research project data

October 27, 2024

load libraries

[27]:
```
!pip install folium
```

```
Requirement already satisfied: folium in c:\users\rhous\anaconda3\lib\site-
packages (0.18.0)
Requirement already satisfied: xyzservices in c:\users\rhous\anaconda3\lib\site-
packages (from folium) (2024.9.0)
Requirement already satisfied: branca>=0.6.0 in
c:\users\rhous\anaconda3\lib\site-packages (from folium) (0.8.0)
Requirement already satisfied: numpy in c:\users\rhous\anaconda3\lib\site-
packages (from folium) (1.20.1)
Requirement already satisfied: jinja2>=2.9 in c:\users\rhous\anaconda3\lib\site-
packages (from folium) (3.1.4)
Requirement already satisfied: requests in c:\users\rhous\anaconda3\lib\site-
packages (from folium) (2.25.1)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\users\rhous\anaconda3\lib\site-packages (from jinja2>=2.9->folium) (2.1.5)
Requirement already satisfied: idna<3,>=2.5 in
c:\users\rhous\anaconda3\lib\site-packages (from requests->folium) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\rhous\anaconda3\lib\site-packages (from requests->folium) (2020.12.5)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
c:\users\rhous\anaconda3\lib\site-packages (from requests->folium) (1.26.4)
Requirement already satisfied: chardet<5,>=3.0.2 in
c:\users\rhous\anaconda3\lib\site-packages (from requests->folium) (4.0.0)
```

[41]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import folium
from datetime import datetime
from folium.plugins import HeatMap
from IPython.display import display
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.preprocessing import LabelEncoder
```

read data

[29]:
```python
# Load datasets
crashes_df = pd.read_csv('Z:\\Documents\\College\\Data 698 - Master\'s␣
↪Thesis\\data\\Motor_Vehicle_Collisions_-_Crashes_20241019.csv')
persons_df = pd.read_csv('Z:\\Documents\\College\\Data 698 - Master\'s␣
↪Thesis\\data\\Motor_Vehicle_Collisions_-_Person_20241019.csv')
vehicles_df= pd.read_csv('Z:\\Documents\\College\\Data 698 - Master\'s␣
↪Thesis\\data\\Motor_Vehicle_Collisions_-_Vehicles.csv')

# Convert 'CRASH DATE' to datetime
crashes_df['CRASH DATE'] = pd.to_datetime(crashes_df['CRASH DATE'],␣
↪errors='coerce')
persons_df['CRASH_DATE'] = pd.to_datetime(persons_df['CRASH_DATE'],␣
↪errors='coerce')
vehicles_df['CRASH_DATE'] = pd.to_datetime(vehicles_df['CRASH_DATE'],␣
↪errors='coerce')
```

[26]:
```python
# Define the date range for filtering
start_date = datetime(2017, 1, 1)
end_date = datetime.now()

# Filter datasets to only include records from 2017 to present
crashes_df = crashes_df[(crashes_df['CRASH DATE'] >= start_date) &␣
↪(crashes_df['CRASH DATE'] <= end_date)]
persons_df = persons_df[(persons_df['CRASH_DATE'] >= start_date) &␣
↪(persons_df['CRASH_DATE'] <= end_date)]
vehicles_df = vehicles_df[(vehicles_df['CRASH_DATE'] >= start_date) &␣
↪(vehicles_df['CRASH_DATE'] <= end_date)]

# Merge the datasets on 'COLLISION_ID'
merged_df = crashes_df.merge(persons_df, how='left', on='COLLISION_ID')
merged_df = merged_df.merge(vehicles_df, how='left', on='COLLISION_ID')
```

[30]:
```python
#get info
# List of columns to drop
drop_columns = [
    'ON STREET NAME', 'CROSS STREET NAME', 'OFF STREET NAME',
    'DRIVER_LICENSE_STATUS', 'DRIVER_LICENSE_JURISDICTION',
    'VEHICLE_DAMAGE', 'VEHICLE_DAMAGE_1', 'VEHICLE_DAMAGE_2',␣
↪'VEHICLE_DAMAGE_3',
    'EMOTIONAL_STATUS', 'EJECTION', 'POSITION_IN_VEHICLE', 'SAFETY_EQUIPMENT',
    'PUBLIC_PROPERTY_DAMAGE', 'PUBLIC_PROPERTY_DAMAGE_TYPE',
    'COMPLAINT', 'PED_ROLE', 'PED_LOCATION', 'PED_ACTION',
    'VEHICLE_MAKE', 'VEHICLE_MODEL', 'VEHICLE_YEAR'
```

```
]

# Drop the columns from the merged DataFrame
merged_df.drop(columns=drop_columns, inplace=True)
merged_df.info()
```

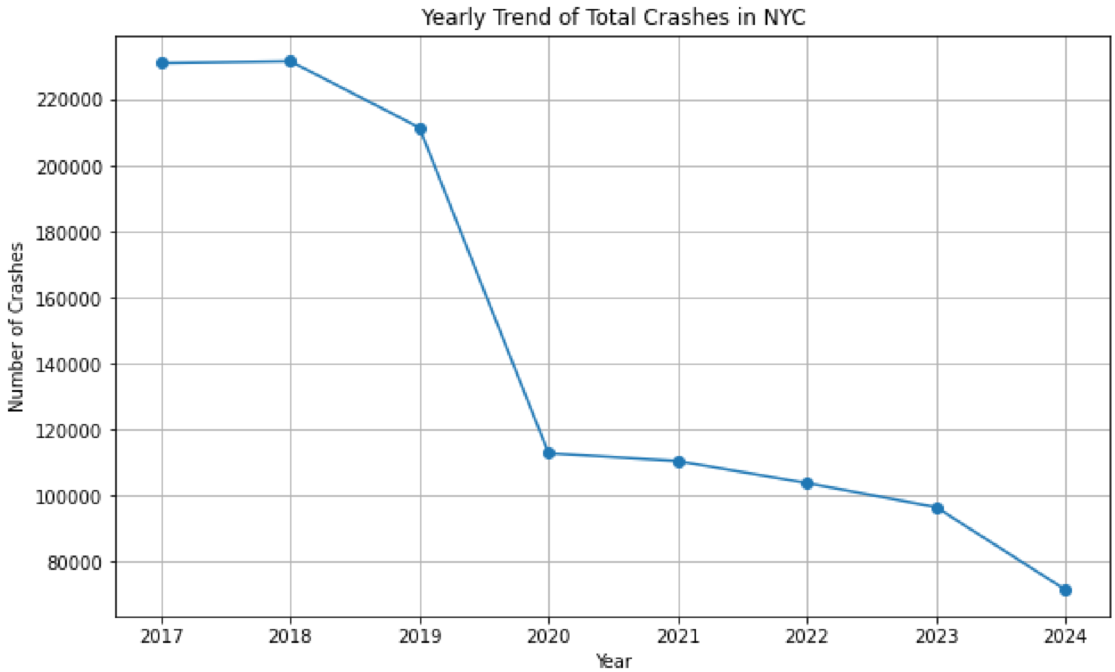
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9651082 entries, 0 to 9651081
Data columns (total 51 columns):
 #   Column                         Dtype
---  ------                         -----
 0   CRASH DATE                     datetime64[ns]
 1   CRASH TIME                     object
 2   BOROUGH                        object
 3   ZIP CODE                       object
 4   LATITUDE                       float64
 5   LONGITUDE                      float64
 6   LOCATION                       object
 7   NUMBER OF PERSONS INJURED      float64
 8   NUMBER OF PERSONS KILLED       float64
 9   NUMBER OF PEDESTRIANS INJURED  int64
 10  NUMBER OF PEDESTRIANS KILLED   int64
 11  NUMBER OF CYCLIST INJURED      int64
 12  NUMBER OF CYCLIST KILLED       int64
 13  NUMBER OF MOTORIST INJURED     int64
 14  NUMBER OF MOTORIST KILLED      int64
 15  CONTRIBUTING FACTOR VEHICLE 1  object
 16  CONTRIBUTING FACTOR VEHICLE 2  object
 17  CONTRIBUTING FACTOR VEHICLE 3  object
 18  CONTRIBUTING FACTOR VEHICLE 4  object
 19  CONTRIBUTING FACTOR VEHICLE 5  object
 20  COLLISION_ID                   int64
 21  VEHICLE TYPE CODE 1            object
 22  VEHICLE TYPE CODE 2            object
 23  VEHICLE TYPE CODE 3            object
 24  VEHICLE TYPE CODE 4            object
 25  VEHICLE TYPE CODE 5            object
 26  UNIQUE_ID_x                    float64
 27  CRASH_DATE_x                   datetime64[ns]
 28  CRASH_TIME_x                   object
 29  PERSON_ID                      object
 30  PERSON_TYPE                    object
 31  PERSON_INJURY                  object
 32  VEHICLE_ID_x                   float64
 33  PERSON_AGE                     float64
 34  BODILY_INJURY                  object
 35  CONTRIBUTING_FACTOR_1_x        object
```

```
 36  CONTRIBUTING_FACTOR_2_x        object
 37  PERSON_SEX                     object
 38  UNIQUE_ID_y                    float64
 39  CRASH_DATE_y                   datetime64[ns]
 40  CRASH_TIME_y                   object
 41  VEHICLE_ID_y                   object
 42  STATE_REGISTRATION             object
 43  VEHICLE_TYPE                   object
 44  TRAVEL_DIRECTION               object
 45  VEHICLE_OCCUPANTS              float64
 46  DRIVER_SEX                     object
 47  PRE_CRASH                      object
 48  POINT_OF_IMPACT                object
 49  CONTRIBUTING_FACTOR_1_y        object
 50  CONTRIBUTING_FACTOR_2_y        object
dtypes: datetime64[ns](3), float64(9), int64(7), object(32)
memory usage: 3.7+ GB
```

[32]:
```
#save dataframe
merged_df.to_csv("Z:\\Documents\\College\\Data 698 - Master\'s␣
↪Thesis\\data\\MergedData.csv")
```

visualize crash trends over time

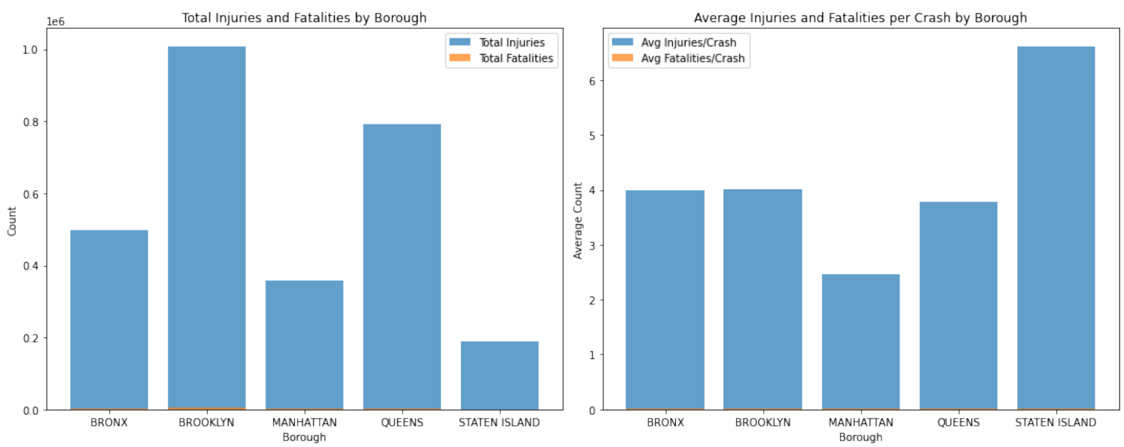
[31]:
```
# Group by year and calculate the total number of crashes
merged_df['CRASH DATE'] = pd.to_datetime(merged_df['CRASH DATE'])
merged_df['Year'] = merged_df['CRASH DATE'].dt.year

# Yearly trend of total crashes
yearly_crash_trend = merged_df.groupby('Year')['COLLISION_ID'].nunique()

# Plotting the yearly trend
plt.figure(figsize=(10, 6))
plt.plot(yearly_crash_trend.index, yearly_crash_trend.values, marker='o')
plt.title('Yearly Trend of Total Crashes in NYC')
plt.xlabel('Year')
plt.ylabel('Number of Crashes')
plt.grid(True)
plt.show()
```



Analyze crash severity by borough

[34]:
```
# Calculate total crashes, injuries, and fatalities by borough
severity_by_borough = merged_df.groupby('BOROUGH').agg(
    total_crashes=('COLLISION_ID', 'nunique'),
    total_injuries=('NUMBER OF PERSONS INJURED', 'sum'),
    total_fatalities=('NUMBER OF PERSONS KILLED', 'sum')
).reset_index()

# Calculate average severity per crash by borough
severity_by_borough['avg_injuries_per_crash'] =
 ↪severity_by_borough['total_injuries'] / severity_by_borough['total_crashes']
severity_by_borough['avg_fatalities_per_crash'] =
 ↪severity_by_borough['total_fatalities'] /
 ↪severity_by_borough['total_crashes']

# Plotting the results
fig, ax = plt.subplots(1, 2, figsize=(15, 6))

# Plot total injuries and fatalities by borough
ax[0].bar(severity_by_borough['BOROUGH'],
 ↪severity_by_borough['total_injuries'], label='Total Injuries', alpha=0.7)
ax[0].bar(severity_by_borough['BOROUGH'],
 ↪severity_by_borough['total_fatalities'], label='Total Fatalities', alpha=0.7)
ax[0].set_title('Total Injuries and Fatalities by Borough')
```

```
ax[0].set_xlabel('Borough')
ax[0].set_ylabel('Count')
ax[0].legend()

# Plot average injuries and fatalities per crash by borough
ax[1].bar(severity_by_borough['BOROUGH'],␣
↪severity_by_borough['avg_injuries_per_crash'], label='Avg Injuries/Crash',␣
↪alpha=0.7)
ax[1].bar(severity_by_borough['BOROUGH'],␣
↪severity_by_borough['avg_fatalities_per_crash'], label='Avg Fatalities/
↪Crash', alpha=0.7)
ax[1].set_title('Average Injuries and Fatalities per Crash by Borough')
ax[1].set_xlabel('Borough')
ax[1].set_ylabel('Average Count')
ax[1].legend()

plt.tight_layout()
plt.show()
```

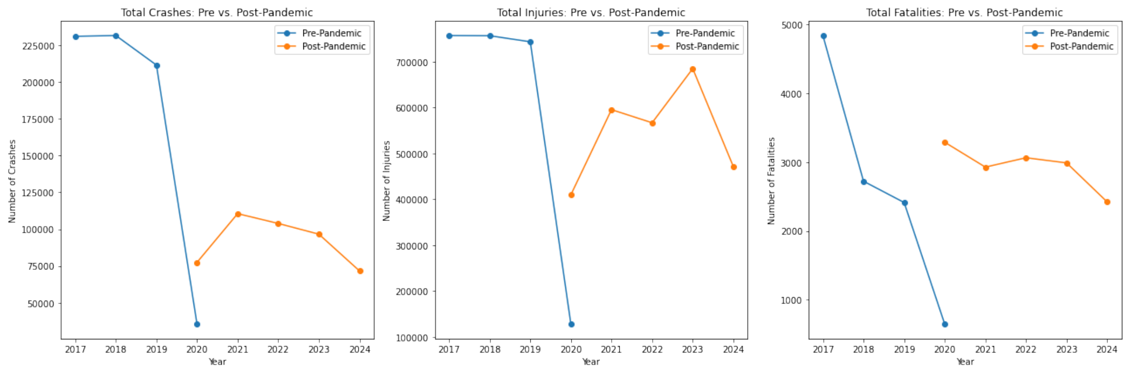


analyze pre and post pandemic data

[35]:
```
# Define the date ranges
pre_pandemic_end = '2020-03-15'
post_pandemic_start = '2020-03-16'

# Filter the data for pre-pandemic and post-pandemic periods
pre_pandemic_df = merged_df[(merged_df['CRASH DATE'] >= '2017-01-01') &␣
↪(merged_df['CRASH DATE'] <= pre_pandemic_end)]
post_pandemic_df = merged_df[(merged_df['CRASH DATE'] >= post_pandemic_start)]

# Calculate total crashes, injuries, and fatalities by year for each period
```

```python
pre_pandemic_trend = pre_pandemic_df.groupby(pre_pandemic_df['CRASH DATE'].dt.year).agg(
    total_crashes=('COLLISION_ID', 'nunique'),
    total_injuries=('NUMBER OF PERSONS INJURED', 'sum'),
    total_fatalities=('NUMBER OF PERSONS KILLED', 'sum')
).reset_index()

post_pandemic_trend = post_pandemic_df.groupby(post_pandemic_df['CRASH DATE'].dt.year).agg(
    total_crashes=('COLLISION_ID', 'nunique'),
    total_injuries=('NUMBER OF PERSONS INJURED', 'sum'),
    total_fatalities=('NUMBER OF PERSONS KILLED', 'sum')
).reset_index()

# Plotting the results
fig, ax = plt.subplots(1, 3, figsize=(18, 6), sharex=True)

# Total crashes
ax[0].plot(pre_pandemic_trend['CRASH DATE'], pre_pandemic_trend['total_crashes'], label='Pre-Pandemic', marker='o')
ax[0].plot(post_pandemic_trend['CRASH DATE'], post_pandemic_trend['total_crashes'], label='Post-Pandemic', marker='o')
ax[0].set_title('Total Crashes: Pre vs. Post-Pandemic')
ax[0].set_xlabel('Year')
ax[0].set_ylabel('Number of Crashes')
ax[0].legend()

# Total injuries
ax[1].plot(pre_pandemic_trend['CRASH DATE'], pre_pandemic_trend['total_injuries'], label='Pre-Pandemic', marker='o')
ax[1].plot(post_pandemic_trend['CRASH DATE'], post_pandemic_trend['total_injuries'], label='Post-Pandemic', marker='o')
ax[1].set_title('Total Injuries: Pre vs. Post-Pandemic')
ax[1].set_xlabel('Year')
ax[1].set_ylabel('Number of Injuries')
ax[1].legend()

# Total fatalities
ax[2].plot(pre_pandemic_trend['CRASH DATE'], pre_pandemic_trend['total_fatalities'], label='Pre-Pandemic', marker='o')
ax[2].plot(post_pandemic_trend['CRASH DATE'], post_pandemic_trend['total_fatalities'], label='Post-Pandemic', marker='o')
ax[2].set_title('Total Fatalities: Pre vs. Post-Pandemic')
ax[2].set_xlabel('Year')
ax[2].set_ylabel('Number of Fatalities')
ax[2].legend()
```

```python
plt.tight_layout()
plt.show()
```



crash factor analysis

[36]:
```python
# Add a column to categorize crash severity
merged_df['SEVERE_CRASH'] = merged_df['NUMBER OF PERSONS INJURED'] + merged_df['NUMBER OF PERSONS KILLED']
merged_df['SEVERE_CRASH'] = merged_df['SEVERE_CRASH'].apply(lambda x: 1 if x > 0 else 0)

# Show the distribution of severe vs. non-severe crashes
severity_counts = merged_df['SEVERE_CRASH'].value_counts()
print(severity_counts)

# Calculate the frequency of contributing factors for severe vs. non-severe crashes
contributing_factors = [
    'CONTRIBUTING FACTOR VEHICLE 1', 'CONTRIBUTING FACTOR VEHICLE 2',
    'CONTRIBUTING FACTOR VEHICLE 3', 'CONTRIBUTING FACTOR VEHICLE 4',
    'CONTRIBUTING FACTOR VEHICLE 5'
]

# Melt the contributing factor columns for easier analysis
factors_df = merged_df.melt(
    id_vars=['SEVERE_CRASH'],
    value_vars=contributing_factors,
    var_name='CONTRIBUTING_FACTOR_TYPE',
    value_name='CONTRIBUTING_FACTOR'
)

# Filter out missing or unknown factors
```

```
factors_df = factors_df[factors_df['CONTRIBUTING_FACTOR'].notna() &␣
↪(factors_df['CONTRIBUTING_FACTOR'] != 'Unspecified')]

# Calculate the distribution of contributing factors by crash severity
severity_factors = factors_df.groupby(['CONTRIBUTING_FACTOR', 'SEVERE_CRASH']).
↪size().unstack(fill_value=0)

# Normalize to get proportions
severity_factors = severity_factors.div(severity_factors.sum(axis=1), axis=0)

# Display the top contributing factors for severe crashes
severity_factors.sort_values(by=1, ascending=False).head(10)
```

```
0    6668079
1    2983003
Name: SEVERE_CRASH, dtype: int64
```

[36]:

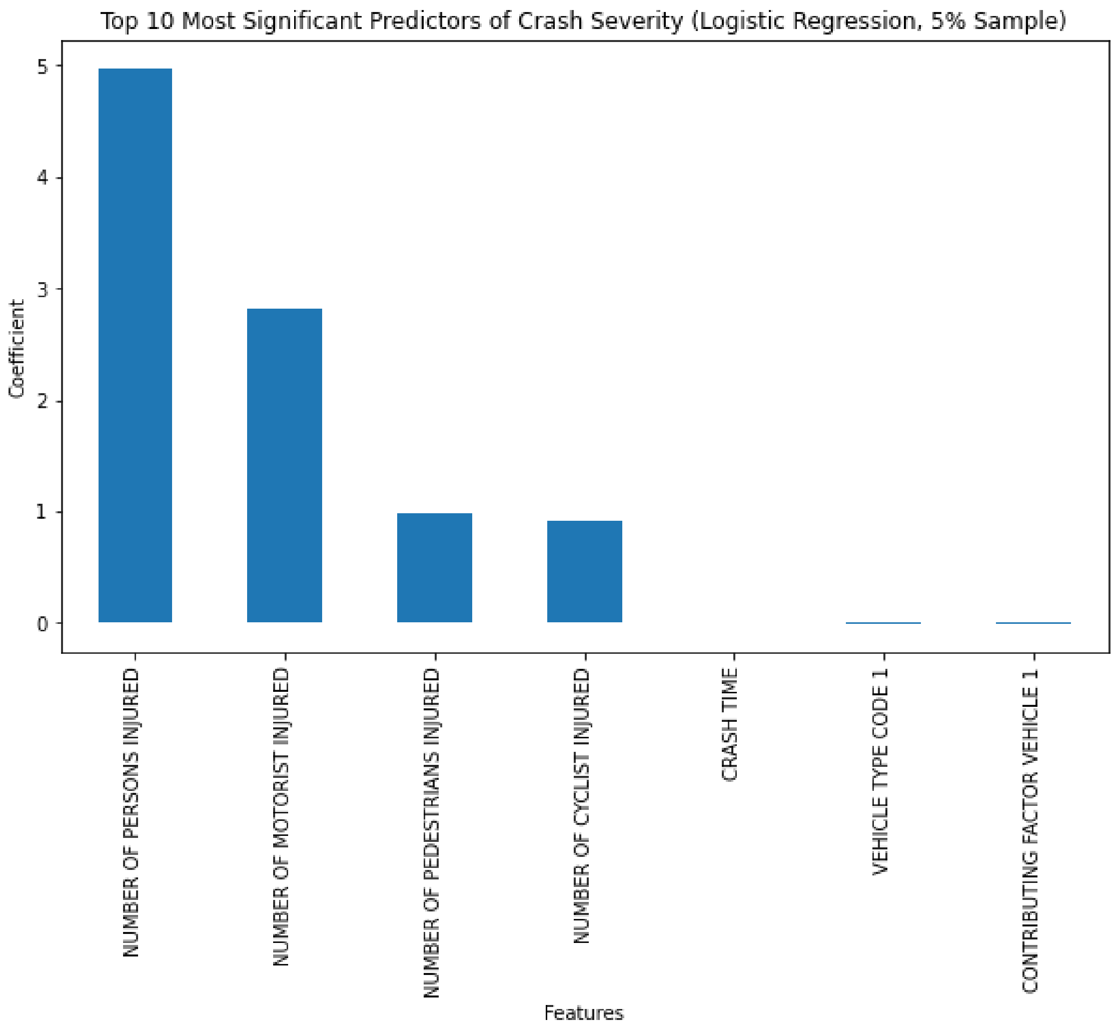
SEVERE_CRASH	0	1
CONTRIBUTING_FACTOR		
Reaction to Other Uninvolved Vehicle	0.000000	1.000000
Lost Consciousness	0.162373	0.837627
Illnes	0.263009	0.736991
Pedestrian/Bicyclist/Other Pedestrian Error/Con…	0.271230	0.728770
Listening/Using Headphones	0.284553	0.715447
Physical Disability	0.398547	0.601453
Drugs (illegal)	0.410354	0.589646
Traffic Control Disregarded	0.441334	0.558666
Unsafe Speed	0.450932	0.549068
Headlights Defective	0.461303	0.538697

[44]:

```
# Select a smaller set of features
selected_features = [
    'NUMBER OF PERSONS INJURED', 'NUMBER OF PEDESTRIANS INJURED',
    'NUMBER OF CYCLIST INJURED', 'NUMBER OF MOTORIST INJURED',
    'CONTRIBUTING FACTOR VEHICLE 1', 'VEHICLE TYPE CODE 1', 'CRASH TIME'
]

# Filter the dataset
model_features = merged_df[selected_features].copy()

# Downcast numeric columns to reduce memory usage
numeric_cols = ['NUMBER OF PERSONS INJURED', 'NUMBER OF PEDESTRIANS INJURED',
                'NUMBER OF CYCLIST INJURED', 'NUMBER OF MOTORIST INJURED']
model_features[numeric_cols] = model_features[numeric_cols].apply(pd.
↪to_numeric, downcast='float')

# Fill missing values in numeric columns with 0
```

```
model_features[numeric_cols] = model_features[numeric_cols].fillna(0)

# Apply Label Encoding to categorical columns, handling NaNs by filling with␣
↪'Unknown'
label_enc = LabelEncoder()
for col in ['CONTRIBUTING FACTOR VEHICLE 1', 'VEHICLE TYPE CODE 1', 'CRASH␣
↪TIME']:
    model_features[col] = model_features[col].fillna('Unknown')
    model_features[col] = label_enc.fit_transform(model_features[col].
↪astype(str))

# Check for and replace infinities
model_features.replace([np.inf, -np.inf], np.nan, inplace=True)
model_features.dropna(inplace=True)

# Define the target variable
y = merged_df['SEVERE_CRASH'].fillna(0)

# Sample 5% of the data to further reduce memory usage
X_sample, _, y_sample, _ = train_test_split(model_features, y, test_size=0.95,␣
↪random_state=42)

# Split the 5% sample into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample,␣
↪test_size=0.3, random_state=42)

# Initialize and fit the Logistic Regression model
logreg = LogisticRegression(max_iter=1000, solver='saga')
logreg.fit(X_train, y_train)

# Calculate feature importance (coefficients) for logistic regression
importance = pd.Series(logreg.coef_[0], index=X_train.columns).
↪sort_values(ascending=False)

# Plot the top predictors
plt.figure(figsize=(10, 6))
importance.head(10).plot(kind='bar')
plt.title('Top 10 Most Significant Predictors of Crash Severity (Logistic␣
↪Regression, 5% Sample)')
plt.xlabel('Features')
plt.ylabel('Coefficient')
plt.show()

# Print the classification report for model performance
y_pred = logreg.predict(X_test)
print(classification_report(y_test, y_pred))
```

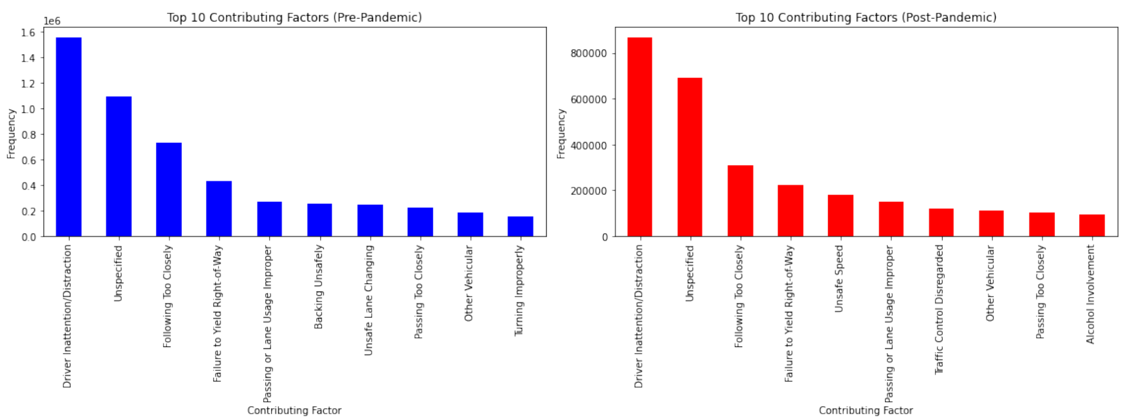
```
C:\Users\rhous\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn("The max_iter was reached which means "
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    100221
           1       1.00      1.00      1.00     44546

    accuracy                           1.00    144767
   macro avg       1.00      1.00      1.00    144767
weighted avg       1.00      1.00      1.00    144767
```

changes in contributing factors after covid

[45]:
```python
# Analyze the top contributing factors pre- and post-pandemic
top_factors_pre = pre_pandemic_df['CONTRIBUTING FACTOR VEHICLE 1'].value_counts().head(10)
top_factors_post = post_pandemic_df['CONTRIBUTING FACTOR VEHICLE 1'].value_counts().head(10)

# Plot the top contributing factors
fig, ax = plt.subplots(1, 2, figsize=(16, 6))

top_factors_pre.plot(kind='bar', ax=ax[0], color='b')
ax[0].set_title('Top 10 Contributing Factors (Pre-Pandemic)')
ax[0].set_xlabel('Contributing Factor')
ax[0].set_ylabel('Frequency')

top_factors_post.plot(kind='bar', ax=ax[1], color='r')
ax[1].set_title('Top 10 Contributing Factors (Post-Pandemic)')
ax[1].set_xlabel('Contributing Factor')
ax[1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```



[46]:
```python
# Pre-pandemic crash locations
pre_pandemic_map = folium.Map(location=[40.7128, -74.0060], zoom_start=11)
pre_heat_data = pre_pandemic_df[['LATITUDE', 'LONGITUDE']].dropna()
HeatMap(data=pre_heat_data.values, radius=10).add_to(pre_pandemic_map)

# Post-pandemic crash locations
post_pandemic_map = folium.Map(location=[40.7128, -74.0060], zoom_start=11)
post_heat_data = post_pandemic_df[['LATITUDE', 'LONGITUDE']].dropna()
HeatMap(data=post_heat_data.values, radius=10).add_to(post_pandemic_map)
```

```python
# Display the maps (for Jupyter Notebook)
pre_pandemic_map
post_pandemic_map
```

[46]: <folium.folium.Map at 0x1bcaa41c340>

[ ]: