PREDICTING THE PRICES OF LAPTOPS
-Vision team-

INTRODUCTION

We make a project for Predicting prices of laptops, The problem statement is that if any user wants to buy a laptop, then our project should be compatible to provide a tentative price of laptop according to the user configurations. The field of the data set is computer Science and Business. It looks like a simple project or just developing a model, but the dataset we have is noisy and needs lots of feature engineering, and preprocessing. It is good that there are no NULL values. And we need little changes in some column to convert them to numeric by removing the unit written after value. So, we will perform data cleaning here to get the correct types of columns.

Database Columns:

1 Company- String -Laptop Manufacturer

2 Product -String -Brand and Model

3 TypeName -String -Type (Notebook, Ultrabook, Gaming, etc.)

4 Inches -Numeric- Screen Size

5 ScreenResolution -String- Screen Resolution

6 Cpu- String -Central Processing Unit (CPU)

7 Ram -String- Laptop RAM

8 Memory -String- Hard Disk / SSD Memory

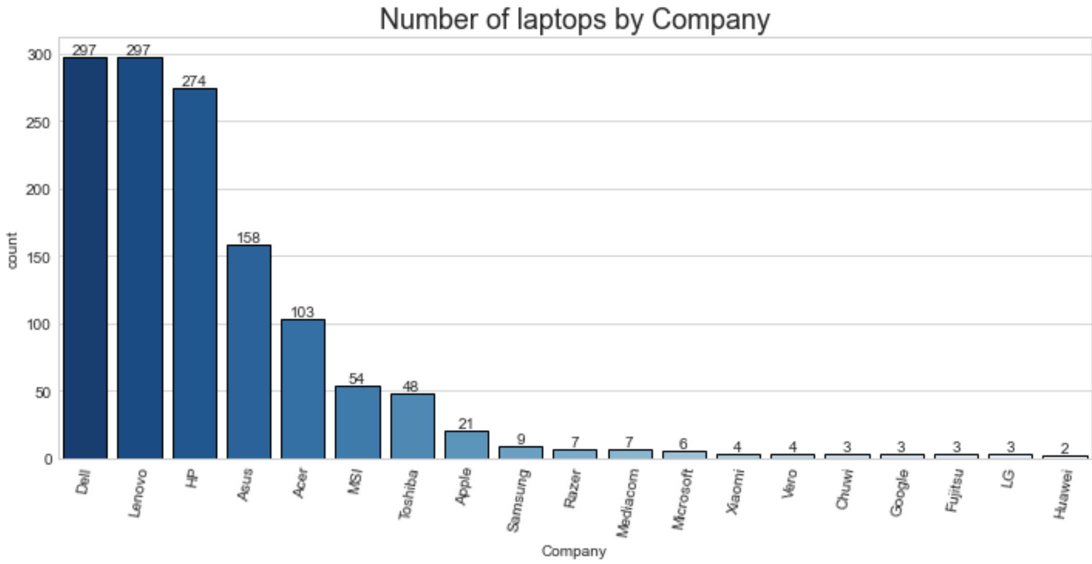
9 GPU -String- Graphics Processing Units (GPU)

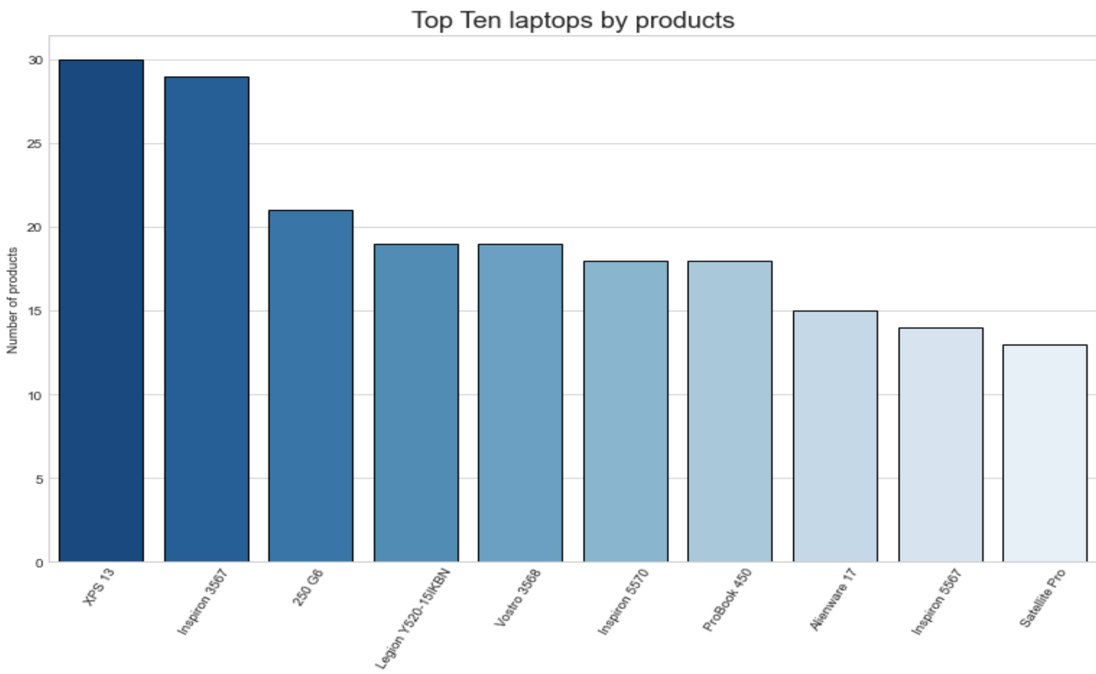
10 OpSys -String- Operating System

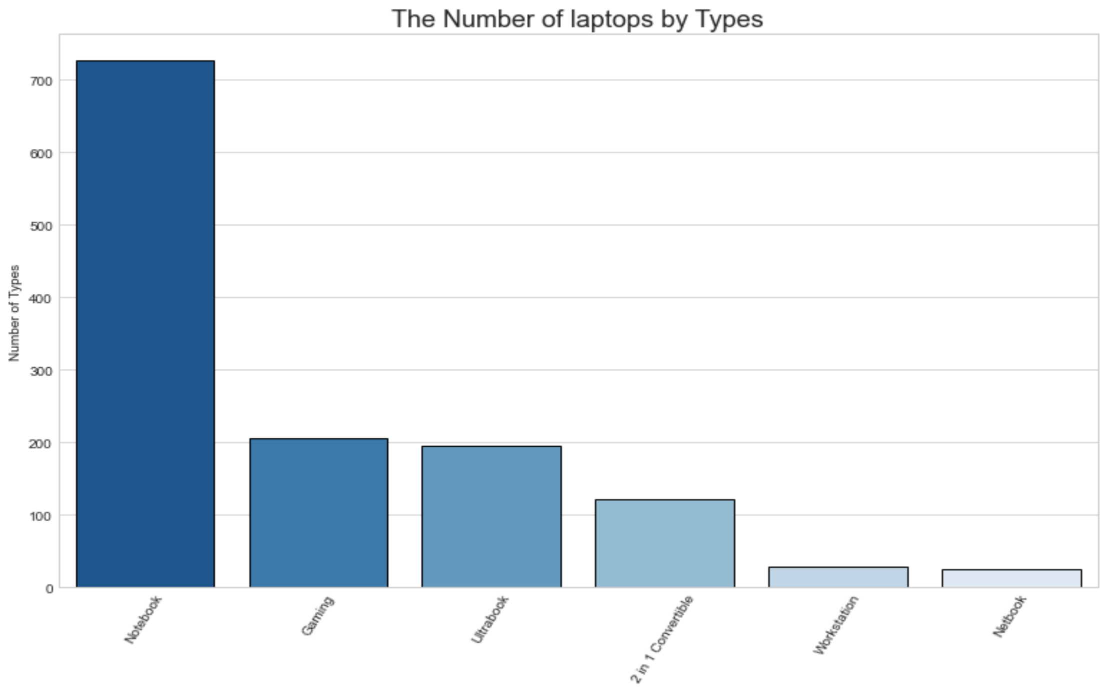
11 Weight -String- Laptop Weight

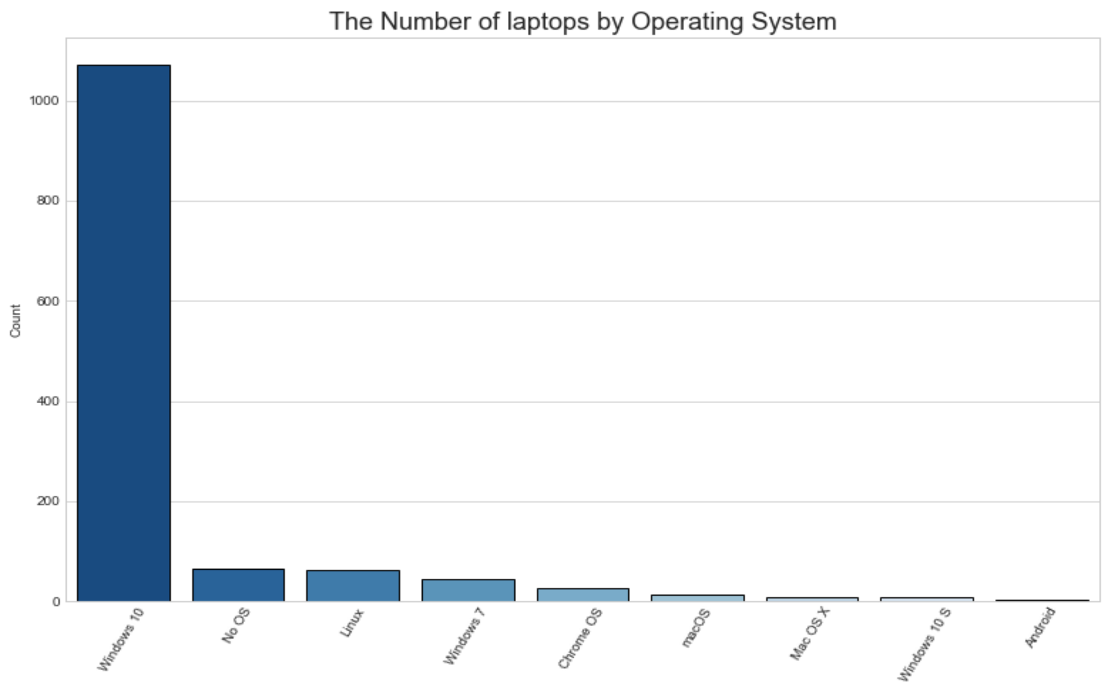
12 Price_euros -Numeric- Price (Euro)

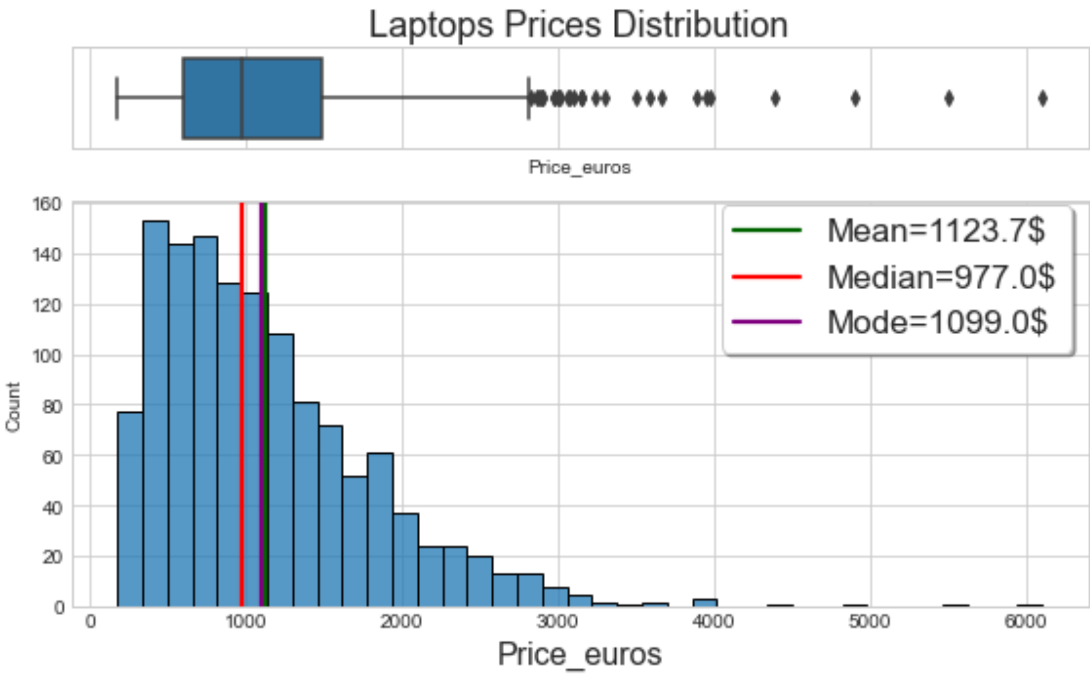
DATA VISUALIZATION
Univariate Feature Visualization



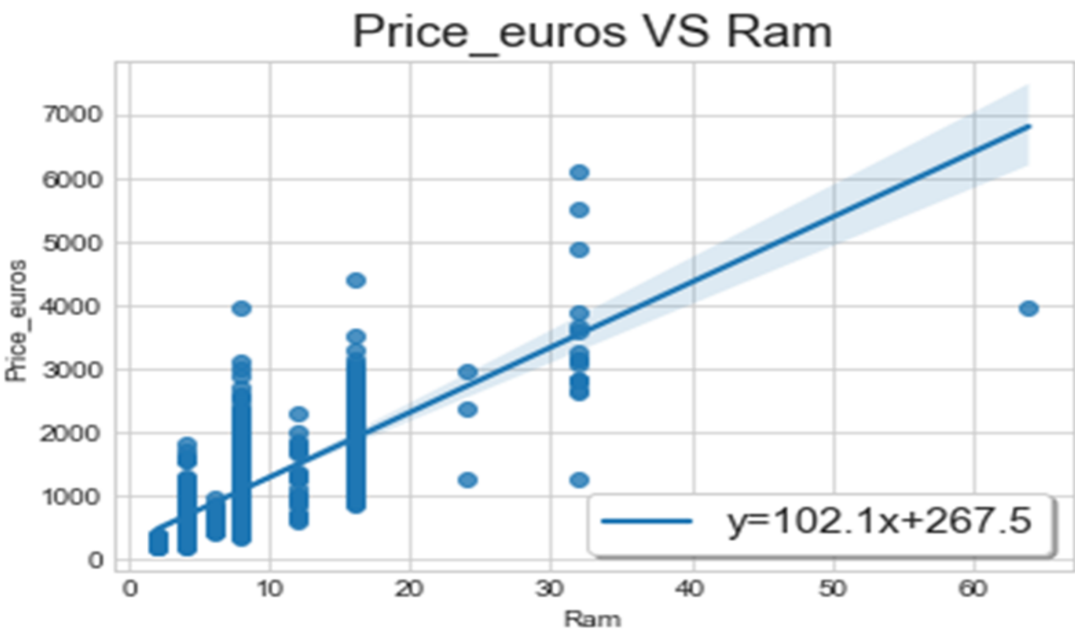



The Number of laptops by Types
Number of Types
0
100
200
300
400
500
600
700
Notebook
Gaming
Ultrabook
2 in 1 Convertible
Workstation
Netbook

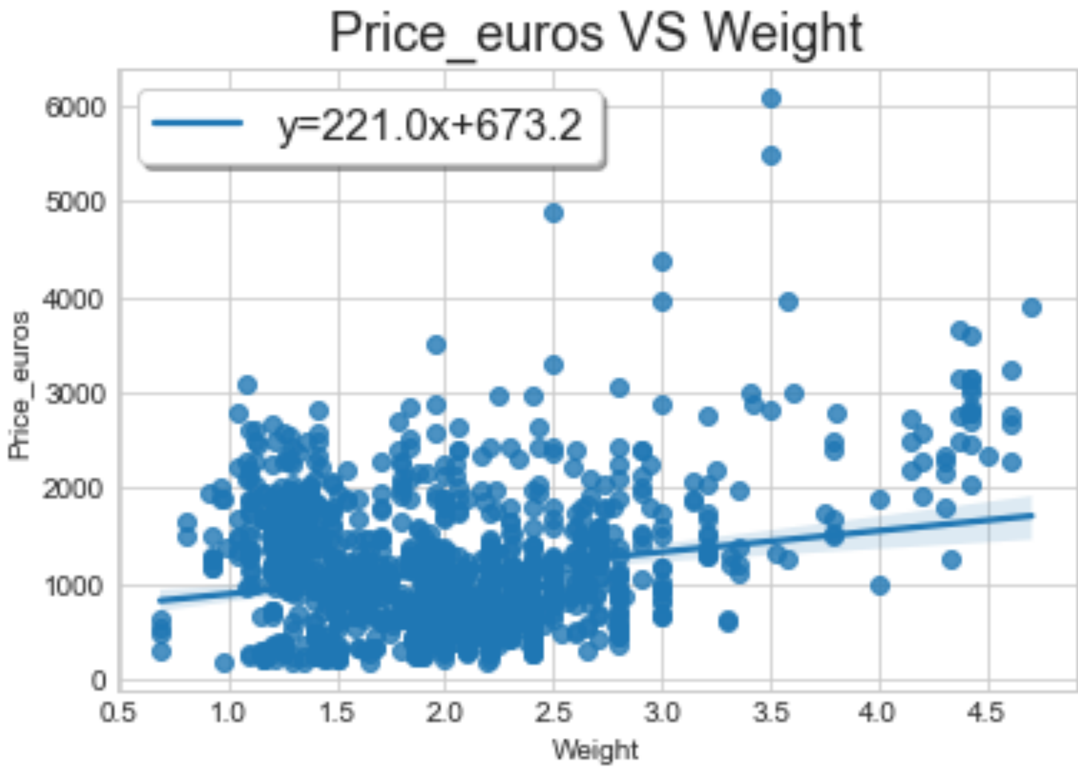

The Number of laptops by Operating System
Count
0
200
400
600
800
1000
Windows 10
No OS
Linux
Windows 7
Chrome OS
macOS
Mac OS X
Windows 10 S
Android

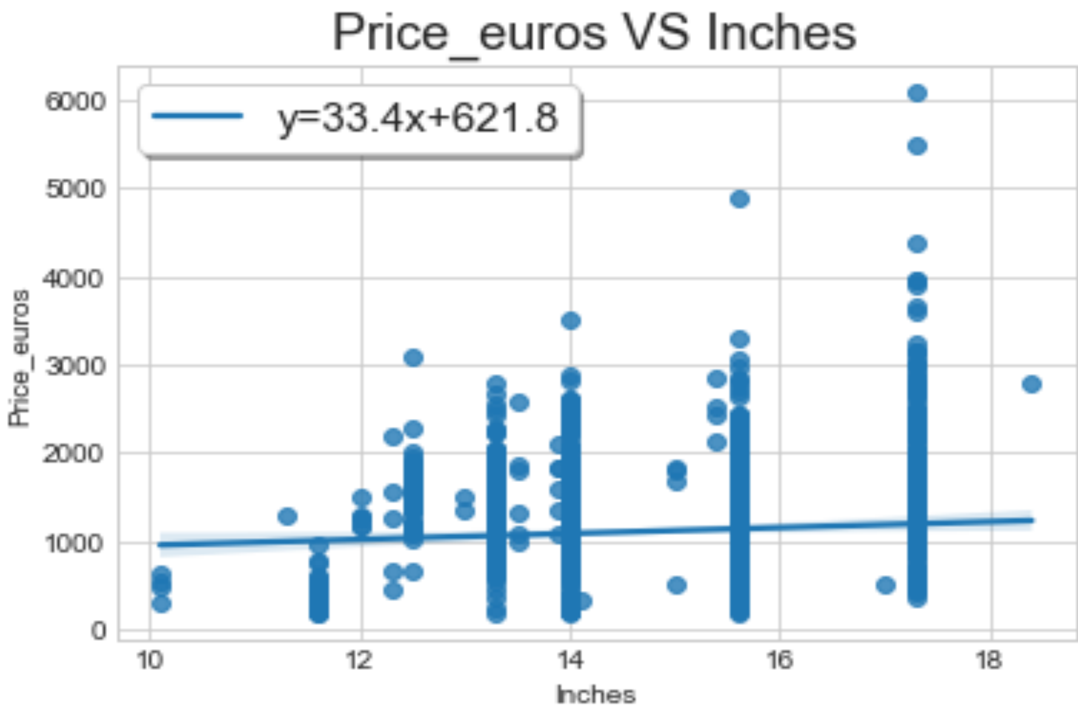


Laptops Prices Distribution
Price_euros
Count
Mean=1123.7$
Median=977.0$
Mode=1099.0$
0
20
40
60
80
100
120
140
160
1000
2000
3000
4000
5000
6000
Price_euros



Price_euros VS Ram
Price_euros
0
1000
2000
3000
4000
5000
6000
7000
10
20
30
40
50
60
y=102.1x+267.5
Ram

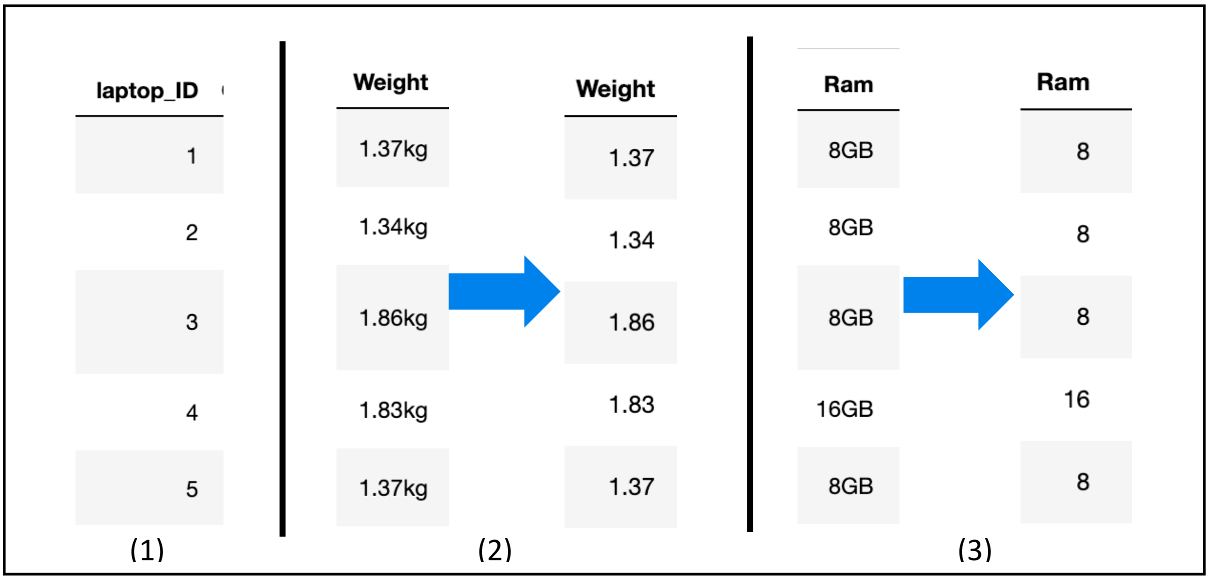

Price_euros VS Weight
y=221.0x+673.2
Price_euros
Weight
0
1000
2000
3000
4000
5000
6000
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5


Price_euros VS Inches
y=33.4x+621.8
Price_euros
Inches
0
1000
2000
3000
4000
5000
6000
10
12
14
16
18

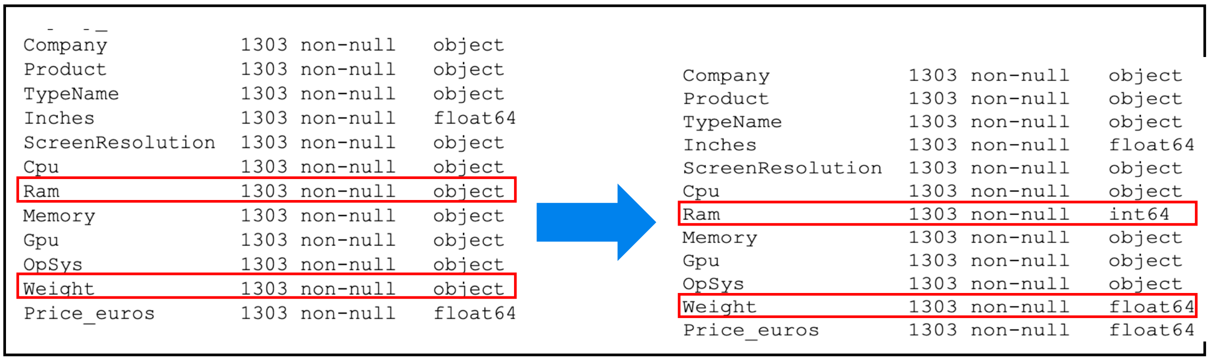
PREPROCESSING AND PREPARING DATA FOR MODELING

We preprocessed data using Jupyter Notebook throw the following steps:

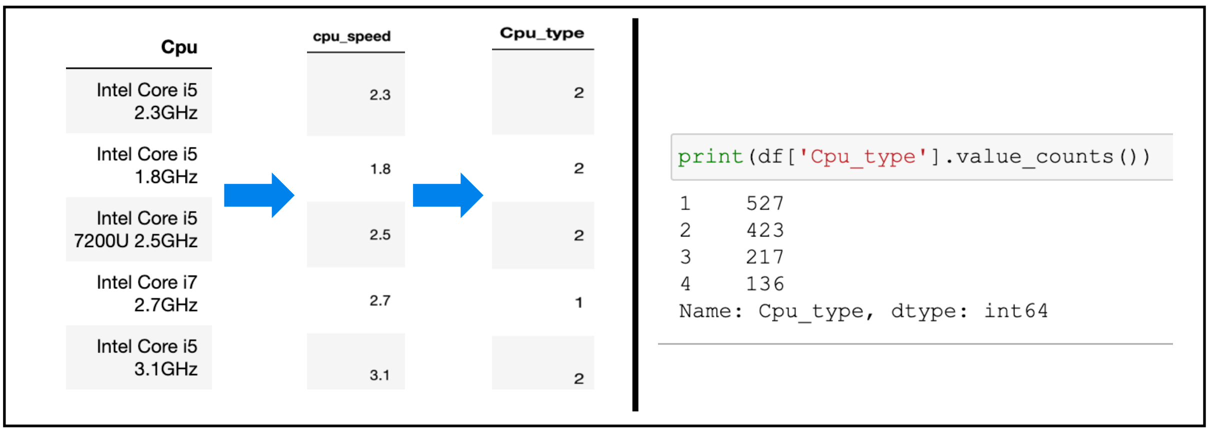
1. Drop laptop_ID column because no need in this model after extracting useful data.
2. Delete GB symbol from RAM column to change their data type.
3. Delete kg symbol from Weight column to change their data.



4. change data type of RAM and Weight columns to integer and float to handle it correctly.



5. Add new column CPU_speed extracting from column CPU.
6. Remove GHz from CPU_speed and change datatype to float.
7. Add new column CPU_type extracting from column CPU and ordering from 1 to 4 in ascending order.



8. Drop CPU column after because no need in this model after extracting useful data.

9. Add new column Touch_screen extracting from column ScreenResolution, and change the data type to integer.

10. Add new column HD_type extracting from column ScreenResolution, change the data type to integer.

```
1    843
2    430
3     30
Name: HD_type, dtype: int64
```

	Company	Product	TypeName	Inches	ScreenResolution	Ram	Memory	Gpu	OpSys	Weight	Price_euros	cpu_speed	Cpu_type	Touch_screen	HD_type
0	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	1339.69	2.3	2	0	2
1	Apple	Macbook Air	Ultrabook	13.3	1440x900	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	898.94	1.8	2	0	2

11. Add new column Resolution_type extracting from column ScreenResolution.

```
1920x1080    841
1366x768     308
3840x2160     43
3200x1800     27
2560x1440     23
1600x900      23
2560x1600      6
2304x1440      6
2256x1504      6
1920x1200      5
1440x900       4
2880x1800      4
2400x1600      4
2160x1440      2
2736x1824      1
Name: Resolution_type, dtype: int64
```

12. Add new column IPS extracting from column ScreenResolution.

Ram	Memory	Gpu	OpSys	Weight	Price_euros	cpu_speed	Cpu_type	Touch_screen	HD_type	Resolution_type	IPS
8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	1339.69	2.3	2	0	2	2560x1600	1
8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	898.94	1.8	2	0	2	1440x900	0

13. Drop ScreenResolution column because no need in this model after extracting useful data.

14. Add new column Memory_type extracting from column Memory, and change data type to integer.

```
1    843
2    375
3     75
4     10
Name: Memory_type, dtype: int64
```

15. Drop Memory column because no need in this model after extracting useful data.

16. Add new column OpSys extracting from column Op_Sys, change data type to integer.

```
1    1125
2      95
3      62
4      21
Name: Op_Sys, dtype: int64
```

17. Drop OpSys column because no need in this model after extracting useful data.
18. Convert all categorical columns to one hot encoding.

```
['Company', 'Gpu', 'Product', 'Resolution_type', 'TypeName']
```

19. Remove the nan row.
20. Divide data to input x and output y
21. Data scaling, and convert y to categorical values

The database after prepared:

	index	Company	Product	TypeName	Inches	Ram	Gpu	Weight	cpu_speed	Cpu_type	Touch_screen	HD_type	Resolution_type	IPS	Memory_type	Op_Sys
0	0	1	300	4	13.3	8	58	1.37	2.3	2	0	2	10	1	1	4
1	1	1	301	4	13.3	8	51	1.34	1.8	2	0	2	1	0	3	4
2	2	7	50	3	15.6	8	53	1.86	2.5	2	0	1	3	0	1	2
3	3	1	300	4	15.4	16	9	1.83	2.7	1	0	2	12	1	1	4
4	4	1	300	4	13.3	8	59	1.37	3.1	2	0	2	10	1	1	4
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

PIPELINE PREDICTING PRICES OF LAPTOPS

Modelling

At a time we do not know which is the best regressor we tried some important algorithms as following:

- LogisticRegression model
- KNeighborsRegressor model
- DecisionTreeRegressor model
- SVR model
- XGBRegressor model

Split in train and test test

```
#Splitting features/target and train/test data
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y_transformed,test_size=.30 ,random_state=2,shuffle=True)
```

Implement Pipeline

Making pipelins

```
#LogisticRegression model
pipeline_LogisticReg=Pipeline([('scalar1',StandardScaler()),
                     ('pca1',PCA(n_components=2)),
                     ('lr_reg',LogisticRegression(random_state=10))])
```

```
#KNeighborsRegressor model
pipeline_KNR=Pipeline([('scalar2',StandardScaler()),
                     ('pca2',PCA(n_components=2)),
                     ('KNR_reg',KNeighborsRegressor())])
```

```
#DecisionTreeRegressor model
pipeline_DecisionTreeReg=Pipeline([('scalar3',StandardScaler()),
                     ('pca3',PCA(n_components=2)),
                     ('DTR_reg',DecisionTreeRegressor())])
```

```
#SVR model
pipeline_SVR=Pipeline([('scalar4',StandardScaler()),
                     ('pca4',PCA(n_components=2)),
                     ('SVR_reg',SVR(kernel = 'rbf'))])
```

```
#XGBRegressor model
pipeline_XGBReg=Pipeline([('scalar5',StandardScaler()),
                     ('pca5',PCA(n_components=2)),
                     ('XGB_reg',XGBRegressor())])
```

Fit the Pipelines

```
# Fit the pipelines
for pipe in pipelines:
    pipe.fit(x_train, y_train)
```

Model evaluation

```
#Models evaluation
for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}".format(pipe_dict[i],model.score(x_test,
```

```
Logistic Regression Test Accuracy: 0.0273224043715847
KNeighborsRegressor Test Accuracy: 0.6799334563123657
DecisionTreeRegressor Test Accuracy: 0.564572812687361
SVR Test Accuracy: 0.4223231757860818
XGBRegressor Test Accuracy: 0.6551795557018605
```

```

for i,model in enumerate(pipelines):
    if model.score(x_test,y_test)>best_accuracy:
        best_accuracy=model.score(x_test,y_test)
        best_pipeline=model
        best_classifier=i
print('Regressor with best accuracy:{}'.format(pipe_dict[best_classifier
```

```
Regressor with best accuracy:KNeighborsRegressor
```