In [1]:

```
import pandas as pd
import eeweather as ee
from folium import plugins
import folium
from folium.plugins import HeatMap
import plotly.express as px
import warnings
import matplotlib.pyplot as plt
%matplotlib inline
```

Read in all of the `csv` files as pandas DataFrames.

In [2]:

```
df1 = pd.read_csv('Dataset_1.csv')
df2 = pd.read_csv('Dataset_2.csv')
df3 = pd.read_csv('Dataset_3.csv')
```

Assignment #2

For assignment two, we are provided a dataset with 14 customer phone numbers, and we're asked to develop code to determine if the phone numbers are correct and then return one of the two phone numbers. We like to point out that while this file shows a method to solve this problem using Python, we also solved this problem using the PROC SQL method in SAS (provided in a separate attachment). All of the code for this and the other assignments can be found at the Github repository below. https://github.com/grantaguinaldo/bi-work/blob/main/bi-work-assignment-02.ipynb (https://github.com/grantaguinaldo/bi-work/blob/main/bi-work-assignment-02.ipynb)

We will also have to code conditions that prioritize Phone Number One over Phone Number Two when Phone Number One is available and in the correct format. On the other hand, we can settle for Phone Number Two if Phone Number One is not available.

Approach

In terms of a high-level approach, we need to compare both numbers, determine which one is correct, and then code in logic to return the valid number.

In [3]:
```
df2
```

Out[3]:

	CustomerID	PhoneNumber1	PhoneNumber2
0	1000	9113458738	1.114897e+10
1	1001	9013458736	1.104897e+09
2	1002	8913458734	NaN
3	1003	8813458732	1.084897e+10
4	1004	aaa	1.074897e+10
5	1005	8613458728	1.064897e+10
6	1006	851345872	1.054897e+10
7	1007	8413458724	4.000000e+00
8	1008	8313458722	1.034897e+10
9	1009	NaN	1.024897e+10
10	1010	8113458718	1.014897e+09
11	1011	8013458716	1.004897e+10
12	1012	NaN	9.994897e+10
13	1013	7813458712	9.824897e+10

In [4]:
```
print('The dataframe contains {} rows and {} columns.'.format(df2.shape[0], df2.shape[1]))
```

```
The dataframe contains 14 rows and 3 columns.
```

Inspection of Data for Data Types and Missing Values

I first looked at the data to figure out if there are any missing values. From the code below, you do see that there are missing values in the data. In particular, you see that the total rows of data are not 14, so that means that there are missing values that need to be addressed since the total number of rows is 14.

From the data below, you also see that the data types between the three fields are different. You have integers and characters/objects, and so we'll need to address those different types of data.

In [5]:

```
# From the code below, we observe that the file has missing values.
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   CustomerID    14 non-null     int64  
 1   PhoneNumber1  12 non-null     object 
 2   PhoneNumber2  13 non-null     float64
dtypes: float64(1), int64(1), object(1)
memory usage: 464.0+ bytes
```

Backfilling Missing Data

Since we know that we have missing values, the first thing was to impute a zero for all the missing values in the data frame.

In [6]:

```
# We will input a `0` for all missing values in the dataframe.
df2.fillna(0, inplace=True)
```

Converting Column Names to Lower Case

Next, I converted the column names to lowercase, which is a practice that I do out of habit. Following that, I checked that the column names have indeed been converted to lowercase.

```
In [7]: #Convert all of the column names to lower case
        df2.columns = map(str.lower, df2.columns)
        df2.columns.to_list()
```

```
Out[7]: ['customerid', 'phonenumber1', 'phonenumber2']
```

Determine Length of Phone Number (to determine which ones are 'correct').

The code below determines each phone number's length, and I am returning the length for each number as its variable. This calculated variable is what will be used to determine if the phone numbers are correct. For this analysis, a number is correct if it has *exactly* ten digits.

```
In [8]: #Determine length of each phone number.
        df2['phone1len'] = df2['phonenumber1'].astype(str).apply(len)
        df2['phonenumber2'] = df2['phonenumber2'].astype(int)
        df2['phone2len'] = df2['phonenumber2'].astype(str).apply(len)
```

Selecting Correct Phone Numbers

The code below is doing the comparison between both numbers, picking out the correct number. In this case, we've assumed that the right phone number length is ten digits. Moreover, if it's 11 or more digits, we considered the number incorrect because you don't know if the first digit or the last digit is the extra digit. Also, without knowing the country where these phone numbers originated or the area code, I don't think there is anything more that can be assumed.

In [9]:

```
#Review each phone number and determine if the number is correct based on the
#number of digits. If the number of digits is 10, we assume that the number is correct.
#In the even that none of the numbers are correct we return "invalid_phone".
df2['customerphone'] = ''
index_row = []

for index, row in df2.iterrows():
    try:
        if row['phone1len'] == 10:
            df2.at[index, 'customerphone'] = row['phonenumber1']
        elif (row['phone1len'] < 10) & (row['phone1len']) == 10:
            df2.at[index, 'customerphone'] = row['phonenumber2']
        else:
            df2.at[index, 'customerphone'] = 'invalid_phone'
    except:
        index_row.append(index)
index_row
```

Out[9]:

```
[]
```

Final Table

We see that out of the 14 phone numbers, four of them have invalid phone numbers (noted as `invalid_phone`) from this work. The four that have invalid phone numbers are so because neither of the numbers contains exactly ten digits.

In summary, we've written a script that looks at two different phone numbers and determines if the phone numbers are a valid phone number defined by the number of characters of each number. The script also returns one number based on whether or not the number is correct and prioritizes phone number 1 over 2. Again, from this work, we see that out of the 14 phone numbers, four of the phone numbers are invalid, and the remaining ten are valid since they contain exactly ten digits.

In [10]:

```
df2
```

Out[10]:

	customerid	phonenumber1	phonenumber2	phone1len	phone2len	customerphone
0	1000	9113458738	11148970949	10	11	9113458738
1	1001	9013458736	1104897094	10	10	9013458736
2	1002	8913458734	0	10	1	8913458734
3	1003	8813458732	10848970943	10	11	8813458732
4	1004	aaa	10748970941	3	11	invalid_phone
5	1005	8613458728	10648970939	10	11	8613458728
6	1006	851345872	10548970937	9	11	invalid_phone
7	1007	8413458724	4	10	1	8413458724
8	1008	8313458722	10348970933	10	11	8313458722
9	1009	0	10248970931	1	11	invalid_phone
10	1010	8113458718	1014897092	10	10	8113458718
11	1011	8013458716	10048970927	10	11	8013458716
12	1012	0	99948970925	1	11	invalid_phone
13	1013	7813458712	98248970923	10	11	7813458712

In [ ]: