Laptop prising

EDA

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')


# Read the file
df=pd.read_csv("/content/laptop - laptop.csv.csv")


df.head()
```

	Unnamed: 0.1	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram
0	0	0.0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	
1	1	1.0	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	
2	2	2.0	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	
3	3	3.0	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	
4	4	4.0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.shape
# This dataset is having
# column:-1303
# row:- 13
```

```
(1303, 13)
```

```
# Dataset information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0.1      1303 non-null   int64  
 1   Unnamed: 0        1273 non-null   float64
 2   Company           1273 non-null   object 
 3   TypeName          1273 non-null   object 
 4   Inches            1273 non-null   object 
 5   ScreenResolution  1273 non-null   object 
 6   Cpu               1273 non-null   object 
 7   Ram(GB)           1273 non-null   float64
 8   Memory            1273 non-null   object 
 9   Gpu               1273 non-null   object 
 10  OpSys             1273 non-null   object 
 11  Weight (kg)       1273 non-null   object 
 12  Price             1273 non-null   float64
dtypes: float64(3), int64(1), object(9)
memory usage: 132.5+ KB
```

```
df = df.drop(columns=['Unnamed: 0.1'])
df = df.drop(columns=['Unnamed: 0'])
```

Here unnamed column 0.1 have 1303 rows but other columns have 1273 rows, means several rows are null

```
# Checking for missing/null value
df.isnull().sum()
```

	0
Company	30
TypeName	30
Inches	30
ScreenResolution	30
Cpu	30
Ram(GB)	30
Memory	30
Gpu	30
OpSys	30
Weight (kg)	30
Price	30

dtype: int64

DATA PRE-PROCESSING

```
# Drop null rows
# Store all edited data in new dataframe called df1
df1=df.dropna()
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1273 entries, 0 to 1302
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Company           1273 non-null   object 
 1   TypeName          1273 non-null   object 
 2   Inches            1273 non-null   object 
 3   ScreenResolution  1273 non-null   object 
 4   Cpu               1273 non-null   object 
 5   Ram(GB)           1273 non-null   float64
 6   Memory            1273 non-null   object 
 7   Gpu               1273 non-null   object 
 8   OpSys             1273 non-null   object 
 9   Weight (kg)       1273 non-null   object 
 10  Price             1273 non-null   float64
dtypes: float64(2), object(9)
memory usage: 119.3+ KB
```

```
# Reset the index actully pandas created dataframe with 1303 indexs
# but now we have drop the null rows so we have values upto 1273,
# so to match the values and index ,reset the index
df1 = df1.reset_index(drop=True)
```

```
df1.shape
```

```
(1273, 11)
```

Datatype of weight is object we have to change it in float

```
# One value in weighr(kg) column is '?' so we have to replace it with null
# Replace '?' with NaN in the 'Weight (kg)' column
df1['Weight (kg)'] = df1['Weight (kg)'].replace('?', np.nan)

# Now convert the 'Weight (kg)' column to float
df1['Weight (kg)'] = df1['Weight (kg)'].astype(float)

# Display info to confirm the change
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1273 entries, 0 to 1272
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Company           1273 non-null   object 
 1   TypeName          1273 non-null   object 
 2   Inches            1273 non-null   object 
 3   ScreenResolution  1273 non-null   object 
 4   Cpu               1273 non-null   object 
 5   Ram(GB)           1273 non-null   float64
 6   Memory            1273 non-null   object 
 7   Gpu               1273 non-null   object 
 8   OpSys             1273 non-null   object 
 9   Weight (kg)       1272 non-null   float64
 10  Price             1273 non-null   float64
dtypes: float64(3), object(8)
memory usage: 109.5+ KB
```

Datatype of Inches is object we have to change it in float

```
# One value in Inches column is '?' so we have to replace it with null
# Replace '?' with NaN in the 'Inches' column
df1['Inches'] = df1['Inches'].replace('?', np.nan)

# Now convert the 'Inches' column to float
```

```
df1['Inches'] = df1['Inches'].astype(float)

# Display info to confirm the change
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1273 entries, 0 to 1272
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Company           1273 non-null   object 
 1   TypeName          1273 non-null   object 
 2   Inches            1272 non-null   float64
 3   ScreenResolution  1273 non-null   object 
 4   Cpu               1273 non-null   object 
 5   Ram(GB)           1273 non-null   float64
 6   Memory            1273 non-null   object 
 7   Gpu               1273 non-null   object 
 8   OpSys             1273 non-null   object 
 9   Weight (kg)       1272 non-null   float64
 10  Price             1273 non-null   float64
dtypes: float64(4), object(7)
memory usage: 109.5+ KB
```

Checking for unique values

```
df1.nunique()
```

	0
Company	19
TypeName	6
Inches	24
ScreenResolution	40
Cpu	118
Ram(GB)	10
Memory	40
Gpu	106
OpSys	9
Weight (kg)	180
Price	777

dtype: int64

Handling Duplicated

```
df1.duplicated().sum() # no duplicate
```

```
np.int64(29)
```

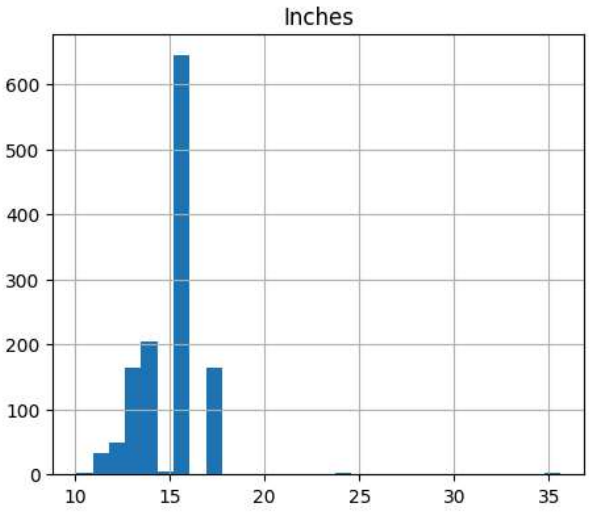
Discription of the dataframe

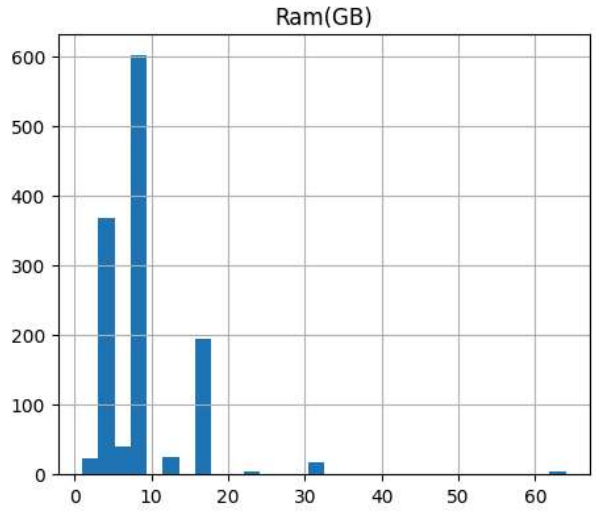
```
#Discription of numerical columns
df1.describe()
```

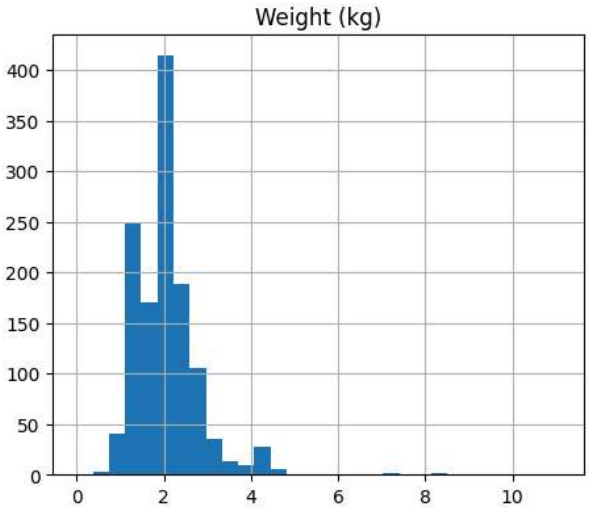
	Inches	Ram(GB)	Weight (kg)	Price
count	1272.000000	1273.000000	1272.000000	1273.000000
mean	15.130818	8.462687	2.077618	59955.804713
std	1.954436	5.564408	0.807808	37332.250492
min	10.100000	1.000000	0.000200	9270.700000
25%	14.000000	4.000000	1.500000	31914.700000
50%	15.600000	8.000000	2.040000	52161.100000
75%	15.600000	8.000000	2.320000	79333.400000
max	35.600000	64.000000	11.100000	324954.700000

