LAPTOP PRICE PREDICTOR

G-10

Group Members

(Group no : 10)

Yash Shinde

Sakshi Khandagale

Ram Yamgar

Snehal Patil

Ayush Jaiswal

Content

1. Describe Problem Statement
2. Overview about dataset
3. Data Cleaning
4. Exploratory Data Analysis
5. Feature Engineering
6. Machine Learning Modeling
7. ML web app development
8. Deployment Machine learning app

P R O B L E M S T A T E M E N T

Problem Statement for Laptop Price Prediction

- We will make a project for Laptop price prediction. The problem statement is that if any user wants to buy a laptop then our application should be compatible to provide a tentative price of laptop according to the user configurations.

- Although it looks like a simple project or just developing a model, the dataset we have is noisy and needs lots of feature engineering, and preprocessing that will drive your interest in developing this project.

Data Set for Laptop Prediction

Most of the columns in a dataset are noisy and contain lots of information. But with feature engineering you do, you will get more good results. The only problem is we are having less data but we will obtain a good accuracy over it. The only good thing is it is better to have a large data, we will develop a website that could predict a tentative price of a laptop based on user configuration.

DATA SET

jupyter laptop_price_predictor Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
1298	1298	Lenovo	2 in 1 Convertible	14.0	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i7 6500U 2.5GHz	4GB	128GB SSD	Intel HD Graphics 520	Windows 10	1.8kg	33992.6400
1299	1299	Lenovo	2 in 1 Convertible	13.3	IPS Panel Quad HD+ / Touchscreen 3200x1800	Intel Core i7 6500U 2.5GHz	16GB	512GB SSD	Intel HD Graphics 520	Windows 10	1.3kg	79866.7200
1300	1300	Lenovo	Notebook	14.0	1366x768	Intel Celeron Dual Core N3050 1.6GHz	2GB	64GB Flash Storage	Intel HD Graphics	Windows 10	1.5kg	12201.1200
1301	1301	HP	Notebook	15.6	1366x768	Intel Core i7 6500U 2.5GHz	6GB	1TB HDD	AMD Radeon R5 M330	Windows 10	2.19kg	40705.9200
1302	1302	Asus	Notebook	15.6	1366x768	Intel Celeron Dual Core N3050 1.6GHz	4GB	500GB HDD	Intel HD Graphics	Windows 10	2.2kg	19660.3200

1303 rows × 12 columns

Basic Understanding of Laptop Price Prediction Data

It is good that there are no NULL values. And we need little changes in weight and Ram column to convert them to numeric by removing the unit written after value. So we will perform data cleaning here to get the correct types of columns.

