Introduction

What is ready to be done under this assignment is to train a machine learning model. Here I chose a laptop price predictor application. The reason for doing something like this is that people started working from home when there were epidemic conditions. Then because of the use of laptop and people are looking to buy a new one, this application has been prepared to easily buy the laptop they want from the ecommerce web site. Here I used a dataset with details about the laptop. The columns here are company, type name, inches, screen resolution, CPU, ram, memory, Gpu, Operation system, weight and price. The machine learning techniques used to train here are linear regression, random forest, lasso, decision tree.

Data set link - https://drive.google.com/file/d/1Gd4i6KdciQn1ecGn2zUUnGTckEWHWMAO/view?usp=sharing

1. Literature reviews

These literature reviews use research done by others in a way that is consistent with my views. I hope to study them and comment on my application. And it will also identify the challenges and problems these judges face.

Ayesha Ayub Syed, Yaya Heryadi, Lukas and Antoni Wibowo have developed an application to classify laptops using machine learning at IMECS Hong Kong (International MultiConference of Engineers and Computer Scientists) in 2021. The reason for doing something like this is that people started working from home when there were epidemic conditions. Then because of the use of laptop and people are looking to buy a new one, this application has been prepared to easily buy the laptop they want from the ecommerce web site. For this, we can select the suitable product based on Company, Product, Type Name, Inches, Screen Resolution, CPU, RAM, Memory, GPU, Operating System and Weight data. Here they have grouped laptops as budget, midrange and flagship. Logistic regression, Decision tree, Artificial Neural Network and vector machine have been used as machine learning models for this task. The results show superior accuracy of 99% for SVM (Linear Kernel), 98% for SVM (Gaussian Kernel), Polynomial Logistic Regression and Decision Tree Classifier, 91% with Artificial Neural Network and 72% with SVM (polynomial kernel).) on our laptop product dataset. Here they have only broken the laptop into three sections (Syed, Heryadi and Wibowo, 2021).

Research professors Vaishali Surjuse, Sankalp Lohakare, Aayush Barapatre and Abhishek Chapke from the Department of Computer Technology and KDKCE & RTMNU University, India 2021 have developed the laptop price prediction system. They started doing this during the lockdown period in India. In India, demand for laptops increased after the nationwide lockdown, resulting in shipments of 4.1 million units in Q6 2021, the highest level in five years. They take the brand and model, RAM, ROM, GPU, and CPU as factors to determine the price. Listen explains in an article written for his master's thesis that a regression model built using a decision tree and a random forest regression tool can predict the price of a rented laptop with better accuracy than simple multivariate or multiple regression. They chose the decision tree algorithm on the basis that it is better at handling high dimensional data sets and is less likely to be over fitted and downsized. One weakness of these tests may be that they show no difference in basic metrics such as mean, variance, or standard deviation of simple regression with decision tree algorithm regression. More advanced (Surjuse *et al.*, 2022).

An application to estimate the cost of mobile phones has been created by students Pritish Arora, Sudhanshu Srivastava, and Professor Bindu Garg at Bharati Vidyapeeth (Deemed to be University) College of Engineering in Pune, India. To find and remove characteristics with the lowest computational cost that are less desired and redundant, certain feature selection techniques are applied. To reach the best level of accuracy, many categories have been applied. Results are measured by obtaining maximum accuracy and using the fewest characteristics possible. The claim is supported by the technique used to choose the optimal features and classifiers for the available data set. In any sort of marketing and business, this function may be used to identify the ideal product (with the lowest price and the greatest number of features). This study will be expanded in the future to offer a more comprehensive answer to the issue at hand and a more precise tool for pricing calculation (Arora, Srivastava and Garg, 2020).

2. Data set overview

The data set for my project had 1303 rows and 12 columns. The columns here are company, type name, inches, screen resolution, CPU, ram, memory, Gpu, Operation system, weight and price. This data set was a very noisy data set. And the presence of less amount of data here also became a little problematic. Although there is a small amount of data, more attention has been paid to its accuracy and this data set has been properly prepared for use.

```
[ ] df.head()
```

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	47895.5232
2	2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	30636.0000
3	3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	135195.3360
4	4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg	96095.8080

Figure 1: dataset head

It was difficult to find a dataset containing laptop prices in Sri Lanka so this is a Pakistani dataset. Therefore, its prices are also in the currency of that country.

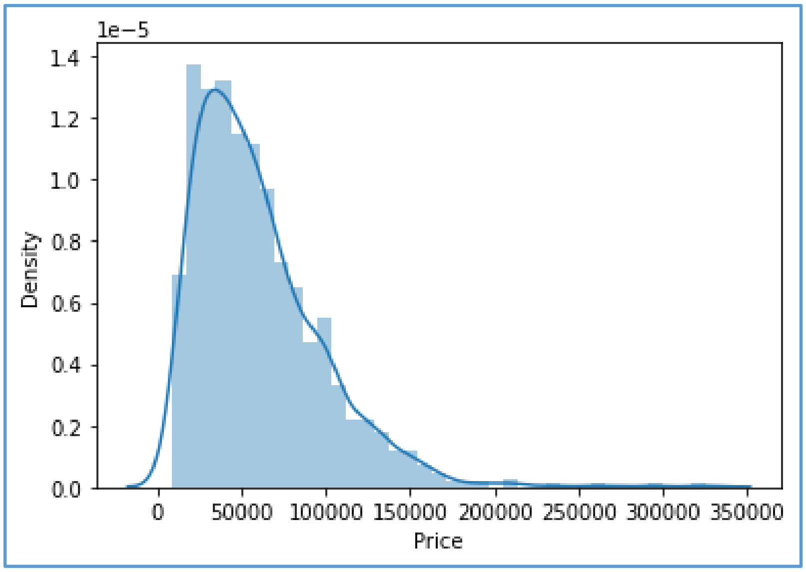


Figure 2: data distribution

Before training a model it is very important to understand the nature of a dataset. As we can see here, there is a positively skewed one. A positive skewed distribution, also known as a right-skewed distribution, is a form of distribution in statistics in which the majority of values are clustered around the left tail and the right tail is longer. The distribution with a positive skew is the exact opposite of one with a negative skew.

3. Data set cleaning

3.1.Import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Figure 3: import libraries

First I imported the libraries and started building the model. My chosen libraries are numpy, pandas and matplotlib. The reason for choosing the numpy library is because when we have to work on numerical data, we prefer the numpy module. Pandas was chosen because when we have to work on tabular data, we prefer the pandas module. The other library was chosen because of the need to draw diagrams.

3.2.Checking for null values

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        1303 non-null   int64
 1   Company           1303 non-null   object
 2   TypeName          1303 non-null   object
 3   Inches            1303 non-null   float64
 4   ScreenResolution  1303 non-null   object
 5   Cpu               1303 non-null   object
 6   Ram               1303 non-null   object
 7   Memory            1303 non-null   object
 8   Gpu               1303 non-null   object
 9   OpSys             1303 non-null   object
 10  Weight            1303 non-null   object
 11  Price             1303 non-null   float64
dtypes: float64(2), int64(1), object(9)
memory usage: 122.3+ KB
```

Figure 4: checking null values

The first step in cleaning the dataset was checking for null values. It was found that the data set is free of null values.

```
[ ] df.isnull().sum()

Unnamed: 0          0
Company             0
TypeName            0
Inches              0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
Price               0
dtype: int64
```

Figure 5: null values sum

Using the code df.isnull().sum() it was proved that null values are unique.

3.3.Cleaning rows

Convert numeric to ram and weight

In the dataset above, the data of ram and weight are in object form. But it cannot be used as such. Therefore, they should be taken as numeric. The string should have been deleted first. Then it was converted to numeric data. So the code below did it.

```
[ ] df['Ram'] = df['Ram'].str.replace('GB', '')
    df['Weight'] = df['Weight'].str.replace('kg','')
```

Figure 6: remove strings code 1

```
[ ] df['Ram'] = df['Ram'].astype('int32')
    df['Weight'] = df['Weight'].astype('float32')

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Company           1303 non-null   object
 1   TypeName          1303 non-null   object
 2   Inches            1303 non-null   float64
 3   ScreenResolution  1303 non-null   object
 4   Cpu               1303 non-null   object
 5   Ram               1303 non-null   int32
 6   Memory            1303 non-null   object
 7   Gpu               1303 non-null   object
 8   OpSys             1303 non-null   object
 9   Weight            1303 non-null   float32
 10  Price             1303 non-null   float64
dtypes: float32(1), float64(2), int32(1), object(7)
memory usage: 101.9+ KB
```

Figure 7: converting numeric

Both int and float are used to make these data numeric. The reason is that decimals are never used when mentioning ram. So the ram was converted to an int. But the data weight was obtained as a float because it has decimal places. After doing so, the dataset can be shown as follows.

