**Mini-project**

**On**

**Data Analysis and Visualization of**

**Laptop usage among JNEC Students**

****

**Diploma in Computer System and Network**

**Jigme Namgyel Engineering College**

**Department of Information Technology**

**Nov 28th, 2024**

**Module Name: Network Data Analysis and Visualization**

**Module Tutor: Mrs. Sonam Wangmo**

**Members: Sonam Pelden 05230135, Tshering Dorji 05230139**

# Abstract

This project explores student laptop preferences at Jigme Namgyal Engineering College by analyzing data on specifications, pricing, and usage trends. A comprehensive dataset was collected using Google Sheets with the collaboration of class representatives and councilors. Despite initial challenges with low participation, these were addressed through manual efforts such as door-to-door data collections. The data was thoroughly cleaned to correct inconsistencies, address missing values, and remove outliers, ensuring its accuracy and reliability. Using Python’s pandas library, the data was extracted, formatted, and analyzed, highlighting valuable patterns and insights. Various visualizations was performed such as line graphs, pie charts, and scatter plots, were created to make trends and relationships easier to identify. This project not only enhanced the understanding of student laptop preferences but also established a strong foundation for future data-driven decision-making.

Contents

[Abstract 2](#_Toc183648143)

[CHAPTER 1: INTRODUCTION 4](#_Toc183648144)

[1.1 BACKGROUND 4](#_Toc183648145)

[1.2 AIM AND OBJECTIVES 5](#_Toc183648146)

[CHAPTER 2: METHODOLOGY 6](#_Toc183648147)

[2.1 Data Collection and Extraction 6](#_Toc183648148)

[2.2 Tools and Libraries Used. 7](#_Toc183648149)

[2.3 DATA CLEANING AND PREPARATION 12](#_Toc183648150)

[HANDLING MISSING VALUES 12](#_Toc183648151)

[REMOVE DUPLICATES 19](#_Toc183648152)

[HANDLING OUTLIERS 24](#_Toc183648153)

[2.4 DATA ANALYSIS 30](#_Toc183648154)

[Exploratory Analysis 30](#_Toc183648155)

[Descriptive Statistical Analysis 34](#_Toc183648156)

[Comparative Analysis 37](#_Toc183648157)

[2.5 Data Visualization 38](#_Toc183648158)

[Line Graph 38](#_Toc183648159)

[Bar Graph 43](#_Toc183648160)

[Scatter Plot 45](#_Toc183648161)

[Pie Chart 46](#_Toc183648162)

[CHAPTER 3: RECOMMENDATIONS 48](#_Toc183648163)

[CHAPTER 4: CONCLUSION 49](#_Toc183648164)

[References: 50](#_Toc183648165)

[APPENDICES 51](#_Toc183648166)

# CHAPTER 1: INTRODUCTION

## 1.1 BACKGROUND

In today’s digital age, laptops play a vital role in students’ academic lives, influencing their productivity, interaction with technology, and learning experiences. At Jigme Namgyal Engineering College (JNEC), students from various courses, such as Mechanical Engineering, Computer Systems and Networks, and other courses depend heavily on portable computing devices. Understanding laptop preferences is important as it helps students make better choices when buying a laptop. The choice of a laptop depends on factors such as brand, specifications, and price. Selecting the right laptop is important, as students have varying needs—from basic tasks like browsing and document editing to advanced computational work. These diverse requirements make the decision-making process challenging, highlighting the need for a detailed analysis to identify trends and patterns in laptop usage.

This project aims to analyze and visualize laptop usage among JNEC students. Data was collected on various aspects of laptops, such as brand, model, RAM, storage, price, and other key features, to understand the diversity of preferences within the student community. Collaborative efforts with class representatives and manual outreach methods ensured the collection of a comprehensive and reliable dataset.

The project’s main objectives include identifying popular laptop trends, understanding budget constraints, and exploring the relationship between laptop specifications and their prices. The findings aim to provide insights based on data to assist students in making better purchasing choices while helping the institution improve technology support and curriculum development.

Advanced data analysis techniques and tools such as Python’s pandas library and Google Sheets were utilized to process the raw data and extract meaningful and useful insights. Through clear visualizations and structured analysis, the study highlights the diverse preferences of students regarding laptops, offering a better understanding of how technology impacts their academic experiences. This project not only reveals current trends in laptop usage but also creates a solid foundation for future research on technology adoption and user behavior in educational environments.

## 1.2 AIM AND OBJECTIVES

**Aim:**

The aim of this project is to collect and analyze a dataset of laptops usage among the JNEC students, focusing on key specifications like brand, model, and pricing.

**Objective:**

To fulfill the aim of this project, the following objectives have been set:

1. To design and implement an effective data collection process that gathers detailed laptop specifications from students, ensuring relevance and accuracy for the analysis.
2. To extract and consolidate the collected data into a structured format suitable for further processing and analysis.
3. To perform data cleansing, including handling missing values, ensuring uniform formatting, and synchronizing column data for consistency across the dataset.
4. To detect and remove outliers to ensure the dataset's integrity and reliability for statistical analysis.
5. To conduct statistical analysis to identify key trends, correlations, and patterns within the laptop specifications and pricing data.
6. To visualize the data through various charts (line graphs, pie charts, scatter plots) to present meaningful insights, making the dataset easier to interpret and understand for decision-making.

# CHAPTER 2: METHODOLOGY

## 2.1 Data Collection and Extraction

**Overview of Data Collection**

The data collection phase is critical to the success of this project as it determines the quality and relevance of the dataset. The data for this project was not pre-existing and needed to be collected specifically to meet the project’s objectives. The data was collected using google sheets as the primary platform for collaboration and data management among team members. The dataset was designed in accordance with the project’s specific requirements, ensuring comprehensive coverage of all necessary fields and relevant information.

**Data Collection Process**

The main source of data was information gathered directly from students regarding their laptops, including details such as the brand, model, and condition of the devices. To ensure that everyone had an opportunity to participate in the data collection, we also engaged class representatives and councilors. These individuals helped share links to their respective classes and blocks, facilitating wider involvement in gathering laptop information.

These are the types of information we collected from the students:

1. Company: The brand or manufacturer of the laptop (e.g., HP, Acer, Asus).
2. Product: The model’s name or identifier of the laptop.
3. TypeName: The type of laptop (e.g., Notebook, Ultrabook).
4. Inches: The screen size of the laptop in inches.
5. Screen Resolution: The resolution of the laptop screen (e.g., Full HD 1920x1080).
6. Ram: The amount of RAM (memory) in the laptop, usually given in GB (e.g., 8GB, 16GB).
7. OpSys: The operating system installed on the laptop (e.g., Windows 10, No OS).
8. Cpu Brand: The brand of the CPU (processor) used in the laptop (e.g., Intel, AMD).
9. Cpu Model: The specific model of the CPU (e.g., Core i5 7200U, A9-Series 9420).
10. Cpu Rate: The speed of the CPU, typically measured in GHz (e.g., 2.5GHz, 3GHz).
11. SSD: The size of the Solid-State Drive (SSD) storage in GB (e.g., 256GB, 512GB).
12. HDD: The size of the Hard Disk Drive (HDD) storage in GB (e.g., 500GB, 0GB if not present).
13. Flash Storage: The amount of extra flash storage in GB (often 0GB if not available).
14. Hybrid: Indicates if the laptop has a hybrid storage system (a mix of SSD and HDD).
15. Gpu Brand: The brand of the Graphics Processing Unit (GPU) (e.g., Intel, Nvidia, AMD).
16. Gpu Model: The specific model of the GPU (e.g., HD Graphics 620, Radeon R5).
17. Price\_Nu: The price of the laptop in Ngultrum.



Figure 1.Data Collection

## 2.2 Tools and Libraries Used.

1. **Preparation:**

* Developed data entry templates on Google Sheets to meet the project’s specific requirements.
* Created instructional documentation outlining the data entry process, which was distributed to all class representatives and councilors.

1. **Manual Data Collection:**

Due to insufficient initial responses from students, data was collected through manual entry:

* Class representatives were contacted to facilitate cooperation among their classmates for data entry.
* Despite the distribution of instructions and the link, low participation required additional efforts, prompting the project team to engage in door-to-door visits to gather laptop information directly from students.
* Each entry was carefully reviewed for accuracy and relevance to the project’s objectives.

**Instruments Used:**

* Google Sheets was selected as a tool for its user-friendly interface, real-time collaboration capabilities, and ease of access for all team members involved in the data collection process.
* To facilitate accurate data entry, detailed notes were created and shared with every class representative group and councilor's group. This documentation included clear instructions on how to correctly input their classmates' laptop details.

**Data Entry Verification**

To ensure data integrity, each data entry was systematically reviewed for consistency and accuracy. Additionally, a secondary verification process was conducted, which involved cross-checking entries with a sample of respondents to confirm the correctness of the data.

**Metrics for Success**

Our goal was to achieve at least an 80% response rate. We verified this through follow-ups and manual data collection to address the initial low participation in the collection of data.

**Data Extraction Process**

Once the data collection was completed, the second phase involved data extraction for analysis. This process was meticulously implemented to ensure accessibility and easy integration into analytical tools further.

**Extraction Methodology**

* **Data Format:**

The collected data was primarily stored in an Excel file containing multiple sheets. This format was chosen for its ability to handle complex datasets and allow for easy organization of distinct categories of data.

* **Extraction Procedure:**
  1. **Importing Additional Libraries:**

The libraries were imported for enhanced analysis and visualization, we imported additional libraries such as NumPy, Matplotlib, and Seaborn:

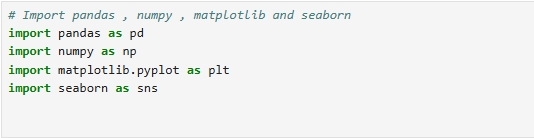


Figure 2.Import Libraries

* + **pandas**: A powerful library used for data manipulation and analysis. It provides data structures like DataFrame and Series for handling structured data.
  + **numpy**: A library for numerical operations, providing support for arrays and matrices, along with mathematical functions to operate on these arrays.
  + **matplotlib.pyplot**: A plotting library used to create static, animated, and interactive visualizations in Python. The pyplot module provides a MATLAB-like interface for creating plots.
  + **seaborn**: A data visualization library based on matplotlib, which provides a high-level interface for drawing attractive and informative statistical graphics.

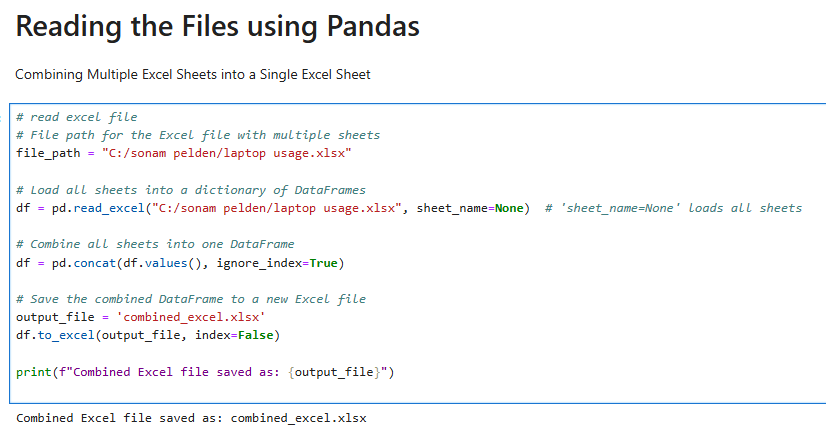


Figure 3. Combined multiples excel sheet

* 1. **Loading the Excel File:**

The Excel file which contained multiple sheets was loaded into a single DataFrame using the following command.

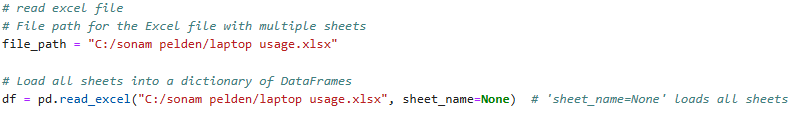


Figure 4.loading excel file

In this step, the pd.read\_excel() function reads the Excel file and loads all sheets into a dictionary of DataFrames, where each key corresponds to a sheet name.

**Combining the Data**:

After loading all sheets, the data was combined into a single DataFrame. This was achieved using the pd.concat() function, which concatenates the DataFrames stored in the dictionary.



This process ensured that all relevant data was consolidated into one manageable DataFrame for more straightforward analysis.

**Saving the Combined DataFrame**:

Once the data was combined the resulting DataFrame was saved into a new Excel file (combined\_excel.xlsx) for easy access in future analyses. The following command was executed to save the data:

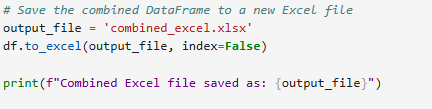


Figure 5. Save combined excel file

index=False ensures that the DataFrame's index was not included in the saved file, maintaining the focus on the original data.

**Displaying the Combined DataFrame**:

To facilitate verification and further exploration of the data, the new Excel file (combined\_excel.xlsx) was loaded, and its contents were displayed:

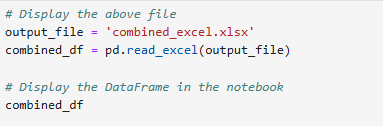


Figure .Displayed the combined excel file

Through this detailed extraction process, the data was effectively organized and consolidated into a single Excel file, setting the foundation for the subsequent analytical phases. This comprehensive approach ensured the preparation of a high-quality dataset for deeper analysis and insight generation.

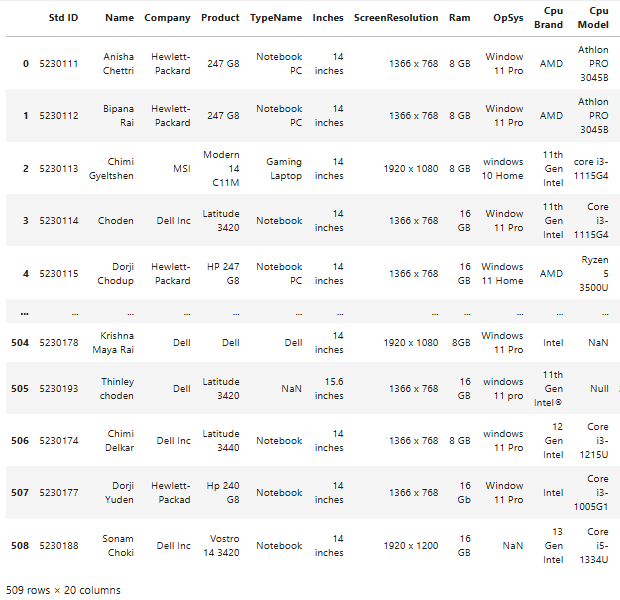
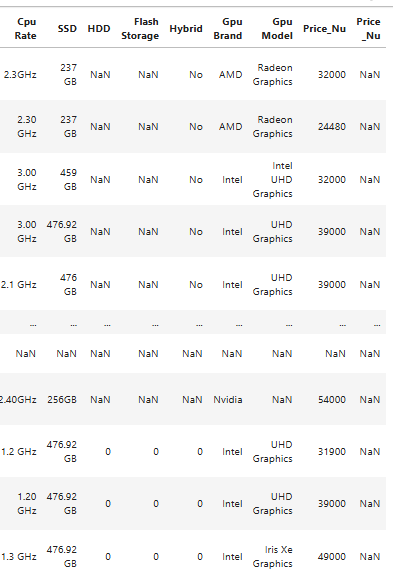


Figure : Combined Excel

## 2.3 DATA CLEANING AND PREPARATION

### HANDLING MISSING VALUES

1. Detecting missing values:
2. Checking for missing values using isnull():

The isnull() function is used to identify missing (null) values in a DataFrame. It returns a Boolean mask where True indicates a missing value (NaN).

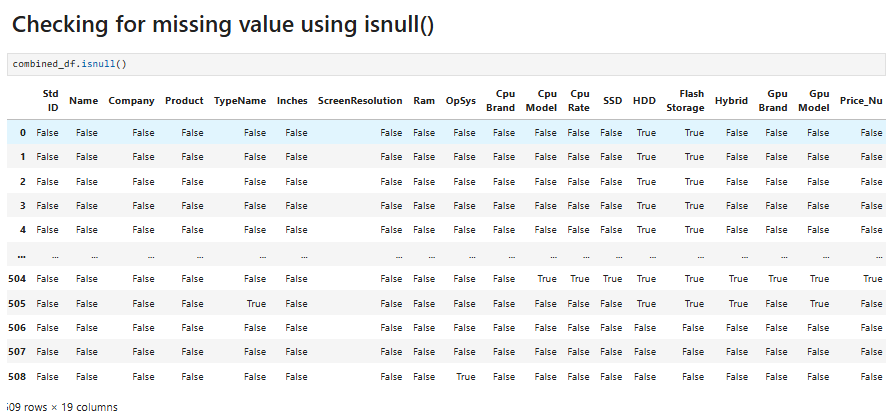


Figure 8. checking null value

1. Counting null values using sum() function

The **isnull().sum()** command is a **pandas** function that helps identify and count **missing values** (NaN or null) in a dataset.

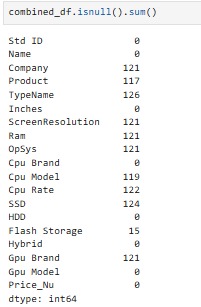


Figure 9. Counting null value

* **isnull()** creates a new DataFrame where each cell contains True if the original cell is NaN and False otherwise.
* **sum()** then adds up the True values (counting them as 1) for each column, resulting in a Series that shows the total number of missing values for each column.

1. Checking for the missing valuses using notnull():

The notnull() function is the inverse of isnull(); it identifies non-missing (non-null) values. It also returns a Boolean mask, but True indicates a value is present.

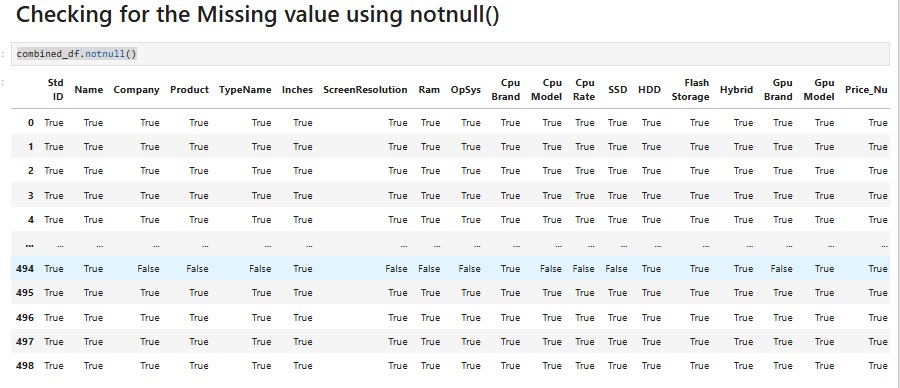


Figure 10. counting null value using notnull'

1. Counting null values using sum() function

The notnull().sum() command in pandas is used to count the non-missing (non-null) values in a dataset. It's the inverse of isnull().sum(). Which we can count the null value too.

