CE C265 & PBHLTH 285
Spring 2025
Assignment 3: Crash Severity Modeling

Submitted by: Chun-hin (Anson) Sit, Jesus Hinojosa
Mar 11, 2025

In this assignment you will develop a crash severity model using an ordered logit regression approach implemented in Python. Your objective is to identify and quantify the factors that influence the severity of crashes by analyzing a variety of explanatory variables such as road characteristics, driver behavior, environmental conditions, and vehicle types. You will use data extracted from TIMS (https://tims.berkeley.edu). Details on the data can be found at (https://tims.berkeley.edu/help/SWITRS.php).

Option 2
As an alternative to building your own model from scratch, a simple model code will be made available to you (on bcourses). With this code, you can calibrate the model across various counties or cities and then compare and discuss the differences in the estimated coefficients across these geographic regions. This approach will allow you to explore how local factors might influence crash severity and to interpret the variations in the results. Regardless of the option you choose, you are expected to provide a thorough interpretation of the model outcomes, similar to option 1.

Introduction

We are doing Option 2.

We have selected 3 counties to look at and compare the estimated coefficients and odd ratios across them. The 3 chosen counties are:

1. Alameda (Where Berkeley is located in)
2. San Francisco (Where a comprehensive public transit systems is developed)
3. Santa Clara (One of the wealthiest counties in the Bay Area)

For all counties, the range of data is from Jan 1 2022 to Dec 31 2023.

Result Interpretation

Summarizing the independent variables, their coefficients and level of statistical significance into the table below.

	Alameda		San Francisco		Santa Clara	
	Coef	P>|z|	Coef	P>|z|	Coef	P>|z|
Weekend	0.1156	0.002	0.1176	0.035	0.2531	0
Time of Day - Evening	0.2907	0	-0.0462	0.46	0.3037	0
Time of Day - Morning	-0.0997	0.02	-0.1186	0.073	-0.0368	0.415
Time of Day - Night	0.5999	0	0.2751	0.001	0.7751	0
Threshold Parameter 1/2	0.3999	0	0.4881	0	0.4439	0
Threshold Parameter 2/3	0.7549	0	0.625	0	0.767	0
Threshold Parameter 3/4	0.6244	0	0.7387	0	0.5195	0

Note: the coefficients are compared against a baseline scenario, which is the weekday afternoon.

Coefficient

<u>Weekend Effect</u>

- Consistent positive effect across all counties
- Magnitude: Strongest in Santa Clara (0.2531), similar between Alameda and San Francisco (~0.12)
- All statistical significant

Weekend crashes are consistently more severe than weekday crashes, with the effect being about twice as strong in Santa Clara compared to the other counties.

<u>Evening Effect</u>

Notable variation across counties:

- Positive and significant in Alameda (0.2907, $p<0.001$)
- Positive and significant in Santa Clara (0.3037, $p<0.001$)
- Negative but not significant in SF (-0.0462, $p=0.460$)
- Magnitude: Strongest in Santa Clara (0.2531), similar between Alameda and San Francisco (~0.12)

Evening crashes are significantly more severe in Alameda and Santa Clara but show no difference from afternoon crashes in San Francisco.

Morning Effect

- Consistent negative direction across all counties:
- Magnitude: Strongest in SF (-0.1186), less pronounced in Alameda (-0.0997), weakest in Santa Clara (-0.0368)
- Significance: Significant in Alameda ($p=0.020$), marginally significant in San Francisco ($p=0.073$), not significant in Santa Clara ($p=0.415$)

Morning crashes tend to be less severe than afternoon crashes, with the effect being more reliable in Alameda and somewhat in San Francisco.

Night Effect

- Strongest positive effect among all variables across all counties
- Magnitude: Highest in Santa Clara (0.7751), followed by Alameda (0.5999), lowest in SF (0.2751)
- Significance: Highly significant in all counties ($p<0.001$ for Alameda and Santa Clara, $p=0.001$ for SF)

Night crashes are consistently the most severe across all three counties, with the effect being nearly three times stronger in Santa Clara than in San Francisco.

Overall Pattern

- Night driving is associated with higher crash severity across all counties, with the effect being strongest in Santa Clara.
- San Francisco shows less time-of-day variation in crash severity.
- Santa Clara shows the strongest effects for both weekend and night variables.
- Evening effect is notably absent in San Francisco but strong in the other counties.
- Morning and Evening effects show more geographic variation in significance
- Time of day appears to have a stronger influence than day of week in all counties, with the night period being the most dangerous time for crash severity.

Threshold Parameters

These thresholds represent cut points on an underlying latent continuous variable (y*) that determines the observed crash severity:

- 1/2: Threshold between severity levels 1 and 2
- 2/3: Threshold between severity levels 2 and 3
- 3/4: Threshold between severity levels 3 and 4

Contrasting the thresholds across the 3 sets of data:

Threshold	Alameda	San Francisco	Santa Clara
1/2	0.4	0.488	0.444
2/3	0.755	0.625	0.767
3/4	0.624	0.739	0.52

The thresholds define the "difficulty" of reaching each severity level:

- Wider gaps between thresholds mean it's harder to move from one severity level to the next
- The specific values determine the baseline probability distribution across severity levels

As observed in the table, the thresholds don't monotonically increase for Alameda and Santa Clara (3/4 is smaller than 2/3). The larger gap between thresholds 1/2 and 2/3 in Alameda and Santa Clara suggests it takes more factors to push a crash from severity 2 to severity 3 than from 1 to 2.

Odd Ratios

The Odd ratios help us understand the different factors like the day of the week and time of day have on the severity of collisions. The ratios found for Alameda, San Francisco, and Santa Clara when the ratio is greater than 1 tells us that the odds of severity increased, and when it is lower than 1 the severity decreased.

As we can see from the Data for all counties looking at the variable “DAY_OF *WEEK*_WEEKEND” Collisions are 1.123-1.125 (depending on the county) more likely to happen on the weekends compared to weekdays for each county. This could be due to people taking more trips or engage in risky behaviors on the

weekend like drinking, speeding, or distracted driving due to social events that occur on the weekends.

For “TIME_OF_DAY_E” in Alameda the time in the evening (18:00 - 23:59), collisions are 1.337320 times higher in the afternoon. Compared to San Francisco and Santa Clara counties where their ratio is 0.955 which means it decreases. This could be due to a more trip occurring in Alameda do most jobs are in San Francisco and Santa Clara, and traffic is moving away from these cities and moving toward Alameda.

For “TIME_OF_DAY_M” all counties have lower ratios than 1 of collins occurring in our of the morning (06:00-11:59) which means they are decreasing. While the morning commute does increase the traffic flow in the morning leading to the belief that more collisions can occur this could indicate that traffic jams cause slower traffic lowering the severity of collisions. Additionally when traffic frees because people are working those on the roads are long-haul drivers or drivers that have strict driving restrictions.

For “TIME_OF_DAY_N” all counties have higher ratios of severe collisions at night (00:00- 5:59) this could be because of lower visibility at night, and a higher risk of ricker behavior due to the decrease in traffic. Alameda is the highest at 1.8 factors that could lead to this possible poor road design which is highlighted when it is darker and people engage in risky behaviors.

Now look at the threshold parameter which shows the odds ratio for the transition from the severity going from level 1 to 2, 2 to 3, and 3 to 4. The higher the ratio the more likely it the to continue to the next level. Level 1 is the least serve, Level 2 is moderately serve, 3 is serve and 4 is the most serve collisions. For levels 1 to 2, the range for all counties is 1.5 to 1.6 showing the high jump for all. For levels 2 to 3 the jump is much higher for all counties meaning more serve collisions are occurring. Levels 3 to 4 have the highest jump for them all meaning that more severe collisions are occurring.

Discussion on the Methodology

Model Fit Statistics

Statistic	Alameda	San Francisco	Santa Clara
Log-Likelihood	-13,058	-5,944	-12,038
AIC/n	1.87	1.82	1.88
BIC/n	1.87	1.83	1.88
Sample Size (n)	14,005	6,545	12,846
Function Value*	0.932	0.908	0.937

*Note: *Function Value = (−Log-Likelihood)/n, smaller values indicate better fit*

The differences of the function values are small, suggesting similar predictive power across all three models.

Regarding the AIC/n and BIC/n values, they are also roughly in the same range, with San Francisco having the smallest values, confirming that San Francisco has slightly better model fit.

Threshold Parameters

Thresholds that differ across countries suggest that the same underlying crash factors might be categorized differently in different jurisdictions.

Prediction on Crash Severity

The current set of independent variables failed to discriminate between different severity levels, as shown in the confusion matrices. They predicted no high severity level crashes in the dataset of San Francisco and Santa Clara. While the chosen set of variables are statistically significant, the models fail to translate these relationships into meaningful severity classifications.

Possible Improvements

- Incorporate other variables, as the current model relies solely on temporal variables (time of day and weekday/weekend), and crash severity is influenced by many factors other than that.

