Introduction to the Code

After first learning about linear regression in class, I wonder if it could be applied to other fields other than just basic information. I learned that regression analysis could be applied to a multitude of different fields and was extremely relevant in machine learning and other statistical applications. My project uses 3-dimensional linear regression to predict the probability of a flood given the cumulative rainfall and elevation. There are four main parts to my program. The first part are the functions that call the api's that get the datasets of historical data from a website called oikolab.com. The second part are the functions that get the evaporation rate, which is crucial in calculating cumulative rainfall. The third part is the function that formats the historical data into a usable list that can then be read by the linear regression function. The fourth part is the function that gets a 3-dimensional line with z-values from 0-1, using linear algebra properties.

Description of the inputs and outputs

There are limited user inputs in the project, since most of the project involves getting historical weather information from APIs and using linear regression to get the probability of the flood. The majority of the outputs is the json file from the weather API, the evaporation rate, the formatted list, and the flood probability slopes and z-axis. The inputs for the functions are usually scalars especially when getting the evaporation rates. Most of the functions utilize a formatted list that holds the historical weather data to find the weather probability 3-d line.

Description of the user-defined functions

```
elevation(longitude, latitude)

This code returns the elevation from an API given user inputed longitude
and latitude.

hist_data(startDate, endDate, latitude, longitude):
This function returns a json from a historical weather api, it needs the
start date, end date you want, latitude, and longitude.
```

```
evap_rate(Temp, altitude, humidity, latitude, hist_Tdew=0)
Based on two research papers, this function returns the evaporation rate
given the temperature, altitude, humidity, latitude, and dew temperature.
If dew temperature is not given another research paper was used to get the
dew temperature, given the humidity and tempiture.

historical_datasets()
This is the backbone of the program. This function gets the json file from
the weather api, then formats it into a list and then further refine the
information to get just 3 columns for the linear analysis: cumulative
rainfall, which takes in account rainfall and evaporation rate, altitude,
and t/f values if there was a flood or not.

cumulative_rainfall(past_rainfall, temp, altitude, humidity, latitude)
This function allows the users to input a list of past rainfall,
temperature, altitude, humidity, and latitude to get the cumulative
rainfall. This takes into account the evaporation rate of the water, so it
better predicts flooding.

prediction_formula(data_list=0)
This is the function where the actual linear regression is done. Using a
stacked array, it creates a coefficient matrix that is then used to get
the linear approximation using the "least squares" method. Since the
z-axis values will only be between 0-1 then the linear function will
output a x-slope, y-slope and z-axis that can be used as a formula and
will return a decimal to represent the probability.

prediction_eff_z(cumulative_rainfall, altitude, m1, m2, d)
This just puts together the user inputs cumulative rainfall values and
altitude to get the flood probability. The eff of this function means that
it is efficient, since the user directly gives the slopes and z-axis in
the input of the function, meaning that the historical datasets and
probability functions don't need to be constantly called.

prediction_formatted_eff(cumulative_rainfall, altitude, m1, m2, d)
This function formats your results from the probability in a clear and
nice way.

demo_flood_prob_location(cumulative_rainfall, altitude)
```

This function combines the prediction_formula and prediction_eff_z together and just returns the m1, m2, d, and z values.

```
demo_cumulative_rainfall()
```

This function demos the cumulative rainfall function.

```
demo_flood_prob_values()
```

This function outputs a ton of different cumulative rainfall and elevation values to showcase the different probability values.

```
import numpy as np
import requests
import json
from geopy.geocoders import Nominatim

def elevation(longitude, latitude):
    data =
requests.get(f"https://api.open-elevation.com/api/v1/lookup?locations={lon
gitude},{latitude}")
    file = data.json() #this a dictonary
    elevation = file['results'][0]['elevation']

    return elevation

# def floodTF(location):
def hist_data(startDate, endDate, latitude, longitude):
    my_headers = {'Cache-Control': 'no-cache',
'api-key':'3ccbe94c35b4443ba14763606d2dd58e'}
    response =
requests.get(f"https://api.oikolab.com/weather/?param=total_precipitation&
param=temperature&param=dewpoint_temperature&param=relative_humidity&start
={startDate}&end={endDate}&lat={latitude}&lon={longitude}&api-key=3ccbe94c
35b4443ba14763606d2dd58e&resample_method=mean&freq=D&format=json",
my_headers)

    return response.json()

geolocator = Nominatim(user_agent="Pranav Srisankar - Ind Project Engr
133")

def coords(address):
    try:
        location = geolocator.geocode(address)
    except:
        return "Error: Location does not exist."
    # Doesn't need to be precise since weather data is rather general over
an area
    try:
        return [round(location.latitude,3), round(location.longitude, 3)]
```

```
    except:
        return "Error: Location does not exist."

def address(coords):
    try:
        location = geolocator.reverse(f"{coords[0]}, {coords[1]}")
    except:
        return "Error: Location does not exist."
    return location.address

import Data_scrapper as dasc
import matplotlib.pyplot as plt
import numpy as np
import math as m
import ast

def evap_rate(Temp, altitude, humidity, latitude, hist_Tdew=0):

    if (isinstance(Temp, (float, int))==True) and
(isinstance(altitude,(float, int))==True) and (isinstance(humidity,(float,
int)) ==True) and (isinstance(humidity,(float, int))==True) and
(isinstance(latitude,(float, int))==True) and
(isinstance(hist_Tdew,(float, int))==True):
        #Temp (C)
        #altitude (m)
        #humidity (0-100)

        gi = 243.12 #conversion constant
        B = 17.62 #constant
        if hist_Tdew !=0:

#http://irtfweb.ifa.hawaii.edu/~tcs3/tcs3/Misc/Dewpoint_Calculation_Humidi
ty_Sensor_E.pdf
            td1 = gi*(m.log(humidity/100)+((B*Temp)/(gi+Temp)))
            td2 = B-(m.log(humidity/100)+((B*Temp)/(gi+Temp)))
            Tdew = td1/td2 #dew tempiture

        else:
            Tdew = hist_Tdew
```

```
        Tm = Temp + .0006*altitude#temp ajustest for altitude

        e1 = (700*Tm/(100-latitude)) + (15*(Temp-Tdew))
        e2 = 80 - Temp
        Evaporation_rate = e1/e2 #mm/day

#https://www.sciencedirect.com/science/article/pii/0002157177900073
        return Evaporation_rate
    else:
        return "Please enter a int or float for all the values!!!"

def historical_datasets():


    location = ['Wilkes-Barre, PA', 'Florida Keys, FL', 'Denham Springs,
LA', 'Rapid City, SD', 'Bloomington, IN', 'Denver, CO', 'Helena, Montana',
"New York City, NY", "Annapolis, MD", "Seattle, WA", "Seattle, WA",
"Seattle, WA","Seattle, WA", "Seattle, WA", 'Helena, Montana', "Denver,
CO", 'Helena, Montana', "Denver, CO"]
    flood_start = ["2011-09-02", "2017-09-02", "2016-08-08", "1972-06-07",
"2008-06-05", "2021-06-01","2018-07-01","2011-05-01", "2008-11-01",
"2001-11-01", "1998-12-01","1999-12-01","2016-12-01", "2017-08-01",
"2017-08-01", "1982-07-01", "1974-07-01"]

    #flood t/f
    flood_tf =
[[0,0,0,0,0,0,1,1,1,1,1],[0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0],[0,0,0,0,0,1,
1,1,1,1,1,1,1],[0,0,1,1,1,1],[0,0,0,1,1,1,1,1,1,1,1,1,1],[0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],[0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]]

    flood_end = ["2011-09-12", "2017-09-18", "2016-08-20", "1972-06-12",
"2008-06-17", "2021-06-30",
"2018-07-31","2011-05-31","2008-11-30","2001-11-30",
"1998-12-31","1999-12-31","2016-12-31", "2017-08-31",
"2017-08-31","1982-07-31", "1974-07-31"]

    #historical_data_names=['day case number', 'location', 'longitude',
'latitude', 'elevation','rainfall(mm)','temp(C)','dewpoint
temp(C)','humdity(0-1)', 'evap rate', 'flood happend (0 or 1)']
    #data = pd.DataFrame(columns=historical_data_names)

    historical_data = []
    historical_data_pts = []
    for i in range(len(flood_start)):
        n=0
        la, lo = dasc.coords(location[i])
        #lat.append(la)
        #long.append(lo)

        data = dasc.hist_data(flood_start[i],flood_end[i],la,lo)
        data = data['data']
        #needed to strip the strings off the data
        data = data.rstrip("'")
        data = data.lstrip("'")
        #coverts string to dict
        data = ast.literal_eval(data)


        cumulative_rainfall = 0

        for j in range(len(data['data'])):
            elevation = data['data'][j][2]
            rainfall = data['data'][j][4]*24
            temp = data['data'][j][5]
            dewTemp = data['data'][j][6]
```

```
            humidity = data['data'][j][7]
            evap = evap_rate(temp, elevation, humidity, la, dewTemp)


            cumulative_rainfall = cumulative_rainfall+rainfall-evap
            if cumulative_rainfall < 0:
                cumulative_rainfall = 0

            #historical_data_names=['day case number', 'location',
'longitude', 'latitude', 'elevation','rainfall(mm)','temp(C)','dewpoint
temp(C)','humdity(0-1)', 'evap rate', 'flood happend (0 or 1)']
            x=[n, location[i], lo, la, elevation, rainfall, temp, dewTemp,
humidity, evap, flood_tf[i][j]]
            y=[cumulative_rainfall, elevation, flood_tf[i][j]]

            historical_data.append(x)
            historical_data_pts.append(y)
            n = n+1


    return historical_data_pts, historical_data


def cumulative_rainfall(past_rainfall, temp, altitude, humidity,
latitude):

    if (isinstance(past_rainfall, list) == True) and (isinstance(temp,
list)==True) and (isinstance(humidity, list)==True) and
(isinstance(altitude,(float, int)) == True) and
(isinstance(latitude,(float, int)) == True):
    #either enter temp, humidit, rainfall as a list
    #altitude and latitude must a be a scalar
        rainfall = 0

        for i in range(len(past_rainfall)):
            evaporation = evap_rate(temp[i], altitude, humidity[i],
latitude)
            rainfall = rainfall + past_rainfall[i] - evaporation

        if rainfall >= 0:
```

```
            return rainfall
        else:
            return 0
    else:
        return "past_rainfall, temp, and humidity must be list. \naltitude
and latitude must be int or float"


def prediction_formula(data_list=0):

    #this one makes it so you dont have to re-load in the past data from
the API or you have your own list gathered from hisorical data
    #this assumes you already have your previous data, if not run the
prediction function
    #def make sure the rain fall is from the past 10 day
    #gets the data from the historical dataset
    if (data_list, list):
         hist_dp = data_list
    else:
        hist_dp, hist_d = historical_datasets()


    #turns the data to a numpy array
    hist_dp_np = np.asarray(hist_dp)

    #cumulative rainfall
    rain = hist_dp_np[:,0]
    #elevation
    elevation = hist_dp_np[:,1]
    #flood t/f
    tf = hist_dp_np[:,2]

    #this create an stacked array to create the Coefficint matrix
    A=np.vstack([rain, elevation, np.ones(len(rain))]).T

    #using linear alegebra properties of method of least squares, we get
the linear flood approximation
    # begin by clarifying exactly what we will mean by a “best approximate
solution” to an inconsistent matrix equation Ax=v
    m1, m2, d = np.linalg.lstsq(A, tf, rcond=None)[0]
```

```
    #m1 is slope for cumulative rainfall, m2 is the slope for elevation,
and d is the z-axis
    #z = (m1*cumulative_rainfall)+(m2*altitude)+d

    return m1, m2, d

def prediction_eff_z(cumulative_rainfall, altitude, m1, m2, d):
    #this just retunrs the z probility given the m1, m2, d and the
rainfall and altititude
    z = (m1*cumulative_rainfall)+(m2*altitude)+d

    return z

import Data_scrapper as dasc
import formulas as f
import datetime
import ast
def prediction_formatted_eff(cumulative_rainfall, altitude, m1, m2, d):
    #outputs a formatted answer of the flood probility given the x,y
slopes and z-axis
    print("cumlative rainfall slope (m1) = ", m1)
    print("elevation slope (m2) = ", m2)
    print("z-axis (d) = ", d)

    print("\n")

    print("probability of flood (z) = m1*cumulative_rainfall + m2*altitude
+ d")
    print("probability of flood (z) = ", m1,"*",cumulative_rainfall, " +
", m2, "*", altitude, " + ", d)
    #add up the slopes to get the z values which will be from 0-1
    z = (m1*cumulative_rainfall)+(m2*altitude)+d
    print("probability of flood (z) = ", z, "    or    ", z*100, "%")

    return z

def demo_flood_prob_location(cumulative_rainfall, altitude):
    data_list, teresa = f.historical_datasets()
```

```
    m1, m2, d = f.prediction_formula(data_list)

    z = prediction_formatted_eff(cumulative_rainfall,altitude,m1,m2,d)
    return z, m1, m2, d


def demo_cumulative_rainfall():
    rainfall_10_days_mm = [20,31,22,23,24,25,26,27,18,19]
    temp = [60,61,62,63,64,65,66,67,68,69],
    humidity = [.60,.61,.62,.63,.64,.65,.66,.67,.68,.69]

    location = "Purdue University"
    la, lo = dasc.coords(location)
    alt = dasc.elevation(lo, la)


    print("rainfall from past 10 days mm pre-set demo values\n",
rainfall_10_days_mm)
    print("tempetures pre-set demo values\n", temp)
    print("humidity pre-set demo values\n", humidity)

    print('\npre-set demo location: ', location)
    print('(latitude,longitude): ', la,lo)


    cumulative_rain = f.cumulative_rainfall(rainfall_10_days_mm, temp,
alt, humidity, la)
    print('demo cumulative rainfall value: ', cumulative_rain, '(mm)')
    return cumulative_rain

def demo_flood_prob_values():

    #oupts the probablity of flood, with different elevations and
cumulative ranifalls
    data_list, useless = f.historical_datasets()
    m1, m2, d = f.prediction_formula(data_list)

    print("\n\n\n\n")
    print(f.prediction_eff_z(10,1000,m1,m2,d))
    print(f.prediction_eff_z(10,500,m1,m2,d))
```

```
    print(f.prediction_eff_z(10,300,m1,m2,d))
    print(f.prediction_eff_z(10,200,m1,m2,d))
    print(f.prediction_eff_z(10,100,m1,m2,d))
    print(f.prediction_eff_z(10,100,m1,m2,d))
    print(f.prediction_eff_z(10, 10,m1,m2,d))
    print(f.prediction_eff_z(10, 5,m1,m2,d))
    print(f.prediction_eff_z(10, 0,m1,m2,d))
    print(f.prediction_eff_z(10, -10,m1,m2,d))
    print("\n\n\n")
    print(f.prediction_eff_z(7.5,1000,m1,m2,d))
    print(f.prediction_eff_z(7.5,500,m1,m2,d))
    print(f.prediction_eff_z(7.5,300,m1,m2,d))
    print(f.prediction_eff_z(7.5,200,m1,m2,d))
    print(f.prediction_eff_z(7.5,100,m1,m2,d))
    print(f.prediction_eff_z(7.5,50,m1,m2,d))
    print(f.prediction_eff_z(7.5, 10,m1,m2,d))
    print(f.prediction_eff_z(7.5, 5,m1,m2,d))
    print(f.prediction_eff_z(7.5, 0,m1,m2,d))
    print(f.prediction_eff_z(7.5, -10,m1,m2,d))
    print("\n\n\n")
    print(f.prediction_eff_z(5,1000,m1,m2,d))
    print(f.prediction_eff_z(5,500,m1,m2,d))
    print(f.prediction_eff_z(5,300,m1,m2,d))
    print(f.prediction_eff_z(5,200,m1,m2,d))
    print(f.prediction_eff_z(5,100,m1,m2,d))
    print(f.prediction_eff_z(5,50,m1,m2,d))
    print(f.prediction_eff_z(5, 10,m1,m2,d))
    print(f.prediction_eff_z(5, 5,m1,m2,d))
    print(f.prediction_eff_z(5, 0,m1,m2,d))
    print(f.prediction_eff_z(5, -10,m1,m2,d))

    print("\n\n\n")
    print(f.prediction_eff_z(5,1000,m1,m2,d))
    print(f.prediction_eff_z(5,500,m1,m2,d))
    print(f.prediction_eff_z(5,300,m1,m2,d))
    print(f.prediction_eff_z(5,200,m1,m2,d))
    print(f.prediction_eff_z(5,100,m1,m2,d))
    print(f.prediction_eff_z(5,50,m1,m2,d))
    print(f.prediction_eff_z(5, 10,m1,m2,d))
    print(f.prediction_eff_z(5, 5,m1,m2,d))
```

```
print(f.prediction_eff_z(5, 0,m1,m2,d))
print(f.prediction_eff_z(5, -10,m1,m2,d))

print("\n\n\n")

print(f.prediction_eff_z(1,1000,m1,m2,d))
print(f.prediction_eff_z(1,500,m1,m2,d))
print(f.prediction_eff_z(1,300,m1,m2,d))
print(f.prediction_eff_z(1,200,m1,m2,d))
print(f.prediction_eff_z(1,100,m1,m2,d))
print(f.prediction_eff_z(1,50,m1,m2,d))
print(f.prediction_eff_z(1, 10,m1,m2,d))
print(f.prediction_eff_z(1, 5,m1,m2,d))
print(f.prediction_eff_z(1, 0,m1,m2,d))
print(f.prediction_eff_z(1, -10,m1,m2,d))


print("\n\n\n")

print(f.prediction_eff_z(.5,1000,m1,m2,d))
print(f.prediction_eff_z(.5,500,m1,m2,d))
print(f.prediction_eff_z(.5,300,m1,m2,d))
print(f.prediction_eff_z(.5,200,m1,m2,d))
print(f.prediction_eff_z(.5,100,m1,m2,d))
print(f.prediction_eff_z(.5,50,m1,m2,d))
print(f.prediction_eff_z(.5, 10,m1,m2,d))
print(f.prediction_eff_z(.5, 5,m1,m2,d))
print(f.prediction_eff_z(.5, 0,m1,m2,d))
print(f.prediction_eff_z(.5, -10,m1,m2,d))

print("\n\n\n")

print(f.prediction_eff_z(.1,1000,m1,m2,d))
print(f.prediction_eff_z(.1,500,m1,m2,d))
print(f.prediction_eff_z(.1,300,m1,m2,d))
print(f.prediction_eff_z(.1,200,m1,m2,d))
print(f.prediction_eff_z(.1,100,m1,m2,d))
print(f.prediction_eff_z(.1,50,m1,m2,d))
print(f.prediction_eff_z(.1, 10,m1,m2,d))
print(f.prediction_eff_z(.1, 5,m1,m2,d))
```

```
    print(f.prediction_eff_z(.1, 0,m1,m2,d))
    print(f.prediction_eff_z(.1, -10,m1,m2,d))

    print("\n\n\n")

    print(f.prediction_eff_z(.00001,1000,m1,m2,d))
    print(f.prediction_eff_z(.00001,500,m1,m2,d))
    print(f.prediction_eff_z(.00001,300,m1,m2,d))
    print(f.prediction_eff_z(.00001,200,m1,m2,d))
    print(f.prediction_eff_z(.00001,100,m1,m2,d))
    print(f.prediction_eff_z(.00001,50,m1,m2,d))
    print(f.prediction_eff_z(.00001, 10,m1,m2,d))
    print(f.prediction_eff_z(.00001, 5,m1,m2,d))
    print(f.prediction_eff_z(.00001, 0,m1,m2,d))
    print(f.prediction_eff_z(.00001, -10,m1,m2,d))

    print("\n\n\n")
    print(f.prediction_eff_z(0,1000,m1,m2,d))
    print(f.prediction_eff_z(0,500,m1,m2,d))
    print(f.prediction_eff_z(0,300,m1,m2,d))
    print(f.prediction_eff_z(0,200,m1,m2,d))
    print(f.prediction_eff_z(0,100,m1,m2,d))
    print(f.prediction_eff_z(0,50,m1,m2,d))
    print(f.prediction_eff_z(0, 10,m1,m2,d))
    print(f.prediction_eff_z(0, 5,m1,m2,d))
    print(f.prediction_eff_z(0, 0,m1,m2,d))
    print(f.prediction_eff_z(0, -10,m1,m2,d))



# def prediction_GUI():
#     location = input("Enter Your Location: ")
#     la, lo = dasc.coords(location)

#     today = datetime.datetime.now()
#     d = datetime.timedelta(days = 10)
#     yesterday = datetime.timedelta(days = 1)
#     day_10 = today - d
#     yesterday = today-yesterday
```

```
#     yesterday = str(yesterday.date())
#     day_10 = str(day_10.date())



#     data = dasc.hist_data(yesterday, day_10, la, lo)

#     data = data['data']
#     #needed to strip the strings off the data
#     data = data.rstrip("'")
#     data = data.lstrip("'")
#     #coverts string to dict
#     data = ast.literal_eval(data)



#     rain = []
#     tempiture = []
#     humid = []
#     elevation = 0
#     for j in range(len(data['data'])):
#         elevation = data['data'][j][2]
#         rainfall = data['data'][j][4]*24
#         temp = data['data'][j][5]
#         humidity = data['data'][j][7]

#         rain.append(rainfall)
#         tempiture.append(temp)
#         humid.append(humidity)

#     c_rainfall = f.cumulative_rainfall(rain, tempiture, elevation,
humid, la)

#     m1,m2,d = f.prediction_formula()

#     prediction_formatted_eff(c_rainfall, elevation, m1, m2, d)

# def main():
```

```
#     # rainfall = demo_cumulative_rainfall()
#     # alt = 10
#     # demo_flood_prob(rainfall, alt)


#     #prediction_GUI()
# if __name__ == '__main__':
#     main()
```