



Fundamentals of Data Mining - IT3051

Group Project – G05

Final Report

IT Number	Student Name
IT20165352	Peiris M.I.M
IT20074340	Nirmal M.D.S
IT20101510	Alwis W.C.H
IT20619558	Samarakkody G.K.G.D.A
IT20201364	Thathsarani R.P.H.S.R

Date of Submit

02/11/2022

Table of Contents

Background.....	4
Target and business Goals	5
Choice of Technology	6
Methodology	7
Figure1 – workflow of model building	7
Data set Introduction	8
Figure2 - Data set details	8
Figure 3- Missing values	8
Figure 4 – Duplicate Rows	8
Data preprocessing and transformation	9
Figure 5	9
Figure 6	9
Figure 7	9
Figure 8	10
Figure 9	10
Figure 10.....	11
Figure 12.....	11
Figure 11.....	11
Figure 13.....	11
Figure 14.....	12
Figure 16.....	12
Figure 15.....	12
Figure 17.....	13
Figure 18.....	13
Figure 19.....	14
Figure 20.....	14
Figure 21.....	14
Figure 22.....	15
Figure 24 - Comparison of how price vary according to the newly created values.....	15
Figure 23- Comparison count of newly created column values.....	15
Figure 25.....	15
Figure 26.....	16
Figure 27.....	16
Figure 28.....	16
Figure 29.....	17

Figure 31.....	17
Figure 32.....	18
Figure 33 – How price vary with newly created GPU values	18
Figure 34.....	18
Figure 35.....	19
Figure 36.....	19
Figure 37- how price varies with new column 'os'	19
Figure 38 – Heat Map for all the columns(varies of correlation between each variable) 20	
Figure 39.....	20
Figure 40.....	20
Implementation of models	21
Figure 41.....	22
Figure 42.....	23
Figure 43.....	23
Figure 44.....	24
Figure 45.....	24
Figure 46.....	25
Figure 47.....	26
Implementation of User Interface	27
Figure 48.....	27
Deliverables	28
References	28

Background

For the following assignment, the group will be focusing on a problem which is faced by many people who are engaging in their day-to-day business ,academic , and professional work with the use of Technology Infrastructure. With the rising component costs, increased supply chain shortages, logistic costs resulting from the pandemic and the energy crisis all over the world. The unusual hike of the prices in the prevailing technology market has proven to be a difficult time for the information technology workforce and the students in the information technology industry in choosing the laptop which matches their requirement as well as fitting their budget. Various models of computers are sold in the market at different prices. The price of the computer will be mainly based on some key features like RAM, CPU, GPU, memory ,screen size and resolution, type name, operating system, and brand(company) of the laptop. As a solution for this business problem, a group of Data Science students has decided to build a web application that integrated by a machine learning model which will be able to make a prediction of laptops' prices based on their key features. Web application will facilitate people in choosing the components they need , and they could obtain an idea about what will be the expected price of the laptop they want. And see whether that laptop fits their budget. And if not, they are able to change or adjust the requirements freely and find the laptop which matches exactly to their budget and the requirements they wish to have. Not only predicting the price of laptop but also the proposed solution will allow users to choose their computer from anywhere, and in few minutes effortlessly. To conclude , as data science team our endeavor is to obtaining high accuracy for “Laptop Price Prediction Model” , thereby contribute to information technology sector to offer better solution for above mentioned term problem.

Target and business Goals

The main objective of this project is to provide a data mining solution for the above-mentioned real-world problem. The problem is related to predicting laptop prices according to the requirements of the user who needs to buy laptops. Prior analysis of the data set revealed the variables and how they impact the prices of laptops. Therefore, regression is the reliable method which use as the data mining function to build models after applying specific algorithms. The optimal model owes the higher accuracy that will be deployed as a web application. As the final output, the optimal model will be developed and deployed as a single convenient web application using web development tools. The user possesses the ability to provide the required features and based on the user's requirements the model will predict the relevant prices.

Choice of Technology

Python

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Heroku

Developers use Heroku to deploy, manage, and scale modern apps. This platform is elegant, flexible, and easy to use, offering developers the simplest path to getting their apps to market. Heroku is fully managed, giving developers the freedom to focus on their core product without the distraction of maintaining servers, hardware, or infrastructure. The Heroku experience provides services, tools, workflows, and polyglot support all designed to enhance developer.

Methodology

The project uses the Google Collaboratory tool to create regression models. The project began with retrieving dataset .The workflow of modeling can be seen in Figure 1. After the dataset is imported, the preprocess stage is started , then Exploratory Data Analysis stage is carried out , and as the last stage is to do model building . The project is about Regression problem. Therefore, we build three regression models to find out best model out of by comparing their accuracy. After finding out the best model, that regression model will be integrated into the web predict

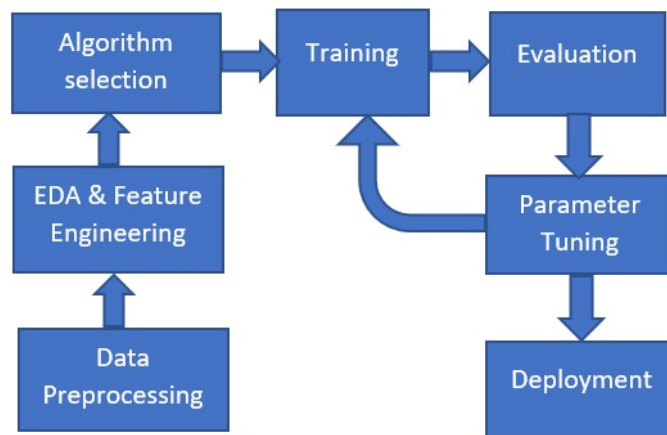


Figure1 – workflow of model building

Data set Introduction

Laptop Price Dataset. – <https://www.kaggle.com/datasets/aggle6666/laptop-price-prediction-dataset>

The data set downloaded from the Kaggle website. This dataset consists of 13 columns and 1303 rows of data. Each column is a specification of a laptop in general, the specifications are Company, Product, TypeName, Inches, Screen Resolution, CPU, Memory, Ram, GPU, operating system, weight, and price. The dataset has missing values in 'OpSys' column also has 28 duplicate rows. Details of this "Laptop Price" dataset can be seen in Figure 2.

```
data.head()
```

	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Memory	Ram	Gpu	OpSys	Weight	Price_euros
0	1	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	128GB SSD	8GB	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69
1	2	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	128GB Flash Storage	8GB	Intel HD Graphics 6000	macOS	1.34kg	898.94
2	3	HP	250 G6	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	256GB SSD	8GB	Intel HD Graphics 620	NaN	1.86kg	575.00
3	4	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	512GB SSD	16GB	AMD Radeon Pro 455	macOS	1.83kg	2537.45
4	5	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	256GB SSD	8GB	Intel Iris Plus Graphics 650	macOS	1.37kg	1803.60

Figure2 - Data set details

```
Print number of null values in each column

data.isnull().sum()

laptop_ID      0
Company        0
Product        0
TypeName       0
Inches        0
ScreenResolution 0
Cpu            0
Memory        0
Ram           0
Gpu           0
OpSys         66
Weight        0
Price_euros   0
dtype: int64

[125] data["OpSys"].isnull().sum()

66
```

Figure 3- Missing values

```
Checking for duplicated rows

[132] dataSet.duplicated().sum()

28

Remove duplicate rows

[133] dataSet = dataSet.drop_duplicates()

After removing duplicates , check changes of number of rows and columns

[134] dataSet.shape

(1275, 12)
```

Figure 4 – Duplicate Rows

Data preprocessing and transformation

Preprocessing part mainly includes Data Cleaning and Feature Engineering techniques. Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. Steps involved in Data Cleaning can be seen in Figure 5.

Data Cleaning



Figure 5

1. Removal of unwanted observations.

This included redundant or irrelevant values in the dataset. Most frequently redundant observations arise during data collection and irrelevant observations are those that do not actually fit the specific problem that trying to solve.

Redundant data alter the efficiency by a great extent also may produce unfaithful results. In this project we find out that there is some number of redundant observations.(Figure 6)

Checking for duplicated rows

```
[ ] dataSet.duplicated().sum()
```

28

Figure 6

Then the second step was removing the duplicate rows.(Figure 7)

Remove duplicate rows

```
[ ] dataSet = dataSet.drop_duplicates()
```

After removing duplicates , check changes of number of rows and columns

```
dataSet.shape
```

(1275, 12)

Figure 7

After analyzed the data set, we find out that the primary key column is not utilized for modeling. Then we drop the primary key column.(Figure 8)

Removing Columns that are not be Utilized for ML Modelling

```
dataset = data.drop(['laptop_ID'],axis=1)
dataset.head()
```

	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Memory	Ram	Gpu	Opsys	Weight	Price_euros
0	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	128GB SSD	8GB	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69
1	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	128GB Flash Storage	8GB	Intel HD Graphics 6000	macOS	1.34kg	898.94
2	HP	250 G6	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	256GB SSD	8GB	Intel HD Graphics 620	No OS	1.86kg	575.00
3	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	512GB SSD	16GB	AMD Radeon Pro 455	macOS	1.83kg	2537.45
4	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	256GB SSD	8GB	Intel Iris Plus Graphics 650	macOS	1.37kg	1803.60

Figure 8

2. Fixing structural errors

Structural errors include typos in the name of features, mislabeled classes, same attribute with different name or inconsistent capitalization.

We converted feature names to lowercase for reduce structural errors.(Figure 9)

```
#column names in the dataset will be changed to lowercase
dataset = dataset.rename(columns = str.lower)
dataset.columns
```

```
Index(['company', 'product', 'typename', 'inches', 'screenresolution', 'cpu',
      'memory', 'ram', 'gpu', 'opsys', 'weight', 'price_euros'],
      dtype='object')
```

```
dataset.head()
```

	company	product	typename	inches	screenresolution	cpu	memory	ram	gpu	opsys	weig
0	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	128GB SSD	8GB	Intel Iris Plus Graphics 640	macOS	1.37
1	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	128GB Flash Storage	8GB	Intel HD Graphics 6000	macOS	1.34
2	HP	250 G6	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	256GB SSD	8GB	Intel HD Graphics 620	No OS	1.86
3	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	512GB SSD	16GB	AMD Radeon Pro 455	macOS	1.83
4	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	256GB SSD	8GB	Intel Iris Plus Graphics 650	macOS	1.37

Figure 9

3. Managing unwanted outliers

Outliers can cause problems with certain types of models. For example, linear regression models are less robust to outliers than decision tree models. While we are using decision tree types of models , we decided to not to remove outliers.

4. Handling missing data

In our data set there is column called 'opsys' has some missing values. Therefore, we decided to use a default value("No OS") for missing values rather than dropping that column because it will impact for the price of laptop.

to deal with missing values is, "fillna()". This can be used to fill the NaN values by indicating the missing values in the column.

```
[8] data['Opsys'].fillna('No OS', inplace = True)
```

```
data["Opsys"]
```

0	macOS
1	macOS
2	No OS
3	macOS
4	macOS
...	...
1298	Windows 10
1299	Windows 10
1300	Windows 10
1301	Windows 10
1302	Windows 10

Name: Opsys, Length: 1303, dtype: object

Figure 10

Feature Engineering

Let's walk through one by one what happens inside each column and what kind of feature engineering techniques use for.

1. Company Column

- to get an understand about what kind of value counts each category have(Figure 11)
- to get an overview how price vary with each company.(Figure 12)

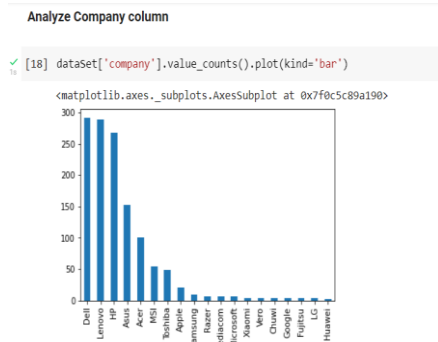


Figure 11

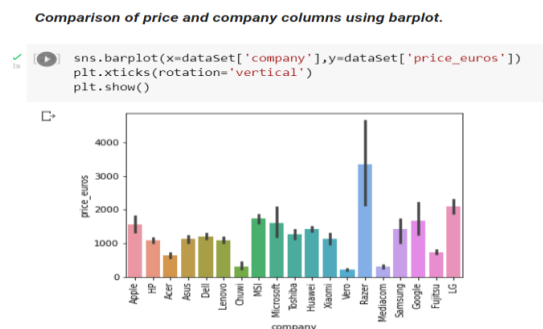


Figure 12

According to the value count of company, initially there are 19 categories. Out of 19, 11 categories' value counts be under 20. Therefore, we assumed to take all these(11 companies) to a one category as 'Others'. Then we defined a function to fulfil above mentioned action. After categorize according to the assumption that we made, final output of company column shown in Figure 13.

After categorized the companies check value counts for each company.

```
dataSet['company'].value_counts()
```

Dell	291
Lenovo	289
HP	268
Asus	152
Acer	101
MSI	54
Other	51
Toshiba	48
Apple	21

Name: company, dtype: int64

Figure 13

2. Product Column

-to get an understand about what kind of value counts each product have(Figure 14)

```
Analyze Product Column

dataset['product'].value_counts()

XPS 13      30
Inspiron 3567 25
250 66      21
Vostro 3568  19
Legion Y520-15IKBN 19
..          ..
VivoBook E201NA 1
Ideapad 520-15IKBR 1
Thinkpad X260  1
Rog G752VL-UM71T 1
X5535A-XX031T (N3050/4GB/500GB/W10) 1
Name: product, Length: 618, dtype: int64

[ ] len(dataset['product'].value_counts())

618
```

Figure 14

After analyzing the product column, there were 618 categories of Products. If we encode those value there will be 618 columns belongs to Product column. Also , there is no common prefixes , therefore we decided to drop product column.

3. TypeName column

-to get an understand about what kind of value counts each type name have(Figure 15)

-to get an overview how price vary with each type name.(Figure 16)

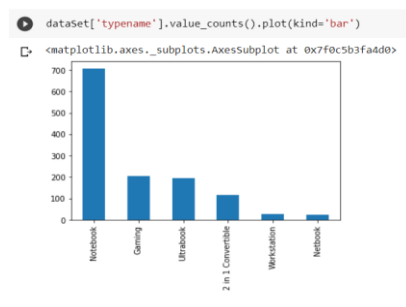


Figure 15



Figure 16

TypeName column has 6 categories. Therefore, onehot encoding is suitable. Suppose to use categories as it is.

4. Inches Column

Inches column already has float values. Then no need a transformation.

5. Screen resolution column

-to get an understand about what kind of value counts (Figure 17)

Analyze ScreenResolution column

```
dataset['screenresolution'].value_counts()
```

Full HD 1920x1080	585
1366x768	263
IPS Panel Full HD 1920x1080	226
IPS Panel Full HD / Touchscreen 1920x1080	51
Full HD / Touchscreen 1920x1080	47
1600x900	23
Touchscreen 1366x768	16
Quad HD+ / Touchscreen 3200x1800	15
IPS Panel 4K Ultra HD 3840x2160	12
IPS Panel 4K Ultra HD / Touchscreen 3840x2160	11
4K Ultra HD / Touchscreen 3840x2160	10
4K Ultra HD 3840x2160	7
Touchscreen 2560x1440	7
IPS Panel 1366x768	7
IPS Panel Retina Display 2560x1600	6
IPS Panel Retina Display 2304x1440	6
Touchscreen 2256x1504	6
IPS Panel Touchscreen 2560x1440	5
IPS Panel Quad HD+ / Touchscreen 3200x1800	4
IPS Panel Touchscreen 1920x1200	4

Figure 17

After analyzing the screen resolution column, we identified that there are same prefixes in some rows such as TouchScreen, IPS. Therefore, we create new columns called TouchScreen and IPS to store dummie values. (Figure 18)

```
[35] dataset['touchscreen'] = dataset['screenresolution'].apply(lambda x:1 if 'TouchScreen' in x else 0)
dataset['ips'] = dataset['screenresolution'].apply(lambda x:1 if 'IPS' in x else 0)
```

```
[36] dataset.head(2)
```

	company	product	typename	inches	screenresolution	cpu	memory	ram	gpu	opsys	weight	price_euros	touchscreen	ips
0	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	128GB SSD	8GB	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69	0	1
1	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	128GB Flash Storage	8GB	Intel HD Graphics 6000	macOS	1.34kg	898.94	0	0

Figure 18

We can clearly see, building new columns touchscreen and ips are extract meaningful vales of screen resolution column. Therefore, we created new cols called x_res, y_res to store values of screen resolution column(2560 x 1600) to separate columns. After get correlation for all columns, we decided to create new column called PPI(Pixels Per Inch) which has good correlation. (Figure 19)

```
#Calculating Pixels Per Inch(Pixel Density)
#PPI = (root(Width*2*height*2)/Screen Size)
#taking into a new column , PPI
data['PPI'] = (((data['X_res']**2) + (data['Y_res']**2))*0.5/data['Inches']).astype('float')
```

```
data.corr()['Price_euros']
```

Inches	0.068197
Price_euros	1.000000
Touchscreen	0.191226
IPS	0.252008
X_res	0.556529
Y_res	0.552809
PPI	0.473487

Name: Price_euros, dtype: float64

We can see that PPI has good correlation with Price so will keep PPI column

```
[ ] data.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Memory	Ram	Gpu	OpSys	Weight	Price_euros	Touchscreen	IPS	X_res	Y_res	PPI
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	128GB SSD	8GB	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69	0	1	2560.0	1600.0	226.983005
1	Apple	Macbook Air	13.3	1440x900	Intel Core i5 1.8GHz	128GB Flash Storage	8GB	Intel HD Graphics 6000	macOS	1.34kg	898.94	0	0	1440.0	900.0	144.000000

Figure 19

Therefore, instead of x_res and y_res we created new column PPI using values of x_res and y_res. This is called feature extraction. Then we decided to drop x_res and y_res.(Figure 20)

```
[ ] #Dropping unwanted Columns -> "X_res","Y_res","Inches" & "ScreenResolution"
data.drop(columns = ["X_res","Y_res","Inches","ScreenResolution"],inplace =True,axis=1)
```

```
[ ] data.head()
```

	Company	TypeName	Cpu	Memory	Ram	Gpu	OpSys	Weight	Price_euros	Touchscreen	IPS	PPI
0	Apple	Ultrabook	Intel Core i5 2.3GHz	128GB SSD	8GB	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69	0	1	226.983005
1	Apple	Ultrabook	Intel Core i5 1.8GHz	128GB Flash Storage	8GB	Intel HD Graphics 6000	macOS	1.34kg	898.94	0	0	127.677940
2	HP	Notebook	Intel Core i5 7200U 2.5GHz	256GB SSD	8GB	Intel HD Graphics 620	No OS	1.86kg	575.00	0	0	141.211998
3	Apple	Ultrabook	Intel Core i7 2.7GHz	512GB SSD	16GB	AMD Radeon Pro 455	macOS	1.83kg	2537.45	0	1	220.534624
4	Apple	Ultrabook	Intel Core i5 3.1GHz	256GB SSD	8GB	Intel Iris Plus Graphics 650	macOS	1.37kg	1803.60	0	1	226.983005

Figure 20

6. CPU column

-to get an understand about cpu column values(Figure 21)

```
#CPU
data['Cpu'].value_counts()
```

Intel Core i5 7200U 2.5GHz	190
Intel Core i7 7700HQ 2.8GHz	146
Intel Core i7 7500U 2.7GHz	134
Intel Core i7 8550U 1.8GHz	73
Intel Core i5 8250U 1.6GHz	72
...	
Intel Core M M3-6Y30 0.9GHz	1
AMD A9-Series 9420 2.9GHz	1
Intel Core i3 6006U 2.2GHz	1
AMD A6-Series 7310 2GHz	1
Intel Xeon E3-1535M v6 3.1GHz	1

Name: Cpu, Length: 118, dtype: int64

Figure 21

In Cpu column there are same prefixes we can see. Therefore, we suggested to create new column and store first three indexes of values in Cpu column and drop the original cpu column. (Figure 22)

```
[ ] data['CPU'] = data['Cpu'].apply(lambda x: "-".join(x.split()[0:3]))
```

```
[ ] def set_processor(name):
    if name == 'Intel Core i7' or name == 'Intel Core i5' or name == 'Intel Core i3':
        return name
    else:
        if name.split()[0] == 'Intel':
            return 'Other Intel Processor'
        else:
            return 'AMD Processor'
```

```
[ ] data['CPU_Brand'] = data['CPU'].apply(set_processor)
```

```
[ ] data.head()
```

	Company	TypeName	Cpu	Memory	Ram	Gpu	OpSys	Weight	Price_euros	Touchscreen	IPS	PPI	CPU	CPU_Brand
0	Apple	Ultrabook	Intel Core i5 2.3GHz	128GB SSD	8GB	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69	0	1	226.983005	Intel Core i5	Intel Core i5
1	Apple	Ultrabook	Intel Core i5 1.8GHz	128GB Flash Storage	8GB	Intel HD Graphics 6000	macOS	1.34kg	898.94	0	0	127.677940	Intel Core i5	Intel Core i5
2	HP	Notebook	Intel Core i5 7200U 2.5GHz	256GB SSD	8GB	Intel HD Graphics 620	No OS	1.86kg	575.00	0	0	141.211998	Intel Core i5	Intel Core i5
3	Apple	Ultrabook	Intel Core i7 2.7GHz	512GB SSD	16GB	AMD Radeon Pro 455	macOS	1.83kg	2537.45	0	1	220.534624	Intel Core i7	Intel Core i7
4	Apple	Ultrabook	Intel Core i5 3.1GHz	256GB SSD	8GB	Intel Iris Plus Graphics 650	macOS	1.37kg	1803.60	0	1	226.983005	Intel Core i5	Intel Core i5

Figure 22

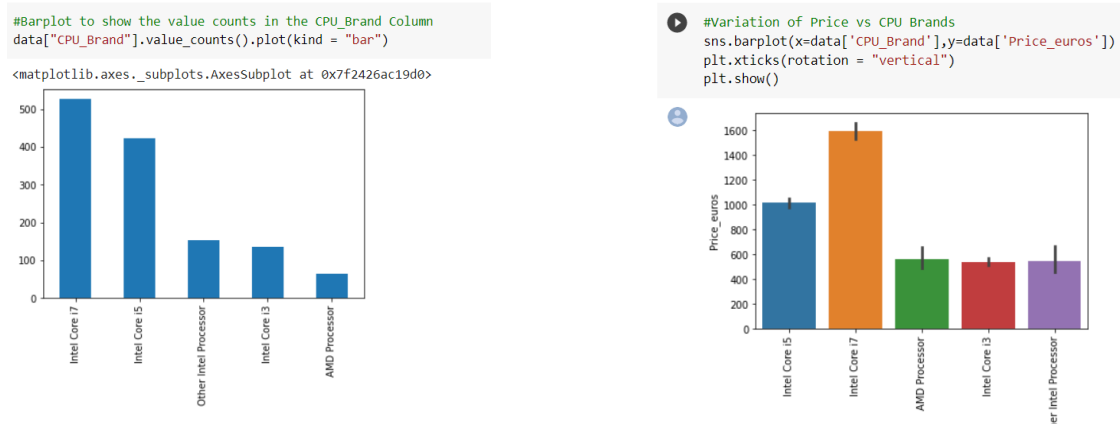


Figure 23- Comparison count of newly created column values

Figure 24 - Comparison of how price vary according to the newly created values.

7. Ram column

Ram column value contains categorical values ; therefore, we converted categorical value to numeric by replacing the 'GB' phrase.

```
[ ] data['Ram'] = data['Ram'].str.replace('GB','') #eliminating the suffixes
```

```
data.head()
```

	Company	TypeName	Memory	Ram	Gpu	OpSys	Weight	Price_euros	Touchscreen	IPS	PPI	CPU_Brand
0	Apple	Ultrabook	128GB SSD	8	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69	0	1	226.983005	Intel Core i5
1	Apple	Ultrabook	128GB Flash Storage	8	Intel HD Graphics 6000	macOS	1.34kg	898.94	0	0	127.677940	Intel Core i5
2	HP	Notebook	256GB SSD	8	Intel HD Graphics 620	No OS	1.86kg	575.00	0	0	141.211998	Intel Core i5
3	Apple	Ultrabook	512GB SSD	16	AMD Radeon Pro 455	macOS	1.83kg	2537.45	0	1	220.534624	Intel Core i7
4	Apple	Ultrabook	256GB SSD	8	Intel Iris Plus Graphics 650	macOS	1.37kg	1803.60	0	1	226.983005	Intel Core i5

Figure 25

8. Weight column

Weight column contains categorical values , therefore converted it into numerical by replacing "kg".

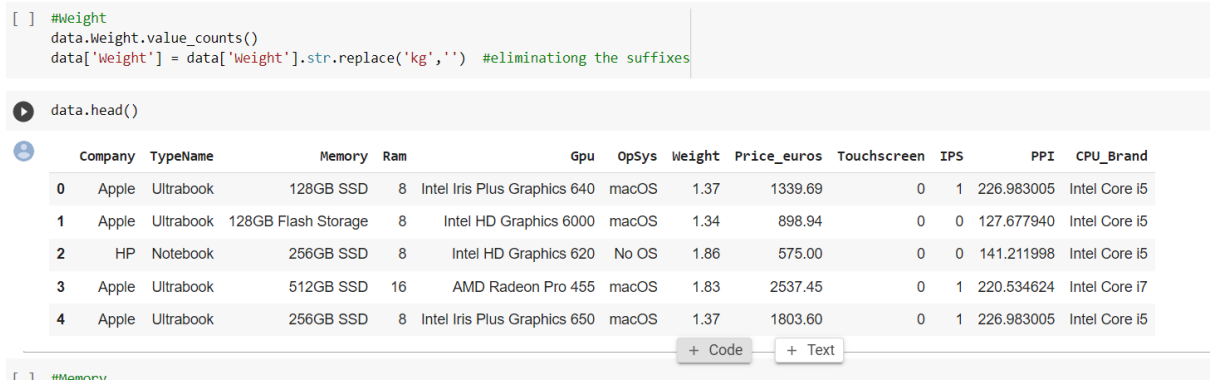


Figure 26

9. Memory Column

Memory column includes more than one value. Also it includes whitespaces and unnecessary symbols. (Figure 27)

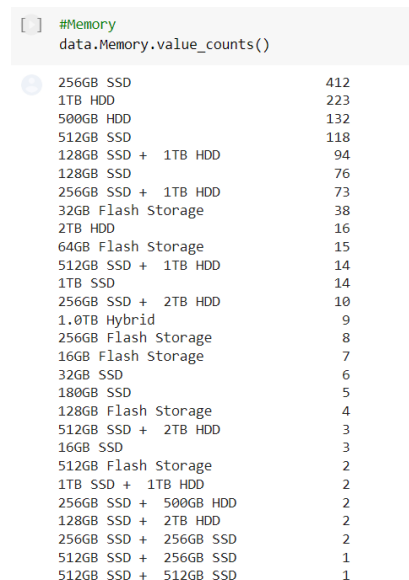


Figure 27

After analyzing the column, we decided to create new four more columns and add each relevant value for each column. The newly created columns are HDD,SDD, Hybrid, Flash_Storage. (Figure 28)



Figure 28

Then find out correlation of each column with price. Then identified some columns have low correlation . We decided to drop low correlation value columns. (Figure 29)

```
[ ] data.corr()['Price_euros']
```

Price_euros	1.000000
Touchscreen	0.191226
IPS	0.252208
PPI	0.473487
HDD	-0.096441
SSD	0.670799
Hybrid	0.007989
Flash_Storage	-0.040511

Name: Price_euros, dtype: float64

```
#Feature Hybrid and Flash Storage showing negligible correlation(correlatrion closer to zero) with Price we will drop those feature
```

```
data.drop(columns = ["Hybrid","Flash_Storage"],inplace = True,axis=1)
```

Figure 29

10. Gpu Column

Gpu column consists of numbers, strings and whitespaces. Therefore, decided to take first word (0th index) each value and store them in new column. (Figure 30, Figure 31)

```
#Gpu
```

```
data.Gpu.value_counts()
```

Intel HD Graphics 620	281
Intel HD Graphics 520	185
Intel UHD Graphics 620	68
Nvidia GeForce GTX 1050	66
Nvidia GeForce GTX 1060	48
...	
AMD Radeon R5 520	1
AMD Radeon R7	1
Intel HD Graphics 540	1
AMD Radeon 540	1
ARM Mali T860 MP4	1

Name: Gpu, Length: 110, dtype: int64

Figure 30

```
#To extract the 0th index of 'Gpu' column vals and taking into a new col named 'Gpu_Brand'
```

```
data['Gpu_Brand'] = data['Gpu'].apply(lambda x:x.split()[0])
```

```
[ ] #Checking the Value Counts
```

```
data['Gpu_Brand'].value_counts()
```

Intel	722
Nvidia	400
AMD	180
ARM	1

Name: Gpu_Brand, dtype: int64

Figure 31

We can clearly see there is one value of ARM type. It is better to delete that value. (Figure 32)

```
[ ] data = data[data['Gpu_Brand'] != 'ARM']
```

```
#Checking the value counts of the newly created 'Gpu_Brand' column
data['Gpu_Brand'].value_counts()
```

```
Intel      722
Nvidia     400
AMD        180
Name: Gpu_Brand, dtype: int64
```

Figure 32

```
#Variation of Price vs Gpu_Brand
sns.barplot(x=data['Gpu_Brand'],y=data['Price_euros'],estimator=np.median)
plt.xticks(rotation='vertical')
plt.show()
```

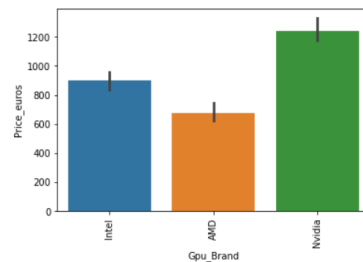


Figure 33 – How price vary with newly created GPU values

Finally , we could drop the original(previous) Gpu column.

11. Operating System column

-to get an understand about value count(Figure 34)

```
#Operating System
#Checking Value Counts
data['OpSys'].value_counts()
```

```
Windows 10      1072
No OS           66
Linux           62
Windows 7       45
Chrome OS       26
macOS           13
Mac OS X        8
Windows 10 S    8
Android         2
Name: OpSys, dtype: int64
```

Figure 34

It clearly determines there are three values started from the same name as 'Windows', also there are two values started from the same name 'mac'. Other values are different from each other. Therefore, suggested to store same name values in new column with that same name also create a new column to store other values inside a separate column. (Figure 35. Figure 36)

```

] #function to

def set_os(inp):
    if inp == 'Windows 10' or inp == 'Windows 7' or inp == 'Windows 10 S':
        return 'Windows'
    elif inp == 'macOS' or inp == 'Mac OS X':
        return 'Mac'
    else:
        return 'Others/No OS/Linux'

data['os'] = data['OpSys'].apply(set_os)

```

Figure 35

	Company	TypeName	Ram	OpSys	Weight	Price_euros	Touchscreen	IPS	PPI	CPU_Brand	HDD	SSD	Gpu_Brand	os
0	Apple	Ultrabook	8	macOS	1.37	1339.69	0	1	226.983005	Intel Core i5	0	128	Intel	Mac
1	Apple	Ultrabook	8	macOS	1.34	898.94	0	0	127.677940	Intel Core i5	0	0	Intel	Mac
2	HP	Notebook	8	No OS	1.86	575.00	0	0	141.211998	Intel Core i5	0	256	Intel	Others/No OS/Linux
3	Apple	Ultrabook	16	macOS	1.83	2537.45	0	1	220.534624	Intel Core i7	0	512	AMD	Mac
4	Apple	Ultrabook	8	macOS	1.37	1803.60	0	1	226.983005	Intel Core i5	0	256	Intel	Mac

Figure 36

```

#barplot to represent the variation of Price vs OS
sns.barplot(x=data['os'],y=data['Price_euros'])
plt.xticks(rotation='vertical')
plt.show()

```

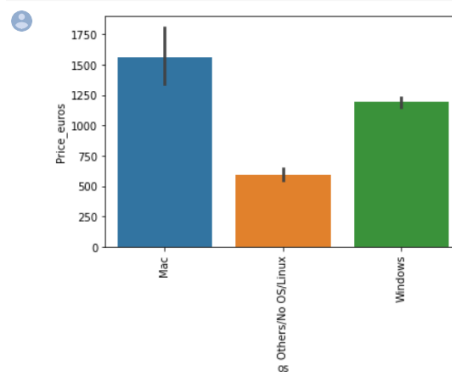


Figure 37- how price varies with new column 'os'

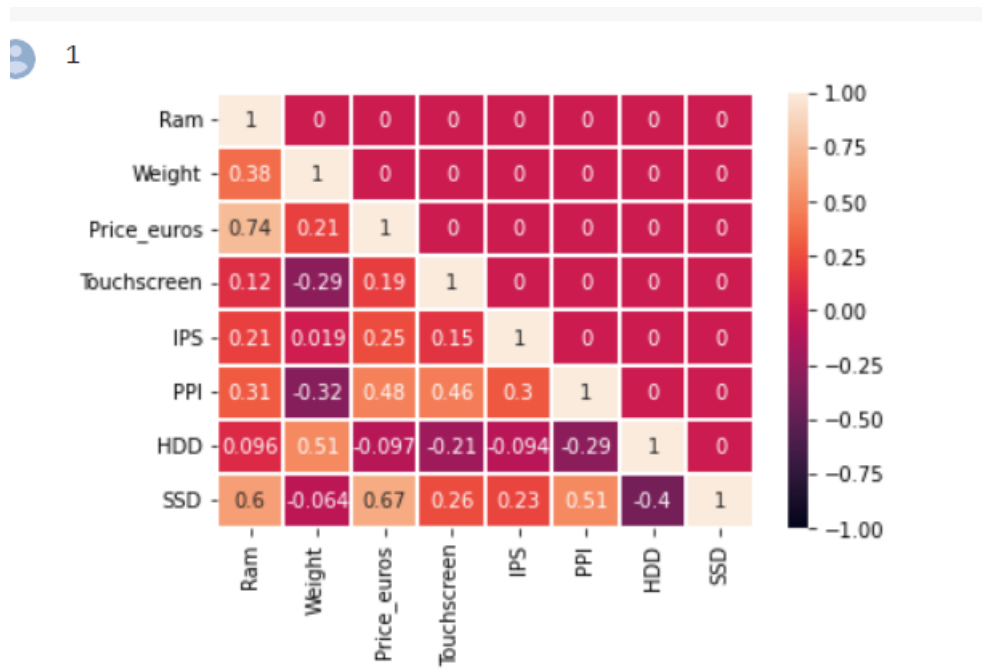


Figure 38 – Heat Map for all the columns(varies of correlation between each variable)

Final output after Data Cleaning and Feature Engineering can be seen in Figure 39.

	Company	TypeName	Ram	Weight	Price_euros	Touchscreen	IPS	PPI	CPU_Brand	HDD	SSD	Gpu_Brand	os
0	Apple	Ultrabook	8	1.37	1339.69	0	1	226.983005	Intel Core i5	0	128	Intel	Mac
1	Apple	Ultrabook	8	1.34	898.94	0	0	127.677940	Intel Core i5	0	0	Intel	Mac
2	HP	Notebook	8	1.86	575.00	0	0	141.211998	Intel Core i5	0	256	Intel	Others/No OS/Linux
3	Apple	Ultrabook	16	1.83	2537.45	0	1	220.534624	Intel Core i7	0	512	AMD	Mac
4	Apple	Ultrabook	8	1.37	1803.60	0	1	226.983005	Intel Core i5	0	256	Intel	Mac

Figure 39

As the final step of preprocessing is get dummies values for each column. (Figure 40)

```
[ ] dataSet = pd.get_dummies(data)
```

dataSet

	Ram	Weight	Price_euros	Touchscreen	IPS	PPI	HDD	SSD	Company_Acer	Company_Apple	...	CPU_Brand_Intel Core i3	CPU_Brand_Intel Core i5	CPU_Brand_Intel Core i7	CPU_Brand_Other Intel Processor	Gpu_Brand_A
0	8	1.37	1339.69	0	1	226.983005	0	128	0	1	...	0	1	0	0	0
1	8	1.34	898.94	0	0	127.677940	0	0	0	1	...	0	1	0	0	0
2	8	1.86	575.00	0	0	141.211998	0	256	0	0	...	0	1	0	0	0
3	16	1.83	2537.45	0	1	220.534624	0	512	0	1	...	0	0	1	0	0
4	8	1.37	1803.60	0	1	226.983005	0	256	0	1	...	0	1	0	0	0
...
1298	4	1.80	638.00	1	1	157.350512	0	128	0	0	...	0	0	1	0	0
1299	16	1.30	1499.00	1	1	276.053530	0	512	0	0	...	0	0	1	0	0
1300	2	1.50	229.00	0	0	111.935204	0	0	0	0	...	0	0	0	0	1
1301	6	2.19	764.00	0	0	100.454670	1000	0	0	0	...	0	0	1	0	0

Figure 40

Implementation of models

Since our main target is to predict the prices of laptops by studying the values of other variables, here we use Regression as our Data mining Technique. We used the following set of Machine Learning Algorithms to allow machines to learn the relationships within the data provided and make predictions based on patterns or rules notified from the dataset. Every Algorithm has its own assumptions, pros, and cons as discussed below.

Random Forest Algorithm

Random Forests are a combination of decision trees. It is a Supervised Learning algorithm used for classification and regression. The input data is passed through multiple decision trees. It executes by constructing a different number of decision trees at training time and outputting the class that is the mode of the classes (for classification) or mean prediction (for regression) of the individual trees.

Assumptions:

- Assumption of no formal distributions. Being a non-parametric model, it can handle skewed and multi-modal data.

Pros:

- Robust to outliers.
- Works well for non-linear data.
- Low risk of overfitting.
- Runs efficiently on large datasets.

Cons:

- Slow training.
- Biased when dealing with categorical variables.

Ridge Algorithm

Assumptions:

- Variables of constant variance, independence, and linearity.

Pros

- Trades variance for bias (i.e. in presence of co-linearity, it is worth having biased results, to lower the variance.)
- Prevents overfitting

Cons

- Increases bias
- Need to select perfect alpha (hyperparameter)
- Model interpret-ability is low

Decision Tree Algorithm

The decision tree models can be applied to all the data which contains numerical features and categorical features. Decision trees are good at capturing non-linear interaction between the features and the target variable. Decision trees somewhat match human-level thinking so it's very intuitive to understand the data.

Simply, a decision tree is a tree where each node represents a feature, each branch represents a decision, and each leaf represents an outcome(numerical value for regression).

Since here we have both numerical and categorical features in this dataset after the preprocessing; we decided to go with this algorithm as well.

Assumptions:

- Initially, the whole training data is considered as root.
- Records are distributed recursively based on the attribute value.

Pros:

- Compared to other algorithms, data preparation requires less time.
- Doesn't require data to be normalized.
- Missing values, to an extent, don't affect its performance much.
- Is very intuitive as can be explained as if-else conditions.

Cons:

- Needs a lot of time to train the model.
- A small change in data can cause a considerably large change in the Decision Tree structure.
- Comparatively expensive to train.
- Not good for regression tasks.

Model Building.

Separating the dataset into testing and training sets.

```
✓ [117] # split the dataset
X_train,X_test,y_train,y_test = train_test_split(dataSet.drop('Price_euros',axis=1),np.log(dataSet["Price_euros"]),test_size=0.25)
```

Figure 41

```

✓ [121] X_train.shape,X_test.shape
0s ((976, 33), (326, 33))

✓ [122] y_train.shape,y_test.shape
0s ((976,), (326,))

✓ [123] X_train.columns
0s Index(['Ram', 'Weight', 'Touchscreen', 'IPS', 'PPI', 'HDD', 'SSD',
       'Company_Acer', 'Company_Apple', 'Company_Asus', 'Company_Dell',
       'Company_HP', 'Company_Lenovo', 'Company_MSI', 'Company_Other',
       'Company_Toshiba', 'TypeName_2 in 1 Convertible', 'TypeName_Gaming',
       'TypeName_Netbook', 'TypeName_Notebook', 'TypeName_Ultrabook',
       'TypeName_Workstation', 'CPU_Brand_AMD Processor',
       'CPU_Brand_Intel Core i3', 'CPU_Brand_Intel Core i5',
       'CPU_Brand_Intel Core i7', 'CPU_Brand_Other Intel Processor',
       'Gpu_Brand_AMD', 'Gpu_Brand_Intel', 'Gpu_Brand_Nvidia', 'os_Mac',
       'os_Others/No OS/Linux', 'os_Windows'],
      dtype='object')

```

Figure 42

Feature selection for reducing the input variable to the model by using only relevant data and getting rid of noise in data.

```

[124] def anovatest(x,y,data):
      model = ols('x~y',data=data).fit()
      anova = sm.stats.anova_lm(model,type=2)
      pvalue = anova['PR(>F)'][0]

      if pvalue < 0.05:
          msg = 'Reject H0: Feature {} is significant'.format(x.name)
      else:
          msg = 'FTR H0: Feature {} is insignificant'.format(x.name)

      return(msg)

[125] dataSet.columns
Index(['Ram', 'Weight', 'Price_euros', 'Touchscreen', 'IPS', 'PPI', 'HDD',
       'SSD', 'Company_Acer', 'Company_Apple', 'Company_Asus', 'Company_Dell',
       'Company_HP', 'Company_Lenovo', 'Company_MSI', 'Company_Other',
       'Company_Toshiba', 'TypeName_2 in 1 Convertible', 'TypeName_Gaming',
       'TypeName_Netbook', 'TypeName_Notebook', 'TypeName_Ultrabook',
       'TypeName_Workstation', 'CPU_Brand_AMD Processor',
       'CPU_Brand_Intel Core i3', 'CPU_Brand_Intel Core i5',
       'CPU_Brand_Intel Core i7', 'CPU_Brand_Other Intel Processor',
       'Gpu_Brand_AMD', 'Gpu_Brand_Intel', 'Gpu_Brand_Nvidia', 'os_Mac',
       'os_Others/No OS/Linux', 'os_Windows'],
      dtype='object')

```

Figure 43

```

✓ [126] #Testing Annova test for feature selection
0s
anovatest(data.Company, data.Price_euros,data)

'Reject H0: Feature Company is significant'

✓ [127] for i in data:
0s
    print(anovatest(data[i], data.Price_euros,data)) #performing Annova test for feature selection for all columns

Reject H0: Feature Company is significant
Reject H0: Feature TypeName is significant
Reject H0: Feature Ram is significant
Reject H0: Feature Weight is significant
Reject H0: Feature Price_euros is significant
Reject H0: Feature Touchscreen is significant
Reject H0: Feature IPS is significant
Reject H0: Feature PPI is significant
Reject H0: Feature CPU_Brand is significant
Reject H0: Feature HDD is significant
Reject H0: Feature SSD is significant
Reject H0: Feature Gpu_Brand is significant
Reject H0: Feature os is significant

All the features are significant

```

Figure 44

Feeding data to the model and getting the accuracy for each model.

Random Forest Algorithm

```

✓ [128] from sklearn.ensemble import RandomForestRegressor
2s
    from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

✓ [129] rf_model = RandomForestRegressor(n_estimators = 100, max_features = 'sqrt', max_depth = 10, random_state = 10).fit(X_train, y_train)

✓ [130] rf_model.fit(X_train,y_train)
2s
    RandomForestRegressor(max_depth=10, max_features='sqrt', random_state=10)

✓ [131] rf_model.score(X_test,y_test)
2s
    0.8388492599427908

✓ [132] prediction = rf_model.predict(X_test)
2s
    mse = mean_squared_error(y_test, prediction)
    rmse = mse**.5
    # print(mse)
    # print(rmse)

✓ [133] print("Model DT \n\tmse={}, \n\tmse={}".format(round(mse,2),round(np.sqrt(mse),2)))
2s
    print('R2 score',r2_score(y_test,prediction))
    print('MAE',mean_absolute_error(y_test,prediction))

Model DT
    mse=0.06,
    rmse=0.25
R2 score 0.8388492599427908
MAE 0.18310949509109786

✓ [134] print(np.exp(mse))
2s
    1.0652977956058098

```

Figure 45

Ridge Algorithm

```
✓ [143] from sklearn.linear_model import Ridge
Ds

✓ [144] ridge = Ridge()

✓ [145] ridge.fit(X_train, y_train)
      Ridge()

✓ [146] ridge.score(X_train, y_train)
Ds
      0.8269629898219333

✓ [147] ridge.score(X_test, y_test)
Ds
      0.7932750653394611

✓ [148] preds = ridge.predict(X_test)
Ds

✓ [149] # Calculating Mean Squared Error for Ridge
Ds
      mse3 = mean_squared_error(y_test, preds)

✓ [150] print("Model DT \n\tmse={}, \n\trmse={}".format(round(mse3,2),round(np.sqrt(mse3),2)))
1s
      print('R2 score',r2_score(y_test,preds))
      print('MAE',mean_absolute_error(y_test,preds))

      Model DT
          mse=0.08,
          rmse=0.28
      R2 score 0.7932750653394611
      MAE 0.22065433213902372
```

Figure 46

Decision Tree Algorithm

```

[135] import statsmodels.api as smapi
      from sklearn.tree import DecisionTreeRegressor
      from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error # calculating different regression metrics

      from sklearn.model_selection import GridSearchCV

[136] DT = DecisionTreeRegressor(random_state=0).fit(X_train,y_train)

[137] pred_DT = DT.predict(X_test)

[138] print(pred_DT[0:5])

[6.75578056 6.1717006 7.05098945 7.07918439 6.95559261]

[139] Df1 = pd.DataFrame({"actual_CCS":y_test,"predCCS_DT":pred_DT}) #Store the prediction into a data frame for analysis

[140] Df1

```

	actual_CCS	predCCS_DT
771	7.494988	6.755781
348	5.926926	6.171701
986	6.692084	7.050989
390	6.946014	7.079184
221	6.801283	6.955593
...
280	6.755769	6.855409
1291	5.666427	5.666427
1061	6.905753	7.549809
1274	5.910797	5.910797
536	6.681369	6.577861

326 rows x 2 columns

```

[141] # Calculating Mean Squared Error for Decision Tree
      mse2 = mean_squared_error(y_test,pred_DT)

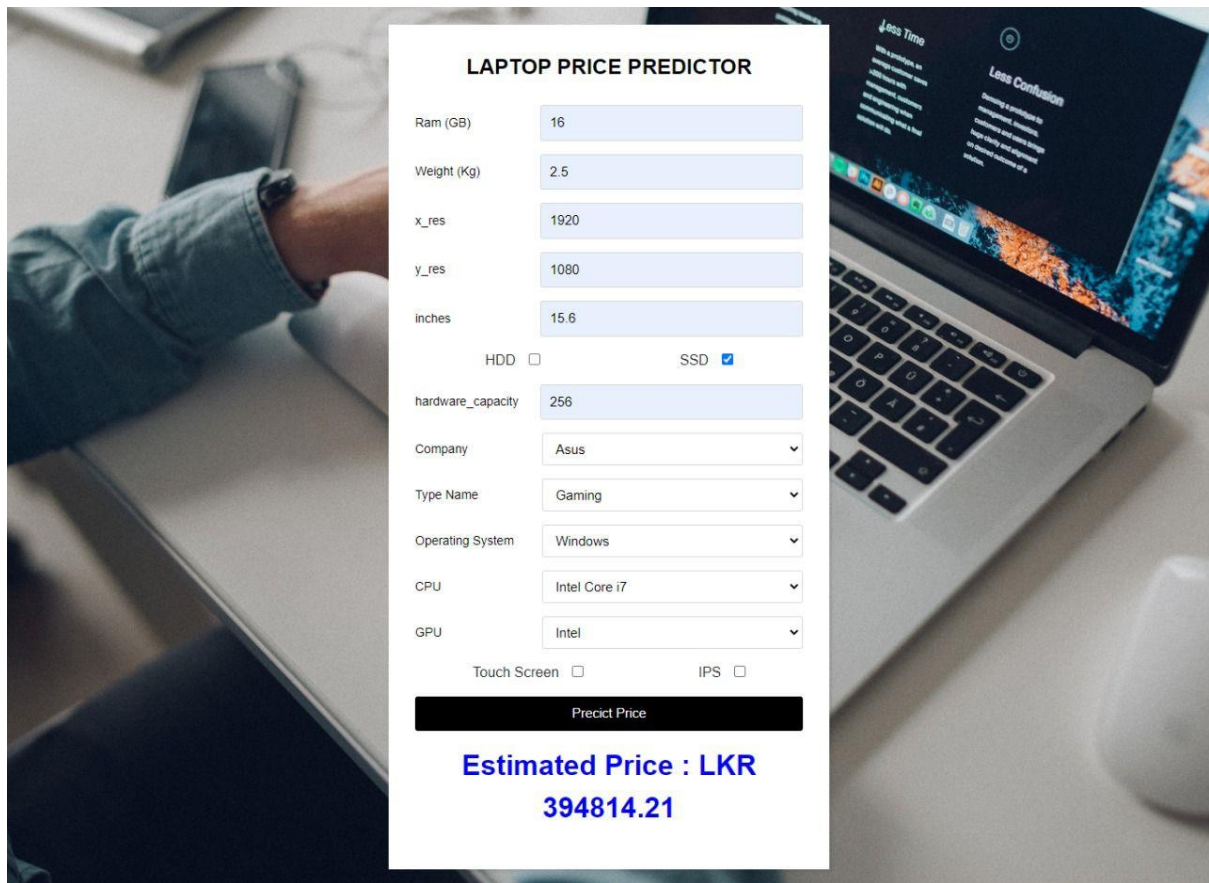
[142] print("Model DT \n\tmse={}, \n\tmse={}".format(round(mse2,2),round(np.sqrt(mse2),2)))
      print('R2 score',r2_score(y_test,pred_DT))
      print('MAE',mean_absolute_error(y_test,pred_DT))

Model DT
      mse=0.1,
      rmse=0.32
R2 score 0.7392748738455285
MAE 0.22703835262844418

```

Figure 47

Implementation of User Interface



LAPTOP PRICE PREDICTOR

Ram (GB)	16
Weight (Kg)	2.5
x_res	1920
y_res	1080
inches	15.6
HDD	<input type="checkbox"/>
SSD	<input checked="" type="checkbox"/>
hardware_capacity	256
Company	Asus
Type Name	Gaming
Operating System	Windows
CPU	Intel Core i7
GPU	Intel
Touch Screen	<input type="checkbox"/>
IPS	<input type="checkbox"/>

Predict Price

Estimated Price : LKR
394814.21

Figure 48

Deliverables

Our main objective is to implement a web application to predict the laptop price according to the user's requirements. The web application will predict the laptop price by getting user inputs from the user.

References

<https://analyticsindiamag.com/top-6-regression-algorithms-used-data-mining-applications-industry/>

<https://medium.com/@itssouravshrivas/exploratory-data-analysis-of-hotel-booking-demand-a-case-study-4a27bff589ca>

<https://towardsdatascience.com/random-forest-regression-5f605132d19d>

<https://hersanyagci.medium.com/detecting-and-handling-outliers-with-pandas-7adbcd5cad8>

Understanding Machine Learning: From Theory to Algorithms (Shai Shalev-Shwartz and Shai Ben-David)

Han, J., Kamber, M., and Pei, J., Data Mining: Concepts and Techniques, 3rd Ed., Elsevier Inc., 2012