predictive analytics

```
import pandas as pd
```

```
import numpy as num
```

```
import matplotlib.pyplot as mp
```

```
import seaborn as sn
```

```
df=pd.read_csv("https://github.com/YBI-foundation/Dataset/raw/main/MPG.csv")
```

```
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	usa
1	15.0	8	350.0	165.0	3693	11.5	70	usa

```
df.tail()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origir
393	27.0	4	140.0	86.0	2790	15.6	82	usa
394	44.0	4	97.0	52.0	2130	24.6	82	europe
395	32.0	4	135.0	84.0	2295	11.6	82	usa

```
df.nunique()
```

```
mpg             129
cylinders         5
displacement     82
horsepower       93
weight          351
```

```
acceleration     95
model_year       13
origin            3
name            305
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64  
 7   origin        398 non-null    object 
 8   name          398 non-null    object 
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

```
df.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.(
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.(
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.6
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.(
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.(
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.(
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.(
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.(

```
df.corr()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	mo
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	

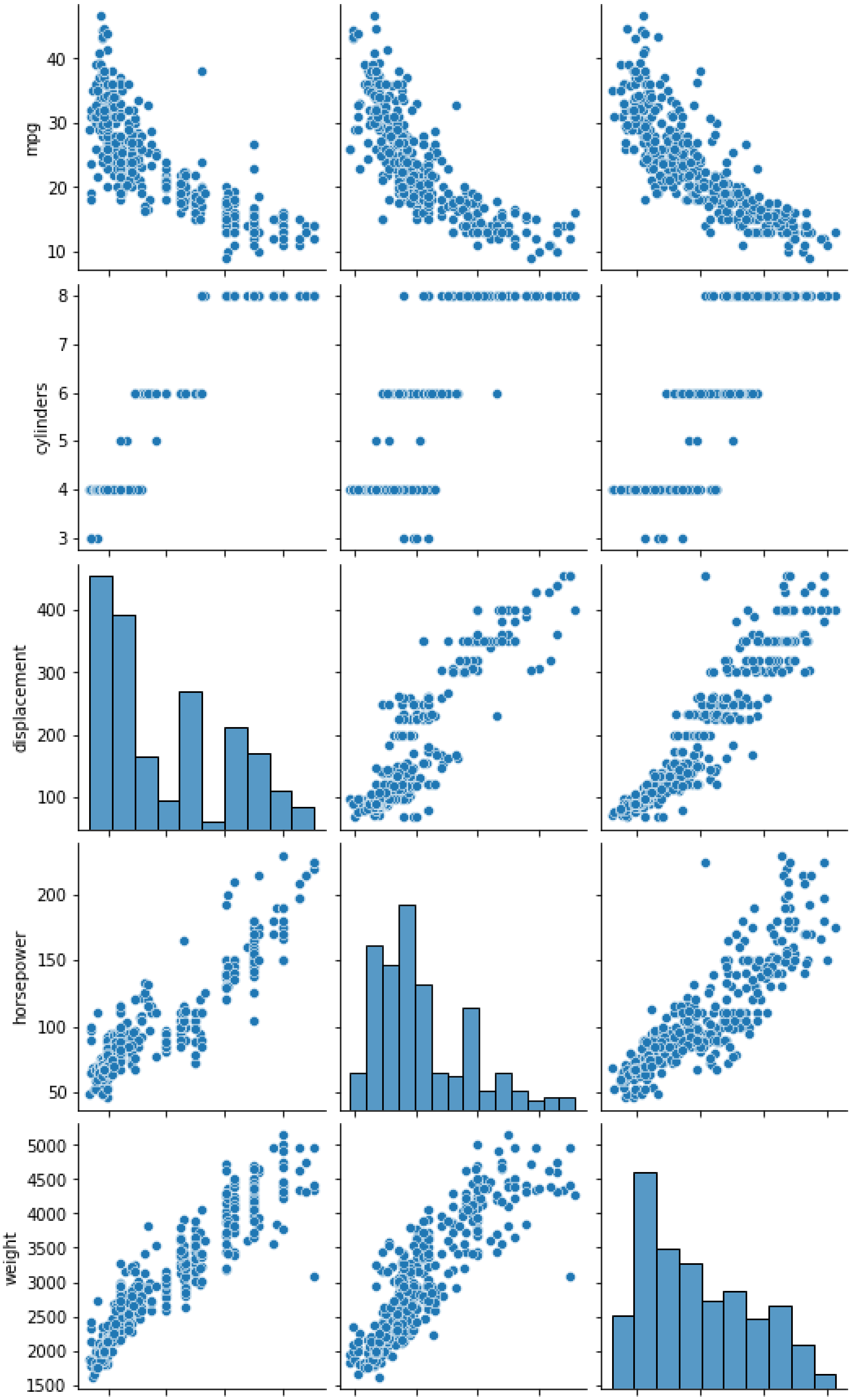
```
df=df.dropna()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        392 non-null    int64
 5   acceleration  392 non-null    float64
 6   model_year    392 non-null    int64
 7   origin        392 non-null    object
 8   name          392 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 30.6+ KB
```

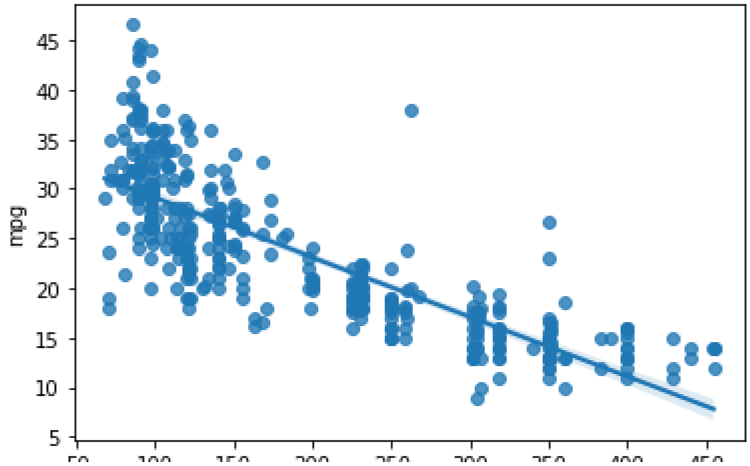
```
sn.pairplot(df,x_vars=['displacement','horsepower','weight'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f0c93f09ed0>
```




```
sn.regplot(x='displacement',y='mpg',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c8eef9e10>
```




```
df.columns
```

```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'name'],
      dtype='object')
```

```
y=df['mpg']
```

```
y.shape
```

```
(392,)
```

```
x=df[['displacement','weight','acceleration']]
```

```
x.shape
```

```
(392, 3)
```

```
from sklearn.preprocessing import StandardScaler
```

```
ss=StandardScaler()
```

```
x=ss.fit_transform(x)
```

```
x
```

```
array([[ 1.07728956,  0.62054034, -1.285258  ],
       [ 1.48873169,  0.84333403, -1.46672362],
       [ 1.1825422 ,  0.54038176, -1.64818924],
       ...,
       [-0.56847897, -0.80463202, -1.4304305 ],
       [-0.7120053 , -0.41562716,  1.11008813],
       [-0.72157372, -0.30364091,  1.40043312]])
```

```
pd.DataFrame(x).describe()
```

	0	1	2
count	3.920000e+02	3.920000e+02	3.920000e+02
mean	-2.537653e-16	5.607759e-17	6.117555e-16
std	1.001278e+00	1.001278e+00	1.001278e+00
min	-1.209563e+00	-1.608575e+00	-2.736983e+00
25%	-8.555316e-01	-8.868535e-01	-6.410551e-01
50%	-4.153842e-01	-2.052109e-01	-1.499869e-02
75%	7.782764e-01	7.510927e-01	5.384714e-01
max	2.493416e+00	2.549061e+00	3.360262e+00

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7,random_state=2535)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((274, 3), (118, 3), (274,), (118,))
```

```
from sklearn.linear_model import LinearRegression
```

```
lr=LinearRegression()
```

```
lr.fit(x_train,y_train)
```

```
LinearRegression()
```

```
lr.intercept_
```

```
23.481757374918818
```

```
lr.coef_
```

```
array([-0.3969492 , -5.81652961,  0.84382013])
```

```
y_pred=lr.predict(x_test)
```

```
y_pred
```

```
array([27.0950622 , 24.86846697, 25.46608867, 20.18920635, 31.58350197,
       31.07258189, 25.8960753 , 20.05161656, 14.19571234, 28.80613919,
       25.41127903, 11.52775544, 28.85413497, 20.1402067 , 19.92816833,
       15.46828926, 15.18457187, 18.64192416, 28.15641361, 14.88968591,
       14.47829349, 20.56365413, 20.64130953, 23.37278745, 32.21165547,
       18.17166849, 13.7990552 , 11.86580018, 30.78406874, 32.56912928,
       24.00456213, 29.42837442, 25.558358  , 13.21581137, 21.1979567 ,
       26.89621049, 31.36274567, 28.55664204, 29.46346568, 30.81338038,
       26.16361613, 29.24905037, 23.45861862, 30.65157817, 33.26899217,
       11.93439594, 18.8961787 , 27.11860968, 13.9210924 , 21.54222525,
       11.94238978, 26.00051733, 27.12174109, 16.78845349, 12.60957108,
       27.76932967, 29.71535153, 27.71270789, 23.26653444, 18.82533404,
       14.21400151, 21.57102879, 21.06774658, 15.00213014, 31.22018153,
       23.85502572, 17.20993611, 26.27905746, 15.47922728, 25.31382302,
       24.44682056,  9.49115678, 18.29693791, 30.14639531, 10.58013426,
       23.24593446, 24.84436754, 29.40520557, 26.75682528,  8.54723467,
       32.78899625, 14.62408239, 15.45600422, 29.3864358 , 23.71451515,
       25.06107848, 21.01760441, 29.1670417 , 21.03359794, 29.34433562,
       30.45713615, 30.60236054, 19.30241702,  6.7896799 , 15.94287186,
       12.39166433, 17.10206798, 26.27292266, 24.1721167 , 12.22118426,
       15.69647721, 17.10525464, 31.64640977, 20.11024545, 26.62598022,
       34.05599176, 12.75776101, 30.64782582, 24.3743673 , 26.02748605,
       25.05729718, 28.32368724, 24.46624597, 26.67891839, 31.32791246,
       27.32951181, 29.34477957, 30.31335789])
```

```
from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,r2_score
```

```
mean_absolute_error(y_test,y_pred)
```

```
3.2445387480016987
```

```
mean_absolute_percentage_error(y_test,y_pred)
```

```
0.14972398521815397
```

```
r2_score(y_test,y_pred)
```

```
0.7088621831979525
```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly=PolynomialFeatures(degree=2,interaction_only=True,include_bias=False,order='c')
```

```
x_train2=poly.fit_transform(x_train)
```

```
x_test2=poly.fit_transform(x_test)
```

```
lr.fit(x_train2,y_train)
```

```
LinearRegression()
```

```
lr.intercept_
```

```
21.76889066826468
```

```
lr.coef_
```

```
array([-1.9108918 , -5.28015114,  1.01166591,  1.67884703, -0.80744566,
        0.80323871])
```

```
y_pred_poly=lr.predict(x_test2)
```

```
r2_score(y_test,y_pred_poly)
```

```
0.7490061639178867
```

```
from sklearn.datasets import load_digits
```

```
df=load_digits()
```

```
df.images.shape
```

```
(1797, 8, 8)
```

```
df.images[0].shape
```

```
(8, 8)
```

```
df.images[0]
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

```
n_samples=len(df.images)
data=df.images.reshape((n_samples,-1))
```

```
data[0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

```
data.shape
```

```
(1797, 64)
```

```
data[0].shape
```

```
(64,)
```

```
data.min()
```

```
0.0
```

```
data.max()
```

```
16.0
```

```
data[0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf=RandomForestClassifier()
```

```
from sklearn.metrics import confusion_matrix,classification_report
```

```
len(df.images)
```

```
1797
```

car price prediction

```
df1=pd.read_csv("https://github.com/YBI-foundation/Dataset/raw/main/Car%20Price.csv")
```

```
df1.head()
```

	Brand	Model	Year	Selling_Price	KM_Driven	Fuel	Seller_Type	Transmission
0	Maruti	Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual
1	Maruti	Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual
		Hyundai						

```
df1.tail()
```

	Brand	Model	Year	Selling_Price	KM_Driven	Fuel	Seller_Type	Transmissior
4335	Hyundai	Hyundai i20 Magna 1.4 CRDi (Diesel)	2014	409999	80000	Diesel	Individual	Manua
		Hyundai i20						

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Brand          4340 non-null   object
 1   Model          4340 non-null   object
 2   Year           4340 non-null   int64 
 3   Selling_Price  4340 non-null   int64 
 4   KM_Driven      4340 non-null   int64 
 5   Fuel           4340 non-null   object
 6   Seller_Type    4340 non-null   object
 7   Transmission   4340 non-null   object
 8   Owner          4340 non-null   object
dtypes: int64(3), object(6)
memory usage: 305.3+ KB
```

```
df1.describe()
```

	Year	Selling_Price	KM_Driven
count	4340.000000	4.340000e+03	4340.000000
mean	2013.090783	5.041273e+05	66215.777419
std	4.215344	5.785487e+05	46644.102194
min	1992.000000	2.000000e+04	1.000000
25%	2011.000000	2.087498e+05	35000.000000
50%	2014.000000	3.500000e+05	60000.000000
75%	2016.000000	6.000000e+05	90000.000000
	[illegible]	[illegible]	[illegible]

```
df1[['Brand']].value_counts()
```

```
Brand
Maruti           1280
Hyundai           821
Mahindra          365
Tata              361
Honda             252
Ford              238
Toyota            206
Chevrolet         188
Renault           146
Volkswagen        107
Skoda              68
Nissan             64
Audi               60
BMW                39
Fiat               37
Datsun             37
Mercedes-Benz      35
Mitsubishi          6
Jaguar              6
Land                5
Ambassador          4
Volvo               4
Jeep                3
OpelCorsa           2
MG                  2
Isuzu               1
Force               1
Daewoo              1
Kia                 1
dtype: int64
```

```
df1[['Fuel']].value_counts()
```

```
Fuel
Diesel      2153
Petrol      2123
```

```
CNG           40
LPG           23
Electric       1
dtype: int64
```

```
df1[['Model']].value_counts()
```

```
Model
Maruti Swift Dzire VDI                    69
Maruti Alto 800 LXI                       59
Maruti Alto LXi                           47
Hyundai EON Era Plus                      35
Maruti Alto LX                            35
                                          ..
Mahindra KUV 100 G80 K4 Plus               1
Mahindra KUV 100 mFALCON D75 K8            1
Mahindra KUV 100 mFALCON D75 K8 AW         1
Mahindra KUV 100 mFALCON G80 K2 Plus       1
Volvo XC60 D5 Inscription                  1
Length: 1491, dtype: int64
```

```
df1.columns
```

```
Index(['Brand', 'Model', 'Year', 'Selling_Price', 'KM_Driven', 'Fuel',
       'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')
```

```
df1.shape
```

```
(4340, 9)
```

```
df1.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)
```

```
y=df1['Selling_Price']
```

```
y.shape
```

```
(4340,)
```

```
x=df1[['Year','Transmission']]
```

```
x.shape
```

```
(4340, 2)
```

```
x
```

	Year	Transmission
0	2007	0
1	2007	0
2	2012	0
3	2017	0
4	2014	0
...	...	...
4335	2014	0
4336	2014	0
4337	2009	0
4338	2016	0
4339	2016	0

4340 rows × 2 columns

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.5,random_state=2539)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((2170, 2), (2170, 2), (2170,), (2170,))
```

```
from sklearn.linear_model import LinearRegression
```

```
lr=LinearRegression()
```

```
lr.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y
```

```
0        60000
1       135000
2       600000
3       250000
4       450000
         ...
4335    409999
4336    409999
```

```
4337    110000
4338    865000
4339    225000
Name: Selling_Price, Length: 4340, dtype: int64
```

```
y_pred=lr.predict(x_test)
```

```
y_pred.shape
```

```
(2170,)
```

```
y_pred
```

```
array([504237.05512911, 357265.74335442, 259284.86883798, ...,
       357265.74335442, 308275.3060962 , 455246.61787088])
```

```
from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,r2_score,mean_
```

```
mean_absolute_error(y_test,y_pred)
```

```
240430.41301709128
```

```
r2_score(y_test,y_pred)
```

```
0.36870936179375813
```

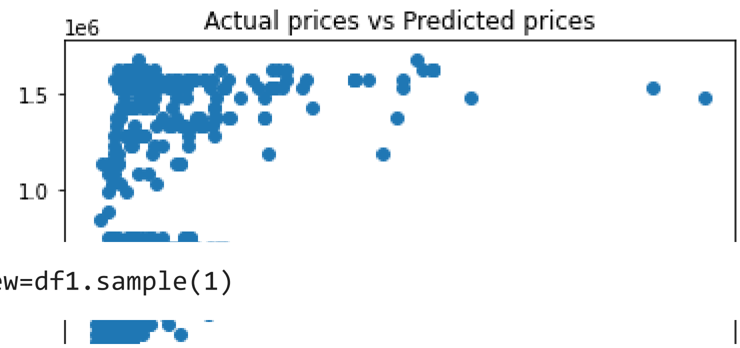
```
mean_squared_error(y_test,y_pred)
```

```
215527527203.4092
```

```
mean_absolute_percentage_error(y_test,y_pred)
```

```
0.6914047191678929
```

```
mp.scatter(y_test,y_pred)
mp.title('Actual prices vs Predicted prices')
mp.show()
```




```
df_new=df1.sample(1)
```

```
df_new
```

	Brand	Model	Year	Selling_Price	KM_Driven	Fuel	Seller_Type	Transmission	C
		Tata							

```
df_new.shape
```

```
(1, 9)
```

bike price prediction

```
df2=pd.read_csv("https://github.com/YBI-foundation/Dataset/raw/main/Bike%20Prices.csv")
```

```
df2.head()
```

	Brand	Model	Selling_Price	Year	Seller_Type	Owner	KM_Driven	Ex_Showroom_Pric
0	TVS	TVS XL 100	30000	2017	Individual	1st owner	8000	30490.
1	Bajaj	Bajaj ct 100	18000	2017	Individual	1st owner	35000	32000.
2	Yo	Yo Style	20000	2011	Individual	1st owner	10000	37675.

