LAB MANUAL

Course Code: ISL66

Course Name: Machine Learning

Course Coordinator: Dr.Mydhili K Nair

MACHINE LEARNING

Course Code: ISL66 **Credits: 0:0:1**
Prerequisites: Scripting Languages **Contact Hours: 14P**
Course Coordinator: Dr.Mydhili K Nair

Laboratory Experiments:

Implement the following programs using Python

1. **Supervised Learning Algorithms - Linear Regression:**
   a) **Simple Linear Regression - Univariate**
   b) **Multiple Linear Regression - Multivariate**
   Consider ***any dataset*** from UCI repository. Create Simple and Multiple Linear Regression models using the training data set. Predict the scores on the test data and find the error in prediction (E.g. RMSE). Include appropriate code snippets to visualize the model. Use Sub-Plots Interpret the result. Write the Inference.
2. **Model Measurement Analysis**: Using ***any dataset*** and ***any classifier***, calculate TP, TN, FP and FN from sklearn library functions. Also calculate different metrics (Accuracy, Precision, Recall(Sensitivity), F1-Score, MCC, Specificity, Negative Predictive Value) by defining our own functions. Compare your values with scikit-learn's library functions. Get the result of Confusion Matrix using sklearn. Using sklearn, plot the ROC & AUC Curves for your test data and random probabilities. Using sklearn, calculate the AUC of your test data and of random probabilities. Interpret the results. Write the inference/analysis of each output.
3. **Supervised Learning Algorithms - KNN:** Build a KNN model for predicting if a person will have diabetes or not with a high accuracy score. Perform some appropriate Pre-Processing steps on the given dataset for better results. Implement the KNN algorithm on your own. Try other possible processes that can be done to dataset and tuning the model to increase accuracy such as Increase K value, Normalization and Different Distance Metrics. Perform Feature Ablation Study. Additional Tries: Weight the features before doing KNN prediction.
4. **Artificial Neural Networks - Multi Layer Perceptron:** Write a program to construct a single layer Perceptron for a dataset. The dataset should have at least 100 records. Each record should have at least 4 floating point features and a binary label (0 - negative or 1 - positive). The program should contain functions to read the dataset from the file, split the data into train and test, ensure the data is split in the same way every time the program runs, initialize the weights of the perceptron, learning rate and epochs. Define the activation function.Train the model. Print the learned weights and the hyperparameters. Predict the outputs on train and test data. Print the confusion matrix, accuracy, precision, recall on train and test data
5. **Un-Supervised Learning Algorithms - K-Means Clustering:** Build a K-Means Model for the given dataset. Build a K-Means Model for the given Dataset. Implement the BIC function that takes the cluster and data points and returns BIC value. Implement a function to pick the best K value, that is maximize the BIC.Visualize the pattern found by plotting K v/s BIC.
6. **Un-Supervised Learning Algorithms - Hierarchical Clustering:** Using any dataset from the UCI repository implement ***any one*** type of Hierarchical Clustering. Plot the Dendrogram for

Hierarchical Clustering and analyze your result. Plot the clustering output for the same dataset using these two partitioning techniques.Compare the results. Write the inference.

7. **Supervised Learning Algorithms - Support Vector Machines:** Use SVM to classify the flowers in Iris dataset. Visualize the results for each of the following combinations - For every pair of (different) features in the dataset (there are 4). Which pair separates the data easily?Using One-vs-Rest and using One-vs-One. Which one fits better? Which one is easier to compute? Why? Using different kernels (Linear, RBF, Quadratic).
8. **Supervised Learning Algorithms - Decision Trees:** Implement decision trees considering a data set of your choice. Create an ID3 Decision Tree. Create a CART Decision Tree. Compare and Contrast the two
9. **Supervised Learning Algorithms - Logistic Regression:** Implement logistic regression and test it using any dataset of your choice from UCI repository. The output should include Confusion Matrix, Accuracy, Error rate, Precision, Recall and F-Measure.
10. **Probabilistic Supervised Learning - Naive Bayes:** Create a dataset from the sample given to you(e.g. "Play Tennis Probability", "Shopper Buying Probability" etc.). Perform the necessary pre-processing steps such as encoding. Train the model using Naive Bayes Classifier. Give new test data and predict the classification output. Handcode the classification probability and compare with the model output. Analyze and write the inference.

Reference:

1. 1. Stephen Marsland, "Machine Learning - An Algorithmic Perspective",Second Edition, CRC Press - Taylor and Francis Group, 2015
2. 2. Ethem Alpaydin, "Introduction to Machine Learning", Second Edition, MIT Press, Prentice Hall of India (PHI) Learning Pvt. Ltd. 2010

Course Outcomes (COs):

At the end of the course, student will be able to -

1.	Design and implement the various Machine Learning Algorithms in the realm of supervised and unsupervised learning. **(PO – 1(2),2(3),3(3),4(3),5(2),12(3)) & (PSO – 1(3), 2(2))**
2.	Demonstrate the working principle of these different ML models, determine their performance, usage and their applications. **(PO – 1(2), 3(2), 10(3)) & (PSO – 1(3),3(3))**
3.	Analyze the results and produce substantial written documentation. **(PO – 1(2),4(3),10(3),12(2)) & PSO – (1(2),2(2),3(2))**

Conduction of Practical Examination:- (50 Marks)

- All laboratory experiments are to be included for practical examination.
- Marks Distribution:
  - ➢ Procedure Writing (20 Marks)
  - ➢ Implementation and Testing (20 Marks)
  - ➢ Viva (10 Marks)

1. Supervised Learning Algorithms - Linear Regression:

a) Simple Linear Regression - Univariate

b) Multiple Linear Regression - Multivariate

Consider ***any dataset*** from UCI repository. Create Simple and Multiple Linear Regression models using the training data set. Predict the scores on the test data and find the error in prediction (E.g. RMSE). Include appropriate code snippets to visualize the model. Use Sub-Plots Interpret the result. Write the Inference.

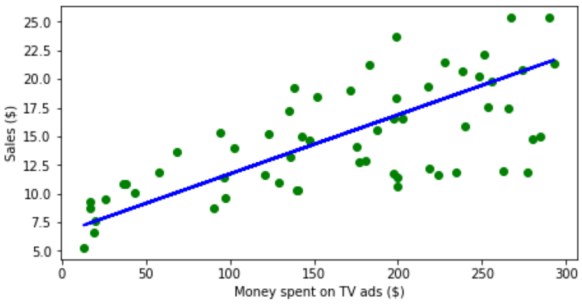
a) Simple Linear Regression - Univariate

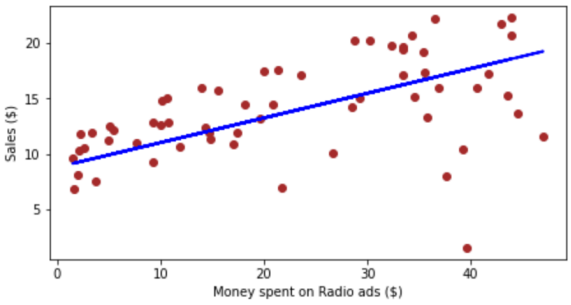
a) Output of Simple Linear Regression

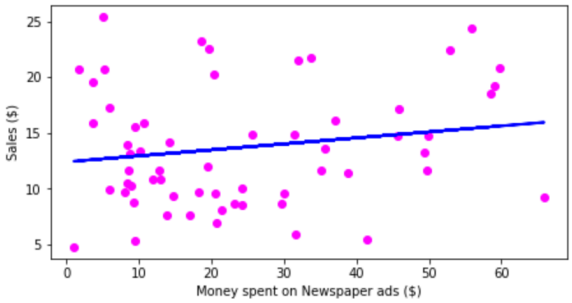
```
Slope(m):  0.05205778050848435
Intercept(c):  6.531631363360591
The linear model of TV versus Sales is: Y = 6.53 + 0.052X

Slope(m):  0.09934578112167217
Intercept(c):  11.144690071120326
The linear model for Newspaper vs Sales is: Y = 11.1 + 0.099X

Slope(m):  0.18864441331874413
Intercept(c):  9.404521200642243
The linear model for Radio versus Sales is: Y = 9.4 + 0.19X
```







Newspaper RMSE = 5.35199409368765

Radio RMSE = 4.027964590815203

TV RMSE = 3.141392475118249

Inference

The RMSE Value for TV is the least. This means that the money spent on TV. Ads has the highest possiblity of a reliable sales income prediction.

Supervised Learning Algorithms - Linear Regression:
In this Program the data set used is ***'Advertising.csv'***
It shows the money spent on **TV**, **Radio** and **Newspaper** Ads and the ***Sales*** Income generated.

The Dataset is 200 rows and 5 columns. (TV, Radio,Newspaper and Sales). It has a Sl# column which is unnames.

Loading the Dataset

a) If Local

b) If Remote (here Google Drive)

```
"""
import pandas as pd
import numpy as np
data = pd.read_csv("/content/sample_data/Advertising.csv")
data.head()

#Getting the Column Labels.
data.columns

#Dropping the first Sl# column which is unnamed
data.drop(['Unnamed: 0'], axis=1)

"""# **Plotting the raw data as. three scatter plots in a grid**"""

import matplotlib.pyplot as plt
myplt=plt.figure(figsize=(15,10))
ax1 = myplt.add_subplot(2,2,1)
ax1=plt.scatter(
 data['TV'],
 data['sales'],
 c='green' #change the color and see output
)
plt.xlabel("Money spent on TV ads ($)")
plt.ylabel("Sales ($)")
#plt.show()

#myplt.figure(figsize=(8,4))
ax2 = myplt.add_subplot(2,2,2)
ax2=plt.scatter(
 data['radio'],
 data['sales'],
 c='brown' #change the color and see output
```

```
)
plt.xlabel("Money spent on Radio ads ($)")
plt.ylabel("Sales ($)")
#plt.show()


#myplt.figure(figsize=(8,4))
ax3 = myplt.add_subplot(2,2,3)
ax3=plt.scatter(
 data['newspaper'],
 data['sales'],
 c='orange' #change the color and see output
)
plt.xlabel("Money spent on Newspaper ads ($)")
plt.ylabel("Sales ($)")
plt.show()

"""# **Applying Linear Regression. on TV Ads versus Sales**
**1)** Splitting the data. into train & test using sklearn library train_test_split

**2)** Finding the Best Fit Line for the data using sklearn inbuilt Linear Regression Functionn
"""

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import train_test_split

X = data['TV'].values.reshape(-1,1) #Unsure of number of rows of TV data. But its one column
y = data['sales'].values.reshape(-1,1) #Unsure of number of rows of Sales data. But its one column
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3) #Using sklearn library 70% Train and 30% Test
reg = LinearRegression() #Using sklearn library
reg.fit(x_train, y_train) #Fit the best fit regression line

"""# **Getting the equation for the Best Fit Line**

Finding the Slope(m) and Intercept (c)
```

```
Creating the Mathematical Equation for the Nest Fit Line
"""

print("Slope(m): ",reg.coef_[0][0])
print("Intercept(c): ",reg.intercept_[0])
print("The linear model of TV versus Sales is: Y = {:.3} +
{:.2}X".format(reg.intercept_[0], reg.coef_[0][0]))
#Y = {:.3} + {:.2}X is done to truncate the decimal points. hange this value and see
the output.

"""# **Plot the Regression Line seperately Sales versus. TV Ads Money**


**Note**: Its plotted seperately for each feature
"""

predictions = reg.predict(x_test)
plt.figure(figsize=(16, 8))
plt.scatter(
 x_test,
 y_test,
 c='green' #change the color and see output
)
plt.plot(
 x_test,
 predictions,
 c='blue',
 linewidth=2
)
plt.xlabel("Money spent on TV ads ($)")
plt.ylabel("Sales ($)")
plt.show()

"""# **Calculate the RMSE for the feature TV Ads Money versus Sales**"""

rmse = np.sqrt(mean_squared_error(y_test,predictions))
print("Root Mean Squared Error = ",rmse)

"""# **Applying Linear Regression. on radio Ads versus Sales**
**1)** Splitting the data. into train & test using sklearn library train_test_split
```

```
**2)** Finding the Best Fit Line for the data using sklearn inbuilt Linear Regression Function

**3)** Getting the mathematical equation with value of slope and intercept
"""

X = data['radio'].values.reshape(-1,1) #Unsure of number of rows of TV data. But its one column
y = data['sales'].values.reshape(-1,1) #Unsure of number of rows of Sales data. But its one column
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3) #Using sklearn library
reg = LinearRegression() #Using sklearn library
reg.fit(x_train, y_train) #Fit the best fit regression line
print("Slope(m): ",reg.coef_[0][0])
print("Intercept(c): ",reg.intercept_[0])
print("The linear model for Radio versus Sales is: Y = {:.3} + {:.2}X".format(reg.intercept_[0], reg.coef_[0][0]))

"""# **Plotting the Regression Line Radio Ads versus Sales**"""

predictions = reg.predict(x_test)
plt.figure(figsize=(16, 8))
plt.scatter(
 x_test,
 y_test,
 c='brown' #change the color and see output
)
plt.plot(
 x_test,
 predictions,
 c='blue',
 linewidth=2
)
plt.xlabel("Money spent on Radio ads ($)")
plt.ylabel("Sales ($)")
plt.show()

"""# **Calculate the RMSE for the feature Radio Ads Money versus Sales**"""
```

```
rmse = np.sqrt(mean_squared_error(y_test,predictions))
print("Root Mean Squared Error = ",rmse)

"""# **Applying Linear Regression. on Newspaper Ads versus Sales**
**1)** Splitting the data. into train & test using sklearn library train_test_split

**2)** Finding the Best Fit Line for the data using sklearn inbuilt Linear Regression Function

**3)** Getting the mathematical equation with value of slope and intercept
"""

X = data['newspaper'].values.reshape(-1,1) #Unsure of number of rows of TV data. But its one column
y = data['sales'].values.reshape(-1,1) #Unsure of number of rows of Sales data. But its one column
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3) #Using sklearn library
reg = LinearRegression() #Using sklearn library
reg.fit(x_train, y_train) #Fit the best fit regression line
print("Slope(m): ",reg.coef_[0][0])
print("Intercept(c): ",reg.intercept_[0])
print("The linear model for Newspaper vs Sales is: Y = {:.3} +
{:.2}X".format(reg.intercept_[0], reg.coef_[0][0]))

"""# **Plotting the Regression Line Newspaper Ads versus Sales**"""

predictions = reg.predict(x_test)
plt.figure(figsize=(16, 8))
plt.scatter(
 x_test,
 y_test,
 c='magenta' #change the color and see output
)
plt.plot(
 x_test,
 predictions,
 c='blue',
 linewidth=2
)
```

```
plt.xlabel("Money spent on Newspaper ads ($)")
plt.ylabel("Sales ($)")
plt.show()

"""# **Calculate the RMSE for the feature Newspaper Ads Money versus Sales**"""

rmse = np.sqrt(mean_squared_error(y_test,predictions))
print("Root Mean Squared Error = ",rmse)

"""# **Plotting in a Grid Using Sub Plot function in Matplotlib**"""

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import train_test_split

X = data['TV'].values.reshape(-1,1) #Unsure of number of rows of TV data. But its one column
y = data['sales'].values.reshape(-1,1) #Unsure of number of rows of Sales data. But its one column
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3) #Using sklearn library
reg = LinearRegression() #Using sklearn library
reg.fit(x_train, y_train) #Fit the best fit regression line

predictions = reg.predict(x_test)
myplt=plt.figure(figsize=(16, 8))
ax1 = myplt.add_subplot(2,2,1)
ax1=plt.scatter(
 x_test,
 y_test,
 c='green' #change the color and see output
)
plt.plot(
 x_test,
 predictions,
 c='blue',
 linewidth=2
)
plt.xlabel("Money spent on TV ads ($)")
plt.ylabel("Sales ($)")
```

```
X = data['radio'].values.reshape(-1,1) #Unsure of number of rows of TV data. But its one column
y = data['sales'].values.reshape(-1,1) #Unsure of number of rows of Sales data. But its one column
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3) #Using sklearn library
reg = LinearRegression() #Using sklearn library
reg.fit(x_train, y_train) #Fit the best fit regression line

predictions = reg.predict(x_test)
ax2 = myplt.add_subplot(2,2,2)
ax2=plt.scatter(
 x_test,
 y_test,
 c='brown' #change the color and see output
)
plt.plot(
 x_test,
 predictions,
 c='blue',
 linewidth=2
)
plt.xlabel("Money spent on Radio ads ($)")
plt.ylabel("Sales ($)")

X = data['newspaper'].values.reshape(-1,1) #Unsure of number of rows of TV data. But its one column
y = data['sales'].values.reshape(-1,1) #Unsure of number of rows of Sales data. But its one column
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3) #Using sklearn library
reg = LinearRegression() #Using sklearn library
reg.fit(x_train, y_train) #Fit the best fit regression line

predictions = reg.predict(x_test)
ax3 = myplt.add_subplot(2,2,3)
ax3=plt.scatter(
 x_test,
```

```
 y_test,
 c='magenta' #change the color and see output
)
plt.plot(
 x_test,
 predictions,
 c='blue',
 linewidth=2
)
plt.xlabel("Money spent on Newspaper ads ($)")
plt.ylabel("Sales ($)")
plt.show()

"""Newspaper RMSE = 5.35199409368765

Radio RMSE = 4.027964590815203

TV RMSE = 3.141392475118249

# **Inference**

The RMSE Value for TV is the least.
This means that the money spent on TV. Ads has the highest possiblity of a reliable sales income prediction.

**Note**: There is another measure called R-Squared. R-Squared only works as intended in a simple linear regression model with one independant variable.

If there are multiuple independant attributes, the effect of each has to be calculated seperately as R-Squared Values
"""
```

b) Multiple Linear Regression - Multivariate

a) Output of Multiple Linear Regression

```
The linear model is: Y = 2.877 + 0.046565*TV + 0.17916*radio +
0.0034505*newspaper
```

Linear Model is: y=2.88+0.0466×TV+0.179×Radio+0.00345×Newspaper

This is a Multi-Variant Linear Regression Model Equation.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

- y is the response
- β_0 is the intercept
- β_1 is the coefficient for x_1 (the first feature)
- β_n is the coefficient for x_n (the nth feature)

In this case:

$$y = \beta_0 + \beta_1 \times TV + \beta_2 \times Radio + \beta_3 \times Newspaper$$

The β values are called the **model coefficients**

- These values are "learned" during the model fitting step using the "least squares" criterion
- Then, the fitted model can be used to make predictions

Inference 1 Meaning of Coefficient

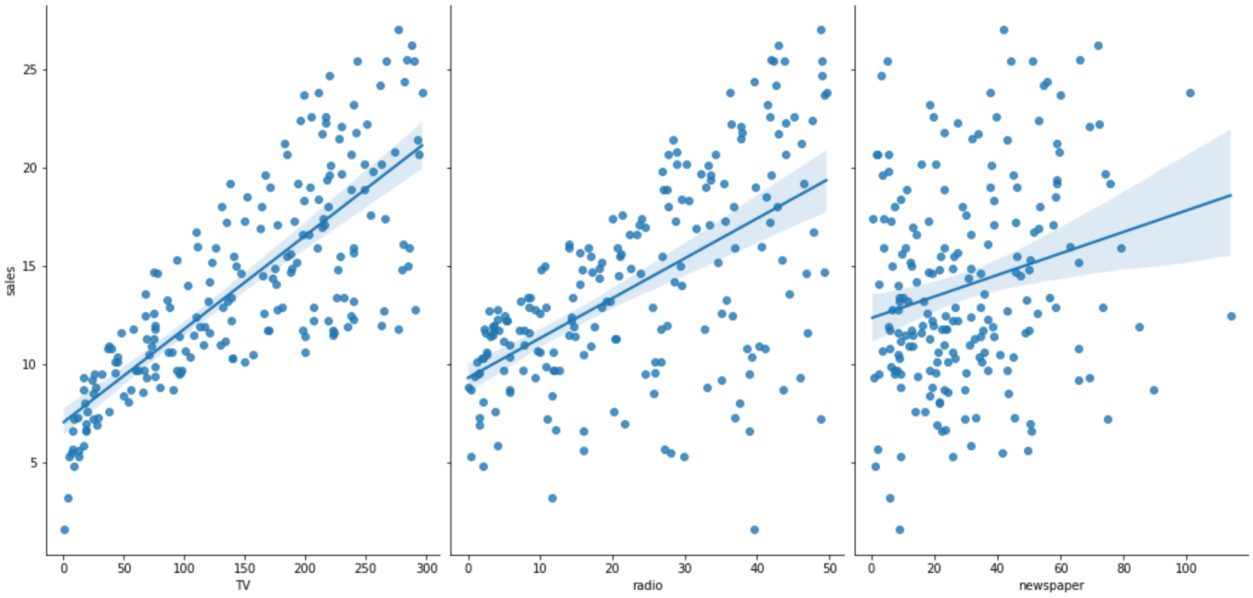
E.g. TV Coefficient 0.0466

For a given amount of Radio and Newspaper ad spending, a "unit" increase in TV ad spending is associated with a 0.0466 "unit" increase in Sales. Or more clearly: For a given amount of Radio and Newspaper ad spending, an additional $1,000 spent on TV ads is associated with an increase in sales of 46.6 items.

How $1,000? 46.6 Units = 0.0466 x 1000

Inference 2

If an increase in TV ad spending was associated with a decrease in sales, β1 would be negative.



Supervised Learning Algorithms- Linear Regression(Multi-Variant):
Step 1: Reading Data Using Pandas
Display first five and last five records in the dataset

Check the shape of the DataFrame (rows, columns)
"""

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
data = pd.read_csv("sample_data/Advertising.csv")
data.head()

data.tail()

# there are 200 rows x 4 columns
data.shape

"""# **Step 2: Interpret the data**
**What are the features?**

**1) TV:** advertising dollars spent on TV for a single product in a given market (in thousands of dollars)

**2) Radio:** advertising dollars spent on Radio

**3) Newspaper:** advertising dollars spent on Newspaper

**What is the response?**

**Sales:** sales of a single product in a given market (in thousands of items)

**Inference**

Because the response variable is *continuous*, this is a *regression problem*. There are 200 observations (represented by the rows), and each observation is a single market.

# **Step 3: Process Dataset & Apply Linear Regression**


*   Pre-process by dropping unwanted columns
*   Split Dataset into test and train set
"""

Xs = data.drop(['sales', 'Unnamed: 0'], axis=1)
```

```
y = data['sales'].values.reshape(-1,1)
#x_train, x_test, y_train, y_test = train_test_split(Xs,y,test_size = 0.3) #Train is 70% and Test is 30%
x_train, x_test, y_train, y_test = train_test_split(Xs,y,random_state=1) # default split is 75% for training and 25% for testing
# 200 records : 75% train = 150 and 25% test = 50
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
reg = LinearRegression()
reg.fit(x_train, y_train)

"""# **Step 4: Display Values of Slope and Intercept and Infer from it**

**Linear Model is: y=2.88+0.0466×TV+0.179×Radio+0.00345×Newspaper**

This is a Multi-Variant Linear Regression Model Equation.

**Inference 1**
Meaning of Coefficient

E.g. TV Coefficient 0.0466

For a given amount of Radio and Newspaper ad spending, a "unit" increase in TV ad spending is associated with a 0.0466 "unit" increase in Sales.
Or more clearly: For a given amount of Radio and Newspaper ad spending, an additional $1,000 spent on TV ads is associated with an increase in sales of 46.6 items.

How $1,000?
46.6 Units = 0.0466 x 1000

**Inference 2**

If an increase in TV ad spending was associated with a decrease in sales,  β1 would be negative.
"""

print("Slope: ",reg.coef_)
print("Intercept: ",reg.intercept_)
```

```
print("The linear model is: Y = {:.5} + {:.5}*TV + {:.5}*radio +
{:.5}*newspaper".format(reg.intercept_[0], reg.coef_[0][0], reg.coef_[0][1],
reg.coef_[0][2]))

"""# **Step 5: Make Predictions**"""

# make predictions on the testing set

y_pred = reg.predict(x_test)
#print(y_pred)

def myfunc(TV,radio,newspaper):
  Y = 2.877 + 0.046565*TV + 0.17916*radio + 0.0034505*newspaper
  return Y
predictedsales = myfunc(39.5,41.1,10.8)
print("Predicted Sales is ", predictedsales)

def myfunc(TV,radio):
  Y = Y = 2.927 + 0.0466*TV + 0.1811*radio
  return Y
predictedsales = myfunc(39.5,41.1)
print("Predicted Sales is ", predictedsales)

"""# **Step 6: Model Evaluation Metrics for Regression**
**Note:** Evaluation metrics for classification problems, such as accuracy, are not
useful for regression problems. Instead, we need evaluation metrics designed for
comparing continuous values.

Three common evaluation metrics for regression problems are:

**Comparing these metrics:**

**MAE** is the easiest to understand, because it's the average error.

**MSE** is more popular than MAE, because MSE "punishes" larger errors.

**RMSE** is even more popular than MSE, because RMSE is interpretable in the
"y" units. Easier to put in context as it's the same units as our response variable
"""

from sklearn.metrics import mean_absolute_error
```

```
predictions = reg.predict(x_test)
mae = mean_absolute_error(y_test,predictions)
print("Mean Absolute Error = ",mae)

from sklearn.metrics import mean_squared_error
predictions = reg.predict(x_test)
mse = mean_squared_error(y_test,predictions)
print("Mean Squared Error = ",mse)

from sklearn.linear_model import LinearRegression
predictions = reg.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test,predictions))
print("Root Mean Squared Error = ",rmse)

reg.score(Xs, y)

"""# **Step : Visualize the data**

Using seaborn package as it has advanced visualization features
"""

# Commented out IPython magic to ensure Python compatibility.
# conventional way to import seaborn
import seaborn as sns

# allow plots to appear within the notebook
# %matplotlib inline

# visualize the relationship between the features and the response using
scatterplots
# this produces pairs of scatterplot as shown
# use aspect= to control the size of the graphs
# use kind='reg' to plot linear regression on the graph
sns.pairplot(data, x_vars=['TV', 'radio', 'newspaper'], y_vars='sales', size=7,
aspect=0.7, kind='reg')

#create a Python list of feature names
feature_cols = ['TV', 'radio']

# use the list to select a subset of the original DataFrame
X = data[feature_cols]
```

```
# select a Series from the DataFrame
Y = data.sales

# split into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=1)

# fit the model to the training data (learn the coefficients)
reg.fit(X_train, Y_train)

print("Slope: ",reg.coef_)
print("Intercept: ",reg.intercept_)
print("The linear model is: Y = 2.927 + 0.0466*TV + 0.1811*radio")

# make predictions on the testing set
Y_pred = reg.predict(X_test)
# Commented out IPython magic to ensure Python compatibility.
# conventional way to import seaborn
import seaborn as sns
# allow plots to appear within the notebook
# %matplotlib inline
# visualize the relationship between the features and the response using
scatterplots
# this produces pairs of scatterplot as shown
# use aspect= to control the size of the graphs
# use kind='reg' to plot linear regression on the graph
sns.pairplot(data, x_vars=['TV', 'radio'], y_vars='sales', size=7, aspect=0.7,
kind='reg')

from sklearn.metrics import mean_absolute_error,  mean_squared_error
predictions = reg.predict(X_test)
mae = mean_absolute_error(Y_test,predictions)
print("Mean Absolute Error = ",mae)

predictions = reg.predict(X_test)
mae = mean_squared_error(Y_test,predictions)
print("Mean Absolute Error = ",mae)

# compute the RMSE of our predictions
rmse = np.sqrt(mean_squared_error(Y_test,Y_pred))
```

```
print("Root Mean Squared Error = ",rmse)

reg.score(X, y)

"""# **Inference:**
The RMSE decreased when we removed Newspaper from the model. (Error is something we want to minimize, so a lower number for RMSE is better.
Thus, it is unlikely that this feature (Newspaper)is useful for predicting Sales, and should be removed from the model.
"""
```

2. Model Measurement Analysis: Using ***any dataset*** and ***any classifier*** do the following:

a) Calculate TP, TN, FP and FN from sklearn library functions

b) Calculate different metrics (Accuracy, Precision, Recall(Sensitivity), F1-Score, MCC, Specificity, Negative Predictive Value) by defining our own functions

c) Get the sklearn metrics of these values

d) Verify them by comparing with scikit-learn's library functions.

e) Get the result of Confusion Matrix using sklearn

f) Using sklearn, plot the ROC Curve of the probability values in our test data

g) Using sklearn, plot the ROC Curve of random probabilities

h) Calculate the AUC of our test data using sklearn

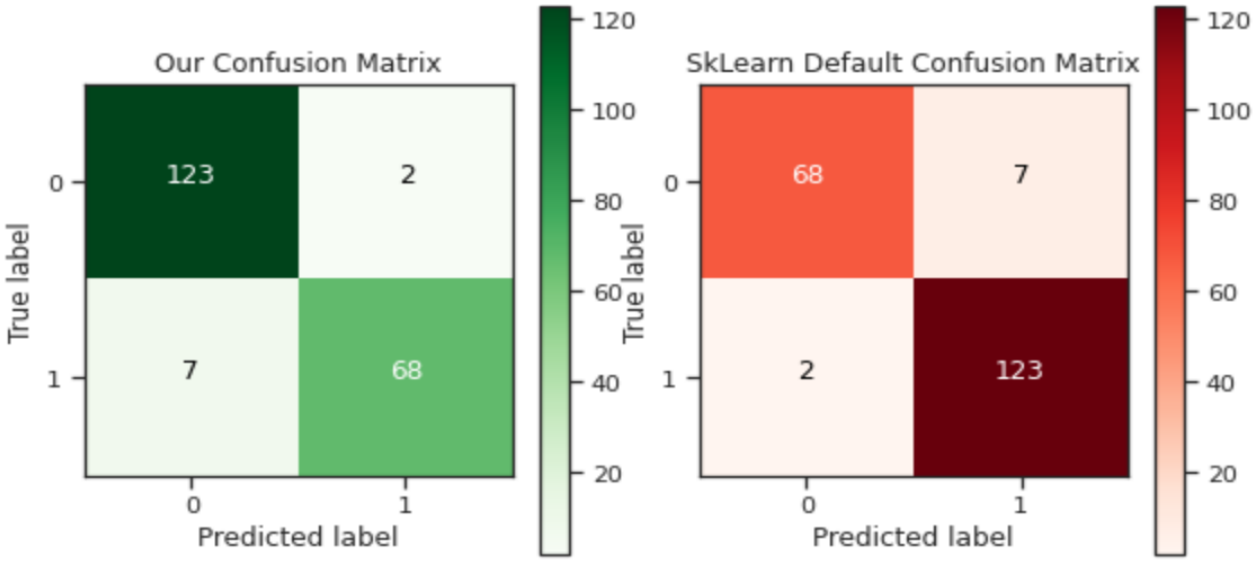
i) Calculate the AUC of random probabilities using sklearn

j) Interpret the results. Write the inference/analysis of each output.

Dataset Used: In-built sklearn's Breast Cancer

OUTPUT ANALYSIS

```
True Negatives 68
True Positives 123
False Positives 2
False Negatives 2
```



Refer Code: Run this *Handcoded_ConfusionMatrix_ROC_AUC.ipynb* to get the above output. This depicts the difference in Confusion Matrix between sklearn and what we want.

- In this program *(Prg#2_Model Measurement Analysis.ipynb)* the following code is used to get the Confusion Matrix we want:
- `conf=metrics.confusion_matrix(y_test, y_preds,labels=[1,0])`
- `#Note to change the labels from the default 0,1 to 1,0`

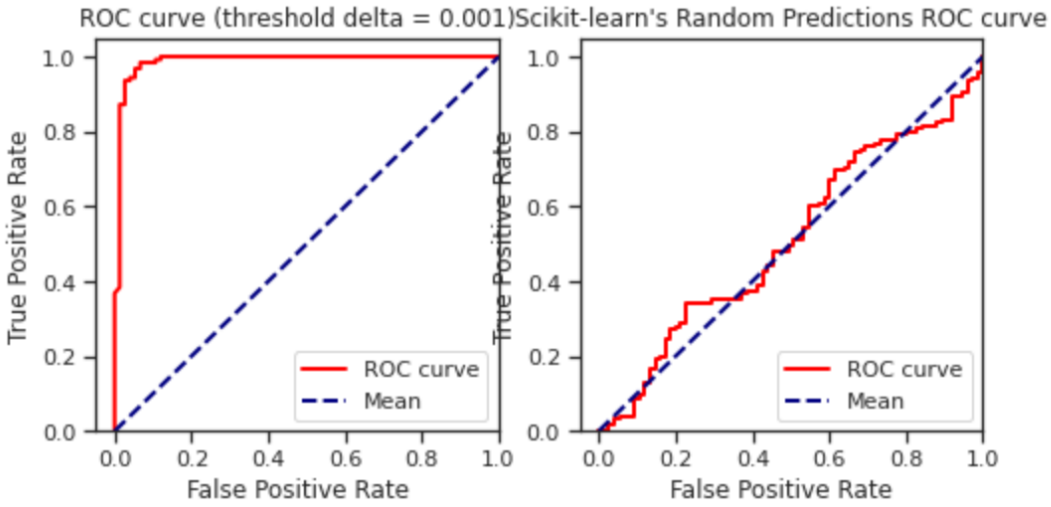
```
Calculated and scikit-learn Accuracy:                              0.979,  0.955
Calculated and scikit-learn Precision score:                       0.984,  0.946
Calculated and scikit-learn Recall score:                          0.984,  0.984
Calculated and scikit-learn F1 score:                              0.984,  0.965
Calculated and scikit-learn Matthew's correlation coefficient:     0.955,  0.904
```

Sample test data probabilities

```
2D to 1D reshaped Probability of benign.  [0.99697262 0.08136644 0.99999168 0.96809392 0.99999907]
```

Random probabilities

```
Randon Probablities of Being Benign are:  [0.37772889 0.53432747 0.49656119 0.38961809 0.29763517]
```



ROC curve (Receiver Operating Characteristic curve)

A receiver operating characteristic curve, i.e. ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) *at various threshold settings. *

Analysis: The above is the ROC curve for the Breast Cancer Dataset.

1. The peak towards left-most corner means near perfect classifier
2. Random prediction will have the curve as the blue dotted straight line.
3. This ROC curve tells us that our model is nearly perfect classifier, with high accuracy!

ROC for Random Predictions

Analysis It is very close to the "Guess" line

Indicator of a Bad Classifier

Will give a low value of AUC

AUC

```
Scikit's ROC-AUC score of SVC model is
0.9872
Scikit's ROC-AUC score of random
predictions is:  0.5111
```

CODE - Model Measurement

```
# **Model Measurement Analysis**"""

import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import itertools

np.random.seed(42)  # for reproducibility
sns.set(rc={"figure.figsize": (8, 8)})
sns.set_style("ticks")

"""# **Dataset**
Breast Cancer Dataset provided by sckit-learn in dataset module is used.
*   Desscribe. this dataset
*   Analyze it
"""
data = load_breast_cancer()
print(data.DESCR[:760])  # print short description

"""# **Analysis of the Dataset**
*   There are 569 instances
*   Each instance has 30 attributes
*   Description - Specification, Name of each attribute is listed
"""
print(f"Types of cancer (targets) are {data.target_names}")

"""# **Analysis of Output:**
# **What are the targets classes i.e. types of cancer.?**
Target Attribute Values are - **['malignant' 'benign']**
This dataset is used for binary classification between two types of
cancer("Malignant" and "Benign").
"""
```

```
X = data.data  # features
y = data.target  # labels
print(f"Shape of features is {X.shape}, and shape of target is {y.shape}")
print("Targets are: ", y)

"""# **Analysis of Output**
We will consider benign as positive class, and malignant as negative class.
Therefore,
*   **0 for is_not_benign(malignant)**
*   **1 for is_benign**
"""
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=200,
random_state=42, stratify=y)
y_train[:10]

"""# **Split the data**
Since we shouldn't train and test our model with the same dataset, it is always a
good idea to split the data in three parts - train data, test data, and validation data.

We won't require validation data here.
We split the dataset into training and testing data
*   200 examples for testing. Code : test_size=200
*   Rest, 369 examples for training
"""
classifier = svm.SVC(kernel='linear', probability=True, verbose=True)

"""# **Analysis of Code:**
**Classifier Used:** scikit's Support Vector Machines classifier for prediction. That
is: sklearn.svm.
"""
classifier.fit(X_train, y_train)

"""# **Analysis of Code:**
*   fit/train the model on our training dataset.
*   It trains quite fast since we are working with relatively small dataset.
"""
y_preds = classifier.predict(X_test)
y_proba = classifier.predict_proba(X_test)
print("Prediction as class - malignanat or benign ", y_preds[:5])
```

```
print("Test Data are: ", y_test[:5])
print("Probability of malignant & probability of benign. ", y_proba[:5])
"""
# **Analysis of Code &. Output**
*   Save the prediction results both as ***probability*** and as ***classes***.
*   **y_preds** is a *1D vector* of one of {0, 1} values, denoting predictions as malignant and benign, respectively.
*   **y_test** is a 1D vector of one of (0,1) values of the 200 samples of test data
*   **y_proba** is a *2D vector*, where for each example, it contains a vector of length 2,[prob. of malignant, prob. of benign]
"""
y_proba = y_proba[:,1].reshape((y_proba.shape[0],))
print("2D to 1D reshaped Probability of benign. ", y_proba[:5])

"""# **Analysis of Code &. Output**
**y_proba** is reshaped into a 1D vector denoting the probability of having benign cancer.
"""
TN, FP, FP, TP = metrics.confusion_matrix(list(y_test), list(y_preds), labels=[0, 1]).ravel() #0,1 is default label of sklearn
print("True Negatives", TN)
print("True Positives", TP)
print("False Positives", FP)
print("False Negatives", FP)

results = {} #Dont miss tbis declaration. Later used to calculate ACcuracy etc.

conf = metrics.confusion_matrix(y_test, y_preds,labels=[1,0]) #Note to change the labels from the default 0,1 to 1,0
print("Confusion Matrix we want is: \n", conf)
"""# **Analysis of code**
**Confusion Matrix**
*   Calculate confusion matrix of the predictions
*   Implemented in Scikit-learn's sklearn.metrics.confusion_matrix.
"""
classes = [0, 1]
# plot confusion matrix
plt.imshow(conf, interpolation='nearest', cmap=plt.cm.Greens)
plt.title("Confusion Matrix")
```

```
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)

fmt = 'd'
thresh = conf.max() / 2.
for i, j in itertools.product(range(conf.shape[0]), range(conf.shape[1])):
    plt.text(j, i, format(conf[i, j], fmt),
             horizontalalignment="center",
             color="white" if conf[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')

"""**Below is. the code**
# **Analysis of Code & above Output**
From the confusion matrix, we can see the number of examples predicted correct by our classifier, for both classes seperately.
*   Get the numbers of True Positives, True Negatives, False Positives, and False Negatives from this confusion matrix.
*   Store these terms in some variables
*   Calculate the Accuracy

# **Calculate Accuracy**
number of examples correctly predicted / total number of examples
"""
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}") #Note the Formatting with f"{metric} and rounding to 3 decimal points with .3f

"""# **Calculate Sensitivity or Recall or True Positive Rate(TPR)**
number of samples actually and predicted as  `Positive` / total number of samples actually `Positive`
"""
metric = "TPR"
results[metric] = TP / (TP + FN)
```

```
print(f"{metric} is {results[metric]: .3f}")

"""# **Calculate Specificity/True Negative Rate(TNR)**
number of samples actually and predicted as  `Negative` / total number of samples actually  `Negative`
"""
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")

"""# **Calculate Precision/Positive Predictive Value(PPV)**
number of samples actually and predicted as  `Positive` / total number of samples predicted as `Positive`
"""
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")

"""# **Calculate Negative Predictive Value(NPV)**
number of samples actually and predicted as  `Negative` / total number of samples predicted as `Negative`
"""
metric = "NPV)"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
"""# **Negative F1-Score**
Harmonic Mean of Precision and Recall.
![alt
text](https://wikimedia.org/api/rest_v1/media/math/render/svg/5663ca95d471868169c4e4ea57c936f1b6f4a588)
"""
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")

"""# **Matthew's correlation coefficient(MCC)**
MCC = (TP*TN – FP*FN) / √(TP+FP)(TP+FN)(TN+FP)(TN+FN)
```

```
Matthew's coefficient range between `[-1, 1]`. `0` usually means totally random predictions. `1` means a perfect classifier, while a negative value (`[-1, 0)`) suggests a negative correlation between predictions and actual values.
Here' the formual for MCC
"""
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")

print(f"Calculated and scikit-learn Accuracy:                    {results['ACC']: .3f}, {metrics.accuracy_score(y_test, y_preds): .3f}")
print(f"Calculated and scikit-learn Precision score:             {results['PPV']: .3f}, {metrics.precision_score(y_test, y_preds): .3f}")
print(f"Calculated and scikit-learn Recall score:                {results['TPR']: .3f}, {metrics.recall_score(y_test, y_preds): .3f}")
print(f"Calculated and scikit-learn F1 score:                    {results['F1']: .3f}, {metrics.f1_score(y_test, y_preds): .3f}")
print(f"Calculated and scikit-learn Matthew's correlation coefficient: {results['MCC']: .3f}, {metrics.matthews_corrcoef(y_test, y_preds): .3f}")

"""#**Comparing these calculated metrics with scikit-learn**

**Analysis**: These values match with the values calculated from `scikit-learn`'s functions.
"""
def get_roc_curve(y_test, y_proba, delta=0.1):
    """
    Return the True Positive Rates (TPRs), False Positive Rates (FPRs),
    and the threshold values, seperated by delta.
    """
    thresh = list(np.arange(0, 1, delta)) + [1]
    TPRs = []
    FPRs = []
    y_pred = np.empty(y_proba.shape)
    for th in thresh:
        y_pred[y_proba < th] = 0
        y_pred[y_proba >= th] = 1
```

```
        # confusion matrix from the function we defined
        (TN, FP), (FN, TP) = get_confusion_matrix(y_test, y_pred)

        TPR = TP / (TP + FN)  # sensitivity
        FPR = FP / (FP + TN)  # 1 - specificity
        TPRs.append(TPR)
        FPRs.append(FPR)
    return FPRs, TPRs, thresh

delta = 0.001
FPRs, TPRs, _ = get_roc_curve(y_test, y_proba, delta)

# Plot the ROC curve
myplt=plt.figure(figsize=(8,8))
ax1 = myplt.add_subplot(2,2,1)
ax1=plt.plot(FPRs, TPRs, color='red',
         lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Mean")
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC curve (threshold delta = {delta})')
plt.legend(loc="lower right")

# Plot the ROC curve for Random Probabilities

# create random predictions
rand_proba = np.random.random(size=(y_proba.shape))
rand_proba[:5]  # 0.5 probability of being 0 or 1

print("\n Randon Probablities of Being Benign are: ",rand_proba[:5],"\n" )
FPRs, TPRs, _ = metrics.roc_curve(y_test, rand_proba)  # passing random preds

ax2 = myplt.add_subplot(2,2,2)
ax2=plt.plot(FPRs, TPRs, color='red',
         lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Mean")
plt.xlim([-0.05, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("Scikit-learn's Random Predictions ROC curve")
plt.legend(loc="lower right")
plt.show()
"""# **ROC curve (Receiver Operating Characteristic curve)**
A receiver operating characteristic curve, i.e. ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
The ROC curve is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) *at various threshold settings. *

**Analysis**: The above is the ROC curve for the Breast Cancer Dataset.
1. The peak towards left-most corner means near perfect classifier
2. Random prediction will have the curve as the blue dotted straight line.
3. This ROC curve tells us that our model is nearly perfect classifier, with high accuracy!
"""
auc_score = metrics.roc_auc_score(y_test, y_proba)
print(f"Scikit's ROC-AUC score of SVC model is {auc_score: .4f}")
"""# **AUC: Area under the Curve**
It is the area under the ROC curve formed by the predictions.
A totally random prediction will have AUC score 0.5, while a perfect classifier will have AUC score of 1.
Our Result is 0.9872 which indicates a good classifier!
# **Creating Random Probabilities for begnin cancer**
Using "random" function of numpy
# **ROC for Random Predictions**
**Analysis** It is very close to the "Guess" line
Indicator of a Bad Classifier
Will give a low value of AUC
"""
auc_score = metrics.roc_auc_score(y_test, rand_proba)
print(f"Scikit's ROC-AUC score of random predictions is: {auc_score: .4f}")
"""# **AUC value is very low for random predictions**"""
```

3. Supervised Learning Algorithms - KNN: Build a KNN model for prediction whether a person will have diabetes or not with a high accuracy score:

a) Perform some appropriate Pre-Processing steps on the given dataset for better results

b) Implement the KNN algorithm on your own. (Don't use any pre built code/lib)

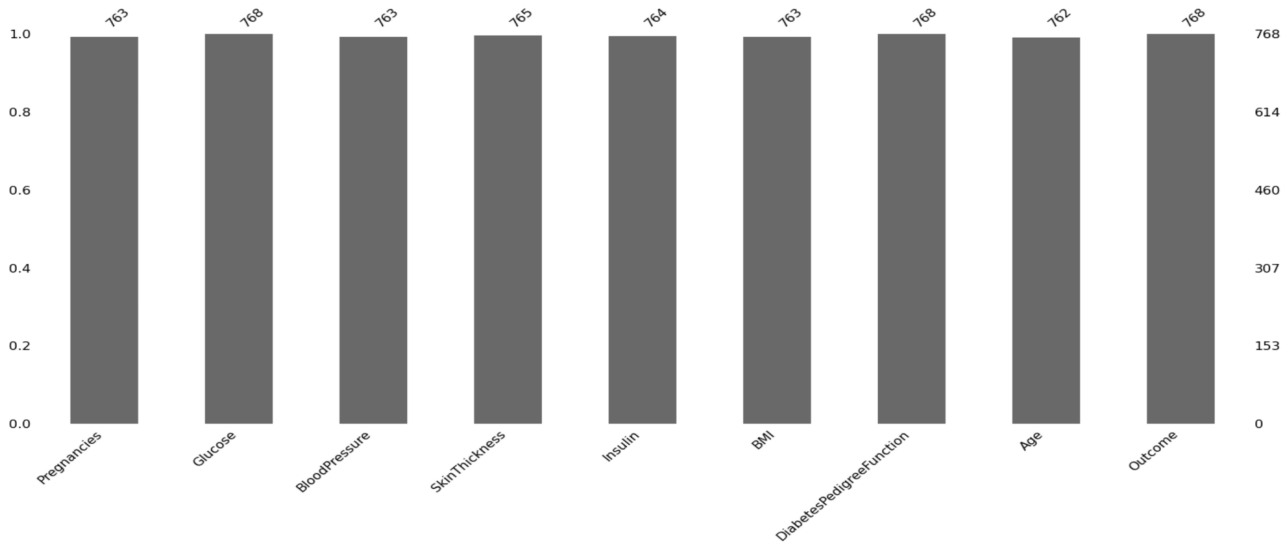
c) Try other possible processes that can be done to dataset and tuning the model to increase
accuracy.

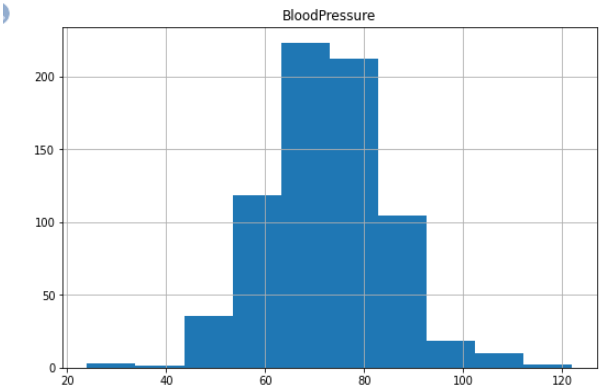
- Increase K value
- Normalization
- Different Distance Metrics

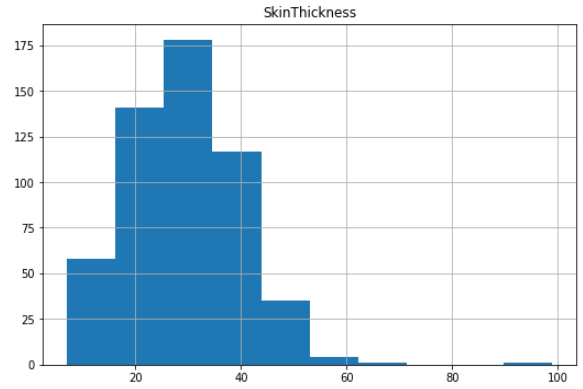
d) Perform Feature Ablation Study

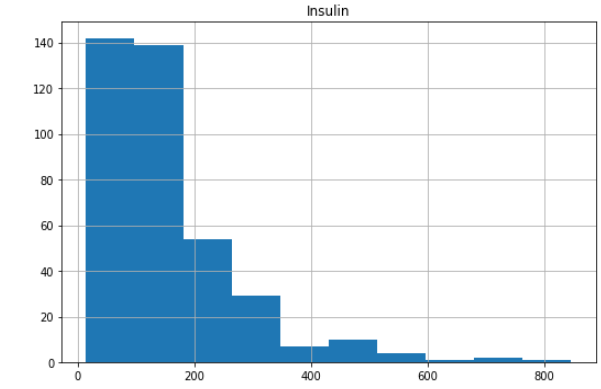
e) Additional Tries: Weight the features before doing KNN prediction.

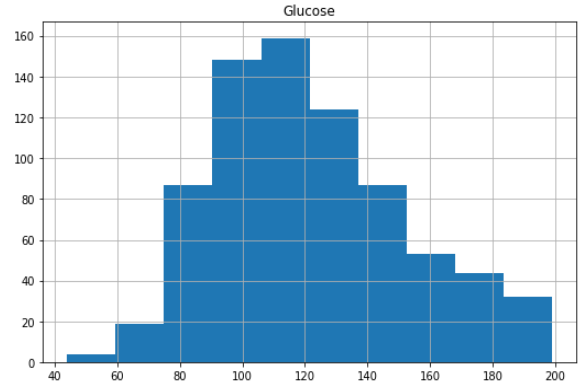
OUTPUT

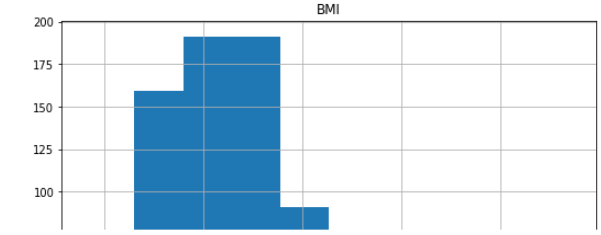

763
768
763
765
764
763
768
762
768
1.0
0.8
0.6
0.4
0.2
0.0
768
614
460
307
153
0
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

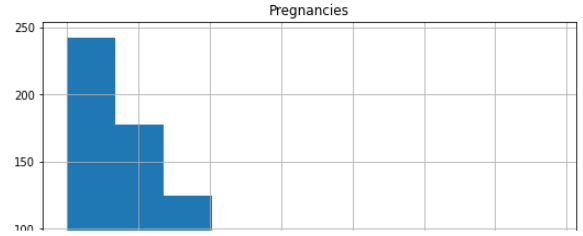

BloodPressure

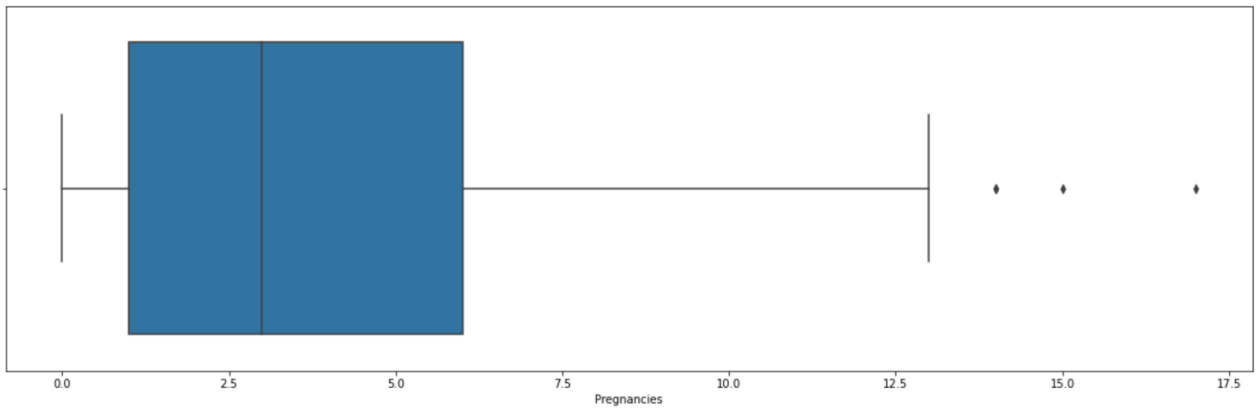

SkinThickness


Insulin

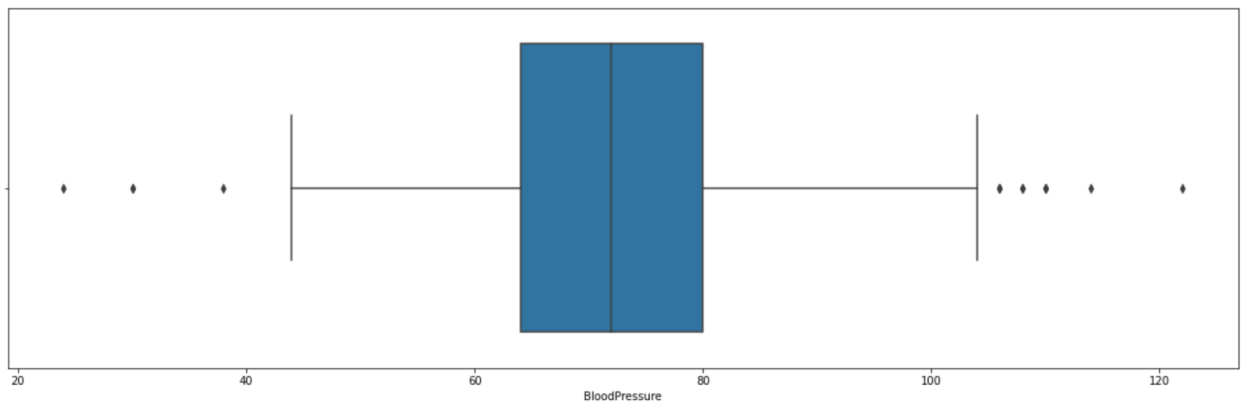

Glucose


BMI

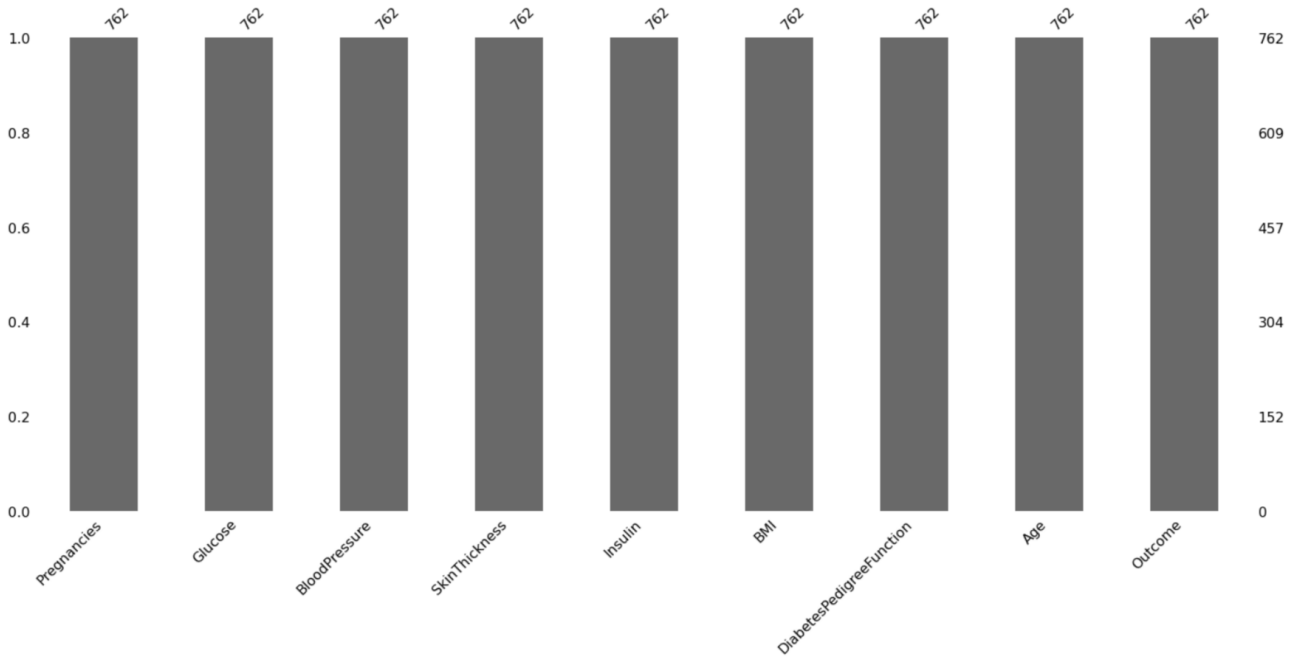

Pregnancies

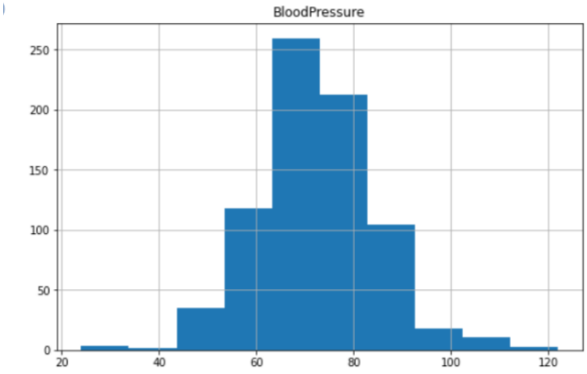


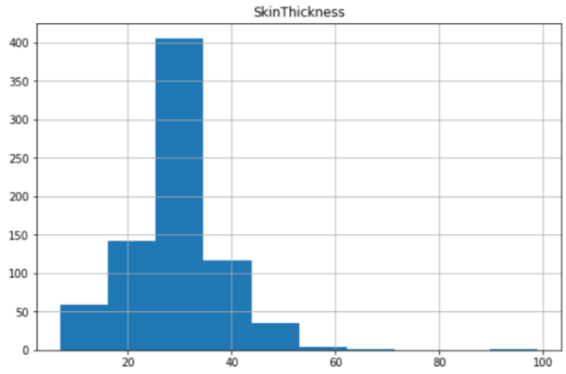
Clearly the distribution is left-skewed, so we go with median imputation of NaN values

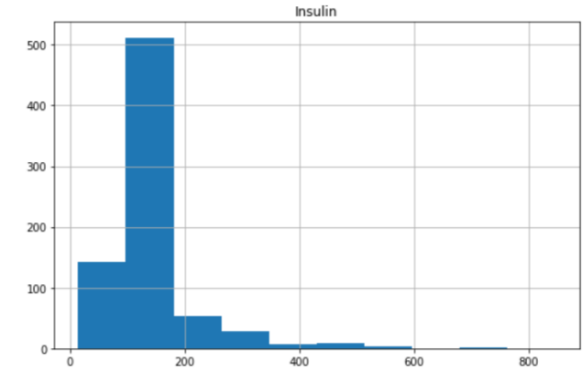


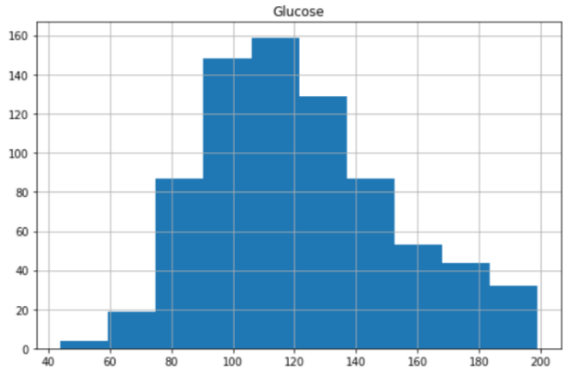
Clearly the distribution is symmetric, so we go with mean imputation of NaN values

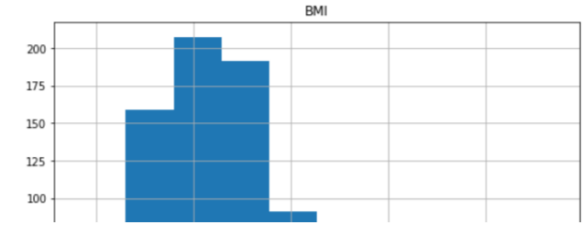


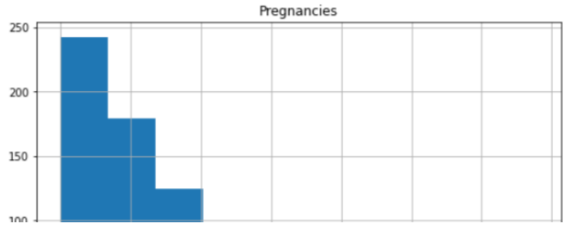


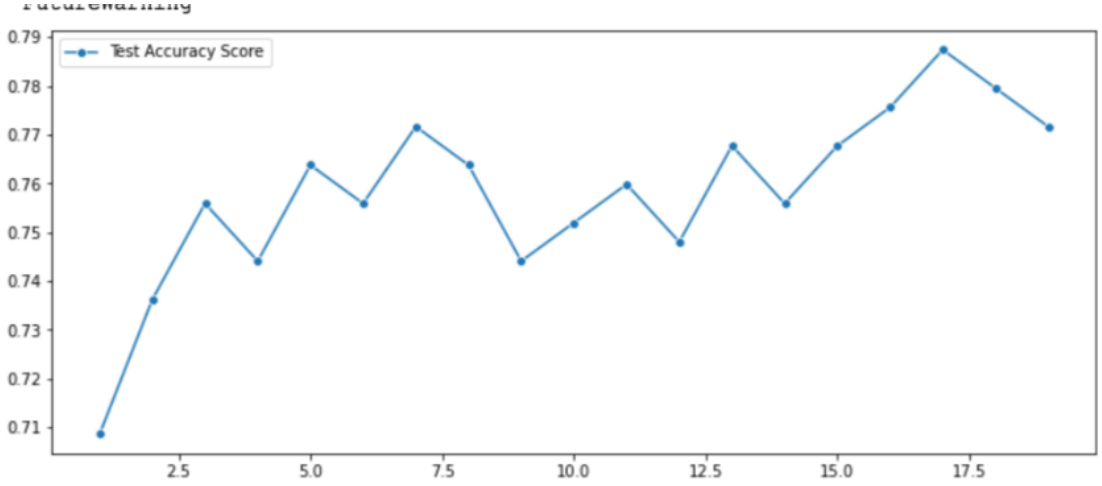












Hence, we have got a knn model with an optimum k value of 17 and accuracy = 78.74%

```
##Importing Data and Necessary Packages
"""
import numpy as np
import pandas as pd
import missingno as msno
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from scipy.stats import mode
from sklearn.metrics import classification_report, accuracy_score

data =
pd.read_csv("https://raw.githubusercontent.com/MSPawanRanjith/FileTransfer/master/diabetes.csv")
data.head()

"""##Pre-Processing Data

###Summary of the available data
"""
data.info(verbose=True)
null_count_plot = msno.bar(data)
"""####Clearly we have some null values in Pregnancies, BloodPressure, SkinThickness, Insulin, BMI, Age columns

#####We are omitting all entries where Age is null, as we can't approximate that value from a mean/median extracted from a distribution plot
"""
data_copy = data.copy(deep = True)
data_copy = data_copy[data_copy["Age"].notnull()]
print("Original Dataset : {} ; After removing null age rows : {}".format(data.shape, data_copy.shape))
"""#####Now, out of the remaining columns, it is clear that the following columns should not have a zero value: -

*   BloodPressure
*   SkinThickness
*   Insulin
```

* Glucose
* BMI

To make handling those values easier, let's replace all instances of "0" with NaN in those columns

"""

```
data_copy[["BloodPressure","SkinThickness","Insulin","Glucose","BMI"]] =
data_copy[["BloodPressure","SkinThickness","Insulin","Glucose","BMI"]].replace(0,
np.NaN)
data_copy.head()
distribution_plot =
data_copy[["BloodPressure","SkinThickness","Insulin","Glucose","BMI","Pregnancie
s"]].hist(figsize = (20,20))
```

"""####Replacing NaN values

#####We are going to use the following imputation tactic for each of these columns: -

1. Boxplot for each column
2. If values in that column are forming a symmetrical distribution with less number of outliers, we use mean to impute the NaN values.
3. If values in the column are forming a skewed distribution with large number of outliers, we use median to impute the NaN values.

#####A) BMI

"""

```
fig, ax = plt.subplots(figsize = (20,6))
sns.boxplot(data_copy["BMI"])
```

"""#####Clearly the distribution is left-skewed, so we go with median imputation of NaN values"""

```
data_copy["BMI"].fillna(data_copy["BMI"].median(), inplace = True)
```

"""B) Glucose"""

```
fig, ax = plt.subplots(figsize = (20,6))
sns.boxplot(data_copy["Glucose"])
```

"""#####Clearly the distribution is symmetric, so we go with mean imputation of NaN values"""

```
data_copy["Glucose"].fillna(data_copy["Glucose"].mean(), inplace = True)

"""C) SkinThickness"""

fig, ax = plt.subplots(figsize = (20,6))
sns.boxplot(data_copy["SkinThickness"])

"""#####Clearly the distribution is left-skewed, so we go with median imputation of NaN values"""

data_copy["SkinThickness"].fillna(data_copy["SkinThickness"].median(), inplace = True)

"""D) Insulin"""

fig, ax = plt.subplots(figsize = (20,6))
sns.boxplot(data_copy["Insulin"])

"""#####Clearly the distribution is left-skewed, so we go with median imputation of NaN values"""

data_copy["Insulin"].fillna(data_copy["Insulin"].median(), inplace = True)

"""E) BloodPressure"""

fig, ax = plt.subplots(figsize = (20,6))
sns.boxplot(data_copy["BloodPressure"])

"""#####Clearly the distribution is symmetric, so we go with mean imputation of NaN values"""

data_copy["BloodPressure"].fillna(data_copy["BloodPressure"].mean(), inplace = True)

"""F) Pregnancies"""

fig, ax = plt.subplots(figsize = (20,6))
sns.boxplot(data_copy["Pregnancies"])

"""#####Clearly the distribution is left-skewed, so we go with median imputation of NaN values"""
```

```
data_copy["Pregnancies"].fillna(data_copy["Pregnancies"].median(), inplace =
True)

"""#### Plotting dataset after NaN imputation"""

final_distribution_plot =
data_copy[["BloodPressure","SkinThickness","Insulin","Glucose","BMI","Pregnancie
s"]].hist(figsize = (20,20))

final_null_plot_count = msno.bar(data_copy)

"""####Scaling the values in each column
#####This is an important step to do in KNN so that one feature with high values
does not overwhelm the impact of a variable with low values
"""

sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(data_copy.drop(["Outcome"],axis = 1),),
        columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'])

X.head()

"""## kNN model and fitting data into the model
### Test-Train Split
"""

y = data_copy["Outcome"]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/3,random_state=42,
stratify=y)

print("X_train = {}, X_test = {}, y_train = {}, y_test = {}".format(X_train.shape,
X_test.shape, y_train.shape, y_test.shape))

"""###Create kNN model
####We are using a class to house the kNN logic. For calculating the distance, we
are using Euclidean Distance. However, using Minkowski or Mahalanobis Distance
is also fine
"""

class knnModel:
```

```
  def euclidean(self, v1, v2):
    # print(v1,v2)
    dist = np.sqrt(np.sum((v1-v2)**2))
    return dist

  def fit(self, X_train, y_train):
    self.X_train = np.array(X_train)
    self.y_train = np.array(y_train)

  def predict(self, X_test, k):
    predicted_outcomes = []

    if(X_test.ndim == 1):
      X_test = np.reshape(X_test,(-1, len(X_test)))
    # print(X_test, X_test[0])

    for test_entry in X_test:
      distances = np.array([self.euclidean(train_data_vector, test_entry) for
train_data_vector in self.X_train])
      #Sorting the array while preserving the index
      #Keeping the first K datapoints
      dist = np.argsort(distances)[:k]

      #Labels of the K datapoints from above
      labels = self.y_train[dist]

      #Majority voting
      label = mode(labels)
      # print(test_entry, "fewfw", dist, labels,"efwef")
      label = label.mode[0]
      predicted_outcomes.append(label)

    return predicted_outcomes

  def __init__(self):
    self.X_train = None
    self.y_train = None

"""### Fit and predict data on kNN model"""
```

```
knn = knnModel()
knn.fit(X_train, y_train)

# for i in range(len(X_test)):
#   x = np.array(X_test)[i]
#   y = np.array(y_test)[i]
#   predicted_val = knn.predict(x,10)
#   print("Predicted Value = {}, Expected Value = {}".format(predicted_val, y))

"""####Find optimal K-value for prediction"""

accuracy_scores = []
for k in range(1,20):
  predicted_vals = knn.predict(np.array(X_test), k)
  accuracy = accuracy_score(y_test, predicted_vals)
  accuracy_scores.append(accuracy)
  # report = classification_report(y_test, predicted_vals)
  print("For k = {}\n".format(k), accuracy)

plt.figure(figsize=(12,5))
p = sns.lineplot(range(1,20),accuracy_scores,marker='o',label='Test Accuracy
Score')

"""### Hence, we have got a knn model with an optimum k value of 17 and
accuracy = 78.74%"""
```