```
[ ] df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080

Figure 8: clean dataset1

Cleaning screen resolution column

When you take the screen resolution column, it seems that several data are grouped together. Therefore, they had to be broken into separate columns. In that column, it is mentioned whether the screen is touchscreen, IPS or 4k. From that data, my model only gets Touchscreen and IPS.

```
[ ] df['ScreenResolution'].value_counts()

Full HD 1920x1080                                507
1366x768                                         281
IPS Panel Full HD 1920x1080                      230
IPS Panel Full HD / Touchscreen 1920x1080         53
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 Quad HD+ / Touchscreen 3200x1800         6
IPS Panel Retina Display 2560x1600                 6
IPS Panel Retina Display 2304x1440                 6
Touchscreen 2256x1504                              6
IPS Panel Touchscreen 2560x1440                    5
IPS Panel Retina Display 2880x1800                 4
IPS Panel Touchscreen 1920x1200                    4
1440x900                                           4
IPS Panel 2560x1440                                4
IPS Panel Quad HD+ 2560x1440                       3
Quad HD+ 3200x1800                                 3
1920x1080                                          3
Touchscreen 2400x1600                              3
2560x1440                                          3
IPS Panel Touchscreen 1366x768                     3
IPS Panel Touchscreen / 4K Ultra HD 3840x2160      2
IPS Panel Full HD 2160x1440                        2
IPS Panel Quad HD+ 3200x1800                       2
IPS Panel Retina Display 2736x1824                 1
IPS Panel Full HD 1920x1200                        1
```

Figure 9: Info of screen resolution column

Let's create a separate column for the touchscreen. Here it is taken as 1 if touchscreen is available and 0 if not.

```
[ ] df['Touchscreen'] = df['ScreenResolution'].apply(lambda x:1 if 'Touchscreen' in x else 0)

[ ] df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0

Figure 10: Create column to touch screen

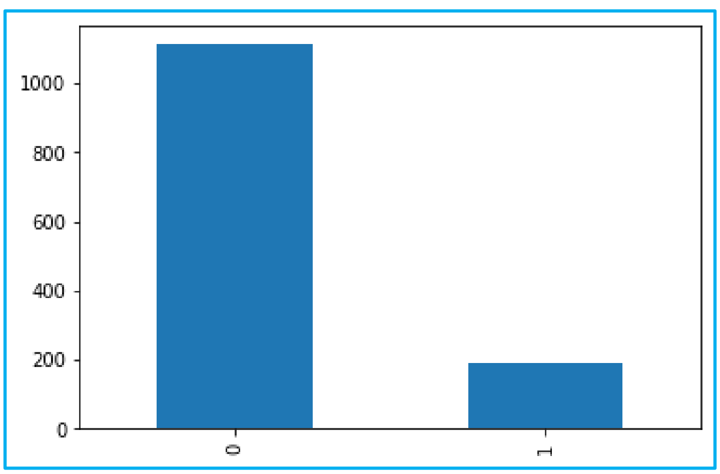
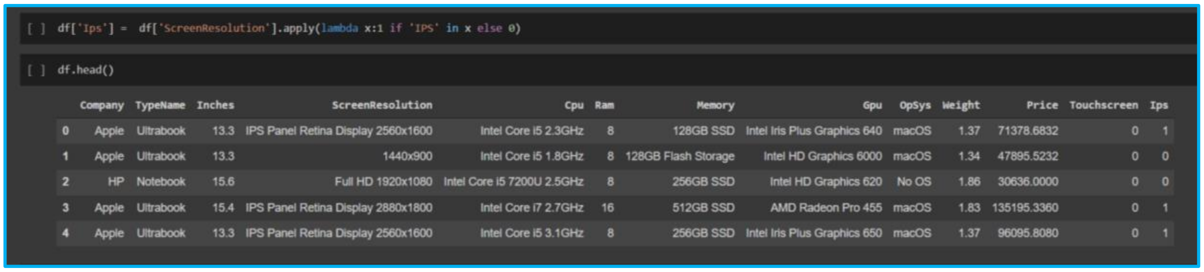


Figure 11: Bar plot of touchscreen (1 or 0)

The above bar plot shows the gap between laptops with and without a touchscreen. There seems to be a clear difference between the two. The number of non-touchscreens is very high.

Then a separate column was created for IPS. It is also divided into 0 and 1.



```
[ ] df['Ips'] = df['ScreenResolution'].apply(lambda x:1 if 'IPS' in x else 0)

[ ] df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1

Figure 12: Create column to IPS

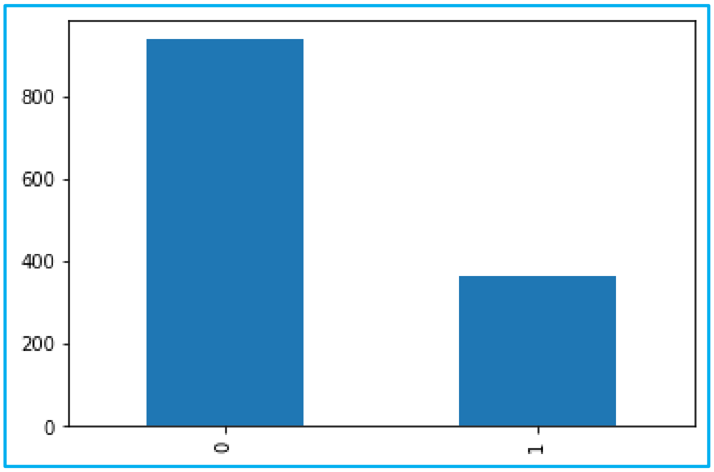


Figure 13: bar plot of Ips (1 or 0)

The above bar plot shows the gap between laptops with and without an Ips. There seems to be a clear difference between the two. The number of non-touchscreens is very high.

Then the resolution is divided into two separate columns as the x-axis resolution and the y-axis resolution. They are named x_res and y_res. The reason for such separation is that it is easier for us to train the model and when the data is fed to the model in this way, the accuracy can be increased.

```
new = df['ScreenResolution'].str.split('x', n=1, expand=True)
```

```
df['X_res'] = new[0]
df['Y_res'] = new[1]
```

```
df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	X_res	Y_res
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	IPS Panel Retina Display 2560	1600
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	1440	900
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	Full HD 1920	1080
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	IPS Panel Retina Display 2880	1800
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	IPS Panel Retina Display 2560	1600

Figure 14: creating X_res and Y_res columns

```
[33] df['X_res'] = df['X_res'].str.replace(',','').str.findall(r'(\d+\.?\d+)').apply(lambda x:x[0])
```

```
[34] df.head()
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	X_res	Y_res
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	2560	1600
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	1440	900
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	1920	1080
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	2880	1800
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	2560	1600

Figure 15: creating X_res and Y_res columns (code2)

Then the x_res and y_res columns were converted to int for ease of use.

```
df['X_res'] = df['X_res'].astype('int')
df['Y_res'] = df['Y_res'].astype('int')
```

Figure 16: converting x_res and y_res to integer

According to the image below, x_res and y_res have been converted to int64.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Company           1303 non-null   object
 1   TypeName          1303 non-null   object
 2   Inches            1303 non-null   float64
 3   ScreenResolution  1303 non-null   object
 4   Cpu               1303 non-null   object
 5   Ram               1303 non-null   int32
 6   Memory            1303 non-null   object
 7   Gpu               1303 non-null   object
 8   OpSys             1303 non-null   object
 9   Weight            1303 non-null   float32
 10  Price             1303 non-null   float64
 11  Touchscreen       1303 non-null   int64
 12  Ips               1303 non-null   int64
 13  X_res             1303 non-null   int64
 14  Y_res             1303 non-null   int64
dtypes: float32(1), float64(2), int32(1), int64(4), object(7)
memory usage: 142.6+ KB
```

Figure 17: data info (checking int)

Replacing inches, X and Y resolutions with PPI If you find the correlation of the column with the price using the corr method, we can see that the inches are not strongly correlated, but the X-axis and Y has a very high resolution, so we can take advantage of that and convert those three columns into a single column called Pixel Per Inch (PPI). Ultimately, our goal is to improve performance by having fewer features.

```
[ ] df['ppi'] = (((df['X_res']**2) + (df['Y_res']**2))**0.5/df['Inches']).astype('float')
```

Figure 18: PPI function

The following image shows how to drop screen resolution, inches, x_res, y_res columns.

```
[ ] df.drop(columns=['ScreenResolution'],inplace=True)
```

```
[ ] df.head()
```

	Company	TypeName	Inches	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	X_res	Y_res	ppi
0	Apple	Ultrabook	13.3	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	2560	1600	226.983005
1	Apple	Ultrabook	13.3	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	1440	900	127.677940
2	HP	Notebook	15.6	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	1920	1080	141.211998
3	Apple	Ultrabook	15.4	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	2880	1800	220.534624
4	Apple	Ultrabook	13.3	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	2560	1600	226.983005

```
[ ] df.drop(columns=['Inches','X_res', 'Y_res'],inplace=True)
```

```
[ ] df.head()
```

	Company	TypeName	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	ppi
0	Apple	Ultrabook	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	226.983005
1	Apple	Ultrabook	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	127.677940
2	HP	Notebook	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	141.211998
3	Apple	Ultrabook	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	220.534624
4	Apple	Ultrabook	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	226.983005

Figure 19: drop (screen resolution, x_res, y_res, and inches)

Cleaning CPU column

When you take the CPU column, it appears that it is also very noisy. As you can see from the diagram below, there are many types. There are 8 of Intel and 8 of AMD. Therefore, a separate column for CPU was separated as Intel other to put Intel i3, Intel i5, Intel i7, AMD processor and other Intel.

