

Here we are importing two essential libraries NumPy and Pandas as alias np and pd respectively.

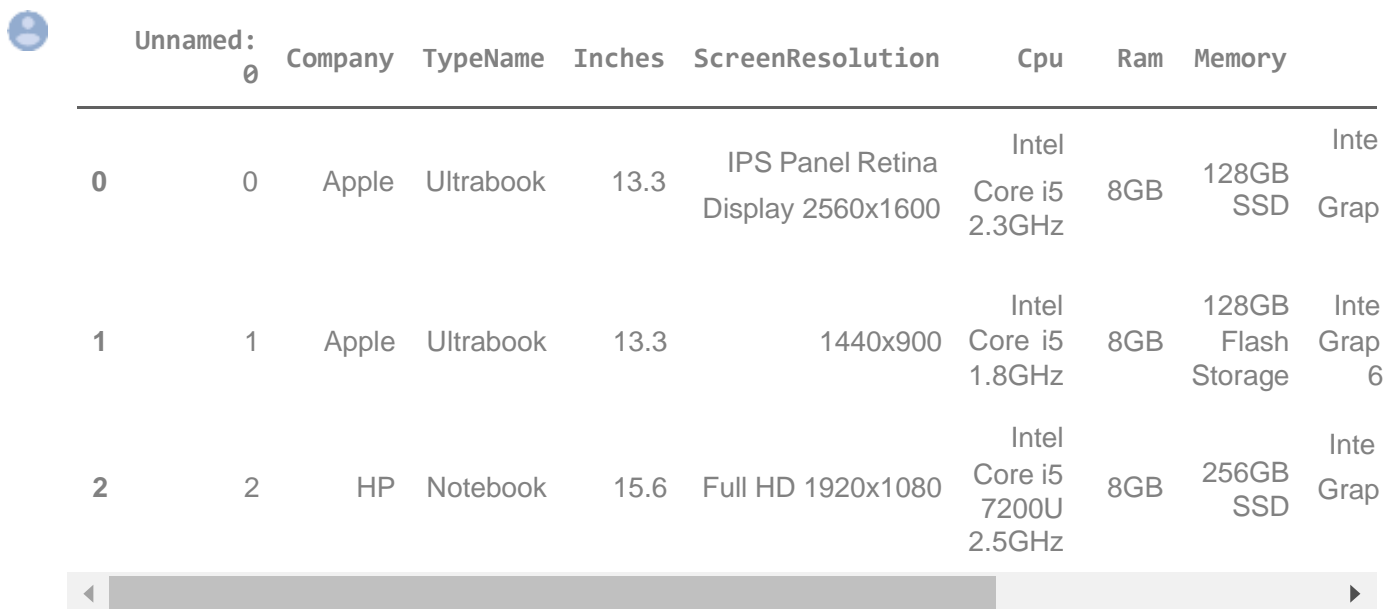
```
import numpy as np
import pandas as pd
import seaborn as sns
```

Next, we import the dataset using the Pandas' read_csv method, because our data is in the form of 'comma separated values' file where ',' acts as a delimiter.

We store the dataset in the form of DataFrame object in the df variable

```
df = pd.read_csv("C:/Users/hp India/Desktop/project_1/laptop_data.csv")
```

```
df.head()
```



	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Graphics
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel Graphics 6
2	2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel Graphics

Using head() method, we display the first 5 tuples of the dataset to get an overview of how the data looks like in the form of DataFrame object.

We have attributes like:

1. Unnamed
2. Company
3. TypeName
4. Inches
5. Screen Resolution
6. CPU
7. Ram
8. Memory

9. GPU
10. OS
11. Weight
12. Price

```
df.shape
```

```
(1303, 12)
```

We come to know that we have 1300 individual laptops with 12 features each in our dataset. Note that `.shape` is not a method, but an attribute of the dataframe object

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Unnamed: 0            1303 non-null   int64
1   Company               1303 non-null   object
2   TypeName              1303 non-null   object
3   Inches               1303 non-null   float64
4   ScreenResolution      1303 non-null   object
5   Cpu                  1303 non-null   object
6   Ram                  1303 non-null   object
7   Memory               1303 non-null   object
8   Gpu                  1303 non-null   object
9   OpSys                1303 non-null   object
10  Weight               1303 non-null   object
11  Price                1303 non-null   float64
dtypes: float64(2), int64(1), object(9)
memory usage: 122.3+ KB
```

Here, we understand what the datatype of our elements in the DataFrame are.

To perform Numerical computations, we need our elements to be of type integer or float.

Hence we witness the need of preprocessing the data herein.

```
df.duplicated().sum()
```

```
0
```

Here, we called the `duplicated` method to return a DataFrame object containing boolean values of whether it is a duplicate row or not.

Then we called the `.sum()` method to give us the total number of duplicated tuples which is zero in this case.

```
df.isnull().sum()
```

```

Unnamed: 0      0
Company         0
TypeName        0
Inches         0
ScreenResolution 0
Cpu            0
Ram            0
Memory         0
Gpu            0
OpSys          0
Weight         0
Price          0
dtype: int64

```

The good thing about this dataset is that there are no null values in any of the features, which we confirmed using the `isnull().sum()` methods respectively

▼ Pre-processing

If we carefully observe the features, the `Unnamed:0` column is out of relevance for our purpose, hence it would be a good decision to drop the column inplace.

```
df.drop(columns = ['Unnamed: 0'], inplace = True)
```

```
df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpS
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macO
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macO

Here, called the `df.drop` method, and passed our desired column to be dropped as an argument.

We chose inplace = True because we need to make permanent changes to this dataset itself

Now, the RAM attribute is essential to make predictions for the laptop price, but we cannot perform numerical computations on that column or to be more precise on that Series object because there is the string 'GB' associated with it.

Hence we will strip each element in the column and convert them into int data type. Here's how:

```
df['Ram'] = df['Ram'].str.replace('GB', '')
df['Ram'] = df['Ram'].astype('int')
```

As we needed to strip 'GB', we called .str on our column because its used to access the string accessor method of the 'Ram' column.

The .str allows us to apply string methods, such as .replace(), to the elements in a pandas Series or DataFrame column.

Following, we replace the 'GB' with an empty string and get the following as the result:

```
df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS

Similarly, weight is also an essential attribute for us but to perform numerical operations we need it to be of the type 'float'. Hence we apply the same process on that column

```
df['Weight'] = df['Weight'].str.replace('kg', '')
df['Weight'] = df['Weight'].astype('float')
```

```
df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 11 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   Company             1303 non-null   object
1   TypeName             1303 non-null   object
2   Inches              1303 non-null   float64
3   ScreenResolution    1303 non-null   object
4   Cpu                 1303 non-null   object
5   Ram                 1303 non-null   int32
6   Memory              1303 non-null   object
7   Gpu                 1303 non-null   object
8   OpSys               1303 non-null   object
9   Weight              1303 non-null   float64
10  Price               1303 non-null   float64
dtypes: float64(3), int32(1), object(7)
memory usage: 107.0+ KB
```

Here we can witness that Ram is int32, Weight is float32.

▼ Feature Engineering (Pre-processing)

Now, our aim is to feed the model best version of numerically consistent data so that it can make most accurate predictions.

So even if a feature is crucial is important in determining the price, we should be able to perform operations on it. But the ScreenResolution column is very distinct and difficult to compute.

Hence we will need to pre-process the data

```
df.ScreenResolution.value_counts()
```

Full HD 1920x1080	507
1366x768	281
IPS Panel Full HD 1920x1080	230
IPS Panel Full HD / Touchscreen 1920x1080	53
Full HD / Touchscreen 1920x1080	47
1600x900	23
Touchscreen 1366x768	16
Quad HD+ / Touchscreen 3200x1800	15
IPS Panel 4K Ultra HD 3840x2160	12
IPS Panel 4K Ultra HD / Touchscreen 3840x2160	11
4K Ultra HD / Touchscreen 3840x2160	10
4K Ultra HD 3840x2160	7
Touchscreen 2560x1440	7
IPS Panel 1366x768	7
IPS Panel Quad HD+ / Touchscreen 3200x1800	6
IPS Panel Retina Display 2560x1600	6
IPS Panel Retina Display 2304x1440	6
Touchscreen 2256x1504	6
IPS Panel Touchscreen 2560x1440	5
IPS Panel Retina Display 2880x1800	4
IPS Panel Touchscreen 1920x1200	4
1440x900	4
IPS Panel 2560x1440	4
IPS Panel Quad HD+ 2560x1440	3
Quad HD+ 3200x1800	3
1920x1080	3
Touchscreen 2400x1600	3
2560x1440	3
IPS Panel Touchscreen 1366x768	3
IPS Panel Touchscreen / 4K Ultra HD 3840x2160	2
IPS Panel Full HD 2160x1440	2
IPS Panel Quad HD+ 3200x1800	2
IPS Panel Retina Display 2736x1824	1
IPS Panel Full HD 1920x1200	1
IPS Panel Full HD 2560x1440	1
IPS Panel Full HD 1366x768	1
Touchscreen / Full HD 1920x1080	1
Touchscreen / Quad HD+ 3200x1800	1
Touchscreen / 4K Ultra HD 3840x2160	1
IPS Panel Touchscreen 2400x1600	1
Name: ScreenResolution, dtype: int64	

Here we can see that there are a lot of different types of values, with very much less common things between them.

But every value has **resolution in pixels** in it.

Then there are also some laptops which are **Touchscreen** or not which is specified in this column itself.

Also, we can find that maximum have **IPS panel** inside them.

