Introduction to R

ALY-6050

Skills

- **Distinguish the differences between R and Excel for data analysis and recognize the advantages of R**
- Navigate the RStudio working environment, using syntax and opening files
- Understand the basic R data types and data objects, syntax of creation and expression and typical applications
- Differentiate between and apply basic data manipulation, such as indexing, subsetting and reshaping data in R

Introduction to R (I)

- R is an open-source statistical programming language
- Supported by CRAN, the Comprehensive R Archive Network
  - Core code is distributed to users for free
  - CRAN maintains the source code and documentation
  - User community can contribute to R by writing packages to increase the functionality of R
- Free integrated development environment (IDE), RStudio

Introduction R (II)

- R is easy to learn
- R is most useful for statistical analysis, data manipulation, data analytics and creating customized, high-end graphics and visualizations
- Many resources from the community:
  - http://www.r-bloggers.com/
  - https://www.r-project.org/

Advantages of R over Excel (I)

- Excel is imited to relatively small data sets, 1,048,576 rows by 16.384 columns, and large .xlsx files can be very slow to navigate and edit
  - R addresses these limitations in two ways:
    - Scripting and setting up plotting and plot formatting can be performed without opening file storing the data, large dataset is not in RAM during scripting
    - Data can be allocated to up to 8TB of RAM, and there are R packages available that allow users to store data in hard drives and analyze it in chunks

Advantages of R over Excel (II)

- Excel has limited functionality for statistical computing, whereas R continues to grow in this regard through package submissions from the user community

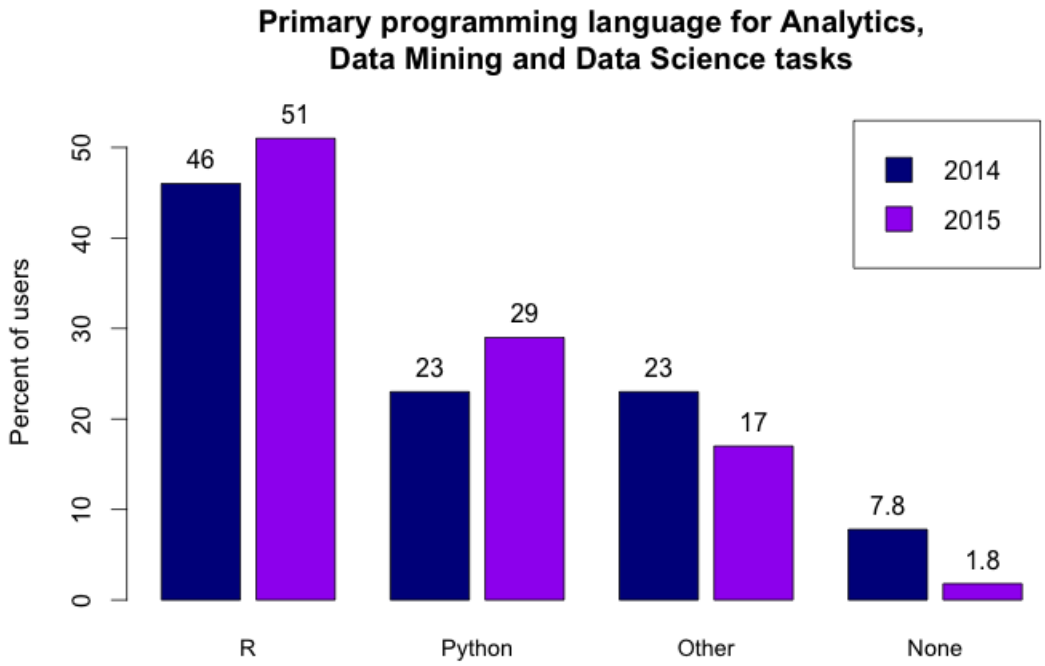
- Difficult to reproduce analysis in Excel:
  - Copying and pasting is prone to human error
  - Raw data is often next to cells requiring data entry or modification, equations are easily, accidentally overwritten
  - Using R scripts, the same analysis can be performed on multiple datasets by changing a single input
  - Similarly, the same graphic can easily be created for more than one data set using R
  - Code can be commented, tested and versioned

Correct tool for the task

- Excel
  - Simple descriptive or inferential statistics on a relatively small dataset
  - Collaborating on a project with people who are not data analysts (and don't know R), so long as the formulas and functions are simple
- R
  - Large datasets
  - Advanced statistical computations
  - Same analysis on more than one data set
  - Complex data visualization
  - Analyses which must be reproducible, require advanced functionality or plotting capabilities

R vs. other languages (II)

- R has similar advantages over Excel to other languages
- R is popular, growing and relatively easy to learn
- It is also free and has a strong support community



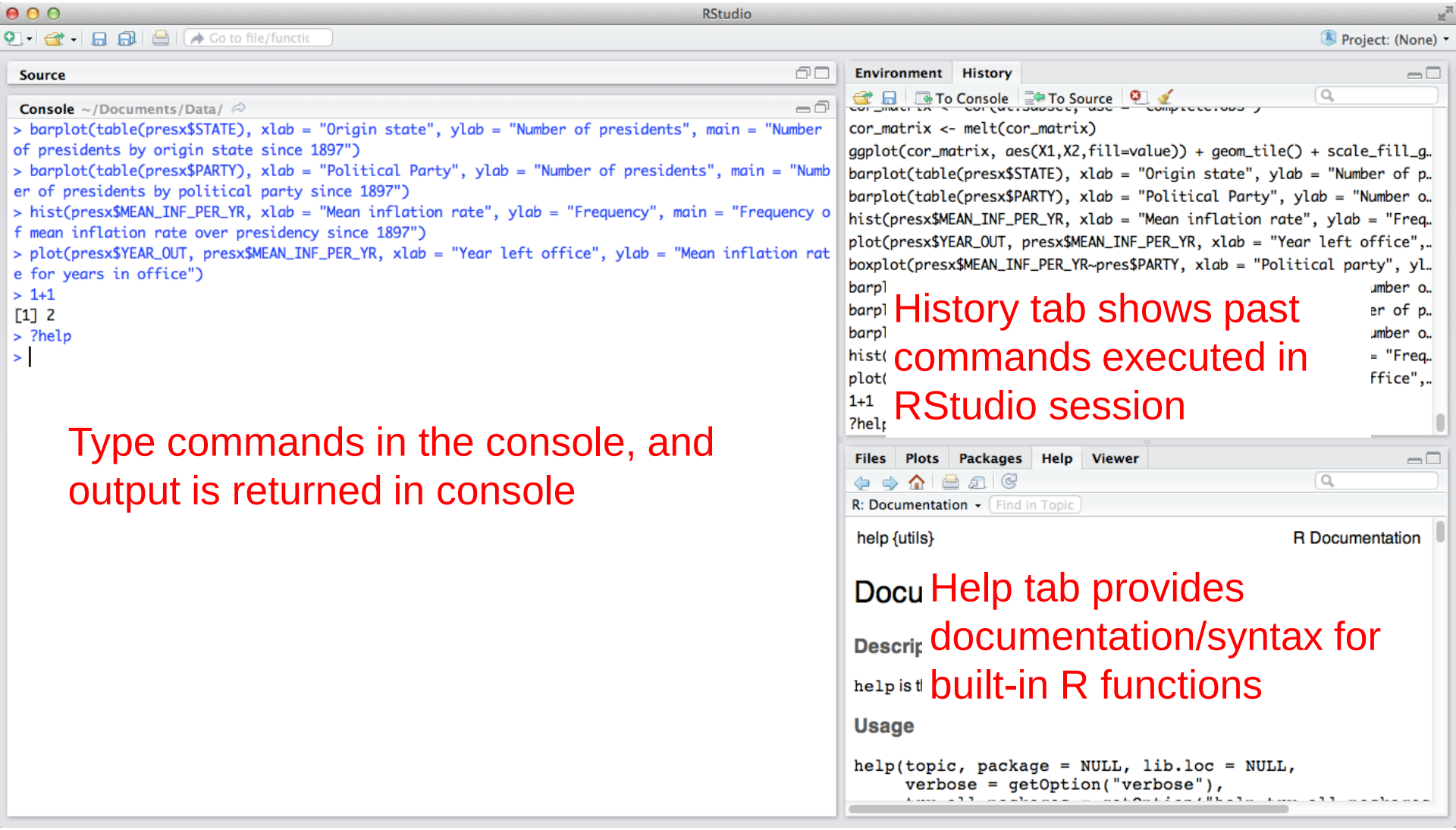
- Note: "Other" includes MATLAB, Java, SAS, Scala, etc.

http://www.kdnuggets.com/polls/2015/r-vs-python.html

Skills

- Distinguish the differences between R and Excel for data analysis and recognize the advantages of R
- **Navigate the RStudio working environment, using syntax and opening files**
- Understand the basic R data types and data objects, syntax of creation and expression and typical applications
- Differentiate between and apply basic data manipulation, such as indexing, subsetting and reshaping data in R

RStudio GUI (I)



Type commands in the console, and output is returned in console

RStudio GUI (II)

RStudio

testcode.R | dt | dt.store19

```
172 plot(dt$FORMAT,dt$SALES_TISP,col="red",ylab="TISP ($)",main="TISP boxplots by store format"
173
174 # Average sales and margin comparion across store formats
175 by(dt[,7:8], dt$FORMAT, colMeans)
176
177 # Question: Can you plot the average total store space and total TISP sales comparison?
178 # by(dtpresx <- rbind(pres2,pres)
179 [,14:15], dt$FORMAT, colMeans)
180
181 # Correlation between total sales and total space
182 cor(dt$TOTAL_TISP,dt$TOTAL_SPACE,method="kendall",use="complete.obs")
183
184 # or, use the cor.test to obtain more details such as significance
185 cor.test(dt$TOTAL_TISP,dt$TOTAL_SPACE,method="kendall",use="complete.obs")
186
190 # Advanced correlation matrix
191 dt.subset <- subset(dt,select=c(SALES_UNITS,SALES_TISP,TXNS,NDSA))
192 cor_matrix <- cor(dt.subset, use = "complete.obs")
193
```

179:1 (Untitled) R Script

Displays open scripts, data and variables

Environment | History

Import Dataset | List

Global Environment

Data

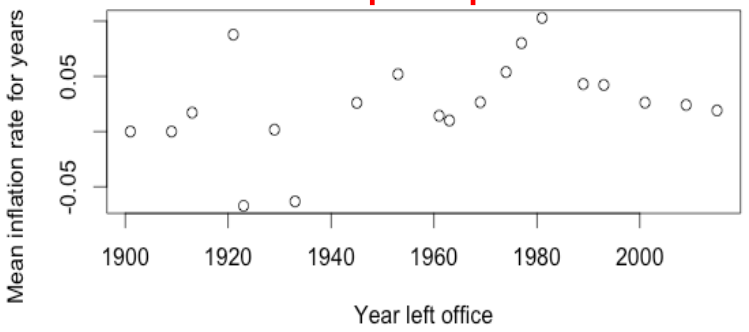
dt	1200 obs. of 16 variables
dt.store19	12 obs. of 16 variables
pres	18 obs. of 6 variables
pres2	2 obs. of 6 variables
pres3	22 obs. of 6 variables
presx	20 obs. of 6 variables

Environment tab shows active data, variables and functions in session

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish

Plot tab shows plot preview



Console ~/Documents/Data/

```
> hist(presx$MEAN_INF_PER_YR, xlab = "Mean inflation rate", ylab = "Frequency", main = "Frequency
of mean inflation rate over presidency since 1897")
> plot(presx$YEAR_OUT, presx$MEAN_INF_PER_YR, xlab = "Year left office", ylab = "Mean inflation r
ate for years in office")
> barplot(table(presx$STATE), xlab = "Origin state", ylab = "Number of presidents", main = "Numbe
r of presidents by origin state since 1897")
> plot(presx$YEAR_OUT, presx$MEAN_INF_PER_YR, xlab = "Year left office", ylab = "Mean inflation r
ate for years in office")
>
```

RStudio GUI (III)

- In this presentation, as well as in RStudio, we will use the following font and color scheme:

```
Input to command line
Printed in console
Error or warning
Commented code
```

- Check out and follow the google style guide for R:
  - Makes code easier to read/share with others
  - https://google.github.io/styleguide/Rguide.xml

Basic syntax guidelines for R

- `#` begins a line of comment
- R is case sensitive: `x` is not the same as `X`
- arguments go between parenthesis (and are separated by commas), for example `help(lapply)`
- multiple arguments are separated with commas
- `<-` is the assignment operator, analogous to equals sign

Useful functions

- `print()` - displays argument value(s) in console
- `install.packages()` – installs (attaches) add-on packages to the basic installation of R, note: packages remain installed until uninstalled
- `installed.packages()` – with empty parentheses, lists packages already installed
- `library()` – opens library of functions within a package, note: empty parenthesis lists libraries/packages available for opening in an R session in upper-left window; note: libraries are closed when R session is closed
- `search()` – with no arguments, lists open packages in console

Getting help in R

- `help()` or `?` – opens help tab for argument, shows syntax options
- `example()` – runs all the R code from the Examples section of the help documentation for argument
- `browseVignettes()` – lists available longer form documentation for argument in a web browser
- `RSiteSearch()` - searches for key words or phrases in help pages, vignettes or task views, using http://search.r-project.org and views them in a web browser
- `apropos()` – returns the names of all objects matching argument

Working directory (I)

- Working directory is the folder currently in use in R
- Location from which files are opened and location to which files are saved
- Important to know and be able to change so that files are read and written to known location

Working directory (II)

- `getwd()` – get working directory, returns current working directory in console
- `setwd()` – set working directory, assigns working directory
  - `setwd(".. ")` – up one folder
  - `setwd("../..")` – up two folders
  - `setwd("Folder1/Folder2/Folder3")` – specifies path within current working directory

- Create a folder on your computer for R projects

Check your knowledge (VIII)

- Create a folder on your computer for R projects, and set it to your working directory, example solution with extra navigation:

```
getwd()
[1] "/Users/YourName/Documents/Data"
setwd("..")
getwd()
[1] "/Users/YourName/Documents"
setwd("../..")
getwd()
[1] "/Users"
setwd("YourName/Documents/RData")
getwd()
[1] "/Users/YourName/Documents/RData"
```

Reference - functions to read and write files in R (I)

- `read.table()` – default delimiter is white space
- `read.csv()` – default delimiter is a comma
- `read.csv2()` – default delimiter is a semicolon, and commas are read as decimal points
- `read.delim()` – default delimiter is a tab
- `read.delim2()` – default delimiter is a tab, and commas are read as decimal points

Reference - functions to read and write files in R (I)

- `read.table(file.choose())` – opens window to browse for file

- See help for: `write.table()`, `write.csv()` and `write.csv2()` for information about saving data in R to file

Reading Excel files in R with XLConnect, xlsx and gdata (I)

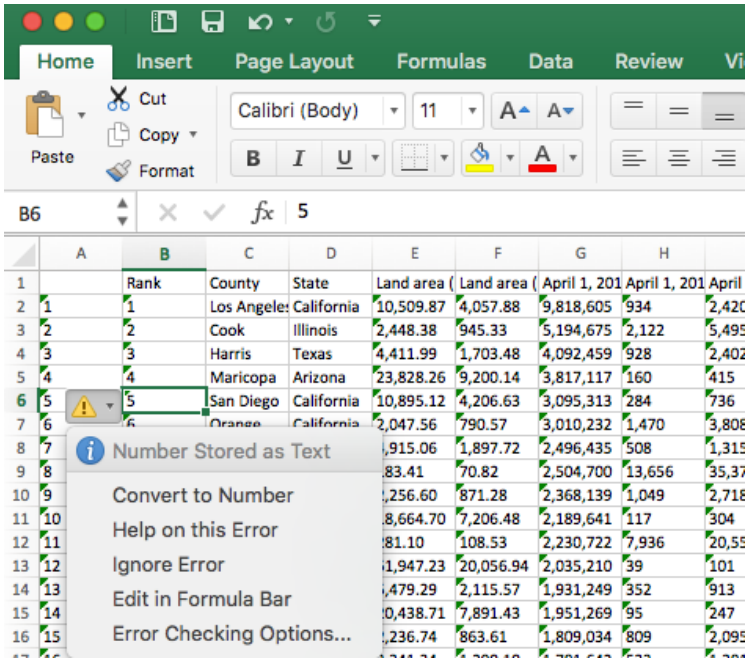
- Many ways to read worksheets and workbooks from Excel into R:
  - `readWorksheetFromFile()` – reads data from worksheets in Excel into data frame (`XLConnect` package)
  - `loadWorkbook()` – reads in entire Excel workbook (`XLConnect` package)
  - `read.xlsx()` – reads data from worksheets in Excel into data frame (`xlsx` package)

Reading Excel files in R with XLConnect, xlsx and gdata (II)

- Many ways to read worksheets and workbooks from Excel into R:
  - `read.xlsx2()` – similar to `read.xlsx()` but faster for large datasets (`xlsx` package)
  - `loadWorkbook()` – reads in entire Excel workbook (`xlsx` package)
  - can write to Excel files with `write.xlsx()` and `write.xlsx2()` (`xlsx` package)
  - `read.xls()` – read Excel file into a data frame using Perl (`gdata` package)

A note about data types in R

- We must be conscious of and intentional about data types in R
  - Soon we will learn how to specify, check and change data types in R



Skills

- Distinguish the differences between R and Excel for data analysis and recognize the advantages of R
- Navigate the RStudio working environment, using syntax and opening files
- **Understand the basic R data types and data objects, syntax of creation and expression and typical applications**
- Differentiate between and apply basic data manipulation, such as indexing, subsetting and reshaping data in R

Data types in R

Atomic Type	Example	typeof()	mode()	class()
character	"hello"	character	character	character
real	1.34	double	numeric	numeric
integer	3L	integer	numeric	integer
complex	3.0 – 4.3i	complex	complex	complex
logical	TRUE or T	logical	logical	logical

Check your knowledge (I)

Match the output of `class(x)` with the corresponding command

1. `x <- "50L"`
2. `x <- 50L`
3. `x <- 50`
4. `x <- 50i`
5. `x <- TRUE`

A. `integer`
B. `numeric`
C. `logical`
D. `character`
E. `complex`

Data objects (I)

- ***Data objects*** store groups of elements in R, and this classification defines what type of operations can be performed on these elements

- ***Vectors*** – store elements of the same ***data type*** in a single column or row
- ***Matrices*** – store elements of the same ***data type*** in multiple columns or rows
- ***Lists*** – store different types of ***data objects*** in the form of a ***vector*** (single column)
- ***Factors*** – store ***integer*** elements to represent discrete categories, known in R as levels
- ***Data frames*** – similar to a single sheet in Excel
  - Store data in a tabular format
  - Each column can be a different ***data type***

Check your knowledge (VIII)

Match the data types to the output, using each exactly once

1. Vector
2. Matrix
3. List
4. Factor
5. Data frame

A.
```
[1] hello   hello   welcome
Levels: hello welcome
```

B.
```
[1] "hello"   "welcome" "goodbye"
```

C.
```
  c..hello....welcome.. c.1..2.
1                 hello       1
2               welcome       2
```

D.
```
[[1]]
[1] "hello" "hello"

[[2]]
[1] hello   welcome
Levels: hello welcome
```

E.
```
     [,1]      [,2]
[1,] "hello"   "goodbye"
[2,] "welcome" "hello"
```

Basic arithmetic in R

- R can perform all of the basic arithmetic of a scientific calculator, order of operations (PEMDAS) applies
- In addition to using R for statistics, may need to add, subtract or multiple columns, much like in Excel

`rowMeans()` and `colMeans()` – computes the arithmetic mean for rows or columns of a matrix or data frame (2D data object)

`rowSums()` and `colSums()` – computes the numeric sum of rows or columns of a matrix or data frame (2D data object)

`sum()` and `prod()` – returns the sum of all values in arguments and the product of all values in arguments, respectively

Skills

- Distinguish the differences between R and Excel for data analysis and recognize the advantages of R
- Navigate the RStudio working environment, using syntax and opening files
- Understand the basic R data types and data objects, syntax of creation and expression and typical applications
- **Differentiate between and apply basic data manipulation, such as indexing, subsetting and reshaping data in R**

Subsetting data in R (I)

- Elements within ***data objects*** in R are indexed in the format [`row, column`]

- A particular element or set of elements can be returned from a ***data object*** using the name of the ***data object***, followed by the square bracket with the desired indices

- Consider a 5x4 ***data frame***, `df`:
  - `df[1,]` returns first row of `df`
  - `df[,2]` returns second column of `df`
  - `df[1,2]` returns single element in the first row and second column of `df`

Subsetting data in R (II)

- Because the indices of ***data objects*** are ***integers***, a colon, `:`, is used to specify more than one row
  - Note: the colon returns all ***integers*** between the ***integers*** on either side of the colon, inclusive

- Consider a 5x4 ***data frame***, `df`:
  - `df[3:5,]` returns third through fifth rows of `df`
  - `df[,1:3]` returns first through third columns of `df`
  - `df[3:5,1:3]` returns nine elements of `df` spanning the third through fifth rows and the first through third columns

Subsetting data in R (III)

- Elements within ***data objects*** in R are indexed in the format [`row, column`]

- Elements with only one dimension, such as ***vectors*** can be subsetted without the colon

- Consider ***vector*** of length 6, `a`:
  - `a[1]` returns first element of `a`
  - `a[3:5]` returns third through fifth elements of `a`

Subsetting data in R (IV)

- Elements within ***data objects*** in R are indexed in the format [`row, column`]

- Exception for ***lists***: list elements are subsetted with two sets of square brackets, followed by the subsetting rules for other ***vectors***, ***matrices***, or ***data frames***

- Consider ***list*** of 6 3x3 ***matrices***, `b`:
  - `b[[3]][2,]` returns second row from the third 3x3 matrix in `b`

Subsetting data in R (V)

- **$** is important for accessing data in ***data frames***
  - ***data frames*** have row and column names, they are automatically generated and can be reassigned
  - Reassign row and column names to something useful
  - Access rows and columns using the name of the ***data frame***, followed by the dollar sign, **$**, followed by the name of the row or column if interest

Check your knowledge (VIII)

Matching, **x** is a data frame of dimensions 5x5

1. Element in Row 2, Column 2 of **x** **D**
2. Row 2 of **x** **F**
3. Column 2 of **x** **B**
4. Rows 2, 3 and 4 of **x** **A**
5. Columns 2, 3 and 4 of **x** **E**
6. Columns 2 and 4 of **x**
7. Rows 2 and 4 of **x**

A. `x[2:4,]`

B. `x[,2]`

C. `x[seq(2,4,2),]`

D. `x[2,2]`

E. `x[,c(2,3,4)]`

F. `x[2,]`

G. `x[,c(2,4)]`

Useful querying functions in R (I)

`attributes()` – returns information about shape, size and type of data object in argument

`dim()` – returns dimensions of argument

`length()` – returns length of argument, number of columns for a matrix or data frame

`str()` – returns structure of argument

`head()` – returns first rows of argument

Useful querying functions in R (II)

`tail()` – returns final rows of argument

`colnames()`/`names()` – returns or assigns column names of argument

`rownames()` – returns or assigns row names of argument

`unique()` – returns unique elements within argument, can be applied to factors as well as character vectors

Useful querying functions in R (III)

`duplicated()` – returns logical vector indicating which elements are duplicated in a given vector or data frame

`summary()` – returns summary descriptive statistics of argument for numerical data, and tabulates number of elements in each factor level of argument for categorical data

Appending, modifying and removing data in R (I)

`subset()` – returns a subset of data from argument based on a selection argument

`merge()` – returns a merged data frame from two input data frame arguments, merged by common columns or row names

`cbind()/rbind()` – used to combine a sequence of ***vectors***, ***matrices*** or ***data frames***, into a single large ***data object***

- `cbind()` – combines by columns
- `rbind()` – combines by rows

Which functions for subsetting data in R (I)

`which` functions return indices of elements that match some argument

`which()` - returns indices of the elements elements that match the argument

`which.min()` – returns indices of element that has the minimum value in the argument

`which.max()` – returns indices of element that has the maximum value in the argument

Sorting data in R (I)

- Useful to sort a data set by some variable
  - Combine with `head()` or `tail()` to see the smallest or largest values

`sort()` – returns sorted argument, use for datasets with a single column

`order()` – returns sorted argument, use for datasets with multiple columns

Some of the many statistics functions in R (I)

- There are many descriptive and inferential statistics functions in R
  - Below are some simple functions which can be used in combination with more advanced functions to calculate statistics for numeric data
  - Can operate every column, row or groups of the data split by a categorical variable

`mean()` – returns mean of argument

`median()` – returns median of argument

Some of the many statistics functions in R (II)

- There are many descriptive and inferential statistics functions in R

`var()` – returns sample variance of argument

`cov()` – returns covariance of argument(s)

`cor()` – returns correlation coefficient of argument(s)

Apply functions in R (I)

- `apply()` functions, which are built-in to R, apply a formula or built-in R function to multiple elements within a data object
  - Select between `apply` functions based on type of data objects we have for our dataset and type of data object we want as the output

`apply()` – applies formula or function to a vector or matrix argument, across some dimension (rows or columns) and returns output in a matrix or vector

`lapply()` – applies formula or function to a list, vector or matrix argument and returns output in a list

Apply functions in R (II)

`sapply()` – applies formula or function to a list, vector or matrix argument and returns output in a vector or matrix

`tapply()` – applies formula or function to categorical subsets of data in a vector or matrix argument and returns output in a list

Built-in functions for processing data in R (I)

`with()` – applies function to selected data in data frame, does not affect data frame

`within()` – applies function to selected data in data frame, appends results to data frame

`by()` – applies function to subsets of data specified by factor or list of factors

`aggregate()` – splits data into subsets by factor, returns descriptive statistics

NA and NaN (I)

- Nonsensical ***coercion*** brings us special and missing values in R

- **NA** is a missing value
  - NA has a ***data type***, such as ***integer*** or ***logical***, but no specific value

- **NaN** is defined as zero divided by zero, stands for "not a number"

- **NaN** is **NA** , but **NA** is not **NaN**

NA and NaN (II)

`is.na()` - returns a logical element; TRUE if argument is `NA`, FALSE if argument is not `NA`

`is.nan()` - returns a logical element; TRUE if argument is `NaN`, FALSE if argument is not `NaN`

NA and NaN (III)

`complete.cases()` – returns logical element for each row in matrix/data frame or each element in vector, TRUE if no `NA` or `NaN` values are present, FALSE if at least one `NA` or `NaN` is present

- Useful for calculating statistics for large samples with missing values

- Also check out `anyNA()`

- Note: many built-in R functions will return `NA` if any of the elements are `NA`, such as `mean()`
  - Some of these functions accept additional arguments, such as `na.rm = T,` to remove `NA` values in calculations