Project Overview:

SmartTech Co. has partnered with our data science team to develop a robust machine learning model that predicts laptop prices accurately. As the market for laptops continues to expand with a myriad of brands and specifications, having a precise pricing model becomes crucial for both consumers and manufacturers.

The Dataset that has been provided to us consists of Laptop details and their Prices.

Exploratory Data Analysis

1. Analysing the Dataset:
   Importing the dataset into Jupyter Notebook we see the all the Column/fields defined in the datset:

Importing the required Libraries

```
[2]: import numpy as np
     import pandas as pd
     np.set_printoptions(suppress=True)
     import matplotlib.pyplot as pyplt
```

```
[3]: laptop = pd.read_csv(r'C:\Users\nsany\Downloads\laptop\laptop.csv')
     laptop.head()
```

[3]:

	Unnamed: 0.1	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	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	1.0	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	2.0	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	3.0	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	4.0	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

2. We check for the Info of the dataset to get a view of the content of the dataset:
   The Datatypes of each column can be fetched and we see that apart form Price all the other fields are object data type.

```
[4]: laptop.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0.1      1303 non-null   int64
 1   Unnamed: 0        1273 non-null   float64
 2   Company           1273 non-null   object
 3   TypeName          1273 non-null   object
 4   Inches            1273 non-null   object
 5   ScreenResolution  1273 non-null   object
 6   Cpu               1273 non-null   object
 7   Ram               1273 non-null   object
 8   Memory            1273 non-null   object
 9   Gpu               1273 non-null   object
 10  OpSys             1273 non-null   object
 11  Weight            1273 non-null   object
 12  Price             1273 non-null   float64
dtypes: float64(2), int64(1), object(10)
memory usage: 132.5+ KB
```

3. Total number of columns in the dataset are 13. Out of that index 0 and index 1 column are not required for our analysis and may cause issue in our analysis. Hence, we drop them.
4. After those columns have been dropped the new trimmed dataset appears to be:

```
[8]: laptop.head()
```

[8]:

	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	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
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	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	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	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

5. We check for other Data issues: NULL values and then we drop the NULL values from the columns.

```
[9]: # Checking for NULL entries in the columns
     laptop.isnull().sum()

[9]: Company             30
     TypeName            30
     Inches              30
     ScreenResolution    30
     Cpu                 30
     Ram                 30
     Memory              30
     Gpu                 30
     OpSys               30
     Weight              30
     Price               30
     dtype: int64

[10]: laptop = laptop.dropna()
```

6. Now we start working on each of the columns to convert them from object to Float or integer based on the type of values they store.

```
[15]: laptop.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1273 entries, 0 to 1272
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Company           1273 non-null   object
 1   TypeName          1273 non-null   object
 2   Inches            1273 non-null   object
 3   ScreenResolution  1273 non-null   object
 4   Cpu               1273 non-null   object
 5   Ram               1273 non-null   object
 6   Memory            1273 non-null   object
 7   Gpu               1273 non-null   object
 8   OpSys             1273 non-null   object
 9   Weight            1273 non-null   object
 10  Price             1273 non-null   float64
dtypes: float64(1), object(10)
memory usage: 109.5+ KB

[16]: #Converting the datatyoes
laptop.Inches = pd.to_numeric(laptop['Inches'], errors='coerce')
```

7. First, we take the 'Inches' column. And convert it to numeric.

```
[17]: laptop.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1273 entries, 0 to 1272
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Company           1273 non-null   object
 1   TypeName          1273 non-null   object
 2   Inches            1272 non-null   float64
 3   ScreenResolution  1273 non-null   object
 4   Cpu               1273 non-null   object
 5   Ram               1273 non-null   object
 6   Memory            1273 non-null   object
 7   Gpu               1273 non-null   object
 8   OpSys             1273 non-null   object
 9   Weight            1273 non-null   object
 10  Price             1273 non-null   float64
dtypes: float64(2), object(9)
memory usage: 109.5+ KB
```

8. Then we move on the RAM column to convert it into numeric. Here we remove 'GB'

Removing 'GB' from the column to convert it into Float

```
[19]: laptop.Ram = pd.to_numeric(laptop.Ram.str.replace('GB',''))
```

```
[20]: laptop.head()
```

[20]:

	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	1.37kg	71378.6832
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	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	1.86kg	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	1.83kg	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	1.37kg	96095.8080