```
df['Touchscreen'] = df['ScreenResolution'].apply(lambda x: 1 if 'Touchscreen' in x else 0)
df['Touchscreen']
```

```

0      0
1      0
2      0
3      0
4      0
..
1298   1
1299   1
1300   0
1301   0
1302   0
Name: Touchscreen, Length: 1303, dtype: int64

```

Now, a laptop being touchscreen or not significantly determines the price, and that information is present inclusively in the ScreenResolution column. Hence we need to fetch it out.

Therefore we created a Touchscreen column where 1 indicates touchscreen else 0.

Implementation: We applied an anonymous function lambda on each element of screenresolution and if it had the string 'Touchscreen' inside it we stored 0 in our new column else 1

We will confirm its significance in Data Visualisation part of EDA

```

df['IPS'] = df['ScreenResolution'].apply(lambda x: 1 if 'IPS' in x else 0)
df['IPS']

```

```

0      1
1      0
2      0
3      1
4      1
..
1298   1
1299   1
1300   0
1301   0
1302   0
Name: IPS, Length: 1303, dtype: int64

```

We did the same thing with IPS display panels. Created a separate column for so. Further in EDA we have also visualised how IPS affect price and what is more expensive IPS or Touchscreen

Now we intuitively know that screen resolution plays an important part in predicting the price, specifically the 1080x1920 part. Hence to analyse them, we would have to extract them from the ScreenResolution column and create two separate columns for so.

```

new = df['ScreenResolution'].str.split('x',n=1,expand=True)
new

```

		0	1
0	IPS Panel Retina Display	2560	1600
1		1440	900
2	Full HD	1920	1080
3	IPS Panel Retina Display	2880	1800
4	IPS Panel Retina Display	2560	1600
...	
1298	IPS Panel Full HD / Touchscreen	1920	1080
1299	IPS Panel Quad HD+ / Touchscreen	3200	1800
1300		1366	768
1301		1366	768
1302		1366	768

1303 rows x 2 columns

Here, we splitted each screenresolution row on the basis of 'x' and `expand = True` specifies that the result of the split should be returned as a DataFrame with each substring occupying its own column. By default, `expand=False`, and the result is returned as a series of lists, where each element is a list of substrings resulting from the split.

```
df['X_resolution'] = new[0]
df['Y_resolution'] = new[1]
```

Here we created two different columns in the dataframe and stored the first column and second of the new dataframe in the original respectively

```
df['X_resolution'] = df['X_resolution'].str.replace(',', '').str.findall(r'(\d+\.\d+)').ap
```

Now, I have a column in which each row has values like IPS LCD 1080, so my work is to extract the numerical part from the element.

So first I replace all the commas by empty strings, so that I can carry out my processes smoothly, then I call `str.findall` which will return me a list of strings satisfying my condition.

I pass a regular expression (regex) that will give me a decimal or integer number occurrences, and at last I apply lambda function where I only get the first occurrence of the Integer

```
df.head()
```


	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS
3	Apple	Ultrabook	15.4	IPS Panel Retina	Intel Core i7	16	512GB	AMD Radeon	macOS

```
df['X_resolution'] = df['X_resolution'].astype('int')
df['Y_resolution'] = df['Y_resolution'].astype('int')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Company                1303 non-null  object
1   TypeName               1303 non-null  object
2   Inches                 1303 non-null  float64
3   ScreenResolution       1303 non-null  object
4   Cpu                    1303 non-null  object
5   Ram                    1303 non-null  int32
6   Memory                 1303 non-null  object
7   Gpu                    1303 non-null  object
8   OpSys                  1303 non-null  object
9   Weight                 1303 non-null  float64
10  Price                  1303 non-null  float64
11  Touchscreen            1303 non-null  int64
12  IPS                     1303 non-null  int64
13  X_resolution           1303 non-null  int32
14  Y_resolution           1303 non-null  int32
dtypes: float64(3), int32(3), int64(2), object(7)
memory usage: 137.6+ KB
```

Now there is no way to find how x and y resolution contribute to the price individually. Hence we are going to create a new column pixels per inch (PPI) which takes into consideration columns

like inches, x and y resolution

```
df['PPI'] = (((df['X_resolution']**2) + (df['Y_resolution']**2))**0.5/df['Inches']).astype
```

Now there is no way to find how x and y resolution contribute to the price individually. Hence we are going to create a new column pixels per inch (PPI) which takes into consideration columns like inches, x and y resolution

show that `df.corr()['Price']` is having high correlation for ppi

```
df.drop(columns = ['ScreenResolution', 'Inches'], inplace=True)
df.drop(columns = ['X_resolution', 'Y_resolution'], inplace=True)
```

We dropped the columns Inches, Screen Resolution, X_res, Y_res because we replaced it with PPI

```
df.head()
```

	Company	TypeName	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	Apple	Ultrabook	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832
1	Apple	Ultrabook	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232
			Intel						

Now our next focus is the CPU column.

```
df.Cpu.value_counts()
```

```
Intel Core i5 7200U 2.5GHz    190
Intel Core i7 7700HQ 2.8GHz    146
Intel Core i7 7500U 2.7GHz    134
Intel Core i7 8550U 1.8GHz     73
Intel Core i5 8250U 1.6GHz     72
...
Intel Core M M3-6Y30 0.9GHz     1
AMD A9-Series 9420 2.9GHz       1
Intel Core i3 6006U 2.2GHz       1
AMD A6-Series 7310 2GHz         1
Intel Xeon E3-1535M v6 3.1GHz     1
Name: Cpu, Length: 118, dtype: int64
```

We can see that there are 188 different types of CPU, so we will group the common ones together and exclude rest in 'others' group

```
df['CPU Name'] = df.Cpu.apply(lambda x: " ".join(x.split()[0:3]))
```

Here as I am only interested in first three words, I split and selected only first three through indexing, then I converted into a string by .join

```
df.head()
```

	Company	TypeName	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touc
0	Apple	Ultrabook	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	
1	Apple	Ultrabook	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	

```
def NameProcessor(string):
    if string == 'Intel Core i5' or string == 'Intel Core i3' or string == 'Intel Core i7':
        return string
    else:
        if string.split()[0] == 'Intel':
            return 'Other Intel Processor'
        else:
            return 'AMD Processor'
```

Here we made a function where our input will be the new CPU Name column and if the String corresponds to i7, i5 or i3, then we will return the String itself. If it is just intel but not i, then it will come under other intel category, else it is an AMD Processor.

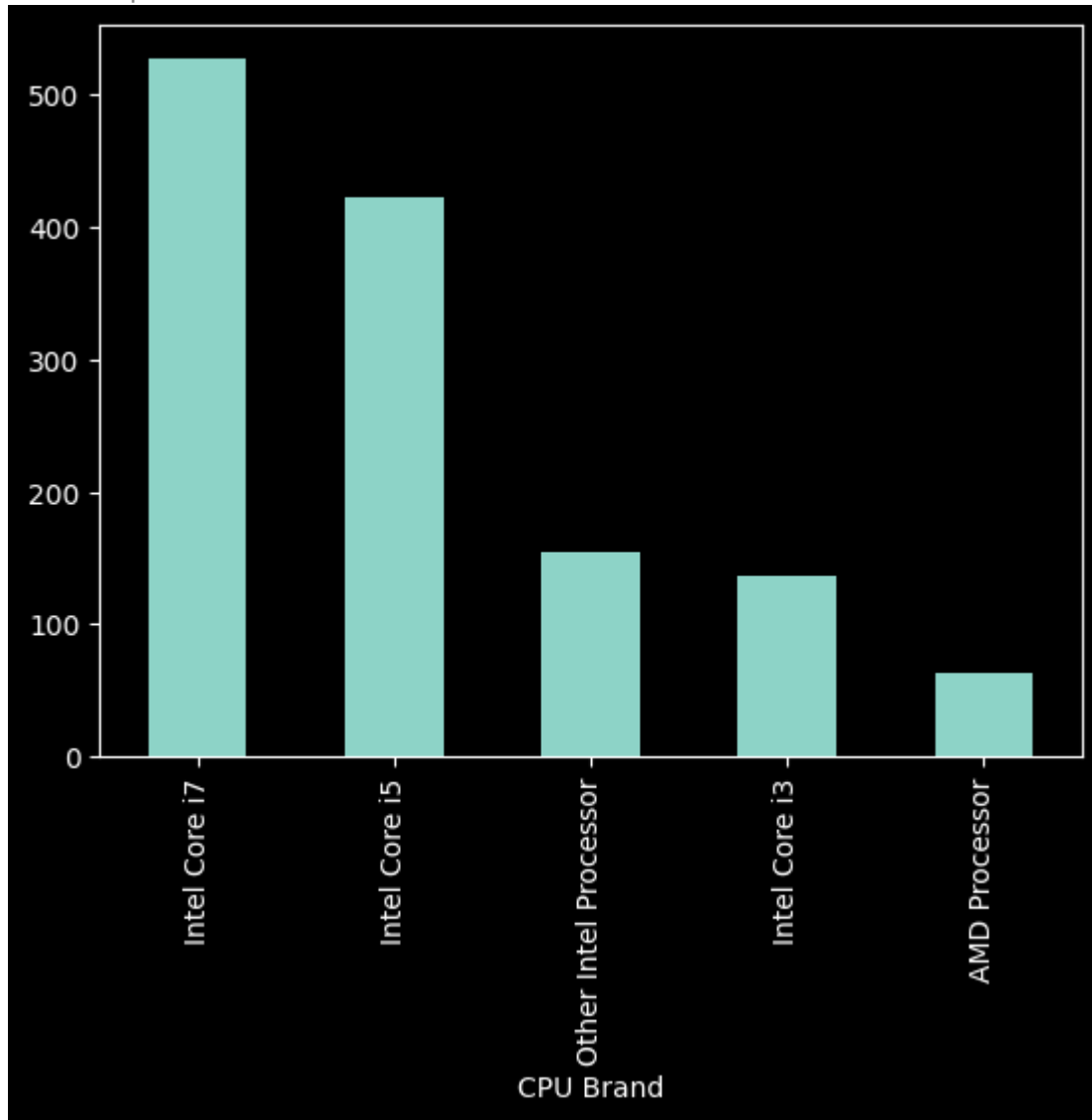
```
df['CPU Brand'] = df['CPU Name'].apply(NameProcessor)
```

