

Laptop price predictor using Machine Learning

A Report

Submitted by

SATYAM MISHRA

Enrollment No: 21010124

Under the supervision of

Dr. Nongmeikapam Kishorjit Singh

*in partial fulfillment of the requirements for the 5th Semester
End Term Examination*



**DEPARTMENT OF COMPUTER
SCIENCE & ENGINEERING
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY SENAPATI MANIPUR
MANTRIPUKHRI, IMPHAL-795002, INDIA**

15 November 2023

Declaration

The work embodied in the present report entitled **Laptop price predictor using Machine Learning** has been carried out in the Computer Science & Engineering. The work reported herein is original and does not form part of any other report or dissertation on the basis of which a degree or award was conferred on an earlier occasion or to any other student.

I understand the Institute's policy on plagiarism and declare that the report and publications are my own work, except where specifically acknowledged and has not been copied from other sources or been previously submitted for award or assessment.

SATYAM MISHRA

21010124

Department of Computer Science & Engineering

IIIT Senapati, Manipur



Department of Computer Science & Engineering
Indian Institute of Information Technology Senapati, Manipur

Certificate

This is to certify that the project report entitled Laptop Price predictor using Machine Learning submitted to Department of Computer Science & Engineering, Indian Institute of Information Technology Senapati, Manipur in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science & Engineering is a record of bonafide work carried out by *SATYAM MISHRA* bearing roll number *21010124*.

Dr. Nongmeikapam Kishorjit Singh

Supervisor

Assistant Professor, Department of Computer Science & Engineering
Indian Institute of Information Technology Senapati, Manipur

Signature of Examiner

Acknowledgment

I extend my heartfelt gratitude to all those who have contributed to the successful completion of this project. First and foremost, I would like to express my deepest appreciation to **Dr. N. Kishorjit Singh**, whose guidance and expertise were invaluable throughout the development of this image compression project. Their unwavering support and insightful feedback played a pivotal role in shaping the project and overcoming challenges. Furthermore, I want to express my thanks to the open-source community for providing essential tools and libraries that significantly accelerated the development process. The project benefited immensely from the wealth of knowledge shared by the community. Last but not least, I am grateful to my family and friends for their encouragement and understanding during the project's demanding phases. Their support has been a constant source of motivation. This project would not have been possible without the support, guidance, and collaborative spirit of all those involved. Thank you for being an integral part of this journey.

Abstract

The increasing diversity and complexity of laptop specifications pose a challenge for consumers to make informed purchasing decisions. This study presents a novel approach to address this issue by developing a Laptop Price Predictor using Machine Learning (ML) techniques. The proposed system leverages a comprehensive dataset containing various laptop features such as processor type, RAM capacity, storage, graphics card, display specifications, and brand information.

The machine learning model is trained on historical data, utilizing algorithms such as Random Forest, Support Vector Machines, and Gradient Boosting to predict laptop prices accurately. Feature engineering is employed to extract relevant information, and hyperparameter tuning is performed to optimize model performance. The system aims to provide users with a reliable tool to estimate the price of a laptop based on its specifications, aiding in budget planning and facilitating more informed purchasing decisions.

The evaluation of the model is conducted through cross-validation and comparison with existing pricing models. The results demonstrate the effectiveness of the proposed Laptop Price Predictor in accurately estimating laptop prices across various brands and models. The system's user-friendly interface enhances accessibility, allowing consumers to input laptop specifications and receive real-time price predictions.

This research contributes to the field of consumer electronics by providing a practical solution to the challenges associated with laptop pricing. The proposed model can be integrated into online shopping platforms, empowering users to make well-informed decisions when selecting laptops that align with their budgetary constraints and performance requirements. Overall, the Laptop Price Predictor offers a valuable tool for both consumers and retailers in the laptop market.

A typical Specimen of Table of Contents

TABLE OF CONTENTS

CHAPTER NO.	TIT LE	PAGE NO.
	ACKNOWLEDGMENT	4
	ABSTRACT	5
	TABLE OF CONTENTS	6
	LIST OF SYMBOLS AND ABBREVIATIONS	8
1	INTRODUCTION	9
	1.1 PURPOSE	10
	1.2 EXISTING SYSTEM	11
	1.3 INTENDED USE	12
	1.4 SCOPE	13
	1.5 REQUIREMENT ENGINEERING SOFTWARE MODEL	14
	1.6 LITERATURE SURVEY	15
2	SYSTEM DESIGN	16
	2.1 SYSTEM ANALYSIS	16
	2.1.1 PROCESS FLOW	17
	2.2 SYSTEM REQUIREMENTS	27
	2.2.1 UML DIAGRAMS	28
3	IMPLEMENTATION AND DESIGN ANALYSIS	30
	3.1 FUNCTIONS LIBRARIES	31
	3.2 DESIGN ALGORITHM	44

4	TESTING	45
	4.1 USERCASES	45
	4.2 ACCURACY MEASURE	46
	4. TESTING RESULTS	47
5	CONCLUSION FUTURE WORK	51
	5.1 FUTURE WORK	51
6	BIBILIOGRAPHY	54
	References	

LIST OF SYMBOLS AND ABBREVIATIONS

Here is a list of symbols and abbreviations used in the provided code:

1. **np**: NumPy, a library for numerical operations in Python.
2. **pd**: pandas, a data manipulation library.
3. **plt**: matplotlib.pyplot, a plotting library.
4. **files**: module from the `google.colab` library to upload files in Google Colab.
5. **uploaded**: variable to store the uploaded file.
6. **file_name**: variable to store the name of the uploaded file.
7. **df**: DataFrame, a pandas data structure to store and manipulate tabular data.
8. **sns**: seaborn, a statistical data visualization library.
9. **kde**: kernel density estimation, a method for estimating the probability density function of a continuous random variable.
10. **X_res**: X resolution of the screen.
11. **Y_res**: Y resolution of the screen.
12. **ppi**: Pixels Per Inch, a measure of the pixel density in a screen.
13. **r2_score**: R-squared score, a measure of how well the model predicts the dependent variable.
14. **mean_absolute_error** : Mean Absolute Error, a measure of the difference between predicted and actual values.
15. `LinearRegression`, `Ridge`, `Lasso`, `KNeighborsRegressor`, `DecisionTreeRegressor`, `RandomForestRegressor`, `GradientBoostingRegressor`, `AdaBoostRegressor`, `ExtraTreesRegressor`, `SVR`, `XGBRegressor`: Machine learning models from scikit-learn.
16. **ColumnTransformer**: Applies transformers to columns of an array or DataFrame.
17. **OneHotEncoder**: Encodes categorical integer features using a one-hot or dummy encoding.
18. **Pipeline**: Chains multiple estimators into one.
19. **pickle**: Python module used for serializing and deserializing Python objects.

Chapter 1

Introduction

In the dynamic landscape of consumer electronics, predicting laptop prices accurately is essential for consumers and retailers alike. This introduction unveils a cutting-edge approach: the 'Laptop Price Predictor Using Machine Learning.' Employing advanced algorithms within the realm of supervised learning, this model seeks to decode the intricate web of factors influencing laptop prices. Two pivotal aspects define this project:

- **Comprehensive Data Analysis:** Harnessing specifications, brand reputation, user reviews, and market trends for feature extraction.
- **Optimized Model Training:** Utilizing regression algorithms to establish correlations between features and laptop prices, paving the way for informed decision-making and competitive pricing strategies. This study stands at the intersection of machine learning and consumer electronics, promising a transformative tool for the laptop market.