Appendix (Jupyter Notebook)

Analysis for Alameda

In [15]:
```python
import pandas as pd
import numpy as np
from statsmodels.miscmodels.ordinal_model import OrderedModel

def remap_severity(severity_series):
    """
    Remap collision severity values to a reversed scale and convert to an ordere

    The function converts severity scores from the original scale to a reversed
        1 → 4
        2 → 3
        3 → 2
        4 → 1

    Parameters:
    severity_series (pd.Series): Series containing collision severity values as

    Returns:
    pd.Categorical: An ordered categorical series with severity levels [1, 2, 3,
                    where 1 represents the least severe and 4 the most severe.
    """
    # Convert severity values to numeric, coercing errors to NaN
    severity_numeric = pd.to_numeric(severity_series, errors="coerce")
    # Define the mapping from original to reversed scale
    severity_mapping = {1: 4, 2: 3, 3: 2, 4: 1}
    remapped = severity_numeric.map(severity_mapping)
    # Define the order for the categorical variable
    severity_order = [1, 2, 3, 4]
    return pd.Categorical(remapped, categories=severity_order, ordered=True)

def categorize_time_of_day(time_str):
    """
    Categorize a time (given as a string) into a time-of-day group.

    The function interprets the input as a time in HHMM format and assigns:
        "M" for Morning (06:00 - 11:59),
        "A" for Afternoon (12:00 - 17:59),
        "E" for Evening (18:00 - 23:59),
        "N" for Night (00:00 - 05:59).

    Parameters:
    time_str (str): A string representing the time in HHMM format (e.g., "0600"

    Returns:
    str: A single character representing the time of day ("M", "A", "E", or "N")
         If the input is invalid, returns np.nan.
    """
    try:
        time_int = int(time_str)
        if 600 <= time_int < 1200:
            return "M"  # Morning
        elif 1200 <= time_int < 1800:
            return "A"  # Afternoon
        elif 1800 <= time_int < 2400:
```

```
            return "E"  # Evening
        elif 0 <= time_int < 600:
            return "N"  # Night
    except (ValueError, TypeError):
        return np.nan

# Load dataset
df = pd.read_csv("Crashes_Alameda_2223.csv", dtype=str)

# Keep only relevant columns
columns_to_keep = ["COLLISION_SEVERITY", "DAY_OF_WEEK", "COLLISION_TIME"]
df = df[columns_to_keep]

# Apply severity remapping using the dedicated function
df["COLLISION_SEVERITY"] = remap_severity(df["COLLISION_SEVERITY"])

# Create TIME_OF_DAY by categorizing COLLISION_TIME and convert to a categorical
df["TIME_OF_DAY"] = df["COLLISION_TIME"].apply(categorize_time_of_day).astype("c
df.drop(columns=["COLLISION_TIME"], inplace=True)

# Collapse DAY_OF_WEEK into "Weekday" (Mon-Fri) vs. "Weekend" (Sat-Sun)
df["DAY_OF_WEEK"] = df["DAY_OF_WEEK"].apply(lambda x: "Weekday" if x in ["1", "2
```

In [16]:

```
# Apply dummy encoding, keeping "Weekday" as the reference category
df = pd.get_dummies(df, columns=["DAY_OF_WEEK", "TIME_OF_DAY"], drop_first=True)

# Define independent variables (all columns except the target)
independent_vars = df.columns.difference(["COLLISION_SEVERITY"])

# Specify and calibrate the Ordered Logit Model
model = OrderedModel(
    df["COLLISION_SEVERITY"],
    df[independent_vars],
    distr="logit"
)

result = model.fit(method="bfgs")
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.932356
         Iterations: 25
         Function evaluations: 27
         Gradient evaluations: 27
                           OrderedModel Results
==============================================================================
Dep. Variable:     COLLISION_SEVERITY   Log-Likelihood:                -13058.
Model:                   OrderedModel   AIC:                         2.613e+04
Method:            Maximum Likelihood   BIC:                         2.618e+04
Date:                Sun, 09 Mar 2025
Time:                        20:22:35
No. Observations:               14005
Df Residuals:                   13998
Df Model:                           4
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
DAY_OF_WEEK_Weekend     0.1156      0.038      3.042      0.002       0.041       0.190
TIME_OF_DAY_E           0.2907      0.043      6.796      0.000       0.207       0.375
TIME_OF_DAY_M          -0.0997      0.043     -2.317      0.020      -0.184      -0.015
TIME_OF_DAY_N           0.5999      0.057     10.571      0.000       0.489       0.711
1/2                     0.3999      0.029     13.926      0.000       0.344       0.456
2/3                     0.7549      0.014     54.880      0.000       0.728       0.782
3/4                     0.6244      0.035     17.860      0.000       0.556       0.693
==============================================================================
```

