Weather Crisis Evacuation in the Greater Philadelphia Area

Daisy Aptovska

Andrew Herman

Computer Science Program at The Pennsylvania State University at Abington

CMPSC 463: Design and Analysis of Algorithms

Professor Janghoon Yang

December 3, 2025

Table of Contents

Description of the Project

In this project, we designed a method to analyze the evacuation routes for different weather events in the Greater Philadelphia Area. We designed a visualization map to show optimal routes from a user's position to the nearest and safest evacuation centers using a graph data structure and applying Dijkstra's and Prim's algorithm. We used Dijkstra's algorithm for the shortest and most optimal route to the center. We integrated Prim's algorithm to find the minimum spanning tree (MST) of all the evacuation centers. Through this project, we analyzed flood zones and hurricane paths, and we provided the users with data from evacuation centers and the routes between them.

Significance of the Project

This project is significant because it impacts a real-world problem. Many individuals in Philadelphia may not be aware of their emergency routes, and this knowledge is crucial in a weather crisis. These algorithms and software can be scaled for many other cities, providing habitants of the cities with essential crisis information. Additionally, this project can be included on local government websites and weather channels to spread essential weather information for habitants.

Code Structure

In the code structure of our project, there are 3 main directories, algorithms, data, and visualization. Outside of these directories, in the main project directory, there is a preprocessing Python file which is used for reading in the data to form the main pickle file for data usage. Our road and evacuation center data was obtained from Open Data Philly. Once the data was downloaded into GeoJson files, it was imported into a Google Drive folder, accessible in our

GitHub readme section. This data was used in the preprocessing file, and we created a hazards pickle file. We used a pickle file for easy and fast access to our data when the program is run.

When the program is run for the first time, preprocessing must be run first to generate the hazards pickle file. Then, the main application can run like normal. When the main program is run for the first time, an evacuation map of HTML file will be created. In this file, the interface is displayed. Based on the user's starting location. The program displays Dijkstra's algorithm on the shortest path found for the closest evacuation center based on the closest street line to the user's location. The minimum spanning tree route was calculated using an array-based version of Prim's Algorithm. It is visualized by showing the MST connecting all evacuation centers with direct paths, useful for aircraft/helicopter map radar. Lastly, we implemented a K-nearest evacuation centers function. To optimize Dijkstra's algorithm, this function provides candidate evacuation centers to prevent calculating every single center in Philadelphia.

Figure 1 visualizes the structure of our code. An arrow going from a file means that it is used by the file the arrow is point to; for example, style.css points to index.html because index.html is using style.css.

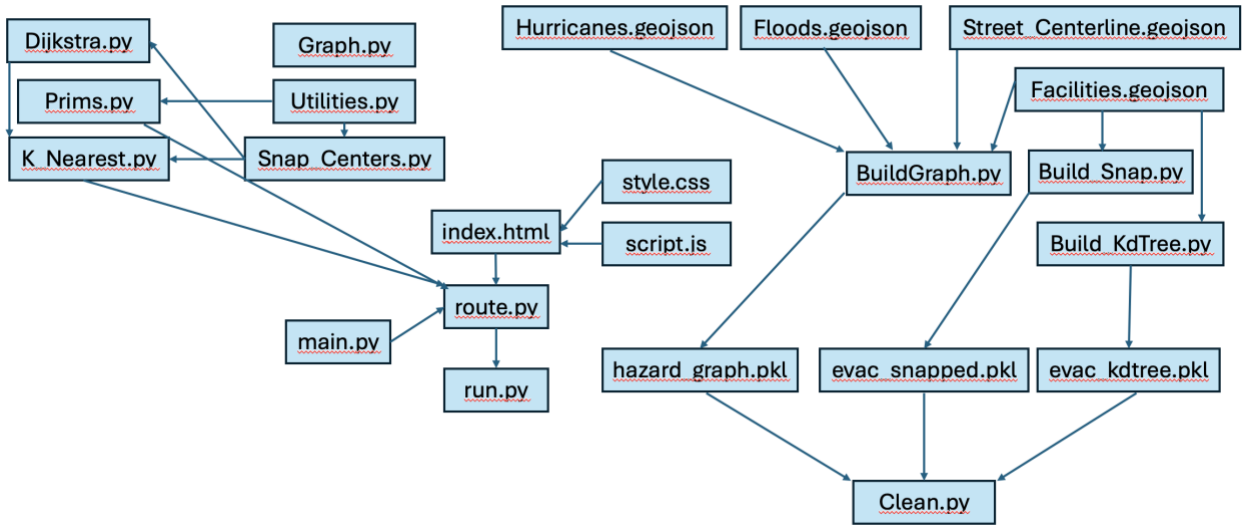

Figure 1: Code/File Structure Diagram of Application

In this diagram, we have multiple files. To begin, we used the BuildGraph.py file to create the hazard pickle file. The BuildGraph.py is quite robust, and it does not have any classes or functions. It simply reads in the raw data from the 4 GeoJson files from the publicly accessible open-source data websites. From there, the hazard pickle file is used with the raw data files in the main execution of the program. When the webpage is loaded, route.py automatically is called to retrieve user input, get K-nearest evacuation centers, complete Dijkstra's algorithm, build the minimum spanning tree using Prim's algorithm, and any other helper functions. Once the calculations are done, the mapFollium.py file is used. This file generates an HTML output to display the map of the Philadelphia region, with markers of the user's position, evacuation centers, and call calculated routes.

Description of Algorithms

We used two main algorithms for our project. Firstly, we used Dijkstra's algorithm as our shortest path algorithm. Dijkstra's algorithm is a greedy shortest path algorithm, and it is commonly used on map route applications to provide the shortest path to a destination from a source point. Our application checks a user's input point, and it checks if it is within the bounds of the Philadelphia region. Once the point is confirmed to be within the Philadelphia area range, it is used in the Dijkstra's algorithm to show the closest path from the user's location to an evacuation center. We followed a heap approach for Dijkstra's algorithm to improve the efficiency of this algorithm. In Dijkstra's, based on the starting node, the algorithm checks each of a node's neighbor nodes and chooses the shortest overall path distance to follow. The heap is used to add the current node neighbors to a heap and compare to find the minimum value

through the heap. Once we find the path from this algorithm, we visualize the path onto the map interface of our application.

Secondly, we used an array-based Prim Algorithm to build the Minimum Spanning Tree (MST). This algorithm returns the MST of a graph, and it uses a linear scan over candidate edges. An MST is the minimum edge-costs to connect all points of a given graph. We applied this algorithm to the graph of the evacuation centers to represent the routes between them all. This array-based approach for this algorithm was used over a heap-based version due to the size of our data. Although the time complexity for an array-based structure is O(V^2) compared to the heaps O(E log V), the evacuation center graph is complete, and the number of edges is approximately V^2, giving the heap no benefit. Once the MST is complete, the visual on the map shows the lowest distance cost to visit each evacuation center. Aerial maps can utilize this graph to follow paths between centers during emergencies.

Lastly, the K-nearest algorithm is used to speed up the search for its nearest neighbors and identify the candidate's evacuation centers before running Dijkstra's algorithm. This structure is a space-partitioning binary search tree to recursively search for neighboring nodes, resulting in an average O(log n) lookup time.

Verification of Algorithms

In our testing process, we tested the program with coordinates we know are within the Philadelphia area, exact locations of evacuation centers, and locations outside of Philadelphia to test our error-catching. Figure 2 shows the application ran with a location with 1 evacuation route. Figure 3 shows the application ran with a different location with 5 evacuation routes. Figure 4 shows the application’s results ran with a location outside of Philadelphia. Our

application finds the nearest edge location to the Philadelphia region, and it creates the routes based on the location closest to the outside location. In the Execution Results & Analysis section, the program is tested in our demonstration video with Center City Philadelphia as the location with 3 evacuation routes.

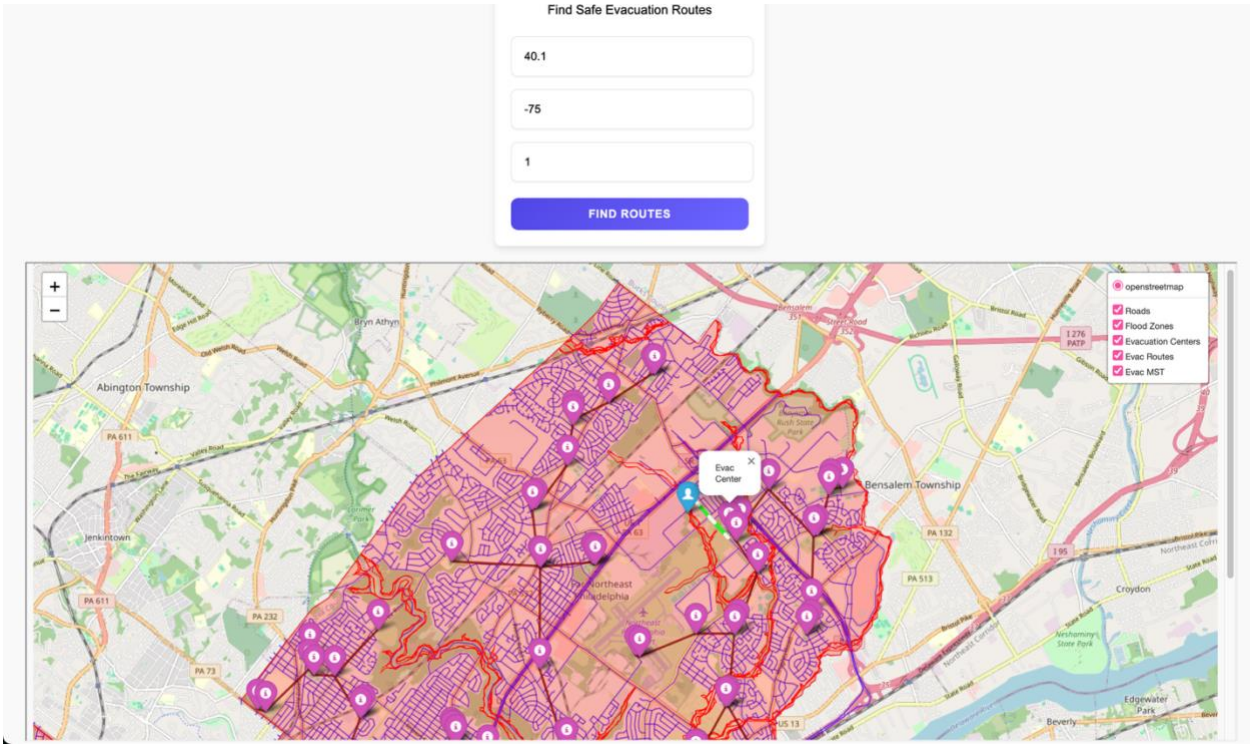

Figure 2: 1 Evacuation Route Results

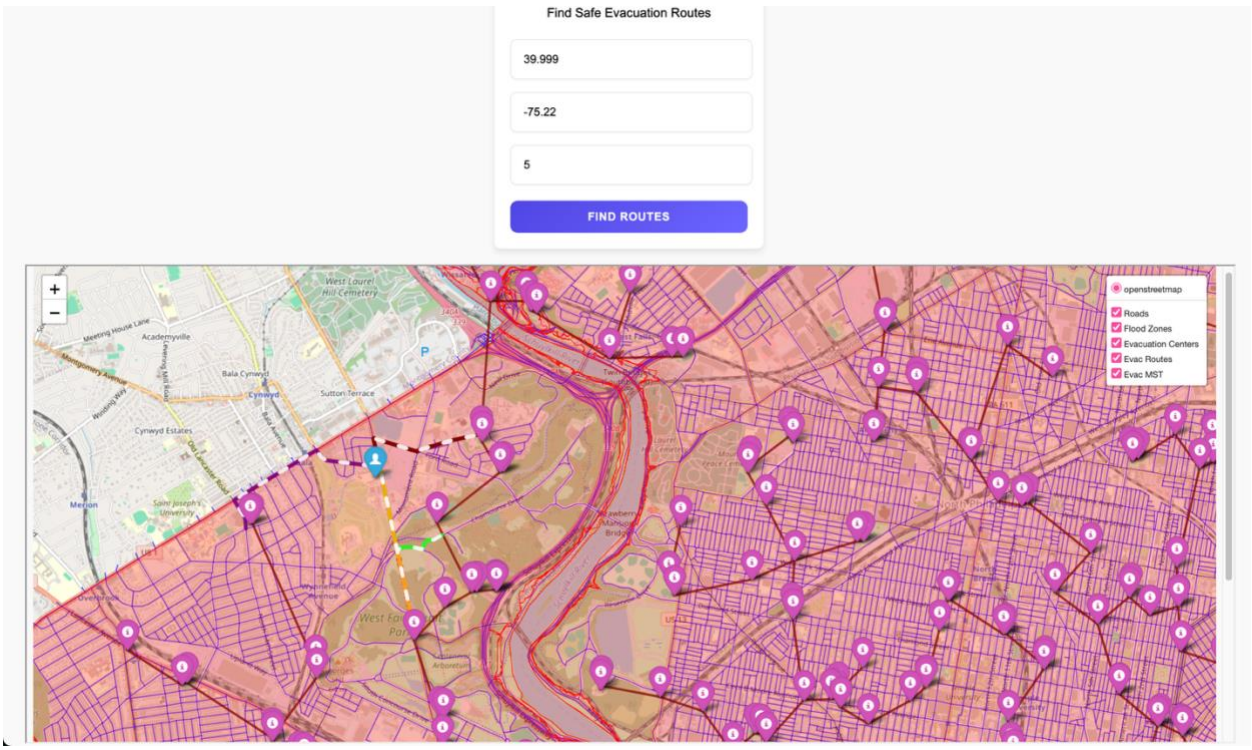

Figure 3: 5 Evacuation Routes Results

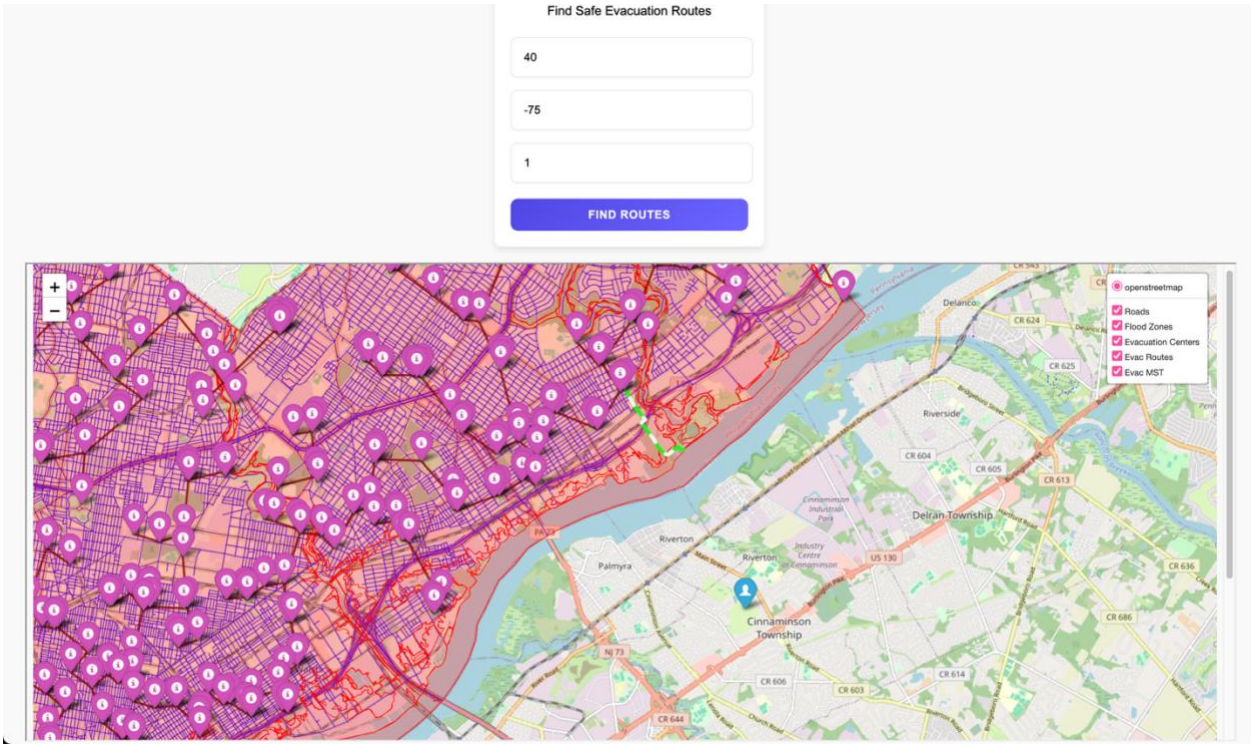

Figure 4: 3 Evacuation Routes with Location on the Edge of Philadelphia Location Results

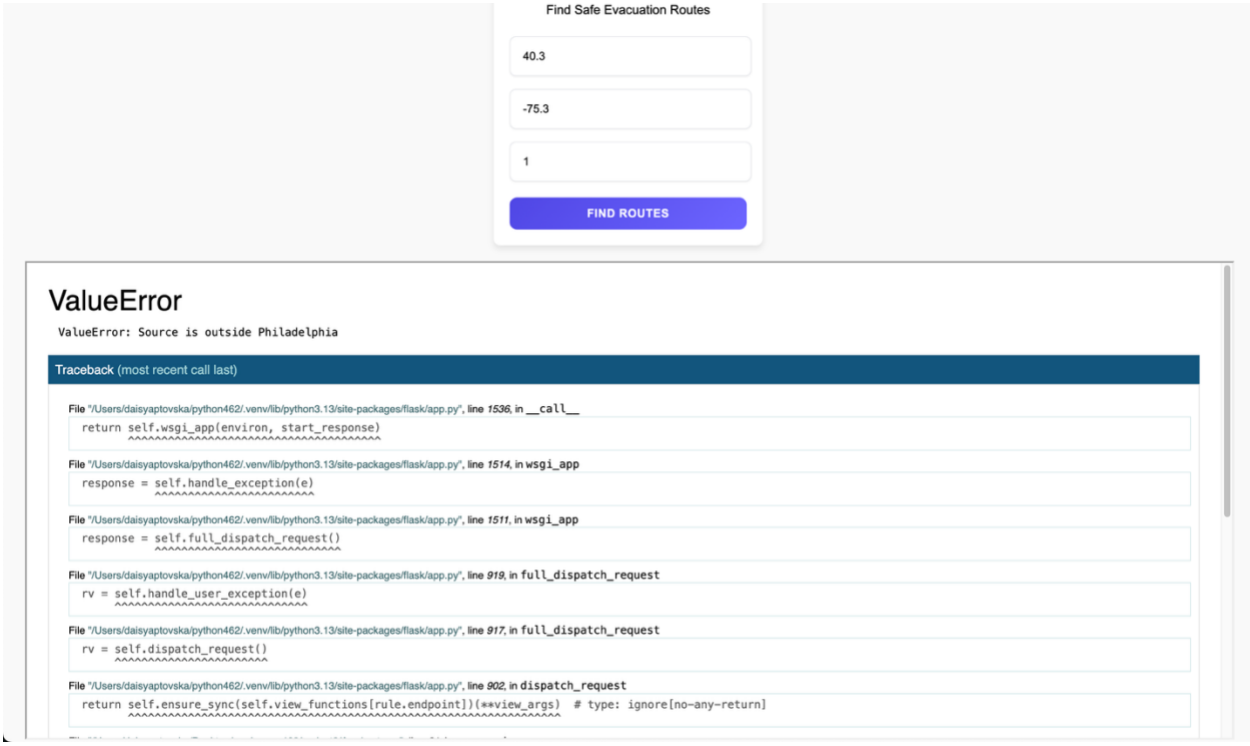

Figure 5: Error Message (Outside of Philadelphia)

Functionalities

In our application, we provide users with a map interface from Folium and GeoPandas libraries. This map interface displays the Philadelphia area, and when the program is run, users can input a starting point. From this starting point, Dijkstra's algorithm provides an outlined route on the interface. The direct Prim's algorithm is run to find the MST of all evacuation centers. Lastly, the road Prim's algorithm is run to find the MST between shortest road connections. Users can zoom in and out on the map to see the different routes mapped out in the generated HTML file. The paths of the 3 algorithm iterations are outlined in different colors. Users can access the results after executing main.py generated HTML file, which is hosted with Flask onto their local browser, viewing the visualized results based on the data used in respect to

their location. These visualizations are organized with an interactive zoom feature, interactive pinned locations, and color-sorted paths/routes of the visualization of the algorithms.

Execution Results & Analysis

After our verification of our algorithm's functionality with multiple examples, we created the demonstration video at the following URL link: https://psu.mediaspace.kaltura.com/media/CMPSC+463+Project+2+Demo+Video/1_yhh7j028. This demonstration shows the core functionality of our program. We tested the application with a coordinate in Center City Philadelphia, and we found 3 evacuation routes based on this location. Our program shows a detailed map visualization of the following categories: roads, flood zones, evacuation centers, evacuation routes, and the aerial evacuation centers MST. Our results showed accurate routes to the K-nearest routes to K evacuation centers based on the location input. Additionally, we were able to display the map data accurately based on the open-source dataset we used from the Philadelphia websites. Lastly, we displayed an aerial view of the Prim MST for helicopter routes between evacuation centers for emergencies. The Prim MST follows a different method than the Dijkstra's/K-Nearest algorithm, using Euclidean distance across the points rather than following the street data for road connections.

Conclusion

In our application development, we created a real-world application for Dijkstra's algorithm and Prim's algorithm, utilizing K-nearest algorithm and libraries such as GeoPands, Folium, Shapely, and Flask. Our application works accurately, and it provides useful information to users in case of a weather emergency. In future works, we would be interested to see how this project can be scaled for larger regions or other cities. Additionally, a limitation we faced in our

development process was we could not deploy our website as a web service. We attempted to utilize Render, an application hosting service; however, the free version of Render only allows for 0.1 of CPU and 512 MB of cloud hosting. Since our application uses a large dataset and hosts a map interface system, we could not host the application within this scale. Our application is hosted at the following URL link, but the application does not properly execute due to the memory overload, lack of computational resource, and limit on the free instance type of Render: https://philadelphia-weather-evacuation-gps.onrender.com/. Overall, our application development was a success, creating a useful, real-world application based on weather crises.

References

Cassie, D. (2025, November 21). *Street centerlines*. OpenDataPhilly. https://opendataphilly.org/datasets/street-centerlines/

Chat GPT & Google Search AI Overview – Flask explanation, debugging suggestions, data validation, Render hosting

Chesneau, B. (n.d.). *Running gunicorn*. Running Gunicorn - Gunicorn 23.0.0 documentation. https://docs.gunicorn.org/en/stable/run.html

CMPSC 463 – Google Colab Notebook 12

Deploying on render. Render. (n.d.-a). https://render.com/docs/deploys

Dijkstra, E. W. (2025, November 6). *Dijkstra's algorithm*. Wikipedia. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

FEMA Map Specialist. (n.d.). FEMA Hazard and Risk Information Platform. https://hazards-fema.maps.arcgis.com/home/index.html

Gaturu, B. (2020, February 28). *Gunicorn command not found, but it's in my requirements.txt*. Stack Overflow. https://stackoverflow.com/questions/60452279/gunicorn-command-not-found-but-its-in-my-requirements-txt

GeeksforGeeks. (2025a, July 23). *Search and insertion in K dimensional tree*. https://www.geeksforgeeks.org/dsa/search-and-insertion-in-k-dimensional-tree/

GeeksforGeeks. (2025b, October 24). *Flask - (creating first simple application)*. https://www.geeksforgeeks.org/python/flask-creating-first-simple-application/

GeoPandas Developers. (n.d.). *Geopandas 1.1.1*. GeoPandas 1.1.1. https://geopandas.org/en/stable/

Index of /data/international-best-track-archive-for-climate-stewardship-ibtracs/v04r01/access/csv/. Index of /data/international-best-track-archive-for-climate-stewardship-ibtracs/V04R01/access/CSV/. (2025, November 25). https://www.ncei.noaa.gov/data/international-best-track-archive-for-climate-stewardship-ibtracs/v04r01/access/csv/

Kartik. (2025, August 29). *Prim's algorithm for minimum spanning tree (MST)*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/prims-minimum-spanning-tree-mst-greedy-algo-5/

Quodomine, R. (2025, November 21). *City facilities (master facilities database)*. OpenDataPhilly. https://opendataphilly.org/datasets/city-facilities-master-facilities-database/

Quickstart. Quickstart - Flask Documentation (3.1.x). (n.d.). https://flask.palletsprojects.com/en/stable/quickstart/#a-minimal-application

Shapely 2.1.2 documentation. Shapely. (n.d.). https://shapely.readthedocs.io/en/stable/

Troubleshooting your deploy. Render. (n.d.). https://render.com/docs/troubleshooting-deploys

user1328021. (2013, May 15). *JSONDECODEERROR: Expecting value: Line 1 column 1 (char 0)*. Stack Overflow. https://stackoverflow.com/questions/16573332/jsondecodeerror-expecting-value-line-1-column-1-char-0

W3schools.com. W3Schools Online Web Tutorials. (n.d.-a). https://www.w3schools.com/dsa/dsa_algo_mst_prim.php

W3schools.com. W3Schools Online Web Tutorials. (n.d.). https://www.w3schools.com/html/html_forms.asp

Yashaswini916. (2025, July 23). *How to run a flask application*. GeeksforGeeks. https://www.geeksforgeeks.org/python/how-to-run-a-flask-application/