**LAPTOP PRICE PREDICTION USING MACHINE**

**LEARNING ALGORITHM**

* **AMARNATH P**
* **DINESH**
* **ROGITH SANJAY**

**INTRODUCTION**

Top of Form

Laptops have become an integral part of our lives, from work to entertainment and everything in between. As the demand for laptops continues to grow, so does the need for accurate price prediction models. A laptop price prediction model can help manufacturers and retailers to set competitive prices and maximize profits, while also helping consumers to make informed purchasing decisions.

In this project, we aim to develop a machine learning model that can accurately predict the prices of laptops based on various features such as processor speed, RAM size, storage capacity, screen size, and brand. We will use a dataset of historical laptop prices and features to train and evaluate our model.

The project's objective is to develop an accurate and reliable laptop price prediction model that can benefit both manufacturers and consumers, and second, to gain insights into the factors that influence laptop prices in the market. By achieving these objectives, we hope to make a significant contribution to the field of laptop price prediction and provide practical benefits to both businesses and consumers.

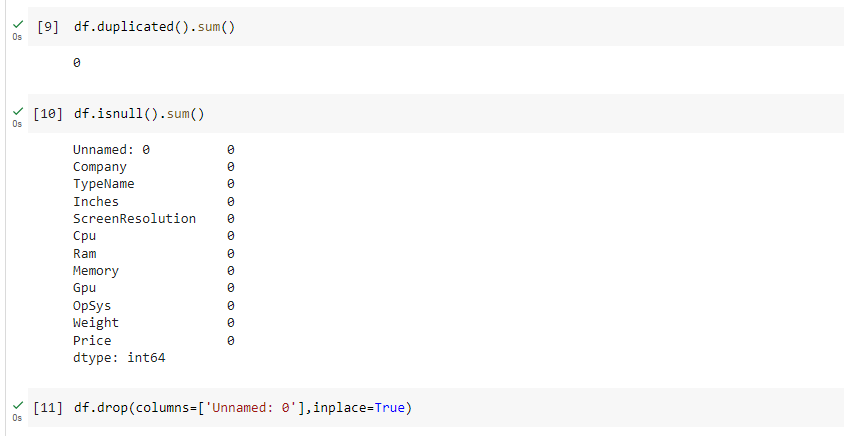
In this report, we will provide a detailed description of the data collection and preparation process, the machine learning models used, the evaluation criteria, and the results.

**DATA COLLECTION AND PREPARATION**

The first step in developing a laptop price prediction model is to gather a comprehensive dataset of historical laptop prices and features. We collected data from KAGGLE dataset.

The dataset consisted of 1,300 laptop entries, each with 10 features including brand, model, processor speed, RAM size, storage capacity, screen size, and price. We also included categorical variables such as operating system, and whether the laptop had a touchscreen or not.

Once we had collected the dataset, we conducted a preliminary inspection to identify any missing or erroneous values.

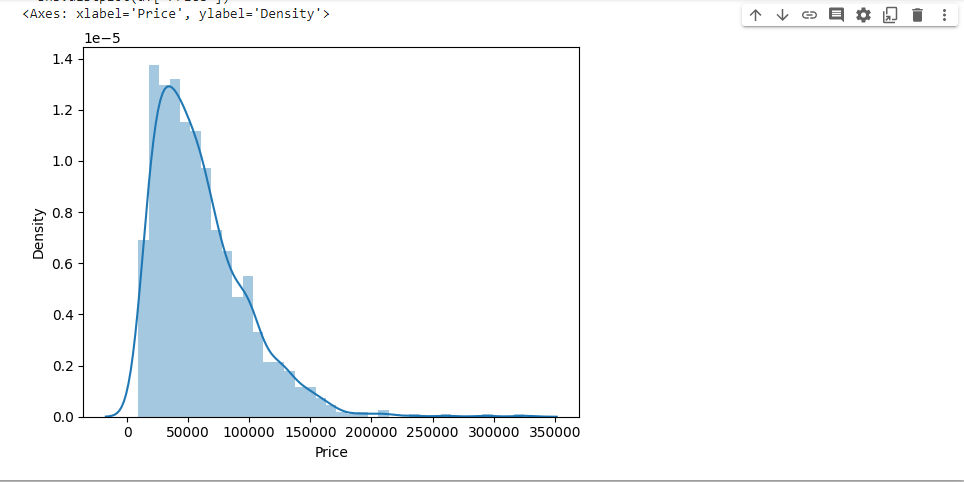


***SKEWNESS OF THE DATA***

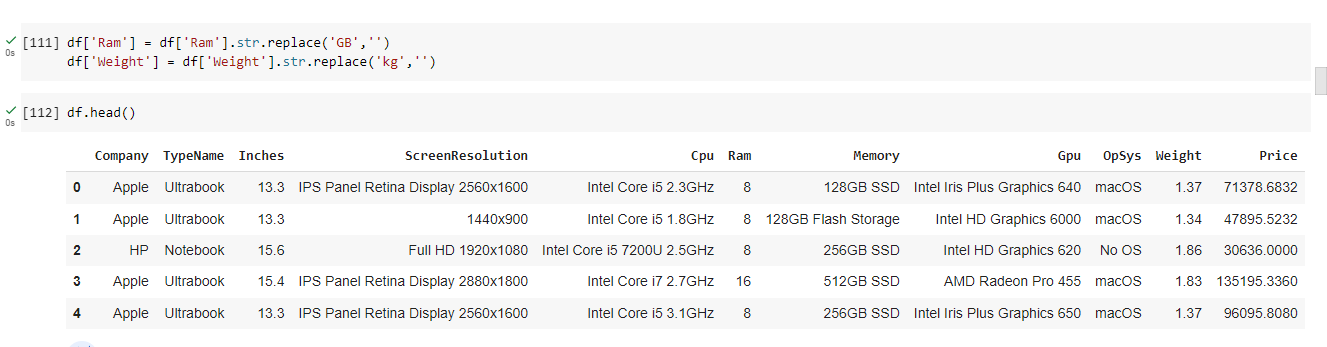
Before starting the data analysis we wanted to find out is there any skewness present in our data set and using the function df['Price'].skew(), we founded that our dataset contained skewness

SKEWNESS SCORE

Skewness of the data: 1.5208655681688517



We first converted GB and WEIGHT as “RAM” and “KG” and then converted the categorical variables into numerical values and standardized the numerical variables to ensure that they were on the same scale.

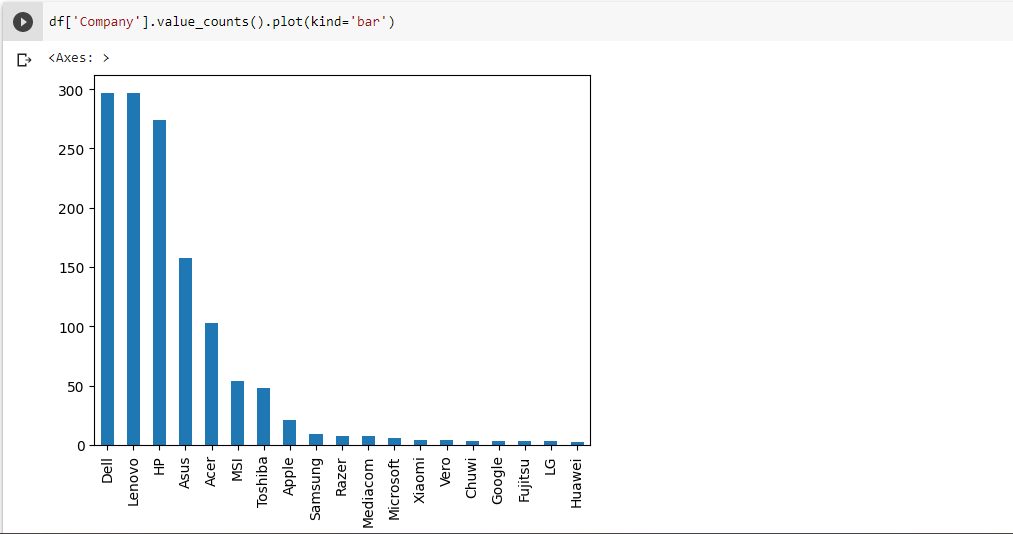




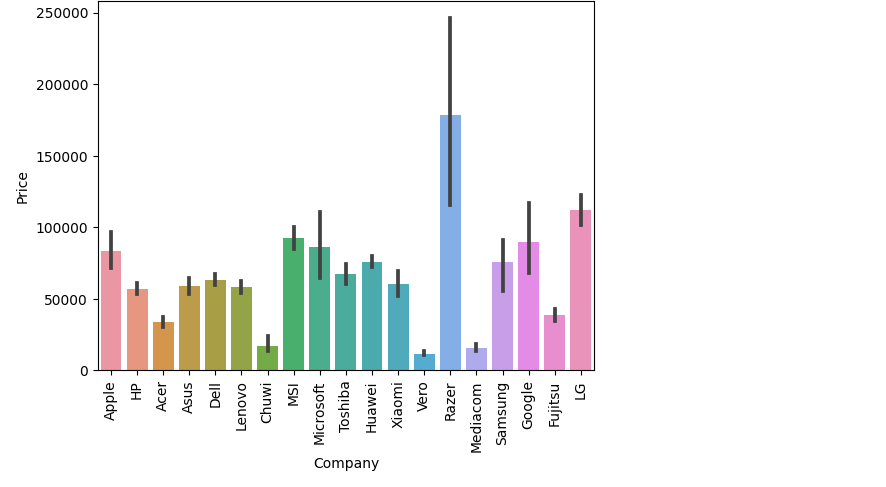
***DATA ANALYSIS***

Next, we conducted exploratory data analysis to gain a better understanding of the dataset's characteristics and We used visualizations such as scatterplots and boxplots to identify any patterns or relationships between the features and the price. We also used correlation analysis to determine the strength of the relationships between the features.

***COMPANY BAR GRAPH***



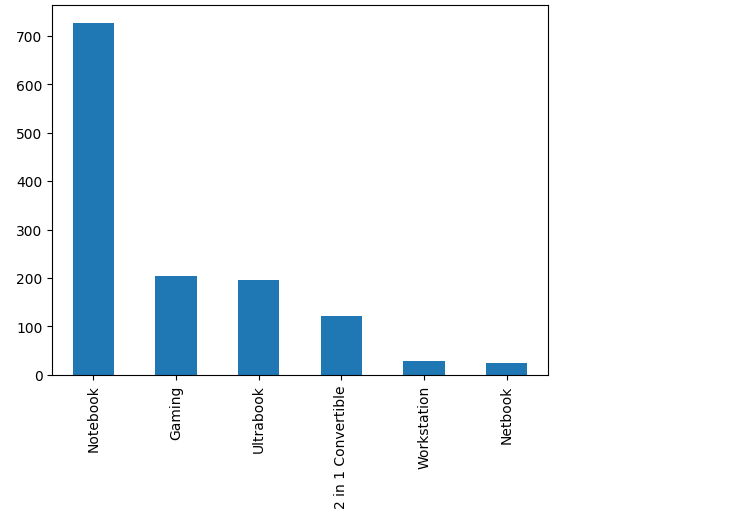
***COMPANY VS PRICE***

****

Each bar in the graph represents a different laptop company, and the height of the bar indicates the average price of laptops for that company. By comparing the heights of the bars, we can easily see which companies tend to sell more expensive laptops, and which companies tend to sell more affordable laptops.

We can notice that laptop brands such as Razer, LG, MSI and APPLE have more high price compared to others which results in skewness of the data.

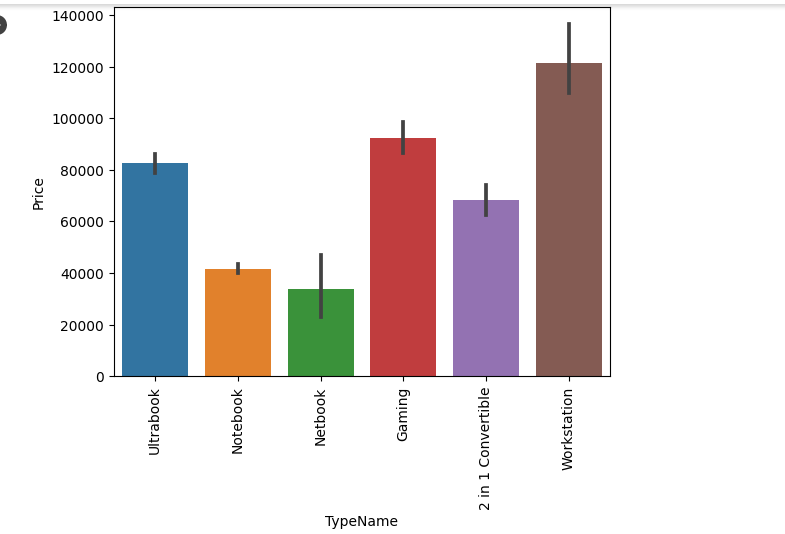
***TYPES OF LAPTOPS BAR PLOT***

****

The graph shows that the most common laptop type in the dataset is notebook, followed by gaming and ultrabook. By looking at the graph, we can quickly see which laptop types are most prevalent in the dataset, and get a sense of the overall distribution of laptop types

This information is particularly important for our project, as the TypeName category is a key factor in determining the price of a laptop. By understanding which laptop types are most common in the dataset, we can better analyze the relationship between laptop type and price, and develop more accurate price prediction models. Overall, the bar graph provides us with valuable insights into the distribution of laptop types in the dataset, and highlights the importance of the TypeName category for our analysis.

