Sumedha Reddy Mannem (G29423995)

Machine Learning CSCI_6364_80

Final Project Report

Title : Data-drive Laptop Price Prediction

Abstract: This project employs various machine learning techniques to predict laptop prices, aiming to assist both buyers and sellers in the market. Using historical price data, the study develops and evaluates multiple regression models to forecast prices based on laptop features. The project's goal is to enhance market transparency and enable stakeholders to make informed decisions.

***Introduction*:** Step into the world of intelligent laptop shopping, "Data-Driven Laptop Price Prediction: An AI Approach for Smart Shoppers." With an arsenal of machine learning techniques and a passion for data, this project sets out on a mission to revolutionize how buyers and sellers navigate the laptop market.

The project focuses on predicting laptop prices through meticulous data cleaning and preprocessing, ensuring the dataset's quality. The project explores various regression models, including linear regression, decision tree regression, random forest regression, and more, to unravel the intricate relationship between laptop features and prices.

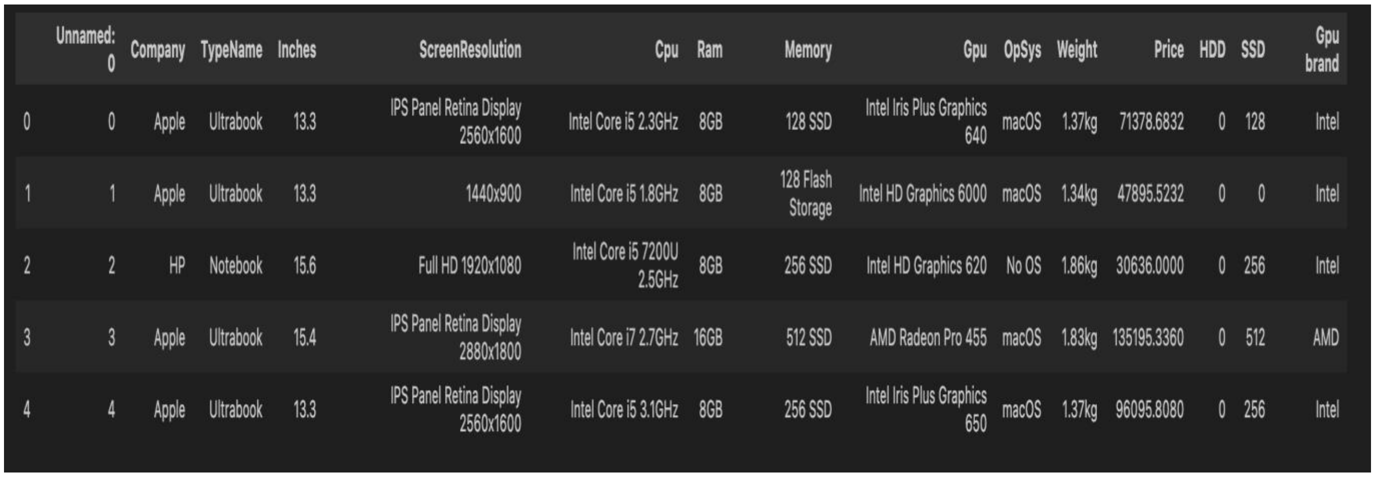
Historical laptop price data becomes the cornerstone for training these advanced models, enabling them to capture market dynamics accurately. The ultimate goal? Providing invaluable insights to both buyers and sellers, empowering them to make informed decisions.

Accurate price predictions are the key for buyers seeking competitive offers and sellers aiming for optimal pricing strategies. This project's sophisticated regression models, coupled with precise data cleaning, enhance market transparency and equip stakeholders with critical information for strategic decision-making.

Through the utilization of multiple regression models, including Linear, Ridge, Gradient Boosting, Decision Tree, Random Forest, SVM, XGB, and stacking regressor, aims to unlock the potential of AI-driven predictions, transforming the landscape of laptop shopping.

Data Description: The dataset utilized in this study comprises various laptop specifications and their corresponding prices, sourced from Kaggle. It includes features such as 'Company', 'TypeName', 'Inches', 'ScreenResolution', 'Cpu', 'Ram', 'Memory', 'Gpu', 'OpSys', 'Weight', 'Price', 'HDD', 'SSD', and 'Gpu brand' which are instrumental in predicting the laptop prices.

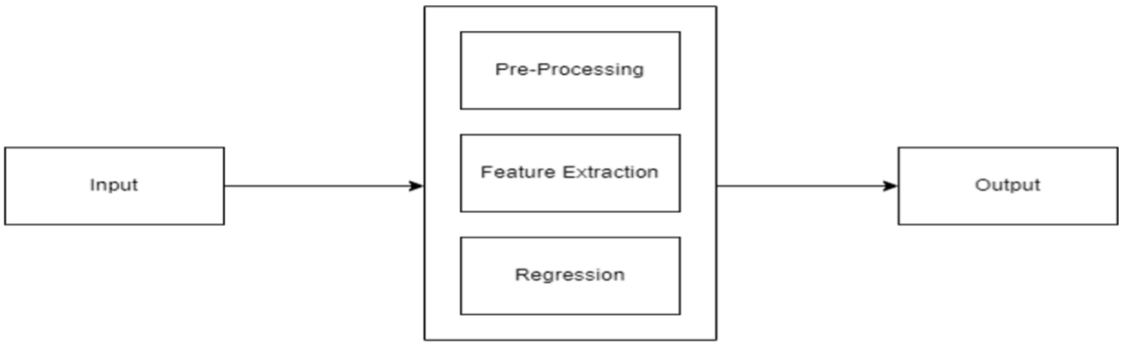
Laptop Dataset:



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

Methodology:

Figure 1: **Workflow of Laptop Price Prediction model**



This diagram depicts the workflow of the laptop price prediction model used in this project. The process begins with the Input stage, where the laptop data is collected. Following the initial data input, the workflow consists of three primary stages:

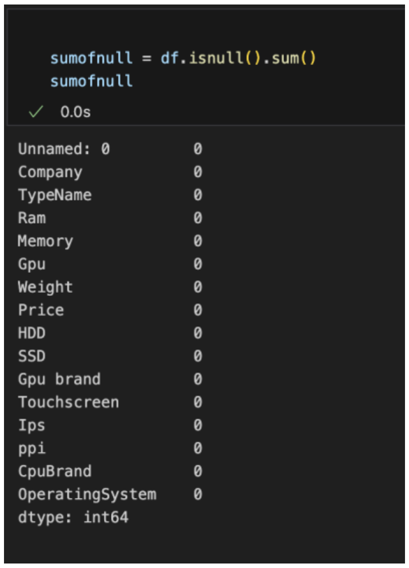
Pre-Processing: In this initial stage, the raw data undergoes cleaning and normalization to ensure quality and consistency. Missing values are handled, and anomalies are corrected, preparing the data for feature extraction.
Feature Extraction: The pre-processed data is then used to extract features that are relevant to the prediction of laptop prices. This stage involves selecting significant attributes that influence laptop prices and may include deriving new features to enhance the predictive accuracy.
Regression: With the features prepared, the data is subjected to regression analysis. Various regression techniques, such as Linear, Ridge, Gradient Boosting, Decision Tree, Random Forest, SVM, XGB, and stacking regressor, are applied to model and predict laptop prices based on the features.

The final stage produces an Output which is the predicted price of the laptops. This output helps in making informed decisions regarding laptop pricing strategies for sellers and purchase choices for buyers. The subsequent details outline the procedures undertaken in each of these stages.

Stage 1 - Data Preprocessing

1. **Identifying Null Values in Laptop Dataset**

This task involves determining the count of null values present in the laptop dataset. By using the 'isnull().sum()' method, aim to understand the extent of missing information within the dataset. Identifying null values is essential for data preprocessing, enabling us to handle missing data appropriately. This step contributes to ensuring the dataset's quality and completeness, which is fundamental for accurate model building in our analysis. There are no null values in the dataset.



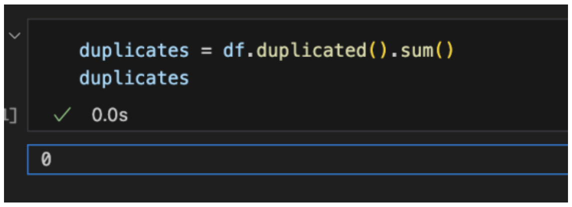
```
sumofnull = df.isnull().sum()
sumofnull
```

```
✓ 0.0s
```

```
Unnamed: 0         0
Company            0
TypeName           0
Ram                0
Memory             0
Gpu                0
Weight             0
Price              0
HDD                0
SSD                0
Gpu brand          0
Touchscreen        0
Ips                0
ppi                0
CpuBrand           0
OperatingSystem    0
dtype: int64
```

2. Identifying Duplicate Entries

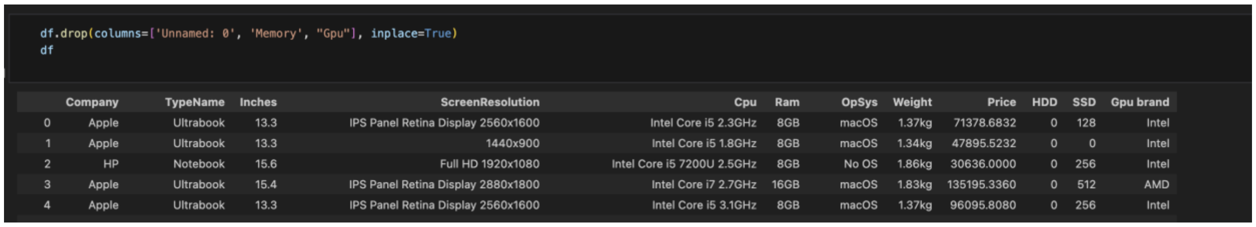
In this task, aim to identify and quantify duplicate entries within the laptop dataset. The count of duplicates is crucial for ensuring data cleanliness and accuracy in subsequent analyses. By detecting and handling duplicate records, we maintain the dataset's integrity, providing reliable information for our price prediction model.



```
duplicates = df.duplicated().sum()
duplicates
✓ 0.0s
0
```

3. Dropping unnecessary columns

In this task, certain columns ('Unnamed: 0', 'Memory', 'Gpu') are removed from the laptop dataset. The 'Unnamed: 0' column is eliminated as it seems unnecessary. Additionally, 'Memory' and 'Gpu' information might be redundant, present in other columns. This step streamlines the dataset for better analysis and model development, enhancing the relevance of features for the price prediction.

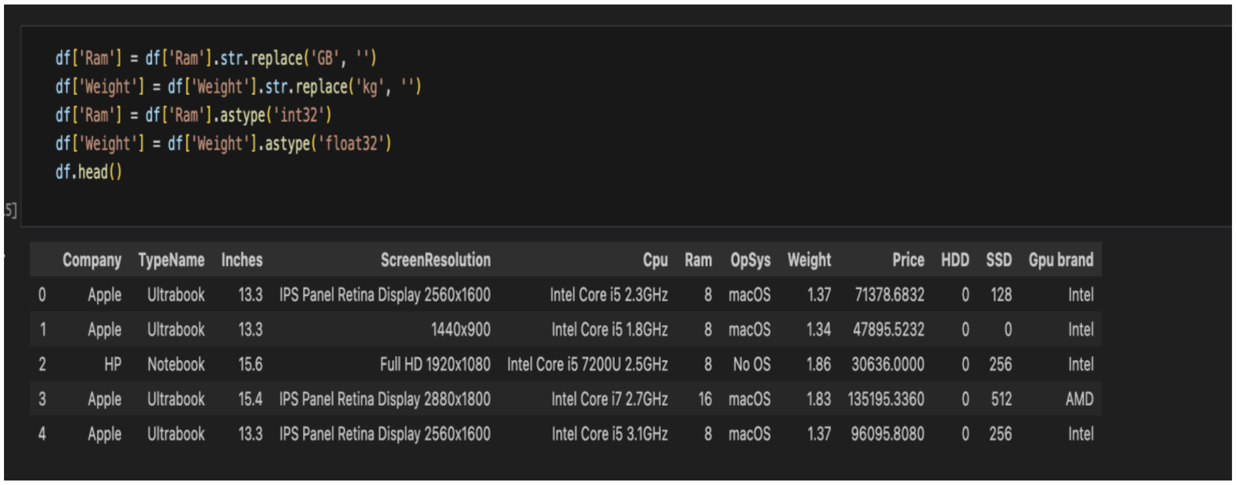


```
df.drop(columns=['Unnamed: 0', 'Memory', "Gpu"], inplace=True)
df
```

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

4. Data transformation – RAM and Weight

This task involves data transformation in the 'Ram' and 'Weight' columns of the laptop dataset. The 'GB' and 'kg' units are removed from the 'Ram' and 'Weight' columns, respectively, and the columns are converted to appropriate data types ('Ram' to integer and 'Weight' to float). This transformation facilitates consistent and numerical data handling for subsequent analysis and machine learning model development.



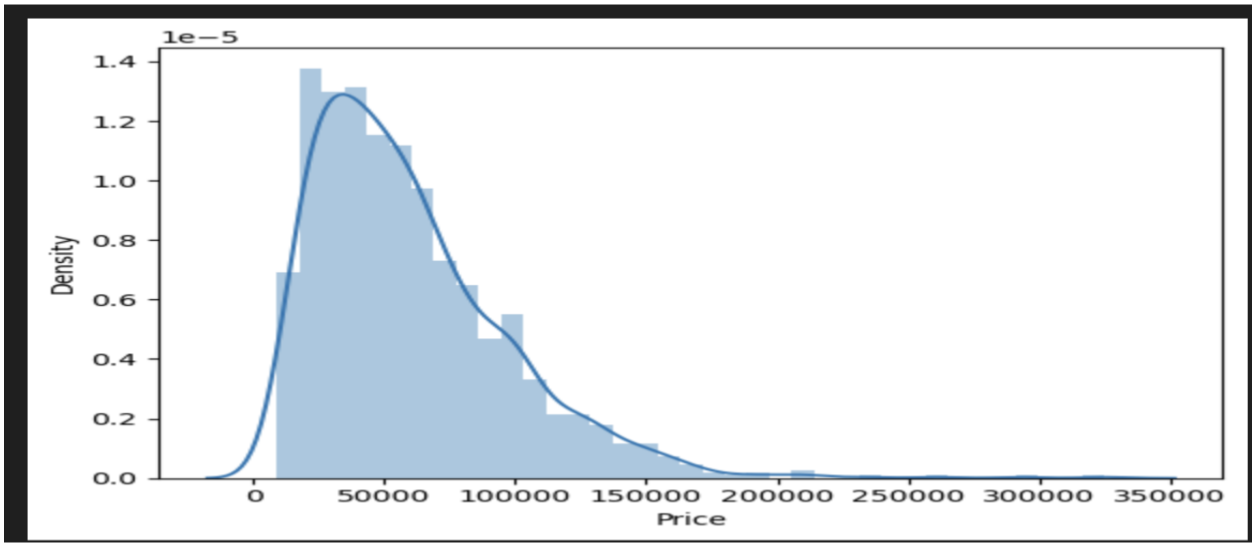
```python
df['Ram'] = df['Ram'].str.replace('GB', '')
df['Weight'] = df['Weight'].str.replace('kg', '')
df['Ram'] = df['Ram'].astype('int32')
df['Weight'] = df['Weight'].astype('float32')
df.head()
```

```
  Company TypeName Inches                ScreenResolution                  Cpu Ram OpSys Weight       Price HDD SSD Gpu brand
0   Apple Ultrabook   13.3 IPS Panel Retina Display 2560x1600   Intel Core i5 2.3GHz   8 macOS   1.37  71378.6832   0 128     Intel
1   Apple Ultrabook   13.3                          1440x900   Intel Core i5 1.8GHz   8 macOS   1.34  47895.5232   0   0     Intel
2      HP  Notebook   15.6                Full HD 1920x1080 Intel Core i5 7200U 2.5GHz   8 No OS   1.86  30636.0000   0 256     Intel
3   Apple Ultrabook   15.4 IPS Panel Retina Display 2880x1800   Intel Core i7 2.7GHz  16 macOS   1.83 135195.3360   0 512       AMD
4   Apple Ultrabook   13.3 IPS Panel Retina Display 2560x1600   Intel Core i5 3.1GHz   8 macOS   1.37  96095.8080   0 256     Intel
```

Exploratory Data Analysis:

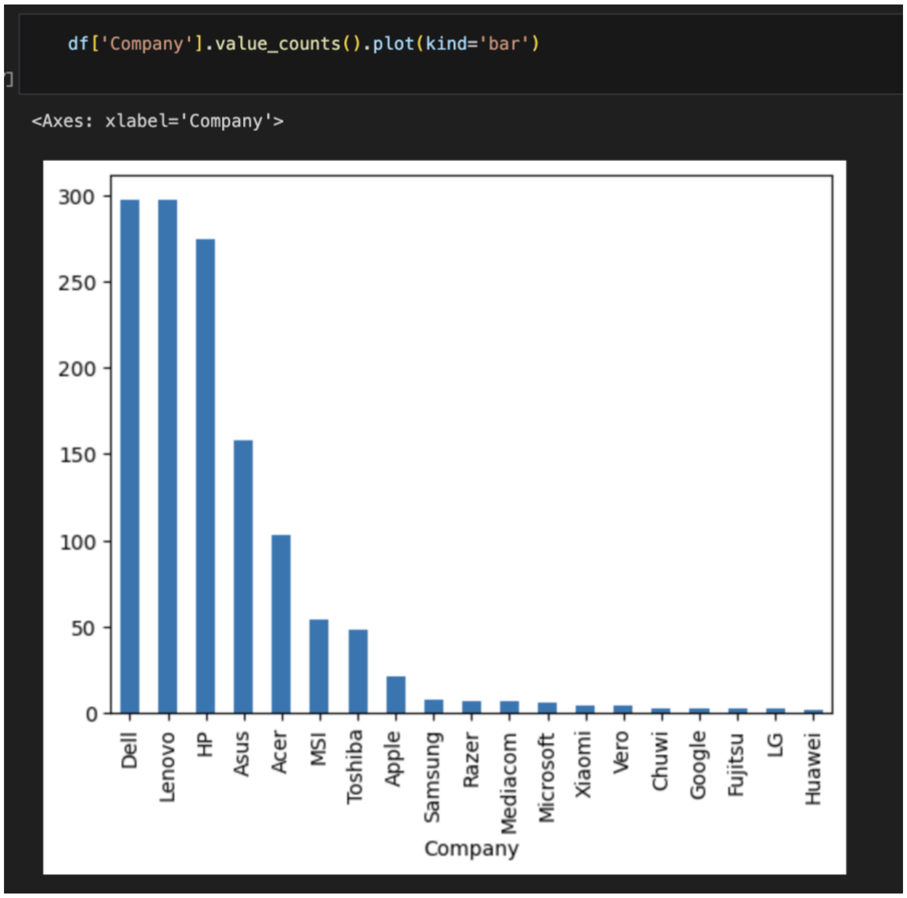
Understanding data is the process of exploratory analysis. It aids in the identification of features and patterns that can be utilized by machine learning algorithms. We can make better decisions and eliminate a lot of guesswork by recognizing trends and commonalities in your data.

Figure 2: Distribution of Laptop Price



The distribution of the target variable is skewed and it is obvious that low-price commodities are sold rather than branded ones.

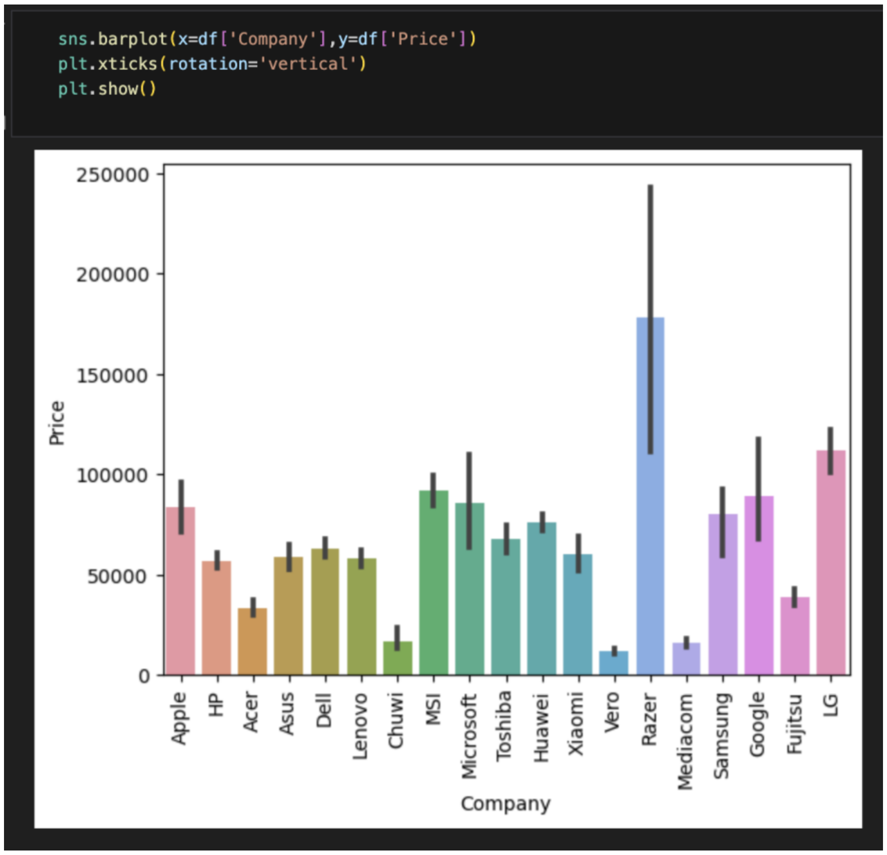
Figure 3: Distribution of company name



```
df['Company'].value_counts().plot(kind='bar')
```

