Applied Analytics Project

Analyzing US Accident Data to Predict High-Risk Areas and Times in Massachusetts

Week 5 — Engineer Features

Major: Applied Analytics

Name: Gefan Wang, Chenhe Shi, Tianchen Liu

Date: 02.23.2025

Last week, we focused on preparing the dataset for modeling by performing data cleaning, handling missing values, and engineering preliminary features to enhance predictive accuracy. We also conducted a correlation analysis to better understand relationships between variables and split the dataset into training, validation, and test sets again. This week's report will focus on **Feature Engineering, Data Augmentation, and Dimensionality Reduction** to further refine our dataset and enhance model performance.

For the Feature Engineering part, we created new features by combining existing variables to extract more meaningful insights. We introduced a **Temp_Humidity_Index**, a composite feature derived from multiplying Temperature(F) with Humidity(%), which this new variable helps represent the relationship between temperature and moisture in affecting accident severity. We also added another key feature, **Wind_Impact**, which was calculated by multiplying Wind_Speed(mph) with Visibility(mi), used to capture how wind speed and reduced visibility correlate to hazard road conditions. These new features were evaluated using correlation analysis, which enhanced their relationship with the meteorological factors, suggesting their potential relevance for accident prediction.

```
# Feature Engineering: Creating new features
# Creating a feature combining Temperature and Humidity to represent heat index effect
data_modelling_df['Temp_Humidity_Index'] = data_modelling_df['Temperature(F)'] * data_modelling_df['Humidity(%)']

# Creating a feature that represents wind impact (Wind Speed * Visibility)
data_modelling_df['Wind_Impact'] = data_modelling_df['Wind_Speed(mph)'] * data_modelling_df['Visibility(mi)']
```

We applied data augmentation by introducing slight variations in numerical features to improve model robustness and generalization. We did this by adding **small amounts of Gaussian noise** to Wind_Speed(mph), Visibility(mi), Humidity(%), Temperature(F), and Pressure(in), simulating **real-world fluctuations**. By duplicating the dataset and incorporating these augmented variations, we expanded the dataset which allows the model to better generalize across similar but slightly altered conditions (adds more uncertainty to the real world application). This data augmentation step ensures that the model is exposed to a wider range of potential accident scenarios, which leads to predictive performance improvements.

```
# Data Augmentation: Creating slight variations in numerical data by adding small noise
augmented_data = data_modelling_df.copy()
for feature in numerical_features:
    augmented_data[feature] += np.random.normal(0, 0.01, augmented_data[feature].shape)

# Concatenating original and augmented data
data_augmented = pd.concat([data_modelling_df, augmented_data], ignore_index=True)
```

For the Dimensionality Reduction part, we applied Principal Component Analysis (PCA) to refine the dataset and maintain computational efficiency while preserving key information. PCA was performed on the numerical feature set, reducing dimensionality while retaining 95% of variance. The resulting principal components replaced original numerical variables, which minimized redundancy and prevented overfitting. After transformation, categorical features such as Weather_Condition and Wind_Direction were retained in their encoded forms to preserve essential classification elements. This dimensionality reduction step significantly improved processing efficiency and optimized feature selection for the next phase of modeling.

```python
# Dimensionality Reduction using PCA
pca = PCA(n_components=0.95)  # Retain 95% variance
principal_components = pca.fit_transform(data_augmented[numerical_features])

# Converting PCA result to DataFrame
pca_df = pd.DataFrame(principal_components, columns=[f'PC{i+1}' for i in range(principal_components.shape[1])])

# Merging PCA-transformed numerical data with categorical features
data_final = pd.concat([pca_df, data_augmented[categorical_features]], axis=1)
```

Next week: we will focus on “description” to do an unique text-classification.

Week5 Data processing and Feature engineering

February 23, 2025

1 Week 1.install and import necessary packages and import dataset

[1]:
```
# install and import necessary packages

import sys
import subprocess

# List of required packages
packages = ['numpy', 'pandas', 'matplotlib', 'seaborn','scikit-learn','plotly']

# Install missing packages
for package in packages:
    try:
        __import__(package)
    except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install", package])

#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import gc
from sklearn.model_selection import train_test_split
import plotly.express as px
from sklearn.preprocessing import LabelEncoder
```

```
Requirement already satisfied: scikit-learn in
/opt/anaconda3/lib/python3.12/site-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (3.5.0)
```

[2]:
```
#import data
accident_data = pd.read_csv("US_Accidents_MA.csv")
```

2 Week 2. Basic EDA

[3]:
```
#look at datatype
accident_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61996 entries, 0 to 61995
Data columns (total 46 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   ID                     61996 non-null  object
 1   Source                 61996 non-null  object
 2   Severity               61996 non-null  int64
 3   Start_Time             61996 non-null  object
 4   End_Time               61996 non-null  object
 5   Start_Lat              61996 non-null  float64
 6   Start_Lng              61996 non-null  float64
 7   End_Lat                7971 non-null   float64
 8   End_Lng                7971 non-null   float64
 9   Distance(mi)           61996 non-null  float64
 10  Description            61996 non-null  object
 11  Street                 61950 non-null  object
 12  City                   61996 non-null  object
 13  County                 61996 non-null  object
 14  State                  61996 non-null  object
 15  Zipcode                61996 non-null  object
 16  Country                61996 non-null  object
 17  Timezone               61996 non-null  object
 18  Airport_Code           61991 non-null  object
 19  Weather_Timestamp      61773 non-null  object
 20  Temperature(F)         61589 non-null  float64
 21  Wind_Chill(F)          45839 non-null  float64
 22  Humidity(%)            61491 non-null  float64
 23  Pressure(in)           61675 non-null  float64
 24  Visibility(mi)         59269 non-null  float64
 25  Wind_Direction         61673 non-null  object
 26  Wind_Speed(mph)        58632 non-null  float64
 27  Precipitation(in)      40353 non-null  float64
 28  Weather_Condition      59298 non-null  object
 29  Amenity                61996 non-null  bool
 30  Bump                   61996 non-null  bool
 31  Crossing               61996 non-null  bool
 32  Give_Way               61996 non-null  bool
 33  Junction               61996 non-null  bool
```

```
 34  No_Exit                 61996 non-null  bool
 35  Railway                 61996 non-null  bool
 36  Roundabout              61996 non-null  bool
 37  Station                 61996 non-null  bool
 38  Stop                    61996 non-null  bool
 39  Traffic_Calming         61996 non-null  bool
 40  Traffic_Signal          61996 non-null  bool
 41  Turning_Loop            61996 non-null  bool
 42  Sunrise_Sunset          61992 non-null  object
 43  Civil_Twilight          61992 non-null  object
 44  Nautical_Twilight       61992 non-null  object
 45  Astronomical_Twilight   61992 non-null  object
dtypes: bool(13), float64(12), int64(1), object(20)
memory usage: 16.4+ MB
```

[4]:
```python
#print number and percentage of null entries per variable
print('Null values per variable')
for column in accident_data.columns:
    print('{}: {} ({}%)'.format(column,pd.isnull(accident_data[column]).
 ↪sum(),(pd.isnull(accident_data[column]).sum()/len(accident_data))*100))
```

```
Null values per variable
ID: 0 (0.0%)
Source: 0 (0.0%)
Severity: 0 (0.0%)
Start_Time: 0 (0.0%)
End_Time: 0 (0.0%)
Start_Lat: 0 (0.0%)
Start_Lng: 0 (0.0%)
End_Lat: 54025 (87.14271888508937%)
End_Lng: 54025 (87.14271888508937%)
Distance(mi): 0 (0.0%)
Description: 0 (0.0%)
Street: 46 (0.07419833537647 59%)
City: 0 (0.0%)
County: 0 (0.0%)
State: 0 (0.0%)
Zipcode: 0 (0.0%)
Country: 0 (0.0%)
Timezone: 0 (0.0%)
Airport_Code: 5 (0.008065036453964771%)
Weather_Timestamp: 223 (0.3597006258468288%)
Temperature(F): 407 (0.6564939673527325%)
Wind_Chill(F): 16157 (26.061358797341764%)
Humidity(%): 505 (0.814568681850442%)
Pressure(in): 321 (0.5177753403445383%)
Visibility(mi): 2727 (4.398670881992387%)
Wind_Direction: 323 (0.5210013549261243%)
```

```
Wind_Speed(mph): 3364 (5.426156526227499%)
Precipitation(in): 21643 (34.91031679463191%)
Weather_Condition: 2698 (4.351893670559391%)
Amenity: 0 (0.0%)
Bump: 0 (0.0%)
Crossing: 0 (0.0%)
Give_Way: 0 (0.0%)
Junction: 0 (0.0%)
No_Exit: 0 (0.0%)
Railway: 0 (0.0%)
Roundabout: 0 (0.0%)
Station: 0 (0.0%)
Stop: 0 (0.0%)
Traffic_Calming: 0 (0.0%)
Traffic_Signal: 0 (0.0%)
Turning_Loop: 0 (0.0%)
Sunrise_Sunset: 4 (0.006452029163171818%)
Civil_Twilight: 4 (0.006452029163171818%)
Nautical_Twilight: 4 (0.006452029163171818%)
Astronomical_Twilight: 4 (0.006452029163171818%)
```

[5]:
```
#look at distribution of data
accident_data.describe()
```

[5]:
```
           Severity     Start_Lat     Start_Lng      End_Lat      End_Lng  \
count  61996.000000  61996.000000  61996.000000  7971.000000  7971.000000
mean       2.293842     42.336970    -71.204913    42.299983   -71.286296
std        0.523010      0.227612      0.350009     0.244489     0.454490
min        1.000000     41.274700    -73.476868    41.442540   -73.477854
25%        2.000000     42.225157    -71.262665    42.178960   -71.344475
50%        2.000000     42.347019    -71.120621    42.318780   -71.133590
75%        3.000000     42.501911    -71.053139    42.467335   -71.052010
max        4.000000     42.877491    -69.957573    42.876040   -69.984614

       Distance(mi)  Temperature(F)  Wind_Chill(F)   Humidity(%)  \
count  61996.000000    61589.000000   45839.000000  61491.000000
mean       0.244122       52.583681      45.853027     67.213950
std        1.299053       19.167085      22.521689     20.612705
min        0.000000      -13.000000     -26.300000      7.000000
25%        0.000000       37.000000      28.500000     51.000000
50%        0.000000       53.000000      43.000000     69.000000
75%        0.000000       68.000000      65.000000     86.000000
max       79.946000       98.100000      98.000000    100.000000

       Pressure(in)  Visibility(mi)  Wind_Speed(mph)  Precipitation(in)
count  61675.000000    59269.000000     58632.000000       40353.000000
mean      29.930176        8.752235         9.175300           0.010531
```

```
std        0.316275      2.795481       5.474319          0.049839
min       27.790000      0.000000       0.000000          0.000000
25%       29.780000     10.000000       5.800000          0.000000
50%       29.950000     10.000000       8.100000          0.000000
75%       30.120000     10.000000      12.700000          0.000000
max       30.890000     10.500000     132.000000          2.820000
```

[6]:
```
# Get the number of rows and columns
num_rows, num_columns = accident_data.shape

print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_columns}")
```

```
Number of rows: 61996
Number of columns: 46
```

[7]:
```
#look at formatting of entries
accident_data.head()
```

[7]:
```
         ID   Source  Severity           Start_Time             End_Time  \
0  A-194264  Source2         2  2016-11-30 15:37:19  2016-11-30 17:08:21
1  A-194268  Source2         2  2016-11-30 16:14:24  2016-11-30 17:28:48
2  A-194269  Source2         3  2016-11-30 16:02:41  2016-11-30 17:25:00
3  A-194270  Source2         4  2016-11-30 14:12:49  2016-11-30 17:25:00
4  A-194271  Source2         3  2016-11-30 16:00:47  2016-11-30 17:15:31

   Start_Lat  Start_Lng  End_Lat  End_Lng  Distance(mi)  … Roundabout  \
0  42.144863 -72.599976      NaN      NaN          0.00  …      False
1  42.304436 -71.325317      NaN      NaN          0.00  …      False
2  42.428036 -71.258476      NaN      NaN          0.01  …      False
3  42.495930 -71.178238      NaN      NaN          0.01  …      False
4  42.525875 -70.972115      NaN      NaN          0.01  …      False

  Station   Stop Traffic_Calming Traffic_Signal Turning_Loop Sunrise_Sunset  \
0   False  False           False          False        False            Day
1   False  False           False           True        False          Night
2   False  False           False          False        False            Day
3   False  False           False          False        False            Day
4   False  False           False          False        False            Day

  Civil_Twilight Nautical_Twilight Astronomical_Twilight
0            Day               Day                   Day
1            Day               Day                   Day
2            Day               Day                   Day
3            Day               Day                   Day
4            Day               Day                   Day

[5 rows x 46 columns]
```

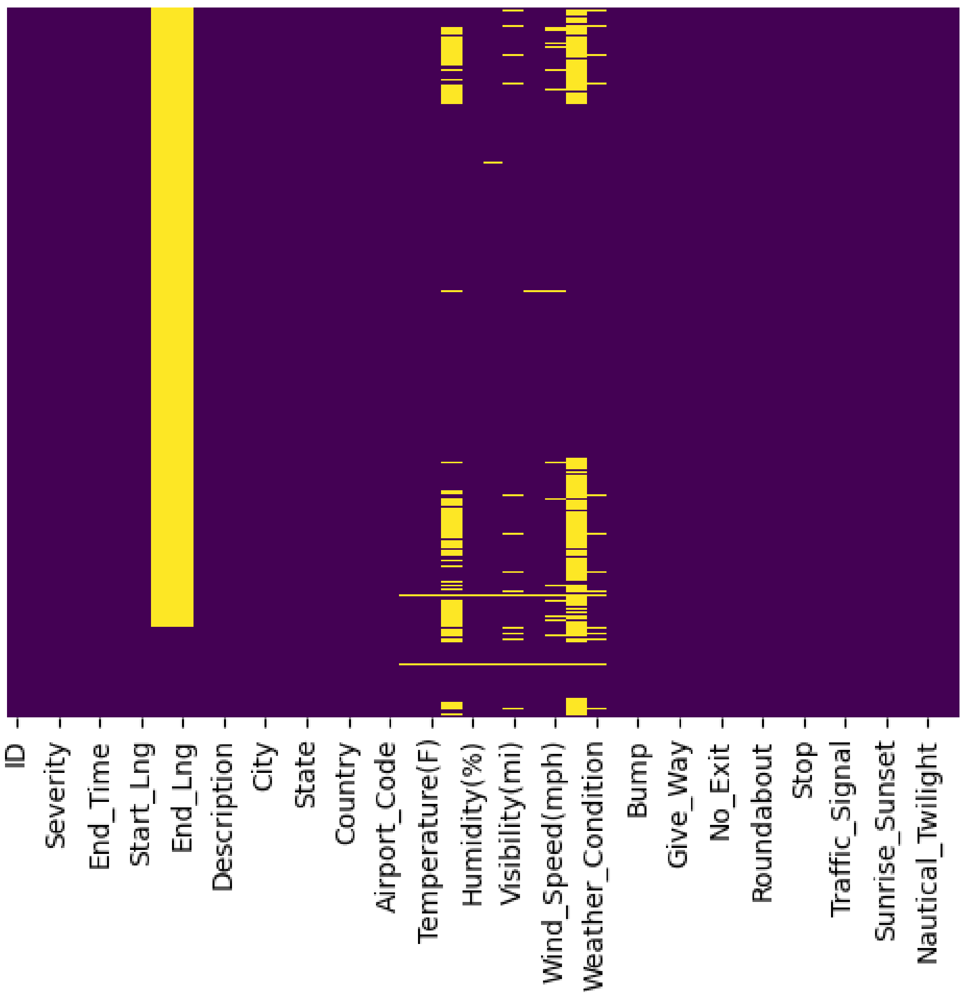
[8]:
```
#looking to see ID format towards end
accident_data.tail()
```

[8]:
```
              ID   Source  Severity           Start_Time             End_Time  \
61991  A-7776267  Source1         2  2019-08-21 18:01:55  2019-08-21 18:31:30
61992  A-7776802  Source1         2  2019-08-22 08:41:32  2019-08-22 09:11:10
61993  A-7777343  Source1         2  2019-08-23 21:40:04  2019-08-23 22:09:12
61994  A-7777349  Source1         2  2019-08-23 16:22:17  2019-08-23 16:52:10
61995  A-7777359  Source1         2  2019-08-23 19:12:21  2019-08-23 19:41:38

       Start_Lat  Start_Lng    End_Lat    End_Lng  Distance(mi)  …  \
61991  42.445630 -71.256440  42.439820 -71.258740         0.418  …
61992  42.383140 -71.076750  42.378460 -71.075840         0.327  …
61993  42.566199 -70.922008  42.567773 -70.919635         0.163  …
61994  42.097100 -71.058500  42.090840 -71.060250         0.442  …
61995  42.456159 -71.751316  42.460374 -71.742290         0.545  …

      Roundabout Station   Stop Traffic_Calming Traffic_Signal Turning_Loop  \
61991      False   False  False           False          False        False
61992      False   False  False           False          False        False
61993      False   False  False           False          False        False
61994      False   False  False           False          False        False
61995      False   False  False           False          False        False

      Sunrise_Sunset Civil_Twilight Nautical_Twilight Astronomical_Twilight
61991            Day            Day               Day                   Day
61992            Day            Day               Day                   Day
61993          Night          Night             Night                 Night
61994            Day            Day               Day                   Day
61995            Day            Day               Day                   Day

[5 rows x 46 columns]
```

3 Week 3 Advanced EDA and Data split

[9]:
```
# Deal with all the missing values

sns.heatmap(accident_data.isnull(),yticklabels=False,cbar=False,cmap='viridis')
# plotting a heatmap of missing values in columns
```

[9]:
```
<Axes: >
```



[10]:
```
# Filling in the missing values in three of the columns related to weather␣
↪condition

# accident_data["Wind_Chill(F)"] = accident_data['Wind_Chill(F)'].
↪fillna(accident_data['Wind_Chill(F)'].mean())
# accident_data["Precipitation(in)"] = accident_data['Precipitation(in)'].
↪fillna(accident_data['Precipitation(in)'].mean())
# accident_data["Wind_Speed(mph)"] = accident_data['Wind_Speed(mph)'].
↪fillna(accident_data['Wind_Speed(mph)'].mean())
```

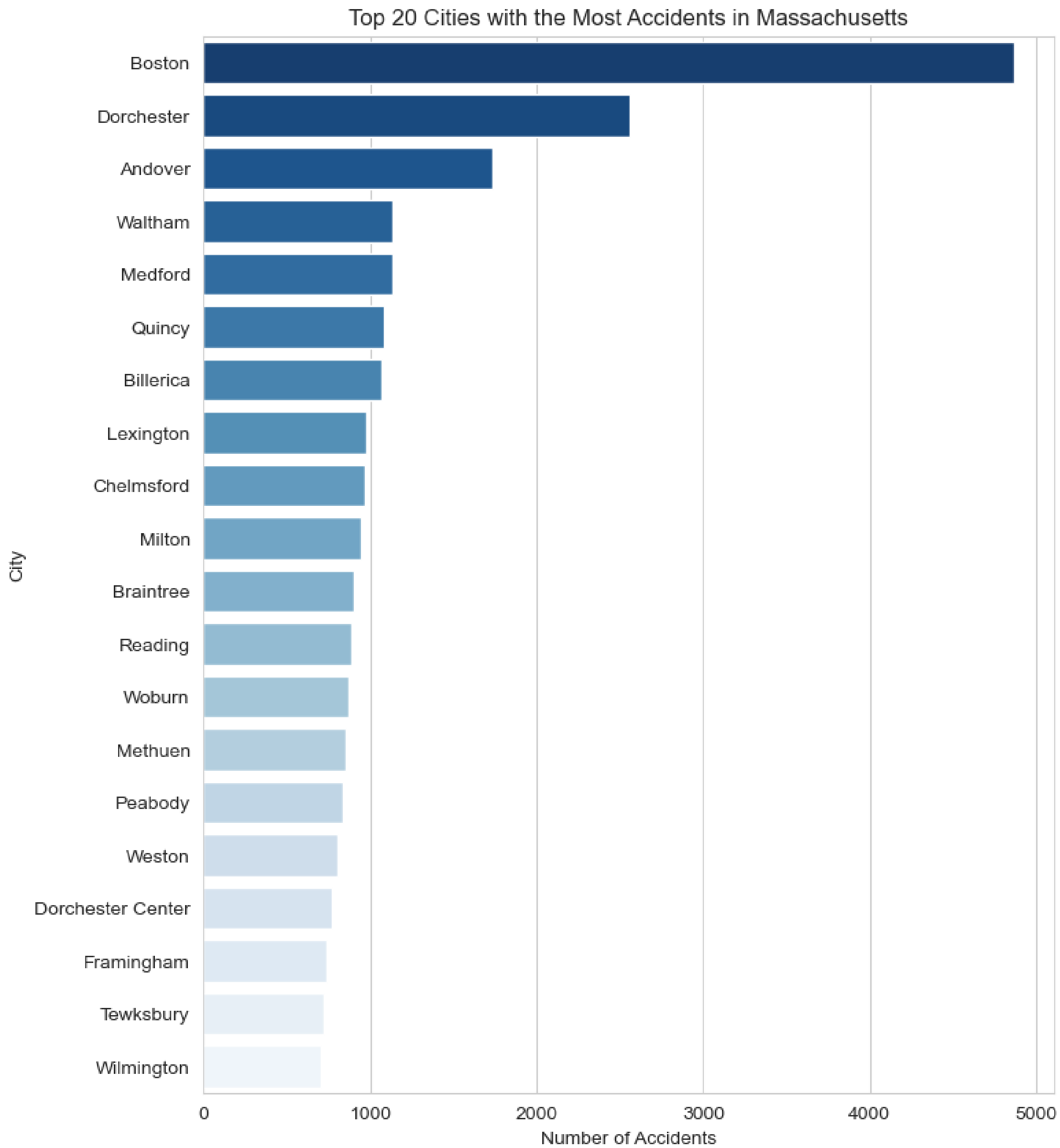
[11]:
```
# Which City has the maximum number of accidents?

city_wise_counts = accident_data.groupby('City')['ID'].count().reset_index()
city_wise_counts = city_wise_counts.sort_values(by = "ID",ascending=False)
```

```
max_accident_city = city_wise_counts.iloc[0]  # Get the top city
print(f"The city with the highest number of accidents in Massachusetts is␣
 ↪{max_accident_city['City']} with {max_accident_city['ID']} accidents.")
```

```
The city with the highest number of accidents in Massachusetts is Boston with
4866 accidents.
```

[12]:
```
# Get top 20 cities
top_20_cities = city_wise_counts.head(20)

# Set Seaborn style
sns.set_style("whitegrid")

# Create the figure
f, ax = plt.subplots(figsize=(8, 10))

# Create the bar plot
sns.barplot(y="City", x="ID", data=top_20_cities, ax=ax, palette="Blues_r")

# Add title and labels
ax.set_title("Top 20 Cities with the Most Accidents in Massachusetts")
ax.set_xlabel("Number of Accidents")
ax.set_ylabel("City")

# Show the plot
plt.show()
```

```
/var/folders/x6/yvl9g72j20q30dzkb5yqg1z80000gn/T/ipykernel_95141/2744639619.py:1
1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(y="City", x="ID", data=top_20_cities, ax=ax, palette="Blues_r")
```



[13]:
```python
# Group accident data by City and State, getting their latitude and longitude
city_wise_counts = accident_data.groupby(["City", "State"])[["Start_Lat",␣
↪"Start_Lng"]].first().reset_index()
city_wise_counts["Accident_Count"] = accident_data.groupby(["City",␣
↪"State"])["ID"].count().values

# Create a scatter geo plot for 50 city-wise accidents
top_cities = city_wise_counts.nlargest(50, "Accident_Count")  # Show only top␣
↪50 cities
fig = px.scatter_geo(
```

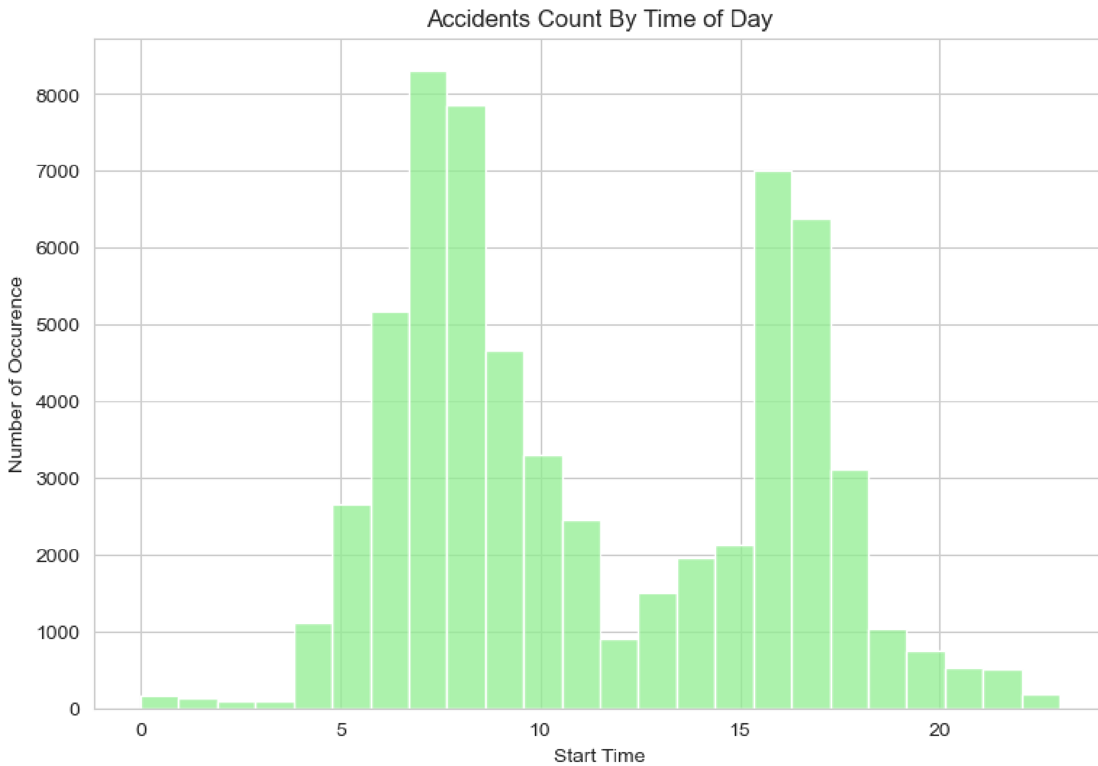
```
    top_cities,
    lat="Start_Lat",
    lon="Start_Lng",
    size="Accident_Count",
    hover_name="City",
    hover_data={"State": True, "Accident_Count": True},
    color="Accident_Count",
    color_continuous_scale="spectral_r",
    title="Top 50 Cities with the Most Accidents in the Massachusetts",
    scope="usa"
)
fig.show()
```

[14]:

```
# Accidents based on Time

accident_data["Start_Time"] = pd.to_datetime(accident_data["Start_Time"],␣
 ↪format="mixed", errors="coerce")

# To find the accidents by time of day

fig, ax = plt.subplots(figsize=(9,6))
sns.histplot(accident_data.Start_Time.dt.
 ↪hour,bins=24,kde=False,color='lightgreen')

plt.xlabel("Start Time")
plt.ylabel("Number of Occurence")
plt.title('Accidents Count By Time of Day')
```

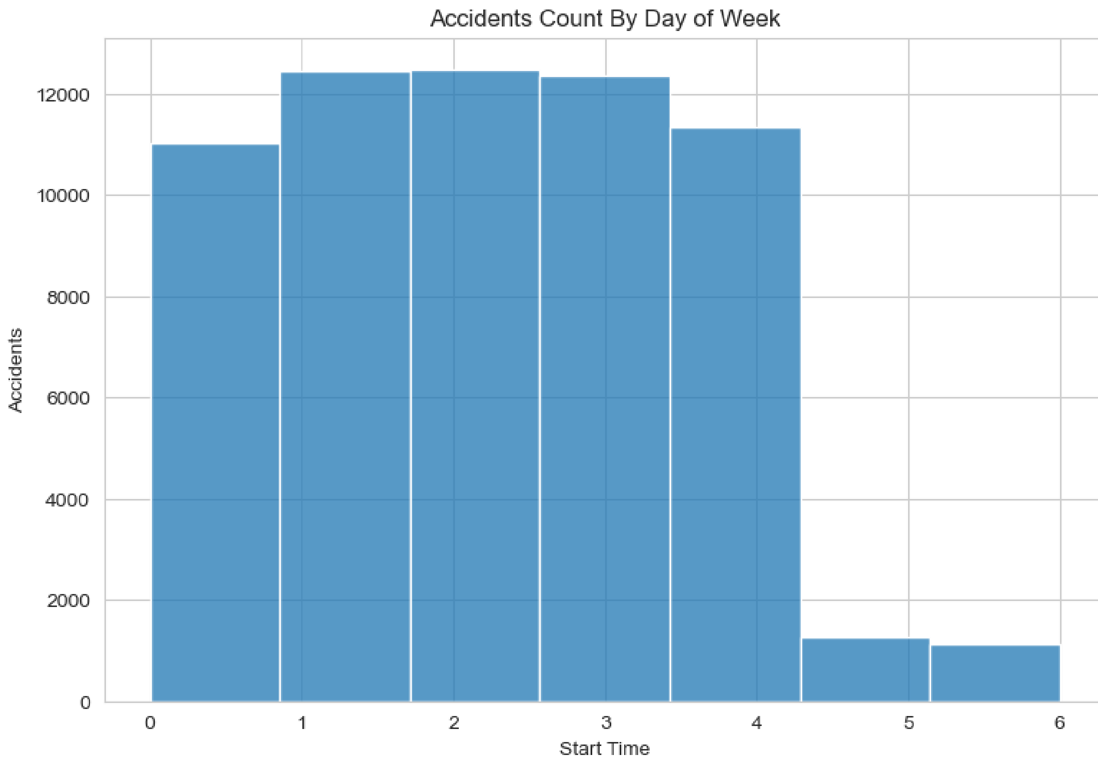
[14]:

```
Text(0.5, 1.0, 'Accidents Count By Time of Day')
```



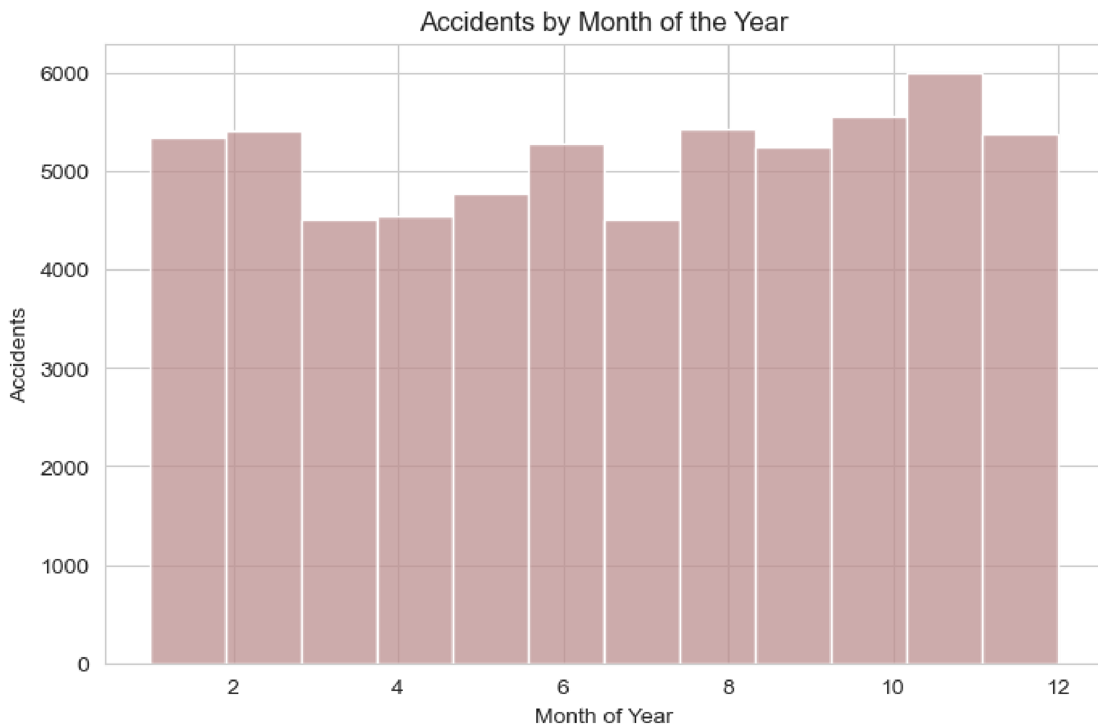
[15]:
```
# To find the accidents by Day of the week

fig, ax = plt.subplots(figsize=(9,6))
sns.histplot(accident_data.Start_Time.dt.dayofweek,bins=7,kde=False)

plt.xlabel("Start Time")
plt.ylabel("Accidents")
plt.title('Accidents Count By Day of Week')
```

[15]:
```
Text(0.5, 1.0, 'Accidents Count By Day of Week')
```



[16]:
```
# To find the accidents by the month of the year

fig, ax = plt.subplots(figsize=(8,5))
sns.histplot(accident_data['Start_Time'].dt.month, bins = 12,color='rosybrown')

plt.xlabel("Month of Year")
plt.ylabel("Accidents")
plt.title('Accidents by Month of the Year')
```

[16]:
```
Text(0.5, 1.0, 'Accidents by Month of the Year')
```



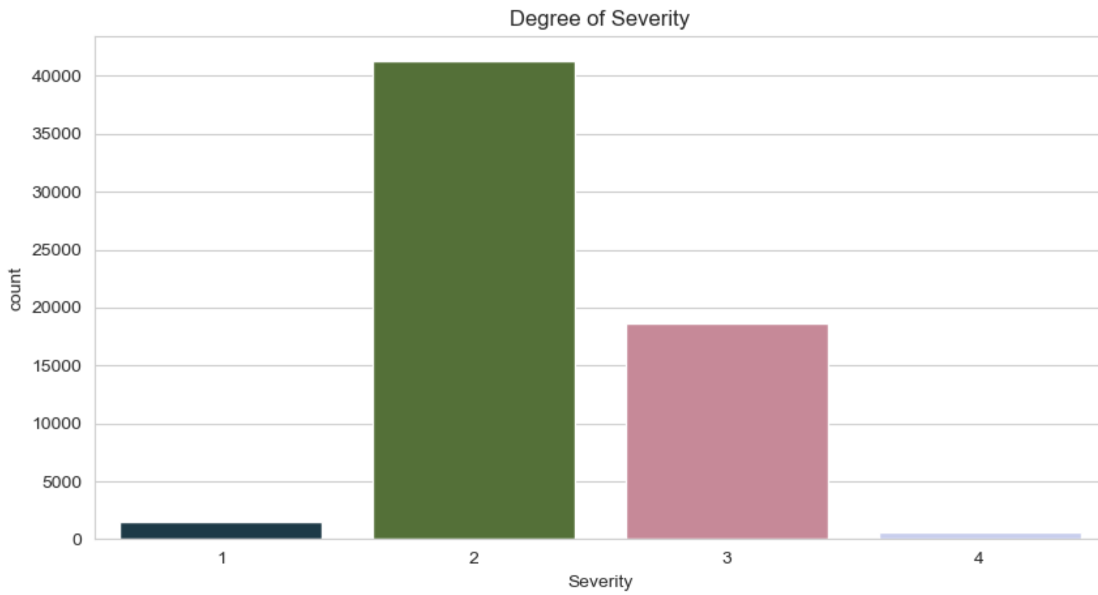
[17]:
```
# Accidents based on Severity and Weather Conditions

df_severity = accident_data.groupby('Severity')['ID'].count()
df_severity
```

[17]:
```
Severity
1     1500
2    41326
3    18623
4      547
Name: ID, dtype: int64
```

[18]:
```
fig, ax = plt.subplots(figsize = (10,5))
sev = sns.countplot(x="Severity", data=accident_data, palette = "cubehelix")
sev.set_title("Degree of Severity")
```

```
/var/folders/x6/yvl9g72j20q30dzkb5yqg1z80000gn/T/ipykernel_95141/2458966032.py:2
: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

[18]: Text(0.5, 1.0, 'Degree of Severity')

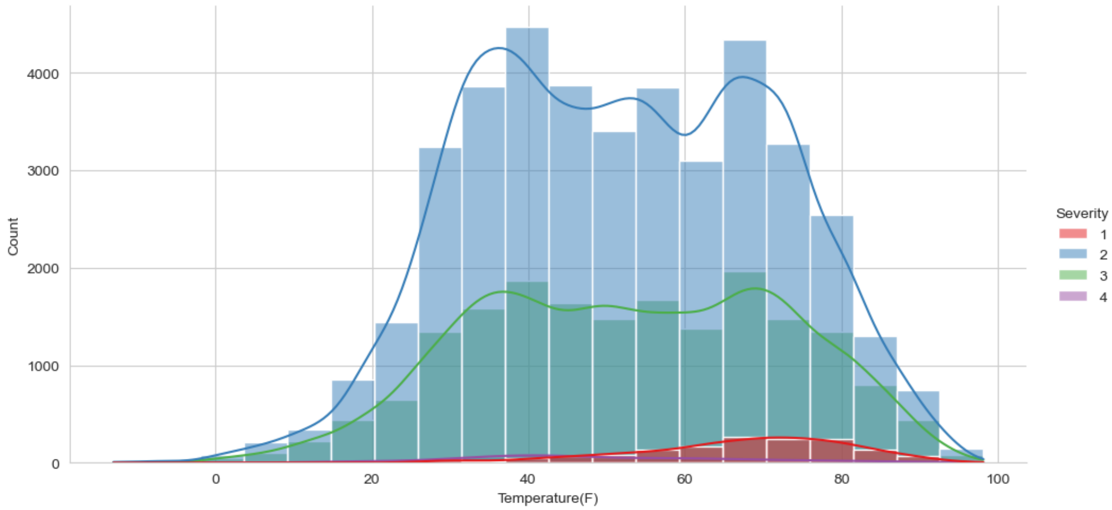


[19]:
```
weather = accident_data.iloc[:, 20:30]
weather['Severity'] = accident_data['Severity']
weather.head()
```

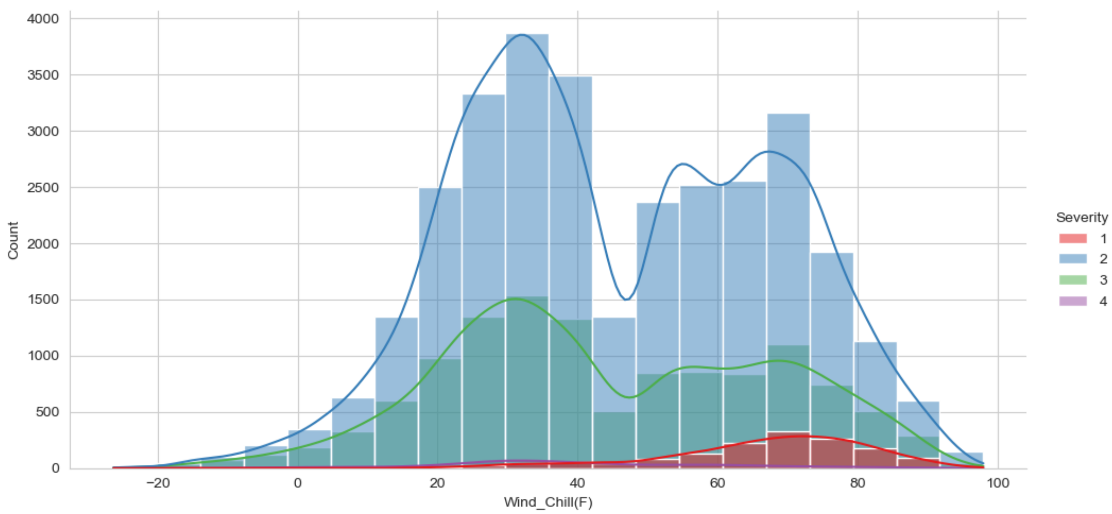
[19]:
```
   Temperature(F)  Wind_Chill(F)  Humidity(%)  Pressure(in)  Visibility(mi)  \
0            48.2            NaN        100.0         29.87             3.0
1            48.0            NaN         89.0         29.96             5.0
2            46.9            NaN         86.0         30.01             5.0
3            46.0           41.9         89.0         30.01             3.0
4            46.0           41.9        100.0         29.97             6.0

  Wind_Direction  Wind_Speed(mph)  Precipitation(in) Weather_Condition  \
0       Variable              3.5                NaN        Light Rain
1            ENE              5.8               0.05              Rain
2            ENE              6.9               0.08              Rain
3           East              8.1               0.01        Light Rain
4            NNE              8.1               0.07        Light Rain

   Amenity  Severity
0    False         2
1    False         2
2    False         3
```

```
3     False           4
4     False           3
```

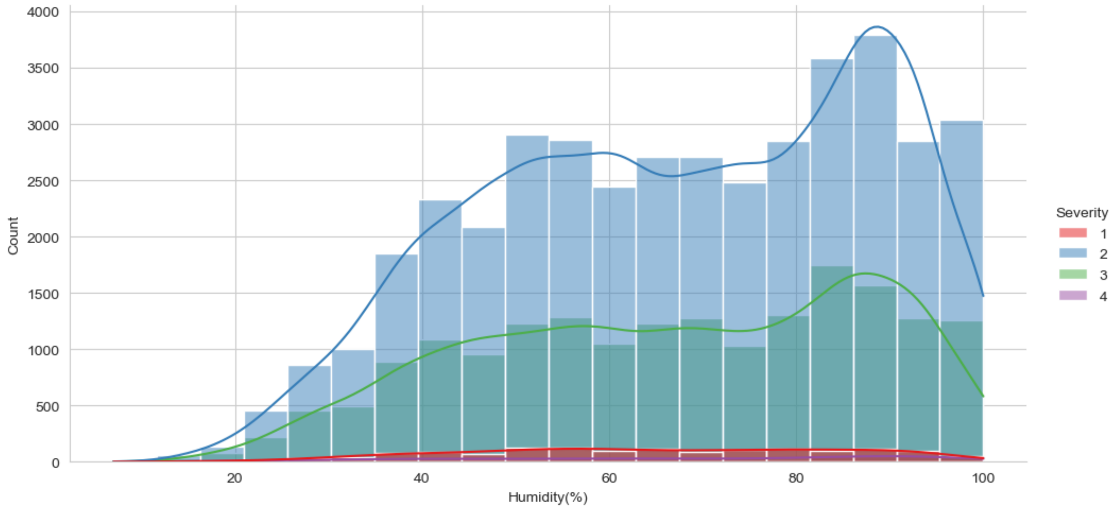
[20]:
```python
sns.displot(weather, x="Temperature(F)", hue="Severity", palette="Set1",␣
 ↪height=5, aspect=2,bins=20,kde=True);
```



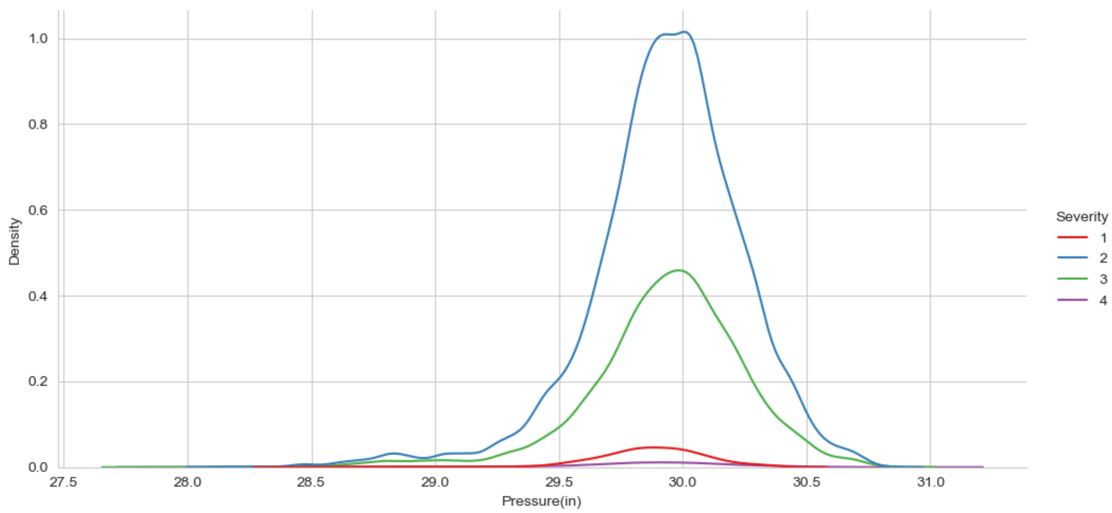
[21]:
```python
sns.displot(weather, x="Wind_Chill(F)", hue="Severity",palette="Set1",␣
 ↪height=5, aspect=2,bins=20,kde=True);
```



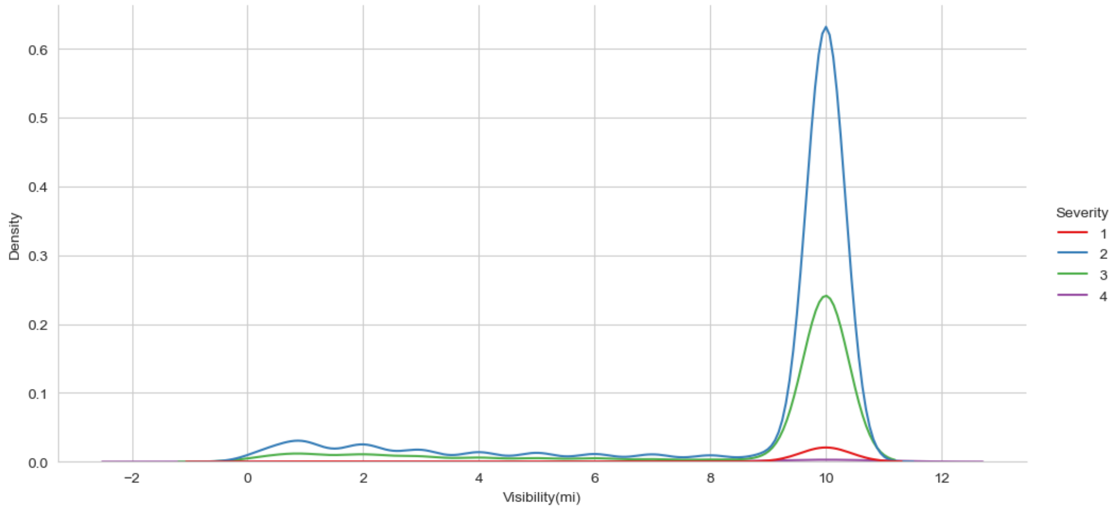
[22]:
```python
sns.displot(weather, x="Humidity(%)", hue="Severity", palette="Set1", height=5,␣
 ↪aspect=2,bins=20,kde=True);
```



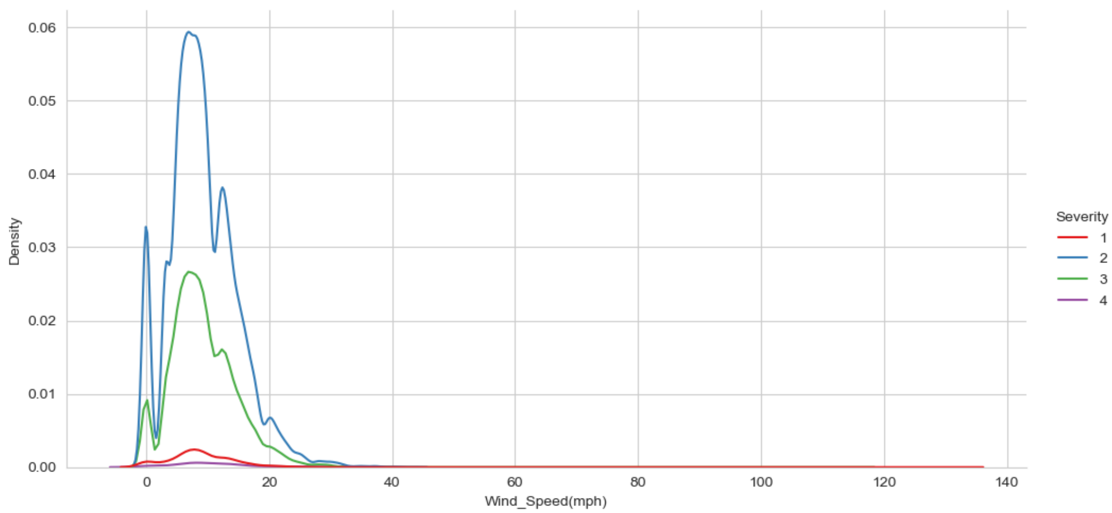
[23]:
```
sns.displot(weather, x="Pressure(in)", hue="Severity", palette="Set1",␣
    ↪height=5, aspect=2,kind='kde');
```



[24]:
```
sns.displot(weather, x="Visibility(mi)", hue="Severity", palette="Set1",␣
    ↪height=5, aspect=2,kind='kde');
```



[25]:
```
sns.displot(weather, x="Wind_Speed(mph)", hue="Severity", palette="Set1",␣
↪height=5, aspect=2,kind='kde');
```



4 Week 4: Data processing & Feature engineering

[26]:
```
# Irrelevant columns
# ID is unique and meaningless for the dataset; Description: I don't do text␣
↪mining, and I will do text classification later, therefore It's useless;␣
↪Country: All the data is from MA ;Weather_Timestamp: The timestamp of␣
↪weather observation record. It's useless here.
irrelavant_columns = ['ID','Description','Country','State','Weather_Timestamp']
```

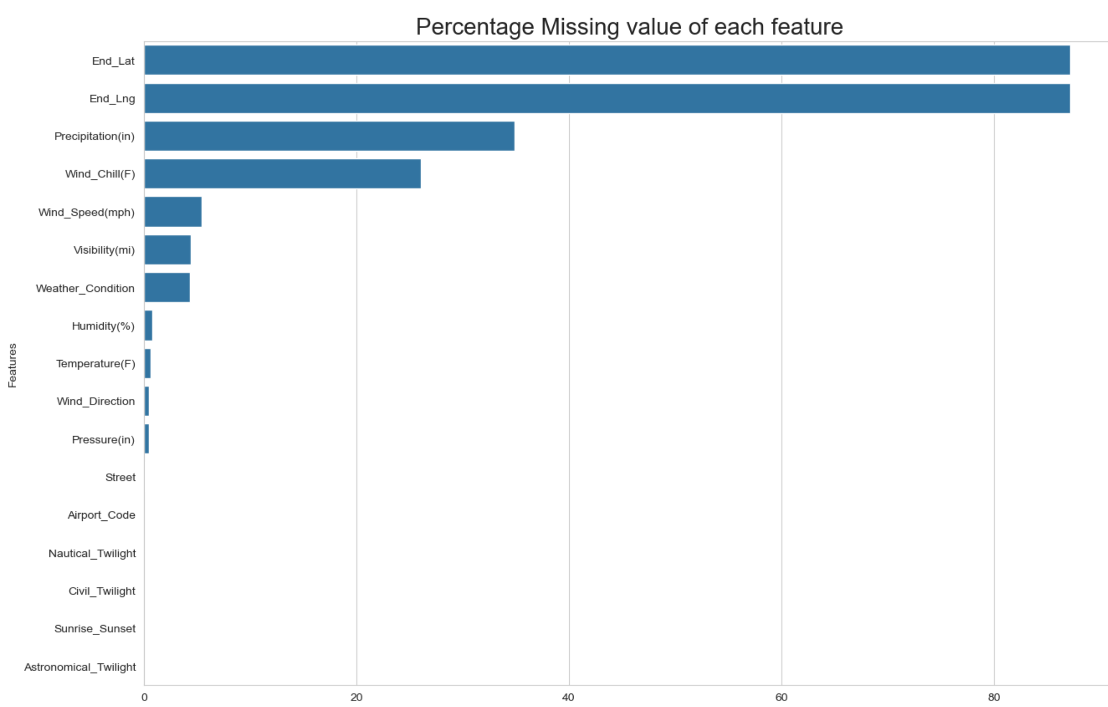
```
data_preprocessed_df = accident_data.drop(irrelavant_columns, axis=1)
```

Drop the column with Missing Value(>40%)

[27]:
```
# Replace the empty data with NaN
data_preprocessed_df.replace("", float("NaN"), inplace=True)
data_preprocessed_df.replace(" ", float("NaN"), inplace=True)

# Count missing value(NaN, na, null, None) of each columns, Then transform the␣
↪result to a pandas dataframe.
count_missing_value = data_preprocessed_df.isna().sum() / data_preprocessed_df.
↪shape[0] * 100
count_missing_value_df = pd.DataFrame(count_missing_value.
↪sort_values(ascending=False), columns=['Missing%'])
```

[28]:
```
# Visualize the percentage(>0) of Missing value in each column.
missing_value_df = count_missing_value_df[count_missing_value_df['Missing%'] >␣
↪0]

plt.figure(figsize=(15, 10)) # Set the figure size
missing_value_graph = sns.barplot(y = missing_value_df.index, x = "Missing%",␣
↪data=missing_value_df, orient="h")
missing_value_graph.set_title("Percentage Missing value of each feature",␣
↪fontsize = 20)
missing_value_graph.set_ylabel("Features")
```

[28]:
```
Text(0, 0.5, 'Features')
```



```
[29]: ## Drop the column with Missing value(>40%)
missing_value_40_df = count_missing_value_df[count_missing_value_df['Missing%']␣
↪> 40]
data_preprocessed_df.drop(missing_value_40_df.index, axis=1, inplace=True)
missing_value_40_df
```

```
[29]:          Missing%
End_Lat  87.142719
End_Lng  87.142719
```

Data type correcting

```
[30]: # Convert Time to datetime64[ns]
data_preprocessed_df['Start_Time'] = pd.
↪to_datetime(data_preprocessed_df['Start_Time'])
data_preprocessed_df['End_Time'] = pd.
↪to_datetime(data_preprocessed_df['End_Time'], errors='coerce')
```

```
[31]: # Display all the missing value
missing_value_df
```

```
[31]:                         Missing%
End_Lat                87.142719
End_Lng                87.142719
Precipitation(in)      34.910317
Wind_Chill(F)          26.061359
Wind_Speed(mph)         5.426157
Visibility(mi)          4.398671
Weather_Condition       4.351894
Humidity(%)             0.814569
Temperature(F)          0.656494
Wind_Direction          0.521001
Pressure(in)            0.517775
Street                  0.074198
Airport_Code            0.008065
Nautical_Twilight       0.006452
Civil_Twilight          0.006452
Sunrise_Sunset          0.006452
Astronomical_Twilight   0.006452
```

```
[32]: # Categorize the missing value to numerical and categorical for imputation␣
↪purpose
numerical_missing = ['Wind_Speed(mph)', 'Visibility(mi)','Humidity(%)',␣
↪'Temperature(F)', 'Pressure(in)']
```

```
categorical_missing = ['Weather_Condition','Wind_Direction', 'Sunrise_Sunset',␣
↪'Civil_Twilight', 'Nautical_Twilight', 'Astronomical_Twilight']
```

Median imputation

[33]:
```
# Imputation by corresponding class Median value
data_preprocessed_median_df = data_preprocessed_df.copy()

# For numerical columns
for column_name in numerical_missing:
    data_preprocessed_median_df[column_name] = data_preprocessed_median_df.
↪groupby('Severity')[column_name].transform(lambda x:x.fillna(x.median()))

# # For categorical columns(Majority value imputation)
for column_name in categorical_missing:
    data_preprocessed_median_df[column_name] = data_preprocessed_median_df.
↪groupby('Severity')[column_name].transform(lambda x:x.fillna(x.fillna(x.
↪mode().iloc[0])))

# Drop NaN and reset index
data_preprocessed_median_df.dropna(inplace=True)
```

Feature engineering

[34]:
```
# Choose relevant features

data_best_df = data_preprocessed_median_df
relevant_features = ['Severity', 'Start_Time', 'End_Time', 'Start_Lat',␣
↪'Start_Lng',
        'Temperature(F)', 'Humidity(%)', 'Pressure(in)', 'Visibility(mi)',
        'Wind_Direction', 'Wind_Speed(mph)', 'Weather_Condition', 'Amenity',
        'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway',
        'Roundabout', 'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal',
        'Turning_Loop', 'Sunrise_Sunset']
data_modelling_df = data_best_df[relevant_features].copy()

print(data_modelling_df.columns)
```

```
Index(['Severity', 'Start_Time', 'End_Time', 'Start_Lat', 'Start_Lng',
       'Temperature(F)', 'Humidity(%)', 'Pressure(in)', 'Visibility(mi)',
       'Wind_Direction', 'Wind_Speed(mph)', 'Weather_Condition', 'Amenity',
       'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway',
       'Roundabout', 'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal',
       'Turning_Loop', 'Sunrise_Sunset'],
      dtype='object')
```

[35]:
```
# Duration = End_Time - Start_Time; Create a new feature for modeling.
```

```
data_modelling_df['Duration'] = (data_modelling_df['End_Time'] -␣
 ↪data_modelling_df['Start_Time']).dt.total_seconds() / 3600
data_modelling_df.drop('End_Time', axis=1, inplace=True)
```

[36]:
```
# Transform Month/week/Hour to different features
data_modelling_df["Month"] = data_modelling_df["Start_Time"].dt.month
data_modelling_df["Week"] = data_modelling_df["Start_Time"].dt.dayofweek
data_modelling_df["Hour"] = data_modelling_df["Start_Time"].dt.hour
data_modelling_df.drop("Start_Time", axis=1, inplace=True)
```

One Hot Encoding

[37]:
```
# Select features that are suitable for One Hot Encoding
one_hot_features = ['Wind_Direction', 'Weather_Condition']

# Wind_Direction Categorizing
data_modelling_df.loc[data_modelling_df['Wind_Direction'].str.startswith('C'),␣
 ↪'Wind_Direction'] = 'C' #Calm
data_modelling_df.loc[data_modelling_df['Wind_Direction'].str.startswith('E'),␣
 ↪'Wind_Direction'] = 'E' #East, ESE, ENE
data_modelling_df.loc[data_modelling_df['Wind_Direction'].str.startswith('W'),␣
 ↪'Wind_Direction'] = 'W' #West, WSW, WNW
data_modelling_df.loc[data_modelling_df['Wind_Direction'].str.startswith('S'),␣
 ↪'Wind_Direction'] = 'S' #South, SSW, SSE
data_modelling_df.loc[data_modelling_df['Wind_Direction'].str.startswith('N'),␣
 ↪'Wind_Direction'] = 'N' #North, NNW, NNE
data_modelling_df.loc[data_modelling_df['Wind_Direction'].str.startswith('V'),␣
 ↪'Wind_Direction'] = 'V' #Variable
```

[38]:
```
# Weather_Condition Categorizing
# Fair, Cloudy, Clear, Overcast, Snow, Haze, Rain, Thunderstorm, Windy, Hail,␣
 ↪Thunder, Dust, Tornado
data_modelling_df['Weather_Fair'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Fair',␣
 ↪case=False, na = False), 1, 0)
data_modelling_df['Weather_Cloudy'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Cloudy',␣
 ↪case=False, na = False), 1, 0)
data_modelling_df['Weather_Clear'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Clear',␣
 ↪case=False, na = False), 1, 0)
data_modelling_df['Weather_Overcast'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Overcast',␣
 ↪case=False, na = False), 1, 0)
data_modelling_df['Weather_Snow'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.
 ↪contains('Snow|Wintry|Sleet', case=False, na = False), 1, 0)
```

```
data_modelling_df['Weather_Haze'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.
 ↪contains('Smoke|Fog|Mist|Haze', case=False, na = False), 1, 0)
data_modelling_df['Weather_Rain'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.
 ↪contains('Rain|Drizzle|Showers', case=False, na = False), 1, 0)
data_modelling_df['Weather_Thunderstorm'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.
 ↪contains('Thunderstorms|T-Storm', case=False, na = False), 1, 0)
data_modelling_df['Weather_Windy'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Windy|Squalls',␣
 ↪case=False, na = False), 1, 0)
data_modelling_df['Weather_Hail'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Hail|Ice␣
 ↪Pellets', case=False, na = False), 1, 0)
data_modelling_df['Weather_Thunder'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Thunder',␣
 ↪case=False, na = False), 1, 0)
data_modelling_df['Weather_Dust'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Dust',␣
 ↪case=False, na = False), 1, 0)
data_modelling_df['Weather_Tornado'] = np.
 ↪where(data_modelling_df['Weather_Condition'].str.contains('Tornado',␣
 ↪case=False, na = False), 1, 0)
```

[39]:
```
# Define the weather categories and keywords
weather_data = {
    "Weather": [
        "Fair", "Cloudy", "Clear", "Overcast", "Snow", "Haze", "Rain",
        "Thunderstorm", "Windy", "Hail", "Thunder", "Dust", "Tornado", "N/A"
    ],
    "Contain": [
        "'Fair / Windy'",
        "'Mostly Cloudy', 'Partly Cloudy', 'Scattered Clouds', 'Cloudy /␣
 ↪Windy', 'Partly Cloudy / Windy', 'Mostly Cloudy / Windy', 'Funnel Cloud'",
        "'Clear'",
        "'Overcast'",
```

```
      "'Light Snow', 'Wintry Mix', 'Heavy Snow', 'Snow', 'Light Snow / Windy', 'Blowing Snow', 'Snow / Windy', 'Snow and Sleet', 'Blowing Snow / Windy', 'Sleet', 'Light Snow and Sleet', 'Light Snow with Thunder', 'Light Snow Showers', 'Heavy Snow with Thunder','Heavy Snow / Windy', 'Light Sleet', 'Heavy Sleet', 'Snow and Sleet / Windy', 'Thunderstorms and Snow', 'Light Thunderstorms and Snow', 'Heavy Blowing Snow','Light Sleet / Windy', 'Sleet / Windy', 'Snow Showers', 'Light Blowing Snow', 'Light Snow Shower','Drifting Snow','Low Drifting Snow','Light Snow and Sleet / Windy', 'Snow Grains', 'Light Snow Grains', 'Rain and Sleet', 'Thunder / Wintry Mix', 'Thunder / Wintry Mix / Windy', 'Wintry Mix / Windy'",
      "'Smoke', 'Fog', 'Mist', 'Shallow Fog', 'Haze / Windy', 'Patches of Fog', 'Light Freezing Fog', 'Fog / Windy', 'Smoke / Windy', 'Partial Fog', 'Patches of Fog / Windy','Light Haze','Light Fog'",
      "'Light Rain', 'Rain', 'Light Drizzle', 'Light Rain Shower', 'Heavy Rain', 'Light Freezing Rain', 'Drizzle', 'Rain / Windy', 'Drizzle and Fog', 'Light Rain with Thunder', 'Light Rain / Windy', 'Heavy Drizzle', 'Heavy Rain / Windy', 'Showers in the Vicinity', 'Light Freezing Drizzle', 'Light Drizzle / Windy', 'Heavy Rain Shower', 'Rain Showers', 'Light Rain Showers', 'Rain Shower', 'Freezing Rain','Light Freezing Rain / Windy', 'Drizzle / Windy','Light Rain Shower / Windy', 'Freezing Drizzle', 'Heavy Freezing Rain','Heavy Rain Showers','Heavy Freezing Drizzle', 'Rain and Sleet', 'Freezing Rain / Windy'",
      "'Thunderstorms and Rain', 'Light Thunderstorms and Rain', 'Heavy Thunderstorms and Rain', 'T-Storm', 'Heavy T-Storm', 'Heavy T-Storm / Windy', 'T-Storm / Windy', 'Heavy Thunderstorms and Snow', 'Thunderstorms and Snow', 'Light Thunderstorms and Snow', 'Light Thunderstorm', 'Heavy Thunderstorms with Small Hail'",
      "'Fair / Windy','Cloudy / Windy','Partly Cloudy / Windy', 'Mostly Cloudy / Windy','Light Snow / Windy','Fog / Windy', 'Smoke / Windy','Rain / Windy','Light Rain / Windy','Heavy Rain / Windy','Light Drizzle / Windy', 'Blowing Dust / Windy', 'Heavy T-Storm / Windy', 'T-Storm / Windy', 'Squalls / Windy', 'Thunder / Windy', 'Blowing Snow / Windy', 'Squalls', 'Heavy Snow / Windy', 'Snow and Sleet / Windy', 'Light Freezing Rain / Windy', 'Patches of Fog / Windy', 'Light Rain Shower / Windy', 'Light Sleet / Windy', 'Sleet / Windy', 'Light Snow and Sleet / Windy', 'Widespread Dust / Windy', 'Thunder / Wintry Mix / Windy', 'Wintry Mix / Windy', 'Thunder and Hail / Windy', 'Freezing Rain / Windy'",
      "'Small Hail', 'Light Ice Pellets', 'Ice Pellets', 'Thunder and Hail','Light Hail','Heavy Ice Pellets', 'Hail', 'Heavy Thunderstorms with Small Hail', 'Thunder and Hail / Windy'",
      "'Thunder in the Vicinity', 'Thunder', 'Thunder / Windy', 'Light Snow with Thunder', 'Heavy Snow with Thunder','Thunder and Hail', 'Thunder / Wintry Mix', 'Thunder / Wintry Mix / Windy', 'Thunder and Hail / Windy'",
      "'Dust Whirls', 'Sand / Dust Whirlwinds', 'Sand / Dust Whirls Nearby', 'Blowing Sand', 'Blowing Dust / Windy', 'Widespread Dust', 'Blowing Dust', 'Widespread Dust / Windy'",
```

```python
        "'Tornado'",
        "'N/A Precipitation'"
    ],
    "Key words": [
        "'Fair'", "'Cloud'", "'Clear'", "'Overcast'", "'Snow', 'Wintry',␣
↪'Sleet'", "'Smoke', 'Fog', 'Mist', 'Haze'", "'Rain', 'Drizzle', 'Showers'",␣
↪"'Thunderstorms', 'T-Storm'", "'Windy', 'Squalls'", "'Hail', 'Ice Pellets'",␣
↪"'Thunder'", "'Dust'", "'Tornado'", "'N/A Precipitation'"
    ]
}

# Create DataFrame
weather_df = pd.DataFrame(weather_data)

# Display table
print(weather_df)
```

```
         Weather                                             Contain  \
0           Fair                                      'Fair / Windy'
1         Cloudy  'Mostly Cloudy', 'Partly Cloudy', 'Scattered C…
2          Clear                                             'Clear'
3       Overcast                                          'Overcast'
4           Snow  'Light Snow', 'Wintry Mix', 'Heavy Snow', 'Sno…
5           Haze  'Smoke', 'Fog', 'Mist', 'Shallow Fog', 'Haze /…
6           Rain  'Light Rain', 'Rain', 'Light Drizzle', 'Light …
7   Thunderstorm  'Thunderstorms and Rain', 'Light Thunderstorms…
8          Windy  'Fair / Windy','Cloudy / Windy','Partly Cloudy…
9           Hail  'Small Hail', 'Light Ice Pellets', 'Ice Pellet…
10       Thunder  'Thunder in the Vicinity', 'Thunder', 'Thunder…
11          Dust  'Dust Whirls', 'Sand / Dust Whirlwinds', 'Sand…
12       Tornado                                           'Tornado'
13           N/A                                 'N/A Precipitation'

                         Key words
0                           'Fair'
1                          'Cloud'
2                          'Clear'
3                       'Overcast'
4         'Snow', 'Wintry', 'Sleet'
5   'Smoke', 'Fog', 'Mist', 'Haze'
6     'Rain', 'Drizzle', 'Showers'
7       'Thunderstorms', 'T-Storm'
8               'Windy', 'Squalls'
9            'Hail', 'Ice Pellets'
10                       'Thunder'
11                          'Dust'
12                       'Tornado'
13             'N/A Precipitation'
```

[40]:
```
# Transform the one-hot features, then delete them
onehot_df = pd.get_dummies(data_modelling_df['Wind_Direction'], prefix='Wind')
data_modelling_df = pd.concat([data_modelling_df, onehot_df], axis=1)
data_modelling_df.drop(one_hot_features, axis=1, inplace=True)
```

[41]:
```
data_modelling_df.columns
```

[41]:
```
Index(['Severity', 'Start_Lat', 'Start_Lng', 'Temperature(F)', 'Humidity(%)',
       'Pressure(in)', 'Visibility(mi)', 'Wind_Speed(mph)', 'Amenity', 'Bump',
       'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout',
       'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop',
       'Sunrise_Sunset', 'Duration', 'Month', 'Week', 'Hour', 'Weather_Fair',
       'Weather_Cloudy', 'Weather_Clear', 'Weather_Overcast', 'Weather_Snow',
       'Weather_Haze', 'Weather_Rain', 'Weather_Thunderstorm', 'Weather_Windy',
       'Weather_Hail', 'Weather_Thunder', 'Weather_Dust', 'Weather_Tornado',
       'Wind_C', 'Wind_E', 'Wind_N', 'Wind_S', 'Wind_V', 'Wind_W'],
      dtype='object')
```

Label Encoding

[42]:
```
# Select features that are suitable for Label Encoding
label_encoding_features = ['Amenity','Bump', 'Crossing', 'Give_Way',␣
↪'Junction', 'No_Exit', 'Railway','Roundabout', 'Station', 'Stop',␣
↪'Traffic_Calming', 'Traffic_Signal','Turning_Loop', 'Sunrise_Sunset']

# Label Encoding
for feature in label_encoding_features:
    data_modelling_df[feature] = LabelEncoder().
↪fit_transform(data_modelling_df[feature])
```

[43]:
```
data_modelling_df
```

[43]:
```
       Severity  Start_Lat  Start_Lng  Temperature(F)  Humidity(%)  \
3             4  42.495930 -71.178238            46.0         89.0
4             3  42.525875 -70.972115            46.0        100.0
6             2  42.251457 -71.002701            44.1        100.0
7             2  42.619003 -71.125229            45.0         97.0
9             2  42.770470 -71.075432            46.0         96.0
…           …          …          …             …           …
61991         2  42.445630 -71.256440            80.0         81.0
61992         2  42.383140 -71.076750            76.0         85.0
61993         2  42.566199 -70.922008            63.0         70.0
61994         2  42.097100 -71.058500            79.0         42.0
61995         2  42.456159 -71.751316            70.0         63.0

       Pressure(in)  Visibility(mi)  Wind_Speed(mph)  Amenity  Bump  …  \
3             30.01             3.0              8.1        0     0  …
4             29.97             6.0              8.1        0     0  …
```

```
6             29.96            10.0             17.3        0      0  …
7             29.98             7.0              8.1        0      0  …
9             29.97             8.0              6.9        0      0  …
…             …            …             …          …    …   …
61991         29.67            10.0              7.0        0      0  …
61992         29.82             1.0              3.0        0      0  …
61993         29.89            10.0              6.0        0      0  …
61994         29.91            10.0              8.0        0      0  …
61995         29.62            10.0              5.0        0      0  …

       Weather_Hail  Weather_Thunder  Weather_Dust  Weather_Tornado  Wind_C  \
3                 0                0             0                0   False
4                 0                0             0                0   False
6                 0                0             0                0   False
7                 0                0             0                0   False
9                 0                0             0                0   False
…               …              …             …              …     …
61991             0                0             0                0   False
61992             0                0             0                0   False
61993             0                0             0                0   False
61994             0                0             0                0   False
61995             0                0             0                0   False

       Wind_E  Wind_N  Wind_S  Wind_V  Wind_W
3        True   False   False   False   False
4       False    True   False   False   False
6        True   False   False   False   False
7        True   False   False   False   False
9        True   False   False   False   False
…         …     …     …     …     …
61991   False   False    True   False   False
61992   False   False    True   False   False
61993   False    True   False   False   False
61994   False   False   False   False    True
61995   False    True   False   False   False

[37243 rows x 45 columns]
```

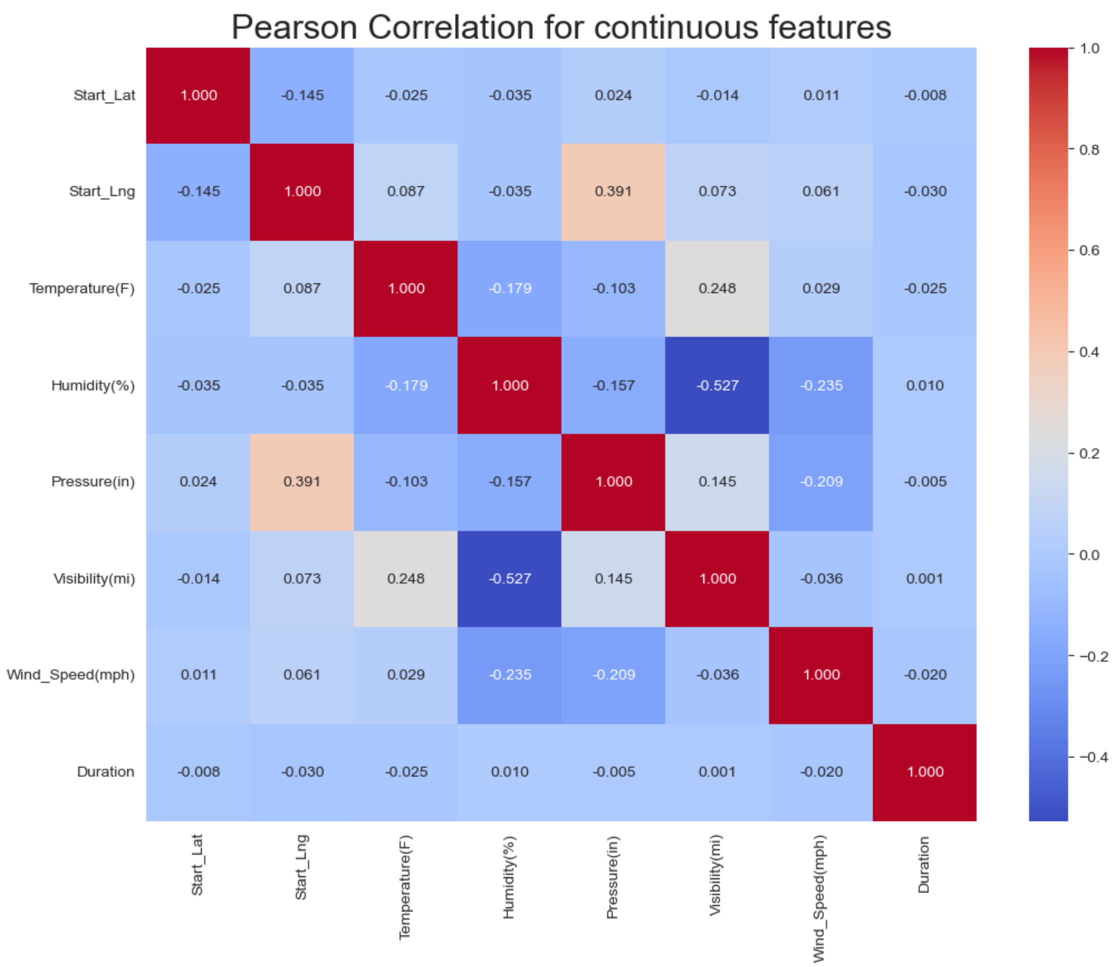
Correlation Analysis

[44]:
```python
def style_corr(v, props=''):
    return props if (v < -0.4 or v > 0.4) and v != 1 else None

continuous_feature = ['Start_Lat', 'Start_Lng', 'Temperature(F)','Humidity(%)',␣
 ↪'Pressure(in)', 'Visibility(mi)', 'Wind_Speed(mph)', 'Duration']
data_modelling_df[continuous_feature].corr().style.map(style_corr, props='color:
 ↪red;')
```

[44]: <pandas.io.formats.style.Styler at 0x34589e0f0>

[45]:
```
# Show the heatmap
plt.figure(figsize=(12,9))
sns.heatmap(data_modelling_df[continuous_feature].corr(), cmap="coolwarm",␣
 ↪annot = True, fmt='.3f').set_title('Pearson Correlation for continuous␣
 ↪features', fontsize=22)
```

[45]: Text(0.5, 1.0, 'Pearson Correlation for continuous features')



There are weak relationship between:

Pressure and Temperature

Pressure and Humidity;

Wind_Speed and Humidity;

Wind_Speed and Pressure;

Visibility and Humidity

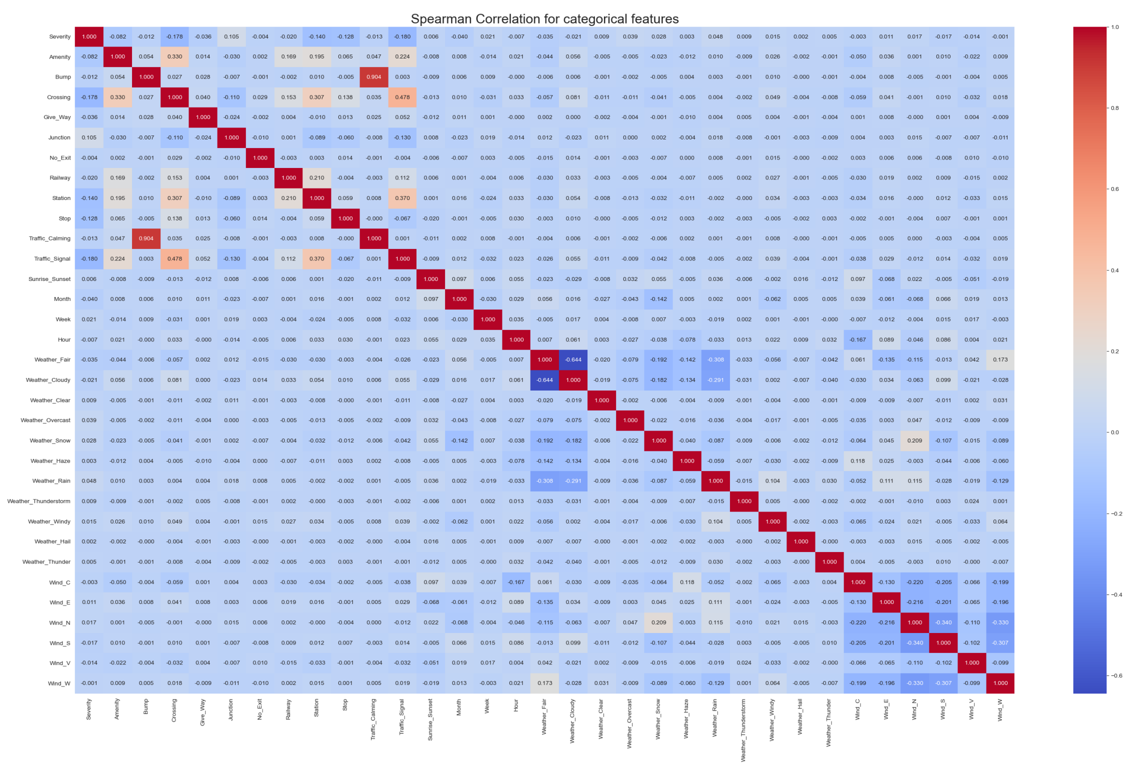
[46]:
```
# Find the data with all the same value and drop
unique_counts = data_modelling_df.drop(continuous_feature, axis=1).
↪astype("object").describe().loc['unique']
feature_all_same = list(unique_counts[unique_counts == 1].index)
data_modelling_df.drop(feature_all_same, axis=1, inplace=True)
```

[47]:
```
# Display the correlation table for categorical features
data_modelling_df.drop(continuous_feature, axis=1).corr(method='spearman').
↪style.map(style_corr, props='color:red;')
```

[47]: <pandas.io.formats.style.Styler at 0x3494264e0>

[48]:
```
# Show the heatmap
plt.figure(figsize=(35,20))
sns.heatmap(data_modelling_df.drop(continuous_feature, axis=1).
↪corr(method='spearman'), cmap="coolwarm", annot = True, fmt='.3f').
↪set_title('Spearman Correlation for categorical features', fontsize=22)
```

[48]: Text(0.5, 1.0, 'Spearman Correlation for categorical features')



We can find Strong relationship:

Bump and Traffic_Calming

moderate relationship:

Crossing and Traffic_Signal;

Crossing and Amenity;

Crossing and station;

station and Traffic_Signal

[49]:
```
data_modelling_df
```

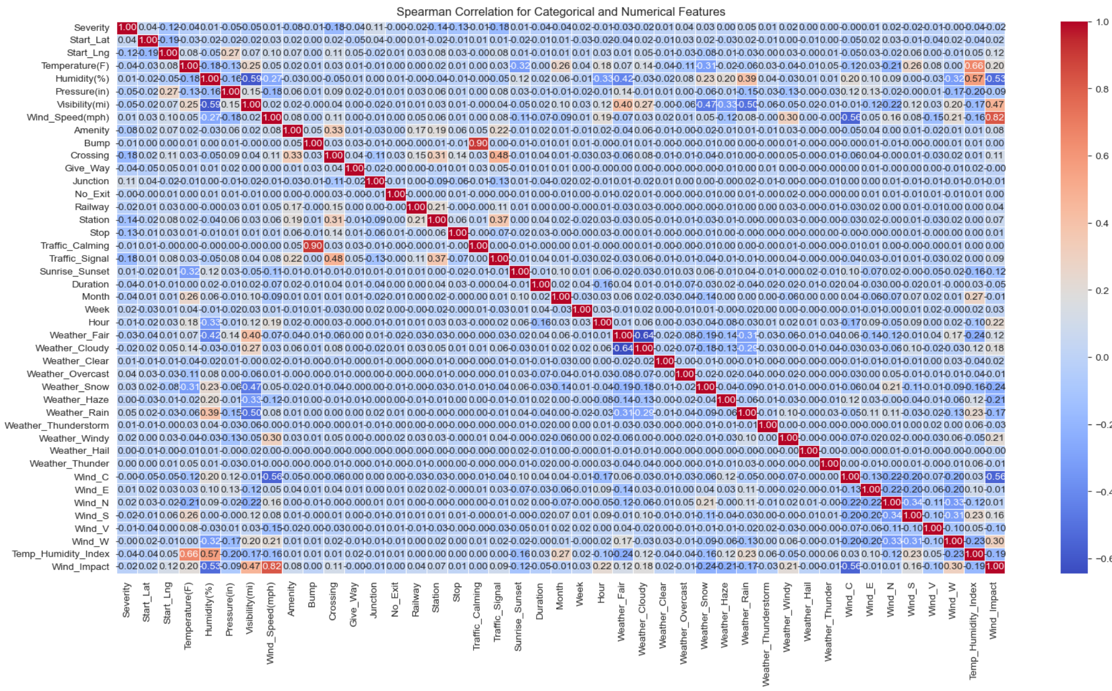
[49]:
```
       Severity  Start_Lat  Start_Lng  Temperature(F)  Humidity(%)  \
3             4  42.495930 -71.178238            46.0         89.0
4             3  42.525875 -70.972115            46.0        100.0
6             2  42.251457 -71.002701            44.1        100.0
7             2  42.619003 -71.125229            45.0         97.0
9             2  42.770470 -71.075432            46.0         96.0
…           …        …          …                …            …
61991         2  42.445630 -71.256440            80.0         81.0
61992         2  42.383140 -71.076750            76.0         85.0
61993         2  42.566199 -70.922008            63.0         70.0
61994         2  42.097100 -71.058500            79.0         42.0
61995         2  42.456159 -71.751316            70.0         63.0

       Pressure(in)  Visibility(mi)  Wind_Speed(mph)  Amenity  Bump  …  \
3             30.01             3.0              8.1        0     0  …
4             29.97             6.0              8.1        0     0  …
6             29.96            10.0             17.3        0     0  …
7             29.98             7.0              8.1        0     0  …
9             29.97             8.0              6.9        0     0  …
…               …             …                …          …    …  …  …
61991         29.67            10.0              7.0        0     0  …
61992         29.82             1.0              3.0        0     0  …
61993         29.89            10.0              6.0        0     0  …
61994         29.91            10.0              8.0        0     0  …
61995         29.62            10.0              5.0        0     0  …

       Weather_Thunderstorm  Weather_Windy  Weather_Hail  Weather_Thunder  \
3                         0              0             0                0
4                         0              0             0                0
6                         0              0             0                0
7                         0              0             0                0
9                         0              0             0                0
…                       …              …            …              …
61991                     0              0             0                0
61992                     0              0             0                0
61993                     0              0             0                0
61994                     0              0             0                0
```

```
61995                          0              0              0                 0

       Wind_C  Wind_E  Wind_N  Wind_S  Wind_V  Wind_W
3       False    True   False   False   False   False
4       False   False    True   False   False   False
6       False    True   False   False   False   False
7       False    True   False   False   False   False
9       False    True   False   False   False   False
…         …      …      …      …      …      …
61991   False   False   False    True   False   False
61992   False   False   False    True   False   False
61993   False   False    True   False   False   False
61994   False   False   False   False   False    True
61995   False   False    True   False   False   False

[37243 rows x 41 columns]
```

5 Week 5: Feature creation & Data augmentatoin & Dimensionality reduction

[50]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

numerical_features = ['Wind_Speed(mph)', 'Visibility(mi)', 'Humidity(%)',␣
 ↪'Temperature(F)', 'Pressure(in)']
categorical_features = ['Amenity', 'Bump', 'Crossing', 'Give_Way', 'Junction',␣
 ↪'No_Exit', 'Railway', 'Station', 'Stop', 'Traffic_Calming',␣
 ↪'Traffic_Signal', 'Sunrise_Sunset',
                        'Weather_Fair', 'Weather_Cloudy', 'Weather_Clear',␣
 ↪'Weather_Overcast', 'Weather_Snow', 'Weather_Haze', 'Weather_Rain',␣
 ↪'Weather_Thunderstorm',
                        'Weather_Windy', 'Weather_Hail', 'Weather_Thunder',␣
 ↪'Wind_C', 'Wind_E', 'Wind_N', 'Wind_S', 'Wind_V', 'Wind_W']

# Feature Engineering: Creating new features
# Creating a feature combining Temperature and Humidity to represent heat index␣
 ↪effect
data_modelling_df['Temp_Humidity_Index'] = data_modelling_df['Temperature(F)']␣
 ↪* data_modelling_df['Humidity(%)']

# Creating a feature that represents wind impact (Wind Speed * Visibility)
```

```python
data_modelling_df['Wind_Impact'] = data_modelling_df['Wind_Speed(mph)'] *␣
↪data_modelling_df['Visibility(mi)']

# Correlation Analysis
correlation_matrix = data_modelling_df.corr(method='spearman')
plt.figure(figsize=(20, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
↪linewidths=0.5)
plt.title("Spearman Correlation for Categorical and Numerical Features")
plt.show()

# Data Augmentation: Creating slight variations in numerical data by adding␣
↪small noise
augmented_data = data_modelling_df.copy()
for feature in numerical_features:
    augmented_data[feature] += np.random.normal(0, 0.01,␣
↪augmented_data[feature].shape)

# Concatenating original and augmented data
data_augmented = pd.concat([data_modelling_df, augmented_data],␣
↪ignore_index=True)

# Dimensionality Reduction using PCA
pca = PCA(n_components=0.95)  # Retain 95% variance
principal_components = pca.fit_transform(data_augmented[numerical_features])

# Converting PCA result to DataFrame
pca_df = pd.DataFrame(principal_components, columns=[f'PC{i+1}' for i in␣
↪range(principal_components.shape[1])])

# Merging PCA-transformed numerical data with categorical features
data_final = pd.concat([pca_df, data_augmented[categorical_features]], axis=1)

# Display the processed dataset
print(data_final.head())
```




```
         PC1        PC2  Amenity  Bump  Crossing  Give_Way  Junction  No_Exit  \
0  22.481782  4.055613        0     0         0         0         0        0
1  31.850303  9.280038        0     0         0         0         0        0
2  31.913842  7.205221        0     0         1         0         0        0
3  29.636155  6.956301        0     0         0         0         0        0
4  28.279297  7.390606        0     0         0         0         0        0

   Railway  Station  …  Weather_Thunderstorm  Weather_Windy  Weather_Hail  \
0        0        0  …                     0              0             0
1        0        0  …                     0              0             0
2        0        0  …                     0              0             0
3        0        0  …                     0              0             0
4        0        0  …                     0              0             0

   Weather_Thunder  Wind_C  Wind_E  Wind_N  Wind_S  Wind_V  Wind_W
0                0   False    True   False   False   False   False
1                0   False   False    True   False   False   False
2                0   False    True   False   False   False   False
3                0   False    True   False   False   False   False
4                0   False    True   False   False   False   False

[5 rows x 31 columns]
```

1. After creating the two features, Temp_Humidity_Index and Wind_Impact, we observed that Temp_Humidity_Index has a moderate negative correlation with Pressure(in) (-0.53), suggesting that higher temperature and humidity levels might be linked to lower atmospheric

pressure. Similarly, Wind_Impact shows a moderate negative correlation with Humidity(%) (-0.30), indicating that increased humidity may reduce visibility and influence wind dynamics.
2. To enhance model robustness, we introduced noise to the dataset, creating slight variations in numerical features. Following this, we applied dimensionality reduction to the numerical features using PCA, retaining key information while reducing feature redundancy and improving computational efficiency.

6 Split Data

[51]:
```python
# Define features (X) and target variable (y)
X = data_modelling_df.drop(columns=["Severity"])
y = data_modelling_df["Severity"]

# Split into training (70%) and temp (30%) using stratification
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Split temp set into validation (15%) and test (15%) using stratification
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
)

# Print dataset distribution
print("Training set distribution:\n", y_train.value_counts(normalize=True))
print("\nValidation set distribution:\n", y_val.value_counts(normalize=True))
print("\nTest set distribution:\n", y_test.value_counts(normalize=True))
```

```
Training set distribution:
 Severity
2    0.704104
3    0.247564
1    0.039509
4    0.008822
Name: proportion, dtype: float64

Validation set distribution:
 Severity
2    0.704082
3    0.247583
1    0.039384
4    0.008951
Name: proportion, dtype: float64

Test set distribution:
 Severity
2    0.704135
```

```
3    0.247539
1    0.039556
4    0.008770
Name: proportion, dtype: float64
```

[52]:

```python
# Plot class distributions
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

sns.histplot(y_train, bins=4, ax=axes[0], kde=False, color="blue")
axes[0].set_title("Training Set Severity Distribution")

sns.histplot(y_val, bins=4, ax=axes[1], kde=False, color="green")
axes[1].set_title("Validation Set Severity Distribution")

sns.histplot(y_test, bins=4, ax=axes[2], kde=False, color="red")
axes[2].set_title("Test Set Severity Distribution")

plt.show()
```