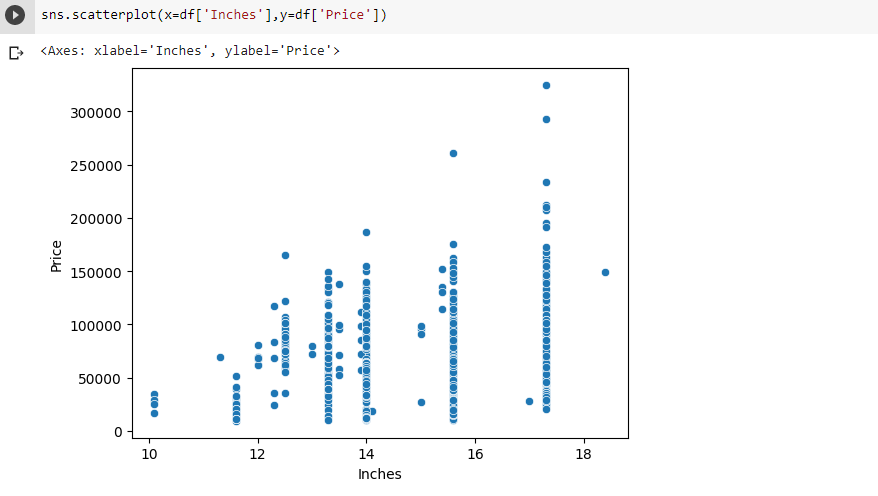
***TYPENAME VS PRICE***



The bar graph shows the relationship between the laptop type and its price. From the graph, we can observe that the Gaming laptop type has the highest average price, while the Notebook type has the lowest average price. The Ultrabook and Netbook types fall in between with similar average prices. This indicates that the type of laptop has a significant impact on its price. Therefore, when predicting the price of a laptop, it is important to consider its type in addition to other relevant features

***INCHES VS PRICE SCATTERPLOT***

To explore the relationship between screen size and laptop prices, we first plotted a scatter plot of the screen size variable (in inches) against the price variable. The scatter plot is shown below:



From the scatter plot, we can see that there is a general trend of increasing prices with increasing screen size. However, there is also a fair amount of variability in the relationship, with some laptops having higher or lower prices than expected based on their screen size alone.

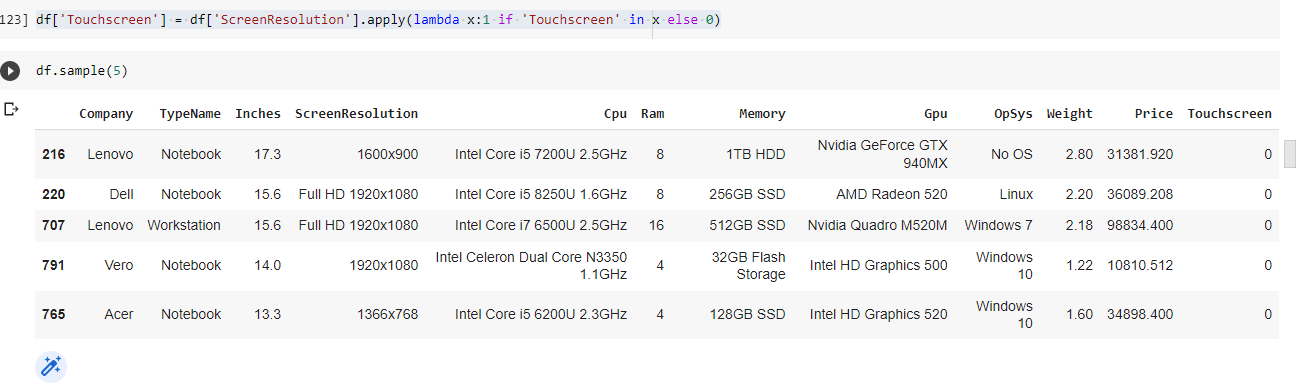
Our analysis suggests that screen size is a significant factor that can affect laptop prices, but it is important to consider other factors as well when predicting laptop prices.

***CREATING A NEW COLUMN CALLED TOUCHSCREEN***

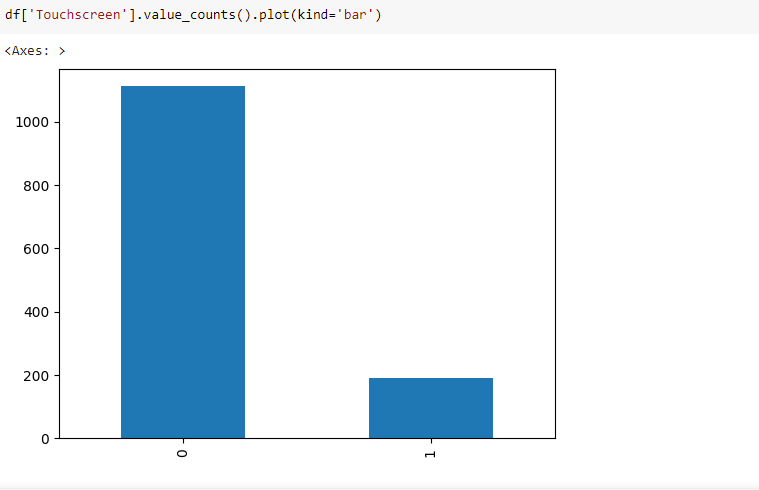
We anlayzed our screen resolution column and noticed the two categories such as touchscreen and ips. So we wanted to create a new column called touchscreen and ips that will help us to predict the price more accurately.

The 'Touchscreen' column was created to identify which laptops have touchscreen displays. This information can be useful in predicting the price of the laptops since touchscreen displays are usually more expensive than non-touchscreen displays.

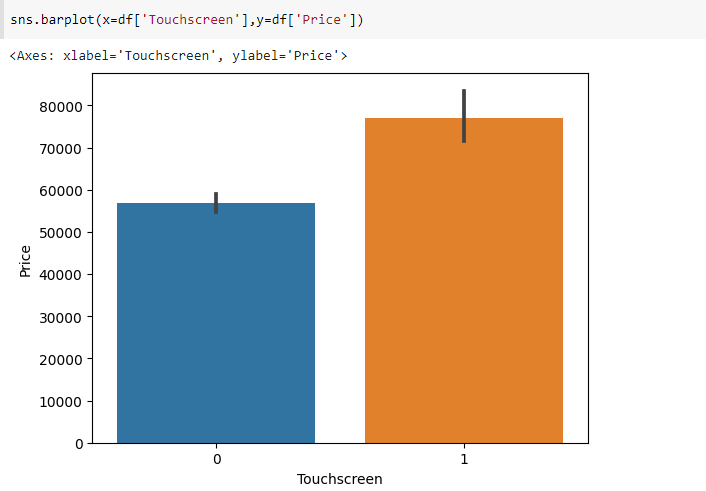
To create a new binary column 'Touchscreen' based on the 'ScreenResolution' column, we used the 'apply' function and used a lambda function to each value in the 'ScreenResolution' column. If the value contains the string 'Touchscreen', the lambda function returns 1, otherwise 0. This way, the 'Touchscreen' column will have 1 if the laptop has a touchscreen display, and 0 if it doesn't.



