Applied Analytics Project

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

Week3 Exploratory Data Analysis (EDA) Report

Major: Applied Analytics

Name: Gefan Wang, Chenhe Shi, Tianchen Liu

Date: 02.09.2025

1. **Dataset Partition Strategy**

To ensure a robust and unbiased analysis, we split the dataset into three subsets:

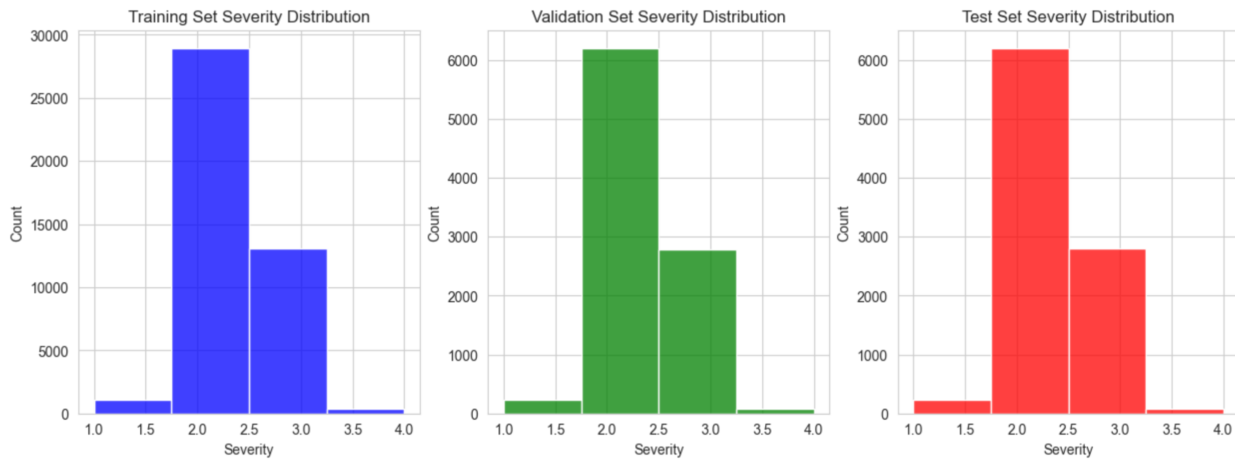
- **Training Set (70%)**: Used for exploratory data analysis (EDA) and model training.
- **Validation Set (15%)**: Used for hyperparameter tuning and performance evaluation.
- **Test Set (15%)**: Used to assess the final model's generalizability.

The rationale behind this split is to prevent overfitting while ensuring sufficient data for analysis and model validation. A 70-15-15 split is commonly used when the dataset is moderately large, ensuring that each subset has enough samples for meaningful analysis. A smaller training set could lead to **underfitting**, where the model fails to recognize patterns effectively. Conversely, allocating too much data to training could reduce the amount available for validation and testing, making it harder to assess and optimize the model properly. Thus, the **70% training split provides a strong foundation for model learning while preserving enough data for validation and testing**. Beyond training, it is crucial to have separate datasets for **hyperparameter tuning (validation set) and final performance evaluation (test set)**. We allocated **15% of the dataset for validation and another 15% for testing**, which translates to approximately **9,299 and 9,300 records, respectively**.

The **validation set** serves as a checkpoint during model development, allowing for hyperparameter tuning and performance monitoring. By evaluating the model on unseen data, we can detect potential **overfitting**, where the model memorizes training data instead of generalizing patterns. A validation set that is too small might result in unreliable tuning decisions, while an excessively large validation set could unnecessarily reduce the training set size.

The **test set** is exclusively reserved for the final evaluation after all model adjustments have been made. This ensures an **unbiased assessment of model performance** on truly unseen data, simulating real-world deployment conditions. A test set that is too small could lead to unreliable estimates of model accuracy, while one that is too large would take away valuable data from training.

After split, we tested out dataset to be reasonable when we test their severity distribution



2 **EDA Analysis & Insights**

2.1 Types of EDA Performed

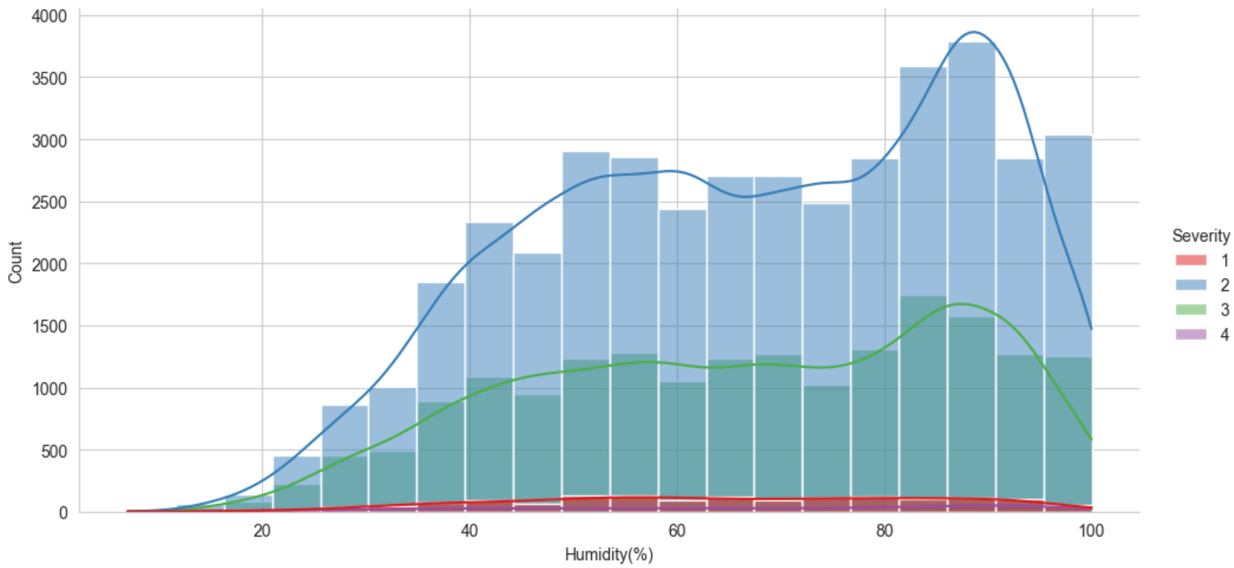
We conducted the following exploratory data analysis:

