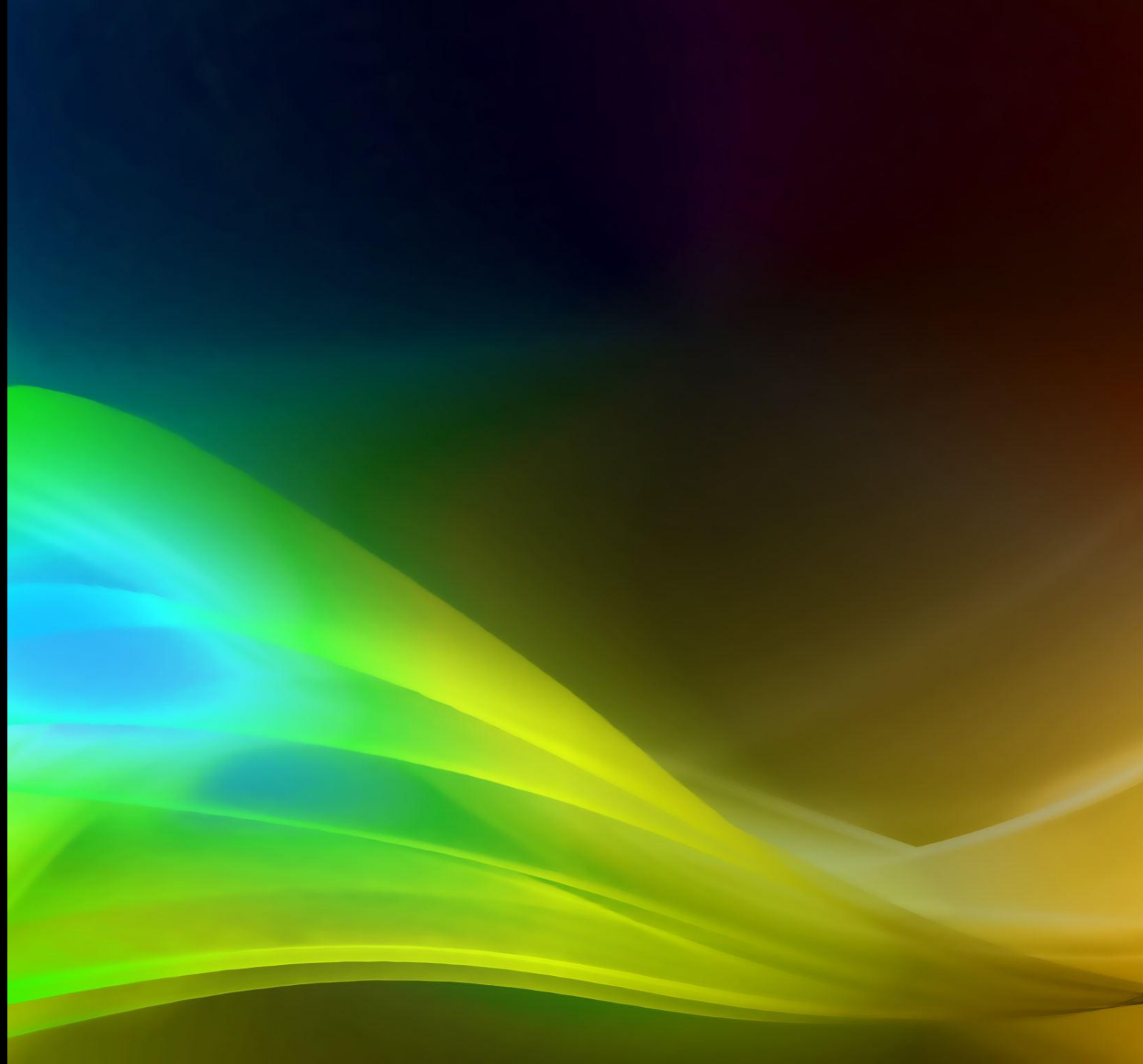
Car Accident Severity Prediction

Shubhaan Saxena, Jinyong Xie, Yash Sharma



Data Overview (US Accidents Dataset (2016-2023)

