Predictive Modeling of Car Accident Severity in the USA from 2016-2023 using Machine Learning Techniques

[redacted]

November 2023

1 Introduction

As a student, who got their driver's license only a few months ago, I am eager, yet cautious to drive. Since I learned that teen drivers have a fatal crash rate almost three times as high as drivers ages 20 and older per mile driven (CDC), I try to keep my driving to a minimum. However, avoiding driving completely is not a practical solution. Being able to drive is an integral skill of our society. Moreover, to drive safely, experience is necessary, experience that I would lack by avoiding driving entirely.

Thus, to limit my driving risk, I have decided to derive a mathematical model to evaluate and advise me on the risk of driving in various driving scenarios. Through this mathematical model, I hope to understand the risks of driving more thoroughly in order to be safer and have more peace of mind when I am on the road.

This IA will also significantly extend my mathematical understanding of how advanced statistics, linear algebra, and multivariable calculus can be used to understand and visualize trends in data, determine the most important features through dimensionality reduction, and create an accurate and applicable Machine Learning model to evaluate the driving risk of a particular situation.

1.1 Research Question

What conditions result in the most dangerous driving accidents?

2 Dataset Overview

Throughout this paper, I will be using the US Accidents (2016 - 2023) Dataset (Moosavi, 2023). Before starting any analysis, it is essential to understand what data is provided. The data columns are described in appendix A.1, appendix A.2, appendix A.3, appendix A.4, and appendix A.5. All data processing, analysis, and visualization are performed in Python appendix A.6.

3 Data Cleaning and Preprocessing

Before analyzing this dataset, it must be cleaned and processed in order to derive accurate insights and predictive models.

3.1 Reporting Sources

The data was compiled from two sources, Bing and MapQuest. These sources report severity differently, thus I had to decide which source to utilize.

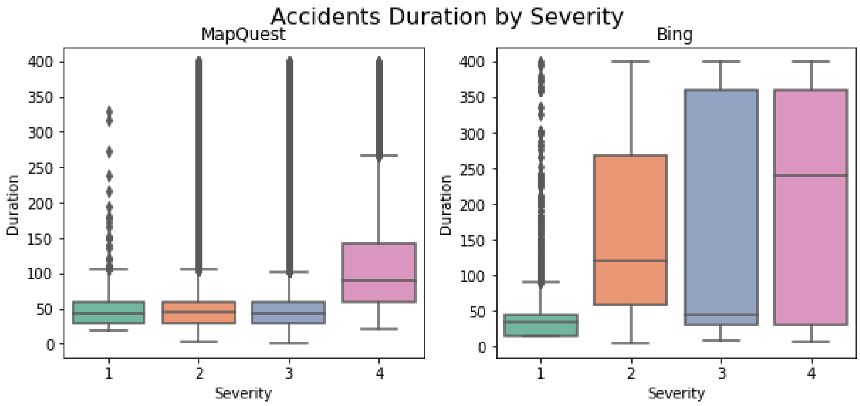

Figure 1: Accident Duration by Severity

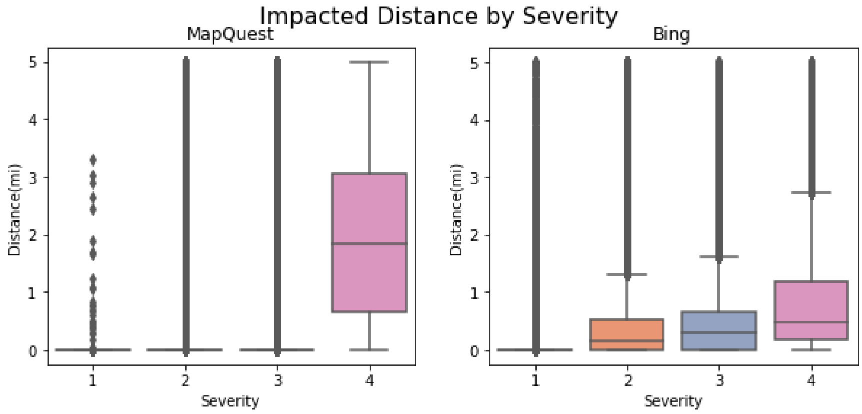

Figure 2: Impacted Distance by Severity

As can be seen by Figures 1 and 2, MapQuest has a clearly higher quartile 1, quartile 2 (median), and quartile 3 for severity 4. Bing seems to have a looser definition as the box and whisker plots are not too statistically significant from each other. For these reasons, I decided to only utilize the MapQuest data.

3.2 Non-Predictive Features

Data for columns 'ID', 'Distance(mi)', 'End Time', 'Duration', 'End Latitude', and 'End Longitude' can only be collected after the accident has already happened and hence cannot be used to predict severe accidents. Furthermore, since categorical variables like 'Country' and 'Turning Loop' only have one class, they cannot be predictive features as the probabilistic surprise and entropy for these random variables is 0 (Lesne, 2014).

3.3 Filtering Incomplete Data

Though this dataset is mostly complete, some entries lack data for certain columns. Rather than removing these columns for all entries, I decided to remove the incomplete row entries instead as the original dataset contains over 7.7 million accidents. Even after this filtering, there is still ample data.

4 Understanding and Exploring Data

Based on Figures 1 and 2, the MapQuest accidents with severity 4 are much more serious than other severity accidents. These other levels of severity are hard to distinguish from each other. Thus, I decided to focus on severity 4 accidents (from now referred to as ”severe”) and group the other severity accidents (from now referred to as ”non-severe”) together.

4.1 Data Sample Normalization

In this processed dataset, there are roughly 2.61 million non-severe accidents and only 9000 severe accidents. This discrepancy must be normalized before conducting any further exploratory analysis. I created a Python function to return a random sample of 50000 undersampled non-severe and 50000 oversampled severe accidents.

4.2 Trends based on Time

Next, I decided to explore if there were trends based on time.

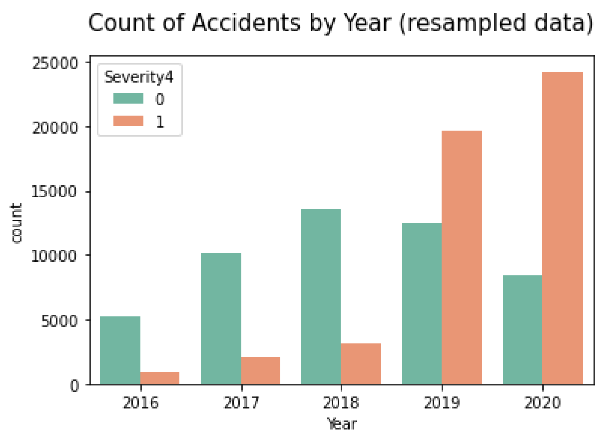

Figure 3: Count of Accidents by Year

Looking at Figure 3, it is highly improbable that severe accidents rose by 5 times from 2018 to 2019. To investigate this, I created a heatmap of these severe accidents to see how the data is actually distributed.

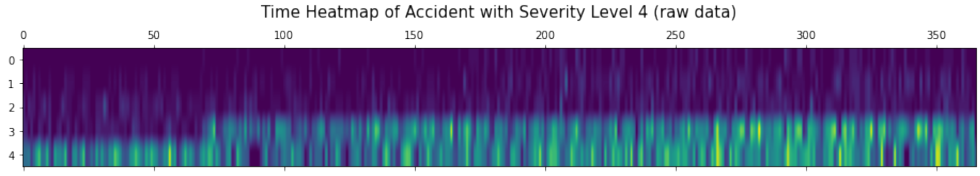

Figure 4: Time Heatmap of Severe Accidents

This heatmap strongly indicates that something changed after February 2019, such as the way that MapQuest defines severity or the way they collected data. Since the data after February 2019 is consistent with data in the future, dropping the data before March 2019 is the best choice for analysis and predictions.

4.3 Log Frequency Normalization

Next, I decided to analyze the accidents per hour. In Figure 5, there are two clear peaks in non-severe accidents occurring at roughly at 7-8 am and 4-5 pm, which likely correlate to maximum traffic times due to work-home commutes.

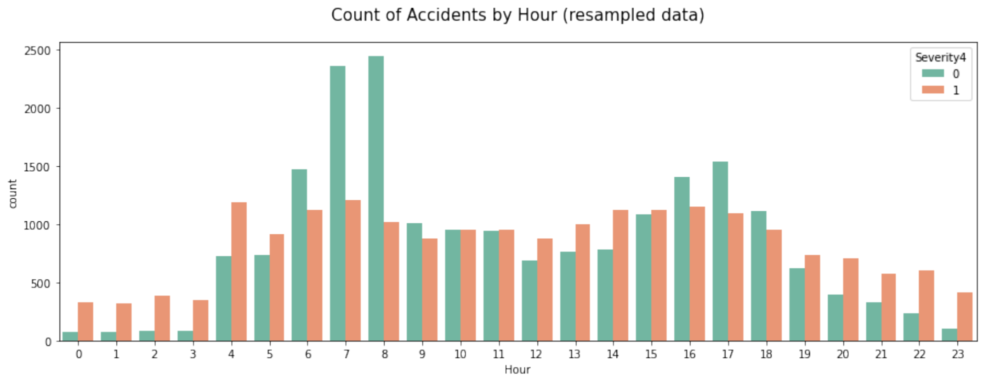

Figure 5: Count of Accidents by Hour

It is also interesting to note that during the hours (early morning or late night) where there were less non-severe accidents, there tended to be relatively greater severe accidents. This observation could be made clearer by using other data for a specific day, hour, or minute.

Since there are too many classes for a categorical variable for hours, I took the frequency of accidents during the hour. However, since certain hours could have a relatively low frequency, while others could have a relatively high frequency, the log of the frequency is taken to normalize the random continuous variable into a Gaussian distribution. Just in case the frequency is 0, 1 is added to all frequencies before taking the logarithm to prevent undefined values. This log frequency normalization (MaCurdy and Pencavel, 1986) is defined in Equation 1.

$$X = \log\left(\left(\frac{n_i}{\sum_j^k n_j} \times N_u\right) + 1\right) \tag{1}$$

where:

- X is the resulting continuous random variable after normalization.
- n_i is the number of times a possible class of a categorical variable occurs.
- $\sum_j^k n_j$ is the total number of samples.
- N_u is the number of unique classes that the categorical variable can take.

Severe	Hour	Hour Freq
1	22	3
0	14	2
0	5	1
0	22	3
0	14	2
1	22	3

Table 1: Example Data for Hour Log Frequency Normalization

For example, using the example data in Table 1, the hour log frequency normalization for the first row can be performed using Equation 2.

$$X = \log\left(\left(\frac{n_i}{\sum_j^k n_j} \times N_u\right) + 1\right) \tag{2}$$

$$= \log\left(\left(\frac{3}{6} \times 24\right) + 1\right) \tag{3}$$

$$= \log 13 \approx 2.56 \tag{4}$$

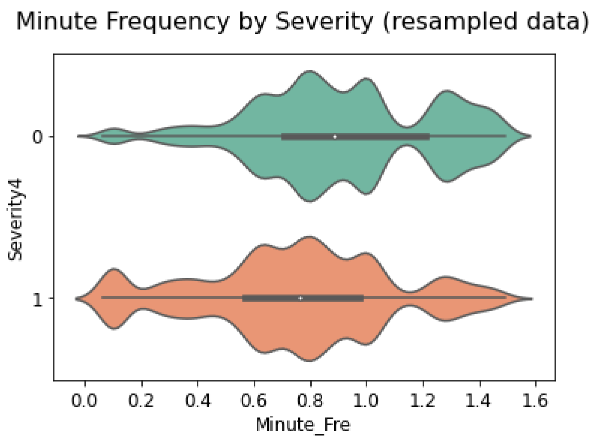

Figure 6: Accidents by Location Frequency

4.4 Location

Figure 7 shows the distribution of accidents based on latitude and longitude. The violin plots for non-severe and severe accidents seem to be very similar, though there are subtle changes in the ratio of the types of accidents.

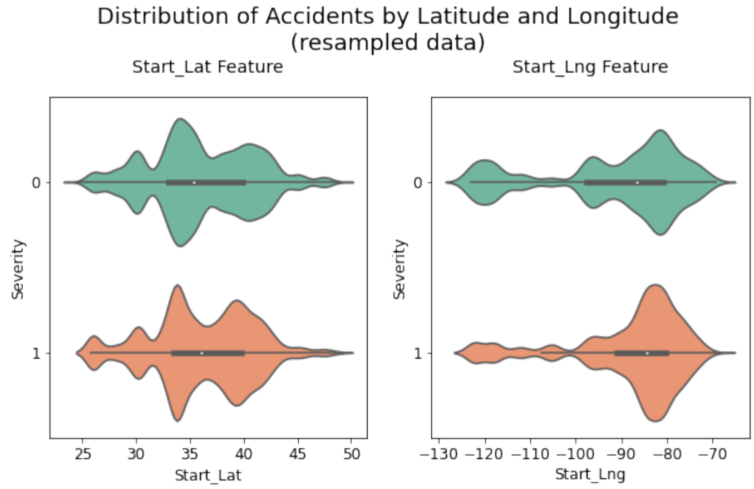

Figure 7: Accidents by Latitude and Longitude

In Figure 8, I plotted the latitude and longitude. I was fascinated to see how clearly the accidents were distributed across the US. The figure clearly shows more densely populated areas tended to have more accidents. More interesting, however, was how the severe accidents seemed to most densely be located in network-like structure. This seemed to indicate the US interstate highway system was where the greatest number of accidents seemed to occur.

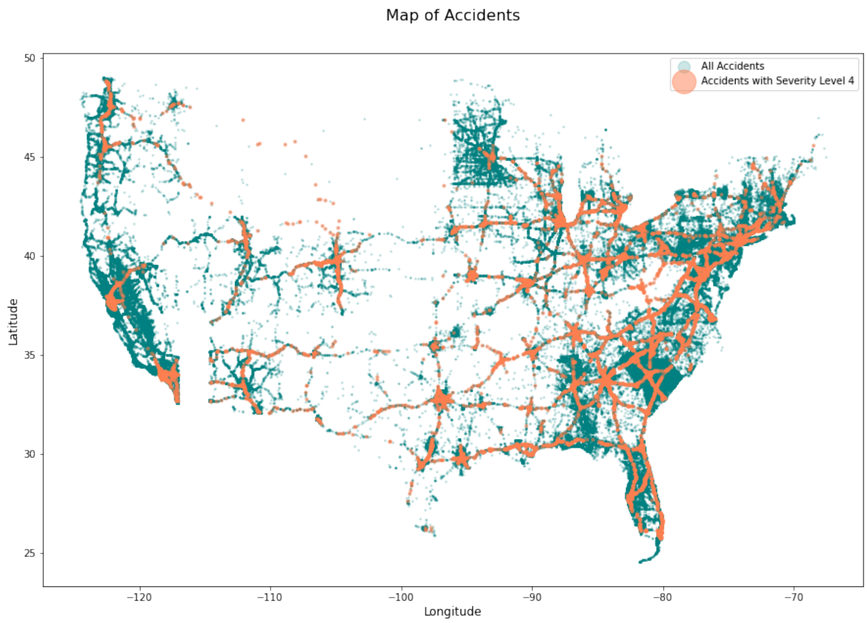

Figure 8: Map of Accidents

Using Equation 1, I used the frequency of local indicators such as 'Street', 'City', 'County', 'Zipcode', 'Airport Code', and 'State' to try to capture the clear correlation to the highway accidents in continuous random variables.

