PART A

1 #FIND S PROG:

```
import pandas as pd
import numpy as np
          #to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")
          #making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)
          #segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)
          #training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis
          #obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

2 #CANDIDATE ELIMINATION PROG:

```
import csv
with open("data.csv") as f:
    csv_file=csv.reader(f)
    data=list(csv_file)
    s=data[1][:-1] #1st record is taken as specific hypothesis
    g=[['?' for i in range(len(s))] for j in range(len(s))] #? is put for all the records

    for i in data:
        if i[-1]=="Yes": # if target attribute is yes, then it should match specific hypothesis
            for j in range(len(s)):
                if i[j]!=s[j]: # if record doesn't match specific hypothesis, put ? in specific columns
                    s[j]='?'
                    g[j][j]='?'
        elif i[-1]=="No": # if target attribute is no, then specific hypothesis should not match
            for j in range(len(s)):
                if i[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]="?"
        print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
        print(s)
        print(g)
    gh=[]
    for i in g:
        for j in i:
            if j!='?':
                gh.append(i)
                break
    print("\nFinal specific hypothesis:\n",s)
    print("\nFinal general hypothesis:\n",gh)
```

3 #ID3 / DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
from sklearn.metrics import confusion_matrix

dataset=pd.read_csv('pima-indians-Diabetes.csv')
print(dataset.head())
train_features=dataset.iloc[:80,:-1]
test_features=dataset.iloc[80:,:-1]
train_targets=dataset.iloc[:80,-1]
test_targets=dataset.iloc[80:,-1]
tree1=DecisionTreeClassifier(criterion='entropy').fit(train_features,train_targets)

prediction=tree1.predict(test_features)
cm=confusion_matrix(test_targets,prediction)
print('Confusion Matrix:\n',cm)
TP=cm[0][0]
FP=cm[0][1]
FN=cm[1][0]
TN=cm[1][1]
print('False Positive \n {}'.format(FP))
print('False Negative \n {}'.format(FN))
print('True Positive \n {}'.format(TP))
print('True Negative \n {}'.format(TN))
TPR=TP/(TP+FN)
print('Senstivity \n {}'.format(TPR))
TNR=TN/(TN+FP)
print('Specificity \n {}'.format(TNR))
Precision=TP/(TP+FP)
print('Precision \n {}'.format(Precision))
```

```
Recall=TP/(TP+FN)

print('Recall \n {}'.format(Recall))

Acc=(TP+TN)/(TP+TN+FP+FN)

print('Accuracy \n {}'.format(Acc))

Fscore=2*(Precision*Recall)/(Precision+Recall)

print('FScore \n {}'.format(Fscore))
```

4.A #BINARIZATION

```
# Python code explaining how to Binarize feature values
#Importing Libraries
import pandas as pd
          #Importing Data
data_set = pd.read_csv('data.csv')
print(data_set.head())
age = data_set.iloc[:, 1].values
salary = data_set.iloc[:, 2].values
print ("\nOriginal age data values : \n",  age)
print ("\nOriginal salary data values : \n",  salary)
          # Binarizing values
from sklearn.preprocessing import Binarizer


x = age
x = x.reshape(1, -1)
y = salary
y = y.reshape(1, -1)
          # For age, let threshold be 35, For salary, let threshold be 61000
binarizer_1 = Binarizer(35) # Age below 35 is binarized to 0
binarizer_2 = Binarizer(61000) # Salary below 61000 is binarized to 0
          # Transformed features
print ("\nBinarized age : \n", binarizer_1.fit_transform(x))
print ("\nBinarized salary : \n", binarizer_2.fit_transform(y))
```

4.B #BINNING

```
import pandas as pd
import numpy as np
        # Generate 20 random integers uniformly between 0 and 99
small_counts = np.random.randint(0, 100, 20)
print(small_counts)
        # Map to evenly spaced bins 0-9 by division
print(np.floor_divide(small_counts, 10))


large_counts = [296, 8286, 64011, 80, 3, 725, 867, 2215, 7689, 11495, 91897, 44, 28, 7971,
926, 12]
        # print(np.floor(np.log10(large_counts)))
        #Map the counts to quartiles into 4 bins (quartile)
print(pd.qcut(large_counts, 4, labels=False))
        #convert large_counts into series data
large_counts_series = pd.Series(large_counts)
        # print(large_counts_series)
print(large_counts_series.quantile([0.25, 0.5, 0.75]))
```

4.C #LOG TRANSFORMATION

```
import pandas as pd
import numpy as np
        #Log Transform Example
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})
data['log+1'] = (data['value']+1).transform(np.log)
print(data)
        #Negative Values Handling. Note that the values are different
data['log'] = (data['value']-data['value'].min()+1) .transform(np.log)
print(data)
```

4.D #FEATURE SCALING

```
        #Importing Libraries
import pandas as pd
        # Sklearn library
from sklearn import preprocessing
        #Import Data
data_set = pd.read_csv('data.csv')
print(data_set.head())
        # here Features - Age and Salary columns
        # are taken using slicing
        # to handle values with varying magnitude
x = data_set.iloc[:, 1:3].values
print ("\nOriginal data values : \n", x)
        #MIN MAX SCALER
min_max_scaler = preprocessing.MinMaxScaler(feature_range =(0, 1))
        # Scaled feature
x_after_min_max_scaler = min_max_scaler.fit_transform(x)
print ("\nAfter min max Scaling : \n", x_after_min_max_scaler)
        # STANDARDIZATION
Standardisation = preprocessing.StandardScaler()
# Scaled feature
x_after_Standardisation = Standardisation.fit_transform(x)
print ("\nAfter Standardisation : \n", x_after_Standardisation)
```

PART B

1 #PCA

```
        # https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
        # load dataset into Pandas DataFrame
        #data = pd.read_csv("data5.csv")
        #df = pd.read_csv("Iris.csv", names=['sepal length','sepal width','petal length','petal
        width','target'])
df = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal width','target'])
print(df.head)

from sklearn.preprocessing import StandardScaler
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
        # Separating out the features
x = df.loc[:, features].values
        # Separating out the target
y = df.loc[:,['target']].values
        # Standardizing the features
x = StandardScaler().fit_transform(x)
print("Standardized features ", x)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x) #Apply PCA
principalDf = pd.DataFrame(data = principalComponents,
columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, df[['target']]], axis = 1)

print(pca.explained_variance_ratio_)
```

2 #K MEANS

```
        #Import libraries
import pandas as pd
from sklearn.cluster import KMeans
        #import the dataset
df = pd.read_csv('iris.csv')
print(df.head(10))

x = df.iloc[:, [0,1,2,3]].values # Load Input Attributes

kmeans5 = KMeans(n_clusters=5) # Create 5 Clusters
y_kmeans5 = kmeans5.fit_predict(x)
print(y_kmeans5) #Prints the clusters for each record

print(kmeans5.cluster_centers_)

Error =[]
for i in range(1, 11):
   kmeans = KMeans(n_clusters = i).fit(x)
   kmeans.fit(x)
   Error.append(kmeans.inertia_)
import matplotlib.pyplot as plt
plt.plot(range(1, 11), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```

3 #NAIVE BAYES

```
from sklearn import datasets


iris = datasets.load_iris()


print("Features: ", iris.feature_names)


print("Labels: ", iris.target_names)


from sklearn.model_selection import train_test_split


x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3,
random_state=109)


from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()


gnb.fit(x_train, y_train)


y_pred = gnb.predict(x_test)


from sklearn import metrics



print("Accuracy: " ,metrics.accuracy_score(y_test, y_pred))
```

4 #ADA BOOST CLAASIFIER

```
        #https://www.datacamp.com/community/tutorials/adaboost-classifier-python
        # Load libraries
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

iris = datasets.load_iris()
X = iris.data
y = iris.target
        # Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
        # Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50,
                learning_rate=1)
        # Train Adaboost Classifer
model = abc.fit(X_train, y_train)
        #Predict the response for test dataset
y_pred = model.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

        #Using Different Base Learners
        # Import Support Vector Classifier
from sklearn.svm import SVC
svc=SVC(probability=True, kernel='linear')
```

```
          # Create adaboost classifer object
abc =AdaBoostClassifier(n_estimators=50, base_estimator=svc,learning_rate=1)
          # Train Adaboost Classifer
model = abc.fit(X_train, y_train)
          #Predict the response for test dataset
y_pred = model.predict(X_test)
          # Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

5 #RANDOM FOREST

```
import pandas as pd
import numpy as np
dataset = pd.read_csv('petrol_consumption.csv')
print(dataset.head())

X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
        # Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```