**Technical Analysis**

**JATO\_Autoscout**

**Overview:**

This Scrapy spider is designed to scrape detailed vehicle information from Autoscout24. It handles pagination, extracts vehicle data from each page, and processes individual vehicle listings to fetch additional details.

**Components:**

**Class Definition:**

* VehicleDetailsSpider is a Scrapy Spider class that inherits from scrapy.Spider.

**Attributes:**

* name: Unique identifier for the spider. It is set to "JATO\_autoscout".
* start\_url: The URL where the spider starts scraping. This is a query URL for vehicle listings on Autoscout24.
* **URL:** https://www.autoscout24.de/lst?atype=C&cy=D&damaged\_listing=exclude&desc=0&hasleasing=true&ocs\_listing=include&offer=N&page=1&powertype=kw&search\_id=2ayclr8z17x
* **custom\_settings**: Custom settings for the spider, such as disabling ROBOTSTXT\_OBEY to allow scraping.

**Methods:**

start\_requests:

* **Purpose:** Initiates the scraping process.
* **Details:** It yields an initial request to the start\_url and specifies pagination as the callback method to handle the response.

Pagination:

* **Purpose:** Handles pagination to request all pages of vehicle listings.
* **Details:**
  + - 1. Extracts the total number of pages from the pagination elements.
      2. Loops through all pages and yields a request for each page.
      3. Constructs the URL for each page based on the current page number.
      4. Includes a break statement for testing purposes to prevent scraping beyond the first page.

Parse:

* **Purpose:** Parses vehicle listings from each page and extracts data.
* **Details:**
  + - 1. Extracts JSON data embedded in the <script> tag with id="\_\_NEXT\_DATA\_\_".
      2. Parses the JSON data to get vehicle listings.
      3. Iterates over each listing to extract vehicle details.
      4. Constructs derived and version names, and extracts additional information related to leasing.
      5. Yields requests to fetch additional data for each vehicle using the data\_parse method.

data\_parse:

* **Purpose:** Fetches detailed data about the vehicle from individual listing pages.
* **Details:**
  + - 1. Extracts JSON data from the <script> tag with id="\_\_NEXT\_DATA\_\_".
      2. Parses the JSON data to get detailed vehicle data.
      3. Extracts additional attributes such as number of doors, body type, and insurance details.
      4. Yields the final item with all extracted data.

**Error Handling:**

* The spider includes exception handling using try and except blocks.
* Logs errors with detailed messages to help diagnose issues during scraping.

**Data Extraction Details:**

**From Listings:**

* + Extracts make, model, trim, power train, derivative, version name, and more.
  + Handles different customer types (Business vs. non-Business) for installment amounts.
  + Constructs URLs for individual vehicle listings

**From Detailed Data:**

* + Extracts number of doors, body type, paint type, monthly payment provider, final installment amount, and insurance details.

**Settings and Configuration:**

* ROBOTSTXT\_OBEY is set to False to bypass robots.txt restrictions. This may be adjusted based on the site's policies and legal considerations.

**Usage:**

* Run this spider using the Scrapy command: “ scrapy crawl JATO\_autoscout.”

**Variables Declared Inside item[ ]:**

| ****Variable Name**** |
| --- |
| item['Make'] |
| item['Model'] |
| item['Body\_Type'] |
| item['Trim'] |
| item['Number\_Of\_Doors'] |
| item['Built\_Option\_Pack'] |
| item['Derivative'] |
| item['Derivative\_Translated\_English'] |
| item['Configurator\_Model\_Year'] |
| item['Version\_Name'] |
| item['Currency'] |
| item['Delivery\_Costs\_Retail'] |
| item['Country'] |
| item['Research\_Date'] |
| item['Customer\_Type'] |
| item['Product\_Name'] |
| item['Data\_Source'] |
| item['Other\_Sources'] |
| item['Vehicle\_Price\_Reference'] |
| item['Yearly\_Mileage\_Km'] |
| item['Yearly\_Mileage\_Miles'] |
| item['Total\_Contract\_Mileage\_Km'] |
| item['Total\_Contract\_Mileage\_Miles'] |
| item['Deposit\_Retail'] |
| item['Regular\_Monthly\_Instalment\_Amount\_Retail'] |
| item['Downpayment\_Allowance\_Base'] |
| item['Regular\_Monthly\_Instalment\_Amount\_Base'] |
| item['Product\_Description'] |
| item['Monthly\_Payment\_Provider\_Name'] |
| item['Price'] |
| item['Insurance'] |
| item['Advertised\_Price\_Point\_Monthly\_Payment'] |
| item['Insurance\_Description'] |

# JATO\_lexautolease

**OVERVIEW:**

The LeaseScrapSpider is a Scrapy spider designed to scrape vehicle leasing data from the Lex Autolease website. This spider performs multiple HTTP POST requests to retrieve data about different vehicle manufacturers for both personal and business leasing customers.

**Breakdown:**

#### Class: LeaseScrapSpider

* **Name:** "JATO\_lexautolease"
* **Allowed Domains:** ["quotes.lexautolease.co.uk", "lexautolease.co.uk"]
* **Start URL:** 'https://quotes.lexautolease.co.uk/Quote/GetQuoteManufacturers'

#### Custom Settings:

The spider includes a custom setting to introduce a delay between requests:

* **DOWNLOAD\_DELAY:** 1 (seconds delay between requests)

#### Cookies:

A dictionary of cookies is included to simulate a session with the website. This allows the spider to maintain state across multiple requests.

#### Headers:

#### A dictionary of headers is provided to make the requests appear as if they are coming from a web browser, ensuring compatibility with the target website's anti-scraping mechanisms.

### Methods:

### start\_requests(self)

* + **Purpose:** Initiates the scraping process by sending POST requests to the target URL for both personal and business customer types.
  + **Parameters:** None
  + **Request Details:**
    - Sends a POST request to 'https://quotes.lexautolease.co.uk/Quote/GetQuoteManufacturers'.
    - Uses a JSON body (manufacturer\_payload) that specifies the busType as either 'personal' or 'business', defaultType as None, and vechicleType as 'ALL'.
    - Adds custom headers and cookies to mimic a browser request.
    - Stores the customer type in the request metadata (meta) for use in the callback function.
    - **Callback:** manufacturer\_parse
  + **Exception Handling:** Logs an error if the request fails.

1. **contract\_termination(self, response)**
   * **Purpose:** Parses the details related to contract termination costs.
   * **Parameters:**
     + response: The response object from the previous request.
   * **Data Extraction:**
     + Uses XPath to extract the description of the cost to terminate the contract from the page.
     + Applies a regular expression to extract a numeric value from the description.
     + Stores the extracted value in an item of type VehicleItem under the key 'Cost\_To\_Terminate\_Contract\_Description'.
   * **Item Storage:** Uses the VehicleItem data structure to hold the parsed information.

#### Usage:

* Run this spider using Scrapy, typically by executing the following command in the terminal:

“scrapy crawl JATO\_lexautolease”

### Key Components:

* **start\_requests():** Initiates requests for different customer types.
* **contract\_termination():** Parses specific data regarding contract termination from the response.

### Error Handling:

The spider handles errors in the start\_requests method by using a try-except block and logging any exceptions that occur during the request process.

**Variables Declared Inside item[ ]:**

| **Variable Name** |
| --- |
| Maintenance\_Description |
| Repair |
| Repair\_Description |
| Service |
| Service\_Description |
| Tyres |
| Tyres\_Description |
| Flexible\_Early\_Cancellation\_Possible |
| Monthly\_Payment\_Provider\_Name |
| Monthly\_Payment\_Provider\_Type |
| Make |
| Model |
| Body\_Type |
| Trim |
| Number\_Of\_Doors |
| Built\_Option\_Pack |
| Derivative |
| Derivative\_Translated\_English |
| Configurator\_Model\_Year |
| Version\_Name |
| Version\_Name\_Translated\_English |
| Currency |
| Country |
| Data\_Source |
| Other\_Sources |
| Web\_Source\_Url |
| Contract\_Duration\_Months |
| Of\_Monthly\_Instalments |
| Of\_Monthly\_Payments\_In\_Advance |
| Maintenance |
| Customer\_Type |
| Regular\_Monthly\_Instalment\_Amount\_Retail |
| First\_Monthly\_Instalment\_Amount\_Retail |
| Regular\_Monthly\_Instalment\_Amount\_Base |
| First\_Monthly\_Instalment\_Amount\_Base |
| Yearly\_Mileage\_Miles |
| Yearly\_Mileage\_Km |
| Total\_Contract\_Mileage\_Miles |
| Total\_Contract\_Mileage\_Km |

**JATO-Sixt-Neuwagen.De**

### Overview:

SixtLeasingSpider is a Scrapy spider designed to scrape vehicle leasing information from the Sixt leasing website. It handles configuration and scraping of vehicle details by navigating to the provided URL and managing requests with specific settings.

### Spider Information:

* **Name:** JATO\_sixt\_neuwagen
* **Allowed Domains:** sixt-neuwagen.de
* **Start URL:** https://www.sixt-neuwagen.de/konfigurieren?customerType=private

### Custom Settings:

* **RETRY\_ENABLED:** True
  + Enables retries for failed requests.
* **RETRY\_TIMES:** 5
  + Sets the number of retries to 5 to handle temporary failures.
* **RETRY\_HTTP\_CODES:** [429, 500, 502, 503, 504]
  + Specifies HTTP status codes that should trigger a retry.
* **DOWNLOAD\_DELAY:** 0.5
  + Reduces the delay between consecutive requests to speed up scraping.
* **AUTOTHROTTLE\_ENABLED:** True
  + Enables auto-throttling to manage request rates dynamically.
* **AUTOTHROTTLE\_START\_DELAY:** 0.5
  + Sets the initial download delay to 0.5 seconds.
* **AUTOTHROTTLE\_MAX\_DELAY:** 5
  + Limits the maximum download delay to 5 seconds.
* **AUTOTHROTTLE\_TARGET\_CONCURRENCY:** 3.0
  + Sets the target concurrency for requests to 3.
* **AUTOTHROTTLE\_DEBUG:** False
  + Disables throttling debug information to reduce logging overhead.

### Cookies:

The spider uses a set of cookies to manage session and user tracking. Example cookies include:

* \_\_cmpconsent44521
* \_\_cmpcccu44521
* mf\_user
* cf\_clearance
* vehicleConfig

### Headers:

The spider sends custom headers with each request to mimic a real browser and manage content type. Key headers include:

* accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7
* accept-language: en-US,en;q=0.9
* cache-control: no-cache
* pragma: no-cache
* sec-ch-ua: "Not)A;Brand";v="99", "Google Chrome";v="127", "Chromium";v="127"

### Parsing Methods:

**parse:**

The main parsing method that initiates the request to the start URL and handles the response.

#### services\_parse:

* **Purpose:** Extracts information about additional services related to the vehicle, such as road assistance, repair, courtesy car, maintenance, insurance, and tyres.
* **Implementation:**
  + **Road Assistance:** Extracts road assistance details from assistance['description'] and assistance['additionalInformation'].
  + **Repair:** Retrieves repair information from assistance['additionalInformation'].
  + **Courtesy Car:** Extracts courtesy car details from assistance['additionalInformation'].
  + **Maintenance:** Extracts maintenance details from maintenance\_data['name'], maintenance\_data['description'], and maintenance\_data['rate']['netAmount'].
  + **Insurance:** Retrieves insurance details from insurance['rate']['netAmount'] and insurance['additionalInformation'].
  + **Tyres:** Extracts tyre details from tyre\_data['name'], tyre\_data['description'], and tyre\_data['rate']['netAmount'].

#### contract\_parse:

* **Purpose:** Handles the parsing of contract-specific details such as pricing, monthly payments, and contract duration.
* **Implementation:**
  + **Contract Duration:** Extracts the contract duration in months from data['config']['term'].
  + **Monthly Payments:** Extracts the advertised price point for monthly payments from data['target']['environmentalBonusAvailable'].
  + **Mileage Information:** Calculates yearly mileage in kilometers and miles, total contract mileage, and total contract mileage in miles.
  + **Financing Details:** Retrieves finance details, including base price, sourced financed amount percentage, final payment percentage, and other financial terms.
  + **Retail and Base Values:** Handles calculations related to retail and base values, including deposit amounts, regular monthly instalments, and final payments.

### Items:

The spider extracts various vehicle details and stores them in VehicleItem, which is defined in scrapy\_project.items. This may include attributes like model, trim, and other leasing details.

### Error Handling:

The spider handles potential errors and retries failed requests according to the specified settings.

### Usage:

To run the spider, use the Scrapy command: “scrapy crawl JATO\_sixt\_neuwagen “

**Variables Declared Inside item[ ]:**

| **Variable** |  |
| --- | --- |
| Make |  |
| Model |  |
| Body\_Type |  |
| Number\_Of\_Doors |  |
| Derivative |  |
| Derivative\_Translated\_English |  |
| Trim |  |
| Power\_Train |  |
| Version\_Name |  |
| Version\_Name\_Translated\_English |  |
| Customer\_Type |  |
| Product\_Name |  |
| Data\_Source |  |
| Other\_Sources |  |
| Currency |  |
| Country |  |
| Web\_Source\_Url |  |
| Advertised\_Price\_Point\_Monthly\_Payment |  |
| Contract\_Duration\_Months |  |
| Of\_Monthly\_Instalments |  |
| Yearly\_Mileage\_Km |  |
| Yearly\_Mileage\_Miles |  |
| Total\_Contract\_Mileage\_Km |  |
| Total\_Contract\_Mileage\_Miles |  |
| Deposit\_Percentage\_Of\_Price |  |
| Price |  |
| Sourced\_Financed\_Amount\_Percentage\_Of\_Price |  |
| Final\_Payment\_Percentage\_Of\_Price |  |
| Deposit\_Base |  |
| Regular\_Monthly\_Instalment\_Amount\_Base |  |
| Sourced\_Financed\_Amount\_Base |  |
| Final\_Payment\_Base |  |
| Deposit\_Retail |  |
| Regular\_Monthly\_Instalment\_Amount\_Retail |  |
| Delivery\_Costs\_Retail |  |
| Sourced\_Financed\_Amount\_Retail |  |
| Final\_Payment\_Retail |  |
| Road\_Assistance |  |
| Road\_Assistance\_Description |  |
| Repair |  |
| Repair\_Description |  |
| Courtesy\_Car |  |
| Courtesy\_Car\_Description |  |
| Maintenance |  |
| Maintenance\_Description |  |
| Maintenance\_Amount |  |
| Insurance |  |
| Insurance\_Description |  |
| Monthly\_Payment\_Provider\_Name |  |
| Tyres |  |
| Tyres\_Description |  |
| Tyres\_Amount |  |

**JATO\_Mobile\_De**

### ****Overview****

MobileSpider is a Scrapy spider designed to scrape vehicle listings from the Mobile.de website. It navigates through search result pages and extracts various details about vehicles, including make, model, pricing, and leasing information. The spider handles pagination to process multiple pages of search results.

### ****Spider Configuration****

* **Name**: JATO\_mobile\_de
* **Allowed Domains**: suchen.mobile.de
* **Start URLs**: https://suchen.mobile.de/

### ****Custom Headers and Cookies****

### ****Headers****:

* + 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
  + 'Accept-Language': 'en-US,en;q=0.9'
  + 'Accept-Encoding': 'gzip, deflate, br'
  + 'Connection': 'keep-alive'

**Cookies**:

* '\_bb\_lat\_long': 'MTIuOTM2OTgwM3w4MC4xNzQwMjAy'

### ****Methods****

#### start\_requests()

* **Description**: Initiates the spider by requesting the initial URL to get to the search results page.
* **Callback**: pagination

#### pagination(response)

* **Description**: Handles pagination by generating URLs for each search results page.
* **Parameters**:
  + response: The Scrapy response object.
* **Pages**: The number of pages is set to 50 for testing purposes.
* **Callback**: parse
* **Note**: The break statement is used to restrict execution to the first page for testing.

#### process\_item(val, response)

* **Description**: Processes and extracts data from individual vehicle entries.
* **Parameters**:
  + val: The vehicle data dictionary.
  + response: The Scrapy response object.
* **Extracted Fields**: See the table below.
* **Error Handling**: Exceptions are caught and silently handled.

#### parse(response)

* **Description**: Parses the search results page and extracts vehicle data from a JSON object embedded in the script tag.
* **Parameters**:
  + response: The Scrapy response object.
* **Data Extraction**:
  + **JSON Parsing**: Extracts and cleans JSON data from the script tag.
  + **Data Extraction**: Yields items using process\_item for each vehicle entry.

**Usage:**

* Run this spider using the Scrapy command:

“scrapy crawl JATO\_mobile\_de”

**Variables Declared Inside item[ ]:**

| **Field Name** |
| --- |
| Make |
| Model |
| Derivative |
| Derivative\_Translated\_English |
| Number\_Of\_Doors |
| Body\_Type |
| Power\_Train |
| Version\_Name |
| Version\_Name\_Translated\_English |
| Currency |
| Price |
| Delivery\_Costs\_Retail |
| Country |
| Research\_Date |
| Customer\_Type |
| Monthly\_Payment\_Type |
| Product\_Name |
| Monthly\_Payment\_Provider\_Name |
| Monthly\_Payment\_Provider\_Type |
| Interest\_Rate\_Apr |
| Interest\_Rate\_Nominal |
| Contract\_Duration\_Months |
| Yearly\_Mileage\_Km |
| Deposit\_Retail |
| Of\_Monthly\_Instalments |
| Regular\_Monthly\_Instalment\_Amount\_Retail |
| Additional\_Fees\_Retail |
| Data\_Source |
| Web\_Source\_Url |
| Other\_Sources |
| Vehicle\_Price\_Reference |
| Insurance |
| Insurance\_Description |
| Reimbursement\_Mileage\_Retail |

**JATO\_SelectaLease**

### Overview:

The LeaseScrapSpider is a Scrapy spider designed for web scraping vehicle leasing data from the selectalease.co.uk website. It extracts detailed information about vehicle leases, including service and maintenance details, vehicle specifications, and financial information.

### Spider Details:

* **Name**: JATO\_selectalease
* **Allowed Domains**: www.selectalease.co.uk
* **Base URL**: https://www.selectalease.co.uk

### Cookies and Headers:

* **Cookies**: Used for session management and maintaining user-specific data.
* **Headers**: Custom headers are set to simulate a real browser request, ensuring better compatibility with the target website.

### Methods:

#### start\_requests():

* **Purpose**: Initiates the scraping process by sending a request to the service and maintenance page.
* **Callback**: services\_and\_maintenance

#### pagination(response):

* **Purpose**: Handles pagination on the vehicle listing page. It constructs URLs for subsequent pages and yields requests for each page.
* **Error Handling**: Logs errors if pagination fails.

#### extract\_financial\_data(script\_content, var\_name):

* **Purpose**: Extracts financial data from JavaScript variables embedded in the page's script tags.
* **Pattern**: Uses regular expressions to locate and extract variable data.
* **Returns**: List of extracted financial data or None if the variable is not found.

#### services\_and\_maintenance(response):

* **Purpose**: Parses the service and maintenance information from the service page.
* **Fields Extracted**: Includes repair, service, tyres, road assistance, and courtesy car details.
* **Callback**: pagination, passing the extracted service data for further vehicle scraping.

#### parse\_url(response):

* **Purpose**: Extracts URLs for individual vehicle listings from the pagination page.
* **Callback**: parse, to handle detailed extraction of vehicle information.

#### parse(response):

* **Purpose**: Parses detailed information from individual vehicle pages, including vehicle specifications, leasing information, and financial details.
* **Fields Extracted**:
  + **Vehicle Information**: Make, Model, Body Type, Derivative, Trim, Number of Doors.
  + **Country and Currency**: Set to Germany and Euro respectively.
  + **Leasing Information**: Customer Type, Product Name, Price, Delivery Costs, etc.
  + **Financial Data**: Extracted using extract\_financial\_data method, including contract duration, monthly payments, and maintenance costs.
* **Data Processing**: Generates multiple items for different leasing scenarios based on contract duration, mileage, and customer type.

### Error Handling:

* Each method includes try-except blocks to catch and log errors, ensuring that issues in data extraction or processing do not halt the entire scraping operation.

**Usage:**

* Run this spider using the Scrapy command:

“scrapy crawl JATO\_SelectaLease ”

**Variables Declared Inside item[ ]:**

| **Variable** |
| --- |
| Repair |
| Repair\_Description |
| Service |
| Service\_Description |
| Maintenance\_Description |
| Tyres |
| Tyres\_Description |
| Road\_Assistance |
| Road\_Assistance\_Description |
| Courtesy\_Car |
| Courtesy\_Car\_Description |
| Make |
| Model |
| Body\_Type |
| Derivative |
| Derivative\_Translated\_English |
| Trim |
| Built\_Option\_Pack |
| Number\_Of\_Doors |
| Currency |
| Country |
| Research\_Date |
| Data\_Source |
| Other\_Sources |
| Web\_Source\_Url |
| Published\_End\_Date |
| Flexible\_Early\_Cancellation\_Possible |
| Road\_Tax |
| Customer\_Type |
| Product\_Name |
| Price |
| Delivery\_Costs\_Retail |
| Advertised\_Price\_Point\_Monthly\_Payment |
| Version\_Name |
| Version\_Name\_Translated\_English |
| Contract\_Duration\_Months |
| Of\_Monthly\_Payments\_In\_Advance |
| Of\_Monthly\_Instalments |
| Yearly\_Mileage\_Miles |
| Regular\_Monthly\_Instalment\_Amount\_Retail |
| First\_Monthly\_Instalment\_Amount\_Retail |
| Additional\_Fees\_Retail |
| Regular\_Monthly\_Instalment\_Amount\_Base |
| First\_Monthly\_Instalment\_Amount\_Base |
| Additional\_Fees\_Base |

**VehicleItem - Items.py**

### Overview

The VehicleItem class is a Scrapy item model representing detailed information about vehicles, their pricing, leasing details, customer information, and other associated data. This class is intended for use in a Scrapy spider to scrape data from automotive websites, particularly for leasing or sales purposes.

### Fields and Categories

The VehicleItem class is divided into several categories, each containing specific fields relevant to the category. Below is a detailed breakdown of these categories and their fields:

1. **Vehicle Information**
   1. **Make:** Brand of the vehicle (e.g., "Toyota").
   2. **Model:** Model of the vehicle (e.g., "Corolla").
   3. **Trim:** Specific trim level of the vehicle (e.g., "LE").
   4. **Derivative:** Derivative model of the vehicle (e.g., "Sedan").
   5. **Derivative\_Translated\_English:** English translation of the derivative name.
   6. **Number\_Of\_Doors:** Number of doors in the vehicle.
   7. **Body\_Type:** Type of the vehicle body (e.g., "SUV", "Sedan").
   8. **Power\_Train:** Information about the vehicle's powertrain (e.g., "Hybrid").
   9. **Version\_Name:** Specific version name of the vehicle.
   10. **Version\_Name\_Translated\_English:** English translation of the version name.
   11. **Manufacturer\_S\_Code:** Manufacturer's internal code for the vehicle.
   12. **Configurator\_Model\_Year:** Model year according to the configurator.
   13. **Built\_Option\_Pack:** Details on any built option packs available.
   14. **Uid:** Unique identifier for the vehicle.
   15. **Data\_Date:** Date when the data was collected.
   16. **Conclude\_Date:** Date when data collection concludes.
   17. **Version\_Availability:** Availability status of the vehicle version.
2. **Pricing and Costs**
   1. **Currency:** Currency in which the vehicle price is listed (e.g., "USD").
   2. **Price:** Total price of the vehicle.
   3. **Delivery\_Costs\_Retail:** Retail delivery costs associated with the vehicle.
   4. **Cost\_To\_Return\_The\_Vehicle\_Retail:** Cost to return the vehicle at the end of the lease or term.
   5. **Total\_Mandatory\_Packs\_Amount:** Total amount for all mandatory option packs.
3. **Dates**
   1. **Country:** Country where the vehicle is listed or sold.
   2. **Research\_Date:** Date on which the research was conducted.
   3. **Start\_Date:** Start date of the offer or pricing.
   4. **End\_Date:** End date of the offer or pricing.
4. **Customer and Payment Information**
   1. **Customer\_Type:** Type of customer (e.g., "Private", "Corporate").
   2. **Monthly\_Payment\_Type:** Type of monthly payment (e.g., "Fixed", "Variable").
   3. **Product\_Name:** Name of the leasing or financing product.
   4. **Product\_Description:** Description of the product offering.
   5. **Monthly\_Payment\_Provider\_Name:** Name of the payment provider.
   6. **Monthly\_Payment\_Provider\_Type:** Type of payment provider (e.g., "Bank", "Captive").
   7. **Interest\_Rate\_Apr:** Annual Percentage Rate (APR) for the financing.
   8. **Interest\_Rate\_Nominal:** Nominal interest rate.
   9. **Contract\_Duration\_Months:** Duration of the contract in months.
   10. **Yearly\_Mileage\_Km:** Allowed yearly mileage in kilometers.
   11. **Yearly\_Mileage\_Miles:** Allowed yearly mileage in miles.
   12. **Total\_Contract\_Mileage\_Km:** Total contract mileage in kilometers.
   13. **Total\_Contract\_Mileage\_Miles:** Total contract mileage in miles.
   14. **Deposit\_Percentage\_Of\_Price:** Percentage of the deposit based on the price.
   15. **Deposit\_Retail:** Retail amount for the deposit.
   16. **Deposit\_Base:** Base amount for the deposit.
   17. **Final\_Payment\_Base:** Base amount for the final payment.
5. **Monthly Payments**
   1. **Of\_Monthly\_Payments\_In\_Advance:** Number of monthly payments in advance.
   2. **Of\_Monthly\_Instalments:** Total number of monthly instalments.
   3. **First\_Monthly\_Instalment\_Amount\_Retail:** Retail amount of the first monthly instalment.
   4. **First\_Monthly\_Instalment\_Amount\_Base:** Base amount of the first monthly instalment.
   5. **Regular\_Monthly\_Instalment\_Amount\_Retail:** Retail amount of the regular monthly instalment.
   6. **Regular\_Monthly\_Instalment\_Amount\_Base:** Base amount of the regular monthly instalment.
   7. **Additional\_Fees\_Retail:** Retail amount of any additional fees.
   8. **Additional\_Fees\_Base:** Base amount of any additional fees.
   9. **Final\_Payment\_Type:** Type of final payment (e.g., "Balloon").
   10. **Final\_Payment\_Percentage\_Of\_Price:** Percentage of final payment based on the price.
   11. **Final\_Payment\_Retail:** Retail amount for the final payment.
6. **Data Sources**
   1. **Data\_Source:** Source of the data (e.g., "API", "Website").
   2. **Web\_Source\_Url:** URL of the web source where the data was obtained.
7. **Advertised Pricing and Published Dates**
   1. **Advertised\_Price\_Point\_Monthly\_Payment:** Advertised monthly payment amount.
   2. **Published\_Start\_Date:** Start date for the published pricing.
   3. **Published\_End\_Date:** End date for the published pricing.
8. **Financial Configurators**
   1. **Oem\_Web\_Financial\_Configurator\_Source:** Source of the OEM web financial configurator.
   2. **Captive\_Financial\_Configurator\_Source:** Source of the captive financial configurator.
   3. **Non\_Captive\_Financial\_Configurator\_Source:** Source of the non-captive financial configurator.
   4. **Bank\_Financial\_Configurator\_Source:** Source of the bank financial configurator.
9. **Sourced Quotes and Sources**
   1. **Sourced\_Dealer\_Quote:** Dealer quote obtained.
   2. **Jato\_Internal\_Source:** Internal JATO source.
   3. **Other\_Sources:** Other miscellaneous sources.
10. **Vehicle Price Reference**
    1. **Vehicle\_Price\_Reference:** Reference point for the vehicle price.
11. **Dealer Contributions and Discounts**
    1. Fields related to mandatory dealer contributions, OEM discounts, and government contributions, all in percentage or retail amounts.
12. **Downpayment Allowance and Rebate**
    1. Fields related to downpayment allowances and captive rebates in percentage or retail amounts.
13. **Financial Configurator Sources**
    1. Fields such as **Free\_Supply\_Amount**, **Residual\_Value\_Percentage\_Of\_Price**, **Residual\_Value\_Retail**, **Residual\_Value\_Base**, **Lease\_Factor\_Index**.
14. **Insurance and Supply**
    1. Fields for various insurance types and amounts, along with financed amounts.
15. **Unique Identifiers and Status**
    1. Unique identifiers, status changes, and backend data fields.
16. **Mileage and Costs**
    1. Fields for mileage costs, both excess and reimbursement types
17. **Courtesy Car**
    1. Fields for courtesy car details such as description, duration, and amount.
18. **Fuel and Electric Supply**
    1. Fields related to fuel and electric supply, their descriptions, and quantities.
19. **Road Assistance**
    1. Fields for road assistance, including duration, distance, and amount.
20. **Cancellation and Termination Costs**
    1. Fields related to early contract termination costs, percentages, and descriptions
21. **Tax and Tyres**
    1. Fields for road tax, tyres, tyre descriptions, and associated costs.
22. **Maintenance and Repair**
    1. Fields for maintenance and repair details, including description, duration, distance, and amounts.
23. **Service**
    1. Fields for service details, including descriptions, duration, distance, and amounts.

**Middleware.py**

### Overview

Middleware in Scrapy serves as a framework of hooks into Scrapy's request/response processing. Middleware can be applied to spiders (Spider Middleware) or the downloaders (Downloader Middleware). They are designed to process requests before they are sent to the downloader, or responses before they are sent to the spider. This document analyzes the provided code for custom spider and downloader middleware in Scrapy.

### Spider Middleware

The spider middleware is responsible for handling the communication between the spider and the Scrapy engine. It can modify requests, process responses, or handle exceptions that occur during the spider's operation.

#### Key Methods and Their Functions

1. from\_crawler(cls, crawler):

* A class method used by Scrapy to create the middleware instance.
* Connects the spider\_opened signal to be triggered when the spider is opened.

1. process\_spider\_input(self, response, spider):

* Called for each response that goes through the spider middleware and is passed to the spider.
* Should return None or raise an exception. If it returns None, Scrapy continues processing the response; otherwise, it stops.

1. process\_spider\_output(self, response, result, spider):

* Called with the results returned from the spider after processing a response.
* Must return an iterable of Request or item objects that will be further processed.

1. process\_spider\_exception(self, response, exception, spider):

* Called when an exception is raised either by the spider or by the process\_spider\_input() method.
* Should return None or an iterable of Request or item objects.

1. process\_start\_requests(self, start\_requests, spider):

* Called with the initial requests of the spider.
* Must return only Request objects to be processed further by the Scrapy engine.

1. spider\_opened(self, spider):

* Handles the spider\_opened signal and logs a message when the spider is opened.

#### Use Cases

* **Custom Handling of Responses**: Modify or filter out unwanted responses before they reach the spider.
* **Exception Handling**: Manage exceptions that occur during spider processing, which helps in logging errors or retrying requests.
* **Processing Start Requests**: Alter or add custom headers or parameters to initial requests sent by the spider.

### Downloader Middleware

The downloader middleware is designed to process requests and responses that pass between the Scrapy engine and the downloader.

#### Key Methods and Their Functions

1. from\_crawler(cls, crawler):
   * A class method similar to the spider middleware, used by Scrapy to create an instance of the downloader middleware.
   * Connects the spider\_opened signal to be triggered when the spider is opened.
2. process\_request(self, request, spider):
   * Called for each request going through the downloader middleware.
   * Must either:
     + Return None to continue processing the request.
     + Return a Response object.
     + Return a Request object.
     + Raise IgnoreRequest to call process\_exception() methods of installed downloader middleware.
3. process\_response(self, request, response, spider):
   * Called with the response returned from the downloader.
   * Must return:
     + A Response object to continue processing.
     + A Request object to retry the request.
     + Raise IgnoreRequest to stop processing.
4. process\_exception(self, request, exception, spider):
   * Called when a download handler or a process\_request() method raises an exception.
   * Must return:
     + None to continue processing the exception.
     + A Response or Request object to stop further processing.
5. spider\_opened(self, spider):
   * Handles the spider\_opened signal and logs a message when the spider is opened.

**Pipelines.py**

### Overview

The item pipeline is a mechanism in Scrapy that processes the scraped data after it is extracted from web pages by spiders. The pipeline typically handles tasks like data cleansing, validation, transformation, storage, and exporting to different file formats or databases. This document provides a technical analysis of the ScrapyProjectPipeline class, explaining its structure and functionality for handling the extracted data and uploading it to Azure Blob Storage.

### Pipeline Initialization

1. \_\_init\_\_(self):
   1. The constructor initializes two key attributes:

* self.items: A dictionary to store scraped items for each spider. This helps to keep track of the data separately for each spider.

1. self.blob\_service\_client and self.container\_client:

* BlobServiceClient is used to connect to Azure Blob Storage using the provided connection string.
* ContainerClient is created for the specific Azure Blob Storage container ("rawdata"), where data will be uploaded.

**Security Note**: The connection string used for Azure Blob Storage should be secured and managed through environment variables or secret management services like Azure Key Vault.

### Key Methods

1. open\_spider(self, spider):

* Called when a spider is opened.
* Initializes an empty list in the self.items dictionary for the given spider name to store the scraped items.

1. close\_spider(self, spider):

* Called when a spider is closed.
* Converts the list of scraped items into a JSON format and saves it to a file locally.
* Attempts to upload the JSON data to Azure Blob Storage directly.
* Logs any errors that occur during the process using Python's logging module.

1. **Data Extraction Methods**:

* These methods are used to clean, transform, or extract specific parts of the scraped data:

1. extract\_derivative(self, value):

* Cleans and extracts a derivative value from the string. Removes content within square brackets ([...]) if present.

1. extract\_build\_option(self, value):

* Extracts build options enclosed in square brackets ([...]) and concatenates them into a single string.

1. extract\_doors(self, value):

* Extracts the number of doors (e.g., 3dr, 5Dr) from the string using a regex pattern.

1. extract\_tyre(self, value):

* Extracts tire descriptions listed in HTML list tags (<li>) and returns them as a comma-separated string.

1. extract\_road(self, value):

* Similar to extract\_tyre, it extracts the road assistance descriptions listed in HTML list tags.

1. extract\_courtesy\_car(self, value):

* Extracts the fourth item in a list of values enclosed in HTML list tags, indicating the courtesy car description.

1. extract\_repair(self, value):

* Extracts specific parts (fifth and eighth items) of a list of repair descriptions enclosed in HTML list tags.

1. extract\_provider\_name(self, value):

* Extracts the provider name from a string up to the first <br> tag.

1. process\_item(self, item, spider):
2. The main method responsible for processing each scraped item.
3. Calls the relevant extraction methods to clean or transform item fields based on their content.
4. Appends the processed item to the list for the specific spider in self.items.
5. Returns the processed item for further handling in the pipeline.

### Use Cases

* **Data Cleaning and Transformation**: The pipeline cleans raw data extracted from web pages, ensuring it is in the required format and structure before being stored or exported.
* **Data Export to Azure Blob Storage**: The pipeline exports the scraped data to Azure Blob Storage in JSON format, allowing centralized and scalable storage.
* **Error Handling**: Logs any errors that occur during data processing or upload to Azure Blob Storage, ensuring issues are identified and managed promptly.

**Settings.py**

**BOT\_NAME**: "scrapy\_project"

* · The name of the Scrapy bot used for logging and identification.

· **SPIDER\_MODULES**: ["scrapy\_project.spiders"]

* · Specifies where Scrapy will look for spider definitions.

· **NEWSPIDER\_MODULE**: "scrapy\_project.spiders"

* · Defines where new spiders will be created by default.

· **ROBOTSTXT\_OBEY**: False

* · Disables following robots.txt rules, allowing scraping even if a site disallows it. Be cautious as this can violate a site's terms of service.

· **ITEM\_PIPELINES**:

· {

"scrapy\_project.pipelines.ScrapyProjectPipeline": 300,

}

* · Enables the custom pipeline ScrapyProjectPipeline for processing scraped items. The number 300 indicates the order of execution; lower numbers are processed first.

· **REQUEST\_FINGERPRINTER\_IMPLEMENTATION**: "2.7"

* · Uses Scrapy's request fingerprinting compatible with version 2.7, helping to identify and avoid duplicate requests.

· **TWISTED\_REACTOR**: "twisted.internet.asyncioreactor.AsyncioSelectorReactor"

* · Configures Scrapy to use the asyncio event loop, which can improve performance for handling many simultaneous requests.

· **FEED\_EXPORT\_ENCODING**: "utf-8"

* · Sets the encoding for exported data files to UTF-8, ensuring compatibility with most text formats.

**Azure Function - Scrapy Spiders(scheduling)**

### Overview

The Azure Function in question is designed to execute multiple Scrapy spiders on a scheduled basis. This function is triggered by a timer and is intended to manage the execution of various spiders in parallel. Key components include logging, error handling, and the execution of an external function to run the spiders.

### Key Components

#### 1. ****Trigger Type****

* **Trigger**: Timer Trigger
* **Schedule**: The function is configured to run according to a CRON-like schedule, which specifies the exact time and frequency at which the function should execute. The function is set to execute on specific days and times as defined by the schedule expression.

#### 2. ****Logging****

* **Purpose**: Provides insights into the execution flow and helps in debugging.
* **Implementation**:
  + Logs a message when the timer is past due.
  + Logs a message when the function is executed.
  + Logs a message upon successful execution of the spiders.
  + Logs an error message if an exception occurs during the execution of the spiders.

#### 3. ****Error Handling****

* **Purpose**: Ensures that any issues encountered during the execution of the spiders are captured and logged.
* **Implementation**:
  + Wraps the spider execution logic in a try-except block.
  + Logs an error message if an exception is raised, providing information about the nature of the error.

#### 4. ****Spider Execution****

* **Function**: Uses a custom function run\_spiders to execute a list of Scrapy spiders.
* **Parameters**:
  + spider\_names: A list of spider names that need to be executed.
  + parallel: Indicates whether the spiders should be run in parallel.

### Execution Flow

1. **Timer Trigger Activation**:
   * The function is activated based on the defined timer schedule.
2. **Check Timer Status**:
   * Logs a message if the timer is past due.
3. **Log Execution Start**:
   * Logs a message indicating the start of the function execution.
4. **Run Spiders**:
   * Calls the run\_spiders function with the specified list of spider names and the parallel execution flag.
   * Logs a success message if the spiders are executed successfully.
5. **Exception Handling**:
   * If an error occurs during spider execution, it is caught and logged.
6. **Log Execution Completion**:
   * Logs a message indicating the completion of the function execution.

**Azure Function - DataProcessing**

### Overview

The script defines an Azure Function that triggers when a new blob is uploaded to the rawdata container in Azure Blob Storage. The function reads the blob content, processes the data by cleaning and transforming it, and then saves the processed data back to the processeddata container. It uses Azure Blob Storage for data storage and pandas for data manipulation.

### Key Components

#### 1. ****Imports****

* azure.functions as func: Provides classes and methods to define and work with Azure Functions.
* logging: Used for logging information, warnings, and errors.
* clean\_data: A function from the Data\_processing.data\_cleaning module for data cleaning.
* transform\_data: A function from the Data\_processing.data\_transformation module for data transformation.
* azure.storage.blob: Provides access to Azure Blob Storage services.
* pandas as pd: The pandas library for data manipulation and analysis.
* datetime.date: Provides methods to work with dates.
* json: Used to parse JSON data.

#### 2. ****Azure Function Configuration****

* **Function Name**: data\_processing\_trigger
* **Trigger**: Blob Trigger
  + **Path**: rawdata/{name}
  + **Connection**: AzureWebJobsStorage

#### 3. ****Function Definition****

##### blob\_trigger\_dp(myblob: func.InputStream)

* **Purpose**: Processes a blob file when triggered by new uploads to the rawdata container.
* **Parameters**:
  + myblob (func.InputStream): The blob that triggered the function.
* **Logging**:
  + Logs the name and size of the processed blob.
* **Process**:
  1. **Read Blob**:
     + Reads the JSON data from the blob.
     + Parses the JSON data into a pandas DataFrame.
  2. **Data Cleaning**:
     + Calls the clean\_data function to clean the DataFrame.
  3. **Data Transformation**:
     + Calls the transform\_data function to transform the DataFrame.
  4. **Create Folder Structure**:
     + Creates a folder name based on the current date.
  5. **Generate Blob Name**:
     + Modifies the original blob name to reflect the new processed folder and adds the date folder.
  6. **Convert DataFrame to JSON**:
     + Converts the cleaned and transformed DataFrame to a JSON format.
  7. **Upload Processed Data**:
     + Uploads the processed JSON data to the processeddata container in Azure Blob Storage.
     + Logs success or failure of the upload operation.
* **Error Handling**:
  + Logs an error message if an exception occurs during processing.

**Execution Flow:**

### ****Blob Trigger Activation****:

* + The function is triggered when a new blob is uploaded to the rawdata container.

1. **Data Reading and Conversion**:
   * Reads the blob content and converts it from JSON to a pandas DataFrame.
2. **Data Processing**:
   * Cleans the data using clean\_data.
   * Transforms the data using transform\_data.
3. **Data Storage**:
   * Creates a folder structure and generates a new blob name.
   * Converts the DataFrame to JSON and uploads it to the processeddata container.
4. **Logging**:
   * Logs details about the processing, including errors and successful uploads.

**Azure Function - Data Processing(Data Cleaning)**

### Overview

The provided script is designed for cleaning and preprocessing data within a pandas DataFrame. It focuses on text and currency value cleaning and handles missing values, duplicates, and index resetting. The script utilizes custom functions and imports for cleaning and field management.

### Key Components

#### 1. ****Imports****

* pandas as pd: The pandas library is used for data manipulation and analysis.
* Field\_List: Imported from field\_list, this class is presumably used to manage field-related configurations.
* re: Python’s regular expression library, used for text pattern matching and replacements.

#### 2. ****Initialization****

* expected\_fields**,** int\_fields**,** date\_fields: Variables initialized from the Field\_List class, representing different types of fields in the dataset. These variables are likely used to handle various field-specific operations.

#### 3. ****Functions****

##### clean\_text(text)

* **Purpose**: Cleans text data by removing unwanted characters and formatting.
* **Parameters**:
  + text (str): The text string to be cleaned.
* **Returns**:
  + A cleaned string with non-ASCII characters, extra spaces, and special characters removed.
* **Process**:
  + Uses regular expressions and string replacement methods to:
    - Remove non-ASCII characters.
    - Replace underscores, commas, and other special characters with spaces.
    - Remove trailing commas.
    - Replace multiple spaces and newlines with a single space.
    - Strip leading and trailing spaces.

##### clean\_currency(value)

* **Purpose**: Cleans and converts currency values to numeric format.
* **Parameters**:
  + value (str): The currency value as a string.
* **Returns**:
  + A float representing the numeric value or 'Not Available' if conversion fails.
* **Process**:
  + Checks if the value is empty or 'Not Available'.
  + Removes non-numeric characters except decimal separators and negative signs.
  + Replaces decimal commas with periods for standard numeric formatting.
  + Handles cases where decimal separators and thousands separators are mixed.

##### clean\_data(df)

* **Purpose**: Cleans a pandas DataFrame by applying text and currency cleaning functions and handling missing values.
* **Parameters**:
  + df (pandas.DataFrame): The DataFrame to be cleaned.
* **Returns**:
  + A cleaned DataFrame with text and currency fields processed, missing values handled, and duplicates removed.
* **Process**:
  + Applies clean\_text to all object (string) columns.
  + Applies clean\_currency to fields related to costs, as specified by Field\_List.
  + Removes rows where all columns are empty.
  + Fills missing values with 'Not Available'.
  + Drops duplicate rows.
  + Resets the DataFrame index.

### Execution Flow

### ****Initialization****:

* + Imports necessary libraries and initializes field lists for cleaning.

1. **Text Cleaning**:
   * Applies clean\_text to all object columns in the DataFrame to standardize text data.
2. **Currency Cleaning**:
   * Applies clean\_currency to columns identified as cost-related.
3. **Data Handling**:
   * Drops rows where all values are missing.
   * Fills remaining missing values with 'Not Available'.
   * Removes duplicate rows.
   * Resets the DataFrame index to ensure a clean, continuous index.

**Azure Function - Data Processing(Data Transformation)**

### Overview

The provided script performs data transformation tasks on a pandas DataFrame. It includes functions for calculating new fields based on existing data, converting values to integers, formatting dates, and ensuring that all expected fields are present. The script utilizes custom functions and imports for data manipulation and formatting.

### Key Components

#### 1. ****Imports****

* pandas as pd: The pandas library is used for data manipulation and analysis.
* Field\_List: Imported from field\_list, this class is used to manage field-related configurations and expected fields.
* re: Python’s regular expression library, used for pattern matching and string replacement.

#### 2. ****Functions****

##### calculate\_fields(df)

* **Purpose**: Calculates new fields based on existing data and conversion factors, and updates the DataFrame.
* **Parameters**:
  + df (pandas.DataFrame): The DataFrame containing the data to be processed.
* **Returns**:
  + A DataFrame with additional calculated fields and updated values.
* **Process**:
  + **Conversion Factors**: Defines conversion factors for kilometers to miles and vice versa.
  + **Field Conversion**: Converts relevant columns to numeric types, coercing errors to NaN.
  + **Mileage Calculations**:
    - Converts Yearly\_Mileage\_Km to Yearly\_Mileage\_Miles if available.
    - Converts Yearly\_Mileage\_Miles to Yearly\_Mileage\_Km if available.
  + **Contract Mileage Calculations**:
    - Calculates Total\_Contract\_Mileage\_Miles and Total\_Contract\_Mileage\_Km based on yearly mileage and contract duration.
  + **Error Handling**: Returns the DataFrame unchanged if an exception occurs during processing.

##### safe\_convert\_to\_int(value)

* **Purpose**: Safely converts values to integers.
* **Parameters**:
  + value (any): The value to be converted.
* **Returns**:
  + An integer if conversion is successful; otherwise, None.
* **Process**:
  + Checks if the value is None or 'Not Available' and returns None.
  + Attempts to convert the value to an integer, handling ValueError and TypeError exceptions.

##### dateFormator(date)

* **Purpose**: Formats date strings into a standardized format.
* **Parameters**:
  + date (str): The date string to be formatted.
* **Returns**:
  + A formatted date string or the original string if formatting fails.
* **Process**:
  + Replaces various date delimiters with hyphens.
  + Reverses the date components to standardize the format (DD-MM-YYYY).
  + Returns the formatted date or the original string if no match is found or if an exception occurs.

##### transform\_data(df)

* **Purpose**: Transforms and processes data in the DataFrame according to predefined field expectations.
* **Parameters**:
  + df (pandas.DataFrame): The DataFrame to be transformed.
* **Returns**:
  + A DataFrame with expected fields, formatted values, and calculated fields.
* **Process**:
  + **Initialization**: Creates a new DataFrame with columns defined in expected\_fields and fills it with 'Not Available'.
  + **Field Matching**: Copies columns from the original DataFrame to the new DataFrame if they are in expected\_fields.
  + **Missing Values**: Fills empty values with 'Not Available'.
  + **Date Formatting**: Applies dateFormator to date fields and converts formatted dates to datetime objects.
  + **Field Calculations**: Applies calculate\_fields to compute additional fields based on existing data.

### Execution Flow

### ****Initialization****:

* + Imports necessary libraries and initializes field lists.

1. **Field and Value Processing**:
   * Initializes a new DataFrame with expected fields.
   * Copies relevant data from the original DataFrame to the new DataFrame.
   * Fills missing values and processes date fields.
2. **Calculations**:
   * Applies calculate\_fields to add and update calculated fields.
3. **Output**:
   * Returns the transformed DataFrame with cleaned and calculated data.

**Amount Prediction**

#### Overview

This documentation details the steps taken to build a machine learning model that predicts the **Regular Monthly Installment Amount** for car leasing using a Random Forest Regressor. The script loads a dataset, preprocesses it, splits it into training and testing sets, builds a predictive model, and provides an interactive widget for making predictions. Additionally, the model's performance is evaluated through visualizations and statistical metrics.

#### Objective

The main objective of this script is to predict the monthly retail installment amount for car leases based on three primary factors:

1. **Price**: The total cost of the car.
2. **Contract Duration (Months)**: The length of the lease contract in months.
3. **Yearly Mileage (Km)**: The distance the car is expected to travel annually.

#### Data Handling

1. **Loading the Dataset**: The dataset is loaded from a CSV file using the Pandas library. It is crucial to replace the file path ('/content/output (1).csv') with the correct path to your local dataset. The dataset should contain relevant columns such as the price, contract duration, yearly mileage, and the target variable — the monthly installment amount.
2. **Feature and Target Variable Selection**: The script selects specific columns from the dataset as features (X) and the target variable (y). The features (X) used for prediction are:
   1. **Price**: Indicates the car’s retail value.
   2. **Contract Duration (Months)**: Represents the term of the leasing agreement.
   3. **Yearly Mileage (Km)**: The maximum mileage allowed per year in the leasing contract.

The target variable (y) is:

* **Regular Monthly Installment Amount (Retail)**: The amount that a customer is expected to pay monthly.

**3. Data Splitting**: The dataset is divided into a training set (80%) and a testing set (20%) to train and evaluate the model's performance. This process ensures that the model is trained on one subset of the data and validated on another to gauge its predictive capabilities on unseen data.

#### Model Building and Training

1. **Choice of Model**: The script uses a **Random Forest Regressor**, a robust ensemble learning method well-suited for regression tasks. A Random Forest model consists of a collection of decision trees that work together to make predictions. It operates by building multiple decision trees during training and outputting the average prediction of the individual trees, thereby reducing variance and avoiding overfitting.
2. **Hyperparameters**: The script sets the following key hyperparameters for the Random Forest Regressor:

* n\_estimators=100: Specifies the number of decision trees to build in the forest. Increasing the number of trees generally improves model performance but also increases computational cost.
* random\_state=42: Ensures reproducibility of the results by fixing the seed for random number generation.

1. **Training the Model**: The model is trained using the training dataset. The fit method is called with the training data (X\_train and y\_train), allowing the model to learn patterns in the data that map the input features to the target variable.

#### Model Evaluation

1. **Prediction on Test Data**: After training, the model is used to predict the target variable on the test dataset (X\_test). The predicted values (y\_pred) are compared with the actual values (y\_test) to evaluate the model’s performance.
2. **Performance Metric – R² Score**: The model’s accuracy is evaluated using the **R² score (coefficient of determination)**. The R² score measures the proportion of variance in the dependent variable that is predictable from the independent variables. An R² score close to 1 indicates that the model can explain most of the variance in the target variable.

* A higher R² score (closer to 1) suggests a better model fit.
* A lower R² score (closer to 0 or negative) indicates that the model poorly fits the data.

#### Visualization

To visually assess the model's performance, a scatter plot is generated comparing the actual versus predicted values:

* **Scatter Plot of Actual vs. Predicted Values**:
  + A scatter plot is created to show the relationship between actual values (y\_test) and predicted values (y\_pred). Each point on the plot represents a test observation.
  + A diagonal reference line is drawn to indicate where the actual values equal the predicted values. Points closer to this line represent more accurate predictions, while points further away indicate larger prediction errors.

#### Interactive Prediction Interface:

1. **Purpose**: The script includes an interactive widget built with ipywidgets that allows users to input feature values (Price, Contract Duration, and Yearly Mileage) and obtain a predicted monthly installment amount without needing to rerun the entire script.
2. **Components**:
   * **Input Fields**:
     + **Price**: A numerical input for the retail price of the car.
     + **Contract Duration (Months)**: An integer input representing the leasing period.
     + **Yearly Mileage (Km)**: A numerical input for the annual mileage.
   * **Predict Button**: A button labeled "Predict" that triggers the prediction function when clicked.
   * **Output Area**: Displays the predicted monthly installment amount.
3. **Functionality**:
   * When the "Predict" button is clicked, the input values are retrieved, and the model predicts the monthly installment amount. The result is displayed in the output area, providing an interactive and user-friendly way to make predictions.

#### Usage Guidelines

* **Environment**: The script should be run in a Jupyter notebook or an environment that supports ipywidgets for the interactive widget functionality.
* **Dataset Preparation**: Ensure the dataset is correctly formatted and includes all necessary columns for features and the target variable.
* **Model Retraining**: If new data becomes available or if model performance needs improvement, consider retraining the model with updated data or fine-tuning the hyperparameters.