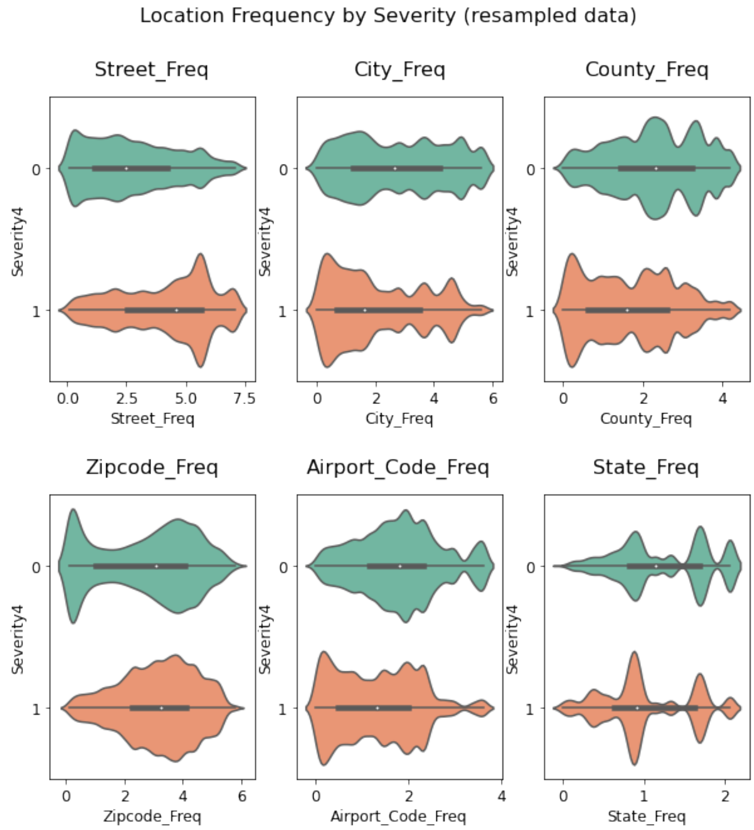

Figure 9: Accidents by Location Frequency

4.5 Weather Features

Since the weather features such as 'Pressure', 'Visibility', and 'Wind Speed' are highly skewed, they must be normalized prior to analysis. A Box Cox transformation (Osborne, 2010) can turn non-normal dependent variables into a normal Gaussian shape. By precisely controlling the parameter λ, I was able to normalize the weather features.

$$y(\lambda) = \begin{cases} \frac{y^{\lambda}-1}{\lambda} & \lambda \neq 0 \\ \log y & \lambda = 0 \end{cases} \tag{5}$$

Overall, Figure 10 shows that accidents are little more likely to be serious during rain or snow while less likely on a cloudy day.

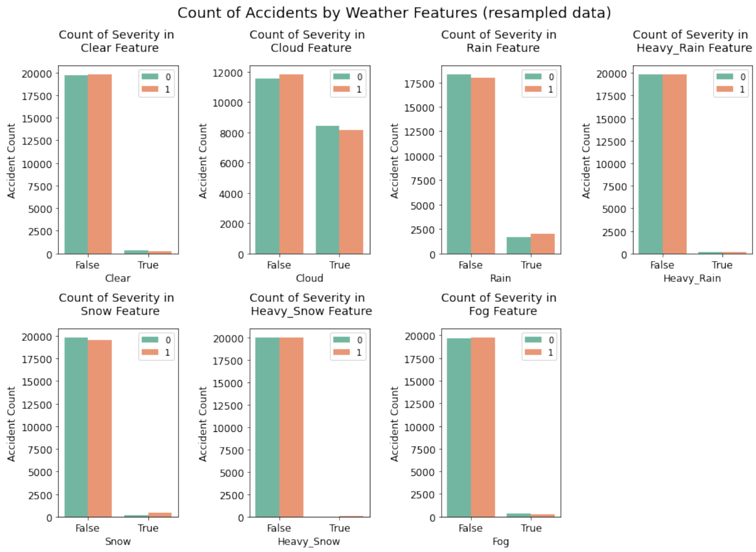

Figure 10: Accidents by Weather

5 Point of Interest Features

Figure 11 shows the number of accidents that occurred at Points of Interest (POI).

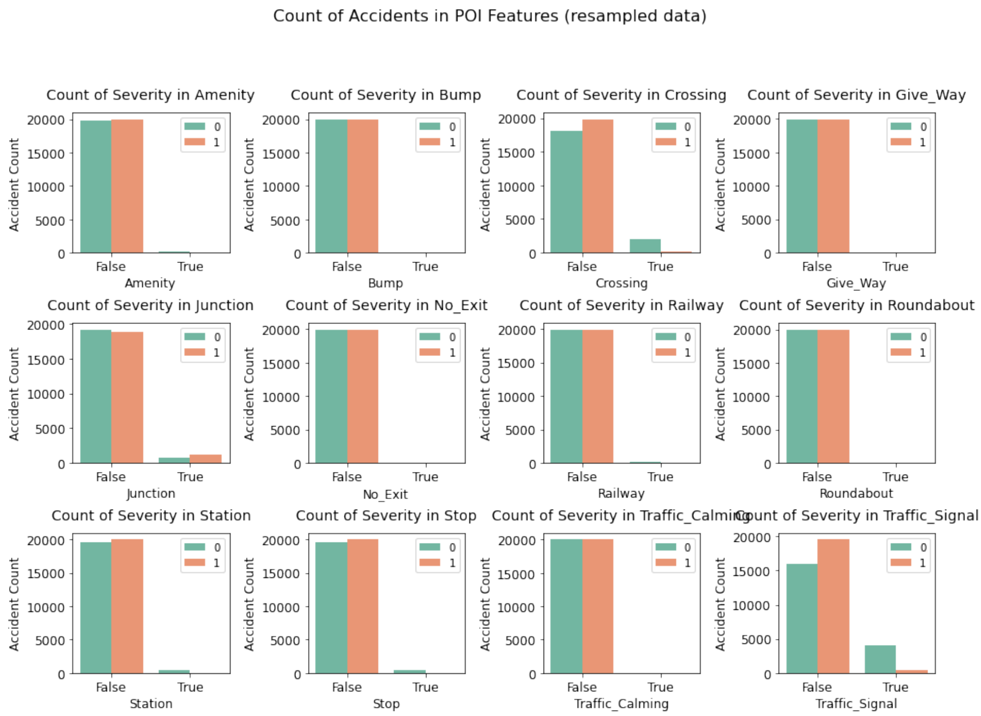

Figure 11: Accidents by Location Frequency

Accidents near traffic signal and crossing are much less likely to be serious accidents while little more likely to be serious if they are near the junction. This may occur because drivers usually slow down in front of crossing and traffic signal but junction and severity are highly related to speed. The other POI features have such little severe accidents that it is hard to tell their relation with severity from plots.

5.1 Heatmap Correlation

After all the normalization and processing performed, the following correlation heatmap was derived.

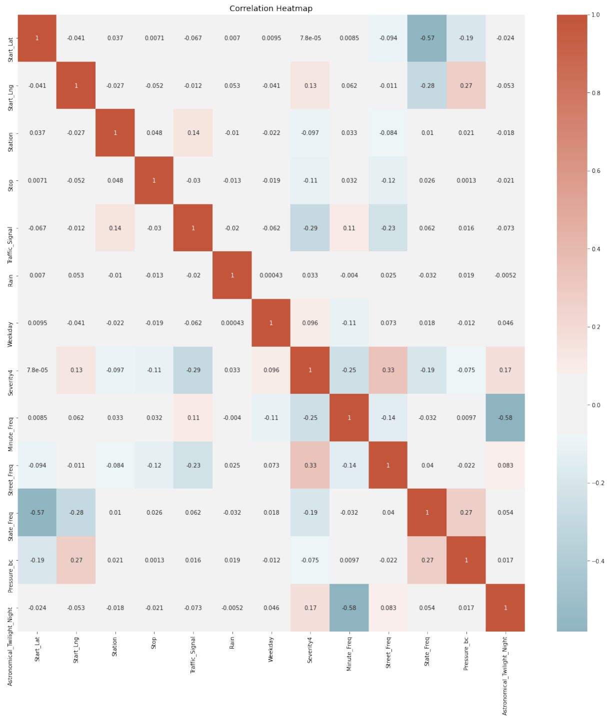

Figure 12: Correlation Heatmap

This heatmap represents the Pearson correlation coefficient (Equation 6) for each pair of continuous random variables in the dataset.

$$r = \frac{N\sum XY - (\sum X \sum Y)}{\sqrt{[N\sum x^2 - (\sum x)^2][N\sum y^2 - (\sum y)^2]}} \tag{6}$$

The Correlation Heatmap is promising as it shows both mildly strong positive and negative correlations of the various data columns to the severity, while being minimally correlated to each other.

This should in theory should enable enable a predictive model to optimize weights of select factors in order to derive a strong fit.

6 Predictive Modeling

Using the processed data from Section 4, I finally could start on designing a predictive model.

6.1 Principal Component Analysis

Even though I had majorly reduced the dimensionality of the data from 48 columns to only 12 columns through the processing steps, this was still too much data to pass into a predictive model. As a result, I decided to use Principal Component Analysis (PCA), which reduces data dimensionality while retaining most of the relevant information.

