2. Finding Geographic Locations of Headlines

October 10, 2019

1 Finding Geographic Locations of Headlines

1.1 Adding Latitude and Longitude Coordinates

Objective: Find the geographic location of each headline in latitude and longitude coordinates from the city/country names. We will use these coordinates to perform clustering of geographically similar headlines in the next part.

Workflow: 1. Load in the Pandas DataFrame with headline, countries, and cities. - If a headline contains multiple cities/countries, decide which single one to keep. 2. For each city/country, match the name to the latitude and longitude in geonamescache. - You can use the function gc.get_cities_by_names("city_name"). - Some cities will return multiple matches with the previous function in different countries. You'll have to decide which city to keep based on a heuristic (rule of thumb). - If you have trouble, work with a single problematic city until you figure it out, then write a function to apply on all headlines. 3. Add longitude and latitude coordinates to your DataFrame for each headline. - It will be helpful to get the countrycode of each headline at this point. - If you were not able to find many countries, think about dropping the column. You also need to decide what to do with headlines that have no coordinates. - You should end up with over 600 headlines that have geographic coordinates

Deliverable:

The deliverable is a Jupyter Notebook documenting your work as you add three additional columns to the DataFrame: longitude, latitude, and countrycode. We will use these coordinates to cluster the headlines in the next section.

1.2 Read Data into a DataFrame

We stored the headline, cities, and countries as a json file that was a list of dictionaries. This can be directly read in a Pandas dataframe.

[1]:
```
import pandas as pd
import numpy as np

data = pd.read_json("../data/headline_cities_and_countries.json")
data = data.replace({None: np.nan})
data.head()
```

[1]:
```
                                 headline countries         cities
0                  Zika Outbreak Hits Miami       NaN          Miami
```

```
1          Could Zika Reach New York City?       NaN  New York City
2         First Case of Zika in Miami Beach       NaN    Miami Beach
3  Mystery Virus Spreads in Recife, Brazil    Brazil         Recife
4  Dallas man comes down with case of Zika       NaN         Dallas
```

[2]:
```
data.iloc[3:5]
```

[2]:
```
                                  headline countries  cities
3  Mystery Virus Spreads in Recife, Brazil    Brazil  Recife
4  Dallas man comes down with case of Zika       NaN  Dallas
```

We'll rename the columns to singular (since they only have one value each).

[3]:
```
data = data.rename(columns=dict(countries="country", cities="city"))
data.tail()
```

[3]:
```
                                              headline country           city
645  Rumors about Rabies spreading in Jerusalem hav…     NaN      Jerusalem
646              More Zika patients reported in Indang     NaN         Indang
647  Suva authorities confirmed the spread of Rotav…     NaN           Suva
648         More Zika patients reported in Bella Vista     NaN    Bella Vista
649                     Zika Outbreak in Wichita Falls     NaN  Wichita Falls
```

From a brief look at some of the headlines and cities, our regular expression pattern matching looks to have worked well. As we go through the project, we'll keep an eye out for places it may have failed.

1.3 Investigate the Data

We can start off using the `.describe()` method to understand our data.

[4]:
```
data.describe()
```

[4]:
```
                              headline country    city
count                              650      15     608
unique                             647      10     573
top     Barcelona Struck by Spanish Flu  Brazil  Madrid
freq                                 2       3       4
```

It looks like there may be some duplicates in the data since at least one headline is mentioned twice. Let's check for duplicates and then drop any that are duplicated.

[5]:
```
data["headline"].value_counts().sort_values().tail()
```

[5]:
```
Zika Outbreak in Hyderabad            1
Ibadan tests new cure for Malaria     1
Spanish Flu Outbreak in Lisbon        2
Spanish Flu Spreading through Madrid  2
```

```
Barcelona Struck by Spanish Flu         2
Name: headline, dtype: int64
```

[6]:
```
print(f"There were {len(data)} rows before dropping duplicates.")
data = data.drop_duplicates()
print(f"There are {len(data)} rows after dropping duplicates.")
```

```
There were 650 rows before dropping duplicates.
There are 647 rows after dropping duplicates.
```

Another useful method for data investigation is `.info()`

[7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 647 entries, 0 to 649
Data columns (total 3 columns):
headline    647 non-null object
country     15 non-null object
city        605 non-null object
dtypes: object(3)
memory usage: 20.2+ KB
```

We can see there are many missing countries (635) and some missing cities (42). The data types look correct at this point.

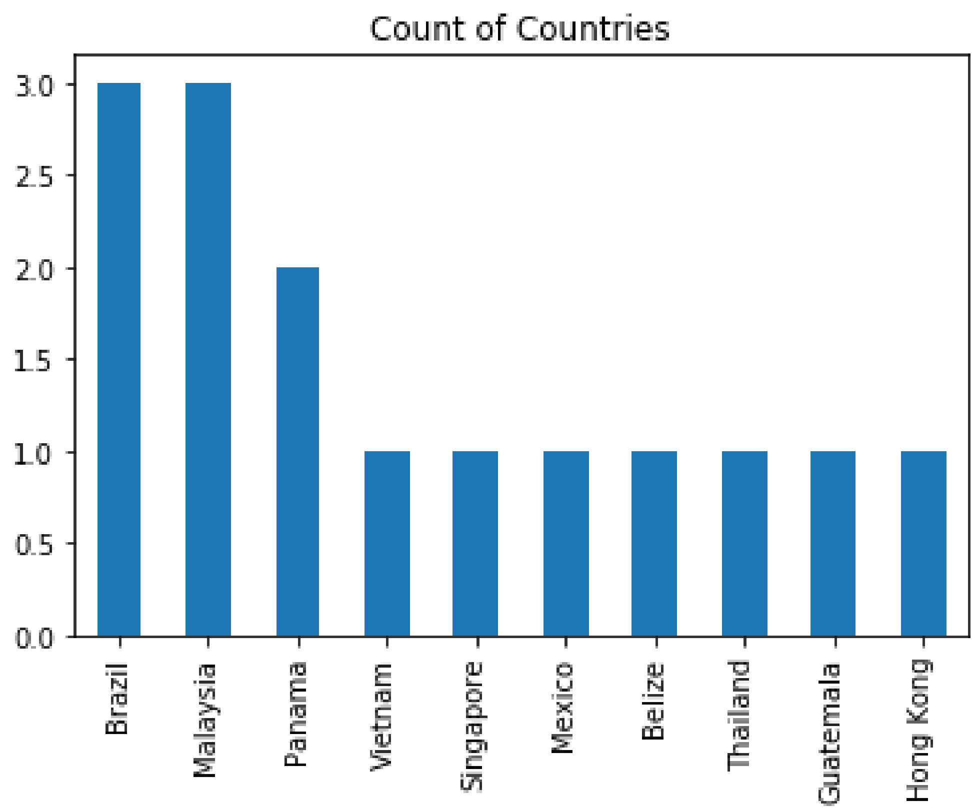
1.3.1 Exploratory Plots

Plots are a great way to visualize data. Let's take a look at the distribution of countries and cities.

[8]:
```
data['country'].value_counts()
```

[8]:
```
Brazil       3
Malaysia     3
Panama       2
Vietnam      1
Singapore    1
Mexico       1
Belize       1
Thailand     1
Guatemala    1
Hong Kong    1
Name: country, dtype: int64
```

[9]:
```
%matplotlib inline

_ = data['country'].value_counts().plot.bar(title='Count of Countries')
```



We have many more cities, so a bar plot might not be the best graphic.

[10]:
```
print(f'There are {data["country"].nunique()} different countries.')
print(f'There are {data["city"].nunique()} different cities.')
```

```
There are 10 different countries.
There are 573 different cities.
```

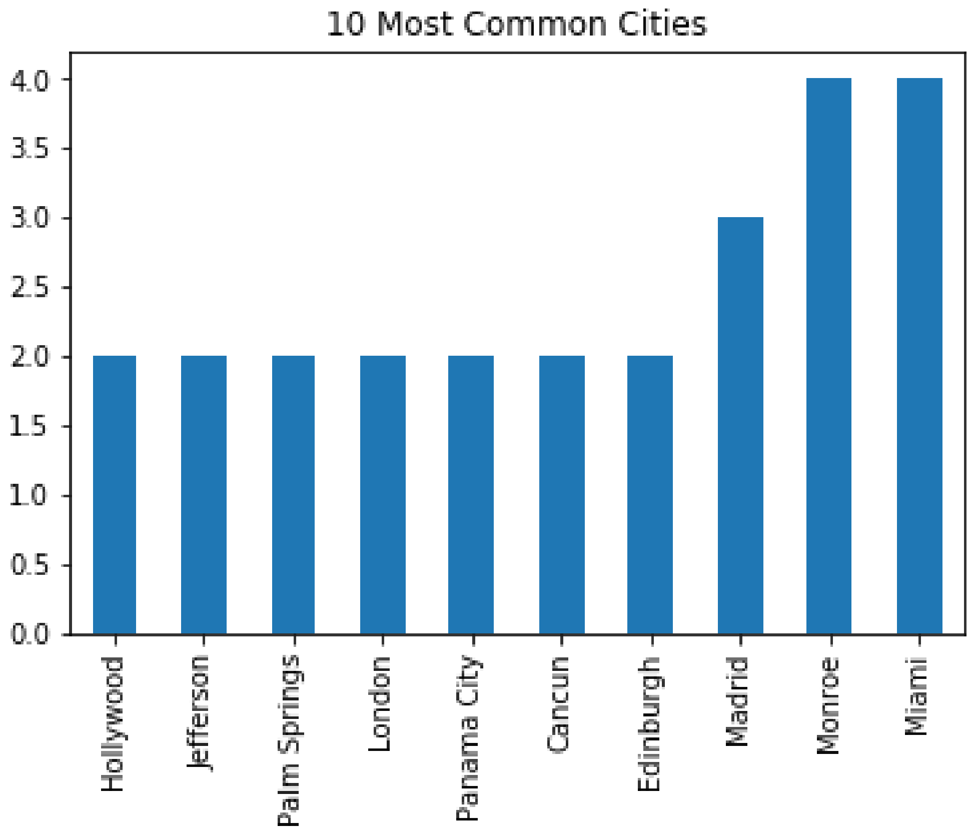
Let's just look at the 10 most common cities.

[11]:
```
data["city"].value_counts().sort_values().tail(10)
```

[11]:
```
Hollywood        2
Jefferson        2
Palm Springs     2
London           2
Panama City      2
Cancun           2
Edinburgh        2
Madrid           3
Monroe           4
Miami            4
```

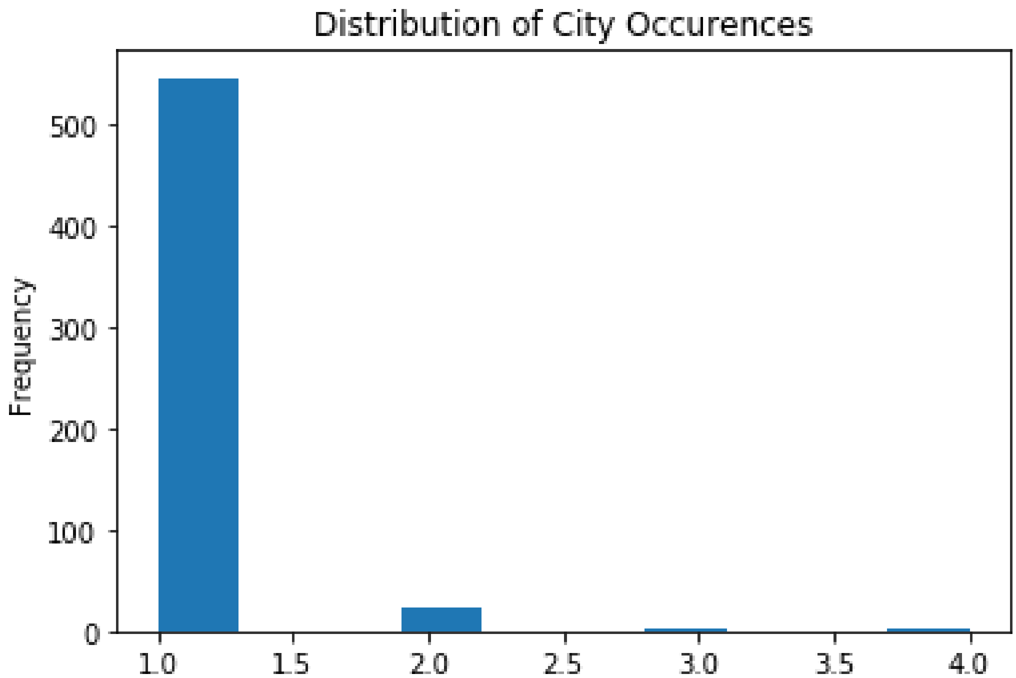
```
Name: city, dtype: int64
```

[12]:
```
_ = (
    data["city"]
    .value_counts()
    .sort_values()
    .tail(10)
    .plot.bar(title="10 Most Common Cities")
)
```



We can see that there are no cities that dominate the headlines.

[13]:
```
_ = data["city"].value_counts().plot.hist(title="Distribution of City␣
↪Occurences")
```



2 Add Latitude and Longitude for Each City

We can now add the latitude and longitude for each city in the headlines. We will not add the country locations since there are a limited number of countries.

2.1 Accented Names

For finding the cities, we need to use accented city names. We'll create an accented name column using our mapping from the previous section.

[14]:
```
import geonamescache
import unidecode

import json

gc = geonamescache.GeonamesCache()

# Read in the saved unaccented:accented mapping
with open("../data/city_accent_mapping.json", "r") as fin:
    city_accented_mapping = json.loads(fin.read())

# Create a column for accented cities
data["accented_city"] = data["city"].map(city_accented_mapping)
data[data["city"] != data["accented_city"]].head()
```

[14]:
```
                              headline country       city accented_city
7   Geneve Scientists Battle to Find Cure      NaN     Geneve        Genève
9      Zika Infested Monkeys in Sao Paulo      NaN  Sao Paulo     São Paulo
17          Louisiana Zika cases up to 26      NaN        NaN           NaN
19    Zika infects pregnant woman in Cebu      NaN        NaN           NaN
47              18 new Zika Cases in Bogota      NaN     Bogota        Bogotá
```

We can see there are several cases where the accented city does not match the original city.

[15]:
```python
print(gc.get_cities_by_name('São Paulo'))
```

```
[{'3448439': {'geonameid': 3448439, 'name': 'São Paulo', 'latitude': -23.5475,
'longitude': -46.63611, 'countrycode': 'BR', 'population': 10021295, 'timezone':
'America/Sao_Paulo', 'admin1code': '27'}}]
```

[16]:
```python
print(gc.get_cities_by_name('Sao Paulo'))
```

```
[]
```

We see the importance of using the accented names!

2.2 Handling Duplicate Cities

This is where we'll handle the duplicate cities. Our approach is relatively basic:

For each city with multiple entries in geonames, we'll choose the city with the greatest population.

This is may occassionally be wrong, but a headline is more likely to mention a larger city (by population).

We can implement this by checking which is the largest entry for each city. Some cities have multiple locations as shown by `Boston`.

[17]:
```python
city = 'Boston'
gc.get_cities_by_name(city)
```

[17]:
```
[{'2655138': {'geonameid': 2655138,
   'name': 'Boston',
   'latitude': 52.97633,
   'longitude': -0.02664,
   'countrycode': 'GB',
   'population': 41340,
   'timezone': 'Europe/London',
   'admin1code': 'ENG'}},
 {'4930956': {'geonameid': 4930956,
   'name': 'Boston',
   'latitude': 42.35843,
   'longitude': -71.05977,
   'countrycode': 'US',
```

```
 'population': 667137,
 'timezone': 'America/New_York',
 'admin1code': 'MA'}}]
```

In this case we want `Boston` in the United States since it has the larger population. To get the largest city, we sort the matches by the `population` key.

[18]:
```
matches = gc.get_cities_by_name(city)
matches = [{k: v for k, v in list(match.values())[0].items()} for match in␣
 ↪matches]
matches = sorted(matches, key=lambda x: x["population"], reverse=True)
matches
```

[18]:
```
[{'geonameid': 4930956,
  'name': 'Boston',
  'latitude': 42.35843,
  'longitude': -71.05977,
  'countrycode': 'US',
  'population': 667137,
  'timezone': 'America/New_York',
  'admin1code': 'MA'},
 {'geonameid': 2655138,
  'name': 'Boston',
  'latitude': 52.97633,
  'longitude': -0.02664,
  'countrycode': 'GB',
  'population': 41340,
  'timezone': 'Europe/London',
  'admin1code': 'ENG'}]
```

This sorts by the population of the cities descending (largest to smallest). If we take the first city, then we'll have the largest.

2.3 Finding Locations for Cities

Now let's find the locations of all the cities in the headlines. We'll want to be careful to go through the accented city names. If there are multiple matches for a city, we'll take the largest city.

[19]:
```
city_locations = []

# Go through all the accented cities
for city in data["accented_city"]:
    # Find matches (if any)
    matches = gc.get_cities_by_name(city)
    if matches:
        # Sort from largest to smallest population
        matches = [
```

```
            {k: v for k, v in list(match.values())[0].items()} for match in␣
↪matches
        ]
        matches = sorted(matches, key=lambda x: x["population"], reverse=True)

        # Find the match with the largest population
        match = matches[0]

        # Record the information
        city_locations.append(
            {
                "name": match["name"],
                "latitude": match["latitude"],
                "longitude": match["longitude"],
                "countrycode": match["countrycode"],
                "pop": match["population"],
            }
        )

city_locations[-5:]
```

[19]:
```
[{'name': 'Jerusalem',
  'latitude': 31.76904,
  'longitude': 35.21633,
  'countrycode': 'IL',
  'pop': 801000},
 {'name': 'Indang',
  'latitude': 14.19528,
  'longitude': 120.87694,
  'countrycode': 'PH',
  'pop': 41159},
 {'name': 'Suva',
  'latitude': -18.14161,
  'longitude': 178.44149,
  'countrycode': 'FJ',
  'pop': 77366},
 {'name': 'Bella Vista',
  'latitude': 18.45539,
  'longitude': -69.9454,
  'countrycode': 'DO',
  'pop': 175683},
 {'name': 'Wichita Falls',
  'latitude': 33.91371,
  'longitude': -98.49339,
  'countrycode': 'US',
  'pop': 104710}]
```

We can convert this list of dictionaries to a dataframe.

[20]:
```
city_locations = pd.DataFrame(city_locations)
city_locations.tail()
```

[20]:
```
              name  latitude  longitude countrycode     pop
600      Jerusalem  31.76904   35.21633          IL  801000
601         Indang  14.19528  120.87694          PH   41159
602           Suva -18.14161  178.44149          FJ   77366
603    Bella Vista  18.45539  -69.94540          DO  175683
604  Wichita Falls  33.91371  -98.49339          US  104710
```

[21]:
```
city_locations = city_locations.drop_duplicates()
print(f"We have the locations for {city_locations.shape[0]} unique cities.")
```

```
We have the locations for 573 unique cities.
```

Next let's merge with the headlines on the `accented_city` and `name`.

[22]:
```
data = pd.merge(
    data, city_locations, left_on="accented_city", right_on="name", how="left"
)
data.head()
```

[22]:
```
                                  headline country           city  \
0                   Zika Outbreak Hits Miami     NaN          Miami
1            Could Zika Reach New York City?     NaN  New York City
2          First Case of Zika in Miami Beach     NaN    Miami Beach
3  Mystery Virus Spreads in Recife, Brazil  Brazil         Recife
4  Dallas man comes down with case of Zika     NaN         Dallas

   accented_city           name  latitude  longitude countrycode        pop
0          Miami          Miami  25.77427  -80.19366          US   441003.0
1  New York City  New York City  40.71427  -74.00597          US  8175133.0
2    Miami Beach    Miami Beach  25.79065  -80.13005          US    92312.0
3         Recife         Recife  -8.05389  -34.88111          BR  1478098.0
4         Dallas         Dallas  32.78306  -96.80667          US  1300092.0
```

Let's make sure keeping the largest city worked. We can try `Boston` as well as `Rochester`, both of which should be in the United States.

[23]:
```
data[data['city'] == 'Boston']
```

[23]:
```
                  headline country    city accented_city    name  latitude  \
27  Flu season hits Boston     NaN  Boston        Boston  Boston  42.35843

    longitude countrycode       pop
27  -71.05977          US  667137.0
```

[24]:
```
data[data['city'] == 'Rochester']
```

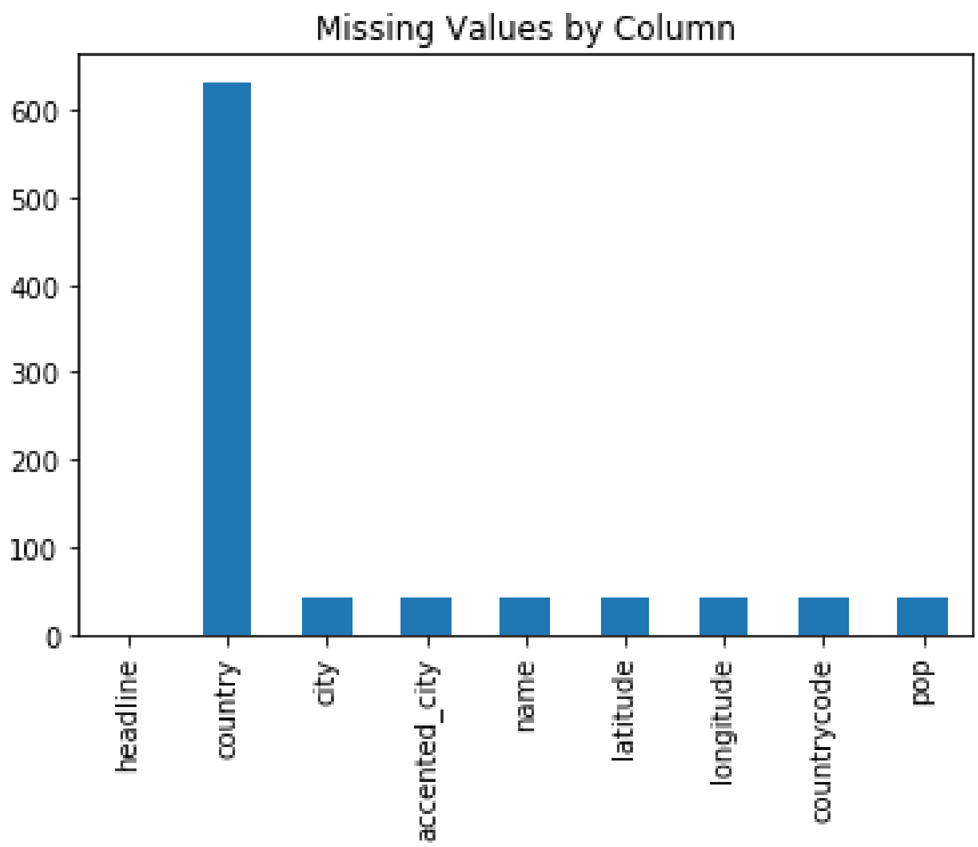
[24]:
```
                                              headline country       city  \
84   Rochester authorities confirmed the spread of …     NaN  Rochester
298                Herpes Keeps Spreading in Rochester     NaN  Rochester

    accented_city       name  latitude  longitude countrycode       pop
84      Rochester  Rochester  43.15478  -77.61556          US  209802.0
298     Rochester  Rochester  43.15478  -77.61556          US  209802.0
```

It looks like our method for finding the largest city worked as expected. For each headline with a city in `geonamescache`, we now have the latitude and longitude.

Due to the limited number of countries found in headlines, we'll stick to only the cities.

2.4 Data Cleaning

[25]:
```
_ = data.isna().sum().plot.bar(title='Missing Values by Column')
```



We can see there are quite a few missing values in the `country` column. Let's just remove the country since it does not give us much information.

[26]:
```
data = data.drop(columns=['country'])
```

Let's investigate the headlines where we don't have a `name`. We might be able to figure out more data cleaning steps to take.

[27]:
```
pd.options.display.max_colwidth = 100

no_name = data[data["name"].isna()].copy()

print(f"There are {len(no_name)} headlines without a city.")

no_name.tail()
```

```
There are 42 headlines without a city.
```

[27]:
```
                                                   headline city  \
596                                    Zika arrives in Dangriga  NaN
601  More Patients in Maynard are Getting Diagnosed with Syphilis  NaN
625                               Zika case reported in Antioquia  NaN
627                          Chikungunya has not Left Pismo Beach  NaN
628                                    Zika spreads to La Joya  NaN

    accented_city name  latitude  longitude countrycode  pop
596           NaN  NaN       NaN        NaN         NaN  NaN
601           NaN  NaN       NaN        NaN         NaN  NaN
625           NaN  NaN       NaN        NaN         NaN  NaN
627           NaN  NaN       NaN        NaN         NaN  NaN
628           NaN  NaN       NaN        NaN         NaN  NaN
```

We should manually check a few of these to make sure we can't find a city for the headline.

[28]:
```
city_set = set(city_accented_mapping.keys())

for city in ["Dangriga", "Maynard", "Antioquia", "Pismo Beach", "La Joya"]:
    if city in city_set:
        print("Found ", city)
    else:
        print("Did Not Find City")
```

```
Did Not Find City
Did Not Find City
Did Not Find City
Did Not Find City
Did Not Find City
```

It appears that the 42 headlines without a city name may have a city, but it is not included in `geonamescache`. We'll have to go ahead and remove these cities since they cannot be used.

[29]:
```
data = data.dropna(subset=['name'])
data.describe()
```

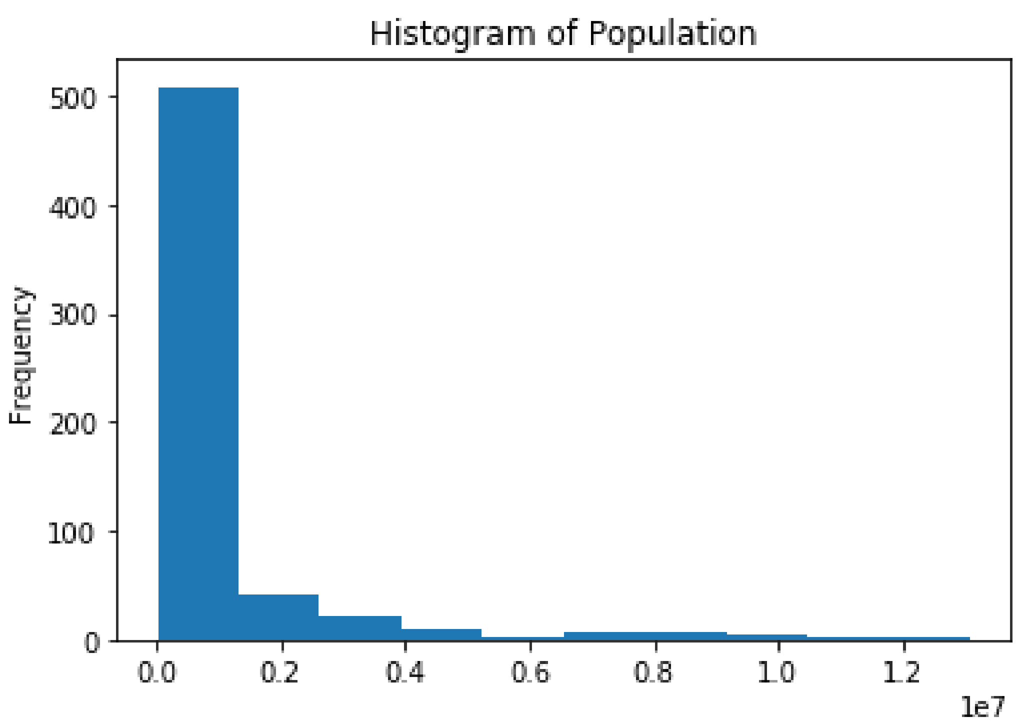
[29]:

	latitude	longitude	pop
count	605.000000	605.000000	6.050000e+02
mean	26.765746	-38.243197	8.904713e+05
std	20.619771	79.480854	1.974091e+06
min	-53.787690	-156.506040	1.338100e+04
25%	16.419040	-90.444300	5.878700e+04
50%	33.749000	-76.496610	1.712140e+05
75%	40.714270	7.095490	6.480340e+05
max	59.938630	179.364510	1.307630e+07

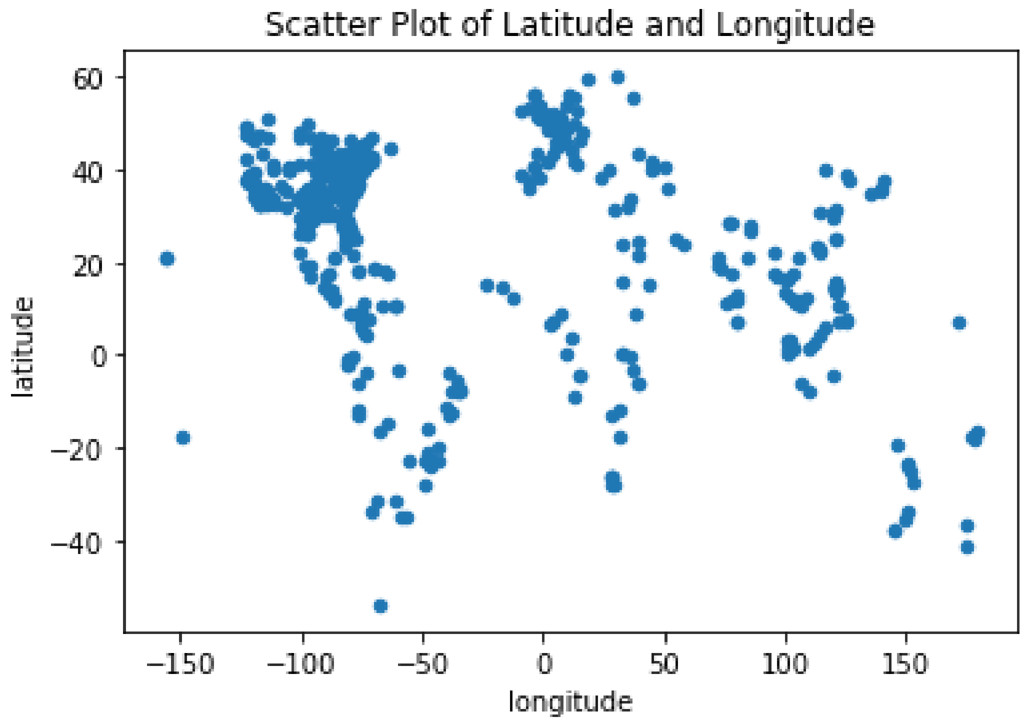
[30]:
```
print(f'We have the city locations for {len(data)} cities.')
```

```
We have the city locations for 605 cities.
```

[31]:
```
_ = data['pop'].plot.hist(title='Histogram of Population')
```



[32]:
```
_ = data.plot.scatter(
    x="longitude", y="latitude", title="Scatter Plot of Latitude and Longitude"
)
```



This looks sort of like a map! We'll have to use a map in the next section.

As a final cleaning step, we can remove the `name` column since it is redundant with city. The final data frame is below.

[33]:
```
data = data.drop(columns=['name'])
data.tail()
```

[33]:
```
                                                headline  \
642  Rumors about Rabies spreading in Jerusalem have been refuted
643                           More Zika patients reported in Indang
644                 Suva authorities confirmed the spread of Rotavirus
645                       More Zika patients reported in Bella Vista
646                                  Zika Outbreak in Wichita Falls

              city  accented_city  latitude  longitude countrycode       pop
642      Jerusalem      Jerusalem  31.76904   35.21633          IL  801000.0
643         Indang         Indang  14.19528  120.87694          PH   41159.0
644           Suva           Suva -18.14161  178.44149          FJ   77366.0
645    Bella Vista    Bella Vista  18.45539  -69.94540          DO  175683.0
646  Wichita Falls  Wichita Falls  33.91371  -98.49339          US  104710.0
```

[34]:
```
data[['headline', 'city', 'latitude', 'longitude', 'countrycode']].head(10)
```

[34]:
```
                                 headline           city  latitude  \
0                  Zika Outbreak Hits Miami          Miami  25.77427
1           Could Zika Reach New York City?  New York City  40.71427
2         First Case of Zika in Miami Beach    Miami Beach  25.79065
3  Mystery Virus Spreads in Recife, Brazil         Recife  -8.05389
4  Dallas man comes down with case of Zika         Dallas  32.78306
5         Trinidad confirms first Zika case       Trinidad -14.83333
6   Zika Concerns are Spreading in Houston        Houston  29.76328
7    Geneve Scientists Battle to Find Cure         Geneve  46.20222
8    The CDC in Atlanta is Growing Worried        Atlanta  33.74900
9       Zika Infested Monkeys in Sao Paulo      Sao Paulo -23.54750

   longitude countrycode
0  -80.19366          US
1  -74.00597          US
2  -80.13005          US
3  -34.88111          BR
4  -96.80667          US
5  -64.90000          BO
6  -95.36327          US
7    6.14569          CH
8  -84.38798          US
9  -46.63611          BR
```

This dataframe is the final outcome from this section. We will use it to cluster headlines based on the geographic location in the next section.

2.5 Saving Data

Let's save the final processed dataframe to a csv file for easy input and output with Pandas.

[35]:
```
data.to_csv('../data/processed_headlines_locations.csv')
```

3 Summary

In this notebook we:

- Read the parsed headlines into a dataframe
- Found the location of the cities mentioned in the headlines
- Kept the largest city if a city was in geonames multiple times
- Joined the cities to the headlines
- Cleaned up the final dataframe to only headlines with a location

The end deliverable is a dataframe containing the headline, the city mentioned in the headline, the location of the city, and the population of the city. We can move on to clustering and visualizing the headline locations in the next section!

[ ]:
```

```