Introduction to Data Science With R

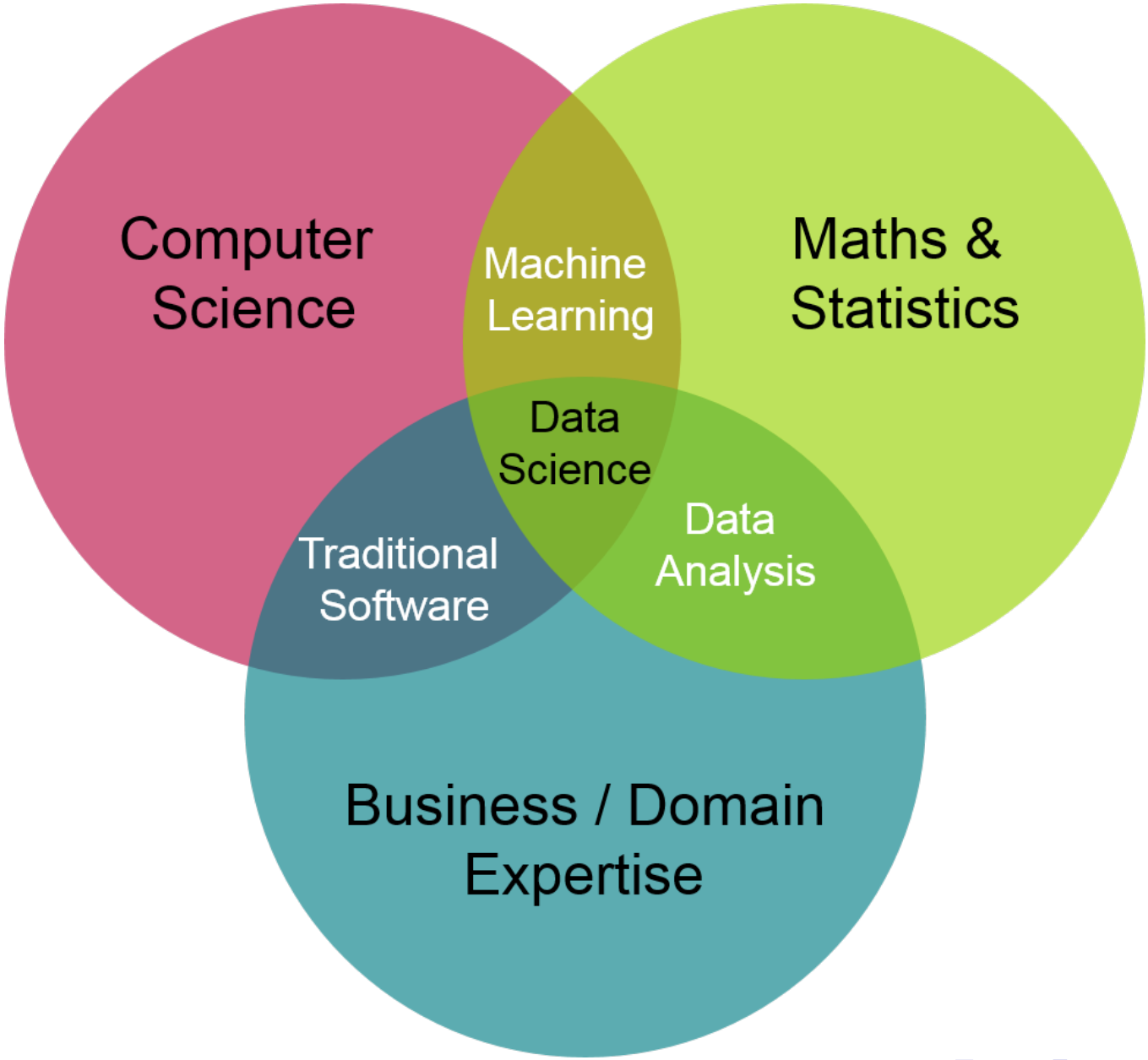
Artür Manukyan

November 26, 2020

Definition of Data Science

- **Data Science** is the scientific approach to the **Data**, covering the entire process of starting from **Data Collection** to **Decision Making**.
- Requires expertise in almost all quantitative disciplines: **Statistics**, **Data Analysis** and **Machine Learning**.
- Especially treding in **marketing research** and **health care**.

Data Science

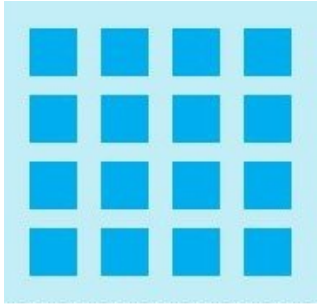


Data Science

Mathematics	Linear Algebra
	Optimization
	Numerical Analysis
Statistics	Estimation
	Hypothesis Testing
	Data Analysis
Computer Science	Machine Learning
	Data Structures
	Visualization
Technology	Storage Platforms
	Computing Platforms
	Statistical Computing Tools

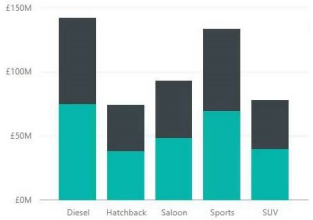
Then and Now

THEN



Structured Data

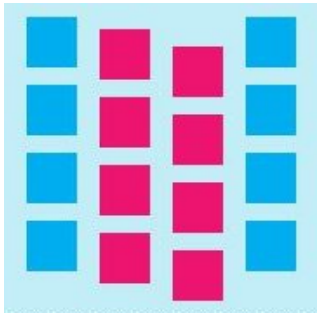
Data Warehouses



Business Intelligence

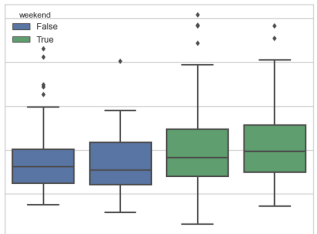
Data Analytics Softwares

NOW



Unstructured Data

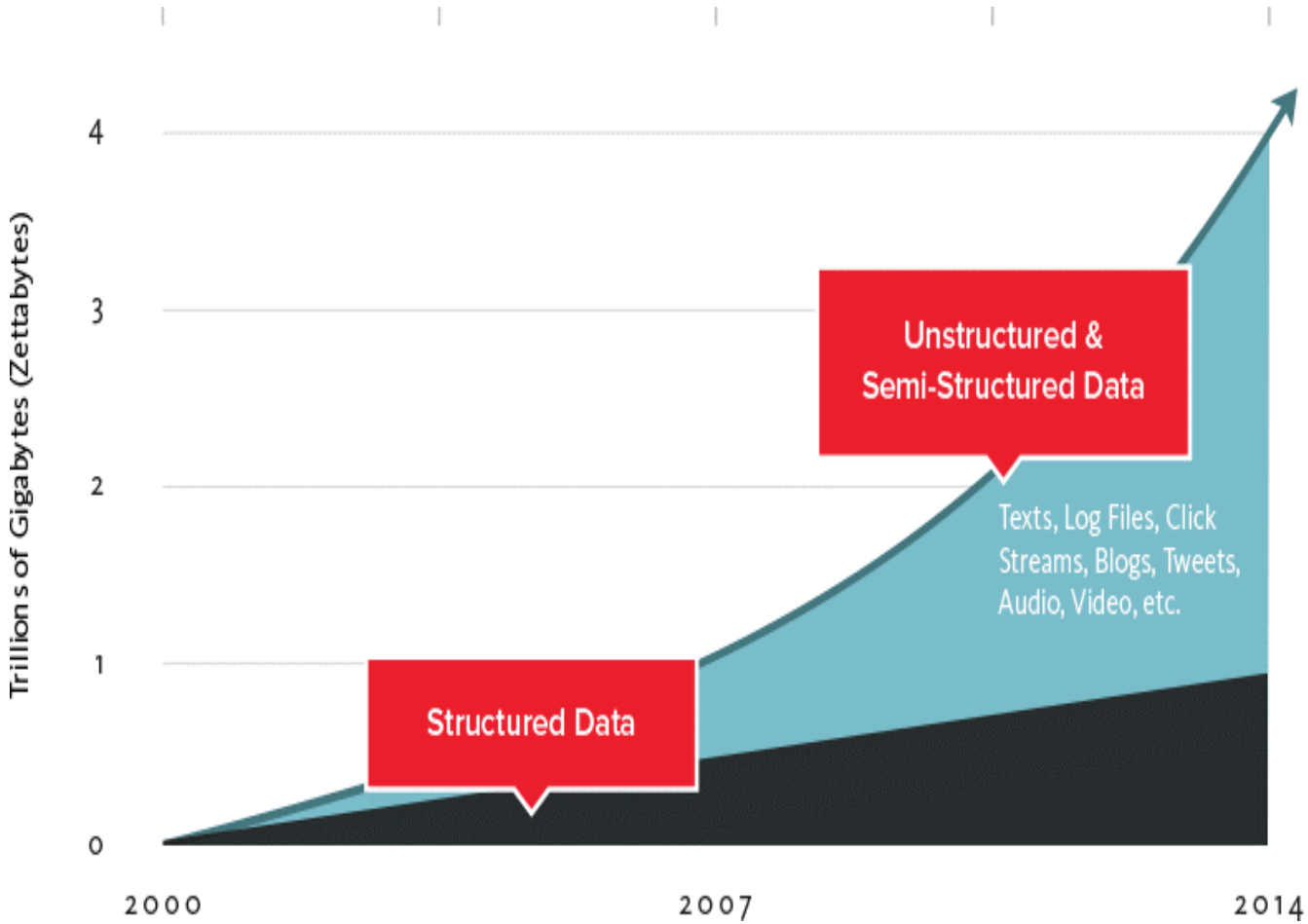
Cloud Computing



Exploratory Data Analysis

Scripting Languages

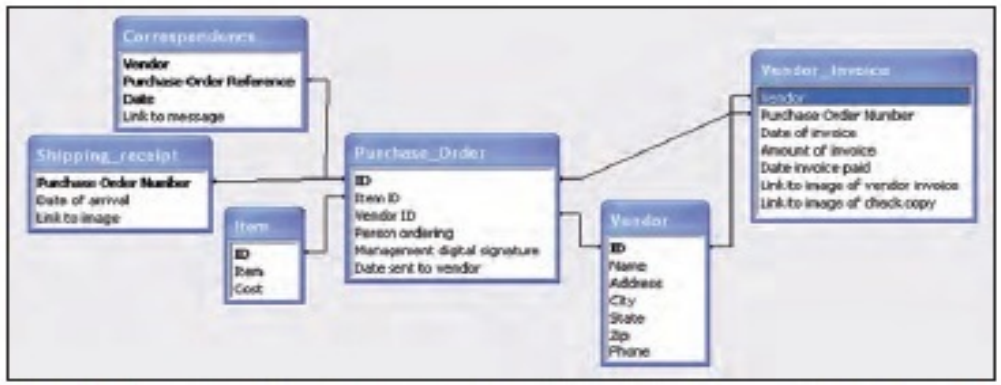
Structured Data vs Unstructured Data



Structured Data

- Usually consists of tables, either connected or single, having data fields (variables or features) for each object (patient, household, student, etc.)

OID	PROCESSINSTANCE	PARENT	ENTRYKEY	XPATH	TYPE_KEY	STRING_VALUE	NUMBER_VALUE
62	1	-1	0	8	-1	-	0
63	1	62	0	2	-1	-	0
64	1	63	0	10	-1	-	0
65	1	64	0	1	-1	-	0
66	1	65	0	4	11	Baker NORT ...	0
67	1	64	1	1	-1	-	0
68	1	67	0	4	8	North CHANGEI	0
69	1	64	2	1	-1	-	0
70	1	69	0	4	11	MY NEW STF ...	0
71	1	64	3	9	8	Smith	0
72	1	63	1	7	8	N1	0
73	1	63	2	7	8	M1	0
74	1	63	3	3	4	-	100
75	1	62	1	2	-1	-	0
76	1	75	0	10	-1	-	0



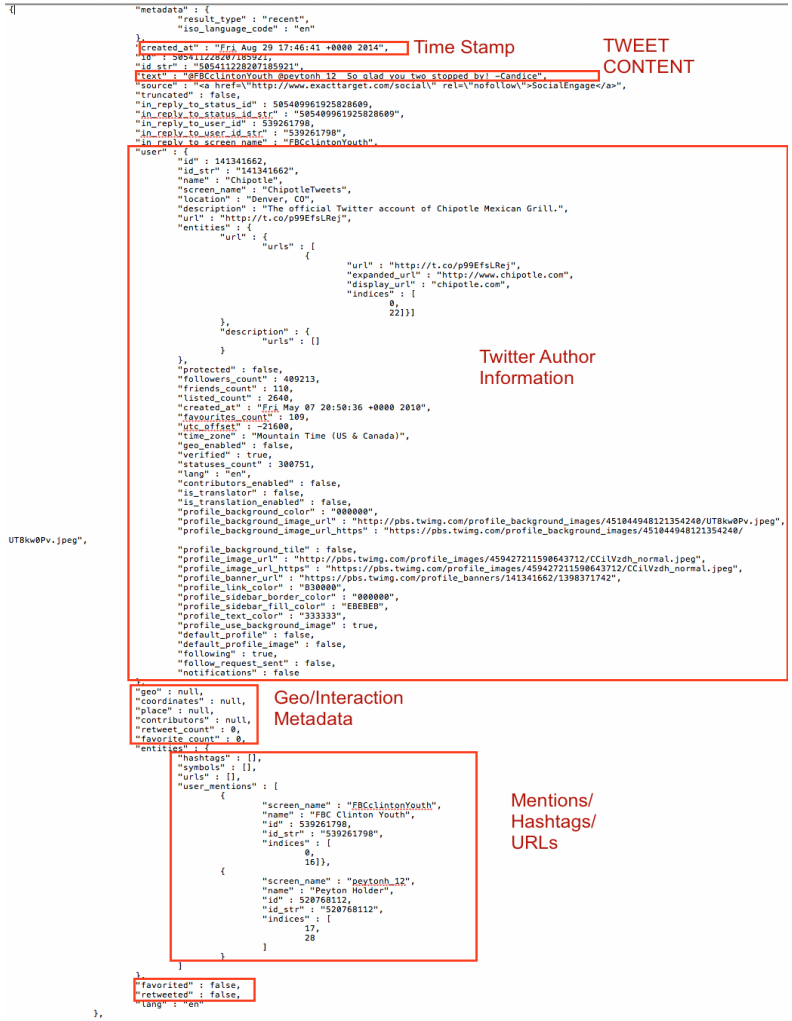
Unstructured Data

- Satellite images
- Photographs and videos
- Social media data (Youtube, Facebook, Twitter, etc.)
- Mobile data

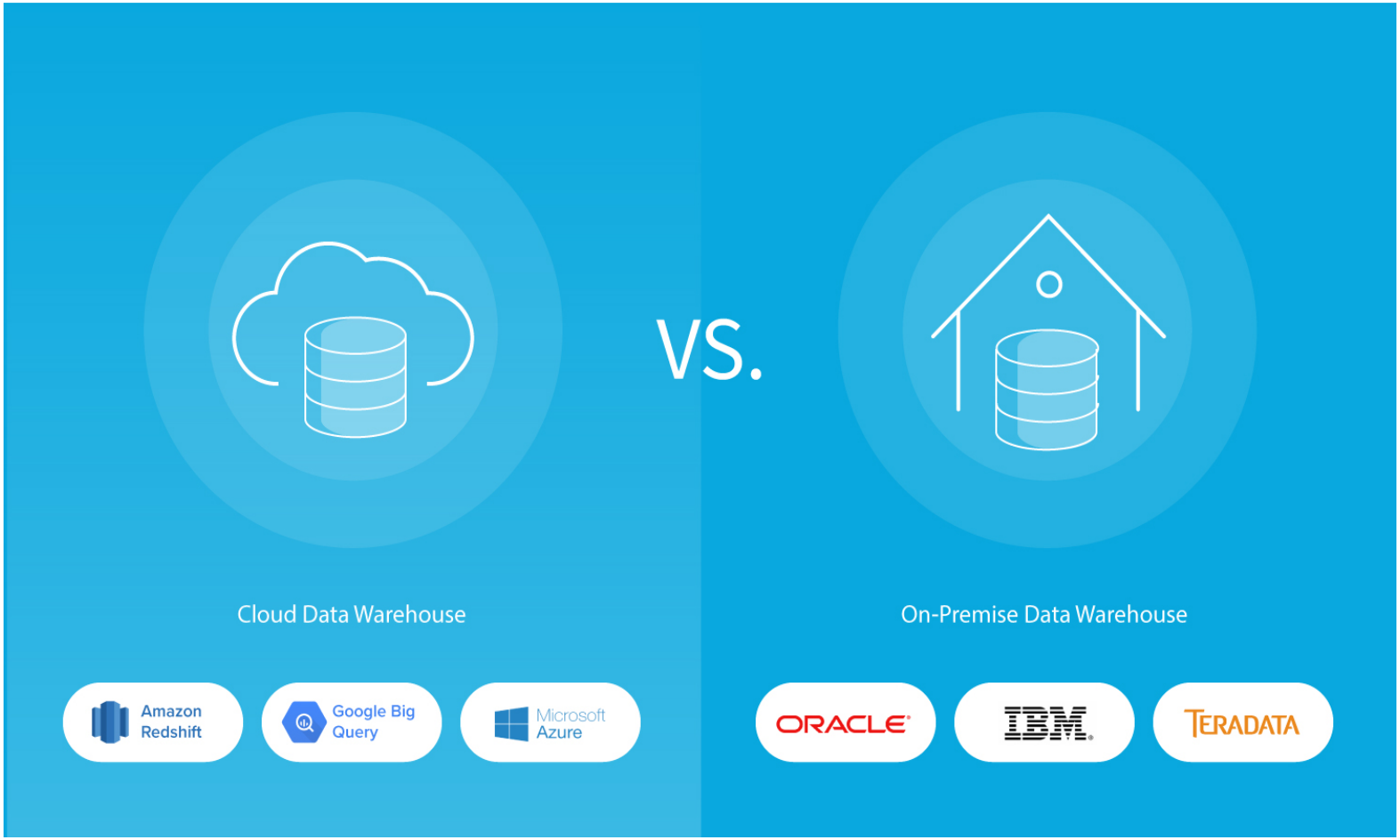
Unstructured Data Example: Twitter

Content of the tweet

- Specific tweet recipient
- Sender of the tweet
- Language of tweet
- Where the tweet originated
- Link to a picture of user
- Date and time of tweet
- Device/platform



Cloud vs Traditional Data Warehouses

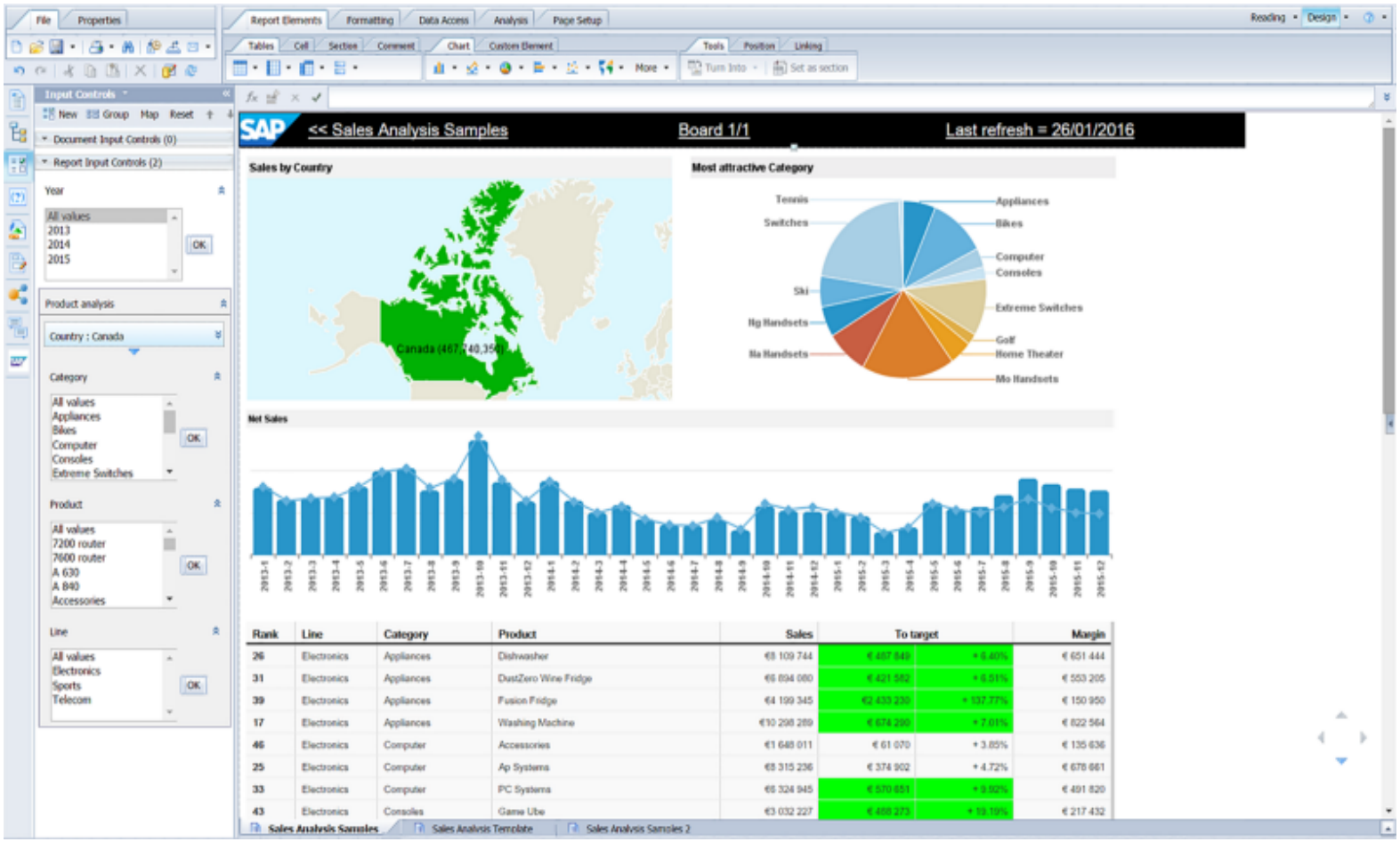


Cloud vs Traditional Data Warehouses

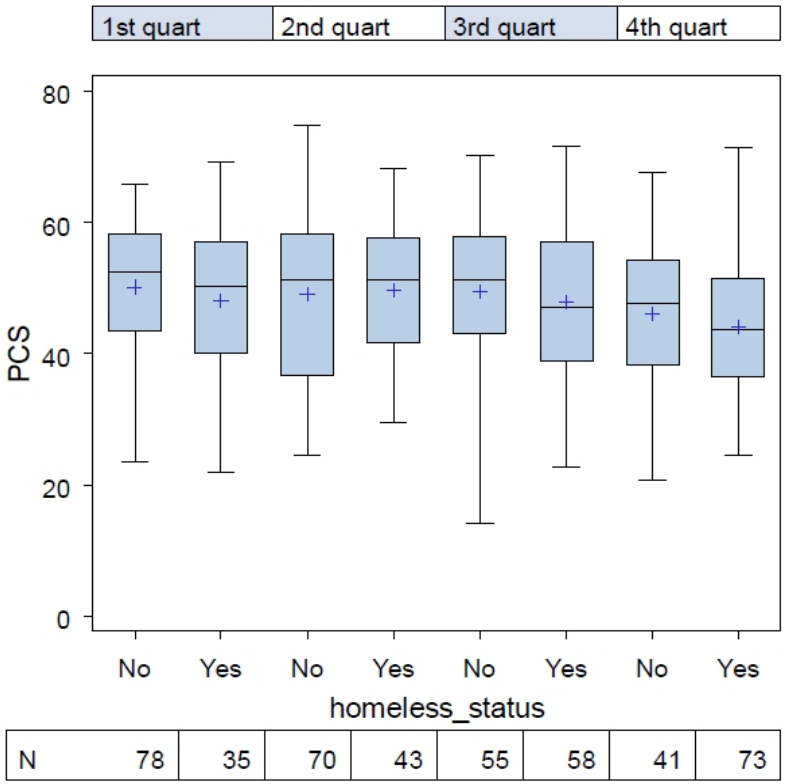
- High cost and infrastracture demand
- More Secure (better for banks and governmental institutions)
- More reliable
- Faster (but not that much!)
- Less scalable

- Low cost and infrastracture demand
- Less secure (acceptable for marketing and health care)
- Considerably reliable (with Amazon and Google)
- Considerably fast
- More scalable (better for storing massive data via using multiple sources)

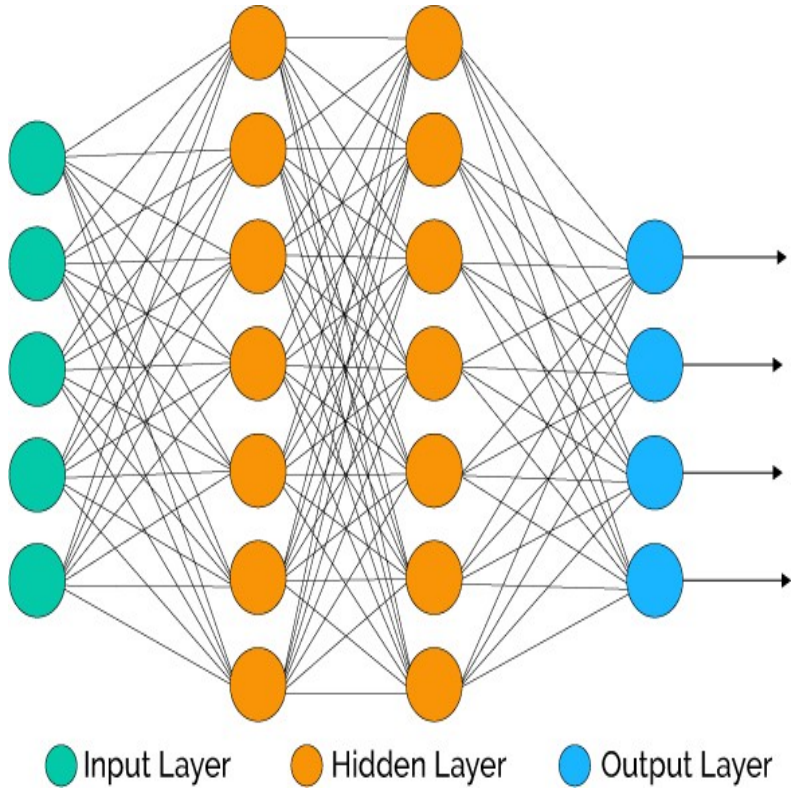
Business Intelligence and Analytics



Data Analysis and Machine Learning

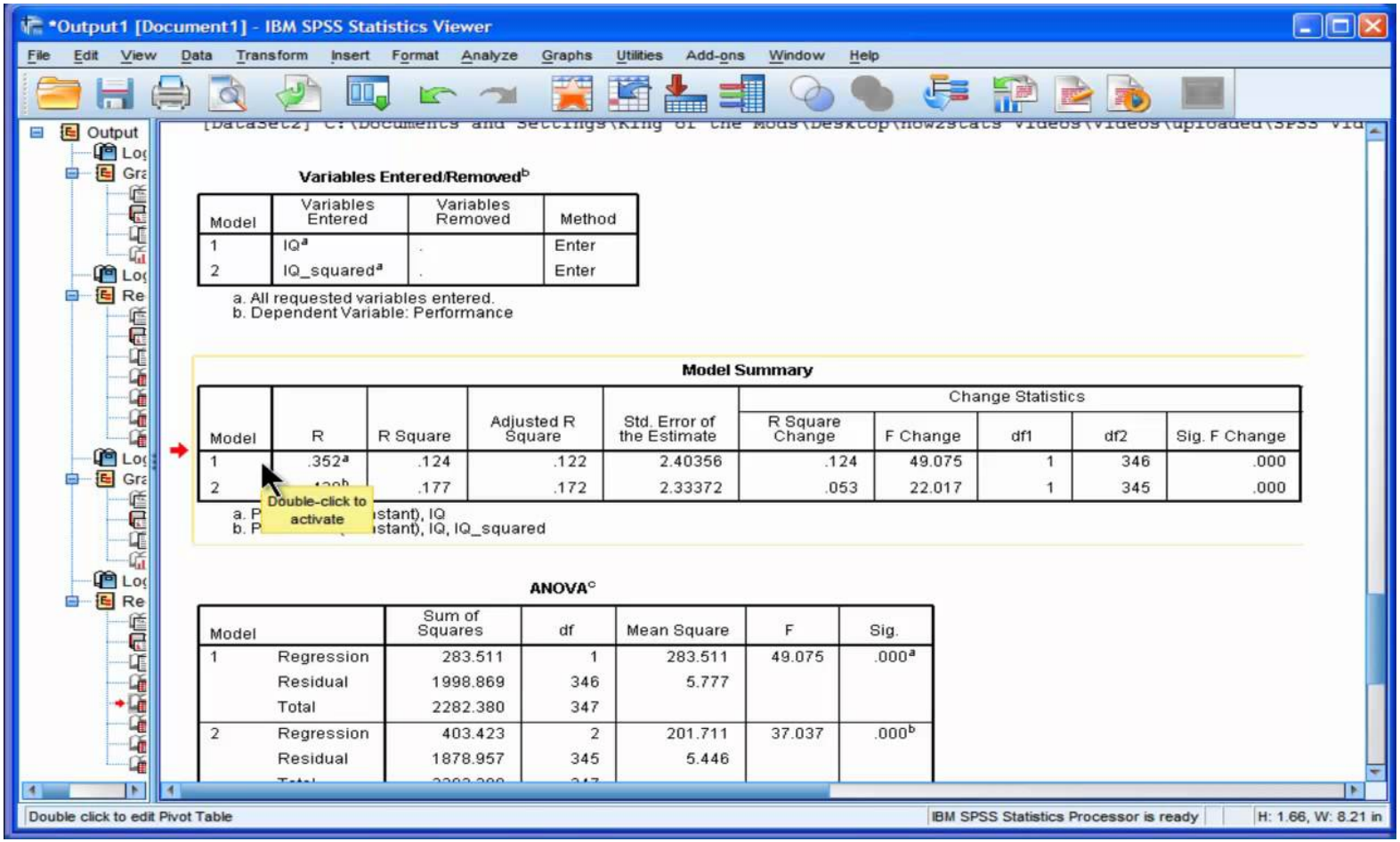


Multiple boxplots
of two layered classes

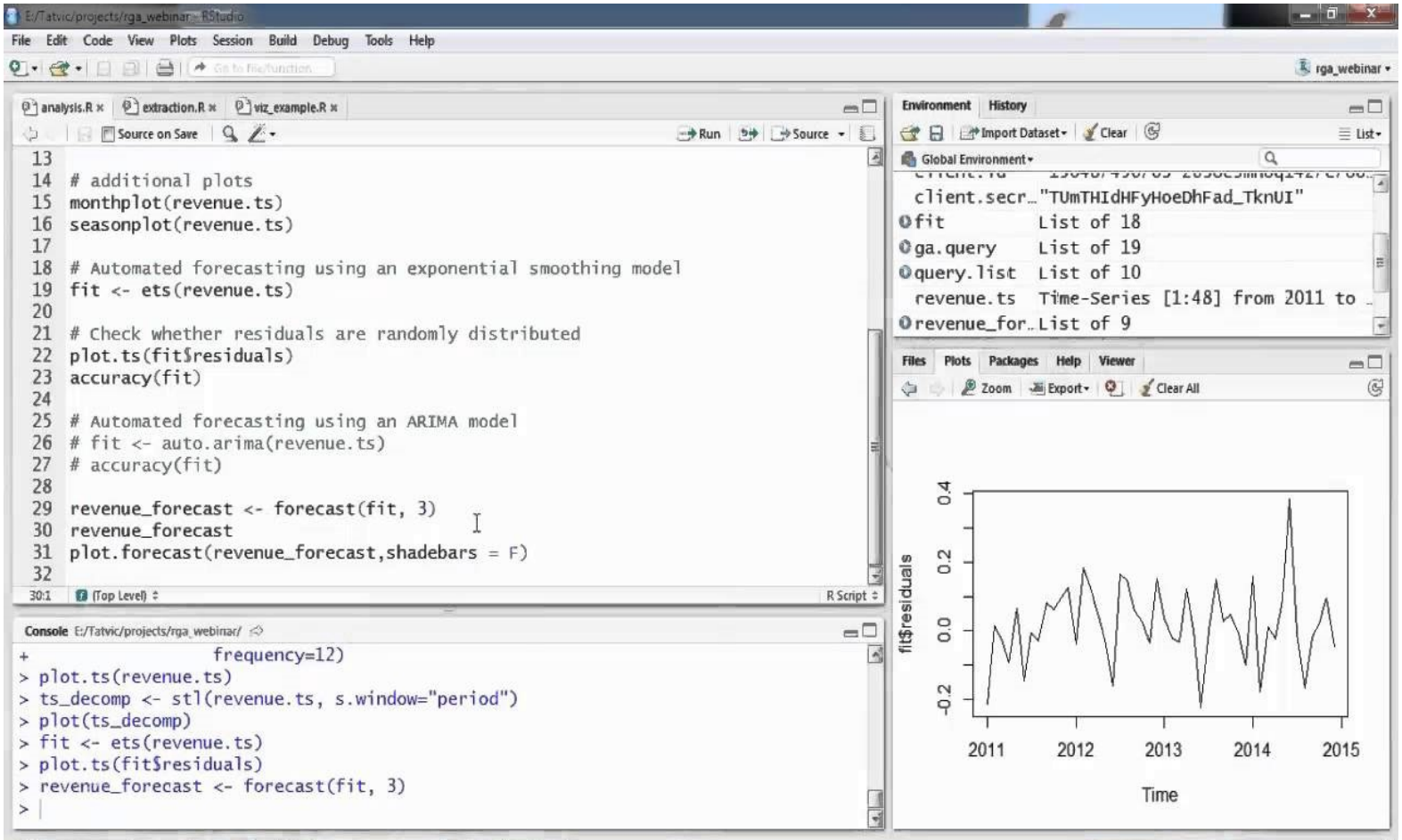


Neural networks
of multiple layers

SPSS


*Output1 [Document1] - IBM SPSS Statistics Viewer
File Edit View Data Transform Insert Format Analyze Graphs Utilities Add-ons Window Help
Output
Variables Entered/Removed[b]
Model | Variables Entered | Variables Removed | Method
1 | IQ[a] | . | Enter
2 | IQ_squared[a] | . | Enter
a. All requested variables entered.
b. Dependent Variable: Performance
Model Summary
Model | R | R Square | Adjusted R Square | Std. Error of the Estimate | Change Statistics | | | | |
Model | | | | | R Square Change | F Change | df1 | df2 | Sig. F Change
1 | .352[a] | .124 | .122 | 2.40356 | .124 | 49.075 | 1 | 346 | .000
2 | | .177 | .172 | 2.33372 | .053 | 22.017 | 1 | 345 | .000
Double-click to activate
b. P stant), IQ, IQ_squared
ANOVA[c]
Model | | Sum of Squares | df | Mean Square | F | Sig.
1 | Regression | 283.511 | 1 | 283.511 | 49.075 | .000[a]
Residual | 1998.869 | 346 | 5.777
Total | 2282.380 | 347
2 | Regression | 403.423 | 2 | 201.711 | 37.037 | .000[b]
Residual | 1878.957 | 345 | 5.446
Double click to edit Pivot Table
IBM SPSS Statistics Processor is ready
H: 1.66, W: 8.21 in

R and RStudio



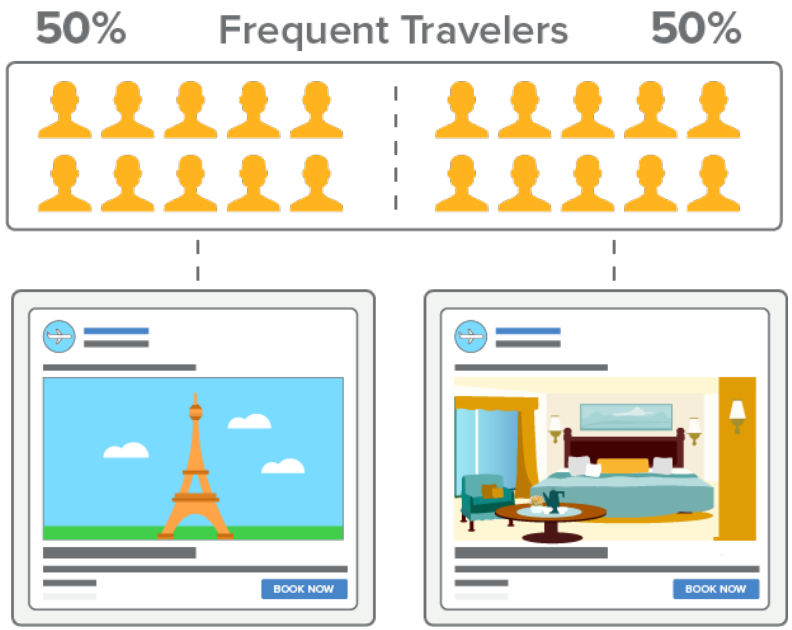
Life Cycle of Data Science

Data Science Examples

- Marketing Research
- Sports
- Medicine
- Banking

Marketing Example: A case from Adphorus Company

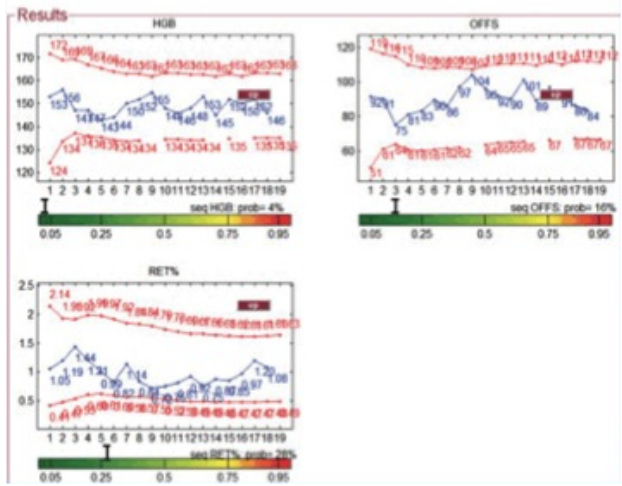
- **Optimization** of Facebook advertisements
- Introducing Customers the products they actually are **interested** in
- Gathering data from Facebook, Instagram and Twitter to find out the **behaviour** of each customer
- Using Hypothesis tests to determine whether a certain ad is **clicked** or **not**.



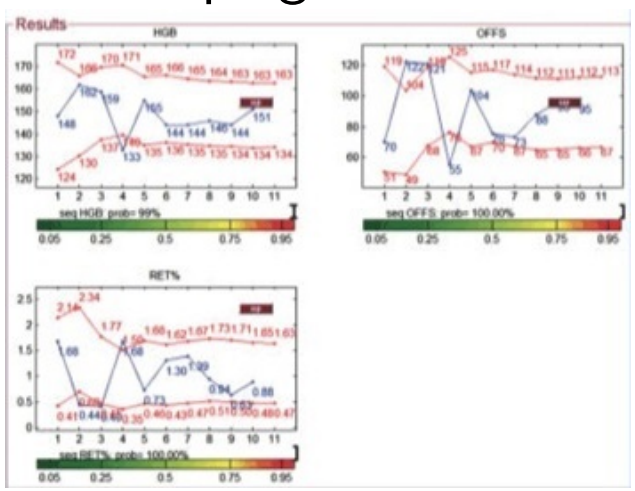
Sports Example: A case from Rio Olympics 2016

- Potential **doping activities** of various athletes
- New substances are developed to trick the system, hard to **detect new drugs**
- Athlete Biological Passport (ABP) has been introduced to monitor certain **biological variables (markers)** of athletes, singling out those who may have used a potential drug.

Clean Athlete



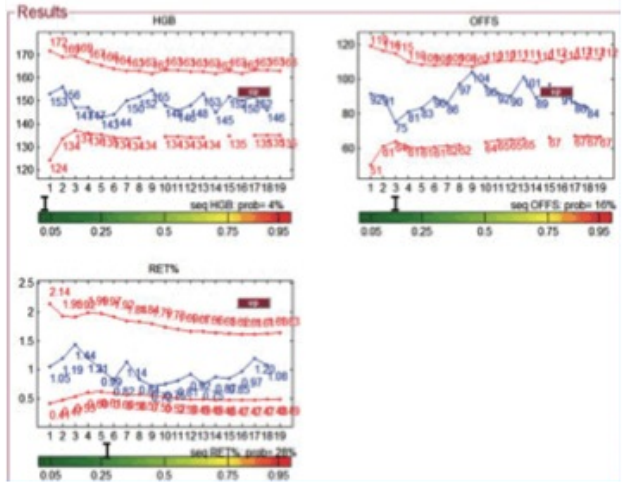
Doping Athlete



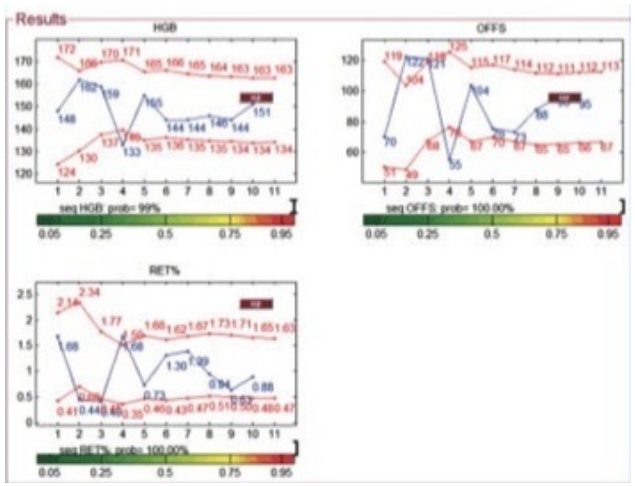
Sports Example: A case from Rio Olympics 2016

- Detecting a **sudden change** in the athletes metobalism or biology by **machine learning algorithms**.
- Find which biological variables should be used, prepare the data, find the best algorithm to detect possible drug use.

Clean Athlete



Doping Athlete



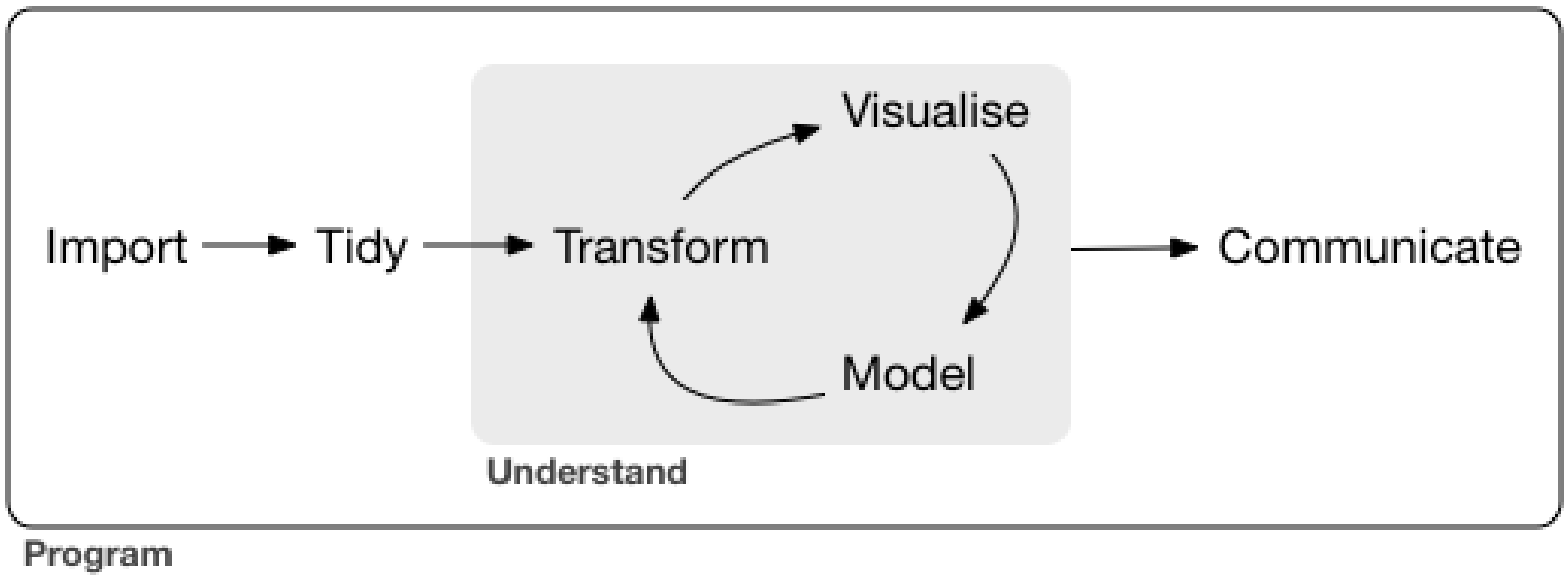
Medicine Example: Personalized Health Care

- **Several drugs** are tested on a patient with high blood pressure.
- Checking patients with similar characteristics to find the **best drug**

Banking Example: Credit Card Assessment

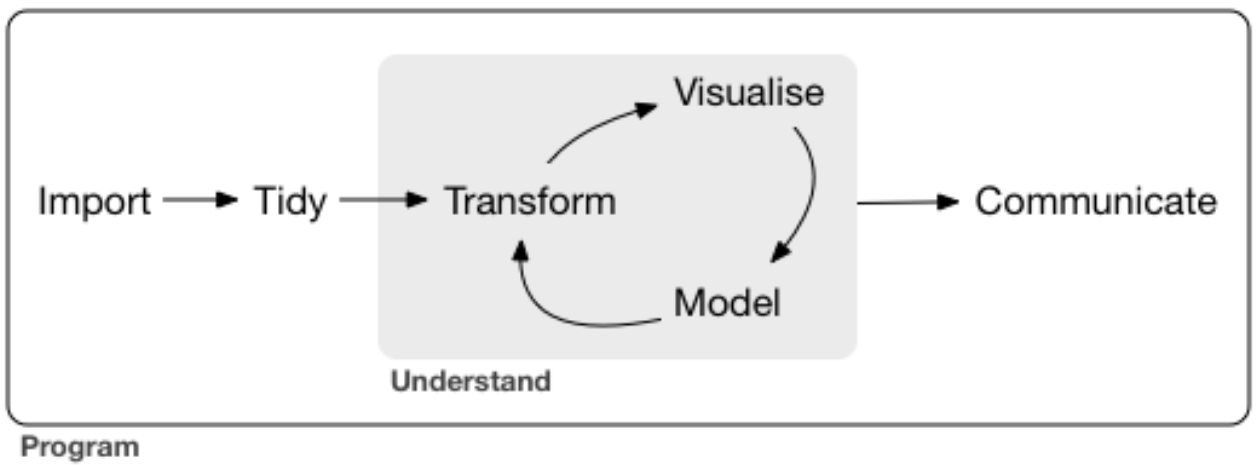
- Estimating whether clients can reject a card?
- Activate herself/himself? activate because the bank called?
- Defining what should be predicted? what model to use?
- Interpretable methods: what indicator contributes how?

The Process of Data Science



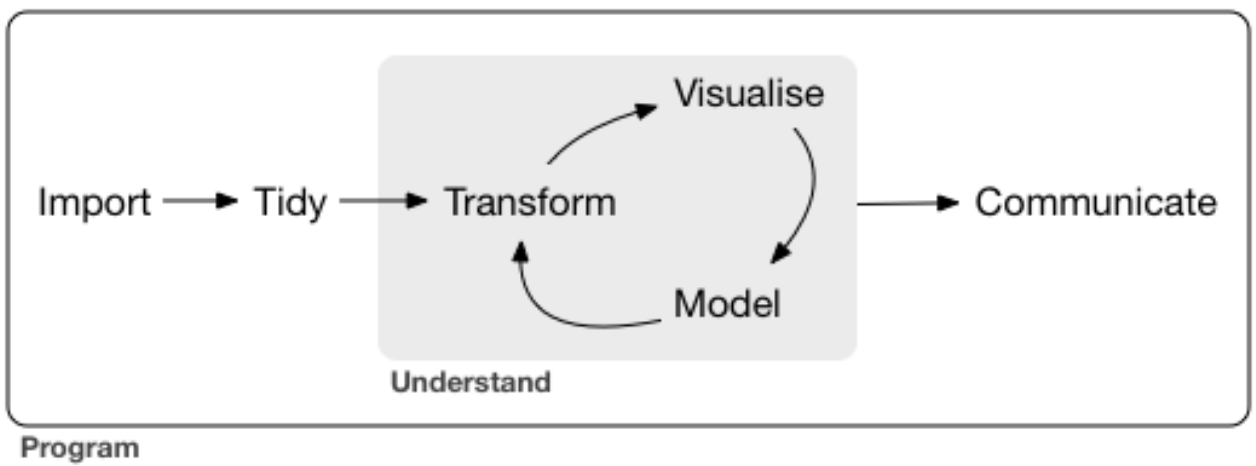
The Process of Data Science

- **Importing data**: Identify the resources for getting the data, means for importing it
- **Tidy data**: reshape the data in a way that its convenient to transform and analyze
- **Transform data**: cleaning, imputation, and further reshaping for analysis and model building



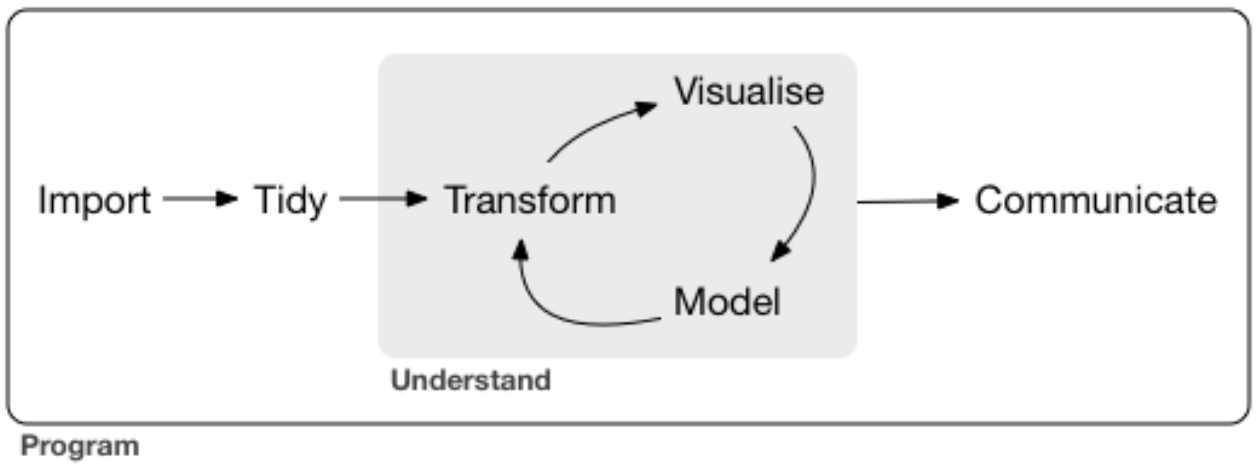
The Process of Data Science

- **Analyze data**: Data analysis tools to investigate data and understand what methods and algorithms to be used.
- **Model data**: The model building phase where the actually decision making and prediction is done.
- **Communicate data**: Describe and explain your models and results to those who use it and benefit from it.



Data Science with R

- All these steps require **flexible**, **easy to write** and **resourceful** programming languages that can analyze data.
- R is a powerful tool for data science that can **import**, **manipulate** and **analyze** data on multiple levels.



Outline of This Course

- Scripting Languages for data analysis
  - R programming
- Data Management
  - Data manipulation
  - Data cleaning and Missing data analysis
- Data Analysis
  - Exploratory data analysis and data visualization
  - Inferential Statistics
- Machine Learning
  - Introduction to Machine Learning
  - Supervised Learning
  - Unsupervised Learning

Outline of This Course

- We will have Quizzes each week (or may be not)
- One Midterm
- Final
- All on computers!
- No paper exam!

Material

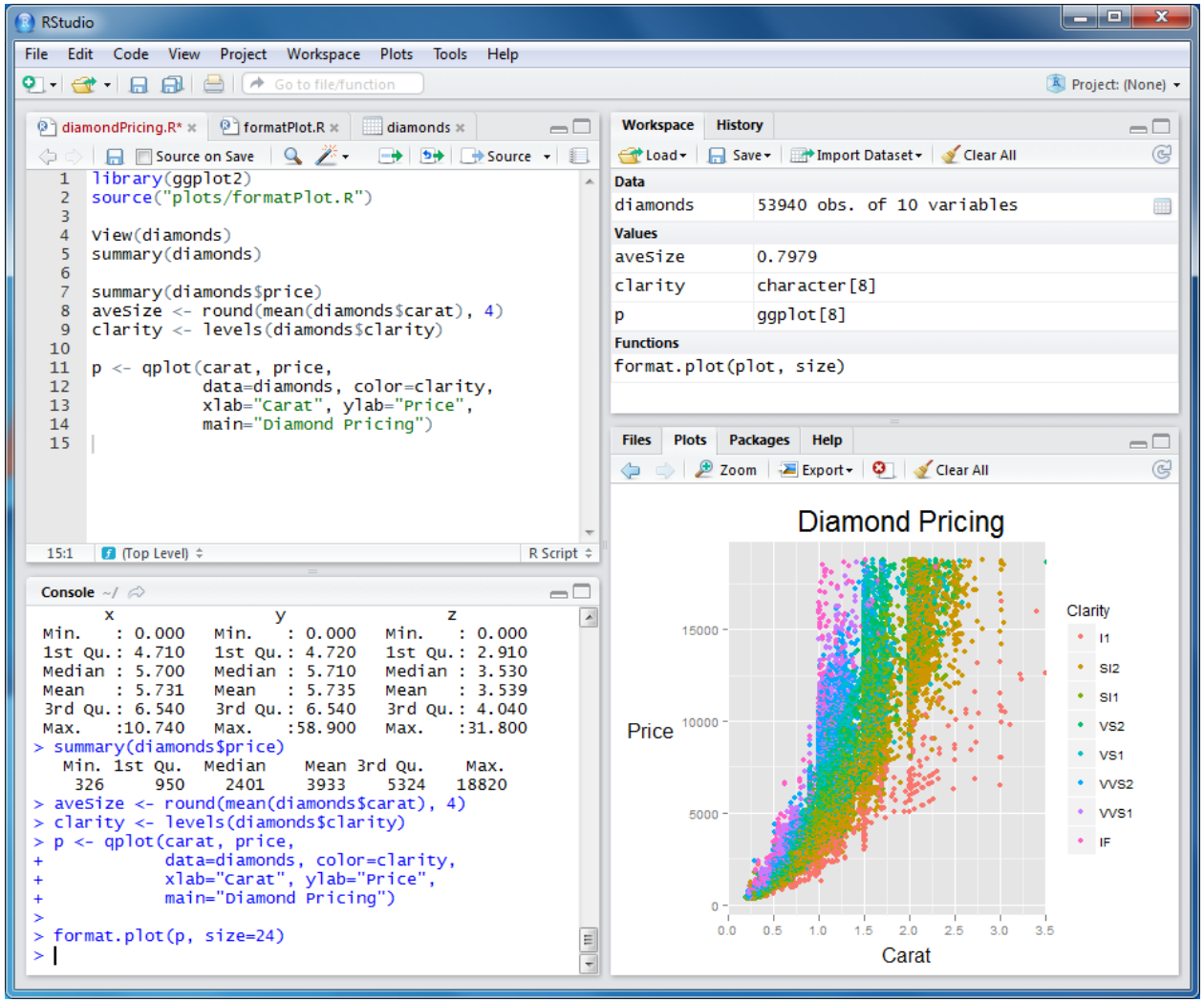
- Practical Data Science with R, Nina Zumel
- Data Manipulation with R, Phil Spector
- Efficient R programming, Colin Gillespie
- Exploratory Data Analysis with R, Roger D. Peng
- Introduction to Statistical Learning with Applications in R, Gareth James

What is R?

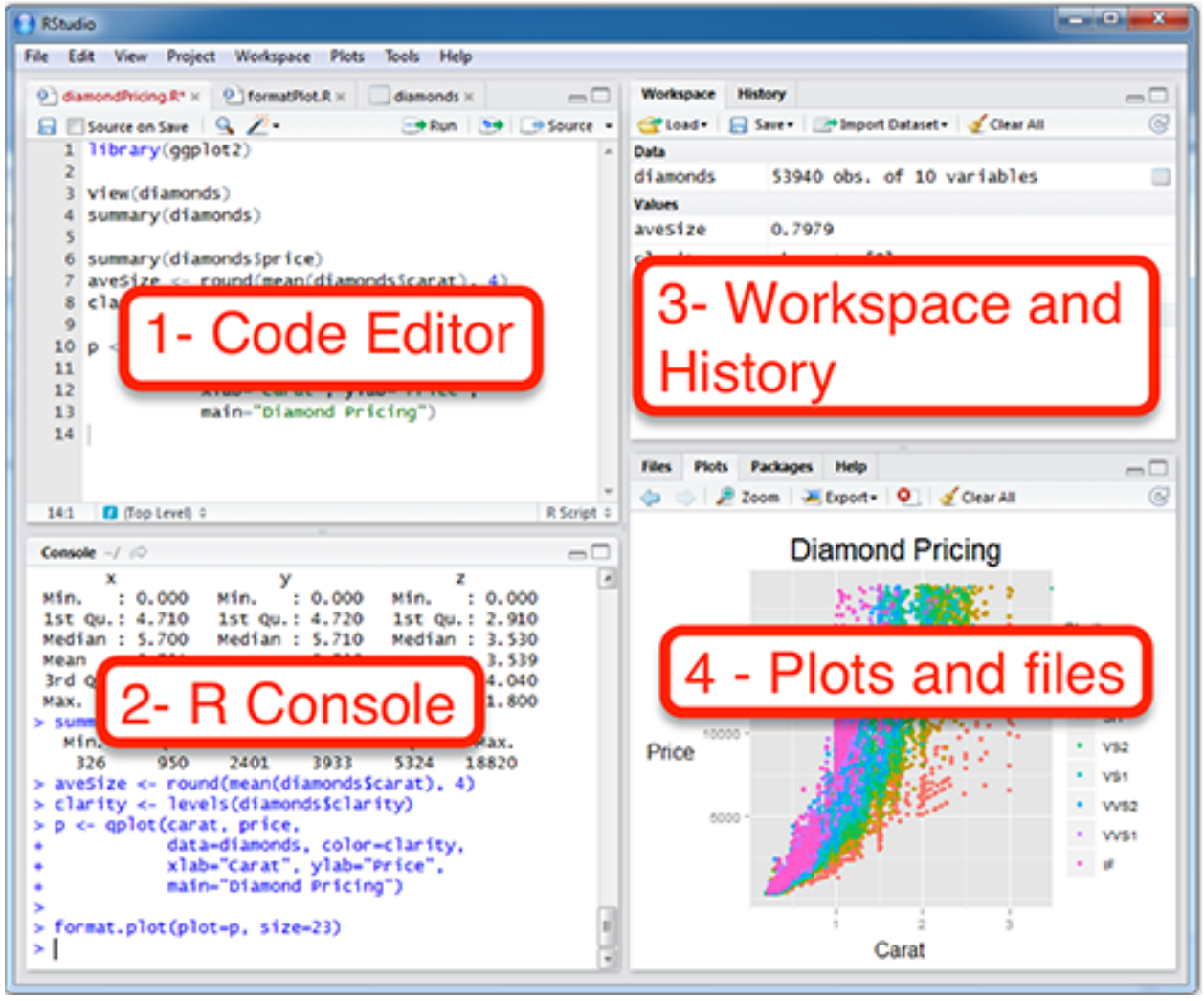
- R is a suite of software facilities for:
  - Reading and manipulating data
  - Computation
  - Conducting statistical data analysis
  - Application and development of Machine Learning Algorithms
  - Displaying the results
- open-source version (i.e. freely available version - no license fee) of the S programming language, a language for manipulating **objects**.
- Software and packages can be downloaded from www.cran.r-project.org

Rstudio

- Also download Rstudio from www.rstudio.com



Rstudio


RStudio
File Edit View Project Workspace Plots Tools Help
diamondPricing.R* formatPlot.R diamonds
Source on Save Run Source
1 library(ggplot2)
2
3 view(diamonds)
4 summary(diamonds)
5
6 summary(diamonds$price)
7 aveSize <- round(mean(diamonds$carat), 4)
8 cla
9
10 p <
11
12
13 main="Diamond Pricing")
14
14:1 (Top Level) R Script
1- Code Editor
Workspace History
Load Save Import Dataset Clear All
Data
diamonds 53940 obs. of 10 variables
Values
aveSize 0.7979
3- Workspace and History
Files Plots Packages Help
Zoom Export Clear All
Diamond Pricing
Price
Carat
4 - Plots and files
Console ~/
x y z
Min. : 0.000 Min. : 0.000 Min. : 0.000
1st Qu.: 4.710 1st Qu.: 4.720 1st Qu.: 2.910
Median : 5.700 Median : 5.710 Median : 3.530
Mean 3.539
3rd Q 4.040
Max. 1.800
> summ
Min. ax.
326 950 2401 3933 5324 18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+ data=diamonds, color=clarity,
+ xlab="Carat", ylab="Price",
+ main="Diamond Pricing")
>
> format.plot(plot=p, size=23)
>
2- R Console

Objects

- R works by creating **objects** and using various functions calls that **create** and **use** these objects. For example;
  - Vectors of numbers, logical values (TRUE and FALSE), character strings and even complex numbers
  - Matrices and general n-way arrays
  - Lists - arbitrary collections of objects of any type; e.g. list of vectors, list of matrices, etc.
  - Data frames - a general data set type
  - functions (yes even functions are objects)
- See **object oriented programming**: each object has a set of values and functions that interacts with other objects of the same or different types.

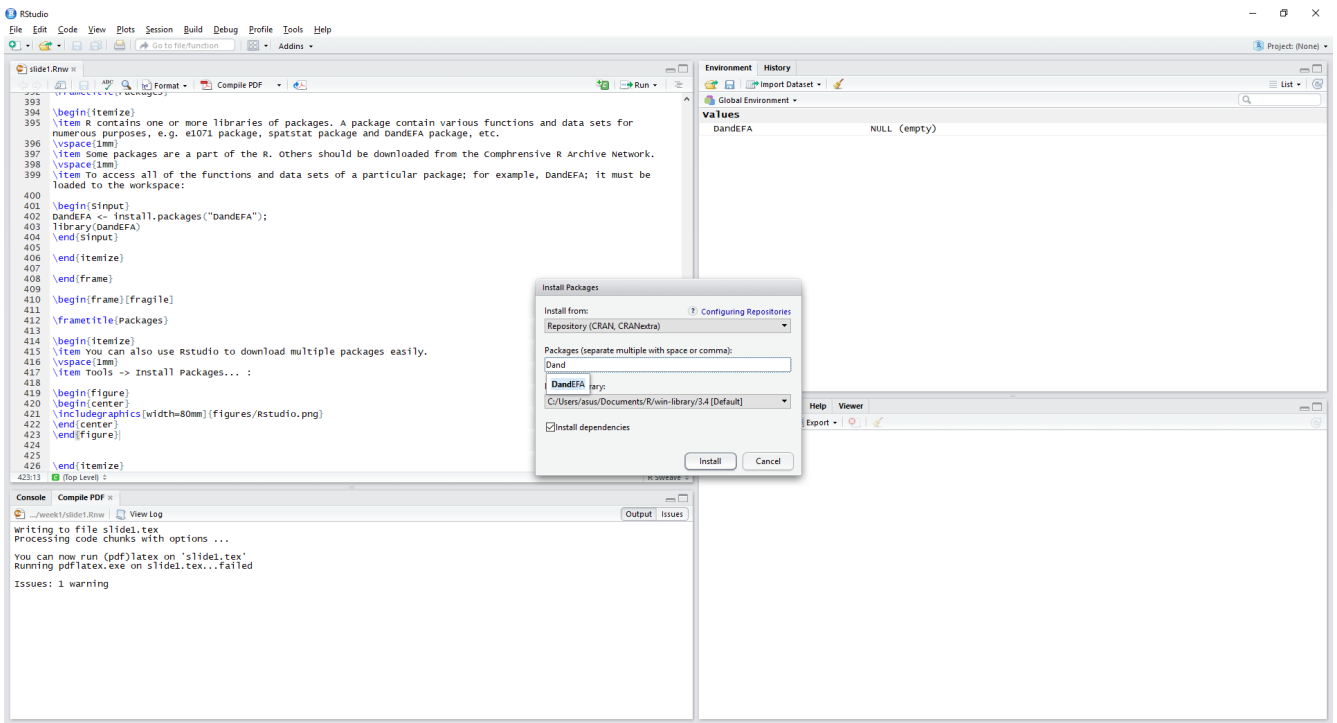
Packages

- R contains one or more libraries of packages. A package contain various functions and data sets for numerous purposes, e.g. e1071 package, spatstat package and DandEFA package, etc.
- Some packages are a part of the R. Others should be downloaded from the Comphrensive R Archive Network.
- To access all of the functions and data sets of a particular package; for example, DandEFA; it must be loaded to the workspace:

```
install.packages("DandEFA");
library(DandEFA)
```

Packages

- You can also use Rstudio to download multiple packages easily.
- Tools -> Install Packages... :



DandEFA Packages

- Using packages to utilize various methods and algorithms.
- **DandEFA** package contains functions for a particular analysis called **factor analysis**.
- Factor Analysis is a method for categorize variables into groups to find the relationship between the variables in the same group
- The package contains functions:
  - **factload**: A method for producing the factor loadings
  - **dandelion**: A method for visualizing the factor loadings

R Packages

- To check what packages are currently loaded in the workspace

```
> search()
```

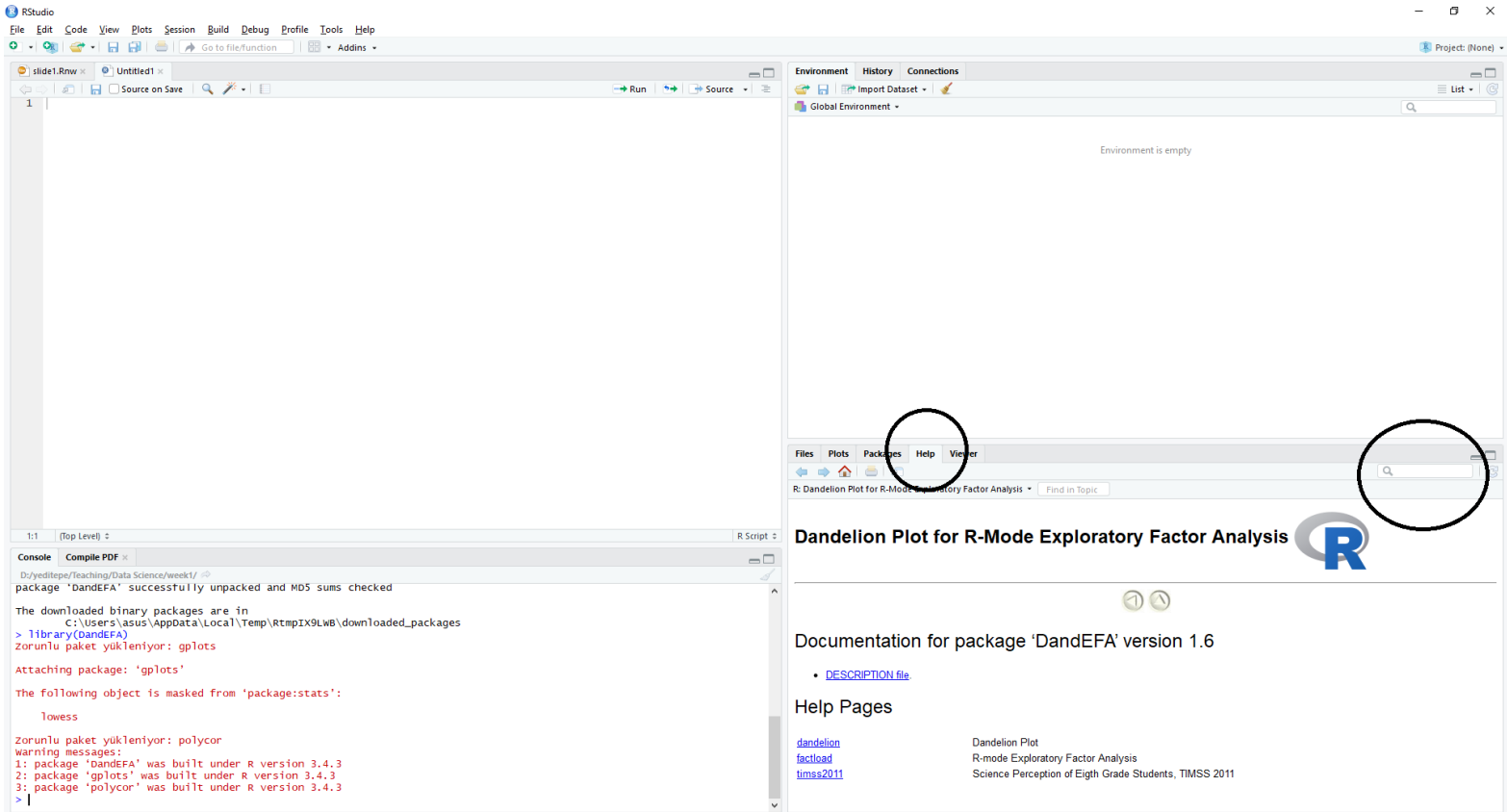
```
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

- To get more information on a package, use the `help()` function

```
> help(package="DandEFA")
```

Information on Packages

- You can also use the bottom-right panel in RStudio to get info on a package:



Description of DandEFA

- DESCRIPTION file includes the information on the package (first thing to look at)

```
> packageDescription("DandEFA")
```

```
Package: DandEFA
Type: Package
Title: Dandelion Plot for R-Mode Exploratory Factor Analysis
Version: 1.6
Date: 2016-03-14
Author: Artur Manukyan, Ahmet Sedef, Erhan Cene, Ibrahim Demir
Depends: R(>= 3.2.3), gplots, polycor
Maintainer: Artur Manukyan <artur-man@hotmail.com>
Description: Contains the function used to create the Dandelion Plot.
        Dandelion Plot is a visualization method for R-mode Exploratory
        Factor Analysis.
License: GPL-2
NeedsCompilation: no
Packaged: 2016-03-14 08:33:37 UTC; Artur
Repository: CRAN
Date/Publication: 2016-03-14 12:57:57
Built: R 4.0.3; ; 2020-11-23 06:22:27 UTC; windows
```

dandelion() function

```
> library(DandEFA)
> help(dandelion)
```

Dandelion Plot

Description

A Dandelion plot for R-mode Exploratory Factor Analysis methods. The loading matrix and the factor variances are being visualized.

Usage

```
dandelion(fact_load, bound = 0.5, mcex=c(1,1), palet)
```

Arguments

`fact_load`	A "loadings" class object. Factor loading matrix.
`bound`	Minimum loadings to visualize. It should be set between 0 and 1. For example, bound=0.5 will only visualize loadings more than 0.5.
`mcex`	A vector with two points. First value determines the size of labels within dandelion plot, and the second determines the size of labels within uniquenesses and communalities graphs
`palet`	A vector of color pallette. The first and the last elements of the vector are the colors of positive and negative loadings.

Details

A Dandelion Plot visualizes both factor variances and loadings in the same time. Each central line represents a different factor and is connected to a star graph. These star graphs visualize the factor loadings for the corresponding factor. Negative and positive loadings are indicated by two different colors. Explained variance of each factor can be observed by the size of each star graph or by the angle between the current and the consecutive central line. For example, explained variance of first factor is determined by the angle between the first and second central line. Communalities and uniquenesses are also given on the right hand side along barchart of cummulative explanation ratios of factors (with individual variances on top).

Author(s)

Artur Manukyan, Ahmet Sedef, Erhan Cene, Ibrahim Demir

References

Artur Manukyan, Erhan Cene, Ahmet Sedef, Ibrahim Demir, *Dandelion plot: a method for the visualization of R-mode exploratory factor analyses.* Computational Statistics 29.6 (2014): 1769-1791.

Examples

```
# E.F.A. of Timss 2011 Student Questionnaire Example for 5 and 8 number of factors
data(timss2011)
timss2011 <- na.omit(timss2011)
dandpal <- rev(rainbow(100, start = 0, end = 0.2))
facl <- factload(timss2011,nfac=5,method="prax",cormeth="spearman")
dandelion(facl,bound=0,mcex=c(1,1.2),palet=dandpal)
facl <- factload(timss2011,nfac=8,method="mle",cormeth="pearson")
dandelion(facl,bound=0,mcex=c(1,1.2),palet=dandpal)
```

DandEFA Package

- Loading the library

```
> library(DandEFA)
```

- Loading the TIMSS 2011 data

```
> data(timss2011)
```

- Deleting those rows that have empty cells

```
> timss2011 <- na.omit(timss2011)
```

- Choosing a coloring palette for the visualization

```
> dand_palette <- rev(rainbow(100, start = 0,
+                             end = 0.2))
```

DandEFA Package

- Producing the factor loadings

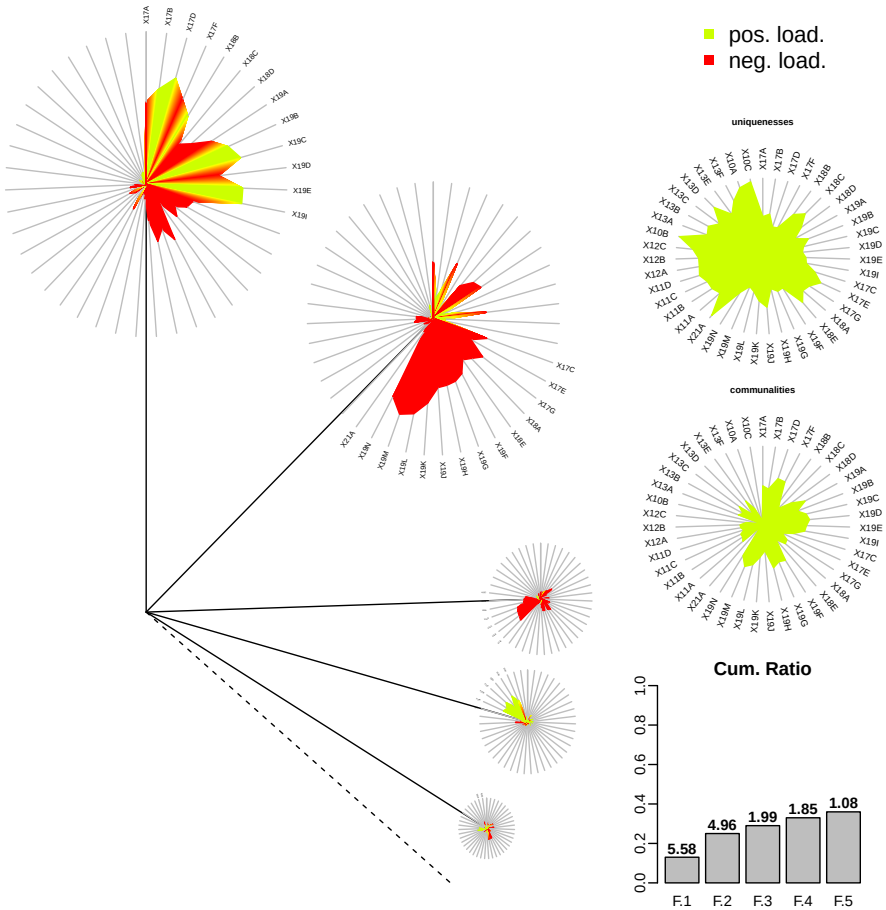
```
> factor_loadings <- factload(timss2011,nfac=5,
+                   method="prax",cormeth="spearman")
> factor_loadings

Loadings:
     [,1]   [,2]   [,3]   [,4]   [,5]
X10A                 0.103 -0.101 -0.224
X10B
X10C         0.106                -0.129
X11A                        -0.544 -0.130
X11B                        -0.514
X11C -0.129 -0.105          -0.500
X11D                        -0.475
X12A -0.116        -0.152 -0.338  0.318
X12B        -0.254 -0.133 -0.328  0.256
X12C        -0.149 -0.136 -0.298  0.249
X13A         0.549
X13B         0.504
X13C         0.583
X13D         0.398
X13E         0.595
X13F         0.458
```

DandEFA Package

- Visualizing the dandelion plot

```
> dandelion(factor_loadings,bound=0,mcex=c(0.5,0.6),
+                palet=dand_palette)
```



R Packages

- In summary, packages provide a flexible environment.
- Employing multiple methods and algorithms in the same time
- Programming and using packages

R programming

Vectors are the simplest type of objects in R. There are 3 main types of vectors:

- Numeric Vectors

  ```
  > x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
  ```

- Character vectors

  ```
  > x <- c("boy","girl","boy","girl","boy","boy")
  ```

- Logical vectors

  ```
  > x <- c(TRUE,TRUE,FALSE,TRUE,TRUE,FALSE)
  ```

  or

  ```
  > x <- c(T,T,F,T,T,F)
  ```

Vectors

- Accessing a particular element of the vectors is straightforward:

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> x[5]
```

```
[1] 21.7
```

- You can access multiple of them by defining indices or logical vectors

```
> ind <- c(1,4,5)
> x[ind]
```

```
[1] 10.4  6.4 21.7
```

```
> ind <- c(T,F,F,T,T)
> x[ind]
```

```
[1] 10.4  6.4 21.7
```

Vectors

- A logical operation over a vector would create a logical vector (important!!)

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> ind <- (x > 7)
> ind
[1]  TRUE FALSE FALSE FALSE  TRUE
> x[ind]
[1] 10.4 21.7
> x[!ind]
[1] 5.6 3.1 6.4
```

- We will use indices to manipulate data sets later. But a shorter version of the code is

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> x[x > 7]
[1] 10.4 21.7
```

Vectors

- A logical operator checks whether the both sides have equal length or one side has length 1

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> y <- c(4,7,8,2,35)
> ind <- (x > y)
> ind
```

```
[1]  TRUE FALSE FALSE  TRUE FALSE
```

- If the number of elements are not equal:

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> y <- c(4,7,8,2)
> ind <- (x > y)
```

```
Warning message:
 In x > y : longer object length is not a multiple of
 shorter object length
```

Modifying vectors

- Any element of the vector can be modified easily:

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> x[4] <- 7.3
```

- A group of elements can be modified too

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> x[x > 7] <- -x[x > 7]
> x
```

```
[1] -10.4   5.6   3.1   6.4 -21.7
```

- Some advance stuff: (data imputation)

```
> x <- c(10.4, NA, 3.1, 6.4, NA)
> is.na(x)
```

```
[1] FALSE  TRUE FALSE FALSE  TRUE
```

```
> x[is.na(x)] <- mean(x, na.rm = TRUE)
> x
```

```
[1] 10.400000  6.633333  3.100000  6.400000  6.633333
```

Manipulating vectors

- Merging vectors with c():

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> y <- c(4, 7, 8, 2, 35)
> z <- c(x,y)
> z
```

```
 [1] 10.4  5.6  3.1  6.4 21.7  4.0  7.0  8.0  2.0 35.0
```

- Summation or multiplication over vectors. **Note**: Again both vectors either have be of same size or one has to be of length one

```
> z <- x + y
> z
```

```
[1] 14.4 12.6 11.1  8.4 56.7
```

```
> z <- x*y
> z
```

```
[1]  41.6  39.2  24.8  12.8 759.5
```

Generating sequences

- the colon ":", e.g.

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x <- 2*(1:10)
> x
 [1]  2  4  6  8 10 12 14 16 18 20
```

- the `seq()` function.

```
> x <- seq(1,10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x <- seq(1,10,by=0.5)
> x
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7
[16]  8.5  9.0  9.5 10.0
```

Generating sequences

- the `rep()` function

```
> x <- rep(3, 10)
> x
 [1] 3 3 3 3 3 3 3 3 3 3
> y <- rep(c(F,T,F,T,T,T),3)
> y
 [1] FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
[10]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
> y <-  rep(c(4,7,8,2,35),each=3)
> y
 [1]  4  4  4  7  7  7  8  8  8  2  2  2 35 35 35
```

Factors

A factor is a special type of vector used to represent categorical data, e.g. gender, social class, etc.

- Stored internally as a numeric vector with values 1, 2, : : : ; k, where k is the number of levels.
- Can have either ordered and unordered factors.
- A factor with k levels is stored internally consisting of 2 items.
  - a vector of k integers
  - a character vector containing strings describing what the k levels are.

Factors

Five people are asked to rate the performance of a product on a scale of 1-5, with 1 representing very poor performance and 5 representing very good performance. The following data were collected.

- We have a numeric vector containing the satisfaction levels.

```
> satisfaction <- c(1, 3, 4, 2, 2, 3, 4, 2, 1, 2, 1, 1, 4, 3)
```

- Want to treat this as a categorical variable and so the second line creates a factor. The levels=1:5 argument indicates that there are 5 levels of the factor. We also set the labels for each factor.

```
> fsatisfaction <- factor(satisfaction, levels=1:5,
+                         labels = c("very poor", "poor",
+                                    "average","good", "very good"))
```