Import the necessary modules.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
```

Import the Crashes dataset.

In [2]:
```
with zipfile.ZipFile("crashes.csv.zip") as z1:
    with z1.open("crashes.csv") as crashes:
        df_crashes = pd.read_csv(crashes)
```

In [3]:
```
df_crashes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 416872 entries, 0 to 416871
Data columns (total 49 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   CRASH_RECORD_ID                416872 non-null  object 
 1   RD_NO                          416813 non-null  object 
 2   CRASH_DATE_EST_I               31821 non-null   object 
 3   CRASH_DATE                     416868 non-null  object 
 4   POSTED_SPEED_LIMIT             416868 non-null  float64
 5   TRAFFIC_CONTROL_DEVICE         416868 non-null  object 
 6   DEVICE_CONDITION               416868 non-null  object 
 7   WEATHER_CONDITION              416868 non-null  object 
 8   LIGHTING_CONDITION             416868 non-null  object 
 9   FIRST_CRASH_TYPE               416867 non-null  object 
 10  TRAFFICWAY_TYPE                416867 non-null  object 
 11  LANE_CNT                       126015 non-null  float64
 12  ALIGNMENT                      416867 non-null  object 
 13  ROADWAY_SURFACE_COND           416867 non-null  object 
```

Filter the dataset to fit the business problem.

We've decided that we only want to focus on crash data that has rainy weather conditions.

In [4]:
```
# Filter the dataset to only include crashes when it is raining
df_crashes = df_crashes.loc[df_crashes["WEATHER_CONDITION"] == 'RAIN']
df_crashes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36006 entries, 12 to 416854
Data columns (total 49 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   CRASH_RECORD_ID                36006 non-null  object 
 1   RD_NO                          35999 non-null  object 
 2   CRASH_DATE_EST_I               2270 non-null   object 
 3   CRASH_DATE                     36006 non-null  object 
 4   POSTED_SPEED_LIMIT             36006 non-null  float64
 5   TRAFFIC_CONTROL_DEVICE         36006 non-null  object 
 6   DEVICE_CONDITION               36006 non-null  object 
 7   WEATHER_CONDITION              36006 non-null  object 
 8   LIGHTING_CONDITION             36006 non-null  object 
 9   FIRST_CRASH_TYPE               36006 non-null  object 
 10  TRAFFICWAY_TYPE                36006 non-null  object 
 11  LANE_CNT                       11627 non-null  float64
 12  ALIGNMENT                      36006 non-null  object 
 13  ROADWAY_SURFACE_COND           36006 non-null  object 
```

In the following code block, we account for missing values, and ultimately drop them from the target column, which we have decided to be injury severity.

In [5]:
```
# Dropping null values from the target variable column
df_crashes = df_crashes.loc[df_crashes["MOST_SEVERE_INJURY"].notna()]
df_crashes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 35894 entries, 12 to 416854
Data columns (total 49 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   CRASH_RECORD_ID                35894 non-null  object 
 1   RD_NO                          35887 non-null  object 
 2   CRASH_DATE_EST_I               2248 non-null   object 
 3   CRASH_DATE                     35894 non-null  object 
 4   POSTED_SPEED_LIMIT             35894 non-null  float64
 5   TRAFFIC_CONTROL_DEVICE         35894 non-null  object 
 6   DEVICE_CONDITION               35894 non-null  object 
 7   WEATHER_CONDITION              35894 non-null  object 
 8   LIGHTING_CONDITION             35894 non-null  object 
 9   FIRST_CRASH_TYPE               35894 non-null  object 
 10  TRAFFICWAY_TYPE                35894 non-null  object 
 11  LANE_CNT                       11598 non-null  float64
 12  ALIGNMENT                      35894 non-null  object 
 13  ROADWAY_SURFACE_COND           35894 non-null  object 
```

FEATURE ENGINEERING FOR THE BINOMIAL TARGET VARIABLE MODELING

We've decided to utilize both binomial and multinomial models to see which model would be the best fit.

So first, we convert the target column to binary, where 0 would indicate no injury in crash, and 1 would simply indicate an injury, disregarding severity level.

In [6]:
```
# Making most severe injury column binary
injury_binary = {'NO INDICATION OF INJURY': 0, 'NONINCAPACITATING INJURY'

df_crashes["INJURY_BINARY"] = df_crashes["MOST_SEVERE_INJURY"].map(injury
df_crashes["INJURY_BINARY"].value_counts()
# Worth noting that it is very imbalanced.
```

Out[6]:
```
0    30036
1     5858
Name: INJURY_BINARY, dtype: int64
```

Going further, we've decided that we're only interested in lighting conditions, crash hour, crash day of week, and crash month.

In [7]:
```
# Further filtering for the columns that we still have interest in
col_names = ['LIGHTING_CONDITION', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CR

df_crashes = df_crashes[col_names]
df_crashes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 35894 entries, 12 to 416854
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   LIGHTING_CONDITION  35894 non-null  object 
 1   CRASH_HOUR          35894 non-null  float64
 2   CRASH_DAY_OF_WEEK   35894 non-null  float64
 3   CRASH_MONTH         35894 non-null  float64
 4   MOST_SEVERE_INJURY  35894 non-null  object 
 5   INJURY_BINARY       35894 non-null  int64  
dtypes: float64(3), int64(1), object(2)
memory usage: 1.9+ MB
```

In [8]:
```
# Double-checking that there aren't any null values
df_crashes.isna().sum()
```

Out[8]:
```
LIGHTING_CONDITION    0
CRASH_HOUR            0
CRASH_DAY_OF_WEEK     0
CRASH_MONTH           0
MOST_SEVERE_INJURY    0
INJURY_BINARY         0
dtype: int64
```

EDA.

Explore the unique values of each categorical column.

In [9]:
```
print("\nUnique values in the 'LIGHTING_CONDITION' column:")
print(df_crashes['LIGHTING_CONDITION'].value_counts())

print("\nUnique values in the 'MOST_SEVERE_INJURY' column:")
print(df_crashes['MOST_SEVERE_INJURY'].value_counts())
```

```
Unique values in the 'LIGHTING_CONDITION' column:
DAYLIGHT                  19727
DARKNESS, LIGHTED ROAD    10780
DARKNESS                   2455
DUSK                       1556
DAWN                        999
UNKNOWN                     377
Name: LIGHTING_CONDITION, dtype: int64

