COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

A Probability and Statistics Primer for Quantitative Finance

Week 2: R Usage

Jake Price
Instructor, Computational Finance and Risk Management
University of Washington
Slides originally produced by Kjell Konis

Outline

Outline

What is R?

R is a language and environment for statistical computing and graphics

R provides . . .

- a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, . . .) techniques
- a wide variety of graphical techniques
- and is highly extensible

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License (either Version 2 or Version 3)

R runs on Linux, Windows, and Mac OS

Installing R

The R installer is available from the *R Project* website

`http://www.r-project.org`

After installing R, run the command

```
> demo("graphics")
```

in the R Console as a quick check that things are working

What is R Studio?

R Studio is an integrated development environment (IDE) for R

R Studio includes ...

- a console
- a syntax-highlighting editor that supports direct code execution
- tools for plotting, history, debugging and workspace management

R Studio is available in an open source (i.e., free) edition – license = AGPL v3

R Studio also runs on Linux, Windows, and Mac OS

Installing R Studio

The R Studio installer is available from the R Studio website

`http://www.rstudio.com`

After installing R Studio, run the command

```
> demo("graphics")
```

in the R Studio Console as a quick check that things are working

Outline

The R Console

The > symbol is called the prompt: R is prompting you to enter a command

R as a calculator: input the command $1 + 1$ at the prompt and press return

```
> 1 + 1
```

The result appears on the next line

```
[1] 2
```

R can replace your 4 function calculator (R follows the usual order of operations)

+	addition	-	subtraction	^	power
*	multiplication	/	division		

Comments: any input following a # is ignored (until the next new line)

Mathematical Functions

R includes a wide variety of mathematical functions

For example, the `log` function returns the *natural* logarithm of its argument

```
> log(5)
[1] 1.609438
```

Some commonly used functions are given in the following table

`log`	logarithm	`exp`	exponential
`sin`	sine	`asin`	inverse sine
`cos`	cosine	`acos`	inverse cosine
`tan`	tangent	`atan`	inverse tangent
`sqrt`	square root		

Note: trigonometric functions use radians

Constants and Special Values

Constants

Exponential base: e^1

```
> exp(1)
[1] 2.718282
```

π

```
> pi
[1] 3.141593
```

Special Values

Plus and minus infinity

```
> 1/0
[1] Inf
> log(0)
[1] -Inf
```

Not a Number

```
> 0/0
[1] NaN
```

Caveats

Numerical calculations in R are done using *double precision* arithmetic

Double precision arithmetic is *not* real arithmetic

- There are a finite number of double precision numbers
- Input numbers are *coerced* to the most appropriate double precision number

Leads to *unexpected* behavior

```
> (1/3)^2 - 1 / (3^2)
[1] 0
```

But . . .

```
> (1/3)^3 - 1 / (3^3)
[1] -6.938894e-18
```

Variables

Values (e.g., the output of a function) can be saved in memory, this chunk of memory is called a *variable*

For example, suppose we want to evaluate $\sin(\frac{\pi}{2})$

First, evaluate $\frac{\pi}{2}$ and store the result

```
> u <- pi / 2
```

The assignment operator in R is `<-`, it assigns the value on the right to the variable on the left

Second, compute the value of sin(`u`)

```
> (v <- sin(u))
```

```
[1] 1
```

Variable Names

Variable names need/should conform to the following guidelines

- Variable names need to start with a character
- Variable names can contain characters, numbers, periods, and underscores
- Variable names cannot be an R reserved word

if	else	repeat	while	function
for	in	next	break	TRUE
FALSE	NULL	Inf	NaN	NA
...				

- Best practice: variable names should not be the name of a common R function
  - Some common collisions are `c`, `t`, `mean`, `var`, and `cov`

Workspace Management

Variables are stored in R's workspace

Use the `ls` function to list the variables in the workspace

```
> ls()
[1] "u" "v"
```

Use the `rm` function to remove variables from the workspace

```
> rm(u)
> ls()
[1] "v"
```

R asks to save the workspace when quitting, saved workspaces are automatically restored the next time R starts

Outline

What is an Array Programming Language?

An array programming language generalizes operations to apply transparently to single and multi-dimensional arrays

In R . . .

