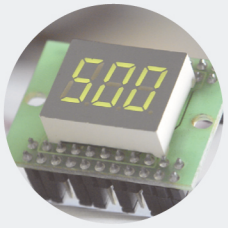
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Agregamento e Análise de Dados de Automóveis

JOÃO LUÍS MAGALHÃES DA SILVA
outubro de 2024

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Web Scraping and Analysis of Car Data

João Silva

Master in Electrical and Computer Engineering
Specialization Area of Systems And Industrial Planning

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

October, 2024

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Systems And Industrial Planning.

Candidate: João Silva, No. 1161559, 1161559@isep.ipp.pt

Scientific Guidance: Susana Nicola, sca@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

October, 2024

Acknowledgements

I would like to express my heartfelt gratitude to my professor and supervisor, Susana Nicola, for her invaluable insights and guidance throughout this process. A special thanks to my colleague Henrique Rocha for his assistance with LaTeX, and to Rodrigo Teixeira for his technical advice and support.

Abstract

The growth of online car marketplaces has created challenges in efficiently gathering and analyzing car data due to price fluctuations and increasing digital reliance. This thesis tackles the problem through web scraping and data analysis to assist in market insights. A review of web scraping tools like BeautifulSoup, Requests, and Selenium, alongside data analysis libraries such as Pandas, was conducted.

A system was developed to scrape car data from Standvirtual and analyze key attributes like price and mileage. The data was processed using Python tools, and a Flask-based server application was built for easy access, with offline analysis supported through Excel.

Challenges such as incomplete data and anti-scraping measures were resolved with advanced extraction techniques and error handling. Further improvements include optimizing the scraping process and integrating machine learning models for more accurate price predictions.

In conclusion, the project demonstrates the potential of web scraping for car market analysis, providing a foundation for future predictive analytics and real-time data applications.

Keywords: Web Scraping, Data Analysis, Application, Data Extraction, Data Insertion, Python Libraries, Car Data

Resumo

O crescimento dos sites de venda de automóveis online criou desafios na recolha e análise de dados de veículos devido à oscilação de preços e à dependência digital. Este projeto aborda este problema por meio de técnicas de web scraping e análise de dados para obter mais informações e conhecimento sobre o mercado. Foi realizado um estudo sobre a ferramentas de web scraping, como Requests, BeautifulSoup e Selenium, juntamente com bibliotecas de análise de dados como Pandas.

Um sistema foi desenvolvido para realizar um agregamento de dados de automóveis do site Stanvirtual e analisar os detalhes chave como preço e quilometragem. Estes dados foram processados utilizando bibliotecas Python, foi também desenvolvida uma aplicação em Flask para facilitar o acesso a esta análise, assim como um ficheiro excel com o propósito de realizar a análise offline.

Desafios como dados incompletos e medidas anti-scraping foram resolvidos com técnicas de extração e tratamento de erros. Melhorias futuras incluem a otimização do processo de agregamento de dados e a integração de modelos de machine learning para previsões de preços mais precisas.

Concluindo, o projeto demonstra o potencial do web scraping para a análise de mercado automóvel, fornecendo uma base para futuras aplicações de análise preditiva e processamento de dados em tempo real.

Palavras-Chave: Agregamento de Dados, Análise de Dados, Aplicação, Extração de Dados, Inserção de Dados, Bibliotecas Python, Dados de Automóveis

Contents

List of Figures

Listings

List of Acronyms

ACID	*Atomicity, consistency, isolation and durability*
AJAX	*Asynchronous Javascript And XML*
API	*Application Programming Interface*
CAPTCHA	*Completely Automated Public Turing test to tell Computers and Humans Apart*
CSRF	*Cross-site Request Forgery*
CSS	*Cascading Style Sheets*
CSV	*Comma-separated values*
DSR	*Design Science Research*
EFB	*Exclusive Feature Bundling*
EV	*Electric Vehicle*
GOSS	*Gradient-Based One-Side Sampling*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
ID	*Identification*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
KBB	*Kelley Blue Book*
LAMP	*Linux, Apache, MySQL, PHP/Python/Perl*

LightGBM *Light Gradient-Boosting Machine*

LLM *Large Language Model*

MATLAB *Matrix Laboratory*

MVT *Model View Template*

ORM *Object-Relational Mapping*

RDBMS *Relational Database Management Systems*

REST *Representational State Transfer*

SQL *Structured Query Language*

ToS *Terms of Service*

URL *Uniform Resource Locator*

VBA *Visual Basic for Applications*

WSGI *Web Server Gateway Interface*

XGBoost *Extreme Gradient Boosting*

XML *Extensible Markup Language*

XSS *Cross-site Scripting*

Chapter 1

Introduction

This chapter introduces the topic of Web Scraping and Data Analysis in the car market. Section 1.1 presents the background of this technology and its various use cases. Section 1.2 outlines the motivations behind the project, while Section 1.3 defines the objectives, section 1.4 introduces the methodology, section 1.5 provides a timeline of the work completed and section 1.6 summarizes the overall structure of the thesis.

1.1 Background

Web scraping and data analysis technologies have become essential tools for extracting and understanding detailed information about cars, especially in today's dynamic automotive market. As the industry experiences significant price fluctuations, the need for timely and accurate data has grown. These price changes are influenced by various interconnected factors, including shifts in economic conditions, such as inflation and interest rates, as well as evolving consumer preferences, like the rising demand for electric vehicles. Additionally, regulatory changes—ranging from

environmental standards to safety requirements—can further impact vehicle pricing and availability.

In this complex environment, businesses such as car dealerships, online marketplaces, and insurance companies, as well as individual consumers, must stay informed about current trends and vehicle valuations to make well-informed decisions. For instance, dealerships rely on accurate data to adjust inventory pricing, while buyers use it to gauge fair market values or identify the best time to purchase. Therefore, the ability to quickly and accurately gather, process, and analyze car data has shifted from being a competitive advantage to a necessity for navigating the ever-changing automotive landscape.

1.2 Motivations

The increasing reliance on digital platforms in the car-buying process has reshaped how consumers engage with the market. More and more buyers are shifting toward online research to compare prices and evaluate car features. This trend is largely driven by the convenience of accessing comprehensive information from car marketplaces at any time. As a result, the demand for automated systems to collect and analyze this vast amount of data has become essential, offering valuable insights into market dynamics, car pricing patterns, and consumer preferences.

Furthermore, online car marketplaces now serve as a primary resource for buyers to gather information about various car models, specifications, and pricing options. This shift is particularly pronounced among electric vehicle (*Electric Vehicle* (EV)) buyers, who often rely more heavily on online research due to the specific details they need, such as battery life, charging infrastructure, and energy efficiency comparisons. As buyers increasingly rely on digital tools for decision-making, the importance of web scraping becomes clear. It enables the extraction of large datasets from these platforms, allowing for real-time analysis of factors such as price trends, mileage, and vehicle condition, which would otherwise be impossible to track manually.

With the growing comfort in completing major portions of the vehicle purchase process online, there is a need, from buyers and sellers, for automated tools that can provide personalized and detailed insights based on consumer preferences. This project addresses that need by studying and using web scraping techniques to

gather comprehensive datasets and applying analysis methods to generate actionable insights, ensuring that buyers and sellers can navigate the car market more effectively[1][2].

1.3 Aims and Objectives

This project aims to develop a program capable of extracting and displaying the analysis of the data extracted.

Therefore the project's objectives are:

- Use web scraping techniques to extract data related to car details from a online car marketplace.
- Analyze the data collected, comparing the prices with different aspects of each car.
- Elaborate an server-based application where it will be possible to display the analysis made.
- Develop solution that enables users to perform the same analysis offline without the need to connect with the server-based application.
- To study possible improvements and further functionalities of the developed application.

1.4 Methodology

The methodology for this project follows the *Design Science Research* (DSR) approach, which is focused on solving real-world problems through the creation and evaluation of artifacts. In this case, the artifact is a web scraping and data analysis tool for collecting car data and presenting insights both online and offline.

DSR is a method used to solve practical problems by designing and testing new solutions or tools. It's commonly applied in fields like technology, engineering, and

information systems, where researchers aim to create something useful while also expanding scientific knowledge[3].

Below, in figure 1.1, is the framework relevant for this project.

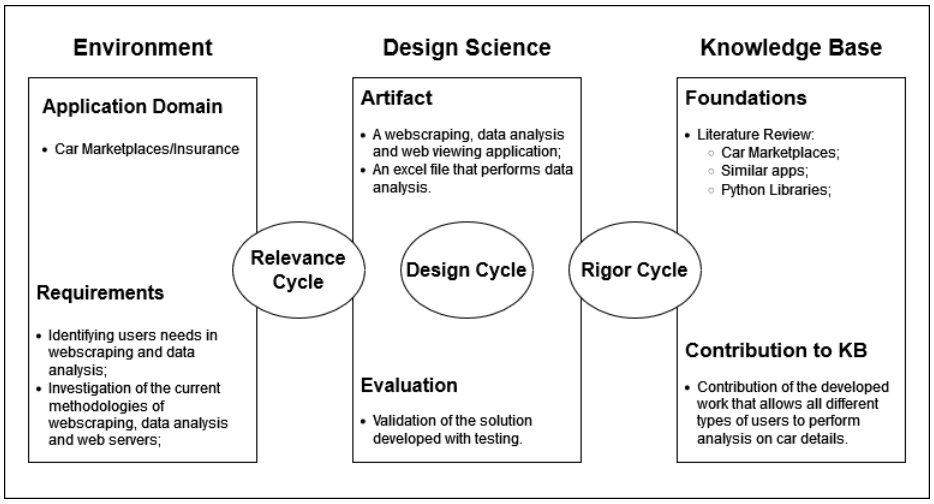


Figure 1.1: DSR Framework

In this project, DSR is used to:

- **Create a Tool**: A web scraping and data analysis application for collecting and analyzing car data.
- **Solve a Real Problem**: Automate the process of gathering car listings and provide insights into market trends.
- **Test**: Test the tool's accuracy and functionality.
- **Expand Knowledge**: Contribute to the broader knowledge of how web scraping and data analysis can be applied in car marketplaces, providing a useful tool for others in the field.

1.5 Work Plan

This project was initially proposed in March 2024 and the workflow and subsequent tasks followed this timeline:

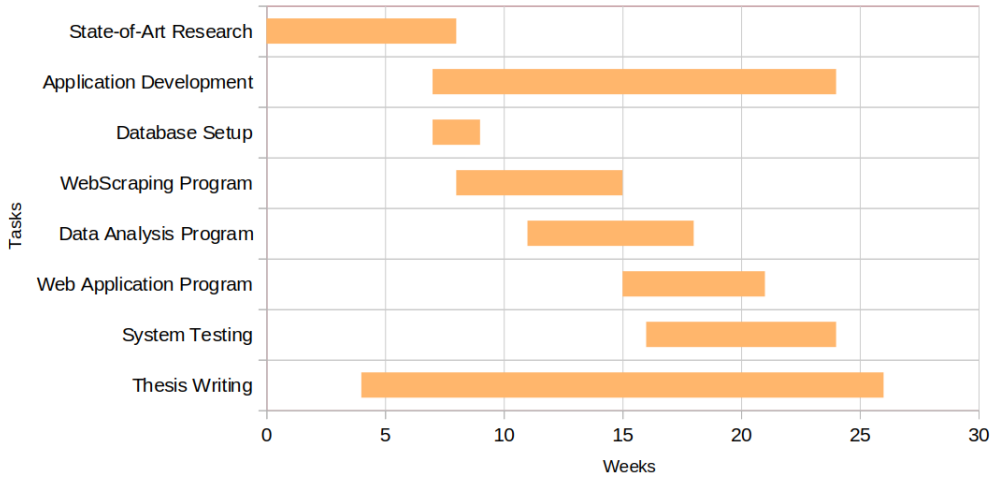


Figure 1.2: Workplan

1.6 Thesis Structure

This Thesis is divided into five chapters.

Chapter 1 provides an overview of the project's motivation, explores various contemporary applications of web scraping and data analysis, and outlines the project's objectives. Chapter 2 reviews the prevalent uses of web scraping and data analysis in car marketplaces, including price prediction tools within the automotive industry. It also covers the most widely used tools and techniques in web scraping, data analysis, server applications, and data management systems. Chapter 3 details the methodology employed in developing the application. It begins with data extraction, thoroughly explaining the workflow and techniques utilized. It then describes the analysis of the collected data and the development of the server-based application that facilitates the visualization of this analysis. This chapter also addresses the challenges encountered and the validation processes required for each step. Chapter 4 analyzes and discusses the results obtained from the application developed. Chapter 5 presents the overall conclusions derived from the project's development and proposes potential enhancements to improve the functionality of the developed application.

Chapter 2

Literature Review

This literature review provides a comprehensive overview of the current research and practical applications in the field of web scraping and data analysis, with a particular focus on car data.

Section 2.1 explores the current state of existing applications and their capabilities, particularly focusing on their use of web scraping and data analysis techniques. It highlights how these platforms aggregate and analyze automotive data to provide valuable insights and services. Section 2.2 delves into the most commonly used tools for web scraping, data analysis, server applications and data management systems, with an emphasis on Python libraries. This section provides an overview of the technologies and frameworks employed to automate data collection, process large datasets, and deploy scalable server-based solutions.

This chapter aims to provide the reader with a deeper understanding of the current state of applications, particularly car marketplaces and price prediction tools, and how they leverage web scraping and data analysis techniques. By examining both the existing platforms and the tools used in these processes, this review offers more understanding on the methodologies and technologies driving advancements in automotive data solutions.

2.1 Web Scraping and Data Analysis of Car Data

Web scraping is essential for collecting large volumes of car-related data from various online sources such as car marketplaces, classified ads, auction sites, and automotive review platforms. By automating the extraction of information like vehicle specifications, pricing, mileage, and seller details, web scraping allows analysts and users to access real-time, comprehensive datasets that would otherwise be difficult to obtain manually. This data forms the foundation for further analysis, enabling platforms to deliver insights such as price predictions, market trends, and competitive analysis, ultimately making complex automotive data accessible and actionable for both professionals and regular users. The automotive industry is highly data-driven, with details about car models, pricing, mileage, availability, and condition being crucial for consumers, dealerships, and market analysts alike. Web scraping allows users to automatically collect this information from websites, which would otherwise require manual data entry or limited *Application Programming Interface* (API) access [2].

In the context of the automotive market, scraping can help in analyzing and tracking real-time trends, such as fluctuating car prices, availability of specific models, or regional market variations. This data is useful for numerous applications, including:

- **Price prediction tools** : Estimating the current and future market value of cars based on historical pricing data and trends.
- **Market research** : Providing dealerships and individual buyers insights into supply and demand patterns across different regions.
- **Competitive analysis** : Tracking competitors' listings and price strategies for similar vehicles.

Web scraping can be used to collect a multitude of car-related data for future analysis such as:

- **Vehicle specifications** (Car brands, models, year, displacement).

- **Mileage and condition**.
- **Pricing information**.
- **Vehicle history data** (If the application provides data on vehicle history reports).
- **Seller information**.

2.2 Existing Applications

Currently there are several prominent applications and tools that rely on car data aggregation to provide valuable insights into the automotive market. These platforms range from car marketplaces to price prediction tools. Below are some of the key categories and examples of relevant applications that available nowadays:

2.2.1 Car MarketPlaces

Car marketplaces like Standvirtual, Autotrader and Cars.com are some examples of platforms where consumers and dealerships list new and used vehicles for sale. These websites aggregate vast amounts of data, including:

- **Vehicle specifications** - Car brands, models, year, displacement.
- **Mileage and condition** - Providing dealerships and individual buyers insights into supply and demand patterns across different regions.
- **Pricing data** - List prices and price trends.
- **Seller information** - Dealerships and individual sellers.
- **Geolocation-based searches** - Filter cars based location of the seller and buyer.

Car Marketplaces can be a great way to collect car details, usually this data is already correctly structured and validated, making it easier to develop a web scraping tool.

There are also platforms like Facebook Marketplace, OLX and CustoJusto that are popular among used car sellers, however unlike car marketplaces, these platforms

don't provide the same level of sophistication in car data validation, meaning that scraping will be more challenging and less trustworthy.

2.2.2 Price Prediction Tools

Tools like *Kelley Blue Book* (KBB) and Edmunds provide users with real-time price estimates for both buying and selling cars [4]. These platforms use vast amounts of historical and current market data to forecast the fair value of a car based on factors such as:

- **Brand, model, year and engine displacement**
- **Mileage and condition**
- **Geographical location**

These price prediction platforms often use a mix of API data and web scraping to continuously update their pricing algorithms. Scraping allows them to aggregate real-time pricing data from multiple sources, ensuring accurate and up-to-date information for users. Edmunds provides APIs for dealer partners, which can help them develop a custom made tool for their own needs, these APIs are used to obtain a set of content elements, in the form of a *JavaScript Object Notation* (JSON) data object, that can be integrated to the dealer's website, Edmunds also provide some support and simple tutorials to integrate the API to the partner's application/website [5].

2.3 Common Used Tools

2.3.1 WebScraping - Requests Library

Requests is a widely used Python library designed to facilitate *Hypertext Transfer Protocol* (HTTP) requests, which are fundamental for web scraping. It provides a user-friendly interface for sending and receiving HTTP requests, making it easier to interact with web servers and retrieve data. It is specifically designed to simplify the process of making several types of HTTP requests, such as GET, POST, PUT, and DELETE, which are commonly required in web scraping tasks. Its primary purpose is to enable users to access web content and APIs effortlessly, allowing them to focus on data extraction and analysis rather than the intricacies of HTTP protocols [6].

Key Features:

- **Ease of Use** – Requests offers a straightforward API that simplifies the process of sending HTTP requests and handling responses. This user-friendly approach makes it accessible to developers of all skill levels.
- **Response Handling** – It provides convenient methods for handling HTTP responses. Users can easily access various components of the response, such as the content, either *HyperText Markup Language* (HTML) or JSON, headers, and status codes. This capability is crucial for extracting and processing data from web pages.
- **Error Handling** – The library includes robust mechanisms for managing HTTP errors and exceptions. This functionality helps users with issues such as network problems, timeouts, and server errors gracefully, ensuring more reliable and resilient scraping operations.
- **Session Management** – Requests supports session management, which allows users to maintain persistent connections and reuse cookies across multiple requests. This feature is especially useful for interacting with websites that require user authentication or maintain session state.

Other Considerations:

- **Managing Dynamic Content** – The Requests library is designed for fetching static content and cannot process JavaScript or content that is dynamically loaded after the initial page request. To access such dynamic content, additional tools or libraries, such as Selenium or Puppeteer, are often required to fully render and capture the data.
- **Rate Limiting** – Frequent or aggressive scraping can lead to rate limiting or blocking by the server. To avoid this, it is essential to implement strategies such as respectful request intervals.
- **Data Extraction** – While Requests facilitates data retrieval, it does not include tools for parsing HTML or other data formats. To extract meaningful data from the content retrieved, users often need to employ additional libraries [6].

2.3.2 WebScraping - BeautifulSoup Library

BeautifulSoup is a popular Python library used for parsing HTML and *Extensible Markup Language* (XML) documents. It creates parse trees from page source code that can be used to extract data from HTML, which is particularly useful in web scraping tasks. The library is well-known for its ease of use and ability to handle poorly structured or broken markups, making it an essential tool for scraping data from a wide variety of web sources [7]. BeautifulSoup plays a critical role in the web scraping process by providing the functionality to navigate and manipulate HTML content:

1. **Parsing HTML and XML Documents**

   - **Parsing Capabilities** – BeautifulSoup can parse HTML and XML documents, converting them into a tree of Python objects such as tags, navigable strings, and comments. This makes it easy to search and modify the HTML content.
   - **Managing Malformed Markup** – One of BeautifulSoup's strengths is its ability to manage imperfect or broken HTML, which is common on the web. This feature ensures that data can be extracted even from less-than-ideal sources.

2. **Navigating the Parse Tree**

   - **Tag Navigation** – BeautifulSoup provides methods to search the parse tree for tags, attributes, text, and more, allowing precise extraction of data elements such as product names, prices, and descriptions.
   - **Search Methods** – It includes powerful search methods like "$.find()$" and "$.find_all()$" to locate specific tags, classes, *Identification* (ID), or other attributes, and *Cascading Style Sheets* (CSS) selectors for more complex queries. This is crucial for targeting specific parts of a webpage that contain the data of interest.

3. **Modifying HTML**

   - **Tag Modification and Creation** – BeautifulSoup allows users to modify the parse tree by adding, editing, or deleting tags and attributes. This can be useful for cleaning up HTML or preparing it for further processing.
   - **Text Extraction** – It provides straightforward methods to extract the text content from tags, which is often necessary when scraping textual data for analysis.

4. **Integration with Other Libraries**

   - **Interoperability with Requests** – BeautifulSoup is often used in conjunction with the Requests library to fetch and parse web pages. Requests can retrieve the raw HTML, which BeautifulSoup then parses and processes.
   - **Data Cleaning and Preparation** – After extracting data with BeautifulSoup, it can be cleaned and transformed using other libraries like Pandas for further analysis [7][8].

2.3.3 WebScraping - *Comma-separated values* (CSV) Library

The CSV library in Python is a standard tool for handling CSV files, most commonly excel. It is essential for managing and analyzing tabular data, which is frequently encountered in data analysis tasks, including those involving car data. Definition and Purpose: The CSV library provides functionality to read from and write to CSV files, which are a common format for storing structured data in a simple, text-based manner. Each line in a CSV file represents a record, and fields within that record are separated by commas (or other delimiters). The library's primary purpose is to facilitate the import, export, and manipulation of such data, making it a critical tool for data analysis and preprocessing [9][10].

Key Features:

- **Reading CSV Files** – The library allows users to easily read data from CSV files into Python. It provides functionality to manage different delimiter characters, manage header rows, and process large files efficiently.

- **Writing CSV Files** – Users can also write data to CSV files, enabling the export of processed or analyzed data in a structured format. The library supports customization of delimiters, quoting styles, and newline characters, catering to various needs and standards.
- **Managing Different Formats** – The CSV library can manage various CSV formats, including those with different delimiters (such as tabs or semicolons) and varying quoting conventions. This flexibility is important for dealing with diverse data sources.
- **Data Manipulation** – While the CSV library itself focuses on reading and writing files, it integrates seamlessly with other Python libraries (like Pandas) for further data manipulation and analysis. This integration allows for more advanced operations, such as filtering, aggregation, and visualization.

Other Considerations:

- **Data Quality** – CSV files may contain inconsistencies or errors, such as missing values, incorrect delimiters, or improperly escaped characters. Managing such issues requires careful preprocessing to ensure data integrity.
- **Performance with Large Files** – When working with large CSV files, performance can become an issue. The CSV library is suitable for many use cases, but more substantial datasets might require more efficient methods or additional tools to manage memory usage and processing time effectively.
- **Limited Data Types** – CSV files are inherently limited in their ability to represent complex data types. All data is stored as text, which can require additional steps to convert data into appropriate types, for example dates or numerical values, for analysis [9].

2.3.4 WebScraping - Selenium Library

The Selenium library is a powerful tool in Python used for automating web browser interactions. It is particularly valuable in web scraping for dealing with dynamic content and web pages that rely heavily on JavaScript to render information. Definition and Purpose: Selenium is designed to automate web browsers, allowing

developers to programmatically interact with web pages as a human user would. It is often employed in web scraping to navigate websites, simulate user actions like clicking buttons or filling out forms, and capture content that is dynamically loaded by JavaScript, which cannot be easily accessed through simpler HTTP request methods[11].

Key Features:

- **Browser Automation** – Selenium can control various web browsers (like Chrome, Firefox, and Edge) to perform tasks such as navigating to an *Uniform Resource Locator* (URL), clicking elements and scrolling pages. This is crucial for scraping websites that require interaction to reveal the data of interest.
- **Handling Dynamic Content** – Unlike basic web scraping libraries that only fetch static HTML, Selenium can render JavaScript-heavy websites fully, allowing the extraction of data that appears only after scripts are executed. This makes it essential for scraping modern, interactive web applications.
- **Form Submission and User Interaction** – Selenium can simulate complex user interactions, such as logging in, submitting forms, or selecting options from drop-down menus. This capability is necessary for scraping content behind login walls or interacting with search filters.
- **Taking Screenshots and Managing Cookies** – Selenium can capture screenshots of web pages, which is useful for debugging or documentation. It also handles cookies, enabling persistent sessions and maintaining login states across multiple interactions.

Other Considerations:

- **Performance Overhead** – Selenium operates by controlling a web browser, which is more resource-intensive and slower compared to libraries that directly send HTTP requests. This can make it less efficient for large-scale scraping tasks unless absolutely necessary.
- **Complex Setup and Maintenance** – Setting up Selenium can be more complex than other scraping tools, especially when configuring browser drivers

or managing headless browsing. Additionally, it requires ongoing maintenance to adapt to changes in website structure or browser updates.

- **Anti-Scraping Measures** – Since Selenium mimics human browsing behavior, it can sometimes trigger anti-bot protections, such as *Completely Automated Public Turing test to tell Computers and Humans Apart* (CAPTCHA) or rate limiting. Handling these challenges often requires additional strategies, such as integrating CAPTCHA-solving services or using proxy servers to rotate IP addresses [11].

2.3.5 WebScraping - Scrapy Library

The Scrapy library is an open-source Python framework specifically designed for web scraping. It is a highly efficient and versatile tool that allows users to extract data from websites, process it, and store it in a structured format. Definition and Purpose: Scrapy is a powerful and flexible framework used to build web crawlers that can navigate websites and extract data in a systematic and scalable manner. It is designed to handle large-scale scraping projects by providing robust features for handling requests, managing data pipelines, and dealing with complex website structures[12].

Key Features:

- **Crawler and Spider Management** – Scrapy enables the creation of spiders, which are autonomous web crawlers that follow links and scrape data across multiple pages. This feature is crucial for efficiently gathering data from entire websites or sections of websites.

- **Request Handling and Scheduling** – Scrapy manages and schedules requests in an optimized manner, ensuring that scraping operations are both fast and respectful of the target website's infrastructure. It handles concurrency, retries, and can automatically throttle requests to avoid overwhelming servers.

- **Data Pipelines** – Scrapy offers a robust pipeline system to process and clean the scraped data before it is stored. This allows for data to be filtered, normalized, and saved in various formats like JSON, CSV, or directly into databases, making it ready for analysis.

- **Middleware and Extensions** – Scrapy is highly extensible, allowing users to customize and extend its capabilities through middleware and extensions. This includes handling cookies, managing user-agent strings, and implementing proxies to avoid detection and blocking.

- **Built-in Support for Export Formats** – Scrapy natively supports exporting scraped data into multiple formats such as JSON, XML, and CSV, streamlining the process of data storage and subsequent analysis.

Other Considerations:

- **Learning Curve** – Although Scrapy is powerful, it has a steeper learning curve compared to simpler scraping libraries. Understanding its architecture, including spiders, pipelines, and middleware, requires some familiarity with web scraping concepts and Python programming.

- **Handling JavaScript-Driven Websites** – Scrapy excels at scraping static content but struggles with JavaScript-heavy websites where content is loaded dynamically. For such cases, it may need to be used in conjunction with tools like Selenium to fully render and capture the necessary data.

- **Anti-Scraping Measures** – As with all web scraping tools, Scrapy can encounter anti-scraping mechanisms such as CAPTCHAs, rate limiting, and *Internet Protocol* (IP) blocking. While Scrapy's extensible middleware allows for some mitigation, these challenges often require additional strategies like rotating proxies or integrating CAPTCHA-solving services[13].

2.3.6 Data Analysis - Pandas Library

The Pandas library is a widely-used open-source data manipulation and analysis tool in Python. It provides data structures and functions needed to work with structured data, making it essential for tasks involving data cleaning, transformation, and analysis, particularly in the context of car data. Pandas is designed to facilitate the manipulation of large datasets with ease and efficiency. It offers powerful data structures, primarily the DataFrame, which is similar to a table in a relational database or an Excel spreadsheet. Pandas is fundamental for analyzing data because it allows users to import, clean, transform, and analyze data quickly, making it a cornerstone of data science workflows[14].

Key Features:

- **DataFrames and Series** – The DataFrame is the core data structure in Pandas, allowing users to store and manipulate two-dimensional, labeled data. Series, another key structure, handles one-dimensional data. These structures are versatile, allowing for the easy handling of different data types, including numerical, categorical, and time-series data.

- **Data Import and Export** – Pandas supports importing data from and exporting data to various formats, including CSV, Excel, JSON, *Structured Query Language* (SQL) databases, and more. This capability is crucial for integrating car data from different sources and formats into a unified analytical process.

- **Data Cleaning and Transformation** – Pandas provides extensive tools for cleaning and transforming data, such as handling missing values, filtering data, merging datasets, and reformatting columns. These functions are essential for preparing car data for analysis, ensuring it is in a usable and consistent format.

- **Descriptive Statistics and Aggregation** – Pandas offers a wide range of functions for generating descriptive statistics, such as mean, median, and standard deviation, as well as for aggregating data. These features allow for the easy summarization and exploration of car data, such as calculating average prices or identifying trends over time.

- **Data Visualization Integration** – While not a visualization tool itself, Pandas integrates well with libraries like Matplotlib and Seaborn, making it straightforward to create visual representations of data. This is useful for visualizing trends, distributions, and other key insights from car data.

Other Considerations:

- **Memory Consumption** – Pandas is memory-intensive, particularly with large datasets. When working with very large car datasets, users may encounter memory limitations, which require careful management or the use of alternative tools like Dask for parallel processing.
- **Performance with Large Datas** – While Pandas is efficient for many operations, its performance can degrade with very large datasets or complex operations. Optimizing code or using more efficient data structures might be necessary to handle extensive car data efficiently.
- **Learning Curve** – Although Pandas is powerful, it has a steep learning curve, especially for beginners. Understanding how to effectively use DataFrames and Series, and how to leverage Pandas' full range of functions, can take time and practice[14][15].

2.3.7 Data Analysis - Matplotlib.pyplot Library

The Matplotlib.pyplot library is a widely-used Python tool for creating static, interactive, and animated visualizations. It is part of the larger Matplotlib library and is essential for visualizing data, including car data, to uncover patterns, trends, and insights. Matplotlib.pyplot is a collection of functions that make Matplotlib work like *Matrix Laboratory* (MATLAB), enabling users to create a wide range of plots and charts easily. The library is designed to offer extensive control over the appearance of plots, making it a versatile tool for creating publication-quality visualizations. In the context of car data analysis, Matplotlib.pyplot helps in visualizing trends such as price distributions, sales over time, and comparisons across different car models or features[16].

Key Features:

- **Wide Range of Plot Types and Series** – Matplotlib.pyplot supports a variety of plot types, including line plots, scatter plots, bar charts, histograms, pie charts, and more. This flexibility allows users to choose the most appropriate visualization for their data, whether it's tracking price trends, comparing features, or displaying distributions.
- **Customization Options** – The library offers extensive customization options, including control over colors, labels, markers, line styles, and axes. This allows users to tailor plots to meet specific presentation needs, such as highlighting particular data points or adjusting plots for clarity and visual appeal.
- **Integration with Pandas** – Matplotlib.pyplot integrates seamlessly with Pandas, allowing for easy plotting of data stored in DataFrames. This is particularly useful for quickly visualizing large datasets, such as car sales data, with minimal additional code.
- **Subplots and Layouts** – Users can create multiple plots within a single figure using subplots, allowing for the comparison of different data sets or variables side by side. This feature is helpful for presenting comprehensive views of car data, such as sales trends across different regions or model years.
- **Interactive Plots** – Although Matplotlib.pyplot primarily creates static images, it also supports interactive features like zooming and panning, which can enhance data exploration and presentation. This is useful for more detailed examination of car data trends and outliers.

Challenges and Considerations:

- **Learning Curve** – While Matplotlib.pyplot is powerful, it can be complex to master, particularly for those new to data visualization. Understanding the multitude of customization options and plot types may require practice and study.
- **Complex Syntax** – Creating highly customized plots often involves complex and verbose code. For beginners or for simpler visualizations, this can be a

barrier, prompting some users to turn to higher-level libraries like Seaborn for more straightforward plotting.

- **Static Plots** – By default, Matplotlib.pyplot generates static plots, which might not be sufficient for all use cases. For more dynamic or interactive visualizations, additional tools or libraries, such as Plotly, might be necessary[17].

2.3.8 Data Analysis - Seaborn Library

The Seaborn library is a Python data visualization library built on top of Matplotlib. It is specifically designed to make creating attractive and informative statistical graphics easier and more straightforward. Seaborn is particularly useful for visualizing complex data relationships, such as those found in car data analysis, with less effort compared to raw Matplotlib. Seaborn provides a high-level interface for drawing attractive and informative statistical graphics. It simplifies the process of creating complex plots and integrates seamlessly with Pandas, making it easier to visualize data stored in DataFrames. Seaborn is particularly effective for exploring relationships between multiple variables, visualizing distributions, and creating multi-plot grids, which are often needed in car data analysis to understand trends, correlations, and patterns[18].

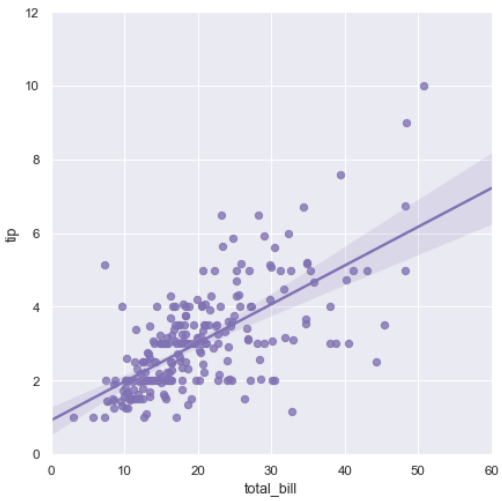


Figure 2.1: Seaborn - Linear Regression [18]

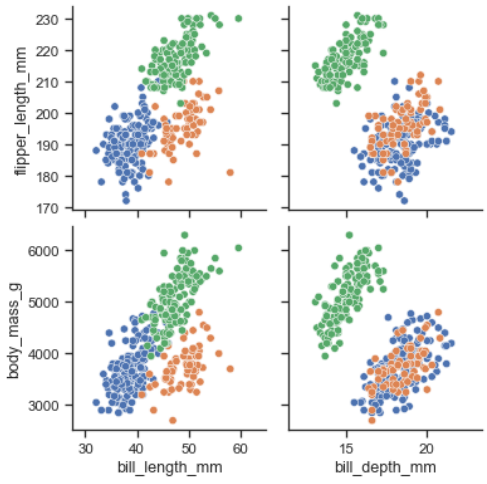


Figure 2.2: Scatter Plot Graph using Seaborn [18]

Key Features:

- **Enhanced Aesthetics** – Seaborn automatically applies aesthetically pleasing color palettes and styles to plots, resulting in more polished and visually appealing graphics. This feature reduces the amount of manual customization needed to create professional-looking visuals.
- **Built-in Themes and Color Palettes** – Seaborn includes several built-in themes and color palettes that can be easily applied to plots to enhance their appearance. This is particularly useful when visualizing car data in presentations or reports where clarity and visual impact are important.
- **Statistical Plotting Functions** – Seaborn simplifies the creation of complex statistical plots, such as box plots, violin plots, and pair plots, which are useful for comparing car features, analyzing price distributions, and exploring relationships between different variables.
- **Seamless Integration with Pandas** – Seaborn works directly with Pandas DataFrames, allowing users to pass DataFrame objects directly to plotting functions. This makes it easier to plot complex data structures, such as those found in car datasets, without needing extensive data preparation.
- **Multi-Plot Grids** – Seaborn supports the creation of multi-plot grids, enabling users to display multiple related plots in a single figure. This is useful for comparing different subsets of car data, such as visualizing trends across different car brands or model years in a cohesive layout.

Other Considerations:

- **Less Customization Flexibility** – While Seaborn simplifies plot creation and provides beautiful default styles, it offers less flexibility than Matplotlib for highly customized plots. For users who need very specific plot adjustments, it might be necessary to combine Seaborn with Matplotlib.
- **Learning Curve for Complex Plots** – Although Seaborn is designed to be user-friendly, creating highly customized or complex plots still requires a good understanding of the library and its interaction with Matplotlib. Some users may find this challenging, especially when moving beyond basic visualizations.

- **Performance with Large Datasets** – Seaborn can be slower when handling very large datasets due to its high-level abstraction and the underlying statistical computations. In such cases, optimizations or the use of more performance-oriented tools may be required[19][20].

2.3.9 Data Analysis - sklearn.linear_model Library

The sklearn.linear_model module is a part of the scikit-learn library, which is widely used in Python for implementing machine learning algorithms. This module focuses on linear models, offering various tools for modeling relationships between variables, making it essential for predictive modeling and regression analysis in data analysis tasks, including car data analysis. The sklearn.linear_model library provides a suite of algorithms for linear regression and classification, which are foundational techniques in machine learning. These models are used to predict outcomes based on input features by assuming a linear relationship between the dependent and independent variables. In the context of car data, sklearn.linear_model can be used to build models that predict car prices, assess the impact of various features on car performance, or classify cars based on certain characteristics[21].

Key Features:

- **Linear Regression** – This is one of the most fundamental models in the sklearn.linear_model module. It predicts a continuous target variable based on one or more input features, assuming a linear relationship. Linear regression is commonly used in car data analysis to predict car prices based on factors like mileage, age, and engine size.

- **Logistic Regression** – Logistic regression is used for binary classification tasks, where the goal is to predict one of two possible outcomes. In car data analysis, logistic regression can be applied to tasks such as predicting whether a car will sell within a certain time frame or whether a car meets specific safety standards.

- **Ridge and Lasso Regression** – These are variations of linear regression that include regularization techniques to prevent overfitting, especially when dealing with datasets with a large number of features. Ridge and Lasso regression

are useful in car data analysis when dealing with complex datasets where some features may be highly correlated or not particularly informative.

- **Support for Multivariate Models** – The module supports multivariate linear models, allowing for the analysis of how multiple features interact to influence the target variable. This is particularly valuable in car data analysis, where multiple factors like engine size, fuel efficiency, and brand reputation might all contribute to the car's price.

- **Integration with Scikit-learn Pipeline** – The sklearn.linear_model module can be easily integrated into scikit-learn's pipelines, allowing for the combination of linear models with preprocessing steps like scaling or feature selection. This streamlines the process of building, training, and evaluating models on car data.

Other Considerations:

- **Assumption of Linearity** – Linear models assume a linear relationship between the independent and dependent variables, which may not always be the case in complex datasets. For non-linear relationships in car data, more advanced models, such as decision trees or neural networks, might be required.

- **Sensitivity to Outliers** – Linear models can be sensitive to outliers, which can disproportionately influence the model's predictions. Careful preprocessing of car data, such as outlier detection and removal, is necessary to ensure robust model performance.

- **Feature Scaling** – The performance of linear models can be significantly impacted by the scale of the features. Before applying linear models, it is often necessary to standardize or normalize the features, particularly in car data where features like price, mileage, and engine size can vary greatly in magnitude[21].

2.3.10 Data Analysis - *Extreme Gradient Boosting* (XGBoost) Library

The XGBoost library is a powerful and widely-used machine learning tool that specializes in gradient boosting algorithms for supervised learning tasks. It is particularly valued for its efficiency, accuracy, and ability to handle large datasets, making it a popular choice for predictive modeling in data-intensive fields such as car data analysis. XGBoost is an implementation of the gradient boosting framework designed to be highly efficient and scalable. It is primarily used for regression, classification, and ranking problems. In the context of car data analysis, XGBoost is often employed to build predictive models, such as estimating car prices based on features like mileage, age, and brand, or predicting car sales trends[22].

Key Features:

- **Gradient Boosting** – XGBoost is built on the principle of gradient boosting, where models are trained sequentially, and each new model attempts to correct the errors made by the previous ones. This iterative process helps create highly accurate models that can capture complex relationships in the data.
- **Regularization** – Includes built-in regularization techniques (L1 and L2) that help prevent overfitting, a common problem in machine learning models, especially when dealing with large and complex car datasets. This makes XGBoost robust and capable of generalizing well to new data.
- **Handling Missing Data** – One of XGBoost's strengths is its ability to handle missing data efficiently. It automatically learns how to treat missing values during training, which is particularly useful in real-world car data where missing entries can be common.
- **Scalability and Performance** – XGBoost is designed for speed and scalability. It supports parallel processing, which significantly speeds up the training process, making it suitable for large datasets. This is crucial when working with extensive car data that includes thousands or even millions of records.

- **Tree Pruning and Sparsity Awareness** – Includes tree pruning techniques that stop the growth of trees once they stop improving the model's performance. Additionally, its sparsity-aware algorithm efficiently handles sparse data, such as datasets with many zero or missing values, often found in car data.

- **Feature Importance** – Provides tools to assess the importance of different features in the predictive model. This is particularly valuable in car data analysis for understanding which factors most influence car prices, customer preferences, or sales trends.

Other Considerations:

- **Complexity of Tuning** – XGBoost provides tools to assess the importance of different features in the predictive model. This is particularly valuable in car data analysis for understanding which factors most influence car prices, customer preferences, or sales trends.

- **Computational Resources** – : Despite its efficiency, it can be resource-intensive, particularly when training very large models. Users need to ensure they have sufficient computational resources, such as memory and processing power, to handle large car datasets effectively.

- **Risk of Overfitting** – While it includes regularization techniques to combat overfitting, the risk still exists, especially with small or noisy datasets. Careful model evaluation and validation are necessary to ensure the model generalizes well to unseen data[22].

2.3.11 Data Analysis - *Light Gradient-Boosting Machine* (LightGBM) Library

The LightGBM (Light Gradient Boosting Machine) library is an advanced machine learning framework specifically designed for high-performance and scalable gradient boosting. Developed by Microsoft, it is widely used for tasks requiring fast and accurate predictions, especially when working with large datasets. In car data analysis, LightGBM is a powerful tool for building predictive models, such as estimating car

prices or predicting sales trends, due to its efficiency and ability to handle complex datasets. LightGBM is a distributed and efficient gradient boosting framework that focuses on decision tree algorithms. It is designed to be faster and more memory-efficient than traditional gradient boosting methods, making it particularly suitable for handling large-scale data and high-dimensional feature spaces common in car data analysis[23].

Key Features:

- ***Gradient-Based One-Side Sampling* (GOSS)** – LightGBM uses GOSS to reduce the number of data instances needed for training, focusing on instances with large gradients. This results in faster training without significant loss of accuracy, making it ideal for large car datasets.
- ***Exclusive Feature Bundling* (EFB)** – LightGBM can combine mutually exclusive features into a single feature, reducing the dimensionality of the data. This feature is particularly useful in car data, where numerous features like car specifications can be bundled, improving both speed and memory usage.
- **High Performance** – LightGBM is known for its high speed and low memory consumption, enabling it to handle large datasets with millions of instances and features efficiently. This performance is especially beneficial in car data analysis, where datasets can be vast and complex.
- **Support for Categorical Features** – LightGBM has native support for categorical features, meaning it can directly handle categorical data without needing to convert it to numerical form. This is particularly useful for car data, where features like brand, model, or color are often categorical.

Other Considerations:

- **Complexity of Hyperparameter Tuning** – While LightGBM offers many parameters that can be fine-tuned for optimal performance, the process can be complex and requires expertise. Proper tuning is essential for achieving the best results in car data analysis.

- **Overfitting Risk** – Due to its powerful learning capabilities, LightGBM can be prone to overfitting, especially if the dataset is small or noisy. Regularization techniques and careful cross-validation are necessary to mitigate this risk.
- **Handling of Sparse Data** – LightGBM is efficient at handling sparse data, but users need to ensure that the data is properly prepared and that categorical features are appropriately managed to avoid any potential issues[23].

2.3.12 Server - Flask Library

Flask is a lightweight web framework for Python that is widely used to build web applications and APIs. It is particularly valued for its simplicity, flexibility, and ease of use, making it an excellent choice for developers who want to create scalable web services without the overhead of a more complex framework. In the context of car data analysis, Flask can be employed to develop web-based interfaces or APIs that allow users to interact with car data, perform analyses, or deploy machine learning models in a user-friendly way. Flask is a micro-framework designed for building web applications in Python. It provides the essential components needed to create a web service, such as routing, request handling, and templating, while remaining lightweight, allowing developers to choose their tools and libraries as needed[24].

Key Features:

- • **Routing** – Flask's routing system allows developers to map URLs to functions in the code, enabling the creation of endpoints that users can interact with. For example, in a car data analysis application, specific routes can be defined to retrieve data, display analysis results, or make predictions based on user inputs.
- **Templating Engine (Jinja2)** – Flask includes the Jinja2 templating engine, which allows developers to dynamically generate HTML pages. This is useful for creating web interfaces where users can input car-related data or view the results of analyses, such as price predictions or trend visualizations.
- **Request Handling** – Flask handles HTTP requests and responses, making it easy to process user input, query databases, and return data in various formats,

JSON or HTML. This feature is essential for interacting with car data, whether it's retrieving data from a database or responding to user queries with analysis results.

- **Extensibility** – Flask is highly extensible, meaning it can be easily integrated with other libraries and tools. For instance, it can be combined with machine learning libraries like scikit-learn or XGBoost to deploy predictive models as web services, allowing users to interact with these models through a web interface.

- **Lightweight and Modular** – Flask is minimalistic, meaning it does not include unnecessary features or dependencies by default. This modularity allows developers to add only the components they need, making the application more efficient and easier to manage, especially in a focused application like car data analysis.

- ***Representational State Transfer* (REST) API Support** – Flask is well-suited for building RESTful APIs, which are often used in modern web applications. In a car data analysis program, a RESTful API built with Flask could allow other applications or services to access the analysis functionality programmatically.

Other Considerations:

- **Manual Configuration** – While Flask's flexibility is an advantage, it also means that developers must manually configure many aspects of the application, such as security and database integration. This can be a challenge for developers who prefer a more out-of-the-box solution.

- **Scalability** – Flask is suitable for small to medium-sized applications, but as the application grows, additional tools and infrastructure, for example *Web Server Gateway Interface* (WSGI) servers like Gunicorn, reverse proxies like Nginx, may be needed to ensure scalability and performance.

- **Security Considerations** – Since Flask is minimalistic, developers need to implement security features themselves, such as input validation, authentication, and protection against common web vulnerabilities. This requires careful

attention to detail to ensure the application remains secure, especially when handling sensitive car data.

- **Limited Built-In Features** – Unlike more comprehensive frameworks, Flask does not include built-in features like user authentication or database abstraction layers. Developers may need to integrate third-party extensions or write custom code to implement these features[24][25].

2.3.13 Server - Django Library

Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design. It is renowned for its "batteries-included" philosophy, meaning it comes with a wide range of built-in features and tools that make it easier to develop robust web applications. In the context of car data analysis, Django can be used to create comprehensive web applications that manage, analyze, and visualize car data, as well as deploy machine learning models in a secure and scalable manner. Django is a full-featured web framework that simplifies the process of building complex, database-driven websites and web applications. It handles many of the common tasks associated with web development, such as routing, authentication, and database management, allowing developers to focus on writing the application-specific code[26].

Key Features:

- ***Model View Template* (MVT) Architecture** – Django follows the MVT architecture, which separates the application's data (Model), the user interface (View), and the control logic (Template). This structure helps in organizing the codebase and maintaining a clear separation of concerns, which is particularly useful in a car data analysis application where different components, such as data models, user views and analytics, need to be managed effectively.

- ***Object-Relational Mapping* (ORM)** – Django's powerful ORM allows developers to interact with databases using Python code instead of SQL. This feature makes it easier to manage car data, enabling the creation of complex queries and data manipulations directly in Python, while still being database-agnostic.

- **Admin Interface** – One of Django's standout features is its automatically generated admin interface, which provides a user-friendly web interface for managing application data. For car data analysis, this means that you can easily manage datasets, user access, and other administrative tasks without needing to build a custom interface.
- **Security Features** – Django includes many built-in security features, such as protection against SQL injection, *Cross-site Scripting* (XSS), and *Cross-site Request Forgery* (CSRF). This is crucial when handling sensitive car data, ensuring that the application is secure by default.
- **Template System** – Django's templating engine allows for dynamic HTML generation, making it easy to create web pages that display car data, analysis results, or visualizations. This system also supports template inheritance, which helps in maintaining a consistent look and feel across the application.
- **REST Framework** – Django REST framework is a powerful and flexible toolkit for building web APIs. In a car data analysis program, this can be used to create APIs that allow external applications or users to access car data and analysis tools programmatically.

Other Considerations:

- **Learning Curve** – Django's comprehensive feature set and conventions can be challenging for beginners to learn. The framework's steep learning curve may require additional time and effort to fully master, especially if you are new to web development or complex frameworks.
- **Overhead for Small Projects** – Django's "batteries-included" approach can sometimes introduce unnecessary overhead for smaller projects. If the car data analysis program is relatively simple, a lighter framework like Flask might be more appropriate.
- **Complexity of Customization** – While Django is highly customizable, making significant changes to its default behaviors can be complex. Developers may need to dive deep into Django's internal workings to achieve certain custom functionalities, which can be time-consuming.

- **Performance Considerations** – Although Django is scalable, for extremely high-performance needs, especially in data-heavy applications, careful optimization is required. This might involve using caching strategies, optimizing database queries, or offloading certain tasks to background processes[26].

2.3.14 Data Management

Data management is a critical component in this project that involves collecting, storing, and analyzing data. In the context of a project focused on web scraping and data analysis of car details, efficient data management involves arranging the flow of data from its initial acquisition via web scraping to its final analysis. The objective is to ensure data accuracy, accessibility, and reliability throughout the whole project, which includes addressing challenges such as data consistency, storage efficiency, and query performance.

The choice of a data management system impacts how data is structured, manipulated, and accessed. The primary categories of database systems used in data management are *Relational Database Management Systems* (RDBMS) and NoSQL databases.

RDBMS vs. NoSQL:

- **RDBMS** are traditional database systems that use a structured query language (SQL) for defining and manipulating data. They are built on a model that organizes data into one or more tables of rows and columns with a predefined schema.

- **NoSQL** databases, on the other hand, are more flexible in terms of data structure, supporting not only traditional table-based data but also more complex structures like documents, graphs, and key-value pairs.

Given the structured nature of car data and the importance of relational integrity (such as linking specific car models to their attributes and history), RDBMS is deemed more suitable for this project. This review will thus focus on RDBMS, specifically exploring SQL Server and MySQL, to understand their roles in supporting the data management needs of the project[27].

2.3.15 Data Management - SQL Server

SQL Server is a relational database management system (RDBMS) developed by Microsoft. It is widely used for data storage and data management. SQL Server is known for its high performance, scalability, and security features which make it suitable for both small and large enterprise applications.

Key features:

- **Transaction Management**: SQL Server provides robust transaction control that ensures data integrity and consistency with features such as *Atomicity, consistency, isolation and durability* (ACID).
- **Performance and Scalability**: SQL Server supports high-performance in-memory transactions and has powerful indexing capabilities, which are essential for querying large datasets of car details efficiently[28].

2.3.16 Data Management - MySQL

MySQL, by contrast, is an open-source RDBMS that is widely used for web applications, particularly those running on *Linux, Apache, MySQL, PHP/Python/Perl* (LAMP) stacks. It is renowned for its simplicity, reliability, and strong performance.

- **Cost-Effectiveness**: Being open-source, MySQL reduces costs, making it ideal for projects with budget constraints.
- **Flexibility and Ease of Use**: MySQL is known for its ease of setup and use, and a wide range of database administration tools and community resources are available[29].

2.4 Summary

This chapter reviews the current research and applications of web scraping and data analysis, focusing on car data. It discusses existing platforms like car marketplaces and price prediction tools that rely on web scraping to provide insights into automotive trends. Commonly used tools for web scraping and data analysis, such as Python libraries (Requests, BeautifulSoup, and Pandas), are explored for their role in automating data collection and processing. The chapter also covers the technologies used to manage and analyze large datasets, emphasizing relational databases like SQL Server and MySQL. Overall, it provides a comprehensive understanding of the methodologies driving innovations in automotive data solutions.

Chapter 3

Developed Work

3.1 Web Scraping

Web scraping is the process of automatically extracting data from websites. It has become an essential tool for gathering large datasets from the web, particularly in cases where data is not readily available through public APIs. In the context of this project, web scraping was employed to gather detailed information on used car listings from online marketplaces. This data includes key attributes such as price, brand, model, year, mileage, and color, which are essential for analyzing trends and patterns in the used car market, as well as other car details such as version, engine displacement, gearbox type, number of gears and doors.

Web scraping is a process used to automatically extract large amounts of data from websites, allowing for efficient and structured data collection from online sources. In recent years, it has become a widely used tool in various fields such as e-commerce, market research, and data science, where access to real-time information is critical. This project leverages web scraping to gather detailed car listings from a Portuguese online marketplace, automating the collection of information such as car prices, models, years, and other attributes that would otherwise be time-consuming to gather

manually.

The primary purpose of using web scraping in this project is to create a comprehensive dataset that can be analyzed to understand trends in car pricing. By automating this data extraction, the project ensures the data is up-to-date and accurately reflects the available Portuguese car market.

In this context, web scraping plays a crucial role in bridging the gap between publicly available data and the structured format required for further analysis. This process feeds the subsequent stages of the project, which involve database management, data cleaning, data analysis and its visualization, forming the foundation of the entire workflow.

3.1.1 Data Source

Standvirtual was chosen as the primary data source for this project due to its reputation as the leading online marketplace for cars and motorcycles in Portugal. Unlike other websites that offer a mix of vehicle parts, accessories, and various unrelated products, Standvirtual focuses exclusively on vehicles. This specialization makes it an ideal source for high-quality, targeted data about car listings, which is critical for conducting a focused analysis of car prices and trends.

The platform's popularity in Portugal ensures that the data collected reflects a comprehensive and up-to-date view of the local car market, providing an accurate basis for the study.

Additionally, Standvirtual's wide range of car listings, including both new and used vehicles from official dealers and independent sellers ensures that the data retrieved covers different segments of the market, making it possible to analyze various factors such as depreciation, brand popularity, and market demand across different vehicle categories.

The data retrieved from Standvirtual focuses on essential car attributes that are critical for price analysis and trend identification. These attributes include:

- **Price** – The listed price of the vehicle, which forms the basis of the price analysis.
- **Brand** – The manufacturer of the car, which allows for brand-specific trend analysis.

- **Model** – The specific model of the vehicle, enabling comparisons between different models.
- **Version** – The version of the car, which is a user input, meaning it is not validated by Standvirtual.
- **Year** – The year of manufacture, which is crucial for analyzing the impact of vehicle age on price.
- **Kilometers** – The total distance the car has traveled, a key factor in determining the vehicle's value and condition.
- **Fuel Type** – Information on whether the car runs on petrol, diesel, electric, or hybrid.
- **Color** – The color of the car, included to investigate potential color preferences.
- **Displacement** – The engine size of the vehicle, which can influence both the car's performance and price.
- **Horsepower** – The engine power of the car, which affects its desirability and pricing.
- **Gearbox Type** – Whether the vehicle has a manual or automatic transmission.
- **Number of Gears** – The number of gears in the gearbox.
- **Number of Doors** – The vehicle's door configuration.

One aspect to mention is that Standvirtual itself offers also a price analysis tool that informs the user if the car he's currently viewing is in a good price bracket, as shown in figure 3.1.

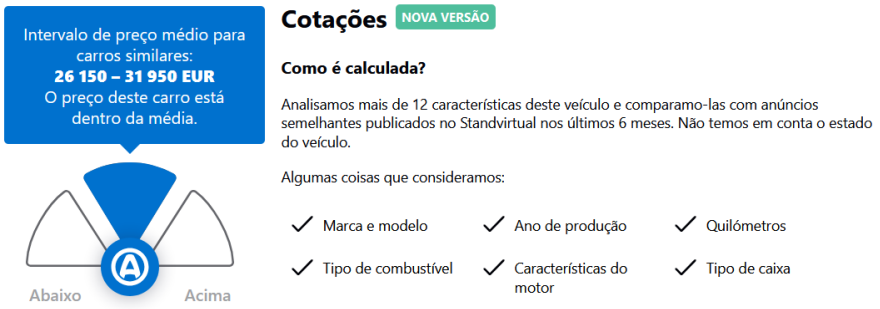


Figure 3.1: Example of Stadvirtual's Price Analysis Tool [30]

3.1.2 Legal Considerations

In the context of this project, where data is scraped from the Standvirtual website, it is essential to consider and comply with the platform's *Terms of Service* (ToS), particularly regarding the permissible use of their data. The Standvirtual ToS outline specific guidelines and restrictions on data usage, ensuring that their intellectual property, user privacy, and commercial interests are respected. Below are the key legal considerations derived from their terms:

1. ***Data Usage Consent*** The scraping and usage of data from the Standvirtual website requires explicit consent from Standvirtual. Any material available on the website cannot be utilized without permission, and using such data in a manner that violates applicable laws or harms the interests of Standvirtual or its advertisers is strictly prohibited.

2. ***Prohibition on Data Scraping and Redistribution*** According to the ToS, the data and information available on Standvirtual must not be aggregated or processed for dissemination to third parties, whether online or offline, without authorization. This means that while the data can be collected for personal use or for analysis within this project, any sharing of this data, especially in aggregated forms would require explicit permission from Standvirtual.This applies to sharing datasets, graphs, or any aggregated reports generated from the scraped data.

3. ***Limitations on Commercial Use*** - Using the data for any commercial purposes is expressly forbidden unless authorized by Standvirtual. Since this

project focuses on analysis and academic research, and not for direct commercial gain, it complies with this rule.

4. ***Personal Data Usage*** - Standvirtual prohibits the use of personal data belonging to its users or advertisers without their consent. Therefore, the project must ensure that no personal data is collected or used.

5. ***No Unreasonable Interference*** - Any scraping processes must not interfere unreasonably with Standvirtual's legitimate interests or the functioning of their website.

3.1.3 Tools and Technologies Used

The programming language selected for this project was Python due to its versatility and the extensive ecosystem of libraries that cater specifically to web scraping, data manipulation, and database operations. Python's syntax is easy to learn and allows for rapid development, making it an ideal choice for a project that involves multiple stages of data collection, processing, and analysis.

Additionally, Python's strong community support ensures that any challenges encountered during development can be resolved with readily available resources and documentation,these can be found in websites such as Stack Overflow, a community base Q/A site where users can asks questions to others on programming matters.

Web Scraping Libraries

- **Requests**:
  - The Requests library is a fundamental tool for making HTTP requests in Python. It is used to send requests to the target web pages hosting car listings. This library abstracts the complexity of handling HTTP connections and responses, offering a simple API to interact with web servers.
  - Key features of the requests library utilized in this project include setting custom headers to mimic a real user's browsing behavior and handling different HTTP response statuses to ensure robust error handling and retries if necessary.

- **BeautifulSoup**:
  - BeautifulSoup is used to parse the HTML content returned by the requests library. This library provides an interface to navigate through the HTML document, allowing for the extraction of specific elements based on their tags, classes, or other attributes.
  - In this project, BeautifulSoup is used to locate and extract critical data points such as car prices, brands, models, and other relevant specifications. The flexibility of BeautifulSoup in handling complex and nested HTML structures is crucial for accurately retrieving the required information from potentially messy or inconsistent web page layouts.

Database Management

- **SQL Server**:
  - SQL Server was chosen for its robust performance in handling large volumes of data, ensuring the scalability of the project as the dataset grows. SQL Server provides strong transactional support, ensuring that the insertion and updating of records are handled efficiently and securely.
  - The database is designed to store the scraped car data in a structured format, with tables that reflect the various attributes of the car listings such as price, brand, model, year. SQL Server's powerful querying capabilities allow for efficient retrieval and manipulation of the data, which is essential for subsequent analysis.
- **PyODBC**:
  - The pyodbc library is a Python module that enables the connection between Python scripts and SQL Server databases using ODBC drivers. It allows for the execution of SQL commands directly from Python, facilitating the insertion, update, and retrieval of data within the SQL Server database.

  - In this project, pyodbc is used to insert scraped data into the database and update records as new information becomes available. It also enables the retrieval of data for export to Excel or for further analysis within Python.

Excel Integration

- **Pandas**:
  - Pandas is a powerful data manipulation library in Python that excels in handling structured data. It is used in this project to load data from the SQL Server database into a DataFrame, a tabular data structure that allows for easy manipulation and analysis of data.
  - Once the data is in a DataFrame, it can be cleaned, filtered, and transformed as needed before being exported to Excel. Pandas provides a seamless integration with Excel, making it easy to write the DataFrame directly into an Excel file.
- **xlwings**:
  - xlwings is a Python library that bridges the gap between Python and Excel. It allows Python scripts to interact with Excel workbooks, facilitating tasks such as writing data to specific sheets, formatting cells, and automating Excel-based workflows.
  - In this project, xlwings is used to export the cleaned and processed data from the Pandas DataFrame to an Excel workbook. It also ensures that the data is inserted into a specific sheet within the workbook, and it can delete and replace sheets as needed to maintain a clean and updated dataset. Additionally, xlwings can automate Excel operations such as saving the workbook and closing it after the data export is complete.

Cloud Storage

Dropbox API:

- The Dropbox API is used to upload the latest Excel file to a cloud storage account, ensuring that the data is backed up and accessible from anywhere. This is particularly useful for sharing the data with team members or accessing it from different devices.
- The integration with Dropbox involves automating the upload process within the Python script. The script first checks if an older version of the file exists in the Dropbox folder and deletes it if necessary. It then uploads the new file, ensuring that the cloud storage always contains the most recent version of the data.
- Security considerations are addressed by using an access token for Dropbox, ensuring that the upload process is secure and only authorized scripts can interact with the Dropbox account.

3.1.4 Scraping Process

Considering the research elaborated and the key decisions made regarding the technologies and applications to be used there is the need to properly define the objectives and outputs of the web scraping process. In this subsection there will be an overview of the web scraping process, therefore it is important to define what outputs will be needed at the end of it. It was implied in the previous subsection that SQL Server and Dropbox were the technologies selected for Data and File storage, therefore the outputs are as follows:

1. ***CarDetails Table*** - This table, belonging to the 'tedi' Database, is where the car data extracted from Standvirtual will be stored.
2. ***cars_detailed.xlsm*** - The output file will contain the data extracted from the CarDetails table and will be used for offline data analysis. Since the analysis involves the use of macros, the file will be saved in the 'Excel Macro-Enabled Workbook' format to support these features effectively.

The two distinct outputs are crucial to consider. Due to the complexity of the overall web scraping process, we will use two separate flowcharts to explain its operation in detail. The first flowchart, figure 3.2, will illustrate the process of

connecting to the database, followed by the initial data extraction and insertion into the database table. The second flowchart, figure 3.3, will detail the steps involved in updating the inserted data and generating the Excel file.

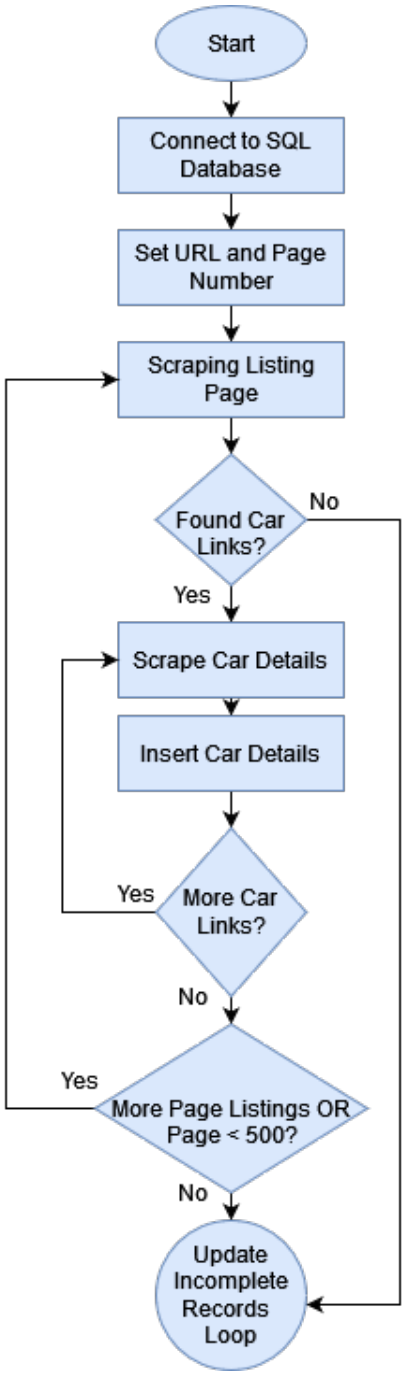


Figure 3.2: Data Insertion Flowchart

This flowchart outlines the steps involved in the data insertion process, starting with establishing a connection to the SQL database. It proceeds through the key

stages of fetching car listings, scraping detailed information for each vehicle, and inserting that data into the database. As the initial part of the web scraping workflow, this flowchart ensures a structured approach to gathering and storing car details for further processing. The individual steps are as follows:

1. The process starts by establishing a connection with the SQL database where the car details will be stored. This is essential to ensure that all extracted data has a designated repository for insertion.

2. The initial URL for scraping car listings is set, along with the page number, starting with the first page. The scraper will move through these pages one at a time.

3. The script scrapes the listing page to extract individual car links that will be processed further. This step focuses on gathering the URLs of car detail pages.

4. A decision point that checks if car links were successfully extracted from the listing page. If no car links are found, it moves directly to the next part of the process which is the "Update Incomplete Records Loop". If links are found, the scraper will continue extracting details for each car.

5. For each car link obtained from the listing page, the script scrapes detailed information such as the car's price, brand, model, year, and other relevant details.

6. Once the car's details are scraped, they are inserted into the SQL database under the appropriate fields, capturing the complete dataset for each car..

7. Another decision point to check whether there are more car links to process on the current listing page. If more car links are present, the process repeats from scraping the next car's details. If no more car links remain, the process moves on.

8. This decision checks if there are additional pages of car listings or if the scraper has reached a limit of 500 pages. If more pages are available, the script continues by fetching the next page, incrementing the page number, and repeating the process from scraping the listing page. If there are no more pages or the page limit has been reached, the process transitions to the update loop.

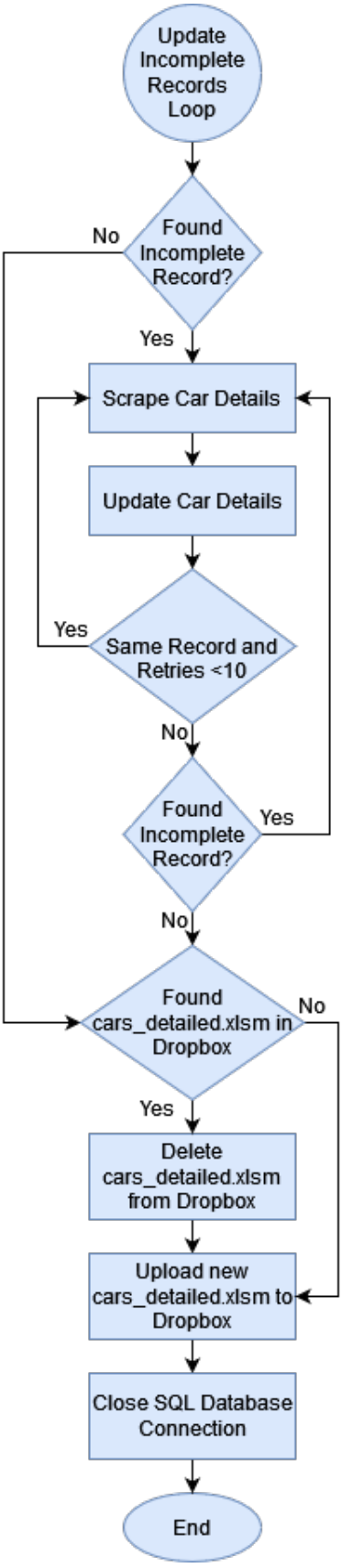


Figure 3.3: Data Insertion Flowchart

This flowchart details the steps involved in updating incomplete records and creating the Excel file. It begins with identifying records that require additional data, followed by scraping and updating those records in the database. After ensuring that all entries are complete, the process moves to creating an Excel file with the updated data. The file is then uploaded to Dropbox, and any outdated versions are removed. The flow concludes with closing the database connection, completing the web scraping process. The individual steps are as follows:

1. This process starts by checking if there is an incomplete record in the CarDetails table, if it finds one it proceeds to the next step. If no incomplete records are found, the loop is exited.

2. If an incomplete record is found, the web scraping process is used to retrieve the missing details from the car listing.

3. The newly scraped data is used to update the incomplete record in the database.

4. A check is performed to see if the same record has been retried fewer than 10 times. If more than 10 retries have been attempted without success, the record is removed from the database.

5. The process repeats for any other incomplete record. If no more records are found, the loop ends.

6. After exiting the loop, the flow checks whether the cars_detailed.xlsm file already exists in Dropbox.

7. If the file is found, the old version is deleted to prevent duplication.

8. The updated version of the Excel file is created from the database and uploaded to Dropbox.

9. Finally, the connection to the SQL database is closed, marking the end of the web scraping process.

After this process is done, the CarDetails table will have all the data updated, which will then be used to perform data analysis and be presented on the server side of the application, where it will be possible to download the excel file created/updated in this process.

3.1.5 CarDetails Table Creation

The CarDetails table is the central component of our database, designed to store comprehensive information about cars scraped from the web. This table is structured to hold the attributes of each car that are crucial for subsequent analysis and reporting. The following image, figure 3.4, shows the table design in SQL Server Management Studio.

Column Name	Data Type	Allow Nulls
ID	int	☐
Price	nvarchar(50)	☑
Brand	nvarchar(100)	☑
Model	nvarchar(100)	☑
Version	nvarchar(100)	☑
FuelType	nvarchar(50)	☑
Year	nvarchar(10)	☑
Kilometres	nvarchar(50)	☑
Displacement	nvarchar(50)	☑
Horsepower	nvarchar(50)	☑
Color	nvarchar(50)	☑
GearboxType	nvarchar(50)	☑
NumGears	nvarchar(10)	☑
NumDoors	nvarchar(10)	☑
PageURL	nvarchar(255)	☐
DateInserted	date	☐

Figure 3.4: CarDetails Table Design

3.1.6 Data Extraction Logic

In this subsection, there is an explanation for the data extraction process, which is accomplished through web scraping techniques using the requests library to fetch the webpage content and BeautifulSoup to parse the HTML and extract relevant data points. CSS selectors are primarily used to identify and locate specific HTML elements, while conditional logic is applied to handle missing or dynamic data. Below, is the break down of the extraction logic for key data points such as price, brand, model, and other car details.

Fetching Web Page Content

First, the HTML content of each car listing is retrieved using the requests.get() method, with appropriate headers to mimic a real browser and avoid being blocked by the website.

```
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, 'html.parser')
```

Listing 3.1: Fetching Data

Here, soup is an object that represents the parsed HTML, which will be navigated using CSS selectors and element search methods.

Locating Data Points Using CSS Selectors

Once the HTML is parsed, the next step is to locate specific elements that contain the desired data using their class names or tag structure.

- **Price**: The price of the car is located in an <h3> tag with a class of "offer-price_number emsdndu4 ooa-1s0hzs7 er34gjf0". The script uses soup.find() to extract this element.

```
price_element = soup.find('h3', class_='offer-price_number
    emsdndu4 ooa-1s0hzs7 er34gjf0')
if price_element:
car_details["price"] = price_element.text.strip()
```

Listing 3.2: 'Price' Extraction

- **Brand, Model and others**: These details are typically found within a series of tags with specific class names. Each contains a <p> element that describes the type of data followed by an <a> tag that holds the actual value, for example "Marca" for brand, In some cases, data points may be missing, either because they are not provided on the page or due to variations in the HTML structure. The script accounts for this by checking if the element exists before attempting to extract its value. If an element is missing, a default empty string (") or None is assigned to the respective field.

  The script loops through these tags, checking the text of <p> elements to determine what information they represent, and ensures that the script doesn't fail due to missing data and assigns a default value where appropriate. For example:

```
info_divs = soup.find_all('div', class_='ooa-162vy3d eyfqfx03')

for div in info_divs:
p_tag = div.find('p')
a_tag = div.find('a')

p_text = p_tag.text.strip() if p_tag else None
a_text = a_tag.text.strip() if a_tag else None

if p_text == "Marca":
car_details["brand"] = a_text if a_text else ''
elif p_text == "Modelo":
car_details["model"] = a_text if a_text else ''
```

Listing 3.3: Example of an extraction for brand and model

Extrating Speficic Data Points

For the Version, Year, Kilometres, Displacement, Horsepower and Number of Doors data points, a different technique is used to extract the data, since these are nested within sibling <p> tags that appear after the main description. To extract, the script uses the find_next_sibling() method to traverse the HTML tree and locate the relevant data, the following example demonstrate how the Year data point is extracted:

- **Year**: The year is stored in the next sibling <p> tag of the "Ano" description.

```
if p_text == "Ano":
year_tag = p_tag.find_next_sibling('p', class_='e6ymdfe0 ooa-1
    pe3502 er34gjf0')
car_details["year"] = year_tag.text.strip() if year_tag else ''
```

Listing 3.4: 'Year' Extraction

Storing the Extracted Data

After all relevant data points are extracted, they are stored in an array (car_details) which is then passed to the database insertion function. The array contains all attributes wanted.

```
car_details = {
  "page_url": url,
  "price": None,
  "brand": '',
  "model": '',
  "version": '',
  "fuel_type": '',
  "year": '',
  "kilometres": '',
  "displacement": '',
  "horsepower": '',
  "color": '',
  "gearbox_type": '',
  "num_gears": '',
  "num_doors": '',
  "date_inserted": datetime.now().strftime('%Y-%m-%d')
}
```

Listing 3.5: Storing Data

3.1.7 Data Insertion Logic

Has mentioned in section 4.1.4.2 SQL Server was selected to be used for database management. Using a relational database like SQL Server ensures data integrity, allows for efficient data retrieval, and supports complex analytical operations. This section explains how the web scraping data is stored into a SQL Server database using Python.

Database Connection

The process begins by establishing a connection to the database. This is achieved using Python's pyodbc library, which facilitates communication between Python applications and SQL databases. Here's how the connection is set up:

```
# Database connection parameters
server = 'example of a server'
database = 'example of a database'
username = 'example of a username'
password = 'example of a password'

# Create a connection string
conn_str = (
r'DRIVER={ODBC Driver 17 for SQL Server};'
f'SERVER={server};'
f'DATABASE={database};'
f'UID={username};'
f'PWD={password};'
)

# Establishing the connection to the SQL Server
conn = pyodbc.connect(conn_str)
cursor = conn.cursor()
```

Listing 3.6: Database Connection

Data Insertion

Once connected, data is prepared and inserted into the database. The data scraped from the web is organized into an array as described in 4.1.6.4 and inserted into the database using an SQL INSERT statement, the following listing shows the function responsible for inserting the data into the CarDetails table:

```
def insert_Data(car_data):
# SQL query for inserting data
insert_query = """
INSERT INTO CarDetails (
Price, Brand, Model, Version, FuelType, Year, Kilometres,
Displacement, Horsepower, Color, GearboxType, NumGears,
NumDoors, PageURL, DateInserted
)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
"""
# Preparing the data tuple from the car data dictionary
car_data_tuple = (
car_data["price"],
car_data["brand"],
car_data["model"],
car_data["version"],
car_data["fuel_type"],
car_data["year"],
car_data["kilometres"],
car_data["displacement"],
car_data["horsepower"],
car_data["color"],
car_data["gearbox_type"],
car_data["num_gears"],
car_data["num_doors"],
car_data["page_url"],
car_data["date_inserted"]
)
# Executing the insert query
cursor.execute(insert_query, car_data_tuple)
conn.commit()  # Committing the transaction
```

Listing 3.7: Data Insertion

Closing the Database Connection

After all operations are completed, it's important to properly close the database connection to free up system resources:

```
# Closing the cursor and the connection
cursor.close()
conn.close()
```

Listing 3.8: Closing Connection to the Database

Properly closing the connection avoids potential memory leaks and ensures that the database operations are cleanly terminated.

3.1.8 Updating Incomplete Records

After the data is inserted, there needs to be a verification that the data inserted is correct and no unexpected records are left in the database, these unwanted records are caused by network issues or due to the rate limiting of the Requests library. In this section an explanation will be provided for each element of this validation.

Finding Incomplete Records

The validation starts by querying the database to find the first record with missing fields. The SQL query looks for any record where essential fields such as brand, model, price, etc., are either null or empty. The following image, figure 3.5, is an example of incomplete records stored in the database.

	ID	Price	Brand	Model	Version	FuelType	Year	Kilometres	Displacement	Horsepower	Color
6203	6203	22 800	DS	DS3 Crossb...	E-Tense Grand Chic	Eléctrico	2020	74 023 km		136 cv	Preto
6204	6204	16 500	Opel	Corsa	1.2 Business	Gasolina	2023	24 738 km	1 199 cm3	75 cv	Branco
6205	6205	26 200	Volvo	XC 40	2.0 D3 Momentum Geartronic	Diesel	2018	233 000 km	1 969 cm3	150 cv	Preto
6206	6206	23 990	BMW	418 Gran C...	d Advantage Auto	Diesel	2020	154 062 km	1 995 cm3	150 cv	Preto
6207	6207	14 990	Ford	Focus SW	1.5 TDCi EcoBlue Business	Diesel	2020	195 162 km	1 499 cm3	120 cv	Preto
6208	6208	NULL									
6209	6209	72 500	Mercedes-Benz	E 300	De AMG Line	Híbrido Plug-In	2024	8 524 km	1 950 cm3	306 cv	Cinzento
6210	6210	9 999	Peugeot	208	1.4 HDi	Diesel	2012	188 350 km	1 398 cm3	68 cv	Preto
6211	6211	13 950	Fiat	Tipo	1.4 Lounge	Gasolina	2019	58 442 km	1 368 cm3	95 cv	Cinzento
6212	6212	52 900	Mercedes-Benz	E 220	d AMG Line	Diesel	2020	44 415 km	1 950 cm3	194 cv	Cinzento
6213	6213	18 900	Nissan	Qashqai	1.3 DIG-T N-Connecta	Gasolina	2018	63 291 km	1 332 cm3	140 cv	Cinzento
6214	6214	12 900	Dacia	Sandero	1.5 dCi Stepway	Diesel	2018	99 335 km	1 461 cm3	90 cv	Cinzento
6215	6215	99 900	Mercedes-Benz	E 53 AMG	4Matic+	Gasolina	2022	9 000 km	2 999 cm3	435 cv	Preto
6216	6216	NULL									
6217	6217	NULL									
6218	6218	15 500	BMW	123	d Auto	Diesel	2012	243 512 km	1 995 cm3	204 cv	Branco
6219	6219	NULL									
6220	6220	NULL									
6221	6221	NULL									
6222	6222	17 950	Hyundai	i30 SW	1.0 T-GDi Style+Navi	Gasolina	2021	36 000 km	998 cm3	120 cv	Cinzento
6223	6223	23 990	Ford	Puma		Gasolina	2024	13 000 km	999 cm3	125 cv	Azul
6224	6224	NULL									
6225	6225	NULL									
6226	6226	12 990	Mercedes-Benz	C 200	CDi Classic BlueEfficiency	Diesel	2011	229 676 km	2 143 cm3	136 cv	Cinzento
6227	6227	NULL									
6228	6228	NULL									

Figure 3.5: Example of Incomplete Records

The function find_first_empty_field_record retrieves such records:

```
def find_first_empty_field_record(cursor, current_date):
query = """
SELECT TOP 1 ID, PageURL
FROM CarDetails
WHERE DateInserted = ?
AND (
(
(Brand IS NULL OR Brand = '')
AND (Model IS NULL OR Model = '')
AND (Version IS NULL OR Version = '')
AND (FuelType IS NULL OR FuelType = '')
AND (Year IS NULL OR Year = '')
AND (Kilometres IS NULL OR Kilometres = '')
AND (Displacement IS NULL OR Displacement = '')
AND (Horsepower IS NULL OR Horsepower = '')
AND (Color IS NULL OR Color = '')
AND (GearboxType IS NULL OR GearboxType = '')
AND (NumGears IS NULL OR NumGears = '')
AND (NumDoors IS NULL OR NumDoors = '')
)
OR (Price IS NULL OR Price = '')
);
"""
cursor.execute(query, current_date)
return cursor.fetchone()
```

Listing 3.9: 'find_first_empty_field_record' function

This function returns the first record where data is incomplete. If no such record is found, the validation ends.

Updating Records

Once an incomplete record is identified, its details are scraped again using the URL saved in the record. The scrape_car_details function is used to fetch the latest data, this is the same function used in the data insertion logic.

This function scrapes the webpage again and extracts all the car details, filling

in the gaps missed in the initial scrape, if it is unable to update the record after retrying for a maximum of 10 times, it will delete this record and try to find the next.

```
if retry_counter[record_id] >= 10:
print(f"Deleting record ID {record_id} after 10 failed tries.")
delete_record(cursor, record_id)
cursor.commit()
del retry_counter[record_id]
```

Listing 3.10: Delete Record Condition

The following image, figure 3.6, shows an example of complete and validated records.

Results Messages

	ID	Price	Brand	Model	Version	FuelType	Year	Kilometres	Displacement	Horsepower	Color
13949	13954	23 500	MG	ZS	1.5 VTi-Tech Luxury	Gasolina	2024	19 568 km	1 499 cm3	106 cv	Vermel...
13950	13955	24 990	Peugeot	508	1.5 BlueHDi Allure EAT8	Diesel	2019	99 400 km	1 499 cm3	130 cv	Preto
13951	13956	15 500	BMW	320	d Touring Navigation	Diesel	2013	256 345 km	1 995 cm3	184 cv	Branco
13952	13957	9 990	Citroën	C1 Airscape	1.2 VTi Shine	Gasolina	2016	65 500 km	1 199 cm3	82 cv	Branco
13953	13958	27 499	BMW	320 Gran Turismo	d xDrive Sport-Aut. M Sport	Diesel	2017	108 000 km	1 995 cm3	190 cv	Azul
13954	13959	30 900	Citroën	Jumpy	1.5 BlueHDi XL	Diesel	2022	94 000 km	1 499 cm3	120 cv	Branco
13955	13960	46 990	Hyundai	Ioniq 6	77kWh Premium	Eléctrico	2023	18 774 km		228 cv	Cinzento
13956	13961	46 000	Lexus	ux-250h	Sport	Híbrido (Gas...	2023	17 155 km	1 987 cm3	184 cv	Cinzento
13957	13962	17 490	Peugeot	308	1.5 BlueHDi Style	Diesel	2020	52 411 km	1 499 cm3	102 cv	Branco
13958	13963	46 000	Lexus	UX 250h	Sport	Híbrido (Gas...	2023	10 250 km	1 987 cm3	184 cv	Preto
13959	13964	52 900	Porsche	Panamera	4 GTS PDK	Gasolina	2012	148 000 km	4 806 cm3	430 cv	Pratea...
13960	13965	22 990	Nissan	Juke	1.0 DIG-T N-Connecta	Gasolina	2023	18 988 km	999 cm3	114 cv	Preto
13961	13966	14 990	SEAT	Ibiza	1.6 TDi Style	Diesel	2019	81 634 km	1 598 cm3	95 cv	Cinzento

Figure 3.6: Example of Complete Records

3.1.9 Excel File Creation

After updating the incomplete records, an excel file is created with the data from the CarDetails table and uploaded to Dropbox. In this section this process will be explained in more detail, explaining with actual code the main elements of this process.

Excel Workbook Creation

After retrieving the data, the next step involves creating an Excel workbook where this data will be stored. This is achieved using the xlwings library, which provides extensive capabilities to interact with Excel files from within Python. xlwings is particularly used here due to its ability to execute complex operations on Excel workbooks, such as deleting and creating sheets.

The script initiates a new Excel workbook from the cars_detailed.xlsm present in the project file, and starts by removing the 'Car Details' sheet and creating a new one, this is because it needs a clean sheet to copy the entire data within the database table. It then populates this worksheet with the data from the DataFrame:

```
excel_file = 'cars_detailed.xlsm'
wb = xw.Book(excel_file)
if 'car_details' in [sheet.name for sheet in wb.sheets]:
wb.sheets['car_details'].delete()
sheet = wb.sheets.add('car_details')
sheet.range('A1').options(index=False, header=True).value = df
sheet.visible = False
wb.save(excel_file)
wb.close()
```

Listing 3.11: Creation of Excel File

Uploading to Dropbox

Once the Excel file is fully prepared, it is uploaded to Dropbox using the Dropbox API. This step is crucial for data sharing and accessibility, it will allow users with the required permissions to download this file directly from the website. The following script ensures that the file on Dropbox is the most current version, replacing any older versions of the file:

```
ACCESS_TOKEN = 'example of an Access Token'

dbx = dropbox.Dropbox(ACCESS_TOKEN)
delete_files(dbx, DROPBOX_FOLDER, 'cars_detailed.xlsm')
upload_file(dbx, LOCAL_FILE_PATH, f"{DROPBOX_FOLDER}/
    cars_detailed.xlsm")
```

Listing 3.12: Upload Excel File to Dropbox script

3.2 Data Analysis

3.2.1 Introduction to Data Analysis

The data analysis for this project focuses on understanding various car attributes, such as price, mileage, year of manufacture, and color, to gain insights into market trends and price influencers. The analysis is performed using two main approaches: Python-based automated analysis through a web-based server application and an offline Excel-based option.

The Python-based approach is an automated system where users can select specific car brands and models, and the system will dynamically generate updated charts showing relationships like price vs. kilometers, price vs. year, and price vs. color. This system is designed for online interaction, providing real-time analysis and visualizations that users can access through a server-side application.

To complement the web-based approach, an offline Excel option is available for users who prefer not to be continuously connected to the website. Users can download an Excel file containing the same car data and analysis tools, allowing them to interact with the data offline. The Excel workbook is equipped with dynamic features such as pivot tables and automatically updating charts based on user selections, similar to the functionality offered in the web-based system.

Both methods offer users the flexibility to analyze car market trends either online or offline, depending on their needs. In the following sections, we will explore the technical details of both approaches, how the data is extracted, and how the visualizations are generated.

3.2.2 Python Script Data Analysis

The Python script approach to data analysis provides an automated, real-time solution for processing and visualizing the car data collected through web scraping. This system allows users to interact with the data by selecting specific car brands and models, after which the system dynamically generates visualizations to reveal trends and insights related to pricing and other attributes. Below is a detailed breakdown of the process.

3.2.3 Data Extraction and Preprocessing in Python

The first step involves extracting the car data from an SQL Server database. The SQL query retrieves all relevant car details, such as price, mileage, year, color, and brand, which are then loaded into a pandas DataFrame for further processing:

```
query = """
SELECT *
FROM dbo.CarDetails
WHERE
Price IS NOT NULL AND Price != '' AND
Brand IS NOT NULL AND Brand != '' AND
Model IS NOT NULL AND Model != '' AND
Year IS NOT NULL AND Year != '' AND
Kilometres IS NOT NULL AND Kilometres != '' AND
Color IS NOT NULL AND Color != '';
"""

df = pd.read_sql(query, engine)
```

Listing 3.13: Database query for all relevant data

Once the data is extracted, preprocessing is required to ensure that numeric values like price, mileage, and year are correctly formatted. The script removes unnecessary characters, such as currency symbols and spaces, and converts the cleaned data into numerical formats for analysis. Missing values are handled by filling them with appropriate defaults:

```
df['Price'] = df['Price'].str.replace(' ', '').str.replace('€',
    '').astype(float)
df['Kilometres'] = df['Kilometres'].str.replace(' ', '').str.
    replace('km', '')
df['Kilometres'] = pd.to_numeric(df['Kilometres'], errors='
    coerce')
df['Kilometres'].fillna(0, inplace=True)
df['Year'] = pd.to_numeric(df['Year'], errors='coerce')
df['Year'].fillna(0, inplace=True)
```

Listing 3.14: Preprocessing of data collected

3.2.4 Data Analysis and Visualizations in Python

Linear Regression Analysis

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. In this case, it is applied to understand how factors such as mileage (kilometers) and year of manufacture influence car prices. The goal is to find the best-fit line that represents the relationship between the independent variable (kilometres) and the dependent variable (price).

In the context of this project:

- **Price vs. Kilometers**: Linear regression is used to model how the price of a car tends to decrease as the number of kilometers increases, reflecting the depreciation over time.
- **Price vs. Year**: This analysis helps to understand how the year of manufacture impacts the price of a car, with newer cars typically retaining higher value.

The method calculates a line that minimizes the differences between the predicted values (on the line) and the actual observed values. This line is represented by the equation:

$$Y = a + bX$$

Where:

- ***Y*** is the dependent variable (price);
- ***X*** is the independent variable (kilometers or year);
- ***a*** is the intercept (the starting value of the price when X=0X=0);
- ***b*** is the slope of the line (indicating how much YY changes with a unit change in XX).

By fitting a linear regression model, the system generates a scatter plot with actual data points from the web scraping process and a regression line that represents

the predicted relationship. This makes it easy to visualize how much the price changes based on the car's mileage or age.

These relationships are visualized using scatter plots with regression lines, offering a clear view of how each variable affects price. The regression is performed using the LinearRegression model from sklearn library:

```
1  def plot_linear_regression(ax, x, y, x_label, y_label):
2      reg = LinearRegression().fit(x.values.reshape(-1, 1), y.values
           .reshape(-1, 1))
3      ax.scatter(x, y, color='blue', alpha=0.5)
4      ax.plot(x, reg.predict(x), color='red', linewidth=2)
5      ax.set_xlabel(x_label)
6      ax.set_ylabel(y_label)
```

Listing 3.15: Plotting Linear Regression Graph

Bar Chart Analysis

A bar chart is generated to show the average price of cars by color. This allows the user to compare how car color impacts the market value of cars:

```
1  color_price = model_df.groupby('Color')['Price'].mean()
2  plot_bar_graph(ax, color_price, 'Color', 'Average Price (€)')
```

Listing 3.16: Plotting Bar Graph

Returning the Analysis in Images

The analysis results are returned in the form of images that can be easily integrated into the web-based server application for display. This is accomplished by converting the plots into base64-encoded images, which allows the images to be embedded in HTML or transmitted to the client application:

```
buf = io.BytesIO()
plt.savefig(buf, format='png')
buf.seek(0)
img_base64 = base64.b64encode(buf.getvalue()).decode('utf-8')
```

Listing 3.17: Converting Plots Into base64-encoded Images

Multiple visualizations, including price vs. kilometers, price vs. year, and price by color, are generated and returned as a list of base64-encoded images. This allows the server-side application to dynamically present the results based on user input.

Below, figure 3.7 and figure 3.8, are examples of a linear regression graph and a bar graph, respectively, generated by this process:

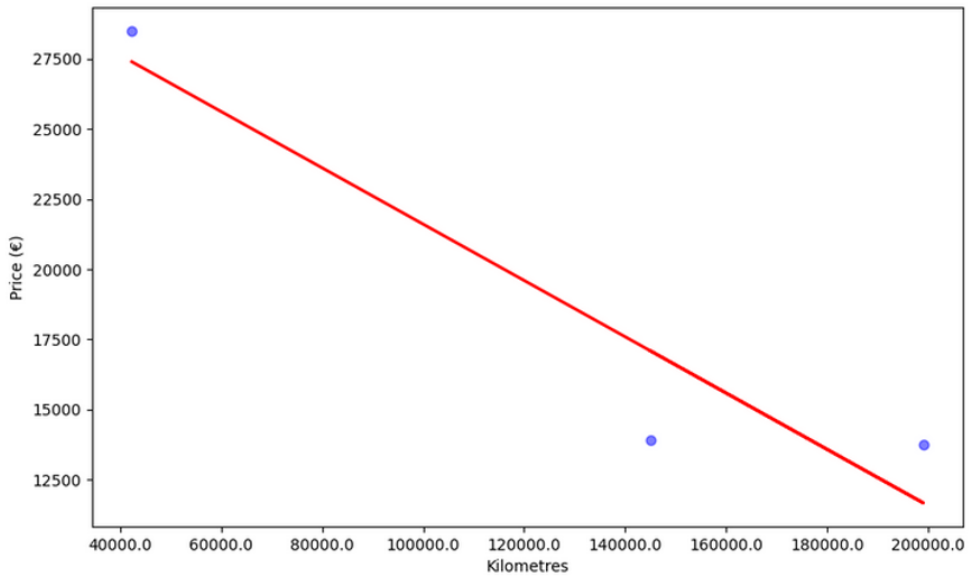


Figure 3.7: Example of a Linear Regression Graph

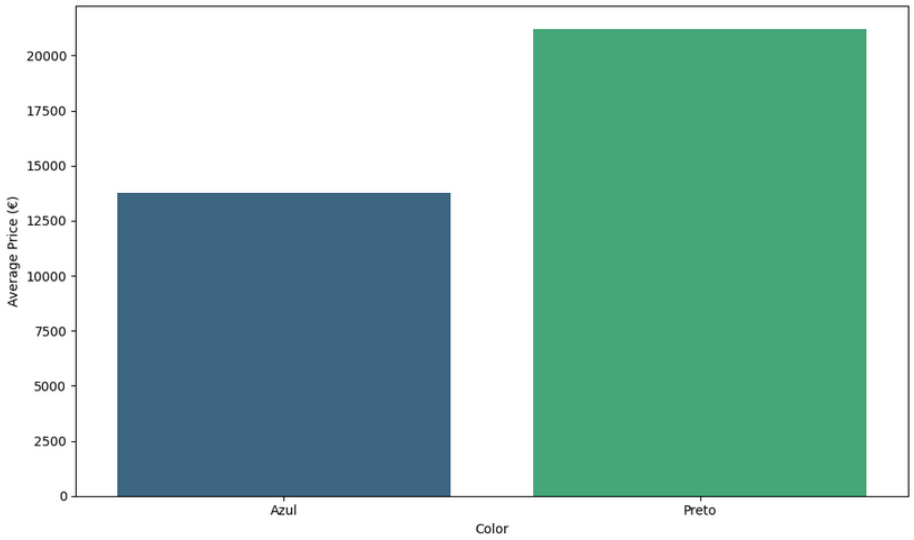


Figure 3.8: Example of a Bar Graph

3.2.5 Excel Data Analysis

In addition to the Python-based automated analysis, this project offers users the option to perform data analysis offline using Excel. This feature is particularly useful for users who do not wish to remain continuously connected to the server-side application. The Excel file contains all the necessary data and tools for users to dynamically explore relationships between car attributes such as price, mileage, year, and color. The Excel-based approach provides flexibility for manual exploration while maintaining the functionality of automatic chart updates based on user input.

3.2.6 Excel Data Analysis Techniques

The Excel-based data analysis in this project is designed to provide a user-friendly, offline alternative to the Python-based approach. The goal is to offer users the flexibility to perform dynamic data analysis without requiring a continuous connection to the web application. To enhance this experience, *Visual Basic for Applications* (VBA) functions were developed to automate the process of preparing data for visualization, ensuring that the charts update automatically based on user selections.

3.2.7 VBA Functions to Automate Data Preparation

In order to streamline the process of data extraction and filtering, VBA (Visual Basic for Applications) functions were created. These functions play a crucial role in listing and retrieving the relevant data points needed to dynamically generate the graphs. Rather than manually filtering the data, users can make selections within the "Car Analysis" worksheet, and the VBA functions automatically perform the necessary operations.

The primary functions developed include:

- **ListMatchingValues**: This function retrieves data based on user-selected criteria (car brand and model) from the dataset stored in the "car_details" worksheet. It searches for matching entries and stores them in the "Lists" worksheet, which serves as the data source for the graphs.
- **AnalyseCar**: This function performs a more detailed search, gathering additional information such as price, kilometers, year, and color based on the user's selection. The retrieved data points are then listed in the "Lists" worksheet, ready for visualization.

These VBA functions ensure that the necessary data is automatically extracted, cleaned, and organized, simplifying the user experience by eliminating manual data handling.

3.2.8 Dynamic Charts

With the data prepared by the VBA functions, Excel's dynamic charting capabilities are employed. When a user selects a specific car brand or model and then press the "Analyse" button, the charts in the "Car Analysis" worksheet update automatically to reflect the new data. Key charts include:

- **Price vs. Kilometers**: This chart displays the relationship between car price and mileage, illustrating how prices generally decrease as kilometers increase.
- **Price vs. Year**: This chart visualizes the correlation between a car's year of manufacture and its price, showing how newer cars tend to retain more value.

- **Price by Color**: A bar chart that compares the average price of cars by their color, revealing trends in how color can influence car value.

These dynamic charts provide users with real-time visual insights into the dataset. Thanks to the VBA automation, the charts refresh seamlessly based on the user's inputs, making it easy to explore and analyze different aspects of the car data.

3.2.9 Retrieving the Latest Excel File Link

To facilitate offline analysis, the project includes functionality for users to download the latest version of the Excel file. The latest Excel file is stored in Dropbox, and the Python script retrieves the download link for this file so that it can be made available to the user.

The script uses the Dropbox API to list the files in a specified folder, sort them by the last modification date, and generate a temporary download link for the most recently updated Excel file. This link is then returned to the server-side application, allowing the user to download the file with the most up-to-date data.

```
1  def get_latest_file_link():
2      ACCESS_TOKEN = 'your_access_token'
3      dbx = dropbox.Dropbox(ACCESS_TOKEN)
4      files = dbx.files_list_folder('/Dropbox/tedi').entries
5      files = sorted(files, key=lambda x: x.server_modified, reverse
           =True)
6      latest_file = files[0].path_lower
7      temp_link = dbx.files_get_temporary_link(latest_file).link
8      return temp_link
```

Listing 3.18: Function that Returns the Latest Excel File Link

- **Access Token**: The function starts by authenticating with Dropbox using the provided access token.
- **File Retrieval**: It fetches all files within the specified Dropbox folder (/Dropbox/tedi) and sorts them by their server_modified timestamp, ensuring that the most recently modified file is selected.

- **Temporary Link Generation**: After identifying the latest file, the function calls Dropbox's API to generate a temporary download link (temp_link) that is valid for a limited period. This link is then returned to the Flask route for redirection.

This feature ensures that users always have access to the most current dataset, even when performing offline analysis. Once downloaded, the Excel file enables users to interact with the car data using Excel's built-in analysis tools without the need for a continuous internet connection.

3.3 Server Application

The server application for this project is built using the Flask framework and serves as the backbone of the web-based interface for analyzing car data. The application manages user authentication, processes requests for data analysis, and renders the results in a visually accessible format. Additionally, the server provides functionality for users to download the latest Excel file for offline analysis.

The primary goals of the server application are:

- **User Authentication**: Implement a secure system for user login, signup, and session management.
- **Data Analysis**: Allow users to perform analysis on car data based on their selected criteria .
- **Dynamic Visualization**: Generate visual representations of car price trends and return the results as charts.
- **Excel File Retrieval**: Enable users with the correct permissions to download the latest car data as an Excel file for offline use.

The Flask framework is used to manage routing and form submission, while SQLAlchemy provides the necessary tools for interacting with the SQL Server database that stores user information and car details. Additionally, Flask-Login is utilized to ensure only authenticated users can access certain parts of the application, such as data analysis and file downloads.

3.3.1 User Authentication and Authorization

The user authentication system is a critical part of the server application, ensuring that only authorized users can access the analysis features and download the latest Excel files. Flask-Login, along with SQLAlchemy, is used to implement user authentication, session management, and access control.

User table creation

The User table is an important component of the application's database, responsible for storing user credentials and roles. This table supports the authentication process, allowing the application to manage user access based on their login information and assigned role.

The table structure consists of four main columns:

- **Id**: An integer serving as the primary key, uniquely identifying each user.
- **username**: A varchar(50) field storing the username, ensuring that each user has a unique identifier for login.
- **password**: An nvarchar(255) field that holds the hashed password, ensuring security.
- **role**: A varchar(20) field that defines the user's role, which is used to control access to different parts of the application.

This structure ensures that the system can securely store and manage user data, while also providing flexibility in managing different user roles.

The following image, figure 3.9, shows a screenshot from the SQL Server Management Studio of the User table design.

Column Name	Data Type	Allow Nulls
id	int	☐
username	varchar(50)	☐
password	nvarchar(255)	☐
role	varchar(20)	☐

Figure 3.9: User Table Design

User Model and Authentication

The User model represents the users of the application and is stored in the User table mentioned previously. It includes fields such as username, password, and role. The role field differentiates between different types of users (regular users and paid users, where paid users have access to additional features like downloading Excel files).

To secure user credentials, passwords are hashed using the generate_password_hash function from Werkzeug. This ensures that the stored passwords are not plain text, making the system more secure. Password verification during login is done using check_password_hash, ensuring that only valid credentials allow access to the system.

```
1  class User(db.Model, UserMixin):
2      __tablename__ = 'Users'
3      id = db.Column(db.Integer, primary_key=True)
4      username = db.Column(db.String(50), unique=True, nullable=
           False)
5      password = db.Column(db.String(255), nullable=False)
6      role = db.Column(db.String(20), nullable=False)
7
8      def set_password(self, password):
9      self.password = generate_password_hash(password)
10
11     def check_password(self, password):
12     return check_password_hash(self.password, password)
```

Listing 3.19: Password Encryption

Login and Signup Functionality

The application provides routes for both user login (/login) and signup (/signup). When a user submits their credentials during login, the application checks the database for a matching username and then verifies the password using check_password. Upon successful authentication, the user is logged in using the login_user() function, which initiates a session for that user.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
  if request.method == 'POST':
  username = request.form['username']
  password = request.form['password']
  user = User.query.filter_by(username=username).first()
  if user and user.check_password(password):
  login_user(user)
  return redirect(url_for('car_analysis_view'))
  else:
  flash('Invalid username or password.', 'danger')
  return render_template('login.html')
```

Listing 3.20: Login Function

The following image, figure 3.10, shows the login screen with all the users inputs.

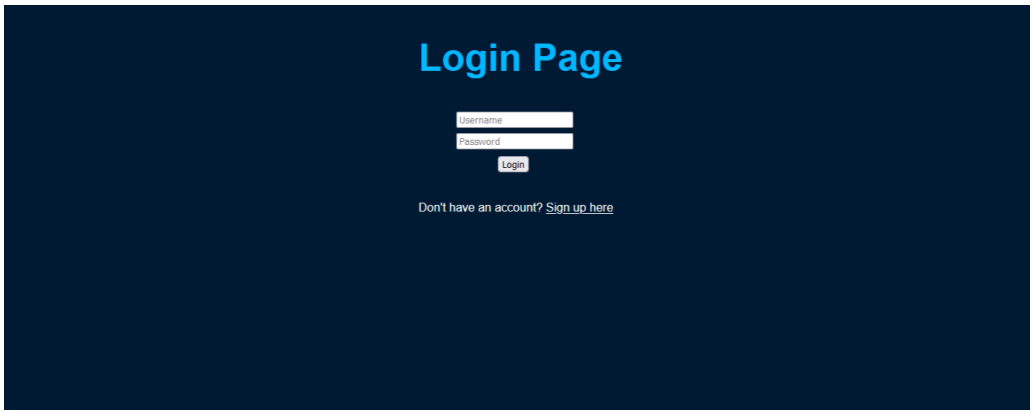


Figure 3.10: Login Screen

The signup route allows new users to create an account. If the username is unique, the application stores the user's hashed password and other details in the database.

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
  if request.method == 'POST':
  username = request.form['username']
  password = request.form['password']
  role = request.form['role']
  if User.query.filter_by(username=username).first():
  flash('Username already exists.', 'danger')
  else:
  new_user = User(username=username, role=role)
  new_user.set_password(password)
  db.session.add(new_user)
  db.session.commit()
  flash('Account created successfully. Please log in.', 'success
      ')
  return redirect(url_for('login'))
  return render_template('signup.html')
```

Listing 3.21: Sign-up Function

The following image, figure 3.11, shows the signup screen with all the users inputs.

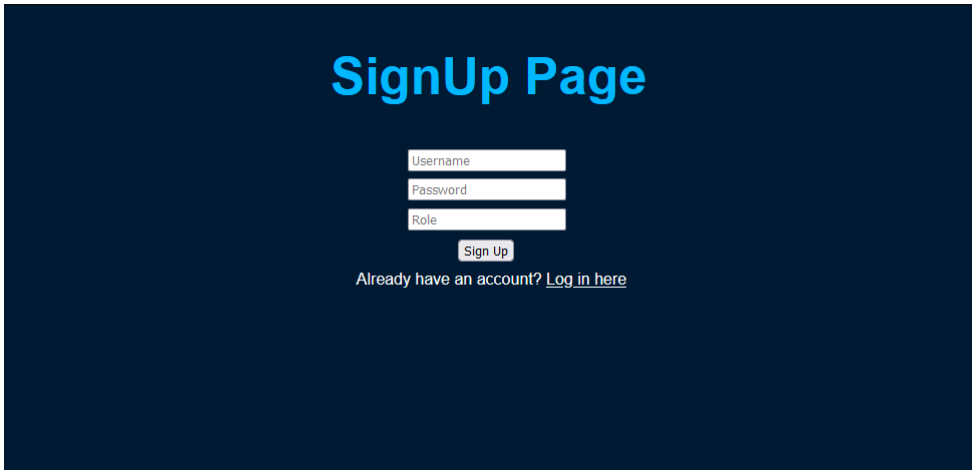


Figure 3.11: Sign-Up Screen

Access Control

The application uses role-based access control to restrict access to certain features, such as downloading the latest Excel file. This control is implemented on both the server side and the frontend.

In the car-analysis route, the server retrieves the current user's role using the current_user.role attribute. This role is then passed to the frontend template through the render_template function. In the template (analysis.html), access to certain features, such as the "Download Latest Excel File" button, is restricted based on the user's role.

For example, the following code snippet shows how the button is only displayed to users with the role of "paid":

```
1    {% if user_role == 'paid' %}
2      <a href="{{ url_for('download_latest_file') }}" class="
           download-button" role="button">Download Latest Excel File
           </a>
3    {% endif %}
```

Listing 3.22: HTML User Role Validation

By using Jinja's template syntax, the button is conditionally rendered only for users who have the appropriate role. This ensures that only paid users can access the feature to download the Excel file, enforcing role-based access control at the frontend level.

On the server side, access to the /car-analysis route is also protected by the @login_required decorator, ensuring that only authenticated users can access the page.

Logout Functionality

The application also includes a logout feature, which allows users to securely end their session. Upon logging out, the user is redirected to the login page, and a success message is displayed using Flask's flash() function.

```
@app.route('/logout')
@login_required
def logout():
  logout_user()
  flash('You have been logged out.', 'success')
  return redirect(url_for('login'))
```

Listing 3.23: Logout Function

3.3.2 Data Analysis Endpoint

The Data Analysis Endpoint is a key component of the web application that allows users to select specific car brands and models, perform detailed analysis on the selected data, and visualize the results through dynamic charts. The selection of car details triggers the backend to perform analysis and generate graphs based on the available data. The results are then displayed in real-time on the front-end.

Car Analysis Route

The /car-analysis route is responsible for handling the user's input for selecting a car brand and model, processing the data, and rendering the visual analysis. It handles both GET and POST requests:

- **GET**: When the user first visits the page, they are presented with a dropdown list of available car brands. At this stage, no analysis is performed.
- **POST**: Once the user selects a brand and model and submits the form, the backend processes the request and returns the analysis results in the form of graphs.

Dynamic Brand and Model Selection

The dynamic selection feature is an important part of the data analysis process. When the user selects a car brand, the available models for that brand are automatically updated in the model dropdown, and vice versa. This ensures that the user can only select models that exist for the chosen brand.

- **Brand Selection**: When the user selects a brand, the available models are filtered based on the selected brand and are ordered alphabetically.
- **Model Selection**: Once the user selects a model, further filtering or analysis can be performed based on that specific model.

This dynamic filtering is achieved using the /get-options route, which updates the available models based on the selected brand:

```
@app.route('/get-options', methods=['GET'])
def get_options():
  brand = request.args.get('brand')
  model = request.args.get('model')

  df = get_car_details()  # Fetch the dataset

  # Filter DataFrame based on selections
  if brand:
  df = df[df['Brand'] == brand]
  if model:
  df = df[df['Model'] == model]

  # Get unique options for each field
  brands = sorted(df['Brand'].unique().tolist())
  models = sorted(df['Model'].unique().tolist())

  return jsonify({
    'brands': brands,
    'models': models,
  })
```

Listing 3.24: 'get_options' function

The dropdown menus on the frontend are updated dynamically using *Asynchronous Javascript And XML* (AJAX)AJAX, which ensures that when the user selects a brand, the model options adjust accordingly. The following HTML snippet illustrates the dropdowns for brand and model:

```
<form method="POST" action="{{ url_for('car_analysis_view') }}">
  <select name="brand" id="brand" required>
    <option value="" disabled selected>Select Brand</option>
    {% for brand in brands %}
      <option value="{{ brand }}">{{ brand }}</option>
    {% endfor %}
  </select>

  <select name="model" id="model" required>
    <option value="" disabled selected>Select Model</option>
    {% for model in models %}
      <option value="{{ model }}">{{ model }}</option>
    {% endfor %}
  </select>

  <button type="submit">Analyse</button>
</form>
```

Listing 3.25: HTML Update Brand and Model Inputs

This ensures a smooth user experience where the selection of a brand triggers the model dropdown to be updated with valid options.

Data Analysis Function

The core analysis logic happens in the car_analysis.analyze_car_prices() function. This function processes the car data for the selected brand and model and generates graphs for price trends and other metrics.

The generated visualizations include:

- **Price vs. Kilometres**: A scatter plot that shows how the price varies with the car's mileage.
- **Price vs. Year**: A scatter plot that displays the relationship between the car's production year and its price.
- **Price by Color**: A bar chart showing the average price for cars based on their color.

Returning Analysis as Images

The graphs generated by the analysis function are returned to the frontend as base64-encoded images. This format allows the images to be easily embedded in HTML pages and viewed directly by the user without the need to download them.

In the HTML template, the encoded images are displayed using the following Jinja2 code:

```
{% if plot_data %}
  <div class="graph-container">
    <div>
      <div class="graph-title">Price vs. Kilometres</div>
      <img src="{{ plot_data[0] }}" alt="
          Price vs. Kilometres">
    </div>
    <div>
      <div class="graph-title">Price vs. Year</div>
      <img src="{{ plot_data[1] }}" alt="
          Price vs. Year">
    </div>
    <div>
      <div class="graph-title">Price vs. Color</div>
      <img src="{{ plot_data[2] }}" alt="
          Price vs. Color">
    </div>
  </div>
{% endif %}
```

Listing 3.26: HTML script to display the generated graph images

This ensures that the generated visualizations are displayed in real-time as the user makes selections.

3.3.3 Excel File Download Feature

The application provides a feature allowing users to download the latest Excel file containing the car data, enabling them to perform offline analysis. This feature is restricted to users with the appropriate role and is implemented through the

/download-latest-file route. The route dynamically generates a link to the most recent Excel file stored in Dropbox, which can be accessed by users who meet the required conditions.

Download Link Generation

The Excel file download feature is implemented in the /download-latest-file route. When this route is accessed, it retrieves the most recent Excel file link from Dropbox and redirects the user to that link. The function get_latest_file_link() is responsible for fetching the URL of the most recently uploaded file.

Here is how the download route is implemented:

```
1  @app.route('/download-latest-file')
2  @login_required
3  def download_latest_file():
4      file_link = get_latest_file_link()
5
6      return redirect(file_link)
```

Listing 3.27: Function Responsible for Fetching the Latest Excel File Link

The get_latest_file_link() function is called to retrieve the most recent Excel file from Dropbox. This function, which was explained in more detail in a previous section, interacts with Dropbox's API to get the URL of the latest file, ensuring that users always receive the most up-to-date data.

3.4 Summary

This chapter discusses the implementation of a web scraping system designed to collect car data from a online marketplace Standvirtual. The process involves extracting car details from car listings using existing Python libraries. The collected data is stored in a SQL database, and incomplete records are periodically updated or removed. An Excel file is generated for offline analysis, including dynamic charts and automated updates through VBA scripts. Additionally, a Flask-based web application allows users to perform data analysis online, featuring authentication and access control for certain functionalities.

Chapter 4

Results and Further Improvements

In this chapter, we will discuss the results obtained from the web scraping and data analysis project and identify potential areas for improvement. The project aimed to develop a system capable of extracting car data from an online marketplace, analyzing the information, and presenting it in a user-friendly way through interactive graphs and downloadable Excel files. While the system successfully meets these objectives, this section will also propose enhancements to improve performance, user experience, and functionality.

In Section 4.1 covers the results, focusing on the data collected from web scraping and the insights generated through data analysis, both in the server application and the Excel file. This section will highlight the achievements, such as the amount of data scraped, the successful implementation of data visualization, and the Excel analysis feature.

Section 4.2 outlines potential improvements to optimize the system. This includes enhancements to the web scraping process, such as scraping from multiple

sources, and improvements to the data analysis features, including advanced machine learning models, predictive price analysis, additional filters, and expanding the capabilities of the Excel file.

The discussion will first address the system's achievements and challenges encountered during implementation and use, followed by a detailed exploration of future improvements to enhance performance and expand the system's capabilities.

4.1 Results Discussion

4.1.1 Overview of the Application's Performance

The application successfully fulfills the initial project objectives, delivering a system that allows users to interact with and analyze car data dynamically. In addition to meeting the core requirements, the application provides extended functionality, such as the ability to download the car data in an Excel format for offline analysis. This extra feature enhances flexibility for users who prefer to perform data analysis outside of the web environment.

Key user-facing components, such as the interactive data analysis and selection of car brands and models, function as intended. Users can select a brand and model, and the application promptly responds by updating the options and generating visual results. These graphs provide clear insights into price trends, including relationships between price, kilometers, year, and color.

The overall user experience has been very positive, primarily due to the intuitive design and fast response times of the app. The interface is easy to navigate, and the analysis is presented in real-time, ensuring that users can swiftly access the information they need. This ensures a seamless and efficient user interaction throughout the analysis process.

4.1.2 Web Scraping Results

The web scraping process was designed to gather a comprehensive dataset of car details from an online marketplace called StandVirtual. This dataset serves as the foundation for the data analysis and visualizations presented in the project. The primary goal of the scraping process was to collect key attributes for thousands of

car listings, including brand, model, year, price, color, and mileage. The data was successfully extracted and stored in a structured format, enabling detailed analysis and insights.

The web scraping script captured a significant amount of data, retrieving nearly 15,000 car listings so far. These listings include a wide range of car attributes, the most important ones are Brand, Model, Year, Price, Kilometres, Color and Page URL.

The number of results collected so far can continue to grow with additional web scraping, ensuring that the dataset remains current and representative of the market. The image below, Figure 4.1, provides a snapshot of the latest car details captured, showcasing the last 20 car listings gathered through the scraping process.

Below is an example of the data collected through the web scraping process. The sample shows key fields such as Brand, Model, Price, Year, Kilometres, and more:

	ID	Price	Brand	Model	Year	Kilometres	Horsepower	Color	PageURL	DateInserted
14017	14435	41 900	Hyundai	Tucson	2024	10 km	230 cv	Cinzento	https://www.standvirtual.com/carros/anuncio/hyund...	2024-09-17
14018	14436	45 800	Mercedes-Benz	Classe E	2017	34 000 km	194 cv	Cinzento	https://www.standvirtual.com/carros/anuncio/merce...	2024-09-17
14019	14437	18 690	Renault	Clio	2022	34 079 km	100 cv	Branco	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14020	14438	14 750	Citroën	C3	2017	34 000 km	110 cv	Outra	https://www.standvirtual.com/carros/anuncio/citron-...	2024-09-17
14021	14439	12 490	Citroën	C3	2018	57 000 km	83 cv	Outra	https://www.standvirtual.com/carros/anuncio/citron-...	2024-09-17
14022	14440	4 499	Chevrolet	Aveo	2009	175 963 km	72 cv	Cinzento	https://www.standvirtual.com/carros/anuncio/chevr...	2024-09-17
14023	14441	71 900	Land Rover	Range Rover	2018	89 500 km	404 cv	Preto	https://www.standvirtual.com/carros/anuncio/land-r...	2024-09-17
14024	14442	16 900	Peugeot	308 SW	2020	142 000 km	130 cv	Azul	https://www.standvirtual.com/carros/anuncio/peug...	2024-09-17
14025	14443	23 900	BMW	Série 3	2018	107 000 km	252 cv	Pratea...	https://www.standvirtual.com/carros/anuncio/bmw-...	2024-09-17
14026	14444	10 500	BMW	Série 1	2012	250 000 km	116 cv	Preto	https://www.standvirtual.com/carros/anuncio/bmw-...	2024-09-17
14027	14445	21 290	Renault	Mégane Sport Tourer	2021	61 376 km	140 cv	Preto	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14028	14446	4 900	Fiat	Punto	1994	143 000 km	136 cv	Vermel...	https://www.standvirtual.com/carros/anuncio/fiat-pu...	2024-09-17
14029	14447	20 990	Renault	Mégane Sport Tourer	2020	63 523 km	140 cv	Preto	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14030	14448	19 990	Renault	Captur	2022	49 406 km	90 cv	Azul	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14031	14449	20 490	Renault	Captur	2022	43 359 km	90 cv	Cinzento	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14032	14450	19 790	Renault	Clio	2023	22 216 km	100 cv	Cinzento	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14033	14451	19 790	Renault	Clio	2023	27 832 km	100 cv	Cinzento	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14034	14452	20 490	Renault	Captur	2022	44 762 km	90 cv	Cinzento	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14035	14453	20 490	Renault	Captur	2022	41 922 km	90 cv	Preto	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17
14036	14454	18 690	Renault	Clio	2023	25 909 km	90 cv	Azul	https://www.standvirtual.com/carros/anuncio/renaul...	2024-09-17

Figure 4.1: Results of web scraping

This table represents the last entries added to the dataset, highlighting the data structure and key attributes captured for each car.

The volume and quality of the data collected during the web scraping process have had a direct impact on the depth and accuracy of the analysis. With nearly 15,000 listings, the dataset provides a robust foundation for comparing trends such as price vs. mileage, price vs. year, and price vs. color. This large dataset ensures that the insights derived from the analysis are both meaningful and representative of the car market.

4.1.3 Data Analysis Results

The application successfully generates visualizations that provide users with insights into car data based on their selected criteria. Users are able to choose a car brand and model, and the system produces various graphs to illustrate key relationships between the selected car attributes. Examples of the generated graphs include:

- **Price vs. Kilometres**: A scatter plot showing the relationship between a car's mileage and its price.
- **Price vs. Year**: A line graph illustrating how car prices change based on the year of manufacture.
- **Price by Color**: A bar chart that compares the average price of cars based on their color.

These graphs are displayed directly in the web application, providing users with real-time feedback based on their selections. The user-friendly interface ensures that these visualizations are clear and easy to interpret, enhancing the overall experience. Figure 4.2 below shows examples of the three main graphs generated by the application.

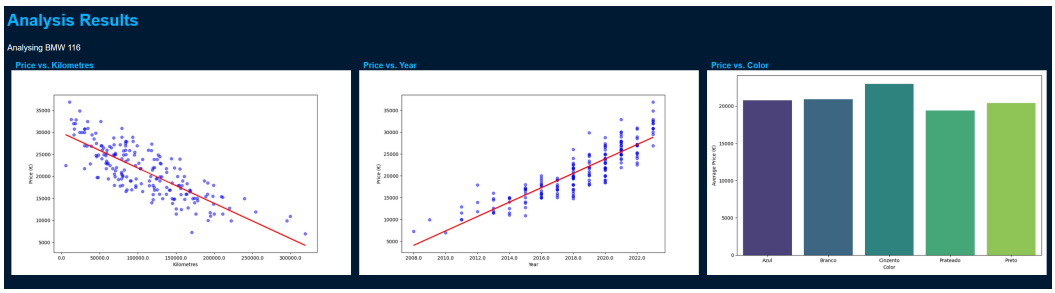


Figure 4.2: Example of car analysis

In addition to the in-app analysis, the system also allows users to download the latest car data in an Excel file. This feature is especially useful for users who prefer to conduct offline analysis or need to access the data without being connected to the web application. The downloadable Excel file includes all relevant car details and enables users to explore the data further in a familiar environment like Excel, where they can apply their own custom analysis or use the provided VBA macros

for generating graphs. The following image, figure 4.3, shows an example of car analysis using the excel file.

Figure 4.3: Example of car analysis performed on the excel file

4.1.4 Challenges Encountered

Web Scraping Challenges

While the web scraping process was successful in collecting a large amount of data, a few key challenges were encountered along the way:

- **Rate Limiting**: One of the primary issues faced during the scraping process was the rate limiting imposed by the target website. When the scraper made too many requests in a short period of time, the website would return blank or incomplete records. This resulted in records with missing or incorrect information, such as missing prices or mileage.

  To resolve this issue, the application implemented an update loop. After the initial scrape, the system would check for any records that had blank or missing fields and attempt to update them with the correct data. If the update failed after 10 retries, the system would delete the incomplete record to maintain the integrity of the dataset. This solution ensured that blank records were minimized, and only high-quality data was kept.

- **Updating the Excel File**: Another challenge involved ensuring that the Excel file, which users can download for offline analysis, was always up to date. Each time new data was scraped, the Excel file needed to be updated to reflect the latest car details. This involved not only inserting the new data

into the file but also updating the relevant sheets (such as the "car_details" sheet) while maintaining a clean and organized structure. Automating this process required careful handling of sheet deletion, insertion, and formatting to ensure that the file was ready for user analysis.

Data Analysis Challenges

During the development of the data analysis feature, several challenges arose related to both the server-side analysis and the Excel-based analysis:

- **Sending Graphs to the Frontend**: A significant challenge in the data analysis phase was generating and sending the relevant visualizations (graphs) to the frontend for the user to view. The application needed to generate visualizations such as price vs. kilometers, price vs. year, and price vs. color dynamically, based on the user's input (selected car brand and model).

  After generating these graphs using Python's Matplotlib and Seaborn libraries, the next step was to send them to the web interface. This involved converting the graphs into base64-encoded images, which could be rendered directly in the user's browser.

- **Development of Excel Macros for Offline Analysis**: For users who preferred to work offline, a challenge was to create a dynamic, easy-to-use Excel file that required little to no Excel knowledge. The goal was to ensure that users could easily analyze the data without needing to manually filter or manipulate the dataset.

  To achieve this, custom VBA macros were developed to automate the data filtering and chart creation process. These macros automatically update the Excel charts based on the user's selection of car brand and model, making the analysis dynamic and user-friendly. In addition, auxiliary sheets were created to help manage and structure the data behind the scenes, ensuring that the charts were always correctly generated. These supporting sheets store and process the lists of car brands, models, prices, and other attributes, allowing the macros to easily update the charts without requiring the user to perform manual tasks.

The automation provided by these macros and auxiliary sheets made it possible for users to simply select a brand and model, and the relevant graphs would update instantly, making the process intuitive even for users with minimal Excel knowledge. The challenge was to ensure that this process worked seamlessly and dynamically, regardless of the dataset's size.

4.2 Possible Improvements

4.2.1 Enhancing Data Scraping

While the project achieved its primary objectives, several areas could be improved to enhance the robustness, accuracy, and scale of the system. Below are some possible improvements for the web scraping process and data analysis that could be explored in future iterations of the project.

Scraping Data from Multiple Marketplaces

Currently, the system scrapes data from a single car marketplace. To provide a more comprehensive dataset, future improvements could involve expanding the web scraping to include additional car marketplaces. By collecting data from multiple sources, the analysis would have broader market coverage, making the insights more accurate and representative of the entire car market.

This improvement would involve identifying other car marketplaces that provide similar types of data and integrating additional scrapers for those sites into the existing system. Each marketplace may have different HTML structures, so scrapers would need to be customized for each one. Additionally, merging the datasets from different sources into a unified database would require standardization of fields such as price, mileage, and car specifications to ensure compatibility for analysis.

Using Large Language Models (LLMs) for Sorting Car Versions

One significant challenge in the current system is handling car version data. Since the version is typically input by users when they create listings, there is often a lack of consistency and validation. As a result, the same car model with the same specifications can be labeled with slightly different version names, making it difficult to group them correctly during analysis.

To address this, *Large Language Model* (LLM) such as GPT-4 could be employed to automatically sort and normalize car versions. LLMs excel at understanding the context and similarities between textual data, and they could be trained to identify different version names that actually refer to the same car configuration. For example, "1.5 BlueHDI GT Line" and "1.5 BlueHDI GT" could be automatically grouped as the same version.

By using LLMs, the system would gain the ability to standardize version data, ensuring that similar cars are grouped correctly and reducing the inconsistency in the dataset. This would improve the accuracy of the analysis, particularly when examining version-specific trends in car prices and features.

4.2.2 Improving Data Analysis

Although the current data analysis features of the application provide valuable insights into car prices and other relevant metrics, there are several areas where improvements can be made to enhance the functionality and usability of the system. This chapter explores possible improvements that could elevate the accuracy and depth of the analysis, offering a more comprehensive tool for users. Below are four key improvements that can be implemented in future versions of the application.

Incorporating More Advanced Machine Learning Models

At present, the data analysis module relies on basic linear regression to model the relationship between variables such as car price, mileage, and year. While linear regression provides a simple and efficient method for understanding linear relationships, car market data is often complex and involves non-linear patterns that may not be captured by linear regression alone.

By incorporating more advanced machine learning models, such as Random Forest, XGBoost, or Neural Networks, the application can improve its ability to model complex relationships between variables. For example, Random Forest and XGBoost are ensemble learning methods that can capture non-linear interactions and offer higher predictive accuracy. These models could analyze features such as car brand, model, fuel type, mileage, and year in combination, yielding more sophisticated insights about how these factors influence price.

Advanced machine learning models would allow the system to better handle large datasets and multiple input variables. For instance, non-linear patterns between mileage and price, such as the depreciation rate slowing down after a certain mileage threshold, could be more accurately captured by these models, providing users with more nuanced insights into the factors affecting car prices.

Support for More Filters and Criteria

The current system allows users to filter car data by brand and model, which provides a good starting point for analysis. However, the addition of more filtering options would significantly improve the flexibility and depth of analysis available to users. Future iterations of the application could introduce filters for additional criteria, such as:

- **Price range**: Allowing users to set a minimum and maximum price to narrow down their results.
- **Year range**: Enabling users to filter by the age of the car, providing a more detailed analysis of older versus newer models.
- **Fuel type**: Providing options to filter by gasoline, diesel, electric, or hybrid cars.

- **Transmission type**: Offering filters for manual and automatic transmissions.
- **Engine Displacement**: Offering filters for car engine displacement.

Expanding the number of available filters would give users the ability to perform more detailed and targeted analysis.

4.3 Summary

This chapter reviewed the project results and outlined areas for potential improvement. The web scraping successfully collected nearly 15,000 car listings, forming a solid base for analysis. The system's data analysis features, both online and through Excel, worked as expected, offering valuable insights into car prices based on various attributes.

Challenges such as rate limiting during web scraping and rendering dynamic graphs were addressed through solutions like the update loop and automated macros for Excel.

Looking ahead, incorporating advanced machine learning models and offering more filtering options would enhance the analysis. Overall, the application meets its goals, with room for further enhancements to expand its capabilities.

Chapter 5

Conclusions

This chapter consolidates the primary conclusions drawn from the project's development, as outlined below.

This project was driven by the significant changes in how consumers engage with the car market. With the increasing dominance of digital platforms for researching and purchasing cars, the demand for automated applications capable of extracting, analyzing, and presenting car-related data has grown. Buyers and sellers need timely, detailed insights into market trends, price fluctuations, and car specifications. This project aimed to meet these needs by developing a system that could efficiently scrape car data from online platforms, analyze it, and present actionable insights to users through a web-based application and offline tools.

The objectives of this project were clearly defined from the beginning . First, the project sought to develop a web scraping solution capable of gathering large datasets from a car marketplace, specifically focusing on essential car attributes like brand, model, year, price, and mileage. Second, it aimed to analyze this data to provide meaningful insights, such as price comparisons based on various car characteristics. Third, the project involved creating a web-based application to display the results of these analyses in a user-friendly format. Finally, the development of an offline tool,

allowing users to perform similar analyses without requiring an internet connection, using a downloadable Excel file with embedded data visualization tools.

The literature review focused on existing technologies and methodologies relevant to the project's objectives. It explored the use of web scraping in the context of car data, detailing the techniques employed by current car marketplaces and price prediction tools to gather and analyze data. The review highlighted the significance of using Python libraries such as BeautifulSoup for web scraping, Pandas for data manipulation, and Matplotlib for visualizing data trends. Furthermore, it examined server-side technologies like Flask, which was essential in developing the web-based application, and the role of data management systems such as SQL for handling large datasets efficiently. The literature review provided the foundational knowledge necessary for the development of the system, ensuring that the selected tools and methodologies were well-suited to the project's needs.

The development phase of the project involved several key steps. First, the web scraping solution was implemented to gather data from the car marketplace StandVirtual, which resulted in the collection of nearly 15,000 unique car listings. This dataset included vital information such as car prices, model years, mileage, and other attributes. The scraping process was automated, ensuring that the data could be updated regularly. After the data was collected, it was cleaned and structured for analysis using Pandas. The analysis focused on key market trends, such as the relationship between car price and mileage, price distribution by year, and price variations based on car color. These analyses were visualized through dynamic charts and graphs, providing users with clear, actionable insights into the market.

The web-based application was developed using Flask and provided a platform for users to interact with the data. Users could filter the data based on their preferences and view real-time analyses, enabling them to make informed decisions when buying or selling cars. In addition to the web-based tool, an Excel-based solution was created for offline use. This tool included VBA macros that automated data preparation and generated dynamic visualizations, allowing users to conduct similar analyses without needing to connect to the web application.

The results of the project demonstrated that the system was effective in meeting its objectives. The web scraping script proved to be robust, successfully collecting

a substantial amount of data that was used to generate meaningful insights. The analysis highlighted several important market trends, and the web application provided a user-friendly interface for exploring these trends. The Excel-based tool also worked as intended, offering users an alternative means of analyzing the data offline. Despite these successes, some challenges were encountered during the development process. These included managing rate limits imposed by the target website during scraping, as well as ensuring that the Excel tool remained synchronized with the web application's data.

Looking toward future improvements, several enhancements could be made to further refine the system. First, expanding the scope of data collection to include multiple car marketplaces would provide a more comprehensive view of the market, allowing for better comparative analysis across different platforms. Additionally, adding more filtering and sorting options to the web application would give users greater flexibility in their analyses, allowing them to focus on specific car models, geographic regions or other variables of interest.

In conclusion, this project successfully developed a comprehensive system for scraping, analyzing, and visualizing car data. It demonstrated the viability of using web scraping and data analysis techniques to provide valuable market insights. The web-based application and Excel tool offered users flexible options for exploring the data, both online and offline. With the proposed future improvements, this system could become even more powerful and useful for a wide range of users.

References

[1] Cox Automotive, "2023 cox automotive car buyer journey study." https://www.coxautoinc.com/market-insights/2023-car-buyer-journey-study/, 2024.

[2] L. M. López Wallace, "Predicting used car price from data collected with web scraping and machine learning techniques," Master's thesis, Torcuato Di Tella University, 2022.

[3] Alan R. Hevner, Salvatore T. March, Jinsoo Park, Sudha Ram, "Design science in information systems research," 2004.

[4] Julia Kaganh, "Blue book: What it is, how it works, effect." `https://www.investopedia.com/terms/b/bluebook.asp`, 2023.

[5] "Accessing edmunds content with dealership api key." `https://developer.edmunds.com/dealership_api_program.html`.

[6] Kenneth Reitz, "Requests: Http for humans™." `https://requests.readthedocs.io/en/latest/`, 2024.

[7] Leonard Richardson, "Beautiful soup documentation™." `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`, 2023.

[8] Leonard Richardson, "Beautiful soup." `https://www.crummy.com/software/BeautifulSoup/`, 2024.

[9] Python Software Foundation, ""csv file reading and writing." python 3.12.0 documentation." `https://docs.python.org/3/library/csv.html`, 2024.

[10] Jon Fincher, "Reading and writing csv files in python." `https://realpython.com/python-csv/`, 2024.

[11] SeleniumHQ, "The selenium browser automation project." https://www.selenium.dev/documentation/, 2023.

[12] "Scrapy 2.11 documentation." https://docs.scrapy.org/en/latest/, 2024.

[13] Zyte, "Scrapy: A fast and powerful scraping and web crawling framework." https://scrapy.org/, 2024.

[14] Pandas Development Team, "Pandas: Python data analysis library." https://pandas.pydata.org/, 2024.

[15] Mirko Stojiljković, "The pandas dataframe: Make working with data delightful." https://realpython.com/pandas-dataframe/, 2020.

[16] Remi M, "What is pyplot in matplotlib?." https://www.activestate.com/resources/quick-reads/what-is-pyplot-in-matplotlib/, 2021.

[17] John Hunter, Darren Dale, Eric Firing, Michael Droettboom, "matplotlib.pyplot." https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html, 2022.

[18] Michael Waskom, "seaborn: statistical data visualization." https://seaborn.pydata.org/, 2024.

[19] Melanie, "Seaborn: Everything you need to know about the python data visualization tool." https://datascientest.com/en/seaborn-everything-you-need-to-know-about-the-python-data-visualization-tool, 2022.

[20] Ian Eyre, "Visualizing data in python with seaborn." https://realpython.com/python-seaborn/, 2024.

[21] "Linear models - scikit-learn 1.3.0 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html, 2024.

[22] xgboost developers, "Xgboost documentation." https://xgboost.readthedocs.io/en/stable/, 2022.

[23] "Lightgbm documentation." `https://lightgbm.readthedocs.io/en/stable/Quick-Start.html`, 2023.

[24] "Flask documentation." `https://flask.palletsprojects.com/en/3.0.x/`, 2024.

[25] Miguel Grinberg, "Flask web development." O'Reilly Media, 2018.

[26] Django Software Foundation, "Django documentation." `https://docs.djangoproject.com/en/5.1/`, 2024.

[27] atlan, "Relational database vs nosql: 15 key differences to know!," 2024.

[28] Randolph West, Mike Ray, "What is sql server?," 2024.

[29] Jeffrey Erickson, "Mysql: Understanding what it is and how it's used," 2024.

[30] https://www.standvirtual.com/, 2024.