In [13]:
```
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
```

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
0
```

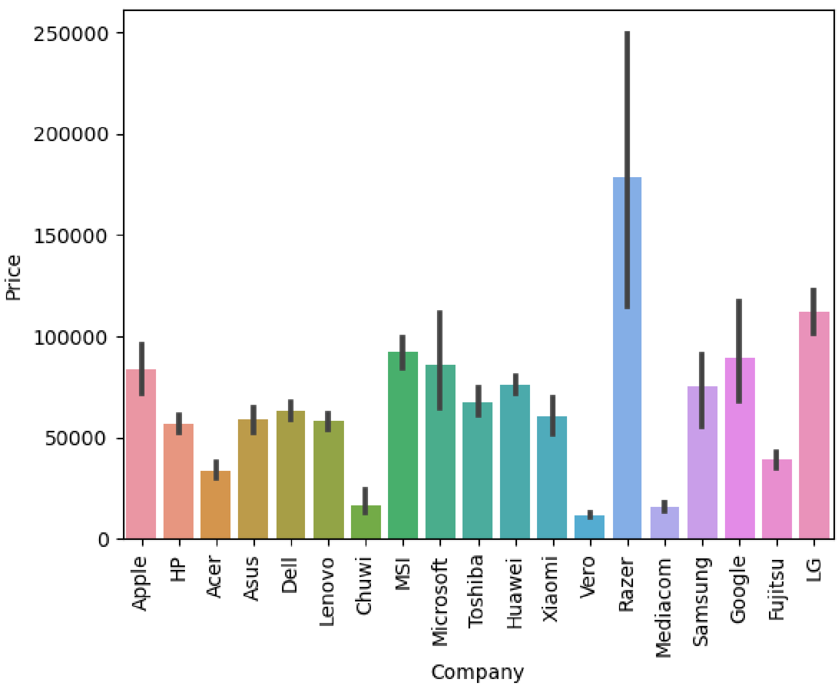
In [7]:
```
df.isnull().sum()
```

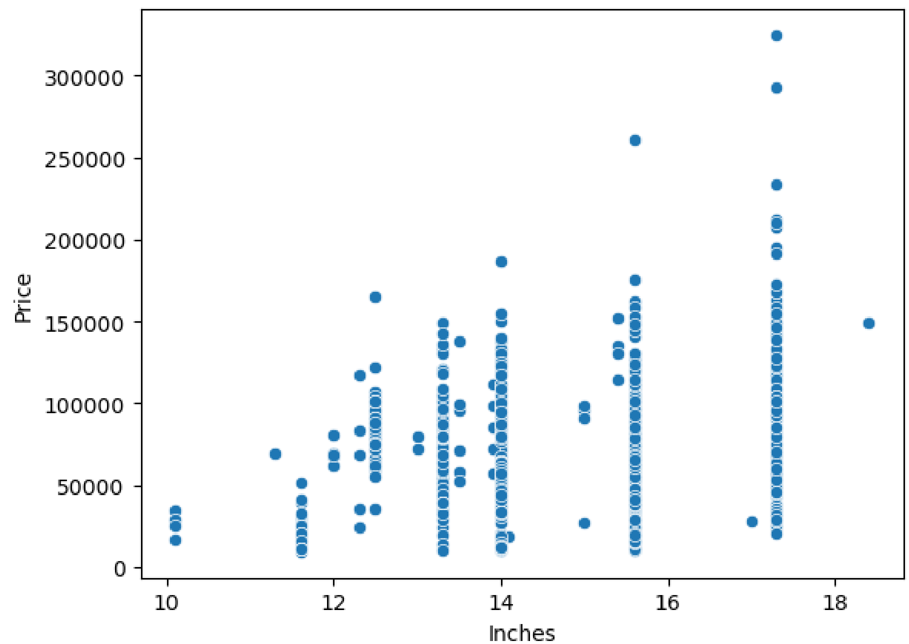
Out[7]:
```
Unnamed: 0          0
Company             0
TypeName            0
Inches              0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
Price               0
dtype: int64
```

EDA of Laptop Price Prediction Dataset

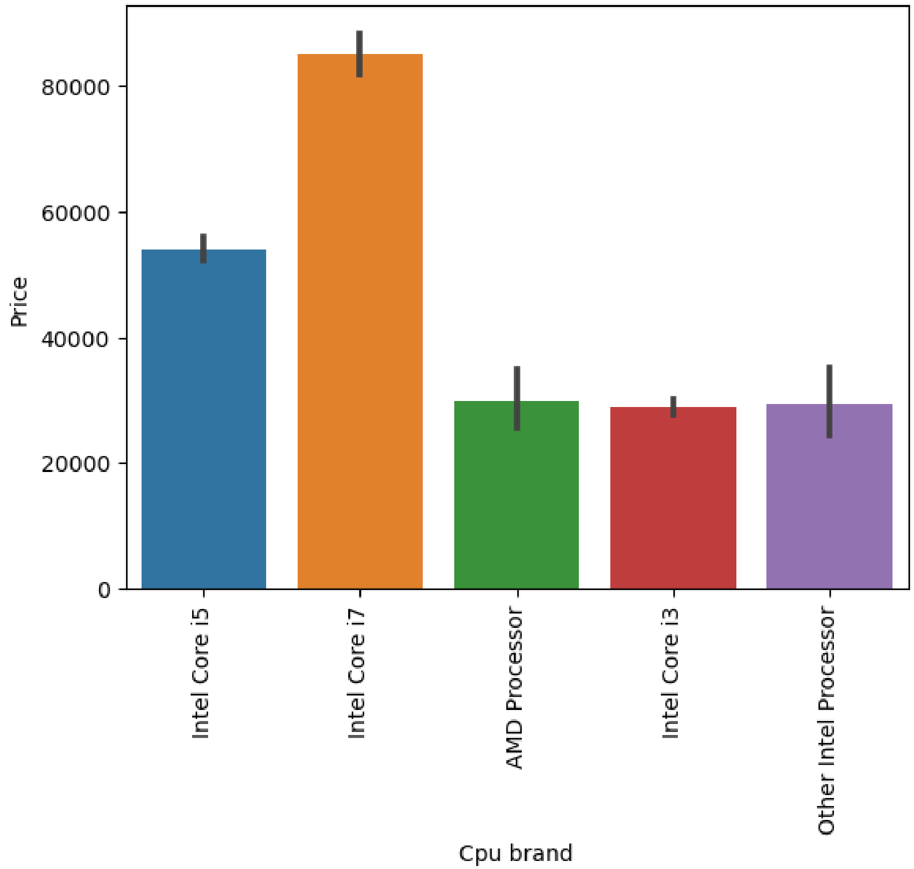