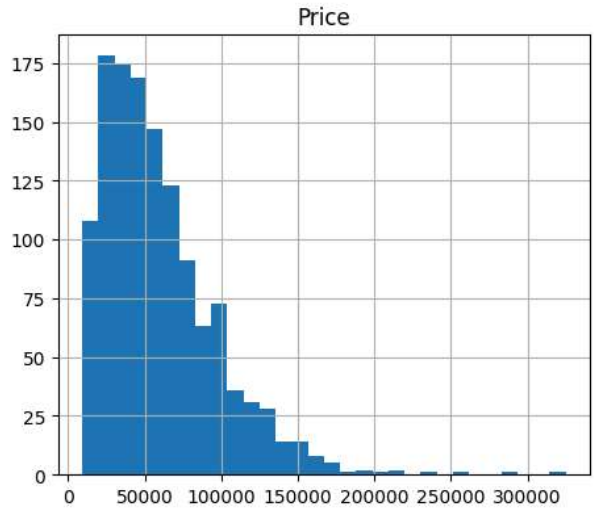
```
# graphical representation of numerical column
df1.hist(figsize=(12,10),bins=30)
plt.suptitle("Numerical feature discription",fontsize=16)
plt.show()
```

Numerical feature discription









```
#categorical columns
df1.select_dtypes(exclude=('int','float')).columns
```

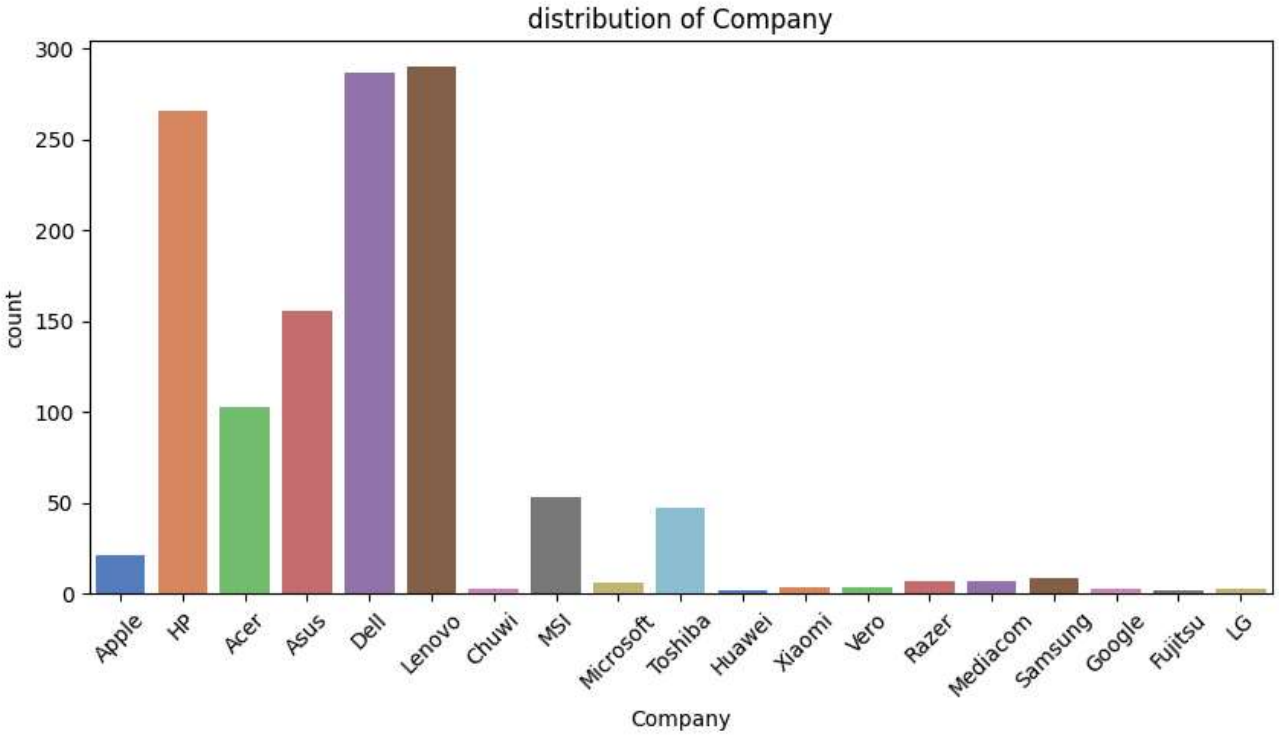
```
Index(['Company', 'TypeName', 'ScreenResolution', 'Cpu', 'Memory', 'Gpu',
       'OpSys'],
      dtype='object')
```

```
#Discription of categorical columns
df1.describe(include=['object'])
```

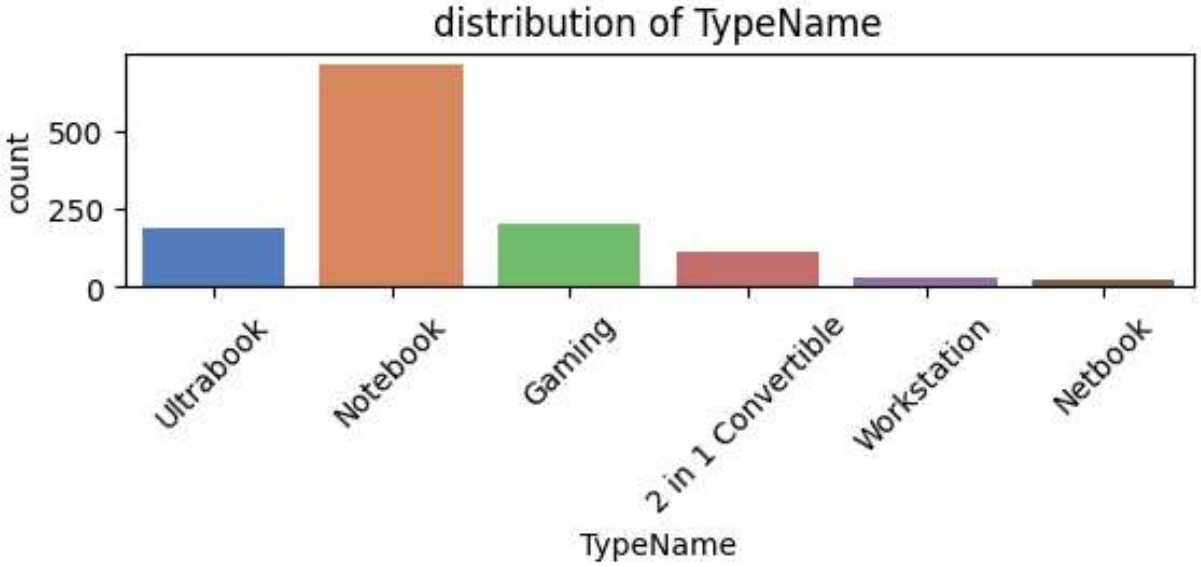
	Company	TypeName	ScreenResolution	Cpu	Memory	Gpu	OpSys
count	1273	1273	1273	1273	1273	1273	1273
unique	19	6	40	118	40	106	9
top	Lenovo	Notebook	Full HD 1920x1080	Intel Core i5 7200U	256 SSD	Intel HD Graphics 620	Windows 10

```
#Graphical representation of categorical columns
cat_col=['Company', 'TypeName','OpSys']
plt.figure(figsize=(10, 16)) # Adjust figure size based on number of rows


for i,col in enumerate(cat_col,1):
  plt.subplot(3,1, i) # Create a subplot in the grid
  plt.suptitle("Categorical feature discription",fontsize=20)
  sns.countplot(data=df1,x=col,palette="muted")
  plt.title(f"distribution of {col}")
  plt.xticks(rotation=45)
  plt.show()
```

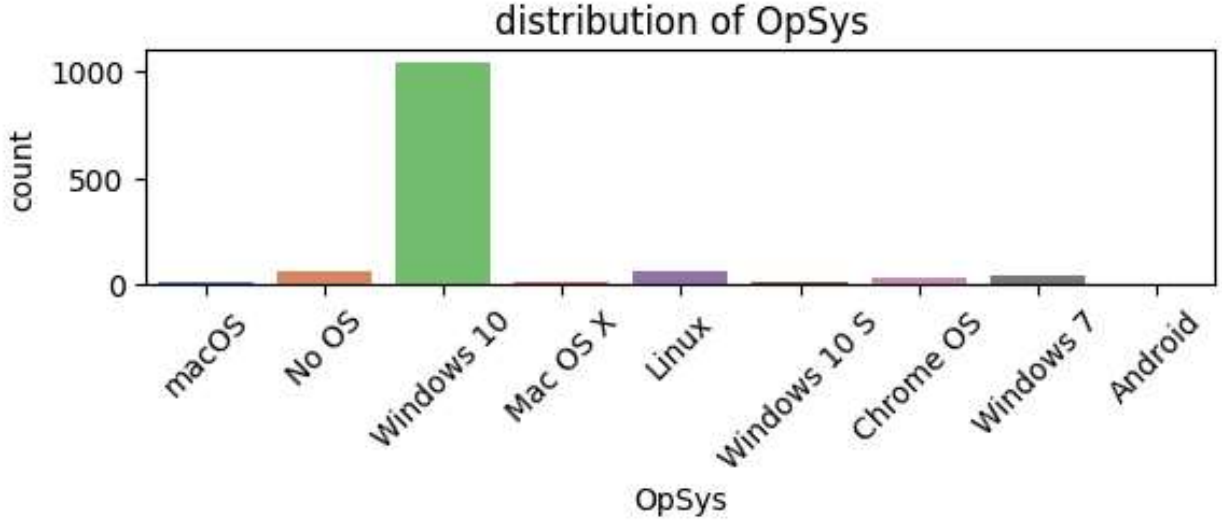
Categorical feature discription



Categorical feature discription



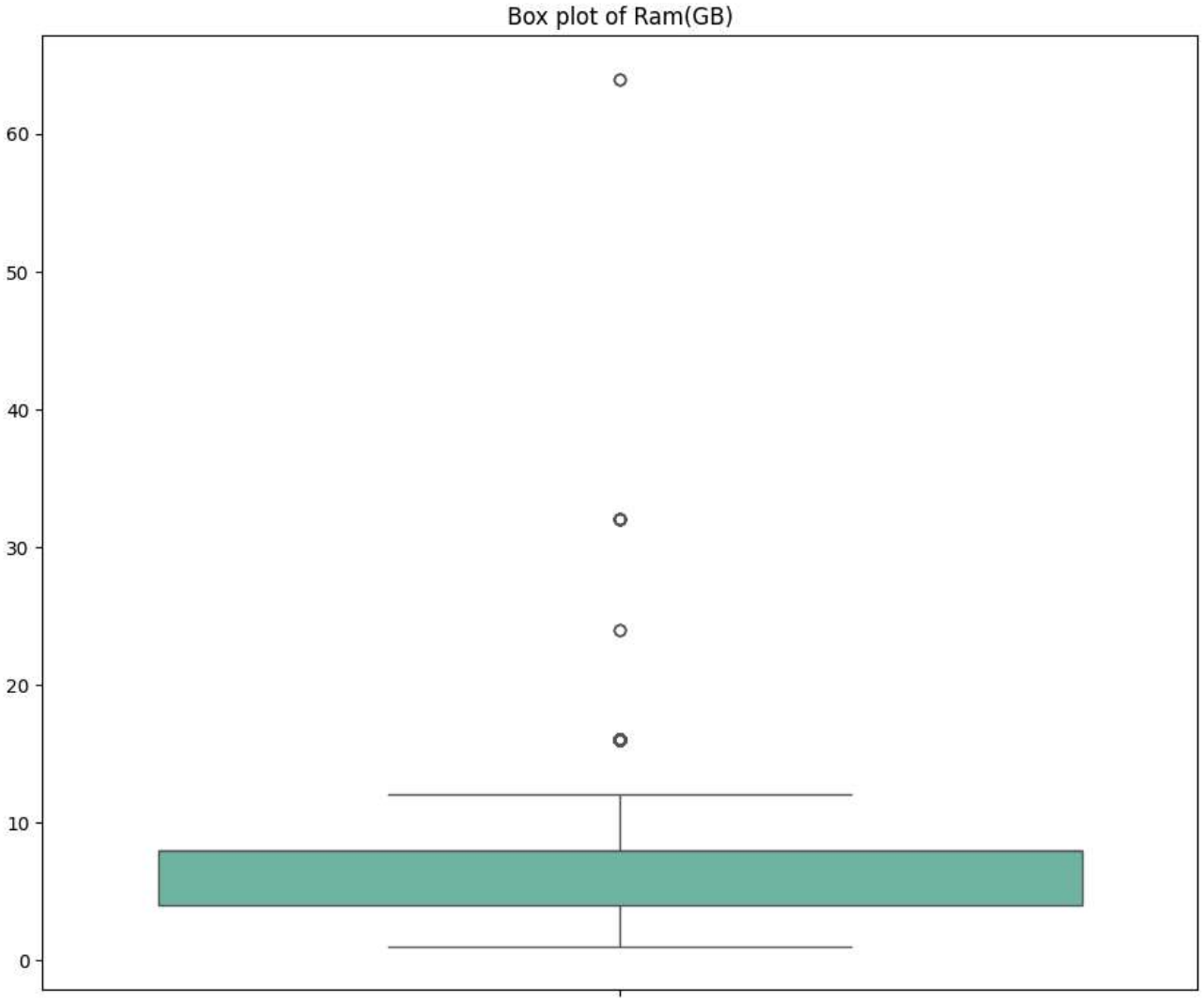
Categorical feature discription

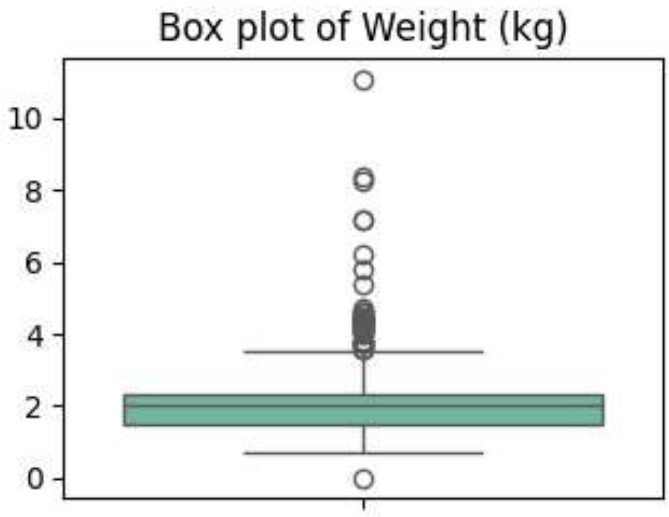

distribution of OpSys
count
1000
500
0
macOS
No OS
Windows 10
Mac OS X
Linux
Windows 10 S
Chrome OS
Windows 7
Android
OpSys