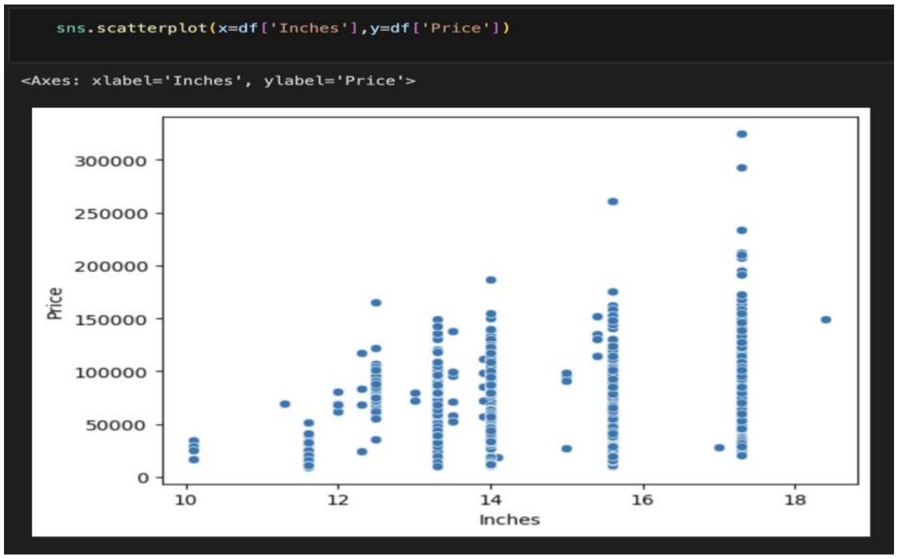
```
<Axes: xlabel='Company'>
```

Figure 4: Laptop Price Distribution Across Companies



```
sns.barplot(x=df['Company'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()
```

Figure 5: Laptop Price Distribution Across inches

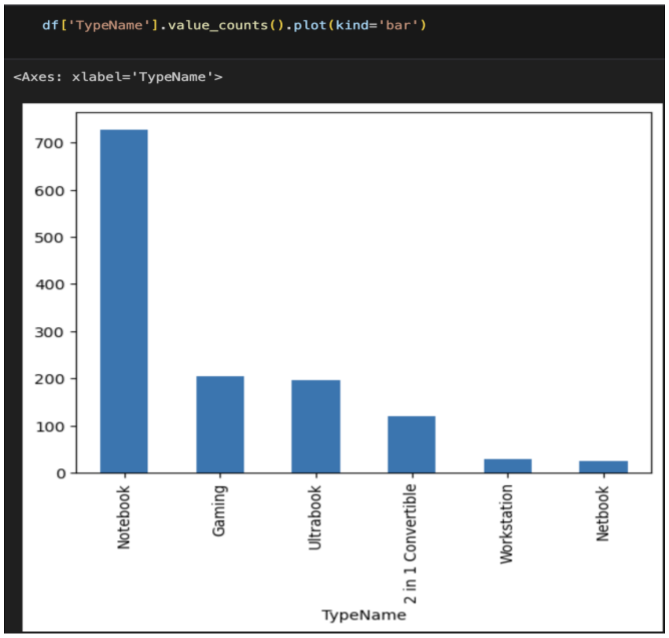


```
sns.scatterplot(x=df['Inches'],y=df['Price'])
```

```
<Axes: xlabel='Inches', ylabel='Price'>
```

Type of Laptop: We can check which type of laptop is available like a gaming laptop, workstation, or notebook. Major people prefer laptops because they are under budget and the same can be concluded from our data.

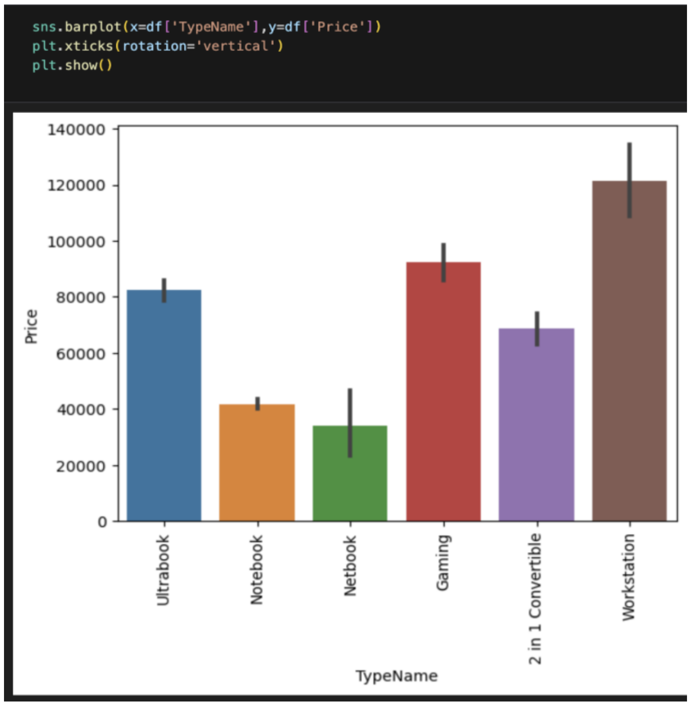
Figure 6: Distribution of Typename



```
df['TypeName'].value_counts().plot(kind='bar')
```

```
<Axes: xlabel='TypeName'>
```

Figure 7: Laptop Price Distribution Across varied types

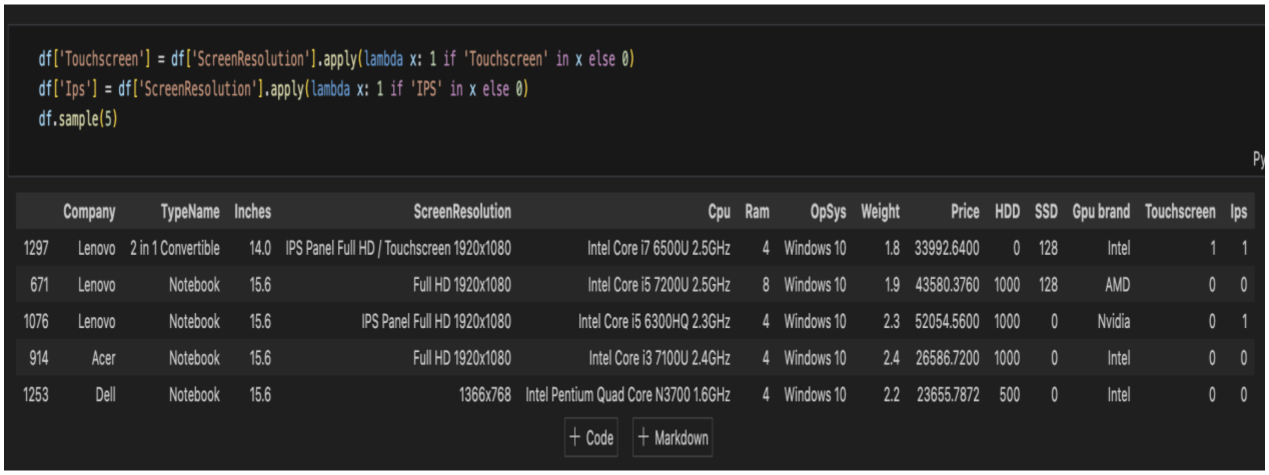


```
sns.barplot(x=df['TypeName'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()
```

Stage 2: Feature Engineering

1. Touchscreen and IPS

This task involves feature engineering by creating binary indicators ('Touchscreen' and 'Ips') based on the 'ScreenResolution' column. The 'Touchscreen' column flags laptops with touchscreen displays, assigning 1 for 'Touchscreen' presence and 0 otherwise. Similarly, the 'Ips' column identifies laptops with IPS screens, assigning 1 for 'IPS' in the screen resolution and 0 otherwise. This feature engineering step aids in capturing specific screen characteristics for further analysis without detailing the code in the task description.

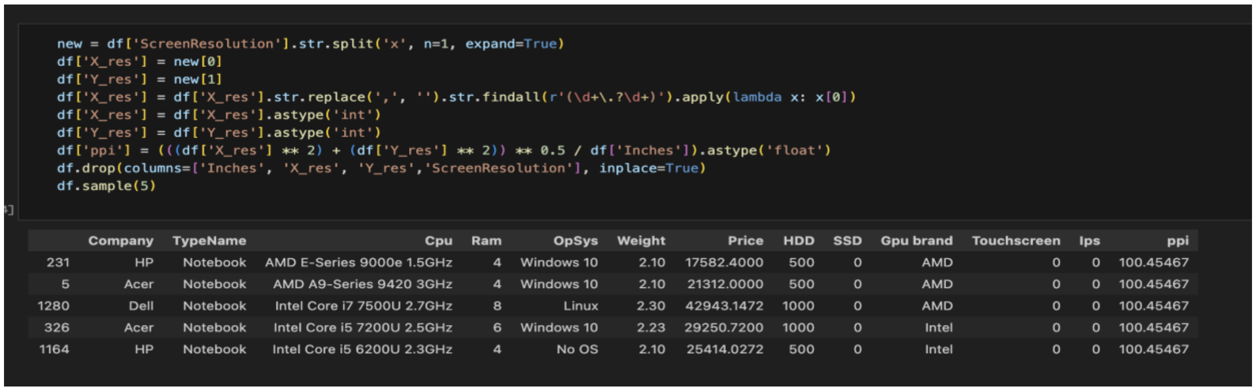


```
df['Touchscreen'] = df['ScreenResolution'].apply(lambda x: 1 if 'Touchscreen' in x else 0)
df['Ips'] = df['ScreenResolution'].apply(lambda x: 1 if 'IPS' in x else 0)
df.sample(5)
```

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	OpSys	Weight	Price	HDD	SSD	Gpu brand	Touchscreen	Ips
1297	Lenovo	2 in 1 Convertible	14.0	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i7 6500U 2.5GHz	4	Windows 10	1.8	33992.6400	0	128	Intel	1	1
671	Lenovo	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	Windows 10	1.9	43580.3760	1000	128	AMD	0	0
1076	Lenovo	Notebook	15.6	IPS Panel Full HD 1920x1080	Intel Core i5 6300HQ 2.3GHz	4	Windows 10	2.3	52054.5600	1000	0	Nvidia	0	1
914	Acer	Notebook	15.6	Full HD 1920x1080	Intel Core i3 7100U 2.4GHz	4	Windows 10	2.4	26586.7200	1000	0	Intel	0	0
1253	Dell	Notebook	15.6	1366x768	Intel Pentium Quad Core N3700 1.6GHz	4	Windows 10	2.2	23655.7872	500	0	Intel	0	0

2. Screen Resolution Analysis

The screen resolution information is extracted and processed to derive the pixels per inch (PPI) for each laptop in the dataset. The 'ScreenResolution' column is split into 'X_res' and 'Y_res' for width and height, respectively. Calculations are performed to determine PPI using the derived width, height, and screen size ('Inches') information. After computation, unnecessary columns ('Inches', 'X_res', 'Y_res', 'ScreenResolution') are dropped. This screen resolution analysis enhances the dataset with a meaningful feature ('ppi') for future analysis and model development.