EDA helps to perform hypothesis testing. We will start from the first column and explore each column and understand what impact it creates on the target column. At the required step, we will also perform preprocessing and feature engineering tasks, our aim in performing in-depth EDA is to prepare and clean data for better machine learning handeling to achieve high performance and generalized models, so let's get started with analyzing and preparing the dataset for prediction

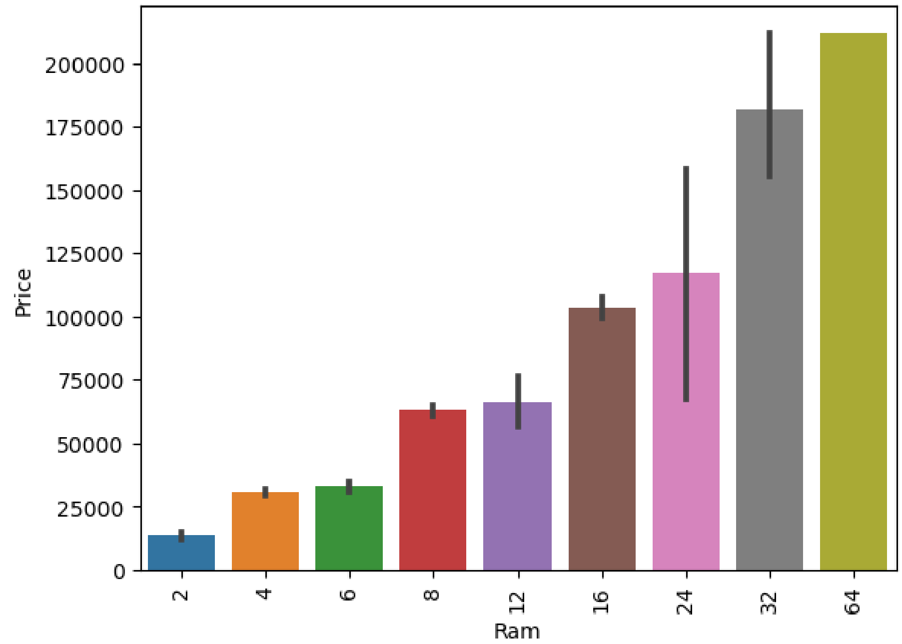
Category wise Laptop price

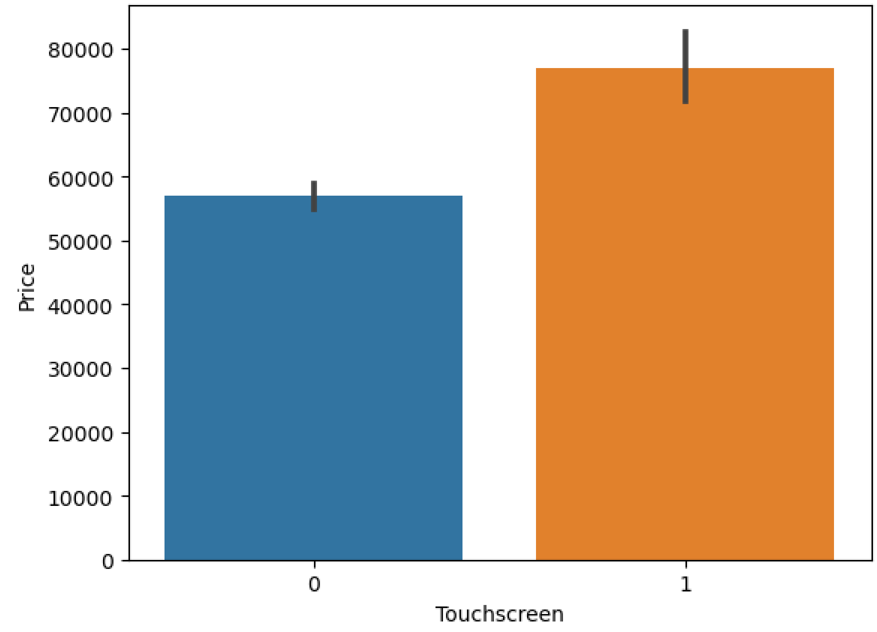




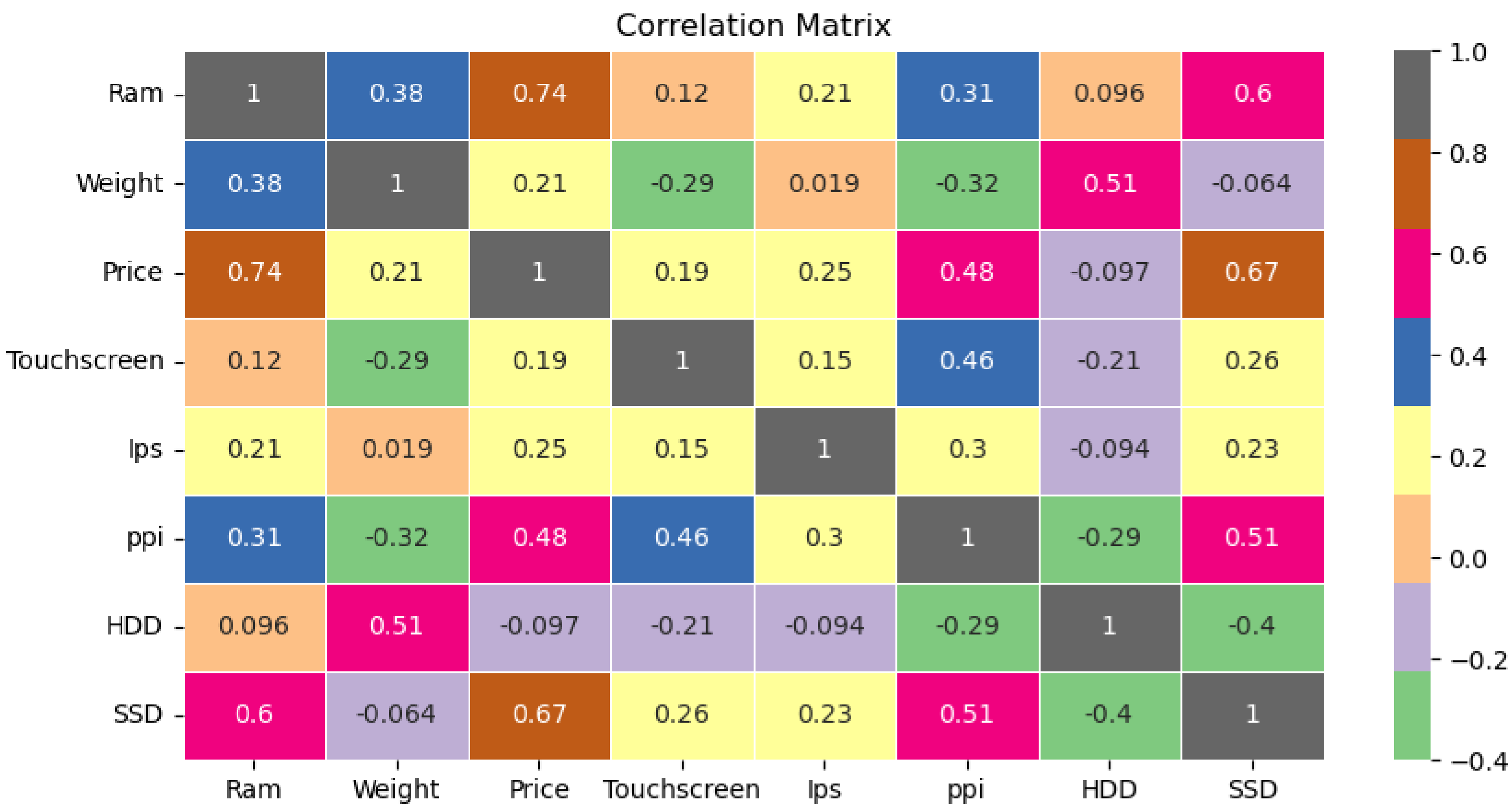
Category wise Laptop price





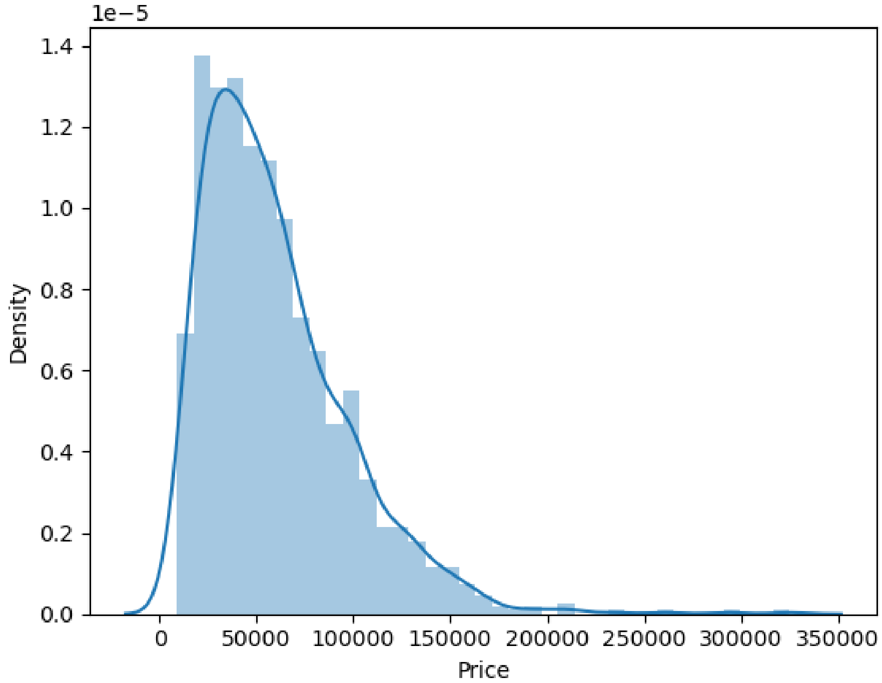


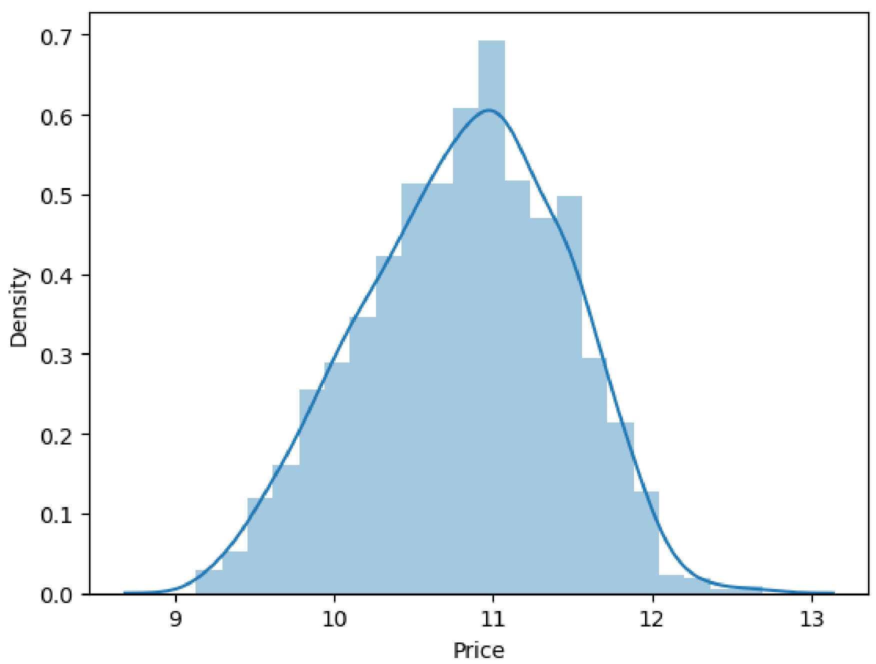
A correlation matrix is a statistical technique used to evaluate the relationship between two variables in a data set. The matrix is a table in which every cell contains a correlation coefficient, where 1 is considered a strong relationship between variables, 0 a neutral relationship and -1 a not strong relationship.



Distribution Plot for Target Price

The distribution of the target variables is left skewed and it is obvious that configuration with low price are sold and purchased more than branded ones





Train test split

We have imported libraries to split data , and algorithms you can try. At a time we do not know which is the best so you can try all imported algorithms.
As discussed we have taken the log of the dependent variables. And the training data looks something below the data frame.

In [89]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.15,random_state=2)
```

In [112]:
```
X_train.head()
```

Out[112]:

	Company	TypeName	Ram	Weight	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Gpu brand	os
183	Toshiba	Notebook	8	2.00	0	0	100.454670	Intel Core i5	0	128	Intel	Windows
1141	MSI	Gaming	8	2.40	0	0	141.211998	Intel Core i7	1000	128	Nvidia	Windows
1049	Asus	Netbook	4	1.20	0	0	135.094211	Other Intel Processor	0	0	Intel	Others/No OS/Linux
1020	Dell	2 in 1 Convertible	4	2.08	1	1	141.211998	Intel Core i3	1000	0	Intel	Windows
878	Dell	Notebook	4	2.18	0	0	141.211998	Intel Core i5	1000	128	Nvidia	Windows

In [113]:
```
y_train.head(5)
```

Out[113]:
```
183     10.651384
1141    11.016798
1049     9.638174
1020    10.655148
878     10.791749
Name: Price, dtype: float64
```

Model Fitting and Pipeline

There are two main differences between the gradient boosting trees and the random forests. We train the former sequentially, one tree at a time, each to correct the errors of the previous ones. In contrast, we construct the trees in a random forest independently.

Random Forest

In [99]:
```
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))
```

```
R2 score 0.8873402378382488
MAE 0.15860130110457718
```

Gradient Boost

In [101]:
```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = GradientBoostingRegressor(n_estimators=500)

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))
```

```
R2 score 0.8811634881251004
MAE 0.1601818221226992
```

A voting regressor is an ensemble meta-estimator that fits several base regressors, each on the whole dataset. Then it averages the individual predictions to form a final prediction.

