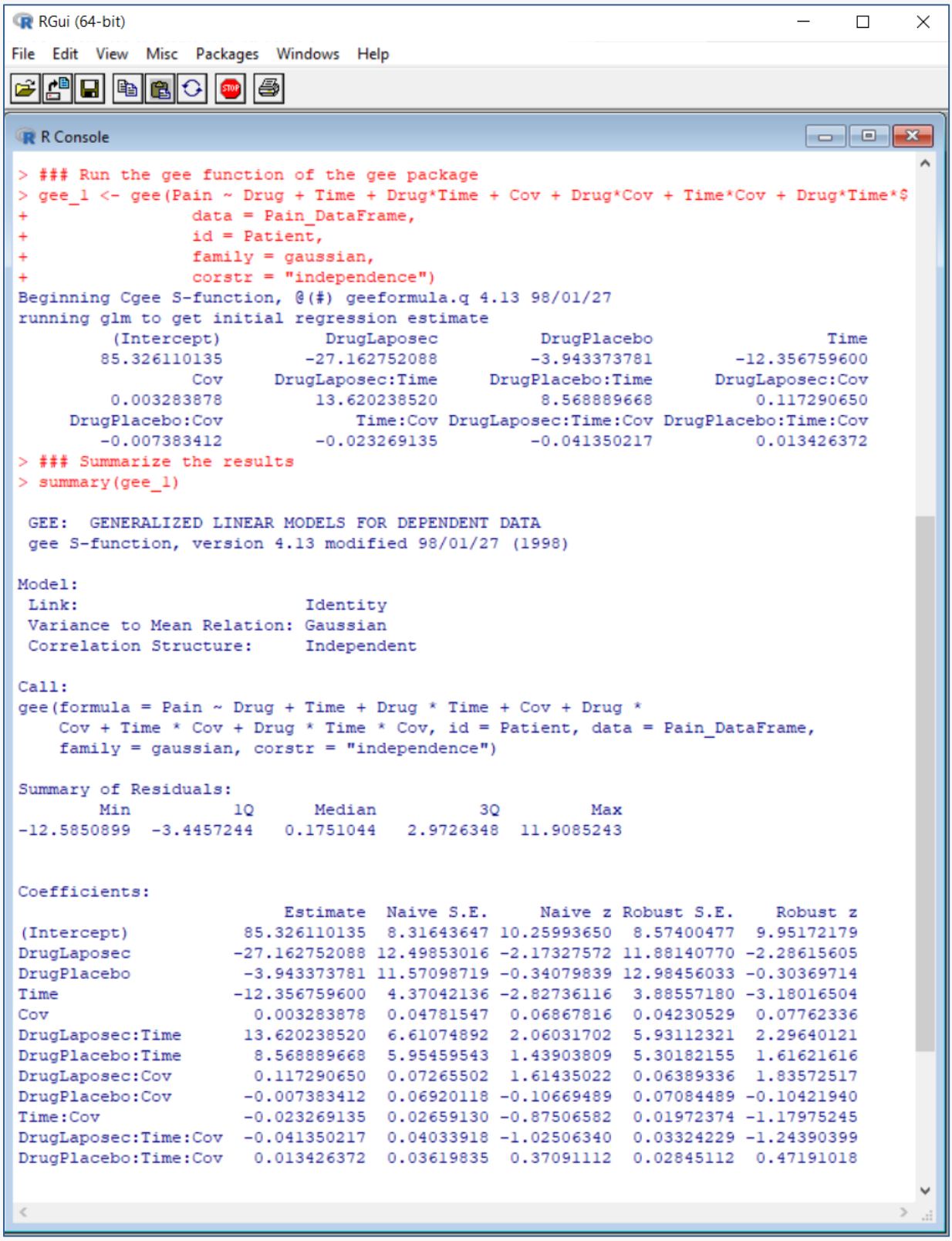
These two lines of **R** code are sufficient to read in the newly created Pain.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for Performing a GEE Analysis

```
### Install the gee package
### If the gee package has been installed previously, this line may be skipped
install.packages("gee")

### Load the gee package
library(gee)

### Run the gee function of the gee package
gee_1 <- gee(Pain ~ Drug + Time + Drug*Time + Cov + Drug*Cov + Time*Cov + Drug*Time*Cov,
             data = Pain_DataFrame,
             id = Patient,
             family = gaussian,
             corstr = "independence")

### Summarize the results
summary(gee_1)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:




```
> ### Run the gee function of the gee package
> gee_1 <- gee(Pain ~ Drug + Time + Drug*Time + Cov + Drug*Cov + Time*Cov + Drug*Time*$
+              data = Pain_DataFrame,
+              id = Patient,
+              family = gaussian,
+              corstr = "independence")
Beginning Cgee S-function, @(#) geeformula.q 4.13 98/01/27
running glm to get initial regression estimate
         (Intercept)          DrugLaposec          DrugPlacebo                 Time
        85.326110135        -27.162752088         -3.943373781        -12.356759600
                 Cov     DrugLaposec:Time     DrugPlacebo:Time      DrugLaposec:Cov
         0.003283878         13.620238520          8.568889668          0.117290650
     DrugPlacebo:Cov             Time:Cov DrugLaposec:Time:Cov DrugPlacebo:Time:Cov
        -0.007383412         -0.023269135         -0.041350217          0.013426372
> ### Summarize the results
> summary(gee_1)

 GEE:  GENERALIZED LINEAR MODELS FOR DEPENDENT DATA
 gee S-function, version 4.13 modified 98/01/27 (1998)

Model:
 Link:                      Identity
 Variance to Mean Relation: Gaussian
 Correlation Structure:     Independent

Call:
gee(formula = Pain ~ Drug + Time + Drug * Time + Cov + Drug *
    Cov + Time * Cov + Drug * Time * Cov, id = Patient, data = Pain_DataFrame,
    family = gaussian, corstr = "independence")

Summary of Residuals:
        Min          1Q      Median          3Q         Max
-12.5850899  -3.4457244   0.1751044   2.9726348  11.9085243


Coefficients:
                       Estimate  Naive S.E.     Naive z Robust S.E.    Robust z
(Intercept)          85.326110135  8.31643647 10.25993650  8.57400477  9.95172179
DrugLaposec         -27.162752088 12.49853016 -2.17327572 11.88140770 -2.28615605
DrugPlacebo          -3.943373781 11.57098719 -0.34079839 12.98456033 -0.30369714
Time                -12.356759600  4.37042136 -2.82736116  3.88557180 -3.18016504
Cov                   0.003283878  0.04781547  0.06867816  0.04230529  0.07762336
DrugLaposec:Time     13.620238520  6.61074892  2.06031702  5.93112321  2.29640121
DrugPlacebo:Time      8.568889668  5.95459543  1.43903809  5.30182155  1.61621616
DrugLaposec:Cov       0.117290650  0.07265502  1.61435022  0.06389336  1.83572517
DrugPlacebo:Cov      -0.007383412  0.06920118 -0.10669489  0.07084489 -0.10421940
Time:Cov             -0.023269135  0.02659130 -0.87506582  0.01972374 -1.17975245
DrugLaposec:Time:Cov -0.041350217  0.04033918 -1.02506340  0.03324229 -1.24390399
DrugPlacebo:Time:Cov  0.013426372  0.03619835  0.37091112  0.02845112  0.47191018
```

While a similar analysis is available in **NCSS** using the Mixed Models procedure, the GEE methods have additional capabilities that are not available in **NCSS**, such as the ability to use a link function (family) for Binomial or Poisson responses. There are also additional **R** packages such as GEEPACK.

Example 7 – Little's Missing Completely at Random Test

One concern that arises in multivariate data with missing values is whether the missing values are missing completely at random or whether missing values depend on the variables of the data set. Little (1988) developed a statistical test for such a scenario. Little's Missing Completely at Random (MCAR) test is not currently available in the **NCSS** procedures. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and performing Little's MCAR test in **R**.

The Missing data set contains what should be 74 responses on each of 8 items. However, there are missing values throughout the data set.

Setup

To run this example, complete the following steps:

1 Open the Missing example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **Missing** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 7** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\Missing.CSV** (The "R Data" folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

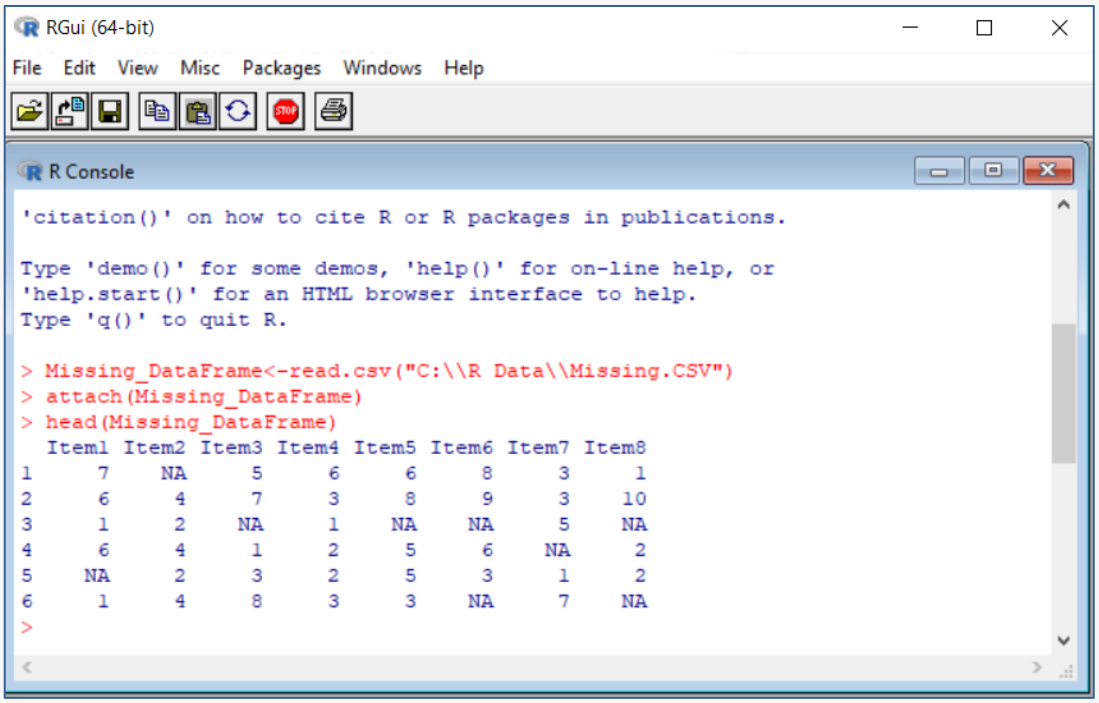
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
Missing_DataFrame<-read.csv("C:\\R Data\\Missing.CSV")
attach(Missing_DataFrame)
```

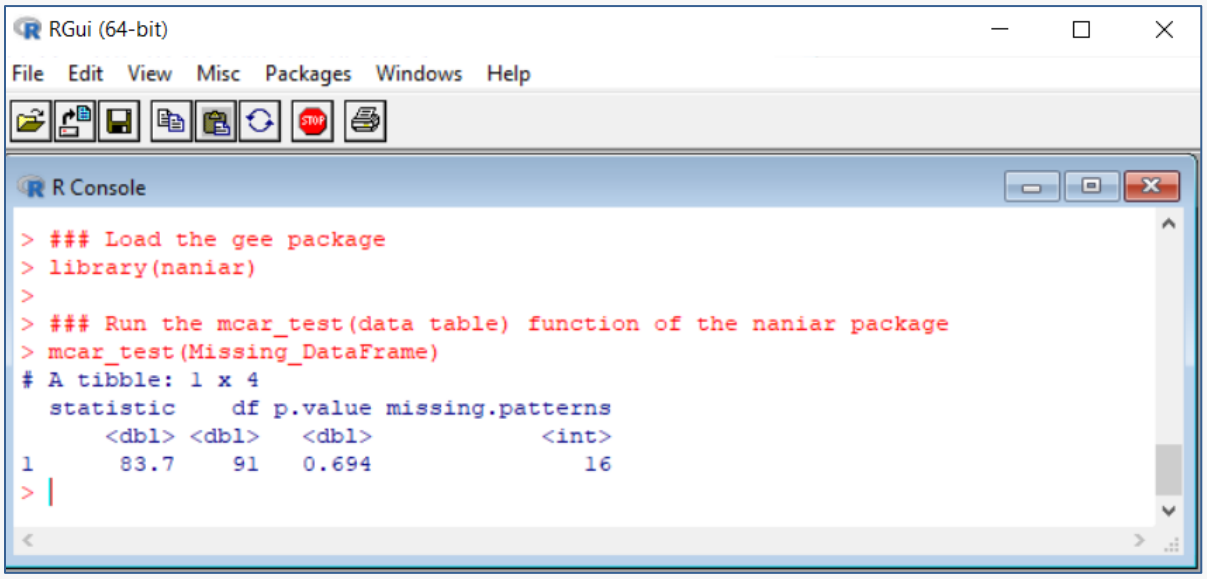
These two lines of **R** code are sufficient to read in the newly created Missing.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for Performing Little's Missing Completely at Random Test

```
### Install the naniar package
### If the naniar package has been installed previously, this line may be skipped
install.packages("naniar")

### Load the naniar package
library(naniar)

### Run the mcar_test(data table) function of the naniar package
mcar_test(Missing_DataFrame)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



The p-value of 0.694 indicates that there is not strong evidence that the missing values are not missing completely at random.

Example 8 – Viewing Data on Maps

Viewing data on maps can be helpful in recognizing patterns across regions. Viewing statistics corresponding to regions on maps is not currently available in the **NCSS** procedures. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and viewing map data in **R**.

The State Population data set contains columns identifying each USA state and the estimated population of that state in 2019. The **R** code used in this example requires a column named 'fips' that contains the appropriate state identifying value.

Setup

To run this example, complete the following steps:

1 Open the State Population example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **State Population** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 8** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\StatePopulation.CSV** (The "R Data" folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

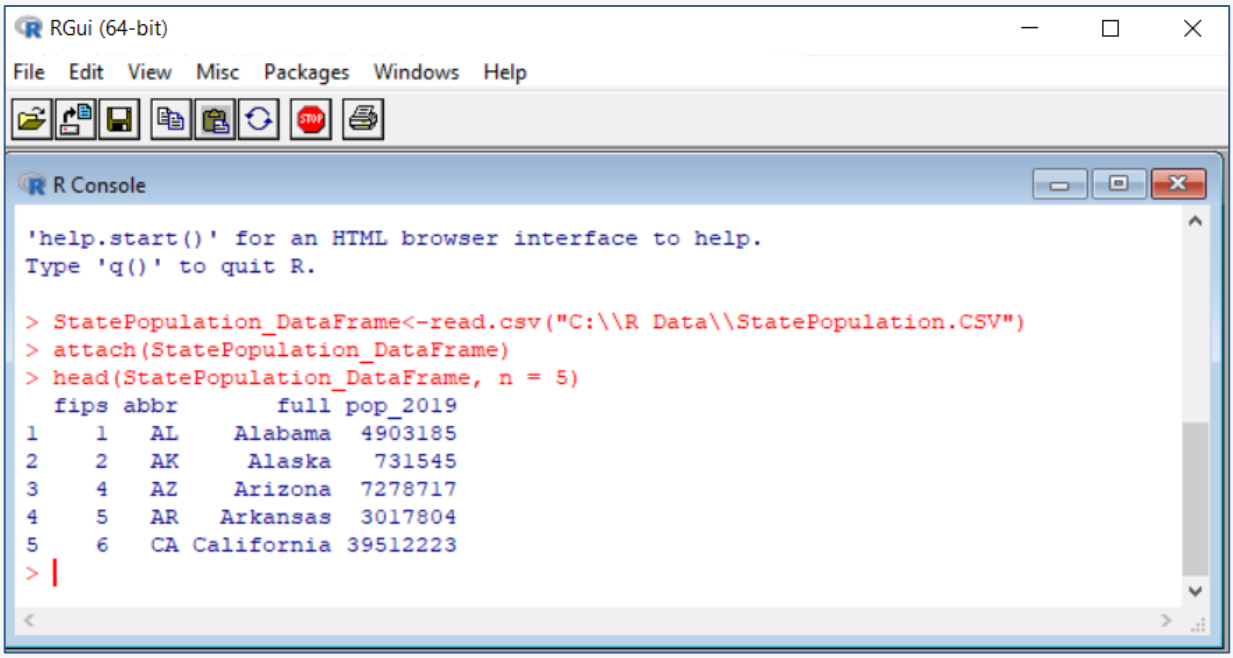
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
StatePopulation_DataFrame<-read.csv("C:\\R Data\\StatePopulation.CSV")
attach(StatePopulation_DataFrame)
```

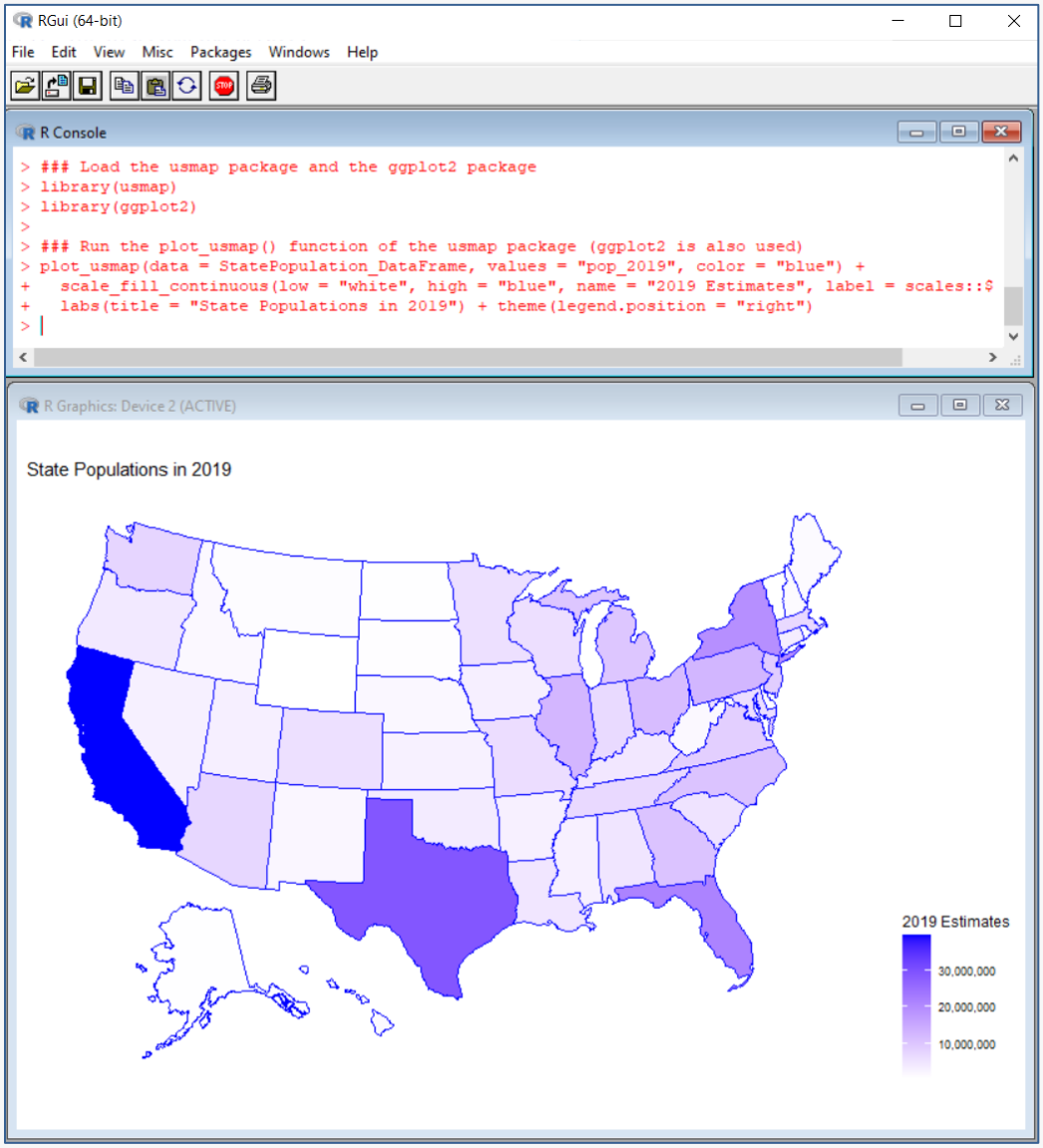
These two lines of **R** code are sufficient to read in the newly created StatePopulation.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for Plotting the State Population Data on a Map

```
### Install the usmap package
### If the usmap package has been installed previously, this line may be skipped
install.packages("usmap")

### Load the usmap package and the ggplot2 package
library(usmap)
library(ggplot2)

### Run the plot_usmap() function of the usmap package (ggplot2 is also used)
plot_usmap(data = StatePopulation_DataFrame, values = "pop_2019", color = "blue") +
  scale_fill_continuous(low = "white", high = "blue", name = "2019 Estimates", label = scales::comma) +
  labs(title = "State Populations in 2019") + theme(legend.position = "right")
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



Options for working with county data or regional data are also available in the usmap package.

Example 9 – Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines, sometimes called MARS or MARS Regression, is a piecewise modeling technique for the multiple regression scenario. MARS is not currently available in the **NCSS** procedures. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and running a basic MARS analysis in **R**.

Setup

To run this example, complete the following steps:

1 Open the Resale example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **Resale** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 9** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\Resale.CSV** (The “R Data” folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

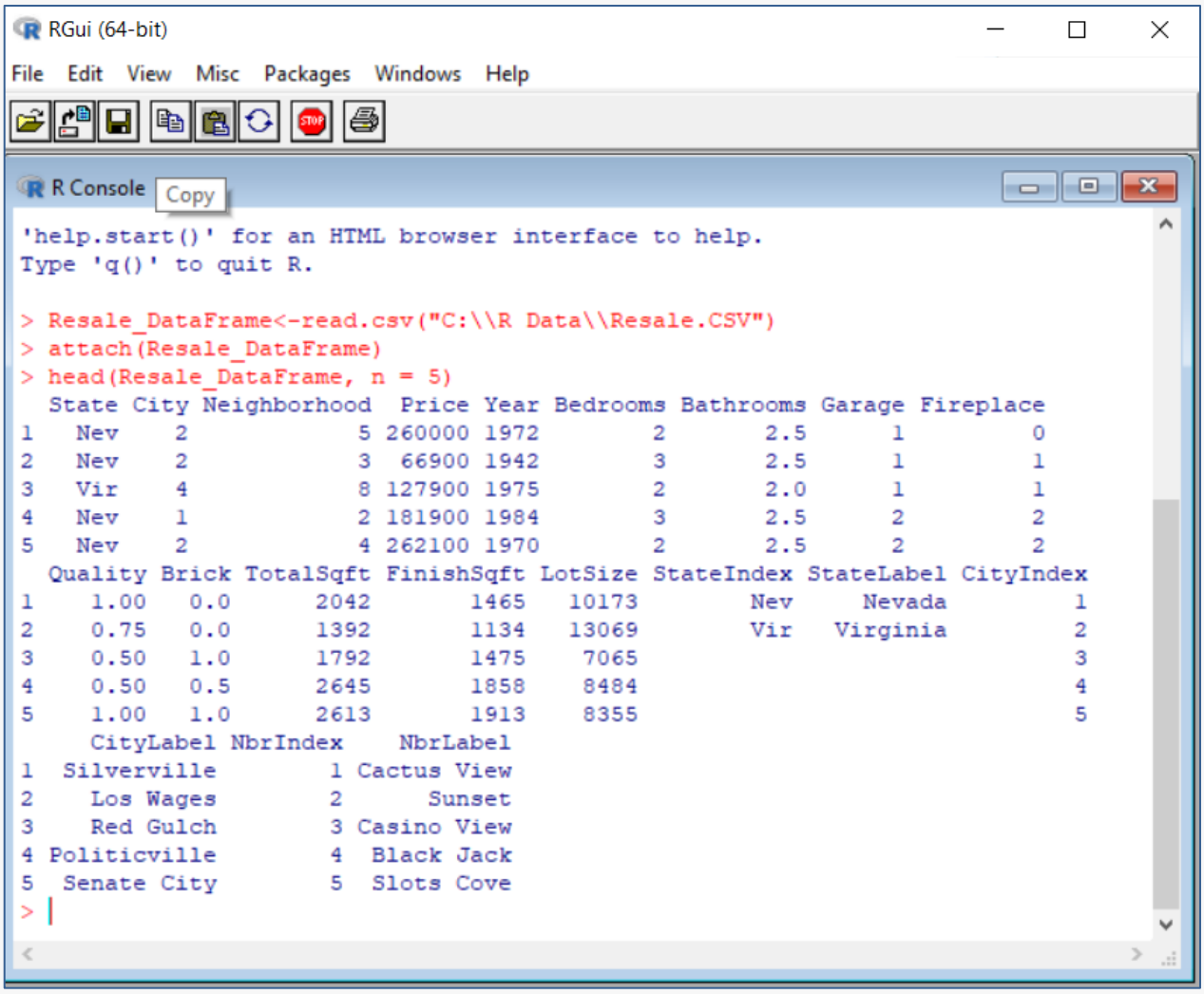
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
Resale_DataFrame<-read.csv("C:\\R Data\\Resale.CSV")
attach(Resale_DataFrame)
```

