Business Memo

Toronto is one of the major cities in Canada, and its larger population increases the risk of traffic collisions. Through the utilization of historical data, our team has performed various analysis to develop a multi-purpose dashboard. Our dashboard visualizes historical data to portray various trends of traffic collisions and also provides the audience with our forecasting analysis and predictions. Our analytics project consists of five processes as listed below:

1. Data Processing and Aggregation

We utilized public historical data obtained from two primary sources to analyze traffic collisions in relation to weather conditions. The traffic collision data was sourced from the Toronto Police Service, which provides a comprehensive dataset through the City of Toronto Open Data portal. Additionally, weather-related data was retrieved from the Government of Canada's climate database, offering detailed daily climate measurements across various stations. These datasets were meticulously merged based on the date of each event, allowing for a robust dataset that combines traffic and weather variables.

2. Feature Engineering

To enhance our understanding of traffic collisions and to facilitate deeper analysis of potential causative factors, we transformed and enriched the raw data through several feature engineering steps:

1. Seasonal Analysis: Recognizing the potential impact of seasonal variations on collision rates, we classified dates into meteorological seasons. This categorization helps identify seasonal trends in traffic collisions and examines whether certain weather conditions linked to specific seasons influence the frequency or severity of accidents.
2. Day Type Classification: To determine if the risk of collisions varies between weekdays and weekends, we categorized each date accordingly. This distinction is crucial for understanding traffic patterns and planning city traffic management and emergency response strategies more effectively.
3. Rush Hour Identification: We introduced a binary feature indicating whether a collision occurred during typical rush hours, which is known for higher traffic volumes and potentially higher accident rates. This feature is essential for assessing the impact of traffic density on collision occurrences.

These engineered features are expected to provide valuable insights into the dynamics of traffic collisions and support the development of targeted strategies to enhance road safety.

3. Classification (Logistic Regression, KNN, and Decision Tree)

To further understand the classification of the predicted value of traffic collision data, we built the model with 3 different methods to test the results before making a business decision on the appropriate one. The one with the best fit is selected based on the criteria of accuracy, precision, and recall. Upon thorough analysis on Logistic Regression, KNN, and Decision Tree, the data shows that KNN is the best fit to use to classify our business case: traffic collision data. The results obtained can be proposed to the City of Toronto to aid in ambulance usage, and only dispatch ambulances to collisions that are injury involved.

4. Forecasting (Time series/Arima)

To better predict traffic accident trends in Toronto, we used timeseries analysis and ARIMA models. In the timeseries analysis, we visualized data from 2014 to 2024 using Single Exponential Smoothing, Holt's Linear Trend Method, and Holt-Winters Triple Exponential Smoothing, showing long-term trends and fluctuations. Holt's Linear Trend Method performed the best, successfully capturing a linear downward trend. Additionally, we used the ARIMA model for monthly forecasting. By converting date formats, analyzing trends and seasonality, and using the ADF test to ensure data stationarity, our final forecast shows that 2024 data nearly returns to pre-pandemic levels, with similar patterns expected in the coming years.

5. *Dashboard*

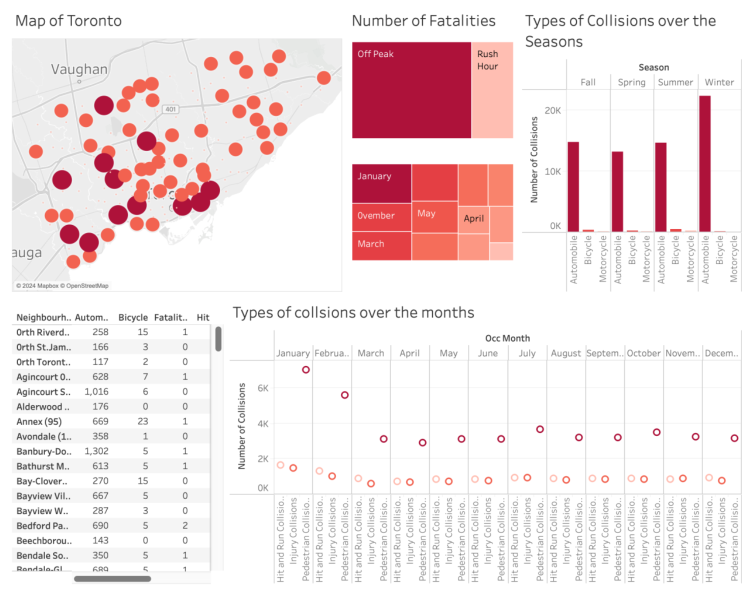
We have presented our story through two different dashboards: one displaying our analysis of the historical data and one with our predictive analysis. The dashboards are easy to navigate and interpret. The historical data dashboard consists of a map of Toronto, neighbourhood data table, two fatalities tree maps, bar graph displaying number of collisions per season, and a side-by-side circle graph displaying the number of collisions over months. The map allows the user to hover over each location to identify the number of automobile, bicycle, and motorcycle collisions per neighbourhood. The size and colour of the points on the maps correlate with the number number of fatalities in the neighborhood. The data table allows the users to look at the number of various types of collisions per neighbourhood and they may filter out the data by their column of choice. For example, filtering by automobile collisions allows us to see that West Humber-Clairville has the highest number of collisions. The first tree-map displays the number of fatalities based on rush hour. We can see that there were more fatalities during off peak hours. The second tree-map displays the number of fatalities per month. January, November, and March had the highest number of fatalities. The bar graph displays the number of automobile, bicycle, and motorcycle collisions per season. Automobile collisions were highest for all seasons, second being bicycle, and last motorcycle. The side-by-side circle graph depicts the number of hit and run, injury, and pedestrian collisions over the months. January and February have the highest number of collisions. The second dashboard depicts the graphical results obtained from our forecasting (time series and Arima) models as stated in process number four.

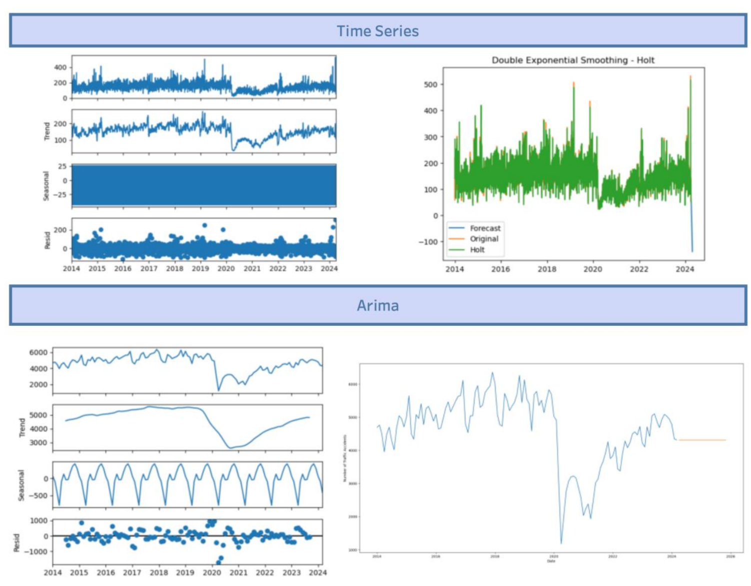
Overall, our project goal is to help the police force identify which neighbourhoods have the highest number of traffic collisions and the correlations between when the different types of collisions occur. Through this information, Toronto Police can gain a deeper understanding of what and where proactive measures need to be implemented. As mentioned, the results from our KNN classification may be utilized to increase efficiency in ambulance usage by only dispatching ambulances to injury collisions. Furthermore, our forecasting results show a downward trend in upcoming traffic collisions, and that the upcoming collision cases resemble 2024 and pre-pandemic times. This analysis could potentially help the City of Toronto decide how much funds should be allocated for collision aid. Lastly, our historical dashboard is able to provide Toronto Police with information on which areas in the city are likely to have collisions, when these collisions are likely to occur, and what type of collisions they may be. This data can help them stay proactive in such areas to control traffic.

This concludes the business memo. For further information, please refer to our dashboard images and Technical Report below.

Thank you,
Team New York
MMA 2025B

DASHBOARD





TECHNICAL REPORT

1. Data processing and aggregation

Merging data sets:

Weather Dataset and merge into traffic data ¶

In [36]:

```python
file_paths = [
    "en_climate_daily_ON_6158359_2015_P1D.csv",
    "en_climate_daily_ON_6158359_2016_P1D.csv",
    "en_climate_daily_ON_6158359_2017_P1D.csv",
    "en_climate_daily_ON_6158359_2018_P1D.csv",
    "en_climate_daily_ON_6158359_2019_P1D.csv",
    "en_climate_daily_ON_6158359_2020_P1D.csv",
    "en_climate_daily_ON_6158359_2021_P1D.csv",
    "en_climate_daily_ON_6158359_2022_P1D.csv",
    "en_climate_daily_ON_6158359_2023_P1D.csv",
    "en_climate_daily_ON_6158359_2024_P1D.csv"
]

# Load each dataset to check the structure
data_samples = [pd.read_csv(file) for file in file_paths]

# Display the first few rows and the info of the first dataset as a sample
data_samples[0].head(), data_samples[0].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Longitude (x)            365 non-null    float64
 1   Latitude (y)             365 non-null    float64
 2   Station Name             365 non-null    object
 3   Climate ID               365 non-null    int64
 4   Date/Time                365 non-null    object
 5   Year                     365 non-null    int64
 6   Month                    365 non-null    int64
 7   Day                      365 non-null    int64
 8   Data Quality             0 non-null      float64
 9   Max Temp (°C)            359 non-null    float64
 10  Max Temp Flag            5 non-null      object
 11  Min Temp (°C)            360 non-null    float64
 12  Min Temp Flag            5 non-null      object
 13  Mean Temp (°C)           358 non-null    float64
 14  Mean Temp Flag           5 non-null      object
 15  Heat Deg Days (°C)       358 non-null    float64
 16  Heat Deg Days Flag       5 non-null      object
 17  Cool Deg Days (°C)       358 non-null    float64
 18  Cool Deg Days Flag       5 non-null      object
 19  Total Rain (mm)          0 non-null      float64
 20  Total Rain Flag          361 non-null    object
 21  Total Snow (cm)          0 non-null      float64
 22  Total Snow Flag          361 non-null    object
 23  Total Precip (mm)        355 non-null    float64
 24  Total Precip Flag        6 non-null      object
 25  Snow on Grnd (cm)        0 non-null      float64
 26  Snow on Grnd Flag        0 non-null      float64
 27  Dir of Max Gust (10s deg)  274 non-null  float64
 28  Dir of Max Gust Flag     0 non-null      float64
 29  Spd of Max Gust (km/h)   361 non-null    object
 30  Spd of Max Gust Flag     0 non-null      float64
dtypes: float64(16), int64(4), object(11)
memory usage: 88.5+ KB
```

Out[36]:

```
(   Longitude (x)  Latitude (y)        Station Name  Climate ID   Date/Time  \
 0          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-01
 1          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-02
 2          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-03
 3          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-04
 4          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-05

    Year  Month  Day  Data Quality  Max Temp (°C)  ...  Total Snow (cm)  \
 0  2015      1    1           NaN           -0.4  ...              NaN
 1  2015      1    2           NaN            0.3  ...              NaN
 2  2015      1    3           NaN            2.9  ...              NaN
 3  2015      1    4           NaN            5.0  ...              NaN
 4  2015      1    5           NaN           -4.5  ...              NaN

    Total Snow Flag Total Precip (mm)  Total Precip Flag Snow on Grnd (cm)  \
 0               M               0.0                NaN               NaN
 1               M               0.0                NaN               NaN
 2               M               6.3                NaN               NaN
 3               M               2.8                NaN               NaN
 4               M               0.0                NaN               NaN

    Snow on Grnd Flag Dir of Max Gust (10s deg)  Dir of Max Gust Flag  \
 0                NaN                      23.0                   NaN
 1                NaN                      27.0                   NaN
 2                NaN                      10.0                   NaN
 3                NaN                      28.0                   NaN
 4                NaN                      26.0                   NaN

   Spd of Max Gust (km/h)  Spd of Max Gust Flag
 0                     69                   NaN
 1                     50                   NaN
 2                     48                   NaN
 3                     61                   NaN
 4                     69                   NaN

 [5 rows x 31 columns],
 None)
```

In [34]:

```python
# Define the seasons
def get_season(month):
    if month in ['December', 'January', 'February']:
        return 'Winter'
    elif month in ['March', 'April', 'May']:
        return 'Spring'
    elif month in ['June', 'July', 'August']:
        return 'Summer'
    elif month in ['September', 'October', 'November']:
        return 'Fall'
    else:
        return None

# Define weekdays/weekends
def weekday_or_weekend(day):
    if day in ['Saturday', 'Sunday']:
        return 'Weekend'
    else:
        return 'Weekday'

# Define rush hour (assuming typical rush hours are from 7-9 AM and 4-6 PM)
def is_rush_hour(hour):
    if (7 <= hour <= 9) or (16 <= hour <= 18):
        return 'Rush Hour'
    else:
        return 'Off Peak'

# Apply these functions to the dataset
data['Season'] = data['OCC_MONTH'].apply(get_season)
data['Day Type'] = data['OCC_DOW'].apply(weekday_or_weekend)
data['Rush Hour'] = data['OCC_HOUR'].apply(is_rush_hour)

# Display the new columns
data.head()
```

Out[34]:

	X	Y	OBJECTID	EVENT_UNIQUE_ID	OCC_DATE	OCC_MONTH	OCC_DOW	OCC_YEAR	OCC_HOUR	DIVISION	...	LONG_WG
0	-8.856427e+06	5.418247e+06	1	GO-20148000005	2014/01/01 05:00:00+00	January	Wednesday	2014	13	D23	...	-79.55
1	-8.825577e+06	5.432191e+06	2	GO-20148000085	2014/01/01 05:00:00+00	January	Wednesday	2014	19	D42	...	-79.28
13	-8.813438e+06	5.431796e+06	14	GO-20148000059	2014/01/01 05:00:00+00	January	Wednesday	2014	15	D43	...	-79.17
14	-8.824062e+06	5.439593e+06	15	GO-20148000023	2014/01/01 05:00:00+00	January	Wednesday	2014	22	D42	...	-79.26
15	-8.839217e+06	5.412544e+06	16	GO-20148000651	2014/01/01 05:00:00+00	January	Wednesday	2014	3	D14	...	-79.40

5 rows × 26 columns

In [35]:

```python
# Define the file path for the new CSV
output_file_path = 'Modified_Traffic_Collisions.csv'

# Save the dataset to CSV
data.to_csv(output_file_path, index=False)

# Return the path for download
output_file_path
```

Out[35]: 'Modified_Traffic_Collisions.csv'

```
None)
```

In [37]:
```python
# Combine all datasets into one
combined_data = pd.concat(data_samples, ignore_index=True)

# Display the shape and first few rows of the combined dataset
combined_data.shape, combined_data.head()
```

Out[37]:
```
((3653, 31),
    Longitude (x)  Latitude (y)        Station Name  Climate ID   Date/Time  \
 0          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-01
 1          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-02
 2          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-03
 3          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-04
 4          -79.4         43.63  TORONTO CITY CENTRE     6158359  2015-01-05

    Year  Month  Day  Data Quality  Max Temp (°C)  ...  Total Snow (cm)  \
 0  2015      1    1           NaN           -0.4  ...              NaN
 1  2015      1    2           NaN            0.3  ...              NaN
 2  2015      1    3           NaN            2.9  ...              NaN
 3  2015      1    4           NaN            5.0  ...              NaN
 4  2015      1    5           NaN           -4.5  ...              NaN

    Total Snow Flag Total Precip (mm)  Total Precip Flag Snow on Grnd (cm)  \
 0               M               0.0                NaN               NaN
 1               M               0.0                NaN               NaN
 2               M               6.3                NaN               NaN
 3               M               2.8                NaN               NaN
 4               M               0.0                NaN               NaN

    Snow on Grnd Flag Dir of Max Gust (10s deg)  Dir of Max Gust Flag  \
 0                NaN                       23.0                  NaN
 1                NaN                       27.0                  NaN
 2                NaN                       10.0                  NaN
 3                NaN                       28.0                  NaN
 4                NaN                       26.0                  NaN

   Spd of Max Gust (km/h)  Spd of Max Gust Flag
 0                     69                   NaN
 1                     50                   NaN
 2                     48                   NaN
 3                     61                   NaN
 4                     69                   NaN

 [5 rows x 31 columns])
```

In [38]:
```python
# Save the combined dataset to a CSV file
output_path = "Combined_Climate_Data_2015_2024.csv"
combined_data.to_csv(output_path, index=False)
```

In [41]:
```python
weather_data=combined_data
traffic_data=data
```

In [42]:
```python
# Convert the 'OCC_DATE' to datetime format and extract the date
traffic_data['OCC_DATE'] = pd.to_datetime(traffic_data['OCC_DATE']).dt.date

# Convert the 'Date/Time' to datetime format
weather_data['Date/Time'] = pd.to_datetime(weather_data['Date/Time']).dt.date

# Merge the datasets on the date columns
merged_data = pd.merge(traffic_data, weather_data, left_on='OCC_DATE', right_on='Date/Time', how='left')

# Display the first few rows of the merged dataset and the columns
merged_data_info = (merged_data.head(), merged_data.columns)
merged_data_info
```

Out[42]:
```
(             X             Y  OBJECTID EVENT_UNIQUE_ID    OCC_DATE OCC_MONTH  \
 0 -8.856427e+06  5.418247e+06         1  GO-20148000005  2014-01-01   January
 1 -8.825577e+06  5.432191e+06         2  GO-20148000085  2014-01-01   January
 2 -8.813438e+06  5.431796e+06        14  GO-20148000059  2014-01-01   January
 3 -8.824062e+06  5.439593e+06        15  GO-20148000023  2014-01-01   January
 4 -8.839217e+06  5.412544e+06        16  GO-20148000651  2014-01-01   January

      OCC_DOW  OCC_YEAR  OCC_HOUR DIVISION  ...  Total Snow (cm)  \
 0  Wednesday      2014        13      D23  ...              NaN
 1  Wednesday      2014        19      D42  ...              NaN
 2  Wednesday      2014        15      D43  ...              NaN
 3  Wednesday      2014        22      D42  ...              NaN
 4  Wednesday      2014         3      D14  ...              NaN

   Total Snow Flag Total Precip (mm) Total Precip Flag Snow on Grnd (cm)  \
 0             NaN               NaN               NaN               NaN
 1             NaN               NaN               NaN               NaN
 2             NaN               NaN               NaN               NaN
 3             NaN               NaN               NaN               NaN
 4             NaN               NaN               NaN               NaN

   Snow on Grnd Flag  Dir of Max Gust (10s deg)  Dir of Max Gust Flag  \
 0               NaN                        NaN                   NaN
 1               NaN                        NaN                   NaN
 2               NaN                        NaN                   NaN
 3               NaN                        NaN                   NaN
 4               NaN                        NaN                   NaN

   Spd of Max Gust (km/h) Spd of Max Gust Flag
 0                    NaN                  NaN
 1                    NaN                  NaN
 2                    NaN                  NaN
 3                    NaN                  NaN
 4                    NaN                  NaN

 [5 rows x 57 columns],
 Index(['X', 'Y', 'OBJECTID', 'EVENT_UNIQUE_ID', 'OCC_DATE', 'OCC_MONTH',
        'OCC_DOW', 'OCC_YEAR', 'OCC_HOUR', 'DIVISION', 'FATALITIES',
        'INJURY_COLLISIONS', 'FTR_COLLISIONS', 'PD_COLLISIONS', 'HOOD_158',
        'NEIGHBOURHOOD_158', 'LONG_WGS84', 'LAT_WGS84', 'AUTOMOBILE',
        'MOTORCYCLE', 'PASSENGER', 'BICYCLE', 'PEDESTRIAN', 'Season',
        'Day Type', 'Rush Hour', 'Longitude (x)', 'Latitude (y)',
        'Station Name', 'Climate ID', 'Date/Time', 'Year', 'Month', 'Day',
        'Data Quality', 'Max Temp (°C)', 'Max Temp Flag', 'Min Temp (°C)',
        'Min Temp Flag', 'Mean Temp (°C)', 'Mean Temp Flag',
        'Heat Deg Days (°C)', 'Heat Deg Days Flag', 'Cool Deg Days (°C)',
        'Cool Deg Days Flag', 'Total Rain (mm)', 'Total Rain Flag',
        'Total Snow (cm)', 'Total Snow Flag', 'Total Precip (mm)',
        'Total Precip Flag', 'Snow on Grnd (cm)', 'Snow on Grnd Flag',
        'Dir of Max Gust (10s deg)', 'Dir of Max Gust Flag',
        'Spd of Max Gust (km/h)', 'Spd of Max Gust Flag'],
       dtype='object'))
```

