**TOROS UNIVERSITY**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF COMPUTER AND SOFTWARE ENGINEERING**

**PREDICTING CAR PRICES IN TURKEY:**

**A MACHINE LEARNING APPROACH**

**BUĞRA CAN GÜNDOĞAN**

**SUPERVISOR**

**Asst. Prof. Dr. Furkan GÖZÜKARA**

**GRADUATION PROJECT**

**February 2023**

PREDICTING CAR PRICES IN TURKEY:A MACHINE LEARNING APPROACH

ABSTRACT

Machine learning is a highly exciting field that holds tremendous potential for improving the way we make decisions and perform analysis. With its ability to learn from data and identify patterns, machine learning has the power to transform traditional methods of decision making and bring about a more data-driven approach. This allows for better judgement and analysis, as the algorithms can analyze large amounts of data much faster and more accurately than a human ever could. The ability to use machine learning to make informed decisions and uncover new insights has far-reaching implications across a range of industries, from healthcare and finance, to marketing and manufacturing. The potential for machine learning to revolutionize the way we make decisions and analyze data makes it one of the most exciting fields of study today.

In this project the concepts and tools machine learning field provides are used in order to predict prices of second-hand cars, and to uncover correlations between prices of cars and their features.

**Keywords:**Machine Learning, Artificial Intelligence, Car Prices, Regression

# TABLE OF CONTENTS

[1. INTRODUCTION 10](#_Toc126113661)

[1.1 Tools Used in the Project 10](#_Toc126113662)

[1.2 Project Development Timeline 11](#_Toc126113663)

[1.3 System Model 12](#_Toc126113664)

[2. KEY THEORETICAL CONCEPTS 13](#_Toc126113666)

[2.1 Artificial Intelligence and Machine Learning 13](#_Toc126113667)

[2.2 Supervised and Unsupervised Learning 13](#_Toc126113668)

[2.3 Regression Problems and Models 14](#_Toc126113669)

[2.4 Categorical and Numerical Values 14](#_Toc126113670)

[2.5 Fast Tree Regression Algorithm 14](#_Toc126113671)

[2.5.1 Decision Trees 15](#_Toc126113672)

[2.5.2 Gradient Boosting 16](#_Toc126113673)

[2.5.3 Histogram Binning 17](#_Toc126113674)

[3. PROJECT DESIGN 18](#_Toc126113675)

[3.1 An Overview of the Folder Structure and Project Files 18](#_Toc126113676)

[3.2 Design of the Database 19](#_Toc126113677)

[3.3 User Interface 23](#_Toc126113678)

[3.4 Design of Classes 28](#_Toc126113679)

[3.5 Interactions Between Classes 35](#_Toc126113680)

[4. CODE IMPLEMENTATION DETAILS 40](#_Toc126113681)

[4.1 Gathering Data for the Research 40](#_Toc126113682)

[4.2 Processing the Data 47](#_Toc126113683)

[4.2.1 Feature Normalization 48](#_Toc126113684)

[4.2.1.1 Categorical One-Hot Encoding: 49](#_Toc126113685)

[4.2.1.2 Replace Missing Values 50](#_Toc126113686)

[4.2.1.3 Concatenate 50](#_Toc126113687)

[4.3 Creating the Model 51](#_Toc126113688)

[4.4 Training the Model 62](#_Toc126113689)

[4.5 Finding the Optimal Values For Fast Tree Algorithm 62](#_Toc126113690)

[4.6 Evaluating Features 67](#_Toc126113691)

[5. OUTCOMES AND IMPLICATIONS 71](#_Toc126113692)

[5.1 Top 10 and Last 10 Brands 71](#_Toc126113693)

[5.2 Top 10 and Last 10 Models 72](#_Toc126113694)

[5.3 Impact of Average Fuel Consumption on the Price 73](#_Toc126113695)

[5.4 Impact of Car Case on the Price 73](#_Toc126113696)

[5.5 Impact of Color on the Price 74](#_Toc126113697)

[5.6 Impact of Gear Type on the Price 74](#_Toc126113698)

[5.7 Impact of City on the Price 75](#_Toc126113699)

[5.8 Impact of Fuel Type on the Price 76](#_Toc126113700)

[6. CONCLUSION 77](#_Toc126113701)

# LIST OF TABLES

[Table 1 Tools Used in the Project 11](#_Toc126111251)

[Table 2 Project Development Timeline 11](#_Toc126111252)

[Table 3 Example Feature Table 49](#_Toc126111253)

[Table 4 One Hot Encoding Example 49](#_Toc126111254)

[Table 5 Model Comparisons 65](#_Toc126111255)

LIST OF FIGURES

[Figure 1. System Model 12](#_Toc126111122)

[Figure 2 Folder Structure 18](#_Toc126111123)

[Figure 3 \_\_EFMigrationsHistory Table Structure 19](#_Toc126111124)

[Figure 4 \_\_EFMigrationsHistory Table Data Sample 19](#_Toc126111125)

[Figure 5 Cars Table Structure 20](#_Toc126111126)

[Figure 6 Cars Table Data Sample 20](#_Toc126111127)

[Figure 7 CarsTest Table Structure 21](#_Toc126111128)

[Figure 8 CarsTest Table Data Sample 21](#_Toc126111129)

[Figure 9 FeatureTests Table Structure 22](#_Toc126111130)

[Figure 10 FeatureTests Table Data Sample 22](#_Toc126111131)

[Figure 11 Users Table Structure 23](#_Toc126111132)

[Figure 12 Users Table Data Sample 23](#_Toc126111133)

[Figure 13 Home Page - 1 23](#_Toc126111134)

[Figure 14 Home Page – 2 24](#_Toc126111135)

[Figure 15 Predict Price Page – 1 24](#_Toc126111136)

[Figure 16 Predict Price Page – 2 24](#_Toc126111137)

[Figure 17 Predict Price Page – 3 25](#_Toc126111138)

[Figure 18 Predict Price Page – 4 25](#_Toc126111139)

[Figure 19 Graphs Page – 1 26](#_Toc126111140)

[Figure 20 Graphs Page – 2 26](#_Toc126111141)

[Figure 21 Graphs Page – 3 27](#_Toc126111142)

[Figure 22 Admin Page 27](#_Toc126111143)

[Figure 23 UserRepository.cs 28](#_Toc126111144)

[Figure 24 AirabamDBContext.cs 28](#_Toc126111145)

[Figure 25 CookieUser.cs 29](#_Toc126111146)

[Figure 26 CookieUserItem.cs 29](#_Toc126111147)

[Figure 27 Hasher.cs 29](#_Toc126111148)

[Figure 28 UserManager.cs 30](#_Toc126111149)

[Figure 29 LoginVm.cs and RegisterVm.cs 30](#_Toc126111150)

[Figure 30 Car.cs and CarTest.cs 31](#_Toc126111151)

[Figure 31 FeatureValueAverage.cs 31](#_Toc126111152)

[Figure 32 FeatureValueOutput.cs 32](#_Toc126111153)

[Figure 33 ErrorViewModel.cs 32](#_Toc126111154)

[Figure 34 HomeController.cs 32](#_Toc126111155)

[Figure 35 AccountController.cs 33](#_Toc126111156)

[Figure 36 PricingModel.cs 34](#_Toc126111157)

[Figure 37 PricingModelForValidation.cs 34](#_Toc126111158)

[Figure 38 Authentication System 35](#_Toc126111159)

[Figure 39 Model Training System 35](#_Toc126111160)

[Figure 40 Scraping System 36](#_Toc126111161)

[Figure 41 Evaluation System 37](#_Toc126111162)

[Figure 42 Prediction System 37](#_Toc126111163)

[Figure 43 Model Validation System 38](#_Toc126111164)

[Figure 44 Feature Evaluation System 39](#_Toc126111165)

[Figure 45 Cars Table - Gathering Data 40](#_Toc126111166)

[Figure 46 Scrape Method - Step 1 42](#_Toc126111167)

[Figure 47 Scrape Method - Step 2 42](#_Toc126111168)

[Figure 48 Scrape Method - Step 3 43](#_Toc126111169)

[Figure 49 Scrape Method - Step 4 44](#_Toc126111170)

[Figure 50 Scrape Details Method 46](#_Toc126111171)

[Figure 51 Fix Model Names Method 47](#_Toc126111172)

[Figure 52 BuildPipeline Method - Feature Normalization 48](#_Toc126111173)

[Figure 53 Steps Explained 51](#_Toc126111174)

[Figure 54 ML.NET Diagram 1 52](#_Toc126111175)

[Figure 55 ML.NET Diagram 2 53](#_Toc126111176)

[Figure 56 ModelInput and ModelOutput classes 55](#_Toc126111177)

[Figure 57 PricingModel.cs - 1 56](#_Toc126111178)

[Figure 58 PricingModel.cs - 2 56](#_Toc126111179)

[Figure 59 PricingModel.cs - 3 57](#_Toc126111180)

[Figure 60 PricingModel.cs - 4 58](#_Toc126111181)

[Figure 61 PricingModel.cs – 5 59](#_Toc126111182)

[Figure 62 PricingModel.cs – 6 59](#_Toc126111183)

[Figure 63 PricingModel.cs – 7 60](#_Toc126111184)

[Figure 64 PricingModel.cs - 8 61](#_Toc126111185)

[Figure 65 TrainPrice Method 62](#_Toc126111186)

[Figure 66 PricingModelForValidation.cs – 1 64](#_Toc126111187)

[Figure 67 CrossValidation Method 65](#_Toc126111188)

[Figure 68 Average Difference Between Prediction and Real Price 67](#_Toc126111189)

[Figure 69 EvaluateFeatures Method 68](#_Toc126111190)

[Figure 70 Method That Creates Example Cars 69](#_Toc126111191)

[Figure 71 Last 10 and Top 10 Brands 71](#_Toc126111192)

[Figure 72 Last 10 and Top 10 Models 72](#_Toc126111193)

[Figure 73 Average Fuel Consumption’s Impact on the Price 73](#_Toc126111194)

[Figure 74 Impact of Car Case on the Price 73](#_Toc126111195)

[Figure 75 Impact of Color on the Price 74](#_Toc126111196)

[Figure 76 Impact of Gear Type on the Price 74](#_Toc126111197)

[Figure 77 Impact of City on the Price 75](#_Toc126111198)

[Figure 78 Impact of Fuel Type on the Price 76](#_Toc126111199)

1. INTRODUCTION

Airabam is an AI powered web application where you can select features of a used car and let the artificial intelligence decide its value. The web application allows users to interact with a machine learning model that has been trained on thousands of car adverts over the last quarter of 2022 in Turkey. The model created is making predictions based on the said training data.

Another use of the project is to use artificial intelligence to make assumptions on the current state of second-hand car market in Turkey.

Project also contains an admin panel where the admin can train the model, evaluate it, gather data for the model and make some modifications to the data before training the model.

1.1 Tools Used in the Project

There are many useful frameworks, wrappers and libraries that we can utilize for a project like this. Below you can see the technologies I used in order to build Airabam.

| Technology | Description |
| --- | --- |
| ASP.NET | A web application framework used for building web applications |
| JQuery | A fast and concise JavaScript library used for HTML document traversal and manipulation |
| Ajax | A technique used for creating fast and dynamic web pages |
| Bootstrap | A responsive front-end framework used for designing websites and web applications |
| SweetAlerts | A library used for creating customized alert messages in a web application |
| Chart.js | A JavaScript library used for creating charts and graphs in a web application |
| SQLite | A database engine used for storing and retrieving data in a lightweight and fast manner |
| ML.NET | A machine learning framework used for building custom machine learning models |
| HtmlAgilityPack | A .NET library used for parsing and manipulating HTML content |
| EntityFramework | A data access technology used for working with relational databases in .NET applications |

Table 1 Tools Used in the Project

1.2 Project Development Timeline

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Oct 2022 | Nov 2022 | Dec 2022 | Jan 2023 |
| Authentication |  |  |  |  |
| Backend Development |  |  |  |  |
| Gathering Data |  |  |  |  |
| Frontend Development |  |  |  |  |
| ML Model |  |  |  |  |
| Graphs |  |  |  |  |
| Project Report |  |  |  |  |

Table 2 Project Development Timeline

1.3 System Model

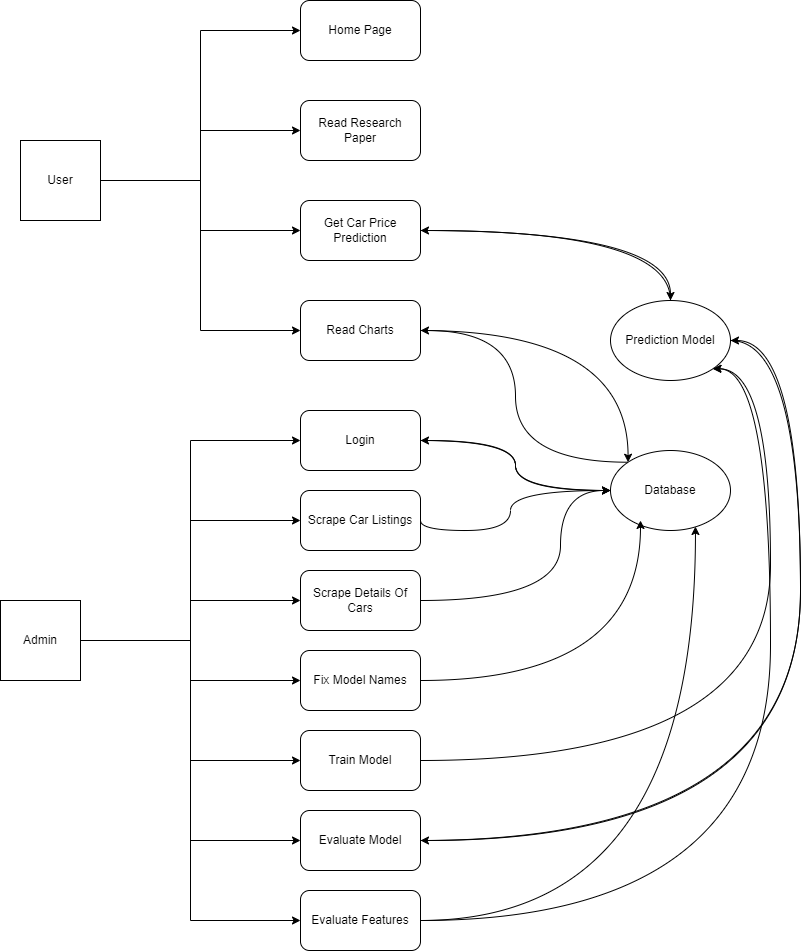


Figure 1. System Model

2. KEY THEORETICAL CONCEPTS

In this chapter key theoritical concepts that are applied while developing this project are explained.

2.1 Artificial Intelligence and Machine Learning

With the use of artificial intelligence, a machine may replicate human behavior. Machine learning, a subfield of artificial intelligence, allows a system to automatically learn from past data without explicit programming. AI attempts to develop smart computer systems that can solve complex problems like humans. ( Pati, 2021)

Through the use of machine learning, computers can access information that people cannot. It's challenging to explain in simple terms how our vision and language systems function. Because of this, we rely on data and feed it to computers so they can mimic what we're doing. Machine learning accomplishes this, allowing machines to learn from data in order to provide accurate output. ( Pati, 2021)

2.2 Supervised and Unsupervised Learning

Supervised learning is a type of machine learning where the model is trained on a labeled dataset, meaning that the desired output (or "label") for each input example is provided. The goal is for the model to learn a mapping from inputs to outputs and be able to make predictions on unseen data. Some examples of supervised learning tasks include classification and regression.

Unsupervised learning, on the other hand, is a type of machine learning where the model is not provided with labeled data. Instead, the model's objective is to discover structure or patterns in the incoming data. Clustering and dimensionality reduction are a few of instances of unsupervised learning problems.

AIrabam uses supervised learning in order to guess the prices of cars (the label) based on the input data.

2.3 Regression Problems and Models

When the output variable is a real or continuous value, like "salary" or "weight," a regression problem exists. The most straightforward model, while there are many more, is linear regression. The optimal hyper-plane that traverses the points is attempted to suit the data. (Shukla, 2023)

So in a regression problem our goal is to predict a point in the space that is as close to the real point as possible. And a regression model is a model that provides the means to achieve this.

A regression model provides an explanation of the relationship between one or more independent variables and a response, dependent, or target variable. (IMSL, 2021)

2.4 Categorical and Numerical Values

Another important thing to learn about before delving into the base of our project is categorical and numerical values. While building our model, we will have to specify which values are categorical and which values are numerical. This will change the way our model handles these values.

To give an example a categorical value can be the car brand, like Wolkswagen. And a numerical value can be the fuel consumption. Then there are some values that seem numerical but actually they are categorical, for an example the ID of the car is not numerical. It is not continuous, and when it increases or decreases it does not refer to an increment to an attribute, it now refer to a totally different car.

2.5 Fast Tree Regression Algorithm

Fast Tree Regression is an algorithm for building decision tree models for regression tasks. It is a variation of the traditional decision tree algorithm that aims to improve the speed of model training and prediction without sacrificing accuracy.

The basic idea behind decision tree algorithms is to recursively divide the input space into smaller and smaller regions, called leaf nodes, and assign a constant value to each of these regions. These constant values are determined by the training data and are used to make predictions on new inputs.

Fast Tree Regression is an improvement over the traditional decision tree algorithm in that it uses a technique called gradient boosting to iteratively train a series of decision trees. Each tree is trained to correct the errors made by the previous trees in the sequence. The final model is a combination of all the trees in the sequence, which are combined by taking a weighted average of their predictions.

The algorithm also uses a technique called histogram binning, which improves the speed of model training by reducing the number of unique input values that need to be considered for each split. It also uses a technique called pruning, which aims to remove unnecessary splits from the tree and reduce the complexity of the model.

In summary, Fast Tree Regression is a variation of the traditional decision tree algorithm that uses techniques such as gradient boosting, histogram binning, and pruning to improve the speed of model training and prediction without sacrificing accuracy.

2.5.1 Decision Trees

A particular kind of machine learning algorithm called a decision tree algorithm is used to create models for classification and regression. It is a type of model that is easy to understand and interpret, and it can handle both numerical and categorical data.

The basic idea behind decision tree algorithms is to recursively divide the input space into smaller and smaller regions, called leaf nodes, and assign a constant value to each of these regions. The constant value can be a class label (in the case of classification) or a real value (in the case of regression).

The process of building a decision tree model starts with the root node, which represents the entire input space. The algorithm then selects the feature and the corresponding threshold value that best splits the input space into two regions such that the resulting regions are as pure as possible. The pure region means one region contains only one class for classification or has similar values for regression.

The process is then repeated for each of the resulting regions, until a stopping criterion is met. The stopping criterion can be a maximum tree depth, a minimum number of samples in a leaf node, or a minimum decrease in impurity.

The final decision tree model is a tree-like structure, where each internal node represents a feature and a threshold value, and each leaf node represents a class label or a real value. To make a prediction on a new input, the algorithm starts at the root node and follows the path down the tree that corresponds to the input's feature values, until it reaches a leaf node. The value assigned to that leaf node is the prediction.

2.5.2 Gradient Boosting

Gradient boosting is a technique used in machine learning to improve the accuracy of a model by iteratively training a series of weak models and combining their predictions. The technique is often used in conjunction with decision trees, but can be used with other types of models as well.

The fundamental concept of gradient boosting is to train a number of basic models, such as decision trees, in sequence. Each model is trained to correct the errors made by the previous models in the sequence. The final model is a combination of all the models in the sequence, which are combined by taking a weighted average of their predictions.

The process starts by training the first model on the input data, and then calculating the errors made by the model on the training data. These errors are used to update the weights of the training examples, so that the next model in the sequence focuses more on the examples that were misclassified by the previous model.

If the model's performance on a validation set stops improving after a predetermined number of iterations, the process is repeated. The final model is a weighted combination of all the individual models in the sequence.

The main advantage of gradient boosting is that it allows to combine weak models, such as shallow decision trees, in a way that can lead to a more accurate final model. This is because each weak model is able to focus on different aspects of the input data and correct the errors made by the previous models in the sequence.

2.5.3 Histogram Binning

Histogram binning is a technique used in machine learning to reduce the number of unique input values that need to be considered when building a model. It is particularly useful for decision tree algorithms and other models that rely on dividing the input space into smaller regions.

The basic idea behind histogram binning is to divide the range of input values into a fixed number of intervals or bins. Each bin represents a range of input values, and all input values within a bin are treated as if they have the same value. This reduces the number of unique input values that need to be considered when building a model, which can improve the speed of model training and prediction.

For example, if you are trying to build a decision tree model to predict the price of a house based on its square footage, you could use histogram binning to group all houses with square footage between 1,000 and 1,499 square feet into the same bin, and all houses with square footage between 1,500 and 1,999 square feet into another bin.

When using histogram binning, it's important to choose the number of bins and the bin size carefully. A too high number of bins can lead to overfitting and a too low number of bins can lead to underfitting. The optimal bin size is usually found by trying different bin sizes and evaluating the model performance with a validation set.

3. PROJECT DESIGN

In this chapter the structure of the project is explained in further detail with the help of diagrams and visuals.

3.1 An Overview of the Folder Structure and Project Files

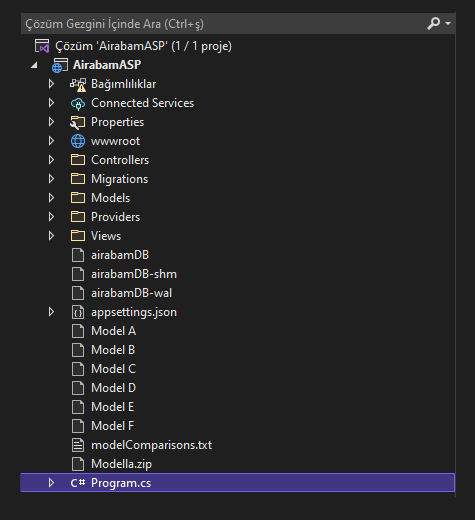


Figure 2 Folder Structure

The project contains several directories and files that are used to build a web application that utilizes AI and machine learning. The "wwwroot" directory holds resources such as images, css files, and javascript files, as well as frontend libraries like Bootstrap and JQuery. The "Controllers" directory has two classes that control the flow of the application, while the "Migrations" directory holds files related to database migrations. The "Models" directory contains ten classes that define the data model for the application, and the "Providers" directory contains classes that manage authentication and provide an interface to the machine learning model. The "Views" directory contains thirteen cshtml files that define the user interface. There are also SQLite database files, machine learning model files, a text file with cross validation results, and a zipped machine learning model named "Modella.zip" used by the web application. The "Program.cs" file is the central file of the web application and contains essential elements such as authentication settings and database connections.

3.2 Design of the Database

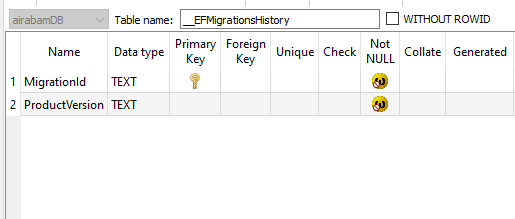


Figure 3 \_\_EFMigrationsHistory Table Structure

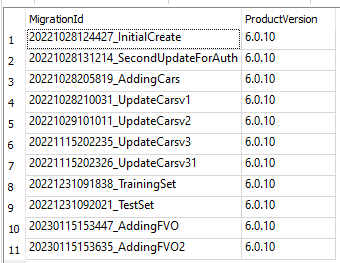


Figure 4 \_\_EFMigrationsHistory Table Data Sample

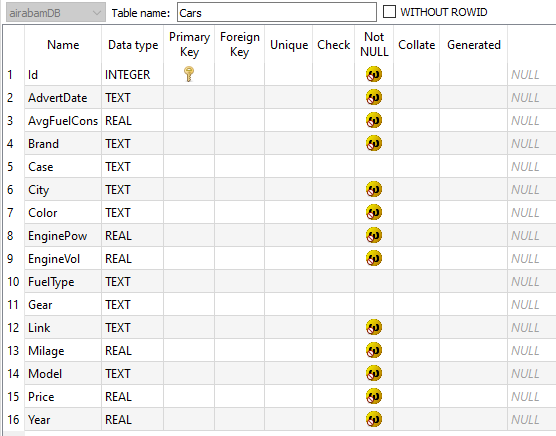


Figure 5 Cars Table Structure

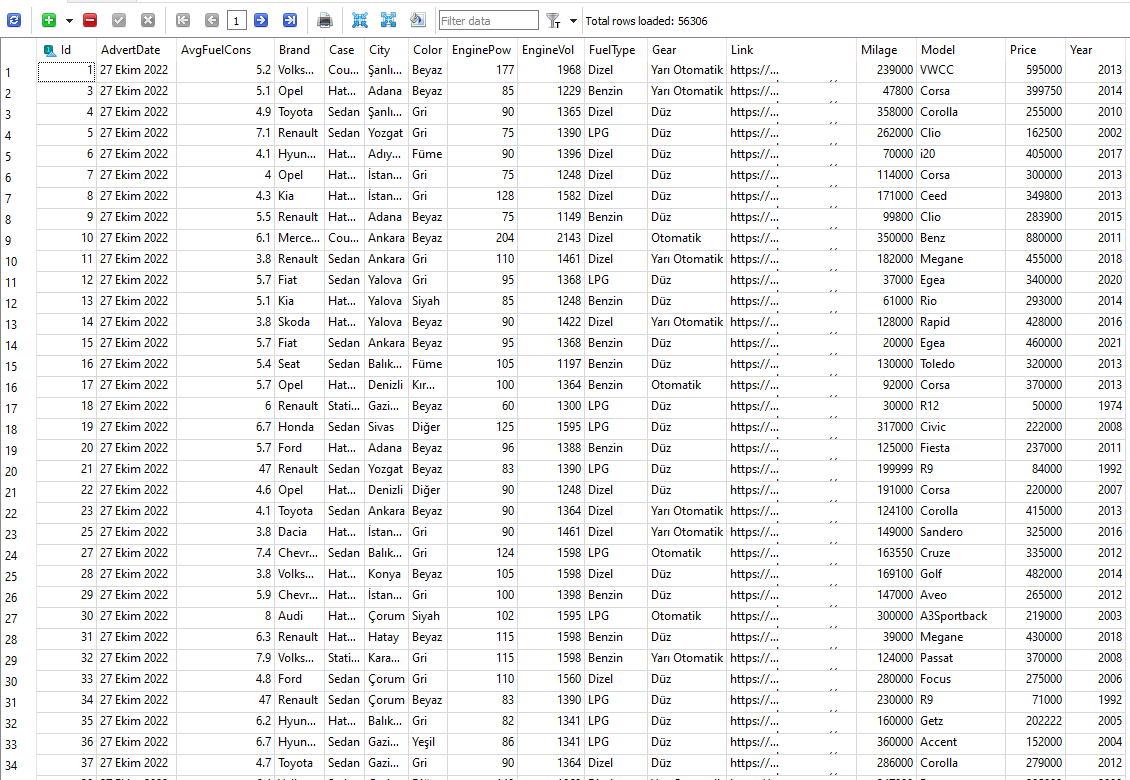


Figure 6 Cars Table Data Sample

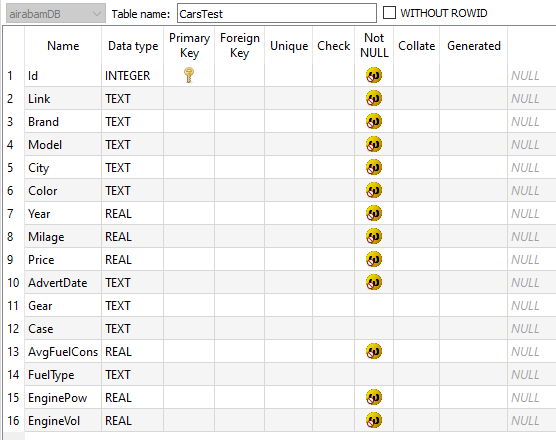


Figure 7 CarsTest Table Structure

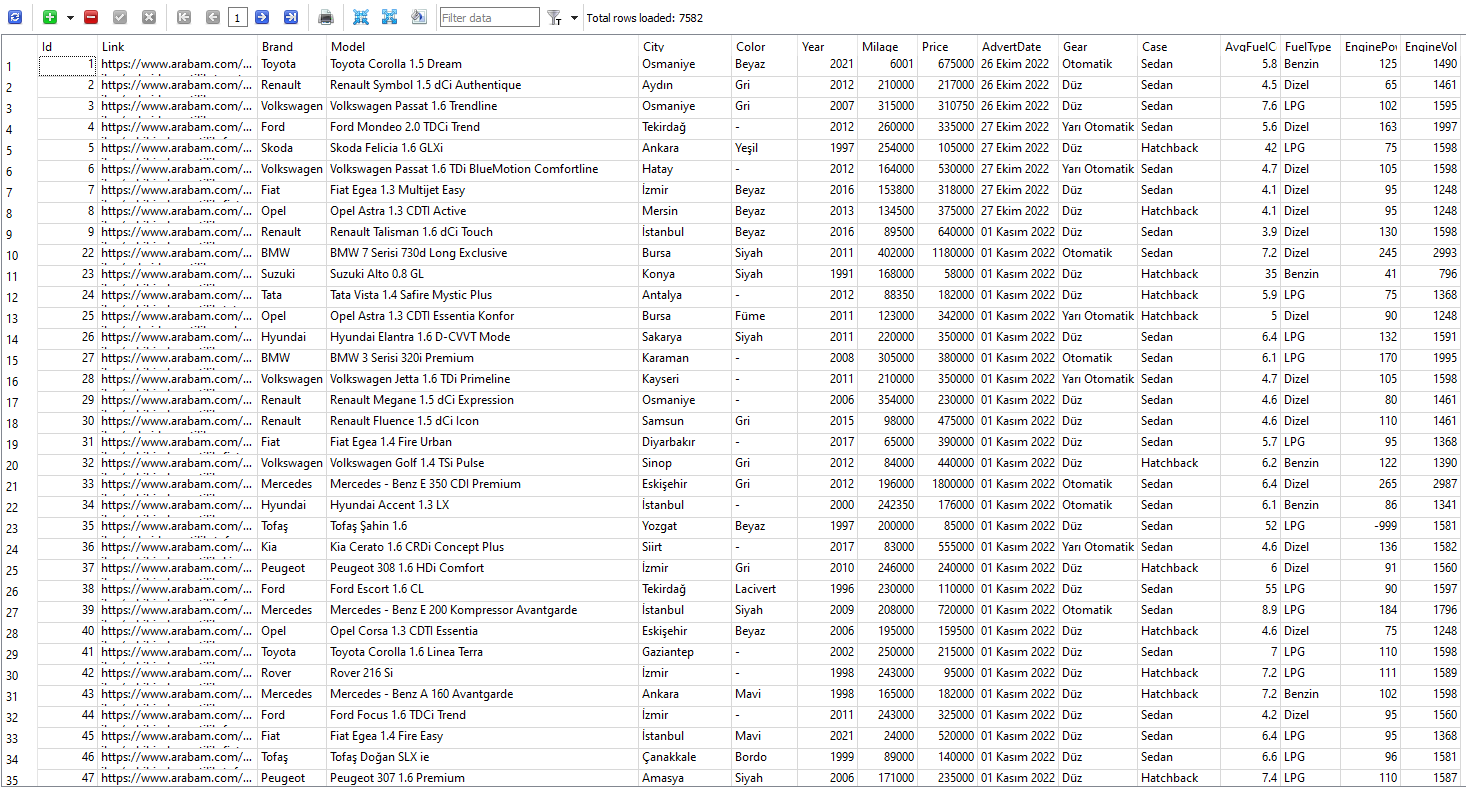


Figure 8 CarsTest Table Data Sample

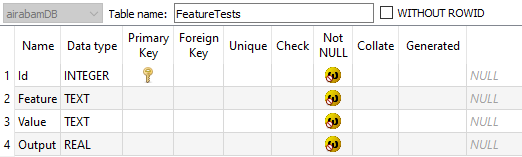


Figure 9 FeatureTests Table Structure

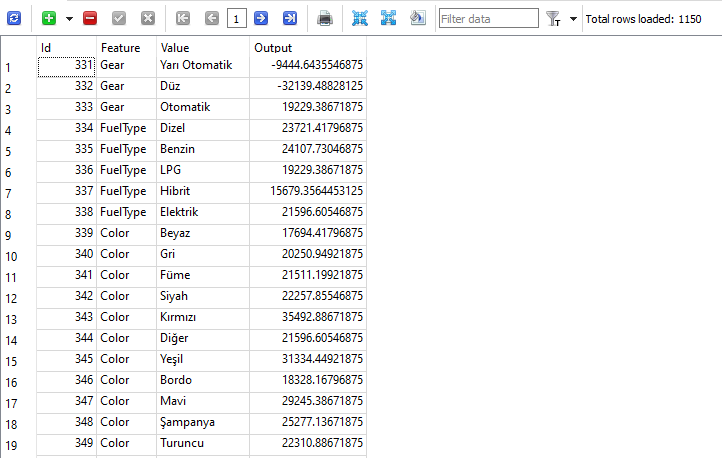


Figure 10 FeatureTests Table Data Sample

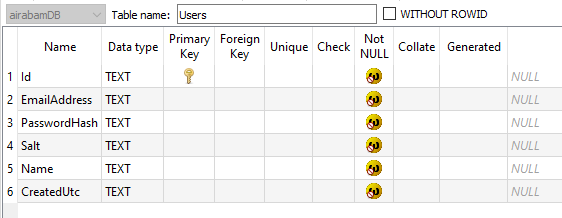


Figure 11 Users Table Structure

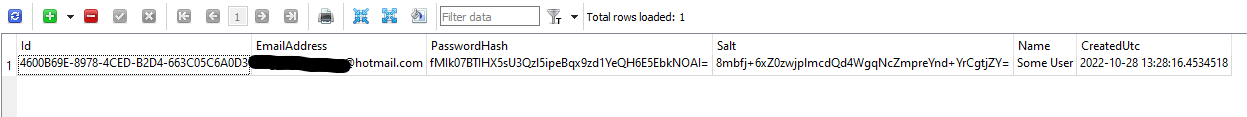


Figure 12 Users Table Data Sample

3.3 User Interface

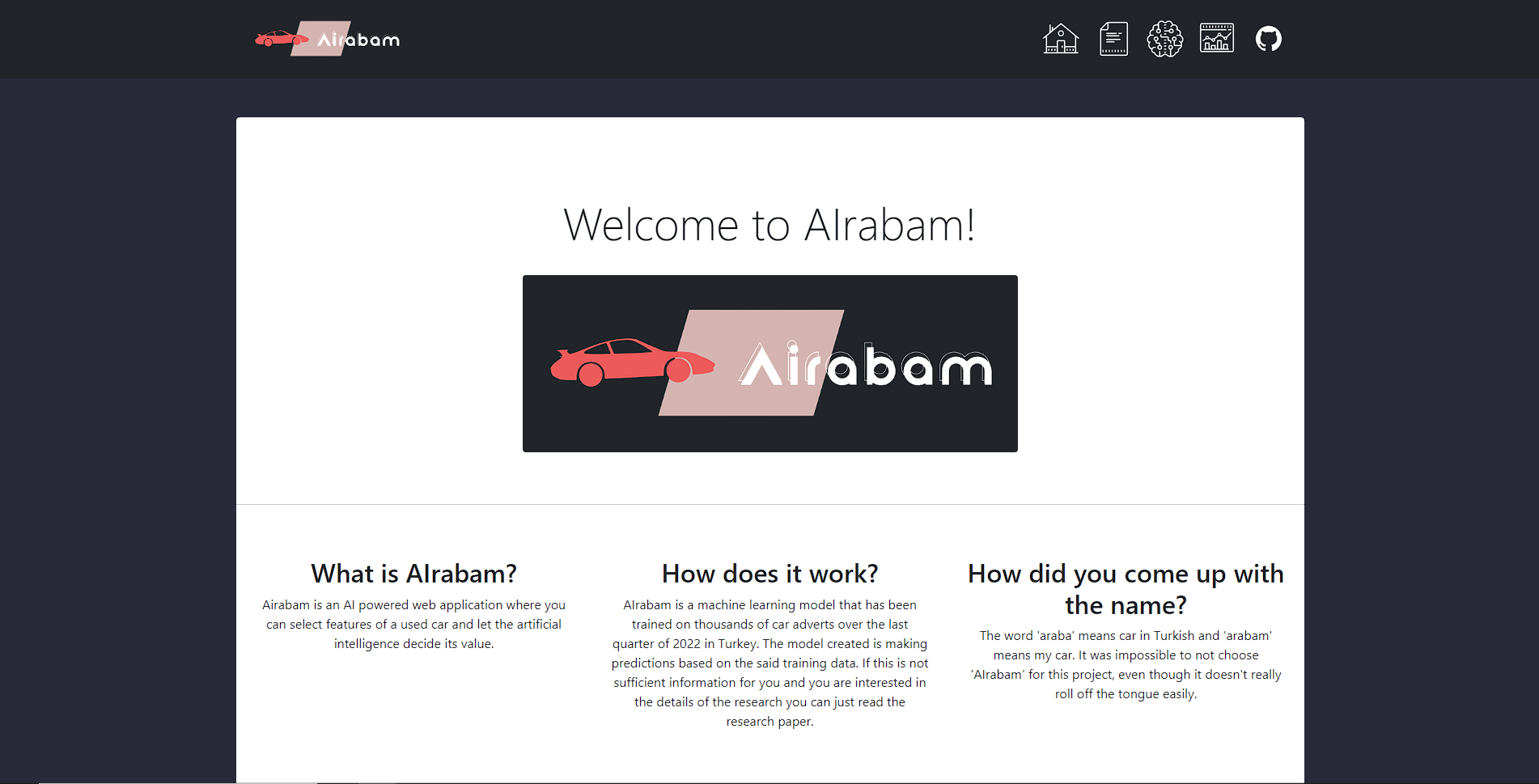


Figure 13 Home Page - 1

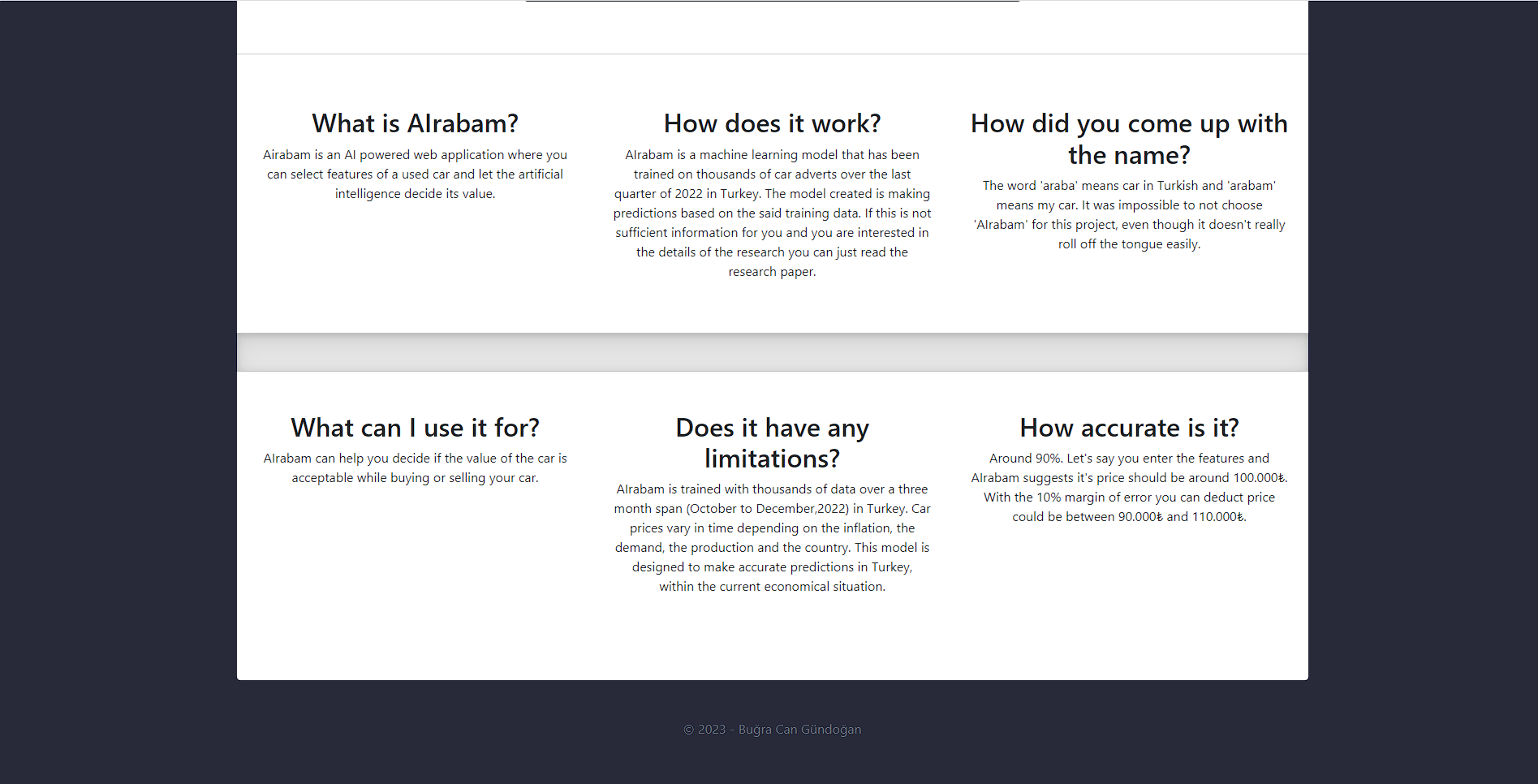


Figure 14 Home Page – 2

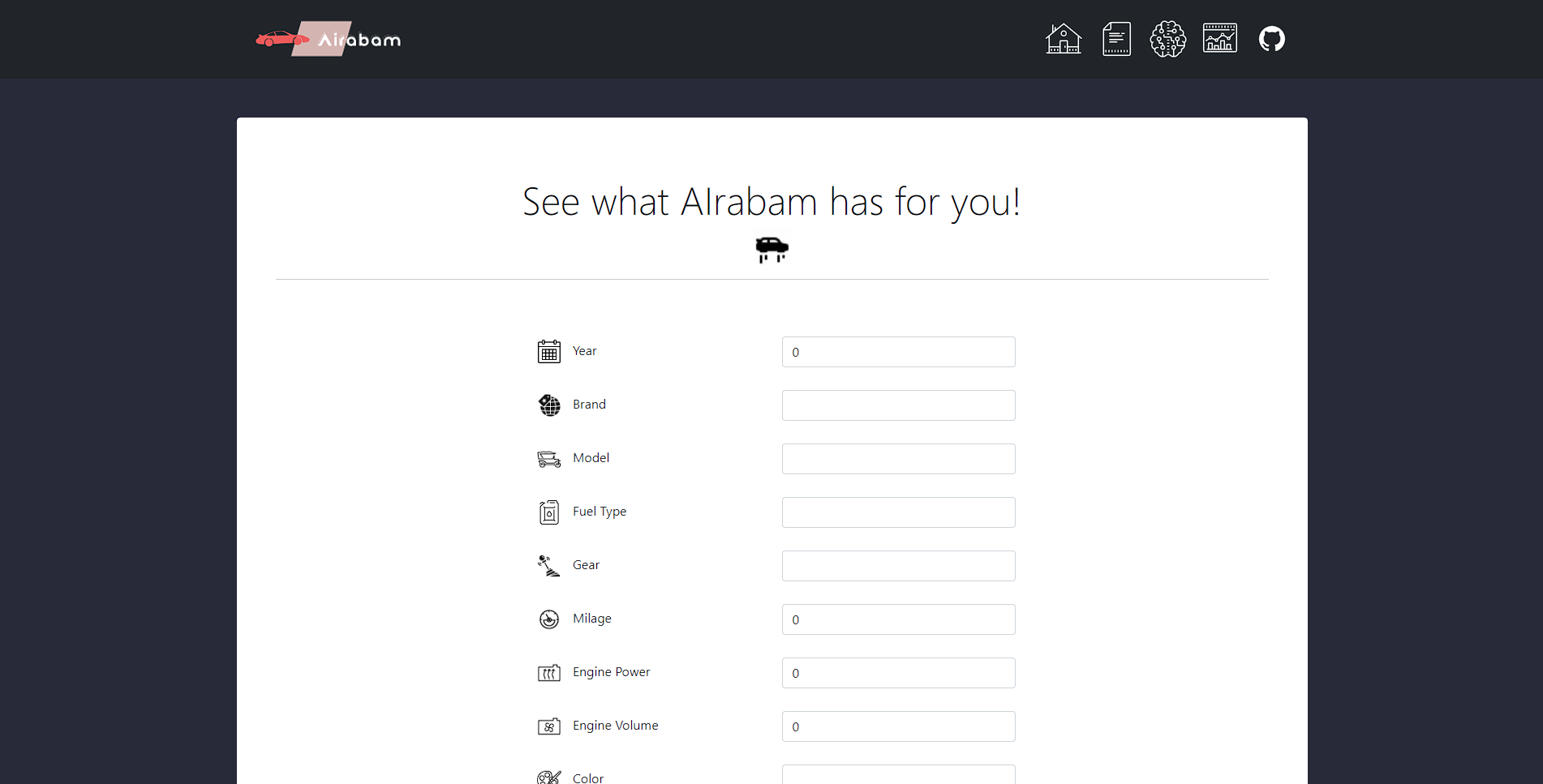


Figure 15 Predict Price Page – 1

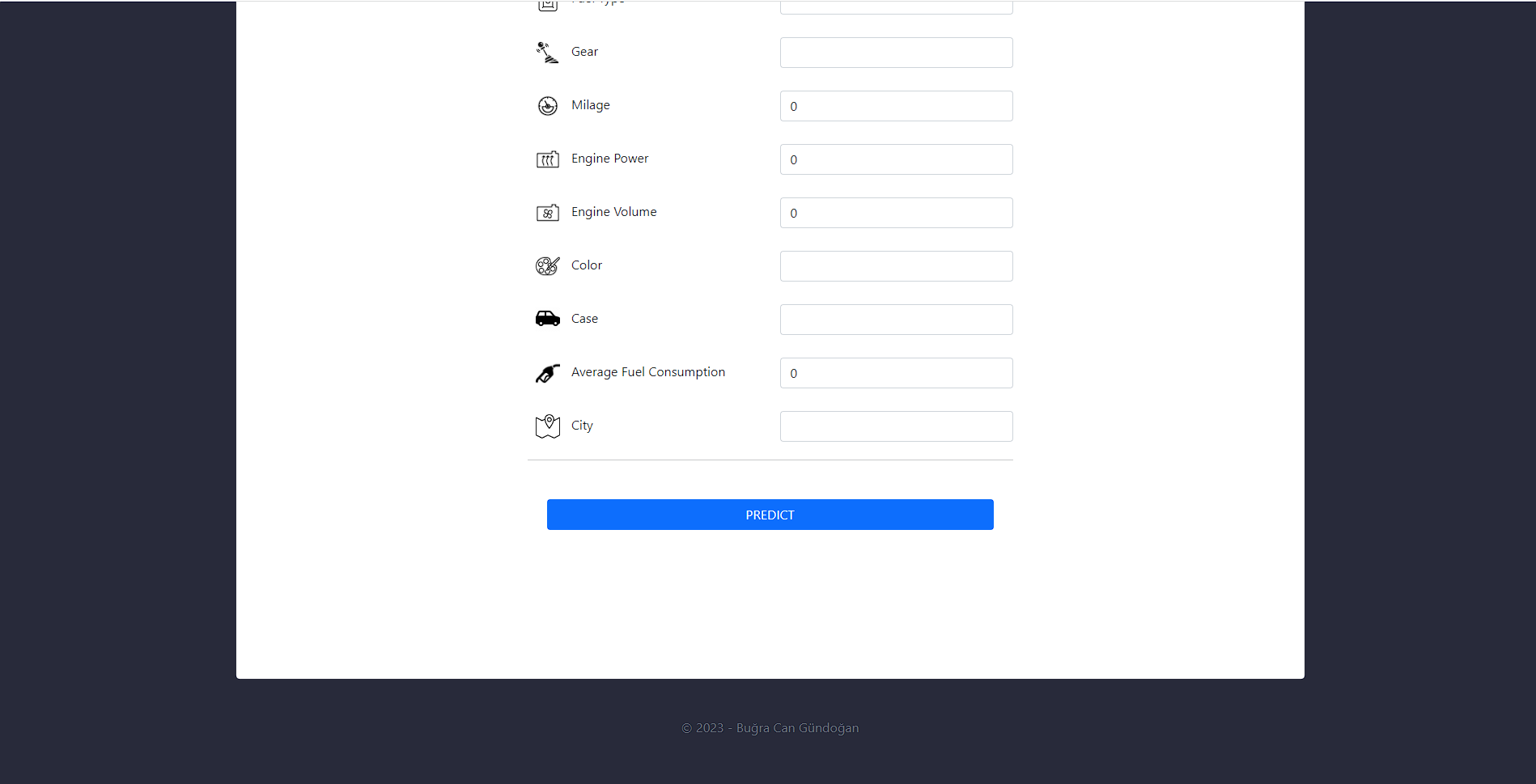


Figure 16 Predict Price Page – 2

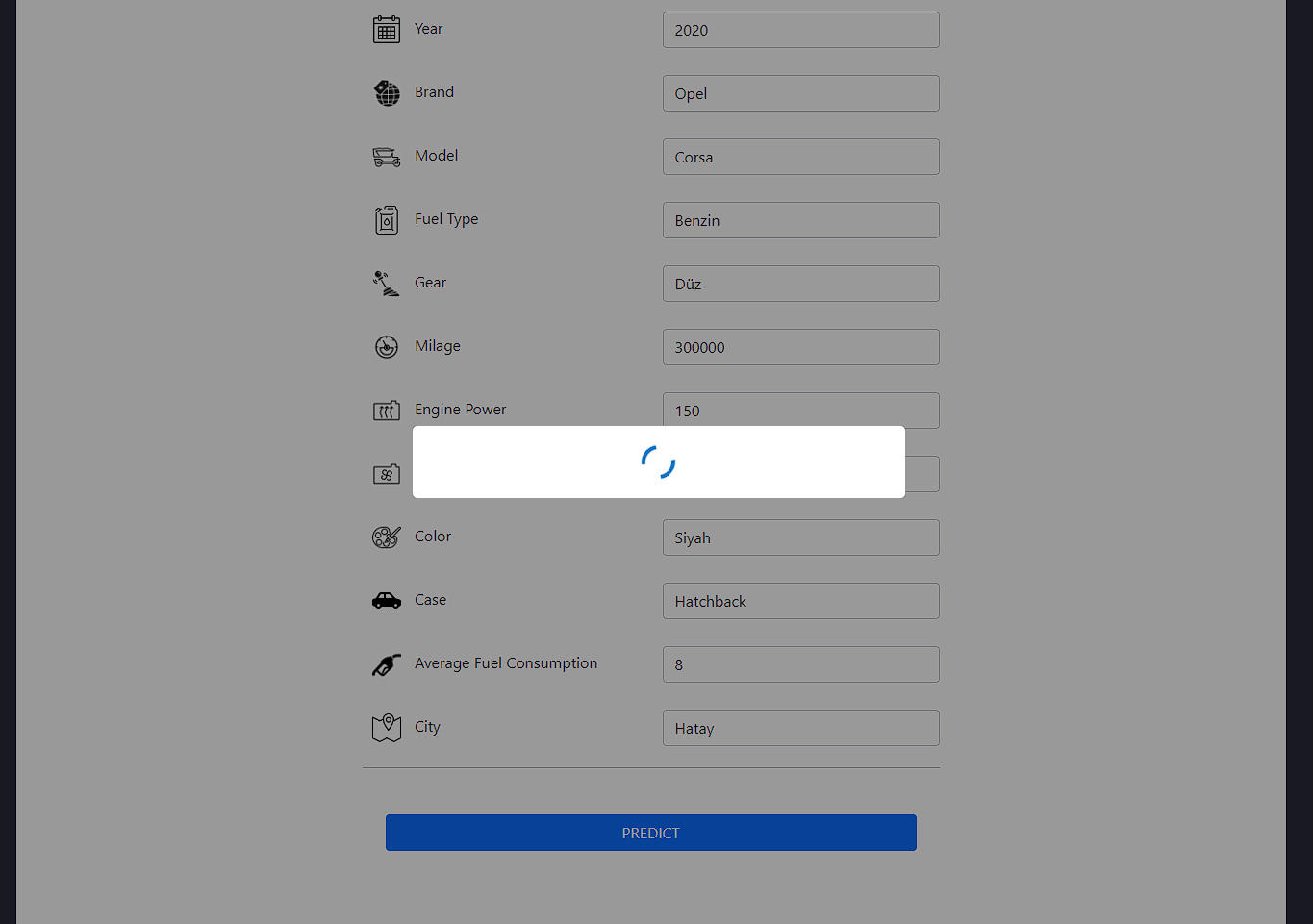


Figure 17 Predict Price Page – 3

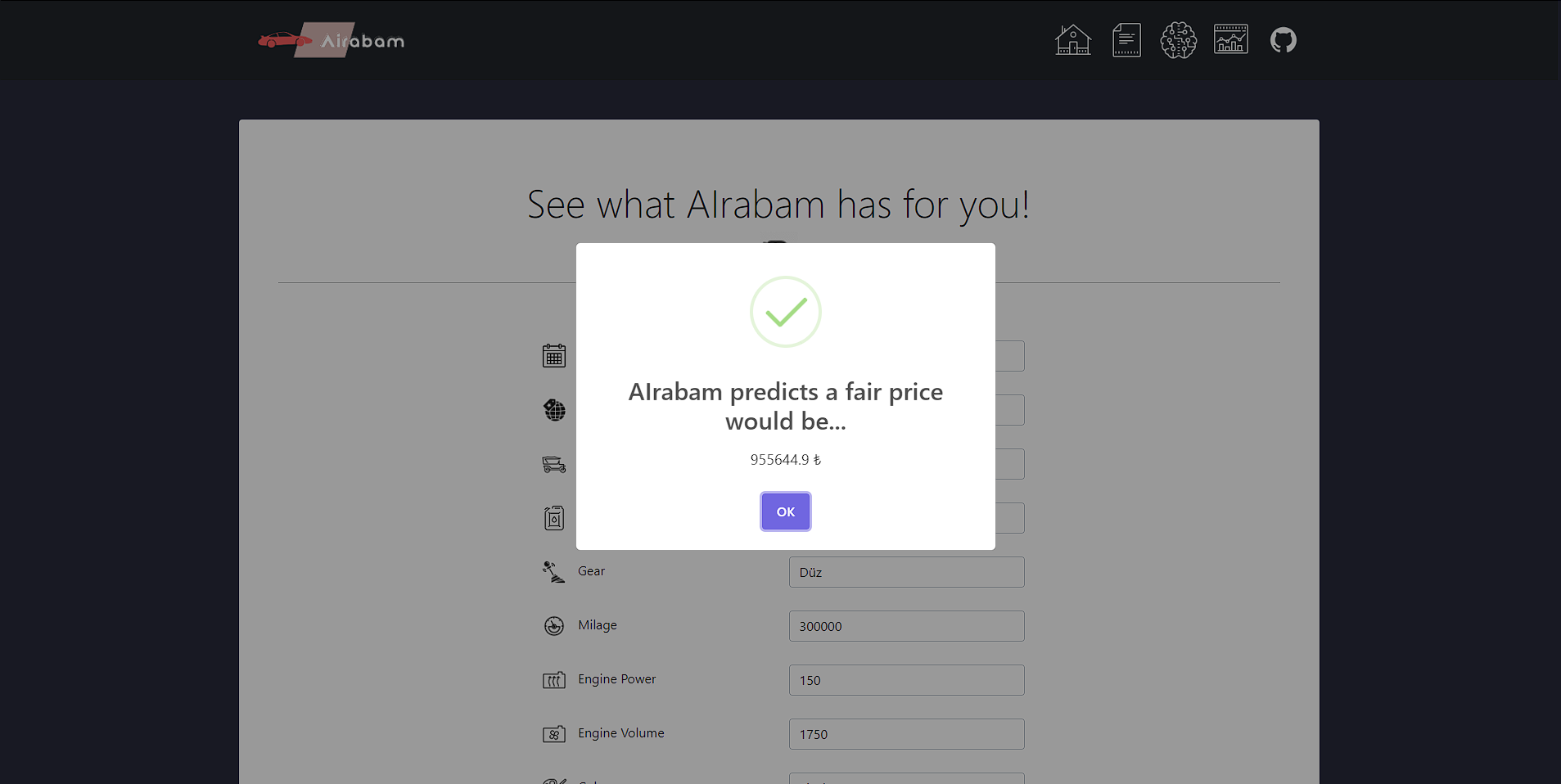


Figure 18 Predict Price Page – 4

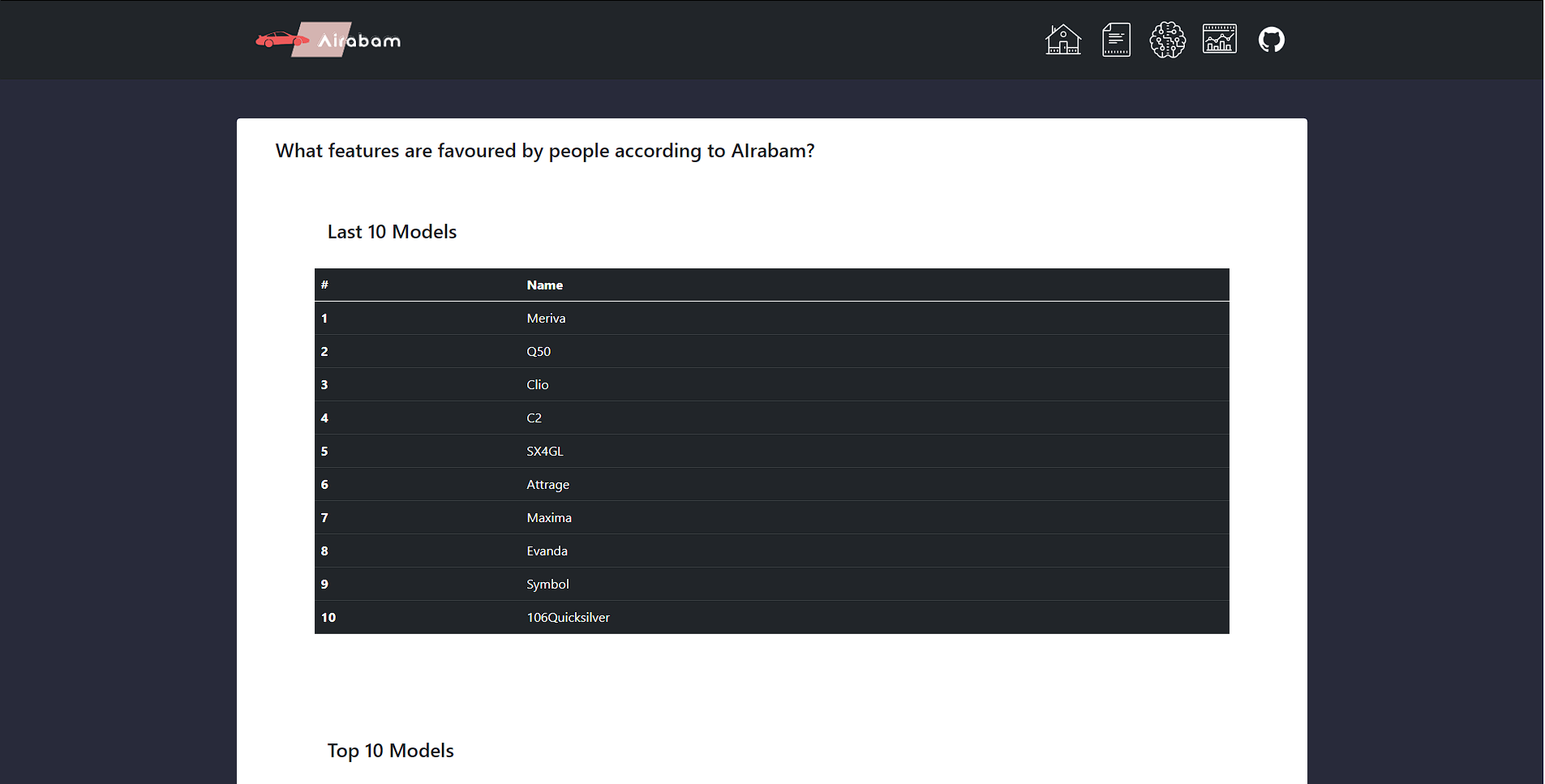


Figure 19 Graphs Page – 1

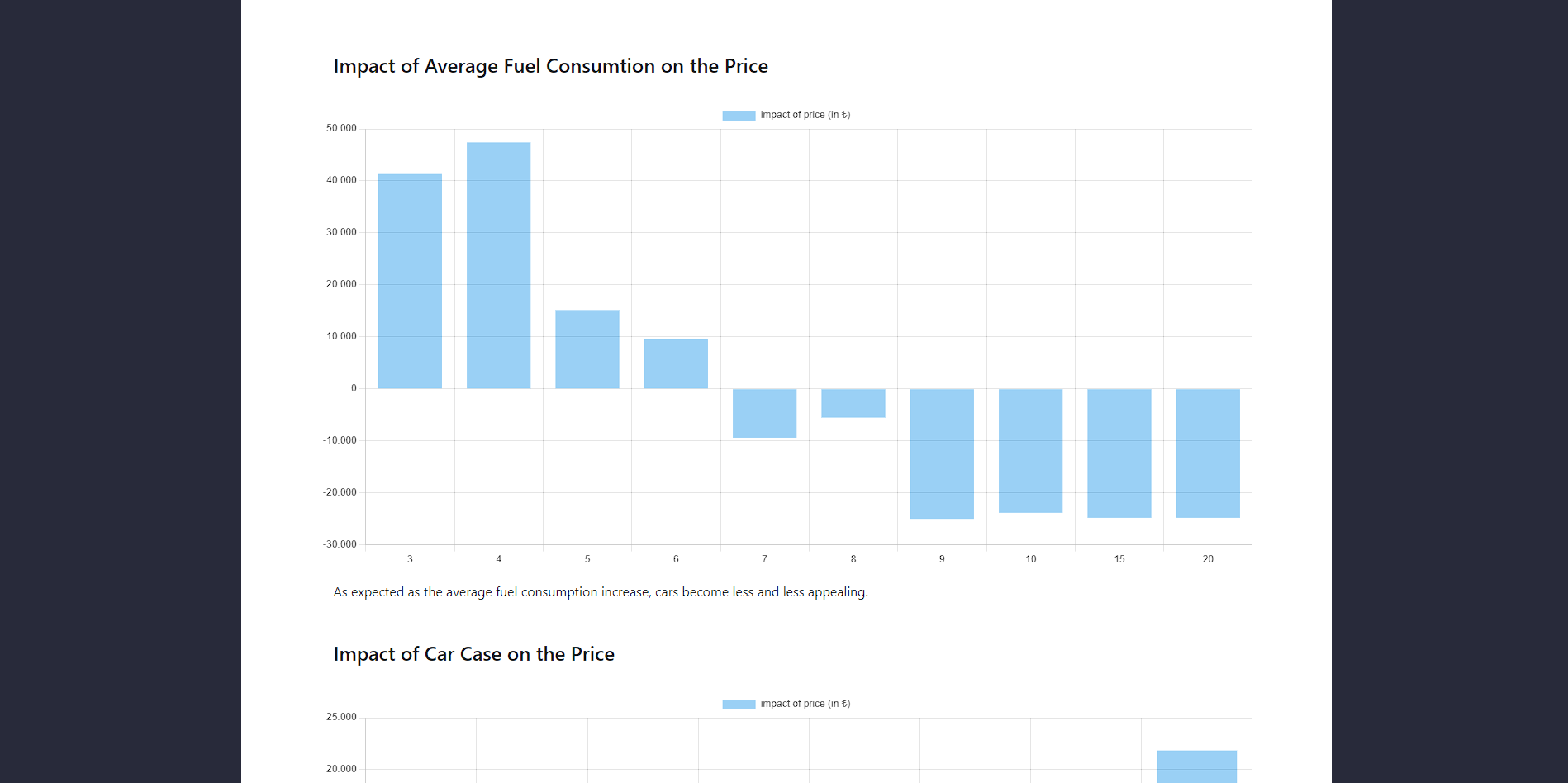


Figure 20 Graphs Page – 2

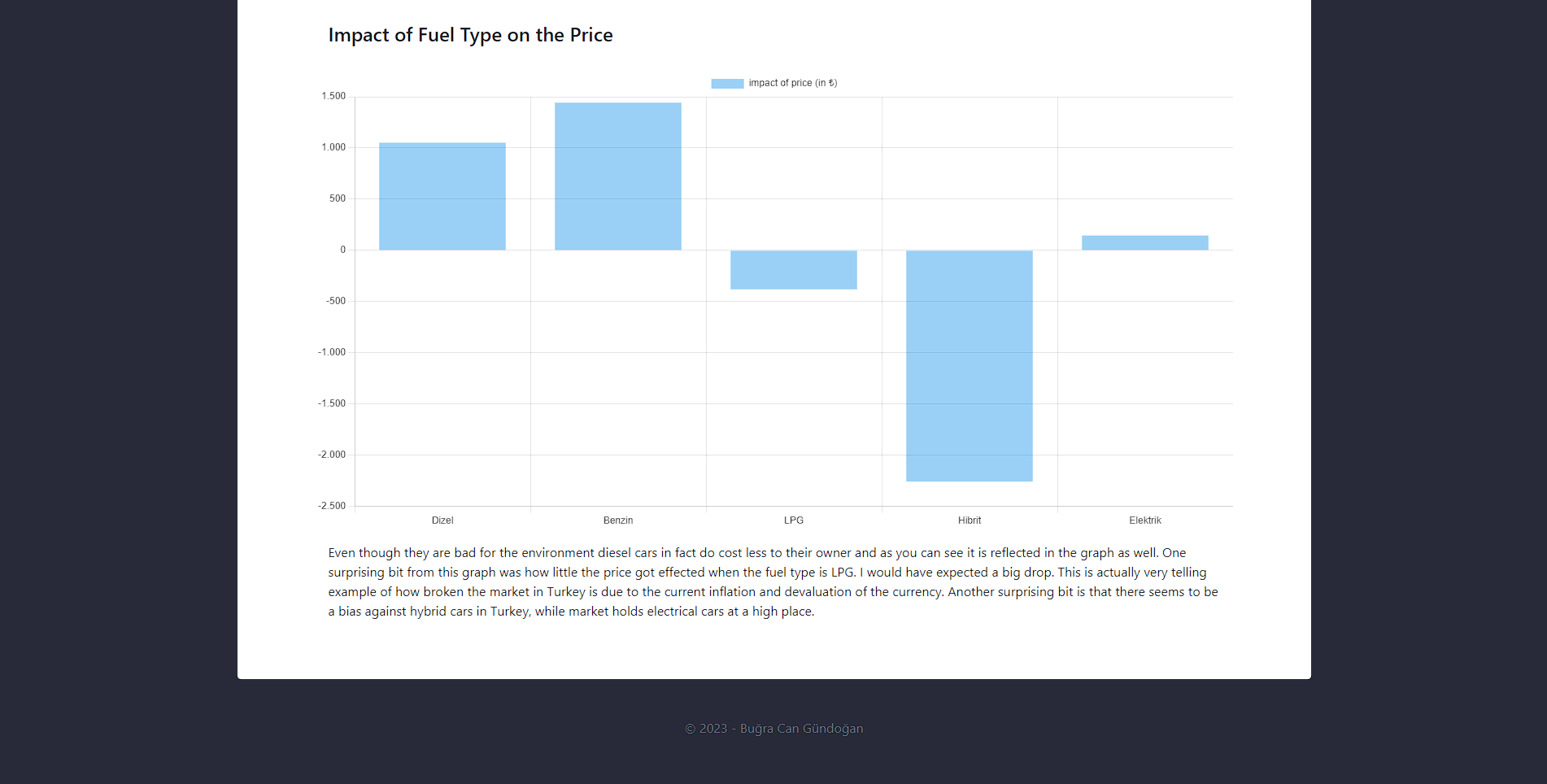


Figure 21 Graphs Page – 3

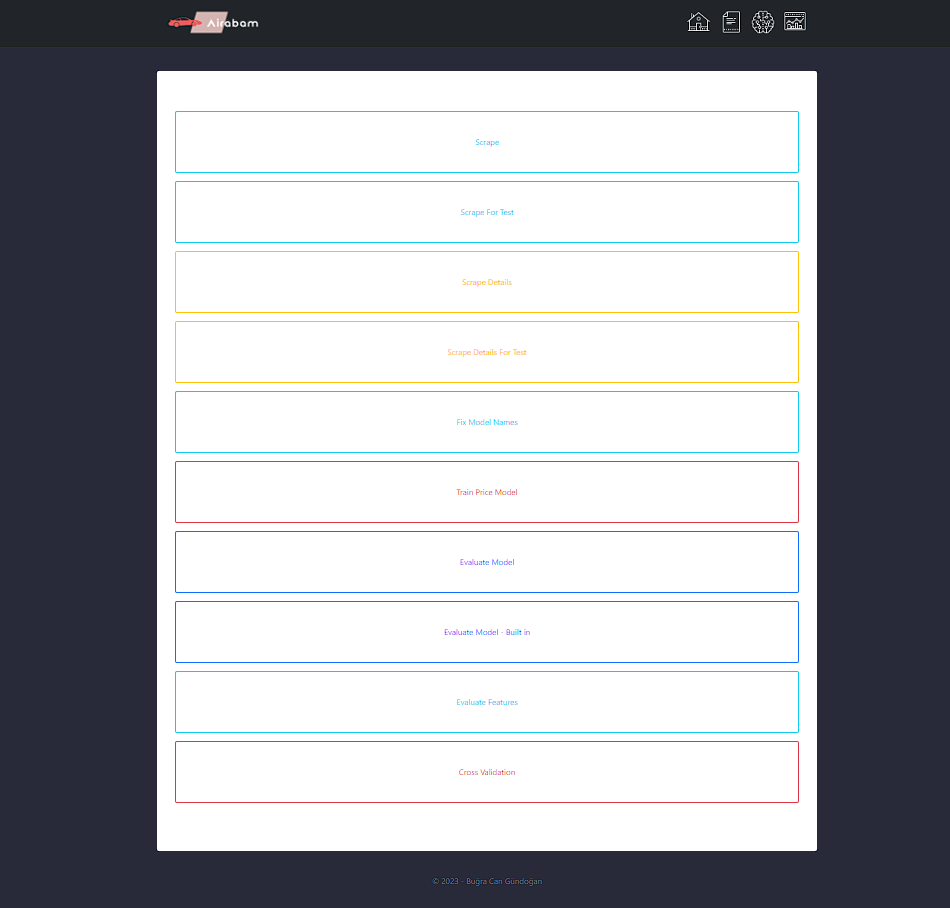


Figure 22 Admin Page

3.4 Design of Classes

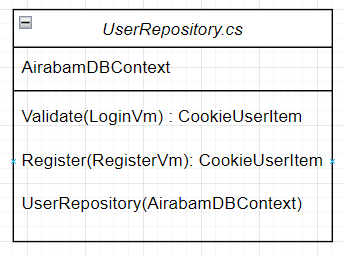


Figure 23 UserRepository.cs

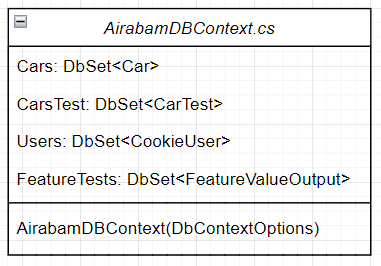


Figure 24 AirabamDBContext.cs

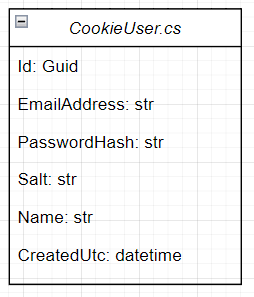


Figure 25 CookieUser.cs

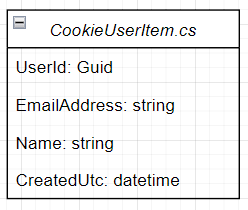


Figure 26 CookieUserItem.cs

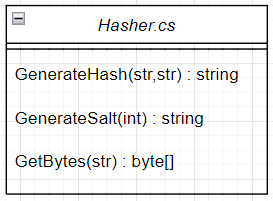


Figure 27 Hasher.cs

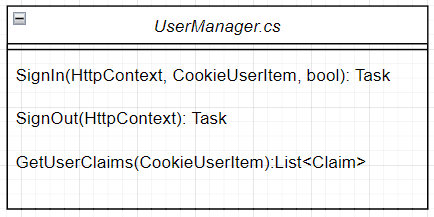


Figure 28 UserManager.cs

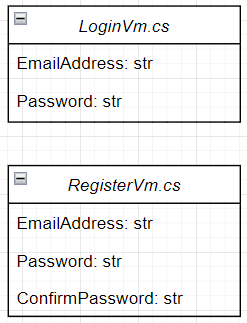


Figure 29 LoginVm.cs and RegisterVm.cs



Figure 30 Car.cs and CarTest.cs

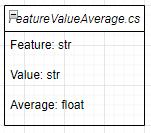


Figure 31 FeatureValueAverage.cs

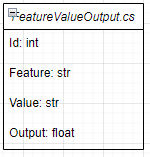


Figure 32 FeatureValueOutput.cs

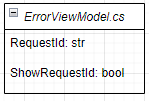


Figure 33 ErrorViewModel.cs

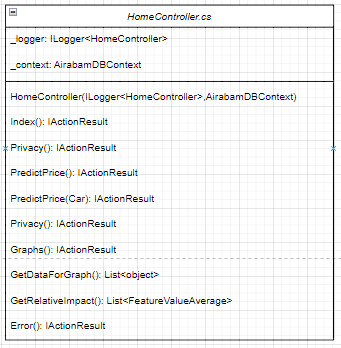


Figure 34 HomeController.cs

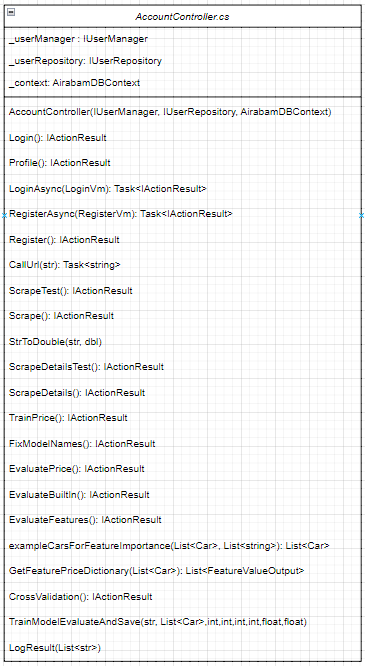


Figure 35 AccountController.cs

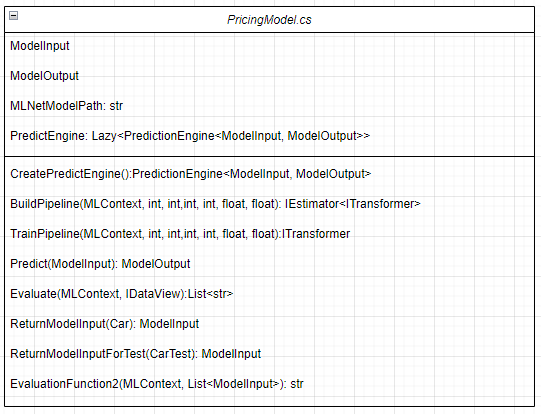


Figure 36 PricingModel.cs

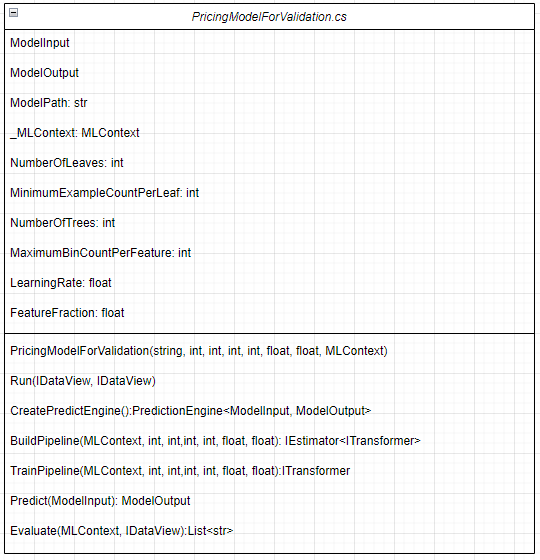


Figure 37 PricingModelForValidation.cs

3.5 Interactions Between Classes

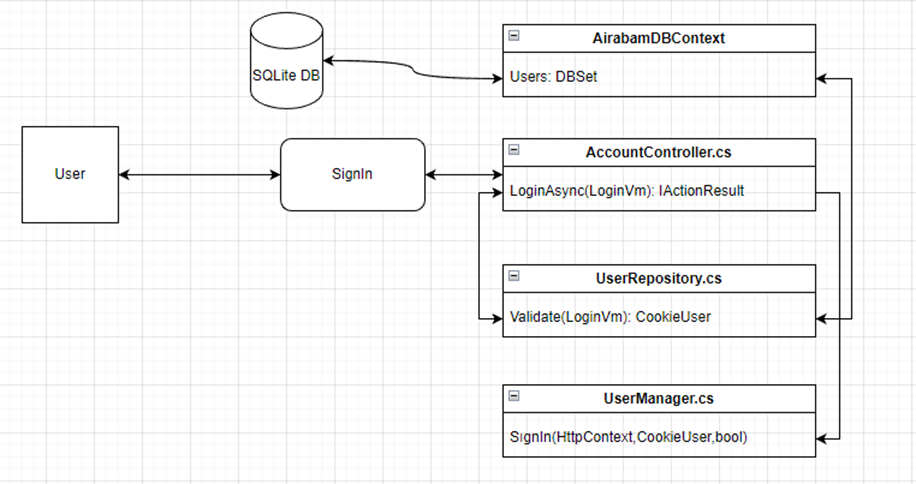


Figure 38 Authentication System

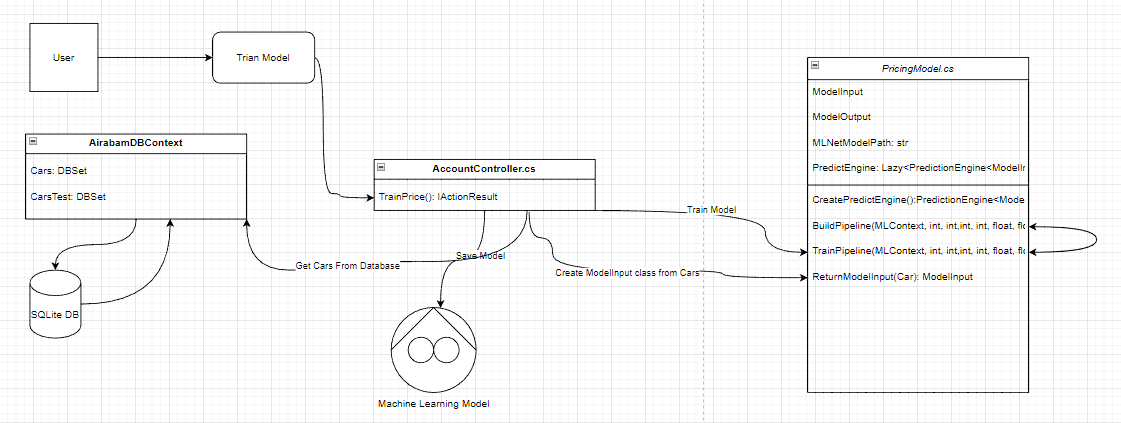


Figure 39 Model Training System

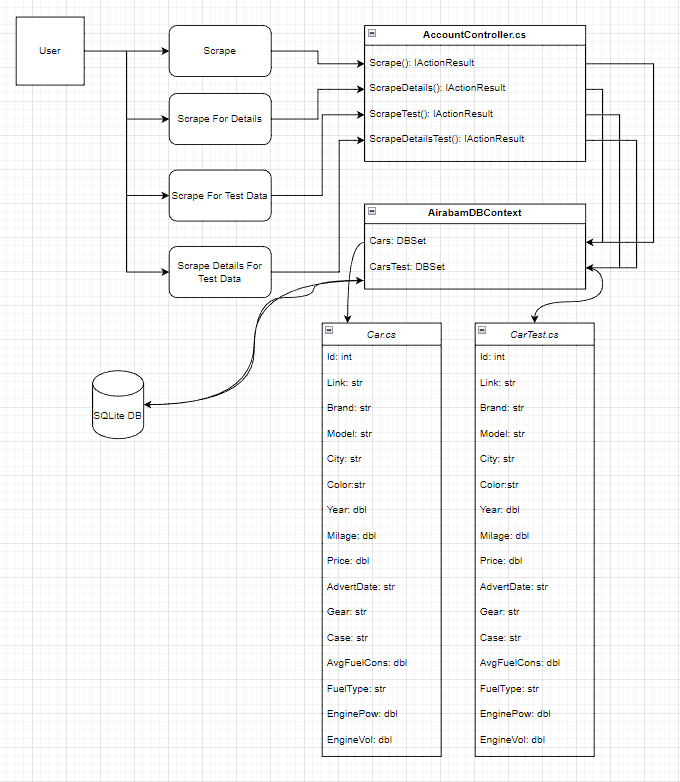


Figure 40 Scraping System

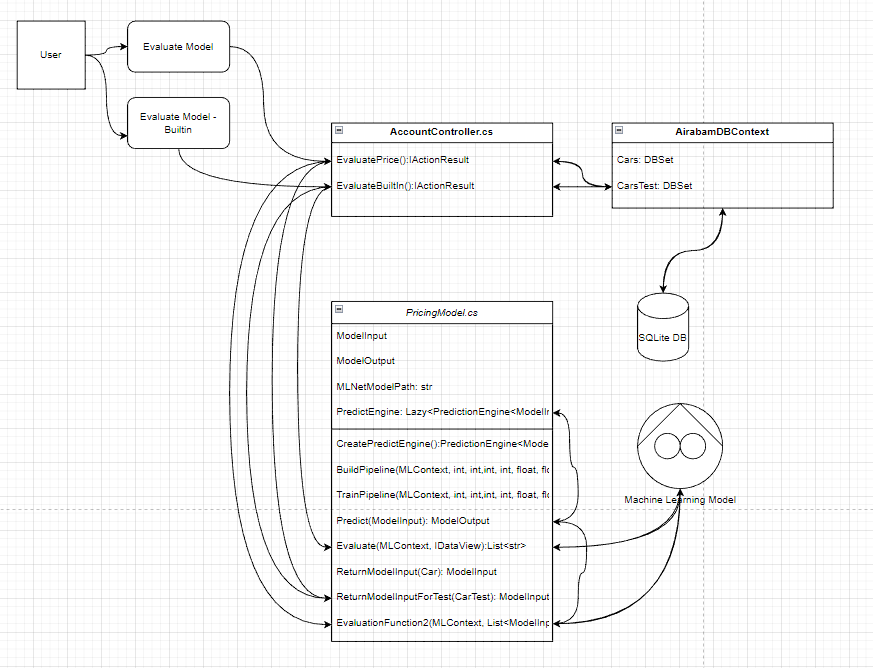


Figure 41 Evaluation System

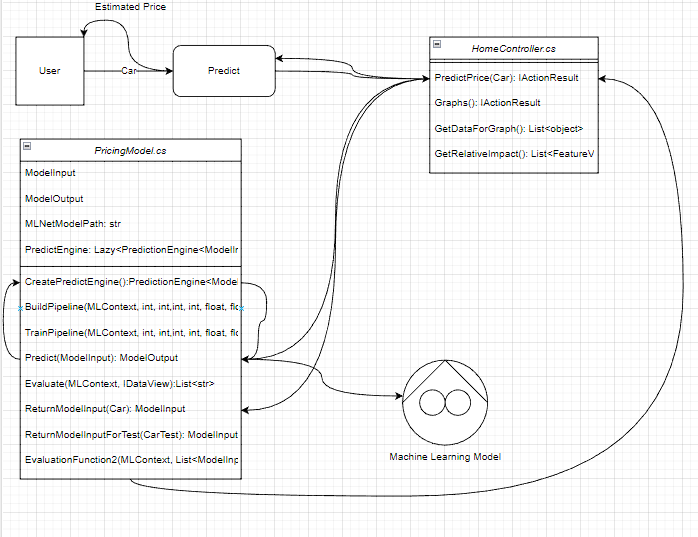


Figure 42 Prediction System

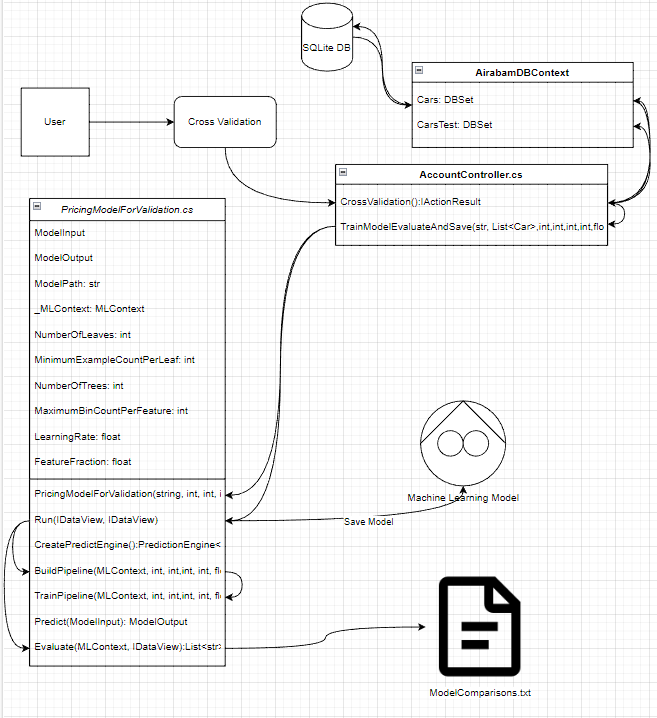


Figure 43 Model Validation System

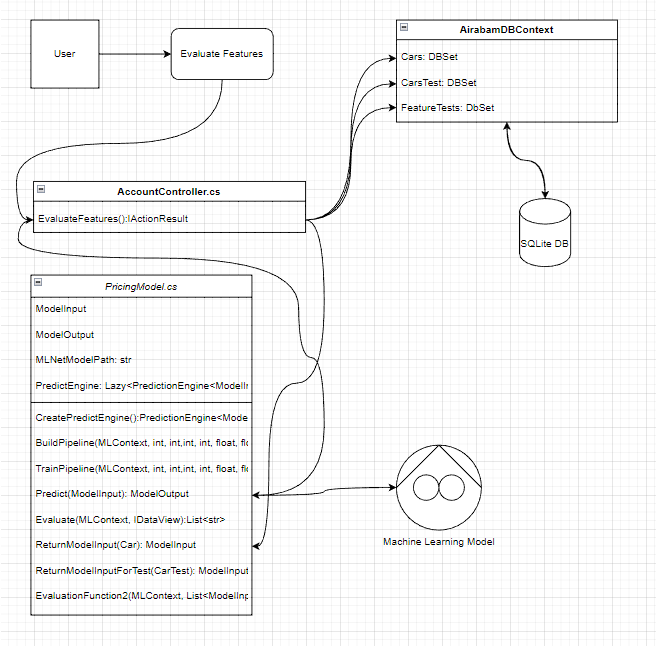


Figure 44 Feature Evaluation System

4. CODE IMPLEMENTATION DETAILS

4.1 Gathering Data for the Research

To have an accurate model we need to feed it data in high quantity and this data should be as realistic and accurate as possible. Then we will use this data to train our model. Now the first step is to select a place where we can find this data. My choice was arabam.com, a pre-owned car dealing website. The reasons of my choice was how easy the website is to crawl using a bot, and the amount of data we can get from this website.

Before getting to the code that allows us to get data from this website, let’s go over what data we want to get first.

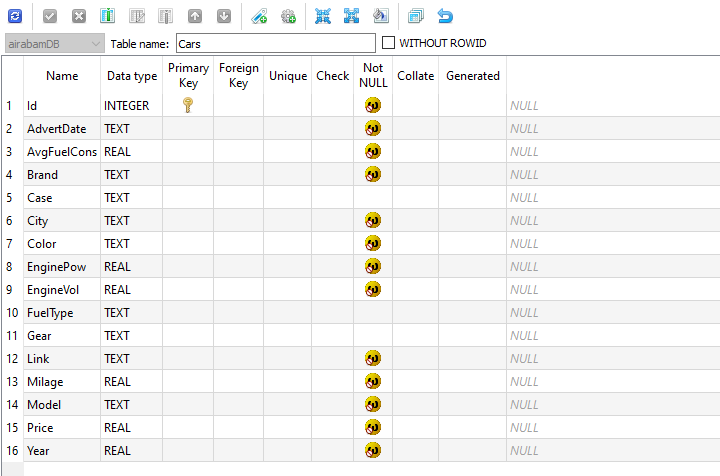


Figure 45 Cars Table - Gathering Data

This above, is the table of Cars. Each car will have:

* Id,
* AdvertDate,
* AvgFuelCons(Average Fuel Consumption)
* Brand,
* Case,
* City,
* Color,
* EnginePow(Engine Power),
* EngineVol(Engine Volume),
* FuelType,
* Gear,
* Link(Link to the advert),
* Milage,
* Model,
* Price,
* Year(Model Year)

Using the Scrape() method below, we will get the brand, model, year, city, color, link, advertdate, milage and price of each car that was posted in the last 14 days.



Figure 46 Scrape Method - Step 1

This is the base skeleton of our function, this function will do the scraping, save the cars to the database and then return to the ‘Profile’. In the next section we will go over the UI of the application, there we can get a better understanding of where this ‘Profile’ takes the user to etc. but for now all we need to focus is what we will write before return statement.

We will add the for loop below to get the cars that were posted in the last 14 days. And then inside the for loop, we will first of all get the response from the url that we create and then get the rows from that response using SelectNodes method.

Then using the counter variable we will create while loop. Counter variable will be our way to navigate in the website. Remember that we are not in control of this process and we do not know how many pages there will be in the last 14 days. This is an issue with this website, they only show last x number of pages as a button, but when you navigate manually using the url, you can actually see there are dozens of more pages. So we will set counter to -1001 when the bot encounters more than 200 cars that are already saved to the database.



Figure 47 Scrape Method - Step 2

Inside the while loop we will enter the code below, that iterates through every car in the available pages until there is no new car and saves the fields we can get from this pages to the database:



Figure 48 Scrape Method - Step 3



Figure 49 Scrape Method - Step 4

However this is not the end of our scraping adventures. This process takes a long time, and the pages we get all these information from does not include some critical information like gear, fuel type, case type, engine volume, engine power and average fuel consumption. To get these we need to go to the link of each advert. We do not do this in the first method because each html page has different fields that we need to handle and doing all this in one go would make the complicated Scrape method more complicated. For this reason we have another function called ScrapeDetails that visits each car’s advert and gets the aforementioned information from the specific page.

After implementing the code below we have every single information we need. Also I need to mention ScrapeTest and ScrapeDetailsTest methods that do the exactly same thing but save the data gathered to a different table called CarTest. This table contains the cars that will be used to test the model.



Figure 50 Scrape Details Method

4.2 Processing the Data

If you were careful enough, you could have noticed that we did not just take the data from the website as it is, instead we made some modifications before saving all of that into the database. Let’s talk about them first.

Most of these are simple conversions like, trimming the string, getting rid of units like ‘hp’, ‘lt’ etc. and converting some of the strings to doubles before saving it to the database. We do not need a milage string, we need a double that represents the milage of the car after all.

Another important modification of data was changing the car model names. The original data has over 4000 different car models. For an example in the original data, Toyota Corolla is not a model, Toyota Corolla 1.6 Vision is a model. This reduces the chances of the model finding a correlation between a model and the price of the car. So the solution I applied was getting rid of anything but the base model name ‘Corolla’. This reduced the numbers of distinct models by 90%. The function below is utilized in the process:



Figure 51 Fix Model Names Method

4.2.1 Feature Normalization

Feature normalization is part of pre processing the data we have before feeding it to the model. But I wanted to talk about this in a different section because ML.NET has very helpful methods that we can use to normalize some features in the process of building the model.

The goal of feature normalization is to increase the accuracy of the model and making its job easy when finding correlations between features and the label, price, in our case by setting a standard on certain features.

Below is the method from PriceModel.cs class where we normalize some features. We will come back to this method in the next chapter when we talk about building a model again, so for now just focus on feature normalization.



Figure 52 BuildPipeline Method - Feature Normalization

4.2.1.1 Categorical One-Hot Encoding:

No matter how deceiving recent projects like ChatGPT can be, machines do not understand human language. We need to translate our sentences for them, so that they can make sense of all this information. Machines understand numbers and we need to give them numbers instead of text.

We talked about categorical values before, and now we need to talk about categorical encoding, the process where we encode categorical values. Categorical values can be encoded using two different methods:

* Ordinal Encoding
* One-Hot Encoding

Let’s say that we have a column where the possible values are “Freshman, Sophomore, Junior, and Senior”. There is a relationship between these values, if you are a senior, you were a junior last year. So we can encode these values as “0,1,2,3” respectively. This is called ordinal encoding, it is more suited to situations where order matters.

But in our case, order does not matter. For an example the Gear column can have three different values:

|  |
| --- |
| Gear |
| Manual |
| Automatic |
| Semi-Automatic |

Table 3 Example Feature Table

And these values can be encoded as:

|  |  |  |  |
| --- | --- | --- | --- |
| Gear | Manual | Automatic | Semi-Automatic |
| Manual | 1 | 0 | 0 |
| Automatic | 0 | 1 | 0 |
| Semi-Automatic | 0 | 0 | 1 |

Table 4 One Hot Encoding Example

This is called One-Hot Encoding. And as you can see we are right now speaking in the mothertongue of the machines, the binary. Using One-Hot Encoding, Manual will be encoded as 100, Automatic will be encoded as 010.

We apply the same procedure for every categorical value we have: Gear, Brand, City, Model, Case, Color and Fuel Type.

4.2.1.2 Replace Missing Values

Many of the adverts have one or two empty columns, after the scraping I got rid of rows that have many empty columns using simple SQL queries but if we got rid of every single row that has an empty column we would have much less data to work with. So we are okay with one or two empty columns, however the machine is not. Machine expects at least the default value for the data type of the input column. Using Transforms.ReplaceMissingValues method we can easily fill empty places with default values. (Example: 0 for integer)

This procedure is applied for: Milage, Year, Average Fuel Consumption, Engine Power and Engine Volume.

4.2.1.3 Concatenate

Right now all these values are separated from each other, we need to concatenate them and present it as a whole to our model. Transforms.Concatenate method achieves this goal.

4.3 Creating the Model

Just to clarify some things, let’s take a look at the steps we need to follow in order to create and use a successfully made machine learning model.

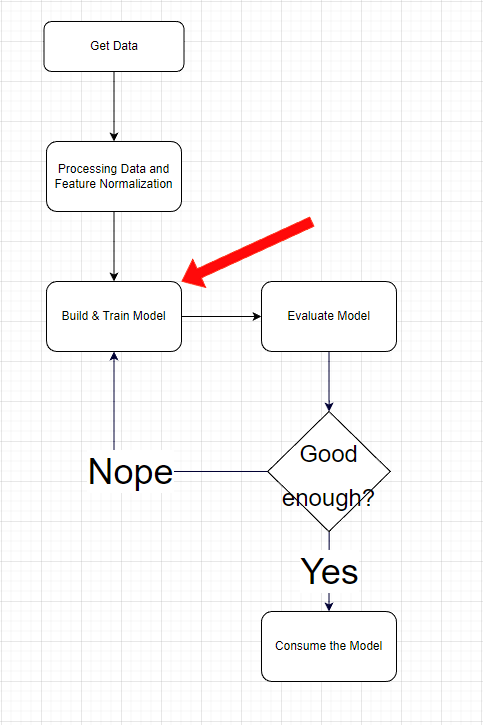


Figure 53 Steps Explained

What we need to do now is to build and train a model. I cover these topics in two different chapters, yet we cannot have a model file if we do not train it first, otherwise it is good for nothing. Therefore these two concepts, build and train are not as seperable as they sound.

Since we are using ML.NET now is a good time to get more familiar with some of its concepts in order to understand the code we write. If we sort of ‘translate’ the diagram above to ML.NET framework we would have something like this:

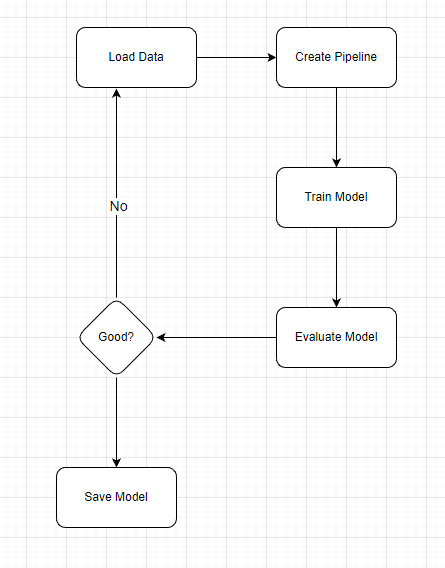


Figure 54 ML.NET Diagram 1

As you can see the Build & Train step is now made of two steps, ‘Create Pipeline’ and ‘Train Model’.

What is a pipeline? Workflows for machine learning are automated using a machine learning pipeline. They function by making it possible to transform and correlate a series of data into a model that can be tested and assessed to provide an outcome, whether positive or negative. (M, 2019)

If you were careful enough in the last chapter you could have noticed that we created a variable named pipeline when doing One-Hot Encoding and other manipulations to the data we have. Every single step in the diagram above is accompanied by different interfaces, classes or methods in the ML.NET framework.

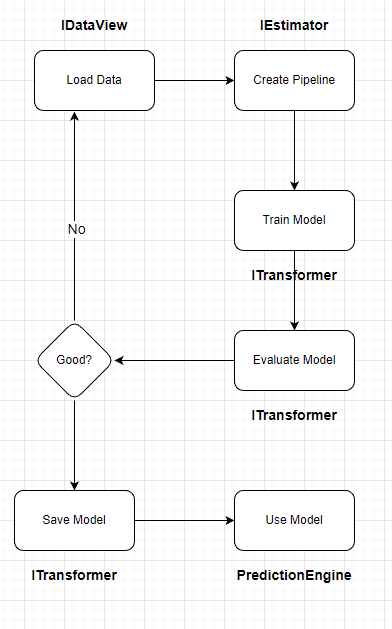


Figure 55 ML.NET Diagram 2

IDataView: According to Microsoft’s documentation on ML.NET IDataView is “the input and output of Query Operators (Transforms). This is the fundamental data pipeline type, comparable to [IEnumerable<T>](https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.ienumerable-1) for LINQ”. (Kershaw) When we work with ML.NET models, we do not pass a list, array or another IEnumerable of inputs to the model, instead we use IDataView to pass the collection of data to the model, whether it is for training, evaluation or consuming does not matter.

IEstimator: Per Microsoft, the estimator (in Spark terminology) is an 'untrained transformer'. It needs to 'fit' on the data to manufacture a transformer. It also provides the 'schema propagation' like transformers do, but over [SchemaShape](https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.schemashape?view=ml-dotnet) instead of [DataViewSchema](https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.dataviewschema?view=ml-dotnet). (Kershaw) It has a very important built-in method; Fit(IDataView). The fit method takes the collection of data as IDataView and trains an ITransformer with this data, then returns it.

ITransformer: To put it all in simple terms this is an interface to transformer which takes data, works on it and returns the processed data. For an example taking the car features and making a prediction on it. Or transforming a text by tokenizing it. ITransformer is one of the most fundamental concepts of ML.NET framework.

PredictionEngine: As Microsoft documentation states PredictionEngine is a “class for making single predictions on a previously trained model (and preceding transform pipeline)”. It has a Predict method, that we will use in order to predict the prices of the cars.

MLContext: This object is used to read data, create estimators, transformers, save models, load them, evaluate, predict and perform all other tasks.

Now is time to look at our PredictModel.cs class to see how we implemented these concepts in our project.

At the top of the class there are two classes that have self-explanatory names: ModelInput and ModelOutput.



Figure 56 ModelInput and ModelOutput classes

We will use these classes to explain our model what we will give it to as an input and what we expect as an output.

Following these two classes there are two fields ‘MLNetModelPath’ the path, where our model will be saved to and loaded from, and a PredictEngine.



Figure 57 PricingModel.cs - 1

We will cover the CreatePredictEngine a couple lines after the declaration of PredictEngine variable, but it is important to note that we are using Lazy<T> in the process. This enables us to instantiate something when it is the first time it will be used. Therefore the cost of creating it until then is avoided.

After these two fields we have our CreatePredictEngine method, which is used to initialize PredictEngine variable above.



Figure 58 PricingModel.cs - 2

In this method first we create an instance of MLContext. Then using this instance we load our saved model, and then create an ITransformer to hold this saved model. Last step is using the CreatePredictionEngine method that will create a prediction engine for one-time prediction based on given input and output classes.

This method is followed by BuildPipeline method:



Figure 59 PricingModel.cs - 3

We talked about this method before. But we left out the last Append() and the part after that where we select our trainer algorithm and return the pipeline. We use this method in the TrainPipeline method below, so we will go over the return part over there. But we should first focus on the options we give and what they mean. Per Microsoft documentation:

* NumberOfLeaves: This parameter controls the maximum number of leaf nodes in a decision tree. A leaf node is a terminal node in the tree, which represents a region in the input space that is assigned a constant value. The number of leaves controls the complexity of the model, and a larger number of leaves will lead to a more complex model with a higher risk of overfitting.
* MinimumExampleCountPerLeaf: This parameter controls the minimum number of examples that are required to be present in a leaf node. If a leaf node contains fewer examples than the minimum specified, the tree will not split further at that node. This parameter is useful in controlling overfitting, as it ensures that each leaf node contains a sufficient number of examples to make a reliable prediction.
* NumberOfTrees: This parameter controls the number of decision trees that are used in the ensemble. A larger number of trees will lead to a more complex model, which can increase the risk of overfitting. However, it also increases the chances of finding the correct global minimum, and improves the generalization of the model.
* MaximumBinCountPerFeature: This parameter controls the maximum number of bins that are used to discretize continuous features. Binning is the process of grouping a set of continuous values into a smaller number of discrete "bins" or intervals. A larger number of bins will result in a more fine-grained discretization of the features and can increase the model's ability to capture subtle patterns in the data, but may also increase the risk of overfitting.
* LearningRate: This parameter controls the step size of the gradient descent algorithm used to optimize the model. A smaller learning rate will result in smaller steps and a longer optimization process, while a larger learning rate will result in larger steps and a shorter optimization process. This parameter can be used to control the trade-off between the speed of convergence and the risk of overshooting the global minimum.
* FeatureFraction: This parameter controls the fraction of features that are used in each decision tree. A smaller feature fraction will lead to a simpler model, while a larger feature fraction will lead to a more complex model. Using a smaller feature fraction can reduce the risk of overfitting, but may also decrease the model's ability to capture subtle patterns in the data.
* LabelColumnName: The column to predict.
* FeatureColumnName: The column to base the prediction upon.

Then comes the TrainPipeline method:



Figure 60 PricingModel.cs - 4

This method takes in an MLContext object and the data to train the model with. First it creates a pipeline using the BuildPipeline method above, and using this pipeline we fit the data and create our model. This model can be then, saved, used or discarded as we choose.

To predict we use the method below:



Figure 61 PricingModel.cs – 5

This method uses the PredictEngine field’s value, and then uses its extension method .Predict() based on the model input provided, in our case this will be a car without a determined price and returns a ModelOutput, which is the class that holds float Score, in other words price of the car.

So now we know we have, a method for training, predicting etc. but how will we evaluate this model that we built?



Figure 62 PricingModel.cs – 6

Using this method we can load our model from the path it is stored and then use MLContext.Regression.Evaluate, providing it with some testData and it will give us important metrics like RSquared Score, Root Mean Squared Error, MeanSq Error Score.

Or we can use another method called EvaluationFunction2 that iterates over a test data and predicts each car, and prints out the average margin of error.



Figure 63 PricingModel.cs – 7

In this class there are two more methods that we use. We have two MVC models, Car and CarTest that interact with our SQLite database tables. But we can’t use these model classes directly in our PredictEngine class, these MVC models are used to get data from user, validate the input data and save it to the database, not for interacting with the ML model we trained. So we have two methods that convert instances of these two models into ModelInput classes.



Figure 64 PricingModel.cs - 8

Now let’s see how we use this class in action.

4.4 Training the Model

We already went over each and every part of PredictModel class. Now we need to use its methods to create and train a model that we can use.



Figure 65 TrainPrice Method

This code is part of the AccountController.cs controller, this website contains only one user, the admin. The admin can go ahead and use these methods to train the model again, evaluate etc.

This method first gets all the cars from the database and then converts them to a List of ModelInput instances, converts this list to an IDataView, then calls the TrainPipeline method to train a model, then saves this model using MLContext.Model.Save() method.

4.5 Finding the Optimal Values For Fast Tree Algorithm

In earlier chapters Fast Tree Algorithm and its parameters in the ML.NET framework were discussed. Setting wrong values for these parameters may result in overfitting or underfitting our machine learning model. But what are the optimal values for fast tree algorithm?

The optimal parameter settings for a fast tree algorithm depend on many factors, such as the structure of our data, the size of our dataset, and the specific problem we are trying to solve. Therefore, there is no one-size-fits-all answer to this question.

Each dataset and the problem it represents is unique. Therefore there is no way we can know which values are best for our model. However using cross validation we can test and find out which of the values we can come up with work better with our model.

Cross-validation is a technique used in machine learning to evaluate the performance of a model. It involves dividing the data into several subsets, and then training and testing the model on different subsets of the data. The goal of cross-validation is to provide an estimate of the model's performance on unseen data, which can be used to compare different models or to tune the parameters of a model.

There are several different types of cross-validation, including:

* + - * K-fold cross-validation
      * Leave-one-out cross-validation
      * Stratified cross-validation
      * Repeated k-fold cross-validation

The approach type of cross-validation I applied in this project is leave-one-out cross-validation. This is a special case of k-fold cross-validation where k is set to the number of samples in the dataset. This means that the model is trained on all but one of the samples and tested on the remaining sample.

We now need to come up with values for the parameters, and create and test a number of machine learning models then pick the most successful one. These models will be trained on the entire Cars table, and their performances will be evaluated on the CarsTest table. The values selected are based upon the documentation of LightGBM and my personal creativity.

These models are trained using the PricingModelForEvaluation class instead of PricingModel class. The reason behind it is simple. PricingModelForEvaluation class is written in a way that once an instance of this class is created, it will keep state on many things unlike the original PricingModel class, for an example the model path.

This approach begs the question: Why not use this class in other parts of the project? Because, machine learning is not cheap. It takes a long time to train a model, evaluate it and predict outcomes, and using static methods are a better way to use the limited resources we have rather than instantiating a class containing the methods. (Gray, 2007)

Apart from how we interact with these two classes, some other differences they have are that PricingModelForEvaluation class contains several fields, a constructor, and a method called Run, which trains a model, evaluates it, logs the performance and saves the model in one call.



Figure 66 PricingModelForValidation.cs – 1

This method takes the parameters of FastTree algorithm as parameters, then creates an instance of the PricingModelForValidation class, and calls the Run method of that class. This method is called in the CrossValidation IActionResult like this:



Figure 67 CrossValidation Method

Below are the ten different models that have different parameters and their performances:

| Model | NOL | MECPL | NOT | MBCPF | LR | FF | RSquared Score | Root Mean Squared Error |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ModelA | 100 | 10 | 100 | 100 | 0.15 | 0.7 | 0.44 | 344694.15 |
| ModelB | 200 | 5 | 1000 | 300 | 0.1 | 0.8 | 0.44 | 344694.96 |
| ModelC | 300 | 20 | 5000 | 1000 | 0.2 | 0.7 | 0.43 | 348374.9 |
| ModelD | 600 | 2 | 32768 | 600 | 0.2 | 0.75 | 0.43 | 347588.09 |
| ModelE | 1000 | 10 | 20000 | 500 | 0.2 | 0.8 | 0.44 | 346060.89 |
| ModelF | 485 | 2 | 32768 | 600 | 0.2 | 0.85 | 0.43 | 347231.11 |
| ModelG | 500 | 4 | 6000 | 750 | 0.14 | 0.9 | 0.44 | 346152.09 |
| ModelH | 385 | 2 | 2500 | 100 | 0.25 | 0.68 | 0.43 | 348407.21 |
| ModelI | 475 | 2 | 32768 | 600 | 0.19 | 0.75 | 0.45 | 342657.32 |
| ModelJ | 485 | 2 | 400 | 600 | 0.2 | 0.75 | 0.43 | 347582.62 |

Table 5 Model Comparisons

To understand the performances of these different models we need to understand two metrics that are used to track performances of regression models; RSquared Score and Root Mean Squared Error.

R-squared is a commonly used evaluation metric in regression problems that measures the proportion of variation in the response variable that can be explained by the predictor variables in a model. It is a value between 0 and 1, where 1 indicates that the model fits the data perfectly and 0 indicates that the model does not fit the data at all.

In other words, R-squared measures how well the model fits the observed data. The higher the R-squared, the better the model fits the data, and the lower the residuals.

It's important to note that just because a model has a high R-squared, it doesn't necessarily mean that the model is a good fit for the data. A high R-squared can also indicate that the model is overfitting the data, or that the predictor variables in the model are not independent of one another.

Root Mean Squared Error (RMSE) is a commonly used metric to evaluate the performance of a regression model. RMSE measures the average magnitude of the differences between the actual and predicted values, i.e., the average of the squares of the residuals (prediction errors).

The lower the RMSE value, the better the fit of the model to the data. In other words, a lower RMSE indicates that the model is making fewer and smaller errors in its predictions. A high RMSE, on the other hand, means that the model is making large prediction errors and is not capturing the patterns in the data accurately.

Considering these two metrics, we can see Model I, which has the highest RSquared Score and the lowest RMSE stands out among the ten models we created. These metrics tell us a lot, but if we want to see the results in a more tangible way using the test data, we can use the Evaluate Price method we built earlier.

We save the Model I as Modella.zip, and run the Evaluate Price method that can be accessed from the AccountController.cs. The results are promising:

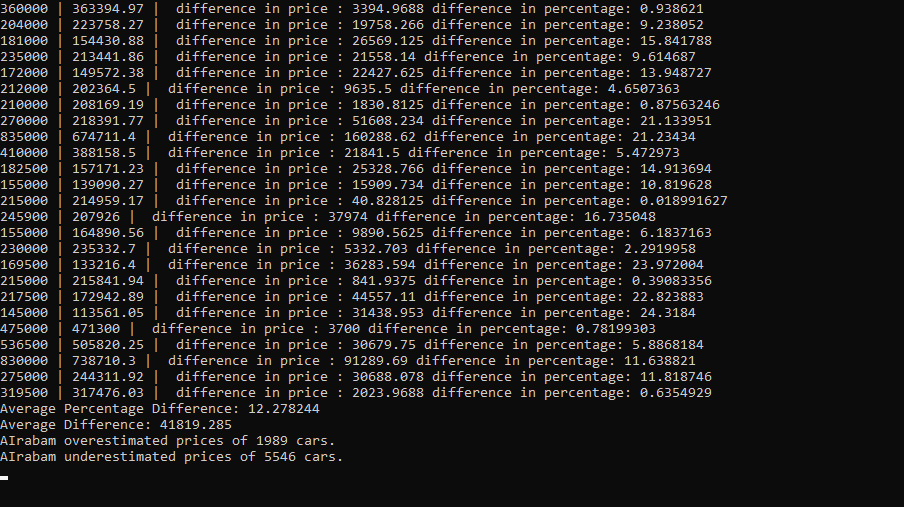


Figure 68 Average Difference Between Prediction and Real Price

Average difference of the actual price of a car and the price predicted by the model is around 42 thousand Turkish Liras. And the percentage difference is 12.278244 percent.

4.6 Evaluating Features

The model can predict prices of a car based on its features. This also means that we can use our machine learning model to have a better understanding of the correlation between prices of cars and certain values in their features.

The method below gets the top ten most used models, then creates a collection of example cars using the exampleCarsForFeatureImportance() method. After that using the GetFeaturePriceDictionary() method it creates a list of FeatureTest objects that will be saved to the FeatureTests table. These two helper methods are explained in more detail below.



Figure 69 EvaluateFeatures Method

The method below uses the top 10 most common models to create one example car for each model, using the most common attributes for the cars that share the same model.



Figure 70 Method That Creates Example Cars

After returning example cars, these example cars are sent to the GetFeaturePriceDictionary() method in the EvaluateFeatures() method. Which is the method that takes every possible value for every feature and then iterates over the example cars collection, creates a new car by changing every single feature to each possible value for that feature. This creates a new car instance which is then sent to the prediction model to predict the price of the car. Then we compare the price of the car, with the predicted price. We also compare the predicted price for the original car and for the car that has a modified feature. Then average of these two margins are calculated. All these predictions are held in an instance of FeatureValueOutput class and then added to a list. This list is then returned to the EvaluateFeatures() method so that they are saved to the database.

All these data are calculated and visualized in Graphs view. The findings from these evaluations will be discussed in the next chapter.

5. OUTCOMES AND IMPLICATIONS

After evaluating the features, we can go to the ‘Graphs’ section to view the results of evaluation. Below, you can find visual representations of the evaluations made and some explanations and assumptions related to them.

5.1 Top 10 and Last 10 Brands

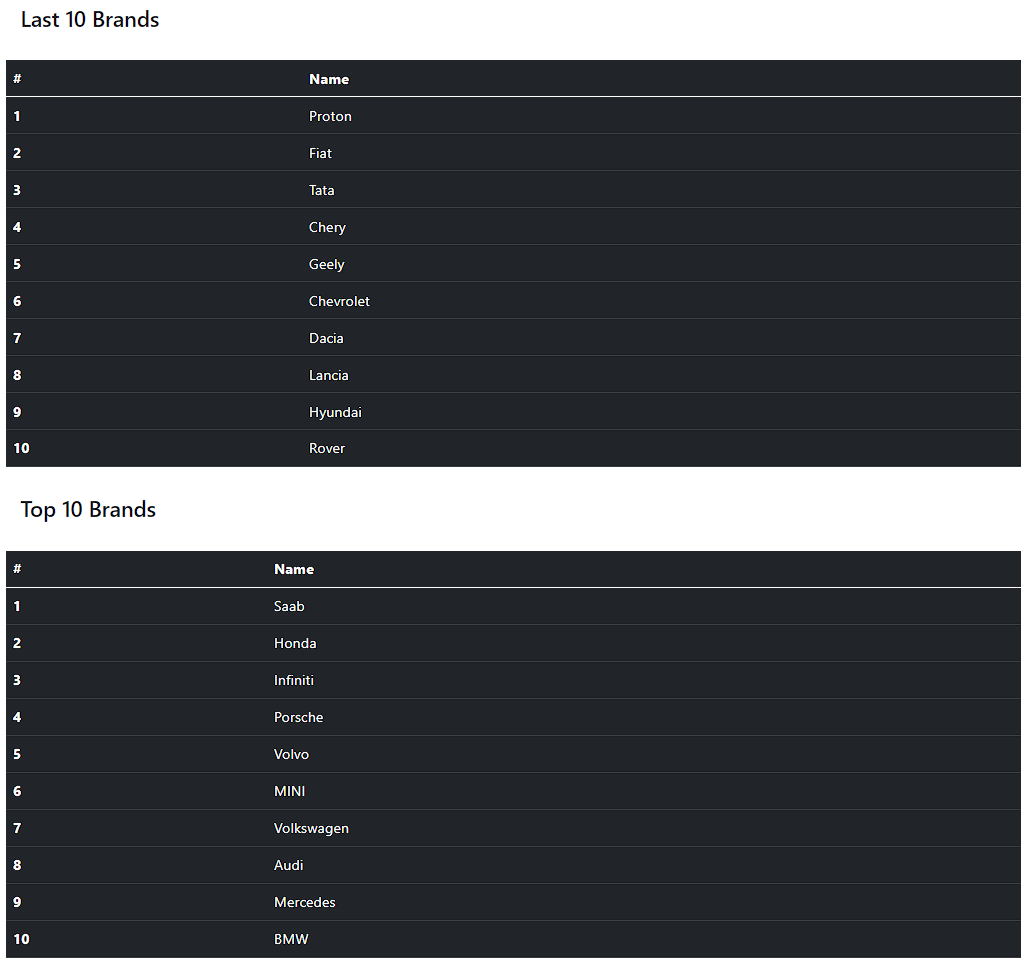


Figure 71 Last 10 and Top 10 Brands

5.2 Top 10 and Last 10 Models

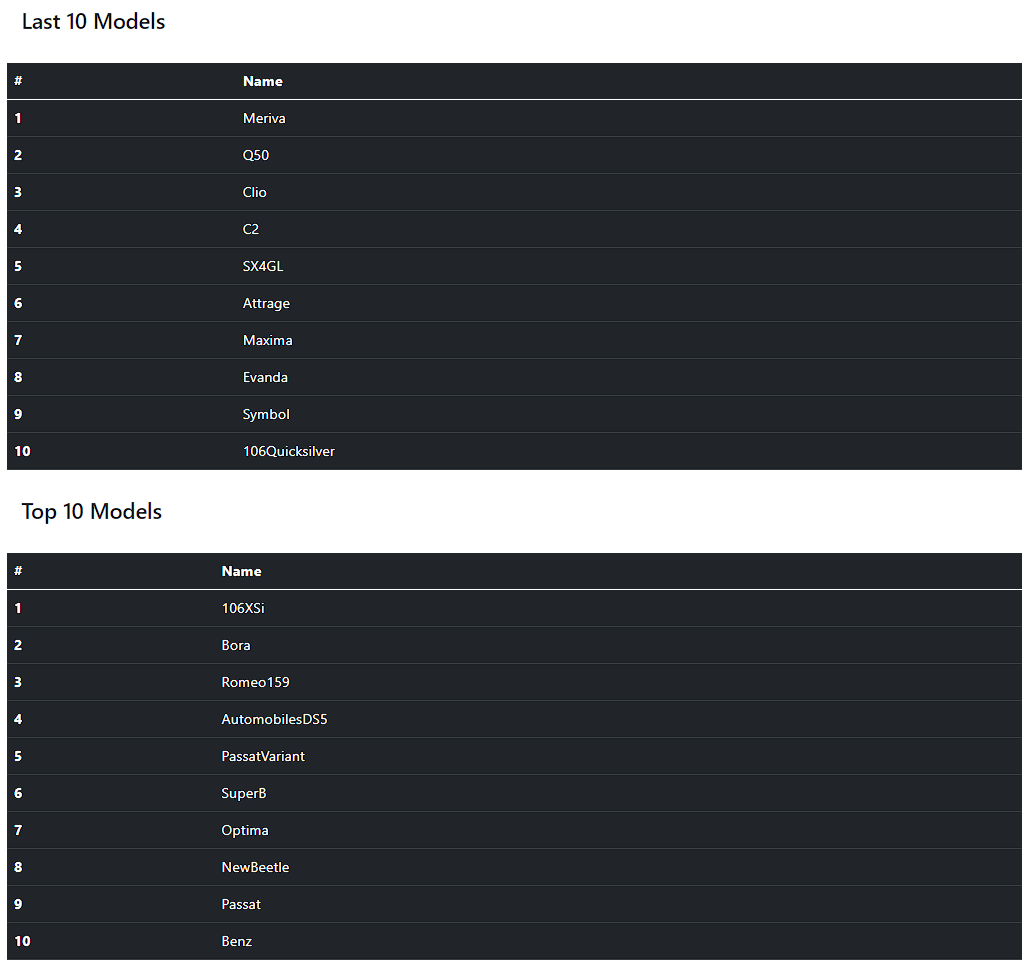


Figure 72 Last 10 and Top 10 Models

5.3 Impact of Average Fuel Consumption on the Price

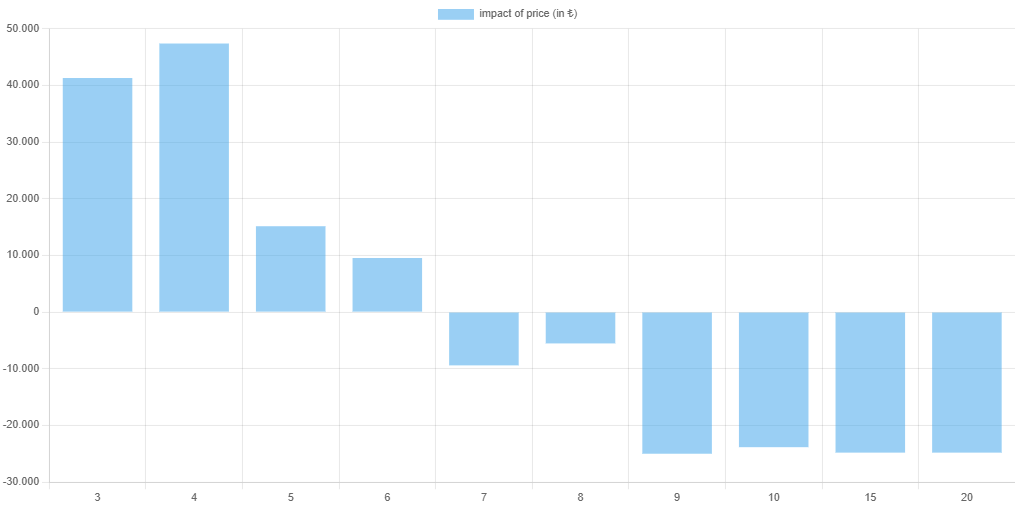


Figure 73 Average Fuel Consumption’s Impact on the Price

As expected as the average fuel consumption increase, cars become less and less appealing. I truly wonder if it is the same for a country where fuel is more affordable.

5.4 Impact of Car Case on the Price

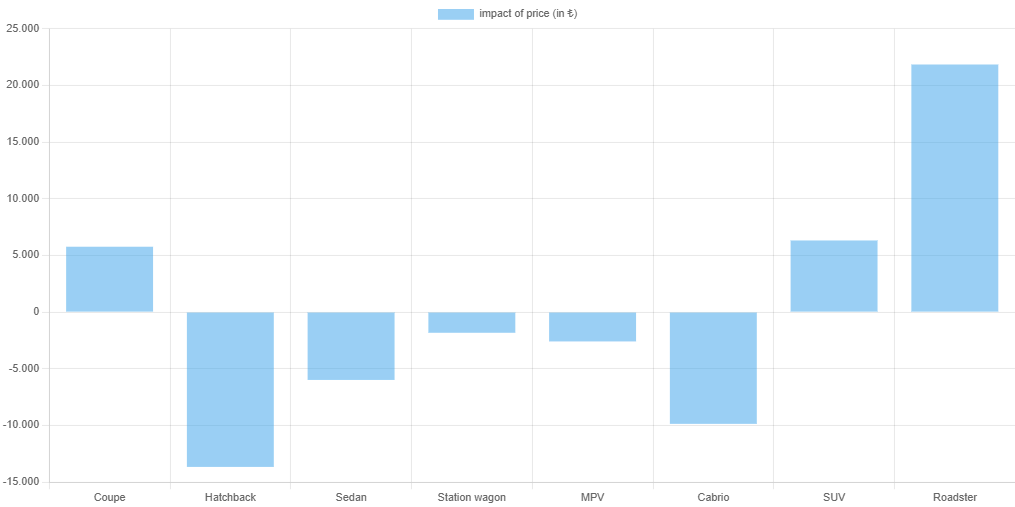


Figure 74 Impact of Car Case on the Price

AIrabam suggests that a Coupe, SUV and Roadster are sold at higher prices when compared to more common case types like Hatchback, Sedan and Station-Wagon. This is not very surprising as vehicles like SUVs are considered a little bit more on the luxurious side of the vehicles. They also typically consume more fuel and the owner needs to pay more taxes.

5.5 Impact of Color on the Price

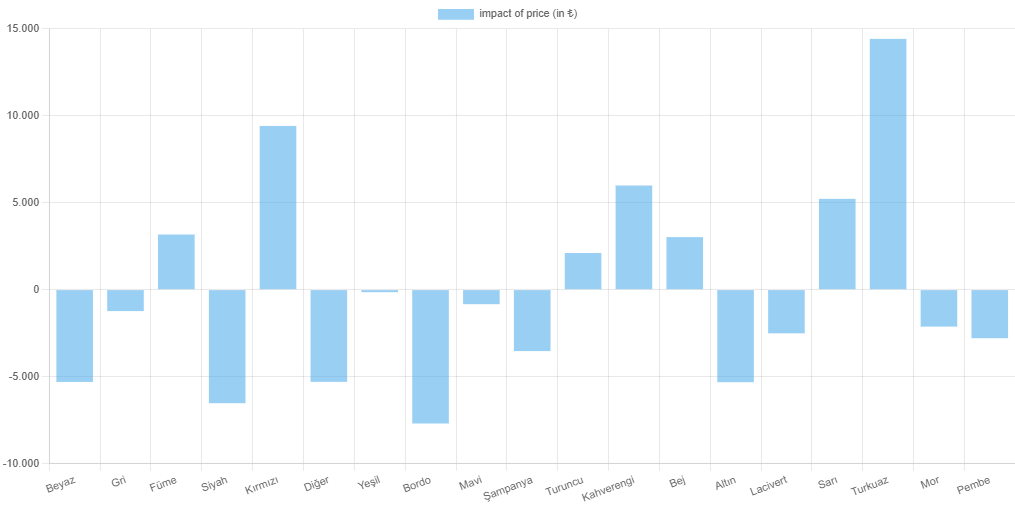


Figure 75 Impact of Color on the Price

According to our model people value a car with colors of turquoise, brown, yellow, red and green more while they don't do the same for cars that are black, pink, gray or golden.

5.6 Impact of Gear Type on the Price

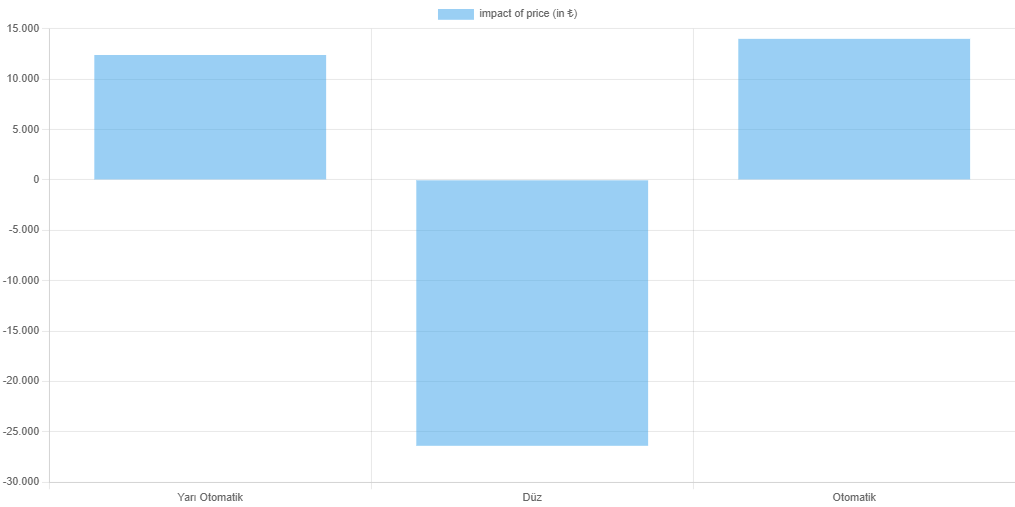


Figure 76 Impact of Gear Type on the Price

This is easily the most predictable result we have on this page. AIrabam suggests that manual cars are less valued by people of Turkey, and they would prefer to get an automatic or a semi-automatic car.

5.7 Impact of City on the Price

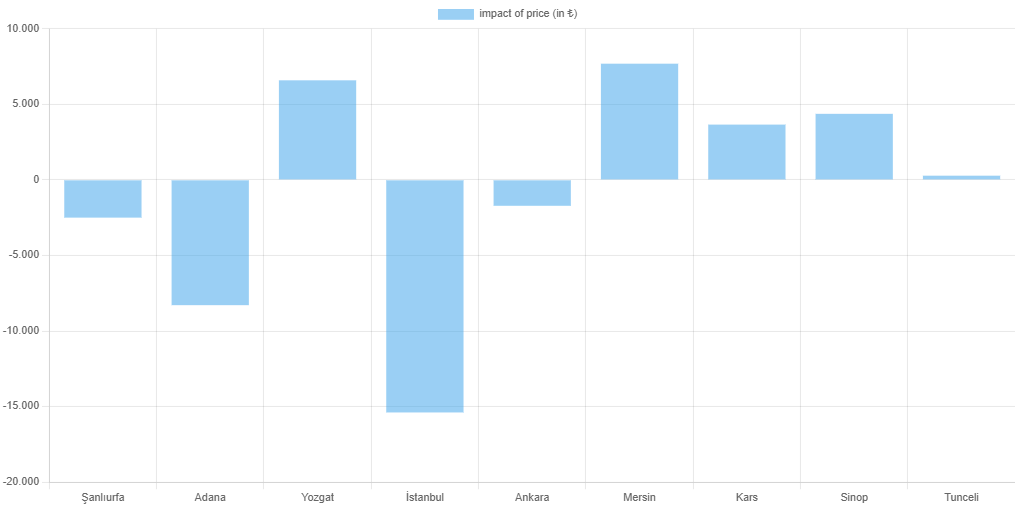


Figure 77 Impact of City on the Price

This is easily the most surprising result we have from this research. Since big cities like İstanbul, Ankara, Adana generally have a higher cost of living, it is very intuitive to think that the car prices would also be higher. But on the contrary since there are a lot more car galleries, and dealerships in these cities it seems like it effects the car price in an unexpected way. The prices are lower in these cities. On the contrary in smaller cities like Kars, Yozgat and Tunceli due to not having many competition over selling cars it seems like the sellers usually go for a higher price.

5.8 Impact of Fuel Type on the Price

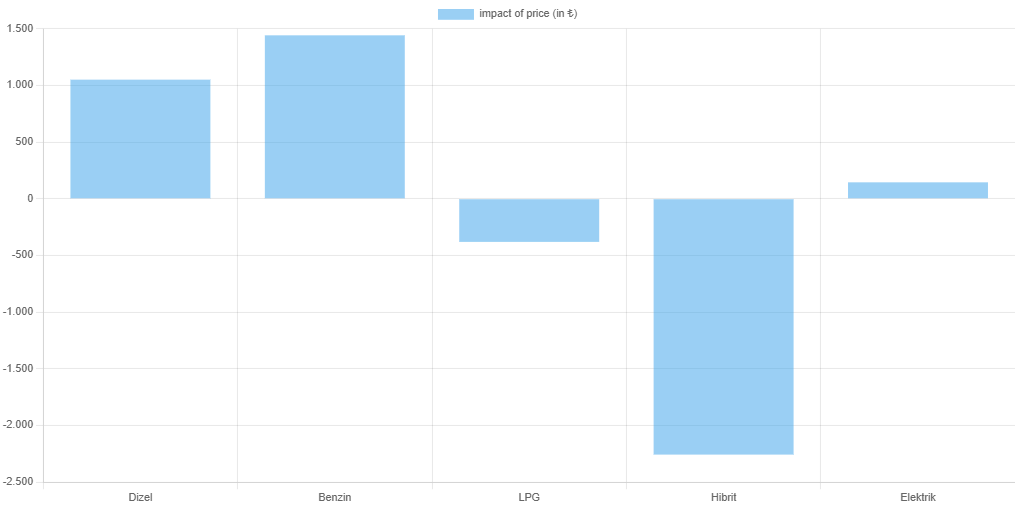


Figure 78 Impact of Fuel Type on the Price

Even though they are bad for the environment diesel cars in fact do cost less to their owner and as you can see it is reflected in the graph as well. One surprising bit from this graph was how little the price got effected when the fuel type is LPG. I would have expected a big drop. This is actually very telling example of how broken the market in Turkey is due to the current inflation and devaluation of the currency. Another surprising bit is that there seems to be a bias against hybrid cars in Turkey, while market holds electrical cars at a high place.

6. CONCLUSION

Having a sufficient quantity of data is crucial for building effective machine learning models. The more data the model has access to, the more accurately it can learn patterns and make predictions. However, it's not just about having a large amount of data, but also about having high-quality data. Data that is irrelevant, inconsistent or biased can negatively impact model performance and lead to incorrect predictions. Therefore, it's important to ensure both the quantity and quality of data in order to build robust and accurate machine learning models.

The second-hand car market is highly volatile and can experience sudden changes and fluctuations in prices. This volatility is due to several factors such as the supply and demand dynamics, seasonal fluctuations, the launch of new models, and economic conditions. The market can also be affected by external factors such as natural disasters, government regulations, and technological advancements. As a result of this volatility, it can be difficult to predict the prices of second-hand cars.

The volatility of the second-hand car market presents a challenge for finding quality data to train a machine learning model that predicts car prices. If the data used to train the model does not accurately reflect the current market conditions, the predictions made by the model may be unreliable. Additionally, if the data used for training the model is limited in scope or outdated, it may not capture the full range of factors that influence car prices. As a result, the model may not be able to make accurate predictions. To overcome this challenge, it's important to ensure that the data used for training the machine learning model is diverse, up-to-date, and reflective of the current market conditions. This will ensure that the model is able to make reliable predictions and support better decision making in the second-hand car market.

In building a machine learning model that predicts prices of cars, I took into account the challenges discussed above. I must emphasize that, the task of finding quality data in large quantities was made even more difficult by the constant shifts and changes in the market. Despite these difficulties, I was able to develop a model that had a sixty thousand Turkish Liras margin of error, demonstrating its effectiveness. This was achieved through careful consideration of these challenges and using the right tools and techniques to overcome them. The result is a model that is able to accurately predict car prices in the current state of the market.

7. REFERENCES

Pati, S. (2021, August 3). The Difference Between Artificial Intelligence and Machine Learning. Analytics Insight. Retrieved January 5, 2023, from https://www.analyticsinsight.net/the-difference-between-artificial-intelligence-and-machine-learning/#:~:text=Artificial%20intelligence%20is%20a%20technology,humans%20to%20solve%20complex%20problems.

Kershaw, N. (n.d.). FastTreeRegressionTrainer.options class (Microsoft.ml.trainers.FastTree). (Microsoft.ML.Trainers.FastTree) | Microsoft Learn. Retrieved January 7, 2023, from https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fasttreeregressiontrainer.options?view=ml-dotnet

Kershaw, N. (n.d.). IDataView interface (Microsoft.ML). (Microsoft.ML) | Microsoft Learn. Retrieved November 24, 2022, from https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.idataview?view=ml-dotnet

Kershaw, N. (n.d.). IEstimator interface (Microsoft.ML). IEstimator Interface (Microsoft.ML) | Microsoft Learn. Retrieved January 1, 2023, from https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.iestimator-1?view=ml-dotnet

LightGBM. (n.d.). Parameters tuning. Parameters Tuning - LightGBM 3.3.5.99 documentation. Retrieved January 1, 2023, from https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html

M, S. (2019, December 13). What is a pipeline in machine LEARNING?HOW to create one? Medium. Retrieved January 13, 2023, from https://medium.com/analytics-vidhya/what-is-a-pipeline-in-machine-learning-how-to-create-one-bda91d0ceaca

Mandot, P. (2018, December 1). What is LIGHTGBM, how to implement it? how to fine tune the parameters? Medium. Retrieved December 23, 2022, from https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

Kershaw, N. (n.d.). FastTreeRegressionTrainer class (Microsoft.ml.trainers.FastTree). (Microsoft.ML.Trainers.FastTree) | Microsoft Learn. Retrieved January 3, 2023, from https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fasttreeregressiontrainer?view=ml-dotnet

Shukla, S. (2023, January 10). Regression and classification: Supervised machine learning. GeeksforGeeks. Retrieved January 13, 2023, from https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/

IMSL. (2021, June 16). What is a regression model? IMSL by Perforce. Retrieved December 12, 2022, from https://www.imsl.com/blog/what-is-regression-model#:~:text=A%20regression%20model%20provides%20a,by%20a%20linear%20regression%20model.

Gray, J. (2007, August). *Writing faster managed code: Know what things cost*. Microsoft Learn. Retrieved February 1, 2023, from https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/ms973852(v=msdn.10)?redirectedfrom=MSDN