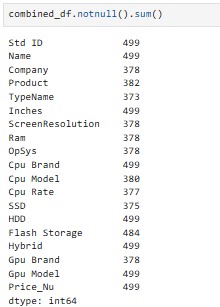


Figure 11. counting null value

**notnull()**:

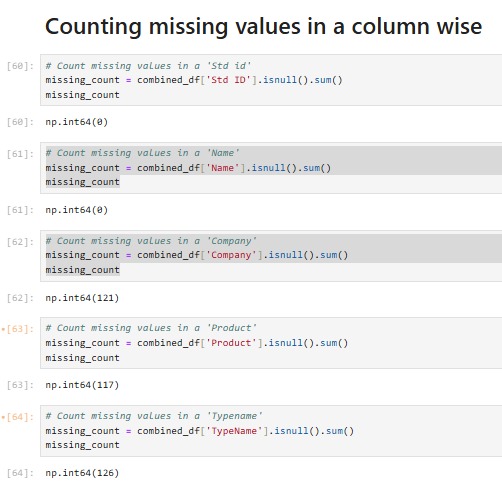
Returns a DataFrame (or Series) of the same shape as the original, with:

* + True where values are **not missing** (i.e., values are present).
  + False where values are **missing** (NaN).

**sum()**:

Counts the True values (since True is equivalent to 1), providing the **count of non-missing values** in each column.

1. Counting missing values in column wise





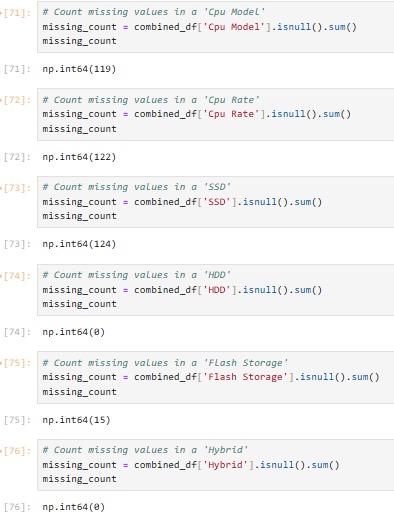




Figure 12.counting missing value in column wise

1. Filling all missing values
   * 1. Filling missing values in all column using ffill()

The **ffill()** method in pandas stands for **"forward fill"**. It is used to fill missing values (NaN) in a DataFrame or Series by propagating the last valid (non-missing) value **forward** along a specified axis.

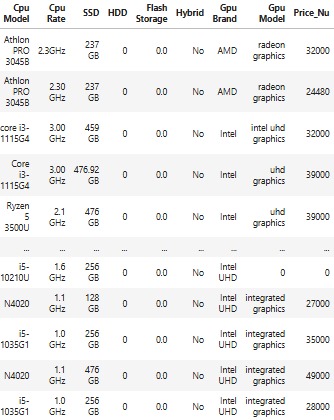


Figure .Filling missing value

* Missing values are replaced by the most recent non-missing value that appears before them.
* If there’s no valid value before a missing value, it remains NaN.
  + 1. Filling missing values in all column using bfill()

The **bfill()** method in pandas stands for **"backward fill"**. It is used to fill missing values (NaN) in a DataFrame or Series by propagating the next valid (non-missing) value **backward** along a specified axis.



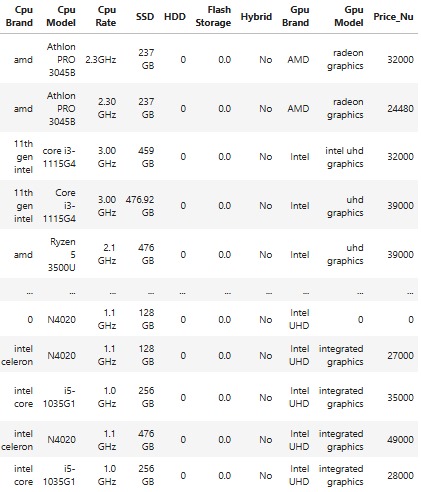


Figure .filling missing value using bfill

* Missing values are replaced by the next non-missing value that appears after them.
* If there’s no valid value after a missing value, it remains NaN

**Dropping missing values using drop()**

The. drop() method is used to remove specific rows or columns from a pandas DataFrame or Series. We can drop them by their labels (column names or row indices).

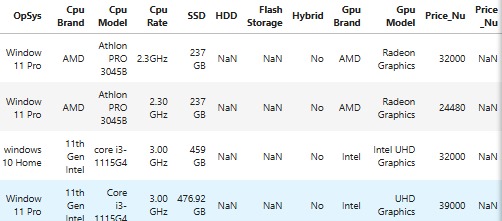


Figure 15 drop extra column

### REMOVE DUPLICATES

**Drop the extra column (Price\_Nu)**



When we load the file we saw that our table have duplicate Price\_Nu column, so we used drop() method to remove the extra column.



* The .drop() method is used to remove the column named 'Price\_Nu' from the combined\_df DataFrame.
* axis=1 specifies that a column (not a row) should be dropped.
* inplace=True ensures that the combined\_df is directly modified (no new DataFrame is returned).

**Save the modified DataFrame to an Excel file:**



* This line saves the modified combined\_df (after removing the column) to a new Excel file named combined\_excel.xlsx.
* index=False prevents the index from being written to the Excel file.

**Display the DataFrame**



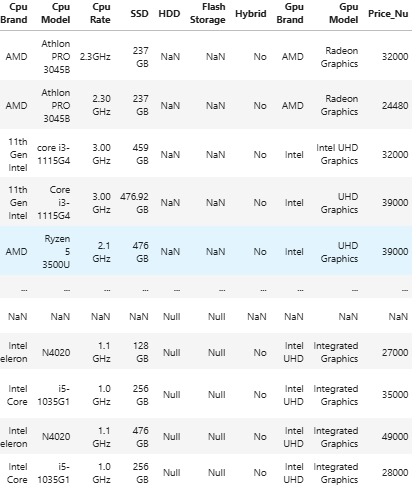


Figure 16. dropping extra Price\_Nu column

**Detecting and Removing Duplicates from all the columns**

* 1. **Detecting duplicate rows across all columns**

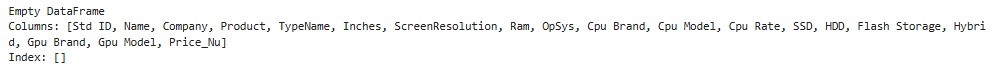
The .duplicated() method in pandas is used to identify duplicate rows in a DataFrame or Series. It returns a boolean Series where:

* True indicates a duplicate row.
* False indicates a unique row.



Figure 17. check the duplicate

* df1.duplicated(keep=False) checks for duplicate rows across all columns.
* The parameter keep=False ensures that all occurrences of duplicate rows are marked as True.



The output confirms that there are no duplicate rows in the DataFrame when checking across all columns. The empty DataFrame with no rows (and only column headers) indicates that no rows in df1 are repeated. This result aligns with the condition specified in the code using duplicated(keep=False).

Format Changing

* + 1. **Extract name with all capital letter**



Figure 18.code to filter the capital letter

**Filter names that are in all uppercase**

* combined\_df['Name'].str.isupper() checks if the strings in the Name column are entirely in uppercase.
* The condition filters rows where the Name column contains only uppercase names.
* The resulting subset of rows is assigned to capital\_names\_df.



**Display the filtered DataFrame:**



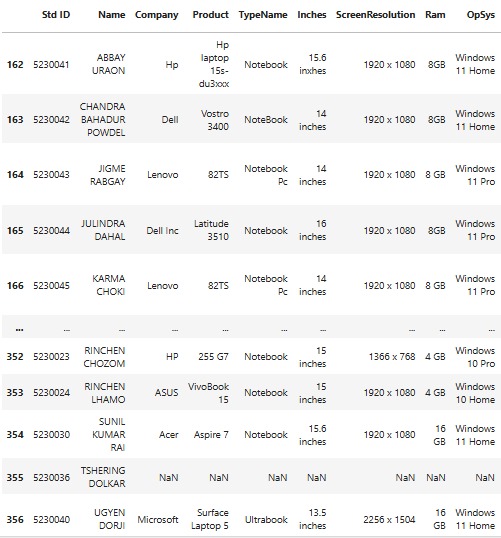


Figure 19. displaying the capital letter

* + 1. **Capitalizing only the first letter and last letter of each name**



Figure 20. code to capitalize only the first letter

**Apply str.capitalize() to capitalize both first and last names**



This code formats the Name column so that each word in the names (e.g., first and last names) starts with an uppercase letter while the rest are lowercase. It uses the apply() function with a lambda expression that:

* Splits each name into individual words (x.split()).
* Capitalizes the first letter of each word (word.capitalize()).
* Joins the capitalized words back into a full name (' '.join()).
* The modified names are reassigned to the Name column of the capital\_names\_df DataFrame**.**

**Display the Result:**



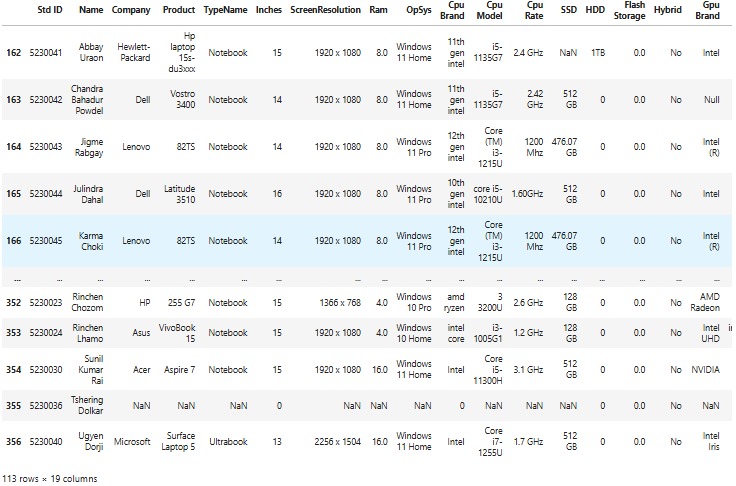


Figure 21. name made upper case to lower case

### HANDLING OUTLIERS

**Detecting outliers in Price**

This code creates a box plot to visualize the distribution of the "Price\_Nu" column in the DataFrame df1. Here's a brief explanation:

* plt.figure(figsize=(8, 6)): Sets up the plotting area with a figure size of 8 inches wide and 6 inches tall for clarity.
* sns.boxplot(data=df1, x='Price\_Nu'): Generates a box plot for the "Price\_Nu" column from the DataFrame df1, summarizing its statistical properties (minimum, maximum, median, interquartile range, and outliers, if any).
* plt.title('Boxplot for Price\_Nu'): Adds a title to the plot to indicate what it represents.
* plt.show(): Displays the box plot.
* This visualization provides insights into the variability, spread, and potential outliers of the "Price\_Nu" data.