Cleaning and validating data:

Data cleaning and validation

In [28]:
```python
import pandas as pd
data = pd.read_csv('Traffic_Collisions_Open_Data_(ASR-T-TBL-001).csv')
```

In [29]:
```python
data.head()
```

Out[29]:

	X	Y	OBJECTID	EVENT_UNIQUE_ID	OCC_DATE	OCC_MONTH	OCC_DOW	OCC_YEAR	OCC_HOUR	DIVISION	...	PD_COLLIS
0	-8.856427e+06	5.418247e+06	1	GO-20148000005	2014/01/01 05:00:00+00	January	Wednesday	2014	13	D23	...	
1	-8.825577e+06	5.432191e+06	2	GO-20148000085	2014/01/01 05:00:00+00	January	Wednesday	2014	19	D42	...	
2	6.327780e-09	5.664924e-09	3	GO-20141260499	2014/01/01 05:00:00+00	January	Wednesday	2014	2	NSA	...	
3	6.327780e-09	5.664924e-09	4	GO-20141260663	2014/01/01 05:00:00+00	January	Wednesday	2014	3	NSA	...	
4	6.327780e-09	5.664924e-09	5	GO-20141261162	2014/01/01 05:00:00+00	January	Wednesday	2014	5	NSA	...	

5 rows × 23 columns

In [30]:
```python
# Check for missing values in each column
missing_values = data.isnull().sum()

# Check for duplicated entries
duplicated_entries = data.duplicated().sum()

missing_values, duplicated_entries
```

Out[30]:
```
(X                    0
 Y                    0
 OBJECTID             0
 EVENT_UNIQUE_ID      0
 OCC_DATE             0
 OCC_MONTH            0
 OCC_DOW              0
 OCC_YEAR             0
 OCC_HOUR             0
 DIVISION             0
 FATALITIES           0
 INJURY_COLLISIONS    4
 FTR_COLLISIONS       4
 PD_COLLISIONS        4
 HOOD_158             0
 NEIGHBOURHOOD_158    0
 LONG_WGS84           0
 LAT_WGS84            0
 AUTOMOBILE           4
 MOTORCYCLE           4
 PASSENGER            4
 BICYCLE              4
 PEDESTRIAN           4
 dtype: int64,
 0)
```

In [31]:
```python
data=data.dropna()
```

In [33]:
```python
# Filter the data to exclude rows where 'HOOD_158' equals 'NSA'
data = data[data['HOOD_158'] != 'NSA']

# Display the first few rows of the filtered data to verify
print(data.head())
```

```
                X             Y  OBJECTID EVENT_UNIQUE_ID  \
0  -8.856427e+06  5.418247e+06         1  GO-20148000005
1  -8.825577e+06  5.432191e+06         2  GO-20148000085
13 -8.813438e+06  5.431796e+06        14  GO-20148000059
14 -8.824062e+06  5.439593e+06        15  GO-20148000023
15 -8.839217e+06  5.412544e+06        16  GO-20148000651

                  OCC_DATE OCC_MONTH    OCC_DOW  OCC_YEAR  OCC_HOUR DIVISION  \
0   2014/01/01 05:00:00+00   January  Wednesday      2014        13      D23
1   2014/01/01 05:00:00+00   January  Wednesday      2014        19      D42
13  2014/01/01 05:00:00+00   January  Wednesday      2014        15      D43
14  2014/01/01 05:00:00+00   January  Wednesday      2014        22      D42
15  2014/01/01 05:00:00+00   January  Wednesday      2014         3      D14

    ... PD_COLLISIONS HOOD_158                  NEIGHBOURHOOD_158  \
0   ...           YES        6   Kingsview Village-The Westway (6)
1   ...            NO      128  Agincourt South-Malvern West (128)
13  ...           YES      133        Centennial Scarborough (133)
14  ...           YES      130                      Milliken (130)
15  ...           YES       78           Kensington-Chinatown (78)

    LONG_WGS84  LAT_WGS84 AUTOMOBILE  MOTORCYCLE  PASSENGER BICYCLE PEDESTRIAN
0   -79.558639  43.694246        YES          NO         NO      NO         NO
1   -79.281506  43.784746        YES          NO         NO      NO         NO
13  -79.172461  43.782185        YES          NO         NO      NO         NO
14  -79.267901  43.832727        YES          NO         NO      NO         NO
15  -79.404033  43.657190        YES          NO         NO      NO         NO

[5 rows x 23 columns]
```

In [43]:
```
# Calculate the number of missing values in each column of the merged dataset
missing_values = merged_data.isnull().sum()
missing_values
```

Out[43]:
```
X                               0
Y                               0
OBJECTID                        0
EVENT_UNIQUE_ID                 0
OCC_DATE                        0
OCC_MONTH                       0
OCC_DOW                         0
OCC_YEAR                        0
OCC_HOUR                        0
DIVISION                        0
FATALITIES                      0
INJURY_COLLISIONS               0
FTR_COLLISIONS                  0
PD_COLLISIONS                   0
HOOD_158                        0
NEIGHBOURHOOD_158               0
LONG_WGS84                      0
LAT_WGS84                       0
AUTOMOBILE                      0
MOTORCYCLE                      0
PASSENGER                       0
BICYCLE                         0
PEDESTRIAN                      0
Season                          0
Day Type                        0
Rush Hour                       0
Longitude (x)               54801
Latitude (y)                54801
Station Name                54801
Climate ID                  54801
Date/Time                   54801
Year                        54801
Month                       54801
Day                         54801
Data Quality               568985
Max Temp (°C)               84828
Max Temp Flag              557850
Min Temp (°C)               83865
Min Temp Flag              557850
Mean Temp (°C)              84991
Mean Temp Flag             557850
Heat Deg Days (°C)          84991
Heat Deg Days Flag         557850
Cool Deg Days (°C)          84991
Cool Deg Days Flag         557850
Total Rain (mm)            568985
Total Rain Flag            346031
Total Snow (cm)            568985
Total Snow Flag            346031
Total Precip (mm)           83590
Total Precip Flag          559792
Snow on Grnd (cm)          568985
Snow on Grnd Flag          568985
Dir of Max Gust (10s deg)  172251
Dir of Max Gust Flag       520729
Spd of Max Gust (km/h)     122653
Spd of Max Gust Flag       520729
dtype: int64
```

In [45]:
```
# Drop rows with any missing values in the weather-related columns
cleaned_data = merged_data.dropna(subset=['Date/Time', 'Max Temp (°C)', 'Min Temp (°C)', 'Mean Temp (°C)', 'Total Precip (mm)

# Check how many rows are left after dropping missing values
remaining_rows = cleaned_data.shape[0]
remaining_rows
```

Out[45]: 479027

In [47]:
```
# Save the cleaned dataset to a new CSV file
cleaned_file_path = 'Cleaned_Traffic_Weather_Data.csv'
cleaned_data.to_csv(cleaned_file_path, index=False)

cleaned_file_path
```

Out[47]: 'Cleaned_Traffic_Weather_Data.csv'

In [ ]:

Data Aggregation for Time Series Model:

Aggregation for time series model

```
In [ ]:

In [48]:
import numpy as np
# Aggregating by hour
hourly_counts = data.groupby('OCC_HOUR').size().reset_index(name='Count')

# Parsing the date and extracting daily counts
data['OCC_DATE'] = pd.to_datetime(data['OCC_DATE']).dt.date
daily_counts = data.groupby('OCC_DATE').size().reset_index(name='Count')

# Aggregating by month and year
monthly_counts = data.groupby(['OCC_YEAR', 'OCC_MONTH']).size().reset_index(name='Count')

# Aggregating by year
yearly_counts = data.groupby('OCC_YEAR').size().reset_index(name='Count')

# Saving to separate CSV files (optional)
hourly_counts.to_csv('hourly_counts.csv', index=False)
daily_counts.to_csv('daily_counts.csv', index=False)
monthly_counts.to_csv('monthly_counts.csv', index=False)
yearly_counts.to_csv('yearly_counts.csv', index=False)

In [ ]:

In [49]:
(hourly_counts.head(), daily_counts.head(), monthly_counts.head(), yearly_counts.head())

Out[49]:
(   OCC_HOUR  Count
 0         0   6720
 1         1   5757
 2         2   5485
 3         3   5347
 4         4   3998,
      OCC_DATE  Count
 0  2014-01-01     58
 1  2014-01-02    141
 2  2014-01-03    222
 3  2014-01-04    139
 4  2014-01-05     94,
    OCC_YEAR OCC_MONTH  Count
 0      2014     April   3957
 1      2014    August   4017
 2      2014  December   4707
 3      2014  February   4762
 4      2014   January   4694,
    OCC_YEAR  Count
 0      2014  54801
 1      2015  60312
 2      2016  63480
 3      2017  65684
 4      2018  65215)

In [ ]:
```

2. <u>Feature Engineering</u>

1. Seasonal Analysis
   - We created a new column for ‘Season’ by grouping dates according to meteorological seasons.
2. Day Type Classification
   - We created a new column for ‘Day Type’ by categorzing each date as either ‘weekend’ or ‘weekday’.
3. Rush Hour Identification
   - We created a new column for ‘Rush Hour’ indication if a collision occurred ‘off peak hours’ or during ‘rush hour’ based on the hours data that was provided.

3. Classification

We have classified the 'Injury Collisions' from the Traffic Collisions Open Data obtained from the Toronto Police Service. We have used Logistic Regression, KNN, and Decision Tree classification methods. Here is the summary of our analysis:

Accuracy: All three models have very high train and test accuracy, with KNN having the highest train accuracy (0.9942) and Decision Tree having the highest test accuracy at 0.9929.

Precision: Logistic Regression shows the highest test precision (0.9926), while Logistic Regression has the highest train precision (0.9915).

Recall: KNN provides highest in both train recall (0.9755) and test recall (0.9678).

Confusion Matrix: The confusion matrices show the number of true positives, false positives, true negatives, and false negatives. While all models perform well, Logistic Regression has the lowest number of false positives and false negatives, suggesting better performance in identifying positive cases.

Conclusion:
While all models perform well, the KNN model strikes a good balance between precision and recall, making it a strong candidate based on the given metrics.

Hyperparameters used:

```
# Models dictionary
models = {
    'Logistic Regression': LogisticRegression(random_state=0,penalty="l2", C=1e42, solver='saga'),
    'Decision Tree': DecisionTreeClassifier(random_state=0,max_depth=7,min_samples_split=50),
    'KNN': KNeighborsClassifier()
}

# Print parameters for each model
for name, model in models.items():
    print(f"Parameters for {name}:")
    print(model.get_params())
    print()  # Adds a newline for better readability
```

```
Parameters for Logistic Regression:
{'C': 1e+42, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_ite
r': 100, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': 0, 'solver': 'saga', 'tol': 0.0001, 'verbos
e': 0, 'warm_start': False}

Parameters for Decision Tree:
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': 7, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 50, 'min_weight_fraction_leaf': 0.0, 'random_stat
e': 0, 'splitter': 'best'}

Parameters for KNN:
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p':
2, 'weights': 'uniform'}
```

Classification results:

Logistic Regression:

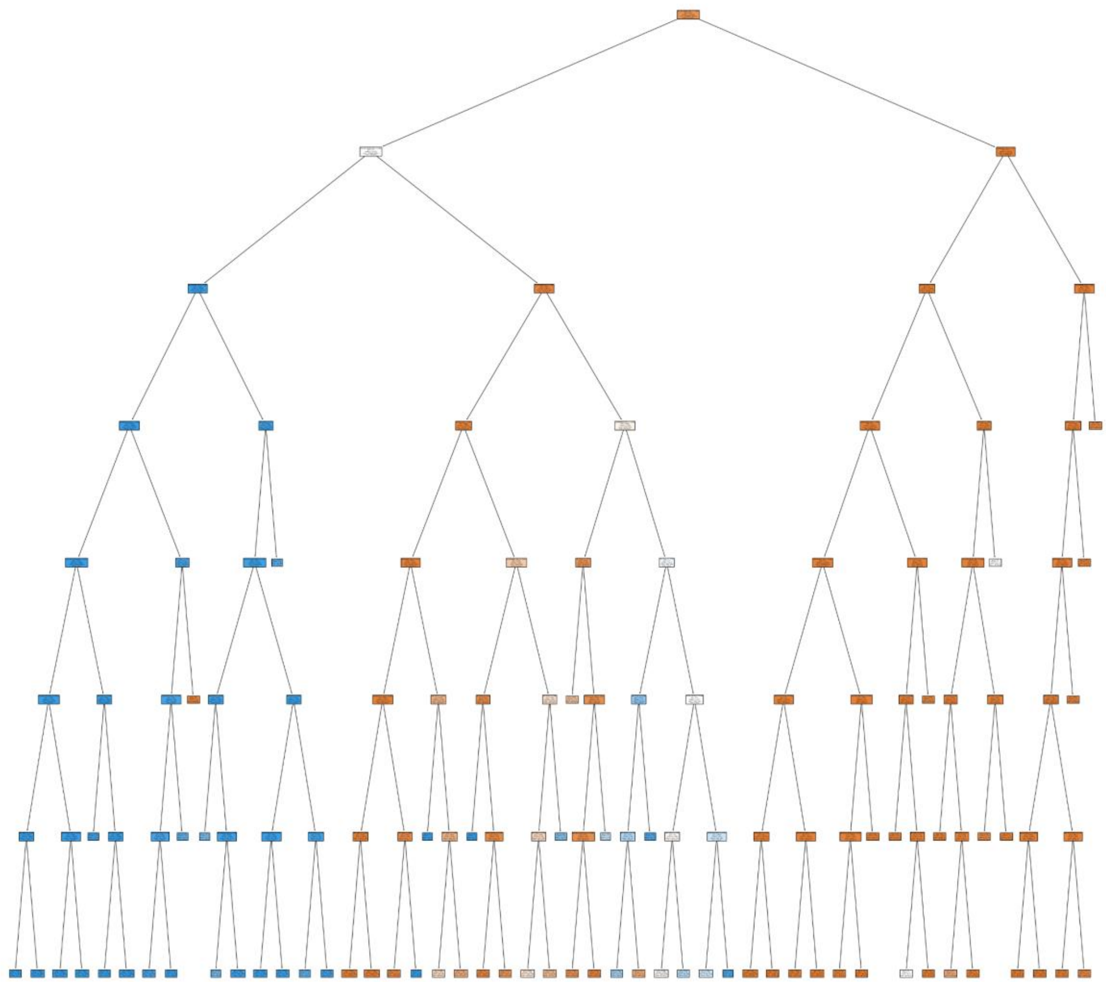
```
'Logistic Regression': {'Train Accuracy': 0.9929678273628458,
 'Test Accuracy': 0.9926517149477757,
 'Train Precision': 0.9915430595528005,
 'Test Precision': 0.9925849639546859,
 'Train Recall': 0.9577452741425028,
 'Test Recall': 0.9548246483059243,
 'Train Confusion Matrix': array([[288170,    382],
        [  1976,  44788]], dtype=int64),
 'Test Confusion Matrix': array([[123375,    144],
        [   912,  19276]], dtype=int64)},
```

KNN:

```
'KNN': {'Train Accuracy': 0.9942472175500126,
 'Test Accuracy': 0.9926238805346991,
 'Train Precision': 0.9830420823547157,
 'Test Precision': 0.9794465610587527,
 'Train Recall': 0.9755795056026003,
 'Test Recall': 0.9678026550425995,
 'Train Confusion Matrix': array([[287765,    787],
        [  1142,  45622]], dtype=int64),
 'Test Confusion Matrix': array([[123109,    410],
        [   650,  19538]], dtype=int64)}}
```

Decision Tree:

```
'Decision Tree': {'Train Accuracy': 0.9933435923129227,
 'Test Accuracy': 0.9929718106981567,
 'Train Precision': 0.9879898308056456,
 'Test Precision': 0.9884372453137734,
 'Train Recall': 0.9639893935505944,
 'Test Recall': 0.9612145829205468,
 'Train Confusion Matrix': array([[288004,    548],
        [  1684,  45080]], dtype=int64),
 'Test Confusion Matrix': array([[123292,    227],
        [   783,  19405]], dtype=int64)},
```



Different Classification Model Performance

Accuracy: The ratio of correct predictions vs all total predictions.

Logistic Regression:

- Train: 99.29%
- Test: 99.26%

Decision Tree:

- Train: 99.33%
- Test: 99.29%

KNN:

- Train: 99.42%

- Test: 99.26%

All 3 models have high accuracy for both train and test set. The accuracy score for train and test are very close indicating there is no signs of overfitting.

Precision: The ratio of correct positive prediction vs the total positive predictions.

Logistic Regression:

- Train: 99.15%
- Test: 99.25%

Decision Tree:

- Train: 98.79%
- Test: 98.84%

KNN:

- Train: 98.30%
- Test: 97.94%

Precision score is also high for all 3 models indicating that the models made correct positive predictions. Logistic Regression has the highest score shows that it made more true positive predictions than Decision Tree and KNN.

Recall: The ratio of correct positive prediction vs all true positive predictions.

Logistic Regression:

- Train: 95.77%
- Test: 95.48%

Decision Tree:

- Train: 96.39%
- Test: 96.12%

KNN:

- Train: 97.55%
- Test: 96.78%

Recall is high for all 3 models indicating that all models are good at making predictions. KNN has the highest recall score shows that KNN made more true positive predictions than Logistic Regression and Decision Tree.

Analysis:

For our business problem, KNN is recommended to be the best fit model classification method to use to classify traffic collisions to be injury or non-injury related.

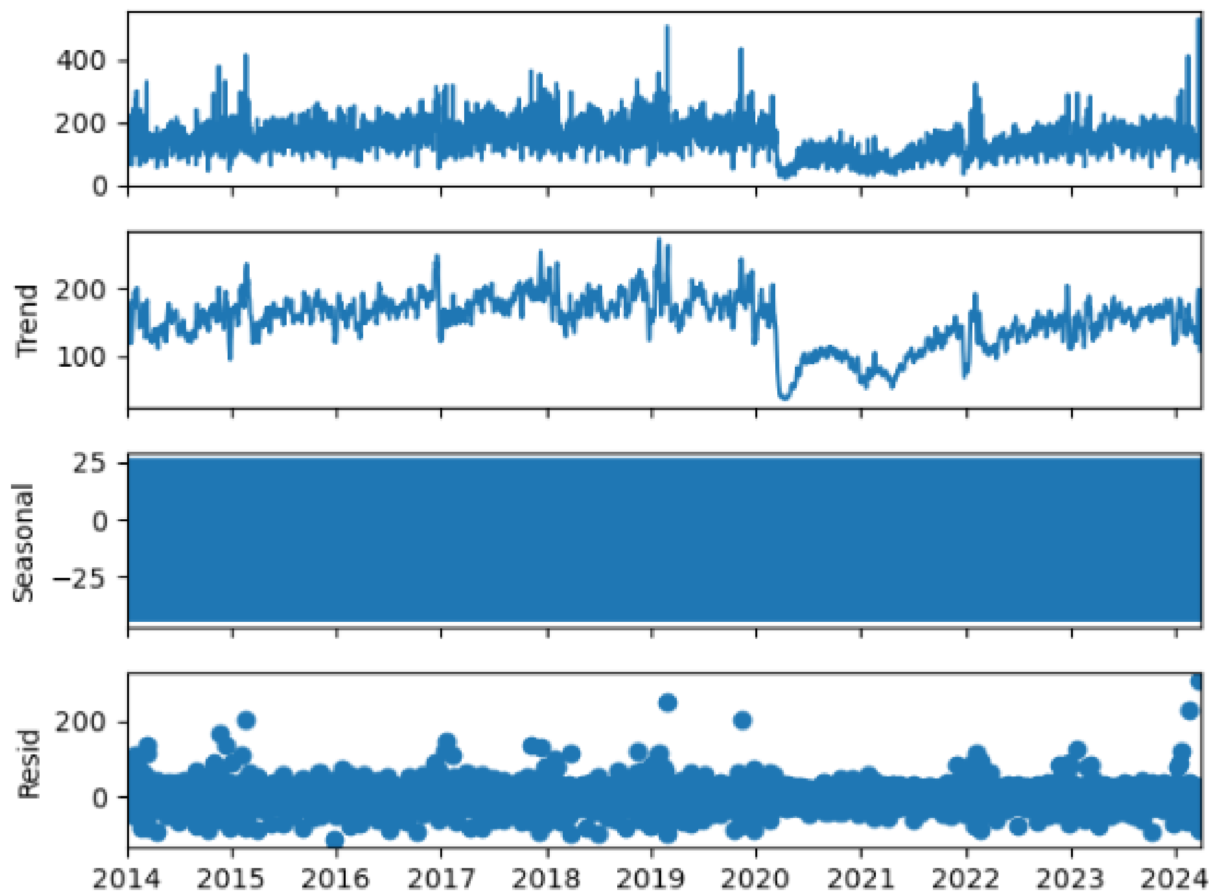
Best Model Selection

The KNN is the best model. The high accuracy on the testing data set, the balancing on precision and recall effectively. It demonstrated credibility and reliability for new, unseen data prediction.

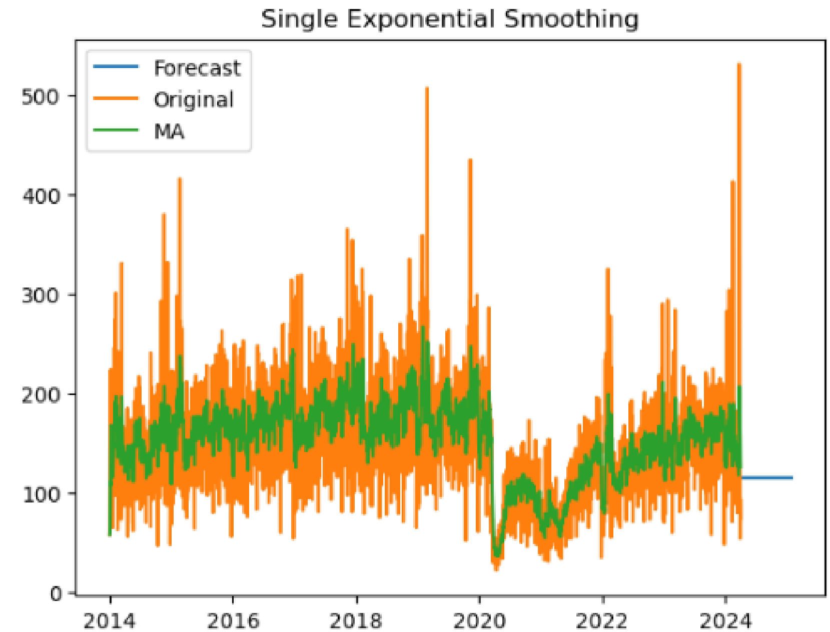
4. Forecasting

Time-series

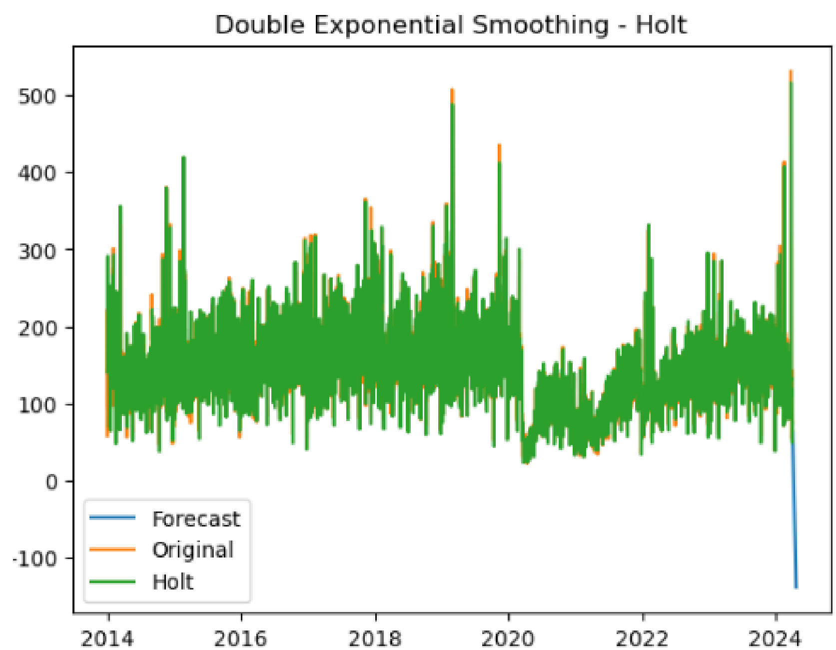
We visualized the traffic accident data from 2014 to 2024, illustrating the long-term trends and fluctuations.



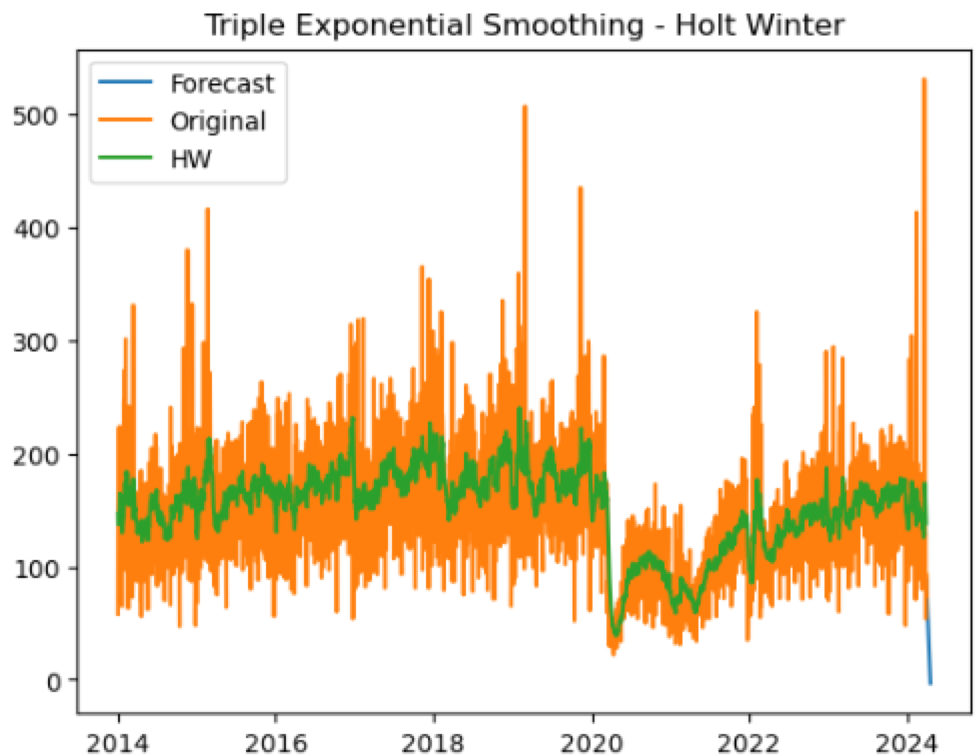
We initially employed time series methods to analyze the traffic accident data, utilizing three smoothing methods: Single Exponential Smoothing, Holt's Linear Trend Method, and Triple Exponential Smoothing (Holt-Winters). These methods were used to capture potential trends and seasonal variations in the data and to visualize the results.



This method predicts that the future number of traffic accidents will remain stable, failing to capture any trend or seasonal variations. The forecasted values appear as a horizontal line, indicating the method's limitation in reflecting changes in the data trends.



The Double Exponential method successfully captures the linear downward trend in the data, predicting a gradual decrease in future traffic accidents. The fitted values show the highest alignment with the actual data values, demonstrating the method's superior performance in capturing long-term trends.



Holt Winter method captures both trend and seasonal variations in the data. However, due to the relatively weak seasonal pattern in the dataset, its performance is slightly inferior to Holt's Linear Trend Method. The fitted values align well with the actual data but are slightly less accurate than those from the Holt method.

By comparing the forecasting results of the three smoothing methods, we found that Holt's Linear Trend Method performs the best in capturing long-term trends with the smallest prediction error. Although the Triple Exponential Smoothing method also captures trend and seasonal variations, its performance is slightly lower due to the weak seasonality in this dataset. Therefore, we recommend using Holt's Linear Trend Method for practical applications.

ARIMA Model

We set up a monthly forecasting flow for our Arima Model as opposed to daily for the time series forecasting above.

First, we converted our **Date** column to datetime.

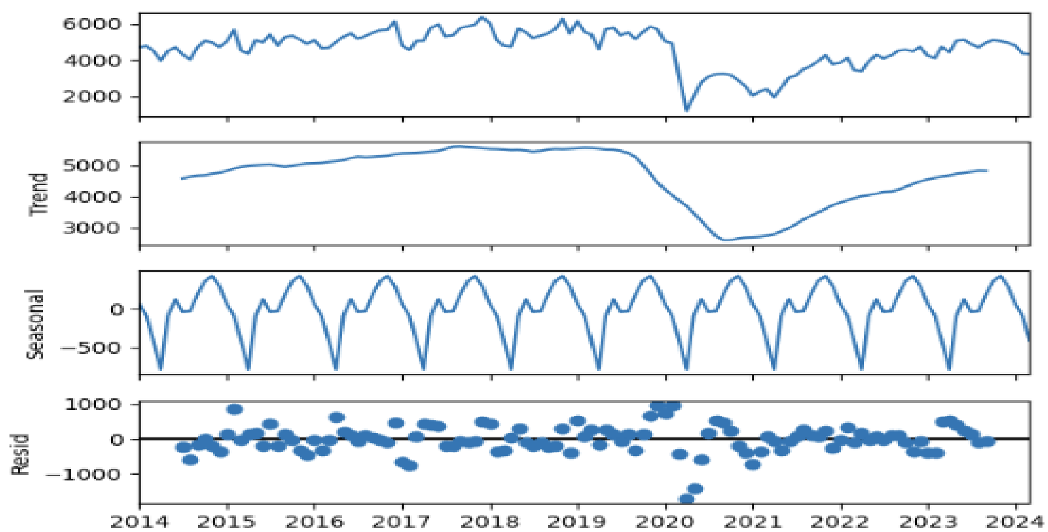
```
data['Date']=pd.to_datetime(data['Date'], infer_datetime_format=True)
data=data.set_index(['Date'])
print(data.head())
print(data.tail())
```

```
C:\Users\promi\AppData\Local\Temp\ipykernel_6800\2820800779.py:1: UserWarning: The argument 'infer_datetime_format' is deprecat
ed and will be removed in a future version. A strict version of it is now the default, see https://pandas.pydata.org/pdeps/0004
-consistent-to-datetime-parsing.html. You can safely remove this argument.
  data['Date']=pd.to_datetime(data['Date'], infer_datetime_format=True)
C:\Users\promi\AppData\Local\Temp\ipykernel_6800\2820800779.py:1: UserWarning: Could not infer format, so each element will be
parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  data['Date']=pd.to_datetime(data['Date'], infer_datetime_format=True)
```

```
            Traffic_Accidents
Date
2014-04-01               3957
2014-08-01               4017
2014-12-01               4707
2014-02-01               4762
2014-01-01               4694
            Traffic_Accidents
Date
2023-10-01               5087
2023-09-01               4939
2024-02-01               4347
2024-01-01               4785
2024-03-01               4316
```

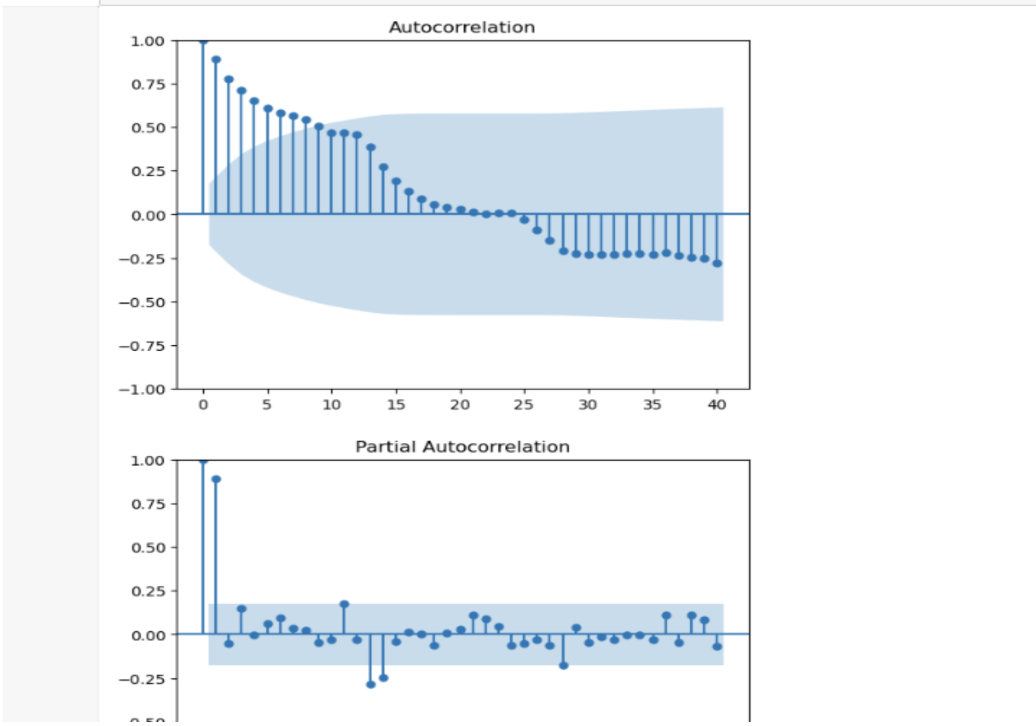
Then we took a look at our Trend, Seasonality and Randomness.

```
from statsmodels.tsa.seasonal import seasonal_decompose
series = data
result = seasonal_decompose(series, model='additive')
result.plot()
plt.show()
```



ADF Statistic: -2.355987 p-value: 0.154546- With a P value greater than the significance level we will fail to reject the null hypothesis. This implies no stationerity exists.

Next we analyze the Autocorrelation/PACF to identify lags in the series.



The above helps dig into a better understanding of Auto Regression and Moving Average. 1.00 is where we have the most significance.

Next Arima;

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1860.740, Time=5.90 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1860.967, Time=0.26 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1862.969, Time=0.56 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1862.967, Time=1.12 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1858.972, Time=0.21 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 8.338 seconds
```

Out[17]:

SARIMAX Results

Dep. Variable:	y	No. Observations:	123
Model:	SARIMAX(0, 1, 0)	Log Likelihood	-928.486
Date:	Wed, 12 Jun 2024	AIC	1858.972
Time:	23:57:06	BIC	1861.776
Sample:	01-01-2014	HQIC	1860.111
	- 03-01-2024		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
sigma2	2.378e+05	2.04e+04	11.649	0.000	1.98e+05	2.78e+05

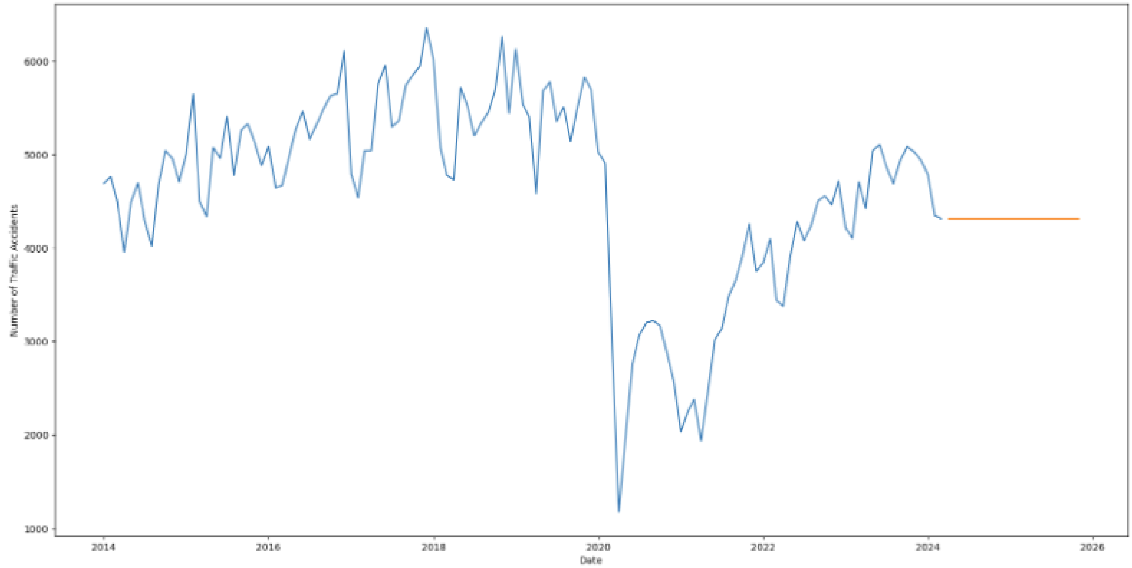
Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	48.31
Prob(Q):	0.95	Prob(JB):	0.00
Heteroskedasticity (H):	0.55	Skew:	-0.97
Prob(H) (two-sided):	0.06	Kurtosis:	5.39

The P,D,Q helps us understand the differentiation, the lag order and the moving average. Leveraging the stepwise method, we are able to view statistics that help us understand our model. Overall, the model indicates that our series might be non stationary.

Finally the forecast;

In [18]:
```
forecasts = arima_model.predict(20)
plt.figure(figsize=(20,10))
plt.xlabel("Date")
plt.ylabel("Number of Traffic Accidents")
plt.plot(data)
plt.plot(forecasts)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x22a726b8f90>]
```

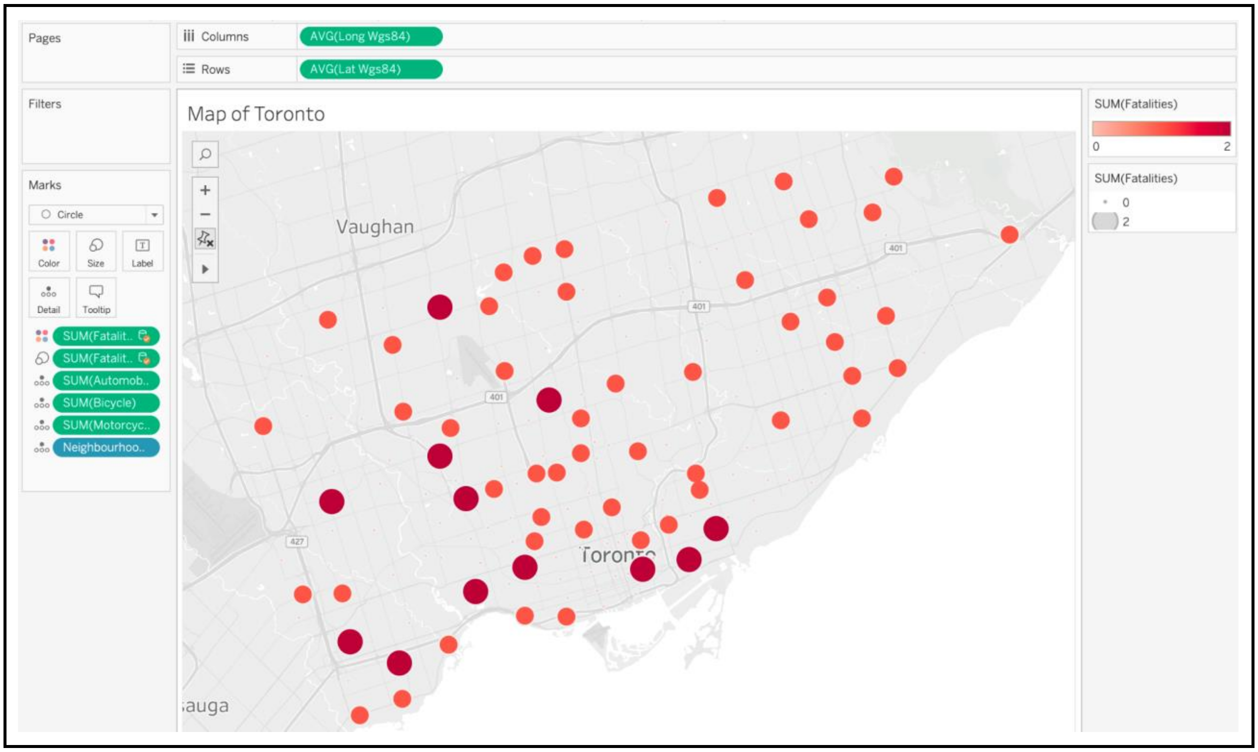


That drop in 2020 might also be skewing our results, it might be better to take it out and look at the other years. The results might change. Due to covid, traffic was non existent so was accidents. However in 2024, we are almost back to pre-covid numbers and we can expect the numbers to follow the same pattern for the the next few years.

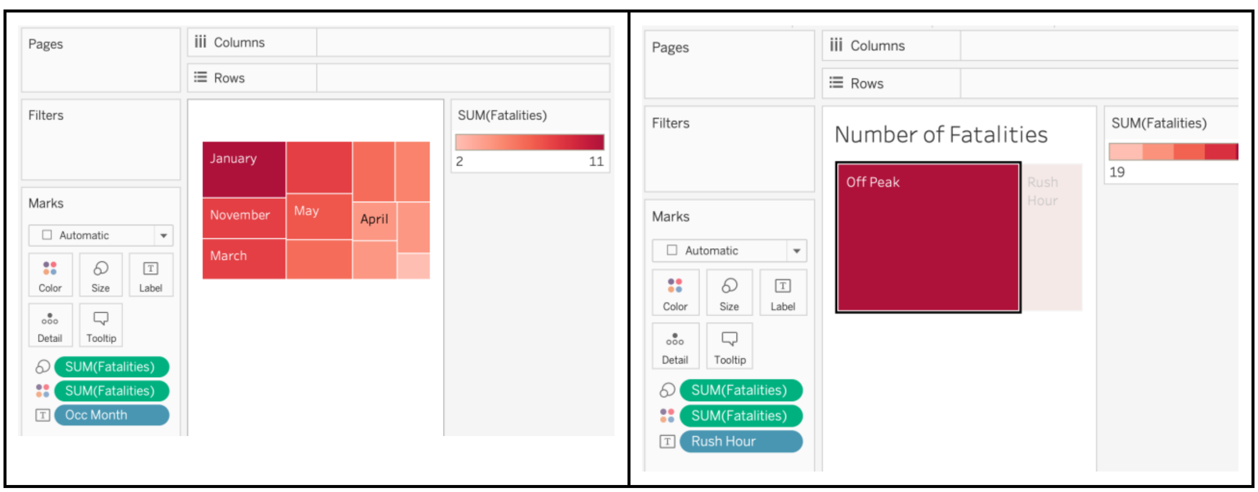
5. **Dashboarding**

Dashboard 1: Historical Data

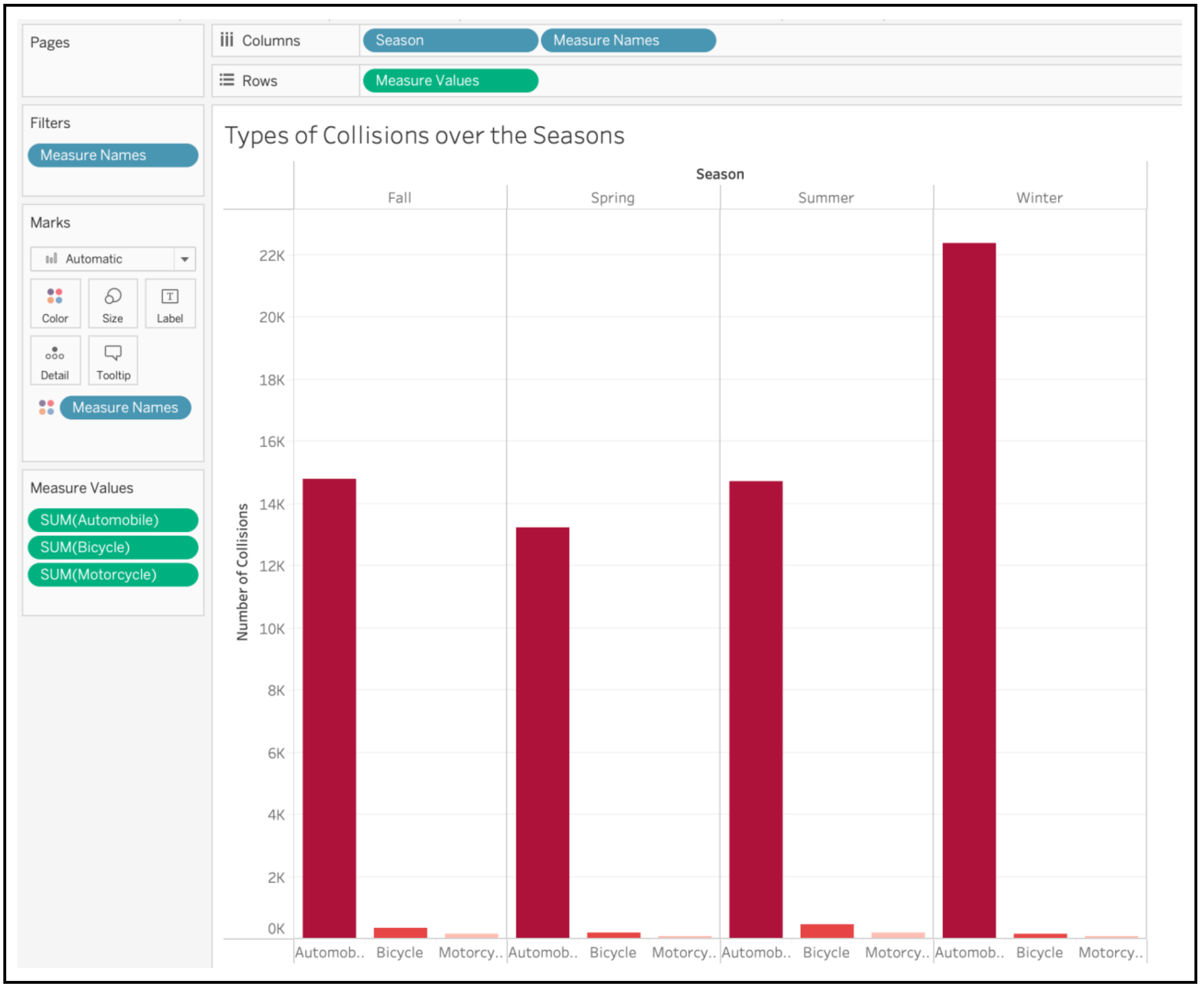
Map of Toronto Tableau inputs:



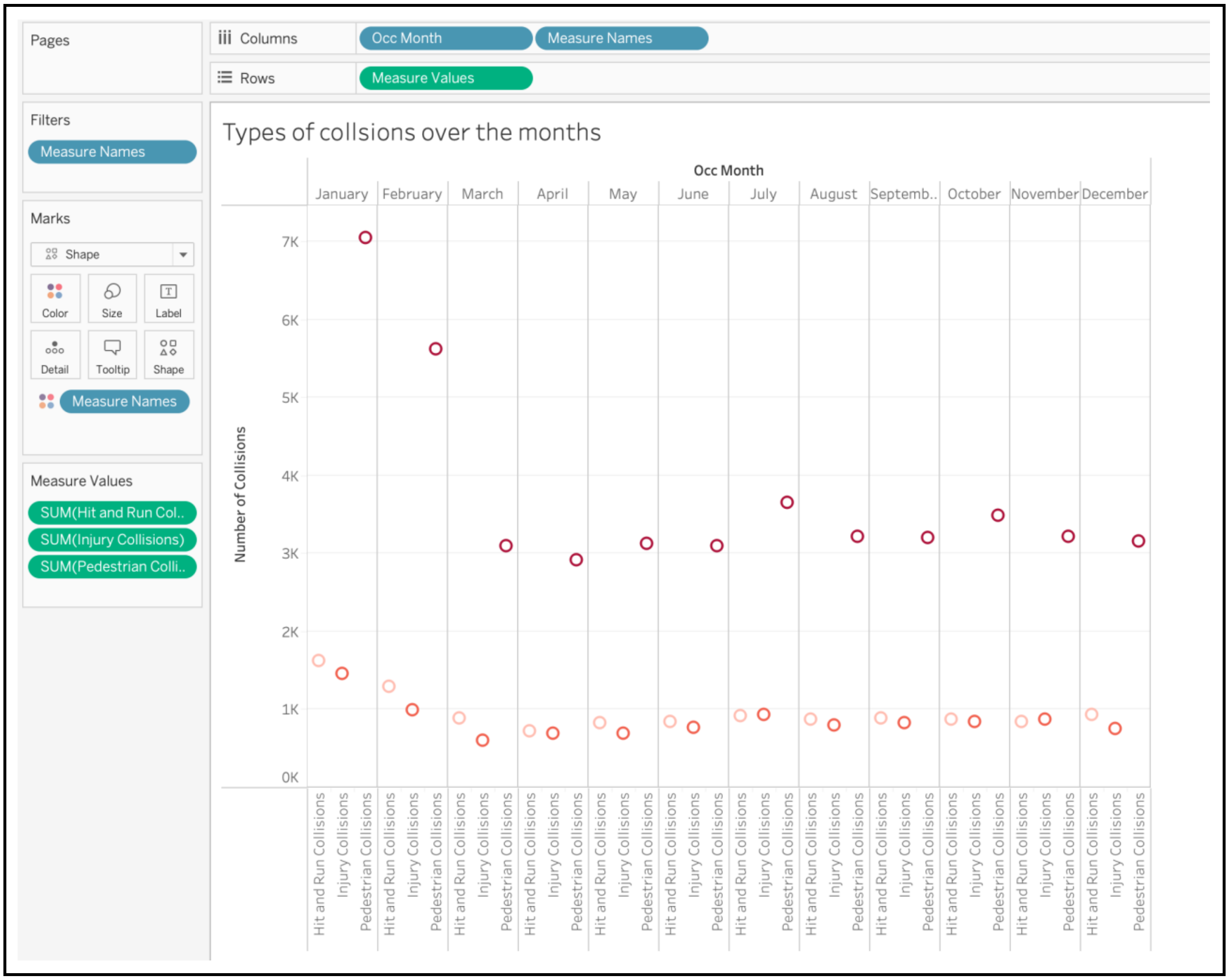
Number of Fatalities Tableau inputs:



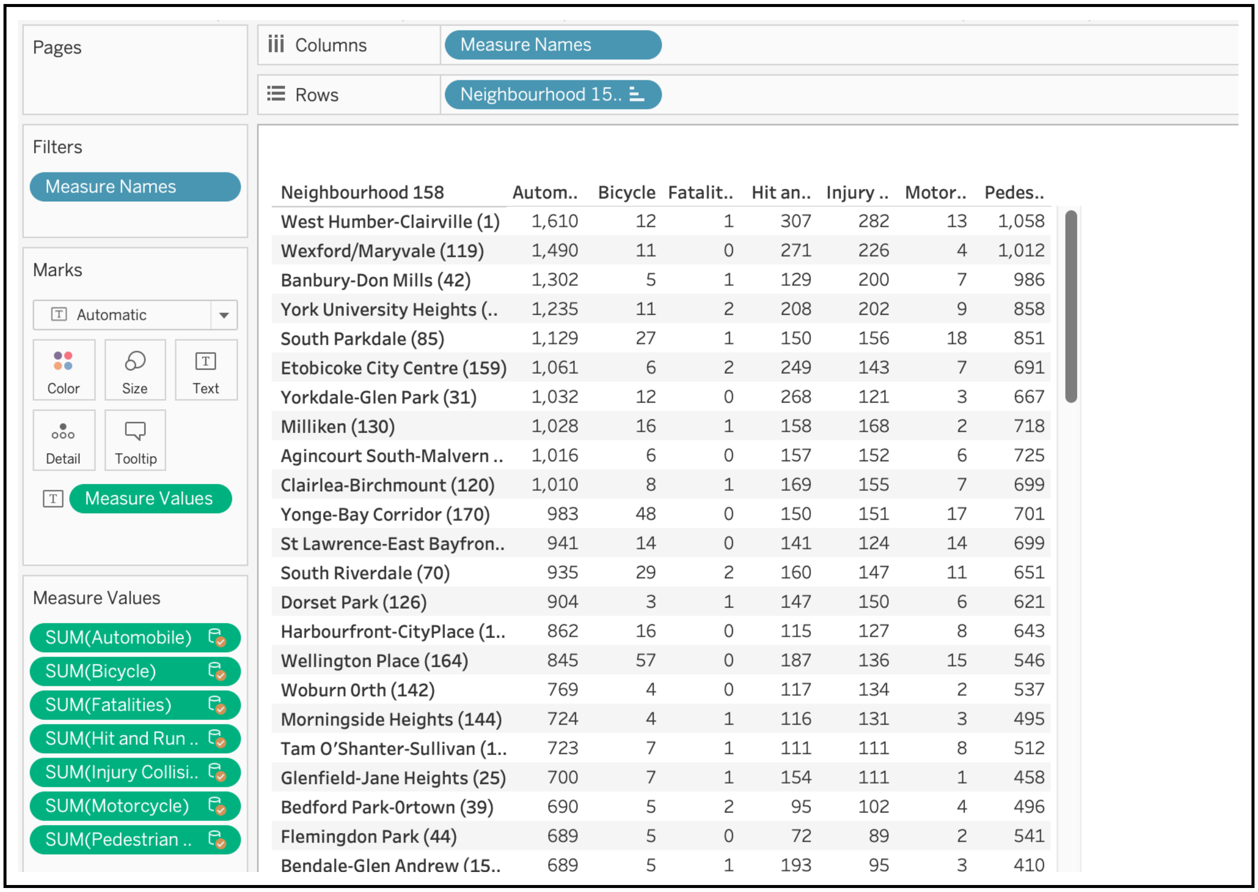
Types of Collisions over the Seasons Tableau inputs:



Types of Collisions over the Months Tableau inputs:

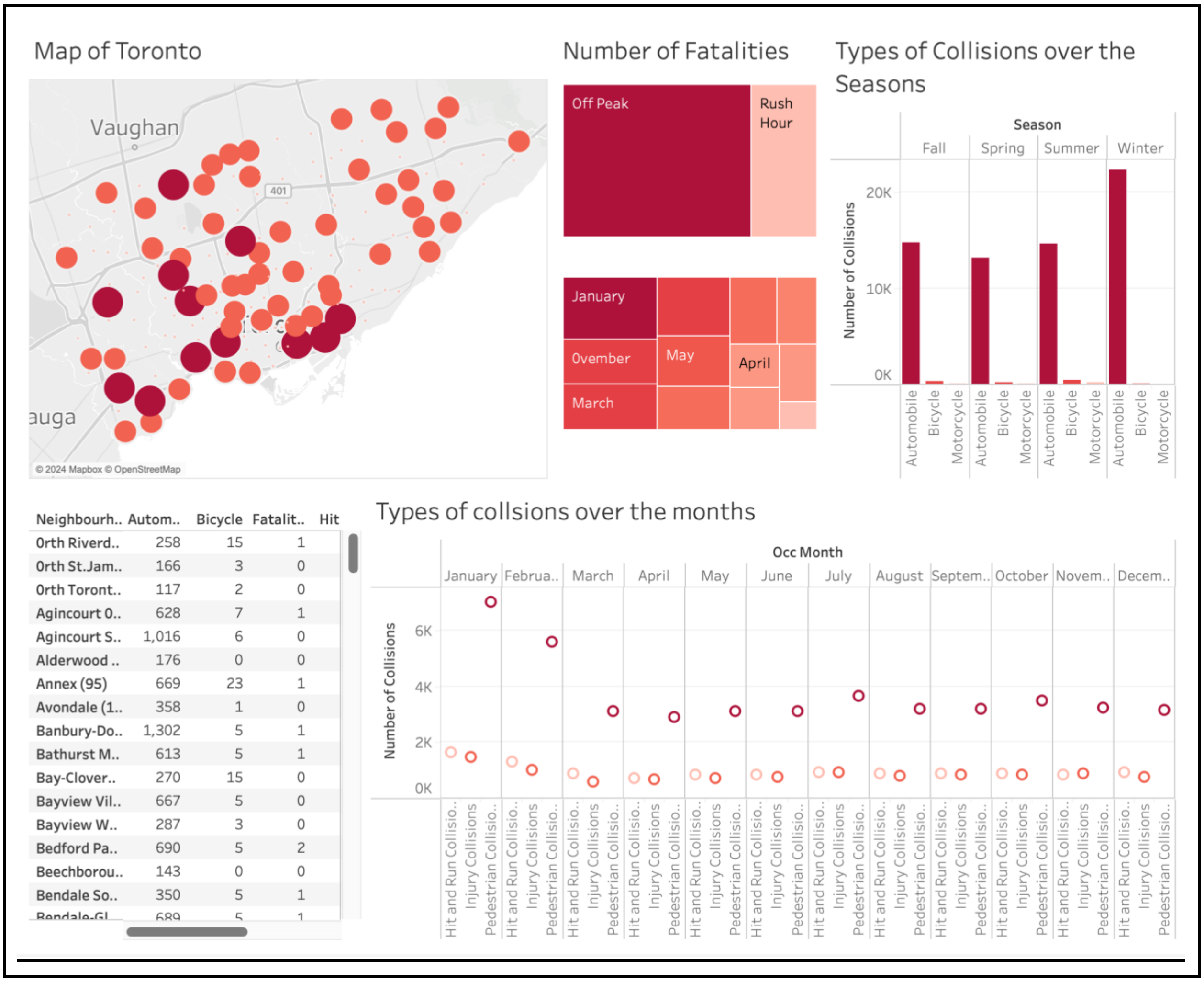


Neighbourhood Table Tableau inputs:



Neighbourhood 158	Autom..	Bicycle	Fatalit..	Hit an..	Injury ..	Motor..	Pedes..
West Humber-Clairville (1)	1,610	12	1	307	282	13	1,058
Wexford/Maryvale (119)	1,490	11	0	271	226	4	1,012
Banbury-Don Mills (42)	1,302	5	1	129	200	7	986
York University Heights (..	1,235	11	2	208	202	9	858
South Parkdale (85)	1,129	27	1	150	156	18	851
Etobicoke City Centre (159)	1,061	6	2	249	143	7	691
Yorkdale-Glen Park (31)	1,032	12	0	268	121	3	667
Milliken (130)	1,028	16	1	158	168	2	718
Agincourt South-Malvern ..	1,016	6	0	157	152	6	725
Clairlea-Birchmount (120)	1,010	8	1	169	155	7	699
Yonge-Bay Corridor (170)	983	48	0	150	151	17	701
St Lawrence-East Bayfron..	941	14	0	141	124	14	699
South Riverdale (70)	935	29	2	160	147	11	651
Dorset Park (126)	904	3	1	147	150	6	621
Harbourfront-CityPlace (1..	862	16	0	115	127	8	643
Wellington Place (164)	845	57	0	187	136	15	546
Woburn Orth (142)	769	4	0	117	134	2	537
Morningside Heights (144)	724	4	1	116	131	3	495
Tam O'Shanter-Sullivan (1..	723	7	1	111	111	8	512
Glenfield-Jane Heights (25)	700	7	1	154	111	1	458
Bedford Park-Ortown (39)	690	5	2	95	102	4	496
Flemingdon Park (44)	689	5	0	72	89	2	541
Bendale-Glen Andrew (15..	689	5	1	193	95	3	410

Final Dashboard:



Dashboard 2: Forecasting Analysis

Significant graphical results taken from the forecasting section were combined to create the second dashboard using the image and text features on Tableau.

Final Dashboard: