Chicago Car Crashes Analysis

This notebook uses three datasets from the City of Chicago's data portal:

1. Car crashes
2. People in crashes
3. Socially disadvantaged areas geographies

Using this data, it applies increasingly sophisticated modeling approaches to detect whether a given crash is likely to result in a fatality or incapacitating injury. The target variable is severely imbalanced, with only 1.8% of accidents involving severe consequences. As a result, accuracy of any model is likely to be naturally high, and also a bad metric. Instead, recall more closely fits the City's goals. At the expense of some false predictions, models targeting an improved recall score will provide actionable insights to the City to minimize these types of accidents.

In [1]:
```
from datetime import datetime
import numpy as np

import pandas as pd
import geopandas as gpd
from shapely.geometry import Point
import folium

from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import recall_score, classification_report, roc_auc_score, confusio
from sklearn.dummy import DummyClassifier

from imblearn.over_sampling import SMOTE

import matplotlib.pyplot as plt
%matplotlib inline
```

Import Data

These datasets are large. Specifying the specific columns to import for each dataset limits the time it takes to bring them into dataframes, and will minimize system resources.

In [2]:
```
crashes_usecols = ['LATITUDE','LONGITUDE','CRASH_RECORD_ID','CRASH_DATE','DEVICE_CONDITI
                   'WEATHER_CONDITION','LIGHTING_CONDITION','ALIGNMENT','ROADWAY_SURFACE_
                   'INJURIES_FATAL','INJURIES_INCAPACITATING','CRASH_HOUR','CRASH_DAY_OF_

people_usecols = ['CRASH_RECORD_ID','SAFETY_EQUIPMENT','PHYSICAL_CONDITION']
```

In [3]:
```
# Import CSV files into DataFrames
crashes_df = pd.read_csv('Data/Traffic_Crashes_-_Crashes_20240412.csv', usecols=crashes_
people_df = pd.read_csv('Data/Traffic_Crashes_-_People_20240412.csv', usecols=people_use
```

In [4]:
```
# Load the SHP file into a GeoDataFrame
districts_gdf = gpd.read_file('Data\SD_geo.shp')
```

Initial Data Exploration

- The Crashes dataset has 48 columns and the People dataset has 29
- Most columns are not useful, or have substantial gaps in data
- Keep 13 from crashes and 3 from people

In [5]:
```
crashes_col_total = len(pd.read_csv('Data/Traffic_Crashes_-_Crashes_20240412.csv', nrows
people_col_total = len(pd.read_csv('Data/Traffic_Crashes_-_People_20240412.csv', nrows=0
pd.DataFrame({'Dataset':('Crashes','People'),
              'Initial Columns':(crashes_col_total, people_col_total),
              'Used Columns':(len(crashes_df.columns),len(people_df.columns)),
              'Records':(len(crashes_df.index),len(people_df.index))
             })
```

Out[5]:

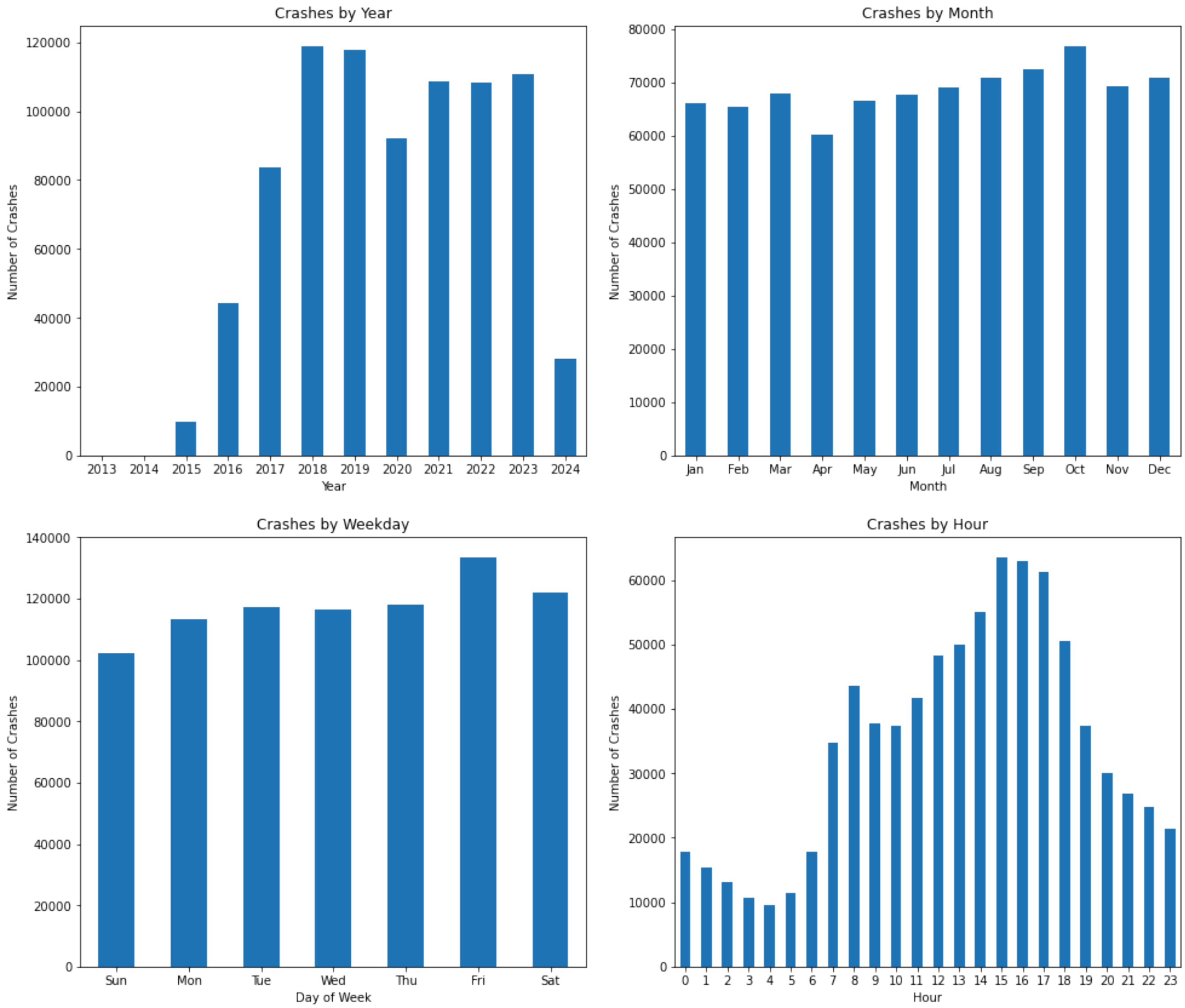
	Dataset	Initial Columns	Used Columns	Records
0	Crashes	48	13	822610
1	People	29	3	1805554

Distribution of Crashes

Examine the distribution of crashes across time to sense whether there are any obvious patterns.

In [6]:

```python
# Extract year and month from CRASH_DATE
year = crashes_df['CRASH_DATE'].str[6:10].astype(int)
month = crashes_df['CRASH_DATE'].str[0:2].astype(int)

# Map numbers to month and weekday names
month_map = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun',
             7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
weekday_map = {1: 'Sun', 2: 'Mon', 3: 'Tue', 4: 'Wed', 5: 'Thu', 6: 'Fri', 7: 'Sat'}

month = month.map(month_map)
weekday = crashes_df['CRASH_DAY_OF_WEEK'].map(weekday_map)

# Plotting
fig, axes = plt.subplots(2, 2, figsize=(14, 12))  # Create a 2x2 grid of subplots

# Histogram of crashes by year
year.value_counts().sort_index().plot(kind='bar', ax=axes[0, 0])
axes[0, 0].set_title('Crashes by Year')
axes[0, 0].set_xlabel('Year')
axes[0, 0].set_ylabel('Number of Crashes')
axes[0, 0].set_xticklabels(axes[0, 0].get_xticklabels(), rotation=0)

# Histogram of crashes by month
month.value_counts().reindex(month_map.values()).plot(kind='bar', ax=axes[0, 1])
axes[0, 1].set_title('Crashes by Month')
axes[0, 1].set_xlabel('Month')
axes[0, 1].set_ylabel('Number of Crashes')
axes[0, 1].set_xticklabels(axes[0, 1].get_xticklabels(), rotation=0)

# Histogram of crashes by weekday
weekday.value_counts().reindex(weekday_map.values()).plot(kind='bar', ax=axes[1, 0])
axes[1, 0].set_title('Crashes by Weekday')
axes[1, 0].set_xlabel('Day of Week')
axes[1, 0].set_ylabel('Number of Crashes')
axes[1, 0].set_xticklabels(axes[1, 0].get_xticklabels(), rotation=0)

# Histogram of crashes by hour
crashes_df['CRASH_HOUR'].value_counts().sort_index().plot(kind='bar', ax=axes[1, 1])
axes[1, 1].set_title('Crashes by Hour')
axes[1, 1].set_xlabel('Hour')
axes[1, 1].set_ylabel('Number of Crashes')
axes[1, 1].set_xticklabels(axes[1, 1].get_xticklabels(), rotation=0)

plt.tight_layout(pad=2, w_pad=2, h_pad=2)
```

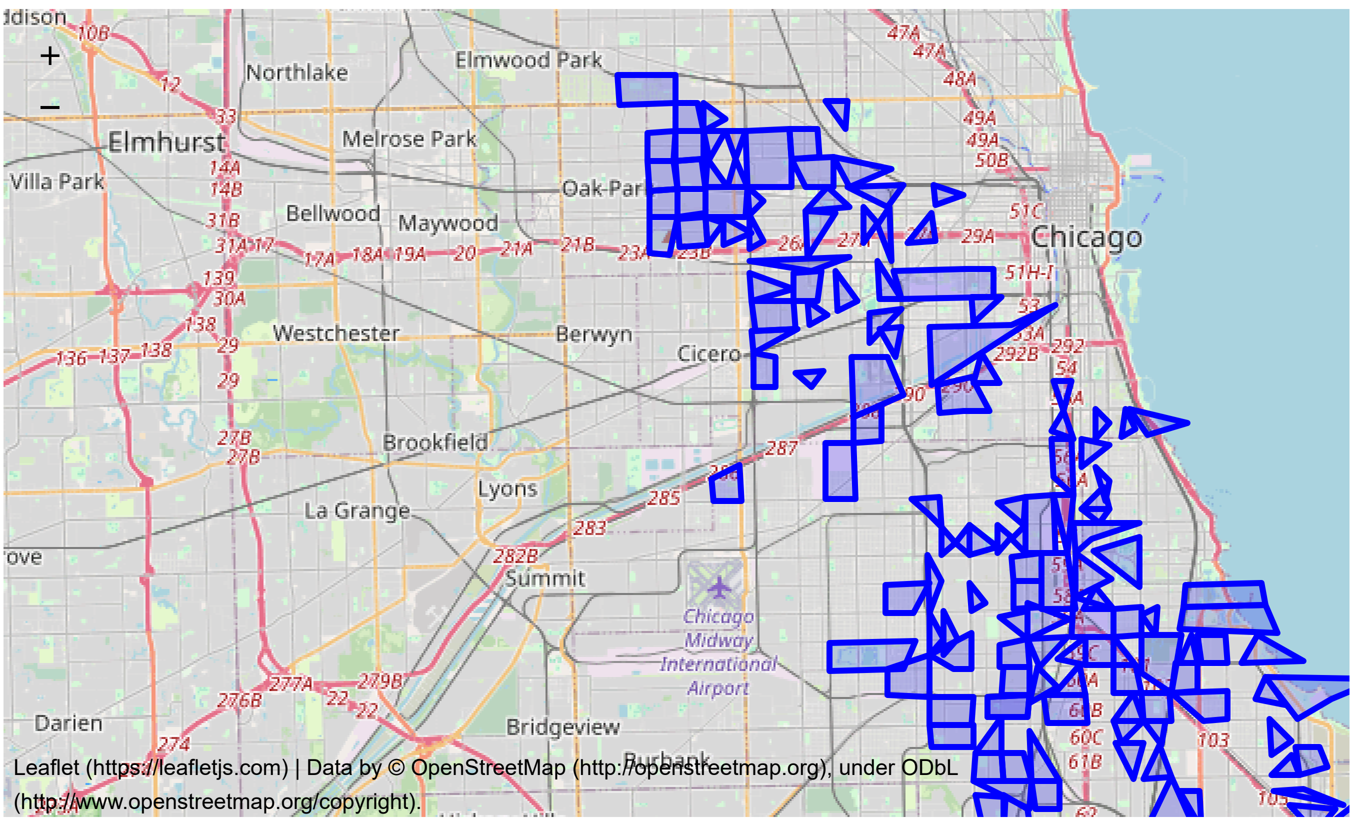


Map - Socially Disadvantaged Districts

Socially disadvantaged districs may play a role in crash outcomes. Display an interactive map showing which parts of Chicago have been deemed 'socially disadvantaged'.

In [7]:
```python
def plot_folium(gdf):
    # Project the geometries to a CRS that uses meters (here using Web Mercator)
    gdf_projected = gdf.to_crs(epsg=3857)

    # Calculate the centroids of the projected geometries
    centroids = gdf_projected.geometry.centroid

    # Convert these centroids back to the geographic CRS to plot on the map
    centroids = centroids.to_crs(epsg=4326)  # EPSG:4326 is the geographic CRS used by m

    # Calculate the center of the map
    center = [centroids.y.mean()-.03, centroids.x.mean()]

    # Initialize the map
    m = folium.Map(location=center, zoom_start=11.2)

    # Simplify and add each geometry to the map
    for _, row in gdf.iterrows():
        sim_geo = row.geometry.simplify(0.005, preserve_topology=False)
        geo_j = folium.GeoJson(data=sim_geo.__geo_interface__,
                               style_function=lambda x: {'fillColor': 'blue', 'color': '
        geo_j.add_to(m)

    return m

# Assuming 'districts_gdf' is your original GeoDataFrame
district_map = plot_folium(districts_gdf)
district_map
```

Out[7]:



Geographic analysis

In [8]:
```
# Drop values with no location information
crashes_df.dropna(subset=['LATITUDE','LONGITUDE'], inplace=True)

# Convert DataFrame into GeoDataFrame
crashes_df['geometry'] = crashes_df.apply(lambda row: Point(row['LONGITUDE'], row['LATIT
crashes_gdf = gpd.GeoDataFrame(crashes_df, geometry='geometry')

# Ensure that both GeoDataFrames use the same CRS
crashes_gdf.crs = districts_gdf.crs

# Spatial join the GeoDataFrames
joined_gdf = gpd.sjoin(crashes_gdf, districts_gdf, how='left', predicate='within')

# Add flag for crashes that are within a district
joined_gdf['WITHIN_DISTRICT'] = joined_gdf['index_right'].apply(lambda x: 1 if pd.notnul

# Drop the geometry, index_right, LATITUDE and LONGITUDE columns
joined_gdf.drop(columns=['LATITUDE','LONGITUDE','geometry','index_right'], axis=1, inpla

# Convert back to DataFrame
crashes_flag_df = pd.DataFrame(joined_gdf)
```

Prepare datetime data for analysis

In [9]:
```
# Helper function to convert dates to seasons

def get_season(date):
    year = date.year
    seasons = [('winter', datetime(year, 1, 1).date(), datetime(year, 2, 28).date()),
               ('spring', datetime(year, 3, 1).date(), datetime(year, 5, 31).date()),
               ('summer', datetime(year, 6, 1).date(), datetime(year, 8, 31).date()),
               ('autumn', datetime(year, 9, 1).date(), datetime(year, 11, 30).date()),
               ('winter', datetime(year, 12, 1).date(), datetime(year, 12, 31).date())]
    if date.year % 4 == 0:  # leap year check
        seasons[0] = ('winter', datetime(year, 1, 1).date(), datetime(year, 2, 29).date(

    for season, start, end in seasons:
        if start <= date <= end:
            return season

    return 'Date is out of range'
```

In [10]:
```
# Convert date field to season
crashes_flag_df['SEASON'] = crashes_flag_df['CRASH_DATE'].apply(lambda x:
                                                    get_season(datetime.strptime(x[:10

# Group CRASH_DAY_OF_WEEK into weekdays and weekends
weekday_mask = [2,3,4,5,6]
crashes_flag_df['WEEKEND'] = crashes_flag_df['CRASH_DAY_OF_WEEK'].apply(lambda x: 0 if x

# Group hours into time blocks (extra bin at end to ensure correct bin treatment)
bins = [-1, 6, 9, 15, 19, 23, 24]
labels = ['night', 'morning_rush', 'midday', 'evening_rush', 'night', 'night']
crashes_flag_df['TIME_BLOCK'] = pd.cut(crashes_flag_df['CRASH_HOUR'], bins=bins, labels=

# Drop unnecessary columns
crashes_flag_df.drop(columns=['CRASH_DATE','CRASH_DAY_OF_WEEK','CRASH_HOUR'], axis=1, in
```

Category flags

In [11]:
```
device_mask = ['NO CONTROLS','FUNCTIONING PROPERLY']
weather_mask = ['CLEAR','UNKNOWN']
lighting_mask = ['DAYLIGHT']
alignment_mask = ['STRAIGHT AND LEVEL']
surface_mask = ['DRY','UNKNOWN']

crashes_flag_df['Malfunctioning_Device'] = crashes_flag_df['DEVICE_CONDITION'].apply(lam
crashes_flag_df['Inclement_Weather'] = crashes_flag_df['WEATHER_CONDITION'].apply(lambda
crashes_flag_df['Not_Daylight'] = crashes_flag_df['LIGHTING_CONDITION'].apply(lambda x:
crashes_flag_df['Not_Straight_Level'] = crashes_flag_df['ALIGNMENT'].apply(lambda x: 0 i
crashes_flag_df['Not_Dry_Surface'] = crashes_flag_df['ROADWAY_SURFACE_COND'].apply(lambd

# Flag for serious accidents (fatal + incapacitating), which is the target
crashes_flag_df['Target'] = crashes_flag_df.apply(lambda row: 1 if
                                                  (row['INJURIES_FATAL']+row['INJURIES_I
                                                  axis=1)

# Drop unnecessary columns
crashes_flag_df = crashes_flag_df.drop(columns=['DEVICE_CONDITION', 'WEATHER_CONDITION',
                                                 'ALIGNMENT', 'ROADWAY_SURFACE_COND',\
                                                'INJURIES_FATAL', 'INJURIES_INCAPACITATIN
```

In [12]:
```
# Masks to assist in binning the PHYSICAL_CONDITION and SAFETY_EQUIPMENT fields
PhysicalMask = ['NORMAL', 'UNKNOWN']
SafetyMask = ['SAFETY BELT USED', 'USAGE UNKNOWN', 'CHILD RESTRAINT USED', 'CHILD RESTRA
             'BICYCLE HELMET (PEDACYCLIST INVOLVED ONLY)', 'CHILD RESTRAINT - TYPE UNKNO
             'CHILD RESTRAINT - REAR FACING', 'HELMET USED', 'DOT COMPLIANT MOTORCYCLE H
             'BOOSTER SEAT', 'WHEELCHAIR', 'STRETCHER']

# Bin all problematic physical and safety conditions and tag with a 1
people_df['PHYSICAL_FLAG'] = people_df['PHYSICAL_CONDITION'].apply(lambda x: 0 if x in P
people_df['SAFETY_FLAG'] = people_df['SAFETY_EQUIPMENT'].apply(lambda x: 0 if x in Safet

# Drop the original columns
people_df = people_df.drop(columns=['PHYSICAL_CONDITION', 'SAFETY_EQUIPMENT'], axis=1)

# For each crash, tag if at least one element had a safety or physical problem
safety_flag = people_df.groupby('CRASH_RECORD_ID')['SAFETY_FLAG'].max().reset_index()
safety_flag.rename(columns={'SAFETY_FLAG': 'Poor_Safety_Behavior'}, inplace=True)
people_df = people_df.drop('SAFETY_FLAG', axis=1).merge(safety_flag, on='CRASH_RECORD_ID

physical_flag = people_df.groupby('CRASH_RECORD_ID')['PHYSICAL_FLAG'].max().reset_index(
physical_flag.rename(columns={'PHYSICAL_FLAG': 'Incapacitated_Person'}, inplace=True)
people_df = people_df.drop('PHYSICAL_FLAG', axis=1).merge(physical_flag, on='CRASH_RECOR

# Drop remaining duplicate rows
people_df.drop_duplicates(inplace=True)
```

Merge the DataFrames for modeling

In [13]:
```python
# Merge the dataframes
combined_df = crashes_flag_df.merge(people_df, on='CRASH_RECORD_ID', how='inner')

# Convert TIME_BLOCK and SEASON to type='category'
combined_df['TIME_BLOCK'] = combined_df['TIME_BLOCK'].astype('category')
combined_df['SEASON'] = combined_df['SEASON'].astype('category')

# Drop CRASH_RECORD_ID
combined_df.drop('CRASH_RECORD_ID', axis=1, inplace=True)

# Reset index
combined_df.reset_index(drop=True)
```

Out[13]:

	WITHIN_DISTRICT	SEASON	WEEKEND	TIME_BLOCK	Malfunctioning_Device	Inclement_Weather	Nc
0	1	summer	1	midday	0	0	
1	0	summer	0	evening_rush	0	0	
2	0	summer	1	midday	0	0	
3	0	summer	1	night	0	0	
4	0	autumn	0	midday	0	0	
...	...	...	...	...	...	...	
816539	0	summer	0	midday	0	0	
816540	1	spring	0	night	1	0	
816541	0	spring	0	evening_rush	0	0	
816542	0	spring	0	night	0	1	
816543	1	summer	0	night	0	1	

816544 rows × 12 columns

Modeling

In [14]:
```python
y = combined_df['Target']
X = combined_df.drop('Target', axis=1)

# Split the data into training and testing sets, stratify the split to ensure sufficient
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=
```

Baseline model

Our baseline model is a dummy model that optimizes using the majority class.

```
In [15]: dummy_clf = DummyClassifier(strategy='most_frequent')
         dummy_clf.fit(X_train, y_train)
         print(classification_report(y_test, dummy_clf.predict_proba(X_test)[:, 1]))
         print("Baseline ROC-AUC:", roc_auc_score(y_test, dummy_clf.predict_proba(X_test)[:, 1]))
```

```
C:\Users\Rick\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classificatio
n.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control th
is behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

           0       0.98      1.00      0.99    160364
           1       0.00      0.00      0.00      2945

    accuracy                           0.98    163309
   macro avg       0.49      0.50      0.50    163309
weighted avg       0.96      0.98      0.97    163309

Baseline ROC-AUC: 0.5
```

Baseline model conclusion: Due to the severe imbalance, a dummy model that optimizes using majority class results in 98.2% accuracy, but **0% recall** and an AUC score of 50% that is no better than random guessing.

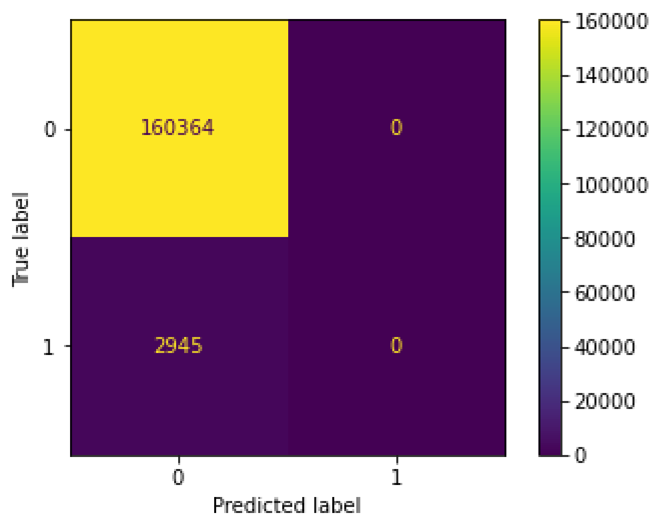
First simple model - logistic regression with three predictors

As a first test to improve on the baseline, this model uses a decision tree with three variables likely to be predictive:

- Weather: Clear, rain, snow, wind, etc
- Safety features: Seat belts, helmets, child car seats
- Driver condition: Intoxication, emotional distress, medication, drugs

In [16]:
```python
# Select variables for simple model
simple_cols = ['Inclement_Weather', 'Poor_Safety_Behavior', 'Incapacitated_Person']
X_train_simple = X_train[simple_cols]
X_test_simple = X_test[simple_cols]

# Instantiate a logistic regression
logreg_simple = LogisticRegression(random_state=1023)
logreg_simple.fit(X_train_simple, y_train)
y_pred = logreg_simple.predict(X_test_simple)

print(classification_report(y_test, y_pred))
print("Logistic Regression ROC-AUC:", roc_auc_score(y_test, logreg_simple.predict_proba(
ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred)).plot();
```

```
C:\Users\Rick\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classificatio
n.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control th
is behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

           0       0.98      1.00      0.99    160364
           1       0.00      0.00      0.00      2945

    accuracy                           0.98    163309
   macro avg       0.49      0.50      0.50    163309
weighted avg       0.96      0.98      0.97    163309

Logistic Regression ROC-AUC: 0.752348885911038
```



First simple model conclusion: This model does not successfully predict **any** positive cases, although the higher AUC suggests setting a lower probability threshold might lead to better predictions.

Logistic regression with many predictors

To improve predictive capabilities, the model includes many more predictors, including some categorical variables which must be encoded.

One Hot Encoding: One hot encoding changes categorical values into 1/0 columns and compares the power of each category against a reference category. Season and Time Block are categorical, and the 'spring' and 'midday' categories are dropped.

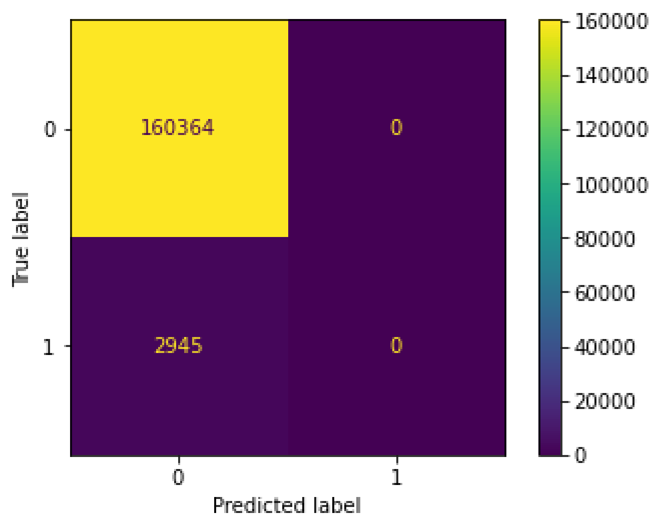
In [17]:
```
# One-hot encode categorical columns
ohe = OneHotEncoder(sparse=False, drop=['spring', 'midday'])
cat_columns = ['SEASON', 'TIME_BLOCK']

# Encode training data
X_train_ohe = ohe.fit_transform(X_train[cat_columns])
feature_names = ohe.get_feature_names(cat_columns)
X_train_ohe_df = pd.DataFrame(X_train_ohe, columns=feature_names, index=X_train.index)
X_train_final = pd.concat([X_train.drop(columns=cat_columns, axis=1), X_train_ohe_df], a

# Encode test data
X_test_ohe = ohe.transform(X_test[cat_columns])
feature_names = ohe.get_feature_names(cat_columns)
X_test_ohe_df = pd.DataFrame(X_test_ohe, columns=feature_names, index=X_test.index)
X_test_final = pd.concat([X_test.drop(columns=cat_columns, axis=1), X_test_ohe_df], axis
```

By adding many more predictors to the model, the predictive power should increase. This model includes:

- Weather: Clear, rain, snow, wind, etc.
- Safety features: Seat belts, helmets, child car seats
- Driver condition: Intoxication, emotional distress, medication, drugs
- Season: Each season as compared to <u>spring</u> as the dropped category
- Lighting: Daylight, darkness, dark with streetlights, etc.
- Time block: Morning rush, midday, evening rush, night-time, with <u>midday</u> as the dropped category
- SDA: Presence in a socially disadvantaged area
- Weekend: Weekend vs. weekday
- Road surface: Water, ice, mud, etc.
- Alignment: Road curves, cresting hill, unlevel, etc.
- Device condition: Broken lights, damaged signs, worn markings

In [18]:
```
# Instantiate a new logistic regression object
logreg_complex = LogisticRegression(random_state=1023)

# Fit the object on the encoded training data
logreg_complex.fit(X_train_final, y_train)

# Make predictions using the fitted model using the test data
y_pred_complex = logreg_complex.predict(X_test_final)

# Display results
print(classification_report(y_test, y_pred_complex))
print("Logistic Regression ROC-AUC:", roc_auc_score(y_test, logreg_complex.predict_proba
ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred_complex)).plot();
```

```
C:\Users\Rick\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classificatio
n.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control th
is behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

           0       0.98      1.00      0.99    160364
           1       0.00      0.00      0.00      2945

    accuracy                           0.98    163309
   macro avg       0.49      0.50      0.50    163309
weighted avg       0.96      0.98      0.97    163309

Logistic Regression ROC-AUC: 0.7743198993088686
```



Second model conclusion: That improved the ROC-AUC marginally, but the recall is still 0. The class imbalance is too great.