PCA involves:

1. Centering the data about the origin.
   - $X_i' = X_i - \bar{X}$
2. Calculating the covariance matrix with n variables.
   - $Cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i' - \bar{X}\right)\left(Y_i' - \bar{Y}\right)$
3. Compute the eigenvalues λ and eigenvectors v of the covariance matrix C.
   - $Cv = \lambda v$
4. Selecting the top k principal component eigenvalues and corresponding eigenvectors.
5. Projecting an original point (X) onto the new feature space along the ith principle component v_i.
   - $X' = X\dot{v}_i$
6. The principal components are linear combinations of the original variables. a_{ij} represents the components of the ith eigenvector. This represents the general form for the principal component i.
   - $PC_i = a_{i1}X_1' + a_{i2}X_2' + \cdots + a_{in}X_n'$

After performing PCA, the explained variance ratio can be derived using the Equation 7. Using PC1, PC2, ..., and PC8, over 80% of the variance in the accident severity is captured. This is described in the Scree Plot in Figure 13.

$$\text{Explained Variance} = \frac{\lambda_i}{\lambda_1 + \lambda_2 + \cdots + \lambda_n} \tag{7}$$

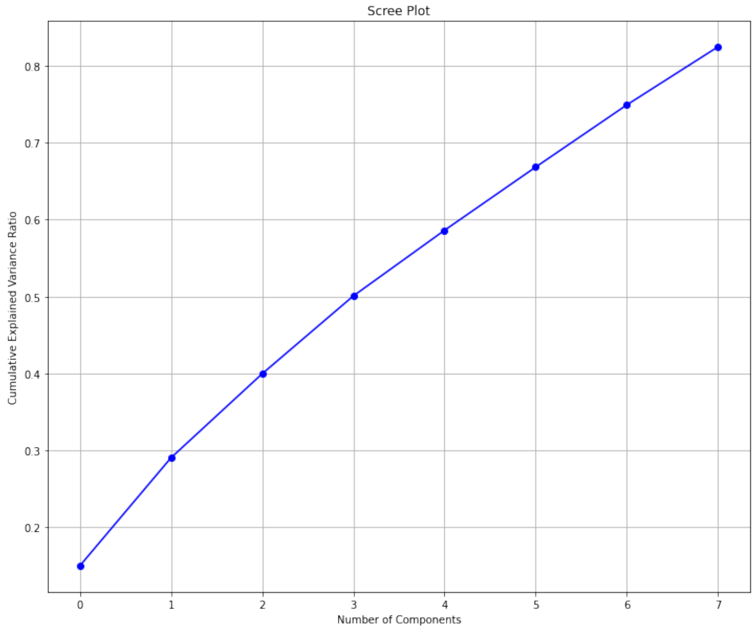

Figure 13: Scree Plot

6.2 Binary Logisitic Regression

In order to prevent overfitting and address the bias-variance tradeoff (Belkin et al., 2019), I split the data into 80% training and 20% testing. Using the PCs derived in the previous step, I decided to try using a binary logistic regression.

A binary logistic regression is defined by Equation 8.

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{8}$$

To tune the parameters β_0 and β_1, a maximum likelihood interpretation is utilized. Equation 9 describes the calculation for joint likelihood $L(\beta_0, \beta)$.

$$L(\beta_0, \beta) = \prod_{i=1}^{n} (p(x_i))^{y_i} \times (1 - p(x_i))^{1-y_i} \tag{9}$$

This can further be simplified by taking the log-likelihood $l(\beta_0, \beta)$ described in Equation 10

$$l(\beta_0, \beta) = \sum_{i=1}^{n} y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i)) \tag{10}$$

After taking the partial derivative of the log-likelihood$l(\beta_0, \beta)$ with respect to β_0 and β parameters individually and simplifying, Equation 11 describes the gradient of the β_0 and β_1. By numerically stepping the β_0 and β_1 using the gradient from Equation 11, the likelihood that the data was derived from the fitted logistic regression is maximized (Menard, 2002).

$$\frac{\partial l}{\partial \beta_j} = -\sum_{i=1}^{n} \left(y_i - p\left(x_i; \beta_0, \beta\right)\right) x_{ij} \tag{11}$$

6.2.1 Results

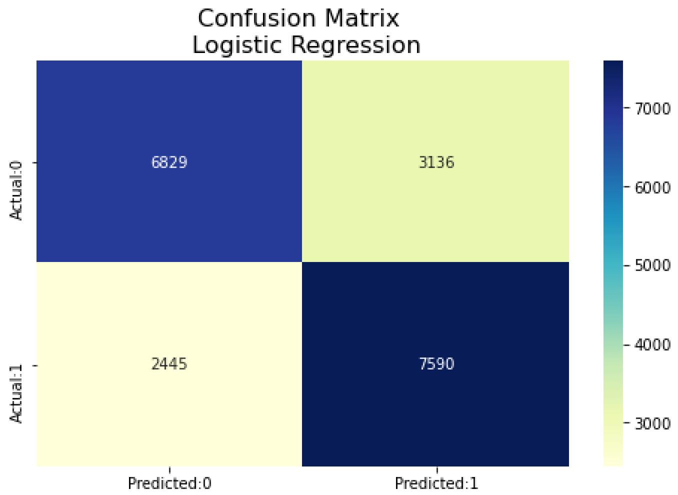

Figure 14: Logistic Regression Confusion Matrix

Using Figure 14 and Equations 13, 16, and 19, I calculated the sensitivity, specificity, and accuracy of my model.

$$\begin{aligned}
\text{Sensitivity} &= \frac{TP}{TP + FN} && (12)\\
&= \frac{7590}{7590 + 2445} && (13)\\
&\approx 75.6\% && (14)\\
\text{Specificity} &= \frac{TN}{TN + FP} && (15)\\
&= \frac{6829}{6829 + 3136} && (16)\\
&\approx 68.5\% && (17)\\
\text{Accuracy} &= \frac{TP + TN}{TP + FP + FN + TN} && (18)\\
&= \frac{7590 + 6829}{7590 + 3136 + 2445 + 6829} && (19)\\
&\approx 72.1\% && (20)
\end{aligned}$$

where:

- TP is the number of True Positives.
- TN is the number of True Negatives.

- FP is the number of False Positives.
- FN is the number of False Negatives.

As can be seen in Equations 13, 16, and 19, the model has a relatively low sensitivity, specificity, and accuracy. I believe that a stronger predictive model can be created.

6.3 Random Forest Classifier

Given relatively weak results from Section 6.2, I decided to try the more advanced Random Forest Classifier using GridSearchCV. Random Forest classifier is a machine learning model that combines the predictions of multiple decision trees to make more accurate and robust classifications. It works by creating a forest of decision trees, each trained on a random subset of the data and using a random subset of the features. The individual tree predictions are then aggregated to make the final classification decision, often through a majority vote or weighted average. Random Forests are known for their ability to handle high-dimensional data, reduce overfitting, and provide feature importance rankings, making them a popular choice for a wide range of classification tasks, from image recognition to financial risk assessment.

6.3.1 Results

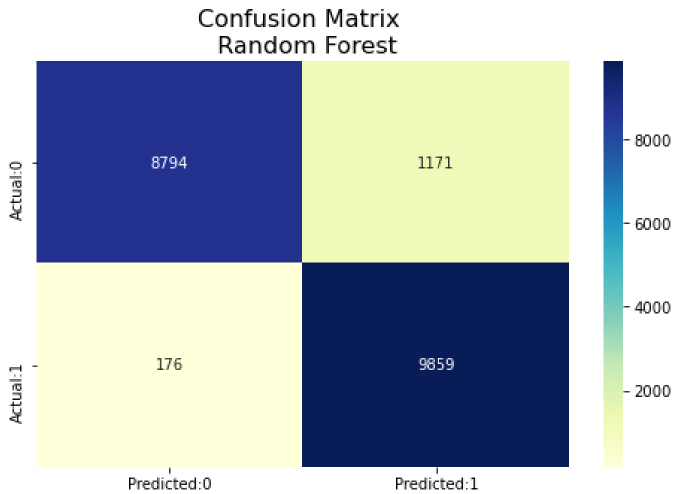

Figure 15: Random Forest Confusion Matrix

$$
\begin{aligned}
\text{Sensitivity} &= \frac{TP}{TP + FN} && (21)\\
&= \frac{9859}{9859 + 176} && (22)\\
&\approx 98.2\% && (23)\\
\text{Specificity} &= \frac{TN}{TN + FP} && (24)\\
&= \frac{8794}{8794 + 1171} && (25)\\
&\approx 88.2\% && (26)\\
\text{Accuracy} &= \frac{TP + TN}{TP + FP + FN + TN} && (27)\\
&= \frac{9859 + 8794}{9859 + 1171 + 176 + 8794} && (28)\\
&\approx 93.3\% && (29)
\end{aligned}
$$

Figure 15 and Equations 22, 25, and 28 clearly show that the Random Forest Model is more sensitive, specific, and accurate than the Logisitic Model in Section 6.2. Since the sensitivity (true positive rate) is 98% and the specificity (true negative rate) is 88%, the Random Forest Classifier seems to miscategorize more non-severe accidents as severe (Zhu et al., 2010). This is acceptable for the practical application of this model as overestimating the severity of accidents would bring greater public awareness of dangerous driving conditions.

7 Conclusion

The most accurate predictive model that I created was the Random Forest Classifier with a 93%. Figure 16 shows the most significant feature weights that the model iteratively learned.

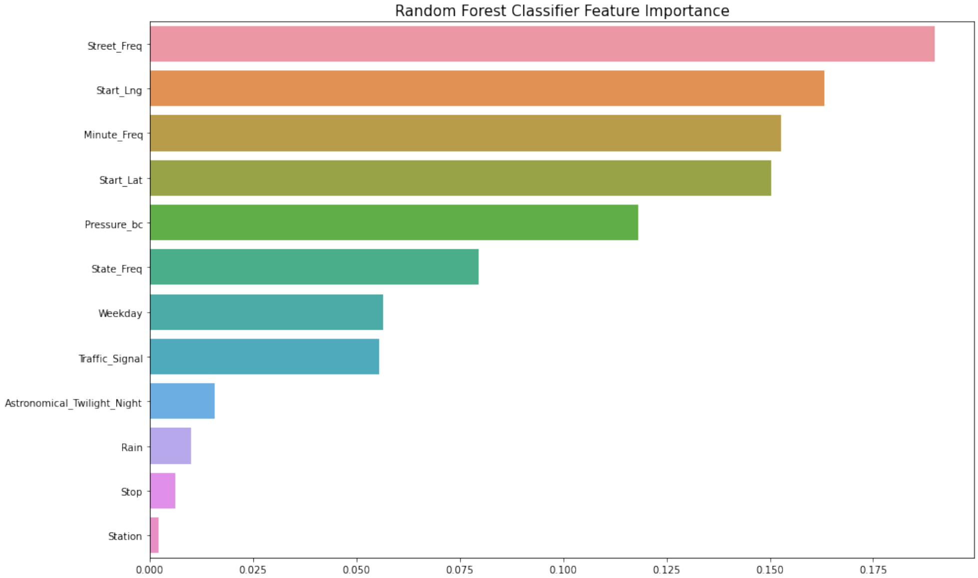

Figure 16: Random Forest Classifier Feature Importance

The 5 most important features it determined were:

1. Street Frequency
2. Start Longitude
3. Minute Frequency
4. Start Latitude
5. Pressure (Box-Cox)

I am not surprised that Street Frequency, Start Longitude, and Start Latitude were in the top 5. As seen in Section 4, most severe accidents seemed to occur on interstate highways in densely populated regions (based on longitude and latitude). Along with the temporal input that the Minute Frequency feature provided, the pattern clearly indicates that severe accidents are likely to occur in the same location and time as severe accidents that have occurred in the past.

Unlike the other factors, I was surprised by the Pressure (Box-Cox) feature being so important. Since a strong negative correlation was found for this factor, I wonder if pressure plays an indirect role in causing severe accidents. This is unlikely to be due to the weather as none of the weather features like Rain seemed to have a strong correlation with severity.

I was also surprised to see factors like Astronomical Twilight Night and Traffic Signal to only be somewhat important in predicting severe accidents as these are often cited as common dangerous driving conditions. As a further exploration, I would like to investigate these anomalies.

References

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Lesne, A. (2014). Shannon entropy: a rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics. *Mathematical Structures in Computer Science*, 24(3):e240311.

MaCurdy, T. E. and Pencavel, J. H. (1986). Testing between competing models of wage and employment determination in unionized markets. *Journal of Political Economy*, 94(3):S3–S39.

Menard, S. (2002). *Applied logistic regression analysis*. Sage.

Moosavi, S. (2023). Us accidents (2016 - 2023).

Osborne, J. (2010). Improving your data transformations: Applying the box-cox transformation. *Practical Assessment, Research, and Evaluation*, 15(1):12.

Zhu, W., Zeng, N., Wang, N., et al. (2010). Sensitivity, specificity, accuracy, associated confidence interval and roc analysis with practical sas implementations. *NESUG proceedings: health care and life sciences, Baltimore, Maryland*, 19:67.

A Appendix

A.1 Traffic Attributes

Column	Description
ID	This is a unique identifier of the accident record.
Source	Indicates source of the accident report.
Severity	Shows the severity of the accident, a number between 1 and 4.
Start Time	Shows start time of the accident in local time zone.
End Time	Shows end time of the accident in local time zone.
Start Latitude	Shows latitude in GPS coordinates of the start point.
Start Longitude:	Shows longitude in GPS coordinate of the start point.
End Latitude	Shows latitude in GPS coordinate of the end point.
End Longitude	Shows longitude in GPS coordinate of the end point.
Distance(mi)	The length of the road extent affected by the accident.
Description	Shows natural language description of the accident.

A.2 Address Attributes

Column	Description
Number	Shows the street number in address field.
Street	Shows the street name in address field.
City	Shows the city in address field.
County	Shows the county in address field.
State	Shows the state in address field.
Zip Code	Shows the zipcode in address field.
Country	Shows the country in address field.
Timezone	Shows timezone based on the location of the accident.

A.3 Weather Attributes

Column	Description
Airport Code	Denotes the closest airport-based weather station.
Weather Timestamp	Shows the time-stamp of weather observation.
Temperature	Shows the temperature (in Fahrenheit).
Wind Chill	Shows the wind chill (in Fahrenheit).
Humidity	Shows the humidity (in percentage).
Pressure	Shows the air pressure (in inches).
Visibility	Shows visibility (in miles).
Wind Direction	Shows wind direction.
Wind Speed	Shows wind speed (in miles per hour).

Precipitation	Shows precipitation amount in inches, if there is any.
Weather Condition	Shows the weather condition.

A.4 Point-Of-Interest Attributes

Column	Description
Amenity	Indicates presence of amenity in a nearby location.
Bump	Indicates presence of speed bump or hump in a nearby location.
Crossing	Indicates presence of crossing in a nearby location.
Give Way	Indicates presence of give way sign in a nearby location.
Junction	Indicates presence of junction in a nearby location.
No Exit	Indicates presence of no exit sign in a nearby location.
Railway	Indicates presence of railway in a nearby location.
Roundabout	Indicates presence of roundabout in a nearby location.
Station	Indicates presence of station (bus, train, etc.) in a nearby location.
Stop	Indicates presence of stop sign in a nearby location.
Traffic Calming	Indicates presence of traffic calming means in a nearby location.
Traffic Signal	Indicates presence of traffic signal in a nearby location.
Turning Loop	Indicates presence of turning loop in a nearby location.

A.5 Period-of-Day Attributes

Column	Description
Sunrise Sunset	Shows the period of day based on sunrise/sunset.
Civil Twilight	Shows the period of day based on civil twilight.
Nautical Twilight	Shows the period of day based on nautical twilight.
Astronomical Twilight	Shows the period of day based on astronomical twilight.

A.6 Python Code for Data Processing, Analysis, and Visualization

```
import numpy as np
import pandas as pd
import json
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
from datetime import datetime
import glob
import seaborn as sns
import re
import os
```

```python
import io
from scipy.stats import boxcox

df = pd.read_csv('../input/us-accidents/US_Accidents_March23.csv')
print("The shape of data is:",(df.shape))
print(df.head(3))

df_source = df.groupby(['Severity','Source']).size().reset_index().pivot(\
    columns='Severity', index='Source', values=0)
df_source.plot(kind='bar', stacked=True, title='Severity Count by Sources')

# fix datetime type
df['Start_Time'] = pd.to_datetime(df['Start_Time'])
df['End_Time'] = pd.to_datetime(df['End_Time'])
df['Weather_Timestamp'] = pd.to_datetime(df['Weather_Timestamp'])

# calculate duration as the difference between end time and start time in minute
df['Duration'] = df.End_Time - df.Start_Time
df['Duration'] = df['Duration'].apply(lambda x:round(x.total_seconds() / 60) )
print("The overall mean duration is: ", (round(df['Duration'].mean(),3)), 'min')

fig, axs = plt.subplots(ncols=2, figsize=(10, 4))
sns.boxplot(x="Severity", y="Duration",
               data=df.loc[(df['Source']=="Source2") & (df['Duration']<400),], palette="Set2", ax=axs[0])
axs[0].set_title('MapQuest')
fig.suptitle('Accidents Duration by Severity', fontsize=16)
sns.boxplot(x="Severity", y="Duration",
               data=df.loc[(df['Source']=="Source1") & (df['Duration']<400),], palette="Set2", ax=axs[1])
axs[1].set_title('Bing')
plt.show()
```

```
fig, axs = plt.subplots(ncols=2, figsize=(10, 4))
sns.boxplot(x="Severity", y="Distance(mi)",
            data=df.loc[(df['Source']=="Source2") &
             ↪  (df['Distance(mi)']<10),], palette="Set2",
             ↪  ax=axs[0])
axs[0].set_title('MapQuest')
fig.suptitle('Impacted Distance by Severity', fontsize=16)
sns.boxplot(x="Severity", y="Distance(mi)",
            data=df.loc[(df['Source']=="Source1") &
             ↪  (df['Distance(mi)']<10),], palette="Set2",
             ↪  ax=axs[1])
axs[1].set_title('Bing')
plt.show()

df = df.loc[df['Source']=="Source2",]
df = df.drop(['Source'], axis=1)
print("The shape of data is:",(df.shape))

df = df.drop(['ID','Description','Distance(mi)', 'End_Time',
 ↪  'Duration',
              'End_Lat', 'End_Lng'], axis=1)

cat_names = ['Country', 'Timezone', 'Amenity', 'Bump',
 ↪  'Crossing',
             'Give_Way', 'Junction', 'No_Exit', 'Railway',
              ↪  'Roundabout', 'Station',
             'Stop', 'Traffic_Calming', 'Traffic_Signal',
              ↪  'Turning_Loop', 'Sunrise_Sunset',
             'Civil_Twilight', 'Nautical_Twilight',
              ↪  'Astronomical_Twilight']
print("Unique count of categorical features:")
for i in cat_names:
  print(i,df[i].unique().size)

df = df.drop(['Country','Turning_Loop'], axis=1)

print("Wind Direction: ", df['Wind_Direction'].unique())

df.loc[df['Wind_Direction']=='Calm','Wind_Direction'] = 'CALM'
df.loc[(df['Wind_Direction']=='West')|(df['Wind_Direction']=='WSW')|(df['Wind_
 ↪  = 'W'
```

```python
df.loc[(df['Wind_Direction']=='South')|(df['Wind_Direction']=='SSW')|(df['Winc
    = 'S'
df.loc[(df['Wind_Direction']=='North')|(df['Wind_Direction']=='NNW')|(df['Winc
    = 'N'
df.loc[(df['Wind_Direction']=='East')|(df['Wind_Direction']=='ESE')|(df['Wind_
    = 'E'
df.loc[df['Wind_Direction']=='Variable','Wind_Direction'] = 'VAR'
print("Wind Direction after simplification: ", df['Wind_Direction'].unique())

# show distinctive weather conditions
weather ='!'.join(df['Weather_Condition'].dropna().unique().tolist())
weather = np.unique(np.array(re.split(
        "!|\s/\s|\sand\s|\swith\s|Partly\s|Mostly\s|Blowing\s|Freezing\s", weather))).tolist()
print("Weather Conditions: ", weather)

df['Clear'] = np.where(df['Weather_Condition'].str.contains('Clear', case=False, na = False), True, False)
df['Cloud'] = np.where(df['Weather_Condition'].str.contains('Cloud|Overcast', case=False, na = False), True, False)
df['Rain'] = np.where(df['Weather_Condition'].str.contains('Rain|storm', case=False, na = False), True, False)
df['Heavy_Rain'] = np.where(df['Weather_Condition'].str.contains('Heavy Rain|Rain Shower|Heavy T-Storm|Heavy Thunderstorms', case=False, na = False), True, False)
df['Snow'] = np.where(df['Weather_Condition'].str.contains('Snow|Sleet|Ice', case=False, na = False), True, False)
df['Heavy_Snow'] = np.where(df['Weather_Condition'].str.contains('Heavy Snow|Heavy Sleet|Heavy Ice Pellets|Snow Showers|Squalls', case=False, na = False), True, False)
df['Fog'] = np.where(df['Weather_Condition'].str.contains('Fog', case=False, na = False), True, False)
```

```
# Assign NA to created weather features where 'Weather_Condition' is null.
weather = ['Clear','Cloud','Rain','Heavy_Rain','Snow','Heavy_Snow','Fog']
for i in weather:
    df.loc[df['Weather_Condition'].isnull(),i] = df.loc[df['Weather_Condition'].isnull(),'Weather_Condition']
    df[i] = df[i].astype('bool')

df.loc[:,['Weather_Condition'] + weather]

df = df.drop(['Weather_Condition'], axis=1)

# average difference between weather time and start time
print("Mean difference between 'Start_Time' and 'Weather_Timestamp': ",
(df.Weather_Timestamp - df.Start_Time).mean())

df = df.drop(["Weather_Timestamp"], axis=1)

df['Year'] = df['Start_Time'].dt.year

nmonth = df['Start_Time'].dt.month
df['Month'] = nmonth

df['Weekday']= df['Start_Time'].dt.weekday

days_each_month = np.cumsum(np.array([0,31,28,31,30,31,30,31,31,30,31,30,31]))
nday = [days_each_month[arg-1] for arg in nmonth.values]
nday = nday + df["Start_Time"].dt.day.values
df['Day'] = nday

df['Hour'] = df['Start_Time'].dt.hour

df['Minute']=df['Hour']*60.0+df["Start_Time"].dt.minute

df.loc[:4,['Start_Time', 'Year', 'Month', 'Weekday', 'Day', 'Hour', 'Minute']]
```

```
missing = pd.DataFrame(df.isnull().sum()).reset_index()
missing.columns = ['Feature', 'Missing_Percent(%)']
missing['Missing_Percent(%)'] = missing['Missing_Percent(%)'].apply(lambda x: x / df.shape[0] * 100)
missing.loc[missing['Missing_Percent(%)']>0,:]

df = df.drop(['Wind_Chill(F)'], axis=1)

df['Precipitation_NA'] = 0
df.loc[df['Precipitation(in)'].isnull(),'Precipitation_NA'] = 1
df['Precipitation(in)'] = df['Precipitation(in)'].fillna(df['Precipitation(in)'].median())
df.loc[:5,['Precipitation(in)','Precipitation_NA']]

df = df.dropna(subset=['City','Zipcode','Airport_Code',
                        'Sunrise_Sunset','Civil_Twilight','Nautical_Twiligh

# group data by 'Airport_Code' and 'Start_Month' then fill NAs with median value
Weather_data=['Temperature(F)','Humidity(%)','Pressure(in)','Visibility(mi)','
print("The number of remaining missing values: ")
for i in Weather_data:
  df[i] = df.groupby(['Airport_Code','Month'])[i].apply(lambda x: x.fillna(x.median()))
  print( i + " : " + df[i].isnull().sum().astype(str))

df = df.dropna(subset=Weather_data)

# group data by 'Airport_Code' and 'Start_Month' then fill NAs with majority value
from collections import Counter
weather_cat = ['Wind_Direction'] + weather
print("Count of missing values that will be dropped: ")
for i in weather_cat:
  df[i] = df.groupby(['Airport_Code','Month'])[i].apply(lambda x: x.fillna(Counter(x).most_common()[0][0]) if all(x.isnull())==False else x)
  print(i + " : " + df[i].isnull().sum().astype(str))

```

```python
# drop na
df = df.dropna(subset=weather_cat)

df['Severity4'] = 0
df.loc[df['Severity'] == 4, 'Severity4'] = 1
df = df.drop(['Severity'], axis = 1)
df.Severity4.value_counts()

def resample(dat, col, n):
    return pd.concat([dat[dat[col]==1].sample(n, replace = True),
                     dat[dat[col]==0].sample(n)], axis=0)

df_bl = resample(df, 'Severity4', 50000)
print('resampled data:', df_bl.Severity4.value_counts())

df_bl.Year = df_bl.Year.astype(str)
sns.countplot(x='Year', hue='Severity4', data=df_bl,palette="Set2")
plt.title('Count of Accidents by Year (resampled data)', size=15, y=1.05)
plt.show()

# create a dataframe used to plot heatmap
df_date = df.loc[:,['Start_Time','Severity4']]          # create a new dateframe only containing time and severity
df_date['date'] = df_date['Start_Time'].dt.normalize() # keep only the date part of start time
df_date = df_date.drop(['Start_Time'], axis = 1)
df_date = df_date.groupby('date').sum()                # sum the number of accidents with severity level 4 by date
df_date = df_date.reset_index().drop_duplicates()

# join the dataframe with full range of date from 2016 to 2020
full_date = pd.DataFrame(pd.date_range(start="2016-01-02",end="2020-12-31"))
df_date = full_date.merge(df_date, how = 'left',left_on = 0, right_on = 'date')
df_date['date'] = df_date.iloc[:,0]
df_date = df_date.fillna(0)
df_date = df_date.iloc[:,1:].set_index('date')
```

```
# group by date
groups = df_date['Severity4'].groupby(pd.Grouper(freq='A'))
years = pd.DataFrame()
for name, group in groups:
    if name.year != 2020:
        years[name.year] = np.append(group.values,0)
    else:
        years[name.year] = group.values


# plot
years = years.T
plt.matshow(years, interpolation=None, aspect='auto')
plt.title('Time Heatmap of Accident with Severity Level 4 (raw
↪   data)', y=1.2, fontsize=15)
plt.show()

df = df.loc[df['Start_Time'] > "2019-03-10",:]
df = df.drop(['Year', 'Start_Time'], axis=1)
df['Severity4'].value_counts()

df_bl = resample(df, 'Severity4', 20000)

plt.figure(figsize=(10,5))
sns.countplot(x='Month', hue='Severity4', data=df_bl
↪   ,palette="Set2")
plt.title('Count of Accidents by Month (resampled data)',
↪   size=15, y=1.05)
plt.show()

plt.figure(figsize=(10,5))
sns.countplot(x='Weekday', hue='Severity4', data=df_bl
↪   ,palette="Set2")
plt.title('Count of Accidents by Weedday (resampled data)',
↪   size=15, y=1.05)
plt.show()

period_features =
↪   ['Sunrise_Sunset','Civil_Twilight','Nautical_Twilight','Astronomical_Twili
fig, axs = plt.subplots(ncols=1, nrows=4, figsize=(13, 5))

```

```
plt.subplots_adjust(wspace = 0.5)
for i, feature in enumerate(period_features, 1):
    plt.subplot(1, 4, i)
    sns.countplot(x=feature, hue='Severity4', data=df_bl ,palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Accident Count', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.legend(['0', '1'], loc='upper right', prop={'size': 10})
    plt.title('Count of Severity in\n{} Feature'.format(feature), size=13, y=1.05)
fig.suptitle('Count of Accidents by Period-of-Day (resampled data)',y=1.08, fontsize=16)
plt.show()

plt.figure(figsize=(15,5))
sns.countplot(x='Hour', hue='Severity4', data=df_bl ,palette="Set2")
plt.title('Count of Accidents by Hour (resampled data)', size=15, y=1.05)
plt.show()

# frequence encoding and log-transform
df['Minute_Freq'] = df.groupby(['Minute'])['Minute'].transform('count')
df['Minute_Freq'] = df['Minute_Freq']/df.shape[0]*24*60
df['Minute_Freq'] = df['Minute_Freq'].apply(lambda x: np.log(x+1))

# resampling
df_bl = resample(df, 'Severity4', 20000)

# plot
df_bl['Severity4'] = df_bl['Severity4'].astype('category')
sns.violinplot(x='Minute_Freq', y="Severity4", data=df_bl, palette="Set2")
plt.xlabel('Minute_Fre', size=12, labelpad=3)
plt.ylabel('Severity4', size=12, labelpad=3)
```

```python
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.title('Minute Frequency by Severity (resampled data)', size=16, y=1.05)
plt.show()

plt.figure(figsize=(6,5))
chart = sns.countplot(x='Timezone', hue='Severity4', data=df_bl ,palette="Set2")
plt.title("Count of Accidents by Timezone (resampled data)", size=15, y=1.05)
plt.show()

plt.figure(figsize=(25,5))
chart = sns.countplot(x='State', hue='Severity4',
                      data=df_bl ,palette="Set2", order=df_bl['State'].value_counts().index)
plt.title("Count of Accidents in State\nordered by accidents' count (resampled data)", size=15, y=1.05)
plt.show()

df_bl['Severity4'] = df_bl['Severity4'].astype('category')
num_features = ['Start_Lat', 'Start_Lng']
fig, axs = plt.subplots(ncols=1, nrows=2, figsize=(10, 5))
plt.subplots_adjust(hspace=0.4,wspace = 0.2)
for i, feature in enumerate(num_features, 1):
    plt.subplot(1, 2, i)
    sns.violinplot(x=feature, y="Severity4", data=df_bl, palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Severity', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.title('{} Feature'.format(feature), size=14, y=1.05)
fig.suptitle('Distribution of Accidents by Latitude and Longitude\n(resampled data)', fontsize=18,y=1.08)
plt.show()

df_4 = df[df['Severity4']==1]
```

```
plt.figure(figsize=(15,10))

plt.plot( 'Start_Lng', 'Start_Lat', data=df, linestyle='', marker='o', markersize=1.5, color="teal", alpha=0.2, label='All Accidents')
plt.plot( 'Start_Lng', 'Start_Lat', data=df_4, linestyle='', marker='o', markersize=3, color="coral", alpha=0.5, label='Accidents with Severity Level 4')
plt.legend(markerscale=8)
plt.xlabel('Longitude', size=12, labelpad=3)
plt.ylabel('Latitude', size=12, labelpad=3)
plt.title('Map of Accidents', size=16, y=1.05)
plt.show()

fre_list = ['Street', 'City', 'County', 'Zipcode', 'Airport_Code','State']
for i in fre_list:
  newname = i + '_Freq'
  df[newname] = df.groupby([i])[i].transform('count')
  df[newname] = df[newname]/df.shape[0]*df[i].unique().size
  df[newname] = df[newname].apply(lambda x: np.log(x+1))

# resample again
df_bl = resample(df, 'Severity4', 50000)

df_bl['Severity4'] = df_bl['Severity4'].astype('category')
fig, axs = plt.subplots(ncols=2, nrows=3, figsize=(10, 10))
plt.subplots_adjust(hspace=0.4,wspace = 0.2)
fig.suptitle('Location Frequency by Severity (resampled data)', fontsize=16)
for i, feature in enumerate(fre_list, 1):
    feature = feature + '_Freq'
    plt.subplot(2, 3, i)
    sns.violinplot(x=feature, y="Severity4", data=df_bl, palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Severity4', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)
```

```python

    plt.title('{}'.format(feature), size=16, y=1.05)
plt.show()

df = df.drop(fre_list, axis  = 1)

df['Pressure_bc']= boxcox(df['Pressure(in)'].apply(lambda x: x+1),lmbda=0.3)
df['Visibility_bc']= boxcox(df['Visibility(mi)'].apply(lambda x: x+1),lmbda = 0.1)
df['Wind_Speed_bc']= boxcox(df['Wind_Speed(mph)'].apply(lambda x: x+1),lmbda=-0.2)
df = df.drop(['Pressure(in)','Visibility(mi)','Wind_Speed(mph)'], axis=1)

# resample again
df_bl = resample(df, 'Severity4', 50000)

df_bl['Severity4'] = df_bl['Severity4'].astype('category')
num_features = ['Temperature(F)', 'Humidity(%)', 'Pressure_bc', 'Visibility_bc', 'Wind_Speed_bc']
fig, axs = plt.subplots(ncols=2, nrows=3, figsize=(15, 10))
plt.subplots_adjust(hspace=0.4,wspace = 0.2)
for i, feature in enumerate(num_features, 1):
    plt.subplot(2, 3, i)
    sns.violinplot(x=feature, y="Severity4", data=df_bl, palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Severity', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.title('{} Feature by Severity'.format(feature), size=14, y=1.05)
fig.suptitle('Density of Accidents by Weather Features (resampled data)', fontsize=18)
plt.show()

fig, axs = plt.subplots(ncols=2, nrows=4, figsize=(15, 10))
plt.subplots_adjust(hspace=0.4,wspace = 0.6)
```

```
for i, feature in enumerate(weather, 1):
    plt.subplot(2, 4, i)
    sns.countplot(x=feature, hue='Severity4', data=df_bl
     ,palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Accident Count', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.legend(['0', '1'], loc='upper right', prop={'size': 10})
    plt.title('Count of Severity in \n {} Feature'.format(feature), size=14, y=1.05)
fig.suptitle('Count of Accidents by Weather Features (resampled data)', fontsize=18)
plt.show()

df = df.drop(['Heavy_Rain','Heavy_Snow','Fog'], axis  = 1)

df = df.drop(['Wind_Direction'], axis=1)

POI_features = ['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Ro

fig, axs = plt.subplots(ncols=3, nrows=4, figsize=(15, 10))

plt.subplots_adjust(hspace=0.5,wspace = 0.5)
for i, feature in enumerate(POI_features, 1):
    plt.subplot(3, 4, i)
    sns.countplot(x=feature, hue='Severity4', data=df_bl
     ,palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Accident Count', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.legend(['0', '1'], loc='upper right', prop={'size': 10})
    plt.title('Count of Severity in {}'.format(feature), size=14,
     y=1.05)
```

```
fig.suptitle('Count of Accidents in POI Features (resampled data)',y=1.02, fontsize=16)
plt.show()

df = df.drop(['Amenity','Bump','Give_Way','No_Exit','Roundabout','Traffic_Calmi
axis=1)

dtype_df = df_bl.dtypes.reset_index()
dtype_df.columns = ["Count", "Column Type"]
print(dtype_df)

# one-hot encoding
df[period_features] = df[period_features].astype('category')
df = pd.get_dummies(df, columns=period_features, drop_first=True)

# plot correlation
df_bl['Severity4'] = df_bl['Severity4'].astype(int)
plt.figure(figsize=(25,25))
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
sns.heatmap(df_bl.corr(), annot=True,cmap=cmap, center=0).set_title("Correlation Heatmap", fontsize=14)
plt.show()

# Plot the proportion of severities.
plt.figure()
df_bl['Severity4'].value_counts().plot.pie(autopct='%1.1f%%')
plt.title('Percentage Severity Distribution')
plt.ylabel('Count')
plt.show()

df = df.drop(['Temperature(F)', 'Humidity(%)', 'Precipitation(in)', 'Precipitation_NA','Visibility_bc', 'Wind_Speed_bc',
                    'Clear','Cloud','Snow','Crossing','Junction','Railway','Mont
                'Hour', 'Day','Minute', 'City_Freq','County_Freq','Airport_Code_Freq','Zipcode_Freq'
                'Sunrise_Sunset_Night', 'Civil_Twilight_Night', 'Nautical_Twilight_Night'], axis=1)

```

```
df = df.drop(['Timezone'], axis=1)

# resample again
df_bl = resample(df, 'Severity4', 50000)

# plot correlation
df_bl['Severity4'] = df_bl['Severity4'].astype(int)
plt.figure(figsize=(20,20))
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
sns.heatmap(df_bl.corr(), annot=True,cmap=cmap,
↪   center=0).set_title("Correlation Heatmap", fontsize=14)
plt.show()

# Pre-process the dataset to extract the most important features
↪   to predict the severity of an accident
# Find all continuous variables
continuous_vars = df.select_dtypes(include=['float64',
↪   'int64']).columns
print('The Dataset Contains, Continuous Variables:
↪   {}'.format(continuous_vars))

# Find all categorical variables
categorical_vars = df.select_dtypes(include=['object', 'bool',
↪   'category']).columns
print('The Dataset Contains, Categorical Variables:
↪   {}'.format(categorical_vars))

labels = []
values = []
for col in continuous_vars:
    if col == 'Severity4':
        continue
    labels.append(col)
    values.append(np.corrcoef(df[col].values,
    ↪   df.Severity4.values)[0,1])
corr_df = pd.DataFrame({'col_labels':labels,
↪   'corr_values':values})
corr_df = corr_df.sort_values(by='corr_values')

ind = np.arange(len(labels))
width = 0.9
```

```
fig, ax = plt.subplots(figsize=(20,20))
rects = ax.barh(ind, np.array(corr_df.corr_values.values),
↪   color='b')
ax.set_yticks(ind)
ax.set_yticklabels(corr_df.col_labels.values,
↪   rotation='horizontal')
ax.set_xlabel("Correlation coefficient")
ax.set_title("Correlation coefficient of the variables")
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression  # You can
↪   use other models depending on your problem.
from sklearn.metrics import classification_report,
↪   confusion_matrix
from sklearn.metrics import confusion_matrix,
↪   ConfusionMatrixDisplay
from sklearn.impute import SimpleImputer
from sklearn.ensemble import AdaBoostClassifier,
↪   RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold,
↪   train_test_split, cross_val_predict
from sklearn.metrics import classification_report,
↪   confusion_matrix
x_train, x_test, y_train, y_test =
↪   train_test_split(df_bl.drop('Severity4', axis=1),
↪   df_bl['Severity4'], test_size=0.2, random_state=42)

my_imputer = SimpleImputer()
x_train = my_imputer.fit_transform(x_train)

# Perform PCA analysis with Skree plot to find the most important
↪   features
# Standardize the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(x_train)
print(df_scaled)

print(x_train.shape)
```

```
# Perform PCA
pca = PCA(n_components=8)
df_pca = pca.fit_transform(df_scaled)

# Plot the Skree plot
plt.figure(figsize=(12, 10))
plt.plot(np.cumsum(pca.explained_variance_ratio_), 'bo-')
plt.grid()
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Scree Plot')
plt.show()

x_test = my_imputer.fit_transform(x_test)
df_scaled_test = scaler.transform(x_test)
df_pca_test = pca.transform(df_scaled_test)
print(df_pca_test)

# Initialize and train a machine learning model (e.g., Logistic Regression)
model = LogisticRegression(max_iter=1000)
model.fit(x_train, y_train)

# Make predictions on the test set
y_pred = model.predict(x_test)

# Calculate the accuracy of the model
print(classification_report(y_test, y_pred))

confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)

conf_matrix = pd.DataFrame(data=confmat,
                                   columns=['Predicted:0','Predicted:1'],index=['
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu").set_title(
    "Confusion Matrix \n Logistic Regression", fontsize=16)
plt.show()

clf_base = RandomForestClassifier()
```

```
grid = {'n_estimators': [10, 50, 100],
        'max_features': ['auto','sqrt']}
clf_rf = GridSearchCV(clf_base, grid, cv=5, n_jobs=8,
 ↪   scoring='f1_macro')

clf_rf.fit(x_train, y_train)
y_pred = clf_rf.predict(x_test)

print(classification_report(y_test, y_pred))

confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)

conf_matrix = pd.DataFrame(data=confmat,

                            ↪   columns=['Predicted:0','Predicted:1'],index=['A
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix,
 ↪   annot=True,fmt='d',cmap="YlGnBu").set_title(
    "Confusion Matrix \n Random Forest", fontsize=16)
plt.show()

importances = pd.DataFrame(np.zeros((x_train.shape[1], 1)),
 ↪   columns=['importance'],
 ↪   index=df.drop('Severity4',axis=1).columns)

importances.iloc[:,0] =
 ↪   clf_rf.best_estimator_.feature_importances_

importances.sort_values(by='importance', inplace=True,
 ↪   ascending=False)
importances30 = importances.head(30)

plt.figure(figsize=(15, 10))
sns.barplot(x='importance', y=importances30.index,
 ↪   data=importances30)

plt.xlabel('')
plt.tick_params(axis='x', labelsize=10)
plt.tick_params(axis='y', labelsize=10)
plt.title('Random Forest Classifier Feature Importance', size=15)

```

```
plt.show()
```