Final Project Milestone 3

April 15, 2022

- CIS 9440 - Data Warehousing for Analytics
- Final Project Milestone 3
- Group Number - **17**
- Student – **KWANG HEUM YEON**

1 ETL Process

This ETL process is for the final project of CIS 9440, Zicklin School of Business in Baruch College. The overall guideline and most of the codes for the process was provided by the professor. The KPI of the project is as stated in project milestone #2. There are three separate datasets to get the KPI. Two of them were extracted from the NYC open data(https://opendata.cityofnewyork.us/) and the other referred to New York Demographics by Cubit (https://www.newyork-demographics.com/zip_codes_by_population). We have 10 stages to get the planned result and those are briefly as follows:

(1) Extract the first two datasets, merge, and clean

(2) Merge the third dataset and clean (*We will combine the third dataset with the completed set narrowed down the range of the dataset to avoid file size issues.)

(3) Create fact tables and each dimension tables

(4) Deliver fact tables and each dimension table to the data warehouse (Big Query)

(5) Appendix

Each code cell will contain a comment. In the appendix, we will attach the test images screen-captured from the data warehouse (Big Query) and BI application (Tableau) to check if the delivered tables are properly recognized.

1.1 Step 1: Extract data

1. connect to NYC Open Data with API Key
2. pull specific dataset as a pandas dataframe
3. Look at shape of extracted data

[1]:
```
# import libraries
import pandas as pd
from sodapy import Socrata
from google.cloud import bigquery
```

```
from google.oauth2 import service_account
```

1.1.1 Bicycle Counts

[2]:
```
data_url = 'data.cityofnewyork.us'     # The Host Name for the API endpoint (the␣
↪https:// part will be added automatically)
data_set = 'uczf-rk3c'     # The data set at the API endpoint
app_token = 'EBzL4gV7ZLeOF6y5zsvwWYm3M'    # The App Token code created in the␣
↪prior steps

# full URL to look at data on NYC Open Data
# https://data.cityofnewyork.us/resource/uczf-rk3c.json
```

[3]:
```
# create the client that points to the API endpoint
client = Socrata(data_url, app_token, timeout = 200)  #time limit run this cell:
↪ 200 sec
```

[4]:
```
print(f"client name is: {client}")
print(f"client data type is: {type(client)}")
```

```
client name is: <sodapy.socrata.Socrata object at 0x000001CB2A3D4130>
client data type is: <class 'sodapy.socrata.Socrata'>
```

[5]:
```
# test the connection to NYC Open Data

# retrieve the first 100 rows from the data_set
test_results = client.get(data_set, limit = 100)

# the test_results are returned as JSON object from the API
# the sodapy library converts this JSON object to a python list of dictionaries
# now, convert the list of dictionaries to a pandas data frame
test_results_df = pd.DataFrame.from_records(test_results)
```

[6]:
```
# examine the test_results_df pandas dataframe
test_results_df.head()
```

[6]:

	id1	counts	date	status	site
0	0	41	2012-08-31T00:00:00.000	4	100005020
1	1	52	2012-08-31T00:15:00.000	4	100005020
2	2	38	2012-08-31T00:30:00.000	4	100005020
3	3	36	2012-08-31T00:45:00.000	4	100005020
4	4	40	2012-08-31T01:00:00.000	4	100005020

sodapy client.get parameters 1. select 2. where 3. order 4. limit 5. group

[7]:
```
# next, get the total number of records in our the entire data set
total_record_count = client.get(data_set, select = "COUNT(*)")
print(f"total records in {data_set}: {total_record_count}")
```

```
total records in uczf-rk3c: [{'COUNT': '4628092'}]
```

[8]:
```python
# next, get the total number of records in our target data set
target_record_count = client.get(data_set,
                                 where = "date > '2021-01-01'",
                                 select= "COUNT(*)")
print(f"target records in {data_set}: {target_record_count}")
```

```
target records in uczf-rk3c: [{'COUNT': '620494'}]
```

[9]:
```python
# loop through data set to pull all rows in chunks (cannot pull all rows at␣
↪once)

# measure time this function takes
import time
start_time = time.time()

start = 0              # start at 0
chunk_size = 2000      # fetch 2000 rows at a time
results = []           # empty out our result list
record_count = target_record_count

while True:

    # fetch the set of records starting at 'start'
    results.extend(client.get(data_set,
                              where = "date > '2021-01-01'",
                              offset = start,
                              limit = chunk_size))

    # update the starting record number
    start = start + chunk_size

    # if we have fetched all of the records (we have reached record_count),␣
↪exit loop
    if (start > int(record_count[0]['COUNT'])):
        break

# convert the list into a pandas data frame
data = pd.DataFrame.from_records(results)

end_time = time.time()
print(f"loop to {round(end_time - start_time, 1)} seconds")

data.info()
```

```
loop to 102.8 seconds
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 620494 entries, 0 to 620493
Data columns (total 5 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id1     620494 non-null  object
 1   counts  620360 non-null  object
 2   date    620494 non-null  object
 3   status  620360 non-null  object
 4   site    620494 non-null  object
dtypes: object(5)
memory usage: 23.7+ MB
```