***BAR PLOT FOR TOUCH SCREEN***

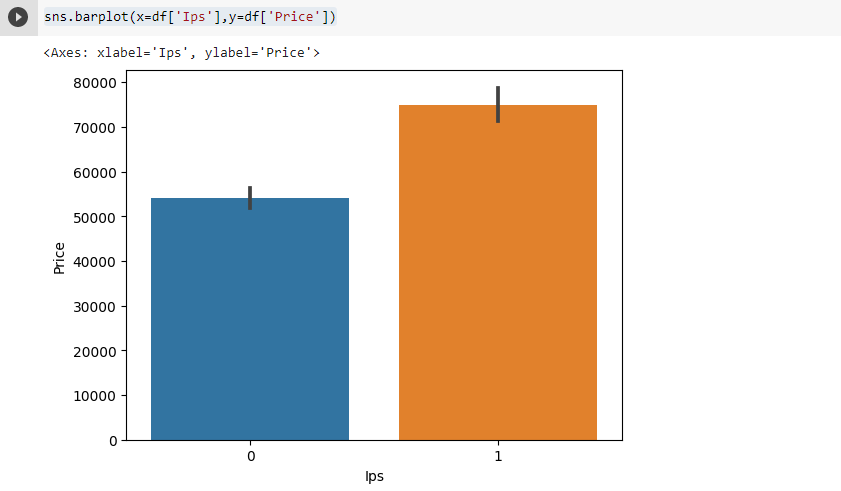
****

As we can see from the chart, touchscreen laptops are more prevalent in the dataset, but the difference is not extremely pronounced. This information can be useful in our analysis as we can investigate if the presence of a touchscreen affects the laptop's price or other features.

****

The barplot suggests that there is a significant difference in prices between touchscreen and non-touchscreen devices. On average, touchscreen devices appear to be more expensive than non-touchscreen devices. This may be due to the additional technology required to enable touch functionality, as well as the added convenience and usability that touchscreens provide.

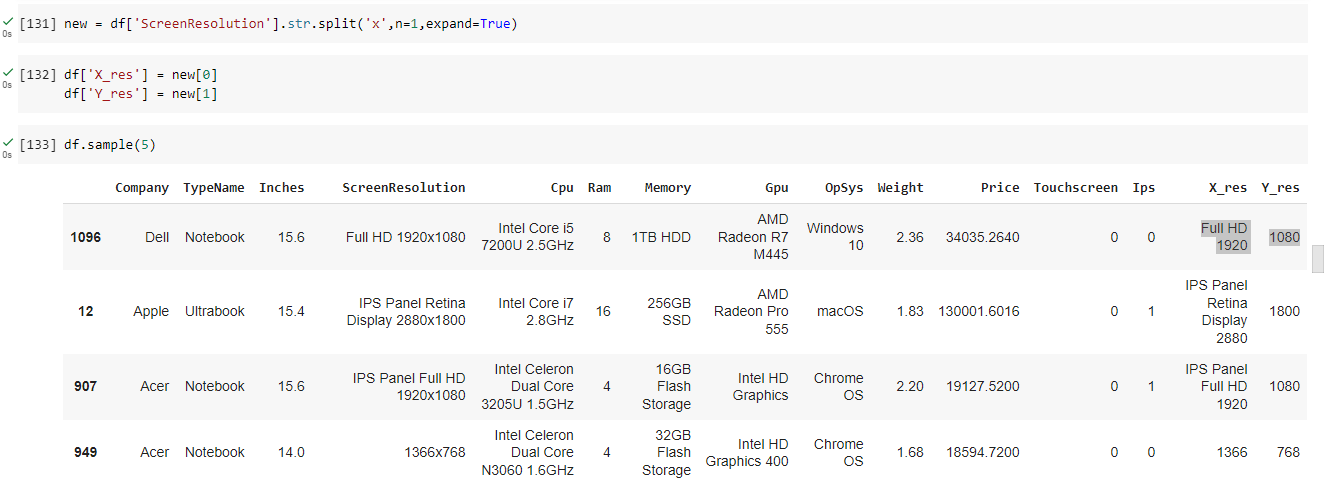
Therefore we can consider the touchscreen category for our prediction.



We created a bar graph to examine the relationship between IPS (In-Plane Switching) technology and laptop prices. The graph shows that laptops with IPS technology have higher average prices compared to laptops without IPS technology. IPS technology is known for providing wider viewing angles, better color accuracy, and higher contrast compared to other display technologies. Our analysis suggests that IPS technology is a premium feature that adds value to a laptop and affects its price. Therefore, when predicting the price of a laptop, it is important to consider whether the laptop has IPS technology or not.

After creating a new column called touchcrreen and ips, We analyzed the screen resolution feature in our dataset and noticed that it was represented as a single variable. However, we realized that dividing the screen resolution into its x and y components could provide additional information that could help improve the accuracy of our laptop price prediction model. Therefore, we created two new variables, x-resolution and y-resolution, by splitting the original screen resolution variable.

|  |  |
| --- | --- |
| Full HD 1920 | 1080 |



First we separated the columns based on the delimeter ‘x’ and assigned it to “new”. And then we added that two columns “X\_res” and “Y\_res” in the main data and thus achieved splitting the entire resolution into x and y resolution.

***FINDING THE CORRELATION***

df.corr()['Price']

Inches 0.068197

Ram 0.743007

Weight 0.210370

Price 1.000000

Touchscreen 0.191226

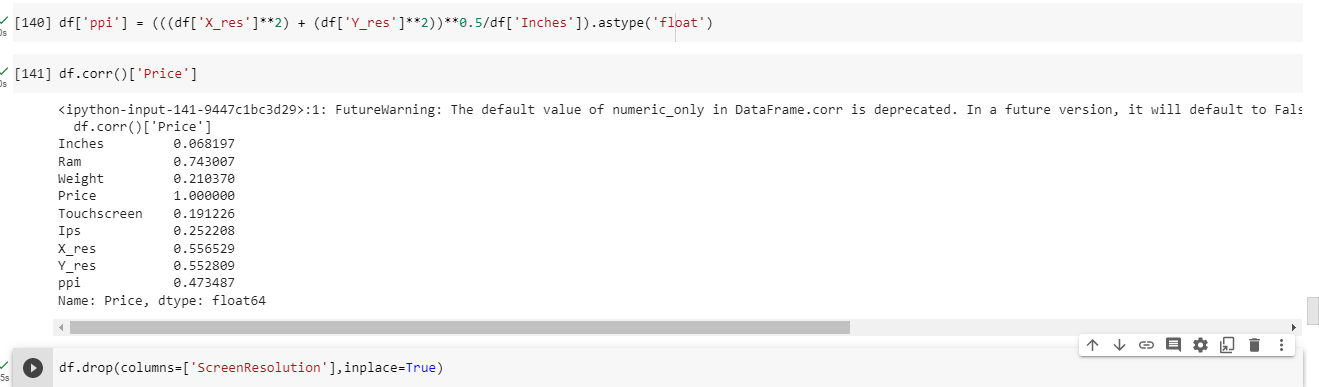
Ips 0.252208

X\_res 0.556529

Y\_res 0.552809

Name: Price, dtype: float64

We found that both x-resolution and y-resolution were positively correlated with laptop price, indicating that higher screen resolutions command higher prices. Therefore, we included these new variables in our feature set for the laptop price prediction model.

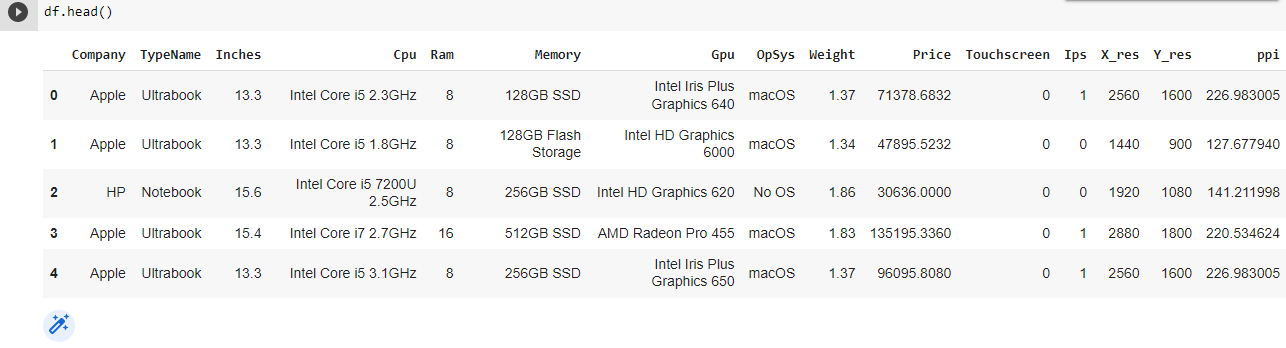


Further, we calculated the PPI (Pixels Per Inch) of the laptop screen using the screen size in inches and the x and y resolution components. PPI is a measure of pixel density and is often used as a metric for screen quality. We used the following formula to calculate PPI:

PPI = sqrt(x\_resolution^2 + y\_resolution^2) / screen\_size\_inches

We then added the PPI feature to our dataset and analyzed its relationship with laptop price. We found that there was a moderate positive correlation between PPI and laptop price, indicating that higher PPI screens tend to command higher prices. Therefore, we included PPI as a feature in our laptop price prediction model and dropped the screen resolution category.

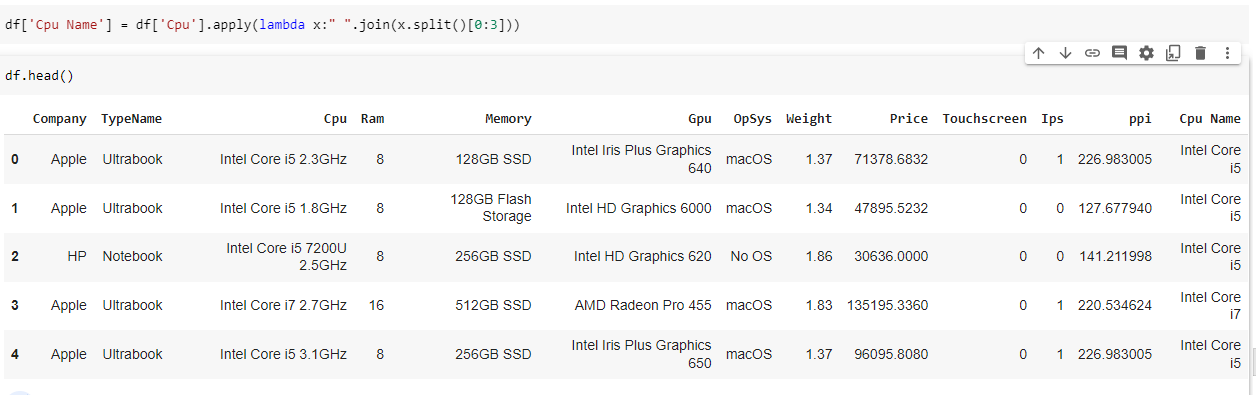
This is the data that we had after doing the preprocessing and analsis. We further wanted to anlayze more to get a more accurate rate. Since inches was used to calculate the ppi, we dropped the inches column later.

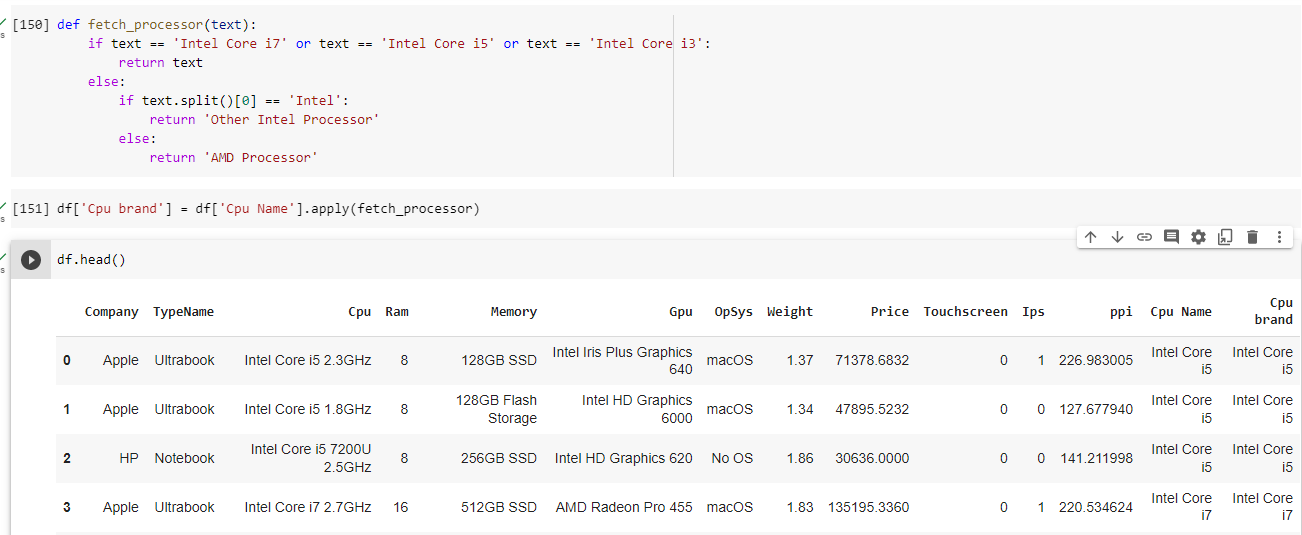


***HANDLING THE CPU BRANDS***

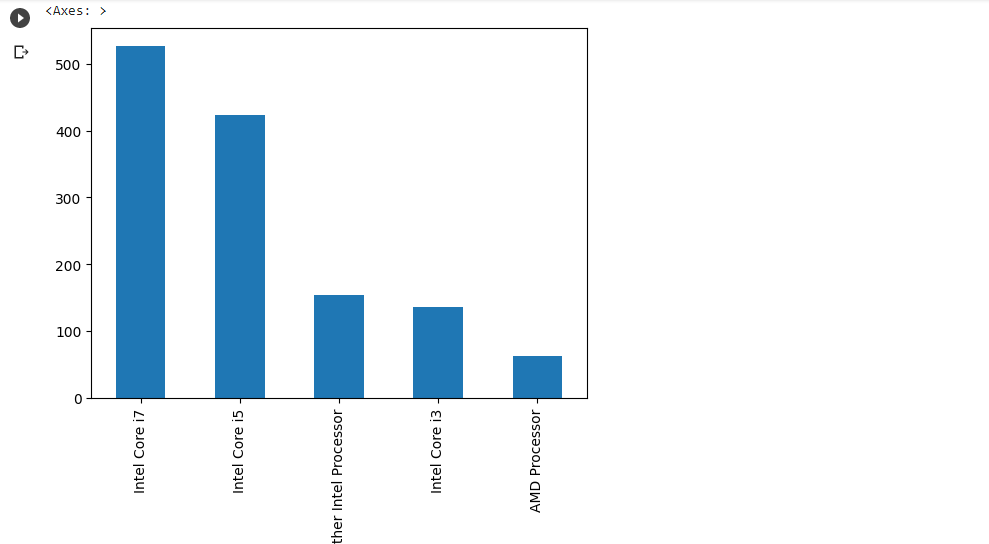


In order to get a better understanding of the different types of processors in our dataset, we decided to extract the first three words from the 'CPU' column using a lambda function and created a new column called 'CPU Name'

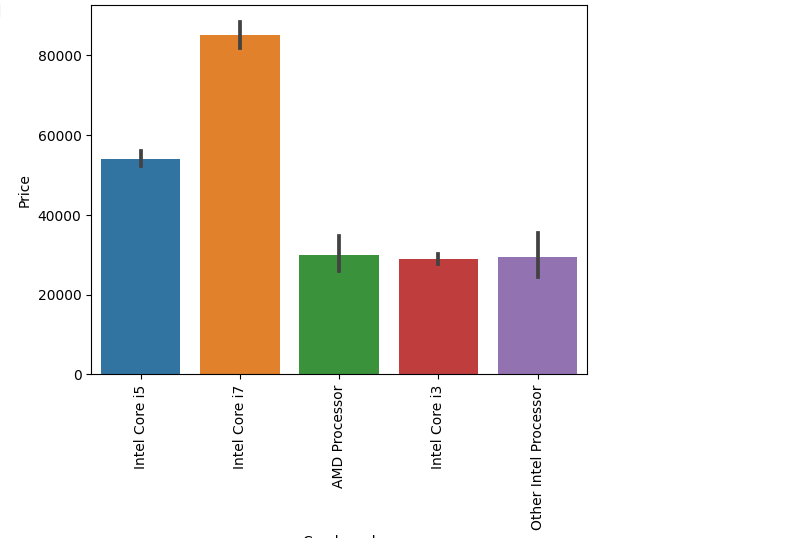


****

We then applied a custom function called 'fetch\_processor' to the 'Cpu Name' column to categorize the processor brands into three groups: Intel Core i7, Intel Core i5, Intel Core i3, and Other Intel Processor and AMD Processor. We used this function to create a new column called 'Cpu brand', which would help us in further analysis. This allowed us to see the distribution of different processor brands in our dataset, which could potentially impact the laptop price.



We visualized the distribution of processor brands using a bar plot, which showed that the majority of laptops in our dataset had Intel Core i5 processors, followed by Intel Core i7 processors and Other Intel Processors, with a small number of laptops having AMD processors.



The bar plot of CPU brand vs price revealed a strong correlation between the type of CPU and the price of the laptop. Laptops equipped with Intel Core i7 processors had the highest average price, followed by Intel Core i5 and i3. On the other hand, laptops with other Intel processors and AMD processors had lower average prices. This indicates that the type of CPU is a significant factor in determining the price of a laptop.

***CLEANING THE MEMORY DATA***

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

We saw there were several data present in a single memory column. So we tried to clean this data.

Initially, the memory information was in the format of "8GB DDR4 RAM + 512GB SSD".

We converted this information into a more useful format that would help us analyze the data better.

***CODE WE USED FOR TRANSFORMING THE MEMORY DATA***

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\_Storage"])

df.drop(columns=['first', 'second', 'Layer1HDD', 'Layer1SSD', 'Layer1Hybrid',

       'Layer1Flash\_Storage', 'Layer2HDD', 'Layer2SSD', 'Layer2Hybrid',

       'Layer2Flash\_Storage'],inplace=True)

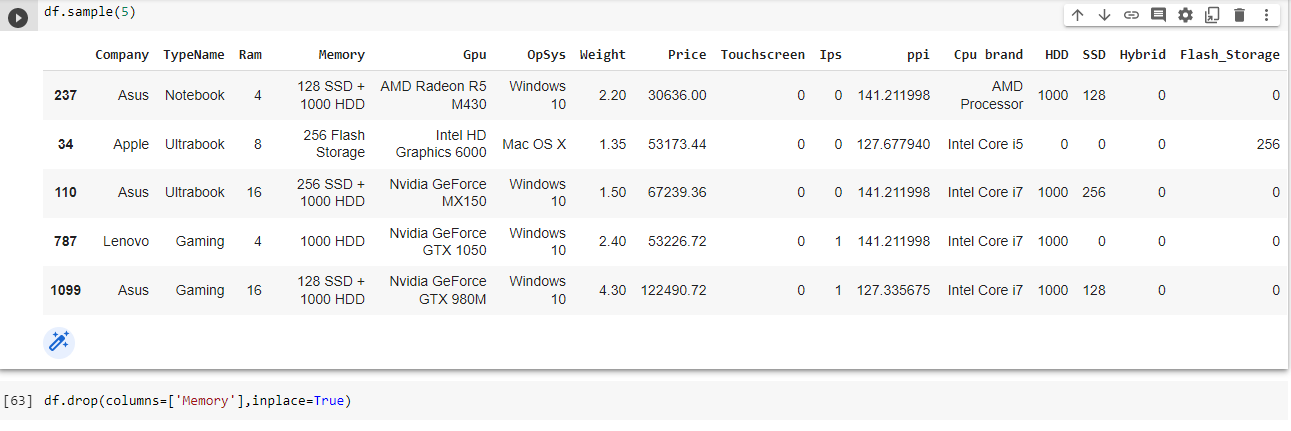
We first removed any decimal points and the 'GB' or 'TB' from the memory information.

