A Brief Introduction to R

James G. MacKinnon

January, 2025

What is R?

What is R?

The R programming environment is a free and open-source programming environment. It includes

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;
- a large collection of built-in tools for data analysis;

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;
- a large collection of built-in tools for data analysis;
- the ability to create and display many types of graph;

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;
- a large collection of built-in tools for data analysis;
- the ability to create and display many types of graph;
- a programming language called “S”;

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;
- a large collection of built-in tools for data analysis;
- the ability to create and display many types of graph;
- a programming language called "S";
- a great many *packages* for statistics, data analysis, and machine learning, beyond the built-in ones.

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;
- a large collection of built-in tools for data analysis;
- the ability to create and display many types of graph;
- a programming language called "S";
- a great many *packages* for statistics, data analysis, and machine learning, beyond the built-in ones.

To begin, go to the Comprehensive R Archive Network (CRAN) at

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;
- a large collection of built-in tools for data analysis;
- the ability to create and display many types of graph;
- a programming language called "S";
- a great many *packages* for statistics, data analysis, and machine learning, beyond the built-in ones.

To begin, go to the Comprehensive R Archive Network (CRAN) at

`https://cran.r-project.org/`

What is R?

The R programming environment is a free and open-source programming environment. It includes

- the ability to store and manipulate data;
- the ability to perform many operations on matrices;
- a large collection of built-in tools for data analysis;
- the ability to create and display many types of graph;
- a programming language called "S";
- a great many *packages* for statistics, data analysis, and machine learning, beyond the built-in ones.

To begin, go to the Comprehensive R Archive Network (CRAN) at

`https://cran.r-project.org/`

Although the availability of many free packages is one of R's strengths, and the reason we are using it, they vary greatly in quality of implementation and documentation.

R began life as S at Bell Labs in the mid 1970s. It became available outside Bell Labs, mainly on Unix systems, in the early 1980s.

R began life as S at Bell Labs in the mid 1970s. It became available outside Bell Labs, mainly on Unix systems, in the early 1980s.

After the breakup of AT&T, a commercial version called S-PLUS largely replaced the free version in the late 1980s.

R began life as S at Bell Labs in the mid 1970s. It became available outside Bell Labs, mainly on Unix systems, in the early 1980s.

After the breakup of AT&T, a commercial version called S-PLUS largely replaced the free version in the late 1980s.

Originally, S was not object-oriented, but it moved in that direction as it was developed.

R began life as S at Bell Labs in the mid 1970s. It became available outside Bell Labs, mainly on Unix systems, in the early 1980s.

After the breakup of AT&T, a commercial version called S-PLUS largely replaced the free version in the late 1980s.

Originally, S was not object-oriented, but it moved in that direction as it was developed.

R was initially designed to run S programs, but it has now almost completely replaced S-PLUS. Work began in 1993, but Version 1.0.0 did not appear until 2000.

R began life as S at Bell Labs in the mid 1970s. It became available outside Bell Labs, mainly on Unix systems, in the early 1980s.

After the breakup of AT&T, a commercial version called S-PLUS largely replaced the free version in the late 1980s.

Originally, S was not object-oriented, but it moved in that direction as it was developed.

R was initially designed to run S programs, but it has now almost completely replaced S-PLUS. Work began in 1993, but Version 1.0.0 did not appear until 2000.

R packages may be written in R itself, or in languages like C, C++, Fortran, or Julia. The computational parts of modern packages like DDML are often the same for the R and Python versions.

R began life as S at Bell Labs in the mid 1970s. It became available outside Bell Labs, mainly on Unix systems, in the early 1980s.

After the breakup of AT&T, a commercial version called S-PLUS largely replaced the free version in the late 1980s.

Originally, S was not object-oriented, but it moved in that direction as it was developed.

R was initially designed to run S programs, but it has now almost completely replaced S-PLUS. Work began in 1993, but Version 1.0.0 did not appear until 2000.

R packages may be written in R itself, or in languages like C, C++, Fortran, or Julia. The computational parts of modern packages like DDML are often the same for the R and Python versions.

At the moment, the current version is 4.4.2.

There are at least two free on-line books called *An Introduction to R*.

There are at least two free on-line books called *An Introduction to R*.

One is by W. N. Venables, D. M. Smith, and the R Core Team. It is available from CRAN and has both online and PDF versions.

There are at least two free on-line books called *An Introduction to R*.

One is by W. N. Venables, D. M. Smith, and the R Core Team. It is available from CRAN and has both online and PDF versions.

The other is by Alex Douglas, Deon Roos, Francesca Mancini, Ana Couto, and David Lusseau. It has a 2024 date and no PDF.

There are at least two free on-line books called *An Introduction to R*.

One is by W. N. Venables, D. M. Smith, and the R Core Team. It is available from CRAN and has both online and PDF versions.

The other is by Alex Douglas, Deon Roos, Francesca Mancini, Ana Couto, and David Lusseau. It has a 2024 date and no PDF.

R is available for Windows, Mac, and Linux. It should not be hard to install it from CRAN.

There are at least two free on-line books called *An Introduction to R*.

One is by W. N. Venables, D. M. Smith, and the R Core Team. It is available from CRAN and has both online and PDF versions.

The other is by Alex Douglas, Deon Roos, Francesca Mancini, Ana Couto, and David Lusseau. It has a 2024 date and no PDF.

R is available for Windows, Mac, and Linux. It should not be hard to install it from CRAN.

Typing "R" in Linux, or clicking on the appropriate icon in Windows, should bring up an interactive window, where you can issue commands and see output.

There are at least two free on-line books called *An Introduction to R*.

One is by W. N. Venables, D. M. Smith, and the R Core Team. It is available from CRAN and has both online and PDF versions.

The other is by Alex Douglas, Deon Roos, Francesca Mancini, Ana Couto, and David Lusseau. It has a 2024 date and no PDF.

R is available for Windows, Mac, and Linux. It should not be hard to install it from CRAN.

Typing "R" in Linux, or clicking on the appropriate icon in Windows, should bring up an interactive window, where you can issue commands and see output.

Sequences of commands can also be stored in files, as can output.

There are at least two free on-line books called *An Introduction to R*.

One is by W. N. Venables, D. M. Smith, and the R Core Team. It is available from CRAN and has both online and PDF versions.

The other is by Alex Douglas, Deon Roos, Francesca Mancini, Ana Couto, and David Lusseau. It has a 2024 date and no PDF.

R is available for Windows, Mac, and Linux. It should not be hard to install it from CRAN.

Typing "R" in Linux, or clicking on the appropriate icon in Windows, should bring up an interactive window, where you can issue commands and see output.

Sequences of commands can also be stored in files, as can output.

Some people prefer to run R from RStudio, which is an IDE like Stata (XStata on Linux) and not open-source. See the Douglas book, or go to

There are at least two free on-line books called *An Introduction to R*.

One is by W. N. Venables, D. M. Smith, and the R Core Team. It is available from CRAN and has both online and PDF versions.

The other is by Alex Douglas, Deon Roos, Francesca Mancini, Ana Couto, and David Lusseau. It has a 2024 date and no PDF.

R is available for Windows, Mac, and Linux. It should not be hard to install it from CRAN.

Typing "R" in Linux, or clicking on the appropriate icon in Windows, should bring up an interactive window, where you can issue commands and see output.

Sequences of commands can also be stored in files, as can output.

Some people prefer to run R from RStudio, which is an IDE like Stata (XStata on Linux) and not open-source. See the Douglas book, or go to

`https://posit.co/download/rstudio-desktop/`

Some Features of R

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

All commands involve parentheses, which may be empty. For example,

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

All commands involve parentheses, which may be empty. For example,

```
plot(x,y)
```

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

All commands involve parentheses, which may be empty. For example,

```
plot(x,y)
```

creates a scatter plot of x and y, where these are vectors of the same length previously created or read in.

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

All commands involve parentheses, which may be empty. For example,

```
plot(x,y)
```

creates a scatter plot of x and y, where these are vectors of the same length previously created or read in.

```
q()
```

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

All commands involve parentheses, which may be empty. For example,

```
plot(x,y)
```

creates a scatter plot of x and y, where these are vectors of the same length previously created or read in.

```
q()
```

simply ends the current session. You will be asked whether or not to save the workspace.

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

All commands involve parentheses, which may be empty. For example,

```
plot(x,y)
```

creates a scatter plot of x and y, where these are vectors of the same length previously created or read in.

```
q()
```

simply ends the current session. You will be asked whether or not to save the workspace.

```
help(plot)
```

Some Features of R

R has an enormous number of built-in commands, plus many more added by various packages.