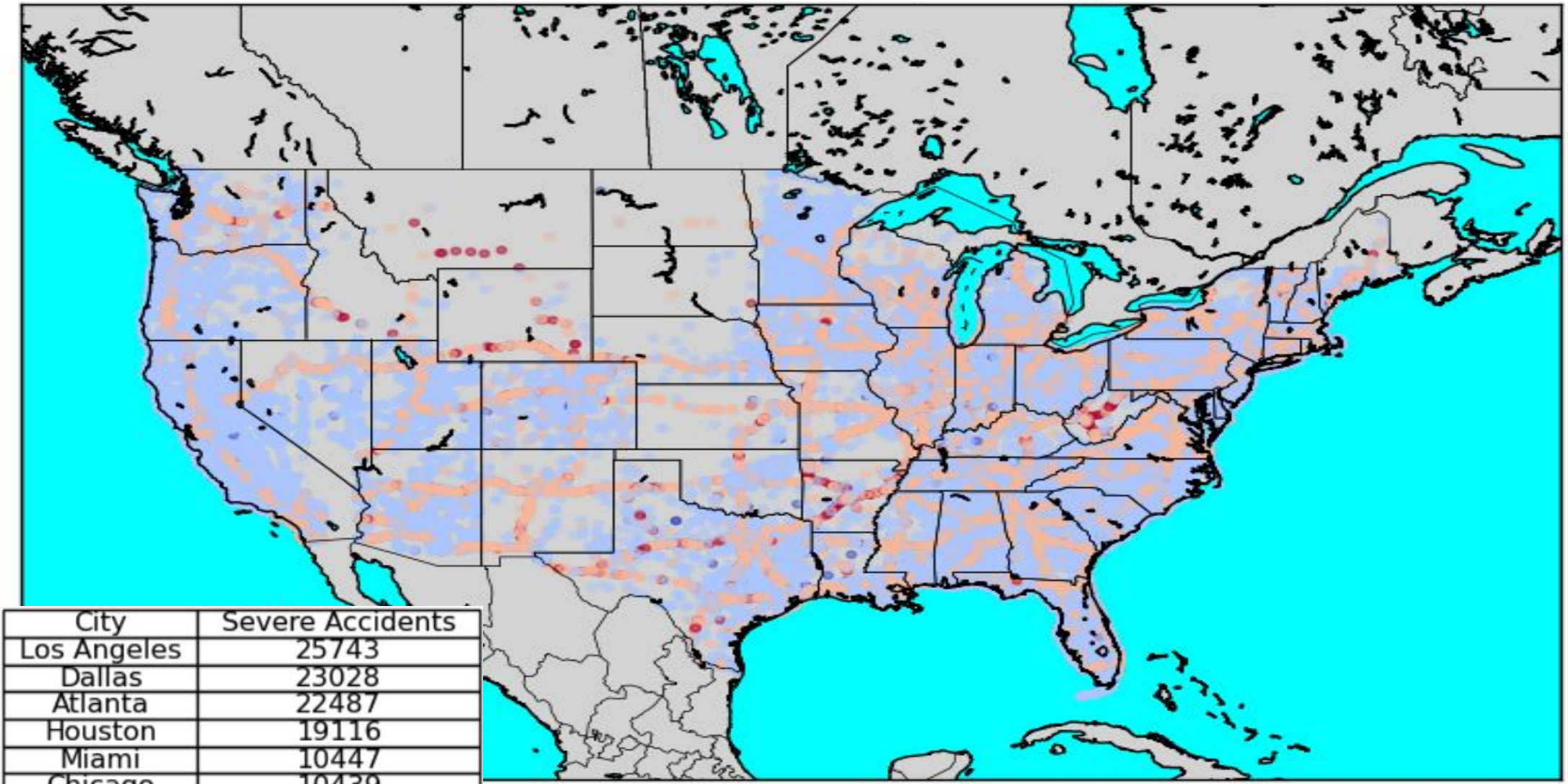
- Total rows: ~7.7 million accidents
- 45 columns
- Time Frame: February 2016 - March 2023
- Coverage 49 US states
- Data Source: Streaming traffic incident data from:
  - State Departments of Transportation
  - Law Enforcement Agencies
  - Traffic Cameras and Sensors

Column Name	Description
Severity (Target Variable)	Impact (1= minor, 4=major traffic impact)
State	State where the accident occured (ex. CA, TX, FL).
Weather Condition	(Rain, snow, fog, etc)
Temperature (F)	Temperature at time of the accident
Visibility (mi)	Visibility at the location
Wind Speed(mph)	Wind speed at the time of the crash
Wind Direction	(N, S, E, W)

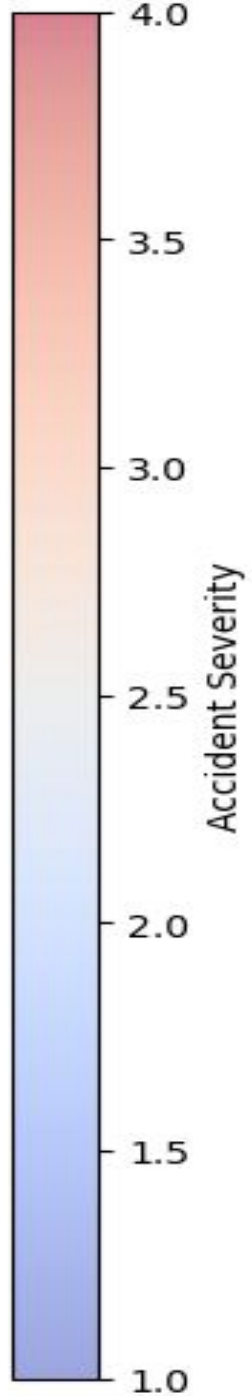
Project Implication & Business Application

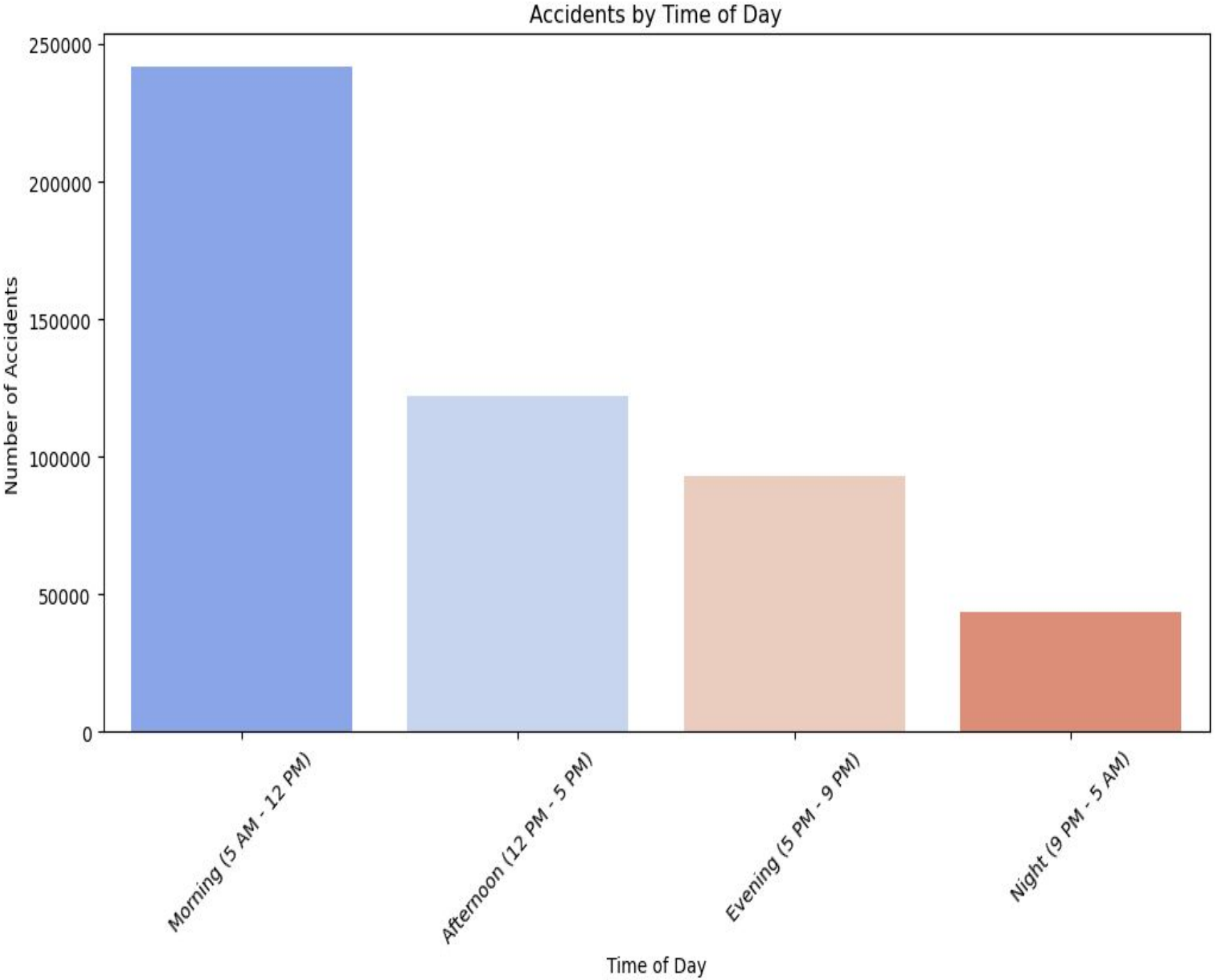
- Use Case: Predict accident severity based on weather conditions, location, and time of the day.
  - Insurance companies can spike rates on highly prone accident areas.
  - Traffic agencies can use accident density maps to design safer intersections.
  - Help in allocating emergency response units more efficiently
- Self-driving car companies (Tesla, Waymo) can train model accident scenarios to improve vehicle decision-making.
  - Companies like Uber, Lyft and google maps can use real-time accident prediction models to reroute drivers.
- Governments can analyze accident trends to propose safer regulations (road improvements, speed limits).