Then we split the memory information into two parts based on the '+' sign.

We created separate columns for the type of memory storage, such as HDD, SSD, Hybrid, or Flash Storage, in the two parts of the memory information.

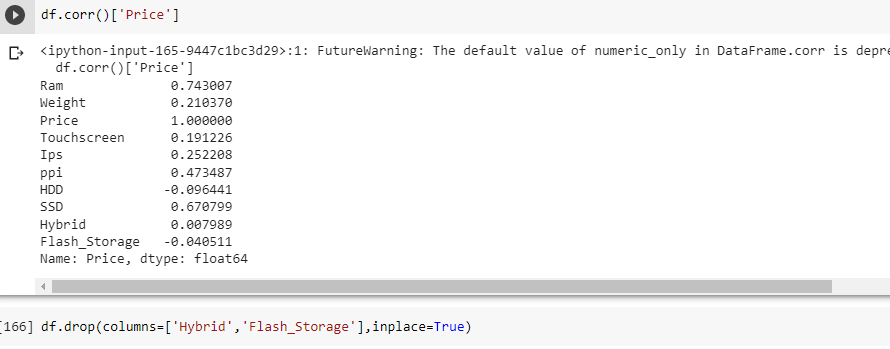
We also converted the memory storage capacity values into integers.

We then summed up the storage capacities of the same type of memory storage to create four new columns for HDD, SSD, Hybrid, and Flash Storage. These columns now give us the total amount of storage capacity in each laptop.



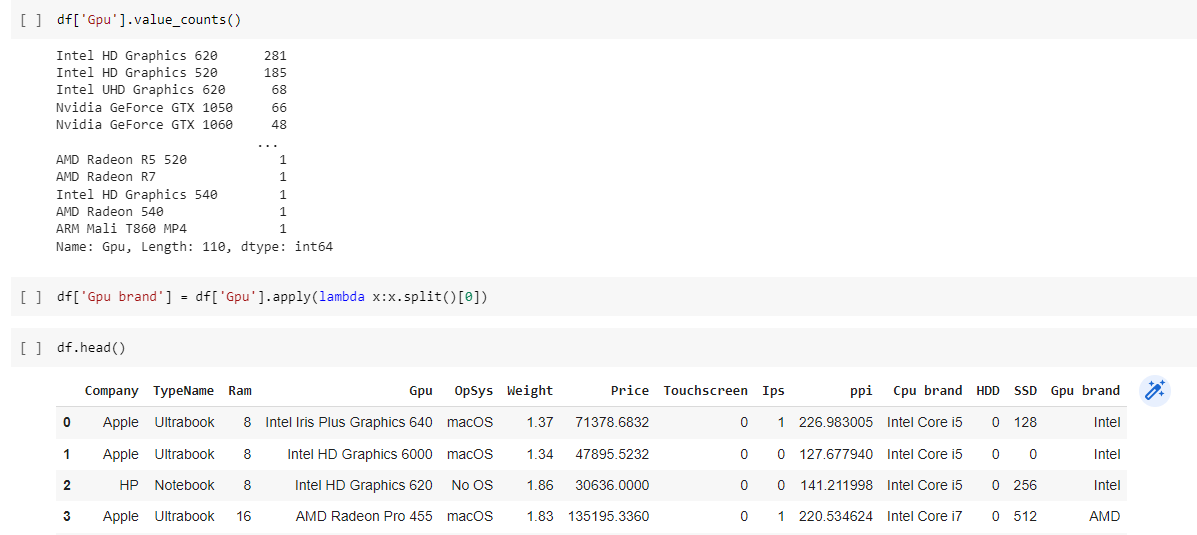
Overall, we transformed the memory information in a more usable format that can be used for analysis in predicting laptop prices based on the storage capacity.

***FINDING THE CORRELATION***



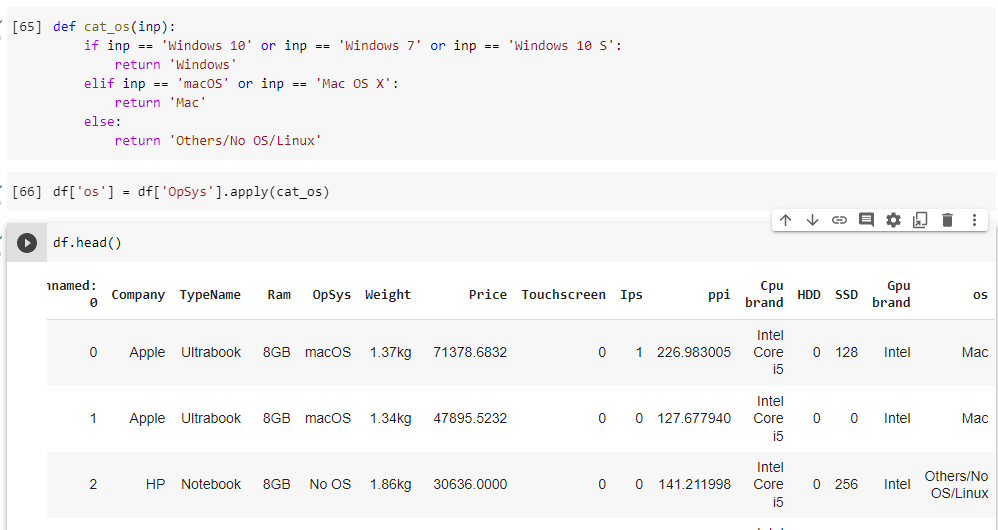
We got a good correlation with respect to price, except flash storage and hybrid. So we dropped the hybrid and flash storage columns.

***DEALING WITH THE GPU AND OPERATING SYSTEM DATA***



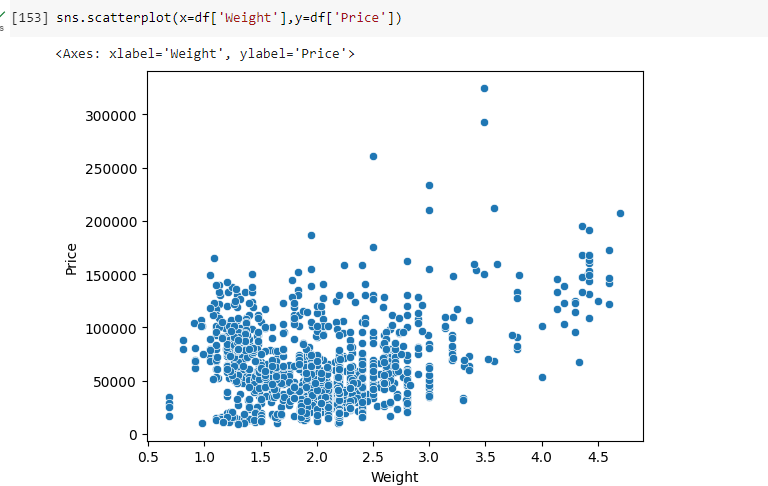
We tried to clean the data present in the Gpu column by getting only the gpu brand and dropping the remaining unwanted data.