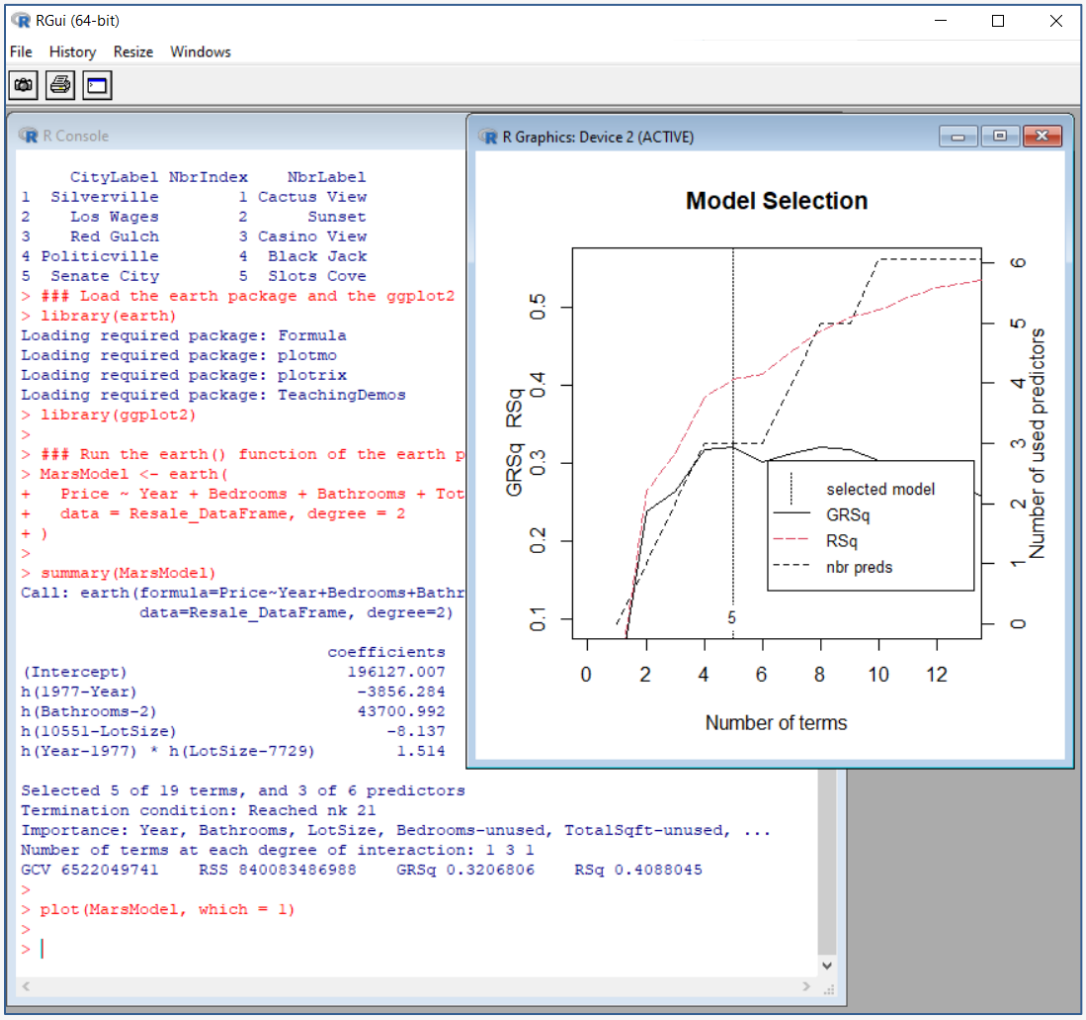
These two lines of **R** code are sufficient to read in the newly created Resale.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for a Basic MARS Analysis

```
### Install the earth package
### If the earth package has been installed previously, this line may be skipped
install.packages("earth")

### Load the earth package and the ggplot2 package
library(earth)
library(ggplot2)

### Run the earth() function of the earth package
MarsModel <- earth(
  Price ~ Year + Bedrooms + Bathrooms + TotalSqft + FinishSqft + LotSize,
  data = Resale_DataFrame, degree = 2
)

summary(MarsModel)

plot(MarsModel, which = 1)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



There are many additional options and plots that are available in **R** for a MARS analysis.

Example 10 – Factor Analysis with Oblique Rotation

Factor Analysis and Principal Components Analysis procedures in **NCSS** currently have only orthogonal rotation options. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and running a basic Factor Analysis with oblique rotation in **R**.

Setup

To run this example, complete the following steps:

1 Open the PCA2Xs example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **PCA2Xs** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 10** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\PCA2Xs.CSV** (The “R Data” folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

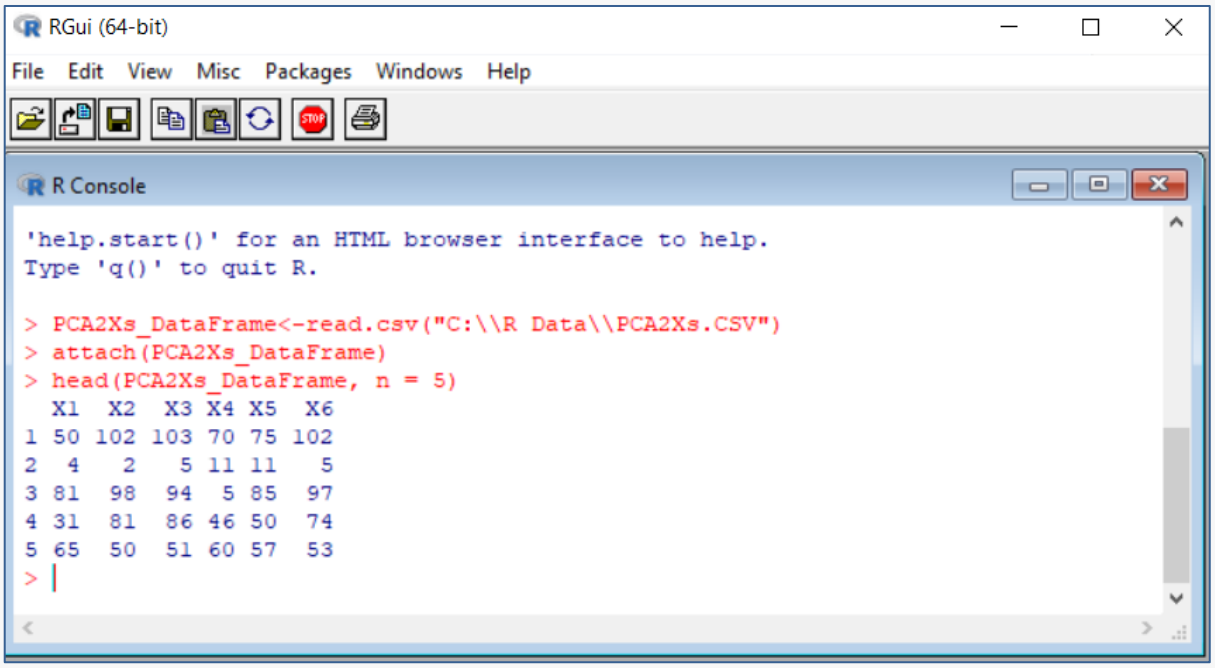
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
PCA2Xs_DataFrame<-read.csv("C:\\R Data\\PCA2Xs.CSV")
attach(PCA2Xs_DataFrame)
```

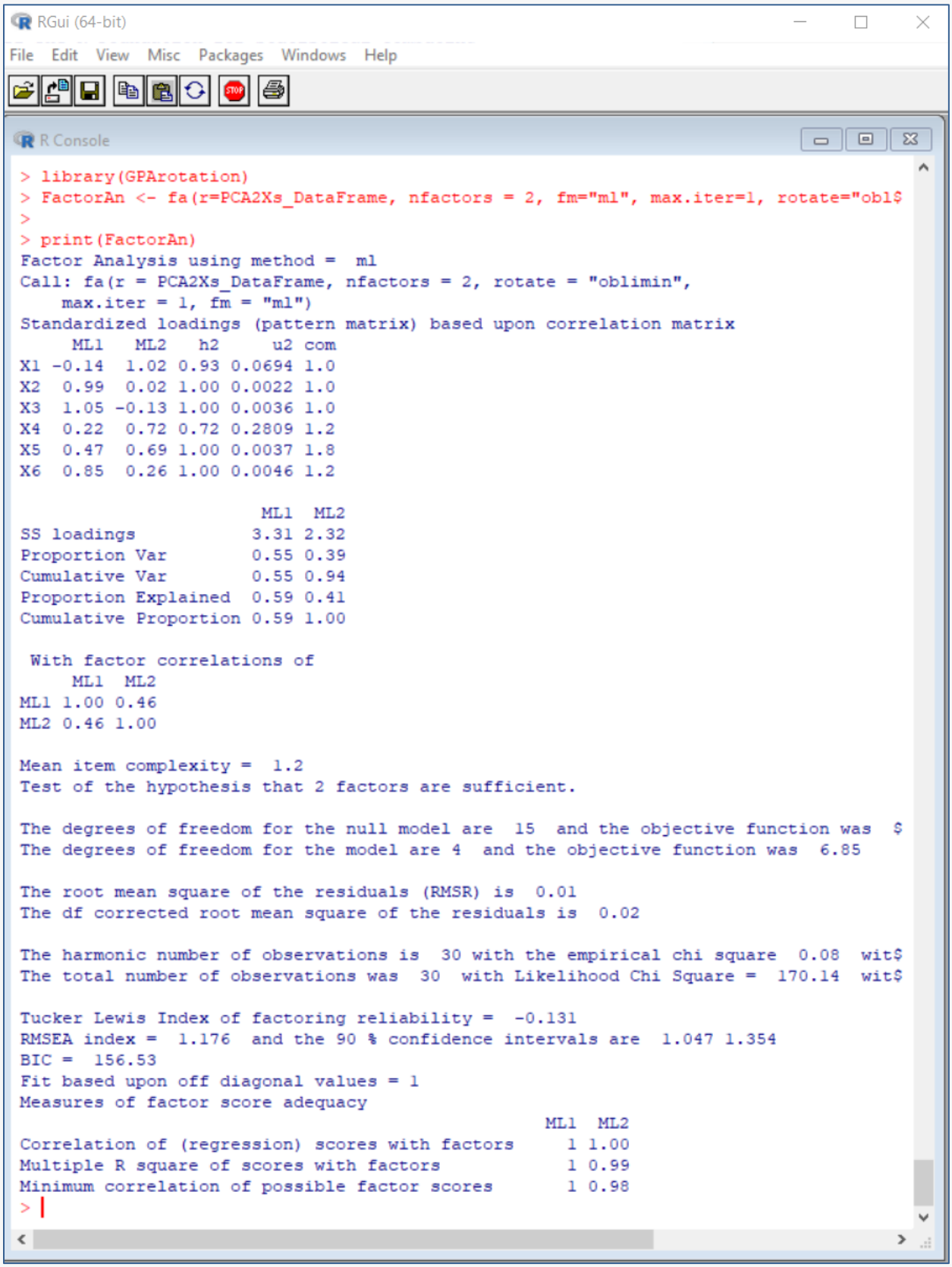
These two lines of **R** code are sufficient to read in the newly created PCA2Xs.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for a Basic Factor Analysis with Oblique Rotation

```
### Install the psych package and the GPArotation package
### If the psych and GPArotation packages have been installed previously, these lines may be skipped
install.packages("psych")
install.packages("GPArotation")

### Load the psych and GPArotation packages
library(psych)
library(GPArotation)

### Run the fa() function of the psych package (also uses the GPArotation package)
FactorAn <- fa(r=PCA2Xs_DataFrame, nfactors = 2, fm="ml", max.iter=1, rotate="oblimin")

print(FactorAn)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



There are many additional options that are available in **R** for Factor Analysis.

Example 11 – Hierarchical Clustering using Squared Euclidean Distance

The Hierarchical Clustering procedure in **NCSS** gives the option for Euclidean or Manhattan distance, but it does not offer the squared Euclidean distance as an option. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and obtaining a hierarchical clustering analysis with squared Euclidean distance in **R**.

Setup

To run this example, complete the following steps:

1 Open the BBallPart example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **BBallPart** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 11** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\BBallPart.CSV** (The "R Data" folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

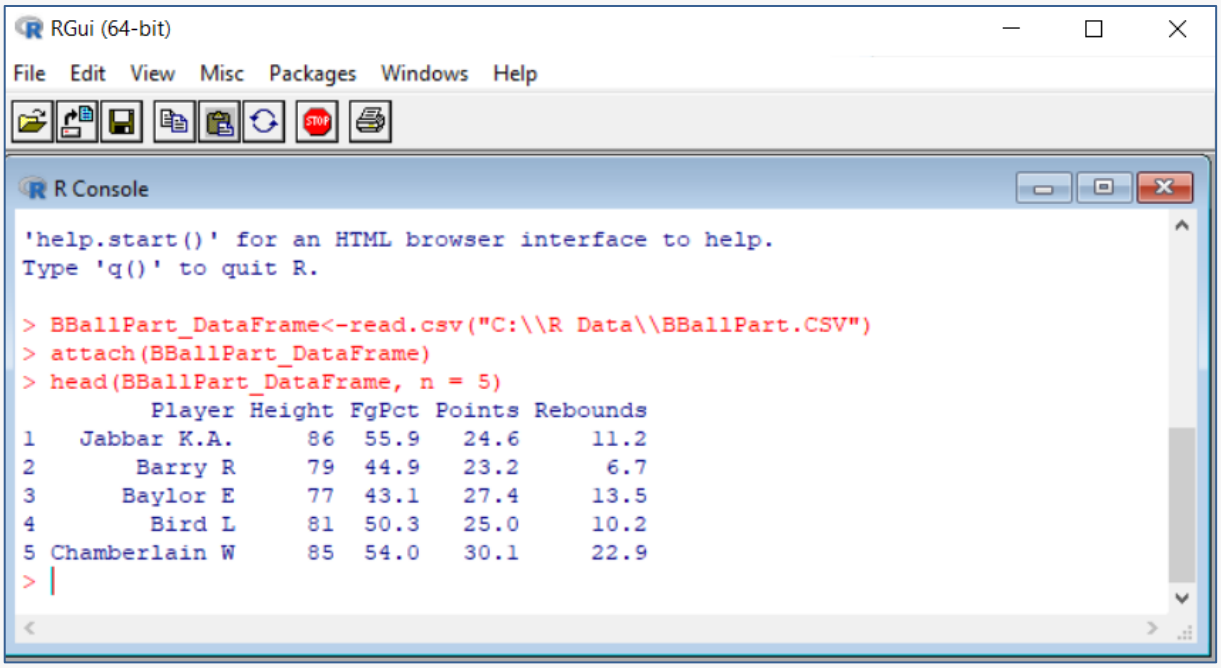
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
BBallPart_DataFrame<-read.csv("C:\\R Data\\BBallPart.CSV")
attach(BBallPart_DataFrame)
```

These two lines of **R** code are sufficient to read in the newly created BBallPart.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for a Basic Cluster Analysis

```
### No packages need to be installed or loaded, since the functions used are part of the native ‘stats’ package

rownames(BBallPart_DataFrame) <- Player

dists1 <- dist(BBallPart_DataFrame[ , 2:5], method = "euclidean")

dists1

hclusts1 <- hclust(dists1, method = "ward.D2")

plot(hclusts1, hang = -1)
```

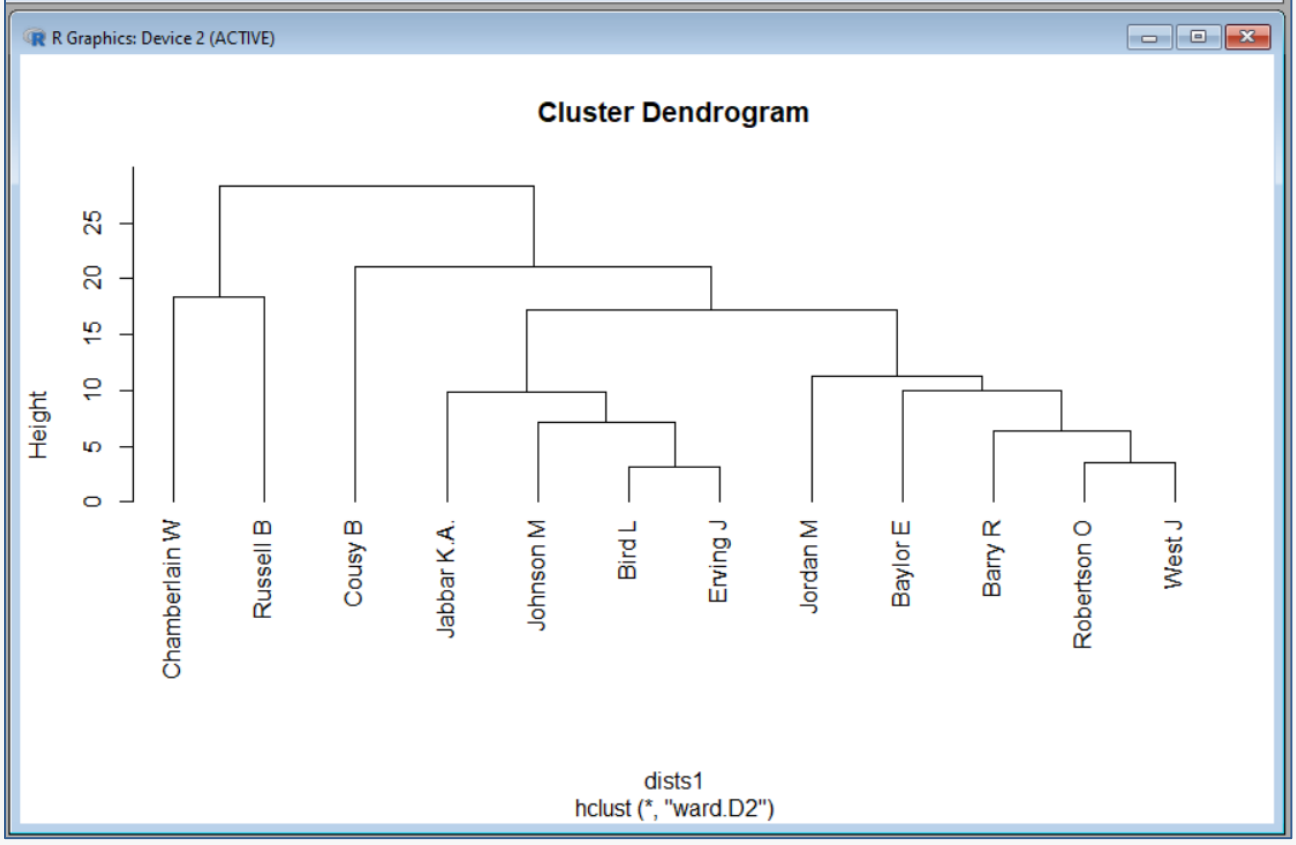
These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:

RGui (64-bit)

File History Resize Windows

R Console

```
> rownames(BBallPart_DataFrame) <- Player
>
> distsl <- dist(BBallPart_DataFrame[ , 2:5], method = "euclidean")
>
> distsl
              Jabbar K.A.   Barry R  Baylor E    Bird L Chamberlain W   Cousy B  Erving J Johnson M  Jord$
Barry R         13.863982                                                                            $
Baylor E        16.061445  8.433267                                                                  $
Bird L           7.584194  6.974955  9.191844                                                        $
Chamberlain W   13.105342 20.708935 16.687121 14.730580                                             $
Cousy B         24.397746 11.058933 14.195070 17.454226     29.642537                               $
Erving J         9.580710  6.081118  9.681942  3.141656     17.204069 15.878917                     $
Johnson M        8.594184  9.153688 14.616087  6.736468     19.225244 17.847969  5.942222           $
Jordan M        13.196969 11.426723 12.188929  9.152049     18.477825 20.574985  8.751571 13.599265 $
Robertson O     12.274364  4.883646  8.249242  5.198077     18.728855 14.136124  3.148015  8.642916  7.62$
Russell B       19.546611 18.041064 15.973728 17.079520     18.366818 20.902153 18.280591 18.211260 25.31$
West J          15.473203  6.461424  9.175511  8.603488     21.350176 13.278931  6.431951 11.507389  7.67$
>
> hclustsl <- hclust(distsl, method = "ward.D2")
>
> plot(hclustsl, hang = -1)
>
```



There are many additional options that are available in **R** for a cluster analysis.

Example 12 – Rolling Correlation

NCSS has several correlation tools and reports, but it does not offer the rolling correlation as an option. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and obtaining a rolling correlation output in **R**.

Setup

To run this example, complete the following steps:

1 Open the IntelDay example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **IntelDay** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 12** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\IntelDay.CSV** (The "R Data" folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

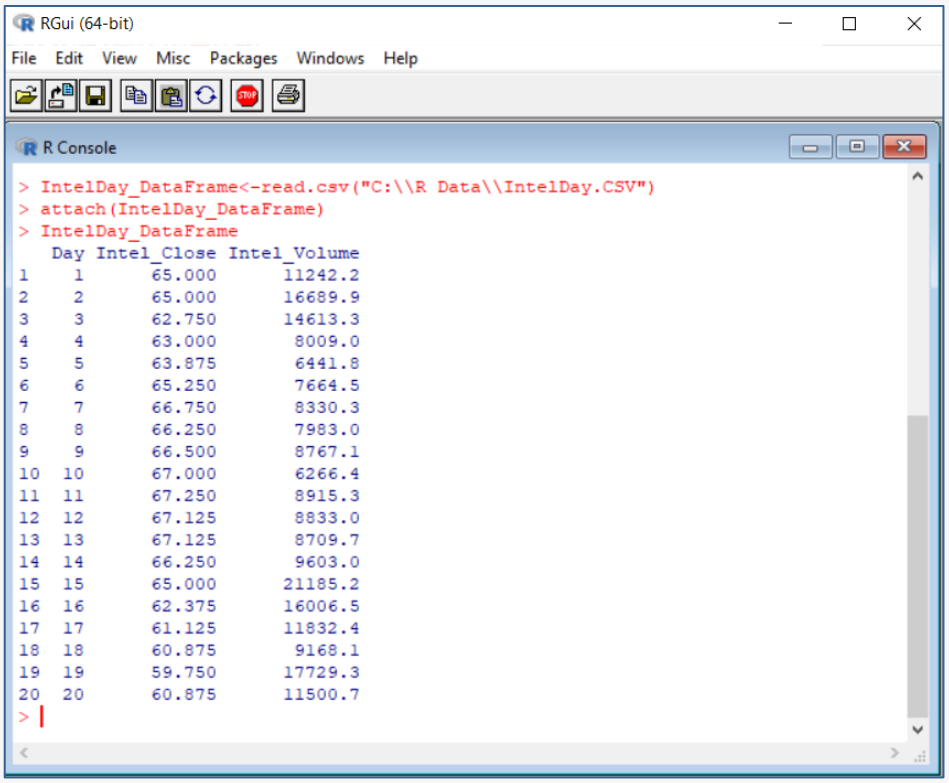
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
IntelDay_DataFrame<-read.csv("C:\\R Data\\IntelDay.CSV")
attach(IntelDay_DataFrame)
```

These two lines of **R** code are sufficient to read in the newly created IntelDay.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code to Obtain Rolling Correlation

```
### Install the roll package
### If the roll package has been installed previously, this line may be skipped
install.packages("roll")

### Load the roll package
library(roll)

### Run the roll_cor() function of the roll package
rollingcorrs <- roll_cor(Intel_Volume, Intel_Close, width = 5)

print(rollingcorrs)

plot(Day,rollingcorrs)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:

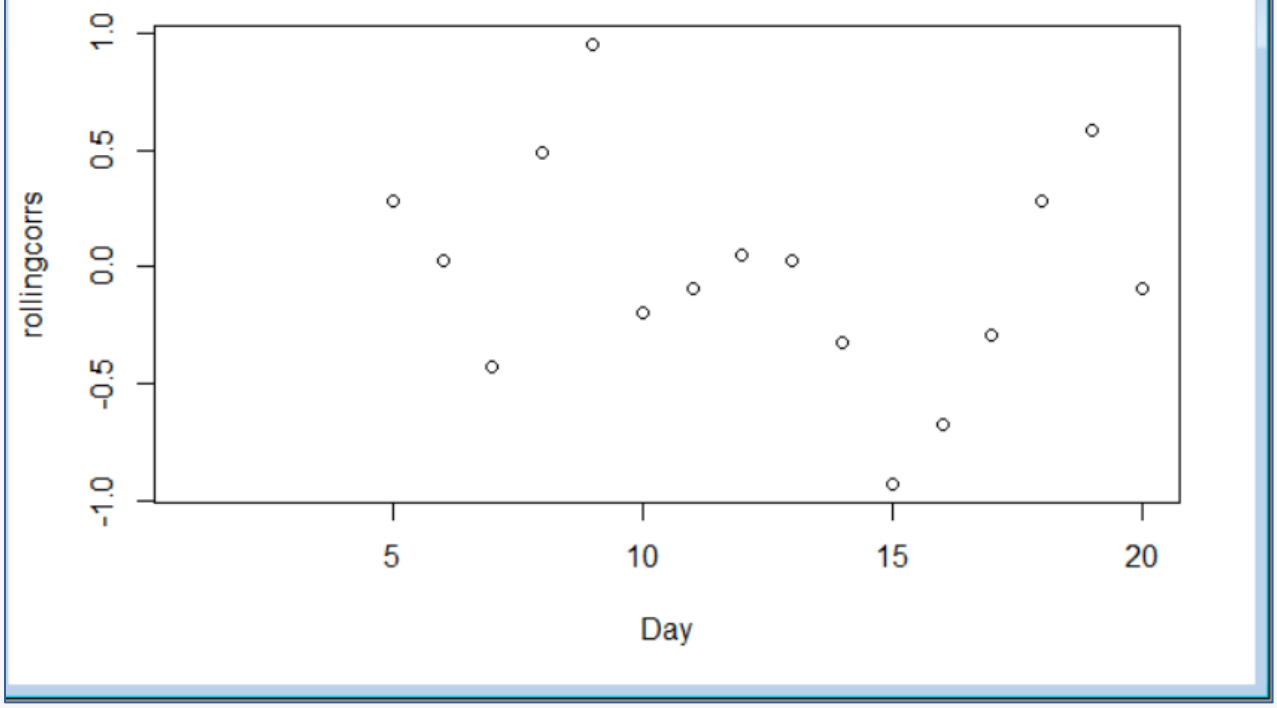
RGui (64-bit)

File History Resize Windows

R Console

```
> ### Load the roll package
> library(roll)
>
> ### Run the roll_cor() function of the roll package
> rollingcorrs <- roll_cor(Intel_Volume, Intel_Close, width = 5)
>
> print(rollingcorrs)
 [1]          NA          NA          NA          NA  0.27857181  0.02501528
 [7] -0.43009151  0.48515707  0.95310754 -0.19798661 -0.09453855  0.04994350
[13]  0.02854985 -0.32403828 -0.93130636 -0.67299945 -0.29035280  0.27783765
[19]  0.58387709 -0.09561951
>
> plot(Day,rollingcorrs)
>
```

R Graphics: Device 2 (ACTIVE)



Additional options are available in this function of **R** for rolling correlation, such partial windows and expanding correlations.

Example 13 – Classification and Regression Trees

Classification and Regression Trees (CART) methods are typically used to develop decision trees. Tools for CART are not currently available in the **NCSS** procedures. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and producing a decision tree in **R**.

Setup

To run this example, complete the following steps:

1 Open the Hypertension example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **Hypertension** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 13** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\Hypertension.CSV** (The "R Data" folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

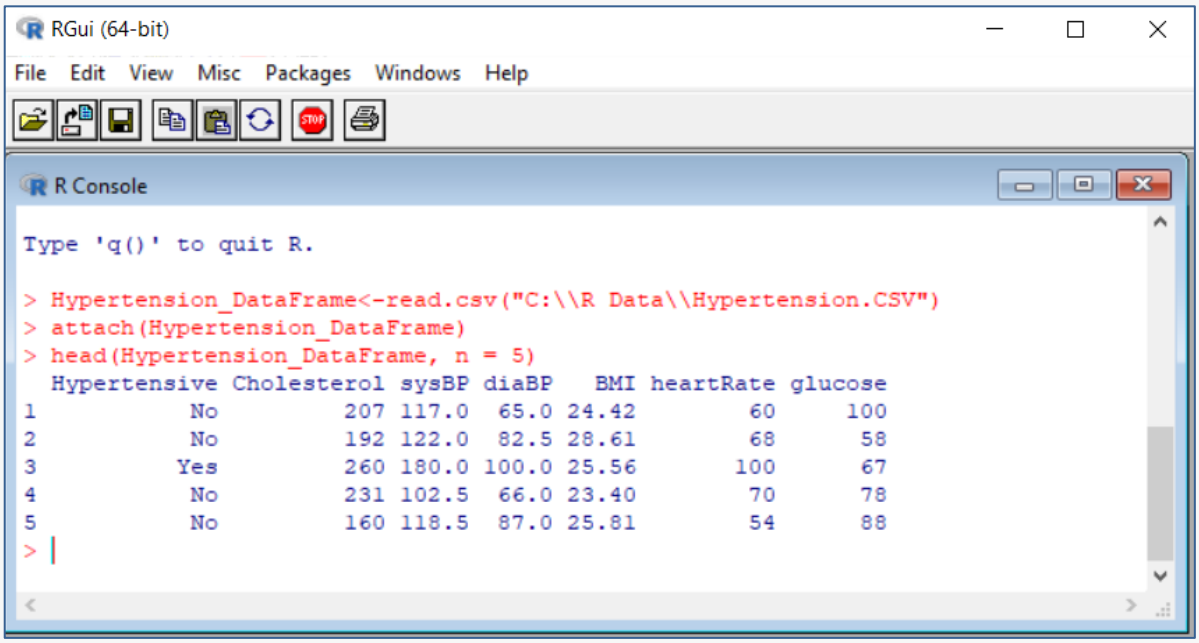
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
Hypertension_DataFrame<-read.csv("C:\\R Data\\Hypertension.CSV")
attach(Hypertension_DataFrame)
```

These two lines of **R** code are sufficient to read in the newly created Hypertension.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.



R Code for a Simple CART Analysis

```
### Install the rpart package
### If the rpart package has been installed previously, this line may be skipped
install.packages("rpart")

### Load the rpart package
library(rpart)

### Run the rpart function of the rpart package
treefit <- rpart(Hypertensive ~ Cholesterol +
  sysBP + diaBP + BMI + heartRate + glucose,
  method="class", data=Hypertension_DataFrame)

printcp(treefit)
plotcp(treefit)

summary(treefit)

plot(treefit, uniform=TRUE,
  main="Hypertension Classification Tree")
text(treefit, use.n=TRUE, all=TRUE, cex=.8)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:

Exporting Data to R

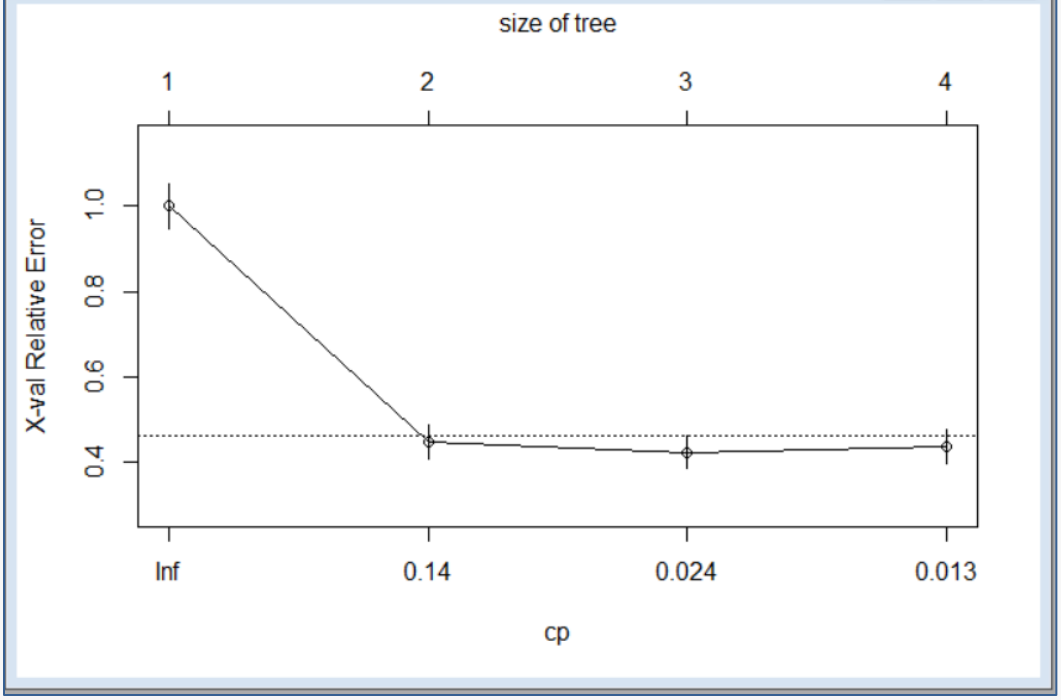
RGui (64-bit)

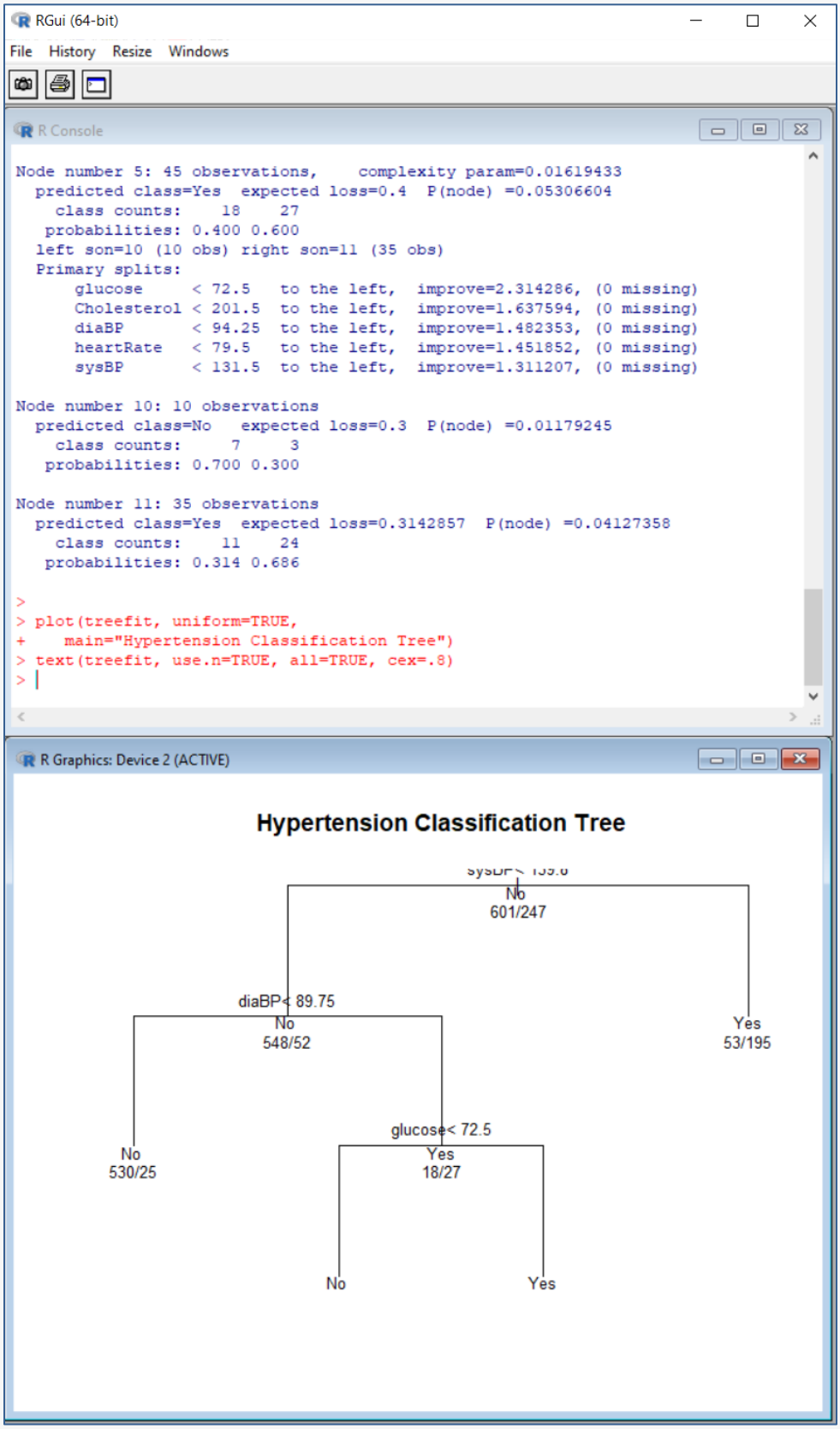
File Edit View Misc Packages Windows Help

R Console

```
> library(rpart)
>
> ### Run the rpart function of the rpart package
> treefit <- rpart(Hypertensive ~ Cholesterol +
+ sysBP + diaBP + BMI + heartRate + glucose,
+    method="class", data=Hypertension_DataFrame)
>
> printcp(treefit)

Classification tree:
rpart(formula = Hypertensive ~ Cholesterol + sysBP + diaBP +
    BMI + heartRate + glucose, data = Hypertension_DataFrame,
    method = "class")

Variables actually used in tree construction:
[1] diaBP   glucose sysBP

Root node error: 247/848 = 0.29127

n= 848

        CP nsplit rel error  xerror     xstd
1 0.574899      0   1.00000 1.00000 0.053566
2 0.036437      1   0.42510 0.44939 0.039765
3 0.016194      2   0.38866 0.42510 0.038832
4 0.010000      3   0.37247 0.43725 0.039304
> plotcp(treefit)
>
```

R Graphics: Device 2 (ACTIVE)





Many additional functions and **R** packages can be used to perform a variety of CART analyses in **R**.

Example 14 – Meta-Analysis of One Proportion

While **NCSS** has a procedure for meta-analysis of two proportions (and two correlated proportions), a procedure for meta-analysis of one proportion is not currently available in **NCSS**. This example shows the process of exporting the data from **NCSS**, reading the data into **R**, and performing a meta-analysis of one proportion in **R**.

Setup

To run this example, complete the following steps:

1 Open the MetaOneProp example dataset

- From the File menu of the NCSS Data window, select **Open Example Data**.
- Select **MetaOneProp** and click **OK**.

2 Specify the Exporting Data to R procedure options

- Find and open the **Exporting Data to R** procedure using the menus or the Procedure Navigator.
- The settings for this example are listed below and are stored in the **Example 14** settings file. To load these settings to the procedure window, click **Open Example Settings File** in the Help Center or File menu.