9. The new dataset become:

```
[21]: laptop.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1273 entries, 0 to 1272
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Company           1273 non-null   object
 1   TypeName          1273 non-null   object
 2   Inches            1272 non-null   float64
 3   ScreenResolution  1273 non-null   object
 4   Cpu               1273 non-null   object
 5   Ram               1273 non-null   int64
 6   Memory            1273 non-null   object
 7   Gpu               1273 non-null   object
 8   OpSys             1273 non-null   object
 9   Weight            1273 non-null   object
 10  Price             1273 non-null   float64
dtypes: float64(2), int64(1), object(8)
memory usage: 109.5+ KB
```

10. Converting the weight column to numeric:

Converting Weight Column into Float

[32]:
```python
laptop.Weight = pd.to_numeric(laptop['Weight'].str.replace('kg', ''))
```

[33]:
```python
laptop.head()
```

[33]:

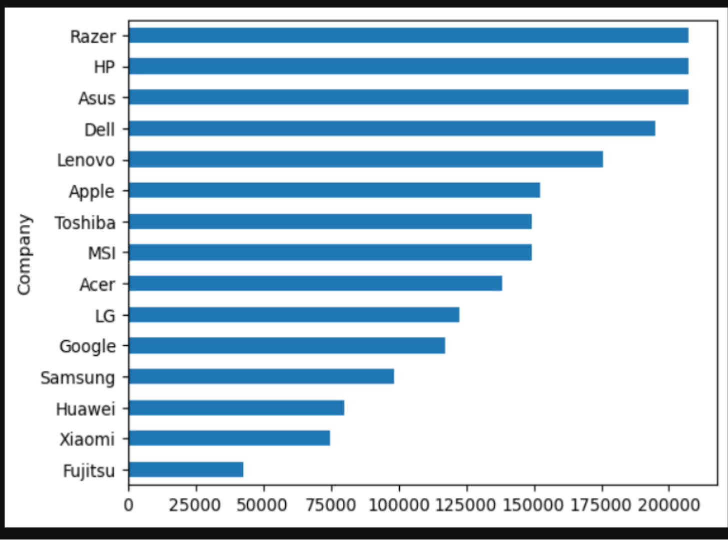
	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	71378.6832
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel 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

11. Grouping of Laptops based on Company Brand and their respective Price:

Understanding the relationship between Comapny Brand and Price

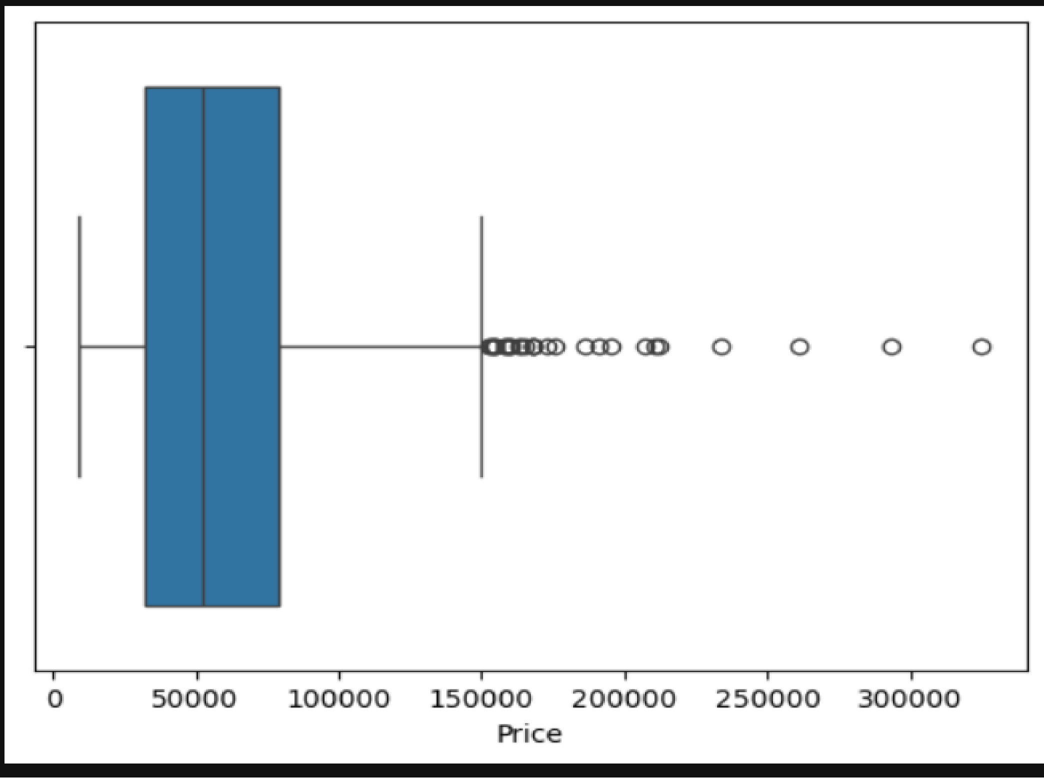
[317]:
```python
laptop.groupby('Company')[ 'Price'].max().sort_values().plot.barh()
```

[317]: <Axes: ylabel='Company'>



12. Checking for Outliers:

```
[37]: import seaborn as sns
      sns.boxplot(x=laptop.Price)

[37]: <Axes: xlabel='Price'>
```



The point of 200,000 can be considered as the outlier. Hence, we modify our dataset based on that.

```
[38]: #Identifying the outliers
      # calculate the quartiles
      q25, q50, q75 = np.percentile(laptop.Price, [25, 50, 75])

      iqr = q75 - q25

      # calculate the min and max
      min_price = q25 - 1.5*iqr
      max_price = q75 + 1.5*iqr

      # show calculations
      min_price, q25, q50, q75, max_price

[38]: (-39213.28079999999, 31914.72, 52161.12, 79333.3872, 150461.38799999998)
```

The lower limit of outlier is in negative. Therefore, we can ignore that and our focus will be on the upper limit outliers.

Imputing the outliers with the Minimum Value

```
[42]: min_Price_val = laptop[laptop.Price > 200000].Price.min()
      min_Price_val

[42]: 207259.2

[43]: laptop.Price = np.where(laptop.Price > 200000, min_Price_val, laptop.Price)
      laptop.Price

[43]: 0        71378.6832
      1        47895.5232
      2        30636.0000
      3       135195.3360
      4        96095.8080
                 ...
      1268     33992.6400
      1269     79866.7200
      1270     12201.1200
      1271     40705.9200
      1272     19660.3200
      Name: Price, Length: 1273, dtype: float64
```

We imputed those outliers with that of the minimum value among with outliers.

13. Once we have dealt with the outliers, do a correlation test between Weight, RAM and Inches with Price. Among the three, Inches had the correlation nearest to '0'. Correlation nearest to zero would mean the co-relationship is very weak. Hence, we can drop the column.

```
[409]: #Correlation between Price and Weight, RAM, INCHES
       print(f'Correlation between Price and Weight: {laptop["Weight"].corr(laptop["Price"])}')
       print(f'Correlation between Price and Ram: {laptop["Ram"].corr(laptop["Price"])}')
       print(f'Correlation between Price and Inches: {laptop["Inches"].corr(laptop["Price"])}')

       Correlation between Price and Weight: 0.17191469014521443
       Correlation between Price and Ram: 0.6827054202956128
       Correlation between Price and Inches: 0.04042081780423125
```

