EDA

Project Sumamry

The fluctuation in laptop prices based on specifications and manufacturers poses a significant challenge for consumers. Determining the fair market value of a laptop becomes a complex task due to various factors at play. Factors such as processor speed, RAM, storage capacity, graphics card, and brand reputation all contribute to the price variation. Additionally, the constant emergence of new models and advancements further complicates the evaluation process. As a result, consumers face the dilemma of choosing a laptop that meets their requirements without overpaying or compromising on quality.

The goal of this project is to develop a predictive model that accurately determines laptop prices based on specifications and manufacturers. To accomplish this, a dataset from Kaggle will be utilized. The dataset contains information on various laptop models, including their specifications, prices, and manufacturers. This dataset will serve as the foundation for the analysis and model-building process, and will be cleaned and modified as necessary.

To gain a better understanding of how the features influence each other and the price, exploratory data analysis (EDA) techniques will be employed on the dataset. This will help uncover insights and patterns regarding the relationships between laptop prices, specifications, and manufacturers.

Multiple models will be constructed to analyze the relationship between laptop prices and their specifications. These models will provide valuable insights into which features have the greatest impact on the price. Various metrics will be used to compare and evaluate the performance of the models, enabling the selection of the best-fitting model.

By accurately predicting laptop prices based on specifications and manufacturers, this project aims to assist consumers in making informed decisions when purchasing laptops. It will provide valuable insights into the fair market value of laptops and help consumers avoid overpaying or compromising on quality.

Data Exploration

In [1]:
```
#import necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#imports the data set and shows the size of it.
laptop_df = pd.read_csv(
    "C:/Users/predi/Documents/GitHub/DSC550/Datasets/Term/laptop_price.csv",
    encoding='latin-1')
laptop_df.shape
```

Out[2]:
```
(1303, 13)
```

In [3]:
```
#checks for any missing values
laptop_df.isnull().sum()
```

Out[3]:
```
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_in_euros      0
dtype: int64
```

In [4]:
```
#shows five random rows from the data set
laptop_df.sample(5)
```

Out[4]:

	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu
231	236	HP	15-rb013nv (E2-9000e/4GB/500GB/W10)	Notebook	15.6	1366x768	AMD E-Series 9000e 1.5GHz	4GB	500GB HDD	AMD Radeon R2
69	71	Asus	FX753VE-GC093 (i7-7700HQ/12GB/1TB/GeForce	Gaming	17.3	Full HD 1920x1080	Intel Core i7 7700HQ 2.8GHz	12GB	1TB HDD	Nvidia GeForce GTX 1050 Ti
1212	1230	MSI	GS73VR Stealth	Gaming	17.3	IPS Panel Full HD 1920x1080	Intel Core i7 6700HQ 2.6GHz	16GB	256GB SSD + 1TB HDD	Nvidia GeForce GTX 1060
500	507	Asus	VivoBook E201NA	Netbook	11.6	1366x768	Intel Celeron Dual Core N3350 1.1GHz	4GB	64GB Flash Storage	Intel HD Graphics 500
1036	1050	HP	ProBook 450	Notebook	15.6	1366x768	Intel Core i5 7200U 2.5GHz	4GB	500GB HDD	Intel HD Graphics 620

In [5]:
```
#Gets summary statistics of the numerical data
laptop_df.describe()
```

Out[5]:

	laptop_ID	Inches	Price_in_euros
count	1303.000000	1303.000000	1303.000000
mean	660.155794	15.017191	1123.686992
std	381.172104	1.426304	699.009043
min	1.000000	10.100000	174.000000
25%	331.500000	14.000000	599.000000
50%	659.000000	15.600000	977.000000
75%	990.500000	15.600000	1487.880000
max	1320.000000	18.400000	6099.000000

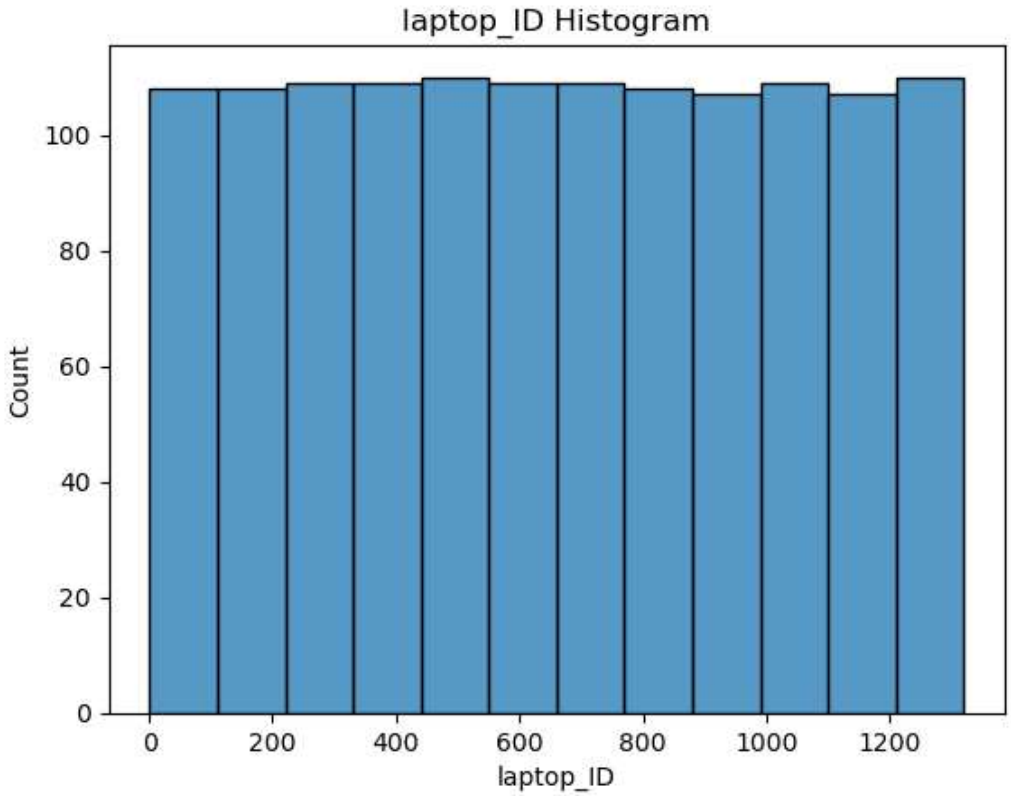
In [6]:
```
#Gets summary statistics of the non-numerical data
laptop_df.describe(include = ['O'])
```

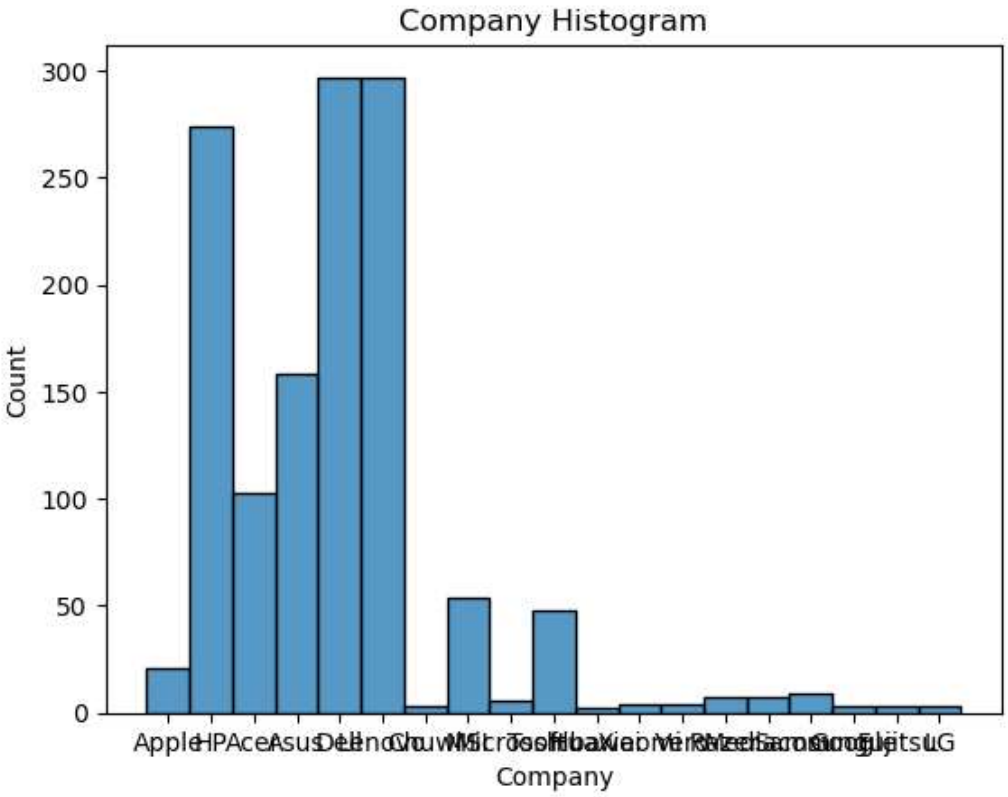
Out[6]:

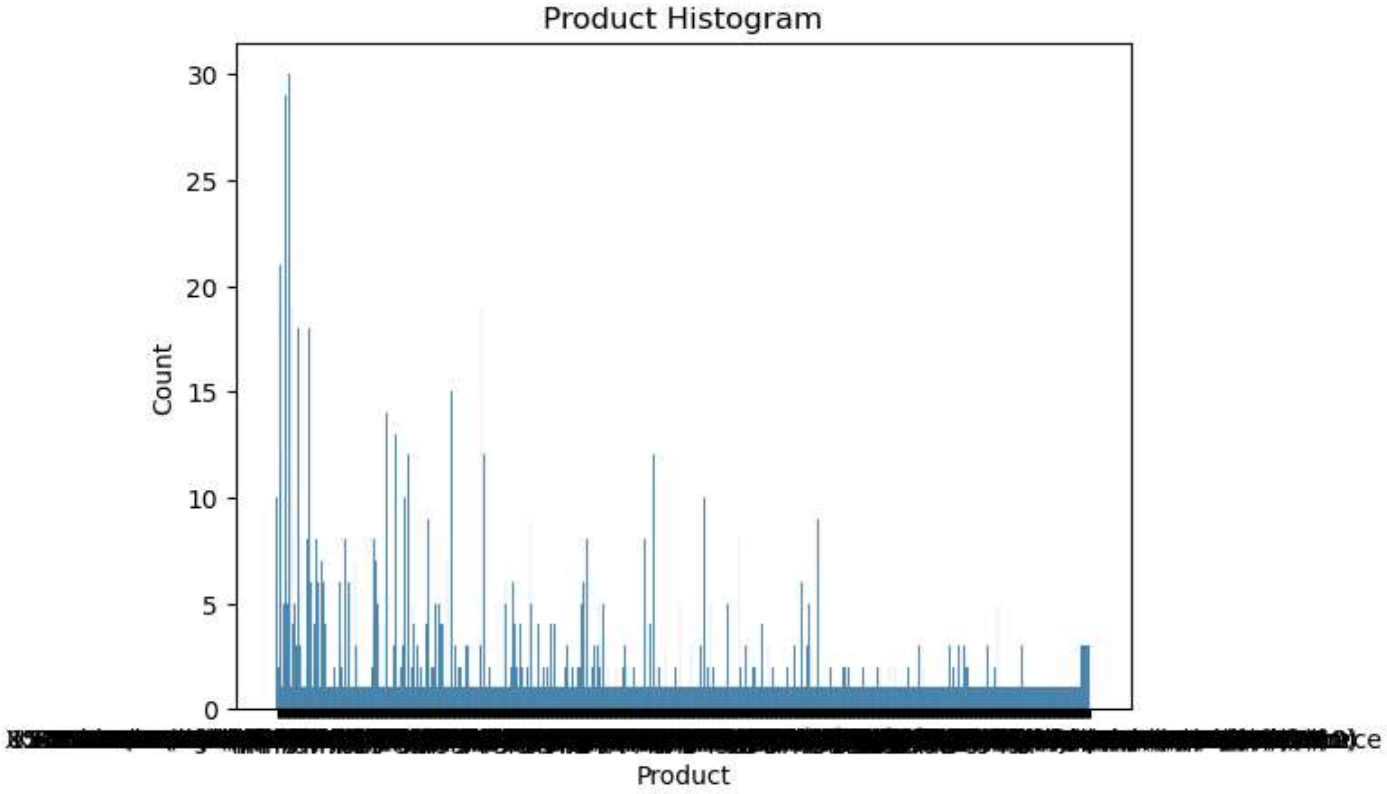
	Company	Product	TypeName	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight
count	1303	1303	1303	1303	1303	1303	1303	1303	1303	1303
unique	19	618	6	40	118	9	39	110	9	179
top	Dell	XPS 13	Notebook	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	Windows 10	2.2kg
freq	297	30	727	507	190	619	412	281	1072	121

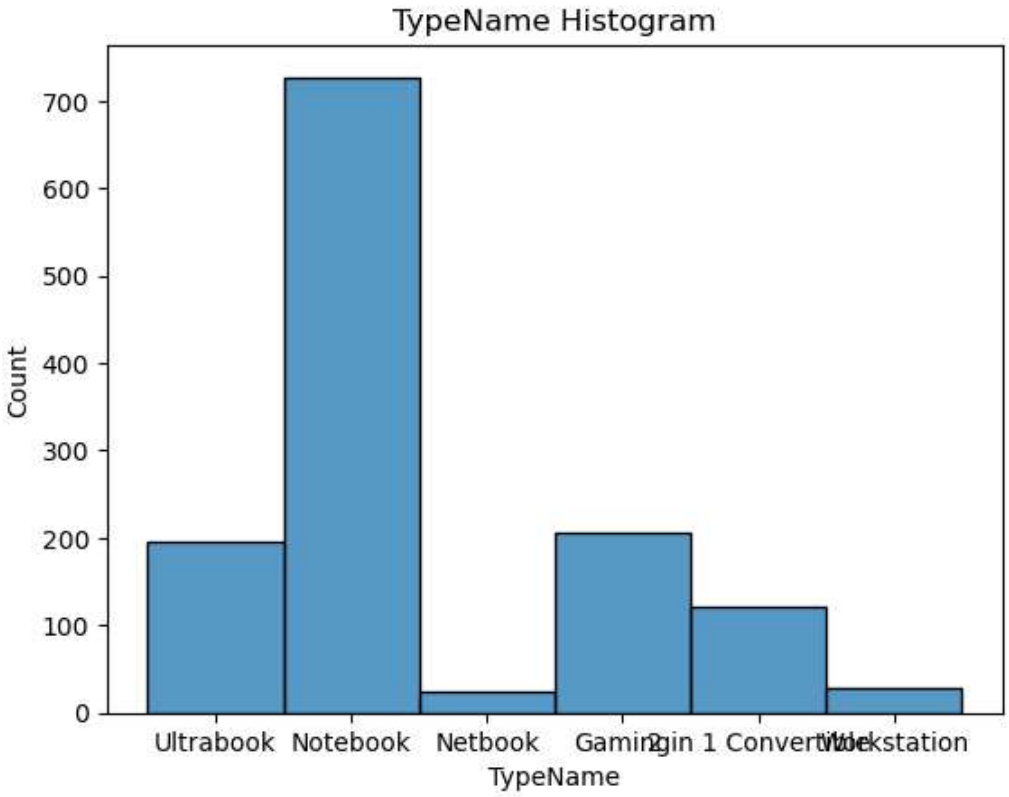
Aware the X-Axis for a few histograms run together. No reason to fix/change though due to the distribution still being understood. The only othe roption would be to change to ordain values, but this wouldn't add any additional insight. So leaving as is for intial EDA.

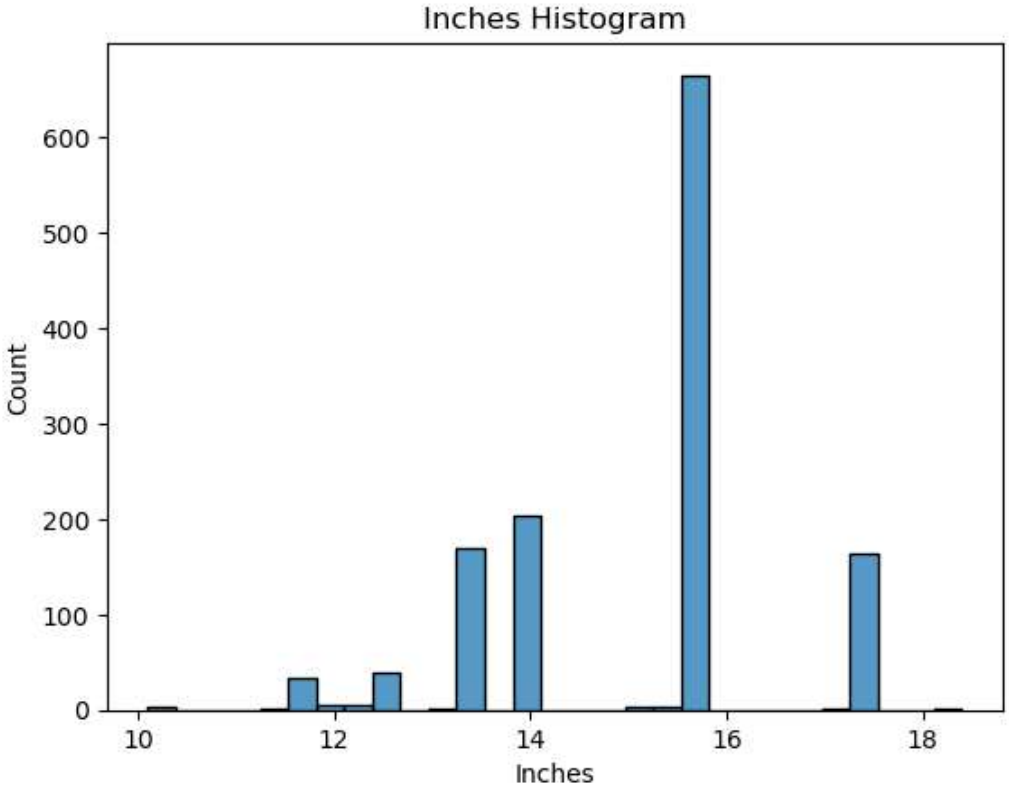
In [7]:
```
#Plots the distrubutions of all the variables in the data set.
#Then give each histogram a unique title
for i, col in enumerate(laptop_df):
    plt.figure(i)
    sns.histplot(x=col,data=laptop_df).set(title=col+' Histogram')
```