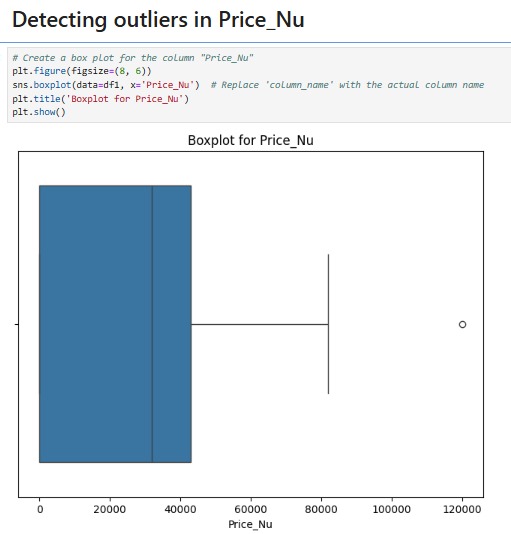


Figure 22:Detecting outliers in Price\_Nu

**Displaying the data inside the price column**

Listing all the data content inside the Price\_Nu column.

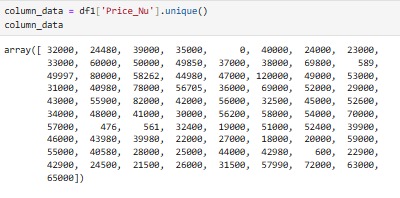


Figure 23.Data inside Price\_Nu

**Removing Outliers in Price**

The given code processes a numerical array, price\_data, by first removing invalid NaN values and then filtering out outliers based on specified bounds. Using the condition ~np.isnan(price\_data), it identifies and excludes all NaN values from the array, storing the cleaned result in price\_data\_cleaned. It then defines bounds (lower\_bound and upper\_bound), which represent the acceptable range of values (e.g., 0 to 90,000). The code applies these bounds with the condition (price\_data\_cleaned >= lower\_bound) & (price\_data\_cleaned <= upper\_bound) to filter out values that fall outside this range. The final cleaned dataset, free of NaN values and outliers, is printed to ensure it's ready for further analysis.



Figure 24.Removing outliers in Price\_Nu

**Displaying the identified rows with outliers**

The code identifies and extracts rows from the DataFrame df1 where the Price\_Nu column contains outlier values outside a defined range (lower\_bound and upper\_bound). It does this by applying two conditions: one to find values less than lower\_bound and another for values greater than upper\_bound, combining them using the logical OR operator (|). The rows that satisfy either condition are filtered from df1 and stored in the variable outliers. Finally, the outliers variable displays these rows, allowing further analysis, removal, or treatment of these extreme values based

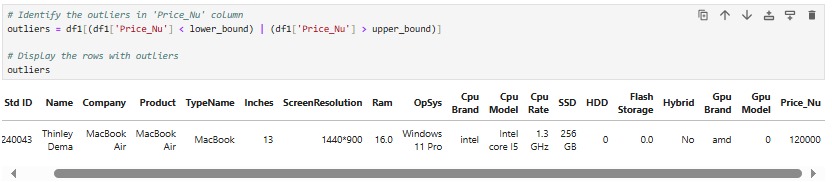
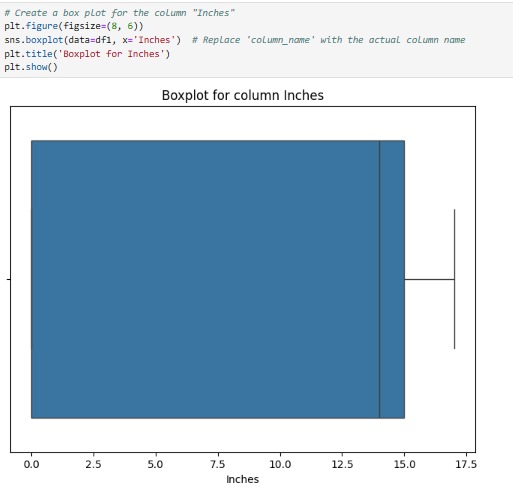


Figure 25.removed rows with outliers

This code helps identify rows in the dataset where the Price\_Nu values are extreme (outside the acceptable range). These rows can be analyzed, removed, or treated separately based on the requirements of the analysis.

**Detecting outliers in Inches**

The code generates a box plot to visualize the distribution of the "Inches" column in the DataFrame df1. It sets the plot size using plt.figure(figsize=(8, 6)) and creates the box plot with sns.boxplot(data=df1, x='Inches'), showing key statistics like the median, quartiles, and outliers. A title is added with plt.title('Boxplot for Inches'), and the plot is displayed using plt.show(). This helps identify the data's spread and any outliers in the "Inches" column.



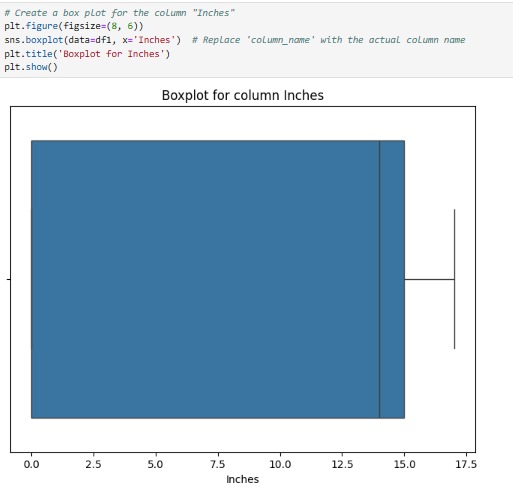


Figure 26 Detecting outliers in Inches

Since there are no outliers in the "Inches" column, the box plot displays a well-defined distribution without any extreme values outside the acceptable range. The median and quartiles are well within the interquartile range, indicating that the data is fairly consistent and does not have any significant deviations.

**Detecting outliers in Ram**

This code generates a box plot to visualize the distribution of the "Ram" column in the DataFrame df1 using Seaborn and Matplotlib. It begins by setting the figure size to 8x6 inches for better readability. The sns.boxplot() function is used to create the box plot, where the data source is specified as df1, and the column to be visualized is "Ram." A title is added to the plot using plt.title() for context, and finally, plt.show() is called to display the box plot. This visualization highlights key statistical insights, such as the minimum, maximum, median, interquartile range, and any outliers in the "Ram" column.

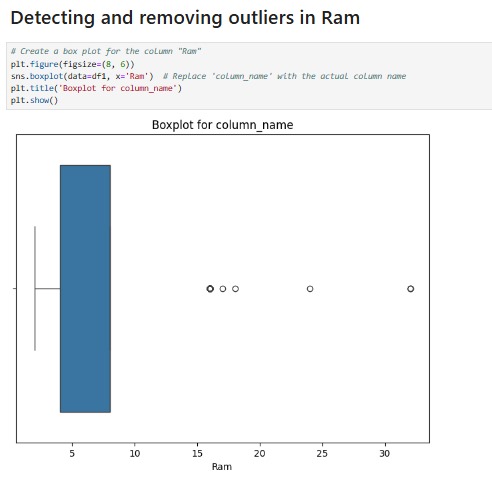


Figure 27.Detecting Outliers in ram

**Displaying all the data in the ram column**

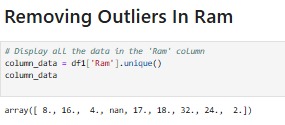


Figure 28Data inside ram column

**Removing outliers in Ram**

The code cleans and filters the ram\_data array by first removing any NaN values using np.isnan(). Then, it defines bounds (lower and upper) for outlier removal, filtering the data to retain values within the range of 5 to 25. The cleaned data, which now contains no NaN values or outliers, is displayed. This process ensures that the data is ready for further analysis, free from missing values and extreme outliers.

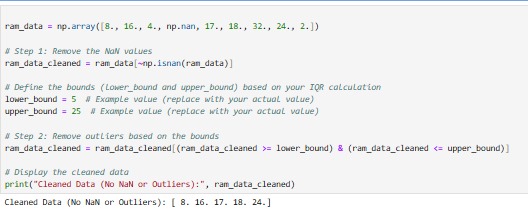


Figure 29.removing Outliers in ram

It can be observed that the outliers from the ram\_data has been removed and displayed with a clean data.

Saving the dataframe after a datacleaning.

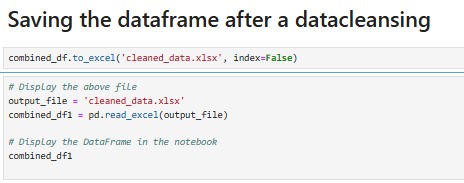


Figure 30.save the data frame after data cleansing'

The code combined\_df.to\_excel('cleaned\_data.xlsx', index=False) saves the combined\_df DataFrame to an Excel file named cleaned\_data.xlsx in the current working directory.

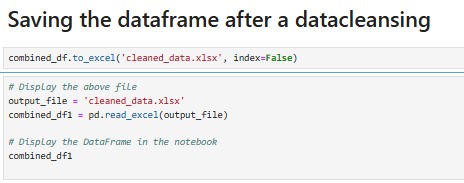


Figure 31.data displayed

The code reads the Excel file cleaned\_data.xlsx into a Pandas DataFrame named combined\_df1 using the pd.read\_excel() function, with the file name stored in the variable output\_file. Once loaded, the DataFrame is displayed in the notebook by simply referencing combined\_df1, allowing the data to be viewed in a tabular format for inspection and further analysis.