- a one dimensional array is called a *vector*
- a two dimensional array is called a *matrix*
- Higher dimensional arrays are possible but we are not going to use them

A goal of array programming is to provide a syntax similar to mathematical notation

- Very useful for *programming with data*

Vectors

A *vector* is an ordered collection of n values with the same underlying type

Example: suppose the vector $\mathbf{u} = (u_1, u_2)$ describes a point in the xy-plane

- Both u_1 and u_2 should be real numbers
- Since the first value defines the x coordinate and the second value the y coordinate, the vector (u_1, u_2) is (in general) different than the vector (u_2, u_1)

The number of values in the vector is called the *length* of the vector

Supported types include

- numeric
- character
- ...

Creating Vectors

The function `c` combines values into a vector

```
> (primes <- c(2, 3, 5, 7))
[1] 2 3 5 7
```

The function `length` returns the length of the input vector

```
> length(primes)
[1] 4
```

The function `seq` creates sequences of equally-spaced values

```
> seq(-1, 1, 0.5)
[1] -1.0 -0.5  0.0  0.5  1.0
```

Naming Vector Components

The = operator assigns a name to a value

Names can be given to the components of a vector

```
> wall.street.wiz <- c(houses = 3, cars = 5, boats = 2)
```

```
> wall.street.wiz
```

```
houses   cars  boats
     3      5      2
```

The `names` function returns the names of the components of a vector

```
> names(wall.street.wiz)
```

```
[1] "houses" "cars"   "boats"
```

Remark: assigning a name to a value (=) often has the same effect as assigning a value to a variable (<-) $\Longrightarrow$ = and <- can be used interchangeably

Matrices

A *matrix* is a two dimensional array of values (of the same type) described by a number of rows and a number of columns

The function `cbind` combines vectors (with the same length) into a matrix

```
> col1 <- c(11, 21, 31, 41)
> col2 <- c(12, 22, 32, 42)
> (mat <- cbind(col1, col2))
```

```
     col1 col2
[1,]   11   12
[2,]   21   22
[3,]   31   32
[4,]   41   42
```

Component-wise Operations on Vectors

What does it mean to

> *...generalize operations to apply transparently to single and multi-dimensional arrays*

When a vector is provided as the argument of a function, the output is a vector containing the function applied to each component of the input vector

```
> (x <- 1:9)  # x <- seq(1, 9, 1)
[1] 1 2 3 4 5 6 7 8 9
> sqrt(x)
[1] 1.000 1.414 1.732 2.000 2.236 2.449 2.646 2.828 3.000
```

Component-wise Composite Functions

Shifting: adding a length n vector and a length 1 vector

```
> x
[1] 1 2 3 4 5 6 7 8 9
> x + 2
[1]  3  4  5  6  7  8  9 10 11
```

Scaling: multiplying a length n vector by a length 1 vector

```
> 3*x
[1]  3  6  9 12 15 18 21 24 27
```

Element-wise Composite Functions

Functions act element-wise on matrices

```
> mat

     col1 col2
[1,]   11   12
[2,]   21   22
[3,]   31   32
[4,]   41   42


> 3*mat + 2

     col1 col2
[1,]   35   38
[2,]   65   68
[3,]   95   98
[4,]  125  128
```

Outline

Logical-Valued Functions

Logical values in R are `TRUE` and `FALSE`

Functions with a yes/no output generally return logical values

```
> x
[1] 1 2 3 4 5 6 7 8 9

> x > 4
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

> (x.gt.eq.4 <- x >= 4)
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

> x == 4  # Remember the double precision caveat
[1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Logical Vectors

The variable `x.gt.eq.4` is a logical vector, that is

- it is a vector
- the underlying type of value is logical

Lets create a second logical vector

```
> (x.lt.eq.7 <- x <= 7)
[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

Operations on Logical Vectors

Three key operations for logical vectors: `&` (AND), `|` (OR), and `!` (NOT)

Suppose that `x` and `y` are logical vectors with the same length

AND: `ans <- x & y`

$\texttt{ans}_i$ is `TRUE` if both $\texttt{x}_i$ and $\texttt{y}_i$ are `TRUE`, otherwise $\texttt{ans}_i$ is `FALSE`

```
> x.gt.eq.4
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> x.lt.eq.7
[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE

> x.gt.eq.4 & x.lt.eq.7
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

Operations on Logical Vectors (continued)

OR: `ans <- x | y`

`ans`$_i$ is `FALSE` if both `x`$_i$ and `y`$_i$ are `FALSE`, otherwise `ans`$_i$ is `TRUE`

```
> x.gt.eq.4
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> x.lt.eq.7
[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

```
> x.gt.eq.4 | x.lt.eq.7
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Operations on Logical Vectors (continued)

NOT: `ans <- !x`

`ans`$_i$ is `FALSE` if `x`$_i$ is `TRUE`, otherwise `ans`$_i$ is `TRUE`

```
> x.gt.eq.4
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
> !x.gt.eq.4
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

Subsetting Vectors

The subset operator in R is square brackets: `[]`

If

- `x` is a vector (with any underlying type)
- `idx` is a logical vector the same length as `x`

then `x[idx]` is a vector (with the same underlying type as `x`) containing the components of `x` for which `idx` is `TRUE`

```
> x[x.gt.eq.4]
[1] 4 5 6 7 8 9

> x[x.lt.eq.7]
[1] 1 2 3 4 5 6 7
```

Subsetting Vectors – More Examples

```
> x[x < 6]
[1] 1 2 3 4 5

> x[x > 2 & x < 8]
[1] 3 4 5 6 7

> x[x <= 2 | x >= 8]
[1] 1 2 8 9

> x[!(x == 3 | x == 6)]
[1] 1 2 4 5 7 8 9
```

Subsetting Vectors by Index

Suppose that `x` is a vector of length n

Let `idx` be a vector of values from the set $\{1, 2, \ldots, n\}$

The vector `ans <- x[idx]` is the same length as `idx` and has components

$$\texttt{ans}_i = \texttt{x}_{\texttt{idx}_i}$$

```
> primes
[1] 2 3 5 7

> primes[c(2, 4)]
[1] 3 7
```

From now on: write `ans[i]` to mean $\texttt{ans}_i$

Outline

R Packages

An R package is a collection of functions and data (and . . .)

$\Longrightarrow$ loading a package into your R session extends the capabilities of R

Packages loaded by default

```
> search()
```

```
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

For the curious: use the `find` function to see what package provides an object

```
> find("log")
```

```
[1] "package:base"
```

Loading R Packages

One useful function is the `fitdistr` function from the `MASS` package

`MASS` is a recommended package, it is already installed on your computer

The `library` function loads and attaches packages to the R session

```
> library(MASS)
```

The functions and data provided by the `MASS` package are now available to the R session

```
> search()
 [1] ".GlobalEnv"        "package:MASS"      "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

```
> find("fitdistr")
```

```
[1] "package:MASS"
```

Installing R Packages

The extensive collection of user-contributed packages is one of the reasons for the popularity of R

The Comprehensive R Archive Network (CRAN) is a clearing house for user-contributed packages

The `install.packages` function downloads packages from CRAN and installs them on your computer

The `getSymbols` function on the `quantmod` package will be the primary source of price data used in this course

```
> install.packages("quantmod")
```

The package must be loaded and attached to the R session

```
> library("quantmod")
> find("getSymbols")
[1] "package:quantmod"
```

Summary R Packages

Installing a package means downloading the package from CRAN and installing it on the computer

- Installing only needs to be done once (per computer)

Loading and attaching a package means telling the R session to look for functions (data, etc.) in the package

- Loading and attaching needs to be done in each R session

Outline

Getting Help

The `log` function returns the natural logarithm of its argument

How were we supposed to know that?

The `help` function displays the documentation associated with an object

```
> help(log)
```

Documentation is divided into sections including

Description	a description of the function or functions
Usage	how to call the function
Arguments	supported types of arguments
Details	additional explanation if needed
Value	a description of the returned object
See Also	related functions
Examples	

Description

```
`log' computes logarithms, by default natural logarithms, `log10'
computes common (i.e., base 10) logarithms, and `log2' computes
binary (i.e., base 2) logarithms.  The general form `log(x, base)'
computes logarithms with base `base'.

`log1p(x)' computes log(1+x) accurately also for |x| << 1.

`exp' computes the exponential function.

`expm1(x)' computes exp(x) - 1 accurately also for |x| << 1.
```

Usage

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

Arguments

```
   x: a numeric or complex vector.

base: a positive or complex number: the base with respect to which
      logarithms are computed.  Defaults to e=`exp(1)'.
```

Details

```
All except `logb' are generic functions: methods can be defined
for them individually or via the `Math' group generic.

`log10' and `log2' are only convenience wrappers, but logs to
bases 10 and 2 (whether computed _via_ `log' or the wrappers) will
be computed more efficiently and accurately where supported by the
OS.  Methods can be set for them individually (and otherwise
methods for `log' will be used).

`logb' is a wrapper for `log' for compatibility with S.  If (S3 or
S4) methods are set for `log' they will be dispatched.  Do not set
S4 methods on `logb' itself.

All except `log' are primitive functions.
```

Value

```
A vector of the same length as `x' containing the transformed
values.  `log(0)' gives `-Inf', and `log(x)' for negative values
of `x' is `NaN'.  `exp(-Inf)' is `0'.

For complex inputs to the log functions, the value is a complex
number with imaginary part in the range [-pi, pi]: which end of
the range is used might be platform-specific.
```

Outline

The `plot` Function

The `plot` function produces scatter plots

`Usage:`

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

The points are (x_i, y_i), the argument `x` is a vector containing the x values and `y` is a vector (the same length as `x`) containing the y values

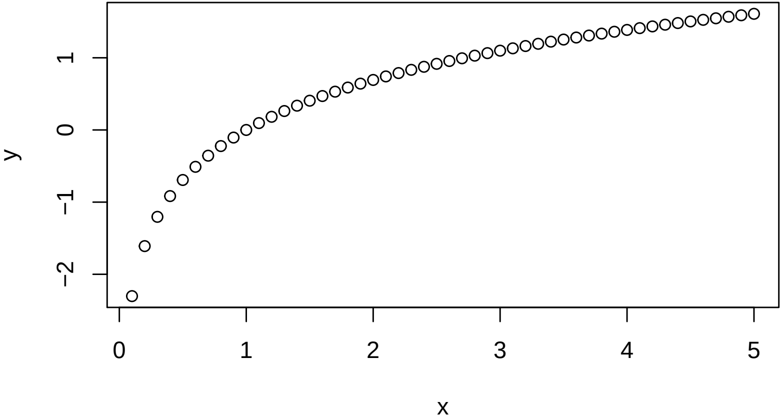
The `...` is a placeholder for optional arguments

Example:

```
> x <- seq(0.1, 5, 0.1)
> y <- log(x)
```

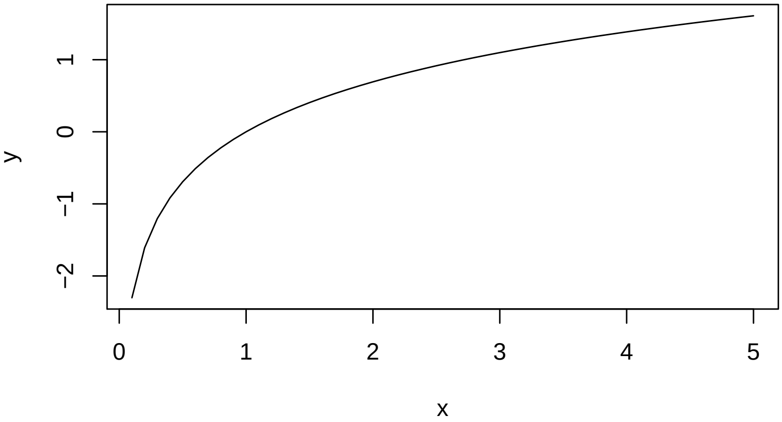
Plot with Defaults

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



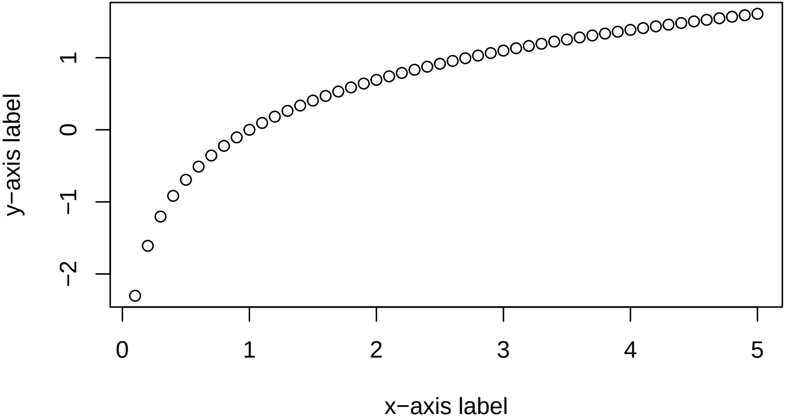
Optional Arguments: type

```
> plot(x, y, type = "l")
```



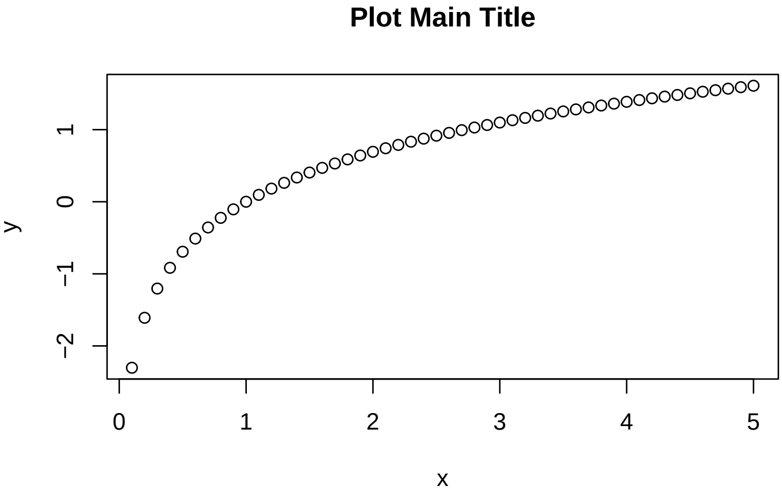
Optional Arguments: Axis Labels

```
> plot(x, y, xlab = "x-axis label", ylab = "y-axis label")
```



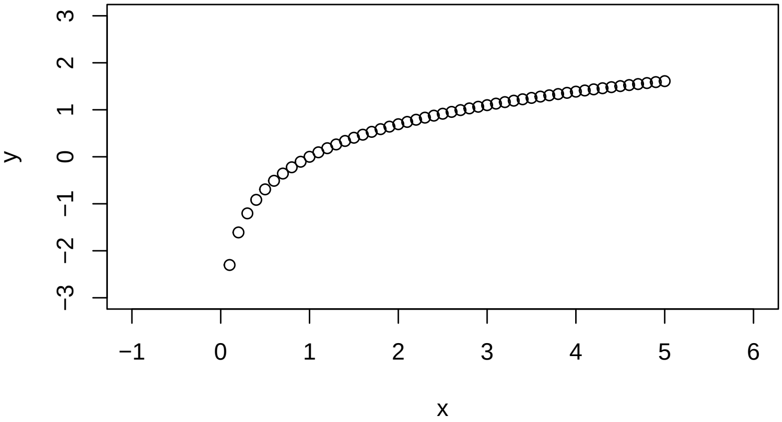
Optional Arguments: Title

```
> plot(x, y, main = "Plot Main Title")
```



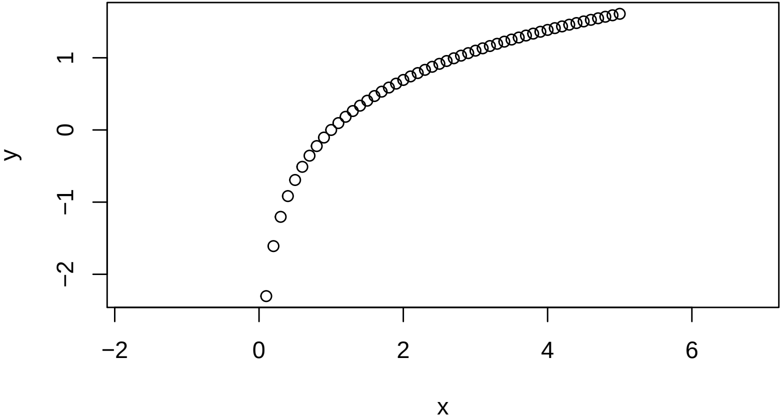
Optional Arguments: Plotting Region

```
> plot(x, y, xlim = c(-1, 6), ylim = c(-3, 3))
```



