**PREDICTING SECOND-HAND CAR PRICES WITH MACHINE LEARNING TECHNIQUES**

**by**

**Emirhan Bal**

**Graduation Project Report**

**Yeditepe University**

**Applied Sciences School**

**Department of Software Development**

**2023**

**Predicting Second-Hand Car Prices with Machine Learning Techniques**

APPROVED BY:

Assist.Prof.Dr. Name Surname Engin Kandıran  
(Supervisor)

Assoc.Prof.Dr. Name Surname ………………………………

Prof.Dr. Name Surname ………………………………

DATE OF APPROVAL: 19 / 05 / 2023

**ACKNOWLEDGEMENTS**

First of all I would like to thank my advisor Assist. Prof. Dr. Engin Kandıran for his guidance and support throughout my project. His expertise and assistance have greatly contributed to the success of this project.

ABSTRACT

Predicting Second-Hand Car Prices with Machine Learning Techniques

This project aims to provide users with the current market value of their desired vehicle by collecting data from second-hand vehicle classified websites such as Arabam.com. The price and feature information of vehicles are scraped and stored in a MongoDB database. Users can select vehicles based on criteria like brand, model, and year, and compare their estimated market value with the listing prices. Additionally, relevant listings matching the user's inputted specifications are presented.

The project utilizes web scraping techniques with Selenium to automatically gather vehicle listings. The collected data is processed using Python in the Jupyter Notebook environment and then stored in the database.

Next, a prediction model is built. The model is trained, validated, and evaluated. The processed data is fed into the model, which makes predictions for our target variable.

The project utilizes the Flask framework to provide users with an interactive web experience.

As a result, users no longer need to browse through multiple listings to determine the market value of their vehicles. This application automates the process for users. Additionally, users can also see the best listings that match their desired specifications.

ÖZET

Makine Öğrenmesi Teknikleri ile İkinci El Araç Fiyat Tahmin Modeli

Bu proje, ikinci el araç ilan sitelerinden alınan verilerle kullanıcılara istedikleri aracın güncel piyasa değerini sunmayı hedeflemektedir. Arabam.com gibi ikinci el araç platformlarından araçların fiyat ve özellik bilgileri toplanarak Mongo DB veritabanına kaydedilir. Kullanıcılar, istedikleri marka, model ve yıl gibi kriterlere sahip araçları seçebilir ve bu araçların tahmini piyasa değerini ilan fiyatlarıyla karşılaştırabilirler. Ayrıca, kullanıcının girdiği özelliklere sahipen uygun ilanlar da kullanıcıya sunulmaktadır.

Proje, Selenium ile web scraping teknikleri kullanarak araç ilanlarını otomatik olarak toplar. Toplanan veriler Jupyter Notebook ortamında Python dili ile düzenlenir ve düzenlenen veriler veritabanına kaydedilir.

Ardından, bir tahmin modeli oluşturulur. Bu model eğitilir, doğrulanır ve değerlendirilir. Düzenlenen veriler bu modele sokulur ve model hedef değişkenimiz için tahmin yapar.

Proje, Flask framework’ü kullanarak kullanıcının web ortamında etkin bir deneyim yaşamasını sağlar.

Bu sayede kullanıcılar, araçlarının piyasa değerini öğrenmek için birden fazla ilanı dolaşmak zorunda kalmazlar. Bu uygulama kullanıcılar için bunu otomatik şekilde yapmaktadır. Ayrıca kullanıcı istediği özelliklere sahip en iyi ilanları da görebilecektir.

**TABLE OF CONTENTS**

[1. INTRODUCTION 1](#_Toc388363258)

[2. BACKGROUND 3](#_Toc388363259)

[2.1. Previous works 3](#_Toc388363260)

[3. ANALYSIS 6](#_Toc388363261)

[3.1. Character Fonts 6](#_Toc388363262)

[3.2. Spacing 6](#_Toc388363263)

[3.3. Left Adjusting 6](#_Toc388363264)

[3.4. Margins 7](#_Toc388363265)

[3.5. Pagination 7](#_Toc388363266)

[3.6. Headings 9](#_Toc388363267)

[3.6.1. Main Headings 9](#_Toc388363268)

[3.6.2. Second Headings 9](#_Toc388363269)

[3.6.3. First Subheadings 10](#_Toc388363270)

[3.6.4. Second Subheadings 10](#_Toc388363271)

[3.7. Footnotes 11](#_Toc388363272)

[3.8. Bibliographical Material 11](#_Toc388363273)

[3.9. Specially Designated Expressions 12](#_Toc388363274)

[3.10. Tables and Figures 12](#_Toc388363275)

[4. DESIGN AND IMPLEMENTATION 14](#_Toc388363276)

[4.1. Title Page 14](#_Toc388363277)

[4.2. Approval Page 14](#_Toc388363278)

[4.3. Dedication 14](#_Toc388363279)

[4.4. Acknowledgements 14](#_Toc388363280)

[4.5. Abstract 15](#_Toc388363281)

[4.6. Table of Contents, List of Figures, Tables and Symbols / Abbreviations 15](#_Toc388363282)

[4.7. Text 15](#_Toc388363283)

[4.8. Appendices 16](#_Toc388363284)

[5. TEST AND RESULTS 17](#_Toc388363285)

[5.1. Typesetting 17](#_Toc388363286)

[5.2. Paper Quality 17](#_Toc388363287)

[5.3. Printer 17](#_Toc388363288)

[5.4. Reproduction 17](#_Toc388363289)

[5.5. Binding 18](#_Toc388363290)

[6. CONCLUSION 20](#_Toc388363291)

[6.1. Use of Copyrighted Material 20](#_Toc388363292)

[6.2. Publications 20](#_Toc388363293)

[Bibliography 21](#_Toc388363294)

[APPENDIX A: SAMPLE PAGES 22](#_Toc388363295)

[APPENDIX B: FORMAT OF DISKETTE, CD or DVD CONTAINING COMPUTER SOFTWARE 23](#_Toc388363296)

**LIST OF FIGURES**

[**Figure 1.1** Lorem ipsum dolor 1](#_Toc365968127)

[**Figure 2.1** Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nam cursus. Morbi ut mi. Nullam enim 2](#_Toc365968128)

**LIST OF TABLES**

[**Table 1.1** Pellentesque habitant morbi tristique senectus 1](#_Toc365968940)

[**Table 1.2** Pellentesque faucibus. Ut accumsan ultricies elit. Maecenas at justo id velit placerat molestie 2](#_Toc365968941)

**LIST OF SYMBOLS / ABBREVIATIONS**

OCR Optical Character Recognition

MATLAB Matrix Laboratory

MAE Mean Absolute Error

RMSE Root Mean Squared Error

MSE Mean Squared Error

R2 Adjusted R squared

 Geometric moment of order (p+q)

f(x,y) Deterministic real image intensity function

 Centralized moment of order (p+q)

 Normalized moment of order (p+q)

I Extended moment invariant

Pt Empirical distribution

Xf Predictor

# INTRODUCTION

Second-hand vehicle buying and selling websites have become an important alternative for many people who want to own a vehicle or sell their existing ones. The traditional approach of selecting and purchasing a vehicle from a dealership or car markets is now outdated. Instead, both buyers and sellers are utilizing various online classified websites. Arabam.com, Sahibinden, Otokoç are some of the leading and actively used second-hand vehicle classified websites in Turkey. These platforms allow us to access thousands of listings without leaving our place and find quality vehicles at more affordable prices.

Second-hand vehicle trading websites provide users with access to a wide range of vehicles and facilitate acquiring market information. However, along with these benefits, there are certain disadvantages. Due to the influence of the free market, vehicle prices are determined based on supply and demand, leading to price fluctuations and often overvaluation. Consequently, it becomes challenging for users to analyze this large amount of data and obtain accurate results regarding the actual value of cars.

Evaluating the second-hand car market is a complex process. The value of a car can be influenced by numerous factors, including brand, model, production year, mileage, technical condition, features, and market demand. The intricate relationships and market dynamics among these factors make it difficult to accurately determine a car's value.

In Turkey, there have been studies conducted to accurately determine the value of a car. However, due to competition among classified websites, vehicle valuations are not directly provided to users but mostly based on the "bidding" system. Although some classified websites provide direct results, they can only rely on their own database, limiting the ability to make comprehensive assessments of the overall market.

Users who wish to sell their vehicles often need to browse through numerous listings on classified websites, comparing them with their own vehicle's specifications to get an idea of its worth. Alternatively, they may end up selling their cars to dealerships below the market value. This process requires time and effort.

Furthermore, the prices in listings are subject to daily fluctuations. In such an environment, obtaining reliable results becomes quite challenging. Artificial intelligence and machine learning can provide much easier and more accurate answers to questions such as "What is the market value of my car?" and "What is the market value of the car I want to buy?"

This project aims to develop an application that will assist users in easily determining the value of a vehicle by collecting data from second-hand vehicle classified websites. The target audience of the project includes both vehicle owners and those who want to become vehicle owners. In our country, a car is not just a means of transportation but also an investment. Therefore, the buying and selling of vehicles play an even more crucial role. The market value of cars purchased as investments is of great importance to vehicle owners. Hence, the project has a much wider target audience than initially assumed.

This project retrieves data dynamically from second-hand vehicle trading websites through Python frameworks and libraries. The collected data is stored in the project's own database. Parameters are determined to better understand the data and create a suitable model. Data visualizations are created using these parameters, making it easier to comprehend and interpret the available data. Subsequently, changes are made to the data to prepare it for analysis. Outliers are detected, and missing values are handled.

Encoding is applied to categorical values to make them compatible with machine learning algorithms. A suitable model is selected for the preprocessed data. The selection is based on the available data and the target variable. The scores of error metrics such as R2, MAE, RMSE are evaluated, and the model with the highest performance is chosen. The selected model is trained and validated. Afterward, the model is tested and evaluated. Following these steps, the model becomes capable of making predictions for new inputs.

The model is then deployed to the web environment using the Flask framework. The application, located on the localhost, is made accessible for global use through "Render."

This project contributes to the vehicle market and its target audience in terms of time, effort, and accuracy. Users will be able to gain insights about the market without the need to browse through dozens of websites and hundreds of listings. Consequently, they will no longer have to rely on dealership visits or pay fees unnecessarily.

# BACKGROUND

## Previous works

The second-hand car market has become a significant alternative for many individuals who aspire to own a car or sell their existing vehicles. While this market offers the opportunity to access quality vehicles at more affordable prices, accurately determining the true market value of these cars has become a major challenge. Evaluating the value of cars holds vital importance for both buyers and sellers, as transactions conducted without proper assessment can pose significant risks.

Assessing the second-hand car market is a complex process. The value of a car can be influenced by numerous factors, including brand, model, production year, mileage, technical condition, features, and market demand. The intricate relationships among these factors and the dynamics within the market make it challenging to accurately determine the value of a car.

The rapid advancement of technology has also had a profound impact on the second-hand car market. The internet and online listing platforms have provided users with access to a wide range of vehicles and facilitated the acquisition of market information. However, sifting through this vast amount of data and obtaining accurate information regarding the true value of cars remains a challenge. In this regard, leveraging technology effectively and applying data analysis methods can offer a significant advantage.