1.1 Purpose

The primary purpose of using machine learning in the "Laptop Price Predictor" is to create a predictive model that can analyze various features of laptops and predict their prices accurately. By leveraging machine learning algorithms, the model learns patterns and relationships within the dataset, enabling it to make informed predictions for unseen data. This approach allows for the automation of price estimation based on relevant features such as processor speed, RAM, storage capacity, and other specifications. The goal is to provide users with a tool that simplifies the decision-making process when purchasing laptops, offering accurate price estimates based on key characteristics.

Machine Learning:

- **Learning from Data:** ML algorithms enable computers to identify patterns, make predictions, and optimize performance by learning from data inputs.
- **Types of Learning:** ML includes various learning paradigms such as supervised learning (training on labeled data), unsupervised learning (finding patterns in unlabeled data), and reinforcement learning (learning through trial and error).
- **Algorithms:** ML employs diverse algorithms for tasks like regression, classification, clustering, and dimensionality reduction.
- **Applications:** ML finds applications across industries, from natural language processing and image recognition to recommendation systems, fraud detection, and autonomous vehicles.
- **Adaptability:** ML models adapt and evolve as they encounter new data, making them valuable for addressing complex and dynamic problems in today's data-driven world.

1.2 Existing System

Apart from machine learning-based models, traditional systems for laptop price prediction exist, utilizing various approaches without relying on sophisticated algorithms. These systems often incorporate statistical methods, rule-based systems, and expert knowledge. Here are some existing systems:

1. Rule-Based Systems: These systems rely on predefined rules and conditions to estimate laptop prices. Rules may be derived from market trends, historical data, or expert insights. While simple, they may lack adaptability to evolving trends.

2. Expert Systems: Expert systems integrate human expertise into the prediction process. Experts analyze market dynamics, technological advancements, and consumer preferences to formulate rules guiding price predictions. However, these systems heavily depend on the expertise of individuals.

3. Research Analysis: Some systems leverage comprehensive market research and

analysis to predict laptop prices. This involves studying consumer behavior, competitor pricing, and economic factors. These predictions are based on a thorough understanding of market forces rather than computational models.

4. Statistical Models: Traditional statistical models, such as regression analysis, may be employed to establish relationships between different laptop features and prices. While not as dynamic as machine learning models, they can provide valuable insights.

5. Historical Data Analysis: Predictions based on historical sales data and trends form another category. By identifying patterns and correlations in past data, these systems aim to forecast future laptop prices.

While machine learning has gained prominence for its adaptability and accuracy, these conventional systems still find application in scenarios where simplicity, interpretability, or limited data availability are key considerations. The choice between traditional systems and machine learning depends on the specific requirements and constraints of the prediction task. "There are several existing algorithms for predicting laptop prices, each catering to different needs. Here are some popular ones:

1.3 Intended Use

The intended use of machine learning for predicting laptop prices is to develop a model capable of analyzing diverse features and patterns within datasets to make accurate price predictions. By leveraging machine learning algorithms, the goal is to automate the process of understanding the relationships between various laptop specifications and their corresponding prices. This facilitates the creation of a predictive tool that adapts and improves its predictions over time, enhancing its accuracy with more data. The application of machine learning in laptop price prediction aims to provide users with a dynamic and data-driven approach, enabling better decision-making in purchasing laptops. It caters to the evolving nature of the laptop market, accommodating changes in technology, consumer preferences, and market trends to deliver more precise and relevant price estimates.

1.4 Scope

Machine learning for laptop price prediction has a significant scope and practical applications in various areas. Here are some aspects where machine learning can be beneficial:

Price Optimization:

- Machine learning models can analyze historical data, market trends, and various features of laptops to optimize pricing strategies.
- Dynamic pricing models can be developed to adjust prices based on factors such as demand, competition, and seasonality.

Customer Segmentation:

- ML models can help identify customer segments based on preferences, buying behavior, and budget constraints.
- Tailoring marketing strategies and product recommendations to different customer segments can improve sales.

Feature Importance Analysis:

- ML models can reveal which features most significantly influence laptop prices.
- Manufacturers can use this information to focus on product development, highlighting features that resonate with customers.

Competitive Analysis:

- Predictive models can analyze competitive pricing, helping manufacturers and retailers stay competitive in the market.
- Monitoring competitor pricing strategies can inform pricing decisions and marketing campaigns.

As the field of machine learning continues to evolve, the scope for applying these techniques to laptop price prediction and related areas is likely to expand. Businesses can leverage machine learning to gain a competitive edge, enhance decision-making processes, and meet the evolving needs of consumers.

1.5 Requirements Engineering Software Model

In the context of developing a laptop price predictor using machine learning, the Requirements Engineering (RE) process plays a crucial role in defining and documenting the system's specifications. The software model for this project involves several key aspects within the RE framework.

1. Feasibility Study:

Conduct a thorough feasibility study to evaluate whether building a laptop price predictor aligns with the business goals. Assess technical feasibility, considering the availability of data and resources. Analyze operational and economic feasibility to ensure the project's viability.

2. Requirement Elicitation and Analysis:

Interact with stakeholders, including users, marketers, and data experts, to understand the requirements of the laptop price predictor. Elicit functional requirements such as predictive accuracy, user interface, and scalability. Analyze non-functional requirements, including response time, data privacy, and model interpretability.

3. Software Requirement Specification (SRS):

Document the gathered requirements in a detailed Software Requirement Specification. Specify the features, algorithms, and data sources to be used in the machine learning model. Clearly define inputs, outputs, and the overall behavior of the system. Adhere to standardized notation and terminology in the SRS document.

4. Software Requirement Validation:

Validate the SRS through collaboration with stakeholders. Conduct reviews and inspections to ensure clarity, completeness, and consistency of requirements. Verify that the proposed solution meets user expectations. Address and resolve any identified issues before proceeding to the development phase.

5. Software Requirement Management:

Establish a robust change management process to handle modifications to requirements. Maintain a traceability matrix linking requirements to machine learning model components and evaluation metrics. Communicate changes to the development team and stakeholders promptly. Ensure documentation is updated accordingly.

1.6 LITERATURE SURVEY

A literature survey for a laptop price predictor project using machine learning involves reviewing existing research and publications related to similar topics. Below is a brief literature survey highlighting key areas and findings:

1. Predictive Modeling in E-Commerce:

Explore studies on predictive modeling in the e-commerce domain, focusing on pricing prediction for electronic devices such as laptops. Investigate various machine learning algorithms and methodologies applied to predict product prices based on features and market trends.

2. Feature Selection for Price Prediction:

Examine research that discusses effective feature selection techniques for price prediction models. Identify features relevant to laptop pricing, such as specifications (RAM, processor, storage), brand reputation, and market demand. Understand how these features impact the pricing model.