Unique values in the 'MOST_SEVERE_INJURY' column:
NO INDICATION OF INJURY     30036
NONINCAPACITATING INJURY     3280
REPORTED, NOT EVIDENT        1821
INCAPACITATING INJURY         716
FATAL                          41
Name: MOST_SEVERE_INJURY, dtype: int64
```

Explore the unique values of each discrete numerical column.

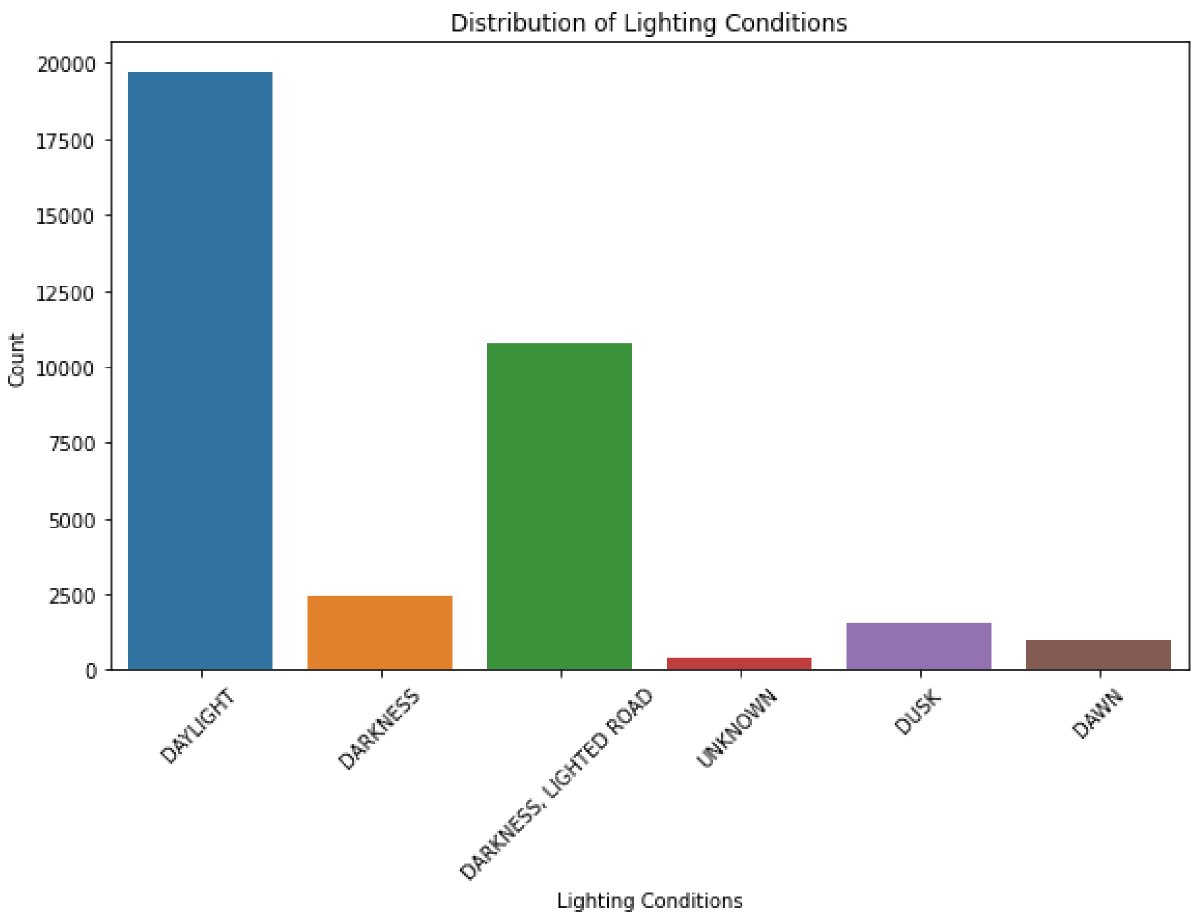
In [10]:
```
print("Unique values in the 'CRASH_HOUR' column:")
print(df_crashes['CRASH_HOUR'].value_counts().sort_index())

print("\nUnique values in the 'CRASH_DAY_OF_WEEK' column:")
print(df_crashes['CRASH_DAY_OF_WEEK'].value_counts().sort_index())

print("\nUnique values in the 'CRASH_MONTH' column:")
print(df_crashes['CRASH_MONTH'].value_counts().sort_index())
```

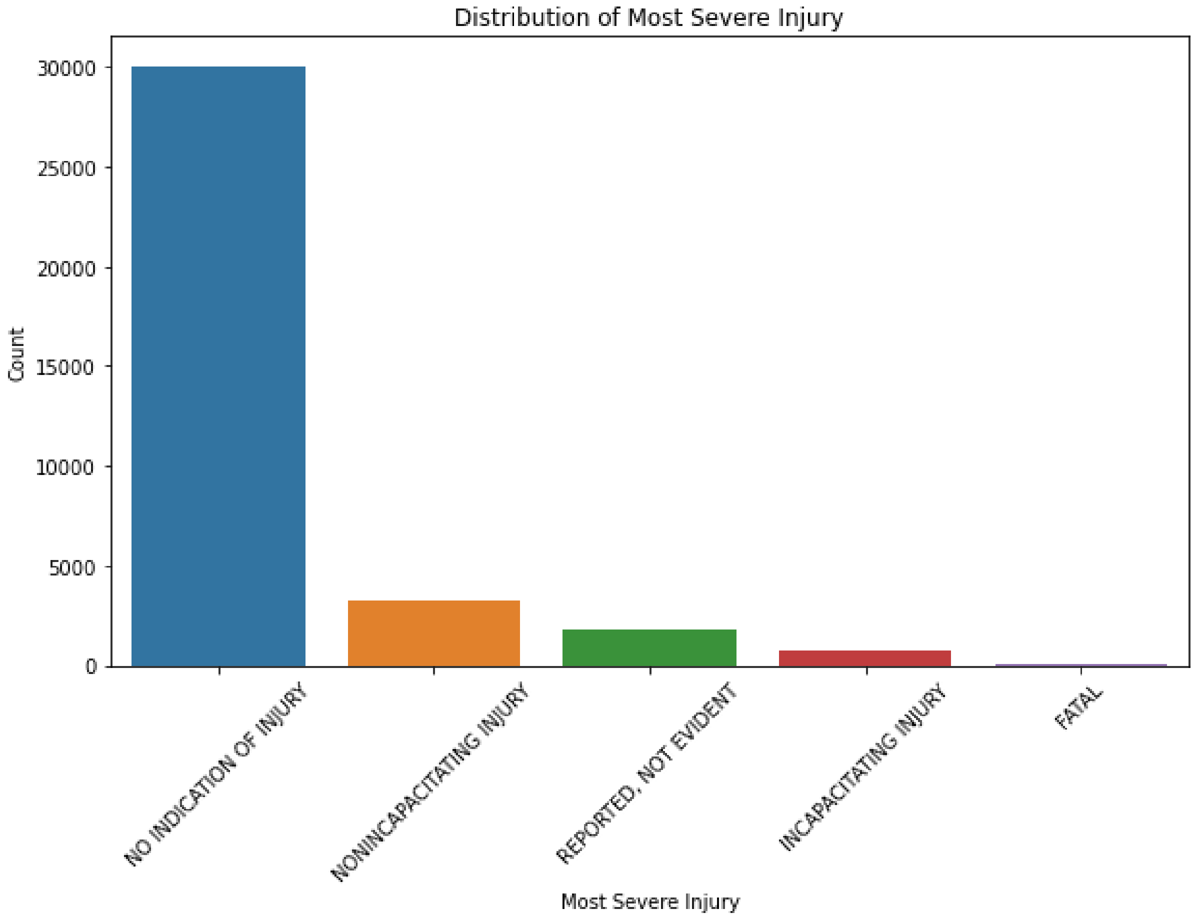
```
Unique values in the 'CRASH_HOUR' column:
0.0      848
1.0      789
2.0      663
3.0      568
4.0      524
5.0      606
6.0      963
7.0     1655
8.0     1867
9.0     1646
10.0    1470
11.0    1588
12.0    1772
13.0    1869
14.0    1929
15.0    2379
16.0    2525
17.0    2600
```

Make countplots for the unique values of each categorical column.

Per our filtered dataframe, the only categorical columns that exist are lighting conditions and injury severities.

In [11]:
```
# Countplot for 'LIGHTING_CONDITION' with adjusted spacing between bars
plt.figure(figsize = (10, 6))
sns.countplot(data = df_crashes, x = 'LIGHTING_CONDITION', dodge = True)
plt.title('Distribution of Lighting Conditions')
plt.xlabel('Lighting Conditions')
plt.ylabel('Count')
plt.xticks(rotation = 45)  # Rotate x-axis labels if needed
plt.show()