Evaluating the value of second-hand cars using traditional methods can come with various disadvantages. Conducting manual research, visiting multiple listing platforms, or relying on expert opinions can be time-consuming and laborious. The abundance of data across numerous listing sites can make it difficult to access accurate information, leading to wasted time. Furthermore, the timeliness and accuracy of the information provided in listings can be questionable. Depending solely on expert opinions may be influenced by personal preferences or biases and may not provide an impartial evaluation. Therefore, a project that utilizes technological solutions and applies data analytics provides a more objective and reliable car valuation process, eliminating the disadvantages of traditional methods.

The aim of this project is to streamline the process of determining the value of second-hand cars in the market and provide users with access to accurate information. The project utilizes web scraping techniques to collect data from online car listing platforms. This data includes crucial information such as brand, model, production year, mileage, price, and other relevant features. The collected data is stored in a database and subjected to thorough analysis.

The project offers users a user interface that assists them in determining the value of a car. Users can apply filters based on specific car features, obtain an estimated value for a desired car, and compare it with the listing prices. Furthermore, the platform also presents users with the best listings that match the specified car features, enabling them to evaluate the most suitable options.

This project provides significant benefits to both car buyers and sellers. Buyers can make more informed decisions when purchasing a car, while sellers have the opportunity to sell their vehicles at market value. Additionally, the development of this project offers valuable experience for individuals interested in data analysis, web scraping, and user interface development.

# ANALYSIS

## Backend Analysis

The Selenium framework plays a significant role in collecting data from classified ad websites. It gathers data dynamically and provides convenience by interacting with the necessary buttons. It forms the backbone of the project's backend by collecting the data.

Selenium navigates to the specified link where it retrieves the links of published ads and saves them in a list or a text file in the Python environment. Selenium is directed to this list or text file containing the ad links. It visits each link and retrieves the data from the specified XPATH accurately.

The retrieved data includes important information about the vehicle, such as brand, model, year, mileage, location, fuel type, and ad image. Then, these details are transferred to the database.

When it comes to managing a large amount of data, MongoDB, which is the most performance-efficient database for data storage, is used. MongoDB is preferred due to its user-friendly interface and compatibility with different data formats.

The collected data is processed and visualized using various Python libraries like Scikit-learn, Matplotlib, Seaborn, and Pandas in the Jupyter environment. The data is organized, visualized, interpreted, and analyzed.

A model is created using machine learning techniques. The model is trained, refined, and tested. Once the model is prepared for usage, it makes its initial predictions, and success results are obtained.

The operations performed in the backend are then implemented in a web environment using Flask.

Within the scope of this project, a user-friendly interface is created using the Flask API. This allows easy access to the collected data and facilitates the smooth delivery of results to the user.

## Frontend Analysis

A user-friendly form has been designed for the user. Here, the user can easily select the criteria of their vehicle or the vehicle they want to determine the market value of, and obtain the result with a single click. The user who sees the market value of the vehicle with the entered criteria can also view the listings of similar vehicles under the "Recommended Listings" section and access these listings with a single click if desired. The user can view the photos, specifications, and prices of the vehicles in the recommended listings without having to go to another link. The interface is open to further development. In the future, various features can be added, and by collaborating with the listing websites, notification of listings can also be sent to the user. This user-friendly interface, created with HTML and CSS, efficiently and dynamically presents the data to the user, guided by Flask, in a fast and flexible manner.

# DESIGN AND IMPLEMENTATION

## Backend Design

The backend infrastructure is the core of the “Predicting Second-Hand Car Prices with Machine Learning Techniques” application. The largest part of the project is carried out here. Most work is done in this part. Collection, processing, organization of data; Selection, training, validation and evaluation of the model are done here. This covers the largest part of the project.

Selenium, a Python Framework, is included in the project, modules are loaded. Selenium collects data via URLs specified in the internet environment. While collecting data, it is made suitable for the project. Then it is saved in the Mongo DB database. The properties of the vehicles in the advertisements are kept in the project's own database. Then the data here is transferred back to the Python environment and various data preprocessing steps are performed.

Machine Learning techniques are used after the data is made available for model building. Various models are tested, test results are analyzed. The model suitable for the data and the target is used and the model is evaluated.

All these backend works are transferred to the web environment with the Flask API and are made suitable for the use of the user.

## Backend Implementation

Python framework Selenium is included in the project, and the necessary modules are loaded. The driver that Selenium will work with is added to the project. Data is collected from the specified URLs. The XPath of the desired data in HTML format is determined, and Selenium retrieves the data corresponding to this XPath. The collected data is saved to a text file. Selenium interacts with buttons and navigates through different pages of the website due to its dynamic nature. MongoDB modules are loaded and added to the project. Then, the data is saved to a collection created in MongoDB. Now, all the data of the listings is stored in the project's own database.

The data from the database is brought back to the Python environment and a dataframe is created. Various data processing steps are performed on this dataframe. Missing values are filled using the KNN algorithm, and various data type conversions are made. "Encoding" is applied to categorical variables. After all the data is in numerical format, machine learning techniques are applied. Various models are imported into the project using the Scikit-Learn library. The models are evaluated and analyzed. The MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), and R2 scores of the models are important. Based on these scores, a preferred model is selected and a prediction model is created.

After all these backend processes are completed, they can be easily transferred to the web environment using Flask API.

To feed the user's inputs into the model, they need to be in numerical format. Therefore, with Flask, the user's string-type inputs are converted to numerical values and fed into the model. Flask can be used to provide input to the model and also to transmit the results to the user. Additionally, we can recommend the best listings from the database to the user. Thus, the Backend part is completed.

## Frontend Design

The aim of the project is to provide the user with results without wasting their time, energy, and money, so the same objective was prioritized for the frontend part as well. The front-end interface was designed with consideration for user experience. Through this user-friendly interface, the user can enter the criteria in a form-like manner and quickly view the results in the same format. It is also made possible for the user to access the recommended listings with just one click.

## Frontend Implementation

# TEST AND RESULTS

## Typesetting

## Paper Quality

## Printer

## Reproduction

## Binding

# CONCLUSION

## Use of Copyrighted Material

Writers of reports must assume full responsibility for use of any copyrighted material in their manuscripts. Written permission of the copyright owner must be obtained when extensive use is contemplated.

Permission to quote extensively from copyrighted material should be obtained by the candidate from the author or the publisher, whichever holds the copyright. Customarily, authorization is granted on the condition that proper acknowledgement is made. In some instances, however, copyright granting permission to include such material should be kept on file by the candidate for later reference in case questions arise.

## Publications

Reports, or extracts from them, may be published only upon release for publication by the major supervisor and provided proper credit is given to Yeditepe University. No report may be published as such before it has been approved by the major supervisor. All reports and separately submitted abstracts are the property of the University.

# Bibliography

**References:**

Online classifieds platforms such as Arabam.com and similar second-hand car listing websites.

Python programming language and related libraries: Selenium, Pandas, NumPy, Scikit-learn, etc.

Jupyter Notebook and other tools used for data analysis.

Flask web framework and other tools used for web form development.

MongoDB database management system.

**Documentation:**

MongoDB documentation for database management https://www.mongodb.com/docs/

**Educational Resources:**

Udemy course on data science with Python: https://www.udemy.com/course/python-egitimi

**Articles:**

"Predicting Used Car Prices with Machine Learning Techniques" by Enes Gökçe: https://towardsdatascience.com/predicting-used-car-prices-with-machine-learning-techniques-8a9d8313952

"Creating Machine Learning Models with Python" article: https://jovian.com/math-nights/used-car-quality-detection-real-world-ml-model-with-python/v/4

**Supportive Resources:**

YouTube channel of John Watson Rooney for Python language support: https://www.youtube.com/@JohnWatsonRooney

Documents and codes shared by Melih Bodur for similar applications: https://github.com/melihbodur

Resource on Selenium usage by the freecodecamp YouTube channel: https://www.youtube.com/@freecodecamp

Article on web scraping with Selenium by Ömer Şenol: https://senolomer0.medium.com/selenium-i̇le-i̇nternetten-veri-çekme-d0a597fa4b79

# APPENDIX A: SAMPLE PAGES

# import the required libraries and install the modules on my computer from the command prompt.

from selenium import webdriver

from time import sleep

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

import os

from selenium.common.exceptions import NoSuchElementException

#I determine the URL of the website from which I want to fetch data using Selenium. I retrieve the links of the listings on the visited site and create a text file. To prevent processing duplicate listings, I also save the fetched links in this file. As long as the "go to the next page" button is active on the site, I continue to use the button. This way, I obtain the links of every listing on every page. I exclude the links of advertisement pages generated by the site.

# Chrome Driver'ı yükleme ve ayarları

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

driver.set\_window\_size(1250, 740)

driver.set\_window\_position(0, 0)

# Ana link ve link listesi dosyası

mainLink = "https://www.arabam.com/ikinci-el"

linklist\_file = "linklist.txt"

# Link listesini dosyadan okuma veya boş liste olarak başlatma

if os.path.exists(linklist\_file):

    with open(linklist\_file, "r") as file:

        linklist = file.read().splitlines() #linklist.txt dosyasını okuyarak her bir satırı ayrı bir öğe olarak içeren bir liste oluşturur.

else:

    linklist = []

# Linkleri çekme ve link listesine ekleme

def get\_links(driver, xpath):

    a = driver.find\_element(By.XPATH, xpath)

    lnks = a.find\_elements(By.TAG\_NAME, "a")

    for lnk in lnks:

        href = lnk.get\_attribute("href")

        if href and "https://www.arabam.com/oto-ekspertiz" not in href and "https://www.arabam.com/turbolar" not in href:

            if href not in linklist:

                linklist.append(href)

    with open(linklist\_file, "w") as file:

        file.write("\n".join(linklist))

    return linklist

# İlk sayfayı ziyaret et

url = mainLink

driver.get(url)

sleep(4)

# Sonraki sayfalara tıklayarak linkleri çekme

while True:

    next\_buttons = driver.find\_elements(By.XPATH, '//\*[@id="pagingNext"]')

    if len(next\_buttons) > 0:

        next\_button = next\_buttons[0]

        if next\_button.is\_enabled():

            next\_button.click()

            sleep(10)

            linklist = get\_links(driver, '//\*[@id="js-hook-missing-space-content"]/div[2]/div/div[2]')

        else:

            break

    else:

        break

#I had previously saved my links to a text file. Now, I will pass these links to Selenium and extract the necessary data from each link. I will store this data in my cardetail list, ensuring that the links and data are aligned in the same order. When the listing links are removed, there is a shift in the database. To handle this, I use NoSuchElementException to check which links have been removed and keep track of the removed links in a separate list.

# Chrome Driver'ı yükleme ve ayarları

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

# Verileri toplama ve dosyaya yazma

#cardetail\_file = "cardetail.txt"

# Link listesini dosyadan okuma veya boş liste olarak başlatma

cardetail = []

removed\_links = []

removed\_img = []

# Linkleri linklist.txt dosyasından okuma

with open("linklist.txt", "r") as file:

    linklist = file.read().splitlines()

for link in linklist:

    driver.get(link)

    try:

        element = driver.find\_element(By.XPATH, '//\*[@id="js-hook-for-observer-detail"]/div[2]')

        item = element.text

        #Resimlerin linkini çekme bloğu

        try:

            image\_element = driver.find\_element(By.XPATH,'//\*[@id="js-hook-for-main-slider"]/div/div/a[1]/img')

            image\_url = image\_element.get\_attribute('src')

        except NoSuchElementException:

            print("Resim bulunamadı:", link)

            image\_url = "None"

            removed\_img.append(link)

        if item:

            cardetail.append((link, item, image\_url))

        else:

            print("İlan kaldırıldı:", link)

            removed\_links.append(link)

    except NoSuchElementException:

        print("İlan kaldırıldı:", link)

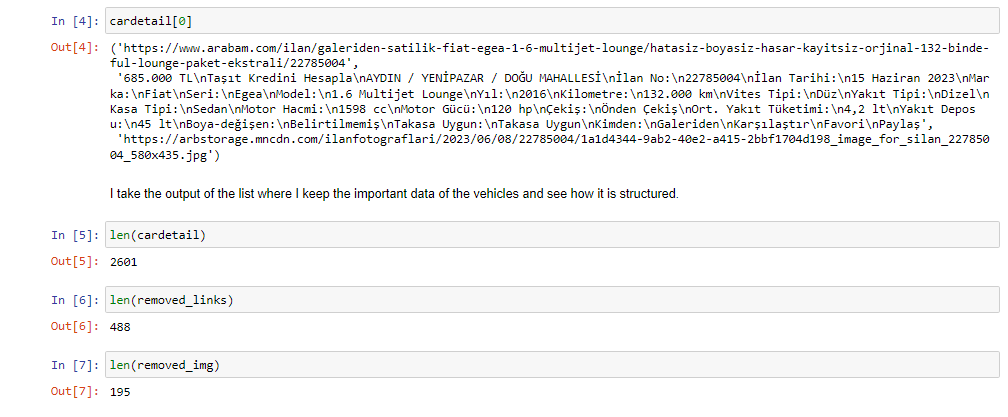
        removed\_links.append(link)

        continue

driver.quit()

Output of the code:





#The retrieved data comes with \n as a delimiter. To handle this, I first identify \n as the separation point for my data. Additionally, the "price" variable was not defined, so I create the "price" variable and assign the price value to it. I check for removed listings and exclude them. I modify Turkish characters and remove string elements like punctuation marks because I need to process my data as integers or floats. This applies to unit abbreviations as well. It is important to properly handle and format the data to avoid any issues in the future.

processed\_data\_file = "processed\_data.txt"

processed\_data = []

for idx, item in enumerate(cardetail):

    data = item[1].split('\n')

    vehicle = {}

    #BURASI ÇOK ÖNEMLİ BURADA LINKLIST ICINDEKI KALDIRILMIS LINKLERI TEMIZLIYORUM !

    linklist = [link for link in linklist if link not in removed\_links]

    # İlgili linki ekleyin

    vehicle['Link'] = linklist[idx]

    vehicle['Image\_URL'] = item[2] if item[2] else "None"

    # Fiyatı ve konumu ayırma

    fiyat = data[0].split('\n')[0]

    fiyat = fiyat.replace('.', '').replace(' TL', '')  # '.' ve 'TL' kaldırılıyor

    vehicle['Fiyat'] = int(fiyat)  # Fiyat integer olarak kaydediliyor

    konum = None

    # Konumu bulma

    for i in range(1, len(data)-1):

        if data[i] != 'Taşıt Kredini Hesapla':

            konum = data[i]

            break

    vehicle['Konum'] = konum

    # Diğer özellikleri ayırma

    for i in range(1, len(data)-1, 2):

        if data[i] == 'Taşıt Kredini Hesapla':

            continue

        key = data[i].strip(':')

        value = data[i+1]

        # İlgili karakter dönüşümleri yapılıyor

        key = key.replace('ı', 'i').replace('İ', 'i').replace('ö', 'o').replace('ü', 'u').replace('ş', 's').replace('ç', 'c')

        key = key.replace('Ç', 'C').replace('Ş', 'S').replace('Ü', 'U').replace('Ö', 'O')

        key = key.replace(' ', '\_').replace('ğ','g').replace('Ğ','G').replace('-','\_')

        if '.' in value:  # '.' kaldırılıyor

            value = value.replace('.', '')

        if '-' in value:  # '.' kaldırılıyor

            value = value.replace('-', '')

        if ',' in value:  # '.' kaldırılıyor

            value = value.replace(',', '.')

        #if 'hp' in value:  # 'hp' kaldırılıyor

        #   value = value.replace(' hp', '')

        #if 'cc' in value:  # 'cc' kaldırılıyor

        #    value = value.replace('cc', '')

        #if 'km' in value:  # 'km' kaldırılıyor

        #    value = value.replace('km', '')

        #if 'cm3' in value:

        #    value = value.replace('cm3','')

        if key == 'Ort.\_Yakit\_Tuketimi' or key == 'Yakit\_Deposu':

            if value is not None:

                value = value.replace('lt', '')

            else:

                value = None

        if key == 'Motor\_Hacmi' or key == 'Motor\_Gucu':

            if value is not None:

                value = value.replace('lt', '').replace('hp','').replace('cc','').replace('cm3','')

            else:

                value = None

        if key == 'Kilometre':

            if value is not None:

                value = value.replace('km', '').replace('kilometre','').replace('KM','')

            else:

                value = None

        if key in ['Yil', 'Kilometre']:  # Yıl ve Kilometre integer olarak kaydediliyor

            value = int(value)

        if key in ['Motor\_Hacmi', 'Ort.\_Yakit\_Tuketimi', 'Motor\_Gucu','Yakit\_Deposu']:

            if value.strip():  # Değerin boş olmadığından emin olunuyor

                try:

                    value = float(value)

                except ValueError:

                    value = None  # Sayısal bir değere dönüştürülemiyorsa 'None' olarak ayarlanıyor

            else:

                value = None

        vehicle[key] = value

    if 'Boya\_degisen' in vehicle:

        if vehicle['Boya\_degisen'] != 'Tamamı orjinal' and vehicle['Boya\_degisen'] != 'Belirtilmemiş':

            vehicle['Boya\_degisen'] = 'Var'

    processed\_data.append(vehicle)

# İşlenmiş verileri dosyaya yazma

with open(processed\_data\_file, "w", encoding="utf-8") as file:

    for vehicle in processed\_data:

        for key, value in vehicle.items():

            file.write(f"{key}: {value}\n")

        file.write("\n")

print("Veriler processed\_data.txt dosyasına kaydedildi.")

# İşlenmiş verileri görüntüle

for vehicle in processed\_data:

    print(vehicle)

Output of the code:



#TRANSFERRING THE PROCESSED DATA TO DATABASE

from pymongo import MongoClient

from dotenv import load\_dotenv, find\_dotenv

import os

import pprint

load\_dotenv(find\_dotenv())

#MongoDB için gerekli kütüphaneler

password = os.environ.get("MONGODB\_PWD")

connection\_string = f"mongodb+srv://emirhanbal:{password}@graduation.r68pz0b.mongodb.net/?retryWrites=true&w=majority"

client = MongoClient(connection\_string)

#MONGODB ile kodumu ilişkilendirme adımı. burada database'imi bağlıyorum.

db = client["ilanlar"]

collection = db["arac\_detay\_arabamcom"]

#Verilerin database'e kaydedilmesi.

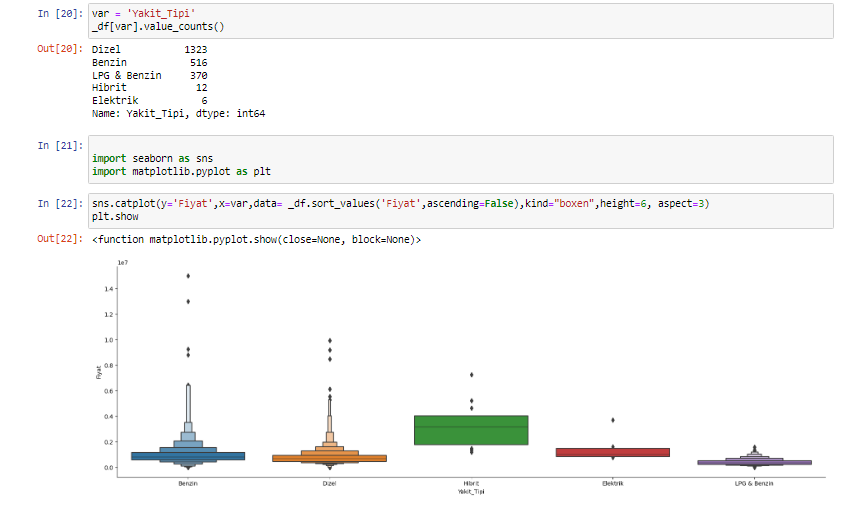
for vehicle in processed\_data:

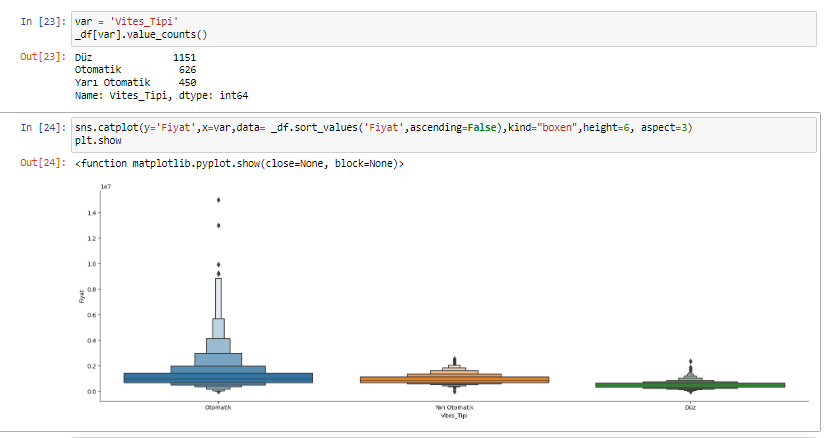
    collection.insert\_one(vehicle)

print("Veriler MongoDB'ye kaydedildi.")