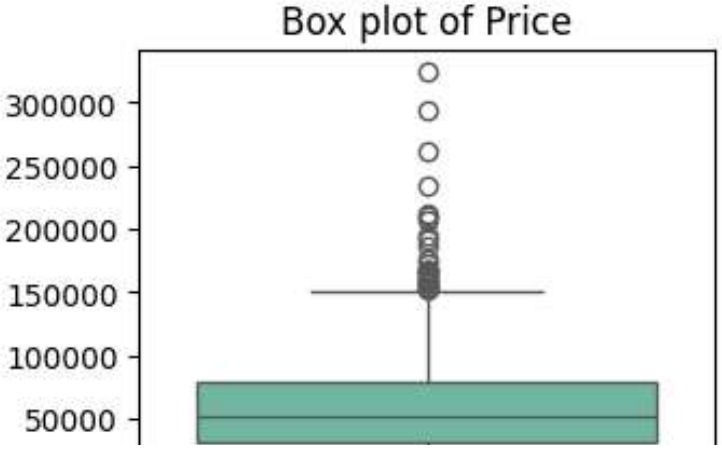
Box-plot

To check for outliers

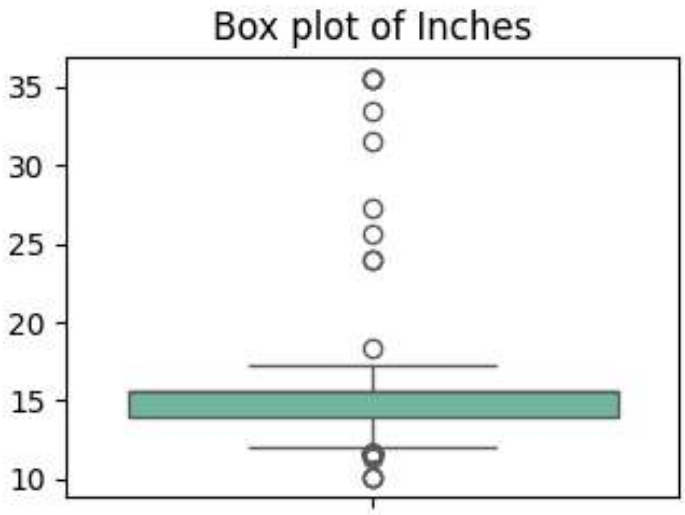
```
#Box-plot for the numerical features to check for the outliers
num_feature=['Ram(GB)','Weight (kg)','Price','Inches']
plt.figure(figsize=(18,15))
for i,col in enumerate(num_feature,1):
  plt.subplot(2,2,i)
  sns.boxplot(data=df1,y=col,palette='Set2')
  plt.ylabel("")
  plt.title(f"Box plot of {col}")
  plt.tight_layout()
  plt.show()
```


Box plot of Ram(GB)
60
50
40
30
20
10
0


Box plot of Weight (kg)
10
8
6
4
2
0


Box plot of Price
300000
250000
200000
150000
100000
50000

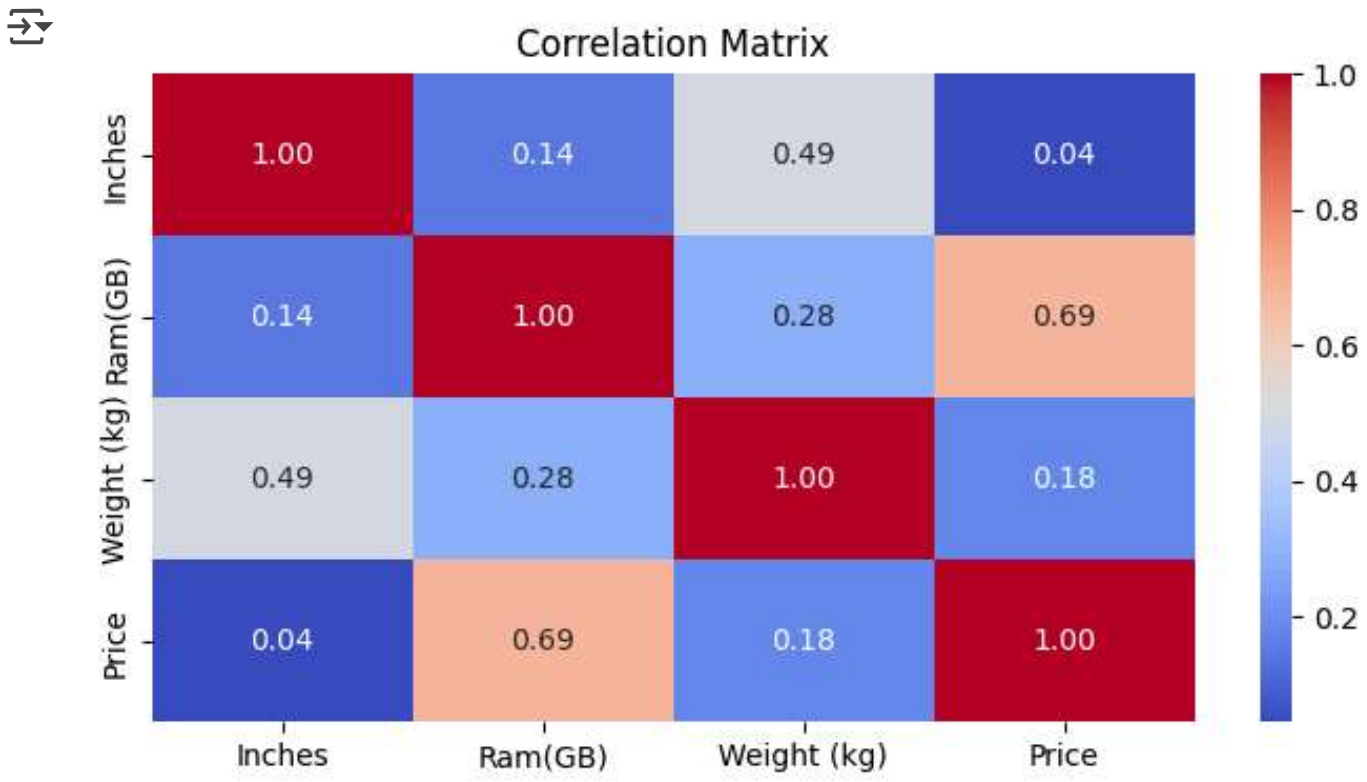

0


Box plot of Inches
35
30
25
20
15
10

Heatmap / Correlation Matrix

- To see the coreelation between the different columns
- |r|>0.85 -> strong correlation
- 0.4>|r|<0.85 -> moderate correlation
- |r|<0.4 -> weak correlation

```
# Heat Map
plt.figure(figsize=(8,4))
sns.heatmap(df1.select_dtypes(include=['number']).corr(),annot=True,cmap="coolwarm"
plt.title("Correlation Matrix")
plt.show()
```



From above matrix we get to know that **Ram and price** of laptop have **good moderate** correlation and **weight and inches** have **moderate correlation**

HANDLING NULL VALUE

```
df1.isnull().sum()
# we have one null value in weight coloum
```

	0
Company	0
TypeName	0
Inches	1
ScreenResolution	0
Cpu	0
Ram(GB)	0
Memory	0
Gpu	0
OpSys	0
Weight (kg)	1
Price	0

dtype: int64

print whole row with null value

```
df1[df1.isna().any(axis=1)]
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
201	Dell	Ultrabook	13.3	Full HD 1920x1080	Intel Core i7 8550U 1.8GHz	8.0	256 SSD	G

Handling null value in weight

> Add blockquote

We minimize the scope to fill the null weight to mean of laptops of **Dell Ultrabook with 13.3 inches and with ram 8gb**