All commands involve parentheses, which may be empty. For example,

```
plot(x,y)
```

creates a scatter plot of x and y, where these are vectors of the same length previously created or read in.

```
q()
```

simply ends the current session. You will be asked whether or not to save the workspace.

```
help(plot)
```

provides information about the many options of the `plot()` command.

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

```
x <- rnorm(100)
y <- rnorm(100)
yols <- lm(y ~ x)
```

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

```
x <- rnorm(100)
y <- rnorm(100)
yols <- lm(y ~ x)
```

The last command here runs a linear regression of y on x, but it produces no output. To see some standard output, use

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

```
x <- rnorm(100)
y <- rnorm(100)
yols <- lm(y ~ x)
```

The last command here runs a linear regression of y on x, but it produces no output. To see some standard output, use

```
summary(yols)
```

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

```
x <- rnorm(100)
y <- rnorm(100)
yols <- lm(y ~ x)
```

The last command here runs a linear regression of y on x, but it produces no output. To see some standard output, use

```
summary(yols)
```

To save the fitted values and plot y against them, use

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

```
x <- rnorm(100)
y <- rnorm(100)
yols <- lm(y ~ x)
```

The last command here runs a linear regression of y on x, but it produces no output. To see some standard output, use

```
summary(yols)
```

To save the fitted values and plot y against them, use

```
yfit <- fitted.values(yols)
plot(y,yfit)
```

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

```
x <- rnorm(100)
y <- rnorm(100)
yols <- lm(y ~ x)
```

The last command here runs a linear regression of y on x, but it produces no output. To see some standard output, use

```
summary(yols)
```

To save the fitted values and plot y against them, use

```
yfit <- fitted.values(yols)
plot(y,yfit)
```

In most cases, we want to read in some data. If they are in a CSV file, we can use

By default, R commands don't print much output. Instead, they store results in an *object*, which can later be accessed in various ways.

```
x <- rnorm(100)
y <- rnorm(100)
yols <- lm(y ~ x)
```

The last command here runs a linear regression of y on x, but it produces no output. To see some standard output, use

```
summary(yols)
```

To save the fitted values and plot y against them, use

```
yfit <- fitted.values(yols)
plot(y,yfit)
```

In most cases, we want to read in some data. If they are in a CSV file, we can use

```
df <- read.csv("loan_data.csv")
```

Packages in R

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Ones that are new, experimental, or of low quality may not be there, and they may be harder to install.

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Ones that are new, experimental, or of low quality may not be there, and they may be harder to install.

To install the package `glmnet` from CRAN, use the command

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Ones that are new, experimental, or of low quality may not be there, and they may be harder to install.

To install the package `glmnet` from CRAN, use the command

```
install.packages("glmnet")
```

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Ones that are new, experimental, or of low quality may not be there, and they may be harder to install.

To install the package `glmnet` from CRAN, use the command

```
install.packages("glmnet")
```

This just has to be done once, unless the package (or R itself) gets updated and a reinstall is either desired or needed.

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Ones that are new, experimental, or of low quality may not be there, and they may be harder to install.

To install the package `glmnet` from CRAN, use the command

```
install.packages("glmnet")
```

This just has to be done once, unless the package (or R itself) gets updated and a reinstall is either desired or needed.

In order to use a package that has already been installed, use the `library` command:

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Ones that are new, experimental, or of low quality may not be there, and they may be harder to install.

To install the package `glmnet` from CRAN, use the command

```
install.packages("glmnet")
```

This just has to be done once, unless the package (or R itself) gets updated and a reinstall is either desired or needed.

In order to use a package that has already been installed, use the `library` command:

```
library(glmnet)
```

Packages in R

Most packages that anyone would want to use are available from CRAN and should be easy to install.

Ones that are new, experimental, or of low quality may not be there, and they may be harder to install.

To install the package `glmnet` from CRAN, use the command

```
install.packages("glmnet")
```

This just has to be done once, unless the package (or R itself) gets updated and a reinstall is either desired or needed.

In order to use a package that has already been installed, use the `library` command:

```
library(glmnet)
```

This has to be done in every session where the package is to be used.

If you are using several packages, it is annoying to have to enter several `library` commands.

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

```
source("startup.R")
```

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

```
source("startup.R")
```

will read the file `startup.R` and execute all the commands in it.

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

```
source("startup.R")
```

will read the file `startup.R` and execute all the commands in it.

This assumes that `startup.R` is in the working directory.

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

```
source("startup.R")
```

will read the file `startup.R` and execute all the commands in it.

This assumes that `startup.R` is in the working directory.

The `read.csv` command used above likewise assumes that the data file to be read is in the working directory.

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

```
source("startup.R")
```

will read the file `startup.R` and execute all the commands in it.

This assumes that `startup.R` is in the working directory.

The `read.csv` command used above likewise assumes that the data file to be read is in the working directory.

On Linux, the working directory is the one from which you start up R. On Windows, you can specify it.

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

```
source("startup.R")
```

will read the file `startup.R` and execute all the commands in it.

This assumes that `startup.R` is in the working directory.

The `read.csv` command used above likewise assumes that the data file to be read is in the working directory.

On Linux, the working directory is the one from which you start up R. On Windows, you can specify it.

The commands `getwd()` and `setwd()` respectively tell you what the working directory is and set a new one. For example,

If you are using several packages, it is annoying to have to enter several `library` commands.

A number of commands, not just `library` commands, can be put into a text file. Suppose it is called `startup.R`. Then the command

```
source("startup.R")
```

will read the file `startup.R` and execute all the commands in it.

This assumes that `startup.R` is in the working directory.

The `read.csv` command used above likewise assumes that the data file to be read is in the working directory.

On Linux, the working directory is the one from which you start up R. On Windows, you can specify it.

The commands `getwd()` and `setwd()` respectively tell you what the working directory is and set a new one. For example,

```
setwd("/home/jgm/papers/fastboot")
```

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

An `Rdata` file can contain multiple objects. The command

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

An `Rdata` file can contain multiple objects. The command

```
load("survey.rdata")
```

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

An `Rdata` file can contain multiple objects. The command

```
load("survey.rdata")
```

loads all of the objects in the file `survey.data` into memory.

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

An `Rdata` file can contain multiple objects. The command

```
load("survey.rdata")
```

loads all of the objects in the file `survey.data` into memory.

An `Rds` file can contain only one object, which can optionally be renamed when it is loaded:

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

An `Rdata` file can contain multiple objects. The command

```
load("survey.rdata")
```

loads all of the objects in the file `survey.data` into memory.

An `Rds` file can contain only one object, which can optionally be renamed when it is loaded:

```
newdata <- readRDS("survey.rdata")
```

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

An `Rdata` file can contain multiple objects. The command

```
load("survey.rdata")
```

loads all of the objects in the file `survey.data` into memory.

An `Rds` file can contain only one object, which can optionally be renamed when it is loaded:

```
newdata <- readRDS("survey.rdata")
```

If the `foreign` package has been loaded, R can read files in various "foreign" formats, such as Stata `.dta` files. Here is an example:

Data do not have to be in .csv files. The `read.table` command can read tabular data with various separators, which have to be specified.

```
newdata <-read.table("survey.dat", header=TRUE, sep=" ")
```

Files can be included in a library, such as the `ISLR` library that you should install.

They can also be stored in files called `Rdata` (or `Rda`) and `Rds`.

An `Rdata` file can contain multiple objects. The command

```
load("survey.rdata")
```

loads all of the objects in the file `survey.data` into memory.

An `Rds` file can contain only one object, which can optionally be renamed when it is loaded:

```
newdata <- readRDS("survey.rdata")
```

If the `foreign` package has been loaded, R can read files in various "foreign" formats, such as Stata `.dta` files. Here is an example:

```
newdata <- read.dta("survey.dta")
```

Some packages require data to be stored in objects called data frames (or dataframes). For example

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

stores the variables `hp` and `hpfit` in a new data frame called `newframe`.

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

stores the variables `hp` and `hpfit` in a new data frame called `newframe`.

Once one or more varables has been stored in this way, they can be sorted by any of the variables:

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

stores the variables `hp` and `hpfit` in a new data frame called `newframe`.

Once one or more varables has been stored in this way, they can be sorted by any of the variables:

```
sortedframe <- newframe[order(hp),]
```

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

stores the variables `hp` and `hpfit` in a new data frame called `newframe`.

Once one or more varables has been stored in this way, they can be sorted by any of the variables:

```
sortedframe <- newframe[order(hp),]
```

Any variable (now in the new order) can then be extracted with commands like

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

stores the variables `hp` and `hpfit` in a new data frame called `newframe`.

Once one or more varables has been stored in this way, they can be sorted by any of the variables:

```
sortedframe <- newframe[order(hp),]
```

Any variable (now in the new order) can then be extracted with commands like

```
hpfit <- sortedframe$hpfit
```

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

stores the variables `hp` and `hpfit` in a new data frame called `newframe`.

Once one or more varables has been stored in this way, they can be sorted by any of the variables:

```
sortedframe <- newframe[order(hp),]
```

Any variable (now in the new order) can then be extracted with commands like

```
hpfit <- sortedframe$hpfit
```

If we just wanted to sort a single variable, we would not have needed the data frame.

Some packages require data to be stored in objects called data frames (or dataframes). For example

```
newframe <- data.frame(hp,hpfit)
```

stores the variables `hp` and `hpfit` in a new data frame called `newframe`.

Once one or more varables has been stored in this way, they can be sorted by any of the variables:

```
sortedframe <- newframe[order(hp),]
```

Any variable (now in the new order) can then be extracted with commands like

```
hpfit <- sortedframe$hpfit
```

If we just wanted to sort a single variable, we would not have needed the data frame.

Some packages seem to like the data to be stored in data frames, but others do not. The `lm` command allows you to specify a data frame using the `data` option

```
library(ISLR)
attach(Auto)
hp <- horsepower
gpm <- 1/mpg
```

```
library(ISLR)
attach(Auto)
hp <- horsepower
gpm <- 1/mpg

modquad <- lm(gpm ~ hp+I(hp^2))
summary(modquad)
fitquad <- fitted.values(modquad)
newquad <- data.frame(hp, fitquad)
newquad <- newquad[order(hp),]
```

```
library(ISLR)
attach(Auto)
hp <- horsepower
gpm <- 1/mpg

modquad <- lm(gpm ~ hp+I(hp^2))
summary(modquad)
fitquad <- fitted.values(modquad)
newquad <- data.frame(hp, fitquad)
newquad <- newquad[order(hp),]

modcubic <- lm(gpm ~ hp+I(hp^2)+I(hp^3))
summary(modcubic)
fitcubic <- fitted.values(modcubic)
newcubic <- data.frame(hp, fitcubic)
newcubic <- newcubic[order(hp),]
```

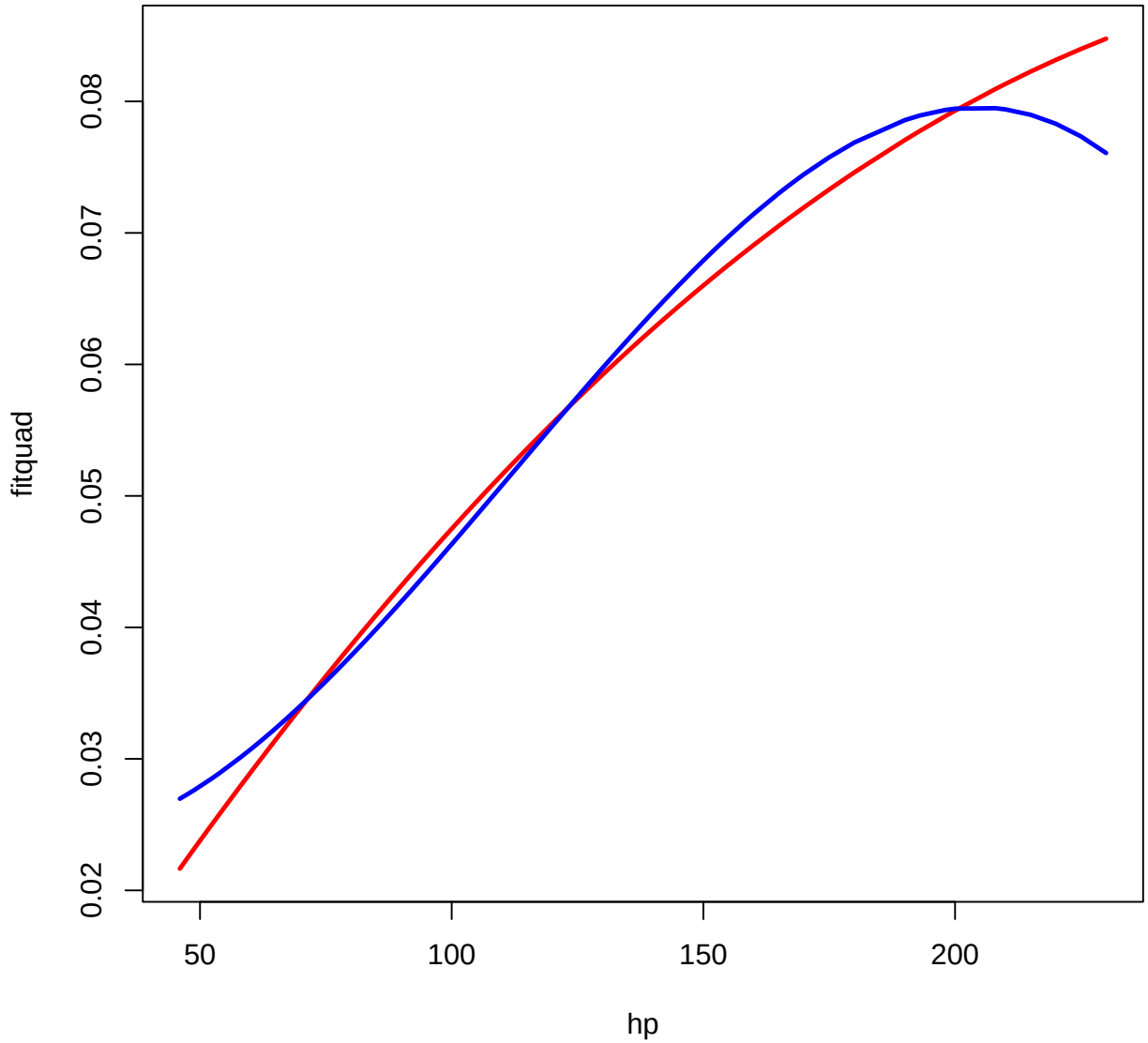
```
library(ISLR)
attach(Auto)
hp <- horsepower
gpm <- 1/mpg

modquad <- lm(gpm ~ hp+I(hp^2))
summary(modquad)
fitquad <- fitted.values(modquad)
newquad <- data.frame(hp, fitquad)
newquad <- newquad[order(hp),]

modcubic <- lm(gpm ~ hp+I(hp^2)+I(hp^3))
summary(modcubic)
fitcubic <- fitted.values(modcubic)
newcubic <- data.frame(hp, fitcubic)
newcubic <- newcubic[order(hp),]

plot(newquad,type="l",col="red",lwd=2.5)
lines(newcubic,col="blue",lwd=2.5)
```

```
Call:
lm(formula = gpm ~ hp + I(hp^2) + I(hp^3))
```

```
Call:
lm(formula = gpm ~ hp + I(hp^2) + I(hp^3))

Residuals:
      Min         1Q     Median         3Q        Max
-0.030018 -0.005080 -0.000989  0.005203  0.032184
```

```
Call:
lm(formula = gpm ~ hp + I(hp^2) + I(hp^3))

Residuals:
      Min        1Q    Median        3Q       Max
-0.030018 -0.005080 -0.000989  0.005203  0.032184

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.584e-02  8.655e-03   2.986 0.003004 **
hp          -2.101e-04  2.236e-04  -0.940 0.348012
I(hp^2)      5.908e-06  1.798e-06   3.286 0.001108 **
I(hp^3)     -1.759e-08  4.510e-09  -3.899 0.000114 ***
```

```
Call:
lm(formula = gpm ~ hp + I(hp^2) + I(hp^3))

Residuals:
      Min        1Q    Median        3Q       Max
-0.030018 -0.005080 -0.000989  0.005203  0.032184

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.584e-02  8.655e-03   2.986 0.003004 **
hp          -2.101e-04  2.236e-04  -0.940 0.348012
I(hp^2)      5.908e-06  1.798e-06   3.286 0.001108 **
I(hp^3)     -1.759e-08  4.510e-09  -3.899 0.000114 ***

Residual standard error: 0.008297 on 388 degrees of freedom
Multiple R-squared:  0.7533,Adjusted R-squared:  0.7514
F-statistic: 394.8 on 3 and 388 DF,  p-value: < 2.2e-16
```


fitquad
0.02
0.03
0.04
0.05
0.06
0.07
0.08
50
100
150
200
hp