Component/Design Specification

Software Components

1) **Visualization Component:**

   Purpose: The main visualization component's main purpose is to give a clear idea of how many alternative-fuel charging stations exist in the nation and the breakdown of each kind of station that exists.

   Input: Input required to enable this component is broken-down government data about the nation's charging stations as well as user clicks and hovers around our map

   1) Python's **Plotly** library has a choropleth map feature that, when implemented, outputs a clean map of the United States and maintains interactivity.
   2) Our method takes in our data frames as arguments and contains an if-else statement to display a full United States map with every state highlighted with a breakdown of stations per state with data imported from **StationsByState.csv.**

   Output: The output of this visualization is a clear state-by-state visual analysis so users can know in what states what kind of charging stations exist for their car (electric, hydrogen, propane).

2) **Classification Component**

   Purpose: The classification component is in charge of recommending the user an electric vehicle based on the "survey" that the user takes at the beginning.

   Input:

   1) The tool accesses our questions file and prompts 4 answers from the user. The questions are instantiated as part of the *Question* class. The **`run_questions(questions)`** method takes in the list of questions for the user and encodes the outputs as numbers with the 3 inner methods:
      a) **`new_dict_rapidcharge()`**
      b) **`new_dict_driving()`**
      c) **`new_dict_budget()`**
   2) The answers are validated by our **(test_questions.py)** file and raises exceptions and errors if an invalid input is found.

3) Once all 4 answers have been validated our next method in the tool encodes each answer as an integer (ex: [4,1,3,2])
4) Our backend Knn algorithm in our **(model.py)** file is written as the **`build_knn_model(results)`** and takes in the results array from Step 3. The results are essentially modeled as a 4-dimensional plane and our model calculates the Euclidean distance between the user's answers and each car in our dataset.

Output: The car that is the smallest "distance" from the user's inputs is returned by our recommender system and the final method in our **(model.py)** file prints out all the attributes of the car that is the closest match for the user. The methods that return the recommendation and its features are `reccomended_EV()` and `get_ev_stats()` methods.

3) App.py component

Purpose: Our app.py file wraps in both the visualization and classification components and incorporates a few more trend charts regarding the statistics of electric vehicles to supplement the content on the page. It acts as the bridge between the main features that our application has to offer.

Input: The input into the App.py component is the rendering of the main Plotly visualization, the rendering of the user survey which connects to the backend classification model, and an assortment of html components that render different charts that supplement our research and our work.

Output: The output of the App.py component is the content that is displayed on the page.

Interactions to accomplish use cases

Our system's two-pronged approach with classification and visualization will take into account things that a user can control well, (i.e. their car preferences) and things that a user cannot control well (where they live). Our data cleaning component will make sure our data is in a good environment for analysis with our classification and visualization tools importing our data straight from that component. Our visualization will be mostly centered around making the user more comfortable with their electric vehicle choices by detailing the charging stations available to them in their particular state. This visualization is supplemented with trend charts regarding the rise of electric vehicle usage and electric car vehicle sales by state over time. After that initial information is given and the users have a better idea of the market for electric vehicles, they will then pick their preferences for what they want and like in a car, and the classification model will use that information

to recommend a modern electric vehicle and give a short summary of why that car might be the best fit for them.

Preliminary plan

1) **Data collection and aggregation:** We are planning to use 4 different datasets from various sources at the base level before aggregation. We get our data downloaded and ready in our repository and our workspace using *pandas* library inside Google CoLab and examine the shape and columns available
2) **Data cleaning:** Merge our datasets as necessary, clean the data into formats we will need for visualization and machine learning analysis using both the *pandas* and *numpy* libraries
3) **Visualization Building:** Import visualization libraries and our visualization-based dataset and begin experimenting ways in which we can create meaningful, interactive, and informative visualizations using the *plotly* and *dash* libraries.
4) **Feature Building for Classification:** For our classification tool, we have to make our features readable and clean for the users. This step is to pre-process those features using the *pandas* library and store this data cleaning code in a separate python file.
5) **User Interaction for Classification:** Create a "survey" for the user in Google CoLab and have their responses be stored in a dictionary we will use in our classification later.
6) **Format Analysis for User Responses:** Create methods for encoding our user's inputs so that they are translated into input values that will be readable by our backend machine learning model.
7) **Define our machine learning model:** Utilize the *sklearn* library and its implementation of *KNearestNeighbors* to fit our data
8) **Feed inputs into model:** Input our encoding of the user's results into the model and return the output and the statistics with a short return method.
9) **Building Unit Testing:** Build out the unit tests for user input validation, our data cleaning tools, and visualization's data access
10) **Program Run:** Make sure we have one *setup.py* file that will be able to execute our entire project (classification and visualization) together in a streamlined and aesthetic way.
11) **Directory structure:** Transfer all our notebooks, unit tests, data, and documents into our GitHub repository and make sure our structure is up to date.