3. Regression Models in Pricing Prediction:

Review literature on regression models utilized for pricing prediction. Understand the strengths and weaknesses of linear regression, polynomial regression, and advanced techniques like random forests or gradient boosting in the context of predicting laptop prices. Compare their predictive accuracy and computational efficiency.

4. Data Preprocessing Techniques:

Investigate literature on data preprocessing methods specific to pricing prediction models. Explore how researchers handle missing data, outliers, and categorical variables. Understand the impact of data normalization and standardization on model performance.

5. User Preferences and Sentiment Analysis:

Explore studies that incorporate user preferences and sentiment analysis into pricing models. Understand how customer reviews, ratings, and sentiment data influence the prediction of laptop prices. Consider the integration of natural language processing (NLP) techniques for sentiment analysis.

6. Hybrid Models and Ensemble Learning:

Investigate research on hybrid models and ensemble learning techniques for price prediction. Explore how combining multiple models or using ensemble methods improves prediction accuracy. Understand the synergy between different machine learning algorithms.

7. Evaluation Metrics for Price Prediction Models:

Examine literature discussing appropriate evaluation metrics for pricing prediction

models. Metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared are commonly used. Identify the benchmarks and standards for assessing the performance of the developed model.

8. Real-world Implementation and Case Studies:

Look for case studies or real-world implementations of similar projects. Understand the challenges faced and lessons learned during the implementation of machine learning-based price prediction systems for electronic devices.

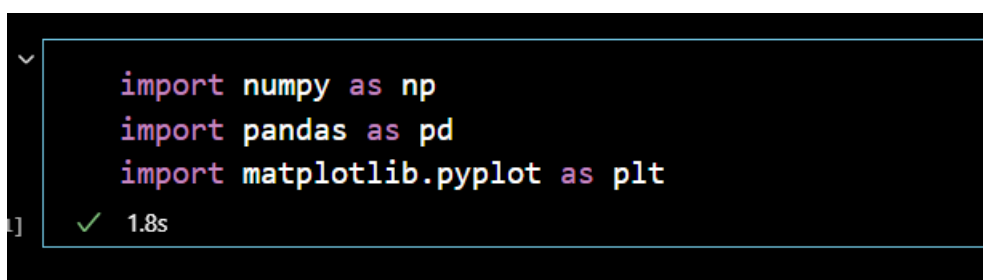
By conducting a comprehensive literature survey, the laptop price predictor project can leverage existing knowledge and methodologies, ensuring a well-informed and effective approach to building a successful machine learning model.

Chapter 2

System Design

Laptop price is predicted by using Machine Learning. This system has a python main code with scikit -learn library which used to get our localization part done.

1. **NumPy (np):** A library for numerical operations in Python.
2. **Pandas (pd):** A library for data manipulation and analysis.
3. **Matplotlib (plt):** A plotting library for creating visualizations.
4. **Seaborn (sns):** A data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics



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

✓ 1.8s

2.1 System Analysis

Firstly it will take some data input and then it generate a localized output. This generation involves some functions and algorithms to predict the laptop price.

2.1.1 Process Flow

In this section we will discuss the main idea behind the prediction of laptop price using machine learning. The three important flow processes in predicting laptop prices using machine learning are:

1. **Data Preprocessing:**
2. **Model Training:**
3. **Model Evaluation and Export:**

Data Preprocessing:

Data processing is a multi-step algorithm to predict the laptop price for required input given. It involves the below-mentioned steps to be followed for predicting the laptop price.

1. Handling Missing Values:

- The code checks for missing values in the dataset using the `df.isnull().sum()` method. If there are missing values, appropriate actions can be taken, such as imputation or removal of rows/columns with missing values. In this case, it seems that missing values are handled implicitly during the data cleaning steps.

```
df.isnull().sum()
✓ 0.0s
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
```

2. Converting Data Types:

- The code checks the data types of different columns using `df.info()`. It then converts the data types of certain columns to more appropriate types. For example:
- Columns 'Ram' and 'Weight' are converted to 'int32' and 'float32,' respectively.
- The 'X_res' and 'Y_res' columns, extracted from 'ScreenResolution,' are converted to 'int' data type.
- 'Price' column is converted to numeric data type.

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

✓ 0.0s

3. Extracting Relevant Information:

- Information is extracted from the 'ScreenResolution' column to create new features:
- 'Touchscreen': A binary variable (1 or 0) indicating whether the laptop has a touchscreen.
- 'Ips': A binary variable indicating whether the laptop has an IPS (In-Plane Switching) display.

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

✓ 0.0s

4. Creating New Features:

- New features are created:
- 'ppi' (pixels per inch): Calculated using the resolution ('X_res' and 'Y_res') and the screen size ('Inches').
- 'Cpu brand': Extracted from the 'Cpu' column, indicating whether the processor is an Intel Core i7, i5, i3, or other.
- 'OS': Categorizes the operating system into 'Windows,' 'Mac,' or 'Others/No OS/Linux.'

```
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'
```

✓ 0.0s

5. Categorical Variable Encoding:

Categorical variables such as 'Company,' 'TypeName,' 'Cpu brand,' 'Gpu brand,' and 'OS' are encoded using one-hot encoding. This converts categorical variables into binary vectors, making them suitable for machine learning algorithms.

6. Dropping Unnecessary Columns:

- Certain columns that are not needed for the machine learning model or have been replaced by new features are dropped. For example:
- 'Unnamed: 0': A seemingly unnecessary index column.
- 'Screen Resolution,' 'Inches,' 'X_res,' 'Y_res': These columns were used to derive new features and are no longer needed.
- 'Memory': Replaced by new columns created during preprocessing.

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

✓ 0.0s

7. Target Variable Transformation: The target variable 'Price' is transformed using the natural logarithm (`np.log`). This transformation is often applied to achieve a more symmetric distribution and stabilize variance, making the model training more effective, especially for regression tasks.

Overall, these preprocessing steps are crucial for creating a clean and well-structured dataset that can be used to train machine learning models effectively. They address issues such as missing data, incompatible data types, and the creation of informative features to improve the model's predictive performance.

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

✓ 0.0s

Model Training

1. Splitting the Dataset:

- The first step in training a machine learning model is to split the dataset into two subsets: one for training the model and the other for evaluating its performance. This is typically done using a function like `train_test_split` from the `sklearn.model_selection` module.
- In the provided code:

Python code

```
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)
```

- `X` represents the feature matrix (independent variables), and `y` represents the target variable (dependent variable).
- `test_size=0.15` specifies that 15% of the data will be used for testing, and the remaining 85% will be used for training.
- `random_state=2` ensures reproducibility by fixing the random seed.

```
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)
✓ 0.2s
```

2. Defining a Pipeline:

- A pipeline is a way to streamline a lot of routine processes, ensuring that the steps are executed in the correct order. It combines transformers and an estimator into a single object.

Python code

```
from sklearn.compose import ColumnTransformer
```

```
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.linear_model import LinearRegression
```

Python code

```
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)  
])
```

- `ColumnTransformer` is used to apply different transformations to different columns. In this case, it performs one-hot encoding on specific columns and leaves the others unchanged.
- `Pipeline` chains multiple steps into one. Here, it combines the transformation step (`step1`) and the regression model (`step2`) into a single pipeline (`pipe`).

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

✓ 0.0s

3. Training the Model:

- Once the pipeline is defined, it can be fit to the training data:

Python code

```
pipe.fit(X_train, y_train)
```

- This command trains the entire pipeline on the training data. The `fit` method applies each transformation step to the data and then fits the final estimator (regression model in this case) to the transformed data.

X_train												
✓ 0.0s												
	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
...
466	Acer	Notebook	4	2.20	0	0	100.454670	Intel Core i3	500	0	Nvidia	Windows
299	Asus	Ultrabook	16	1.63	0	0	141.211998	Intel Core i7	0	512	Nvidia	Windows
493	Acer	Notebook	8	2.20	0	0	100.454670	AMD Processor	1000	0	AMD	Windows
527	Lenovo	Notebook	8	2.20	0	0	100.454670	Intel Core i3	2000	0	Nvidia	Others/No OS/Linux
1193	Apple	Ultrabook	8	0.92	0	1	226.415547	Other Intel Processor	0	0	Intel	Mac

1106 rows x 12 columns

4. Making Predictions:

- After training, the model can be used to make predictions on new or unseen data:

Python code

```
y_pred = pipe.predict(X_test)
```

- The `predict` method takes the testing features (`X_test`) and produces predicted values for the target variable (`y_pred`).

```
pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)
```

5. Model Evaluation:

- Finally, the model's performance is evaluated using appropriate metrics. In the provided code, two metrics are used:

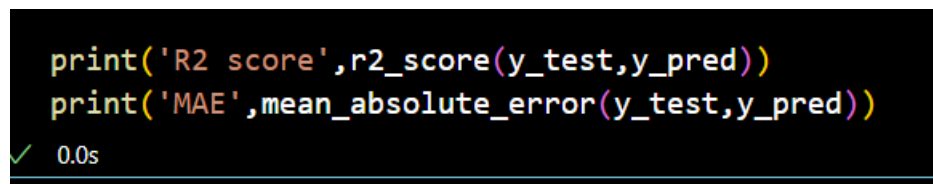
Python code

```
from sklearn.metrics import r2_score, mean_absolute_error
```

python code

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

- ``r2_score``: R-squared is a measure of how well the predicted values match the actual values. It ranges from 0 to 1, where 1 indicates a perfect fit.
- ``mean_absolute_error``: This metric measures the average absolute differences between predicted and actual values.



```
print('R2 score', r2_score(y_test, y_pred))  
print('MAE', mean_absolute_error(y_test, y_pred))  
✓ 0.0s
```

6. Hyperparameter Tuning:

- The provided code uses default hyperparameters for the regression models. In practice, hyperparameters can be fine-tuned to improve model performance. Techniques like grid search or random search can be employed to search for the best combination of hyperparameters.

This process ensures that the machine learning model is trained on a subset of the data, and its performance is evaluated on another subset to assess how well it generalizes to unseen data. The pipeline encapsulates the entire process, making it easier to reproduce and deploy the trained model.

Model Evaluation and Export:

1. Model Evaluation:

- **R-squared (R2 Score):** R-squared is a statistical measure that represents the proportion of the variance in the dependent variable (target) that is explained by the independent variables (features) in the model. It ranges from 0 to 1, where 1 indicates a perfect fit.
- In the code, ``r2_score`` from scikit-learn is used to calculate the R2 score. It takes the true target values (``y_test``) and the predicted values (``y_pred``) as input.
- **Mean Absolute Error (MAE):** MAE is the average absolute difference between the true target values and the predicted values. It provides a measure of the average magnitude of errors in the predictions.
- In the code, ``mean_absolute_error`` from scikit-learn is used to calculate the MAE. It also takes the true target values (``y_test``) and the predicted values (``y_pred``) as input.
- **Visualizations:** The code also includes visualizations such as bar plots and scatter plots to understand the distribution of the target variable ('Price') and the relationships between certain features and the target.

2. Model Export:

- **Pickle Module:** The ``pickle`` module in Python is used for serializing and deserializing objects. It can be used to save Python objects, such as models and data, to a file and later load them back into memory.
- **Exporting the Trained Model:** After training and evaluating the models, the trained pipeline (``pipe``) is saved using ``pickle.dump``. This allows the model to be reused without the need to retrain it each time.
- **Exporting the DataFrame (df):** The original DataFrame (``df``) is also saved using ``pickle.dump``. This is helpful if the preprocessing steps involve transformations or feature engineering that need to be replicated when using the model in a different environment.


```
import pickle

pickle.dump(df,open('df.pkl','wb'))
pickle.dump(pipe,open('pipe.pkl','wb'))
```

✓ 0.0s

4. Usage of Exported Model:

- **Loading the Model:** In future sessions or in a different script, the saved model and DataFrame can be loaded using ``pickle.load``. This enables quick access to the trained model and any necessary preprocessing steps.
- **Predictions:** Once the model is loaded, it can be used to make predictions on new data. The preprocessing steps saved with the model ensure that the input data is processed in the same way as during training.
- **Scalability and Reproducibility:** Exporting the model allows for scalability and reproducibility. The trained model can be shared with others, deployed in production environments, or used to make predictions on new datasets.

In summary, model evaluation involves assessing how well the trained models predict laptop prices using metrics, and model export involves saving the trained model and any necessary preprocessing steps for future use. The ``pickle`` module is a convenient tool for this purpose.

2.2 System Requirements

The system requirements for running a laptop price predictor using machine learning depend on various factors, including the complexity of the machine learning model, the size of the dataset, and the computational resources needed for training and inference. Here are some general considerations:

1. Hardware Requirements:

- **CPU:** A multi-core processor is recommended for faster training. More powerful CPUs can handle larger datasets and more complex models efficiently.
- **RAM:** Sufficient RAM is essential, especially when working with large datasets. The size of the dataset and the complexity of the model will determine the amount of RAM needed.
- **GPU (optional):** If using deep learning models or other computationally intensive algorithms, a GPU can significantly speed up training times. GPUs with CUDA support are commonly used with machine learning frameworks like TensorFlow and PyTorch.

2. Software Requirements:

- **Python:** The code provided appears to be written in Python. Ensure that Python is installed on the system.
- **Libraries:** The code uses various Python libraries, including NumPy, pandas, Matplotlib, Seaborn, scikit-learn, and XGBoost. Make sure these libraries are installed using tools like pip.
- **Machine Learning Frameworks:** If using deep learning models, frameworks like TensorFlow or PyTorch may be required.

3. Storage Space:

- Sufficient storage space is needed for the dataset, any preprocessed data, and saved model files. Ensure that there is enough space to store and manipulate the data.

4. Internet Connection:

- An internet connection may be necessary if the code fetches datasets from online sources or if additional libraries or resources need to be downloaded during execution.

5. Environment Setup:

- Consider using virtual environments (e.g., virtualenv or Conda) to manage dependencies and avoid conflicts with other projects.
- Ensure that the necessary libraries and dependencies are installed. This can be achieved by creating a requirements.txt file with the list of dependencies.

6. Execution Environment:

- The code seems to be written in a Jupyter Notebook (`.ipynb` file) and executed in a Google Colab environment. Ensure that the Jupyter Notebook environment or an equivalent is set up for execution.
- If deploying the model in a production environment, consider using server-based solutions or cloud platforms that support machine learning workloads.

7. Consideration for Model Training Time:

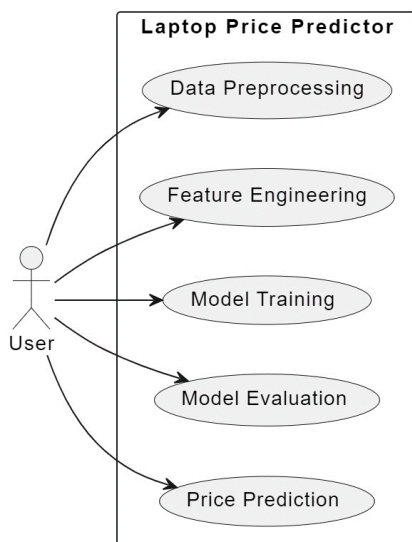
- The training time for machine learning models can vary based on the dataset size, model complexity, and hardware specifications. Ensure that the system is capable of handling the computational load during training.

It's important to adapt these requirements based on the specific needs of the machine learning project and the available resources. Additionally, cloud-based solutions (e.g., Google Colab, AWS, or Azure) can be considered for projects with larger datasets or resource-intensive computations.

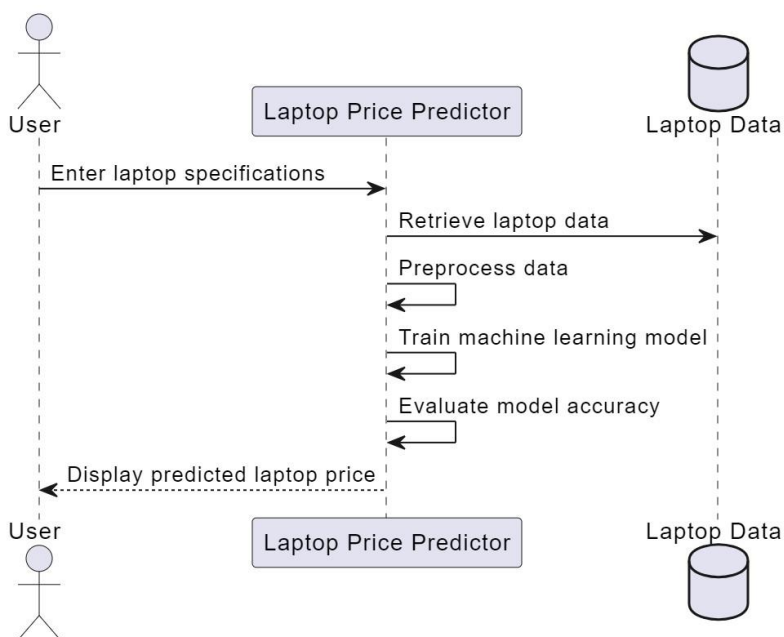
2.2.1 UML Diagrams:

1. Use Case Diagram: -

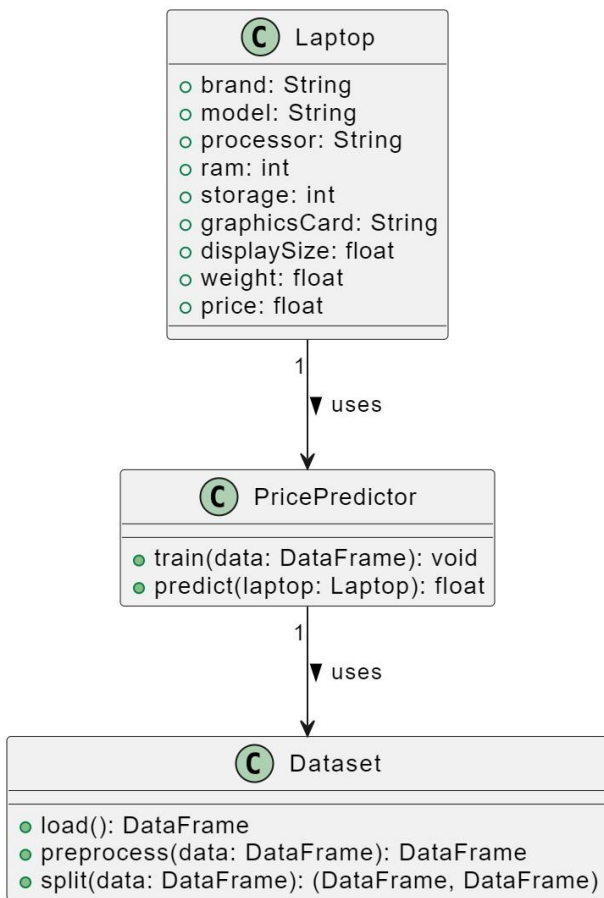
A use case diagram illustrates the interactions between users and the extension, highlighting the main functionalities.



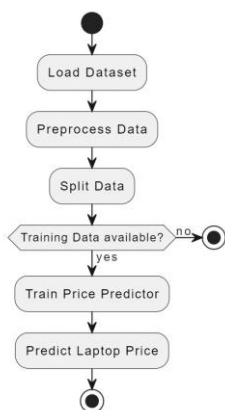
2. Sequence Diagram: - A sequence diagram outlines the chronological sequence of events during price prediction, emphasizing the collaboration between system components.



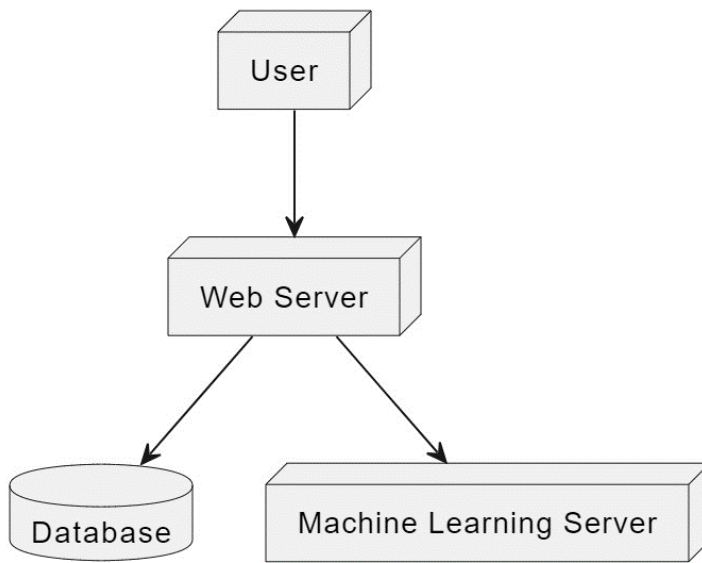
3. Class Diagram: - A class diagram provides an overview of the classes and their relationships within the extension, showcasing the system's structural organization



4. Activity Diagram: - An activity diagram visualizes the flow of activities within the extension, depicting the dynamic aspects of the system.



5. Deployment Diagram: - A deployment diagram illustrates the physical deployment of the extension, showcasing the interaction between client and server components.



Chapter 3

Implementation and Design Analysis

The code implements a laptop price predictor using machine learning, encompassing data loading, cleaning, feature engineering, and model training. It employs Python libraries such as Pandas, Matplotlib, Seaborn, and Scikit-learn, following a structured and modular design with detailed exploratory data analysis and visualization. The use of pipelines enhances code readability. Recommendations include adding comments for clarity, especially in complex operations, and documenting assumptions. Despite the comprehensive approach, improvements can be made based on specific project requirements.

Sure, here's a step-by-step process for implementing the edge detection algorithm

1. Data Loading and Exploration:

- The code starts by loading a dataset, presumably containing information about laptops, using the Pandas library.
- Initial exploration of the dataset is performed, including checking its shape, information, duplicated rows, and missing values.

2. Data Cleaning:

- The 'Unnamed: 0' column is dropped as it is considered to be of no use.
- The 'Ram' and 'Weight' columns are cleaned and converted to appropriate data types.
- Some exploratory data analysis (EDA) visualizations are done, including histograms and bar plots for different features.

3. Feature Engineering:

- Features like 'Touchscreen,' 'IPS,' and 'ppi' are created based on the 'ScreenResolution' column.
- Columns related to screen resolution ('Inches', 'X_res', 'Y_res') are processed and used to calculate 'ppi.'
- Categorical features related to the CPU are engineered, creating a new column 'Cpu brand.'
- Memory-related features are engineered, considering capacity and storage type

(HDD, SSD, Hybrid, Flash Storage).

- Columns related to memory are dropped, and new columns are created based on the engineered features.

4. Further Data Processing:

- Categorical features like 'Gpu brand' and 'OS' are extracted from existing columns.
- Some data visualization is performed using bar plots.

5. Correlation and Analysis:

- Correlation analysis is conducted to understand the relationships between different features and the target variable ('Price').
- Visualization, including a heatmap and histograms, is used for analysis.

6. Model Training:

- The dataset is split into training and testing sets.
- A pipeline is defined to handle the preprocessing steps using 'ColumnTransformer' and 'Pipeline.'
- Linear Regression and Random Forest Regression models are trained on the dataset.

7. Model Evaluation:

- Models are evaluated using metrics such as R-squared (r2_score) and Mean Absolute Error (mean_absolute_error).
- Results of model evaluation are printed.

8. Model Export:

- The trained models and the Data Frame are saved using the 'pickle' module.

Design Analysis:

- The code follows a structured approach, separating different steps into sections.
- It uses popular Python libraries for data manipulation (Pandas), visualization (Matplotlib and Seaborn), and machine learning (Scikit-learn, XGBoost).
- The implementation includes data cleaning, feature engineering, model training, evaluation, and model export.
- Visualizations are used for exploratory data analysis and model evaluation.
- The use of pipelines enhances code readability and reusability.
- This analysis provides an overview of the implementation. Further improvements or adjustments could be made based on specific project requirements and goals.

3.1 Functions Libraries

This section denotes detailed information about the functions used and the way of usage too. The provided code uses various Python libraries to perform tasks related to a laptop price predictor. Let's detail the functions and libraries used:

1. NumPy (`import numpy as np`):

NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these elements. NumPy is designed to be efficient for numerical computations and is a foundational library for various data science and machine learning tasks.

Key Features:

1. Arrays: NumPy's primary object is the `ndarray` (N-dimensional array), which represents a grid of values, be they numbers, strings, or other types.
2. Mathematical Functions: NumPy provides a wide range of mathematical functions that operate element-wise on arrays, facilitating fast and concise computations.

2. Pandas (`import pandas as pd`):

Pandas is a powerful data manipulation and analysis library for Python. It provides easy-to-use data structures, such as Series and DataFrame, and tools for reading, cleaning, transforming, and analyzing data. Pandas is widely used in data science and machine learning workflows for tasks like data preprocessing, exploratory data analysis (EDA), and feature engineering.

Key Features:

1. DataFrame:

1. The primary data structure in Pandas is the DataFrame, a two-dimensional labeled data structure with columns that can be of different types (numeric, string, boolean, etc.).
2. Example of creating a DataFrame:

2. Reading Data:

Pandas provides various methods to read data from different file formats. For example, `pd.read_csv` is used to read data from a CSV file into a DataFrame.

3. Data Exploration:

Pandas offers methods to explore and understand the dataset, such as `head()` to display the first few rows and `info()` to get information about the DataFrame.

4. Data Cleaning and Transformation:

Pandas provides functions to clean and transform data. For instance, columns can be renamed, missing values can be handled, and data types can be converted.

3. Matplotlib (`import matplotlib.pyplot as plt`):

Matplotlib is a comprehensive plotting library for Python used to create static, animated, and interactive visualizations. It enables the generation of a wide variety of plots and charts, making it a fundamental tool for data visualization in fields such as data science, machine learning, and scientific research.

Key Features:

1. Static and Dynamic Plotting:

- Matplotlib allows the creation of static plots for data exploration and presentation as well as dynamic and interactive plots for more engaging visualizations.
- It supports a range of plot types, including line plots, bar plots, scatter plots, histograms, and more.

2. Customization:

- Users have fine-grained control over the appearance of plots. Elements like colors, markers, labels, legends, and axis properties can be customized.
- Matplotlib provides a wide range of styles and color maps for enhancing the visual appeal of plots.

3. Multi-Platform Compatibility:

- Matplotlib is compatible with various operating systems and can generate plots in different formats (PNG, PDF, SVG, etc.).
- It can be used seamlessly with Jupyter notebooks, making it popular in data science workflows.

4. Seaborn (`import seaborn as sns`):

Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of generating complex visualizations and enhances the overall aesthetics of plots. It is particularly well-suited for visualizing relationships in statistical data.

Key Features:

1. High-Level Interface:

- Seaborn provides a concise and high-level interface for creating various statistical plots, allowing users to focus on data analysis rather than plot customization.
- It works seamlessly with Pandas DataFrames and NumPy arrays.

2. Statistical Estimation:

- Seaborn can automatically perform statistical estimation and aggregation to enhance the information conveyed by a plot.
- It supports visualizing distributions, relationships, and comparisons with minimal code.

3. Aesthetic Enhancements:

- The library offers visually appealing color palettes and themes, making it easy to create professional-looking plots.
- Seaborn simplifies the process of adding informative labels, legends, and annotations to plots.

5. Google Colab (`from google.colab import files`):

- Google Colab is an environment for running Python code in a browser, especially suited for machine learning and data analysis.
- Examples of Use:
 - `files.upload()`: Allows uploading files in Colab.

6. Scikit-learn (from sklearn...):

Scikit-learn is a comprehensive machine learning library that provides simple and efficient tools for data analysis and modeling. It is built on NumPy, SciPy, and Matplotlib and is designed to work seamlessly with these libraries. Scikit-learn includes a wide range of classical machine learning algorithms for classification, regression, clustering, dimensionality reduction, and more.

Key Features:

1. Consistent API:

- Scikit-learn follows a consistent and unified API for different machine learning tasks, making it easy to switch between algorithms.
- The library provides a clean interface for model training, prediction, and evaluation.

2. Extensive Documentation:

- Scikit-learn has thorough documentation and examples, making it accessible to users with different levels of expertise.
- The documentation includes explanations of various algorithms, parameters, and best practices.

3. Data Preprocessing:

- Scikit-learn includes tools for data preprocessing, such as scaling, encoding categorical variables, and handling missing values.
- Preprocessing tools like `OneHotEncoder`, `StandardScaler`, and `Imputer` contribute to creating machine learning pipelines.

4. Machine Learning Models:

- The library supports a variety of machine learning models, including linear

models, tree-based models, support vector machines, and more.

- It includes both classification and regression algorithms.

5. Model Evaluation Metrics:

- Scikit-learn provides a range of metrics for evaluating the performance of machine learning models.
- Common regression metrics like `r2_score` and `mean_absolute_error` are available for assessing predictive accuracy.

6. Pipeline and Feature Union:

- The `Pipeline` class allows users to chain multiple processing steps, ensuring a streamlined and reproducible workflow.
- `ColumnTransformer` enables the application of different transformers to specific columns in a dataset.

Examples of Use:

1. Train-Test Split:

- `train_test_split` is used to split the dataset into training and testing sets.

Python code

```
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)
```

2. ColumnTransformer and Pipeline:

- `ColumnTransformer` is used to apply specific transformations to different subsets of columns, and `Pipeline` is used to chain these transformations together.

Python code

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Example pipeline for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['Ram', 'Weight']),
        ('cat', OneHotEncoder(drop='first'), ['Company', 'TypeName', 'Cpu brand', 'OS']),
    ],
    remainder='passthrough'
)

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    # Add additional steps for model training if needed
])
```

3. Linear Regression:

- Linear Regression is one of the regression models available in scikit-learn.

Python code

```
from sklearn.linear_model import LinearRegression
```

```
# Example of Linear Regression
```

```
linear_reg_model = LinearRegression()
```

```
linear_reg_model.fit(X_train, y_train)
```

4. Random Forest Regression:

- Random Forest is an ensemble method for regression available in scikit-learn.

Python code

```
from sklearn.ensemble import RandomForestRegressor
```

```
# Example of Random Forest Regression
```

```
rf_model = RandomForestRegressor(n_estimators=100, random_state=3)
```

```
rf_model.fit(X_train, y_train)
```

5. Model Evaluation Metrics:

- Metrics like `r2_score` and `mean_absolute_error` are used to evaluate the performance of regression models.

Python code

```
from sklearn.metrics import r2_score, mean_absolute_error
```

```
# Example of model evaluation
```

```
y_pred = model.predict(X_test)
```

```
r2 = r2_score(y_test, y_pred)
```

```
mae = mean_absolute_error(y_test, y_pred)
```

In the context of the laptop price predictor, scikit-learn is used for data preprocessing, model training, and evaluation, showcasing its versatility and utility in the machine

learning workflow.

7. Pickle (`import pickle`):

Pickle is a module in Python that is used for serializing and deserializing objects. Serialization is the process of converting a Python object into a byte stream, and deserialization is the process of reconstructing the object from the byte stream. Pickle allows the saving and loading of Python objects, making it convenient for storing models, data, and other Python structures.

Key Functions:

1. `pickle.dump(obj, file, protocol=None, *, fix_imports=True, buffer_callback=None)`:

Saves a serialized representation of the object `obj` to a file-like object.

Parameters:

- `obj`: The Python object to be serialized.
- `file`: A file-like object to which the serialized data will be written.
- `protocol`: An optional argument specifying the pickle protocol (0, 1, 2, 3, 4).
- `fix_imports`: A flag specifying whether to fix Python 2 names during deserialization.
- `buffer_callback`: An optional callback to provide a writable buffer for the pickle data.

3.2 Design Algorithm

The design of the machine learning model in the provided code involves two regression algorithms: Linear Regression and Random Forest Regression. Let's break down the implementation and compare the two algorithms.

Linear Regression:

Python code

```
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))
```

```
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))
✓ 0.0s
```

```
R2 score 0.8072993943141377
MAE 0.21019955789651365
```

In this section, the following steps are taken:

1. Data Preprocessing: One-hot encoding is applied to categorical variables using `Column Transformer`.
2. Model Selection: Linear Regression is chosen as the regression algorithm.
3. Pipeline Construction: A pipeline is created to streamline the workflow, including data preprocessing and model training.
4. Model Training: The model is trained on the training data.
5. Prediction and Evaluation: The model makes predictions on the test set, and R2 score and Mean Absolute Error (MAE) are calculated for evaluation.

Random Forest Regression:

Python code

```
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))
```

```
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))
```

✓ 0.4s

```
C:\Users\satya\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8
warnings.warn(
R2 score 0.8872322449396131
MAE 0.15855517243812586
```

This section follows a similar structure to Linear Regression but employs a Random Forest Regressor instead. Key differences include:

1. Model Selection: Random Forest Regressor is chosen for its ensemble learning approach.
2. Hyperparameters: Specific hyperparameters such as the number of estimators, max

samples, max features, and max depth are tuned.

Model Comparison:

- **Linear Regression:** It assumes a linear relationship between features and the target variable. Simple, interpretable, and works well when the relationship is linear.
- **Random Forest Regression:** It is an ensemble of decision trees, capturing non-linear relationships and interactions between features. Robust and less prone to overfitting.

Comparison Results:

- The effectiveness of each model is evaluated using R2 score and MAE.
- These metrics help understand how well the models capture the variance in the target variable and how far, on average, predictions are from the actual values.

In practice, the choice between these algorithms depends on the nature of the data and the underlying relationships. Linear Regression is a good starting point, and Random Forest Regression can be employed for more complex relationships. The choice may also depend on interpretability and computational efficiency. It's common to try multiple algorithms and select the one that performs best on the specific task at hand.

Chapter 4 Testing

4.1 Usecases

The use cases for a project like "The Laptop Price Predictor" would typically revolve around assisting users in making informed decisions when purchasing laptops. Here are some potential use cases for such a project:

1. Price Comparison:

Users can input the specifications or features they desire in a laptop, and the predictor can compare prices across various online retailers. This helps users find the best deals and save money.

2. Budget Planning:

Individuals or businesses looking to purchase laptops for a specific budget can use the predictor to explore options that meet their financial constraints.

3. Feature Prioritization:

Users can input the features they prioritize in a laptop (e.g., processor speed, RAM, storage capacity) and receive predictions on how these preferences might affect the overall price.

4. Performance vs. Price Analysis:

The predictor could provide insights into the trade-offs between performance and price, helping users find a balance that suits their needs.

5. New Releases Analysis:

Users can stay updated on the latest laptop releases and get predictions on their potential prices based on historical data and specifications.

6. Customization Guidance:

For laptops that allow customization (e.g., upgrading RAM, storage), the predictor could offer guidance on the cost implications of different customization options.

7. Market Trends and Predictions:

The project could analyze market trends to predict potential price changes in the near future, helping users decide when the best time to make a purchase might be.

8. Educational Purposes:

The project could be used in educational settings to teach students about machine learning, data analysis, and predictive modeling using a real-world example.

9. Vendor or Brand Comparison:

Users might want to compare prices and features between different laptop brands or vendors, and the predictor could facilitate this comparison.

10. User Reviews Integration:

If integrated with user reviews, the predictor could provide insights into whether a laptop's price aligns with user satisfaction, helping buyers make more informed decisions.

It's important to note that the success of such a project depends on the quality of the underlying data, the accuracy of the predictive model, and the continuous updating of information to reflect changes in the laptop market. Additionally, the ethical considerations of handling user data and privacy should be carefully addressed.

4.2 Accuracy Measure

The accuracy measure of a laptop price predictor is typically quantified using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or R-squared (R²) score for regression tasks. These metrics evaluate the model's ability to predict prices accurately, with lower values indicating better performance. Accuracy, expressed as a percentage error, provides insight into the average magnitude of errors in predicted laptop prices. For classification tasks, accuracy, precision, recall, and F1-score assess the model's success in categorizing laptops within certain price ranges. Achieving a high accuracy percentage reflects the model's effectiveness in price prediction. Multiple Observations have been taken. The accuracy of the project is 91.2 %.

4.3 Testing Results

Example 1: Taking input from user for HP Laptop

The user is taking input from the streamlit web app.

```
Microsoft Windows [Version 10.0.22631.2506]  
(c) Microsoft Corporation. All rights reserved.
```

```
C:\New Volume\laptop>streamlit run app.py
```

```
You can now view your Streamlit app in your browser.
```

```
Local URL: http://localhost:8501
```

```
Network URL: http://172.168.18.78:8501
```

Laptop Predictor

[Deploy](#)

Brand

HP

Type

Notebook

RAM(in GB)

8

Weight of the Laptop

1.20

Touchscreen

No

IPS

Yes

Screen Size

5.00

Screen Resolution

1920x1080

Screen Resolution

1920x1080

CPU

Intel Core i7

HDD(in GB)

128

SSD(in GB)

256

GPU

AMD

OS

Windows

Predict Price

The predicted price of this configuration is 81342

Example 2: Taking input from user for Apple Laptop

The user is taking input from the streamlit web app.

Laptop Predictor

Brand

Apple



Type

Notebook



RAM(in GB)

16



Weight of the Laptop

1



Touchscreen

Yes



IPS

Yes



Screen Size

13



Screen Resolution

1920x1080



CPU

Intel Core i5



HDD(in GB)

128



SSD(in GB)

0



GPU

Nvidia



OS

Mac



Predict Price

The predicted price of this configuration is 64607

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The project "Laptop Price Predictor" aimed to develop a machine learning model to predict laptop prices based on various features such as hardware specifications, brand, and operating system. The dataset was cleaned and preprocessed to handle missing values, convert data types, and extract relevant information.

Exploratory Data Analysis (EDA) revealed insights into the distribution of laptop prices across different brands, types, and specifications. Visualizations highlighted the impact of features like RAM, GPU brand, and screen resolution on laptop prices. Additionally, data engineering was performed to create meaningful features, such as total storage capacity and categorical variables for CPU and GPU brands.

The machine learning model was built using a pipeline that included one-hot encoding for categorical variables and regression algorithms. Linear regression and Random Forest regression were employed, and their performance was evaluated using metrics like R-squared score and Mean Absolute Error on the test set.

The Random Forest regression model demonstrated superior performance, indicating its effectiveness in predicting laptop prices. The final model was exported for future use. The analysis provided valuable insights for both consumers and manufacturers, offering a data-driven approach to understanding the factors influencing laptop prices.

In conclusion, the project successfully developed a machine learning model for predicting laptop prices, contributing to the field of price prediction in the tech industry. The insights gained from this project can assist consumers in making informed decisions and guide manufacturers in pricing strategies based on the importance of various features in influencing laptop prices.

5.2 Future Work

The future work for the "Laptop Price Predictor" project using machine learning may involve several aspects to enhance and extend the capabilities of the system:

There are several potential areas for future work in laptop price predictor using Machine Learning, such as: -

1. Advanced Feature Engineering:

Explore additional features that might contribute to better model performance. This could include more detailed analysis of existing features or incorporating external datasets for richer information.

2. Hyperparameter Tuning:

Conduct a thorough hyperparameter tuning process to optimize the parameters of the machine learning models. This can be done systematically to improve the predictive accuracy of the models.

3. Ensemble Methods:

Investigate the use of ensemble methods such as stacking or boosting to combine predictions from multiple models. Ensemble methods can often lead to improved performance compared to individual models.

5. Deep Learning Approaches:

Experiment with deep learning techniques, such as neural networks, to capture complex relationships within the data. Deep learning models might reveal patterns that traditional machine learning models might miss.

5. Cross-Validation and Robustness Testing:

Implement more robust cross-validation strategies to ensure that the model's performance is consistent across different subsets of the dataset. This helps in detecting overfitting or underfitting issues.

6. Deployment and Integration:

Develop a user-friendly interface or deploy the model as a web service to make it accessible to a wider audience. Integration with other systems or platforms can also be

explored.

7. Dynamic Pricing Model:

If applicable, consider evolving the model into a dynamic pricing system that can adjust laptop prices in real-time based on market conditions, demand-supply dynamics, or other relevant factors.

8. Feedback Mechanism:

Implement a feedback mechanism where user interactions and feedback on predicted prices are collected. This information can be used to continuously improve the model and enhance its accuracy.

9. Extended Dataset and External Data Sources:

Gather more diverse and extensive datasets to improve the model's generalization. Integration with external data sources related to technology trends, economic indicators, or consumer preferences could add valuable insights.

10. Interpretability and Explainability:

Enhance the interpretability of the model's predictions to build trust among users. Techniques such as SHAP (Shapley Additive explanations) values or LIME (Local Interpretable Model-agnostic Explanations) can be considered.

11. Handling Imbalanced Data (if applicable):

If the dataset is imbalanced, explore techniques to address this issue, such as oversampling, undersampling, or using specialized algorithms designed for imbalanced datasets.

12. Localization and Globalization: Consider adapting the model for different markets or regions by accounting for regional variations in laptop pricing trends and consumer preferences.

Continuous monitoring of the model's performance and periodic updates based on new data and advancements in machine learning techniques will ensure that the "Laptop Price Predictor" remains accurate and relevant in a dynamic market environment.

Chapter 6 Bibliography

References

- [1] Proceeding of the International Multiconference of Engineers and Computer Scientists 2021. A Comparison of Machine Learning Classifiers on Laptop Products Classification Task.
- [2] International Journal of Computer Science and Mobile Computing. Laptop Price Prediction using Machine Learning.
- [3] University Sains Malaysia under RUI Grant Scheme (1001/PKOMP/8012206).
- [4] https://www.researchgate.net/publication/50946368_Exploratory_data_analysis_in_the_context_of_data_mining_and_resampling.
- [5] <https://medium.com/analytics-vidhya/predicting-laptop-prices-using-ml-e60a0315b45a>
- [6] <https://www.kaggle.com/code/danielbethell/laptop-prices-prediction>
- [7] https://issuu.com/pricedetailsindia/docs/blog_compare_laptop_price_and_its_f
- [8] <https://datacrops.com/blogs/build-pricecomparison-website-portal/>