WEEK 1

sky	Airtemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	change	No
Sunny	Warm	High	Strong	Cool	change	Yes

```
#importing the packages
import io
import csv
import pandas as pd
num_attributes = 6
a = []
print("The given Dataset is")
with open('Climate.csv','r') as csvfile:
  reader = csv.reader(csvfile)
  for row in reader:
    a.append(row)
    print(row)
print("The initial Hypothesis is")
hypothesis=['0']*num_attributes
print(hypothesis)
for j in range(0,num_attributes):
  hypothesis[j]=a[1][j]
print(hypothesis)
print("FIND S: Finding maximal Specific Hypothesis")
for i in range(1,len(a)):
  if a[i][num_attributes]=='Yes' or a[i][num_attributes]=='yes':
    for j in range(0,num_attributes):
      if a[i][j]!=hypothesis[j]:
        hypothesis[j]='?'
      else:
        hypothesis[j]=a[i][j]
  print("For Training instance No: {0} the hypothesis is ".format(i),hypothesis)
```

WEEK-2

```
import pandas as pd
df=pd.read_csv("week2.csv")
print(df)
def entropy(probs):
  import math
  return sum(-prob*math.log(prob,2)for prob in probs)
def entropy_of_list(a_list):
  from collections import Counter
  cnt = Counter (x for x in a_list)
  num_instances =len(a_list)
  probs=[x/num_instances for x in cnt.values()]
  return entropy(probs)
total_entropy= entropy_of_list(df['PlayTennis'])
print(total_entropy)
def information_gain(df,split_attribute_name, target_attribute_name, trace=0):
  df_split =df.groupby(split_attribute_name)
  for name,group in df_split:
    nobs=len(df.index)*1.0
    df_agg_ent=df_split.agg({target_attribute_name: [entropy_of_list,lambda x:
len(x)/nobs] })[target_attribute_name]
    avg_info=sum(df_agg_ent['entropy_of_list'] * df_agg_ent['<lambda_0>'])
    old_entropy=entropy_of_list(df[target_attribute_name])
    return old_entropy-avg_info
def id3DT(df, target_attribute_name, attribute_names, default_class=None):
  from collections import Counter
  cnt = Counter(x for x in df[target_attribute_name])
  if len(cnt)==1:
    return next(iter(cnt))
  elif df.empty or (not attribute_names):
    return default_class
  else:
    default_class =max(cnt.keys())
    gainz=[information_gain(df,attr, target_attribute_name) for attr in
attribute_names]
    index_of_max=gainz.index(max(gainz))
    best_attr=attribute_names[index_of_max]
    tree={best_attr:{}}
    remaining_attributes_names=[i for i in attribute_names if i != best_attr]
    for attr_val, data_subset in df.groupby(best_attr):

subtree=id3DT(data_subset,target_attribute_name,remaining_attributes_names,
default_class)
      tree[best_attr][attr_val]=subtree
    return tree
attribute_names=list(df.columns)
attribute_names.remove('PlayTennis')
from pprint import pprint
tree= id3DT(df,'PlayTennis',attribute_names)
print("The Resultant Decision Tree is ")
pprint(tree)
attribute=next(iter(tree))
print("Best Attribute: \n",attribute)
print("Tree Keys\n", tree[attribute].keys())

def classify(instance, tree, default=None):
  attribute=next(iter(tree))
  print("Key:",tree.keys())
  print("Attribute",attribute)
  if instance[attribute] in tree[attribute].keys():
    result=tree[attribute][instance[attribute]]
    print("Instance Attribute",instance[attribute],
"TreeKeys:",tree[attribute].keys())
    if isinstance(result,dict):
      return classify(instance,result)
    else:
      return result
  else:
    return default
tree1={'Outlook':['Rainy','Sunny'],'Temperature':['Mild','Hot'],'Humidity':['Normal'
,'High'],'Windy':['Weak','Strong']}
df2=pd.DataFrame(tree1)
df2['Predicted']=df2.apply(classify,axis=1, args=(tree,'No'))
```

WEEK-3

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point, xmat, k):
   m,n = np.shape(xmat)
   weights = np.mat(np.eye((m)))
   for j in range(m):
     diff = point - X[j]
     print("Point",point)
     print("Diff",diff)
     weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
     print("Weights",weights)
   return weights
def localWeight(point, xmat, ymat, k):
  wei = kernel(point,xmat,k)
  W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
  print("W",W)
  return W
def localWeightRegression(xmat, ymat, k):
  m,n = np.shape(xmat)
  ypred = np.zeros(m)
  for i in range(m):
     ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
  return ypred
# load data points
data = pd.read_csv('C:\MLDS CSV FILES\hotel-bill1.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)
#preparing and add 1 in bill
mbill = np.mat(bill)
print("MBILL",mbill)
mtip = np.mat(tip)
print("Mtip",mtip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
print("X",X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')  next-line plt.show();
```

WEEK-4

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris =datasets.load_iris()
X=pd.DataFrame(iris.data)
print(X.shape)
X.columns=['Sepal_Length','Sepal_Width', 'Petal_length', 'Petal_Width']
y=pd.DataFrame(iris.target)
y.columns=['target']
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
plt.subplot(1,2,1)
plt.scatter(X.Sepal_Length,X.Sepal_Width,c=colormap[y.target],s=40)
plt.title('Sepal')
plt.subplot(1,2,2)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y.target],s=40)
plt.title('Petal')
plt.show()
model=KMeans(n_clusters=3)
model.fit(X)
print(model.labels_)
plt.subplot(1,2,1)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y.target],s=40)
plt.title('Real Classification')
plt.subplot(1,2,2)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[model.labels_],s=40)
plt.title( 'KMEANS Classfication')
plt.show()
print(sm.accuracy_score(y,model.labels_))
print(sm.confusion_matrix(y,model.labels_))

from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
clf.fit(X,y)
y_cluster_gmm=clf.predict(X)
print(y_cluster_gmm)
plt.subplot(1,2,1)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y.target],s=40)
plt.title('Real Classification')
plt.subplot(1,2,2)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title("Naive Bayesian Classification")
plt.show()
print(sm.accuracy_score(y,y_cluster_gmm))
print(sm.confusion_matrix(y,y_cluster_gmm))
#print(confusion_matrix)
```

WEEK-5

```
import sklearn
import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
#iris = pd.read_csv("iris.csv")
print(iris.keys())
df=pd.DataFrame(iris['data'])
print(df)
print(iris['target_names'])
print(iris['feature_names'])
print(iris['target'])
print("Feature Names")
print(iris.feature_names)
print("Target Names")
print(iris.target_names)
print("DataFrame with header Fields")
df=pd.DataFrame(iris.data,columns=iris.feature_names)
print(df.head())
print("shape and size of the dataset")
print(df.shape)
print("Index of the each colors with target")
df['target']=iris.target
print(df.head())
print(df[df.target==0].head())
print(df[df.target==1].head())
print(df[df.target==2].head())
print("Flower names with target of eacg features")
df['flower_name']=df.target.apply(lambda x: iris.target_names[x])
print(df.head())
print("instances with different indexes")
df0=df[:49]
df1=df[50:99]
df2=df[100:]
import matplotlib.pyplot as plt
print("sepal length and sepal width of setosa,versicolor, and virginica")
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],color="blue",marker='_')
plt.scatter(df2['sepal length (cm)'],df2['sepal width (cm)'],color="orange",marker='.')
plt.show()
print("petallength and petal width of setosa,versicolor, and virginica")
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color="blue",marker='_')
plt.scatter(df2['petal length (cm)'],df2['petal width (cm)'],color="orange",marker='.')
plt.show()
X=df
```

```
y=iris['target']
X=iris.data
y=iris.target
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=42)
print("Training data and Test data split")
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(X_test))
print(df)
#LOGISTICS REGRESSION
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X, y)
logreg.predict(X)
y_pred = logreg.predict(X)
len(y_pred)
print("Create KNN")
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
knn = KNeighborsClassifier(n_neighbors=3)
print("KNN FIT",knn.fit(X, y))
y_pred = knn.predict(X)
print(metrics.accuracy_score(y, y_pred))
#from sklearn import metrics  nex-#knn=KNeighborsClassifier(n_neighbors=3)
print("knn score",knn.score(X_test,y_test))
print("Hi")    next line-#Confusion Matrix
print("Confusion Matrix")
from sklearn.metrics import confusion_matrix
y_pred=knn.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
print(cm)
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True)
plt.xlabel=('Predicted')
plt.ylabel=('Truth')
plt.show()    next line- # Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))  next line- #Accuracy Score
from sklearn.metrics import accuracy_score
print("Correct prediction", accuracy_score(y_test,y_pred))
y_pred=knn.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
print(cm)
print("Wrong prediction", (1-accuracy_score(y_test,y_pred)))
y_testtrain=knn.predict(X_train)
cm1=confusion_matrix(y_train,y_testtrain)   next line- print(cm1)
```

WEEK-6

```
from inspect import BoundArguments
import numpy as np
X=np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
x=X/np.amax(X,axis=0)
y=y/100
def sigmoid(x):
  return 1/(1+np.exp(-x))
def derivatives_sigmoid(x):
  return x*(1-x)
epoch=1000
lr=0.01
inputlayer_neurons =2
hiddenlayer_neurons=3
output_neurons=1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
  hinp1=np.dot(X,wh)
  hinp=hinp1+bh
  hlayer_act=sigmoid(hinp)
  outinp1=np.dot(hlayer_act,wout)
  outinp=outinp1+bout
  output=sigmoid(outinp)
  EO=y-output
  outgrad=derivatives_sigmoid(output)
  d_output=EO*outgrad
  EH=d_output.dot(wout.T)
  hiddengrad=derivatives_sigmoid(hlayer_act)
  d_hiddenlayer=EH*hiddengrad
  wout+=hlayer_act.T.dot(d_output)*lr
  wh+=x.T.dot(d_hiddenlayer)*lr
print("Input: \n"+str(X))
print("Actual output: \n"+str(y))
print("Predicted output: \n",output)
```

WEEK 7

```
from textblob import TextBlob
text1=TextBlob('he is a good boy')
text2=TextBlob('he is working in MNC company')
print(text1.sentiment)
print(text2.sentiment)
import tweepy
from tweepy import OAuthHandler
from textblob import TextBlob
consumer_key = 'bgTDpmECgUtX6PfIcDTF8nWjr'
consumer_secret = 'PksMohOWT0et39DM5zCjYZxM0BnNA2odyv26R0pJm7dCZz
W1PB'
access_token = '173415763-FERWxppHuynJpIElmG7CxMSO0yYXeJ1I5Jp9XA2s'
access_token_secret = 'oJObpDU6chJEmRGCrnbJWZ4PRYDvT0w5OdQuaj8TLFf20'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
public_tweets=api.search('news')
for tweet in public_tweets:
  print(tweet.text)
  analysis=TextBlob(tweet.text)
  print(analysis.sentiment)
  if analysis.sentiment[0] > 0:
    print('positive')
  elif analysis.sentiment[0]< 0:
    print('negative')
  else:
    print('neutral')
```

WEEK-8

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Query - 1
train=pd.read_csv("train.csv")
print(train.head(10))
test=pd.read_csv("test.csv")
print(test.head(10))
#Query - 2
print(train.info())
#Query - 3
train['Gender'].value_counts(normalize=True).plot.bar(title='Gender')
plt.show()
#Query - 4
train.drop(columns=["Loan_ID"], inplace=True)
print(train.info())
#Query - 5
print(train.columns)
#Query - 6
print(train.isnull().sum())
#Query - 7
from sklearn.preprocessing import LabelEncoder
train.Loan_Status = train.Loan_Status.replace({"Y": 1, "N" : 0})
train.Gender = train.Gender.replace({"Male": 1, "Female" : 0})
test.Gender = test.Gender.replace({"Male": 1, "Female" : 0})
le = LabelEncoder()
train["Education"] = le.fit_transform(train["Education"])
test["Education"] = le.fit_transform(test["Education"])
print(train.head())
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
logistic_model = LogisticRegression()
train_features = ['ApplicatIncome', 'Education', 'Gender']
x_train = train[train_features].values
y_train = train['Loan_Status'].values
x_test = test[train_features].values
logistic_model.fit(x_train, y_train)
predicted = logistic_model.predict(x_test)
print('Coefficient of model :', logistic_model.coef_)
print('Intercept of model',logistic_model.intercept_)
score = logistic_model.score(x_train, y_train)
print('accuracy_score overall :', score)
print('accuracy_score percent :', round(score*100,2))
predict_test = logistic_model.predict(x_test)
```

```
print('Target on test data',predict_test)
```

x=len(predict_test) **next line- print(x)**

WEEK 9

```
import numpy as np
import pandas as pd
```

#Query - 1

```
data= pd.read_csv('water_dataX.csv')
print(data.shape)
```

#Query -2

```
data['temp']=pd.to_numeric(data['temp'],errors='coerce')
data.replace('NAN',0)
print(data)
print(data.info())
```

#Query -3

```
data['ntemp'] = data.temp.apply(lambda x: (100 if (80 >= x >= 60)
                       else (80 if (100 >= x >= 80) or (40 >= x >= 20)
                           else (60 if (60 >= x >= 40) or (20 >= x >= 0)
                              else (40 if (40 >= x >= 20) or (60 >= x >= 40)
                                 else 0)))))
data['nDO'] = data['D-O'].apply(lambda x: (100 if (80 >= x >= 60)
                       else (80 if (100 >= x >= 80) or (40 >= x >= 20)
                           else (60 if (60 >= x >= 40) or (20 >= x >= 0)
                              else (40 if (40 >= x >= 20) or (80 >= x >= 60)
                                 else 0)))))

data['nBDO'] = data['B-O-D'].apply(lambda x: (100 if (80 >= x >= 60)
                        else (80 if (100 >= x >= 80) or (40 >= x >= 20)
                            else (60 if (60 >= x >= 40) or (20 >= x >= 0)
                               else (40 if (40 >= x >= 20) or (80 >= x >= 60)
                                  else 0)))))

data['nPH'] = data['PH'].apply(lambda x: (100 if (8.5 >= x >= 7)
                       else (80 if (8.6 >= x >= 8.5) or (6.9 >= x >= 6.8)
                          else (60 if (8.8 >= x >= 8.6) or (6.8 >= x >= 6.7)
                             else (40 if (9 >= x >= 8.8) or (6.7 >= x >= 6.5)
                                 else 0)))))

data['nnn'] = data['Nitrate-Nitrogen'].apply(lambda x: (100 if (80 >= x >= 60)
                                      else (80 if (100 >= x >= 80) or (40 >= x >= 20)
                                         else (60 if (60 >= x >= 40) or (20 >= x >= 0)

else (40 if (40 >= x >= 20) or (80 >= x >= 60)
                                                         else 0)))))
data['wph']=data.nPH * 0.52
data['wDO']=data.nDO * 0.48
data['wBDO']=data.nBDO * 0.35
data['wtemp']=data.ntemp* 0.25
```

```
data['wnnn']=data.nnn * 0.028
data['wqi']=data.wph+data.wDO+data.wBDO+data.wtemp+data.wnnn
print(data['wqi'])
#Query - 4
ag=data.groupby('Year')['wqi'].mean()
type(ag)
print(ag)
#Query - 5
#scatter plot of data points
import matplotlib.pyplot as plt
y = data['wqi']
x=data['Year']
plt.scatter(x,y)
plt.show()
#Query -6
from sklearn import linear_model
from sklearn.model_selection import train_test_split
cols =['Year']
y = data['wqi']
x=data[cols]
reg=linear_model.LinearRegression()
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=4)
print(x_train)
print(y_train)
reg.fit(x_train,y_train)
a=reg.predict(x_test)
print(a)
print(y_test)
from sklearn.metrics import mean_squared_error
print('mse:%.2f'%mean_squared_error(y_test,a))
```

WEEK-10

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

#Query - 1

```
data=pd.read_csv("IPL1.csv")
print(data.head())
```

#Query - 2

```
data = {'Venue': ['Hyderabad', 'Pune', 'Vishakapatham'],
'Matches': [7, 3, 10]}
df = pd.DataFrame(data)
print(df)
```

#Query - 3

```
from sklearn.model_selection import train_test_split
df = pd.read_csv('IPL1.csv')
train_df, test_df = train_test_split(df, test_size=0.3, random_state=42)
print('Training set has {} rows'.format(len(train_df)))
print('Testing set has {} rows'.format(len(test_df)))
print(train_df)
train_df.head()
train_df.drop(['Match-id'],axis=1,inplace=True)
```

#Query - 5

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('IPL1.csv')
teams = df['Winning Team'].value_counts()
teams.plot(kind='bar')
plt.title('Number of Matches Won by IPL Teams')
plt.xlabel('Teams')
plt.ylabel('Number of Matches Won')
plt.show()
```

#Query - 4,6

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, confusion_matrix
features = ['Venue', 'Team1', 'Team2', 'Toss Status']
target = 'Winning Team'
X_train = pd.get_dummies(train_df[features])
y_train = train_df[target]
X_test = pd.get_dummies(test_df[features])
y_test = test_df[target]
X_train, X_test = X_train.align(X_test, join='outer', axis=1, fill_value=0)
clf = SVC(kernel='linear', C=1, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(confusion_matrix(y_test, y_pred))
print('Accuracy:', accuracy)
```