LAPTOP_PRICE_ANALYSIS

Introduction

The Laptop Price Dataset is a comprehensive collection of data related to laptop prices, specifications, and various factors that influence the pricing of laptops. This dataset provides valuable insights into the dynamics of the laptop market, making it a valuable resource for researchers, analysts, and businesses in the technology industry.

Import all the liabraries -

Pandas - Used to analyse data. It has function for analysing,cleaning,exploring and manipulating data.

Numpy - Mostly work on numerical values for making Arithmatic Operations.

Matplotlib - Comprehensive library for creating static,animated and intractive visualization.

Seaborn - Seaborn is a python data visualization library based on matplotlib. It provides a high-level interface for drawing intractive and informative statastical graphics.

Warnings - warnings are provided to warn the developer of situation that are not necessarily exceptions and ignore them.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

By read_csv() function we are reading the dataset present in "laptop_train.csv" file.

Took dataset on kaggle

We will be making predictions models using Machine Learning Algorithams.

In [2]:
```
df=pd.read_csv("laptops_train.csv")
```

In [3]:
```
df
```

Out[3]:

	Manufacturer	Model Name	Category	Screen Size	Screen	CPU	RAM	Storage	GPU	Opera Sys
0	Apple	MacBook Pro	Ultrabook	13.3"	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	ma
1	Apple	Macbook Air	Ultrabook	13.3"	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	ma
2	HP	250 G6	Notebook	15.6"	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No
3	Apple	MacBook Pro	Ultrabook	15.4"	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	ma
4	Apple	MacBook Pro	Ultrabook	13.3"	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	ma
...	...	...	...	...	...	...	...	...	...	
972	Dell	Alienware 17	Gaming	17.3"	Full HD 1920x1080	Intel Core i7 6700HQ 2.6GHz	32GB	256GB SSD + 1TB HDD	Nvidia GeForce GTX 1070	Wind
973	Toshiba	Tecra A40-C-1DF	Notebook	14.0"	Full HD 1920x1080	Intel Core i5 6200U 2.3GHz	8GB	256GB SSD	Intel HD Graphics 520	Wind
974	Asus	Rog Strix	Gaming	17.3"	Full HD 1920x1080	Intel Core i7 7700HQ 2.8GHz	16GB	256GB SSD + 1TB HDD	Nvidia GeForce GTX 1060	Wind
975	HP	Probook 450	Notebook	15.6"	IPS Panel Full HD 1920x1080	Intel Core i5 7200U 2.70GHz	8GB	128GB SSD + 1TB HDD	Nvidia GeForce 930MX	Wind
976	Lenovo	ThinkPad T460	Notebook	14.0"	1366x768	Intel Core i5 6200U 2.3GHz	4GB	508GB Hybrid	Intel HD Graphics 520	Wind

977 rows × 13 columns

Performing Exploratory Data Analysis

Find information about data by using df.info()

This laptops_train.csv dataset contains the information of patients health condition through their tests peroforms during covid19 with 977 rows and 13 columns.

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 977 entries, 0 to 976
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Manufacturer              977 non-null    object 
 1   Model Name                977 non-null    object 
 2   Category                  977 non-null    object 
 3   Screen Size               977 non-null    object 
 4   Screen                    977 non-null    object 
 5   CPU                       977 non-null    object 
 6   RAM                       977 non-null    object 
 7   Storage                   977 non-null    object 
 8   GPU                       977 non-null    object 
 9   Operating System          977 non-null    object 
 10  Operating System Version  841 non-null    object 
 11  Weight                    977 non-null    object 
 12  Price                     977 non-null    float64
dtypes: float64(1), object(12)
memory usage: 99.4+ KB
```

df.info() function display information about your DataFrame, including the data types of each column, the number of non-null values, and memory usage.

This dataset contain 977 rows and 13 columns out of there are 1 numerical column and 12 categorical column.

To find how much null values in DataSet

With the help of isnull().sum() function we got the null count of all columns.

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Manufacturer                0
Model Name                  0
Category                    0
Screen Size                 0
Screen                      0
CPU                         0
RAM                         0
Storage                     0
GPU                         0
Operating System            0
Operating System Version    136
Weight                      0
Price                       0
dtype: int64
```

df.isnull() method in pandas is used to check for missing or null values in a DataFrame.

Drop Unwanted Columns

In [6]:
```
df.drop("Operating System Version",axis=1,inplace=True)
```

use drop method in pandas to remove a column named "Operating System Version" from your DataFrame.

Change Datatype

In [7]:
```
df["Screen Size"] = df["Screen Size"].replace(r'[^0-9.]', '', regex=True)
df["RAM"] = df["RAM"].replace(r'[^0-9.]', '', regex=True)
df["Weight"] = df["Weight"].replace(r'[^0-9.]', '', regex=True)
```

In that removing any characters from the "Screen Size", "RAM", "Weight" column that are not digits (0-9) or a period (.). It uses a regular expression to match and replace those characters with an empty string.

In [8]:
```
df["Screen Size"]=df["Screen Size"].astype("float")
df["RAM"]=df["RAM"].astype("float")
df["Weight"]=df["Weight"].astype("float")
```

is converting the "Screen Size," "RAM," and "Weight" columns to the float data type.

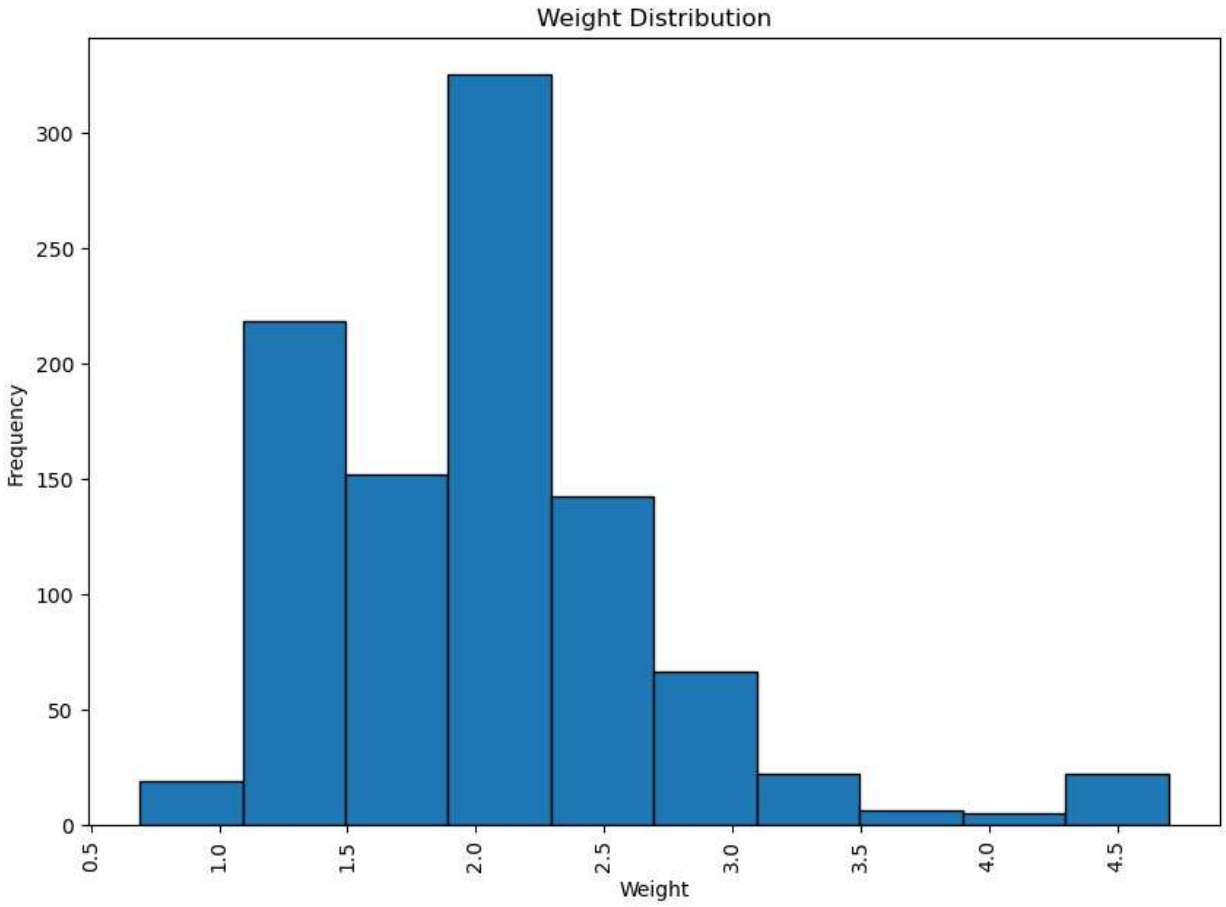
In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 977 entries, 0 to 976
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Manufacturer      977 non-null    object 
 1   Model Name        977 non-null    object 
 2   Category          977 non-null    object 
 3   Screen Size       977 non-null    float64
 4   Screen            977 non-null    object 
 5   CPU               977 non-null    object 
 6   RAM               977 non-null    float64
 7   Storage           977 non-null    object 
 8   GPU               977 non-null    object 
 9   Operating System  977 non-null    object 
 10  Weight            977 non-null    float64
 11  Price             977 non-null    float64
dtypes: float64(4), object(8)
memory usage: 91.7+ KB
```

Histogram

In [10]:
```
plt.figure(figsize=(10, 7))
plt.hist(df["Weight"], bins=10, edgecolor="black")
plt.xlabel("Weight")
plt.ylabel("Frequency")
```