```
df2.tail()
```

	Brand	Model	Selling_Price	Year	Seller_Type	Owner	KM_Driven	Ex_Showroom_Pr

```
df2.describe()
```

	Selling_Price	Year	KM_Driven	Ex_Showroom_Price
count	1061.000000	1061.000000	1061.000000	6.260000e+02
mean	59638.151744	2013.867107	34359.833176	8.795871e+04
std	56304.291973	4.301191	51623.152702	7.749659e+04
min	5000.000000	1988.000000	350.000000	3.049000e+04
25%	28000.000000	2011.000000	13500.000000	5.485200e+04
50%	45000.000000	2015.000000	25000.000000	7.275250e+04
75%	70000.000000	2017.000000	43000.000000	8.703150e+04
max	760000.000000	2020.000000	880000.000000	1.278000e+06

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1061 entries, 0 to 1060
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Brand              1061 non-null   object 
 1   Model              1061 non-null   object 
 2   Selling_Price      1061 non-null   int64  
 3   Year               1061 non-null   int64  
 4   Seller_Type        1061 non-null   object 
 5   Owner              1061 non-null   object 
 6   KM_Driven          1061 non-null   int64  
 7   Ex_Showroom_Price  626 non-null    float64
dtypes: float64(1), int64(3), object(4)
memory usage: 66.4+ KB
```

```
df2=df2.dropna()
```

```
df2[['Brand']].value_counts()
```

```
Brand
Honda       170
Bajaj       143
Hero        108
Yamaha       94
Royal        40
TVS          23
Suzuki       18
KTM           6
```

```
Mahindra      6
Kawasaki      4
UM            3
Activa        3
Harley        2
Vespa         2
BMW           1
Hyosung       1
Benelli       1
Yo            1
dtype: int64
```

```
df2[['Owner']].value_counts()
```

```
Owner
1st owner    556
2nd owner     66
3rd owner      3
4th owner      1
dtype: int64
```

```
df2['Model'].value_counts()
```

```
Honda Activa [2000-2015]      23
Honda CB Hornet 160R          22
Bajaj Pulsar 180              20
Bajaj Discover 125            16
Yamaha FZ S V 2.0             16
                              ..
TVS Radeon                     1
Hero  Ignitor Disc             1
Bajaj V12                      1
Yamaha Cygnus Ray ZR           1
Harley-Davidson Street Bob     1
Name: Model, Length: 183, dtype: int64
```

```
df2.columns
```

```
Index(['Brand', 'Model', 'Selling_Price', 'Year', 'Seller_Type', 'Owner',
       'KM_Driven', 'Ex_Showroom_Price'],
      dtype='object')
```

```
df2.shape
```

```
(626, 8)
```

```
df2.replace({'Seller_Type':{'Individual':0,'Dealer':1}},inplace=True)
```

```
y=df2["Selling_Price"]
```

```
y
```

```
0       30000
1       18000
2       20000
3       25000
4       24999
        ...
621    330000
622    300000
623    425000
624    760000
625    750000
Name: Selling_Price, Length: 626, dtype: int64
```

```
x=df2[["Year","Seller_Type"]]
```

```
x.shape
```

```
(626, 2)
```

```
x
```

	Year	Seller_Type
0	2017	0
1	2017	0
2	2011	0
3	2010	0
4	2012	0
...	...	...
621	2014	0
622	2011	0
623	2017	0
624	2019	0
625	2013	0

626 rows × 2 columns

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.6,random_state=2532)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

    ((375, 2), (251, 2), (375,), (251,))

```
from sklearn.linear_model import LinearRegression
```

```
lr=LinearRegression()
```

```
lr.fit(x_train,y_train)
```

    LinearRegression()

```
y
```

```
0       30000
1       18000
2       20000
3       25000
4       24999
        ...
621    330000
622    300000
623    425000
624    760000
625    750000
Name: Selling_Price, Length: 626, dtype: int64
```

```
y_pred=lr.predict(x_test)
```

```
y.shape
```

    (626,)

```
y_pred
```

```
array([ 62064.35694737,  47153.40893919,  76975.30495554,  39697.9349351 ,
        69519.83095145,  76975.30495554,  69519.83095145,   9876.03891874,
        39697.9349351 ,  54608.88294328,  62064.35694737,  69519.83095145,
        84430.77895963,  62064.35694737,  76975.30495554,  84430.77895963,
        84430.77895963,  69519.83095145,  62064.35694737, -12490.38309353,
        76975.30495554,  62064.35694737,  69519.83095145,  62064.35694737,
        47153.40893919,  84430.77895963,  69519.83095145,  47153.40893919,
        62064.35694737,  69519.83095145,  69519.83095145,  76975.30495554,
        84430.77895963,  47153.40893919,  47153.40893919,  91886.25296372,
       -19945.85709762,  69519.83095145,  91886.25296372,  62064.35694737,
        62064.35694737,  39697.9349351 ,  47153.40893919,  76975.30495554,
        62064.35694737,  62064.35694737,  69519.83095145,  62064.35694737,
```

```
62064.35694737,  76975.30495554,  62064.35694737,  69519.83095145,
47153.40893919,  62064.35694737,  76975.30495554,  84430.77895963,
39697.9349351 ,  47153.40893919,  91886.25296372,  84430.77895963,
84430.77895963,  69519.83095145,  69519.83095145,  84430.77895963,
69519.83095145,  76975.30495554,  47153.40893919,  32242.46093101,
62064.35694737,  17331.51292283,  76975.30495554,  39697.9349351 ,
24786.98692692,  39697.9349351 ,  54608.88294328,  76975.30495554,
54608.88294328,  84430.77895963,  84430.77895963,  54608.88294328,
39697.9349351 ,  69519.83095145,  39697.9349351 ,  17331.51292283,
62064.35694737,  76975.30495554,  54608.88294328,  39697.9349351 ,
76975.30495554,  76975.30495554,  84430.77895963,  24786.98692692,
54608.88294328,  69519.83095145,  47153.40893919,  62064.35694737,
62064.35694737,  62064.35694737,  54608.88294328,  69519.83095145,
91886.25296372,  39697.9349351 ,  69519.83095145,  24786.98692692,
24786.98692692,  91886.25296372,  69519.83095145,  54608.88294328,
62064.35694737,  84430.77895963,  84430.77895963,  54608.88294328,
76975.30495554,  54608.88294328,  54608.88294328,   9876.03891874,
84430.77895963,  76975.30495554,  47153.40893919,  32242.46093101,
69519.83095145,  17331.51292283,  76975.30495554,  76975.30495554,
24786.98692692,  91886.25296372,  62064.35694737,  84430.77895963,
69519.83095145,  62064.35694737,  84430.77895963,  69519.83095145,
39697.9349351 ,  84430.77895963,  76975.30495554,  69519.83095145,
47153.40893919,  24786.98692692,  32242.46093101,  47153.40893919,
84430.77895963,  84430.77895963,  62064.35694737,  47153.40893919,
47153.40893919,  76975.30495554,  69519.83095145,  76975.30495554,
62064.35694737,   2420.56491465,  39697.9349351 ,  39697.9349351 ,
84430.77895963,  32242.46093101,  62064.35694737,  62064.35694737,
84430.77895963,  54608.88294328,  54608.88294328,  32242.46093101,
39697.9349351 ,  84430.77895963,  69519.83095145,  -5034.90908944,
32242.46093101,  84430.77895963,  47153.40893919,   2420.56491465,
76975.30495554,  62064.35694737,  47153.40893919,  54608.88294328,
69519.83095145,  32242.46093101,  62064.35694737,  76975.30495554,
91886.25296372,  91886.25296372,  54608.88294328,  54608.88294328,
84430.77895963,  76975.30495554,  32242.46093101,  54608.88294328,
91886.25296372,  62064.35694737,  62064.35694737,  47153.40893919,
62064.35694737,  62064.35694737,  76975.30495554,  54608.88294328,
54608.88294328,  84430.77895963,  47153.40893919,  15633.57798773,
91886.25296372,  69519.83095145,  24786.98692692,  62064.35694737,
24786.98692692,  62064.35694737,  24786.98692692,  91886.25296372,
76975.30495554,  39697.9349351 ,  76975.30495554,  84430.77895963,
69519.83095145,  84430.77895963,  69519.83095145,  69519.83095145,
76975.30495554,   2420.56491465,  17331.51292283,  24786.98692692,
84430.77895963,  62064.35694737,  39697.9349351 ,  39697.9349351 ,
91886.25296372,  54608.88294328,  62064.35694737,  54608.88294328,
69519.83095145,  69519.83095145,  84430.77895963,  54608.88294328,
54608.88294328,  69519.83095145,  76975.30495554,  54608.88294328,
```

```
y_pred.shape
```

```
(251,)
```

```
from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,r2_score,mean_
```

```
mean_absolute_error(y_test,y_pred)
```

```
28689.77223478409
```

```
r2_score(y_test,y_pred)
```

```
0.06731083195243104
```

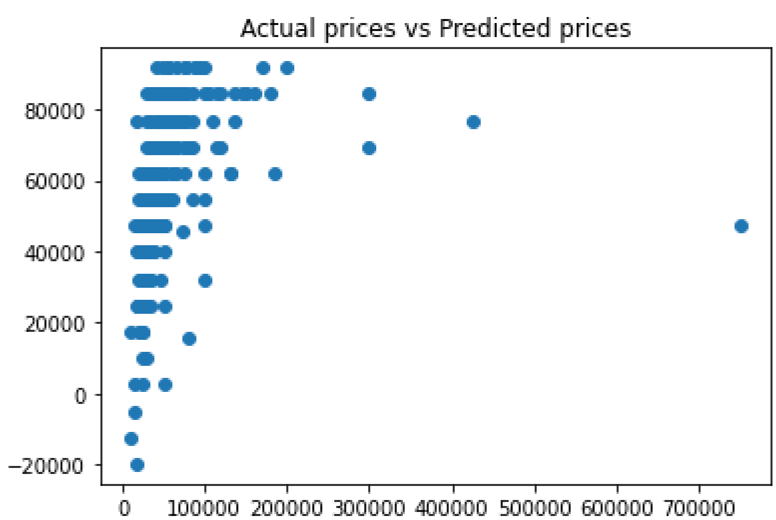
```
mean_squared_error(y_test,y_pred)
```

```
3701406312.046656
```

```
mean_absolute_percentage_error(y_test,y_pred)
```

```
0.5629069619670382
```

```
mp.scatter(y_test,y_pred)
mp.title('Actual prices vs Predicted prices')
mp.show()
```




```
df1_new=df2.sample(1)
```

```
df1_new.shape
```

```
(1, 8)
```

finamcial market news

```
df3=pd.read_csv(r'https://raw.githubusercontent.com/YBI-Foundation/Dataset/main/Financial%20M
```

```
df3.head()
```

	Date	Label	News 1	News 2	News 3	News 4	News 5	News 6	News
0	01-01-2010	0	McIlroy's men catch cold from Gudjonsson	Obituary: Brian Walsh	Workplace blues leave employers in the red	Classical review: Rattle	Dance review: Merce Cunningham	Genetic tests to be used in setting premiums	Op review: Bohè
1	02-01-2010	0	Warning from history points to crash	Investors flee to dollar haven	Banks and tobacco in favour	Review: Llama Farmers	War jitters lead to sell-off	Your not-so-secret history	Revie T North Sinfo
2	03-01-2010	0	Comment: Why Israel's peaceniks feel betrayed	Court deals blow to seizure of drug assets	An ideal target for spooks	World steps between two sides intent on war	What the region's papers say	Comment: Fear and rage in Palestine	Pove a resentm fu Palestin f
3	04-01-2010	1	£750,000-a-goal Weah aims parting shot	Newcastle pay for Fletcher years	Brown sent to the stands for Scotland qualifier	Tourists wary of breaking new ground	Canary Wharf climbs into the FTSE 100	Review: Bill Bailey	Revie Classi
4	05-01-2010	1	Leeds arrive in Turkey to the silence of the fans	One woman's vision offers loan lifeline	Working Lives: How world leaders worked	Working Lives: Tricks of the trade	Working Lives: six-hour days, long lunches and...	Pop review: We Love UK	Wo mu revie Mar Mo

5 rows × 27 columns

```
df3.tail()
```

	Date	Label	News 1	News 2	News 3	News 4	News 5	News 6
4096	20-03-2021	0	Barclays and RBS shares suspended from trading...	Pope says Church should ask forgiveness from g...	Poland 'shocked' by xenophobic abuse of Poles ...	There will be no second referendum, cabinet ag...	Scotland welcome to join EU, Merkel ally says	Sterling dips below Friday's 31-year low amid ...
4097	21-03-2021	1	2,500 Scientists To Australia: If You Want To ...	The personal details of 112,000 French police ...	S&P cuts United Kingdom sovereign credit r...	Huge helium deposit found in Africa	CEO of the South African state broadcaster qui...	Brexit cost investors $2 trillion, the worst o...
4098	22-03-2021	1	Explosion At Airport In Istanbul	Yemeni former president: Terrorism is the offs...	UK must accept freedom of movement to access E...	Devastated: scientists too late to captive bre...	British Labor Party leader Jeremy Corbyn loses...	A Muslim Shop in the UK Was Just Firebombed Wh...
4099	23-03-2021	1	Jamaica proposes marijuana dispensers for tour...	Stephen Hawking says pollution and 'stupidity'...	Boris Johnson says he will not run for Tory pa...	Six gay men in Ivory Coast were abused and for...	Switzerland denies citizenship to Muslim immig...	Palestinian terrorist stabs israeli teen girl ...
			A 117-	IMF chief	The			

```
df3.describe()
```

	Label
count	4101.000000
mean	0.528164
std	0.499267
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	1.000000

```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4101 entries, 0 to 4100
Data columns (total 27 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Date     4101 non-null   object
 1   Label    4101 non-null   int64 
 2   News 1   4101 non-null   object
 3   News 2   4101 non-null   object
 4   News 3   4101 non-null   object
 5   News 4   4101 non-null   object
 6   News 5   4101 non-null   object
 7   News 6   4101 non-null   object
 8   News 7   4101 non-null   object
 9   News 8   4101 non-null   object
 10  News 9   4101 non-null   object
 11  News 10  4101 non-null   object
 12  News 11  4101 non-null   object
 13  News 12  4101 non-null   object
 14  News 13  4101 non-null   object
 15  News 14  4101 non-null   object
 16  News 15  4101 non-null   object
 17  News 16  4101 non-null   object
 18  News 17  4101 non-null   object
 19  News 18  4101 non-null   object
 20  News 19  4101 non-null   object
 21  News 20  4101 non-null   object
 22  News 21  4101 non-null   object
 23  News 22  4101 non-null   object
 24  News 23  4100 non-null   object
 25  News 24  4098 non-null   object
 26  News 25  4098 non-null   object
dtypes: int64(1), object(26)
memory usage: 865.2+ KB
```

```
df3.shape
```

```
(4101, 27)
```

```
df3.columns
```

```
Index(['Date', 'Label', 'News 1', 'News 2', 'News 3', 'News 4', 'News 5',
       'News 6', 'News 7', 'News 8', 'News 9', 'News 10', 'News 11', 'News 12',
       'News 13', 'News 14', 'News 15', 'News 16', 'News 17', 'News 18',
       'News 19', 'News 20', 'News 21', 'News 22', 'News 23', 'News 24',
       'News 25'],
      dtype='object')
```

```
df3.index
```

```
RangeIndex(start=0, stop=4101, step=1)
```

```
len(df3.index)
```

```
4101
```

```
news=[]
for row in range(0,len(df3.index)):
  news.append(' '.join(str(x) for x in df3.iloc[row,2:27]))
```

```
type(news)
```

```
list
```

```
news[0]
```

```
'McIlroy's men catch cold from Gudjonsson Obituary: Brian Walsh Workplace blues leave e
mployers in the red Classical review: Rattle Dance review: Merce Cunningham Genetic tes
ts to be used in setting premiums Opera review: La Bohème Pop review: Britney Spears Th
eatre review: The Circle Wales face a fraught night Under-21  round-up Smith off to blo
t his copybook Finns taking the mickey Praise wasted as Brown studies injury options Ir
eland wary of minnows Finland 0 - 0 England Healy a marked man Happy birthday Harpers &
```

```
x=news
```

```
type(x)
```

```
list
```

```
from sklearn.feature_extraction.text import  CountVectorizer
```

```
cv=CountVectorizer(lowercase=True,ngram_range=(1,1))
```

```
x=cv.fit_transform(x)
```

```
x.shape
```

```
(4101, 48527)
```

```
y=df3['Label']
```

```
y.shape
```

```
(4101,)
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.4,stratify=y,random_state=252
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf=RandomForestClassifier(n_estimators=200)
```

```
rf.fit(x_train,y_train)
```

```
RandomForestClassifier(n_estimators=200)
```

```
y_pred=rf.predict(x_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[310, 851],
       [316, 984]])
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      0.27      0.35      1161
           1       0.54      0.76      0.63      1300

    accuracy                           0.53      2461
   macro avg       0.52      0.51      0.49      2461
weighted avg       0.52      0.53      0.50      2461
```

```
df4=pd.read_csv(r"https://raw.githubusercontent.com/YBI-Foundation/Dataset/main/Movies%20Reco
```

```
df4.head()
```

	Movie_ID	Movie_Title	Movie_Genre	Movie_Language	Movie_Budget	Movie_Popularity
0	1	Four Rooms	Crime Comedy	en	4000000	22.876230
1	2	Star Wars	Adventure Action Science Fiction	en	11000000	126.393695
2	3	Finding Nemo	Animation Family	en	94000000	85.688789
3	4	Forrest Gump	Comedy Drama Romance	en	55000000	138.133331
4	5	American	Drama	en	15000000	80.878605

```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4760 entries, 0 to 4759
Data columns (total 21 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Movie_ID                 4760 non-null   int64  
 1   Movie_Title              4760 non-null   object 
 2   Movie_Genre              4760 non-null   object 
 3   Movie_Language           4760 non-null   object 
 4   Movie_Budget             4760 non-null   int64  
 5   Movie_Popularity         4760 non-null   float64
 6   Movie_Release_Date       4760 non-null   object 
 7   Movie_Revenue            4760 non-null   int64  
 8   Movie_Runtime            4758 non-null   float64
 9   Movie_Vote               4760 non-null   float64
 10  Movie_Vote_Count         4760 non-null   int64  
 11  Movie_Homepage           1699 non-null   object 
 12  Movie_Keywords           4373 non-null   object 
 13  Movie_Overview           4757 non-null   object 
 14  Movie_Production_House   4760 non-null   object 
 15  Movie_Production_Country 4760 non-null   object 
```

```
 16  Movie_Spoken_Language     4760 non-null   object
 17  Movie_Tagline             3942 non-null   object
 18  Movie_Cast                4733 non-null   object
 19  Movie_Crew                4760 non-null   object
 20  Movie_Director            4738 non-null   object
dtypes: float64(3), int64(4), object(14)
memory usage: 781.1+ KB
```

```
df4.shape
```

```
(4760, 21)
```

```
df4.columns
```

```
Index(['Movie_ID', 'Movie_Title', 'Movie_Genre', 'Movie_Language',
       'Movie_Budget', 'Movie_Popularity', 'Movie_Release_Date',
       'Movie_Revenue', 'Movie_Runtime', 'Movie_Vote', 'Movie_Vote_Count',
       'Movie_Homepage', 'Movie_Keywords', 'Movie_Overview',
       'Movie_Production_House', 'Movie_Production_Country',
       'Movie_Spoken_Language', 'Movie_Tagline', 'Movie_Cast', 'Movie_Crew',
       'Movie_Director'],
      dtype='object')
```

```
df_features=df4[['Movie_ID', 'Movie_Title', 'Movie_Genre', 'Movie_Language']].fillna('')
```

```
df_features.shape
```

```
(4760, 4)
```

```
df_features
```

	Movie_ID	Movie_Title	Movie_Genre	Movie_Language
0	1	Four Rooms	Crime Comedy	en
2	3	Finding Nemo	Animation Family	en

```
x=df_features['Movie_Genre']+' '+df_features['Movie_Title']+' '+df_features['Movie_Language']
```

```
x
```

```
0                             Crime Comedy Four Rooms en
1        Adventure Action Science Fiction Star Wars en
2                          Animation Family Finding Nemo en
3                      Comedy Drama Romance Forrest Gump en
4                               Drama American Beauty en
                                   ...
4755                          Horror Midnight Cabaret en
4756            Comedy Family Drama Growing Up Smith en
4757                              Thriller Drama 8 Days en
4758                            Family Running Forever en
4759       Documentary To Be Frank, Sinatra at 100 en
Length: 4760, dtype: object
```

4760 rows × 4 columns

```
x.shape
```

```
(4760,)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tf=TfidfVectorizer()
```

```
x=tf.fit_transform(x)
```

```
x.shape
```

```
(4760, 4678)
```

```
print(x)
```

```
  (0, 1367)     0.09007714202335212
  (0, 3486)     0.7429127511854295
  (0, 1613)     0.5912191932404761
  (0, 869)      0.1708587827138408
  (0, 976)      0.24745483330221305
  (1, 4487)     0.6508001682997612
  (1, 3902)     0.550439231907864
  (1, 1509)     0.2814439744740556
  (1, 3582)     0.2817769790975338
  (1, 86)       0.21313740258007258
  (1, 102)      0.2462864718546055
  (1, 1367)     0.09371359348263858
```

```
  (2, 2862)     0.6732410589493643
  (2, 1523)     0.6200613847496372
  (2, 1467)     0.24661415234213488
  (2, 208)      0.30787730152862025
  (2, 1367)     0.08162954584662571
  (3, 1850)     0.6742499796285102
  (3, 1602)     0.6742499796285102
  (3, 3475)     0.2057784569495271
  (3, 1238)     0.13304095730802998
  (3, 1367)     0.08175187608144899
  (3, 869)      0.15506737578583457
  (4, 401)      0.7754819374908652
  (4, 177)      0.5986761264571135
  :     :
  (4755, 2027)  0.2706809764473197
  (4755, 1367)  0.08948840682655666
  (4756, 1833)  0.6172310324083766
  (4756, 4359)  0.4518371988005941
  (4756, 3790)  0.5684756204008815
  (4756, 1238)  0.121790151891555
  (4756, 1467)  0.2260971844085065
  (4756, 1367)  0.07483840771174535
  (4756, 869)   0.14195387369812426
  (4757, 1072)  0.8961993131347343
  (4757, 4150)  0.3345007373814785
  (4757, 1238)  0.2483049922847849
  (4757, 1367)  0.15258007286186046
  (4758, 3518)  0.6750368818728584
  (4758, 1597)  0.6750368818728584
  (4758, 1467)  0.28266019908870965
  (4758, 1367)  0.09356082553006834
  (4759, 2)     0.4622024356803555
  (4759, 3737)  0.4622024356803555
  (4759, 1622)  0.44084567731400226
  (4759, 389)   0.36782643816611105
  (4759, 1185)  0.25065051674667876
  (4759, 289)   0.335899903341829
  (4759, 4187)  0.2535775056782416
  (4759, 1367)  0.05604140509889691
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
ss=cosine_similarity(x)
```

```
ss
```

```
array([[1.        , 0.00844145, 0.00735296, ..., 0.01374398, 0.00842769,
        0.00504805],
       [0.00844145, 1.        , 0.0076498 , ..., 0.01429883, 0.00876792,
        0.00525184],
       [0.00735296, 0.0076498 , 1.        , ..., 0.01245504, 0.07734533,
        0.00457463],
```

```
       ...,
       [0.01374398, 0.01429883, 0.01245504, ..., 1.        , 0.01427552,
        0.0085508 ],
       [0.00842769, 0.00876792, 0.07734533, ..., 0.01427552, 1.        ,
        0.00524328],
       [0.00504805, 0.00525184, 0.00457463, ..., 0.0085508 , 0.00524328,
        1.        ]])
```

```
ss.shape
```

```
(4760, 4760)
```

```
favourite_movie_name=input('enter movie name:')
```

```
enter movie name:avatar
```

```
movie_title_list=df4['Movie_Title'].tolist()
```

```
import difflib
```

```
movie_recommendation=difflib.get_close_matches(favourite_movie_name,movie_title_list)
print(movie_recommendation)
```

```
['Avatar']
```

```
index_of_close_matchmovie=df4['Movie_ID'].values[0]
print(index_of_close_matchmovie)
```

```
1
```

```
recommended_score=list(enumerate(ss[index_of_close_matchmovie]))
print(recommended_score)
```

```
[(0, 0.008441452669654322), (1, 1.0000000000000002), (2, 0.00764979807564309), (3, 0.007
```

```
len(recommended_score)
```

```
4760
```

```
sorted_similar_movies=sorted(recommended_score,key=lambda x:x[1],reverse=True)
print(sorted_similar_movies)
```

```
[(1, 1.0000000000000002), (4676, 0.7418973411852932), (601, 0.6207065009413768), (2233,
```

```
print('top 10 suggested movies : \n')
```

```
i=1
for movie in sorted_similar_movies:
  index=movie[0]
  title_from_index=df4[df4.index==index]['Movie_Title'].values[0]
  if(i<=10):
    print(i,')', title_from_index)
    i+=1
```

```
top 10 suggested movies :

1 ) Star Wars
2 ) Star Wars: Clone Wars: Volume 1
3 ) Star Wars: Episode I - The Phantom Menace
4 ) Star Trek
5 ) Star Wars: Episode III - Revenge of the Sith
6 ) Star Wars: Episode II - Attack of the Clones
7 ) U.F.O.
8 ) Star Trek Beyond
9 ) Star Trek: Generations
10 ) Star Trek: Insurrection
```

```
df5=pd.read_csv("https://github.com/YBI-foundation/Dataset/raw/main/WhiteWineQuality.csv",sep
```

```
df5.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulp
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	

```
df5.tail()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	s
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	

```
df5.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	d
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.

```
df5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

```
df5.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
df5.shape
```

```
(4898, 12)
```

```
df5['quality'].value_counts()
```

```
6    2198
5    1457
7     880
8     175
4     163
3      20
9       5
Name: quality, dtype: int64
```

```
df5.groupby('quality').mean()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	densi
quality								
3	7.600000	0.333250	0.336000	6.392500	0.054300	53.325000	170.600000	0.9948
4	7.129448	0.381227	0.304233	4.628221	0.050098	23.358896	125.279141	0.9942
5	6.933974	0.302011	0.337653	7.334969	0.051546	36.432052	150.904598	0.9952
6	6.837671	0.260564	0.338025	6.441606	0.045217	35.650591	137.047316	0.9939
7	6.734716	0.262767	0.325625	5.186477	0.038191	34.125568	125.114773	0.9924
8	6.657143	0.277400	0.326514	5.671429	0.038314	36.720000	126.165714	0.9922

```
y=df5['quality']
```

```
y.shape
```

```
(4898,)
```

```
y
```

```
0       6
1       6
2       6
3       6
4       6
       ..
4893    6
4894    5
4895    6
4896    7
4897    6
Name: quality, Length: 4898, dtype: int64
```

```
x=df5[['density','pH','sulphates','alcohol']]
```

```
x=df5.drop(['quality'], axis=1)
```

```
x.shape
```

```
(4898, 11)
```

```
x
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	s
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	
...	...	...	...	...	...	...	...	...	...	
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	

```
from sklearn.preprocessing import StandardScaler
```

```
ss=StandardScaler()
```

```
x=ss.fit_transform(x)
```

```
x
```

```
array([[ 1.72096961e-01, -8.17699008e-02,  2.13280202e-01, ...,
        -1.24692128e+00, -3.49184257e-01, -1.39315246e+00],
       [-6.57501128e-01,  2.15895632e-01,  4.80011213e-02, ...,
         7.40028640e-01,  1.34184656e-03, -8.24275678e-01],
       [ 1.47575110e+00,  1.74519434e-02,  5.43838363e-01, ...,
```

```
          4.75101984e-01, -4.36815783e-01, -3.36667007e-01],
        ...,
        [-4.20473102e-01, -3.79435433e-01, -1.19159198e+00, ...,
         -1.31315295e+00, -2.61552731e-01, -9.05543789e-01],
        [-1.60561323e+00,  1.16673788e-01, -2.82557040e-01, ...,
          1.00495530e+00, -9.62604939e-01,  1.85757201e+00],
        [-1.01304317e+00, -6.77100966e-01,  3.78559282e-01, ...,
          4.75101984e-01, -1.48839409e+00,  1.04489089e+00]])
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.4,random_state=2529)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((1959, 11), (2939, 11), (1959,), (2939,))
```

```
from sklearn.svm import SVC
```

```
svc=SVC()
```

```
svc.fit(x_train,y_train)
```

```
SVC()
```

```
y_pred=svc.predict(x_test)
```

```
y_pred.shape
```

```
(2939,)
```

```
y_pred
```

```
array([6, 5, 6, ..., 6, 6, 5])
```

```
from sklearn.metrics import confusion_matrix,classification_report
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[  0   0   6   6   0   0   0]
 [  0   1  62  34   4   0   0]
 [  0   0 499 350   4   0   0]
 [  0   0 273 982  61   0   0]
 [  0   0  23 389 132   0   0]
 [  0   0   1  83  28   0   0]
 [  0   0   0   0   1   0   0]]
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00        12
           4       1.00      0.01      0.02       101
           5       0.58      0.58      0.58       853
           6       0.53      0.75      0.62      1316
           7       0.57      0.24      0.34       544
           8       0.00      0.00      0.00       112
           9       0.00      0.00      0.00         1

    accuracy                           0.55      2939
   macro avg       0.38      0.23      0.22      2939
weighted avg       0.55      0.55      0.51      2939

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
```

```
y=df5['quality'].apply(lambda y_value: 1 if y_value>=10 else 0)
```

```
y.value_counts()
```

```
0    4898
Name: quality, dtype: int64
```

```
df2_new=df5.sample(1)
```

```
df2_new
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	s

```
df2_new.shape
```

```
(1, 12)
```

```
x_new=df2_new.drop(['quality'],axis=1)
```

```
x_new=ss.fit_transform(x_new)
```

```
y_pred_new=svc.predict(x_new)
```

```
y_pred_new
```

```
array([6])
```

big sales prediction

```
df6=pd.read_csv(r"https://raw.githubusercontent.com/YBI-Foundation/Dataset/main/Big%20Sales%2
```

```
df6.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP
0	FDT36	12.3	Low Fat	0.111448	Baking Goods	33.4874
1	FDT36	12.3	Low Fat	0.111904	Baking Goods	33.9874
2	FDT36	12.3	LF	0.111728	Baking Goods	33.9874
3	FDT36	12.3	Low Fat	0.000000	Baking Goods	34.3874
4	FDP12	9.8	Regular	0.045523	Baking Goods	35.0874

```
df6.tail()
```

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item
				Starchy	

```
df6.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outl
count	11815.000000	14204.000000	14204.000000	14204.000000	1420
mean	12.788355	0.065953	141.004977	1997.830681	218
std	4.654126	0.051459	62.086938	8.371664	182
min	4.555000	0.000000	31.290000	1985.000000	3
25%	8.710000	0.027036	94.012000	1987.000000	92
50%	12.500000	0.054021	142.247000	1999.000000	176
75%	16.750000	0.094037	185.855600	2004.000000	298
max	30.000000	0.328391	266.888400	2009.000000	3122

```
df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            14204 non-null  object
 1   Item_Weight                11815 non-null  float64
 2   Item_Fat_Content           14204 non-null  object
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  object
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  object
 7   Outlet_Establishment_Year  14204 non-null  int64
 8   Outlet_Size                14204 non-null  object
 9   Outlet_Location_Type       14204 non-null  object
 10  Outlet_Type                14204 non-null  object
 11  Item_Outlet_Sales          14204 non-null  float64
dtypes: float64(4), int64(1), object(7)
memory usage: 1.3+ MB
```

```
df6.shape
```

```
(14204, 12)
```

```
df6.columns
```

```
Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility',
       'Item_Type', 'Item_MRP', 'Outlet_Identifier',
```

```
       'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
       'Outlet_Type', 'Item_Outlet_Sales'],
      dtype='object')
```

```
df6['Item_Weight'].fillna(df6.groupby(['Item_Type'])['Item_Weight'].transform('mean'),inplace
```

```
df6.info()
```

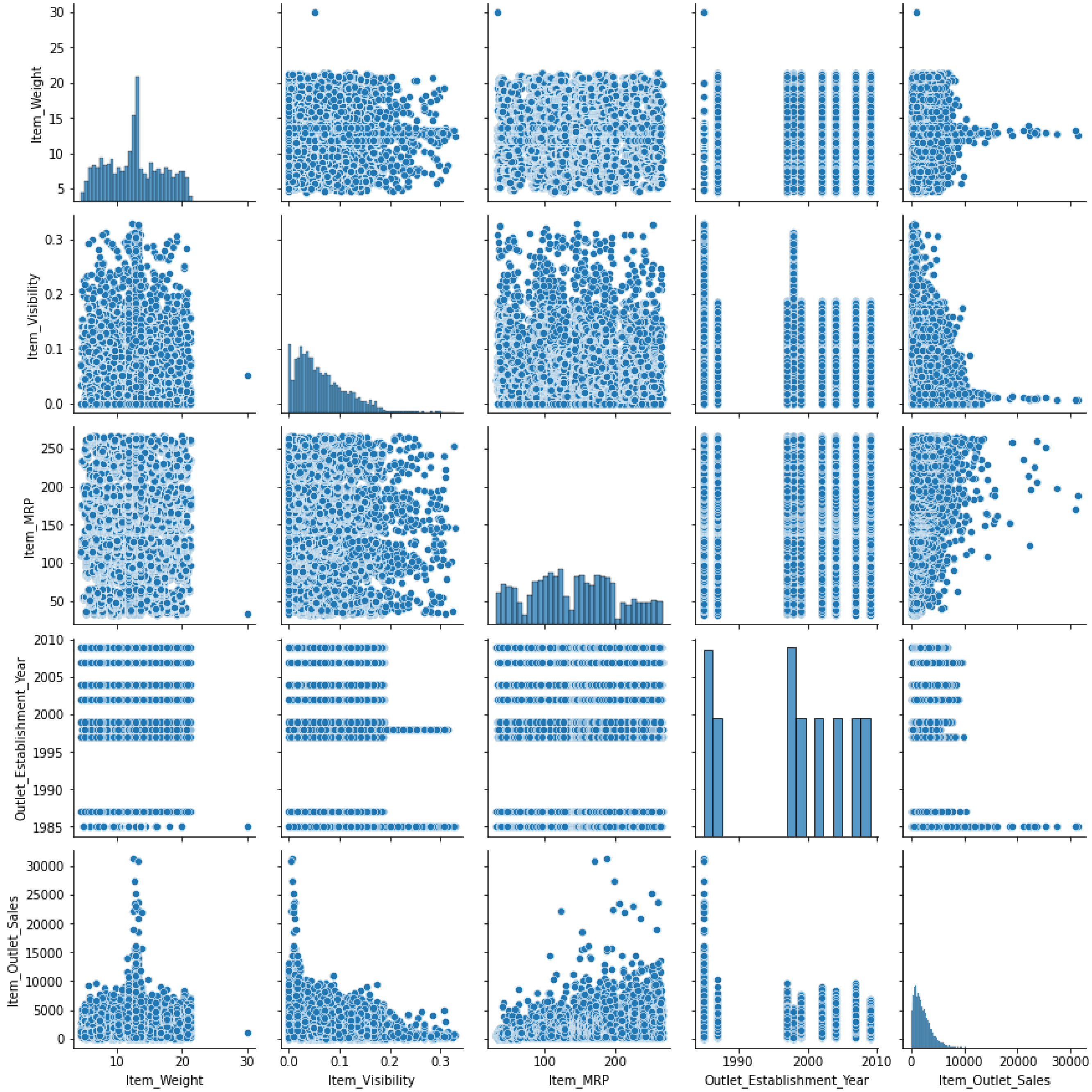
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            14204 non-null  object 
 1   Item_Weight                14204 non-null  float64
 2   Item_Fat_Content           14204 non-null  object 
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  object 
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  object 
 7   Outlet_Establishment_Year  14204 non-null  int64  
 8   Outlet_Size                14204 non-null  object 
 9   Outlet_Location_Type       14204 non-null  object 
 10  Outlet_Type                14204 non-null  object 
 11  Item_Outlet_Sales          14204 non-null  float64
dtypes: float64(4), int64(1), object(7)
memory usage: 1.3+ MB
```

```
df6.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outl
count	14204.000000	14204.000000	14204.000000	14204.000000	1420
mean	12.790642	0.065953	141.004977	1997.830681	218
std	4.251186	0.051459	62.086938	8.371664	182
min	4.555000	0.000000	31.290000	1985.000000	3
25%	9.300000	0.027036	94.012000	1987.000000	92
50%	12.800000	0.054021	142.247000	1999.000000	176
75%	16.000000	0.094037	185.855600	2004.000000	298
max	30.000000	0.328391	266.888400	2009.000000	312

```
sn.pairplot(df6)
```

```
<seaborn.axisgrid.PairGrid at 0x7fe585354cd0>
```




```
df6['Item_Weight'].value_counts()
```

```
13.194406    346
12.867687    335
13.332012    261
```

```
12.566713    250
13.238358    195
            ...
5.860000       7
7.850000       6
9.035000       6
4.615000       6
30.000000      1
Name: Item_Weight, Length: 432, dtype: int64
```

```
df6['Item_Fat_Content'].value_counts()
```

```
Low Fat    8485
Regular    4824
LF          522
reg         195
low fat     178
Name: Item_Fat_Content, dtype: int64
```

```
df6.replace({'Item_Fat_Content':{'Low Fat':0,'Regular':1}},inplace=True)
```

```
df6['Item_Fat_Content'].value_counts()
```

```
0          8485
1          4824
LF          522
reg         195
low fat     178
Name: Item_Fat_Content, dtype: int64
```

```
df6['Item_Type'].value_counts()
```

```
Fruits and Vegetables    2013
Snack Foods              1989
Household                1548
Frozen Foods             1426
Dairy                    1136
Baking Goods             1086
Canned                   1084
Health and Hygiene        858
Meat                      736
Soft Drinks               726
Breads                    416
Hard Drinks               362
Others                    280
Starchy Foods             269
Breakfast                 186
Seafood                    89
Name: Item_Type, dtype: int64
```

```
df6.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP
0	FDT36	12.3	0	0.111448	Baking Goods	33.4874
1	FDT36	12.3	0	0.111904	Baking Goods	33.9874
2	FDT36	12.3	LF	0.111728	Baking Goods	33.9874
3	FDT36	12.3	0	0.000000	Baking Goods	34.3874
4	FDP12	9.8	1	0.045523	Baking Goods	35.0874

```
y=df6['Item_Outlet_Sales']
```

```
y
```

```
0         436.608721
1         443.127721
2         564.598400
3        1719.370000
4         352.874000
            ...
14199    4984.178800
14200    2885.577200
14201    2885.577200
14202    3803.676434
14203    3644.354765
Name: Item_Outlet_Sales, Length: 14204, dtype: float64
```

```
y.shape
```

```
(14204,)
```

```
x=df6[['Item_Type','Item_Weight','Item_MRP',"Outlet_Size",'Outlet_Type']]
```

```
x=df6.drop(['Item_Identifier'],axis=1)
```

```
x.shape
```

```
(14204, 11)
```

```
x
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	12.300000	0	0.111448	Baking Goods	33.4874	C
1	12.300000	0	0.111904	Baking Goods	33.9874	C
2	12.300000	LF	0.111728	Baking Goods	33.9874	C
3	12.300000	0	0.000000	Baking Goods	34.3874	C
4	9.800000	1	0.045523	Baking Goods	35.0874	C
...	...	...	...	...	...	
14199	12.800000	0	0.069606	Starchy Foods	261.9252	C
14200	12.800000	0	0.070013	Starchy Foods	262.8252	C
14201	12.800000	0	0.069561	Starchy Foods	263.0252	C
14202	13.659758	0	0.069282	Starchy Foods	263.5252	C
14203	12.800000	0	0.069727	Starchy Foods	263.6252	C

14204 rows × 11 columns

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
```

```
x_std=df6[['Item_Weight','Item_MRP','Item_Visibility']]
```

```
x_std=sc.fit_transform(x_std)
```

```
x_std
```

```
array([[-0.11541705, -1.73178716,  0.88413635],
       [-0.11541705, -1.72373366,  0.89300616],
       [-0.11541705, -1.72373366,  0.88958331],
```

```
       ...,
       [ 0.00220132,  1.96538148,  0.07011952],
       [ 0.20444792,  1.97343499,  0.06469366],
       [ 0.00220132,  1.97504569,  0.07334891]])
```

```
x[['Item_Weight','Item_MRP','Item_Visibility']]=pd.DataFrame(x_std,columns=[['Item_Weight','I
```

```
x
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	-0.115417	0	0.884136	Baking Goods	-1.731787	(
1	-0.115417	0	0.893006	Baking Goods	-1.723734	(
2	-0.115417	LF	0.889583	Baking Goods	-1.723734	(
3	-0.115417	0	-1.281712	Baking Goods	-1.717291	(
4	-0.703509	1	-0.397031	Baking Goods	-1.706016	(
...	...	...	...	...	...	
14199	0.002201	0	0.070990	Starchy Foods	1.947664	(
14200	0.002201	0	0.078898	Starchy Foods	1.962160	(
14201	0.002201	0	0.070120	Starchy Foods	1.965381	(
14202	0.204448	0	0.064694	Starchy Foods	1.973435	(
14203	0.002201	0	0.073349	Starchy Foods	1.975046	(

14204 rows × 11 columns

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.2,random_state=2529)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((2840, 11), (11364, 11), (2840,), (11364,))
```

```
from sklearn.linear_model import LogisticRegression
```

```
lr=LogisticRegression()
```

```
lr.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
y_pred=lr.predict(x_test)
```

```
from sklearn.svm import SVC
```

```
sc=SVC()
```

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
mean_absolute_error(y_test,y_pred)
```

```
0.42438130155820347
```

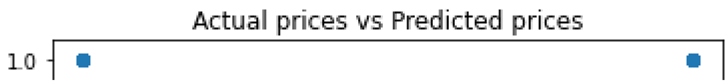
```
mean_squared_error(y_test,y_pred)
```

```
0.42438130155820347
```

```
r2_score(y_test,y_pred)
```

```
-0.6975266323890179
```

```
mp.scatter(y_test,y_pred)
mp.title('Actual prices vs Predicted prices')
mp.show()
```



hill and valley prediction

```
df7=pd.read_csv("https://github.com/YBI-foundation/Dataset/raw/main/Hill%20Valley%20Dataset.c
```

```
df7.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	39.02	36.49	38.20	38.85	39.38	39.74	37.02	39.53	38.81
1	1.83	1.71	1.77	1.77	1.68	1.78	1.80	1.70	1.75
2	68177.69	66138.42	72981.88	74304.33	67549.66	69367.34	69169.41	73268.61	74465.84
3	44889.06	39191.86	40728.46	38576.36	45876.06	47034.00	46611.43	37668.32	40980.89
4	5.70	5.40	5.28	5.38	5.27	5.61	6.00	5.38	5.34

5 rows × 101 columns

```
df7.tail()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9
1207	13.00	12.87	13.27	13.04	13.19	12.53	14.31	13.33	13.63
1208	48.66	50.11	48.55	50.43	50.09	49.67	48.95	48.65	48.63
1209	10160.65	9048.63	8994.94	9514.39	9814.74	10195.24	10031.47	10202.28	9152.99
1210	34.81	35.07	34.98	32.37	34.16	34.03	33.31	32.48	35.63
1211	8489.43	7672.98	9132.14	7985.73	8226.85	8554.28	8838.87	8967.24	8635.14

5 rows × 101 columns

```
df7.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1212 entries, 0 to 1211
Columns: 101 entries, V1 to Class
dtypes: float64(100), int64(1)
memory usage: 956.5 KB
```

```
df7.describe()
```

	V1	V2	V3	V4	V5	
count	1212.000000	1212.000000	1212.000000	1212.000000	1212.000000	1212.0
mean	8169.091881	8144.306262	8192.653738	8176.868738	8128.297211	8173.0
std	17974.950461	17881.049734	18087.938901	17991.903982	17846.757963	17927.1
min	0.920000	0.900000	0.850000	0.890000	0.880000	0.8
25%	19.602500	19.595000	18.925000	19.277500	19.210000	19.5
50%	301.425000	295.205000	297.260000	299.720000	295.115000	294.3
75%	5358.795000	5417.847500	5393.367500	5388.482500	5321.987500	5328.0
max	117807.870000	108896.480000	119031.350000	110212.590000	113000.470000	116848.3

8 rows × 101 columns

```
print(df7.columns.tolist())
```

```
['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14
```

```
df7.shape
```

```
(1212, 101)
```

```
df7['Class'].value_counts()
```

```
0    606
1    606
Name: Class, dtype: int64
```

```
df7.groupby('Class').mean()
```

	V1	V2	V3	V4	V5	V6	
Class							
0	7913.333251	7825.339967	7902.497294	7857.032079	7775.610198	7875.436337	7804.
1	8424.850512	8463.272558	8482.810182	8496.705396	8480.984224	8470.623680	8572.

2 rows × 100 columns

```
y=df7['Class']
```

```
y.shape
```

```
(1212,)
```

```
y
```

```
0       0
1       1
2       1
3       0
4       0
       ..
1207    1
1208    0
1209    1
1210    1
1211    0
Name: Class, Length: 1212, dtype: int64
```

```
x=df7[['V1','V2','V3','V4','V5','V6','V7',"V8","V9","V10"]]
```

```
x=df7.drop('Class',axis=1)
```

```
x.shape
```

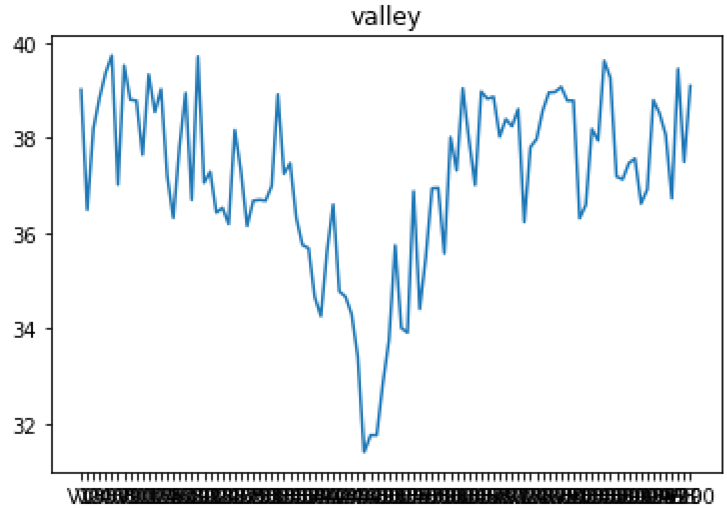
```
(1212, 100)
```

```
x
```

	V1	V2	V3	V4	V5	V6	V7	V8	
0	39.02	36.49	38.20	38.85	39.38	39.74	37.02	39.53	38
1	1.83	1.71	1.77	1.77	1.68	1.78	1.80	1.70	'

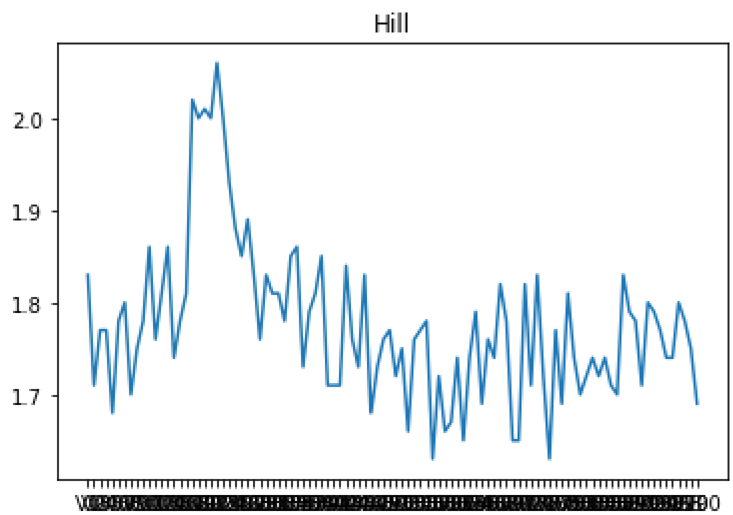
```
mp.plot(x.iloc[0,:])
mp.title('valley')
```

```
Text(0.5, 1.0, 'valley')
```




```
mp.plot(x.iloc[1,:])
mp.title('Hill')
```

```
Text(0.5, 1.0, 'Hill')
```




```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
```

```
x=sc.fit_transform(x)
```

```
x
```

```
array([[-0.45248681, -0.45361784, -0.45100881, ..., -0.45609618,
        -0.45164274, -0.45545496],
       [-0.45455665, -0.45556372, -0.45302369, ..., -0.45821768,
        -0.45362255, -0.45755405],
       [ 3.33983504,  3.24466709,  3.58338069, ...,  3.5427869 ,
         3.27907378,  3.74616847],
       ...,
       [ 0.11084204,  0.0505953 ,  0.04437307, ...,  0.12533312,
         0.04456025,  0.06450317],
       [-0.45272112, -0.45369729, -0.45118691, ..., -0.45648861,
        -0.45190136, -0.45569511],
       [ 0.01782872, -0.02636986,  0.05196137, ...,  0.03036056,
         0.01087365,  0.03123129]])
```

```
x.shape
```

```
(1212, 100)
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.1,stratify=y,random_state=253
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((121, 100), (1091, 100), (121,), (1091,))
```

```
from sklearn.linear_model import LogisticRegression
```

```
lr=LogisticRegression()
```

```
lr.fit(x_train,y_train )
```

```
LogisticRegression()
```

```
y_pred=lr.predict(x_test)
```

```
y_pred.shape
```

```
(1091,)
```

```
y_pred
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
lr.predict_proba(x_test)
```

```
array([[0.49339312, 0.50660688],
       [0.49226548, 0.50773452],
       [0.49340597, 0.50659403],
       ...,
       [0.49339979, 0.50660021],
       [0.49342196, 0.50657804],
       [0.49339765, 0.50660235]])
```

```
from sklearn.metrics import confusion_matrix,classification_report
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[101 445]
 [ 18 527]]
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.18      0.30       546
           1       0.54      0.97      0.69       545

    accuracy                           0.58      1091
   macro avg       0.70      0.58      0.50      1091
weighted avg       0.70      0.58      0.50      1091
```

```
x1_new=df7.sample(2)
```

```
x1_new
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	
1202	1.86	1.93	1.91	1.90	1.97	1.89	2.02	1.87	2.02	2.03	...	
612	409.53	393.75	388.72	388.45	413.10	406.14	415.73	413.29	411.99	411.11	...	41

2 rows × 101 columns

```
x1_new.shape
```

```
(2, 101)
```

```
x1_new=x1_new.drop('Class',axis=1)
```

```
x1_new
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	
1202	1.86	1.93	1.91	1.90	1.97	1.89	2.02	1.87	2.02	2.03	...	
612	409.53	393.75	388.72	388.45	413.10	406.14	415.73	413.29	411.99	411.11	...	38

2 rows × 100 columns

```
x1_new
```


	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	
1202	1.86	1.93	1.91	1.90	1.97	1.89	2.02	1.87	2.02	2.03	...	
612	409.53	393.75	388.72	388.45	413.10	406.14	415.73	413.29	411.99	411.11	...	38

2 rows × 100 columns


```
x1_new=ss.fit_transform(x1_new)
```

```
y_pred_new=lr.predict(x1_new)
```

```
y_pred_new
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
lr.predict_proba(x1_new)
```

```
array([[0.49299481, 0.50700519],
       [0.49193574, 0.50806426],
       [0.49300773, 0.50699227],
       ...,
       [0.49300159, 0.50699841],
       [0.49302497, 0.50697503],
       [0.49299931, 0.50700069]])
```