

2021

Data Analytics lab Manual



Authors:

Dr Mahesh Kaluti, Dr. Nagarathna,

DR. Umesh D R, V. Chethan Kumar,

Bhavya .D, Yoga .B.S

P.E.S. College of Engineering, Mandya



P.E.S College of Engineering Mandya

Department of Computer Science & Engineering



Preface

We are proud to Present this *“Data ANALYTICS Lab Manual”* Generally Comprises the Detailed Laboratory sessions for the third year Batchular of Engineering Undergraduate students of the Computer Science & Engineering stream.

As always we are thankful for the Guidelines and support for those who had involved directly & indirectly for pulling this valume together with their costant support.



P.E.S College of Engineering Mandya 571401

Department of Computer Science & Engineering



Data Analytics Lab Manual

Sl. No	Experiment Name												
1	Demonstrate the Negative (-ve) and Positive (+ve) Correlation between two attributes of Women data set & mtcars dataset												
2	Create box plot for the two Variables group of LungCapData dataset. having 6 variables each signifying lung capacity, age, height, smoke('yes' for a smoker and 'no' for a non-smoker), gender(male/female), and Caesarean(yes/no) of a person divide the ages into groups and then try to plot stratified boxplots for the lung capacity of smokers vs non-smokers with age strata.												
3	Perform Data Cleaning on Air Quality data set Load Air Quality dataset and also Perform the followings. <div><div>a. Check all the observations with missing values</div><div>b. Check the outliers with boxplo</div><div>c. Clean the data by removing outliers and treat missing values.</div><div>d. Impute the missing values in the original dataset with "mean" of the respective variables</div></div>												
4	Principal Component Analysis Perform Multivariate Analysis using PCA on IRIS data set for developing a predictive model.												
5	Similarity Measure with Data Normalization: Three friends with age and education is given in the table below <table><tr><th>Name</th><th>Age(in years)</th><th>Education</th></tr><tr><td>Bala</td><td>43</td><td>2.0</td></tr><tr><td>Ganesh</td><td>38</td><td>4.2</td></tr><tr><td>Jeevan</td><td>42</td><td>4.1</td></tr></table> <div>Compute the following</div> <div>1)Calculate the Euclidean distance between these friends to find the most similar friends</div> <div>2)Do the same calculation measuring the ages in decades(Divide the age by 10)</div> <div>3)Normalize the data using min-max method and find the most similar friends</div> <div>4)Compare the results with normalized and without normalized data</div>	Name	Age(in years)	Education	Bala	43	2.0	Ganesh	38	4.2	Jeevan	42	4.1
Name	Age(in years)	Education											
Bala	43	2.0											
Ganesh	38	4.2											
Jeevan	42	4.1											
6	Data Conversion from Qualitative to Quantitative Dimensionality Reduction: Attribute Selection – Filters In the given table, name of the contact, the maximum temperature registered last week in their town, their weight, height, year of experience and gender, together with the information on how good their company is given. Show how similar the behaviour of each predictive attribute is to the target attribute Company and rank the attributes according to Pearson correlation and filter the predictive attribute with correlation below the given threshold <table><tr><td>Contact</td><td>Maxtemp</td><td>Height</td><td>Years</td><td>Gender</td><td>Company</td></tr><tr><td colspan="6">Weight</td></tr></table>	Contact	Maxtemp	Height	Years	Gender	Company	Weight					
Contact	Maxtemp	Height	Years	Gender	Company								
Weight													

Lab Exp No: 1

Demonstrate the Negative (-ve) and Positive (+ve) Correlation between two attributes of Women data set.

Theory

Correlation is a mathematical concept which is commonly used in the field of probability and statistics. Where concepts describe the relationship between two variables.

Correlation –

1. It show whether and how strongly pairs of variables are related to each other.
2. Correlation takes values between -1 to +1, wherein values close to +1 represent strong positive correlation and values close to -1 represents strong negative correlation.
3. In this variable are indirectly related to each other.
4. It gives the direction and strength of relationship between variables.

Formula –

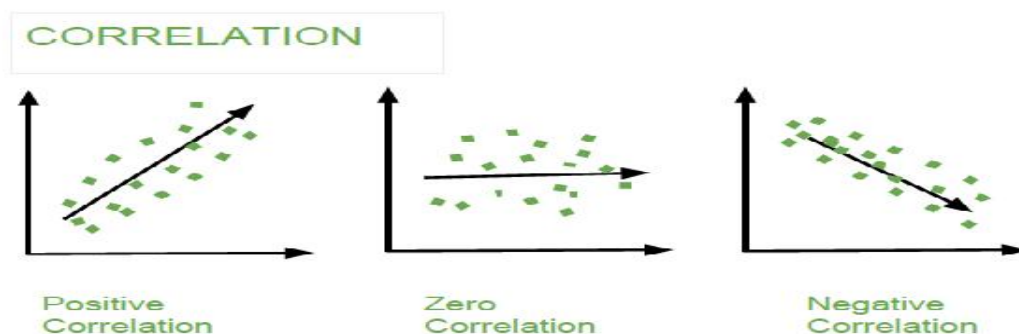
$$\text{Corr}(x, y) = \frac{\sum_{i=1}^n (x_i - x') (y_i - y')}{\sqrt{\sum_{i=1}^n (x_i - x')^2 \sum_{i=1}^n (y_i - y')^2}}$$

Here,

x' and y' = mean of given sample set

n = total no of sample

x_i and y_i = individual sample of set



Syntax for Correlation:

`Cor(x, y, method)`

Where x and y represents the data vectors and method defines the type of method to be used to compute correlation.

Example Program:

```
x<- c(1 ,3, 5, 10)
y<- c(2 ,4, 6, 20)
# Print Correlation using different methods
print(cor(x, y))
print(cor(x, y, method = "pearson"))
print(cor(x, y, method = "kendall"))
print(cor(x, y, method = "spearman"))
```

Algorithm:

Step1: Start with R-Console and check the availability of Dataset in the R Console

With data() command

Step-2: Find the “women” dataset and “mtcar” data sets and identify the any attributes for which correlation is need to be find

Step-3: Assign the one attribute to X and second attribute to Y variables

x represents the x data vector

y represents the y data vector

x' represents mean of x data vector

y' represents mean of y data vector

Step-4: Print covariance using different methods

Step-5: Plot the Graph with main and axis titles

Step-6: Add the Regression Line.

R Code for Positive Correlation

```
x <- women$height
```

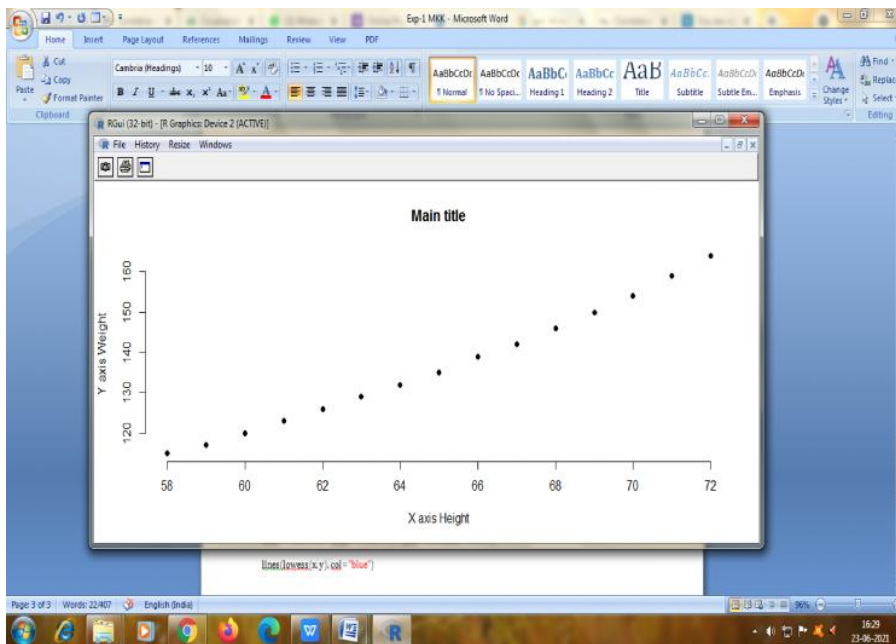
```
y <- women$weight
```

```
# Plot with main and axis titles
```

```
# Change point shape (pch = 19) and remove frame.
```

```
plot(x, y, main = "Main title", xlab = "X axis Height", ylab = "Y axis Weight", pch = 19, frame = FALSE)
```

Output: 1



R Code for Negative Correlation

```
x<-c(1,12,45,46,47,38,29,40)
```

```
x
```

```
[1] 1 12 45 46 47 38 29 40
```

```
y<-c(10,15,20,25,30,35,40,45,50)
```

```
y
```

```
[1] 10 15 20 25 30 35 40 45 50
```

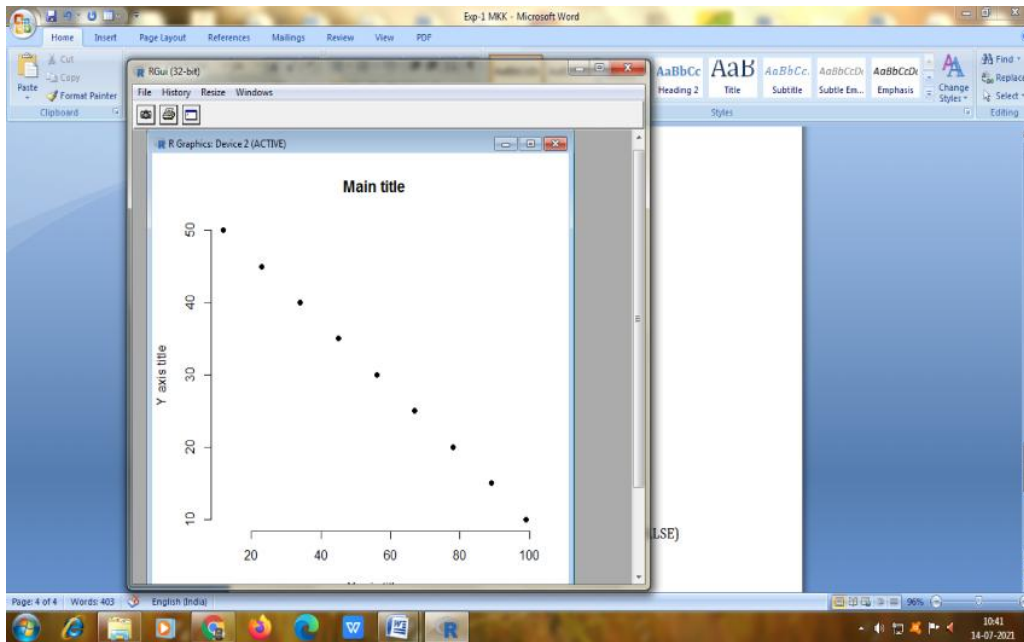
```
cor(x,y)
```

Output:

```
[1] -0.9999566
```

```
plot(x, y, main = "Main title",
```

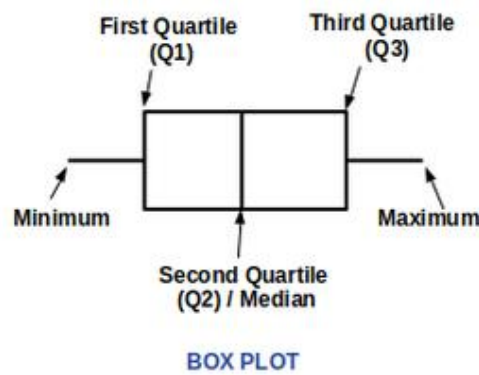
```
+ xlab = "X axis title", ylab = "Y axis title", pch = 19, frame = FALSE)
```



Lab Exp-2

Create box plot for the two Variables group of LungCapData dataset having 6 variables each signifying lung capacity, age, height, smoke('yes' for a smoker and 'no' for a non-smoker), gender(male/female), and Caesarean(yes/no) of a person divide the ages into groups and then try to plot stratified boxplots for the lung capacity of smokers vs non-smokers with age strata

Box Plot: A box plot is a method to represent the group of numerical data in the form of quartiles. The quartiles are the values at a particular percentile in the whole dataset. Box plots indicate the five-number summary of the set of data. The five-number summary has the value of the data as a minimum, first quartile, second quartile (median), third quartile, and maximum.



As shown in the above image, a box plot also has whiskers from the first quartile to minimum and from the third quartile to maximum. This article focuses on displaying a boxplot without whiskers.

Components of Boxplot

- Minimum : The lowest value in the dataset excluding outliers.
- First Quartile (Q1) : The value in the dataset at 25th Percentile.
- Second Quartile (Q2) : The value in the dataset at 50th Percentile. It is also known as the median of the data.
- Third Quartile (Q3) : The value in the dataset at 75th Percentile.
- Maximum : The highest value in the dataset excluding outliers.
- Interquartile Range (IQR) : The distance between first quartile (Q1) and third quartile (Q3). IQR is defined as follows :
- Whiskers : The lines shown above from minimum to Q1 and Q3 to maximum are whiskers.
- Outliers : Any values less than minimum and greater than maximum are the outliers of the data.

Function used:

Syntax: `boxplot(x, data, notch, varwidth, names, main)`

Parameters:

- **x:** This parameter sets as a vector or a formula.
- **data:** This parameter sets the data frame.
- **notch:** This parameter is the label for horizontal axis.
- **varwidth:** This parameter is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **main:** This parameter is the title of the chart.
- **names:** This parameter are the group labels that will be showed under each boxplot.

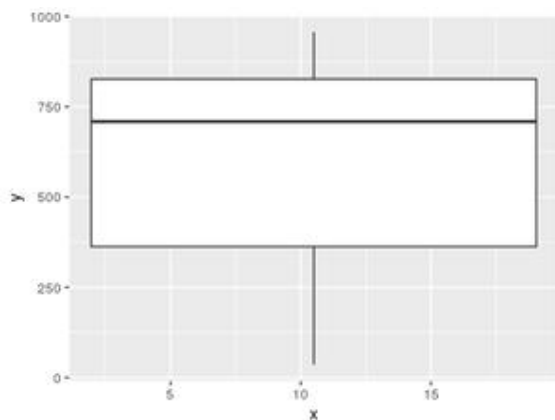
Sample Example Code:

```
x <- 1:20
y <- sample(1000,20, replace = TRUE)

df <- data.frame(x,y)
library(ggplot2)

ggplot(df, aes(x,y, group =1))+geom_boxplot()
```

Output :



Boxplot with whiskers

Now for creating the same plot without whiskers `coef` parameter of the `geom_boxplot()` function should be set to 0. Here, parameter `coef` is the length of the whiskers as the multiple of IQR. The default value is 1.5 but here we have set it to 0. So, the whiskers are eliminated.

Coding Part:

```
# Load the dataset
LungCapData <- read.csv("LungCapData.csv", header = T)
LungCapData <- data.frame(LungCapData)
attach(LungCapData)

# Catgorise Age into groups
AgeGroups <- cut(LungCapData$Age,
                 breaks = c(0, 13, 15, 17, 25),
                 labels = c("<13", "14/15", "16/17", ">=18"))
head(LungCapData)

# BoxPlot 1
boxplot(LungCapData$LungCap~LungCapData$Smoke,
        ylab = "Capacity",
        main = "Lung Capacity of Smokers Vs Non-Smokers",
        las = 1)

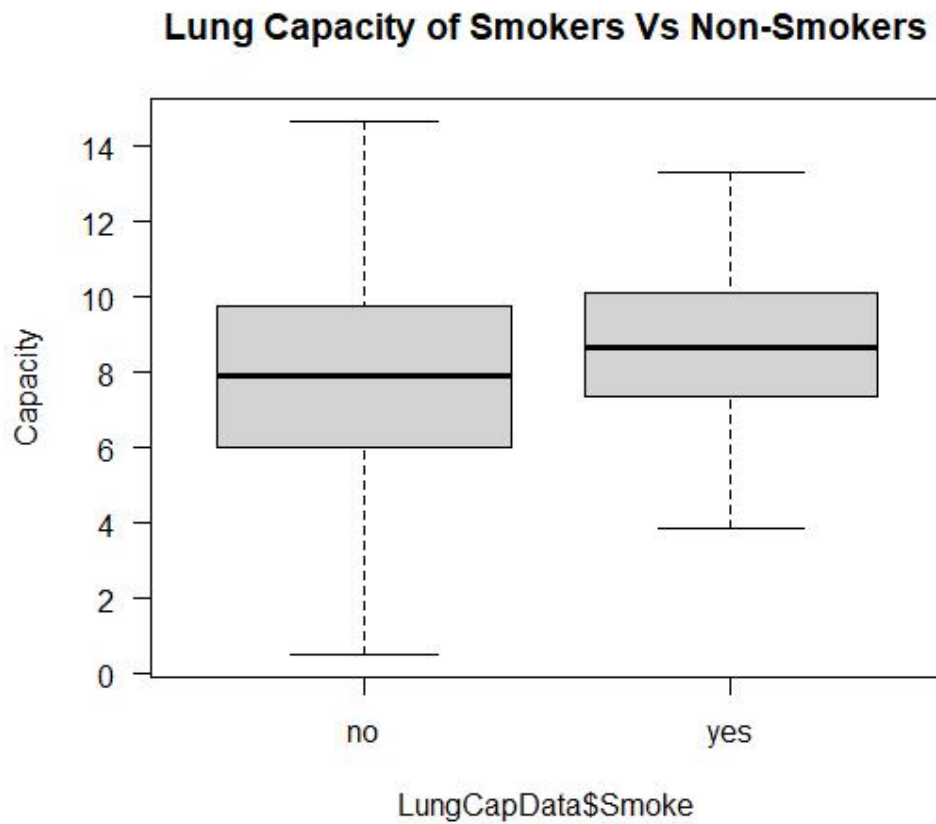
# BoxPlot 2
boxplot(LungCapData$LungCap[LungCapData$Age>=18]~LungCapData$Smoke[LungCapData
$Age>=18],
        ylab = "Capacity",
        main = "Lung Capacity of Smokers Vs Non-Smokers",
        las = 1)

# BoxPlot 3
boxplot(LungCapData$LungCap~LungCapData$Smoke*AgeGroups,
        ylab = "Capacity", xlab = "",
        main = "Lung Capacity of Smokers Vs Non-Smokers",
        col = c(4, 2), las = 2)
```

Output:

Boxplot 1

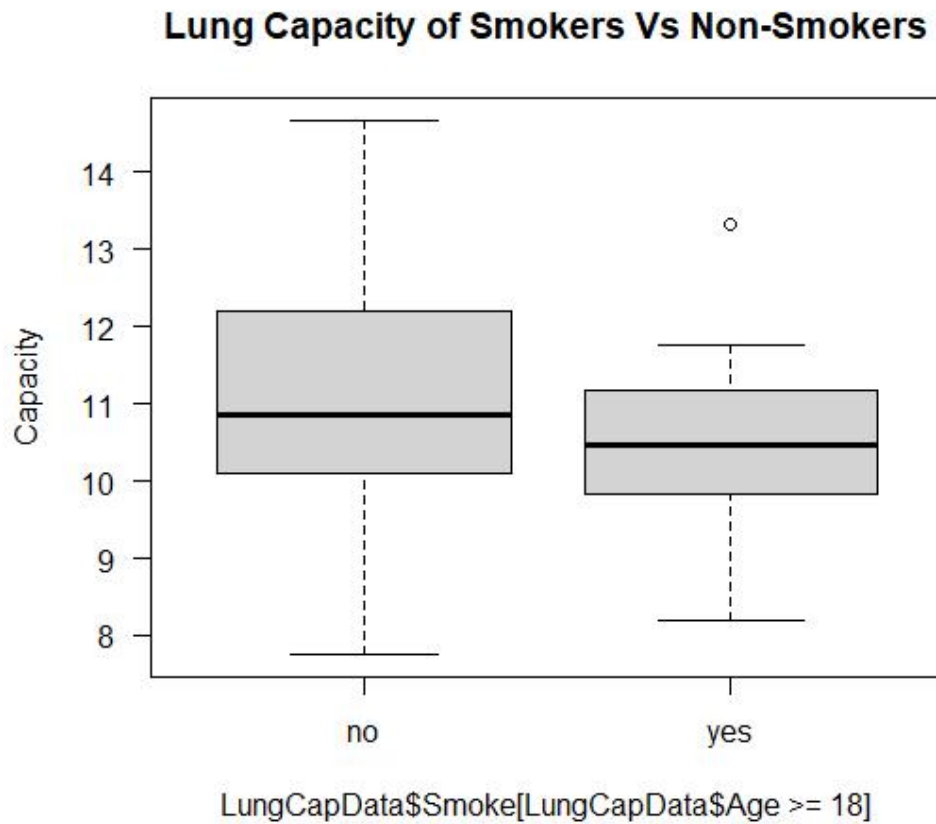
Boxplot 1 plots the lung capacity of smokers and non-smokers, where no symbolize non-smokers, and yes symbolizes smokers.



By analyzing the above-shown boxplot we can clearly say the lung capacity of non-smokers is lower as compared to that of smokers on an average.

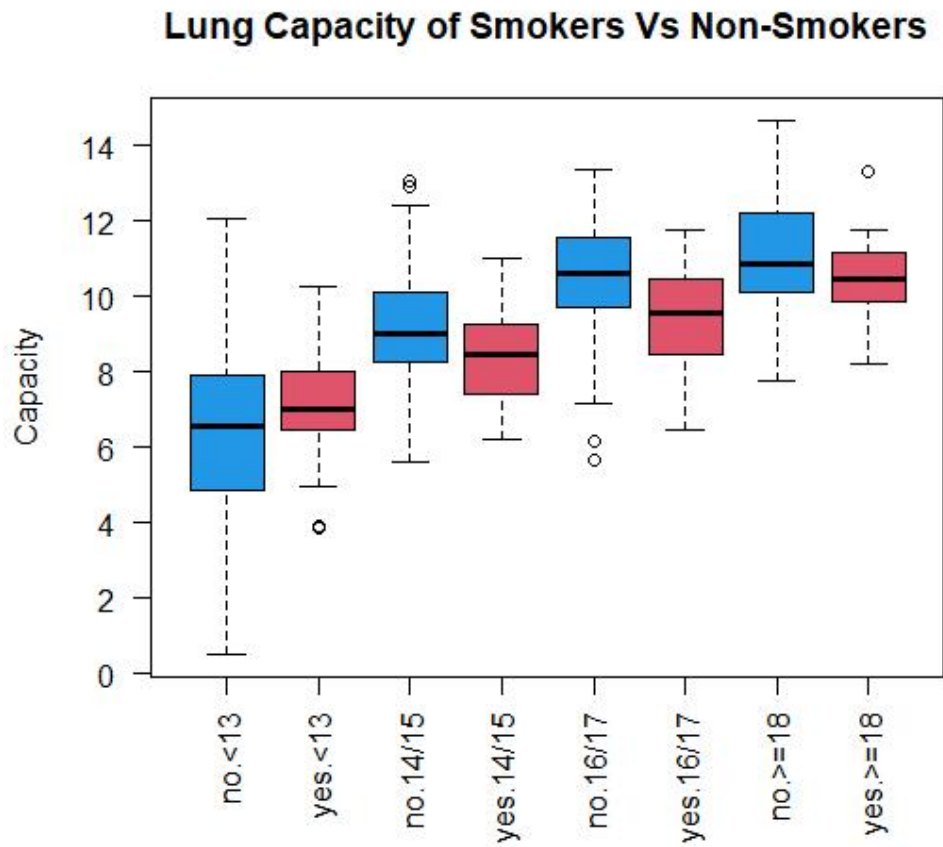
Boxplot 2

Boxplot 2 plots the lung capacity of smokers and non-smokers of age group greater or equal to 18, where no symbolizes non-smokers and yes symbolizes smokers.



Boxplot 3

Boxplot 3 plots the lung capacity of smokers and non-smokers of the different age groups in the dataset where blue-colored boxplots are for non-smokers and red is for smokers.



Lab 3. DATA CLEANING ON AIR QUALITY DATASET

- e. Load Air Quality dataset
- f. Check all the observations with missing values
- g. Check the outliers with boxplot
- h. Clean the data by removing outliers and treat missing values.
- i. Impute the missing values in the original dataset with "mean" of the respective variables

Data Cleaning

What is data cleaning?

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset.

When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset. But it is crucial to establish a template for your data cleaning process so you know you are doing it the right way every time.

What is the difference between data cleaning and data transformation?

Data cleaning is the process that removes data that does not belong in your dataset. Data transformation is the process of converting data from one format or structure into another. Transformation processes can also be referred to as data wrangling, or data munging, transforming and mapping data from one "raw" data form into another format for warehousing and analyzing. This article focuses on the processes of cleaning that data.

How do you clean data?

While the techniques used for data cleaning may vary according to the types of data your company stores, you can follow these basic steps to map out a framework for your organization.

Steps to Clean the Data

Step1: Remove duplicate or irrelevant observations

Step2: Fix structural errors

Step3: Filter unwanted outliers

Step4: Handle missing data & there are a couple of ways to deal with missing data.

- ➔ As a first option, you can drop observations that have missing values, but doing this will drop or lose information, so be mindful of this before you remove it.

- ➔ As a second option, you can input missing values based on other observations; again, there is an opportunity to lose integrity of the data because you may be operating from assumptions and not actual observations.
- ➔ As a third option, you might alter the way the data is used to effectively navigate null values.

Step 5: Validate and QA

Components of quality data

Determining the quality of data requires an examination of its characteristics, then weighing those characteristics according to what is most important to your organization and the application(s) for which they will be used.

5 characteristics of quality data

1. **Validity.** The degree to which your data conforms to defined business rules or constraints.
2. **Accuracy.** Ensure your data is close to the true values.
3. **Completeness.** The degree to which all required data is known.
4. **Consistency.** Ensure your data is consistent within the same dataset and/or across multiple data sets.
5. **Uniformity.** The degree to which the data is specified using the same unit of measure.

Algorithm

1. Read the Data from dataset.
2. Assign the data set to variable Airqual
3. Check the missing Value from the data set
4. Check the dataset for the outliers
5. Check the Summary ones again for the missing Values
6. Remove the Outliers.

R Code

```
airqual=airquality //airquality is a dataset from the package "datasets"
summary(airquality) //summary to check the missing values, mean and median of the data
distributions
airqual[!complete.cases(airqual),]
//we can see there are 37 observations missing in Ozone for the variable and 7 observation missing in
Solar.R
boxplot(airqual)
```

```
//Checking all the observations with missing values
//Checking the outliers with boxplot
//we can see the variables Ozone and Wind has outliers.
```

Steps to clean the data by removing outliers and treating missing values.

- a) Mean is badly affected by outliers, hence to impute missing values with mean we have to remove Outliers.
- b) The outliers are removed and data is stored in new dataset called Updated_airqual.
- c) The purpose of Updated_airqual dataset is to impute missing values in airqual dataset.
- d) Impute the missing values in airqual dataset with respective mean from Updated_airqual dataset.
- e) Now airqual dataset has no missing values, so we can remove outliers as we did in the step-b
- f) Data is ready for analysis.

R-Code

```
boxplot(airqual$Ozone,horizontal = TRUE) // Lets exclude the outliers for this two variables
boxplot(airqual$Wind,horizontal = TRUE) // we see that observation above 120 could be outlier, hence
// we will consider all below 120 Similarly for ozone We can
// see that observations above 17 are outliers, hence we will
// consider all below 17 therefore removing the outliers
Updated_airqual=subset(airqual,Ozone<130 & Wind<17) //lets check if all the outliers are removed
boxplot(Updated_airqual)

//Using subset will also remove all the NA's present in the variable selected hence reducing the dataset.
// So after removing the outliers we have to fill missing values in the original dataset("airquality") to restore.
// Let impute the missing values in the original dataset with "mean" of the respective variables.
airqual$Ozone[is.na(airqual$Ozone)]<-mean(Updated_airqual$Ozone)
//Check summary if their are any mising values in airquality data
summary(airqual)
// There are no NA's for Ozone. Similarly we have to do the same with Wind variable.
airqual$Solar.R[is.na(airqual$Solar.R)]<-mean(Updated_airqual$Solar.R,na.rm = TRUE)
// Check summary if all the missing values are treated with mean.
summary(airqual)
Now we remove the outliers from the airqual dataset.
data_airquality=subset(airqual,Ozone<70 & Wind<17)
boxplot(data_airquality)
nrow(data_airquality)
data_airquality=subset(airqual,Ozone<70 & Wind<17 & Wind>2)
boxplot(data_airquality)
```

LAB 4. PRINCIPAL COMPONENT ANALYSIS (PCA)

Perform Multivariate Analysis using PCA on IRIS data set for developing a predictive model.

What Is Principal Component Analysis?

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

PCA finds a new set of dimensions (or a set of basis of views) such that all the dimensions are orthogonal (and hence linearly independent) and ranked according to the variance of data along them. It means more important principle axis occurs first. (more important = more variance/more spread out data)

Algorithm

Step1: Load the Data Set
Step2: Display Compactly the Internal Data Set
Step3: Find the Summary of Data Set (Descriptive Statistics)
Step 4: Perform the Partition Data
Step 5: Perform Random Sampling of the Data Set
Step 6: Draw the Scatter plot for the Data Set.
Step 7: Conduct PCA

Source Code

```
data("iris") //Load the iris dataset

str(iris) //Compactly displaying the internal structure of a dataset

summary(iris) //The summary function returned descriptive statistics

set.seed(111) // function sets the starting number used to generate a sequence of random numbers

ind <- sample(2, nrow(iris),

             replace = TRUE,

             prob = c(0.8, 0.2))

training <- iris[ind==1,]

testing <- iris[ind==2,] //sample() function allows to take a random sample of elements from a
                        data-set
```

```
library(psych) // Scatter Plot & Correlations
```

```
    // pairs.panels [in psych package] is used to create a scatter plot of matrices
```

```
pairs.panels(training[,-5],
```

```
    gap = 0,
```

```
    bg = c("red", "yellow", "blue")[training$Species],
```

```
    pch=21)
```

```
pc <- prcomp(training[,-5],
```

```
    center = TRUE,
```

```
    scale. = TRUE )  ## Principal Component Analysis
```

```
attributes(pc)
```

```
pc$center
```

```
pc$scale
```

```
print(pc)
```

```
summary(pc)
```

Lab 5: Similarity Measure with Data Normalization

Three friends with age and education is given in the table below

Name	Age(in years)	Education
Bala	43	2.0
Ganesh	38	4.2
Jeevan	42	4.1

Compute the following

- Calculate the Euclidean distance between these friends to find the most similar friends
- Do the same calculation measuring the ages in decades(Divide the age by 10)
- Normalize the data using min-max method and find the most similar friends
- Compare the results with normalized and without normalized data

Similarity Measure

The similarity measure is a way of measuring how data samples are related or closed to each other. The purpose of a measure of similarity is to compare two or more sample data (i.e. vectors). Measures of similarity provide a numerical value which indicates the strength of associations between objects or variables. The basis of many measures of similarity and dissimilarity is that covers a wide range of scores and measures for assessing the differences among various kinds of data. Euclidean distance measures similarity between 2 points.

Euclidean distance

- Euclidean distance measures similarity between 2 entities
- Lesser the value more similar they are
- Euclidean distance between 2 values (x_1, y_1) and (x_2, y_2) is Given by

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

One problem with comparing two variables is that they may not be measured on the same scale. Solution to this problem is to have same scale for objects to be compared

Converting to a Different Scale

Converting data in a scale to another scale of the same type is necessary when using distance measures. This is because; the results can be different depending on the measure in which the values of a given attribute are expressed. This conversion is done in order to have different attributes expressed on the same scale. This a process known as “normalization”.

Normalization

The normalization is carried out for each attribute individually. There are two ways to normalize the data

- Standardization
- Min–Max rescaling

Min –Max rescaling

Min–max rescaling converts numerical values to values in a given interval. To convert a set of values to values in the interval [0.0, ...,1.0]

1. Subtract the smallest value from all the values in the set
2. Divide the new values by the amplitude(max-min)

Algorithm

Step 1: Initialize the input data frame

Step 2: Extract age and education data and form a matrix M1

Step 3: Find the Euclidean distance of the matrix M1 call E1

Step 4: Replace ages in decades in M1 and get matrix M2 and find the Euclidean distance of the matrix and Call E2

Step 5: Normalize the attributes age and education of the matrix M1 using Min-Max method and call M3

Step 6: Find the Euclidean distance of the matrix M3 call E3

Step 7: Observe the values E1,E2,E3

Data Frames in R

- A structure for storing and accessing several variables of possibly different data types
- It is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column

Matrices in R

- Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout
- They contain elements of the same atomic types
- We can create a matrix containing only characters or only logical values and numeric elements to be used in mathematical calculations
- A Matrix is created using the matrix() function

CODE IN R

```
rm(list = ls(all.names = TRUE))    # Remove all objects in the present in the workspace
x <- data.frame(                   # Initialize the input data frame
  name = c("Bala", "Ganesh", "Geevan"),
  age = c(43, 38, 42),
  education = c(2.0, 4.2, 4.1)
)
print(x)
```

```
age = x$age                        # Extract age and Education from data frame using $
print(age)
edu = x$education
print(edu)
```

```
M1 <- matrix(c(age, edu), nrow = 3, byrow = FALSE)    # Form a Matrix M1
print(M1)
E1 = dist(M1, method = 'euclidean')    # Call Euclidean distance and get E1
print(E1)
```

```
# Age in years to Decade-----
```

```
ageD = age/10
print(ageD)
```

```

M2 <- matrix(c(ageD,edu), nrow = 3, byrow = FALSE)    # Form a Matrix M2
print(M2)
E2 = dist(M2,method= 'euclidean')          # Call Euclidean distance and get E2
print(E2)
#-----Min-Max Normalize Age-----
rangA = max(age) - min(age)    # compute range
print(r)
mi_maA = (age-min(age))/rangA
print(mi_maA)
#-----Normalize Education attribute-----
rangE = max(edu) - min(edu)
print(r)
mi_maE = (edu-min(edu))/rangE
print(mi_maE)
#----- Form a Matrix M3 with Normalized values

M3 <- matrix(c(mi_maA,mi_maE), nrow = 3, byrow = FALSE)
print(M3)
E3 = dist(M3,method= 'euclidean')    # Call Euclidean distance and get E3
print(E3)
print("_____Data frame with different similarity measure matrixresults_____")
print(x)
print(E1)
print(E2)
print(E3)

```

OUTPUT

Data frame

1	Bala	43	2.0
2	Ganesh	38	4.2
3	Geevan	42	4.1

>print(E1) Euclidean distance with given age and education

	1	2
2	5.462600	
3	2.325941	4.001250

>print(E2)Euclidean distance with age in decade and education)

	1	2
2	2.2561028	
3	2.1023796	0.4123106

>print(E3) Euclidean distance on Normalized data(age and education))

	1	2
2	1.4142136	
3	0.9752728	0.8012903

Lab 6:

Data Conversion from Qualitative to Quantitative Dimensionality Reduction: Attribute Selection – Filters

In the given table, name of the contact, the maximum temperature registered last week in their town, their weight, height, year of experience and gender, together with the information on how good their company is given. Show how similar the behaviour of each predictive attribute is to the target attribute **Company** and rank the attributes according to Pearson correlation and filter the predictive attribute with correlation below the given threshold

Contact	Maxtemp	Weight	Height	Years	Gender	Company
Andrew	25	77	175	10	M	Good
Bernhard	31	110	195	12	M	Good
Carolina	15	70	172	2	F	Bad
Dennis	20	85	180	16	M	Good
Eve	10	65	168	0	F	Bad
Fred	12	75	173	6	M	Good
Gwyneth	16	75	180	3	F	Bad
Hayden	26	63	165	2	F	Bad
Irene	15	55	158	5	F	Bad
James	21	66	163	14	M	Good
Kevin	30	95	190	1	M	Bad
Lea	13	72	172	11	F	Good
Marcus	8	83	185	3	F	Bad
Nigel	12	115	192	15	M	Good

Note: Create a CSV file of the contact information after conversion

Converting Nominal to Relative

Since the nominal scale (gender and company) does not assume an order between its values, to keep this information, nominal values should be converted to relative or binary values. A binary attribute has only two values, 0 or 1. In the contact information gender and Company attributes are having qualitative values (M/F and Good/Bad). To compute the Pearson correlation these attribute values should be converted to numerical values. Here in the contact table for gender attribute 'M' is replaced by "1" and 'F' is replaced by "0". Similarly, for company attribute "Good" is replaced by "1" and "Bad" by "0" and the data in the CSV file is as shown below (Your CSV file).

Contact	Maxtemp	Weight	Height	Years	Gender	Company
Andrew	25	77	175	10	1	1

Bernhard	31	110	195	12	1	1
Carolina	15	70	172	2	0	0
Dennis	20	85	180	16	1	1
Eve	10	65	168	0	0	0
Fred	12	75	173	6	1	1
Gwyneth	16	75	180	3	0	0
Hayden	26	63	165	2	0	0
Irene	15	55	158	5	0	0
James	21	66	163	14	1	1
Kevin	30	95	190	1	1	0
Lea	13	72	172	11	0	1
Marcus	8	83	185	3	0	0
Nigel	12	115	192	15	1	1

Dimensionality Reduction or Dimension reduction

The transformation of data from a high-dimensional space into a low- dimensional space so that the low-dimensional representation retains some meaningful properties of the original data. To represent more attributes in a clear and easy-to-understand plot, reduce the level of detail for each attribute.

Benefits of Dimensionality Reduction

- Reducing the training time, decreasing memory needed and improving performance of ML algorithms
- Eliminating irrelevant attributes and reducing the number of noisy attributes
- Allowing the induction of simpler and therefore more easily interpretable models
- Making the data visualization easier to understand and allowing visualization of data sets with a high number of attributes
- Reducing the cost of feature extraction, making ML-based technologies accessible to a larger number of people

There are two alternatives to reduce the number of attributes:

☐ Attribute aggregation

OR

☐ Attribute selection

Attribute aggregation: Replace a group of attributes with a new attribute: a combination of the attributes in the group

Attribute selection (also called “feature selection”): Select a subset of the attributes that keeps most of the information present in the original data set. Attribute selection techniques can be roughly divided into three categories:

- ☐ Filters
- ☐ Wrappers
- ☐ Embedded

Filtering:

- Filters look for simple, individual, relations between the predictive attribute values and the target attribute
- Rank the attributes according to this relation
- Predictive attribute having a strong relation with a label attribute value receives a high ranking position
- High ranked attributes are retained and low ranked attributes are eliminated

Pearson correlation

Correlation is a statistic that measures the degree to which two variables move in relation to each other. To rank and filter the attributes, apply a statistical measure. Pearson correlation is computed for all predictive attributes with respect to target attribute “company”. The equation for Pearson correlation is

- r_{xy} – the correlation coefficient of the linear relationship between the variables x and y
- x_i – the values of the x-variable in a sample
- \bar{x} – the mean of the values of the x-variable
- y_i – the values of the y-variable in a sample
- \bar{y} – the mean of the values of the y-variable

$$r_{xy} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Algorithm

STEP 1: Read the data from CSV file

STEP 2: Display the data

STEP 3: Compute correlation for all predictive attributes Maxtemp, Weight, Height, Year, and Gender to the target attribute Company

STEP 4: Print the scatter plot matrix for correlation matrix

STEP 5: Extract the last row of correlation matrix excluding company

STEP 6: Fix the threshold value

STEP 7: Write a loop to display the predictive attributes with correlation with target attribute greater than threshold

Attribute selection by filtering

```
rm(list = ls(all.names = TRUE)) # Remove all objects in the present in the workspace
input = read.csv("filter.csv")    # Read the CSV file from working directory

print(input)

x = cor(input[,c(2:7)])    # Find correlation of all rows with 2nd to 7th column (attributes)

print(x)

pairs(input[,c(2:7)]) )    #print the scatter plot matrix for correlation matrix

y = x[6,1:5]    #Extract the last row of correlation matrix excluding company

print(y)

x= sort(y,decreasing = TRUE)

print(x)

cnt =1

print("selected attributes")

while(cnt<=length(y))    # Loop to display all attributes having correlation with target
    # attribute greater than threshold (0.5)

    { if (y[cnt]>.5)

print(y[cnt])

cnt=cnt+1

}
```

Note : Generalize the Program

OUTPUT

Correlation Matrix

	Maxtemp	Weight	Height	Years	Gender	Company
Maxtemp	1.0000000	0.2660784	0.2408598	0.1382285	0.4775233	0.1392776
Weight	0.2660784	1.0000000	0.9441944	0.4311897	0.5970054	0.4008465
Height	0.2408598	0.9441944	1.0000000	0.2206131	0.4481060	0.2108734

Years 0.1382285 0.4311897 0.2206131 1.0000000 0.6291870 0.8913483
Gender 0.4775233 0.5970054 0.4481060 0.6291870 1.0000000 0.7142857
Company 0.1392776 0.4008465 0.2108734 0.8913483 0.7142857 1.0000000

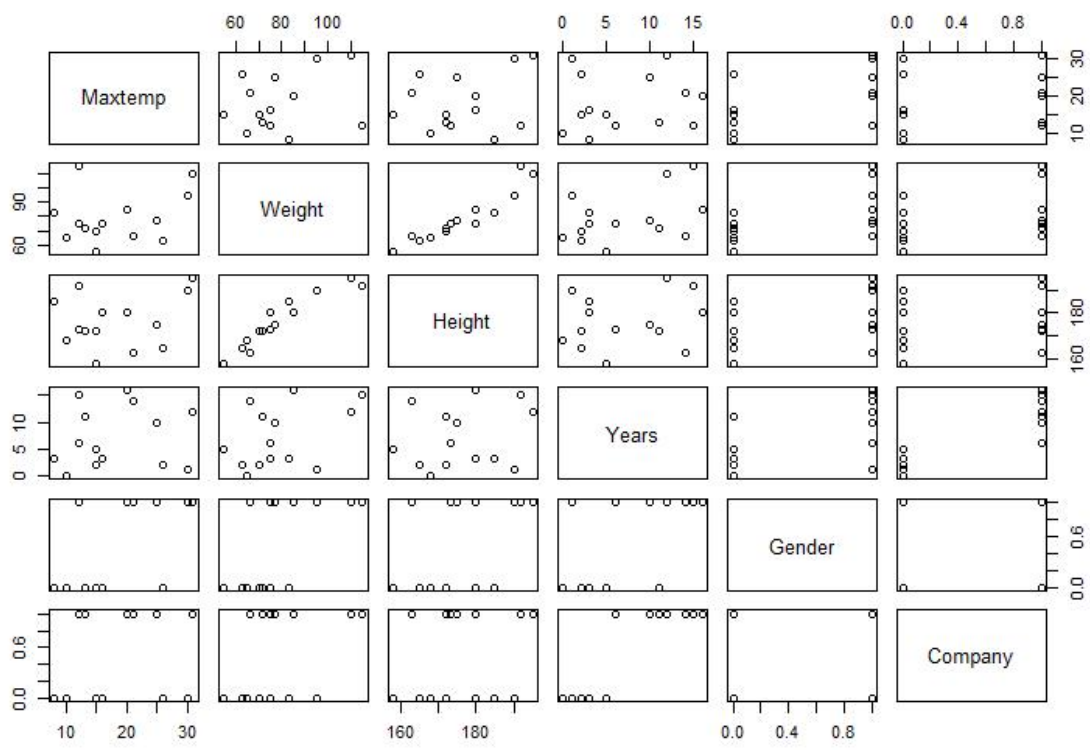
Correlation of Predictive attributes with company (Last Row)

Maxtemp Weight Height Years Gender
0.1392776 0.4008465 0.2108734 0.8913483 0.7142857

SELECTED ATTRIBUTES

Years
0.8913483
Gender
0.7142857

Correlation Matrix



Lab 7: K-Means Clustering in R Programming

K Means Clustering in R Programming is an Unsupervised Non-linear algorithm that cluster data based on similarity or similar groups. It seeks to partition the observations into a pre-specified number of clusters. Segmentation of data takes place to assign each training example to a segment called a cluster. In the unsupervised algorithm, high reliance on raw data is given with large expenditure on manual review for review of relevance is given. It is used in a variety of fields like Banking, healthcare, retail, Media, etc.

Theory

K-Means clustering groups the data on similar groups. The algorithm is as follows:

1. Choose the number **K** clusters.
2. Select at random K points, the centroids(Not necessarily from the given data).
3. Assign each data point to closest centroid that forms K clusters.
4. Compute and place the new centroid of each centroid.
5. Reassign each data point to new cluster.

After final reassignment, name the cluster as Final cluster.

The Dataset

iris dataset consists of 50 samples from each of 3 species of Iris(Iris setosa, Iris virginica, Iris versicolor) and a multivariate dataset introduced by British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems. Four features were measured from each sample i.e length and width of the sepals and petals and based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Coding

```
# Loading data
```

```
data(iris)
```

```
# Structure
```

```
str(iris)
```

Performing K-Means Clustering on Dataset

Using K-Means Clustering algorithm on the dataset which includes 11 persons and 6 variables or attributes

```
# Installing Packages
```

```
install.packages("ClusterR")
```

```
install.packages("cluster")
```

```
# Loading package
```

```
library(ClusterR)
```

```
library(cluster)
```

```
# Removing initial label of
```

```
# Species from original dataset
```

```
iris_1 <- iris[, -5]
```

```
# Fitting K-Means clustering Model
```

```
# to training dataset
```

```
set.seed(240) # Setting seed
```

```
kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)
```

```
kmeans.re
```

```
# Cluster identification for
```

```
# each observation
```

```
kmeans.re$cluster
```

```
# Confusion Matrix
```

```
cm <- table(iris$Species, kmeans.re$cluster)
```

```
cm
```

```
# Model Evaluation and visualization
```

```
plot(iris_1[c("Sepal.Length", "Sepal.Width")])
```

```
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
```

```
      col = kmeans.re$cluster)
```

```
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
```

```
      col = kmeans.re$cluster,
```

```
      main = "K-means with 3 clusters")
```

```
## Plotting cluster centers
```

```
kmeans.re$centers
```

```
kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]
```

```
# cex is font size, pch is symbol
```

```
points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")],
```

```
      col = 1:3, pch = 8, cex = 3)
```

```
## Visualizing clusters
```

```
y_kmeans <- kmeans.re$cluster
```

```
clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],
```

```
        y_kmeans,
```

```
        lines = 0,
```

```
        shade = TRUE,
```

```
        color = TRUE,
```

```
labels = 2,  
plotchar = FALSE,  
span = TRUE,  
main = paste("Cluster iris"),  
xlab = 'Sepal.Length',  
ylab = 'Sepal.Width')
```

Output:

Model kmeans_re:

[illegible]

The 3 clusters are made which are of 50, 62, and 38 sizes respectively. Within the cluster, the sum of squares is 88.4%.

Cluster identification

```
> confusionMatrix(cm)
Confusion Matrix and Statistics

      setosa versicolor virginica
setosa      20         0         0
versicolor   0        20         0
virginica     0         0        20

Overall Statistics

          Accuracy : 1
          95% CI   : (0.9404, 1)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

          Kappa : 1

  Mcnemar's Test P-Value : NA

Statistics by Class:

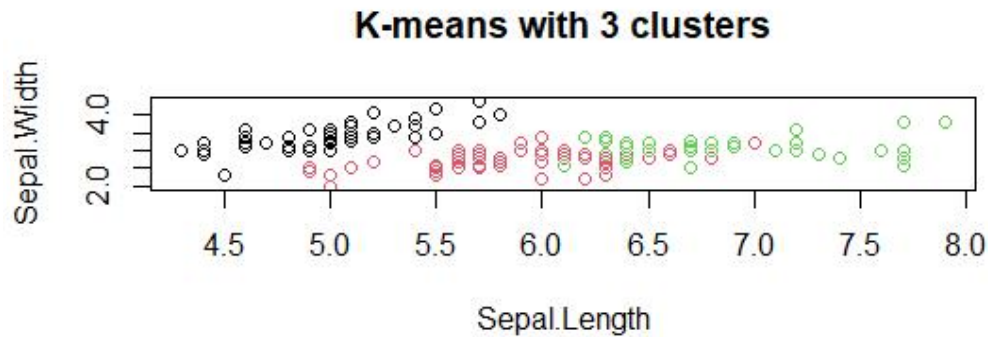
                class: setosa class: versicolor class: virginica
Sensitivity                1.0000                1.0000                1.0000
Specificity                1.0000                1.0000                1.0000
Pos Pred Value              1.0000                1.0000                1.0000
Neg Pred Value              1.0000                1.0000                1.0000
Prevalence                  0.3333                0.3333                0.3333
Detection Rate              0.3333                0.3333                0.3333
Detection Prevalence        0.3333                0.3333                0.3333
Balanced Accuracy           1.0000                1.0000                1.0000
```

- The model achieved an accuracy of 100% with a p-value of less than 1. This indicates the model is good.
- **Confusion Matrix**

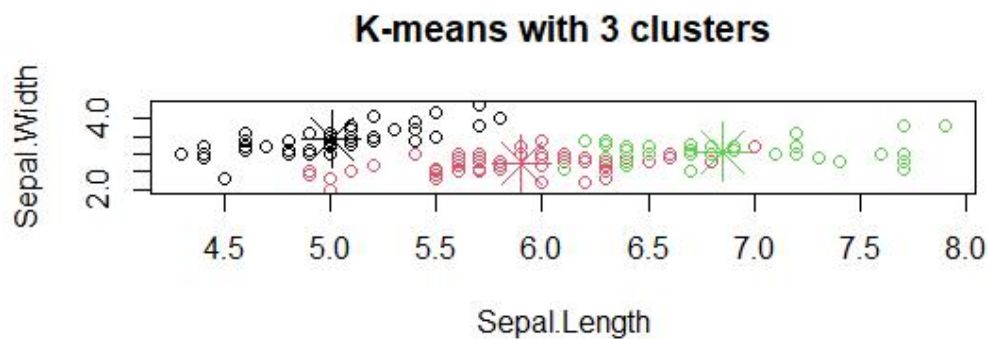
```
> cm

      1  2  3
setosa 50  0  0
versicolor  0 48  2
virginica  0 14 36
```

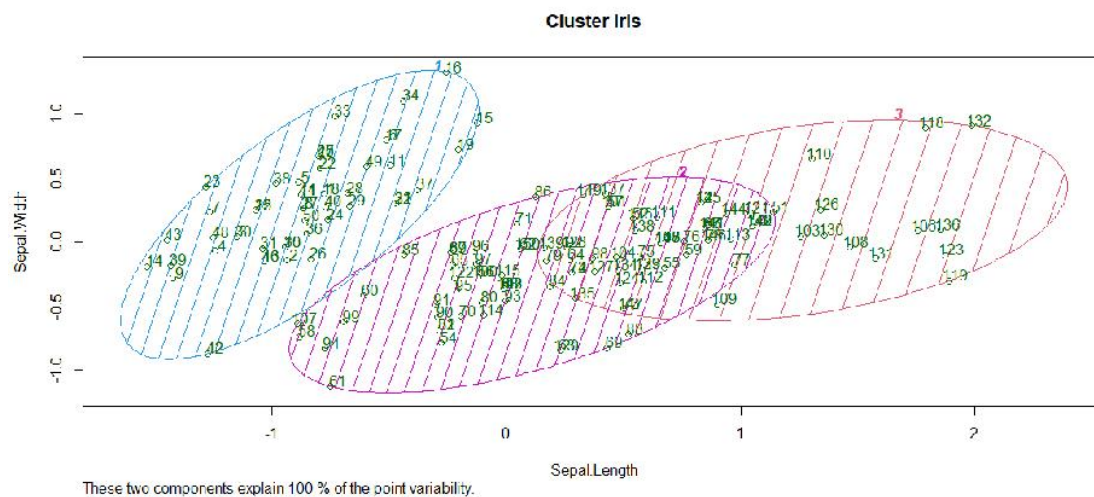
K-means with 3 clusters plot:



- The model showed 3 cluster plots with three different colors and with Sepal.length and with Sepal.width.
- **Plotting cluster centers:**



- In the plot, centers of clusters are marked with cross signs with the same color of the cluster.
- **Plot of clusters:**



So, 3 clusters are formed with varying sepal length and sepal width.

Lab 8:

Find the **frequent itemsets** and generate **association rules** for the following given **transaction dataset**. Assume that minimum support threshold (support = 50%) and minimum confident threshold (confidence = 80%)

Transaction ID	Items
T1	Hot Dogs, Buns, Ketchup
T2	Hot Dogs, Buns
T3	Hot Dogs, Coke, Chips
T4	Chips, Coke
T5	Chips, Ketchup
T6	Hot Dogs, Coke, Chips

Theory: Apriori Algorithm

Apriori algorithm is used for finding frequent itemsets in a dataset for **association rule mining**. It is called Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets. To improve the efficiency of the level-wise generation of frequent itemsets an important property is used called Apriori property which helps by reducing the search space. It's very easy to implement this **algorithm using the R programming language**.

Apriori Property: All non-empty subsets of a frequent itemset must be frequent. Apriori assumes that all subsets of a frequent itemset must be frequent (Apriori property). If an itemset is infrequent, all its supersets will be infrequent.

Important Terminologies

- **Support:** Support is an indication of how frequently the itemset appears in the dataset. It is the count of records containing an item 'x' divided by the total number of records in the database.
- **Confidence:** Confidence is a measure of times such that if an item 'x' is bought, then item 'y' is also bought together. It is the support count of (x U y) divided by the support count of 'x'.

- **Lift:** Lift is the ratio of the observed support to that which is expected if 'x' and 'y' were independent. It is the support count of (x U y) divided by the product of individual support counts of 'x' and 'y'.

Algorithm

Step1:Read each item in the transaction

Step2:Calculate the support of every item

Step3:If support is less than minimum support, discard the item. Else,
insert it into frequent itemset

Step4:Calculate confidence for each non- empty subset

Step 5:If confidence is less than minimum confidence, discard the
subset. Else, it into strong rules

Example:

Dataset:

Transaction ID	Items
T1	Hot Dogs, Buns, Ketchup
T2	Hot Dogs, Buns
T3	Hot Dogs, Coke, Chips
T4	Chips, Coke
T5	Chips, Ketchup
T6	Hot Dogs, Coke, Chips

Step 1: Read each item in the transaction

The items are

I1 - Hot Dogs

I2 – Buns

I3 - Ketchup

I4 – Coke

I5 - Chips

Step 2: Calculate the support of every item

Here find the **frequent itemsets** where the support will be 50%.

(Note: Given the support(or count) in terms of percentage but how to consider support(or count) in terms of values using the formula as mentioned below:)

$$\begin{array}{c} 50\% * \text{total_number_of_transactions in a given dataset} \\ \downarrow \qquad \qquad \downarrow \\ \text{i.e } (50/100) * 6 = 3 \longleftarrow \text{support} \end{array}$$

So, finally support(or count) is 3, then Create a table containing support (or count) of each item present in transaction dataset given in the problem.

Note: find the occurrence of each item in the given dataset as shown in the Table-1.

Table-1

1-Itemset	support(or count)
Hot Dogs	4
Buns	2
Ketchup	2
Coke	3
Chips	4

In Table-1,

- **Hot Dogs** whose support is 4(i.e **Hot Dogs** item frequently purchased **4 times**).

- **Buns** and **Ketchup** whose support is 2(i.e **Buns** and **ketchup** item frequently purchased **2 times**).
- **Coke** whose support is 3(i.e **Coke** item frequently purchased **3 times**).
- **Chips** whose support is 3(i.e **Chips** item frequently purchased **4 times**).

Step 3: If support is less than minimum support, discard the item. Else,
insert it into frequent itemset

Table-2

1-Itemset	support(or count)
Hot Dogs	4
Coke	3
Chips	4

In **Table -2**, set of items are placed whose support ≥ 3 (i.e **Hot Dogs**, **Coke**, **Chips** will meet support (or count) ≥ 3). The items **Buns** and **Ketchup** whose support is 2 does not meet minimum support i.e 3. So, **Buns** and **Ketchup** items are deleted from the table - 2.

From **Table -2**, find out the occurrences of 2-itemset as shown in the **Table-3**.

Table-3

2-Itemset	support(or count)
Hot Dogs, Coke	2
Hot Dogs, Chips	2
Coke, Chips	3

In the above **TABLE -3**, {**Hot Dogs, Coke**}, {**Hot Dogs, Chips**} and {**Coke, Chips**} are frequently occurred 2-itemset. {**Hot Dogs, Chips**} and {**Hot Dogs, Coke**} frequently occurred **itemsets** in a set of transaction whose support is 2, {**Coke, Chips**} itemset whose support is 3. So, {**Hot Dogs, Coke**}, {**Hot Dogs, Chips**} does not meet support ≥ 3 , thus it is deleted from the **table-3**.

There is only one itemset {**Coke, Chips**} whose support is 3 as shown in the table-4.

Table - 4

2-Itemset	support(or count)
Coke, Chips	3

Whether it is possible to consider three item set, it won't possible in this case....So only one itemset is frequent i.e { **Coke, Chips**}

Step 4: Calculate confidence for each non- empty subset

Confidence is a measure of times such that if an item 'X' is bought, then item 'Y' is also bought together. It is the support count of (XUY) divided by the support count of 'X'.

$$\text{Confidence}(X \rightarrow Y) = \text{Support_count}(XUY) / \text{Support_count}(X)$$

Identified the frequently occurred itemset from given transaction dataset is { **Coke, Chips**}. Now generate association rule for the discovered itemset i.e { **Coke, Chips**}.

Rule1: Find the confidence for the item set { **Coke, Chips}.**

User can take item **Coke** first and then take **Chips**, so find the confidence,

$$\{\text{Coke} \rightarrow \text{Chips}\}: \text{support}(\text{Coke} \cup \text{Chips}) / \text{support}(\text{Coke})$$

$$3/3 * 100 = 1 * 100 = 100\%$$

Rule 2: Find the confidence for the item set {Chips, Coke**}.**

User can take item **Chips** first and then take **Coke**, so find the confidence,

$$\{\text{Chips} \rightarrow \text{Coke}\}: \text{support}(\text{Chips} \cup \text{Coke}) / \text{support}(\text{Chips})$$

$$3/4 * 100 = 0.75 * 100 = 75\%$$

The Rule1 giving a confidence is 100% is greater than confidence mentioned in the problem is 80%.

The Rule2 giving a confidence is 75% is less than confidence mentioned the problem(i.e $75\% < 80\%$)...so, Rule2 cannot be considered as strong rule but Rule1 is the strong rule and frequent itemset is

{Coke-> Chips}

Program:

```
# to add the package arules to the R script
```

```
library(arules)
```

```
# read the CSV file from the destination
```

```
SupMarTra<- read.csv("D:/New folder/supermarket3.csv", header = T, colClasses =  
"factor")
```

```
# to get more information about the dataset
```

```
summary(SupMarTra)
```

```
#to find the length of the dataset
```

```
length(SupMarTra)
```

```
#to find the dimensions
```

```
dim(SupMarTra)
```

```
#view the file
```

```
View(SupMarTra)
```

```
#to find the association rules with Aprior
```

```
rules<- apriori(SupMarTra)
```

```
#set support and confidence (support = 3 and confidence = 70%)
```

```
rules<-apriori(SupMarTra, parameter=list( supp = .5, conf = .8))
```

```
inspect(rules)
```

```
# set the maximum and minimum length of rules , the null set will be removed
```

```
rules<- apriori(SupMarTra, parameter=list(minlen = 2, maxlen = 5, supp=.5, conf=.8))
```

```
inspect(rules)
```

```
#remove all = NO because it won't give any sense
```

```
rules<- apriori(SupMarTra, parameter=list(minlen = 2, maxlen = 5, supp=.5, conf=.8),  
appearance = list(none=c("I1=NO", "I2=NO", "I3=NO", "I4=NO", "I5=NO")))
```

```
inspect(rules)
```

```
#writing rules to CSV files
```

```
write(rules, file = "D:/data3.CSV", sep = ",")
```

```
#plotting the graph
```

```
library(arulesViz)
```

```
plot(rules)
```

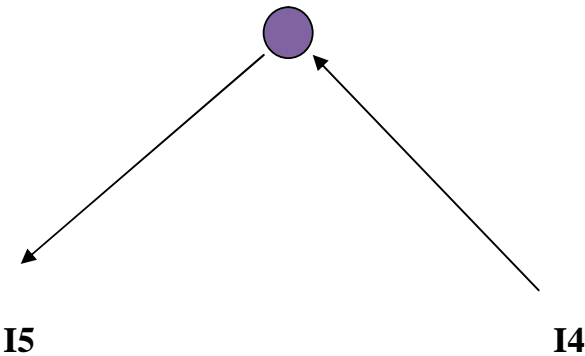
```
plot(rules, method = "grouped")
```

```
plot(rules, method = "graph", control = list(type="items"))
```

Output:

lhs	rhs	support	confidence	Lift	count
{I4=YES}	=> {I5=YES}	0.500000	1.00000	1.5	3

Graph



Lab 9:

K NEAREST NEIGHBORS

Theory

K Nearest Neighbor is a Supervised Machine Learning algorithm that classifies a new data point into the target class, depending on the features of its neighboring data points.

K-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically. Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

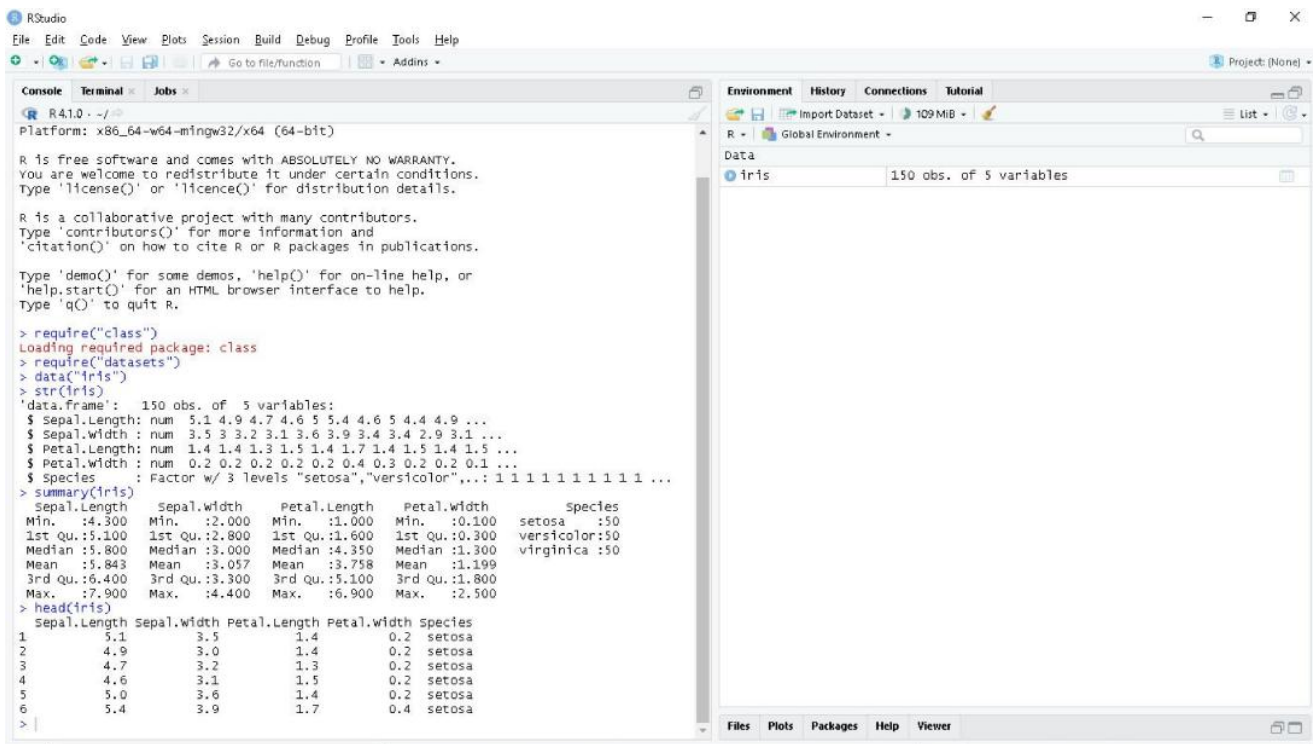
For example, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for *k*-NN classification) or the object property value (for *k*-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

STEP 1: LOAD AND VIEW DATASET

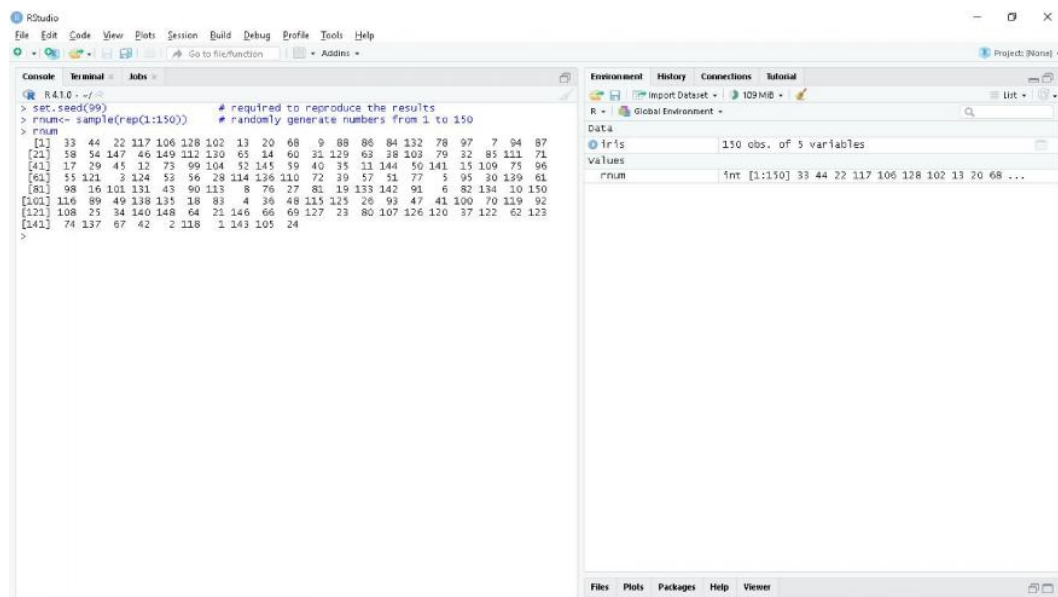
For this example we will load a pre existing IRIS dataset.

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). These measures were used to create a linear discriminant model to classify the species



STEP 2 PREPROCESS THE DATA

Since classification is a type of Supervised Learning, we would require two sets of data i.e. Training and Testing Data (generally in 80:20 ratio). We would load Iris Dataset which is available in RStudio by default and then divide the dataset into two subsets. Our knn classification model would then be trained using subsetiris.train and tested using iris.test. Since the iris dataset is sorted by “Species” by default, we will first jumble the data rows and then take subset.



```
R4.1.0 - ~/ -  
> set.seed(99) # required to reproduce the results  
> rnum<- sample(rep(1:150)) # randomly generate numbers from 1 to 150  
> rnum  
[1] 33 44 22 117 106 128 102 13 20 68 9 88 86 84 132 78 97 7 94 87  
[21] 58 54 147 46 149 112 130 65 14 60 31 129 63 38 103 79 32 85 111 71  
[41] 17 29 45 12 73 99 104 52 145 59 40 35 11 144 50 141 15 109 75 96  
[61] 55 121 3 124 53 56 28 114 136 110 72 39 57 51 77 5 95 30 139 61  
[81] 98 16 101 131 43 90 113 8 76 27 81 19 133 142 91 6 82 134 10 150  
[101] 116 89 49 138 135 18 83 4 36 48 115 125 26 93 47 41 100 70 119 92  
[121] 108 25 34 140 148 64 21 146 66 69 127 23 80 107 126 120 37 122 62 123  
[141] 74 137 67 42 2 118 1 143 105 24  
> iris<- iris[rnum,] #randomize "iris" dataset  
> head(iris)  
      Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
63         6.0         2.2         4.0         1.0 versicolor  
12         4.8         3.4         1.6         0.2 setosa  
54         5.5         2.3         4.0         1.3 versicolor  
100        5.7         2.8         4.1         1.3 versicolor  
18         5.1         3.5         1.4         0.3 setosa  
146        6.7         3.0         5.2         2.3 virginica  
>
```

- Since iris dataset contains 50 instances of each example we randomly generate numbers from 1 to 150 and randomize the dataset

```
R4.1.0 - ~/ -  
> set.seed(99) # required to reproduce the results  
> rnum<- sample(rep(1:150)) # randomly generate numbers from 1 to 150  
> rnum  
[1] 33 44 22 117 106 128 102 13 20 68 9 88 86 84 132 78 97 7 94 87  
[21] 58 54 147 46 149 112 130 65 14 60 31 129 63 38 103 79 32 85 111 71  
[41] 17 29 45 12 73 99 104 52 145 59 40 35 11 144 50 141 15 109 75 96  
[61] 55 121 3 124 53 56 28 114 136 110 72 39 57 51 77 5 95 30 139 61  
[81] 98 16 101 131 43 90 113 8 76 27 81 19 133 142 91 6 82 134 10 150  
[101] 116 89 49 138 135 18 83 4 36 48 115 125 26 93 47 41 100 70 119 92  
[121] 108 25 34 140 148 64 21 146 66 69 127 23 80 107 126 120 37 122 62 123  
[141] 74 137 67 42 2 118 1 143 105 24  
> iris<- iris[rnum,] #randomize "iris" dataset  
> head(iris)  
      Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
63         6.0         2.2         4.0         1.0 versicolor  
12         4.8         3.4         1.6         0.2 setosa  
54         5.5         2.3         4.0         1.3 versicolor  
100        5.7         2.8         4.1         1.3 versicolor  
18         5.1         3.5         1.4         0.3 setosa  
146        6.7         3.0         5.2         2.3 virginica  
> # Normalize the dataset between values 0 and 1  
> normalize <- function(x){  
+   return ((x-min(x))/(max(x)-min(x)))  
+ }  
>  
> iris.new<- as.data.frame(lapply(iris[,c(1,2,3,4)],normalize))  
> head(iris.new)  
      Sepal.Length Sepal.Width Petal.Length Petal.Width  
1  0.4722222 0.08333333 0.50847458 0.37500000  
2  0.1388889 0.58333333 0.10169492 0.04166667  
3  0.3333333 0.12500000 0.50847458 0.50000000  
4  0.3888889 0.33333333 0.5254373 0.50000000  
5  0.2222222 0.62500000 0.06779661 0.08333333  
6  0.6666667 0.41666667 0.71186441 0.91666667  
>
```

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
Project: (None)

Console
R4.1.0 ~./
[61] 55 121 3 124 53 56 28 114 136 110 72 39 57 51 77 5 95 30 139 61
[81] 98 16 101 131 43 90 113 8 76 27 81 19 133 142 91 6 82 134 10 150
[101] 116 89 49 138 135 18 83 4 36 48 115 125 26 93 47 41 100 70 119 92
[121] 108 25 34 140 148 64 21 146 66 69 127 23 80 107 126 120 37 122 62 123
[141] 74 137 67 42 2 118 1 143 105 24
> iris<- iris[rnum,] #randomize "iris" dataset
> head(iris)
  Sepal.Length Sepal.width Petal.Length Petal.width Species
63            6.0         2.2         4.0         1.0 versicolor
12            4.8         3.4         1.6         0.2 setosa
54            5.5         2.3         4.0         1.3 versicolor
100           5.7         2.8         4.1         1.3 versicolor
18            5.1         3.5         1.4         0.3 setosa
146           6.7         3.0         5.2         2.3 virginica
> # Normalize the dataset between values 0 and 1
> normalize <- function(x){
+   return ((x-min(x))/(max(x)-min(x)))
+ }
>
> iris.new<- as.data.frame(apply(iris[,c(1,2,3,4)],normalize))
> head(iris.new)
  Sepal.Length Sepal.width Petal.Length Petal.width
1  0.4722222 0.08333333 0.50847458 0.37500000
2  0.1388889 0.58333333 0.10169492 0.04166667
3  0.3333333 0.12500000 0.50847458 0.50000000
4  0.3888889 0.33333333 0.52542373 0.50000000
5  0.2222222 0.62500000 0.06779661 0.08333333
6  0.6666667 0.41666667 0.71186441 0.91666667
> # subset the dataset
> iris.train<- iris.new[1:130,]
> iris.train.target<- iris[1:130,5]
> iris.test<- iris.new[131:150,]
> iris.test.target<- iris[131:150,5]
> summary(iris.new)
  Sepal.Length Sepal.width Petal.Length Petal.width
Min. :0.0000 Min. :0.0000 Min. :0.0000 Min. :0.00000
1st Qu.:0.2222 1st Qu.:0.3333 1st Qu.:0.1017 1st Qu.:0.08333
Median :0.4167 Median :0.4167 Median :0.5678 Median :0.50000
Mean :0.4287 Mean :0.4406 Mean :0.4675 Mean :0.45806
3rd Qu.:0.5833 3rd Qu.:0.5417 3rd Qu.:0.6949 3rd Qu.:0.70833
Max. :1.0000 Max. :1.0000 Max. :1.0000 Max. :1.00000
>

Environment History Connections Tutorial
R - Global Environment
Data
iris 150 obs. of 5 variables
iris.new 150 obs. of 4 variables
iris.test 20 obs. of 4 variables
iris.train 130 obs. of 4 variables
Values
iris.test.target Factor w/ 3 levels "setosa","versicolor",...: 1 3 2...
iris.train.target Factor w/ 3 levels "setosa","versicolor",...: 2 1 2...
rnum int [1:150] 33 44 22 117 106 128 102 13 20 68 ...
Functions
normalize function (x)
Files Plots Packages Help Viewer

```

- In the above snapshot we are creating a subset of the iris dataset as the algorithm requires a training set and a test set

STEP 3 APPLY KNN ALGORITHM

In this step we will use `knn()` function which is present in a pre installed package in Rstudio called "class"

`knn()` function is used to train a model The `knn()` function identifies the k- nearest neighbors using Euclidean distance where k is a user-specified number

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
Project: (None)

Console
R4.1.0 ~./
> model1<- knn(train=iris.train, test=iris.test, cl=iris.train.target, k=16) #model1
>
>

Environment History Connections Tutorial
R - Global Environment
Data
iris 150 obs. of 5 variables
iris.new 150 obs. of 4 variables
iris.test 20 obs. of 4 variables
iris.train 130 obs. of 4 variables
Values
iris.test.target Factor w/ 3 levels "setosa","versicolor",...: 1 3 2...
iris.train.target Factor w/ 3 levels "setosa","versicolor",...: 2 1 2...
model1 Factor w/ 3 levels "setosa","versicolor",...: 1 3 2...
rnum int [1:150] 33 44 22 117 106 128 102 13 20 68 ...
Functions
normalize function (x)
Files Plots Packages Help Viewer

```

STEP4 EXECUTE AND VERIFY RESULTS

The screenshot shows the RStudio interface with the following content:

Console:

```
R41.0.~ />
> model1<- knn(train=iris.train, test=iris.test, cl=iris.train.target, k=16)
> #model1
> table(iris.test.target, model1)
      model1
iris.test.target setosa versicolor virginica
      setosa      6          0          0
      versicolor  0          4          0
      virginica   0          2          8
>
```

Environment:

Object	Size
iris	150 obs. of 5 variables
iris.new	150 obs. of 4 variables
iris.test	20 obs. of 4 variables
iris.train	130 obs. of 4 variables

Values:

Object	Value
iris.test.target	Factor w/ 3 levels "setosa","versicolor",...: 3 1 2...
iris.train.target	Factor w/ 3 levels "setosa","versicolor",...: 1 1 1...
model1	Factor w/ 3 levels "setosa","versicolor",...: 3 1 2...
rnum	int [1:150] 33 44 22 117 106 128 102 13 20 68 ...

Functions:

Function	Value
normalize	function (x)

The values on the diagonal shows number of correctly classified instances out of total 153 instances. The value not on the diagonal implies that they have been incorrectly instances. Hence, there is a scope of further improvement in classifiermodel. Improvement may be done in terms of trying different values of “k” and choosing the one with maximum accuracy.

Lab 10:

Simple Linear Regression

Theory

It is a commonly used type of predictive analysis. It is a statistical approach for modelling relationship between a dependent variable and a given set of independent variables.

It is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables. One variable denoted x is regarded as an independent variable and other one denoted y is regarded as a dependent variable. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

In a nutshell, this technique finds a line that best “fits” the data and takes on the following form:

$$\hat{y} = b_0 + b_1x$$

Where:

- \hat{y} : The estimated response value
- b_0 : The intercept of the regression line
- b_1 : The slope of the regression line

This equation can help us understand the relationship between the explanatory and response variable, and (assuming it's statistically significant) it can be used to predict the value of a response variable given the value of the explanatory variable.

Step 1: Load the Data

For this example, we'll create a fake dataset that contains the following two variables for 15 students:

- Total hours studied for some exam
- Exam score

We'll attempt to fit a simple linear regression model using *hours* as the explanatory variable and *exam score* as the response variable.

The following code shows how to create this

```
fake dataset in R: #create dataset
df <- data.frame(hours=c(1, 2, 4, 5, 5, 6, 6, 7, 8, 10, 11, 11, 12, 12, 14),
                  score=c(64, 66, 76, 73, 74, 81, 83, 82, 80, 88, 84, 82, 91, 93, 89))
```

```
#view first six rows
```

```
of datasethead(df)
```

```
hours
```

```
score 1
```

```
1 64
```

```
2 2 66
```

```
3 4 76
```

```
4 5 73
```

```
5 5 74
```

```
6 6 81
```

#attach dataset to make it more

convenient to work with `attach(df)`

Step 2: Visualize the Data

Before we fit a simple linear regression model, we should first visualize the data to gain an understanding of it.

First, we want to make sure that the relationship between *hours* and *score* is roughly linear, since that is a massive underlying assumption of simple linear regression. We can create a simple scatter plot to view the relationship between the two variables:

`scatter.smooth(hours, score, main='Hours studied vs. Exam Score')`

```
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.
```

```
[Workspace loaded from ~/.RData]
```

```
> #create dataset  
> df <- data.frame(hours=c(1, 2, 4, 5, 5, 6, 8, 7, 8, 10, 11, 11, 12, 12, 14),  
+                  score=c(64, 66, 76, 73, 74, 81, 83, 82, 80, 88, 84, 82, 91, 93, 89))
```

```
> #view first six rows
```

```
> head(df)
```

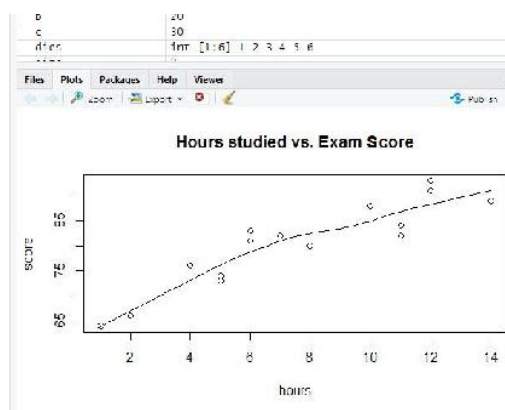
```
  hours score  
1     1    64  
2     2    66  
3     4    76  
4     5    73  
5     5    74  
6     6    81
```

```
> #attach dataset to make it easier to work with  
> attach(df)
```

```
> #create a scatterplot to check for linear relationship
```

```
> scatter.smooth(hours, score, main='Hours studied vs. Exam Score')
```

```
> |
```



From the plot we can see that the relationship does appear to be linear. As *hours* increases, *score* tends to increase as well in a linear fashion.

Next, we can create a boxplot to visualize the distribution of exam scores and check for [outliers](#). By default, R defines an observation to be an outlier if it is 1.5 times the interquartile range greater than the third quartile (Q3) or 1.5 times the interquartile range less than the first quartile (Q1).

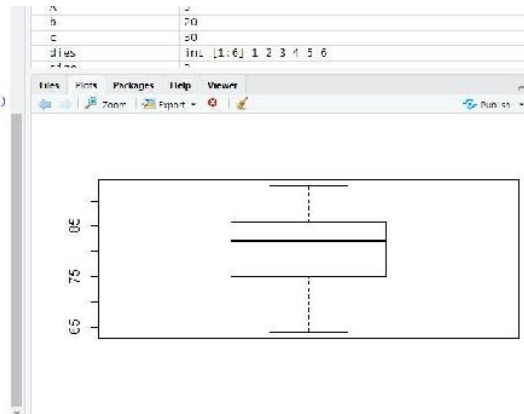
If an observation is an outlier, a tiny circle will

appear in the boxplot: `boxplot(score)`

```

> #attach dataset to make it easier to work with
> attach(df)
>
> #create a scatterplot to check for linear relationship
> scatter.smooth(hours, score, main="Hours studied vs. Exam Score")
> #create subset
> df <- data.frame(hours=c(1, 2, 4, 5, 5, 6, 6, 7, 8, 10, 11, 11, 12, 12, 14),
+                 score=c(64, 66, 76, 73, 74, 81, 83, 82, 80, 88, 84, 82, 91, 93, 89))
>
> #view first six rows
> head(df)
  hours score
1      1    64
2      2    66
3      4    76
4      5    73
5      5    74
6      6    81
>
> #attach dataset to make it easier to work with
> attach(df)
The following objects are masked from df (pos = 3):
    hours, score
>
> #create a scatterplot to check for linear relationship
> scatter.smooth(hours, score, main="Hours studied vs. Exam Score")
>
> #create a boxplot to check for outliers
> boxplot(score)
>

```



There are no tiny circles in the boxplot, which means there are no outliers in our dataset.

Step 3: Perform Simple Linear Regression

Once we've confirmed that the relationship between our variables is linear and that there are no outliers present, we can proceed to fit a simple linear regression model using *hours* as the explanatory variable and *score* as the response variable:

```
#fit simple linear  
regression model  
model <- lm(score~hours)
```

```
#view model  
summary  
summary(model)
```

Call:
lm(formula = score ~ hours)

Residuals:
Min 1Q Median 3Q Max
-5.140 -3.219 -1.193 2.816 5.772

Coefficients:
Estimate Std. Error t value
Pr(>|t|) (Intercept) 65.334 2.106
31.023 1.41e-13 ***
hours 1.982 0.248 7.995 2.25e-06 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.641 on 13
degrees of freedom Multiple R-squared:
0.831, Adjusted R-
squared: 0.818
F-statistic: 63.91 on 1 and 13 DF, p-value: 2.253e-06
From the model summary we can see that the fitted regression equation is:

$$\text{Score} = 65.334 + 1.982 * (\text{hours})$$

This means that each additional hour studied is associated with an average increase in exam score of **1.982** points. And the intercept value of **65.334** tells us the average expected exam score for a student who studies zero hours.

We can also use this equation to find the expected exam score based on the number of hours that a student studies. For example, a student who

studies for 10 hours is expected to receive an exam score of **85.15**:

$$\text{Score} = 65.334 + 1.982*(10) = 85.15$$

Step 4: Create Residual Plots

After we've fit the simple linear regression model to the data, the last step is to create residual plots.

One of the key assumptions of linear regression is that the residuals of a regression model are roughly normally distributed and are homoscedastic at each level of the explanatory variable. If these assumptions are violated, then the results of our regression model could be misleading or unreliable.

To verify that these assumptions are met, we can create the following residual plots:

Residual vs. fitted values plot: This plot is useful for confirming homoscedasticity. The x-axis displays the fitted values and the y-axis displays the residuals. As long as the residuals appear to be randomly and evenly distributed throughout the chart around the value zero, we can assume that homoscedasticity is not violated:

```
#define
residuals res
<-
resid(model
)
```

```
#produce residual vs.
fitted plot
plot(fitted(model),
res)
```

```
#add a horizontal
line at 0
abline(0,0)
```

```
>
> #view model summary
> summary(model)

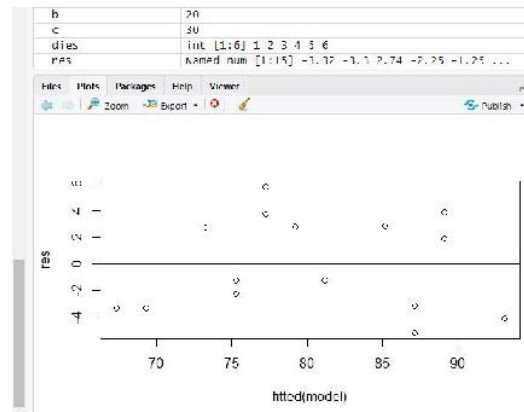
Call:
lm(formula = score ~ hours)

Residuals:
    Min       1Q   median       3Q      Max
-5.110 -3.219 -1.193  2.816  5.772

Coefficients:
(Intercept)    62.224      2.106  51.022  1.41e-13 ***
      hours       1.982      0.748   7.995  2.75e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.641 on 13 degrees of freedom
Multiple R-squared:  0.631,    Adjusted R-squared:  0.618
F-statistic: 63.92 on 1 and 13 DF,  p-value: 2.233e-06

>
> #define residuals of model
> res <- resid(model)
>
> #produce residual vs. fitted plot
> plot(fitted(model), res)
>
> #add a horizontal line at 0
> abline(0,0)
>
```



The residuals appear to be randomly scattered around zero and don't exhibit any noticeable patterns, so this assumption is met.

Q-Q plot: This plot is useful for determining if the residuals follow a normal distribution. If the data values in the plot fall along a roughly straight line at a 45-degree angle, then the data is normally distributed:

```
#create Q-Q plot
for residuals
qqnorm(res)
```

```
#add a straight diagonal
line to the plot qqline(res)
```

The residuals stray from the 45-degree line a bit, but not enough to cause serious concern. We can assume that the normality assumption is met.

Since the residuals are normally distributed and homoscedastic, we've

verified that the assumptions of the simple linear regression model are met. Thus, the output from our model is reliable.

```

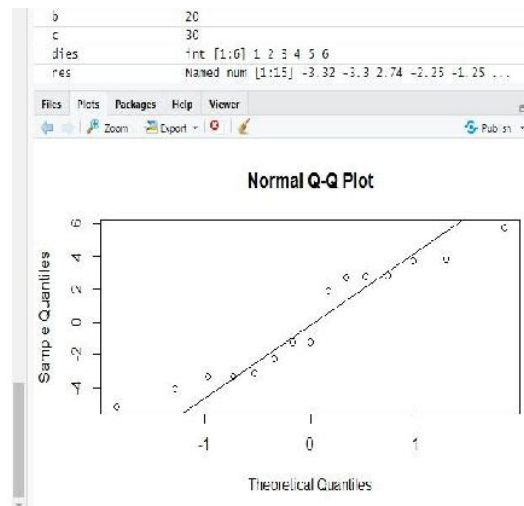
Residuals:
    Min       1Q   Median       3Q      Max
-3.140 -3.219 -2.193  2.816  5.772

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  85.834      7.106   12.077 1.51e-13 ***
hours        1.987       0.124    15.945 7.75e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.641 on 13 degrees of freedom
Multiple R-squared:  0.831,    Adjusted R-squared:  0.818
F-statistic: 63.91 on 1 and 13 DF,  p-value: 2.23e-06

> #define residuals of model
> res <- resid(model)
>
> #produce residual vs. fitted plot
> plot(fitted(model), res)
>
> #add a horizontal line at 0
> abline(0,0)
>
> #create Q-Q plot for residuals
> qqnorm(res)
>
> #add a straight diagonal line to the plot
> qqline(res)
>

```



Complete R code for linear regression:

#create dataset

```
df <- data.frame(hours=c(1, 2, 4, 5, 5, 6, 6, 7, 8, 10, 11, 11, 12, 12, 14),
                 score=c(64, 66, 76, 73, 74, 81, 83, 82, 80, 88, 84, 82, 91, 93, 89))
```

#view first six rows

```
head(df)
```

```
#attach dataset to make it
easier to work with attach(df)
```

```
#create a scatterplot to check for linear
relationship scatter.smooth(hours, score,
main='Hours studied vs. Exam Score')
```

```
#create a boxplot to
check for outliers
boxplot(score)
```

```
#fit simple linear
regression model model
<- lm(score~hours)
```

```
#view model  
summary  
summary(mod  
el)
```

```
#define residuals  
of modelres <-  
resid(model)
```

```
#produce residual vs.  
fitted plot  
plot(fitted(model),  
res)
```

```
#add a horizontal  
line at 0  
abline(0,0)
```

```
#create Q-Q plot  
for residuals  
qqnorm(res)
```

```
#add a straight diagonal  
line to the plotqqline(res)
```



shutterstock.com · 1153070891