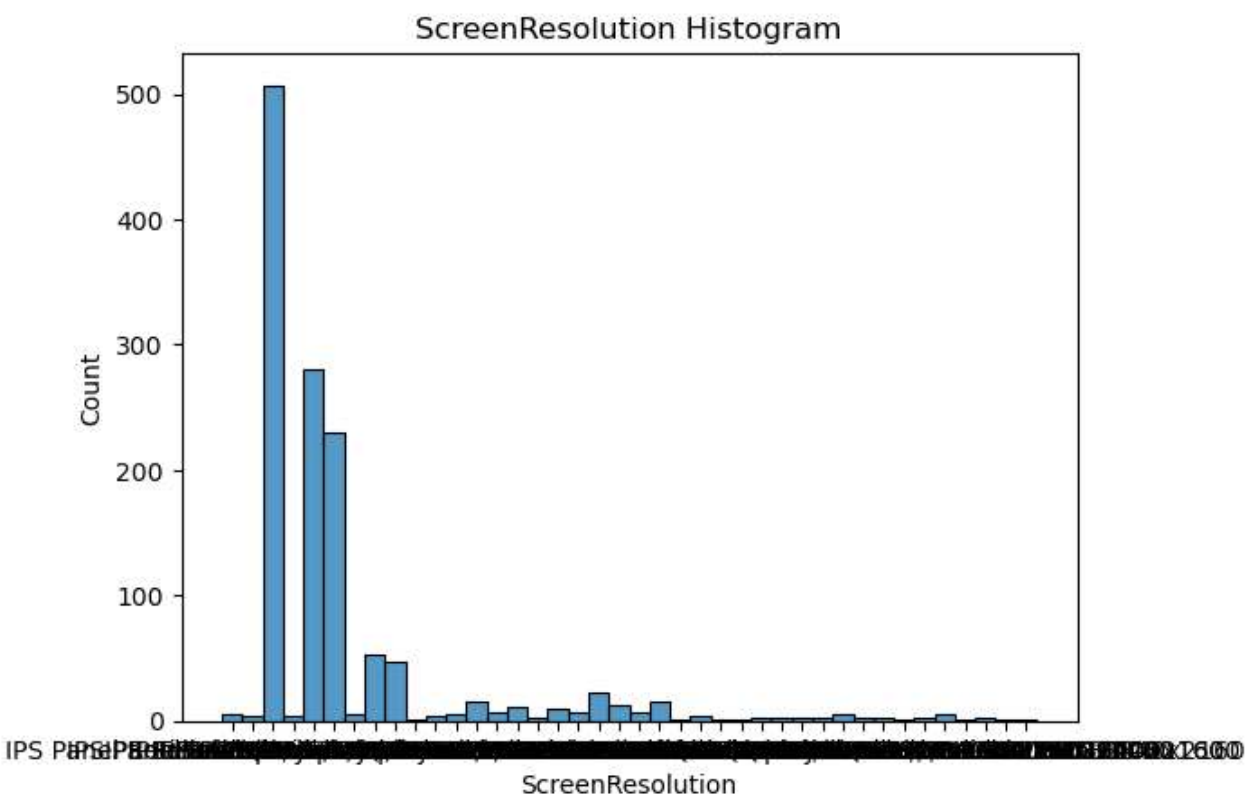


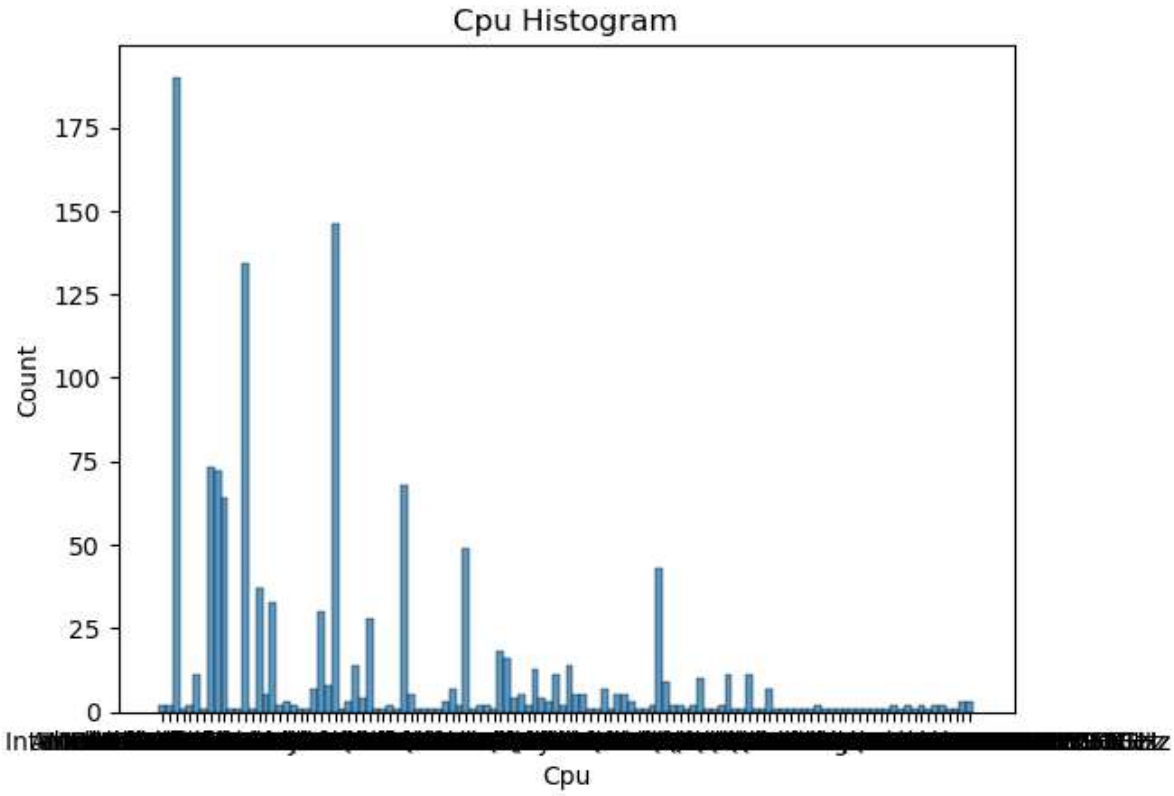

Company Histogram
Count
Company
0
50
100
150
200
250
300

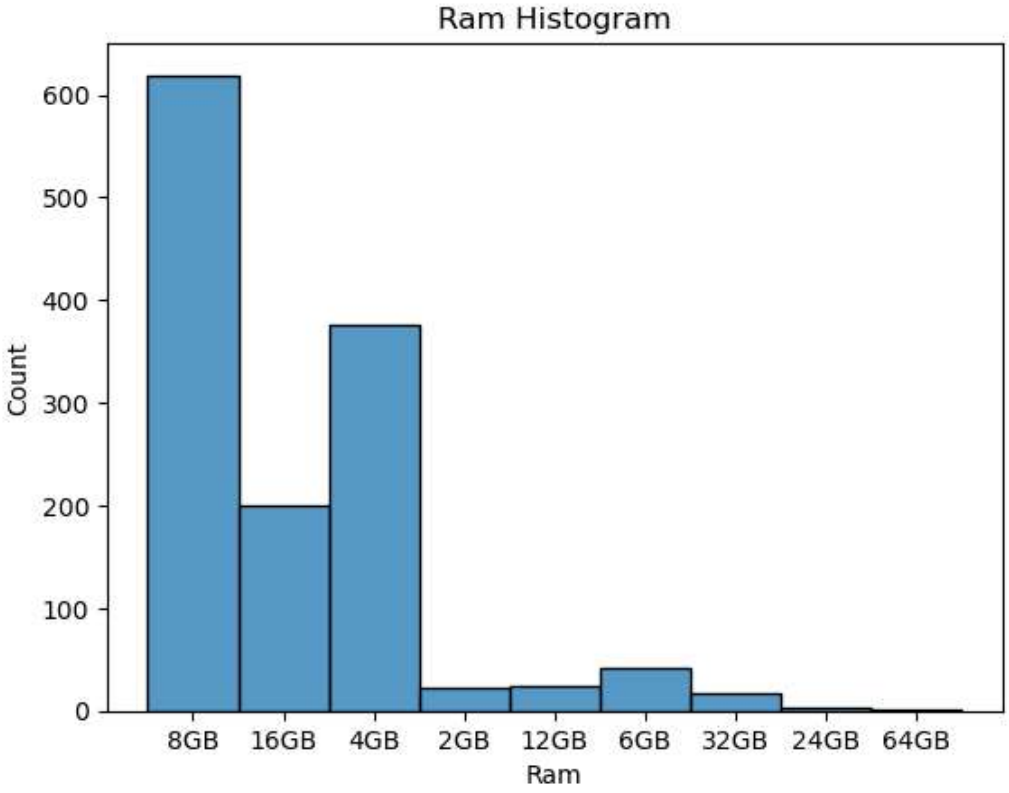

Product Histogram
Count
Product
0
5
10
15
20
25
30

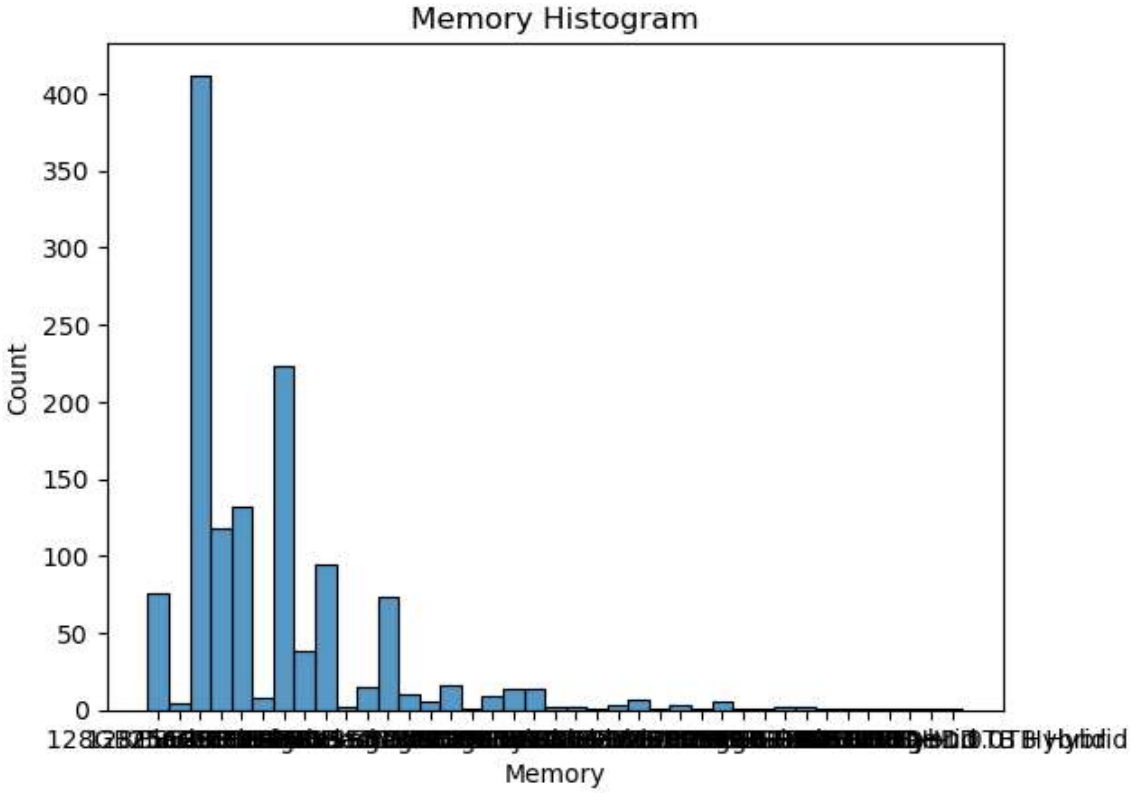

TypeName Histogram
Count
0
100
200
300
400
500
600
700
Ultrabook
Notebook
Netbook
Gaming
2 in 1 Convertible
Workstation
TypeName

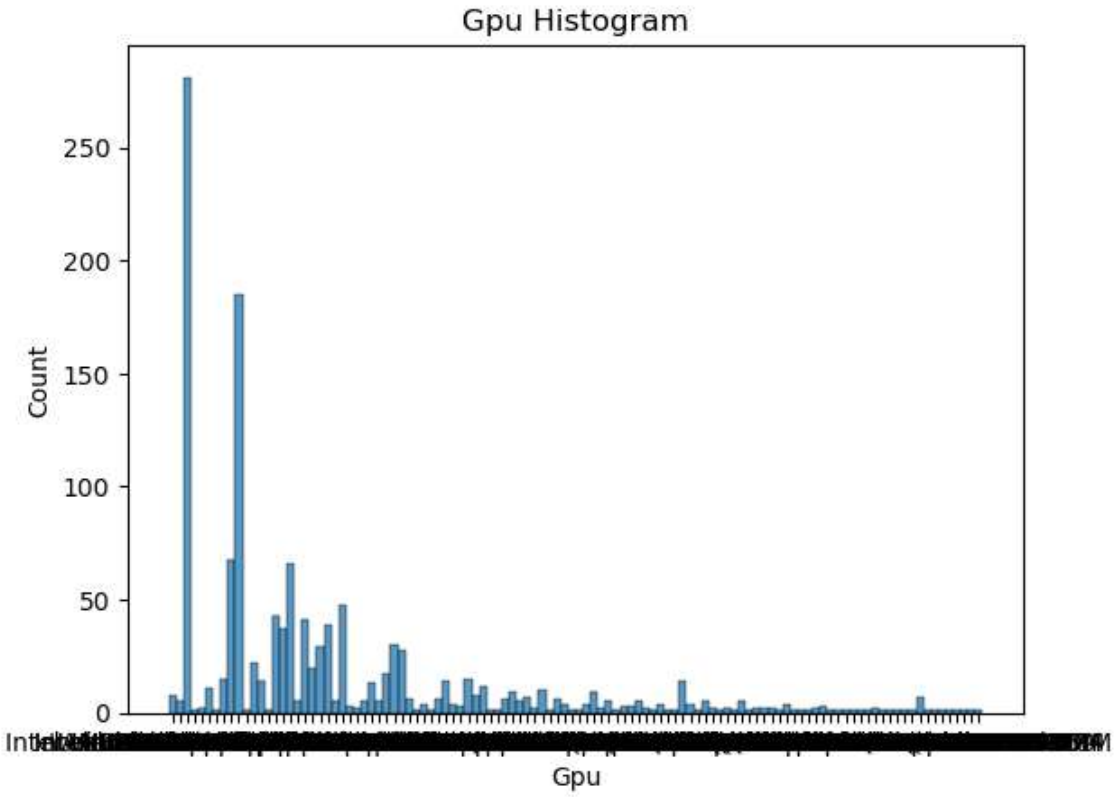

Inches Histogram
Count
0
100
200
300
400
500
600
10
12
14
16
18
Inches

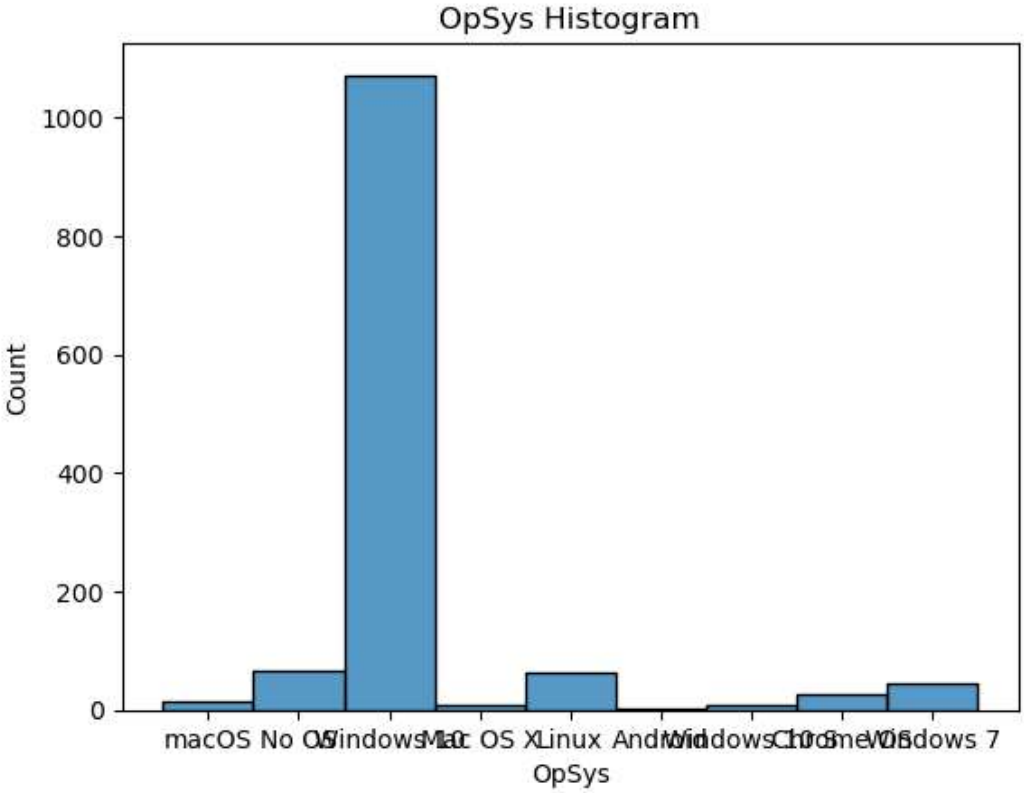

ScreenResolution Histogram
Count
500
400
300
200
100
0
ScreenResolution

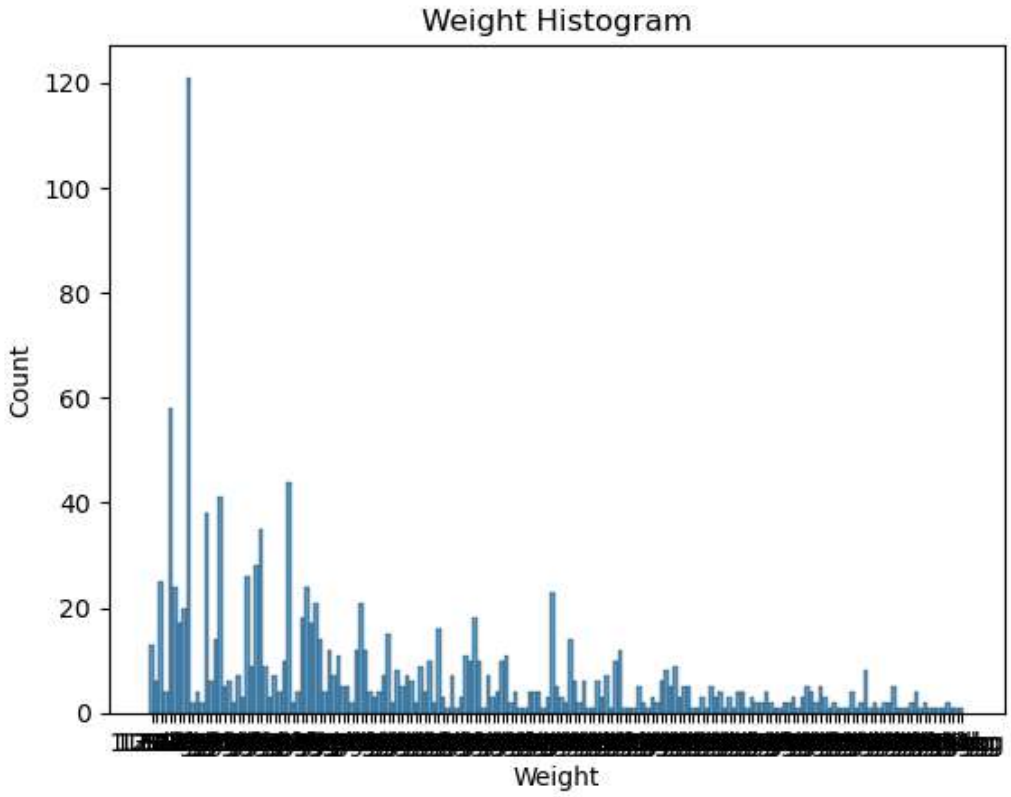

Cpu Histogram
Count
175
150
125
100
75
50
25
0
Cpu

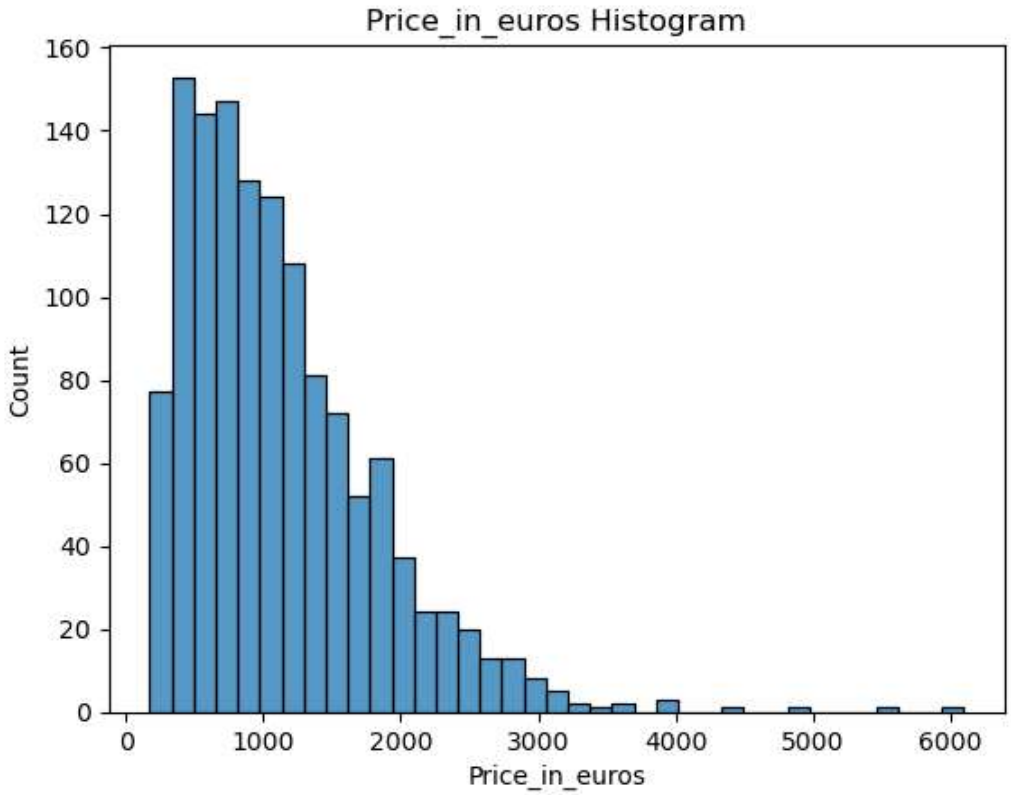

Ram Histogram
Count
0
100
200
300
400
500
600
8GB
16GB
4GB
2GB
12GB
6GB
32GB
24GB
64GB
Ram


Memory Histogram
Count
0
50
100
150
200
250
300
350
400
Memory


Gpu Histogram
Count
250
200
150
100
50
0
Gpu

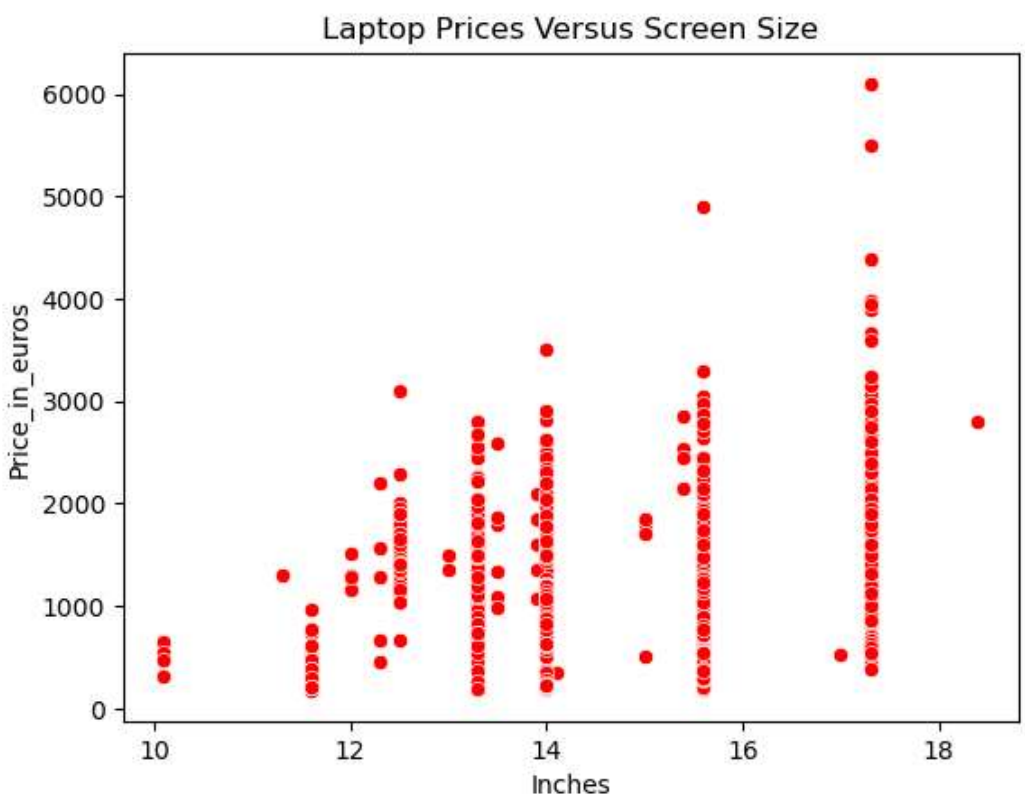

OpSys Histogram
Count
1000
800
600
400
200
0
macOS
Windows 10
Linux
Windows 7
OpSys





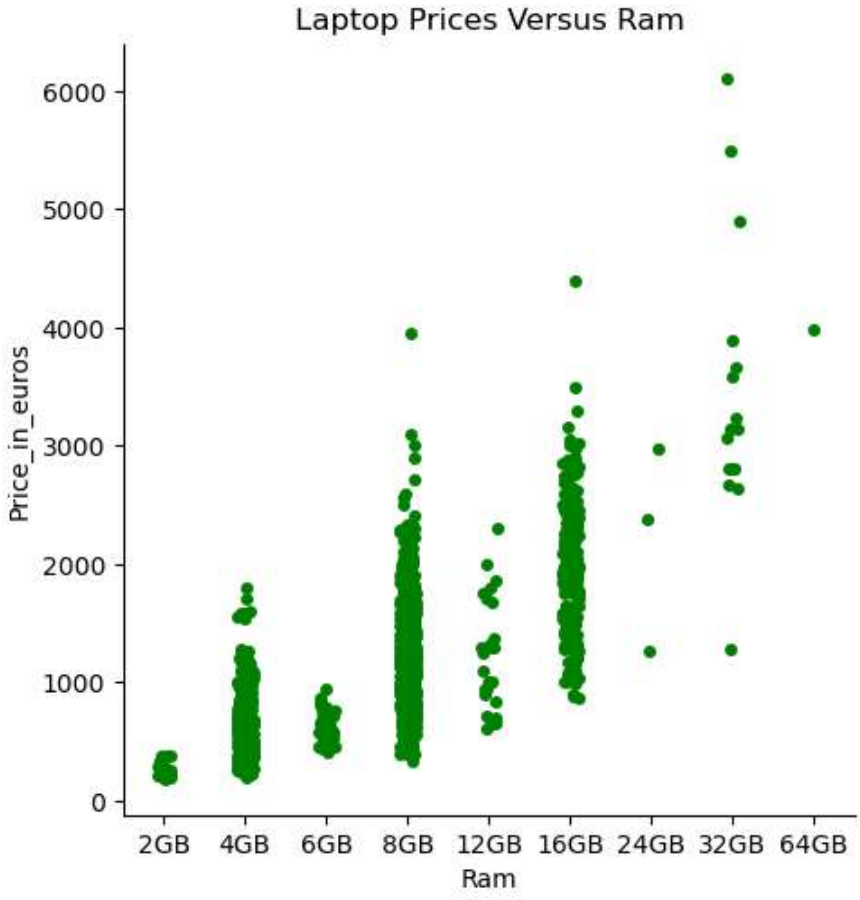
In [8]:
```
#creates a scatter plot of laptop price versus screen size.
sns.scatterplot(data=laptop_df, y='Price_in_euros', x='Inches',
                color='red').set(
    title='Laptop Prices Versus Screen Size')
```

Out[8]:
```
[Text(0.5, 1.0, 'Laptop Prices Versus Screen Size')]
```



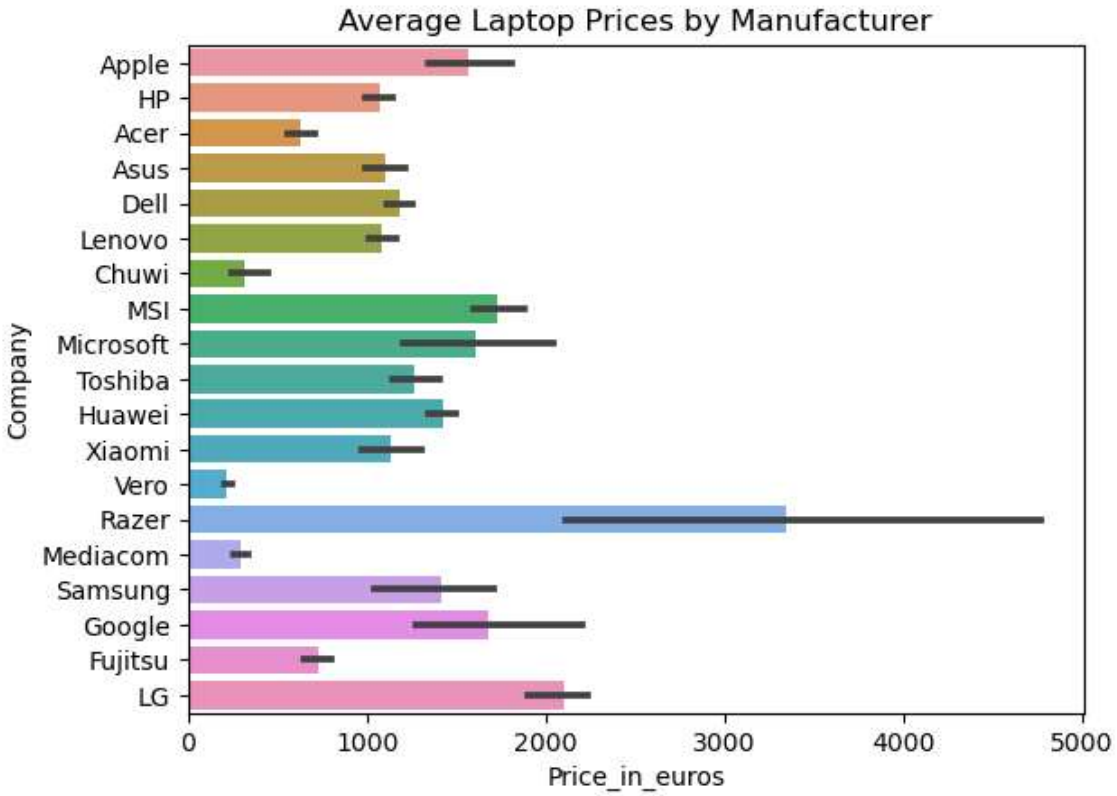
In [9]:
```
#creates a categorical scatter plot of laptop price versus Ram.
orl=['2GB','4GB','6GB','8GB','12GB','16GB','24GB','32GB','64GB']
sns.catplot(data=laptop_df, y='Price_in_euros', x='Ram',
            order=orl, color='green').set(
    title='Laptop Prices Versus Ram')
```

Out[9]:
```
<seaborn.axisgrid.FacetGrid at 0x255d1940be0>
```



In [10]:
```
#Creates a bar plot of average laptop price by manufacturer.
sns.barplot(data=laptop_df, x='Price_in_euros', y='Company').set(
    title='Average Laptop Prices by Manufacturer')
```

Out[10]: [Text(0.5, 1.0, 'Average Laptop Prices by Manufacturer')]



Conclusion

Based on the analysis and histograms, it is evident that significant cleanup is required to transform the dataset into a usable format for a predictive model. One crucial aspect of the cleanup process involves separating or splitting multiple features that are currently combined as strings into categorical and numeric features.

Analyzing the scatter plot for screen size reveals that there is an upward trend in the maximum possible price as the screen size increases. However, the relationship between screen size and price does not appear to be strictly linear. This suggests that factors other than screen size alone may influence the pricing of laptops.On the other hand, the scatter plot for price versus RAM size demonstrates a relatively linear increase in price, which aligns with expectations. As the RAM size increases, there is a corresponding rise in the price of laptops. This relationship suggests that RAM size has a more direct impact on pricing compared to screen size.

Furthermore, examining the average laptop price by manufacturer reveals significant price differences among different manufacturers. To gain a better understanding of the reasons behind these disparities, further exploration is required to determine whether the variations in price are primarily driven by performance differences or branding/marketing strategies employed by the manufacturers.

Data Preparation

In [11]:
```
#import needed packages
import re
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

In [12]:
```
#imports the data set again (easy of finding code when reopening)
laptop_df = pd.read_csv(
    "C:/Users/predi/Documents/GitHub/DSC550/Datasets/Term/laptop_price.csv",
    encoding='latin-1')
```

To standardize the data in the data frame, I performed two tasks. Firstly, removed any white spaces from the column headers and text strings. Secondly, converted all the text to lowercase. It ensures uniformity in the data, preventing any issues that may arise from inconsistent white spaces or capitalization during the encoding process.

In [13]:
```
#lowers and strips the column headers
laptop_df.columns=[x.lower() for x in laptop_df.columns]
laptop_df.columns=[x.strip() for x in laptop_df.columns]

#lowers and strips the strings in the data frame
for col in laptop_df:
    if laptop_df[col].dtype == 'object':
        laptop_df[col]=laptop_df[col].str.strip()
        laptop_df[col]=laptop_df[col].str.lower()
```

In [14]:
```
laptop_df.sample(5)
```

Out[14]:

	laptop_id	company	product	typename	inches	screenresolution	cpu	ram	memory	gpu	opsys	weight	p
1282	1300	hp	stream 11-y000na	netbook	11.6	1366x768	intel celeron dual core n3060 1.6ghz	2gb	32gb flash storage	intel hd graphics 400	windows 10	1.17kg	
125	128	hp	250 g6	notebook	15.6	1366x768	intel celeron dual core n3060 1.6ghz	4gb	500gb hdd	intel hd graphics 400	no os	1.86kg	
1086	1101	hp	zbook 15u	workstation	15.6	full hd 1920x1080	intel core i7 6500u 2.5ghz	8gb	256gb ssd	amd firepro w4190m	windows 7	1.9kg	
1090	1105	dell	inspiron 3552	notebook	15.6	1366x768	intel pentium quad core n3700 1.6ghz	4gb	500gb hdd	intel hd graphics	linux	2.2kg	
420	427	lenovo	yoga 720-15ikb	2 in 1 convertible	15.6	ips panel 4k ultra hd / touchscreen 3840x2160	intel core i7 7700hq 2.8ghz	16gb	512gb ssd	nvidia geforce gtx 1050	windows 10	2kg	

In the next step, the data frame underwent multiple transformations. The 'laptop_id' and 'product' columns were removed as they were irrelevant to the model. Next, the 'ram' and 'weight' columns had all text removed and were converted to numeric values. Weight measurements were converted from kilograms to pounds. Finally, the 'price' column was converted from euros to USD using a static conversion ratio, preventing any variations caused by daily fluctuations in currency values.

In [15]:
```
#drops ID column and product name
laptop_drop_df=laptop_df.drop(columns=['laptop_id','product'])

#changes ram column to numeric and removed GB text
laptop_drop_df['ram']=laptop_drop_df['ram'].str.replace('gb','')
laptop_drop_df['ram']=pd.to_numeric(laptop_drop_df['ram'])

#changes weight column to numeric, removes KG text, and converts to pounds
laptop_drop_df['weight']=laptop_drop_df['weight'].str.replace('kg','')
laptop_drop_df['weight']=pd.to_numeric(laptop_drop_df['weight'])
laptop_drop_df['weight']=laptop_drop_df['weight'].apply(lambda x:x*2.20462)

#coverts price column to USD and renames column
```

```
#used static conversion number to reduce day to day pricetions.
laptop_drop_df.rename(columns={'price_in_euros':'price'}, inplace=True)
laptop_drop_df['price']=laptop_drop_df['price'].apply(lambda x:x*1.1133)
```

In [16]:
```
laptop_drop_df.sample(5)
```

Out[16]:

	company	typename	inches	screenresolution	cpu	ram	memory	gpu	opsys	weight	price
314	asus	2 in 1 convertible	11.6	touchscreen 1366x768	intel celeron dual core n3350 1.1ghz	2	32gb flash storage	intel hd graphics 500	windows 10	2.425082	306.1575
815	lenovo	ultrabook	14.0	full hd 1920x1080	intel core i7 7500u 2.7ghz	8	256gb ssd	intel hd graphics 620	windows 10	2.910098	2069.6247
321	lenovo	notebook	17.3	1600x900	intel core i5 7200u 2.5ghz	4	1tb hdd	nvidia geforce 920mx	windows 10	6.150890	655.7337
288	lenovo	gaming	15.6	ips panel full hd 1920x1080	intel core i7 7700hq 2.8ghz	16	256gb ssd + 1tb hdd	nvidia geforce gtx 1050 ti	windows 10	5.511550	1312.5807
1144	hp	2 in 1 convertible	13.3	touchscreen 2560x1440	intel core i7 6600u 2.6ghz	8	256gb ssd	intel hd graphics 520	windows 10	3.262838	2002.8267

In the following step, the text was removed from the 'screen resolution' column, retaining only the pixel dimensions. This decision was made because the type of screens provided inconsistent information and often aligned with the pixel dimensions, making the text redundant for analysis. By extracting and keeping only the pixel dimensions, the data becomes more concise and focused.

In [17]:
```
#Removes all text from screenresolution field that isn't the pixel dimensions
laptop_drop_df[['screenresolution']]= laptop_drop_df[
    'screenresolution'].str.extract(r'(\d{4}x\d{4}|\d{4}x\d{3})')
```

The following code enhances the dataset by splitting the 'cpu' column into two new columns: 'cpu_brand' and 'cpu_speed.' This transformation provides more insightful information for the model. The newly introduced 'cpu_brand' column aids in assessing the impact of branding on CPU cost, while the 'cpu_speed' column provides a quantifiable measure for each CPU. Although the dataset lacks information on the number of cores, it is expected that the rated speed generally rises with an increase in cores. So this limitation is unlikely to have a significant impact on the model's performance.

In [18]:
```
#Breaks up cpu column brand and product
laptop_split_df = laptop_drop_df.copy()
laptop_split_df[['cpu_brand', 'cpu']] = laptop_split_df[
    "cpu"].str.split(" ", 1, expand=True)
#Extracts the speed from the cpu info, drops 'ghz', and changes dtype to numeric
laptop_split_df['cpu_speed'] = laptop_split_df[
    "cpu"].str.extract(r'(\d+\.\d+ghz|\d+ghz)')
laptop_split_df['cpu_speed']=laptop_split_df[
    'cpu_speed'].str.replace('ghz','')
laptop_split_df['cpu_speed']=pd.to_numeric(
    laptop_split_df['cpu_speed'])
#Drops original cpu column
laptop_split_df.drop(columns=['cpu'],inplace=True)
```

```
C:\Users\predi\AppData\Local\Temp\ipykernel_7512\3372058847.py:3: FutureWarning: In a future version of panda
s all arguments of StringMethods.split except for the argument 'pat' will be keyword-only.
  laptop_split_df[['cpu_brand', 'cpu']] = laptop_split_df[
```

In [19]:
```
#Breaks up gpu column brand and product
laptop_split1_df = laptop_split_df.copy()
laptop_split1_df[['gpu_brand', 'gpu']] = laptop_split1_df[
    "gpu"].str.split(" ", 1, expand=True)
```

```
#combines gpu's into various groups to reduce unique values
laptop_split1_df['gpu'] = laptop_split1_df['gpu'].apply(
    lambda x: 'hd graphics' if 'hd graphics' in x else x)
laptop_split1_df['gpu'] = laptop_split1_df['gpu'].apply(
    lambda x: 'gtx' if 'gtx' in x else x)
laptop_split1_df['gpu'] = laptop_split1_df['gpu'].apply(
    lambda x: 'iris' if 'iris' in x else x)
laptop_split1_df['gpu'] = laptop_split1_df['gpu'].apply(
    lambda x: 'radeon' if 'radeon' in x else x)
laptop_split1_df['gpu'] = laptop_split1_df['gpu'].apply(
    lambda x: 'quadro' if 'quadro' in x else x)
laptop_split1_df['gpu'] = laptop_split1_df['gpu'].apply(
    lambda x: 'geforce' if 'geforce' in x else x)

other_list=['firepro','graphics','mali','r17m']
laptop_split1_df['gpu'] = laptop_split1_df['gpu'].apply(
    lambda x:'other' if any(item in x for item in other_list) else x)
```

```
C:\Users\predi\AppData\Local\Temp\ipykernel_7512\3095492546.py:3: FutureWarning: In a future version of panda
s all arguments of StringMethods.split except for the argument 'pat' will be keyword-only.
  laptop_split1_df[['gpu_brand', 'gpu']] = laptop_split1_df[
```

In [20]:
```
laptop_split1_df.sample(5)
```

Out[20]:

	company	typename	inches	screenresolution	ram	memory	gpu	opsys	weight	price	cpu_brand	cpu_spe
1108	hp	notebook	15.6	1920x1080	4	1tb hdd	radeon	windows 10	4.629702	443.093400	amd	
428	hp	gaming	17.3	1920x1080	12	256gb ssd + 1tb hdd	gtx	windows 10	7.385477	2225.486700	intel	
966	dell	notebook	15.6	1366x768	4	500gb hdd	other	windows 10	4.188778	918.472500	intel	
38	hp	notebook	15.6	1366x768	4	1tb hdd	other	windows 10	4.100593	544.058577	intel	
378	asus	notebook	14.0	1366x768	4	32gb flash storage	other	windows 10	3.306930	318.403800	intel	

The final data cleaning step involved splitting the memory column into three separate columns: '2nd_hd', 'memory', and 'memory_type'. This was done to ensure its suitability for a regression model. The decision to create a boolean column for the second hard drive ('2nd_hd') was based on the fact that the majority of second hard drives had the same size and type. As a result, this approach helped reduce the need for dummy variables. The other two columns, 'memory' (representing size) and 'memory_type', were also beneficial for the model. By representing size as a numeric value and separating the memory type, the data became more useful for analysis.

In [21]:
```
#Creates second harddrive boolean column
laptop_split2_df = laptop_split1_df.copy()
laptop_split2_df['2nd_hd']= laptop_split2_df['memory'].str.contains(r'\+')
#splits the remaining information into size and type of primary harddrive column.
laptop_split2_df['memory']=laptop_split2_df['memory'].str.split(' \+',n=1,).str[0]
laptop_split2_df[['memory','memory_type']]=laptop_split2_df[
    'memory'].str.split(' ',n=1,expand=True)
laptop_split2_df['memory']=laptop_split2_df['memory'].str.replace('gb|tb','')
laptop_split2_df['memory']=pd.to_numeric(laptop_split2_df['memory'])
```

```
C:\Users\predi\AppData\Local\Temp\ipykernel_7512\4024067187.py:8: FutureWarning: The default value of regex w
ill change from True to False in a future version.
  laptop_split2_df['memory']=laptop_split2_df['memory'].str.replace('gb|tb','')
```

In [22]:
```
laptop_split2_df.sample(5)
```

Out[22]:

	company	typename	inches	screenresolution	ram	memory	gpu	opsys	weight	price	cpu_brand	cpu_speed
509	lenovo	gaming	15.6	1920x1080	8	256.0	gtx	no os	5.291088	1279.18170	intel	2.8
362	lenovo	notebook	17.3	1600x900	6	128.0	gtx	windows 10	6.172936	800.46270	intel	2.5
1006	hp	notebook	14.0	1920x1080	4	256.0	other	windows 10	4.299009	1341.52650	intel	2.5
288	lenovo	gaming	15.6	1920x1080	16	256.0	gtx	windows 10	5.511550	1312.58070	intel	2.8
974	asus	gaming	17.3	1920x1080	16	256.0	gtx	windows 10	6.018613	2282.15367	intel	2.8

The following code performs data preprocessing on the dataset. It splits the cleaned dataset into training and test sets, allocating 80% for training and 20% for testing. Next, the categorical features in the training and test datasets are one-hot encoded. Once the encoding is complete, the datasets are reindexed and merged with the numerical features. Now, the datasets are fully prepared and can be directly used in a model.

In [23]:
```
#breaks up data set into training and test sets
x=laptop_split2_df.drop(columns=['price'])
y=laptop_split2_df['price']
x_train,x_valid,y_train,y_valid=train_test_split(x,y, test_size=0.2, random_state=42)
```

In [24]:
```
#creates a list of the columns with a string data type
s=(x_train.dtypes=='object')
object_cols = list(s[s].index)
print(object_cols)
```

```
['company', 'typename', 'screenresolution', 'gpu', 'opsys', 'cpu_brand', 'gpu_brand', 'memory_type']
```

In [25]:
```
# Apply one-hot encoder to each column with categorical data
ohe=OneHotEncoder(handle_unknown='ignore',sparse_output=False)
ohe_train=pd.DataFrame(ohe.fit_transform(x_train[object_cols]))
ohe_valid=pd.DataFrame(ohe.transform(x_valid[object_cols]))
# Reindexing the dataframes
ohe_train.index=x_train.index
ohe_valid.index=x_valid.index
# Removes categorical columns
num_x_train=x_train.drop(columns=object_cols)
num_x_valid=x_valid.drop(columns=object_cols)
#combine numerical and ohe catgorical fetaures back
oh_x_train=pd.concat([num_x_train,ohe_train], axis=1)
oh_x_valid=pd.concat([num_x_valid,ohe_valid], axis=1)
# Ensure all columns have string type
oh_x_train.columns = oh_x_train.columns.astype(str)
oh_x_valid.columns = oh_x_valid.columns.astype(str)
```

In [26]:
```
oh_x_train.sample(10)
```

Out[26]:

	inches	ram	memory	weight	cpu_speed	2nd_hd	0	1	2	3	...	56	57	58	59	60	61	62	63	64	65
990	12.5	8	512.0	2.138481	1.2	False	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
733	15.6	4	500.0	5.291088	2.5	False	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
981	13.3	4	128.0	2.645544	2.3	False	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
728	15.6	8	1.0	5.114718	2.5	False	0.0	0.0	0.0	0.0	...	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1036	15.6	4	500.0	4.497425	2.5	False	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
269	15.6	8	256.0	4.519471	1.8	False	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
461	11.6	4	128.0	3.086468	1.6	False	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
242	17.3	8	128.0	5.930428	2.7	True	0.0	0.0	1.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
491	13.3	32	512.0	2.314851	2.7	False	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
706	13.3	8	256.0	3.527392	1.6	False	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0

10 rows × 72 columns

Model

In [27]:
```
#imports all needed packages
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from xgboost import XGBRegressor
import catboost as cb
import lightgbm as lgb
from sklearn.model_selection import  GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

I chose to test XGBoost, CatBoost, and LightGBM over other models due to their strong performance in various machine learning tasks and data science competitions. These models are well-known for their ability to handle large datasets efficiently, capture complex non-linear relationships, and provide built-in feature importance metrics.The decision to select these models was also influenced by their gradient boosting framework, which combines multiple weak learners to create a powerful predictive model. This enables the models to deliver superior predictive performance compared to individual decision trees.

I am also using a grid search to test multiple hyperparemeters for each type of model. The three hyperparameters that will be tested in the grid search are; n_estimators, max_depth, and learning_rate. All three can play a major role in optimizing a model. There will be a maximum of four different variables for each hyperparameter to reduce computational time required. The grid searches are broken up by model, instead of a single pipeline to reduce indivdual computational time. This was decided to allow changes to be made quicker.

In [28]:
```
# Define hyperparameters
xgb_params = {'regressor__n_estimators': [250, 500, 1000],
    'regressor__max_depth': [3, 5, 7, 10],
    'regressor__learning_rate': [0.01, 0.1, 0.2]}

#Create pipeline
xgb_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', XGBRegressor())])

# Grid search
xgb_grid_search = GridSearchCV(xgb_pipe, param_grid=xgb_params, cv=5)
xgb_grid_search.fit(oh_x_train, y_train)

#Score and best parameters for model
```

```
print("XGBRegressor - Best Score:", xgb_grid_search.best_score_)
print("XGBRegressor - Best Parameters:", xgb_grid_search.best_params_)
```

```
XGBRegressor - Best Score: 0.8444697108696237
XGBRegressor - Best Parameters: {'regressor__learning_rate': 0.1, 'regressor__max_depth': 3, 'regressor__n_es
timators': 1000}
```

In [29]:
```
#Define hyperparameters
cat_params = {'regressor__n_estimators': [250, 500, 1000],
    'regressor__depth': [3, 5, 7, 10],
    'regressor__learning_rate': [0.01, 0.1, 0.2]}

#Create pipeline
cat_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', cb.CatBoostRegressor(logging_level='Silent',
                                       loss_function='RMSE'))])
# Grid search
cat_grid_search = GridSearchCV(cat_pipe, param_grid=cat_params, cv=5)
cat_grid_search.fit(oh_x_train, y_train)

#Score and best parameters for model
print("CatBoostRegressor - Best Score:", cat_grid_search.best_score_)
print("CatBoostRegressor - Best Parameters:", cat_grid_search.best_params_)
```

```
CatBoostRegressor - Best Score: 0.8529994400234961
CatBoostRegressor - Best Parameters: {'regressor__depth': 3, 'regressor__learning_rate': 0.1, 'regressor__n_e
stimators': 1000}
```

In [30]:
```
# Define the hyperparameters for CatBoostRegressor
lgbm_params = {'regressor__n_estimators': [250, 500, 1000],
    'regressor__depth': [3, 5, 7, 10],
    'regressor__learning_rate': [0.01, 0.1, 0.2]}

#Create pipeline
lgbm_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', lgb.LGBMRegressor( verbose=-1))])

# Grid search
lgbm_grid_search = GridSearchCV(lgbm_pipe, param_grid=lgbm_params, cv=5)
lgbm_grid_search.fit(oh_x_train, y_train)

#Score and best parameters for model
print("LGBMRegressor - Best Score:", lgbm_grid_search.best_score_)
print("LGBMRegressor - Best Parameters:", lgbm_grid_search.best_params_)
```

```
LGBMRegressor - Best Score: 0.8027993992780423
LGBMRegressor - Best Parameters: {'regressor__depth': 3, 'regressor__learning_rate': 0.1, 'regressor__n_estim
ators': 250}
```

In [31]:
```
#Get the best model/hyperparameters from each type of regession.
xgb_model = xgb_grid_search.best_estimator_
cat_model = cat_grid_search.best_estimator_
lgbm_model = lgbm_grid_search.best_estimator_
```

In [32]:
```
#Predicts price for each model type for training and test data
#XGB model
xgb_pred_train=xgb_model.predict(oh_x_train)
xgb_pred_test=xgb_model.predict(oh_x_valid)

#cat model
cat_pred_train=cat_model.predict(oh_x_train)
cat_pred_test=cat_model.predict(oh_x_valid)

#LGBM model
lgbm_pred_train=lgbm_model.predict(oh_x_train)
lgbm_pred_test=lgbm_model.predict(oh_x_valid)
```

The three evaluation metrics, MAE, RMSE, and R-squared, were chosen because they collectively provide a comprehensive assessment of the regression model's performance. MAE was selected due to its ability to measure the average absolute

deviation of predictions from the actual values, making it robust to outliers and easily interpretable. RMSE was chosen as it gives more weight to larger errors, providing a balanced view of the model's precision while also being sensitive to outliers. R-squared was included to evaluate the goodness of fit, indicating how much of the variance in the target variable the model explains. Combining these metrics offers a well-rounded evaluation of the model's effectiveness.

Both the training and testing data sets will be evaluated with these metrics. This will help with better understanding any overfitting in the various models

In [33]:
```
print('-'*25,'XGB Training Data','-'*25)
print('MAE:', mean_absolute_error(xgb_pred_train,y_train))
print('RMSE:', mean_squared_error(xgb_pred_train,y_train, squared=False))
print('R_squared:', r2_score(xgb_pred_train,y_train))

print('-'*25,'XGB Testing Data','-'*25)
print('MAE:', mean_absolute_error(xgb_pred_test,y_valid))
print('RMSE:', mean_squared_error(xgb_pred_test,y_valid, squared=False))
print('R_squared:', r2_score(xgb_pred_test,y_valid))
```

```
------------------------- XGB Training Data -------------------------
MAE: 94.90927336185108
RMSE: 135.99701006228236
R_squared: 0.9671519470618974
------------------------- XGB Testing Data -------------------------
MAE: 186.49238429118773
RMSE: 306.4036297484758
R_squared: 0.8520250622093166
```

In [34]:
```
print('-'*25,'Cat Training Data','-'*25)
print('MAE:', mean_absolute_error(cat_pred_train,y_train))
print('RMSE:', mean_squared_error(cat_pred_train,y_train, squared=False))
print('R_squared:', r2_score(cat_pred_train,y_train))

print('-'*25,'Cat Testing Data','-'*25)
print('MAE:', mean_absolute_error(cat_pred_test,y_valid))
print('RMSE:', mean_squared_error(cat_pred_test,y_valid, squared=False))
print('R_squared:', r2_score(cat_pred_test,y_valid))
```

```
------------------------- Cat Training Data -------------------------
MAE: 127.72344736280094
RMSE: 177.270299296372
R_squared: 0.9421654464184152
------------------------- Cat Testing Data -------------------------
MAE: 191.23976185322854
RMSE: 314.00491652610435
R_squared: 0.8326448148988104
```

In [35]:
```
print('-'*25,'LGBM Training Data','-'*25)
print('MAE:', mean_absolute_error(lgbm_pred_train,y_train))
print('RMSE:', mean_squared_error(lgbm_pred_train,y_train, squared=False))
print('R_squared:', r2_score(lgbm_pred_train,y_train))

print('-'*25,'LGBM Testing Data','-'*25)
print('MAE:', mean_absolute_error(lgbm_pred_test,y_valid))
print('RMSE:', mean_squared_error(lgbm_pred_test,y_valid, squared=False))
print('R_squared:', r2_score(lgbm_pred_test,y_valid))
```

```
------------------------- LGBM Training Data -------------------------
MAE: 93.5304615977107
RMSE: 148.3851342899567
R_squared: 0.9600971177648788
------------------------- LGBM Testing Data -------------------------
MAE: 213.6816301682625
RMSE: 348.2602005305565
R_squared: 0.7961830036330262
```

The XGBoost model performed the best with both the training and testing data in predicting laptop prices. However, there is a significant amount of overfitting, as evidenced by the relatively large gap between the evaluation metrics of the training and testing datasets. The mean absolute error (MAE) suggests that, on average, the model's predictions deviate by

about 15% from the actual laptop prices, given that the mean price is $1251. The calculated R-squared value of 0.852 indicates that approximately 85.2% of the variance in the predicted prices can be explained by the model.

The main reason for the variance between the predicted and actual prices is the lack of information on quality of life features for the laptops, such as laptop build materials, backlit keyboards, cameras, and build quality. These features can significantly impact the laptop's price, but unfortunately, they are not available in the dataset. Another issue contributing to the price variance is the method of obtaining the prices. All the prices were webscraped in one go, which could lead to fluctuations in prices due to some models being on sale while others were not at that specific time. To address this problem and improve the understanding of laptop price fluctuations, prices should be tracked over multiple points throughout the years.