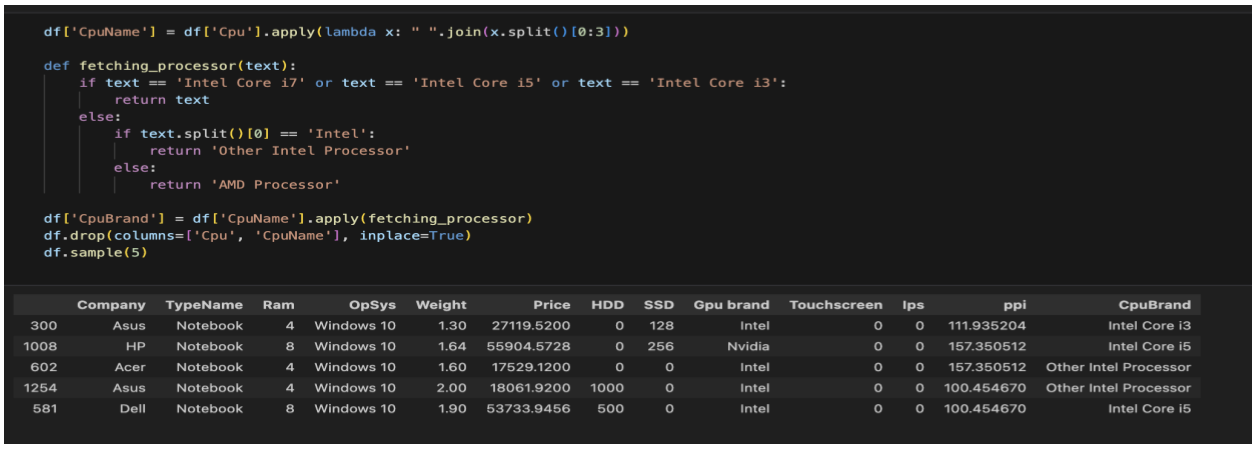


```python
new = df['ScreenResolution'].str.split('x', n=1, expand=True)
df['X_res'] = new[0]
df['Y_res'] = new[1]
df['X_res'] = df['X_res'].str.replace(',', '').str.findall(r'(\d+\.?\d+)').apply(lambda x: x[0])
df['X_res'] = df['X_res'].astype('int')
df['Y_res'] = df['Y_res'].astype('int')
df['ppi'] = (((df['X_res'] ** 2) + (df['Y_res'] ** 2)) ** 0.5 / df['Inches']).astype('float')
df.drop(columns=['Inches', 'X_res', 'Y_res','ScreenResolution'], inplace=True)
df.sample(5)
```

```
      Company  TypeName                        Cpu  Ram      OpSys  Weight       Price   HDD  SSD  Gpu brand  Touchscreen  Ips        ppi
 231       HP  Notebook  AMD E-Series 9000e 1.5GHz    4  Windows 10    2.10  17582.4000   500    0        AMD            0    0  100.45467
   5     Acer  Notebook    AMD A9-Series 9420 3GHz    4  Windows 10    2.10  21312.0000   500    0        AMD            0    0  100.45467
1280     Dell  Notebook   Intel Core i7 7500U 2.7GHz  8       Linux    2.30  42943.1472  1000    0        AMD            0    0  100.45467
 326     Acer  Notebook   Intel Core i5 7200U 2.5GHz  6  Windows 10    2.23  29250.7200  1000    0      Intel            0    0  100.45467
1164       HP  Notebook   Intel Core i5 6200U 2.3GHz  4       No OS    2.10  25414.0272   500    0      Intel            0    0  100.45467
```

3. Processor Analysis and Categorization

The dataset is enriched by extracting the processor's brand and type from the 'Cpu' column. Initially, the 'Cpu' column is processed to obtain the 'Cpu Name' with the first three elements. Then, a function categorizes processor brands into 'Intel Core i7', 'Intel Core i5', 'Intel Core i3', 'Other Intel Processor', and 'AMD Processor'. After assigning the processor brand to the 'Cpu brand' column, the original 'Cpu' and 'Cpu Name' columns are dropped. This analysis helps categorize and simplify processor information for further analysis and model development without explicitly including code details.

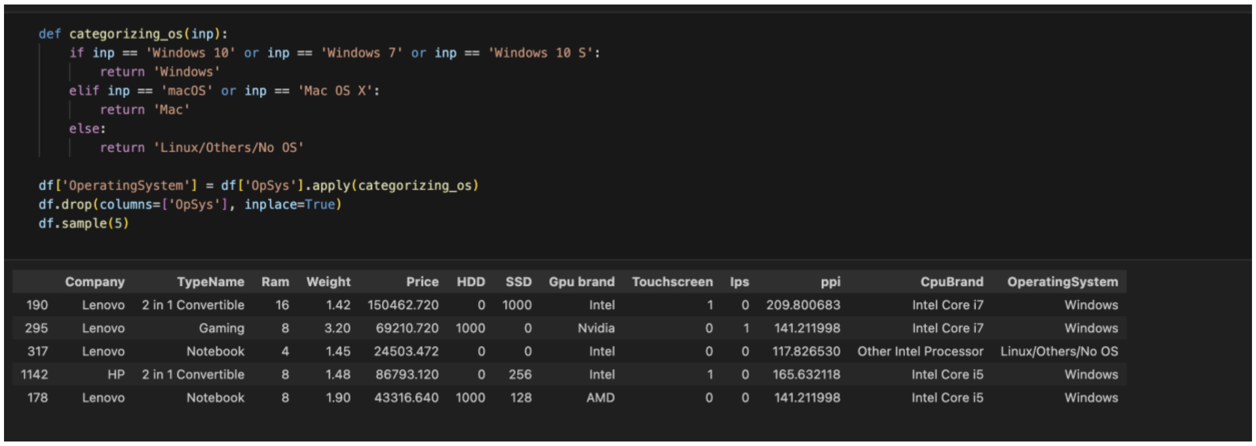


```python
df['CpuName'] = df['Cpu'].apply(lambda x: " ".join(x.split()[0:3]))

def fetching_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['CpuBrand'] = df['CpuName'].apply(fetching_processor)
df.drop(columns=['Cpu', 'CpuName'], inplace=True)
df.sample(5)
```

```
      Company  TypeName  Ram       OpSys  Weight       Price   HDD  SSD  Gpu brand  Touchscreen  Ips         ppi               CpuBrand
 300     Asus  Notebook    4  Windows 10    1.30  27119.5200     0  128      Intel            0    0  111.935204          Intel Core i3
1008       HP  Notebook    8  Windows 10    1.64  55904.5728     0  256     Nvidia            0    0  157.350512          Intel Core i5
 602     Acer  Notebook    4  Windows 10    1.60  17529.1200     0    0      Intel            0    0  157.350512  Other Intel Processor
1254     Asus  Notebook    4  Windows 10    2.00  18061.9200  1000    0      Intel            0    0  100.454670  Other Intel Processor
 581     Dell  Notebook    8  Windows 10    1.90  53733.9456   500    0      Intel            0    0  100.454670          Intel Core i5
```

4. Operating System Categorization

In this task, the laptop dataset is enriched by categorizing the operating system information into broader groups. A function 'cat_os' categorizes different operating systems into 'Windows', 'Mac', and 'Others/No OS/Linux' categories based on their names. The 'OpSys' column is transformed into the 'os' column to represent these categories. Following this categorization, the original 'OpSys' column is dropped. This categorization aids in simplifying the operating system information for further analysis and model development.



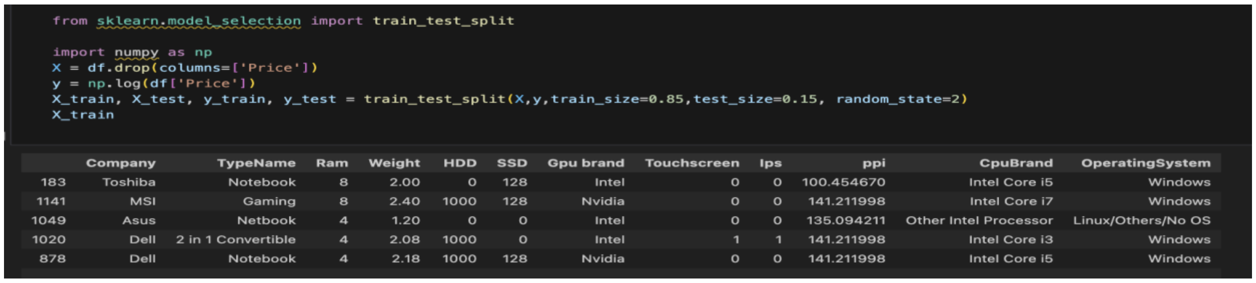
```
def categorizing_os(inp):
    if inp == 'Windows 10' or inp == 'Windows 7' or inp == 'Windows 10 S':
        return 'Windows'
    elif inp == 'macOS' or inp == 'Mac OS X':
        return 'Mac'
    else:
        return 'Linux/Others/No OS'

df['OperatingSystem'] = df['OpSys'].apply(categorizing_os)
df.drop(columns=['OpSys'], inplace=True)
df.sample(5)
```

```
      Company        TypeName  Ram  Weight       Price   HDD   SSD  Gpu brand  Touchscreen  Ips         ppi               CpuBrand     OperatingSystem
190    Lenovo  2 in 1 Convertible   16    1.42  150462.720     0  1000      Intel            1    0  209.800683          Intel Core i7             Windows
295    Lenovo          Gaming    8    3.20   69210.720  1000     0     Nvidia            0    1  141.211998          Intel Core i7             Windows
317    Lenovo        Notebook    4    1.45   24503.472     0     0      Intel            0    0  117.826530  Other Intel Processor  Linux/Others/No OS
1142       HP  2 in 1 Convertible    8    1.48   86793.120     0   256      Intel            1    0  165.632118          Intel Core i5             Windows
178    Lenovo        Notebook    8    1.90   43316.640  1000   128        AMD            0    0  141.211998          Intel Core i5             Windows
```

Stage 3: Model Building and Evaluation

Data splitting for Model training

The dataset is prepared for machine learning model training and testing. The 'Price' column is separated from the predictors, denoted as 'X' for features and 'y' for the logarithm of the price. Using 'train_test_split' from the 'sklearn.model_selection' module, the dataset is divided into training and testing sets ('X_train', 'X_test', 'y_train', 'y_test') for model evaluation and prediction purposes without detailing the code in the task description.