Export Tab	
Rows to Export	**Export All Rows**
Columns to Export	**Export All Columns**
Path and Name	**C:\R Data\MetaOneProp.CSV** (The “R Data” folder will need to be created if it is not already.)
R Code - Without Notes	**Checked**
R Code - With Notes	**Checked**
R Code - Summary Functions	**Checked**

3 Run the procedure

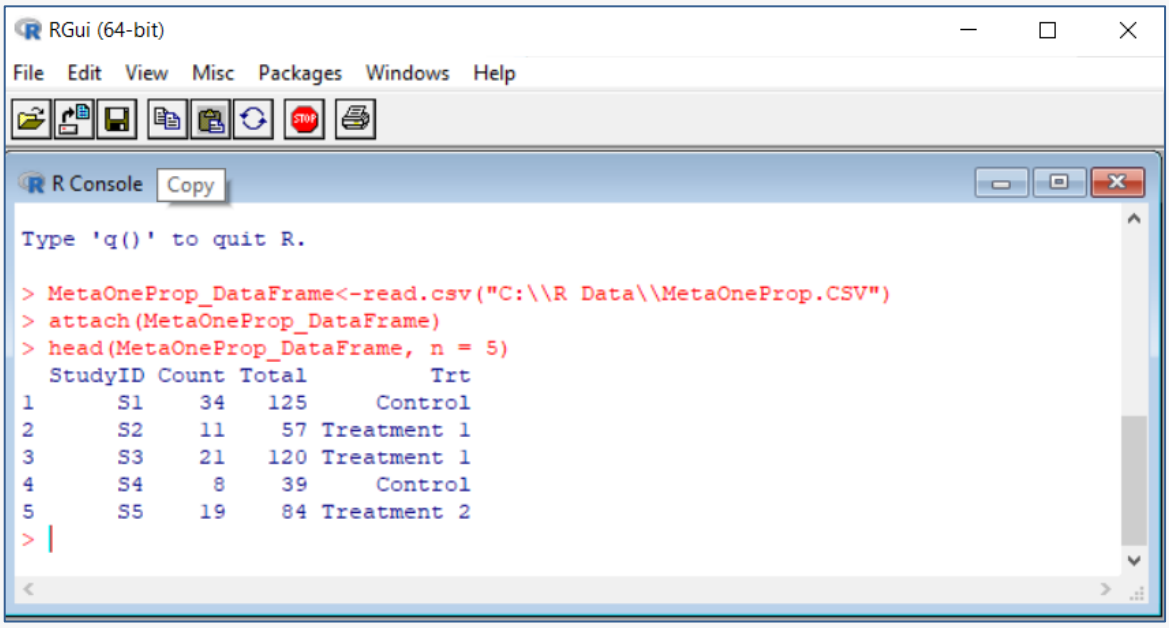
- Click the **Run** button to perform the calculations and generate the output.

R Code for Reading in the CSV Data File (No Notes)

R Code for Reading in the CSV Data File (No Notes)

```
MetaOneProp_DataFrame<-read.csv("C:\\R Data\\MetaOneProp.CSV")
attach(MetaOneProp_DataFrame)
```

These two lines of **R** code are sufficient to read in the newly created MetaOneProp.CSV file and make the columns ready for use by their individual names. These lines can be copied and pasted into the **R** Console.

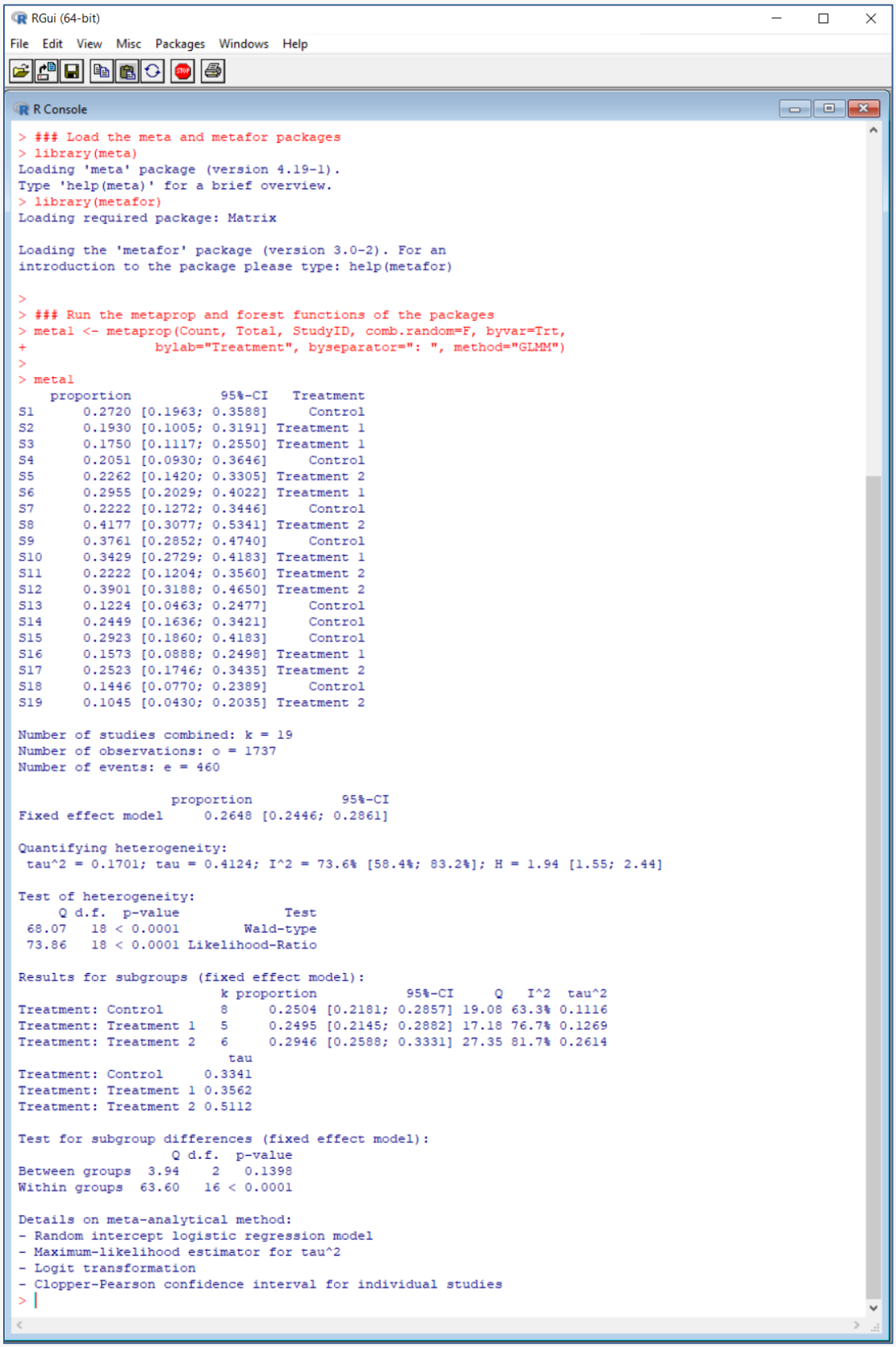


R Code for a Meta-Analysis of One Proportion

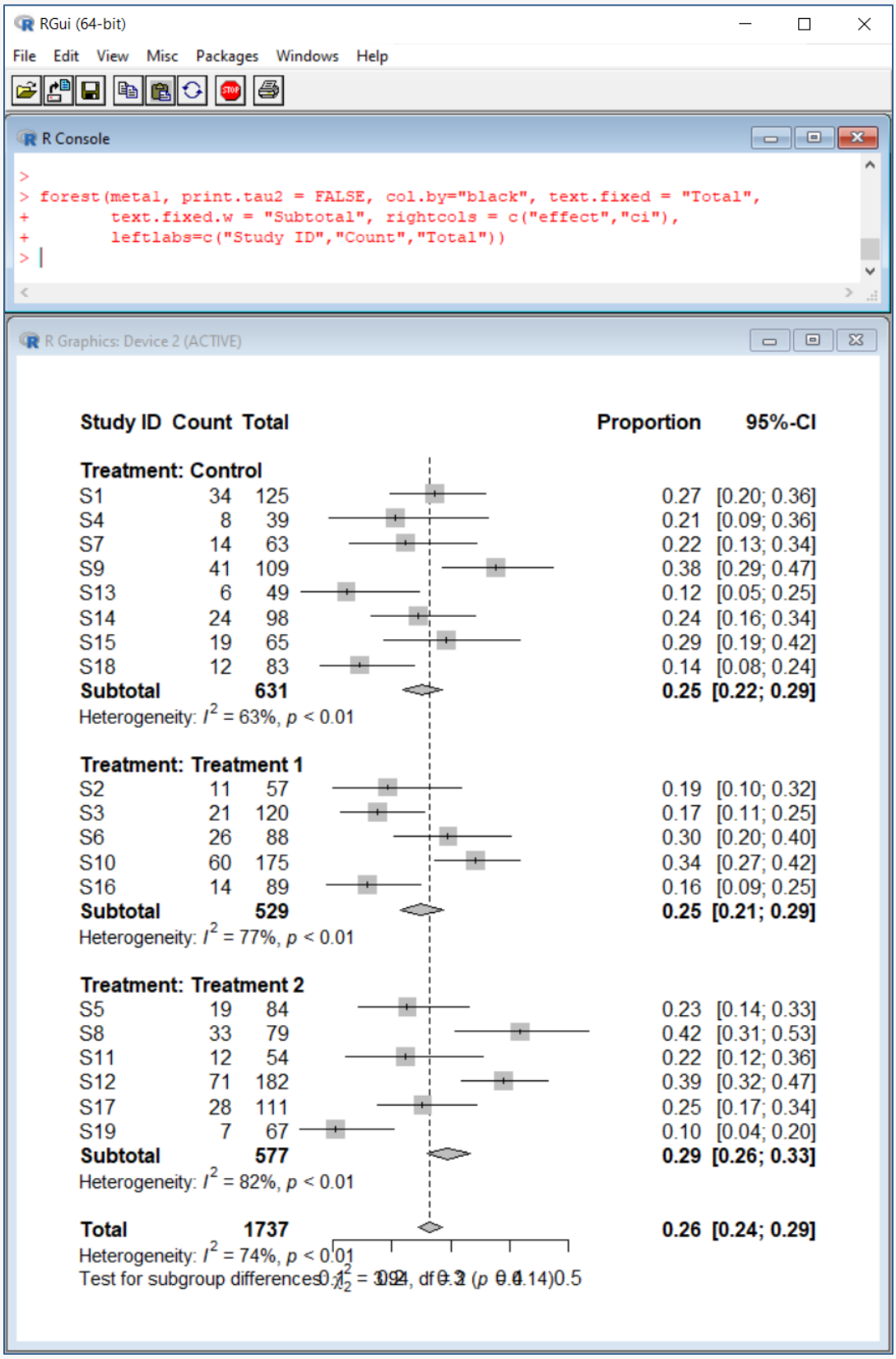
```
### Install the meta and metafor packages
### If the meta and metafor packages have been installed previously, these lines may be skipped
install.packages("meta")
install.packages("metafor")

### Load the meta and metafor packages
library(meta)
library(metafor)

### Run the metaprop and forest functions of the packages
meta1 <- metaprop(Count, Total, StudyID, comb.random=F, byvar=Trt,
            bylab="Treatment", byseparator=": ", method="GLMM")

meta1

forest(meta1, print.tau2 = FALSE, col.by="black", text.fixed = "Total",
      text.fixed.w = "Subtotal", rightcols = c("effect","ci"),
      leftlabs=c("Study ID","Count","Total"))
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:

Exporting Data to R




```
> ### Load the meta and metafor packages
> library(meta)
Loading 'meta' package (version 4.19-1).
Type 'help(meta)' for a brief overview.
> library(metafor)
Loading required package: Matrix

Loading the 'metafor' package (version 3.0-2). For an
introduction to the package please type: help(metafor)

>
> ### Run the metaprop and forest functions of the packages
> metal <- metaprop(Count, Total, StudyID, comb.random=F, byvar=Trt,
+                bylab="Treatment", byseparator=": ", method="GLMM")
>
> metal
    proportion           95%-CI   Treatment
S1      0.2720 [0.1963; 0.3588]     Control
S2      0.1930 [0.1005; 0.3191] Treatment 1
S3      0.1750 [0.1117; 0.2550] Treatment 1
S4      0.2051 [0.0930; 0.3646]     Control
S5      0.2262 [0.1420; 0.3305] Treatment 2
S6      0.2955 [0.2029; 0.4022] Treatment 1
S7      0.2222 [0.1272; 0.3446]     Control
S8      0.4177 [0.3077; 0.5341] Treatment 2
S9      0.3761 [0.2852; 0.4740]     Control
S10     0.3429 [0.2729; 0.4183] Treatment 1
S11     0.2222 [0.1204; 0.3560] Treatment 2
S12     0.3901 [0.3188; 0.4650] Treatment 2
S13     0.1224 [0.0463; 0.2477]     Control
S14     0.2449 [0.1636; 0.3421]     Control
S15     0.2923 [0.1860; 0.4183]     Control
S16     0.1573 [0.0888; 0.2498] Treatment 1
S17     0.2523 [0.1746; 0.3435] Treatment 2
S18     0.1446 [0.0770; 0.2389]     Control
S19     0.1045 [0.0430; 0.2035] Treatment 2

Number of studies combined: k = 19
Number of observations: o = 1737
Number of events: e = 460

                  proportion           95%-CI
Fixed effect model     0.2648 [0.2446; 0.2861]

Quantifying heterogeneity:
 tau^2 = 0.1701; tau = 0.4124; I^2 = 73.6% [58.4%; 83.2%]; H = 1.94 [1.55; 2.44]

Test of heterogeneity:
     Q d.f.  p-value            Test
 68.07   18 < 0.0001        Wald-type
 73.86   18 < 0.0001 Likelihood-Ratio

Results for subgroups (fixed effect model):
                       k proportion           95%-CI     Q   I^2  tau^2
Treatment: Control     8     0.2504 [0.2181; 0.2857] 19.08 63.3% 0.1116
Treatment: Treatment 1 5     0.2495 [0.2145; 0.2882] 17.18 76.7% 0.1269
Treatment: Treatment 2 6     0.2946 [0.2588; 0.3331] 27.35 81.7% 0.2614
                        tau
Treatment: Control     0.3341
Treatment: Treatment 1 0.3562
Treatment: Treatment 2 0.5112

Test for subgroup differences (fixed effect model):
                 Q d.f.  p-value
Between groups  3.94    2   0.1398
Within groups  63.60   16 < 0.0001

Details on meta-analytical method:
- Random intercept logistic regression model
- Maximum-likelihood estimator for tau^2
- Logit transformation
- Clopper-Pearson confidence interval for individual studies
>
```



Many additional options and **R** functions can be used to perform a variety of meta-analyses in **R**.

Example 15 – Exporting Data from R

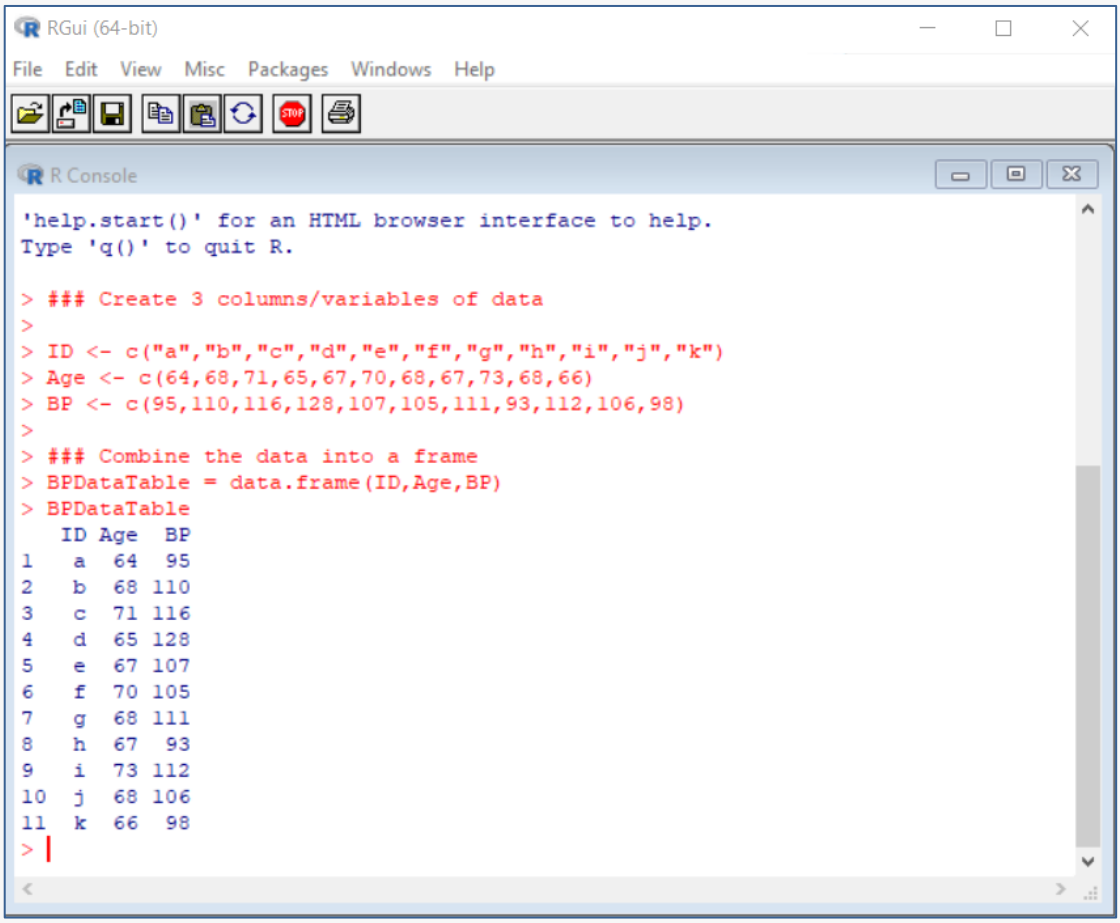
This example gives steps that can be used to export data from **R**. There are many ways to create and export data. This is only an example of one way. The steps to import the data into **NCSS** are also given.

R Code for Creating an Example Dataset

A data frame is created from scratch here, but it could be obtained as a result of an import, analysis, transformation, or something else.

```
### Create 3 columns/variables of data

ID <- c("a","b","c","d","e","f","g","h","i","j","k")
Age <- c(64,68,71,65,67,70,68,67,73,68,66)
BP <- c(95,110,116,128,107,105,111,93,112,106,98)

### Combine the data into a frame

BPDataTable = data.frame(ID,Age,BP)
BPDataTable
```

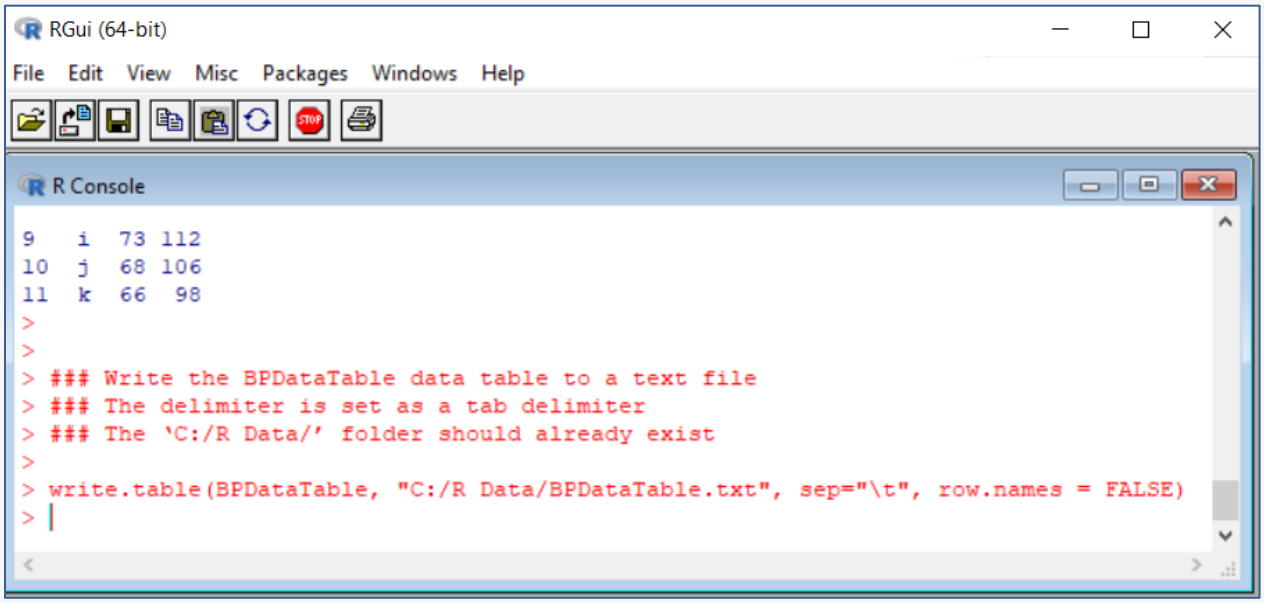
If this code is copied and pasted into **R**, the following is the result:



R Code for Exporting Data as a Text File

```
### Write the BPDataTable data table to a text file
### The delimiter is set as a tab delimiter
### The 'C:/R Data/' folder should already exist

write.table(BPDataTable, "C:/R Data/BPDataTable.txt", sep="\t", row.names = FALSE)
```

These lines of **R** code may be copied and pasted into the **R** Console to achieve the following result:



The BPDataTable.txt file has been created and put in the C:/R Data/ folder.

Importing the File into NCSS

To import the BPDataTable.txt file, complete the following steps:

1 Choose to Import a Text File

- From the File menu of the NCSS Data window, select **Import** and **Text File**.
- Select **BPDataTable.txt** and click **Open**.

2 Make the Appropriate Selections in the Import Wizard

- For Step 1, select Delimited. For Record Containing Column Names, enter 1. Click Next.
- For Step 2, select **Tab**. Click **Next**.
- For Step 3, click **Finish**.

The data have been imported into **NCSS**. It is usually recommended that the data be saved as a .NCSS file at this point.