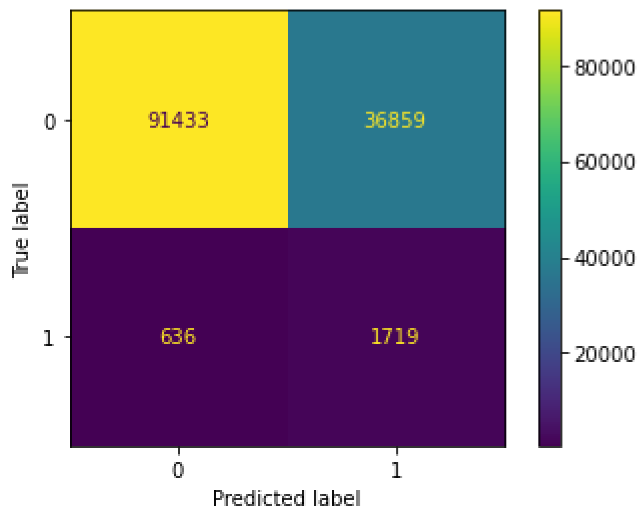
Logistic Regression model with SMOTE

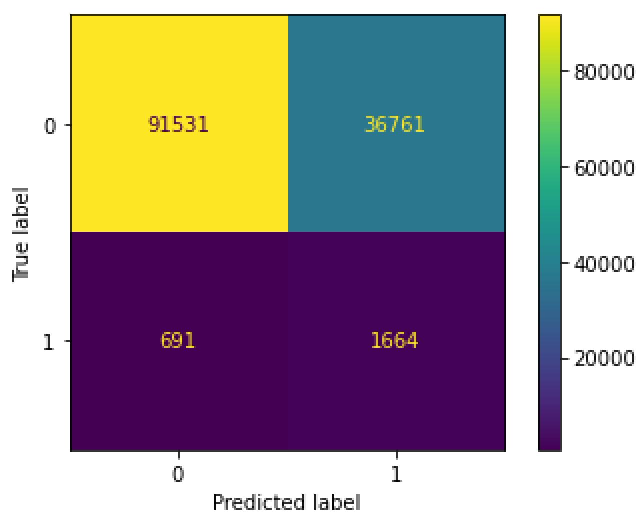
Synthetic Minority Oversampling TEchnique, or SMOTE, is used in order to address the class imbalance and train the model to detect the weaker signal. SMOTE generates new randomized data using the features of existing target variables. Because this is the final model, Stratified K Fold is used to cross validate as well.

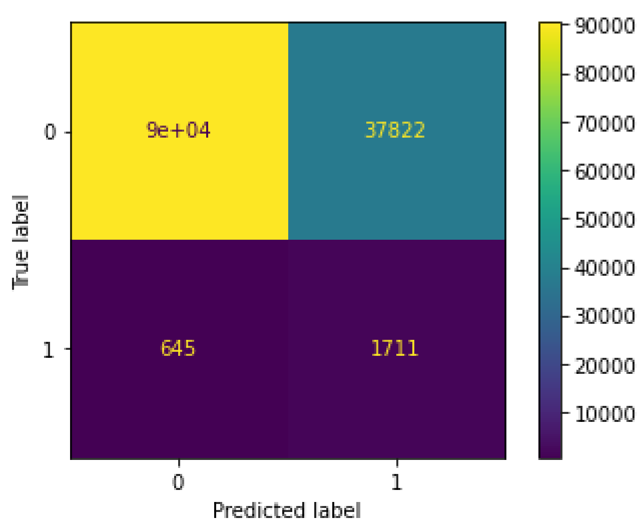
In [19]:
```
# Setup StratifiedKFold
skf = StratifiedKFold(n_splits=5, random_state=1023, shuffle=True)

# Initialize a list to store accuracy scores for each fold
recall_scores = []

i=0
# Iterate over each split
for train_index, val_index in skf.split(X_train_final, y_train):
    # Split the data
    X_train_fold, X_val_fold = X_train_final.iloc[train_index], X_train_final.iloc[val_i
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    # Apply SMOTE only to the training data in this fold
    smote = SMOTE(random_state=1023)
    X_train_smote, y_train_smote = smote.fit_resample(X_train_fold, y_train_fold)

    # Train the model
    logreg_fold = LogisticRegression(random_state=1023)
    logreg_fold.fit(X_train_smote, y_train_smote)

    # Validate the model
    y_pred_fold = logreg_fold.predict(X_val_fold)
    recall = recall_score(y_val_fold, y_pred_fold)
    recall_scores.append(recall)

    # Print results
    print(f"{i+1} Fold")
    i+=1
    print("-----------------")
    print(classification_report(y_val_fold, y_pred_fold))
    print("Logistic Regression ROC-AUC:", roc_auc_score(y_val_fold, logreg_complex.predi
    ConfusionMatrixDisplay(confusion_matrix(y_val_fold, y_pred_fold)).plot()
    print("")

# Print the average accuracy across all folds
print("Mean recall across all folds:", np.mean(recall_scores))
```

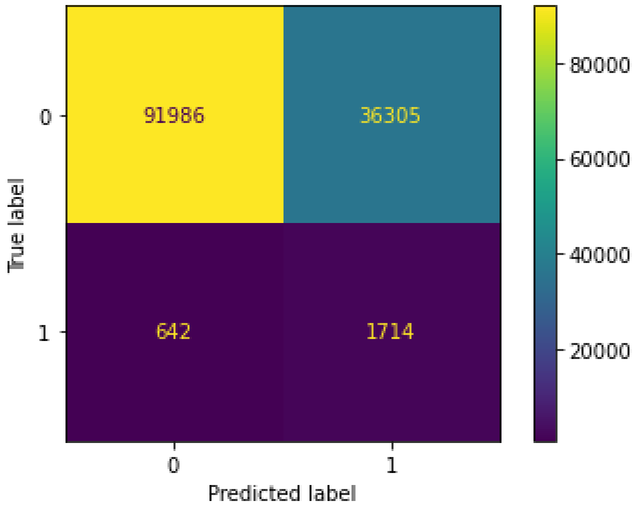
```
1 Fold
-----------------
              precision    recall  f1-score   support

           0       0.99      0.71      0.83    128292
           1       0.04      0.73      0.08      2355

    accuracy                           0.71    130647
   macro avg       0.52      0.72      0.46    130647
weighted avg       0.98      0.71      0.82    130647

Logistic Regression ROC-AUC: 0.7812778214348199

2 Fold
-----------------
              precision    recall  f1-score   support

           0       0.99      0.71      0.83    128292
           1       0.04      0.71      0.08      2355

    accuracy                           0.71    130647
   macro avg       0.52      0.71      0.46    130647
weighted avg       0.98      0.71      0.82    130647

Logistic Regression ROC-AUC: 0.7660627994139961

3 Fold
-----------------
              precision    recall  f1-score   support

           0       0.99      0.71      0.82    128291
           1       0.04      0.73      0.08      2356

    accuracy                           0.71    130647
   macro avg       0.52      0.72      0.45    130647
weighted avg       0.98      0.71      0.81    130647

Logistic Regression ROC-AUC: 0.7741543495151667

4 Fold
-----------------
              precision    recall  f1-score   support

           0       0.99      0.72      0.83    128291
           1       0.05      0.73      0.08      2356

    accuracy                           0.72    130647
   macro avg       0.52      0.72      0.46    130647
weighted avg       0.98      0.72      0.82    130647

Logistic Regression ROC-AUC: 0.7736953938506658

5 Fold
-----------------
              precision    recall  f1-score   support

           0       0.99      0.71      0.83    128291
           1       0.04      0.72      0.08      2356

    accuracy                           0.71    130647
   macro avg       0.52      0.72      0.46    130647
```

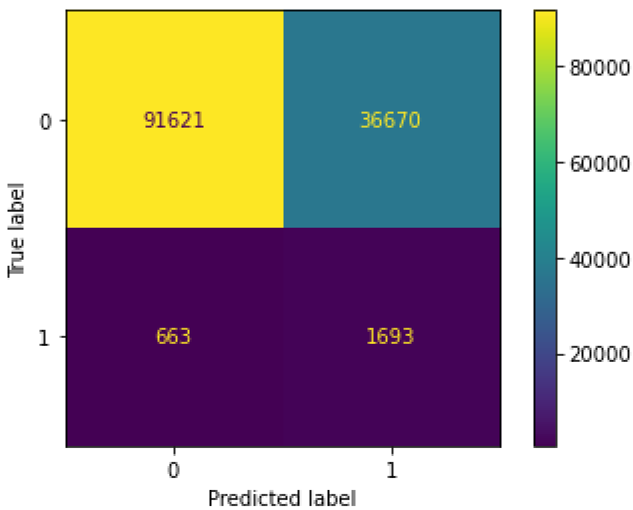
```
weighted avg       0.98      0.71      0.82    130647

Logistic Regression ROC-AUC: 0.766821737002593

Mean recall across all folds: 0.7217688045880059
```







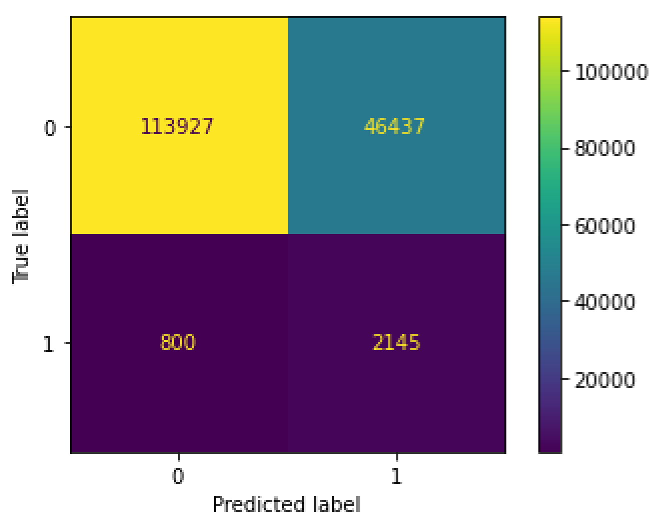




This model seems like it will perform really well on test data, since it is very stable across many folds. Finally, the model will train on all training data with SMOTE, and test on the test data.

In [20]:
```
# Instantiate a SMOTE object
smote = SMOTE(random_state=1023)

# Apply SMOTE on one-hot encoded *training* data
X_train_smote, y_train_smote = smote.fit_resample(X_train_final, y_train)

# Instantiate a new logistic regression object
logreg_smote = LogisticRegression(random_state=1023)

# Fit the object on the encoded training data
logreg_smote.fit(X_train_smote, y_train_smote)

# Make predictions using the fitted model
y_pred_smote = logreg_smote.predict(X_test_final)

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred_smote)

# Print results
print(classification_report(y_test, y_pred_smote))
print("ROC-AUC:", roc_auc_score(y_test, logreg_smote.predict_proba(X_test_final)[:, 1]))
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot();
```

```
              precision    recall  f1-score   support

           0       0.99      0.71      0.83    160364
           1       0.04      0.73      0.08      2945

    accuracy                           0.71    163309
   macro avg       0.52      0.72      0.46    163309
weighted avg       0.98      0.71      0.81    163309

ROC-AUC: 0.7722717913520933
```

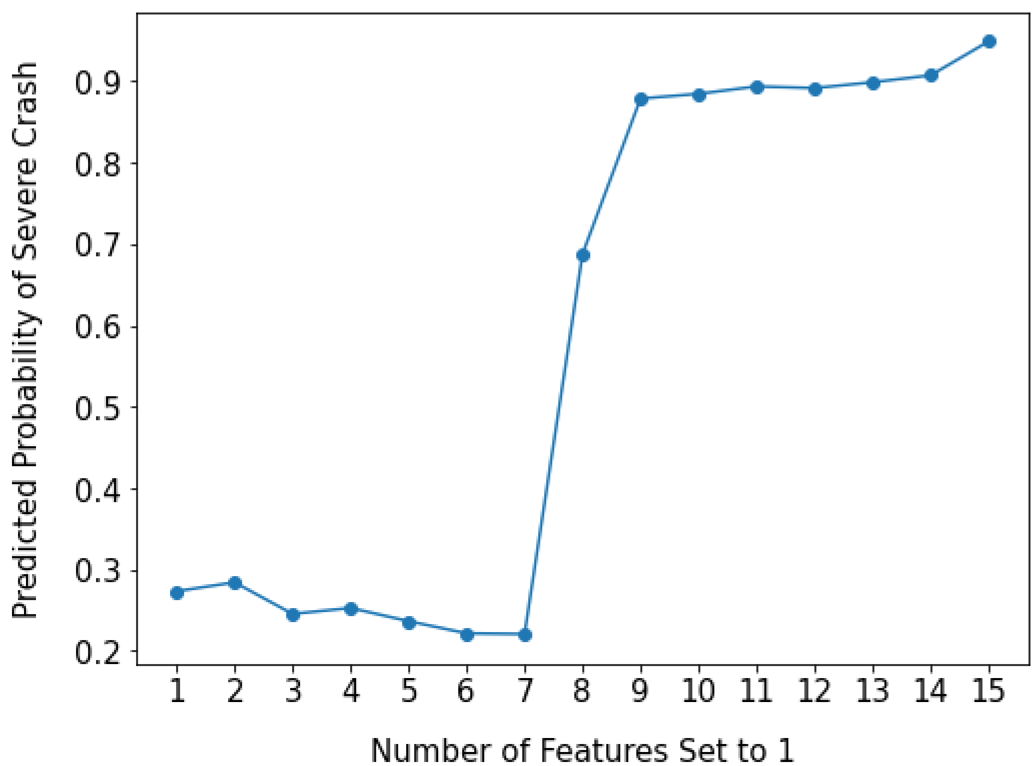


<u>Final complex model conclusions</u>: This is a very good model which finally captures **73% of true positives**. There are many more false positives as well, which is a predicted effect of improving recall. In this case, it is an acceptable trade-off. Finally, show the coefficients and calculated odds increase of each factor, and plot the effect on the odds of a severe accident.

In [21]:
```python
# Features, coefficients and increased odds
odds_effect = []
for coef in logreg_smote.coef_[0]:
    odds_effect.append(f"{round(100*(np.exp(coef)-1),1)}%")

coefficients = pd.DataFrame({
    'Feature': X_train_smote.columns,
    'Coefficient': logreg_smote.coef_[0],
    'Odds Increase': odds_effect
}).sort_values(by='Coefficient', ascending=False)

print(coefficients)
```

```
                  Feature  Coefficient Odds Increase
7      Poor_Safety_Behavior     2.051945        678.3%
8      Incapacitated_Person     1.189897        228.7%
14         TIME_BLOCK_night     0.641992         90.0%
0           WITHIN_DISTRICT     0.250086         28.4%
13  TIME_BLOCK_morning_rush     0.097271         10.2%
10            SEASON_summer     0.092347          9.7%
12  TIME_BLOCK_evening_rush     0.075787          7.9%
9             SEASON_autumn     0.055924          5.8%
1                   WEEKEND     0.054552          5.6%
3         Inclement_Weather     0.038844          4.0%
6           Not_Dry_Surface    -0.004982         -0.5%
11            SEASON_winter    -0.020861         -2.1%
5        Not_Straight_Level    -0.084764         -8.1%
4              Not_Daylight    -0.087671         -8.4%
2     Malfunctioning_Device    -0.199479        -18.1%
```

In [22]:
```
# Initialize all features to 0
feature_values = np.zeros((1, 15))

# Store probabilities when incrementally setting more features to 1
probabilities = []

for i in range(15):
    # Set the i-th feature to 1
    feature_values[0, i] = 1

    # Predict the probability with i features set to 1
    prob = logreg_smote.predict_proba(feature_values)[0, 1]
    probabilities.append(prob)

# Plot the results
plt.figure(figsize=(8, 6))
plt.plot(range(1, 16), probabilities, marker='o')  # Adjust the range to start at 1

plt.xlabel('Number of Features Set to 1', fontsize=15, labelpad=15)
plt.ylabel('Predicted Probability of Severe Crash', fontsize=15, labelpad=15)
plt.grid(False)
plt.xticks(range(1, 16), labels=[str(i) for i in range(1, 16)], fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```



Conclusions

Four factors seem to have a large effect:

- The use of safety features
- The driver's condition
- Whether the crash happened during the night

Potential future inquiry should focus on:

- Age of the driver
- Proximity of the crash to major holidays
- Geographic proximity of the crash to a hospital
- Whether the make/model of the car leads to differential outcomes (i.e. for EVs, pickups)
- Whether severe outcomes have different predictors for pedestrians, bicyclists and drivers