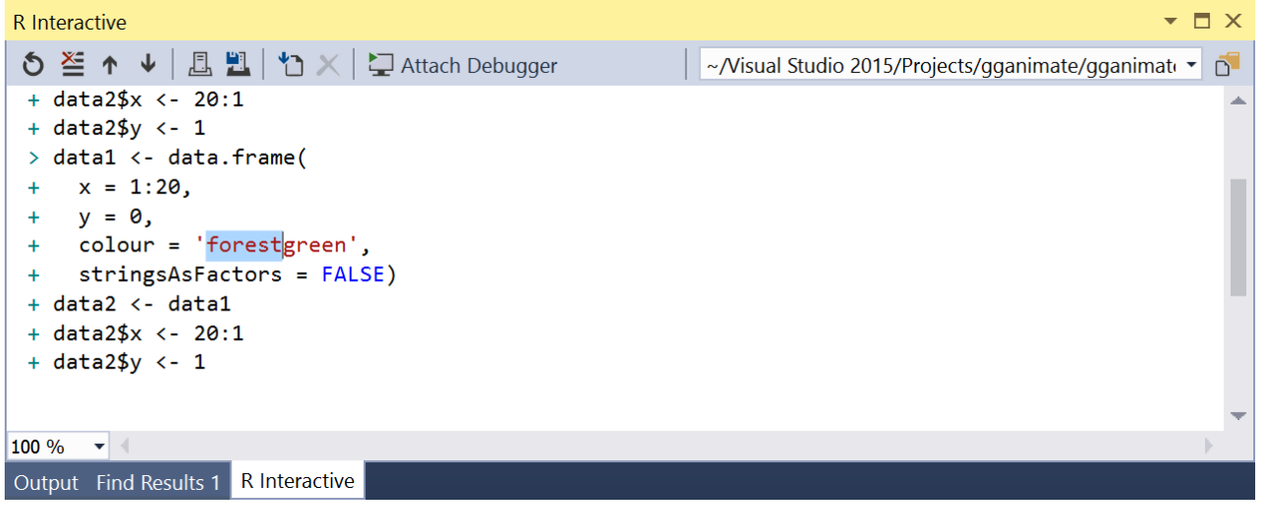
Microsoft

Business Analytics with Power BI

Microsoft Services

Module 3: Predictive Analytics with Power BI and R

Lesson 3: Introduction to R Language

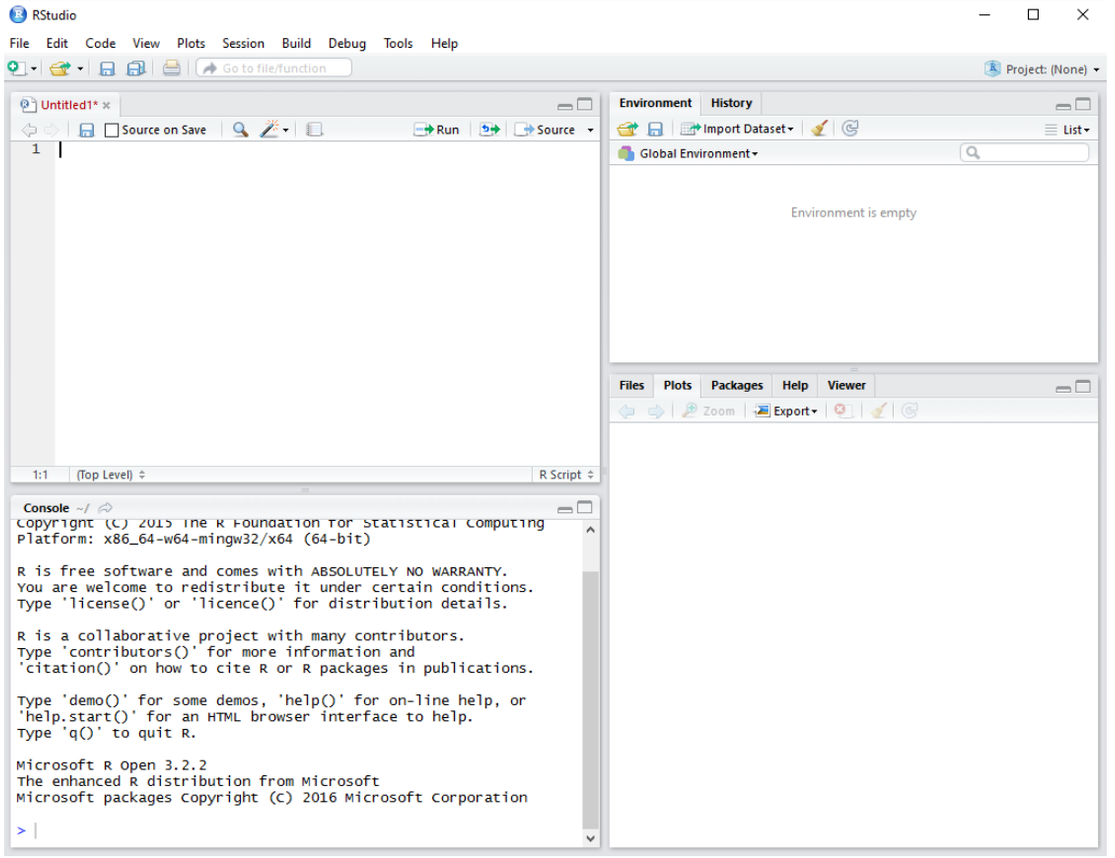


Introduction to R

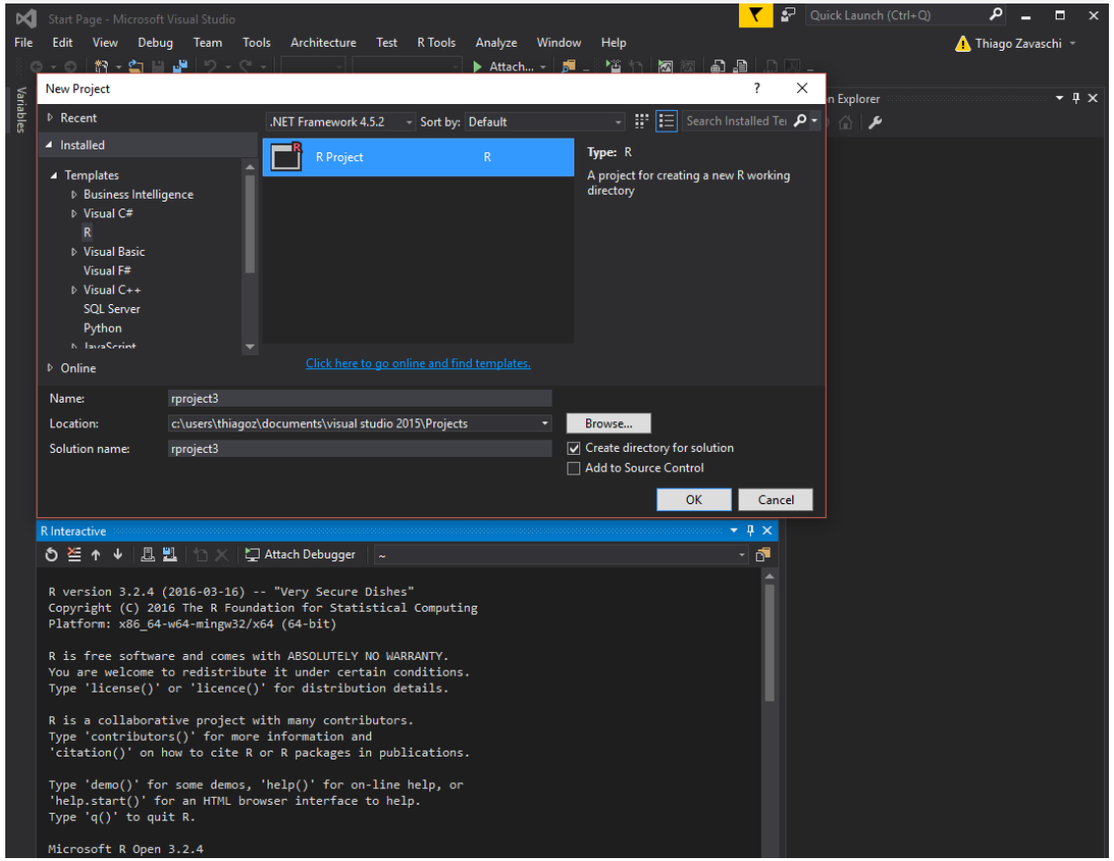
- "R" is a programming language (open-source from "S" language)
- Created by statisticians (from New Zealand)
- Simple to use, highly extensible, and cross-platform
  - Runs on Windows, Linux, and Mac OS X
- It has several packages developed by community
  - There are plenty of useful packages for Machine Learning scenarios
- Can create models (statistical computing) and powerful visualizations
- Script language (CL interface) but has several integrated development enviroments (IDEs) available

R IDEs

RStudio



R Tools for Visual Studio



R Basic Concepts

Variables, workspace, and comments

Variables are typed and you do not need to declare them prior to their use

Use "<-" (more common) or "=" to assign values.

```
> variable1 <- "valor"
> variable1
[1] "valor"
> var2 = 2
> var2
[1] 2
> x <- var2 + 1
> x
[1] 3
```

R and mathematical expressions

```
> a <- 1
> b <- 2
> a * b
[1] 2
> b^2
[1] 4
> b^3
[1] 8
> b^3/2*a
[1] 4
```

`ls()` returns all variables in the workspace

```
> ls()
[1] "a"         "b"         "var2"      "variable1" "x"
```

`rm()` removes a variable from memory

```
> rm(a)
> ls()
[1] "b"         "var2"      "variable1" "x"
```

Use # to comment code

```
# The below script does something
ls() #Returns all variables in the workspace
```

You can use `class()` to see the variable type

```
> l <- TRUE
> a <- 3
> class(l)
[1] "logical"
> class(a)
[1] "numeric"
```

R Basic Concepts

Getting help

- You can use the `help()` function or a "`?`" character to get help about a specific object or function
- Syntax: `help(`*`function`*`)` or `?`*`function`*

```
> help(mtcars)
> ?mtcars
```

mtcars {datasets} R Documentation

Motor Trend Car Road Tests

Description

The data was extracted from the 1974 *Motor Trend* US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

Usage

```
mtcars
```

Format

A data frame with 32 observations on 11 variables.

[, 1] mpg Miles/(US) gallon
[, 2] cyl Number of cylinders
[, 3] disp Displacement (cu.in.)
[, 4] hp Gross horsepower
[, 5] drat Rear axle ratio

R Basic Concepts

Getting help (cont..)

- If you want to search for a term inside the R documentation you can use "`??`" characters to search about it
- Syntax: `??term`

```
> ??mtcars
```

Demonstration: R Basics

Introduction to IDEs: R Basic commands

Vector

It is a data structure to hold a set of values of the same type. You can create a vector by using `c()` function

```
> x <- c(1, 3, 7)
> x
[1] 1 3 7
> y <- c(TRUE, F, T, FALSE)
> y
[1]  TRUE FALSE  TRUE FALSE
> z <- c("a","b","c")
> z
[1] "a" "b" "c"
```

If you try to create a vector with elements of different types all arguments are coerced to a common type

```
> a <- c("a", 1, TRUE)
> a
[1] "a"    "1"    "TRUE"
```

The members of a vector can have a name as well. This name can be used to access that value

```
> x <- c("one" = 1, "two" = 2)
> x
one two
  1   2
```

```
> names(x)
[1] "one" "two"
```

The index to access the values starts at 1

```
> x["one"]
one
  1
> x["two"]
two
  2
> x[2]
two
  2
```

Matrix

A matrix has the same properties as a vector. The difference is the number of dimensions.

Use the `matrix()` function to create a matrix.

```
> m <- matrix(1:10)
> m
      [,1]
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
```

```
> m1 <- matrix(1:10, ncol = 2)
> m1
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

Values filled by row

```
> matrix(1:12, nrow = 5, byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
[5,]    1    2    3
Warning message:
In matrix(1:12, nrow = 5, byrow = TRUE) :
  data length [12] is not a sub-multiple or multiple of the number of rows [5]
```

It continues to fill with the same values and throws a warning

Matrix

Accessing Data

Columns and rows can have names

```
> m2 <- matrix(c(4,3,2,17,18,19), nrow = 2, ncol = 3, byrow = TRUE,
 dimnames = list(c("row1", "row2"), c("C1", "C2", "C3")))
> m2
     C1 C2 C3
row1  4  3  2
row2 17 18 19


> m3 <- matrix(c(4,3,2,17,18,19), nrow = 2, ncol = 3, byrow = TRUE)
> m3
     [,1] [,2] [,3]
[1,]    4    3    2
[2,]   17   18   19
> rownames(m3) <- c("row1", "row2")
> colnames(m3) <- c("C1", "C2", "C3")
> m3
     C1 C2 C3
row1  4  3  2
row2 17 18 19
```

Accessing values

```
> m2[1,1]
[1] 4
> m3[1,1]
[1] 4
> m2["row1","C1"]
[1] 4
> m3["row1","C1"]
[1] 4
> m2[1,]
C1 C2 C3
 4  3  2
> m3[,1]
row1 row2
   4   17
> m3["row2",1]
[1] 17
> m3["row2",]
C1 C2 C3
17 18 19
```

Matrix

`as.matrix()` attempts to turn its argument into a matrix
`is.matrix()` tests if its argument is a matrix
There are similar functions for vectors (`as.vector()` / `is.vector()`)

```
> is.matrix(as.matrix(1:10))
[1] TRUE
> is.matrix(mtcars)  # It is a data.frame, not a matrix!
[1] FALSE
```

Factor

- The `factor()` function is used to encode a vector as a factor
- "Category" and "enumerated type" terms are used for factors too

- If argument "ordered" is TRUE, the factor levels are assumed to be ordered

```
> f1 <- factor(c("small", "medium", "large", "xlarge"))
> f1
[1] small  medium large  xlarge
Levels: large medium small xlarge
>
> f2 <- factor(c("small", "medium", "large", "xlarge"), ordered = TRUE)
> f2
[1] small  medium large  xlarge
Levels: large < medium < small < xlarge
```

List

- While vector and matrices do a good work, sometimes we need a structure to hold more than one data type
- To solve this need R implements lists
- There are a lot of similarities with vector but the data access is different

Use " [[ ]] " to access list values.

```
> x <- list(a = 1, b ="char", c = FALSE)
> x
$a
[1] 1

$b
[1] "char"

$c
[1] FALSE

> x[["b"]]
[1] "char"
> x[[3]]
[1] FALSE
> is.numeric(x[["a"]])
[1] TRUE
```

List

Lists can store another lists, vectors or matrices

```
> y <- list(v = c(1,2), m = matrix(1:6, nrow = 2), list(l = 1))
> y
$v
[1] 1 2

$m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

[[3]]
[[3]]$l
[1] 1


> y[["m"]]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> y[["m"]][2,3]
[1] 6
```

First you locate the list member and then you access the information in the same way you did before

Data Frame

- You might be thinking of creating a list of vectors to simulate a “data set”
- While this would work, the maintenance would be very difficult
- To solve this need, R implements data frames
- Data frame is the “data set structure” for R, where each column has a name and may be from different data types
- Data frames are more flexible than data sets implemented in another programming languages
- Data frame rows can have names too

Data Frame

R has several built-in data frames that you can use, or you can create your own

head() returns first 6 rows

str() shows the structure of a data frame

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

mtcars,
iris,
boston (MASS library),
etc.

Data Frame

data.frame()

To create your own data frame you should use the `data.frame()` function

Each parameter passed to the `data.frame()` function will become a column

```
> df1 <- data.frame(numbers = c(1,2,3))
> df1
  numbers
1       1
2       2
3       3
```

Column name

```
> df3 <- data.frame(1:4, "A", sample(LETTERS, 10))
Error in data.frame(1:4, "A", sample(LETTERS, 10)) :
  arguments imply differing number of rows: 4, 1, 10
> df3 <- data.frame(1:4, "A", sample(LETTERS, 4))
> df3
  X1.4 X.A. sample.LETTERS..4.
1    1    A                   P
2    2    A                   I
3    3    A                   U
4    4    A                   G
```

```
> df2 <- data.frame(numbers = c(1,2,3), Letter = "A", One = 1)
> df2
  numbers Letter One
1       1      A   1
2       2      A   1
3       3      A   1
```

Values are repeated to satisfy the rows number

It will be repeated only constant values

Data Frame – rbind()

Concatenate data frames

You can concatenate two different data frames using `rbind()` function.

Columns needs to have same names

```
> df1 <- data.frame(C1 = 1:4, C2 = "A", C3 = sample(LETTERS, 4))
> df2 <- data.frame(1:3, "B", sample(LETTERS, 3))
> names(df2) <- names(df1)
> df3 <- data.frame(1:3, 1, sample(LETTERS, 3))
> names(df3) <- names(df1)
> df <- rbind(df1, df2)
> df
  C1 C2 C3
1  1  A  K
2  2  A  U
3  3  A  W
4  4  A  R
5  1  B  Y
6  2  B  V
7  3  B  X
```

If desired rbind() can receive more than 2 parameters to concatenate

Data Frame – rbind()

Append data frames

You can concatenate two different data frames using `rbind()` function.

```
> df1 <- data.frame(C1 = 1:4, C2 = "A", C3 = sample(LETTERS, 4))
> df2 <- data.frame(1:3, "B", sample(LETTERS, 3))
> names(df2) <- names(df1)
> df3 <- data.frame(1:3, 1, sample(LETTERS, 3))
> names(df3) <- names(df1)
> df <- rbind(df1, df2, df3)
Warning message:
In `[<-.factor`(`*tmp*`, ri, value = c(1, 1, 1)) :
  invalid factor level, NA generated
> df
   C1   C2 C3
1   1    A  W
2   2    A  J
3   3    A  V
4   4    A  I
5   1    B  N
6   2    B  U
7   3    B  J
8   1 <NA>  D
9   2 <NA>  M
10  3 <NA>  S
```

`rbind()` in two incompatible data frames will result in <NA> generation plus a warning message.

Data Frame – cbind()

Add new columns

You can merge two different data frames using `cbind()` function. This works for matrices/vectors as well.

```
> m <- data.frame(cbind(1:5, 1))
> m
  X1 X2
1  1  1
2  2  1
3  3  1
4  4  1
5  5  1
> m <- cbind(m, L = c("A","B","C","D","E"))
> m
  X1 X2 L
1  1  1 A
2  2  1 B
3  3  1 C
4  4  1 D
5  5  1 E
```

The result from this cbind is a matrix that is converted into a data frame

Here we merge the existing data frame with a vector

Data Frame

Add new columns

You can create columns and return values from a single column.

The common way to do that is using $ sign.

```
> m <- data.frame(cbind(1:5, 1))
> m
  X1 X2
1  1  1
2  2  1
3  3  1
4  4  1
5  5  1
> m$L <- c("A","B","C","D","E")
> m
  X1 X2 L
1  1  1 A
2  2  1 B
3  3  1 C
4  4  1 D
5  5  1 E
```

Data Frame

Add new columns

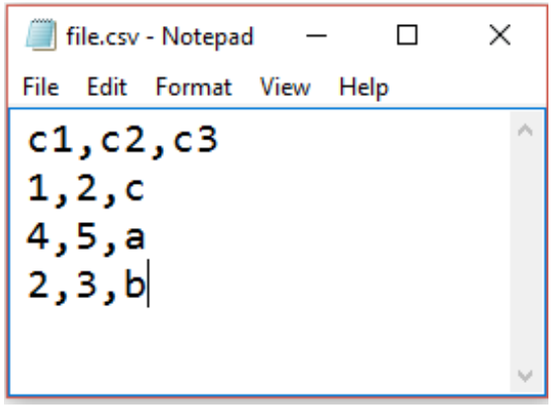
```
> m1 <- mtcars
> m1$new_column <- "A"
> head(m1)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb new_column
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4          A
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4          A
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1          A
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1          A
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2          A
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1          A
> m1$new_column
 [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
[22] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
> m1$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7
[18] 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```

Returning values from a single column

Data Frame

Create from Files

Data frames can be created from files (.csv's, etc.) or text.




```
> read.csv("C:\\temp\\file.csv")
  c1 c2 c3
1  1  2  c
2  4  5  a
3  2  3  b
> read.table(header = TRUE, text = "
+ a b
+ 1 2
+ 3 4
+ ")
  a b
1 1 2
2 3 4
```

Lab: Introduction to R

Exercise 01 – R Programming

Extending R with New Packages

You can use packages from community that are not included with R by default.
Most packages are available in CRAN (Comprehensive R Archive Network).

You can use `install.packages()` function that will install packages from CRAN*.

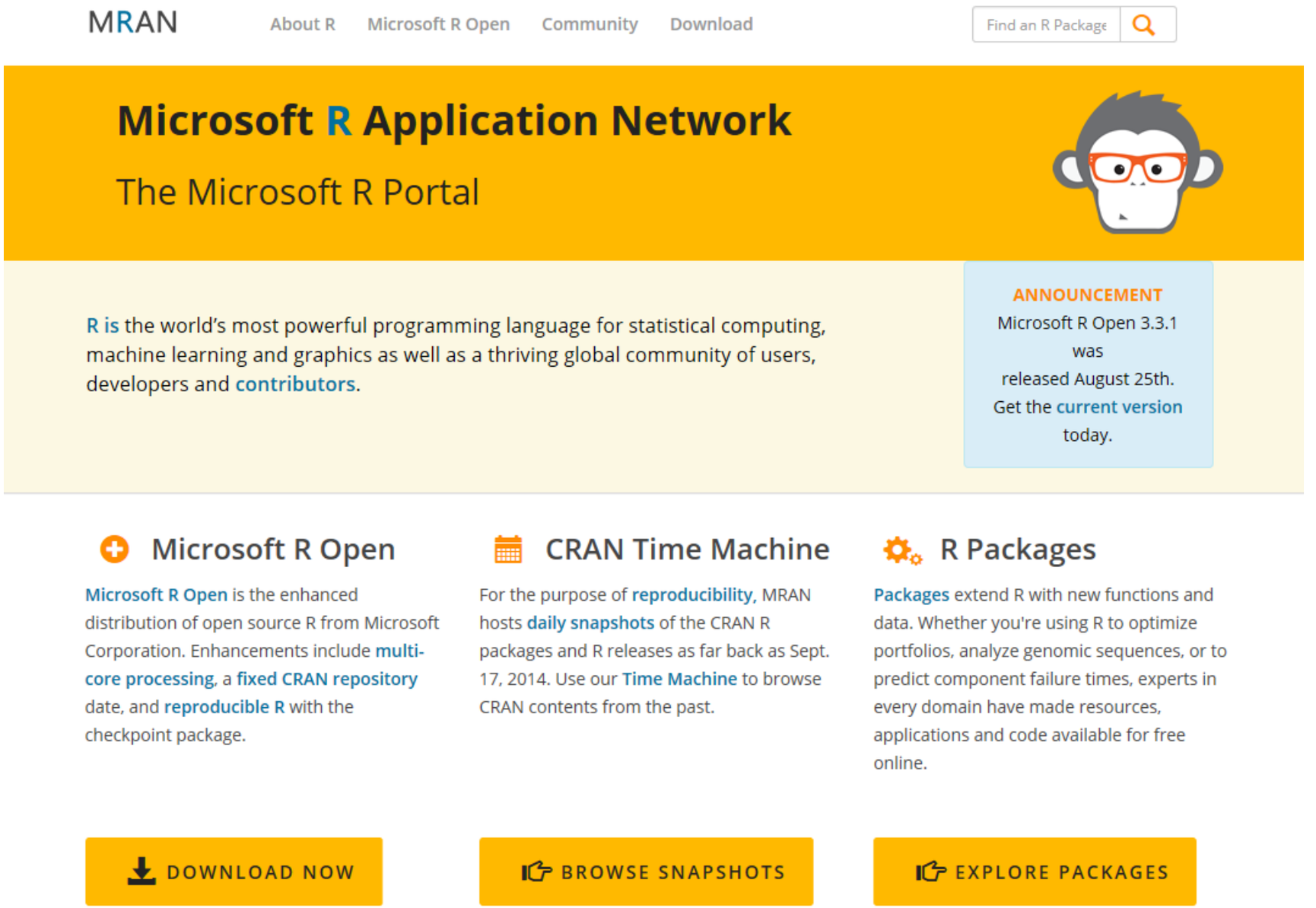
While CRAN may be the most used, there are other repositories, like MRAN from Microsoft. The `repos` parameter is used to change it.

*Internet access is required if the CRAN/MRAN is the desired repository.

Extending R with New Packages

```
Microsoft R Open 3.2.2
The enhanced R distribution from Microsoft
Microsoft packages Copyright (C) 2016 Microsoft Corporation

> install.packages("ggplot2")
Installing package into ‘C:/Users/thiagoz/Documents/R/win-library/3.2’
(as ‘lib’ is unspecified)
also installing the dependencies ‘Rcpp’, ‘RColorBrewer’, ‘dichromat’, ‘munsell’, ‘labeling’, ‘ply
r’, ‘gtable’, ‘reshape2’, ‘scales’, ‘proto’

trying URL 'https://mran.revolutionanalytics.com/snapshot/2015-11-30/bin/windows/contrib/3.2/Rcpp
_0.12.2.zip'
Content type 'application/zip' length 3195141 bytes (3.0 MB)
downloaded 3.0 MB

trying URL 'https://mran.revolutionanalytics.com/snapshot/2015-11-30/bin/windows/contrib/3.2/RCol
orBrewer_1.1-2.zip'
Content type 'application/zip' length 26681 bytes (26 KB)
downloaded 26 KB

trying URL 'https://mran.revolutionanalytics.com/snapshot/2015-11-30/bin/windows/contrib/3.2/dich
romat_2.0-0.zip'
Content type 'application/zip' length 147785 bytes (144 KB)
```

Extending R with New Packages - MRAN



Extending R with New Packages - CRAN

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

The Comprehensive R Archive Network

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
- Download R for (Mac) OS X
- 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 (Tuesday 2016-06-21, Bug in Your Hair) R-3.3.1.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

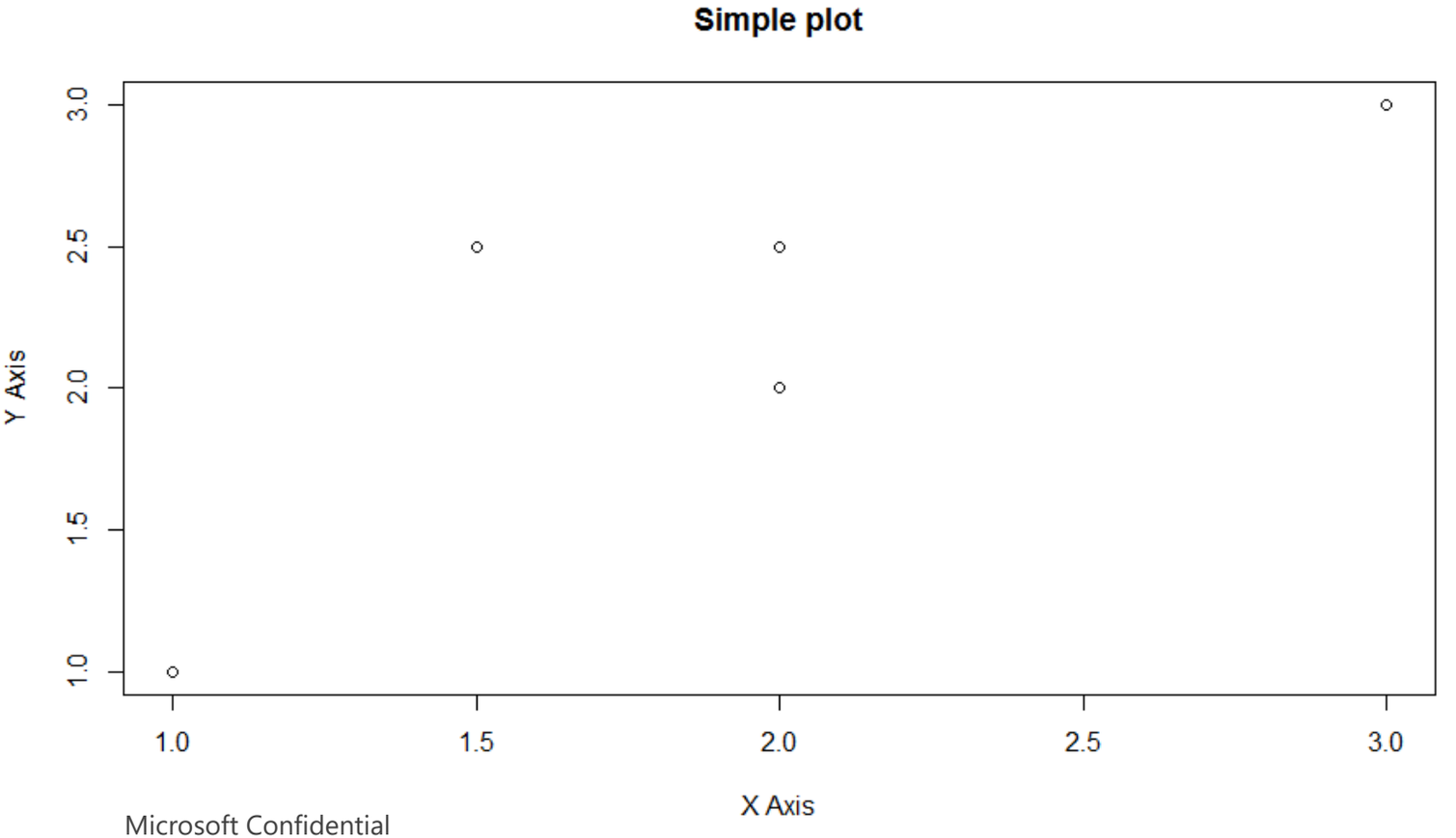
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

Plotting Data – plot()

The simplest plot function is `plot()`.

```
> plot(x= c(1,1.5,2,2,3), y = c(1,2.5,2,2.5,3), main = "Simple plot", xlab = "X Axis", ylab = "Y Axis")
```

First value from “x” with first value from “y”, second value from “x” with second value from “y”, and so on.