```
#Dataframe with having laptops of  Dell Ultrabook with 13.3 inches and with ram 8gb
df1_weight=df1[(df1.Company=='Dell') & (df1.TypeName=='Ultrabook')&(df1.Inches==13.
df1_weight
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
19	Dell	Ultrabook	13.3	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i5 8250U 1.6GHz	8.0	128 SSD	Gra
86	Dell	Ultrabook	13.3	IPS Panel Full HD 1920x1080	Intel Core i7 8550U 1.8GHz	8.0	256 SSD	Gra
108	Dell	Ultrabook	13.3	Quad HD+ / Touchscreen 3200x1800	Intel Core i7 7560U 2.4GHz	8.0	256 SSD	In Gra
111	Dell	Ultrabook	13.3	IPS Panel Full HD 1920x1080	Intel Core i5 8250U 1.6GHz	8.0	256 SSD	Gra
175	Dell	Ultrabook	13.3	Full HD 1920x1080	Intel Core i5 8250U 1.6GHz	8.0	256 SSD	Gra
200	Dell	Ultrabook	13.3	IPS Panel 4K Ultra HD / Touchscreen 3840x2160	Intel Core i7 8550U 1.8GHz	8.0	256 SSD	Gra
201	Dell	Ultrabook	13.3	Full HD 1920x1080	Intel Core i7 8550U 1.8GHz	8.0	256 SSD	Gra
207	Dell	Ultrabook	13.3	IPS Panel Full HD 1920x1080	Intel Core i7 8550U 1.8GHz	8.0	256 SSD	Ra
247	Dell	Ultrabook	13.3	Full HD 1920x1080	Intel Core i5 8250U 1.6GHz	8.0	256 SSD	Gra
334	Dell	Ultrabook	13.3	Full HD 1920x1080	Intel Core i7 8550U 1.8GHz	8.0	256 SSD	Gra
462	Dell	Ultrabook	13.3	Full HD / Touchscreen 1920x1080	Intel Core i5 8250U 1.6GHz	8.0	256 SSD	Gra

498	Dell	Ultrabook	13.3	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i5 7200U 2.5GHz	8.0	128 SSD	Int Gra
528	Dell	Ultrabook	13.3	Quad HD+ / Touchscreen 3200x1800	Intel Core i7 8550U 1.8GHz	8.0	256 SSD	Gra
				IPS Panel 4K Ultra	Intel			

Next steps: Generate code with df1_weight | View recommended plots | New interactive shee

```
# We will use the mean of weight of df1_weight dataframe to replace null value
df1_weight_mean=round(df1_weight['Weight (kg)'].mean(),2)
df1_weight_mean
```

```
np.float64(1.26)
```

```
df1['Weight (kg)']=df1['Weight (kg)'].fillna(df1_weight_mean)
```

Handling null value in inches

```
#Dataframe with having laptops of  Dell Ultrabook with 13.3 inches and with ram 8gb
df1_inches=df1[(df1.Company=='Dell') & (df1.TypeName=='Workstation')&(df1['Ram(GB)
df1_inches
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
376	Dell	Workstation	15.6	Full HD 1920x1080	Intel Core i5 6440HQ 2.6GHz	8.0	500 HDD	W
377	Dell	Workstation	15.6	IPS Panel Full HD 1920x1080	Intel Core i7 6820HQ 2.7GHz	8.0	256 SSD	
387	Dell	Workstation	15.6	4K Ultra HD / Touchscreen 3840x2160	Intel Core i7 7700HQ 2.8GHz	8.0	256 SSD	
436	Dell	Workstation	15.6	Full HD 1920x1080	Intel Xeon E3-1505M V6 3GHz	8.0	64 Flash Storage + 1TB HDD	

Next steps: Generate code with df1_inches | View recommended plots | New interactive shee

```
# We will use the mean of inches of df1_inches dataframe to replace null value
df1_inches_mean=df1_inches['Inches'].mean()
df1_inches_mean
```

```
np.float64(15.6)
```

```
df1['Inches']=df1['Inches'].fillna(df1_inches_mean)
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1273 entries, 0 to 1272
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Company           1273 non-null   object 
 1   TypeName          1273 non-null   object 
 2   Inches            1273 non-null   float64
 3   ScreenResolution  1273 non-null   object 
 4   Cpu               1273 non-null   object 
 5   Ram(GB)           1273 non-null   float64
 6   Memory            1273 non-null   object 
 7   Gpu               1273 non-null   object 
 8   OpSys             1273 non-null   object 
 9   Weight (kg)       1273 non-null   float64
```

```
 10  Price             1273 non-null   float64
dtypes: float64(4), object(7)
memory usage: 109.5+ KB
```

Handling Outliers

```
df1.describe()
```

	Inches	Ram(GB)	Weight (kg)	Price
count	1273.000000	1273.000000	1273.000000	1273.000000
mean	15.131186	8.462687	2.076976	59955.804713
std	1.953712	5.564408	0.807815	37332.250492
min	10.100000	1.000000	0.000200	9270.700000
25%	14.000000	4.000000	1.500000	31914.700000
50%	15.600000	8.000000	2.040000	52161.100000
75%	15.600000	8.000000	2.320000	79333.400000
max	35.600000	64.000000	11.100000	324954.700000

```
# checking for max ram to check if outliers are genuine or not
df1_35inc=df1[(df1.Inches>25.6)]
df1_35inc
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
397	Lenovo	Notebook	35.6	Full HD 1920x1080	Intel Core i3 6006U 2GHz	4.0	500 HDD	I G
412	Asus	Notebook	35.6	Full HD 1920x1080	Intel Core i3 7100U 2.4GHz	8.0	1TB HDD	G
413	Dell	Gaming	27.3	IPS Panel 2560x1440	Intel Core i7 7820HK 2.9GHz	16.0	256 SSD + 1TB HDD	G
				Touchscreen	Intel Core i7		256	I

Next steps: Generate code with df1_35inc | View recommended plots | New interactive sheet

```
# checking for max price to check if outliers are genuine or not

df1_max_price=df1[(df1.Price==324954.7)]
df1_max_price
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
189	Razer	Gaming	17.3	4K Ultra HD / Touchscreen	Intel Core i7	32.0	1TB	N GeF

```
# checking for max ram to check if outliers are genuine or not
max_ram=df1[df1['Ram(GB)']==64]
max_ram
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
68	Dell	Ultrabook	13.3	IPS Panel Full HD 1920x1080	Intel Core i7 8550U 1.8GHz	64.0	256 SSD	R
702	Lenovo	Ultrabook	14.0	IPS Panel Quad HD+ 2560x1440	Intel Core i7 6500U	64.0	512 SSD	In Gr

Next steps: Generate code with max_ram | View recommended plots | New interactive sheet

```
# checking for mimum weight in laptops
min_weight=df1[df1['Weight (kg)']<1]
min_weight
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory
14	Apple	Ultrabook	12.0	IPS Panel Retina Display 2304x1440	Intel Core M m3 1.2GHz	8.0	256 SSD
78	Apple	Ultrabook	12.0	IPS Panel Retina Display 2304x1440	Intel Core i5 1.3GHz	8.0	512 SSD
141	HP	Ultrabook	12.5	IPS Panel 4K Ultra HD / Touchscreen 3840x2160	Intel Core M 6Y75 1.2GHz	8.0	512 SSD
339	Dell	Ultrabook	15.6	Full HD 1920x1080	Intel Core i5 8250U 1.6GHz	8.0	1TB HDD
477	Asus	Ultrabook	12.5	Full HD 1920x1080	Intel Core i7 7500U 2.7GHz	16.0	512 SSD
727	Samsung	Ultrabook	13.3	Full HD 1920x1080	Intel Core i7 7500U 2.7GHz	16.0	256 SSD
772	Apple	Ultrabook	12.0	IPS Panel Retina Display 2304x1440	Intel Core M 1.2GHz	8.0	512 Flash Storage
790	Samsung	Ultrabook	13.3	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8.0	256 SSD
887	LG	Ultrabook	14.0	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i7 7500U 2.7GHz	8.0	512 SSD
964	HP	Netbook	12.5	Full HD 1920x1080	Intel Core M 6Y75 1.2GHz	8.0	512 SSD
1041	Apple	Ultrabook	12.0	IPS Panel Retina Display 2304x1440	Intel Core M 1.1GHz	8.0	256 Flash Storage

After looking at this data we get one minimum value i.e. 0.0002kg weight of laptop, the weight of laptop cant we such low, lets replace this value with most likely one

```
df1_weight2=df1[(df1.Company=='Dell') & (df1.TypeName=='Ultrabook')&(df1.Inches==15
df1_weight2
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
27	Dell	Ultrabook	15.6	Full HD 1920x1080	Intel Core i7 8650U 1.9GHz	8.0	256 SSD + 256 SSD	Grap
77	Dell	Ultrabook	15.6	IPS Panel Full HD 1920x1080	Intel Core i5 8250U	8.0	256 SSD	Grap

Next steps: Generate code with df1_weight2 | View recommended plots | New interactive she

```
# Handle the minimum value with approprite value match with similer kind of specific

df1['Weight (kg)']=df1['Weight (kg)'].replace(0.0002,1.8800)


#Check for weight of laptop above 5 kg
max_weight=df1[df1['Weight (kg)']<5]
max_weight
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	Memory	
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8.0	128 SSD	C
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8.0	128 Flash Storage	C
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8.0	256 SSD	C
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16.0	512 SSD	
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8.0	256 SSD	C
...	...	...	...	...	...	...	...	
1268	Lenovo	2 in 1 Convertible	14.0	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i7 6500U 2.5GHz	4.0	128 SSD	C
1269	Lenovo	2 in 1 Convertible	13.3	IPS Panel Quad HD+ / Touchscreen 3200x1800	Intel Core i7 6500U 2.5GHz	16.0	512 SSD	C

Next steps: Generate code with `max_weight` View recommended plots New interactive shee

from above checking of rows and columns we get to know that outliers data in **columns inches,ram,and price** is legitimates but data in weight column is need correction

```
#creat the copy of dataframe to protect data changes
dfo2=df1.copy()
```

```
# convert datatype of price into object
# to run price as target variable in model we need to normalise it or scale it
# So scale it we need object data to make it categorical data
dfo2['Price'] = dfo2['Price'].astype(object)
dfo2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1273 entries, 0 to 1272
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Company           1273 non-null   object 
 1   TypeName          1273 non-null   object 
 2   Inches            1273 non-null   float64
 3   ScreenResolution  1273 non-null   object 
 4   Cpu               1273 non-null   object 
 5   Ram(GB)           1273 non-null   float64
 6   Memory            1273 non-null   object 
 7   Gpu               1273 non-null   object 
 8   OpSys             1273 non-null   object 
 9   Weight (kg)       1273 non-null   float64
 10  Price             1273 non-null   object 
dtypes: float64(3), object(8)
memory usage: 109.5+ KB
```

Winsorization (Capping Outliers)

- we will use winsorization method becasuse we want to keep data
- extreme great will replace with max value of the box
- extreme lower will replace with min value of the box

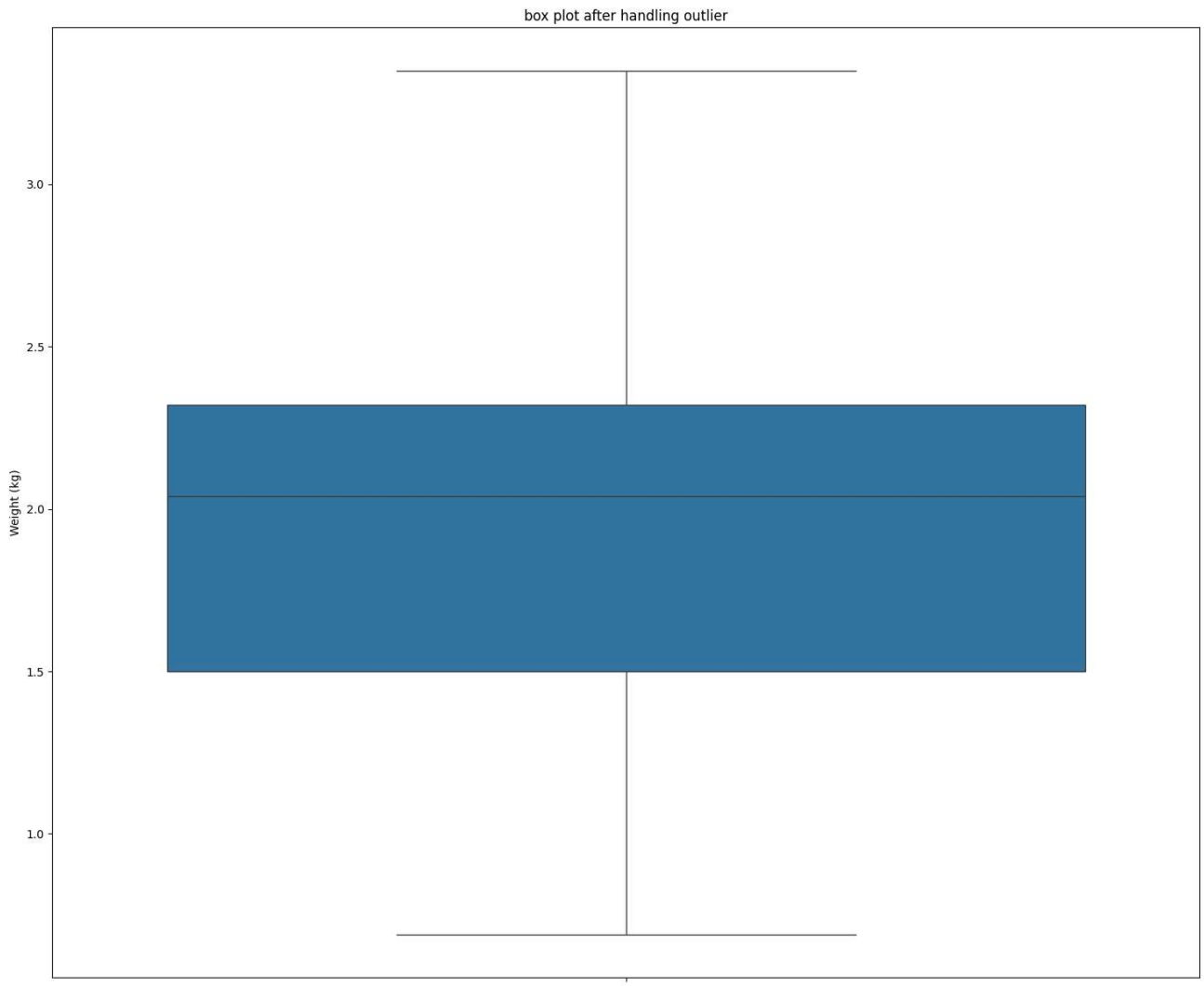
```
# Applying winsarization
from scipy.stats.mstats import winsorize
# Apply Winsorization to weight columns (capping extreme max values at 95%)
dfo2['Weight (kg)'] = winsorize(dfo2['Weight (kg)'], limits=[0, 0.05])
# lower limit there is no outlier

print("Original Data Shape:", dfo2.shape)
print("After Winsorization (Capping):", dfo2.shape)
```

```
Original Data Shape: (1273, 11)
After Winsorization (Capping): (1273, 11)
```

```
# We can clearly see the we change the values of outliers in weight

dfo2.describe()
```

	Inches	Ram(GB)	Weight (kg)
count	1273.000000	1273.000000	1273.000000
mean	15.131186	8.462687	2.020911
std	1.953712	5.564408	0.587612
min	10.100000	1.000000	0.690000
25%	14.000000	4.000000	1.500000
50%	15.600000	8.000000	2.040000
75%	15.600000	8.000000	2.320000
max	35.600000	64.000000	3.350000

```
# Visualize the boxplot after handling outliers
plt.figure(figsize=(18,15))
sns.boxplot(data=dfo2,y='Weight (kg)')
plt.title("box plot after handling outlier")
plt.show()
```

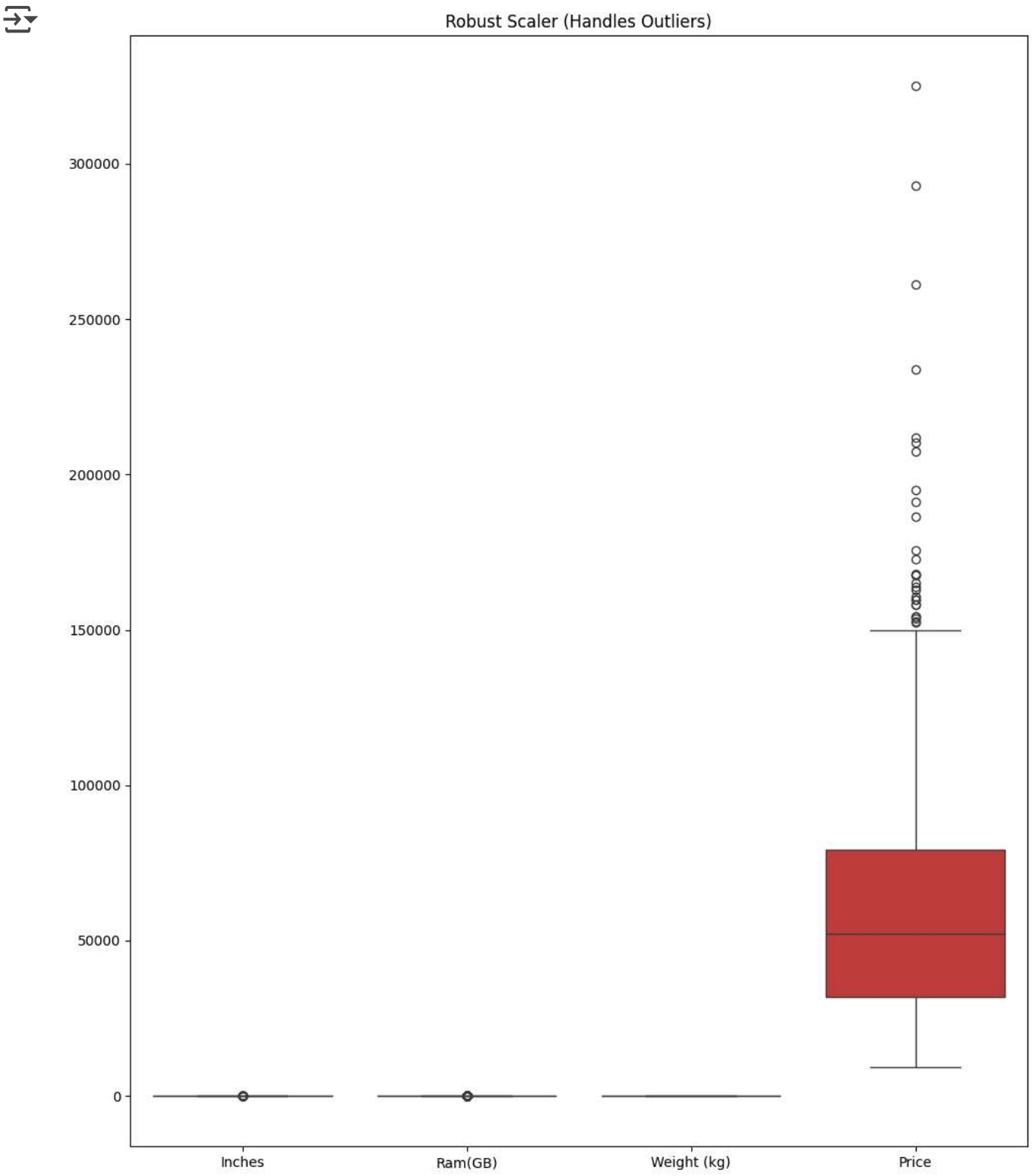

box plot after handling outlier
Weight (kg)
3.0
2.5
2.0
1.5
1.0

SCALING

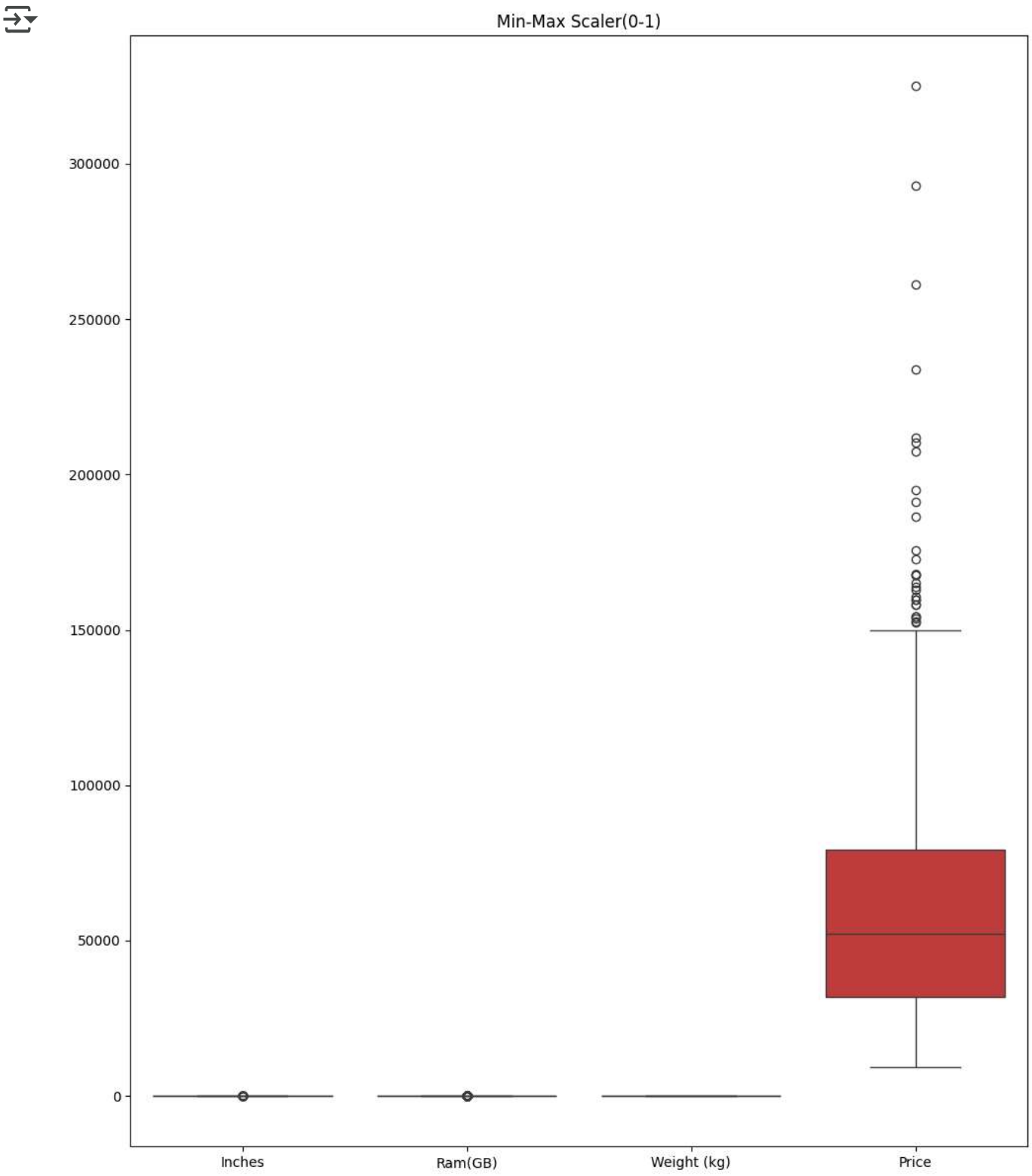
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler



df_robust = dfo2.copy()
df_minmax = dfo2.copy()
```

RobustScaler

```
# As we have lots of outliers we will use Robust scaler

# Selecting only numerical columns for scaling
num_cols = df_robust.select_dtypes(include=['number']).columns

robust_scaler = RobustScaler()
df_robust[num_cols] = robust_scaler.fit_transform(df_robust[num_cols])


# Min-Max Scaling (0 to 1 Range)
minmax_scaler = MinMaxScaler()
df_minmax[num_cols] = minmax_scaler.fit_transform(df_minmax[num_cols])


# Create a single figure and axes for the Robust Scaler boxplot
fig, axes = plt.subplots(figsize=(10, 12))

# Plot the boxplot on the single axes object
sns.boxplot(data=df_robust, ax=axes) # Use 'axes' directly, not axes[2]
axes.set_title("Robust Scaler (Handles Outliers)")

plt.tight_layout()
plt.show()
```


Robust Scaler (Handles Outliers)
300000
250000
200000
150000
100000
50000
0
Inches
Ram(GB)
Weight (kg)
Price

```
# Create a single figure and axes for the Robust Scaler boxplot
fig, axes = plt.subplots(figsize=(10, 12))

# Plot the boxplot on the single axes object
sns.boxplot(data=df_minmax, ax=axes) # Use 'axes' directly, not axes[2]
axes.set_title("Min-Max Scaler(0-1)")

plt.tight_layout()
plt.show()
```


Min-Max Scaler(0-1)
300000
250000
200000
150000
100000
50000
0
Inches
Ram(GB)
Weight (kg)
Price

ENCODING

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

Robust method encoding

```
# Selecting categorical columns for Robust scaler encoding
cat_colsR = df_robust.select_dtypes(include=['object']).columns
```

Label Encoding

```
# Label Encoding
df_label_encodedR = df_robust.copy()
label_encoder = LabelEncoder()
for col in cat_colsR:
    df_label_encodedR[col] = label_encoder.fit_transform(df_label_encodedR[col])


# Display results
print("Label Encoded Data:")
print(df_label_encodedR.head())
print("\nUnique Values in Each Column:")
print(df_label_encodedR.shape)
```

```
Label Encoded Data:
   Company  TypeName  Inches  ScreenResolution  Cpu  Ram(GB)  Memory  Gpu  \
0        1         4 -1.4375                23   65      0.0       4   56
1        1         4 -1.4375                 1   63      0.0       2   50
```

```
2         7         3  0.0000                 8   74      0.0      16   52
3         1         4 -0.1250                25   85      2.0      29    8
4         1         4 -1.4375                23   67      0.0      16   57

   OpSys  Weight (kg)  Price
0      8    -0.817073    517
1      8    -0.853659    338
2      4    -0.219512    164
3      8    -0.256098    732
4      8    -0.817073    627

Unique Values in Each Column:
(1273, 11)
```

Unlike some other scalers (like MinMaxScaler which scales values to a fixed range like 0 to 1), RobustScaler scales features using statistics that are robust to outliers. Specifically, it centers the data by removing the median and scales the data by dividing by the interquartile range (IQR).

The formula it uses is:

Scaled Value = (Original Value - Median) / IQR

Here's why this can result in negative values:

If an original value is less than the median of that column, the term (Original Value - Median) will be negative. Since the IQR is always a positive value, dividing a negative number by a positive number results in a negative number. So, the negative values in your output indicate that those particular data points had original values that were below the median value of their respective columns before scaling.

This is a normal and expected behavior when using RobustScaler, as its primary goal is to scale data relative to the median and IQR, rather than to a strictly positive range.

one hot encoded

```
df_one_hot_encodedR = pd.get_dummies(df_robust, columns=cat_colsR, drop_first=True)


print("\nOne-Hot Encoded Data:")
print(df_one_hot_encodedR.head())
print(df_one_hot_encodedR.shape)
```

```
One-Hot Encoded Data:
   Inches  Ram(GB)  Weight (kg)  Company_Apple  Company_Asus  Company_Chuwi  \
0 -1.4375      0.0    -0.817073           True         False          False
1 -1.4375      0.0    -0.853659           True         False          False
```

```
2  0.0000      0.0    -0.219512          False         False          False
3 -0.1250      2.0    -0.256098           True         False          False
4 -1.4375      0.0    -0.817073           True         False          False

   Company_Dell  Company_Fujitsu  Company_Google  Company_HP  ...  \
0         False            False           False       False  ...
1         False            False           False       False  ...
2         False            False           False        True  ...
3         False            False           False       False  ...
4         False            False           False       False  ...

   Price_186426.7  Price_191211.3  Price_194972.8  Price_207259.2  \
0           False           False           False           False
1           False           False           False           False
2           False           False           False           False
3           False           False           False           False
4           False           False           False           False

   Price_210424.0  Price_211788.0  Price_233845.9  Price_261018.7  \
0           False           False           False           False
1           False           False           False           False
2           False           False           False           False
3           False           False           False           False
4           False           False           False           False

   Price_292986.7  Price_324954.7
0           False           False
1           False           False
2           False           False
3           False           False
4           False           False

[5 rows x 1110 columns]
(1273, 1110)
```

Min max scaler encoding

```
# Selecting categorical columns for Robust scaler encoding
cat_colsM = df_minmax.select_dtypes(include=['object']).columns
```

label encoding for min max scale

```
# Label Encoding
df_label_encodedM = df_minmax.copy()
label_encoder = LabelEncoder()
for col in cat_colsM:
    df_label_encodedM[col] = label_encoder.fit_transform(df_label_encodedM[col])
```

```
# Display results
print("Label Encoded Data:")
print(df_label_encodedM.head())
print("\nUnique Values in Each Column:")
print(df_label_encodedM.shape)
```

```
Label Encoded Data:
   Company  TypeName    Inches  ScreenResolution  Cpu   Ram(GB)  Memory  Gpu  \
0        1         4  0.125490                23   65  0.111111       4   56
1        1         4  0.125490                 1   63  0.111111       2   50
2        7         3  0.215686                 8   74  0.111111      16   52
3        1         4  0.207843                25   85  0.238095      29    8
4        1         4  0.125490                23   67  0.111111      16   57

   OpSys  Weight (kg)  Price
0      8     0.255639    517
1      8     0.244361    338
2      4     0.439850    164
3      8     0.428571    732
4      8     0.255639    627

Unique Values in Each Column:
(1273, 11)
```

one hot encoadin for min max scaler

```
df_one_hot_encodedM = pd.get_dummies(df_minmax, columns=cat_colsM, drop_first=True)


print("\nOne-Hot Encoded Data:")
print(df_one_hot_encodedM.head())
print(df_one_hot_encodedM.shape)
```

```
One-Hot Encoded Data:
     Inches   Ram(GB)  Weight (kg)  Company_Apple  Company_Asus  \
0  0.125490  0.111111     0.255639           True         False
1  0.125490  0.111111     0.244361           True         False
2  0.215686  0.111111     0.439850          False         False
3  0.207843  0.238095     0.428571           True         False
4  0.125490  0.111111     0.255639           True         False

   Company_Chuwi  Company_Dell  Company_Fujitsu  Company_Google  Company_HP  \
0          False         False            False           False       False
1          False         False            False           False       False
2          False         False            False           False        True
3          False         False            False           False       False
4          False         False            False           False       False

   ...  Price_186426.7  Price_191211.3  Price_194972.8  Price_207259.2  \
0  ...           False           False           False           False
1  ...           False           False           False           False
```

```
2  ...           False           False           False           False
3  ...           False           False           False           False
4  ...           False           False           False           False

   Price_210424.0  Price_211788.0  Price_233845.9  Price_261018.7  \
0           False           False           False           False
1           False           False           False           False
2           False           False           False           False
3           False           False           False           False
4           False           False           False           False

   Price_292986.7  Price_324954.7
0           False           False
1           False           False
2           False           False
3           False           False
4           False           False

[5 rows x 1110 columns]
(1273, 1110)
```

Data Splitting (Train-Test Split)

```
from sklearn.model_selection import train_test_split


# Define features (X) and target variable (y)
# we used label encoding done on the robust data to test andret train
X = df_label_encodedR.drop(columns=['Price'])  # Features
y = df_label_encodedR['Price']  # Target variable


# code to train and test split 20-80

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Display the shapes of the resulting sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (1018, 10)
Shape of X_test: (255, 10)
Shape of y_train: (1018,)
Shape of y_test: (255,)
```

Model Building

As this is supervised learning model with target variable as continous we are using regressision mode.

AS our data is complex and high-dimension and target veriable is non-linear,we use Random Forest Algorethim as it combine prediction from multiple decision tree to produce more accurate and stable prediction

```
from sklearn.ensemble import RandomForestRegressor
# Initialize the Random Forest Regressor model
# You can adjust parameters like n_estimators (number of trees)
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_model.predict(X_test)
```

Model evaluation

```
from sklearn.metrics import mean_squared_error, r2_score

# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Random Forest Regressor Performance:")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Random Forest Regressor Performance:
Root Mean Squared Error (RMSE): 75.68
R-squared (R2): 0.88
```

```
# Visualize predicted vs actual prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2) #
```