# Codebase Documentation: [Gas Price Analysis](https://github.com/QAQ0701/GasPriceAnalysis.git) Pipeline

## Introduction

This document provides an overview of the gas price data pipeline consisting of three Python scripts: scrape\_gasprice.py, clean\_data.py, and visualization.py.

[Visualization Graphs：](https://htmlpreview.github.io/?https://github.com/QAQ0701/GasPriceAnalysis/blob/main/visualization.html)

1. <https://github.com/QAQ0701/GasPriceAnalysis/blob/main/output/heatmap.html>
2. <https://github.com/QAQ0701/GasPriceAnalysis/blob/main/output/time_series.png>
3. <https://github.com/QAQ0701/GasPriceAnalysis/blob/main/output/interactive_graph.html>

## Overview of the Pipeline

The pipeline is executed in three sequential steps:  
1. scrape\_gasprice.py: Collects raw gas price data for specified geographic locations using the GasBuddy API and saves it to an Excel file.  
2. clean\_data.py: Reads the raw data, performs cleaning operations such as timestamp conversion, deduplication, and tagging, and outputs a cleaned dataset.  
3. visualization.py: Loads the cleaned data and generates two visual outputs—a time-series plot of average gas prices by time of day and an interactive geographic heatmap of station prices. These outputs are saved as image files and 2 HTML file for interactive graphs.

## scrape\_gasprice.py

Purpose:  
• Connects to the GasBuddy API to retrieve gas station information and current prices for multiple locations (latitude and longitude coordinates).  
  
Key Functionalities:  
• Configuration: Specifies a list of geographic coordinates for targeted searches.  
• Asynchronous Requests: Uses asyncio to perform API calls concurrently, retrieving station lists and detailed price data.  
• Data Parsing: For each station, extracts fields such as station name, address, ID, pricing details (regular and premium), units, and timestamps.  
• Data Storage: Consolidates the parsed data into a tabular structure and appends new entries to a master Excel file (gas\_prices.xlsx), logging each operation.  
• Logging: Records debug information and errors to a log file, aiding QA in diagnosing issues.  
• Execution Flow: The main function runs parse\_gas\_stations for each predefined location in sequence, with a 60-second pause between each run to avoid exceeding API rate limits.  
  
Output:  
• A single Excel file (./data/gas\_prices.xlsx) containing cumulative gas price records with timestamps, appendable across multiple script executions.

## clean\_data.py

Purpose:  
• Loads the raw gas price data from gas\_prices.xlsx and prepares it for analysis by cleaning, filtering, and deduplicating records.  
  
Key Functionalities:  
• Data Loading: Reads the Excel file into a Pandas DataFrame.  
• Timestamp Conversion: Converts the "Query Time" column to datetime objects, dropping any rows where parsing fails.  
• Filtering: Removes records where both regular and premium prices are missing.  
• Time Tagging: Adds a "Time Tag" column that categorizes data into time-of-day buckets (e.g., morning, afternoon, evening) based on the hour of the query.  
• Date Extraction: Derives "Query Date" by normalizing timestamp values, used for deduplication.  
• Deduplication: Drops duplicate records for the same Station ID, Time Tag, and Query Date to ensure one entry per station per time bucket per day.  
• Sorting & Output: Sorts the cleaned DataFrame by Station ID and writes it to a new Excel file (cleaned\_gas\_prices.xlsx). Logging captures steps and any encountered errors.  
  
Output:  
• clean\_data output file: ./data/cleaned\_gas\_prices.xlsx, ready for visualization and analysis.

## visualization.py

Purpose:  
• Generates visual insights from the cleaned gas price data, providing time-based trends and spatial distribution of prices.  
  
Key Functionalities:  
• Data Loading: Reads the cleaned dataset from cleaned\_gas\_prices.xlsx  
• Time-Series Plot (plotTimeGraph): Creates a line chart showing average regular and premium gas prices over time, segmented by Time Tag (morning, afternoon, evening). The chart is saved as a high-resolution PNG file (time\_plot.png).

• Un-Aggregated Time-Series Plot (plotInteractive): Creates a scatter plot graph showing every data point of regular and premium gas prices over time.

• Geographic Heatmap (plotHeatMap):   
 – Extracts latitude and longitude from the Location field (stored as a dictionary).   
 – Clips extreme price values to handle anomalies.  
 – Aggregates data by station to compute average prices.  
 – Uses Folium to generate an interactive map with color-coded circle markers representing premium and regular prices. Outputs an HTML file (heatmap.html) and a standalone map view.  
• Execution Flow: Calls plotTimeGraph, plotHeatMap, and plotInteractive sequentially after data loading.  
  
Outputs:  
• ./output/time\_plot.png  
• ./output/heatmap.html  
• ./output/interactive\_graph.html

## Dependencies

The pipeline relies on the following Python packages:  
• asyncio (standard library) for asynchronous HTTP requests.  
• logging (standard library) for debug and error logs.  
• pandas for data manipulation and Excel I/O.  
• datetime and time (standard library) for timestamp handling.  
• gasbuddy (third-party) for accessing the GasBuddy API.  
• matplotlib for plotting time-series graphs.  
• folium, plotly, and branca for generating interactive maps.  
  
Ensure that these packages are installed in your Python environment before running the scripts.

## Running the Pipeline

1. Execute scrape\_gasprice.py to collect raw gas price data. This step may take several minutes depending on the number of locations and network latency.  
2. Execute clean\_data.py to clean and deduplicate the raw data. Verify that cleaned\_gas\_prices.xlsx is generated successfully.  
3. Execute visualization.py to produce the visual outputs.

The pipeline can be run automatically by running the bash script autorun.sh in a loop that checks for current times. The time windows are "08:00" (morning), "14:00" (afternoon), "20:00" (evening), "02:00" (midnight) with +1 hour window for each time slot due to constraint’s with the application shortcuts (MacOS).

## Notes:

• Logs are stored in ./log/debug\_log.txt. Review this file to troubleshoot errors or unexpected data issues.  
• The gas\_prices.xlsx file accumulates records across runs—periodic cleanup or archiving may be needed to manage file size.  
• Time Tag buckets are predefined (morning, afternoon, evening, midnight, other). Verify that these categories align with reporting requirements.  
• The heatmap uses clipped price ranges to avoid skewed color scales. Adjust thresholds in visualization.py if station prices fall outside default ranges.  
• Ensure API usage complies with rate limits. Scrape script includes delays, but additional throttling or error handling may be necessary.