```
df['CPU Brand'].value_counts()
```

```
Intel Core i7          527
Intel Core i5          423
Other Intel Processor   154
Intel Core i3          136
AMD Processor           63
Name: CPU Brand, dtype: int64
```

```
df['CPU Brand'].value_counts().plot(kind = 'bar')
```

<AxesSubplot: xlabel='CPU Brand'>



Now we can see that we have structured the CPU column We will see how price varies with CPU Brand type in EDA

Now our CPU and CPU Name columns are useless, hence we will drop them

```
df.drop(columns = ['Cpu', 'CPU Name'], inplace = True)
```

```
df.head()
```

	Company	TypeName	Ram	Memory	Gpu	OpSys	Weight	Price	Touchsc
0	Apple	Ultrabook	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	
1	Apple	Ultrabook	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	

Now we will tackle with the Memory Column

```
df.Memory.value_counts()
```

256GB SSD	412
1TB HDD	223
500GB HDD	132
512GB SSD	118
128GB SSD + 1TB HDD	94
128GB SSD	76
256GB SSD + 1TB HDD	73
32GB Flash Storage	38
2TB HDD	16
64GB Flash Storage	15
512GB SSD + 1TB HDD	14
1TB SSD	14
256GB SSD + 2TB HDD	10
1.0TB Hybrid	9
256GB Flash Storage	8
16GB Flash Storage	7
32GB SSD	6
180GB SSD	5
128GB Flash Storage	4
512GB SSD + 2TB HDD	3
16GB SSD	3
512GB Flash Storage	2
1TB SSD + 1TB HDD	2
256GB SSD + 500GB HDD	2
128GB SSD + 2TB HDD	2
256GB SSD + 256GB SSD	2
512GB SSD + 256GB SSD	1
512GB SSD + 512GB SSD	1
64GB Flash Storage + 1TB HDD	1
1TB HDD + 1TB HDD	1
32GB HDD	1
64GB SSD	1
128GB HDD	1
240GB SSD	1
8GB SSD	1
508GB Hybrid	1
1.0TB HDD	1
512GB SSD + 1.0TB Hybrid	1
256GB SSD + 1.0TB Hybrid	1

Name: Memory, dtype: int64

We can see that we have a lot of different types of values. But the main divisions are SSD, HDD, Flash Storage and Hybrid. So we have to apply pre-processing here, but if we carefully see that one of them has **1.0TB** which is very tedious for further operations, hence we will replace that with an empty string

In this case, the value to be replaced is `'1.0'`, which is a regular expression that matches a decimal point followed by a zero. The value to replace it with is an empty string, `''`. The `regex=True` argument indicates that we are using a regular expression to perform the replacement.

```
df.Memory = df['Memory'].astype(str).replace('\0.', "", regex = True)
```

Below we just replaced GB with an empty string, and correspondingly TB with 000 (TB = 1000x GB)

```
df.Memory = df['Memory'].str.replace('GB', "")
df.Memory = df.Memory.str.replace('TB', '000')
```

```
df.sample(5)
```

	Company	TypeName	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscre
733	Acer	Notebook	4	500 HDD	Intel HD Graphics 620	Windows 10	2.40	29783.52	
1262	Dell	Notebook	6	1000 HDD	Intel HD Graphics 620	Windows 10	2.30	24455.52	
					AMD				

Now we have a plus sign in between therefore we will split the column into two, on the basis of plus sign

```
df['Memory'] = df['Memory'].astype(str).replace('\.0', '', regex=True)
df["Memory"] = df["Memory"].str.replace('GB', '')
df["Memory"] = df["Memory"].str.replace('TB', '000')
new = df["Memory"].str.split("+", n = 1, expand = True)

df["first"] = new[0]
df["first"] = df["first"].str.strip()

df["second"] = new[1]

df["Layer1HDD"] = df["first"].apply(lambda x: 1 if "HDD" in x else 0)
df["Layer1SSD"] = df["first"].apply(lambda x: 1 if "SSD" in x else 0)
df["Layer1Hybrid"] = df["first"].apply(lambda x: 1 if "Hybrid" in x else 0)
df["Layer1Flash_Storage"] = df["first"].apply(lambda x: 1 if "Flash Storage" in x else 0)

df['first'] = df['first'].str.replace(r'\D', '')

df["second"].fillna("0", inplace = True)

df["Layer2HDD"] = df["second"].apply(lambda x: 1 if "HDD" in x else 0)
df["Layer2SSD"] = df["second"].apply(lambda x: 1 if "SSD" in x else 0)
df["Layer2Hybrid"] = df["second"].apply(lambda x: 1 if "Hybrid" in x else 0)
df["Layer2Flash_Storage"] = df["second"].apply(lambda x: 1 if "Flash Storage" in x else 0)

df['second'] = df['second'].str.replace(r'\D', '')
```

```

df["first"] = df["first"].astype(int)
df["second"] = df["second"].astype(int)

df["HDD"]=(df["first"]*df["Layer1HDD"]+df["second"]*df["Layer2HDD"])
df["SSD"]=(df["first"]*df["Layer1SSD"]+df["second"]*df["Layer2SSD"])
df["Hybrid"]=(df["first"]*df["Layer1Hybrid"]+df["second"]*df["Layer2Hybrid"])
df["Flash_Storage"]=(df["first"]*df["Layer1Flash_Storage"]+df["second"]*df["Layer2Flash_St

df.drop(columns=['first', 'second', 'Layer1HDD', 'Layer1SSD', 'Layer1Hybrid',
                'Layer1Flash_Storage', 'Layer2HDD', 'Layer2SSD', 'Layer2Hybrid',
                'Layer2Flash_Storage'],inplace=True)

C:\Users\hp India\AppData\Local\Temp\ipykernel_2724\2716506507.py:16: FutureWarning:
  df['first'] = df['first'].str.replace(r'\D', '')
C:\Users\hp India\AppData\Local\Temp\ipykernel_2724\2716506507.py:25: FutureWarning:
  df['second'] = df['second'].str.replace(r'\D', '')

```

We have made multiple changes here necessary to separate memory column into 3 numerically operatable columns

```
df.sample(5)
```

	Company	TypeName	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscre
453	Lenovo	Notebook	8	256 SSD	Intel HD Graphics 620	Windows 10	1.65	71661.6000	
1016	HP	Notebook	4	256 SSD	Intel HD Graphics 620	Windows 10	2.04	41025.0672	
					Intel HD				

```
df.drop(columns=['Memory'],inplace=True)
```

```
df.head()
```

	Company	TypeName	Ram	Gpu	OpSys	Weight	Price	Touchscreen	IPS
0	Apple	Ultrabook	8	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1 22
1	Apple	Ultrabook	8	Intel HD Graphics	macOS	1.34	47895.5232	0	0 12

```
df.corr()['Price']
```

```
C:\Users\hp India\AppData\Local\Temp\ipykernel_2724\815546952.py:1: FutureWarning: T
df.corr()['Price']
Ram          0.743007
Weight       0.210370
Price        1.000000
Touchscreen  0.191226
IPS          0.252208
PPI          0.473487
HDD          -0.096441
SSD          0.670799
Hybrid       0.007989
Flash_Storage -0.040511
Name: Price, dtype: float64
```

as we can see Hybrid and Flash storage has very less correlation in predicting the price. Hence we will drop those columns as a part of feature engineering

```
df.drop(columns = ['Hybrid', 'Flash_Storage'], inplace = True)
```

```
df.head()
```

	Company	TypeName	Ram	Gpu	OpSys	Weight	Price	Touchscreen	IPS
0	Apple	Ultrabook	8	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1 22
1	Apple	Ultrabook	8	Intel HD Graphics	macOS	1.34	47895.5232	0	0 12

Now if we look closely, the gpu column has many categories of values. We will have to deal with it

```
df['Gpu'].value_counts()
```

```
Intel HD Graphics 620      281
Intel HD Graphics 520      185
Intel UHD Graphics 620      68
Nvidia GeForce GTX 1050     66
Nvidia GeForce GTX 1060     48
...
AMD Radeon R5 520           1
AMD Radeon R7               1
Intel HD Graphics 540        1
AMD Radeon 540              1
ARM Mali T860 MP4           1
Name: Gpu, Length: 110, dtype: int64
```



```
df['Gpu brand'] = df['Gpu'].apply(lambda x:x.split()[0])
```

```
df.head()
```

	Company	TypeName	Ram	Gpu	OpSys	Weight	Price	Touchscreen	IPS
0	Apple	Ultrabook	8	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1 22
1	Apple	Ultrabook	8	Intel HD Graphics	macOS	1.34	47895.5232	0	0 12

```
df['Gpu brand'].value_counts()
```

```
Intel      722
Nvidia     400
AMD        180
ARM         1
Name: Gpu brand, dtype: int64
```

```
df = df[df['Gpu brand'] != 'ARM']
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.barplot(x=df['Gpu brand'],y=df['Price'],estimator=np.median)
plt.xticks(rotation='vertical')
plt.show()
```

```

70000
60000
50000
df.drop(columns = ['Gpu'], inplace = True)
df['OpSys'].value_counts()

```