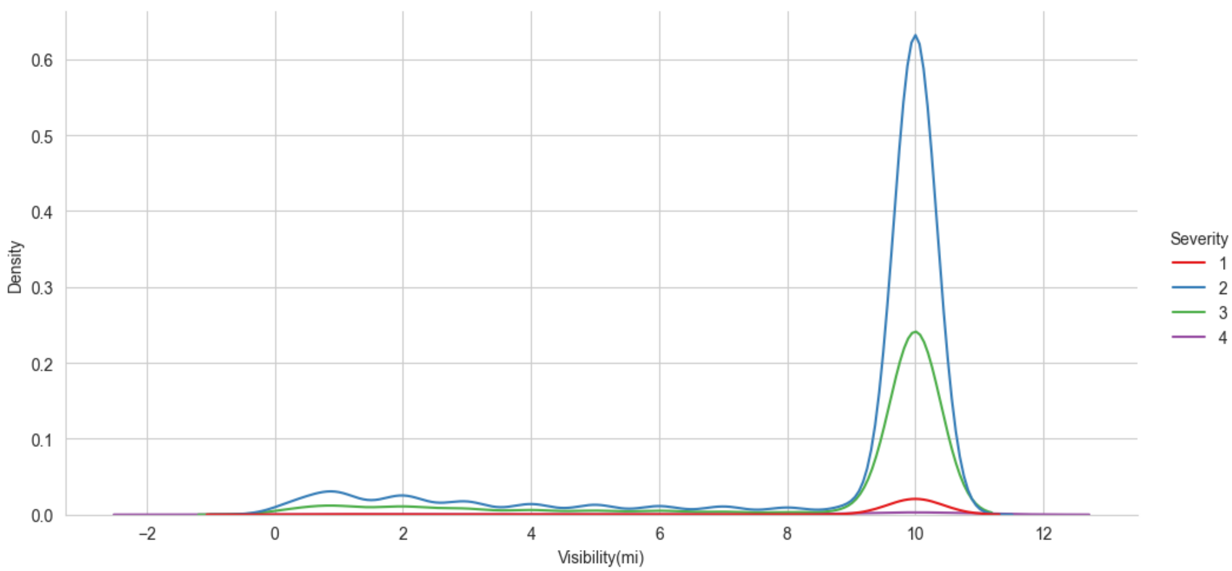
- **Data Structure Analysis**: Used .info() to check data types and missing values.
- **Missing Values Analysis**: Identified missing percentages per column.
- **Descriptive Statistics**: Used .describe() to analyze numerical data distribution.
- **Categorical Data Distribution**: Visualized key categorical features with bar charts.
- **Correlation Analysis**: Generated a heatmap to explore relationships between numerical features.
- **Outlier Detection**: Examined distributions for potential anomalies in key numerical variables.

2.2 Key Findings

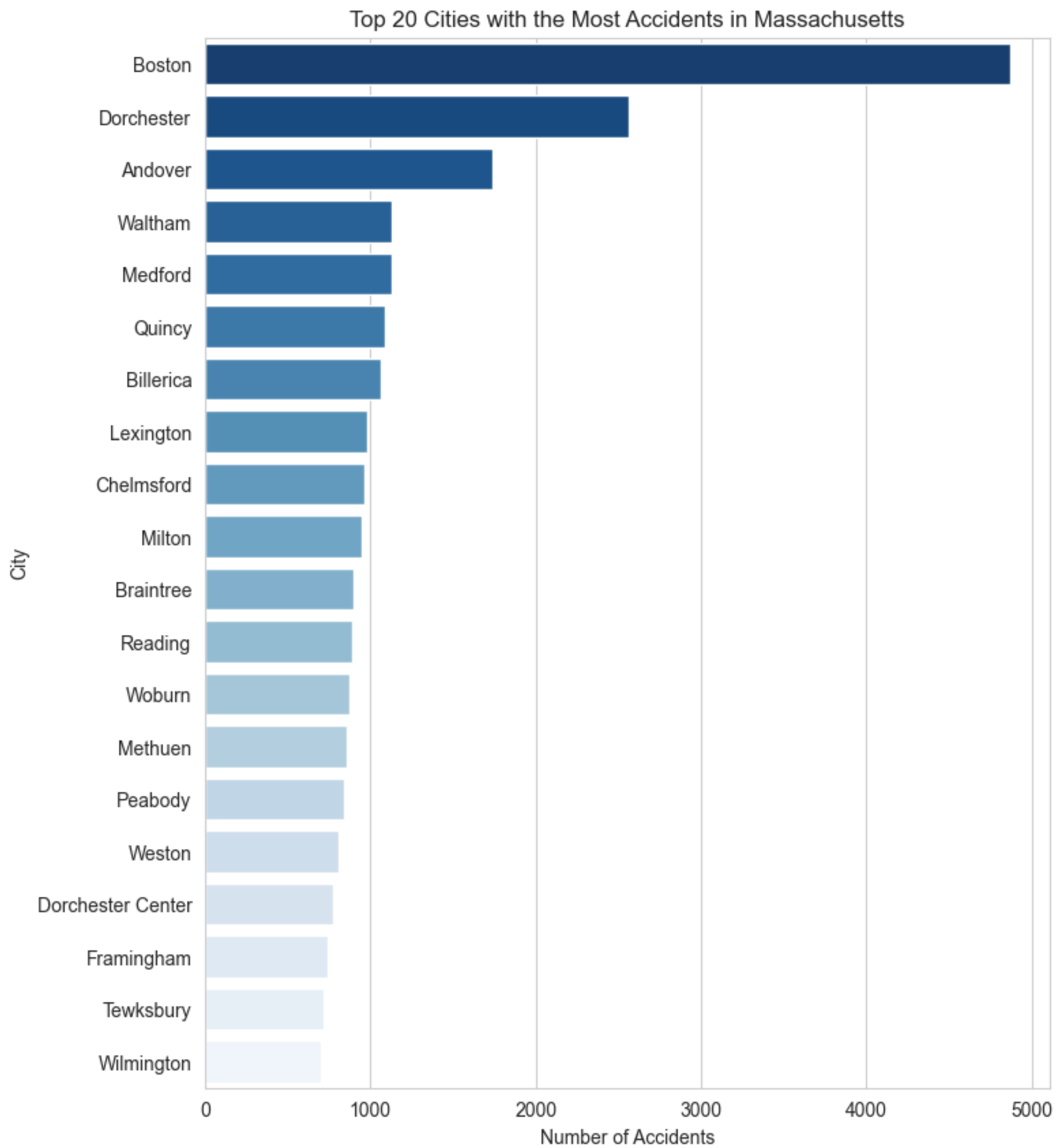
One of the most important aspects of this analysis is identifying how different factors influence accident severity. The EDA reveals that:

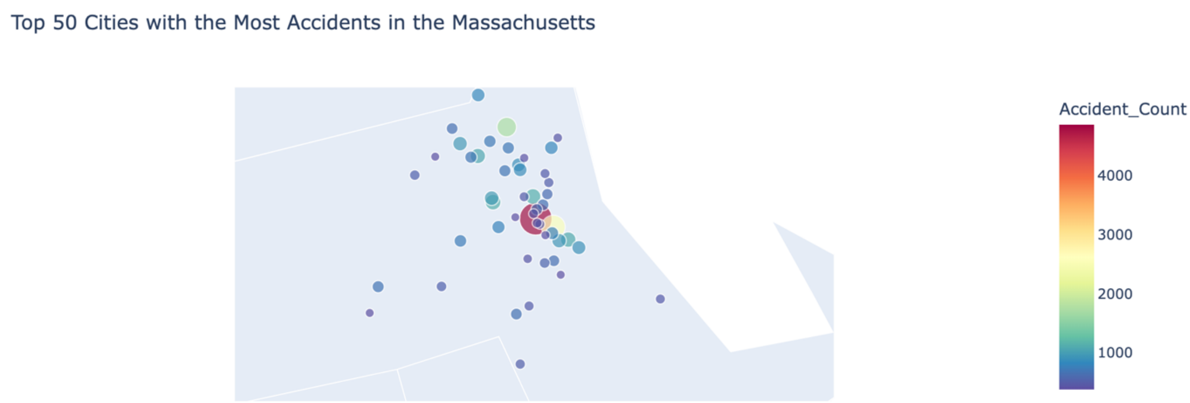
- **Weather conditions play a significant role** in accident severity. Factors like **low visibility, heavy precipitation, and strong winds** appear to correlate with more severe accidents. This aligns with the expectation that poor weather can reduce driver reaction times and increase the likelihood of serious crashes.





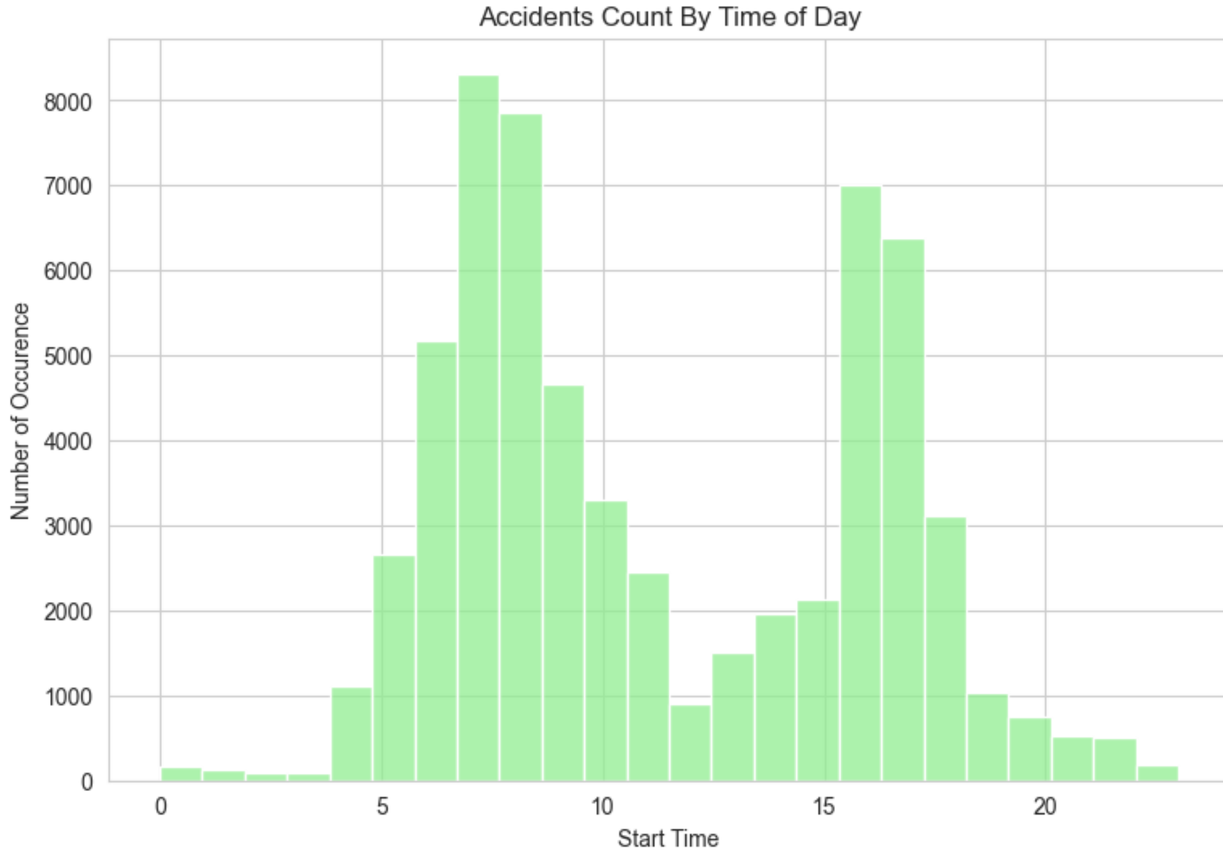
- **Geographical trends** suggest that certain locations experience more severe accidents than others. Citiwise speaking, Boston is the city where most of the accidents happened in MA.





- High-density urban areas with **intersections, crossings, and complex road networks** show a higher frequency of severe accidents. In contrast, highways, where vehicles move at higher speeds but with fewer obstacles, also contribute to severe accidents but for different reasons—such as high-speed collisions.
- **Time-based factors reveal critical patterns.** The analysis suggests that accidents occurring **at night or during twilight periods** tend to be more severe. This could be due to reduced visibility, driver fatigue, and higher chances of impaired driving. Rush-

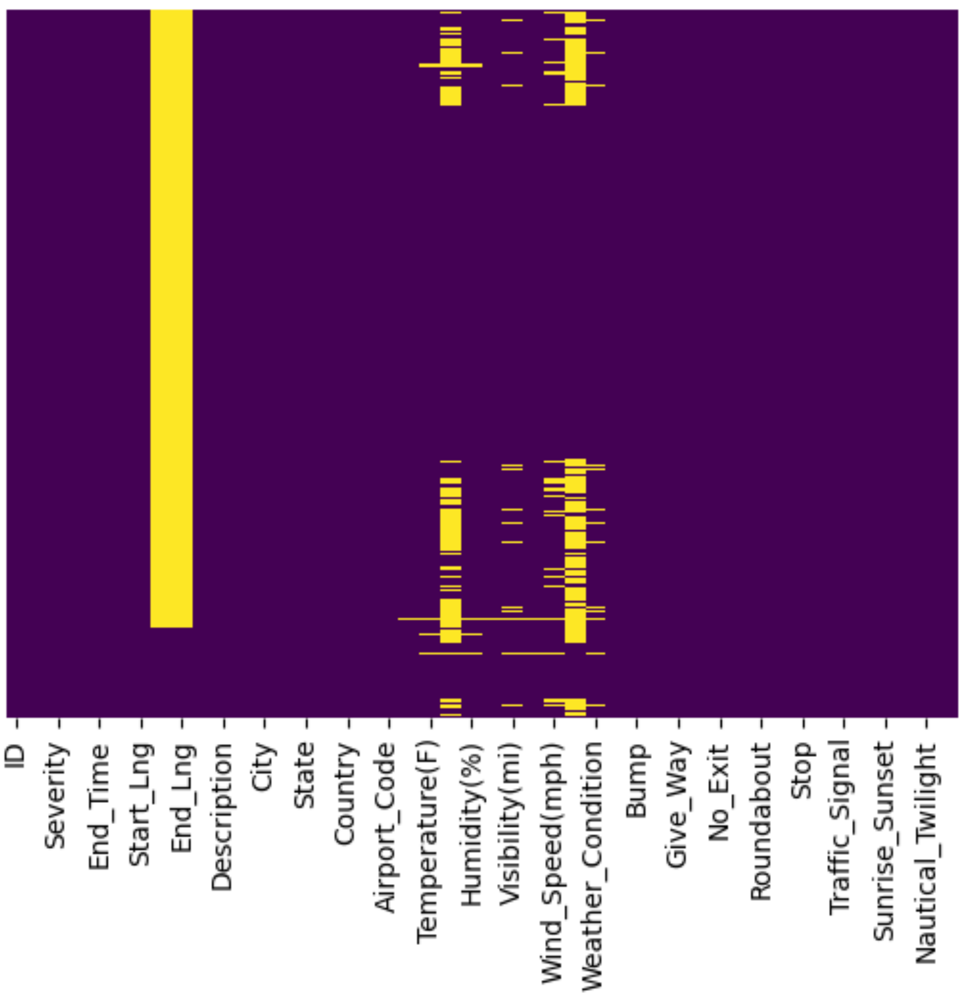
hour periods, while high in accident volume, may not necessarily lead to the most severe incidents due to lower speeds during congestion.



EDA also uncovered several challenges that must be addressed before proceeding with predictive modeling:

- Missing Data in Critical Fields
  Several important variables, particularly those related to weather conditions (wind chill, precipitation) and accident locations (end latitude and longitude), contain missing values. If these missing values are not handled properly, the model could suffer from data bias or reduced accuracy. Imputation techniques, such as using median values for weather conditions or estimating missing coordinates, will be necessary. We have plotted a

missing value heatmap which would help us to better understand of the missing values for this dataset.



- Class Imbalance in the Severity Variable
  The distribution of accident severity is skewed toward lower severity levels. This means that a model trained on the raw data could struggle to predict severe accidents accurately because it would be biased toward the more common minor accidents. To counteract this, we may need to apply resampling techniques, such as oversampling severe cases or using weighted loss functions in machine learning models.

- Outliers in Numerical Variables
  The dataset includes extreme values in temperature, wind chill, and visibility. Some of these values (e.g., -89°F or 207°F) are likely erroneous and must be corrected. These outliers could distort model training if not removed or adjusted.

2.3 Dataset Challenges and Opportunities

Given these findings, we found the opportunities with this dataset lies in:

- Weather-Responsive Traffic Measures:
  Since weather conditions have a strong impact on accident severity, policymakers could implement dynamic speed limits or enhanced warning systems in areas prone to extreme weather conditions. This is especially relevant during winter months when ice and snow create hazardous road conditions.

- Infrastructure Improvements for High-Risk Areas:
  The data suggests that certain types of road structures, such as intersections, crossings, and areas with frequent stops, contribute to higher accident severity. Installing better signage, traffic calming measures, or additional safety barriers in these areas could reduce severe crashes.

- Targeted Resource Allocation for Emergency Services:
  The model could help identify locations that frequently experience high-severity accidents, enabling better deployment of ambulances and emergency response teams. Quick response times in high-risk zones could help minimize fatalities and long-term injuries.

3. Identified Data Problems & Recommendations

3.1 Data Problems

1. **High Percentage of Missing Values**: Several columns have more than 30% missing values, requiring imputation or removal.

2. **Class Imbalance**: Severity levels are not evenly distributed, which might bias model predictions.

3. **Presence of Outliers**: Extreme values in Wind_Speed(mph), Precipitation(in), and Temperature(F) could distort statistical analysis.

4. **String-Based Date Fields**: Start_Time and End_Time need conversion into datetime format for analysis.

3.2 Data Preprocessing Recommendations

1. **Handling Missing Data**:
   - Columns with >50% missing values (End_Lat, End_Lng) should be dropped.
   - Weather-related missing values should be imputed using mean/median values or regression-based approaches.
2. **Balancing Categorical Data**:
   - Use oversampling techniques like SMOTE for underrepresented Severity levels.
   - Weighting loss functions appropriately in predictive models.
3. **Handling Outliers**:
   - Cap extreme values for Wind_Speed(mph), Precipitation(in), and Temperature(F) to prevent model distortion.
4. **Feature Engineering**:
   - Convert Start_Time and End_Time into datetime objects.
   - Extract useful time-based features (e.g., Hour of the Day, Day of the Week).
   - Engineer interaction terms between weather and accident severity for better predictive power.

Week3 Advanced EDA & Data Split

February 9, 2025

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

[3]:
```
# 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
```

```
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)
```

[4]:
```
#import data
accident_data = pd.read_csv("US_Accidents_MA.csv")
```

2 Week 2. Basic EDA

[5]:
```
#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
```

[6]:
```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 (0.0741983353764759%)
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%)
```

[7]:
```
#look at distribution of data
accident_data.describe()
```

[7]:
```
           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
```

[8]:
```python
# 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
```

[9]:
```python
#look at formatting of entries
accident_data.head()
```

[9]:
```
          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]
```

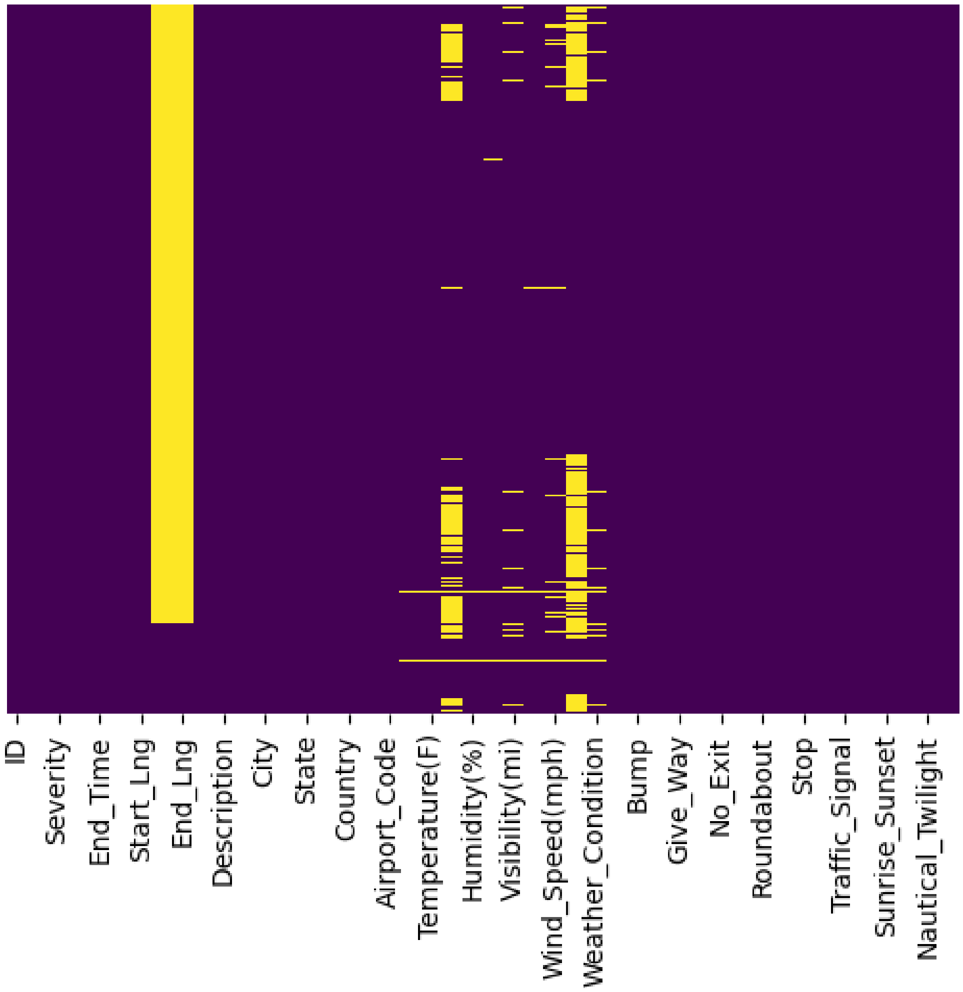
[10]:
```
#looking to see ID format towards end
accident_data.tail()
```

[10]:
```
                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

[11]:
```
# 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
```

[11]: <Axes: >



[12]:
```
# 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())
```

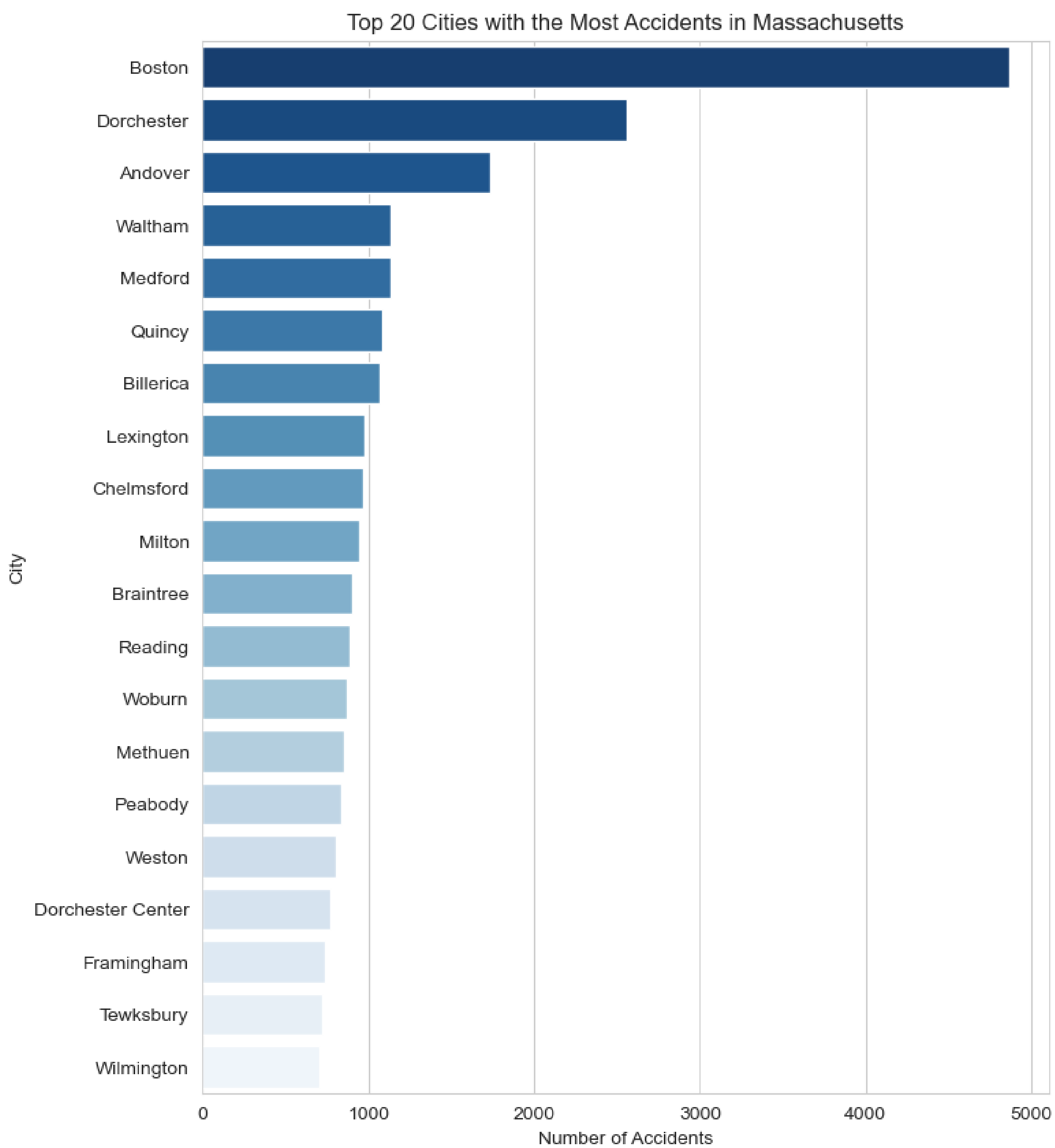
[13]:
```
# Which City has the maximum no: of accidents?

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.
```

[14]:
```
# 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_7265/2744639619.py:11
: 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")
```

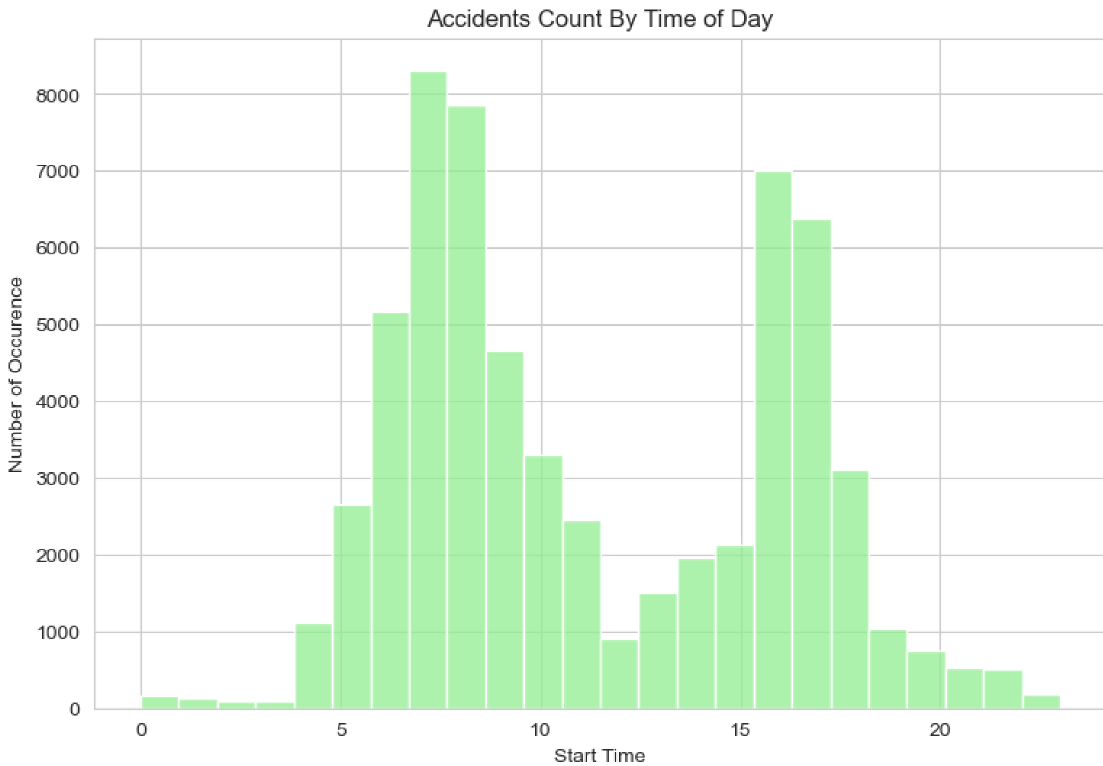


[30]:
```python
# Group accident data by City and State, getting their latitude and longitude
city_wise_counts = accident_data.groupby(["City", "State"])[["Start_Lat",␣
  ↪"Start_Lng"]].first().reset_index()
city_wise_counts["Accident_Count"] = accident_data.groupby(["City",␣
  ↪"State"])["ID"].count().values

# 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()
```

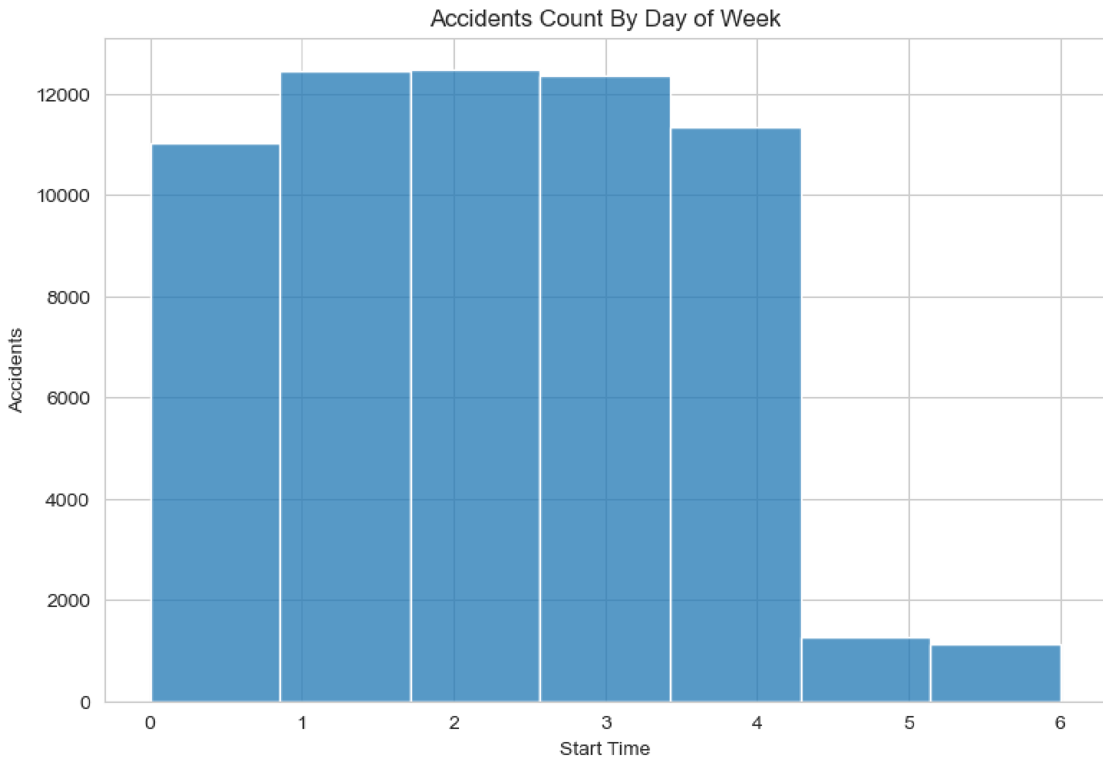
[16]:
```
# Accidents based on Time

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')
```

[16]:
```
Text(0.5, 1.0, 'Accidents Count By Time of Day')
```



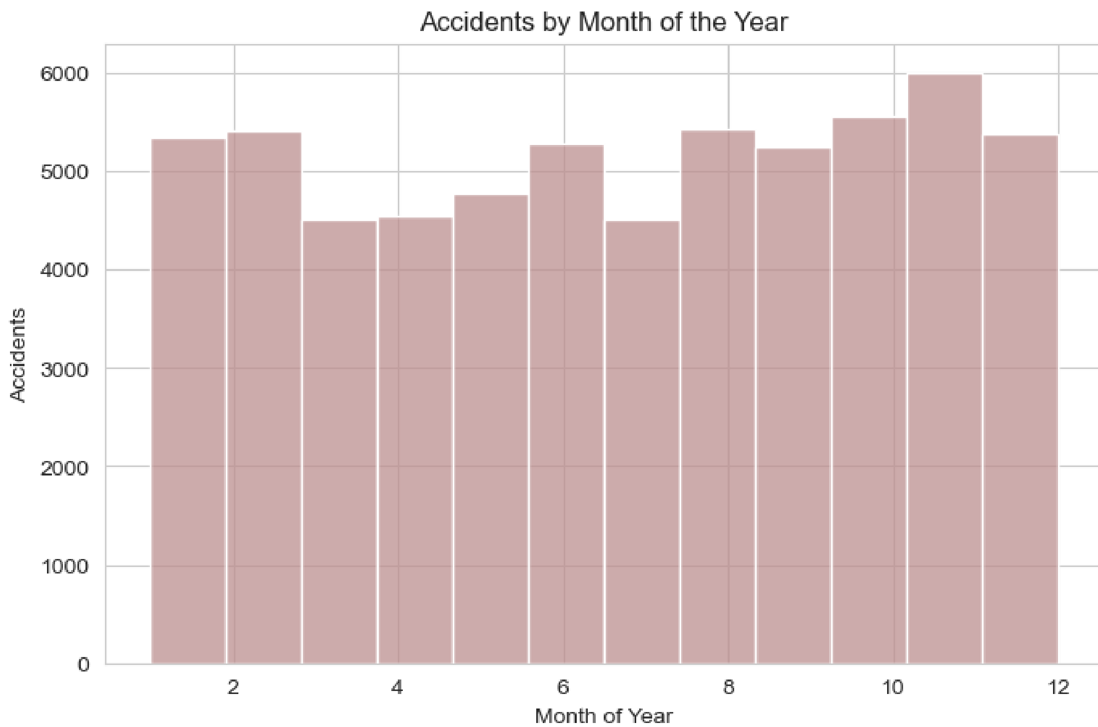
[17]:
```
# To find the accidents by Day of the week

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')
```

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



[18]:
```
# 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')
```

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



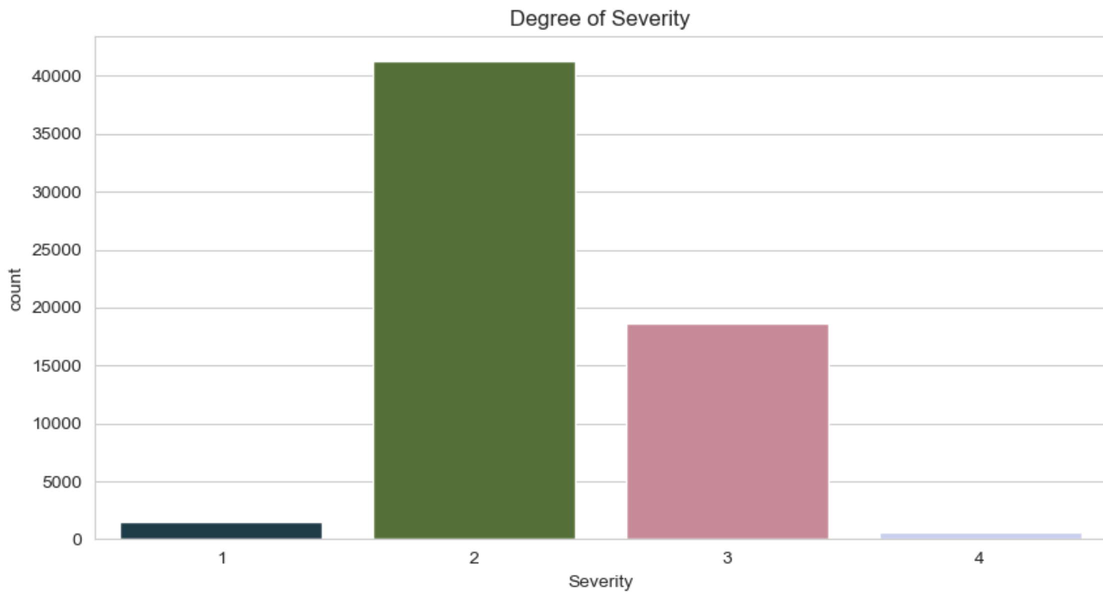
[19]:
```
# Accidents based on Severity and Weather Conditions

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

[19]:
```
Severity
1     1500
2    41326
3    18623
4      547
Name: ID, dtype: int64
```

[20]:
```
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_7265/2458966032.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.
```

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



[21]:
```
weather = accident_data.iloc[:, 20:30]
weather['Severity'] = accident_data['Severity']
weather.head()
```

[21]:

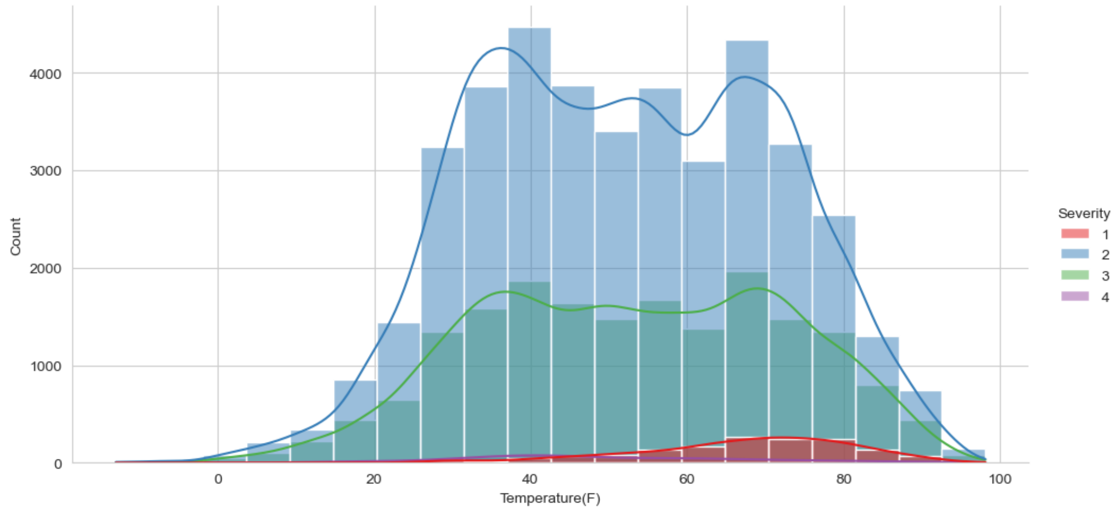
	Temperature(F)	Wind_Chill(F)	Humidity(%)	Pressure(in)	Visibility(mi)	\
0	48.2	45.853027	100.0	29.87	3.0	
1	48.0	45.853027	89.0	29.96	5.0	
2	46.9	45.853027	86.0	30.01	5.0	
3	46.0	41.900000	89.0	30.01	3.0	
4	46.0	41.900000	100.0	29.97	6.0	

	Wind_Direction	Wind_Speed(mph)	Precipitation(in)	Weather_Condition	\
0	Variable	3.5	0.010531	Light Rain	
1	ENE	5.8	0.050000	Rain	
2	ENE	6.9	0.080000	Rain	
3	East	8.1	0.010000	Light Rain	
4	NNE	8.1	0.070000	Light Rain	

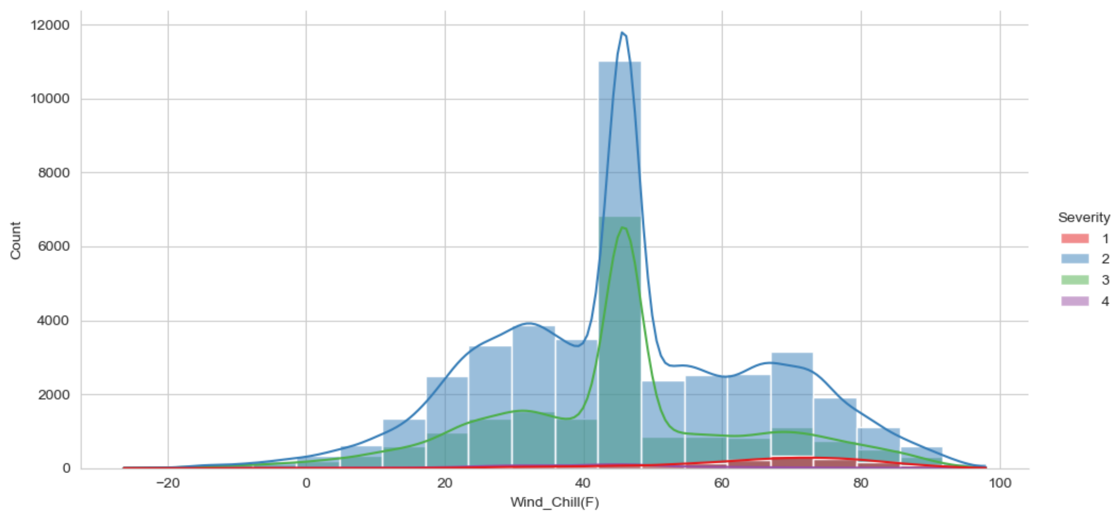
	Amenity	Severity
0	False	2
1	False	2
2	False	3

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

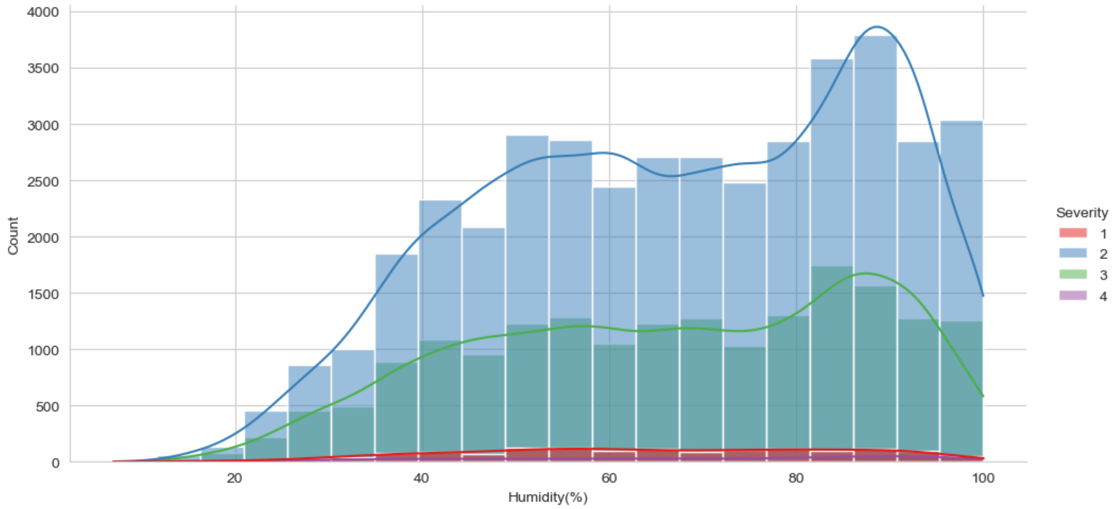
[22]:
```
sns.displot(weather, x="Temperature(F)", hue="Severity", palette="Set1",␣
↪height=5, aspect=2,bins=20,kde=True);
```



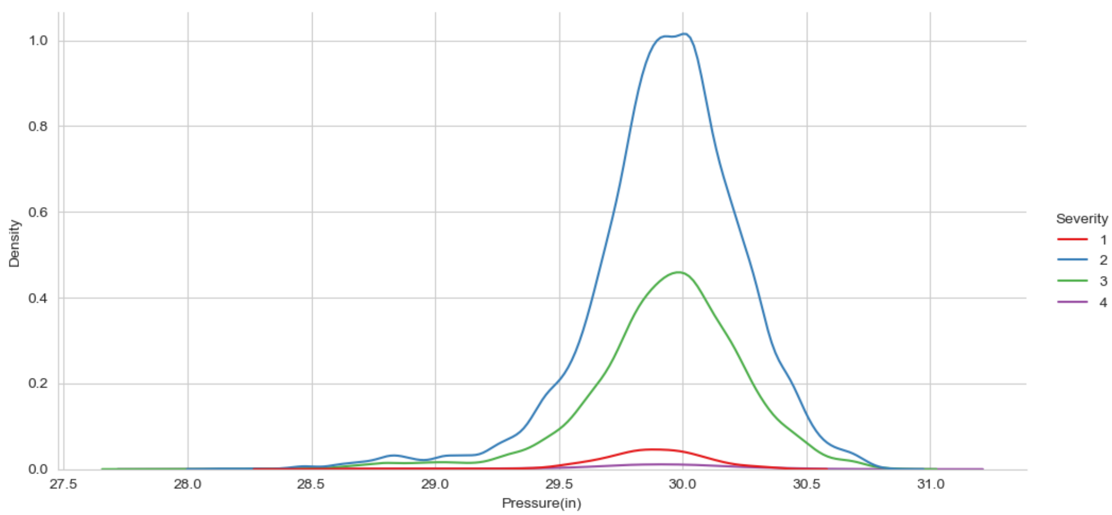
[23]:
```
sns.displot(weather, x="Wind_Chill(F)", hue="Severity",palette="Set1",␣
↪height=5, aspect=2,bins=20,kde=True);
```



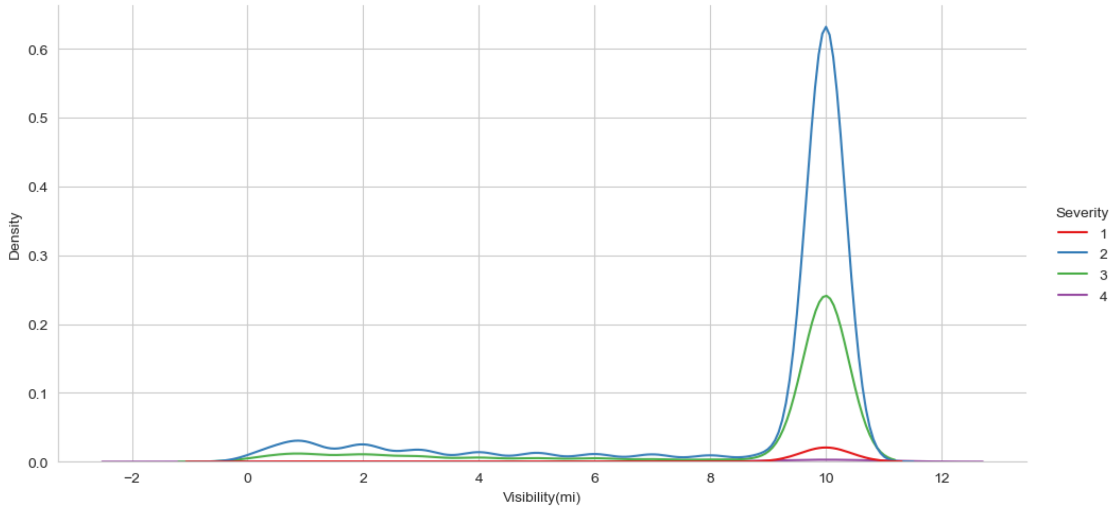
[24]:
```
sns.displot(weather, x="Humidity(%)", hue="Severity", palette="Set1", height=5,␣
↪aspect=2,bins=20,kde=True);
```



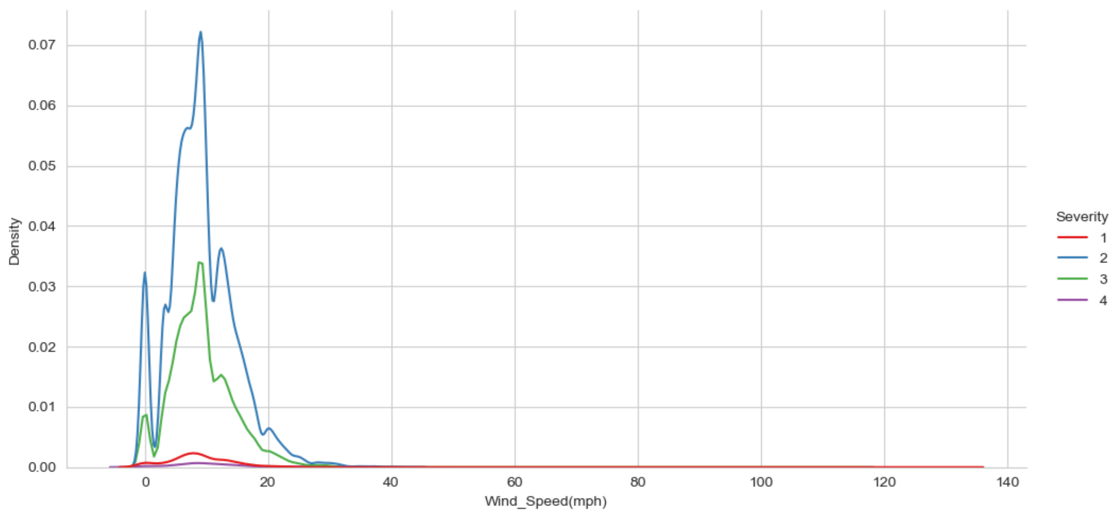
[25]:
```
sns.displot(weather, x="Pressure(in)", hue="Severity", palette="Set1",␣
    ↪height=5, aspect=2,kind='kde');
```



[26]:
```
sns.displot(weather, x="Visibility(mi)", hue="Severity", palette="Set1",␣
    ↪height=5, aspect=2,kind='kde');
```



[27]:
```
sns.displot(weather, x="Wind_Speed(mph)", hue="Severity", palette="Set1",␣
↪height=5, aspect=2,kind='kde');
```



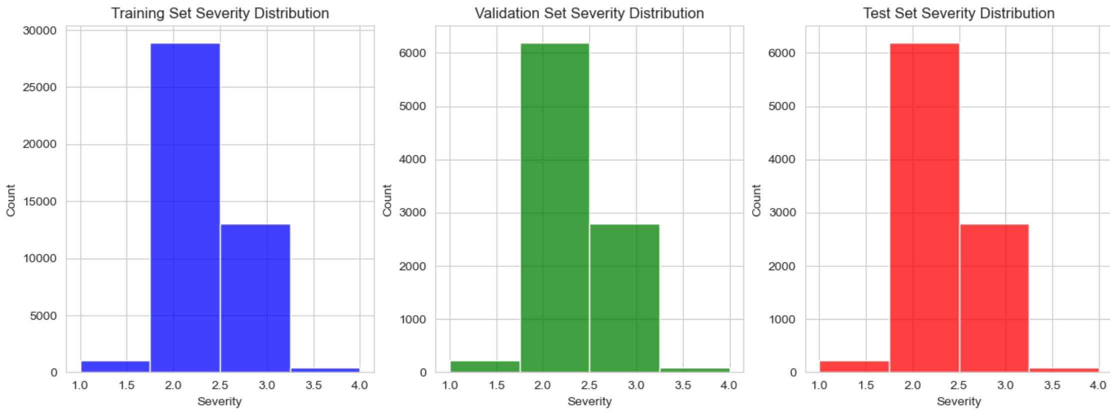
[28]:
```
# Split data

# Define features (X) and target variable (y)
X = accident_data.drop(columns=["Severity"])
y = accident_data["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    0.666590
3    0.300389
1    0.024195
4    0.008825
Name: proportion, dtype: float64

Validation set distribution:
 Severity
2    0.666631
3    0.300355
1    0.024196
4    0.008818
Name: proportion, dtype: float64

Test set distribution:
 Severity
2    0.666559
3    0.300430
1    0.024194
4    0.008817
Name: proportion, dtype: float64
```

[29]:
```
# 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()
```



Training Set Severity Distribution
Validation Set Severity Distribution
Test Set Severity Distribution
Count
Severity