**LAPTOP PRICE PREDICTION – FINANCIAL ANALYSIS**

**Overview**

*This project revolves around creating a web application focused on the prediction of the prices of the laptop. The model is used to calculate the ideal price of the laptop given the different input , its significance lies in assisting the people in understanding the price of the laptop on basis of the component they select and help them in making good investment.*

*The web application's functionality is straightforward: users input the* ***brand, type of the laptop, ram, weight, touchscreen, IPS, screen size, resolution, cpu, hdd, sdd, gpu, operating system*** *of the laptop they want to buy. In return, the application provides them with the potential estimated price of the laptop they are looking to buy. This empowers individual with valuable insights, enabling them to assess the risk and potential reward of their investment choices more effectively.*

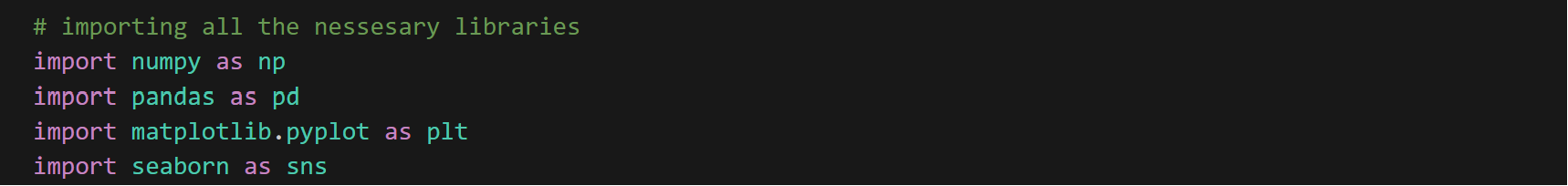
**Coding:**

**0. Doing the Essentials**

**(Laptop-price-predictor.ipynb)**

* 1. **Importing all the necessary libraries**

**Code:**



This section imports the necessary libraries for the project. Here's a list of what each library does:

* **numpy**: A Python library used for creating web applications with simple Python scripts.
* **matplotlib**: Matplotlib is a comprehensive 2D plotting library for Python, enabling the creation of high-quality visualizations with ease
* **seaborn**: is a statistical data visualization library based on Matplotlib, providing an aesthetically pleasing and informative interface for creating appealing statistical graphics in Python.
* **pandas**: A popular library for data manipulation and analysis.

**0.2 Importing the dataset**

**Code:**



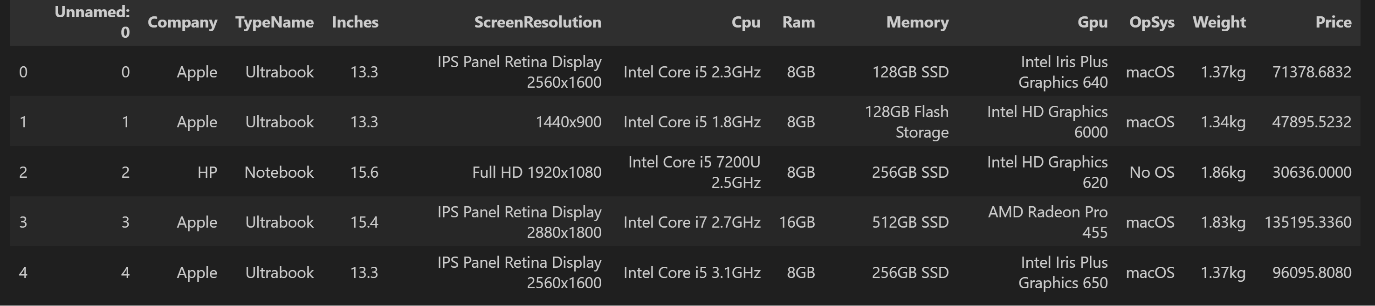
* This line of code uses the Pandas library in Python to read and import data from a CSV file named 'laptop\_data.csv' into a Data Frame called 'df.'.

**0.2 Overview of the Dataset**

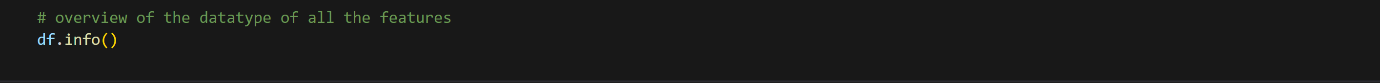
**Code:**   


* This code displays the first few rows of the DataFrame 'df' using the **'head**()' method, offering a quick preview of the dataset's structure and content.

**Output:**



**0.3 Overview of the datatype of all the features**

**Code:**  


* This code provides a concise summary of the **DataFrame** 'df' using the **'info()'** method, offering information on data types, non-null values, and memory usage, aiding in understanding the dataset's characteristics..

**0.4 Finding duplicate values in the dataset**

**Code:**



* This code calculates and returns the total number of duplicated rows in the DataFrame 'df' using the **'duplicated()'** method.
* Thankfully we do not have any duplicated values in our dataset

**0.5 Finding null values in the dataset**

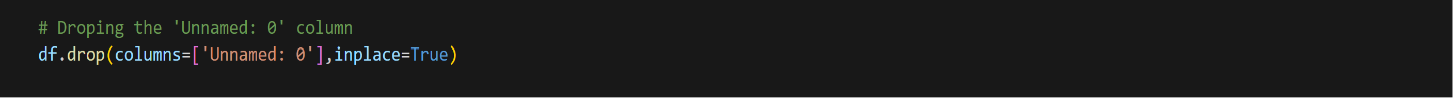
**Code:**



* This code computes and displays the sum of null (missing) values in each column of the **DataFrame 'df'** using the **'isnull()'** method..
* There is no missing value in our dataset

**0.6 Dropping the unnecessary column**

**Code:**



* This code removes the column named **'Unnamed: 0'** from the DataFrame 'df' in place, modifying the DataFrame directly.

**0.7 Removing the Units from the “Ram” and “Weight” column**

**Code:**



* This code removes the '**GB' and 'KG'** units from the **'Ram'** and **'Weight'** columns in the DataFrame 'df', respectively.

**0.8 Changing the datatype of “Ram” and “Weight” column**

**Code:**

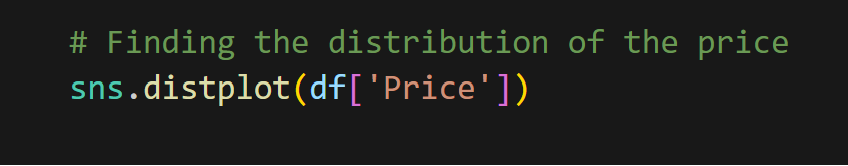


* This code converts the 'Ram' column in the DataFrame 'df' from object datatype to integer (int32) and the 'Weight' column from object datatype to floating-point (float32).

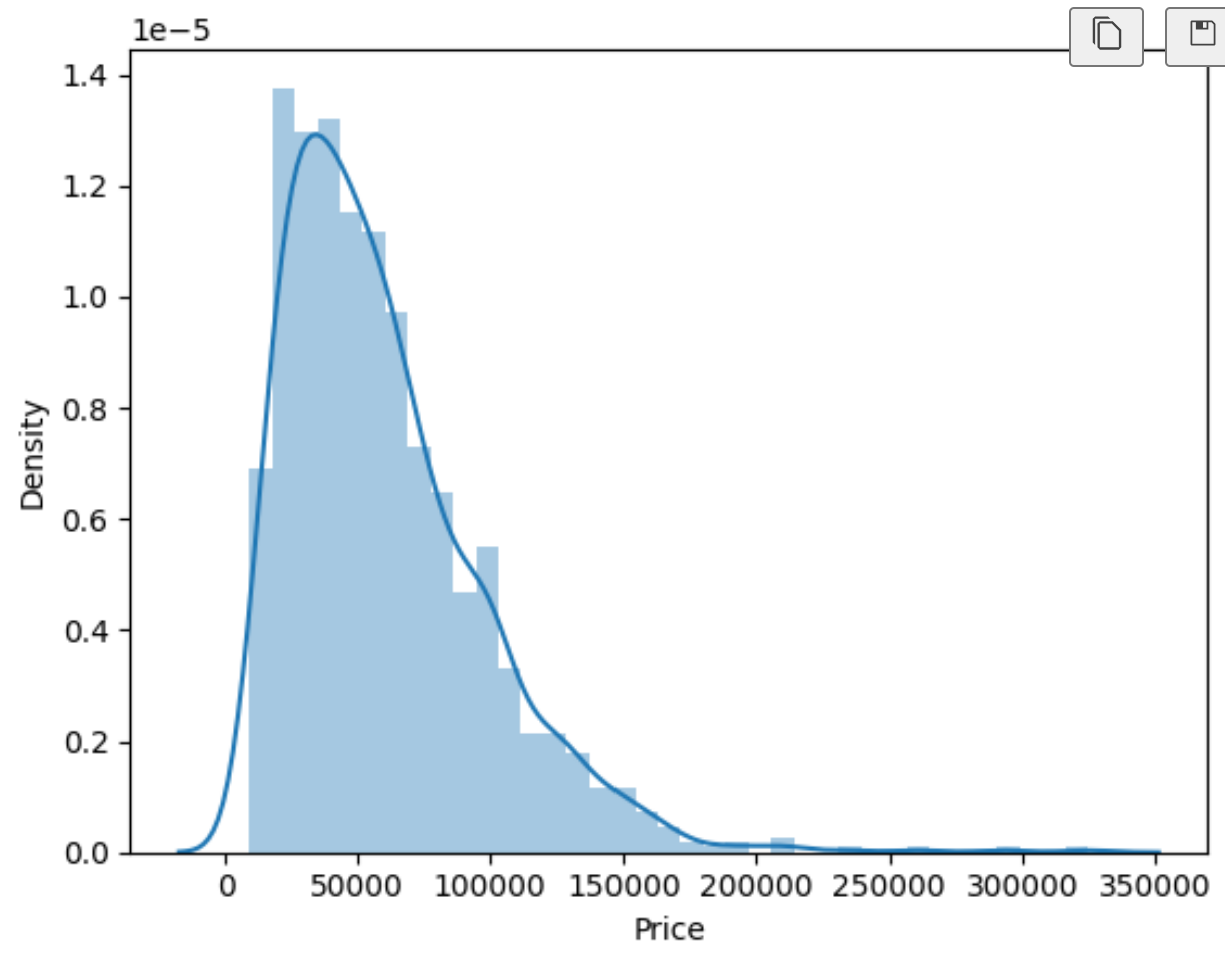
**1. Exploratory Data Analysis**

**1.1 Distribution of the price**

**Code**



* This code uses Seaborn to create a distribution plot **(histogram)** for the **'Price'** column in the DataFrame 'df', providing a visual representation of the data's distribution.

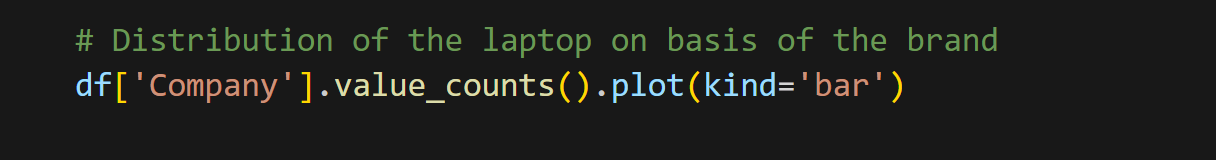


**Observation:**

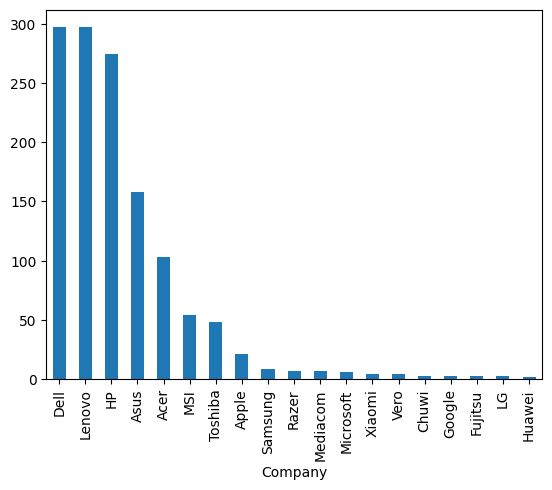
* laptop price estimates spanning from **Rs.9,270 to Rs.3,24,954**, with a notable concentration within the range of **Rs.30,000 to Rs.1,50,000**. This distribution is characterized by a right-skewed pattern, indicative of a prevalence of laptops within the specified moderate to high-value spectrum.

**1.2 Distribution of the laptop on basis of the brand**

**Code**



* This code generates a bar chart using Matplotlib, depicting the count of each unique value in the 'Company' column of the DataFrame 'df'

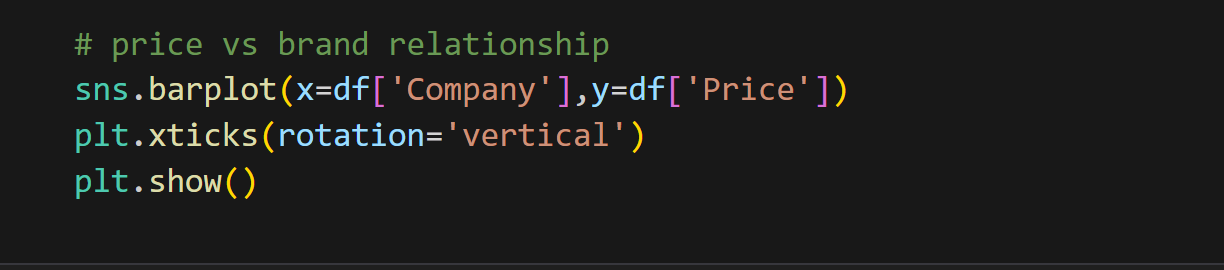


**Observation**

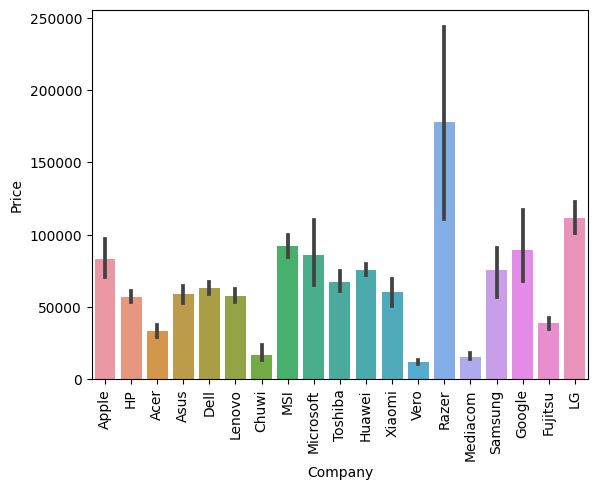
* The dataset prominently features three leading laptop brands: **Dell, Lenovo, and HP**, exhibiting the highest occurrence. Subsequently, **Asus, Acer, MSI, and Toshiba** follow suit in descending order, with counts surpassing 50. Beyond these major players, a multitude of other brands registers in the dataset, each with a count below 50, as illustrated in the bar chart where brand names are represented along the X-axis and the corresponding laptop counts are depicted along the Y-axis.

**1.3 Price vs brand relationship**

**Code**



* This code utilizes **Seaborn** to create a vertical bar plot depicting the average **'Price'** for each unique value in the **'Company'** column of the DataFrame 'df', with rotated x-axis labels for better readability.

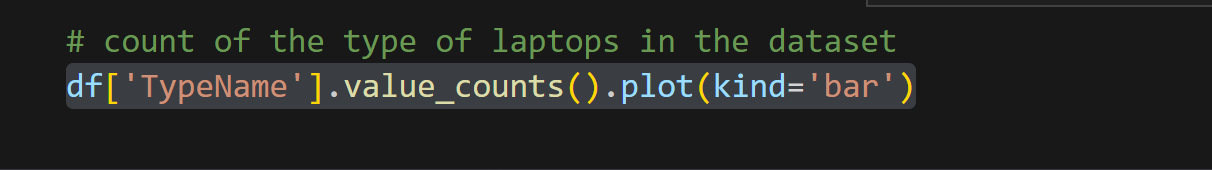


**Observation**

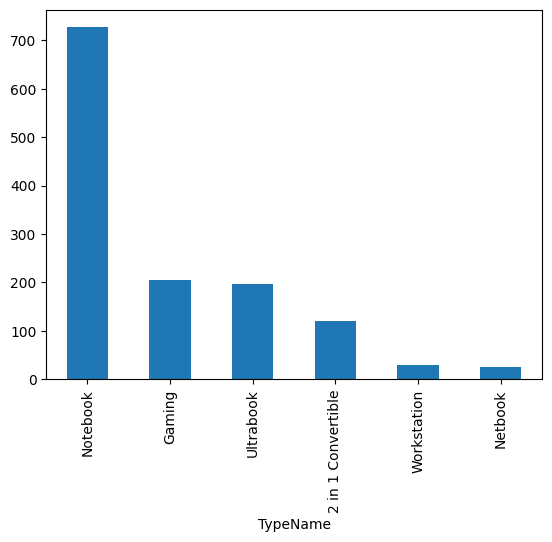
* Notably, **Apple MacBooks** consistently appear with prices below the **Rs.1,00,000** thresholds in the dataset. This revelation is surprising, considering the prevalent trend of Apple products in India typically falling within the upper price range, often exceeding **Rs.1,00,000**. However, it's important to clarify that the observed lower prices in the dataset are attributed to older models of MacBooks, which are reflective of historical pricing. In contrast, Razer laptops stand out with a notable prevalence of high-cost models, often associated with gaming configurations.
* Meanwhile, the trio of **Dell, Lenovo, and HP,** collectively constituting a substantial portion of sales, share a relatively similar pricing bracket. Other brands within the dataset fall within the spectrum, bridging the gap between the comparatively lower-priced **Apple MacBooks** and the **higher-end Razer laptops**, forming a diverse range.

**1.4 Count of the type of laptops**

**Code**



* This code generates a bar chart using Matplotlib, illustrating the count of each unique value in the 'TypeName' column of the DataFrame 'df'.

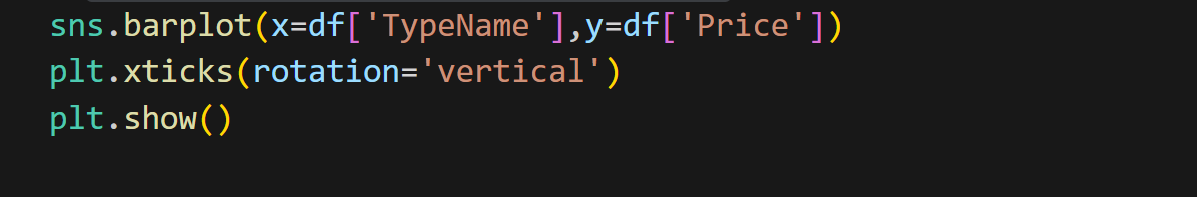


**Observation**

* The dataset predominantly features over 700 instances of "Notebook" laptops, surpassing "Gaming" and "Ultrabook" categories, each with around 200 occurrences. A notable contrast exists, particularly with a 500-instance difference between "Notebook" and "Gaming"/"Ultrabook," highlighting significant user preferences.
* Conversely, "2-in-1 Convertible," "Workstation," and "Netbook" exhibit lower counts, with the latter two falling below 100 occurrences. The "2-in-1 Convertible" category, though less frequent than "Notebook," still stands at approximately 100 instances, suggesting a moderate presence.

**1.5 Prices of the different type laptop**

**Code**



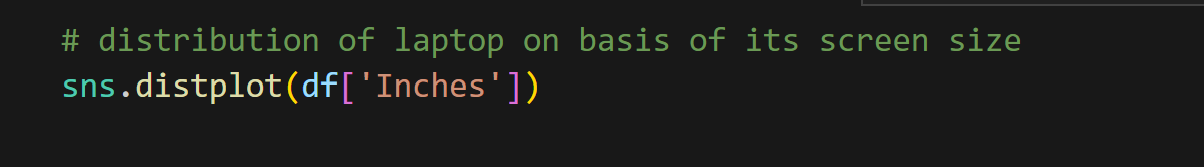
* This code employs Seaborn to create a vertical bar plot, showcasing the average 'Price' for each unique value in the 'TypeName' column of the DataFrame 'df'. The x-axis labels are rotated vertically for improved readability.

|  |  |
| --- | --- |
|  | **Observation:**   * In terms of pricing, Ultrabooks and Gaming laptops exhibit a close resemblance, with Ultrabooks priced at approximately Rs. 80,000 and Gaming laptops hovering around Rs. 85,000. |

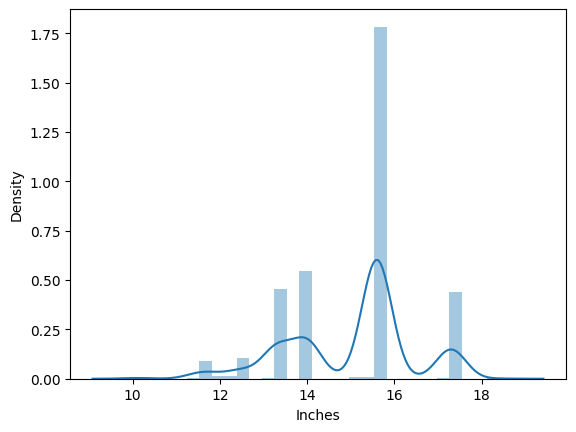
* Notably, the Workstation category stands out as the most expensive, commanding a premium price point. The 2-in-1 Convertible laptops are positioned around Rs. 60,000, while Notebooks and Netbooks emerge as the more budget-friendly options, ranging from Rs. 30,000 to Rs. 40,000, with Netbooks representing the lower end of this spectrum. This diversity in pricing underscores the wide range of options available in the laptop market, catering to varying budget considerations and preferences.
* The error bars also show that some types of laptops have more consistent prices than others. For example, the error bar for the Ultrabook laptop is relatively small, which means that most Ultrabook laptops have similar prices. On the other hand, the error bar for the Workstation laptop is relatively large, which means that there is a wide range of prices for Workstation laptops. This could indicate that some types of laptops are more standardized than others, or that some types of laptops have more diverse options and preferences than others.

**1.6 distribution of laptop on basis of its screen size**

**Code**

****

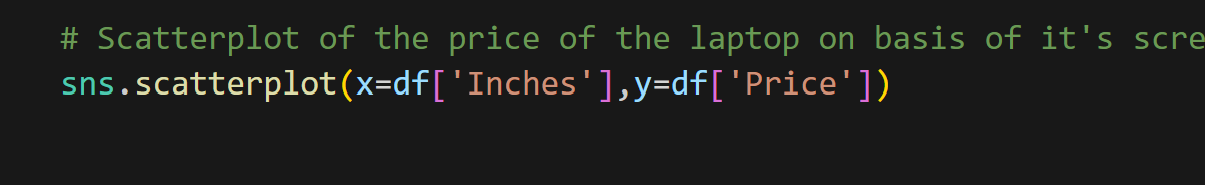
* This code utilizes Seaborn to create a distribution plot (histogram) for the 'Inches' column in the DataFrame 'df', offering insights into the distribution of laptop screen sizes



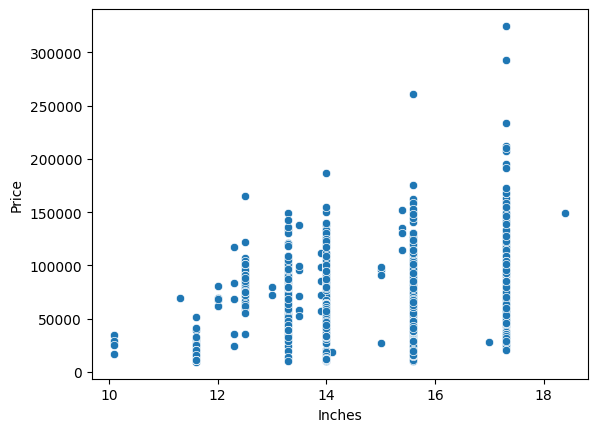
**Observation:**

* The graph shows the relationship between the density of the laptops and its length in inches. The density is measured in units of mass per unit volume, such as grams per cubic centimetre.
* The graph has a bell-shaped curve, which means that the density follows a normal distribution. This means that most of the values are clustered around the mean or average, and the values become less frequent as they deviate from the mean.
* The graph has a peak at around **16 inches**, which means that this is the mode or the most common value of the screen size. The density at this point is about **1.5.**
* The graph has a smaller peak at around **14 inches**, which means that this is another common value of the screen size, but less frequent than 16 inches. The density at this point is about **1.25.**
* The graph has another peak around **17 inches**, which means that this is another common value of the screen size. The density of this point is about **0.20**

**1.7 Scatterplot of the price of the laptop on basis of its screen size**



* This illustrating the relationship between 'Inches' (laptop screen size) and code employs Seaborn to generate a scatter plot, 'Price' in the DataFrame 'df'.



**Observation:**

* The dataset reveals a predominant clustering of laptop screen sizes within the range of **13 inches to 17 inches**, indicating a commonality in the dimensions of most laptops. However, noteworthy outliers exist, particularly among laptops featuring a **17-inch** screen size.
* These specific outliers are associated with significantly higher costs, suggesting that the larger screen size contributes to an elevated price range. The presence of such outliers emphasizes the diversity within the dataset, with certain laptops deviating from the typical screen size and pricing patterns.

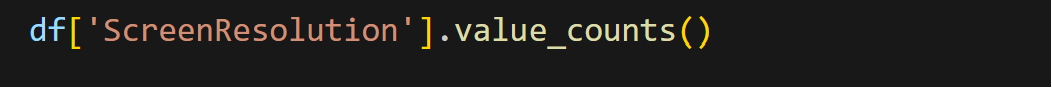
**2. Fixing the Necessary columns for the prediction**

**2.1 Fixing the “Screen Resolution” columns**

**Overview:**

* *Before we start analysing the data, it's important to understand why we're taking a specific approach. In this dataset, key information, like IPS display, touchscreen, and screen resolution, is all packed into one column. To simplify our analysis, we want to calculate the Pixel Per Inch (PPI) metric. This allows us to combine these various screen features into one standardized value, which we can use in our predictive model. Using PPI helps us take a more thorough and precise approach to predicting laptop prices.*

**Code:**

****

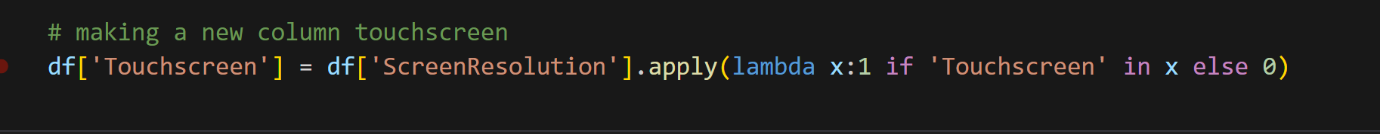
* This code calculates and displays the count of each unique value in the **'ScreenResolution'** column of the **DataFrame** **'df'**, providing an overview of the distribution of screen resolutions in the dataset.

**Output:**

|  |  |
| --- | --- |
|  | **Observation:**   * The 'resolution' column contains a wealth of information. We can extract whether the laptop has an IPS display and determine if it's a touchscreen device. |

**2.1.1 Creating a “Touchscreen” column from “Screen Resolution” Column**

**Code:**



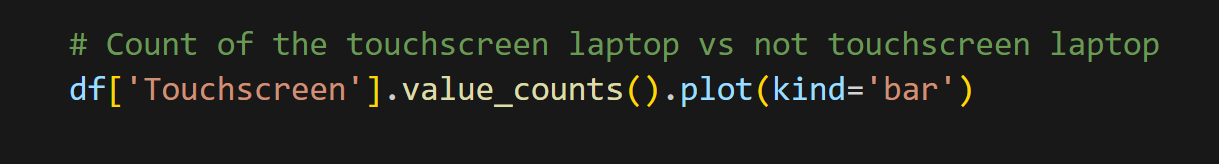
* This code creates a new **'Touchscreen'** column in the DataFrame **'df**,' assigning a value of 1 if the corresponding **'Screen Resolution'** contains 'Touchscreen,' and 0 otherwise, utilizing a lambda function with the 'apply' method.
* We've introduced a novel column termed **'Touchscreen'** in the dataset. The values within this column are designated as **'1'** to denote laptops equipped with touchscreen functionality and **'0'** to signify laptops lacking touchscreen features.
* This strategic categorization allows for a clear and binary representation, enhancing the dataset's capacity to discern and analyse touchscreen attributes within the broader context of laptop characteristics.

**Output:**

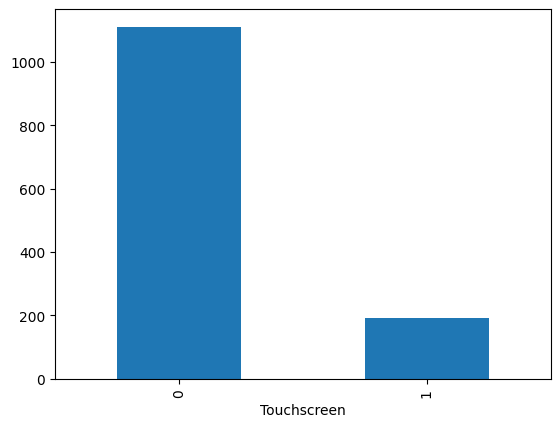
|  |  |
| --- | --- |
|  | **Observation:**   * A new column called "Touchscreen" has been introduced. For rows where the value is 1, it indicates that the laptop is a touchscreen device, while a value of 0 signifies the absence of touchscreen functionality |

**2.1.1 Count of the touchscreen laptop vs not touchscreen laptop**

**Code:**



* This code generates a bar chart using Matplotlib, illustrating the count of laptops with and without touchscreen functionality in the **'Touchscreen'** column of the **DataFrame** 'df'.

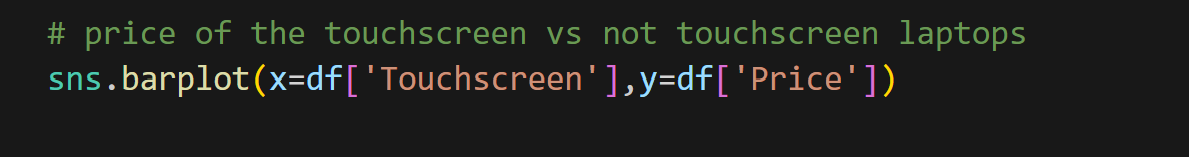


**Observation:**

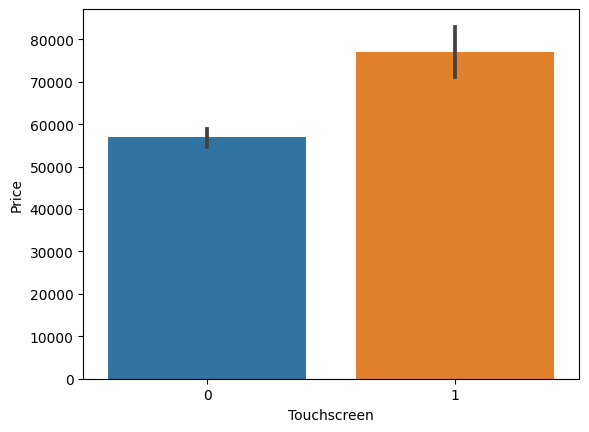
* A substantial proportion of laptops, totalling around **1000 units**, do not feature touchscreen functionality, while a more modest count of **200 laptops** incorporates this touch-enabled capability.
* This distribution underscores the prevalence of non-touchscreen laptops in the dataset, highlighting the varying technological specifications within the sampled devices.

**2.1.2 Price of the touchscreen vs not touchscreen laptops**

**Code**



* This code utilizes Seaborn to create a bar plot, showcasing the relationship between touchscreen **presence (x-axis)** and corresponding **laptop prices (y-axis)** in the Data Frame 'df'.

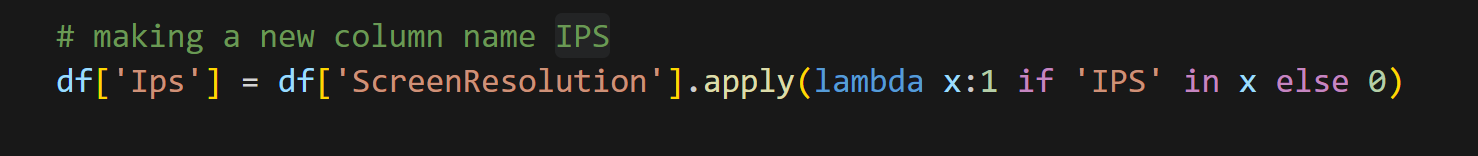


**Observation:**

* The dataset reveals a notable price disparity between touchscreen and non-touchscreen laptops. Touchscreen laptops, albeit in lower numbers, command a higher average price of approximately **Rs. 75,000**. In contrast, non-touchscreen laptops, more abundant in the dataset, exhibit a comparatively lower average price, hovering around **Rs. 58,000**. The limited representation of touchscreen laptops contributes to a substantial variance in pricing, underlining the significance of touchscreen functionality as a determining factor in laptop costs.
* the error bars also show the **"non-touchscreen laptop"** have more consistent prices than the **"touchscreen laptop"**. For example, the error bar for the "non-touchscreen laptop" is relatively small, which means that most "non-touchscreen laptop" laptops have similar prices.
* On the other hand, the error bar for the **"touchscreen laptop"** is relatively large, which means that there is a wide range of prices for **"touchscreen laptop".**

**2.1.3 Making a new column name IPS**

**Code - 1**

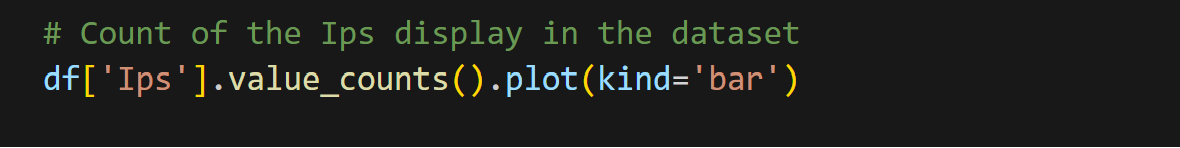


* This code adds a new column named 'Ips' to the **DataFrame** **'df**,' assigning a value of 1 if the **'ScreenResolution'** contains **'IPS'** and 0 otherwise, utilizing a lambda function with the 'apply' method.

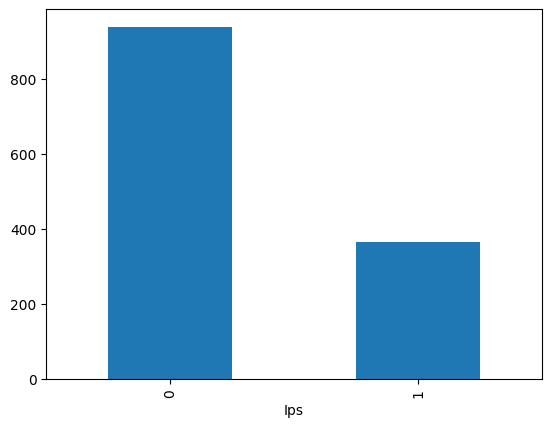
**Output:**

|  |  |
| --- | --- |
|  | **Observation:**   * We've introduced a novel column termed **'IPS'** in the dataset. The values within this column are designated as **'1'** to denote laptops equipped with IPS functionality and **'0'** to signify laptops lacking IPS features. * This strategic categorization allows for a clear and binary representation, enhancing the dataset's capacity to discern and analyse IPS attributes within the broader context of laptop characteristics. |

**Code - 2**



* This code generates a bar chart using Matplotlib to visually represent the count of laptops with and without **IPS (In-Plane Switching)** technology in the **'Ips'** column of the DataFrame 'df'.

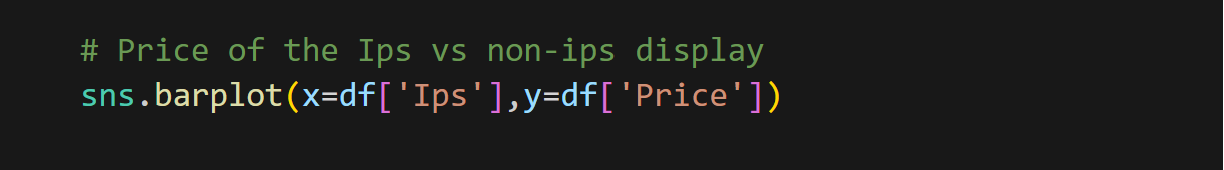


**Observation**

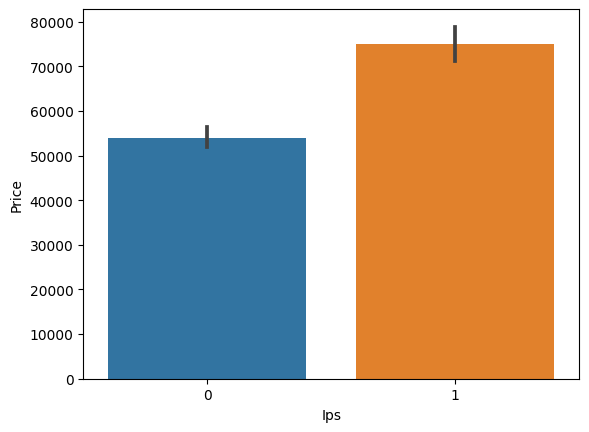
* There are more than **800** **laptop** which do not have the 'Ips' feature in it where around **400** **laptops** have **'Ips'** feature in it

**2.1.4 Price of the Ips vs non-Ips display**

**Code:**



* This Seaborn bar plot illustrates the relationship between the presence or absence of **IPS** (In-Plane Switching) **technology** **(x-axis)** and **laptop prices** **(y-axis)** in the DataFrame 'df'.

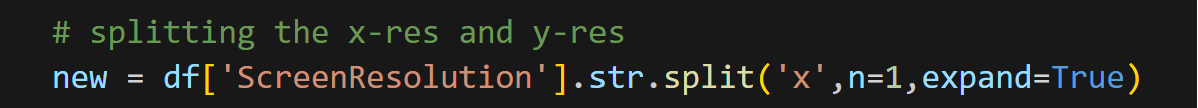


**Observation:**

* The price of the **"non-ips"** display are less around **Rs.55,000** whereas **'Ips'** display cost more around **Rs.70,000**.
* The error bars also show the **" non-ips laptop"** have more consistent prices than the "ips laptop". For example, the error bar for the **"non-ips"** is relatively small, which means that most **"non-ips"** laptops have similar prices.
* On the other hand, the error bar for the **"ips"** is relatively large, which means that there is a wide range of prices for **"touchscreen laptop"**.

**2.1.5 Splitting the Screen Resolution column into two other column**

**Code:**

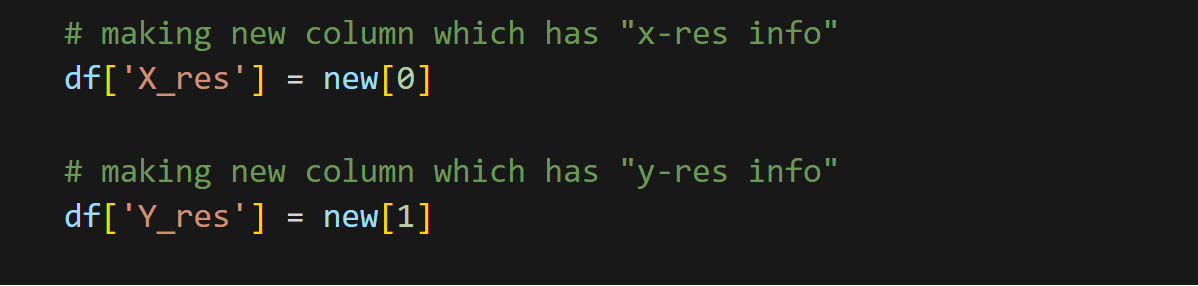


* This code splits the **'ScreenResolution'** column in the DataFrame 'df' at the first occurrence of **'x'**, creating a new DataFrame 'new' with two columns, utilizing the **'str.split'** method with **'n=1'** and **'expand=True'.**

**Note:**

* We are splitting the x-res and y-res column to create a new column dpi with the help of the values of x-res and y-res

**Code:**



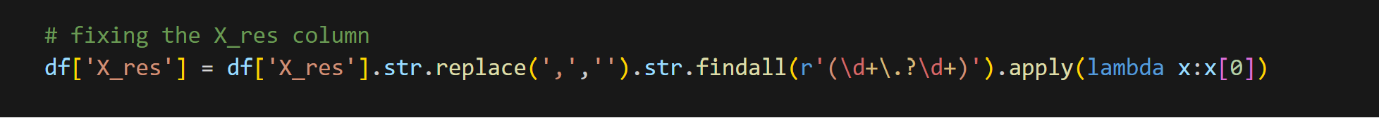
* This code creates two new columns, **'X\_res'** and **'Y\_res,'** in the DataFrame 'df' to store information extracted from splitting the 'Screen Resolution' column at the **'x'** delimiter, representing the horizontal and vertical screen resolutions, respectively.

**Output:**

|  |  |
| --- | --- |
|  | * Although we've successfully created the **'X\_res'** column, it appears to retain some attached features or unwanted elements. * Now, we will try to separate the “feature” from the **‘x\_res’** * In short, we are trying to remove it as we have already made new columns for these features like **“IPS”** and **“Touchscreen”**. |

**2.1.6 Fixing the X\_res column**

**Code:**

****

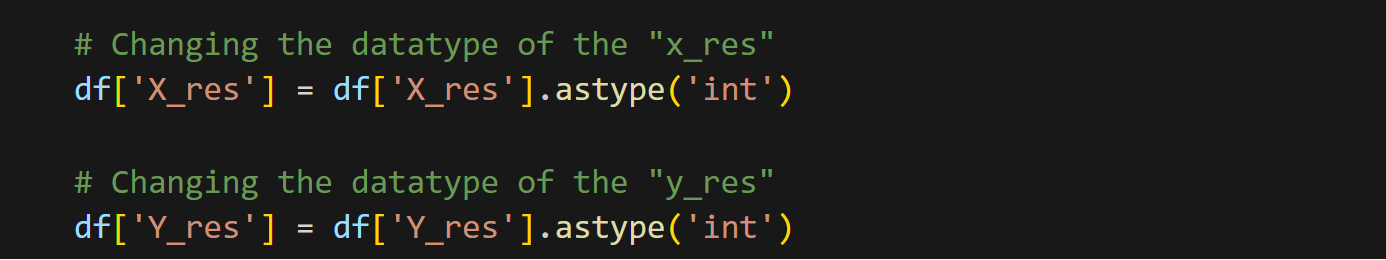
* This code corrects the **'X\_res'** column in the DataFrame 'df' by removing commas and extracting the first numerical occurrence using regular expressions, ensuring it contains only numeric values

**Output:**

|  |  |
| --- | --- |
|  | * Now the **‘X\_res’** is fixed it only have numeric values now, now we can use it to calculate the **Pixel Per Inch (PPI)** * **Pixel Per Inch (PPI)** is a metric that measures the pixel density on a display screen. It represents the number of pixels present per inch of the screen's surface. The concept is particularly relevant when assessing the sharpness and clarity of images and text on electronic devices like smartphones, tablets, monitors, and laptops. * A higher PPI indicates a greater concentration of pixels within each inch, resulting in sharper and more detailed visuals. PPI is calculated by dividing the diagonal resolution of a screen **(in pixels)** by the diagonal size of the screen **(in inches).** * In practical terms, a higher PPI is often associated with crisper images and text, contributing to an improved viewing experience, especially in devices with smaller screens where individual pixels may be more noticeable at lower pixel densities. |

**2.1.6 Changing the datatype of “X\_res” & “Y\_res”**

**Code:**

****

* The provided code is changing the data type of two columns, "**X\_res**" and "**Y\_res**," in a DataFrame named 'df' to integers. This conversion is likely performed to ensure that the resolution values stored in these columns are treated as whole numbers rather than strings or floats, facilitating numerical operations and analysis

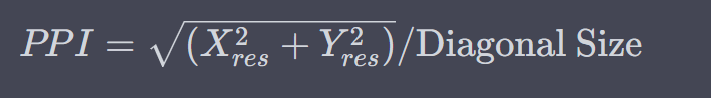
**Output:**

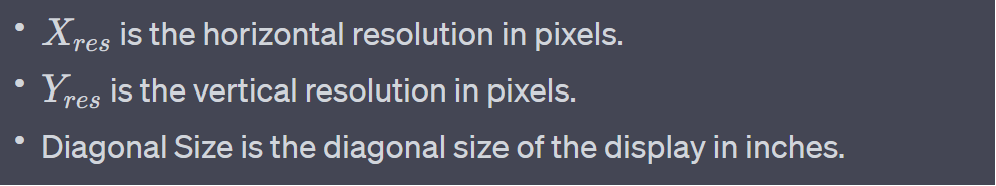
|  |  |
| --- | --- |
|  |  |

**2.1.6 Calculating the PPI**

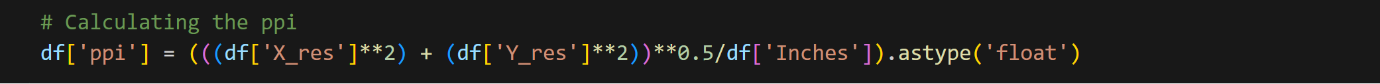
**Overview:**

The Pixels Per Inch (PPI) can be calculated using the following formula:



****

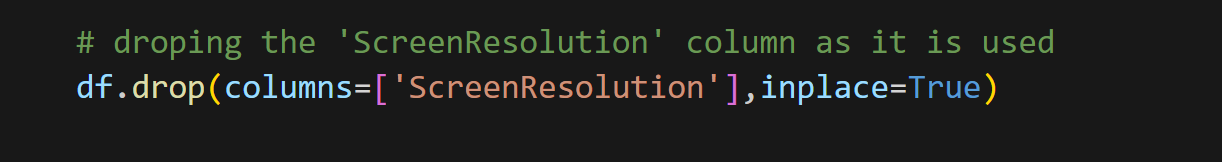
**Code:**

****

* This code computes the **Pixels Per Inch (PPI)** for each laptop in a DataFrame by applying the PPI formula, considering horizontal and vertical resolutions, and the diagonal size of the display. The result is stored in a new column **'ppi**,' with data type explicitly set to float for accuracy.

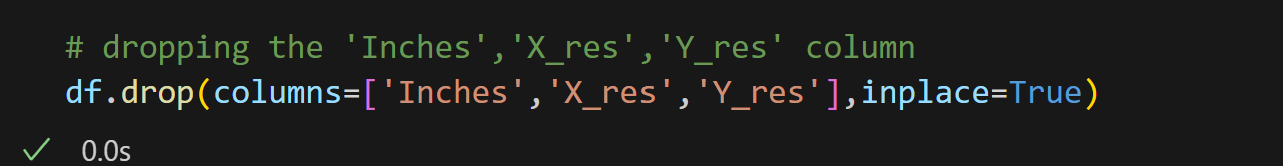
**2.1.7 Dropping all the unwanted columns**

**Code:**



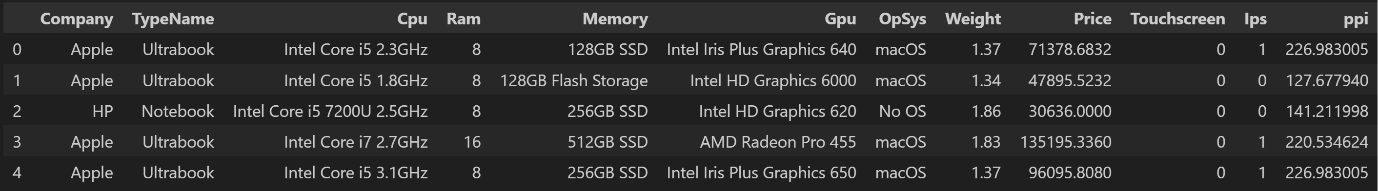
* This code removes the **'ScreenResolution'** column from the DataFrame 'df' to eliminate redundancy and streamline the dataset, using the **'drop'** method with **'inplace=True'**.
* We have extracted all the necessary information from the column and made other new columns therefore we will simply drop this column

**Code:**



* This line of code in a DataFrame named 'df' removes three columns’ 'Inches', **'X\_res'**, and **'Y\_res'** effectively eliminating these columns from the dataset. The removal is done in place, meaning it modifies the DataFrame directly. This operation might be performed when these columns are no longer needed for analysis or to simplify the dataset structure.

**Output**:

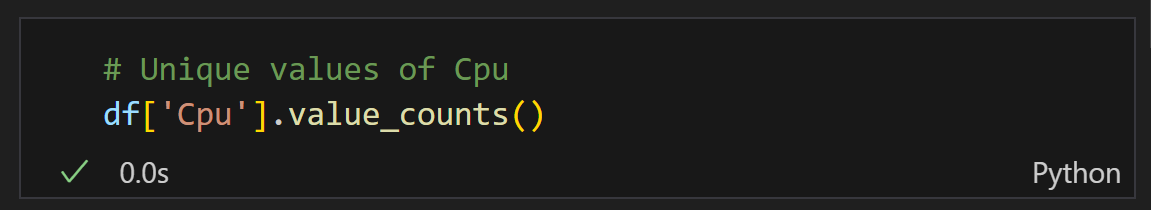


* We're getting rid of the 'Inches', 'X\_res', and 'Y\_res' columns in our dataset. We've already used these columns to calculate DPI, and now that we have that information, we don't need these columns cluttering our dataset anymore

**2.2 Fixing the “CPU” columns**

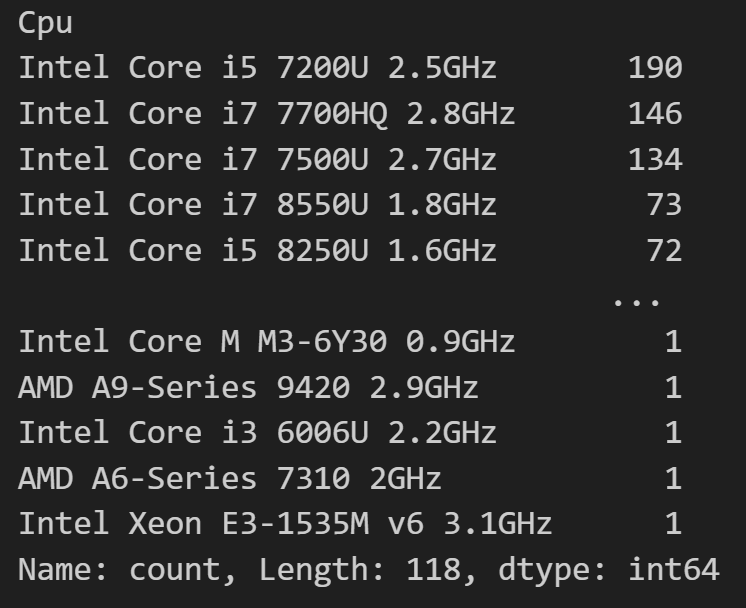
**2.2.1 Unique values of the cpu**

**Code:**

****

* This line of code counts the occurrences of each unique value in the 'Cpu' column of the DataFrame 'df'.

**Output:**

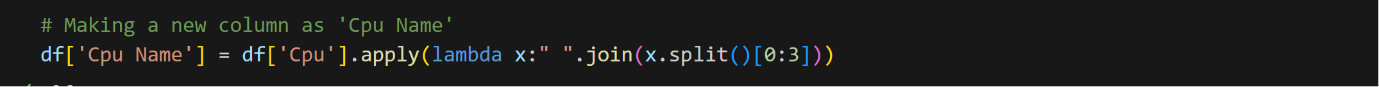
****

**Observation:**

* There are many different kinds of processors, but some of them are not used very often. To make things simpler, we'll group these less common processors together and call them "**Other Processors**." This helps us organize and understand the variety of processors more easily.

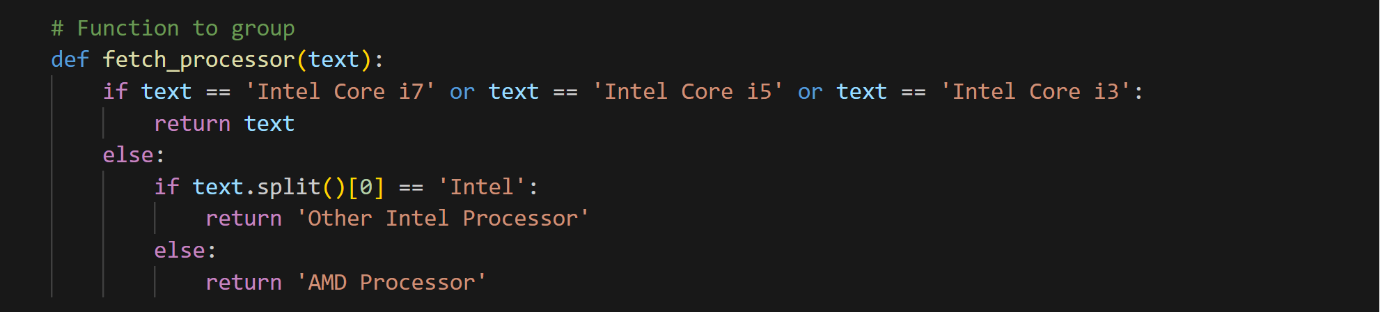
**2.2.2 Making a new column as “Cpu Name”**

**Code:**

****

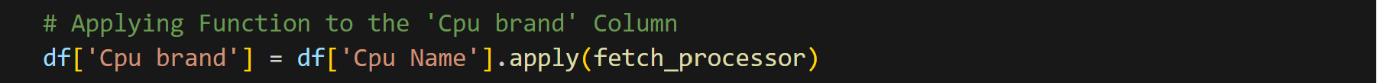
* This code creates a new column 'Cpu Name' in the DataFrame 'df', extracting the first three words from each entry in the 'Cpu' column and joining them together with a space.

**Code:**

****

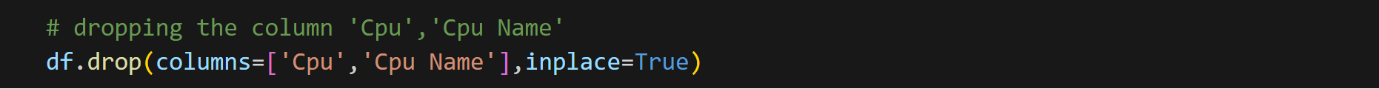
* This function, named **fetch\_processor**, categorizes processors into groups. If the processor is identified as **'Intel Core i7,' 'Intel Core i5,' or 'Intel Core i3,'** it is assigned to its respective category. Otherwise, if the processor is from Intel but not in the specified categories, it falls under 'Other Intel Processor.' If the processor is from AMD, it is grouped as an **'AMD Processor'**. This function is designed to classify and simplify processor types for analysis.

**Code:**



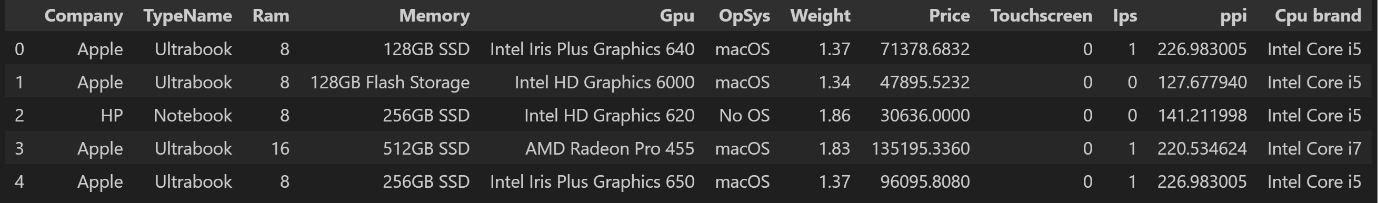
* This line of code applies the **fetch\_processor** function to the **'Cpu Name'** column in the DataFrame 'df'. The function categorizes processor types based on specific criteria, and the results are stored in a new column named **'Cpu brand**.' This process simplifies and groups processor brands for further analysis.

**Code:**

****

* This line of code removes the **'Cpu'** and **'Cpu Name'** columns from the DataFrame 'df'. These columns were dropped as they are no longer needed, having been replaced or consolidated during the data processing steps. The **'inplace=True'** parameter ensures that the DataFrame is modified directly.

**Output:**

****

**Observation:**

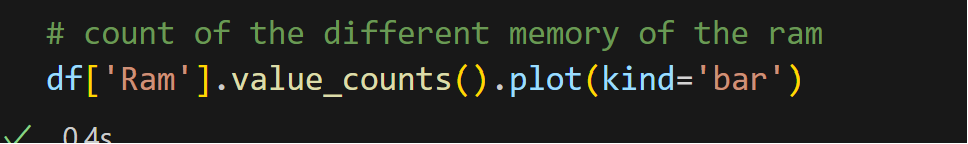
We removed the '**Cpu**' and **'Cpu Name'** columns and introduced a new column, **'Cpu brand**' to simplify our analysis. This helps manage the small dataset better by reducing the number of unique values, which could otherwise lead to increased variance.

**2.3 Exploring the “Ram” columns**

**Note:**

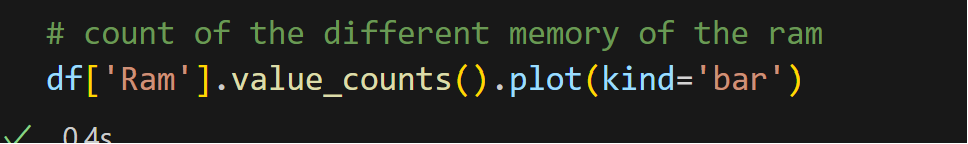
* *There is nothing to fix in this column or any information to extract we will just analyse it a bit*

**2.3.1 count of the different memory of the ram**

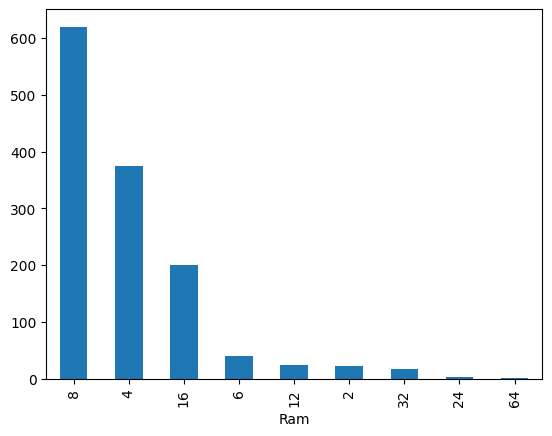
****

* This code generates a bar chart representing the count of different RAM memory sizes in the 'Ram' column of the DataFrame 'df'. Each bar corresponds to a specific RAM size category, providing a visual representation of the distribution of RAM sizes in the dataset.

**2.3.1 count of the different memory of the ram**

****

* This line of code creates a bar chart displaying the count of each unique RAM size in the 'Ram' column of the DataFrame 'df'.

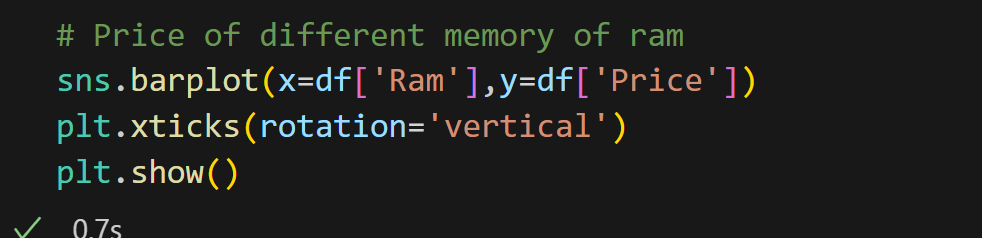
****

**Observation:**

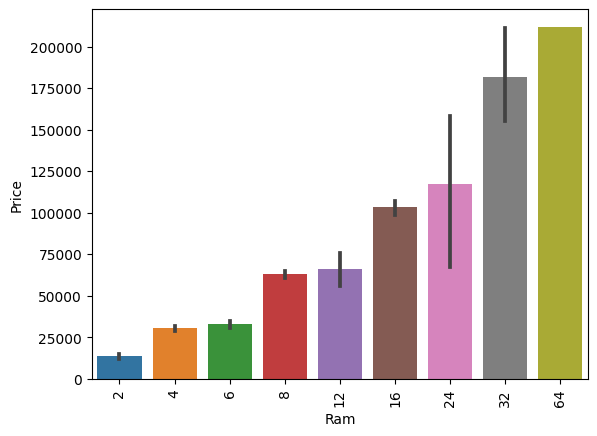
* The majority of laptops come equipped with **8GB RAM**, considered a sweet spot that allows for potential future upgrades. Surprisingly, laptops featuring **4GB RAM** outnumber those with **16GB**. Specifically, there are approximately 380 laptops with **4GB RAM**, while the **16GB** category includes around **200 laptops**. The remaining RAM configurations, including those under **100 in count**, seem to be less common among the available laptops.

**2.3.2 Price of different memory of ram**

**Code:**

****

* This code utilizes Seaborn to create a bar plot where the x-axis represents different RAM sizes from the **'Ram'** column, and the y-axis corresponds to the corresponding prices from the 'Price' column in the DataFrame 'df'. The **plt.xticks(rotation='vertical'**) line rotates the x-axis labels for better readability, and **plt.show()** displays the plot. The visualization provides insights into the relationship between RAM size and laptop prices.

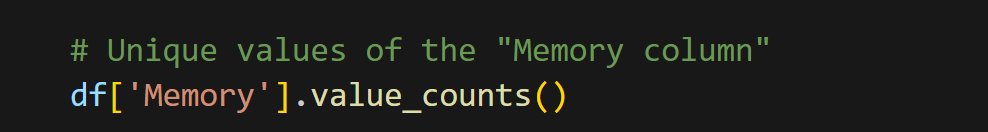


Observation:

* As anticipated, the price of RAM exhibits a direct correlation with the memory size of the RAM module. In simpler terms, larger RAM sizes tend to come with higher prices, aligning with the expectation that greater memory capacity typically incurs a higher cost.

**2.4 Fixing the “Memory” column**

**2.4.1 Unique values of the “Memory column”**

****

* This code counts the occurrences of each unique value in the 'Memory' column of the DataFrame 'df'. It provides a summary of the distribution of different memory configurations, showcasing the count for each distinct value in the 'Memory' column.

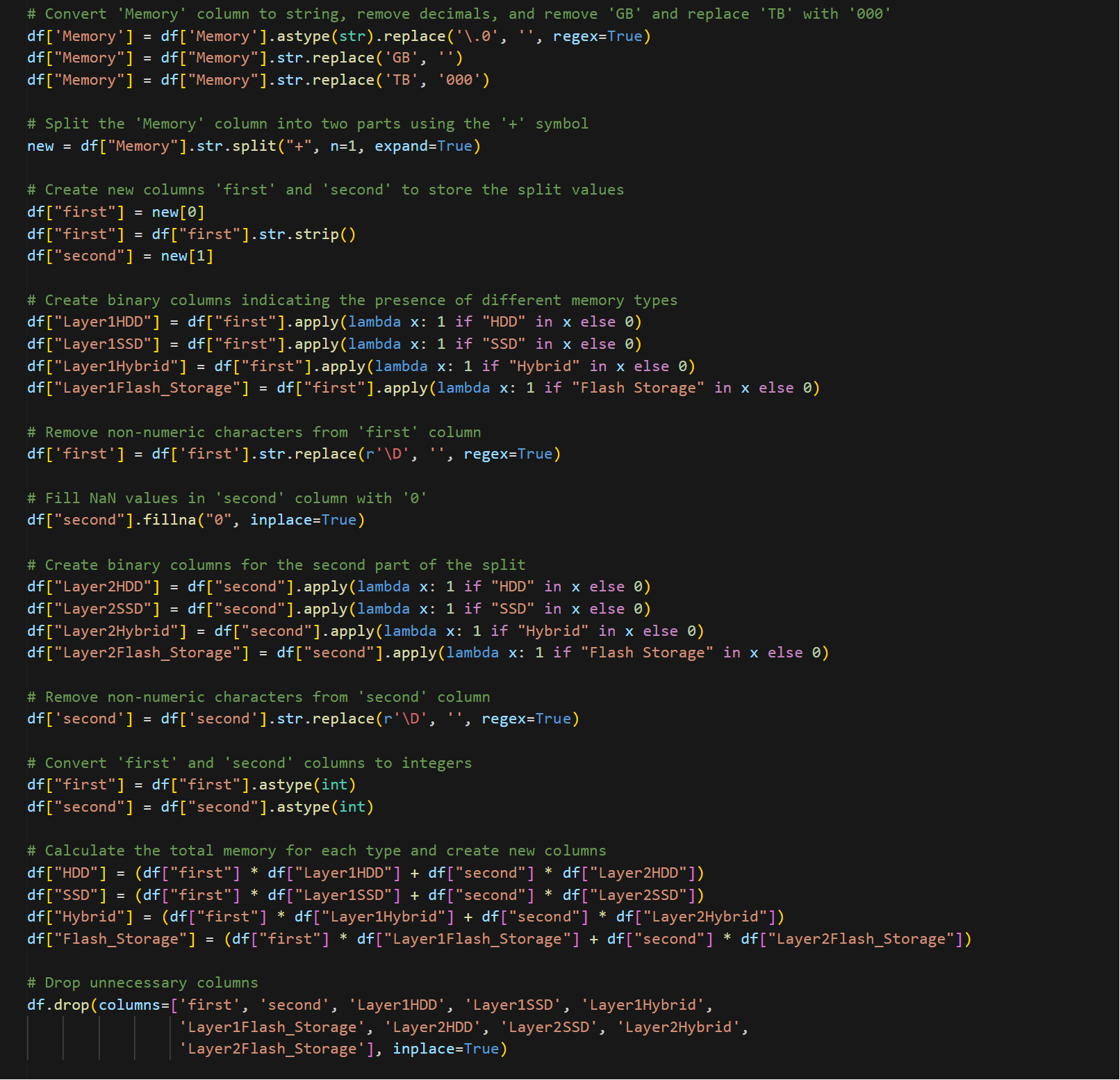
**Output:**

|  |  |
| --- | --- |
|  | **Observation:**   * The column contains a wealth of information regarding the type of memory a laptop possesses, indicating whether it's an SSD or HDD. It also provides details on the memory size. In cases where a laptop has both HDD and SSD, signifying a hybrid setup, this information is also included. |

* So, to break it down, we're planning to create several new columns based on the existing column. These new columns will include ones for HDD, SSD, Flash Drive, and Hybrid configurations. This way, we can organize and categorize the information more effectively.

**2.4.2 Data preprocessing “Memory column”**

**Code:**

****

1. **Cleaning Memory Data:**

* Converts 'Memory' column to string, removes decimals, and replaces 'GB' with an empty string and 'TB' with '000'.

1. **Splitting Memory Information:**

* Splits 'Memory' into two parts using the '+' symbol and creates new columns 'first' and 'second' to store these parts.

1. **Creating Binary Columns:**

* Generates binary columns indicating the presence of different memory types (HDD, SSD, Hybrid, Flash Storage) in both 'first' and 'second'.

1. **Numeric Conversion:**

* Removes non-numeric characters from 'first' and 'second' columns and fills NaN values in 'second' with '0'.

1. **Calculating Total Memory:**

* Computes total memory for each type (HDD, SSD, Hybrid, Flash Storage) based on the split values and binary indicators.

1. **Final Cleanup:**

* Drops unnecessary columns related to the intermediate processing steps, leaving the DataFrame with new columns representing different memory types and their respective totals.

**Output:**

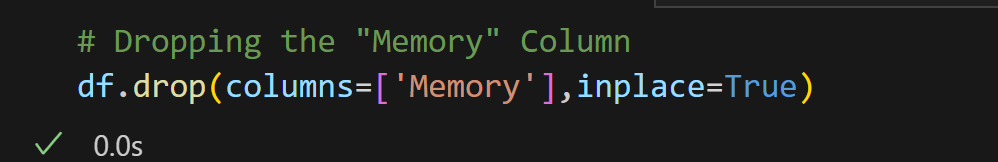
****

**Observation:**

* We've generated four new columns, but considering that Hybrid and Flash Storage features are relatively uncommon, especially in older laptops, we're opting to drop these columns. This simplification helps streamline the data and focuses on more prevalent and relevant memory types

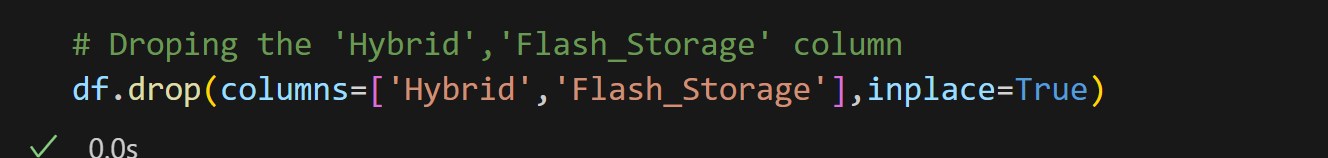
**2.4.3 Dropping the unnecessary Columns**

**Code:**



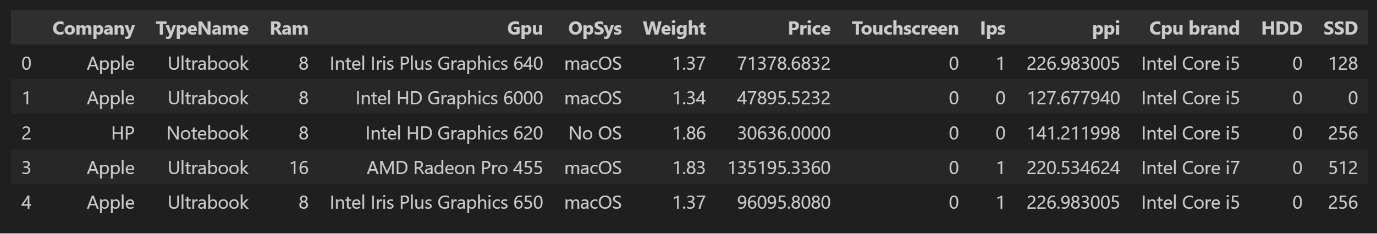
* This line of code removes the original 'Memory' column from the DataFrame 'df' after the necessary processing steps. The 'Memory' column is dropped as it has been replaced by new columns representing different memory types and their totals, providing a more structured and simplified dataset for analysis

**Code:**

****

* This code drops the **'Hybrid'** and **'Flash\_Storage'** columns from the DataFrame 'df'. These columns are removed, likely because they are no longer needed for the analysis, simplifying the dataset further.

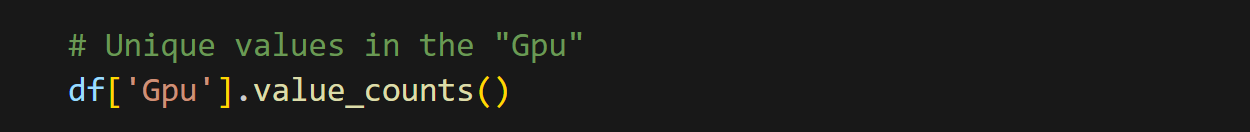
**Output:**

****

**2.5 Fixing the “Gpu” column**

**2.5.1 Dropping the unnecessary Columns**

**Code:**

****

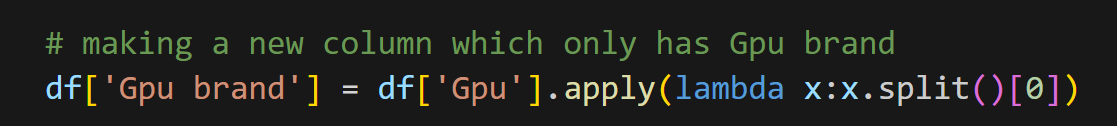
* This line of code provides the count of each unique value in the 'Gpu' column of the DataFrame 'df', revealing the distribution of different GPU types

**Output:**

|  |  |
| --- | --- |
|  | **Observation:**   * Since there's a variety of GPUs with limited data for each type, predicting accurately might be challenging. To simplify, we plan to categorize the graphics cards based on their brands, such as "Nvidia," "AMD," and "Intel." This segmentation will help focus our analysis on broader GPU categories, potentially improving prediction reliability. |

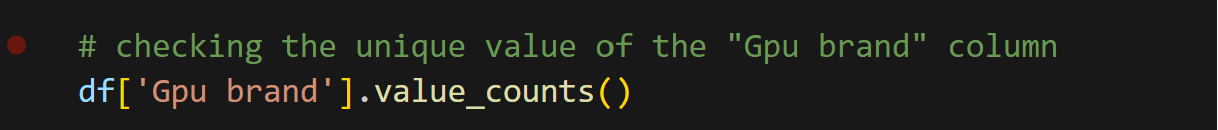
**2.5.2 Making a new column which only has Gpu brand**

**Code:**

****

* This code creates a new column named **'Gpu brand'** in the DataFrame 'df'. It extracts the brand name of the GPU from the **'Gpu'** column by applying a lambda function that splits the **'Gpu'** entry into words and selects the first word (the brand name).

**Code:**

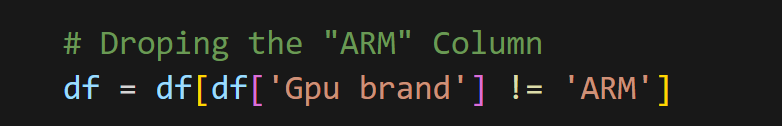
****

**Output:**

|  |  |
| --- | --- |
|  | **Observation:**   * Since the brand "ARM" appears in just one laptop, it's a rare occurrence. To keep our analysis focused and avoid potential outliers, we've decided to drop this specific brand from our dataset. |

**2.5.3 Dropping the unnecessary Columns**

**Code:**

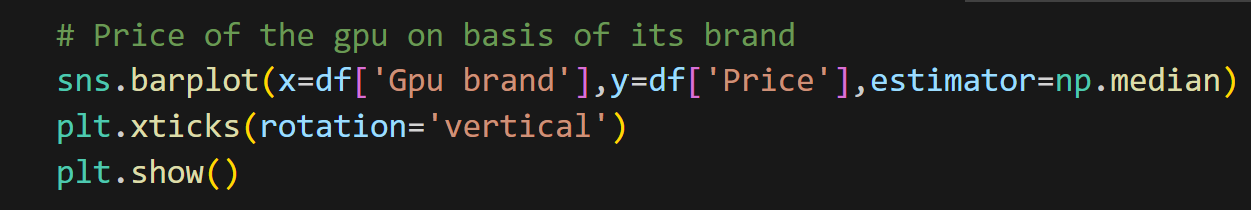
****

**Output:**

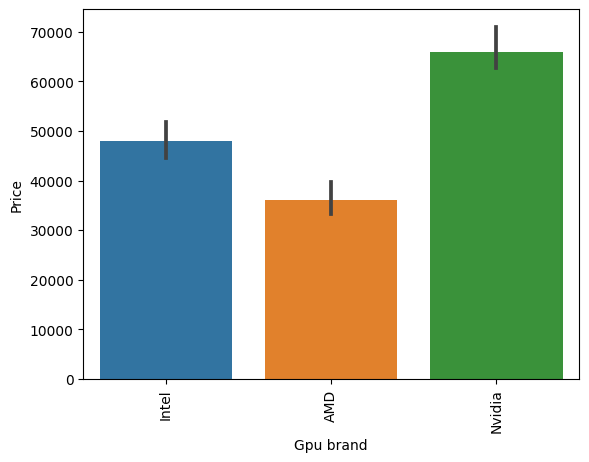
|  |  |
| --- | --- |
|  | **Observation:**   * Now the dataset has only three Gpu brand |

**2.5.4 Price of the Gpu on basis of its brand**

**Code:**

****

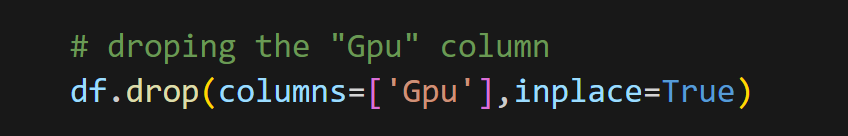
* The code uses Seaborn to plot the median prices of different GPU brands (from the 'Gpu brand' column) against the x-axis. Rotated x-axis labels enhance readability, offering insights into the median price distribution for each GPU brand.



**Observation:**

* Nvidia processors take the top spot in terms of pricing, likely reflecting their exceptional performance. Nvidia is widely recognized as one of the leading companies in producing top-notch graphics cards in recent years
* Moving on to Intel graphics, it's important to note that Intel doesn't include a dedicated graphics card in their laptops. Instead, they offer integrated graphics with the CPU. Essentially, when you buy an Intel-powered laptop, you're paying for the CPU with a bit of GPU power included.
* AMD graphics cards stand out as the most budget-friendly option in the market. They are known for providing a good budget experience. However, it's worth mentioning that laptops with AMD processors are not as common; the market is largely dominated by Intel and Nvidia. AMD tends to have a stronger presence in the desktop market.

**2.5.5 Dropping the "Gpu" column**

****

* This line of code removes the 'Gpu' column from the DataFrame 'df', streamlining the dataset by dropping the GPU details as they have been processed and incorporated into a new 'Gpu brand' column

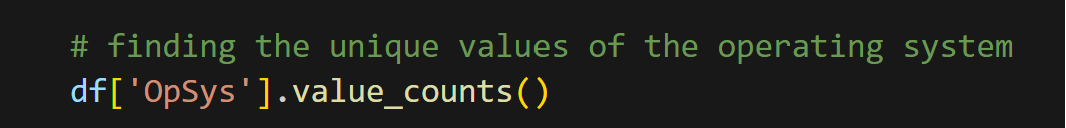
**Output:**



**2.6 Fixing the "Opsys" column**

**2.6.1 Unique values the "OpSys" column**

**Code:**



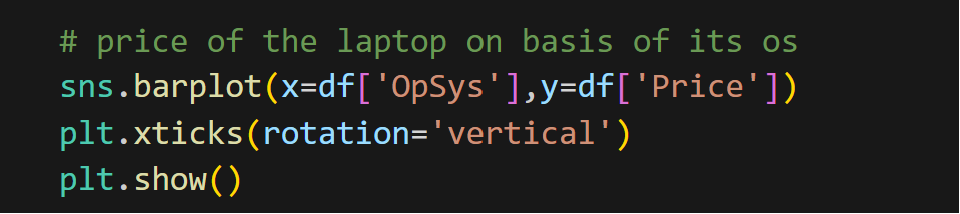
* This code drops the **'Gpu'** column from the DataFrame 'df'. The removal is done to simplify the dataset after processing GPU information, enhancing the clarity and efficiency of subsequent analyses

**Output:**

|  |  |
| --- | --- |
|  | **Observation:**   * The newest operating system, Windows 10, holds the top spot in the market. Surprisingly, the category labelled "No OS" comes next. Following closely is Linux, well-known among cybersecurity developers. Despite being older, some people still use Windows 7. Notably, all MacBooks exclusively run on the macOS operating system. |

**2.6.1 Unique values the "OpSys" column**

**Code:**

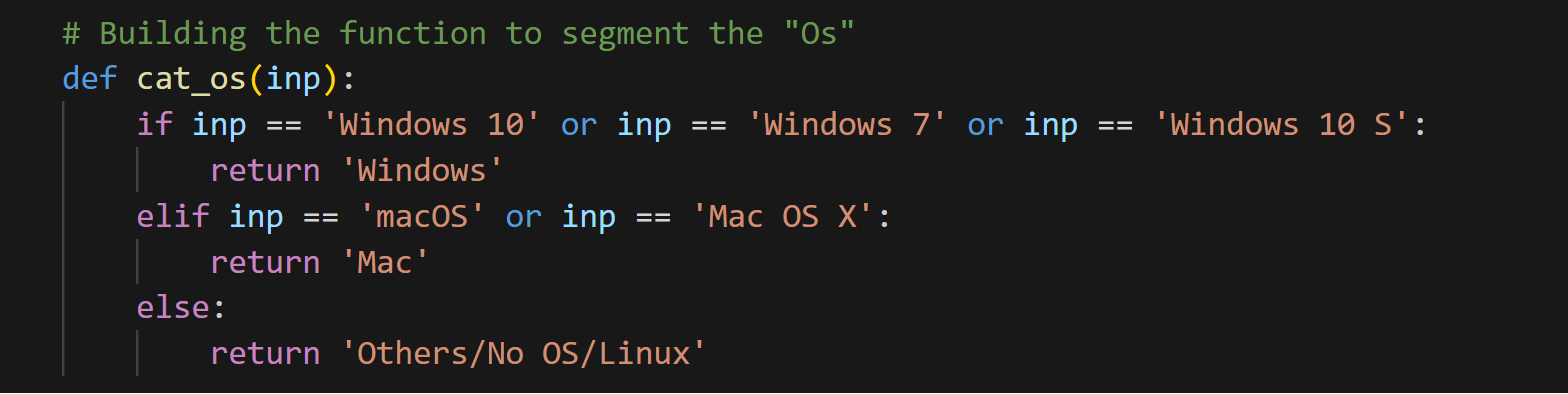
****

* This code uses Seaborn to create a vertical bar plot, comparing laptop prices across different operating systems from the 'OpSys' column.

|  |  |
| --- | --- |
|  | **Observation:**   * Given the multitude of operating systems in laptops with relatively low numbers, the price difference is quite significant. To simplify our analysis, we're categorizing the operating systems into a few segments: Windows, macOS, and others. This segmentation will help us better understand and manage the variance in prices. |

**2.6.1 Unique values the "OpSys" column**

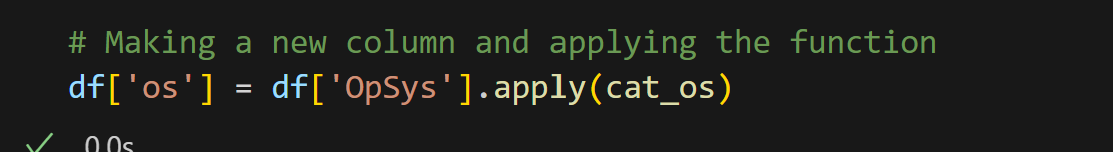
**Code:**

****

* This function, named cat\_os, categorizes operating systems into three groups. If the input is **'Windows 10,' 'Windows 7,' or 'Windows 10 S**,' it returns **'Windows**.' If the input is 'macOS' or 'Mac OS X,' it returns 'Mac.' Otherwise, it assigns **'Others**/**No** **OS**/**Linux**.' This function simplifies and groups operating systems for analysis.

**2.6.2 Applying ‘cat\_os’ function to ‘os’ column**

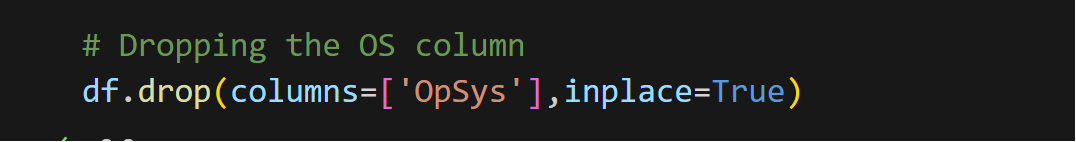
**Code:**



* Applying function to each row in the column

**2.6.3 Dropping unwanted columns**

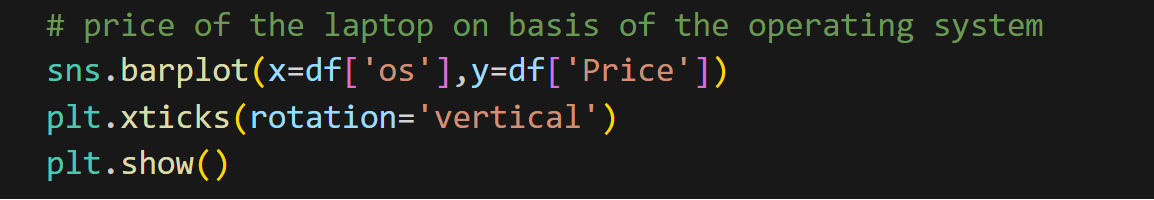
**Code:**

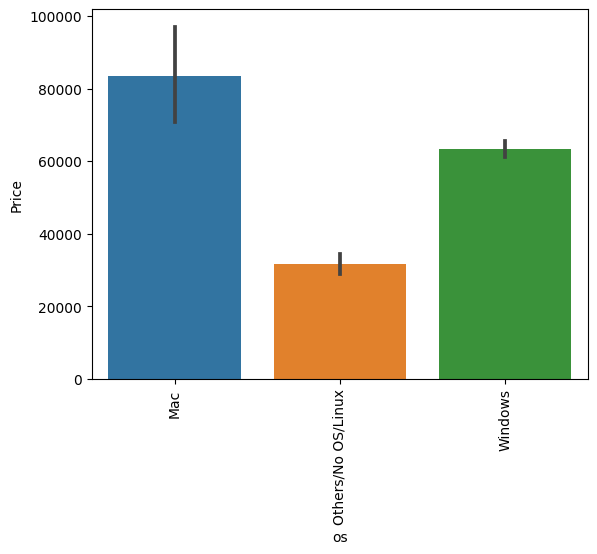
****

* Dropping the **‘OpSys’** Column

**2.6.4 Price of the laptop on basis of the operating system**

**Code:**





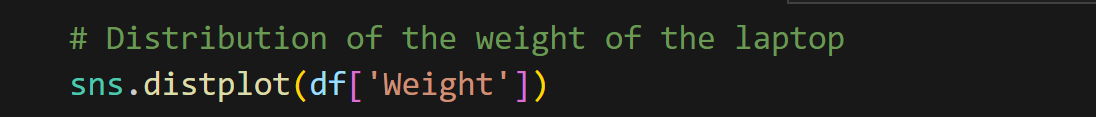
**Observation:**

* Laptops featuring macOS have the highest price, approximately Rs 80,000. However, it's essential to note that the bar error variance is exceptionally high, indicating that the number of laptops with macOS is limited in the dataset.
* In contrast, Windows, the most popular operating system, comes at a cost of around Rs 60,000. Other operating systems, collectively, have an average price of about Rs 30,000.

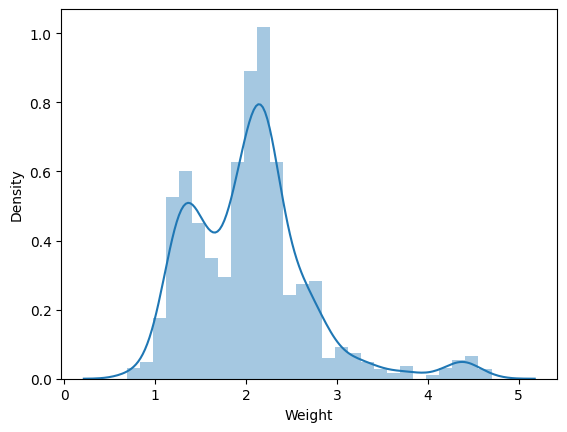
**2.7 Exploring the "Weight" Column**

**2.7.1 Distribution of the weight of the laptop**

**Code:**

****

* This code utilizes Seaborn to create a distribution plot for the 'Weight' column in the DataFrame 'df.' The plot visualizes the distribution of laptop weights, showing the frequency of different weight values.

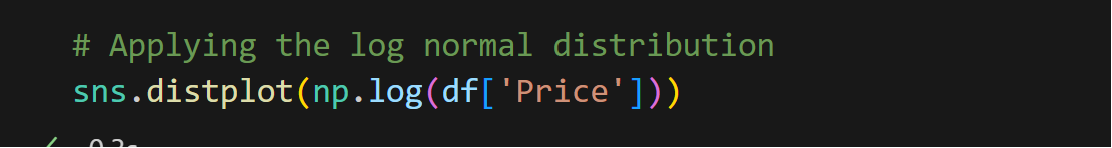


Observation:

* The chart has a mean and median of around 2.5. This means that the average and the middle value of the weight variable are both close to 2.5.
* The chart has a skewness and kurtosis of close to zero. This means that the data is not skewed to either side, and has a normal distribution shape.

**2.7 Log Normal distribution**

**Code:**

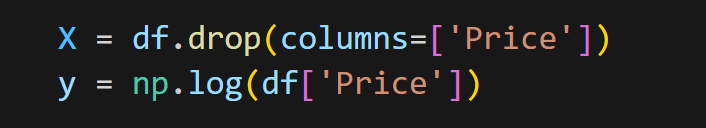


|  |
| --- |
|  |
| **Observation:**   * Before the distribution was right skewed but after applying the log normal function the distribution of the price is uniform |

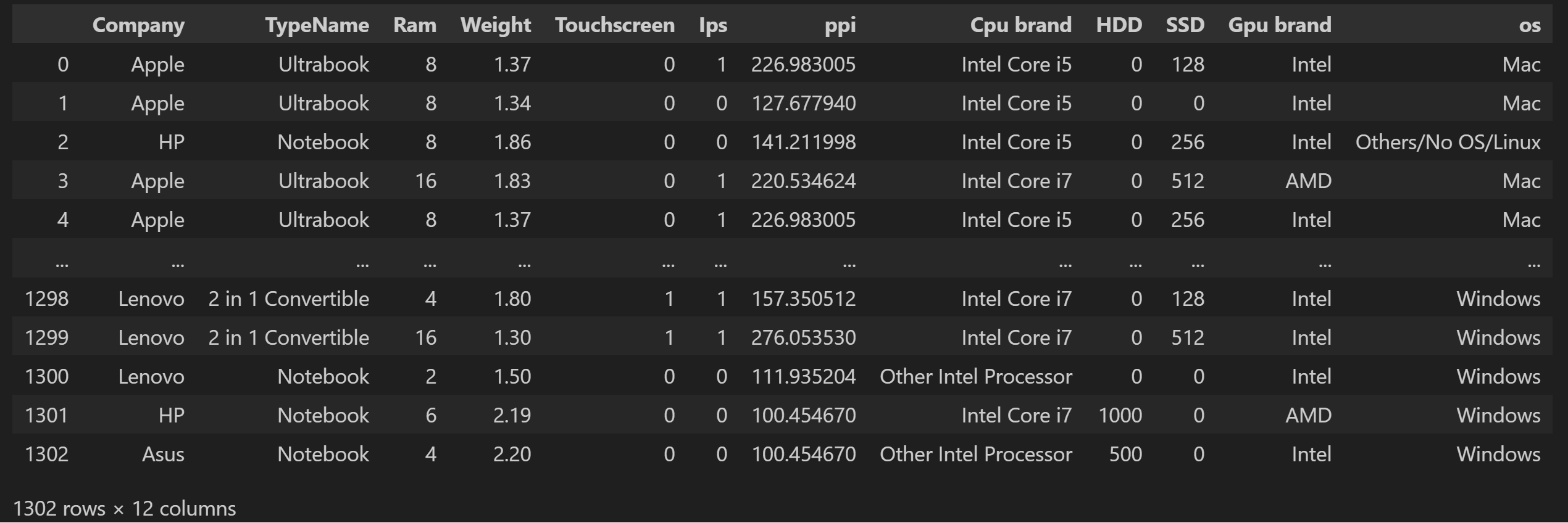
**3. Machine Learning**

**3.1 Making Input and output variable**

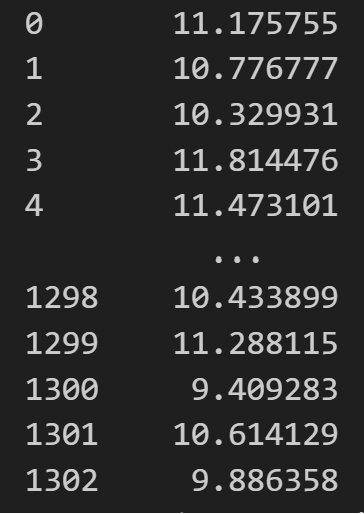
**Code:**

****

**Output of X variable:**

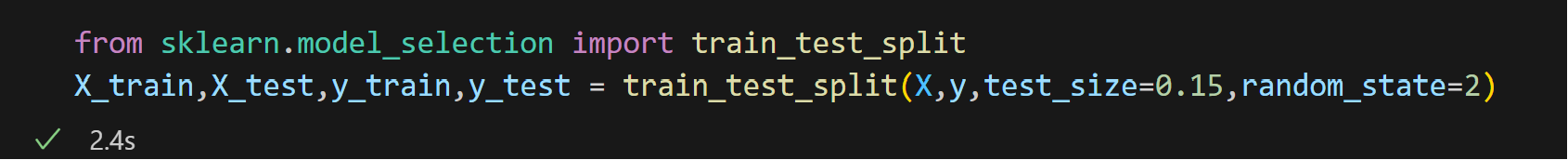
****

**Output of Y variable:**

****

**3.2 Train-test split**

**Code:**



* This code uses scikit-learn's train\_test\_split function to split the dataset into training and testing sets.
* The features (X) and corresponding labels (y) are divided into training **(X\_train, y\_train)** and testing **(X\_test, y\_test)** sets, with a test size of 15% and a random state of 2 for reproducibility