Laptop Dataset Analysis

Dataset description:

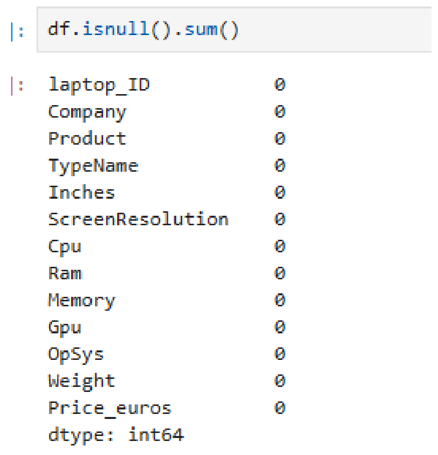
laptop_ID: Unique identifier for each laptop.
Company: The manufacturer or brand of the laptop.
Product: The specific model or product name of the laptop.
TypeName: The type or category of the laptop (e.g., Ultrabook, Gaming, Workstation).
Inches: The size of the laptop screen in inches.
ScreenResolution: The resolution of the laptop screen.
Cpu: The central processing unit (CPU) of the laptop.
RAM (in GB): The amount of random-access memory (RAM) in gigabytes.
Memory: The storage capacity of the laptop (e.g., SSD, HDD, SSD+HDD).
Gpu: The graphics processing unit (GPU) of the laptop.
OpSys: The operating system installed on the laptop.
Weight (in kg): The weight of the laptop in kilograms.
Price_euros: The price of the laptop in euros.

The dataset comprises 1303 entries and contains 13 columns.

#Dataset reading and cleaning

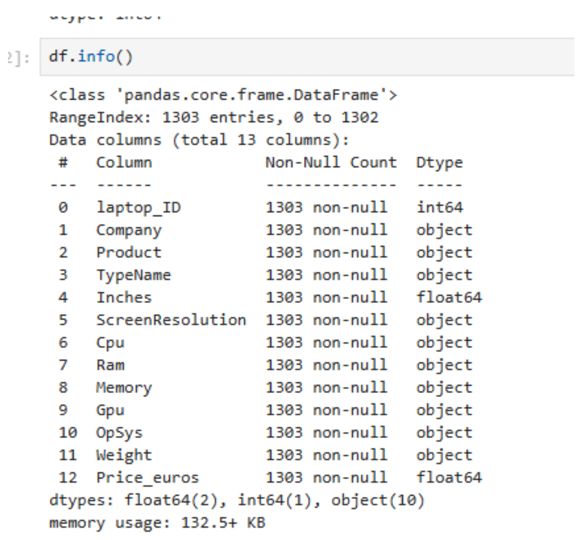
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
file_path = r'C:\Users\ishan\Downloads\laptop_price.csv'
df = pd.read_csv(file_path, encoding='ISO-8859-1')
```

#Checking null values



```
]: df.isnull().sum()

]: laptop_ID           0
   Company             0
   Product             0
   TypeName            0
   Inches              0
   ScreenResolution    0
   Cpu                 0
   Ram                 0
   Memory              0
   Gpu                 0
   OpSys               0
   Weight              0
   Price_euros         0
   dtype: int64
```

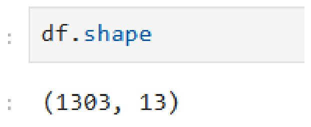
#Dataset Info



```
df.info()
```

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

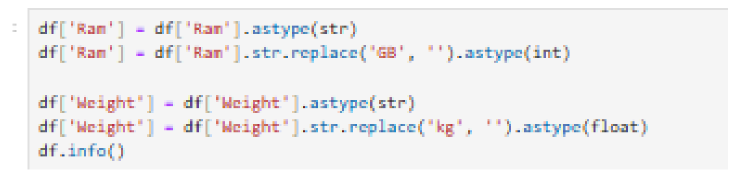
#Dataset shape (before removing outliers)



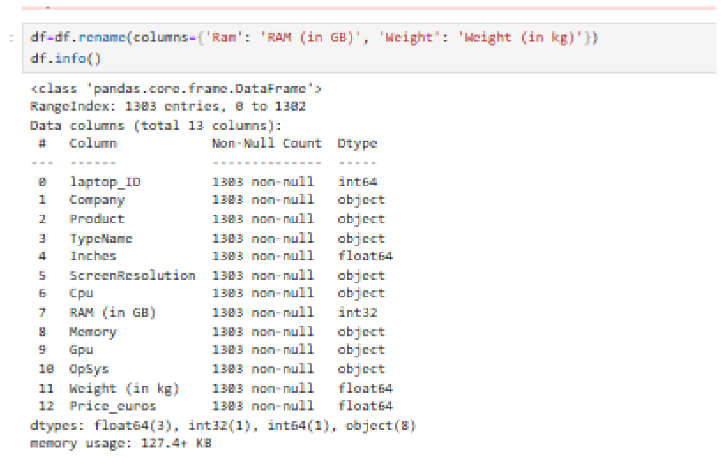
```
df.shape
```

```
(1303, 13)
```

#Changing column data types and name



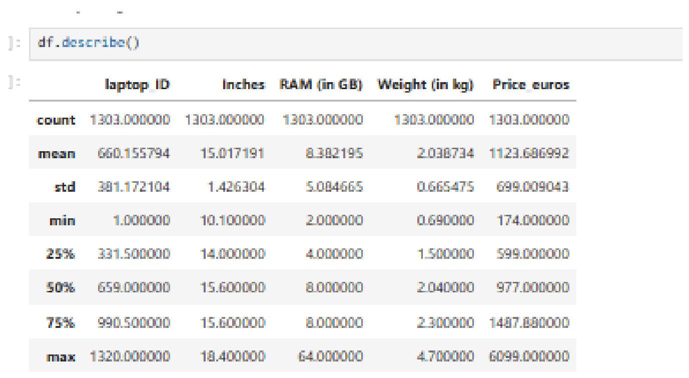
```
df['Ram'] = df['Ram'].astype(str)
df['Ram'] = df['Ram'].str.replace('GB', '').astype(int)

df['Weight'] = df['Weight'].astype(str)
df['Weight'] = df['Weight'].str.replace('kg', '').astype(float)
df.info()
```



```
df=df.rename(columns={'Ram': 'RAM (in GB)', 'Weight': 'Weight (in kg)'})
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   laptop_ID         1303 non-null   int64
 1   Company           1303 non-null   object
 2   Product           1303 non-null   object
 3   TypeName          1303 non-null   object
 4   Inches            1303 non-null   float64
 5   ScreenResolution  1303 non-null   object
 6   Cpu               1303 non-null   object
 7   RAM (in GB)       1303 non-null   int32
 8   Memory            1303 non-null   object
 9   Gpu               1303 non-null   object
 10  OpSys             1303 non-null   object
 11  Weight (in kg)    1303 non-null   float64
 12  Price_euros       1303 non-null   float64
dtypes: float64(3), int32(1), int64(1), object(8)
memory usage: 127.4+ KB
```

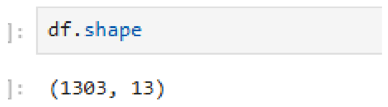
#Dataset description



```
df.describe()
```

	laptop_ID	Inches	RAM (in GB)	Weight (in kg)	Price_euros
count	1303.000000	1303.000000	1303.000000	1303.000000	1303.000000
mean	660.155794	15.017191	8.382195	2.038734	1123.686992
std	381.172104	1.426304	5.084665	0.665475	699.009043
min	1.000000	10.100000	2.000000	0.690000	174.000000
25%	331.500000	14.000000	4.000000	1.500000	599.000000
50%	659.000000	15.600000	8.000000	2.040000	977.000000
75%	990.500000	15.600000	8.000000	2.300000	1487.880000
max	1320.000000	18.400000	64.000000	4.700000	6099.000000

#Dataset shape



```
df.shape
```

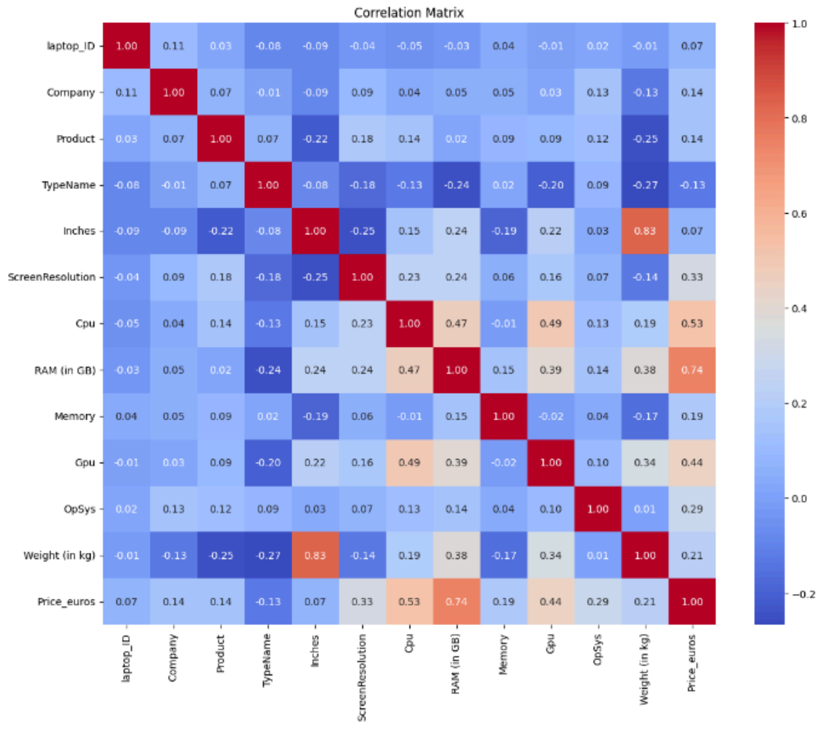
```
(1303, 13)
```

#Label encoder (Converting categorical columns into numeric values)

