The Effects of Traffic Light Conditions on Vehicle Accident Rates in New York City

CIS 4400 CMWA

Jake Li, JIANHUI.LI2@baruchmail.cuny.edu
Jeffrey Zhen, JEFFREY.ZHEN1@baruchmail.cuny.edu
Jonathan Garcia, Jonathan.garcia5@baruchmail.cuny.edu
Kamilla Sharipova, kamilla.sharipova@baruchmail.cuny.edu

Project Description

Over the course of this project, the group hopes to dig deeper into the effects of traffic signal conditions on vehicle collisions. We will be using the 311 complaints dataset and focusing on the traffic light complaints only. The 311 complaints data includes all the complaints/service request made to 311 in NYC. The 311 complaints dataset will be integrated with the Motor Vehicle Collisions - Crashes dataset. The Motor Vehicle Collisions crash dataset contains details on the crash event. Each row represents a crash event. A police report is required to be filled out for collisions where someone is injured or killed, or where there is at least $1000 worth of damage.
We hope this data warehouse can be used to identify the hot spot of complaints and collisions to enhance the decision-making process for where to deploy personnel, what's the most common type of traffic light complaints, etc.

Datasets:
311 Service Requests:
https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9
Motor Vehicle Collisions - Crashes:
https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95

311 complaints data columns:
Unique Key, Created Date, Closed Date, Agency, Agency Name, Complaint Type, Descriptor, Location Type, Incident Zip, Incident Address, Street Name, Cross Street 1, Cross Street 2, Intersection Street 1, Intersection Street 2, Address Type, City, Landmark, Facility Type, Status, Due Date, Resolution Description, Resolution Action Updated Date, Community Board, BBL, Borough, X Coordinate (State Plane), Y Coordinate (State Plane), Open Data Channel Type, Park Facility Name, Park Borough, Vehicle Type, Taxi Company Borough, Taxi Pick Up Location, Bridge Highway Name, Bridge Highway Direction, Road Ramp, Bridge Highway Segment, Latitude, Longitude, Location

Vehicle collisions data columns:
Crash Date, Crash Time, Borough, Zip Code, Latitude, Longitude, Location, On Street Name, Cross Street Name, Off Street Name, Number Of Persons Injured, Number Of Persons Killed, Number Of Pedestrians Injured, Number Of Pedestrians Killed, Number Of Cyclist Injured, Number Of Cyclist Killed, Number Of Motorist Injured, Number Of Motorist Killed, Contributing Factor Vehicle 1, Contributing Factor Vehicle 2, Contributing Factor Vehicle 3, Contributing Factor Vehicle 4, Contributing Factor Vehicle 5, Collision_Id, Vehicle Type Code 1, Vehicle Type Code 2, Vehicle Type Code 3, Vehicle Type Code 4, Vehicle Type Code 5
311 complaints data sample:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type
0	56428613	1/3/2023 7:35	1/3/2023 8:35	DOT	Department of Transportation	Traffic Signal Condition	Controller	NaN
1	56433096	1/3/2023 12:23	1/3/2023 19:17	DOT	Department of Transportation	Traffic Signal Condition	Ped Multiple Lamps	NaN
2	56433618	1/3/2023 20:45	1/3/2023 21:05	DOT	Department of Transportation	Traffic Signal Condition	LED Pedestrian Unit	NaN
3	56434268	1/3/2023 11:50	1/3/2023 16:45	DOT	Department of Transportation	Traffic Signal Condition	Controller	NaN
4	56730815	2/7/2023 0:23	2/7/2023 1:17	DOT	Department of Transportation	Traffic Signal Condition	Controller	NaN
5	56731052	2/6/2023 9:54	2/7/2023 2:50	DOT	Department of Transportation	Traffic Signal Condition	LED Lense	NaN
6	56731500	2/6/2023 19:19	2/7/2023 2:25	DOT	Department of Transportation	Traffic Signal Condition	Ped Lamp	NaN
7	56738254	2/6/2023 13:41	2/8/2023 12:04	DOT	Department of Transportation	Traffic Signal Condition	Veh Signal Visor	NaN
8	56403066	12/30/2022 17:43	1/2/2023 10:25	DOT	Department of Transportation	Traffic Signal Condition	LED Lense	NaN
9	56412323	12/31/2022 23:47	1/3/2023 16:05	DOT	Department of Transportation	Traffic Signal Condition	Controller	NaN

Vehicle collisions data sample:

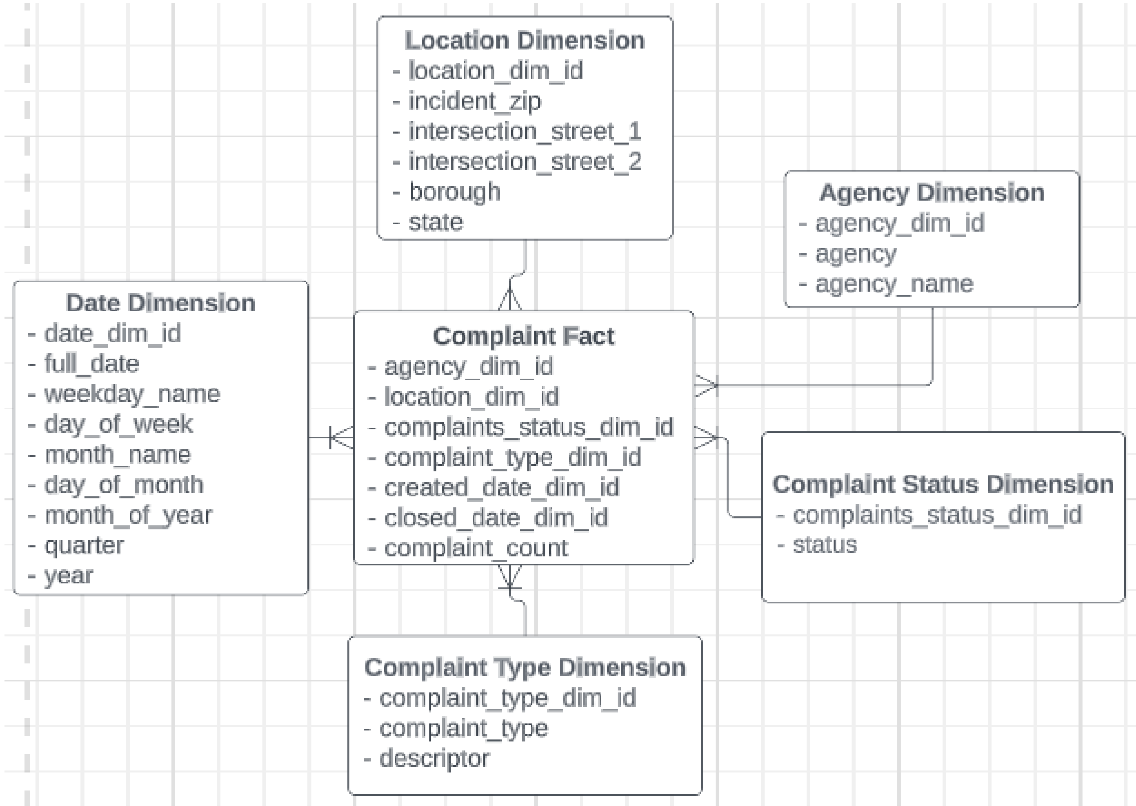
	CRASH DATE	CRASH TIME	BOROUGH	ZIP CODE	LATITUDE	LONGITUDE	LOCATION	ON STREET NAME
0	1/1/2019	18:41	NaN	NaN	NaN	NaN	NaN	VERRAZANO BRIDGE UPPER
1	1/1/2019	1:30	NaN	NaN	NaN	NaN	NaN	NARROWS ROAD NORTH
2	1/1/2019	23:55	NaN	NaN	NaN	NaN	NaN	MARTIN LUTHER KING JR
3	1/1/2019	0:20	QUEENS	11377.0	40.743137	-73.915855	(40.743137, -73.915855)	ROOSEVELT AVENUE
4	1/1/2019	3:30	QUEENS	11103.0	40.759020	-73.913450	(40.75902, -73.91345)	NaN
5	1/1/2019	14:18	BROOKLYN	11220.0	40.640415	-74.011620	(40.640415, -74.01162)	6 AVENUE
6	1/1/2019	12:00	NaN	NaN	40.878483	-73.861630	(40.878483, -73.86163)	EAST 213 STREET
7	1/1/2019	2:30	BROOKLYN	11211.0	40.703434	-73.960350	(40.703434, -73.96035)	WILLIAMSBURG STREET WEST
8	1/1/2019	2:00	NaN	NaN	40.722683	-73.819700	(40.722683, -73.8197)	MAIN STREET
9	1/1/2019	1:00	NaN	NaN	40.688920	-73.999150	(40.68892, -73.99915)	BROOKLYN QUEENS EXPRESSWAY

Initial list of Key Performance Indicators (KPI)

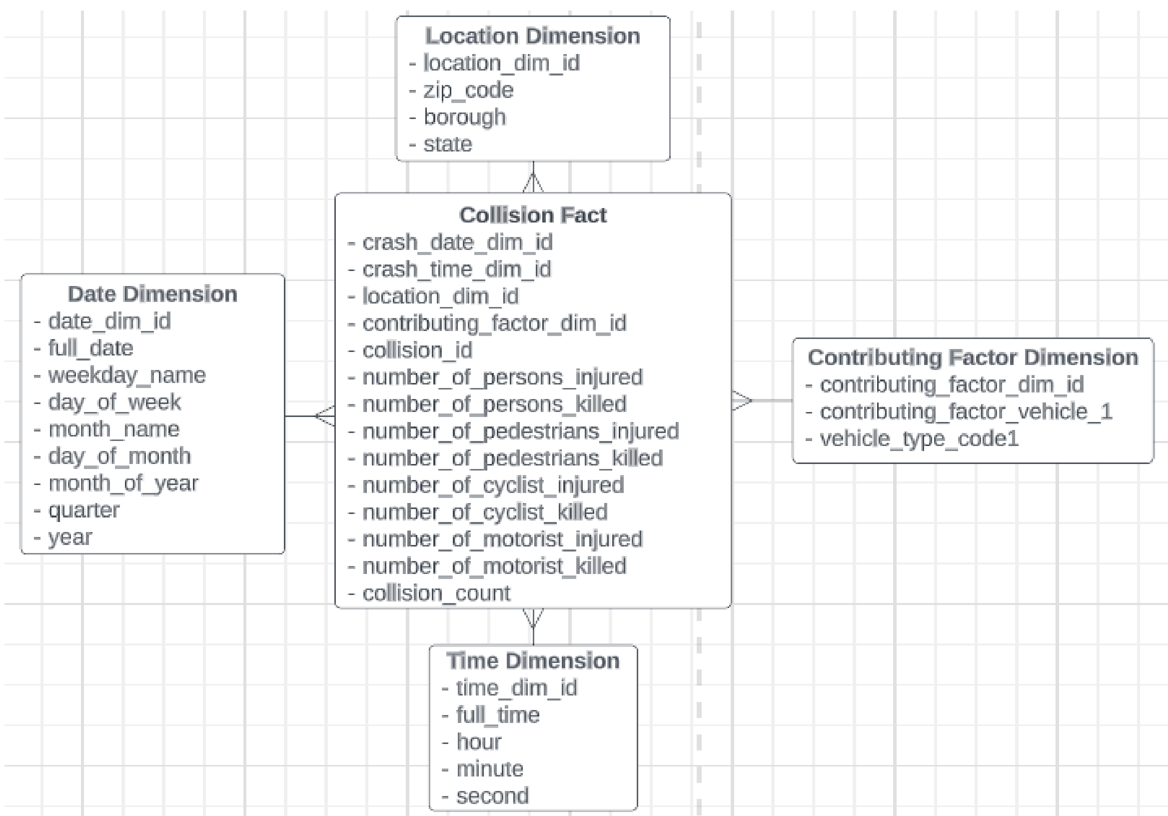
KPIs:
- number of vehicle collisions per number of traffic signal complaints
- number of traffic signal complaints vs number of vehicle collisions per zip code
- number of traffic signal complaints per zip code
- number of vehicle collisions per zip code
- number of traffic signal complaints per month
- number of vehicle collisions per month
- number of accidents reported per specific day (weekdays vs weekends)
- number of vehicle collisions per borough
- number of traffic complaints per hour (day vs night)

Dimensional Model Diagrams

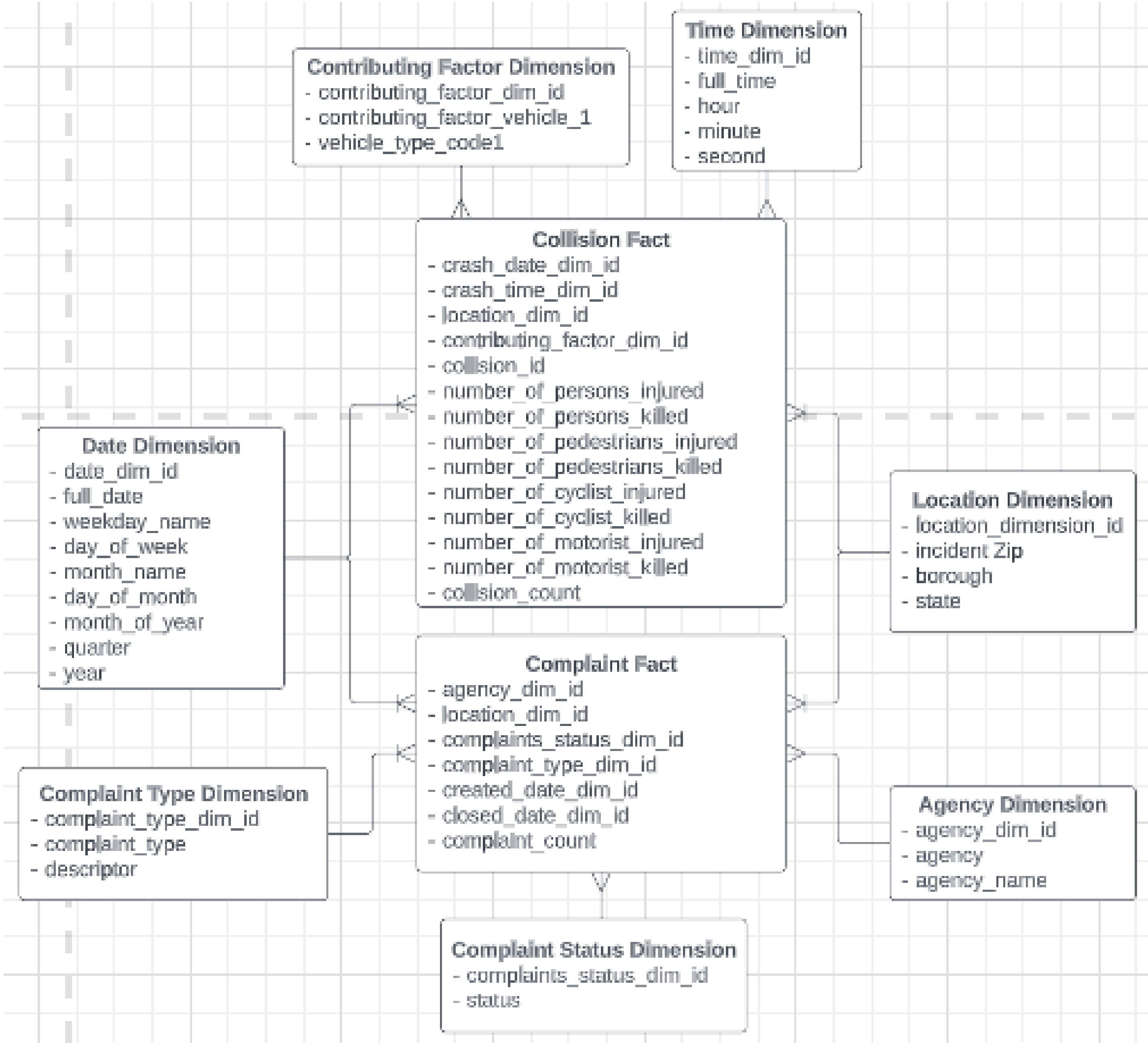
Finalized 311 complaints dimensional model:



Finalized Vehicle collision dimensional model:



Integrated Data Warehouse model:



ETL Processes

For this part of the project, we decided to write our ETL code in Python. All the codes can be found in this GitHub repo.
Note: Most of the ETL codes/functions were adapted from here.

1) First, we extracted the 311 complaints and vehicle collision data using the sodapy API.
See Appendix A for the code to extract the data or Jupyter Notebook.

2) Next, we curated a couple of different functions in order to create the dimension tables; these include

- `def load_csv_data_file(logging, file_name, df)`
- `def transform_data(logging, columns, df)`
- `def create_bigquery_client(logging)`
- `def upload_bigquery_table(logging, bqclient, table_path, write_disposition, df)`
- `def bigquery_table_exists(bqclient, table_path)`
- `def query_bigquery_table(logging, table_path, bqclient, surrogate_key)`
- `def add_surrogate_key(df, dimension_name, offset=1)`
- `def add_update_date(df, current_date)`
- `def add_update_timestamp(df)`
- `def build_new_table(logging, bqclient, dimension_table_path, dimension_name, df)`
- `def insert_existing_table(logging, bqclient, dimension_table_path, dimension_name, surrogate_key, df)`

We used these functions to create the dimension tables for both the 311 complaints and the vehicle collision data.
See Appendix B or Jupyter Notebook for the code to create the dimension tables.

3) Then we defined two new functions `def generate_time_dimension(start, end)` and `def generate_date_dimension(start, end)` as well as using the ones defined in the previous step to create the date and time dimension.
We used these functions to create a date dimension for both 311 complaints and vehicle collision data. We also created the time dimension for the vehicle collision data.
See Appendix C or Jupyter Notebook for the code to create the date and time dimensions.

4) Last but not least, we used the functions defined in step 2 and also defined 3 new functions to create the fact table for both datasets. The 3 new functions are `def dimension_lookup(logging, dimension_name, lookup_columns, df)`, `def date_dimension_lookup(logging, dimension_name, lookup_column, df)`, and `def time_dimension_lookup(logging, dimension_name, lookup_column, df)`. These functions allowed us to create fact tables for both datasets.
311 Complaint fact table
This Python code is responsible for the process that reads CSV data files for traffic signal complaints from multiple years and transforms the data by performing lookups for various dimensions. It then aggregates the data by surrogate keys and loads it into a BigQuery fact table. The ETL process is executed for each year in the range 2019-2023.

Vehicle Collision fact table
This Python code loads motor vehicle collision data for years 2019-2023 from CSV files, performs data cleansing and transformation tasks such performing dimension lookups, and aggregates the data at the daily snapshot grain. It then checks if the target table exists in BigQuery, and either load all the data into a new table or performs incremental load if the table already exists. The final result is a fact table in BigQuery with aggregated data on motor vehicle collisions.
See Appendix D or Jupyter Notebook for the code to create the fact tables.

Final Dimensional Schema

The following screenshots show the final dimensional schema in Google BigQuery for each dimension after running the ETL pipeline.

311 complaints dimension tables

Agency dimension:

Row	agency_dim_id	agency	agency_name	update_timestamp
1	1	DOT	Department of Transportation	2023-04-29 22:09:13 UTC

Complaint type dimension:

Row	complaint_type_dim_id	complaint_type	descriptor	update_timestamp
1	1	Traffic Signal Condition	Controller	2023-04-29 23:09:26 UTC
2	2	Traffic Signal Condition	Veh Signal Head	2023-04-29 23:09:26 UTC
3	3	Traffic Signal Condition	Underground	2023-04-29 23:09:26 UTC
4	4	Traffic Signal Condition	Cable	2023-04-29 23:09:26 UTC
5	5	Traffic Signal Condition	Ped Multiple Lamps	2023-04-29 23:09:26 UTC

Complaint Status dimension:

Row	complaints_status_dim_id	status	update_timestamp
1	1	Closed	2023-04-29 22:09:59 UTC
2	2	Open	2023-04-29 22:09:59 UTC
3	3	Assigned	2023-04-29 22:09:59 UTC
4	4	Pending	2023-04-29 22:10:09 UTC

Location dimension:

Row	location_dim_id	incident_zip	intersection_street_1	intersection_street_2	borough	state	update_timestamp
1	22413	null	CLOVE RD	NARROWS RD NORTH	null	NY	2023-04-29 22:16:06 UTC
2	22421	null	CLOVE RD	HIP CENTER	null	NY	2023-04-29 22:16:06 UTC
3	22425	null	DECKER AVE	BARRETT AVE	null	NY	2023-04-29 22:16:06 UTC
4	22403	null	FOREST AVE	GOETHALS RD NORTH	null	NY	2023-04-29 22:16:06 UTC
5	22406	null	HYLAN BLVD	NARROWS RD SOUTH	null	NY	2023-04-29 22:16:06 UTC

Date dimension:

Row	date_dim_id	full_date	weekday_name	day_of_week	month_name	day_of_month	month_of_year	quarter	year	update_timestamp
1	4	2019-01-04 00:00:00 UTC	Friday	5	January	04	01	1	2019	2023-04-29 23:26:36 UTC
2	11	2019-01-11 00:00:00 UTC	Friday	5	January	11	01	1	2019	2023-04-29 23:26:36 UTC
3	18	2019-01-18 00:00:00 UTC	Friday	5	January	18	01	1	2019	2023-04-29 23:26:36 UTC
4	25	2019-01-25 00:00:00 UTC	Friday	5	January	25	01	1	2019	2023-04-29 23:26:36 UTC
5	368	2020-01-03 00:00:00 UTC	Friday	5	January	03	01	1	2020	2023-04-29 23:26:36 UTC

311 complaints fact table:

Row	agency_dim_id	location_dim_id	complaints_status_dim_id	complaint_type_dim_id	created_date_dim_id	closed_date_dim_id	complaint_count
1	1	29	1	1	256	256.0	1
2	1	258	1	1	256	256.0	1
3	1	563	1	1	256	256.0	1
4	1	667	1	1	256	256.0	1
5	1	1150	1	1	256	256.0	1

Vehicle collision dimension tables

Contributing factor dimension:

Row	contributing_factor_dim_id	contributing_factor_vehicle_1	vehicle_type_code1	update_timestamp
1	3452	*null*	*null*	2023-05-03 23:37:57 UTC
2	3456	*null*	Station Wagon/Sport Utility Vehicle	2023-05-03 23:37:57 UTC
3	3474	*null*	Sedan	2023-05-03 23:37:57 UTC
4	3480	*null*	Box Truck	2023-05-03 23:37:57 UTC
5	3490	*null*	Pick-up Truck	2023-05-03 23:37:57 UTC

Location dimension:

Row	location_dim_id	zip_code	borough	state	update_timestamp
1	2	*null*	*null*	NY	2023-05-03 23:36:39 UTC
2	228	*null*	*null*	NY	2023-05-03 23:36:49 UTC
3	232	*null*	*null*	NY	2023-05-03 23:36:58 UTC
4	236	*null*	*null*	NY	2023-05-03 23:37:07 UTC
5	240	*null*	*null*	NY	2023-05-03 23:37:15 UTC

Date dimension:

Row	date_dim_id	full_date	weekday_name	day_of_week	month_name	day_of_month	month_of_year	quarter	year	update_timestamp
1	4	2019-01-04 00:00:00 UTC	Friday	5	January	04	01	1	2019	2023-05-03 23:22:12 UTC
2	11	2019-01-11 00:00:00 UTC	Friday	5	January	11	01	1	2019	2023-05-03 23:22:12 UTC
3	18	2019-01-18 00:00:00 UTC	Friday	5	January	18	01	1	2019	2023-05-03 23:22:12 UTC
4	25	2019-01-25 00:00:00 UTC	Friday	5	January	25	01	1	2019	2023-05-03 23:22:12 UTC
5	368	2020-01-03 00:00:00 UTC	Friday	5	January	03	01	1	2020	2023-05-03 23:22:12 UTC

Time dimension:

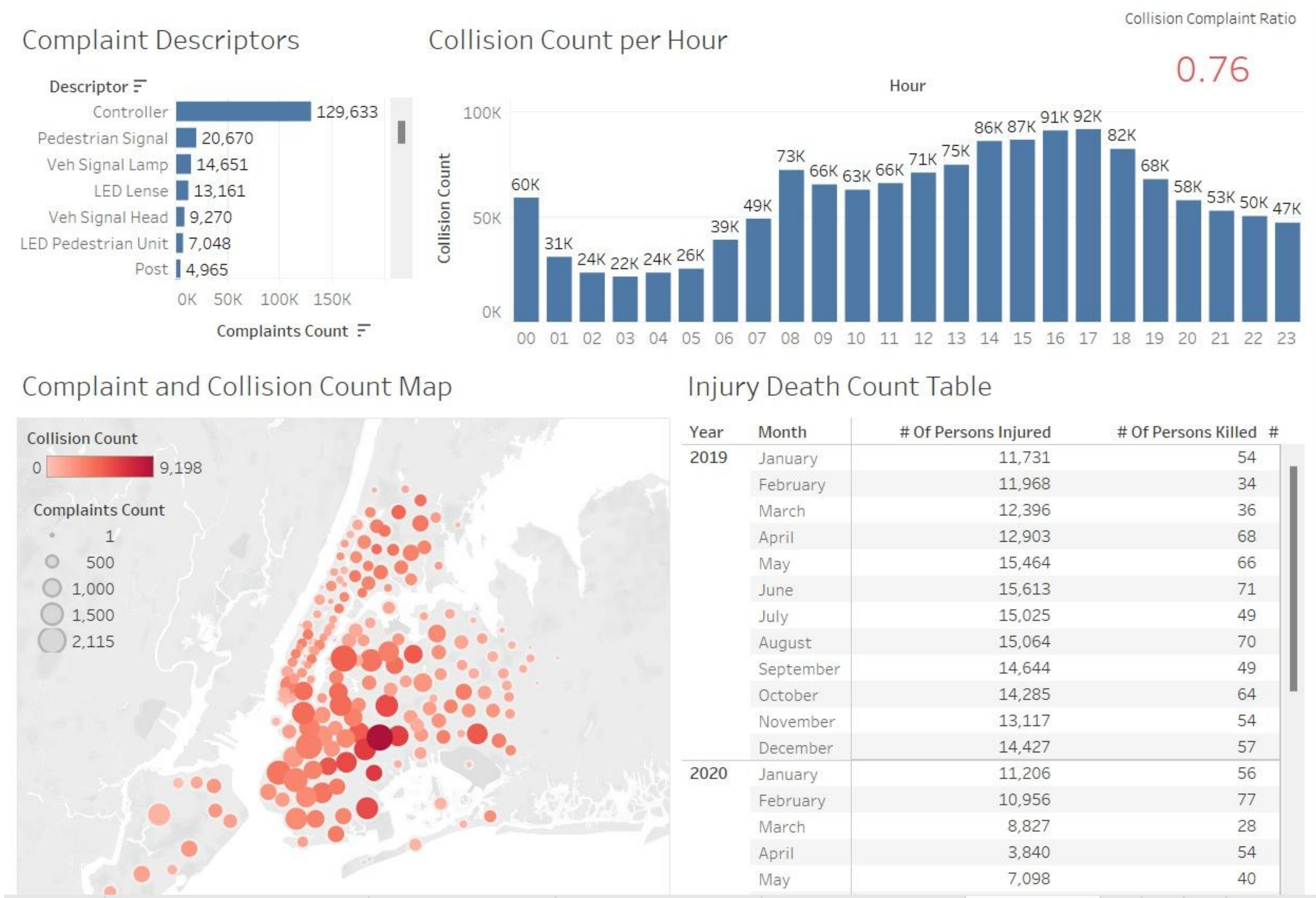
Row	time_dim_id	full_time	hour	minute	second	update_timestamp
1	1	00:00:00	00	00	00	2023-05-03 23:28:05 UTC
2	2	00:00:01	00	00	01	2023-05-03 23:28:05 UTC
3	3	00:00:02	00	00	02	2023-05-03 23:28:05 UTC
4	4	00:00:03	00	00	03	2023-05-03 23:28:05 UTC
5	5	00:00:04	00	00	04	2023-05-03 23:28:05 UTC

Vehicle collision fact table:

Row	crash_date_dim_id	crash_time_dim_id	location_dim_id	contributing_factor_dim_id	collision_id	number_of_persons_injured	number_of_pedestrians_injured	number_of_pedestrian
1	1	1	2	56	4060779	0	0	
2	1	1	2	109	4060899	0	0	
3	1	1	2	123	4056364	0	0	
4	1	1	2	1760	4056364	0	0	
5	1	1	2	2432	4056364	0	0	

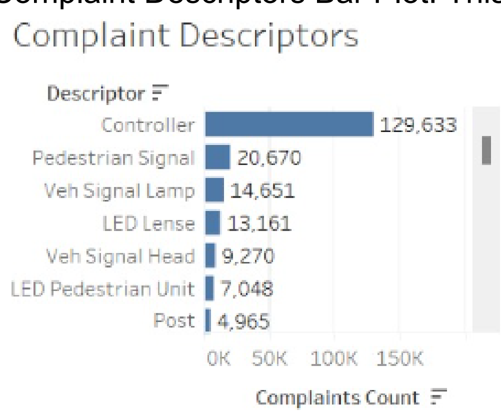
KPI Visualizations and Dashboard

Tableau Dashboard:

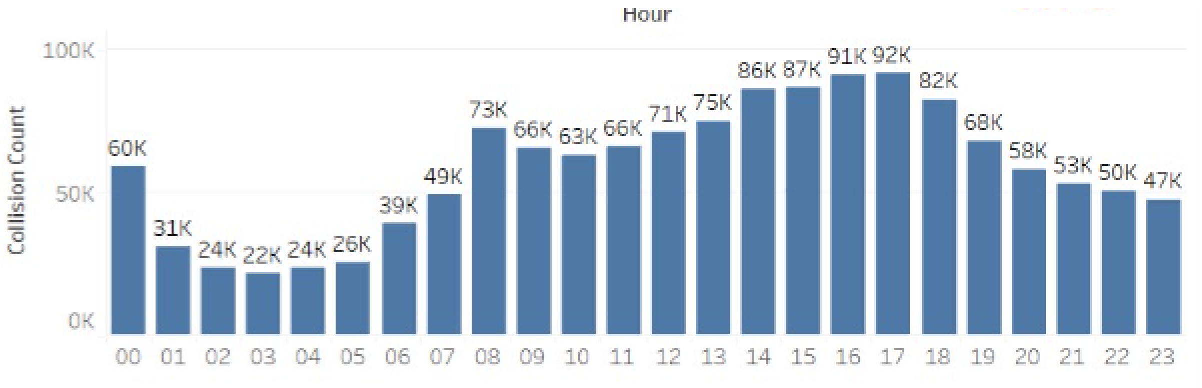


Collision Complaints Ratio: This number shows how many collisions occur per complaint.

Complaint Descriptors Bar Plot: This plot shows the most common type of traffic light complaints.



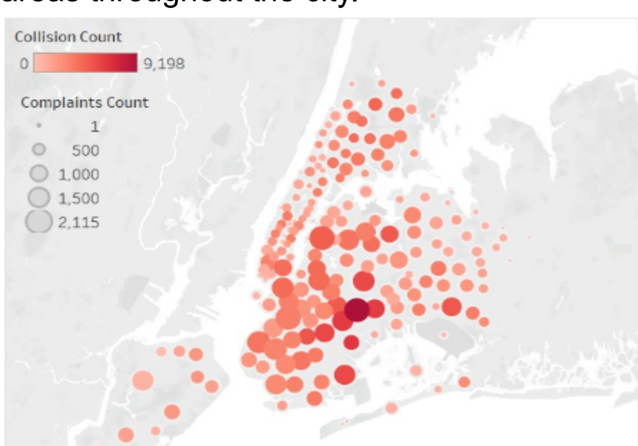
Collision Count per Hour: This number shows how many collisions occur each hour of the day.



Injury Death Count Table: This table shows how many injuries and deaths occur per month from collisions.

Year	Month	# Of Persons Injured	# Of Persons Killed	#
2019	January	11,731	54	
	February	11,968	34	
	March	12,396	36	
	April	12,903	68	
	May	15,464	66	
	June	15,613	71	
	July	15,025	49	
	August	15,064	70	
	September	14,644	49	
	October	14,285	64	
	November	13,117	54	
	December	14,427	57	
2020	January	11,206	56	
	February	10,956	77	
	March	8,827	28	
	April	3,840	54	
	May	7,098	40	

Complaint and Collision map: This Bubble map shows the density of collisions and complaints in areas throughout the city.



Descriptions of Tools Used

The tools we used:
LucidChart: A web-based diagramming tool that allows users to create, collaborate on, and share flowcharts, diagrams, and visualizations.
Python: A programming language is known for its simplicity, readability, and versatility in various domains.
Google BigQuery: A fully managed, serverless data warehouse on the cloud that enables scalable analysis over petabytes of data.
Tableau: Tableau is a data visualization and business intelligence tool that enables users to explore and analyze data through interactive dashboards and reports.

Conclusion

Software and database tools:

- Google BigQuery: We hosted our dimension tables here and it allowed us to cooperate in the cloud environment which eliminated some of the compatibility concerns for using different operating systems.
- Python and Jupyter Notebook: For coding and testing out the ETL pipeline.
- SQL: Used to query the data inside Google BigQuery and to create data visualizations.
- Tableau: To create the data visualizations for the KPIs.

Group's experience with the project:
The hardest part of the project was to write the ETL pipeline in Python and the easiest part is creating the dimensional models. We learned a lot about Google BigQuery, service accounts, and APIs. If we have to do it all over again, we would probably include more traffic-related complaints instead of just traffic light complaints so that we can analyze more information related to vehicle collisions.

The proposed benefits can be realized by the new system: Yes, the data warehouse can be used to find the most common type of traffic light complaints and the hot spots in NYC. The dashboard can make it easier to spot areas in need of personnel and speed up the decision-making process.

References

ETL code from professor: https://github.com/professorholowczak/Data_Warehousing/
311 Service Requests data:
https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9
Motor Vehicle Collisions - Crashes:
https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95

Appendix

A) Extract data

```
data_url = 'data.cityofnewyork.us'
app_token = nyc_opendata_key.api_app_token
client = Socrata(data_url, app_token)
client.timeout = 240

for x in range(2019, 2024):
    # get data
    start = 0
    chunk_size = 2000
    results = []
    where_clause = f"complaint_type LIKE 'Traffic Signal Condition%' AND
date_extract_y(created_date)={x}"
    data_set = 'erm2-nwe9'
    record_count = client.get(data_set, where=where_clause, select='COUNT(*)')
    print(f'Fetching Traffic Signal Condition complaints data from {x}')
    while True:
        results.extend(client.get(data_set, where=where_clause, offset=start,
limit=chunk_size))
        start += chunk_size
        if (start > int(record_count[0]['COUNT'])):
            break
    # export data to csv
    df = pd.DataFrame.from_records(results)
    df.to_csv(f'data/311_bicycle_complaints_{x}.csv', index=False)

for x in range(2019, 2024):
    # get data
    start = 0
    chunk_size = 2000
    results = []
    where_clause = f"date_extract_y(crash_date)={x}"
    data_set = 'h9gi-nx95'
    record_count = client.get(data_set, where=where_clause, select='COUNT(*)')
    print(f'Fetching Motor Vehicle Collision data from {x}')
```

```
    while True:
        results.extend(client.get(data_set, where=where_clause, offset=start,
limit=chunk_size))
        start += chunk_size
        if (start > int(record_count[0]['COUNT'])):
            break
    # export data to csv
    df = pd.DataFrame.from_records(results)
    df.to_csv(f'data/motor_vehicle_collision_{x}.csv', index=False)
```

B) Create Dimension Tables

B-1) 311 complaints dimension table

```
# create dictionary and list for loops
dim_dict = {
    'location': ['zip_code', 'borough'],
    'contributing_factor': ['contributing_factor_vehicle_1', 'vehicle_type_code1']
}

for key, value in dim_dict.items():
    dimension_name = key
    surrogate_key = f'{dimension_name}_dim_id'
    business_key = f'{dimension_name}_id'

    table_name = f'{dimension_name}_dimension'
    dimension_table_path = f'{gcp_project}.{bq_dataset}.{table_name}'

    for handler in logging.root.handlers[:]:
        logging.root.removeHandler(handler)

    current_date = datetime.today().strftime('%Y%m%d')
    log_filename = '_'.join(['etl', dimension_name, current_date]) + '.log'
    logging.basicConfig(filename=log_filename, encoding='utf-8', format='%(asctime)s
%(message)s', level=logging.DEBUG)

logging.info('========================================================================')
    logging.info(f'Starting ETL Run for dimension {dimension_name} on date
{current_date}')

    columns = value

    for year in range(2019, 2024):

logging.info('========================================================================')
        if __name__ == '__main__':
            if dimension_name == 'location':
                df = pd.DataFrame
```

```
                df = load_csv_data_file(logging,
f'data/311_traffic_signal_complaints_{year}.csv', df)
                df = transform_data(logging, columns, df)
                df['state'] = 'NY'
                bqclient = create_bigquery_client(logging)
                target_table_exists = bigquery_table_exists(bqclient,
dimension_table_path)
                if not target_table_exists:
                    build_new_table(logging, bqclient, dimension_table_path,
dimension_name, df)
                if target_table_exists:
                    insert_existing_table(logging, bqclient, dimension_table_path,
dimension_name, surrogate_key, df)
                logging.shutdown()

            else:
                df = pd.DataFrame
                df = load_csv_data_file(logging,
f'data/311_traffic_signal_complaints_{year}.csv', df)
                df = transform_data(logging, columns, df)
                bqclient = create_bigquery_client(logging)
                target_table_exists = bigquery_table_exists(bqclient,
dimension_table_path)
                if not target_table_exists:
                    build_new_table(logging, bqclient, dimension_table_path,
dimension_name, df)
                if target_table_exists:
                    insert_existing_table(logging, bqclient, dimension_table_path,
dimension_name, surrogate_key, df)
                logging.shutdown()
```

B-2) Motor Vehicle Collision Data

```
# create dictionary and list for loops
dim_dict = {
    'location': ['zip_code', 'borough'],
    'contributing_factor': ['contributing_factor_vehicle_1', 'vehicle_type_code1']
}

for key, value in dim_dict.items():
    dimension_name = key
    surrogate_key = f'{dimension_name}_dim_id'
    business_key = f'{dimension_name}_id'

    table_name = f'{dimension_name}_dimension'
    dimension_table_path = f'{gcp_project}.{bq_dataset}.{table_name}'
```

```
    for handler in logging.root.handlers[:]:
        logging.root.removeHandler(handler)

    current_date = datetime.today().strftime('%Y%m%d')
    log_filename = '_'.join(['etl', dimension_name, current_date]) + '.log'
    logging.basicConfig(filename=log_filename, encoding='utf-8', format='%(asctime)s
%(message)s', level=logging.DEBUG)

logging.info('=========================================================================')
    logging.info(f'Starting ETL Run for dimension {dimension_name} on date
{current_date}')

    columns = value

    for year in range(2019, 2024):

logging.info('=========================================================================')
        if __name__ == '__main__':
            if dimension_name == 'location':
                df = pd.DataFrame
                df = load_csv_data_file(logging,
f'data/motor_vehicle_collision_{year}.csv', df)
                df = transform_data(logging, columns, df)
                df['state'] = 'NY'
                bqclient = create_bigquery_client(logging)
                target_table_exists = bigquery_table_exists(bqclient,
dimension_table_path)
                if not target_table_exists:
                    build_new_table(logging, bqclient, dimension_table_path,
dimension_name, df)
                if target_table_exists:
                    insert_existing_table(logging, bqclient, dimension_table_path,
dimension_name, surrogate_key, df)
                logging.shutdown()

            else:
                df = pd.DataFrame
                df = load_csv_data_file(logging,
f'data/motor_vehicle_collision_{year}.csv', df)
                df = transform_data(logging, columns, df)
                bqclient = create_bigquery_client(logging)
                target_table_exists = bigquery_table_exists(bqclient,
dimension_table_path)
                if not target_table_exists:
                    build_new_table(logging, bqclient, dimension_table_path,
dimension_name, df)
```

```
                if target_table_exists:
                    insert_existing_table(logging, bqclient, dimension_table_path,
dimension_name, surrogate_key, df)
                logging.shutdown()
```

C) Create Date and Time Dimensions

C-1) Date dimension

```
# date dimension 311
if __name__ == "__main__":
    df = pd.DataFrame
    df = generate_date_dimension(start='2019-01-01', end='2023-12-31')
    bqclient = create_bigquery_client(logging)
    target_table_exists = bigquery_table_exists(bqclient, dimension_table_path  )
    if not target_table_exists:
        build_new_table(logging, bqclient, dimension_table_path, dimension_name, df)
    if target_table_exists:
        print("Date dimension already exists. Will not overwrite it")
    logging.shutdown()

# date dimension vehicle collision
if __name__ == "__main__":
    df = pd.DataFrame
    df = generate_date_dimension(start='2019-01-01', end='2023-12-31')
    bqclient = create_bigquery_client(logging)
    target_table_exists = bigquery_table_exists(bqclient, dimension_table_path  )
    if not target_table_exists:
        build_new_table(logging, bqclient, dimension_table_path, dimension_name, df)
    if target_table_exists:
        print("Date dimension already exists. Will not overwrite it")
    logging.shutdown()
```

C-2) Time dimension for vehicle collision data

```
# time dimension
if __name__ == "__main__":
    df = pd.DataFrame
    df = generate_time_dimension(start='2019-01-01 00:00:00', end='2019-01-01 23:59:59')
    bqclient = create_bigquery_client(logging)
    target_table_exists = bigquery_table_exists(bqclient, dimension_table_path  )
    if not target_table_exists:
        build_new_table(logging, bqclient, dimension_table_path, dimension_name, df)
    if target_table_exists:
        print("Time dimension already exists. Will not overwrite it")
    logging.shutdown()
```

D) Create Fact Tables

D-1) Fact table for 311 complaints data

```
for year in range(2019, 2024):

logging.info('==========================================================================')
    if __name__ == '__main__':
        df = pd.DataFrame

        bqclient = create_bigquery_client(logging)

        df = load_csv_data_file(logging, f'data/311_traffic_signal_complaints_{year}.csv',
df)

        df = dimension_lookup(logging, dimension_name='agency', lookup_columns=['agency',
'agency_name'], df=df)
        df = dimension_lookup(logging, dimension_name='location',
lookup_columns=['incident_zip', 'intersection_street_1', 'intersection_street_2',
'borough'], df=df)
        df = dimension_lookup(logging, dimension_name='complaints_status',
lookup_columns=['status'], df=df)
        df = dimension_lookup(logging, dimension_name='complaint_type',
lookup_columns=['complaint_type', 'descriptor'], df=df)

        df = date_dimension_lookup(logging, dimension_name='date',
lookup_column='created_date', df=df)
        df = date_dimension_lookup(logging, dimension_name='date',
lookup_column='closed_date', df=df)


surrogate_keys=['agency_dim_id','location_dim_id','complaints_status_dim_id','complaint_ty
pe_dim_id','created_date_dim_id','closed_date_dim_id']
        df = df[surrogate_keys]

        # Add complaint count (for daily snapshot grain)
        df['complaint_count'] = 1
        df = df.groupby(surrogate_keys)['complaint_count'].agg('count').reset_index()

        # See if the target table exists
        target_table_exists = bigquery_table_exists(fact_table_path, bqclient)
        # If the target table does not exist, load all of the data into a new table
        if not target_table_exists:
            build_new_table(logging, bqclient, fact_table_path, df)
        # If the target table exists, then perform an incremental load
        if target_table_exists:
            insert_existing_table(logging, bqclient, fact_table_path, df)
        logging.shutdown()
```

D-2) Fact table for vehicle collision data

```
for year in range(2019, 2024):

logging.info('==========================================================================')
    if __name__ == '__main__':
        df = pd.DataFrame
        bqclient = create_bigquery_client(logging)
```

```
        df = load_csv_data_file(logging, f'data/motor_vehicle_collision_{year}.csv', df)

        df = dimension_lookup(logging, dimension_name='location',
lookup_columns=['zip_code', 'borough'], df=df)
        df = dimension_lookup(logging, dimension_name='contributing_factor',
lookup_columns=['contributing_factor_vehicle_1','vehicle_type_code1'], df=df)

        df = date_dimension_lookup(logging, dimension_name='date',
lookup_column='crash_date', df=df)
        df = time_dimension_lookup(logging, dimension_name='time',
lookup_column='crash_time', df=df)


surrogate_keys=['crash_date_dim_id','crash_time_dim_id','location_dim_id','contributing_fa
ctor_dim_id','collision_id','number_of_persons_injured','number_of_pedestrians_injured','n
umber_of_pedestrians_killed','number_of_cyclist_injured','number_of_cyclist_killed','numbe
r_of_motorist_injured','number_of_motorist_killed']
        df = df[surrogate_keys]

        # Add collision count (for daily snapshot grain)
        df['collision_count'] = 1
        df = df.groupby(surrogate_keys)['collision_count'].agg('count').reset_index()

        # See if the target table exists
        target_table_exists = bigquery_table_exists(fact_table_path, bqclient)
        # If the target table does not exist, load all of the data into a new table
        if not target_table_exists:
            build_new_table(logging, bqclient, fact_table_path, df)
        # If the target table exists, then perform an incremental load
        if target_table_exists:
            insert_existing_table(logging, bqclient, fact_table_path, df)
        logging.shutdown()
```