```
[ ] df['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 20: value count of CPU

First, in the CPU column, we can only train the model by CPU type, so we took only its name. It took only the first three words of the column.

```
[ ] df['Cpu Name'] =  df['Cpu'].apply(lambda x:" ".join(x.split()[0:3]))
```

```
[ ] df.head()
```

	Company	TypeName	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu Name
0	Apple	Ultrabook	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5
1	Apple	Ultrabook	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5
2	HP	Notebook	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5
3	Apple	Ultrabook	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7
4	Apple	Ultrabook	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5

Figure 21: CPU column cleaning

Then separate CPU types with a function as follows.

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

```
[ ] df['Cpu brand'] = df['Cpu Name'].apply(fetch_processor)
```

```
[ ] df.head()
```

	Company	TypeName	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu Name	Cpu brand
0	Apple	Ultrabook	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5	Intel Core i5
1	Apple	Ultrabook	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5	Intel Core i5
2	HP	Notebook	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5	Intel Core i5
3	Apple	Ultrabook	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7	Intel Core i7
4	Apple	Ultrabook	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5	Intel Core i5

Figure 22: CPU column cleaning (code 2)

In this case, the following bar chart shows the types of CPU and how many of them there are.

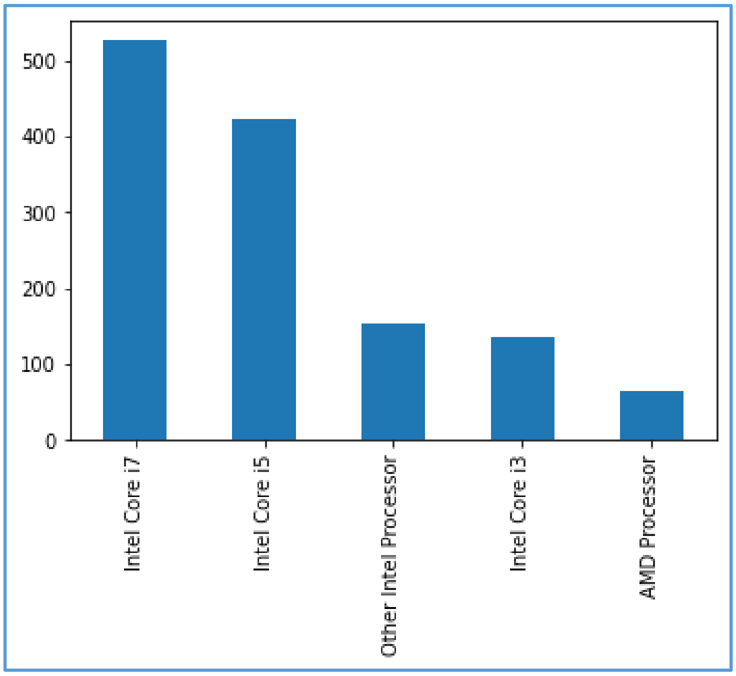


Figure 23: bar chart of CPU brands

This shows that Intel i7 exceeds 500 laptops in the data set. Intel i5s are second. They seem to exceed 400. This shows that the Intel i3 and other Intel processors are at the same level. AMD is the lowest among CPU types.

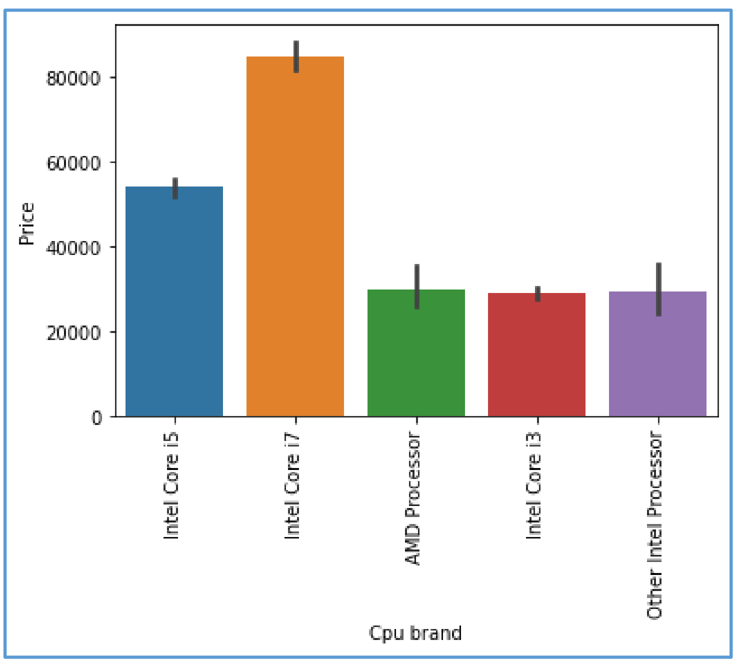
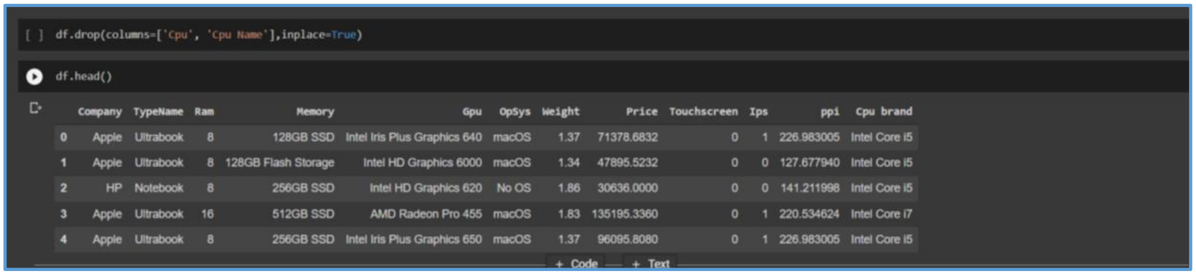


Figure 24: Price and CPU bar chart

The above graph shows the variation in laptop price depending on the type of CPU. It can be seen that the Intel i7 affects the price of the laptop here. Intel i5 shows the second highest value. Other types show a similar pattern.

Then the two columns CPU name and CPU were dropped from the dataset.



```
df.drop(columns=['Cpu', 'Cpu Name'],inplace=True)
```

```
df.head()
```

	Company	TypeName	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu brand
0	Apple	Ultrabook	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5
1	Apple	Ultrabook	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5
2	HP	Notebook	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5
3	Apple	Ultrabook	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7
4	Apple	Ultrabook	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5

Figure 25: data set (drop CPU and CPU name)

Cleaning Ram column

There is nothing much to do in this column. Because the word GB was removed from the above and made numeric.

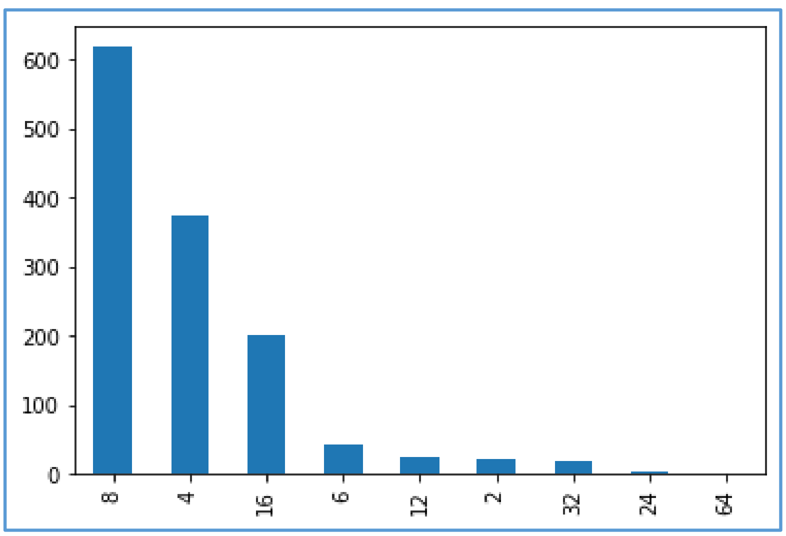


Figure 26: Ram value count

This ram column is 2, 4, 6, 8, 12, 16, 32, and 64. According to the picture above, there are many of them with 8 GB ram size. It exceeds the size of 600. The second is 4GB and the third is 16GB. Their sizes are between 300-400 and 200 respectively in the dataset. Others are minor.

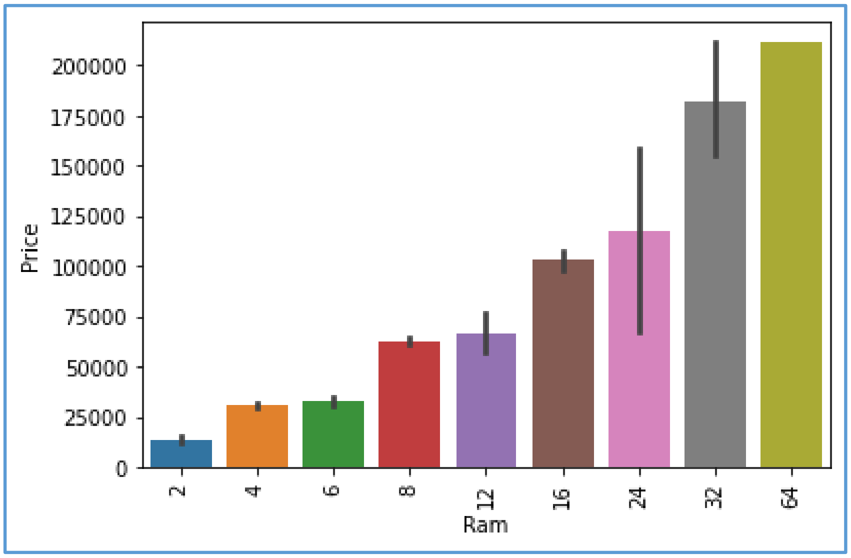
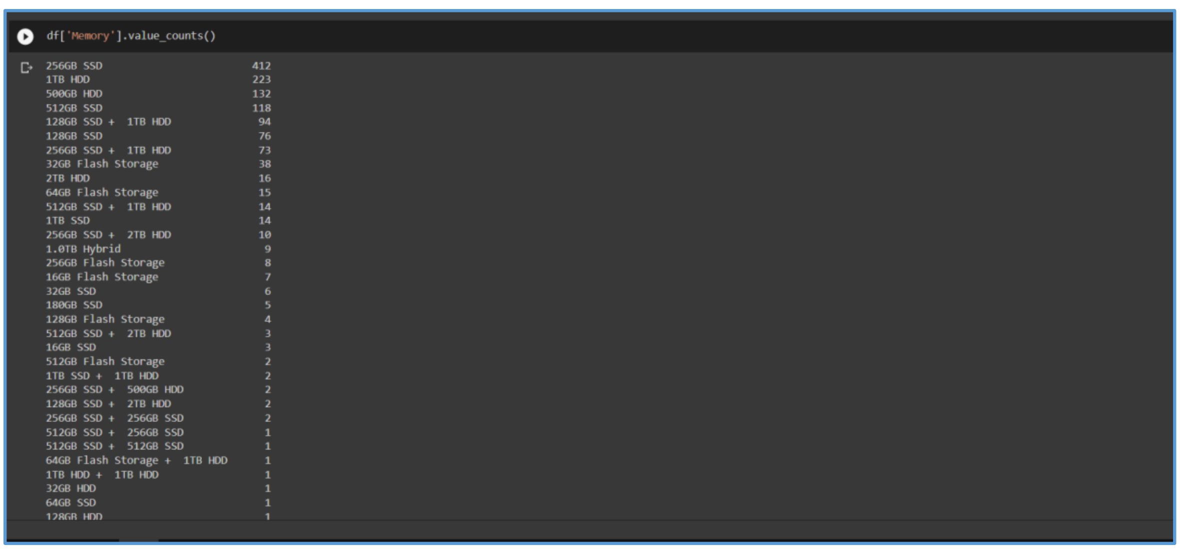


Figure 27: Price and ram (Bar chart)

As can be seen from this, the change in the price of the laptop can be indicated on the size of the ram. A laptop with 64GB ram seems to be very expensive. Others appear to be declining. It appears that the price is affected by the nature of the ram.

Cleaning Memory column



```
df['Memory'].value_counts()

256GB SSD                         412
1TB HDD                           223
500GB HDD                         132
512GB SSD                         118
128GB SSD +  1TB HDD               94
128GB SSD                          76
256GB SSD +  1TB HDD               73
32GB Flash Storage                 38
2TB HDD                            16
64GB Flash Storage                 15
512GB SSD +  1TB HDD               14
1TB SSD                            14
256GB SSD +  2TB HDD               10
1.0TB Hybrid                        9
256GB Flash Storage                 8
16GB Flash Storage                  7
32GB SSD                            6
180GB SSD                           5
128GB Flash Storage                 4
512GB SSD +  2TB HDD                3
16GB SSD                            3
512GB Flash Storage                 2
1TB SSD +  1TB HDD                  2
256GB SSD +  500GB HDD              2
128GB SSD +  2TB HDD                2
256GB SSD +  256GB SSD              2
512GB SSD +  256GB SSD              1
512GB SSD +  512GB SSD              1
64GB Flash Storage +  1TB HDD       1
1TB HDD +  1TB HDD                  1
32GB HDD                            1
64GB SSD                            1
128GB HDD                           1
```

Figure 28: Value count of memory

As can be seen from this, there are many types of memory. Therefore, they had to be divided into separate columns.

```python
df['Memory'] = df['Memory'].astype(str).replace('\.0', '', regex=True)
df["Memory"] = df["Memory"].str.replace('GB', '')
df["Memory"] = df["Memory"].str.replace('TB', '000')
new = df["Memory"].str.split("+", n = 1, expand = True)

df["first"]= new[0]
df["first"]=df["first"].str.strip()

df["second"]= new[1]

df["Layer1HDD"] = df["first"].apply(lambda x: 1 if "HDD" in x else 0)
df["Layer1SSD"] = df["first"].apply(lambda x: 1 if "SSD" in x else 0)
df["Layer1Hybrid"] = df["first"].apply(lambda x: 1 if "Hybrid" in x else 0)
df["Layer1Flash_Storage"] = df["first"].apply(lambda x: 1 if "Flash Storage" in x else 0)

df['first'] = df['first'].str.replace(r'\D', '')

df["second"].fillna("0", inplace = True)

df["Layer2HDD"] = df["second"].apply(lambda x: 1 if "HDD" in x else 0)
df["Layer2SSD"] = df["second"].apply(lambda x: 1 if "SSD" in x else 0)
df["Layer2Hybrid"] = df["second"].apply(lambda x: 1 if "Hybrid" in x else 0)
df["Layer2Flash_Storage"] = df["second"].apply(lambda x: 1 if "Flash Storage" in x else 0)

df['second'] = df['second'].str.replace(r'\D', '')

df["first"] = df["first"].astype(int)
df["second"] = df["second"].astype(int)

df["HDD"]=(df["first"]*df["Layer1HDD"]+df["second"]*df["Layer2HDD"])
df["SSD"]=(df["first"]*df["Layer1SSD"]+df["second"]*df["Layer2SSD"])
df["Hybrid"]=(df["first"]*df["Layer1Hybrid"]+df["second"]*df["Layer2Hybrid"])
df["Flash_Storage"]=(df["first"]*df["Layer1Flash_Storage"]+df["second"]*df["Layer2Flash_Storage"])

df.drop(columns=['first', 'second', 'Layer1HDD', 'Layer1SSD', 'Layer1Hybrid',
       'Layer1Flash_Storage', 'Layer2HDD', 'Layer2SSD', 'Layer2Hybrid',
       'Layer2Flash_Storage'],inplace=True)
```

Figure 29: Creating columns of Memory

Here, HDD, SSD, Hybrid, Flash Storage are separated separately. It is also divided into 1 and 0.

```python
df.head()
```

	Company	TypeName	Ram	Memory	Gpu	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Hybrid	Flash_Storage
0	Apple	Ultrabook	8	128 SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5	0	128	0	0
1	Apple	Ultrabook	8	128 Flash Storage	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5	0	0	0	128
2	HP	Notebook	8	256 SSD	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5	0	256	0	0
3	Apple	Ultrabook	16	512 SSD	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7	0	512	0	0
4	Apple	Ultrabook	8	256 SSD	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5	0	256	0	0

Figure 30: creating columns of memory (data set)

Then the memory column was dropped from the data set.

```python
df.drop(columns=['Memory'],inplace=True)
```

```python
df.head()
```

	Company	TypeName	Ram	Gpu	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Hybrid	Flash_Storage
0	Apple	Ultrabook	8	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5	0	128	0	0
1	Apple	Ultrabook	8	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5	0	0	0	128
2	HP	Notebook	8	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5	0	256	0	0
3	Apple	Ultrabook	16	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7	0	512	0	0
4	Apple	Ultrabook	8	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5	0	256	0	0

Figure 31: data set (drop column of memory)

Cleaning GPU column

```python
df['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
```

```python
df['Gpu brand'] = df['Gpu'].apply(lambda x:x.split()[0])
```

Figure 32: value count of GPU

As can be seen from this, GPUs also have a large amount of data. But for us, only the type is enough, so we entered the type and created a column.

```
df.head()
```

	Company	TypeName	Ram	Gpu	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Hybrid	Flash_Storage	Gpu brand
0	Apple	Ultrabook	8	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5	0	128	0	0	Intel
1	Apple	Ultrabook	8	Intel HD Graphics 6000	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5	0	0	0	128	Intel
2	HP	Notebook	8	Intel HD Graphics 620	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5	0	256	0	0	Intel
3	Apple	Ultrabook	16	AMD Radeon Pro 455	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7	0	512	0	0	AMD
4	Apple	Ultrabook	8	Intel Iris Plus Graphics 650	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5	0	256	0	0	Intel

```
[ ] df['Gpu brand'].value_counts()

Intel     722
Nvidia    400
AMD       180
ARM         1
Name: Gpu brand, dtype: int64

[ ] df = df[df['Gpu brand'] != 'ARM']
```

Figure 33: data set (creating GPU)

As you can see the gpu types are mentioned in the above image. It is a data containing only one type. It cannot affect our model. So it was removed.

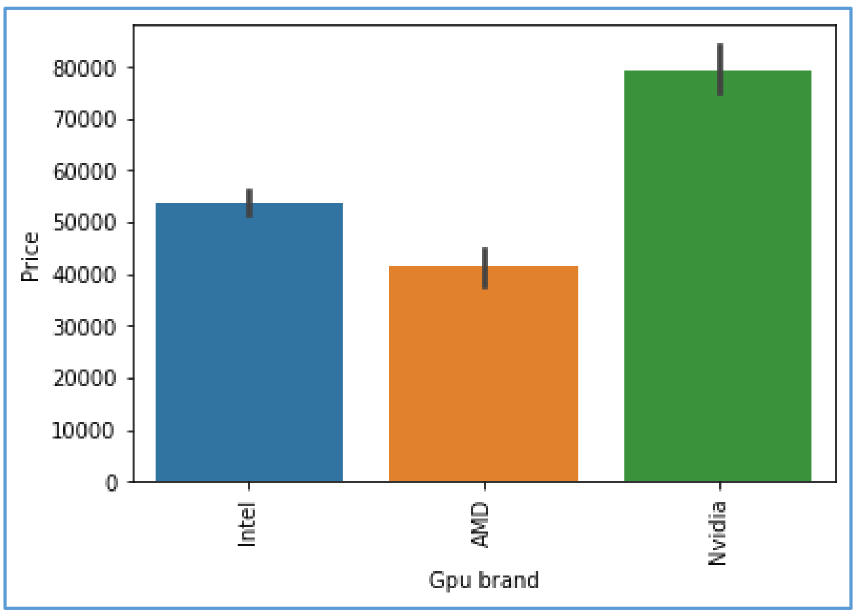


Figure 34: Price and GPU bar chart

As seen here, laptops with NVidia are expensive. Second is Intel and third is AMD. The gpu also affects the price.

```
[ ] df.drop(columns=['Gpu'],inplace=True)
```

```
[ ] df.head()
```

	Company	TypeName	Ram	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Hybrid	Flash_Storage	Gpu brand
0	Apple	Ultrabook	8	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5	0	128	0	0	Intel
1	Apple	Ultrabook	8	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5	0	0	0	128	Intel
2	HP	Notebook	8	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5	0	256	0	0	Intel
3	Apple	Ultrabook	16	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7	0	512	0	0	AMD
4	Apple	Ultrabook	8	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5	0	256	0	0	Intel

Figure 35: dropping gpu column

Cleaning Operating System

```
[ ] df['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 36: value count of operating system

As shown in the image above, there are many operating systems. Therefore, the most common windows operating system is put separately, mac is separately and others are put in the OS column as others.

```
[ ] def cat_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'
```

```
[ ] df['os'] = df['OpSys'].apply(cat_os)
```

```
df.head()
```

	Company	TypeName	Ram	OpSys	Weight	Price	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Hybrid	Flash_Storage	Gpu brand	os
0	Apple	Ultrabook	8	macOS	1.37	71378.6832	0	1	226.983005	Intel Core i5	0	128	0	0	Intel	Mac
1	Apple	Ultrabook	8	macOS	1.34	47895.5232	0	0	127.677940	Intel Core i5	0	0	0	128	Intel	Mac
2	HP	Notebook	8	No OS	1.86	30636.0000	0	0	141.211998	Intel Core i5	0	256	0	0	Intel	Others/No OS/Linux
3	Apple	Ultrabook	16	macOS	1.83	135195.3360	0	1	220.534624	Intel Core i7	0	512	0	0	AMD	Mac
4	Apple	Ultrabook	8	macOS	1.37	96095.8080	0	1	226.983005	Intel Core i5	0	256	0	0	Intel	Mac

```
[ ] df.drop(columns=['OpSys'],inplace=True)
```

Figure 37: creating parts of windows

Then opsys dropped the column from the dataset.

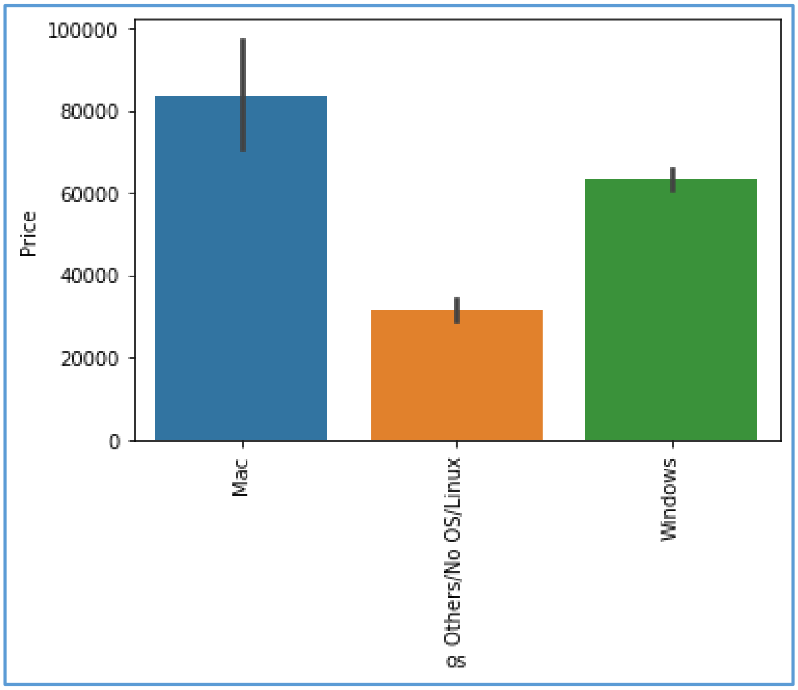
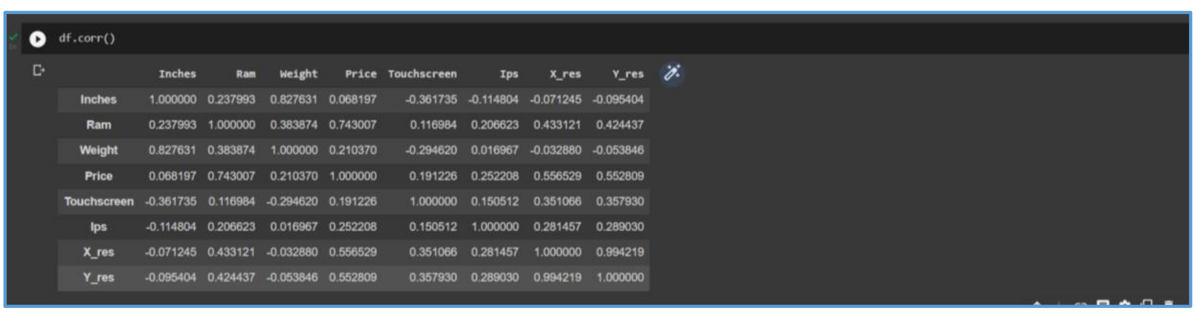


Figure 38: operating System (Bar chart)

The chart above shows the relationship between price and OS. Here the mac type ones are more expensive and the second one is the windows os laptop.

Correlations

Correlation can be taken as a very important thing when creating a prediction model. It can see how other data affects the data we predict.



```
df.corr()
```

	Inches	Ram	Weight	Price	Touchscreen	Ips	X_res	Y_res
Inches	1.000000	0.237993	0.827631	0.068197	-0.361735	-0.114804	-0.071245	-0.095404
Ram	0.237993	1.000000	0.383874	0.743007	0.116984	0.206623	0.433121	0.424437
Weight	0.827631	0.383874	1.000000	0.210370	-0.294620	0.016967	-0.032880	-0.053846
Price	0.068197	0.743007	0.210370	1.000000	0.191226	0.252208	0.556529	0.552809
Touchscreen	-0.361735	0.116984	-0.294620	0.191226	1.000000	0.150512	0.351066	0.357930
Ips	-0.114804	0.206623	0.016967	0.252208	0.150512	1.000000	0.281457	0.289030
X_res	-0.071245	0.433121	-0.032880	0.556529	0.351066	0.281457	1.000000	0.994219
Y_res	-0.095404	0.424437	-0.053846	0.552809	0.357930	0.289030	0.994219	1.000000

Figure 39: correlations

The above image shows the correlation between all the data in the dataset. Regarding the model, what is important for us is how other data affects the price.

