23907-MDSC-102P-ESE-REPORT

The dataset I selected for the MDSC-102P is on Laptop information.

This dataset contains the following columns:

Manufacturer – provides the name of the manufacturing company, for instance Apple

Model Name – gives the model name of the laptop, for instance MacBook Air

Category – gives the details of the laptop to which category it belongs to, for instance Notebook

Screen Size – gives size of laptop's screen in inches, for instance 13.3 inches

Screen – provides more information on laptop's screen, for instance Full HD 1920 x 1080

CPU – provides information about the processor, for instance Intel Core i7

RAM – gives information about RAM, for instance 16GB

Storage – gives information about the storage, for instance 256 GB SSD

GPU – provides more information about GPU, for instance Nvidia GeForce GTX 1070

Operating System – provides information about the OS, for instance Windows

Operating System Version – gives OS Version information, for instance 10 i.e., Windows 10

Weight – gives the weight of the laptop in kg's, for instance 1.83kg

Price – gives the price of laptop in lakhs, for instance 248900

Importing required libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

✓ 15.5s Python

Reading the dataset

```
df = pd.read_csv("E:/Siva/SSSIHL/MSc Data Science/1st Sem/102 Lab/final/laptop/laptops.csv")
df
```

✓ 0.3s Python

	Manufacturer	Model Name	Category	Screen Size	Screen	CPU	RAM	Storage	GPU	Operating System	Operating System Version	Weight	Price
0	Apple	MacBook Pro	Ultrabook	13.3"	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	NaN	1.37kg	11912523.48
1	Apple	Macbook Air	Ultrabook	13.3"	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	NaN	1.34kg	7993374.48
2	HP	250 G6	Notebook	15.6"	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	NaN	1.86kg	5112900.00
3	Apple	MacBook Pro	Ultrabook	15.4"	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	NaN	1.83kg	22563005.40
4	Apple	MacBook Pro	Ultrabook	13.3"	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	NaN	1.37kg	16037611.20
...	...	...	...	...	...	...	...	...	...	...	...	...	...
972	Dell	Alienware 17	Gaming	17.3"	Full HD 1920x1080	Intel Core i7 6700HQ 2.6GHz	32GB	256GB SSD + 1TB HDD	Nvidia GeForce GTX 1070	Windows	10	4.42kg	24897600.00
973	Toshiba	Tecra A40-C-1DF	Notebook	14.0"	Full HD 1920x1080	Intel Core i5 6200U 2.3GHz	8GB	256GB SSD	Intel HD Graphics 520	Windows	10	1.95kg	10492560.00
974	Asus	Rog Strix	Gaming	17.3"	Full HD 1920x1080	Intel Core i7 7700HQ 2.8GHz	16GB	256GB SSD + 1TB HDD	Nvidia GeForce GTX 1060	Windows	10	2.73kg	18227710.80
975	HP	Probook 450	Notebook	15.6"	IPS Panel Full HD 1920x1080	Intel Core i5 7200U 2.70GHz	8GB	128GB SSD + 1TB HDD	Nvidia GeForce 930MX	Windows	10	2.04kg	8705268.00
976	Lenovo	ThinkPad T460	Notebook	14.0"	1366x768	Intel Core i5 6200U 2.3GHz	4GB	508GB Hybrid	Intel HD Graphics 520	Windows	7	1.70kg	8909784.00

977 rows × 13 columns

Data Preprocessing

By looking at the above data frame, it looks complex with their respective data types. So, let's try to minimize the complexity and do data preprocessing to understand the dataframe with more ease.

We can achieve this by applying certain techniques for each of the attributes available.

First, let's look at the missing values (null values) present in the dataframe.

```
# Checking for null values in all columns
df.isnull().sum()
```

✓ 0.0s

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

So, we can observe that there are no null values present in the dataframe except for Operating system version attribute.

Now, here it's really tough to handle the missing values as each product is of different manufacture, OS and depends on several other factors.

We might drop the missing values, but that may lead to data loss. So, to keep the data available, let's do the following:

For Operating System version:

```
df['Operating System Version'].value_counts()
```

✓ 0.0s

```
Operating System Version
10      819
7        10
10 S      8
X         4
Name: count, dtype: int64
```

Looking at the value counts of OS version, we got to know that most of the laptops have version 10

So, to keep the data available, and the most used OS Version is 10, so let's replace missing values with 10

From above, we observe that roman language X is used, which is 10 and also 10 S is used. So, let's convert both of them into 10 and now let's check at the unique values present in the column.

```
df['Operating System Version'].unique()
```

✓ 0.0s

```
array(['10', '7'], dtype=object)
```

Unique values in the column are 10 and 7.

For Screen Size:

```python
df['Screen Size'].unique()
```
✓ 0.0s

```
array(['13.3"', '15.6"', '15.4"', '14.0"', '12.0"', '11.6"', '17.3"',
       '10.1"', '13.5"', '12.5"', '13.0"', '18.4"', '13.9"', '12.3"',
       '17.0"', '15.0"', '14.1"', '11.3"'], dtype=object)
```

It's in object type, we need to minimize this to int or float now.

```python
df['Screen Size'] = df['Screen Size'].str.replace('"','')
df['Screen Size'] = df['Screen Size'].astype('float')
```
✓ 0.0s

For Screen:

Upon checking the unique values, we can segment the values of Screen to smaller unique values basing on Touch Screen and Non-Touch Screen, so let's write a function to achieve that.

```python
def fun(display):
    if 'Touchscreen' in display:
        return 'Touch'
    else:
        return 'Non Touch'
```
✓ 0.0s

```python
df['Screen'] = df['Screen'].apply(fun)
```
✓ 0.0s

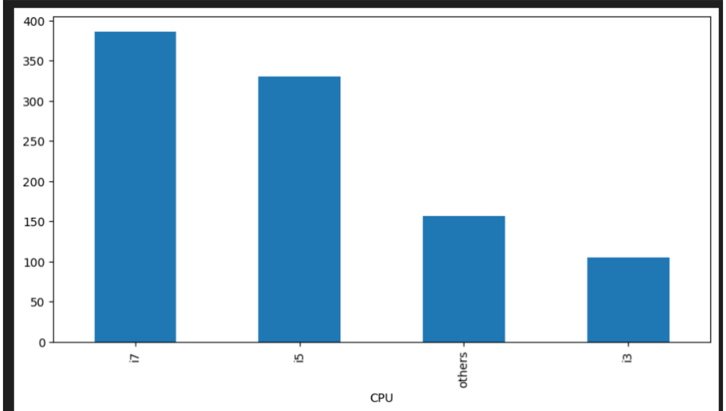
For CPU:

Upon checking the unique values, we can segment the values of CPU to smaller unique values such as i3, i5, i7 and others. Let's write a function for it and plot a bar plot for it.

```python
def fun(cpu):
    if 'i3' in cpu:
        return 'i3'
    elif 'i5'in cpu:
        return 'i5'
    elif 'i7' in cpu:
        return 'i7'
    else:
        return 'others'
```
✓ 0.0s

```python
df['CPU'] = df['CPU'].apply(fun)
```
✓ 0.0s Python

```python
plt.figure(figsize=(10,5))
df['CPU'].value_counts().plot(kind='bar')
plt.show()
```
✓ 0.5s Python



For Storage:

```
df['Storage'].unique()
✓ 0.0s

array(['128GB SSD', '128GB Flash Storage', '256GB SSD', '512GB SSD',
       '500GB HDD', '256GB Flash Storage', '1TB HDD',
       '32GB Flash Storage', '128GB SSD +  1TB HDD',
       '256GB SSD +  256GB SSD', '64GB Flash Storage',
       '256GB SSD +  1TB HDD', '256GB SSD +  2TB HDD', '32GB SSD',
       '2TB HDD', '64GB SSD', '1TB Hybrid', '512GB SSD +  1TB HDD',
       '1TB SSD', '256GB SSD +  500GB HDD', '128GB SSD +  2TB HDD',
       '512GB SSD +  512GB SSD', '16GB SSD', '16GB Flash Storage',
       '512GB SSD +  256GB SSD', '512GB SSD +  2TB HDD',
       '64GB Flash Storage +  1TB HDD', '1GB SSD', '1TB HDD +  1TB HDD',
       '32GB HDD', '1TB SSD +  1TB HDD', '512GB Flash Storage',
       '128GB HDD', '240GB SSD', '8GB SSD', '508GB Hybrid'], dtype=object)
```

Let's write a function to segment the values into unique values like SSD, HDD, Flash, and others.

```
def fun(storage):
    if 'SSD' in storage:
        return 'SSD'
    elif 'HDD'in storage:
        return 'HDD'
    elif 'Flash' in storage:
        return 'Flash'
    else:
        return 'others'
```

For RAM:

The unique values present are 2GB, 4GB, 8GB, 12GB, 16GB, 24GB so on. As we are aware that RAM values are of GB, let's convert them into numerical and into integer data type.

```
df['RAM'].unique()
✓ 0.0s

array(['8GB', '16GB', '4GB', '2GB', '12GB', '6GB', '32GB', '24GB'],
      dtype=object)
```

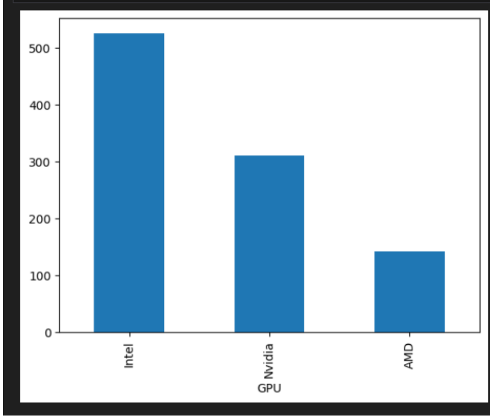
```
df['RAM'] = df['RAM'].str.split('GB').str[0]
df['RAM'] = df['RAM'].astype('int')
✓ 0.0s
```

For GPU:

Upon checking the unique values, we can segment the values of GPU to smaller unique values such as Nvidia, AMD, Intel, and others. Let's write a function for it and plot bar plot for it.

```
def fun(gpu):
    if 'Nvidia' in gpu:
        return 'Nvidia'
    elif 'AMD' in gpu:
        return 'AMD'
    elif 'Intel' in gpu:
        return 'Intel'
    else:
        return 'others'
✓ 0.0s
```

```
df['GPU'].value_counts().plot(kind='bar')
plt.show()
✓ 0.7s
```



For Operating System:

```
df['Operating System'].value_counts()
✓ 0.0s

Operating System
Windows      837
No OS         52
Linux         48
Chrome OS     22
macOS         13
Mac OS         4
Android        1
Name: count, dtype: int64
```

As we can observe macOS and Mac OS, let's combine them both with Mac OS by replacing macOS with Mac OS.

```
df['Operating System'] = df['Operating System'].str.replace('macOS','Mac OS')
```

```
df['Operating System'].value_counts()
✓ 0.0s

Operating System
Windows      837
No OS         52
Linux         48
Chrome OS     22
Mac OS        17
Android        1
Name: count, dtype: int64
```

For Weight:

As we know that all laptops weigh in kg's only, so let's extract the numerical value & convert to float.

```
df['Weight'].str.split('kg').str[0].unique()
```

```
df['Weight'] = df['Weight'].astype('float')
```

For Price:

```
df['Price']
✓ 0.0s

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

As most of the laptop prices will be in lakhs and not crores, so let’s get price in lakhs.

```
df['Price'].astype('str').str[:7]
```

Now let’s convert it into integer data type.

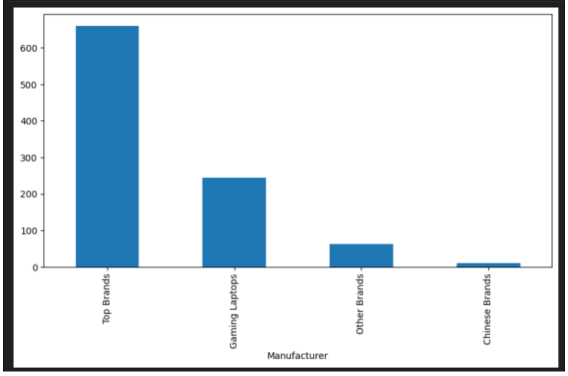
```
df['Price'] = df['Price'].astype('int')
```

For Manufacturer:

```
df['Manufacturer'].unique()
✓ 0.0s

array(['Apple', 'HP', 'Acer', 'Asus', 'Dell', 'Lenovo', 'Chuwi', 'MSI',
       'Microsoft', 'Toshiba', 'Huawei', 'Xiaomi', 'Vero', 'Razer',
       'Mediacom', 'Samsung', 'Google', 'Fujitsu', 'LG'], dtype=object)
```

We can observe many company names here, so let's classify them as following using a function and plot it using bar plot.

```
plt.figure(figsize=(10,5))
df['Manufacturer'].value_counts().plot(kind='bar')
plt.show()
✓ 0.5s                                                        Python
```




```
def fun(manu):
    if manu in ['Apple', 'HP', 'Dell', 'Lenovo', 'Microsoft']:
        return 'Top Brands'
    elif manu in ['Acer', 'Asus', 'MSI', 'Razer']:
        return 'Gaming Laptops'
    elif manu in ['Chuwi', 'Huawei', 'Xiaomi', 'Vero']:
        return 'Chinese Brands'
    else:
        return 'Other Brands'
✓ 0.0s
```

For Category:

```
df['Category'].unique()
✓ 0.0s

array(['Ultrabook', 'Notebook', 'Netbook', 'Gaming', '2 in 1 Convertible',
       'Workstation'], dtype=object)
```

We need not perform any changes to the values present in Category.

For Model Name:

Since we have all features required (manufacturer and category) and there are a greater number of unique values in Model Name and also this feature is of less importance when compared to others. So, upon this basis, we don't need this column, so let's drop it from the dataframe.

```
df.drop('Model Name',axis=1,inplace=True)
```

So, all the columns have been minimized for better understanding and converted to optimal data types, let's look at the dataframe and the data types of the columns.

Dataframe:

```
df
✓ 0.0s
```

	Manufacturer	Category	Screen Size	Screen	CPU	RAM	Storage	GPU	Operating System	Operating System Version	Weight	Price
0	Top Brands	Ultrabook	13.3	Non Touch	i5	8	SSD	Intel	Mac OS	10	1.37	1191252
1	Top Brands	Ultrabook	13.3	Non Touch	i5	8	Flash	Intel	Mac OS	10	1.34	7993374
2	Top Brands	Notebook	15.6	Non Touch	i5	8	SSD	Intel	No OS	10	1.86	5112900
3	Top Brands	Ultrabook	15.4	Non Touch	i7	16	SSD	AMD	Mac OS	10	1.83	2256300
4	Top Brands	Ultrabook	13.3	Non Touch	i5	8	SSD	Intel	Mac OS	10	1.37	1603761
...	...	...	...	...	...	...	...	...	...	...	...	...
972	Top Brands	Gaming	17.3	Non Touch	i7	32	SSD	Nvidia	Windows	10	4.42	2489760
973	Other Brands	Notebook	14.0	Non Touch	i5	8	SSD	Intel	Windows	10	1.95	1049256
974	Gaming Laptops	Gaming	17.3	Non Touch	i7	16	SSD	Nvidia	Windows	10	2.73	1822771
975	Top Brands	Notebook	15.6	Non Touch	i5	8	SSD	Nvidia	Windows	10	2.04	8705268
976	Top Brands	Notebook	14.0	Non Touch	i5	4	others	Intel	Windows	7	1.70	8909784

977 rows × 12 columns

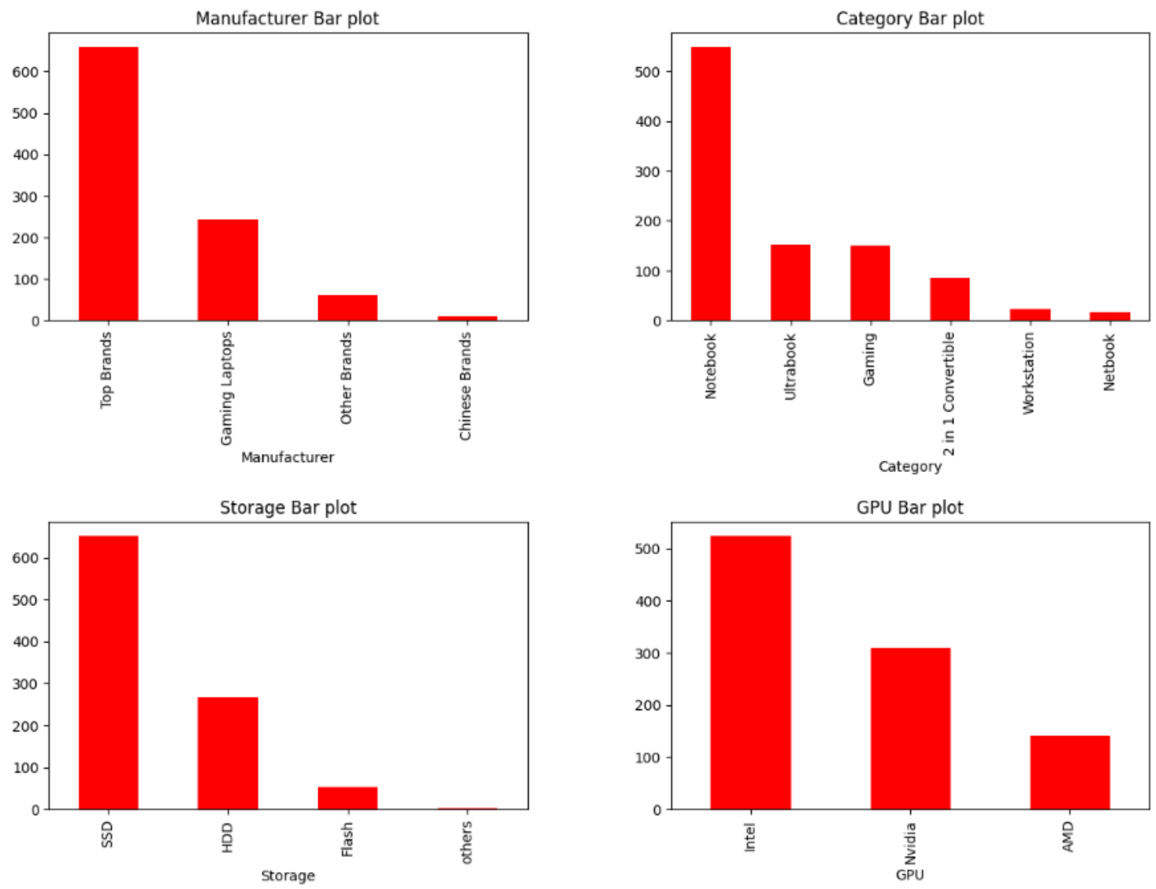
Data types of the columns:

```
df.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 977 entries, 0 to 976
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Manufacturer              977 non-null    object 
 1   Category                  977 non-null    object 
 2   Screen Size               977 non-null    float64
 3   Screen                    977 non-null    object 
 4   CPU                       977 non-null    object 
 5   RAM                       977 non-null    int32  
 6   Storage                   977 non-null    object 
 7   GPU                       977 non-null    object 
 8   Operating System          977 non-null    object 
 9   Operating System Version  977 non-null    object 
 10  Weight                    977 non-null    float64
 11  Price                     977 non-null    int32  
dtypes: float64(2), int32(2), object(8)
memory usage: 84.1+ KB
```

Now, as we are done with preprocessing, let's perform Exploratory Data Analysis using certain visualizations like bar plot, histogram, pie chart, box plot, scatter plot etc.

Exploratory Data Analysis

Bar Plot for all the categorical attributes(object) along with their inferences.

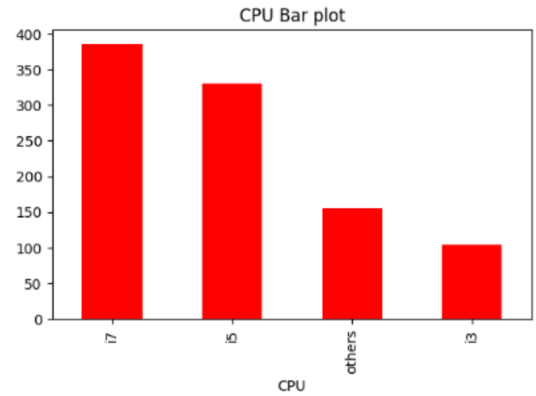


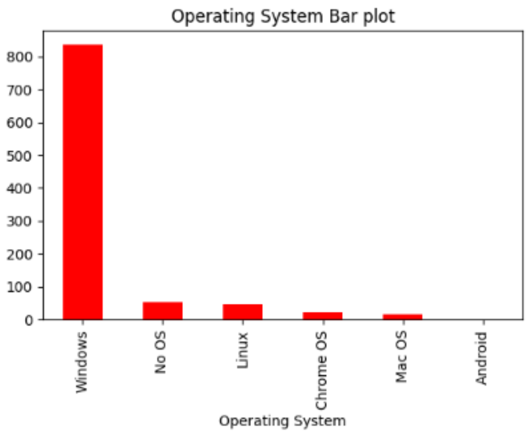
From the above snippet, we can observe that currently in the market, manufacturing of the top brand laptops (Apple, HP, Dell, Lenovo) are more than the gaming laptops or the other brands.

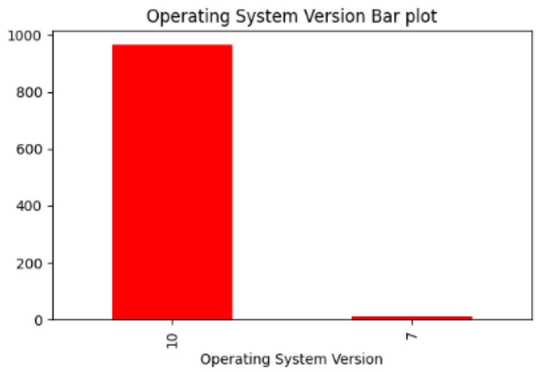
Also, more people are preferring notebooks with Intel processor which has SSD storage.

We can also infer that less population is going for AMD when compared with Nvidia, whereas when we compare Nvidia and Intel, more people are going for Intel Processor.

Day by day, the usage of Flash as storage is being reduced, and good amount of people are preferring for external storage like HDD, but most of the people are opting for SSD.



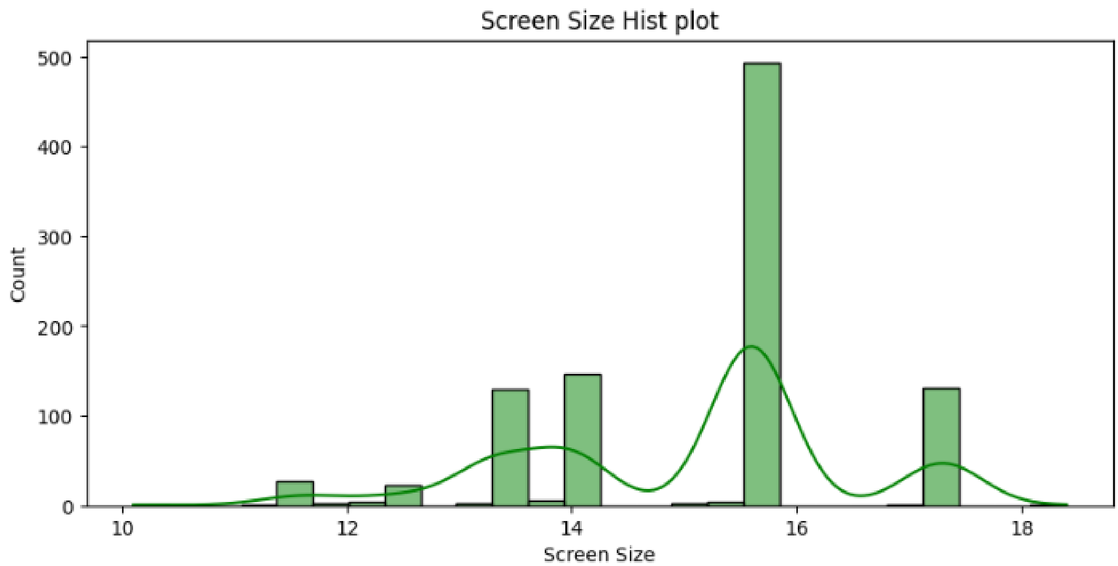


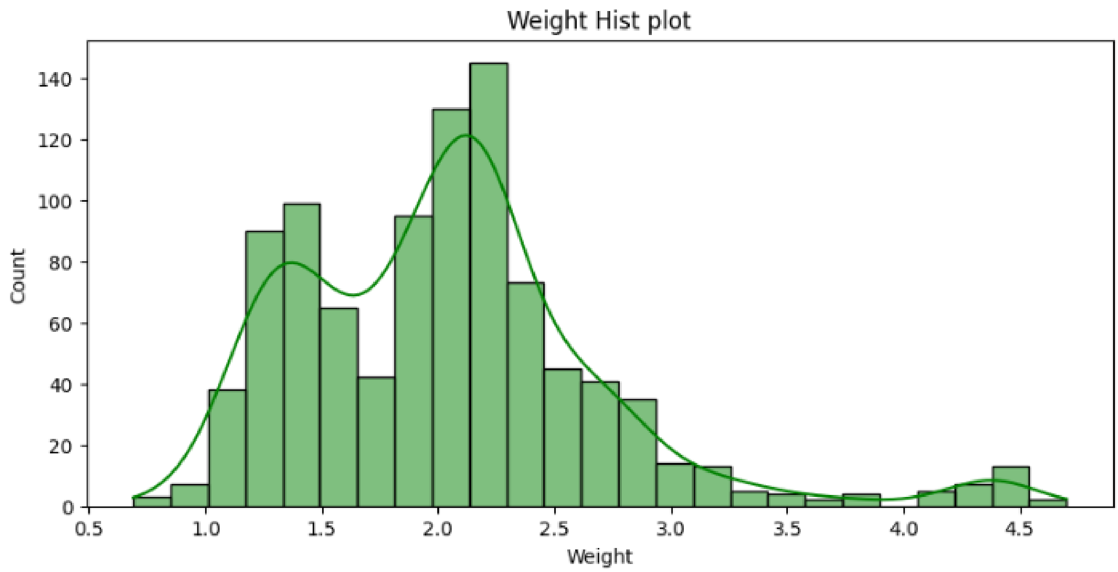


From the above snippet, we can observe that currently in the market, more preference of laptop screen is for non-touch when compared to touch screen laptops and with i7 processor as the latest trend and promising processor when compared to others. At start i3 use to be promising, as technology advancements taken place, i5 emerged and now its i7era and the i9 will be in the near future. This will continue gradually as new promising processor comes in to market.

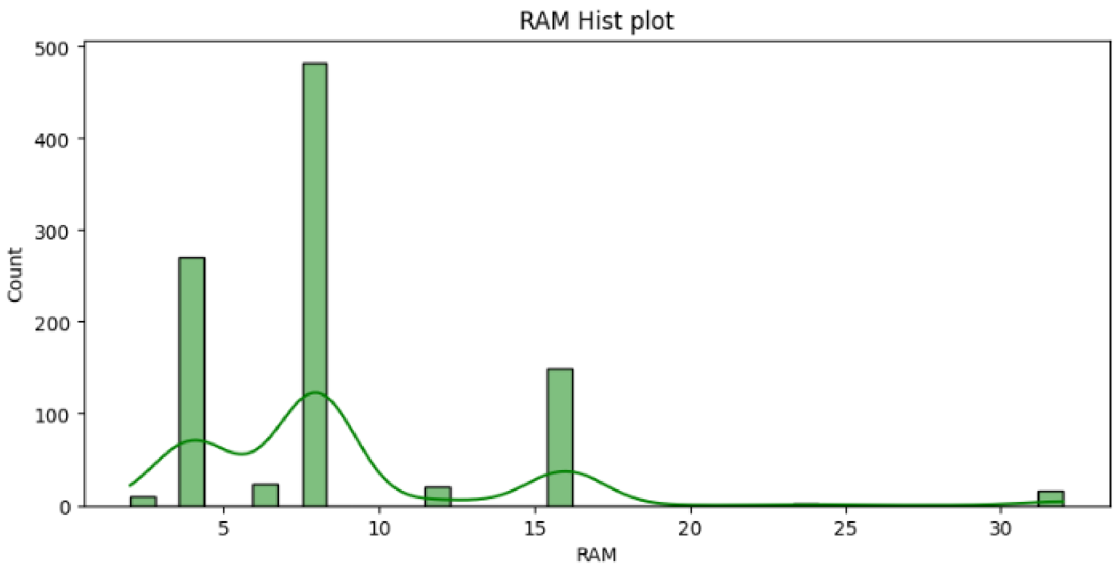
To talk about the popular operating system and its version, as we all are aware and from the above bar plot is true that Windows is the leading trader in Operating systems with Mac Os in the second place and the stable and promising version of Operating System Version of Windows is 10 today. As days passes, new versions will be coming into the picture, and this will be carried forward.

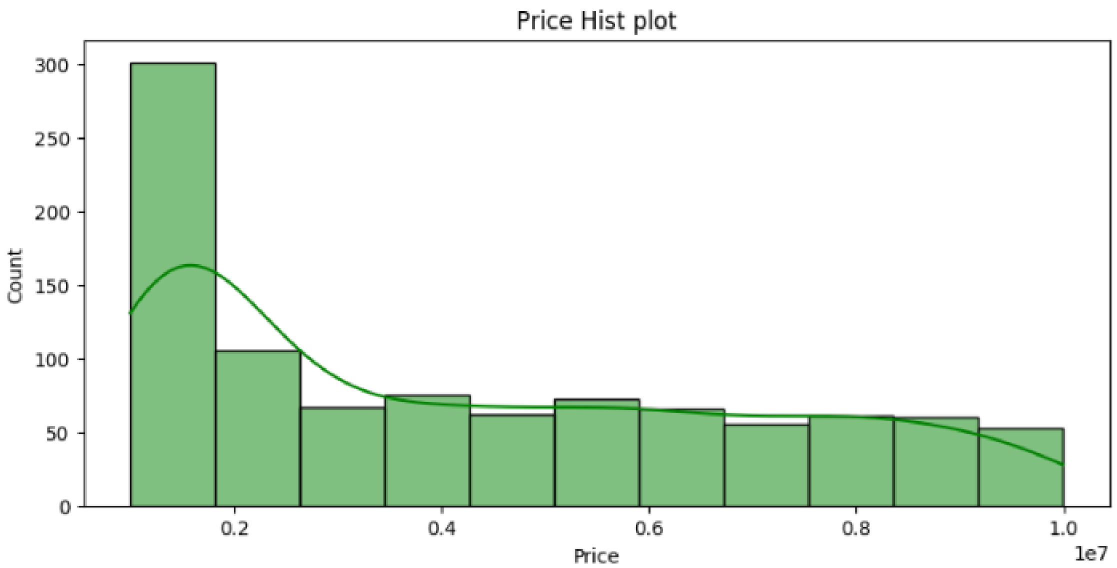
Histogram (Hist Plot) for all the continuous attributes (int and float) along with their inferences.





From the above snippet, it's clear that the Screen Size of laptops is from 10 to 19 approximately in today's market. And most consumers are preferring 15inches laptop as their top preference where people who require minimal screen are going for 13inches laptop and people who prefer larger screen size are preferring around 17inches laptop and the weight of the laptop is ranging from 1kg to 5kg, where most customers are preferring ideal weight around 2.3kgs.



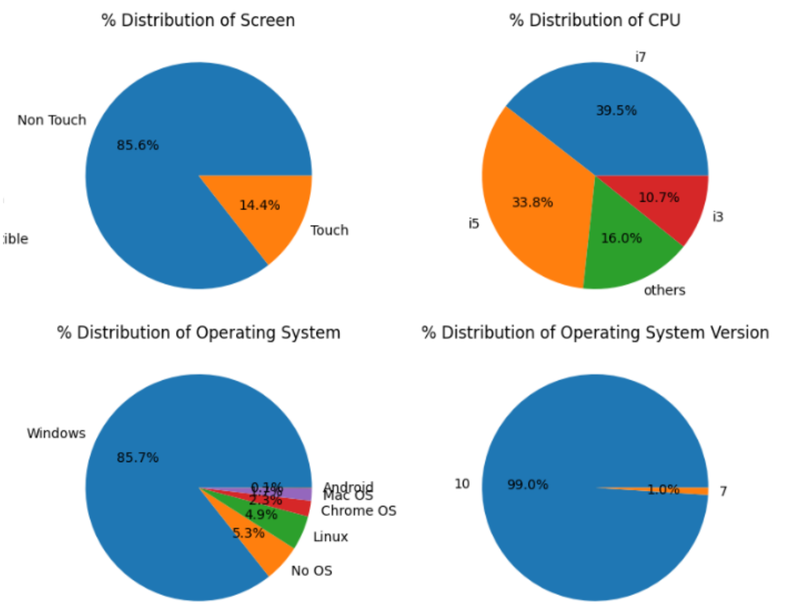


From the above snippet, we can infer that RAM Capacity in the laptop generally ranges from 4 to 32, and we can see that few customers who have brought early has RAM storage around 4GB, and it moved on to 8GB where currently most users are using and few prefer 8GB and extra 4GB in the second slot to make it 12, and others who need high performance for instance gaming, these category of people go for 16GB of RAM, also for people if necessary they go for 32 too.

Most of the buyers prefer budget friendly laptop for general usage, and the people who require for a specific usage, will go for a high-end laptop where it costs more.

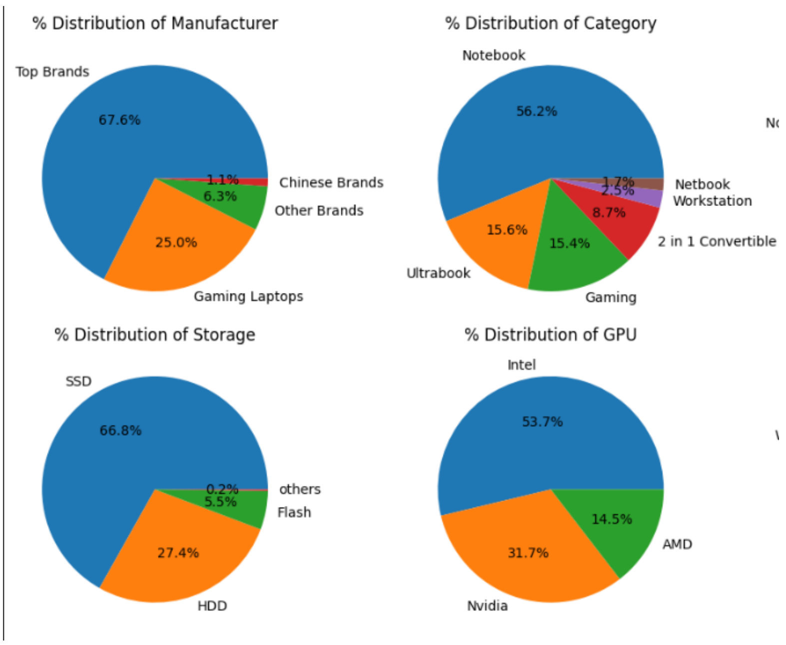
Pie-chart for all the categorical attributes:

As we have discussed earlier, we can draw similar inferences but now in percentage for each of the item using a pie chart.



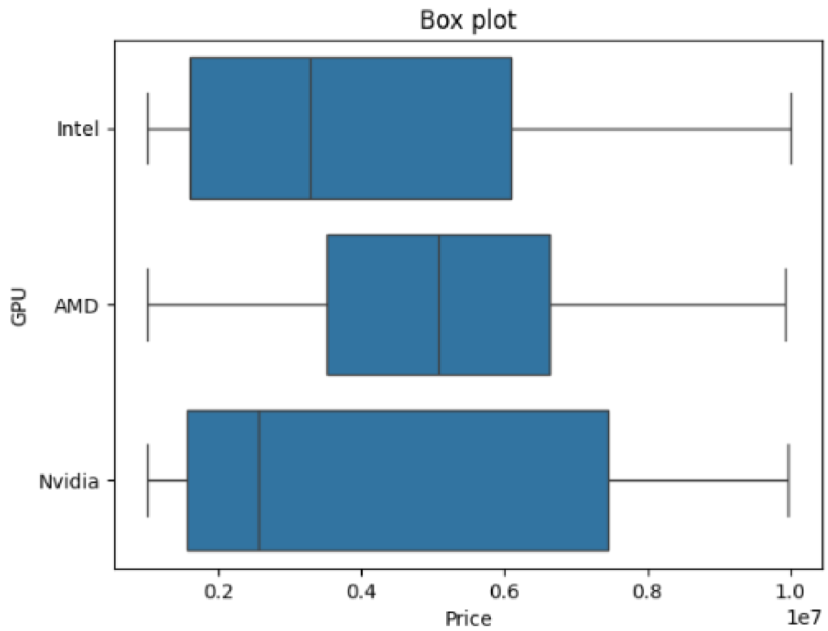
Here, the percentage distribution of screen is 85.6% Non-Touch Screen and the other 14.6% who go for Touch Screen.

Many buyers are preferring i7 than all other CPU Processors, where they go for Windows Operating System of version10.

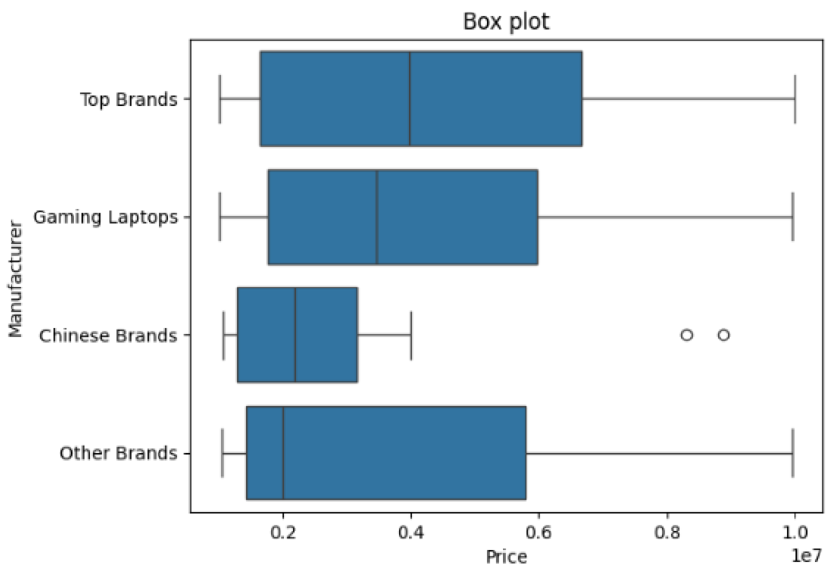


Coming to manufacturing, more top brands laptops are getting manufactured than any other laptops where more notebooks are getting manufactured for daily usage with SSD storage with Intel Processor.

Box Plot for few attributes w.r.t price and their inferences:



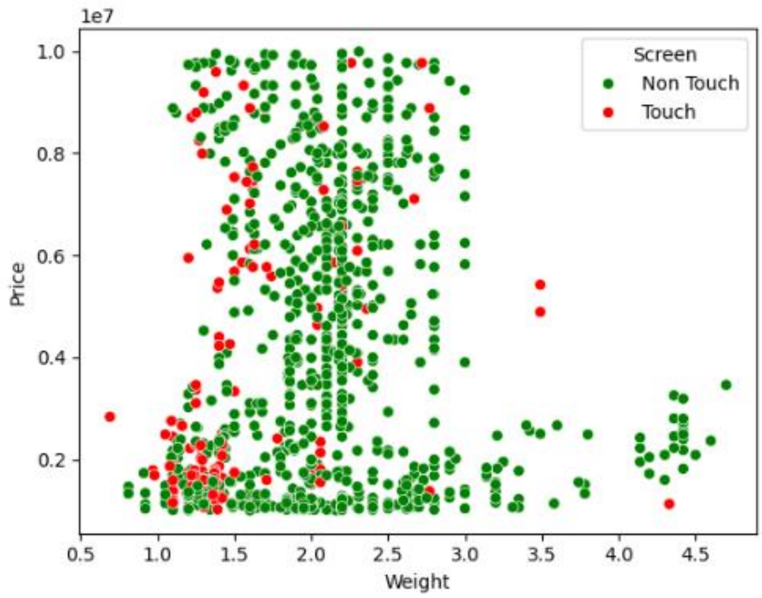
From this box plot, we can observe that generally laptops with Intel GPU starts from low price to mid-price, with more products of comparatively less price, and AMD maintains certain standards with it's price ranges at a good cost, whereas Nvidia starts with it basic variants to the high end one's.



From this box plot, the major observation is w.r.t the Chinese brands where the cost generally ranges low, but there are a few with high prices too.

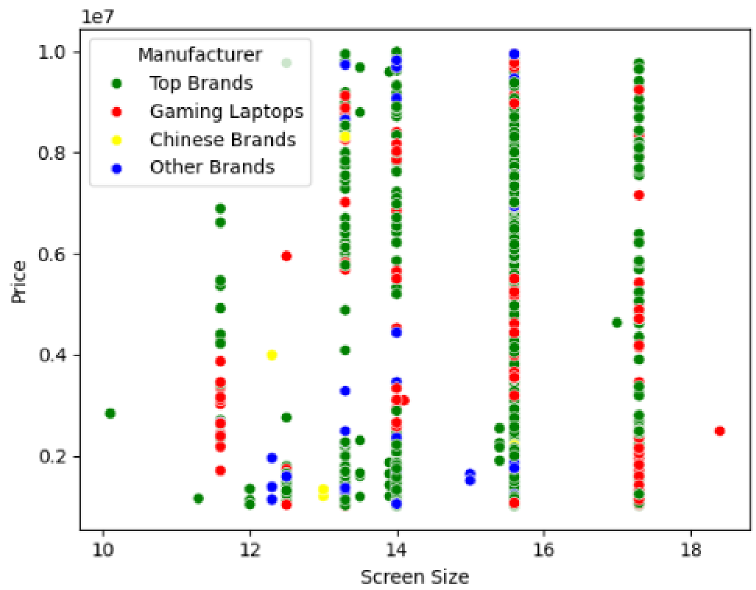
Scatter Plot for few attributes w.r.t price and their inferences:

Scatter Plot is done for two continuous variables by taking one on the x-axis and the other on y-axis based on other variable (hue).



This scatter plot is based on Weight and Price of the laptops based on hue Screen.

We can infer that, generally the weight of the touch screen laptop generally ranges low, as they are used for commercial purpose for instance office etc. And Non touch laptops are distributed across many weights and price ranges



This scatter plot is based on Screen Size and Price of the laptops based on hue Screen.

We can infer that, most of the top brands laptops Screen Size is nearly 15inches, but there also equivalently many distributed between 11,13 and 17inches.

Now, let's try to perform some **statistical tests**, but before doing that let's select the continuous attributes and normalize them using boxcox transformation.

Before performing boxcox transformation, let's check the skewness.

```
cont_df= df.select_dtypes(include=['int', 'float'])
✓ 0.0s

cont_df.skew()
✓ 0.0s

Screen Size   -0.397473
RAM            2.078221
Weight         1.184921
Price          0.507658
dtype: float64
```

Here, we can see the skewness of each continuous attribute. Now, we should try to find the lambda values and then apply the boxcox transformation.

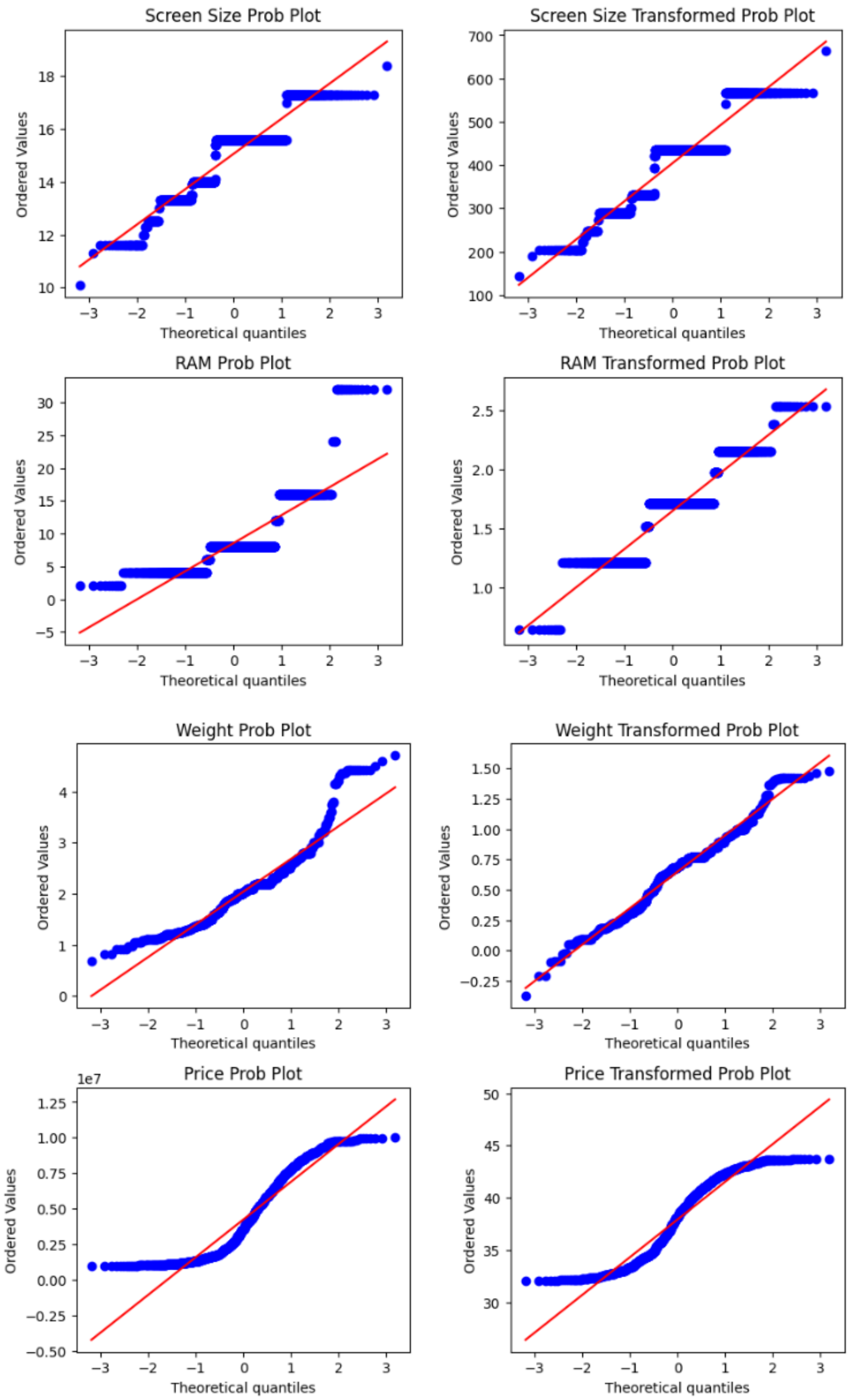
```
import scipy as scipy
from scipy import stats
# Using boxcox and finding the lambda value for all the attributes
transformed_df = pd.DataFrame()
lam_values = {}

for feature in cont_df.columns:
    x = cont_df[feature]
    arr, lam = stats.boxcox(x)
    transformed_df[feature] = arr
    lam_values[feature] = lam

# Printing lambda values for each numeric feature
for feature, lam in lam_values.items():
    print("{}: {}".format(feature, lam))
✓ 0.0s

Screen Size: 2.5530744481914813
RAM: -0.19069331965440928
Weight: -0.05791356131221097
Price: 0.10846911482608998
```

We got the lambda values, now let's perform boxcox transformation to find the required values.

```
transformed_skewval = {}

for feature in cont_df.columns:
    lam = lam_values[feature]
    boxcox = ( (cont_df[feature] ** lam) - 1 ) / lam
    transformed_skewval[feature] = boxcox.skew()

for feature, boxcox in transformed_skewval.items():
    print("{}: {}".format(feature, boxcox))
✓ 0.0s

Screen Size: -0.03831087297389192
RAM: 0.005253033337232968
Weight: -0.00041547735459021134
Price: -0.0383403858639293
```

Normalized values

Let’s look at the ***probability plot*** below to see the difference before and after transformation.



Here, we can clearly see the difference between before and after transformation of each attribute with its probability plot.

```python
# To estimate μ when variance is not known and n is large.

alpha = 0.05
interval_estimation_z = {}

for column in cont_df.columns:
    mean = cont_df[column].mean()
    n = len(cont_df)
    sample_variance = cont_df[column].var()
    standard_deviation = math.sqrt(sample_variance)
    z_value = stats.norm.ppf(1 - (1 - alpha) / 2)
    rhs = z_value * standard_deviation
    lb = mean - rhs
    ub = mean + rhs
    interval_estimation_z[column] = (lb, ub)
```
✓ 0.0s

```python
for column, interval in interval_estimation_z.items():
    print(f'{column} CI: ({interval[0]}, {interval[1]})')
```
✓ 0.0s

```
Screen Size CI: (14.963632275302528, 15.141587786109957)
RAM CI: (8.214771103835686, 8.8415236761029)
Weight CI: (1.9973646412201755, 2.080891244143182)
Price CI: (4068046.576725168, 4417597.951422222)
```

Now, let's find the interval in which the mean ranges for each of the attribute. And we cross check it with mean of each attribute.

```python
cont_df.mean()
```
✓ 0.0s

```
Screen Size    1.505261e+01
RAM            8.528147e+00
Weight         2.039128e+00
Price          4.242822e+06
dtype: float64
```

```python
# To estimate Variance

alpha = 0.05
interval_estimation_var = {}

for column in cont_df.columns:
    mean = cont_df[column].mean()
    n = len(cont_df)
    sample_variance = cont_df[column].var()
    chi_lower = stats.chi2.ppf(alpha / 2, n-1)
    chi_upper = stats.chi2.ppf(1 - alpha / 2, n-1)
    lb = ((n - 1) * sample_variance) / chi_upper
    ub = ((n - 1) * sample_variance) / chi_lower
    interval_estimation_var[column] = (lb, ub)

print('chi_lower',chi_lower)
print('chi_upper',chi_upper)
```
✓ 0.0s

```
chi_lower 891.3155497191265
chi_upper 1064.472503438975
```

```python
for column, interval in interval_estimation_var.items():
    print(f'{column} CI: ({interval[0]}, {interval[1]})')
```
✓ 0.0s

```
Screen Size CI: (1.8460747817093386, 2.2047139703115746)
RAM CI: (22.899112816936437, 27.347751258751302)
Weight CI: (0.4067017528480328, 0.48571219602708426)
Price CI: (7122762040843.634, 8506509668103.714)
```

Now, let's find the interval in which the variance ranges for each of the attribute and display the chi square values (lower & upper). And we cross check it with variance of each attribute.

```python
cont_df.var()
```
✓ 0.0s

```
Screen Size    2.013418e+00
RAM            2.497487e+01
Weight         4.435685e-01
Price          7.768427e+12
dtype: float64
```

Now, let's write a function for the following ***testing hypothesis***:

$H_0 : \mu = \mu 0 vs H_1 : \mu \neq \mu 0$

```
def Testing_Hypothesis(array,alpha,µ0):
    n = len(array)
    mean = array.mean()
    var = array.var()
    z_cal = (mean - µ0) / (np.sqrt(var / n))
    p = 2 * (1 - stats.norm.cdf(np.abs(z_cal)))

    if p < alpha:
        print('Reject µ0')
    else:
        print('Do not reject µ0')
✓ 0.0s
```

This function takes the array as the input where we will pass the values of an attribute with level of significance alpha and with µ0

Now, let's perform testing hypothesis for the following:

```
Testing_Hypothesis(df['Weight'],0.05,1.56)
✓ 0.0s
Reject µ0
```

```
Testing_Hypothesis(df['Weight'],0.1,3.56)
✓ 0.0s
Reject µ0
```

```
Testing_Hypothesis(df['Price'],0.05,4231125)
✓ 0.0s
Do not reject µ0
```

```
Testing_Hypothesis(df['Price'],0.05,223000)
✓ 0.0s
Reject µ0
```

```
Testing_Hypothesis(df['Screen Size'],0.1,12)
✓ 0.0s
Reject µ0
```

```
Testing_Hypothesis(df['Screen Size'],0.05,15.10)
✓ 0.0s
Do not reject µ0
```

```
Testing_Hypothesis(df['RAM'],0.05,8)
✓ 0.0s
Reject µ0
```

```
Testing_Hypothesis(df['RAM'],0.1,16)
✓ 0.0s
Reject µ0
```

Now, let's write a function for the following ***testing hypothesis***:

$$H_0 : \sigma 0 = \sigma 2 vs H_1 : \sigma 0 \neq \sigma 2$$

```python
def Testing_Hypothesis_Var(array, alpha, sigma0):
    n = len(array)
    var = array.var(ddof=1)
    chi_square = (n - 1) * var / sigma0
    p = 1 - stats.chi2.cdf(chi_square, df=n - 1)

    if p < alpha:
        print('Reject σ^2')
    else:
        print('Do not reject σ^2')
```
✓ 0.0s

This function takes the array as the input where we will pass the values of an attribute with level of significance alpha and with σ0

Now, let's perform testing hypothesis for the following:

```python
Testing_Hypothesis_Var(df['Weight'], 0.05, 2.56)
```
✓ 0.0s

```
Do not reject σ^2
```

```python
Testing_Hypothesis_Var(df['Price'], 0.05, 784692.45)
```
✓ 0.0s

```
Reject σ^2
```

```python
Testing_Hypothesis_Var(df['Screen Size'], 0.1, 13)
```
✓ 0.0s

```
Do not reject σ^2
```

```python
Testing_Hypothesis_Var(df['RAM'], 0.1, 16)
```
✓ 0.0s

```
Reject σ^2
```