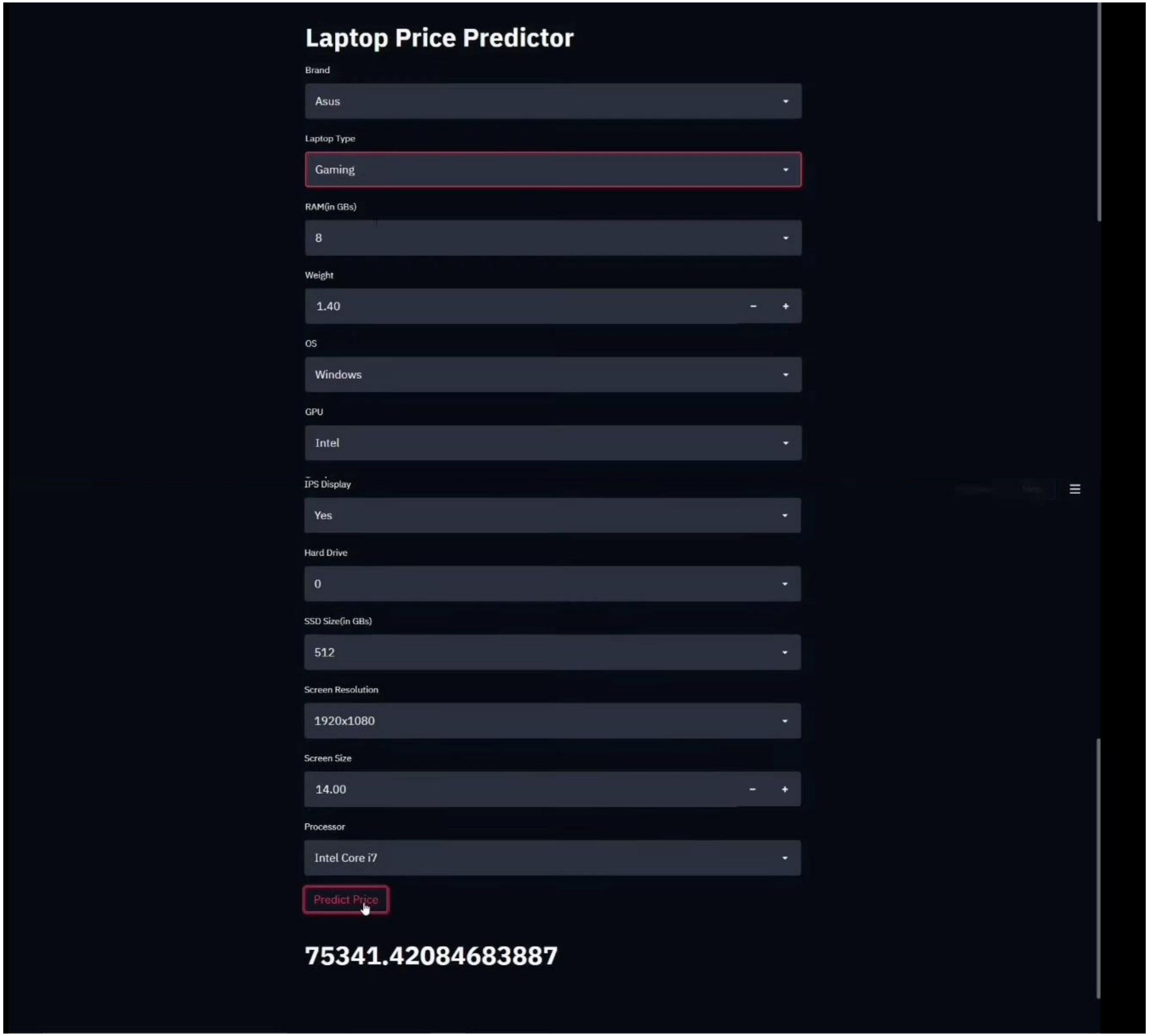
Voting Regressor

In [102]:

```
from sklearn.ensemble import VotingRegressor,StackingRegressor

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


rf = RandomForestRegressor(n_estimators=350,random_state=3,max_samples=0.5,max_features=0.75,max_depth=15)
gbdt = GradientBoostingRegressor(n_estimators=100,max_features=0.5)

step2 = VotingRegressor([('rf', rf), ('gbdt', gbdt)],weights=[6,4])

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))
```

```
R2 score 0.8867264619890043
MAE 0.15981882677581108
```

WEB PAGE



Laptop Price Predictor
Brand
Asus
Laptop Type
Gaming
RAM(in GBs)
8
Weight
1.40
OS
Windows
GPU
Intel
IPS Display
Yes
Hard Drive
0
SSD Size(in GBs)
512
Screen Resolution
1920x1080
Screen Size
14.00
Processor
Intel Core i7
Predict Price
75341.42084683887

Exporting the model