Brendan Sherman

Prof. Alvarez

CSCI 2291

February 20, 2022

Homework 4 Master Writeup

1. First, you will consider the relationship between the percentage of a country's population that is fully vaccinated and that country's total number of (presumably COVID-19-related) deaths per one million people. Both of these quantities are functions of time. Here, you will limit your attention to the maximum value of each of the two quantities for a given country.

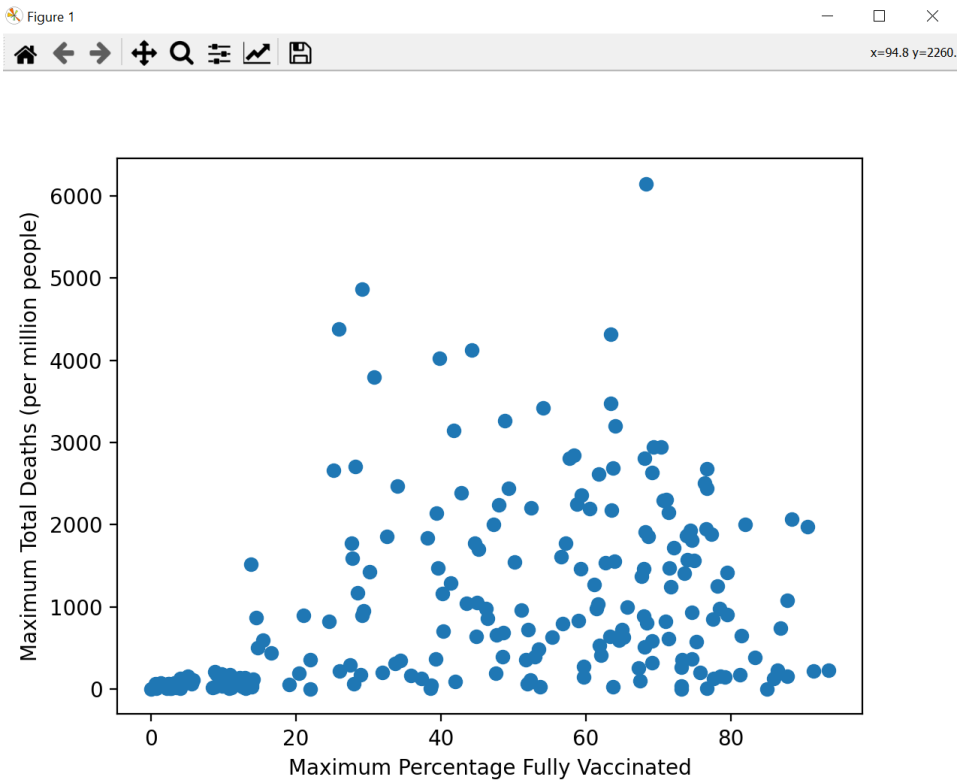
Explanation: The code above generates the below output, creating a scatter plot to illustrate and calculating the Pearson correlation coefficient for the relationship between the maximum percentage of people fully vaccinated and maximum total deaths (per million people) within the COVID-19 dataset. To accomplish this, I iterate through the one-dimensional array of unique ISO codes, accessing all rows of data for each country using Boolean indexing. I then use the Pandas *max()* function to get the maximum value for both relevant columns for each country, appending both into a 2-dimensional NumPy array. In this way, this array contains a row with both maximum values for each country (assuming that data exists for that country). I then use MatPlotLib to generate a scatter plot, using each column of the array as an axis, thereby plotting maximum percentage of people fully vaccinated vs. maximum total deaths (per million people). Finally, I use the Numpy *corrcoef* function to generate the correlation matrix, then accessing the correlation coefficient between the two variables which evaluates to .314. These results do not seem to suggest a strong correlation, with no obvious patterns occurring in the scatterplot. Similarly, the correlation coefficient suggests a positive correlation between vaccination rate and COVID mortality, which is opposite to what would be expected. An explanation for this could be that at the beginning of the pandemic, when vaccination rates were nonexistent, COVID-related deaths went unreported due to lack of testing.

Code:

```
def q1(df):
        iso_codes = np.unique(df['iso_code']) #1D array of unique countries


        #cleans rows with 'nan' values under specified columns
        df = df.dropna(subset=['total_deaths_per_million',
'people_fully_vaccinated_per_hundred'])
```

```
        arr = np.empty((0, 2))
        for j in range(iso_codes.size):
                country_rows = df[df['iso_code'] == iso_codes[j]] #array of data  for given
country
                max_deaths = country_rows['total_deaths_per_million'].max()
                max_vaccination_rate = country_rows['people_fully_vaccinated_per_hundred'].max()
                #append paired maximum values, given that data is present
                #also accounts for erroneous percentages (>100)
                if((not np.isnan(max_vaccination_rate)) and (not np.isnan(max_deaths))
                and max_vaccination_rate <= 100):
                        arr = np.append(arr, np.array([[max_vaccination_rate, max_deaths]]),
axis=0)


        scatter(arr[:, 0], arr[:, 1])
        xlabel("Maximum Percentage Fully Vaccinated")
        ylabel("Maximum Total Deaths (per million people)")
        show()


        r = np.corrcoef(arr[:, 0], arr[:, 1])
        print("Pearson Correlation Coefficient: " + str(round(r[0,1], 3)))


def main():
        df = pd.read_csv('owid-covid-data.csv')
        q1(df)
```

Output:




```
(base) C:\Users\Brend\Documents\Spring 2022\Data Science\homeworks\hwcode\hw4>python hw4code.py
[[1.         0.31356545]
 [0.31356545 1.        ]]
Pearson Correlation Coefficient: 0.314
```

2. The preceding task considers information that has been aggregated over all of the dates that occur in the data set, covering a period of approximately 2 years (the range varies by country). In order to provide finer time-granularity, consider the correlation between the percentage of the population that has been fully vaccinated (as of a given date) with the number of new deaths per million (as of the given date). The two quantities considered here are time-series, as both vaccination prevalence and new deaths per million vary with time. For each country you will be computing the correlation of those two time-series. Because the new deaths per million time-series is "noisy", use the smoothed version of new deaths per million instead.

Explanation:

The code below generates the outputs attached below, creating a boxplot representing an array of the correlations between vaccination rate and new covid deaths (per million people) by date for each country. In this way, we get finer time-granularity in considering the relationship between these two attributes for each country. To do this, I use the Pandas *DateTimeIndex()* function to create time indexes for each country's data, iterating through each iso code as in part 1. I then use this index to create a time series for both attributes, where the data is indexed according to the date where it was collected. Finally, I use the *Series.corr()* function to calculate the correlation between the two time series for each country, storing each correlation in an array. Finally, I use this array to create the boxplot and calculate the median shown below. Unlike part 1, here a negative correlation coefficient (**-.212**) suggests a negative correlation

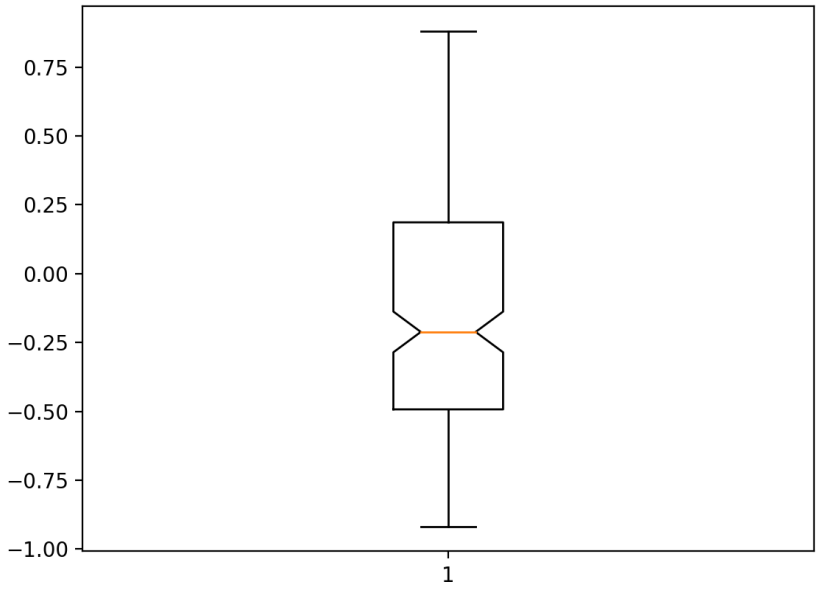
between vaccination rate and new deaths, meaning that a higher vaccination rate is correlated with less new COVID deaths, which would be expected. As evidenced by the notched boxplot, this correlation is significant (the median value differs significantly from 0), because the "range" of the notch is entirely below 0. Therefore, the possible range of median values is entirely negative meaning that the median correlation value **does** significantly different from 0.

Code:

```
#Problem 2- Continuing problem 1 analysis, now considering time granularity

def q2(df):
        iso_codes = np.unique(df['iso_code']) #1D array of unique countries
        corrs = np.empty((0, 2))
        for i in range(iso_codes.size):
                country_rows = df[df['iso_code'] == iso_codes[i]] #all data for given country
                #time-indexing data
                index = pd.DatetimeIndex(country_rows['date'])
                timed_vaccination_rate =
pd.Series(country_rows['people_fully_vaccinated_per_hundred'].values, index=index)
                timed_new_deaths =
pd.Series(country_rows['new_deaths_smoothed_per_million'].values, index=index)
                #computing correlation between time series
                corr = timed_vaccination_rate.corr(timed_new_deaths)
                corrs = np.append(corrs, [corr])
        #use boolean indexing to clear 'nan' values (missing data)
        corrs = corrs[~np.isnan(corrs)]
        print("Median Correlation Value: " + str(round(np.median(corrs), 3)))
        boxplot(corrs, notch=True)
        show()
def main():
        df = pd.read_csv('owid-covid-data.csv')
        q2(df)
```

Output:

```
Median Correlation Value: -0.212
```



3.

A) Write a function in Python that takes as input an alphanumeric string, s, and that returns the corresponding integer value according to this base 256 representation. Use Python's ord function to get each character's ASCII code.

Code:

```
def q3a(s):
        s_reverse = s[::-1]
        val = 0
        i = 0
        for c in s_reverse:
                val += (ord(c) * (256**i))
                i += 1
        return val
```

Explanation:

The function above accomplishes the task outlined in the problem, returning an integer representation (in base 256) of a given string using the ASCII code of the string's characters. First, I reverse the string using slicing. I then iterate through the reversed string, multiplying each character's ascii code by subsequent powers of 256 (starting at 256^0) and adding this product to a total. I then return this total, which represent the string passed.

B) Complete the computation of the two lists of countries (low mortality, high mortality) as described at the beginning of the statement.

Explanation: The code below generates the lists below, showing the countries within the highest and lowest quartiles for maximum COVID deaths (per million people). To do this, I first construct a dictionary as described in the problem, which relates the integer representation of each ISO code with the maximum deaths for that country. I then use numpy's *nanquartile()* function to calculate the 25th and 75th percentile for maximum deaths, using all values in the dictionary to do so. Finally, I iterate through each country's dictionary entry, comparing their maximum deaths value with the percentiles to generate lists of countries within the lowest and highest quartiles for this attribute. I use the fact that Python dictionaries are ordered to extract each ISO code (within the one-dimensional array) from its integer representation, saving processing power of converting each integer back into a string. Finally, I sort the quartile lists alphabetically and print/return them, generating the output attached below.

Code:

```
def q3b(df):
	iso_codes = np.unique(df['iso_code']) #1D array of unique countries

	country_max_deaths = {} #dictionary relating iso_code to max deaths for each country

	for i in range(iso_codes.size):
		country_rows = df[df['iso_code'] == iso_codes[i]] #extract all rows for given country
		max_deaths = country_rows['total_deaths_per_million'].max()
		key = q3a(iso_codes[i])

		country_max_deaths[key] = max_deaths

	#computing 25th and 75th percentiles for maximum deaths
	q1 = np.nanquantile(list(country_max_deaths.values()), .25)
	q3 = np.nanquantile(list(country_max_deaths.values()), .75)

	top_quarter = []
	low_quarter = []

	i = 0
	for key, val in country_max_deaths.items():
		if(val >= q3 and not(np.isnan(val)) and len(iso_codes[i]) == 3):
			top_quarter.append(iso_codes[i])
		elif(val < q1 and not(np.isnan(val)) and len(iso_codes[i]) == 3):
			low_quarter.append(iso_codes[i])
		i += 1
```

```
        top_quarter = sorted(top_quarter)
        low_quarter = sorted(low_quarter)

        print("Countries within top quarter of maximum covid deaths (per million people):")
        print(top_quarter)
        print("Countries within bottom quarter of maximum covid deaths (per million people):")
        print(low_quarter)
        return [low_quarter, top_quarter]
def main():
        df = pd.read_csv('owid-covid-data.csv')
        q3b(df)
```

Output: *(First list is high mortality, second is low mortality)*

```
(base) C:\Users\Brend\Documents\Spring 2022\Data Science\homeworks\hwcode\hw4>python hw4code.py
Countries within top quarter of maximum covid deaths (per million people):
['ABW', 'AND', 'ARG', 'ARM', 'BEL', 'BGR', 'BHS', 'BIH', 'BMU', 'BOL', 'BRA', 'CHL', 'COL', 'CZE', 'ECU', 'ESP', 'FRA', 'GBR', 'GEO', 'GIB',
 'GRC', 'GRD', 'HRV', 'HUN', 'ITA', 'LCA', 'LIE', 'LTU', 'LVA', 'MDA', 'MEX', 'MKD', 'MNE', 'PER', 'POL', 'PRT', 'PRY', 'PYF', 'ROU', 'RUS',
 'SMR', 'SRB', 'SUR', 'SVK', 'SVN', 'TTO', 'TUN', 'UKR', 'URY', 'USA']
Countries within bottom quarter of maximum covid deaths (per million people):
['AGO', 'BDI', 'BEN', 'BFA', 'BTN', 'CAF', 'CHN', 'CIV', 'CMR', 'COD', 'COG', 'ERI', 'ETH', 'GAB', 'GHA', 'GIN', 'GNB', 'GNQ', 'GRL', 'HKG',
 'HTI', 'ISL', 'KEN', 'KOR', 'LAO', 'LBR', 'MDG', 'MLI', 'MOZ', 'MWI', 'NER', 'NGA', 'NIC', 'NZL', 'PAK', 'PNG', 'RWA', 'SDN', 'SEN', 'SLB',
 'SLE', 'SOM', 'SSD', 'TCD', 'TGO', 'TJK', 'TLS', 'TWN', 'TZA', 'UGA', 'UZB', 'VUT', 'YEM']
```

4.

A) *Define a Python function that accepts a string and that returns the result of stripping blank spaces and quotation marks from that string; Python's string.strip function will do the work for you. Next, use DataFrame.apply to apply your function to both the 3-letter country code and average latitude columns of the geographic DataFrame, and replace the respective columns in the geographic DataFrame with the resulting stripped columns.*

Explanation:

The code below accomplishes the task outlined above. First, the function *strip_quotes_spaces* uses Python's String.*strip()* function to remove all trailing and leading quotation and white space characters. I use the escape character '\"' to remove quotations and " " to remove whitespaces. The function *Q4a* then uses this function along with the Pandas Series.*apply()* function to strip these characters from all values within the relevant columns (Alpha-3 code and Latitude (average)). Finally, I replace both columns within the original dataframe with the new, "stripped" columns.

Code:

```
#given a string s, remove quotation and whitespace characters
def strip_quotes_spaces(s):
```

```
        return s.strip("\" ")


#Q4a: Replace iso_code and latitude column data with stripped strings
def q4a(geo_df):
        stripped_iso_col = geo_df['Alpha-3 code'].apply(strip_quotes_spaces)
        stripped_lat_col = geo_df['Latitude (average)'].apply(strip_quotes_spaces)


        geo_df['Alpha-3 code'] = stripped_iso_col
        geo_df['Latitude (average)'] = stripped_lat_col
```

B)

Explanation:

The code below produces the output attached, listing the average latitude of each country within the high mortality and low mortality quartile lists generated in problem 3. The first function, as described in the problem, returns a country's average latitude given its iso code. First I use Boolean indexing and the Pandas s*eries.str.fullmatch()* function to extract the row (instance) within the geographic dataset for the given iso code. I then use *Series.iat[0]* to obtain the average latitude for that row, using the fact that there is only one row per country to do so. The second function then matches each country within the high and low mortality arrays from Q3 with its average latitude, listing these values for both quartile arrays. I call q3b() to access the quartile arrays, iterating through both and using the *get_avg_lat()* function to append each country's average latitude to a new set of arrays, which are printed in the output shown below. In this way, I create arrays of average latitudes for each country within the highest and lowest quartile for COVID mortality.

Code:

```
#Q4b: Given a country (iso code), return its average latitude
def get_avg_lat(geo_df, iso):
        row = geo_df[geo_df['Alpha-3 code'].astype('str').str.fullmatch(iso)]
        avg_lat = float(row['Latitude (average)'].iat[0])


        if(not np.isnan(avg_lat)):
                return avg_lat
        else:
                return np.nan


#Q4b: create list of avg latitudes for low and high mortality countries (from q3)
def q4b(df, geo_df):
        q4a(geo_df)
        iso_lists = q3b(df)
```

```
        low_mortality_quarter = iso_lists[0]
        high_mortality_quarter = iso_lists[1]

        high_mortality_avg_lats = []
        low_mortality_avg_lats = []

        for i in high_mortality_quarter:
                high_mortality_avg_lats.append(get_avg_lat(geo_df, i))

        for i in low_mortality_quarter:
                low_mortality_avg_lats.append(get_avg_lat(geo_df, i))

        print("Average latitude of lower quartile COVID mortality countries:")
        print(low_mortality_avg_lats)
        print("Average latitude of higher quartile COVID mortality countries:")
        print(high_mortality_avg_lats)

        return [low_mortality_avg_lats, high_mortality_avg_lats]
def main():
        df = pd.read_csv('owid-covid-data.csv')
        geo_df = pd.read_csv("countries_codes_and_coordinates.csv")
        q4b(df, geo_df)
```

Output:

```
Average latitude of lower quartile COVID mortality countries:
[-12.5, -3.5, 9.5, 13.0, 27.5, 7.0, 35.0, 8.0, 6.0, 0.0, -1.0, 15.0, 8.0, -1.0, 8.0, 11.0, 12.0, 2.0, 72.0, 22.25, 19.0, 65.0, 1.0, 37.0, 18.0, 6.5, -20.0, 17.0
, -18.25, -13.5, 16.0, 10.0, 13.0, -41.0, 30.0, -6.0, -2.0, 15.0, 14.0, -8.0, 8.5, 10.0, 8.0, 15.0, 8.0, 39.0, -8.55, 23.5, -6.0, 1.0, 41.0, -16.0, 15.0]
Average latitude of higher quartile COVID mortality countries:
[12.5, 42.5, -34.0, 40.0, 50.8333, 43.0, 24.25, 44.0, 32.3333, -17.0, -10.0, -30.0, 4.0, 49.75, -2.0, 40.0, 46.0, 54.0, 42.0, 36.1833, 39.0, 12.1167, 45.1667, 4
7.0, 42.8333, 13.8833, 47.1667, 56.0, 57.0, 47.0, 23.0, 41.8333, 42.0, -10.0, 52.0, 39.5, -23.0, -15.0, 46.0, 60.0, 43.7667, 44.0, 4.0, 48.6667, 46.0, 11.0, 34.
0, 49.0, -33.0, 38.0]
```

C) Compute the medians of the two latitude lists from the preceding part. Plot notched box plots of these lists side by side in matplotlib. Include your source code, numerical median values, and box plots in your writeup. Is there a significant difference in latitude between the two groups of countries?

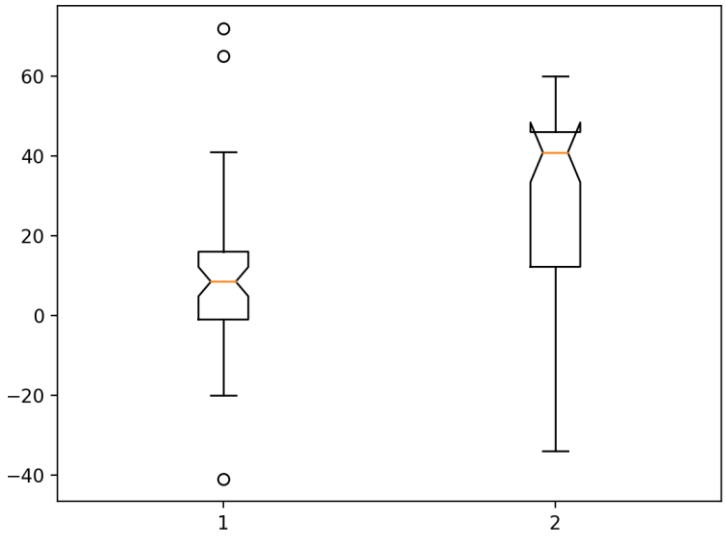
Explanation:

The code below generates the outputs attached, showing the numerical mean values for each latitude array generated in part B and creating notch boxed plots representing the data in both arrays. I first access the arrays by calling *q4b()*, then using *np.median()* to print the median values of both arrays. Finally, I use Matplotlib's *boxplot()* function to generate notched boxplots for both arrays. From these notched boxplots, it becomes clear that there *is* a significant difference in the average latitude of high COVID-mortality and low COVID-mortality countries. This is because the notches for the two boxplots, representing the range of *possible* median average latitudes, do not overlap whatsoever. This means that whatever the uncertainty, the median average latitude of high and low mortality countries will always differ, and therefore the difference is significant. In terms of magnitude, the two median average latitudes also differ, with the median latitude for low-mortality countries being 8.5 and for high-mortality countries being 40.917.

Code:

```
#Q4c: Compute medians of both average latitude lists, evaluate significance

def q4c(df, geo_df):

        avg_lats = q4b(df, geo_df)


        print("Median latitutde for countries in lowest quartile: " +
str(round(np.median(avg_lats[0]), 3)))

        print("Median latitutde for countries in highest quartile: " +
str(round(np.median(avg_lats[1]), 3)))


        boxplot(avg_lats, notch=True)

        show()

def main():

        df = pd.read_csv('owid-covid-data.csv')

        geo_df = pd.read_csv("countries_codes_and_coordinates.csv")

        q4c(df, geo_df)
```

Output:




```
Median latitutde for countries in lowest quartile: 8.5
Median latitutde for countries in highest quartile: 40.917
```