# Countplot for 'MOST_SEVERE_INJURY' with adjusted spacing between bars
plt.figure(figsize = (10, 6))
sns.countplot(data = df_crashes, x = 'MOST_SEVERE_INJURY', dodge = True)
plt.title('Distribution of Most Severe Injury')
plt.xlabel('Most Severe Injury')
plt.ylabel('Count')
plt.xticks(rotation = 45)  # Rotate x-axis labels if needed
plt.show()
```

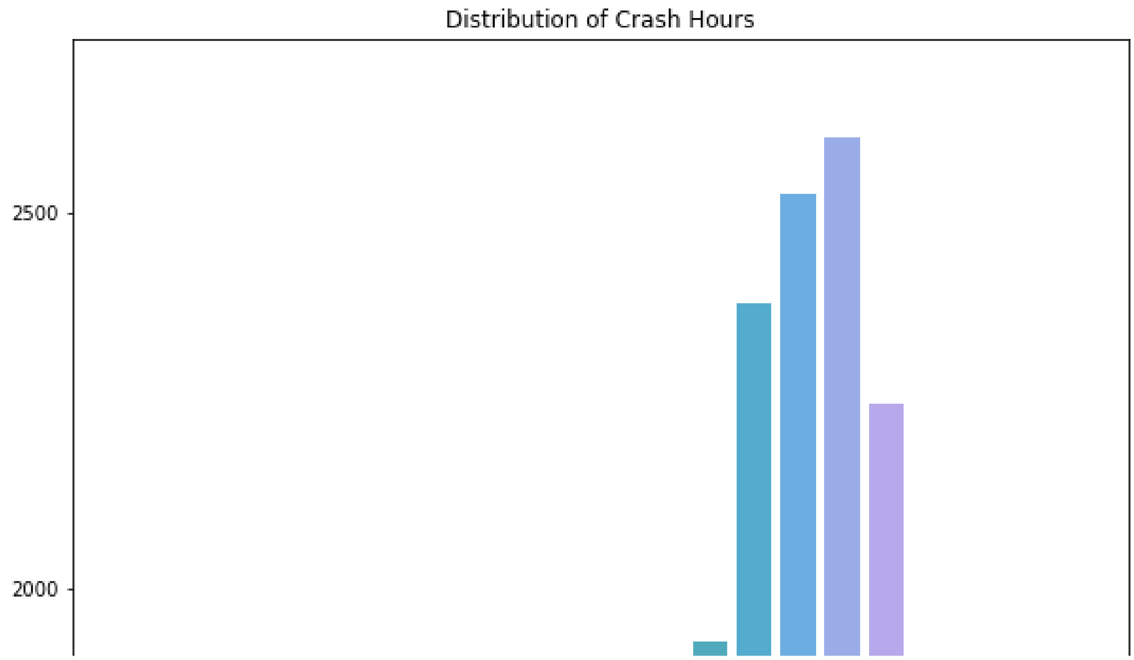




Distribution of Most Severe Injury
30000
25000
20000
15000
10000
5000
0
Count
NO INDICATION OF INJURY
NONINCAPACITATING INJURY
REPORTED, NOT EVIDENT
INCAPACITATING INJURY
FATAL
Most Severe Injury

Make countplots for the unique values of each discrete numerical column.

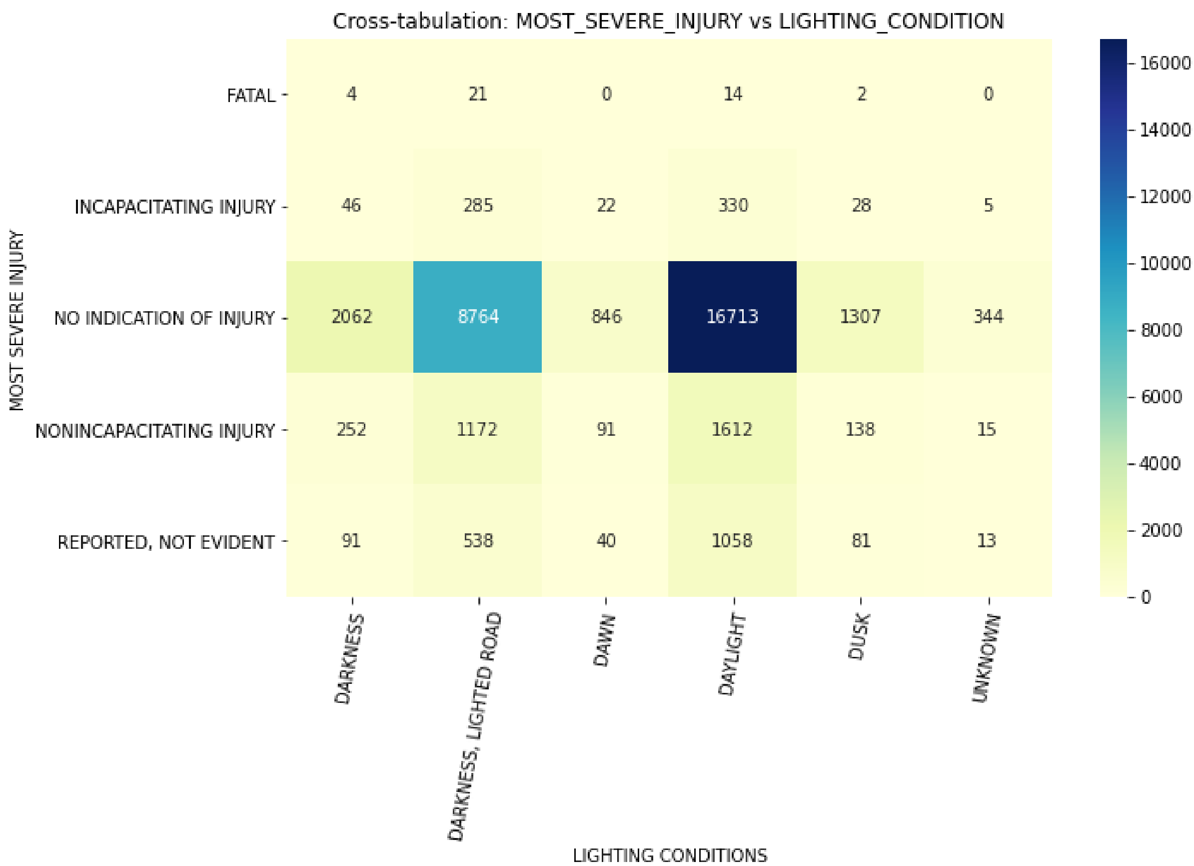
In [12]:
```
# Countplot for 'CRASH_HOUR'
plt.figure(figsize = (10, 20))
sns.countplot(data = df_crashes, x = 'CRASH_HOUR', dodge = True)
plt.title('Distribution of Crash Hours')
plt.xlabel('Crash Hours')
plt.ylabel('Number of Crashes')
plt.xticks(range(24), ['12 am', '1 am', '2 am', '3 am', '4 am', '5 am', '
                       '8 am', '9 am', '10 am', '11 am', '12 pm', '1 pm',
                       '4 pm', '5 pm', '6 pm', '7 pm', '8 pm', '9 pm', '1
           rotation = 45)  # Rotate x-axis labels if needed
plt.show()

# Countplot for 'CRASH_DAY_OF_WEEK' with adjusted spacing between bars
plt.figure(figsize = (10, 6))
sns.countplot(data = df_crashes, x = 'CRASH_DAY_OF_WEEK', dodge = True)
plt.title('Distribution of Crash Days of Weeks')
plt.xlabel('Crash Days of Weeks')
plt.ylabel('Number of Crashes')
plt.xticks(range(7), ['Sun', 'Mon', 'Tue', 'Wed', 'Thur', 'Fri', 'Sat'],
           rotation = 45)  # Rotate x-axis labels if needed
plt.show()

# Countplot for 'CRASH_MONTH' with adjusted spacing between bars
plt.figure(figsize = (10, 6))
sns.countplot(data = df_crashes, x = 'CRASH_MONTH', dodge = True)
plt.title('Distribution of Crash Months')
plt.xlabel('Crash Months')
plt.ylabel('Number of Crashes')
plt.xticks(range(12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                    'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
           rotation = 45)  # Rotate x-axis labels if needed
plt.show()
```



Make heatmaps of cross-tabulation between every possible 'MOST_SEVERE_INJURY' column pair.

Again, in our filtered dataframe, the only other categorical column outside of injury severity is lighting condition.

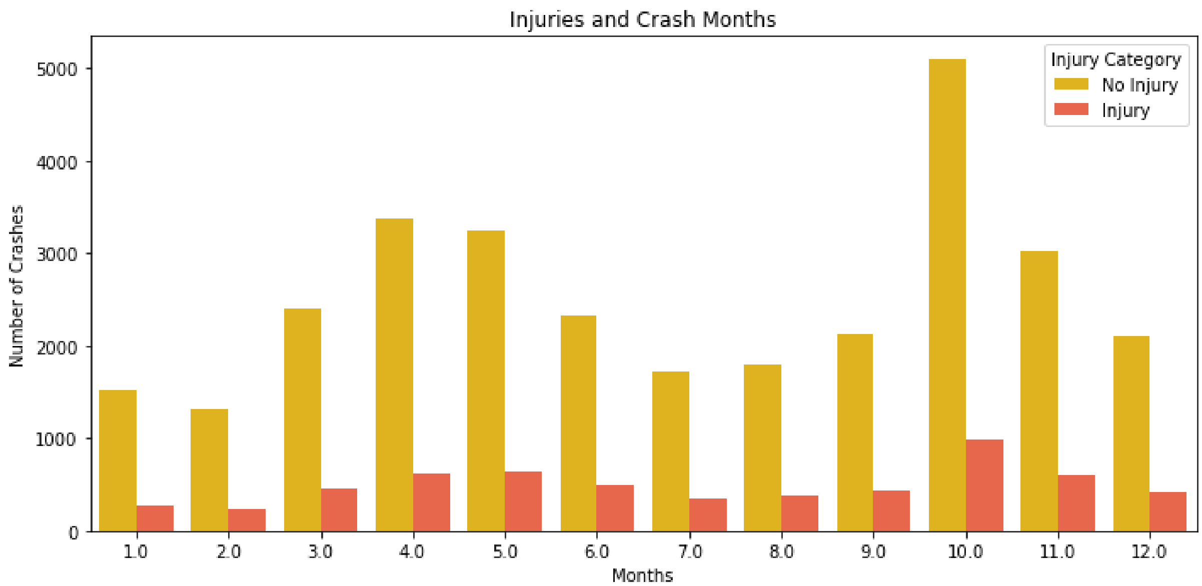
In [13]:
```
# Cross-tabulation between 'MOST_SEVERE_INJURY' and 'LIGHTING_CONDITION'
cross_tab_severe_lighting = pd.crosstab(df_crashes['MOST_SEVERE_INJURY'],
plt.figure(figsize = (10, 6))
sns.heatmap(cross_tab_severe_lighting, cmap = "YlGnBu", annot = True, fmt
plt.title("Cross-tabulation: MOST_SEVERE_INJURY vs LIGHTING_CONDITION")
plt.xlabel("LIGHTING CONDITIONS")
plt.xticks(rotation = 80)  # Rotate x-axis labels if needed
plt.ylabel("MOST SEVERE INJURY")
plt.show()
```



Below, we are interested in which month contained the most injury-positive crashes.

In [14]:
```
# groupby for EDA
grouped_crashes_binary = df_crashes.groupby(['INJURY_BINARY'])
```

In [15]:
```
# A Grouped bar graph on No Injury/Injury and Crash Months

# Define a dictionary for mapping values to labels
injury_labels = {0: 'No Injury', 1: 'Injury'}

# Define custom colors for the bars
custom_colors = ['#FFC300', '#FF5733']

# Plot the countplot
fig, ax = plt.subplots(figsize=(10, 5))
sns.countplot(data = df_crashes, x = 'CRASH_MONTH', hue = 'INJURY_BINARY'

ax.set_title("Injuries and Crash Months")
ax.set_xlabel("Months")
ax.set_ylabel("Number of Crashes")

plt.legend(title = 'Injury Category', labels = injury_labels.values(), lo
plt.tight_layout()

# Show the plot
plt.show()

# I would have guessed more crashes during the winter months
# But there seems to be no dramatic differences between months
```



BINOMIAL MODELING

Below begins our attempts at performing binomial modeling, using models such as a dummy classifier, a binomial decision tree, and a binomial logistic regression model.

TRAIN-TEST SPLIT FOR BINOMIAL MODELING.

In [16]:
```
from sklearn.model_selection import train_test_split

# Split the features and target variable
X = df_crashes.drop(['INJURY_BINARY','MOST_SEVERE_INJURY'], axis = 1)  #
y = df_crashes.INJURY_BINARY  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
```

PERFORM ONE-HOT ENCODING FOR ANY CATEGORICAL COLUMN(S) FOR BINOMIAL MODELING.

In [17]:
```
from sklearn.preprocessing import OneHotEncoder

# Specify the categorical columns to encode (assuming they are categorica
categorical_cols = ['LIGHTING_CONDITION']

# Create an instance of the OneHotEncoder
ohe = OneHotEncoder(drop = 'first')

# Transform the categorical columns in both the training and testing sets
X_train_encoded = pd.DataFrame (ohe.fit_transform(X_train[categorical_col
X_test_encoded = pd.DataFrame (ohe.transform(X_test[categorical_cols]).to

X_train_merge = X_train.drop(['LIGHTING_CONDITION'], axis=1)
X_train_merge.reset_index(drop=True, inplace=True)
X_test_merge = X_test.drop(['LIGHTING_CONDITION'], axis=1)
X_test_merge.reset_index(drop=True, inplace=True)

X_train_encoded = pd.concat([X_train_encoded, X_train_merge], axis=1)
X_test_encoded = pd.concat([X_test_encoded, X_test_merge], axis=1)
```

PERFORM SMOTE TO ADDRESS CLASS IMBALANCE (BINOMIAL).

In [18]:
```
from imblearn.over_sampling import SMOTE

# Create an instance of SMOTE
smote = SMOTE(random_state = 42)

# Apply SMOTE on the training data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_encoded
```

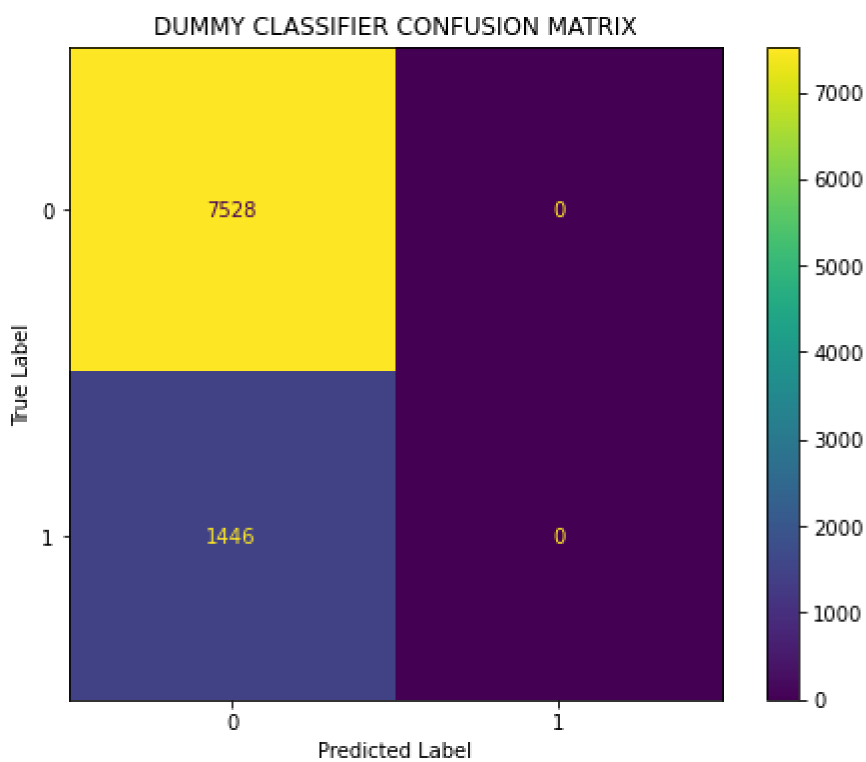
APPLY A DUMMY CLASSIFIER AS THE BASELINE MODEL.

Then produce a classification report to see the model metrics.

In [19]:
```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import classification_report

# Create a baseline model
dummy_model = DummyClassifier(strategy = 'most_frequent')

# Train the model on the resampled training data
dummy_model.fit(X_train_resampled, y_train_resampled)

# Predict on the testing data
y_pred = dummy_model.predict(X_test_encoded)

# Evaluate the model with a classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      7528
           1       0.00      0.00      0.00      1446

    accuracy                           0.84      8974
   macro avg       0.42      0.50      0.46      8974
weighted avg       0.70      0.84      0.77      8974


c:\Users\simon\anaconda3\envs\learn-env\lib\site-packages\sklearn\met
rics\_classification.py:1469: UndefinedMetricWarning: Precision and F
-score are ill-defined and being set to 0.0 in labels with no predict
ed samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\simon\anaconda3\envs\learn-env\lib\site-packages\sklearn\met
rics\_classification.py:1469: UndefinedMetricWarning: Precision and F
-score are ill-defined and being set to 0.0 in labels with no predict
ed samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [20]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix = cm)

# Plot the confusion matrix
fig, ax = plt.subplots(figsize=(8, 6))
disp.plot(ax=ax)

# Add a title and axis labels
plt.title("DUMMY CLASSIFIER CONFUSION MATRIX")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

# Show the plot

plt.show()
```



APPLY A BINOMIAL DECISION TREE MODEL.

Then produce a classification report to see the model metrics.

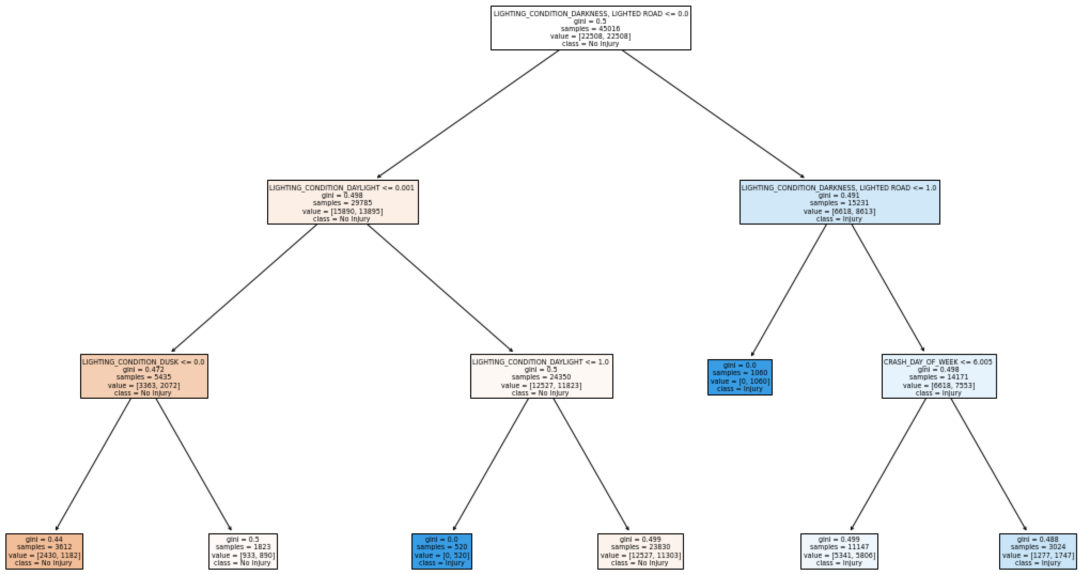
In [21]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix


# Create an instance of the DecisionTreeClassifier
bn_tree_model = DecisionTreeClassifier(max_depth = 3)

# Train the model
bn_tree_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred_tree = bn_tree_model.predict(X_test_encoded)

# Evaluate the model's performance using classification metrics
print("Classification Report:")
print(classification_report(y_test, y_pred_tree))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.71      0.78      7528
           1       0.18      0.34      0.24      1446

    accuracy                           0.65      8974
   macro avg       0.52      0.53      0.51      8974
weighted avg       0.74      0.65      0.69      8974
```

The recall for '1', meaning 'indicative of injury in car crash', is .34, meaning that the binomial decision tree model correctly predicts 33.7% of all injuries.

BINOMIAL DECISION TREE VISUALIZATION

In [22]:
```python
from sklearn import tree

# Plot the decision tree
plt.figure(figsize=(20, 12))
tree.plot_tree(bn_tree_model, feature_names= X_train_encoded.columns.toli
plt.show()
```



BINOMIAL DECISION TREE FEATURE IMPORTANCES

In sum, this decision tree illustrates which column features are the most important when it comes to prediction strength, which in this case, would be darkness (little to no lighting).

In [23]:
```python
feature_imp = {}
for fi, feature in zip(bn_tree_model.feature_importances_, bn_tree_model.
    feature_imp[feature] = fi

feature_imp
```

Out[23]:
```
{'LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD': 0.5791452323613984,
 'LIGHTING_CONDITION_DAWN': 0.0,
 'LIGHTING_CONDITION_DAYLIGHT': 0.3487380774608818,
 'LIGHTING_CONDITION_DUSK': 0.05792755862291699,
 'LIGHTING_CONDITION_UNKNOWN': 0.0,
 'CRASH_HOUR': 0.0,
 'CRASH_DAY_OF_WEEK': 0.014189131554802841,
 'CRASH_MONTH': 0.0}
```

BINOMIAL LOGISTIC REGRESSION

Then produce a classification report to see the model metrics.

In [24]:
```
from sklearn.linear_model import LogisticRegression

# Create an instance of the logistic regression model
bn_logreg_model = LogisticRegression()

# Train the logistic regression model
bn_logreg_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred_bn_logreg = bn_logreg_model.predict(X_test_encoded)

# Evaluate the logistic regression
print("Classification Report:")
print(classification_report(y_test, y_pred_bn_logreg))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.71      0.77      7528
           1       0.18      0.34      0.24      1446

    accuracy                           0.65      8974
   macro avg       0.51      0.52      0.50      8974
weighted avg       0.74      0.65      0.68      8974
```

The recall for '1', meaning 'indicative of injury in car crash', is .34, meaning that the binomial logistic regression model correctly predicts 34.1% of all injuries.

COEFFICIENTS FROM BINOMIAL LOGISTIC REGRESSION MODEL

In [25]:
```
# Get the coefficients and intercept from the logistic regression model
coefficients = bn_logreg_model.coef_
intercept = bn_logreg_model.intercept_

# Create a DataFrame to display the coefficients
coefficients_df = pd.DataFrame(coefficients, columns=X_train_encoded.colu
coefficients_df['Intercept'] = intercept

# Print the coefficients
print("Coefficients:")
print(coefficients_df)
```

```
Coefficients:
   LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD  LIGHTING_CONDITION_DAWN
\
0                                  0.413239                 0.002866

   LIGHTING_CONDITION_DAYLIGHT  LIGHTING_CONDITION_DUSK  \
0                     0.131102                 0.184448

   LIGHTING_CONDITION_UNKNOWN  CRASH_HOUR  CRASH_DAY_OF_WEEK  CRASH_MON
TH  \
0                   -0.913818   -0.004657           0.003763     0.0083
05

   Intercept
0  -0.219059
```

ODDS RFOM THE COEFFICIENTS OF THE BINOMIAL LOGISTIC REGRESSION MODEL

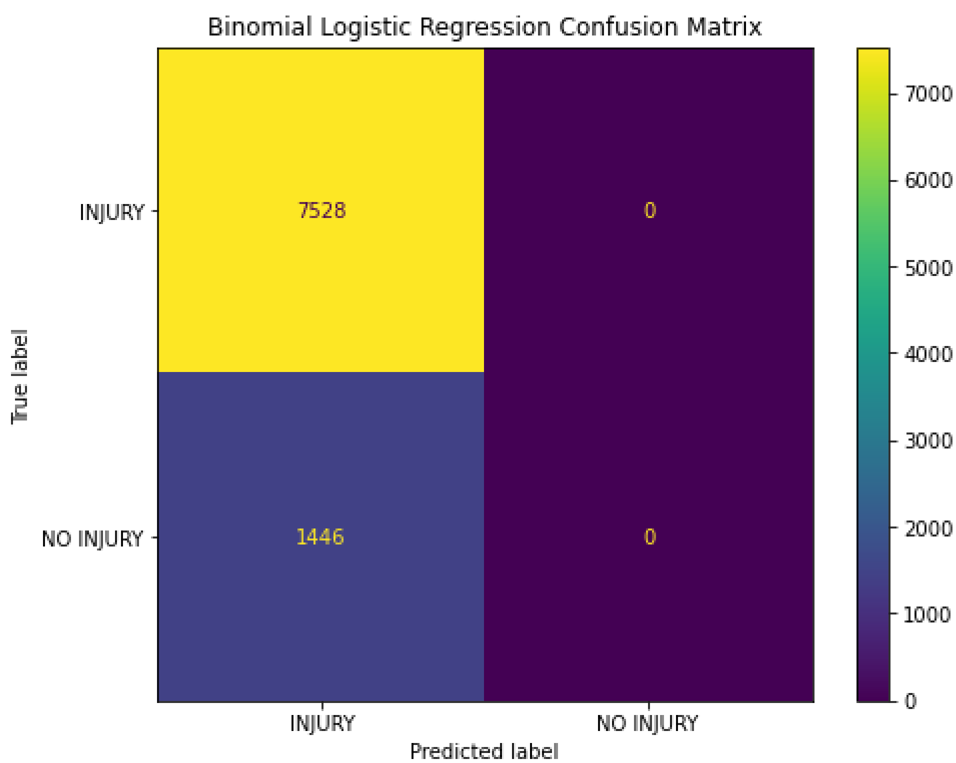
We've concluded, based on the odds received from the coefficients of the binomial logistic regression model, that dark lighting conditions with a lit road results in a 1.5x more likelihood of injury in the event of a rainy-day car crash.

In [26]:
```
odds = np.exp(coefficients)

# Create a DataFrame to display the odds
odds_df = pd.DataFrame(odds, columns=X_train_encoded.columns)

# Print the odds
print("Odds:")
print(odds_df)
```

```
Odds:
   LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD  LIGHTING_CONDITION_DAWN
\
0                                   1.511706                  1.00287

   LIGHTING_CONDITION_DAYLIGHT  LIGHTING_CONDITION_DUSK  \
0                     1.140084                 1.202554

   LIGHTING_CONDITION_UNKNOWN  CRASH_HOUR  CRASH_DAY_OF_WEEK  CRASH_MON
TH
0                     0.40099    0.995354            1.00377      1.008
34
```

In [27]:
```
import numpy as np
from sklearn.metrics import ConfusionMatrixDisplay

# Get the predictions and the actual values
y_pred_bn_logreg = bn_logreg_model.predict(X_test_encoded)
y_true = y_test

# Create a confusion matrix display
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['INJUR

# Plot the confusion matrix
fig, ax = plt.subplots(figsize=(8, 6))
disp.plot(ax=ax)

# Set the title
ax.set_title('Binomial Logistic Regression Confusion Matrix')
```

Out[27]:
```
Text(0.5, 1.0, 'Binomial Logistic Regression Confusion Matrix')
```



MULTINOMIAL MODELING

Below begins our attempts at performing multinomial modeling, using models such as a dummy classifier, a multinomial decision tree, and a multinomial logistic regression model.

TRAIN-TEST SPLIT FOR MULTINOMIAL MODELING.

In [28]:
```
# Perform a new train-test split with a multiclass target variable
y2 = df_crashes.MOST_SEVERE_INJURY # <------

X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X, y2, random
```

PERFORM ONE-HOT ENCODING FOR ANY CATEGORICAL COLUMN(S) FOR MULTINOMIAL MODELING.

In [29]:
```
from sklearn.preprocessing import OneHotEncoder

# Specify the categorical columns for one-hot encoding
categorical_cols = ['LIGHTING_CONDITION']

# Create an instance of the OneHotEncoder
ohe = OneHotEncoder(drop='first')

X_train_encoded_2 = pd.DataFrame (ohe.fit_transform(X_train_2[categorical
X_test_encoded_2 = pd.DataFrame (ohe.transform(X_test_2[categorical_cols]

X_train_merge_2 = X_train_2.drop(['LIGHTING_CONDITION'], axis=1)
X_train_merge_2.reset_index(drop=True, inplace=True)
X_test_merge_2 = X_test_2.drop(['LIGHTING_CONDITION'], axis=1)
X_test_merge_2.reset_index(drop=True, inplace=True)

X_train_encoded_2 = pd.concat([X_train_encoded_2, X_train_merge_2], axis=
X_test_encoded_2 = pd.concat([X_test_encoded_2, X_test_merge_2], axis=1)
```

PERFORM SMOTE TO ADDRESS CLASS IMBALANCE (MULTINOMIAL).

In [30]:
```
from imblearn.over_sampling import SMOTE

# Create an instance of the SMOTE algorithm
smote_2 = SMOTE(random_state = 42)

# Apply SMOTE to the encoded training data
X_train_resampled_2, y_train_resampled_2 = smote_2.fit_resample(X_train_e
```

APPLY A DUMMY CLASSIFIER AS THE BASELINE MODEL AGAIN.

Then produce a classification report to see the model metrics.

In [31]:
```
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train_resampled_2, y_train_resampled_2)

y_pred_dummy_clf = dummy_clf.predict(X_test_encoded_2)

# Evaluate the model with a classification report
report = classification_report(y_test_2, y_pred_dummy_clf)
print(report)
```

```
                          precision    recall  f1-score   support

                   FATAL       0.00      1.00      0.00         8
   INCAPACITATING INJURY       0.00      0.00      0.00       178
  NO INDICATION OF INJURY      0.00      0.00      0.00      7528
NONINCAPACITATING INJURY       0.00      0.00      0.00       790
   REPORTED, NOT EVIDENT       0.00      0.00      0.00       470

                accuracy                           0.00      8974
               macro avg       0.00      0.20      0.00      8974
            weighted avg       0.00      0.00      0.00      8974


c:\Users\simon\anaconda3\envs\learn-env\lib\site-packages\sklearn\met
rics\_classification.py:1469: UndefinedMetricWarning: Precision and F
-score are ill-defined and being set to 0.0 in labels with no predict
ed samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\simon\anaconda3\envs\learn-env\lib\site-packages\sklearn\met
rics\_classification.py:1469: UndefinedMetricWarning: Precision and F
```

APPLY A MULTINOMIAL DECISION TREE

Then produce a classification report to see the model metrics.

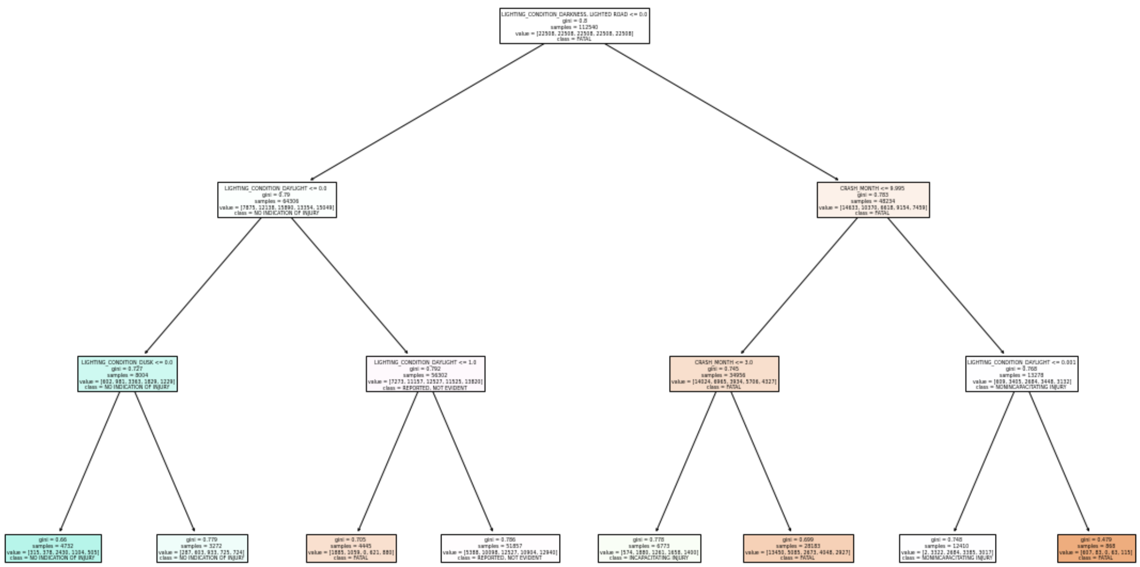
In [32]:
```
# Create the multinomial decision tree classifier
mn_tree_model = DecisionTreeClassifier(criterion='gini', splitter='best',

# Fit the model to the SMOTE training data
mn_tree_model.fit(X_train_resampled_2, y_train_resampled_2)

# Predict on the test data
y_pred_2 = mn_tree_model.predict(X_test_encoded_2)

# Evaluate the model's performance
classification_report_2 = classification_report(y_test_2, y_pred_2)

# Print the classification report
print("Classification Report:")
print(classification_report_2)
```

```
Classification Report:
                            precision    recall  f1-score   support

                     FATAL       0.00      0.12      0.00         8
   INCAPACITATING INJURY       0.02      0.06      0.03       178
    NO INDICATION OF INJURY       0.85      0.16      0.27      7528
NONINCAPACITATING INJURY       0.09      0.13      0.11       790
   REPORTED, NOT EVIDENT       0.06      0.59      0.10       470

                  accuracy                           0.18      8974
                 macro avg       0.20      0.21      0.10      8974
              weighted avg       0.72      0.18      0.24      8974
```

The weighted avg for recall is 0.18, meaning that the multinomial decision tree model correctly predicts 17.7% of all injuries.

MULTINOMIAL DECISION TREE VISUALIZATION

In [33]:
```
from sklearn import tree

# Get the unique values from the 'MOST_SEVERE_INJURY' column
target_class_names = np.unique(df_crashes['MOST_SEVERE_INJURY']).tolist()

# Plot the decision tree
plt.figure(figsize=(20, 12))
tree.plot_tree(mn_tree_model, feature_names= X_train_encoded_2.columns.to
plt.show()
```



MULTINOMIAL DECISION TREE FEATURE IMPORTANCES

Again, this decision tree illustrates which column features are the most important when it comes to prediction strength, which in this case, would be darkness (little to no lighting).

In [34]:
```
feature_imp_2 = {}
for fi, feature in zip(mn_tree_model.feature_importances_, mn_tree_model.
    # feature_imp[feature] = fi
    print(fi, feature)
```

```
0.24907245742022144 LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD
0.0 LIGHTING_CONDITION_DAWN
0.2767544152758435 LIGHTING_CONDITION_DAYLIGHT
0.02574958138793218 LIGHTING_CONDITION_DUSK
0.0 LIGHTING_CONDITION_UNKNOWN
0.0 CRASH_HOUR
0.0 CRASH_DAY_OF_WEEK
0.44842354591600286 CRASH_MONTH
```

MULTINOMIAL LOGISTIC REGRESSION MODEL

Then produce a classification report to see the model metrics.

In [35]:
```
# Create an instance of the multinomial logistic regression model
mn_logreg_model = LogisticRegression(multi_class = 'multinomial', solver

# Fit the model on the training data
mn_logreg_model.fit(X_train_resampled_2, y_train_resampled_2)

# Predict on the test data
y_pred_mn_logreg = mn_logreg_model.predict(X_test_encoded_2)

# Evaluate the accuracy of the model
print("Classification Report:")
print(classification_report(y_test_2, y_pred_mn_logreg))
```

```
Classification Report:
                          precision    recall  f1-score   support

                   FATAL       0.00      0.50      0.00         8
    INCAPACITATING INJURY       0.02      0.08      0.03       178
   NO INDICATION OF INJURY       0.84      0.08      0.14      7528
NONINCAPACITATING INJURY       0.10      0.06      0.08       790
    REPORTED, NOT EVIDENT       0.06      0.58      0.10       470

                accuracy                           0.10      8974
               macro avg       0.21      0.26      0.07      8974
            weighted avg       0.72      0.10      0.13      8974


c:\Users\simon\anaconda3\envs\learn-env\lib\site-packages\sklearn\linea
r_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression (https://scikit-learn.org/stable/modules/linear_model.html#l
ogistic-regression)
  n_iter_i = _check_optimize_result(
```

The weighted average for recall is .1, meaning that the multinomial logistic regression model correctly predicts 10.2% of all injuries.

COEFFICIENTS FROM THE MULTINOMIAL

In [36]:
```
coefficients = mn_logreg_model.coef_

# Print the coefficients
for i, class_name in enumerate(mn_logreg_model.classes_):
    print(f"Coefficients for class '{class_name}':")
    for j, feature_name in enumerate(X_train_encoded_2.columns):
        print(f"{feature_name}: {coefficients[i, j]}")
    print()
```

```
Coefficients for class 'FATAL':
LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD: -0.08342714922539403
LIGHTING_CONDITION_DAWN: -1.0784672713471242
LIGHTING_CONDITION_DAYLIGHT: -0.880712986434862
LIGHTING_CONDITION_DUSK: 0.20553951612889756
LIGHTING_CONDITION_UNKNOWN: -0.3423905415827352
CRASH_HOUR: -0.025491010196693628
CRASH_DAY_OF_WEEK: 0.03927227123349062
CRASH_MONTH: -0.07776783037073115

Coefficients for class 'INCAPACITATING INJURY':
LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD: 0.34396708322589753
LIGHTING_CONDITION_DAWN: -0.02061435156303981
LIGHTING_CONDITION_DAYLIGHT: 0.22440758345031106
LIGHTING_CONDITION_DUSK: -0.0060958979951853225
LIGHTING_CONDITION_UNKNOWN: 0.13729872698439577
CRASH_HOUR: -0.0053338218595217635
CRASH_DAY_OF_WEEK: 0.010960496738882759
CRASH_MONTH: 0.022585974197239976
```

ODDS FROM THE COEFFICIENTS OF THE MULTINOMIAL LOGISTIC REGRESSION MODEL

In [37]:
```
odds = np.exp(coefficients)

# Print the odds
for i, class_name in enumerate(mn_logreg_model.classes_):
    print(f"Odds for class '{class_name}':")
    for j, feature_name in enumerate(X_train_encoded_2.columns):
        print(f"{feature_name}: {odds[i, j]}")
    print()
```

```
Odds for class 'FATAL':
LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD: 0.9199581038905521
LIGHTING_CONDITION_DAWN: 0.34011643253996665
LIGHTING_CONDITION_DAYLIGHT: 0.41448728249446354
LIGHTING_CONDITION_DUSK: 1.228187513219042
LIGHTING_CONDITION_UNKNOWN: 0.7100708383620885
CRASH_HOUR: 0.9748311424665776
CRASH_DAY_OF_WEEK: 1.040053621785724
CRASH_MONTH: 0.9251792001225231

Odds for class 'INCAPACITATING INJURY':
LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD: 1.4105322045735622
LIGHTING_CONDITION_DAWN: 0.9795966716589808
LIGHTING_CONDITION_DAYLIGHT: 1.251581039201794
LIGHTING_CONDITION_DUSK: 0.9939226442945642
LIGHTING_CONDITION_UNKNOWN: 1.1471707882101385
CRASH_HOUR: 0.994680377711082
CRASH_DAY_OF_WEEK: 1.011020783037869
CRASH_MONTH: 1.0228429684874687
```

We've concluded, based on the odds received from the coefficients of the multinomial logistic regression model, that dark lighting conditions with a lit road results in a 1.4x more likelihood of an incapacitating injury in the event of a rainy-day car crash.

RECOMMENDATIONS

Install safety measures to poorly lit areas.

Proactive weather monitoring focusing on these areas.

Optimize first-response personnel being dispatched.

Type *Markdown* and LaTeX: α^2