In [17]:

```python
# Compute and display Odds Ratios
odds_ratios = np.exp(result.params)
print("\nOdds Ratios:\n", odds_ratios)

# Predict probabilities for each severity level
predicted_probs = result.predict()

# Convert predictions to a DataFrame
predicted_probs_df = pd.DataFrame(predicted_probs)

# Determine the most likely severity level for each observation
df["Predicted_Severity"] = predicted_probs_df.idxmax(axis=1)

# Display performance metrics with a confusion matrix (objective results only)
confusion_matrix = pd.crosstab(df["COLLISION_SEVERITY"], df["Predicted_Severity"
                               rownames=['Actual'], colnames=['Predicted'])
print("\nConfusion Matrix:\n", confusion_matrix)
```

```
Odds Ratios:
 DAY_OF_WEEK_Weekend    1.122559
TIME_OF_DAY_E          1.337320
TIME_OF_DAY_M          0.905084
TIME_OF_DAY_N          1.821933
1/2                    1.491617
2/3                    2.127494
3/4                    1.867154
dtype: float64

Confusion Matrix:
 Predicted     0    1
Actual
1          7609  299
2          4651  245
3           907   92
4           176   26
```

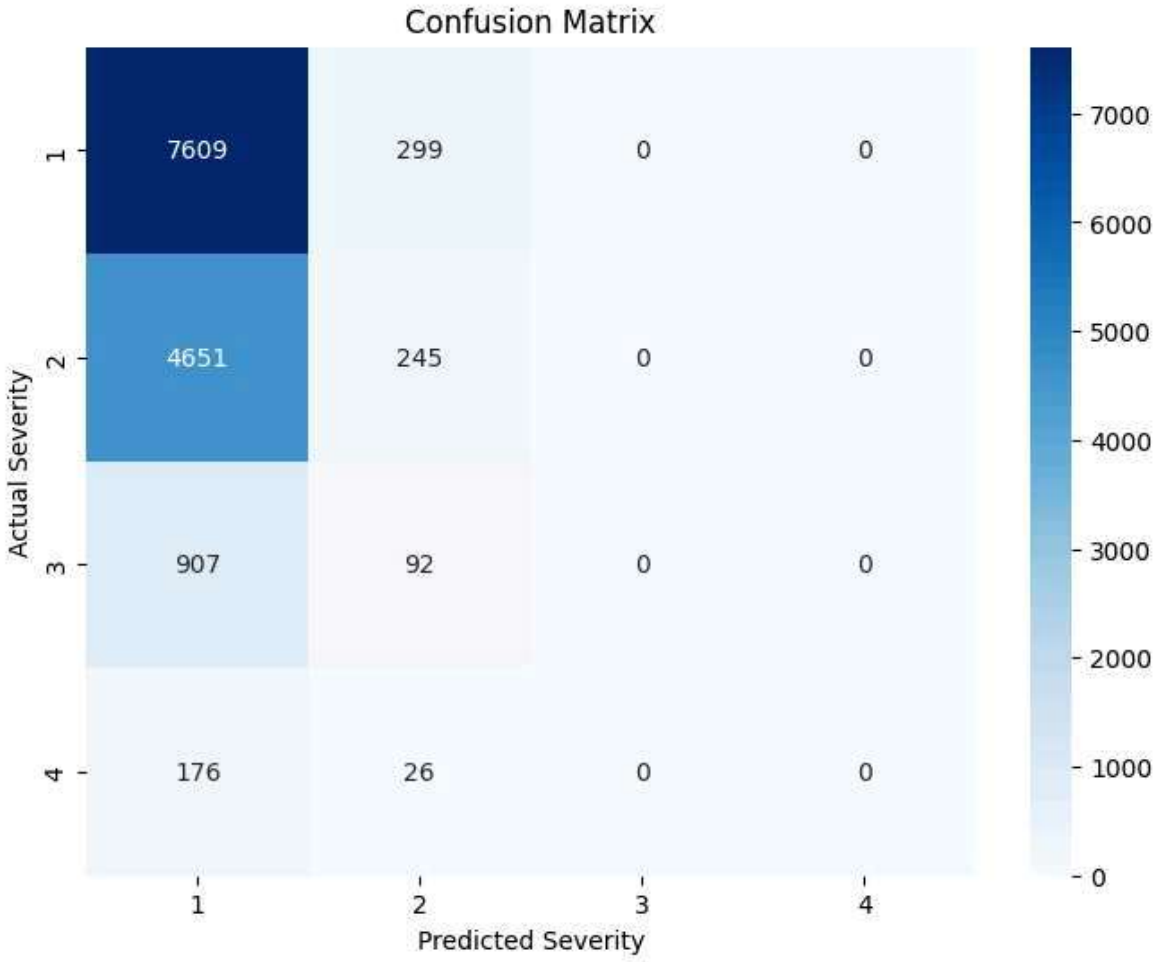
In [18]:
```
# Get the predicted class (severity level)
predicted = result.predict().argmax(axis=1) + 1  # +1 because argmax is zero-ina
actual = df["COLLISION_SEVERITY"]

# Create confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(actual, predicted)
print(conf_matrix)

# Visualize it
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=[1, 2, 3, 4],
            yticklabels=[1, 2, 3, 4])
plt.xlabel('Predicted Severity')
plt.ylabel('Actual Severity')
plt.title('Confusion Matrix')
plt.show()
```

```
[[7609  299    0    0]
 [4651  245    0    0]
 [ 907   92    0    0]
 [ 176   26    0    0]]
```



