```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read_csv("uber.csv")

df.head()

df.info()

df.shape

df.isnull().sum()

df.dropna(inplace = True)

df.isnull().sum()

df.drop(labels='Unnamed: 0',axis=1,inplace=True)

df.drop(labels='key',axis=1,inplace=True)

df.head()

df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"])

df.dtypes

df.describe()

import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

sns.distplot(df['fare_amount'])

sns.distplot(df['pickup_latitude'])

sns.distplot(df['pickup_longitude'])

sns.distplot(df['dropoff_longitude'])

sns.distplot(df['dropoff_latitude'])

def find_outliers_IQR(df):

  q1 = df.quantile(0.25)

  q3 = df.quantile(0.75)

  IQR = q3-q1

  outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]
```

```
  return outliers

outliers = find_outliers_IQR(df["fare_amount"])

print("number of outliers: "+ str(len(outliers)))

print("max outlier value: "+ str(outliers.max()))

print("min outlier value: "+ str(outliers.min()))

outliers

outliers = find_outliers_IQR(df[["passenger_count","fare_amount"]])

outliers

corrMatrix = df.corr()

sns.heatmap(corrMatrix, annot=True)

plt.show()

import calendar

df['day']=df['pickup_datetime'].apply(lambda x:x.day)

df['hour']=df['pickup_datetime'].apply(lambda x:x.hour)

df['month']=df['pickup_datetime'].apply(lambda x:x.month)

df['year']=df['pickup_datetime'].apply(lambda x:x.year)

df['weekday']=df['pickup_datetime'].apply(lambda x: calendar.day_name[x.weekday()])

df.drop(['pickup_datetime'],axis =1 , inplace = True)

df.weekday =
df.weekday.map({'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturday':6})

df.head()

df.info()

from sklearn.model_selection import train_test_split

x=df.drop("fare_amount", axis=1)

x

y=df["fare_amount"]

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=101)

x_train.head()

x_test.head()
```

```
y_train.head()

y_test.head()

from sklearn.linear_model import LinearRegression

lrmodel=LinearRegression()

lrmodel.fit(x_train, y_train)

predictedvalues = lrmodel.predict(x_test)

from sklearn.metrics import mean_squared_error

lrmodelrmse = np.sqrt(mean_squared_error(predictedvalues, y_test))

print("RMSE value for Linear regression is", lrmodelrmse)

from sklearn.ensemble import RandomForestRegressor

rfrmodel = RandomForestRegressor(n_estimators=100, random_state=101)

rfrmodel.fit(x_train,y_train)

rfrmodel_pred= rfrmodel.predict(x_test)

rfrmodel.fit(x_train,y_train)

rfrmodel_pred= rfrmodel.predict(x_test)

rfrmodel.fit(x_train,y_train)

rfrmodel_pred= rfrmodel.predict(x_test)

rfrmodel_pred.shape

test = pd.read_csv("https://raw.githubusercontent.com/piyushpandey758/Uber-Fare-
Prediction/master/testt.csv")

test.head()

test.drop(test[['Unnamed: 0.1.1','Unnamed: 0','Unnamed: 0.1','key']],axis=1,inplace=True)

test.isnull().sum()

test["pickup_datetime"] = pd.to_datetime(test["pickup_datetime"])

test['day']=test['pickup_datetime'].apply(lambda x:x.day)

test['hour']=test['pickup_datetime'].apply(lambda x:x.hour)

test['month']=test['pickup_datetime'].apply(lambda x:x.month)

test['year']=test['pickup_datetime'].apply(lambda x:x.year)

test['weekday']=test['pickup_datetime'].apply(lambda x: calendar.day_name[x.weekday()])
```

```
test.weekday =
test.weekday.map({'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturday':6})

test.drop(['pickup_datetime'], axis = 1, inplace = True)

test.head(5)

rfrmodel_pred= rfrmodel.predict(test)

df_pred = pd.DataFrame(rfrmodel_pred)

df_pred

df_pred.to_csv('pred.csv')
```

2

```
import matplotlib as plot

import numpy as np

import sympy as sym       #Lib for Symbolic Math

from matplotlib import pyplot

def objective(x):

  return (x+3)**2

def derivative(x):

  return 2*(x + 3)

def gradient_descent(alpha, start, max_iter):

  x_list = list()

  x= start;

  x_list.append(x)

  for i in range(max_iter):

    gradient = derivative(x);

    x = x - (alpha*gradient);

    x_list.append(x);

  return x_list

alpha = 0.1       #Step_size

start = 2         #Starting point
```

```
max_iter = 30     #Limit on iterations

x = sym.symbols('x')

expr = (x+3)**2;   #target function

x_cordinate = np.linspace(-15,15,100)

pyplot.plot(x_cordinate,objective(x_cordinate))

pyplot.plot(2,objective(2),'ro')

x_cordinate = np.linspace(-15,15,100)

pyplot.plot(x_cordinate,objective(x_cordinate))

pyplot.plot(2,objective(2),'ro')
```

3 KNN

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, f1_score, recall_score, precision_score,accuracy_score

df=pd.read_csv("C:\\Users\\HP\\Downloads\\diabetes.csv")

df.head()

df.shape

df.describe()

df.columns

df.isnull().sum()

X=df.drop('Outcome',axis=1)

y=df['Outcome']

from sklearn.preprocessing import scale

X=scale(X)
```

```
#split into train test

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)

knn=KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train,y_train)

y_pred=knn.predict(X_test)

print("Confusion matrix")

cs=confusion_matrix(y_test,y_pred)

print(cs)

accuracy_score(y_test,y_pred)

precision_score(y_test,y_pred)

recall_score(y_test,y_pred)

error_rate=1-accuracy_score(y_test,y_pred)

error_rate

from sklearn import metrics

print("classification report ",metrics.classification_report(y_test,y_pred))
```

4 K MEAN

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read_csv("C:\\Users\\HP\\Downloads\\sales_data_sample.csv", encoding='Latin-1')

data.head()

data.shape

data.isnull().sum()

data.drop(["ORDERNUMBER", "PRICEEACH", "ORDERDATE", "PHONE", "ADDRESSLINE1", "ADDRESSLINE2",
"CITY", "STATE", "TERRITORY", "POSTALCODE", "CONTACTLASTNAME", "CONTACTFIRSTNAME"], axis = 1,
inplace=True)

data.head()

data.isnull().sum()
```

```
data.describe()

sns.countplot(data = data , x = 'STATUS')

import seaborn as sns

data['PRODUCTLINE'].unique()

data.drop_duplicates(inplace=True)

data.info()

list_cat = data.select_dtypes(include=['object']).columns.tolist()

list_cat

for i in list_cat:

  sns.countplot(data = data ,x = i)

  plt.xticks(rotation = 90)

  plt.show()

from sklearn import preprocessing

le = preprocessing.LabelEncoder()


# Encode labels in column 'species'.

for i in list_cat:

  data[i]= le.fit_transform(data[i])

data.info()

data['SALES'] = data['SALES'].astype(int)

data.info()

data.describe()

X = data[['SALES','PRODUCTCODE']]

data.columns

from sklearn.cluster import KMeans

km=KMeans(1)

from yellowbrick.cluster import KElbowVisualizer

model = KMeans()

visualizer = KElbowVisualizer(model, k=(1,12)).fit(X)
```

```
visualizer.show()

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4, init='k-means++', random_state=0).fit(X)

kmeans.labels_

kmeans.inertia_

kmeans.n_iter_

kmeans.cluster_centers_

from collections import Counter

Counter(kmeans.labels_)

sns.scatterplot(data=X, x="SALES", y="PRODUCTCODE", hue=kmeans.labels_)

plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],

        marker="X", c="r", s=80, label="centroids")

plt.legend()

plt.show()
```