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

1.1 Tools Used in the Project 10

1.2 Project Development Timeline 11

1.3 System Model 12

2. KEY THEORETICAL CONCEPTS 13

2.1 Artificial Intelligence and Machine Learning 13

2.2 Supervised and Unsupervised Learning 13

2.3 Regression Problems and Models 14

2.4 Categorical and Numerical Values 14

2.5 Fast Tree Regression Algorithm 14

2.5.1 Decision Trees 15

2.5.2 Gradient Boosting 16

2.5.3 Histogram Binning 17

3. PROJECT DESIGN 18

3.1 An Overview of the Folder Structure and Project Files 18

3.2 Design of the Database 19

3.3 User Interface 23

3.4 Design of Classes 28

3.5 Interactions Between Classes 35

4. CODE IMPLEMENTATION DETAILS 40

4.1 Gathering Data for the Research 40

4.2 Processing the Data 47

4.2.1 Feature Normalization 48

4.2.1.1 Categorical One-Hot Encoding: 49

4.2.1.2 Replace Missing Values 50

4.2.1.3 Concatenate 50

4.3 Creating the Model **51**

4.4 Training the Model **62**

4.5 Finding the Optimal Values For Fast Tree Algorithm **62**

4.6 Evaluating Features **67**

5. OUTCOMES AND IMPLICATIONS **71**

5.1 Top 10 and Last 10 Brands **71**

5.2 Top 10 and Last 10 Models **72**

5.3 Impact of Average Fuel Consumption on the Price **73**

5.4 Impact of Car Case on the Price **73**

5.5 Impact of Color on the Price **74**

5.6 Impact of Gear Type on the Price **74**

5.7 Impact of City on the Price **75**

5.8 Impact of Fuel Type on the Price **76**

6. CONCLUSION **77**

LIST OF TABLES

Table 1 Tools Used in the Project 11
Table 2 Project Development Timeline 11
Table 3 Example Feature Table 49
Table 4 One Hot Encoding Example 49
Table 5 Model Comparisons 65

LIST OF FIGURES

Figure 1. System Model 12
Figure 2 Folder Structure 18
Figure 3 __EFMigrationsHistory Table Structure 19
Figure 4 __EFMigrationsHistory Table Data Sample 19
Figure 5 Cars Table Structure 20
Figure 6 Cars Table Data Sample 20
Figure 7 CarsTest Table Structure 21
Figure 8 CarsTest Table Data Sample 21
Figure 9 FeatureTests Table Structure 22
Figure 10 FeatureTests Table Data Sample 22
Figure 11 Users Table Structure 23
Figure 12 Users Table Data Sample 23
Figure 13 Home Page - 1 23
Figure 14 Home Page – 2 24
Figure 15 Predict Price Page – 1 24
Figure 16 Predict Price Page – 2 24
Figure 17 Predict Price Page – 3 25
Figure 18 Predict Price Page – 4 25
Figure 19 Graphs Page – 1 26
Figure 20 Graphs Page – 2 26
Figure 21 Graphs Page – 3 27
Figure 22 Admin Page 27
Figure 23 UserRepository.cs 28
Figure 24 AirabamDBContext.cs 28
Figure 25 CookieUser.cs 29
Figure 26 CookieUserItem.cs 29
Figure 27 Hasher.cs 29
Figure 28 UserManager.cs 30
Figure 29 LoginVm.cs and RegisterVm.cs 30
Figure 30 Car.cs and CarTest.cs 31
Figure 31 FeatureValueAverage.cs 31

Figure 32 FeatureValueOutput.cs 32
Figure 33 ErrorViewModel.cs 32
Figure 34 HomeController.cs 32
Figure 35 AccountController.cs 33
Figure 36 PricingModel.cs 34
Figure 37 PricingModelForValidation.cs 34
Figure 38 Authentication System 35
Figure 39 Model Training System 35
Figure 40 Scraping System 36
Figure 41 Evaluation System 37
Figure 42 Prediction System 37
Figure 43 Model Validation System 38
Figure 44 Feature Evaluation System 39
Figure 45 Cars Table - Gathering Data 40
Figure 46 Scrape Method - Step 1 42
Figure 47 Scrape Method - Step 2 42
Figure 48 Scrape Method - Step 3 43
Figure 49 Scrape Method - Step 4 44
Figure 50 Scrape Details Method 46
Figure 51 Fix Model Names Method 47
Figure 52 BuildPipeline Method - Feature Normalization 48
Figure 53 Steps Explained 51
Figure 54 ML.NET Diagram 1 52
Figure 55 ML.NET Diagram 2 53
Figure 56 ModelInput and ModelOutput classes 55
Figure 57 PricingModel.cs - 1 56
Figure 58 PricingModel.cs - 2 56
Figure 59 PricingModel.cs - 3 57
Figure 60 PricingModel.cs - 4 58
Figure 61 PricingModel.cs – 5 59
Figure 62 PricingModel.cs – 6 59
Figure 63 PricingModel.cs – 7 60
Figure 64 PricingModel.cs - 8 61

Figure 65 TrainPrice Method 62
Figure 66 PricingModelForValidation.cs – 1 64
Figure 67 CrossValidation Method 65
Figure 68 Average Difference Between Prediction and Real Price 67
Figure 69 EvaluateFeatures Method 68
Figure 70 Method That Creates Example Cars 69
Figure 71 Last 10 and Top 10 Brands 71
Figure 72 Last 10 and Top 10 Models 72
Figure 73 Average Fuel Consumption's Impact on the Price 73
Figure 74 Impact of Car Case on the Price 73
Figure 75 Impact of Color on the Price 74
Figure 76 Impact of Gear Type on the Price 74
Figure 77 Impact of City on the Price 75
Figure 78 Impact of Fuel Type on the Price 76

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 dynamic websites**
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**

Technology	Description
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

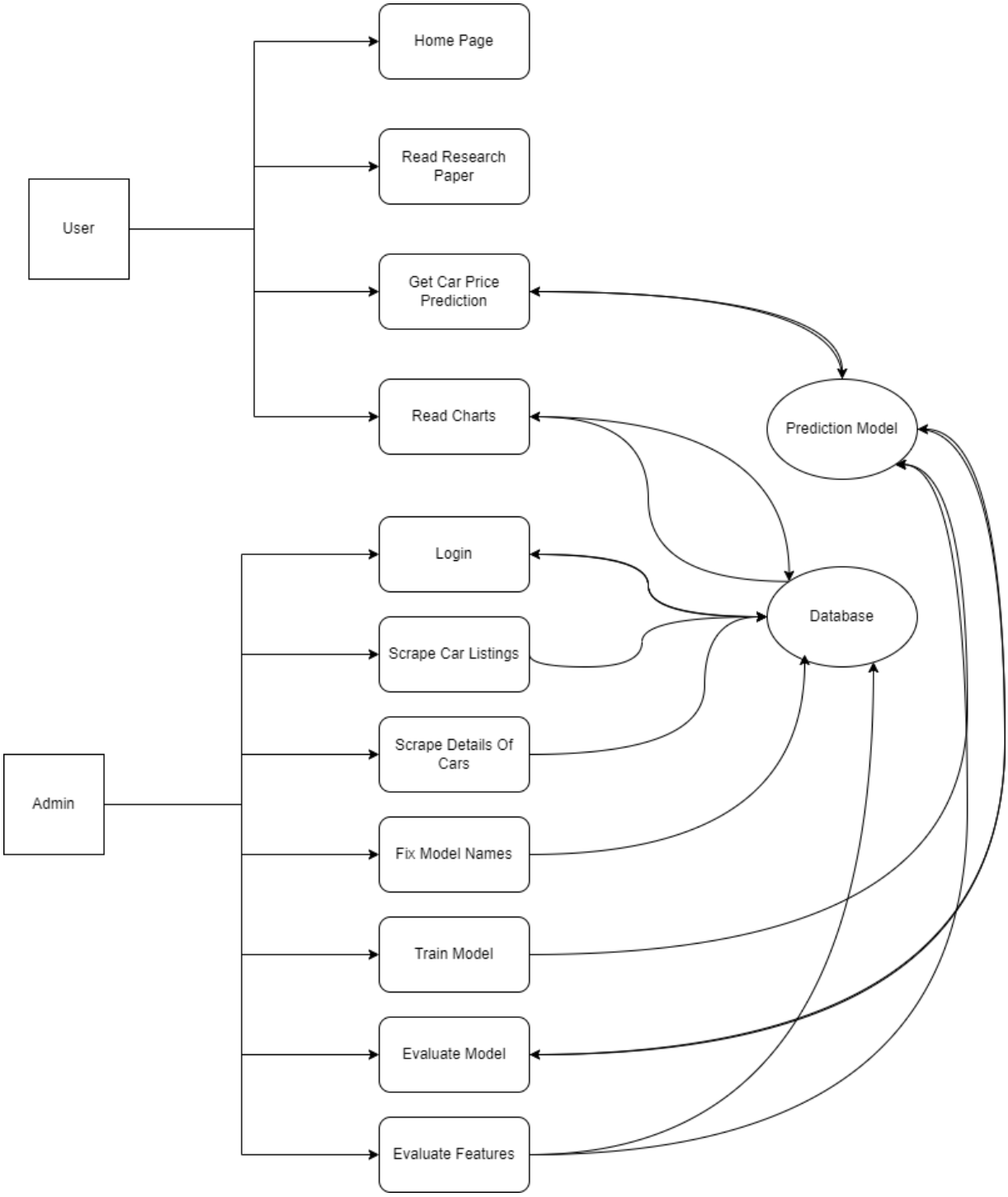


Figure 1. System Model

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. A subset of artificial intelligence called "machine learning" enables a system to automatically learn from prior data without explicit programming. AI aims to create intelligent computer systems that can tackle challenging issues like humans do. (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 goal is for the model to find patterns or structure in the input data. Some examples of unsupervised learning tasks include clustering and dimensionality reduction.

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

A regression problem is when the output variable is a real or continuous value, such as “salary” or “weight”. Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points. (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 a function that describes 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 decision tree algorithm is a type of machine learning algorithm that is used to build models for classification and regression tasks. 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 basic idea behind gradient boosting is to train a series of simple 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.

The process is repeated for a fixed number of iterations, or until the performance of the model on a validation set stops improving. 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

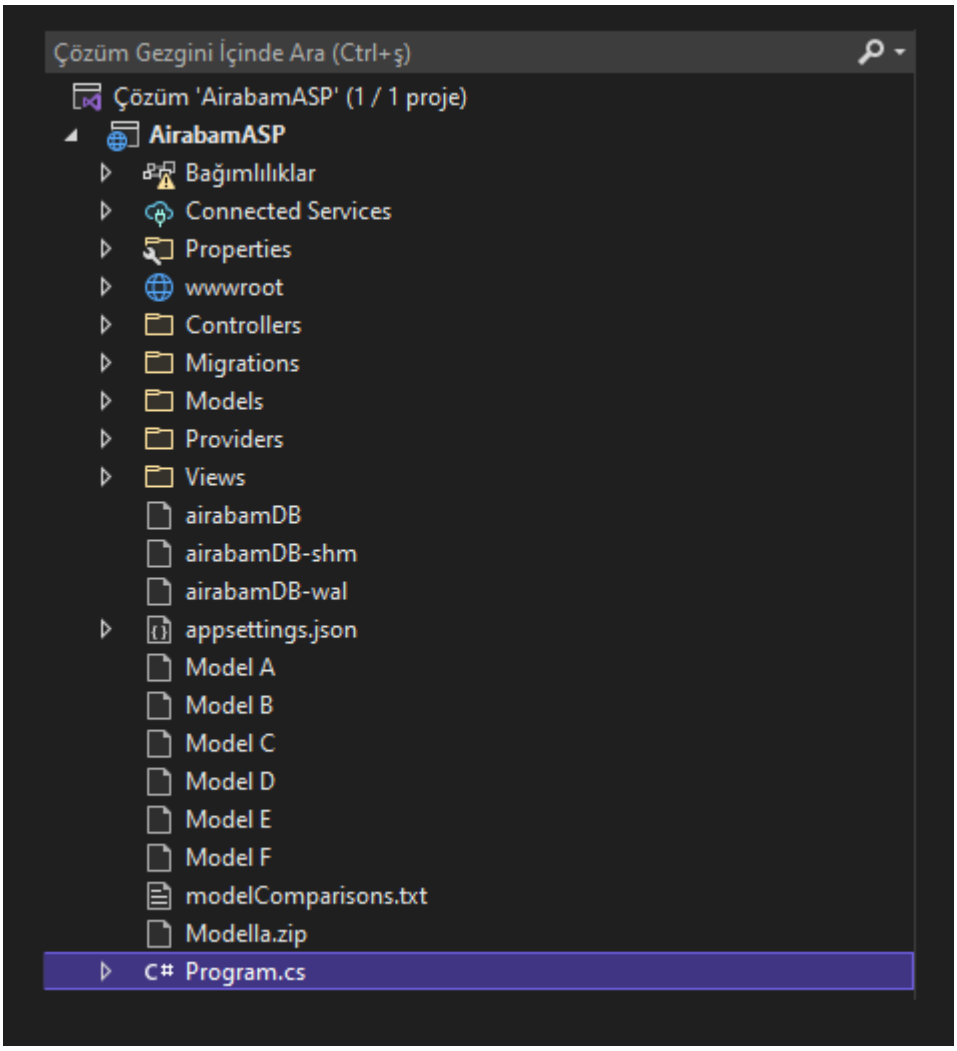


Figure 2 Folder Structure

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

airabamDB Table name: __EFMigrationsHistory WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated
1	MigrationId	TEXT							
2	ProductVersion	TEXT							

Figure 3 __EFMigrationsHistory Table Structure

	MigrationId	ProductVersion
1	20221028124427_InitialCreate	6.0.10
2	20221028131214_SecondUpdateForAuth	6.0.10
3	20221028205819_AddingCars	6.0.10
4	20221028210031_UpdateCarsv1	6.0.10
5	20221029101011_UpdateCarsv2	6.0.10
6	20221115202235_UpdateCarsv3	6.0.10
7	20221115202326_UpdateCarsv31	6.0.10
8	20221231091838_TrainingSet	6.0.10
9	20221231092021_TestSet	6.0.10
10	20230115153447_AddingFVO	6.0.10
11	20230115153635_AddingFVO2	6.0.10

Figure 4 __EFMigrationsHistory Table Data Sample

airabamDB | Table name: Cars | WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	Id	INTEGER								NULL
2	AdvertDate	TEXT								NULL
3	AvgFuelCons	REAL								NULL
4	Brand	TEXT								NULL
5	Case	TEXT								NULL
6	City	TEXT								NULL
7	Color	TEXT								NULL
8	EnginePow	REAL								NULL
9	EngineVol	REAL								NULL
10	FuelType	TEXT								NULL
11	Gear	TEXT								NULL
12	Link	TEXT								NULL
13	Milage	REAL								NULL
14	Model	TEXT								NULL
15	Price	REAL								NULL
16	Year	REAL								NULL

Figure 5 Cars Table Structure

Filter data | Total rows loaded: 56306

	Id	AdvertDate	AvgFuelCons	Brand	Case	City	Color	EnginePow	EngineVol	FuelType	Gear	Link	Milage	Model	Price	Year
1	1	27 Ekim 2022	5.2	Volks...	Cou...	Şanlı...	Beyaz	177	1968	Dizel	Yarı Otomatik	https://...	239000	VWCC	595000	2013
2	3	27 Ekim 2022	5.1	Opel	Hat...	Adana	Beyaz	85	1229	Benzin	Yarı Otomatik	https://...	47800	Corsa	399750	2014
3	4	27 Ekim 2022	4.9	Toyota	Sedan	Şanlı...	Gri	90	1365	Dizel	Düz	https://...	358000	Corolla	255000	2010
4	5	27 Ekim 2022	7.1	Renault	Sedan	Yozgat	Gri	75	1390	LPG	Düz	https://...	262000	Clio	162500	2002
5	6	27 Ekim 2022	4.1	Hyun...	Hat...	Adıy...	Füme	90	1396	Dizel	Düz	https://...	70000	i20	405000	2017
6	7	27 Ekim 2022	4	Opel	Hat...	İstan...	Gri	75	1248	Dizel	Düz	https://...	114000	Corsa	300000	2013
7	8	27 Ekim 2022	4.3	Kia	Hat...	İstan...	Gri	128	1582	Dizel	Düz	https://...	171000	Ceed	349800	2013
8	9	27 Ekim 2022	5.5	Renault	Hat...	Adana	Beyaz	75	1149	Benzin	Düz	https://...	99800	Clio	283900	2015
9	10	27 Ekim 2022	6.1	Merce...	Cou...	Ankara	Beyaz	204	2143	Dizel	Otomatik	https://...	350000	Benz	880000	2011
10	11	27 Ekim 2022	3.8	Renault	Sedan	Ankara	Gri	110	1461	Dizel	Yarı Otomatik	https://...	182000	Megane	455000	2018
11	12	27 Ekim 2022	5.7	Fiat	Sedan	Yalova	Gri	95	1368	LPG	Düz	https://...	37000	Egea	340000	2020
12	13	27 Ekim 2022	5.1	Kia	Hat...	Yalova	Siyah	85	1248	Benzin	Düz	https://...	61000	Rio	293000	2014
13	14	27 Ekim 2022	3.8	Skoda	Hat...	Yalova	Beyaz	90	1422	Dizel	Yarı Otomatik	https://...	128000	Rapid	428000	2016
14	15	27 Ekim 2022	5.7	Fiat	Sedan	Ankara	Beyaz	95	1368	Benzin	Düz	https://...	20000	Egea	460000	2021
15	16	27 Ekim 2022	5.4	Seat	Sedan	Balık...	Füme	105	1197	Benzin	Düz	https://...	130000	Toledo	320000	2013
16	17	27 Ekim 2022	5.7	Opel	Hat...	Denizli	Kır...	100	1364	Benzin	Otomatik	https://...	92000	Corsa	370000	2013
17	18	27 Ekim 2022	6	Renault	Stati...	Gazi...	Beyaz	60	1300	LPG	Düz	https://...	30000	R12	50000	1974
18	19	27 Ekim 2022	6.7	Honda	Sedan	Sivas	Diğer	125	1595	LPG	Düz	https://...	317000	Civic	222000	2008
19	20	27 Ekim 2022	5.7	Ford	Hat...	Adana	Beyaz	96	1388	Benzin	Düz	https://...	125000	Fiesta	237000	2011
20	21	27 Ekim 2022	47	Renault	Sedan	Yozgat	Beyaz	83	1390	LPG	Düz	https://...	199999	R9	84000	1992
21	22	27 Ekim 2022	4.6	Opel	Hat...	Denizli	Diğer	90	1248	Dizel	Düz	https://...	191000	Corsa	220000	2007
22	23	27 Ekim 2022	4.1	Toyota	Sedan	Ankara	Beyaz	90	1364	Dizel	Yarı Otomatik	https://...	124100	Corolla	415000	2013
23	25	27 Ekim 2022	3.8	Dacia	Hat...	İstan...	Gri	90	1461	Dizel	Yarı Otomatik	https://...	149000	Sandero	325000	2016
24	27	27 Ekim 2022	7.4	Chevr...	Sedan	Balık...	Gri	124	1598	LPG	Otomatik	https://...	163550	Cruze	335000	2012
25	28	27 Ekim 2022	3.8	Volks...	Hat...	Konya	Beyaz	105	1598	Dizel	Düz	https://...	169100	Golf	482000	2014
26	29	27 Ekim 2022	5.9	Chevr...	Hat...	İstan...	Gri	100	1398	Benzin	Düz	https://...	147000	Aveo	265000	2012
27	30	27 Ekim 2022	8	Audi	Hat...	Çorum	Siyah	102	1595	LPG	Otomatik	https://...	300000	A3Sportback	219000	2003
28	31	27 Ekim 2022	6.3	Renault	Hat...	Hatay	Beyaz	115	1598	Benzin	Düz	https://...	39000	Megane	430000	2018
29	32	27 Ekim 2022	7.9	Volks...	Stati...	Kara...	Gri	115	1598	Benzin	Yarı Otomatik	https://...	124000	Passat	370000	2008
30	33	27 Ekim 2022	4.8	Ford	Sedan	Çorum	Gri	110	1560	Dizel	Düz	https://...	280000	Focus	275000	2006
31	34	27 Ekim 2022	47	Renault	Sedan	Çorum	Beyaz	83	1390	LPG	Düz	https://...	230000	R9	71000	1992
32	35	27 Ekim 2022	6.2	Hyun...	Hat...	Balık...	Gri	82	1341	LPG	Düz	https://...	160000	Getz	202222	2005
33	36	27 Ekim 2022	6.7	Hyun...	Sedan	Gazi...	Yeşil	86	1341	LPG	Düz	https://...	360000	Accent	152000	2004
34	37	27 Ekim 2022	4.7	Toyota	Sedan	Gazi...	Gri	90	1364	Dizel	Düz	https://...	286000	Corolla	279000	2012

