National Park Mapping

HCI: 584 – Developer Doc

Kaitlin Pohren

**Overview:**

The idea for this project is to process weather data, as well as the distance from your location to all National Parks in the USA*.* This is shown on an interactive map with markers for each location, and pop-ups when clicked with additional park information. This would be used by travelers looking to plan a trip to a National Park in the US.

**Specification:**

External Data Used:

* Scrape Data from NPS Stats: <https://irma.nps.gov/Stats/Reports/Park/ARCH>
* Scrape weather data from NCEI <https://www.ncei.noaa.gov/access/monitoring/climate-at-a-glance/city/time-series>
* Use of Flask and Folium to create web interface

Task Vignettes

* 1. Task One: Set initial location
     + Have user input their zip code
     + User Interfacing:
* User reaches web page, inputs location (Check validity)
* User gets map of all parks zoomed on their location

1. Task Two: View and analyze results
   * Clicking on markers on map will reveal more information
     + 1. plot of precipitation
       2. Show plot of temperatures
       3. Show distance from user
   * Interfacing:

* User views map within travel range defaulted
* Green markers indicate NP
* User hover and click for two plots pop-up with data, distance and park title

Technical Information

1. def haversine\_distance(lat1, lon1, lat2, lon2):
   * calculates the distance between two locations
   * Pulls in each park location and user location
2. def city\_location():
   * Main feature
     + Collects user input for location
     + Processes through csv data for np locations
     + Creates map of user & np locations
     + Adds in additional information on each of the parks

**Install/deployment/admin issues:**

User will need the following installed to use the app:

* pandas
* folium
* requests
* math
* matplotlib.pyplot
* IPython.display
* Flask

(**End) User interaction & Walkthrough**

* + Recap:
    1. Once the user clicks run on the code, the link to the webpage will show up in the terminal for them to click
    2. The webpage will bring them to a home screen where they can input their location and click to enter the map
       - The UX here has been improved so there is a drop down of state abbreviations instead of having to remember each state abrev.
       - Also this is mistake proofed so that if an incorrect location is entered they will get an error message and directed to hit the return to previous screen arrow

A screenshot of a computer error

Description automatically generated

* + 1. After clicking, they will come to a page with a purple marker (their location) & green markers (national parks)
       - They can hover over to read the park name, and if they click on the markers a pop-up will open showing more information on each park
       - They can also zoom in & out on the map in the top right corner

A map with green and purple pins

Description automatically generated

Details:

* + The first chunk of code is called upon later, but it calculates the distance between two locations
  + In def city\_location():
    - Asks the user for the location and pulls it in as city & state abbreviation
    - Runs it through a website to get the latitude & longitude data for the location
    - Checks if entry is valid, and if it is saves the user location to *lat* & *lon*
    - Opens the npdata.csv as a dataframe to parse through all national park information
    - Creates a new column called *miles* to save the calculated distances and sets all values to zero
    - Runs through each row in the npdata.csv, pulls out the location and sends it to the def haversine\_distance() to complete the distance calculation and save the distance in miles in the dataframe
    - Sorts the dataframe by the closest ones & reindexes it
    - creates map with folium centered around the user location
    - reformats the user location into a string for print out purposes
    - This for loop then runs through the dataframe (ie. each park) and for each one does a series of steps
      * Creates a marker at the location
      * Creates a hover bubble stating the location
      * Creates a popup so when the marker is clicked, the user can see the park name, distance from the user, a link to the national park service webpage to learn more, a precipitation graph, and a average temperature graph
    - Then launches the map
    - Else returns an error message if the user did not type a valid location
    - Final return calls the html file in templates to create the nice user input page
    - Then runs the app

Other code:

* + - If the user did want to add/change the data shown on the map they could do that through editing the npdata.csv, but would not get temperature or precipitation plots to show. To do this the following would need done:
      * Add/change data as needed in the precipdata.csv & tempdata.csv for precipitation & temperature plots respectively
      * Open maincode\_plotting.py
        + This code iterates through both the temp & precipitation csv’s and creates a the bar graphs shown in the app pop-ups
        + First the code creates all the temperature plots, distinguishing the red as average high temp, and blue for average low temp of each month
        + Then saves and closes the graph, and the code moves on to do the same for the precipitation plot but with only the average water level of each month shown in a slightly different blue
      * The plots are saved in the general folder, but any new plot file names would need to be added to the npdata.csv, the file would need added to the static folder, and renamed so no spaces exist in the name

**Known Issues: You should mention any issues you know about (or suspect)**

* Minor: anything that's a minor bug (non-breaking) that could easily be fixed given some time
  + If the user enters a town name that is outside the USA it sometimes will find a city somewhere in the world with that name and get the distance from that place
  + If the user does try to add a new park to the map, it is quite a bit of work to get plots created and find the data for the temperature and precipitation (all manual)
    - Might be best to handle this through doing a developer update anytime a new national park is created (not very often but does happen)

**Future work:**

* Automatically select starting location
* Add general summary of park
* Add picture of the park
* Add link directly to the National Park service webpage for that park vs general
* Add filters
* Add total annual visitor information
  + Possibly plot and show at the user location if clicked on to give high level overview
* Select up to three parks to compare on a different page