Plotting Data

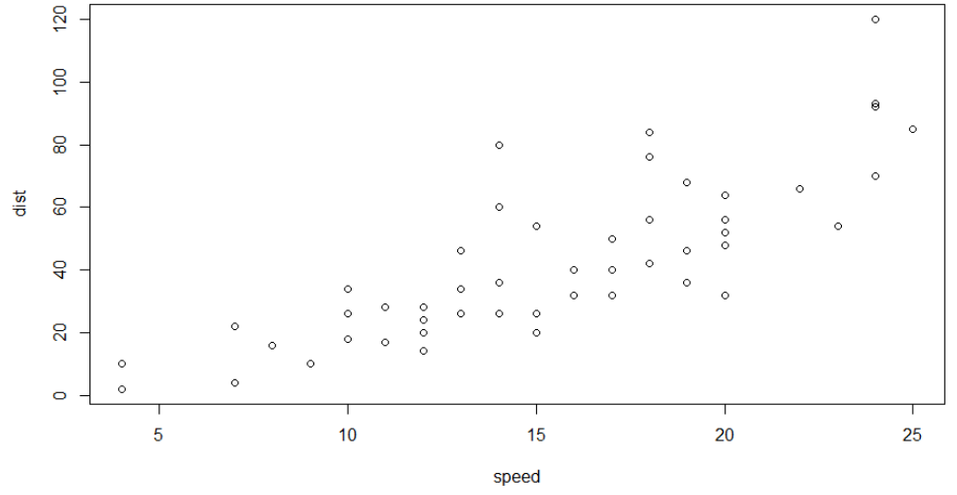
It is possible to add elements to the current plot.

```
> plot(cars)
> lines(lowess(cars))
```

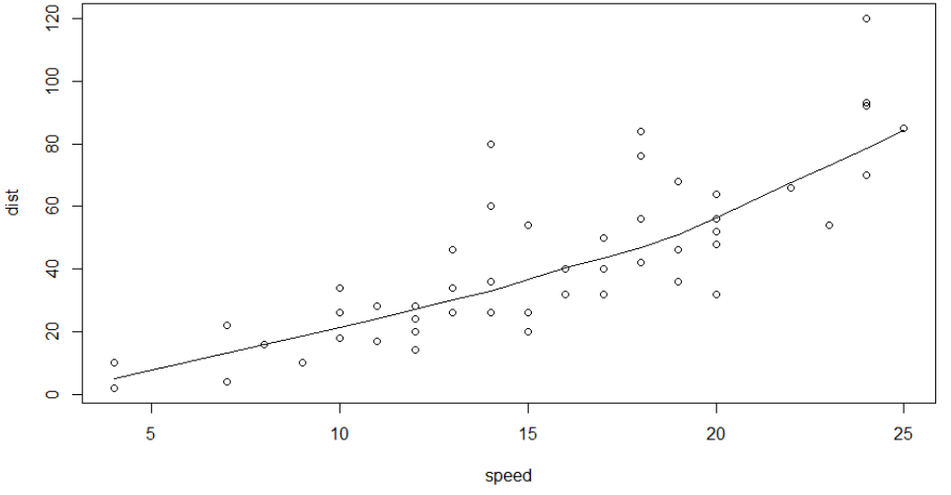
```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
```

Shows statistical info about a dataframe

After plot()



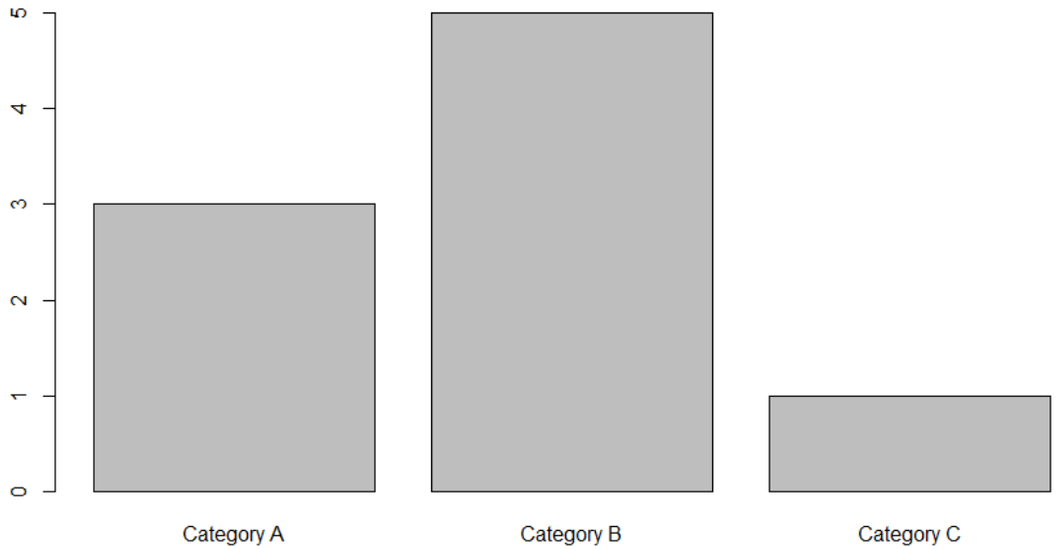
After lines()



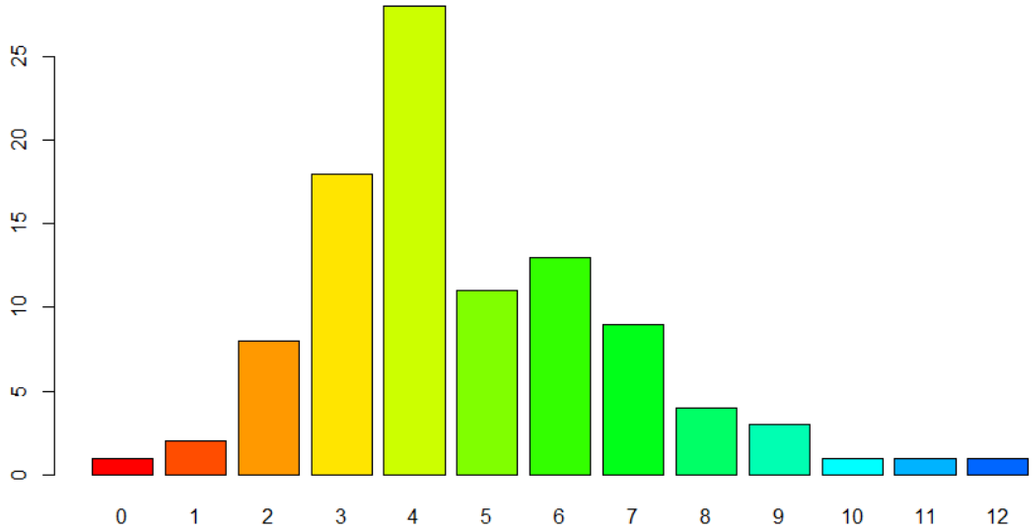
Plotting Data – barplot()

`barplot()` function is used to create bar charts.

```
> v <- c(3,5,1)
> names(v) <- c("Category A", "Category B", "Category C")
> barplot(v)
```



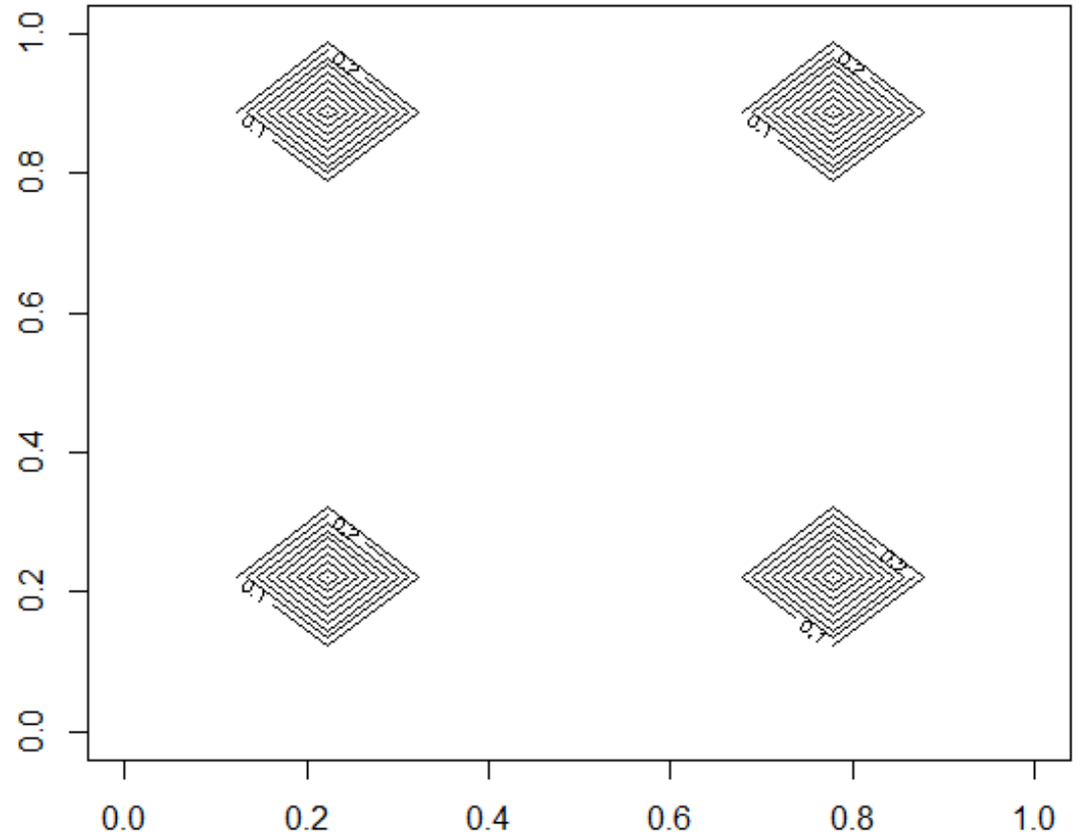

```
> tN <- table(Ni <- stats::rpois(100, lambda = 5))
> r <- barplot(tN, col = rainbow(20))
```



Plotting Data – contour()

```
> area <- matrix(0, 10, 10)
> area[c(3,8),c(3,9)] <- 1
> area
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0    0    0    0    0    0    0    0    0     0
 [2,]    0    0    0    0    0    0    0    0    0     0
 [3,]    0    0    1    0    0    0    0    0    1     0
 [4,]    0    0    0    0    0    0    0    0    0     0
 [5,]    0    0    0    0    0    0    0    0    0     0
 [6,]    0    0    0    0    0    0    0    0    0     0
 [7,]    0    0    0    0    0    0    0    0    0     0
 [8,]    0    0    1    0    0    0    0    0    1     0
 [9,]    0    0    0    0    0    0    0    0    0     0
[10,]    0    0    0    0    0    0    0    0    0     0
```

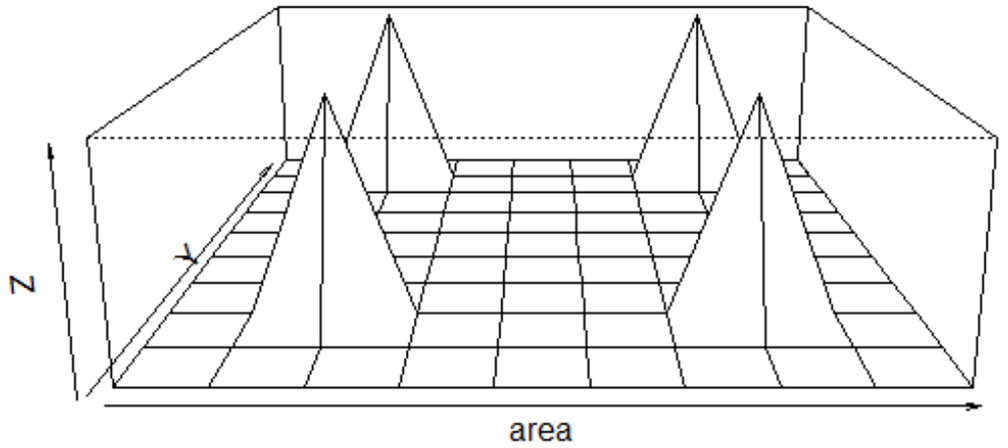
```
> contour(area)
```



Plotting Data – persp()

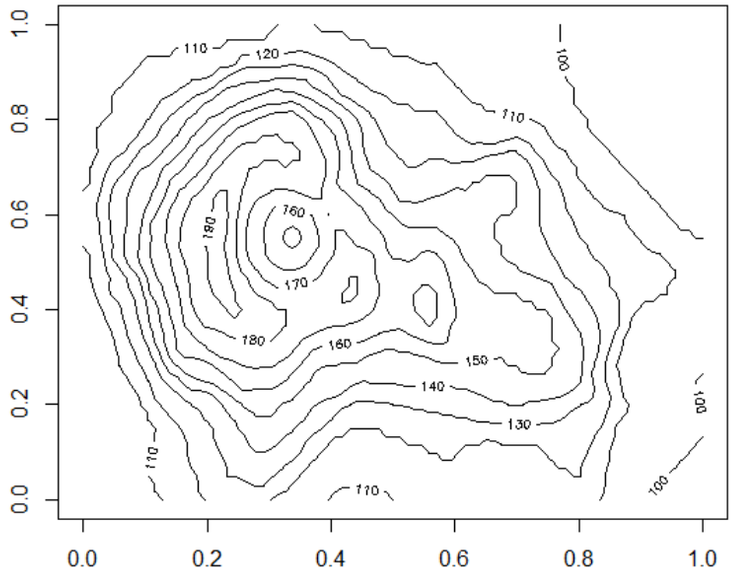
```
> area <- matrix(0, 10, 10)
> area[c(3,8),c(3,9)] <- 1
> area
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0    0    0    0    0    0    0    0    0     0
 [2,]    0    0    0    0    0    0    0    0    0     0
 [3,]    0    0    1    0    0    0    0    0    1     0
 [4,]    0    0    0    0    0    0    0    0    0     0
 [5,]    0    0    0    0    0    0    0    0    0     0
 [6,]    0    0    0    0    0    0    0    0    0     0
 [7,]    0    0    0    0    0    0    0    0    0     0
 [8,]    0    0    1    0    0    0    0    0    1     0
 [9,]    0    0    0    0    0    0    0    0    0     0
[10,]    0    0    0    0    0    0    0    0    0     0
```

```
> persp(area, expand=0.3)
```

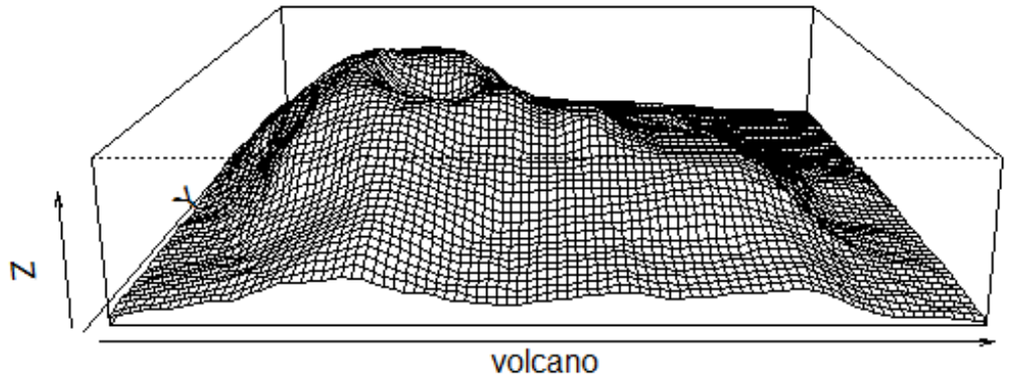


Plotting Data – volcano data frame

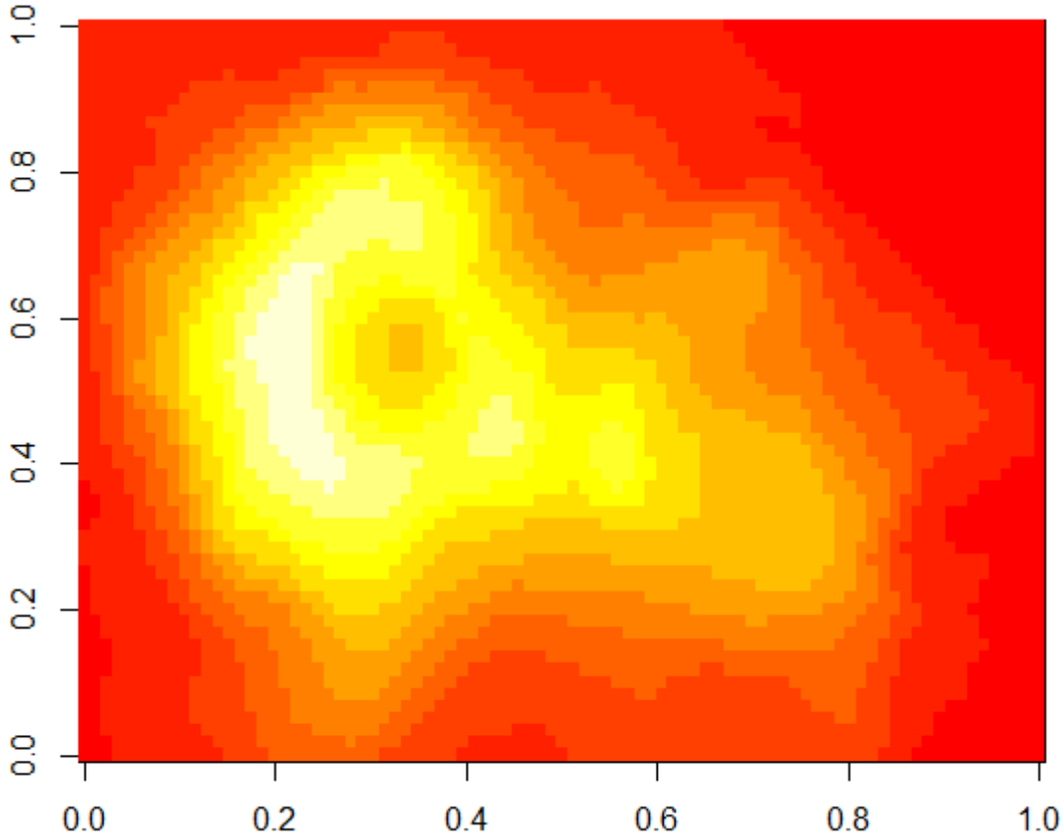
```
> contour(volcano)
```




```
> persp(volcano, expand = 0.2)
```




```
> image(volcano)
```



Plotting Data – corrplot()

One good way to see correlation between a data frame is using `cor()*` function. However it generates a matrix that is not easy to read. `Corrplot()` helps to show this information.

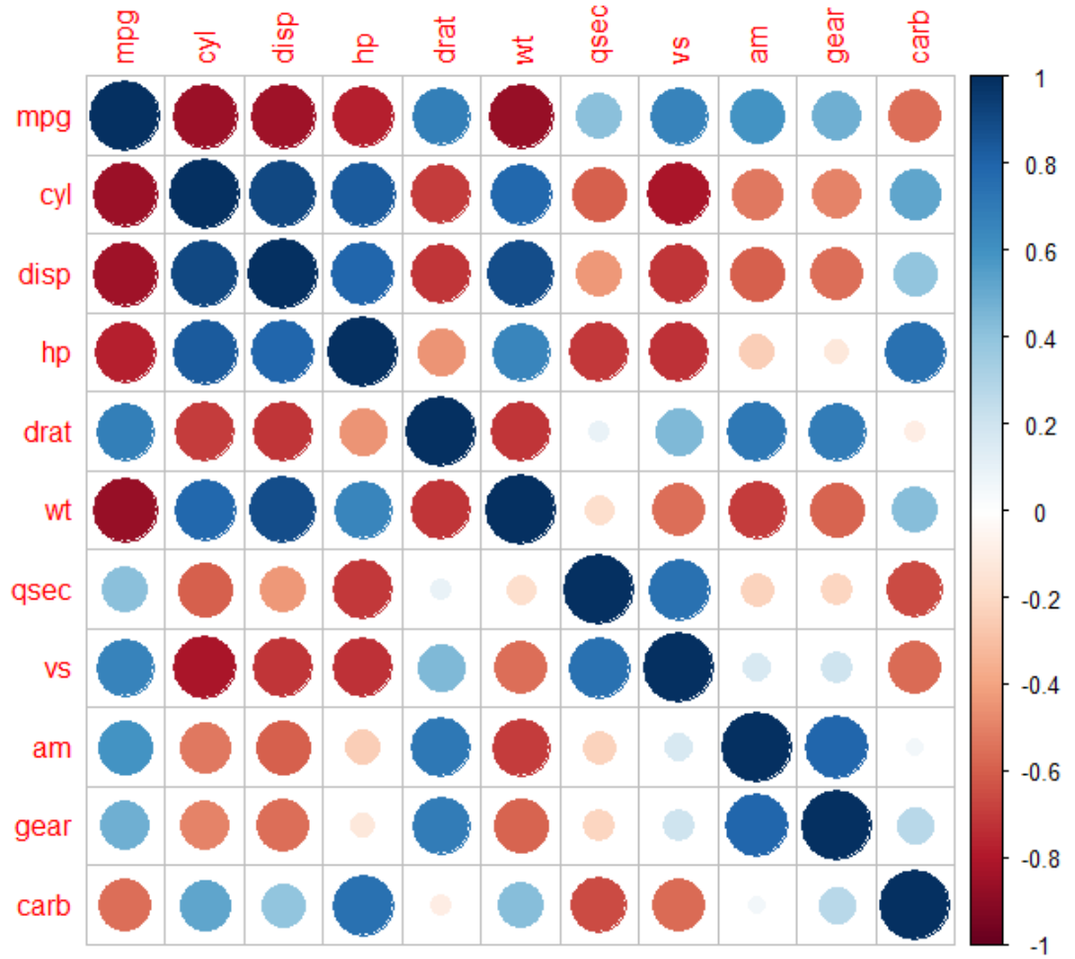
Correlation inside `mtcars` data frame:

```
> M <- cor(mtcars)
> M
             mpg        cyl       disp         hp        drat         wt        qsec         vs          am       gear        carb
mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684  0.68117191 -0.8676594  0.41868403  0.6640389  0.59983243  0.4802848 -0.55092507
cyl  -0.8521620  1.0000000  0.9020329  0.8324475 -0.69993811  0.7824958 -0.59124207 -0.8108118 -0.52260705 -0.4926866  0.52698829
disp -0.8475514  0.9020329  1.0000000  0.7909486 -0.71021393  0.8879799 -0.43369788 -0.7104159 -0.59122704 -0.5555692  0.39497686
hp   -0.7761684  0.8324475  0.7909486  1.0000000 -0.44875912  0.6587479 -0.70822339 -0.7230967 -0.24320426 -0.1257043  0.74981247
drat  0.6811719 -0.6999381 -0.7102139 -0.4487591  1.00000000 -0.7124406  0.09120476  0.4402785  0.71271113  0.6996101 -0.09078980
wt   -0.8676594  0.7824958  0.8879799  0.6587479 -0.71244065  1.0000000 -0.17471588 -0.5549157 -0.69249526 -0.5832870  0.42760594
qsec  0.4186840 -0.5912421 -0.4336979 -0.7082234  0.09120476 -0.1747159  1.00000000  0.7445354 -0.22986086 -0.2126822 -0.65624923
vs    0.6640389 -0.8108118 -0.7104159 -0.7230967  0.44027846 -0.5549157  0.74453544  1.0000000  0.16834512  0.2060233 -0.56960714
am    0.5998324 -0.5226070 -0.5912270 -0.2432043  0.71271113 -0.6924953 -0.22986086  0.1683451  1.00000000  0.7940588  0.05753435
gear  0.4802848 -0.4926866 -0.5555692 -0.1257043  0.69961013 -0.5832870 -0.21268223  0.2060233  0.79405876  1.0000000  0.27407284
carb -0.5509251  0.5269883  0.3949769  0.7498125 -0.09078980  0.4276059 -0.65624923 -0.5696071  0.05753435  0.2740728  1.00000000
```

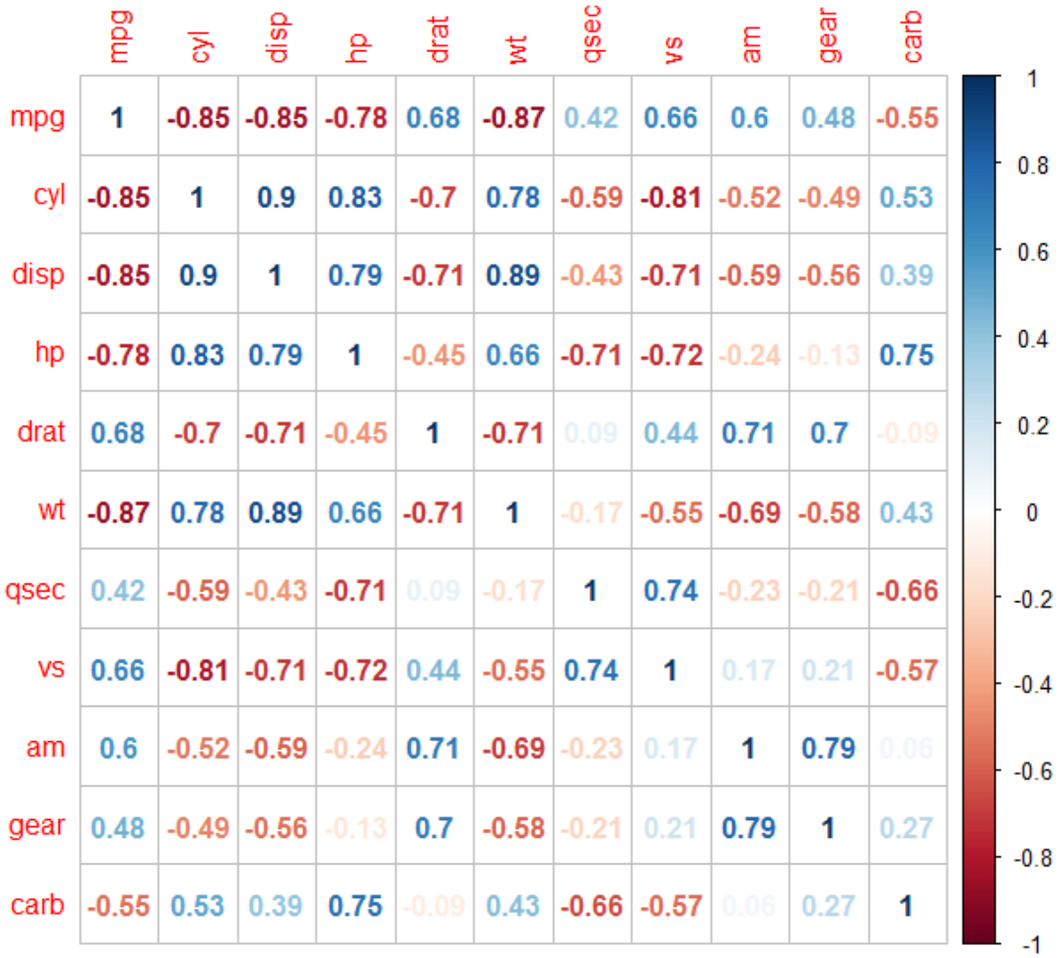
*Needs corrplot package

Plotting Data – corrplot()

```
> library(corrplot)
> corrplot(M)
```



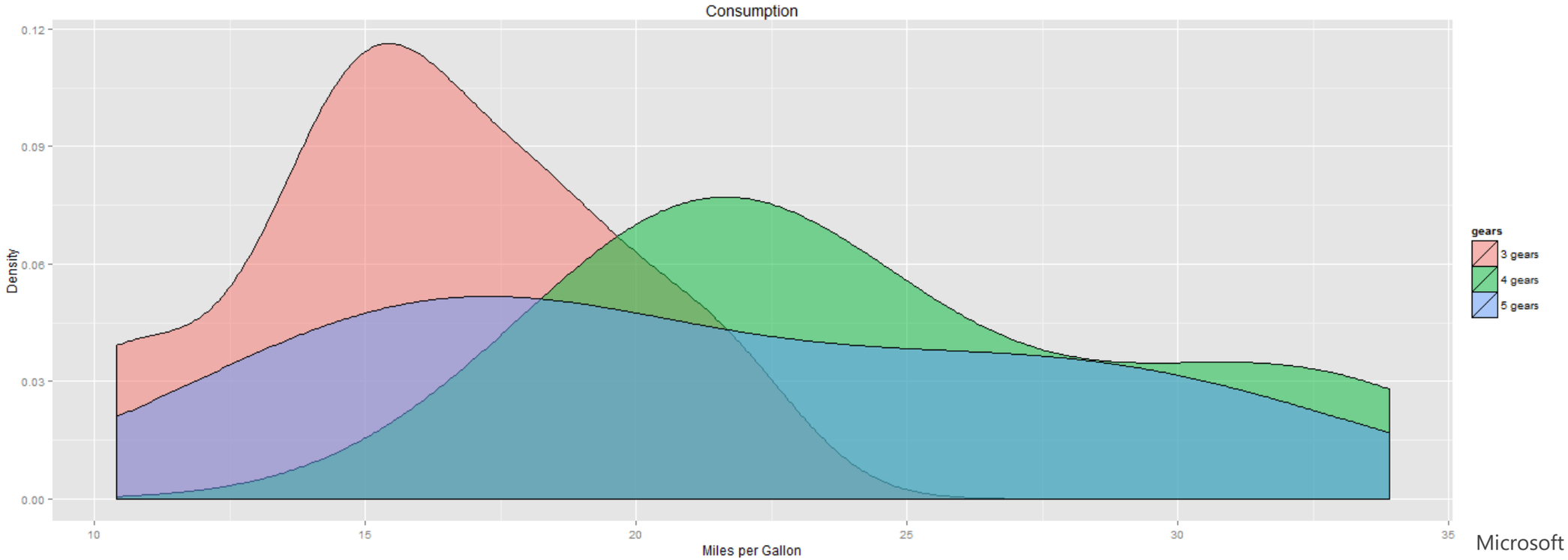

```
> library(corrplot)
> corrplot(M, method="number")
```



Plotting Data – ggplot2()

One of the most used graphics library is ggplot2, due to high power of customization that can be achieved.

```
> library(ggplot2)
> mt <- mtcars
> mt$gears <- factor(mt$gear,levels=c(3,4,5), labels=c("3 gears","4 gears","5 gears"))
> qplot(mpg, data=mt, geom="density", fill=gears, alpha=I(.5), main="Consumption", xlab="Miles per Gallon", ylab="Density")
```

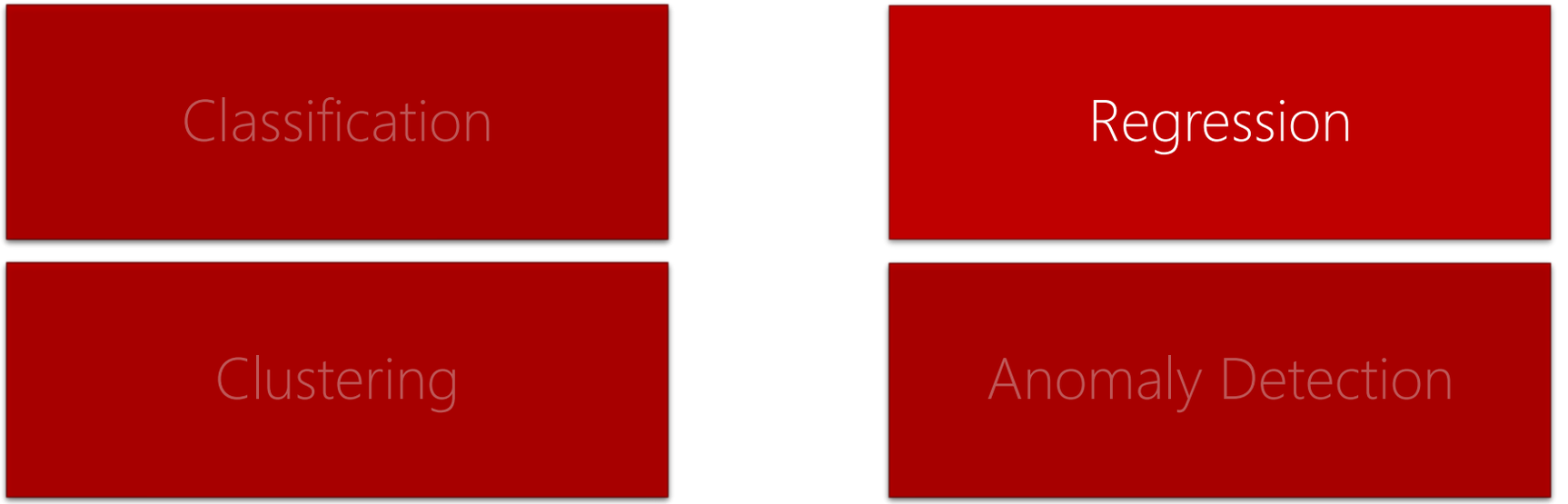


Lab: Introduction to R

Exercise 02 – Plotting Data

Creating Models with R

- R can be used to create different types of models
- Let's say that our problem is: **What is the oil consumption (mpg) of a specific car?**
- What kind of algorithms we can use to solve it?



Creating Models with R

We need to know/discover what features describes car consumption
The most important thing is: what features are really useful for our experiment?

This type of question is what usually is inside a data scientist head ☺

```
> tail(mtcars)
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

For our experiment, we are going to use just one column as a feature: wt (weight).
We need mpg as well (label)

Creating Models with R

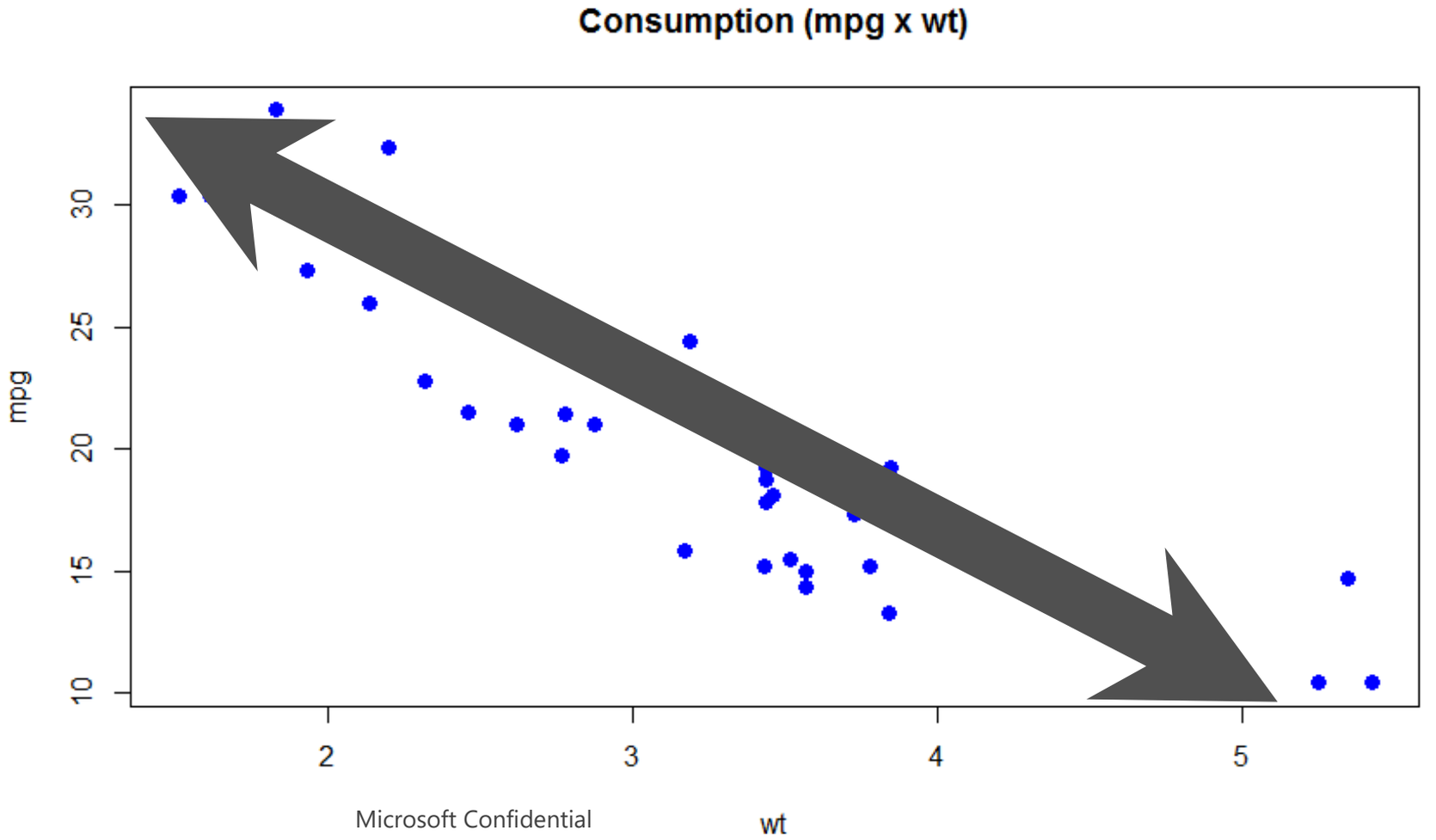
Linear Regression

- Linear regression it is the simplest regression model. We find the line equation that generalizes the problem
- A linear regression creates a line that explain a scalar dependent variable and one (our case) or more explanatory variables (features)
- Not all problems can fit well in a line. When we have a lot of variables, it is not possible to see in a chart. In these cases (real world) we need to use statistics to measure error to see if the model works well
- Our dataset (mtcars dataframe)
  - Dependent variable: mpg
  - Explanatory variable: wt

Creating Models with R

Linear Regression

```
> mt <- mtcars
> plot(mt$wt, mt$mpg, pch = 16, cex = 1.3, col = "blue", main = "Consumption (mpg x wt)", x
lab = "wt", ylab = "mpg")
```



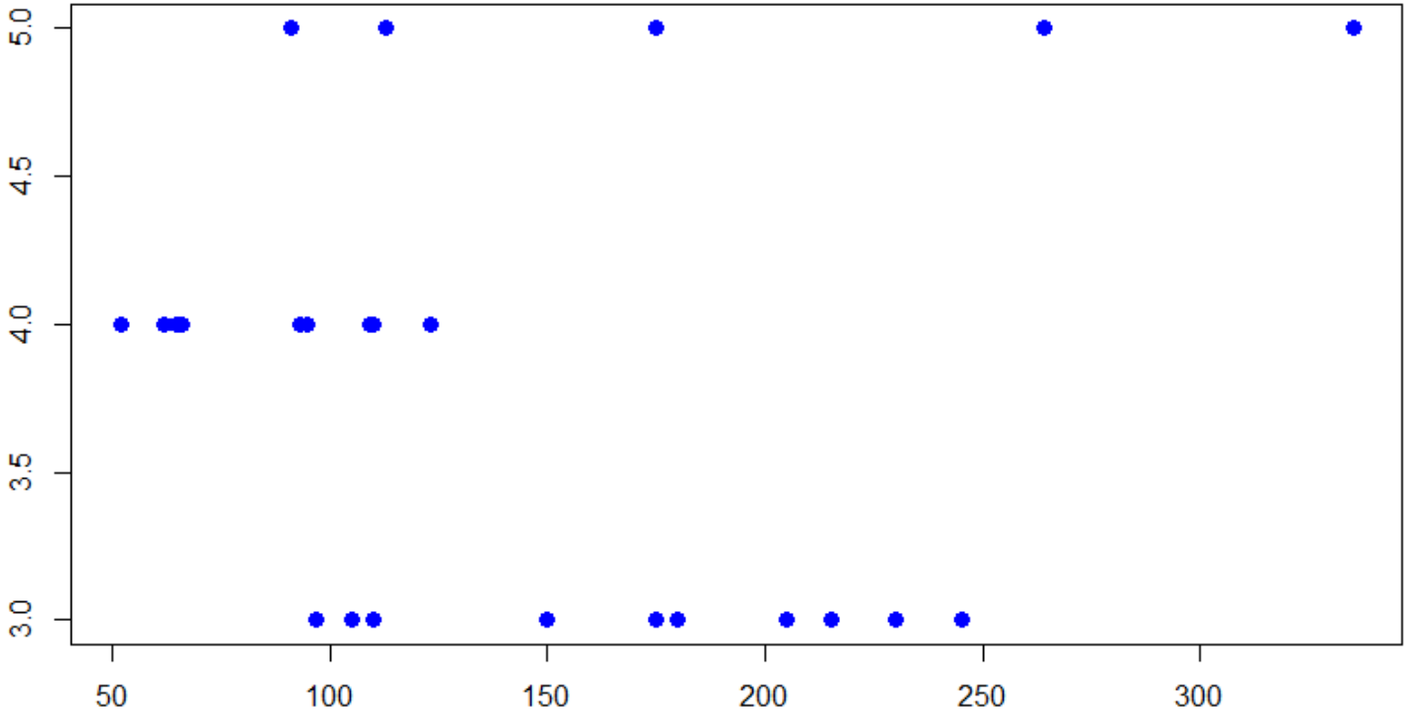
Creating Models with R

Linear Regression

Before the next step, let's analyze another quick example

With all we already see about models and linear regression:

What do you think about the below chart. *Linear regression would be a good choice for it/to describe it? What we need to do so we can predict values from y axis?*



Creating Models with R

Linear Regression

lm(mpg ~ wt)

Dependent scalar variable (label).

Explanatory variables (features). Use plus sign to use more than one feature lm(lbl ~ f1 + f2 + … + fn)

`lm()` is used to the fit linear models. It can be used to carry out regression, single stratum analysis of variance, and analysis of covariance

```
> mt <- mtcars
> m <- lm(mt$mpg ~ mt$wt)
> m

Call:
lm(formula = mt$mpg ~ mt$wt)

Coefficients:
(Intercept)          mt$wt
     37.285         -5.344
```

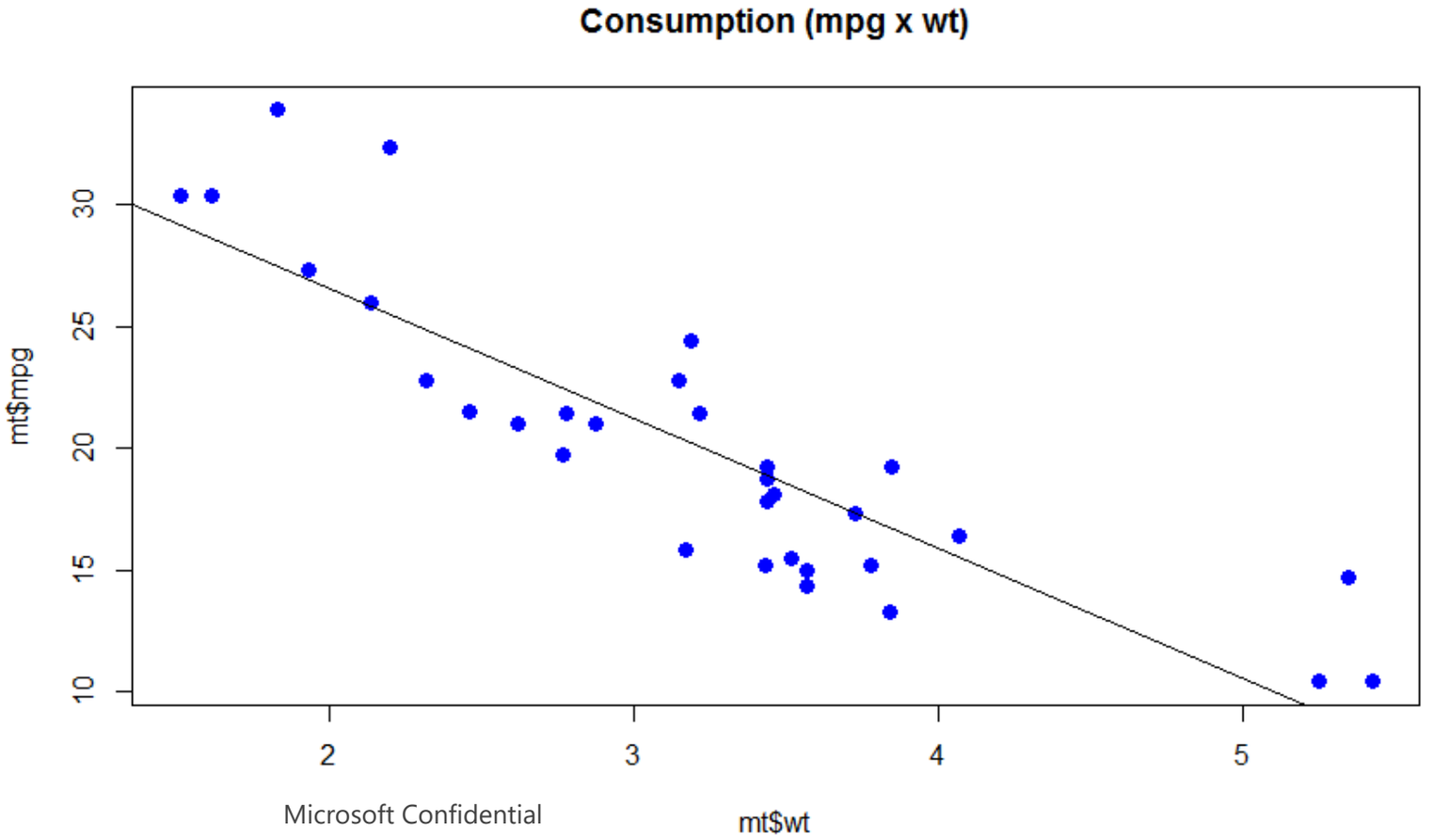
Line coefficients

Creating Models with R

Linear Regression

The calculated line can be plotted by using the `abline()` function

```
> plot(mt$wt, mt$mpg, pch = 16, cex = 1.3, col = "blue", main = "Consumption (mpg x wt)")
> m

Call:
lm(formula = mt$mpg ~ mt$wt)

Coefficients:
(Intercept)          mt$wt
     37.285         -5.344

> abline(m[[1]][1], m[[1]][2])
```



Creating Models with R

Linear Regression

The model is stored in "m". To use it we can use `predict()` function. See below the full experiment

```
> mpg <- mtcars$mpg
> wt <- mtcars$wt
> m <- lm(mpg ~ wt)
> m

Call:
lm(formula = mpg ~ wt)

Coefficients:
(Intercept)           wt
     37.285       -5.344

> sample_data <- mtcars[1:5, ] # get first 5 lines
> data.frame(actual = sample_data$mpg, predict = predict(m, sample_data))
                  actual  predict
Mazda RX4           21.0 23.28261
Mazda RX4 Wag       21.0 21.91977
Datsun 710          22.8 24.88595
Hornet 4 Drive      21.4 20.10265
Hornet Sportabout   18.7 18.90014
```

Demonstration: Creating a Model by Using R

Creating a Machine Learning Model by Using R

Lab: Creating ML Model by Using R

Exercise 01 – Creating Machine Learning Models by Using R