INTRODUCTION TO DATA ANALYSIS

Partha Padmanabhan

UCSC Silicon Valley Extension
YOUR UC IN SILICON VALLEY

Lecture 4

Flow Control/Loops

Often when we are coding we want to control the flow of our actions.
Control flow (or flow of control) is simply the order in which we code and have out statements evaluated. That can be done by setting things to happen only if a condition or a set of conditions are met.

- Control structures in R contains conditional, loop statements like any other programming languages.

- Loops are very important and forms backbone to any programming languages. These allow one to control the flow of execution of a script typically inside of a function. Common ones include:

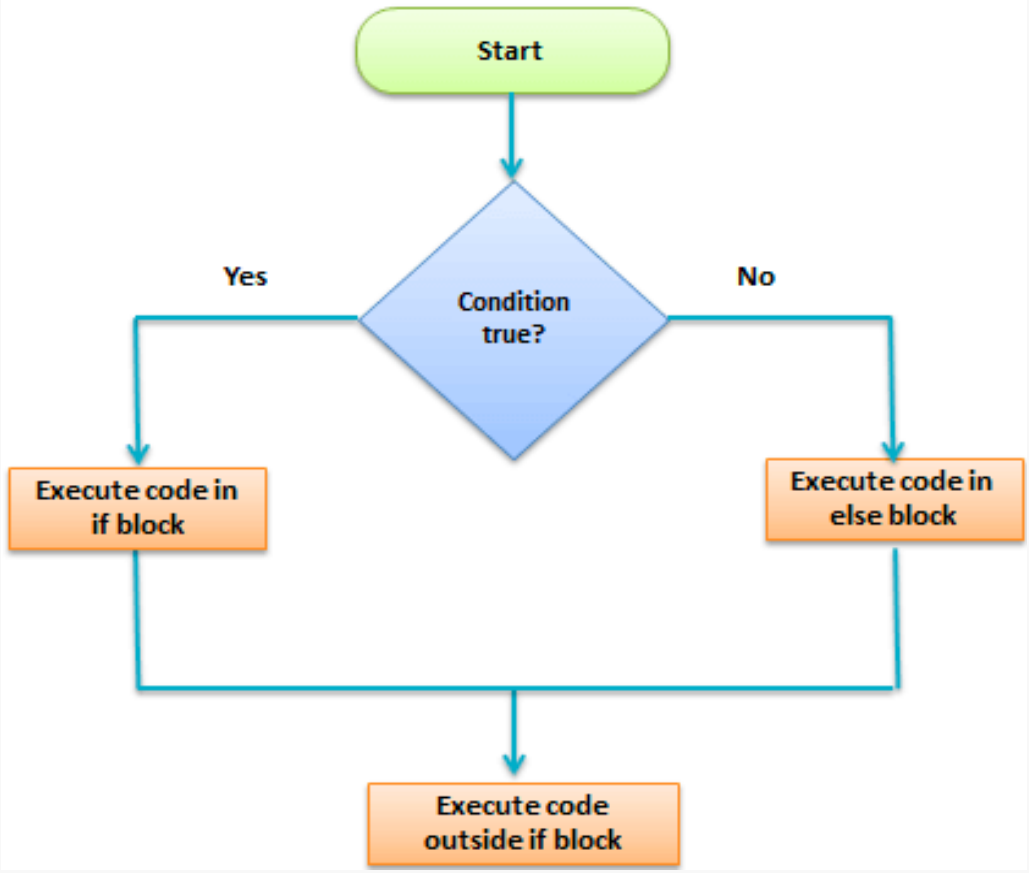
– if, else – for
– while – repeat – break – next
– return

If...else statement
Decision making is an important part of programming. This can be achieved in R porgramming using the conditional “if...else” statement

The syntax of if statement is
If (test expression) { statement
}

If the test expression is TRUE, the statement gets executed. But if it’s FALSE, nothing happens. Test expression can be logical or numeric. In case of numeric vector, zero is taken as FALSE, rest as TRUE

Flow Chart of If Statement



Example: we want to check if the number is positive, if so print positive number.

```
x<-10
 if(x>0) {
print(“Positive Number”) }
```

Output is:

```
[1] "positive Number"
```

if...else statement

The syntax of if...else statement is:
if (test1) { statement1
} else { statement2
}

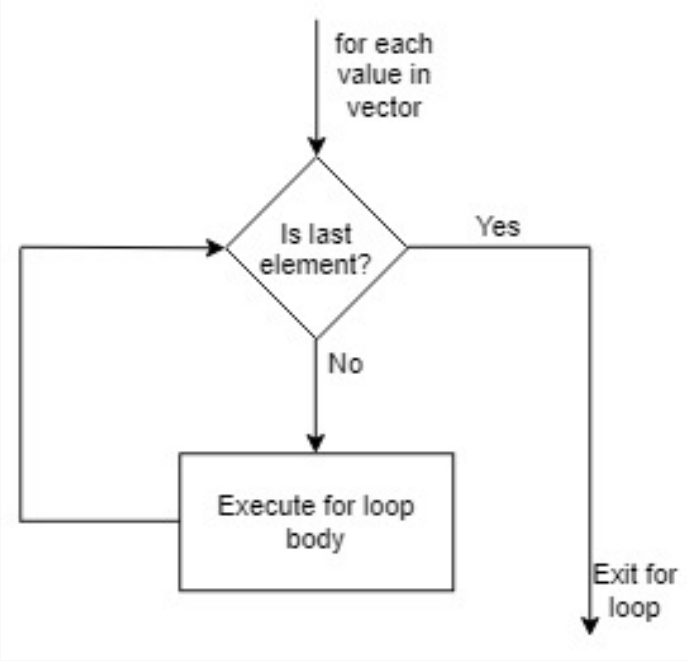
Flow chart of if...else statement
x <- -10
 if(x > 0){
print(”Positive number") } else {
print("Negative number") }
Output is
[1] “Negative number”

Nested if...else statement
We can nest as many if...else statement as we want as follows. The syntax of nested if...else statement is:

```
if ( test1) { statement1
} else if ( test2) { statement2
} else if ( test3) { statement3
} else statement4
```

Here you will learn for loop in R . A for loop is used to iterate over a vector in R programming
for (val in vect1) {
statement }

Here vect1 is a vector and val takes on each of its value during the loop. In each iteration, statement is evaluated.


for each value in vector
Is last element?
Yes
No
Execute for loop body
Exit for loop

Example: for loop
Below is an example to count the number of even numbers in a vector.

```
x <- c(21,15,13,9,8,10,5)
count <- 0
for (val in x) {
if(val %% 2 == 0) count = count+1 }
print(count)
```

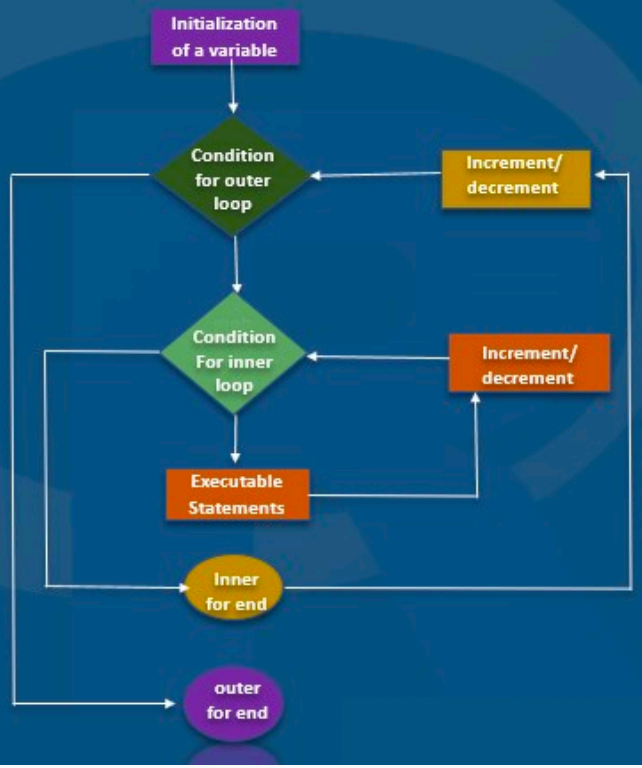
Output is

```
print(count)
 [1] 2
```

In the above example, the loop iterates 7 times as the vector x has 7 elements. In each iteration, val takes on the value of corresponding element of x.

We have used a counter to count the number of even numbers in x. We can see that x contains 2 even numbers.

Nested For Loop in R



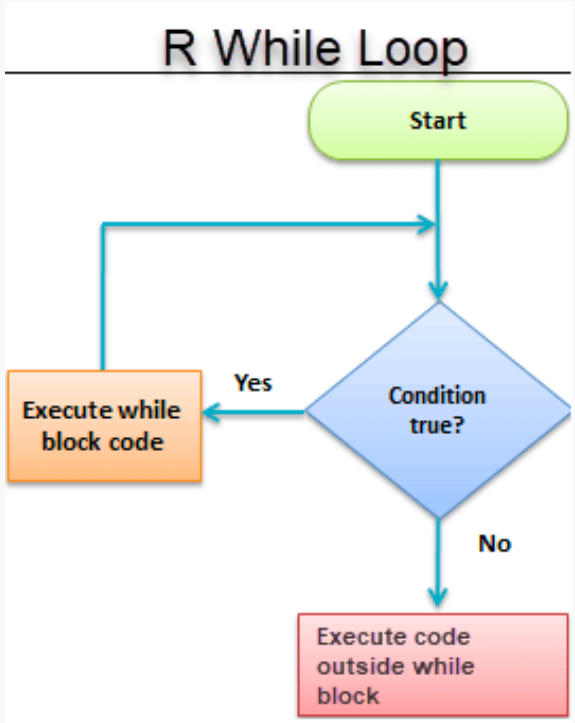
while loop
while loops are used to loop until a specific condition is met.

while(test) { statement
}

test is evaluated and the body of the loop is entered if the result is TRUE.