14. We do the same with the rest of the columns in the Dataset.

15. We re-order the columns in order to put ‘Price’ column at the end.

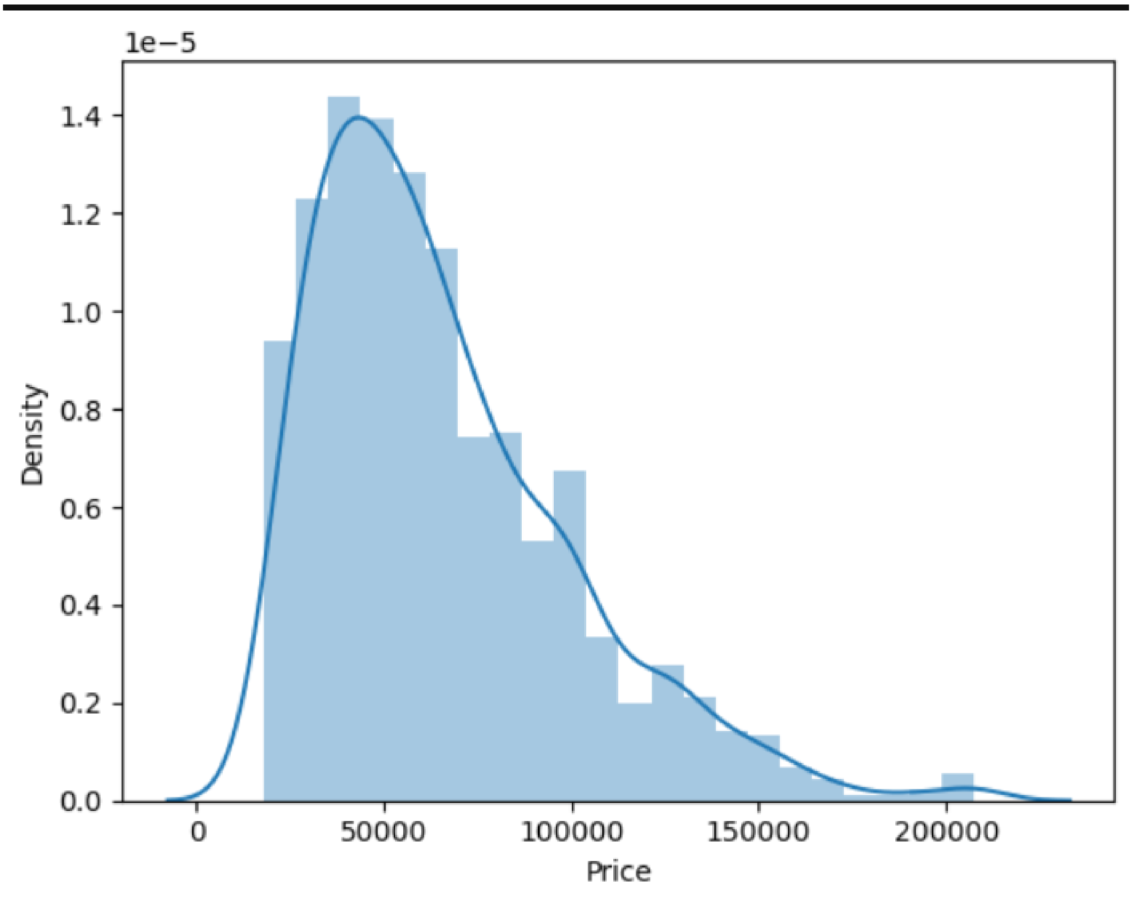
Reordering the Columns in the Laptop DataFrame

```
[67]: laptop_order = ['Company', 'TypeName', 'ScreenResolution', 'Cpu', 'Ram', 'Gpu', 'OpSys', 'Weight', 'MemoryStorage', 'MemoryType', 'Price']
      laptop = laptop[laptop_order]

[68]: laptop.head()
```

[68]:

