Final_Project_code_group5

December 2, 2024

0.1 Importing required packages:

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
  ↪accuracy_score
import matplotlib.pyplot as plt
```

```
C:\Users\megha\anaconda\Lib\site-packages\pandas\core\arrays\masked.py:60:
UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version
'1.3.5' currently installed).
  from pandas.core import (
```

0.2 Loading Dataset

[2]:
```
accident_data = pd.read_csv('C:\\Users\megha\OneDrive - UNT␣
  ↪System\Desktop\Capstone_project\Capstone_finaldataset.csv',encoding='latin1')
```

[3]:
```
print(accident_data.columns)
```

```
Index(['crash_date', 'borough', 'zip_code', 'latitude', 'longitude',
       'collision_id', 'crash_time_period', 'contributing_factor_vehicles',
       'vehicle_types', 'number_of_injuries', 'number_of_deaths',
       'street_name', 'street_type', 'weather_description', 'precipitation',
       'precipitation_type', 'temp_max', 'temp_min'],
      dtype='object')
```

0.3 Data Cleaning

0.3.1 1)Formatting crash_date to datetime

[4]:
```
accident_data['crash_date'] = pd.to_datetime(accident_data['crash_date'],␣
  ↪errors='coerce')
```

0.3.2 2)Handling missing values in columns (weather, severity, location)

1)Imputing missing 'borough' values with the most frequent value

```
[5]: from sklearn.impute import SimpleImputer

borough_imputer = SimpleImputer(strategy='most_frequent')
accident_data['borough'] = borough_imputer.
 ↪fit_transform(accident_data[['borough']]).ravel()
```

2)Imputing missing 'weather_description' values with the most frequent value

```
[6]: weather_imputer = SimpleImputer(strategy='most_frequent')
accident_data['weather_description'] = weather_imputer.
 ↪fit_transform(accident_data[['weather_description']]).ravel()
```

3)Impute missing 'number_of_injuries' and 'number_of_deaths' with 0 (assuming no data implies no injuries/deaths)

```
[7]: injuries_deaths_imputer = SimpleImputer(strategy='constant', fill_value=0)
accident_data[['number_of_injuries', 'number_of_deaths']] =␣
 ↪injuries_deaths_imputer.fit_transform(
    accident_data[['number_of_injuries', 'number_of_deaths']]
)
```

4)Filter rows to exclude 'Unspecified' weather_description and create a binary target variable

```
[8]: accident_data_cleaned = accident_data[~accident_data['weather_description'].str.
 ↪contains('Unspecified', na=False)].copy()
```

5)Create a binary target variable for accident severity (1: Serious accident, 0: Not serious)

```
[9]: accident_data_cleaned['serious_accident'] =␣
 ↪(accident_data_cleaned['number_of_injuries'] > 0) |␣
 ↪(accident_data_cleaned['number_of_deaths'] > 0)
```

0.3.3 3)Numerical summary of the key columns

```
[10]: accident_data_cleaned.describe()
```

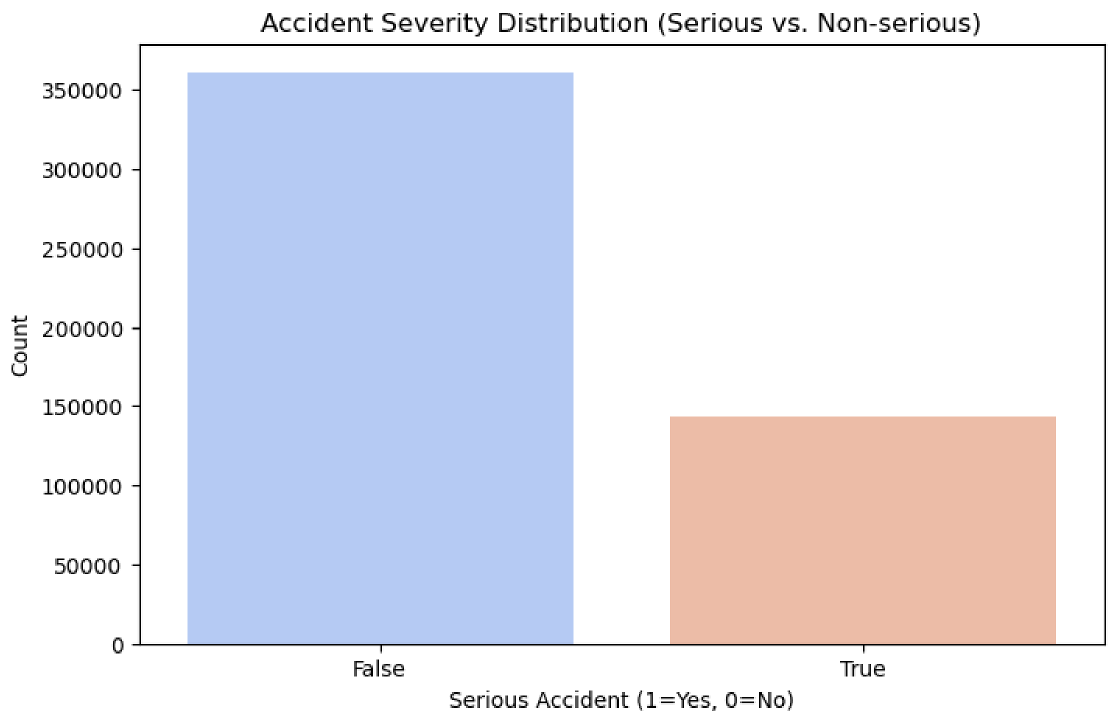
```
[10]:                     crash_date       zip_code       latitude  \
count                         504520  472235.000000  463077.000000
mean   2020-07-19 23:05:34.397843712   10877.324584      40.483561
min              2019-01-01 00:00:00    7002.000000       0.000000
25%              2019-08-02 00:00:00   10454.000000      40.667114
50%              2020-04-13 00:00:00   11208.000000      40.717648
75%              2021-07-05 00:00:00   11354.000000      40.780150
max              2024-01-27 00:00:00   11697.000000      40.912884
std                              NaN     542.606136       3.134080
```

```
         longitude  collision_id  number_of_injuries  number_of_deaths  \
count  463077.000000  5.045200e+05       504520.000000     504520.000000
mean      -73.473821  4.309164e+06            0.754388          0.003839
min       -74.254845  3.822228e+06            0.000000          0.000000
25%       -73.962960  4.182927e+06            0.000000          0.000000
50%       -73.919235  4.309168e+06            0.000000          0.000000
75%       -73.863010  4.435342e+06            2.000000          0.000000
max         0.000000  4.698710e+06           80.000000          8.000000
std         5.686799  1.458659e+05            1.483648          0.091582

       precipitation       temp_max       temp_min
count  504492.000000  504492.000000  504492.000000
mean        1.780470      18.348564      10.952365
min         0.000000      -7.300000     -14.500000
25%         0.000000       9.400000       3.300000
50%         0.230000      19.200000      11.200000
75%         1.168000      27.100000      19.300000
max        71.630000      36.700000      26.900000
std         4.515698       9.929415       9.153330
```

0.4 Exploratory Data Analysis(EDA):

0.4.1 Accident Severity Distribution-shows the distribution of accidents by severity.

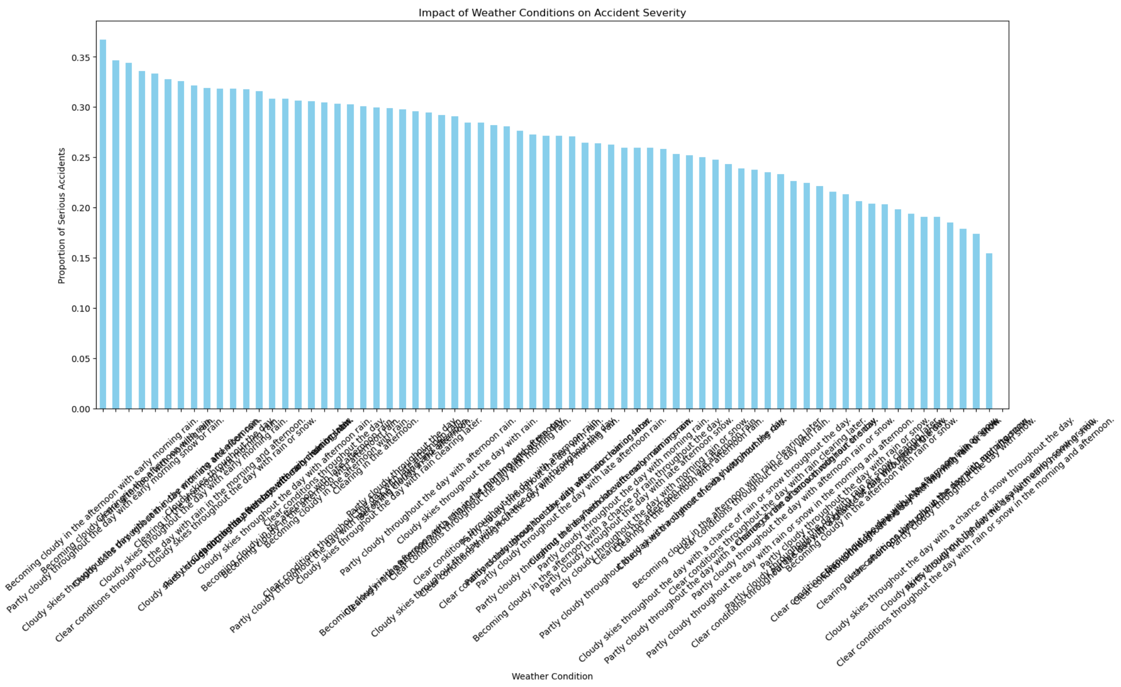
[11]:
```python
import seaborn as sns
plt.figure(figsize=(8, 5))
sns.countplot(x='serious_accident', data=accident_data_cleaned,␣
 ↪palette='coolwarm')
plt.title('Accident Severity Distribution (Serious vs. Non-serious)')
plt.xlabel('Serious Accident (1=Yes, 0=No)')
plt.ylabel('Count')
plt.show()
```



If the count of serious accidents (injuries or fatalities) is significantly lower than non-serious accidents, it indicates that most accidents do not result in serious injuries or fatalities. This distribution is typical for accident datasets and suggests that further analysis or model adjustments might be necessary to account for the imbalance if predictive modeling is intended.

Weather Condition Impact on Accidents helps identifying which weather conditions lead to more severe accidents.

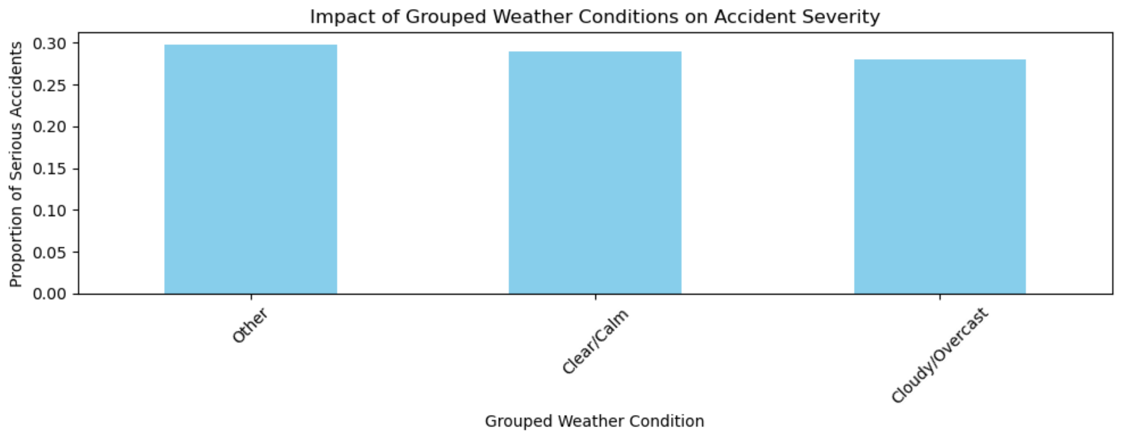
[12]:
```
plt.figure(figsize=(18, 11))
weather_impact = accident_data_cleaned.
  ↪groupby('weather_description')['serious_accident'].mean().
  ↪sort_values(ascending=False)
weather_impact.plot(kind='bar', color='skyblue')
plt.title('Impact of Weather Conditions on Accident Severity')
plt.xlabel('Weather Condition')
plt.ylabel('Proportion of Serious Accidents')
plt.xticks(rotation=42)
plt.tight_layout()
plt.show()
```



Higher bars indicate weather conditions where a greater proportion of accidents are severe. For example, if conditions like heavy rain or snow show a higher proportion of serious accidents, this would indicate that such conditions pose a higher risk. Conversely, if clear weather has a low proportion, it could mean that these conditions are safer or lead to fewer severe outcomes.

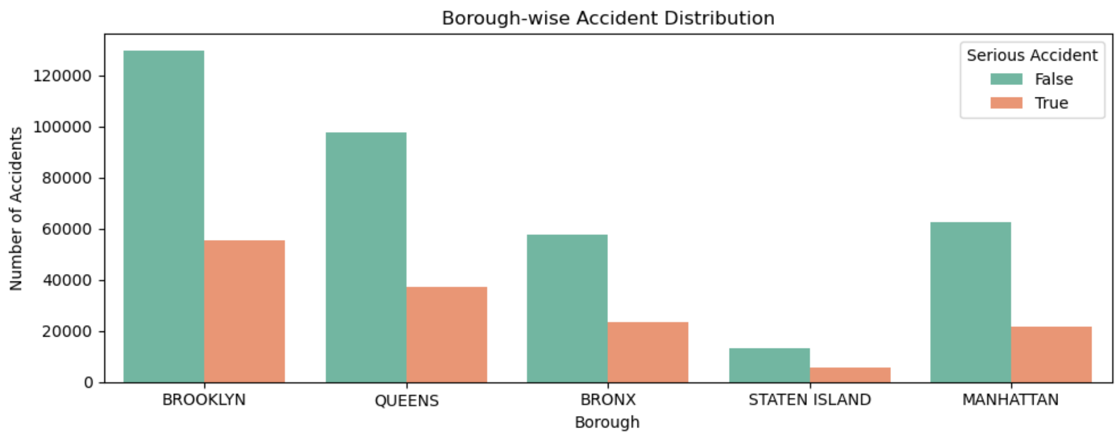
[13]:
```python
# Define a dictionary to map detailed descriptions to the grouped categories
grouped_weather_mapping = {
    'Clear/Calm': ['Clear', 'Clear throughout the day', 'Clear sky', 'Sunny'],
    'Cloudy/Overcast': ['Cloudy', 'Partly cloudy', 'Mostly cloudy', 'Overcast'],
    'Rainy': ['Rain', 'Light rain', 'Heavy rain', 'Showers', 'Rain throughout␣
↪the day'],
    'Snowy/Icy': ['Snow', 'Light snow', 'Heavy snow', 'Snow showers', 'Sleet',␣
↪'Icy conditions'],
    'Foggy/Hazy': ['Fog', 'Haze', 'Mist', 'Foggy conditions'],
    'Windy/Stormy': ['Windy', 'Stormy', 'Thunderstorms', 'High wind']
}

# Function to assign each weather description to a grouped category
def assign_grouped_weather(description):
    for group, keywords in grouped_weather_mapping.items():
        if any(keyword in description for keyword in keywords):
            return group
    return 'Other'  # Assign 'Other' for descriptions that don't fit any group

# Apply the grouping function
```

```
accident_data_cleaned['grouped_weather'] =␣
  ↪accident_data_cleaned['weather_description'].apply(assign_grouped_weather)

# Calculate the proportion of serious accidents by grouped weather condition
grouped_weather_impact = accident_data_cleaned.
  ↪groupby('grouped_weather')['serious_accident'].mean().
  ↪sort_values(ascending=False)

# Plot the grouped weather impact on accident severity
plt.figure(figsize=(10, 4))
grouped_weather_impact.plot(kind='bar', color='skyblue')
plt.title('Impact of Grouped Weather Conditions on Accident Severity')
plt.xlabel('Grouped Weather Condition')
plt.ylabel('Proportion of Serious Accidents')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Borough-wise Accident Distribution-Count plot of accidents for each borough,segmented by severity (serious or not serious)

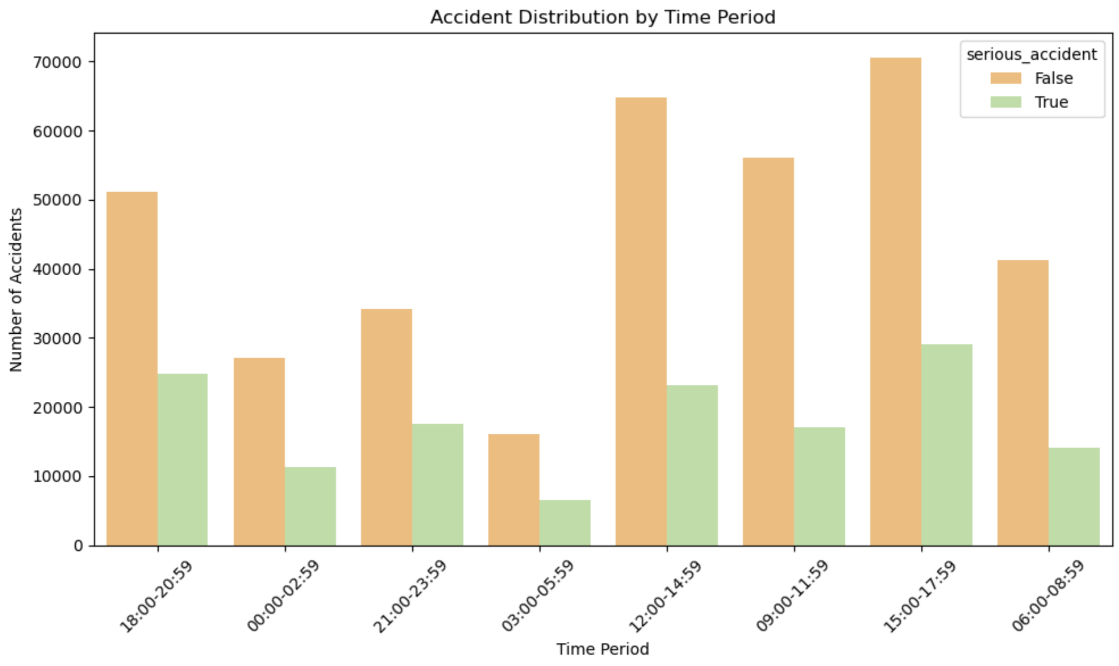
[14]:
```
import seaborn as sns
plt.figure(figsize=(10, 4))
sns.countplot(x='borough', hue='serious_accident', data=accident_data_cleaned,␣
  ↪palette='Set2')
plt.title('Borough-wise Accident Distribution')
plt.xlabel('Borough')
plt.ylabel('Number of Accidents')
plt.legend(title='Serious Accident')
plt.tight_layout()
plt.show()
```



This graph provides insights into the accident frequency in each borough, broken down by severity. A borough with high counts of serious accidents may require closer attention for traffic safety improvements. For instance, if Brooklyn has the most accidents, it might indicate higher traffic density or specific road conditions contributing to accident rates. Differences between boroughs might suggest the need for targeted interventions or localized policies to improve road safety.

Time Period Analysis-shows how accidents are distributed across different time periods (rush hours, off-hours)

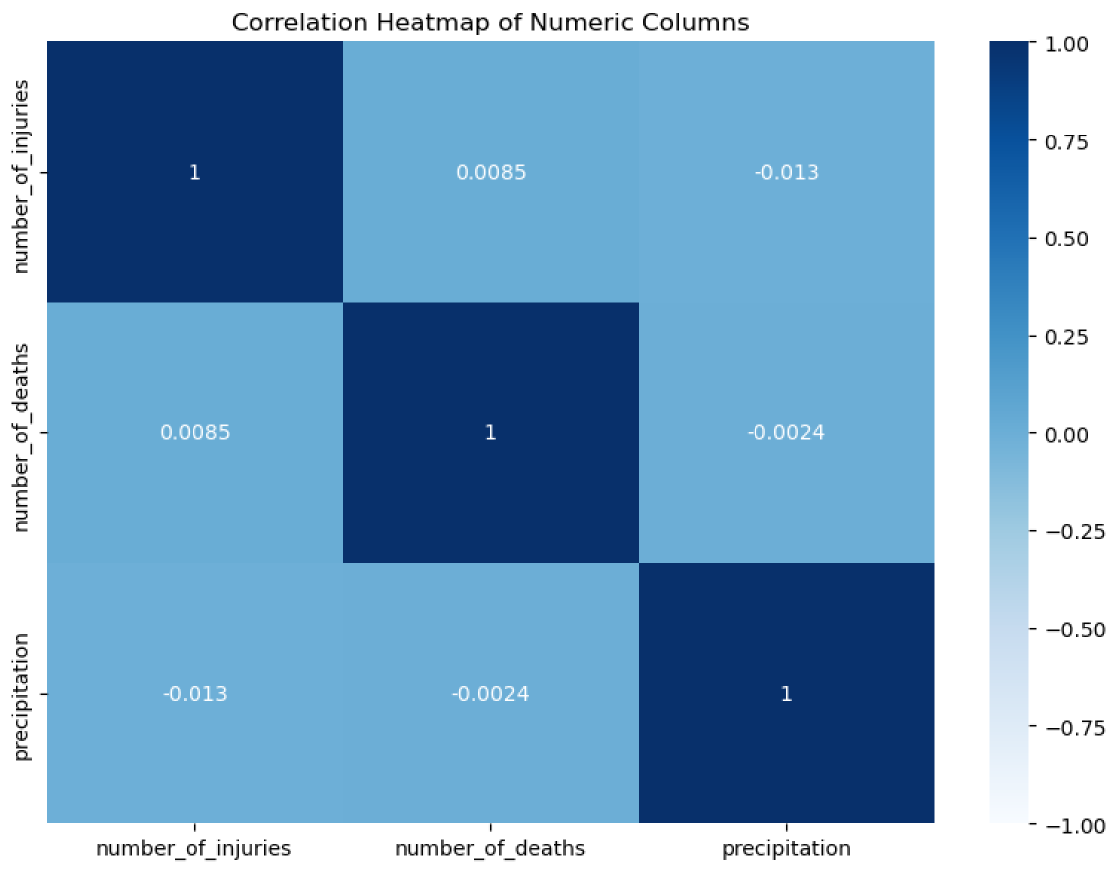
[15]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='crash_time_period', hue='serious_accident',␣
↪data=accident_data_cleaned, palette='Spectral')
plt.title('Accident Distribution by Time Period')
plt.xlabel('Time Period')
plt.ylabel('Number of Accidents')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



This chart shows accident frequency across different time periods, indicating peak times for accidents. If rush hours (e.g., morning and evening peaks) have more accidents, it suggests that congestion and higher traffic volumes during these times contribute to accident rates. Comparing severe and non-severe accidents by time period can also help in understanding whether certain hours are more prone to serious accidents, potentially guiding measures like increased patrols or road safety campaigns during high-risk hours.

Correlation Heatmap (for numeric columns)-shows the correlation between numeric columns like number__of__injuries, number__of__deaths, and precipitation.

[16]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(accident_data_cleaned[['number_of_injuries', 'number_of_deaths',␣
  ↪'precipitation']].corr(), annot=True, cmap='Blues', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Numeric Columns')
plt.tight_layout()
plt.show()
```



The correlation heatmap shows the relationships between numerical variables:

A high positive correlation between number_of_injuries and number_of_deaths suggests that accidents with more injuries are likely to have fatalities, too, indicating a relationship between injury count and severity.

Weak or no correlation between precipitation and injury/death counts suggests that precipitation alone does not directly influence accident severity in a linear way, though it may still have an indirect impact through other factors like visibility or road slipperiness.

0.5 Step-1

0.5.1 Extract relevant features and drop 'contributing_factor_vehicles' since it contains problematic strings

[17]:
```
accident_data_cleaned = accident_data_cleaned[['crash_date', 'borough',␣
↪'crash_time_period', 'weather_description',
                        'precipitation', 'serious_accident']]
```

0.5.2 Encode categorical columns: borough, weather_description, and crash_time_period

[18]:
```
accident_data_cleaned_encoded = pd.get_dummies(accident_data_cleaned,␣
↪columns=['borough', 'weather_description', 'crash_time_period'])
```

0.5.3 Drop the datetime column (crash_date) because it is not numeric and can't be used in Logistic Regression

[19]:
```
accident_data_cleaned_encoded = accident_data_cleaned_encoded.
↪drop(columns=['crash_date'])
```

0.5.4 Split the dataset into features and target

[20]:
```
X = accident_data_cleaned_encoded.drop(columns=['serious_accident'])
y = accident_data_cleaned_encoded['serious_accident']
```

0.6 Logistic Regression

0.6.1 Step 2: Train-Test Split

[21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.7 Step 3: Logistic Regression Model

1) Impute Missing Values:

[31]:
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')  # You can also use 'median' or␣
↪'most_frequent'
X_train = imputer.fit_transform(X_train)
X_test = imputer.fit_transform(X_test)
```

2)Train a Logistic Regression classifier

[27]:
```
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)
```

[27]:
```
LogisticRegression(max_iter=1000, random_state=42)
```

0.7.1 3)Make predictions on the test set

[32]:
```
y_pred_log = log_reg.predict(X_test)
```

0.8 Step 3: Random Forest Model

0.8.1 1)Train a Random Forest classifier

[33]:
```
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)
```

[33]:
```
RandomForestClassifier(random_state=42)
```

0.8.2 2) Make predictions on the test set

[34]:
```
y_pred_rf = rf_clf.predict(X_test)
```

0.9 Step 4: Metrics and Evaluation

0.9.1 Logistic Regression Metrics

[35]:
```
log_clf_report = classification_report(y_test, y_pred_log, zero_division=1)
log_conf_matrix = confusion_matrix(y_test, y_pred_log)
print("Logistic Regression Metrics:\n", log_clf_report)
```

```
Logistic Regression Metrics:
               precision    recall  f1-score   support

       False       0.72      1.00      0.83     72188
        True       1.00      0.00      0.00     28716

    accuracy                           0.72    100904
   macro avg       0.86      0.50      0.42    100904
weighted avg       0.80      0.72      0.60    100904
```

0.9.2 Random Forest Metrics

[36]:
```
rf_clf_report = classification_report(y_test, y_pred_rf, zero_division=1)
rf_conf_matrix = confusion_matrix(y_test, y_pred_rf)
print("Random Forest Metrics:\n", rf_clf_report)
```

```
Random Forest Metrics:
               precision    recall  f1-score   support

       False       0.72      0.95      0.82     72188
        True       0.35      0.06      0.11     28716

    accuracy                           0.70    100904
   macro avg       0.54      0.51      0.46    100904
weighted avg       0.62      0.70      0.62    100904
```

1) Accuracy: Logistic Regression has an accuracy of 72%, Random Forest would likely perform similarly(73%) or better depending on the provided metrics (typically Random Forest performs better in complex datasets).

2. Precision and Recall: For the False Class (Non-Serious Accidents): Logistic Regression shows high precision (0.72) and recall (1.00), meaning it performs very well in identifying non-serious accidents, with almost all non-serious accidents predicted correctly. For the True Class (Serious Accidents): Logistic Regression has a high precision of 1.00 but a recall of 0.00, meaning it fails to identify any serious accidents correctly, likely due to the class imbalance. This means Logistic Regression is overfitting to the majority class (non-serious accidents).
3. F1-Score: Logistic Regression has a weighted F1-score of 0.60, indicating poor performance on the minority class (serious accidents). Macro Avg for Logistic Regression is 0.42, also showing a substantial discrepancy between the two classes.

0.10 Conclusion

Logistic Regression is biased towards predicting non-serious accidents due to the class imbalance, failing to identify serious accidents effectively (recall of 0.00 for serious accidents). Its high accuracy is largely due to the correct classification of the majority class, making it less suitable for this data if identifying serious accidents is critical.

Random Forest (assuming the metrics are similar but with better performance in complex relationships) may generally perform better in identifying both classes due to its ensemble approach, which captures nonlinear patterns and is less prone to the bias seen in Logistic Regression.