```
plt.title("Weight Distribution")
plt.xticks(rotation=90)
plt.show()
```

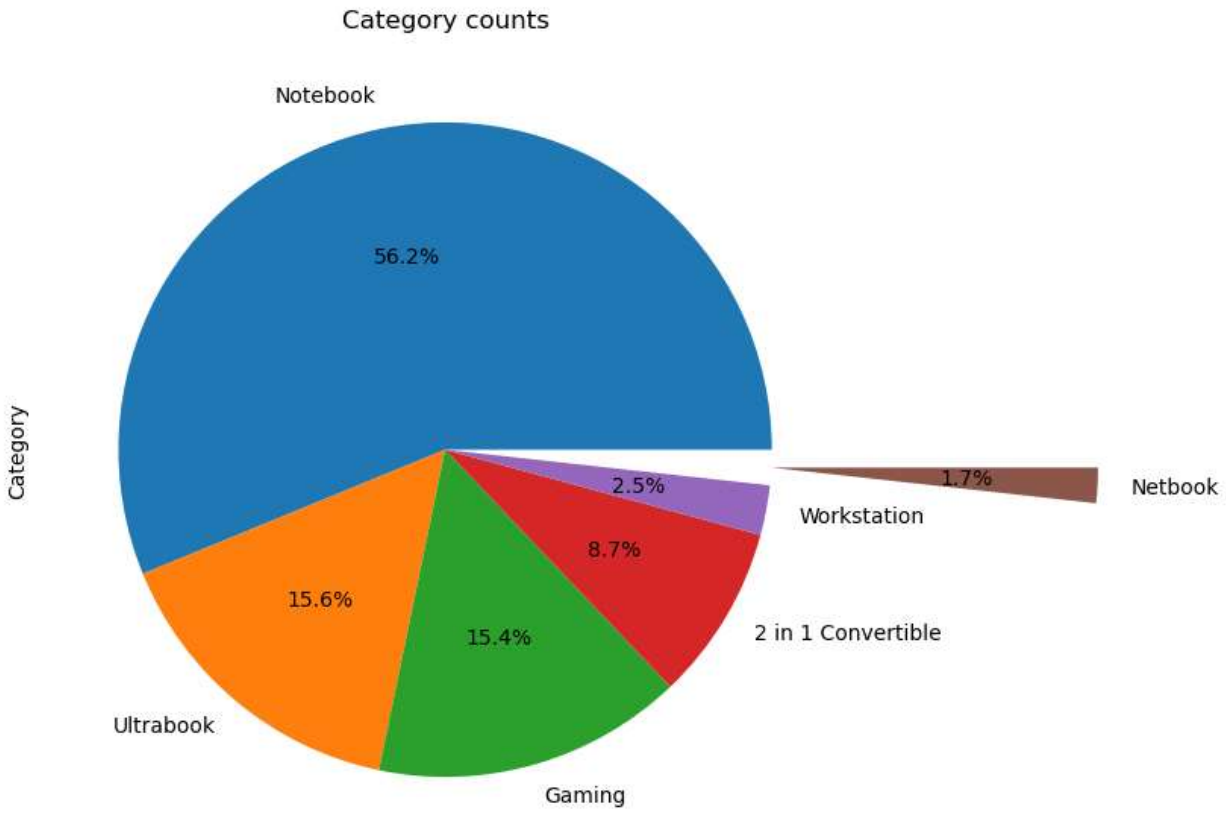


Pieplot

In [11]:
```
df["Category"].value_counts()
```

Out[11]:
```
Notebook              549
Ultrabook             152
Gaming                150
2 in 1 Convertible     85
Workstation            24
Netbook                17
Name: Category, dtype: int64
```

In [12]:
```
plt.figure(figsize=(7,7))
df["Category"].value_counts().plot.pie(autopct="%1.1f%%",explode=(0,0,0,0,0,1))
plt.title("Category counts")
plt.show()
```



Scatterplot

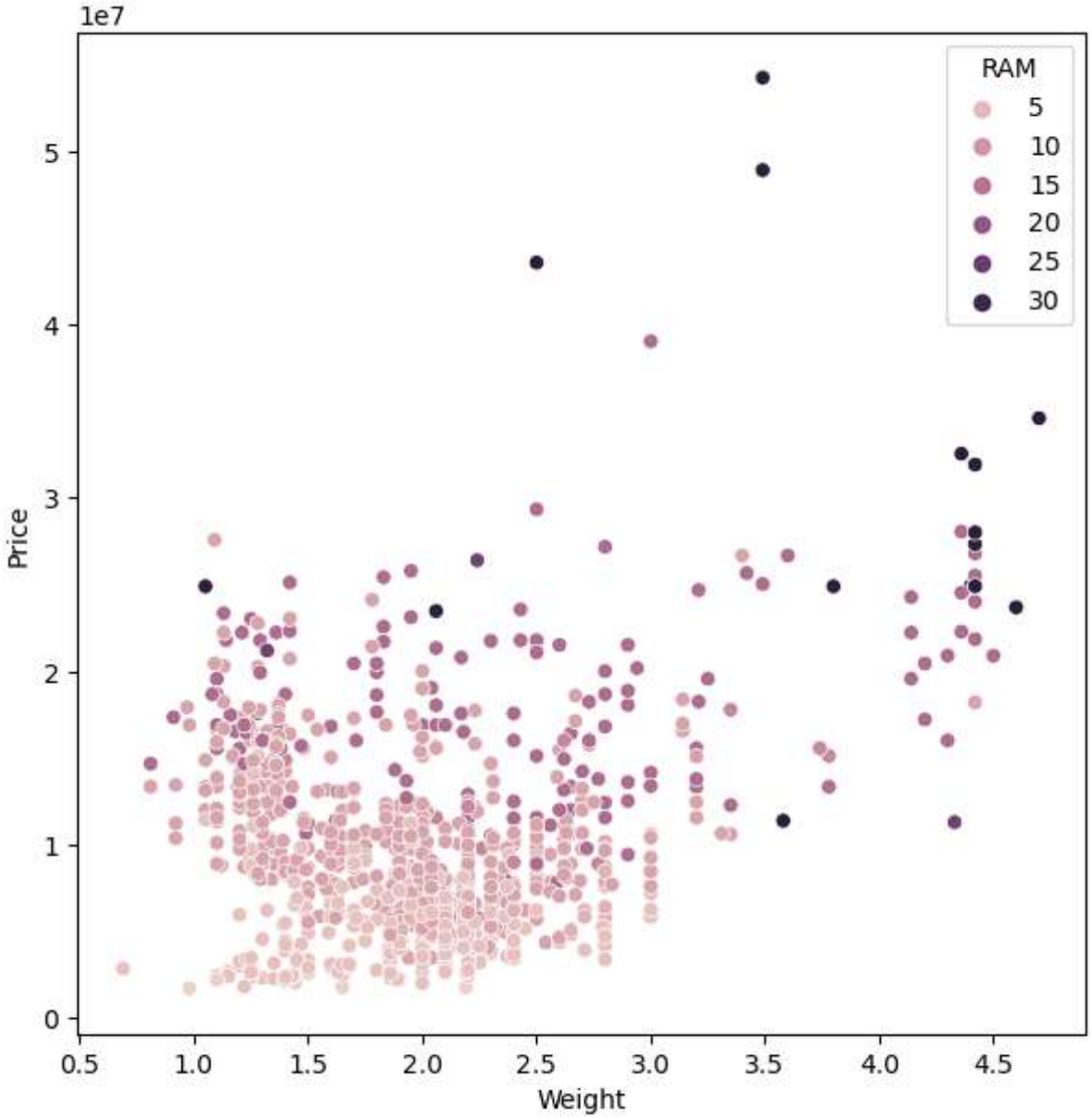
In [13]:
```
df.corr()
```

Out[13]:

	Screen Size	RAM	Weight	Price
Screen Size	1.000000	0.232315	0.822261	0.075152
RAM	0.232315	1.000000	0.390161	0.764005
Weight	0.822261	0.390161	1.000000	0.224415
Price	0.075152	0.764005	0.224415	1.000000

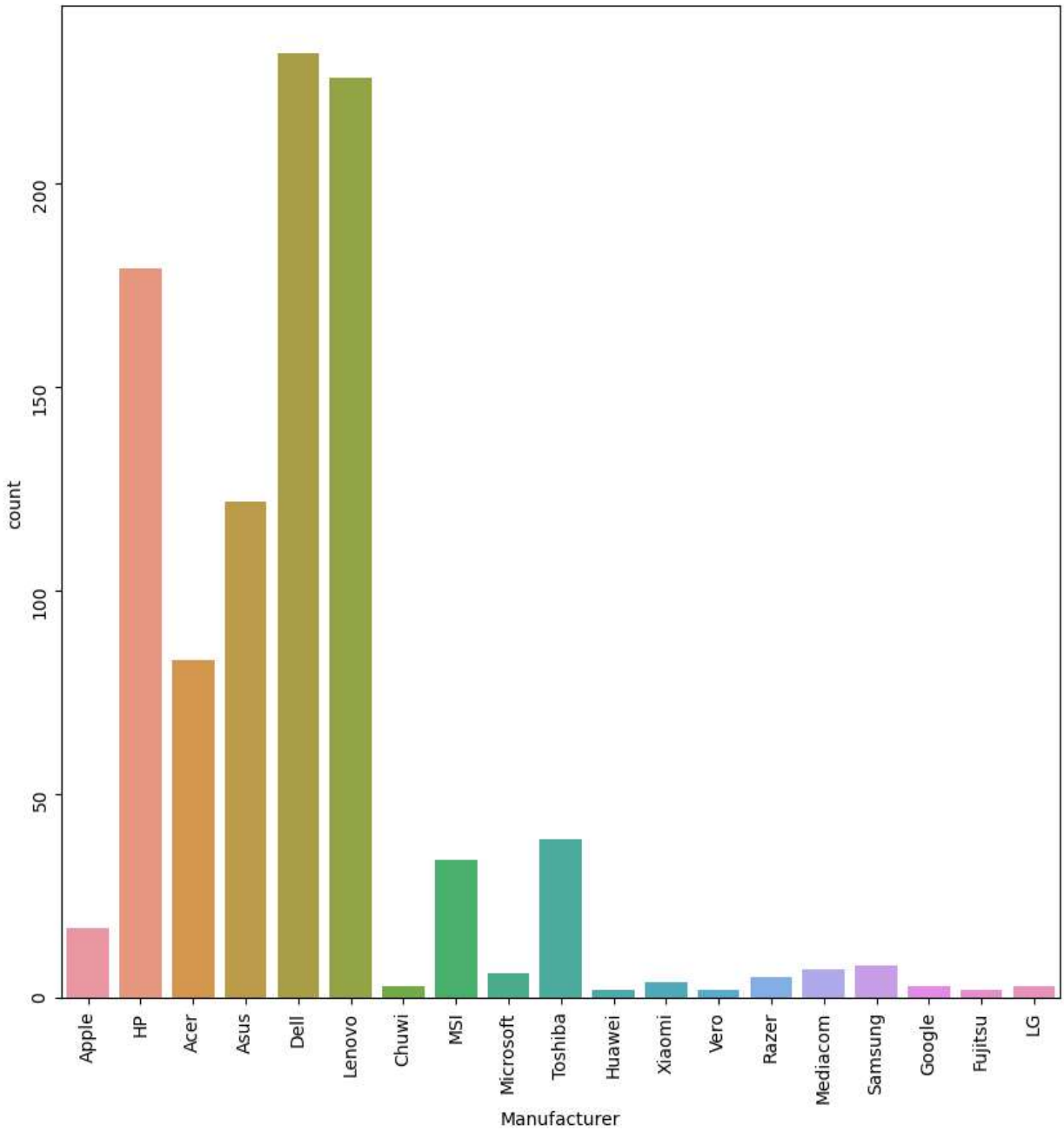
In [14]:
```
plt.figure(figsize=(7,7))
sns.scatterplot(data=df,x="Weight",y="Price",hue="RAM")
plt.plot()
```

Out[14]:
```
[]
```



Countplot

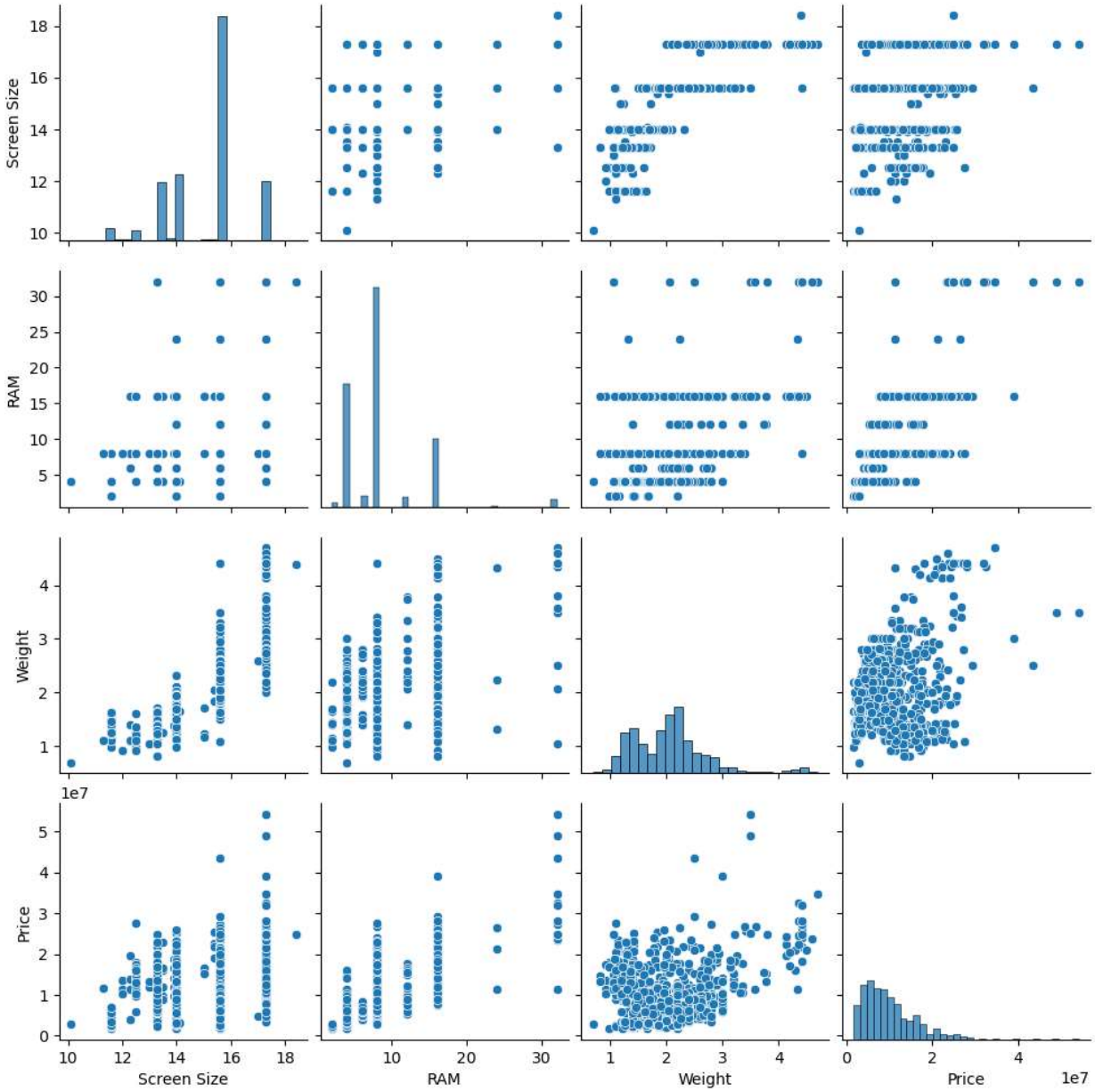
In [15]:
```
plt.figure(figsize=(10,10))
sns.countplot(data=df,x="Manufacturer")
plt.yticks(rotation=90)
plt.xticks(rotation=90)
plt.show()
```



Find Relation between each other

In [16]:
```
sns.pairplot(df)
```

Out[16]:
```
<seaborn.axisgrid.PairGrid at 0x220e1203c70>
```



The pairplot function in seaborn is a great way to quickly visualize relationships between variables.

Divide the Data into Numeric and Categorical form

In [17]:
```
num_feature=df.select_dtypes(["int64","float64"])
cat_feature=df.select_dtypes(["object"]).columns
```

Here we are saperate numerical and categorical features for perform encoding on categorical data.

In [18]:
```
num_feature
```

Out[18]:

	Screen Size	RAM	Weight	Price
0	13.3	8.0	1.37	11912523.48
1	13.3	8.0	1.34	7993374.48
2	15.6	8.0	1.86	5112900.00
3	15.4	16.0	1.83	22563005.40
4	13.3	8.0	1.37	16037611.20
...	...	...	...	...
972	17.3	32.0	4.42	24897600.00
973	14.0	8.0	1.95	10492560.00
974	17.3	16.0	2.73	18227710.80
975	15.6	8.0	2.04	8705268.00
976	14.0	4.0	1.70	8909784.00

977 rows × 4 columns

In [19]:
```
cat_feature
```

Out[19]:
```
Index(['Manufacturer', 'Model Name', 'Category', 'Screen', 'CPU', 'Storage',
       'GPU', 'Operating System'],
      dtype='object')
```

In [20]:
```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
df[cat_feature]=oe.fit_transform(df[cat_feature])
```

OrdinalEncoder is transform categorical features into numerical representations.

In [21]:
```
df
```

Out[21]:

	Manufacturer	Model Name	Category	Screen Size	Screen	CPU	RAM	Storage	GPU	Operating System	Weight
0	1.0	236.0	4.0	13.3	23.0	58.0	8.0	2.0	51.0	6.0	1.37
1	1.0	237.0	4.0	13.3	1.0	56.0	8.0	0.0	45.0	6.0	1.34
2	7.0	38.0	3.0	15.6	8.0	67.0	8.0	15.0	47.0	4.0	1.86
3	1.0	236.0	4.0	15.4	25.0	77.0	16.0	27.0	7.0	6.0	1.83
4	1.0	236.0	4.0	13.3	23.0	60.0	8.0	15.0	52.0	6.0	1.37
...	...	...	...	...	...	...	...	...	...	...	...
972	4.0	43.0	1.0	17.3	8.0	83.0	32.0	16.0	72.0	5.0	4.42
973	16.0	339.0	3.0	14.0	8.0	61.0	8.0	15.0	41.0	5.0	1.95
974	2.0	318.0	1.0	17.3	8.0	93.0	16.0	16.0	71.0	5.0	2.73
975	7.0	293.0	3.0	15.6	15.0	68.0	8.0	3.0	62.0	5.0	2.04
976	10.0	362.0	3.0	14.0	0.0	61.0	4.0	25.0	41.0	5.0	1.70

977 rows × 12 columns

Splitting Data Into Features and Target

In [22]:
```
x=df.iloc[:,:-1]
x
```

Out[22]:

	Manufacturer	Model Name	Category	Screen Size	Screen	CPU	RAM	Storage	GPU	Operating System	Weight
0	1.0	236.0	4.0	13.3	23.0	58.0	8.0	2.0	51.0	6.0	1.37
1	1.0	237.0	4.0	13.3	1.0	56.0	8.0	0.0	45.0	6.0	1.34
2	7.0	38.0	3.0	15.6	8.0	67.0	8.0	15.0	47.0	4.0	1.86
3	1.0	236.0	4.0	15.4	25.0	77.0	16.0	27.0	7.0	6.0	1.83
4	1.0	236.0	4.0	13.3	23.0	60.0	8.0	15.0	52.0	6.0	1.37
...	...	...	...	...	...	...	...	...	...	...	...
972	4.0	43.0	1.0	17.3	8.0	83.0	32.0	16.0	72.0	5.0	4.42
973	16.0	339.0	3.0	14.0	8.0	61.0	8.0	15.0	41.0	5.0	1.95
974	2.0	318.0	1.0	17.3	8.0	93.0	16.0	16.0	71.0	5.0	2.73
975	7.0	293.0	3.0	15.6	15.0	68.0	8.0	3.0	62.0	5.0	2.04
976	10.0	362.0	3.0	14.0	0.0	61.0	4.0	25.0	41.0	5.0	1.70

977 rows × 11 columns

We are splitting overall data except target column that is "target" in x variable

In [23]:
```
y=df["Price"]
y
```

Out[23]:
```
0      11912523.48
1       7993374.48
2       5112900.00
3      22563005.40
4      16037611.20
          ...
972    24897600.00
973    10492560.00
974    18227710.80
975     8705268.00
976     8909784.00
Name: Price, Length: 977, dtype: float64
```

In that separate "target" column as a target in y variable.

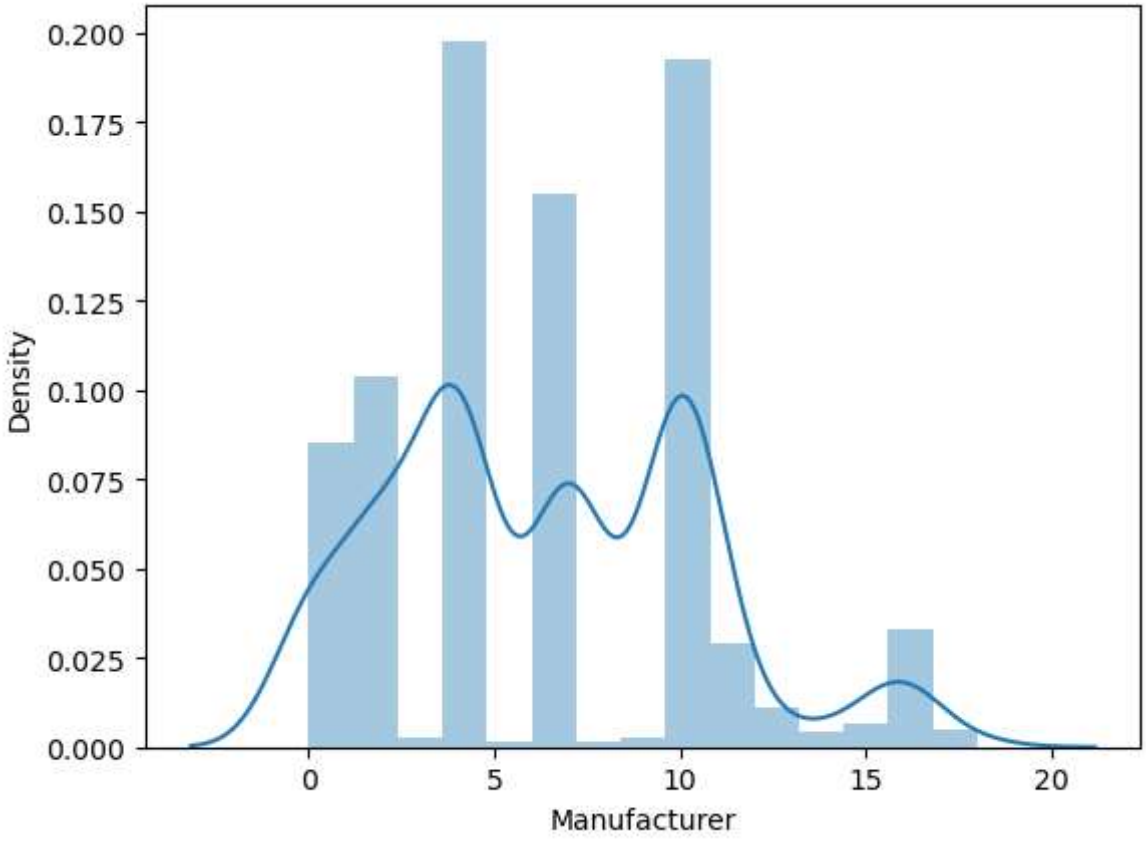
Skewness

In [24]:
```
from scipy.stats import skew
```

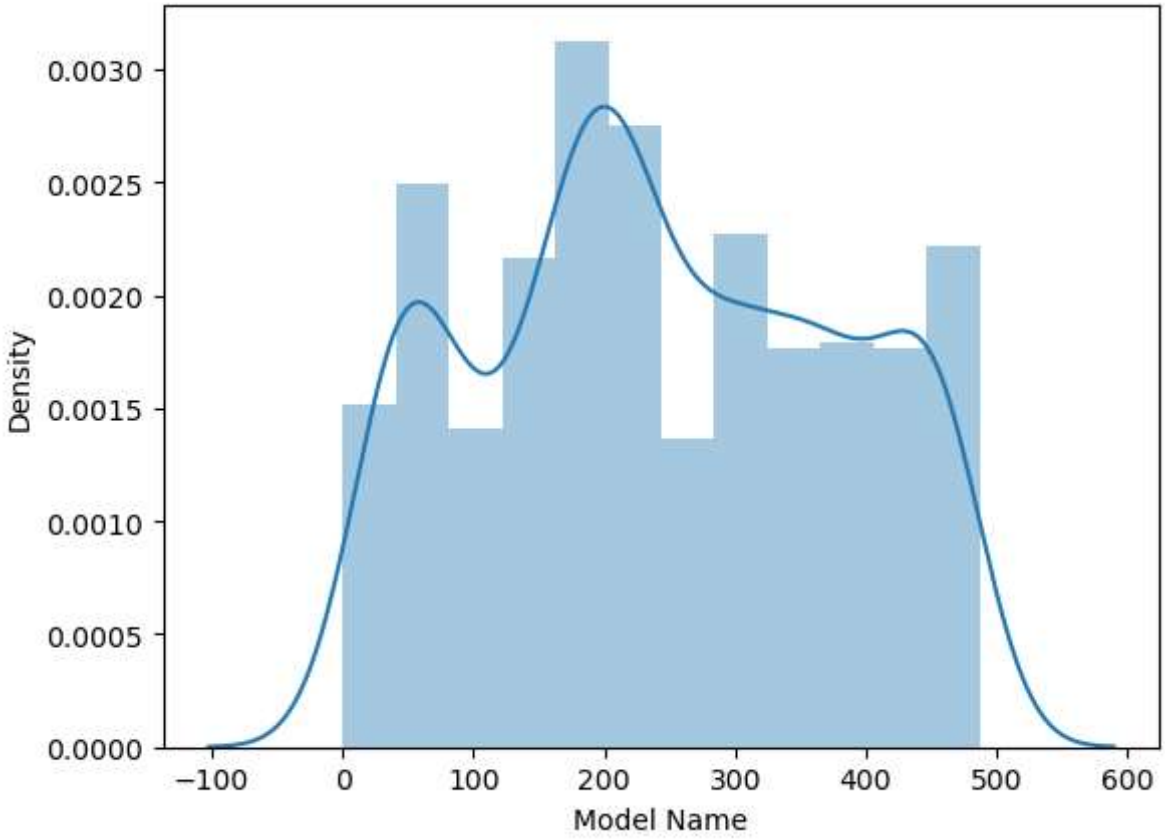
Here we are calculate the skewness and visualize their distributions using sns.distplot from the Seaborn library.

In [25]:
```
from scipy.stats import skew

for i in x:
    print(i)
    print(skew(x[i]))

    plt.figure
    sns.distplot(x[i])
    plt.show()
```

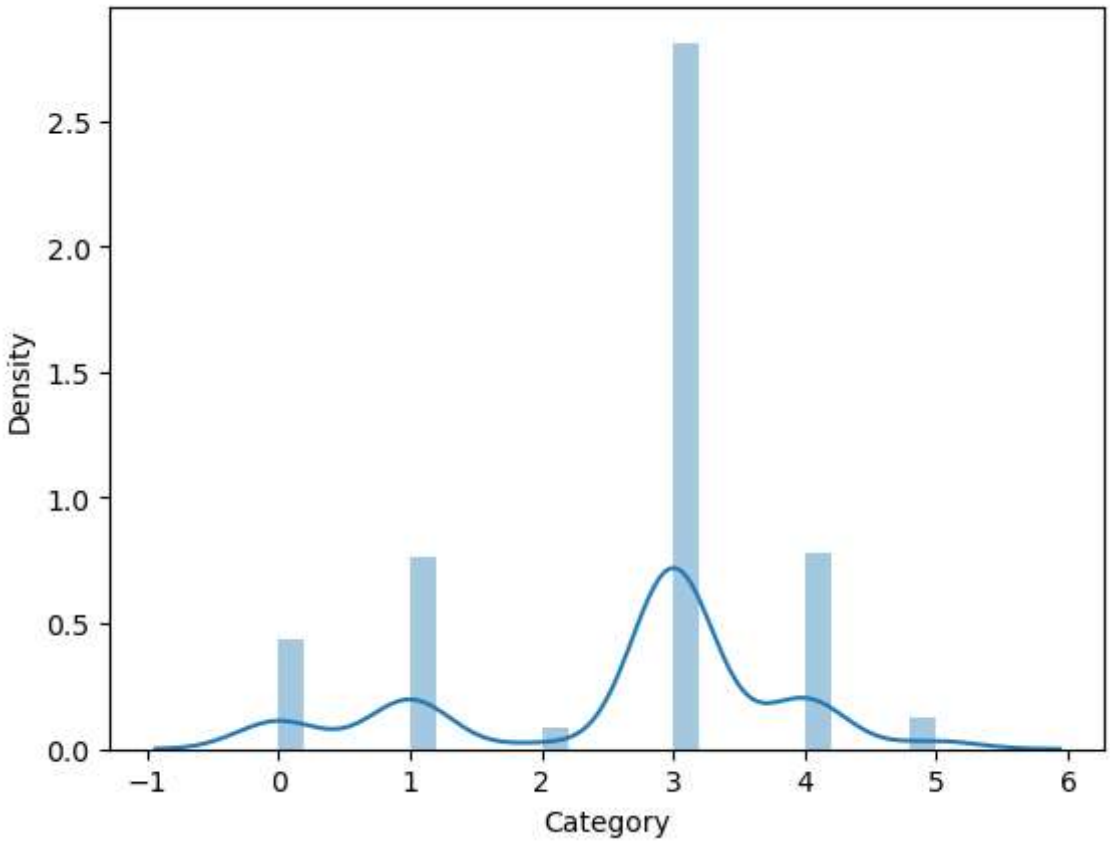
```
Manufacturer
0.4188993369524993
```



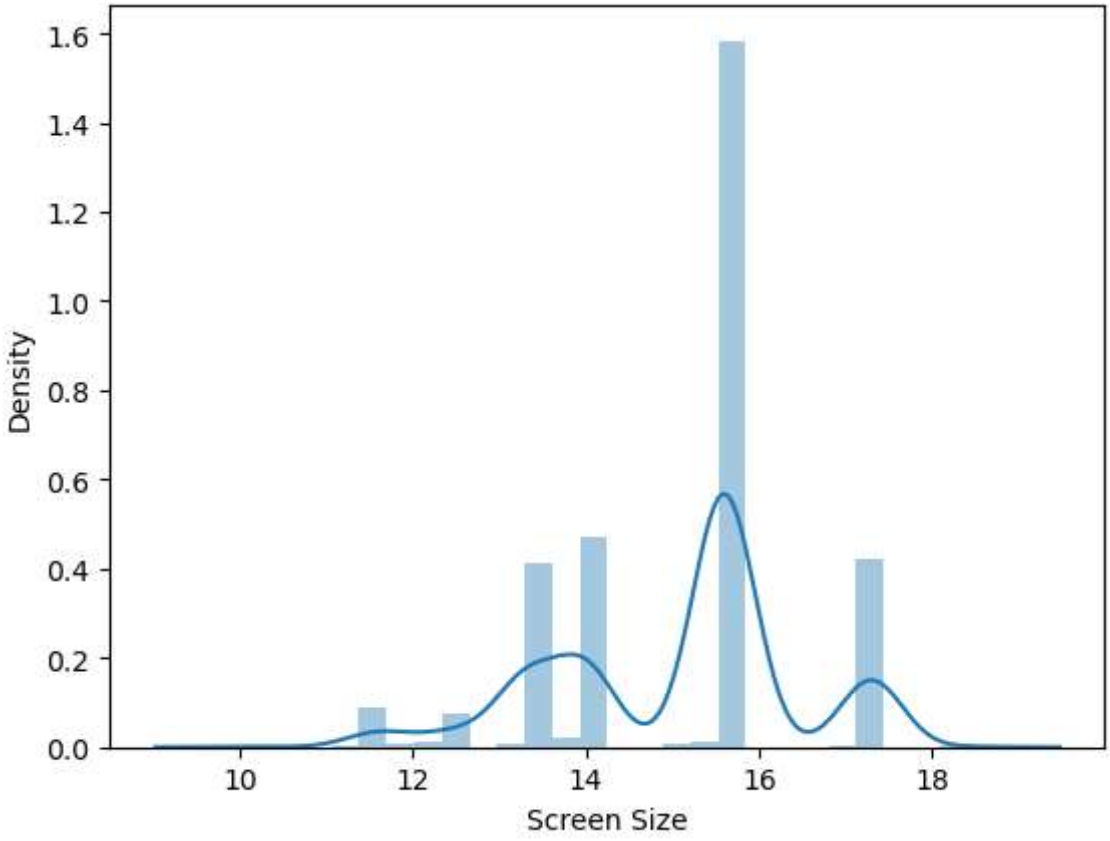

```
Model Name
0.06108869273436197
```



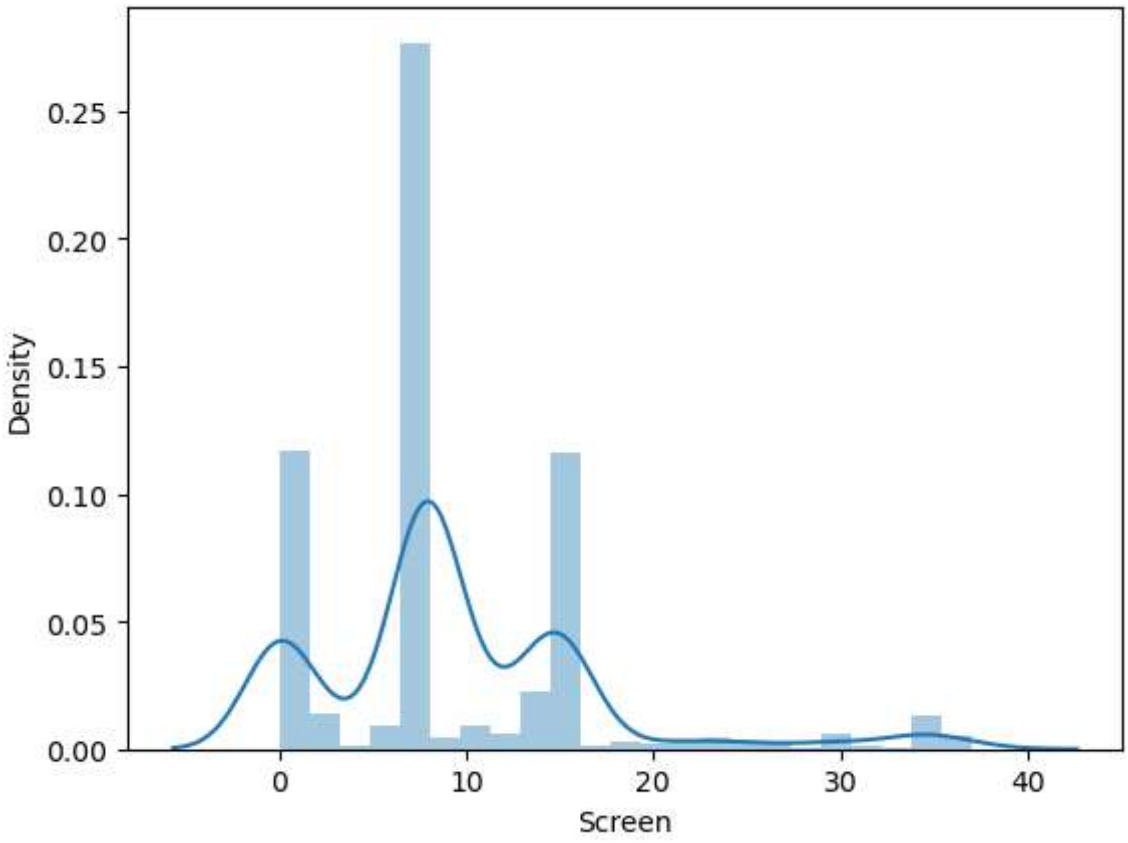

```
Category
-0.7697022136932742
```



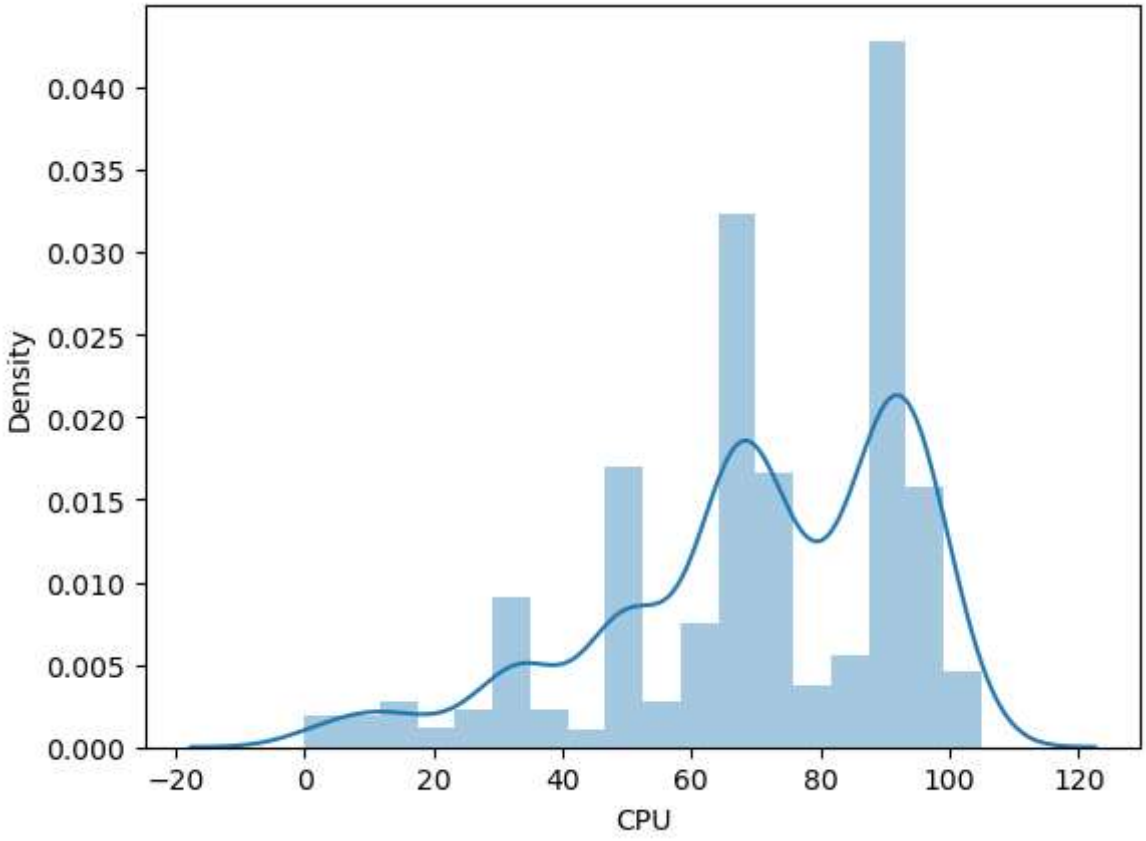

```
Screen Size
-0.39686241113727416
```



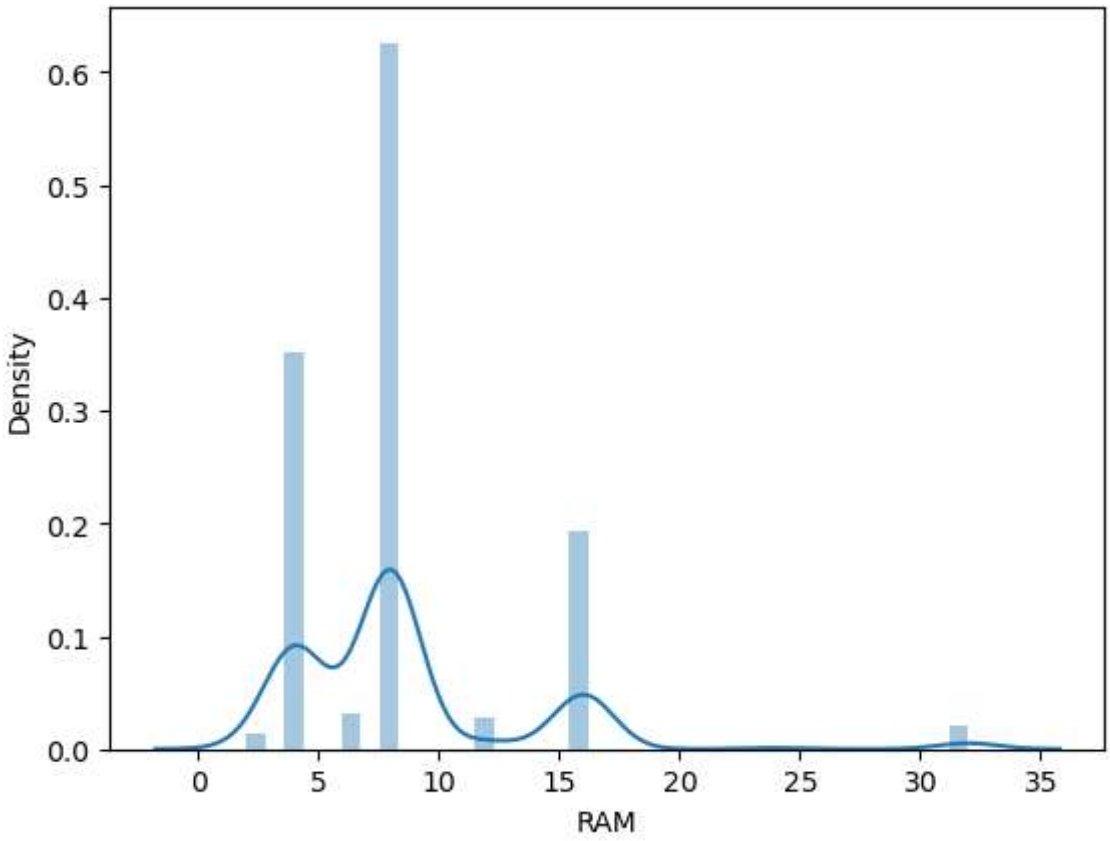

```
Screen
1.3478044984144508
```



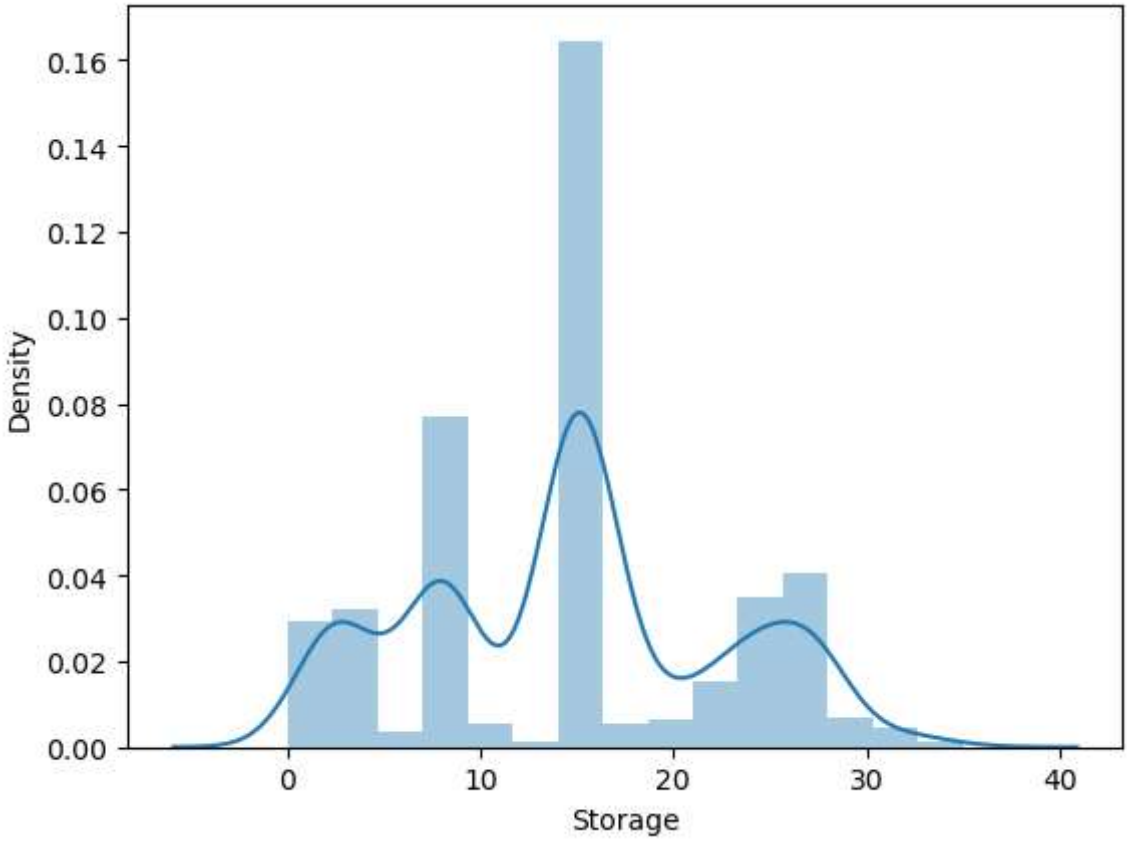
CPU
-0.8706961076531857



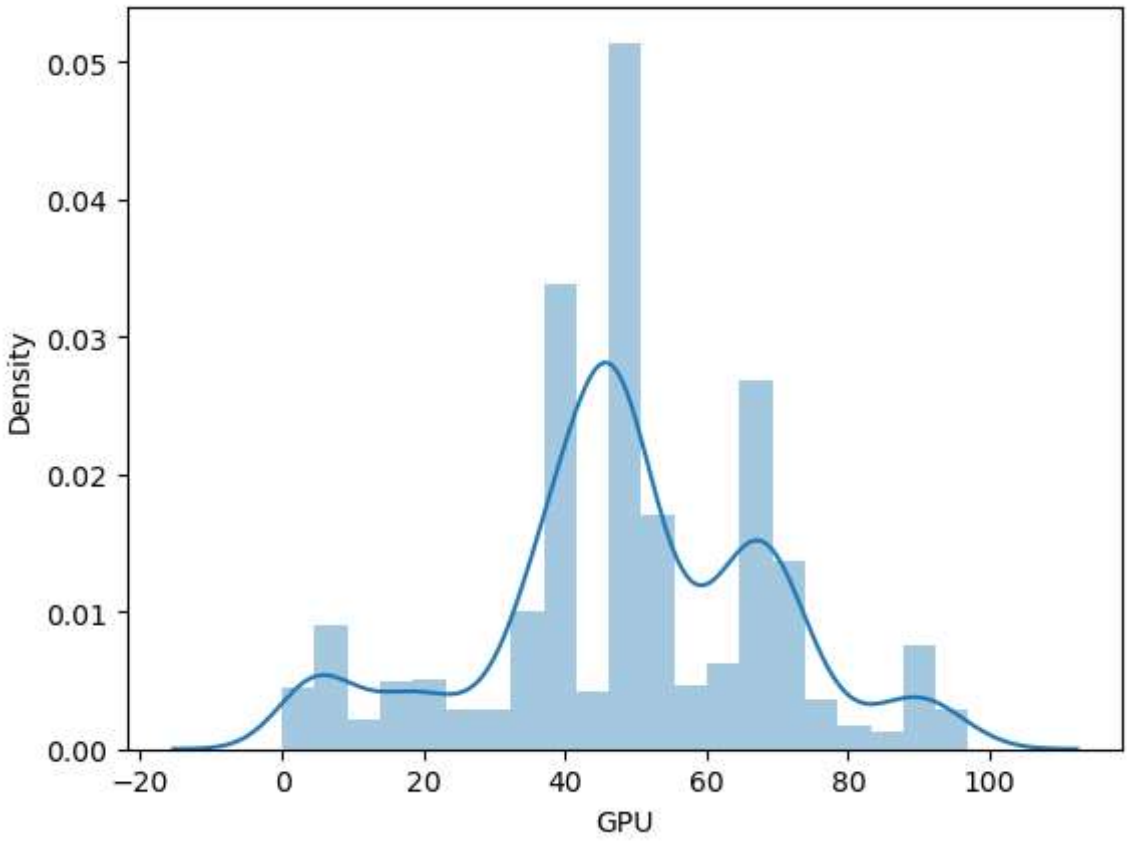
RAM
2.075028643064393



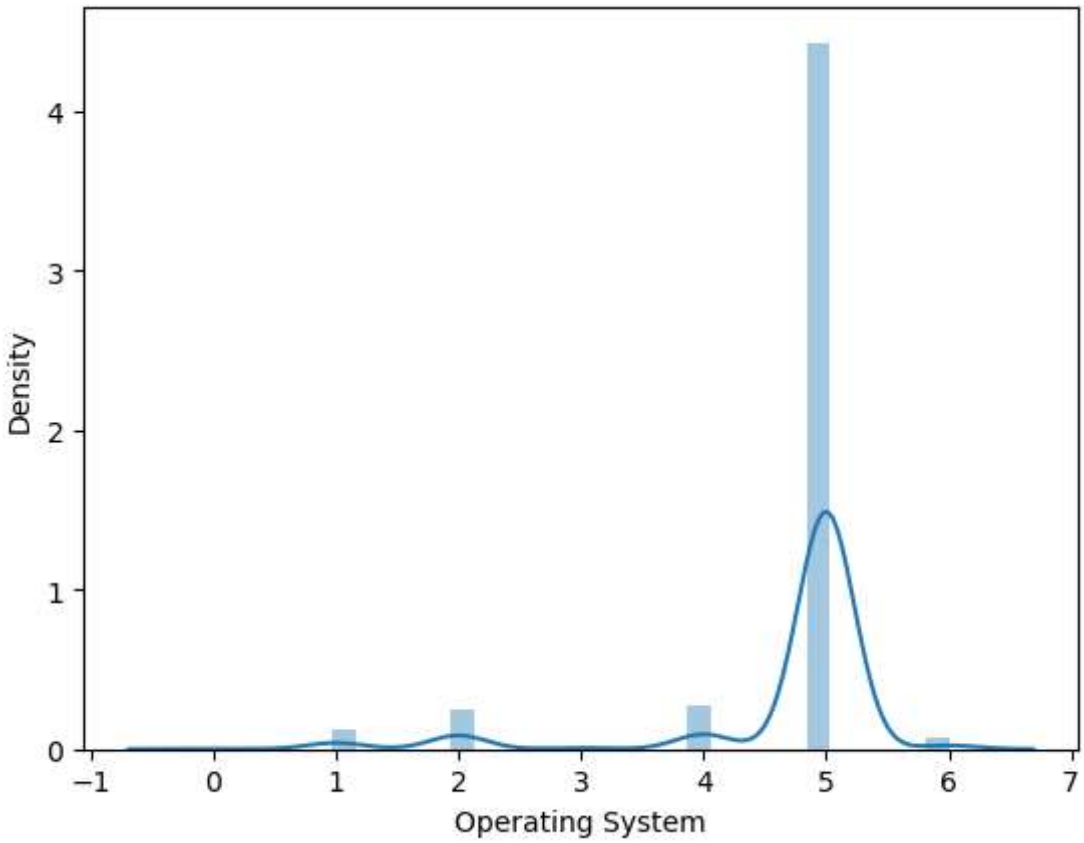

```
Storage
0.16586773790568834
```



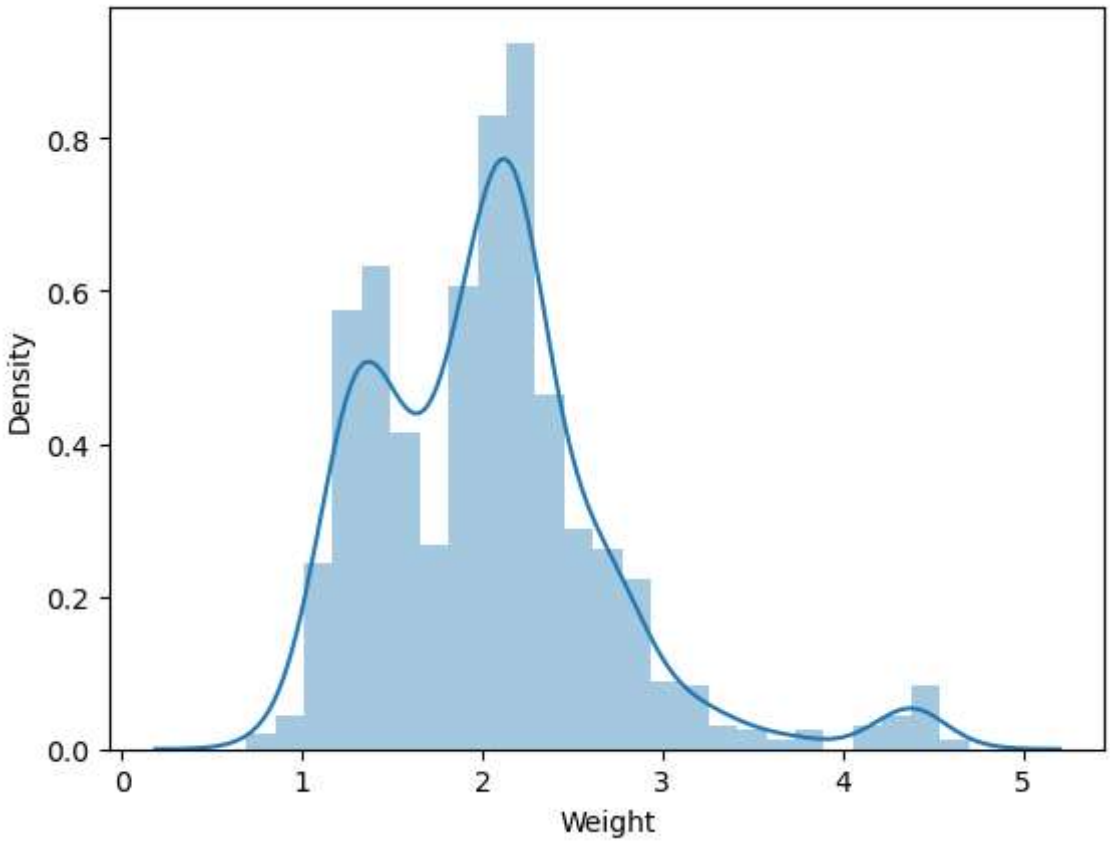

```
GPU
-0.19314119056515489
```




```
Operating System
-2.958763721948362
```




```
Weight
1.1831011510738085
```



To calculate Correlation

In [26]:
```
pd.concat([x,y],axis=1).corr().style.background_gradient()
```

Out[26]:

	Manufacturer	Model Name	Category	Screen Size	Screen	CPU	RAM	Storag
Manufacturer	1.000000	0.072437	0.018076	-0.079306	0.078551	0.026120	0.045832	0.08975
Model Name	0.072437	1.000000	0.100517	-0.226888	0.136383	0.172541	0.010054	0.06559
Category	0.018076	0.100517	1.000000	-0.059133	-0.143071	-0.097896	-0.221729	0.02619
Screen Size	-0.079306	-0.226888	-0.059133	1.000000	-0.252059	0.136170	0.232315	-0.18111
Screen	0.078551	0.136383	-0.143071	-0.252059	1.000000	0.230355	0.209217	0.09915
CPU	0.026120	0.172541	-0.097896	0.136170	0.230355	1.000000	0.477799	0.03893
RAM	0.045832	0.010054	-0.221729	0.232315	0.209217	0.477799	1.000000	0.21135
Storage	0.089756	0.065597	0.026195	-0.181117	0.099151	0.038931	0.211359	1.00000
GPU	0.021641	0.101139	-0.156478	0.190842	0.143795	0.477749	0.375900	0.00329
Operating System	0.140874	0.133870	0.027243	0.085771	0.070838	0.163975	0.165447	0.05636
Weight	-0.128727	-0.273516	-0.262282	0.822261	-0.135066	0.178046	0.390161	-0.15525
Price	0.133153	0.140762	-0.119578	0.075152	0.303763	0.533643	0.764005	0.25070

Apply Standard Scaler to Scale the data at one level

In [29]:
```
from sklearn.preprocessing import StandardScaler
```

Standardization is the process of rescaling the features so that they have the properties of a standard normal distribution with a mean of 0 and a standard deviation of 1

In [30]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

train_test_split function from scikit-learn to split your dataset into training and testing sets. This is a common in machine learning to evaluate the performance of a model.

In that testing data is 20% and random state is 1.

In [31]:
```
sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
xtest=sc.fit_transform(xtest)
```

We make object of standard scaler. In that we have to pass xtrain and xtest for transform into standard scaler.

In [32]:
```
xtrain
```

Out[32]:
```
array([[-1.07903279, -0.22805363,  0.31227921, ...,  1.41651338,
         0.31524252,  1.0089276 ],
       [-0.59035514, -1.45062335, -1.2975562 , ...,  1.12643858,
         0.31524252,  3.68665268],
       [ 1.12001664, -0.71708152, -1.2975562 , ...,  0.30455997,
         0.31524252,  3.65569632],
       ...,
       [ 0.87567782, -0.47997709,  0.31227921, ...,  1.56155078,
         0.31524252,  0.86962398],
       [-0.59035514, -0.34660584,  0.31227921, ..., -1.14581404,
        -3.08850506,  0.49814767],
       [-0.59035514, -0.30955828,  0.31227921, ..., -2.11273005,
         0.31524252,  1.17918758]])
```

This is our standardize xtrain data on which we performed Standard Scaling

In [33]:
```
xtest
```

Out[33]:
```
array([[ 0.16368452, -1.62605867,  0.29312381, ..., -0.74450367,
        -0.65139055, -0.24773833],
       [ 1.04870976, -0.96346511, -1.37215928, ...,  1.02848961,
         0.33831912,  0.88954803],
       [ 0.82745345,  0.75630017,  0.29312381, ..., -0.42214126,
         0.33831912, -0.03111235],
       ...,
       [-0.27882811, -0.17430875,  0.29312381, ..., -0.09977884,
         0.33831912,  0.21259187],
       [-0.94259704,  1.61246038, -2.20480083, ..., -0.09977884,
         0.33831912, -1.0465466 ],
       [-0.50008442,  1.47100782,  0.29312381, ...,  0.97476254,
         0.33831912,  0.02304414]])
```

This is our standardize xtest data on which we performed Standard Scaling

Train_Test_Split for separating data into training and testing phase

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

train_test_split function from scikit-learn to split your dataset into training and testing sets. This is a common in machine learning to evaluate the performance of a model.

In that testing data is 20% and random state is 1.

Build a Model by using LinearRegression Algoritham

In [36]:
```
from sklearn.linear_model import LinearRegression
```

In [37]:
```
lr=LinearRegression()
lr.fit(xtrain,ytrain)
ypred=lr.predict(xtest)
```

We are using the LinearRegression class from scikit-learn to fit a linear regression model in our training data (xtrain, ytrain) and make predictions on our test data (xtest).

Evaluate a model

In [38]:
```
from sklearn.metrics import r2_score
```

In [39]:
```
r2_score(ytest,ypred)
```

Out[39]:
```
0.7184981491612041
```

The r2_score function compares the predicted values (ypred) with the true values (ytest) and computes the R-squared score.

By applying Polynominal Linear Regression

In [40]:
```
from sklearn.preprocessing import PolynomialFeatures
pf=PolynomialFeatures()
polyx=pf.fit_transform(xtrain)

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(polyx,ytrain)
ypred_train=lr.predict(polyx)
```

In [41]:
```
from sklearn.metrics import r2_score
r2_score(ytrain,ypred_train)
```

Out[41]:
```
0.802676214175564
```

By using Polunominal Linear Regression we have 80% accuracy on training data.

In [42]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(ytrain,ypred_train)
```

Out[42]:
```
1964792.059993598
```

In [43]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(ytrain,ypred_train)
```

Out[43]:
```
7465365288343.721
```

In [44]:
```
np.sqrt(mean_squared_error(ytrain,ypred_train))
```

Out[44]:
```
2732282.066029004
```

Here we perform lost functions on training data for reduce error rate.

In [45]:
```
from sklearn.preprocessing import PolynomialFeatures
pf=PolynomialFeatures()
polyx=pf.fit_transform(xtest)

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(polyx,ytest)
ypred_test=lr.predict(polyx)
```

In [46]:
```
from sklearn.metrics import r2_score
r2_score(ytest,ypred_test)
```

Out[46]:
```
0.733590578229725
```

By using Polunominal Linear Regression we have 73% accuracy on testing data.

In [47]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(ytest,ypred_test)
```

Out[47]:
```
2727101.4497959185
```

In [48]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(ytest,ypred_test)
```

Out[48]:
```
12490377168834.443
```

In [49]:
```
np.sqrt(mean_squared_error(ytest,ypred_test))
```

Out[49]:
```
3534172.7700884184
```

Here we perform lost functions on testing data for reduce error rate.

Support Vector Regression

In [56]:
```
from sklearn.svm import SVR

# Create an SVR model
svr = SVR(kernel='rbf', C=1.0, epsilon=0.1)

# Train the model
svr.fit(xtrain, ytrain)

# Make predictions
ypred_test = svr.predict(xtest)
```

In [57]:
```
r2_score(ytest,ypred)
```

Out[57]:
```
0.7184981491612041
```

Here we evaluate the performance of our SVR model using appropriate regression metrics such as R2_score.