In [ ]:
```
# ASSIGNMENT N0-1 (UBER)
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import calendar
import datetime
df = pd.read_csv("uber.csv")
df.drop(['Unnamed:0','key','pickup_datetime'],axis=1,inplace=True)
df.isnull().sum()
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)
df
import scipy.stats as stats
Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
IQR=df.apply(stats.iqr)

df1=df[~((df < (Q1-1.5*IQR)) | (df > (Q3+1.5*IQR))).any(axis=1)]
df1
df.columns
df.isnull().sum()
df.dtypes
corr = df.corr()
corr
x=df[['pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude']]
y=df['fare_amount']
from sklearn.model_selection import train_test_split
(xtrain,xtest,ytrain,ytest)=train_test_split(x,y,test_size=0.2,random_state=42)
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(xtrain,ytrain)
pred=lr.predict(xtest)
pred
#ytest
from sklearn.metrics import r2_score,mean_squared_error
r2score=r2_score(ytest,pred)
r2score
#rmse=np.sqrt(mse)
#rmse
mse=mean_squared_error(ytest,pred)
mse
rmse=np.sqrt(mse)
rmse
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(n_estimators=100)
rf.fit(xtrain,ytrain)
rfpred=rf.predict(xtest)
r2score=r2_score(ytest,rfpred)
r2score
mse1=mean_squared_error(ytest,rfpred)
mse
rmse1=np.sqrt(mse1)
rmse
```

In [ ]:
```
#ASSIGNMENT N0-2 (email)
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
df=pd.read_csv('emails.csv')
df.head()
df.columns
df.isnull().sum()
df.dropna(inplace = True)
df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
from sklearn.preprocessing import scale
X = scale(X)
df
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Prediction",y_pred)
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
# cost C = 1
model = SVC(C = 1)
# fit
model.fit(X_train, y_train)
# predict
y_pred = model.predict(X_test)
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

In [ ]:

```
#ASSIGNMENT NO-3 (churn modelling)
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
df = pd.read_csv("Churn_Modelling.csv")
df
df.head()
df.shape
df.describe()
df.isnull()
df.isnull().sum()
df.info()
df.dtypes
df.columns
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary columns
df.head()
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary']]
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
df = pd.concat([df,gender,states], axis = 1)
df.head()
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male','Germany','Sp
y = df['Exited']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
X_test
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors initially.
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
classifier = Sequential()
#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializer = "uniform"))
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform")) #Adding seco
nd hidden layers
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform")) #Final neur
on will be having siigmoid function
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy']) #To com
pile the Artificial Neural Network. Ussed Binary crossentropy as we just have only two output
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training dataset
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
cm = confusion_matrix(y_test,y_pred)
cm
accuracy = accuracy_score(y_test,y_pred)
accuracy
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print(classification_report(y_test,y_pred))
```

In [ ]:

```
#ASSIGNMENT NO-4 (GRADIENT DESCENT)
```

```
#Step 1 : Initialize parameters
cur_x = 3 # The algorithm starts at x=3
rate = 0.01 # Learning rate
precision = 0.000001 #This tells us when to stop the algorithm
previous_step_size = 1 #
max_iters = 10000 # maximum number of iterations
iters = 0 #iteration counter
df = lambda x: 2*(x+5) #Gradient of our function

#Step 2 : Run a loop to perform gradient descent :i. Stop loop when difference between x values from 2 consecutive iterations is l
while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x #Store current x value in prev_x
    cur_x = cur_x - rate * df(prev_x) #Grad descent
    previous_step_size = abs(cur_x - prev_x) #Change in x
    iters = iters+1 #iteration count
    print("Iteration",iters,"\nX value is",cur_x) #Print iterations
    print("The local minimum occurs at", cur_x)
```

In [ ]:

```
#ASSIGNMENT N0-5
#Importing required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries

#Loading the dataset
df = pd.read_csv('diabetes.csv')
df
#Print the first 5 rows of the dataframe.
df.head()

#Confusion Matrix
#import confusion_matrix
from sklearn.metrics import confusion_matrix
#Let us get the predictions using the classifier we had fit above
y_pred = knn.predict(X_test)
confusion_matrix(y_test,y_pred)
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)

y_pred = knn.predict(X_test)
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

#2.Classification Report
#Precision = TP/TP+FP
#Recall = TP/TP+FN
#F1 Score = 2(Recall Precision) / (Recall + Precision)
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
precision recall f1 -score support
0 0.80 0.85 0.83 167
1 0.68 0.61 0.64 89
micro avg 0.77 0.77 0.77 256
macro avg 0.74 0.73 0.73 256
weighted avg 0.76 0.77 0.76 256

#Accuracy:
from sklearn.metrics import roc_curve
y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=11) ROC curve')
plt.show()

#Area under ROC curve
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```