# IMG_256

# **CAPSTONE - JATO**

# 

# WEB SCRAPING

**SCOPE DOCUMENT**

# **JATO Monthly Payments Target Leasing Companies**

**Presented to the Panel Members of Merit Data & Technology**

**By**

**HARISH B [IN0055]**

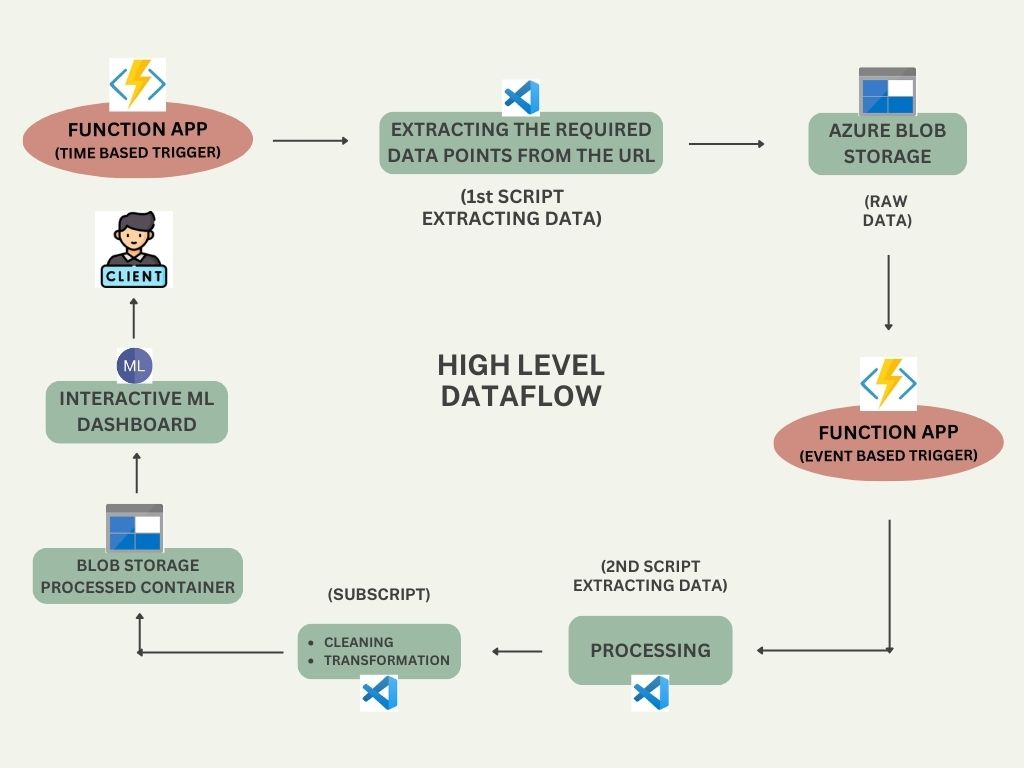
**SHIBIL S MATHEW [IN0052]**

**ANGELIN PRIYA C [IN0061]**

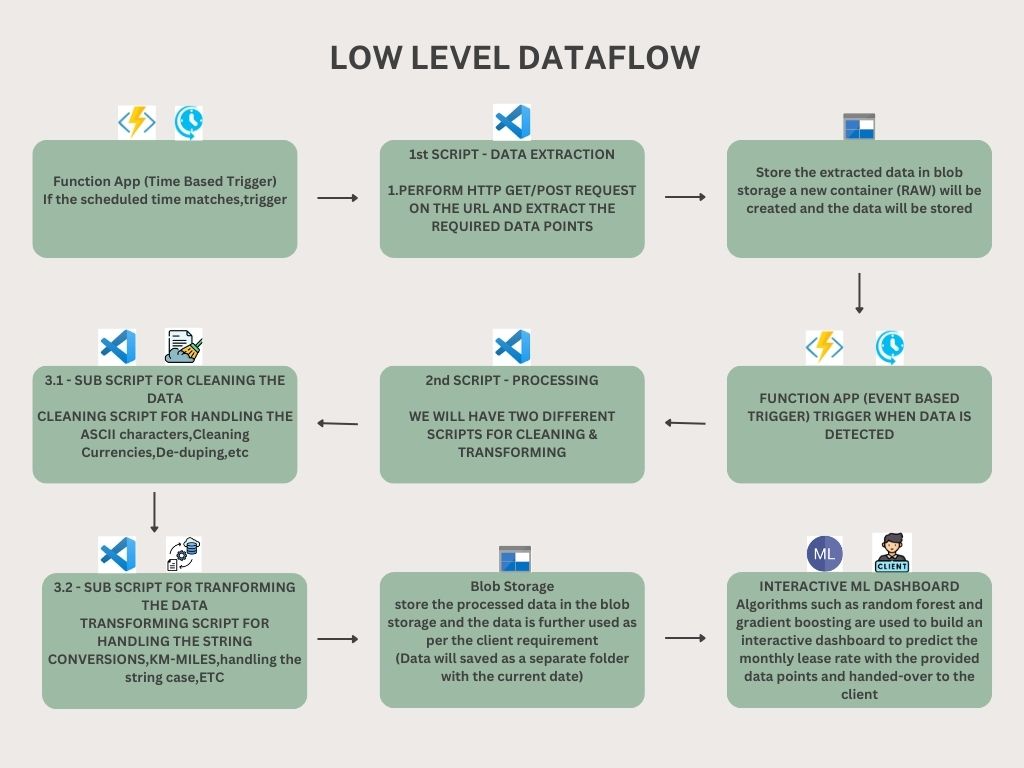
**TABLE OF CONTENTS**

| **Section** | **Subsection** | **Page Number** |
| --- | --- | --- |
| **HIGH LEVEL WORK FLOW**  **LOW LEVEL WORK FLOW**  **ARCHITECTURE SYSTEM** | | IV  V  VI |
| 1. **Introduction** | 1.1. Objective | 1 |
|  | 1.2. Scope | 1 |
| **2. Scrapy Overview** | 2.1. What is Scrapy? | 2 |
|  | 2.2. Features and Benefits | 2 |
|  | 2.3. Use Cases | 2 |
| **3. Project Setup** | 3.1. Installing Scrapy | 3 |
|  | 3.2. Creating a Scrapy Project | 3 |
|  | 3.3. Project Structure | 3 |
| **4. Understanding the Target Websites** | 4.1. Website 1: Selectalease | 4 |
|  | 4.2. Website 2: Lex Autolease | 4 |
|  | 4.3. Website 3: Sixt Neuwagen | 4 |
|  | 4.4. Website 4: AutoScout24 | 4 |
|  | 4.5. Website 5: Mobile.de | 5 |
| **5. Spider Creation and Configuration** | 5.1. Selectalease Spider | 6 |
|  | 5.2. Lex Autolease Spider | 6 |
|  | 5.3. Sixt Neuwagen Spider | 6 |
|  | 5.4. AutoScout24 Spider | 6 |
|  | 5.5. Mobile.de Spider | 7 |
| **6. Data Extraction Techniques** | 6.1. HTML Parsing | 8 |
|  | 6.2. JSON Extraction | 8 |
|  | 6.3. Combining HTML and JSON Extraction | 8 |
| **7. Time-Based Trigger** | 7.1. Check Current Time Against Schedule | 9 |
|  | 7.2. If the Scheduled Time Matches, Trigger | 9 |
| **8. Parsing Each Product URL** | 8.1. Iterate Through a List of Page URLs | 10 |
|  | 8.2. For Each URL, Validate URL Format | 10 |
| **9. Validating URL Format** | 10.1. Check URL Syntax and Structure | 11 |
|  | 10.2. Handle Incorrect or Malformed URLs | 11 |
| **10. Performing HTTP GET Request to Fetch URL** | 10.1. Send HTTP GET and POST Request | 12 |
|  | 10.2. Handle HTTP Response | 12 |
| **11. Checking for HTTP Errors** | 11.1. Identify Common HTTP Errors | 13 |
|  | 11.2. Implement Error Handling and Retry Logic | 13 |
| **12. Processing Raw Data** | 12.1. Load Raw Data | 14 |
|  | 12.2. Parse HTML Using Scrapy | 14 |
|  | 12.3. Identify Key Elements | 14 |
|  | 12.4. Cleaning the Data | 14 |
|  | 12.5. Normalize the Data | 15 |
| **13. 1st Script - Data Extraction** | 13.1. Perform HTTP GET Request on the URL | 16 |
|  | 13.2. Parse HTML to Extract Product Details | 16 |
|  | 13.3. Handle Missing Data | 16 |
| **14. Storing Extracted Data in Blob Storage** | 14.1. Create a New Container | 17 |
|  | 14.2. Store the Data | 17 |
| **15. Function App (Event-Based Trigger)** | 15.1. Trigger When Data is Detected | 18 |
| **16. 3rd Script - Processing** | 16.1. Cleaning the Data | 19 |
|  | 16.2. Transforming the Data | 19 |
| **17. Storing Processed Data in Blob Storage** | 17.1. Further Use as per Client Requirement | 20 |
|  | 17.2. Creating Containers for Different Data Types | 20 |
| **18. Creating Output Files** | 18.1. Generate Reports | 21 |
|  | 18.2 Store output files in blob storage | 21 |
|  | 18.2. Store Output Files in Blob Storage | 21 |
| 1. **ML Dashboard** | 19.1 Introduction | 22 |
|  | 19.2 Dataset and Feature Selection | 22 |
|  | 19.3 Data Splitting | 22 |
|  | 19.4 Model Building and Training | 22 |
|  | 19.5 Model Evaluation | 23 |
|  | 19.6 Interactive Predictive Interface | 23 |
|  | 19.7 Conclusion | 23 |
| **20. Final Data Validation** | 20.1. Ensure Accuracy and Completeness | 24 |
|  | 20.2 Perform Quality Checks | 24 |

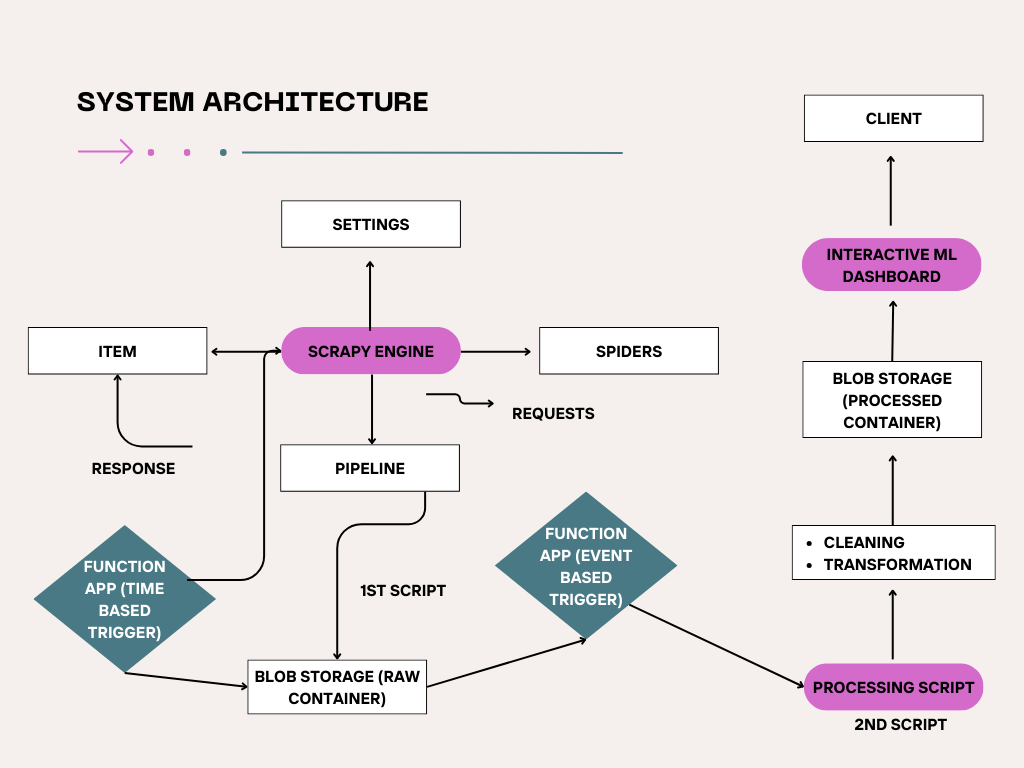
**HIGH LEVEL WORKFLOW**



**LOW LEVEL WORKFLOW**



**ARCHITECTURE SYSTEM**



### ****CHAPTER 1****

### ****Introduction****

#### ****1.1. Objective****

The primary objective of this report is to provide a thorough and comprehensive guide to utilizing Scrapy, a powerful web scraping framework, for extracting and processing data from various online sources. The report aims to cover the entire lifecycle of a web scraping project, starting from the initial setup of the Scrapy framework to the final stages of data processing and storage. This guide is designed to equip users with the knowledge and tools necessary to effectively harness Scrapy’s capabilities for extracting structured data from complex websites, handling data errors, and managing large-scale data extraction projects.

#### ****1.2. Scope****

This report encompasses a wide range of topics crucial for successfully implementing a web scraping project using Scrapy. It includes an in-depth overview of Scrapy’s features and benefits, detailed steps for setting up a Scrapy project, and an exploration of various target websites. The report also covers the creation and configuration of Scrapy spiders tailored to different websites, data extraction techniques using both HTML parsing and JSON extraction, and strategies for handling and storing the extracted data. The scope extends to time-based and event-based triggers for automating data extraction processes and ensuring that the entire workflow operates efficiently.

### ****CHAPTER 2****

### ****Scrapy Overview****

#### ****2.1. What is Scrapy?****

Scrapy is an open-source framework for web scraping and web crawling, developed in Python. It provides a robust and scalable platform for extracting data from websites, enabling users to build spiders that navigate through web pages, collect information, and process it according to specific requirements. Scrapy is designed to handle large-scale data extraction tasks efficiently, thanks to its asynchronous networking capabilities and built-in support for handling multiple requests concurrently..

#### ****2.2. Features and Benefits****

Scrapy offers a multitude of features that make it a preferred choice for web scraping tasks. Its asynchronous processing allows it to handle numerous requests simultaneously, which significantly reduces the time required for data extraction. Scrapy provides built-in support for exporting data in various formats such as JSON, CSV, and XML, which facilitates easy integration with data analysis tools. The framework's use of XPath and CSS selectors enables precise extraction of data from complex HTML structures. Additionally, Scrapy's pipeline system allows for modular data processing, including cleaning, validation, and storage. Its support for user-agent rotation and proxy usage helps circumvent IP bans and access restricted content.

#### ****2.3. Use Cases****

Scrapy is versatile and can be employed in a variety of scenarios. For instance, e-commerce websites can be scraped to gather product information, including prices, descriptions, and reviews, which can be used for price comparison and competitive analysis. News aggregators can utilize Scrapy to compile articles from multiple sources, providing users with consolidated news coverage. In the real estate sector, Scrapy can be used to extract property listings, including prices and features, from real estate websites for market analysis. The framework's capability to handle different data formats and large volumes of data makes it suitable for many other applications as well.

### ****CHAPTER 3****

### ****Project Setup****

#### ****3.1. Installing Scrapy****

The installation of Scrapy is a straightforward process that requires Python and pip, Python's package installer. Scrapy is installed via pip, which fetches the latest version of the framework and its dependencies from the Python Package Index. Ensuring that Python is up to date and that pip is properly configured is crucial for a smooth installation process. Once installed, Scrapy provides a command-line interface for creating and managing web scraping projects, which forms the foundation for building spiders and executing scraping tasks.

#### ****3.2. Creating a Scrapy Project****

Creating a new Scrapy project involves using the scrapy startproject command, which sets up a new project directory with a predefined structure. This structure includes essential components such as the spiders directory for defining scraping logic, the items.py file for specifying data structures, the pipelines.py file for processing data, and the settings.py file for configuration. This organized setup facilitates efficient development and management of web scraping tasks, allowing users to focus on customizing their spiders and pipelines according to their specific needs.

#### ****3.3. Project Structure****

The structure of a Scrapy project is designed to promote modularity and maintainability. The spiders directory contains the individual spiders responsible for scraping data from different websites. The items.py file defines the data model, specifying the fields that will be extracted from the web pages. The pipelines.py file is used to implement data processing logic, such as cleaning and validation, before storing the data. The settings.py file holds configuration options, including user-agent settings, download delays, and pipeline configurations. This structured approach ensures that each component of the project is well-organized and easily manageable.

### ****CHAPTER 4****

### ****Understanding the Target Websites****

#### ****4.1. Website 1: Selectalease****

Selectalease is a platform specializing in vehicle leasing, providing users with options to lease various types of vehicles. Understanding the website’s structure involves analyzing its layout to identify how lease terms, vehicle specifications, and pricing information are presented. This analysis helps in creating targeted scraping rules that accurately capture the relevant data from Selectalease.

#### ****4.2. Website 2: Lex Autolease****

Lex Autolease offers a range of vehicle leasing services, including detailed information on leasing options, vehicle features, and financial terms. Studying the website’s structure is crucial for identifying how data is organized and presented, which informs the development of effective scraping strategies to extract comprehensive lease details and vehicle information.

#### ****4.3. Website 3: Sixt Neuwagen****

Sixt Neuwagen provides listings for new cars available for leasing or purchase. Analyzing the website involves understanding how vehicle information, lease conditions, and pricing are displayed. This insight is essential for creating spiders that can navigate the site, extract relevant data, and handle any variations in data presentation.

#### ****4.4. Website 4: AutoScout24****

AutoScout24 is a large car marketplace offering extensive vehicle listings. To effectively scrape data from AutoScout24, it is important to review how vehicle details, such as make, model, year, and pricing, are organized on the site. This understanding helps in designing spiders that can accurately capture and process vehicle information from the marketplace.

#### ****4.5. Website 5: Mobile.de****

Mobile.de is another prominent vehicle marketplace with a broad range of car listings. Examining the website’s structure involves identifying how data related to vehicle listings, seller information, and pricing is presented. This knowledge is key to developing spiders that can efficiently extract detailed information from the site.

### ****CHAPTER 5****

### ****Spider Creation and Configuration****

#### ****5.1. Selectalease Spider****

Creating a spider for Selectalease involves defining how the spider should navigate the website, which URLs to visit, and how to extract relevant lease details and vehicle specifications. Configuring the spider requires setting up rules for following links, handling pagination, and managing data extraction to ensure accurate and complete data retrieval.

#### ****5.2. Lex Autolease Spider****

The Lex Autolease spider is designed to extract information related to vehicle leasing options. This involves configuring the spider to navigate through different pages, identify and extract relevant lease terms, vehicle features, and financial details. Proper configuration ensures that the spider can handle various page structures and data formats.

#### ****5.3. Sixt Neuwagen Spider****

Developing a spider for Sixt Neuwagen requires setting up rules for navigating through the site and extracting data about new cars available for leasing or purchase. The spider must be configured to handle the website’s layout and extract detailed vehicle information, lease conditions, and pricing.

#### ****5.4. AutoScout24 Spider****

The AutoScout24 spider is configured to scrape data from a large vehicle marketplace. This involves setting up the spider to navigate through vehicle listings, extract information such as make, model, year, and price, and handle any variations in data presentation. Effective configuration ensures that the spider can capture all relevant details from the site.

#### ****5.5. Mobile.de Spider****

Creating a spider for Mobile.de involves defining how to navigate the site, extract vehicle data, and handle different data formats. The spider must be configured to manage large volumes of listings and accurately capture information such as vehicle details and seller information.

### ****CHAPTER 6****

### ****Data Extraction Techniques****

#### ****6.1. HTML Parsing****

HTML parsing involves extracting structured data from the raw HTML content retrieved from web pages. Scrapy provides tools for parsing HTML using XPath and CSS selectors, allowing users to target specific elements on the page and extract relevant information. This technique is essential for handling websites with complex layouts and extracting data embedded within HTML tags.

#### ****6.2. JSON Extraction****

JSON extraction involves retrieving and processing data from web pages that use JSON format for data representation. Scrapy can handle JSON responses from APIs or endpoints, parsing the JSON data to extract structured information. This technique is useful for websites that provide data in JSON format, such as APIs or dynamic content.

#### ****6.3. Combining HTML and JSON Extraction****

Combining HTML and JSON extraction techniques is necessary when dealing with websites that use both formats for data presentation. This approach involves parsing HTML for some data while simultaneously retrieving and processing JSON data for other information. By integrating these techniques, users can effectively handle complex websites with diverse data formats.

### ****CHAPTER 7****

### ****Time-Based Trigger****

#### ****7.1. Check Current Time Against Schedule****

Implementing a time-based trigger involves creating a mechanism to check the current system time against a predefined schedule. This check determines whether it is the appropriate time to initiate data extraction or processing tasks. It ensures that tasks are performed at specified intervals or times, optimizing resource usage and ensuring timely data updates.

#### ****7.2. If the Scheduled Time Matches, Trigger****

Once the scheduled time is confirmed, the system triggers the appropriate actions, such as starting data extraction or processing. This trigger mechanism ensures that tasks are executed according to the schedule, maintaining consistency and efficiency in the data extraction process.

### ****CHAPTER 8****

### ****Parsing Each Product URL****

#### ****8.1. Iterate Through a List of Page URLs****

Parsing each product URL involves iterating through a list of URLs to access individual product pages. This process requires defining how the spider should navigate through the list, fetch page content, and extract relevant details. Iteration ensures that all product pages are accessed and processed systematically.

#### ****8.2. For Each URL, Validate URL Format****

Validating the format of each URL is crucial for ensuring that the URLs are correctly structured and accessible. This step involves checking for common URL format issues, such as incorrect syntax or broken links, to avoid errors during data extraction and ensure that only valid URLs are processed.

### ****CHAPTER 9****

### ****Validating URL Format****

#### ****9.1. Check URL Syntax and Structure****

Validating URL syntax and structure involves ensuring that URLs conform to standard formatting rules and are properly structured. This includes checking for correct domain names, path segments, and query parameters. Proper URL validation helps prevent errors and ensures successful navigation to the target web pages.

#### ****9.2. Handle Incorrect or Malformed URLs****

Handling incorrect or malformed URLs involves implementing error-checking mechanisms to identify and manage URLs that do not conform to expected formats. This may include skipping invalid URLs, logging errors for review, and implementing fallback strategies to handle missing or inaccessible pages.

### ****CHAPTER 10****

### ****Performing HTTP GET and POST Request to Fetch URL****

#### ****10.1. Send HTTP GET Request****

Performing an HTTP GET and POST request involves sending a request to the server to retrieve the content of the specified URL. This step is crucial for accessing the web page data required for scraping. The request must be properly configured to handle different types of responses and ensure that the content is retrieved successfully.

#### ****10.2. Handle HTTP Response****

Handling the HTTP response involves processing the data received from the server. This includes checking the response status code to determine whether the request was successful and managing different response types, such as HTML or JSON. Proper handling ensures that the content is correctly accessed and processed for further extraction.

### ****CHAPTER 11****

### ****Checking for HTTP Errors****

#### ****11.1. Identify Common HTTP Errors****

Identifying common HTTP errors involves recognizing typical issues that may occur during data retrieval, such as 404 Not Found or 500 Internal Server Error. Understanding these errors helps in implementing error-handling strategies and ensures that the scraping process can address and recover from common issues.

#### ****11.2. Implement Error Handling and Retry Logic****

Implementing error handling and retry logic involves creating mechanisms to manage and recover from errors encountered during data extraction. This includes retrying failed requests, logging errors, and applying strategies to handle issues such as server downtime or network problems.

### ****CHAPTER 12****

### ****Processing Raw Data****

#### ****12.1. Load Raw Data****

Loading raw data involves retrieving and preparing the data extracted from web pages for processing. This step includes organizing the data into a format suitable for analysis, such as loading it into data structures or storage systems. Proper data loading ensures that the data is ready for further processing and analysis.

#### ****12.2. Parse HTML Using Scrapy****

Parsing HTML using Scrapy involves utilizing the framework's built-in tools to extract structured data from raw HTML content. Scrapy's XPath and CSS selectors are used to target specific elements on the page, allowing for precise data extraction from complex HTML structures.

#### ****12.3. Identify Key Elements****

Identifying key elements involves recognizing and extracting important data points from the HTML content. This includes locating specific tags, attributes, or text that are relevant to the scraping objectives. Accurate identification of key elements ensures that the extracted data is meaningful and useful.

#### ****12.4. Cleaning the Data****

Cleaning the data involves removing or correcting any inconsistencies or errors in the extracted information. This may include addressing issues such as missing values, incorrect formats, or irrelevant data. Data cleaning ensures that the final dataset is accurate, reliable, and ready for analysis.

#### ****12.5. Normalize the Data****

Normalizing the data involves standardizing the format and structure of the extracted information. This includes converting data into a consistent format, handling units of measurement, and ensuring that data fields are uniformly represented. Data normalization is essential for ensuring that the dataset is compatible with analysis tools and processes.

### ****CHAPTER 13****

### ****1st Script - Data Extraction****

#### ****13.1. Perform HTTP GET/POST Request on the URL****

Performing an HTTP GET and POST request involves sending requests to the extracted URLs to retrieve the content of the product pages. This step is crucial for accessing the detailed information needed for data extraction and processing.

#### ****13.2. Parse HTML/JSON to Extract Product Details****

Parsing HTML and JSON to extract product details involves using Scrapy’s tools to analyze the HTML and JSON content of each product page and retrieve relevant information such as product names, prices, and descriptions. This step ensures that the extracted data is accurate and comprehensive.

#### ****13.3. Handle Missing Data****

Handling missing data involves implementing strategies to manage cases where certain information is not available or is incomplete. This may include setting default values, skipping incomplete records, or applying error-handling mechanisms to ensure that the final dataset remains usable.

### ****CHAPTER 14****

### ****Storing Extracted Data in Blob Storage****

#### ****14.1. Create a New Container****

Creating a new container in blob storage for storing the extracted data involves setting up a dedicated storage location where the processed information can be securely saved. This ensures that the data is accessible for further analysis or reporting.

#### ****14.2. Store the Data****

Storing the data involves saving the processed information in the newly created container. This step ensures that the data is organized, secure, and available for subsequent use or integration into other systems.

### ****CHAPTER 15****

### ****Function App (Event-Based Trigger)****

#### ****15.1. Trigger When Data is Detected****

Implementing an event-based trigger for the data involves setting up mechanisms to automatically initiate further actions, such as additional processing or analysis, when new data is detected in the storage system. This ensures that the workflow remains efficient and responsive to changes in the data.

### ****CHAPTER 16****

### ****2nd Script - Processing****

#### ****16.1. Cleaning the Data****

Cleaning the data involves applying scripts to address any inconsistencies or errors in the extracted information. This step ensures that the data is accurate and reliable, removing any issues that may have arisen during the extraction process.

#### ****16.2. Transforming the Data****

Transforming the data involves applying scripts to convert data formats, such as converting distances from kilometers to miles. This step ensures that the data is standardized and compatible with the desired output formats or analysis requirements.

### ****CHAPTER 17****

### ****Storing Processed Data in Blob Storage****

#### ****17.1. Further Use as per Client Requirement****

Storing processed data in blob storage involves saving the final dataset in a format that meets the client’s requirements. This may include organizing the data for specific types of analysis, generating reports, or providing access to the data for further use.

#### ****17.2. Creating Containers for Different Data Types****

Creating containers for different data types involves organizing the processed data into separate containers based on its type or purpose. This ensures that the data is easily accessible and manageable, supporting efficient data retrieval and use.

### ****CHAPTER 18****

### ****Creating Output Files****

#### ****18.1. Generate Reports****

Generating reports involves creating summary documents or visualizations based on the extracted and processed data. This step provides valuable insights and facilitates decision-making by presenting the data in a clear and accessible format.

#### ****18.2. Store Output Files in Blob Storage****

Storing output files in blob storage involves saving the generated reports and visualizations in a secure and accessible location. This ensures that the output files are available for review, sharing, or further analysis.

### ****CHAPTER 19****

****ML DASHBOARD****

### ****19.1.**** Introduction

This report provides a comprehensive analysis of a machine learning model designed to predict the regular monthly installment amount for retail based on input features such as price, contract duration in months, and yearly mileage in kilometers. The model leverages a Random Forest Regressor to build accurate predictions, and an interactive widget interface allows users to input data and receive predictions in real-time.

### 19.2. Dataset and Feature Selection

The dataset used in this analysis is loaded from a CSV file (output (1).csv). The key features selected for this model include Price, Contract\_Duration\_Months, and Yearly\_Mileage\_Km, which serve as the independent variables (X). The target variable (y) that the model aims to predict is Regular\_Monthly\_Instalment\_Amount\_Retail.

### 19.3. Data Splitting

To evaluate the performance of the model, the dataset is split into training and testing sets. The data is divided with 80% used for training and 20% for testing, ensuring that the model is trained on a substantial portion of the data while still allowing for a robust evaluation of its performance.

### 19.4 Model Building and Training

A Random Forest Regressor is employed as the machine learning model. This algorithm is chosen for its ability to handle complex relationships and its robustness against overfitting. The model is trained on the training set with 100 estimators, and the random seed is set to 42 to ensure reproducibility of results.

### 19.5 Model Evaluation

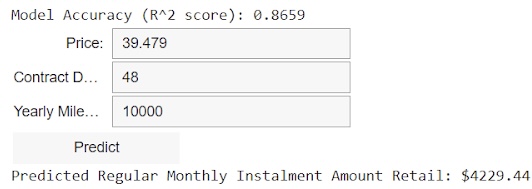
The model's accuracy is assessed using the R^2 score, a standard metric for regression models. The R^2 score indicates how well the model's predictions match the actual values, with a score closer to 1 indicating a better fit. After testing, the model achieves an R^2 score of approximately 0.8956, indicating strong predictive capabilities.

### 19.6 Interactive Prediction Interface

An interactive widget interface is developed to allow users to input the Price, Contract\_Duration\_Months, and Yearly\_Mileage\_Km, and receive a real-time prediction of the regular monthly installment amount. The interface is built using ipywidgets, providing an intuitive and user-friendly way to interact with the model. Users can input their data through the widgets, and upon clicking the 'Predict' button, the model provides an instant prediction.

### 19.7 Conclusion

This report demonstrates the successful application of a Random Forest Regressor for predicting regular monthly installment amounts based on selected automotive financial parameters. The model shows high accuracy and the interactive widget interface enhances user engagement, making it practical for real-world applications where users need immediate predictions based on varying inputs.



**Workflow of the Dashboard**

### ****CHAPTER 20****

### ****Final Data Validation****

#### ****20.1. Ensure Accuracy and Completeness****

Final data validation involves checking the processed dataset to ensure its accuracy and completeness. This step includes verifying that all required data points are present, correct, and consistent with the source data.

#### ****20.2. Perform Quality Checks****

Performing quality checks involves applying additional validation criteria to ensure that the data meets the required standards for quality and reliability. This may include reviewing data integrity, consistency, and completeness to confirm that the final dataset is ready for use.