E:\JNEC\BE\Sem1\DS\09-07-24.R

```r
titanic = as.data.frame(Titanic)
str(titanic)
mean(cars$speed)
median(cars$speed)
mean(cars$dist)
median(cars$dist)
mode(cars$speed)

table(cars$speed)
which.max(table(cars$speed))


Mode= function(x){
  n = table(x)
  n[which.max(n)]
}

Mode(cars$speed)
Mode(cars$dist)

sum(cars$dist)


var(cars$speed)
var(cars$dist)

sqrt(var(cars$speed))
sd(cars$speed)

sd(cars$dist)

range(cars$speed)
range(cars$dist)


hist(cars$speed)
lines(cars$speed)


hist(cars$dist)

y = boxplot(cars$dist)
y$out


plot(density(cars$speed))
plot(density((cars$dist)))

```

```r
skewness(cars$dist)

skewness(cars$speed)

kurtosis(cars$speed)
kurtosis(cars$dist)


qqnorm(cars$speed)
qqline(cars$speed)

qqnorm(cars$dist)
qqline(cars$dist)

qqnorm(log(cars$dist))
qqline(log(cars$dist))

```

Practical 3

Aim : Program for basic statistics (mean median mode variance standard deviation range and summary)

Output :

View(cars) :

	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10
7	10	18

Mean :

```
> mean(cars$speed)
[1] 15.4
> mean(cars$dist)
[1] 42.98
```

Median :

```
> median(cars$speed)
[1] 15
> median(cars$dist)
[1] 36
```

Mode :

```
#user defined function for mode :
Mode <- function(x){
  table(which.max(table(x)))
}

Mode(cars$dist)
Mode(cars$speed)
```

```
> Mode(cars$dist)

11
 1
> Mode(cars$speed)

15
 1
>
```

Sum :

```
> sum(cars$dist)
[1] 2149
> sum(cars$speed)
[1] 770
```

Summary :

```
> summary(cars)
     speed           dist
 Min.   : 4.0   Min.   :  2.00
 1st Qu.:12.0   1st Qu.: 26.00
 Median :15.0   Median : 36.00
 Mean   :15.4   Mean   : 42.98
 3rd Qu.:19.0   3rd Qu.: 56.00
 Max.   :25.0   Max.   :120.00
> summary(cars$speed)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    4.0    12.0    15.0    15.4    19.0    25.0
```

Variance :

```
> var(cars$speed)
[1] 27.95918
> var(cars$dist)
[1] 664.0608
```

Standard Deviation :

```
> sd(cars$speed)
[1] 5.287644
> sd(cars$dist)
[1] 25.76938
```

```
> sqrt(var(cars$speed))
[1] 5.287644
> sqrt(var(cars$dist))
[1] 25.76938
```

Range :

```
> range(cars$speed)
[1]  4 25
> range(cars$dist)
[1]   2 120
```

Skewness and Kurtosis :

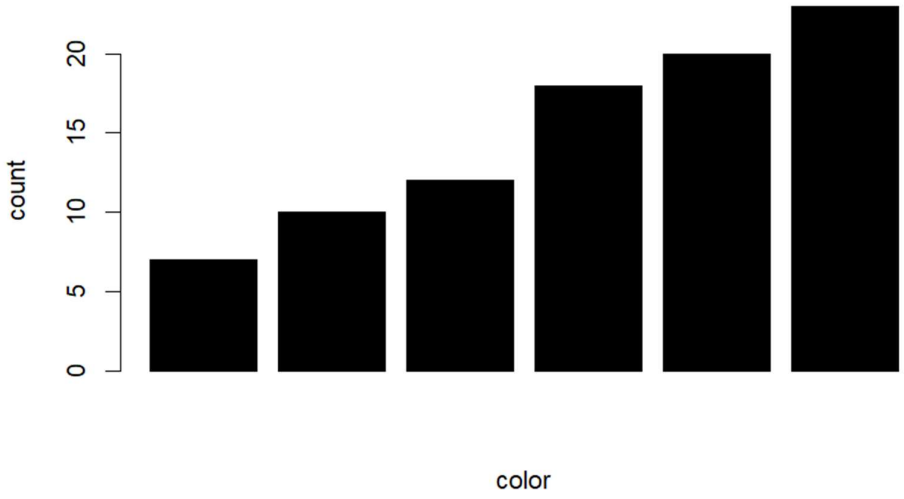
```
> skewness(cars$dist)
[1] 0.7824835
> skewness(cars$speed)
[1] -0.1139548
>
> kurtosis(cars$speed)
[1] 2.422853
> kurtosis(cars$dist)
[1] 3.248019
```

Data Visualization

#1. barplot 2. Linechart 3. Dot plot 4. Pie chart, 3D pie 5. Histogram 6. boxplot

#Barplot

```
data<-c(7,10,12,18,20,23)
barplot
barplot(data,xlab ="color",ylab = "count",col="Black")
labels<-c("A","B","C","D","E","f")
```



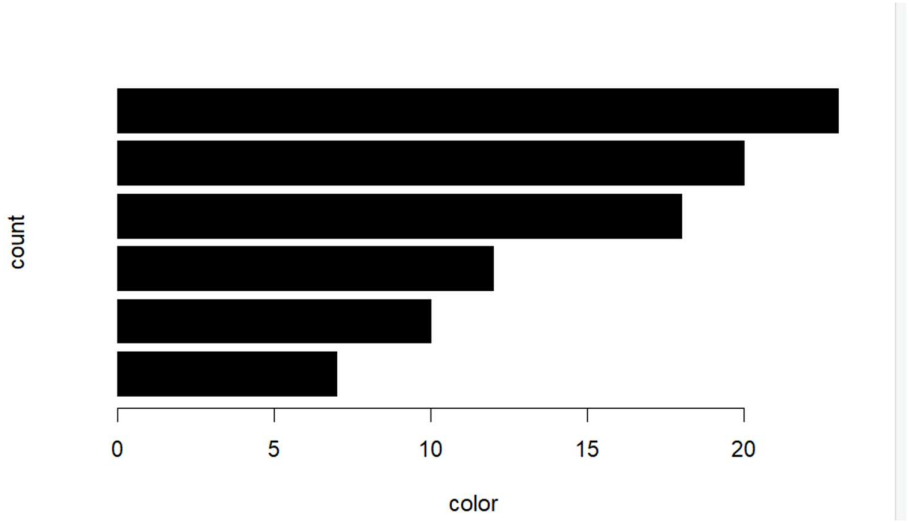
#for horizontal graph

data<-c(7,10,12,18,20,23)

barplot

barplot(data,horiz=TRUE,xlab ="color",ylab = "count",col="Black")

labels<-c("A","B","C","D","E","f")



#Linechart

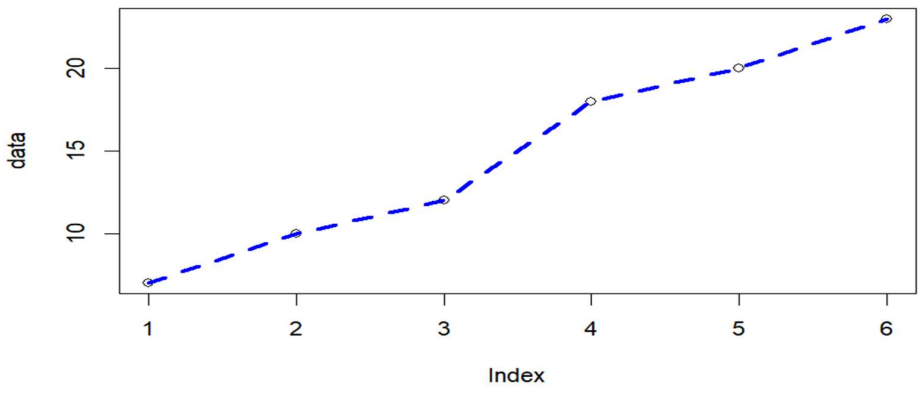
dotchart(data)

?dotchart#for help

plot(data)

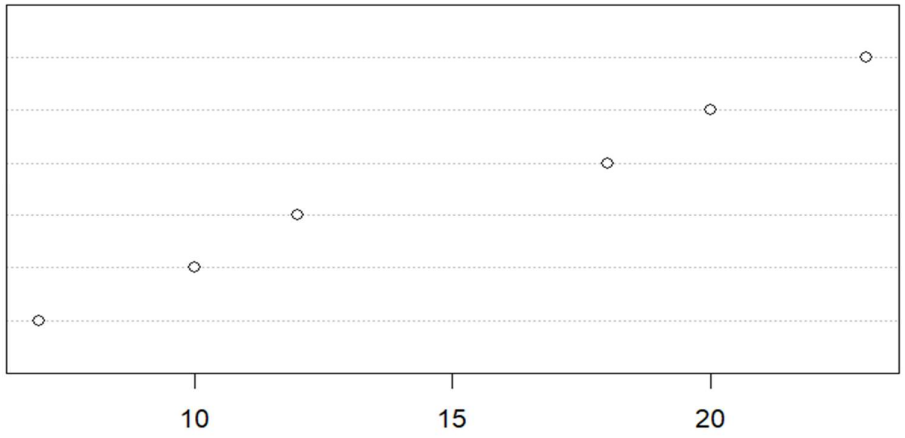
lines(data,col="blue",lwd=3,lty=2)#line type=lty,line width=lwd

?lines



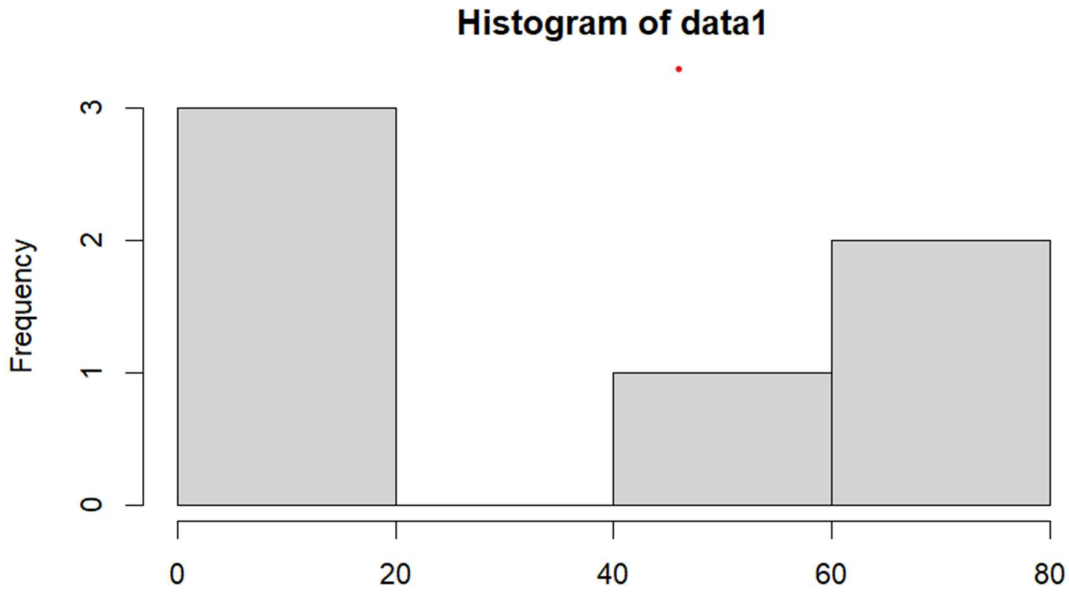
#Dotplot

```
plot(data,col='green',ylim=range(c(data,data1)))
lines(data,col='red')
dotchart(data)
```



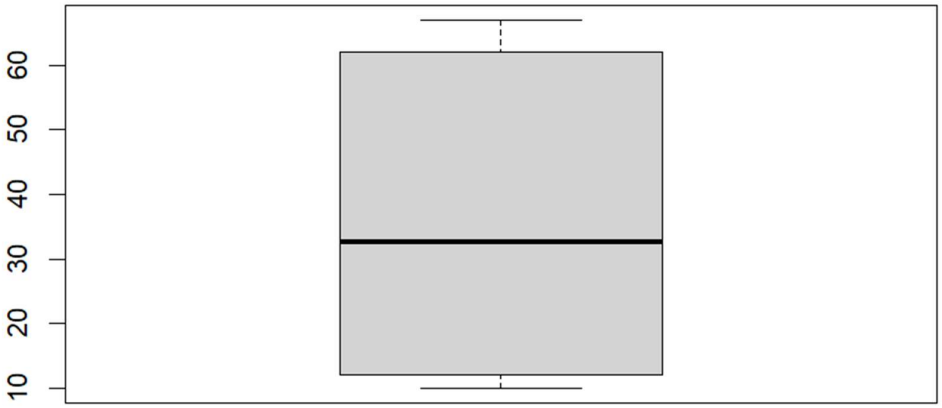
#histogram

```
data1<-c(10,20,45,62,12,67)
hist(data1)
```



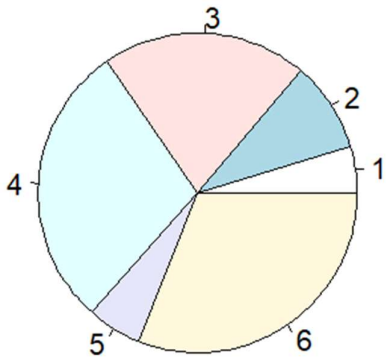
#boxplot

```
data1<-c(10,20,45,62,12,67)
boxplot(data1)
```

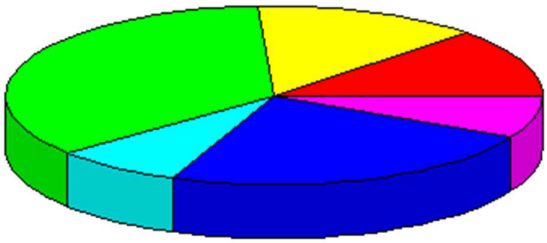


#piechart

```
data1<-c(10,20,45,62,12,67)
pie(data1)
```



```
#3D piechart
install.packages("plotrix")
library(plotrix)
data <- c(19, 21, 54, 12, 36, 12)
pie3D(data)
```



Practical 5

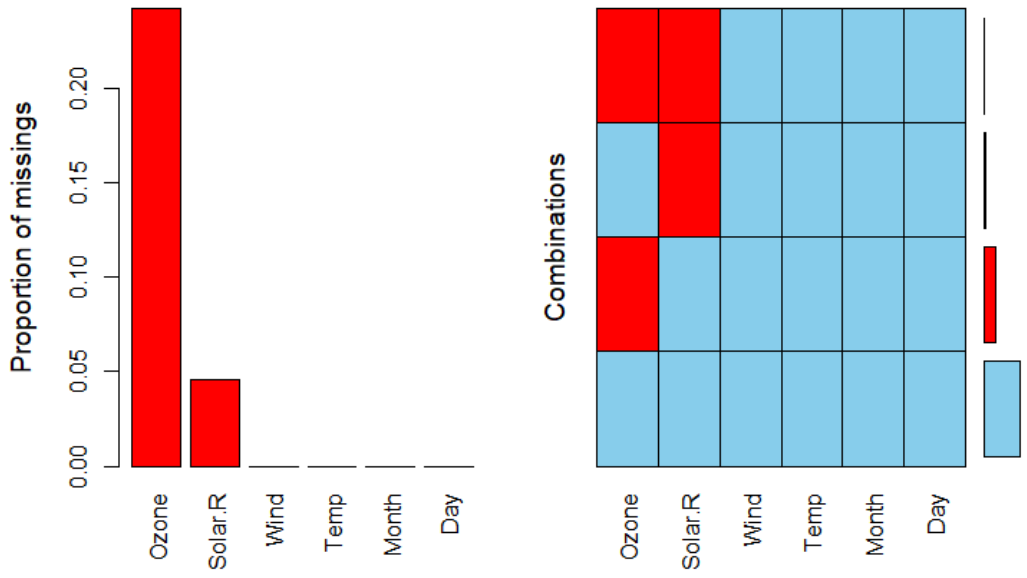
Aim : Implementation of data preprocessing/ Exploratory Data Analysis

CODE :

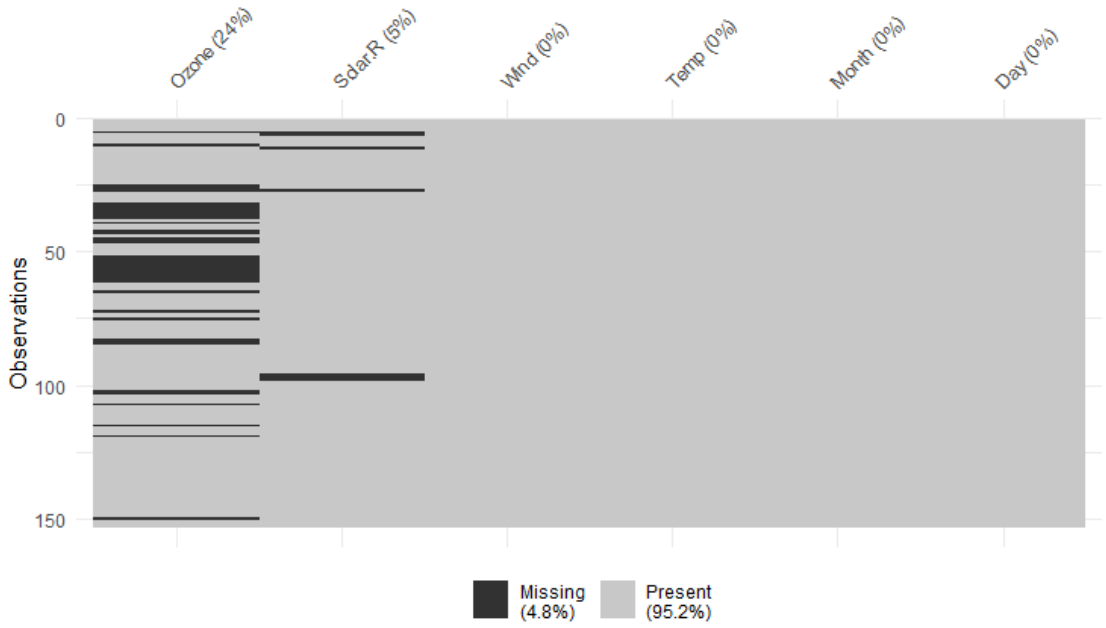
```
# Step 1: Loading the Dataset

data("airquality")  # Load the 'airquality' dataset
df <- as.data.frame(airquality)  # Convert it to a dataframe
View(df)

# Checking the dimensions and structure of the data

dim(df)  # Check the number of rows and columns
str(df)  # View the structure of the dataset
summary(df)  # Summary statistics of the dataset

# Step 2: Missing Values

# Identify missing values
sum(is.na(df))  # Total number of missing values in the dataset
sum(is.na(df$Ozone))  # Missing values in a specific column
```

```
> sum(is.na(df))  # Total number of missing values in the dataset
[1] 44
> sum(is.na(df$Ozone))  # Missing values in a specific column
[1] 37
```

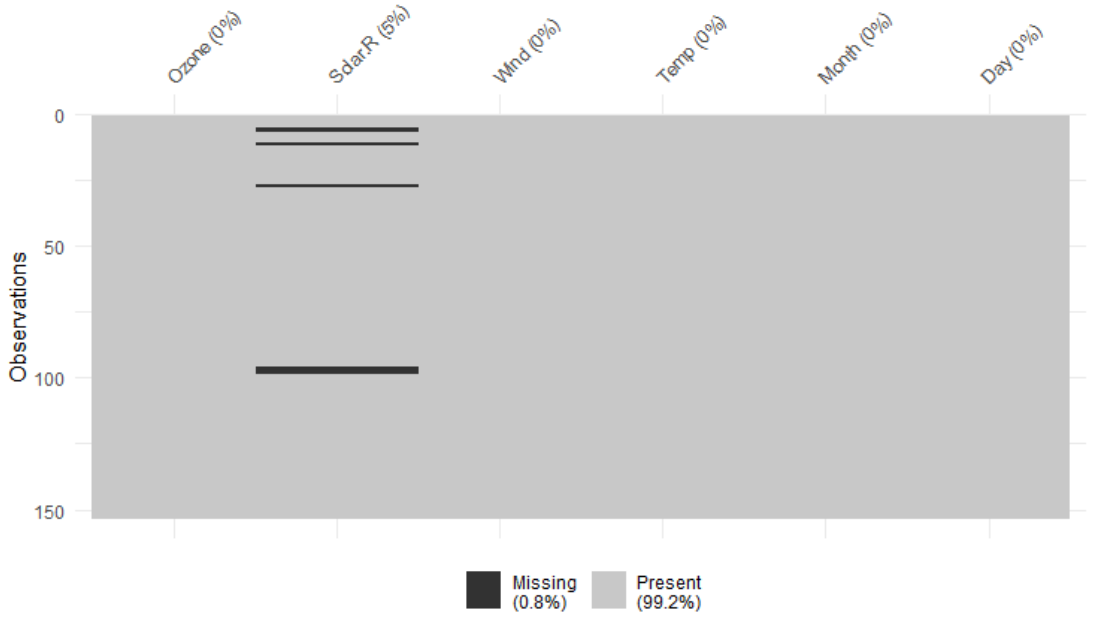
```
# Visualize missing values

library(VIM)
aggr(df)  # Missing value aggregation plot
```



```
library(visdat)
vis_miss(df)  # Visualizing missing values using visdat
```

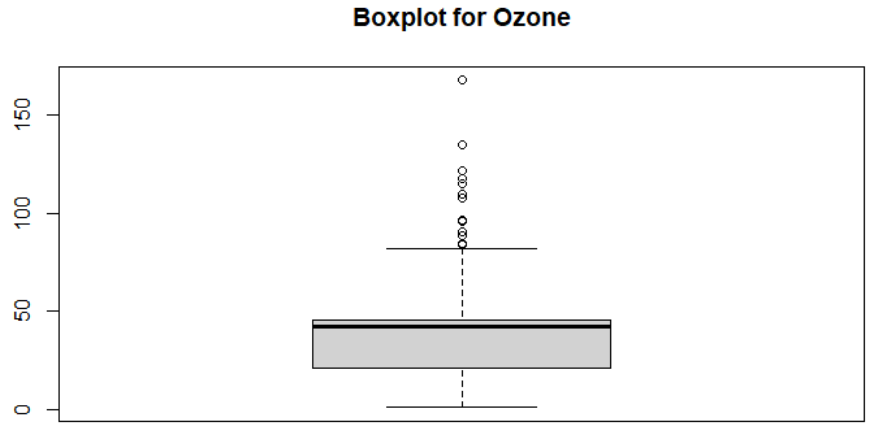



```
# Handling missing values by imputing with mean
mean_ozone <- mean(df$Ozone, na.rm = TRUE)
df$Ozone[is.na(df$Ozone)] <- mean_ozone  # Replace missing values
with mean
```

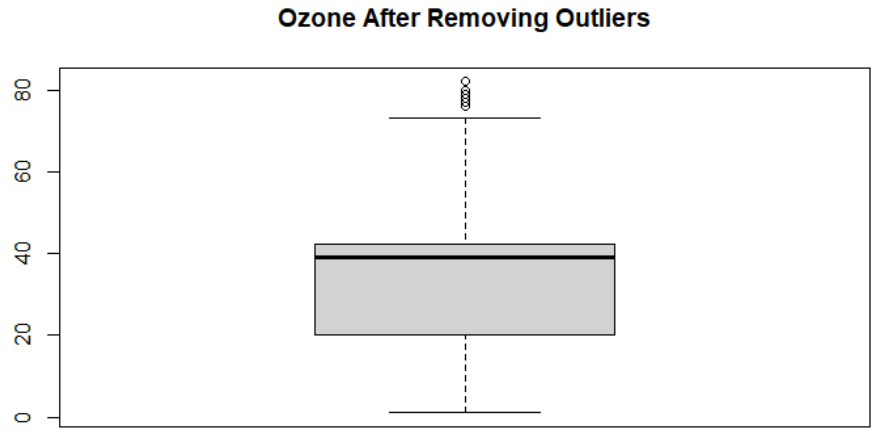


 # Step 3: Outlier Detection and Handling

```
# Boxplot to detect outliers
boxplot(df$Ozone, main = "Boxplot for Ozone")
```




```
# Detecting outliers using the IQR method

Q1 <- quantile(df$Ozone, 0.25)
Q3 <- quantile(df$Ozone, 0.75)
IQR_value <- Q3 - Q1
lower_bound <- Q1 - 1.5 * IQR_value
upper_bound <- Q3 + 1.5 * IQR_value

# Identifying rows with outliers

outliers <- df$Ozone < lower_bound | df$Ozone > upper_bound
df_with_no_outliers <- df[!outliers, ]  # Removing outliers
boxplot(df_with_no_outliers$Ozone, main = "Ozone After Removing
Outliers")
```



```
# Winsorization for Outlier Handling

library(DescTools)
df$Ozone <- Winsorize(df$Ozone, probs = c(0.05, 0.95))  # Winsorizing
data to handle extreme outliers
```

 # Step 4: Feature Encoding

```
# Load 'Salaries' dataset for feature encoding example
View(Salaries)
```

	Age	Gender	Education.Level	Job.Title	Years.of.Experience	Salary
1	32	Male	Bachelor's	Software Engineer	5	90000
2	28	Female	Master's	Data Analyst	3	65000
3	45	Male	PhD	Senior Manager	15	150000
4	36	Female	Bachelor's	Sales Associate	7	60000

```
# Label encoding using factor()
Salaries$Gender <- as.numeric(factor(Salaries$Gender))
```

	Age	Gender	Education.Level	Job.Title	Years.of.Experience	Salary
1	32	3	Bachelor's	Software Engineer	5.0	90000
2	28	2	Master's	Data Analyst	3.0	65000
3	45	3	PhD	Senior Manager	15.0	150000
4	36	2	Bachelor's	Sales Associate	7.0	60000

```
# One-hot encoding
one_hot <- model.matrix(~Salaries$Gender - 1)
Salaries <- cbind(Salaries, one_hot)  # Combine the one-hot encoding
with the original data
View(Salaries)
```

 # Step 5: Standardization (Z-score normalization)

```
# Standardize using manual calculation
View(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1

```
mean_disp <- mean(mtcars$disp)
sd_disp <- sd(mtcars$disp)
mtcars$std_disp <- (mtcars$disp - mean_disp) / sd_disp  #
Standardized disp
View(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	std_disp
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	-0.57061982
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	-0.57061982
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	-0.99018209
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	0.22009369

```
# Standardization using scale function

mtcars_scaled <- scale(mtcars[, 2:ncol(mtcars)])
View(mtcars_scaled)
```

	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	std_disp
Mazda RX4	-0.1049878	-0.57061982	-0.53509284	0.56751369	-0.610399567	-0.77716515	-0.8680278	1.1899014	0.4235542	0.7352031	-0.57061982
Mazda RX4 Wag	-0.1049878	-0.57061982	-0.53509284	0.56751369	-0.349785269	-0.46378082	-0.8680278	1.1899014	0.4235542	0.7352031	-0.57061982
Datsun 710	-1.2248578	-0.99018209	-0.78304046	0.47399959	-0.917004624	0.42600682	1.1160357	1.1899014	0.4235542	-1.1221521	-0.99018209
Hornet 4 Drive	-0.1049878	0.22009369	-0.53509284	-0.96611753	-0.002299538	0.89048716	1.1160357	-0.8141431	-0.9318192	-1.1221521	0.22009369

 # Step 6: Normalization (Min-Max scaling)

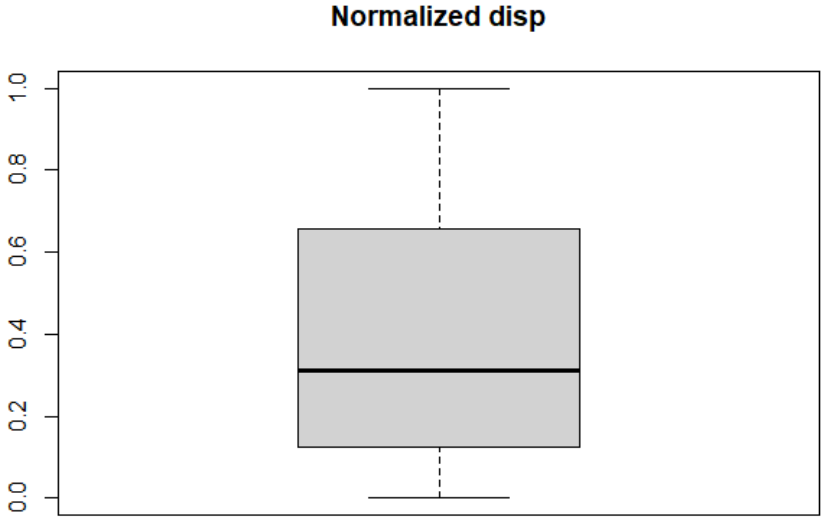
```
# Define normalization function
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply normalization on mtcars dataset
mtcars_normalized <- as.data.frame(lapply(mtcars, normalize))
View(mtcars_normalized)
```

Untitled1* | mtcars_normalized

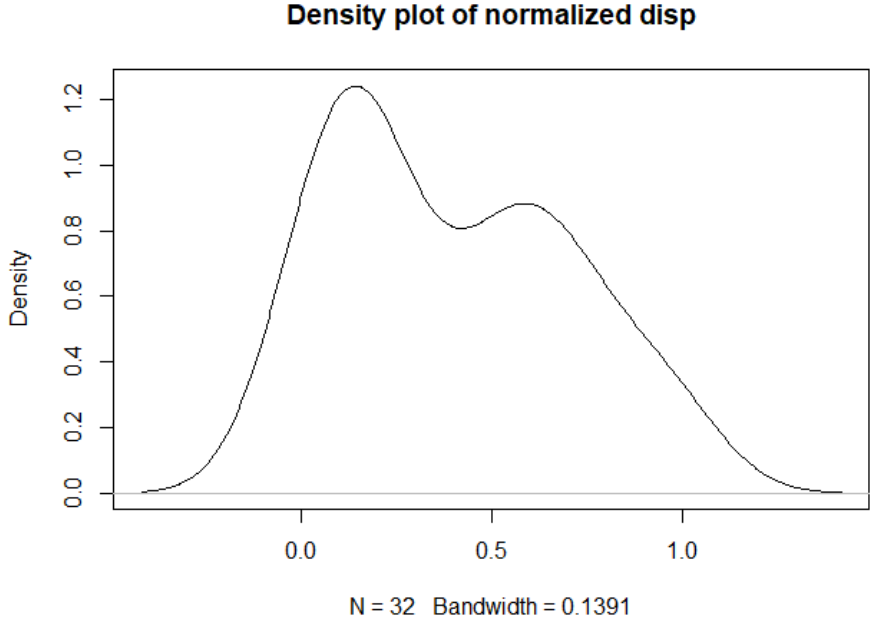
Filter

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	std_disp
1	0.4510638	0.5	0.22175106	0.20494700	0.52534562	0.28304781	0.23333333	0	1	0.5	0.4285714	0.22175106
2	0.4510638	0.5	0.22175106	0.20494700	0.52534562	0.34824853	0.30000000	0	1	0.5	0.4285714	0.22175106
3	0.5276596	0.0	0.09204290	0.14487633	0.50230415	0.20634109	0.48928571	1	1	0.5	0.0000000	0.09204290
4	0.4680851	0.5	0.46620105	0.20494700	0.14746544	0.43518282	0.58809524	1	0	0.0	0.0000000	0.46620105

```
# Step 7: Visualizing the final results
# Boxplot visualization after normalization
boxplot(mtcars_normalized$disp, main = "Normalized disp")
```




```
# Visualize density
plot(density(mtcars_normalized$disp), main = "Density plot of
normalized disp")
```



Practical 6

Aim : Implementation of ANOVA for the given dataset , also calculate the difference of variance between groups

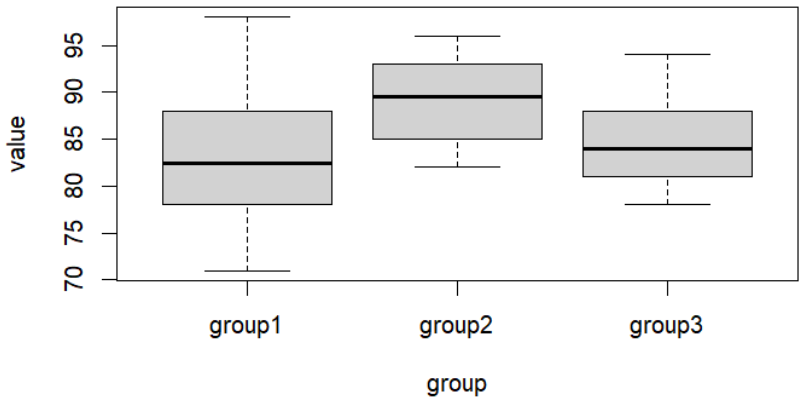
CODE :

```
data <- data.frame(
  group = factor(rep(c("group1", "group2", "group3"), each =
10)),
  value = c(85, 86, 88, 75, 78, 94, 98, 79, 71, 80,
            91, 92, 93, 85, 87, 84, 82, 88, 95, 96,
            79, 78, 88, 94, 92, 85, 83, 85, 82, 81)
)
View(data)
boxplot(value~ group ,data)
anova_result <- aov(value ~ group, data = data)
summary(anova_result)
oneway.test(value ~ group ,
            data = data
            )

data2 <- data.frame(
  group2 = factor(rep(c("x1", "x2", "x3" , "x4"), each = 5)),
  value2 = c(8,10,12,8,7,
            12,11,9,14,4,
            18,12,16,6,8,
            13,9,12,16,15)
)
View(data2)
boxplot(value2~ group2 ,data2)
anova_result2 <- aov(value2 ~ group2, data = data2)
summary(anova_result2)
oneway.test(value2 ~ group2 ,
            data = data2

)
```

DATA :

	group	value
1	group1	85
2	group1	86
3	group1	88
4	group1	75
5	group1	78
6	group1	94
7	group1	98
8	group1	79
9	group1	71
10	group1	80
11	group2	91
12	group2	92
13	group2	93
14	group2	85
15	group2	87
16	group2	84
17	group2	82
18	group2	88
19	group2	95
20	group2	96
21	group3	79
22	group3	78
23	group3	88
24	group3	94
25	group3	92
26	group3	85
27	group3	83
28	group3	85
29	group3	82
30	group3	81

OUTPUT :

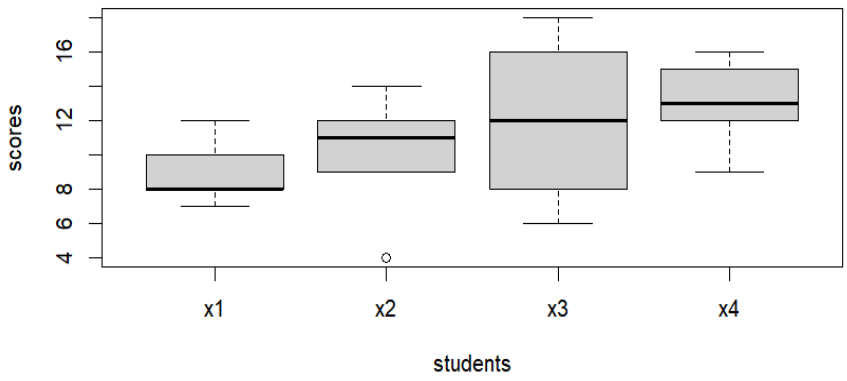



```
R 4.4.1 · ~/
> anova_result <- aov(value ~ group, data = dat
a)
> summary(anova_result)
            Df Sum Sq Mean Sq F value Pr(>F)
group        2  192.2   96.10   2.358  0.114
Residuals   27 1100.6   40.76
> oneway.test(value ~ group ,
+             data = data
+             )

        One-way analysis of means (not assuming
equal variances)

data:  value and group
F = 2.8305, num df = 2.000, denom df = 17.311,
p-value = 0.08639

>
```

DATA :

	students	scores
1	x1	8
2	x1	10
3	x1	12
4	x1	8
5	x1	7
6	x2	12
7	x2	11
8	x2	9
9	x2	14
10	x2	4
11	x3	18
12	x3	12
13	x3	16
14	x3	6
15	x3	8
16	x4	13
17	x4	9
18	x4	12
19	x4	16
20	x4	15

OUTPUT :

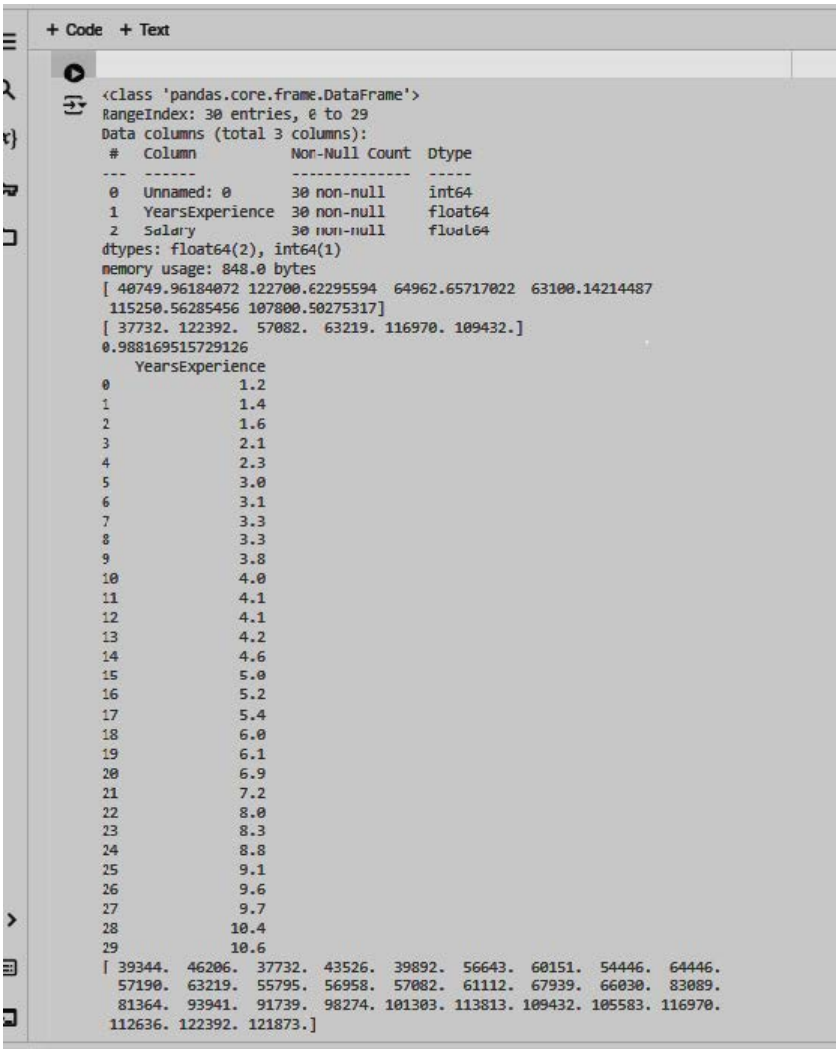



```
R 4.4.1 · ~/
> anova_result2 <- aov(scores ~ students, data = data
2)
> summary(anova_result2)
            Df Sum Sq Mean Sq F value Pr(>F)
students     3     50   16.67   1.282  0.314
Residuals   16    208   13.00
>
> oneway.test(scores ~ students ,
+             data = data2
+ )

        One-way analysis of means (not assuming equal
variances)

data:  scores and students
F = 2.1587, num df = 3.0000, denom df = 8.4774, p-valu
e =
0.1668
```

linearReg in py:

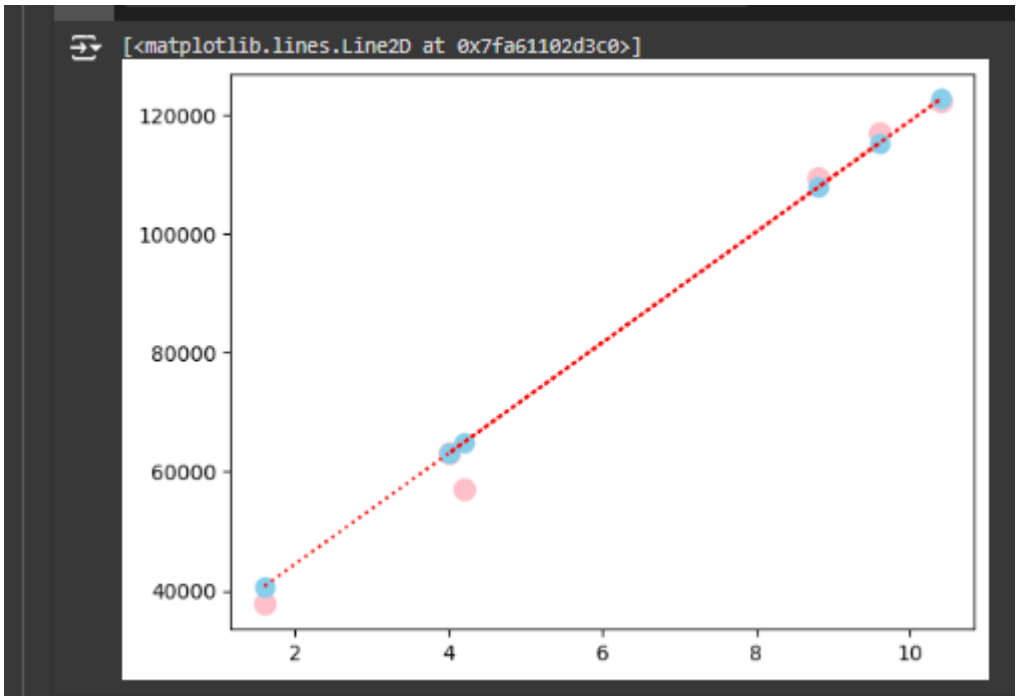
python code linear reg:- import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data = pd.read_csv("/content/Salary_dataset.csv")
data.head()
data.info()
x = data["YearsExperience"]
y = data["Salary"]
#step 3
x = data[["YearsExperience"]]
y = data["Salary"].values
#step 4 - sk learn function
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split (x,y,train_size=0.8,random_state=0)
#step 5 - training model using linear reg & fit-function
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train , y_train)
y_pred= model.predict(x_test)
print(y_pred)
print(y_test)
from sklearn.metrics import r2_score
score = r2_score (y_test , y_pred)
print(score)

Output 1:



Code:- plt.scatter(x_test,y_test,color='pink',s=100)
plt.scatter(x_test,y_pred,color='skyblue',s=80)
plt.plot(x_test,y_pred,color='red',linestyle='dotted')

Output2:



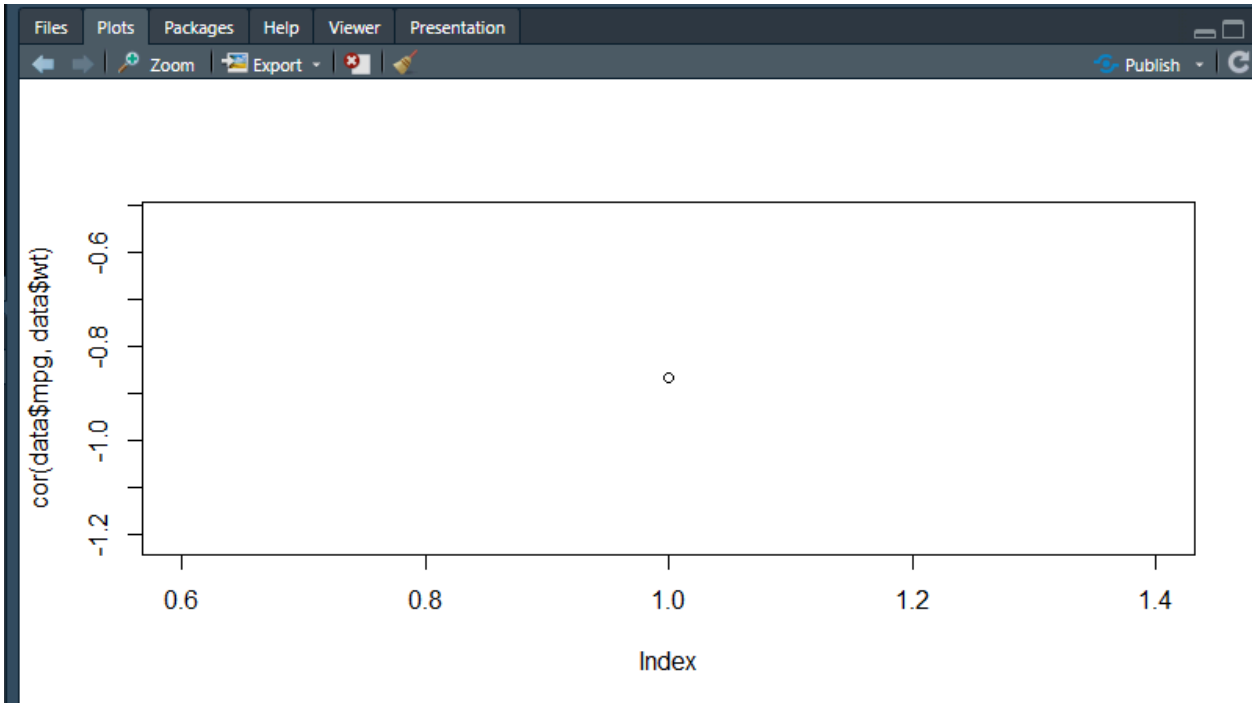
r code-

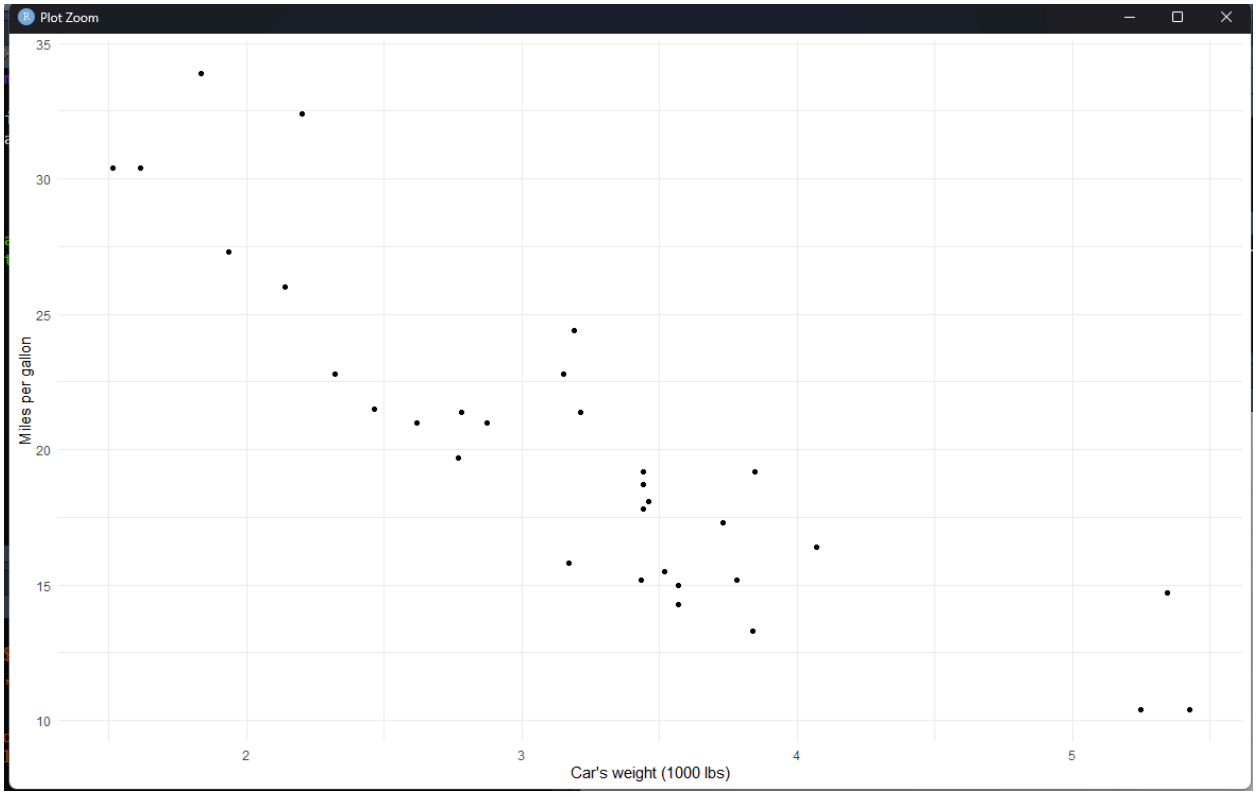
refer statsandr.com website
#for mtcars dataset
data<-mtcars
cor(data$mpg, data$disp)
plot(cor(data$mpg, data$wt))
plot(mtcars)
library(ggplot2)
ggplot(data, aes(x = wt, y = mpg)) +
  geom_point() +
  labs(
    y = "Miles per gallon",
    x = "Car's weight (1000 lbs)"
) +
 theme_minimal()
#simple linear regression
model <- lm(mpg ~ wt, data = data)
summary(model)

# multiple linear regression
model1 <- lm(mpg ~ wt + hp + disp, data = data)
summary(model1)

solution-
mtcars

```
> data<-mtcars
> cor(data$mpg, data$disp)
[1] -0.8475514
> cor(data$mpg, data$wt)
[1] -0.8676594
```





```
+   theme_minimal()
> model <- lm(mpg ~ wt, data = data)
> summary(model)

Call:
lm(formula = mpg ~ wt, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,    Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

```
> model1 <- lm(mpg ~ wt + hp + disp, data = data)
> summary(model1)

Call:
lm(formula = mpg ~ wt + hp + disp, data = data)

Residuals:
   Min     1Q Median     3Q    Max
-3.891 -1.640 -0.172  1.061  5.861

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.105505   2.110815  17.579  < 2e-16 ***
wt          -3.800891   1.066191  -3.565  0.00133 **
hp          -0.031157   0.011436  -2.724  0.01097 *
disp        -0.000937   0.010350  -0.091  0.92851
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.639 on 28 degrees of freedom
Multiple R-squared:  0.8268,    Adjusted R-squared:  0.8083
F-statistic: 44.57 on 3 and 28 DF,  p-value: 8.65e-11

>
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data=pd.read_csv("/content/Titanic-Dataset.csv")
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
				Futrelle, Mrs. Jacques Heath								

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

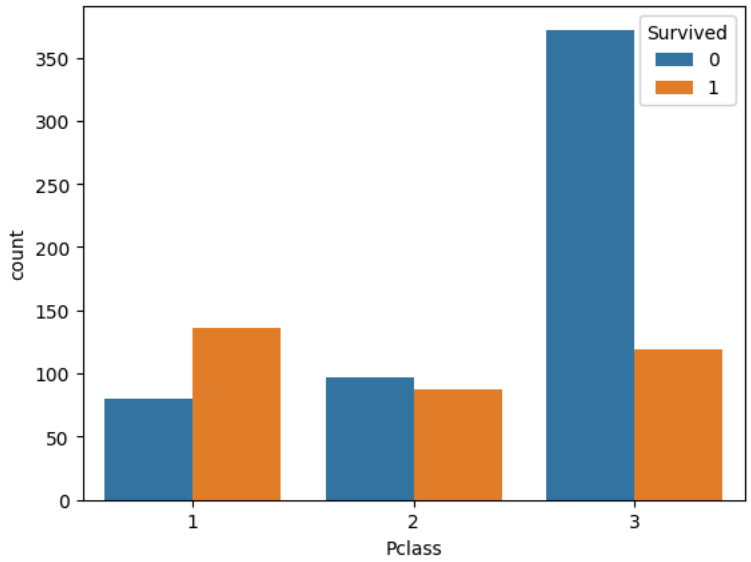
```
data.isnull().sum()
```

	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

dtype: int64

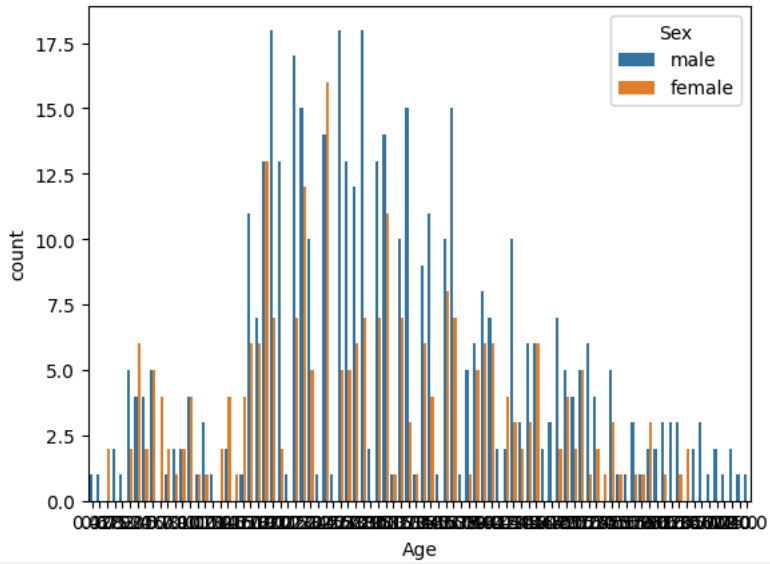
```
sns.countplot(x=data['Pclass'],hue=data['Survived'])
```

```
<Axes: xlabel='Pclass', ylabel='count'>
```




```
sns.countplot(x=data['Age'],hue=data['Sex'])
```

```
<Axes: xlabel='Age', ylabel='count'>
```




```
##replacing null values
data['Age']=data['Age'].fillna(round(data['Age'].mean(),2))
```

```
data.isnull().sum()
```

	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

dtype: int64

```
##droping the column

data.drop(['PassengerId','Name','Fare','Cabin','Embarked','Ticket'],axis=1,inplace=True)


from sklearn.preprocessing import LabelEncoder

encoder=LabelEncoder()
data['Sex']=encoder.fit_transform(data['Sex'])


data.head()
```

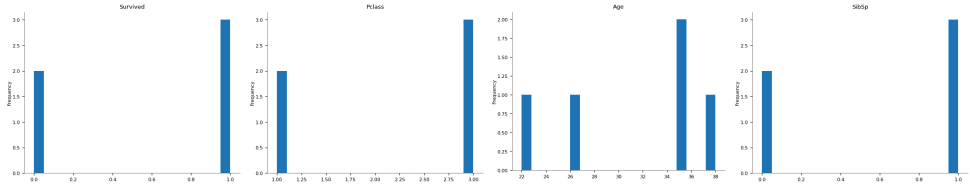
	Survived	Pclass	Sex	Age	SibSp	Parch
0	0	3	1	22.0	1	0
1	1	1	0	38.0	1	0
2	1	3	0	26.0	0	0
3	1	1	0	35.0	1	0
4	0	3	1	35.0	0	0

Next steps: Generate code with data | View recommended plots | New interactive sheet

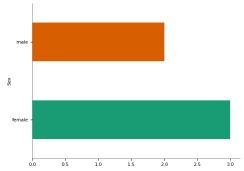
```
data.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch
0	0	3	male	22.0	1	0
1	1	1	female	38.0	1	0
2	1	3	female	26.0	0	0
3	1	1	female	35.0	1	0
4	0	3	male	35.0	0	0

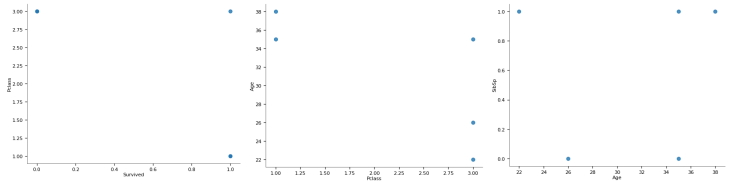
Distributions



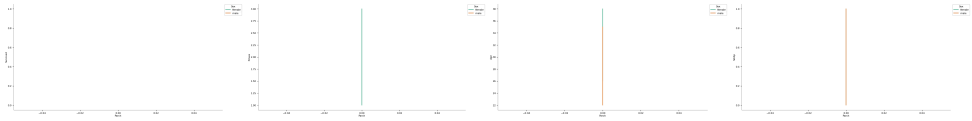
Categorical distributions



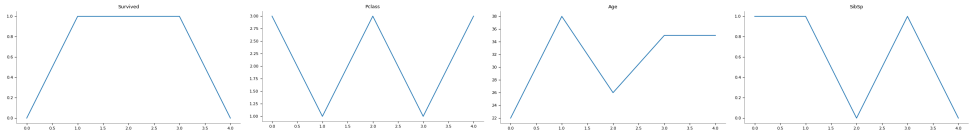
2-d distributions



Time series

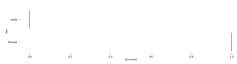


Values

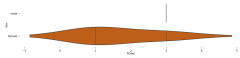


Faceted distributions

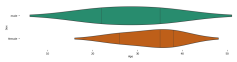
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set
```

Next steps: Generate code with `data` | View recommended plots | New interactive sheet

```
## display data in x and y

y=data['Survived'].values
x=data.drop(['Survived'],axis=1).values



y
```

	Survived
0	0
1	1
2	1
3	1
4	0
...	...
886	0
887	1
888	0
889	1
890	0

891 rows × 1 columns

```
##split the data into
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=0)
```

```
len(x_train)
```

```
712
```

```
len(x_test)
```

```
179
```

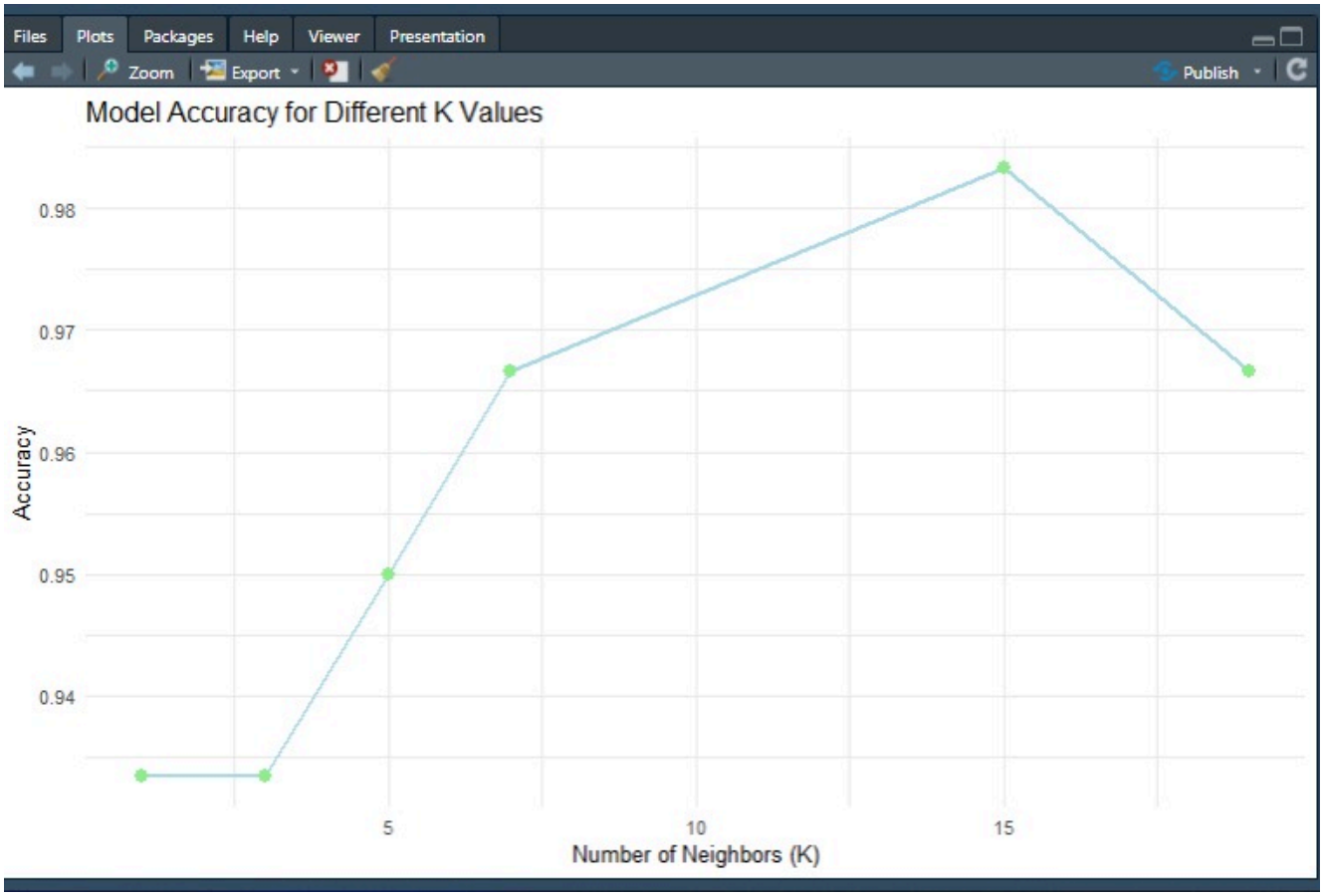
```
len(y_test)
len(y_train)
```

```
data("iris")
View(iris)
str(iris)



install.packages("e1071")
install.packages("caTools")
install.packages("class")



library(e1071)
library(caTools)
library(class)



data(iris)
head(iris)



split <- sample.split(iris, SplitRatio = 0.7)
train_cl <- subset(iris, split == "TRUE")
test_cl <- subset(iris, split == "FALSE")



train_scale <- scale(train_cl[, 1:4])
test_scale <- scale(test_cl[, 1:4])
```

```
head(train_scale)
head(test_scale)



# Fitting KNN Model to training dataset
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 1)
classifier_knn



cm <- table(test_cl$Species, classifier_knn)
cm



misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))

# K = 3
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 3)
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))

# K = 5
```

```
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 5)
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))

# K = 7
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 7)
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))

# K = 15
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 15)
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))

# K = 19
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 19)
```

```
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))

library(ggplot2)

# Data preparation
k_values <- c(1, 3, 5, 7, 15, 19)

# Calculate accuracy for each k value
accuracy_values <- sapply(k_values, function(k) {
  classifier_knn <- knn(train = train_scale,
                        test = test_scale,
                        cl = train_cl$Species,
                        k = k)
  1 - mean(classifier_knn != test_cl$Species)
})

# Create a data frame for plotting
accuracy_data <- data.frame(K = k_values, Accuracy = accuracy_values)

# Plotting
ggplot(accuracy_data, aes(x = K, y = Accuracy)) +
  geom_line(color = "lightblue", size = 1) +
  geom_point(color = "lightgreen", size = 3) +
  labs(title = "Model Accuracy for Different K Values",
       x = "Number of Neighbors (K)",
       y = "Accuracy") +
  theme_minimal()
```

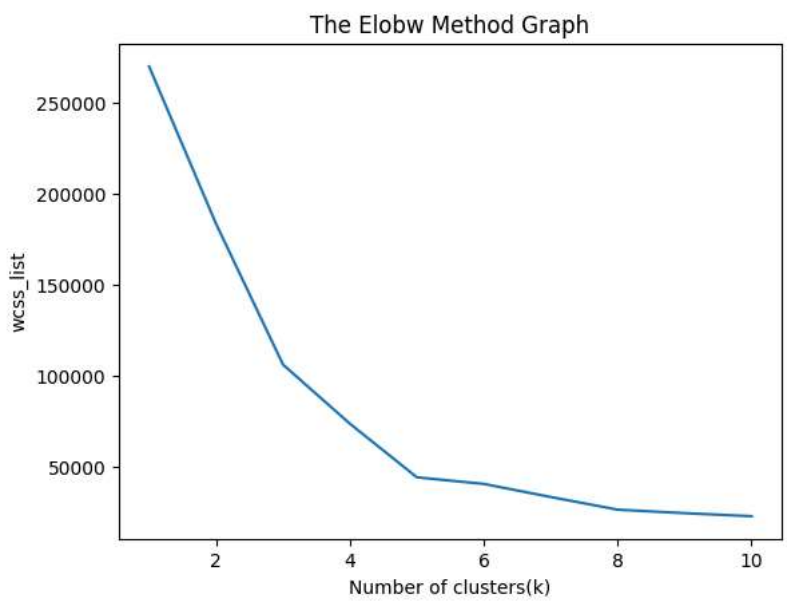



```
> # Fitting KNN Model to training dataset
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 1)
> classifier_knn
 [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
[10] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
[19] setosa     setosa     versicolor virginica  versicolor versicolor versicolor versicolor versicolor
[28] versicolor virginica  versicolor versicolor versicolor versicolor versicolor versicolor versicolor
[37] versicolor versicolor versicolor versicolor virginica  virginica  virginica  virginica  virginica
[46] virginica  virginica  versicolor virginica  virginica  virginica  virginica  virginica  versicolor
[55] virginica  virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
>
>
> cm <- table(test_cl$Species, classifier_knn)
> cm
            classifier_knn
             setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         18         2
  virginica       0          2        18
>
>
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.933333333333333"
> # K = 3
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 3)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.933333333333333"
>
> # K = 5
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 5)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.95"
```

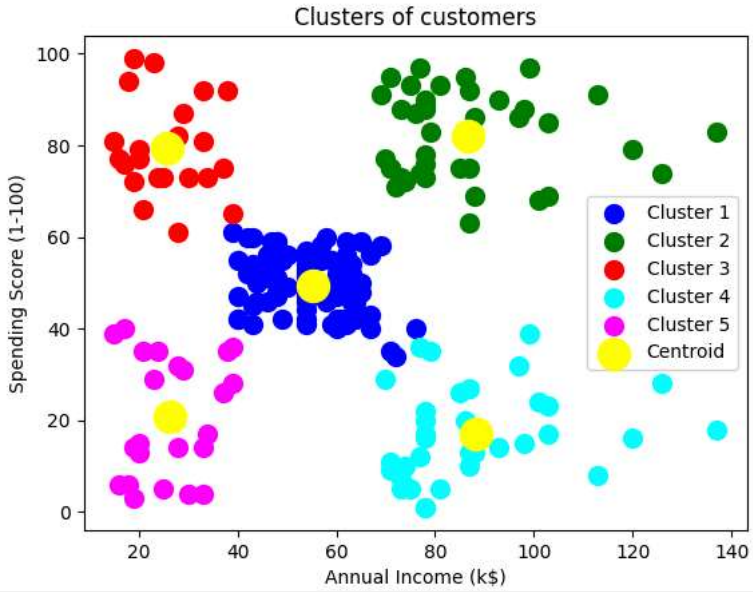
```
The downloaded binary packages are in
        C:\Users\Student\AppData\Local\Temp\RtmpMHtCrL\downloaded_packages
>
>
> library(e1071)
> library(caTools)
> library(class)
>
>
> data(iris)
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> split <- sample.split(iris, SplitRatio = 0.7)
> train_cl <- subset(iris, split == "TRUE")
> test_cl <- subset(iris, split == "FALSE")
>
>
> train_scale <- scale(train_cl[, 1:4])
> test_scale <- scale(test_cl[, 1:4])
>
>
> head(train_scale)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
2   -1.0878207 -0.05556229    -1.316362   -1.333546
3   -1.3257402  0.39903828    -1.372617   -1.333546
4   -1.4446999  0.17173799    -1.260107   -1.333546
7   -1.4446999  0.85363885    -1.316362   -1.198237
8   -0.9688609  0.85363885    -1.260107   -1.333546
9   -1.6826194 -0.28286258    -1.316362   -1.333546
> head(test_scale)
   Sepal.Length Sepal.Width Petal.Length Petal.Width
1    -0.9664683  0.91775243    -1.354158   -1.272374
5    -1.0893245  1.15107932    -1.354158   -1.272374
6    -0.5978999  1.85105998    -1.183823   -1.022889
10   -1.2121806 -0.01555513    -1.297380   -1.397117
11   -0.5978999  1.38440621    -1.297380   -1.272374
15   -0.1064753  2.08438687    -1.467714   -1.272374
> # Fitting KNN Model to training dataset
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
```

```
> # K = 7
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 7)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.966666666666667"
>
> # K = 15
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 15)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.983333333333333"
>
> # K = 19
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 19)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.966666666666667"
> # K = 3
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 3)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.933333333333333"
>
> # K = 5
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 5)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.95"
>
> # K = 7
> classifier_knn <- knn(train = train_scale,
+                       test = test_scale,
+                       cl = train_cl$Species,
+                       k = 7)
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy =', 1-misClassError))
[1] "Accuracy = 0.966666666666667"
```

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd


dataset=pd.read_csv('/content/Mall_Customers.csv')


x = dataset.iloc[:, [3, 4]].values


#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```



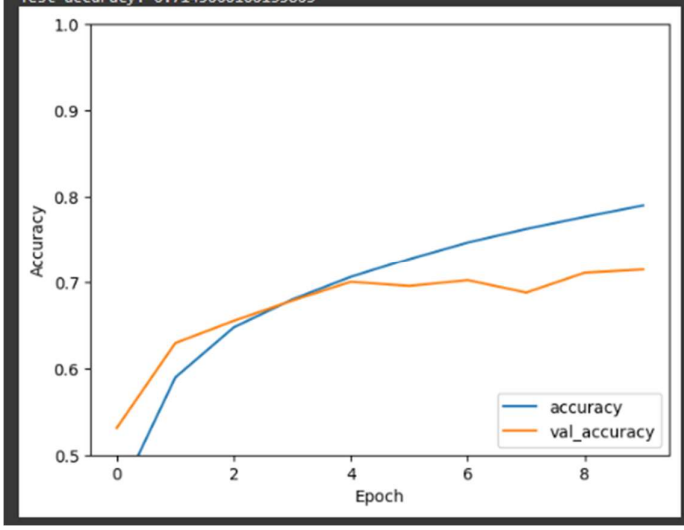

```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)


#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```


Clusters of customers
Spending Score (1-100)
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroid
0
20
40
60
80
100
20
40
60
80
100
120
140

Practical 11

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
# Load and preprocess the CIFAR10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0

# Define the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")

# Plot training history (optional)
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_s
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 50ms/step - accuracy: 0.3526 - loss: 1.7454 - val_accuracy: 0.5313 - val_loss: 1.2987
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 46ms/step - accuracy: 0.5758 - loss: 1.1994 - val_accuracy: 0.6293 - val_loss: 1.0616
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 47ms/step - accuracy: 0.6423 - loss: 1.0168 - val_accuracy: 0.6550 - val_loss: 0.9724
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 46ms/step - accuracy: 0.6752 - loss: 0.9240 - val_accuracy: 0.6784 - val_loss: 0.9329
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 71s 46ms/step - accuracy: 0.7077 - loss: 0.8337 - val_accuracy: 0.7002 - val_loss: 0.8751
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 47ms/step - accuracy: 0.7294 - loss: 0.7701 - val_accuracy: 0.6954 - val_loss: 0.9079
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 84s 48ms/step - accuracy: 0.7483 - loss: 0.7219 - val_accuracy: 0.7021 - val_loss: 0.8801
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 47ms/step - accuracy: 0.7676 - loss: 0.6647 - val_accuracy: 0.6877 - val_loss: 0.9188
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 49ms/step - accuracy: 0.7811 - loss: 0.6213 - val_accuracy: 0.7107 - val_loss: 0.8640
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 72s 46ms/step - accuracy: 0.7952 - loss: 0.5811 - val_accuracy: 0.7145 - val_loss: 0.8668
313/313 - 4s - 11ms/step - accuracy: 0.7145 - loss: 0.8668
Test accuracy: 0.7145000100135803
```



Object detection using CNN

```
import tensorflow as tf

import numpy as np

import cv2

from tensorflow.keras.losses import mse


# Load the saved model

model = tf.keras.models.load_model('my_cifar10_model.h5', custom_objects={'mse': mse})


# Define class names for CIFAR-10

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

               'dog', 'frog', 'horse', 'ship', 'truck']


# Display the uploaded image

plt.imshow(img)

plt.axis('off')
```

```
plt.show()



# Load and preprocess an image (replace with your own image)
img_path = '/content/download.jpg'  # Replace with the path to your image
img = cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (32, 32))  # Resize to CIFAR-10 input size
img = img / 255.0  # Normalize


# Make a prediction
img_array = np.expand_dims(img, axis=0)
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions[0])

# Display the result
print(f"Predicted class: {class_names[predicted_class]}")
```