

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
15558872 1	10/5/2022	Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk	11229	1815 East 17 Street	East 17 Street	Brooklyn	Brooklyn

Dataset 1: 311 Complaints

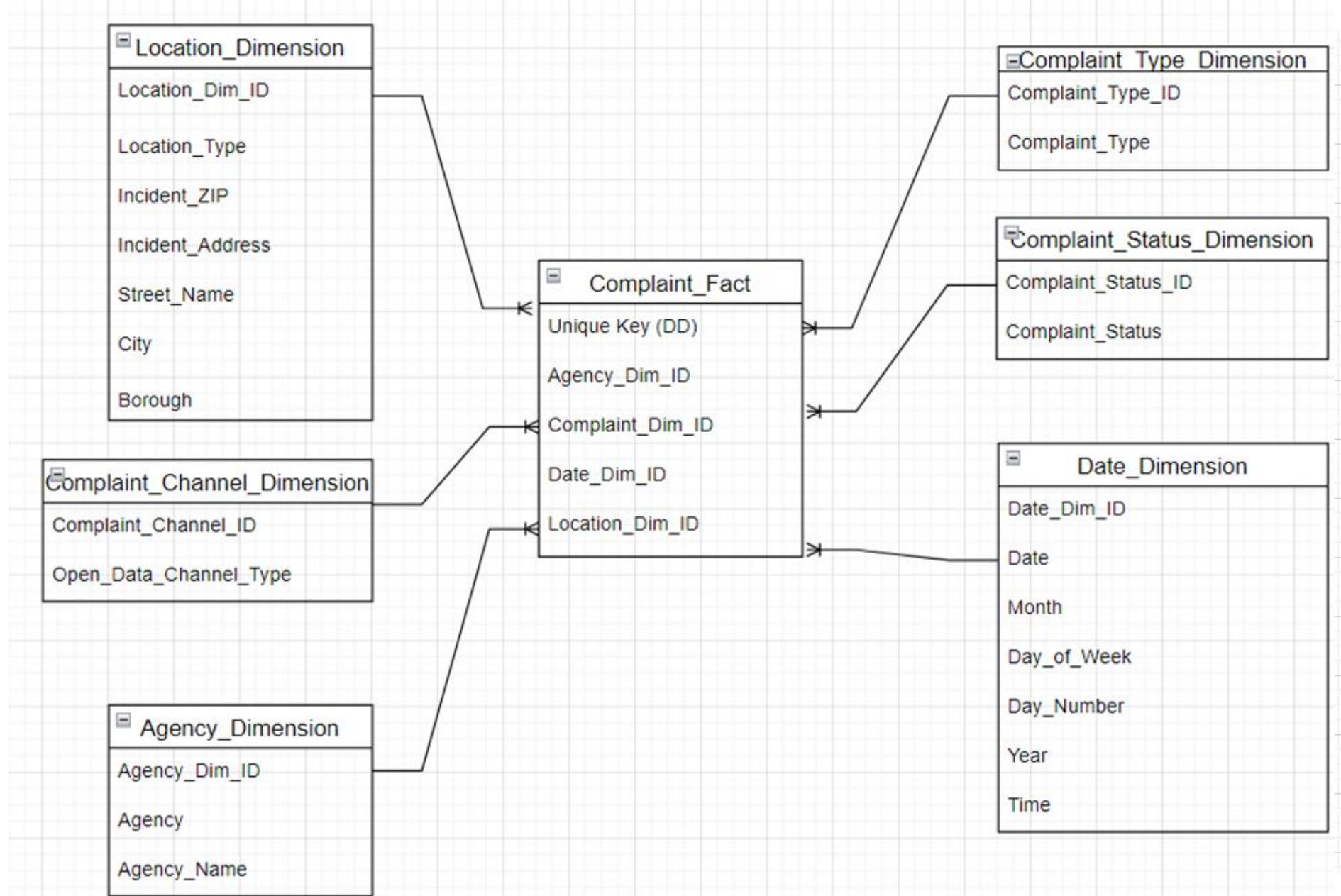
Date	Location	Maximum temperature	Minimum temperature	Average temperature	Precipitation	New Snow	Snow Depth
10/4/2022	NY Central Area	55	46	50.5	1.85	0	0

Dataset 2: Weather dataset

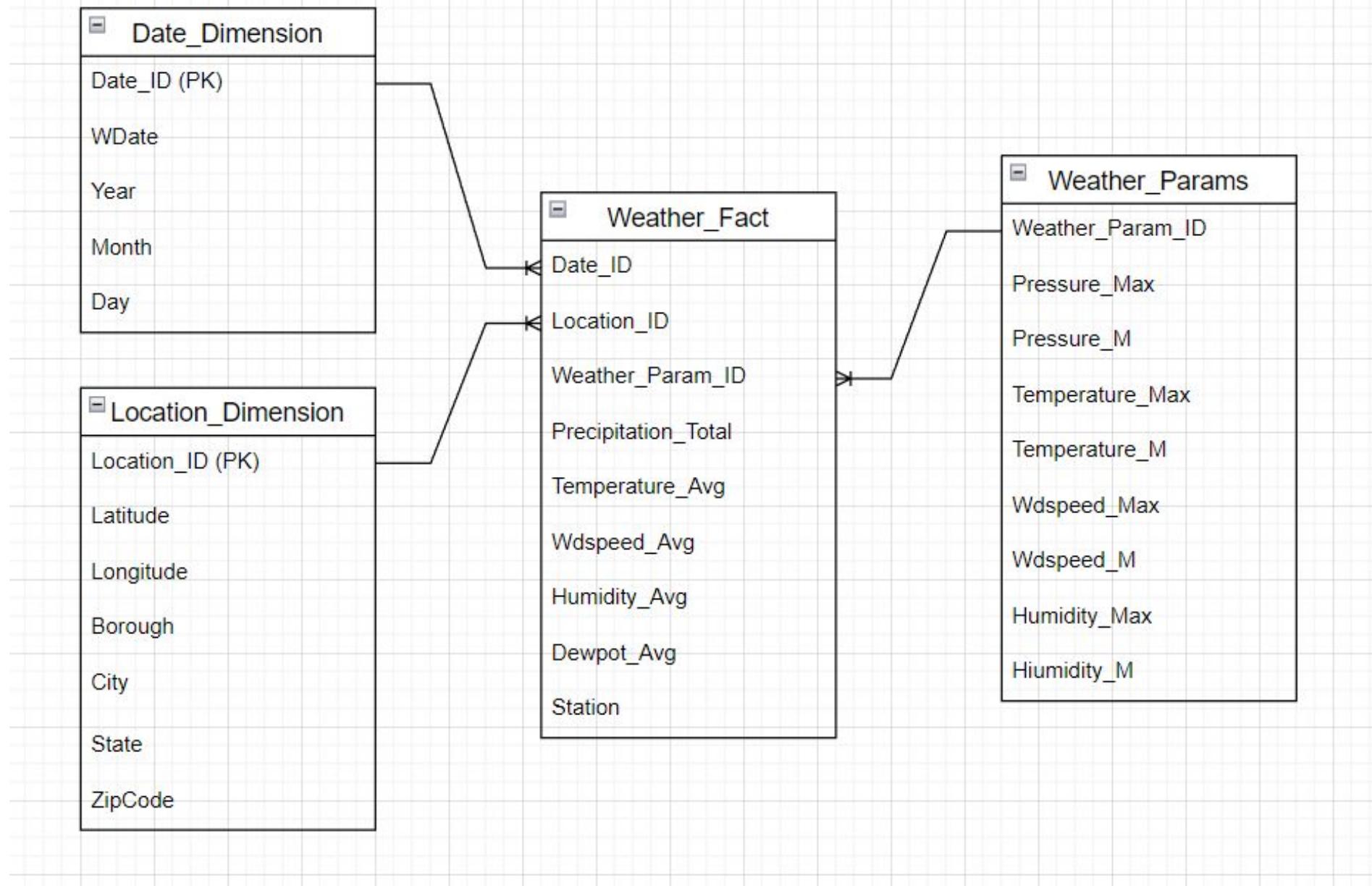
WDate	Borough	Temperature_Max	Temperature_Avg	Temperature_M	Precipitation_Total
10/4/2022	Manhattan	52.9	46	39.9	1.85

Dataset 3: Weather dataset 2

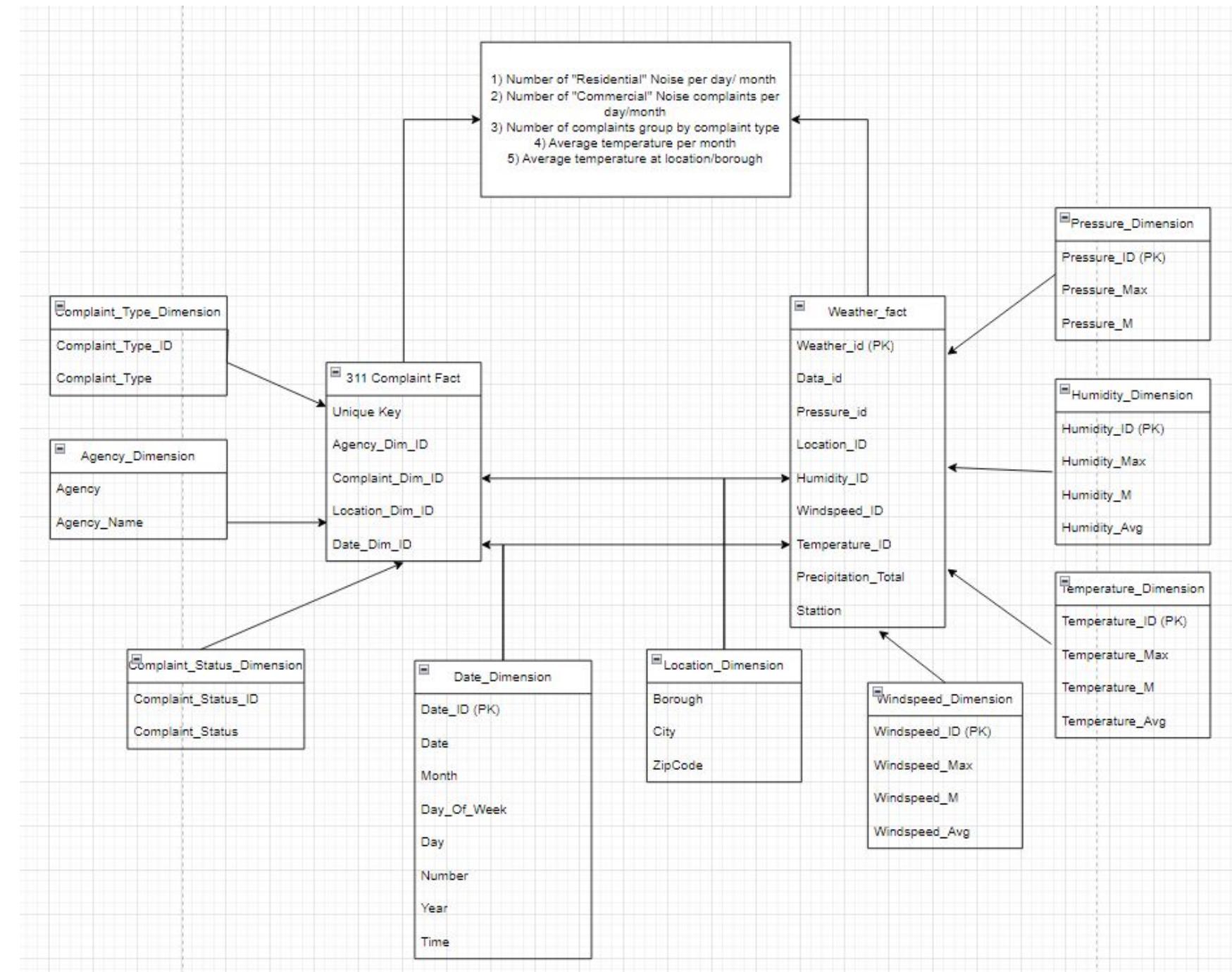
Updated 311 Complaints Dimensional Model



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 1o1ZAnsDfC2PgrQFPQ5RBNkhwVvB5h8Iv  
  
/local/lib/python3.8/dist-packages/gdown/cli.py:127: FutureWarning: Option `--id` was depr  
rnings.warn(  
loading...  
: https://drive.google.com/uc?id=1o1ZAnsDfC2PgrQFPQ5RBNkhwVvB5h8Iv  
/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	Bridge 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	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	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	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	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	40.7	

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

```
1 master_columns =df[['Created Date', 'Agency', 'Agency Name', 'Complaint Type', 'Location Type','Incident Zip',
2                      'Incident Address', 'Street Name', 'City', 'Status', 'Borough',
3 'Open Data Channel Type']]
4 print(len(master_columns))
5 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

```
1 # Filling the columns that are known to have NaN values and filling them with
2 # 'Not Applicable' will remove all remaining NaN values from the dataframe
3
4 # The borough column will always have a value according to the data profiling done
5 # Most Cities in New York City share the same name as their borough (for the exception of Queens)
6 # So fill NaN values from the City column with the name of the borough
7 master_columns[['City']] = master_columns[['City']].fillna(master_columns[['Borough']])
8
9 print(len(master_columns))
10 master_columns.head()
```

```
100586
/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	BR
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	07/06/2019 04:36:32 AM	NYPD	New York City Police Department	Noise - Commercial	Store/Commercial	11372	95-15B NORTHERN BOULEVARD	NORTHERN BOULEVARD	

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

```
1 location_dim_tbl = master_columns[['Location Type', 'Incident Zip', 'Incident Address', 'Street Name', 'City', 'Borough']]  
2 print(len(location_dim_tbl))  
3 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	95-158 NORTHERN BOULEVARD	NORTHERN BOULEVARD	JACKSON HEIGHTS	QUEENS

```
1 # We need to rename the column names to have underscores as required for columns in GBQ  
2 location_dim_tbl = location_dim_tbl.rename(columns = {'Location Type':'Location_Type', 'Incident Zip': 'Incident_Zip', 'Incident Address':'Incident_Address', 'Street Name': 'Street_Name'})  
3 print(len(location_dim_tbl))  
4 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	95-158 NORTHERN BOULEVARD, JACKSON HEIGHTS, NE...	Store/Commercial	11372	95-158 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

```
1 # Creating the composite key, which will temporarily serve as the surrogate key for the data
2 location_dim_tbl.insert(0,'Location_Dim_ID', range(1, 1+ len(location_dim_tbl)))
3
4 # Concatenating the columns which are crucial for distinguishing a row entry
5 # How to combine two columns of text: https://sparkbyexamples.com/pandas/pandas-combine-two-columns-of-text-in-dataframe/
6
7 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))
8
9
10 print(len(location_dim_tbl)) # It turns out each row is unique, no duplicates
11 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

```
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_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

```
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_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-f. 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

```
1 # Creating the surrogate key
2 agency_dim_tbl.insert(0,'Agency_Dim_ID', agency_dim_tbl['Agency'] + ' - ' + agency_dim_tbl['Agency_Name'])
3 print(len(agency_dim_tbl))
4 agency_dim_tbl.head()
5
```

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`)

```
1 complaint_type_dim_tbl2 = complaint_type_dim_tbl.copy()
2 complaint_type_dim_tbl2 = complaint_type_dim_tbl.drop_duplicates()
3 print(len(complaint_type_dim_tbl2))
4 complaint_type_dim_tbl2.head()
```

9

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

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

9

	Complaint_Type_ID	Complaint Type	edit
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

```
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_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`)

```
1 complaint_status_dim_tbl2 = complaint_status_dim_tbl.copy()
2 complaint_status_dim_tbl2 = complaint_status_dim_tbl.drop_duplicates()
3 print(len(complaint_status_dim_tbl2))
4 complaint_status_dim_tbl2.head()
```

7

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

```
1 # Creating the surrogate key
2 complaint_status_dim_tbl2.insert(0,'Complaint_Status_ID', range(1, 1+ len(complaint_status_dim_tbl2)))
3 print(len((complaint_status_dim_tbl2)))
4 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

```
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_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

```
1 # splitting the contents of a column using the .split method: https://stackoverflow.com/questions/37600711/pandas-split-column-into-multiple-columns-by-comma
2 date_slice = master_columns['Created Date'].str.split(' ', expand = True)
3 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

```
1 # splitting the contents of a column using the .split method: https://stackoverflow.com/questions/37600711/pandas-split-column-into-multiple-columns-by-comma
2 date_slice2 = date_slice[0].str.split('/', expand = True)
3
4 # How to convert the values of an dataframe column to int:
5 # https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/#:~:text=to%20int%20\(Integer\)-,Use%20pandas%20DataFrame.,int64%20%2C%20numpy.
6
7 day_nums = date_slice2[1].astype('int')
8 date_slice2
9
10
```

	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

```
1 # splitting the contents of a column using the .split method: https://stackoverflow.com/questions/37600711/pandas-split-column-into-multiple-columns-by-comma
2 date_slice2 = date_slice[0].str.split('/', expand = True)
3
4 # How to convert the values of an dataframe column to int:
5 # https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/#:~:text=to%20int%20\(Integer\)-,Use%20pandas%20DataFrame.,int64%20%2C%20numpy.
6
7 day_nums = date_slice2[1].astype('int')
8 date_slice2.head()
```

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

```
1 date_slice[0] = pd.to_datetime(date_slice[0])
2 date_slice[1]= pd.to_datetime(date_slice[1]) # this is really the format of the data
3 date_slice[0] = date_slice[0].copy()
4 date_slice[0] = pd.to_datetime(date_slice[0])
5
6 #date_slice = date_slice.drop(2, axis=1)
7
8 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

```
1 # Source: https://www.geeksforgeeks.org/get-month-and-year-from-date-in-pandas-python/
```

```
2 year_num = date_slice[0].dt.year
3 year_int = year_num.astype('int')
4 year_int
5
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1010	10
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	----

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

```
1 date_dim_tbl.insert(1, 'Date', date_slice[0])
2
3 # How to get the name of a month give a date (in datetime format):
4 # Source: https://pynative.com/python-get-month-name-from-number/#:~:text=Use%20the%20dt.,the%20month%20name%20in%20pandas.
5 date_dim_tbl.insert(2, 'Month', date_slice[0].dt.month_name())
6
7 # How to get the name of a day give a date (in datetime format):
8 # Source: https://www.geeksforgeeks.org/get-day-from-date-in-pandas-python/
9 date_dim_tbl.insert(3, 'Day_Of_Week', date_slice[0].dt.day_name() ) # date_slice[0].dt.dayofweek
10
11 # The following commented code was used to create the variable day_nums
12 # in the cell above:
13 # date_slice2 = date_slice[0].str.split('/', expand = True)
14 # How to convert the values of an dataframe column to int:
15 # https://sparkbyexamples.com/pandas/pandas-convert-column-to-int/#:~:text=to%20int%20(Integer)-,Use%20pandas%20DataFrame.,int64%20%2C%20numpy.
16 # day_nums = date_slice2[1].astype('int')
17 date_dim_tbl.insert(4, 'Day_Number', day_nums)
18
19 # How to get the name of a year give a date (in datetime format):
20 # The following commented code was used to create the variable day_nums
21 # in the cell above:
22 # Source: https://www.geeksforgeeks.org/get-month-and-year-from-date-in-pandas-python/
23 # year_num = date_slice[0].dt.year
24 # year_int = year_num.astype('int')
25 date_dim_tbl.insert(5, 'Year', year_int)
26
27 date_dim_tbl.insert(6, 'Time', date_slice[1])
28
29 #date_dim_tbl = date_slice.drop(0, axis=1) # Causes everything to get deleted
30 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)

```
1 # Creating the composite key
2 # date_dim_tbl['Date_And_Time'] = date_dim_tbl['Date'] + ' - ' + date_dim_tbl['Time']
3 # Don't need this variable as "Created_Date" has date and time
4 # date_dim_tbl = date_dim_tbl.drop_duplicates()
5
6 # Created date already gives both date and time, which are critical to distinguishing
7 # Unique rows
8 date_dim_tbl = date_dim_tbl.rename(columns = {'Created_Date': 'Date_Dim_ID'})
9
10 print(len(date_dim_tbl))
11 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

```
1 date_dim_tbl2 = date_dim_tbl.copy()
2 date_dim_tbl2 = date_dim_tbl2.drop_duplicates()
3 print(len(date_dim_tbl2))
4 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

```
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_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)

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

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

```
1 print(len(complaint_fact_tbl))
2 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

```
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_311_fact = dataset_name + '.' + 'complaint_fact_tbl'
```

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

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

```
1 ! gdown --id 1BFPm2cptd9EWD_187KS1YQrMUVy9sNzf
```

```
/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\_187KS1YQrMUVy9sNzf
```

```
To: /content/cis4400proj-370416-74e9d8391c22.json
```

```
100% 2.32k/2.32k [00:00<00:00, 1.43MB/s]
```

```
1 # Reading/Writing into Google Cloud project
```

```
2 # https://data.cityofnewyork.us/resource/erm2-nwe9.json
```

```
3
```

```
4 # Setting up the environment variable
```

```
5 os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'cis4400proj-370416-74e9d8391c22.json'
```

```
6 client = bigquery.Client()
```

13. Loading Data for the Location Dimension into Google BigQuery

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

```
1 job_config = bigquery.LoadJobConfig(
2
3     # Specify a partial schema. All columns are always written to the table
4     # The schema is used to assist in data type definitions
5
6     schema = [
7
8         bigquery.SchemaField("Location_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
9         bigquery.SchemaField("Location_Type", bigquery.enums.SqlTypeNames.STRING),
10        bigquery.SchemaField("Incident_Zip", bigquery.enums.SqlTypeNames.INTEGER),
11        bigquery.SchemaField("Incident_Address", bigquery.enums.SqlTypeNames.STRING),
12        bigquery.SchemaField("Street_Name", bigquery.enums.SqlTypeNames.STRING),
13        bigquery.SchemaField("City", bigquery.enums.SqlTypeNames.STRING),
14        bigquery.SchemaField("Borough", bigquery.enums.SqlTypeNames.STRING)
15    ],
16
17    write_disposition = "WRITE_TRUNCATE",
18
19 )
```

```
1 job = client.load_table_from_dataframe(
2     location_dim_tbl2, table_id_loc_dim, job_config= job_config
3 )
4 job.result()
```

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

The screenshot shows the Google BigQuery interface for a table named 'location_dim_tbl2'. At the top, there are tabs for 'SCHEMA', 'DETAILS', and 'PREVIEW'. Below the tabs, there is a search bar labeled 'Filter' with the placeholder 'Enter property name or value'. A table lists the schema fields:

<input type="checkbox"/>	Field name	Type	Mode
<input type="checkbox"/>	Location_Dim_ID	STRING	NULLABLE
<input type="checkbox"/>	Location_Type	STRING	NULLABLE
<input type="checkbox"/>	Incident_Zip	INTEGER	NULLABLE
<input type="checkbox"/>	Incident_Address	STRING	NULLABLE
<input type="checkbox"/>	Street_Name	STRING	NULLABLE
<input type="checkbox"/>	City	STRING	NULLABLE
<input type="checkbox"/>	Borough	STRING	NULLABLE

At the bottom, there are two buttons: 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'.

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

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

1 job_config = bq.LoadJobConfig(
2
3     # Specify a partial schema. All columns are always written to the table
4     # The schema is used to assist in data type definitions
5
6     schema = [
7
8         bq.SchemaField("Complaint_Channel_ID", bq.enums.SqlTypeNames.INTEGER),
9         bq.SchemaField("Open_Data_Channel_Type", bq.enums.SqlTypeNames.STRING)
10    ],
11
12    write_disposition = "WRITE_TRUNCATE",
13
14 )
```

```
1 job = client.load_table_from_dataframe(
2     complaint_channel_dim_tbl2, table_id=odcc, job_config=job_config
3 )
4 job.result()
```

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

complaint_channel_dim_tbl2			
SCHEMA	DETAILS	PREVIEW	
Filter Enter property name or value			
<input type="checkbox"/> Field name	Type	Mode	
<input type="checkbox"/> Complaint_Channel_ID	INTEGER	NULLABLE	
<input type="checkbox"/> Open_Data_Channel_Type	STRING	NULLABLE	
EDIT SCHEMA	VIEW ROW ACCESS POLICIES		

15. Loading Data for the Agency Dimension into Google BigQuery

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

```
1 job_config = bigquery.LoadJobConfig(
2
3     # Specify a partial schema. All columns are always written to the table
4     # The schema is used to assist in data type definitions
5
6     schema = [
7
8         bigquery.SchemaField("Agency_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
9         bigquery.SchemaField("Agency", bigquery.enums.SqlTypeNames.STRING),
10        bigquery.SchemaField("Agency_Name", bigquery.enums.SqlTypeNames.STRING),
11    ],
12
13    write_disposition = "WRITE_TRUNCATE",
14
15 )
```

```
1 job = client.load_table_from_dataframe(
2     agency_dim_tbl2, table_id_agency, job_config= job_config
3 )
4 job.result()
```

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

The screenshot shows the Google BigQuery web interface. At the top, there is a search bar labeled 'QUERY' and a '+' button. Below the search bar, the table name 'agency_dim_tbl2' is displayed. Underneath the table name, there are three tabs: 'SCHEMA' (which is selected), 'DETAILS', and 'PREVIEW'. A 'Filter' input field is present above the schema table. The schema table has four columns: 'Field name', 'Type', and 'Mode'. There are four rows in the schema table, each corresponding to one of the three fields defined in the Python code: 'Agency_Dim_ID', 'Agency', and 'Agency_Name', all defined as STRING type and NULLABLE mode.

<input type="checkbox"/>	Field name	Type	Mode
<input type="checkbox"/>	Agency_Dim_ID	STRING	NULLABLE
<input type="checkbox"/>	Agency	STRING	NULLABLE
<input type="checkbox"/>	Agency_Name	STRING	NULLABLE

[EDIT SCHEMA](#) [VIEW ROW ACCESS POLICIES](#)

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

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

1 job_config = bq.LoadJobConfig()
2
3 # Specify a partial schema. All columns are always written to the table
4 # The schema is used to assist in data type definitions
5
6 schema = [
7
8     bq.SchemaField("Complaint_Type_ID", bq.enums.SqlTypeNames.INTEGER),
9     bq.SchemaField("Complaint_Type", bq.enums.SqlTypeNames.STRING)
10 ],
11
12 write_disposition = "WRITE_TRUNCATE",
13
14 )

1 job = client.load_table_from_dataframe(
2     complaint_type_dim_tbl2, table_id_comp_type, job_config= job_config
3 )
4 job.result()

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

complaint_type_dim_tbl2 QUERY ▾

SCHEMA DETAILS PREVIEW

Filter Enter property name or value

<input type="checkbox"/> Field name	Type	Mode
Complaint_Type_ID	INTEGER	NULLABLE
Complaint_Type	STRING	NULLABLE

[EDIT SCHEMA](#) [VIEW ROW ACCESS POLICIES](#)

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

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

1 job_config = bq.LoadJobConfig(
2
3     # Specify a partial schema. All columns are always written to the table
4     # The schema is used to assist in data type definitions
5
6     schema = [
7
8         bq.SchemaField("Complaint_Status_ID", bq.enums.SqlTypeNames.INTEGER),
9         bq.SchemaField("Complaint_Status", bq.enums.SqlTypeNames.STRING)
10    ],
11
12    write_disposition = "WRITE_TRUNCATE",
13
14 )
```

```
1 job = client.load_table_from_dataframe(
2     complaint_status_dim_tbl2, table_id_comp_status, job_config= job_config
3 )
4 job.result()
```

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

complaint_status_dim_tbl2 QUERY ▾

SCHEMA DETAILS PREVIEW

Filter Enter property name or value

<input type="checkbox"/> Field name	Type	Mode
Complaint_Status_ID	INTEGER	NULLABLE
Complaint_Status	STRING	NULLABLE

[EDIT SCHEMA](#) [VIEW ROW ACCESS POLICIES](#)

18. Loading Data for the Date Dimension into Google BigQuery

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

1 job_config = bigquery.LoadJobConfig(
2
3     # Specify a partial schema. All columns are always written to the table
4     # The schema is used to assist in data type definitions
5
6     # Converting all the date values to string works
7
8     schema = [
9
10         bigquery.SchemaField("Date_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
11         bigquery.SchemaField("Date", bigquery.enums.SqlTypeNames.DATE),
12         bigquery.SchemaField("Month", bigquery.enums.SqlTypeNames.STRING),
13         bigquery.SchemaField("Year", bigquery.enums.SqlTypeNames.INTEGER),
14         bigquery.SchemaField("Day_Of_Week", bigquery.enums.SqlTypeNames.STRING),
15         bigquery.SchemaField("Day_Number", bigquery.enums.SqlTypeNames.INTEGER),
16         bigquery.SchemaField("Time", bigquery.enums.SqlTypeNames.TIME)
17     ],
18
19     write_disposition = "WRITE_TRUNCATE",
20
21 )
```

```
1 job = client.load_table_from_dataframe(
2     date_dim_tbl2, table_id_date, job_config= job_config
3 )
4 job.result()
```

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

The screenshot shows the Google BigQuery UI for a table named 'date_dim_tbl2'. The top navigation bar includes a search icon, 'QUERY', and a user icon. Below the header, there are three tabs: 'SCHEMA' (which is selected), 'DETAILS', and 'PREVIEW'. A 'Filter' input field is present. The schema table lists eight fields:

<input type="checkbox"/>	Field name	Type	Mode
<input type="checkbox"/>	Date_Dim_ID	STRING	NULLABLE
<input type="checkbox"/>	Date	DATE	NULLABLE
<input type="checkbox"/>	Month	STRING	NULLABLE
<input type="checkbox"/>	Day_Of_Week	STRING	NULLABLE
<input type="checkbox"/>	Day_Number	INTEGER	NULLABLE
<input type="checkbox"/>	Year	INTEGER	NULLABLE
<input type="checkbox"/>	Time	TIME	NULLABLE

At the bottom, there are two buttons: 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'.

19. Loading Data for the Fact Table into Google BigQuery

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

```
1 job_config = bigquery.LoadJobConfig(
2
3     # Specify a partial schema. All columns are always written to the table
4     # The schema is used to assist in data type definitions
5
6     # Converting all the date values to string works
7
8     schema = [
9
10         bigquery.SchemaField("Unique_Key", bigquery.enums.SqlTypeNames.INTEGER),
11         bigquery.SchemaField("Agency_Dim_ID", bigquery.enums.SqlTypeNames.STRING),
12         bigquery.SchemaField("Complaint_Type", bigquery.enums.SqlTypeNames.STRING),
13         bigquery.SchemaField("Complaint_Channel", bigquery.enums.SqlTypeNames.STRING),
14         bigquery.SchemaField("Complaint_Status", bigquery.enums.SqlTypeNames.STRING),
15         bigquery.SchemaField("Date", bigquery.enums.SqlTypeNames.DATE),
16         bigquery.SchemaField("Location_Dim_ID", bigquery.enums.SqlTypeNames.STRING)
17     ],
18
19     write_disposition = "WRITE_TRUNCATE",
20
21 )
```

```
1 job = client.load_table_from_dataframe(
2     complaint_fact_tbl, table_id_311_fact, job_config= job_config
3 )
4 job.result()
```

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

The screenshot shows the Google BigQuery web interface. At the top, it displays the table name 'complaint_fact_tbl'. To the right are navigation buttons for 'QUERY', 'SCHEMA' (which is underlined in blue), 'DETAILS', and 'PREVIEW'. Below these buttons is a search bar labeled 'Filter' with the placeholder 'Enter property name or value'. The main area contains a table with eight rows, each representing a schema field. The columns are 'Field name' (containing links like Unique_Key, Agency_Dim_ID, etc.), 'Type' (containing INTEGER, STRING, DATE, etc.), and 'Mode' (containing NULLABLE). At the bottom of the table are two buttons: 'EDIT SCHEMA' (in a blue box) and 'VIEW ROW ACCESS POLICIES'.

<input type="checkbox"/>	Field name	Type	Mode
<input type="checkbox"/>	Unique_Key	INTEGER	NULLABLE
<input type="checkbox"/>	Agency_Dim_ID	STRING	NULLABLE
<input type="checkbox"/>	Complaint_Type	STRING	NULLABLE
<input type="checkbox"/>	Complaint_Channel	STRING	NULLABLE
<input type="checkbox"/>	Complaint_Status	STRING	NULLABLE
<input type="checkbox"/>	Date	DATE	NULLABLE
<input type="checkbox"/>	Location_Dim_ID	STRING	NULLABLE

Weather Dataset ETL

```
In [25]: 1 import pandas as pd  
2 from google.cloud import bigquery  
3 import os  
4 data = pd.read_csv(r'C:\Users\apoll\Downloads\weather_data_5_boroughs_daily (2).csv')
```

```
In [26]: 1 data.head()
```

Out[26]:

	Station	latitude	longitude	Borough	City	State	ZipCode	WDate	Temperature_Max	Temperature_Avg	...	Humidity_M	Wdspeed_Max	Wdspe
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]: 1 df = data.iloc[:, :-3]
2 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]: 1 df.isnull().values.any()
```

```
Out[5]: False
```

```
In [6]: 1 df["WeatherParamsID"] = df['WDate'].astype(str) + "-" + df["Temperature_Avg"].astype(str) + " - " + df["ZipCode"].astype(str)
2 df['Year'] = pd.DatetimeIndex(df['WDate']).year
3 df['Month'] = pd.DatetimeIndex(df['WDate']).month
4 df['Day'] = pd.DatetimeIndex(df['WDate']).day
5 df["DateID"] = df['WDate'].astype(str) + "-" + df["ZipCode"].astype(str)
6 df['WDate'] = pd.to_datetime(df['WDate']).dt.date
7 #df["WDate"] = pd.to_datetime(df['WDate']).dt.date
8
9 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]: 1 df_address = df[["Borough","City","State","ZipCode","Station"]]
```

```
In [8]: 1 df_fact = df[['WeatherParamsID','ZipCode','WDate','Temperature_Avg','Dewpot_Avg','Humidity_Avg','Wdspeed_Avg','Precipitation']]
```

```
In [13]: 1 df_address = df_address.drop_duplicates()  
2 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]: 1 df_params = df[['WeatherParamsID', 'Temperature_Max', 'Temperature_M', 'Dewpot_Max', 'Dewpot_M', 'Humidity_Max', 'Humidity_M', 'Wds  
2 'Wdspeed_M', 'Pressure_Max', 'Pressure_M']]
```

```
In [8]: 1 df_date = df[["DateID", "WDate", "Year", "Month", "Day"]]  
2
```

```
In [9]: 1 os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = "cis4400proj-370416-74e9d8391c22.json"
```

```
In [15]: 1 client = bq.Client()
```

```
In [16]: 1 dataset_name = 'weather'
2 table_id = dataset_name + '.address_dim_tbl'
3 table_id2 = dataset_name + '.facts_dim_tbl'
4 table_id3 = dataset_name + '.params_dim_tbl'
5 table_id4 = dataset_name + '.date_dim_tbl'
```

```
In [27]: 1 job_config = bq.LoadJobConfig(
2     schema = [bq.SchemaField("Borough",bq.enums.SqlTypeNames.STRING)
3               ], write_disposition = "WRITE_TRUNCATE")
```

```
In [18]: 1 job_config2 = bq.LoadJobConfig(
2     schema = [bq.SchemaField("Station",bq.enums.SqlTypeNames.STRING)
3               ], write_disposition = "WRITE_TRUNCATE")
```

```
In [19]: 1 job_config3 = bq.LoadJobConfig(
2     schema = [bq.SchemaField("WeatherParamsID",bq.enums.SqlTypeNames.STRING)
3               ], write_disposition = "WRITE_TRUNCATE")
```

```
In [20]: 1 job_config4 = bq.LoadJobConfig(
2     schema = [bq.SchemaField("DateID",bq.enums.SqlTypeNames.STRING)
3               ], write_disposition = "WRITE_TRUNCATE")
```

```
In [28]: 1 job = client.load_table_from_dataframe(  
2 df_address, table_id, job_config=job_config)  
3  
4 job.result()
```

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

```
In [22]: 1 job2 = client.load_table_from_dataframe(  
2 df_fact, table_id2, job_config=job_config2)  
3  
4 job.result()
```

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

```
In [23]: 1 job3 = client.load_table_from_dataframe(  
2 df_params, table_id3, job_config=job_config3)  
3  
4 job.result()
```

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

```
In [24]: 1 job3 = client.load_table_from_dataframe(  
2 df_date, table_id4, job_config=job_config4)  
3  
4 job.result()
```

Final Dimension Schema

The screenshot shows the Google Cloud BigQuery Explorer interface. The top navigation bar includes 'Google Cloud' and 'CIS4400Proj'. A search bar at the top right says 'Search Products, resources, docs (/)'. Below the search bar, there's a breadcrumb trail: '*Joins > location_dim_tbl2 > facts_dim_tbl'. The main area is titled 'facts_dim_tbl' with tabs for 'QUERY' and 'SHARE'. The 'SCHEMA' tab is selected, showing a table structure with columns: WeatherParamsID (STRING, NULLABLE), ZipCode (INTEGER, NULLABLE), WDate (DATE, NULLABLE), Temperature_Avg (FLOAT, NULLABLE), Dewpot_Avg (FLOAT, NULLABLE), Humidity_Avg (FLOAT, NULLABLE), Wdspeed_Avg (FLOAT, NULLABLE), and Precipitation_Total (FLOAT, NULLABLE). Below the schema, there are buttons for 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'. At the bottom, there are links for 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

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

There are two data sets:
Complaints and Weather.

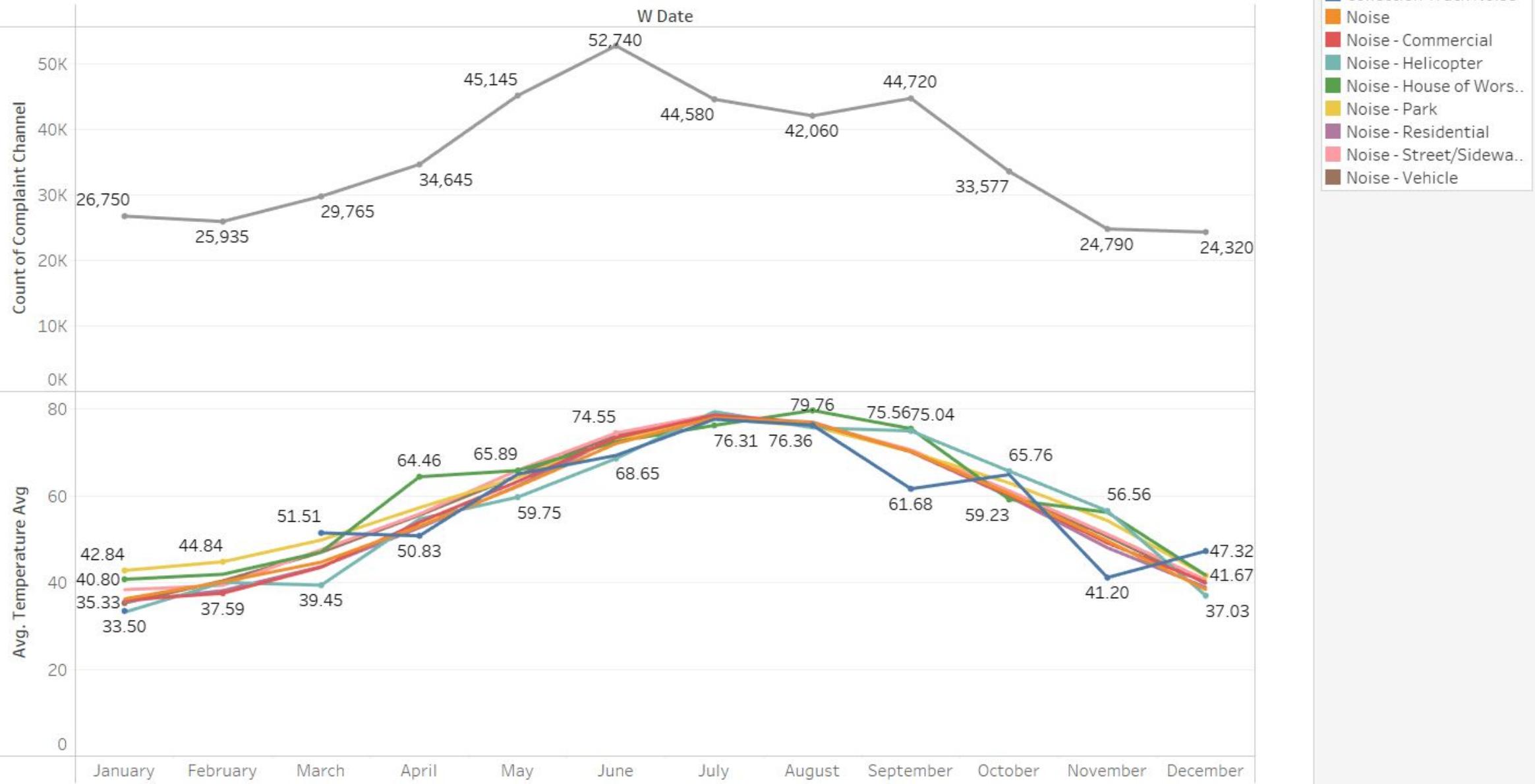
The 2 data sets were joined using left join on the Weather's Fact table to determine the correlation between complaints made throughout the year and different weather conditions.

Query to Join Complaints ETL and Weather ETL

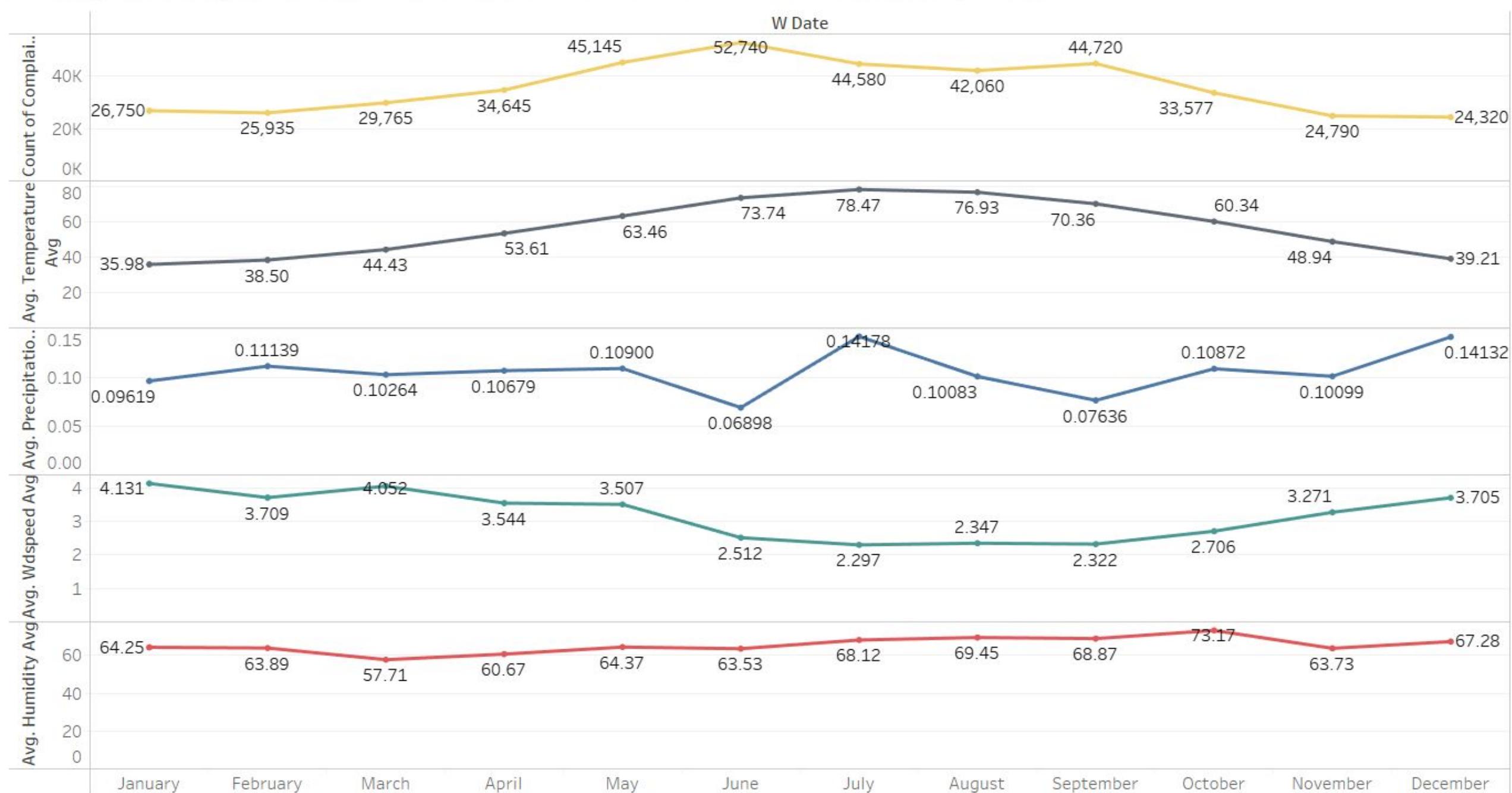
```
1 SELECT Agency_Dim_ID,Complaint_Type,Complaint_Status,ZipCode,  
2 WDate,Temperature_Avg,Humidity_Avg,Wdspeed_Avg,Precipitation_Total,  
3 Borough,City,State,Station,Complaint_Channel  
4 FROM `weather.facts_dim_tbl`  
5 LEFT JOIN `weather.address_dim_tbl`  
6 USING (ZipCode)  
7 LEFT JOIN `complaints.complaint_fact_tbl`  
8 ON Date = WDate  
9 LEFT JOIN `complaints.complaint_channel_dim_tbl2`  
10 ON Complaint_Channel = Open_Data_Channel_Type  
11 LEFT JOIN `complaints.complaint_status_dim_tbl2`  
12 USING (Complaint_Status)  
13 LEFT JOIN `complaints.complaint_type_dim_tbl2`  
14 USING (Complaint_Type)  
15 LEFT JOIN `complaints.agency_dim_tbl2`  
16 USING (Agency_Dim_ID)  
17  
18
```

KPI Visualization

Count of Complaints Made over the Year Compared to Temperature



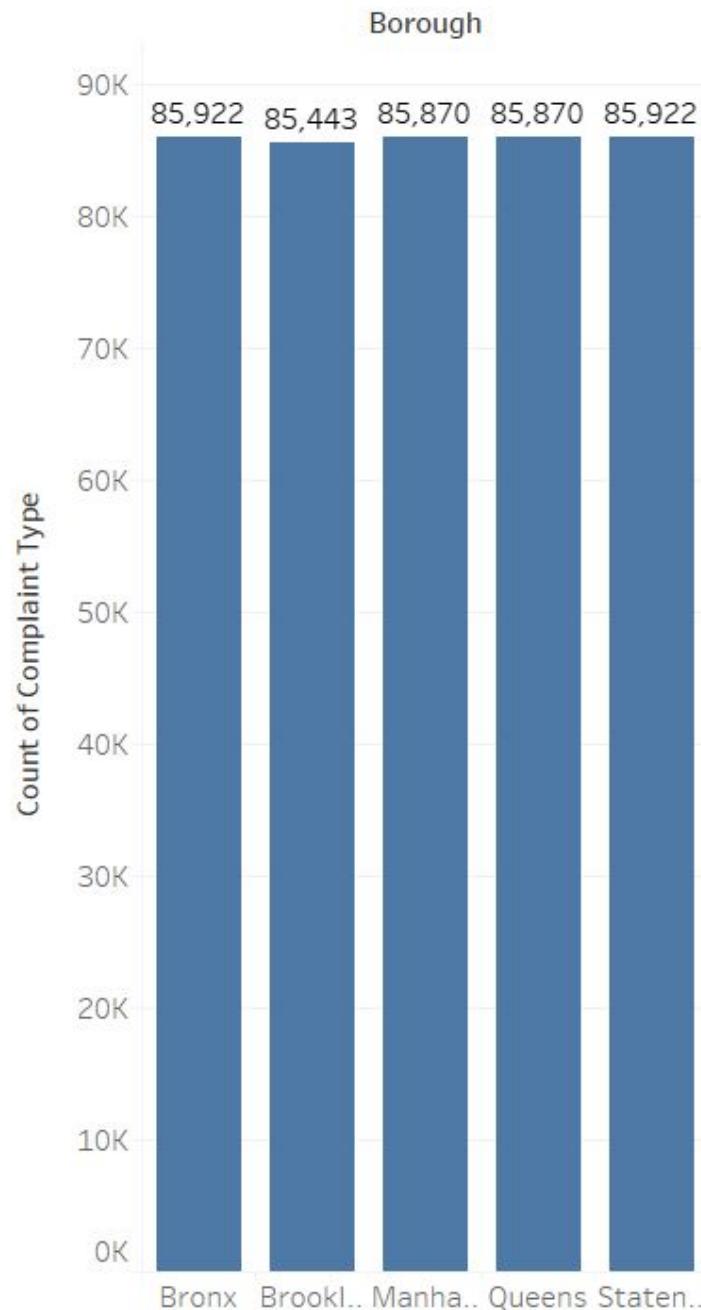
Average Monthly Weather Forecast from 2016 - 2021 and Number of Complaints



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 programming 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.](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](#))