Student Autograde Results

Student: suleanne
@2023-12-04 17:29:26.598384

Test ID	Description	Status	Points
test0	always true	Passed	1
test1	test get_earthquakes	Passed	1
test2	test get_plate_boundaries, keys	Passed	1
test2b	test get_plate_boundaries, values shape	Passed	1
test3	test get_coastlines	Passed	1
test4	test parse_earthquakes_to_np, lats	Passed	1
test5	test parse_earthquakes_to_np, lons	Passed	1
test6	test parse_earthquakes_to_np, depths	Passed	1
test7	test parse_earthquakes_to_np, mags	Passed	1
test8	test parse_earthquakes_to_np, times	Passed	1
total/10	...	...	10

Normalized grade: 10.0 / 10

-------- BEGIN STUDENT MODULE --------

```
import numpy as np
import pandas as pd
from datetime import datetime

#Function 1
def get_coastlines(coasts_file):
    '''
    Reads longitudes and latitudes of the world's coastlines from csv file.
    Returns individual 1D arrays of longitudes along world coastline and latitudes
    ... along world coastline.
    input: coasts_file
    outputs: lon_coast, lat_coast
    '''
    try:
        df = pd.read_csv(coasts_file)
        lon_coast = df.iloc[:,0]
        lat_coast = df.iloc[:,1]
        return lon_coast, lat_coast
    except:
        raise IOError


#Function 2
def get_plate_boundaries(plates_files):
    '''
    Reads csv file containing three columns (column 1: plate boundary name
    ... abbreviations, column 2: latitudes in degrees, column
    3: longitudes in degrees) and organizes the columns into a dictionary.
    Returns a dictionary where keys correspond to tectonic plate abbreviations, and
    ... values are 2D arrays containing longitudes in
    the first column and latitudes in the second column
    input: plates_file
    output: pb_dict
    '''
    try:
        df = pd.read_csv(plates_files)
        plate = np.array(df.iloc[:, 0])
        lat = np.array(df.iloc[:, 1])
        lon = np.array(df.iloc[:, 2])
        pb_dict = dict()

        for i in range(len(plate)):
            if plate[i] not in pb_dict:
                pb_dict[plate[i]] = np.array([[lon[i], lat[i]]])
            else:
                pb_dict[plate[i]] = np.append(pb_dict[plate[i]], np.array([[lon[i],
                ... lat[i]]]), axis=0)
        return pb_dict
    except:
        raise IOError
```

```
#Function 3
def get_earthquakes(filename):
    '''
    Reads the content from csv file into a pandas dataframe and returns the
    ... dataframe.
    input: filename
    output: earthquakes
    '''
    try:
        earthquakes = pd.read_csv(filename)
        return earthquakes
    except:
        raise IOError

#Function 4
def parse_earthquakes_to_np(df):
    '''
    Extracts columns latitude, longitude, depth, magnitude, and time from the input
    ... dataframe.
    Converts the time column to datetime objects.
    Returns the columns as individual 1D arrays
    input: df
    outputs: lats, lons, depths, magnitudes, times
    '''
    lats = np.array(df["Latitude"])
    lons = np.array(df["Longitude"])
    depths = np.array(df["Depth"])
    magnitudes = np.array(df["Magnitude"])
    times_object = np.array(df["Time"])
    times = pd.to_datetime(times_object)
    return lats, lons, depths, magnitudes, times

#helper function for c2 graph

def break_line_at_boundary(pb_dict, threshold=180):
    broken_lines = []
    '''
    To solve the issue of plate boundaries "crossing the map" when it needs to
    ... transpose between -180 to 180 and vice versa.
    Creates a list of points at the boundary 180 and ensures that the lines are
    ... temporarily split just only at that point and
    continuous everywhere else
    returns list as broken_lines
    '''
    for bound_lons, bound_lats in pb_dict.items():
        boundary_indices = np.where(np.abs(np.diff(bound_lats[:, 0])) > threshold)[0]
        ... + 1
        line_segments = np.split(bound_lats, boundary_indices)

        broken_lines.extend(line_segments)

    return broken_lines



-------- END STUDENT MODULE --------
```

Looking good

Bohao Hu #33131277 Leanne Su #55285753 Grady Chen #18865568

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import earthquake_fns as eq

# C1

longitudes, latitudes = eq.get_coastlines("./m_coasts.csv")
pb_dict = eq.get_plate_boundaries("./all_boundaries.csv")
earthquakes = eq.get_earthquakes("./IRIS_eq_010100_112422_mag4.csv")
lats, lons, depths, magnitudes, times = eq.parse_earthquakes_to_np(earthquak

largest_magnitude_index = np.argmax(magnitudes)
largest_magnitude = {
    'magnitude': magnitudes[largest_magnitude_index],
    'date/time': times[largest_magnitude_index],
    'latitude': lats[largest_magnitude_index],
    'longitude': lons[largest_magnitude_index] ,
    'depth': depths[largest_magnitude_index],
}

print(f"Largest Magnitude: {largest_magnitude['magnitude']}\n",
      f"Date/Time: {largest_magnitude['date/time']}\n",
      f"Latitude:{largest_magnitude['latitude']}\n",
      f"Longitude: {largest_magnitude['longitude']}\n",
      f"Depths: {largest_magnitude['depth']}")

# C2

last_2500_lats = lats[-2500:]
last_2500_lons = lons[-2500:]
last_2500_depths = depths[-2500:]
last_2500_magnitudes = magnitudes[-2500:]
last_2500_times = times[-2500:]

largest_magnitude_index_2 = np.argmax(last_2500_magnitudes)
deepest_depth_index = np.argmax(last_2500_depths)

largest_magnitude_lat = last_2500_lats[largest_magnitude_index_2]
largest_magnitude_lon = last_2500_lons[largest_magnitude_index_2]

deepest_depth_lat = last_2500_lats[deepest_depth_index]
deepest_depth_lon = last_2500_lons[deepest_depth_index]

start_date = last_2500_times.min().strftime('%Y-%m-%d %H:%M:%S')
end_date = last_2500_times.max().strftime('%Y-%m-%d %H:%M:%S')

fig, ax = plt.subplots(figsize=(10, 5))


ax.plot(longitudes, latitudes, color='black', linewidth=0.5)
```

```
broken_lines = eq.break_line_at_boundary(pb_dict) #for plate boundaries

for line_segment in broken_lines:
    ax.plot(line_segment[:, 0], line_segment[:, 1], linewidth=1)

ax.scatter(last_2500_lons, last_2500_lats, c='gray', s=20, edgecolors='none'

ax.scatter(largest_magnitude_lon, largest_magnitude_lat, c='red', marker='*'
ax.scatter(deepest_depth_lon, deepest_depth_lat, c='blue', marker='^', s=100
plt.title(f"Seismicity from {start_date} to {end_date}")
plt.legend(fontsize='small', loc='upper center', bbox_to_anchor=(0.5, -0.15)

ax.set_xlim(-180, 180)
ax.set_ylim(-90, 90)

# C3

earthquakes['Year'] = [date.year for date in times]
df_2022 = earthquakes[(earthquakes['Year'] == 2022)]
lats_2022, lons_2022, depths_2022, mags_2022, times_2022 = eq.parse_earthqua

deepest_quake = np.max(depths_2022)
deepest_quake_indice = np.argmax(depths_2022)
deepest_quake_time = times_2022[deepest_quake_indice]

largest_quake = np.max(mags_2022)
largest_quake_indice = np.argmax(mags_2022)
largest_quake_time = times_2022[largest_quake_indice]

fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(8, 6))

ax[0].hist(depths_2022, bins= 25, color='green', alpha = 0.7)
ax[0].axvline(x = deepest_quake, color='red', linestyle='--')

ax[0].annotate(f'Deepest Quake: {deepest_quake}m at {deepest_quake_time}', x
                arrowprops=dict(facecolor='black', arrowstyle='->', connect

ax[0].set_title('Earthquake Depths in 2022')
ax[0].set_xlabel('Depth (m)')
ax[0].set_ylabel('Occurrences')


ax[1].hist(mags_2022, bins = 25, color='orange', alpha = 0.7)

ax[1].axvline(x = largest_quake, color='blue', linestyle='--')

ax[1].annotate(f'Highest Magnitude Quake: {largest_quake} at {largest_quake_
                arrowprops=dict(facecolor='black', arrowstyle='->', connect

ax[1].set_title('Magnitudes in 2022')
ax[1].set_xlabel('Magnitude')
ax[1].set_ylabel('Occurrences')

plt.tight_layout()
```

```
Largest Magnitude: 9.1
 Date/Time: 2011-03-11 05:46:23
 Latitude:38.2963
 Longitude: 142.498
 Depths: 19.7
```