Figure 6 Cars Table Data Sample

airabamDB | Table name: CarsTest | WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	Id	INTEGER								NULL
2	Link	TEXT								NULL
3	Brand	TEXT								NULL
4	Model	TEXT								NULL
5	City	TEXT								NULL
6	Color	TEXT								NULL
7	Year	REAL								NULL
8	Milage	REAL								NULL
9	Price	REAL								NULL
10	AdvertDate	TEXT								NULL
11	Gear	TEXT								NULL
12	Case	TEXT								NULL
13	AvgFuelCons	REAL								NULL
14	FuelType	TEXT								NULL
15	EnginePow	REAL								NULL
16	EngineVol	REAL								NULL

Figure 7 CarsTest Table Structure

Filter data | Total rows loaded: 7582

	Id	Link	Brand	Model	City	Color	Year	Milage	Price	AdvertDate	Gear	Case	AvgFuelC	FuelType	EnginePo	EngineVol
1	1	https://www.arabam.com/...	Toyota	Toyota Corolla 1.5 Dream	Osmaniye	Beyaz	2021	6001	675000	26 Ekim 2022	Otomatik	Sedan	5.8	Benzin	125	1490
2	2	https://www.arabam.com/...	Renault	Renault Symbol 1.5 dCi Authentique	Aydın	Gri	2012	210000	217000	26 Ekim 2022	Düz	Sedan	4.5	Dizel	65	1461
3	3	https://www.arabam.com/...	Volkswagen	Volkswagen Passat 1.6 Trendline	Osmaniye	Gri	2007	315000	310750	26 Ekim 2022	Düz	Sedan	7.6	LPG	102	1595
4	4	https://www.arabam.com/...	Ford	Ford Mondeo 2.0 TDCi Trend	Tekirdağ	-	2012	260000	335000	27 Ekim 2022	Yarı Otomatik	Sedan	5.6	Dizel	163	1997
5	5	https://www.arabam.com/...	Skoda	Skoda Felicia 1.6 GLXi	Ankara	Yeşil	1997	254000	105000	27 Ekim 2022	Düz	Hatchback	42	LPG	75	1598
6	6	https://www.arabam.com/...	Volkswagen	Volkswagen Passat 1.6 TDi BlueMotion Comfortline	Hatay	-	2012	164000	530000	27 Ekim 2022	Yarı Otomatik	Sedan	4.7	Dizel	105	1598
7	7	https://www.arabam.com/...	Fiat	Fiat Egea 1.3 Multijet Easy	İzmir	Beyaz	2016	153800	318000	27 Ekim 2022	Düz	Sedan	4.1	Dizel	95	1248
8	8	https://www.arabam.com/...	Opel	Opel Astra 1.3 CDTI Active	Mersin	Beyaz	2013	134500	375000	27 Ekim 2022	Düz	Hatchback	4.1	Dizel	95	1248
9	9	https://www.arabam.com/...	Renault	Renault Talisman 1.6 dCi Touch	İstanbul	Beyaz	2016	89500	640000	01 Kasım 2022	Düz	Sedan	3.9	Dizel	130	1598
10	22	https://www.arabam.com/...	BMW	BMW 7 Serisi 730d Long Exclusive	Bursa	Siyah	2011	402000	1180000	01 Kasım 2022	Otomatik	Sedan	7.2	Dizel	245	2993
11	23	https://www.arabam.com/...	Suzuki	Suzuki Alto 0.8 GL	Konya	Siyah	1991	168000	58000	01 Kasım 2022	Düz	Hatchback	35	Benzin	41	796
12	24	https://www.arabam.com/...	Tata	Tata Vista 1.4 Safire Mystic Plus	Antalya	-	2012	88350	182000	01 Kasım 2022	Düz	Hatchback	5.9	LPG	75	1368
13	25	https://www.arabam.com/...	Opel	Opel Astra 1.3 CDTI Essentia Konfor	Bursa	Füme	2011	123000	342000	01 Kasım 2022	Yarı Otomatik	Hatchback	5	Dizel	90	1248
14	26	https://www.arabam.com/...	Hyundai	Hyundai Elantra 1.6 D-CVVT Mode	Sakarya	Siyah	2011	220000	350000	01 Kasım 2022	Düz	Sedan	6.4	LPG	132	1591
15	27	https://www.arabam.com/...	BMW	BMW 3 Serisi 320i Premium	Karaman	-	2008	305000	380000	01 Kasım 2022	Otomatik	Sedan	6.1	LPG	170	1995
16	28	https://www.arabam.com/...	Volkswagen	Volkswagen Jetta 1.6 TDi Primeline	Kayseri	-	2011	210000	350000	01 Kasım 2022	Yarı Otomatik	Sedan	4.7	Dizel	105	1598
17	29	https://www.arabam.com/...	Renault	Renault Megane 1.5 dCi Expression	Osmaniye	-	2006	354000	230000	01 Kasım 2022	Düz	Sedan	4.6	Dizel	80	1461
18	30	https://www.arabam.com/...	Renault	Renault Fluence 1.5 dCi Icon	Samsun	Gri	2015	98000	475000	01 Kasım 2022	Düz	Sedan	4.6	Dizel	110	1461
19	31	https://www.arabam.com/...	Fiat	Fiat Egea 1.4 Fire Urban	Diyarbakır	-	2017	65000	390000	01 Kasım 2022	Düz	Sedan	5.7	LPG	95	1368
20	32	https://www.arabam.com/...	Volkswagen	Volkswagen Golf 1.4 TSi Pulse	Sinop	Gri	2012	84000	440000	01 Kasım 2022	Düz	Hatchback	6.2	Benzin	122	1390
21	33	https://www.arabam.com/...	Mercedes	Mercedes - Benz E 350 CDI Premium	Eskişehir	Gri	2012	196000	1800000	01 Kasım 2022	Otomatik	Sedan	6.4	Dizel	265	2987
22	34	https://www.arabam.com/...	Hyundai	Hyundai Accent 1.3 LX	İstanbul	-	2000	242350	176000	01 Kasım 2022	Otomatik	Sedan	6.1	Benzin	86	1341
23	35	https://www.arabam.com/...	Tofaş	Tofaş Şahin 1.6	Yozgat	Beyaz	1997	200000	85000	01 Kasım 2022	Düz	Sedan	52	LPG	-999	1581
24	36	https://www.arabam.com/...	Kia	Kia Cerato 1.6 CRDi Concept Plus	Siirt	-	2017	83000	555000	01 Kasım 2022	Yarı Otomatik	Sedan	4.6	Dizel	136	1582
25	37	https://www.arabam.com/...	Peugeot	Peugeot 308 1.6 HDi Comfort	İzmir	Gri	2010	246000	240000	01 Kasım 2022	Düz	Hatchback	6	Dizel	91	1560
26	38	https://www.arabam.com/...	Ford	Ford Escort 1.6 CL	Tekirdağ	Lacivert	1996	230000	110000	01 Kasım 2022	Düz	Sedan	55	LPG	90	1597
27	39	https://www.arabam.com/...	Mercedes	Mercedes - Benz E 200 Kompressor Avantgarde	İstanbul	Siyah	2009	208000	720000	01 Kasım 2022	Otomatik	Sedan	8.9	LPG	184	1796
28	40	https://www.arabam.com/...	Opel	Opel Corsa 1.3 CDTI Essentia	Eskişehir	Beyaz	2006	195000	159500	01 Kasım 2022	Düz	Hatchback	4.6	Dizel	75	1248
29	41	https://www.arabam.com/...	Toyota	Toyota Corolla 1.6 Linea Terra	Gaziantep	-	2002	250000	215000	01 Kasım 2022	Düz	Sedan	7	LPG	110	1598
30	42	https://www.arabam.com/...	Rover	Rover 216 Si	İzmir	-	1998	243000	95000	01 Kasım 2022	Düz	Hatchback	7.2	LPG	111	1589
31	43	https://www.arabam.com/...	Mercedes	Mercedes - Benz A 160 Avantgarde	Ankara	Mavi	1998	165000	182000	01 Kasım 2022	Düz	Hatchback	7.2	Benzin	102	1598
32	44	https://www.arabam.com/...	Ford	Ford Focus 1.6 TDCi Trend	İzmir	-	2011	243000	325000	01 Kasım 2022	Düz	Sedan	4.2	Dizel	95	1560
33	45	https://www.arabam.com/...	Fiat	Fiat Egea 1.4 Fire Easy	İstanbul	Mavi	2021	24000	520000	01 Kasım 2022	Düz	Sedan	6.4	LPG	95	1368
34	46	https://www.arabam.com/...	Tofaş	Tofaş Doğan SLX ie	Çanakkale	Bordo	1999	89000	140000	01 Kasım 2022	Düz	Sedan	6.6	LPG	96	1581
35	47	https://www.arabam.com/...	Peugeot	Peugeot 307 1.6 Premium	Amasya	Siyah	2006	171000	235000	01 Kasım 2022	Düz	Hatchback	7.4	LPG	110	1587

Figure 8 CarsTest Table Data Sample

airabamDB Table name: FeatureTests WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	Id	INTEGER								NULL
2	Feature	TEXT								NULL
3	Value	TEXT								NULL
4	Output	REAL								NULL

Figure 9 FeatureTests Table Structure

	Id	Feature	Value	Output
1	331	Gear	Yarı Otomatik	-9444.6435546875
2	332	Gear	Düz	-32139.48828125
3	333	Gear	Otomatik	19229.38671875
4	334	FuelType	Dizel	23721.41796875
5	335	FuelType	Benzin	24107.73046875
6	336	FuelType	LPG	19229.38671875
7	337	FuelType	Hibrit	15679.3564453125
8	338	FuelType	Elektrik	21596.60546875
9	339	Color	Beyaz	17694.41796875
10	340	Color	Gri	20250.94921875
11	341	Color	Füme	21511.19921875
12	342	Color	Siyah	22257.85546875
13	343	Color	Kırmızı	35492.88671875
14	344	Color	Diğer	21596.60546875
15	345	Color	Yeşil	31334.44921875
16	346	Color	Bordo	18328.16796875
17	347	Color	Mavi	29245.38671875
18	348	Color	Şampanya	25277.13671875
19	349	Color	Turuncu	22310.88671875

Figure 10 FeatureTests Table Data Sample

airabamDB Table name: Users WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	Id	TEXT								NULL
2	EmailAddress	TEXT								NULL
3	PasswordHash	TEXT								NULL
4	Salt	TEXT								NULL
5	Name	TEXT								NULL
6	CreatedUtc	TEXT								NULL

Figure 11 Users Table Structure

Filter data Total rows loaded: 1

	Id	EmailAddress	PasswordHash	Salt	Name	CreatedUtc
1	4600B69E-8978-4CED-B2D4-663C05C6A0D3	[redacted]hotmail.com	fMlk07BTlHX5sU3Qzl5ipeBqx9zd1YeQH6E5EbkNOAI=	8mbfj+6xZ0zwjpImcdQd4WgqNcZmpreYnd+YrCgtjZY=	Some User	2022-10-28 13:28:16.4534518

Figure 12 Users Table Data Sample

3.3 User Interface

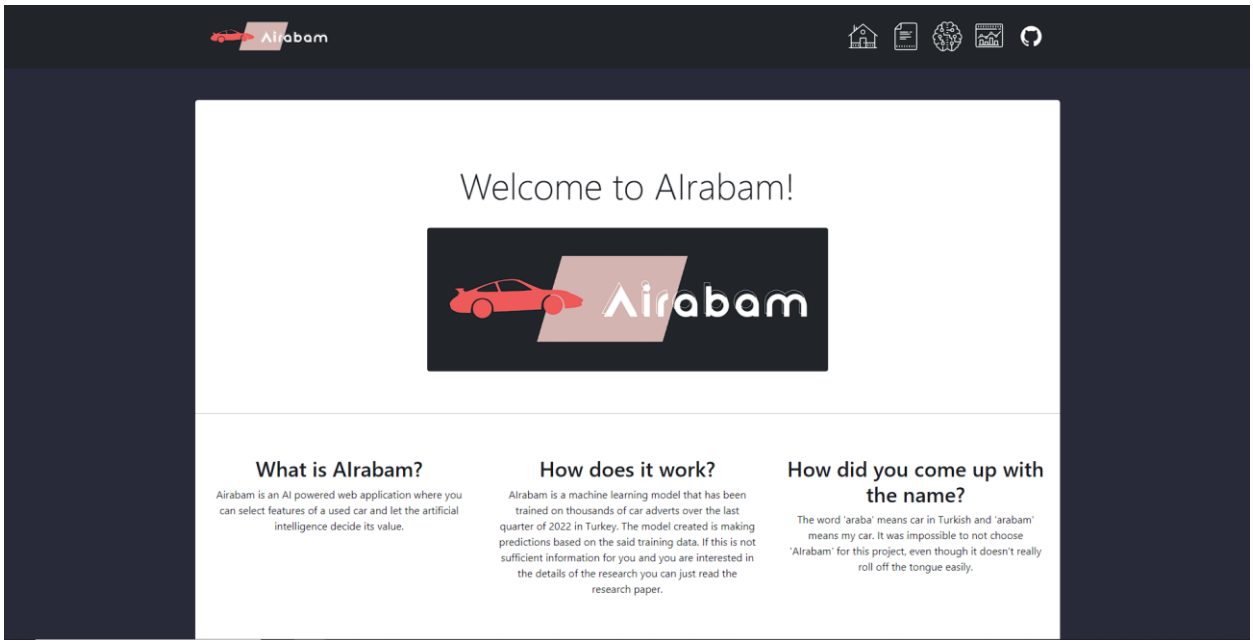


Figure 13 Home Page - 1

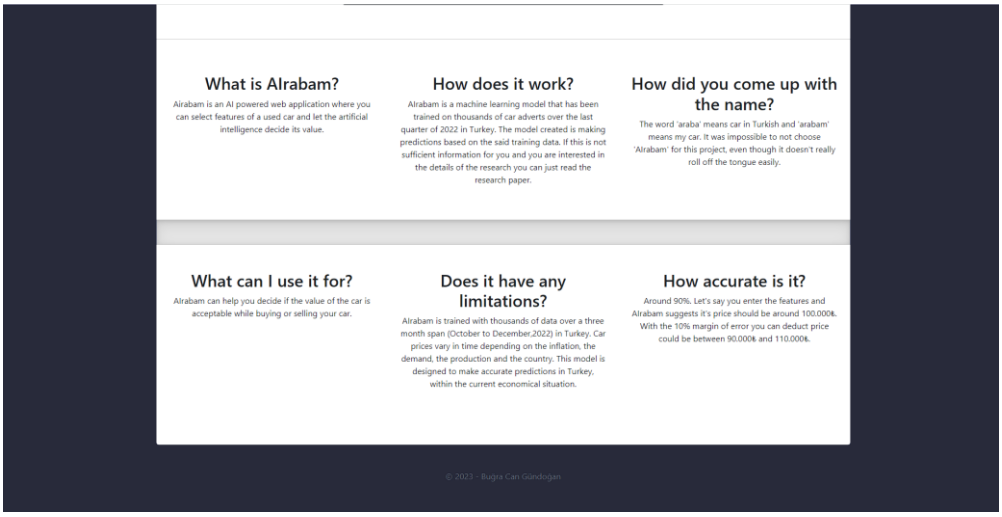


Figure 14 Home Page – 2

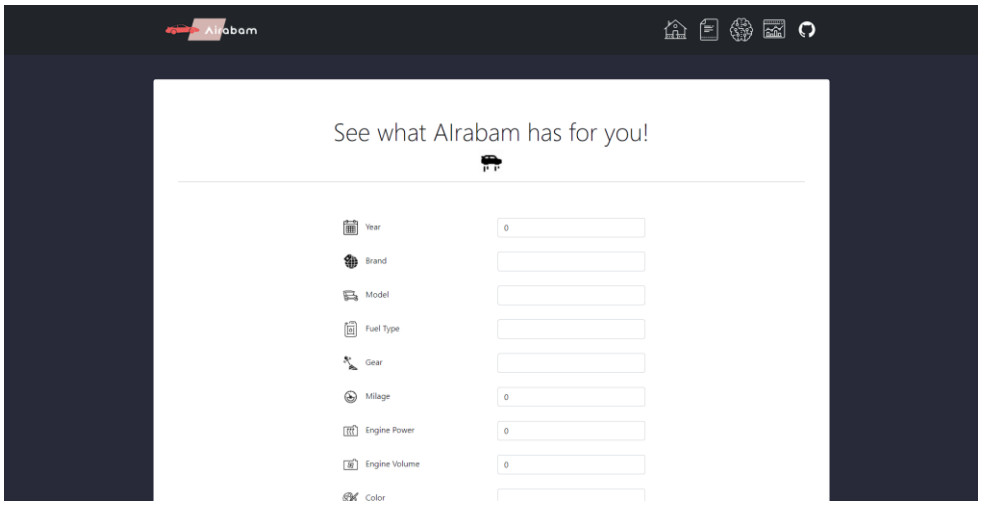


Figure 15 Predict Price Page – 1

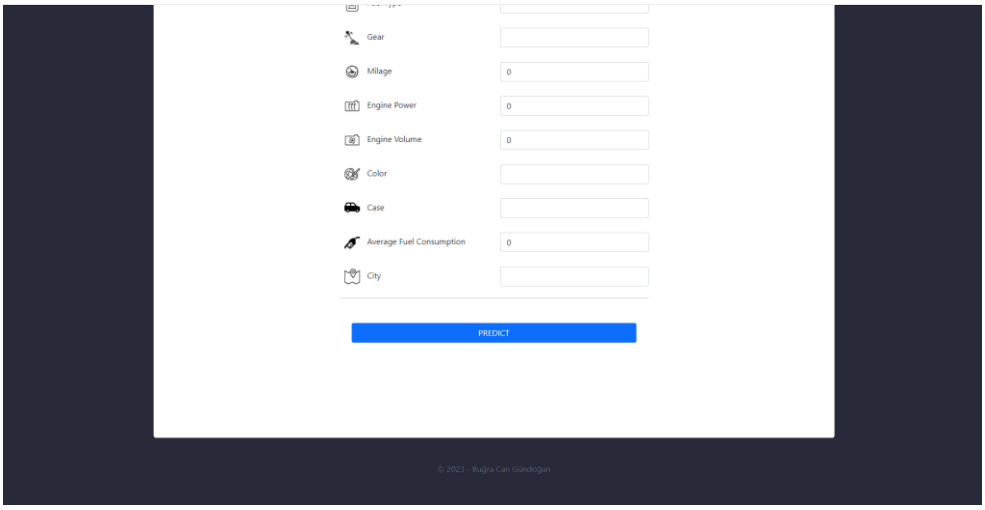


Figure 16 Predict Price Page – 2

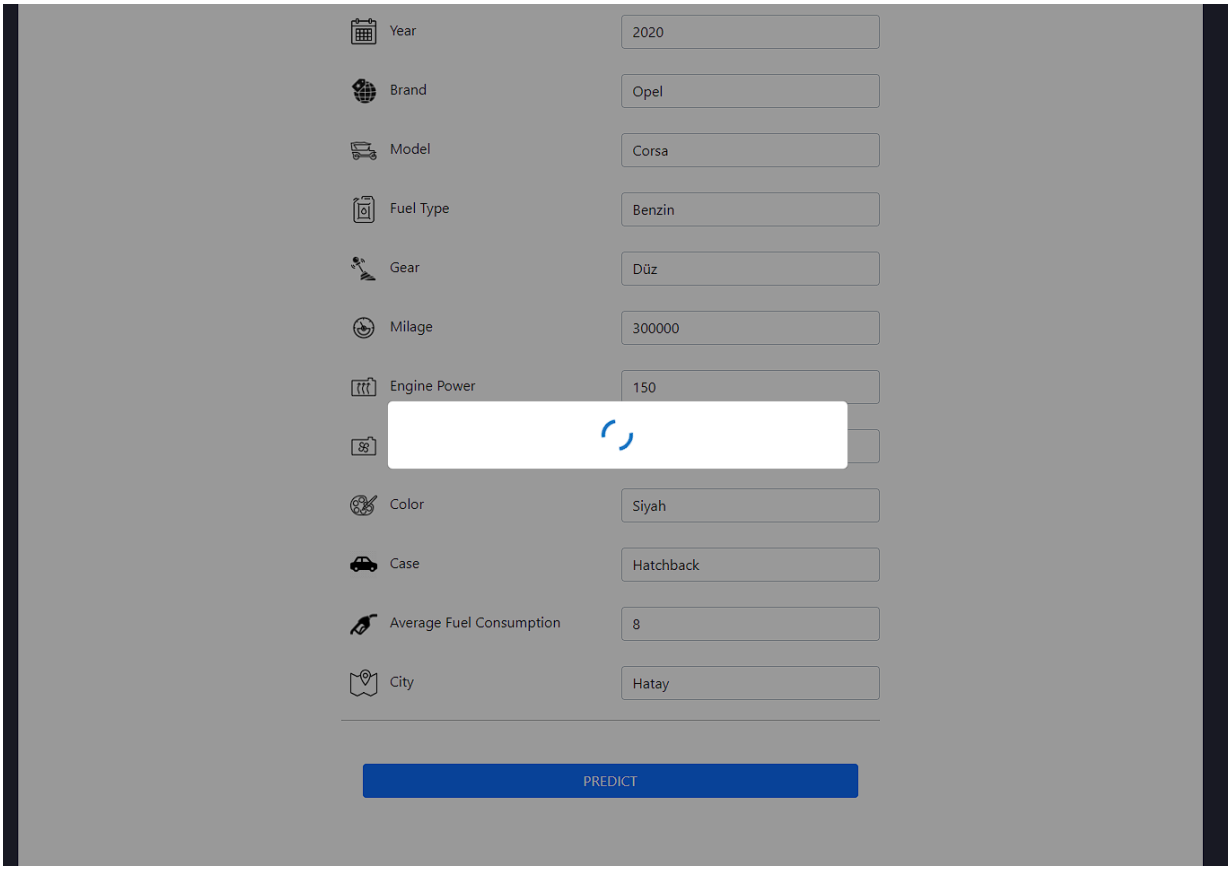


Figure 17 Predict Price Page – 3

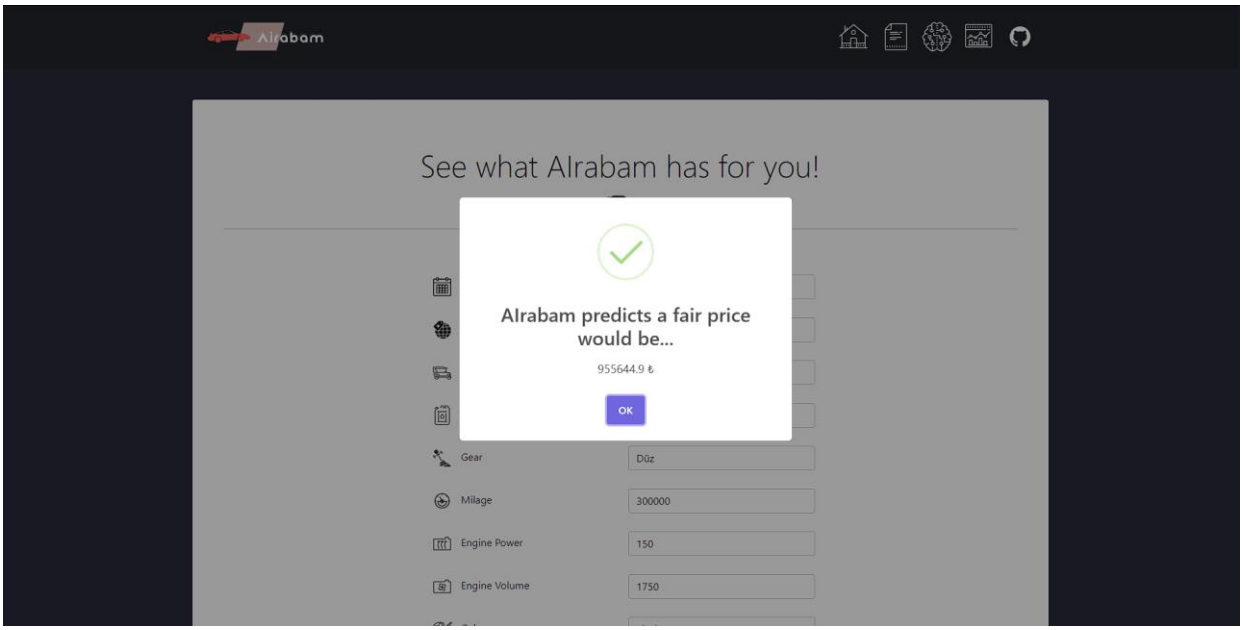
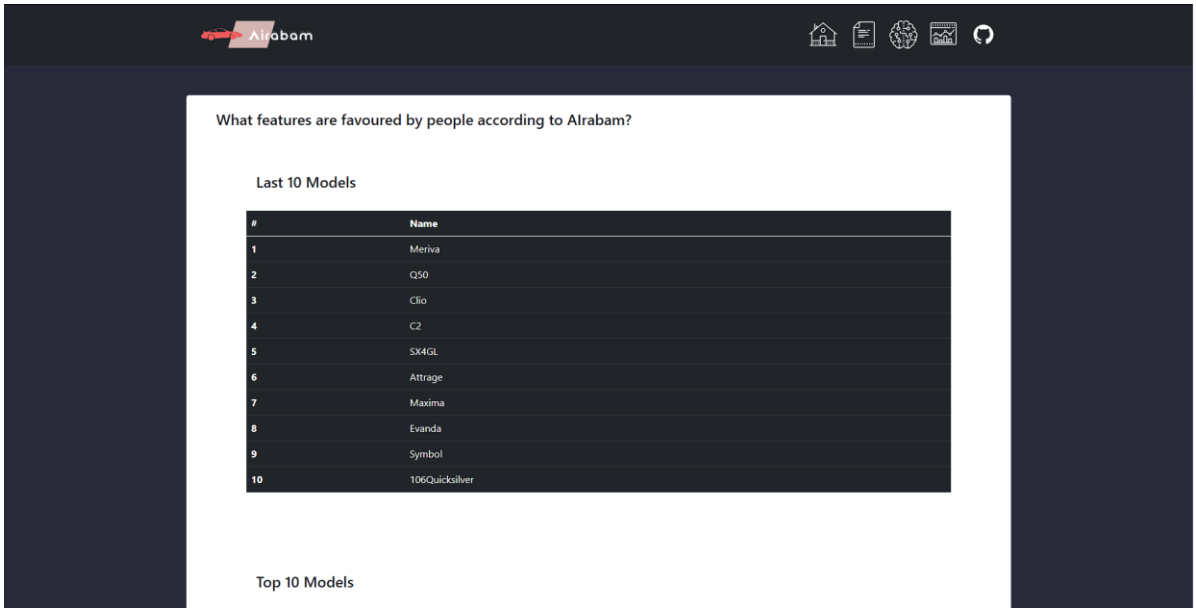


