DATA SCIENCE LABORATORY

Course Code	**: 20ISL57**	**Credits**	**: 1.5**
L:T:P	**: 0:0:1.5**	**CIE Marks**	**: 25**
Exam Hours	**: 3**	**SEE Marks**	**: 25**

Course Outcomes: At the end of the Course, the Student will be able to:

CO1	Understand basic operations of NumPy,pandas and Matplotlib
CO2	Implement Regression models for the sample datasets
CO3	Develop classification models and optimize the performance
CO4	Develop clustering models and apply on suitable datasets

Mapping of Course Outcomes to Program Outcomes:

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	3	2	3	-	1	1	-	1	-	1
CO2	3	3	3	2	3	-	1	1	-	1	-	1
CO3	3	3	3	2	3	-	1	1	-	1	-	1
CO4	3	3	3	2	3	-	2	1	-	1	-	1

Experiment No.	**Experiment**
	PART-A
1	Using pandas in python demonstrate the following operations for the sample dataset given, i) Indexing of Dataframe ii) Grouping and aggregating iii) Adding and removing attributes iv) Joining dataframes v) Filtering the data vi) Handling Missing values
2	Using pandas and Matplotlib demonstrate the following operations for the sample dataset given, i) Bar chart and Histogram

	ii) Comparing Distribution iii) Box plot and mention quartiles iv) Correlation using pairplot and heatmap v)
3	Using Numpy,pandas and Matplotlib demonstrate the following operations for the sample dataset given, i) Central tendency ii) Dispersion and Distribution iii) ANOVA iv) Hypothesis testing
4	Develop a program to implement Simple Linear Regression model and evaluate the model by verifying the performance
5	Develop a program to implement Multiple Linear Regression model and evaluate the model by verifying the performance
6	Develop a program to implement Logistic Regression and indicate the class label for the test dataset
PART-B	
7	Develop a program to implement Naive Bayes classifier model and analyze the model using confusion matrix
8	Develop a program to implement Decision Tree model and analyze the model using confusion matrix
9	Develop a program to implement Random Forest classifier model and analyze the model using confusion matrix
10	Develop a program to implement KNN classifier model and analyse the model using confusion matrix
11.	Develop a program to implement K Means clustering model for the given value of K, where K is number of clusters
12	Develop a program to implement Hierarchical clustering model for the given value of N, where N is number of clusters

For SEE Examination:

- One experiment from part A & One experiment from part B to be given
- Examination will be conducted for 50 marks and scaled down to 25 marks
- Marks Distribution : Procedure write-up – 20%
  Conduction – 60%
  Viva – Voce – 20%

- Change of the experiment is allowed only once and procedure write-up marks will be considered as ‘0’

CIE - Continuous Internal Evaluation (25 Marks)

Bloom’s Category	**Tests (25 Marks)**
Remember	-
Understand	5
Apply	5
Analyze	5
Evaluate	-
Create	10

SEE – Semester End Examination (25 Marks)

Bloom’s Taxonomy	**Mark s**
Remember	-
Understand	5
Apply	5
Analyze	5
Evaluate	-
Create	10

1. Using pandas in python demonstrate the following operations for the sample dataset given,
   i) Indexing of Dataframe
   ii) Grouping and aggregating
   iii) Adding and removing attributes
   iv) Joining dataframes
   v) Filtering the data
   vi) Handling Missing values

```
# # PRELIMNIRAY INFORMATION
import pandas as pd
ipldf=pd.read_csv('IPL IMB381IPL2013.csv')
pd.set_option('display.max_columns',10)
ipldf.head()
names = list(ipldf.columns)
names
ipldf.shape
ipldf.info()
# # INDEXING
ipldf.loc[0:9]
ipldf[['PLAYER NAME','COUNTRY']][0:9]
ipldf.iloc[4:9,1:4]
# # GROUPING AND AGGREGATING
sold_price_by_age = ipldf.groupby('AGE')['SOLD
PRICE'].mean().reset_index()
sold_price_by_age
# # ADDING AND REMOVING COLUMNS
ipldf['PREMIUM']=ipldf['SOLD PRICE']-ipldf['BASE PRICE']
ipldf[['PLAYER NAME','BASE PRICE','SOLD PRICE','PREMIUM']][0:5]
#ipldf.shape
ipldf = ipldf.drop('ECON',axis = 1)
ipldf.shape
# # MERGING DATAFRAME
sold_price_by_age_role = ipldf.groupby(['AGE','PLAYING ROLE'])['SOLD
PRICE'].mean().reset_index()
```

```
sold_price_by_age_role
soldprice_comparison=sold_price_by_age_role.merge(sold_price_by_age,on
= 'AGE',how = 'outer')
soldprice_comparison
soldprice_comparison.rename(columns =
{'SOLD PRICE_x':'SOLD_PRICE_AGE_ROLE','SOLD
PRICE_y':'SOLD_PRICE_AGE'},
                            inplace = True)
soldprice_comparison

# # FILTERING
soldprice_comparison['CHANGE']=soldprice_comparison.apply(lambda
x:(x.SOLD_PRICE_AGE_ROLE-x.SOLD_PRICE_AGE)/x.SOLD_PRICE_AGE,axis = 1)
soldprice_comparison
soldprice_comparison[soldprice_comparison.CHANGE > 0]
# # HANDLING NULL VALUES
soldprice_comparison[soldprice_comparison.CHANGE < 0]= None
soldprice_comparison
soldprice_comparison[soldprice_comparison.CHANGE.isnull()]
soldprice_comparison = soldprice_comparison.dropna(subset =
['CHANGE'])
soldprice_comparison[soldprice_comparison.CHANGE.isnull()]
```

SAMPLE OUTPUTS :

	SI.NO.	PLAYER NAME	AGE	COUNTRY	TEAM	PLAYING ROLE	T-RUNS	T-WKTS	ODI-RUNS-S	ODI-SR-B	...	SR-B
0	1	Abdulla, YA	2	SA	KXIP	Allrounder	0	0	0	0.00	...	0.00
1	2	Abdur Razzak	2	BAN	RCB	Bowler	214	18	657	71.41	...	0.00
2	3	Agarkar, AB	2	IND	KKR	Bowler	571	58	1269	80.62	...	121.01
3	4	Ashwin, R	1	IND	CSK	Bowler	284	31	241	84.56	...	76.32
4	5	Badrinath, S	2	IND	CSK	Batsman	63	0	79	45.93	...	120.71

5 rows × 26 columns

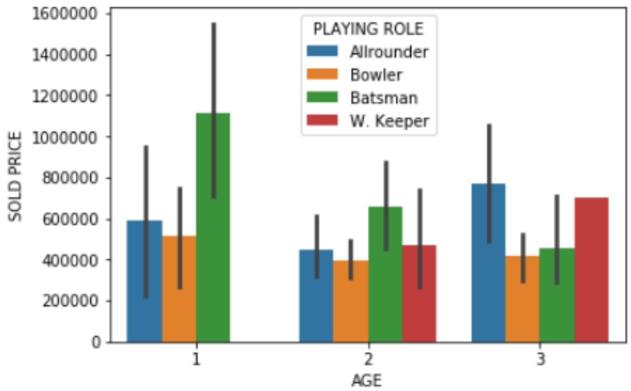
	AGE	PLAYING ROLE	SOLD_PRICE_AGE_ROLE	SOLD_PRICE_AGE	CHANGE
0	1	Allrounder	5.875000e+05	720250.000000	-0.184311
1	1	Batsman	1.110000e+06	720250.000000	0.541132
2	1	Bowler	5.177143e+05	720250.000000	-0.281202
3	2	Allrounder	4.494000e+05	484534.883721	-0.072513
4	2	Batsman	6.547619e+05	484534.883721	0.351320
5	2	Bowler	3.979310e+05	484534.883721	-0.178736
6	2	W. Keeper	4.677273e+05	484534.883721	-0.034688
7	3	Allrounder	7.666667e+05	520178.571429	0.473853
8	3	Batsman	4.576923e+05	520178.571429	-0.120125
9	3	Bowler	4.143750e+05	520178.571429	-0.203399
10	3	W. Keeper	7.000000e+05	520178.571429	0.345692

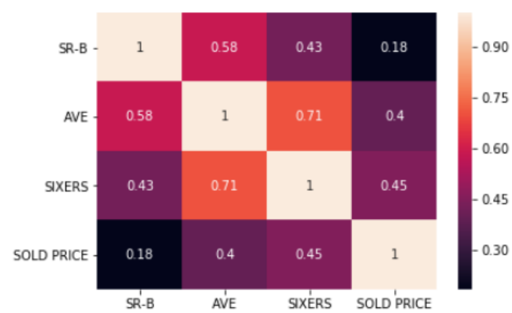
2. Using pandas and Matplotlib demonstrate the following operations for the sample dataset given, i) Bar chart and Histogram ii) Comparing Distribution iii) Box plot and mention quartiles iv) Correlation using pairplot and heatmap

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')
ipldf = pd.read_csv('IPL IMB381IPL2013.csv')
ipldf.head()
# BAR CHART
sns.barplot(x='AGE',y='SOLD PRICE',data = ipldf)
sns.barplot(x='AGE',y='SOLD PRICE',data = ipldf,hue='PLAYING ROLE')
# HISTOGRAM
plt.hist(ipldf['SOLD PRICE'])
plt.hist(ipldf['SOLD PRICE'],bins = 20)
# DIstribution or Density Chart
sns.distplot(ipldf['SOLD PRICE'])
box = sns.boxplot(ipldf['SOLD PRICE'])
box = plt.boxplot(ipldf['SOLD PRICE'])
[item.get_ydata()[0] for item in box['caps']]
[item.get_ydata()[0] for item in box['whiskers']]
[item.get_ydata()[0] for item in box['medians']]
ipldf[ipldf['SOLD PRICE']>1350000.0][['PLAYER NAME','PLAYING
ROLE','SOLD PRICE']]
# Comparing Distributions
sns.distplot(ipldf[ipldf['CAPTAINCY EXP']==1]['SOLD
PRICE'],color='y',label = 'Captaincy Experience')
sns.distplot(ipldf[ipldf['CAPTAINCY EXP']==0]['SOLD
PRICE'],color='r',label = 'No Captaincy Experience')
```

```
plt.legend()
sns.boxplot(x='PLAYING ROLE',y='SOLD PRICE',data = ipldf)
# Scatter Plot
ipldf_batsman = ipldf[ipldf['PLAYING ROLE']=='Batsman']
plt.scatter(x=ipldf_batsman.SIXERS,y=ipldf_batsman['SOLD PRICE'])
sns.regplot(x='SIXERS',y='SOLD PRICE',data =ipldf_batsman)
# Pair Plot
infl_features = ['SR-B','AVE','SIXERS','SOLD PRICE']
sns.pairplot(ipldf[infl_features],size = 2)
# Correlation and HeatMap
ipldf[infl_features].corr()
sns.heatmap(ipldf[infl_features].corr(),annot = True)
```

SAMPLE OUTPUT :





3. Using Numpy,pandas and Matplotlib demonstrate the following operations for the sample dataset given,
   - i) Central tendency
   - ii) Dispersion and Distribution
   - iii) ANOVA
   - iv) Hypothesis testing

```
# # PRELIMINARY INFORMATION
import pandas as pd
import numpy as np
df=pd.read_excel('IBM-313 Marks.xlsx')
print(df.head())
print(df.columns)
# # CENTRAL TENDENCY
import scipy
from scipy import stats
data = df['Total']
#print('MEAN = ',np.mean(df['Total']))
#print('MEDIAN = ',np.median(df['Total']))
print('MEAN = ',scipy.mean(data))
print('MEDIAN = ',scipy.median(data))
print('MODE = ',stats.mode(data))
```

```
from scipy import stats
x=df['Total']
y=np.array(x)
print('Percentile = ',np.percentile(y,30))
# # DISPERSION
range=max(y)-min(y)
print("RANGE = ",range)
Q1 = np.percentile(y,25)
Q3 = np.percentile(y,75)
print("IQR = ",Q3-Q1)
print("VARIANCE = ",np.var(y))
import statistics
print("POPULATION STANDARD DEVIATION = ",statistics.pstdev(y))
print("SAMPLE STANDARD DEVIATION = ",statistics.stdev(y))
from scipy.stats import skew
print(skew(y))
# # HYPOTHESIS TESTING
pp_df=pd.read_excel('passport.xlsx')
pp_df.head()
print(list(pp_df.processing_time))
import math
def z_test(p_mean,p_std,sample):
    z_score = (sample.mean() - p_mean)/(p_std/math.sqrt(len(sample)))
    return z_score,stats.norm.cdf(z_score)
z_test(30,12.5,pp_df.processing_time)
# # ANOVA
anova_df = pd.read_excel('discounts.xlsx')
anova_df.head()
import seaborn as sns
sns.distplot(anova_df['discount_0'],label = 'No Discount')
sns.distplot(anova_df['discount_10'],label = '10% Discount')
sns.distplot(anova_df['discount_20'],label = '20% Discount')
plt.legend()
from scipy.stats import f_oneway
f_oneway(anova_df['discount_0'],anova_df['discount_10'],anova_df['disc
ount_20'])
```

SAMPLE OUTPUT:

```
POPULATION STANDARD DEVIATION =  16.210536046955966
SAMPLE STANDARD DEVIATION =  16.31411880088133
(-1.4925950555994747, 0.06777160919961511)
F_onewayResult(statistic=65.86986401283694, pvalue=3.821500669725641e-18)
```

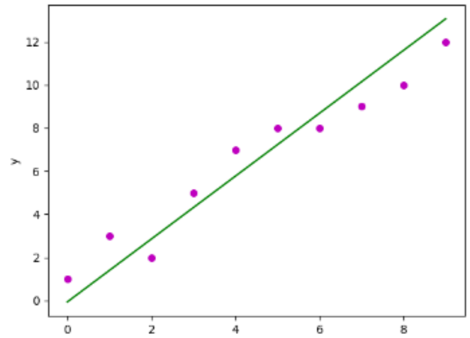
4. Develop a program to implement Simple Linear Regression model and evaluate the model by verifying the performance

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
      # number of observations/points
      n = np.size(x)
```

```
      # mean of x and y vector
      m_x, m_y = np.mean(x), np.mean(y)
      # calculating cross-deviation and deviation about x
      SS_xy = np.sum(y*x) - n*m_y*m_x
      SS_xx = np.sum(x*x) - n*m_x*m_x
      # calculating regression coefficients
      b_1 = SS_xy / SS_xx
      b_0 = m_y - b_1*m_x
      return(b_0, b_1)
def plot_regression_line(x, y, b):
      # plotting the actual points as scatter plot
      plt.scatter(x, y, color = "m",
                  marker = "o", s = 30)
      # predicted response vector
      y_pred = b[0] + b[1]*x
      # plotting the regression line
      plt.plot(x, y_pred, color = "g")
      # putting labels
      plt.xlabel('x')
      plt.ylabel('y')
      # function to show plot
      plt.show()
def main():
      # observations
      x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
      y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
      # estimating coefficients
      b = estimate_coef(x, y)
      print("Estimated coefficients:\nb_0 = {} \
            \nb_1 = {}".format(b[0], b[1]))
      # plotting regression line
      plot_regression_line(x, y, b)
if __name__ == "__main__":
      main()
```

SAMPLE OUTPUT:

```
Estimated coefficients:
b_0 = -0.0586206896552
b_1 = 1.45747126437
```



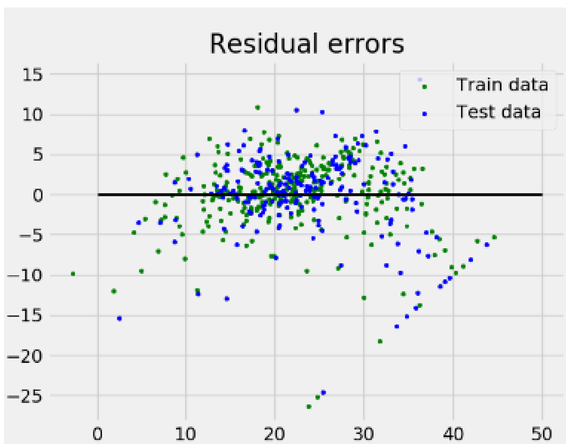
5. Develop a program to implement Multiple Linear Regression model and evaluate the model by verifying the performance

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics
# load the boston dataset
boston = datasets.load_boston(return_X_y=False)
# defining feature matrix(X) and response vector(y)
X = boston.data
y = boston.target
# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4,random_state=1)
# create linear regression object
reg = linear_model.LinearRegression()
# train the model using the training sets
reg.fit(X_train, y_train)
# regression coefficients
print('Coefficients: \n', reg.coef_)
# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test,
y_test)))
# plot for residual error
## setting plot style
plt.style.use('fivethirtyeight')
## plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) -
y_train,
               color = "green", s = 10, label = 'Train data')
## plotting residual errors in test data
```

```
plt.scatter(reg.predict(X_test),    reg.predict(X_test)    -
y_test,
                color = "blue", s = 10, label = 'Test data')
## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)
## plotting legend
plt.legend(loc = 'upper right')

## plot title
plt.title("Residual errors")
## function to show plot
plt.show()
```

SAMPLE OUTPUT:

```
Coefficients:
[ -8.80740828e-02   6.72507352e-02   5.10280463e-02   2.18879172e+00
-1.72283734e+01   3.62985243e+00   2.13933641e-03  -1.36531300e+00
2.88788067e-01  -1.22618657e-02  -8.36014969e-01   9.53058061e-03
-5.05036163e-01]
Variance score: 0.720898784611
```



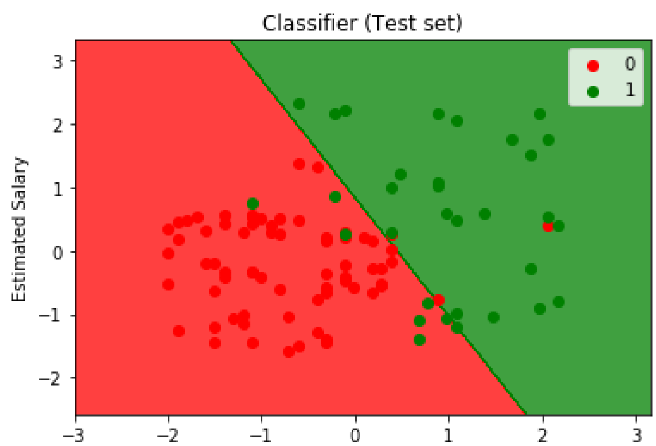
6. Develop a program to implement Logistic Regression and indicate the class label for the test dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset = pd.read_csv('...\\User_Data.csv')
# input
x = dataset.iloc[:, [2, 3]].values

# output
y = dataset.iloc[:, 4].values
```

```
from sklearn.cross_validation import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(
         x, y, test_size = 0.25, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)
print (xtrain[0:10, :])
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
y_pred = classifier.predict(xtest)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, y_pred)
print ("Confusion Matrix : \n", cm)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(ytest, y_pred))
```

SMPLE OUTPUT:

```
Accuracy :  0.89
```



7. Develop a program to implement Naive Bayes classifier model and analyze the model using confusion matrix

```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target
# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,
test_size=0.4, random_state=1)
# training the model on training set
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# making predictions on the testing set
y_pred = gnb.predict(X_test)
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)
```

SAMPLE OUTPUT:

```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

8. Develop a program to implement Decision Tree model and analyze the model using confusion matrix

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Function importing Dataset
def importdata():
 balance_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
 sep= ',', header = None)
 # Printing the dataswet shape
 print ("Dataset Length: ", len(balance_data))
 print ("Dataset Shape: ", balance_data.shape)
 # Printing the dataset obseravtions
 print ("Dataset: ",balance_data.head())
 return balance_data
# Function to split the dataset
def splitdataset(balance_data):
 # Separating the target variable
      X = balance_data.values[:, 1:5]
      Y = balance_data.values[:, 0]
      # Splitting the dataset into train and test
      X_train, X_test, y_train, y_test = train_test_split(
      X, Y, test_size = 0.3, random_state = 100)
      return X, Y, X_train, X_test, y_train, y_test
 # Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
```

```
 # Creating the classifier object
       clf_gini = DecisionTreeClassifier(criterion = "gini",
             random_state = 100,max_depth=3, min_samples_leaf=5)
       # Performing training
       clf_gini.fit(X_train, y_train)
       return clf_gini
 # Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):
 # Decision tree with entropy
       clf_entropy = DecisionTreeClassifier(criterion = "entropy",
random_state = 100,max_depth = 3, min_samples_leaf = 5)
 # Performing training
       clf_entropy.fit(X_train, y_train)
       return clf_entropy
# Function to make predictions
def prediction(X_test, clf_object):
 # Predicton on test with giniIndex
       y_pred = clf_object.predict(X_test)
       print("Predicted values:")
       print(y_pred)
       return y_pred
 # Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
       print("Confusion Matrix: ",
       confusion_matrix(y_test, y_pred))
       print ("Accuracy : ",
       accuracy_score(y_test,y_pred)*100)
       print("Report : ",
       classification_report(y_test, y_pred))

# Driver code
def main():
       # Building Phase
       data = importdata()
       X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
       clf_gini = train_using_gini(X_train, X_test, y_train)
       clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
       # Operational Phase
       print("Results Using Gini Index:")
       # Prediction using gini
       y_pred_gini = prediction(X_test, clf_gini)
       cal_accuracy(y_test, y_pred_gini)

       print("Results Using Entropy:")
       # Prediction using entropy
```

```
      y_pred_entropy = prediction(X_test, clf_entropy)
      cal_accuracy(y_test, y_pred_entropy)
# Calling main function
if __name__=="__main__":
main()
```

SAMPLE OUTPUT:

```
Data Infomation:


Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
```

9. Develop a program to implement Random Forest classifier model and analyze the model using confusion matrix

```
# importing required libraries
# importing Scikit-learn library and datasets package
from sklearn import datasets
# Loading the iris plants dataset (classification)
iris = datasets.load_iris()
print(iris.target_names)
print(iris.feature_names)
# dividing the datasets into two parts i.e. training datasets and
test datasets
X, y = datasets.load_iris( return_X_y = True)
# Spliting arrays or matrices into random train and test subsets
from sklearn.model_selection import train_test_split
# i.e. 80 % training dataset and 30 % test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.70)
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
# creating dataframe of IRIS dataset
data = pd.DataFrame({'sepallength': iris.data[:, 0], 'sepalwidth':
iris.data[:, 1],
                           'petallength':    iris.data[:,    2],
'petalwidth': iris.data[:, 3],
```

```
                              'species': iris.target})
# printing the top 5 datasets in iris dataset
print(data.head())
# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)
# Training the model on the training dataset
# fit function is used to train the model using the training sets
as parameters
clf.fit(X_train, y_train)
# performing predictions on the test dataset
y_pred = clf.predict(X_test)
# metrics are used to find accuracy or error
from sklearn import metrics
print()
# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test,
y_pred))
```

SAMPLEL OUTPUT :

```
ACCURACY OF THE MODEL: 0.9238095238095239
```

10. Develop a program to implement KNN classifier model and analyse the model using confusion matrix

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("Data", index_col = 0)
df.head()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('TARGET CLASS', axis = 1))
scaled_features = scaler.transform(df.drop('TARGET CLASS', axis =
1))
df_feat = pd.DataFrame(scaled_features, columns = df.columns[:-1])
df_feat.head()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
  scaled_features, df['TARGET CLASS'], test_size = 0.30)
# Remember that we are trying to come up
# with a model to predict whether
# someone will TARGET CLASS or not.
# We'll start with k = 1.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
```

```
# Predictions and Evaluations
# Let's evaluate our KNN model !
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

SAMPLLE OUTPUT:

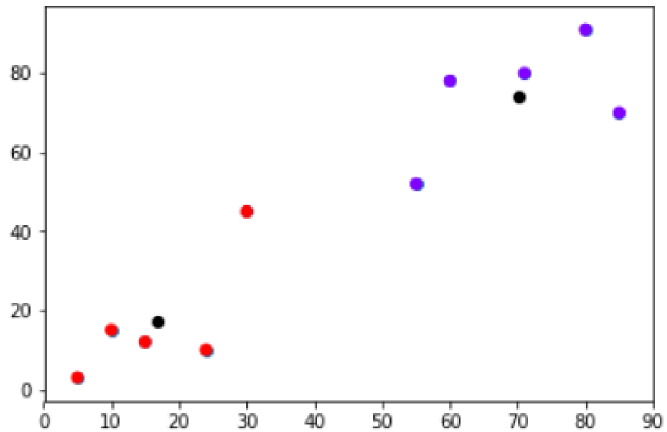
```
[[133  16]
 [ 15 136]]

              precision    recall  f1-score   support

           0       0.90      0.89      0.90       149
           1       0.89      0.90      0.90       151

    accuracy                           0.90       300
   macro avg       0.90      0.90      0.90       300
weighted avg       0.90      0.90      0.90       300
```

11. Develop a program to implement K Means clustering model for the given value of K, where K is number of clusters

```
import matplotlib.pyplot as plt
#matplotlib inline
import numpy as np
from sklearn.cluster import KMeans
X = np.array([[5,3],
     [10,15],
     [15,12],
     [24,10],
     [30,45],
     [85,70],
     [71,80],
     [60,78],
     [55,52],
     [80,91],])
plt.scatter(X[:,0],X[:,1], label='True Position')
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0]
,kmeans.cluster_centers_[:,1], color='black')
```

SAMPEL OUTPUT:

```
[[70.2 74.2]
 [16.8 17. ]]
[1 1 1 1 1 0 0 0 0 0]
```



12. Develop a program to implement Hierarchical clustering model for the given value of N, where N is number of clusters

```
from sklearn.cluster import AgglomerativeClustering
import numpy as np
# randomly chosen dataset
X = np.array([[1, 2], [1, 4], [1, 0],
            [4, 2], [4, 4], [4, 0]])
# here we need to mention the number of clusters
# otherwise the result will be a single cluster
# containing all the data
clustering = AgglomerativeClustering(n_clusters = 2).fit(X)
# print the class labels
print(clustering.labels_)
```

SAMPLE OUTPUT:

```
[1, 1, 1, 0, 0, 0]
```