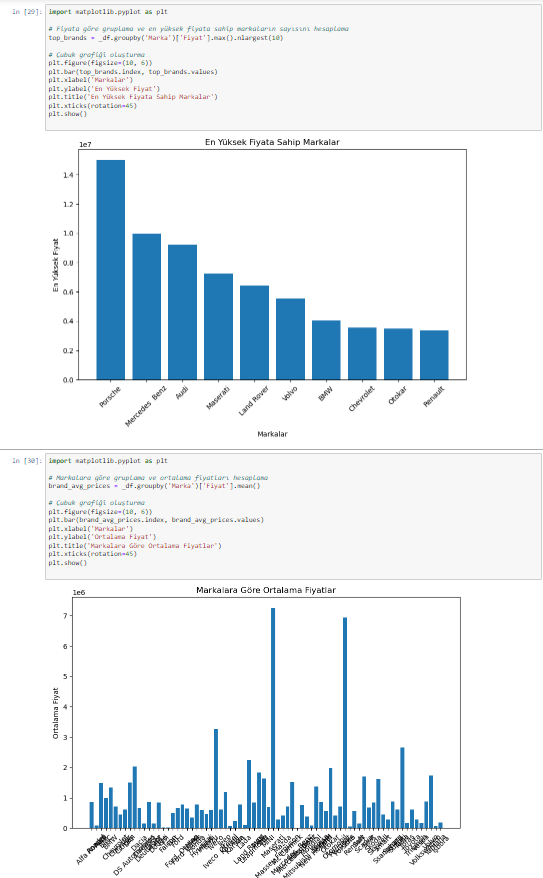
# Bağlantıyı kapatma

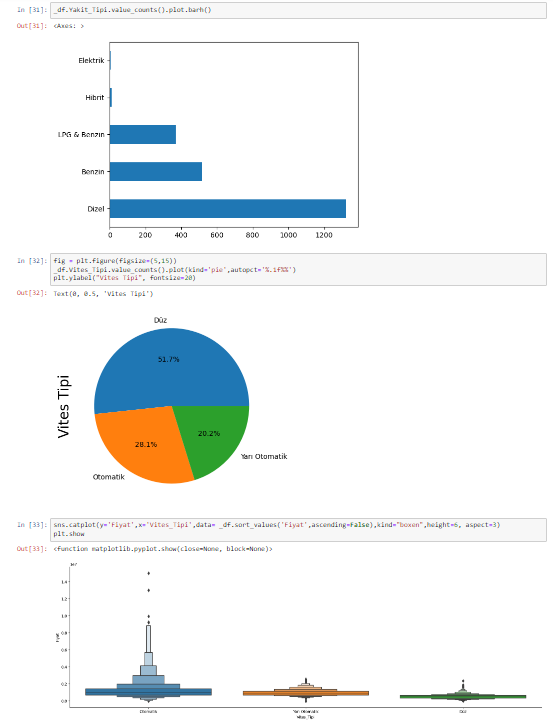
Data Exploration and Data Visualization

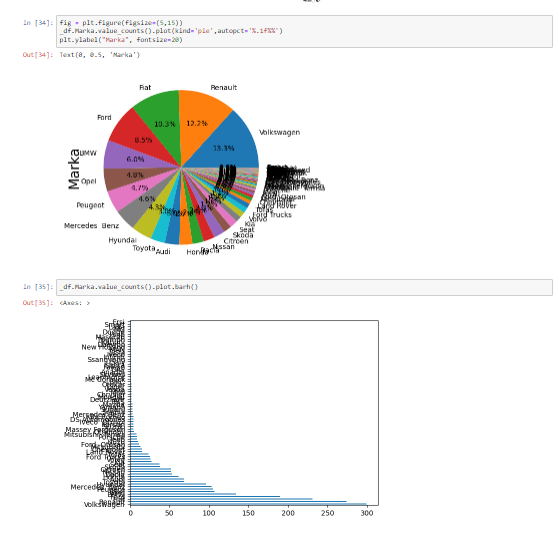












# some editing so that there are no outliers to skew my data.

# verilerimin içerisinde 10binden küçük veya 15milyondan büyük fiyatlı olanları kontrol ediyorum.

\_df[(\_df.Fiyat<10000) | (\_df.Fiyat>15000000) | (\_df.Kilometre>1000000)]

# ve bunları verilerim arasından çıkarıyorum

\_df.drop(\_df[(\_df.Fiyat < 10000) | (\_df.Fiyat > 15000000) | (\_df.Kilometre>1000000)].index, inplace=True)

#NOW I'M MAKING VARIOUS COMBINATIONS TO SEE WHAT DATAFRAME WORKS SUITABLE WITH WHAT COLUMNS AND WITH WHAT MODELS.

#COMBINATION #1 First, I hold all the columns and insert them into the model.

columns\_to\_keep = ['Fiyat', 'Marka', 'Seri', 'Yil', 'Kilometre', 'Vites\_Tipi', 'Yakit\_Tipi', 'Boya\_degisen']

\_df = \_df[columns\_to\_keep]

#COMBINATION #2 then leave out the very diverse columns, namely the Brand and Series columns.

columns\_to\_keep2 = ['Fiyat', 'Yil', 'Kilometre', 'Vites\_Tipi', 'Yakit\_Tipi', 'Boya\_degisen']

\_df = \_df[columns\_to\_keep2]

#COMBINATION #3 Taking the Brand into account, but excluding the more diverse Series

columns\_to\_keep3 = ['Fiyat', 'Yil', 'Marka', 'Kilometre', 'Vites\_Tipi', 'Yakit\_Tipi', 'Boya\_degisen']

\_df = \_df[columns\_to\_keep3]

#!!! To avoid repeating the same steps for each combination, I take advantage of the benefits of the Jupyter Notebook environment. I specify the operations at the beginning and then iterate through the different combinations, executing the code for each one.

\_df.dtypes

nan\_sayisi = \_df.isna().sum()

print(nan\_sayisi)

from sklearn.impute import KNNImputer

for column in \_df.columns:

    if \_df[column].dtype == 'object':

        most\_frequent\_value = \_df[column].mode().values[0]

        \_df[column].fillna(most\_frequent\_value, inplace=True)

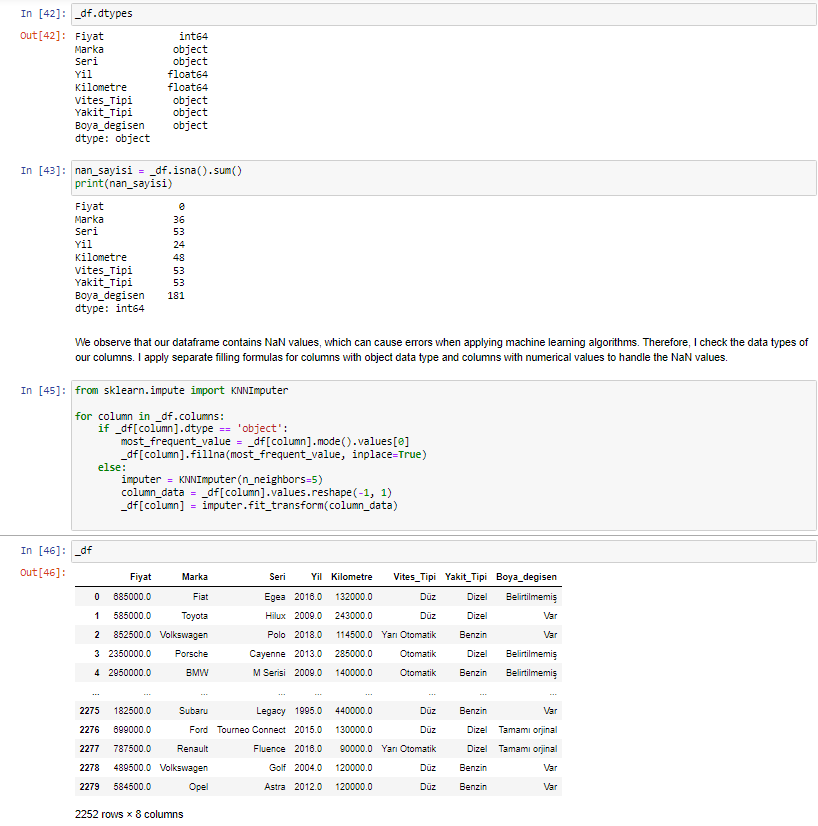
    else:

        imputer = KNNImputer(n\_neighbors=5)

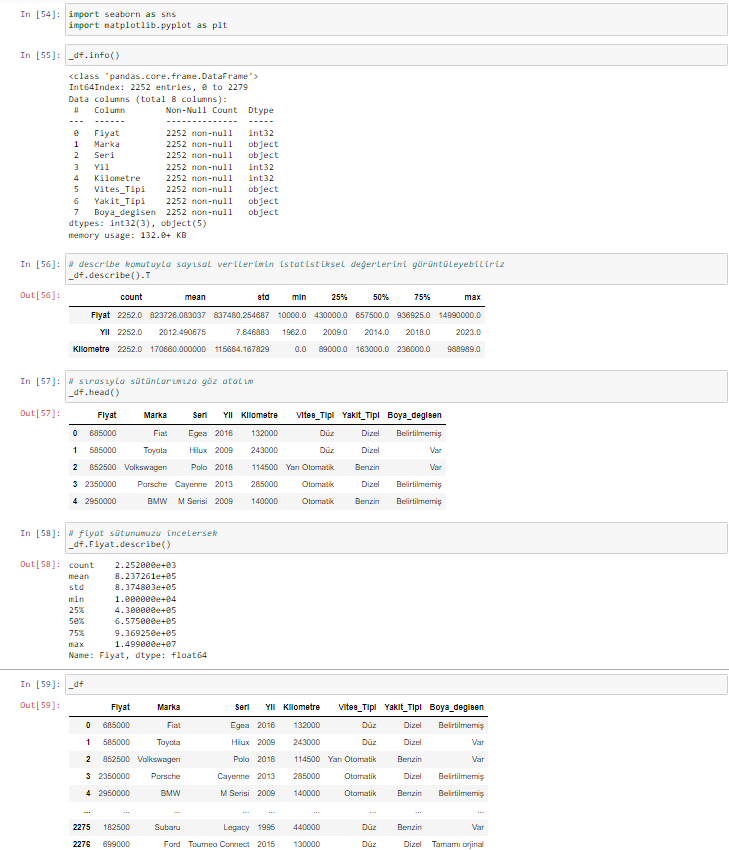
        column\_data = \_df[column].values.reshape(-1, 1)

        \_df[column] = imputer.fit\_transform(column\_data)

Output of the Code:



Statistical analysis with numerical values:



#Applying the One-Hot Encoding technique to convert categorical data into numerical values:

import pandas as pd

# One-Hot Encoding işlemi

yakit\_Tipi\_encoded = pd.get\_dummies(\_df['Yakit\_Tipi'], prefix='Yakit\_Tipi')

vites\_Tipi\_encoded = pd.get\_dummies(\_df['Vites\_Tipi'], prefix='Vites\_Tipi')

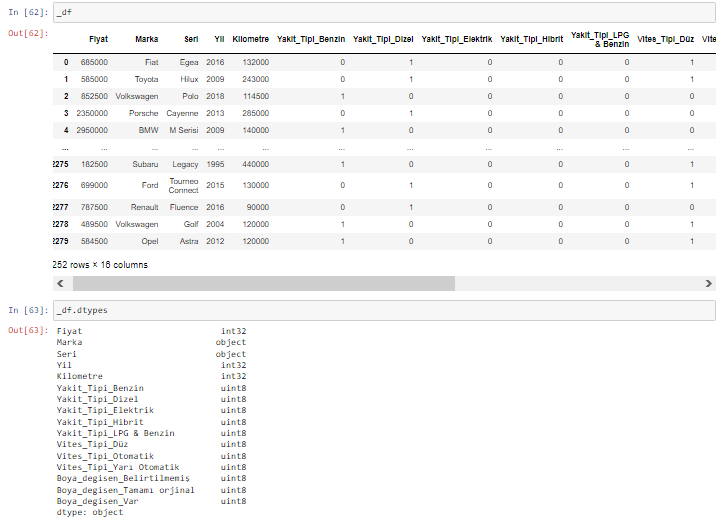
boya\_degisen\_encoded = pd.get\_dummies(\_df['Boya\_degisen'], prefix='Boya\_degisen')

# Yeni sütunları ekleme

\_df = pd.concat([\_df, yakit\_Tipi\_encoded, vites\_Tipi\_encoded, boya\_degisen\_encoded], axis=1)

\_df = \_df.drop(['Yakit\_Tipi', 'Vites\_Tipi', 'Boya\_degisen'], axis=1)

Output of the Code:



#Label encoding for categorical variables like brand and model:

from sklearn.preprocessing import LabelEncoder

# Marka sütununu seçme

Marka = \_df['Marka']

# Seri sütununu seçme

Seri = \_df['Seri']

# LabelEncoder nesnesini oluşturma ve dönüşümü yapma

label\_encoder = LabelEncoder()

# Marka sütununu dönüştürme

marka\_encoded = label\_encoder.fit\_transform(Marka)

# Dönüştürülen değerleri \_df'e ekleme

\_df['Marka\_Encoded'] = marka\_encoded

# Her bir sayısal değerin karşılık gelen markasını elde etme

marka\_degerleri = label\_encoder.classes\_

# Her bir sayısal değerin ve karşılık gelen markanın ekrana yazdırılması

marka\_sozlugu = {sayisal\_deger: marka for sayisal\_deger, marka in enumerate(marka\_degerleri)}

# Seri sütununu dönüştürme

seri\_encoded = label\_encoder.fit\_transform(Seri)

# Dönüştürülen değerleri \_df'e ekleme

\_df['Seri\_Encoded'] = seri\_encoded

# Her bir sayısal değerin karşılık gelen serisini elde etme

seri\_degerleri = label\_encoder.classes\_

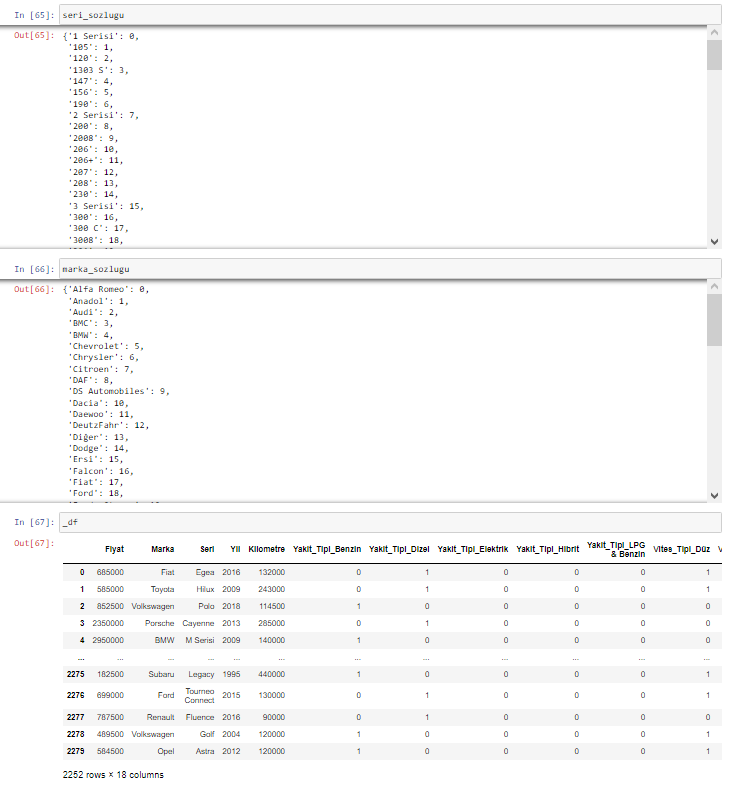
# Her bir sayısal değerin ve karşılık gelen serinin ekrana yazdırılması

seri\_sozlugu = {sayisal\_deger: seri for sayisal\_deger, seri in enumerate(seri\_degerleri)}

\_df.drop(['Seri'],axis=1,inplace=True)

\_df.drop(['Marka'],axis=1,inplace=True)

Output of the Code:



Creating Training and Test Models

#All our data is now in numerical format, and we are ready to feed it into the model.

#Now that all the column values are numerical, they are suitable for machine learning modeling.

#Splitting the Data

from sklearn.model\_selection import train\_test\_split

# Bağımsız değişkenler (X) ve hedef değişken (y) olarak ayırma

X = \_df.drop('Fiyat', axis=1)  # Hedef sütunu çıkararak bağımsız değişkenleri alıyoruz

y = \_df['Fiyat']  # Hedef sütunu olarak ayarlanmış olan sütunu alıyoruz

# Veri kümesini eğitim ve test kümelerine ayırma

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Eğitim ve test kümelerinin boyutlarını kontrol etmek

print("Eğitim kümesi boyutu:", X\_train.shape)

print("Test kümesi boyutu:", X\_test.shape)

Model Selection and Training

#Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

regressor=RandomForestRegressor(n\_estimators=200,min\_samples\_split=2,min\_samples\_leaf=2,max\_features='sqrt', max\_depth=80, bootstrap=True)

regressor.fit(X\_train,y\_train)

y\_pred\_randf=regressor.predict(X\_test)

from sklearn.tree import DecisionTreeRegressor

regr=DecisionTreeRegressor(max\_depth=4)

regr.fit(X\_train,y\_train)

y\_pred\_dectree=regr.predict(X\_test)

from sklearn.ensemble import AdaBoostRegressor

adaregr=AdaBoostRegressor(DecisionTreeRegressor(max\_depth=4),n\_estimators=291)

adaregr.fit(X\_train,y\_train)

y\_pred\_adaboost=adaregr.predict(X\_test)

from sklearn.linear\_model import Ridge

rr=Ridge(alpha=50)

rr.fit(X\_train,y\_train)

y\_pred\_ridge=rr.predict(X\_test)

from sklearn.linear\_model import LinearRegression

# Lineer regresyon modelini oluşturma

lr = LinearRegression()

# Modeli eğitme

lr.fit(X\_train, y\_train)

# Test verileri üzerinde tahmin yapma

y\_pred\_linear = lr.predict(X\_test)

from sklearn.linear\_model import SGDRegressor

clf=SGDRegressor(max\_iter=1000,tol=1e-3)

clf.fit(X\_train,y\_train)

y\_pred\_sgd=clf.predict(X\_test)

from sklearn import svm

clf = svm.SVR(kernel='rbf')

clf.fit(X\_train,y\_train)

y\_pred\_svr=clf.predict(X\_test)

Model Evaluation

def accuracy(y\_pred, y\_test):

    from sklearn import metrics

    import matplotlib.pyplot as plt

    import numpy as np

    import pandas as pd

    print("MAE:", metrics.mean\_absolute\_error(y\_test, y\_pred))

    print("RMSE:", np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

    r2 = metrics.r2\_score(y\_test, y\_pred)

    print("R2 Score:", r2)

    # Scatter plot

    plt.scatter(y\_test, y\_pred)

    plt.xlabel('True Values')

    plt.ylabel('Predicted Values')

    plt.show()

**Label Encoding Brand and Model and Obtained Results for Combination 1**

print("Random Forest")

accuracy(y\_pred\_randf,y\_test)

print("Decision Tree")

accuracy(y\_pred\_dectree,y\_test)

print("ADABoost Regressor")

accuracy(y\_pred\_adaboost,y\_test)

print("Ridge Regression")

accuracy(y\_pred\_ridge,y\_test)

print("Linear Regression")

accuracy(y\_pred\_linear,y\_test)

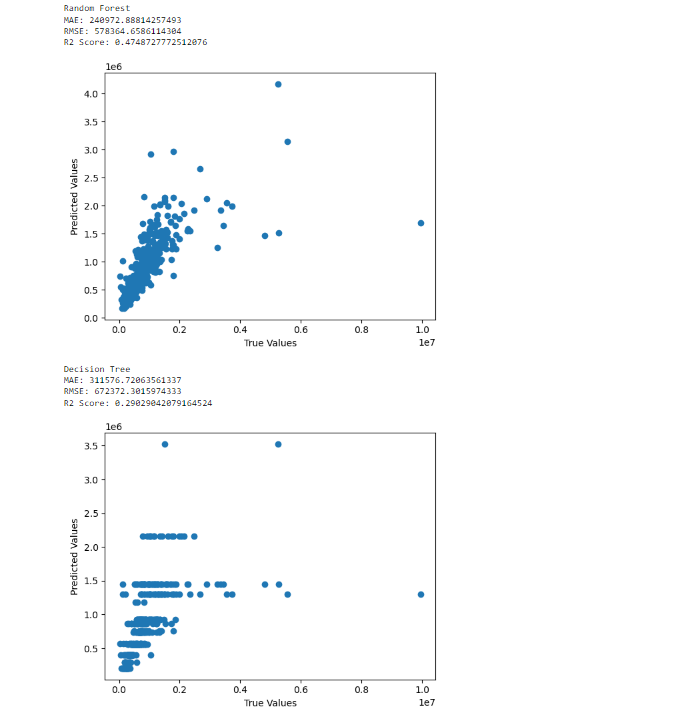
print("SGD Regression")

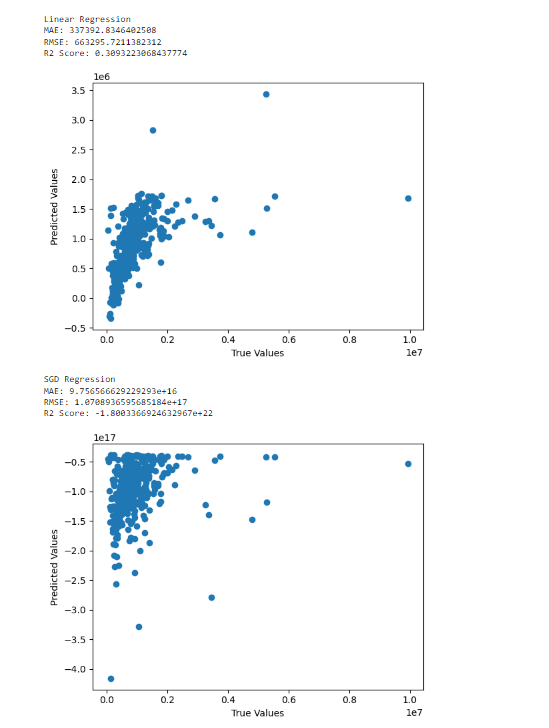
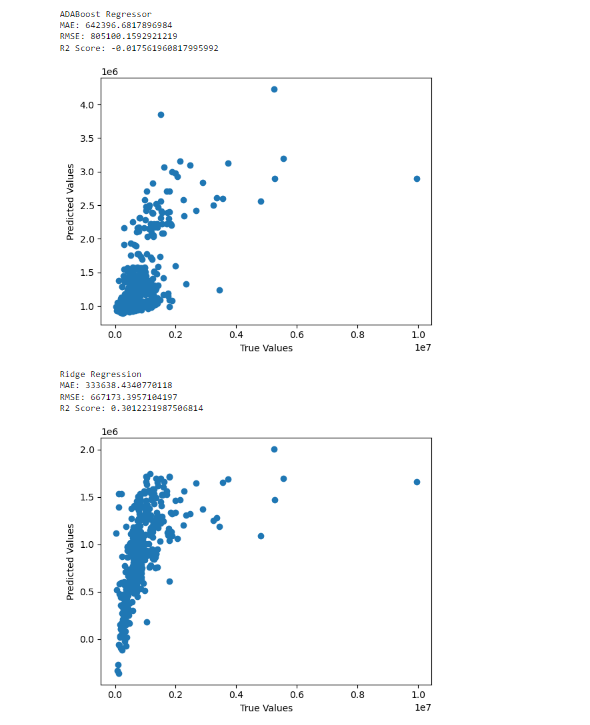
accuracy(y\_pred\_sgd,y\_test)

print("SVR Regression")

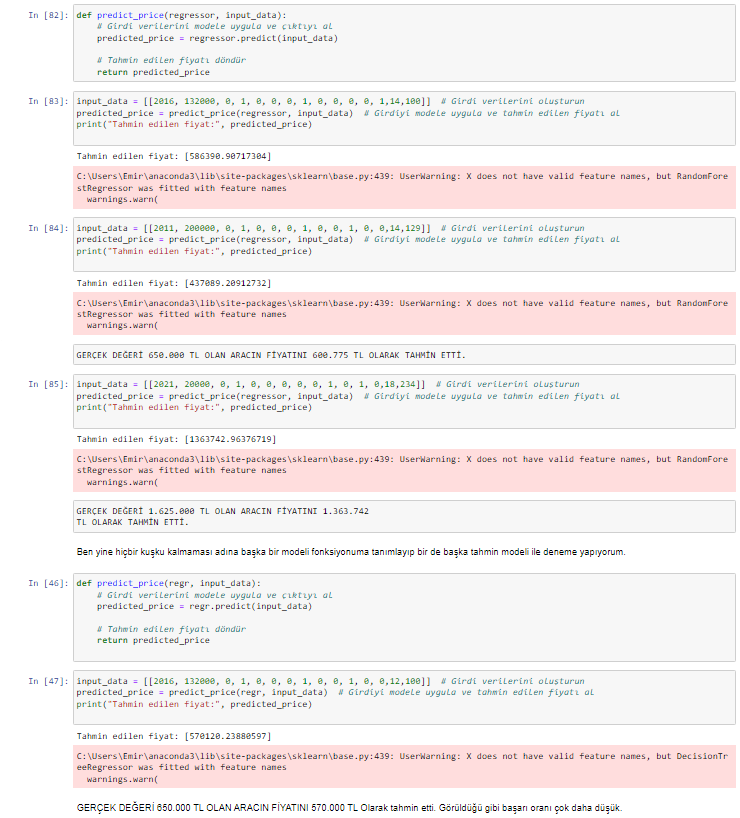
accuracy(y\_pred\_svr,y\_test)

Output of the Code:





Providing Input to the Prediction Model



**Obtaining Results without Including Brand and Model in the DataFrame - Combination 2**

(Before running this part, I reset the kernel and execute the Combination 2 columns defined at the top. I then proceed to execute the remaining steps in the same manner.)

print("Random Forest")

accuracy(y\_pred\_randf,y\_test)

print("Decision Tree")

accuracy(y\_pred\_dectree,y\_test)

print("ADABoost Regressor")

accuracy(y\_pred\_adaboost,y\_test)

print("Ridge Regression")

accuracy(y\_pred\_ridge,y\_test)

print("Linear Regression")

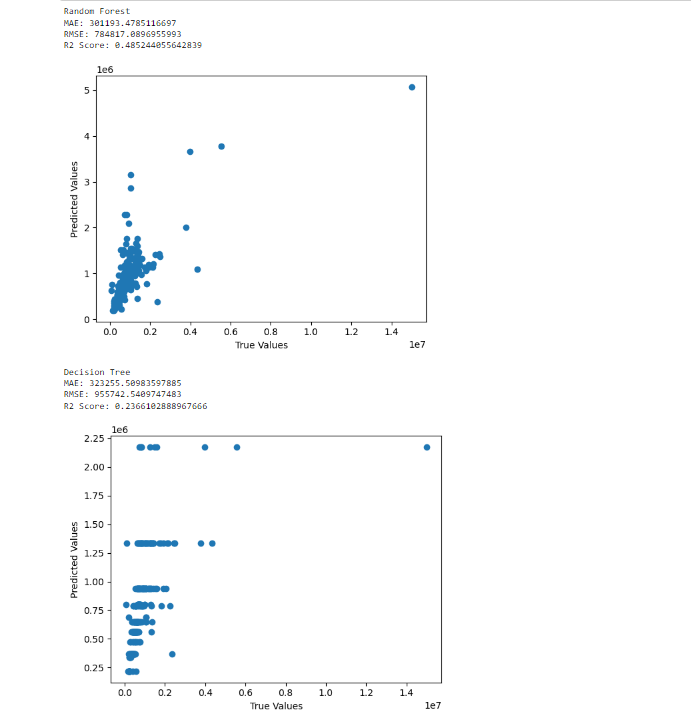
accuracy(y\_pred\_linear,y\_test)

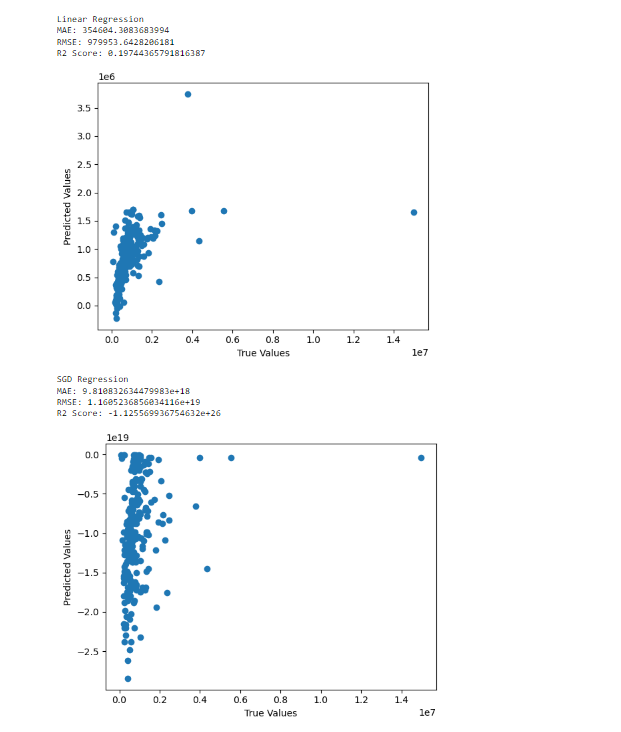
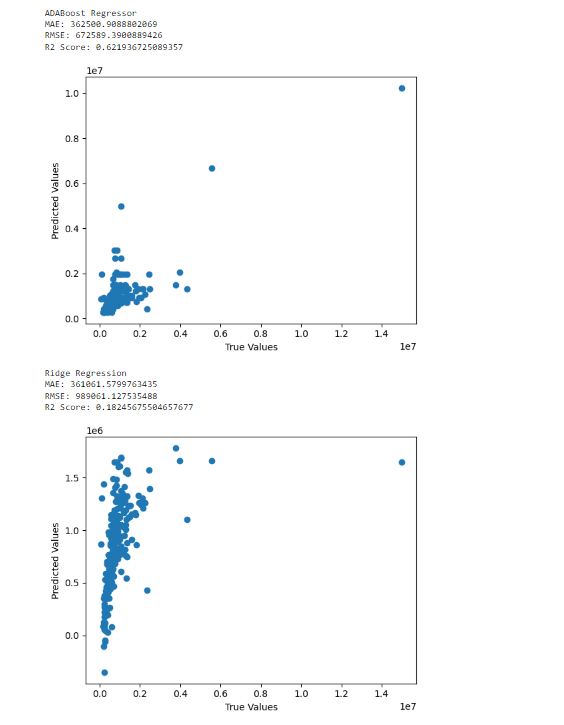
print("SGD Regression")

accuracy(y\_pred\_sgd,y\_test)

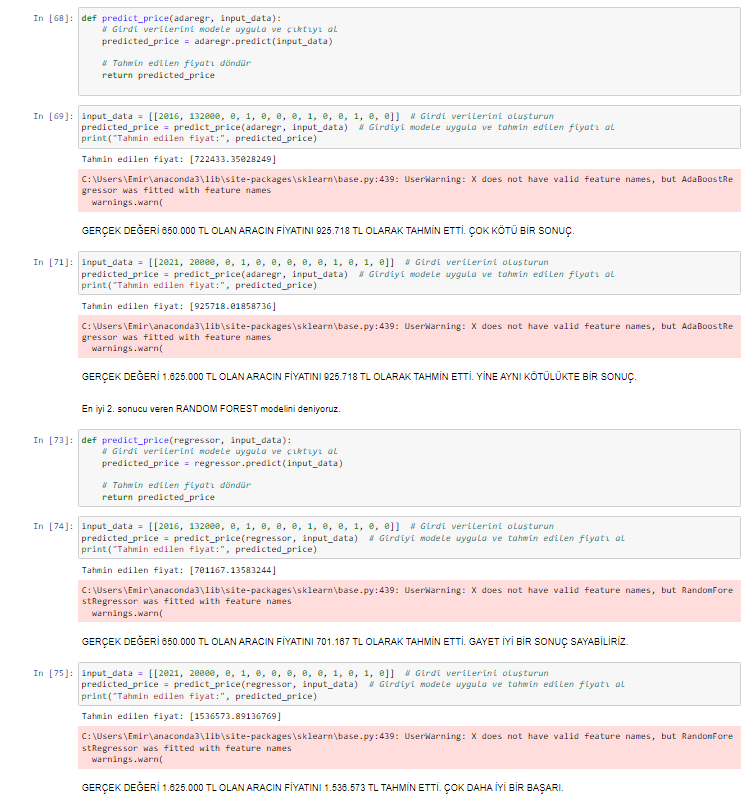
print("SVR Regression")

accuracy(y\_pred\_svr,y\_test)





Providing Input to the Prediction Model



**The results obtained by including only the brand encoding are labeled as Combination 3:**

print("Random Forest")

accuracy(y\_pred\_randf,y\_test)

print("Decision Tree")

accuracy(y\_pred\_dectree,y\_test)

print("ADABoost Regressor")

accuracy(y\_pred\_adaboost,y\_test)

print("Ridge Regression")

accuracy(y\_pred\_ridge,y\_test)

print("Linear Regression")

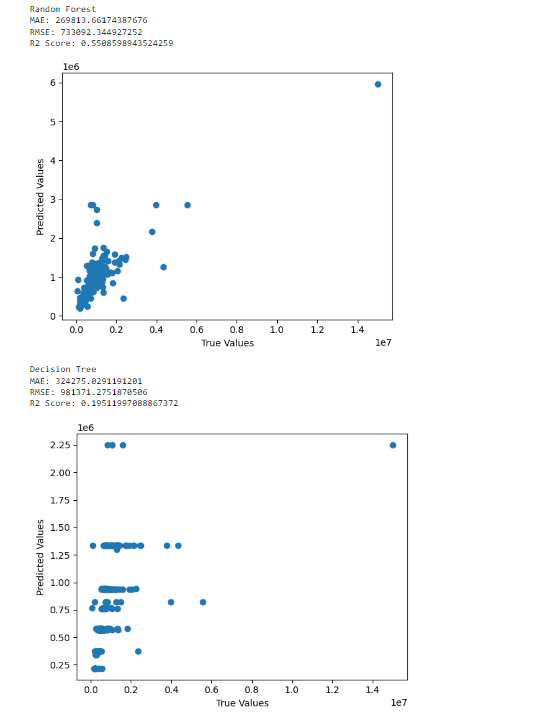
accuracy(y\_pred\_linear,y\_test)

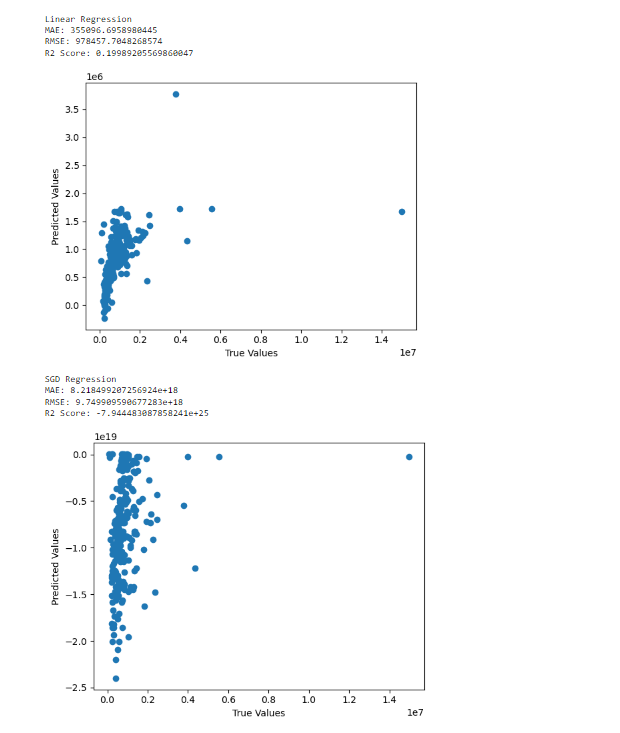
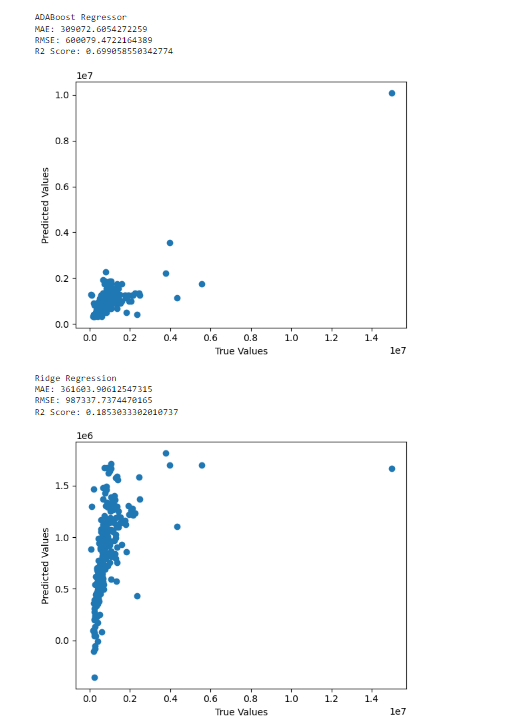
print("SGD Regression")

accuracy(y\_pred\_sgd,y\_test)

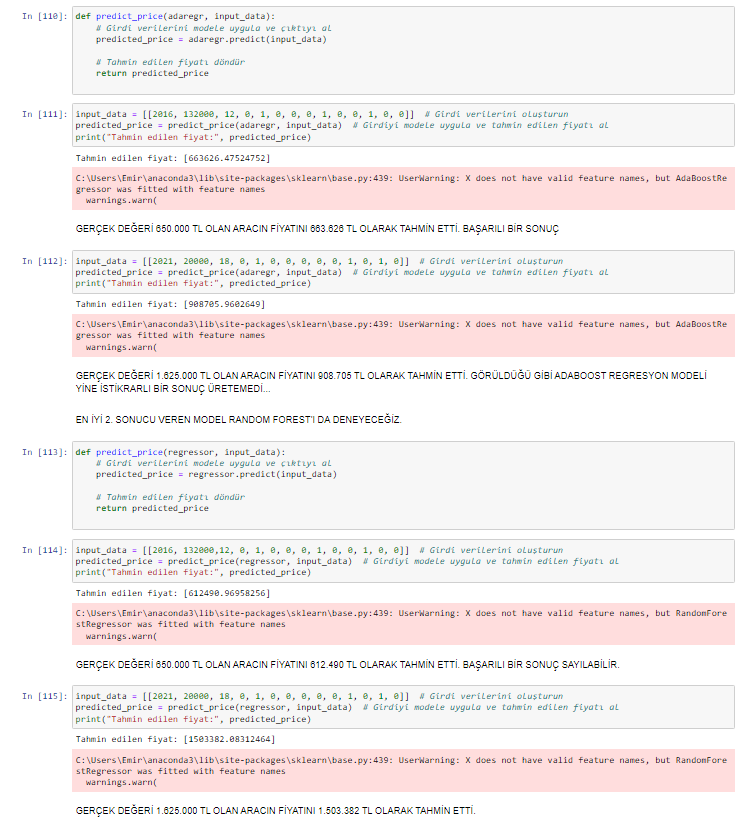
print("SVR Regression")

accuracy(y\_pred\_svr,y\_test)





Providing Input to the Prediction Model



***After going through all these steps to create the prediction model, let's now take a look at the simplified version of the project that has been implemented using Flask.***

from flask import Flask, request, jsonify , render\_template

app = Flask(\_\_name\_\_)

from pymongo import MongoClient

from dotenv import load\_dotenv, find\_dotenv

import os

import pprint

load\_dotenv(find\_dotenv())

#MongoDB için gerekli kütüphaneler

password = os.environ.get("MONGODB\_PWD")

connection\_string = f"mongodb+srv://emirhanbal:{password}@graduation.r68pz0b.mongodb.net/?retryWrites=true&w=majority"

client = MongoClient(connection\_string)

#MONGODB ile kodumu ilişkilendirme adımı. burada database'imi bağlıyorum.

db = client["ilanlar"]

collection = db["arac\_detay\_arabamcom"]

dataFromDatabase = list(collection.find())

# Convert entire collection to Pandas dataframe

import pandas as pd

\_df = pd.DataFrame(dataFromDatabase)

\_df.drop(\_df[(\_df.Fiyat < 10000) | (\_df.Fiyat > 15000000) | (\_df.Kilometre>1000000)].index, inplace=True)

columns\_to\_keep = ['Fiyat', 'Marka', 'Seri', 'Yil', 'Kilometre', 'Vites\_Tipi', 'Yakit\_Tipi', 'Boya\_degisen']

\_df = \_df[columns\_to\_keep]

from sklearn.impute import KNNImputer

for column in \_df.columns:

    if \_df[column].dtype == 'object':

        most\_frequent\_value = \_df[column].mode().values[0]

        \_df[column].fillna(most\_frequent\_value, inplace=True)

    else:

        imputer = KNNImputer(n\_neighbors=5)

        column\_data = \_df[column].values.reshape(-1, 1)

        \_df[column] = imputer.fit\_transform(column\_data)

\_df['Yil'] = \_df['Yil'].astype(int)

\_df['Fiyat'] = \_df['Fiyat'].astype(int)

\_df['Kilometre'] = \_df['Kilometre'].astype(int)

# One-Hot Encoding işlemi

yakit\_Tipi\_encoded = pd.get\_dummies(\_df['Yakit\_Tipi'], prefix='Yakit\_Tipi')

vites\_Tipi\_encoded = pd.get\_dummies(\_df['Vites\_Tipi'], prefix='Vites\_Tipi')

boya\_degisen\_encoded = pd.get\_dummies(\_df['Boya\_degisen'], prefix='Boya\_degisen')

# Yeni sütunları ekleme

\_df = pd.concat([\_df, yakit\_Tipi\_encoded, vites\_Tipi\_encoded, boya\_degisen\_encoded], axis=1)

\_df = \_df.drop(['Yakit\_Tipi', 'Vites\_Tipi', 'Boya\_degisen'], axis=1)

#DENEME

from sklearn.preprocessing import LabelEncoder

# Marka sütununu seçme

Marka = \_df['Marka']

# Seri sütununu seçme

Seri = \_df['Seri']

# LabelEncoder nesnesini oluşturma ve dönüşümü yapma

label\_encoder = LabelEncoder()

# Marka sütununu dönüştürme

marka\_encoded = label\_encoder.fit\_transform(Marka)

# Dönüştürülen değerleri \_df'e ekleme

\_df['Marka\_Encoded'] = marka\_encoded

# Her bir sayısal değerin karşılık gelen markasını elde etme

marka\_degerleri = label\_encoder.classes\_

# Her bir sayısal değerin ve karşılık gelen markanın ekrana yazdırılması

marka\_sozlugu = {marka: sayisal\_deger for sayisal\_deger, marka in enumerate(marka\_degerleri)}

# Seri sütununu dönüştürme

seri\_encoded = label\_encoder.fit\_transform(Seri)

# Dönüştürülen değerleri \_df'e ekleme

\_df['Seri\_Encoded'] = seri\_encoded

# Her bir sayısal değerin karşılık gelen serisini elde etme

seri\_degerleri = label\_encoder.classes\_

# Her bir sayısal değerin ve karşılık gelen serinin ekrana yazdırılması

seri\_sozlugu = {seri: sayisal\_deger for sayisal\_deger, seri in enumerate(seri\_degerleri)}

\_df.drop(['Seri'],axis=1,inplace=True)

\_df.drop(['Marka'],axis=1,inplace=True)

from sklearn.model\_selection import train\_test\_split

# Bağımsız değişkenler (X) ve hedef değişken (y) olarak ayırma

X = \_df.drop('Fiyat', axis=1)  # Hedef sütunu çıkararak bağımsız değişkenleri alıyoruz

y = \_df['Fiyat']  # Hedef sütunu olarak ayarlanmış olan sütunu alıyoruz

# Veri kümesini eğitim ve test kümelerine ayırma

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

#Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

regressor=RandomForestRegressor(n\_estimators=200,min\_samples\_split=2,min\_samples\_leaf=2,max\_features='sqrt', max\_depth=80, bootstrap=True)

regressor.fit(X\_train,y\_train)

y\_pred\_randf=regressor.predict(X\_test)

def predict\_price(regressor, input\_data):

    # Girdi verilerini modele uygula ve çıktıyı al

    predicted\_price = regressor.predict(input\_data)

    predicted\_price = int(predicted\_price)

    # Tahmin edilen fiyatı döndür

    return predicted\_price

yakit\_tipi\_sozlugu = {

    'Benzin': [1, 0, 0, 0, 0],

    'Dizel': [0, 1, 0, 0, 0],

    'Elektrik': [0, 0, 1, 0, 0],

    'Hibrit': [0, 0, 0, 1, 0],

    'LPG & Benzin': [0, 0, 0, 0, 1]

}

vites\_tipi\_sozlugu = {

    'Düz': [1, 0, 0],

    'Otomatik': [0, 1, 0],

    'Yarı Otomatik': [0, 0, 1]

}

boya\_degisen\_sozlugu = {

    'Belirtilmemiş': [1, 0, 0],

    'Tamamı orjinal': [0, 1, 0],

    'Var': [0, 0, 1]

}

# HTML formunu görüntülemek için GET isteği

@app.route('/', methods=['POST', 'GET'])

def predict():

    if request.method == 'POST':

        # Process the form data and predict

        yil = int(request.form['yil'])

        kilometre = int(request.form['kilometre'])

        yakit\_tipi = request.form['yakit\_tipi']

        vites\_tipi = request.form['vites\_tipi']

        boya\_degisen = request.form['boya\_degisen']

        marka = request.form['marka']

        seri = request.form['seri']

        yakit\_tipi\_kod = yakit\_tipi\_sozlugu.get(yakit\_tipi, [0] \* len(yakit\_tipi\_sozlugu))

        vites\_tipi\_kod = vites\_tipi\_sozlugu.get(vites\_tipi, [0] \* len(vites\_tipi\_sozlugu))

        boya\_degisen\_kod = boya\_degisen\_sozlugu.get(boya\_degisen, [0] \* len(boya\_degisen\_sozlugu))

        # Marka and Series encoding remains the same

        marka\_kod = marka\_sozlugu.get(marka, -1)

        seri\_kod = seri\_sozlugu.get(seri, -1)

        input\_sorgu = [marka, seri, yil, kilometre, yakit\_tipi, vites\_tipi, boya\_degisen]

        input\_data = [[yil, kilometre] + yakit\_tipi\_kod + vites\_tipi\_kod + boya\_degisen\_kod + [marka\_kod, seri\_kod]]

        predictions = []

        user\_input = collection.find({"Marka" : marka, "Seri": seri, "Boya\_degisen":boya\_degisen }).sort("Fiyat", 1).limit(3) #, "Boya\_degisen" : "Belirtilmemiş"

        for arac in user\_input:

            link = arac['Link']

            fiyat = arac['Fiyat']

            marka = arac['Marka']

            seri = arac['Seri']

            yil = arac['Yil']

            kilometre = arac['Kilometre']

            vites\_tipi = arac['Vites\_Tipi']

            yakit\_tipi = arac['Yakit\_Tipi']

            boya\_degisen = arac['Boya\_degisen']

            image\_URL = arac['Image\_URL']

            predictions.append((link, fiyat, marka, seri, yil, kilometre, vites\_tipi, yakit\_tipi,boya\_degisen,image\_URL))

        predicted\_price = predict\_price(regressor, input\_data)

        return render\_template('result.html', predicted\_price=predicted\_price , input\_data=input\_data, predictions=predictions, input\_sorgu=input\_sorgu)

    else:

        return render\_template('ilanlar.html')

#return render\_template('index.html')

    # You should extract the required features from the data received

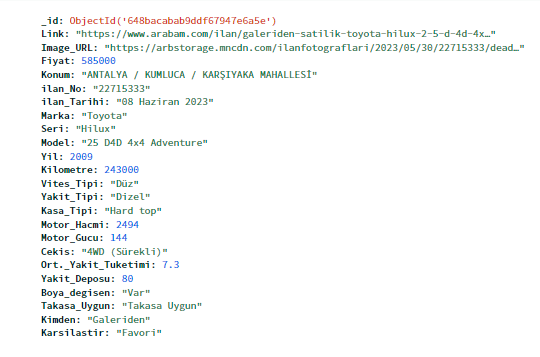
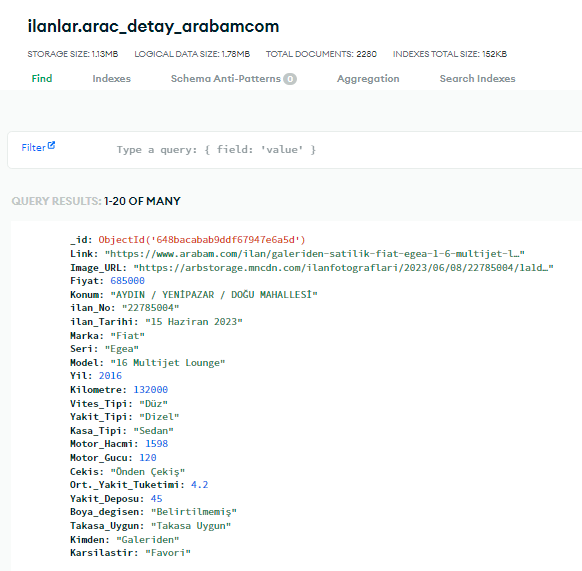
    # You should sanitize your data here

    # Process your data here

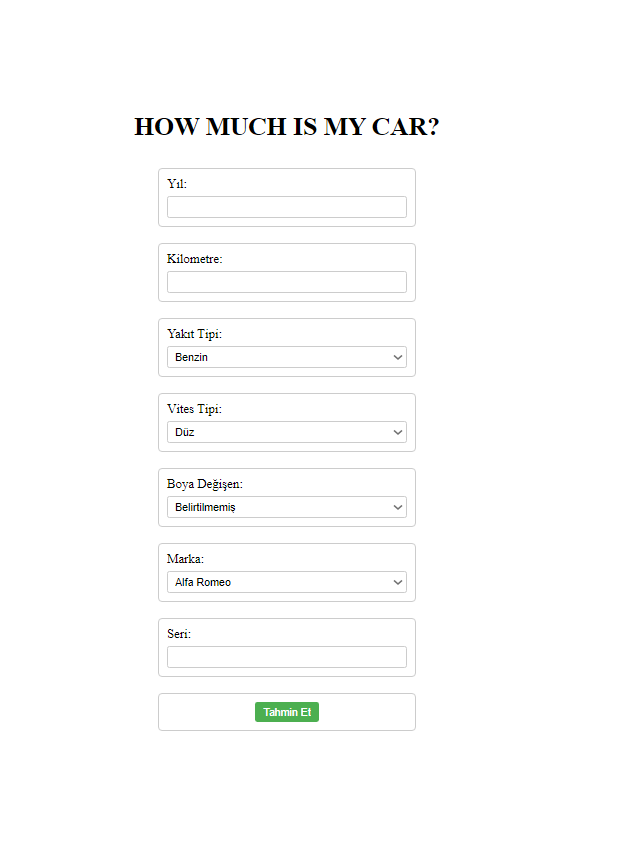
if \_\_name\_\_ == '\_\_main\_\_':

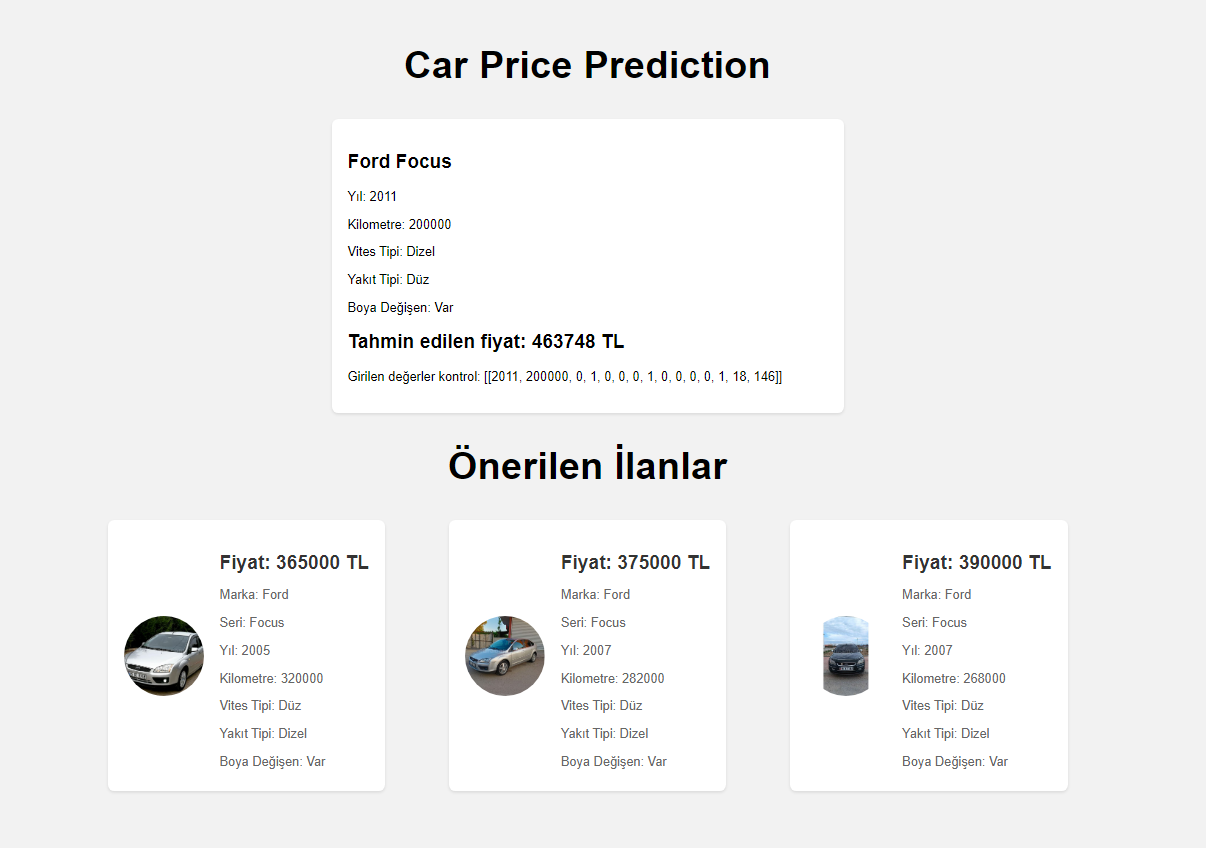
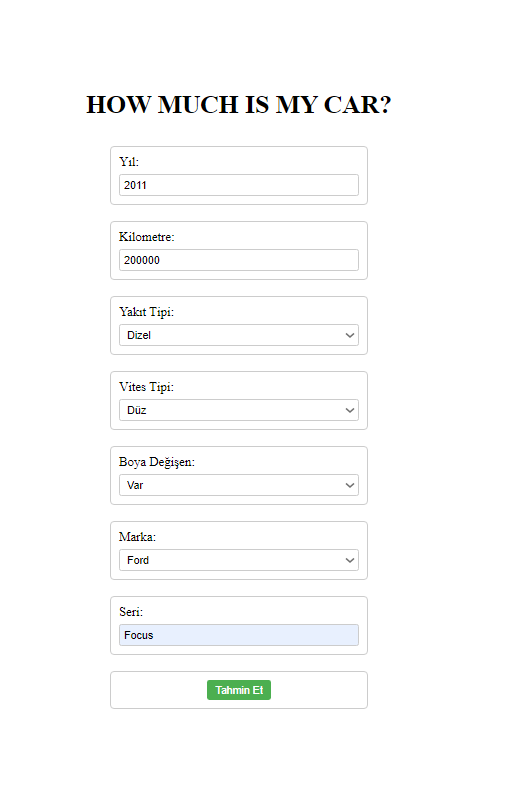
    app.run()

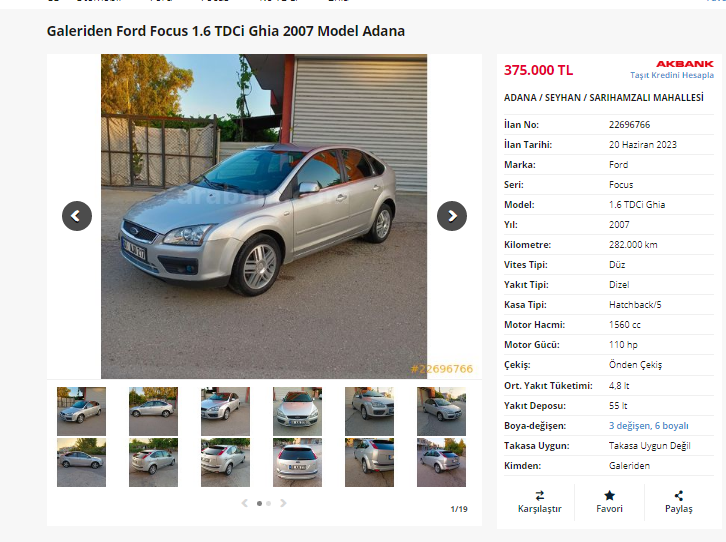
Image from Mongo DB:

******

Some images from the project:







# APPENDIX B: FORMAT OF DISKETTE, CD or DVD CONTAINING COMPUTER SOFTWARE

* <https://github.com/emirhanbal/CarPricePredictionModel.git>
* <https://how-much-is-my-car.onrender.com>