Team 3
Analysis of 311 Noise Complaints

CIS 4400 Data Warehousing for Analytics
Section CMWA - Dr. Ramah Al Balawi

Team Members

Name	Email
Christian Casino	<CHRISTIAN.CASINO@baruchmail.cuny.edu>
Emil Purisic	<EMIL.PURISIC@baruchmail.cuny.edu>
Hali Siew	<HALI.SIEW@baruchmail.cuny.edu>
Jason Chen	<JASON.CHEN15@baruchmail.cuny.edu>
Rosie Morgan	<ROSIANE.MORGAN@baruchmail.cuny.edu>

Team 3 Project Introduction

An introduction section similar to the proposal section, including the narrative description of the business or organization used for the data warehouse being created and a description of the source data. Be sure to include appropriate citations of each data source (web site, curator, date accessed, etc.)

For this project we have analyzed two different dataset, “311 Service Requests from 2010 to Present” and the Weather dataset. The 311 Service Request is an online dataset consisting of all the different complaints made to 311. The data provides information like complaint type, location and time. The weather dataset consists of the daily weather for the past 6 years, providing date, location, temperature, precipitation amount, dewpoint and wind speed.

With the two datasets, we have mapped the following KPIs:

- Number of “Residential” Noise complaints per day/month
- Number of “Commercial” Noise complaints per day/month
- Number of Complaints per Complaint Type
- Average temperature per month
- Average temperature at location/borough

Citation:

DoITT, & 311. (2011, October 10). *311 service requests from 2010 to present: NYC Open Data*. 311 Service Requests from 2010 to Present | NYC Open Data. Retrieved December 10, 2022, from https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9

Dimensional Model Diagram

Compliant ID	Date (of compliant)	Compliant Type	Descriptor	Location Type	Incident Zip	Incident Address	Street Name	City	Borough
155588721	10/5/2022	Noise - Street/SideWalk	Loud Talking	Street/Sidewalk	11229	1815 East 17 Street	East 17 Street	Brooklyn	Brooklyn

Dataset 1: 311 Complaints

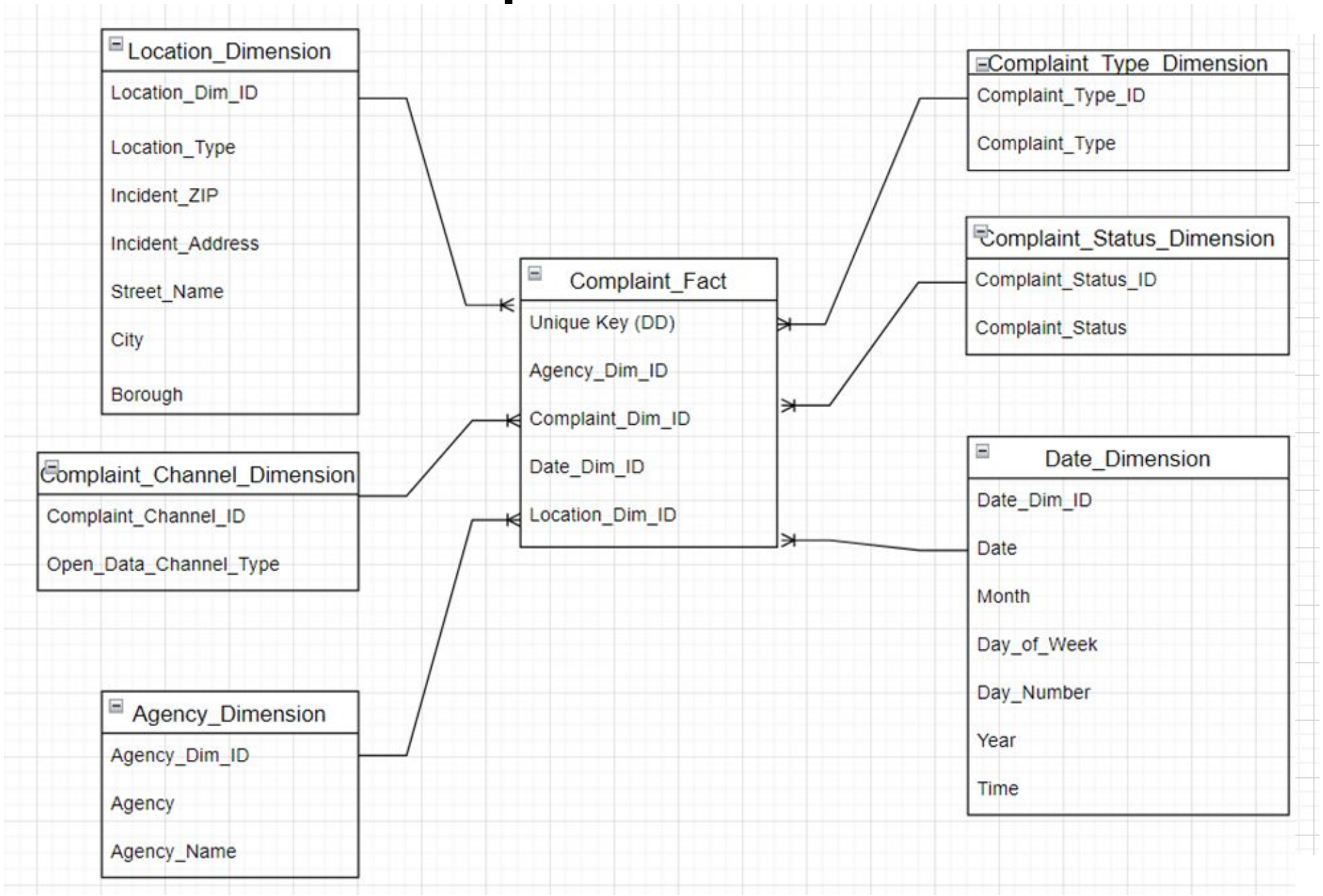
Date	Location	Maximum temperature	Minimum temperature	Average temperature	Precipitation	New Snow	Snow Depth
10/4/2022	NY Central Area	55	46	50.5	1.85	0	0

Dataset 2: Weather dataset

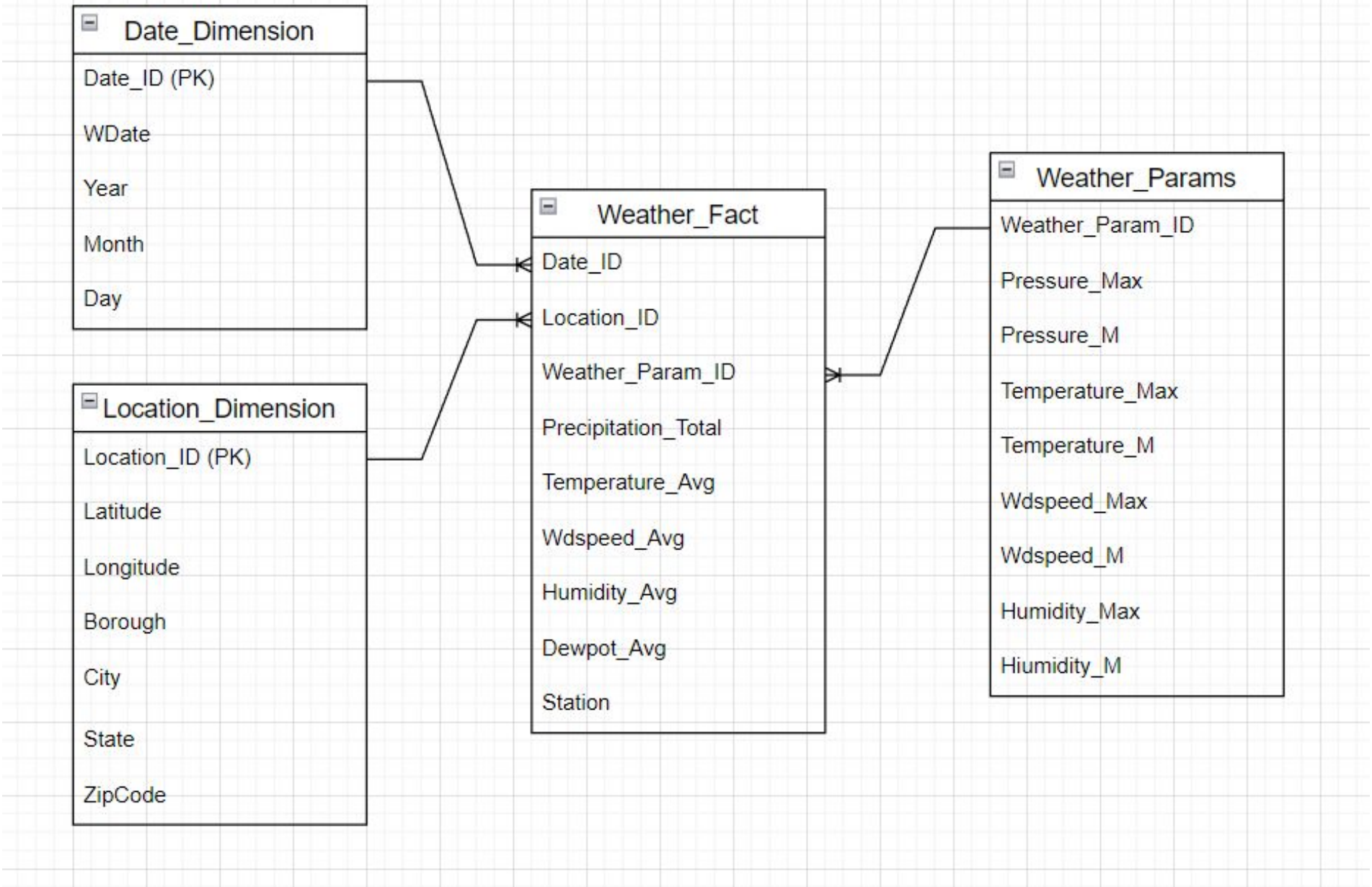
WDate	Borough	Temperature_Max	Temperature_Avg	Temperature_M	Precipitation_Total
10/4/2022	Manhattan	52.9	46	39.9	1.85

Dataset 3: Weather dataset 2

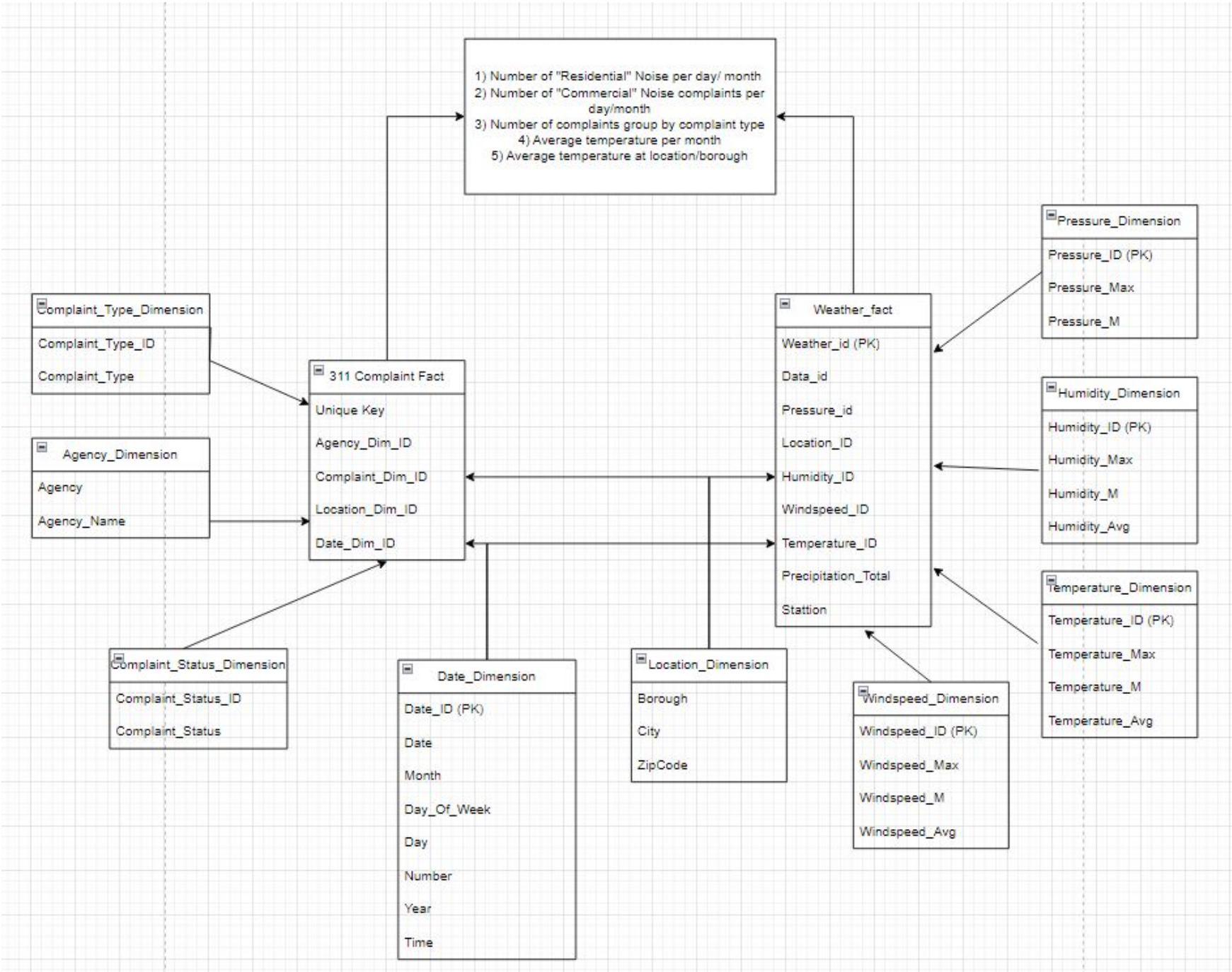
Updated 311 Complaints Dimensional Model



Updated Weather Dataset Dimensional Model



Updated Integrated Warehouse Model



311 Complaints ETL Programming

A description and screen pictures of each of the ETL processes and any custom code used to process the source data

1. Uploading necessary packages, loading in dataset into IDE (Google Colab)

```
1 import pandas as pd
2 from google.cloud import bigquery # package allows us to access GCP platform project
3 import os
```

```
[2]  1 ! gdown --id 1olZAnsDfC2PgrQFPQ5RBNkhwVvB5h8Iv
```

```
/local/lib/python3.8/dist-packages/gdown/cli.py:127: FutureWarning: Option `--id` was depr
rnings.warn(
loading...
: https://drive.google.com/uc?id=1olZAnsDfC2PgrQFPQ5RBNkhwVvB5h8Iv
/content/311_Service_Requests_from_2010_to_Present.csv
 63.0M/63.0M [00:00<00:00, 89.3MB/s]
```

2. Creating dataframe for newly loaded dataset

```
1 df = pd.read_csv("311_Service_Requests_from_2010_to_Present.csv")
2 print(len(df))
3 df.head()
```

```
108586
```

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Vehicle Type	Taxi Company Borough	Taxi Pick Up Location	Bridge Highway Name	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Lat
0	46367528	06/02/2020 06:09:00 PM	06/03/2020 07:00:00 PM	DEP	Department of Environmental Protection	Noise	Noise, Ice Cream Truck (NR4)	NaN	11228	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.6
1	43207999	07/06/2019 01:03:33 AM	07/06/2019 04:48:42 AM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10458	2779 BRIGGS AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.8
2	43211313	07/06/2019 07:49:10 PM	07/06/2019 11:24:59 PM	NYPD	New York City Police Department	Noise - Residential	Loud Music/Party	NaN	10458	357 EAST 201 STREET	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.8
3	43223100	07/09/2019 12:20:14 AM	07/09/2019 06:55:43 AM	NYPD	New York City Police Department	Noise - Vehicle	Car/Truck Music	Street/Sidewalk	10458	2711 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.8
4	43228977	07/08/2019 04:36:32 AM	07/08/2019 05:29:55 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.7

3. Data Profiling 1- Create new dataframe called master_columns, which contains columns for each attribute in 311 Complaints Dimensional Model

```
master_columns =df[[ 'Created Date', 'Agency', 'Agency Name', 'Complaint Type', 'Location Type','Incident Zip',
                    'Incident Address', 'Street Name', 'City', 'Status', 'Borough',
'Open Data Channel Type']]
print(len(master_columns))
master_columns.head()
```

```
108586
```

	Created Date	Agency	Agency Name	Complaint Type	Location Type	Incident Zip	Incident Address	Street Name	City	Status	Borough	Open Data Channel Type
0	06/02/2020 06:09:00 PM	DEP	Department of Environmental Protection	Noise	NaN	11228	NaN	NaN	BROOKLYN	Closed	BROOKLYN	ONLINE
1	07/06/2019 01:03:33 AM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Street/Sidewalk	10458	2779 BRIGGS AVENUE	BRIGGS AVENUE	BRONX	Closed	BRONX	MOBILE
2	07/06/2019 07:49:10 PM	NYPD	New York City Police Department	Noise - Residential	NaN	10458	357 EAST 201 STREET	EAST 201 STREET	BRONX	Closed	BRONX	PHONE
3	07/09/2019 12:20:14 AM	NYPD	New York City Police Department	Noise - Vehicle	Street/Sidewalk	10458	2711 VALENTINE AVENUE	VALENTINE AVENUE	BRONX	Closed	BRONX	MOBILE
4	07/08/2019 04:36:32 AM	NYPD	New York City Police Department	Noise - Commercial	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	NORTHERN BOULEVARD	JACKSON HEIGHTS	Closed	QUEENS	ONLINE

4. Data Profiling 2- Use the '.isna' and '.any()' methods to see which columns in the master_columns dataframe have missing values

```
1 master_columns.isna().any()
```

```
Created Date                False
Agency                      False
Agency Name                 False
Complaint Type              False
Location Type               False
Incident Zip                False
Incident Address            False
Street Name                 False
City                         True
Status                      False
Borough                     False
Open Data Channel Type      False
dtype: bool
```

4. Data Profiling 2- Use the .fillna method to replace 'NaN' values in the master_columns[['City']] column with the name of the borough (which will never be null according to the master_columns.isna().any() method in the previous slide.

* If more time was allotted, we would address the caveat raised in the code snippet below

```
# Filling the columns that are known to have NaN values and filling them with
# 'Not Applicable' will remove all remaining NaN values from the dataframe

# The borough column will always have a value according to the data profiling done
# Most Cities in New York City share the same name as their borough (for the exception of Queens)
# So fill NaN values from the City column with the name of the borough
master_columns[['City']] = master_columns[['City']].fillna(master_columns[['Borough']])

print(len(master_columns))
master_columns.head()
```

```
108586
/usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

	Created Date	Agency	Agency Name	Complaint Type	Location Type	Incident Zip	Incident Address	Street Name	
0	06/02/2020 06:09:00 PM	DEP	Department of Environmental Protection	Noise	Not Applicable	11228	Not Applicable	Not Applicable	BR
1	07/06/2019 01:03:33 AM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Street/Sidewalk	10458	2779 BRIGGS AVENUE	BRIGGS AVENUE	
2	07/06/2019 07:49:10 PM	NYPD	New York City Police Department	Noise - Residential	Not Applicable	10458	357 EAST 201 STREET	EAST 201 STREET	
3	07/09/2019 12:20:14 AM	NYPD	New York City Police Department	Noise - Vehicle	Street/Sidewalk	10458	2711 VALENTINE AVENUE	VALENTINE AVENUE	
4	07/08/2019 04:36:32 AM	NYPD	New York City Police Department	Noise - Commercial	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	NORTHERN BOULEVARD	J

5. Creating dataframes for dimensions & doing further transformations in preparation for loading data in Google BigQuery (Location Dimension Table)

5-A. Creating a new dataframe with columns representing the attributes of the Location Dimension and renaming them to correspond to naming criteria of columns on Google BigQuery

```
location_dim_tbl = master_columns[['Location Type', 'Incident Zip', 'Incident Address', 'Street Name', 'City', 'Borough']]
print(len(location_dim_tbl))
location_dim_tbl.head()
```

```
108586
```

	Location Type	Incident Zip	Incident Address	Street Name	City	Borough
0	Not Applicable	11228	Not Applicable	Not Applicable	BROOKLYN	BROOKLYN
1	Street/Sidewalk	10458	2779 BRIGGS AVENUE	BRIGGS AVENUE	BRONX	BRONX
2	Not Applicable	10458	357 EAST 201 STREET	EAST 201 STREET	BRONX	BRONX
3	Street/Sidewalk	10458	2711 VALENTINE AVENUE	VALENTINE AVENUE	BRONX	BRONX
4	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	NORTHERN BOULEVARD	JACKSON HEIGHTS	QUEENS

```
# We need to rename the column names to have underscores as required for columns in GBQ
location_dim_tbl = location_dim_tbl.rename(columns = {'Location Type':'Location_Type', 'Incident Zip': 'Incident_Zip', 'Incident Address':'Incident_Address', 'Street Name': 'Street_Name'})
print(len(location_dim_tbl))
location_dim_tbl.head()
```

```
108586
```

	Location_Dim_ID	Location_Type	Incident_Zip	Incident_Address	Street_Name	City	Borough
0	Not Applicable, BROOKLYN, NEW YORK, 11228	Not Applicable	11228	Not Applicable	Not Applicable	BROOKLYN	BROOKLYN
1	2779 BRIGGS AVENUE, BRONX, NEW YORK, 10458	Street/Sidewalk	10458	2779 BRIGGS AVENUE	BRIGGS AVENUE	BRONX	BRONX
2	357 EAST 201 STREET, BRONX, NEW YORK, 10458	Not Applicable	10458	357 EAST 201 STREET	EAST 201 STREET	BRONX	BRONX
3	2711 VALENTINE AVENUE, BRONX, NEW YORK, 10458	Street/Sidewalk	10458	2711 VALENTINE AVENUE	VALENTINE AVENUE	BRONX	BRONX
4	95-15B NORTHERN BOULEVARD, JACKSON HEIGHTS, NE...	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	NORTHERN BOULEVARD	JACKSON HEIGHTS	QUEENS

5-B. Creating a composite key column called 'Location_Dim_ID" within the location_dim_tbl dataframe, which will temporarily serve as the surrogate key column for the table that will be loaded into GBQ

```
# Creating the composite key, which will temporarily server as the surrogate key for the data
#location_dim_tbl.insert(0,'Location_Dim_ID', range(1, 1+ len(location_dim_tbl)))

# Concatenating the columns which are crucial for distinguishing a row entry
# How to combine two columns of text: https://sparkbyexamples.com/pandas/pandas-combine-two-columns-of-text-in-dataframe/

location_dim_tbl.insert(0,'Location_Dim_ID', location_dim_tbl['Incident_Address'] + ', ' + location_dim_tbl['City'] + ', NEW YORK' + ', ' + location_dim_tbl['Incident_Zip'].astype(str))


print(len(location_dim_tbl)) # It turns out each row is unique, no duplicates
location_dim_tbl.head()
```

```
108586
```

	Location_Dim_ID	Location_Type	Incident_Zip	Incident_Address	Street_Name	City	Borough
0	Not Applicable, BROOKLYN, NEW YORK, 11228	Not Applicable	11228	Not Applicable	Not Applicable	BROOKLYN	BROOKLYN
1	2779 BRIGGS AVENUE, BRONX, NEW YORK, 10458	Street/Sidewalk	10458	2779 BRIGGS AVENUE	BRIGGS AVENUE	BRONX	BRONX
2	357 EAST 201 STREET, BRONX, NEW YORK, 10458	Not Applicable	10458	357 EAST 201 STREET	EAST 201 STREET	BRONX	BRONX
3	2711 VALENTINE AVENUE, BRONX, NEW YORK, 10458	Street/Sidewalk	10458	2711 VALENTINE AVENUE	VALENTINE AVENUE	BRONX	BRONX
4	95-15B NORTHERN BOULEVARD, JACKSON HEIGHTS, NE...	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	NORTHERN BOULEVARD	JACKSON HEIGHTS	QUEENS

5-C. Creating a copy of the location_dim_tbl dataframe called location_dim_tbl2 & drop duplicates from dataframe.

- The location_dim_tbl2 dataframe will be loaded into BigQuery as the Location Dimension Table, and the Location_Dim_ID index from the location_dim_tbl dataframe (the original dataframe), will be added to the fact table, which will be created later.
- Notice that the length of location_dim_tbl2 is 19,265 compared to 108,586 (as in location_dim_tbl)

```
[61] 1 # Create copy of dimension table, and drop duplicates. This dataframe will
     2 # be loaded in as Location Dimension Table
     3 location_dim_tbl2 = location_dim_tbl.copy()
     4 location_dim_tbl2 = location_dim_tbl2.drop_duplicates()
     5 print(len(location_dim_tbl2))
     6 location_dim_tbl2.head()
```

```
19265
```

	Location_Dim_ID	Location_Type	Incident_Zip	Incident_Address	Street_Name	City	Borough
0	Not Applicable, BROOKLYN, NEW YORK, 11228	Not Applicable	11228	Not Applicable	Not Applicable	BROOKLYN	BROOKLYN
1	2779 BRIGGS AVENUE, BRONX, NEW YORK, 10458	Street/Sidewalk	10458	2779 BRIGGS AVENUE	BRIGGS AVENUE	BRONX	BRONX
2	357 EAST 201 STREET, BRONX, NEW YORK, 10458	Not Applicable	10458	357 EAST 201 STREET	EAST 201 STREET	BRONX	BRONX
3	2711 VALENTINE AVENUE, BRONX, NEW YORK, 10458	Street/Sidewalk	10458	2711 VALENTINE AVENUE	VALENTINE AVENUE	BRONX	BRONX
4	95-15B NORTHERN BOULEVARD, JACKSON HEIGHTS, NE...	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	NORTHERN BOULEVARD	JACKSON HEIGHTS	QUEENS

5-D. Creating the dataset_name and table_id variables, which will be important when using Python to load dataframe as a table to Google BigQuery later

```
# Creating the table_id for this dataset for GBQ
# The hierarchy of how things are named in Google Big Query
dataset_name = 'complaints'
table_id_loc_dim = dataset_name + '.' + 'location_dim_tbl2'
```

6. Creating dataframes for dimensions & doing further transformations in preparation for loading data in Google BigQuery (Complaint Channel Dimension Table)

6-A. Creating a new dataframe with columns representing the attributes of the Complaint Channel Dimension and renaming them to correspond to naming criteria of columns on Google BigQuery

```
1 complaint_channel_dim_tbl = master_columns[['Open Data Channel Type']]
2 print(len(complaint_channel_dim_tbl))
3 complaint_channel_dim_tbl.head()
```

```
108586
```

	Open Data Channel Type
0	ONLINE
1	MOBILE
2	PHONE
3	MOBILE
4	ONLINE

```
1 # We need to rename the column names to have underscores as required for columns in GBQ
2 complaint_channel_dim_tbl = complaint_channel_dim_tbl.rename(columns = {'Open Data Channel Type':'Open_Data_Channel_Type' })
3
4 print(len(complaint_channel_dim_tbl))
5 complaint_channel_dim_tbl.head()
6
7
```

```
108586
```

	Open_Data_Channel_Type
0	ONLINE
1	MOBILE
2	PHONE
3	MOBILE
4	ONLINE

6-B. Creating a copy of the complaint_channel_dim dataframe called complaint_channel_dim2 & drop duplicates from dataframe.
6-C. Creating a surrogate key on the complaint_channel_dim2 dataframe by using the .insert() method and the range() function

- The complaint_channel_dim2 dataframe will be loaded into BigQuery as the Complaint Channel Table, and the Complaint_Channel_ID index from the complaint_channel_dim dataframe (the original dataframe), will be added to the fact table, which will be created later.
- Notice that the length of complaint_channel_dim2 is 4 compared to 108,586 (as in complaint_channel_dim)

```
1 complaint_channel_dim_tbl2= complaint_channel_dim_tbl.copy()
2 complaint_channel_dim_tbl2 = complaint_channel_dim_tbl2.drop_duplicates()
3 print(len(complaint_channel_dim_tbl2))
4 complaint_channel_dim_tbl2.head()
```

4

	Open_Data_Channel_Type
0	ONLINE
1	MOBILE
2	PHONE
6461	UNKNOWN

```
1 # Creating the surrogate key
2 complaint_channel_dim_tbl2.insert(0,'Complaint_Channel_ID', range(1, 1+ len(complaint_channel_dim_tbl2)))
3 print(len((complaint_channel_dim_tbl2)))
4 complaint_channel_dim_tbl2.head()
```

4

	Complaint_Channel_ID	Open_Data_Channel_Type
0	1	ONLINE
1	2	MOBILE
2	3	PHONE
6461	4	UNKNOWN

6-D. Creating the dataset_name and table_id variables, which will be important when using Python to load dataframe as a table to Google BigQuery later

```
# # Creating the table_id for this dataset for GBQ
# The hierarchy of how things are named in Google Big Query
dataset_name = 'complaints'
table_id_odcc = dataset_name + '.' + 'complaint_channel_dim_tbl2'
```

7. Creating dataframes for dimensions & doing further transformations in preparation for loading data in Google BigQuery (Agency Dimension Table)

7-A. Creating a new dataframe with columns representing the attributes of the Agency Dimension and renaming them to correspond to naming criteria of columns on Google BigQuery

```
1 agency_dim_tbl = master_columns[['Agency', 'Agency Name']]
2 print(len(agency_dim_tbl))
3 agency_dim_tbl.head()
```

```
108586
```

	Agency	Agency Name
0	DEP	Department of Environmental Protection
1	NYPD	New York City Police Department
2	NYPD	New York City Police Department
3	NYPD	New York City Police Department
4	NYPD	New York City Police Department

```
1 # We need to rename the column names to have underscores as required for columns in GBQ
2 agency_dim_tbl = agency_dim_tbl.rename(columns = {'Agency Name': 'Agency_Name'})
3 agency_dim_tbl.head()
```

	Agency	Agency_Name
0	DEP	Department of Environmental Protection
1	NYPD	New York City Police Department
2	NYPD	New York City Police Department
3	NYPD	New York City Police Department
4	NYPD	New York City Police Department

7-B. Creating a composite key column called "Agency_Dim_ID" within the agency_dim_tbl dataframe, which will temporarily serve as the surrogate key column for the table that will be loaded into GBQ

```
# Creating the surrogate key
agency_dim_tbl.insert(0,'Agency_Dim_ID', agency_dim_tbl['Agency'] + ' - ' + agency_dim_tbl['Agency_Name'])
print(len((agency_dim_tbl)))
agency_dim_tbl.head()

```

```
108586
```

	Agency_Dim_ID	Agency	Agency_Name
0	DEP - Department of Environmental Protection	DEP	Department of Environmental Protection
1	NYPD - New York City Police Department	NYPD	New York City Police Department
2	NYPD - New York City Police Department	NYPD	New York City Police Department
3	NYPD - New York City Police Department	NYPD	New York City Police Department
4	NYPD - New York City Police Department	NYPD	New York City Police Department

7-C. Creating a copy of the agency_dim_tbl dataframe called agency_dim_tbl2 & drop duplicates from dataframe.
7-D. Creating the dataset_name and table_id variables, which will be important when using Python to load dataframe as a table to Google BigQuery later

- The agency_dim_tbl 2 dataframe will be loaded into BigQuery as the Complaint Channel Table, and the Agency_Dim_ID index from the agency_dim_tbl dataframe (the original dataframe), will be added to the fact table, which will be created later.
- Notice that the length of agency_dim_tbl2 is 4 compared to 108,586 (as in agency_dim_tbl)

```
1 agency_dim_tbl2 = agency_dim_tbl.copy()
2 agency_dim_tbl2 = agency_dim_tbl2.drop_duplicates()
3 print(len(agency_dim_tbl2))
4 agency_dim_tbl2.head()
```

```
4
```

	Agency_Dim_ID	Agency	Agency_Name
0	DEP - Department of Environmental Protection	DEP	Department of Environmental Protection
1	NYPD - New York City Police Department	NYPD	New York City Police Department
747	EDC - Economic Development Corporation	EDC	Economic Development Corporation
44963	DSNY - Department of Sanitation	DSNY	Department of Sanitation

```
1 # # Creating the table_id for this dataset for GBQ
2 # The hierarchy of how things are named in Google Big Query
3 dataset_name = 'complaints'
4 table_id_agency = dataset_name + '.' + 'agency_dim_tbl2'
```

8. Creating dataframes for dimensions & doing further transformations in preparation for loading data in Google BigQuery (Complaint Type Dimension Table)

8-A. Creating a new dataframe with columns representing the attributes of the Complaint Type Dimension and renaming them to correspond to naming criteria of columns on Google BigQuery

```
1 complaint_type_dim_tbl = master_columns[['Complaint Type']]
2 print(len(complaint_type_dim_tbl))
3 complaint_type_dim_tbl.head()
```

```
108586
```

	Complaint Type
0	Noise
1	Noise - Street/Sidewalk
2	Noise - Residential
3	Noise - Vehicle
4	Noise - Commercial

```
1 # We need to rename the column names to have underscores as required for columns in GBQ
2 complaint_type_dim_tbl = complaint_type_dim_tbl.rename(columns = {'Complaint Type': 'Complaint_Type'})
3 print(len(complaint_type_dim_tbl))
4 complaint_type_dim_tbl.head()
```

```
108586
```

	Complaint_Type
0	Noise
1	Noise - Street/Sidewalk
2	Noise - Residential
3	Noise - Vehicle
4	Noise - Commercial

8-B. Creating a copy of the complaint_type_dim_tbl dataframe called complaint_type_dim_tbl2 & drop duplicates from dataframe.
8-C. Creating a surrogate key on the complaint_type_dim_tbl2 dataframe by using the .insert() method and the range() function

- The complaint_type_dim_tbl2 dataframe will be loaded into BigQuery as the Complaint Type Table, and the Complaint_Type_ID index from the complaint_type_dim_tbl dataframe (the original dataframe), will be added to the fact table, which will be created later.
- Notice that the length of complaint_type_dim_tbl2 is 9 compared to 108,586 (as in complaint_type_dim_tbl)

```
complaint_type_dim_tbl2 = complaint_type_dim_tbl.copy()
complaint_type_dim_tbl2 = complaint_type_dim_tbl.drop_duplicates()
print(len(complaint_type_dim_tbl2))
complaint_type_dim_tbl2.head()
```

```
9
```

	Complaint Type
0	Noise
1	Noise - Street/Sidewalk
2	Noise - Residential
3	Noise - Vehicle
4	Noise - Commercial

```
# Creating the surrogate key
complaint_type_dim_tbl2.insert(0,'Complaint_Type_ID', range(1, 1+ len(complaint_type_dim_tbl2)))
print(len((complaint_type_dim_tbl2)))
complaint_type_dim_tbl2.head()
```

```
9
```

	Complaint_Type_ID	Complaint Type
0	1	Noise
1	2	Noise - Street/Sidewalk
2	3	Noise - Residential
3	4	Noise - Vehicle
4	5	Noise - Commercial

8-D. Creating the dataset_name and table_id variables, which will be important when using Python to load dataframe as a table to Google BigQuery later

```
# # Creating the table_id for this dataset for GBQ
# The hierarchy of how things are named in Google Big Query
dataset_name = 'complaints'
table_id_comp_type = dataset_name + '.' + 'complaint_type_dim_tbl2'
```

9. Creating dataframes for dimensions & doing further transformations in preparation for loading data in Google BigQuery (Complaint Status Dimension Table)

9-A. Creating a new dataframe with columns representing the attributes of the Complaint Status Dimension and renaming them to correspond to naming criteria of columns on Google BigQuery

```
1 complaint_status_dim_tbl = master_columns[['Status']]
2 print(len(complaint_status_dim_tbl))
3 complaint_status_dim_tbl.head()
```

```
108586
```

	Status
0	Closed
1	Closed
2	Closed
3	Closed
4	Closed

```
1 # We need to rename the column names to have underscores as required for columns in GBQ
2 complaint_status_dim_tbl = complaint_status_dim_tbl.rename(columns = {'Status': 'Complaint_Status'})
3 print(len(complaint_status_dim_tbl))
4 complaint_status_dim_tbl.head()
5
```

```
108586
```

	Complaint_Status
0	Closed
1	Closed
2	Closed
3	Closed
4	Closed

9-B. Creating a copy of the complaint_status_dim_tbl dataframe called complaint_status_dim_tbl2 & drop duplicates from dataframe.
9-C. Creating a surrogate key on the complaint_status_dim_tbl2 dataframe by using the .insert() method and the range() function

- The complaint_status_dim_tbl2 dataframe will be loaded into BigQuery as the Complaint Status Table, and the Complaint_Status_ID index from the complaint_status_dim_tbl dataframe (the original dataframe), will be added to the fact table, which will be created later.
- Notice that the length of complaint_status_dim_tbl2 is 7 compared to 108,586 (as in complaint_status_dim_tbl)

```
complaint_status_dim_tbl2 = complaint_status_dim_tbl.copy()
complaint_status_dim_tbl2 = complaint_status_dim_tbl.drop_duplicates()
print(len(complaint_status_dim_tbl2))
complaint_status_dim_tbl2.head()
```

```
7
```

	Complaint_Status
0	Closed
4567	Started
12019	Pending
13274	Open
16884	In Progress

```
# Creating the surrogate key
complaint_status_dim_tbl2.insert(0,'Complaint_Status_ID', range(1, 1+ len(complaint_status_dim_tbl2)))
print(len((complaint_status_dim_tbl2)))
complaint_status_dim_tbl2.head()
```

```
7
```

	Complaint_Status_ID	Complaint_Status
0	1	Closed
4567	2	Started
12019	3	Pending
13274	4	Open
16884	5	In Progress

9-D. Creating the dataset_name and table_id variables, which will be important when using Python to load dataframe as a table to Google BigQuery later

```
# # Creating the table_id for this dataset for GBQ
# The hierarchy of how things are named in Google Big Query
dataset_name = 'complaints'
table_id_comp_status = dataset_name + '.' + 'complaint_status_dim_tbl2'
```

10. Creating dataframes for dimensions & doing further transformations in preparation for loading data in Google BigQuery (Complaint Status Dimension Table)

10-A. Creating a new dataframe with columns representing the attributes of the Date Dimension and renaming them to correspond to naming criteria of columns on Google BigQuery

```
1 date_dim_tbl = master_columns[['Created Date']]
2
3 # We need to rename the column names to have underscores as required for columns in GBQ
4 date_dim_tbl = date_dim_tbl.rename(columns = {'Created Date': 'Created_Date'})
5
6 print(len(date_dim_tbl))
7 date_dim_tbl.head()
```

```
108586
```

	Created_Date
0	06/02/2020 06:09:00 PM
1	07/06/2019 01:03:33 AM
2	07/06/2019 07:49:10 PM
3	07/09/2019 12:20:14 AM
4	07/08/2019 04:36:32 AM

10-B. Creating new dataframes in order to slice data from columns, which will need to be inserted into the date_dim_tbl dataframe

```
# splitting the contents of a column using the .split method: https://stackoverflow.com/questions/37600711/pandas-split-column-into-multiple-columns-by-comma
date_slice = master_columns['Created Date'].str.split(' ', expand = True)
date_slice.head()
```

	0	1	2
0	06/02/2020	06:09:00	PM
1	07/06/2019	01:03:33	AM
2	07/06/2019	07:49:10	PM
3	07/09/2019	12:20:14	AM
4	07/08/2019	04:36:32	AM

```
# splitting the contents of a column using the .split method: https://stackoverflow.com/questions/37600711/pandas-split-column-into-multiple-columns-by-comma
date_slice2 = date_slice[0].str.split('/', expand = True)

# How to convert the values of an dataframe column to int:
# https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/#:~:text=to%20int%20(Integer)-,Use%20pandas%20DataFrame.,int64%20%2C%20numpy.

day_nums = date_slice2[1].astype('int')
date_slice2


```

	0	1	2
0	06	02	2020
1	07	06	2019
2	07	06	2019
3	07	09	2019
4	07	08	2019

10-B (cont'd). Creating new dataframes in order to slice data from columns, which will need to be inserted into the date_dim_tbl dataframe

```
# splitting the contents of a column using the .split method: https://stackoverflow.com/questions/37600711/pandas-split-column-into-multiple-columns-by-comma
date_slice2 = date_slice[0].str.split('/', expand = True)

# How to convert the values of an dataframe column to int:
# https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/#:~:text=to%20int%20(Integer)-,Use%20pandas%20DataFrame.,int64%20%2C%20numpy.

day_nums = date_slice2[1].astype('int')
date_slice2.head()
```

	0	1	2
0	06	02	2020
1	07	06	2019
2	07	06	2019
3	07	09	2019
4	07	08	2019

```
date_slice[0] = pd.to_datetime(date_slice[0])
date_slice[1]= pd.to_datetime(date_slice[1]) # this is really the format of the data
date_slice[0] = date_slice[0].copy()
date_slice[0] = pd.to_datetime(date_slice[0])

#date_slice = date_slice.drop(2, axis=1)

date_slice.head()
```

	0	1	2
0	2020-06-02	2022-12-18 06:09:00	PM
1	2019-07-06	2022-12-18 01:03:33	AM
2	2019-07-06	2022-12-18 07:49:10	PM
3	2019-07-09	2022-12-18 12:20:14	AM

```
# Source: https://www.geeksforgeeks.org/get-month-and-year-from-date-in-pandas-python/
year_num = date_slice[0].dt.year
year_int = year_num.astype('int')
year_int

```

```
0         2020
1         2019
2         2019
3         2019
4         2019
          ...
108581    2022
108582    2022
108583    2022
108584    2022
108585    2022
Name: 0, Length: 108586, dtype: int64
```

10-B (cont'd). Creating new dataframes in order to slice data from columns, which will need to be inserted into the date_dim_tbl dataframe

```
date_dim_tbl.insert(1, 'Date', date_slice[0])

# How to get the name of a month give a date (in datetime format):
# Source: https://pynative.com/python-get-month-name-from-number/#:~:text=Use%20the%20dt.,the%20month%20name%20in%20pandas.
date_dim_tbl.insert(2, 'Month', date_slice[0].dt.month_name())

# How to get the name of a day give a date (in datetime format):
# Source: https://www.geeksforgeeks.org/get-day-from-date-in-pandas-python/
date_dim_tbl.insert(3, 'Day_Of_Week', date_slice[0].dt.day_name() ) # date_slice[0].dt.dayofweek

# The following commented code was used to create the variable day_nums
# in the cell above:
# date_slice2 = date_slice[0].str.split('/', expand = True)
# How to convert the values of an dataframe column to int:
# https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/#:~:text=to%20int%20(Integer)-,Use%20pandas%20DataFrame.,int64%20%2C%20numpy.
# day_nums = date_slice2[1].astype('int')
date_dim_tbl.insert(4, 'Day_Number', day_nums)

# How to get the name of a year give a date (in datetime format):
# The following commented code was used to create the variable day_nums
# in the cell above:
# Source: https://www.geeksforgeeks.org/get-month-and-year-from-date-in-pandas-python/
# year_num = date_slice[0].dt.year
# year_int = year_num.astype('int')
date_dim_tbl.insert(5, 'Year', year_int)

date_dim_tbl.insert(6, 'Time', date_slice[1])

#date_dim_tbl = date_slice.drop(0, axis=1) # Causes everything to get deleted
date_dim_tbl.head()
```

	Created_Date	Date	Month	Day_Of_Week	Day_Number	Year	Time
0	06/02/2020 06:09:00 PM	2020-06-02	June	Tuesday	2	2020	2022-12-18 06:09:00
1	07/06/2019 01:03:33 AM	2019-07-06	July	Saturday	6	2019	2022-12-18 01:03:33
2	07/06/2019 07:49:10 PM	2019-07-06	July	Saturday	6	2019	2022-12-18 07:49:10
3	07/09/2019 12:20:14 AM	2019-07-09	July	Tuesday	9	2019	2022-12-18 12:20:14
4	07/08/2019 04:36:32 AM	2019-07-08	July	Monday	8	2019	2022-12-18 04:36:32

10-C. Creating a copy of the date_dim_tbl dataframe called date_dim_tbl2 & drop duplicates from dataframe.

10-D. Create a composite key on the date_dim_tbl2 dataframe that will act as the surrogate key for the Date Dimension Table

* Because, date_dim_tbl['Created_Date'] already contains date and time, we can use it as the composite key

- The date_dim_tbl2 dataframe will be loaded into BigQuery as the Date Dimension Table, and the Date_Dim_ID index from the date_dim_tbl dataframe (the original dataframe), will be added to the fact table, which will be created later.
- Notice that the length of date_dim_tbl2 is 108,408 compared to 108,586 (as in date_dim_tbl)

```
# Creating the composite key
# date_dim_tbl['Date_And_Time'] = date_dim_tbl['Date'] + ' - ' + date_dim_tbl['Time']
# Don't need this variable as "Created_Date" has date and time
# date_dim_tbl = date_dim_tbl.drop_duplicates()

# Created date already gives both date and time, which are critical to distinguishing
# Unique rows
date_dim_tbl = date_dim_tbl.rename(columns = {'Created_Date': 'Date_Dim_ID'})

print(len(date_dim_tbl))
date_dim_tbl.head()
```

```
108586
```

	Date_Dim_ID	Date	Month	Day_Of_Week	Day_Number	Year	Time
0	06/02/2020 06:09:00 PM	2020-06-02	June	Tuesday	2	2020	2022-12-18 06:09:00
1	07/06/2019 01:03:33 AM	2019-07-06	July	Saturday	6	2019	2022-12-18 01:03:33
2	07/06/2019 07:49:10 PM	2019-07-06	July	Saturday	6	2019	2022-12-18 07:49:10
3	07/09/2019 12:20:14 AM	2019-07-09	July	Tuesday	9	2019	2022-12-18 12:20:14
4	07/08/2019 04:36:32 AM	2019-07-08	July	Monday	8	2019	2022-12-18 04:36:32

```
date_dim_tbl2 = date_dim_tbl.copy()
date_dim_tbl2 = date_dim_tbl2.drop_duplicates()
print(len(date_dim_tbl2))
date_dim_tbl2.head()
```

```
108408
```

	Date_Dim_ID	Date	Month	Day_Of_Week	Day_Number	Year	Time
0	06/02/2020 06:09:00 PM	2020-06-02	June	Tuesday	2	2020	2022-12-18 06:09:00
1	07/06/2019 01:03:33 AM	2019-07-06	July	Saturday	6	2019	2022-12-18 01:03:33
2	07/06/2019 07:49:10 PM	2019-07-06	July	Saturday	6	2019	2022-12-18 07:49:10
3	07/09/2019 12:20:14 AM	2019-07-09	July	Tuesday	9	2019	2022-12-18 12:20:14
4	07/08/2019 04:36:32 AM	2019-07-08	July	Monday	8	2019	2022-12-18 04:36:32

10-E. Creating the dataset_name and table_id variables, which will be important when using Python to load dataframe as a table to Google BigQuery later

```
# # Creating the table_id for this dataset for GBQ
# The hierarchy of how things are named in Google Big Query
dataset_name = 'complaints'
table_id_date = dataset_name + '.' + 'date_dim_tbl2'
```

11. Creating dataframes for dimensions & doing further transformations in preparation for loading data in Google BigQuery (Complaint Fact Table)

11-A. Creating a new dataframe with columns representing the attributes of the Complaint Fact Table and first adding the agency_dim_tbl['Agency_Dim_ID'] from the dimension table (where duplicates have NOT been dropped) created earlier

11-B. Continue to add the columns from previous dataframes created earlier (where duplicates have NOT been dropped)

```
complaint_fact_tbl = pd.DataFrame(agency_dim_tbl['Agency_Dim_ID'], columns = ['Agency_Dim_ID'])
```

```
complaint_fact_tbl.insert(0,'Unique_Key', range(1, 1+ len(complaint_fact_tbl)))
complaint_fact_tbl.insert(2,'Complaint_Type', complaint_type_dim_tbl['Complaint_Type'])
complaint_fact_tbl.insert(3,'Complaint_Channel', complaint_channel_dim_tbl['Open_Data_Channel_Type'])
complaint_fact_tbl.insert(4,'Complaint_Status', complaint_status_dim_tbl['Complaint_Status'])
complaint_fact_tbl.insert(5,'Date', date_dim_tbl['Date'])
complaint_fact_tbl.insert(6,'Location_Dim_ID', location_dim_tbl['Location_Dim_ID'])
```

```
print(len(complaint_fact_tbl))
complaint_fact_tbl.head()
```

```
108586
```

	Unique_Key	Agency_Dim_ID	Complaint_Type	Complaint_Channel	Complaint_Status	Date	Location_Dim_ID
0	1	DEP - Department of Environmental Protection	Noise	ONLINE	Closed	2020-06-02	Not Applicable, BROOKLYN, NEW YORK, 11228
1	2	NYPD - New York City Police Department	Noise - Street/Sidewalk	MOBILE	Closed	2019-07-06	2779 BRIGGS AVENUE, BRONX, NEW YORK, 10458
2	3	NYPD - New York City Police Department	Noise - Residential	PHONE	Closed	2019-07-06	357 EAST 201 STREET, BRONX, NEW YORK, 10458
3	4	NYPD - New York City Police Department	Noise - Vehicle	MOBILE	Closed	2019-07-09	2711 VALENTINE AVENUE, BRONX, NEW YORK, 10458
4	5	NYPD - New York City Police Department	Noise - Commercial	ONLINE	Closed	2019-07-08	95-15B NORTHERN BOULEVARD, JACKSON HEIGHTS, NE...

11-C. Creating the dataset_name and table_id variables, which will be important when using Python to load dataframe as a table to Google BigQuery later

```
# # Creating the table_id for this dataset for GBQ
# The hierarchy of how things are named in Google Big Query
dataset_name = 'complaints'
table_id_311_fact = dataset_name + '.' + 'complaint_fact_tbl'
```

12. Connecting to Google BigQuery via Python using the os and google.cloud libraries

```
# Feeding in Json file generated from GBQ
```

```
! gdown --id 1BFPm2cptd9EWD_187KSlYQrMUVy9sNzf
```

```
/usr/local/lib/python3.8/dist-packages/gdown/cli.py:127: FutureWarning: Option `--id` was
  warnings.warn(
Downloading...
From: https://drive.google.com/uc?id=1BFPm2cptd9EWD_187KSlYQrMUVy9sNzf
To: /content/cis4400proj-370416-74e9d8391c22.json
100% 2.32k/2.32k [00:00<00:00, 1.43MB/s]
```

```
# Reading/Writing into Google Cloud project
# https://data.cityofnewyork.us/resource/erm2-nwe9.json

# Setting up the environment variable
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'cis4400proj-370416-74e9d8391c22.json'
client = bigquery.Client()
```

13. Loading Data for the Location Dimension into Google BigQuery

```
# Client Object: How you read and write data into Google Big Query
client = bigquery.Client()
```

```
job_config = bigquery.LoadJobConfig(

  # Specify a partial schema. All columns are always written to the table
  # The schema is used to assist in data type definitions

  schema = [

      bigquery.SchemaField("Location_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Location_Type", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Incident_Zip", bigquery.enums.SqlTypeNames.INTEGER),
      bigquery.SchemaField("Incident_Address", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Street_Name", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("City", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Borough", bigquery.enums.SqlTypeNames.STRING)
  ],

  write_disposition = "WRITE_TRUNCATE",

)
```

```
job = client.load_table_from_dataframe(
    location_dim_tbl2, table_id_loc_dim, job_config= job_config
)
job.result()
```

```
LoadJob<project=cis4400proj-370416, location=US, id=325ba92a-0b40-4e0e-a13d-c9a394b3f07a>
```

location_dim_tbl2 QUERY S

SCHEMA DETAILS PREVIEW

Filter Enter property name or value

Field name	Type	Mode
Location_Dim_ID	STRING	NULLABLE
Location_Type	STRING	NULLABLE
Incident_Zip	INTEGER	NULLABLE
Incident_Address	STRING	NULLABLE
Street_Name	STRING	NULLABLE
City	STRING	NULLABLE
Borough	STRING	NULLABLE

EDIT SCHEMA VIEW ROW ACCESS POLICIES

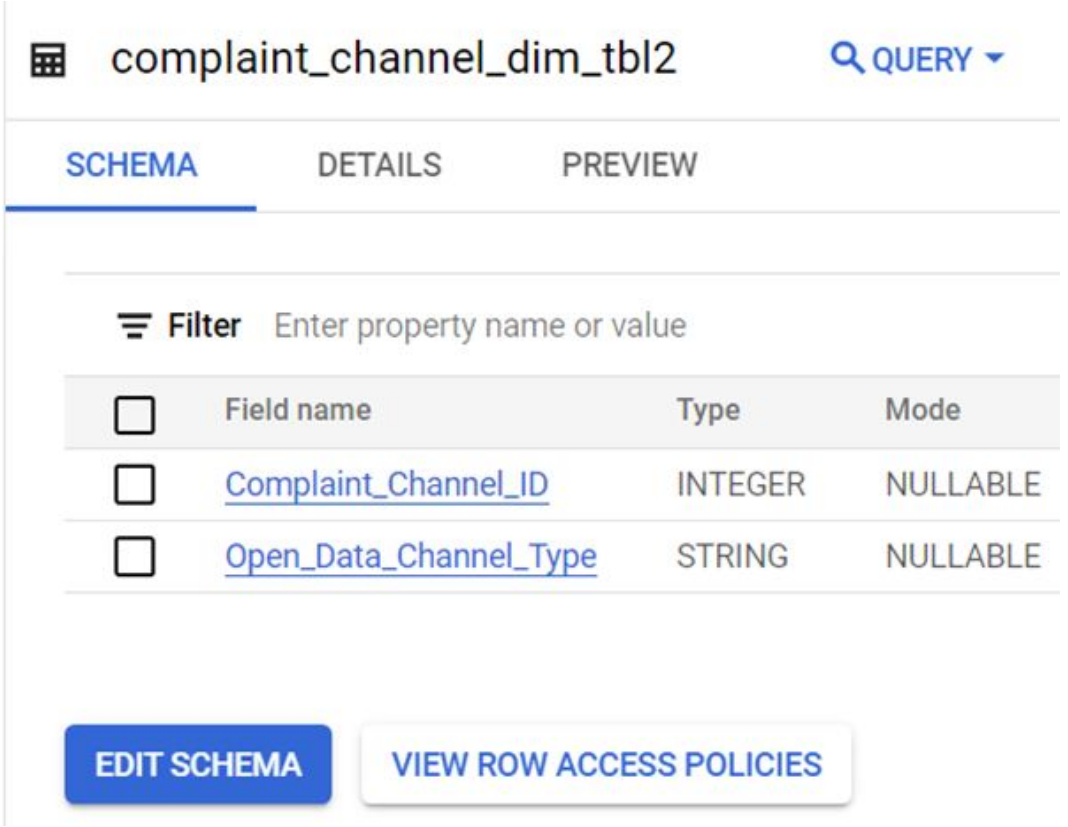
14. Loading Data for the Complaint Channel Dimension into Google BigQuery

```
# Client Object: How you read and write data into Google Big Query
client = bigquery.Client()
```

```
job_config = bigquery.LoadJobConfig(

  # Specify a partial schema. All columns are always written to the table
  # The schema is used to assist in data type definitions

  schema = [

      bigquery.SchemaField("Complaint_Channel_ID", bigquery.enums.SqlTypeNames.INTEGER),
      bigquery.SchemaField("Open_Data_Channel_Type", bigquery.enums.SqlTypeNames.STRING)
  ],

  write_disposition = "WRITE_TRUNCATE",

)
```

```
job = client.load_table_from_dataframe(
    complaint_channel_dim_tbl2, table_id_odcc, job_config= job_config
)
job.result()
```

```
LoadJob<project=cis4400proj-370416, location=US, id=07679c16-b44b-43be-b5d3-afda2f77f575>
```



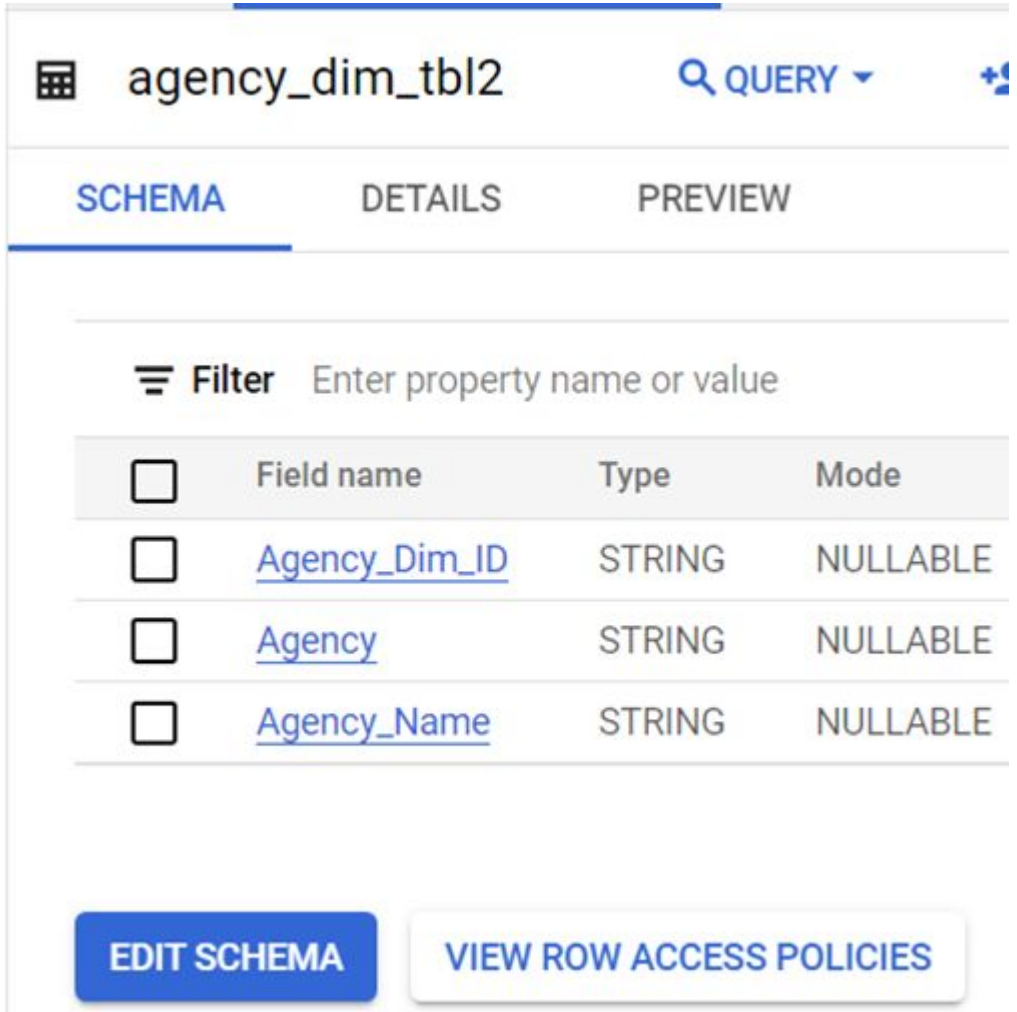
15. Loading Data for the Agency Dimension into Google BigQuery

```
# Client Object: How you read and write data into Google Big Query
client = bigquery.Client()
```

```
job_config = bigquery.LoadJobConfig(

  # Specify a partial schema. All columns are always written to the table
  # The schema is used to assist in data type definitions

  schema = [

      bigquery.SchemaField("Agency_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Agency", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Agency_Name", bigquery.enums.SqlTypeNames.STRING),
  ],

  write_disposition = "WRITE_TRUNCATE",

)
```

```
job = client.load_table_from_dataframe(
    agency_dim_tbl2, table_id_agency, job_config= job_config
)
job.result()
```

```
LoadJob<project=cis4400proj-370416, location=US, id=3fa61286-52e6-4c70-b332-da3831524812>
```



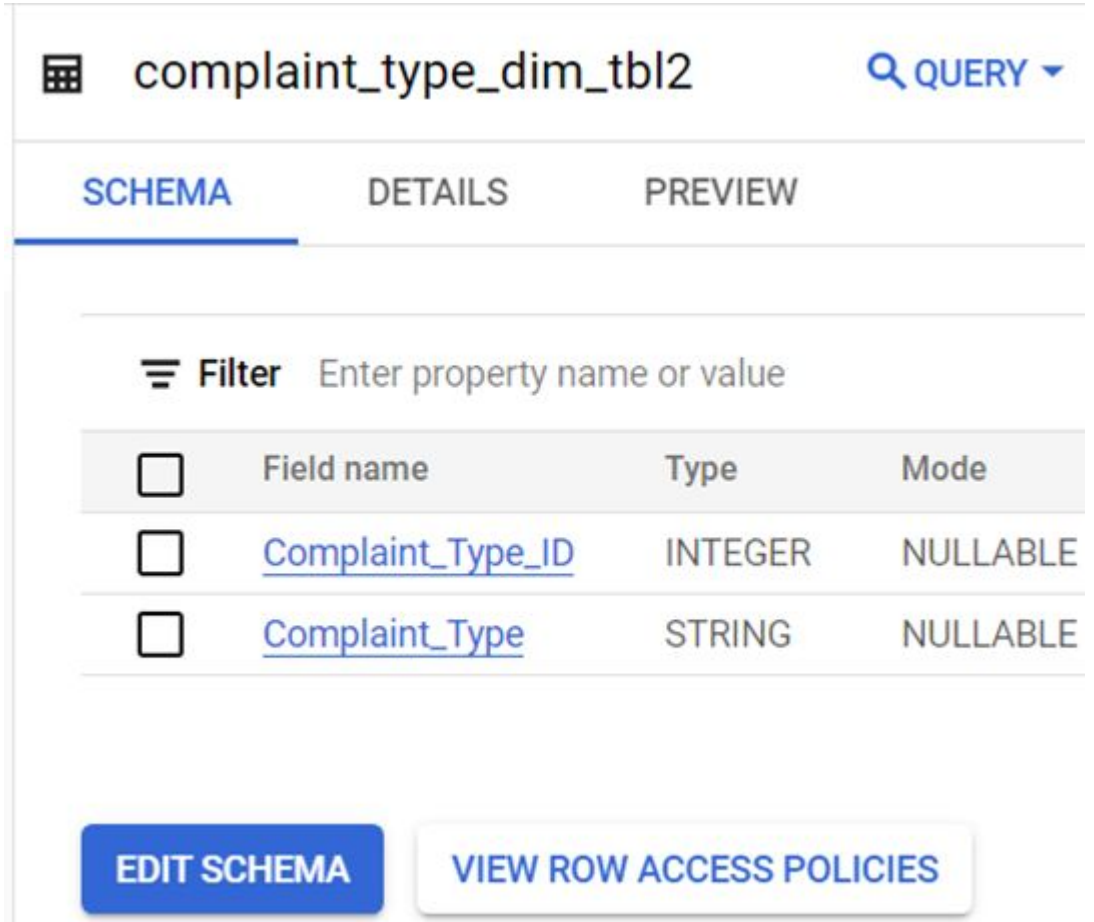
16. Loading Data for the Complaint Type Dimension into Google BigQuery

```
# Client Object: How you read and write data into Google Big Query
client = bigquery.Client()
```

```
job_config = bigquery.LoadJobConfig(

  # Specify a partial schema. All columns are always written to the table
  # The schema is used to assist in data type definitions

  schema = [

      bigquery.SchemaField("Complaint_Type_ID", bigquery.enums.SqlTypeNames.INTEGER),
      bigquery.SchemaField("Complaint_Type", bigquery.enums.SqlTypeNames.STRING)
  ],

  write_disposition = "WRITE_TRUNCATE",

)
```

```
job = client.load_table_from_dataframe(
    complaint_type_dim_tbl2, table_id_comp_type, job_config= job_config
)
job.result()
```

```
LoadJob<project=cis4400proj-370416, location=US, id=22809313-8487-4a9f-a3a9-4b96007dfc7f>
```



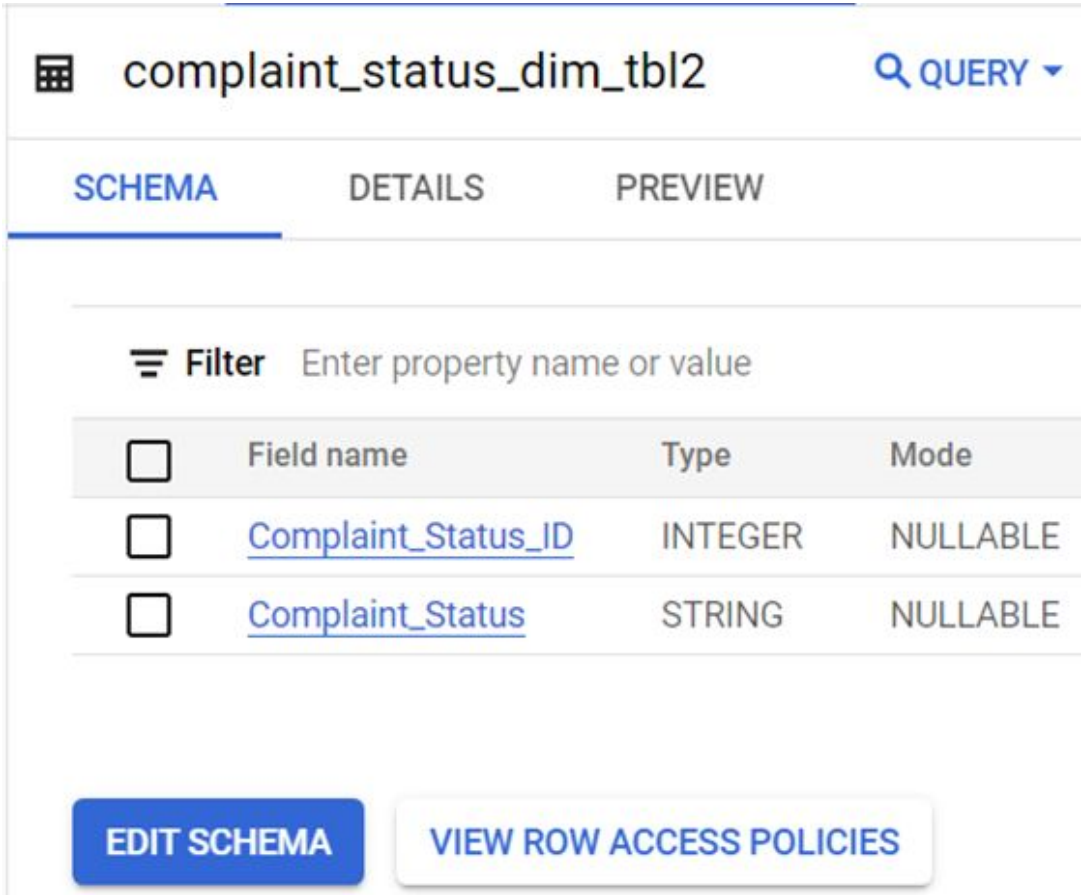
17. Loading Data for the Complaint Status Dimension into Google BigQuery

```
# Client Object: How you read and write data into Google Big Query
client = bigquery.Client()
```

```
job_config = bigquery.LoadJobConfig(

  # Specify a partial schema. All columns are always written to the table
  # The schema is used to assist in data type definitions

  schema = [

      bigquery.SchemaField("Complaint_Status_ID", bigquery.enums.SqlTypeNames.INTEGER),
      bigquery.SchemaField("Complaint_Status", bigquery.enums.SqlTypeNames.STRING)
  ],

  write_disposition = "WRITE_TRUNCATE",

)
```

```
job = client.load_table_from_dataframe(
    complaint_status_dim_tbl2, table_id_comp_status, job_config= job_config
)
job.result()
```

```
LoadJob<project=cis4400proj-370416, location=US, id=759166de-c103-44eb-b6b7-f286cabca360>
```



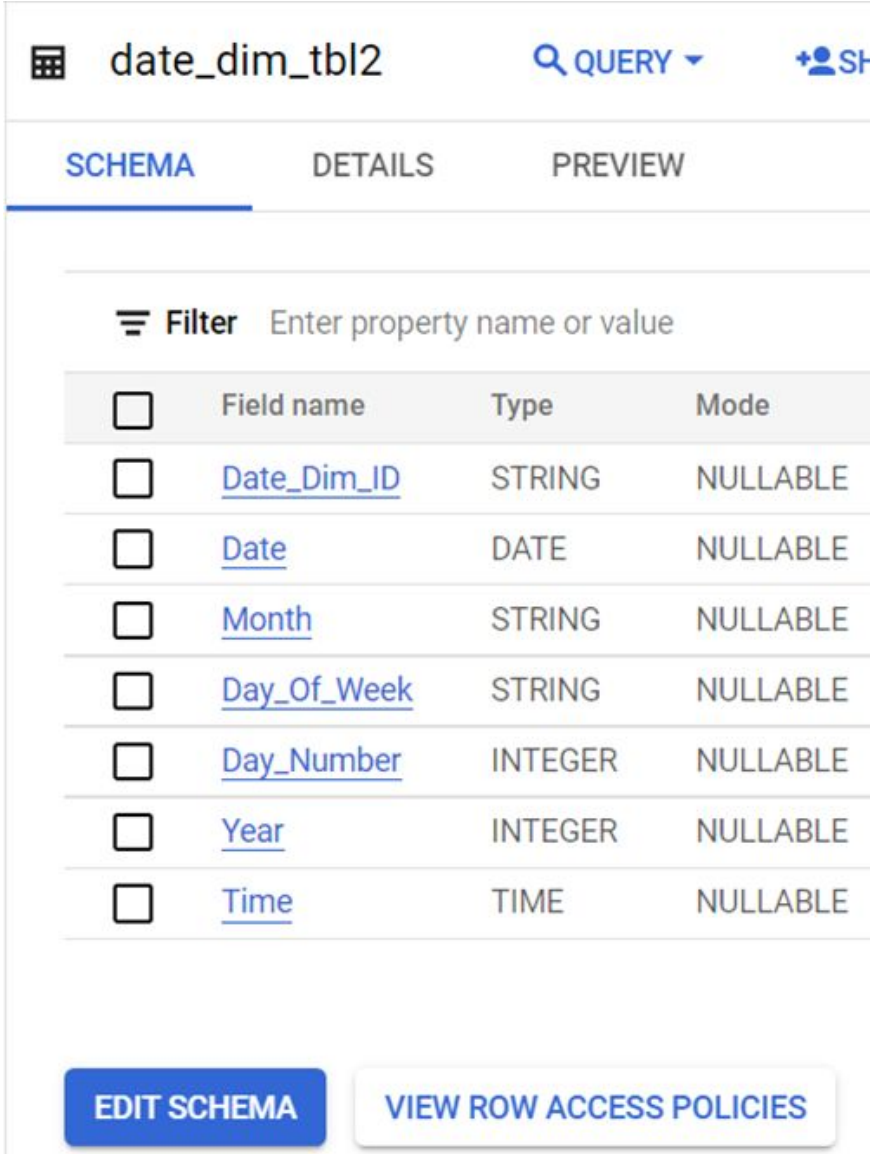
18. Loading Data for the Date Dimension into Google BigQuery

```
# Client Object: How you read and write data into Google Big Query
client = bigquery.Client()
```

```
job_config = bigquery.LoadJobConfig(

  # Specify a partial schema. All columns are always written to the table
  # The schema is used to assist in data type definitions

  # Converting all the date values to string works

  schema = [

      bigquery.SchemaField("Date_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Date", bigquery.enums.SqlTypeNames.DATE),
      bigquery.SchemaField("Month", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Year", bigquery.enums.SqlTypeNames.INTEGER),
      bigquery.SchemaField("Day_Of_Week", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Day_Number", bigquery.enums.SqlTypeNames.INTEGER),
      bigquery.SchemaField("Time", bigquery.enums.SqlTypeNames.TIME)
  ],

  write_disposition = "WRITE_TRUNCATE",

)
```

```
job = client.load_table_from_dataframe(
    date_dim_tbl2, table_id_date, job_config= job_config
)
job.result()
```

```
LoadJob<project=cis4400proj-370416, location=US, id=ab53d63f-add7-45ae-b04f-f303abc5e0c2>
```



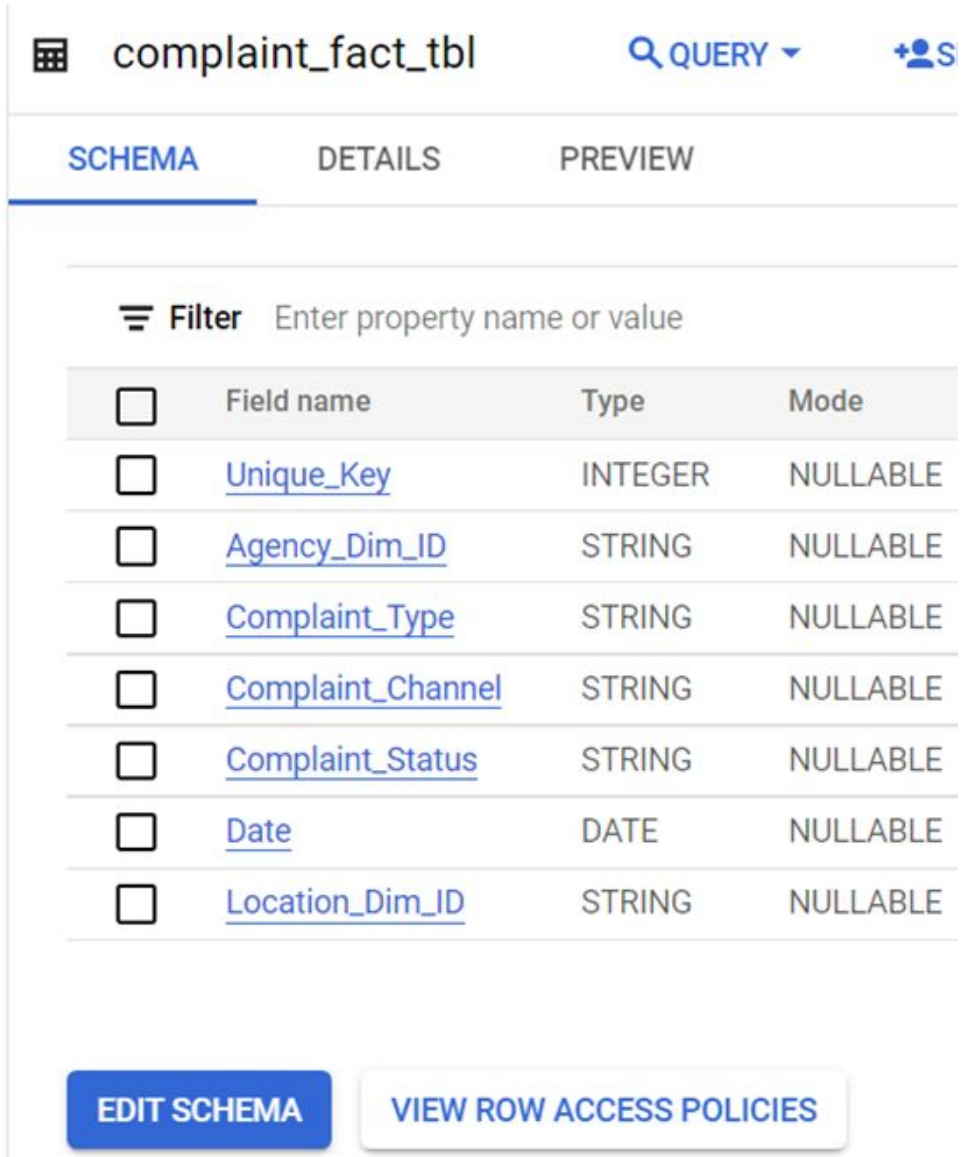
19. Loading Data for the Fact Table into Google BigQuery

```
# Client Object: How you read and write data into Google Big Query
client = bigquery.Client()
```

```
job_config = bigquery.LoadJobConfig(

  # Specify a partial schema. All columns are always written to the table
  # The schema is used to assist in data type definitions

  # Converting all the date values to string works

  schema = [

      bigquery.SchemaField("Unique_Key", bigquery.enums.SqlTypeNames.INTEGER),
      bigquery.SchemaField("Agency_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Complaint_Type", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Complaint_Channel", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Complaint_Status", bigquery.enums.SqlTypeNames.STRING),
      bigquery.SchemaField("Date", bigquery.enums.SqlTypeNames.DATE),
      bigquery.SchemaField("Location_Dim_ID", bigquery.enums.SqlTypeNames.STRING)
  ],

  write_disposition = "WRITE_TRUNCATE",

)
```

```
job = client.load_table_from_dataframe(
    complaint_fact_tbl, table_id_311_fact, job_config= job_config
)
job.result()
```

```
LoadJob<project=cis4400proj-370416, location=US, id=de2ba344-2650-47fd-ba37-9bd9cd76032b>
```



Weather Dataset ETL

In [25]:
```
import pandas as pd
from google.cloud import bigquery
import os
data = pd.read_csv(r'C:\Users\apoll\Downloads\weather_data_5_boroughs_daily (2).csv')
```

In [26]:
```
data.head()
```

Out[26]:

	Station	latitude	longitude	Borough	City	State	ZipCode	WDate	Temperature_Max	Temperature_Avg	...	Humidity_M	Wdspeed_Max	Wdspee
0	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	1/1/2016	41.2	38.1	...	45	28.4	
1	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	1/2/2016	39.4	35.2	...	42	25.3	
2	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	1/3/2016	44.7	38.6	...	36	28.4	
3	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	1/4/2016	35.6	25.9	...	31	36.5	
4	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	1/5/2016	28.8	18.5	...	21	27.5	

5 rows × 26 columns

In [4]:

```
df = data.iloc[: , :-3]
df.drop(columns=['latitude','longitude'])
```

Out[4]:

	Station	Borough	City	State	ZipCode	WDate	Temperature_Max	Temperature_Avg	Temperature_M	Dewpot_Max	...	Dewpot_M
0	KNYBRONX14	Bronx	Botanical Garden	NY	10458	1/1/2016	41.2	38.1	33.9	26.9	...	16.9
1	KNYBRONX14	Bronx	Botanical Garden	NY	10458	1/2/2016	39.4	35.2	32.4	19.3	...	14.0
2	KNYBRONX14	Bronx	Botanical Garden	NY	10458	1/3/2016	44.7	38.6	34.5	23.5	...	19.5
3	KNYBRONX14	Bronx	Botanical Garden	NY	10458	1/4/2016	35.6	25.9	12.1	23.8	...	-0.8
4	KNYBRONX14	Bronx	Botanical Garden	NY	10458	1/5/2016	28.8	18.5	10.0	6.5	...	-6.9
...	...	...	...	...	...	...	...	...	...	...	...	...
10521	KNYNEWYO103	Staten Island	Richmondtown	NY	10308	10/3/2021	79.7	67.1	53.8	68.0	...	52.3

In [5]:
```
df.isnull().values.any()
```

Out[5]: False

In [6]:
```
df["WeatherParamsID"] = df['WDate'].astype(str) +"-"+ df["Temperature_Avg"].astype(str) + "-" + df["ZipCode"].astype(str)
df['Year'] = pd.DatetimeIndex(df['WDate']).year
df['Month'] = pd.DatetimeIndex(df['WDate']).month
df['Day'] = pd.DatetimeIndex(df['WDate']).day
df["DateID"] = df['WDate'].astype(str) +"-"+ df["ZipCode"].astype(str)
df['WDate'] = pd.to_datetime(df['WDate']).dt.date
#df["WDate"] = pd.to_datetime(df['WDate']).dt.date

df.head()
```

Out[6]:

	Station	latitude	longitude	Borough	City	State	ZipCode	WDate	Temperature_Max	Temperature_Avg	...	Wdspeed_Avg	Wdspeed_M	Pressure
0	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	2016-01-01	41.2	38.1	...	11.3	1.3	
1	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	2016-01-02	39.4	35.2	...	10.1	0.0	
2	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	2016-01-03	44.7	38.6	...	10.8	1.6	
3	KNYBRONX14	40.8616	-73.8809	Bronx	Botanical Garden	NY	10458	2016-01-04	35.6	25.9	...	10.2	1.1	

In [7]:
```
df_address = df[["Borough","City","State","ZipCode","Station"]]
```

In [8]:
```
df_fact = df[['WeatherParamsID','ZipCode',"WDate","Temperature_Avg",'Dewpot_Avg','Humidity_Avg','Wdspeed_Avg','Precipitation
```

In [13]:
```
df_address = df_address.drop_duplicates()
df_address
```

Out[13]:

	Borough	City	State	ZipCode
0	Bronx	Botanical Garden	NY	10458
2107	Brooklyn	Dyker Heights	NY	11228
4207	Manhattan	New York	NY	10018
6313	Queens	Jackson Heights	NY	11372
8419	Staten Island	Richmondtown	NY	10306
10247	Staten Island	Richmondtown	NY	10308

In [7]:
```
df_params = df[['WeatherParamsID','Temperature_Max','Temperature_M','Dewpot_Max','Dewpot_M','Humidity_Max','Humidity_M','Wds
               'Wdspeed_M','Pressure_Max','Pressure_M']]
```

In [8]:
```
df_date = df[["DateID","WDate","Year","Month","Day"]]

```

In [9]:
```
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = "cis4400proj-370416-74e9d8391c22.json"
```

In [15]:
```
client = bigquery.Client()
```

In [16]:
```
dataset_name = 'weather'
table_id = dataset_name + '.address_dim_tbl'
table_id2 = dataset_name + '.facts_dim_tbl'
table_id3 = dataset_name + '.params_dim_tbl'
table_id4 = dataset_name + '.date_dim_tbl'
```

In [27]:
```
job_config = bigquery.LoadJobConfig(
    schema = [bigquery.SchemaField("Borough",bigquery.enums.SqlTypeNames.STRING)
              ], write_disposition = "WRITE_TRUNCATE")
```

In [18]:
```
job_config2 = bigquery.LoadJobConfig(
    schema = [bigquery.SchemaField("Station",bigquery.enums.SqlTypeNames.STRING)
              ], write_disposition = "WRITE_TRUNCATE")
```

In [19]:
```
job_config3 = bigquery.LoadJobConfig(
    schema = [bigquery.SchemaField("WeatherParamsID",bigquery.enums.SqlTypeNames.STRING)
              ], write_disposition = "WRITE_TRUNCATE")
```

In [20]:
```
job_config4 = bigquery.LoadJobConfig(
    schema = [bigquery.SchemaField("DateID",bigquery.enums.SqlTypeNames.STRING)
              ], write_disposition = "WRITE_TRUNCATE")
```

In [28]:
```
job = client.load_table_from_dataframe(
df_address, table_id, job_config=job_config)

job.result()
```

Out[28]:
```
LoadJob<project=cis4400proj-370416, location=US, id=970792cb-9bec-4e5c-b075-22ddc4ddc299>
```

In [22]:
```
job2 = client.load_table_from_dataframe(
df_fact, table_id2, job_config=job_config2)

job.result()
```

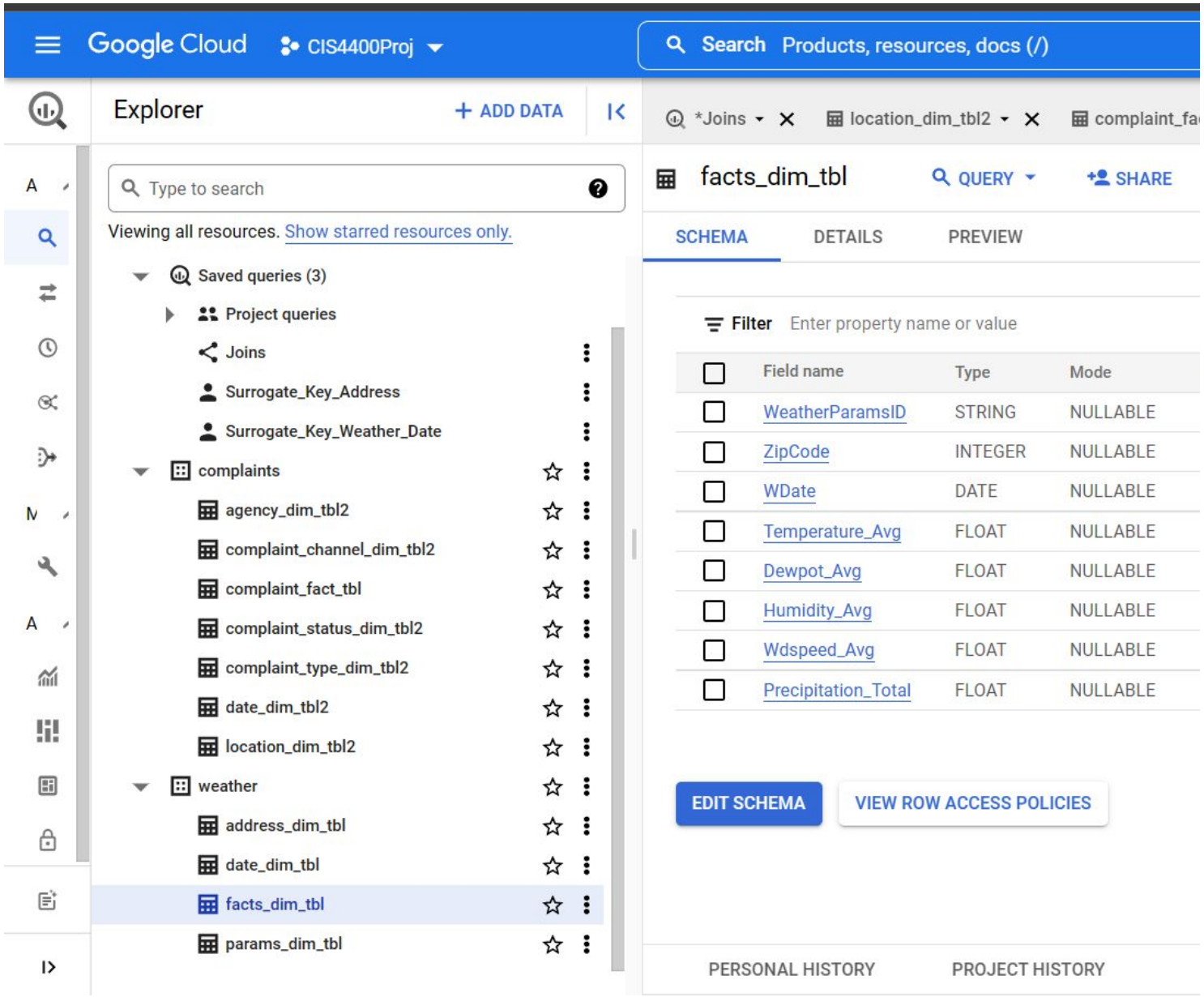
Out[22]:
```
LoadJob<project=cis4400proj-370416, location=US, id=2e220974-491b-45f2-bdad-d04f3225e954>
```

In [23]:
```
job3 = client.load_table_from_dataframe(
df_params, table_id3, job_config=job_config3)

job.result()
```

Out[23]:
```
LoadJob<project=cis4400proj-370416, location=US, id=2e220974-491b-45f2-bdad-d04f3225e954>
```

In [24]:
```
job3 = client.load_table_from_dataframe(
df_date, table_id4, job_config=job_config4)

job.result()
```

Final Dimension Schema



Python was used for the ETL process and to load the data set into Google BigQuery.

There are two data sets: Complaints and Weather.

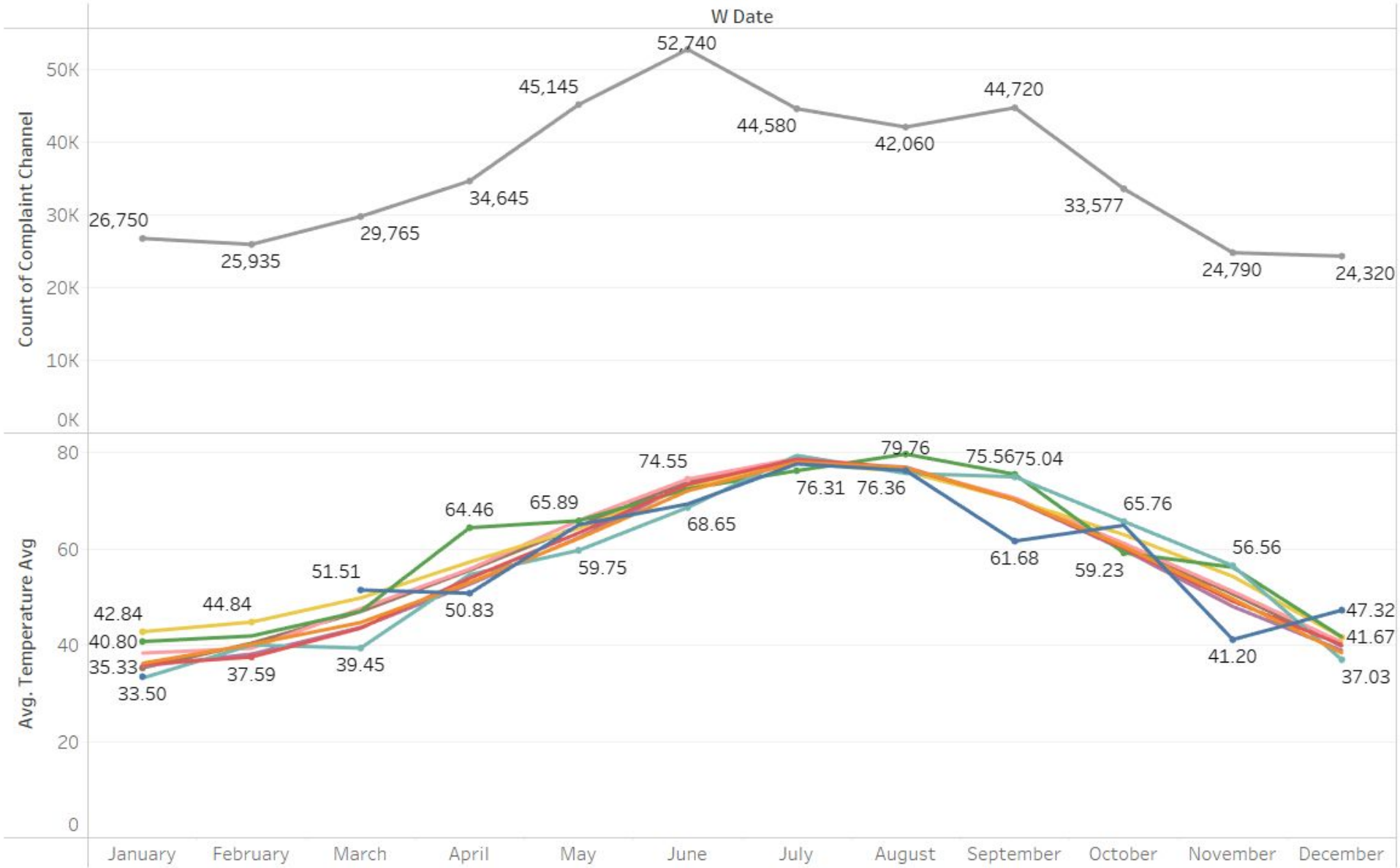
The 2 data sets were joined using left join on the Weather's Fact table to determine the correlation between complaints made throughout the year and different weather conditions.

Query to Join Complaints ETL and Weather ETL

```
SELECT Agency_Dim_ID,Complaint_Type,Complaint_Status,ZipCode,
WDate,Temperature_Avg,Humidity_Avg,Wdspeed_Avg,Precipitation_Total,
Borough,City,State,Station,Complaint_Channel
FROM `weather.facts_dim_tbl`
LEFT JOIN `weather.address_dim_tbl`
USING (ZipCode)
LEFT JOIN `complaints.complaint_fact_tbl`
ON Date = WDate
LEFT JOIN `complaints.complaint_channel_dim_tbl2`
ON Complaint_Channel = Open_Data_Channel_Type
LEFT JOIN `complaints.complaint_status_dim_tbl2`
USING (Complaint_Status)
LEFT JOIN `complaints.complaint_type_dim_tbl2`
USING (Complaint_Type)
LEFT JOIN `complaints.agency_dim_tbl2`
USING (Agency_Dim_ID)
```

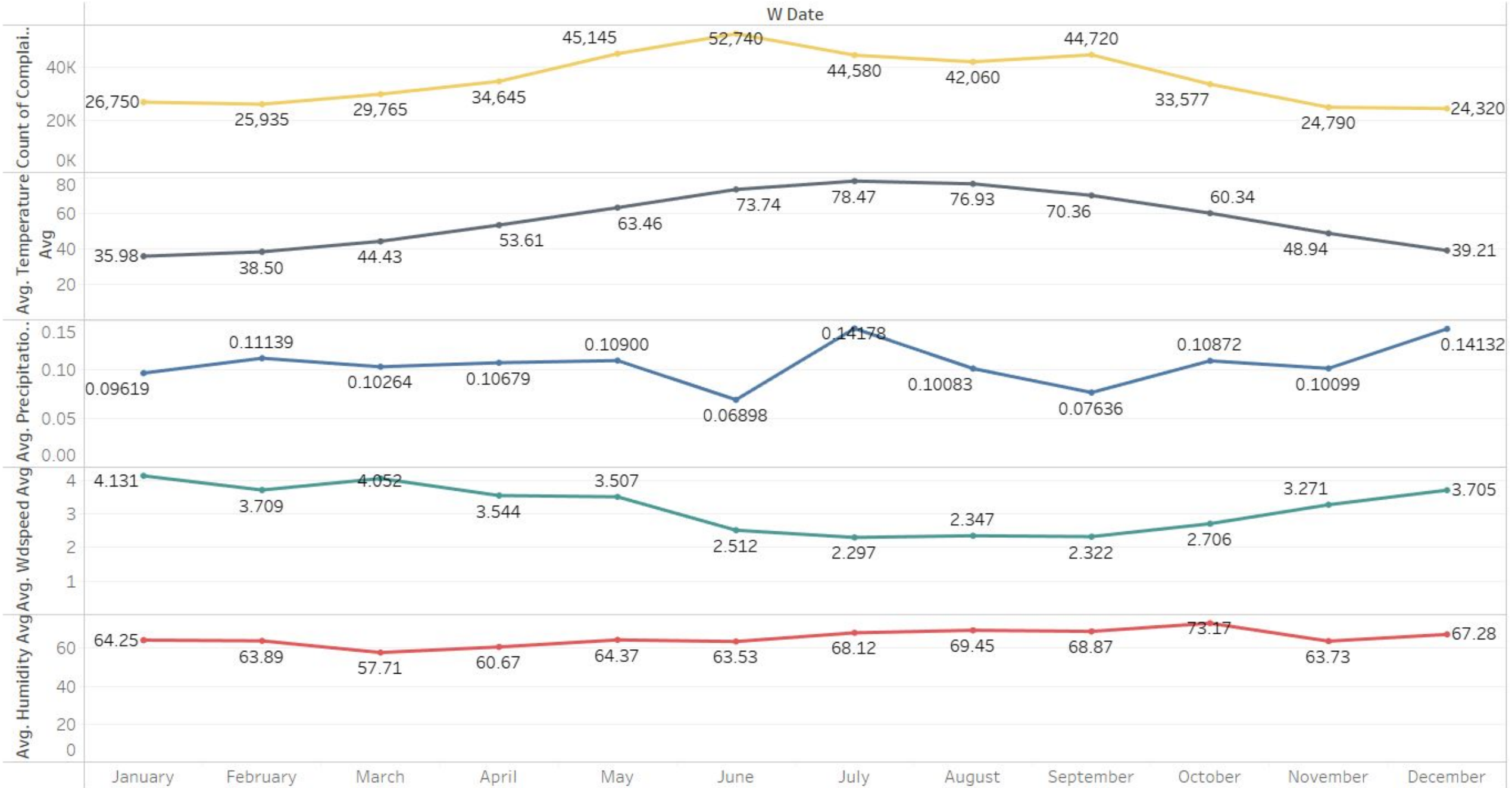
KPI Visualization

Count of Complaints Made over the Year Compared to Temperature



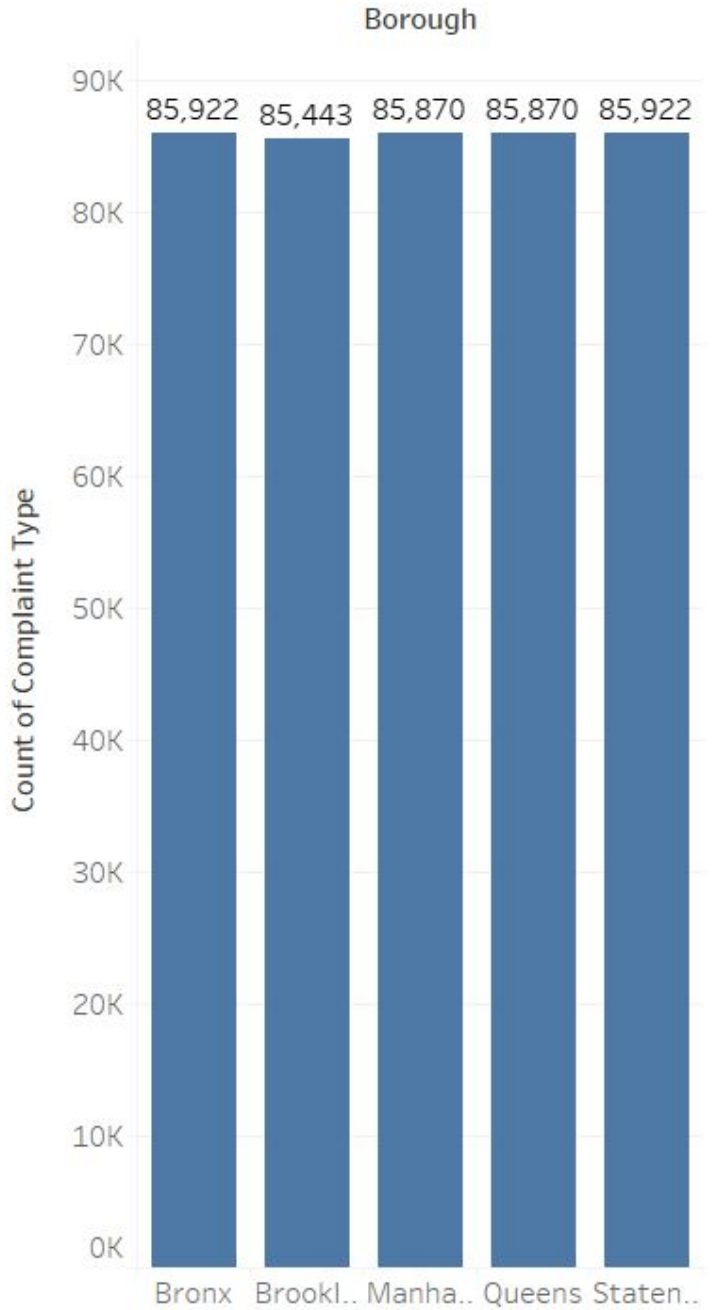
Complaint Type
- Collection Truck Noise
- Noise
- Noise - Commercial
- Noise - Helicopter
- Noise - House of Wors..
- Noise - Park
- Noise - Residential
- Noise - Street/Sidewa..
- Noise - Vehicle

Average Monthly Weather Forecast from 2016 - 2021 and Number of Complaints



Complaint Groups

Count of Complaints per Borough



Tools Description

Descriptions of the tools (databases, visualization tools, ETL, programming languages, etc.) used to complete the project

- **Programming Language.** Python: we used Python as the programing language due to its functional capabilities and features that supports functional programming. (via Google Collab & Jupyter Notebook)
- **Visualization Tool.** Tableau: Tableau is a software that helps visualize data and reveal patterns for analysis in business intelligence, making the data more understandable.
- **Database Tool.** Google BigQuery: Google BigQuery is a scalable data warehouse that provides a built-in query engine.
- **Other:** Diagram.drawio: Free and open-source software for making diagrams and charts.

Conclusion

A narrative conclusion section that describes:

a) the software and database tools the group used to coordinate and manage the project as well as carry out the programming tasks (list of bullet points with software or service and one sentence of what it was used for)

- Python
  - Pandas - python package that allowed us to manipulate data and create tables from the data provided.
  - BigQuery - python package that connects python with google bigquery, allowed us to send extracted data into BigQuery.

b) the group's experience with the project (which steps were the most difficult? Which were the easiest? what did you learn that you did not imagine you would have? if you had to do it all over again, what would you have done differently?)

- As for a group project, the hardest part was to find time for the group to meet and collaborate together
- For the project steps and phases, the ETL, the transformation part was the most challenging. We also found choosing the ETL tool, and choosing a target DBMS to host the Data Warehouse challenging. As for the ETL tool, we chose to build the ETL code by programming it in Python as we needed to work with the skill-set that our team had. We could have used different tools but that would have required more researching and more tool learning.
- The visualization part was the easiest and probably the most rewarding since you can finally use the queried data, to visualize, analyze, understand and aid business decisions

c) if the proposed benefits can be realized by the new system.

- Understanding the data is critical for decision making. For example, analyzing the 311 noise complaint data would be important when trying to learn what are the problematic areas in the city. With BI Tools, Stakeholders can use data to do sophisticated analysis, do data mining, create predicting models and eventually invest resources and solutions that better fits the needs of a particular areas to mitigate problems even before their occur.

d) any final comments and conclusions

- People and organizations need to be more data-driven. With new technologies being introduced that are more data being captured that can be used to aid the decision-making process. Data is abundant and is powerful. Using data, makes decisions more objective, analytical and less subjective and bias.

Reference

A References list that provides the web sites and other sources for data, techniques, methods, software, etc. used to complete the project

- Python: https://www.python.org/
- Tableau: https://www.tableau.com/
- Google BigQuery: https://app.diagrams.net/
- Diagram.drawio: https://app.diagrams.net/
- How to combine two columns of text: https://sparkbyexamples.com/pandas/pandas-combine-two-columns-of-text-in-dataframe/
- splitting the contents of a column using the .split method: https://stackoverflow.com/questions/37600711/pandas-split-column-into-multiple-columns-by-comma
- How to convert the values of an dataframe column to int: https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/#:~:text=to%20int%20(Integer)-,Use%20pandas%20DataFrame.,int64%20%2C%20numpy.
- How to get a year number in pandas, given a date (in datetime format): https://www.geeksforgeeks.org/get-month-and-year-from-date-in-pandas-python/
- How to get the name of a month give a date (in datetime format): https://pynative.com/python-get-month-name-from-number/#:~:text=Use%20the%20dt.,the%20month%20name%20in%20pandas.
- How to get the name of a day give a date (in datetime format): https://www.geeksforgeeks.org/get-day-from-date-in-pandas-python/
- Getting Started with NYC OpenData and the Socrata API: https://holowczak.com/getting-started-with-nyc-opendata-and-the-socrata-api/?doing_wp_cron=1671416671.5727460384368896484375

Additional Slides

Team 3 Project Proposal

For this project we are tasked with analyzing data taken over a 5-year period concerning 311 complaints to identify areas in the city originating the largest number of 311 complaints. Our team will then map the type of complaints and the average number of complaints by neighborhood. The data will help compare the amount of noise complaints and complaints related to weather

Issue for analysis:

Which type of noise complaints happen throughout the day. For example, is it more likely that car noise would more likely happen in the night? Is it more likely for domestic noises to happen during the morning?
We would look for the relation time of day versus the type of noise complaint so we can find out which noise complaints would likely happen during certain times of the day.

Data Sets:

311 Service Requests from 2010 to Present | NYC Open Data
Daily Climate Report
weather_data_5_boroughs_daily

Meeting Logs

Meeting Log Milestone 1

Meeting Log:

- **Date/Meeting time:**
  - Wednesday September 28, 12:30 -1:00 pm
  - Wednesday October 5, 12:00 pm -1:00pm
- **Attendees**:
  - Rosie Morgan
  - Christian Casino,
  - Emil Purisic
  - Hali Siew
  - Jason Chen
- **Agenda** (Zoom Meeting)
  - Discuss the changes to project proposal, the type of complains our team will focus on
  - Discuss data and dimensional models
  - Discuss roles and responsibilities

Meeting Log Milestone 2

Meeting Log:

- **Date/Meeting time:**
  - Wednesday October 12, 12:00 pm -1:00pm
  - Wednesday October 19, 12:00 pm -1:00pm
- **Attendees**:
  - Rosie Morgan
  - Christian Casino
  - Emil Purisic
  - Hali Siew
  - Jason Chen
- **Agenda** (Zoom Meeting)
  - Discuss the changes to project proposal, the type of complains our team will focus on
  - Discuss data and dimensional models
  - Discuss roles and responsibilities

Meeting Log Milestone 3

Meeting Log:

- **Date/Meeting time:**
  - Monday, October 31, 12:00 pm -12:15pm
- **Attendees**:
  - Rosie Morgan
  - Christian Casino
  - Emil Purisic
  - Hali Siew
  - Jason Chen
- **Agenda** (Zoom Meeting)
  - Discuss Professor Feedback given on October 27th
  - Delegated tasks regarding revision of the models (311 and Weather data)
  - Briefly discussed next steps (i.e. ETL process, etc.)

Meeting Log Milestone 4

Meeting Log:

- **Date/Meeting time:**
  - Wednesday, November 7, 12:00 pm -12:15pm
- **Attendees**:
  - Rosie Morgan
  - Christian Casino
  - Emil Purisic
  - Hali Siew
  - Jason Chen
- **Agenda** (Zoom Meeting)
  - Discuss the tools we will be using for ETL as well the Target DBMS our team will select for the project

Meeting Log Milestone 5

Meeting Log:

- **Date/Meeting time:**
  - Monday, November 28, 12:00 pm -12:15pm
  - Wednesday, December 7, 12:00 pm -12:15pm
- **Attendees**:
  - Rosie Morgan
  - Christian Casino
  - Emil Purisic
  - Hali Siew
  - Jason Chen
- **Agenda** (Zoom Meeting)
  - Next steps: Discuss the tools we will be using for ETL as well the Target DBMS our team will select for the project

Figure out how to do ETL on Python (Christian and Hali)

- Check with the teacher if there are resources that we could leverage to create the ETL process in Python? (Rosie)

- Create the SQL code that creates the tables
  - Team should get a refresher on SQL statements (data camp or homework) so we can all collaborate
    - Jason, Rosie to work on the 311 tables (Jason, Rosie)
      - Look at the existing dimensions and create the sql statements
  - Copy and paste the code for each of the dimension on that module that creates the specific tables (Jason)
    - 311 and weather
  - We have the modules but need to create the dataset (Jason)

Meeting Log - final Milestone

Date/Meeting time:

- Wednesday, December 12, 12:00 pm -12:15pm
  - Attendees: Rosie, Christian, Hali, Jason, Emil
- Wednesday, December 14, 12:00 pm -12:15pm
  - Attendees: Rosie, Christian, Hali, Jason, Emil
- Friday, December 16, 12:00 pm -12:15pm
  - Attendees: Rosie, Christian, Hali, Jason
- Sunday, December 18, 12:00 pm - 8:00 pm
  - Attendees: Rosie, Christian, Hali, Jason

- **Agenda** (Zoom Meeting)
  - Collaborate on finalizing the project
  - Team to work on the various slides for the final project:
    - A separate cover page listing the course, group members' names and project title (Rosie)
    - An introduction section similar to the proposal section, including the narrative description of the business or organization used for the data warehouse being created and a description of the source data. Be sure to include appropriate citations of each data source (web site, curator, date accessed, etc.) (Rosie & Hali)
    - The dimensional model diagram. (Hali)
    - A description and screen pictures of each of the ETL processes and any custom code used to process the source data. (Christian & Jason)
    - Screenshots and brief descriptions of the final dimensional schema that the business analytics tools are working with (Christian & Jason)
    - Screenshots and descriptions of the KPI visualizations (at least 3) on the dashboard application developed based on the data warehouse data (Jason)
    - Descriptions of the tools (databases, visualization tools, ETL, programming languages, etc.) used to complete the project (Rosie)
    - A narrative conclusion section (Rosie)

      a) the software and database tools the group used to coordinate and manage the project as well as carry out the programming tasks (list of bullet points with software or service and one sentence of what it was used for)

      b) the group's experience with the project (which steps were the most difficult? Which were the easiest? what did you learn that you did not imagine you would have? if you had to do it all over again, what would you have done differently?)

      c) if the proposed benefits can be realized by the new system.

      d) any final comments and conclusions
    - A References list that provides the web sites and other sources for data, techniques, methods, software, etc. used to complete the project. (Rosie, Hali, Christian)