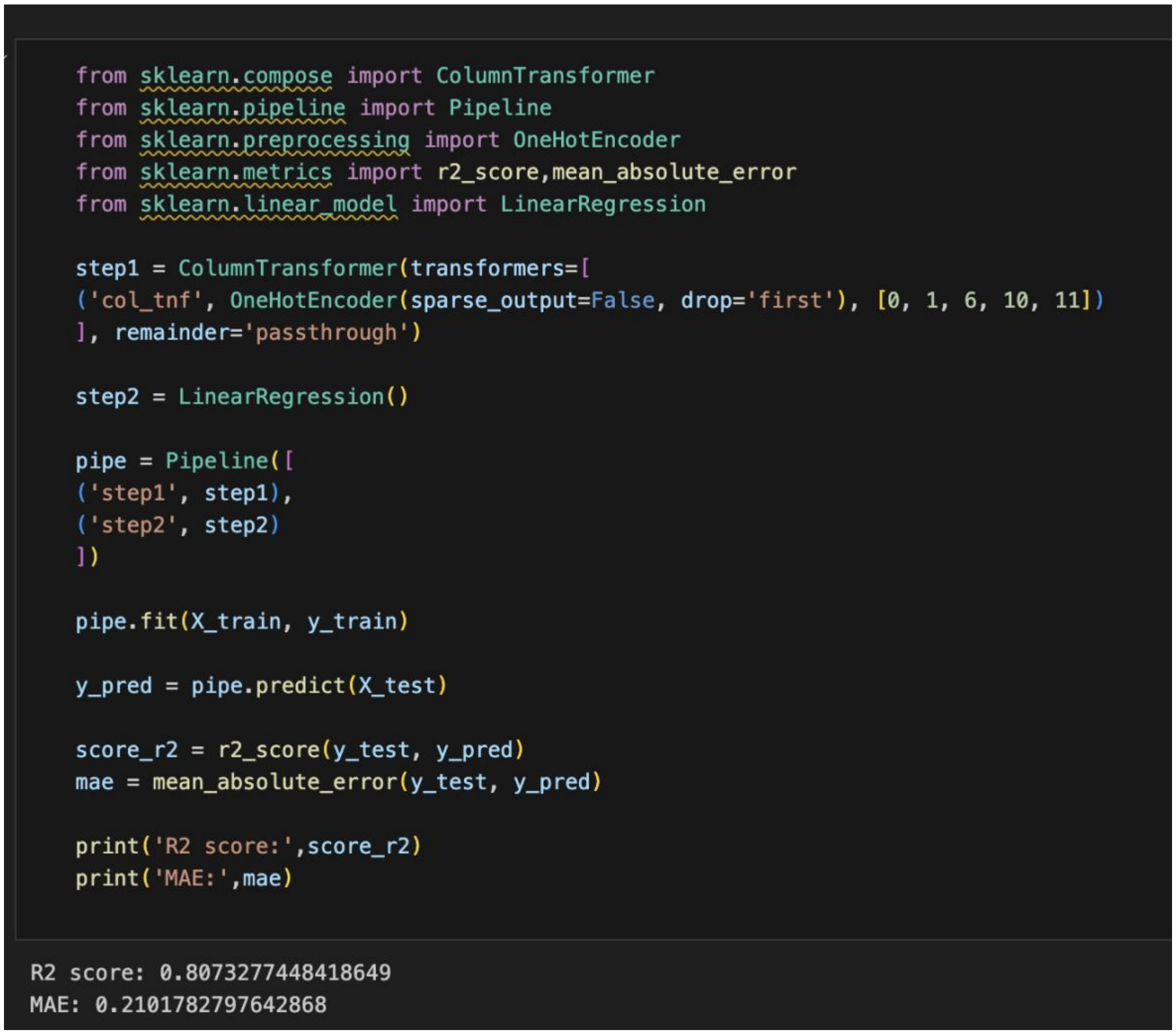


```
from sklearn.model_selection import train_test_split

import numpy as np
X = df.drop(columns=['Price'])
y = np.log(df['Price'])
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.85,test_size=0.15, random_state=2)
X_train
```

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

1. Linear Regression

A machine learning pipeline using Linear Regression model is created and trained using the training dataset. The pipeline consists of two steps:
ColumnTransformer: Encodes categorical variables using OneHotEncoder while preserving sparsity and dropping the first column to avoid multicollinearity issues.
LinearRegression: Implements the Linear Regression model for prediction. The pipeline is fitted on the training data ('X_train', 'y_train') and then used to predict the target variable on the test set ('X_test'). Two evaluation metrics, R-squared (r2_score) and Mean Absolute Error (MAE), are calculated to assess the model's performance.
Linear Regression is a basic yet effective model for regression tasks, fitting a linear relationship between input features and the target variable. R-squared (R2) measures the proportion of variance in the dependent variable that is predictable from the independent variables. Mean Absolute Error (MAE) represents the average absolute difference between predicted and actual values, providing insights into the model's prediction accuracy.



```python
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.linear_model import LinearRegression

step1 = ColumnTransformer(transformers=[
('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 10, 11])
], remainder='passthrough')

step2 = LinearRegression()

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

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

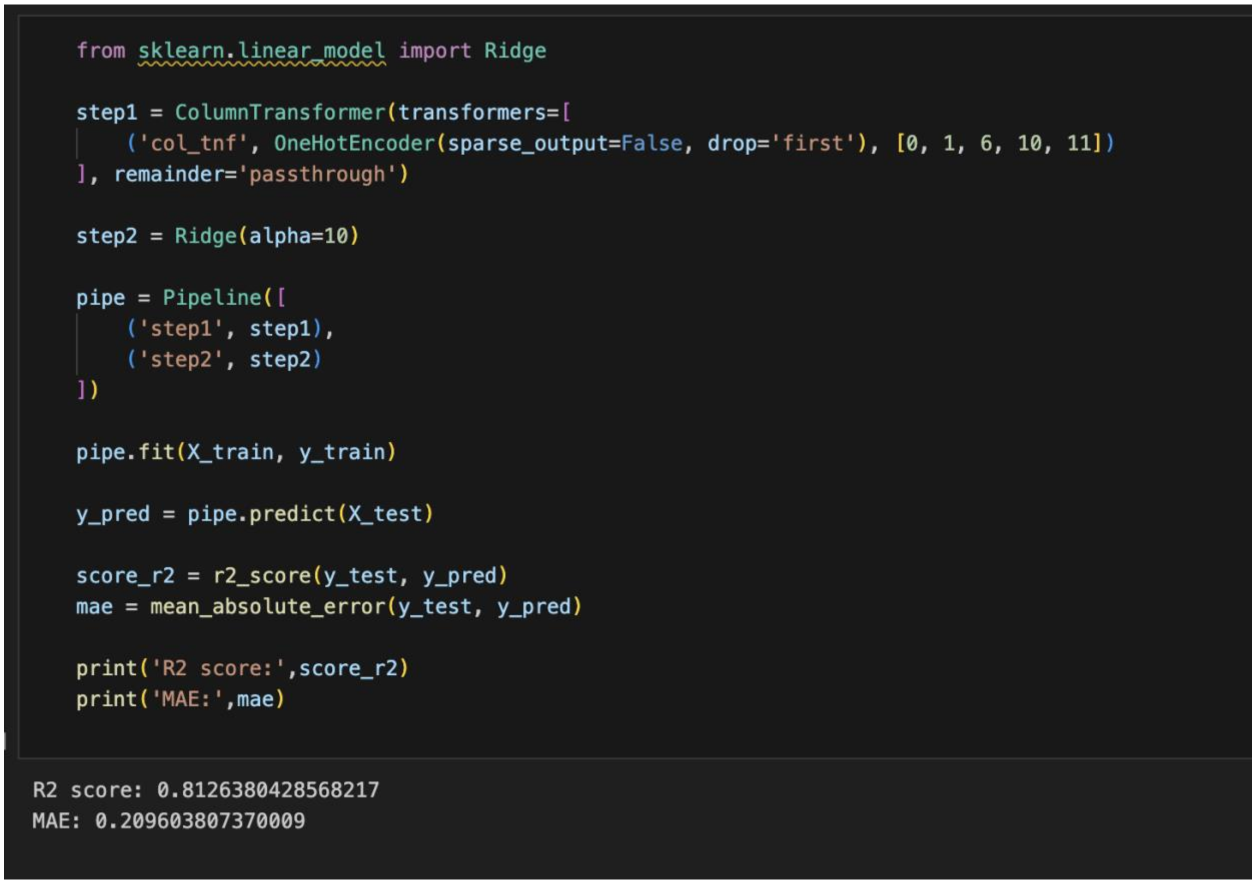
```
R2 score: 0.8073277448418649
MAE: 0.2101782797642868
```

2. Ridge Regression

It involves training and evaluating a Ridge Regression model using a pipeline structure similar to the Linear Regression model. The ColumnTransformer encodes categorical variables using OneHotEncoder, and the Ridge Regression model is instantiated with an alpha value of 10, controlling the regularization strength.

The pipeline is fitted on the training data ('X_train', 'y_train') and predicts the target variable on the test set ('X_test'). Subsequently, R-squared (r2_score) and Mean Absolute Error (MAE) metrics are calculated to assess the model's performance.

Ridge Regression is a linear regression model that uses L2 regularization to prevent overfitting by penalizing large coefficients. The alpha parameter controls the regularization strength, with higher values indicating stronger regularization. The evaluation metrics (R-squared and MAE) provide insights into the model's performance in predicting the logarithm of laptop prices.



```
from sklearn.linear_model import Ridge

step1 = ColumnTransformer(transformers=[
    ('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 10, 11])
], remainder='passthrough')

step2 = Ridge(alpha=10)

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

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

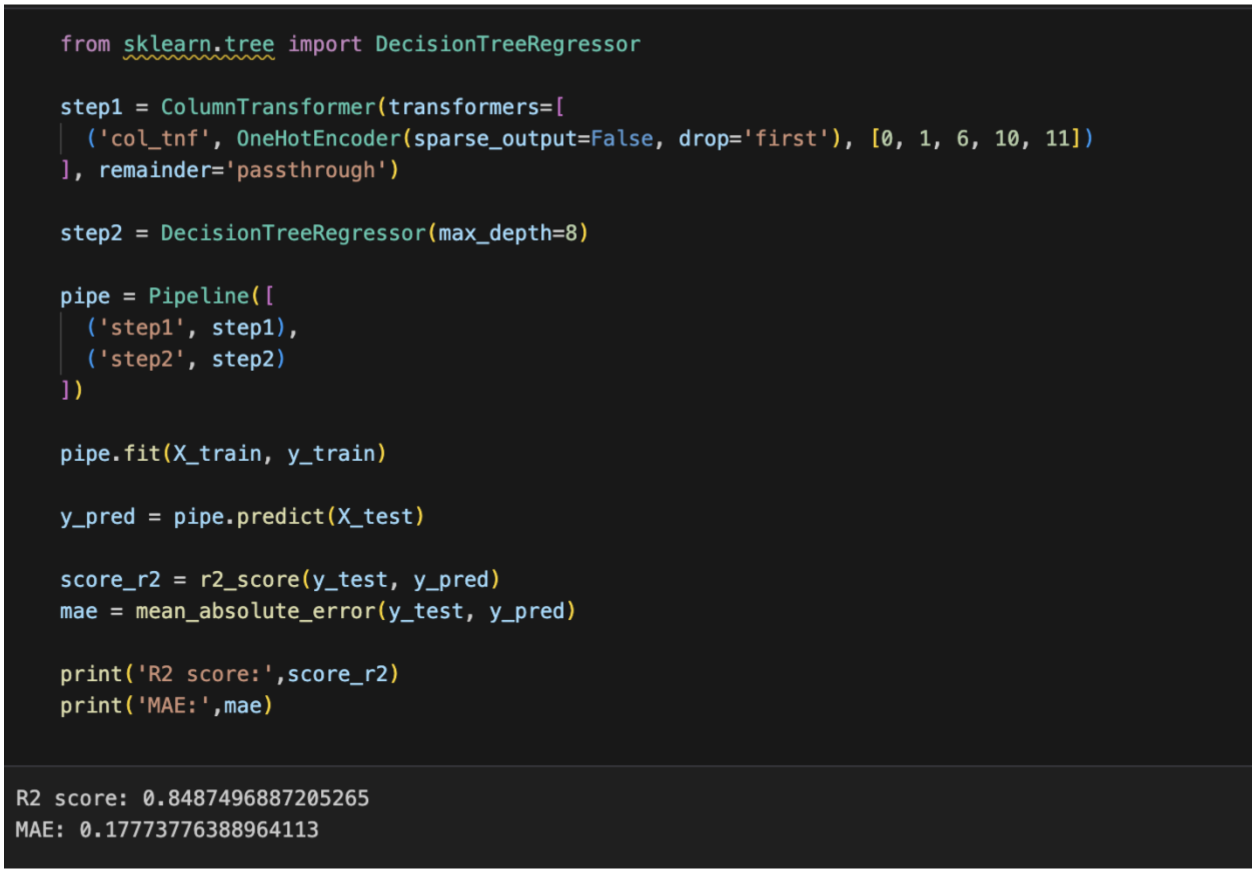
```
R2 score: 0.8126380428568217
MAE: 0.209603807370009
```

3. Decision Tree Regression

Decision Tree Regression model is trained and evaluated using a pipeline structure similar to previous models. The ColumnTransformer encodes categorical variables using OneHotEncoder, and the DecisionTreeRegressor is instantiated with a max_depth of 8, limiting the depth of the decision tree.

The pipeline is fitted on the training data ('X_train', 'y_train') and predicts the target variable on the test set ('X_test'). Subsequently, R-squared (r2_score) and Mean Absolute Error (MAE) metrics are calculated to evaluate the model's performance.

Decision Tree Regression constructs a tree structure to make predictions by splitting the dataset based on features. The max_depth parameter controls the maximum depth of the tree, preventing overfitting by limiting the number of splits. The evaluation metrics (R-squared and MAE) provide insights into how well the model predicts the logarithm of laptop prices.



```python
from sklearn.tree import DecisionTreeRegressor

step1 = ColumnTransformer(transformers=[
    ('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 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)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

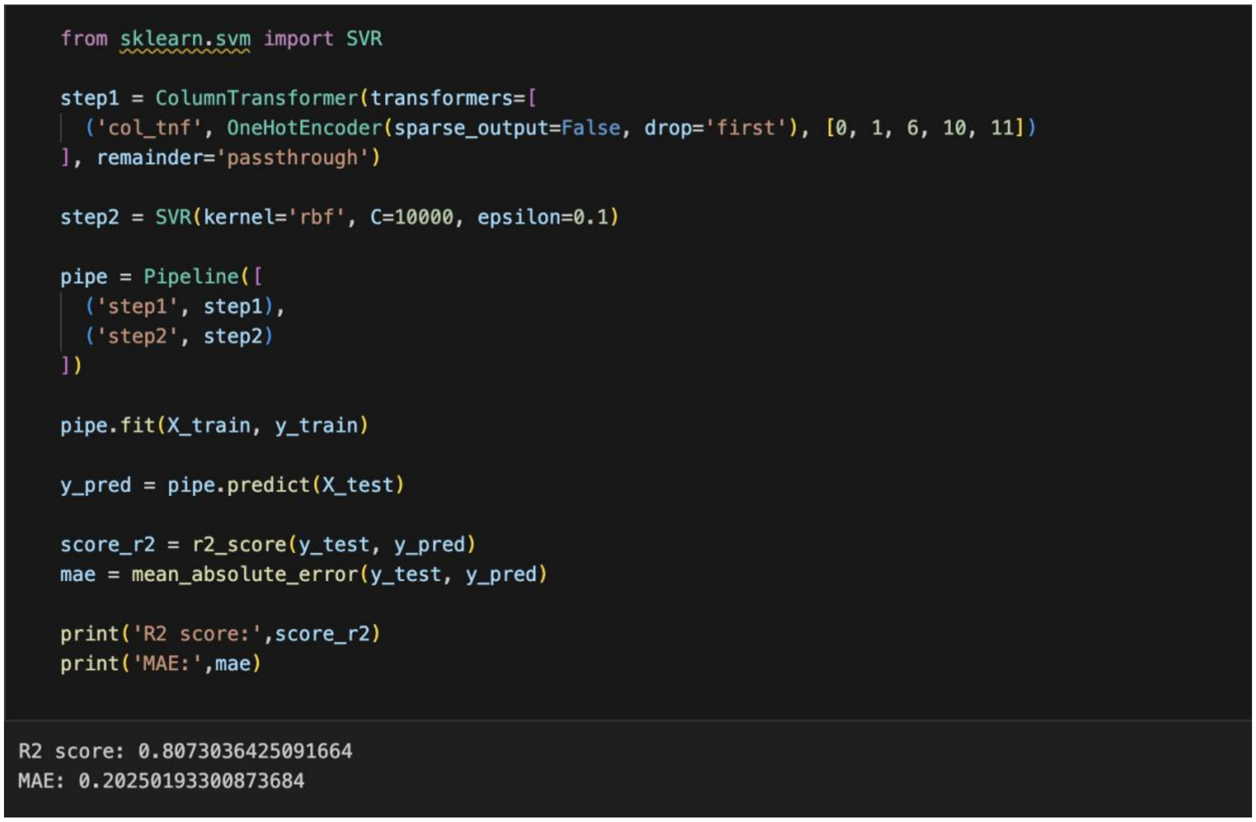
```
R2 score: 0.8487496887205265
MAE: 0.17773776388964113
```

4. Support Vector Regression

Support Vector Regression (SVR) model is trained and evaluated using a pipeline structure similar to the previous models. The ColumnTransformer encodes categorical variables using OneHotEncoder, and the SVR model is instantiated with the 'rbf' kernel, a regularization parameter 'C' set to 10000, and an epsilon value of 0.1.

The pipeline is fitted on the training data ('X_train', 'y_train') and predicts the target variable on the test set ('X_test'). Subsequently, R-squared (r2_score) and Mean Absolute Error (MAE) metrics are calculated to evaluate the model's performance.

Support Vector Regression uses support vector machines to perform regression tasks, aiming to find a hyperplane that best fits the data while maintaining a margin of tolerance (epsilon) around the fitted line. The 'rbf' kernel specifies the radial basis function used in the SVR model. The evaluation metrics (R-squared and MAE) provide insights into how well the SVR model predicts the logarithm of laptop prices.

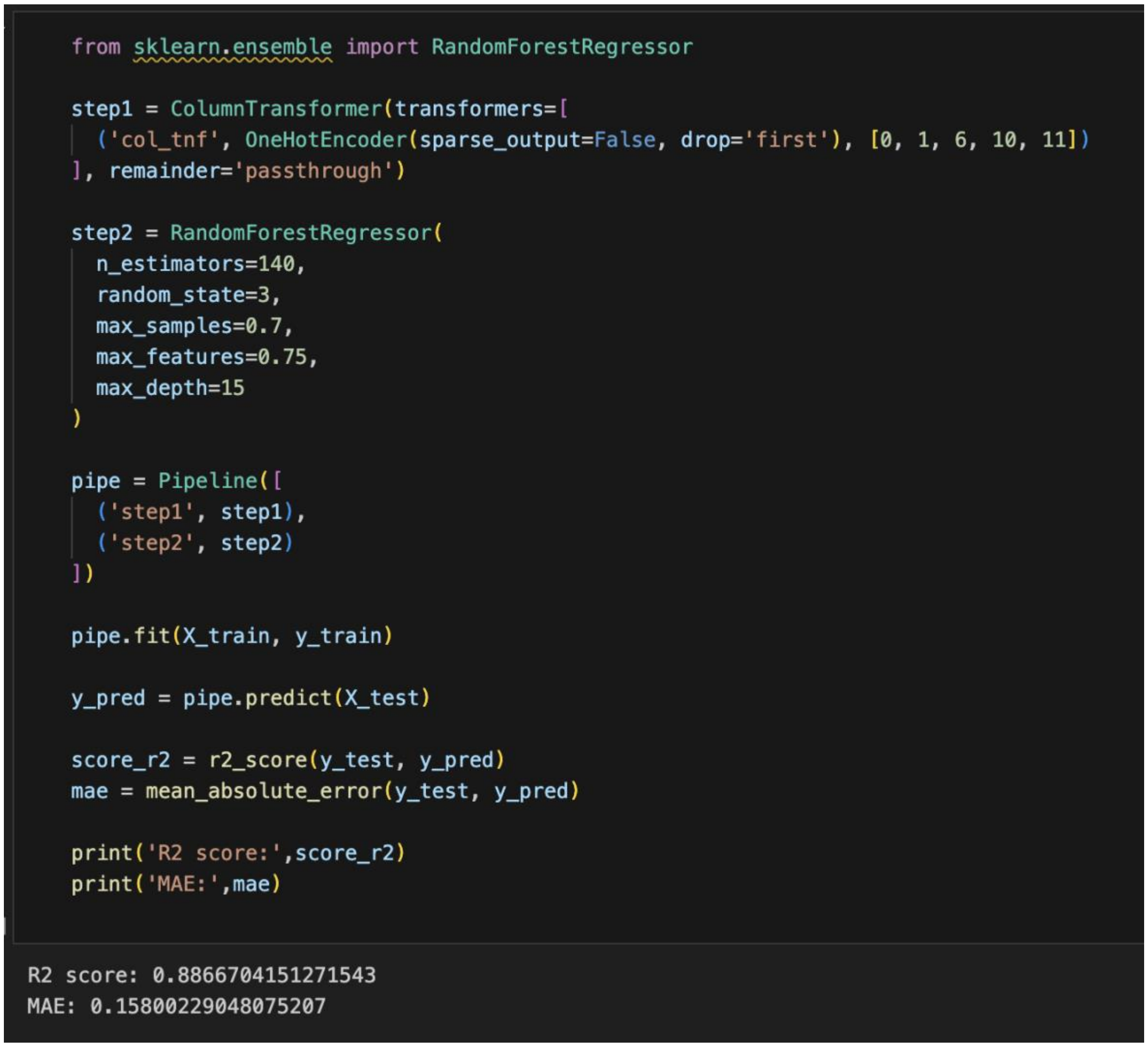


```python
from sklearn.svm import SVR

step1 = ColumnTransformer(transformers=[
    ('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 10, 11])
], remainder='passthrough')

step2 = SVR(kernel='rbf', C=10000, epsilon=0.1)

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

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

```
R2 score: 0.8073036425091664
MAE: 0.20250193300873684
```

5. Random Forest Regression

Here evaluating a Random Forest Regression model using a pipeline similar to previous models. The ColumnTransformer encodes categorical variables using OneHotEncoder, and the RandomForestRegressor is instantiated with specific hyperparameters, including 100 estimators, a random_state of 3, and constraints on 'max_samples', 'max_features', and 'max_depth'.
The pipeline is fitted on the training data ('X_train', 'y_train') and predicts the target variable on the test set ('X_test'). Subsequently, R-squared (r2_score) and Mean Absolute Error (MAE) metrics are calculated to assess the model's performance.
Random Forest Regression builds multiple decision trees and aggregates their predictions to improve accuracy and reduce overfitting. The specified hyperparameters control the number of trees, randomness in feature selection, and tree depth. The evaluation metrics (R-squared and MAE) provide insights into how well the Random Forest model predicts the logarithm of laptop prices.



```
from sklearn.ensemble import RandomForestRegressor

step1 = ColumnTransformer(transformers=[
    ('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 10, 11])
], remainder='passthrough')

step2 = RandomForestRegressor(
    n_estimators=140,
    random_state=3,
    max_samples=0.7,
    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)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

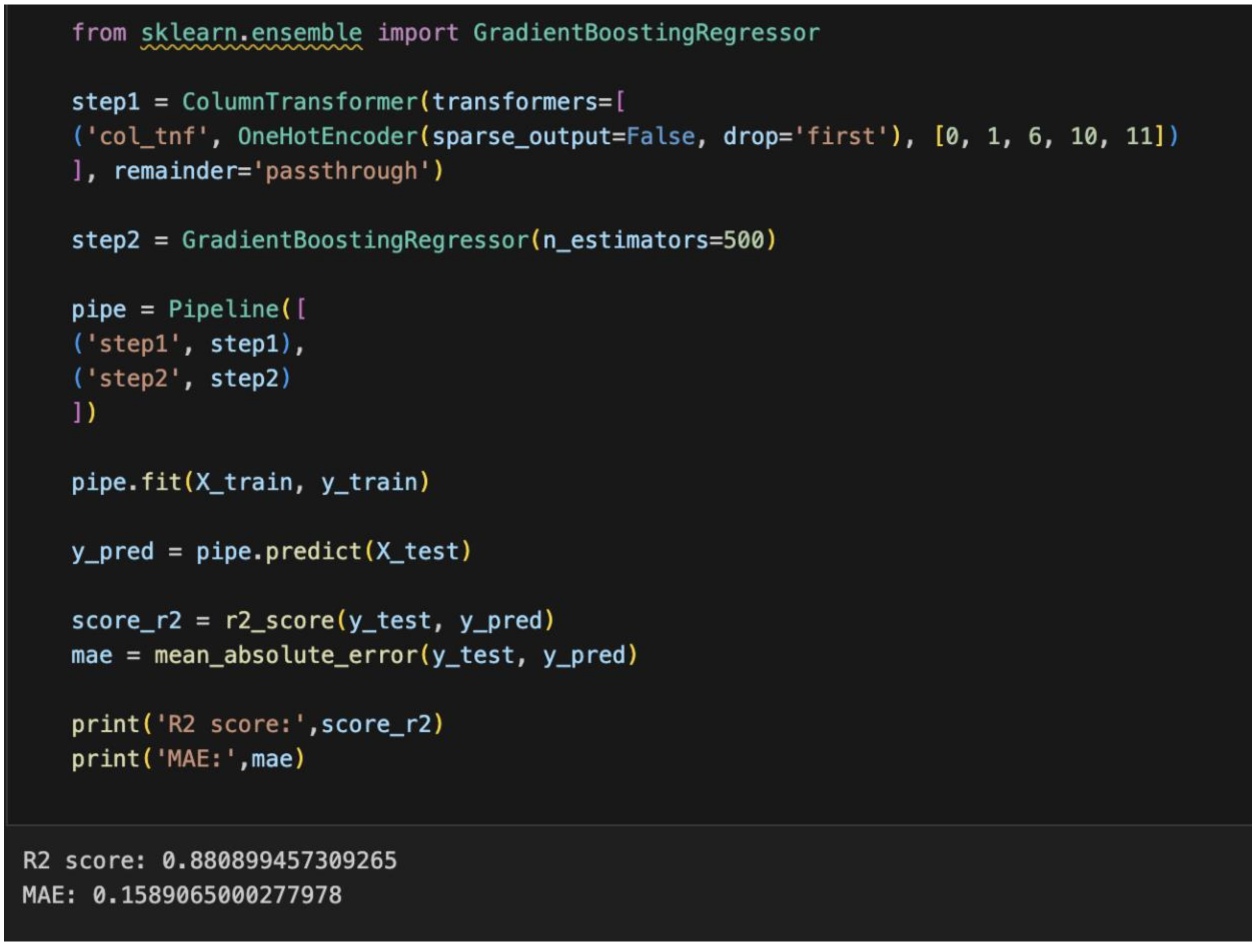
```
R2 score: 0.8866704151271543
MAE: 0.15800229048075207
```

6. Gradient Boosting Regression

Here, a Gradient Boosting Regression model is trained and evaluated using a pipeline structure similar to previous models. The ColumnTransformer encodes categorical variables using OneHotEncoder, and the GradientBoostingRegressor is instantiated with 500 estimators.

The pipeline is fitted on the training data ('X_train', 'y_train') and predicts the target variable on the test set ('X_test'). Subsequently, R-squared (r2_score) and Mean Absolute Error (MAE) metrics are calculated to evaluate the model's performance.

Gradient Boosting Regression builds an ensemble of weak prediction models (typically decision trees) sequentially, where each subsequent model focuses on the errors made by the previous models. The 'n_estimators' parameter controls the number of boosting stages. The evaluation metrics (R-squared and MAE) provide insights into how well the Gradient Boosting model predicts the logarithm of laptop prices.



```python
from sklearn.ensemble import GradientBoostingRegressor

step1 = ColumnTransformer(transformers=[
('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 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)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

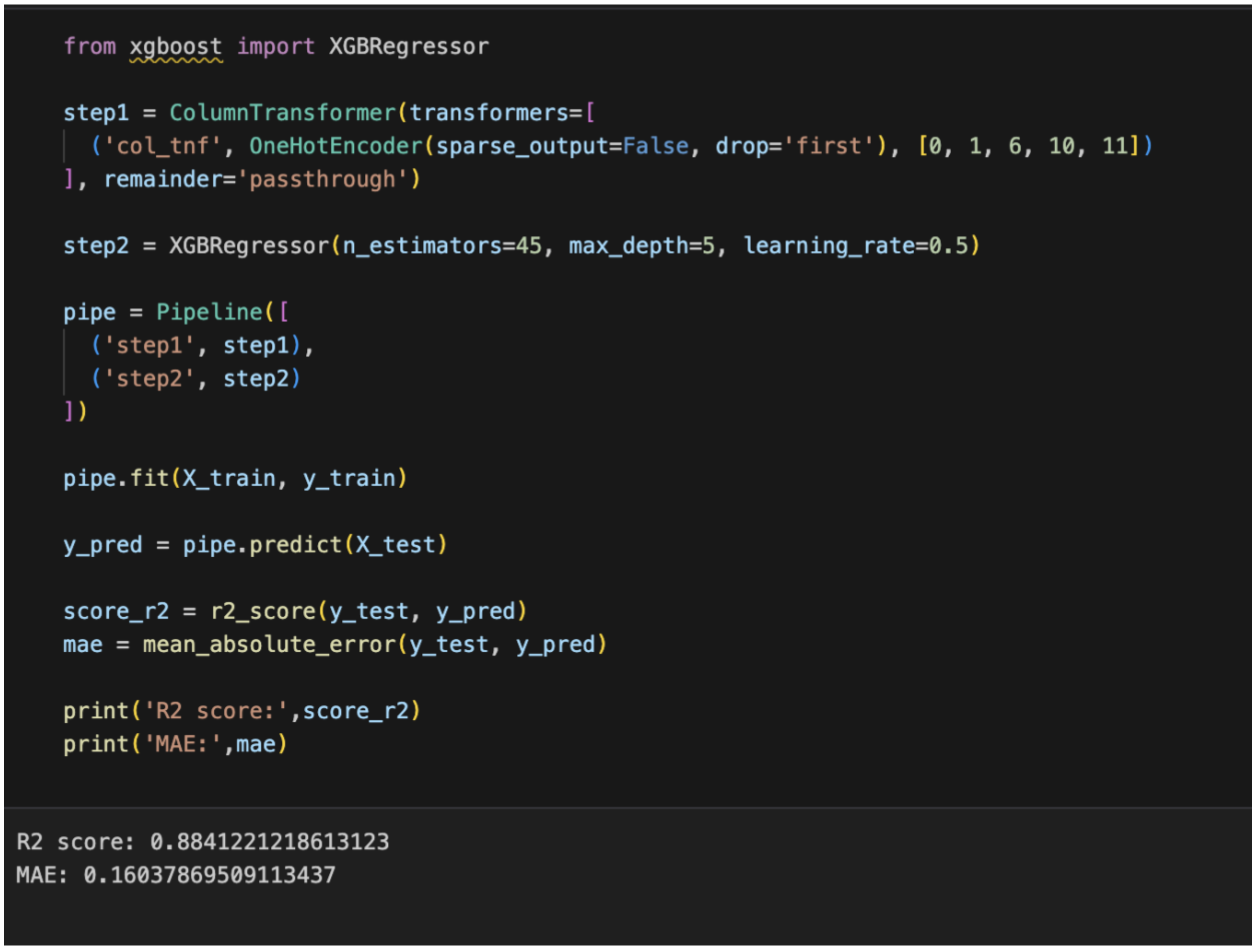
```
R2 score: 0.880899457309265
MAE: 0.1589065000277978
```

7. XGBoost Regression

XGBoost Regression model is trained and evaluated using a pipeline structure similar to previous models. The ColumnTransformer encodes categorical variables using OneHotEncoder, and the XGBRegressor is instantiated with specific hyperparameters, including 45 estimators, a max_depth of 5, and a learning_rate of 0.5.

The pipeline is fitted on the training data ('X_train', 'y_train') and predicts the target variable on the test set ('X_test'). Subsequently, R-squared (r2_score) and Mean Absolute Error (MAE) metrics are calculated to evaluate the model's performance.

