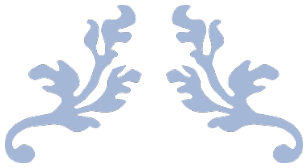


INTERACTIVE MAP CREATOR

ECE 220 Spring 2022: Honors Project Final Report

ARHAN GOYAL
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
NetID: arhang2 | UIN: 663017399

Table of Contents

Acknowledgements

I would like to thank the ECE 220 course staff for giving me the opportunity to work on a project as part of the Honors section of the course. I would also like to thank Prof. Yuting W. Chen (*Teaching Associate Professor, Electrical and Computer Engineering, University of Illinois at Urbana-Champaign*) for her support during the ideation stage of the project and for making me love ECE even more through her wonderful lectures.

Link to GitHub Repository

https://github.com/arhangoyal/interactive-map-creator

Link to Meteorite dataset: https://www.kaggle.com/datasets/nasa/meteorite-landings

---

Technical Skills Required

Languages: Python3
Frameworks: NumPy, Pandas, Folium, WebBrowser, OS
Developer Tools: Visual Studio Code, Git

Abstract

Maps are wonderful. They can take a huge CSV file and present it in a condensed form, making it easier for the human brain to process the data. Interactive maps are even better. They allow the viewer to interact with data on a fundamentally new level. Data scientists, meteorologists, historians, and several other professions involve analyzing decades of data. Many of these applications involve studying trendlines or even the concentration of events from a particular time period in various regions of the world. For instance, people studying meteorites may want to analyze heatmaps or individual meteorites from different time periods.

As someone who loves visual and interactive ways of learning and working, I have created a tool to create an interactive heatmap for any inputted dataset with map data (latitude, longitude). To make the interface clean for the user, I designed a solution to identify the latitude and longitude columns in the input dataset and use it to create an interactive heatmap. I then overlay the heatmap with individual datapoints (clickable icons with an information wall) based on a filter which the user creates. The idea is that while we may want to observe the pattern shown by a neat heatmap, we might only want to see information about specific "important" plot points. I feel this is a beautiful way to view the pattern/general trend and important data points together.

Introduction/Motivation

I have always loved fun, innovate, and interactive methods of learning. I believe this method of learning was inculcated in me by my parents, "Time Life" books that I used to read with their signature "Walter" (scan code on a little device to play text as audio with additional facts), and the hordes of science TV shows that I have watched over the years.

A couple of years ago in grade 10, I completed a few courses from The University of Michigan's online course "Applied Data Science with Python Specialization" on Coursera, where I was amazed by how one dataset can be represented in different ways. Every data representation method has its own bias. I also learnt about design and graphical heuristics for creating effective visualizations.

Then one day, my friends and I were talking about different ways to represent same dataset. That is where I suggested an interactive map. After the discussion, I designed an interactive map for the meteorite dataset I have included in the repository. I then decided to take my representation down another path by generalizing it to work with any dataset.

Implementation Details

I have split the program into two files: `interactive-map-creator.py` and `myfunctions.py`. The former contains the main program, and the latter is a library I have creator to make my main program neater.

The program contains the following sections (the functions I have referenced have been created by me and exist in the file `myfunctions.py`):

1. Clean Up Data (meteorite dataset specific):

   My interactive map creator expects a fairly cleaned up dataset, but the dataset from NASA that I am using for my demo requires some more cleaning. That is why I had to add this section in. Here, I:
   a. Remove rows where mass is NaN
   b. Remove rows where meteorite type is not valid
   c. Drop the last column (GeoLocation) because it is redundant (latitude and longitude data also exist elsewhere in the dataset)
   d. Sort rows in ascending order of year (time): just to make our data look nice!
   e. Convert mass data from gram to kilogram

2. Find Coordinate Columns:

   The function `find_coordinate_cols()` takes the path to a CSV file as an argument. It then:
   a. Creates a view on each column. Creating views is an efficient way to work with large datasets in pandas.
   b. Figures out whether the column is an integer/float column or not. The test I have devised for this is randomly testing 100 to 200 entries in index range [100, number of rows – 100) for being an int/float. If at least 95% of the entries are int/float, I test for latitude/longitude values. Also note that the 100 row padding that I have added is to avoid any headers in the dataset.

c. If the column is an int/float column, check if it is a latitude or longitude column or not. For latitude, at least 80% of all values in the column must be in the range [-90, 90]. For longitude, at least 80% of all values in the column must be in the range [-180, 180].
d. Rename the latitude and longitude columns to "Latitude" and "Longitude" after columns are found.
e. Push the updated dataset with renamed columns to a new CSV file called `coord-data-found.csv` inside the `data` folder of the repository ("`./data/coord-data-found.csv`").

3. Filter Out-of-Bounds Coordinate Data:

   The function `find_coordinate_cols()` takes the path to a CSV file as an argument. It then removes:
   a. Rows with latitude and longitude values outside ranges [-90, 90] and [-180, 180] respectively.
   b. Rows with NaN entries in any column.
   c. It then saves the updated dataframe to a new CSV file called `coord-filtered-data.csv` inside the `data` folder of the repository ("`./data/coord-filtered-data.csv`").

4. Take Input Filters for Heatmap and Plot Points:

   The function `get_map_filter()` takes a data frame and a message as arguments. It uses the message to ask the user for a column name and range of values to include from the column. Since column names can be complicated, it prompts the user again and again till the user gets the column name right.

   In the program, I call this function twice: once to get a filter which will apply to both the heatmap and the plot points, and once to get a filter which will only apply to the plot points.

   With the filters ready, I create masks on the dataset and the masks are ready for plotting.

5. Plot Individual Plot Points Using Both Filters:

   First, I create a map object using the Folium library. I then use the same library to plot the individual plot points. To make the map look nicer, I have scaled and used a custom meteorite icon for the plot points since I will be using the meteorite dataset for the demo.

   I then create an information wall (`info_wall`) using all the column name and corresponding value of that data point. The point is then added to the map using the coordinates from the coordinate columns I found earlier.

6. Add a Heatmap to the Same Map Object Using Filter 1 only:

   I add a heatmap to the same map object using Folium.

7. Save and Open Map in Default Web Browser (Safari for me):

   The map object is saved as an HTML file in the output folder ("`./output/map.html`"). Then, I use the OS library to extract the full, real pathname of the file and automatically open it in my default web browser (Safari) using the Webbrowser library.

Program: interactive-map-creator.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import plotly.express as px
from IPython.display import display
from folium.plugins import HeatMap
from folium import plugins, Map, Marker, features, Icon
from folium.plugins import HeatMap
import webbrowser
import os
from random import randint

# myfunctions.py contains the functions I have written for the interactive map creator
from myfunctions import find_coordinate_cols, clean_coord_data, get_map_filter



# main

infile = input("Enter full/relative path to (CSV) dataset on system: ") #
./input/meteorite-landings.csv

# clean data (meteorite data specific)
dirty_df = pd.read_csv(infile, delimiter=",")
filtered_mass = ~pd.isna(dirty_df["mass"])
filtered_type = ((dirty_df["nametype"] == "Valid"))
dirty_df = dirty_df[filtered_mass & filtered_type]
dirty_df.drop(["GeoLocation", "nametype"], axis=1, inplace=True)
dirty_df = dirty_df.sort_values(by="year", ascending=True)
dirty_df["mass"] = dirty_df["mass"].div(1000) # convert gram to kilogram. This is
specific to the meteorite dataset
dirty_df.to_csv("./data/pre-cleaned-data.csv", index=False)

# Find coordinate columns, rename them, and filter out data outside valid coordinate
range
find_coordinate_cols("./data/pre-cleaned-data.csv")
clean_coord_data("./data/coord-data-found.csv")


# Plot data

# open coordinate filtered data file
clean_df = pd.read_csv("./data/coord-filtered-data.csv", delimiter=",")
```

```
# input: filter 1 (col1) for heatmap and plot points and filter 2 (col2) for plot
points only
col1, col1_min, col1_max = get_map_filter(clean_df, "Enter name of column 1
(int/float) to filter out data: ")
col2, col2_min, col2_max = get_map_filter(clean_df, "Enter name of column 2
(int/float) to filter out data for individual plot points only: ")

# create view on dataframe according to inputted filters
mask1 = (clean_df[col1] >= col1_min) & (clean_df[col1] <= col1_max) & (clean_df[col2]
>= col2_min) & (clean_df[col2] <= col2_max)
mask2 = (clean_df[col1] >= col1_min) & (clean_df[col1] <= col1_max)
include = clean_df[mask1]

# updated mask for heatmap
mask2 = clean_df[mask2]


map = Map(location=[0, 0], zoom_start=2, control_scale=True)


# plot points using mask1
for index, row in include.iterrows():
    icon = features.CustomIcon("https://i.imgur.com/YCwSiQa.png",
icon_size=(53.75,66.25)) # custom icon
    col_names = [col_name for col_name in clean_df.columns.values.tolist()]
    info_wall = ""
    for i, name in enumerate(col_names):
        info_wall = info_wall + name + " " + str(row[name]) + "\n"
    Marker([row["Latitude"], row["Longitude"]],
    popup = info_wall,
    icon=icon ).add_to(map)

# plot heatmap using mask2
mask2 = mask2[['Latitude', 'Longitude']]
map_data = [[row['Latitude'],row['Longitude']] for index, row in mask2.iterrows()]
HeatMap(map_data).add_to(map)

# save map as html file and open in browser
map.save("./output/map.html")
webbrowser.open('file://' + os.path.realpath("./output/map.html")) # open file in
default browser
```

Program: myfunctions.py

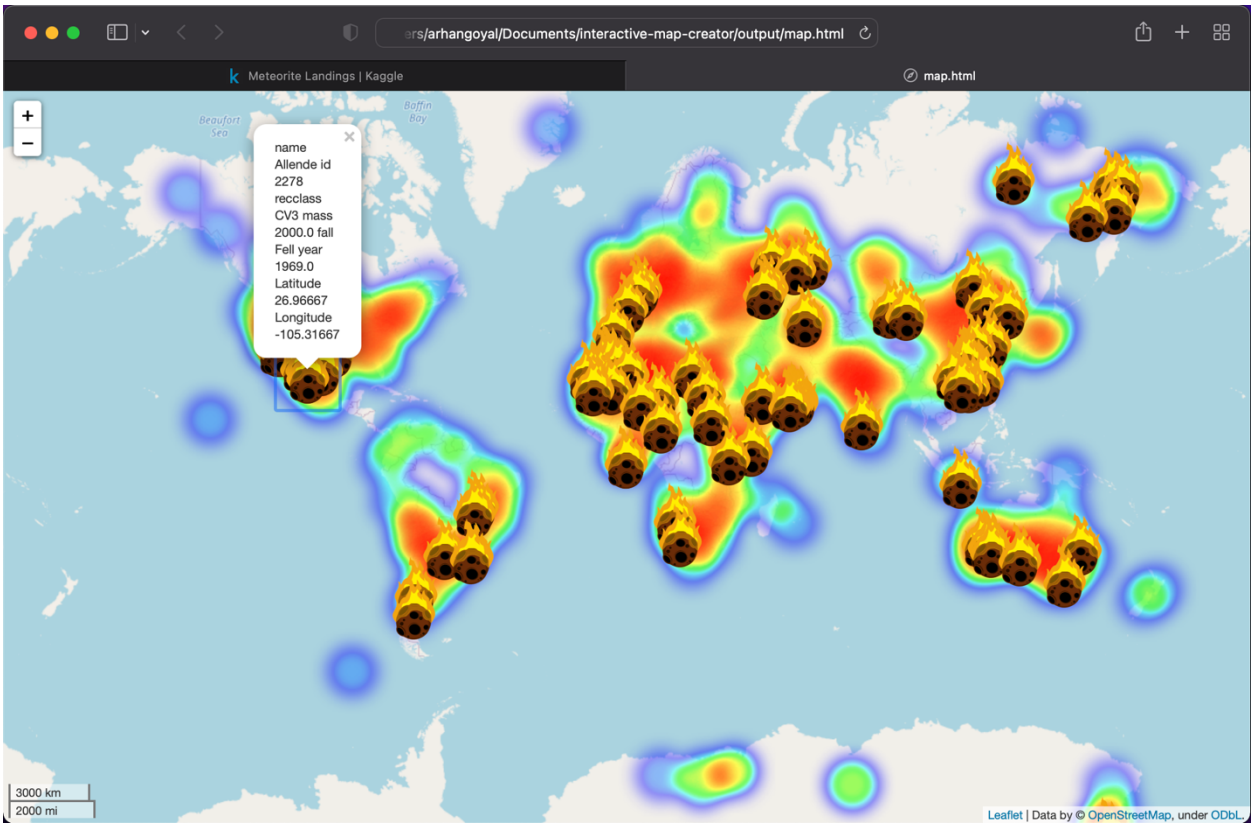
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import plotly.express as px
from IPython.display import display
from folium.plugins import HeatMap
from folium import plugins, Map, Marker, features, Icon
from folium.plugins import HeatMap
import webbrowser
import os
from random import randint


def find_coordinate_cols(csv_infile):
    # Identify latitude and longitude columns (Assume latitude will be before
longitude)
    df = pd.read_csv(csv_infile, delimiter=",")
    # meteorites = meteorites1.dropna(axis=0, inplace=False)
    num_row, num_col = df.shape[0], df.shape[1]
    lat_idx, lon_idx = -1, -1
    # coord_col = []

    for col_idx in range(num_col):
        col = df.iloc[: , col_idx] #create a view on column at index col_idx

        # Check if most entries in column are likely to be int/float
        int_percent = 0
        num_tests = randint(100, 200)
        for test_num in range(num_tests): #test randomly on 100 to 200 entries in
column being integers
            entry = col[randint(100, num_row-100)] #pick any entry between index 100
and index num_row-100 to avoid any padding in data
            if isinstance(entry, int) or isinstance(entry, float):
                int_percent += 1 #number of int/float entries
        int_percent = int_percent * 100 / num_tests # convert to percentage: percent
of tests which were int/float

        if int_percent > 95:
            # Check for latitude column
            lat_filter = col[(col >= -90) & (col <= 90)]
            # If more than 80% values are in our coordinates range
            if lat_filter.shape[0] * 100/num_row > 80:
                lat_idx = col_idx
```

```
            # Check for longitude column
            lon_filter = col[(col >= -180) & (col <= 180)]
            # If more than 80% values are in our coordinates range
            if lon_filter.shape[0] * 100/num_row > 80:
                lon_idx = col_idx

    # Rename the coordinate columns to "Latitude" and "Longitude"
    df.rename(columns={ df.columns[lat_idx]: "Latitude", df.columns[lon_idx]:
"Longitude" }, inplace = True)
    # df.rename(columns={ df.columns[coord_col[0]]: "Latitude",
df.columns[coord_col[1]]: "Longitude" }, inplace = True)

    # Push the data to outfile
    df.to_csv("./data/coord-data-found.csv", index=False)


def clean_coord_data(csv_infile):
    df = pd.read_csv(csv_infile, delimiter=",")
    filtered_coords = (df["Longitude"] >= -180) & (df["Longitude"] <= 180) &
(df["Latitude"] >= -90) & (df["Latitude"] <= 90)
    filtered_df = df[filtered_coords]
    filtered_df.dropna(axis=0, inplace=True)
    filtered_df.to_csv("./data/coord-filtered-data.csv", index=False)


def get_map_filter(df, message):
    col1 = "-1"
    cols_lower = [col_name.lower() for col_name in df.columns.values.tolist()]
#convert all column names to lower case for comparison
    while col1 not in cols_lower: # keep asking for input till user matches a column
name
        col1 = input(message).lower() #convert input to lowercase to compare with
lower case column names
    col1_min = float(input(f"Input minimum value for {col1}: "))
    col1_max = float(input(f"Input maximum value for {col1}: "))
    return (col1, col1_min, col1_max)
```

Sample Output

Here is a picture of the map for the time period (years) [1960, 2016] with mass (kg) in range [100, 100000]:

Note: The heatmap and icon sizes adjust themselves as users zoom in and out of the image rendered above!



Future Works

1. I would like to create and use a training set to figure out the ideal latitude and longitude value percentage buffers (currently 80%) to maximize accuracy when dealing with different datasets.

2. I would also like to figure out a way to make the information wall (which appears on clicking map icons) more pleasant to look at.

3. I also had an idea to create a more advanced heatmap which can tell the user the density of datapoints in a particular region around the mouse pointer (say 2% radius of image size in current zoom level).