OUT THE GARAGE

Aubrey L. Baker, Amy Manning, Sresha Ventrapragada

COE332: Software Engineering and Design

1 Project Description

There is an increasing effort in transitioning the world to sustainable energy. Many aspects of an individual's lifestyle can make a large difference in their carbon footprint. To address this, it is critical for the population to have easy access to information on how they can contribute to the movement. This application identifies key data that would help an individual reduce their automotive carbon footprint. The automotive CO_2 data is public information and can be accessed online from the EPA's (Environmental Protection Agency) database in CSV format. This application retrieves the data in the CSV and returns it in a useful and readable way for the user to understand automotive trends that will help guide them in making automotive decisions that will reduce their carbon footprint.

Consumers can use this information to make informed financial and environmental decisions when purchasing a vehicle. Similarly, manufacturers can see where they stand amongst competitors. Not only is the data important for the consumers that use transportation vehicles, but also the rest of the population that utilizes fossil-fuel-dependent mechanisms. If the whole population was to drive vehicles with an inefficient MPG, the available fossil fuels would quickly deplete and an alternative energy source would need to be established and readily available.

This information is critical to the sustainability of transportation as concerns about climate change and pollution rise.

2 Data

The data source for this web application is the EPA's (Environmental Protection Agency) and NHTSA's (National Highway Traffic Safety Administration) greenhouse gas emissions, fuel economy, and technology information from 1975 to prelim 2022. The data set can be accessed at this link [1]. The website is owned by a government organization and is public information. An example of the data shown within the link is shown below:

Real-World MPG_City	Real-World MPG_Hwy	Real-World CO2 (g/mi)	Real-World CO2_City (g/mi)	Real-World CO2_Hwy (g/mi)	Weight (lbs)	Footp... (sq. ft.)	Engine Displ...	Horse... (HP)	Accel... (0-60 time in	H(V)E... Displ...
35.3	36.1	248	252	246	5000	57.0	152	189	7.4	1.245
35.5	36.2	248	250	246	5000	57.1	152	189	7.6	1.745
29.6	25.9	375	424	338	5030	52.9	174	343	6.0	1.931
29.6	25.9	375	424	338	5030	52.9	174	343	6.0	1.931
21.1	25.0	369	411	338	5009	52.7	174	327	6.0	1.995
21.1	25.9	369	411	338	5009	52.7	174	327	6.0	1.995
22.8	31.8	310	382	271	4365	50.9	156	326	5.9	2.000
16.3	22.2	457	551	403	5309	57.5	302	362	6.8	1.270
24.5	31.6	361	397	298	3914	48.6	131	284	6.5	1.979
23.8	28.5	336	370	310	4388	50.9	168	250	7.4	1.593
23.8	28.5	336	370	310	4388	50.9	168	250	7.4	1.593
15.9	21.6	478	562	414	5416	57.2	296	362	6.9	1.239
24.2	28.6	333	366	300	4402	48.2	188	233	7.7	1.256
24.1	28.2	322	363	291	4381	56.7	177	227	7.8	1.263
24.1	28.5	335	387	311	4345	48.1	182	234	7.8	1.384

Manuf...	Regulatory Class	Vehicle Type	Model Year	Production (built)	Production Share	2-Cycle MPG	Real-World MPG	Real-World MPG_City	Real-World MPG_Hwy
Toyota	Truck	Minivan/Van	2021	93	0.007	48.6	35.8	35.3	36.1
Toyota	Truck	Minivan/Van	Prelim. 2022	-	-	48.7	35.9	35.5	36.2
BMW	Truck	Truck SUV	Prelim. 2022	-	-	30.0	23.3	20.6	25.0
BMW	Truck	All Truck	Prelim. 2022	-	-	30.0	23.5	20.6	25.9
BMW	Truck	Truck SUV	2021	151	0.011	30.5	23.6	21.1	25.0
BMW	Truck	All Truck	2021	151	0.011	30.5	23.6	21.1	25.9
Mercedes	Car	Sedan/Wagon	Prelim. 2022	-	-	34.7	27.2	22.8	31.8
Stellantis	Truck	Pickup	Prelim. 2022	-	-	24.0	19.2	16.3	22.2
VW	Car	Sedan/Wagon	Prelim. 2022	-	-	37.3	29.0	24.5	33.6
Hyundai	Truck	Truck SUV	Prelim. 2022	-	-	34.2	26.3	23.8	28.6
Hyundai	Truck	All Truck	Prelim. 2022	-	-	34.2	26.3	23.8	28.5
Stellantis	Truck	Pickup	2021	470	0.035	25.4	18.7	15.9	21.6
Toyota	Truck	Truck SUV	2021	1012	0.075	34.7	26.5	24.2	28.6
All	Truck	Minivan/Van	2021	238	0.022	35.5	27.3	24.1	30.2
Toyota	Truck	Truck SUV	Prelim. 2022	-	-	35.0	26.4	24.1	28.5
BMW	All	All	Prelim. 2022	-	-	33.1	25.7	22.3	29.8

Figure 1: Table A-1 from epa.gov

The EPA (Environmental Protection Agency) and NHTSA (National Highway Traffic Safety Administration) collects data from car manufacturers annually on each vehicle type's CO_2 emissions. There are two types of data the EPA uses in this set, preliminary and final. Preliminary is only used until the manufacturer reports the data, then the reported data is

recorded as final. The preliminary data in this set is from 2022. As such, the analyzed data excludes the year 2022 so that the analyzed data is *only* finalized data.

The data is categorized by manufacturer and is organized based on the categories as seen in the tree diagram below following CAFE (corporate average fuel economy) and GHG (greenhouse gas) regulations of light-duty vehicles. Each statistic is an average taken by arithmetic weighted average and harmonic weighted average methods. For better identification of long-term trends, EPA applies new manufacturer definitions to prior model years for estimated real-world CO_2 emission and fuel economy trends.

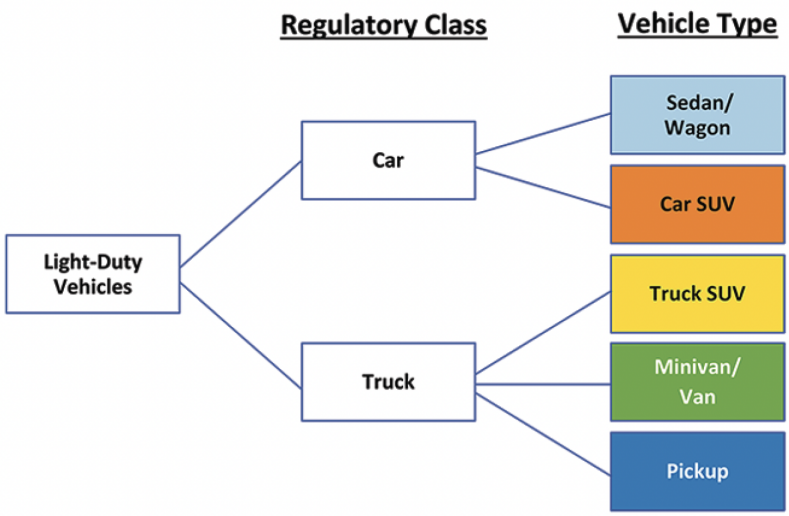

Figure 2: Data Organization Tree Diagram

This data set includes information about manufacturer and vehicle type, which are categorical values, and model year, production share, compliance and estimated real-world MPG, CO_2 emissions, weight, horsepower, and many more, which are numerical values. These numeric variables reflect arithmetic and harmonic production-weighted averages for each vehicle type.

3 Tools and Software

3.1 Flask

Flask is known for its simplicity, flexibility, and ease of use. It is used for building web applications, particularly for creating web APIs. Flask provides a number of features that make it a popular choice for web development.

One of the key benefits of Flask is its lightweight nature. It does not require any particular tools or libraries to be installed, making it easy to set up and use. Flask is also highly customizable, allowing developers like me, to add or remove features as needed.

Another advantage of Flask is its extensive documentation and large community of developers. Since this is one of our first web applications, the abundance of documentation allowed us to easily debug any errors that we ran into and quickly enhance our knowledge in using the Flask framework.

3.2 Docker

Docker is a containerization platform that allows developers to package and deploy their applications in a self-contained environment, known as a container. A container is a standalone executable package that contains everything an application needs to run, including code, libraries, dependencies, and more.

Implementing Docker in this project allowed us to understand the scalability and potential of containerization for more advanced projects.

3.3 Redis

Redis is a NoSQL database and "data structure store" with many features such as key-value store and hash maps. This allows for an organized collection of structured information, or data stored in a computer system. Databases provide a query language - a small, domain-specific language for interacting with the data. The query language is not like a typical programming language such as Python or C++; you cannot create large, complex programs with it. Instead, it is intended to allow for easy, efficient access to the data.

Our data needs permanence and we want to be able to stop and start our Flask API without losing data. We want multiple Python processes to be able to access the data at the same time. This includes Python processes that may be running on different computers.

3.4 Kubernetes

Kubernetes is also known as *K8*. The main purpose of utilizing Kubernetes is to serve as a container orchestration system that automates tasks needed to manage the containers. Kubernetes allows for ease of deployment, scaling, and monitoring applications.

Kubernetes also includes built-in monitoring tools that allow you to keep track of the health and performance of the application. This can help quickly identify and resolve any issues that arise, ensuring that the applications remain available and responsive to users.

4 Route Endpoints

Route	Purpose
/help	returns all the routes and their purpose
/data	-X GET returns the entire data set (hundreds of dictionaries) -X POST adds the data to the redis database -X DELETE deletes all the data from redis database
/years	returns a list of all the years recorded in the dataset
/years/<year>	returns a list of all the data from the specified year
/manufacturers	returns a list of the manufacturers currently in redis for the Auto Trends Database
/manufacturers/<manufacturer>	returns a list with all cars from that manufacturer if found in the Auto Trends database
/manufacturers/<manufacturer>... /years	returns a list of the years where there is data for a specific manufacturer
/manufacturer/<manufacturer>... /years/<year>	returns a list for the data for the specified manufacturer and year if found in the Auto Trends database route
/jobs	API route for creating a new job to do analysis. This route accepts a JSON payload describing the job to be created

Route Cont.	Purpose Cont.
/download/<jobid>	downloads an image that was generated by the worker from Redis given the job ID
/status/<jobid>	returns the status of the specified job ID
/CO_2_year_plot	-X POST loads a plot of the total CO_2 emissions for a user specified range of years -X GET <some location>:<port>/CO_2_year_plot –output output.png returns a plot of total CO_2 emissions for a range of years to redis data base -X DELETE deletes the image from the Redis data base
/weight_mpg_plot/<year>	-X POST loads a plot of the weight vs fuel economy of every vehicle for a user specified year -X GET <some location>:<port>/weight_mpg... _plot/<year> –output output.png returns a plot of weight vs fuel economy to redis data base -X DELETE deletes the image from the Redis data base
/vehicleType_mpg_plot/<year>	-X POST loads a plot of the weight vs fuel economy of every vehicle for a user specified year -X GET <some location>:<port>/vehicleType... _mpg_plot/<year> –output output.png returns a plot of weight vs fuel economy to redis data base -X DELETE deletes the image from the Redis data base

5 File Contents and Usage

5.1 File Contents

`README.md` contains all details and instructions on how to get the software running.

5.1.1 Docker

`Dockerfile.api` utilizes Docker to allow the user to execute the API within a container with all appropriate packages the user might not already have.

`Dockerfile.wrk` utilizes Docker to allow the user to execute the worker that executes jobs consumed of the hotqueue within a container with all appropriate packages.

`docker-compose.yml` is a powerful tool that simplifies the management of multi-container applications. It defines all containers and allows for less keystrokes and more straightforward commands to start the Flask application.

5.1.2 Kubernetes

Deployments are a resource type in Kubernetes that are used to represent long-running application components. They create an amount of "pods" specified in the file and will restart if the health of the deployment is failing. Services are another type of Kubernetes resource that abstracts APIs and databases to easily communicate on a network to other Kubernetes pods. They are especially useful in this program for abstracting IP addresses and ports.[2]

`autotrends-prod-flask-deployment.yml` is the Flask deployment that will create pods for the API. This file contains the Redis IP address, container port, and docker image.

`autotrends-prod-flask-service.yml` is the Flask service that communicates with the flask app and port 5000.

`autotrends-prod-redis-deployment.yml` creates deployment pods for the Redis database. It connects with persistent volume mounts to offload the data in a separate node in the Kubernetes cluster where it will be safe.

`autotrends-prod-redis-pvc.yml` is a persistent volume claim for the redis database to store our data in the volume mount cluster.

`autotrends-prod-redis-service.yml` is a service for the redis database connecting port 6379 to 6379.

`autotrends-prod-wrk-deployment.yml` is a deployment for the HotQueue worker. The queue is an additional data structure that acts as a middleman between the Flask API and Redis. The worker will take on the jobs as it can, ensuring the system does not overload itself.

`flaskprod_ingress.yml` specifies a subdomain to make the Flask API available on and maps its domain to the public port made by the `flaskprod_nodeport_service.yml`

`flaskprod_nodeport_service.yml` selects the Flask deployment using its label and exposes the Flask API on a public port.

5.1.3 Source

`auto_trends_api.py` is the Flask API that contains all the routes the user can curl.

`jobs.py` receives the request from the user, record the request in Redis and give it a unique ID and immediately respond to the user. The user may not get the result back immediately, but eventually will once the job is complete.

`worker.py` retrieves a job ID from the task queue and executes the job. It also monitors the job until complete and updates the database accordingly.

5.2 Using the Files

In order to use the files described above and located in the auto-trends-api repository, first the user must clone the repository to their local system using `git clone git@github.com:sreshaven/auto-trends-api.git` and then `cd auto-trends-api/`. Next, the user can either use Docker or Kubernetes to start up the Flask app and utilize the API routes.

5.2.1 Docker

Method 1: The user can get the existing images for the Flask app and the worker on Docker Hub by running `docker pull sreshaven/auto_trends_api:final` and `docker pull sreshaven/worker_api:final`

Method 2: The user could also build their own Docker images from the Dockerfiles provided in the repository to start up the containers. To do this, they must first get the .csv file from this link [1] by clicking on the blue button that says "Export Detailed Data by Manufacturer" (which is Table A-1 on the site), and next, the user should rename the file to `auto_trends_data.csv` and move this file to the `auto-trends-api/src/` directory. Then, the user must run `docker build -f docker/Dockerfile.wrk -t <username>/worker_api .` and `docker build -f docker/Dockerfile.api -t <username>/auto_trends_api:final .` and replace **<username>** with their username to build the images. Lastly, to prepare for launching the containers, in the docker-compose.yml file, the user should change the lines that say `image: sreshaven/auto_trends_api:final` and `image: sreshaven/worker_api:final` by replacing sreshaven with their username. In order for the user to use their built images with Kubernetes, they should push the images to Docker Hub using `docker push <username>/auto_trends_api:final` and `docker push <username>/worker_api:final` and replace **<username>** with their username.

To finish the set of the Flask app with Docker, the user should make directory for Redis to mount a volume by running the command `mkdir docker/data/` and lastly, launch the Redis, Flask app and worker containers by running `docker-compose -f docker/docker-compose.yml up -d`.

5.2.2 Kubernetes

To start setting up the Flask app with Kubernetes, the user should be on a system that has access to the Kubernetes API for the cluster and on this system, they should change directory to the kubernetes/prod/ folder by running `cd kubernetes/prod/`.

Redis: For the set up of Redis for the Flask app to store data, there will be three parts: a Persistent Volume Claim (PVC), deployment, and service. The user should set up the PVC to save the Redis data from the Flask app by running `kubectl apply -f autotrends-prod-redis-pvc.yml`. Next, the user should create a deployment for the Redis database so that the desired state for Redis is always met by running `kubectl apply -f autotrends-prod-redis-deployment.yml`. Lastly, the user should start the service so that there is a persistent IP address that the user can use to talk to Redis by running `kubectl apply -f autotrends-prod-redis-service.yml`.

Worker: To set up the worker to take care of analysis jobs submitted through the Flask app, there will be a deployment. The user should set up the deployment for the worker on the Kubernetes cluster by running `kubectl apply -f autotrends-prod-wrk-deployment.yml`.

Flask: To set up the Flask app in order for the user to run routes to explore the Auto Trends data set, there are multiple parts: deployment, service, and (optional) nodeport service and ingress, To set up the deployment for the Flask app on the Kubernetes cluster, the user should run `kubectl apply -f autotrends-prod-flask-deployment.yml`. The user can get a persistent IP address for the Flask app with a service by running `kubectl apply -f autotrends-prod-flask-service.yml`. The user can now run the Flask app routes in a Python debug container with the IP address for the Flask app which can be found using `kubectl get services`.

5.2.3 Using the API

To use the API, the user must have the containers/pods up and use http://127.0.0.1:5000 for Docker set up or the IP address for the Flask service for Kubernetes. Another option is that the user can utilize the domain that is already available on the internet with the examples below by replacing http://127.0.0.1:5000 with otg.coe332.tacc.cloud.

Route: /data By setting the method to GET, this route can be used by the user after posting the data to get all of the data directly from the Redis database. The user can run the command `curl -X GET http://127.0.0.1:5000/data` and below is an example of what the output looks like:

```
[
  {
```

```
    "2-Cycle MPG": "19.42850",
    "4 or Fewer Gears": "0.861",
    "5 Gears": "0.139",
    "6 Gears": "-",
    "7 Gears": "-",
    "8 Gears": "-",
    "9 or More Gears": "-",
    "Acceleration (0-60 time in seconds)": "15.5542",
    "Average Number of Gears": "3.5",
    "Cylinder Deactivation": "-",
    "Drivetrain - 4WD": "0.211",
    "Drivetrain - Front": "0.018",
    "Drivetrain - Rear": "0.771",
    "Engine Displacement": "235.7915",
...
```


Route: /years/<year> To get all of the data points from a certain model year, the user can run `curl http://127.0.0.1:5000/years/<year>` and replace <year> with a model year that the user specifies. Below is an example of what `curl http://127.0.0.1:5000/years/2007` looks like:


```
[
  {
    "2-Cycle MPG": "20.33974",
    "4 or Fewer Gears": "0.979",
    "5 Gears": "0.018",
    "6 Gears": "0.004",
    "7 Gears": "-",
    "8 Gears": "-",
    "9 or More Gears": "-",
    "Acceleration (0-60 time in seconds)": "8.8777",
    "Average Number of Gears": "4.0",
    "Cylinder Deactivation": "0.309",
    "Drivetrain - 4WD": "0.453",
    "Drivetrain - Front": "-",
    "Drivetrain - Rear": "0.547",
    "Engine Displacement": "297.9739",
...
```


Route: /manufacturers/<manufacturer> To get all of the data associated with a specific manufacturer, the user can use the command `curl http://127.0.0.1:5000/manufacturers/<manufacturer>` and replace <manufacturer> with the name of a specific manufacturer. Below is an example of what `curl http://127.0.0.1:5000/manufacturers/Toyota` looks like:

```
[
  {
    "2-Cycle MPG": "30.58902",
    "4 or Fewer Gears": "0.770",
    "5 Gears": "0.230",
    "6 Gears": "-",
    "7 Gears": "-",
    "8 Gears": "-",
    "9 or More Gears": "-",
    "Acceleration (0-60 time in seconds)": "11.6225",
    "Average Number of Gears": "4.0",
    "Cylinder Deactivation": "-",
    "Drivetrain - 4WD": "-",
    "Drivetrain - Front": "0.847",
    "Drivetrain - Rear": "0.153",
    "Engine Displacement": "120.2838",
...
```


Route: /manufacturers/<manufacturer>/years/<year> To get the data points for a specific manufacturer during a certain year, run the command `curl http://127.0.0.1:5000/manufacturers/<manufacturer>/years/<year>` and replace <manufacturer> with the name of the manufacturer and <year> with the model year. Below is an example of what running curl http://127.0.0.1:5000/manufacturers/Tesla/years/2019 could return:


```
[
  {
    "2-Cycle MPG": "125.19278",
    "4 or Fewer Gears": "1.000",
    "5 Gears": "-",
    "6 Gears": "-",
    "7 Gears": "-",
    "8 Gears": "-",
    "9 or More Gears": "-",
    "Acceleration (0-60 time in seconds)": "4.2058",
    "Average Number of Gears": "1.0",
    "Cylinder Deactivation": "-",
    "Drivetrain - 4WD": "1.000",
    "Drivetrain - Front": "-",
    "Drivetrain - Rear": "-",
    "Engine Displacement": "-",
    "Footprint (sq. ft.)": "54.80000",
...
```

Route: /CO_2_year_plot The user can use this route to get the image of a graph of the average CO_2 emissions per year plotted over time locally if it is present in the database with the method GET. To do this, the user can run `curl -X GET http://127.0.0.1:5000/CO2_year_plot --output filename.png` and replace filename with a name for the file. This will return the image of the graph in redis to the output file path specified and will return an output and an image like below:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 26827  100 26827    0     0  3274k      0 --:--:-- --:--:-- --:--:-- 3274k
```

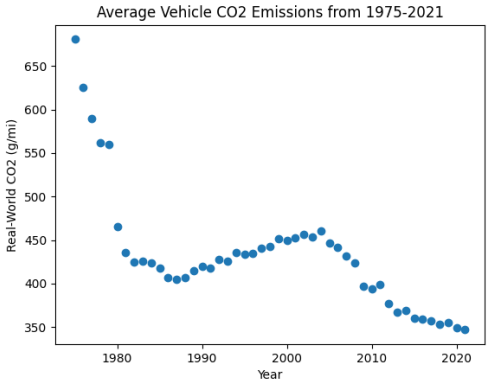

Figure 3: Plot for /CO_2_year_plot

Route: /weight_mpg_plot/<year> The user can use this route to get the image of a graph that compares vehicles weights to fuel efficiencied from the year specified by the user if it is present in the database with the method GET. To do this, the user can run `curl -X GET http://127.0.0.1:5000/weight_mpg_plot/<year> --output filename.png` and replace filename with a name for the file and <year> with a specific year. This will return the image of the graph in redis to the output file path specified and will return an output and a plot like below for the parameter year of 2021:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 23828  100 23828    0     0  2908k      0 --:--:-- --:--:-- --:--:-- 2908k
```

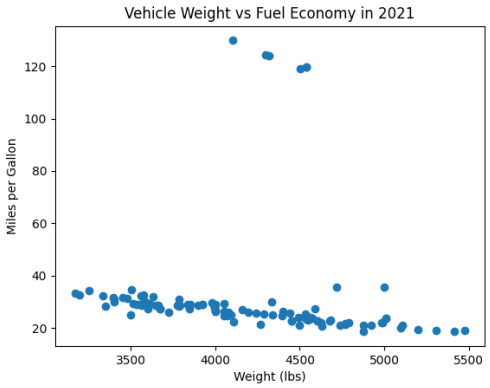

Figure 4: Plot for /weight_mpg_plot/<year>

Route: /vehicleType_mpg_plot/<year> The user can use this route to get the image of a graph that compares vehicle type to its fuel efficiency from the year specified by the user if it is present in the database with the method GET. To do this, the user can run `curl -X GET http://127.0.0.1:5000/vehicleType_mpg_plot/<year> --output filename.png` and replace filename with a name for the file and <year> with a specific year. This will return the image of the graph in redis to the output file path specified and will return an output and a plot like below for the year 2021:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 20576  100 20576    0     0  9222      0 --:--:-- --:--:-- --:--:-- 9222
```

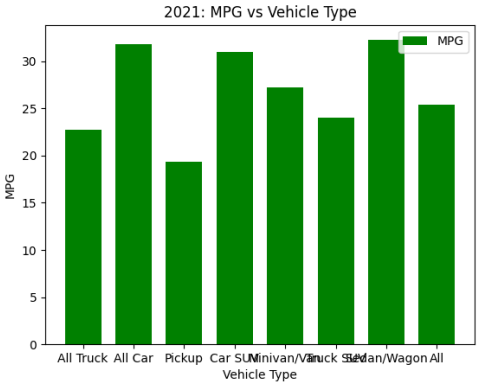

Figure 5: Plot for /vehicleType_mpg_plot/<year>

Route: /jobs To submit a job for analysis, the user can run the command `curl http://127.0.0.1:5000/jobs -X POST -H "Content-Type: application/json" -d '"start": "<start>", "end": "<end>"'` and replace <start> and <end> with years between 1975 and 2021. This will return a JSON object with information about

your job including the job ID. Below is an example output for `curl http://127.0.0.1:5000/jobs -X POST -H "Content-Type: application/json" -d '"start": "1975", "end": "2021"'`:

```
{"id": "8c556a8b-9e2c-4dd3-8dbf-bf667826ca87", "status": "submitted", "start": "1975", "end": "2021"}
```

Route: /download/<jobid> To download the image from a specific job ID, you can run the command `curl http://127.0.0.1:5000/download/<jobid> --output filename.png` and replace <jobid> with the specific job ID and filename with a name for the output file. Below is an example output and plot for `curl http://127.0.0.1:5000/download/8c556a8b-9e2c-4dd3-8dbf-bf667826ca87 --output image.png`:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 63626  100 63626    0     0  59519      0  0:00:01  0:00:01 --:--:-- 59519
```

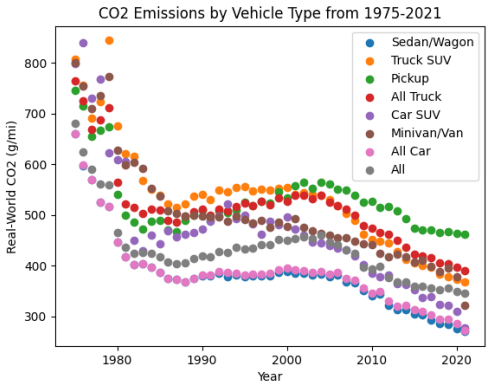

Figure 6: Analysis Job Plot for 1975-2021

6 Ethical and Professional Responsibilities

Ethics are principles that are important to consider in engineering situations, and it should be a responsibility for engineers as part of their profession to take into account ethical issues and values in their work. This is to ensure that an engineer's decisions are aligned with their obligations to the public and the industry to respect different perspectives and their choices remain honest.

One ethical responsibility of engineers is to cite data when using outside sources. This is important because it provides the producer with appropriate credit and it provides a way to enable the reproducibility of the analysis on the data and users can locate the data source more easily as well.

Another professional responsibility is to check the accuracy and quality of data that is used by applications. This is critical to ensure the usability of the users of the application

and also it allows better decision making for users with the results from the app. The next professional responsibility is documenting and communicating to users and other engineers the process and knowledge in creating something. This is vital for engineers because it encourages knowledge sharing and also allows for users to easily understand how to use the creation or application. This also ensures that the engineer is acting with integrity in their methods by being clear in public communications and documentation.

The last example of an ethical and professional responsibility for engineers is to design solutions with all people and values in mind. Considering the different cultures, perspectives, and needs in the user population is becoming more important because it helps avoid excluding people and prevent bias in the solution so that it does not favor one person over another and standard behavior exists for everyone. Overall, these responsibilities will allow for better quality applications and more honest engineering situations.

7 Software Design Principles

7.1 Modularity

Modularity involves dividing the components of software into parts. This application displays how programs can integrate together to form a complete system.

7.1.1 Intra-module Cohesion

The use of different files that focus on specific tasks and communicate with each other to provide one output helps to achieve intra-module cohesion. For example, in one file, there are functions related to the API, while another file contains functions related to the jobs. This makes it easier to manage the code and understand the different parts of the software.

7.1.2 Inter-module Coupling

Intermodule coupling is achieved by grouping together certain modules depending on what functions they need. For example, the API depends on the jobs module, and the worker module also depends on the jobs module. This ensures that the different parts of the software are communicating with each other effectively to achieve the desired outcome.

7.2 Abstraction

Abstraction replaces an exact concept with a simplified model. For instance, using Kubernetes to deploy software with containers to the local system, demonstrates how a complex deployment process can be abstracted into a simplified model. By using containers, each user can have their own version of the software, eliminating the need for a shared environment and simplifying the deployment process.

Another example of abstraction within our software was deploying the web app to the internet, demonstrates how abstraction can simplify the user experience by providing an easy-to-use interface for downloading images. Rather than requiring users to manually download each image, the web app abstracts this process, making it more accessible and convenient.

Lastly, using file bytes to represent data stored in plots, demonstrates how abstraction can simplify the representation of complex data. By abstracting the data into a simpler format, the software can more easily manipulate and store it.

7.3 Generalization

Generalization allows the same software component to be used in different situations. Creating functions that can be used for many different types of input from different manufacturers or years, demonstrates how generalization can improve the reusability of software. By identifying commonalities in the input data and creating functions that can handle those commonalities, the software can be used with different types of input, without requiring modification to the underlying code.

By generalizing the functions to handle different inputs, this eliminates the need to write new code for each specific case. Not only does this save time and effort but also reduces the risk of errors and increases the reliability of the software.

7.4 Portability and Reproducibility

Portability refers to the ability to access and execute software from different kinds of computers, while reproducibility means that the results obtained from executing the software at one time can be obtained later. Through the use of our docker image and the redis databases, multiple users can access this application and reuse it. In addition, anyone from any computer can execute this same application and get the same expected output.

Portability and reproducibility can be achieved through the use of a Docker image and Redis databases. By packaging the application in a Docker image, users can access and reuse the application from any computer, regardless of the underlying operating system. This makes the application more portable and accessible to a wider audience, improving its overall usefulness.

Furthermore, by using Redis databases, the application can achieve reproducibility by ensuring that the same data is used every time the application is run. This means that the results obtained from executing the application at one time can be obtained later, making it easier for users to verify and validate their results.

The fact that there are detailed steps for users to reproduce the Flask app outlined in the README.md file also contributes to the overall portability and reusability of the software. This makes it easier for users to set up and run the application, regardless of their level of expertise or the tools they have available. [3]

References

[1] O. US EPA, "Explore the automotive trends data," 10 2018.

[2] W. J. Allen, J. Stubbs, and C. Dey, "Coe 332: Software engineering and design," 2023.

[3] A. N. Knaian, *Electropermanent magnetic connectors and actuators: devices and their application in programmable matter*. PhD thesis, Massachusetts Institute of Technology, 2010.