```
[ ] df.corr()['Price']

Ram              0.742905
Weight           0.209867
Price            1.000000
Touchscreen      0.192917
Ips              0.253320
ppi              0.475368
HDD             -0.096891
SSD              0.670660
Hybrid           0.007942
Flash_Storage   -0.040067
Name: Price, dtype: float64
```

Figure 40: correlation (code 2)

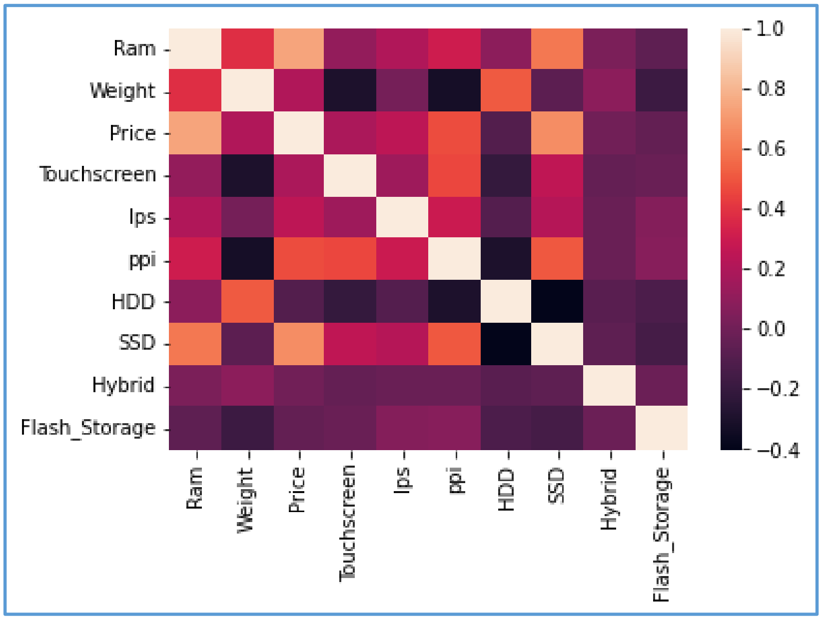


Figure 41: heat map correlation

The dataset created to train the model after all these cleaning works is given below.

	Company	TypeName	Ram	Weight	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Hybrid	Flash_Storage	Gpu brand	os
0	Apple	Ultrabook	8	1.37	0	1	226.983005	Intel Core i5	0	128	0	0	Intel	Mac
1	Apple	Ultrabook	8	1.34	0	0	127.677940	Intel Core i5	0	0	0	128	Intel	Mac
2	HP	Notebook	8	1.86	0	0	141.211998	Intel Core i5	0	256	0	0	Intel	Others/No OS/Linux
3	Apple	Ultrabook	16	1.83	0	1	220.534624	Intel Core i7	0	512	0	0	AMD	Mac
4	Apple	Ultrabook	8	1.37	0	1	226.983005	Intel Core i5	0	256	0	0	Intel	Mac
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1298	Lenovo	2 in 1 Convertible	4	1.80	1	1	157.350512	Intel Core i7	0	128	0	0	Intel	Windows
1299	Lenovo	2 in 1 Convertible	16	1.30	1	1	276.053530	Intel Core i7	0	512	0	0	Intel	Windows
1300	Lenovo	Notebook	2	1.50	0	0	111.935204	Other Intel Processor	0	0	0	64	Intel	Windows
1301	HP	Notebook	6	2.19	0	0	100.454670	Intel Core i7	1000	0	0	0	AMD	Windows
1302	Asus	Notebook	4	2.20	0	0	100.454670	Other Intel Processor	500	0	0	0	Intel	Windows

1302 rows × 14 columns

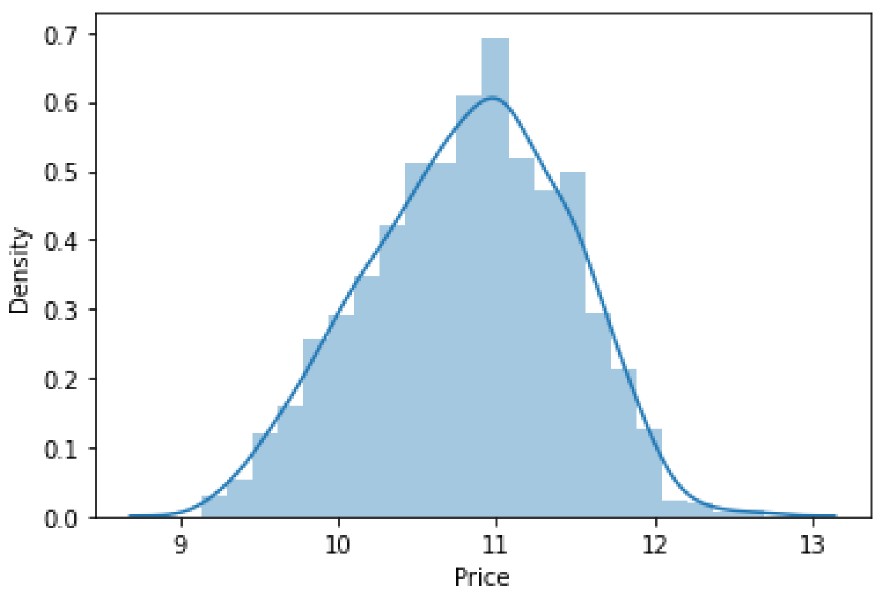
Figure 42: final data set

4. Model Training

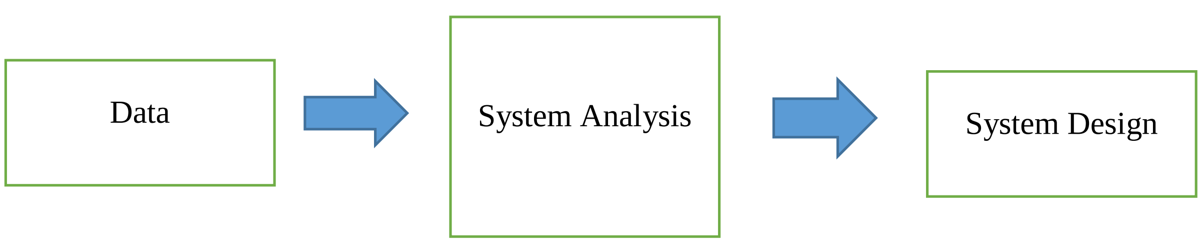
4.1. Log normal transformation

Transformation using logarithms we can observe that the target variable's distribution is skewed to the right. The algorithm's performance will improve by changing it to a normal distribution. As you can see below, we converted the logarithm of the transform values into a normal distribution. We will thus take the price's logarithm and express the exponent when showing the findings while separating the dependent and independent variables.

Figure 43: ng.log code



Machine Learning Models for Predicting Laptop Prices Now that our data has been prepared, we know more about the dataset. In order to identify the optimal method with the best hyper parameter for maximum accuracy, let's start with the machine learning model. Library import.



4.2.Import libraries

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score,mean_absolute_error
```

Figure 44: importing libraries

Column Transformer enables us to transform a specific set of columns. It helps us apply multiple transformations to multiple columns with a single fit () or fit_transform () statement.

A machine learning **pipeline** is a predetermined series of operations carried out to create, implement, and track a machine learning model. A machine learning model's development, training, implementation, and monitoring are all mapped using this method. The procedure is frequently automated using it.

For use in machine learning, **one-hot Encoder** is the process of transforming category data into numerical data.

A crucial indicator for assessing the effectiveness of a regression machine learning model is the **R2 score**. The coefficient of determination, which is also known by the pronunciation R squared, it measures the variation in forecasts that the data set can account for.

The average discrepancy between the computed and real values is calculated using the **mean absolute error**.

