PROJECT DOCUMENTATION “Vehicle Catalog Web Scraping”

1. Project Overview

The “Vehicle Catalog Web Scraping” project aims to automate and ethically collect data about automobiles from publicly available web sources. By combining web scraping techniques—using BeautifulSoup for parsing static HTML and Playwright for handling dynamic JavaScript content—the project navigates websites containing information about car models (technical specifications, trims, pricing, etc.), extracts the data, and stores it in a MongoDB database with a standardized schema.

The final deliverables are:

- A cleaned and indexed MongoDB database with a consistent structure for each vehicle’s specifications.
- The ability to export data to JSON, CSV, or Parquet formats for further analysis (e.g., market research, BI visualizations, machine learning).
- Sample Jupyter notebooks demonstrating the entire workflow: scraping, data normalization, and generating a “final_output.json” file with aggregated data.

Why this project is useful:

- **Quick access to structured vehicle data**: Instead of manually browsing multiple auto sites, an analyst or developer can instantly obtain a complete, clean catalog.
- **Support for business decisions and market research**: The collected data can be used by dealerships, insurance companies, ride-sharing startups, or auto sales platforms to identify market trends, compare specifications, or evaluate sales performance.
- **Automation and reusability**: The scraping component is modular and can be extended to new sources/sites, so coverage increases as new scripts are added.
- **Educational value**: For anyone interested in data and web analytics, this project serves as a detailed case study by integrating Python, Playwright, BeautifulSoup, MongoDB, and Pandas within a single pipeline.

2. Development Environment and Recommended IDE

To work comfortably on this project, use the following tools:

Visual Studio Code (VSCode)

- **Why VSCode?**
    - Excellent Python support via the “Python” extension.
    - Dedicated Jupyter integration (“Jupyter” extension) so you can open, run, and edit .ipynb notebooks directly within the editor.
    - Robust debugging for Python code and Playwright scripts (you can launch test/execution scripts directly from the IDE).
    - Built-in Git integration for version control and collaboration.
- **Recommended extensions**:
    - **Python** (Microsoft): syntax highlighting, IntelliSense, debugging, linting (Pylint/Flake8).
    - **Jupyter** (Microsoft): view and run notebooks inside VSCode.
    - **Playwright Test for VSCode** (Microsoft): if you want to run Playwright tests directly within VSCode.
    - **MongoDB for VSCode** (MongoDB, Inc.): view and query your MongoDB data from the editor.
- **Specific settings** (create or include these in the repo’s .vscode folder if desired):
    - .vscode/settings.json to define python.pythonPath or default linter settings.
    - .vscode/launch.json for debugging, for example:

```
{
"version": "0.2.0",
"configurations": [
{
"name": "Python: Launch scraper",
"type": "python",
"request": "launch",
"program": "${workspaceFolder}/scripts/auto-data_scraping.ipynb",
"console": "integratedTerminal"
}
]
}
```

(Note: For interactive debugging in notebooks, configure the kernel using the Jupyter extension.)

Alternatives

- PyCharm Community/Professional
  - Very good for more complex Python projects. The Professional edition has built-in Jupyter support; otherwise use the DataSpell plugin.
- Jupyter Notebook or JupyterLab standalone
  - If you prefer not to use a full IDE, you can work only in Jupyter Notebook/Lab for the notebooks in the scripts/ folder. Install with pip install jupyterlab and run jupyter lab.

Additional configurations:

- **Playwright**: After installing the Python package playwright, run playwright install in your terminal to download browser binaries (Chromium, Firefox, WebKit).
- **MongoDB**: Ensure you have a local MongoDB instance (version 4.4+ recommended). You can use MongoDB Community Edition installed locally or run a Docker container:

bash

CopyEdit

```
docker run -d -p 27017:27017 --name mongodb mongo:latest
```

- **Environment variable for MongoDB URI**:
  In the project root, create a .env file (if it doesn't exist) and add:

bash

CopyEdit

```
MONGODB_URI=mongodb://localhost:27017/vehicle_catalog
```

The VSCode extension "Env File Support" or "DotENV" can help load these values automatically.

3. **Installation and Configuration**

Follow these steps to prepare the environment and have everything ready to run:

Clone or extract the ZIP

- If you downloaded the ZIP file, extract it into a folder named, for example, vehicle-scraping.
- In a terminal, navigate to the project root:

bash

CopyEdit

```
cd /path/to/vehicle-scraping/vehicle-scraping
```

Set up a Python virtual environment (recommended)

bash

CopyEdit

```
python3 -m venv venv
source venv/bin/activate     # on Linux/macOS
venv\Scripts\activate.bat    # on Windows
```

This isolates the project's dependencies.

Install required libraries

Because there is no explicit requirements.txt in the repo, install packages manually:

nginx

CopyEdit

```
pip install beautifulsoup4
pip install playwright
pip install pymongo
pip install pandas
pip install notebook
pip install python-dotenv
```

After installation, run:

nginx

CopyEdit

```
playwright install
```

This command downloads the browser engines (Chromium, Firefox, WebKit) needed for Playwright.

Configure the .env file

In the project root, create or edit the file named .env with the following content:

bash

CopyEdit

```
MONGODB_URI=mongodb://localhost:27017/vehicle_catalog
```

If you want to use a different port or database name, modify accordingly.

Start MongoDB

- **If installed locally**: start the MongoDB service (e.g., sudo service mongod start on Linux or using the Windows Service Manager).
- **If using Docker**:

sql

CopyEdit

```
docker start mongodb
```

(Assuming you previously created the container; otherwise run docker run ... as shown above.)

Verify the connection in Python (optional)

In a Python shell, test the connection:

python

CopyEdit

```
from pymongo import MongoClient

import os

from dotenv import load_dotenv


load_dotenv()
```

```
client = MongoClient(os.getenv("MONGODB_URI"))

db = client.get_database()

print("Connected to database:", db.name)
```

You should see “Connected to database: vehicle_catalog” printed.

4. Project Structure

Below is a description of the main directories and files in the repository:

kotlin

CopyEdit

```
vehicle-scraping/
├─ LICENSE
├─ Project_Notebook.ipynb
├─ README.md
├─ .git/                  # Git metadata and configuration
├─ data/
│  ├─ auto-data/
│  │  ├─ auto-data-model-links-table.csv
│  │  ├─ auto-data-spec-links-table.csv
│  │  ├─ auto-data-table-final.csv
│  │  ├─ auto-data-variants-links-table.csv
│  ├─ cars-data/
│  │  ├─ cars-data-model-links-table.csv
│  │  ├─ cars-data-spec-links-table.csv
│  │  ├─ cars-data-table.csv
│  │  ├─ cars-data-variants-links-table.csv
│  ├─ merged-data/
│  │  └─ combined_data(4).csv
```

```
├─ out/
│  └─ final_output.json
├─ scripts/
│  ├─ auto-data_scraping.ipynb
│  ├─ auto_data_scraping-final.ipynb
│  ├─ carsdata_scraping.ipynb
│  ├─ data_normalization.ipynb
└─ requirements.txt? (optional)
```

- **LICENSE**: The MIT license under which the project is distributed.
- **Project_Notebook.ipynb**: A Jupyter notebook providing a complete end-to-end demonstration of the pipeline: from initial link collection, through data normalization and unification, to writing the final_output.json file.
- **README.md**: This file (and the documentation you are copying) describes the purpose, installation, and overall structure.
- **data/**: Contains intermediate and final data files:
    - auto-data/ and cars-data/ are two separate data sources (likely two different websites) from which the initial list of models, specifications, and variants were scraped. Each CSV file holds tables of links to model detail pages (*-model-links-table.csv), specification pages (*-spec-links-table.csv), and the final unified table (*-table-final.csv).
    - merged-data/combined_data(4).csv represents a merging step combining data from both “auto-data” and “cars-data” sources (e.g., concatenation or join).
- **out/final_output.json**: The final pipeline output—a JSON collection ready for import into MongoDB or further analysis.
- **scripts/**: Contains Jupyter notebooks demonstrating the scraping process and data normalization:
    - auto-data_scraping.ipynb and auto_data_scraping-final.ipynb: Two versions (raw and final) of the script for extracting data from the “auto-data” website.
    - carsdata_scraping.ipynb: A similar notebook for the “cars-data” website.

- data_normalization.ipynb: Notebook detailing cleaning of CSV tables, schema unification, and generation of the final CSV files.

There is no requirements.txt included by default, but dependencies have been specified above.

Running the Scraping Pipeline

After following the installation steps and setting environment variables, proceed as follows:

Scraping for “auto-data”

- Open the notebook scripts/auto_data_scraping-final.ipynb in VSCode (or in JupyterLab).
- Ensure the Python kernel is using the venv environment.
- Run cells in order:
    1. Library imports and setting up output file paths.
    2. Fetch the list of model page links—this saves data/auto-data/auto-data-model-links-table.csv.
    3. Iterate over each model link to extract technical specifications and save to auto-data-spec-links-table.csv.
    4. For each variant of each model, collect data (price, engine, features) and save to auto-data-variants-links-table.csv.
    5. Generate the final CSV table (auto-data-table-final.csv) with a unified schema (standard columns: Brand, Model, Year, Engine, Power, Torque, Consumption, etc.).
- After completion, you will see new CSVs in data/auto-data/.

Scraping for “cars-data”

- Open scripts/carsdata_scraping.ipynb.
- Run cells sequentially. The final output will be data/cars-data/cars-data-table.csv plus auxiliary link tables.

Data Normalization and Merging

- Open scripts/data_normalization.ipynb.

  - This notebook takes the two *-table-final.csv files generated above and unifies them into a single Pandas DataFrame. It applies cleaning rules (removing duplicates, normalizing text, converting units, validating missing data).
  - At the end, it writes data/merged-data/combined_data(4).csv (the number reflects previous iterations). You may rename it to combined_data.csv if desired.

Final Export and MongoDB Import

  - Open Project_Notebook.ipynb. This notebook integrates the above steps but also includes logic to export directly into MongoDB and/or generate the JSON file.
  - Run all cells: it will read combined_data.csv and insert each record as a document in the vehicles collection in MongoDB.
  - Finally, run the cell that writes out/final_output.json. If you only need the JSON and not the database import, you can skip the insert step and use the generated file.

Example CLI-style workflow (if you want to automate without interactive notebooks, extract code into a .py script, though notebooks are recommended for clarity):

- Convert relevant notebook cells into a Python script like run_scraper.py, then run python run_scraper.py.

Database Structure and Data Model

After importing into MongoDB (see cells in Project_Notebook.ipynb), you will have a database with the following structure:

- **Database**: vehicle_catalog
- **Collection**: vehicles
- **Sample document**:

```
{
"_id": ObjectId("..."),
"brand": "Toyota",
"model": "Corolla",
"year": 2021,
"trim": "1.8 Hybrid",
"transmission": "CVT",
"fuel": "Hybrid",
"power_hp": 122,
"torque_nm": 142,
"consumption_l_100km": 3.9,
"co2_emission_g_km": 89,
"price_eur": 24000,
"source": "auto-data",
"scrape_date": "2025-05-28T14:32:00Z"
}
```

- Fields may vary depending on what each source provides, but the unified schema defines a minimum set of common columns.
- There is also a source field (e.g., “auto-data” or “cars-data”) and scrape_date for auditing purposes.

If you need to extend the schema (e.g., adding “optional_features” or “safety_rating”), update the data_normalization.ipynb and reimport to MongoDB.

Usage and Quick Examples

Accessing MongoDB with a CLI or GUI

- Using Mongo Shell:

sql

CopyEdit

```
mongo --uri "mongodb://localhost:27017/vehicle_catalog"

show collections;

db.vehicles.findOne();
```

- Using MongoDB Compass or the MongoDB extension for VSCode, you can view the contents of the vehicles collection, create indexes (e.g., on brand + model), and run complex queries (e.g., find({ "fuel": "Electric" })).

Retrieving JSON/CSV for analysis

- The file out/final_output.json is ready for loading into a BI tool (Power BI, Tableau) or for analysis in Pandas.
- Example in Pandas:

python

CopyEdit

```
import pandas as pd
df = pd.read_json("out/final_output.json", lines=True)  # if it’s line-delimited JSON
df.head()
```

  - To export to CSV, add a cell in the notebook:

python

CopyEdit

```
df.to_csv("out/final_output.csv", index=False)
```

Sample MongoDB queries

  - **Total number of vehicles per brand**:

js

CopyEdit

```
db.vehicles.aggregate([
  { $group: { _id: "$brand", total: { $sum: 1 } } },
  { $sort: { total: -1 } }
]);
```

  - **Hybrid vehicles with consumption < 4 L/100km**:

js

CopyEdit

```
db.vehicles.find({
  fuel: "Hybrid",
  consumption_l_100km: { $lt: 4 }
});
```

  - **Average price by year**:

js

CopyEdit

```
db.vehicles.aggregate([
 { $group: { _id: "$year", average_price: { $avg: "$price_eur" } } },
 { $sort: { _id: 1 } }
]);
```

Features and Key Characteristics

- **Ethical and Moderate Web Scraping**: Uses time delays and rate limiting in Playwright to avoid overwhelming target servers, and respects robots.txt (you can add a component later to validate polite crawling).
- **Modularity**:
  - Each source has its own notebook: “auto-data” and “cars-data” can be executed independently.
  - The normalization pipeline (data_normalization.ipynb) ingests all sources and produces a unified table.
- **Flexible Storage**:
  - MongoDB’s document model is flexible, so you can extend the schema with new fields without major code changes.
  - Export to JSON/CSV/Parquet allows integration with data analysis tools (Apache Spark, Pandas, Power BI, etc.).
- **Explanatory Jupyter Notebooks**:
  - Each notebook includes extensive comments so someone new to scraping can understand HTTP requests, HTML parsing, and data normalization.
- **Potential Future Extensions**:
  - Adding a CLI module (python run_scraper.py --source auto-data).
  - Integrating with a cloud scheduler (AWS Lambda, Google Cloud Functions) for periodic scraping.

Quick Start Tutorial
Goal: Get a JSON file with car data in under 5 minutes.

Prepare the environment (1 minute)

bash

CopyEdit

```
cd /path/to/vehicle-scraping/vehicle-scraping
python3 -m venv venv
source venv/bin/activate        # or the Windows equivalent
pip install beautifulsoup4 playwright pymongo pandas notebook python-dotenv
playwright install
```

Start MongoDB (if you want direct import)

bash

CopyEdit

```
docker run -d -p 27017:27017 --name mongodb mongo:latest
```

Create the .env file
In vehicle-scraping/vehicle-scraping/.env:

bash

CopyEdit

```
MONGODB_URI=mongodb://localhost:27017/vehicle_catalog
```

Run the main notebook (interactively)

bash

CopyEdit

```
jupyter lab
```

- Open Project_Notebook.ipynb.
- Run all cells from start to finish (scraping, normalization, Mongo import, and JSON output).

Result

- You will have a file out/final_output.json containing all vehicle documents.
- If MONGODB_URI is configured correctly, data will also be in the vehicles collection in MongoDB.

Testing
Although there is no traditional tests/ directory, testing can be done as follows:

Verify intermediate CSV files

- Open data/auto-data/auto-data-table-final.csv in a CSV editor (Excel, VSCode) and ensure:
  - Expected columns (e.g., brand, model, year, etc.) are present.
  - No rows are entirely empty.
- Do the same for data/cars-data/cars-data-table.csv.

Check merged data integrity

- Open data/merged-data/combined_data(4).csv:
  - Verify there are no duplicates (brand + model + trim).
  - Check that missing values are handled consistently (e.g., as NaN or empty string).
- You can run a quick Pandas script (in a Jupyter cell) to sanity-check:

python

CopyEdit

```
import pandas as pd

df = pd.read_csv("data/merged-data/combined_data(4).csv")

print("Total records:", len(df))

print("Duplicate count:", df.duplicated(subset=["brand","model","trim"]).sum())

print("Missing values per column:\n", df.isnull().sum())
```

MongoDB import verification

- Check that data was imported:

bash

CopyEdit

```
mongo --eval 'db.getSiblingDB("vehicle_catalog").vehicles.count()'
```

(If it returns a number > 0, import worked.)

- Run a few sample queries in Mongo Shell or Compass to confirm document structure.

Contributions

To contribute, follow these steps:

Fork & Clone

bash

CopyEdit

```
git clone https://github.com/username/vehicle-scraping.git

cd vehicle-scraping
```

1. Create a new branch

bash

CopyEdit

```
git checkout -b feature/add-new-source
```

2. Follow coding conventions
   - Python code follows PEP-8 standards.
   - Notebooks must include clear comments and titled cells.
   - If you add a new scraping module, mirror the structure in scripts/:
     - Provide a Jupyter notebook with numbered steps.
     - Save intermediate data under data/<source-name>/*.
     - Update data_normalization.ipynb to include the new source.
3. Open a Pull Request
   After finishing changes locally, run:

bash

CopyEdit

```
git add .

git commit -m "Add support for scraping site XYZ"
git push origin feature/add-new-source
```

Then open a Pull Request on GitHub against the main branch. Describe your additions and testing steps.

4. Code Review
   The maintainers will review, comment, and merge once everything meets quality standards.

5. License and Credits

- This project is published under the **MIT License** (see the LICENSE file).
- Major contributors can be listed under "Contributors" in the README.
- External dependencies and licenses:
  - **BeautifulSoup4** (MIT License) – for HTML parsing.
  - **Microsoft Playwright** (Apache 2.0) – for browser control in scraping.
  - **Pandas** (BSD-like) – for data processing and normalization.
  - **MongoDB** (Server Side Public License) – for document storage.

13. Resources and Useful Links

- **BeautifulSoup Documentation:**
  https://www.crummy.com/software/BeautifulSoup/bs4/doc/
- **Playwright Python Documentation:**
  https://playwright.dev/python
- **MongoDB Python (PyMongo) Tutorial:**
  https://pymongo.readthedocs.io/en/stable/
- **Pandas Documentation:**
  https://pandas.pydata.org/docs/
- **Jupyter Notebook / JupyterLab:**
  https://jupyter.org/
- **MongoDB Docker Image:**
  https://hub.docker.com/_/mongo
- **Best Practices for Web Scraping (Real Python):**
  https://realpython.com/python-web-scraping-practical-introduction/

Made by :

Chis Bogdan, Varga Sergiu, Faur Cristian, Ciurescu Raul