Laptop Price Predictions

Oleh:

Dhoni Hanif Supriyadi

1. Business Understanding

At this stage, a Data Scientist will try to understand the business first. Here, I want to make a price prediction for laptops. As we know, laptop prices will change every year and laptops also have different prices between one type and another. By knowing that this data is continuous data, what I will use is the regression method. Here, I will make a laptop price prediction with an emphasis on Exploratory Data Analysis (EDA) to get insight from the existing data so I can conclude what this data looks like.

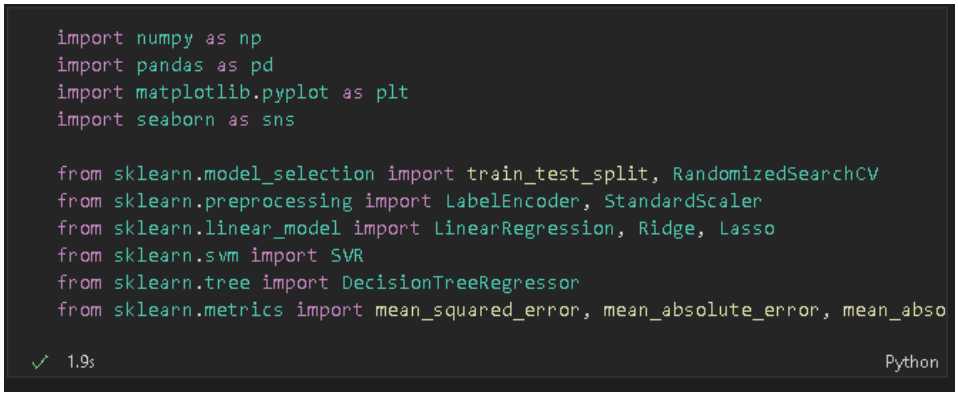
2. Data Collection

At this stage, a Data Scientist will collect the data first. Here, I have collected laptop data with id, name, type, product, ram, weight, screen size, and others sourced from kaggle.com with a total of 1303 data and 13 variables.

3. Data Preparation

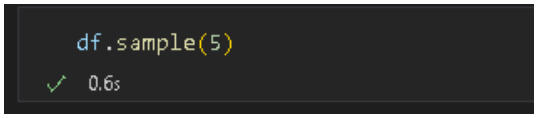
At this stage, a Data Scientist will read, check, and clean the data first. If there is a null value, a Data Scientist will fill it in with the mean, median, and mode values and can even delete data that contains the null value. Here, I use python and jupyter notebook.

The first thing I did was import the libraries needed as shown below.



Next, I did a read on the csv data. Seeing that this data is in csv form, I use the pandas.read_csv method as shown below.

Then, I checked 5 sample data randomly with pandas



	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price_euros
985	999	Dell	Inspiron 3567	Notebook	15.6	1366x768	Intel Core i5 7200U 2.5GHz	4GB	500GB HDD	AMD Radeon R5 M430	Windows 10	2.25kg	599.0
661	669	Lenovo	Ideapad 320-15IAP	Notebook	15.6	1366x768	Intel Celeron Dual Core N3350 1.1GHz	4GB	500GB HDD	Intel HD Graphics 500	Windows 10	2.2kg	419.0
283	288	Lenovo	Ideapad 320-15IKBN	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	6GB	256GB SSD	Intel HD Graphics 620	Windows 10	2.2kg	579.0
368	374	Dell	Inspiron 5567	Notebook	15.6	Full HD 1920x1080	Intel Core i7 7500U 2.7GHz	8GB	256GB SSD	AMD Radeon R7 M445	Windows 10	2.33kg	899.0
210	215	Acer	Aspire 7	Notebook	15.6	Full HD 1920x1080	Intel Core i7 7700HQ 2.8GHz	8GB	1TB HDD	Nvidia GeForce GTX 1050	Linux	2.4kg	779.0

Next, I checked whether there was a missing value or not with the isnull() method from the pandas library as follows.

```
df.isnull().sum()
```
✓ 0.6s

```
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
```

As we can see, this data holds no null or missing values. Next we check whether there is duplicate data or not with pandas.

```
df.duplicated().sum()
```
✓ 0.6s

```
0
```

As we can see, data doesn't store duplicate values either. In this way, we can continue the process by checking the data type, mean, median, standard deviation and others.

We check the information from the data first as follows.

```
df.info()
```
✓ 0.7s

```
<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
```

Here, the data does not contain null values with a total of 1303 data and data types, there are several data types such as object, integer, and float with memory usage of 132.5+ KB.

Next, we check the amount of data, unique value, mode and frequency of each variable of type object to get that information. We do as follows.

```
df.describe(exclude="number").T
```
✓ 0.8s

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

As we can see, some data has a unique value that is not that big so we can say that the variable does not store data with high variation and some data also says the opposite, that data stores data with high variation such as product, cpu, gpu data and weights.Melihat data Weight yang bervariasi dan menyimpan data angka, maka kita dapat mengubahnya menjadi data numerik. Sebelum itu, kita harus menghilangkan kata “kg” dalam data tersebut seperti berikut.

```
for i in range(len(df["Weight"])):
    df.loc[i, "Weight"] = df.loc[i, "Weight"][:-2]
df.loc[:, "Weight"] = df.loc[:, "Weight"].astype("float")
```
✓ 0.3s

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_8144\4162820238.py:3: FutureW
set the values inplace instead of always setting a new array. To retain the old b
non-unique, `df.isetitem(i, newvals)`
  df.loc[:, "Weight"] = df.loc[:, "Weight"].astype("float")
```

After we remove the word "kg" in the data, then we change the data type to float. Next, we check the mean, median, standard deviation, quartiles, min, and max values of some numeric type data as follows.

```
df.describe().T
```
✓ 0.6s

	count	mean	std	min	25%	50%	75%	max
laptop_ID	1303.0	660.155794	381.172104	1.00	331.5	659.00	990.50	1320.0
Inches	1303.0	15.017191	1.426304	10.10	14.0	15.60	15.60	18.4
Weight	1303.0	2.038734	0.665475	0.69	1.5	2.04	2.30	4.7
Price_euros	1303.0	1123.686992	699.009043	174.00	599.0	977.00	1487.88	6099.0

As we can see, the 2 data sets have very high standard deviations compared to the other 2 data sets. These 2 data also store a max value that is so large compared to the other 2 data. Seeing that the laptop_ID data has such a large standard deviation, it can be concluded that this data varies greatly. Maybe we'll see in the analysis stage.

Then, we try to print describe of object variable like this.

```
df.describe(exclude="number").T
```
✓ 0.7s

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

We can say that:

- Product have high unique value and TypeName have low unique value
- So much users use windows 10 with frequency 1072

- The most product that people use is XPS 13
- We can use TypeName as the type of product

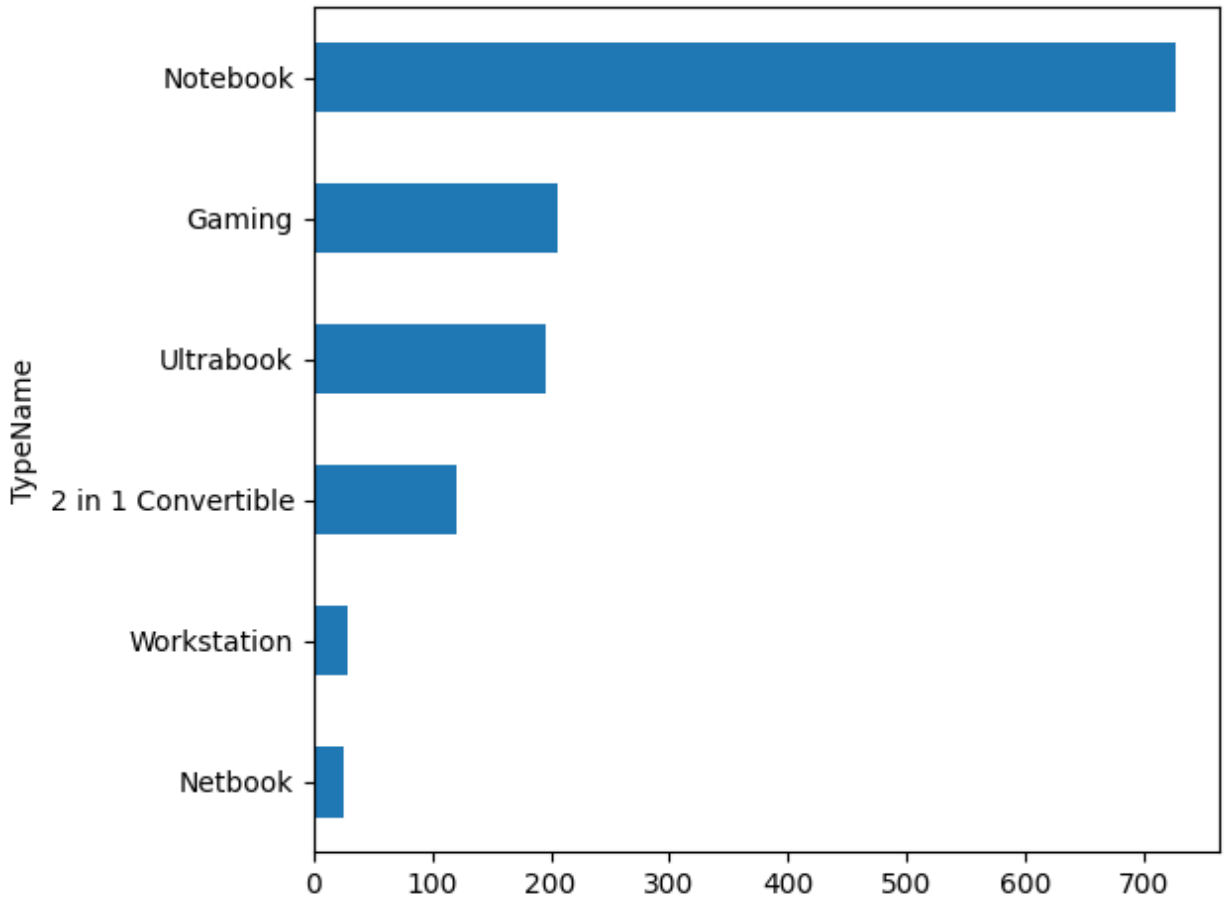
4. Exploratory Data Analysis (EDA)

First, lets make group data based on TypeName like this.

```
model_no_grp = df.groupby("TypeName", axis=0)
```
✓ 0.5s

Then, lets plotting first the size or how much data of data TypeName.

```
model_no_grp.size().sort_values().plot.barh()
plt.tight_layout()
plt.show()
```
✓ 0.2s

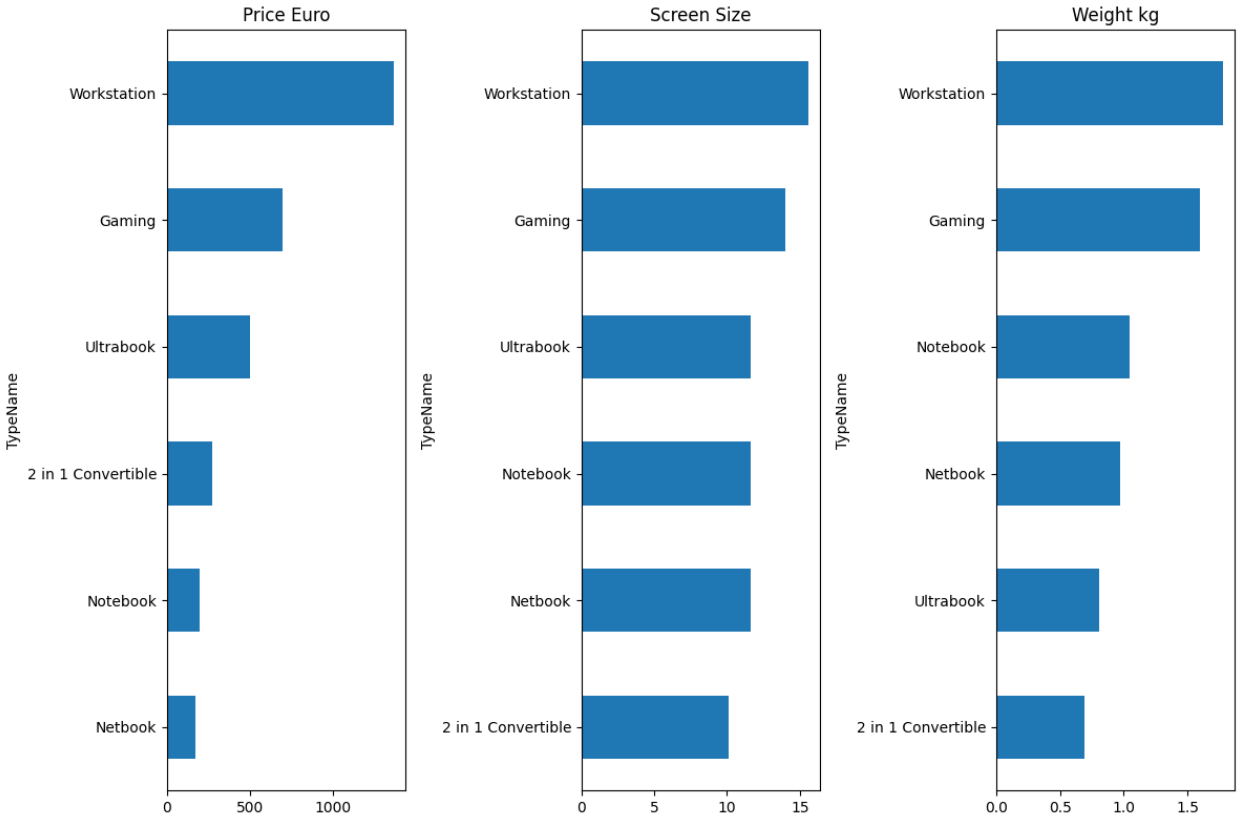


We can say that so much users use Notebook in this data and just little users who use Netbook. People who use Gaming type is the second highest data of data TypeName.

Then, lets visualize the minimum price euros, screen size, and weight of data based on TypeName.

```
fig, ax = plt.subplots(1, 3, figsize=(12, 8))
model_no_grp.min()["Price_euros"].sort_values().plot.barh(ax=ax[0])
model_no_grp.min()["Inches"].sort_values().plot.barh(ax=ax[1])
model_no_grp.min()["Weight"].sort_values().plot.barh(ax=ax[2])
ax[0].set_title("Price Euro")
ax[1].set_title("Screen Size")
ax[2].set_title("Weight kg")
plt.tight_layout()
plt.show()
```

✓ 0.5s



We can say that:

- The minimum price, screen size, and weight of Workstation is higher than the other

- The minimum price of Netbook is lower than the other but the minimum screen size of Netbook is higher than 2 in 1 Convertible and the minimum weight of Netbook is higher than Ultrabook and 2 in 1 Convertible

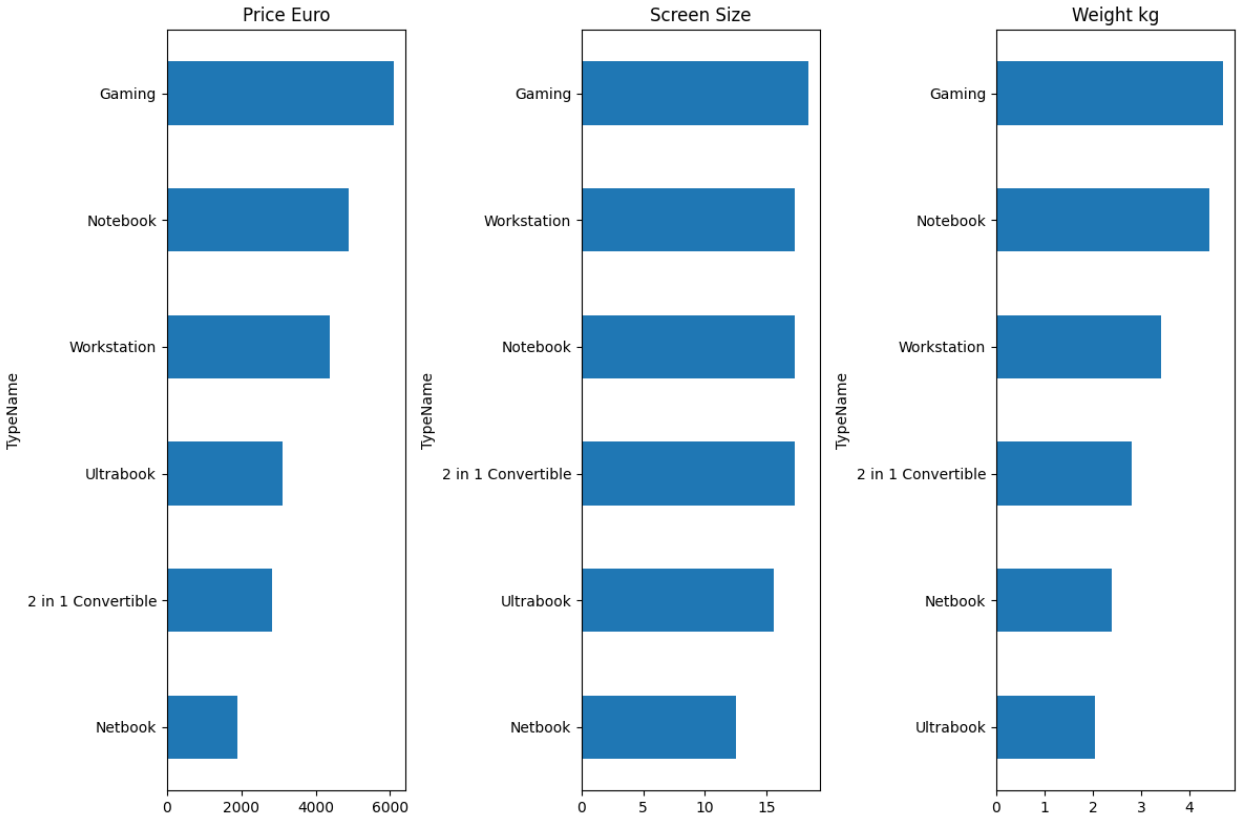
- 2 in Convertible is the type that has higher minimum price than Notebook and Netbook but this type is the lowest minimum screen size and weight

After that, lets visualize the maximum of price euros, screen size, and weight of data based on TypeName.

```
fig, ax = plt.subplots(1, 3, figsize=(12, 8))
model_no_grp.max()["Price_euros"].sort_values().plot.barh(ax=ax[0])
model_no_grp.max()["Inches"].sort_values().plot.barh(ax=ax[1])
model_no_grp.max()["Weight"].sort_values().plot.barh(ax=ax[2])

ax[0].set_title("Price Euro")
ax[1].set_title("Screen Size")
ax[2].set_title("Weight kg")
plt.tight_layout()
plt.show()
```

✓ 0.5s



From this, we can say that :

- The maximum of price, screen size, and weight is Gaming type

- Workstation is the highest minimum price, screen size, and weight than the other but has lower maximum price, screen size, and weight than Gaming

- Netbook is the lowest maximum price and screen size but this type has higher maximum weight than Ultrabook

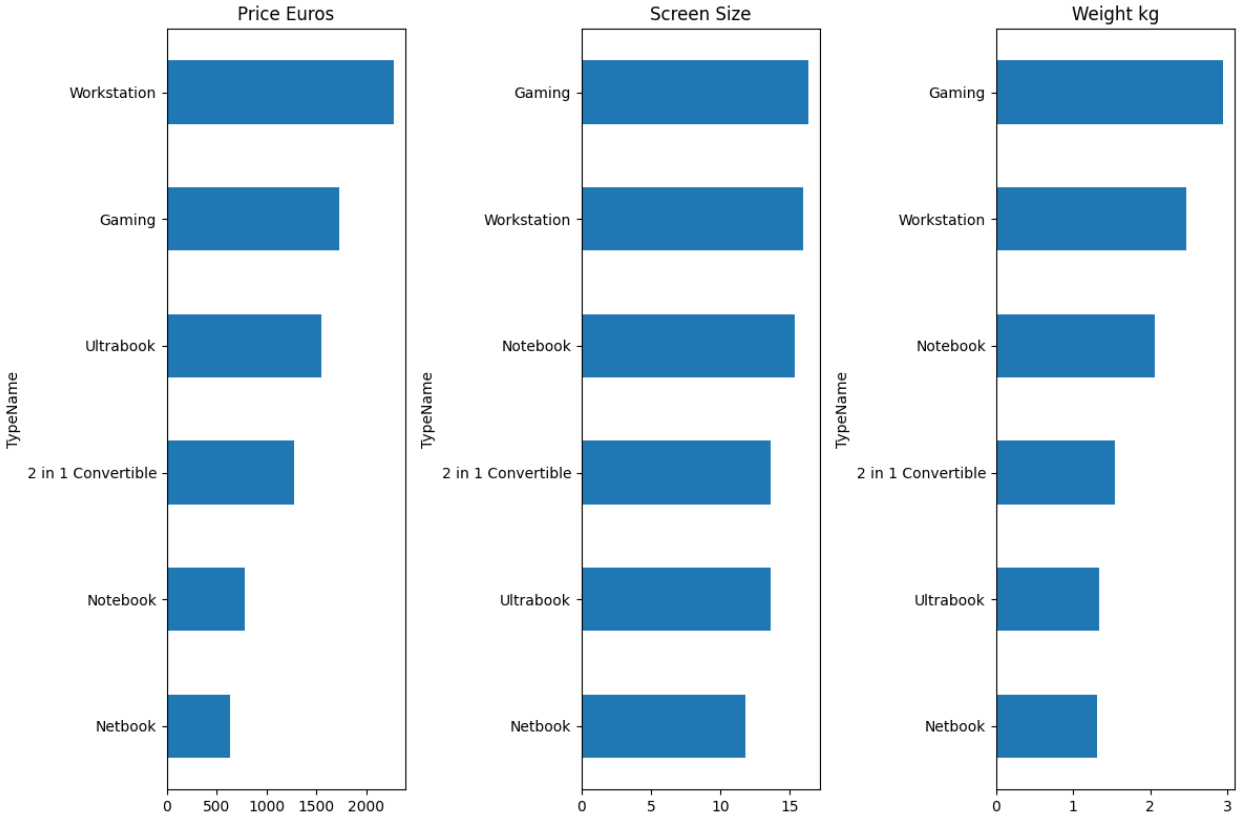
- Ultrabook has higher maximum price than 2 in 1 Convertible and Netbook, but this type has lower screen size than 2 in 1 Convertible and higher than Netbook and this type is the lowest weight

Then, lets visualize the mean of price euros, screen size, and weight of data based on TypeName.

```python
fig, ax = plt.subplots(1, 3, figsize=(12, 8))
model_no_grp.mean()["Price_euros"].sort_values().plot.barh(ax=ax[0])
model_no_grp.mean()["Inches"].sort_values().plot.barh(ax=ax[1])
model_no_grp.mean()["Weight"].sort_values().plot.barh(ax=ax[2])

ax[0].set_title("Price Euros")
ax[1].set_title("Screen Size")
ax[2].set_title("Weight kg")
plt.tight_layout()
plt.show()
```

0.5s



From this, we can say that:

- Workstation is the highest average price but has lower average screen size and weight than Gaming type

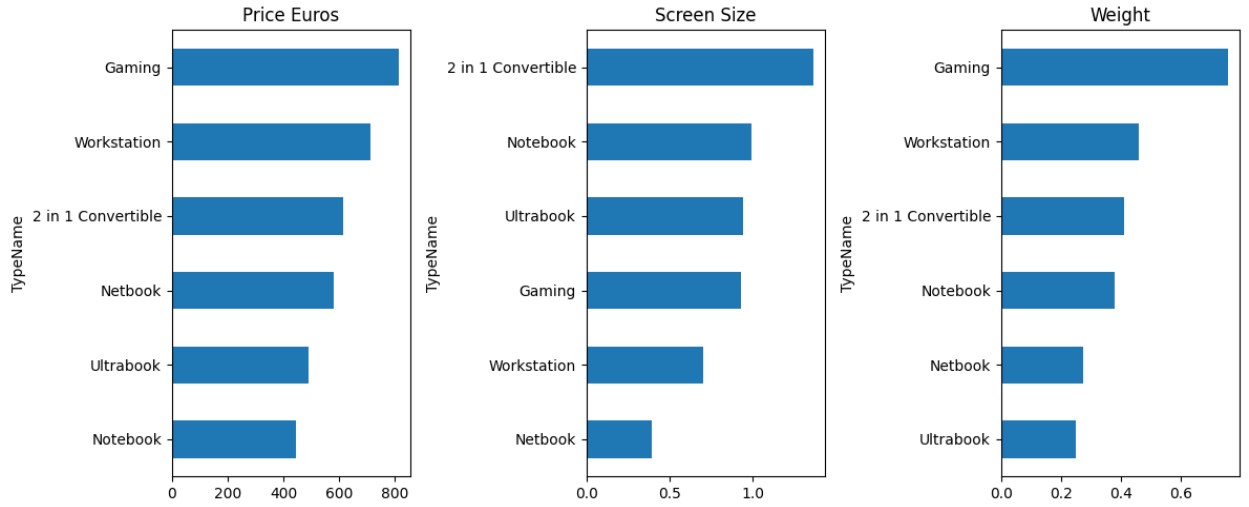
- Netbook is the lowest average price, screen size, and weight

After we know about min, max, and mean like above, lets visualize about standard deviation for data dissemination.

```
fig, ax = plt.subplots(1, 3, figsize=(12, 5))
model_no_grp.std()["Price_euros"].sort_values().plot.barh(ax=ax[0])
model_no_grp.std()["Inches"].sort_values().plot.barh(ax=ax[1])
model_no_grp.std()["Weight"].sort_values().plot.barh(ax=ax[2])

ax[0].set_title("Price Euros")
ax[1].set_title("Screen Size")
ax[2].set_title("Weight")
plt.tight_layout()
plt.show()
```

0.5s



From this, we can say that:

- Gaming is the highest standard deviation in price euros and weight

- Netbook is the lowest standard deviation in price euros and screen size but has higher standard deviation in weight than Ultrabook

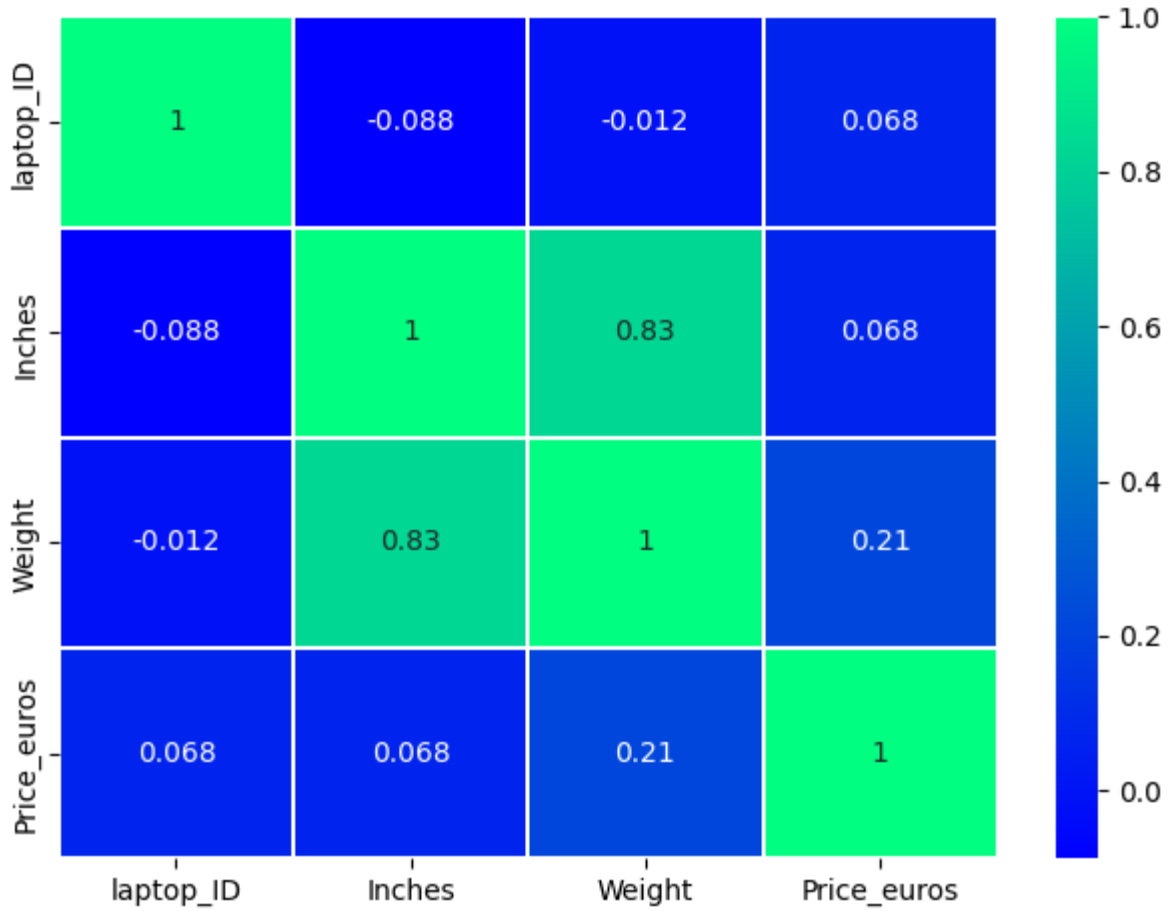
- 2 in 1 Convertible is the highest standard deviation in screen size

- Workstation is the second highest standard deviation in price euros and weight but has lower standard deviation in screen size than other except Netbook

Now, we know about min, max, mean and standard deviation from this data based on TypeName. After that, lets check about correlation from this data.

```
sns.heatmap(df.corr(), annot=True, linewidths=0.2, linecolor="white", cmap="winter")
plt.tight_layout()
plt.show()
```

0.3s



From this, we can say that:

- Laptop_ID is has weak correlation with other data

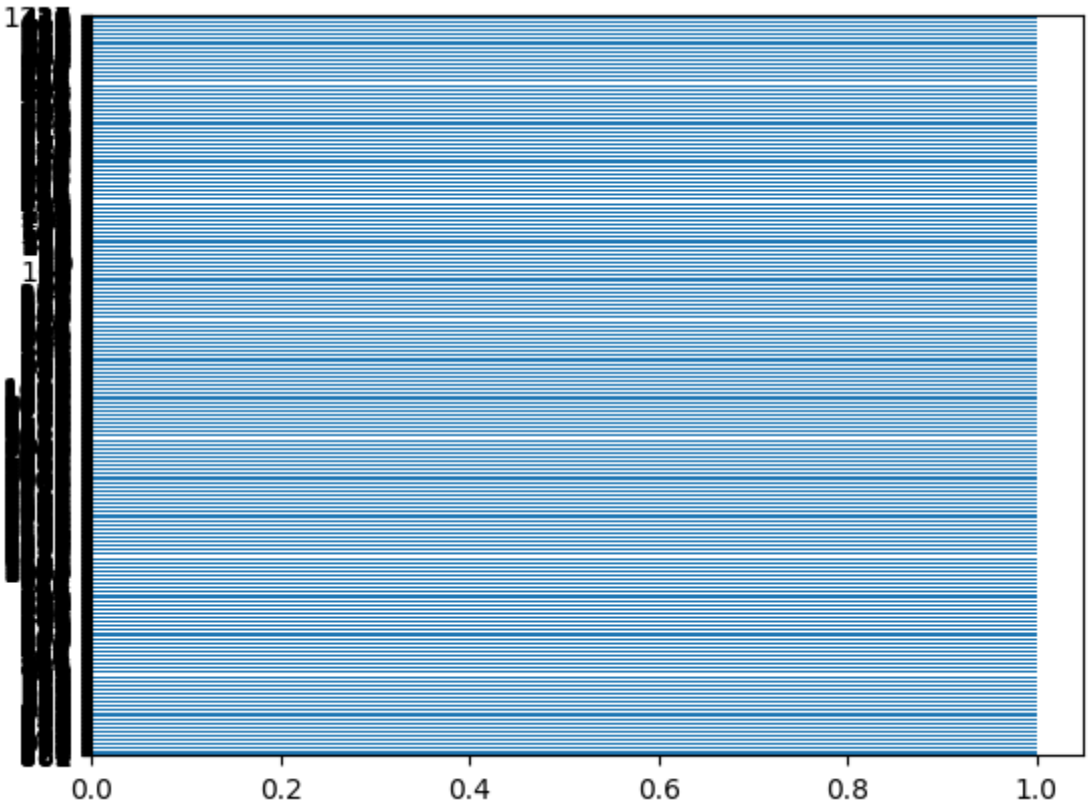
- Inches has weak correlation with price but has high correlation with weight

- Weight has weak correlation with price but still higher than inches and has high correlation with inches

We know that Laptop_ID has high standard deviation but has weak correlation with other data. So, lets check about values in laptop_ID.

```
df["laptop_ID"].value_counts().plot.barh()
```

✓ 9.9s



From this, we can say that:

- This data have no unique values. Thats why this data has weak correlation with other data

- We dont need this data. So, we can drop this data

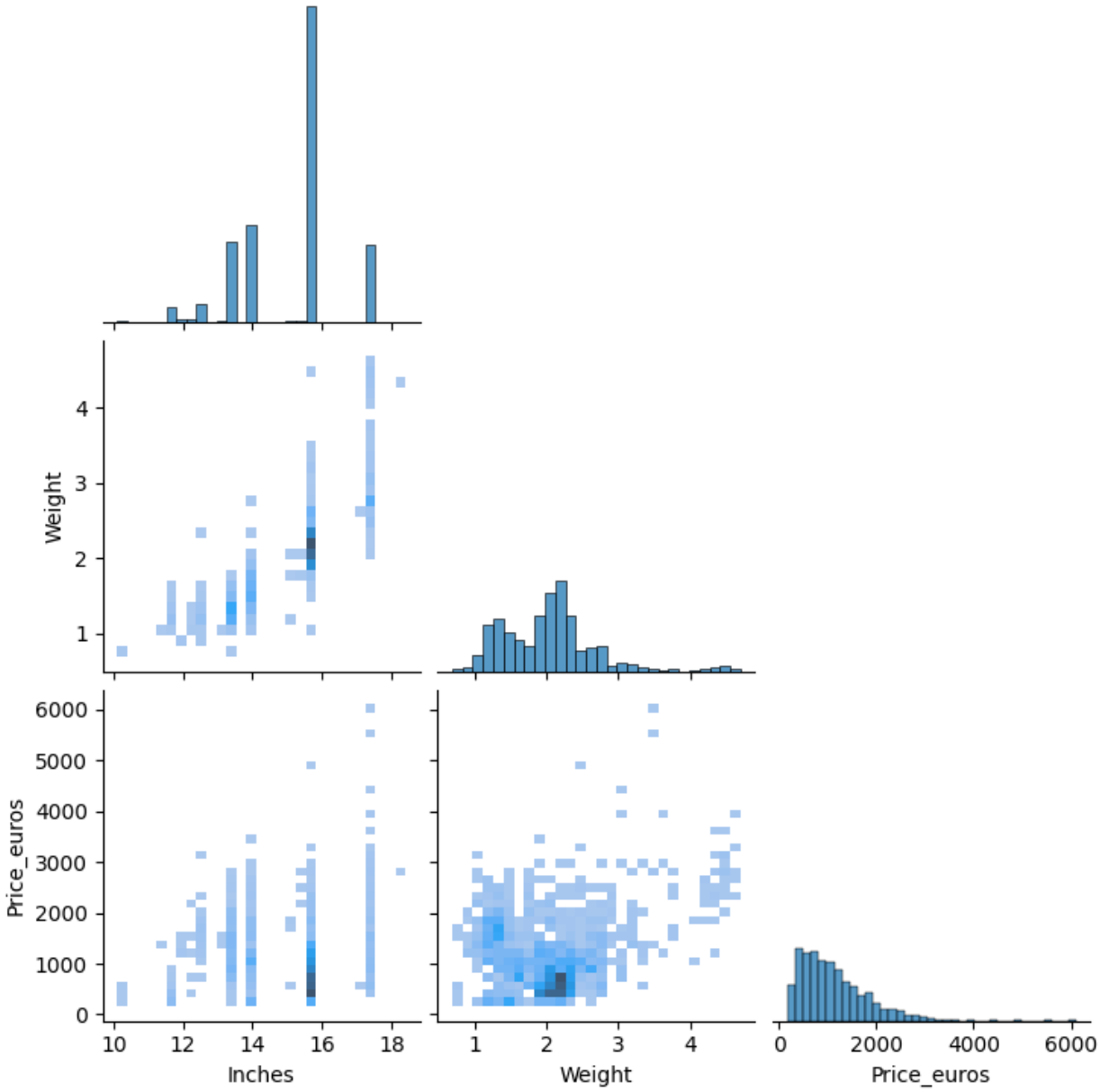
Lets, drop the data.

```
df.drop("laptop_ID", axis=1, inplace=True)
```
✓ 0.4s

Now, we finally drop the data of laptop_ID. We don't need this data because this data has so much unique values and low correlation with other data.

Then, lets check the pairplot of the data for sure.

```
# Pairplot
sns.pairplot(df, corner=True, kind="hist")
```
✓ 1.7s



From This, we can say that:

- The data has positive correlation with other data

- We get the same insights as from the above heatmap. Need to confirm the same with further analysis.

Lets, make function that can make annotate for the visualization.

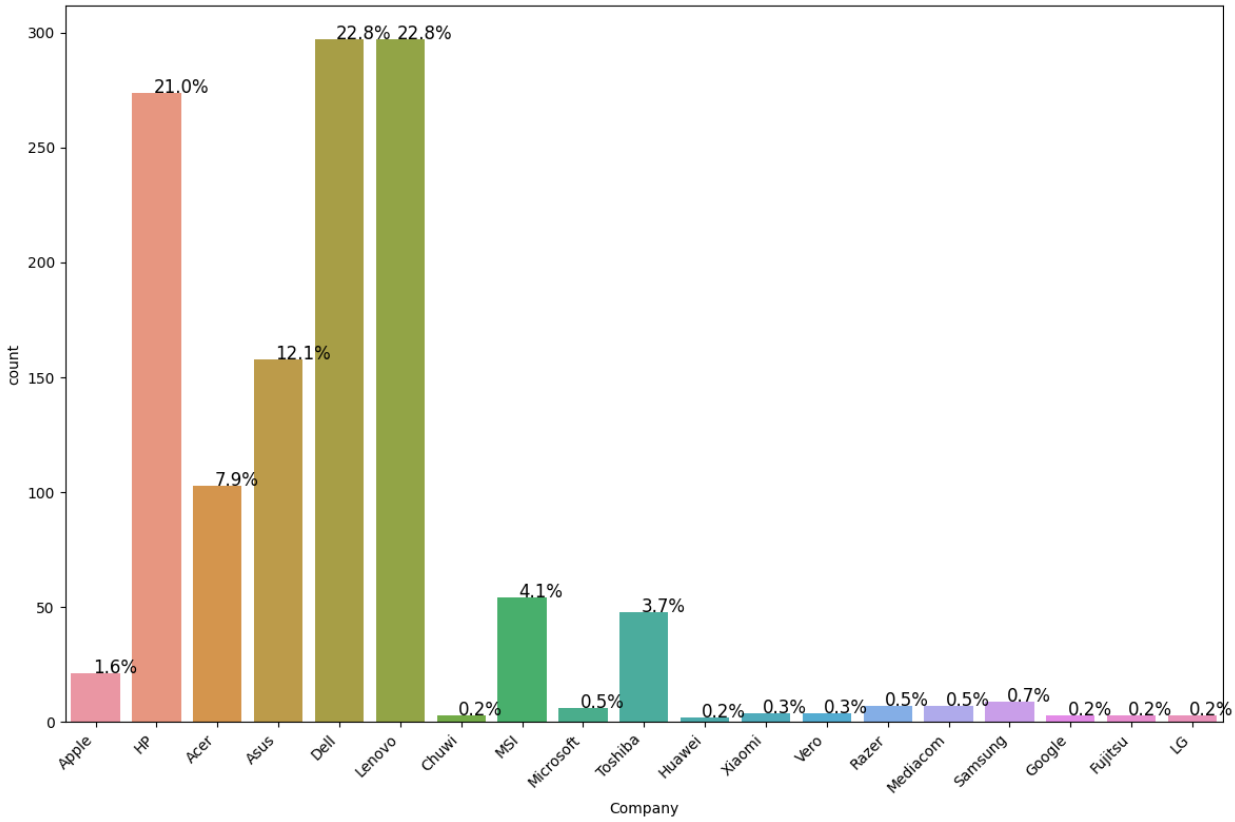
```python
def bar_perc(plot, feature):
    total = len(feature)
    for p in plot.patches:
        percentage = "{:.1f}%".format(100 * p.get_height() / total)
        x = p.get_x() + p.get_width() / 2 - 0.05
        y = p.get_y() + p.get_height()
        plot.annotate(percentage, (x, y), size=12)
```

✓ 0.4s

Now, lets do the Univariate Analysis for categorical variables like this.

```python
# Univariate Analysis of Categorical Variables
plt.figure(figsize=(12, 8))
ax = sns.countplot(data=df, x="Company")
bar_perc(ax, df["Company"])
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.tight_layout()
plt.show()
```

✓ 0.4s



From this, we can say that:

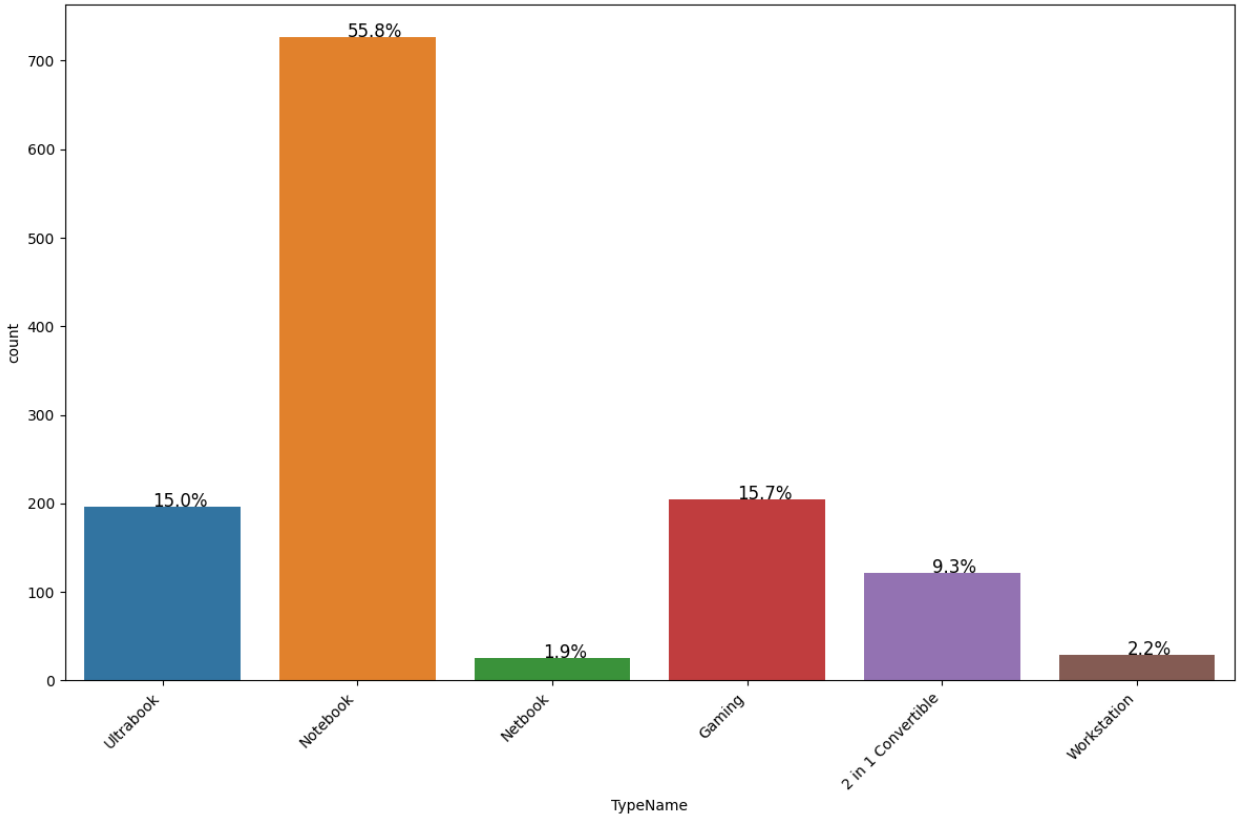
- The company that has highest values is Dell and Lenovo

- The company that has lowest values is LG, Fujitsu, Google, Huawei, and Chuwi

Then, lets visualize the TypeName variable.

```
# Univariate Analysis of Categorical Variables
plt.figure(figsize=(12, 8))
ax = sns.countplot(data=df, x="TypeName")
bar_perc(ax, df["TypeName"])
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.tight_layout()
plt.show()
```

✓ 0.2s



From this, we can say:

- The data Type which is highest value is Notebook. It means, so much people buy Notebook than other

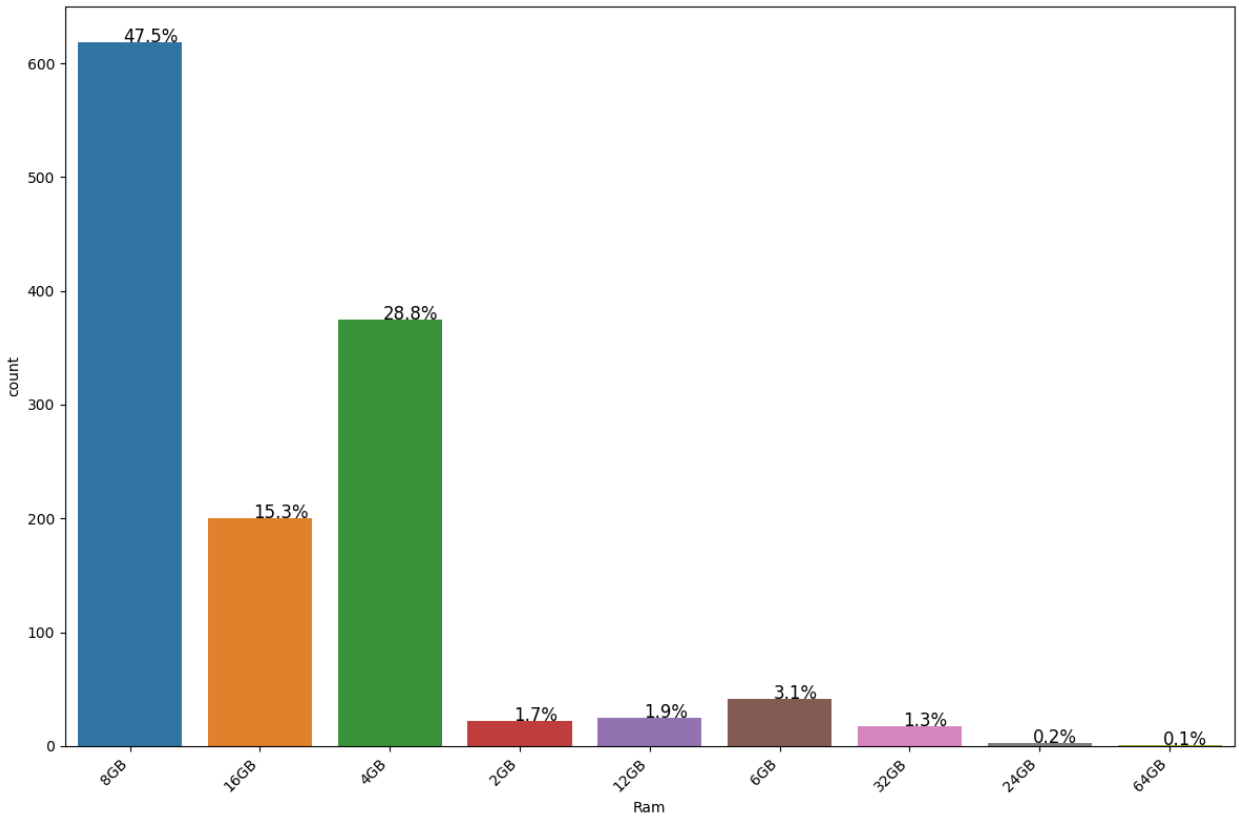
- The data Type which is lowest value is Netbook. It just 1.9 % from all data TypeName

- Gaming is the second highest value after Notebook

Then, lets visualize the Ram variable.

```
# Univariate Analysis of Categorical Variables
plt.figure(figsize=(12, 8))
ax = sns.countplot(data=df, x="Ram")
bar_perc(ax, df["Ram"])
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.tight_layout()
plt.show()
```

✓ 0.3s



From this, we can say that:

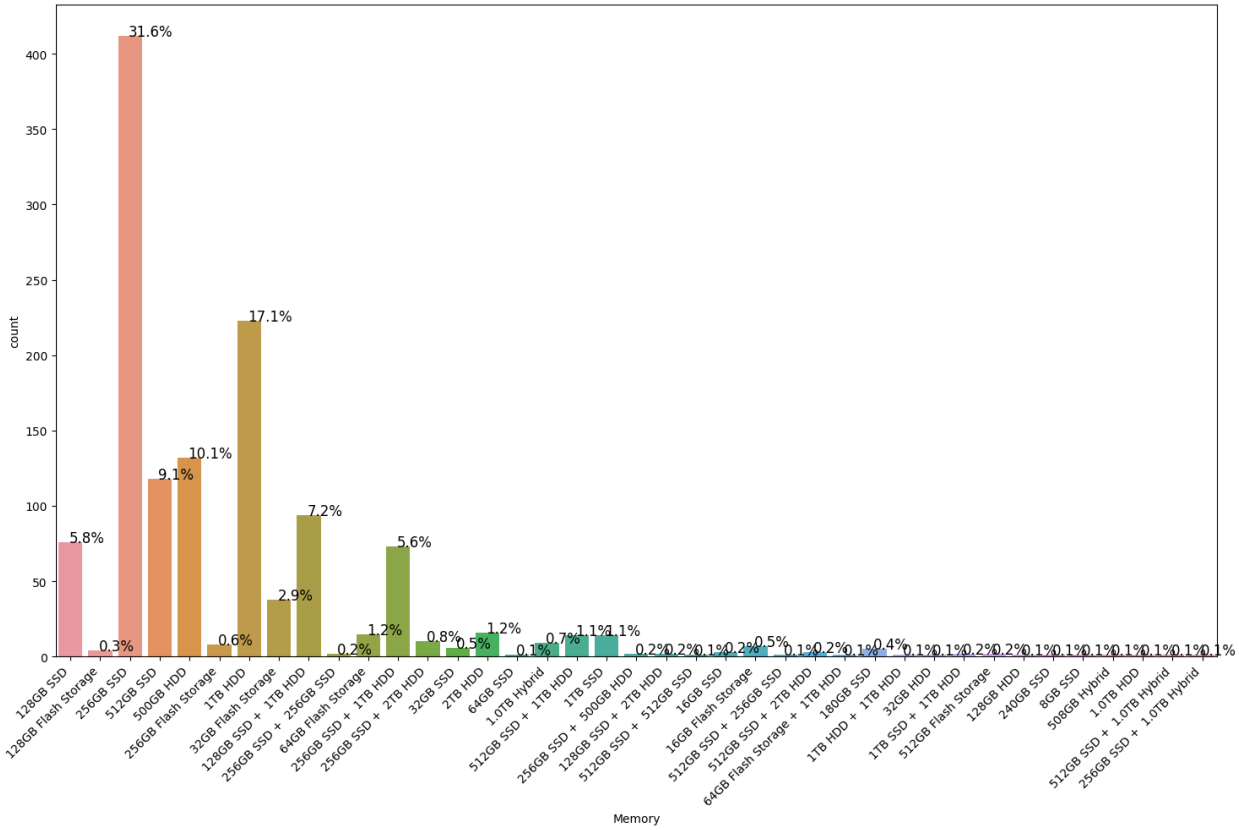
- The data Ram which is highest value is 8GB. It's mean so much people use ram 8GB than other ram

- The data Ram which is lowest value is 64GB. It's 0.1 % people from all data that use 64GB

Then, lets visualize the Memory variable.

```
# Univariate Analysis of Categorical Variables
plt.figure(figsize=(15, 10))
ax = sns.countplot(data=df, x="Memory")
bar_perc(ax, df["Memory"])
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.tight_layout()
plt.show()
```

✓ 0.7s



From this, we can say that:

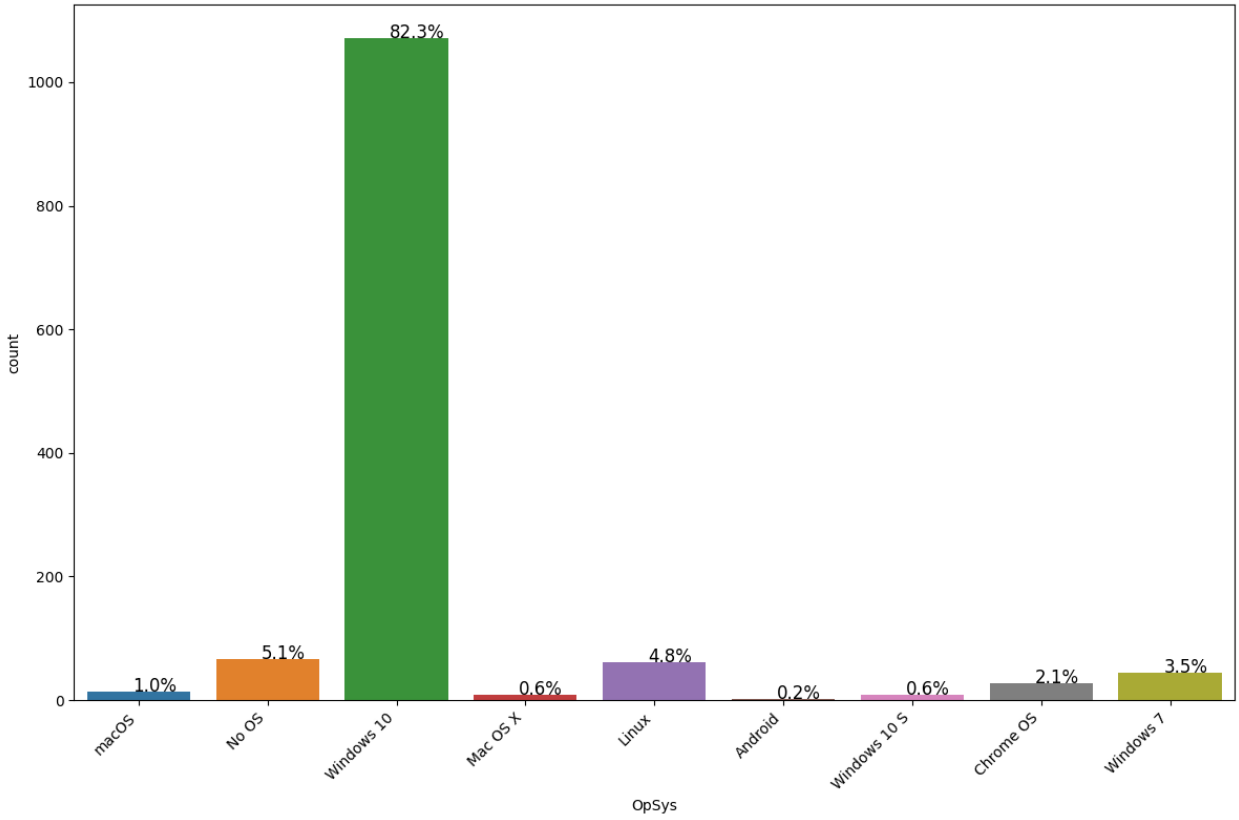
- The data Memory which is highest value is 256GB SSD. It's mean 31.6% people from all data use 256GB SSD

- The data Memory which is lowest value is very much that has just 0.1 % data

Then, lets visualize the Operating System variable.

```
# Univariate Analysis of Categorical Variables
plt.figure(figsize=(12, 8))
ax = sns.countplot(data=df, x="OpSys")
bar_perc(ax, df["OpSys"])
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.tight_layout()
plt.show()
```

0.3s



From this, we can say that:

- The data Operation System which is highest value is Windows 10. It's mean 82.3% people from all data use Windows 10

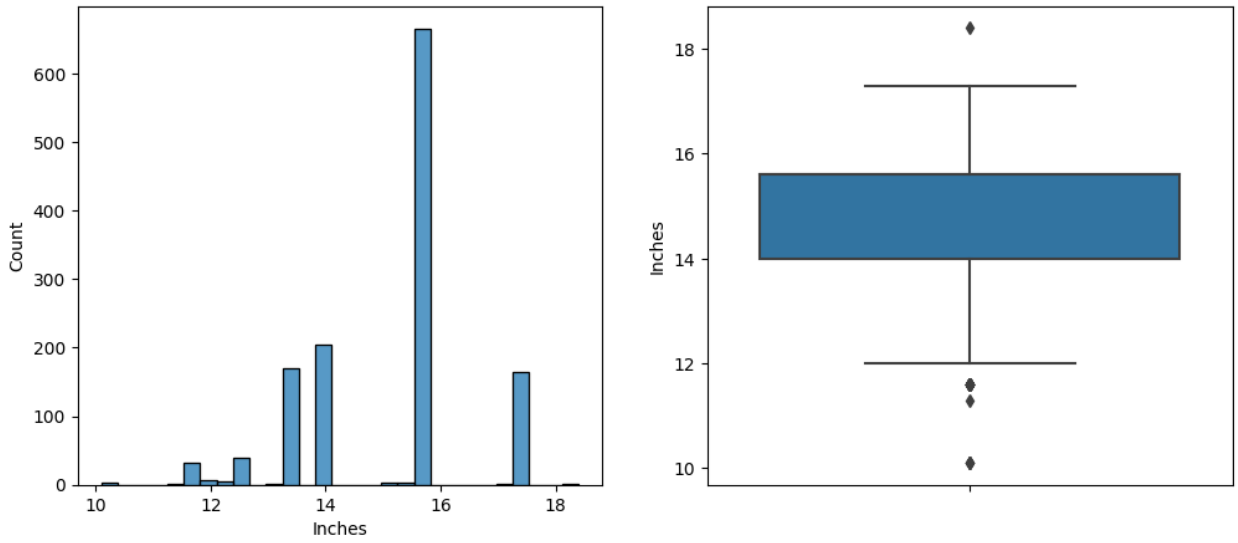
- The data Operation System which is lowest value is Android. It just 0.2 % people from all data use Android

After we know about categorical variables, now lets do the Univariate Analysis for numerical variables. First, lets visualize the Inches or screen size variable.

```
# Univariate Analysis for Numerical Variables

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(df["Inches"])
plt.subplot(1, 2, 2)
sns.boxplot(y=df["Inches"])
plt.show()
```

✓ 0.3s



From this, we can say that:

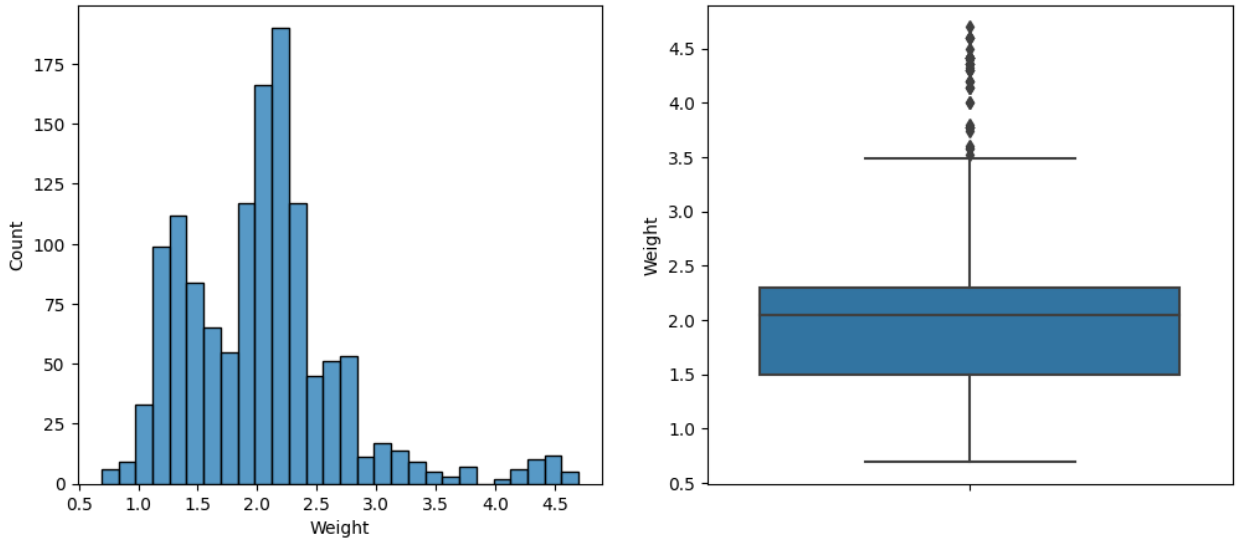
- The highest data from inches data is around 15 - 16
- Data inches has 3 low outlier and 1 high outlier
- The minimum value is 10.1 and the maximum is 18.4
- The interquartile range (IQR) is between 14 and around 15

Then, lets visualize the weight variable.

```
# Univariate Analysis for Numerical Variables

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(df["Weight"])
plt.subplot(1, 2, 2)
sns.boxplot(y=df["Weight"])
plt.show()
```

✓ 0.3s



From this, we can say that:

- The minimum value is 0.69 and the maximum value is 4.7

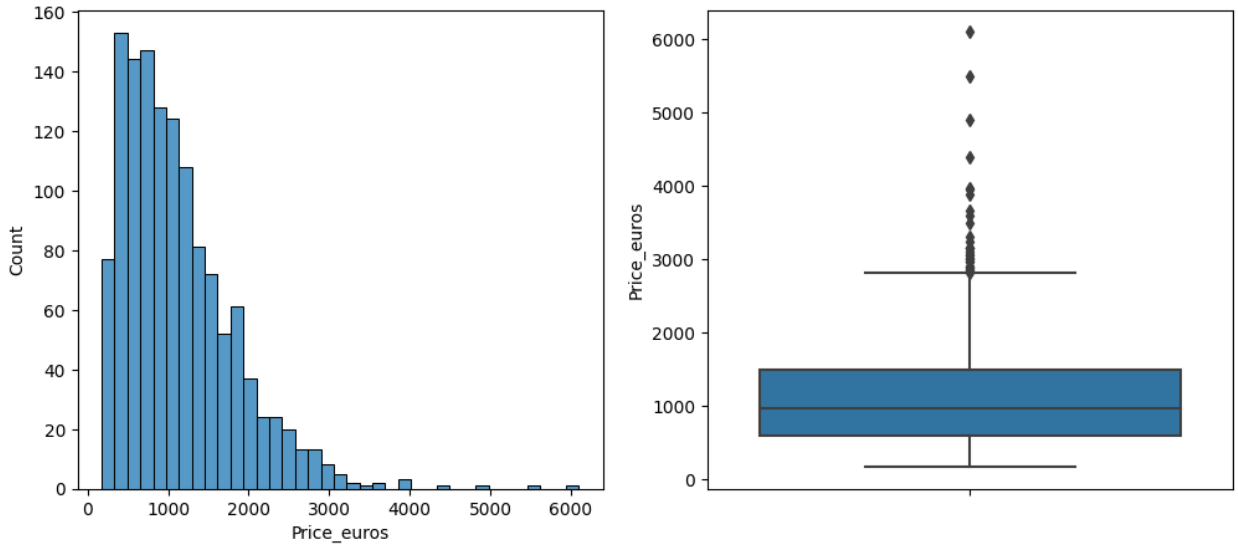
- Data has many high outlier

- The interquartile range (IQR) is between 1.5 and around 2

```
# Univariate Analysis for Numerical Variables

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(df["Price_euros"])
plt.subplot(1, 2, 2)
sns.boxplot(y=df["Price_euros"])
plt.show()
```

✓ 0.3s



From this, we can say that:

- The minimum value of data is 174.0 and the maximum value of data is 6099.0

- The highest value of data is 474 which has 174 data

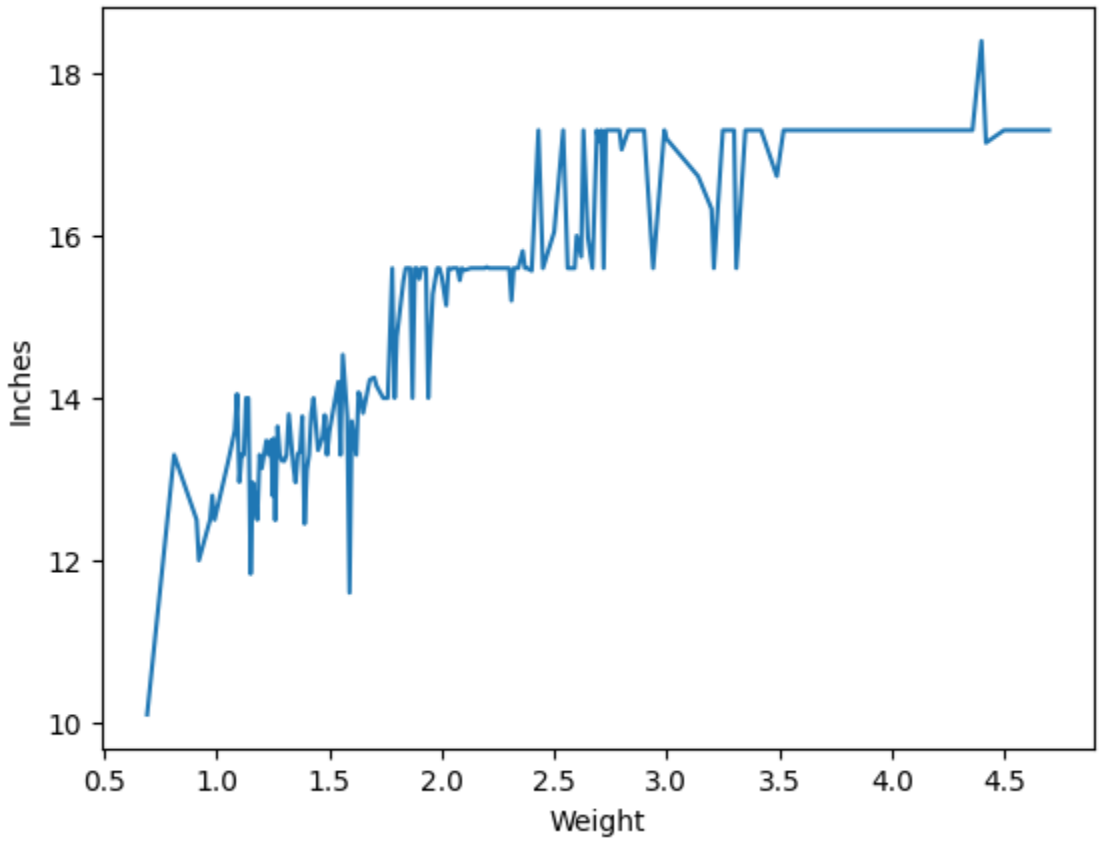
- Many product that has price under 4000 euro

- Data has high outlier

Now, we finally did the Univariate Analysis. Now, lets do the Bivariate Analysis. First, lets do the Bivariate Analysis of weight and inches.

```
# Bivariate Analysis of Numerical Variables Weight vs Inches
sns.lineplot(data=df, x="Weight", y="Inches", ci=None)
```

✓ 0.3s



From this, we can say that:

- These 2 data has positive correlation

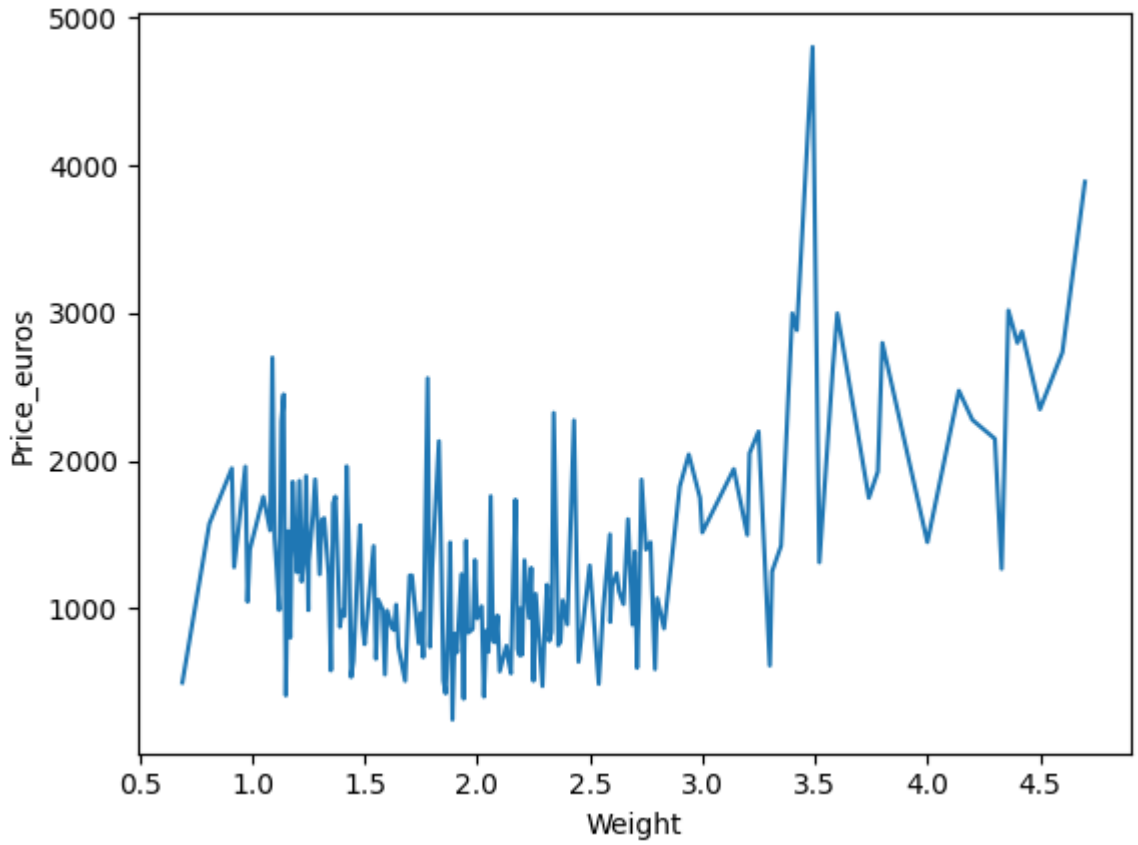
- The bigger weight has higher inches

- Some high weight has decreased

Then, lets do the Bivariate Analysis of weight and price euros.

```
# Bivariate Analysis of Numerical Variables Weight vs Price
sns.lineplot(data=df, x="Weight", y="Price_euros", ci=None)
```

✓ 0.3s



From this, we can say that:

- Lots of data weight around 1.0 kg - 3.0 kg has low price

- some data has higher price with weight around 3.4kg - 3.5kg

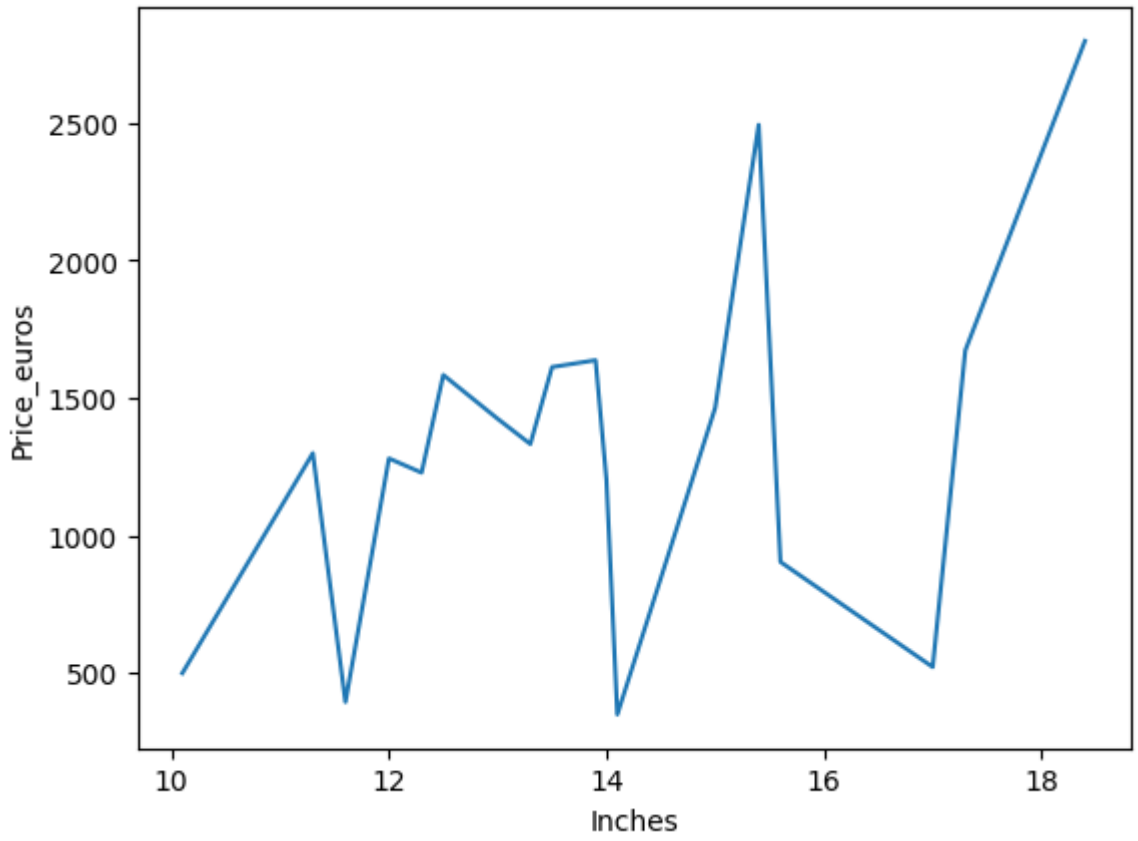
- Data around 3kg has decreased greatly

- The maximum weight has high price which is above 3000 euro

Then, lets do the Bivariate Analysis of Inches and prices euros.

```
# Bivariate Analysis of Numerical Variables Inches vs Price
sns.lineplot(data=df, x="Inches", y="Price_euros", ci=None)
```

✓ 0.2s



From this, we can say that:

- The minimum of screen size has low price and the maximum of screen size has highest price

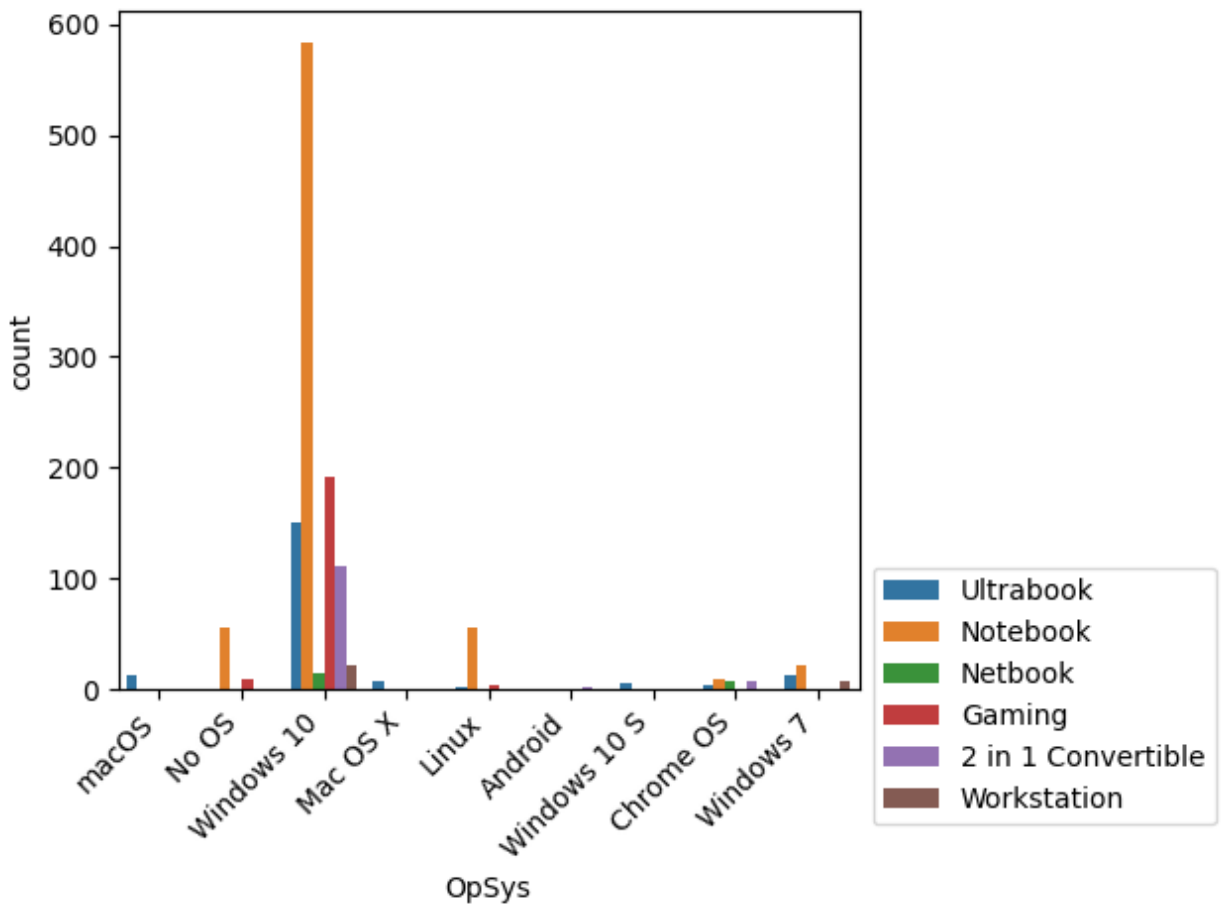
- Some data of screen size has low price and get decreased

- lots of data of screen size has high screen size and high price

We did the Bivariate Analysis for numerical variables. Now, lets do the Bivariate Analysis for categorical variables. First, lets do the Bivariate Analysis for Operating System and TypeName Variables.

```
# Bivariate Analysis of Categorical Variables OpSys vs TypeName
ax = sns.countplot(data=df, x="OpSys", hue="TypeName")
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.legend(bbox_to_anchor=(1, 0.2))
plt.tight_layout()
plt.show()
```

✓ 0.4s



From this, we can say that:

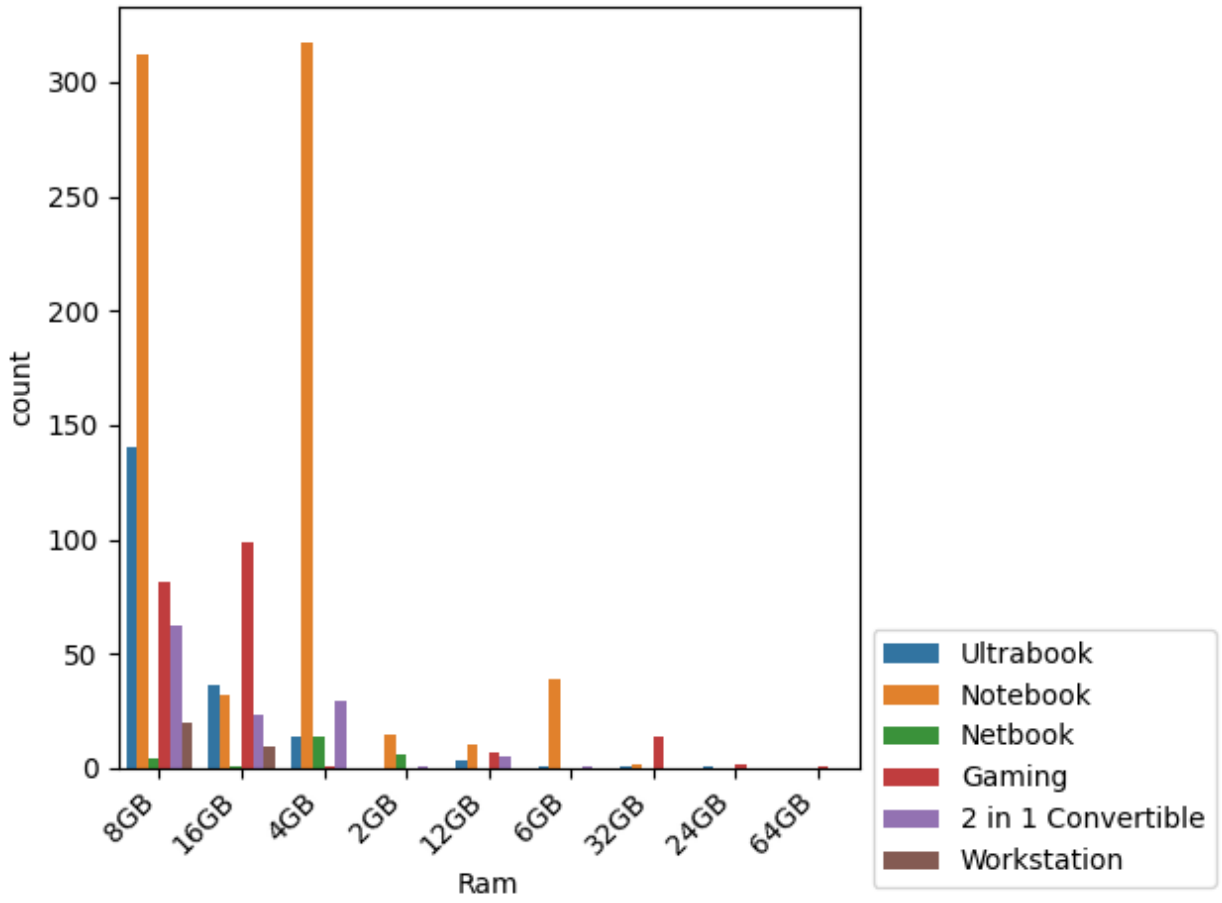
- Lots of people use Windows 10

- The highest type that people use in Windows 10 is Notebook and the lowest type that people use in Windows 10 is Netbook

- Android has just 1 type that people use and it is Workstation

Then, lets do the Bivariate Analysis for Ram and TypeName variables.

```
# Bivariate Analysis of Categorical Variables Ram vs TypeName
ax = sns.countplot(data=df, x="Ram", hue="TypeName")
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.legend(bbox_to_anchor=(1, 0.2))
plt.tight_layout()
plt.show()
```



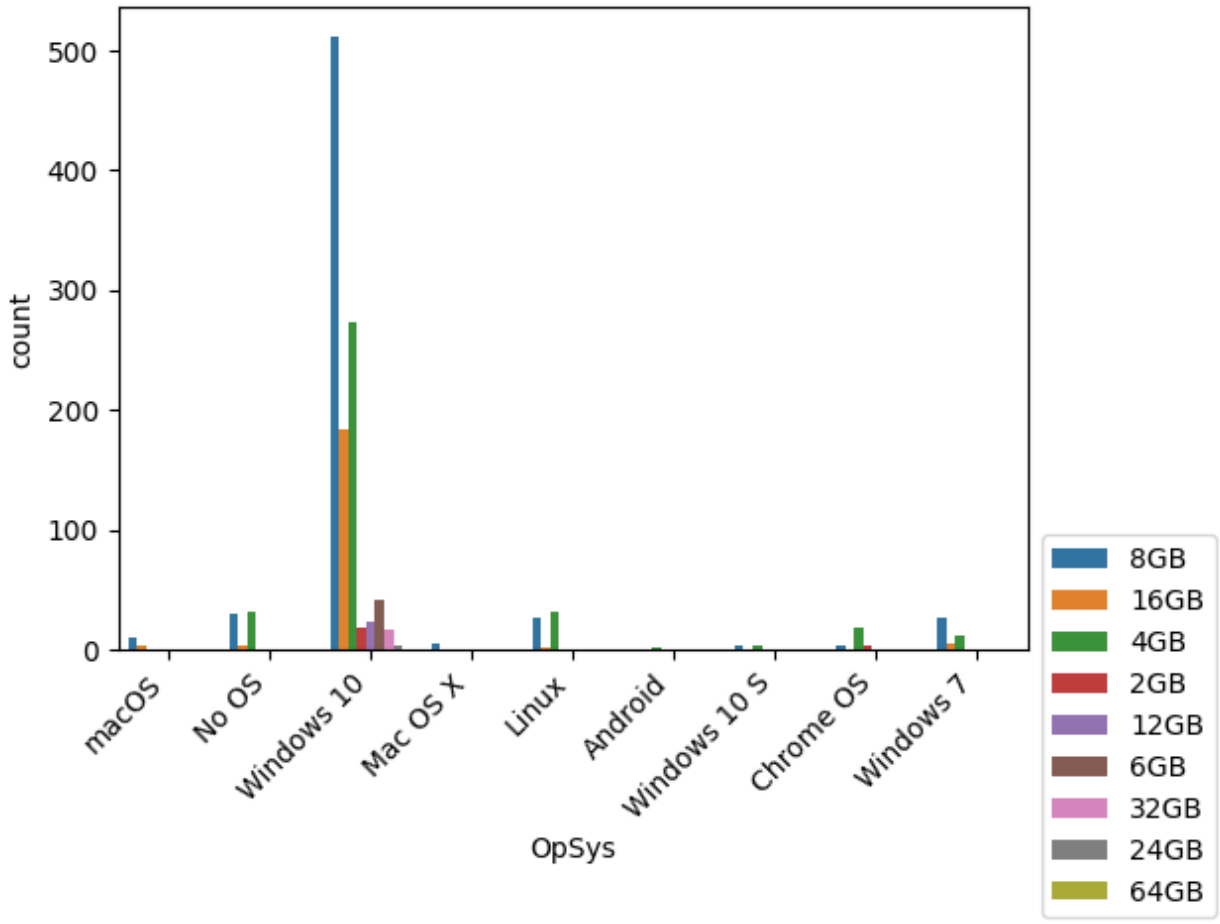
From This, we can say that:

- Lots of people use Ram 8GB and the highest is Notebook
- The highest value is Notebook
- Lots of people that use Ram 4GB use Notebook
- People who use 64GB use Gaming type
- Lots of people who use above 16GB use Gaming type
- Lots of people who use under 16GB use Notebook

Then, lets do the Bivariate Analysis for Operating System and Ram variables.

```
# Bivariate Analysis of Categorical Variables OpSys vs Ram
ax = sns.countplot(data=df, x="OpSys", hue="Ram")
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.legend(bbox_to_anchor=(1, 0.2))
plt.tight_layout()
plt.show()
```

✓ 0.5s



From this, we can say that:

- Lots of people who use macOS, No OS, Windows 10, and Windows 7 prefer use 8GB than other

- Most people using Windows 10 with Ram 8GB

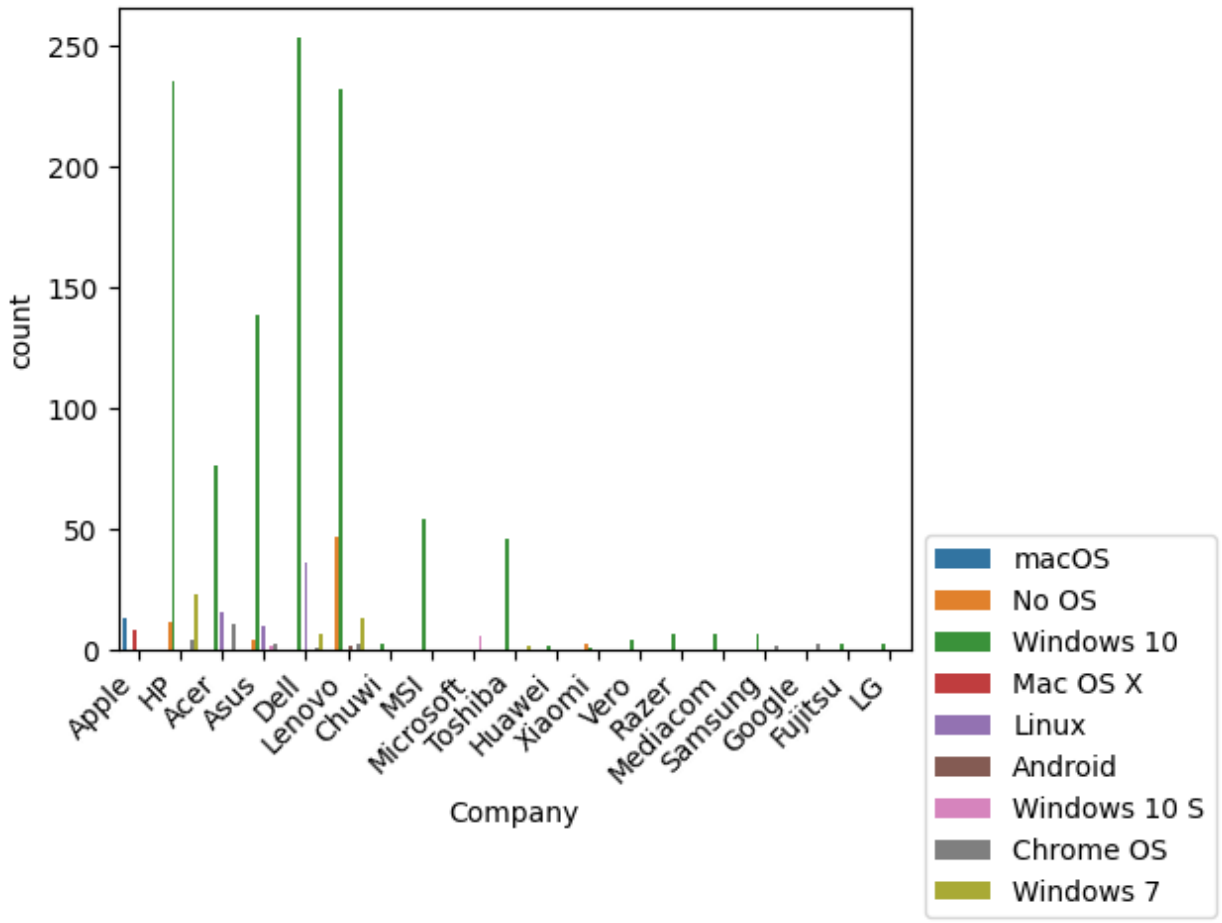
- The highest people who use Linux is preper use Ram 4GB than 8GB

- 1 people use Android with Ram 4GB

Then, lets do the Bivariate Analysis for Company and Operating System variables.

```
# Bivariate Analysis of Categorical Variables Company vs OpSys
ax = sns.countplot(data=df, x="Company", hue="OpSys")
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.legend(bbox_to_anchor=(1, 0.2))
plt.tight_layout()
plt.show()
```

✓ 0.8s



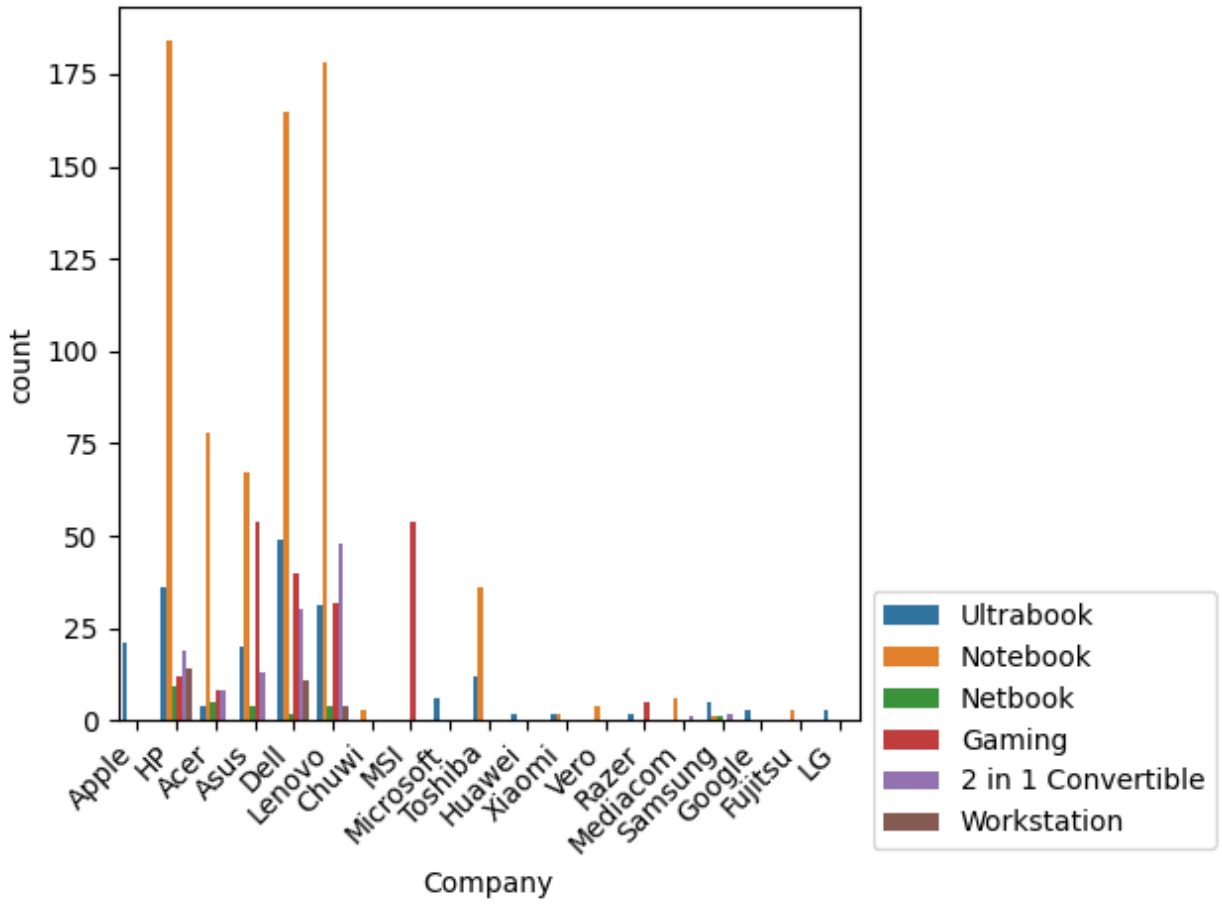
From this, we can say that:

- Many companies sell Operation System Windows 10 than other Operation System
- Dell has the highest sell for Windows 10
- Lenovo has the highest sell for No OS
- Apple has the highest sell for macOS and mac OS X
- Dell has the highest sell for Linux
- Microsoft has the highest sell for Windows 10 S

Then, Lets do the Bivariate Analysis for Company and TypeName.

```
# Bivariate Analysis of Categorical Variables Company vs TypeName
ax = sns.countplot(data=df, x="Company", hue="TypeName")
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment="right")
plt.legend(bbox_to_anchor=(1, 0.2))
plt.tight_layout()
plt.show()
```

✓ 0.6s



From this, we can say that:

- Many companies sell type of product is Notebook

- HP company is the highest sell for type Notebook and Lenovo company is the second highest sell for type Notebook after HP company

- Apple, Microsoft, Huawei, Google, and LG company just sell for type Ultrabook

- MSL company is the highest sell for type Gaming

- Dell company is the highest sell for type Ultrabook

- Hp, Acer, Asus, Dell, Lenovo, and Samsung company sell much type of product

Now, we finally clear the Exploratory Data Analysis (EDA). Lets continue to the next step.

5. Model Building

At this stage, a Data Scientist will create a machine learning model. According to the requirements of this task, the models that I will create are Linear Regression, Ridge, Lasso, Decision Tree, and Support Vector Machine models.

First, lets encode the data object with Label Encoder.

```
le = LabelEncoder()
df2 = df.copy()
for i in df.columns:
    if df[i].dtypes.name == "object":
        df[i] = le.fit_transform(df[i])
```

✓ 0.7s

And then, lets check 5 data from top.

```
df.head()
```

✓ 0.1s

	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price_euros
0	1	300	4	13.3	23	65	8	4	58	8	1.37	1339.69
1	1	301	4	13.3	1	63	8	2	51	8	1.34	898.94
2	7	50	3	15.6	8	74	8	16	53	4	1.86	575.00
3	1	300	4	15.4	25	85	1	29	9	8	1.83	2537.45
4	1	300	4	13.3	23	67	8	16	59	8	1.37	1803.60

+ Code + Markdown

After that, lets transform the data cause the data has high scala with other data and divided to training data and testing data.

```
scaler = StandardScaler()
XX = scaler.fit_transform(df)
X = XX[:, :-1]
y = XX[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

✓ 0.1s

```
(1042, 11) (261, 11) (1042,) (261,)
```

Now, lets make the first model. We make Linear Regression model first like this.

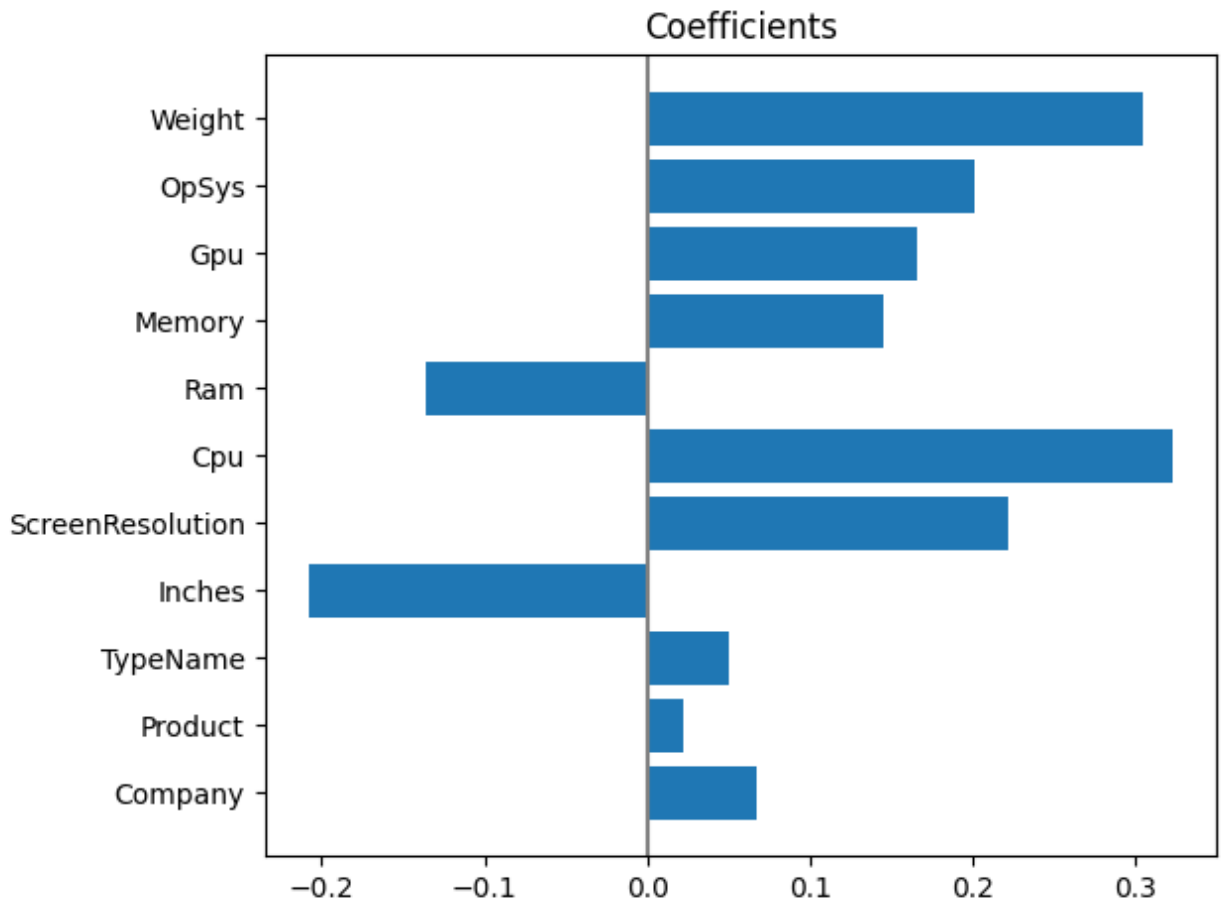
```
lr = LinearRegression().fit(X_train, y_train)
```

✓ 0.8s

Then, lets check the coefficient of each variables.

```
plt.barh(df.columns[:-1], lr.coef_.ravel())
plt.title("Coefficients")
plt.axvline(x=0, color=".5")
plt.tight_layout()
plt.show()
```

✓ 0.2s



From this we can say that:

- Weights is the highest coefficient

- Product has no coeffient and it means product dont have any correlation with price

Then, lets check the performance of model.

```
y_pred = lr.predict(X_test)
print("Coefficient of determination of training data\t :", r2_score(y_train, lr.predict(X_train)))
print("Coefficient of determination of testing data\t :", r2_score(y_test, y_pred))
print("MAE", mean_absolute_error(y_test, y_pred))
print("MSE", mean_squared_error(y_test, y_pred))
print("RMSE", mean_squared_error(y_test, y_pred) ** 2)
print("MAPE", mean_absolute_percentage_error(y_test, y_pred))
```

✓ 0.5s

```
Coefficient of determination of training data    : 0.511488134063621
Coefficient of determination of testing data     : 0.43355621013465506
MAE 0.5224178222680951
MSE 0.5892852307133787
RMSE 0.34725708313691994
MAPE 1.9499661608880172
```

The model has bad performance. Maybe we must set the parameter of model for get performance.

```
lr_params = {
    "fit_intercept": [True, False],
    "copy_X": [True, False],
    "positive": [True, False]
}
search_lr = RandomizedSearchCV(lr, lr_params, random_state=42)
search_lr.fit(X_train, y_train)
```

✓ 0.1s

```
c:\Users\Administrator\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\model_selection
parameters 8 is smaller than n_iter=10. Running 8 iterations. For exhaustive searches, use GridSearchCV
  warnings.warn(
```

RandomizedSearchCV
estimator: LinearRegression
LinearRegression
LinearRegression()

The parameter has built same like before, lets check the best score of model.

```
search_lr.best_score_
```

✓ 0.4s

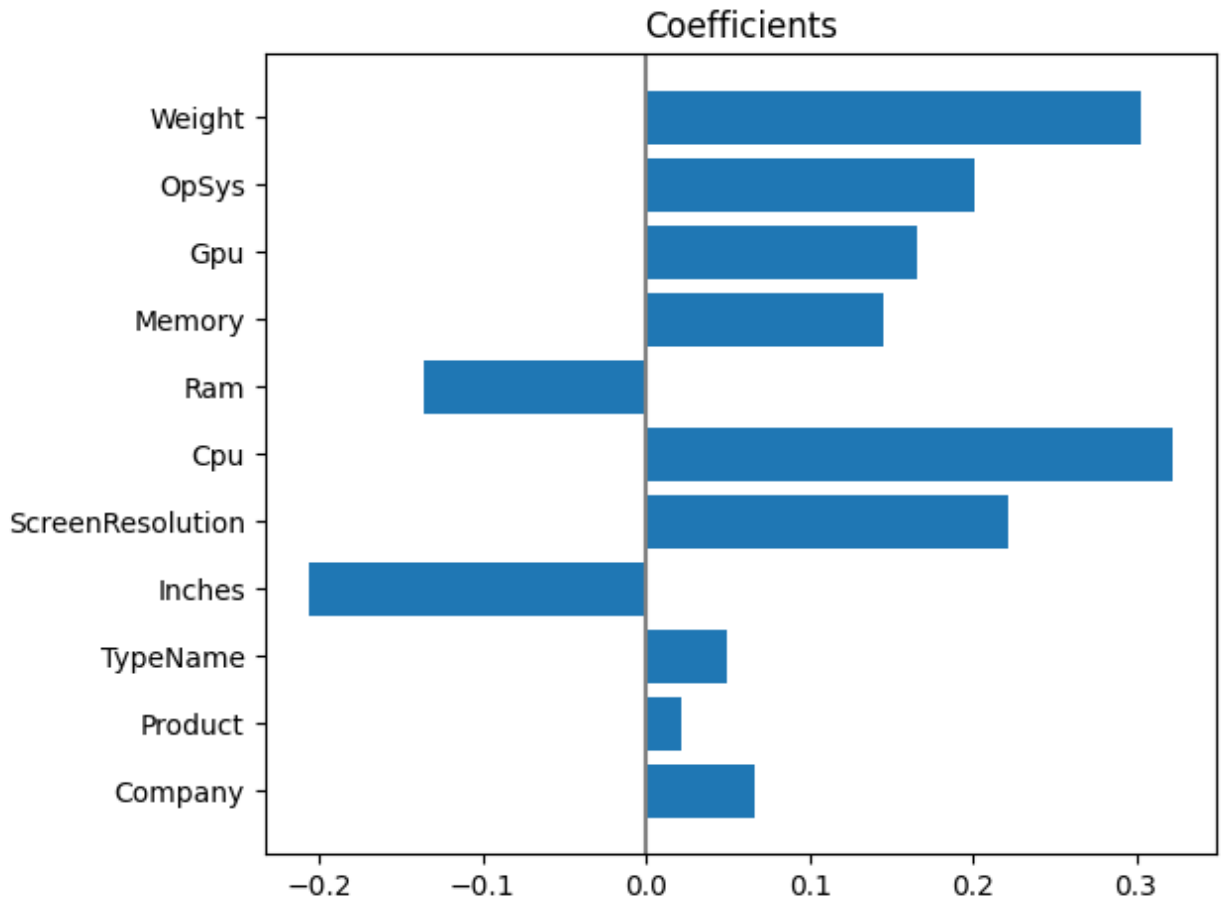
```
0.4910831240084675
```

The performance of model still bad. Lets make other model like Ridge.

```
ridge = Ridge().fit(X_train, y_train)

✓ 0.6s
```

And then, lets check coefficient of each variables.



From this, we can say that:

- All variables has coeffient and correlation with the dependen variable
- CPU is the highest variable and Product is the lowest variable

Then, lets check the performance of model.

```
y_pred = ridge.predict(X_test)
print("Coefficient of determination of training data\t :", r2_score(y_train, ridge.predict(X_train)))
print("Coefficient of determination of testing data\t :", r2_score(y_test, y_pred))
print("MAE", mean_absolute_error(y_test, y_pred))
print("MSE", mean_squared_error(y_test, y_pred))
print("RMSE", mean_squared_error(y_test, y_pred) ** 2)
print("MAPE", mean_absolute_percentage_error(y_test, y_pred))
```

✓ 0.4s

```
Coefficient of determination of training data    : 0.51148719445405
Coefficient of determination of testing data     : 0.4334507749878167
MAE 0.5224937733355224
MSE 0.5893949174571326
RMSE 0.3473863687243001
MAPE 1.965246190661255
```

The performance of model is worse than before. Lets set the best parameter of the model.

```
ridge_params = {
    "alpha": np.linspace(0.0001, 10, 1000),
    "fit_intercept": [True, False],
    "copy_X": [True, False],
    "solver": ["auto", "svd", "cholesky", "lsqr", "sparse_cg", "sag", "saga", "lbfgs"],
    "positive": [True, False],
    "random_state": [0, 42, 101]
}
search_ridge = RandomizedSearchCV(ridge, ridge_params, random_state=42)
search_ridge.fit(X_train, y_train)
```

✓ 0.3s

```
search_ridge.best_score_
```

✓ 0.1s

```
0.49135341747346795
```

The performance of model still bad and worse then before. Lets make Lasso model.

```
lasso = Lasso().fit(X_train, y_train)
```

✓ 0.1s

And then, lets make Support Vector Machine model.

```
svm = SVR().fit(X_train, y_train)
```

✓ 0.1s Python

Lets, check the evaluation from this model

```
y_pred = svm.predict(X_test)
print("Coefficient of determination of training data\t :", r2_score(y_train, s
print("Coefficient of determination of testing data\t :", r2_score(y_test, y_p
print("MAE", mean_absolute_error(y_test, y_pred))
print("MSE", mean_squared_error(y_test, y_pred))
print("RMSE", mean_squared_error(y_test, y_pred) ** 2)
print("MAPE", mean_absolute_percentage_error(y_test, y_pred))
```

✓ 0.1s Python

```
Coefficient of determination of training data    : 0.8300207793214978
Coefficient of determination of testing data     : 0.6324640698766956
MAE 0.3279586067724965
MSE 0.3823565537361685
RMSE 0.1461965341849995
MAPE 2.877638770254518
```

Now, get better model than before. Before we use this model, lets check other model like Decision Tree model.

```
tree = DecisionTreeRegressor(max_depth=5, random_state=42)
tree.fit(X_train, y_train)
```

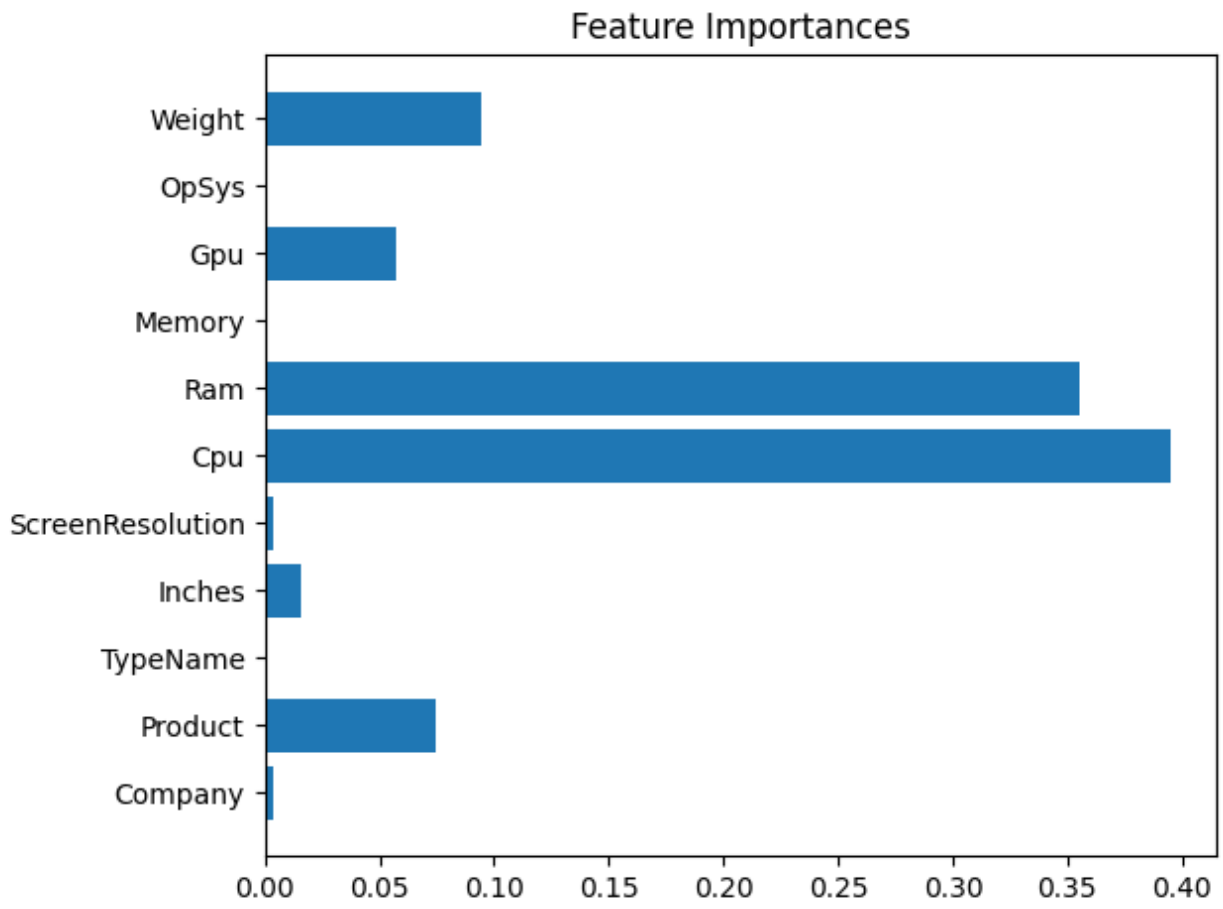
✓ 0.5s Python

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, random_state=42)
```

We made Decision Tree model. Lets check the feature importances.

```
plt.barh(df.columns[:-1], tree.feature_importances_)
plt.axvline(x=0, color=".5")
plt.title("Feature Importances")
plt.tight_layout()
plt.show()
```

✓ 0.2s



From this, we can say that:

- Some feature don't have data or 0 feature importances like OpSys, Memory, and TypeName
- Cpu is the highest feature importances
- We know that Weight must be get more feature importances cause that feature have strong correlation with target

Then, lets check the performance of this model.

```
y_pred = tree.predict(X_test)
print("Coefficient of determination of training data\t :", r2_score(y_train, t
print("Coefficient of determination of testing data\t :", r2_score(y_test, y_p
print("MAE", mean_absolute_error(y_test, y_pred))
print("MSE", mean_squared_error(y_test, y_pred))
print("RMSE", mean_squared_error(y_test, y_pred) ** 2)
print("MAPE", mean_absolute_percentage_error(y_test, y_pred))
```

✓ 0.3s Python

```
Coefficient of determination of training data    : 0.7831924634068845
Coefficient of determination of testing data     : 0.6375255962624704
MAE 0.3926914843189776
MSE 0.37709092491761936
RMSE 0.14219756565522565
MAPE 2.899262430876708
```

Now, we get best performance than before. We can use this model, too. But, before that, we can test of any parameter with this model with RandomizedSearchCV.

```
tree_params = {
    "criterion": ["squared_error", "friedman_mse", "absolute_error", "poisson"
    "splitter": ["best", "random"],
    "max_depth": np.arange(50),
    "min_samples_split": np.arange(11),
    "min_samples_leaf": np.arange(11),
}
search_tree = RandomizedSearchCV(tree, tree_params, random_state=42)
search_tree.fit(X_train, y_train)
```

✓ 0.2s Python

RandomizedSearchCV

estimator: DecisionTreeRegressor

DecisionTreeRegressor

```
DecisionTreeRegressor(criterion='absolute_error', max_depth=36,
                      max_features='sqrt', min_samples_split=10,
                      random_state=42)
```

Lets check the best score from this model.

```
search_tree.best_score_
```

✓ 0.3s

```
0.6725235409156765
```

We get nice best score. Now, lets make Decision Tree model again.

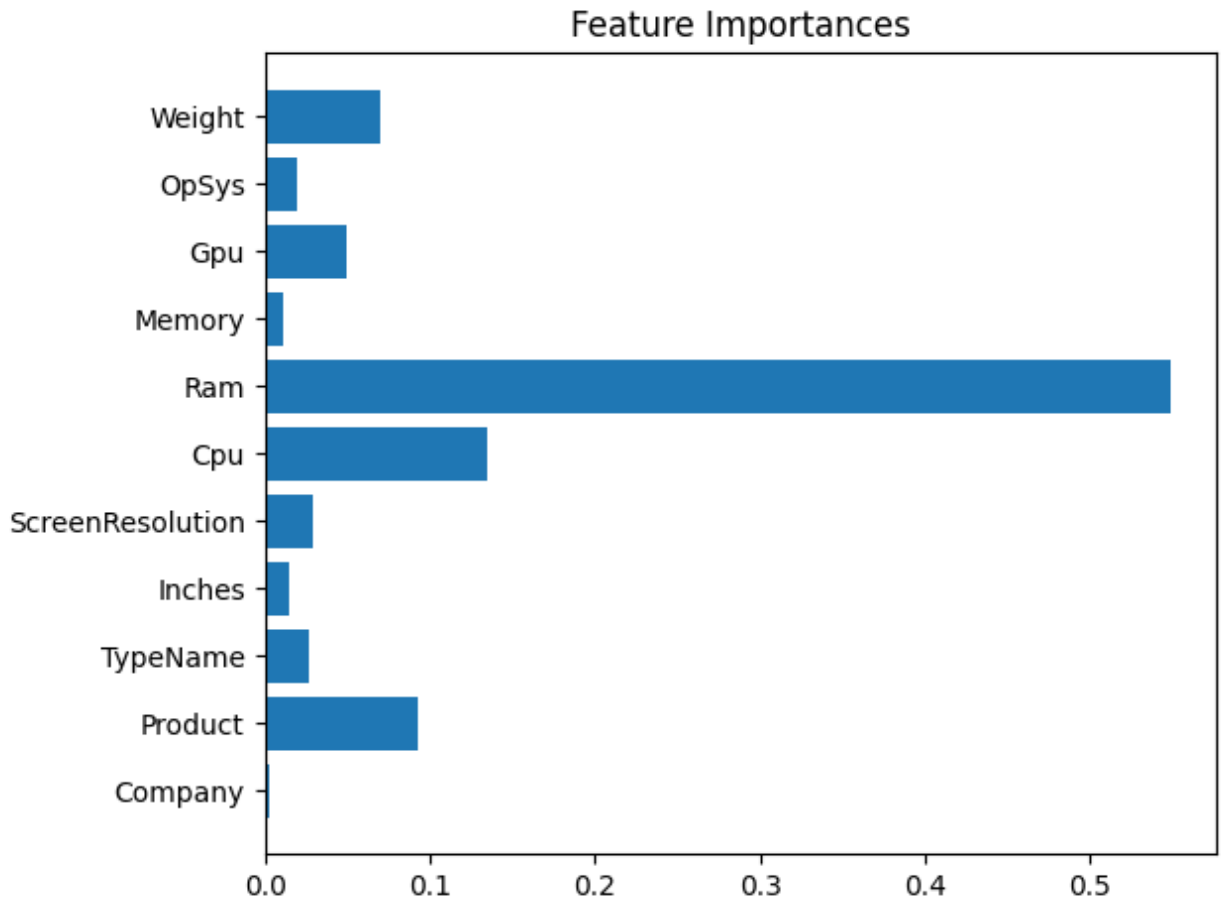
```
tree = search_tree.best_estimator_.fit(X_train, y_train)
```

✓ 0.3s

After that, lets check the feature importances again.

```
plt.barh(df.columns[:-1], tree.feature_importances_)
plt.axvline(x=0, color=".5")
plt.title("Feature Importances")
plt.tight_layout()
plt.show()
```

✓ 0.3s



From this, we can say that:

- Now, all feature is importance for this data
- Ram is the highest feature importance for this data

- OpSys, Memory, and TypeName have feature importance now. Its mean these feature is importance too for this data
- Company is the lowest feature importances

Then, lets check the performance of this model.

```
y_pred = tree.predict(X_test)
print("Coefficient of determination of training data\t :", r2_score(y_train, t
print("Coefficient of determination of testing data\t :", r2_score(y_test, y_p
print("MAE", mean_absolute_error(y_test, y_pred))
print("MSE", mean_squared_error(y_test, y_pred))
print("RMSE", mean_squared_error(y_test, y_pred) ** 2)
print("MAPE", mean_absolute_percentage_error(y_test, y_pred))
```

✓ 0.4s Python

```
Coefficient of determination of training data    : 0.8351101810038516
Coefficient of determination of testing data     : 0.7277906909956371
MAE 0.32491617024602926
MSE 0.28318595477425546
RMSE 0.08019428498140665
MAPE 1.887423659832916
```

The performance of this model is better than the other model that we built.

6. Evaluation Model

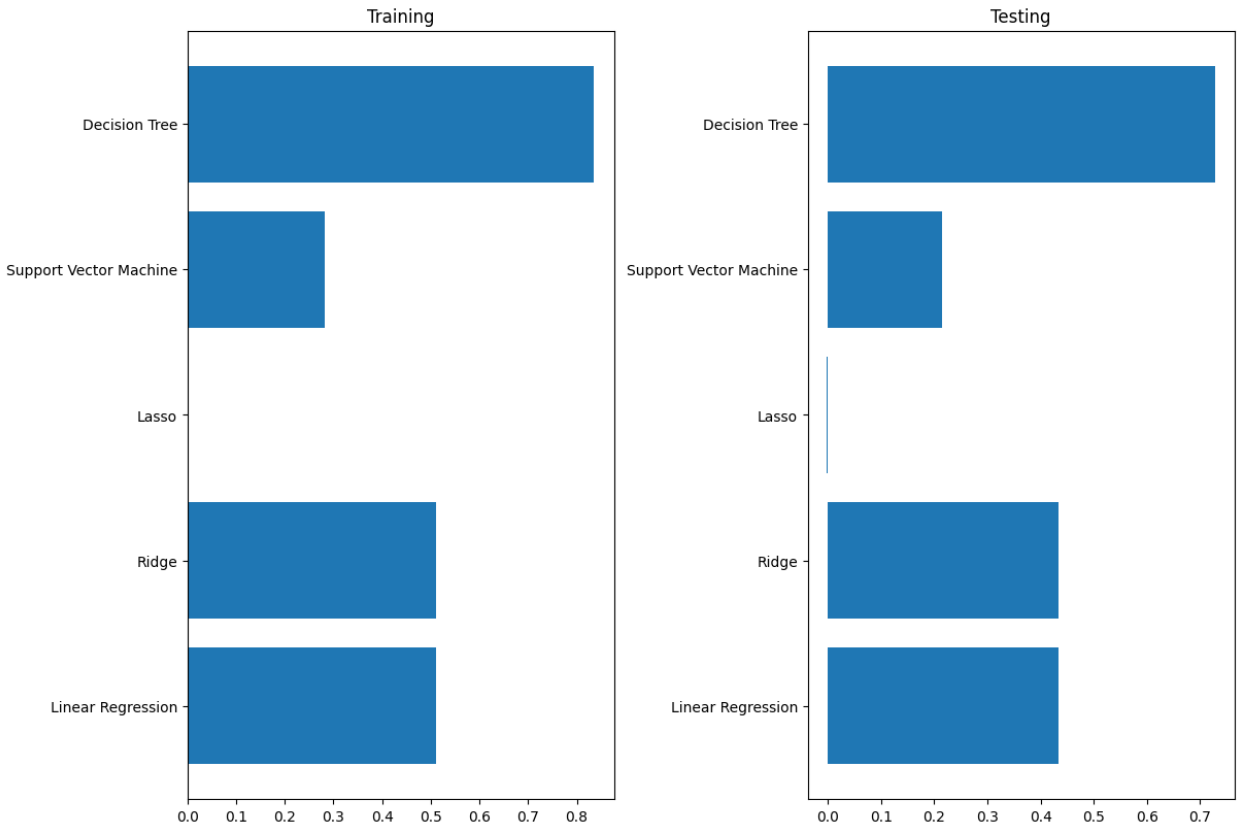
We have built many model like Linear Regression, Ridge, Lasso, Support Vector Machine, and Decision Tree. Now, we have to choose which model is better than the other model that we built. We can do the comparison of model first like this.

```
models = [lr, ridge, lasso, svm, tree]
model_names = ["Linear Regression", "Ridge", "Lasso", "Support Vector Machine", "Decision Tree"]
fig, ax = plt.subplots(1, 2, figsize=(12, 8), layout="constrained")
ax[0].barh([i for i in model_names], [r2_score(y_train, i.predict(X_train)) for i in models])
ax[1].barh([i for i in model_names], [r2_score(y_test, i.predict(X_test)) for i in models])

ax[0].set_title("Training")
ax[1].set_title("Testing")

plt.show()
```

✓ 0.5s

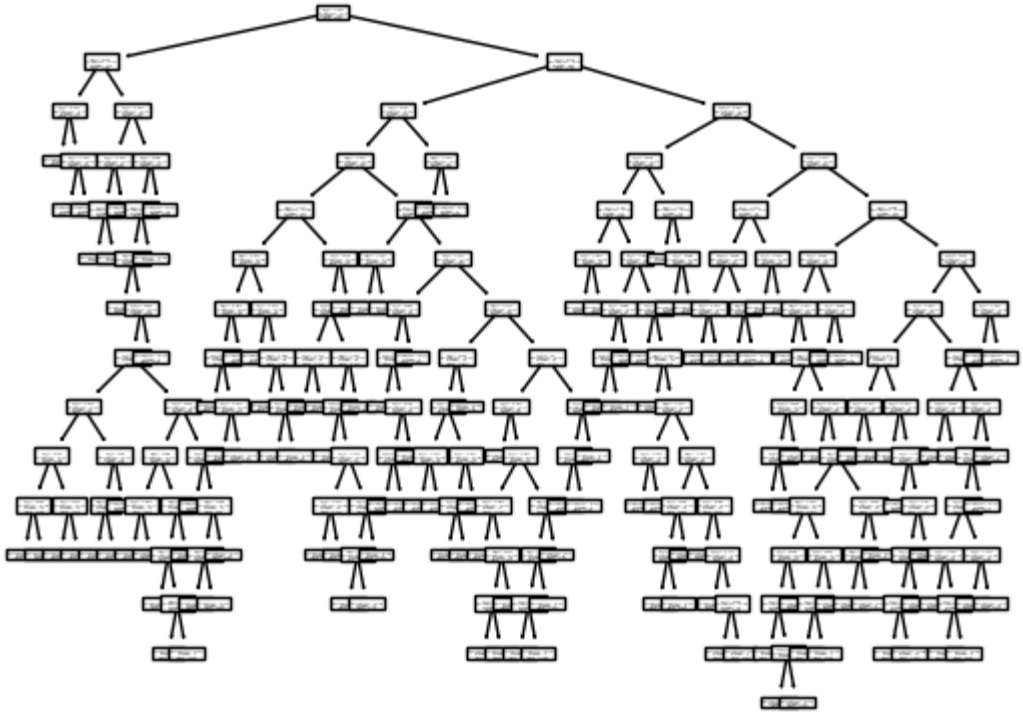


As we can see graphic above, we can conclude that, we can use this Decision Tree model for sure. We can also check why this Decision Tree model has higher performance than the other model.

```
from sklearn.tree import plot_tree

plot_tree(tree)
```

✓ 16.1s



We can understand why this model has higher parameter than the other model from this plot tree