```

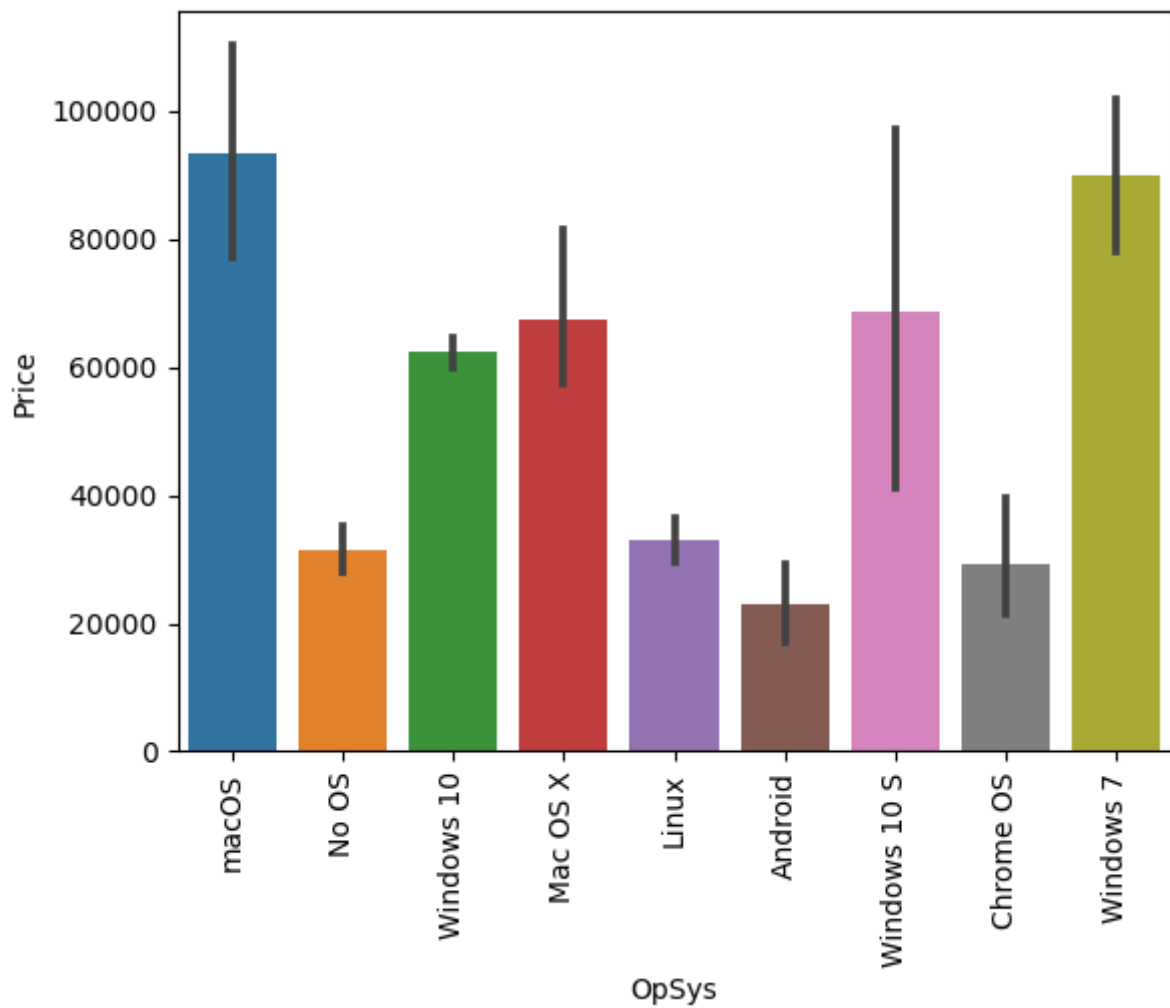
Windows 10      1072
No OS           66
Linux           62
Windows 7       45
Chrome OS       26
macOS           13
Mac OS X        8
Windows 10 S    8
Android         2
Name: OpSys, dtype: int64

```

```

sns.barplot(x=df['OpSys'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()

```



```
def cat_os(inp):  
    if inp == 'Windows 10' or inp == 'Windows 7' or inp == 'Windows 10 S':  
        return 'Windows'  
    elif inp == 'macOS' or inp == 'Mac OS X':  
        return 'Mac'  
    else:  
        return 'Others/No OS/Linux'
```

```
df['os'] = df['OpSys'].apply(cat_os)
```

```
df.drop(columns=['OpSys'], inplace=True)
```

▼ Exploratory Data Analysis

Double-click (or enter) to edit

```
import seaborn as sns  
import matplotlib.pyplot as plt
```

Now we will perform Univariate Analysis on our data.

histplot is a function in the seaborn library used to plot a univariate distribution of observations.

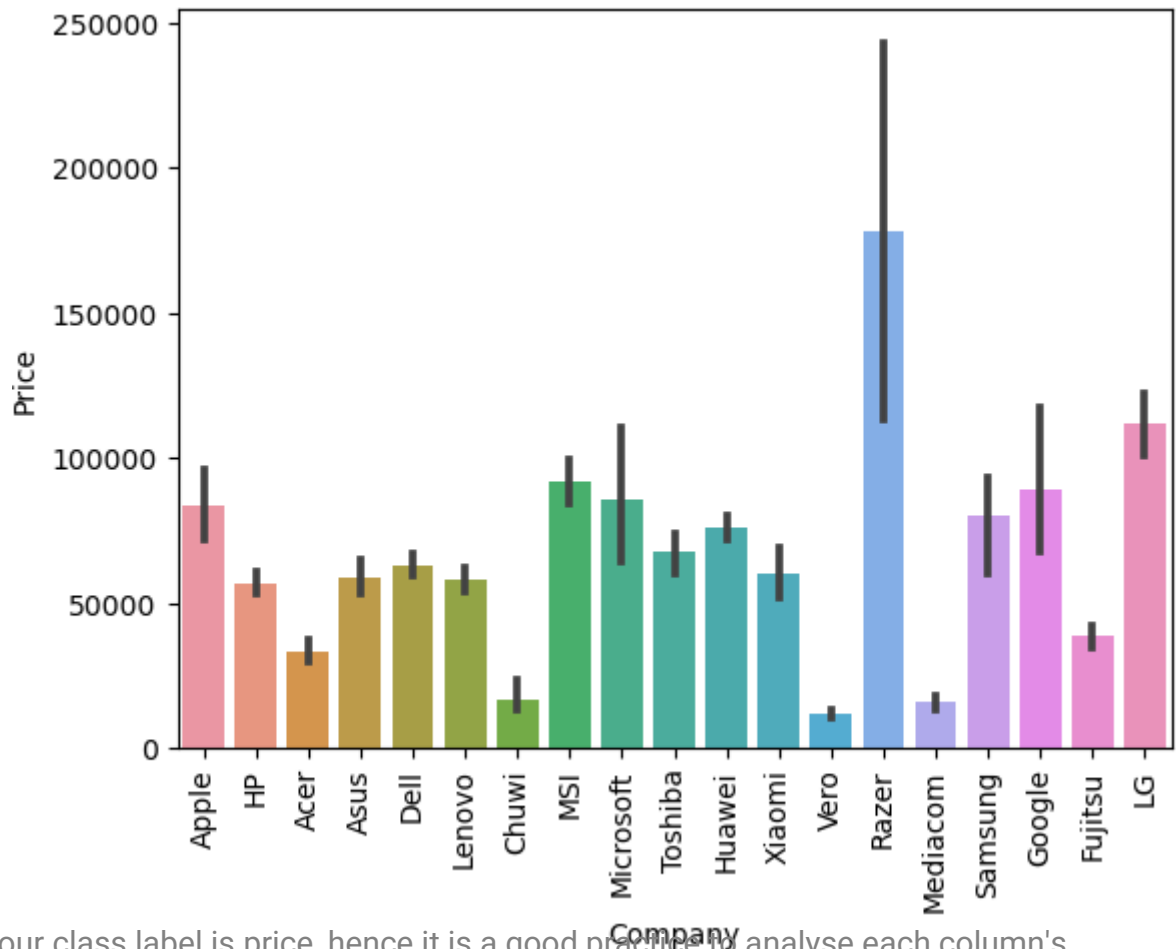
KDE is a way to estimate the probability distribution function of our continuous random variable (here price). Hence we superimposed it on our histogram to get a good idea.