## 2.4 DATA ANALYSIS

### Exploratory Analysis

* 1. **Head():** The head() function is used to display the first few rows of a DataFrame or Series.

The head(10) function in Pandas displays the first 10 rows of the DataFrame or Series, providing a quick and detailed preview of the initial part of your data.

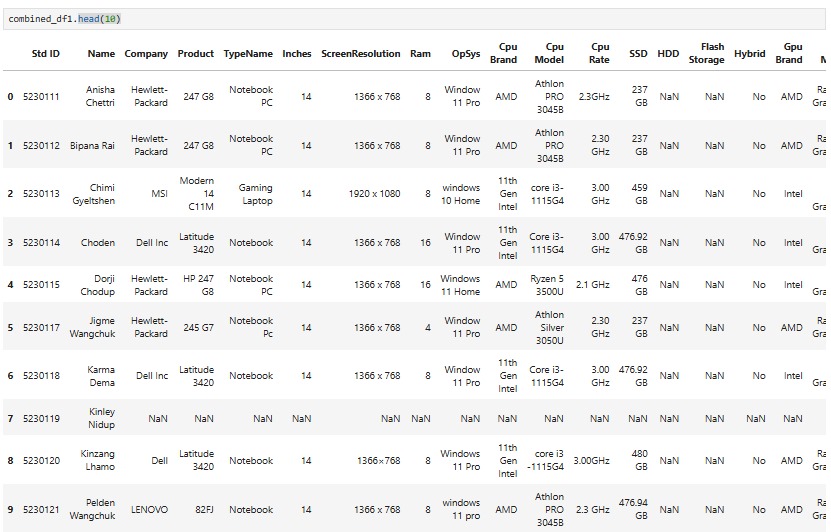


Figure 32. First 10 data

* 1. **Tail():**The .tail() function in Pandas is used to display the last few rows of a DataFrame or Series.

The .tail(10) function in Pandas retrieves and displays the last 10 rows of a DataFrame

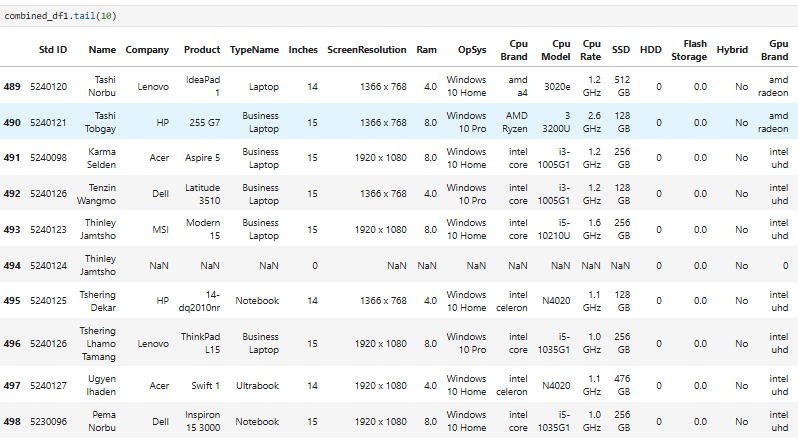


Figure 33Bottom 10 data

* 1. **info():** The .info() function in Pandas provides a concise summary of a DataFrame, including metadata about its structure, such as the number of rows and columns, column names, data types, non-null values, and memory usage.

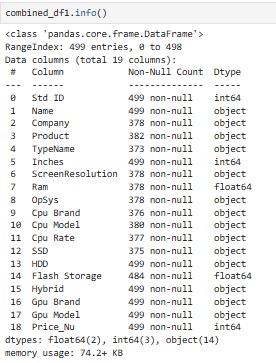


Figure 34.show data type

* Index Range: Shows the range of indices (e.g., RangeIndex: 100 entries for a DataFrame with 100 rows).
* Column Names and Data Types: Lists all column names with their respective data types (int64, float64, object, etc.).
* Non-Null Values: Displays the count of non-null values for each column, helping identify missing data.
* Memory Usage: Indicates the memory footprint of the DataFrame.
  1. **.shape:** The .shape attribute in Pandas provides the dimensions of a DataFrame or Series in the form of a tuple (rows, columns).
* rows: Represents the number of rows in the DataFrame or Series.
* columns: Represents the number of columns in a DataFrame (not applicable for a Series).



Figure 35. Total number of rows and column

The command combined\_df1.shape returns the dimensions of the DataFrame combined\_df1, which is (499, 19).

* 499: The number of rows in the DataFrame.
* 19: The number of columns in the DataFrame.  
  + 1. **.columns:** The .columns attribute in Pandas is used to access the column names of a DataFrame**.**
* It shows the column names (or headers) of the DataFrame.
* The result will be an index object containing the names of all columns in the DataFrame.

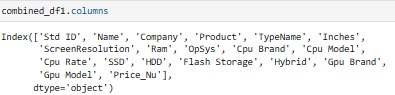


Figure 36. Displayed columns

* + 1. **.dtypes**: The .dtypes attribute in Pandas displays the data types of each column in a DataFrame, making it easy to understand the type of data stored in each column.

This is particularly useful for data cleaning and preprocessing, as it helps identify columns that may need type conversion or handling for compatibility during analysis or transformations.

Data types can include:

* int64 for integers
* float64 for floating-point numbers
* object for strings or mixed types
* datetime64 for datetime objects
* bool for boolean values, etc.

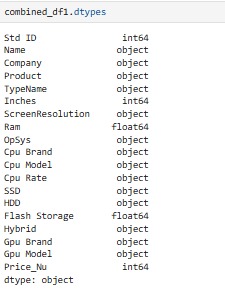
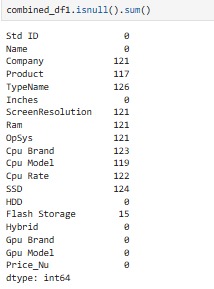


Figure 37. data type

* + 1. **.isnull().sum():** The isnull().sum() command in Pandas is used to identify and count the number of missing (NaN) values in each column of a DataFrame or Series.



### Descriptive Statistical Analysis

**Statistical Analysis of the Inches Column**

* 1. **mean():** The .mean() function calculates the arithmetic average (mean) of a numerical series or column.

**.round():** The .round() function rounds the given numerical value or series to a specified number of

decimal places.



The code combined\_df1['Inches'].mean().round(2) is used to calculate the average (mean) of the values in the "Inches" column of the combined\_df1 DataFrame and then round the result to two decimal places.

* 1. **.mode():**

The .mode() function in Pandas is used to find the most frequent (mode) value(s) in a dataset. It returns the value that appears most often in a column or series. If there are multiple values with the same highest frequency, it returns all those values in a sorted order.



* The code combined\_df1['Inches'].mode() returns the most frequent value (mode) in the 'Inches' column of the combined\_df1 DataFrame. In this case, the mode is 14, indicating that 14 appears the most often in the column. The result is a Pandas Series with the mode value 14.
* The .mode() function is useful for identifying the most common value in a dataset, which can help in understanding the distribution of data.
  1. **.std():** The .std() function in Pandas is used to calculate the standard deviation of a column or series.

****

* combined\_df1['Inches'] .std(): This calculates the standard deviation of the values in the 'Inches' column of the combined\_df1 DataFrame.
* The standard deviation measures how spread out the values are around the mean. A low standard deviation means the values are close to the mean, while a high standard deviation indicates that the values are more spread out.

4. **.var():** The .var() function in Pandas is used to calculate the variance of a column or series.

****

* The code **combined\_df1['Inches'].var().round(3)** calculates the **variance** of the values in the 'Inches' column of the combined\_df1 DataFrame and then rounds the result to three decimal places.
* **Variance** is a measure of how spread out the values are. It is the square of the standard deviation. While the standard deviation gives a sense of how much values deviate from the mean in the same units as the data, variance gives this information in squared units.

5. **value\_counts()**: The .value\_counts() function in Pandas is used to count the frequency of unique values in a column or series.

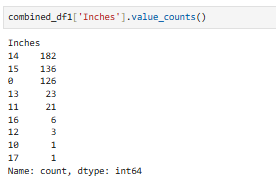
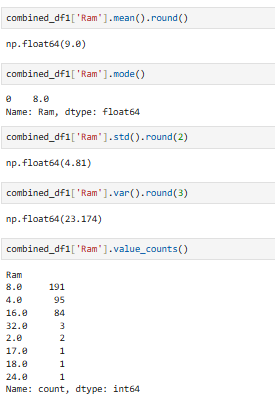


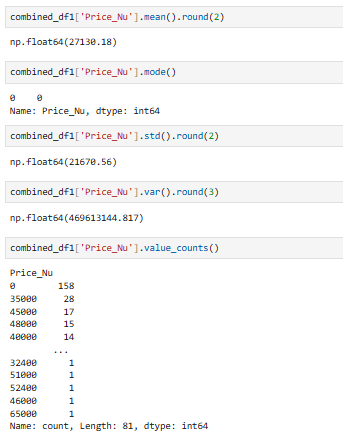
Figure 38. count values inside inches columns

This function counts the occurrences of each unique value in the Inches column and sorts them in descending order.

**Statistical Analysis of the Ram Column**

****

**Statistical Analysis of the Price\_Nu Column**

****

### Comparative Analysis

The command combined\_df1['Inches'].cov(combined\_df1['Ram']).round(2) calculates the covariance between the 'Inches' column and the 'Ram' column in the combined\_df1 DataFrame, and rounds the result to two decimal places.



The command combined\_df1['Inches'].cov(combined\_df1['Price\_Nu']).round(2) calculates the covariance between the 'Inches' and 'Price\_Nu' columns of the combined\_df1 DataFrame and rounds the result to two decimal places.

****

The command combined\_df1['Price\_Nu'].cov(combined\_df1['Ram']).round(2) calculates the covariance between the 'Price\_Nu' (price) and 'Ram' (RAM) columns in the combined\_df1 DataFrame and rounds the result to two decimal places.



## 2.5 Data Visualization

### Line Graph

**1. Line graph Ram vs Price\_Nu range from 4 - 6 GB of range and price of 50000 - 100000**

This code cleans and filters a DataFrame (combined\_df1) to analyze the relationship between RAM and price. It begins by replacing invalid values like 'Null' with None and removing rows with missing values in the Ram and Price\_Nu columns. It converts these columns to numeric data types, coercing invalid entries to NaN, which are then dropped. The data is filtered to include rows where RAM is between 4GB and 8GB and price is between 50,000 and 100,000 Nu. From the filtered dataset, the Ram and Price\_Nu columns are extracted and used to create a line graph. The graph plots RAM on the x-axis and price on the y-axis, with markers, gridlines, labels, and a legend, visually showing the relationship between these two variables in the specified ranges.

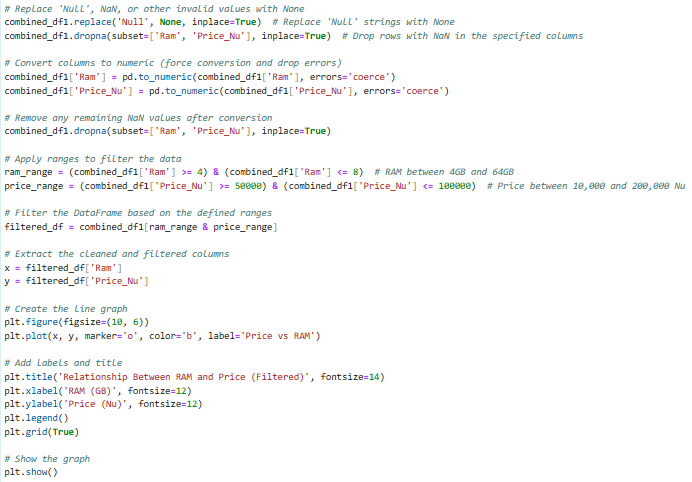
****

Figure 39. code to create line graph (Price vs ram)

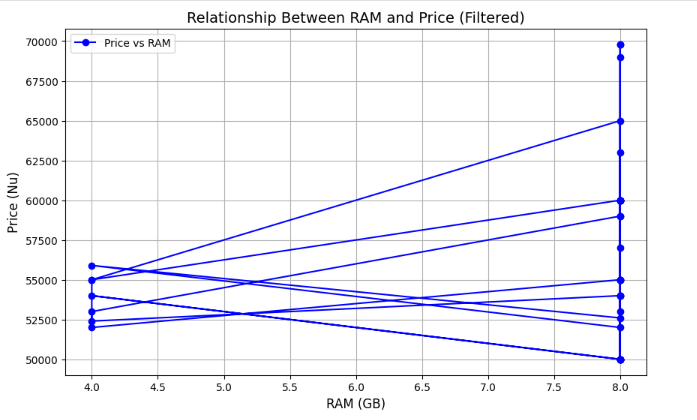
****

Figure 40. line graph (Price vs Ram)

**2. Line graph Ram vs Price\_Nu range from 6 - 16 GB of range and price of 10000 – 40000**

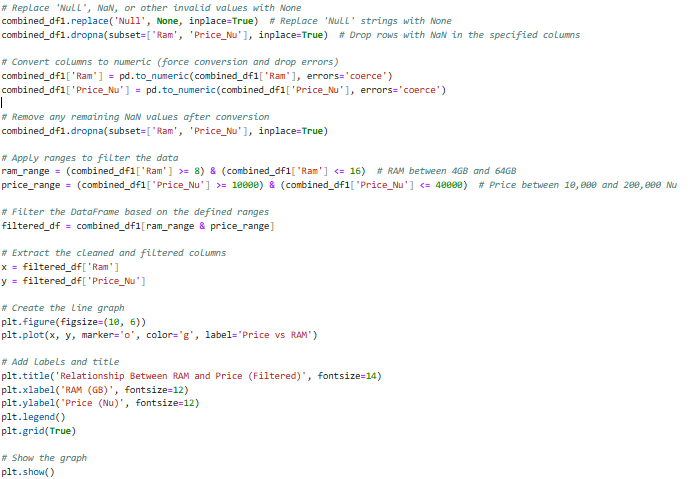
****

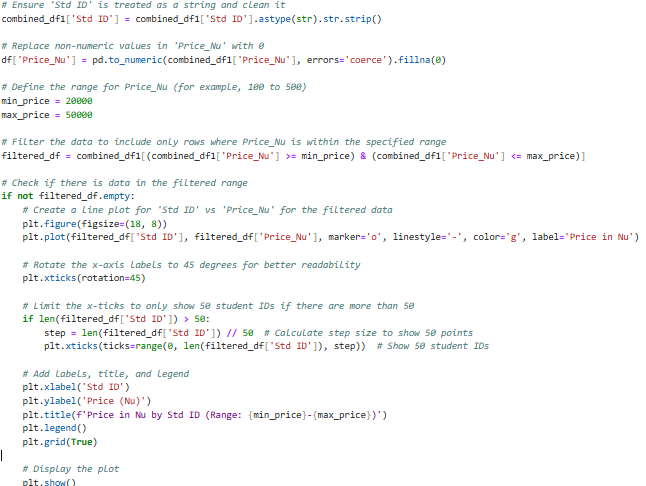
Figure 41. line graph code

This code processes and visualizes the relationship between RAM and price in a DataFrame (combined\_df1). It replaces invalid values like 'Null' with None and removes rows with missing data in the Ram and Price\_Nu columns. Both columns are converted to numeric types, and rows with non-convertible values are dropped. The data is filtered to include only rows where RAM is between 8GB and 16GB and price is between 10,000 and 40,000 Nu. From this filtered data, the Ram (x-axis) and Price\_Nu (y-axis) columns are plotted in a line graph with green markers. The graph includes a title, labels for axes, a legend, and gridlines, visually showing the relationship between RAM and price for the specified range of values.

****

Figure 42. line graph (price vs ram)

**Line plot with a range of Price\_Nu**

****

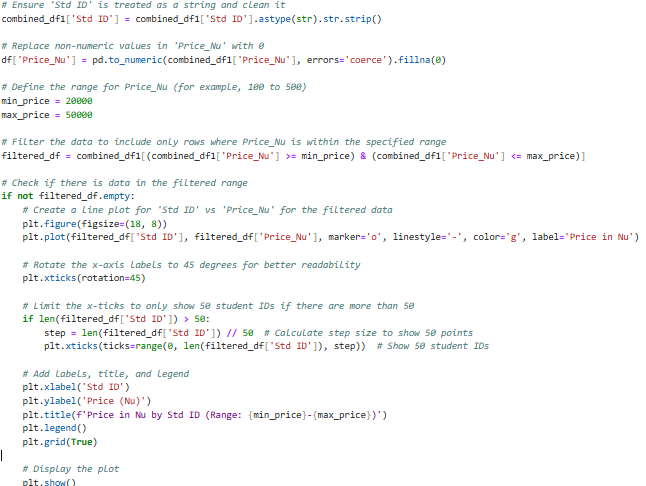
****

Figure 43. code to create line graph (price\_Nu vs std\_id)

The updated code filters the dataset by replacing invalid values like 'Null' with None and dropping rows with NaN in the Ram and Price\_Nu columns. It converts these columns to numeric, forcing invalid entries to NaN and dropping them again. The data is then filtered to include only rows where Ram is between 8GB and 16GB and Price\_Nu is between 10,000 and 40,000 Nu. The filtered data is used to plot a line graph showing the relationship between RAM and Price, with green markers for clarity, proper axis labels, a grid, and a legend, providing a clear visualization of the filtered dataset.

****

Figure 44.line graph (Price\_Nu vs Std\_ID)

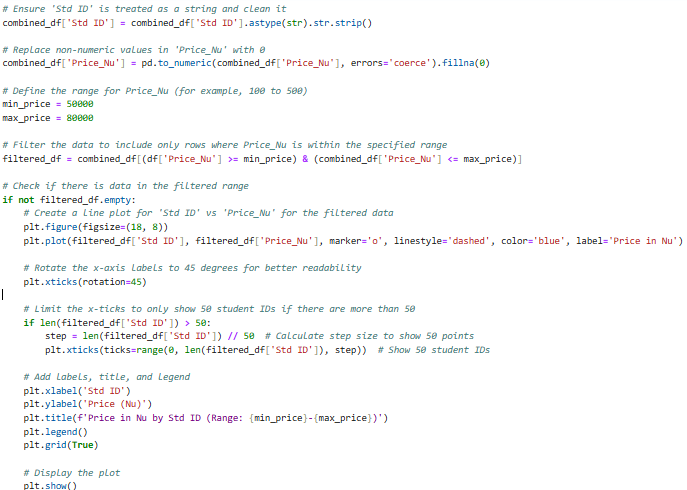
****

Figure 45. price vs Std\_Id in different range

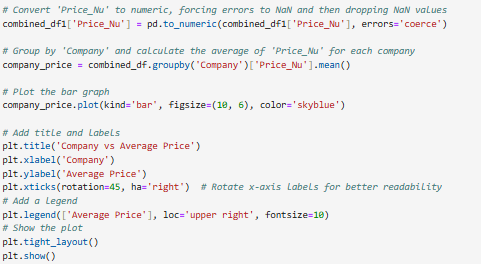
The code ensures the Std ID column is treated as a string by stripping any extra spaces, then cleans the Price\_Nu column by converting non-numeric values to 0. It filters the data within a specified price range (50,000 to 80,000 Nu) and checks if the resulting dataset is non-empty before proceeding. For visualization, a line plot is created to display the relationship between Std ID and Price\_Nu. The x-axis labels are rotated 45 degrees for readability, and if there are more than 50 student IDs, only 50 evenly spaced IDs are shown to maintain clarity. The graph includes a title, axis labels, a legend, and a grid, ensuring it is both informative and easy to interpret.

****

Figure 46. line graph

### Bar Graph

**Bar Graph 'Company vs Price'**

****

This code processes a DataFrame (combined\_df1) to calculate and visualize the average price (Price\_Nu) for each company. It begins by converting the Price\_Nu column to numeric, forcing invalid values to NaN, and then dropping rows with missing values. The data is grouped by the Company column, and the mean price for each company is calculated. A bar chart is plotted with companies on the x-axis and their average prices on the y-axis, using sky-blue bars. The plot is customized with a title, axis labels, rotated x-axis labels for readability, and a legend indicating "Average Price." Finally, the layout is adjusted to prevent overlap, and the chart is displayed, showcasing the average price per company.

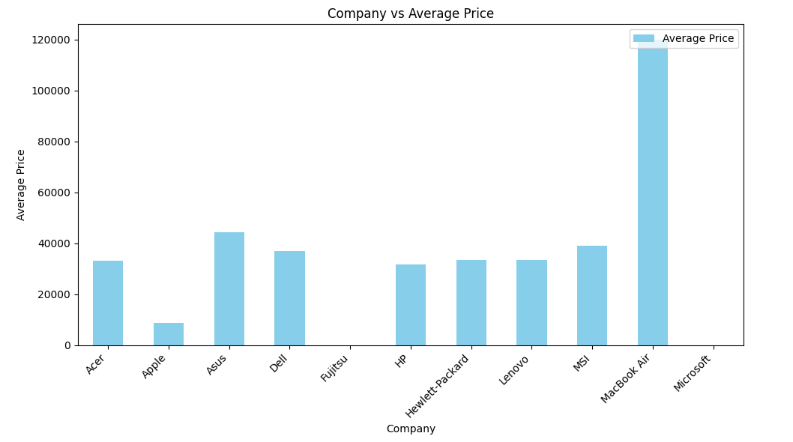
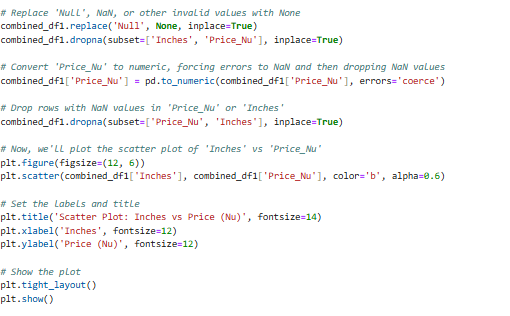


Figure 47. Bar graph (Price\_Nu vs Company)

### Scatter Plot

**Scatter plot: Inches vs Price**

****

This code processes a DataFrame (combined\_df1) to clean and visualize the relationship between the Inches and Price\_Nu columns. It starts by replacing 'Null' values with None and removing rows that have missing values in either Inches or Price\_Nu. The Price\_Nu column is then converted to numeric, with any non-numeric values coerced into NaN, and these NaN values are dropped from both Price\_Nu and Inches. After cleaning the data, a scatter plot is generated with Inches on the x-axis and Price\_Nu on the y-axis, using blue dots (color='b') with some transparency (alpha=0.6) for better clarity. The plot is customized with a title, axis labels, and adjusted layout to ensure proper display. The resulting scatter plot visually shows how Price\_Nu is distributed across different Inches values.

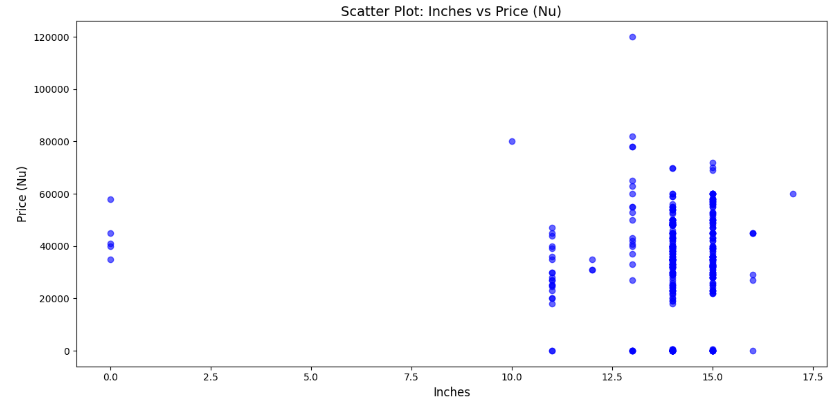
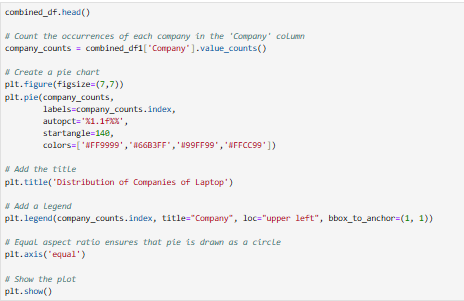


Figure 48 . Scatter plot Price vs inches

### Pie Chart

**Pie chart – Distribution of laptop company among students**



The code shows the first few rows of the dataset using combined\_df.head(). It then counts how many times each company appears in the 'Company' column of the combined\_df1 DataFrame using the value\_counts() function. This creates a list with each company and its count, which is shown as a pie chart. The pie chart is made using plt.pie(), where the company names are the labels, the percentages are shown with one decimal place, and the chart starts at an angle of 140 degrees. Different colors are used for each slice to make it look nice. A title is added with plt.title(), explaining what the chart shows. A legend is added with plt.legend(), placed outside the pie chart to avoid crowding. The legend lists the company names with the title "Company" and is placed in the top-left corner. The plt.axis('equal') makes sure the pie chart is a circle, and plt.show() displays the chart. This results in a clear pie chart that shows the distribution of companies in the data.

**Output:**

****

Figure 49. Pie chart - laptop company

# CHAPTER 3: RECOMMENDATIONS

**Improving Data Collection Participation**

To get better data, it’s important to involve more people in the data collection process. Offering incentives, simplifying the process, and spreading awareness about the benefits of the project can encourage students to participate.

**Ensure Data Quality**

Good data means better results. Using tools to check for errors, standardizing entries, and monitoring during collection can help keep the data clean and reliable.

**Better Communication and Collaboration**

Clear communication and teamwork make projects run smoother. Regular updates, shared tools, and clear roles can help everyone stay on the same page and work more efficiently.

**Provide More Detailed Data Visualizations**

Detailed and interactive charts, like dashboards, can make the data easier to understand. Highlighting key trends and allowing users to explore different aspects of the data adds more value.

**Use More Advanced Statistical Methods for Deeper Insights**

Using techniques like regression or clustering can reveal deeper insights and help predict trends. These methods make the analysis more detailed and actionable

# CHAPTER 4: CONCLUSION

In conclusion, this project successfully collected, analyzed, and visualized a dataset of laptop specifications and pricing information from students. By utilizing Google Sheets for data collection and ensuring active collaboration through class representatives, the project overcame initial low response rates to achieve an 80% participation rate. The subsequent data cleansing process addressed inconsistencies and missing values, ensuring that the dataset was accurate and reliable for analysis. Through statistical methods, outlier removal, and data synchronization, the project enhanced the quality of the dataset, making it suitable for deeper insights. Visualization techniques, including line graphs, pie charts, and scatter plots, effectively presented trends and relationships within the data, making complex information accessible and actionable. This structured approach not only fulfilled the project’s aim of understanding laptop preferences and market trends but also provided a solid foundation for future data-driven decisions and further analysis in the field.

# References:

Data Science Tutorials. (2022, October 12). Creating histograms in Jupyter Notebook [Video]. YouTube. https://www.youtube.com/watch?v=-otKrWkZ\_GQ

Khan Academy. (2023, January 15). How to solve equations using algebra [Video]. YouTube. https://www.youtube.com/watch?v=abc123xyz

Matplotlib. (n.d.). Types of plots. Matplotlib. Retrieved November 27, 2024, from https://matplotlib.org/stable/plot\_types/index.html

Python Tutorial. (2023, March 15). Numpy analysis in Jupyter Notebook [Video]. YouTube. https://www.youtube.com/watch?v=IMrxB8Mq5KU

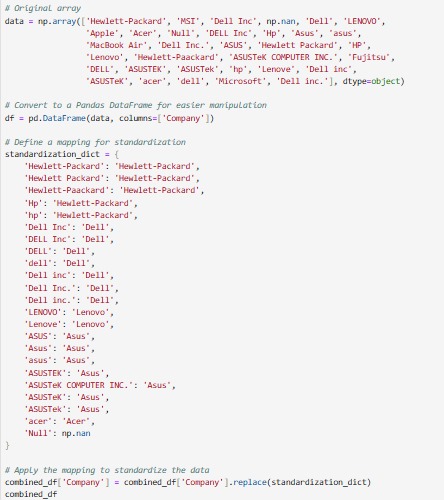
W3Schools. (n.d.). Python file open() method. W3Schools. Retrieved November 27, 2024, from https://www.w3schools.com/python/python\_file\_open.asp

# APPENDICES

**Synchronizing Column Data in a DataFrame**



Figure 50.list the data inside company column



**Displaying the data inside the original columns(company)**

combined\_df['Company']:

* This extracts the column named Company from the DataFrame combined\_df.
* The result is a Series (a one-dimensional array-like structure) containing all the values from the Company column.

.unique():

* The unique() method returns the distinct (unique) values present in the Series.
* This is helpful for understanding the variety of values in the column without duplicates.

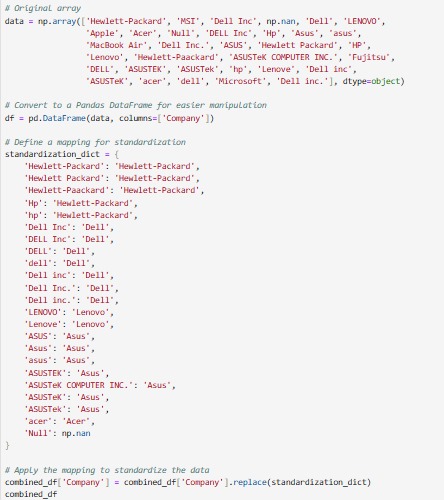
column\_data:

* The resulting array of unique values is stored in the variable column\_**data.**



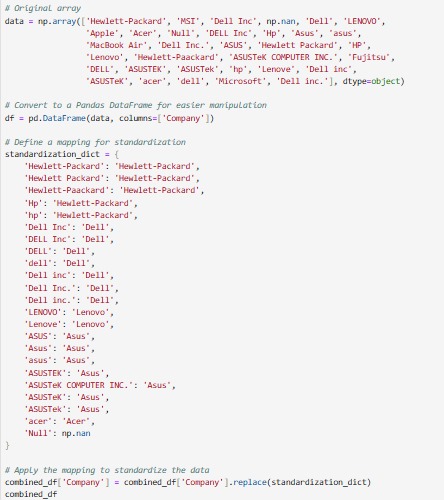
**Original Array**

* Purpose: Create a NumPy array named data containing various entries representing company names.
* Some company names are inconsistently formatted (e.g., different casing, typos, duplicates like "Dell Inc" and "dell inc"), and others are invalid (np.nan, 'Null').



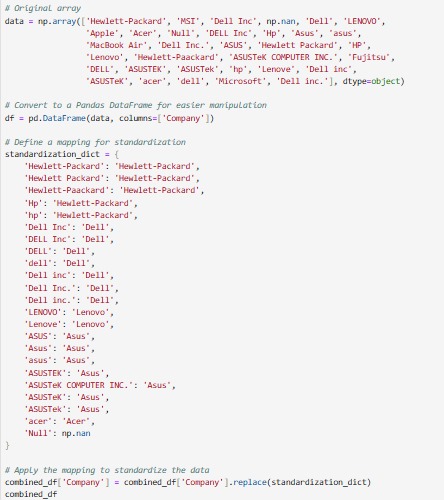
**Convert to a Pandas DataFrame**

* Converts the NumPy array data into a Pandas **DataFrame** for easier data manipulation.
* The resulting DataFrame has a single column named 'Company'.



**Define a Standardization Dictionary**

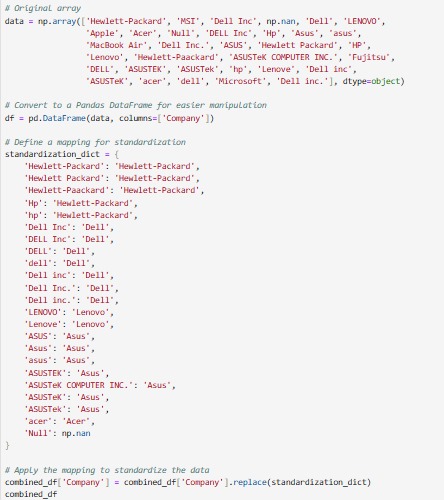
* Purpose: Create a dictionary mapping inconsistent entries to standardized values.
* Example: 'Hp', 'hp', and 'Hewlett-Packard' are mapped to 'Hewlett-Packard'.
* Invalid values like 'Null' or np.nan are standardized to np.nan.



**Apply the Mapping for Standardization**

replace():

* Replaces each value in the 'Company' column of the DataFrame with its corresponding value from the standardization\_dict.
* Any value not found in the dictionary remains unchanged.



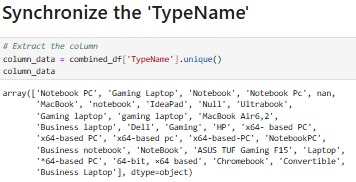
**Resulting DataFrame**

The DataFrame combined\_df now has a cleaned and standardized version of the 'Company' column where:

* All duplicates are merged into standardized names.
* Invalid values like 'Null' and np.nan are consistently treated as np.nan.



* + 1. **TypeName**





Displaying the results



* + 1. **Inches**



**Display the Cleaned Data**

Handling Invalid Entries:

* If the value is NaN (pd.isna(value)) or explicitly 'Null', replace it with 0.

Remove Variations of "Inches":

* Variants such as ' inches', ' inch', '"', or 'Inch' are removed using the replace() method.
* This ensures all size values are numeric.

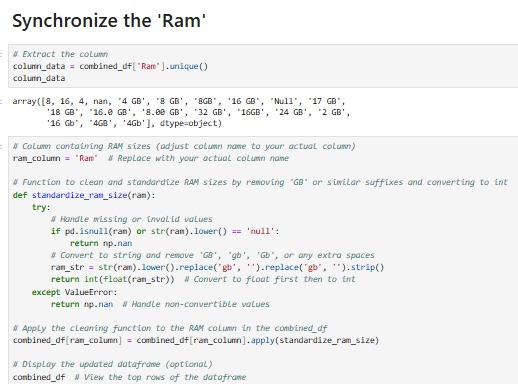
Convert to Numeric:

* Attempt to convert the cleaned string to a float and then cast it to an integer using int(value).
* If conversion fails (e.g., for invalid strings), return 0.

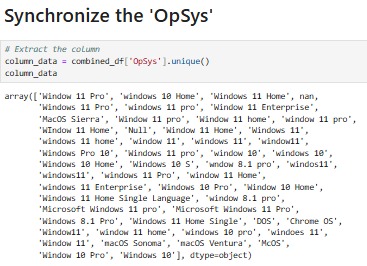
Apply(clean\_value):

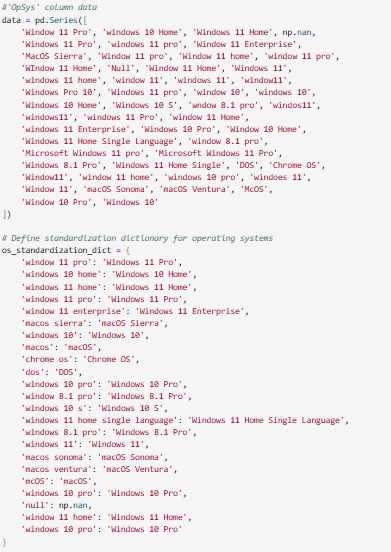
* Applies the clean\_value function to every entry in the Inches column.
* Each value is processed, cleaned, and converted to an integer (or replaced with 0 for invalid entries).
  + 1. **Ram**

Displaying the data



* + 1. **OpSys**

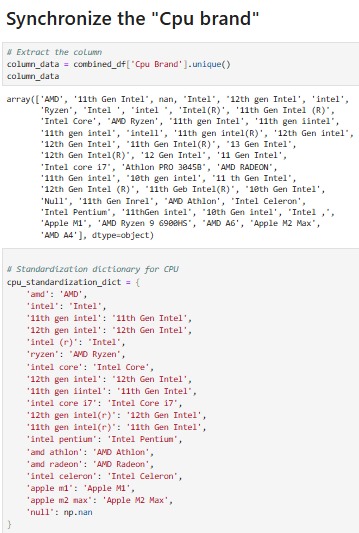




Display the updated DataFrame



* + 1. **Cpu Brand**



Display the Updated DataFrame



* + 1. **Cpu Model**



Display Updated Dataframe



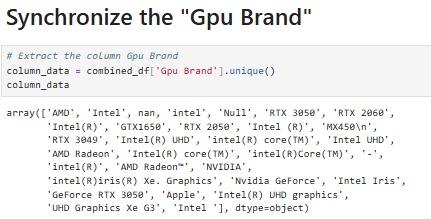


* + 1. **Hybrid**



Display updated DataFrame

* + 1. Gpu Brand

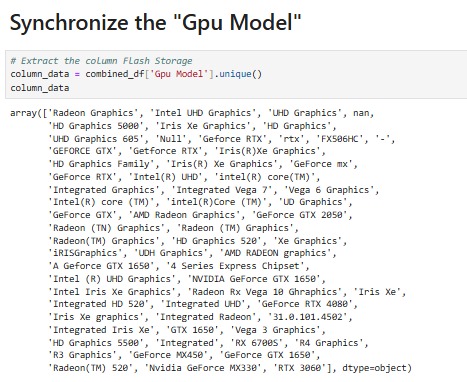


Display Updated DataFrame

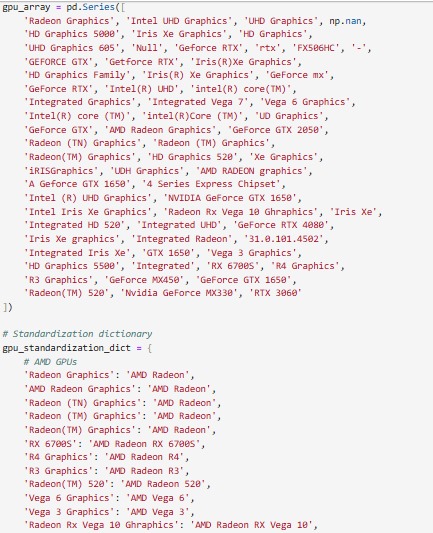


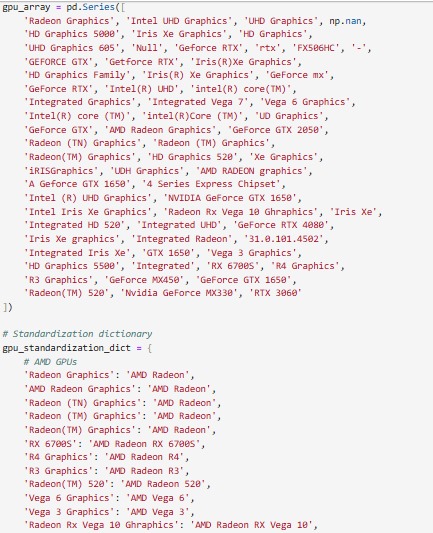


* + 1. Gpu Model



Display Updated DataFrame









* + 1. Price\_Nu

Display Updated DataFrame

