Data Mining & Machine Learning

# Laptop Price Prediction Project Documentation

Martina Speciale

University of Pisa
Artificial Intelligence and Data Engineering

Academic Year 2023/2024

# Contents

# Chapter 1

# Introduction

In an era where laptops have become indispensable tools for both personal and professional use, understanding and predicting their prices can offer significant value to consumers, retailers, and manufacturers. This project aims to leverage the power of regression models to predict laptop prices based on various features.

In the next sections we'll focus on each step specifically: preprocessing, feature engineering, model development, evaluation, and deployment. By comparing the performance of different models, we aim at identifying the most effective technique for our prediction task.

To make our solution accessible and user-friendly, we will deploy the best-performing model in a simple web application[1]. This app will enable users to input various laptop features and receive a predicted price, thus providing a practical tool for potential buyers and sellers in the market.

## 1.1 Problem Definition

Our primary objective is to develop a predictive model that can accurately estimate the price of a laptop based on its features. Given the wide variety of laptop specifications and brands available in the market, predicting the price is a complex task that involves analyzing numerous factors.

To tackle this problem, we will perform extensive exploratory data analysis (EDA) and data preprocessing on the original dataset. EDA will help us understand the underlying patterns, distributions, and relationships between different features.

Data preprocessing will include feature engineering to create new relevant features and transform existing ones to better represent the underlying information. This step is crucial to enhance the model's performance and ensure that we extract the most out of the available data.

## 1.2 The Dataset

### 1.2.1 Description of Features in Original Dataset

- **`Company`**: The manufacturer or brand of the laptop.
- **`TypeName`**: The type or category of the laptop (e.g., Notebook, Gaming).
- **`Inches`**: The screen size of the laptop measured in inches.
- **`ScreenResolution`**: The resolution and type of the laptop screen (e.g., Full HD 1920x1080, IPS Panel).
- **`Cpu`**: The CPU model with specifications (e.g., Intel Core i7 8550U 1.8GHz).
- **`Ram`**: The amount of RAM in the laptop (e.g., 8GB, 16GB).
- **`Memory`**: The storage configuration, including type and capacity (e.g., 256GB SSD, 256GB SSD + 1TB HDD).
- **`Gpu`**: The GPU model and specifications (e.g., Nvidia GeForce MX150, Intel HD Graphics 620).
- **`OpSys`**: The operating system installed on the laptop (e.g., Windows 10, No OS).
- **`Weight`**: The weight of the laptop (e.g., 1.95kg, 2.62kg).
- **`Price`**: The target variable representing the price of the laptop in indian rupees (INR).

# Chapter 2

# Data Pre-Processing

## 2.1 Exploratory Data Analysis

During the Exploratory Data Analysis (EDA) phase, we examined the dataset to understand its main characteristics, identify patterns, detect anomalies, and test hypotheses. We used various visualization techniques provided by the *Seaborn* library to explore the features and their relationships with the target variable, `Price`. We specifically employed:

- **Countplots** : to explore the distribution of categorical features, we utilized Seaborn's `countplot`. The resulting plot displays the count of observations in each categorical bin using bars. Countplots are useful for identifying any imbalances or patterns within categorical variables. For example, a countplot of the `Company` feature can reveal which brands are more prevalent in our dataset.

- **Barplots** : To examine the relationship between individual features and the target variable `Price`, we used Seaborn's `barplot`. A barplot allows us to visualize the average *price* for different categories of a feature, highlighting how different values for a certain category influence laptop prices. By displaying the mean price for each category, we can identify trends and relationships that may be useful for prediction. We can look at some in 2.4.

- **Distplots** : To analyze the distribution of numerical features, we employed Seaborn's `distplot`. This plot combines a histogram with a kernel density estimate (KDE) to provide a comprehensive view of the data distribution. Distplots are particularly useful to identify skewness and to visualize the spread of numerical features. A distplot of our target feature, `Price`, reveals a right-skewed distribution (figure 2.5, (before)), that we will discuss and handle in subsection 2.1.1

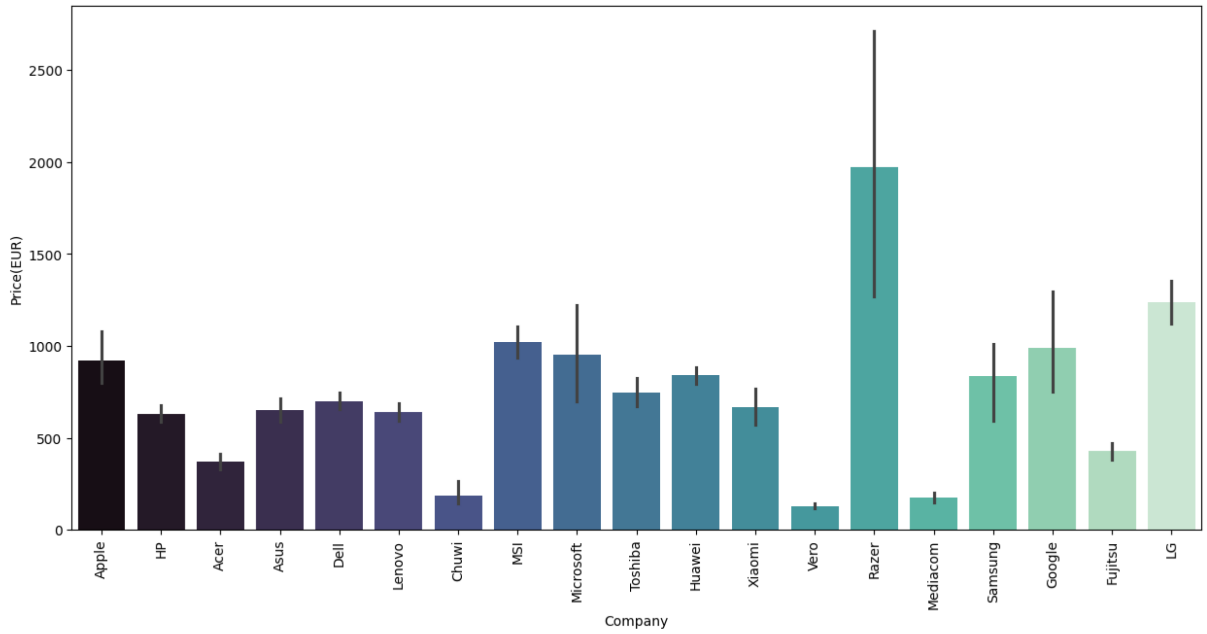


**Figure 2.1:** Company vs Price

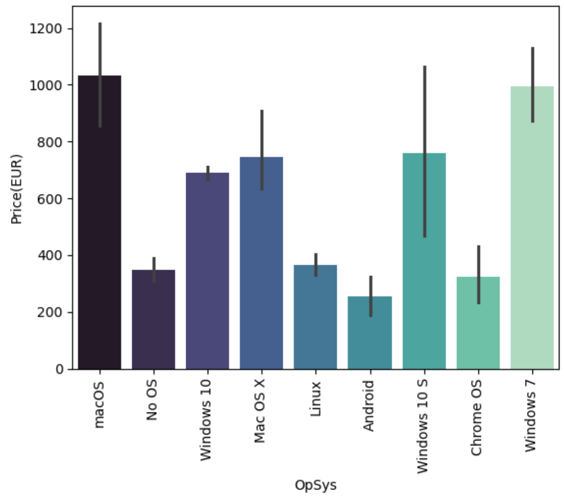


**Figure 2.2:** Operating Systems vs Price

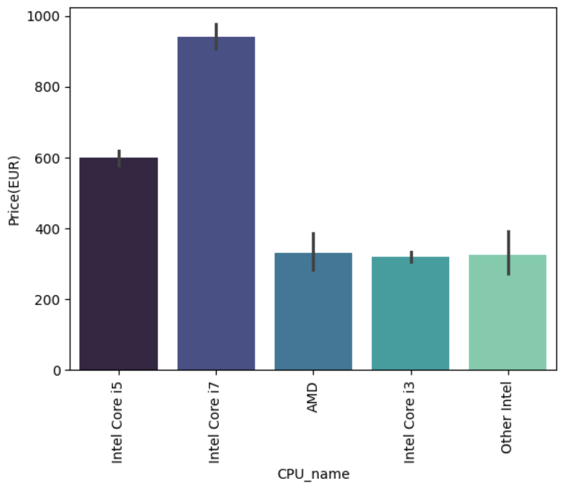


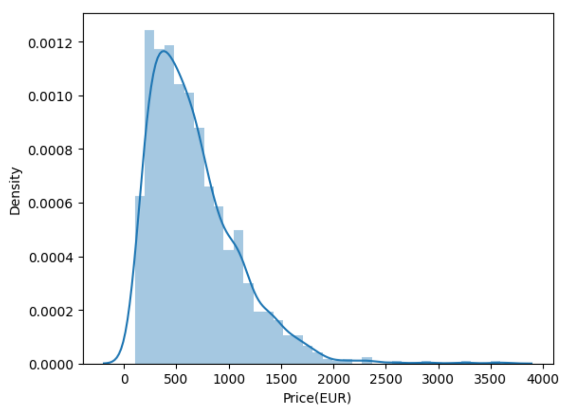
**Figure 2.3:** CPU vs Price

**Figure 2.4:** Barplots

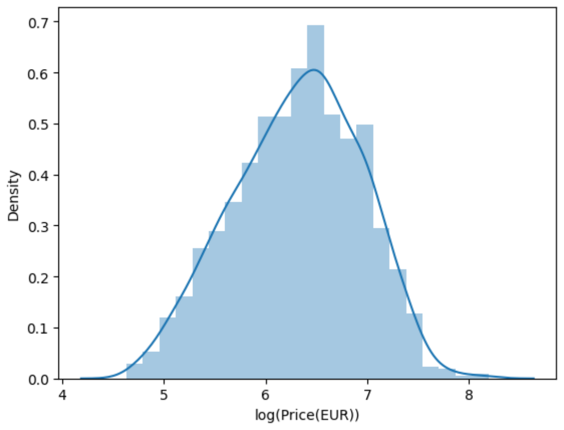
By leveraging Seaborn's visualization capabilities, we gained a deeper understanding of our dataset, which informed our data preprocessing and feature engineering steps, ultimately contributing to the development of more accurate and robust regression models.

### 2.1.1 Target Variable Transformation

Upon analyzing the distribution of the target variable, *price*, we observed that it is right-skewed. A right-skewed distribution indicates that there are a few extremely high values



(before)



(after)

**Figure 2.5:** Distplots that show the `Price` column distribution before and after the log transformation

that stretch the distribution's tail to the right, which can negatively impact the performance and accuracy of regression models. Regression models, especially linear models, often perform better when the target variable follows a normal (Gaussian) distribution.
To address this issue, we applied a logarithmic transformation to the *price* variable. The log transformation helps to normalize the distribution, reducing skewness and making it more Gaussian-like, as can be clearly observed in figure 2.5. The transformation is simply applied when defining our labels `y`, as follows:

```
y = np.log(df['Price(€)'])
```

By applying the log transformation, the distribution of the transformed variable, approximates a normal distribution.
This normalization is beneficial for the construction of regression models as it stabilizes variance and improves the linear relationship between the target variable and the predictors.

## 2.2 Preprocessing

During the preprocessing phase, we performed various transformations on the original features to extract more meaningful information, making the original dataset suitable for regression modeling. Key steps included:

- **Feature Engineering**: New features were created from existing ones to capture more information.

- **Encoding Categorical Variables**: Categorical variables were encoded numerically to be used in the regression model.
- **Unit Conversion**: Features such as `HDD`, `SSD` - created from the original `Memory` column - and `RAM` were converted to consistent units to ensure uniformity across the dataset.
- **Removing Irrelevant Features**: Features that were deemed not useful for the prediction, or were redundant, were removed from the dataset.
- **Removing Unique Instances**: Instances with unique or infrequent categories that did not provide meaningful insight for the model were removed.

### 2.2.1 Feature Engineering

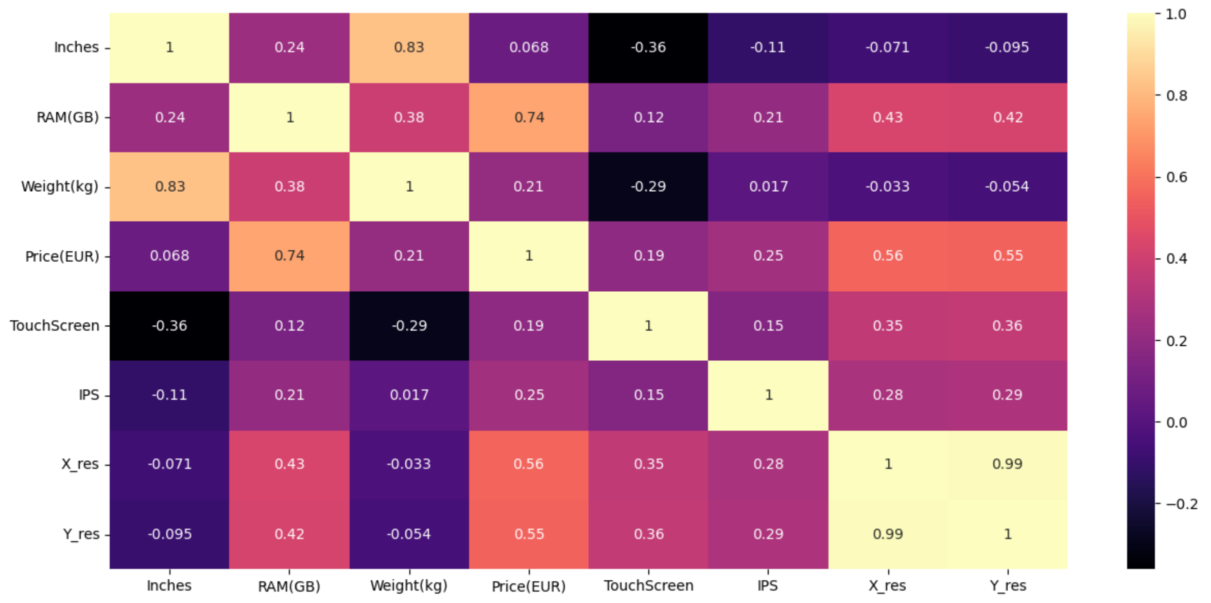


**Figure 2.6:** Heatmap

`PPI` (pixels per inch) was introduced as a measure of screen resolution density. This feature was engineered using the original `ScreenResolution` (e.g., Full HD 1920x1080) and `Inches` features. Specifically, from the original feature `ScreenResolution` we extracted two new features: `X_resolution` and `Y_resolution`. By examining the heatmap (figure 2.6) and the correlation coefficients with the target variable (`Price`), we observed that while both `X_resolution` and `Y_resolution` had a good correlation coefficient with `Price`, combining the two with another original feature, `Inches` (that shows a considerably lower correlation coefficient with `Price`), could provide a more significant insight. It is also clear how our two independent variables `X_resolution`

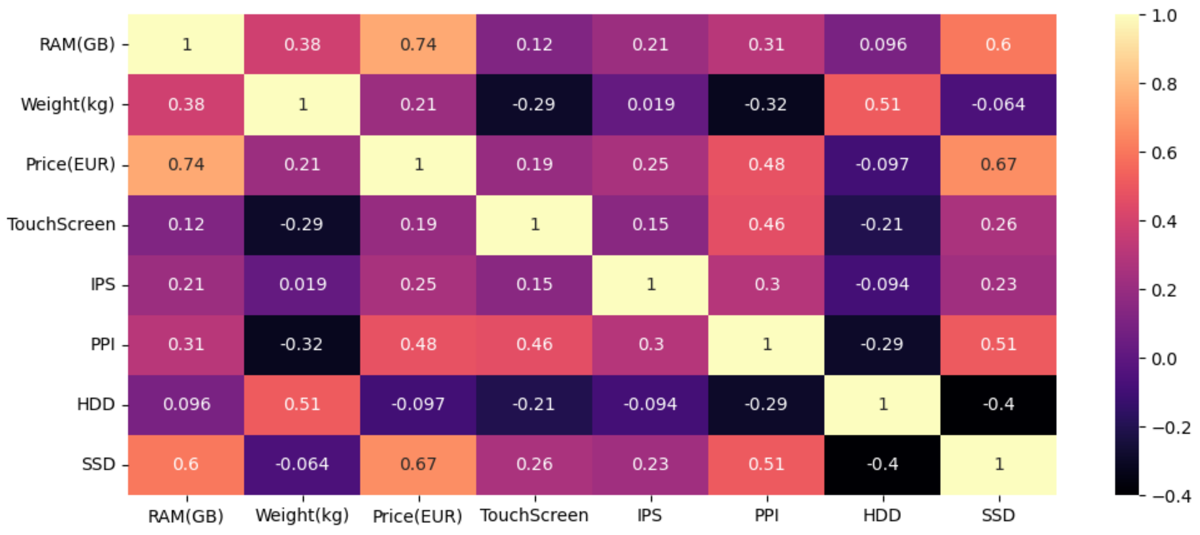


**Figure 2.7:** Heatmap over preprocessed data

and Y_resolution are highly correlated with each other (having a Pearson's correlation equal to 0.99). We say that they are *collinear*.
Hence, we calculated a new feature, PPI (Pixels Per Inch), which effectively combines X_resolution, Y_resolution, and Inches. The formula used to calculate PPI, considering $X_{res}$=X_resolution and $Y_{res}$=Y_resolution, is:

$$PPI = \frac{\sqrt{(X_{res}^2 + Y_{res}^2)}}{inches} \tag{2.1}$$

By evaluating the correlation of this new feature with the laptop prices, we found that the two showed a good correlation. As a result, we decided to include *PPI* in our model and remove the original features X_resolution, Y_resolution, and Inches, thereby simplifying our model while retaining the predictive power of the relevant information. In figure 2.7 we can see the resulting *heatmap* with the updated set of features.

### 2.2.2 Description of Features in Preprocessed Dataset

- **Company**: The manufacturer or brand of the laptop.
- **TypeName**: The type or category of the laptop (e.g., Notebook, Gaming).
- **RAM(GB)**: The amount of RAM in the laptop, standardized to gigabytes.
- **OpSys**: The operating system installed on the laptop. For simplification and consistency, the operating systems were grouped into broader categories as follows:
  - Windows: includes 'Windows 10', 'Windows 7', and 'Windows 7 S'.
  - MacOS: includes 'macOS' and 'mac OS X'.

  - `Linux`: Kept unchanged.
  - `Other`: includes all other operating systems not listed above.

- **`Weight(kg)`**: The weight of the laptop in kilograms.
- **`TouchScreen`**: Indicator of whether the laptop has a touchscreen (0 = No, 1 = Yes). It was obtained from the original `ScreenResolution` column.
- **`IPS`**: Indicator of whether the laptop has an IPS panel (0 = No, 1 = Yes). It was obtained from the original `ScreenResolution` feature.
- **`PPI`**: Pixels per inch, a measure of screen resolution density. Computed as

$$PPI = \frac{\sqrt{X_{res}^2 + Y_{res}^2}}{Inches}$$

  where $X_{res}$ and $Y_{res}$ where extracted from the original `ScreenResolution` feature.

- **`CPU`**: The CPU model and specifications. For simplification, the CPU models were grouped into broader categories as follows:
  - `Intel Core i7, Intel Core i5, Intel Core i3`: Kept unchanged.
  - `Other Intel Processor`: includes other Intel models not specifically listed above.
  - `AMD`: includes all remaining CPU models from AMD.
- **`HDD`**: The hard disk drive capacity in gigabytes. It was obtained from the original `Memory` column.
- **`SSD`**: The solid-state drive capacity in gigabytes. It was obtained from the original `Memory` column.
- **`GPU brand`**: The brand of the laptop's GPU.
  - `Intel`
  - `Nvidia`
  - `AMD`
- **`Price(€)`**: The target variable representing the price of the laptop, converted from indian rupees (₹) -as it was stored in the original `Price` column- to euros (€). We used the following *exchange rate* for the conversion:

$$1₹ = € \ 0.0110699$$

# Chapter 3

# Building the Models

We'll employ a variety of regression models, each with its unique strengths and characteristics. Linear, lasso, and ridge regression provide straightforward, interpretable models with varying degrees of complexity and regularization. Decision trees, KNN, and random forests offer more flexible, non-linear approaches capable of capturing complex relationships in the data. Through careful evaluation and comparison, we aim to identify the most effective model for predicting laptop prices.

## Model Pipelines

Each model was built as a pipeline using *scikit-learn*'s `Pipeline` class. The pipelines were constructed to include necessary preprocessing steps and the respective regression models.

### Structure of each Model Pipeline

Each pipeline (`pipe_model`) has been defined as follows:

```
from sklearn.pipeline import Pipeline

pipe_model = Pipeline([
    ('step1', column_transformer),  # always includes OneHotEncoder
    ('step2', model) # model is the specific regression model
])
```

In this pipeline:

- `step1` represents the preprocessing step, which includes a `ColumnTransformer` (`column_transformer`) to handle different preprocessing tasks such as `OneHotEncoder` (`ohe`) applied to categorical variables.
- `'step2'` represents the regression model adopted (`model`).

The `ColumnTransformer` instance (`column_transformer`) allows for specific transformations to be applied to different subsets of features, ensuring robust preprocessing before model fitting.

## 3.1 Metrics for Regression Models

Evaluating the performance of regression models involves several metrics that provide insights into the accuracy and reliability of the model's predictions. Here we have a brief description of the ones we used:

- **Root Mean Squared Error (RMSE)** The Root Mean Squared Error (RMSE) measures the average magnitude of the errors between predicted and actual values. It is defined as the square root of the average of the squared differences between the predicted and actual values. The formula for RMSE is:

  $$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

  where $y_i$ represents the actual values, $\hat{y}_i$ represents the predicted values, and $n$ is the number of observations. RMSE is sensitive to outliers and provides a measure of how well the model predicts the target variable. We will use to define our *Prediction Intervals* (see 5.1.3)

- **Mean Absolute Error (MAE)** The Mean Absolute Error (MAE) measures the average magnitude of the errors in a set of predictions, without considering their direction. It is the average over the absolute differences between the predicted and actual values. The formula for MAE is:

  $$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  where $y_i$ represents the actual values, $\hat{y}_i$ represents the predicted values, and $n$ is the number of observations. MAE is less sensitive to outliers compared to RMSE and provides an intuitive measure of the average prediction error.

- **R-Squared Score (R$^2$ Score)** : The R-Squared Score (R$^2$ Score) measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It is also known as the coefficient of determination. The formula for R$^2$ Score is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

  where $y_i$ represents the actual values, $\hat{y}_i$ represents the predicted values, $\bar{y}$ is the mean of the actual values, and $n$ is the number of observations. The R$^2$ Score ranges from 0 to 1, with higher values indicating a better fit of the model to the data. R$^2$ Score is the one over which we'll compare results (the ones we obtained versus the ones provided by the reference article)

These metrics provide different perspectives on the performance of regression models and are often used together to get a comprehensive understanding of the model's accuracy and effectiveness.

## 3.2 Regression Models

We now take a look at the models we developed for our regression task. We'll dive into the theoretical foundations under each one of them. For a reference to the code, the notebook in which they were developed is available here[2]

### 3.2.1 Linear Regression

- Notebook reference

The goal of linear regression is to establish a linear relationship between a dependent variable $y$ - `Price` in our case - and one or more independent variables $X$. The model can be represented as:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$

where:

- $\beta_0$ is the intercept,
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the independent variables,
- $\epsilon$ is the error term.

The coefficients are estimated using the least squares method, which minimizes the sum of the squared differences between the observed and predicted values.
We can look at the results obtained in table 3.1.

| **Metric** | **Value** |
|---|---|
| $R^2$ Score | 0.8067729390471987 |
| Mean Absolute Error (MAE) | 0.21011396168486424 |
| Root Mean Squared Error (RMSE) | 0.07391922852834454 |

**Table 3.1:** Performance Metrics for Linear Regression Model

### 3.2.2 Lasso Regression

- Notebook reference

It stands for Least Absolute Shrinkage and Selection Operator. It includes a regularization term in the objective function. This regularization term is the L1 norm of the coefficients, which encourages sparsity in the model, potentially leading to feature selection. The objective function for lasso regression is:

$$\min_{\beta} \left( \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j X_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right)$$

where $\lambda$ is the regularization parameter that controls the amount of shrinkage applied to the coefficients.
We can look at the results obtained in table 3.2.

| **Metric** | **Value** |
|---|---|
| $R^2$ score | 0.8071016911884541 |
| MAE | 0.21156263228051508 |
| RMSE | 0.07379346402859581 |

**Table 3.2:** Lasso Regression Metrics

### 3.2.3 Ridge Regression

- Notebook reference

**Ridge Regression** is another form of linear regression that includes a regularization term, but instead of the L1 norm, it uses the L2 norm of the coefficients. This helps to prevent overfitting by shrinking the coefficients towards zero, though it does not set any coefficients exactly to zero. The objective function for ridge regression is:

$$\min_{\beta}\left(\sum_{i=1}^{n}(y_i-\beta_0-\sum_{j=1}^{p}\beta_jX_{ij})^2+\lambda\sum_{j=1}^{p}\beta_j^2\right)$$

We can look at the results obtained in table 3.3.

| **Metric** | **Value** |
|---|---|
| $R^2$ score | 0.8127240627874481 |
| MAE | 0.2095188204743929 |
| RMSE | 0.07164261947789995 |

**Table 3.3:** Ridge Regression Metrics

### 3.2.4 Decision Tree

- Notebook reference

**Decision Trees** are a non-linear regression method that models the data by splitting it into subsets based on the value of the input features. Each internal node represents a decision rule on a feature, each branch represents the outcome of the decision, and each leaf node represents a predicted value. The tree is built by recursively splitting the data until a stopping criterion is met, such as a minimum number of samples in a leaf or a maximum tree depth.
We can look at the results obtained in table 3.4.

| **Metric** | **Value** |
|---|---|
| $R^2$ score | 0.8408519835422144 |
| MAE | 0.1816587155122773 |
| RMSE | 0.06088225189767472 |

**Table 3.4:** Decision Tree Regressor Metrics

### 3.2.5 Random Forests

- Notebook reference

**Random Forests** are an ensemble learning method that combines multiple decision trees to improve the model's performance and reduce overfitting. Each tree is trained on a random subset of the data and a random subset of the features. The final prediction is obtained by averaging the predictions of all individual trees. This approach leverages the wisdom of crowds to produce more accurate and robust predictions.
We can look at the results obtained in table 3.5.

| Metric | Value |
|---|---|
| $R^2$ score | 0.8819807610185106 |
| MAE | 0.16267052308685204 |
| RMSE | 0.045148392021265395 |

**Table 3.5:** Random Forest Regressor (Base) Metrics

## Hyperparameter Tuning

In the process of optimizing our regression models, we employed two common hyperparameter tuning methods: *GridSearchCV* and *RandomizedSearchCV*. Both methods aim to identify the best combination of hyperparameters that maximizes the model's performance. However, we observed that *RandomizedSearchCV* produced worse results compared to *GridSearchCV* (over the chosen ranges for the parameters we considered for various different runs).

- **GridSearchCV** : `GridSearchCV` performs an exhaustive search over a specified parameter grid. It evaluates all possible combinations of the provided hyperparameter values, which ensures that the best possible combination within the defined grid is found. This method is thorough and guarantees that if the optimal hyperparameters are within the specified grid, they will be identified. For this very reason it can be computationally expensive and time-consuming, especially when dealing with large datasets and a high number of hyperparameters. Fortunately that wasn't our case. It provided both configuration below (KNN (3.2.6) and Random Forest (3.3))

- **RandomizedSearchCV** : `RandomizedSearchCV`, on the other hand, performs a random search over a specified parameter distribution. It samples a fixed number of hyperparameter combinations from the distribution and evaluates them. This method is generally faster and more efficient. However, since it does not exhaustively search all possible combinations, it may miss the optimal set of hyperparameters, especially if the sampled combinations do not cover the best region of the hyperparameter space.

In our case, the thorough search provided by *GridSearchCV* resulted in better model performance, highlighting the importance of method selection and grid design in hyperparameter tuning.

### 3.2.6 K-Nearest Neighbors Regression with Hyperparameter Tuning

- Notebook reference

**K-Nearest Neighbors (KNN) Regression** is a non-parametric method that predicts the value of a new data point based on the values of its $k$-nearest neighbors in the training data. The prediction is typically the average of the values of the nearest neighbors. The algorithm can be summarized as:

1. Compute the distance between the new data point and all points in the training set.
2. Identify the $k$-nearest neighbors.
3. Predict the value as the average of the $k$-nearest neighbors' values.

We can look at the results obtained in table 3.6. The table above shows the performance

| **Metric** | **Value** |
|---|---|
| Best parameters found | { 'algorithm': 'auto', 'n_neighbors': 5, 'weights': 'distance' } |
| $R^2$ score | 0.8310506543164828 |
| MAE | 0.17873976296376842 |
| RMSE | 0.0646317613677554 |

**Table 3.6:** KNN with Hyperparameter Tuning Metrics

metrics for the KNN model after hyperparameter tuning using GridSearchCV. The parameters listed under "Best parameters found" represent the optimal configuration identified through the hyperparameter tuning process.

## 3.3 Best Model Found : Random Forest with Hyperparameter Tuning

- Notebook reference

We can look at the results obtained in table 3.7.

| **Metric** | **Value** |
|---|---|
| $R^2$ score | 0.9025200371480295 |
| MAE | 0.14795834217732115 |
| RMSE | 0.037291068939611066 |

**Table 3.7:** Random Forest with Hyperparameter Tuning Metrics

# Chapter 4

# Results

## Model Performance Comparison

The bar charts in figure 4.1 compare the performance of the various regression models we developed based on three key metrics: R$^2$ score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

- **R$^2$ Score**

  The R$^2$ score measures the proportion of variance in the dependent variable that is predictable from the independent variables. A higher R$^2$ score indicates a better fit of the model to the data.

  - **Random Forest (Tuned)** achieved the highest R$^2$ score, indicating the best overall fit.
  - **Random Forest (Base)** and **Decision Tree** also performed well, showing strong predictive capabilities.
  - Linear models such as **Ridge Regression**, **Lasso Regression**, and **Linear Regression** had lower R$^2$ scores compared to the ensemble and tree-based models.

- **Mean Absolute Error (MAE)**

  MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. Lower MAE values indicate better predictive accuracy.

  - **Random Forest (Tuned)** had the lowest MAE, demonstrating the most accurate predictions on average.
  - **Random Forest (Base)** and **Decision Tree** also had relatively low MAE values, reflecting good performance.

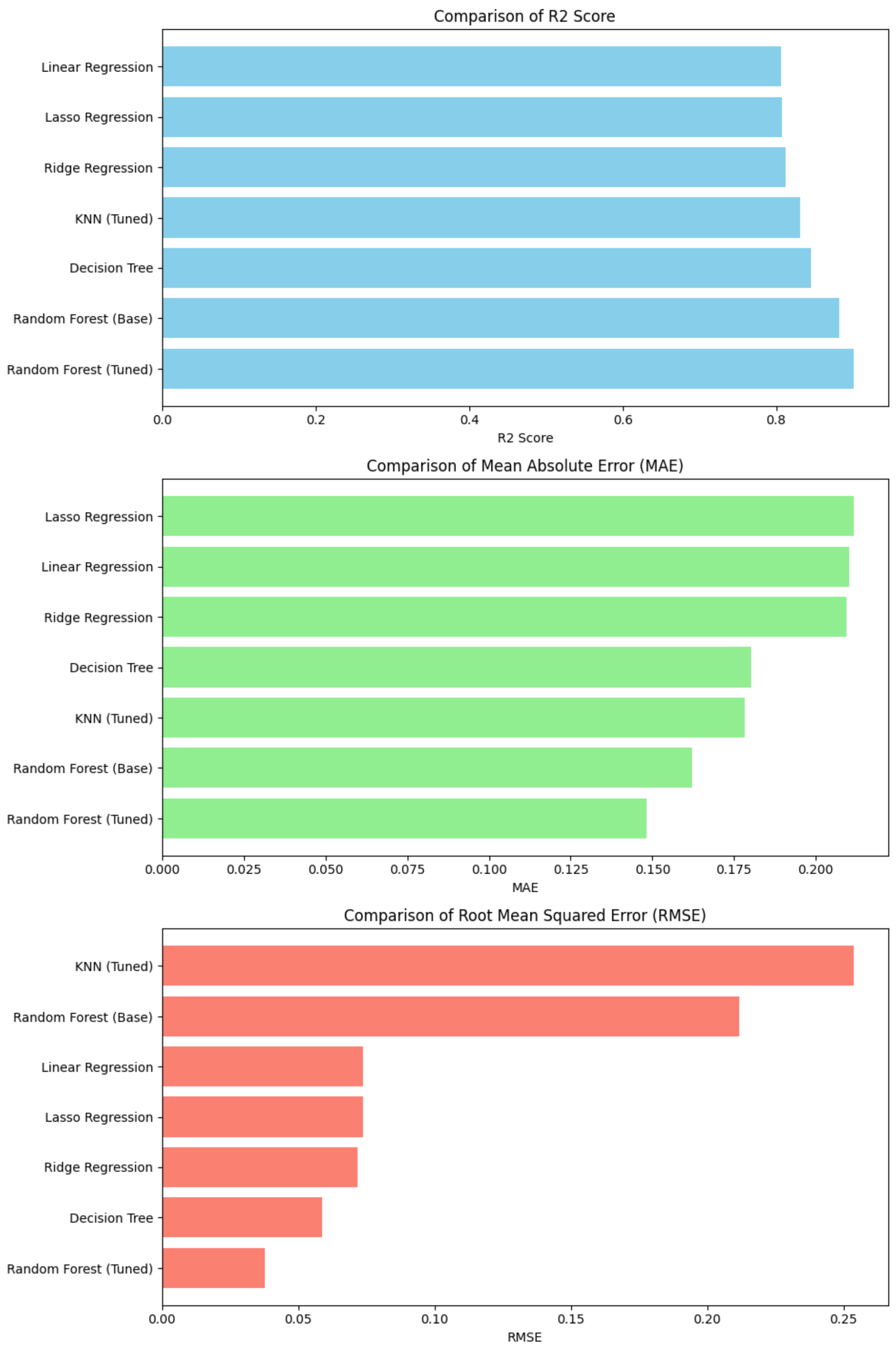


**Figure 4.1:** Comparative Performance of Different Regression Models

- The linear models, while still competitive, had higher MAE values, indicating less accurate predictions on average compared to the tree-based models.

- **Root Mean Squared Error (RMSE)**

RMSE measures the square root of the average squared differences between predicted and actual values. It is sensitive to larger errors, making it a more punitive measure compared to MAE. Lower RMSE values indicate better model performance.

- **Random Forest (Tuned)** again demonstrated superior performance with the lowest RMSE, indicating the least prediction error.
- **Decision Tree** also showed low RMSE, performing well in this metric.
- **KNN (Tuned)** had the highest RMSE, indicating greater prediction errors compared to other models.

Overall, the **Random Forest (Tuned)** model consistently outperformed the other models across all three metrics ($R^2$, MAE, and RMSE), making it the most robust and accurate model in this comparison. The **Decision Tree** and **Random Forest (Base)** models also showed strong performance, particularly in the $R^2$ and RMSE metrics. Linear models, while useful, did not perform as well as the ensemble and tree-based methods in this particular analysis.

## 4.1 Accuracy

### Model Accuracy Visualization

The plots in figure 4.6 illustrate the accuracy of our regression models by comparing the actual values with the predicted values over a series of samples (*test set*). A close alignment between the blue (actual) and red (predicted) lines indicates high accuracy of the model's predictions. Significant deviations between these lines suggest areas where the model's predictions do not match the actual values, highlighting potential areas for model improvement.
This visual representation helps us assessing the model's predictive accuracy.

## 4.2 Comparative Study

The project originated from the idea presented in the scientific paper "Laptop Price Prediction using Machine Learn- ing Algorithms". In: *2022 International Conference on Emerging Trends in Engineering and Medical Science* ([Sha+22]). Our development process was inspired by the concepts and methodologies outlined in this paper. As we progress, a crucial step involves comparing the results we have obtained with those documented in the paper. This comparative analysis will provide valuable insights into the efficacy of our approach. As already said, we evaluated our models using three key metrics: $R^2$ score,

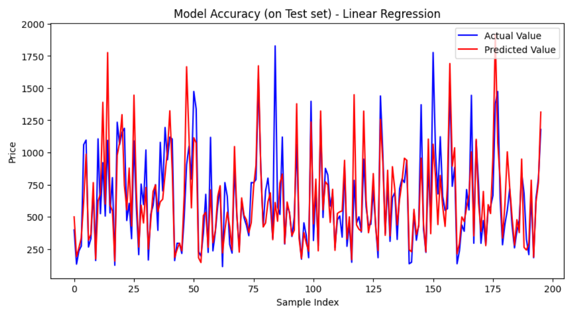


**Figure 4.2:** Linear Regression

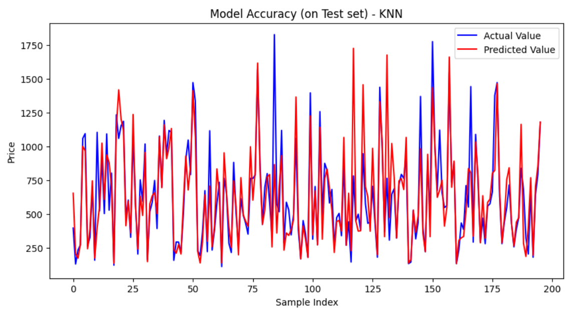


**Figure 4.3:** K-Nearest Neighbors

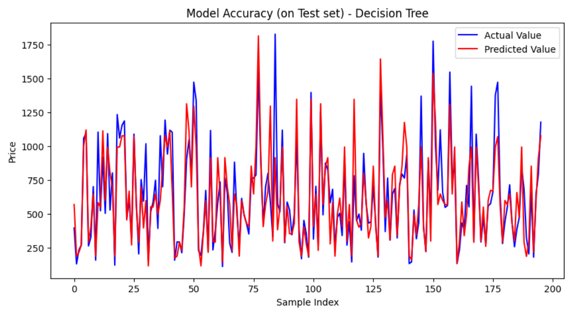


**Figure 4.4:** Decision Tree

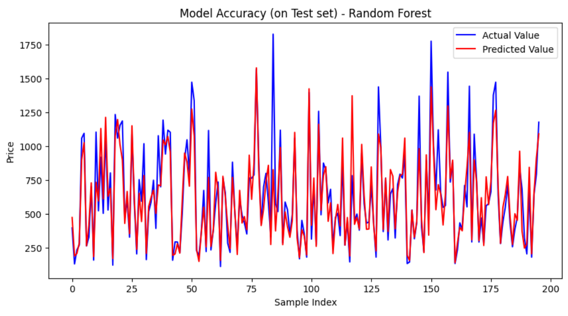


**Figure 4.5:** Random Forest

**Figure 4.6:** Models' Accuracy Visualization

Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). The paper's performance metrics are presented in terms of "Accuracy" ($R^2$) and Mean Absolute Error (MAE).

| Model | R2 score | MAE | RMSE |
|---|---|---|---|
| **Random Forest (with Hyperparameter Tuning)** | **0.9011** | **0.1482** | **0.0378** |
| KNN (with Hyperparameter Tuning) | 0.8316 | 0.1783 | 0.2538 |
| Linear Regression | 0.8068 | 0.2101 | 0.0739 |
| Ridge Regression | 0.8127 | 0.2095 | 0.0716 |
| Lasso Regression | 0.8071 | 0.2116 | 0.0738 |
| Decision Tree Regressor | 0.846 | 0.1802 | 0.0589 |
| Random Forest Regressor (Base) | 0.8827 | 0.1622 | 0.2119 |

**Table 4.1:** Performance of our regression models

| **Model** | **Accuracy** | **Mean Absolute Error** |
|---|---|---|
| Linear Regression | 0.8073 | 0.2102 |
| **Random Forest** | **0.8875** | **0.1588** |
| Decision Tree | 0.8469 | 0.1822 |
| KNN | 0.8022 | 0.1932 |

**Table 4.2:** Performance from scientific paper

Comparing the results from our models with those from the scientific paper, we observe that our Random Forest model with hyperparameter tuning achieved the highest $R^2$ score of 0.9011 and the lowest MAE of 0.1482, outperforming the paper's Random Forest model, which had an accuracy of 0.8875 and an MAE of 0.1588. Our KNN model with hyperparameter tuning also showed competitive performance with an $R^2$ score of 0.8316 and an MAE of 0.1783, slightly better than the paper's KNN model with an accuracy of 0.8022 and an MAE of 0.1932.

Linear Regression, Ridge Regression, and Lasso Regression models in our study achieved similar $R^2$ scores and MAE values, around 0.807 and 0.21 respectively, which are comparable to the paper's Linear Regression model with an accuracy of 0.8073 and an MAE of 0.2102. Our Decision Tree Regressor also demonstrated strong performance with an $R^2$ score of 0.846 and an MAE of 0.1802, aligning closely with the paper's Decision Tree model.

Overall, the Random Forest model with hyperparameter tuning stands out as the best-performing model in our study, highlighting the importance of hyperparameter optimization in improving model accuracy and reducing prediction errors.

# Chapter 5

# Deployment on Streamlit Cloud

The application is deployed on Streamlit Cloud, allowing easy access. The deployment provides a user-friendly interface for predicting laptop prices exploiting the models we developed. Models are stored in pickle files and loaded dynamically based on user selection.

## 5.1 User Guide

### 5.1.1 Accessing the Application

The deployed application can be accessed using this link[1]

### 5.1.2 Using the Application

1. **Home Page**: By accessing the provided link, the home page of the application will be displayed

2. **Other Pages** : Two additional pages were included in the application just to simplify accessibility to the dataset and the results obtained. They are accessible from the sidebar

   - **Data**
   - **Results**

3. **Input Features**: In the *Home Page* enter details about the laptop you want to predict the price of (brand, screen size, CPU type, RAM, storage etc.)

4. **Prediction Output**: Click on the *"Predict Price"* button after entering the laptop specifications to display the predicted price and the predicted interval (5.1.3).

5. **Selecting Models**: By default, the application uses the best-performing model (the one associated with the best performance scores we found). The other models can be selected from a dropdown menu for prediction by disabling the *default model.*

### 5.1.3 Prediction Output

#### Prediction Intervals

In regression analysis, prediction intervals provide a range within which future observations are expected to fall, with a certain level of confidence. A commonly used approach to calculate prediction intervals involves the Root Mean Square Error (RMSE). Specifically, the prediction interval for a predicted value $\hat{y}$ can be computed by subtracting and adding two times the RMSE from the predicted value. Mathematically, this is represented as $\hat{y} \pm 2 \cdot \text{RMSE}$. This method assumes that the residuals (the differences between the observed and predicted values) are normally distributed and that the RMSE provides a reasonable estimate of the standard deviation of the residuals. The factor of two corresponds to approximately a 95% prediction interval for normally distributed residuals, meaning that we can expect about 95% of the actual future values to lie within this interval around the predicted value. This approach provides a straightforward and interpretable way to quantify the uncertainty associated with regression predictions.

Since we are working with a dataset that has been transformed to a log scale to achieve a Gaussian distribution, both the predicted values and the prediction intervals need to be adjusted back to the original scale for proper interpretation.

The Root Mean Square Error (RMSE) is first calculated on the log-transformed data. To construct the prediction interval for a predicted value $\hat{y}$ (in the log scale), one would traditionally compute $\hat{y} \pm 2 \cdot \text{RMSE}$. However, to convert this interval back to the original scale, the exponential function is applied. Therefore, the prediction interval is $e^{\hat{y} \pm 2 \cdot RMSE}$.

Similarly, the predicted value itself, $\hat{y}$, must also be transformed back to the original scale using the exponential function, resulting in $e^{\hat{y}}$. Consequently, the prediction interval on the original scale is $[e^{\hat{y} - 2 \cdot RMSE}; e^{\hat{y} + 2 \cdot RMSE}]$.

This approach ensures that both the predicted values and the intervals are expressed in the same scale as the original data, providing a meaningful range for the actual future values. It is important to note that the assumption of normally distributed residuals in the log scale must hold for this method to provide valid intervals. By converting the RMSE and predicted values back to the original scale, we maintain the interpretability and relevance of the prediction intervals in the context of the original dataset.

## 5.2 Future Improvements

Future updates may include integration with more recent data (since the dataset we are using to be able to compare results[2] is not up do date), additional features for prediction and optimization of model performance.

# Bibliography

[1] *Streamlit Application - Laptop Price Predictor*. `https://laptoppricepredictor-unipi.streamlit.app/`.

[2] *Google Colab Notebook - Laptop Price Predictor*. `https://drive.google.com/file/d/16NgJB3pe72YEYBxCOfKbB9KIV8pKWH_2/view?usp=sharing`.

[Sha+22] Mohammed Ali Shaik et al. "Laptop Price Prediction using Machine Learning Algorithms". In: *2022 International Conference on Emerging Trends in Engineering and Medical Sciences (ICETEMS)*. 2022, pp. 226–231. DOI: `10.1109/ICETEMS56252.2022.10093357`.