```
sns.histplot(df.Price, kde =True, alpha = 0.5)
```



<Axes: >





Now, our class label is price, hence it is a good practice to analyse each column's significance/ability to help us predict us the class label, or is the attribute useful or relevant in predicting price?

To check this, we plotted the Companies v/s their price. Hence we can see that some companies are too much expensive while others are in the midrange segment, hence it is a useful feature.

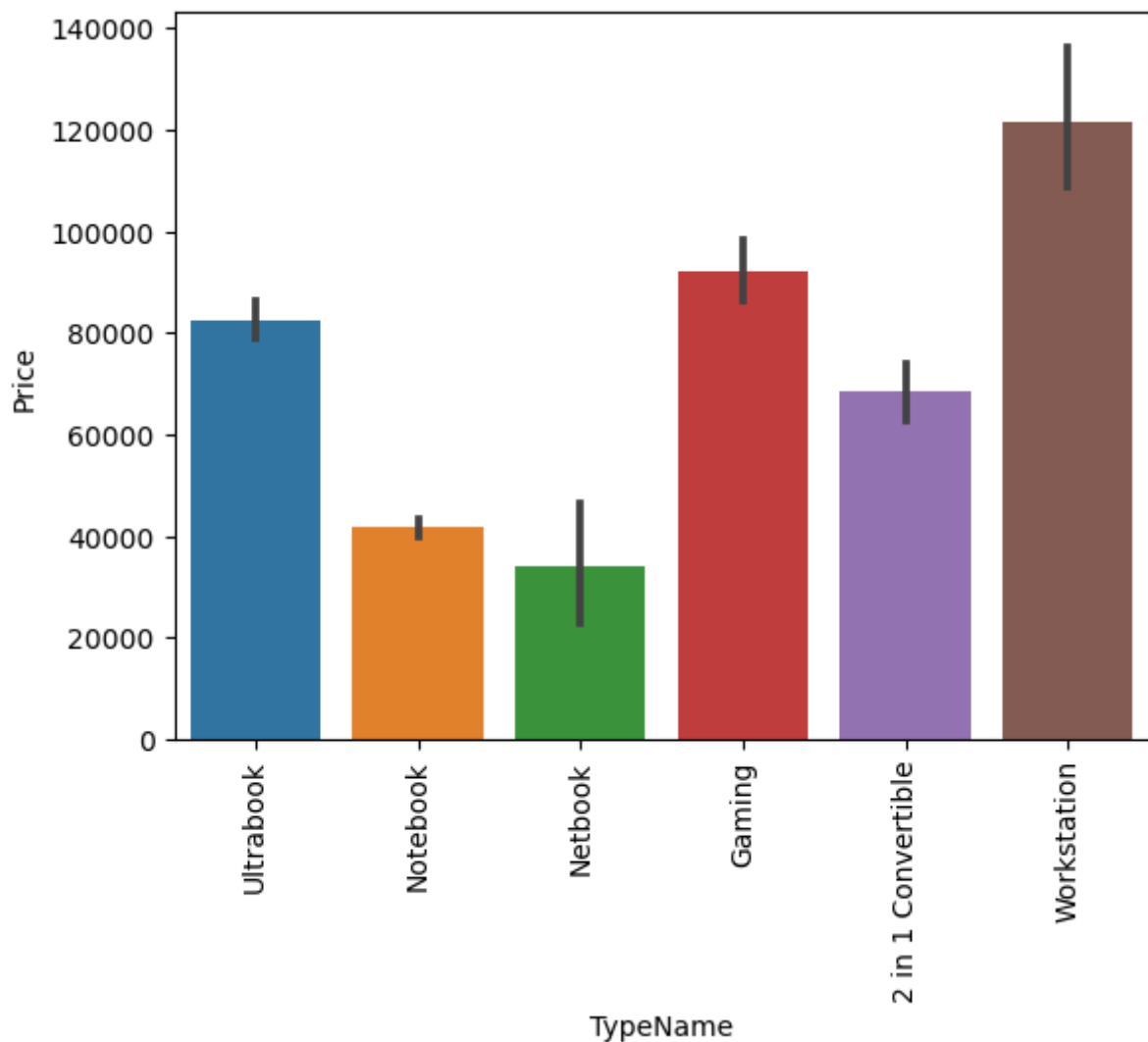
```
df['TypeName'].value_counts().plot(kind = 'bar')
```

<Axes: >



Similarly, we check the the occurancy of the type of Laptops. We witness that mostly there are Notebooks.

```
sns.barplot(x = df.TypeName, y = df.Price)
plt.xticks(rotation = 'vertical')
plt.show()
```



Here we can clearly notice that the type of the laptops have different prices alltogether hence they are going to help us determine our price

```
'''sns.barplot(x = df.Inches,y = df.Price)
plt.xticks(rotation = 'vertical')
plt.show()'''

"sns.barplot(x = df.Inches,y = df.Price)\nplt.xticks(rotation =
'vertical')\nplt.show()"
```

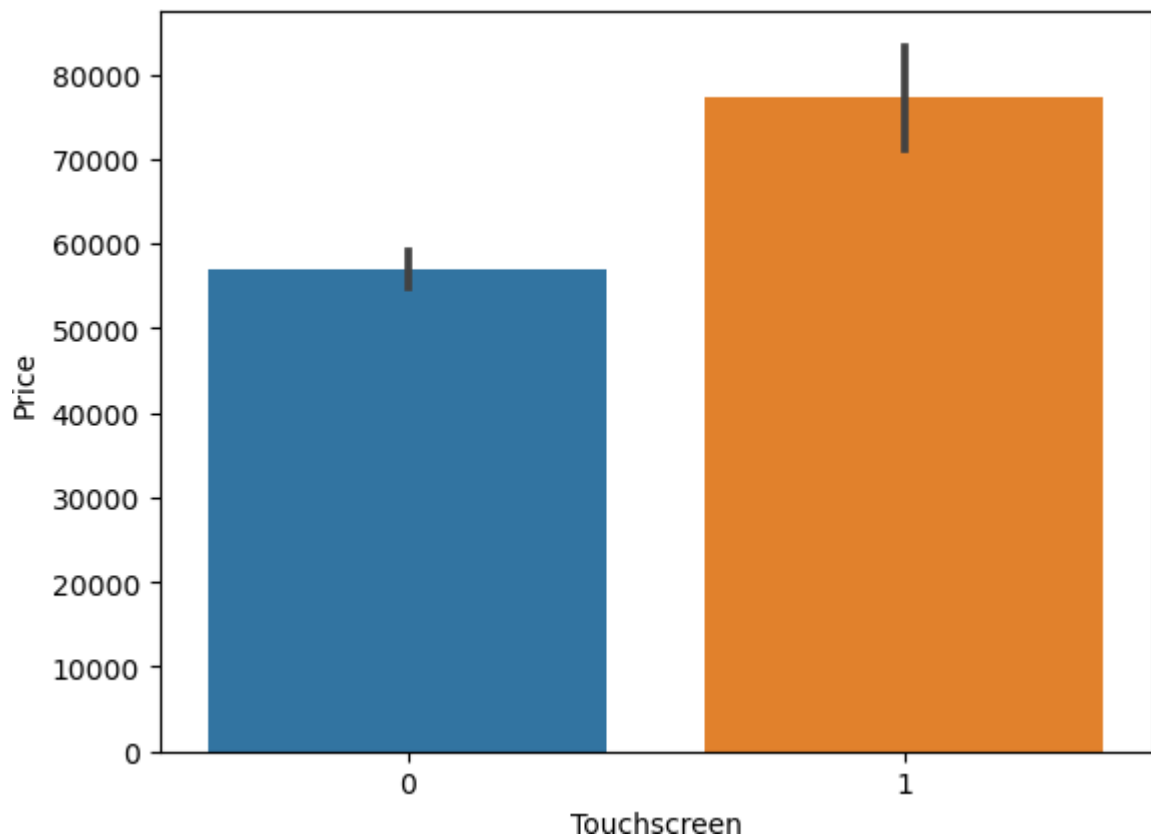
Here, more or less the price is same for different inches, but the difference is not insignificant. We will have to take this feature into consideration.

We made an assumption that touchscreen laptops differ from the price than normal, and we can see the proof of so.

Touchscreen laptops' mean price is higher significantly.

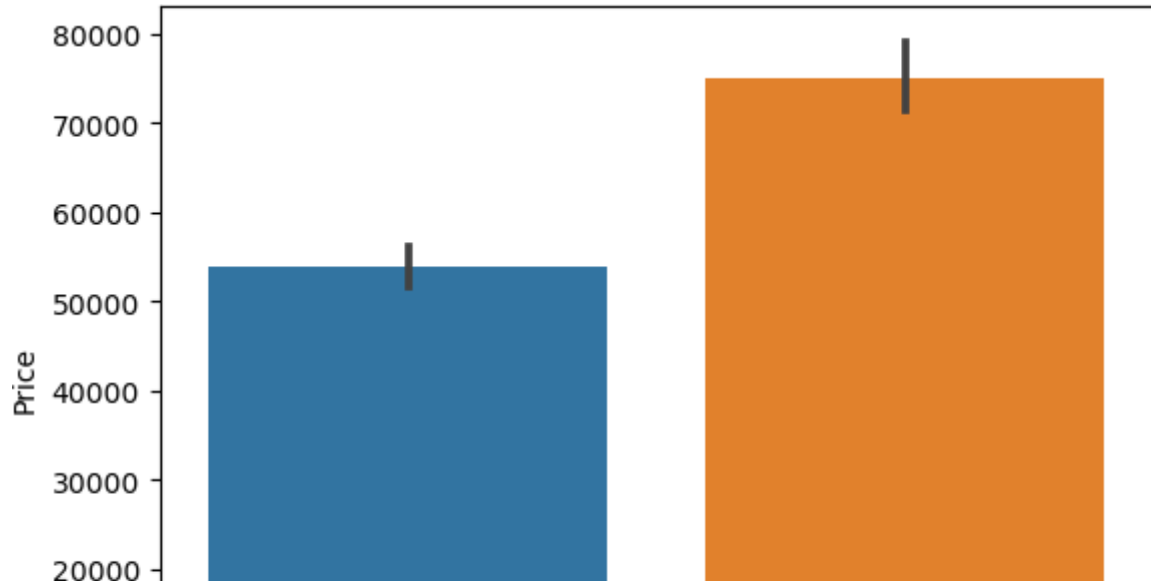
```
sns.barplot(x = df['Touchscreen'], y = df.Price)

<Axes: xlabel='Touchscreen', ylabel='Price'>
```

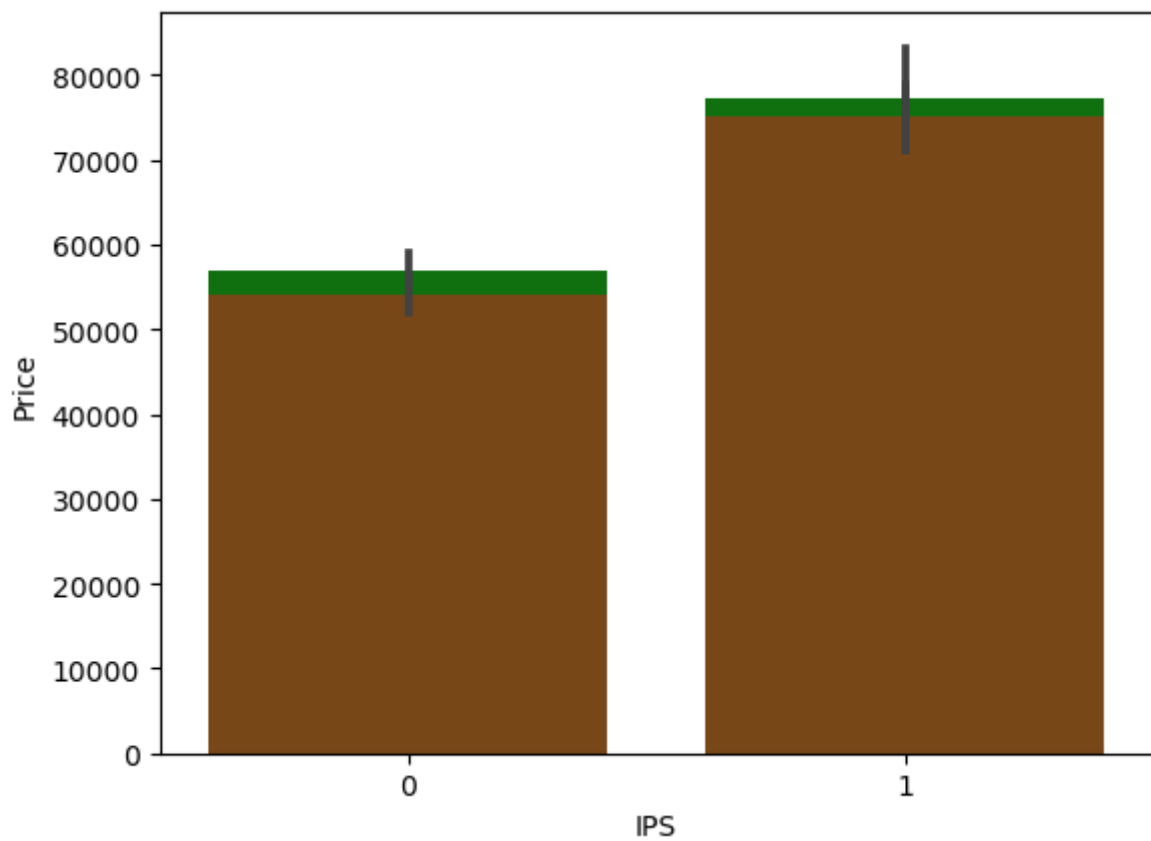


```
sns.barplot(x = df.IPS, y = df.Price)
```

<Axes: xlabel='IPS', ylabel='Price'>



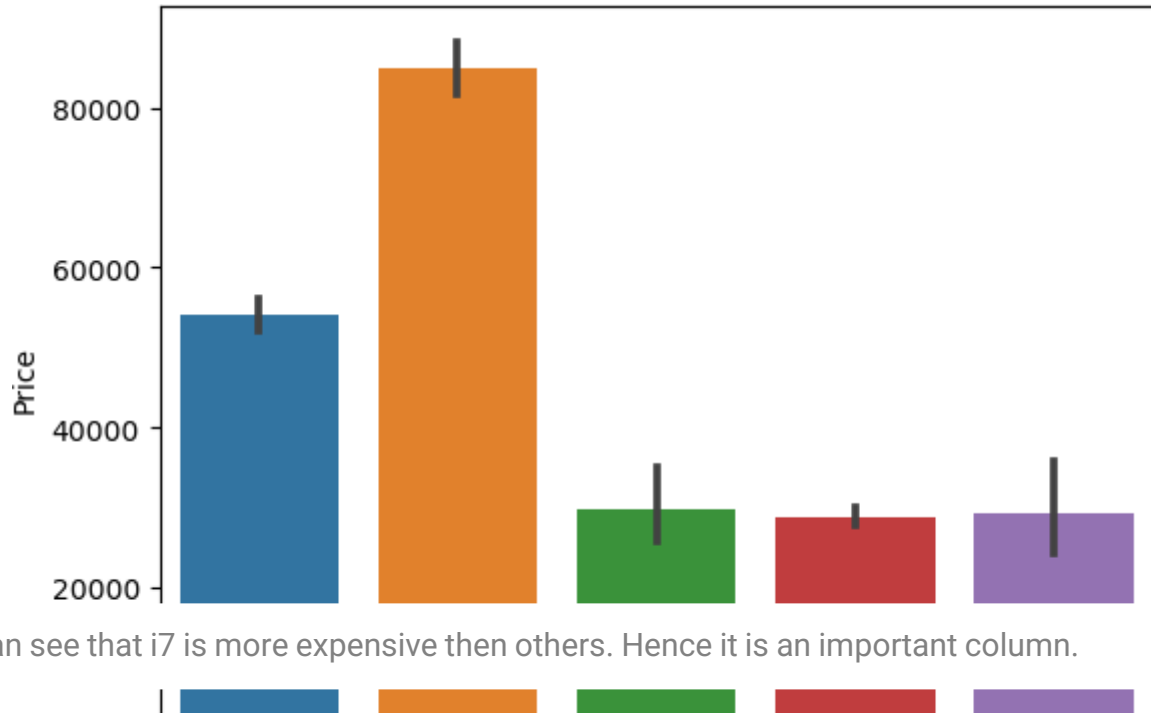
```
sns.barplot(x = df.Touchscreen, y = df.Price, color = 'green')
sns.barplot(x = df.IPS, y = df.Price, color = 'red', alpha = 0.5)
plt.show()
```



```
sns.barplot(x = df['CPU Brand'] , y = df.Price)
```



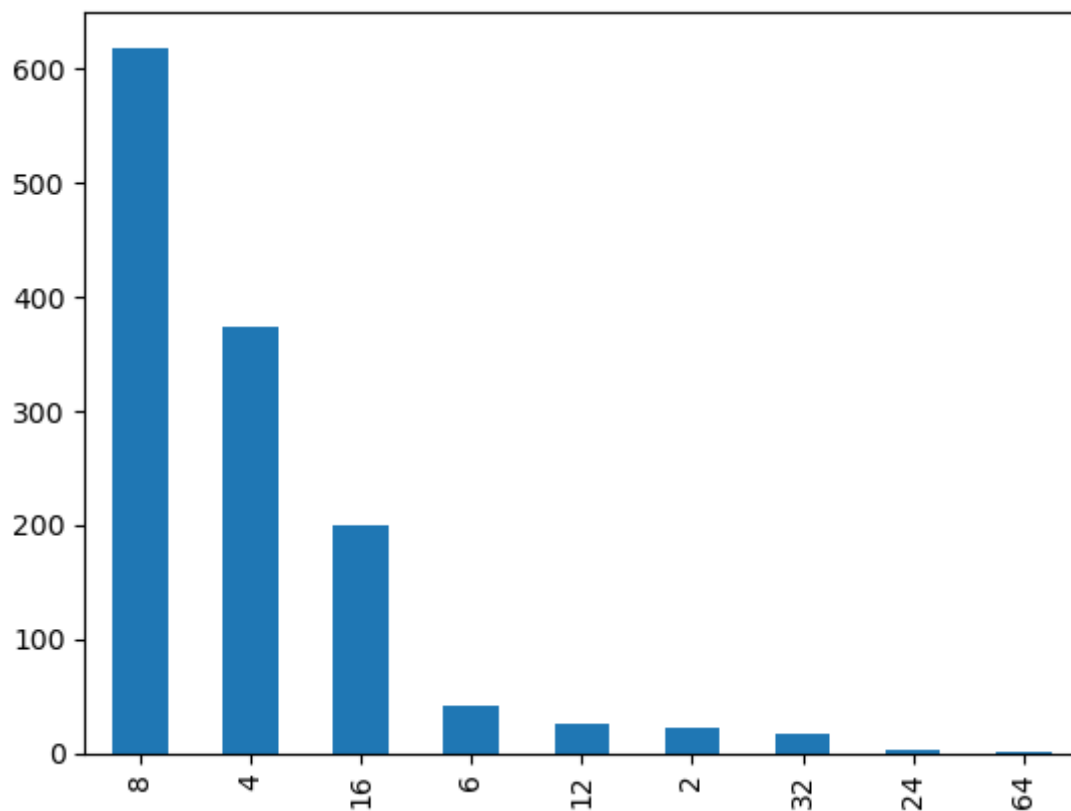
```
<Axes: xlabel='CPU Brand', ylabel='Price'>
```



We can see that i7 is more expensive than others. Hence it is an important column.

```
df.Ram.value_counts().plot(kind = 'bar')
```

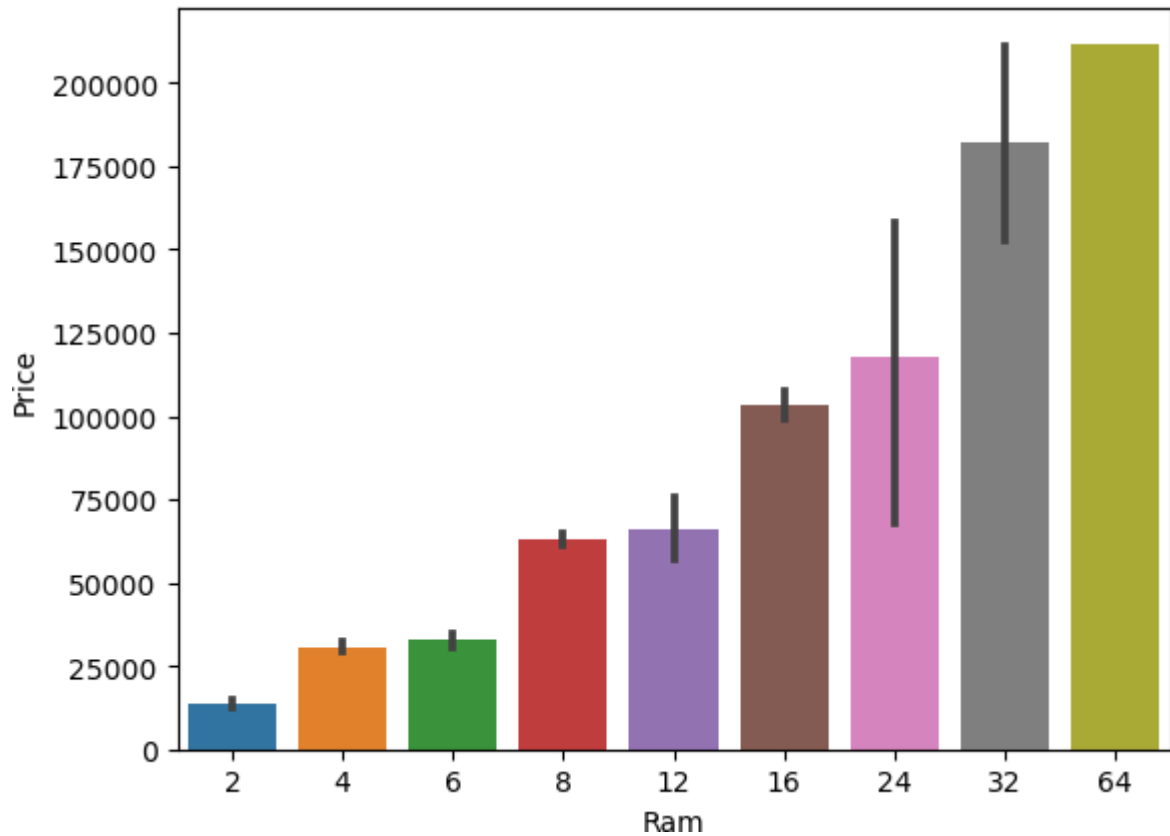
```
<Axes: >
```



We can see that maximum people buy 8GB ram laptops, followed by 4GB (People still buy budget laptops), 16gb and so on. Let us see if Ram contributes to the price or not

```
sns.barplot(x = df.Ram, y = df.Price)
```

```
<Axes: xlabel='Ram', ylabel='Price'>
```



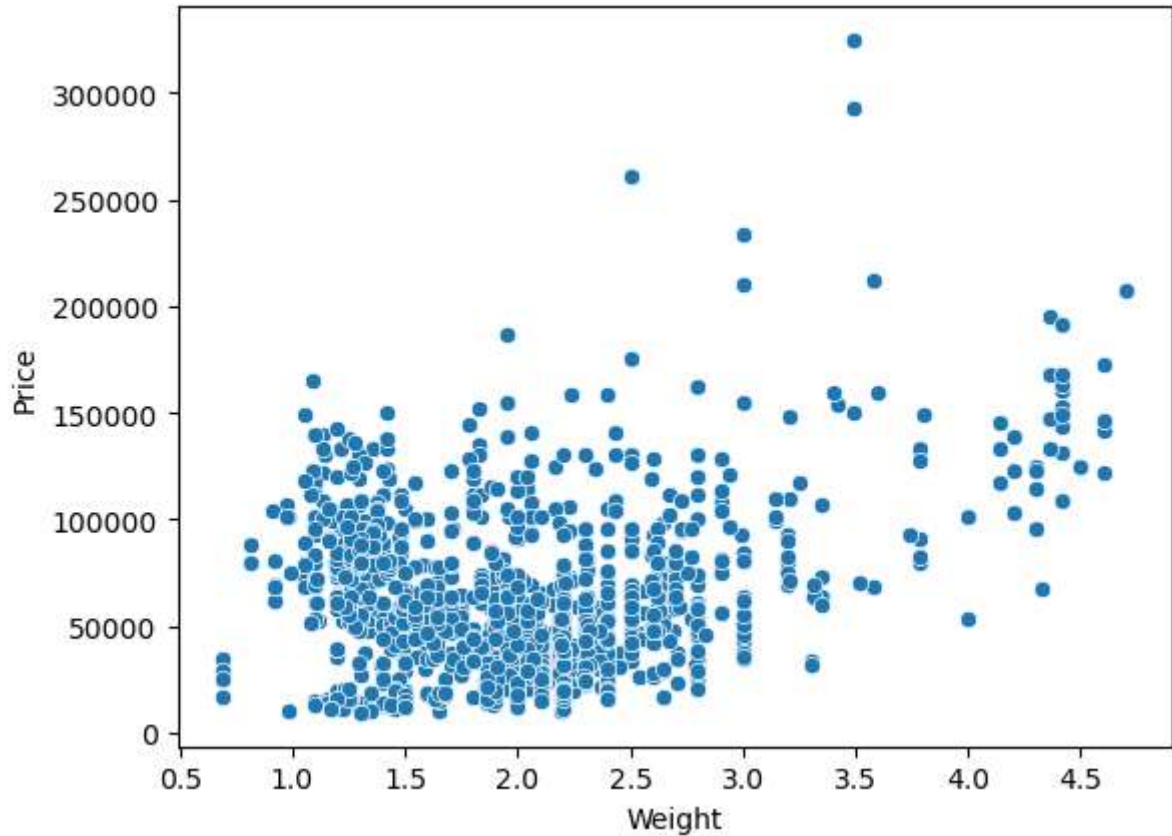
We can clearly see that RAM of a laptop definitely determines the price and affects the price. This graph also shows the consistency of the data that higher ram means higher price.

```
sns.barplot(x=df['os'],y=df['Price'])  
plt.xticks(rotation='vertical')  
plt.show()
```



```
sns.scatterplot(x=df['Weight'],y=df['Price'])
```

```
<Axes: xlabel='Weight', ylabel='Price'>
```



Double-click (or enter) to edit

```
df.corr()['Price']
```

```
C:\Users\hp India\AppData\Local\Temp\ipykernel_2724\815546952.py:1: FutureWarning: T
```

```
df.corr()['Price']
```

```
Ram      0.742905
```

```
Weight    0.209867
```

```
Price     1.000000
```

```
Touchscreen 0.192917
```

```
IPS       0.253320
```

```
PPI       0.475368
```

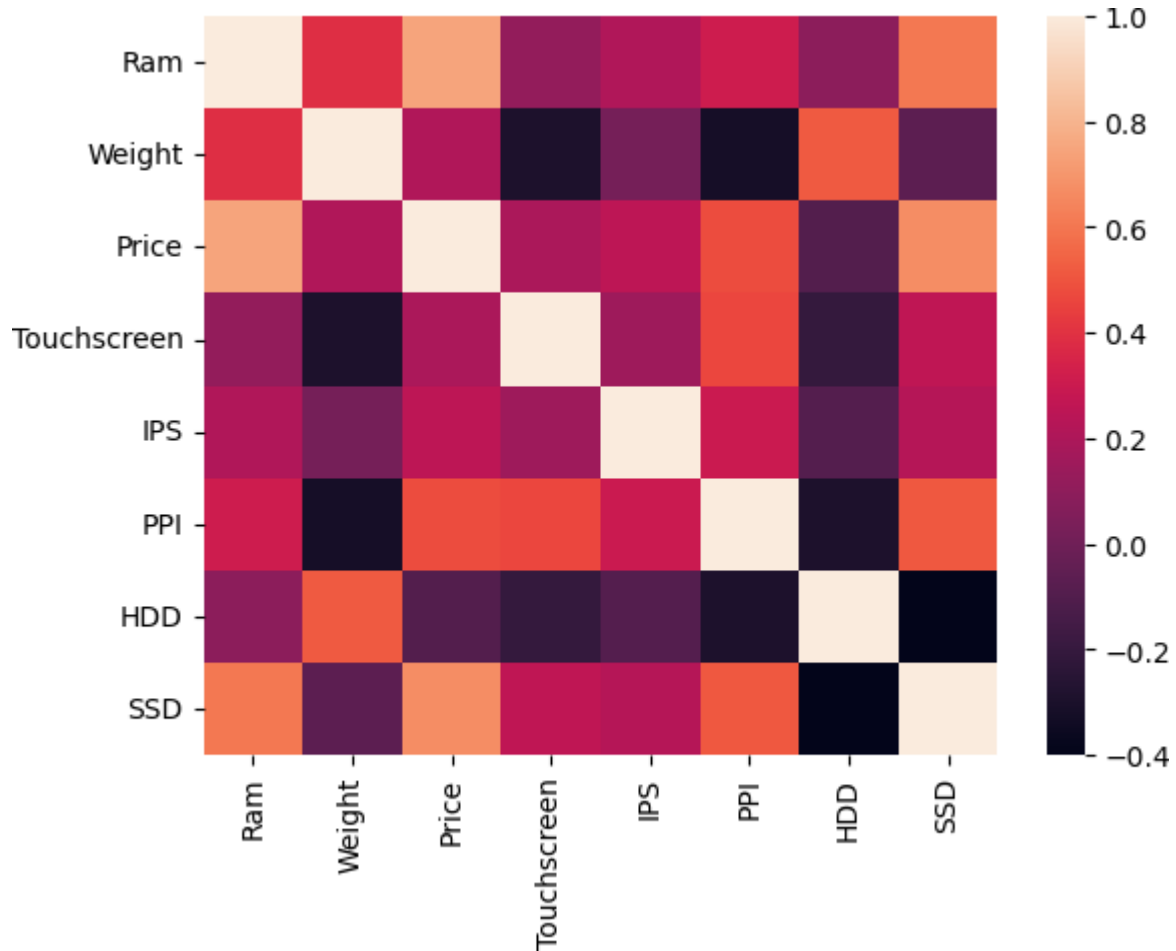
```
HDD      -0.096891
```

```
SSD       0.670660
```

```
Name: Price, dtype: float64
```

```
sns.heatmap(df.corr())
```

```
C:\Users\hp India\AppData\Local\Temp\ipykernel_2724\58359773.py:1: FutureWarning: Th
sns.heatmap(df.corr())
<Axes: >
```



▼ Building a Model

the target column is the most important as we have to predict it, but as we can see below, our target column is skewed. So we have to apply some transformation on it.