Analysis for San Francisco

In [19]:
```
import pandas as pd
import numpy as np
from statsmodels.miscmodels.ordinal_model import OrderedModel

def remap_severity(severity_series):
    """
    Remap collision severity values to a reversed scale and convert to an ordere

    The function converts severity scores from the original scale to a reversed
        1 → 4
        2 → 3
        3 → 2
        4 → 1

    Parameters:
    severity_series (pd.Series): Series containing collision severity values as

    Returns:
    pd.Categorical: An ordered categorical series with severity levels [1, 2, 3,
                    where 1 represents the least severe and 4 the most severe.
    """
    # Convert severity values to numeric, coercing errors to NaN
    severity_numeric = pd.to_numeric(severity_series, errors="coerce")
    # Define the mapping from original to reversed scale
    severity_mapping = {1: 4, 2: 3, 3: 2, 4: 1}
    remapped = severity_numeric.map(severity_mapping)
```

```
    # Define the order for the categorical variable
    severity_order = [1, 2, 3, 4]
    return pd.Categorical(remapped, categories=severity_order, ordered=True)

def categorize_time_of_day(time_str):
    """
    Categorize a time (given as a string) into a time-of-day group.

    The function interprets the input as a time in HHMM format and assigns:
        "M" for Morning (06:00 - 11:59),
        "A" for Afternoon (12:00 - 17:59),
        "E" for Evening (18:00 - 23:59),
        "N" for Night (00:00 - 05:59).

    Parameters:
    time_str (str): A string representing the time in HHMM format (e.g., "0600"

    Returns:
    str: A single character representing the time of day ("M", "A", "E", or "N")
         If the input is invalid, returns np.nan.
    """
    try:
        time_int = int(time_str)
        if 600 <= time_int < 1200:
            return "M"  # Morning
        elif 1200 <= time_int < 1800:
            return "A"  # Afternoon
        elif 1800 <= time_int < 2400:
            return "E"  # Evening
        elif 0 <= time_int < 600:
            return "N"  # Night
    except (ValueError, TypeError):
        return np.nan

# Load dataset
file_path = "Crashes.csv"  # Update with the actual file path
df = pd.read_csv("Crashes_SF_2223.csv", dtype=str)

# Keep only relevant columns
columns_to_keep = ["COLLISION_SEVERITY", "DAY_OF_WEEK", "COLLISION_TIME"]
df = df[columns_to_keep]

# Apply severity remapping using the dedicated function
df["COLLISION_SEVERITY"] = remap_severity(df["COLLISION_SEVERITY"])

# Create TIME_OF_DAY by categorizing COLLISION_TIME and convert to a categorical
df["TIME_OF_DAY"] = df["COLLISION_TIME"].apply(categorize_time_of_day).astype("c
df.drop(columns=["COLLISION_TIME"], inplace=True)

# Collapse DAY_OF_WEEK into "Weekday" (Mon-Fri) vs. "Weekend" (Sat-Sun)
df["DAY_OF_WEEK"] = df["DAY_OF_WEEK"].apply(lambda x: "Weekday" if x in ["1", "2
```

In [20]:

```
# Apply dummy encoding, keeping "Weekday" as the reference category
df = pd.get_dummies(df, columns=["DAY_OF_WEEK", "TIME_OF_DAY"], drop_first=True)

# Define independent variables (all columns except the target)
independent_vars = df.columns.difference(["COLLISION_SEVERITY"])

# Specify and calibrate the Ordered Logit Model
model = OrderedModel(
```

```
    df["COLLISION_SEVERITY"],
    df[independent_vars],
    distr="logit"
)

result = model.fit(method="bfgs")
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.908192
         Iterations: 24
         Function evaluations: 26
         Gradient evaluations: 26
                           OrderedModel Results
==============================================================================
Dep. Variable:     COLLISION_SEVERITY   Log-Likelihood:                -5944.1
Model:                   OrderedModel   AIC:                         1.190e+04
Method:            Maximum Likelihood   BIC:                         1.195e+04
Date:                Sun, 09 Mar 2025
Time:                        20:22:36
No. Observations:                6545
Df Residuals:                    6538
Df Model:                           4
==============================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
DAY_OF_WEEK_Weekend     0.1176      0.056      2.110      0.035       0.008
0.227
TIME_OF_DAY_E          -0.0462      0.062     -0.739      0.460      -0.169
0.076
TIME_OF_DAY_M          -0.1186      0.066     -1.794      0.073      -0.248
0.011
TIME_OF_DAY_N           0.2751      0.084      3.289      0.001       0.111
0.439
1/2                     0.4881      0.044     11.209      0.000       0.403
0.573
2/3                     0.6250      0.022     28.580      0.000       0.582
0.668
3/4                     0.7387      0.051     14.486      0.000       0.639
0.839
==============================================================================
```

In [21]:
```
# Compute and display Odds Ratios
odds_ratios = np.exp(result.params)
print("\nOdds Ratios:\n", odds_ratios)

# Predict probabilities for each severity level
predicted_probs = result.predict()

# Convert predictions to a DataFrame
predicted_probs_df = pd.DataFrame(predicted_probs)

# Determine the most likely severity level for each observation
df["Predicted_Severity"] = predicted_probs_df.idxmax(axis=1)

# Display performance metrics with a confusion matrix (objective results only)
```

```
confusion_matrix = pd.crosstab(df["COLLISION_SEVERITY"], df["Predicted_Severity"
                               rownames=['Actual'], colnames=['Predicted'])
print("\nConfusion Matrix:\n", confusion_matrix)
```

```
Odds Ratios:
 DAY_OF_WEEK_Weekend    1.124841
TIME_OF_DAY_E          0.954860
TIME_OF_DAY_M          0.888160
TIME_OF_DAY_N          1.316645
1/2                    1.629240
2/3                    1.868245
3/4                    2.093161
dtype: float64

Confusion Matrix:
 Predicted     0
Actual
1          4018
2          1945
3           504
4            78
```

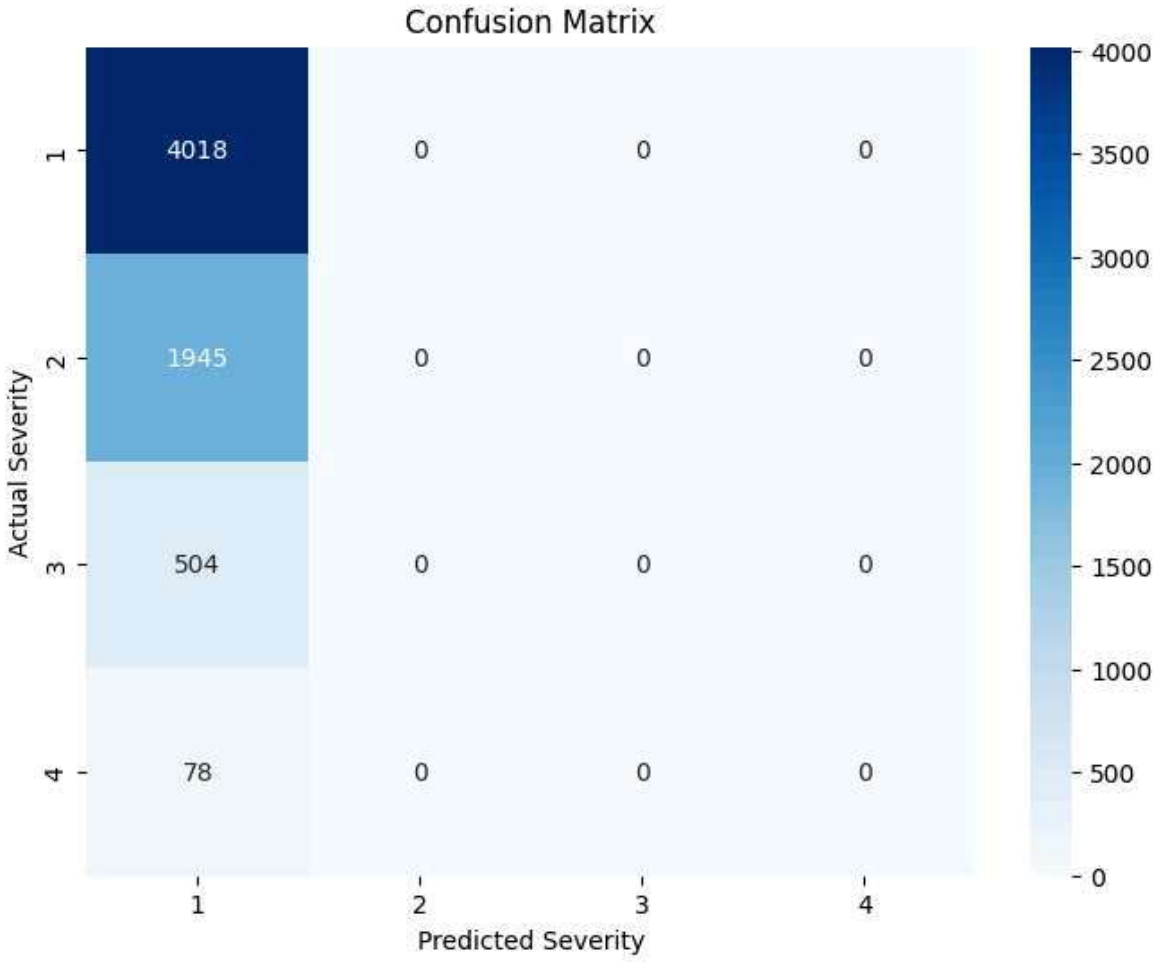
In [22]:
```
# Get the predicted class (severity level)
predicted = result.predict().argmax(axis=1) + 1  # +1 because argmax is zero-ina
actual = df["COLLISION_SEVERITY"]

# Create confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(actual, predicted)
print(conf_matrix)

# Visualize it
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=[1, 2, 3, 4],
            yticklabels=[1, 2, 3, 4])
plt.xlabel('Predicted Severity')
plt.ylabel('Actual Severity')
plt.title('Confusion Matrix')
plt.show()
```

```
[[4018    0    0    0]
 [1945    0    0    0]
 [ 504    0    0    0]
 [  78    0    0    0]]
```



Analysis for Santa Clara

In [7]:
```
import pandas as pd
import numpy as np
from statsmodels.miscmodels.ordinal_model import OrderedModel

def remap_severity(severity_series):
    """
    Remap collision severity values to a reversed scale and convert to an ordere

    The function converts severity scores from the original scale to a reversed
        1 → 4
        2 → 3
        3 → 2
        4 → 1

    Parameters:
    severity_series (pd.Series): Series containing collision severity values as

    Returns:
    pd.Categorical: An ordered categorical series with severity levels [1, 2, 3,
                    where 1 represents the least severe and 4 the most severe.
    """
    # Convert severity values to numeric, coercing errors to NaN
    severity_numeric = pd.to_numeric(severity_series, errors="coerce")
    # Define the mapping from original to reversed scale
    severity_mapping = {1: 4, 2: 3, 3: 2, 4: 1}
    remapped = severity_numeric.map(severity_mapping)
```

```
    # Define the order for the categorical variable
    severity_order = [1, 2, 3, 4]
    return pd.Categorical(remapped, categories=severity_order, ordered=True)

def categorize_time_of_day(time_str):
    """
    Categorize a time (given as a string) into a time-of-day group.

    The function interprets the input as a time in HHMM format and assigns:
        "M" for Morning (06:00 - 11:59),
        "A" for Afternoon (12:00 - 17:59),
        "E" for Evening (18:00 - 23:59),
        "N" for Night (00:00 - 05:59).

    Parameters:
    time_str (str): A string representing the time in HHMM format (e.g., "0600"

    Returns:
    str: A single character representing the time of day ("M", "A", "E", or "N")
         If the input is invalid, returns np.nan.
    """
    try:
        time_int = int(time_str)
        if 600 <= time_int < 1200:
            return "M"  # Morning
        elif 1200 <= time_int < 1800:
            return "A"  # Afternoon
        elif 1800 <= time_int < 2400:
            return "E"  # Evening
        elif 0 <= time_int < 600:
            return "N"  # Night
    except (ValueError, TypeError):
        return np.nan

# Load dataset
file_path = "Crashes.csv"  # Update with the actual file path
df = pd.read_csv("Crashes_SantaClara_2223.csv", dtype=str)

# Keep only relevant columns
columns_to_keep = ["COLLISION_SEVERITY", "DAY_OF_WEEK", "COLLISION_TIME"]
df = df[columns_to_keep]

# Apply severity remapping using the dedicated function
df["COLLISION_SEVERITY"] = remap_severity(df["COLLISION_SEVERITY"])

# Create TIME_OF_DAY by categorizing COLLISION_TIME and convert to a categorical
df["TIME_OF_DAY"] = df["COLLISION_TIME"].apply(categorize_time_of_day).astype("c
df.drop(columns=["COLLISION_TIME"], inplace=True)

# Collapse DAY_OF_WEEK into "Weekday" (Mon-Fri) vs. "Weekend" (Sat-Sun)
df["DAY_OF_WEEK"] = df["DAY_OF_WEEK"].apply(lambda x: "Weekday" if x in ["1", "2
```

In [8]:

```
# Apply dummy encoding, keeping "Weekday" as the reference category
df = pd.get_dummies(df, columns=["DAY_OF_WEEK", "TIME_OF_DAY"], drop_first=True)

# Define independent variables (all columns except the target)
independent_vars = df.columns.difference(["COLLISION_SEVERITY"])

# Specify and calibrate the Ordered Logit Model
model = OrderedModel(
```

```
    df["COLLISION_SEVERITY"],
    df[independent_vars],
    distr="logit"
)

result = model.fit(method="bfgs")
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.937131
         Iterations: 25
         Function evaluations: 27
         Gradient evaluations: 27
                           OrderedModel Results
==============================================================================
Dep. Variable:     COLLISION_SEVERITY   Log-Likelihood:                -12038.
Model:                   OrderedModel   AIC:                             2.409e+04
Method:            Maximum Likelihood   BIC:                             2.414e+04
Date:                Sun, 09 Mar 2025
Time:                        20:21:00
No. Observations:               12846
Df Residuals:                   12839
Df Model:                           4
==============================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
DAY_OF_WEEK_Weekend     0.2531      0.040      6.305      0.000       0.174
0.332
TIME_OF_DAY_E           0.3037      0.044      6.898      0.000       0.217
0.390
TIME_OF_DAY_M          -0.0368      0.045     -0.815      0.415      -0.125
0.052
TIME_OF_DAY_N           0.7751      0.061     12.649      0.000       0.655
0.895
1/2                     0.4439      0.029     15.083      0.000       0.386
0.502
2/3                     0.7670      0.014     53.855      0.000       0.739
0.795
3/4                     0.5195      0.036     14.361      0.000       0.449
0.590
==============================================================================
                           OrderedModel Results
==============================================================================
Dep. Variable:     COLLISION_SEVERITY   Log-Likelihood:                -12038.
Model:                   OrderedModel   AIC:                             2.409e+04
Method:            Maximum Likelihood   BIC:                             2.414e+04
Date:                Sun, 09 Mar 2025
Time:                        20:21:00
No. Observations:               12846
Df Residuals:                   12839
Df Model:                           4
==============================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
DAY_OF_WEEK_Weekend     0.2531      0.040      6.305      0.000       0.174
0.332
TIME_OF_DAY_E           0.3037      0.044      6.898      0.000       0.217
0.390
TIME_OF_DAY_M          -0.0368      0.045     -0.815      0.415      -0.125
0.052
TIME_OF_DAY_N           0.7751      0.061     12.649      0.000       0.655
```

```
0.895
1/2                    0.4439      0.029     15.083      0.000       0.386
0.502
2/3                    0.7670      0.014     53.855      0.000       0.739
0.795
3/4                    0.5195      0.036     14.361      0.000       0.449
0.590
==========================================================================
======
```

In [9]:
```python
# Compute and display Odds Ratios
odds_ratios = np.exp(result.params)
print("\nOdds Ratios:\n", odds_ratios)

# Predict probabilities for each severity level
predicted_probs = result.predict()

# Convert predictions to a DataFrame
predicted_probs_df = pd.DataFrame(predicted_probs)

# Determine the most likely severity level for each observation
df["Predicted_Severity"] = predicted_probs_df.idxmax(axis=1)

# Display performance metrics with a confusion matrix (objective results only)
confusion_matrix = pd.crosstab(df["COLLISION_SEVERITY"], df["Predicted_Severity"
                               rownames=['Actual'], colnames=['Predicted'])
print("\nConfusion Matrix:\n", confusion_matrix)
```

```
Odds Ratios:
 DAY_OF_WEEK_Weekend    1.288056
TIME_OF_DAY_E          1.354834
TIME_OF_DAY_M          0.963877
TIME_OF_DAY_N          2.170786
1/2                    1.558733
2/3                    2.153203
3/4                    1.681262
dtype: float64

Confusion Matrix:
 Predicted     0    1
Actual
1          6662  519
2          4026  528
3           726  160
4           162   63
```

In [10]:
```python
# Get the predicted class (severity level)
predicted = result.predict().argmax(axis=1) + 1  # +1 because argmax is zero-ina
actual = df["COLLISION_SEVERITY"]

# Create confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(actual, predicted)
print(conf_matrix)

# Visualize it
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=[1, 2, 3, 4],
            yticklabels=[1, 2, 3, 4])
plt.xlabel('Predicted Severity')
plt.ylabel('Actual Severity')
plt.title('Confusion Matrix')
plt.show()
```

```
[[6662  519    0    0]
 [4026  528    0    0]
 [ 726  160    0    0]
 [ 162   63    0    0]]
```