US Accidents (2016-2023) - Severity

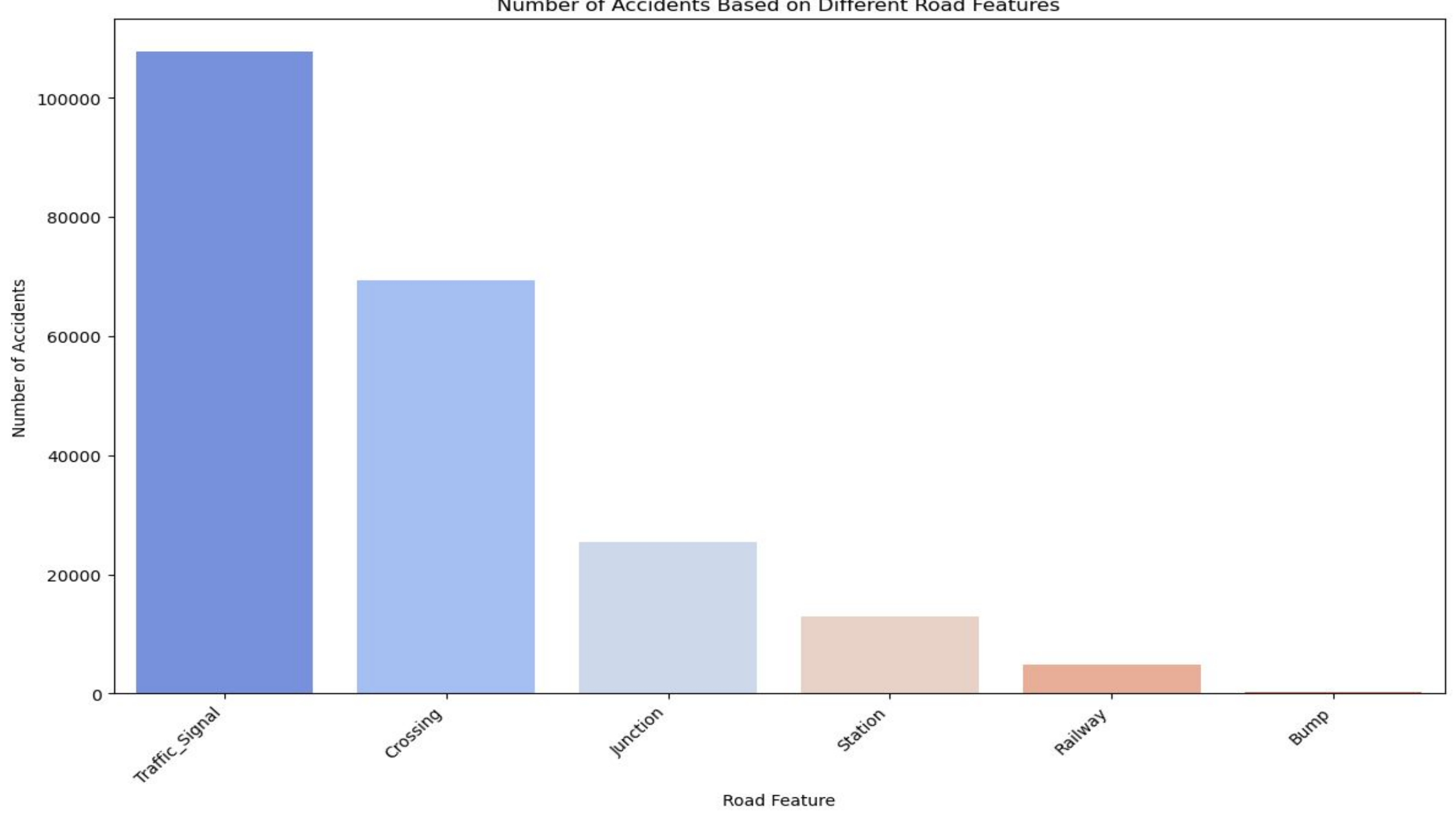


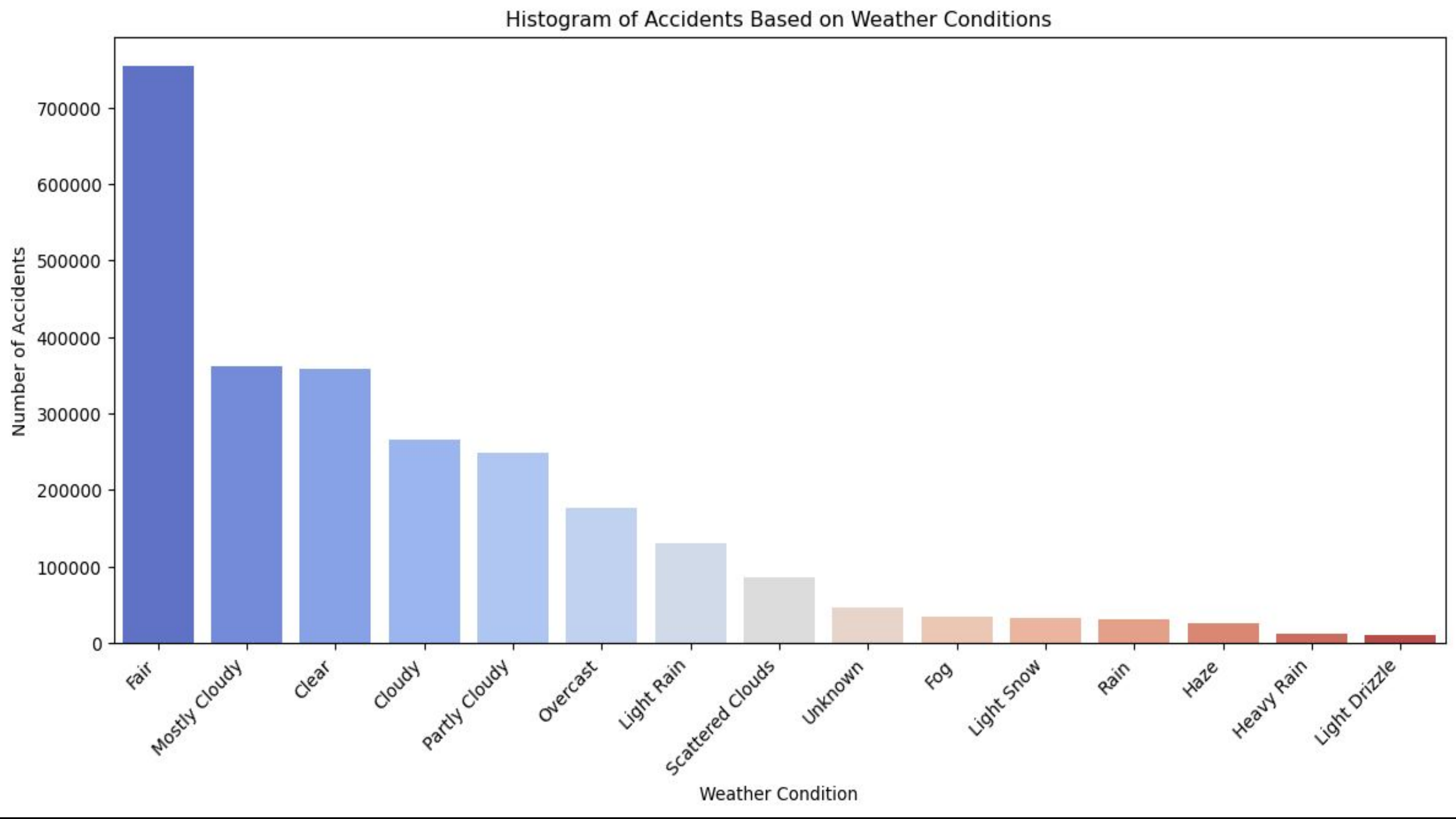
City	Severe Accidents
Los Angeles	25743
Dallas	23028
Atlanta	22487
Houston	19116
Miami	10447
Chicago	10439
Jacksonville	8982
San Diego	8742
Saint Louis	7343
Bronx	6800





Time of Day	Top State with Most Accidents
Afternoon (12 PM - 5 PM)	CA
Evening (5 PM - 9 PM)	CA
Morning (5 AM - 12 PM)	CA
Night (9 PM - 5 AM)	CA


Number of Accidents Based on Different Road Features
Number of Accidents
0
20000
40000
60000
80000
100000
Traffic_Signal
Crossing
Junction
Station
Railway
Bump
Road Feature


Histogram of Accidents Based on Weather Conditions
Number of Accidents
0
100000
200000
300000
400000
500000
600000
700000
Fair
Mostly Cloudy
Clear
Cloudy
Partly Cloudy
Overcast
Light Rain
Scattered Clouds
Unknown
Fog
Light Snow
Rain
Haze
Heavy Rain
Light Drizzle
Weather Condition

Cleaning & Preprocessing Overview

1. Drop columns with majority of missing value & imputation (numerical→ median;categorical→ “unknown”)
2. Datatype Conversation (eg. boolean, datetype)
3. Feature Engineering (hours, dayofweek,)
4. Target Variable Conversion (severity → severity binary)
5. StandardScaler (distance, temperature, humanity)
6. OneHotEncoder
7. SMOTE for oversampling training set

Drop columns with majority of missing value & imputation

```
df_nan.columns = ['Feature','Count']
drop = df_nan.loc[df_nan['Count']>250000]
droplist = drop.Feature.tolist()
droplist
```

```
['End_Lat', 'End_Lng', 'Wind_Chill(F)', 'Wind_Speed(mph)', 'Precipitation(in)']
```

```
df = df.drop(columns = droplist)
```

```
categorical_cols = df.select_dtypes(include=['object', 'category']).columns
df[categorical_cols] = df[categorical_cols].fillna("Unknown")
```

```
df[numeric_list] = df[numeric_list].fillna(df[numeric_list].median())
```

Datatype Conversation (eg. boolean, datetype)

```
# select all boolean datatype
boolean_columns = df_cleaned.select_dtypes(include=['bool'])
boolean_columns.info()
```

```
df_cleaned["Start_Time"] = df_cleaned["Start_Time"].str.split('.').str[0]  # Remove nanoseconds
df_cleaned["End_Time"] = df_cleaned["End_Time"].str.split('.').str[0]  # Remove nanoseconds

df_cleaned["Start_Time"] = pd.to_datetime(df_cleaned["Start_Time"])
df_cleaned["End_Time"] = pd.to_datetime(df_cleaned["End_Time"])
```

Feature Engineering

```
df_cleaned["Start_Hour"] = df_cleaned["Start_Time"].dt.hour
df_cleaned["Start_DayOfWeek"] = df_cleaned["Start_Time"].dt.dayofweek
df_cleaned["End_Hour"] = df_cleaned["End_Time"].dt.hour
```

Target Variable Conversation (severity → severity binary)

```python
# Define mapping for binomial classification
df_cleaned['Severity_Binary'] = df_cleaned['Severity'].map(lambda x: 0 if x in [1, 2] else 1)

# Verify class distribution
print(df_cleaned['Severity_Binary'].value_counts())

df_cleaned.drop(columns=['Severity'], inplace=True)
```

```
Severity_Binary
0    6224347
1    1504047
Name: count, dtype: int64
```

Deal with specific important feature before encoder

```python
# Define weather categories based on provided weather conditions
weather_categories = {
    'Clear': ['Clear', 'Fair', 'Mostly Clear'],
    'Cloudy': ['Partly Cloudy', 'Mostly Cloudy', 'Overcast', 'Cloudy'],
    'Rain': ['Light Rain', 'Rain', 'Heavy Rain', 'Rain Shower', 'Drizzle', 'Heavy Drizzle',
             'Light Drizzle', 'Light Rain Showers', 'Rain Showers', 'Heavy Rain Showers'],
    'Thunderstorm': ['Thunderstorms and Rain', 'Thunderstorm', 'T-Storm', 'Thunder',
                     'Heavy Thunderstorms and Rain', 'Thunder in the Vicinity',
                     'Thunder / Windy', 'Thunder and Hail', 'Thunder and Hail / Windy',
                     'Thunder / Wintry Mix'],
    'Snow': ['Light Snow', 'Snow', 'Heavy Snow', 'Light Snow Showers', 'Snow Showers',
             'Snow / Windy', 'Blowing Snow', 'Blowing Snow Nearby', 'Heavy Blowing Snow',
             'Snow and Thunder', 'Snow and Sleet', 'Light Snow and Sleet', 'Snow and Sleet / Windy',
             'Heavy Snow with Thunder', 'Heavy Snow / Windy'],
    'Fog/Haze': ['Haze', 'Fog', 'Shallow Fog', 'Mist', 'Patches of Fog', 'Light Fog',
                 'Fog / Windy', 'Dense Fog', 'Partial Fog', 'Patches of Fog / Windy',
                 'Shallow Fog / Windy'],
    'Windy/Dusty': ['Blowing Sand', 'Blowing Dust', 'Blowing Dust / Windy', 'Widespread Dust',
                    'Widespread Dust / Windy', 'Duststorm', 'Sand', 'Blowing Snow / Windy',
                    'Sand / Dust Whirlwinds', 'Sand / Dust Whirls Nearby', 'Sand / Windy',
                    'Heavy Duststorm'],
    'Icy/Freezing': ['Light Freezing Drizzle', 'Light Freezing Rain', 'Freezing Drizzle',
                     'Freezing Rain', 'Light Freezing Rain / Windy', 'Heavy Freezing Rain',
                     'Heavy Freezing Rain / Windy', 'Ice Pellets', 'Heavy Ice Pellets',
                     'Light Ice Pellets'],
    'Hail/Sleet': ['Hail', 'Small Hail', 'Light Hail', 'Heavy Sleet', 'Sleet',
                   'Light Sleet', 'Heavy Sleet / Windy', 'Sleet / Windy'],
    'Extreme': ['Tornado', 'Volcanic Ash', 'Squalls', 'Squalls / Windy', 'Funnel Cloud',
                'Dust Whirls', 'Heavy Smoke', 'Wintry Mix', 'Wintry Mix / Windy']
}
```

StandardScaler & OneHotEncoder

```
scaler = StandardScaler()
df_scaled = df_cleaned.copy()

df_scaled[numerical_cols] = scaler.fit_transform(df_cleaned[numerical_cols])
```

```
encoder = OneHotEncoder(handle_unknown='ignore')

# Fit and transform the categorical features for the training set
X_train_encoded = encoder.fit_transform(X_train[categorical_cols])
X_test_encoded = encoder.transform(X_test[categorical_cols])

# Convert sparse matrix to dense array
X_train_encoded = X_train_encoded.toarray()
X_test_encoded = X_test_encoded.toarray()

# Ensure that we get the correct feature names from the encoder
encoded_columns = encoder.get_feature_names_out(categorical_cols)

# Check the shape of the encoded arrays and column names to avoid mismatch
print("Shape of X_train_encoded:", X_train_encoded.shape)
print("Encoded columns:", encoded_columns)
```

SMOTE for oversampling

```python
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
# Apply SMOTE to the training data
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_encoded, y_train)
```

```python
y_train_resampled.value_counts()
y_train.value_counts()
```

```
Severity_Binary
0    128668
1    128668
Name: count, dtype: int64
Severity_Binary
0    128668
1     31332
Name: count, dtype: int64
```

Model Building

- We used the following models to see how results would vary
  - Logistic Regression
  - Random Forest
  - XGBoost
  - SVM
  - Neural Network
- We sampled 200,000 rows of our dataset.
- We used RandomizedSearchCV() to tune our models as GridSearchCV() was too exhaustive.

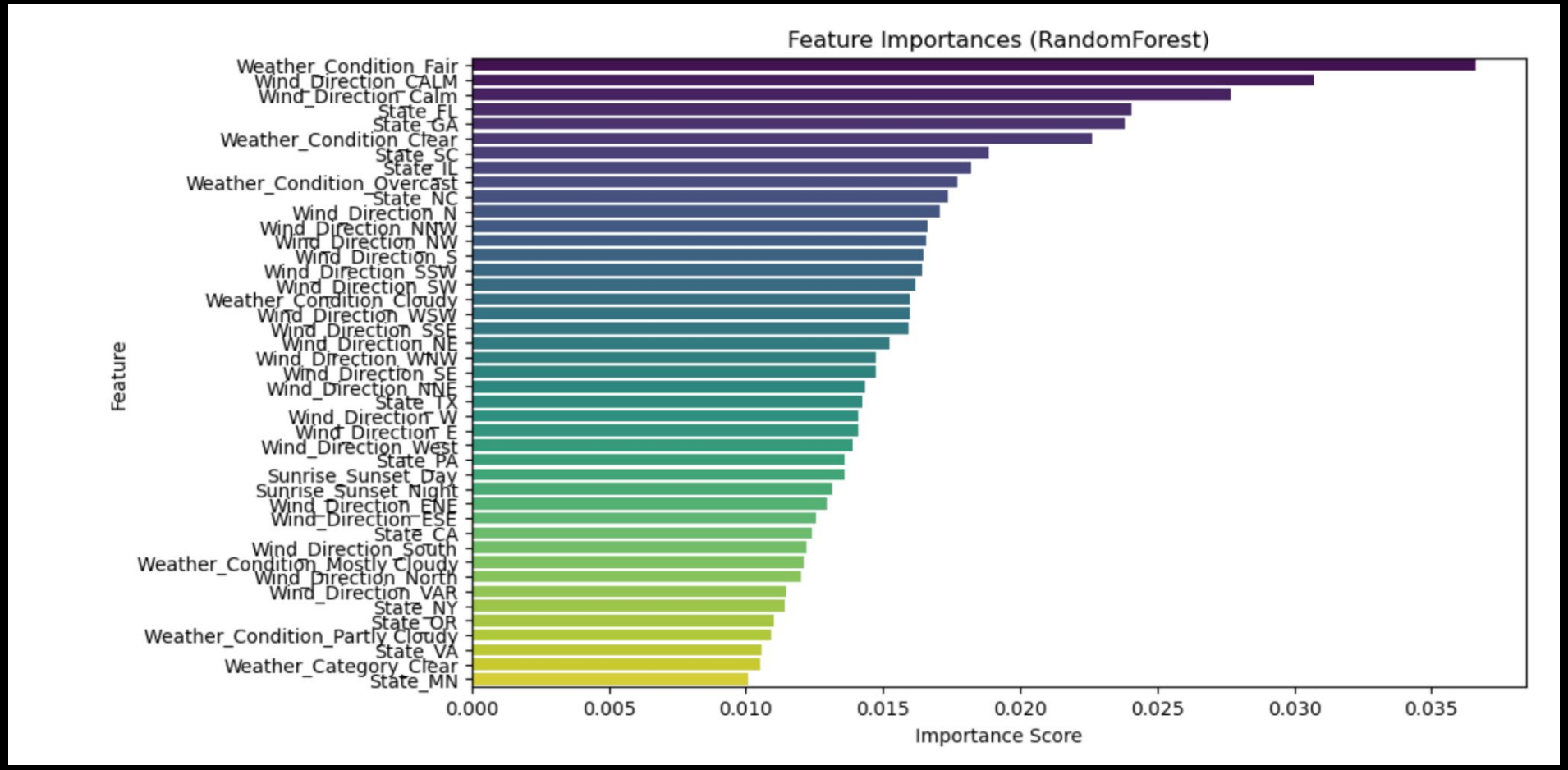
Logistic Regression

```
param_dist = {
    'C': np.logspace(-3, 3, 7),  # Regularization strength (log scale)
    'penalty': ['l2'],
    'solver': ['liblinear', 'saga'],  # Solvers for optimization
    'max_iter': [100, 200, 500, 1000]  # Number of iterations for convergence
}

# Initialize the model
model = LogisticRegression()

random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist,
                                   n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)

random_search.fit(X_train_resampled, y_train_resampled)

# Best parameters
print("Best Parameters from RandomizedSearchCV:", random_search.best_params_)

# Use the best estimator
best_model = random_search.best_estimator_

# predict
y_pred = best_model.predict(X_test_encoded)
```

Random Forest

```
# Define the distribution of hyperparameters to sample from
param_dist_rf = {
    'n_estimators': [100, 200, 500, 1000],  # Number of trees
    'max_depth': [10, 20, 30, None],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Min samples to split a node
    'min_samples_leaf': [1, 2, 4],  # Min samples in a leaf node
    'bootstrap': [True, False]  # Whether to use bootstrap sampling
}

# Initialize the model
rf_model = RandomForestClassifier(random_state=42)

# Initialize RandomizedSearchCV
random_search_rf = RandomizedSearchCV(estimator=rf_model, param_distributions=param_dist_rf,
                                      n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)

# Fit to training data
random_search_rf.fit(X_train_resampled, y_train_resampled)

# Best parameters
print("Best Parameters from RandomizedSearchCV (RF):", random_search_rf.best_params_)

# Use the best model f
best_rf_model = random_search_rf.best_estimator_

# predict
y_pred_rf = best_rf_model.predict(X_test_encoded)
```

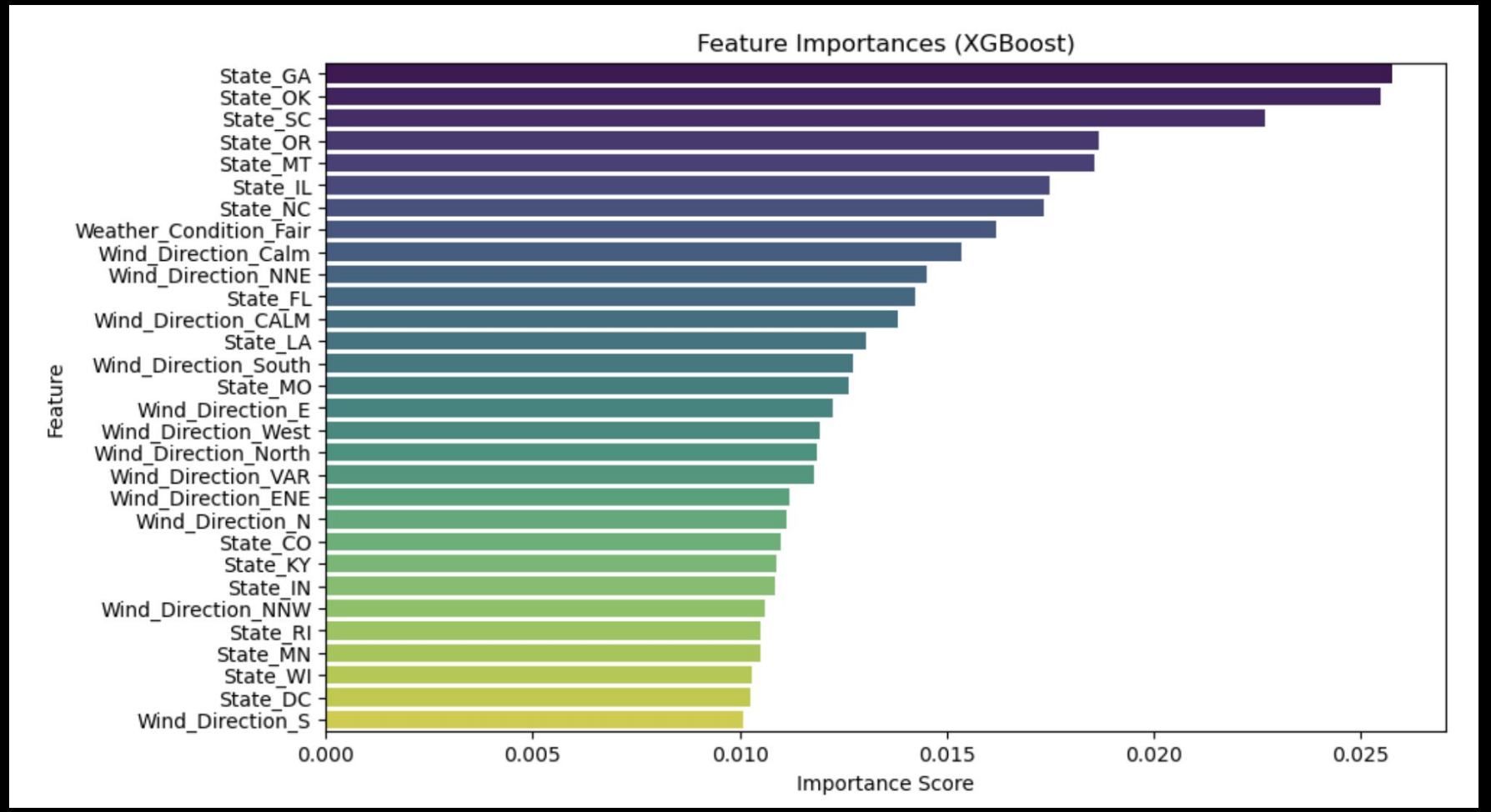
Random Forest



XGBoost

```
param_dist_xgb = {
    'n_estimators': [100, 200, 500, 1000],  # Number of boosting rounds
    'max_depth': [3, 6, 9, 12],  # Maximum depth of trees
    'learning_rate': [0.01, 0.05, 0.1, 0.2],  # Step size shrinkage
    'subsample': [0.6, 0.8, 1.0],  # Fraction of data used for training each tree
    'colsample_bytree': [0.6, 0.8, 1.0],  # Fraction of features used per tree
    'gamma': [0, 0.1, 0.2, 0.5],  # Minimum loss reduction required to make a split
    'reg_lambda': [1, 5, 10]  # L2 regularization term
}

# Initialize the model
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

# Initialize RandomizedSearchCV
random_search_xgb = RandomizedSearchCV(estimator=xgb_model, param_distributions=param_dist_xgb,
                                       n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)

# Fit to training data
random_search_xgb.fit(X_train_resampled, y_train_resampled)

# Best parameters
print("Best Parameters from RandomizedSearchCV (XGBoost):", random_search_xgb.best_params_)

# Use the best model
best_xgb_model = random_search_xgb.best_estimator_

# predict
y_pred_xgb = best_xgb_model.predict(X_test_encoded)
```

XGBoost


Feature Importances (XGBoost)
State_GA
State_OK
State_SC
State_OR
State_MT
State_IL
State_NC
Weather_Condition_Fair
Wind_Direction_Calm
Wind_Direction_NNE
State_FL
Wind_Direction_CALM
State_LA
Wind_Direction_South
State_MO
Wind_Direction_E
Wind_Direction_West
Wind_Direction_North
Wind_Direction_VAR
Wind_Direction_ENE
Wind_Direction_N
State_CO
State_KY
State_IN
Wind_Direction_NNW
State_RI
State_MN
State_WI
State_DC
Wind_Direction_S
Feature
0.000
0.005
0.010
0.015
0.020
0.025
Importance Score

SVM

```
# Define hyperparameter distribution
param_dist = {
    'C': np.logspace(-3, 3, 7),  # Regularization parameter
    'gamma': ['scale', 'auto'] + list(np.logspace(-3, 2, 6)),  # Kernel coefficient for RBF/Sigmoid
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],  # Types of kernel
    'degree': [2, 3, 4],  # Degree for polynomial kernel
}

# Initialize  model
svm_model = SVC()

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=svm_model, param_distributions=param_dist,
                                   n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)

# Fit to the training data
random_search.fit(X_train_resampled, y_train_resampled)

# Best parameters
print("Best Parameters from RandomizedSearchCV:", random_search.best_params_)

# Use the best model
best_svm_model = random_search.best_estimator_

# predict
y_pred = best_svm_model.predict(X_test_encoded)
```

Neural Network

```
# Define the Neural Network Model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_resampled.shape[1],)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')  # Sigmoid for binary classification
])

# Compile the model
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(X_train_resampled, y_train_resampled, epochs=20, batch_size=32, validation_data=(X_test_encoded,
                                                                                                     y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test_encoded, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

Evaluation

- Recall - true positive rate - focuses on minimizing false negatives and correctly predicting true positives.

Logistic Regression Results

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.68	0.77	16087
1	0.33	0.64	0.43	3913
accuracy			0.67	20000
macro avg	0.61	0.66	0.60	20000
weighted avg	0.78	0.67	0.70	20000

Random Forest Results

Classification Report (RF):

	precision	recall	f1-score	support
0	0.84	0.80	0.82	16087
1	0.32	0.39	0.35	3913
accuracy			0.72	20000
macro avg	0.58	0.59	0.59	20000
weighted avg	0.74	0.72	0.73	20000

XGBoost Results

Classification Report (XGBoost):

	precision	recall	f1-score	support
0	0.85	0.82	0.83	16087
1	0.34	0.39	0.36	3913
accuracy			0.73	20000
macro avg	0.59	0.60	0.60	20000
weighted avg	0.75	0.73	0.74	20000

SVM Results

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.84	0.83	1612
1	0.31	0.30	0.30	388
accuracy			0.73	2000
macro avg	0.57	0.57	0.57	2000
weighted avg	0.73	0.73	0.73	2000

Neural Network Results

	precision	recall	f1-score	support
0	0.87	0.74	0.80	16087
1	0.34	0.53	0.41	3913
accuracy			0.70	20000
macro avg	0.60	0.64	0.61	20000
weighted avg	0.76	0.70	0.72	20000

Evaluation

- Best Models
  - Recall: Logistic Regression
  - Overall Accuracy: XGBoost

- Future Considerations
  - Setting ‘scoring’ parameter to recall
  - Tuning the neural network

Summary

- Accidents are highly concentrated in urban areas, especially in major cities like Los Angeles, Dallas, Atlanta, Houston, and Miami.
  - high traffic congestion
  - frequent stop and go traffic
  - Infrastructure challenges
  - The midwest and Northern states show fewer accidents due to lower traffic density
- Morning rush hours contribute to the most accidents, requiring better urban planning, traffic control, and public awareness.
- The most important features are Wind Direction, Weather Conditions, and State.
- A logistic regression or a neural network should be used to best predict the severity of a car accident.

Thank you!
Questions?