PES UNIVERSITY

ಪಿಇಎಸ್ ವಿಶ್ವವಿದ್ಯಾಲಯ

IDS PROJECT ON LAPTOP DETAILS

Project team members

Name
Saiprakash L Shetty
Dhanik Shetty
Sumanth Prasad

DETAILS OF THE DATASET

Total number of columns:12
Total number of rows:1304
Categorical column:8
Numerical column:4
Data as missing values and NaN: 3.69% (578)

```
sample.describe()
```

	Inches	Ram	Weight	Price_euros
count	1190.000000	1107.000000	1101.000000	1303.000000
mean	14.997983	8.345077	2.049114	1123.686992
std	1.417040	5.010674	0.677242	699.009043
min	10.100000	2.000000	0.690000	174.000000
25%	14.000000	4.000000	1.540000	599.000000
50%	15.600000	8.000000	2.040000	977.000000
75%	15.600000	8.000000	2.300000	1487.880000
max	18.400000	64.000000	4.700000	6099.000000

2. Data Cleaning

- In categorical columns, all the NaN values are replaced by previous row values.
- In numerical columns, all the NaN values are replaced by the mean of that column

```
#data cleaning for numerical column
sample['Ram']=sample['Ram'].fillna(sample['Ram'].mean())
sample['Ram']
```

```
#categoirical cleaning
sample['Company']=sample['Company'].fillna(method='ffill')
sample['Company']
```

Before data cleaning

	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price_euros
0	Apple	MacBook Pro	Ultrabook	13.30	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8.0	128GB SSD	Intel Iris Plus Graphics 640	macOS	NaN	1339.69
1	NaN	Macbook Air	Ultrabook	13.30	1440x900	Intel Core i5 1.8GHz	8.0	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.340000	898.94
2	HP	250 G6	Notebook	15.60	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8.0	256GB SSD	Intel HD Graphics 620	No OS	1.860000	575.00
3	Apple	MacBook Pro	Ultrabook	15.40	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16.0	512GB SSD	AMD Radeon Pro 455	macOS	1.860000	2537.45
4	Apple	MacBook Pro	Ultrabook	13.30	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	10.0	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.370000	1803.60
5	Acer	Aspire 3	Notebook	15.60	1366x768	AMD A9-Series 9420 3GHz	4.0	500GB HDD	AMD Radeon R5	Windows 10	2.100000	400.00
6	Apple	MacBook Pro	Ultrabook	15.40	IPS Panel Retina Display 2880x1800	Intel Core i7 2.2GHz	16.0	256GB Flash Storage	Intel Iris Pro Graphics	Mac OS X	2.040000	2139.97
7	NaN	Macbook Air	Ultrabook	13.30	1440x900	Intel Core i5 1.8GHz	8.0	256GB Flash Storage	Intel HD Graphics 6000	macOS	1.340000	1158.70

After data cleaning

	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price_euros
0	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8.000000	128GB SSD	Intel Iris Plus Graphics 640	macOS	2.049114	1339.69
1	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8.000000	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.340000	898.94
2	HP	250 G6	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8.000000	256GB SSD	Intel HD Graphics 620	No OS	1.860000	575.00
3	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16.000000	512GB SSD	AMD Radeon Pro 455	macOS	1.860000	2537.45
4	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8.345077	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.370000	1803.60
5	Acer	Aspire 3	Notebook	15.6	1366x768	AMD A9-Series 9420 3GHz	4.000000	500GB HDD	AMD Radeon R5	Windows 10	2.100000	400.00
6	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.2GHz	16.000000	256GB Flash Storage	Intel Iris Pro Graphics	Mac OS X	2.040000	2139.97
7	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8.000000	256GB Flash Storage	Intel HD Graphics 6000	macOS	1.340000	1158.70

BEFORE DATA CLEANING

```
df.isnull().sum()
```

```
Company               67
Product                0
TypeName               0
Inches               113
ScreenResolution       0
Cpu                    0
Ram                  196
Memory                 0
Gpu                    0
OpSys                  0
Weight               202
Price_euros            0
dtype: int64
```

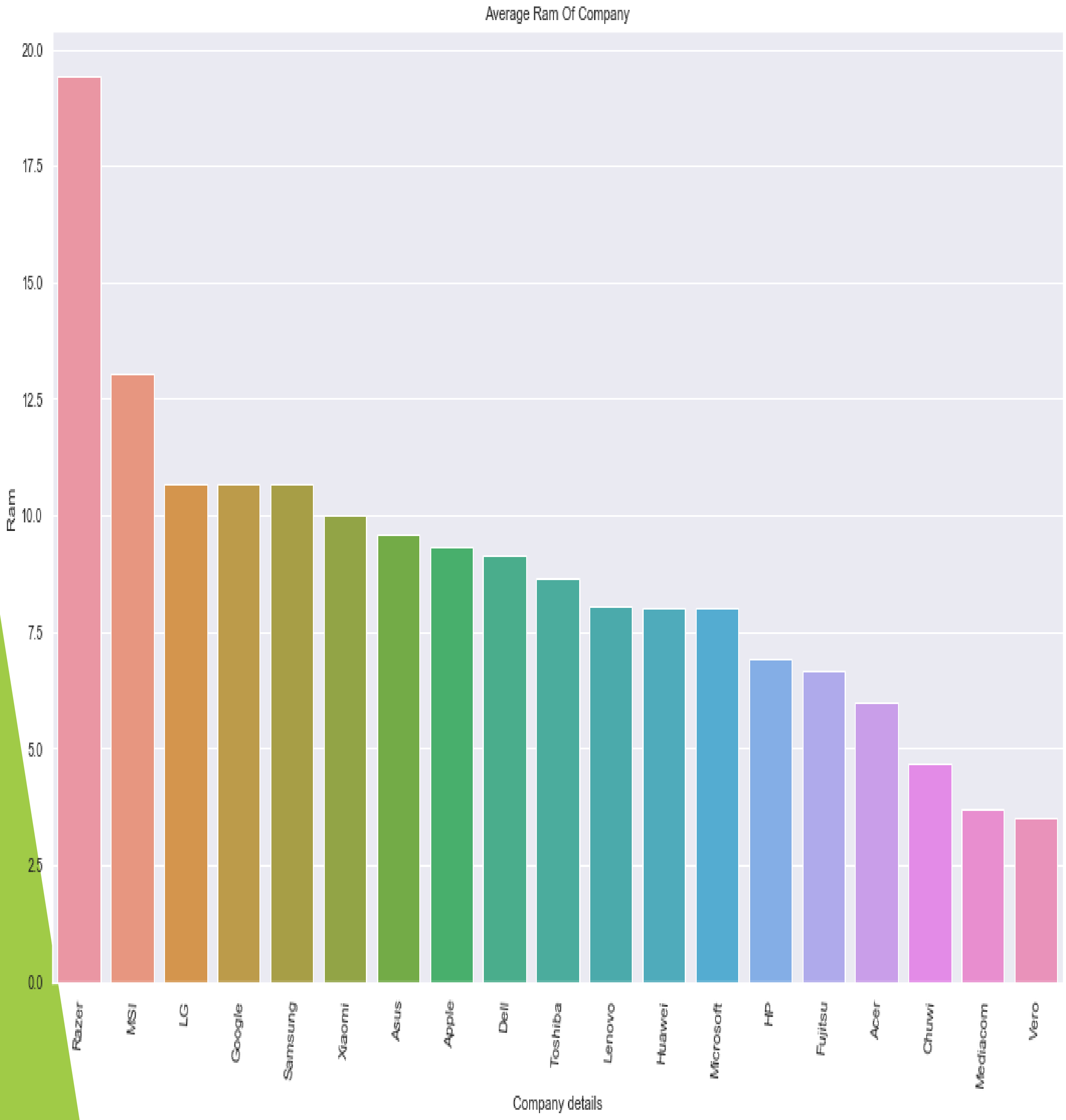
AFTER DATA CLEANING

```
df.isnull().sum()
```

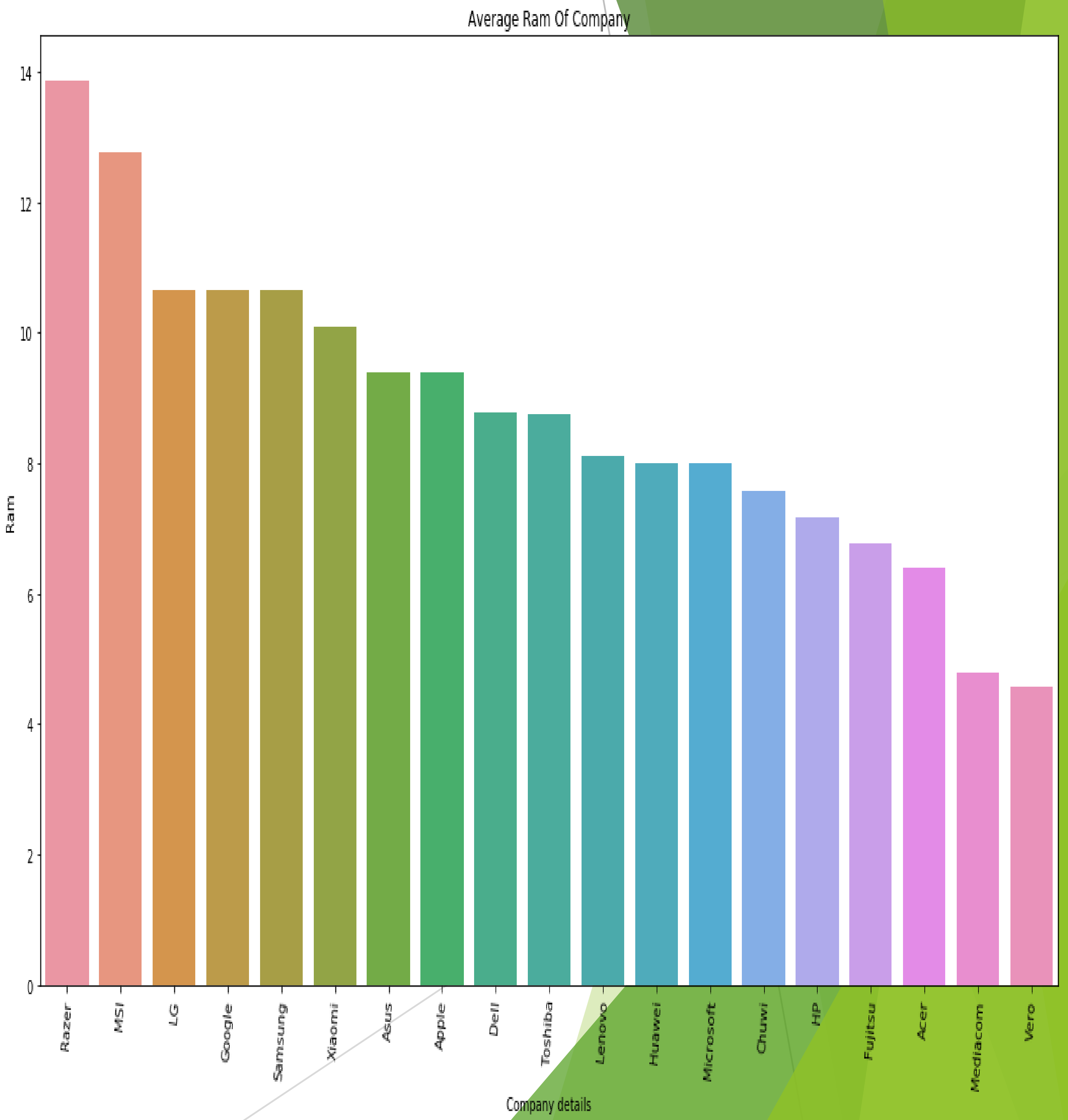
```
Company             0
Product             0
TypeName            0
Inches              0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
Price_euros         0
dtype: int64
```

Company and the RAM used

Before cleaning

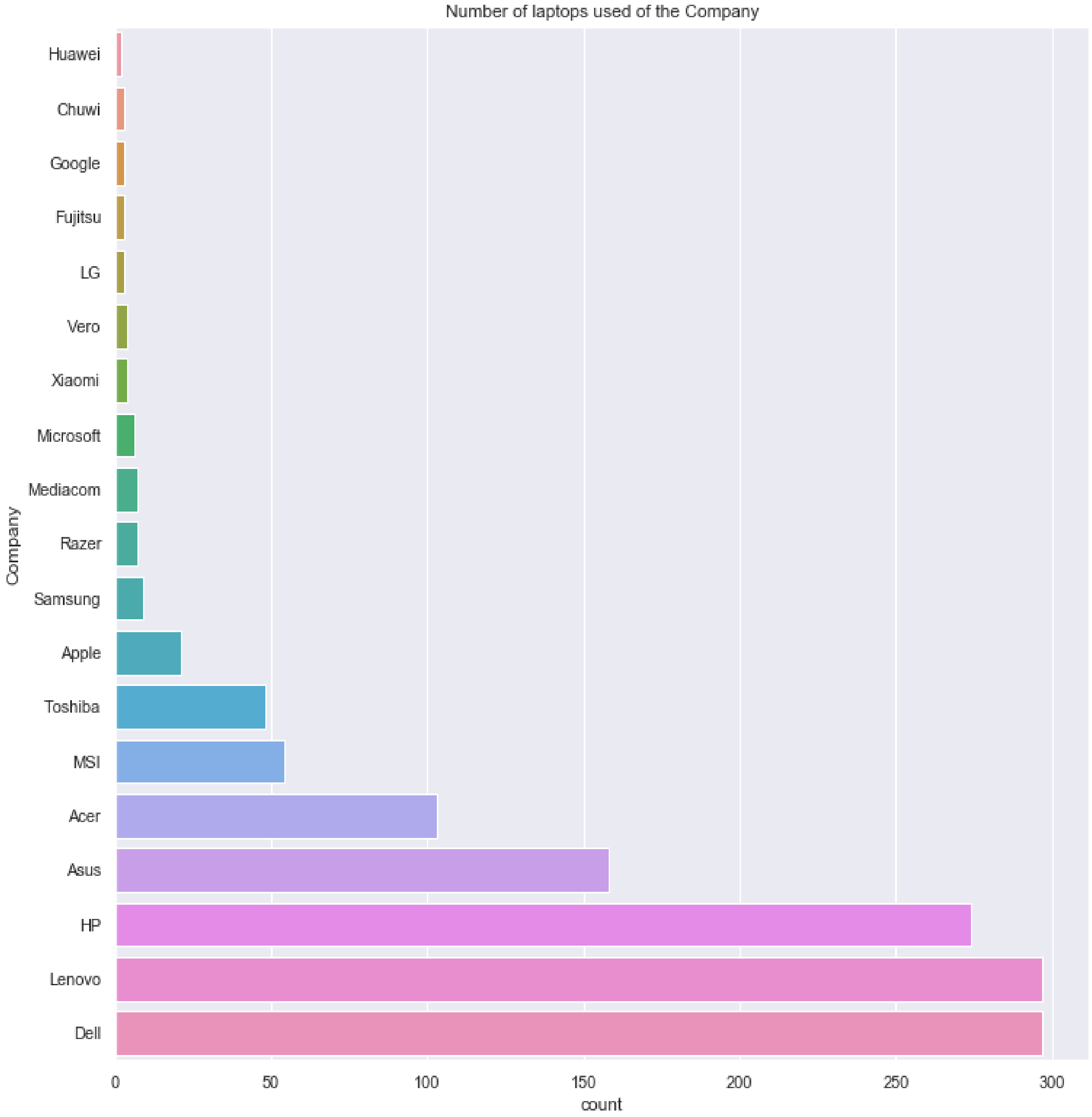


After cleaning

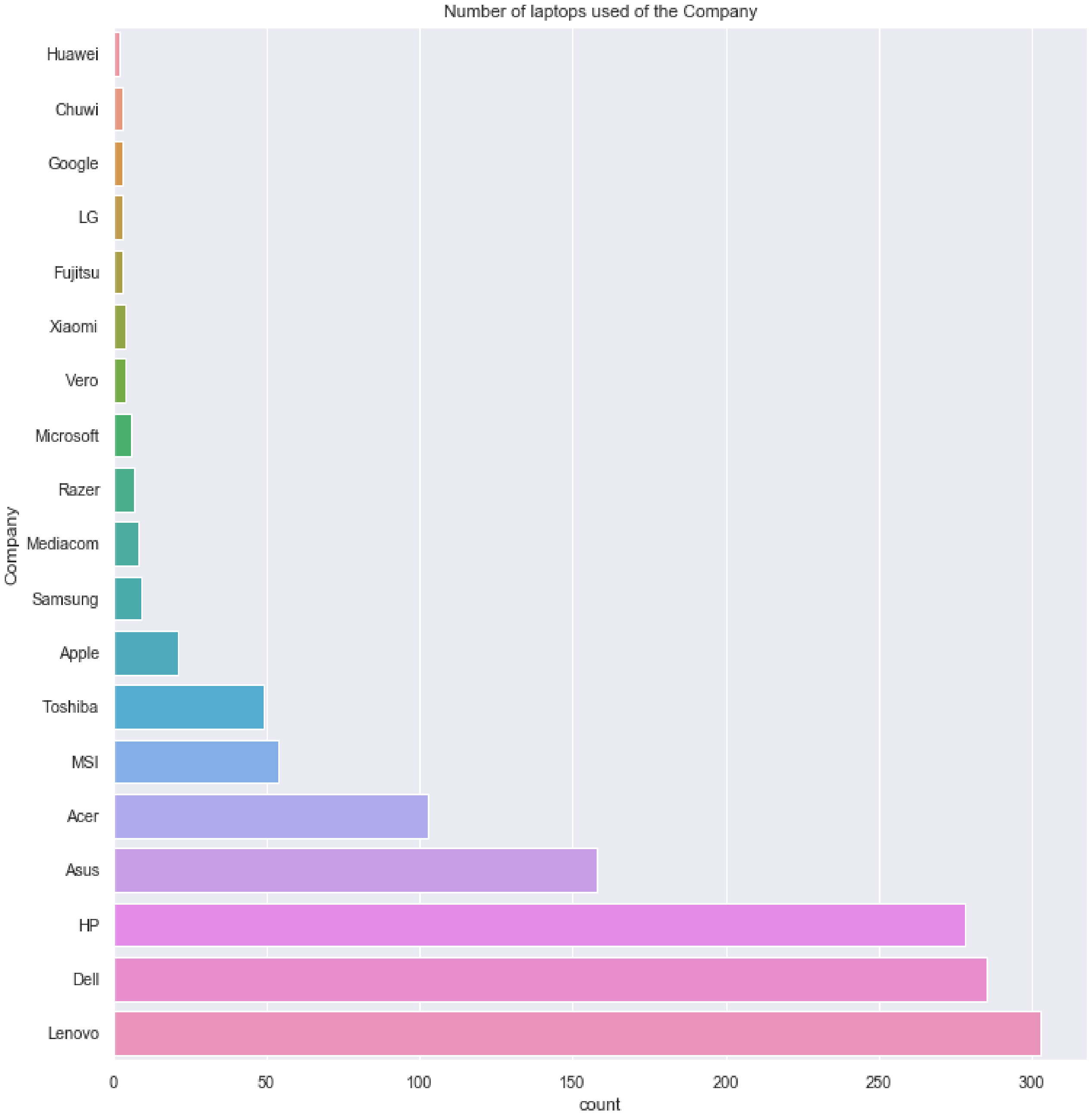


Number of laptops of a company

Before cleaning



After cleaning



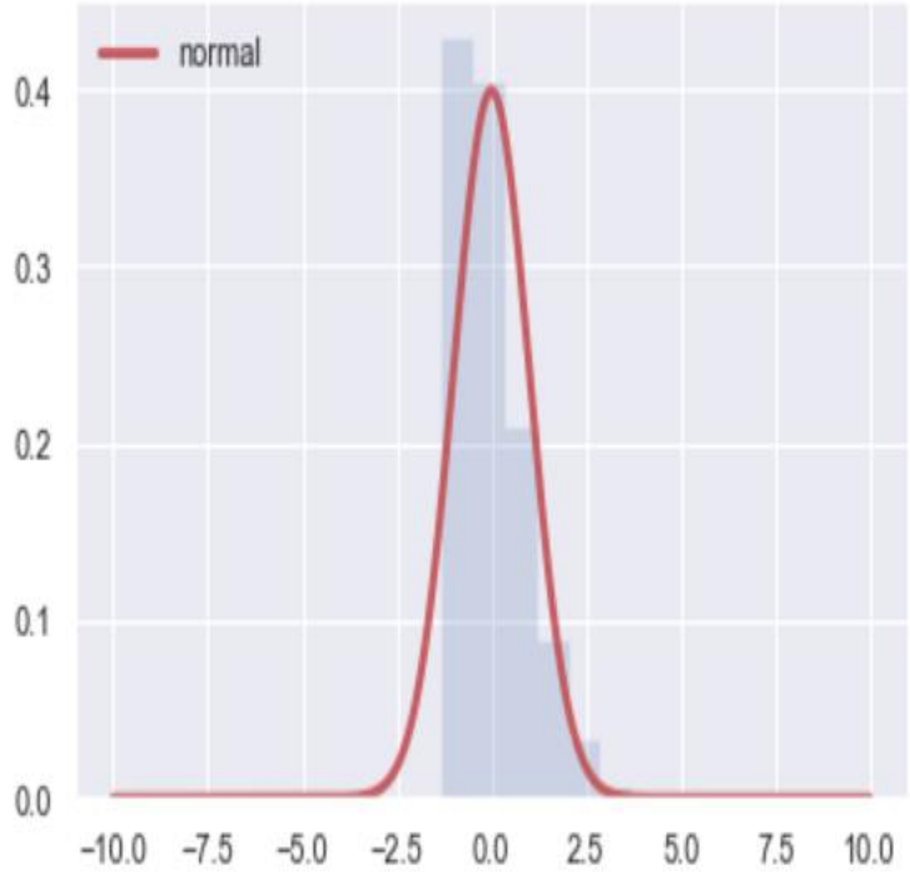
3. Normalization

Normalization usually means to scale a variable to have a values between 0 and 1

Is Should we normalize data?

- Normalizing data eliminates the units of measurement for data, enabling us to more easily compare data from different places.
- Normalization helps in reducing complexity of the data.
- It could eliminate outliers(if any)
- Normalizing will ensure that a convergence problem does not have a massive variance, making optimization feasible.

AFTER NORMALIZATION



Standardised mean

```
lines=std_std
fig, ax = plt.subplots(1, 1)
mean, var, skew, kurt = norm.stats(moments='mvsk')

#Here I delete some lines aimed to fill the list with values

Long = len(lines)
Maxim = max(lines) #MaxValue
Minim = min(lines) #MinValue
av = np.mean(lines) #Average
StDev = np.std(lines) #Standard Dev.

x = np.linspace(-10, +10, Long)
ax.plot(x, norm.pdf(x, av, StDev),'r-', lw=3, alpha=0.9, label='normal')

weights = np.ones_like(lines)/len(lines)

ax.hist(lines, weights = weights, density=True, histtype='stepfilled', alpha=0.2)
ax.legend(loc='best', frameon=False)
```

Standardization:

Standardization transforms data to have a mean of zero and a standard deviation of 1.

Before standardization:

```
print('Mean before standardization:\nRam={:.2f}, Price_euros={:.2f},Inches={:.2f},Weight={:.2f}'
      .format(sample['Ram'].mean(), sample['Price_euros'].mean(), sample['Inches'].mean(), sample['Weight'].mean()))
print('\nStandard deviation before standardization:\nRam={:.2f}, Price_euros={:.2f},Inches={:.2f},Weight={:.2f}'
      .format(sample['Ram'].std(), sample['Price_euros'].std(), sample['Inches'].std(), sample['Weight'].std()))
```

```
Mean before standardization:
Ram=8.35, Price_euros=1123.69,Inches=15.00,Weight=2.05

Standard deviation before standardization:
Ram=4.62, Price_euros=699.01,Inches=1.35,Weight=0.62
```

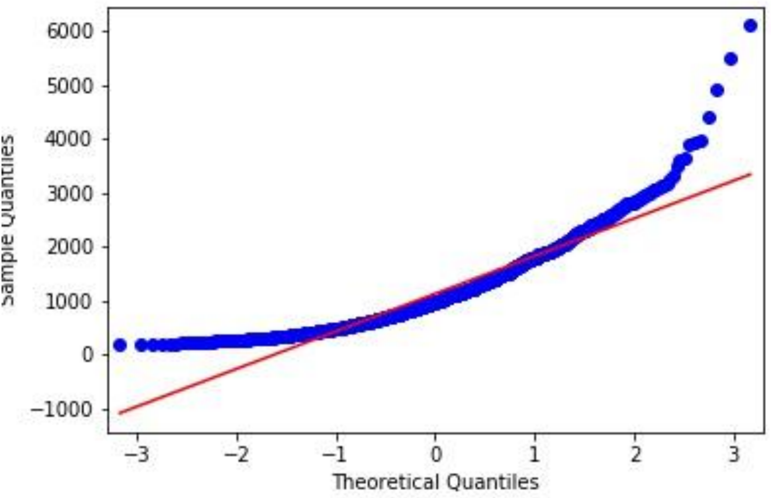
After standardization:

```
std_scale = preprocessing.StandardScaler().fit(sample[['Ram', 'Price_euros','Inches','Weight']])
df_std = std_scale.transform(sample[['Ram', 'Price_euros','Inches','Weight']])
```

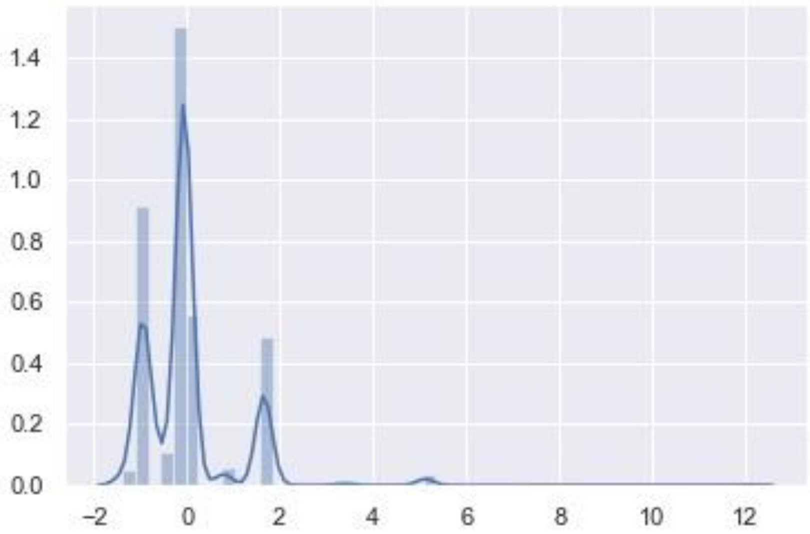
```
minmax_scale = preprocessing.MinMaxScaler().fit(df[['Ram', 'Price_euros','Inches','Weight']])
df_minmax = minmax_scale.transform(df[['Ram', 'Price_euros','Inches','Weight']])
```

```
print('Mean after standardization:\nRam={:.2f}, Price_euros={:.2f},Inches={:.2f},Weight={:.2f}'
      .format(df_std[:,0].mean(), df_std[:,1].mean(), df_std[:,2].mean(), df_std[:,3].mean()))
print('\nStandard deviation after standardization:\nRam={:.2f}, Price_euros={:.2f},Inches={:.2f},Weight={:.2f}'
      .format(df_std[:,0].std(), df_std[:,1].std(), df_std[:,2].std(), df_std[:,3].std()))
```

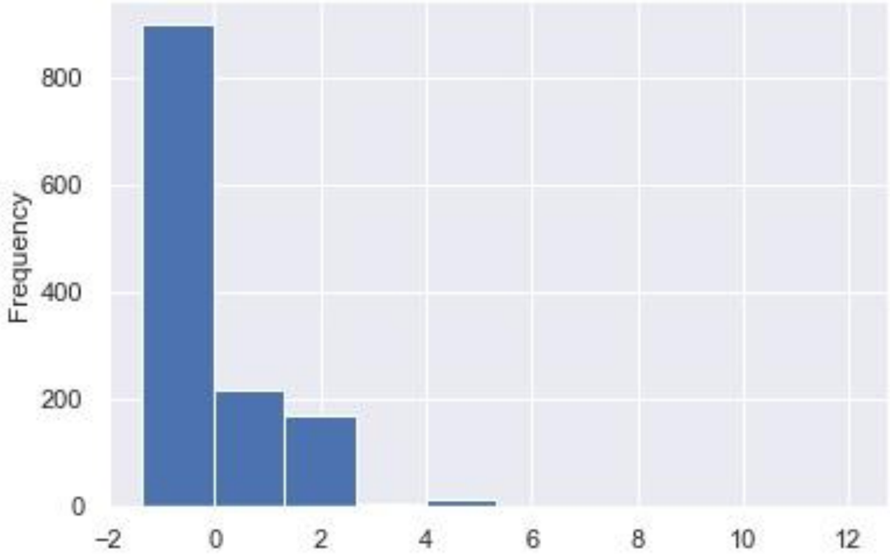
```
Mean after standardization:
Ram=0.00, Price_euros=0.00,Inches=0.00,Weight=0.00

Standard deviation after standardization:
Ram=1.00, Price_euros=1.00,Inches=1.00,Weight=1.00
```



←QQ plot of Price_euros
After standardization

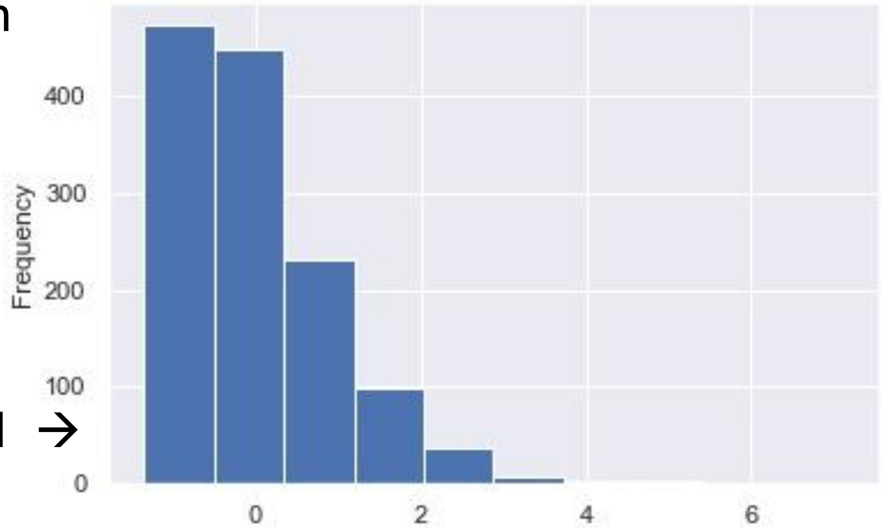


Distribution plot of standard→
Mean after standardization



←Histogram of standard mean
After standardization

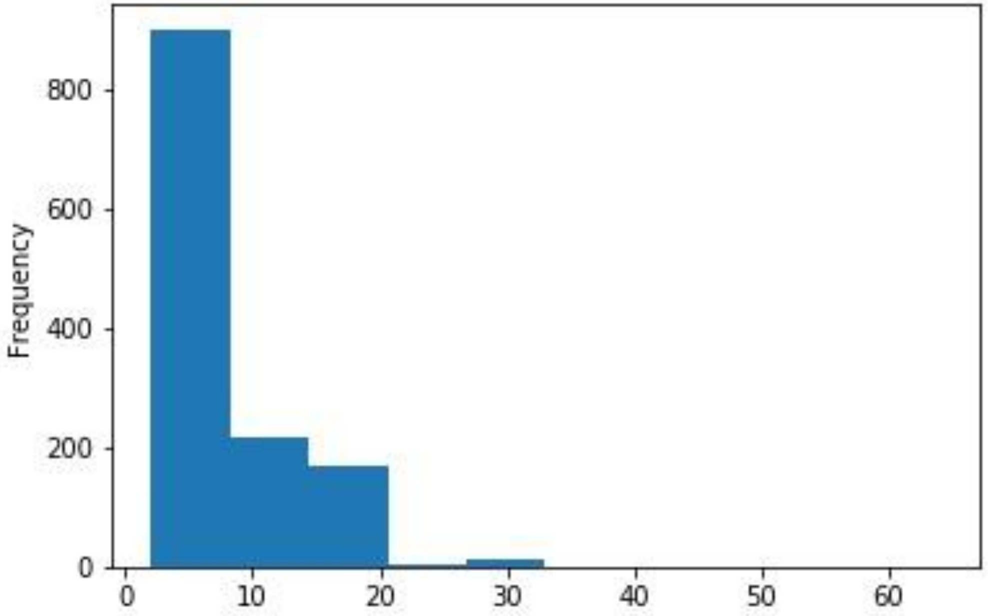
Distribution plot of standard →
Mean after standardization



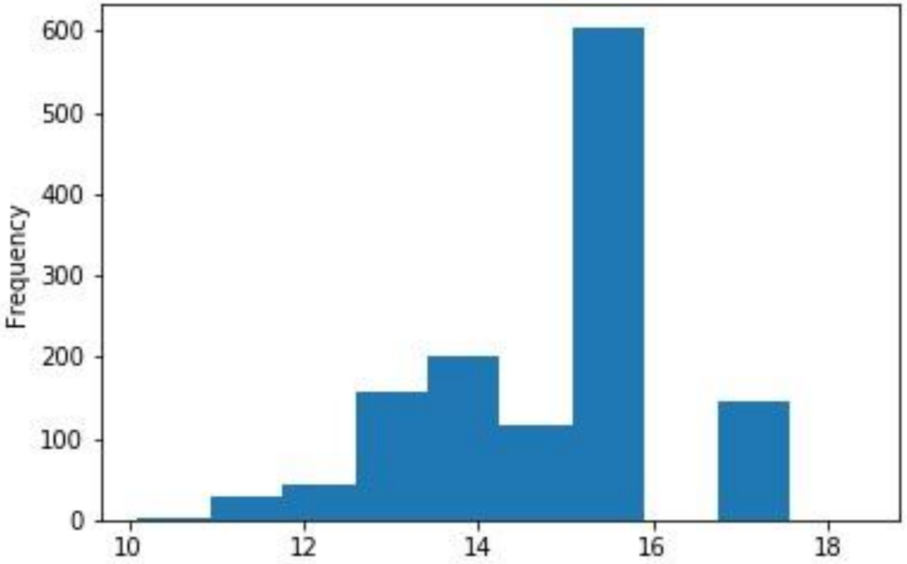
4. Graph visualization:

Histogram Plots

1.Ram of the laptops



2.Screen size in Inches of the laptops

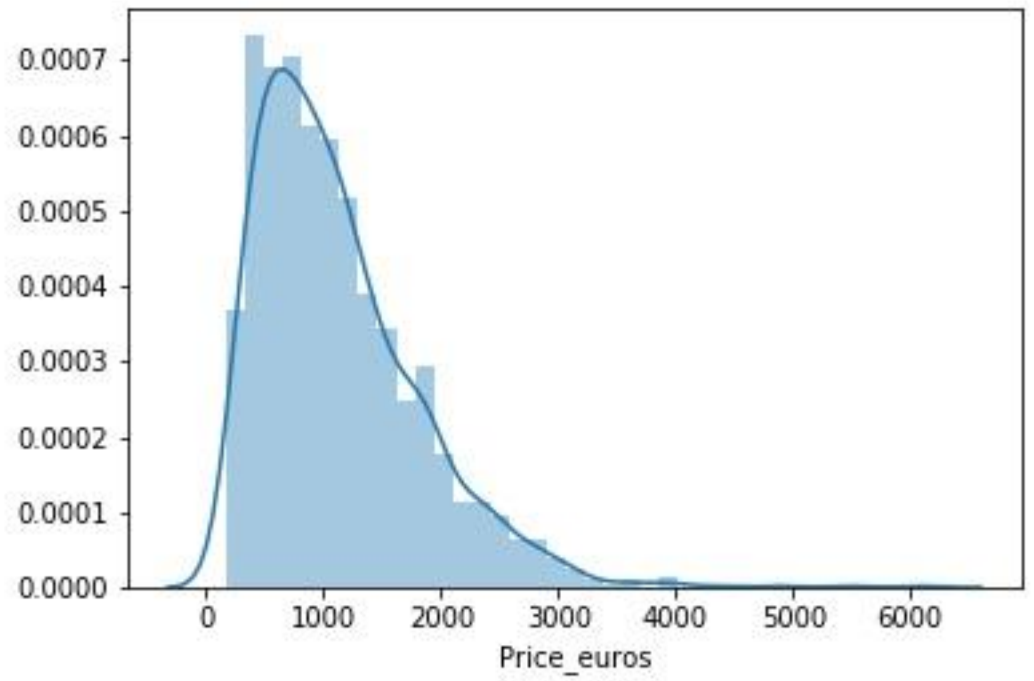


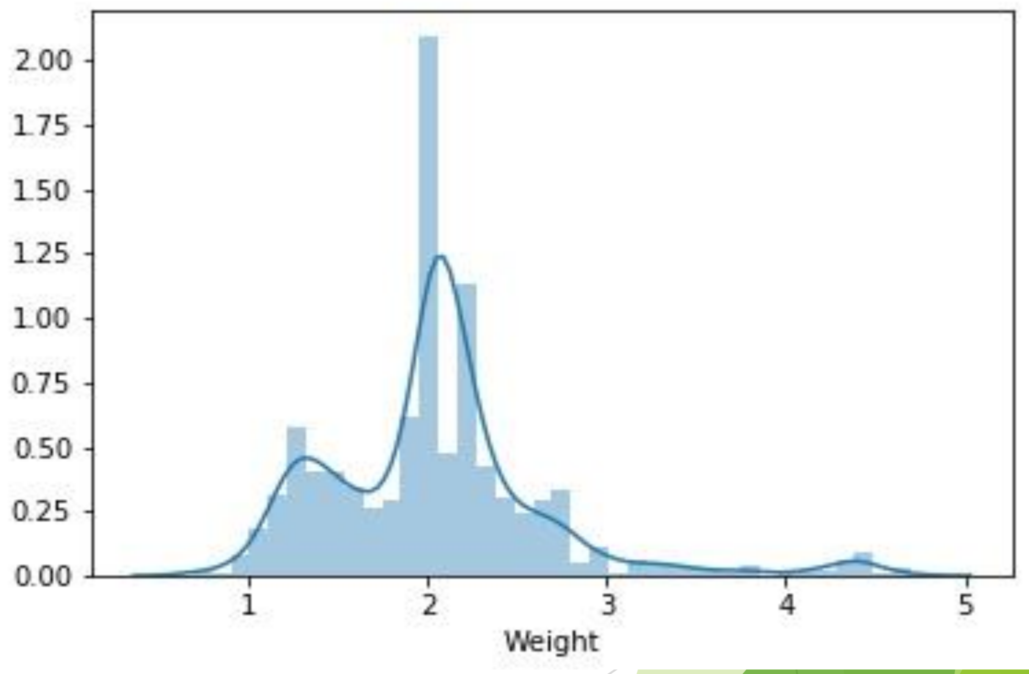
- Right skewed
- Mode < median <mean.

- Left skewed
- Mode > median >mean.

Distribution Plots

It tells us how normalised the graph is.

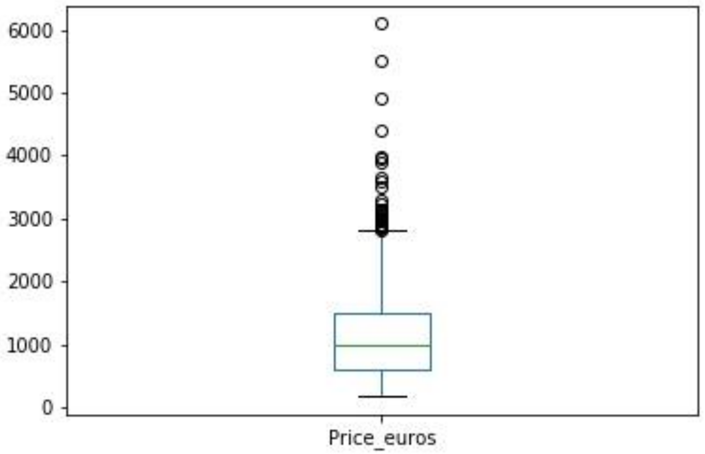




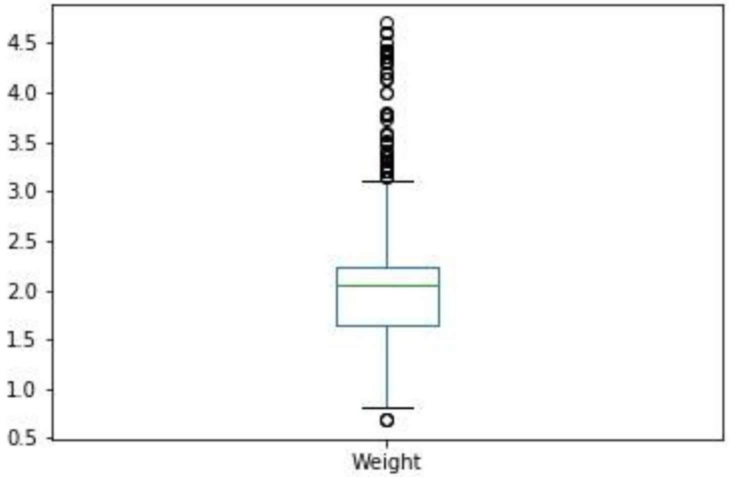
The Price_euros distribution is much more normalised than that of Weight distribution plot.

Box Plots

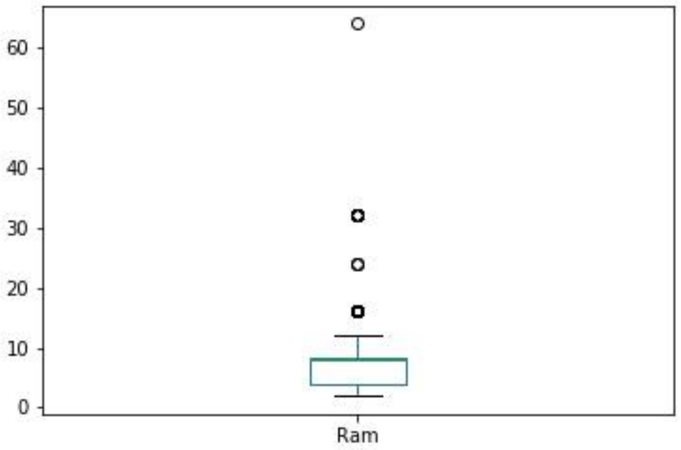
- Box plot tells about the outliers present in the data
- Weight column (in our data) contains a few number number of outliers



Price_euros box plot

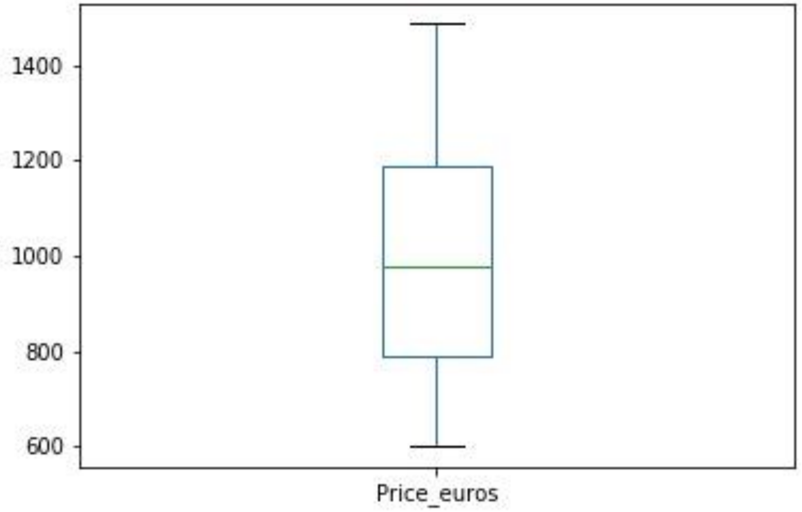


Weight box plot

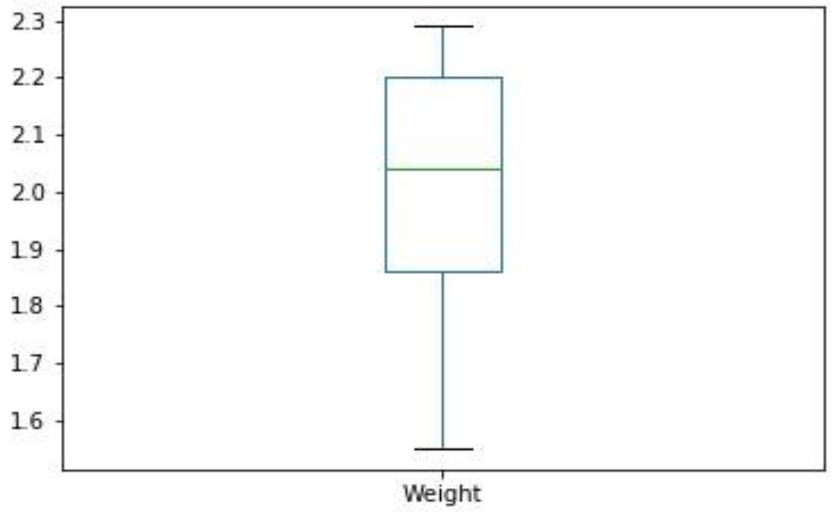


Ram box plot

Box Plots After Removing Outliers



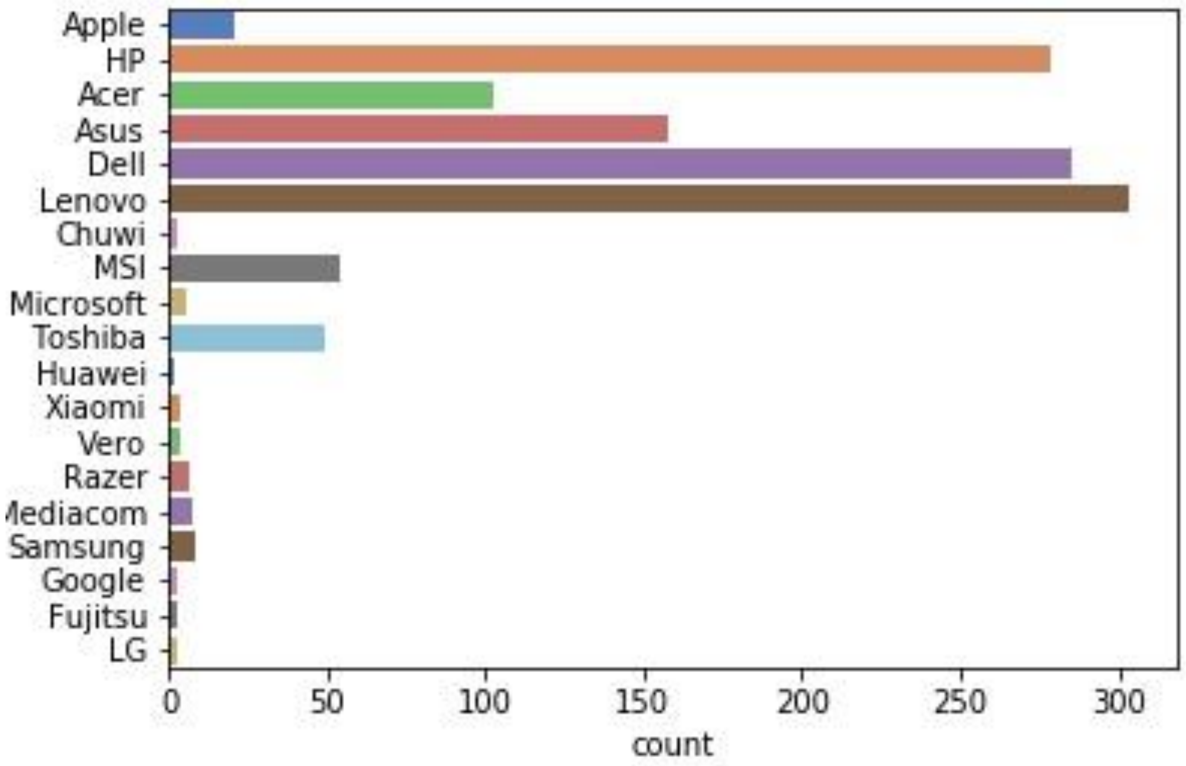
Price_euros box plot

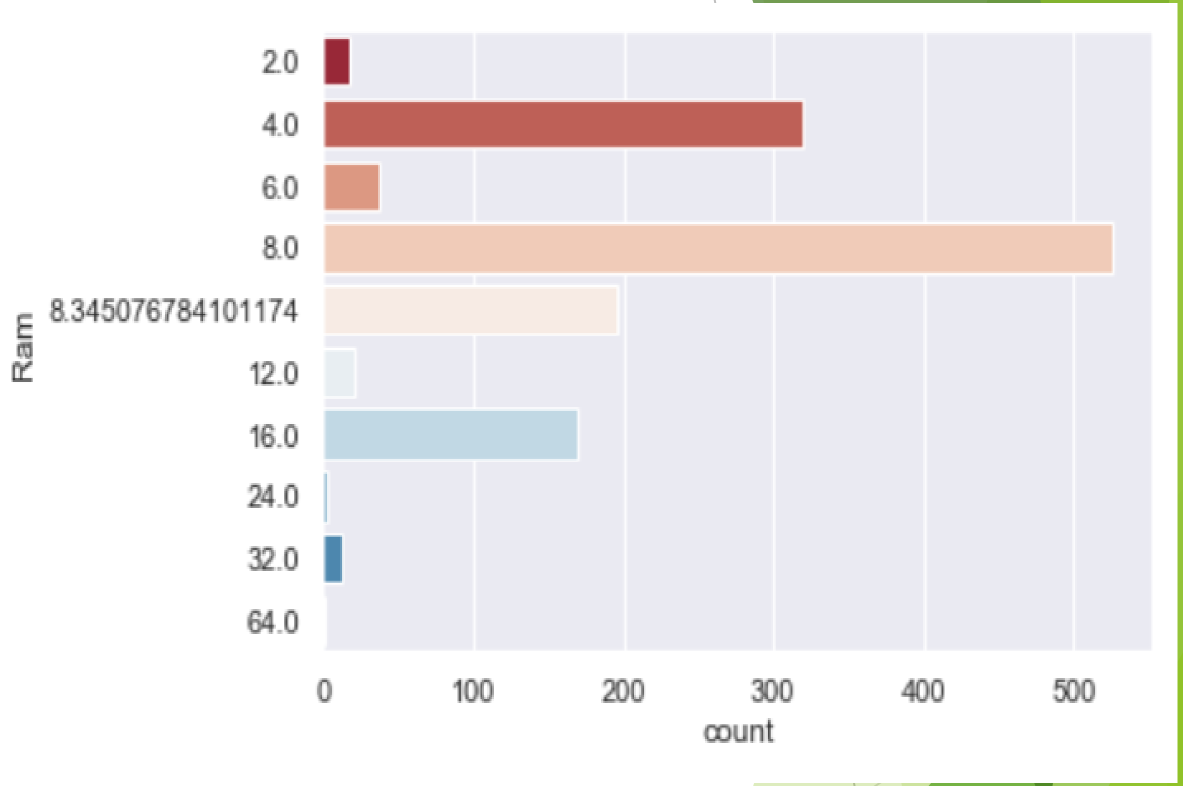


Weight box plot

```
res=sample.Weight.quantile([0.25,0.75])
true_index=(res.loc[0.25]<sample.Weight) & (sample.Weight<res.loc[0.75])
sample.Weight=sample.Weight[true_index]
sample['Weight'].fillna(method="bfill",inplace=True)
sample.fillna(sample.median(skipna=True),inplace=True)
sample
```

COUNTER PLOT

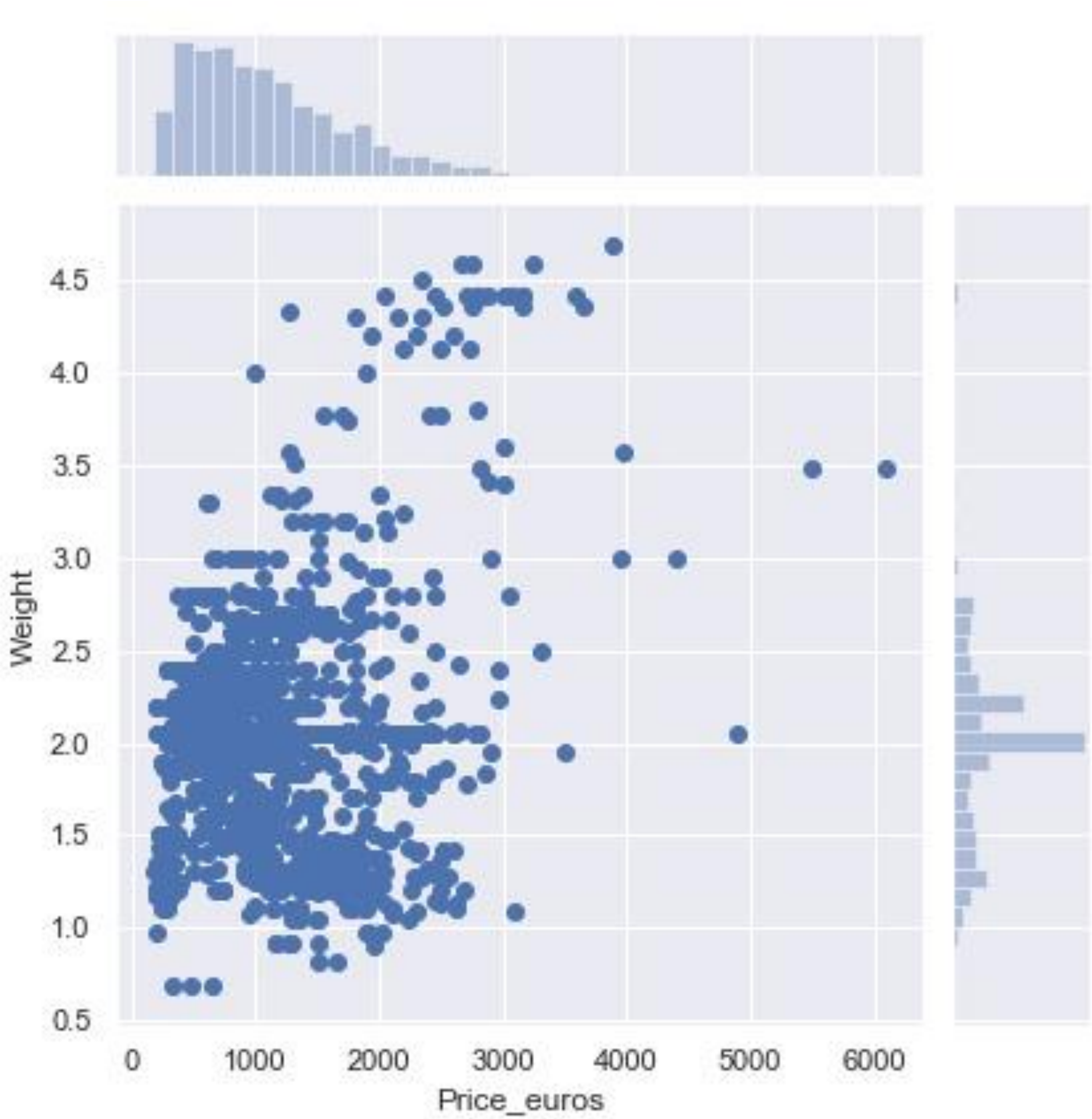




→Lenevo,Dell,HP are the most widely used laptops

→Laptops with 8 GB RAM are most actively used in laptops

Scatter Plots



Like distribution plots they also tell how normalized our data is ,unlike distribution scatter plots is for plot between two parameters(here Weight and Price_euros)

5. Hypothesis Testing:

Hypothesis testing is a statistical procedure that uses sample data to determine, whether a statement about the value of a population parameter should rejected or not.

Hypothesis testing is done for Ram column.
population mean is 8.345 so we assume that the sample mean is same as population mean

Null hypothesis =Ho: μ=8.345
Alternate hypothesis=Ha: μ!=8.345

Two sided hypothesis testing on Ram:

```
#hypothesis testing for ram
def two_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha):
    actual_z = abs(norm.ppf(alpha/2))
    hypo_z = (sample_mean - pop_mean) / (std_dev/sqrt(sample_size))
    print('actual z value :', actual_z)
    print('hypothesis z value :', hypo_z, '\n')
    if hypo_z >= actual_z or hypo_z <= -(actual_z): return True
    else: return False
alpha = 0.05
sample_mean = sam['Ram'].mean()
pop_mean = sample['Ram'].mean()
sample_size =  500
std_dev = sample['Ram'].std()
print('H0 : μ =', pop_mean)
print('H1 : μ !=', pop_mean)
print('alpha value is :', alpha, '\n')
reject = two_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha)
if reject: print('Reject NULL hypothesis')
else: print('Failed to reject NULL hypothesis')
```

```
H0 : μ = 8.345076784101165
H1 : μ != 8.345076784101165
alpha value is : 0.05

actual z value : 1.959963984540054
hypothesis z value : 0.8271225772458872

Failed to reject NULL hypothesis
```

Here hypothesis testing is done for ‘Weight’ column.

Null hypothesis =Ho: μ <=2.049

Alternate hypothesis=Ha: μ >2.049

```
def one_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha):
    actual_z = abs(norm.ppf(alpha))
    hypo_z = (sample_mean - pop_mean) / (std_dev/sqrt(sample_size))
    print('actual z value :', actual_z)
    print('hypothesis z value :', hypo_z, '\n')
    if hypo_z >= actual_z: return True
    else: return False
alpha = 0.05
sample_mean =sam['Weight'].mean()
pop_mean = sample['Weight'].mean()
sample_size =  500
std_dev =sample['Weight'].std()
print('H0 : μ <=', pop_mean)
print('H1 : μ >', pop_mean)
print('alpha value is :', alpha, '\n')
reject = one_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha)
if reject: print('Reject NULL hypothesis')
else: print('Failed to reject NULL hypothesis')
#variation with different parameters can be shown here
```

```
H0 : μ <= 2.049113728777479
H1 : μ > 2.049113728777479
alpha value is : 0.05

actual z value : 1.6448536269514729
hypothesis z value : 0.9137550256019928

Failed to reject NULL hypothesis
```

6. Correlation:

- It measures the strength and direction of a linear relationship between two variables
- r value always lies b/w -1 to 1

- A correlation of:
    - 1 indicates a perfect positive correlation.
    - -1 indicates a perfect negative correlation.
    - 0 indicates there is no relationship between the different variables.
    - Values between -1 and 1 denote the strength of the correlation.

```
print(df.corr(method='pearson'))
```

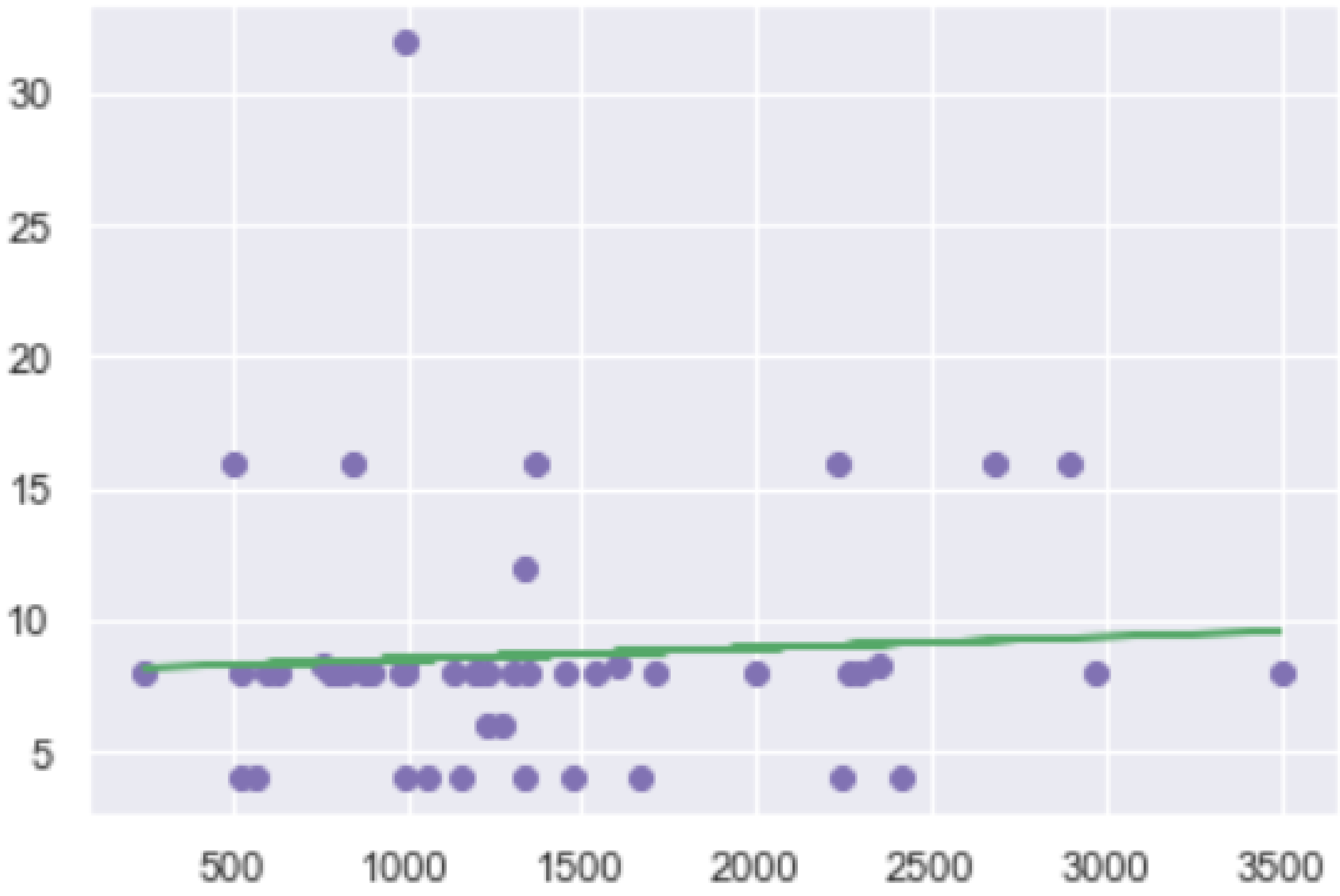
	Inches	Ram	Weight	Price_euros
Inches	1.000000	0.215903	0.824278	0.063014
Ram	0.215903	1.000000	0.364566	0.733122
Weight	0.824278	0.364566	1.000000	0.236923
Price_euros	0.063014	0.733122	0.236923	1.000000

```python
#linear regression
def linear_regression(x, y):
    x = [[i] for i in x]
    # Create linear regression object
    regr = linear_model.LinearRegression()

    # Train the model using the training sets
    regr.fit(x, y)

    #prediction
    y_preds = regr.predict(x)

    print('Coefficients: \n', regr.coef_)
    print("RMSE: %.2f" % RMSE(y, y_preds))

    plt.scatter(x, y,  color='m')
    plt.plot(x, y_preds, color='g')
    plt.show()

y = sample['Ram'].sample(50)
x = sample['Price_euros'].sample(50)
linear_regression(x, y)
```

Correlation between Ram and Price_euros .

```
Coefficients:
 [0.00042643]
RMSE: 22.35
```

We conclude by the plot that as Ram increase,the price of the laptop also increases

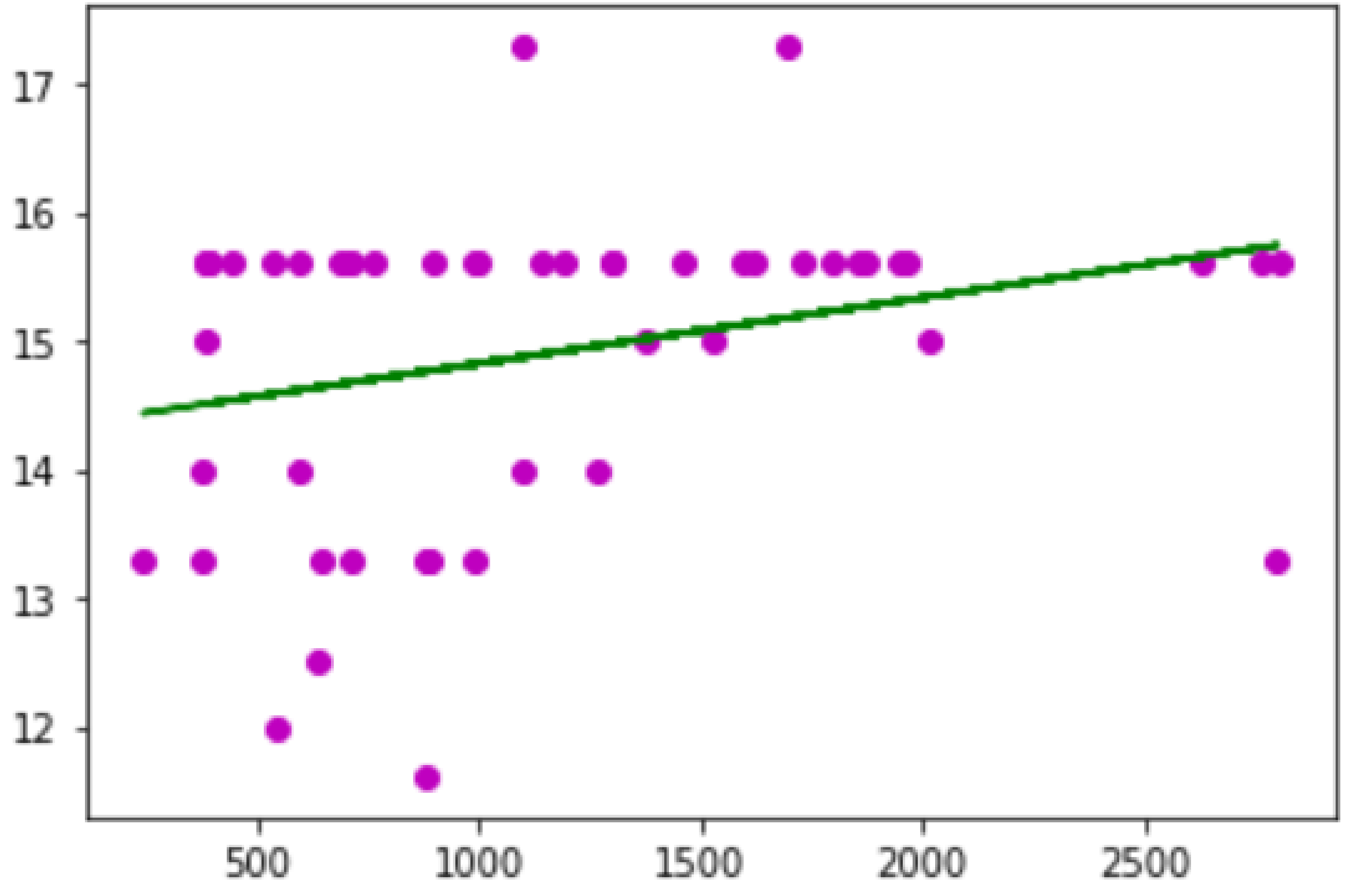


```python
#Linear regression
def linear_regression(x, y):
    x = [[i] for i in x]
    # Create linear regression object
    regr = linear_model.LinearRegression()

    # Train the model using the training sets
    regr.fit(x, y)

    #prediction
    y_preds = regr.predict(x)

    print('Coefficients: \n', regr.coef_)
    print("RMSE: %.2f" % RMSE(y, y_preds))

    plt.scatter(x, y,  color='m')
    plt.plot(x, y_preds, color='g')
    plt.show()

x = sample['Inches'].sample(50)
y = sample['Weight'].sample(50)
linear_regression(x, y)
```

Correlation between Inches (Screen size)Weight of the laptop.

```
Coefficients:
 [0.0005117]
RMSE: 1.37
```

We can say the number of laptops with screensize-Inches Within 15-16 and weight 1 -2.5 kg are the ones which are heavily used



$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1\overline{x}$$

Line Equation for dependency of Weight (x) and Inches(y)

$\hat{\beta}_1 =$ 3.3094

$\hat{\beta}_0 =$ 8.9354

$y = \hat{\beta}_1 x + \hat{\beta}_0$


THANK YOU