```
label_encoders = {}
for column in ['Company', 'Product', 'TypeName', 'ScreenResolution', 'Cpu', 'Memory', 'Gpu',
'OpSys']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

#Correlation matrix (before removing outlier)

```
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.tight_layout()
plt.show()
```

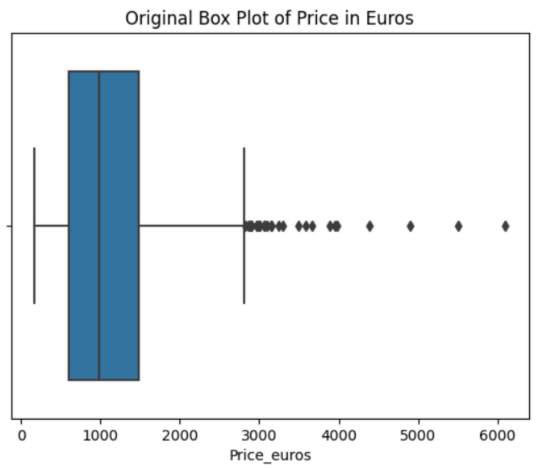


	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	RAM (in GB)	Memory	Gpu	OpSys	Weight (in kg)	Price_euros
laptop_ID	1.00	0.11	0.03	-0.08	-0.09	-0.04	-0.05	-0.03	0.04	-0.01	0.02	-0.01	0.07
Company	0.11	1.00	0.07	-0.01	-0.09	0.09	0.04	0.05	0.05	0.03	0.13	-0.13	0.14
Product	0.03	0.07	1.00	0.07	-0.22	0.18	0.14	0.02	0.09	0.09	0.12	-0.25	0.14
TypeName	-0.08	-0.01	0.07	1.00	-0.08	-0.18	-0.13	-0.24	0.02	-0.20	0.09	-0.27	-0.13
Inches	-0.09	-0.09	-0.22	-0.08	1.00	-0.25	0.15	0.24	-0.19	0.22	0.03	0.83	0.07
ScreenResolution	-0.04	0.09	0.18	-0.18	-0.25	1.00	0.23	0.24	0.06	0.16	0.07	-0.14	0.33
Cpu	-0.05	0.04	0.14	-0.13	0.15	0.23	1.00	0.47	-0.01	0.49	0.13	0.19	0.53
RAM (in GB)	-0.03	0.05	0.02	-0.24	0.24	0.24	0.47	1.00	0.15	0.39	0.14	0.38	0.74
Memory	0.04	0.05	0.09	0.02	-0.19	0.06	-0.01	0.15	1.00	-0.02	0.04	-0.17	0.19
Gpu	-0.01	0.03	0.09	-0.20	0.22	0.16	0.49	0.39	-0.02	1.00	0.10	0.34	0.44
OpSys	0.02	0.13	0.12	0.09	0.03	0.07	0.13	0.14	0.04	0.10	1.00	0.01	0.29
Weight (in kg)	-0.01	-0.13	-0.25	-0.27	0.83	-0.14	0.19	0.38	-0.17	0.34	0.01	1.00	0.21
Price_euros	0.07	0.14	0.14	-0.13	0.07	0.33	0.53	0.74	0.19	0.44	0.29	0.21	1.00

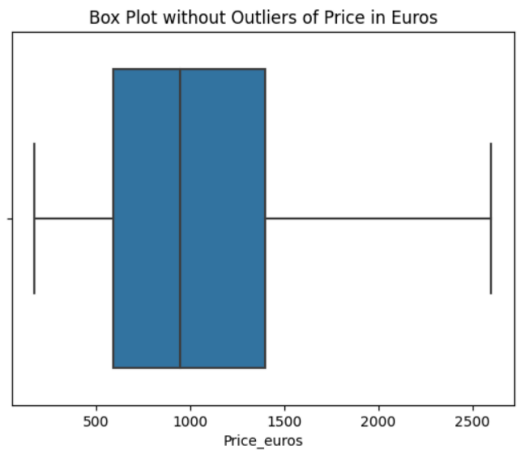
#Outlier detection and removal

Price in Euros

```
sns.boxplot(x='Price_euros', data=df)
plt.title(f'Original Box Plot of Price in Euros')
plt.show()
```

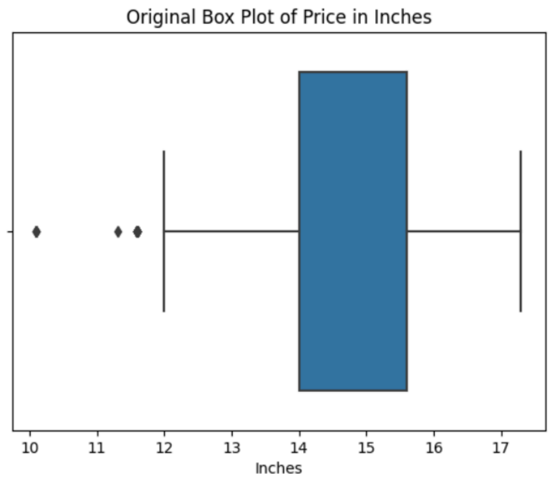


```
df1=df[df['Price_euros']<2620]
sns.boxplot(x='Price_euros', data=df1)
plt.title(f'Box Plot without Outliers of Price in Euros')
plt.show()
```

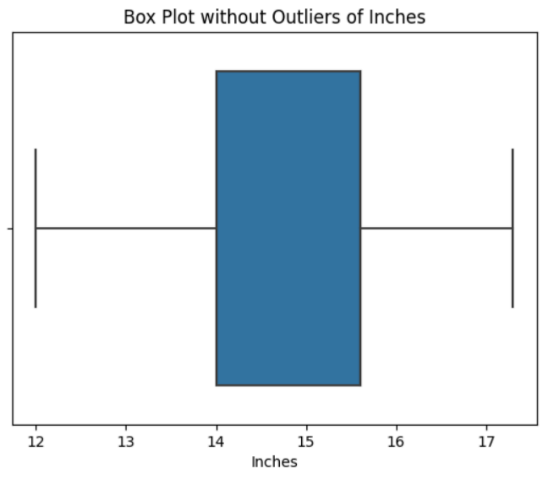


Inches

```
sns.boxplot(x='Inches', data=df)
plt.title(f'Original Box Plot of Price in Inches')
plt.show()
```

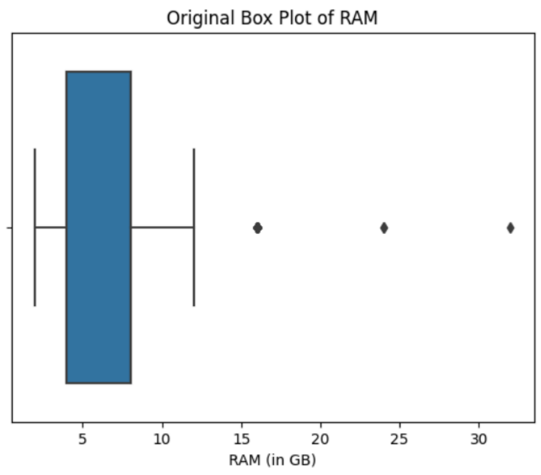


```
]: df2=df[df['Inches']>11.7]
sns.boxplot(x='Inches', data=df2)
plt.title(f'Box Plot without Outliers of Inches')
plt.show()
```

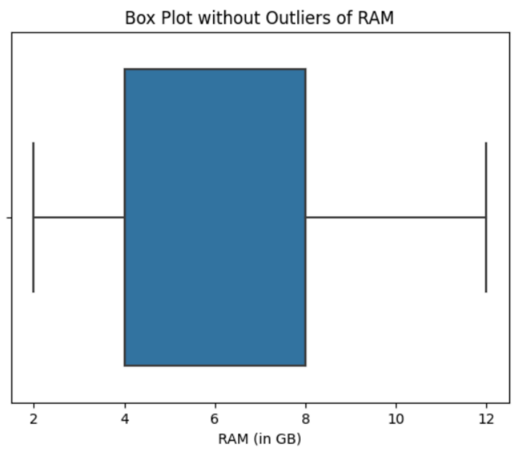


RAM (in GB)

```
sns.boxplot(x='RAM (in GB)', data=df)
plt.title(f'Original Box Plot of RAM')
plt.show()
```

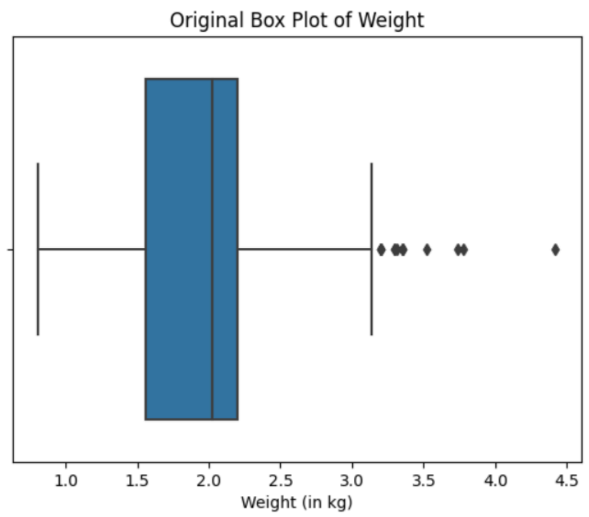


```
df3=df[df['RAM (in GB)']<16]
sns.boxplot(x='RAM (in GB)', data=df3)
plt.title(f'Box Plot without Outliers of RAM')
plt.show()
df=df[df['RAM (in GB)']<16]
```

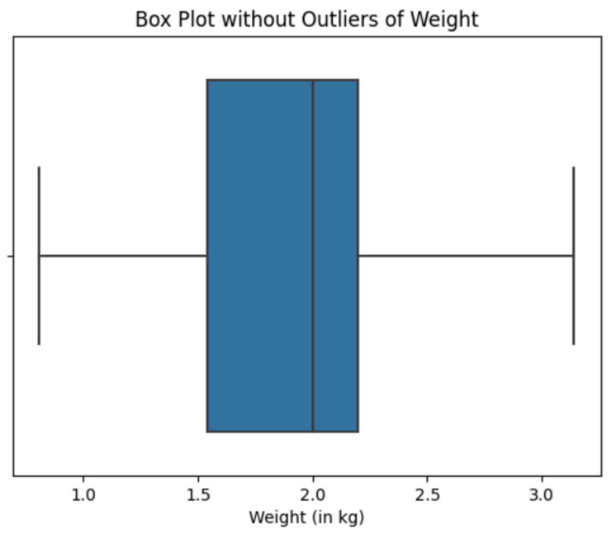


Weight (in kg)

```
sns.boxplot(x='Weight (in kg)', data=df)
plt.title(f'Original Box Plot of Weight')
plt.show()
```



```
df4=df[df['Weight (in kg)']<3.2]
sns.boxplot(x='Weight (in kg)', data=df4)
plt.title(f'Box Plot without Outliers of Weight')
plt.show()
df=df[df['Weight (in kg)']<3.2]
```



#Dataset shape (after removing outliers)

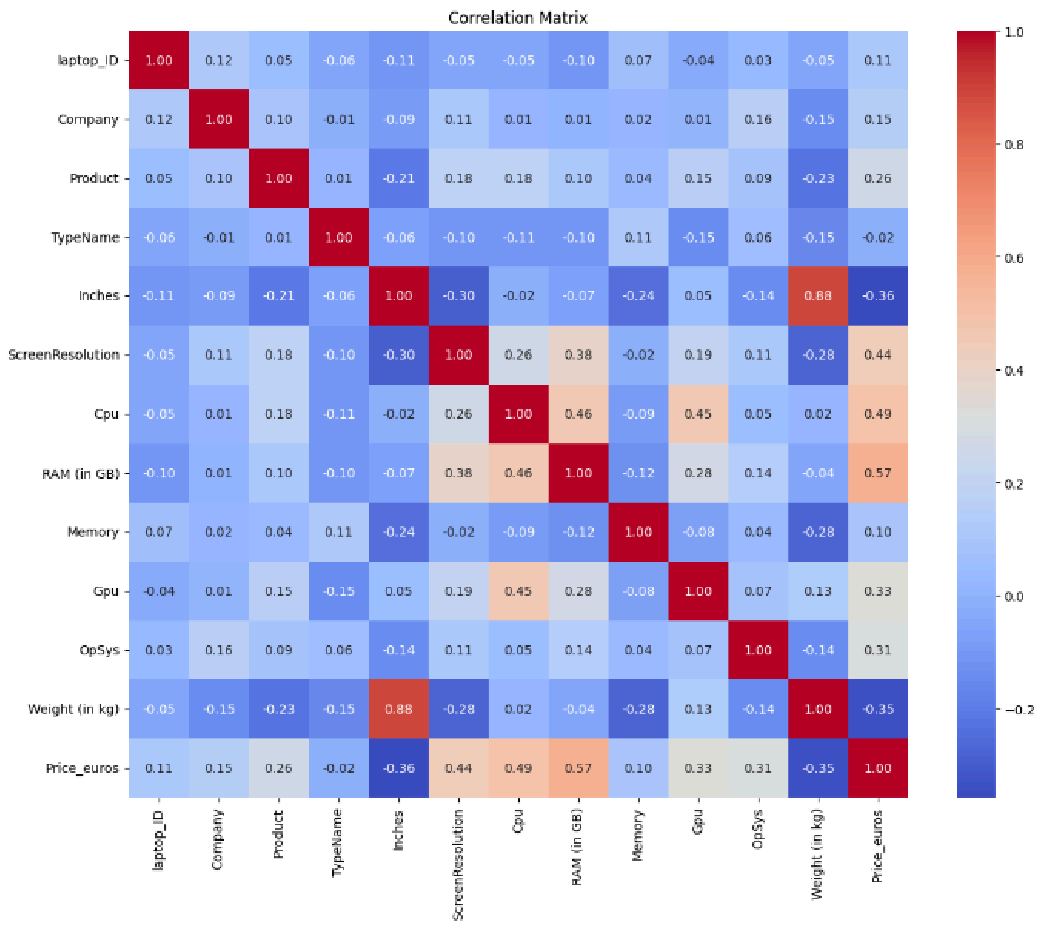
```
df.shape
```

```
(1025, 13)
```

After removing the outliers, the dataset has been reduced from 1303 records to 1025 records, with a total of 278 outlier points eliminated.

#Correlation matrix (after removing outlier)

```
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.tight_layout()
plt.show()
```



	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	RAM (in GB)	Memory	Gpu	OpSys	Weight (in kg)	Price_euros
laptop_ID	1.00	0.12	0.05	-0.06	-0.11	-0.05	-0.05	-0.10	0.07	-0.04	0.03	-0.05	0.11
Company	0.12	1.00	0.10	-0.01	-0.09	0.11	0.01	0.01	0.02	0.01	0.16	-0.15	0.15
Product	0.05	0.10	1.00	0.01	-0.21	0.18	0.18	0.10	0.04	0.15	0.09	-0.23	0.26
TypeName	-0.06	-0.01	0.01	1.00	-0.06	-0.10	-0.11	-0.10	0.11	-0.15	0.06	-0.15	-0.02
Inches	-0.11	-0.09	-0.21	-0.06	1.00	-0.30	-0.02	-0.07	-0.24	0.05	-0.14	0.88	-0.36
ScreenResolution	-0.05	0.11	0.18	-0.10	-0.30	1.00	0.26	0.38	-0.02	0.19	0.11	-0.28	0.44
Cpu	-0.05	0.01	0.18	-0.11	-0.02	0.26	1.00	0.46	-0.09	0.45	0.05	0.02	0.49
RAM (in GB)	-0.10	0.01	0.10	-0.10	-0.07	0.38	0.46	1.00	-0.12	0.28	0.14	-0.04	0.57
Memory	0.07	0.02	0.04	0.11	-0.24	-0.02	-0.09	-0.12	1.00	-0.08	0.04	-0.28	0.10
Gpu	-0.04	0.01	0.15	-0.15	0.05	0.19	0.45	0.28	-0.08	1.00	0.07	0.13	0.33
OpSys	0.03	0.16	0.09	0.06	-0.14	0.11	0.05	0.14	0.04	0.07	1.00	-0.14	0.31
Weight (in kg)	-0.05	-0.15	-0.23	-0.15	0.88	-0.28	0.02	-0.04	-0.28	0.13	-0.14	1.00	-0.35
Price_euros	0.11	0.15	0.26	-0.02	-0.36	0.44	0.49	0.57	0.10	0.33	0.31	-0.35	1.00

Correlation Analysis

The correlation values provided indicate the strength and direction of the linear relationship between Price_euros and other features in the dataset. Here is an analysis of these correlations:

Positive Correlations

Ram (in GB) (0.57): There is a strong positive correlation between Price_euros and Ram, suggesting that laptops with more RAM tend to be more expensive.

Cpu (0.49): There is a strong positive correlation between Price_euros and Cpu, indicating that laptops with more powerful CPUs tend to be more expensive.

ScreenResolution (0.44): A moderately strong positive correlation indicates that higher screen resolutions are associated with higher prices.

Gpu (0.33): A moderate positive correlation suggests that laptops with better GPUs tend to be more expensive.

OpSys (0.31): A moderate positive correlation indicates that the operating system has a notable effect on price.

Product (0.26): A weak positive correlation shows that certain products are priced higher.
Company (0.15): A weak positive correlation indicates that the manufacturer (company) has a slight effect on price.

Memory (0.10): A very weak positive correlation suggests that the memory size has a minimal effect on the price.

laptop_ID (0.11): This weak positive correlation is likely coincidental and not meaningful for price determination.

Negative Correlations

Inches (-0.36): A moderate negative correlation indicates that larger screen sizes tend to be associated with lower prices, which is counterintuitive and may require further investigation.

Weight (in kg) (-0.35): A moderate negative correlation suggests that lighter laptops tend to be more expensive.

TypeName (-0.02): This negligible negative correlation shows that the type of laptop has little to no linear relationship with the price.

Conclusion

1] Ram (in GB), Cpu, and ScreenResolution are significant predictors of Price_euros and have the highest positive correlations, indicating that these features strongly impact the price.

2] Gpu and OpSys also show moderate positive correlations, suggesting that better graphics and certain operating systems are associated with higher prices.

3] The Inches and Weight (in kg) correlations are negative, which suggests that larger and heavier laptops might be cheaper, possibly due to market preferences for more portable devices.

4] The weak correlations of Company, Product, Memory, and laptop_ID suggest that these features have minimal impact on the price compared to others.

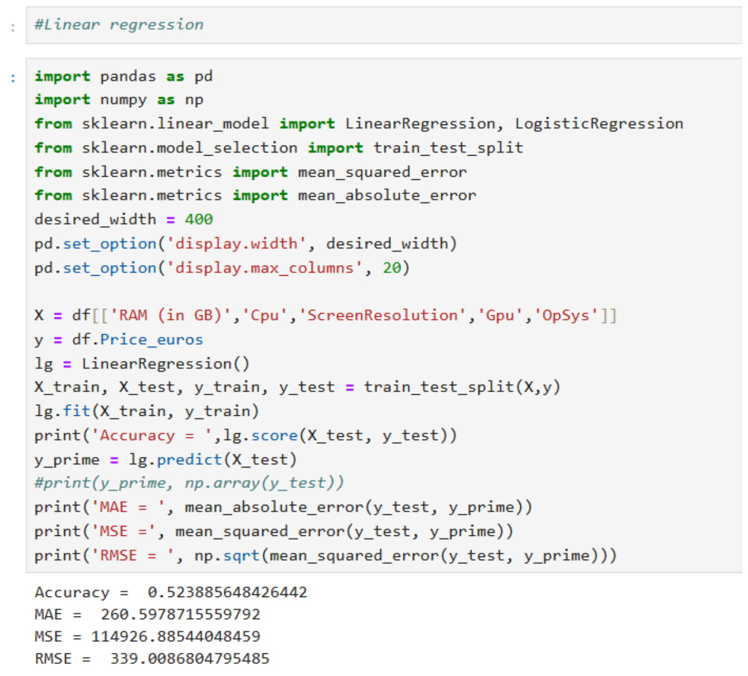
5] Overall, the data suggests that higher prices are associated with better specifications (more RAM, better CPU, higher screen resolution, and better GPU).

Conversely, larger and heavier laptops tend to be cheaper, which might reflect consumer preferences for portability.

Data Mining

1] Linear regression

Based on the correlation analysis, the selected independent variables are RAM (in GB), CPU, ScreenResolution, GPU, and OpSys.



```
#Linear regression
```

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)

X = df[['RAM (in GB)','Cpu','ScreenResolution','Gpu','OpSys']]
y = df.Price_euros
lg = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(X,y)
lg.fit(X_train, y_train)
print('Accuracy = ',lg.score(X_test, y_test))
y_prime = lg.predict(X_test)
#print(y_prime, np.array(y_test))
print('MAE = ', mean_absolute_error(y_test, y_prime))
print('MSE =', mean_squared_error(y_test, y_prime))
print('RMSE = ', np.sqrt(mean_squared_error(y_test, y_prime)))
```

```
Accuracy =  0.523885648426442
MAE =  260.5978715559792
MSE = 114926.88544048459
RMSE =  339.0086804795485
```

The results of your linear regression model can be analyzed as follows:

Accuracy (R^2 Score):

Accuracy = 0.523885648426442
The R^2 score, or coefficient of determination, is approximately 0.52. This means that around 52% of the variance in the target variable (Price_euros) is explained by the independent variables in the model. While this indicates that the model has some predictive power, there is still a substantial portion of the variance (48%) that is not captured by the model. This suggests that other factors not included in the model might also play a significant role in determining laptop prices.

Mean Absolute Error (MAE):

MAE = 260.5978715559792
The MAE indicates that, on average, the predictions of the model differ from the actual prices by about 260.60 euros. This metric is useful for understanding the average magnitude of errors in the predictions, regardless of their direction (positive or negative).

Mean Squared Error (MSE):

MSE = 114926.88544048459
The MSE measures the average squared difference between the predicted and actual values. The large value of MSE suggests that there are some predictions with large errors, as squaring the errors disproportionately increases the impact of larger errors.

Root Mean Squared Error (RMSE):

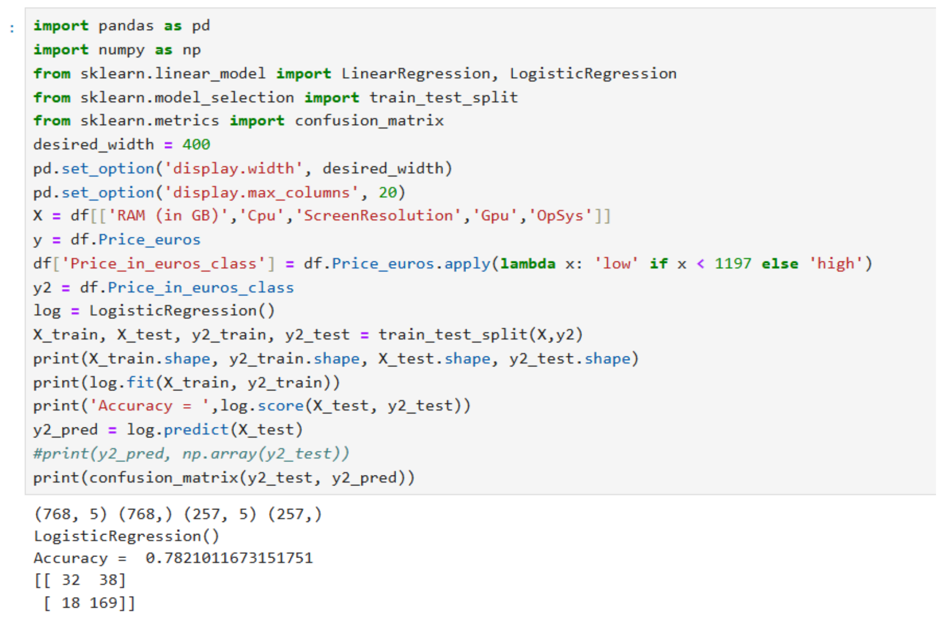
RMSE = 339.0086804795485
The RMSE is the square root of the MSE and provides an error metric in the same units as the target variable (euros). An RMSE of about 339.01 euros indicates the standard deviation of the residuals (prediction errors). It gives a sense of how far off the predictions typically are from the actual prices.

Conclusion:

The model's R^2 score of 0.52 suggests moderate predictive power, indicating that while the selected features do have a significant impact on predicting laptop prices, they do not capture the entire complexity of price determination.

The error metrics (MAE, MSE, and RMSE) show that the predictions are not extremely accurate, with an average error of about 260.60 euros and a standard deviation of prediction errors of about 339.01 euros.

2] Logistic regression



```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
X = df[['RAM (in GB)','Cpu','ScreenResolution','Gpu','OpSys']]
y = df.Price_euros
df['Price_in_euros_class'] = df.Price_euros.apply(lambda x: 'low' if x < 1197 else 'high')
y2 = df.Price_in_euros_class
log = LogisticRegression()
X_train, X_test, y2_train, y2_test = train_test_split(X,y2)
print(X_train.shape, y2_train.shape, X_test.shape, y2_test.shape)
print(log.fit(X_train, y2_train))
print('Accuracy = ',log.score(X_test, y2_test))
y2_pred = log.predict(X_test)
#print(y2_pred, np.array(y2_test))
print(confusion_matrix(y2_test, y2_pred))
```

```
(768, 5) (768,) (257, 5) (257,)
LogisticRegression()
Accuracy =  0.7821011673151751
[[ 32  38]
 [ 18 169]]
```

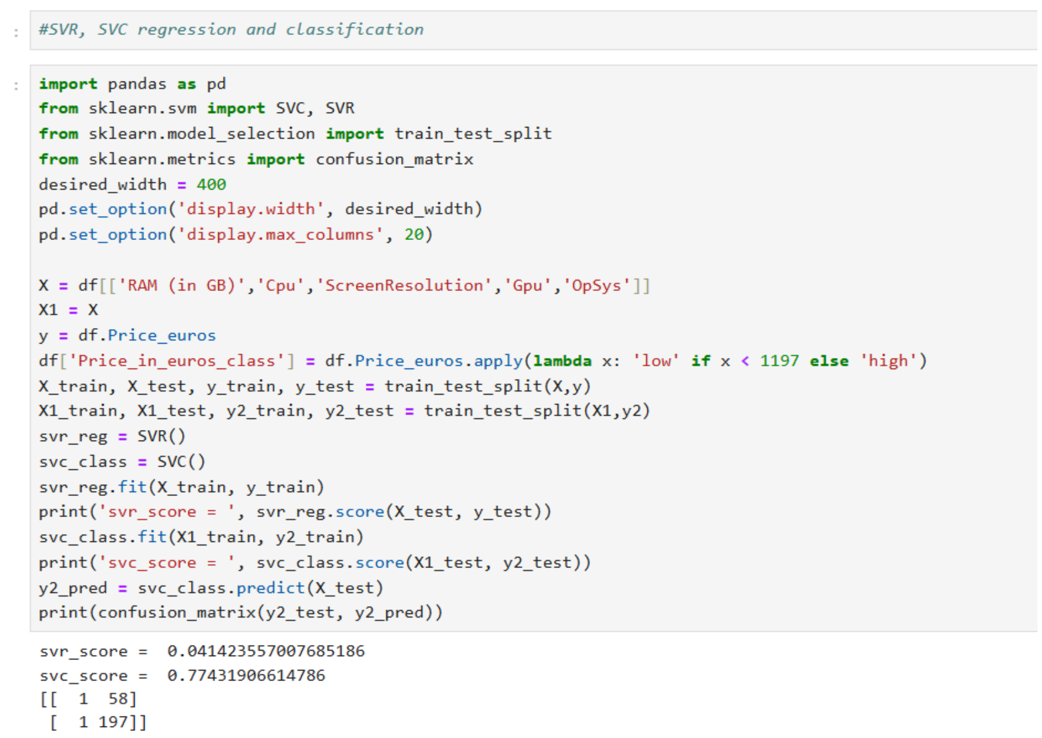
Results:

The accuracy of the logistic regression model on the test data is approximately 0.78 (or 78.21%).
The confusion matrix shows the following:
True Negative (TN): 32
False Positive (FP): 38
False Negative (FN): 18
True Positive (TP): 169

Interpretation:

The accuracy of 78.21% indicates that the model performs reasonably well in predicting whether the price of a laptop is "low" or "high" based on the given features.
The confusion matrix provides more insight into the model's performance. It shows that the model misclassified 38 instances of "low" prices as "high" (FP) and 18 instances of "high" prices as "low" (FN).

3] SVR, SVC Regression and Classification



```
#SVR, SVC regression and classification
```

```
import pandas as pd
from sklearn.svm import SVC, SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)

X = df[['RAM (in GB)','Cpu','ScreenResolution','Gpu','OpSys']]
X1 = X
y = df.Price_euros
df['Price_in_euros_class'] = df.Price_euros.apply(lambda x: 'low' if x < 1197 else 'high')
X_train, X_test, y_train, y_test = train_test_split(X,y)
X1_train, X1_test, y2_train, y2_test = train_test_split(X1,y2)
svr_reg = SVR()
svc_class = SVC()
svr_reg.fit(X_train, y_train)
print('svr_score = ', svr_reg.score(X_test, y_test))
svc_class.fit(X1_train, y2_train)
print('svc_score = ', svc_class.score(X1_test, y2_test))
y2_pred = svc_class.predict(X_test)
print(confusion_matrix(y2_test, y2_pred))
```

```
svr_score =  0.041423557007685186
svc_score =  0.77431906614786
[[  1  58]
 [  1 197]]
```

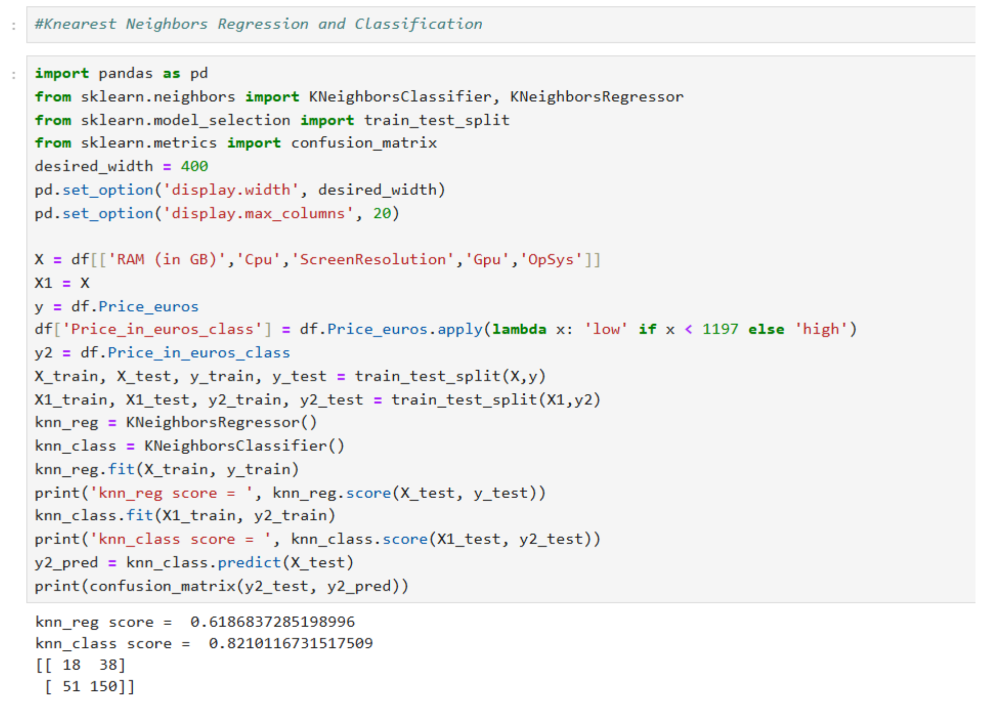
Support Vector Machine Regression (SVR):

Score: 0.0414 (4.14%)
Brief Interpretation: Poor performance in predicting continuous target variable (likely laptop prices).

Support Vector Machine Classification (SVC):

Score: 0.7743 (77.43%)
Confusion Matrix:
TN: 1
FP: 58
FN: 1
TP: 197
Interpretation: Decent performance in classifying laptop prices as "low" or "high," but room for improvement in reducing misclassifications.

4] K nearest neighbors regression and classification



```
#Knearest Neighbors Regression and Classification
```

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)

X = df[['RAM (in GB)','Cpu','ScreenResolution','Gpu','OpSys']]
X1 = X
y = df.Price_euros
df['Price_in_euros_class'] = df.Price_euros.apply(lambda x: 'low' if x < 1197 else 'high')
y2 = df.Price_in_euros_class
X_train, X_test, y_train, y_test = train_test_split(X,y)
X1_train, X1_test, y2_train, y2_test = train_test_split(X1,y2)
knn_reg = KNeighborsRegressor()
knn_class = KNeighborsClassifier()
knn_reg.fit(X_train, y_train)
print('knn_reg score = ', knn_reg.score(X_test, y_test))
knn_class.fit(X1_train, y2_train)
print('knn_class score = ', knn_class.score(X1_test, y2_test))
y2_pred = knn_class.predict(X_test)
print(confusion_matrix(y2_test, y2_pred))
```

```
knn_reg score =  0.6186837285198996
knn_class score =  0.8210116731517509
[[ 18  38]
 [ 51 150]]
```

K Nearest Neighbors Regression (KNN Reg):

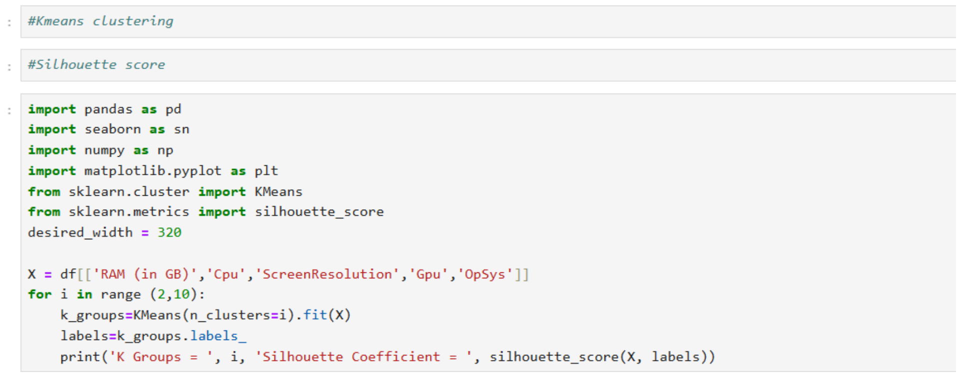
Score: 0.6187 (61.87%)
Interpretation: Moderate performance in predicting continuous target variables (likely laptop prices).

K Nearest Neighbors Classification (KNN Class):

Score: 0.8210 (82.10%)
Confusion Matrix:
TN: 18
FP: 38
FN: 51
TP: 150
Interpretation: Good performance in classifying laptop prices as "low" or "high," but some misclassifications observed, especially false negatives.

5] KMeans Clustering

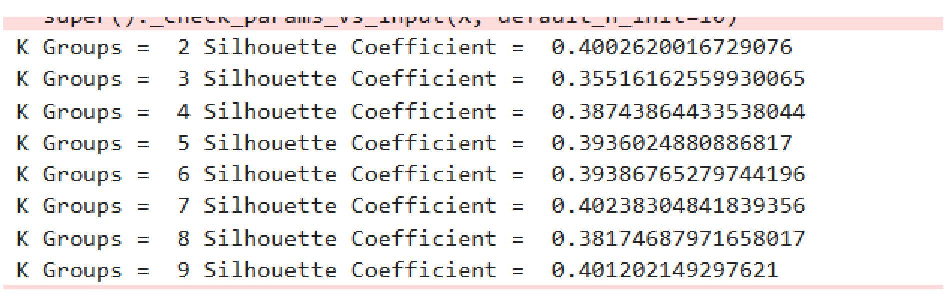
#Silhouette score



```
#Kmeans clustering
```

```
#Silhouette score
```

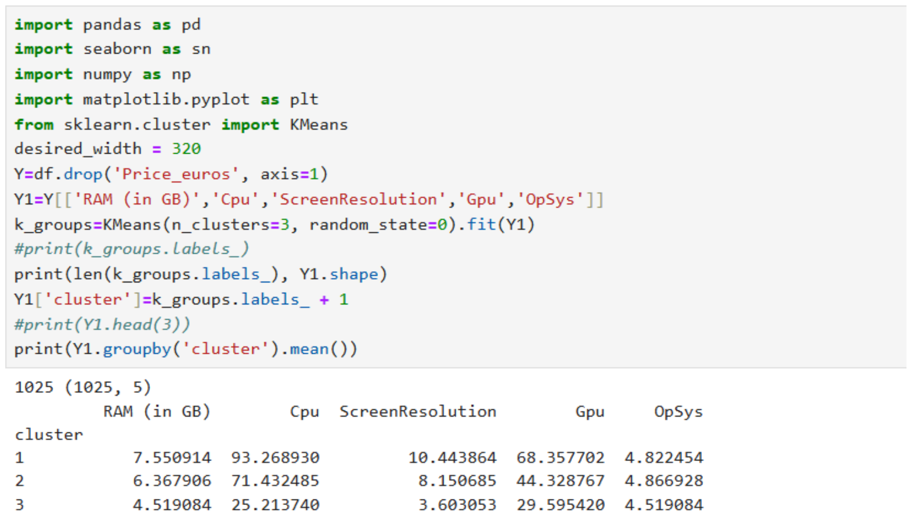
```
import pandas as pd
import seaborn as sn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
desired_width = 320

X = df[['RAM (in GB)','Cpu','ScreenResolution','Gpu','OpSys']]
for i in range (2,10):
    k_groups=KMeans(n_clusters=i).fit(X)
    labels=k_groups.labels_
    print('K Groups = ', i, 'Silhouette Coefficient = ', silhouette_score(X, labels))
```



```
  super()._check_params_vs_input(X, default_n_init=10)
K Groups =  2 Silhouette Coefficient =  0.4002620016729076
K Groups =  3 Silhouette Coefficient =  0.35516162559930065
K Groups =  4 Silhouette Coefficient =  0.38743864433538044
K Groups =  5 Silhouette Coefficient =  0.3936024880886817
K Groups =  6 Silhouette Coefficient =  0.39386765279744196
K Groups =  7 Silhouette Coefficient =  0.40238304841839356
K Groups =  8 Silhouette Coefficient =  0.38174687971658017
K Groups =  9 Silhouette Coefficient =  0.401202149297621
```

The Silhouette score indicates that for values of k equal to 2, 7, and 9, the score peaks at around 40-41%.

#Clustering into 3 groups



```
import pandas as pd
import seaborn as sn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
desired_width = 320
Y=df.drop('Price_euros', axis=1)
Y1=Y[['RAM (in GB)','Cpu','ScreenResolution','Gpu','OpSys']]
k_groups=KMeans(n_clusters=3, random_state=0).fit(Y1)
#print(k_groups.labels_)
print(len(k_groups.labels_), Y1.shape)
Y1['cluster']=k_groups.labels_ + 1
#print(Y1.head(3))
print(Y1.groupby('cluster').mean())
```

```
1025 (1025, 5)
         RAM (in GB)        Cpu  ScreenResolution        Gpu     OpSys
cluster
1           7.550914  93.268930         10.443864  68.357702  4.822454
2           6.367906  71.432485          8.150685  44.328767  4.866928
3           4.519084  25.213740          3.603053  29.595420  4.519084
```

Cluster 1 - High-Performance Laptops:

This cluster consists of laptops with high specifications across all performance parameters, including high RAM, CPU, GPU, and screen resolution ratings. They likely offer top-tier performance and are suitable for demanding tasks such as gaming, video editing, or professional software development.

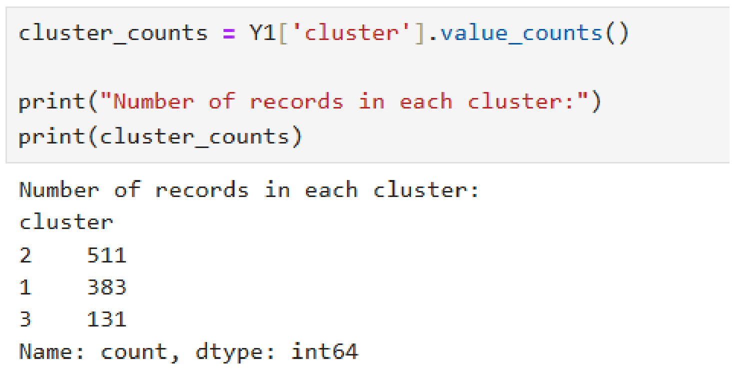
Cluster 2 - Mid-Range Performance Laptops:

Laptops in this cluster have moderate specifications, with slightly lower ratings compared to Cluster 1. While they may not offer the highest level of performance, they still provide respectable performance for everyday tasks, multimedia consumption, and some light gaming or productivity work.

Cluster 3 - Entry-Level Laptops:

This cluster represents entry-level laptops with lower specifications across all performance parameters. They offer basic functionality for general use, such as web browsing, word processing, and multimedia consumption, but may struggle with more demanding tasks like gaming or heavy multitasking.
These names reflect the relative performance levels and target audiences of the laptops in each cluster.

#Cluster counts

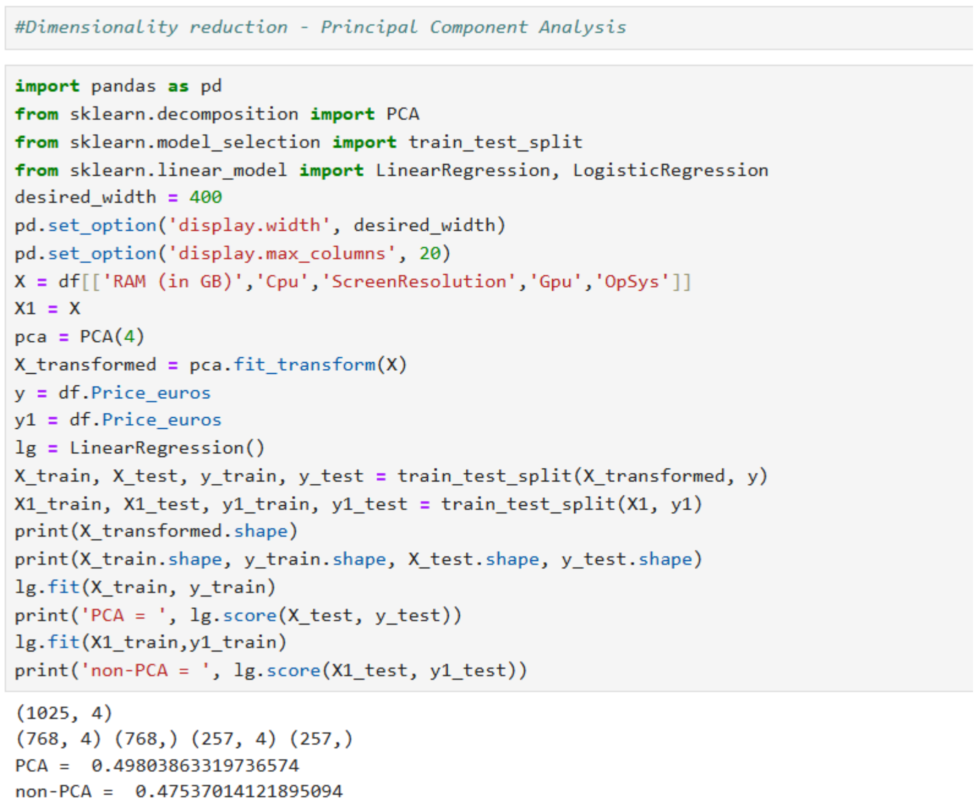


```
cluster_counts = Y1['cluster'].value_counts()

print("Number of records in each cluster:")
print(cluster_counts)
```

```
Number of records in each cluster:
cluster
2    511
1    383
3    131
Name: count, dtype: int64
```

Out of the total 1025 records, 383 records are categorized under Cluster 1, denoting high-performance laptops, 511 records fall under Cluster 2, representing laptops with medium performance, and 131 records are assigned to Cluster 3, indicative of low-performance laptops.

6] Principal Component Analysis (PCA)



```
#Dimensionality reduction - Principal Component Analysis
```

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
X = df[['RAM (in GB)','Cpu','ScreenResolution','Gpu','OpSys']]
X1 = X
pca = PCA(4)
X_transformed = pca.fit_transform(X)
y = df.Price_euros
y1 = df.Price_euros
lg = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1)
print(X_transformed.shape)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
lg.fit(X_train, y_train)
print('PCA = ', lg.score(X_test, y_test))
lg.fit(X1_train,y1_train)
print('non-PCA = ', lg.score(X1_test, y1_test))
```

```
(1025, 4)
(768, 4) (768,) (257, 4) (257,)
PCA =  0.49803863319736574
non-PCA =  0.47537014121895094
```

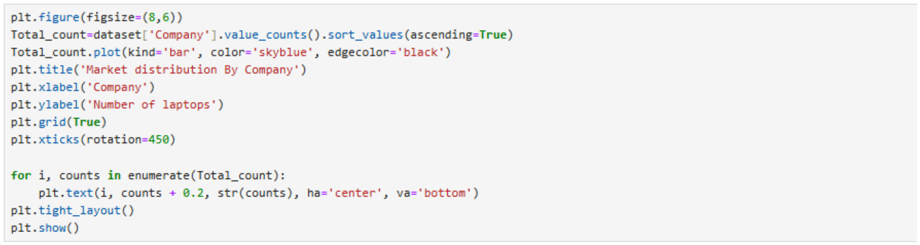
The linear regression model trained on the PCA-transformed data outperformed the model trained on the original data, with a higher R^2 score of 0.498 compared to 0.475.

This suggests that PCA helped in capturing important patterns or reducing noise in the data, resulting in a more effective linear regression model.

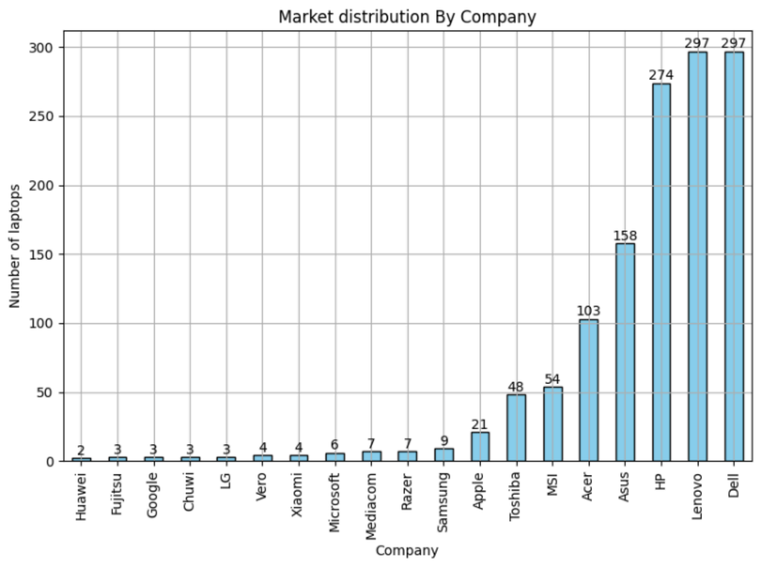
Data Visualization (Managerial questions that can be answered):

Market Share by Company:

Which companies hold the largest share of the laptop market based on the number of products sold?



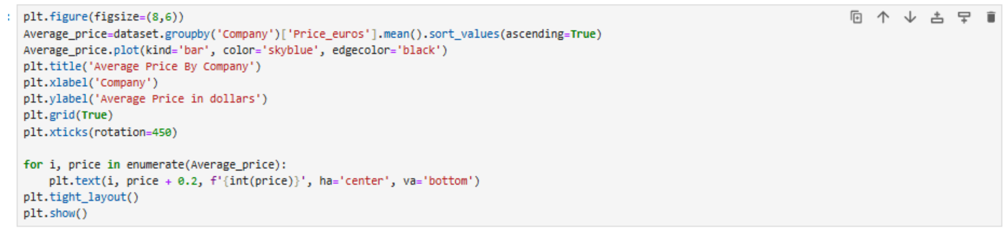
```
plt.figure(figsize=(8,6))
Total_count=dataset['Company'].value_counts().sort_values(ascending=True)
Total_count.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Market distribution By Company')
plt.xlabel('Company')
plt.ylabel('Number of laptops')
plt.grid(True)
plt.xticks(rotation=450)

for i, counts in enumerate(Total_count):
    plt.text(i, counts + 0.2, str(counts), ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



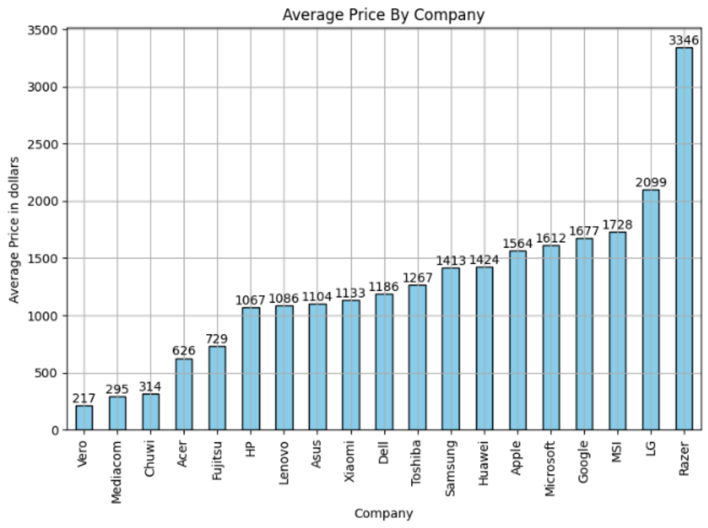
From the bar graph, it is evident that Dell, Lenovo, and HP have the highest market share among the companies in the laptop market.

Product Pricing:

How do laptop prices vary across different companies?



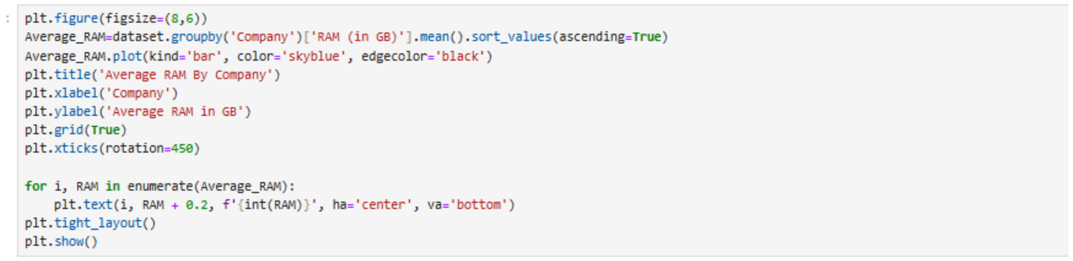
```
plt.figure(figsize=(8,6))
Average_price=dataset.groupby('Company')['Price_euros'].mean().sort_values(ascending=True)
Average_price.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Average Price By Company')
plt.xlabel('Company')
plt.ylabel('Average Price in dollars')
plt.grid(True)
plt.xticks(rotation=450)

for i, price in enumerate(Average_price):
    plt.text(i, price + 0.2, f'{int(price)}', ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



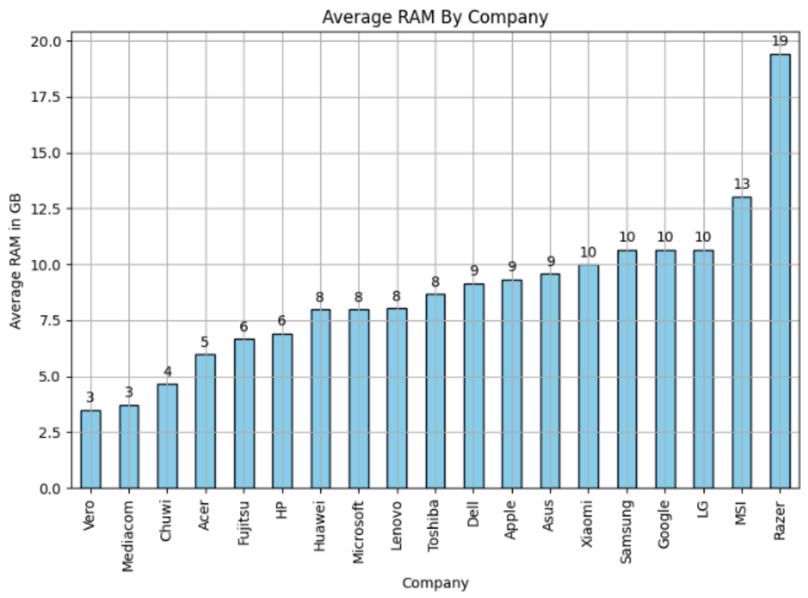
The bar graph clearly shows that Razer has the highest average price, with LG and MSI following closely behind.

Performance Analysis:

What are the average specifications (RAM) of laptops offered by different companies?



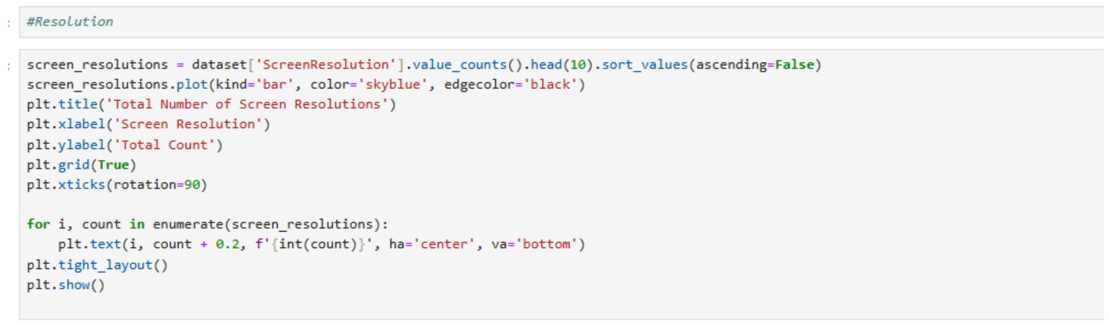
```
plt.figure(figsize=(8,6))
Average_RAM=dataset.groupby('Company')['RAM (in GB)'].mean().sort_values(ascending=True)
Average_RAM.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Average RAM By Company')
plt.xlabel('Company')
plt.ylabel('Average RAM in GB')
plt.grid(True)
plt.xticks(rotation=450)

for i, RAM in enumerate(Average_RAM):
    plt.text(i, RAM + 0.2, f'{int(RAM)}', ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



Based on the figure, it appears that Razer, MSI, LG, Google, Samsung, and Xiaomi are among the companies offering laptops with the highest RAM specifications.

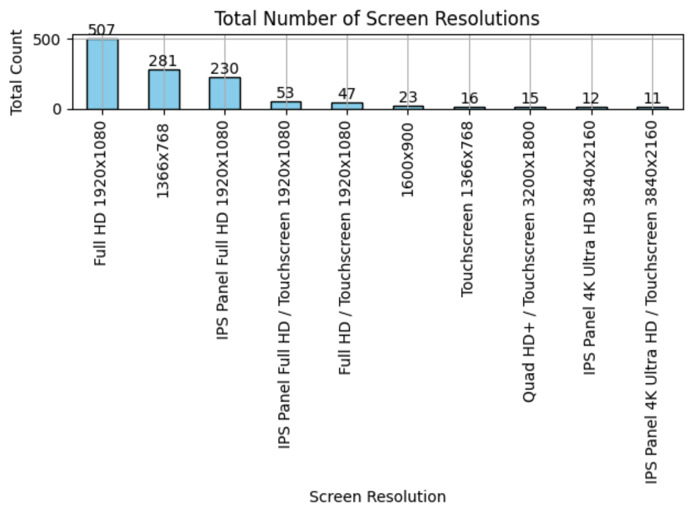
Customer Preferences:

What are the most common screen resolutions and sizes preferred by customers?



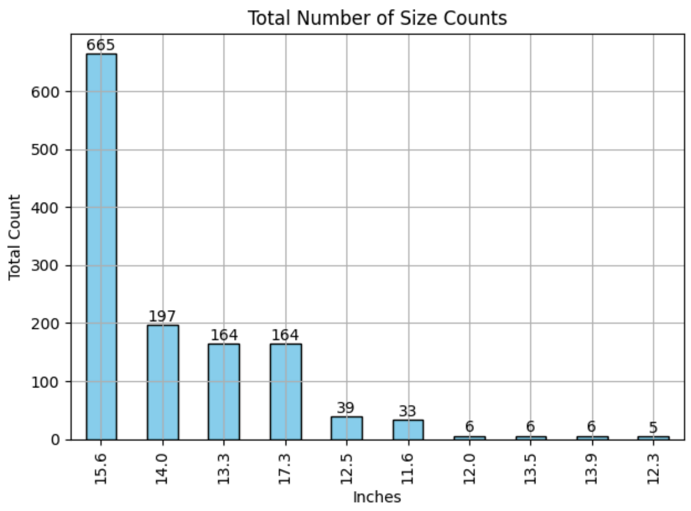
```
#Resolution
```

```
screen_resolutions = dataset['ScreenResolution'].value_counts().head(10).sort_values(ascending=False)
screen_resolutions.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Total Number of Screen Resolutions')
plt.xlabel('Screen Resolution')
plt.ylabel('Total Count')
plt.grid(True)
plt.xticks(rotation=90)

for i, count in enumerate(screen_resolutions):
    plt.text(i, count + 0.2, f'{int(count)}', ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



From the graph, the most common resolutions are Full HD 1920x1080, 1366x768, IPS Panel Full HD 1920x1080, IPS Panel Full HD/Touchscreen 1920x1080, and Full HD/Touchscreen 1920x1080.

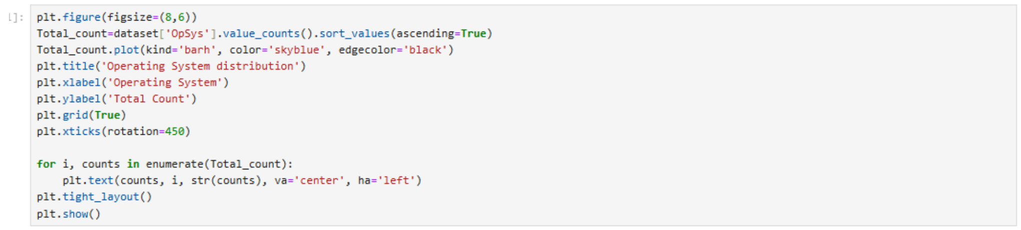
```python
Inches = dataset['Inches'].value_counts().head(10).sort_values(ascending=False)
Inches.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Total Number of Size Counts')
plt.xlabel('Inches')
plt.ylabel('Total Count')
plt.grid(True)
plt.xticks(rotation=90)

for i, count in enumerate(Inches):
    plt.text(i, count + 0.2, f'{int(count)}', ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



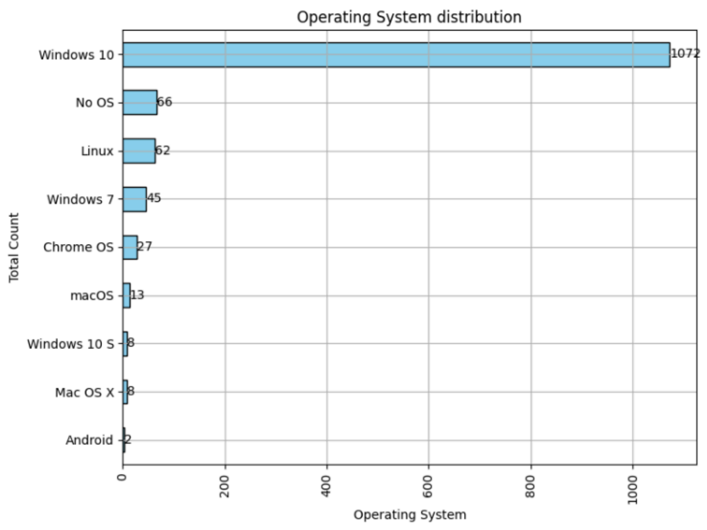
The most preferred screen sizes by customers are 15.6 inches, 14 inches, 13.3 inches, and 17.3 inches.

Operating System Distribution:

What is the distribution of operating systems (OpSys) used across different laptop models?



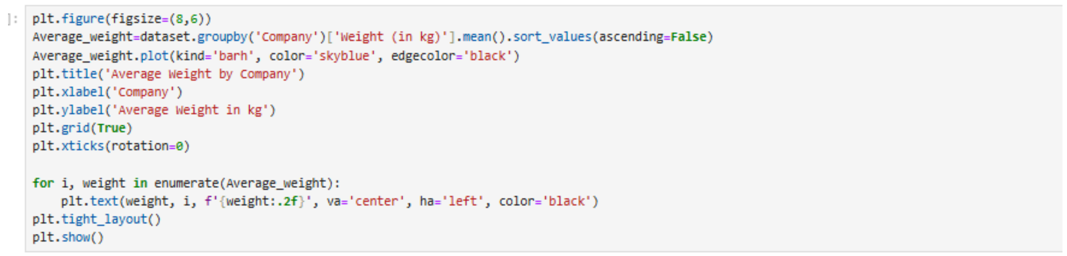
```
plt.figure(figsize=(8,6))
Total_count=dataset['OpSys'].value_counts().sort_values(ascending=True)
Total_count.plot(kind='barh', color='skyblue', edgecolor='black')
plt.title('Operating System distribution')
plt.xlabel('Operating System')
plt.ylabel('Total Count')
plt.grid(True)
plt.xticks(rotation=450)

for i, counts in enumerate(Total_count):
    plt.text(counts, i, str(counts), va='center', ha='left')
plt.tight_layout()
plt.show()
```



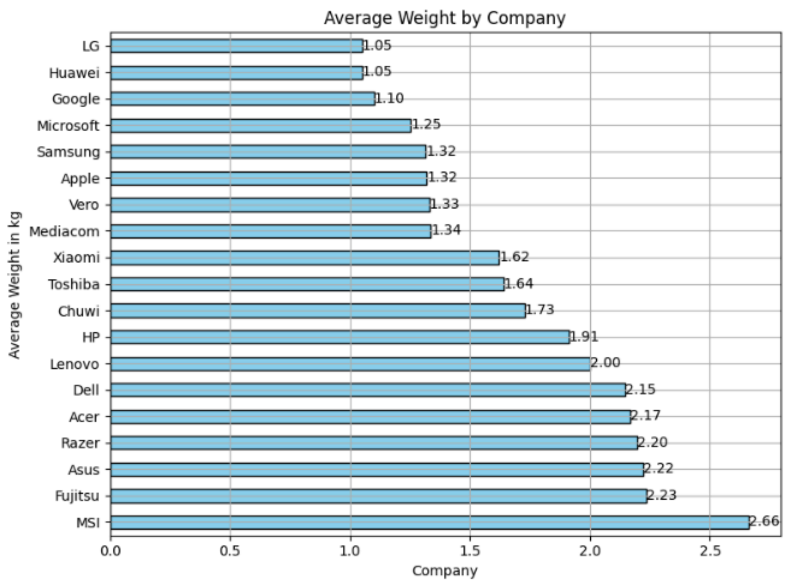
According to the figure, customers predominantly favor the Windows 10 operating system.

Weight and Portability:

How does the weight of laptops vary across different types and companies, and which brands offer the lightest laptops?



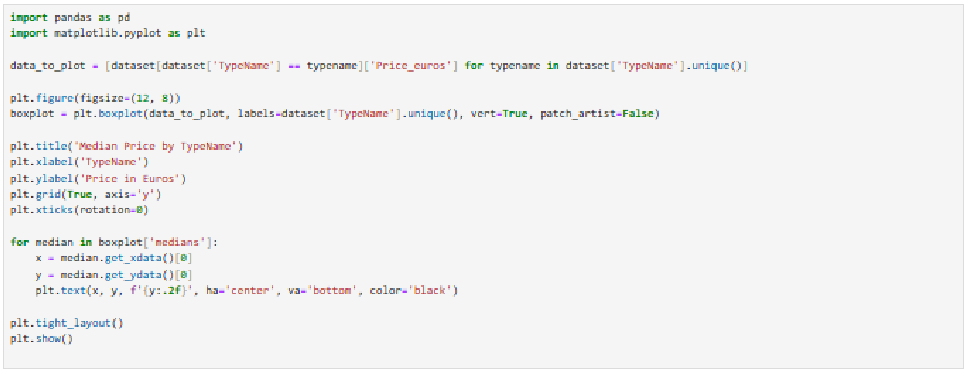
```
[ ]: plt.figure(figsize=(8,6))
     Average_weight=dataset.groupby('Company')['Weight (in kg)'].mean().sort_values(ascending=False)
     Average_weight.plot(kind='barh', color='skyblue', edgecolor='black')
     plt.title('Average Weight by Company')
     plt.xlabel('Company')
     plt.ylabel('Average Weight in kg')
     plt.grid(True)
     plt.xticks(rotation=0)

     for i, weight in enumerate(Average_weight):
         plt.text(weight, i, f'{weight:.2f}', va='center', ha='left', color='black')
     plt.tight_layout()
     plt.show()
```



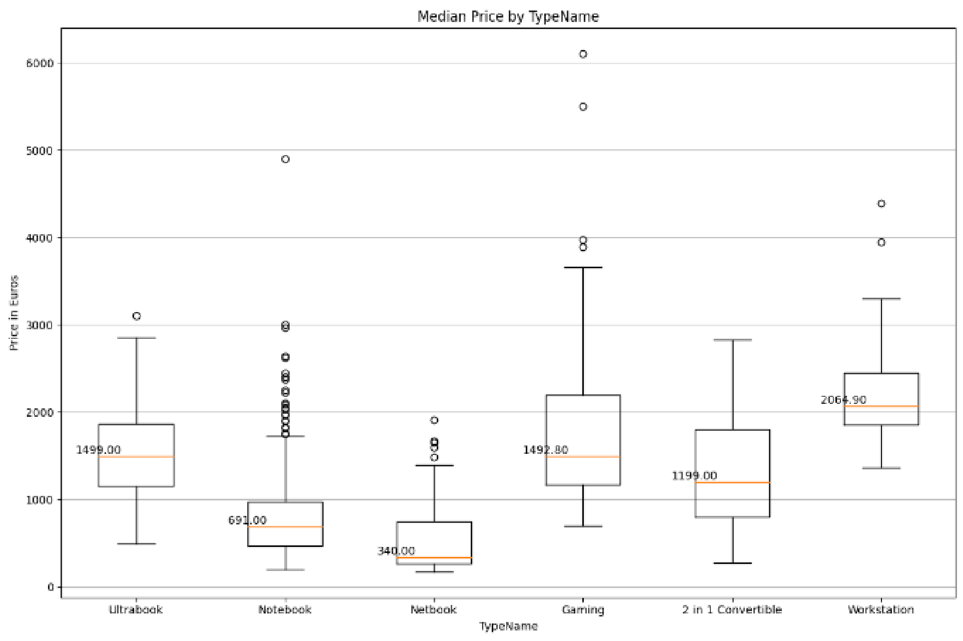
The figure clearly shows that LG, Huawei, and Google are the brands offering the lightest laptops.

Product Segmentation:

What are the different segments (TypeName) of laptops offered by companies, and how are they priced?



```python
import pandas as pd
import matplotlib.pyplot as plt

data_to_plot = [dataset[dataset['TypeName'] == typename]['Price_euros'] for typename in dataset['TypeName'].unique()]

plt.figure(figsize=(12, 8))
boxplot = plt.boxplot(data_to_plot, labels=dataset['TypeName'].unique(), vert=True, patch_artist=False)

plt.title('Median Price by TypeName')
plt.xlabel('TypeName')
plt.ylabel('Price in Euros')
plt.grid(True, axis='y')
plt.xticks(rotation=0)

for median in boxplot['medians']:
    x = median.get_xdata()[0]
    y = median.get_ydata()[0]
    plt.text(x, y, f'{y:.2f}', ha='center', va='bottom', color='black')

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



The boxplot reveals that Workstation, Ultrabook, and Gaming TypeNames have the highest median price values.