Figure 18 Predict Price Page – 4



#	Name
1	Meriva
2	Q50
3	Clio
4	C2
5	SX4GL
6	Attrage
7	Maxima
8	Evanda
9	Symbol
10	106Quicksilver

Figure 19 Graphs Page – 1

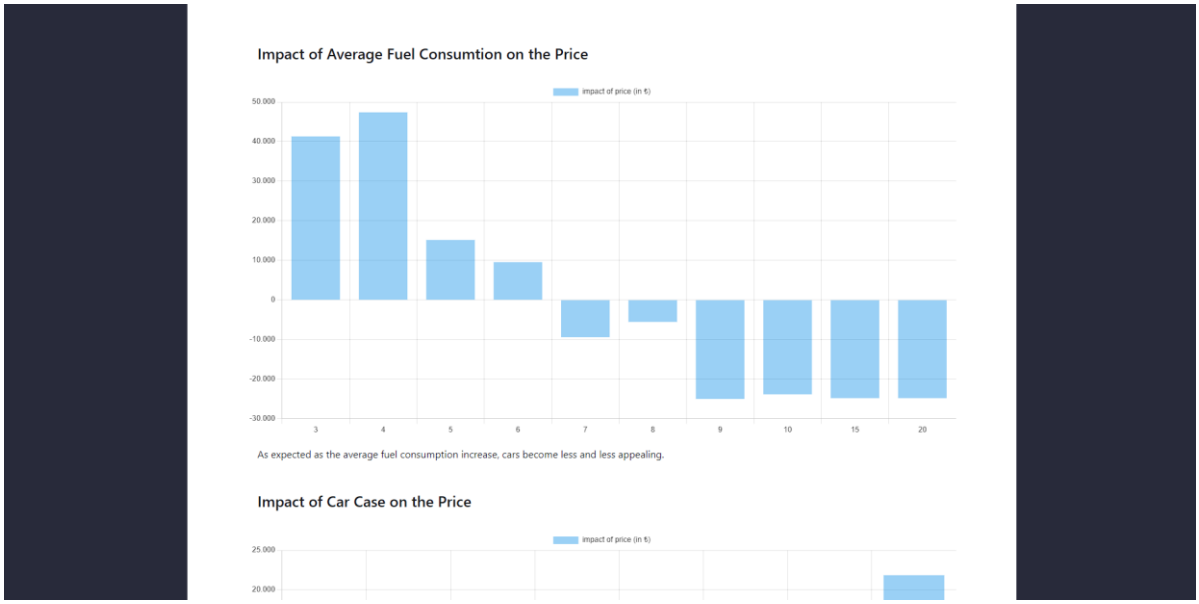


Figure 20 Graphs Page – 2

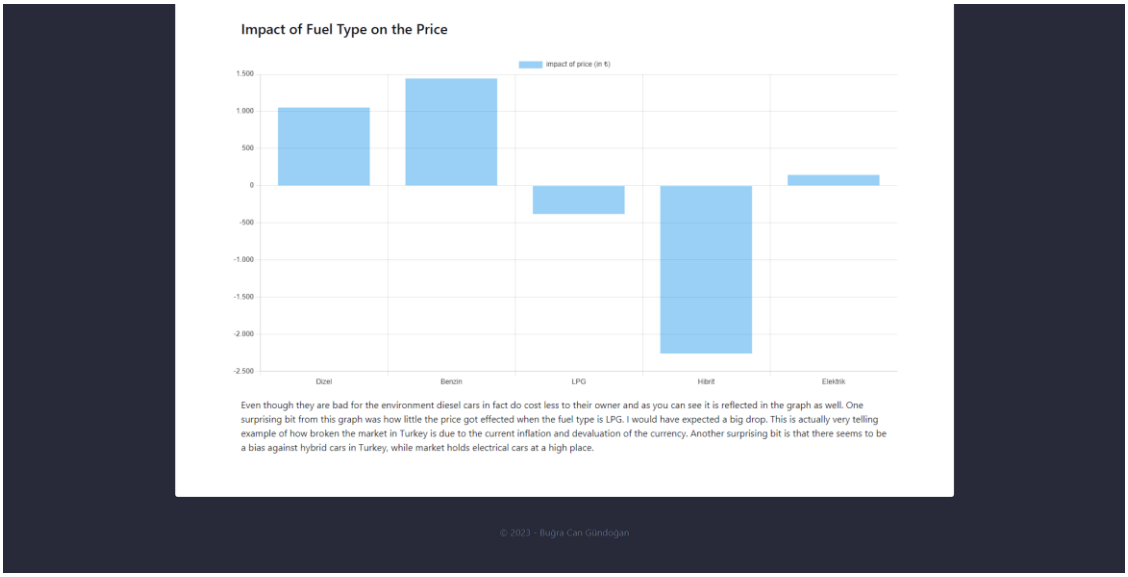


Figure 21 Graphs Page – 3

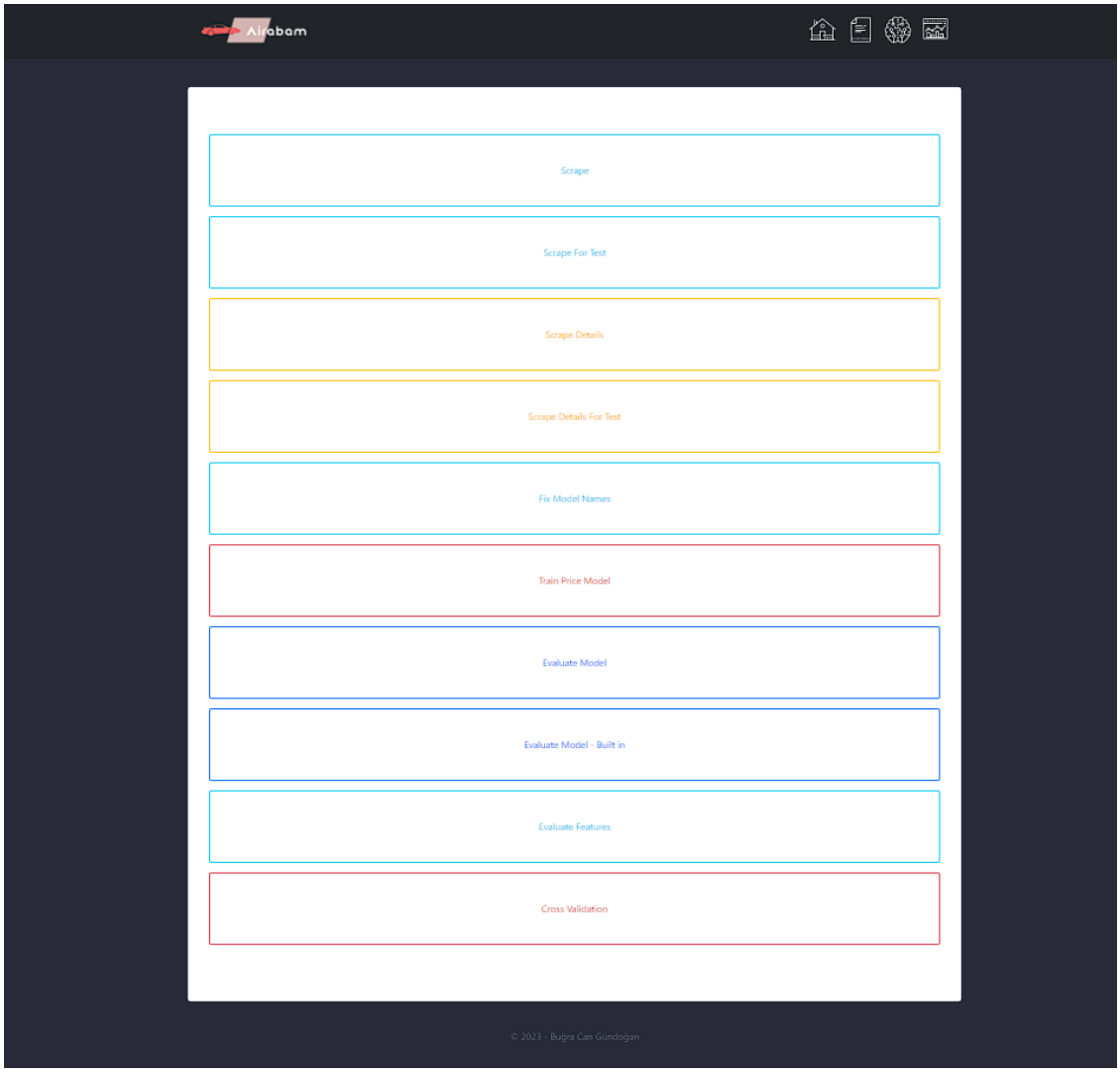


Figure 22 Admin Page

3.4 Design of Classes

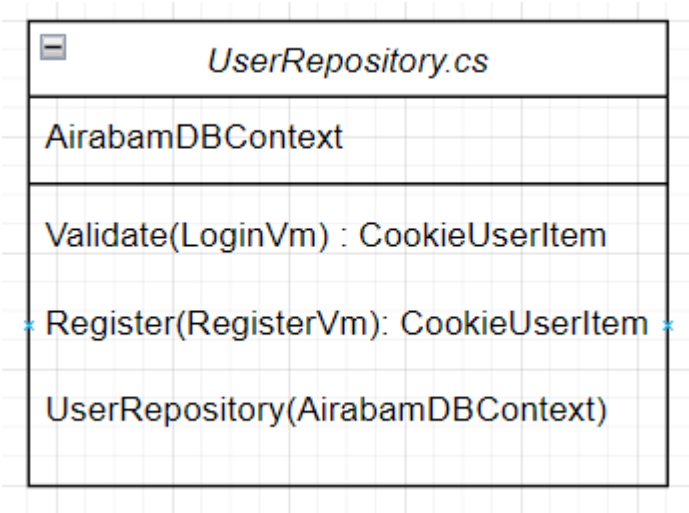


Figure 23 UserRepository.cs

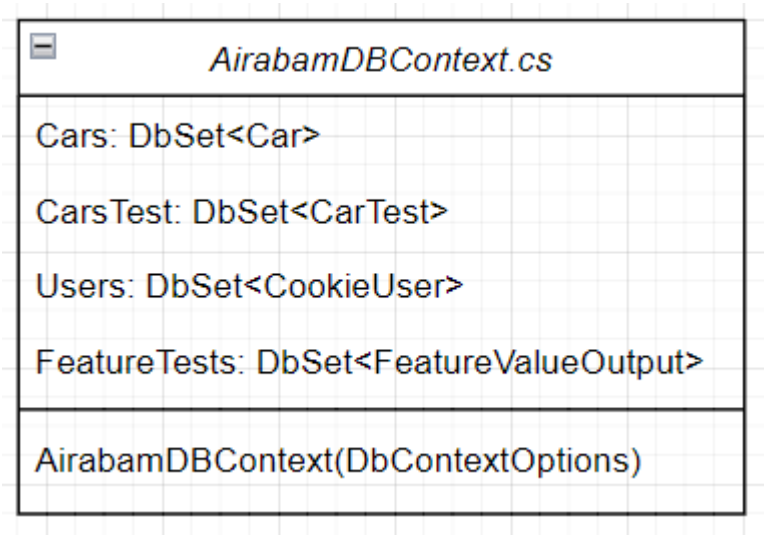


Figure 24 AirabamDBContext.cs

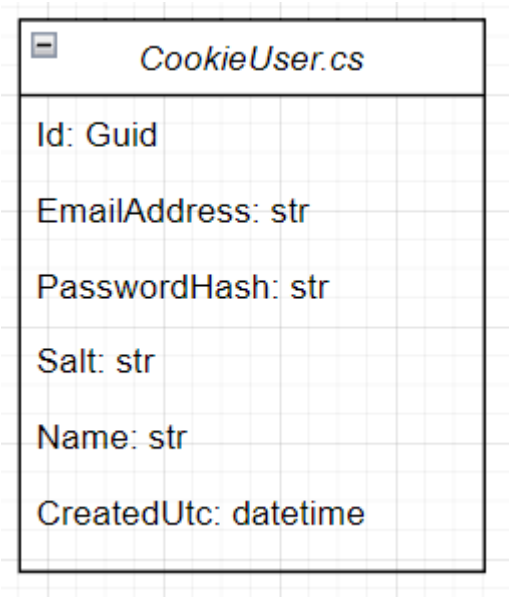


Figure 25 CookieUser.cs

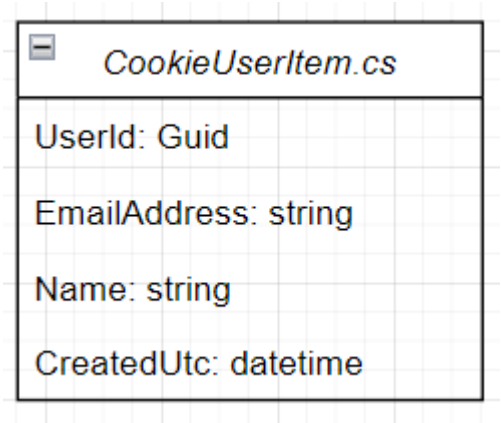


Figure 26 CookieUserItem.cs

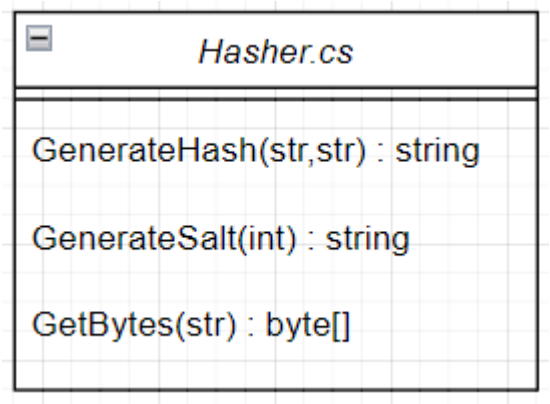


Figure 27 Hasher.cs

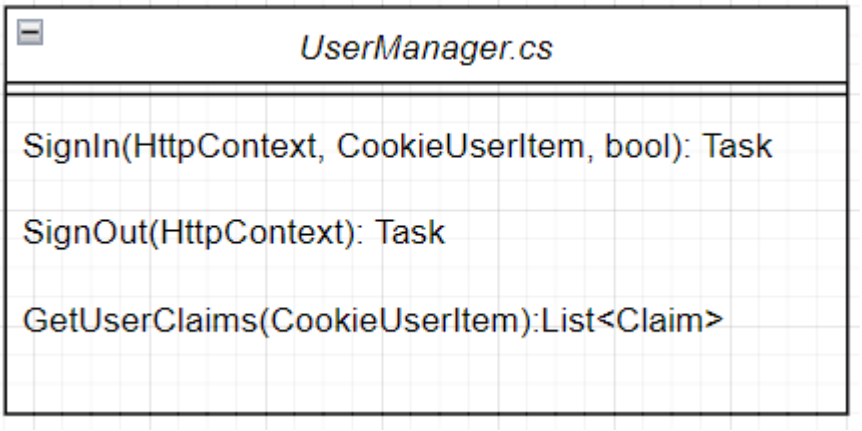


Figure 28 UserManager.cs

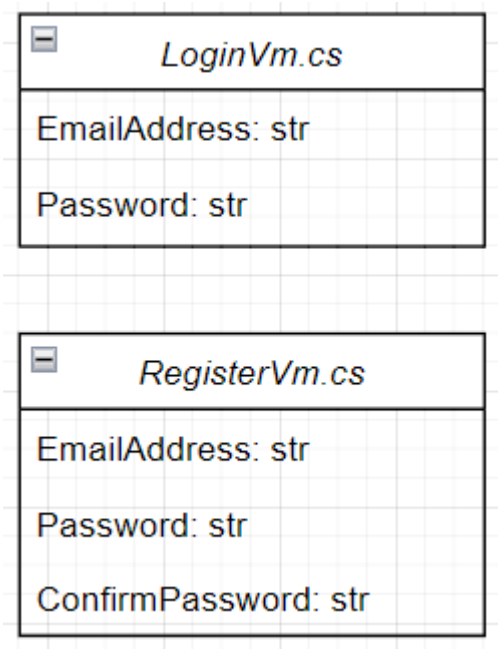


Figure 29 LoginVm.cs and RegisterVm.cs

Figure 30 Car.cs and CarTest.cs

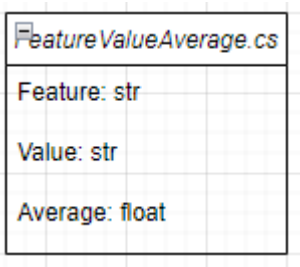


Figure 31 FeatureValueAverage.cs

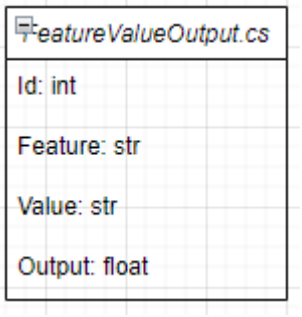


Figure 32 FeatureValueOutput.cs

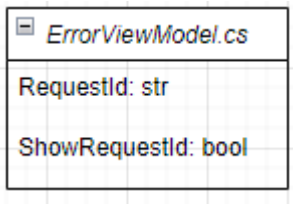


Figure 33 ErrorViewModel.cs

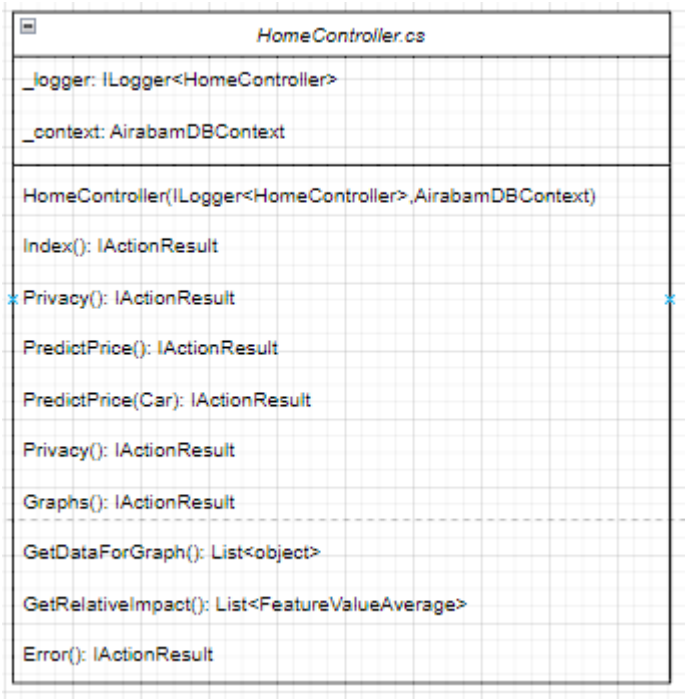


Figure 34 HomeController.cs

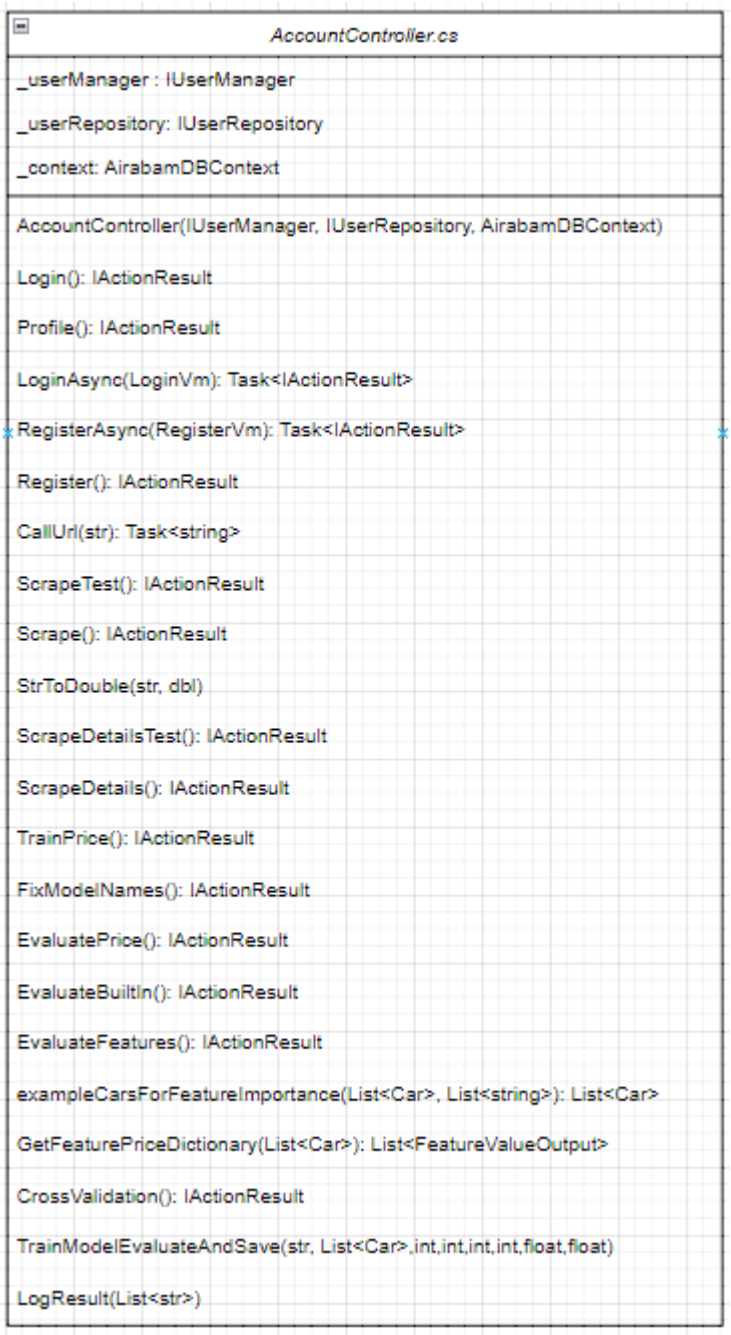


Figure 35 AccountController.cs

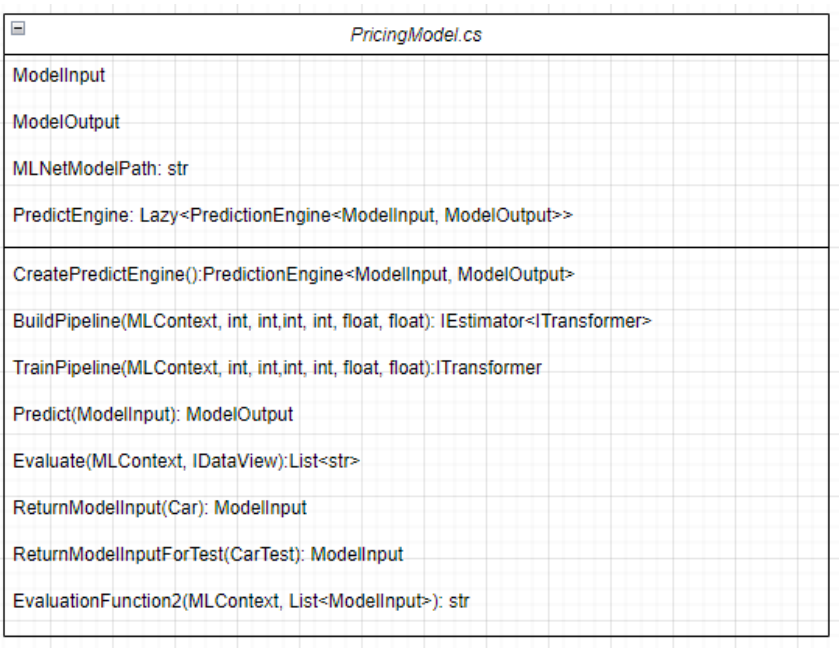


Figure 36 PricingModel.cs

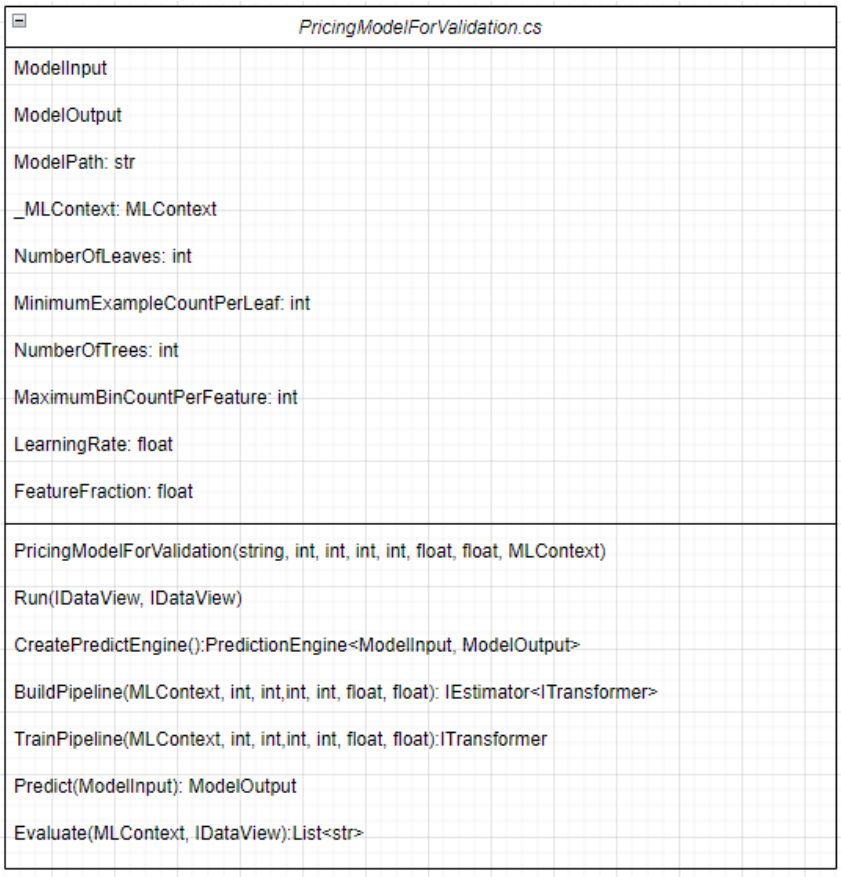


Figure 37 PricingModelForValidation.cs

3.5 Interactions Between Classes

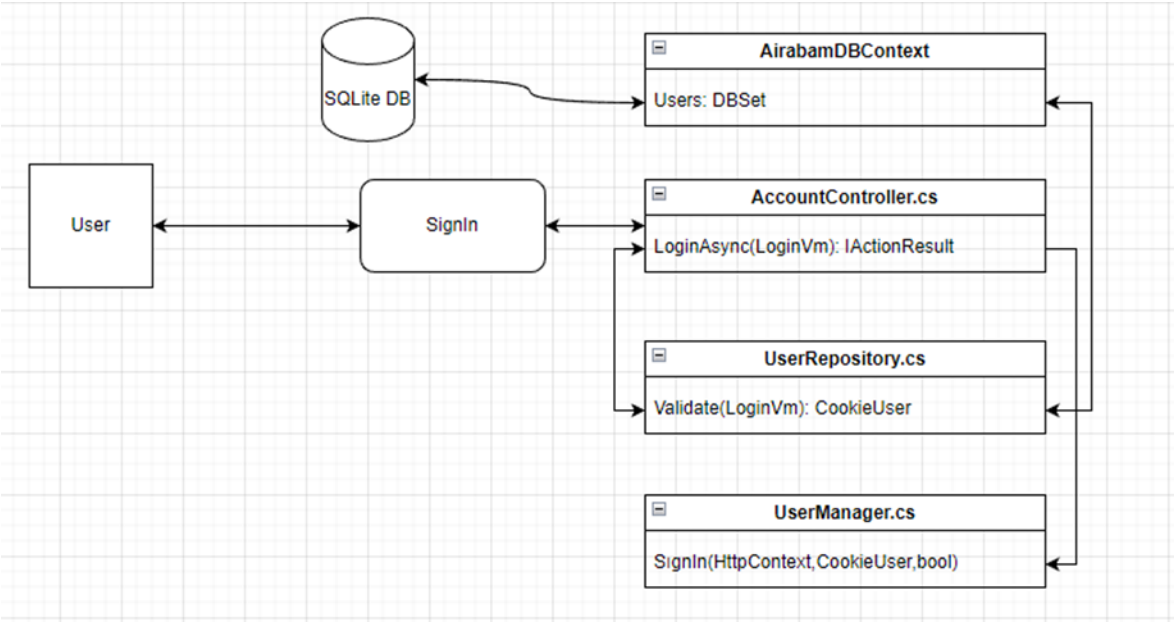


Figure 38 Authentication System

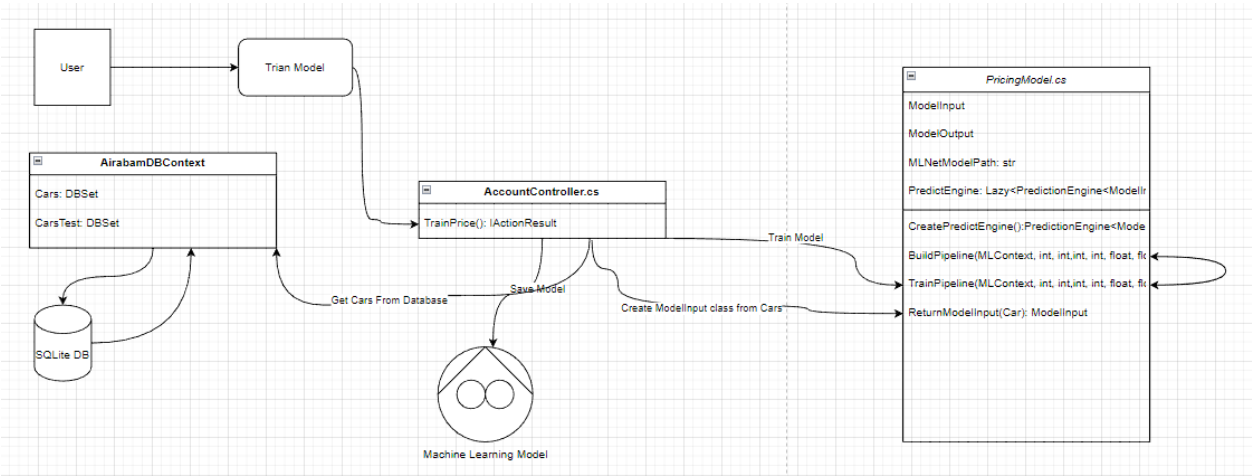


Figure 39 Model Training System

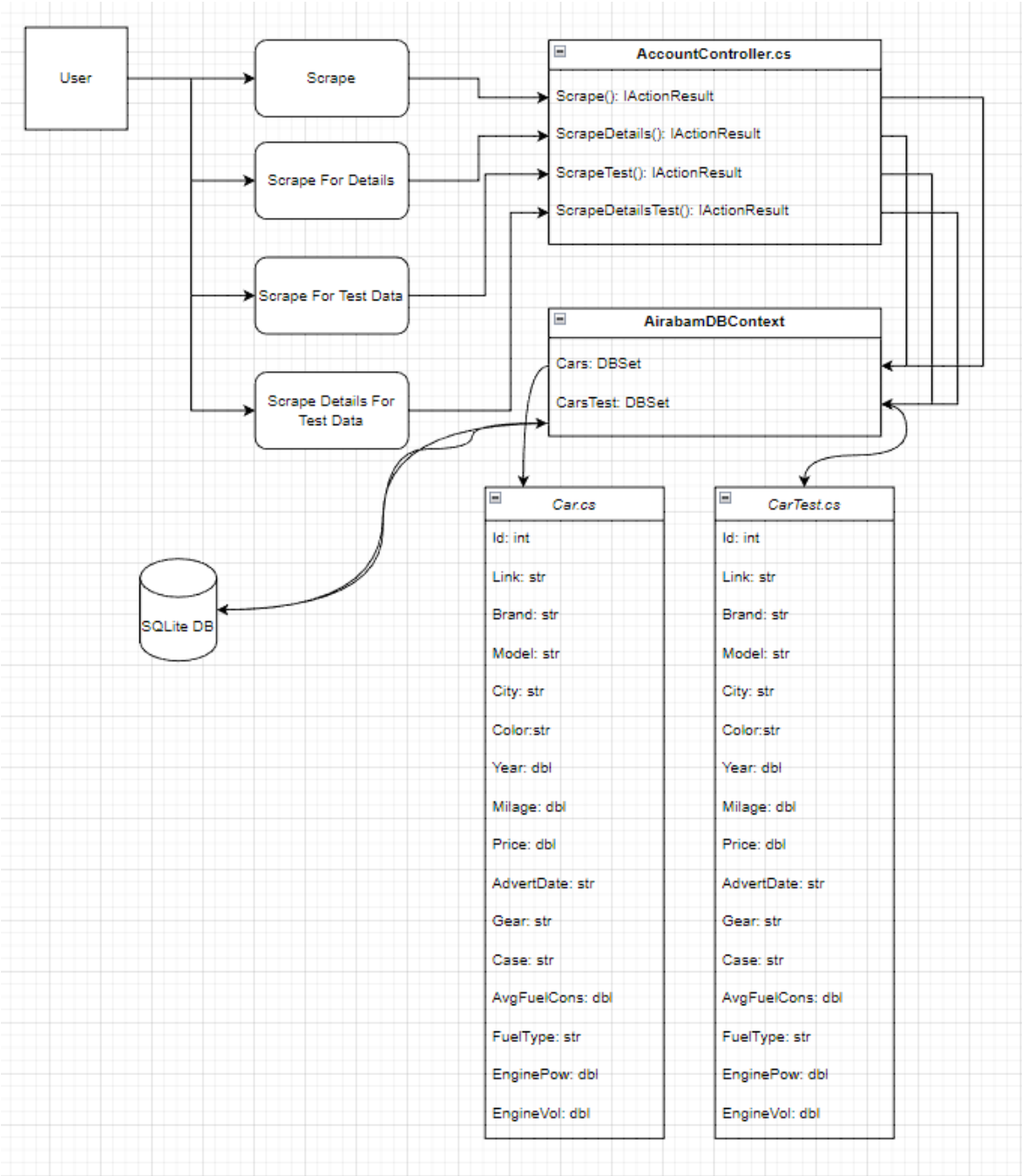


Figure 40 Scraping System

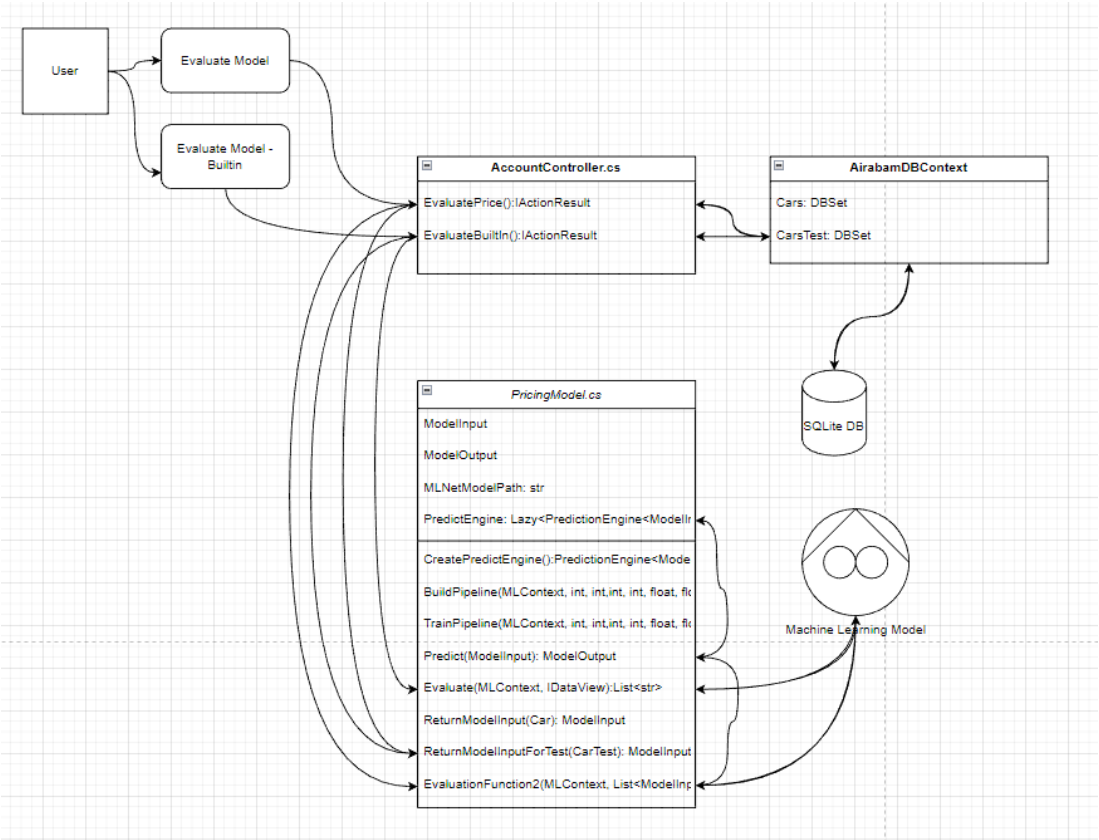


Figure 41 Evaluation System

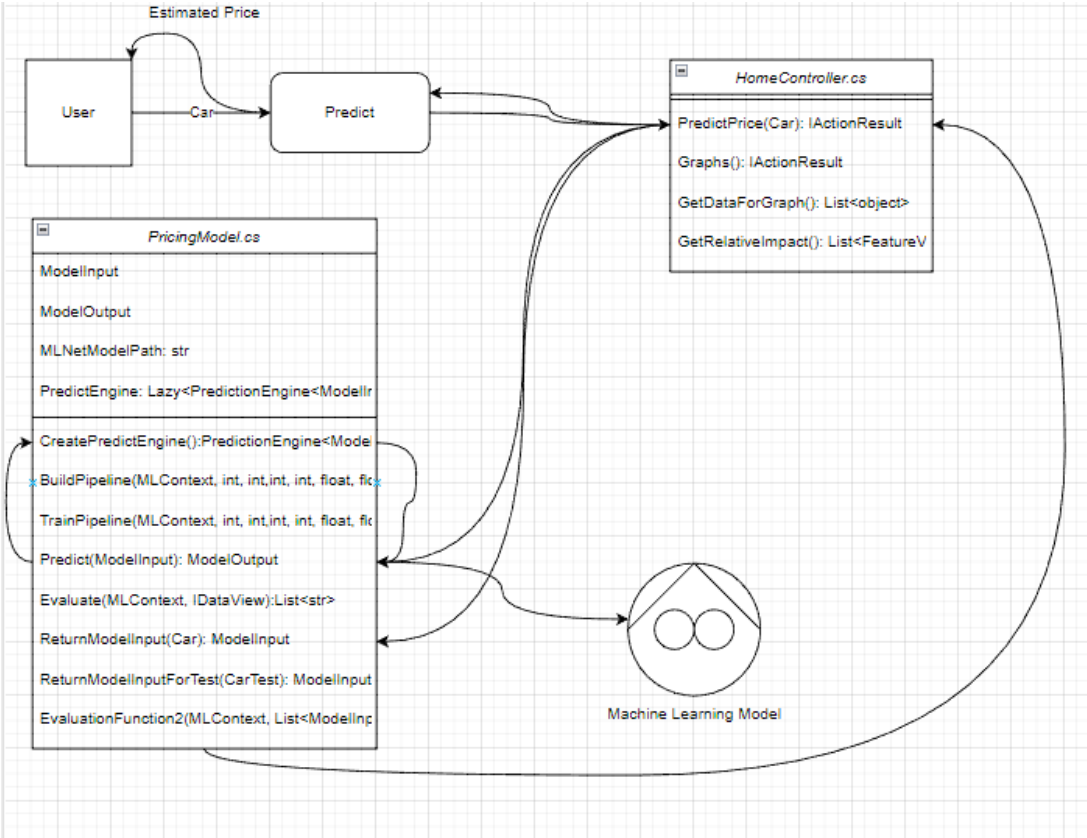


Figure 42 Prediction System

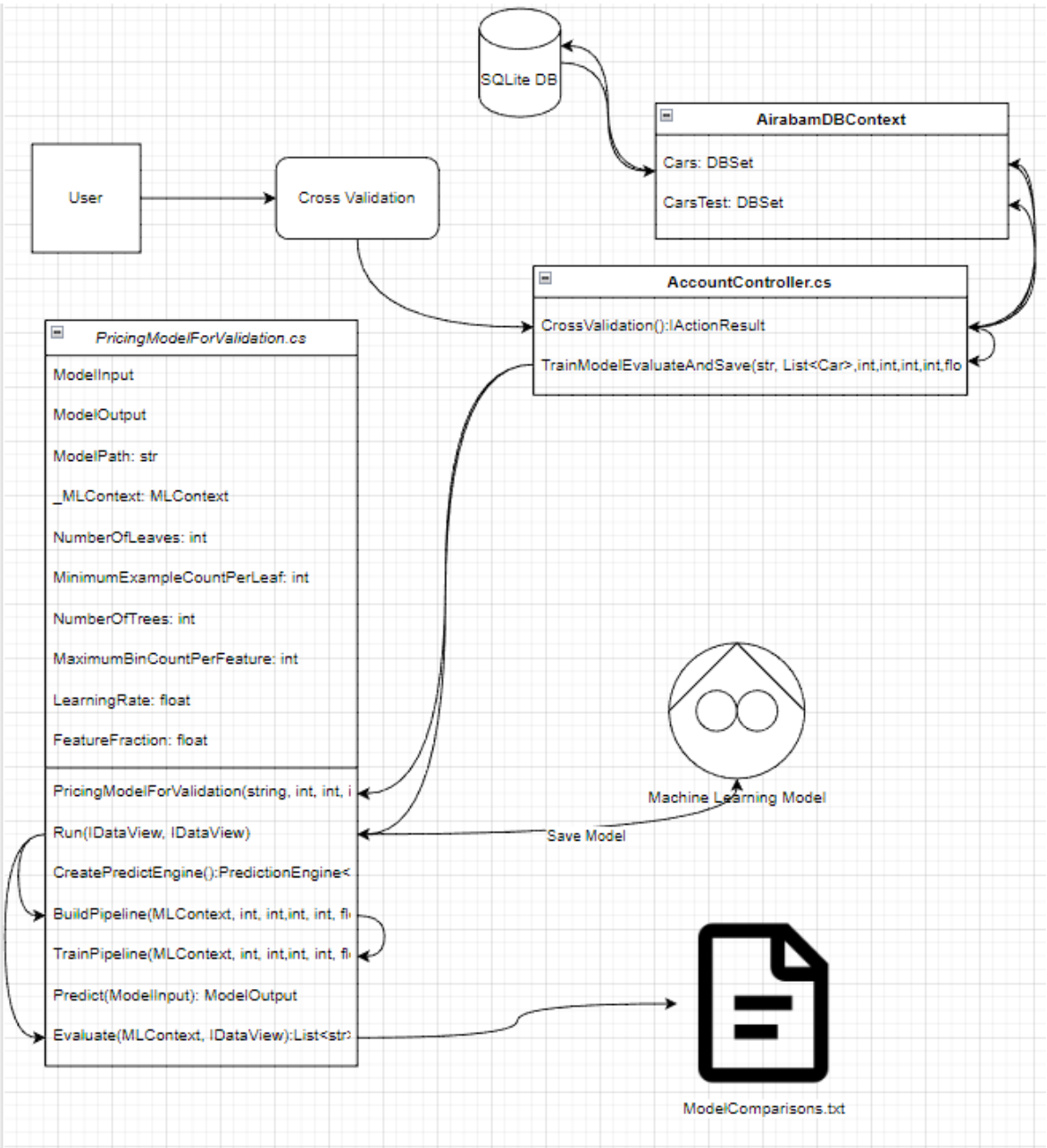


Figure 43 Model Validation System

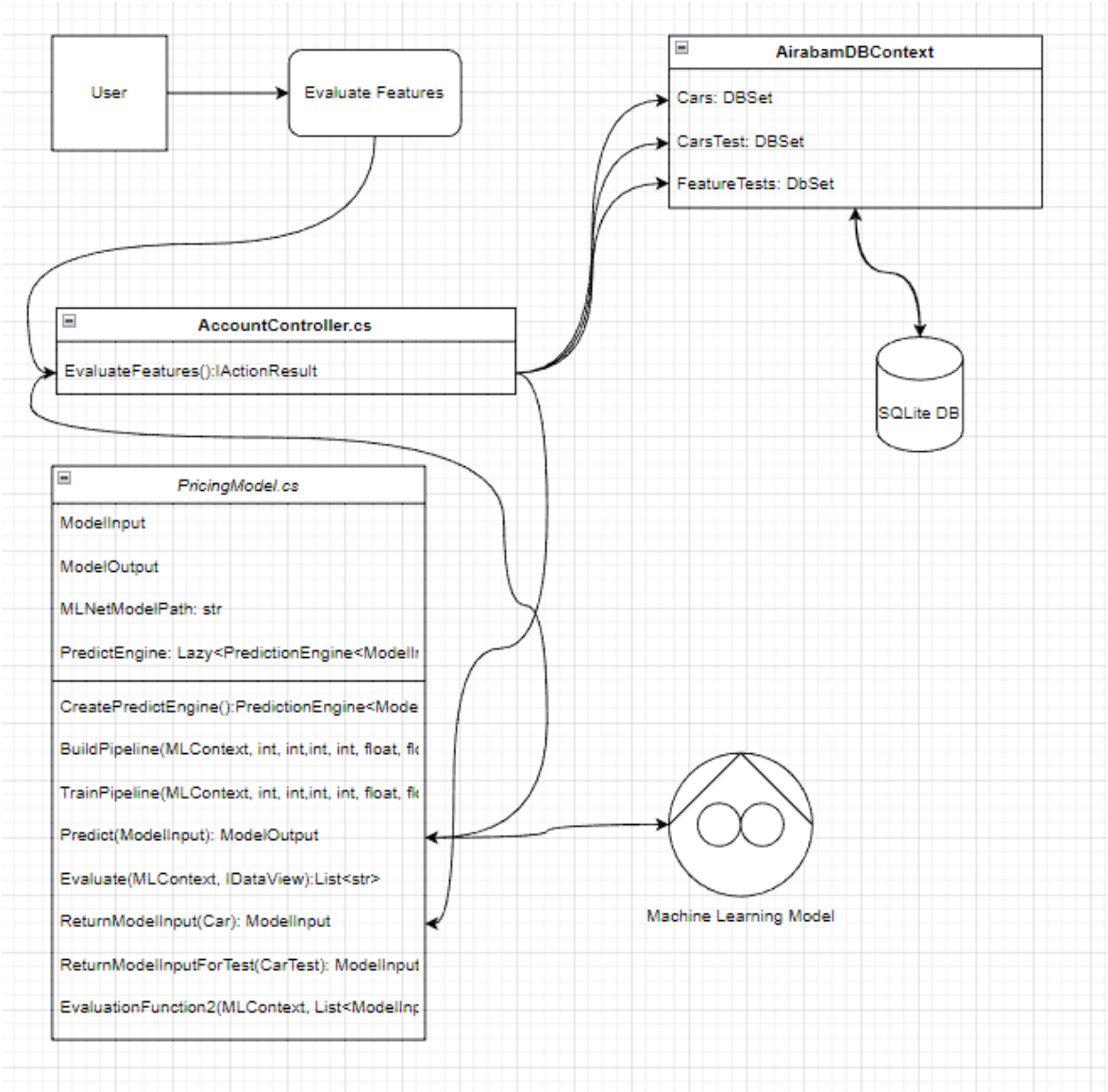


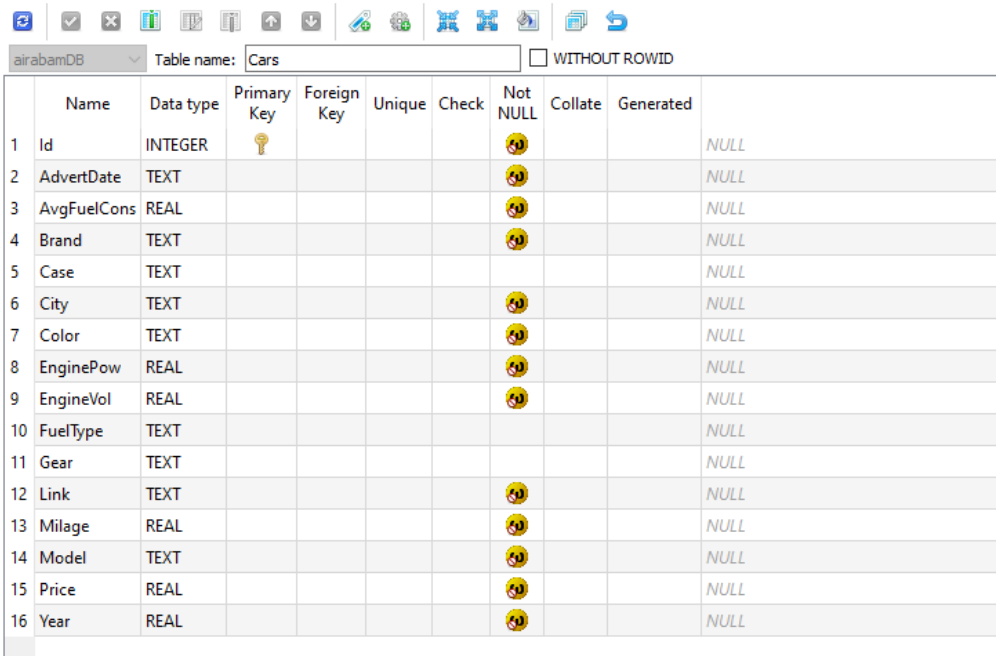
Figure 44 Feature Evaluation System

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.



airabamDB Table name: Cars WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	Id	INTEGER								NULL
2	AdvertDate	TEXT								NULL
3	AvgFuelCons	REAL								NULL
4	Brand	TEXT								NULL
5	Case	TEXT								NULL
6	City	TEXT								NULL
7	Color	TEXT								NULL
8	EnginePow	REAL								NULL
9	EngineVol	REAL								NULL
10	FuelType	TEXT								NULL
11	Gear	TEXT								NULL
12	Link	TEXT								NULL
13	Milage	REAL								NULL
14	Model	TEXT								NULL
15	Price	REAL								NULL
16	Year	REAL								NULL

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.

```
[Authorize]
public IActionResult Scrape()
{

   return RedirectToAction(nameof(Profile));

}
```

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.

```
for (int i = 0; i <= 14; i++)
{

        var response = CallUrl($"https://www.arabam.com/ikinci-
        el/otomobil?days={i}&sort=startedAt.asc").Result;
         HtmlDocument doc = new HtmlDocument();
         doc.LoadHtml(response);
         int counter = 0;
         int existedCounter = 0;
         string nextPage = "";

         var rows = doc.DocumentNode.SelectNodes("//tr[@class='listing-list-item pr
        should-hover bg-white']");

        while (counter > -999)
        {


        }

}
```

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:

```
                    if (counter != 0)
                    {
                        nextPage = $"https://www.arabam.com/ikinci-
el/otomobil?days={i}&sort=startedAt.asc&;take=50&page={counter}";
                        var res = CallUrl(nextPage);
                        if (!res.IsFaulted)
                        {
                            response = res.Result;
                        }
                        else
                        {
                            Console.WriteLine("Bitti");
                            break;
                        }
                        HtmlDocument newDoc = new HtmlDocument();
                        newDoc.LoadHtml(response);
                        rows = newDoc.DocumentNode.SelectNodes("//tr[@class='listing-list-item pr
should-hover bg-white']");
                    }
```

Figure 48 Scrape Method - Step 3

```
foreach (var item in rows)
                    {
                        var model = item.SelectSingleNode("./td[2]/h3/div").InnerHtml;
                        var brand = model.ToString().Split(" ")[0];
                        var year = item.SelectSingleNode("./td[4]/div[1]/a").InnerHtml;
                        var link = "https://www.arabam.com/" +
item.SelectSingleNode("./td[4]/div[1]/a").Attributes[0].Value;
                        var km = item.SelectSingleNode("./td[5]/div[1]/a").InnerHtml;
                        var price =
item.SelectSingleNode("./td[7]/div[1]/a/span").InnerHtml.ToString().Trim().Split(" ")[0];
                        var color = item.SelectSingleNode("./td[6]/div[1]/a").InnerHtml;
                        var city =
item.SelectSingleNode("./td[9]/div[1]/div[1]/a/span[1]").InnerHtml;
                        var date = item.SelectSingleNode("./td[8]/div[1]/a").InnerHtml;

                        Car car = new Car();
                        car.Brand = brand;
                        car.Model = model;
                        car.Year = Convert.ToDouble(year);
                        car.City = city;
                        car.Color = color;
                        car.Link = link;
                        car.AdvertDate = date.ToString();
                        if (km.ToString().Contains('.'))
                        {
                            string fullmilage = "";
                            foreach (var part in km.ToString().Split('.'))
                            {
                                fullmilage += part;
                            }
                            car.Milage = Convert.ToDouble(fullmilage);
                        }
                        else
                        {
                            car.Milage = Convert.ToDouble(km);
                        }
                        if (price.ToString().Contains('.'))
                        {
                            string fullprice = "";
                            foreach (var part in price.ToString().Split('.'))
                            {
                                fullprice += part;
                            }
                            car.Price = Convert.ToDouble(fullprice);
                        }
                        else
                        {
                            car.Price = Convert.ToDouble(price);
                        }

                        var listOfExist = _context.Cars.Where(c => c.Link == car.Link).ToList();
                        if (listOfExist.Count() == 0)
                        {

                            _context.Cars.Add(car);
                            _context.SaveChanges();
                        }
                        else
                        {
                            existedCounter++;
                            if (existedCounter > 200)
                            {
                                counter = -1001;
                            }
                        }
                    }

                    counter++;
```

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.

```
        public IActionResult ScrapeDetails()
        {
            var cars = _context.CarsTest.ToList();
            int counter = 0;
            int carcounter = 0;

            foreach (var car in cars)
            {
                Thread.Sleep(100);
                var link = car.Link;
                string res;
                try
                {
                    res = CallUrl(link).Result;
                }
                catch (Exception)
                {
                    continue;
                }
                HtmlDocument doc = new HtmlDocument();
                doc.LoadHtml(res);
                doc.OptionEmptyCollection = true;
                var rows =
doc.DocumentNode.SelectSingleNode("//html/body/div[2]/div[2]/div[3]/div/div[1]/div[1]/div[2]/ul");
                if (rows == null) continue;
                if (rows.SelectSingleNode("./li[8]/span[2]") == null ||
                    rows.SelectSingleNode("./li[9]/span[2]") == null ||
                    rows.SelectSingleNode("./li[10]/span[2]") == null ||
                    rows.SelectSingleNode("./li[11]/span[2]") == null ||
                    rows.SelectSingleNode("./li[12]/span[2]") == null ||
                    rows.SelectSingleNode("./li[14]/span[2]") == null)
                { continue;}
                var gear = rows.SelectSingleNode("./li[8]/span[2]").InnerHtml.ToString().Trim();
                var fueltype = rows.SelectSingleNode("./li[9]/span[2]").InnerHtml.ToString().Trim();
                var caseType =
rows.SelectSingleNode("./li[10]/span[2]").InnerHtml.ToString().Trim();
                var engVol = rows.SelectSingleNode("./li[11]/span[2]").InnerHtml.ToString().Trim();
                var engPow = rows.SelectSingleNode("./li[12]/span[2]").InnerHtml.ToString().Trim();
                var avgFuelCon =
rows.SelectSingleNode("./li[14]/span[2]").InnerHtml.ToString().Trim();

                string fuelCon = avgFuelCon.Replace("lt", "");
                fuelCon = fuelCon.Replace(',', '.').Trim();

                string motGuc = engPow.Replace("hp", "").Trim();
                string motHac = engVol.Replace("cc", "").Trim();

                double dbl_FuelCon = StrToDouble(fuelCon, -999);
                double dbl_motGuc = StrToDouble(motGuc, -999);
                double dbl_motHac = StrToDouble(motHac, -999);

                car.AvgFuelCons = dbl_FuelCon;
                car.EngineVol = dbl_motHac;
                car.EnginePow = dbl_motGuc;
                car.Gear = gear;
                car.Case = caseType;
                car.FuelType = fueltype;

                counter++;

                if (counter % 1000 == 0)
                {
                    _context.SaveChanges();
                }
            }
            _context.SaveChanges();
            return RedirectToAction(nameof(Profile));
        }
```

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:

```
[Authorize]
public IActionResult FixModelNames() {

    var cars = _context.Cars;
    Dictionary<string, int> modelNames = new Dictionary<string, int>();


    foreach (Car car in cars)
    {
        string modelNam = car.Model;
        var x = modelNam.Split(" ");

        string name = "";

        for (int i = 0; i < 3; i++) {
            if (x[i] == car.Brand || x[i].Contains(".")) {
                continue;
            }
            name += x[i];
        }

        if (!modelNames.ContainsKey(name))
        {

            modelNames.Add(name, 1);
        }
        else {
            modelNames[name]++;
        }

        car.Model = name;
    }

    _context.SaveChanges();
    return RedirectToAction(nameof(Profile));
}
```

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.

```
public static IEstimator<ITransformer> BuildPipeline(MLContext mlContext, int NumberOfLeaves, int
MinimumExampleCountPerLeaf,
            int NumberOfTrees, int MaximumBinCountPerFeature, float LearningRate, float
FeatureFraction)
        {
            // Data process configuration with pipeline data transformations
            var pipeline = mlContext.Transforms.Categorical.OneHotEncoding(new[] {
                                        new InputOutputColumnPair(@"Brand", @"Brand"),
                                        new InputOutputColumnPair(@"City", @"City"),
                                        new InputOutputColumnPair(@"Color", @"Color"),
                                        new InputOutputColumnPair(@"Model", @"Model"),
                                        new InputOutputColumnPair(@"Case", @"Case"),
                                        new InputOutputColumnPair(@"FuelType", @"FuelType"),
                                        new InputOutputColumnPair(@"Gear", @"Gear") })
                                    .Append(mlContext.Transforms.ReplaceMissingValues(new[] {
                                        new InputOutputColumnPair(@"Milage", @"Milage"),
                                        new InputOutputColumnPair(@"Year", @"Year"),
                                        new InputOutputColumnPair(@"AvgFuelCons", @"AvgFuelCons"),
                                        new InputOutputColumnPair(@"EnginePow", @"EnginePow"),
                                        new InputOutputColumnPair(@"EngineVol", @"EngineVol") }))
                                    .Append(mlContext.Transforms.Text.FeaturizeText(@"AdvertDate",
@"AdvertDate"))
                                    .Append(mlContext.Transforms.Concatenate(@"Features", new[] {
                                        @"Brand",
                                        @"City",
                                        @"Color",
                                        @"Model",
                                        @"Case",
                                        @"FuelType",
                                        @"Gear",
                                        @"Milage",
                                        @"Year",
                                        @"AvgFuelCons",
                                        @"EnginePow",
                                        @"EngineVol",
                                        @"AdvertDate" }))
                                    .Append(mlContext.Regression.Trainers.FastTree(new
FastTreeRegressionTrainer.Options() {
                                        NumberOfLeaves = NumberOfLeaves,
                                        MinimumExampleCountPerLeaf = MinimumExampleCountPerLeaf,
                                        NumberOfTrees = NumberOfTrees,
                                        MaximumBinCountPerFeature = MaximumBinCountPerFeature,
                                        LearningRate = LearningRate,
                                        FeatureFraction = FeatureFraction,
                                        LabelColumnName = @"Price",
                                        FeatureColumnName = @"Features" }));

            return pipeline;
        }
```

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.