XGBoost (Extreme Gradient Boosting) is an optimized gradient boosting library that excels in speed and performance. The 'n_estimators' parameter controls the number of boosting rounds, 'max_depth' defines the maximum depth of the trees, and 'learning_rate' controls the contribution of each tree. The evaluation metrics (R-squared and MAE) provide insights into how well the XGBoost model predicts the logarithm of laptop prices.



```
from xgboost import XGBRegressor

step1 = ColumnTransformer(transformers=[
  ('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 10, 11])
], remainder='passthrough')

step2 = XGBRegressor(n_estimators=45, max_depth=5, learning_rate=0.5)

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

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

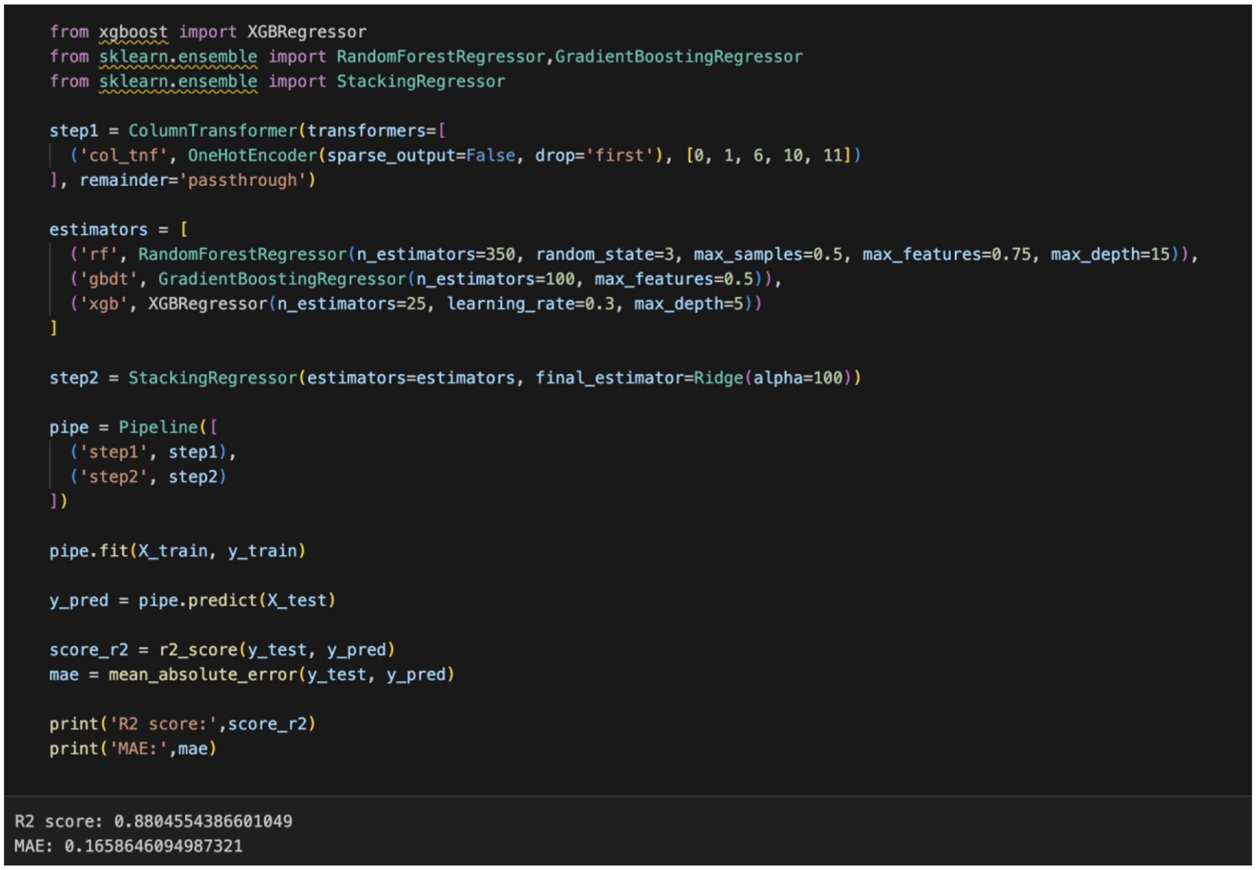
```
R2 score: 0.8841221218613123
MAE: 0.16037869509113437
```

8. Stacking Regression

Stacking Regression model is trained and evaluated using a pipeline structure that combines predictions from multiple base models (Random Forest, Gradient Boosting, XGBoost) with a final estimator (Ridge Regression).

The ColumnTransformer encodes categorical variables using OneHotEncoder, and the StackingRegressor is configured with a list of base estimators, including RandomForestRegressor, GradientBoostingRegressor, and XGBRegressor. The final_estimator is Ridge Regression with an alpha value of 100.

The pipeline is fitted on the training data ('X_train', 'y_train') and predicts the target variable on the test set ('X_test'). Subsequently, R-squared (r2_score) and Mean Absolute Error (MAE) metrics are calculated to evaluate the model's performance. Stacking Regressor combines the predictions of multiple base estimators, enabling the final estimator to learn how to best combine their predictions. This ensemble technique leverages the strengths of various models to potentially improve predictive performance. The evaluation metrics (R-squared and MAE) provide insights into how well the Stacking model predicts the logarithm of laptop prices.



```
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor

step1 = ColumnTransformer(transformers=[
    ('col_tnf', OneHotEncoder(sparse_output=False, drop='first'), [0, 1, 6, 10, 11])
], remainder='passthrough')

estimators = [
    ('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)),
    ('xgb', XGBRegressor(n_estimators=25, learning_rate=0.3, max_depth=5))
]

step2 = StackingRegressor(estimators=estimators, final_estimator=Ridge(alpha=100))

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

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print('R2 score:',score_r2)
print('MAE:',mae)
```

```
R2 score: 0.8804554386601049
MAE: 0.1658646094987321
```

Comparison of evaluation metrics among different models:

Regression Model	**R-squared**	**Mean Absolute Error**
Linear	0.807	0.210
Ridge	0.812	0.209
Decision Tree	0.835	0.177
Support Vector	0.807	0.202
Random Forest	0.886	0.158
Gradient Boosting	0.880	0.159
XGBoost	0.884	0.160
Stacking	0.880	0.165

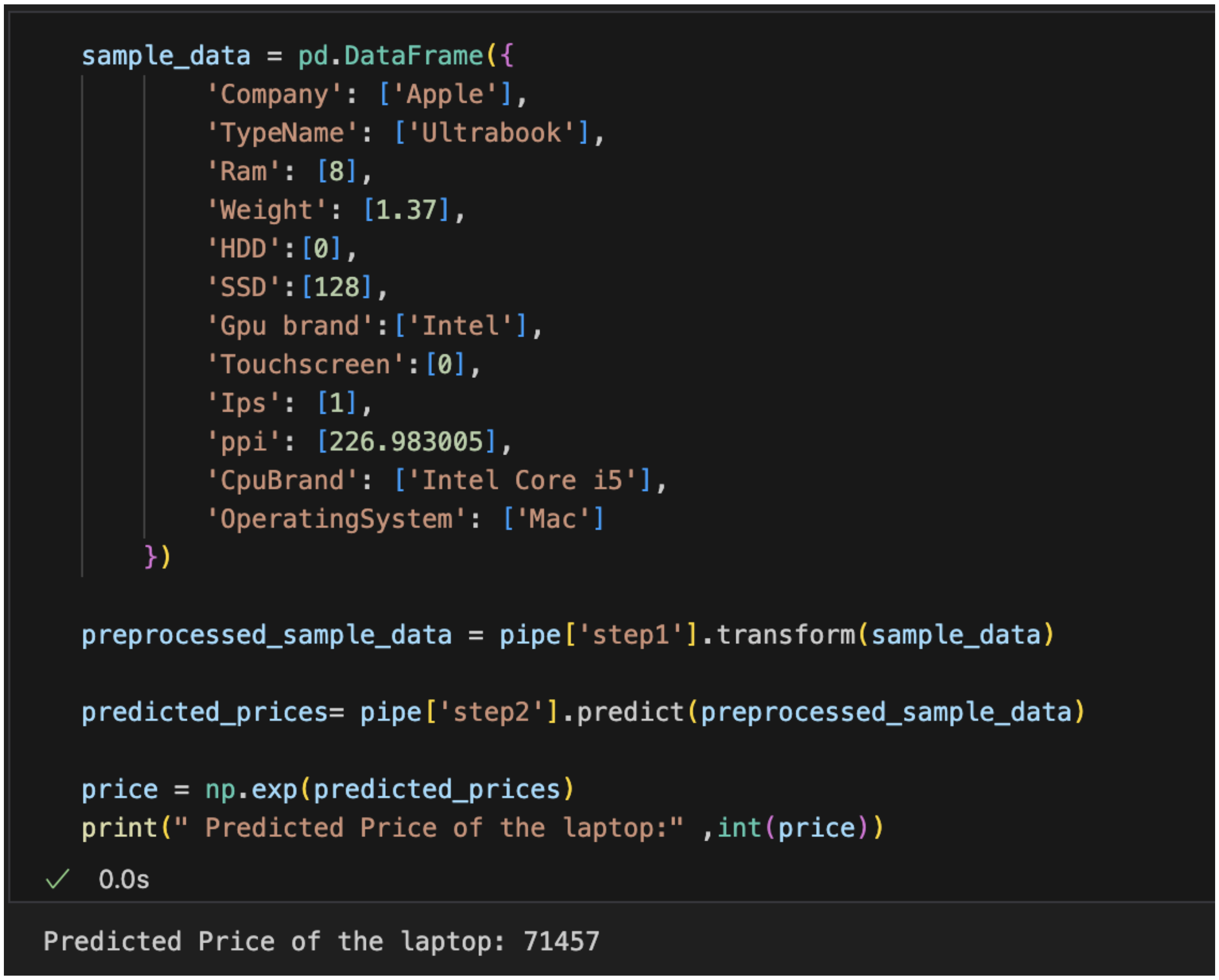
Among the models listed:

The **Random Forest model** has the highest R-squared value (0.886), indicating it explains the variance in laptop prices most effectively among the models tested and also has a low MAE value (0.158) suggesting it provides predictions with minimal error on average.
Therefore, the Random Forest model would be the best choice to use for predicting laptop prices based on the provided performance metrics, as it both explains the data well and provides accurate predictions with minimal error.

Stage 4: Predicting Laptop Price for Sample Data

Using the pre-trained pipeline, the given sample data, which represents laptop specifications, is preprocessed using the pipeline's 'step1' to transform the data. The transformed data is then used to predict the price using the 'step2' of the pipeline. The Random Forest model is used in the 'step2' of the pipeline for predicting laptop prices.

The predicted price is obtained by exponentiating the predicted logarithm of prices and extracting the resulting price value. This process predicts the price for the provided laptop specifications based on the trained model.



```
sample_data = pd.DataFrame({
        'Company': ['Apple'],
        'TypeName': ['Ultrabook'],
        'Ram': [8],
        'Weight': [1.37],
        'HDD':[0],
        'SSD':[128],
        'Gpu brand':['Intel'],
        'Touchscreen':[0],
        'Ips': [1],
        'ppi': [226.983005],
        'CpuBrand': ['Intel Core i5'],
        'OperatingSystem': ['Mac']
    })

preprocessed_sample_data = pipe['step1'].transform(sample_data)

predicted_prices= pipe['step2'].predict(preprocessed_sample_data)

price = np.exp(predicted_prices)
print(" Predicted Price of the laptop:" ,int(price))
```

✓ 0.0s

```
Predicted Price of the laptop: 71457
```

Conclusion:

In this project, various machine learning models were developed and evaluated to predict laptop prices based on historical data and relevant laptop specifications. After careful preprocessing and feature engineering, multiple regression models were implemented, including Linear Regression, Ridge Regression, Decision Tree Regression, Support Vector Regression, Random Forest Regression, Gradient Boosting Regression, XGBoost Regression, and Stacking Regression. The performance of each model was assessed using R-squared and Mean Absolute Error (MAE) metrics.

1. Random Forest Regression: Outperformed all other models with the highest R-squared value (0.886) and the lowest MAE (0.158). This model proved to be the most accurate and consistent predictor for laptop prices in this project.
2. XGBoost, Gradient Boosting Regression and Stacking Regression: Achieved competitive results close to Random Forest Regression, highlighting their effectiveness in predictive modeling.
3. Feature Importance: The Random Forest model identified the most critical features influencing laptop prices, providing valuable insights into market trends and customer preferences.

This project's results can be leveraged to assist buyers and sellers in making data-driven decisions, ultimately fostering transparency and efficiency in the laptop market.