**Then while dealing operating system, we changed the operating system as only the brand name as given below using function.**



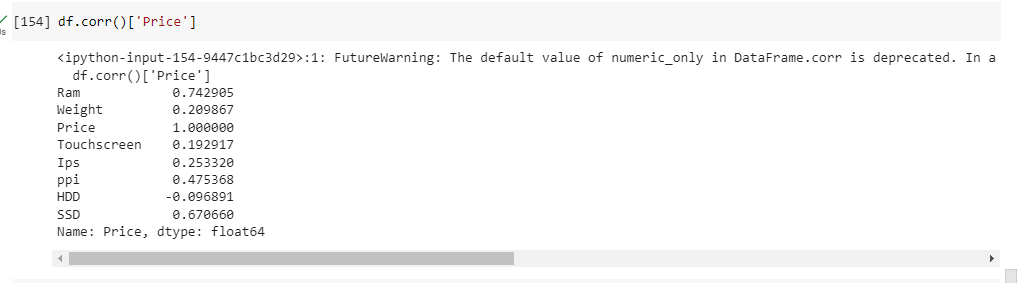
**At last we dropped the OpSys column and added it to our data to predict the price.**

***Scatter plot between Weight and Price***



To visualize the relationship between weight and price, a scatterplot was created, with weight on the x-axis and price on the y-axis. The scatterplot shows a positive trend, with a slight increase in price as weight increases.

Upon analyzing the scatterplot, we observed that there is a slight increase in price as weight increases.



**After all the analysis and data cleaning process, we checked the correlation of our data with price and almost every data had a positive correlation and thus our data is ready to train.**

**At last we wanted to decrease the skewness present in our data using log transformation and reduced the skewness score as**

Skewness score of the 'price' column: 1.5197503994318975

***SPLITTING THE DATASET AS X AND Y***

We seperated the dataframe into two variables: X and y. X contains all the columns of the original dataframe except for the 'Price' column, while y contains the values of the 'Price' column that have been transformed using the natural logarithm function (np.log()).

X = df.drop(columns=['Price'])

y = np.log(df['Price'])

The purpose of transforming the 'Price' column using the natural logarithm function is often to improve the distribution of the data, especially if it is skewed.

***Test and Train***

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=2)

In this code snippet, we are using a machine learning library called scikit-learn to split our data into a training set and a testing set. This is important because it allows us to train our machine learning model on a portion of the data and then evaluate how well it performs on the remaining data.

We are using the train\_test\_split function from scikit-learn, which randomly splits the data into a training set and a testing set. The X variable contains all the columns of the original data except for the 'Price' column, while the y variable contains the values of the 'Price' column that have been transformed using the natural logarithm function.

The test\_size parameter is set to 0.15, which means that 15% of the data will be used for testing and the remaining 85% will be used for training. The random\_state parameter is set to 2, which ensures that the same split is generated each time the code is run.

The resulting variables are X\_train, X\_test, y\_train, and y\_test. X\_train and y\_train contain the training data, while X\_test and y\_test contain the testing data. These variables can then be used to train and evaluate a machine learning model.

By splitting our data into a training set and a testing set, we can train our machine learning model on one set of data and then evaluate how well it performs on a separate set of data. This helps to ensure that our model is not overfitting to the training data and can generalize well to new data.

***MODEL AND FITTING***



In this code snippet, we are using scikit-learn to create a machine learning pipeline that consists of two main steps: preprocessing and model training.

In the preprocessing step, we are using a ColumnTransformer to transform some of the columns of our data using one-hot encoding. This is done to convert categorical variables into numerical variables that can be used in the machine learning model. The resulting transformed columns are combined with the original numerical columns using the remainder='passthrough' parameter.

In the model training step, we are using a RandomForestRegressor model with 100 decision trees. This model is a type of ensemble learning method that combines multiple decision trees to improve performance. We are setting various hyperparameters of the model such as max\_samples, max\_features, and max\_depth to control how the decision trees are constructed.

We then combine the preprocessing step and the model training step into a pipeline using the Pipeline class from scikit-learn. This allows us to chain together multiple steps into a single object that can be fit to our training data and used to make predictions on our testing data.

After fitting the pipeline to the training data, we use it to make predictions on the testing data using the predict method. We then use two evaluation metrics, R2 score and mean absolute error (MAE), to measure the performance of our model on the testing data.

The R2 score measures how well our model is able to predict the variation in the target variable. It ranges from 0 to 1, with a higher score indicating a better fit. The MAE measures the average absolute difference between our predicted values and the actual values. A lower MAE indicates a better fit.

***PREDICTING THE PRICE OF A LAPTOP***

 Take user input

Company = input("Enter the company name: ")

Type\_name = input("Enter the type name: ")

Ram = int(input("Enter the RAM size in GB: "))

Weight = float(input("Enter the weight in kg: "))

Touchscreen = int(input("Does it have touchscreen? (0 or 1): "))

Ips = int(input("Does it have IPS screen? (0 or 1): "))

ppi = float(input("Enter the PPI value: "))

Cpu\_brand = input("Enter the CPU brand: ")

HDD = int(input("Enter the HDD size in GB (0 if not applicable): "))

SSD = int(input("Enter the SSD size in GB (0 if not applicable): "))

Gpu\_brand = input("Enter the GPU brand: ")

os = input("Enter the OS name: ")

# Create a dictionary with the input data

input\_dict = {

    'Company': [Company],

    'TypeName': [Type\_name],

    'Ram': [Ram],

    'Weight': [Weight],

    'Touchscreen': [Touchscreen],

    'Ips': [Ips],

    'ppi': [ppi],

    'Cpu brand': [Cpu\_brand],

    'HDD': [HDD],

    'SSD': [SSD],

    'Gpu brand': [Gpu\_brand],

    'os': [os]

}

# Convert the dictionary to a DataFrame

input\_df = pd.DataFrame(input\_dict)

# Make

predicted\_price = pipe.predict(input\_df)

predicted\_price = np.exp(predicted\_price)

# Print the predicted price

print('Predicted price:', predicted\_price[0])

This code allows the user to input various specifications of a laptop (such as company name, RAM size, weight, etc.) and stores the input data in a dictionary. The dictionary is then converted to a DataFrame, and this This code allows the user to input various specifications of a laptop (such as company name, RAM size, weight, etc.) and stores the input data in a dictionary. The dictionary is then converted to a DataFrame, and this DataFrame is used to predict the laptop price using a pre-trained machine learning model called pipe.

The predicted price is initially in log form, so we use np.exp() to convert it back to the original form. Finally, the predicted price is printed using print().

***OUTPUT :***