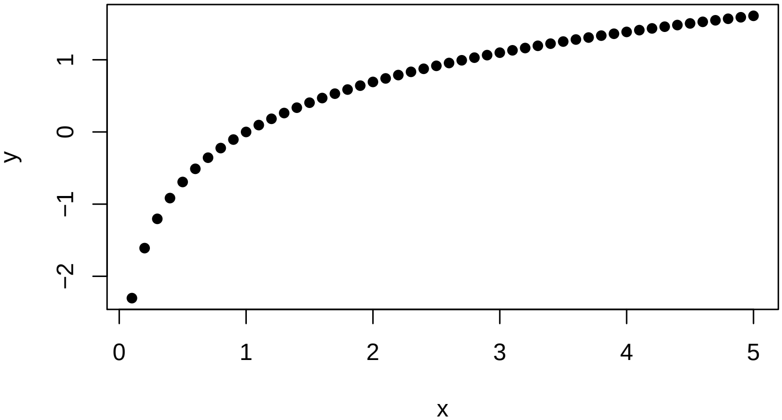
Optional Arguments: Setting the Aspect Ratio

```
> plot(x, y, asp = 1)
```



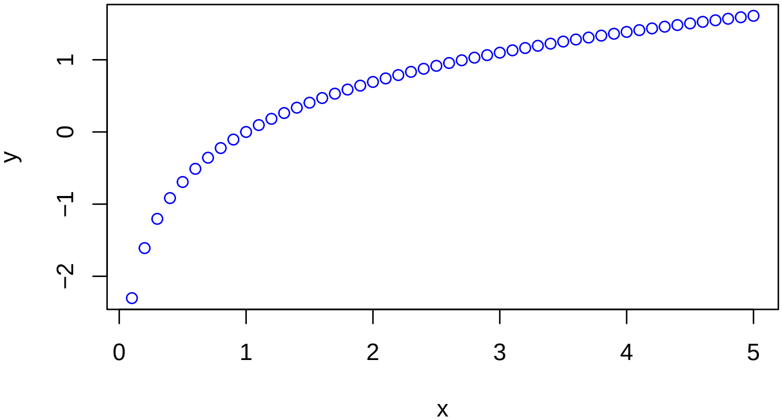
Optional Arguments: Plotting Symbol

```
> plot(x, y, pch = 16)
```



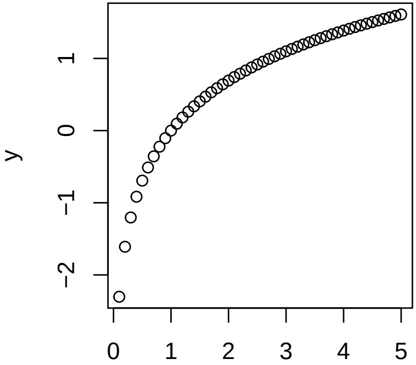
Optional Arguments: Color

```
> plot(x, y, col = "blue")
```



Square Plotting Region

```
> par(pty = "s")
> plot(x, y)
```



Object-Oriented Plotting

R is an object-oriented programming language $\Longrightarrow$ behavior of the plot function depends on what `x` is

If `x` is a vector then we get a scatter plot

If `x` is a financial prices series

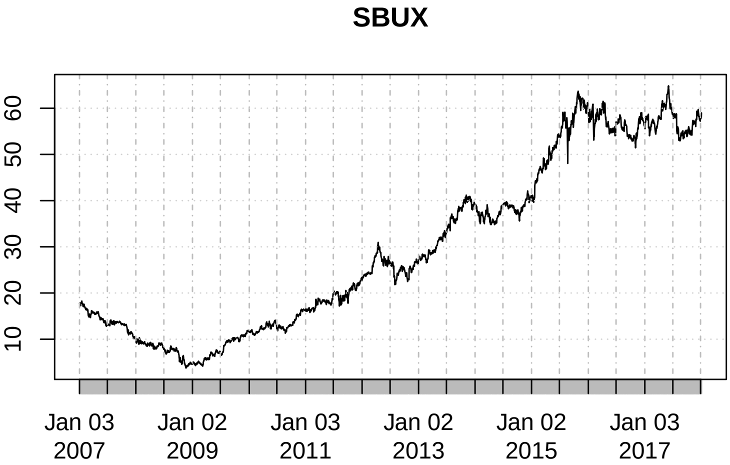
```
> getSymbols("SBUX")
```

```
[1] "SBUX"
```

The variable `SBUX` contains 10 years of price data for Starbucks

Plotting a Price Series

```
> plot(SBUX)
```



Outline

Additional Resources for Learning R

DataCamp: The easiest way to learn R programming and data science

`https://www.datacamp.com`

COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

http://computational-finance.uw.edu