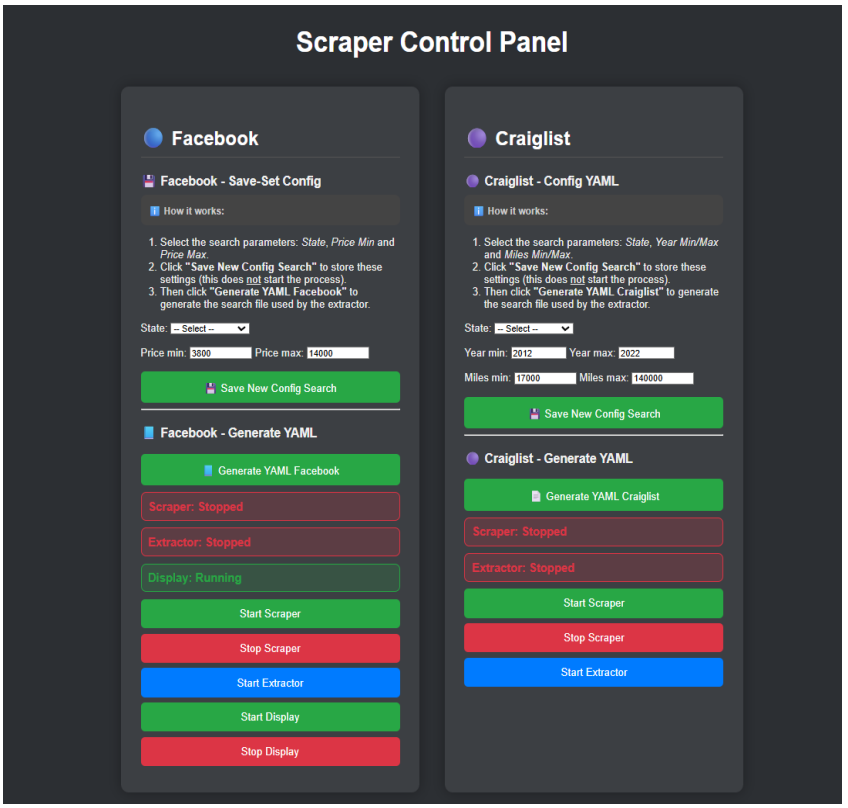
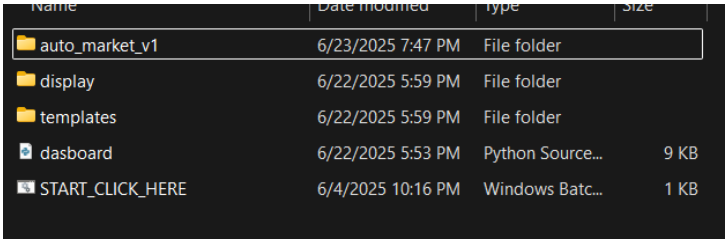
Project Presentation: Auto Market Scraper & Dashboard



What is this project?
It is an application created in Python that:
automatically searches car sale ads from Facebook Marketplace and Craigslist,
saves the results,
and displays them in an interactive control panel, where the user can filter, sort and analyze the data quickly.

What do you need to do to start the application?
Have Python installed (version 3.9+).
Click on the file START_CLICK_HERE.bat
That's it. The rest is handled by the application:
Automatically installs all necessary libraries.
Starts the server.
Automatically opens the browser with the control panel.

What does the application contain?
Structure:
auto_market_v1 – the part that extracts the data
display – the part that displays the data
templates – the graphical web interface
dashboard.py – the main server
START_CLICK_HERE.bat – the launcher file

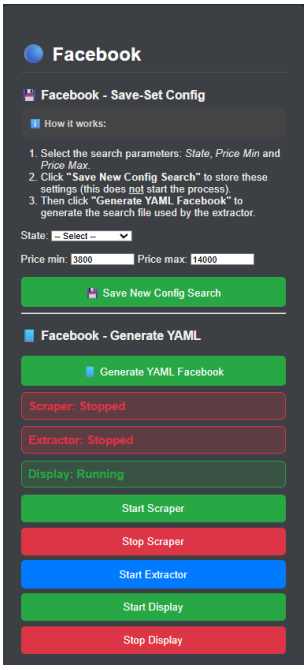


How does the control panel look and what does it do?
A modern, easy-to-use web interface.
Buttons for starting/stopping Scraper, Extractor and Display.
Quick setup: choose the state, minimum and maximum price, car year, etc.
You can automatically generate YAML files for each search.

Auto Dashboard (Car Display)
After the data is extracted, you see it in a large, interactive table.
You can filter by anything: year, price, brand, city, fuel type, score, VIN, etc.
You also have buttons like "Top 1", "Top 2", "Top 3" – for quick filtering.
Everything is live, the data loads automatically from the local server.

What does it bring extra?
It is a complete system, from extraction to display.
It works locally, without needing external databases or complex servers.
It is modularly built, easy to extend and adapt.+

Control Panel – Full Control Over Searches



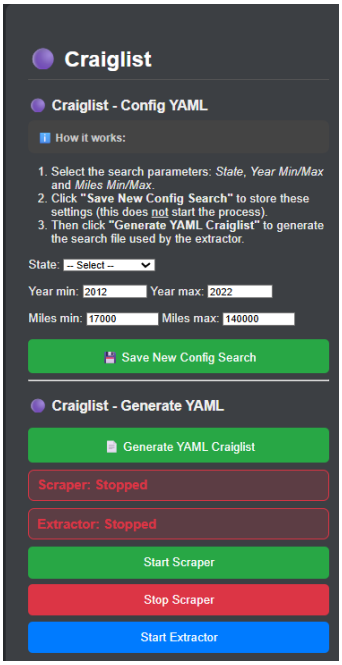
The panel is divided into two areas:

Facebook

You choose the state, minimum and maximum price.
Click “Save New Config Search” to save the settings.
Then generate the YAML configuration file.

Why only price filter?
Because Facebook requires real login and human behavior.
The app uses Selenium and simulates a human:
it fills filters using actual clicks,
scrolls the page to load new ads,
remembers already visited ads (smart logging),
avoids reloading the same data again and again.
We avoid complex queries that might get the account slowed down or blocked.

Craigslist



You can set: state, minimum and maximum year, minimum and maximum mileage.
No login is needed, so filtering is automatic and more flexible.
After setting, you generate the corresponding YAML file.

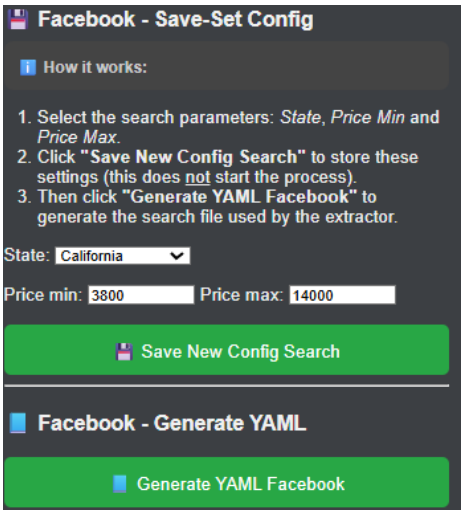
Why this approach?

This project was designed with responsibility and efficiency in mind:
For Facebook: a human-like, realistic, safe approach
For Craigslist: fast, direct, fully filterable scraping
We avoided overcomplicating the app needlessly.
The goal: reliability, efficiency, and full control without risk or account restrictions.

Smart Components
Logs for each search
Memory of already visited ads – they won’t be checked again
Start/Stop system for every module:
Scraper
Extractor
Display

Automated, but manually controlled
The user has full control:
What filters to use
When to start or stop the process
What data to display

Explanation of Control Panel Buttons



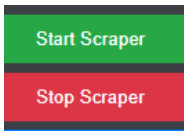
After you configure your search (state, price, year, etc.), you have a set of control buttons that allow you to run the entire system precisely:

State & Minimum Price Selection (Facebook & Craigslist)
The app contains all U.S. states and cities.
When you select a state and a minimum price, that state is placed first in the scan list.
The program starts scanning with that state, then continues with all others, in order.
The price filter is applied to all states.
On Craigslist, it works the same way, but filters also include year and mileage.

Start / Stop Scraper

Start Scraper

Starts the scraping program using the current YAML configuration file.
If you changed filters and clicked "Generate YAML", the scraper will follow the new rules.
If not, it will use the existing YAML data.

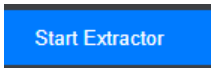


Stop Scraper

To stop the scraper safely, there's a well-thought-out internal mechanism:
When the scraper starts, it saves its process ID (PID) to a file.
When you press Stop, a separate program reads the PID and shuts down only the related processes.
The rest of the application (dashboard, extractor, display) keeps running.

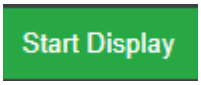
Start Extractor

This component processes all collected data.
It reads the raw data saved by the scraper and converts it into files ready to be displayed in the dashboard.
When processing is done, the CMD window closes automatically, to avoid clutter.



Start Display

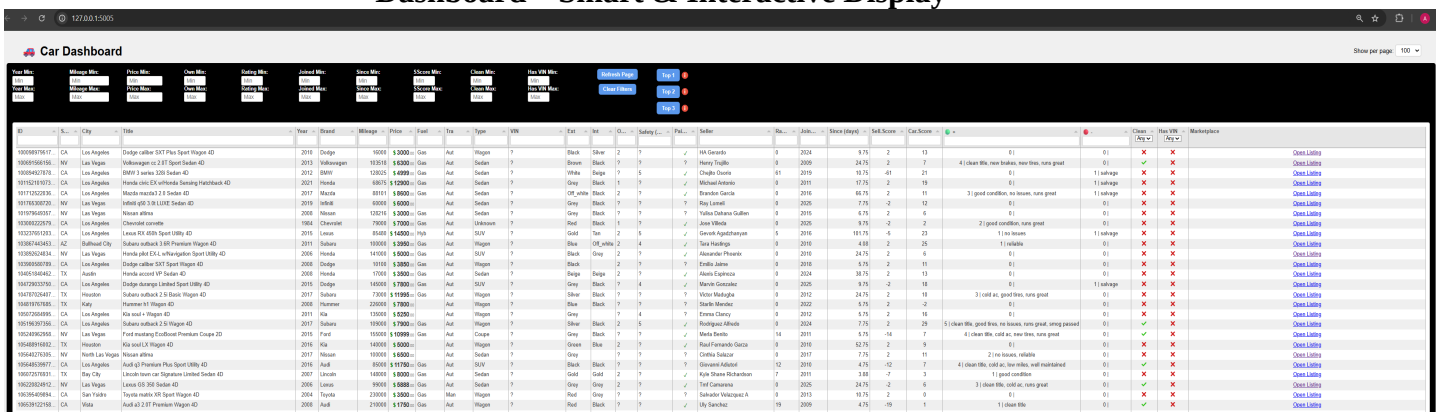
Opens a separate browser window at 127.0.0.1:5005.
Here you'll see all collected listings in an interactive table.
The dashboard includes:
Advanced filters (year, price, brand, city, etc.)
Quick filter buttons: "Top 1", "Top 2", "Top 3"
Instant search and sorting
All key details about each vehicle



The Complete Flow in One Logical Sequence

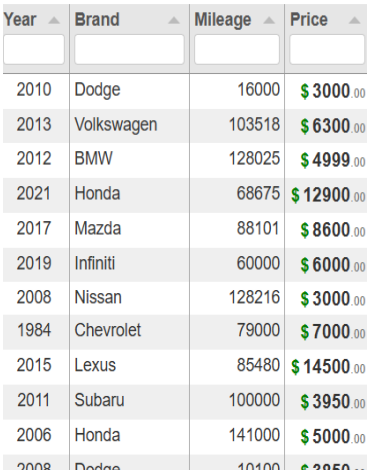
Press Start Scraper → searches and collects listings
Press Start Extractor → processes and prepares the data
Press Start Display → opens the visual dashboard with all results

Dashboard – Smart & Interactive Display



Dynamic Updates

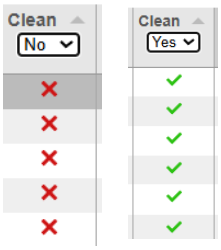
The dashboard is updated in real time after running Start Extractor.
This button launches a series of Python scripts that:
Analyze the raw collected data
Apply multiple filters and interpretation rules
Convert it into a clean, structured, and complete format for display



Year	Brand	Mileage	Price
2010	Dodge	16000	$ 3000.00
2013	Volkswagen	103518	$ 6300.00
2012	BMW	128025	$ 4999.00
2021	Honda	68675	$ 12900.00
2017	Mazda	88101	$ 8600.00
2019	Infiniti	60000	$ 6000.00
2008	Nissan	128216	$ 3000.00
1984	Chevrolet	79000	$ 7000.00
2015	Lexus	85480	$ 14500.00
2011	Subaru	100000	$ 3950.00
2006	Honda	141000	$ 5000.00

What Does the Extractor Do Behind the Scenes?

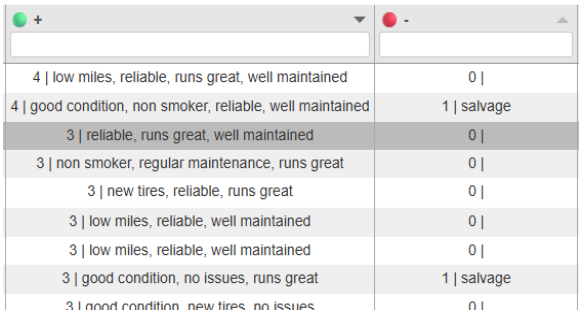
Analyzes the price and detects the currency (e.g., $, €)
Searches for the "clean title" status (even with typos, repetition, or variations)
Recognizes positive expressions (e.g., “no issues,” “good condition”) and negative ones (e.g., “salvage,” “needs repair”)
Detects the VIN using Regex patterns
Identifies the year, brand, and model of the vehicle
Automatically detects the state and city
Optimizes and organizes all data into the 28 dashboard columns



Clean (No)	Clean (Yes)
✗	✓
✗	✓
✗	✓
✗	✓
✗	✓
✗	✓

What’s in the Dashboard?

Column	Examples of Information
ID	Unique ad code
State, City	Vehicle location
Title	Full ad title
Year, Brand, Model	Vehicle specifications
Mileage, Price	Odometer and price
Fuel, Type, Transmission	Technical details
VIN, Clean, Status, Score	Verified info and quality score
Seller, Join, Since, Rating	Seller info and trust indicators
Marketplace	Direct link – Open Listing



+	-
4 | low miles, reliable, runs great, well maintained	0 |
4 | good condition, non smoker, reliable, well maintained	1 | salvage
3 | reliable, runs great, well maintained	0 |
3 | non smoker, regular maintenance, runs great	0 |
3 | new tires, reliable, runs great	0 |
3 | low miles, reliable, well maintained	0 |
3 | low miles, reliable, well maintained	0 |
3 | good condition, no issues, runs great	1 | salvage
3 | good condition, new tires, no issues	0 |

Advanced Features

Filter by any column (top input, min/max values)
Top 1 / Top 2 / Top 3 buttons – quickly show the best-rated entries
Instant sorting – all columns are sortable
Direct link to ad – every row has an "Open Listing" button
Extracted comments – if the ad includes key phrases, they are displayed and interpreted

Why Is This System So Powerful?

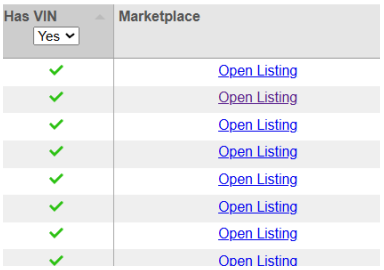
Because it doesn't just extract raw data – it:
Understands, cleans, interprets, and structures it
Handles real ads with typos, informal expressions, and natural language
Displays everything in a clear, easy-to-navigate, and analytical dashboard

Connection to the Original Listing – Direct Contact with the Seller

One of the most important and useful features of the dashboard is that each row contains a direct link to the original ad, exactly as it was posted on:
Facebook Marketplace or Craigslist

How Does It Work?

In the last column (Marketplace), there is a button labeled “Open Listing.”
When you click it, the real ad page opens in a new browser tab.
This allows you to contact the seller directly, without needing to search manually.



Technology Used for This Dashboard

Tabulator.js – a powerful JavaScript library for building interactive tables

Custom JavaScript – used for:
managing filters and sorting,
dynamically generating links,
enabling real-time interaction with backend-processed data
Flask API + AJAX – connects Python data with the visual interface
Dynamic Refresh – no need to manually reload the page

The Result?

You get a user-friendly, fully interactive dashboard that lets you:
View all processed data clearly
Apply advanced filters instantly
Open the original listing with one click
Contact the seller in just a few seconds

Technologies Used in the Project

Backend – Core Application Logic

Technology	Purpose
Python	Main language used for backend logic and scripting.
Flask	Lightweight web framework used for the control panel and API server.
YAML	Configuration format for defining search criteria.
Selenium	Browser automation (simulates human behavior on Facebook Marketplace).
Regex (re)	Extracts complex data like VINs, years, statuses, and keyword patterns.
PID Handling	Manages process IDs to control Start/Stop mechanisms reliably.
Logging	Tracks visited listings and saved entries to avoid duplicates.

Data Processing – Extractor Module

Technology	Purpose
Python Scripts	Analyze raw scraped data.
Smart Filters	Normalize incorrect labels (e.g., "cleean title") and detect insights.
Scoring Engine	Generates auto-ratings based on listing quality, seller reputation, etc.

Frontend – User Interface & Interaction

Technology	Purpose
HTML5 / CSS3	Web page structure and styling.
JavaScript (ES6)	Interactive logic for dashboard and control buttons.
Tabulator.js	High-performance interactive table (filtering, sorting, links).
AJAX (fetch)	Loads data dynamically without refreshing the page.
Bootstrap (optional)	Provides responsive design and modern layout (if used).

Networking & Interaction Flow

Component	Role
localhost:5000	Control panel (Start/Stop system, generate configs).
localhost:5005	Interactive dashboard (view, filter, analyze listings).
Open Listing Link	Direct link to original ad (Facebook or Craigslist).

Project Structure

Folder/File	Description
auto_market_v1/	Scraper modules & YAML config system.
display/	Server handling dashboard interface.
templates/	HTML templates used in the dashboard.
dashboard.py	Main Flask app entry point.
START_CLICK_HERE.bat	Auto-launcher: installs requirements, starts app & browser.

Project Structure – auto_market_v1
Main Folders & Files (Core Components)

Name	Description
facebook/, craigslist/	Main scraping modules for Facebook Marketplace and Craigslist.
alfa_facebook.py, alfa_craigslist.py	Main Python scripts for starting data extraction per platform.
fb_extract_json.py, craig_extract_json.py	Data extractor modules that clean, validate, and prepare raw listings.
dashboard_facebook/, dashboard_craiglist/	Contains the logic and frontend files to display listings interactively via web dashboard.
fb_script_json/	Stores browser session/profile info (used for auto-login on Facebook).
logs/	Log files for monitoring sessions, useful for debugging and avoiding duplicates.
processed_pages/, saved_pages/	Stores raw and processed versions of the scraped HTML content.
chrome_profile/	Stores Chrome profile used by Selenium for automation with saved login.
session/	Temp session storage for ongoing scraping processes.
config/	Configuration folder with YAML files to define search parameters, filters, etc.
config.yaml, config_seller.yaml	Specific YAML files used by the system to customize filtering and search logic.
craiglist_json, results.json	Parsed and structured listing data, used by the dashboard frontend.
program_pid, craiglist_pid	PID files used to track running Python processes (for clean Start/Stop).
install_and_run.bat	Windows script for one-click install and launch.
First_run.py, login_first_time.py	Scripts for setting up the system and logging into platforms for the first time.
README.md, INSTALL.md	Documentation files explaining usage, install steps, and structure.
requirements.txt	Lists all Python dependencies required to run the project.

Quick Summary
The system is built 100% in Python.
Combines Selenium, Regex, and smart filtering for high-quality data extraction.
Connects to a Flask-powered API that feeds a live interactive dashboard built with Tabulator.js.
Offers real-time data control, filtering, scoring, and direct links to live listings.

Final Note – Project Scope and Constraints

This software was designed to be local, compact, robust, and easy to launch, without relying on complex external services. Therefore:

Key Principles

Entirely written in pure Python.
Runs completely locally from a single folder.
No need for external databases or complicated installations.
No internet connection required for control or display functionality.

Accepted Limitations

To respect the minimalistic and user-friendly architecture, a few trade-offs were accepted:
No dedicated domain for live online dashboard hosting.
No SQL-based database or live sync via remote APIs.
No standalone GUI application; everything is integrated and served locally.
Yet, despite these limitations, the application:
Works smoothly, with no crashes or bugs.
Provides a clean and functional interface.
Is highly useful for individuals or businesses looking to track car listings.
Stands as a reliable, real-world automation tool.

Respect for the Client & Development Ethics

This project is the result of a freelance collaboration, created at the request and with the support of a real client.
The source code will not be published publicly on GitHub.
It will not be distributed freely, out of respect for the client's vision, funding, and initiative.
Although the project was developed on a limited budget, it was treated with full seriousness and dedication.
In fact, more hours were invested than initially estimated — because:
“A respectful freelancer honors both their craft and the trust of the client.”

Personal Reflection – A Shift in Approach

This is the first project where I truly focused on practical use and real interaction with the end-user.

My Previous Projects:

Were often too technical, tailored for developers and IT professionals.
Sometimes hard to follow for non-technical users.

What Changed This Time?

From the beginning, this project was designed to be:
Easy to use.
Easy to understand.
Useful even for those with minimal technical knowledge.
The explanations were written as if talking to a car enthusiast, not just a programmer.
The real goal: To build not just functional software, but a logical, intuitive, and helpful tool.

Future Plans & Functionalities

This project was built modularly, with future improvements already in mind.

Database Integration (Internal + External)

Connection to a central database that:
Calculates average market price for each car model.
Enables intelligent price comparisons.
This will be the base for an automated alert system.

Smart Notification System (SMS / App)

If a profitable listing is detected (clean, under market value):
An alert will be sent automatically to the phone.
“Profitable” = Based on logical filtering:
No fake listings, no missing VIN, fair mileage, etc.
Clean, rare, undervalued listings get priority.

Improved Human Behavior Simulation

Enhance Selenium with:
Human-like delays, mouse movement, scroll simulation, etc.
Avoid unnecessary Facebook navigation ⇒ reduce risk of account ban.

Support for More Platforms

Add other car listing websites:
Autotrader, Cars.com, Mobile.de, OLX, and more.
Each integration will respect each site's structure and limits.

Editable & Smart Dashboard

Allow manual editing of any listing:
Status, price, notes, etc.
Mark modified data with a visual indicator (icon or color).
Add internal tracking:
Who modified a listing, what, and when.