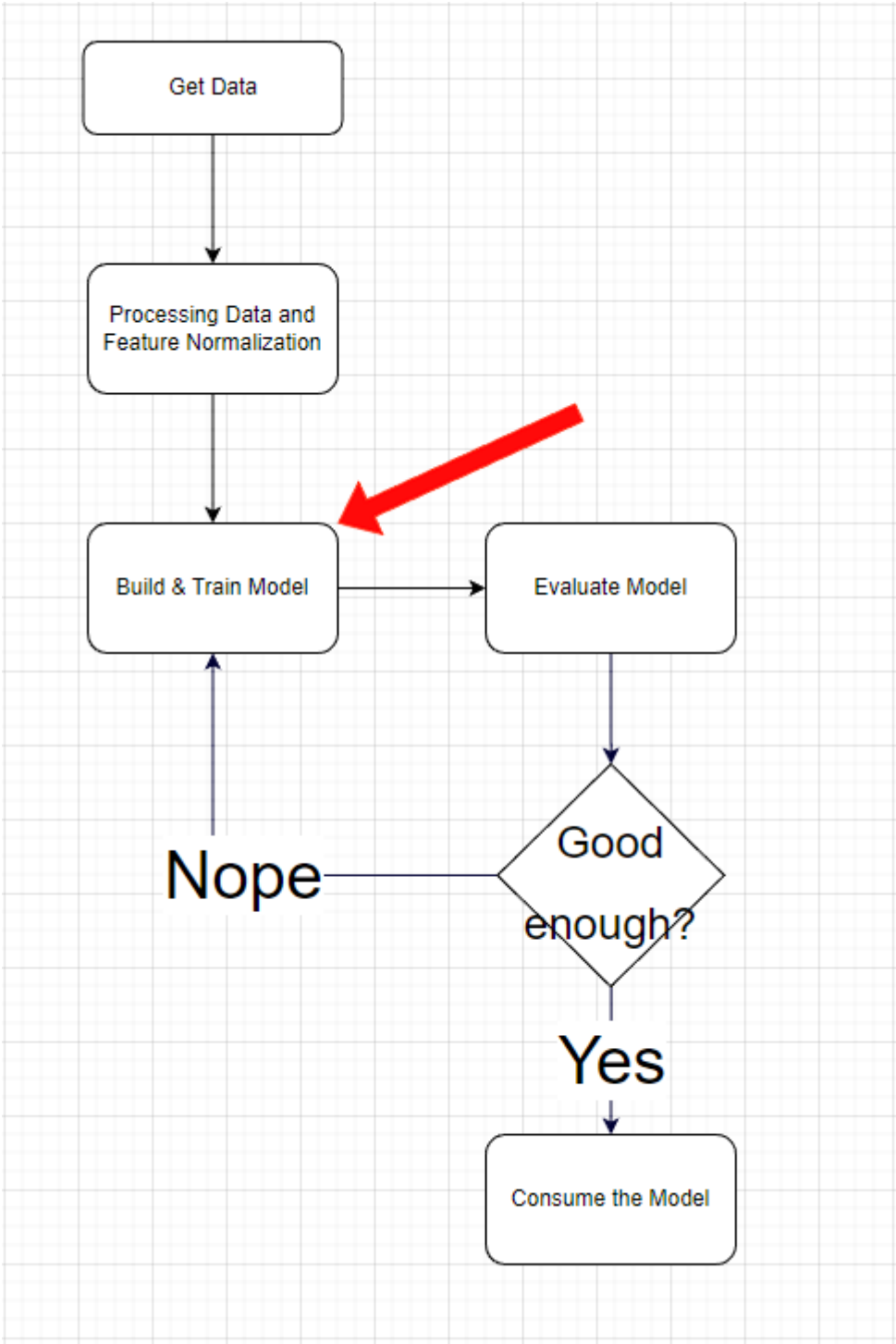


Figure 53 Steps Explained

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:

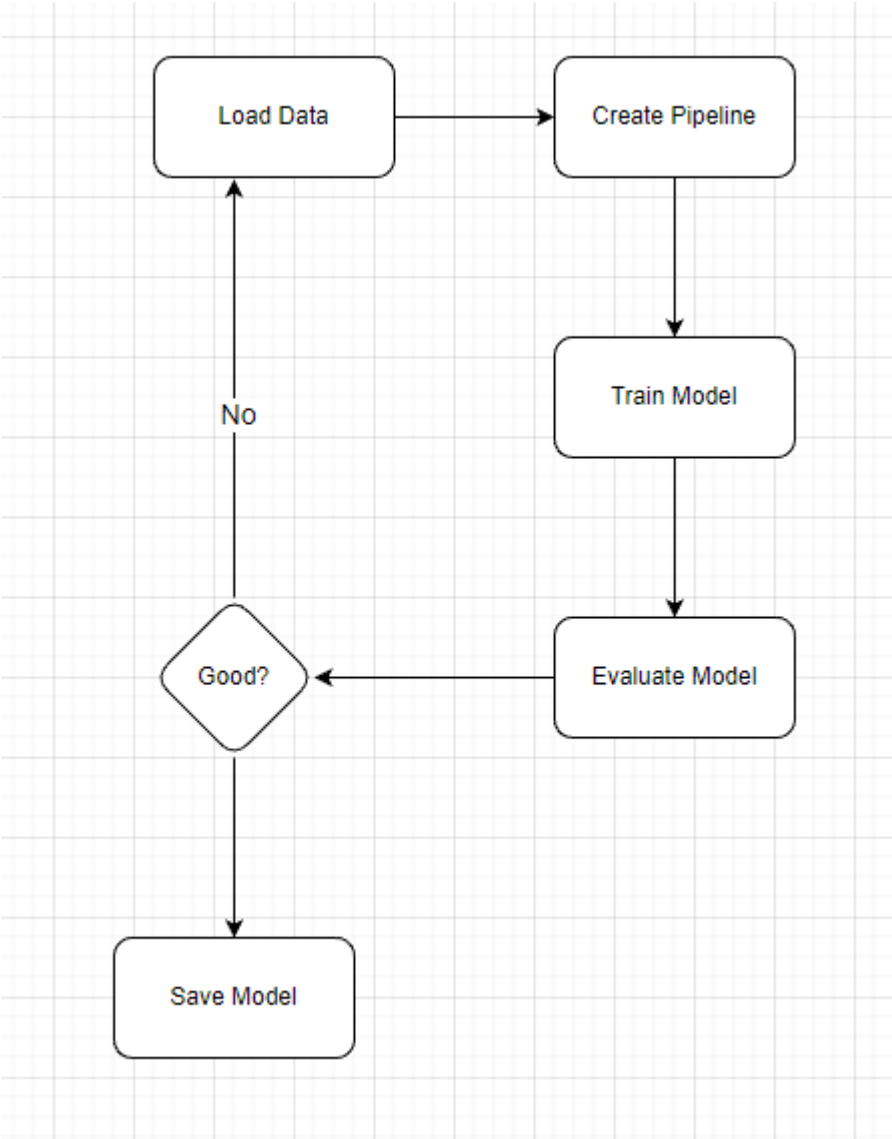


Figure 54 ML.NET Diagram 1

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

What is a pipeline? A machine learning pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve 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.

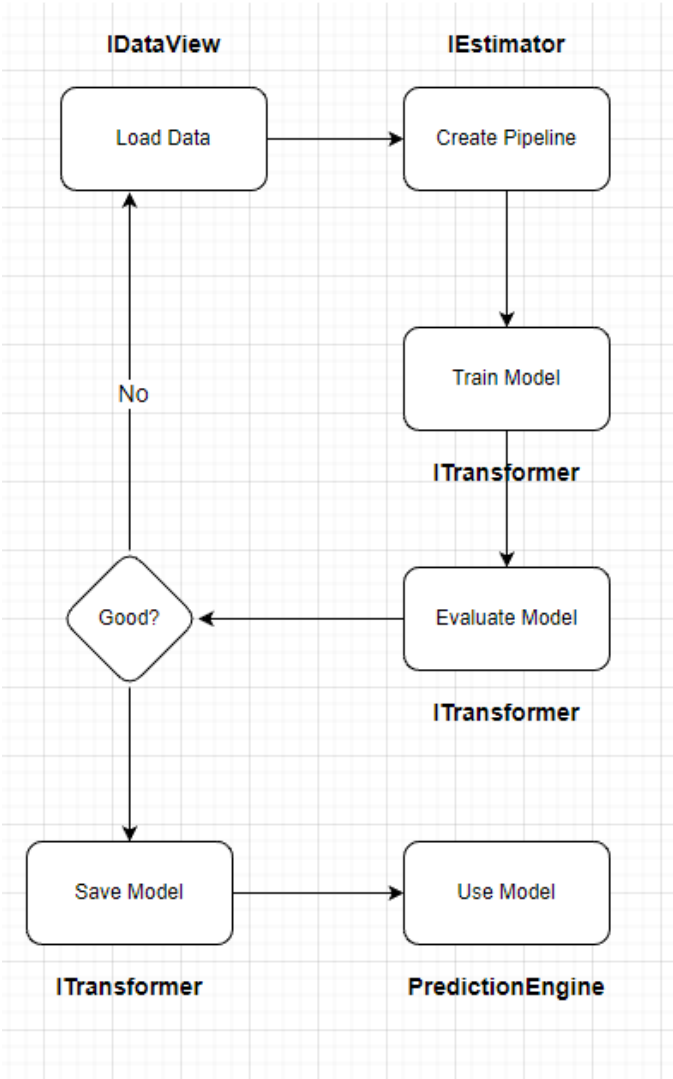


Figure 55 ML.NET Diagram 2

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> 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 instead of DataViewSchema. (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.

```
public class ModelInput
        {
            [ColumnName(@"Id")]
            public float Id { get; set; }

            [ColumnName(@"AdvertDate")]
            public string AdvertDate { get; set; }

            [ColumnName(@"Brand")]
            public string Brand { get; set; }

            [ColumnName(@"City")]
            public string City { get; set; }

            [ColumnName(@"Color")]
            public string Color { get; set; }

            [ColumnName(@"Link")]
            public string Link { get; set; }

            [ColumnName(@"Milage")]
            public float Milage { get; set; }

            [ColumnName(@"Model")]
            public string Model { get; set; }

            [ColumnName(@"Price")]
            public float Price { get; set; }

            [ColumnName(@"Year")]
            public float Year { get; set; }

            [ColumnName(@"AvgFuelCons")]
            public float AvgFuelCons { get; set; }

            [ColumnName(@"Case")]
            public string Case { get; set; }

            [ColumnName(@"EnginePow")]
            public float EnginePow { get; set; }

            [ColumnName(@"EngineVol")]
            public float EngineVol { get; set; }

            [ColumnName(@"FuelType")]
            public string FuelType { get; set; }

            [ColumnName(@"Gear")]
            public string Gear { get; set; }

        }

        public class ModelOutput
        {
            public float Score { get; set; }
        }
```

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.

```
        private static string MLNetModelPath = Path.GetFullPath("Modella.zip");

        public static readonly Lazy<PredictionEngine<ModelInput, ModelOutput>> PredictEngine = new
Lazy<PredictionEngine<ModelInput, ModelOutput>>(() => CreatePredictEngine(), true);
```

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.

```
private static PredictionEngine<ModelInput, ModelOutput> CreatePredictEngine()
        {
            var mlContext = new MLContext();
            ITransformer mlModel = mlContext.Model.Load(MLNetModelPath, out var _);
            return mlContext.Model.CreatePredictionEngine<ModelInput,
ModelOutput>(mlModel);
        }
```

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:

```
public static IEstimator<ITransformer> BuildPipeline(MLContext mlContext, int NumberOfLeaves, int
MinimumExampleCountPerLeaf,
            int NumberOfTrees, int MaximumBinCountPerFeature, float LearningRate, float
FeatureFraction)
        {
            // Data process configuration with pipeline data transformations
            var pipeline = mlContext.Transforms.Categorical.OneHotEncoding(new[] {
                                        new InputOutputColumnPair(@"Brand", @"Brand"),
                                        new InputOutputColumnPair(@"City", @"City"),
                                        new InputOutputColumnPair(@"Color", @"Color"),
                                        new InputOutputColumnPair(@"Model", @"Model"),
                                        new InputOutputColumnPair(@"Case", @"Case"),
                                        new InputOutputColumnPair(@"FuelType", @"FuelType"),
                                        new InputOutputColumnPair(@"Gear", @"Gear") })
                                    .Append(mlContext.Transforms.ReplaceMissingValues(new[] {
                                        new InputOutputColumnPair(@"Milage", @"Milage"),
                                        new InputOutputColumnPair(@"Year", @"Year"),
                                        new InputOutputColumnPair(@"AvgFuelCons", @"AvgFuelCons"),
                                        new InputOutputColumnPair(@"EnginePow", @"EnginePow"),
                                        new InputOutputColumnPair(@"EngineVol", @"EngineVol") }))
                                    .Append(mlContext.Transforms.Text.FeaturizeText(@"AdvertDate",
@"AdvertDate"))
                                    .Append(mlContext.Transforms.Concatenate(@"Features", new[] {
                                        @"Brand",
                                        @"City",
                                        @"Color",
                                        @"Model",
                                        @"Case",
                                        @"FuelType",
                                        @"Gear",
                                        @"Milage",
                                        @"Year",
                                        @"AvgFuelCons",
                                        @"EnginePow",
                                        @"EngineVol",
                                        @"AdvertDate" }))
                                    .Append(mlContext.Regression.Trainers.FastTree(new
FastTreeRegressionTrainer.Options() {
                                        NumberOfLeaves = NumberOfLeaves,
                                        MinimumExampleCountPerLeaf = MinimumExampleCountPerLeaf,
                                        NumberOfTrees = NumberOfTrees,
                                        MaximumBinCountPerFeature = MaximumBinCountPerFeature,
                                        LearningRate = LearningRate,
                                        FeatureFraction = FeatureFraction,
                                        LabelColumnName = @"Price",
                                        FeatureColumnName = @"Features" }));

            return pipeline;
        }
```

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:

```
        public static ITransformer TrainPipeline(MLContext context, IDataView
trainData)
        {
            var pipeline = BuildPipeline(context);
            var model = pipeline.Fit(trainData);

            return model;
        }
```

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:

```
public static ModelOutput Predict(ModelInput input)
        {
            var predEngine = PredictEngine.Value;
            return predEngine.Predict(input);
        }
```

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?

```
        public List<String> Evaluate(MLContext mlContext,  IDataView testData)
        {
            var list = new List<String>();
            ITransformer model = mlContext.Model.Load(MLNetModelPath, out var _);
            var predictions = model.Transform(testData);
            var metrics = mlContext.Regression.Evaluate(predictions, "Price",
"Score");

            Console.WriteLine();
            Console.WriteLine($"*************************************************");
            Console.WriteLine($"*       Model quality metrics evaluation         ");
            Console.WriteLine($"*------------------------------------------------");
            Console.WriteLine($"*       RSquared Score:
{metrics.RSquared:0.##}");
            Console.WriteLine($"*       MeanSq Error Score:
{metrics.MeanSquaredError:0.##}");
            Console.WriteLine($"*       Root Mean Squared Error:
{metrics.RootMeanSquaredError:#.##}");
            list.Add($"*       RSquared Score:      {metrics.RSquared:0.##}");
            list.Add($"*       Root Mean Squared Error:
{metrics.RootMeanSquaredError:#.##}");
            return list;
        }
```

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.

```
public static void EvaluationFunction2(MLContext mLContext, List<ModelInput>
carsList) {

            ITransformer model = mLContext.Model.Load(MLNetModelPath, out var _);

            List<Tuple<float, float>> tuples = new List<Tuple<float, float>>();

            float sumOfDifferences = 0;

            foreach (var car in carsList) {
                float score = Predict(car).Score;
                float price = car.Price;

                sumOfDifferences += Math.Abs(score - price);

                tuples.Add(new Tuple<float, float>(price, score));
            }

            foreach (var tuple in tuples) {
                Console.WriteLine(tuple.Item1 + " | " + tuple.Item2 + " | " + "
difference : " + (Math.Abs(tuple.Item1 - tuple.Item2)));
            }

            Console.WriteLine("Average Difference: " +
(sumOfDifferences/carsList.Count));

        }
```

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.

```
public static ModelInput ReturnModelInput(Car car)
{
    ModelInput model = new ModelInput
    {
        Brand = car.Brand,
        Model = car.Model,
        City = car.City,
        Color = car.Color,
        Year = (float)car.Year,
        Milage = (float)car.Milage,
        Price = (float)car.Price,
        AdvertDate = car.AdvertDate,
        Gear = car.Gear,
        Case = car.Case,
        AvgFuelCons = (float)car.AvgFuelCons,
        FuelType = car.FuelType,
        EnginePow = (float)car.EnginePow,
        EngineVol = (float)car.EngineVol
    };

    return model;
}

public static ModelInput ReturnModelInputForTest(CarTest car)
{
    ModelInput model = new ModelInput
    {
        Brand = car.Brand,
        Model = car.Model,
        City = car.City,
        Color = car.Color,
        Year = (float)car.Year,
        Milage = (float)car.Milage,
        Price = (float)car.Price,
        AdvertDate = car.AdvertDate,
        Gear = car.Gear,
        Case = car.Case,
        AvgFuelCons = (float)car.AvgFuelCons,
        FuelType = car.FuelType,
        EnginePow = (float)car.EnginePow,
        EngineVol = (float)car.EngineVol
    };

    return model;
}
```

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.

```
[Authorize]
 public IActionResult TrainPrice()
 {
     var cars = _context.Cars;
     MLContext mLContext = new MLContext();
     List<ModelInput> miList = new List<ModelInput>();

     foreach (Car car in cars)
     {
         miList.Add(PricingModel.ReturnModelInput(car));
     }

     IDataView dw = mLContext.Data.LoadFromEnumerable(miList);
     var model = PricingModel.TrainPipeline(mLContext, dw, 475, 2, 32768,
600, 0.19f, 0.75f);

     mLContext.Model.Save(model, dw.Schema, "Modella.zip");

     //var car1 = _context.Cars.Find(1);
     //var x  = PricingModel.Predict(ReturnModelInput(car1));

     return RedirectToAction(nameof(Profile));
 }
```

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.

```
        private static string ModelPath;
        private static MLContext _MLContext;
        private static int NumberOfLeaves;
        private static int MinimumExampleCountPerLeaf;
        private static int NumberOfTrees;
        private static int MaximumBinCountPerFeature;
        private static float LearningRate;
        private static float FeatureFraction;

        public PricingModelForValidation(string modelPath, int numberOfLeaves, int
minimumExampleCountPerLeaf, int numberOfTrees, int maximumBinCountPerFeature, float
learningRate, float featureFraction, MLContext mLContext)
        {
            ModelPath = modelPath;
            NumberOfLeaves = numberOfLeaves;
            MinimumExampleCountPerLeaf = minimumExampleCountPerLeaf;
            NumberOfTrees = numberOfTrees;
            MaximumBinCountPerFeature = maximumBinCountPerFeature;
            LearningRate = learningRate;
            FeatureFraction = featureFraction;
            _MLContext = mLContext;
        }

        public void Run(IDataView trainData, IDataView testData)
        {

            var model = TrainPipeline(_MLContext, trainData);
            var messagelist = Evaluate(_MLContext, testData,model);

            string toAppend = $"{ModelPath} \r\n \r\n {messagelist[0]} \r\n
{messagelist[1]} \r\n \r\n";
            System.IO.File.AppendAllText("modelComparisons.txt", toAppend);

            _MLContext.Model.Save(model, trainData.Schema, ModelPath);
        }
```

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:

```
public IActionResult CrossValidation()
        {
            var cars = _context.Cars;

            /** MODEL A
             * NumberOfLeaves = 100;
             * MinimumExampleCountPerLeaf = 10;
             * NumberOfTrees = 100;
             * MaximumBinCountPerFeature = 100;
             * LearningRate = 0.15;
             * FeatureFraction = 0.7;
             * **/

            TrainModelEvaluateAndSave("ModelA.zip",
cars.ToList(),100,10,100,100,0.15f,0.7f);

       }
```

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:

```
68850 | 56959.49 |  difference : 11890.512
336000 | 405654.34 |  difference : 69654.34
360000 | 337930.72 |  difference : 22069.281
204000 | 220392.34 |  difference : 16392.344
181000 | 151831.56 |  difference : 29168.438
235000 | 226733.1 |  difference : 8266.906
172000 | 185082.83 |  difference : 13082.828
212000 | 191979.78 |  difference : 20020.219
210000 | 194134.62 |  difference : 15865.375
270000 | 230559 |  difference : 39441
835000 | 682745.1 |  difference : 152254.88
410000 | 367848.12 |  difference : 42151.875
182500 | 151467.11 |  difference : 31032.89
155000 | 132347.53 |  difference : 22652.469
215000 | 199372.6 |  difference : 15627.406
245900 | 255134.62 |  difference : 9234.625
155000 | 169411.06 |  difference : 14411.0625
230000 | 208402.69 |  difference : 21597.312
169500 | 158930.92 |  difference : 10569.078
215000 | 184295.92 |  difference : 30704.078
217500 | 163178.83 |  difference : 54321.17
145000 | 103683.85 |  difference : 41316.15
475000 | 455507.78 |  difference : 19492.219
536500 | 501480.28 |  difference : 35019.72
830000 | 700856.75 |  difference : 129143.25
275000 | 248101.17 |  difference : 26898.828
6500000 | 1530007.8 |  difference : 4969992
319500 | 346356.34 |  difference : 26856.344
Average Difference: 65292.777
```

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 65 thousand Turkish Liras.

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.

```
[Authorize]
 public IActionResult EvaluateFeatures()
 {
     var cars = _context.Cars.ToList();

     var top10Models = _context.Cars.GroupBy(i => i.Model)
              .Where(x => x.Count() > 1).OrderByDescending(x => x.Count())
              .Select(val => val.Key).Take(10).ToList();


     var exampleCars = exampleCarsForFeatureImportance(cars, top10Models);

     var featuresAndValues = GetFeaturePriceDictionary(exampleCars);

     foreach (var item in _context.FeatureTests)
     {
         _context.FeatureTests.Remove(item);
     }

     _context.SaveChanges();

     _context.FeatureTests.AddRange(featuresAndValues);

     _context.SaveChanges();

     ViewBag.Message = "Features are evaluated!";

     return RedirectToAction(nameof(Profile));
 }
```

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.

```
public List<Car> exampleCarsForFeatureImportance(List<Car> cars, List<string> top10Models) {
    List<Car> result = new List<Car>();
    foreach (var model in top10Models) {
        var carsWithModel = cars.Where(x=>x.Model == model).ToList();
        var MostRepeatedGear = carsWithModel.GroupBy(q => q.Gear)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedYear = carsWithModel.GroupBy(q => q.Year)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedAvgFuel = carsWithModel.GroupBy(q => q.AvgFuelCons)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedBrand = carsWithModel.GroupBy(q => q.Brand)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedCase = carsWithModel.GroupBy(q => q.Case)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedCity = carsWithModel.GroupBy(q => q.City)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedColor = carsWithModel.GroupBy(q => q.Color)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedEnginePow = carsWithModel.GroupBy(q => q.EnginePow)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedEngineVol = carsWithModel.GroupBy(q => q.EngineVol)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedFuelType = carsWithModel.GroupBy(q => q.FuelType)
                            .OrderByDescending(gp => gp.Count())
                            .Take(1)
                            .Select(g => g.Key).ToList().First();
        var MostRepeatedMilage = carsWithModel.Select(q => q.Milage).Average();

        var MostRepeatedPrice = carsWithModel.Select(q => q.Price).Average();

        Car exampleCar = new Car();
        exampleCar.Model = model;
        exampleCar.Year = MostRepeatedYear;
        exampleCar.Brand = MostRepeatedBrand;
        exampleCar.Price = MostRepeatedPrice;
        exampleCar.City = MostRepeatedCity;
        exampleCar.Gear = MostRepeatedGear;
        exampleCar.Milage = MostRepeatedMilage;
        exampleCar.FuelType = MostRepeatedFuelType;
        exampleCar.EnginePow = MostRepeatedEnginePow;
        exampleCar.EngineVol = MostRepeatedEngineVol;
        exampleCar.Color = MostRepeatedColor;
        exampleCar.Case = MostRepeatedCase;
        exampleCar.AvgFuelCons = MostRepeatedAvgFuel;

        result.Add(exampleCar);

    }

    return result;
}
```

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

Last 10 Brands

#	Name
1	Proton
2	Fiat
3	Tata
4	Chery
5	Geely
6	Chevrolet
7	Dacia
8	Lancia
9	Hyundai
10	Rover

Top 10 Brands

#	Name
1	Saab
2	Honda
3	Infiniti
4	Porsche
5	Volvo
6	MINI
7	Volkswagen
8	Audi
9	Mercedes
10	BMW

Figure 71 Last 10 and Top 10 Brands

5.2 Top 10 and Last 10 Models

Last 10 Models

#	Name
1	Meriva
2	Q50
3	Clio
4	C2
5	SX4GL
6	Attrage
7	Maxima
8	Evanda
9	Symbol
10	106Quicksilver

Top 10 Models

#	Name
1	106XSi
2	Bora
3	Romeo159
4	AutomobilesDS5
5	PassatVariant
6	SuperB
7	Optima
8	NewBeetle
9	Passat
10	Benz

Figure 72 Last 10 and Top 10 Models

5.3 Impact of Average Fuel Consumption on the Price

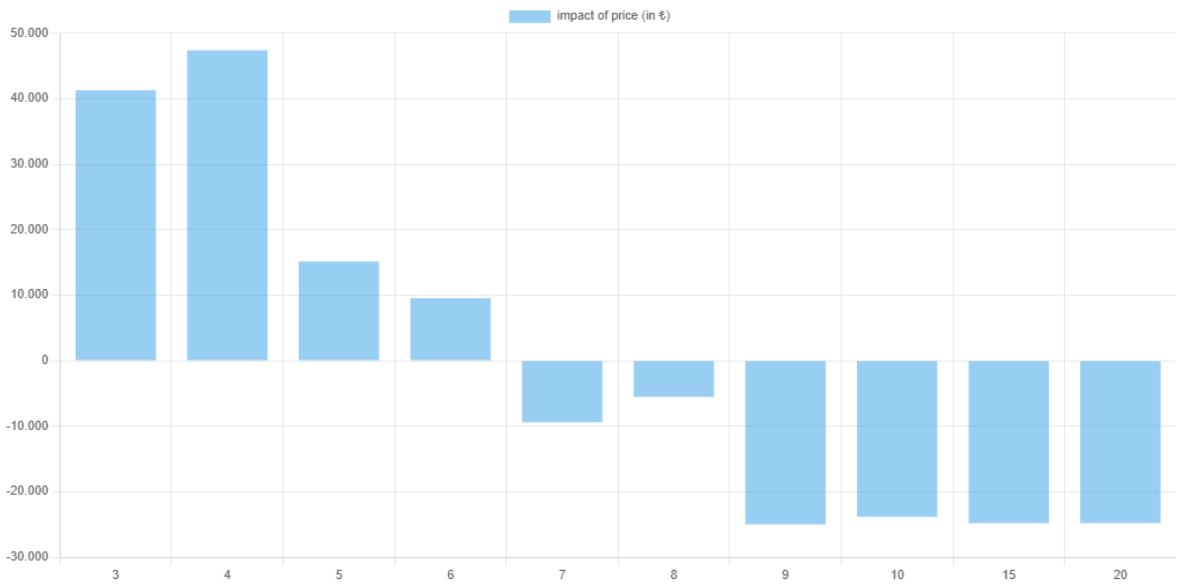


Figure 73 Average Fuel Consumption's Impact on the Price

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

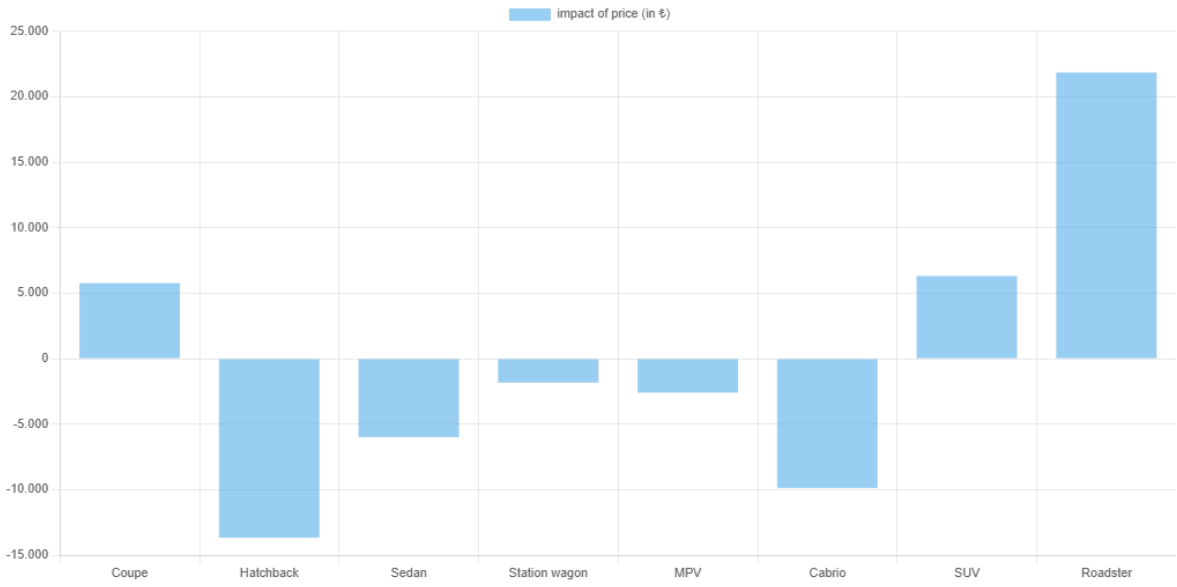


Figure 74 Impact of Car Case on the Price

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

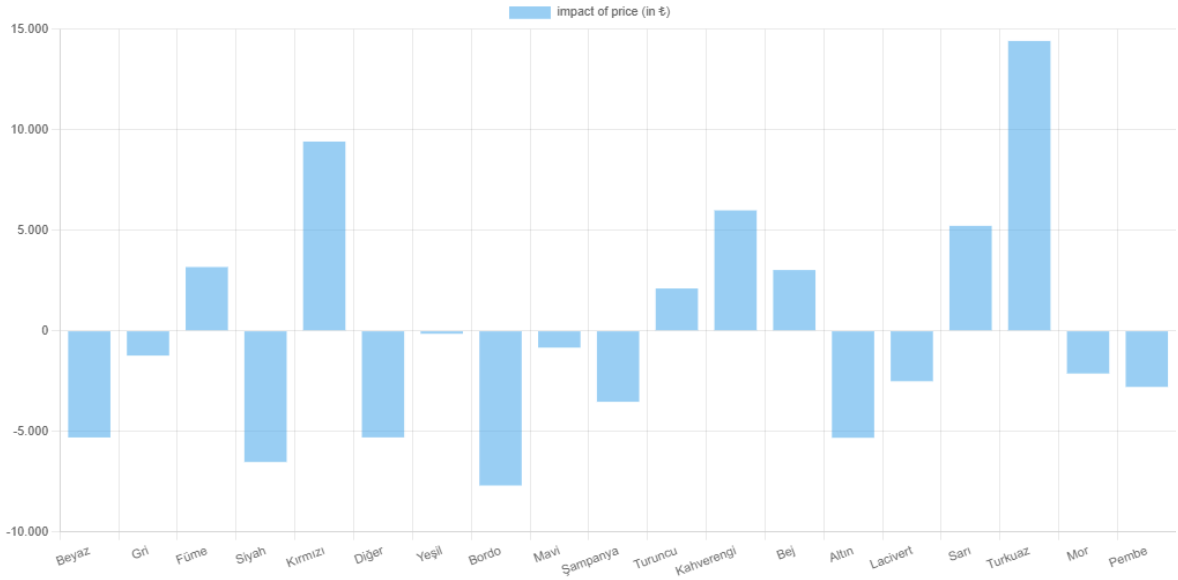


Figure 75 Impact of Color on the Price

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

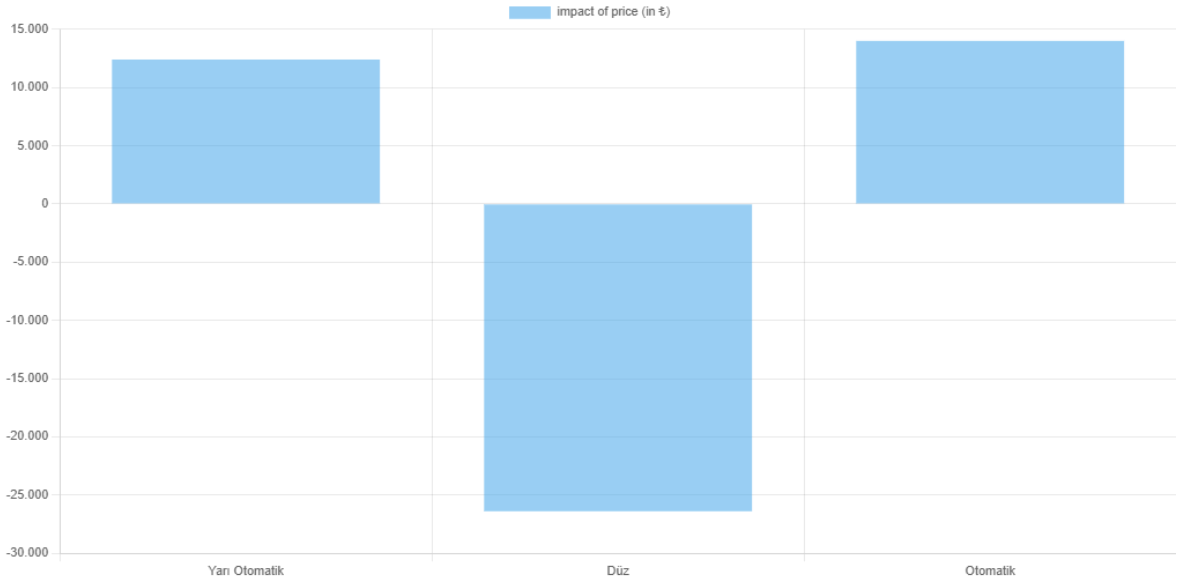


Figure 76 Impact of Gear Type on the Price

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

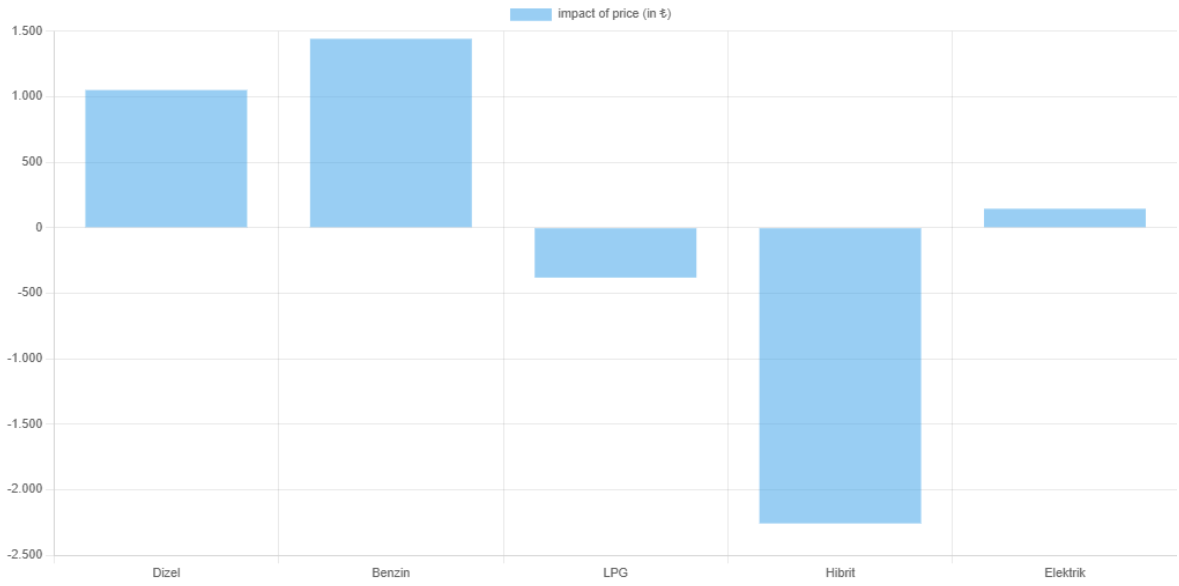


Figure 78 Impact of Fuel Type on the Price

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