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

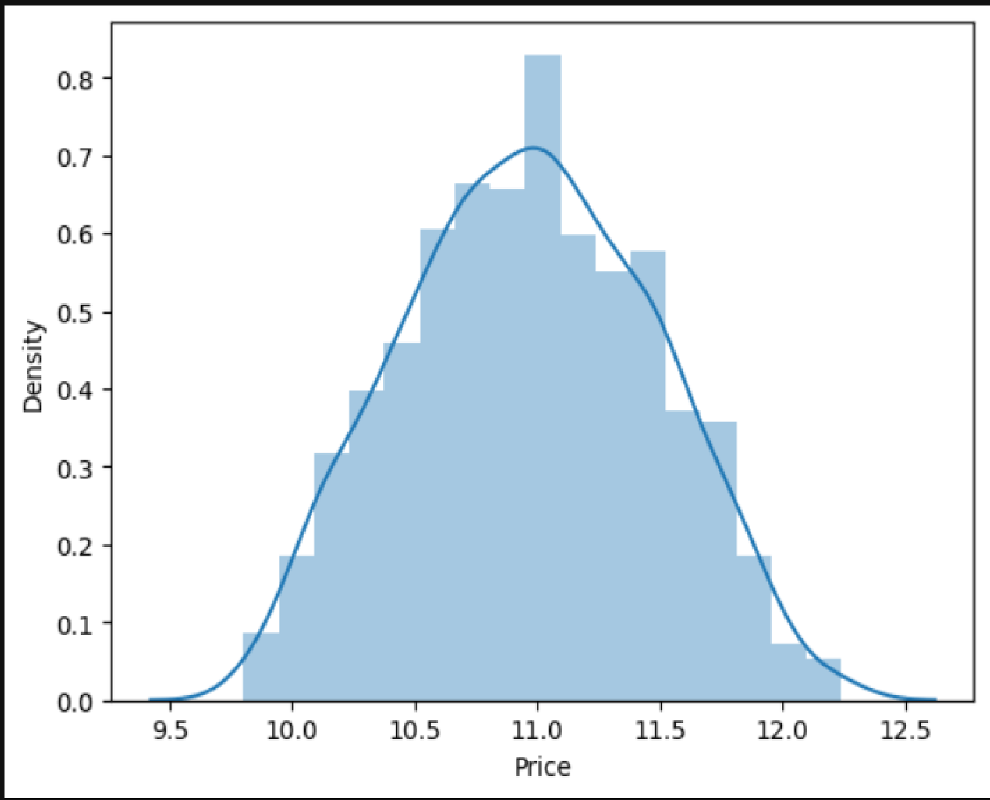
Data Preprocessing



1. Price is distributed with a right skewed distribution.

Hence, we take logarithmic function of price.

```
[133]:  <Axes: xlabel='Price', ylabel='Density'>
```



2. X axis and Y-axis assigning.

```
[135]:  X = laptop.drop(columns= ['Price'])
        y = np.log(laptop.Price)

[136]:  X.info() # Checking the values of X

        <class 'pandas.core.frame.DataFrame'>
        Index: 1050 entries, 0 to 1271
        Data columns (total 13 columns):
         #   Column           Non-Null Count  Dtype
        ---  ------           --------------  -----
         0   Company          1050 non-null   object
         1   TypeName         1050 non-null   object
         2   Touchscreen      1050 non-null   int64
         3   Ram              1050 non-null   int64
         4   Gpu_Comp         1050 non-null   object
         5   Processor_Type   1050 non-null   object
         6   Weight           1050 non-null   float64
         7   MemoryStorage    1050 non-null   float64
         8   SSD              1050 non-null   int64
         9   Hybrid           1050 non-null   int64
         10  Windows          1050 non-null   int64
         11  No OS            1050 non-null   int64
         12  Others           1050 non-null   int64
        dtypes: float64(2), int64(7), object(4)
        memory usage: 114.8+ KB

[137]:  y # Checking the values of y

[137]:  0       11.175755
        1       10.776777
        2       10.329931
        3       11.814476
        4       11.473101
                  ...
        1265    10.667632
        1267    10.555257
```

Applying Regression Models:

Then we do the Hot Encoding and Run all the regression models.

One HOT Encoding and aaplying on different Regression Models

Random Forest

```
[143]: from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
st1 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse_output=False, drop='first'), [0,1,4,5])], remainder='passthrough') #column which h

st2 = RandomForestRegressor(n_estimators=100, max_depth=15,
                                          max_features=0.75,
                                          random_state=3,
                                          max_samples=0.4)
pipe = Pipeline([('1st Step',st1),
                ('2nd Step', st2)
               ])

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

# Printing the R2_score and Mean absolute Error for the Prediction Model
print('R2_score',r2_score(y_test,y_pred))
print('Mean Absolute Error',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.8150172555398887
Mean Absolute Error 0.1592662697292521
```

SVR Model

```
[145]: from sklearn.svm import SVR
t1 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse_output=False, drop='first'), [0,1,4,5])], remainder='passthrough')

t2 = SVR(kernel='rbf', C=10000,epsilon=0.1)
pipe = Pipeline([('Step 1',t1),
                 ('Step 2', t2)
])

pipe.fit(X_train,y_train)
y_pred = pipe.predict(X_test)

print('R2_Score', r2_score(y_test,y_pred))
print('Mean Absolute Error', mean_absolute_error(y_test,y_pred))
```

```
R2_Score 0.7787336937629761
Mean Absolute Error 0.18125445287472597
```

Ridge Regression Model

[147]:
```
from sklearn.linear_model import Ridge
f1 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse_output=False, drop='first'), [0,1,4,5])], remainder='passthrough')

f2 = Ridge(alpha=10)

pipe = Pipeline([('1st Step',f1),
                 ('2nd Step', f2)
                ])
pipe.fit(X_train,y_train)
y_pred = pipe.predict(X_test)

print('R2_Score', r2_score(y_test,y_pred))
print('Mean Absolute Error', mean_absolute_error(y_test,y_pred))
```

```
R2_Score 0.7269222586129696
Mean Absolute Error 0.21015906377510282
```

Linear Regression Model

[149]:
```
from sklearn.linear_model import LinearRegression
h1 =  ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse_output=False, drop='first'), [0,1,4,5])], remainder='passthrough')

h2 = LinearRegression()

pipe = Pipeline([('1st Step',h1),
                  ('2ms Step',h2)
])

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)

print('R2_Score', r2_score(y_test,y_pred))
print('Mean Absolute Error', mean_absolute_error(y_test,y_pred))
```

```
R2_Score 0.7319810481966442
Mean Absolute Error 0.20817154570730814
```

Decision Tree Model

51]:
```
from sklearn.tree import DecisionTreeRegressor
a1 =  ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse_output=False, drop='first'), [0,1,4,5])], remainder='passthrough')

a2 = DecisionTreeRegressor(max_depth=8)

pipe = Pipeline([('1st Step',a1),
        ('2nd Step',a2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2_Score', r2_score(y_test,y_pred))
print('Mean Absolute Error', mean_absolute_error(y_test,y_pred))
```

```
R2_Score 0.7558737742555084
Mean Absolute Error 0.19242470574466108
```

KNN Model

```
[153]: from sklearn.neighbors import KNeighborsRegressor

g1 =  ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse_output=False, drop='first'), [0,1,4,5])], remainder='passthrough')

g2 = KNeighborsRegressor(n_neighbors=5)

pipe = Pipeline([('1st Stage',g1),
                 ('2nd Stage',g2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2_Score', r2_score(y_test,y_pred))
print('Mean Absolute Error', mean_absolute_error(y_test,y_pred))
```

```
R2_Score 0.764409934463457
Mean Absolute Error 0.18584041800543605
```

Since Random Forest gives us the highest R square score therefore, we will select Random Forest Regression as our predicting model.

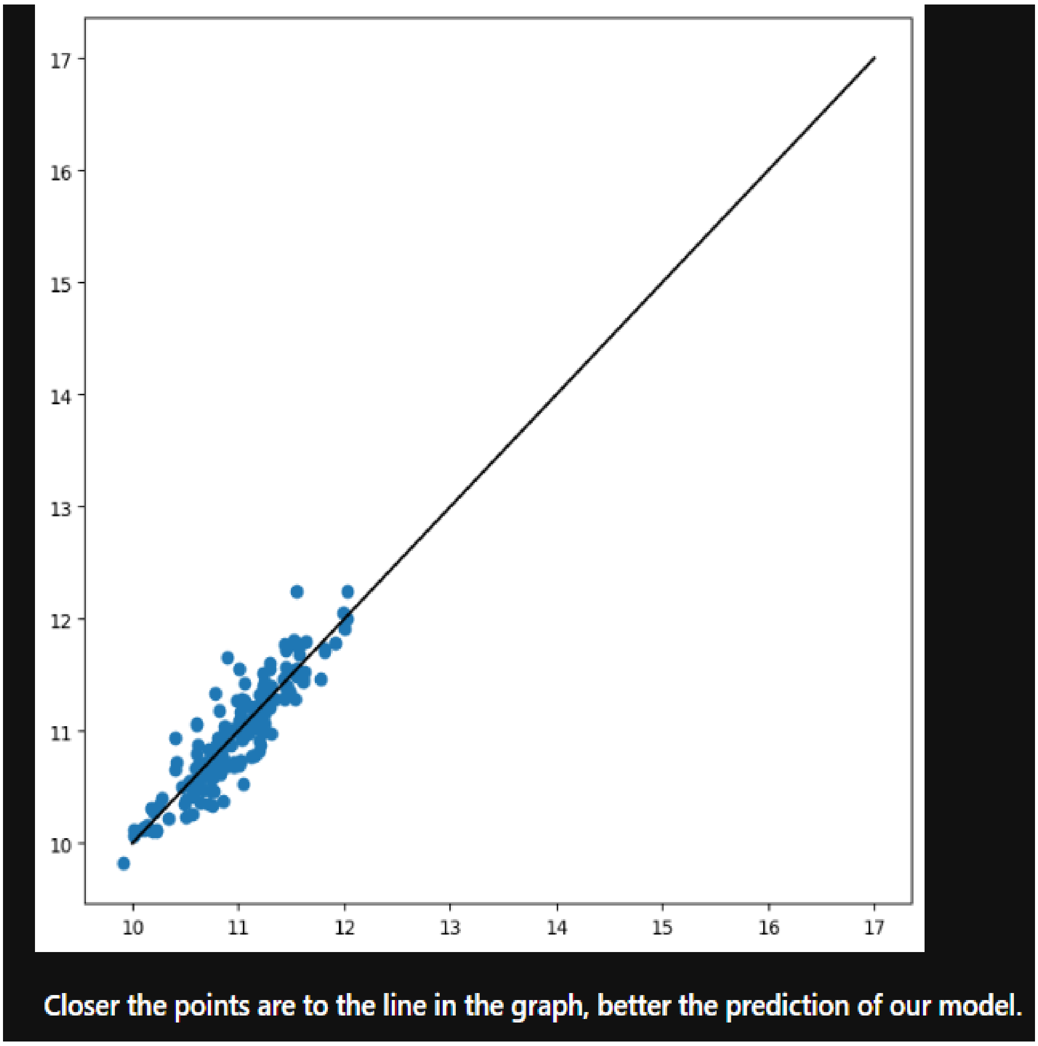
Now we do the hypertuning of the model:

Hypertuning the Model

```
[156]: from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0,1,4,5])], remainder='passthrough')
X = np.array(ct.fit_transform(X))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.15, random_state= 42)


parameters = {
    'n_estimators': [50, 100, 150, 200, 250, 300],
    'max_depth': [15, 20, 25, 30],
    'max_features': [0.1, 0.4, 0.6, 0.7, 0.8, 0.9, 1],
    'max_samples': [0.2, 0.4, 0.6]
}

best_parameter = GridSearchCV(RandomForestRegressor(), parameters, cv=3, n_jobs= -1, verbose=2, error_score='raise')
best_parameter.fit(X_train, y_train)
# Best Score
v_model = best_parameter.best_score_
print('Best Score is:', v_model)
print('Best Parameter is:', best_parameter.best_params_ )
```

```
Fitting 3 folds for each of 504 candidates, totalling 1512 fits
Best Score is: 0.8102759564465245
Best Parameter is: {'max_depth': 30, 'max_features': 0.1, 'max_samples': 0.6, 'n_estimators': 50}
```

The prediction of the model is as per the expectations as the observed and predicted va;ues are clustering around the line of best fit here.



We create the function required for Predicting the Price:

```
[173]: def predictor_price(Company,
                         TypeName,
                         Touchscreen,
                         Ram,
                         Gpu_Comp,
                         Processor_Type,
                         Weight,
                         MemoryStorage,
                         SSD,
                         Hybrid,
                         Windows,
                         No_OS):

           arr = np.array([Company, TypeName, Touchscreen, Ram, Gpu_Comp, Processor_Type, Weight, MemoryStorage, SSD, Hybrid, Windows, No_OS])
           k = arr.reshape(1,12)

           prediction = np.exp(pipe.predict(k)[0])


           return round(prediction)
```

Prediction based on the input given by the user:

```
[174]: predictor_price(Company ='Apple',
                         TypeName='Ultrabook',
                         Touchscreen=0,
                         Ram=8,
                         Gpu_Comp='Intel',
                         Processor_Type= 'i5',
                         Weight= 1.37,
                         MemoryStorage= 128,
                         SSD = 512,
                         Hybrid=0,
                         Windows=0,
                         No_OS=0)

       C:\Users\nsany\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but OneHotEncoder was fitted with featu
       re names
         warnings.warn(

[174]: 66397
```

```
[197]: predictor_price(Company ='Samsung',
                       TypeName='Ultrabook',
                       Touchscreen=0,
                       Ram=16,
                       Gpu_Comp='Intel',
                       Processor_Type= 'i7',
                       Weight= 1.71,
                       MemoryStorage= 256,
                       SSD = 1,
                       Hybrid=0,
                       Windows=1,
                       No_OS=0)
```

```
C:\Users\nsany\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but OneHotEncoder was fitted with feat
re names
  warnings.warn(
```

```
[197]: 92589
```