4.3.X and Y

```
X = df.drop(columns=['Price'])
y = np.log(df['Price'])
```

```
X
```

	Company	TypeName	Ram	Weight	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Hybrid	Flash_Storage	Gpu brand	os
0	Apple	Ultrabook	8	1.37	0	1	226.983005	Intel Core i5	0	128	0	0	Intel	Mac
1	Apple	Ultrabook	8	1.34	0	0	127.677940	Intel Core i5	0	0	0	128	Intel	Mac
2	HP	Notebook	8	1.86	0	0	141.211998	Intel Core i5	0	256	0	0	Intel	Others/No OS/Linux
3	Apple	Ultrabook	16	1.83	0	1	220.534624	Intel Core i7	0	512	0	0	AMD	Mac
4	Apple	Ultrabook	8	1.37	0	1	226.983005	Intel Core i5	0	256	0	0	Intel	Mac
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1298	Lenovo	2 in 1 Convertible	4	1.80	1	1	157.350512	Intel Core i7	0	128	0	0	Intel	Windows
1299	Lenovo	2 in 1 Convertible	16	1.30	1	1	276.053530	Intel Core i7	0	512	0	0	Intel	Windows
1300	Lenovo	Notebook	2	1.50	0	0	111.935204	Other Intel Processor	0	0	0	64	Intel	Windows
1301	HP	Notebook	6	2.19	0	0	100.454670	Intel Core i7	1000	0	0	0	AMD	Windows
1302	Asus	Notebook	4	2.20	0	0	100.454670	Other Intel Processor	500	0	0	0	Intel	Windows

1302 rows × 14 columns

Figure 45: X and Y values

X is assigned feature variable by dropping "price" column because the column is the targeted variable Y.

```
y

0       11.175755
1       10.776777
2       10.329931
3       11.814476
4       11.473101
          ...
1298    10.433899
1299    11.288115
1300     9.409283
1301    10.614129
1302     9.886358
Name: Price, Length: 1302, dtype: float64
```

Figure 46: Y

4.4.Implementing the pipe line

```python
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = RandomForestRegressor(n_estimators=100,
                              random_state=3,
                              max_samples=0.5,
                              max_features=0.75,
                              max_depth=15)


pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)
```

Figure 47: pipeline

We are currently developing a pipeline to streamline the training and testing process. We first use a column transformer to encode the categorical variables, which is the first step. Then we create an object in our algorithm and pass step two to fillinin. Using pipeline objects, we predict scores on new data and show accuracy.

4.5.Selecting Best model

```python
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor,AdaBoostRegressor,ExtraTreesRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
```

Figure 48: import models

I have imported many models here. But I chose only 4 of these.

LassoRegression

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = Lasso(alpha=0.001)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

Figure 49: Lasso regression

First choose the lasso model. The R2 score was 0.8071853945317105 and the MAE value was 0.21114361613472565.

Decision Tree

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = DecisionTreeRegressor(max_depth=8)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

Figure 50: Decision tree

Then I choose the decision tree model. The R2_score was 0.8466456692979233 and the MAE value was 0.1806340977609143.

LinearRegression

```
step1 = ColumnTransformer(transformers=[
        ('col_tnf', OneHotEncoder(sparse=False, drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = LinearRegression()

pipe = Pipeline([
                ('step1', step1),
                ('step2', step2)
])
pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score', r2_score(y_test, y_pred))
print('MAE', mean_absolute_error(y_test, y_pred))
```

Figure 51: Linear regression

Then I choose the linear regression model. The R2_Score was 0.8073277448418521 and the MAE value was 0.21017827976429174.

Random Forest

```
[ ] step1 = ColumnTransformer(transformers=[
        ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
    ],remainder='passthrough')

    step2 = RandomForestRegressor(n_estimators=100,
                                  random_state=3,
                                  max_samples=0.5,
                                  max_features=0.75,
                                  max_depth=15)

    pipe = Pipeline([
        ('step1',step1),
        ('step2',step2)
    ])

    pipe.fit(X_train,y_train)

    y_pred = pipe.predict(X_test)

    print('R2 score',r2_score(y_test,y_pred))
    print('MAE',mean_absolute_error(y_test,y_pred))
```

Figure 52: Random Forest

Then I choose the random forest model. The R2_Score was 0.8873402378382488 and the MAE value was 0.15860130110457718.

Machine Learning Model for Predicting Laptop Prices We changed the index of the encrypted columns and the means to send the remaining numeric columns as-is in the first phase of category coding. Random Forest is my absolute favorite and has the finest accuracy I've experienced. But if you change the algorithm and its inputs, you may use this code again. A random woodland is displayed. Hyper parameter matching may be done using GridsearchCV or Random Search CV. The characteristics can be expanded as well, but the random forest is unaffected.

5. Creating GUI

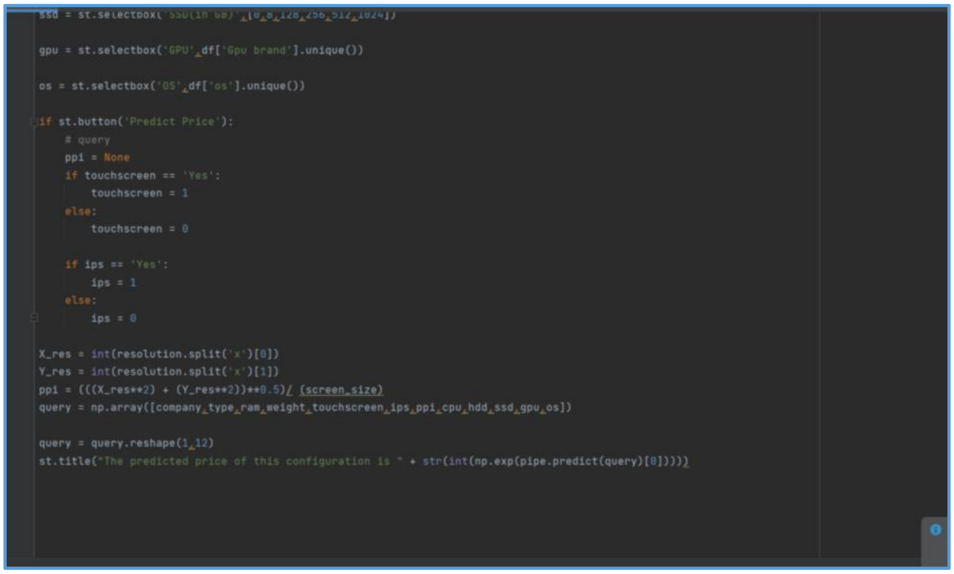
I used python to create the GUI. Also used libraries like numpy and pandas.

```python
import streamlit as st
import pickle
import numpy as np


# import the model
pipe = pickle.load(open('pipe.pkl', 'rb'))
df = pickle.load(open('df.pkl', 'rb'))

st.title("laptop Price Predictor")

# Brand
company = st.selectbox('Brand', df['Company'].unique())

# type of laptop
type = st.selectbox('Type', df['TypeName'].unique())

#Ram
ram = st.selectbox('RAM(in GB)', [2,4,6,8,12,24,32,64])

#weight
weight = st.number_input('weight of the laptop')

#Touchscreen
touchscreen = st.selectbox('Touchscreen',['No','Yes'])

#IPS
ips = st.selectbox('IPS',['No','Yes'])

# screen size
```

Figure 53: python code 1

```python
screen_size = st.number_input('Screen Size')

# resolution
resolution = st.selectbox('Screen Resolution',['1920x1080','1366x768','1600x900','3840x2160',
                                               '3200x1800','2880x1800','2560x1600','2560x1440','2304x1440'])

#cpu
cpu = st.selectbox('CPU',df['Cpu brand'].unique())

hdd = st.selectbox('HDD(in GB)',[0,128,256,512,1024,2048])

ssd = st.selectbox('SSD(in GB)',[0,8,128,256,512,1024])

gpu = st.selectbox('GPU',df['Gpu brand'].unique())

os = st.selectbox('OS',df['os'].unique())

if st.button('Predict Price'):
    # query
    ppi = None
    if touchscreen == 'Yes':
        touchscreen = 1
    else:
        touchscreen = 0

    if ips == 'Yes':
        ips = 1
    else:
        ips = 0
```

Figure 54: python code 2



```
ssd = st.selectbox('SSD(in GB)',[0,8,128,256,512,1024])

gpu = st.selectbox('GPU',df['Gpu brand'].unique())

os = st.selectbox('OS',df['os'].unique())

if st.button('Predict Price'):
    # query
    ppi = None
    if touchscreen == 'Yes':
        touchscreen = 1
    else:
        touchscreen = 0

    if ips == 'Yes':
        ips = 1
    else:
        ips = 0

X_res = int(resolution.split('x')[0])
Y_res = int(resolution.split('x')[1])
ppi = (((X_res**2) + (Y_res**2))**0.5)/ (screen_size)
query = np.array([company,type,ram,weight,touchscreen,ips,ppi,cpu,hdd,ssd,gpu,os])

query = query.reshape(1,12)
st.title("The predicted price of this configuration is " + str(int(np.exp(pipe.predict(query)[0]))))
```