[10]:
```
data.head(2)
```

[10]:

	id1	counts	date	status	site
0	145537	11	2021-01-01T00:15:00.000	0	100009425
1	248161	5	2021-01-01T00:15:00.000	0	100009426

[11]:
```
count = data.copy()
```

1.1.2 Bicycle Sites

[12]:
```
data_url = 'data.cityofnewyork.us'     # The Host Name for the API endpoint (the␣
↪https:// part will be added automatically)
data_set = 'smn3-rzf9'     # The data set at the API endpoint
app_token = 'EBzL4gV7ZLe0F6y5zsvwWYm3M'    # The App Token code created in the␣
↪prior steps

# full URL to look at data on NYC Open Data
# https://data.cityofnewyork.us/resource/smn3-rzf9.json
```

[13]:
```
# create the client that points to the API endpoint
client = Socrata(data_url, app_token, timeout = 200)  #time limit run this cell:
↪ 200 sec
```

[14]:
```
print(f"client name is: {client}")
print(f"client data type is: {type(client)}")
```

```
client name is: <sodapy.socrata.Socrata object at 0x000001CB2A467790>
client data type is: <class 'sodapy.socrata.Socrata'>
```

[15]:
```
# test the connection to NYC Open Data

# retrieve the first 100 rows from the data_set
test_results = client.get(data_set, limit = 100)

# the test_results are returned as JSON object from the API
# the sodapy library converts this JSON object to a python list of dictionaries
```

```
# now, convert the list of dictionaries to a pandas data frame
test_results_df = pd.DataFrame.from_records(test_results)
```

[16]:
```
# examine the test_results_df pandas dataframe
test_results_df.head(2)
```

[16]:
```
  id                                 name   latitude  longitude  \
0  0  Manhattan Bridge 2012 Test Bike Counter   40.69981  -73.98589
1  5    Ed Koch Queensboro Bridge Shared Path  40.751038  -73.94082

              domain       site                      timezone interval  \
0  New York City DOT  100005020  (UTC-05:00) US/Eastern;DST       15
1  New York City DOT  100009428  (UTC-05:00) US/Eastern;DST       15

  :@computed_region_efsh_h5xi :@computed_region_f5dn_yrer  \
0                       16865                          68
1                       16858                          53

  :@computed_region_yeji_bk3q :@computed_region_92fq_4b7q  \
0                           2                          38
1                           3                          33

  :@computed_region_sbqj_enih      counter
0                          54          NaN
1                          66  Y2H19111445
```

sodapy client.get parameters 1. select 2. where 3. order 4. limit 5. group

[17]:
```
# next, get the total number of records in our the entire data set
total_record_count = client.get(data_set, select = "COUNT(*)")
print(f"total records in {data_set}: {total_record_count}")
```

```
total records in smn3-rzf9: [{'COUNT': '26'}]
```

[18]:
```
# extract the entire data set
target_record_count = client.get(data_set,
                                 select= "COUNT(*)")
print(f"target records in {data_set}: {target_record_count}")
```

```
target records in smn3-rzf9: [{'COUNT': '26'}]
```

[19]:
```
# loop through data set to pull all rows in chunks (cannot pull all rows at␣
 ↪once)

# measure time this function takes
import time
start_time = time.time()

start = 0              # start at 0
```

```python
chunk_size = 2000      # fetch 2000 rows at a time
results = []           # empty out our result list
record_count = target_record_count

while True:

    # fetch the set of records starting at 'start'
    results.extend(client.get(data_set,
                              offset = start,
                              limit = chunk_size))

    # update the starting record number
    start = start + chunk_size

    # if we have fetched all of the records (we have reached record_count),␣
↪exit loop
    if (start > int(record_count[0]['COUNT'])):
        break

# convert the list into a pandas data frame
data = pd.DataFrame.from_records(results)

end_time = time.time()
print(f"loop to {round(end_time - start_time, 1)} seconds")

data.info()
```

```
loop to 0.2 seconds
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26 entries, 0 to 25
Data columns (total 14 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   id                           26 non-null     object
 1   name                         26 non-null     object
 2   latitude                     26 non-null     object
 3   longitude                    26 non-null     object
 4   domain                       26 non-null     object
 5   site                         26 non-null     object
 6   timezone                     26 non-null     object
 7   interval                     26 non-null     object
 8   :@computed_region_efsh_h5xi  22 non-null     object
 9   :@computed_region_f5dn_yrer  23 non-null     object
 10  :@computed_region_yeji_bk3q  23 non-null     object
 11  :@computed_region_92fq_4b7q  23 non-null     object
 12  :@computed_region_sbqj_enih  23 non-null     object
 13  counter                      16 non-null     object
dtypes: object(14)
```

```
memory usage: 3.0+ KB
```

[20]:
```
data.head(2)
```

[20]:
```
   id                                   name   latitude  longitude  \
0   0  Manhattan Bridge 2012 Test Bike Counter   40.69981  -73.98589
1   5    Ed Koch Queensboro Bridge Shared Path  40.751038  -73.94082

             domain       site                      timezone interval  \
0  New York City DOT  100005020  (UTC-05:00) US/Eastern;DST       15
1  New York City DOT  100009428  (UTC-05:00) US/Eastern;DST       15

  :@computed_region_efsh_h5xi :@computed_region_f5dn_yrer  \
0                       16865                          68
1                       16858                          53

  :@computed_region_yeji_bk3q :@computed_region_92fq_4b7q  \
0                           2                          38
1                           3                          33

  :@computed_region_sbqj_enih      counter
0                          54          NaN
1                          66  Y2H19111445
```

[21]:
```
site = data.copy()
```

1.2 Step 2: Data Profiling

1. Distinct values per column
2. Null values per column
3. Summary statistics per numeric column

[22]:
```
# merge dataset, 'counts' and 'sites'
data = pd.merge(count, site, on = 'site', how = 'inner')
original_data = data.copy()
data.head(2)
```

[22]:
```
       id1 counts                     date status       site id  \
0  145537     11  2021-01-01T00:15:00.000      0  100009425  2
1  145538      8  2021-01-01T00:30:00.000      0  100009425  2

                 name     latitude     longitude             domain  \
0  Prospect Park West  40.67128846  -73.97138165  New York City DOT
1  Prospect Park West  40.67128846  -73.97138165  New York City DOT

                        timezone interval :@computed_region_efsh_h5xi  \
0  (UTC-05:00) US/Eastern;DST       15                          NaN
1  (UTC-05:00) US/Eastern;DST       15                          NaN
```

```
  :@computed_region_f5dn_yrer :@computed_region_yeji_bk3q  \
0                          14                          2
1                          14                          2

  :@computed_region_92fq_4b7q :@computed_region_sbqj_enih      counter
0                          27                         50  Y2H13094304
1                          27                         50  Y2H13094304
```

[23]:
```
# check if all columns are merged without Key column, 'site'
print(f"True if all columns are successfully merged:    {count.shape[1] + site.
 ↪shape[1] -1 == data.shape[1]}")
print(f"Zero if all sites are successfully merged:      {data.name.isna().
 ↪sum()}")
```

```
True if all columns are successfully merged:    True
Zero if all sites are successfully merged:      0
```

[24]:
```
# what are the columns in our dataframe?
data.columns
```

[24]:
```
Index(['id1', 'counts', 'date', 'status', 'site', 'id', 'name', 'latitude',
       'longitude', 'domain', 'timezone', 'interval',
       ':@computed_region_efsh_h5xi', ':@computed_region_f5dn_yrer',
       ':@computed_region_yeji_bk3q', ':@computed_region_92fq_4b7q',
       ':@computed_region_sbqj_enih', 'counter'],
      dtype='object')
```

[25]:
```
# select required columns

data = data[['date', 'site', 'name', 'latitude', 'longitude', 'counts']]
data.reset_index(drop = True, inplace = True)
data.head(2)
```

[25]:
```
                      date       site                name     latitude  \
0  2021-01-01T00:15:00.000  100009425  Prospect Park West  40.67128846
1  2021-01-01T00:30:00.000  100009425  Prospect Park West  40.67128846

      longitude counts
0  -73.97138165     11
1  -73.97138165      8
```

[26]:
```
# create a dataframe to gather information about each column
data_profiling_df = pd.DataFrame(columns = ["column_name",
                                            "column_type",
                                            "unique_values",
                                            "duplicate_values",
                                            "null_values",
```

```
                                                   "non_null_values",
                                                   "percent_null"])
```

[27]:
```
# loop through each column to add rows to the data_profiling_df dataframe
for column in data.columns:

    info_dict = {}

    try:
        info_dict["column_name"] = column
        info_dict["column_type"] = data[column].dtypes
        info_dict["unique_values"] = len(data[column].unique())
        info_dict["duplicate_values"] = (data[column].shape[0] - data[column].
 ↪isna().sum()) - len(data[column].unique())
        info_dict["null_values"] = data[column].isna().sum()
        info_dict["non_null_values"] = data[column].shape[0] - data[column].
 ↪isna().sum()
        info_dict["percent_null"] = round((data[column].isna().sum()) /␣
 ↪(data[column].shape[0]), 3)

    except:
        print(f"unable to read column: {column}")

    data_profiling_df = data_profiling_df.append(info_dict, ignore_index=True)

data_profiling_df.sort_values(by = ['unique_values', "non_null_values"],
                              ascending = [False, False],
                              inplace=True)
```

[28]:
```
data_profiling_df
```

[28]:
```
  column_name column_type unique_values duplicate_values null_values  \
0        date      object         43679           576815           0
5      counts      object           336           620024         134
1        site      object            15           620479           0
2        name      object            15           620479           0
3    latitude      object            13           620481           0
4   longitude      object            13           620481           0

  non_null_values  percent_null
0          620494           0.0
5          620360           0.0
1          620494           0.0
2          620494           0.0
3          620494           0.0
4          620494           0.0
```

1.3 Step 3: Data Cleansing (Initial)

1. drop unneeded columns
2. drop duplicate rows
3. check for outliers

[29]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 620494 entries, 0 to 620493
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   date       620494 non-null  object
 1   site       620494 non-null  object
 2   name       620494 non-null  object
 3   latitude   620494 non-null  object
 4   longitude  620494 non-null  object
 5   counts     620360 non-null  object
dtypes: object(6)
memory usage: 28.4+ MB
```

[30]:
```
data[data.duplicated()]
```

[30]:
```
Empty DataFrame
Columns: [date, site, name, latitude, longitude, counts]
Index: []
```

[31]:
```
# find number of duplicate rows

print(f"number of duplicate rows: {len(data[data.duplicated()])}")
```

```
number of duplicate rows: 0
```

[32]:
```
# drop duplicate rows
## drop duplicates here
## print new shape of data

data = data.drop_duplicates(keep = 'first')
print(f"new shape of data: {data.shape}")
```

```
new shape of data: (620494, 6)
```

[33]:
```
len(data[data.duplicated()])
```

[33]:
```
0
```

[34]:
```
# find number of Null rows

print(data.isna().sum(), '\n')
```

```
print(f"shape of date: {data.shape}")
```

```
date           0
site           0
name           0
latitude       0
longitude      0
counts       134
dtype: int64

shape of date: (620494, 6)
```

[35]:
```
# delete Null rows

data.dropna(inplace = True)
```

[36]:
```
# print new shape of data

print(data.isna().sum(), '\n')
print(f"new shape of date: {data.shape}")
```

```
date         0
site         0
name         0
latitude     0
longitude    0
counts       0
dtype: int64

new shape of date: (620360, 6)
```

[37]:
```
# data type change
df = pd.DataFrame()

df['date'] = data['date'].astype('datetime64[ns]').dt.date
df['time'] = data['date'].astype('datetime64[ns]').dt.time
df['site_address'] = data['name']
df['latitude'] = pd.to_numeric(data['latitude'])
df['longitude'] = pd.to_numeric(data['longitude'])
df['counts'] = data['counts'].astype(int)
```

[38]:
```
data = df.copy()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 620360 entries, 0 to 620493
Data columns (total 6 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
```

```
 0   date          620360 non-null  object
 1   time          620360 non-null  object
 2   site_address  620360 non-null  object
 3   latitude      620360 non-null  float64
 4   longitude     620360 non-null  float64
 5   counts        620360 non-null  int32
dtypes: float64(2), int32(1), object(3)
memory usage: 30.8+ MB
```

[39]:
```
# check the number of values having Zero
# if True, None zero value
data.all()
```

[39]:
```
date             True
time             True
site_address     True
latitude         True
longitude        True
counts          False
dtype: bool
```

We do not delete these values to find the number of check-frequency per bicycle site

[40]:
```
data.head(2)
```

[40]:
```
         date      time        site_address   latitude  longitude  counts
0  2021-01-01  00:15:00  Prospect Park West  40.671288 -73.971382      11
1  2021-01-01  00:30:00  Prospect Park West  40.671288 -73.971382       8
```

1.4 Step 4: Additional data

1.4.1 Site Zipcode

[41]:
```
# add Zipcode from external reference

# get site names
site_names = data.site_address.value_counts().reset_index()['index'].values
print(f"Number of unique site name: {len(site_names)}")
```

```
Number of unique site name: 15
```

[42]:
```
print(site_names[0:5],'\n')
print(site_names[5:10],'\n')
print(site_names[10:16],'\n')
```

```
['Prospect Park West' 'Manhattan Bridge Ped Path'
 'Williamsburg Bridge Bike Path' 'Ed Koch Queensboro Bridge Shared Path'
 'Staten Island Ferry']

['Pulaski Bridge' 'Kent Ave btw North 8th St and North 9th St'
```

```
 'Brooklyn Bridge Bike Path' 'Manhattan Bridge Display Bike Counter'
 'Amsterdam Ave at 86th St.']

['Columbus Ave at 86th St.' 'Manhattan Bridge Bike Comprehensive'
 'Comprehensive Brooklyn Bridge Counter' '8th Ave at 50th St.'
 'Brooklyn Bridge Bicycle Path (Roadway)']
```

Reference link: https://www.unitedstateszipcodes.org/

[43]:
```
# To double-check the zipcode referencing with the site name, we do not use␣
↪loop here.

data.loc[data['site_address'] == 'Prospect Park West', 'zipcode'] = 11215
data.loc[data['site_address'] == 'Manhattan Bridge Ped Path', 'zipcode'] = 10002
data.loc[data['site_address'] == 'Williamsburg Bridge Bike Path', 'zipcode'] =␣
↪10002
data.loc[data['site_address'] == 'Ed Koch Queensboro Bridge Shared Path',␣
↪'zipcode'] = 11101
data.loc[data['site_address'] == 'Staten Island Ferry', 'zipcode'] = 10301

data.loc[data['site_address'] == 'Pulaski Bridge', 'zipcode'] = 11101
data.loc[data['site_address'] == 'Kent Ave btw North 8th St and North 9th St',␣
↪'zipcode'] = 11249
data.loc[data['site_address'] == 'Brooklyn Bridge Bike Path', 'zipcode'] = 11214
data.loc[data['site_address'] == 'Manhattan Bridge Display Bike Counter',␣
↪'zipcode'] = 10002
data.loc[data['site_address'] == 'Amsterdam Ave at 86th St.', 'zipcode'] = 10024

data.loc[data['site_address'] == 'Columbus Ave at 86th St.', 'zipcode'] = 10024
data.loc[data['site_address'] == 'Manhattan Bridge Bike Comprehensive',␣
↪'zipcode'] = 10002
data.loc[data['site_address'] == 'Comprehensive Brooklyn Bridge Counter',␣
↪'zipcode'] = 10038
data.loc[data['site_address'] == '8th Ave at 50th St.', 'zipcode'] = 11220
data.loc[data['site_address'] == 'Brooklyn Bridge Bicycle Path (Roadway)',␣
↪'zipcode'] = 10038

# data type change
data['zipcode'] = data['zipcode'].astype(int)
data['zipcode'] = data['zipcode'].astype(str) # to merge with 'Zip per␣
↪population' dataset
```

[44]:
```
data.head(2)
```

[44]:
```
         date      time        site_address   latitude  longitude  counts  \
0  2021-01-01  00:15:00  Prospect Park West  40.671288 -73.971382      11
```

```
1  2021-01-01  00:30:00  Prospect Park West  40.671288 -73.971382       8

  zipcode
0   11215
1   11215
```

1.4.2 Population per zipcode

[45]:
```
zip_pop = pd.read_csv('https://raw.githubusercontent.com/cpasean/Projects/main/
↪zip%20per%20population.csv')
zip_pop.head(2)
```

[45]:
```
  Rank Zip Code Population
0    1    11368    112,088
1    2    11385    107,796
```

[46]:
```
# ignore warning message
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore")  # ignore runtime warning

# get only wanted columns
zip_pop = zip_pop[['Zip Code', 'Population']]
zip_pop.rename(columns = {'Zip Code' : 'zipcode', 'Population' : 'population'},␣
↪inplace = True)
```

[47]:
```
data.shape
```

[47]:
```
(620360, 7)
```

[48]:
```
data = pd.merge(data, zip_pop, how = 'inner', on = 'zipcode')
data.head(2)
```

[48]:
```
         date      time         site_address   latitude  longitude  counts  \
0  2021-01-01  00:15:00  Prospect Park West  40.671288 -73.971382      11
1  2021-01-01  00:30:00  Prospect Park West  40.671288 -73.971382       8

  zipcode population
0   11215     69,873
1   11215     69,873
```

[49]:
```
data.shape
```

[49]:
```
(576689, 8)
```

1.5 Step 5: Data Cleansing (Final)

[50]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 576689 entries, 0 to 576688
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   date          576689 non-null  object
 1   time          576689 non-null  object
 2   site_address  576689 non-null  object
 3   latitude      576689 non-null  float64
 4   longitude     576689 non-null  float64
 5   counts        576689 non-null  int32
 6   zipcode       576689 non-null  object
 7   population    576689 non-null  object
dtypes: float64(2), int32(1), object(5)
memory usage: 37.4+ MB
```

[51]:
```
# data type change as integer for zipcode and population

data['zipcode'] = data['zipcode'].astype(int)

pop = []
for i in data['population']:
    pop.append(int(i.replace(',', '')))
data['population'] = pop
```

[52]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 576689 entries, 0 to 576688
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   date          576689 non-null  object
 1   time          576689 non-null  object
 2   site_address  576689 non-null  object
 3   latitude      576689 non-null  float64
 4   longitude     576689 non-null  float64
 5   counts        576689 non-null  int32
 6   zipcode       576689 non-null  int32
 7   population    576689 non-null  int64
dtypes: float64(2), int32(2), int64(1), object(3)
memory usage: 35.2+ MB
```

[53]:
```python
# find number of duplicate rows

print(f"number of duplicate rows: {len(data[data.duplicated()])}")
```

```
number of duplicate rows: 0
```

[54]:
```python
# find number of Null rows

print(data.isna().sum(), '\n')
print(f"Final shape of date: {data.shape}")
```

```
date            0
time            0
site_address    0
latitude        0
longitude       0
counts          0
zipcode         0
population      0
dtype: int64

Final shape of date: (576689, 8)
```

[55]:
```python
data.head(2)
```

[55]:
```
         date      time         site_address   latitude  longitude  counts  \
0  2021-01-01  00:15:00  Prospect Park West  40.671288 -73.971382      11
1  2021-01-01  00:30:00  Prospect Park West  40.671288 -73.971382       8

   zipcode  population
0    11215       69873
1    11215       69873
```

1.6 Step 6: Create Location Dimension

[56]:
```python
# first, copy the entire table

location_dim = data.copy()
```

[57]:
```python
# second, subset for only the wanted columns in the dimension

location_dim = location_dim[['longitude',
                             'latitude',
                             'site_address',
                             'zipcode']]
```

[58]:
```python
# third, drop duplicate rows in dimension
```

```python
location_dim = location_dim.drop_duplicates(subset = ["site_address"], keep =␣
 ↪"first")
location_dim = location_dim.reset_index(drop = True)

location_dim.head()
```

[58]:

```
    longitude   latitude                           site_address  zipcode
0 -73.971382  40.671288                     Prospect Park West    11215
1 -73.994950  40.714573               Manhattan Bridge Ped Path    10002
2 -73.961450  40.710530           Williamsburg Bridge Bike Path    10002
3 -73.994750  40.715600  Manhattan Bridge Display Bike Counter    10002
4 -73.994750  40.715600    Manhattan Bridge Bike Comprehensive    10002
```

[59]:

```python
# fourth, add location_id as a surrogate key

location_dim.insert(0, "location_id",
                    range(10, 10+len(location_dim)))
```

[60]:

```python
location_dim.head(2)
```

[60]:

```
   location_id  longitude   latitude               site_address  zipcode
0           10 -73.971382  40.671288         Prospect Park West    11215
1           11 -73.994950  40.714573  Manhattan Bridge Ped Path    10002
```

[61]:

```python
# fifth, add the location_id to the Fact table

data = data.merge(location_dim[["site_address", "location_id"]],
                 left_on = "site_address",
                 right_on = "site_address",
                 how = "left")

data.head(2)

## merge with data to put location_id into Fact table
```

[61]:

```
         date      time        site_address   latitude  longitude  counts  \
0  2021-01-01  00:15:00  Prospect Park West  40.671288 -73.971382      11
1  2021-01-01  00:30:00  Prospect Park West  40.671288 -73.971382       8

   zipcode  population  location_id
0    11215       69873           10
1    11215       69873           10
```

[62]:

```python
# check latitude / longitude

import folium
```

```
lat = location_dim.latitude.mean()
long = location_dim.longitude.mean()

m = folium.Map(location=[lat, long], zoom_start = 9) # center location

for i in location_dim.index[:]:
    tooltip = location_dim.loc[i,"site_address"]
    lat = location_dim.loc[i,"latitude"]
    long = location_dim.loc[i,"longitude"]

    folium.Marker([lat, long], tooltip = tooltip).add_to(m)

# visual images are separately attached
m
```

[62]: <folium.folium.Map at 0x1cb32dfceb0>

1.7 Step 7: Create Time Dimension

[63]:
```
# first, copy the entire table

time_dim = data.copy()
```

[64]:
```
# second, subset for only the wanted columns in the dimension

time_dim = time_dim[['time']]
```

[65]:
```
# third, drop duplicate rows in dimension

time_dim = time_dim.drop_duplicates(subset = ["time"], keep = "first")
time_dim = time_dim.reset_index(drop = True)

time_dim.head()
```

[65]:

	time
0	00:15:00
1	00:30:00
2	00:45:00
3	01:00:00
4	01:15:00

[66]:
```
# fourth, add location_id as a surrogate key

time_dim.insert(0, "time_id", range(100, 100+len(time_dim)))
```

[67]:
```
time_dim.tail(2)
```

[67]:
```
    time_id      time
94      194  23:45:00
95      195  00:00:00
```

[68]:
```python
# fifth, add the location_id to the Fact table

data = data.merge(time_dim,
                 left_on = "time",
                 right_on = "time",
                 how = "left")

data.head(2)

## merge with data to put location_id into Fact table
```

[68]:
```
         date      time         site_address   latitude  longitude  counts  \
0  2021-01-01  00:15:00  Prospect Park West  40.671288 -73.971382      11
1  2021-01-01  00:30:00  Prospect Park West  40.671288 -73.971382       8

  zipcode  population  location_id  time_id
0   11215       69873           10      100
1   11215       69873           10      101
```

1.8 Step 8: Create Date Dimension

[69]:
```python
# first, create a BigQuery client to connect to BigQuery
from google.cloud import bigquery
from google.oauth2 import service_account

key_path = r'C:\Users\aicpa\Google␣
↪Drive\_CPADataScientistValueInvestor\_DataWarehouse\cis9440-340819-fdb3569fc29a.
↪json' # must edit to your credentials json file location
credentials = service_account.Credentials.from_service_account_file(key_path,
                                                                              ␣
↪scopes=["https://www.googleapis.com/auth/cloud-platform"],)
client = bigquery.Client(credentials = credentials,
                         project = credentials.project_id)
```

[70]:
```python
print(client)
```

```
<google.cloud.bigquery.client.Client object at 0x000001CB3B2A5C10>
```

[71]:
```python
sql_query = """
            SELECT
              CONCAT␣
↪(FORMAT_DATE("%Y",d),FORMAT_DATE("%m",d),FORMAT_DATE("%d",d)) as date_id,
              d AS full_date,
              FORMAT_DATE('%w', d) AS week_day,
```

```
                FORMAT_DATE('%A', d) AS day_name,
                EXTRACT(DAY FROM d) AS year_day,
                EXTRACT(WEEK FROM d) AS week,
                EXTRACT(WEEK FROM d) AS year_week,
                EXTRACT(MONTH FROM d) AS month,
                FORMAT_DATE('%B', d) as month_name,
                FORMAT_DATE('%Q', d) as fiscal_qtr,
                EXTRACT(YEAR FROM d) AS year,
                (CASE WHEN FORMAT_DATE('%A', d) IN ('Sunday', 'Saturday') THEN 0␣
 ↪ELSE 1 END) AS day_is_weekday,
              FROM (
                SELECT
                  *
                FROM
                  UNNEST(GENERATE_DATE_ARRAY('2021-01-01', '2023-01-01', INTERVAL␣
 ↪1 DAY)) AS d )
              """

# store extracted data in new dataframe
date_dim = client.query(sql_query).to_dataframe()

# validate that >0 stories have been extracted and return dataframe
if len(date_dim) > 0:
    print("date dimension created")
else:
    print("date dimension FAILED")
```

```
date dimension created
```

[72]:
```
date_dim.head()
```

[72]:

	date_id	full_date	week_day	day_name	year_day	week	year_week	month
0	20210101	2021-01-01	5	Friday	1	0	0	1
1	20210102	2021-01-02	6	Saturday	2	0	0	1
2	20210103	2021-01-03	0	Sunday	3	1	1	1
3	20210104	2021-01-04	1	Monday	4	1	1	1
4	20210105	2021-01-05	2	Tuesday	5	1	1	1

	month_name	fiscal_qtr	year	day_is_weekday
0	January	1	2021	1
1	January	1	2021	0
2	January	1	2021	0
3	January	1	2021	1
4	January	1	2021	1

[73]:
```
# create date_id column in the Fact Table
```

```
data['date_id'] = data['date'].apply(lambda x: pd.to_datetime(x).
↪strftime("%Y%m%d"))
```

1.9 Step 9: Creating Fact(s)

[74]:
```
# Creating Bicycle Fact Table

fact_bicycle = data[["date_id",
                     "time_id",
                     "location_id",
                     "counts"]]
fact_bicycle.sample(5)
```

[74]:

	date_id	time_id	location_id	counts
175605	20210110	157	14	81
372684	20210831	188	18	39
298200	20220112	182	16	19
283296	20210810	158	16	20
280446	20210711	188	16	23

[75]:
```
# Creating Census Fact Table

fact_census = data[["date_id",
                    "location_id",
                    "population"]]
fact_census.sample(5)
```

[75]:

	date_id	location_id	population
163750	20211208	13	74479
515792	20210119	22	23311
205720	20211120	14	74479
226197	20210323	15	31366
544155	20211111	22	23311

1.10 Step 10: Deliver Facts and Dimensions to Data Warehouse (BigQuery)

[76]:
```
# build a function to load tables to BigQuery

def load_table_to_bigquery(df, table_name):

    dataset_id = 'cis9440-340819.final_project_etl_nyc_bicycle'

    dataset_ref = client.dataset(dataset_id)
    job_config = bigquery.LoadJobConfig()
    job_config.autodetect = True
    job_config.write_disposition = "WRITE_TRUNCATE"
```

```python
    upload_table_name = f"cis9440-340819.final_project_etl_nyc_bicycle.
↪{table_name}"

    load_job = client.load_table_from_dataframe(df,
                                                upload_table_name,
                                                job_config = job_config)

    print(f"starting job {load_job}")
```

[77]:
```python
load_table_to_bigquery(df = location_dim,
                       table_name = "location_dim")
```

```
starting job LoadJob<project=cis9440-340819, location=US, id=40b6c24f-3550-41ad-
bcdb-6e044b5a986f>
```

[78]:
```python
load_table_to_bigquery(df = time_dim,
                       table_name = "time_dim")
```

```
starting job LoadJob<project=cis9440-340819, location=US,
id=c1c79eef-5419-48d6-9886-8c373f7eff59>
```

[79]:
```python
load_table_to_bigquery(df = date_dim,
                       table_name = "date_dim")
```

```
starting job LoadJob<project=cis9440-340819, location=US,
id=b63f7c91-7a19-42d7-b34d-129977c7318a>
```

[80]:
```python
load_table_to_bigquery(df = fact_bicycle,
                       table_name = "fact_bicycle")
```

```
starting job LoadJob<project=cis9440-340819, location=US,
id=9072e453-baba-42d2-834a-9cff73eaf8b3>
```

[81]:
```python
load_table_to_bigquery(df = fact_census,
                       table_name = "fact_census")
```

```
starting job LoadJob<project=cis9440-340819, location=US,
id=a820ab20-2460-494c-88eb-ec465bb7ad06>
```

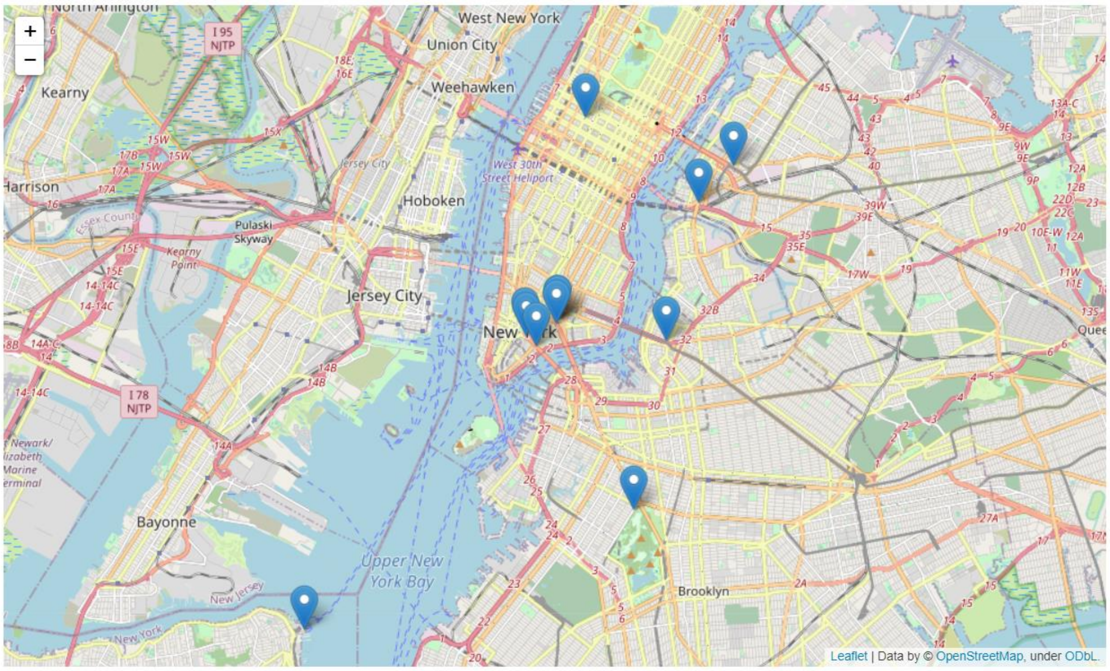
[82]:
```python
# send to *.csv

location_dim.to_csv('location_dim.csv')
time_dim.to_csv('time_dim.csv')
date_dim.to_csv('date_dim.csv')
fact_bicycle.to_csv('fact_bicycle.csv')
fact_census.to_csv('fact_census.csv')
```

1.11 Appendix

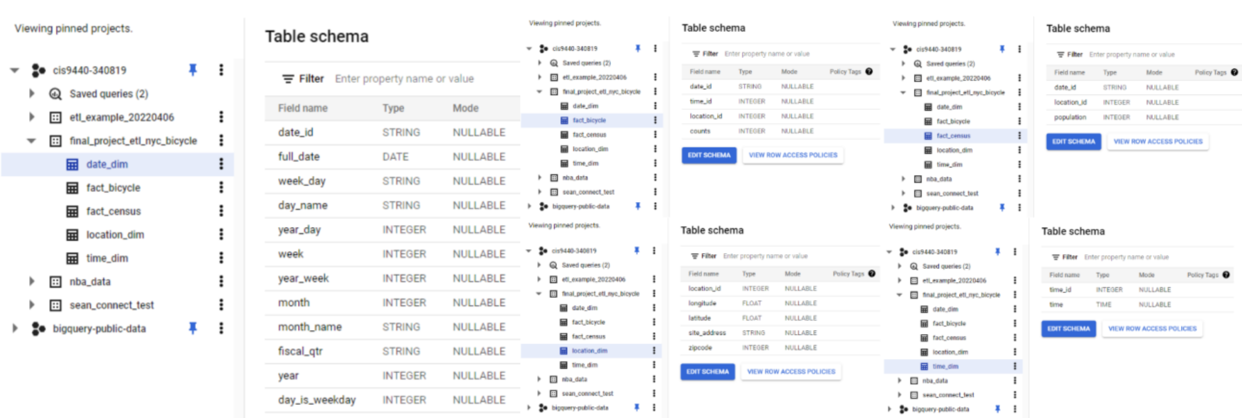
Test Image #1: Site location

```
In [100]: # check latitude / longitude

          import folium

          lat = location_dim.latitude.mean()
          long = location_dim.longitude.mean()

          m = folium.Map(location=[lat, long], zoom_start = 9) # center location

          for i in location_dim.index[:]:
              tooltip = location_dim.loc[i,"site_address"]
              lat = location_dim.loc[i,"latitude"]
              long = location_dim.loc[i,"longitude"]

              folium.Marker([lat, long], tooltip = tooltip).add_to(m)
          m
```

Out[100]:



Test Image #2: Big Query



Test Image #3: Tableau