```
sns.distplot(df.Price)
```

C:\Users\hp India\AppData\Local\Temp\ipykernel_2724\824416946.py:1: UserWarning:

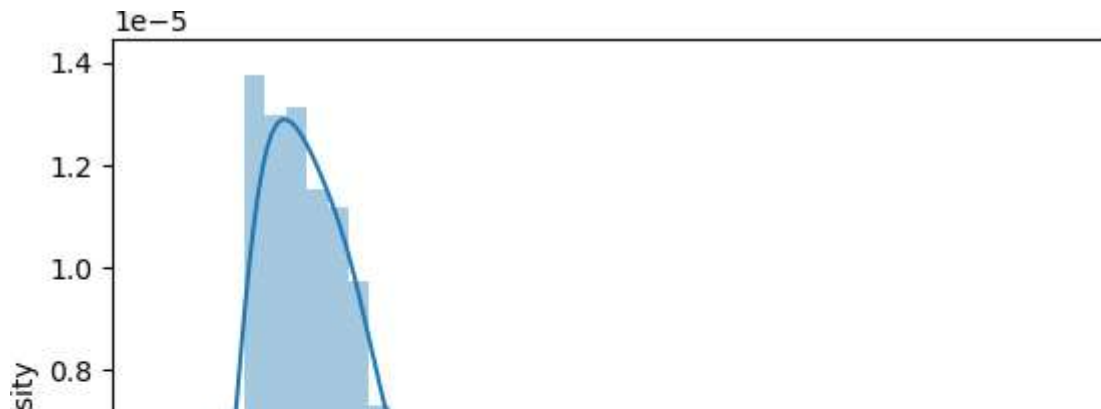
``distplot` is a deprecated function and will be removed in seaborn v0.14.0.`

Please adapt your code to use either ``displot`` (a figure-level function with similar flexibility) or ``histplot`` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see

<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
sns.distplot(df.Price)
<Axes: xlabel='Price', ylabel='Density'>
```



```
sns.distplot(np.log(df.Price))
```

C:\Users\hp India\AppData\Local\Temp\ipykernel_2724\2699817997.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with

Here we applied log transformation and made the graph less skewed. We will further see that it also increases our R2 score

One important factor to notice is that when we make predictions we will have to use exponent on our log predictions to get the original price value

0.7 1

Now we will Separate the class label from the data

```
X = df.drop(columns = ['Price'])
y = np.log(df['Price'])
```

X

	Company	TypeName	Ram	Weight	Touchscreen	IPS	PPI	CPU Brand	HDD
0	Apple	Ultrabook	8	1.37	0	1	226.983005	Intel Core i5	0
1	Apple	Ultrabook	8	1.34	0	0	127.677940	Intel Core i5	0
2	HP	Notebook	8	1.86	0	0	141.211998	Intel Core i5	0
3	Apple	Ultrabook	16	1.83	0	1	220.534624	Intel Core i7	0
4	Apple	Ultrabook	8	1.37	0	1	226.983005	Intel Core i5	0
...
1298	Lenovo	2 in 1 Convertible	4	1.80	1	1	157.350512	Intel Core i7	0
		2 in 1						Intel Core	

y

```
0    11.175755
1    10.776777
2    10.329931
3    11.814476
4    11.473101
...
1298 10.433899
1299 11.288115
```

```
1300      9.409283
1301     10.614129
1302      9.886358
Name: Price, Length: 1302, dtype: float64
```

▼ Now we will split the data into train and test using scikit learn

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.15, random_state =
```

▼ Now, for the categorical columns like CPU, GPU, TypeName, we have to apply OnehotEncoding

One-hot encoding is a technique used in machine learning and data analysis to convert categorical data into a numerical format that can be easily processed by algorithms. In one-hot encoding, each category in a categorical variable is represented as a binary vector, where each position in the vector corresponds to a particular category.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

This is particularly useful when we have a dataset with both numerical and categorical features, and we want to apply different preprocessing steps to these different types of features.

```
from sklearn.pipeline import Pipeline
```

A pipeline is a sequence of data processing steps that are chained together, with the output of one step serving as the input of the next.

in this case, the sparse parameter is set to False, which means that the transformer will output a dense array instead of a sparse matrix. This can be useful when working with smaller datasets, or when downstream processing steps require a dense array.

We are applying onehotencoder because our machine learning model takes numerical inputs only

```
step1 = ColumnTransformer(transformers = [
    ('col_tnf', OneHotEncoder(sparse = False, drop = 'first'), [0,1,7,10,11])
],remainder = 'passthrough')
```

```
from sklearn.linear_model import LinearRegression,Ridge,Lasso
step2 = LinearRegression()
```

We created two steps, and then we will feed it to the pipeline

```
pipe = Pipeline([
    ('step1', step1),
    ('step2', step2)
])

pipe.fit(X_train, y_train)
#Fitting the Pipeline into the training Data
y_pred = pipe.predict(X_test)
```

C:\Users\hp India\anaconda3\lib\site-packages\sklearn\preprocessing_encoders.py:828
warnings.warn(



```
from sklearn.metrics import r2_score, mean_absolute_error
print('R2 score is', r2_score(y_test, y_pred))
print('Mean Absolute Error is', mean_absolute_error(y_test, y_pred))
```

```
R2 score is 0.8073277450155012
Mean Absolute Error is 0.21017827953019166
```

Here we are using Regression Model because our final output is a continuous value. A good way to evaluate the performance i.e how our model fit the data is using metrics like R2 score and mean absolute error R2 score is $1 - \frac{(\text{difference between actual and predicted values})^2}{(\text{difference between mean and actual values})^2}$ It ranges from 0 - 1, where 0 indicates a very bad fit and 1 indicates perfect fit

Here we can see our R2 score is very good, but still we will try different models

Double-click (or enter) to edit

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.svm import SVR
```

▼ Ridge Regression

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 7, 10, 11])
], remainder='passthrough')
```



```

step2 = Ridge(alpha=10)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.8127331033739459
MAE 0.20926802210371445

```

▼ Lasso Regression

```

step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse_output=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = Lasso(alpha=0.001)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.8071853947620582
MAE 0.21114361575113458

```

▼ KNN

```

step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse_output=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = KNeighborsRegressor(n_neighbors=3)

pipe = Pipeline([

```

```

    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.8031008164264897
MAE 0.19268746498695286

```

▼ Decision Tree

```

step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse_output=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = DecisionTreeRegressor(max_depth=8)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.834966288261521
MAE 0.18434427017312613

```

▼ Support Vector Machine

```

step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse_output=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = SVR(kernel='rbf',C=10000,epsilon=0.1)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

```

```
pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2 score 0.8083168388457612
MAE 0.20239400567814725
```

▼ Random Forest (Best Performance)

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse_output=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')
```

```
step2 = RandomForestRegressor(n_estimators=100,
                              random_state=3,
                              max_samples=0.5,
                              max_features=0.75,
                              max_depth=15)
```

```
pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])
```

```
pipe.fit(X_train,y_train)
```

```
y_pred = pipe.predict(X_test)
```

```
print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2 score 0.8873402378382488
MAE 0.15860130110457718
```

```
import numpy as np
query = np.array([df.Company.unique(), df.TypeName.unique(), df.Ram.unique(), df.Weight.un
```

```
df.head(1)
```

Company	TypeName	Ram	Weight	Price	Touchscreen	IPS	PPI	CPU Brand	HD
<div> <div></div> <div></div> </div>									

```
'''Companies : Apple', 'HP', 'Acer', 'Asus', 'Dell', 'Lenovo', 'Chuwi', 'MSI',
'Microsoft', 'Toshiba', 'Huawei', 'Xiaomi', 'Vero', 'Razer',
'Mediacom', 'Samsung', 'Google', 'Fujitsu', 'LG
```

```

'''
company = 'Apple'

'''
'Ultrabook', 'Notebook', 'Netbook', 'Gaming', '2 in 1 Convertible',
    'Workstation'
'''
TypeName = 'Notebook'

# 8, 16, 4, 2, 12, 6, 32, 24, 64
Ram = 8

Weight = 1.37

# 1 for Yes, 0 for No
Touchscreen = 1

# 1 for Yes, 0 for No
IPS = 1

# 150 - 300
PPI = 225

#'Intel Core i5', 'Intel Core i7', 'AMD Processor', 'Intel Core i3', 'Other Intel Processo
CPU_Brand = 'Intel Core i7'

# 0, 500, 1000, 2000, 32, 128
HDD = 1000

# 128, 0, 256, 512, 32, 64, 1000, 1024, 16, 768, 180, 240, 8
SSD = 512

#'Intel', 'AMD', 'Nvidia'
Gpu_Brand = 'Intel'

# 'Mac', 'Others/No OS/Linux', 'Windows'
os = 'Windows'

query1 = np.array([company, TypeName, Ram, Weight, Touchscreen, IPS, PPI, CPU_Brand, HDD, SSD, Gpu_B
query1 = query1.reshape(1,12)

np.exp(pipe.predict(query1))

```

```

C:\Users\hp India\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X do
warnings.warn(
array([89664.2713822])

```



Here we applied the Random Forest Regressor, as it has the highest R2 score. We created 12 variables, giving options to choose various types from, and then we feed it to the pipe.predict and get the output successfully Hence our project ends here