Figure 55: python code3

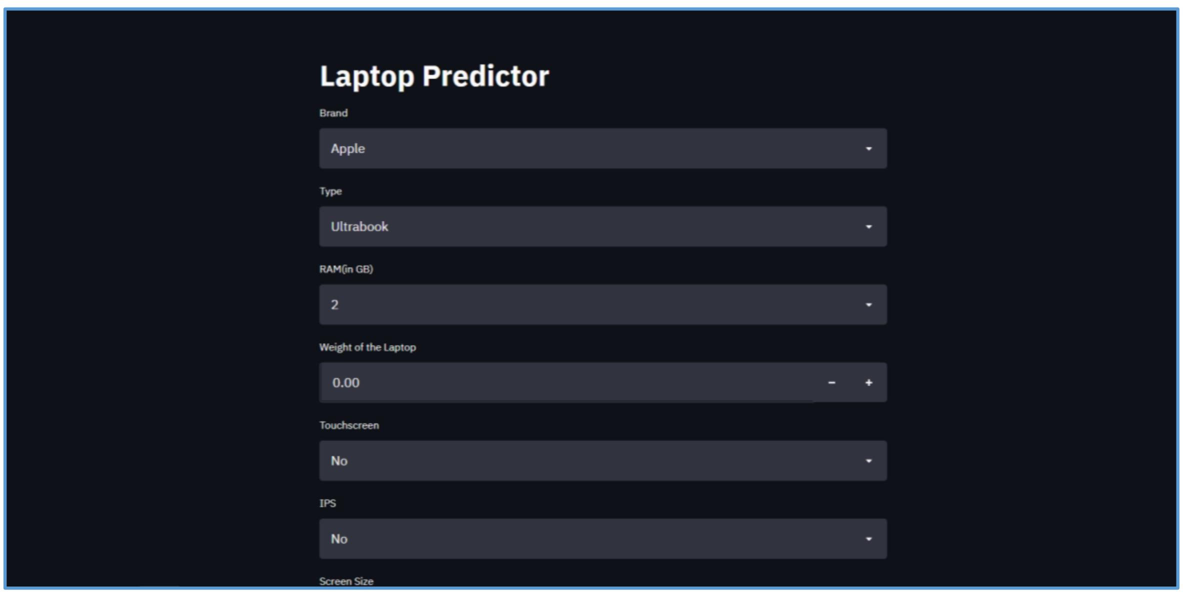


Figure 56: final project

First we load the previously saved model and data frame. Then, depending on the training data columns, we design an HTML form with each user input field. We set the first parameter in the category columns as the name of the input field, and the second parameter as the select, which is just the individual categories in the dataset. We offer users added value or devaluation in the digital world. When the prediction node is active, it creates a 2D input list, encodes the variable, and sends it to the model to display the prediction on the screen.

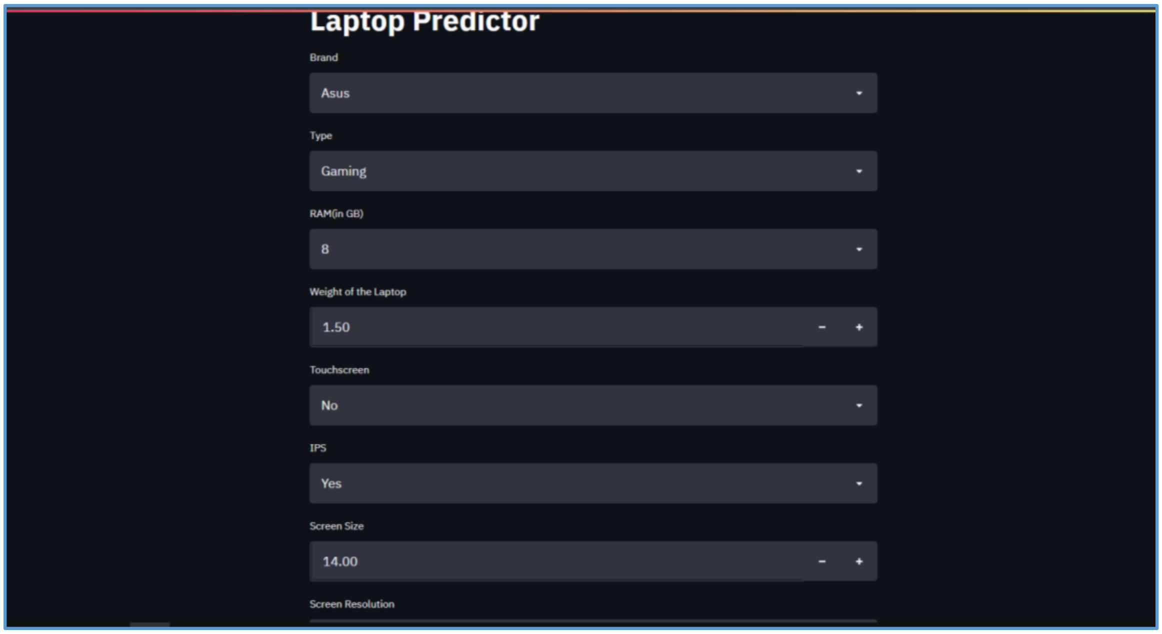


Figure 57: example predict code1

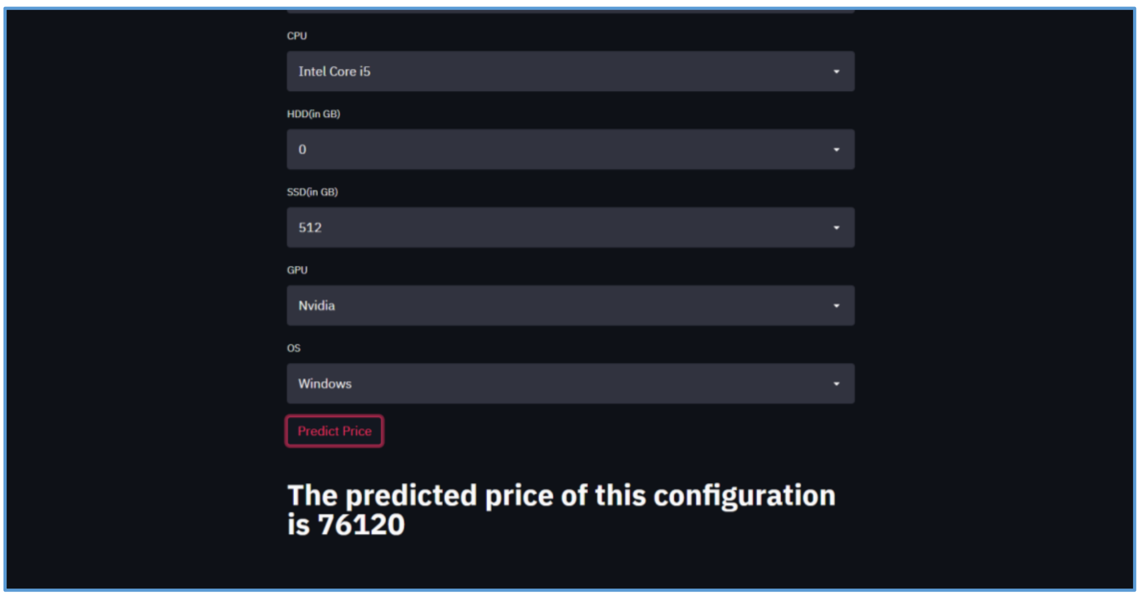


Figure 58: example predict code2

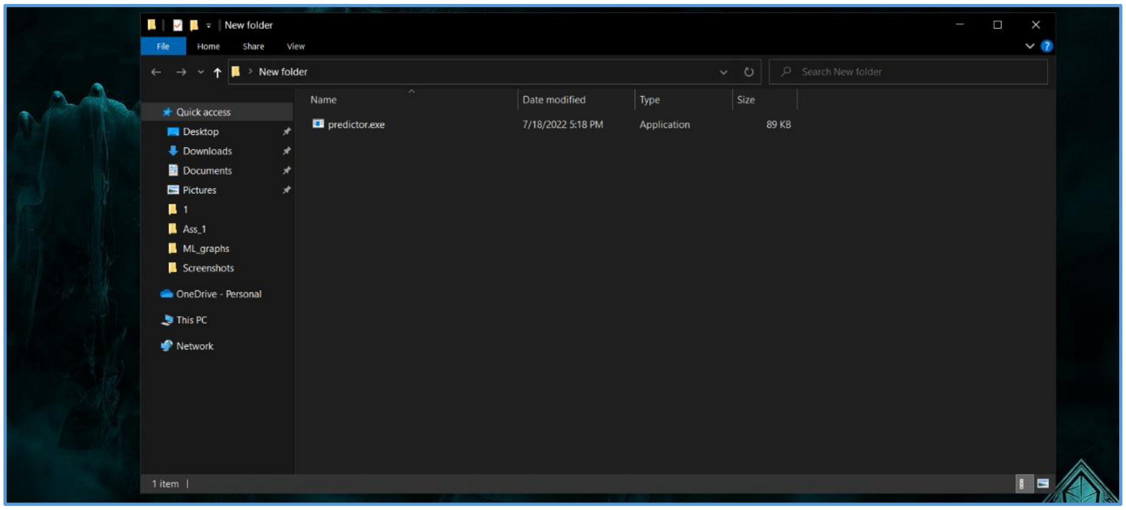


Figure 59: system EXE file

After all the work thus converted to an EXE file.

Conclusion

The machine learning model training portion of this work is complete. I've selected a laptop price forecasting tool for this. People begin working from home when there are pandemic circumstances, which is the cause for this. Then, due to the popularity of laptops and the need for new ones, this software has been developed to enable users to quickly purchase the laptops they desire from an online retailer. I used a dataset containing information about laptops here. Company, Type Name, Inches, Screen Resolution, CPU, RAM, Memory, GPU, Operating System, Weight, and Price are the columns shown here. Here, decision tree, lasso, random forest, and linear regression are all employed as machine learning training methods. Random forest was chosen as the best model. Then a GUI was also created for the model.