The statements inside the loop is executed and the flow returns to evaluate the test again.
This is repeated each time until test evaluates to FALSE, in which case the loop exits

Flowchart of while loop



Functions

Functions are used to break our code into simpler parts which become easy to maintain and understand and which can be reused later. Functions allow you to automate common tasks in a ore powerful and general way than copy-and-pasting. Though R provides lots of built-in functions, you can create your own also. Writing a function has following advantages over copy-and-paste:

- You can give a function a name that makes your code easier to understand
- As requirements change, you only need to update code in one place, instead of many
- You eliminate the chance of making incidental mistakes when you copy and paste.

When should you write a function?
You should consider writing a function whenever you've copied and pasted a block of code more than twice.

Functions in R are created using function() and are stored as R objects just like anything else. It is pretty straight forward to create your own function in R.

```
sum_2num <-function(arg1, agr2, ...) { statement
}
```

There are three key steps to creating a new function:

- You need to pick a name for the function. Here I've used sum_2num
- You list the inputs, or arguments, to function inside function.
- You place the code you have developed in body of the function, a { block that immediately follows function().

Functions have names arguments which potentially have default values

- The formal arguments are the arguments included in the function definition
- The formals function returns a list of all the formal arguments of a function
- Not every function call in R makes use of all the formal arguments
- Function arguments can be missing or might have default values
- R functions arguments can be matched positionally or by name.

Load the function into R session
For R to be able to execute your function, it needs first to be read into memory. This is just like loading a library, until you do it the functions contained within it cannot be called. There are two method for loading functions into the memory:

1. Copy the function text and paste it into the console
2. Use the source() function to load your function from file

It is recommended that you should use the second of these options.
Put your functions into a file with an intuitive name, like power-fun.R and save this file within the R folder in your working project. You can then read the function into memory by calling:
source("/Users/home/projects/power-fun.R")

From the point of view of writing nice code, this approach is nice because it leaves you with an uncluttered analysis script, and a repository of useful functions that can be loaded into any analysis script in your project. It also lets you group related functions together easily.

Using your function

You can use the function anywhere in your analysis. In thinking about how you use functions consider the following:

- Functions in R can be treated much like any other R object
- Functions can be passed as arguments to other functions or returned from other functions
- You can define a function inside of another function.

Example

```
fahrenheit_to_celsius <- function(temp_F) {
  temp_C <- (temp_F - 32) * 5 / 9
  return(temp_C)
}

fahrenheit_to_celsius(40)
```

4.444444

Default Values for arguments
We can assign default values to arguments in a function in R. This is done by providing an appropriate value to the formal argument in the function declaration

```
pow <-function(x1,x2 = 2){ temp = x1^x2
print( paste(x1,"raised to the power", x2, "is" temp))
}
```

The use of default value to an argument makes it optional when calling the function

```
> pow(3)
[1] "3 raised to the power 2 is 9"
```

- pow(3,1)
  [1] "3 raised to the power 1 is 3”

Here, x2 is optional and will take the value 5 when not provided

Help in R

Help in R

There is a large amount of documentation and help available. Some help is automatically installed. There are various ways to get help

>?<func name> # when you know the function >help(command name) # gives description of function help.search("string", package = "ggplot2")

> HTML-based global help can be called with >help.start()

>??<topic>
The example() function
Each of the entries comes with examples. One really nice feature of R is that the example() function will actually run those examples for you.

Help in R

If you don't know quite what you're looking for,
you can use the function help.search() to do a Google-style search through R's documentation.

For example, say you need a function to generate random variates from multivariate normal distributions. To determine which function, if any, does this, you could try something like this:

>help.search("multivaraite normal")

There is also a question-mark shortcut to help.search()
You can also learn about the entire package by typing this: >help(package = "e1071")

Data Analysis Methodology

Project Definition
Involves establishing the scope of project and understanding what the business case would like to improve on or the problem it wants solved

Data Collection
Gathering accurate and valid information on target variables identified as data requirements or are relevant to the case on hand

Data Processing
Involves the structuring and transformation of data so that it is organized for analysis

Data Cleaning
Processed data may contain outliers, duplicate or missing values. Data Cleaning ensures that data is complete and error free for analysis

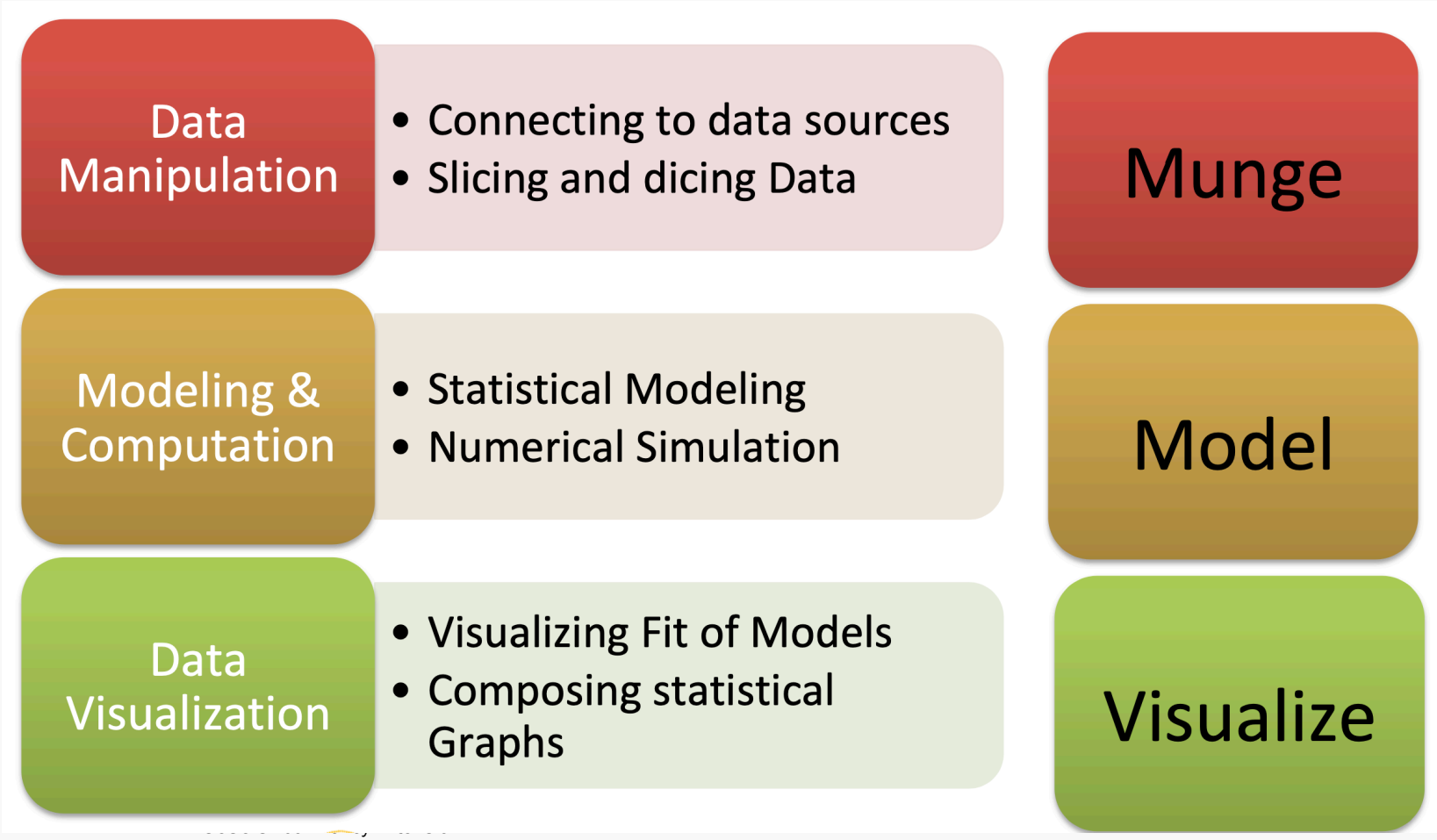
Exploratory Data Analysis
Involves the initial visualization and understanding of the data such as the identification of skewness, quartiles and distribution

Model Development
By using Machine Learning techniques and algorithms, we develop predictive models to meet our clients' unique needs

Model Refinement
By further refining our predictive models, we attempt to develop the most accurate and robust predictive models

Insights & Reporting
From the new insights derived from our predictive models, we offer our clients with actionable recommendations to aid their decision making

3 things you can do with R



Top 10 things to work with R

1. **You can write reproducible Word or PowerPoint documents from R markdown**
2. **You can build and host interactive web apps in just a few lines of code**
3. **You can host your web apps in one more line of R code**
4. **You can connect to almost any database under the sun and pull data with dplyr/dbplyr**
5. **You can use the same dplyr grammar locally or on data on multiple different data stores**
6. **You can fit deep learning models with keras and Tensorflow**
7. **You can build APIs and serve them from R**
8. **You can make video game interfaces with R**
9. **You can analyze data using Spark clusters right from R**
10. **You can build and learn R interactively in R**

Useful Links

The following links can be very useful:

- http://cran.r-project.org/doc/manuals/R-intro.pdf
- https://www.r-project.org/other-docs.html
- http://cran.r-project.org/doc/contribute/Short-refcard.pdf

http://zoonek2.free.fr/UNIX/48_R/all.html

A very rich source of example

- http://rwiki.sciviews.org/doku.php
- http://www.statmethods.net/
- http://mathesaurus.sourceforge.net/

A long list of Books on ro Using R

https://www.r-project.org/doc/bib/R-books.html

Useful Links

•Cran-R
– http://cran.r-project.org/other-docs.html – http://cran.r-project.org/web/views/
•• R-bloggers.com (Every analyst should subscribe)
•• Quick-R, Inside-R
•• http://www.cookbook-r.com
•• StackOverflow
•htp://stackoverflow.com/questions/tagged/r

Data Analysis

- The starting point for data analysis is a data set which contains the measured or collected data values represented as numbers or text. The data is raw before it has been transformed or modified.
- All the disciplines collect data about items that are important to that field. These items are organized into a table for data analysis where each row , referred to as an observation, contains information about the specific item. Data table also contain information about the object and is known as attribute.

What is Data?

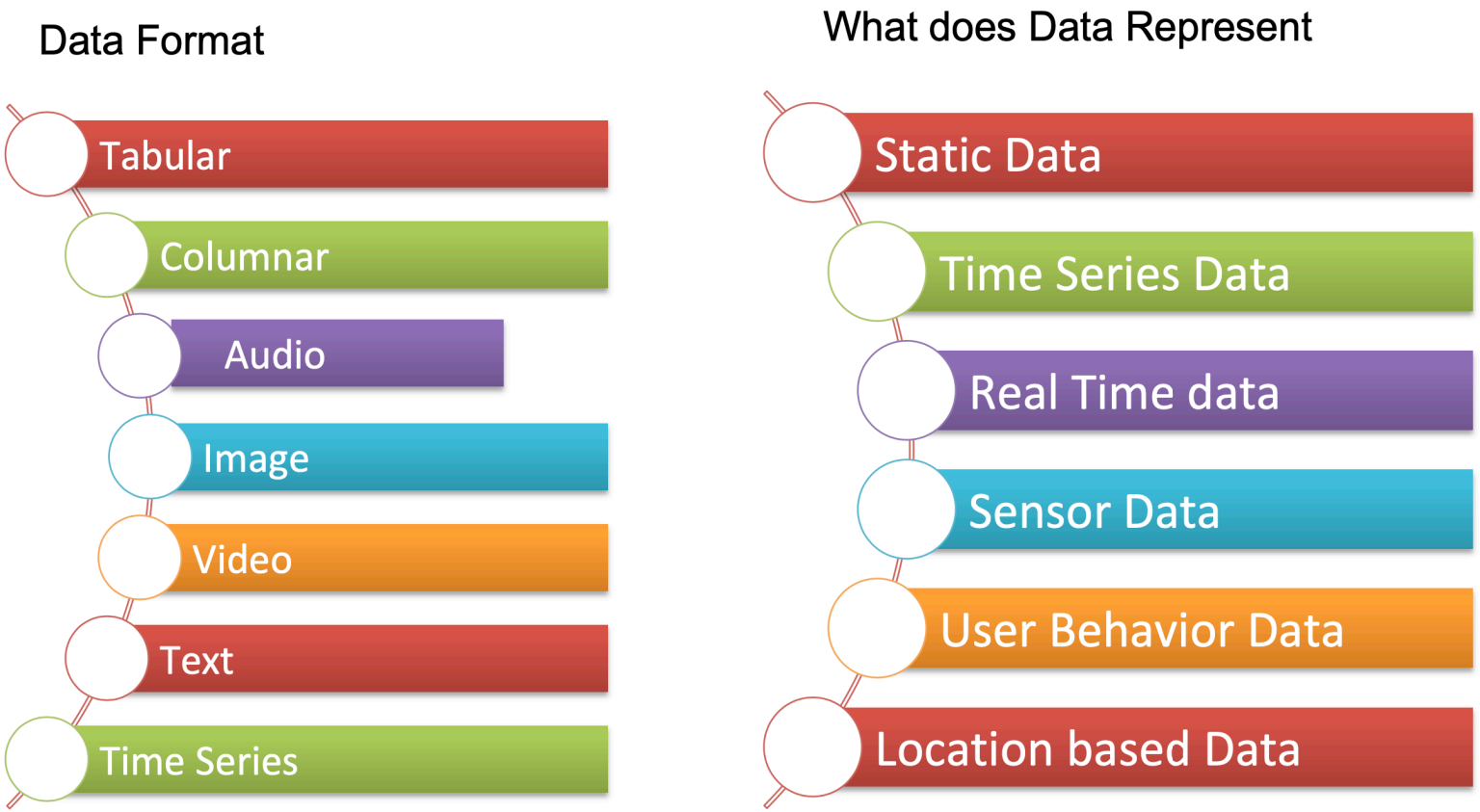
- Collection of data objects and their attributes/variable, characteristic or feature
- An attribute is a property of an object, e.g., height of a person, or temperature of furnace
- A collection of features describe an object. Object is also known as record, observation, sample etc.
- Data is arranged in a table. It simply arranges data in a convenient form

Row ID	Order ID	Order Date	Ship Date	Ship Mod	Customer	Customer	Segment	Country/Region	City	State	Postal Co	Region	Product ID	Category	Sub-Categ	Product N	Sales	Quantity	Discount	Profit
1	CA-2018-152156	11/8/18	11/11/18	Second Cl	CG-12520	Claire Gut	Consumer	United States	Henderson	Kentucky	42420	South	FUR-BO-	Furniture	Bookcases	Bush Som	261.96	2	0	41.9136
2	CA-2018-152156	11/8/18	11/11/18	Second Cl	CG-12520	Claire Gut	Consumer	United States	Henderson	Kentucky	42420	South	FUR-CH-	Furniture	Chairs	Hon Delux	731.94	3	0	219.582
3	CA-2018-138688	6/12/18	6/16/18	Second Cl	DV-13045	Darrin Van	Corporate	United States	Los Angel	California	90036	West	OFF-LA-1	Office Sup	Labels	Self-Adhes	14.62	2	0	6.8714
4	US-2017-108966	10/11/17	10/18/17	Standard C	SO-20335	Sean O'Do	Consumer	United States	Fort Laud	Florida	33311	South	FUR-TA-1	Furniture	Tables	Bretford C	957.5775	5	0.45	-383.031
5	US-2017-108966	10/11/17	10/18/17	Standard C	SO-20335	Sean O'Do	Consumer	United States	Fort Laud	Florida	33311	South	OFF-ST-1	Office Sup	Storage	Eldon Fol	22.368	2	0.2	2.5164
6	CA-2016-115812	6/9/16	6/14/16	Standard C	BH-11710	Brosina H	Consumer	United States	Los Angel	California	90032	West	FUR-FU-1	Furniture	Furnishing	Eldon Exp	48.86	7	0	14.1694
7	CA-2016-115812	6/9/16	6/14/16	Standard C	BH-11710	Brosina H	Consumer	United States	Los Angel	California	90032	West	OFF-AR-1	Office Sup	Art	Newell 32	7.28	4	0	1.9656
8	CA-2016-115812	6/9/16	6/14/16	Standard C	BH-11710	Brosina H	Consumer	United States	Los Angel	California	90032	West	TEC-PH-1	Technolog	Phones	Mitel 532	907.152	6	0.2	90.7152
9	CA-2016-115812	6/9/16	6/14/16	Standard C	BH-11710	Brosina H	Consumer	United States	Los Angel	California	90032	West	OFF-BI-1	Office Sup	Binders	DXL Angl	18.504	3	0.2	5.7825
10	CA-2016-115812	6/9/16	6/14/16	Standard C	BH-11710	Brosina H	Consumer	United States	Los Angel	California	90032	West	OFF-AP-1	Office Sup	Appliances	Belkin F5	114.9	5	0	34.47
11	CA-2016-115812	6/9/16	6/14/16	Standard C	BH-11710	Brosina H	Consumer	United States	Los Angel	California	90032	West	FUR-TA-1	Furniture	Tables	Chromcraf	1706.184	9	0.2	85.3092
12	CA-2016-115812	6/9/16	6/14/16	Standard C	BH-11710	Brosina H	Consumer	United States	Los Angel	California	90032	West	TEC-PH-1	Technolog	Phones	Konftel 25	911.424	4	0.2	68.3568
13	CA-2019-114412	4/15/19	4/20/19	Standard C	AA-10480	Andrew A	Consumer	United States	Concord	North Car	28027	South	OFF-PA-1	Office Sup	Paper	Xerox 196	15.552	3	0.2	5.4432
14	CA-2018-161389	12/5/18	12/10/18	Standard C	IM-15070	Irene Mad	Consumer	United States	Seattle	Washingto	98103	West	OFF-BI-1	Office Sup	Binders	Fellowes I	407.976	3	0.2	132.5922
15	US-2017-118983	11/22/17	11/26/17	Standard C	HP-14815	Harold Pa	Home Offi	United States	Fort Wortl	Texas	76106	Central	OFF-AP-1	Office Sup	Appliances	Holmes R	68.81	5	0.8	-123.858
16	US-2017-118983	11/22/17	11/26/17	Standard C	HP-14815	Harold Pa	Home Offi	United States	Fort Wortl	Texas	76106	Central	OFF-BI-1	Office Sup	Binders	Storex Du	2.544	3	0.8	-3.816
17	CA-2016-105893	11/11/16	11/18/16	Standard C	PK-19075	Pete Kriz	Consumer	United States	Madison	Wisconsin	53711	Central	OFF-ST-1	Office Sup	Storage	Stur-D-Sto	665.88	6	0	13.3176

Data Representation

- Data has form: $\{(x_1,y_1),...,(x_n,y_n)\}$ (labeled), or $\{x_1,...,x_n\}$ (unlabeled)
- What the label y looks like is task-specific
- What about x which denotes a real-world object (e.g., image or text document)?
- Each example x is a set of (numeric) features/attributes/dimensions
- Features encode properties of the object which x represents
- x is commonly represented as a D × 1 vector
- Representing a 28 × 28 image: x can be a 784 × 1 vector of pixel values
- Representing a text document: x can be a vector of word-counts of words appearing in that document

Types of Data sets



Types of Variables

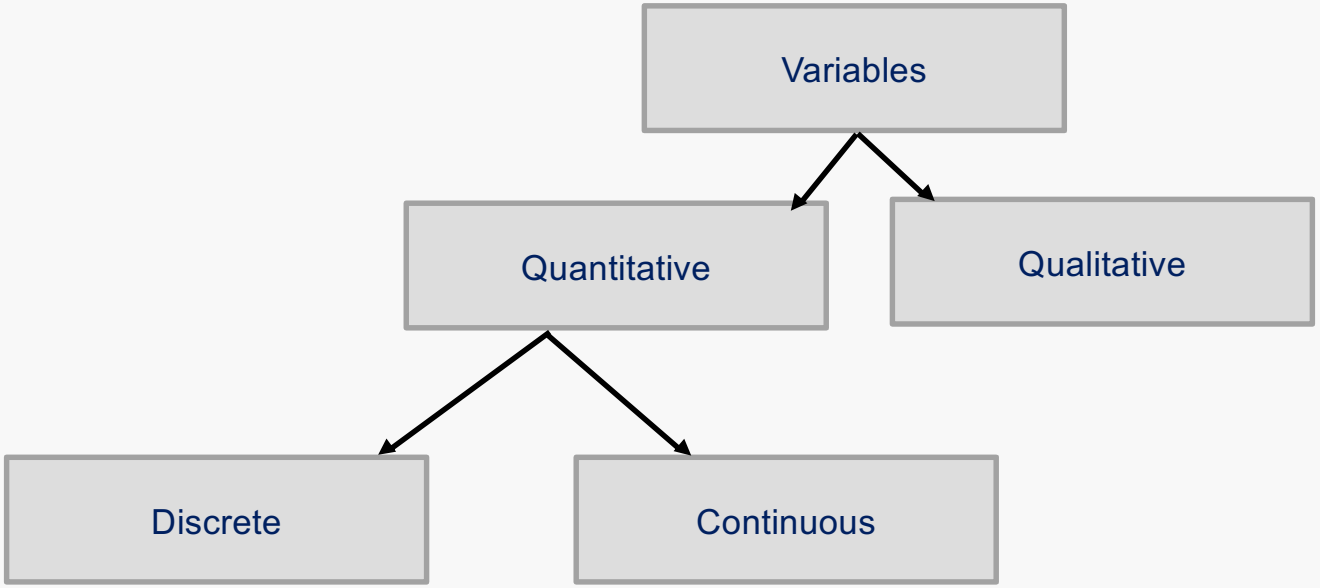
Qualitative variable: When a variable can be placed into well-defined groups or categories, that does not depend on order. It can take only specific value, e.g., color of a car or gender

Nominal: describe a variable with limited number of different values that can not be ordered. (ID number, color, industry type (financial, engineering, retail)), Gender (Male, Female)

Ordinal: a variable whose value can be ordered or ranked (height: {tall, med, short})

Types of Variables

Quantitative variable: when a variable represents a count or quantity. It can take any value between a given minimum and maximum, e.g., temperature.



Preparing Data

- Preparing the data is one of the most time-consuming parts of data analysis.
- The way in which the data is collected and prepared is critical to the confidence with which decisions can be made.
- The data needs to be merged into a table and this may involve integration of the data from multiple sources.
- Once the data is in a tabular format, it should be fully characterized.
- The data should be cleaned by resolving ambiguities and errors, removing redundant and problematic data, and eliminating columns of data irrelevant to the analysis.
- This whole process is called *pre-processing/Data preparation*

Measures of Data Quality

• Raw data is often not useful without some kind of organization or manipulation. Raw data seems to be just a bunch of meaningless values without any context or some level of organization.

•In recent years, data quality has gained more an more attention due to extended use of data warehouse systems and a higher relevance of customer relationship management.

•Due to this fact for decision makers, the benefits of data depends heavily on their completeness, correctness, and timeliness, respectively. Such properties are known as data quality dimensions.

•The consequences of poor data quality are manifold: They range from worsening customer relationships and customer satisfaction by falsely addressing customers to insufficient decision support for managers.

•Organizations select the data quality dimensions and associated dimension threshold based on their business context, requirements, level of risks etc.

Measures of Data Quality

The following measure can be used to test the quality of data –

– ***Completeness****: The proportion of stored data against the potential of "100% complete"*

– ***Uniqueness****: No observation will be recorded more than once based upon how that observation is identified.*

– ***Velocity****: The rate at which data is coming especially for streaming data*

– ***Accuracy****: The degree to which data correctly describes the "real world" event being described*

– ***Consistency****: The absence of difference, when comparing two or more representations of a thing against a definition.*

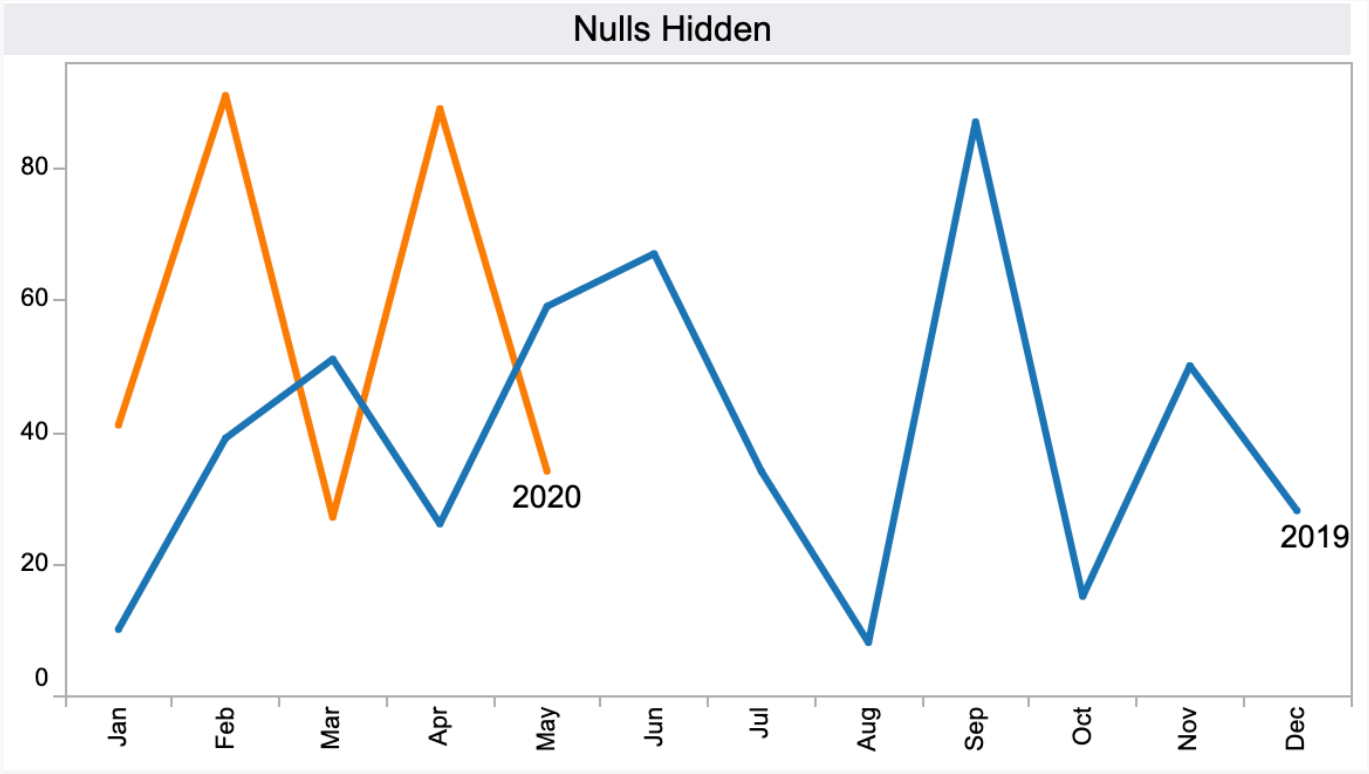
– ***Accessibility****: How easy it is to access the data.*

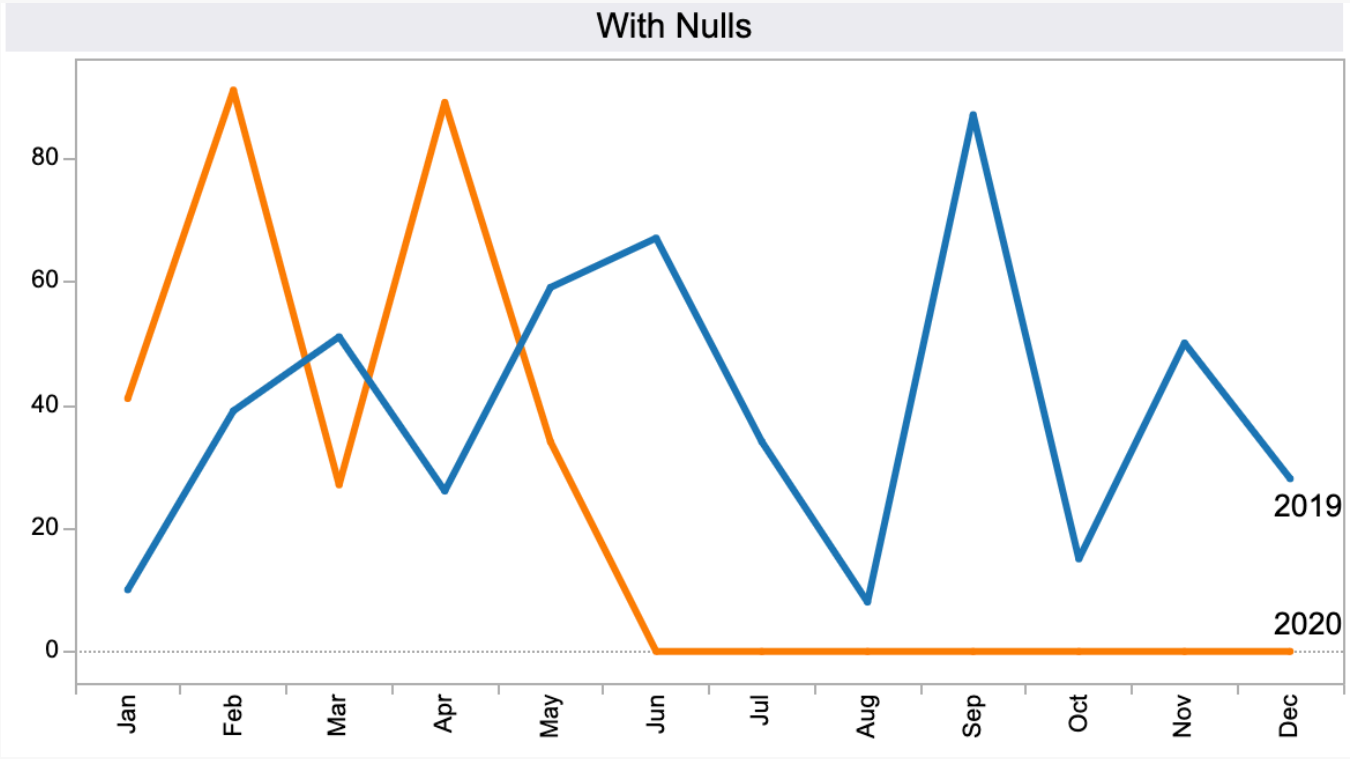
Data Quality

What kind of data quality problems?

• How can we detect problems with the data? • What can we do about these problems?

• Examples of data quality problems:
  - Missing Values
  - Noise and Outliers
  - Duplicate values
  - Inconsistent Dates
  - Impossible values (Negative Sales)
  - check using if condition


Nulls Hidden
80
60
40
20
0
2020
2019
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec



If you want the quality of the data to be visible for cleansing the data, display the noise & outliers

Preparing Data

Below are the steps involved to understand, clean and prepare the data for building the predictive model:

1. Variable Identification
2. Univariate Analysis
3. Bi-variate Analysis
4. Missing values treatment
5. Outlier treatment
6. Variable transformation
7. Variable creation

Finally, we will need to iterate over steps 4 – 7 multiple times before we come up with our refined model.

Univariate analysis is perhaps the simplest form of statistical analysis. Like other forms of statistics, it can be inferential or descriptive. The key fact is that only one variable is involved. Univariate analysis can yield misleading results in cases in which multivariate analysis is more appropriate.

- Age
- Height
- Univariate would not look at the above 2 variables at the same time

Bivariate analysis is one of the simplest forms of quantitative analysis. It involves the analysis of two variables, for the purpose of determining the empirical relationship between them. Bivariate analysis can be helpful in testing simple hypotheses of association.

- Ice cream sales compared to the temperature that day.
- Traffic accidents along with the weather on a particular day.

Univariate Analysis

Data : Credit Default Data

Variables:
Credit Amount (Continuous)
Deposit (Categorical)

Import Dataset
library(psych)
describe(Credit_Default). - Mean, Median, Mode, min, max, se – Standard error)
> boxplot(default_of_credit_card_clients$X12,main = "boxplot", ylab = "Credit Amount", col = 5)

- plot(default_of_credit_card_clients$X12, col = 3). - This is Scatterplot
- hist(CREDIT_DEFAULT$BILL_AMT1, col = 7)
- barplot(deposit, main = "Simple Bar plot", xlab = "Bill Amount", ylab = "Frequency")

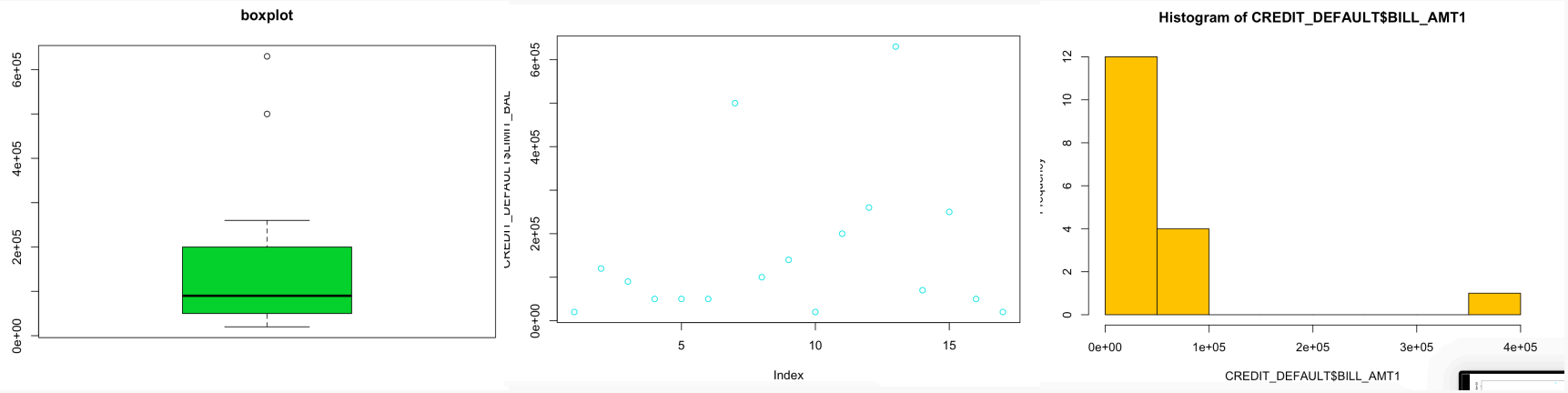
```
if(!require(psych)){install.packages("psych")}
if(!require(DescTools)){install.packages("DescTools")}
if(!require(Rmisc)){install.packages("Rmisc")}
if(!require(FSA)){install.packages("FSA")}
if(!require(plyr)){install.packages("plyr")}
if(!require(boot)){install.packages("boot")}
describe(filename)
```

Univariate Analysis:

Credit Default:

Variable : Credit Amount

Analysts use Box Plot, Scatter Plot & Histogram to visualize Continuous Variable



Bivariate Analysis

Taking the time to study the data you have will help you in ways that are less obvious. You build an intuition for the data and for the entities that individual records or observations represent. These can bias you towards specific techniques (for better or worse), but you can also be inspired.
For example, examine your data in detail may trigger ideas for specific techniques to investigate

Data Cleaning. You may discover missing or corrupt data and think of various data cleaning operations to perform such as marking or removing bad data and imputing missing data.

Data Transforms. You may discover that some attributes have familiar distributions such as Gaussian or exponential giving you ideas of scaling or log or other transforms you could apply

Data Modeling. You may notice properties of the data such as distributions or data types that suggest the use (or to not use) specific machine learning algorithms.

Bivariate Analysis

mydata<- read_excel("Downloads/CAR_DATA.xlsx")

- boxplot(mydata$mileage ~ mydata$transmission)
- ggplot2.boxplot(data = mydata, xname = "transmission", yname = "mileage")

Multivariate Analysis

```
wine <- read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data",
        sep=",")

wine

library("car")

wine[2:6]

scatterplotMatrix(wine[2:6])


plot(wine$V4, wine$V5)

text(wine$V4, wine$V5, wine$V1, cex=0.7, pos=4, col="red")
```

Preparing data also prepares the scientist so that when using prepared data the scientist produces better models, and faster.

- Good data is a prerequisite for producing effective models of any type.
- Several data mining methods are sensitive to the scale and/or type of the variables
- Different variables (columns of our data sets) may have rather different scales
- Some methods are not able to handle either nominal or numeric Variables
- We may need to “create” new variables to achieve our objectives

Sometimes we are more interested in relative values (variations) than absolute values

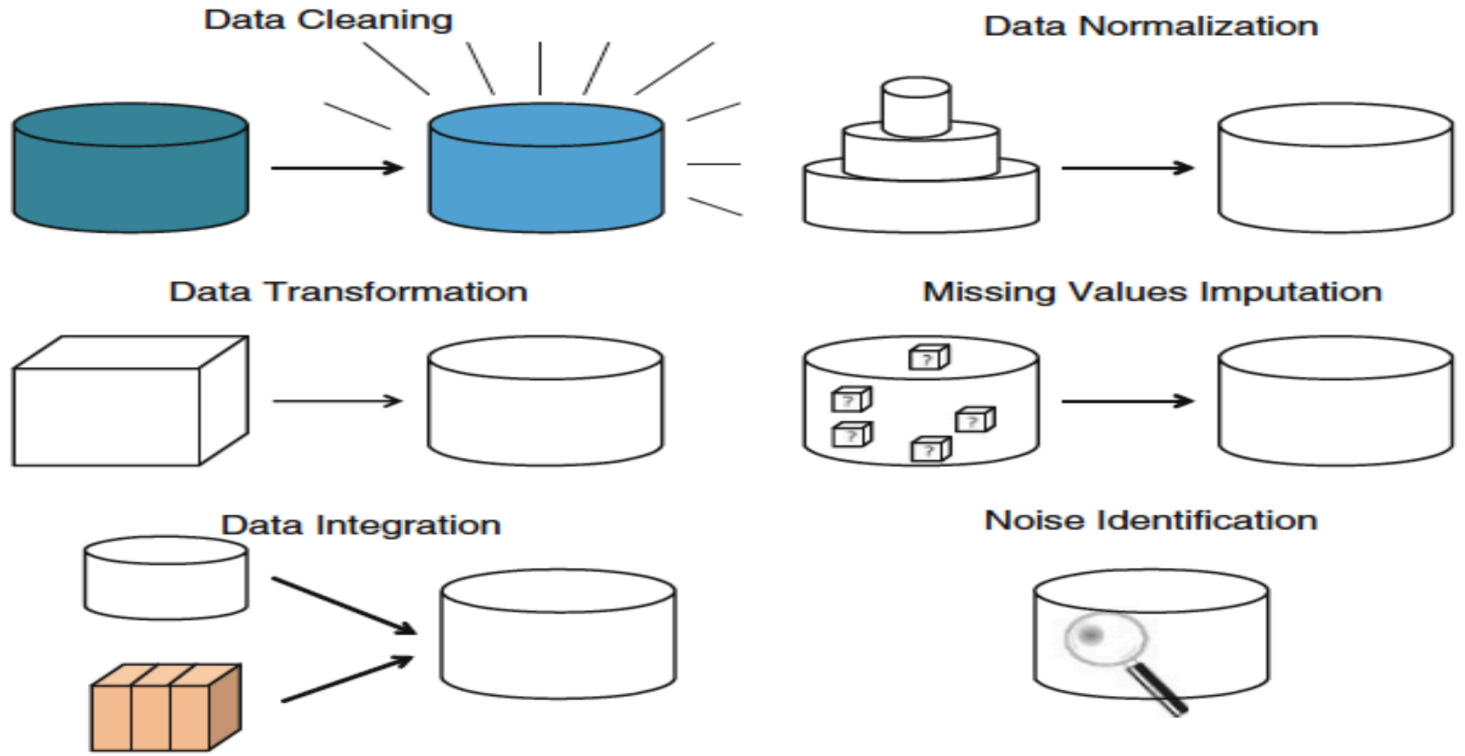
- We may be aware of some domain-specific mathematical relationship among two or more variables that is important for the task
- Frequently we have data sets with unknown variable values
- Our data set may be too large for some methods to be applicable

Major Tasks involved in Data Analysis

- **Integration of data**
  – Integration of data from multiple databases, or files
- **Data discretization (for numerical data)**
- **Data Cleaning**
  – Fill in missing values, smooth noisy data, remove outliers and resolve inconsistencies
- **Data Transformation**
  – Scaling/normalization and aggregation. Normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements.
- **Data reduction**
  – Optimize the features/attributes and obtain reduced representation in volume.

Pre-Processing

Forms of data preparation



•These techniques are not mutually exclusive; they may work together. For example, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a date field to a common format.

• Data cleaning techniques, when applied before mining, can substantially improve the overall quality of the modeling.

• Data cleaning routines work to “clean” the data by filling in missing values, smoothing noisy data, identifying o removing outliers, and resolving inconsistencies.

• If the user believe that the data are dirty, they are unlikely to trust the results of any data mining that has been applied to it. Furthermore, dirty data can cause confusion for the mining procedure, resulting in unreliable output.

Data Integration

Data integration is the process of combining data from multiple sources. These sources may include multiple databases, data cubes or flat files

- The data may also need to be transformed into forms appropriate for mining.

- Integrate metadata from different sources

- Removing duplicates and redundant data

- Detect and resolve data value conflicts

– For the same real world entity, attribute values from different sources are different, e.g., different scales, different units (miles vs. km)

Missing Values

•It might happen that dataset is not complete, and when information is not available we call it missing values. In real world missing data is ubiquitous. Missing data can reduce the accuracy of analysis or can lead to a biased model because we have not analyzed the behavior and relationship with other variables correctly and it can lead to wrong predictions.

• The impact of missing data is a subject that most of us want to avoid. In R the missing values are coded by the symbol NA.

Missing Values

Missing data in the training data set can reduce the accuracy/fit of a model or can lead to a biased model because we have not analyzed the behavior and relationship with other variables correctly and it can lead to wrong prediction or classification.

- Data is not always available

- e.g., many records have not values for attribute, such as customer age or income in sales data

Missing Values

Reasons for missing values
– Information is not collected
(e.g., people decline to give their age and weight)
– Attributes may not be applicable to all cases (e.g., annual income is not applicable to children)
– NA, Inf, NaN, NULL
- Equipment malfunction
- Data not entered properly
- Certain data may not be considered important at the time of data entry or collection
- Deleted due to inconsistent

Most functions in R handle missing data appropriately by default, but a couple of basic functions require care when missing data are present and it's always a good idea to check for missing data in a data set.

Handling Missing Values

Ignore the Missing Value During Analysis: this is usually done when class label is missing
(assuming the mining task involves classification). This method is not very effective, unless the record contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably.

Fill in the missing value manually: In general, this approach is time-consuming and may not be feasible given a large data set with many missing values

Use a global constant/mean or median to fill in the missing value: Replace all missing feature values by the same constant/mean or median of the attribute

Use the most probable value to fill in the missing value: This may be determined with regression, inference-based tools using a Bayesian formalism, or decision tree induction.

Handling Missing Values

Prediction Model: Prediction model is one of the sophisticated method for handling missing data. Here, we create a predictive model to estimate values that will substitute the missing data. In this case, we divide our data set into two sets: One set with no missing values for the variable and another one with missing values. First data set become training data set of the model while second data set with missing values is test data set and variable with missing values is treated as target variable. Next, we create a model to predict target variable based on other attributes of the training data set and populate missing values of test data set. We can use regression, ANOVA, Logistic regression and various modeling technique to perform this. There are 2 drawbacks for this approach:

- The model estimated values are usually more well-behaved than the true values
- If there are no relationships with attributes in the data set and the attribute with missing values, then the model will not be precise for estimating missing values.

How to see the missings in the dataset with is.na() function

```
> x <- c(45, 53, NA)
> is.na(x)
[1] FALSE FALSE TRUE
> which(is.na(x))
[1] 3
```

You can find the sum and percentage of missing in your dataset > x <- c(45, 53, NA)

```
> is.na(x)
[1] FALSE FALSE TRUE
> sum(is.na(x))
[1] 1
> mean(is.na(x))
[1] 0.3333333
```

Another useful function in R to deal with missing values is na.omit() which delete incomplete observations.

```
>na.omit(x)
```

To find the number of non-missing observations >length(na.omit(x))

```
>length(na.omit(x))
[1] 2
```

Some functions also have options to deal with missing data. For example, the mean() function has the 'na.rm = TRUE' option to remove missing values from the calculation. So another way to calculate the mean of non-missing values for a variable

```
> mean(x, na.rm = TRUE)
[1] 49
```

Handling Missing Values

```
iris_sample = iris[1:10,1:4]
 iris_sample
iris_sample[10,1]<-NA
iris_sample
 iris_sample_rm<-na.omit(iris_sample)
 library(e1071)
iris_sample_fix <-impute(iris_sample, what = 'mean’)
 iris_sample_fix
iris_sample_fix <-impute(iris_sample, what = ’median')
```

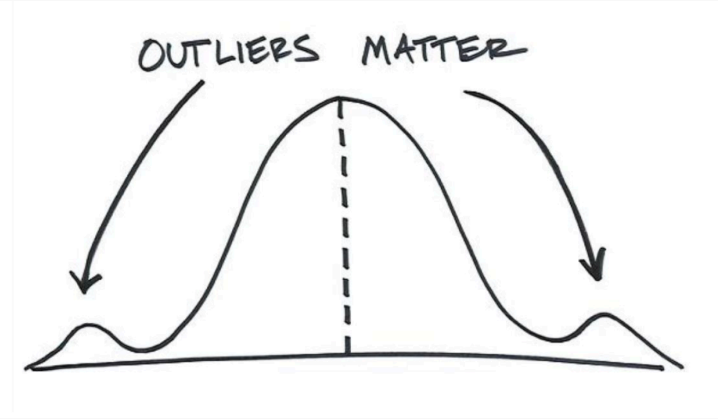
Polyreg – Multinomial Logistic Regression
Imputation - In statistics, imputation is the process of replacing missing data with substituted values. When substituting for a data point, it is known as "unit imputation"; when substituting for a component of a data point, it is known as "item imputation”
Pmm – Predictive Mean Matching

mice (multivariate imputation by chained equations) package.

Outliers

An **outlier** is a value or an **observation that is distant from other observations.**

Outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in an outlier is an observation strikingly far from some central value. It is an unusual value relative to the bulk of the data. Commonly computed quantities such as averages and least squares lines can be drastically affected by such values. Methods to detect outliers and to moderate their effects are needed.



Outliers

What is the impact of Outliers on a dataset?
Outliers can drastically change the results of the data analysis and statistical modeling. There are numerous unfavorable impacts of outliers in the data set:

- It increases the error variance and reduces the power of statistical tests
- If the outliers are non-randomly distributed, they can decrease normality
- They can bias or influence estimates that may be of substantive interest
- They can also impact the basic assumption of Regression, ANOVA and other
•statistical model assumptions.

Without Outlier	With Outlier
4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7	4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7,300
Mean = 5.45	Mean = 30.00
Median = 5.00	Median = 5.50
Mode = 5.00	Mode = 5.00
Standard Deviation = 1.04	Standard Deviation = 85.03

Outliers

How to detect Outliers?
Most commonly used method to detect outliers is visualization. We use various visualization methods, like Box-plot, Histogram, Scatter Plot. Some analysts also use various thumb rules to detect outliers. Some of them are:

- Any value, which is beyond the range of -1.5 x IQR to 1.5 x IQR

- Use capping methods. Any value which out of range of 5th and 95th percentile can be considered as outlier
- Data points, three or more standard deviation away from mean are considered outlier

- Outlier detection is merely a special case of the examination of data for influential data points and it also depends on the business understanding

How to handle Outliers

Clustering
• Detect and remove outliers

• Outliers may be detected by clustering, where similar values are organized into groups, or “clusters”, Intuitively, values that fall outside of the set of clusters may be considered outliers.

• There are many choices of clustering definitions and clustering algorithms.
• Combined computer and human inspection • Tedious and time consuming

Outlier Treatment

OS.XLSX
Outlier has proportion <0.10

```
library(readxl)
> OS <- read_excel("Downloads/OS.xlsx")
> View(OS)
> OS.hi <- subset(OS, Proportion > 0.1)
```

Duplicate Data

•Data set may include data objects that are duplicates, or almost duplicates of one another

• Major issue when merging data from heterogeneous sources

• Examples:

Same person with multiple email addresses
• Data cleaning

Process of dealing with duplicate data issues
A very useful application of subsetting data is to find and remove duplicate values. R has a useful function, duplicated(), that finds duplicate values and returns a logical vector that tells you whether the specific value is a duplicate. This means that for duplicated values, duplicated() returns FALSE for the first occurrence and TRUE for every following occurrence of that value

```
> duplicated(c(1,2,3,4,1,5,1,3))
[1] FALSE FALSE FALSE FALSE TRUE FALSE TRUE TRUE
```

Extract duplicated elements:
> x[duplicated(x)]
[1] 1 1 3

• If you want to remove duplicated elements, use !duplicated(), where ! Is a logical negation:

> x[!duplicated(x)]
>[1] 1 2 3 4 5

The function distinct() in dplyr package can be used to keep only unique/distinct rows from a data frame. If there are duplicate rows, only the first row is preserved. It's an efficient version of the R base unique() function.

- mydata<-distinct(iris)
- > dim(mydata)
  [1] 149 5

Remove duplicate rows based on certain columns (variables): #
remove duplicated rows based on Sepal. Length
>distinct(iris, Sepal.Length)

remove duplicated rows based on Sepal. Length and Petal.Width
>distinct(iris, Sepal.Length, Petal.Width)

Sorting

The function **order**() returns a permutation vector which will reorder a vector so that it becomes sorted. This is especially useful if you wish to sort a matrix by the values in one of it's rows:

```
mat = matrix(c(12,3,7,9,10,11),ncol=2)
 > mat[order(mat[,1]),]
[,1] [,2] [1,] 3 10 [2,] 7 11 [3,] 12 9
```

Merging Matrices and Data Frames

Many times two matrices or data frames need to be combined by matching values in a column in one to values in a column of the other. Consider the following two data frames:

```
> df1 <- data.frame(a = c(1,7,9,3), b = c(12,18,24,9))
> df2 <- data.frame(a = c(1,7,19,31), b = c(112,118,214,91))
> df1 ab
1 1 12 2 7 18 3 9 24 439
> df2
ab
1 1112 2 7118 3 19 214 4 31 91
```

Merging Matrices and Data Frames

We wish to combine observations with common values of a in the
two data frames (like a join in SQL) we use merge(0 function. The merge() function makes this simple:

```
> merge(df1,df2,by="a") a b.x b.y
11 12112
27 18118
```

By default, only the rows that matched are included in the output. Using the all=TRUE argument will display all the rows, adding NAs where appropriate.

```
> merge(df1,df2,by="a", all = TRUE)
a b.x b.y
1 1 12112 2 3 9 NA
3 7 18118 4 9 24 NA 5 19 NA 214 631 NA 91
```

Merging Datasets

Merge () # R equivalent to SQL JOIN
merge(df1, df2, by = “common Column”)

```
##Let's create a data frame with state populations
statepop <- data.frame(State = c("CA", "AZ", "OR", "WA","MT"), pop = c(38, 6.6, 3.9, 6.9, 1.0))
#Let's create a data frame with state Areas (4 states)
areas_western <- data.frame(State = c("CA", "OR", "WA", "NV"), SqMilesTh = c(163, 98, 71,
110))
```

```
merge(statepop,areas_western)
 merge(statepop, areas_western, all= TRUE)
merge(statepop, areas_western, all.x = TRUE)
```

Aggregate()

The aggregate() function presents a summary of a scalar valued statistic, broken down by one or more groups. While similar operations can be performed using tapply(), aggregate() presents the results in a data frame instead of a table.

```
> testdata = data.frame(one=c(1,1,1,2,2,2),two=c(1,2,3,1,2,3),three=rnorm(6))

> aggregate(testdata$three,list(testdata$one,testdata$two),mean)
Group.1 Group.2 x
1.1  1
2.2  2
3.3  1
4.4  2
5.5  1
6 2
1 0.4503872 1 0.6673121 2 0.4901954 2 -1.4188940 3 0.2561912
3 1.4463768
tapply(testdata$three,list(testdata$one,testdata$two),mean)
123
1 0.4503872 0.4901954 0.2561912 2 0.6673121 -1.4188940 1.4463768
```

Aggregate() Continued..

If the first argument to aggregate() is a matrix or multicolumn data frame, the statistic will be calculated separately for each column.

>testdata = cbind(sample(1:5,size=100,replace=TRUE), matrix(rnorm(500),100,5))

> dimnames(testdata) = list(NULL,c("grp","A","B","C","D","E"))

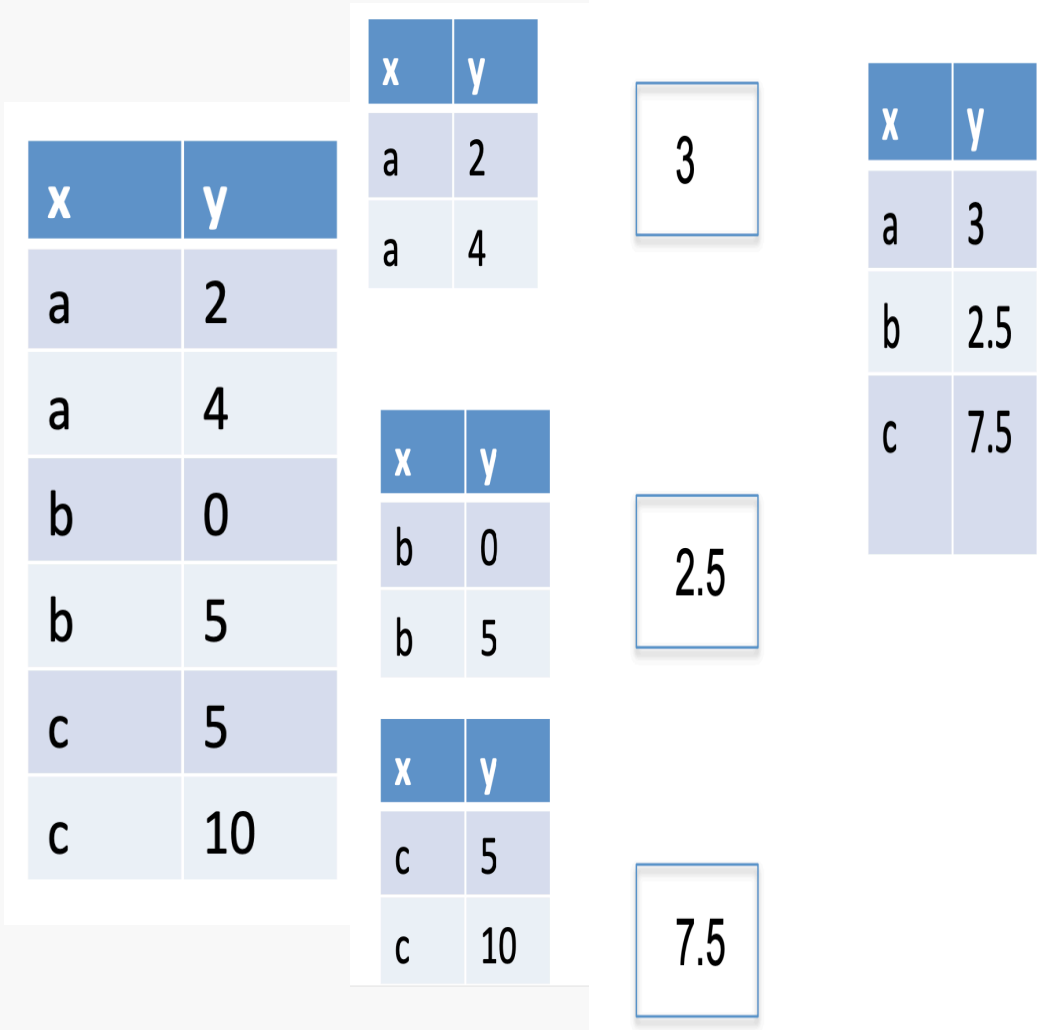
- aggregate(testdata[,-1],list(grp=testdata[,1]),min)
- grp A B C D E

1.1 1 -2.081877 -1.654819 -1.949135 -1.428763 -2.309056
2.2 2 -2.907306 -2.371320 -1.408994 -1.435984 -2.236848
3.3 3 -1.985370 -1.826547 -2.405930 -2.125746 -1.770902
4.4 4 -1.544151 -2.032521 -2.046227 -1.714147 -2.276399
5 5 -1.381750 -1.359896 -1.926122 -2.679803 -2.082310

Note that the data frame returned by aggregate() has the appropriate column names, and that grouping variable names can be supplied with the list of grouping variables.

dplyr

- SAC – Split – Apply – Combine
- Split up a big dataset
- Apply a function to each piece
- Combine all pieces back together



The data frame is a key data structure in R.

Working with large and complex sets of data is a day-to-day reality. The package dplyr provides a well structured set of functions for manipulating such data collections and performing typical operations with standard syntax that makes them easier to remember.

Great for data exploration and transformation It is very fast, even with large collections.
The functions work with connections to databases as well as data frames.

dplyr is built on plyr and incorporates features of Data.table. In other words, it is an optimize and distilled version of plyr package.

It does not provide any “new” functionality per se, but greatly simplifies existing functionality in R

Much work with data involves subsetting, defining new columns, sorting or otherwise manipulating the data. dplyr has five functions (verbs) for such actions, that start with a data frame and produce another one.

select: return a subset of columns of a data frame

filter: extract a subset of rows from a data frame based on logical conditions

arrange: reorder rows of a data frame

mutate: add new columns/variables or transform existing variables

summarise/summarize: generate summary statistics of different variables in the data frame.

Joins: inner join, semi-join, anti-join

dplyr Usage

The first argument is a data frame

The subsequent arguments decsribe what to do with it, and you can refer to columns in the data frame directly without using the $ operator

The result is a new data frame

Data frames must be properly formatted and annotated for this to all be useful

Load the dplyr package
library(dplyr)
> library(hflights) > data(hflights)

```
> head(hflights)
Year Month
DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum TailNum
5424 2011 5425 2011 5426 2011 5427 2011 5428 2011 5429 2011
1 1 1 1 1 1
1 6 2 7 3 1 4 2 5 3 6 4
1400 1500 AA 1401 1501 AA 1352 1502 AA 1403 1513 AA 1405 1507 AA 1359 1503 AA
428 N576AA 428 N557AA 428 N541AA 428 N403AA 428 N492AA 428 N262AA
ActualElapsedTime
AirTime ArrDelay DepDelay Origin Dest Distance TaxiIn TaxiOut
5424.5424  60
5425.5425  60
5426.5426  70
5427.5427  70
5428.5428  62
5429.5429  64
40 -10 0 IAH DFW
224 7 13 224 6 9
224 5 17 224 9 22
224 9 9 224 6 13
45 -9 1 48 -8 -8 39 33 44 -3 5 45 -7 -1
IAH DFW IAH DFW
IAH DFW IAH DFW IAH DFW
Cancelled CancellationCode Diverted 54240 0
54250 0
54260 0
54270 0 54280 0 54290 0
```

>select: pick columns by name

Base R approach

>hflights[,c(“AirTime”, “ArrDelay”, “UniqueCarrier”)]

dplyr approach# dplyr App

>x<-select(hflights, ArrTime, ArrDelay, UniqueCarrier) > > head(x)

ArrTime ArrDelay UniqueCarrier

5424.5424 1500 -10 AA

5425.5425 1501 -9 AA

5426.5426 1502 -8 AA

54271513 3 AA 5428 1507 -3 AA 5429 1503 -7 AA

```
>f_df_select<-dplyr::select(hflights, Year:DayOfWeek, TailNum, ActualElapsedTime)
```

```
head(f_df_select)
```

	Year	Month	DayofMonth	DayOfWeek	TailNum	ActualElapsedTime
5424	2011	1	1	6	N576AA	60
5425	2011	1	2	7	N557AA	60
5426	2011	1	3	1	N541AA	70
5427	2011	1	4	2	N403AA	70
5428	2011	1	5	3	N492AA	62
5429	2011	1	6	4	N262AA	64

>filter: keeps rows matching criteria

Base R approach to filtering forces you to repeat the data frame's name dplyr approach is simpler to write and read

```
# Base R approach
>hflights[hflights$UniqueCarrier == "AA" & hflights$DayofMonth ==1,]

# dplyr approach
>filter(hflights, Month==1, DayofMonth==1)
# use pipe for OR condition
>filter(hflights, UniqueCarrier == "AA" | UniqueCarrier =="UA")
```

>arrange: Reorder rows

```
# Base R approach
>hflights[order(hflights$DepDelay), c( “Month”, “UniqueCarrier”)]
# dplyr approach
f_df_arrange<-select(arrange(hflights, DepDelay), Month, UniqueCarrier)
```

>mutate: Add new variables
Create new variables that are functions of existing variables

```
# Base R approach
>hflights$gain <-hflights$ArrDelay-hflights$DepDelay

 >hflights$gain_per_hour = hflights$gain/60]
# dplyr approach

f_df_mutate<-mutate(hflights, gain = ArrDelay - DepDelay,
gain_per_hour = gain/60) head(f_df_mutate)
```

>summarize/summarise: Reduces variables to value
Primarily useful with data that has been grouped by one or more variable group_by
creates the group that will be operated on
summarise uses the provided aggregation function to summarise each group

```
# group the data into daily flights >Dest<-group_by(hflights, Dest)
# group the data into daily flights
>daily <- group_by(hflights, Year, Month, DayofMonth)
# to get the number of flights per day
per_day <- summarize(daily, number_flights = n())
head(per_day)

# We have access to each of the grouping variables.
# Notice that in the summary data.frame,
# we have Year and Month as grouping variables.
# We can get the number of flights per month by summarizing as follows per_month <-
summarize(per_day, number_flights = sum(number_flights)) head(per_month)
```

```
#  Chaining #
#  There is a nice way to pass the result of one function to another.
#  This is possible because so many dplyr functions take a data
#  table as input and output another data table.
#  For example:

a1 <- group_by(hflights, Year, Month, DayofMonth)
a2 <- select(a1, Year:DayofMonth, ArrDelay, DepDelay)
a3 <- summarise(a2, arr = mean(ArrDelay, na.rm = TRUE), dep = mean(DepDelay,na.rm =
TRUE))
a4 <- filter(a3, arr > 30 | dep > 30)
a4<-hflights %>% group_by(Year, Month, DayofMonth) %>%
select(Year:DayofMonth, ArrDelay, DepDelay) %>%
summarise(arr = mean(ArrDelay, na.rm = TRUE), dep = mean(DepDelay, na.rm = TRUE)) %>%
filter(arr > 30 | dep > 30)
```

Rounding Functions

The following functions are available for rounding numerical values:

- round() - uses IEEE standard to round up or down; optional digits= argument controls precision

- signif() - rounds numerical values to the specified digits= number of significant digits

- trunc() - rounds by removing non-integer part of number

- floor(), ceiling() - rounds to integers not greater or not less than their arguments, respectively

- zapsmall() - accepts a vector or array, and makes numbers close to zero (compared to others in the input) zero. digits= argument controls the rounding.

Summary

Data preparation is a big issue for data mining

- Data preparation includes
  - Data cleaning and data integration
  - Data reduction and feature selection
  - Discretization
- Many methods have been proposed but still an active area of research

Summarize Apply functions

We have discussed about loops. We introduced vectorized function. We will discuss some of the most used vectorized functions: the **apply** functions.

The apply family pertains to the R base package and is populated with functions to manipulate slices of data from matrices, arrays, lists and data frames in a repetitive way. These functions avoid explicit use of loop.

They act on an input list, matrix or array and apply a named function with one or several optional arguments. The called function could be:

- An aggregation function, like for example the mean, or the sum
- Other transforming or sub-setting functions
- And other vectorized functions, which return more complex

strictures like list, vectors, matrices and arrays

The apply functions help to perform operations with very few lines of code. The family comprises:

apply: Apply functions over array margins
by: Apply a function to a data frame split by factors

lapply: Apply a function over list or vector

mapply: Apply a function to multiple list or vector arguments

rapply: Recursively apply a function to a list

tapply: Apply a function over a ragged array

apply: returns a vector or array or list of values obtained by applying a function to margins of an array or matrix

```
x<-matrix(c(11:30), nrow = 10, ncol =2)

> # means of the rows
> apply(x,1,mean)
[1] 16 17 18 19 20 21 22 23 24 25

> # means of the columns
> apply(x,2,mean)
[1] 15.5 25.5
```

```
> apply(x,1:2, function(x) x/2)
      [,1] [,2]
[1,] 5.5 10.5
[2,] 6.0 11.0
[3,] 6.5 11.5
[4,] 7.0 12.0
[5,] 7.5 12.5
[6,] 8.0 13.0
[7,] 8.5 13.5
[8,] 9.0 14.0
[9,] 9.5 14.5
[10,] 10.0 15.0
```

lapply: returns a list of the same length as x, each element of which is the result of applying FUN to the corresponding element of x

```
> lst <- list(x1 = 1:10, x2 = 11:20)
> lapply(lst,mean)
$x1
[1] 5.5
$x2
[1] 15.5
> lapply(lst,sum)
 $x1
[1] 55
$x2
[1] 155
```

sapply: is a user-friendly version of lapply by default returning a vector or matrix if approriate, i.e., if lapply would have returned a list with elements $x1, an d$x2, sapply will return either a vector with elements [[‘x1’]] and [[‘x2’]], or matrix with column names “x1” and “x2”

```
lst <- list(x1 = 1:10, x2 = 11:20)
> lst_mean<-sapply(lst,mean)
> lst_mean
x1 x2
5.5 15.5
> class(lst_mean)
 [1] "numeric"
```

```
data <-mtcars
mpg_category <- function(mpg){
 if (mpg >30) {
    return ("High")
}else if (mpg > 20){
   return ("Medium")
}
return ("Low")
}

lapply(X = data$mpg, FUN = mpg_category)

sapply(X = data$mpg, FUN = mpg_category)
```

```
within_range <- function(mpg, low,high){
   if (mpg >= low & mpg <= high){
    return(TRUE)
   }
    return(FALSE)
}

index <- sapply(X = data$mpg, FUN = within_range, low =
15, high = 20)
index
data[index,]
```

```
mpg_within_std_range <- function(mpg,cyl){
  if ( cyl == 4 ){
  return(within_range(mpg,low = 23, high = 31))
  }
  else if (cyl == 6){
  return(within_range(mpg, low = 18, high = 23))
  }
  return(within_range(mpg, low = 13, high = 18))
}

index <- mapply(FUN = mpg_within_std_range, mpg = data$mpg, cyl = data$cyl)
index
data[!index,]
sapply(data, FUN = median)
```

vapply: is similar to sapply, but has a pre-specified type of return, o it can be safer to use. A third argument if supplied to vapply

```
lst <- list(x1 = 1:10, x2 = 11:20)
> lst.fivenum <- vapply(lst, fivenum, c(Min.=0, "1st Qu."=0, Median=0, "3rd Qu."=0, Max.=0))

> lst.fivenum
x1 x2
Min. 1.0 11.0
1st Qu. 3.0 13.0
Median 5.5 15.5
3rd Qu. 8.0 18.0
Max. 10.0 20.0

> class(lst.fivenum)
 [1] "matrix"
```

mapply: is a multivariate version of sapply. It applies FUN to the first elements of each (...) argument, the second elements, the third elements, and so on.

```
>lst <- list(x1 = 1:10, x2 = 11:20)
> lst2 <- list(x3 = c(21:30), x4 = c(31:40))
> mapply(sum, lst$x1, lst$x2, lst2$x3, lst2$x4)
[1] 64 68 72 76 80 84 88 92 96100
```

tapply: apply a function to each cell.

Mean of Sepal length by species

```
> tapply(iris$Sepal.Length, iris$Species, mean)
 setosa versicolor virginica
5.006      5.936      6.588
```