D214_Capstone

November 6, 2022

[1]:
```python
# standard packages
import pandas as pd
import numpy as np

# visuals
import matplotlib.pyplot as plt
import seaborn as sns



#statistics
import statsmodels.api as sm
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# balancing the model
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE

#accuracy of model
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score,␣
↪roc_auc_score,roc_curve,classification_report


import warnings
warnings.filterwarnings("ignore")
```

1 Load Data

[2]:
```python
#file path
file = 'Desktop/US_Accidents_Dec21_updated.csv.zip'

# to search for missing values
missing_values = ['N/A', 'NA', 'None', 'n/a', 'na', 'nAn', 'NaN', '-', '.', ' ']

# parse through Start and End
```

```python
df = pd.read_csv(file, compression='zip', na_values=missing_values)

df.head()
```

```
[2]:    ID  Severity           Start_Time             End_Time  Start_Lat  \
     0  A-1         3  2016-02-08 00:37:08  2016-02-08 06:37:08  40.108910
     1  A-2         2  2016-02-08 05:56:20  2016-02-08 11:56:20  39.865420
     2  A-3         2  2016-02-08 06:15:39  2016-02-08 12:15:39  39.102660
     3  A-4         2  2016-02-08 06:51:45  2016-02-08 12:51:45  41.062130
     4  A-5         3  2016-02-08 07:53:43  2016-02-08 13:53:43  39.172393

        Start_Lng    End_Lat    End_Lng  Distance(mi)  \
     0 -83.092860  40.112060 -83.031870         3.230
     1 -84.062800  39.865010 -84.048730         0.747
     2 -84.524680  39.102090 -84.523960         0.055
     3 -81.537840  41.062170 -81.535470         0.123
     4 -84.492792  39.170476 -84.501798         0.500

                                              Description  …  Roundabout Station  \
     0  Between Sawmill Rd/Exit 20 and OH-315/Olentang…  …       False   False
     1                At OH-4/OH-235/Exit 41 - Accident.  …       False   False
     2                  At I-71/US-50/Exit 1 - Accident.  …       False   False
     3                   At Dart Ave/Exit 21 - Accident.  …       False   False
     4                At Mitchell Ave/Exit 6 - Accident.  …       False   False

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

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

     [5 rows x 47 columns]
```

2 Add Time Columns

[3]:
```
# add Time Columns
df['Start_Time'] = pd.to_datetime(df['Start_Time'])
df['Month'] = df['Start_Time'].dt.month
df['Year'] = df['Start_Time'].dt.year
df['Hour'] = df['Start_Time'].dt.hour
df['Day'] = df['Start_Time'].dt.weekday
```

[4]:
```
# convert to string
df['Month'] = df['Month'].replace([1,2,3,4,5,6,7,8,9,10,11,12],
                                  ['January', 'February', 'March', 'April',␣
 ↪'May',
                                  'June', 'July', 'August', 'September',
                                  'October', 'November', 'December'])
df['Day'] = df['Start_Time'].dt.day_name()
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2845342 entries, 0 to 2845341
Data columns (total 51 columns):
 #   Column                 Dtype
---  ------                 -----
 0   ID                     object
 1   Severity               int64
 2   Start_Time             datetime64[ns]
 3   End_Time               object
 4   Start_Lat              float64
 5   Start_Lng              float64
 6   End_Lat                float64
 7   End_Lng                float64
 8   Distance(mi)           float64
 9   Description            object
 10  Number                 float64
 11  Street                 object
 12  Side                   object
 13  City                   object
 14  County                 object
 15  State                  object
 16  Zipcode                object
 17  Country                object
 18  Timezone               object
 19  Airport_Code           object
 20  Weather_Timestamp      object
 21  Temperature(F)         float64
 22  Wind_Chill(F)          float64
```

```
 23  Humidity(%)             float64
 24  Pressure(in)            float64
 25  Visibility(mi)          float64
 26  Wind_Direction          object
 27  Wind_Speed(mph)         float64
 28  Precipitation(in)       float64
 29  Weather_Condition       object
 30  Amenity                 bool
 31  Bump                    bool
 32  Crossing                bool
 33  Give_Way                bool
 34  Junction                bool
 35  No_Exit                 bool
 36  Railway                 bool
 37  Roundabout              bool
 38  Station                 bool
 39  Stop                    bool
 40  Traffic_Calming         bool
 41  Traffic_Signal          bool
 42  Turning_Loop            bool
 43  Sunrise_Sunset          object
 44  Civil_Twilight          object
 45  Nautical_Twilight       object
 46  Astronomical_Twilight   object
 47  Month                   object
 48  Year                    int64
 49  Hour                    int64
 50  Day                     object
dtypes: bool(13), datetime64[ns](1), float64(13), int64(3), object(21)
memory usage: 860.2+ MB
```

[6]:
```
df.shape
```

[6]:
```
(2845342, 51)
```

[7]:
```
# check to see if any columns are duplicated
df.columns.duplicated().any()
```

[7]:
```
False
```

[8]:
```
# look at unique values per column
df.nunique()
```

[8]:
```
ID                       2845342
Severity                       4
Start_Time               1807311
End_Time                 2351505
```

Start_Lat	1093618
Start_Lng	1120365
End_Lat	1080811
End_Lng	1105404
Distance(mi)	14165
Description	1174563
Number	46402
Street	159651
Side	3
City	11681
County	1707
State	49
Zipcode	363085
Country	1
Timezone	4
Airport_Code	2004
Weather_Timestamp	474214
Temperature(F)	788
Wind_Chill(F)	897
Humidity(%)	100
Pressure(in)	1068
Visibility(mi)	76
Wind_Direction	24
Wind_Speed(mph)	136
Precipitation(in)	230
Weather_Condition	127
Amenity	2
Bump	2
Crossing	2
Give_Way	2
Junction	2
No_Exit	2
Railway	2
Roundabout	2
Station	2
Stop	2
Traffic_Calming	2
Traffic_Signal	2
Turning_Loop	1
Sunrise_Sunset	2
Civil_Twilight	2
Nautical_Twilight	2
Astronomical_Twilight	2
Month	12
Year	6
Hour	24
Day	7

```
dtype: int64
```

to clean up later -> Street, Weather_Condition, Wind_Direction

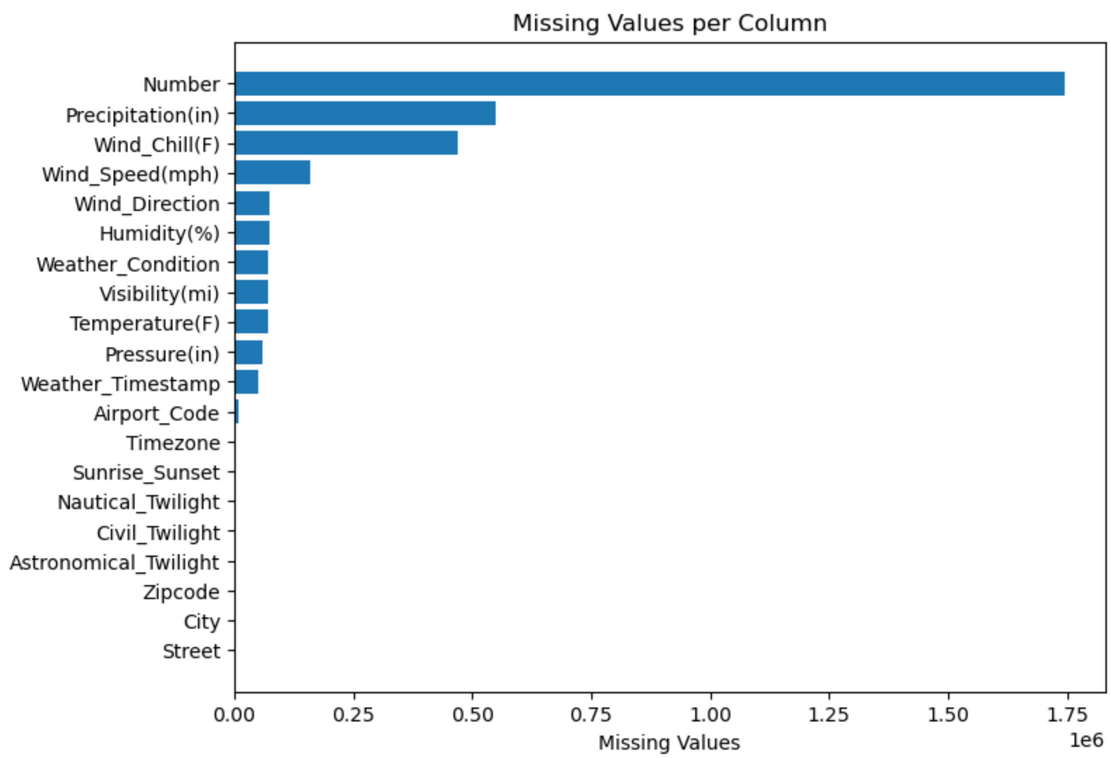
[9]:
```
# check for missing values based off of list of missing values
df.isna().any()
```

[9]:
```
ID                       False
Severity                 False
Start_Time               False
End_Time                 False
Start_Lat                False
Start_Lng                False
End_Lat                  False
End_Lng                  False
Distance(mi)             False
Description              False
Number                    True
Street                    True
Side                     False
City                      True
County                   False
State                    False
Zipcode                   True
Country                  False
Timezone                  True
Airport_Code              True
Weather_Timestamp         True
Temperature(F)            True
Wind_Chill(F)             True
Humidity(%)               True
Pressure(in)              True
Visibility(mi)            True
Wind_Direction            True
Wind_Speed(mph)           True
Precipitation(in)         True
Weather_Condition         True
Amenity                  False
Bump                     False
Crossing                 False
Give_Way                 False
Junction                 False
No_Exit                  False
Railway                  False
Roundabout               False
Station                  False
Stop                     False
```

```
Traffic_Calming          False
Traffic_Signal           False
Turning_Loop             False
Sunrise_Sunset            True
Civil_Twilight            True
Nautical_Twilight         True
Astronomical_Twilight     True
Month                    False
Year                     False
Hour                     False
Day                      False
dtype: bool
```

[10]:

```
# look at number of missing values in each column

missing_df = df.isnull().sum(axis=0).reset_index()
missing_df.columns = ['column_name','missing_count']
missing_df = missing_df[missing_df['missing_count']>0]
missing_df = missing_df.sort_values(by='missing_count')

ind = np.arange(missing_df.shape[0])
width = 0.5
fig,ax = plt.subplots(figsize=(8,6))
rects = ax.barh(ind,missing_df.missing_count.values)
ax.set_yticks(ind)
ax.set_yticklabels(missing_df.column_name.values, rotation='horizontal')
ax.set_xlabel("Missing Values")
ax.set_title("Missing Values per Column")
plt.show()
```



3 Clean Data

[11]:
```
# fill street number with 0, since it's house/business number
df['Number'] = df['Number'].fillna(0)
df['Number'].isna().sum()
```

[11]: 0

[12]:
```
print('Shape before dropna()', df.shape)

df = df.dropna()

print('Shape after drapna()', df.shape)
```

```
Shape before dropna() (2845342, 51)
Shape after drapna() (2207325, 51)
```

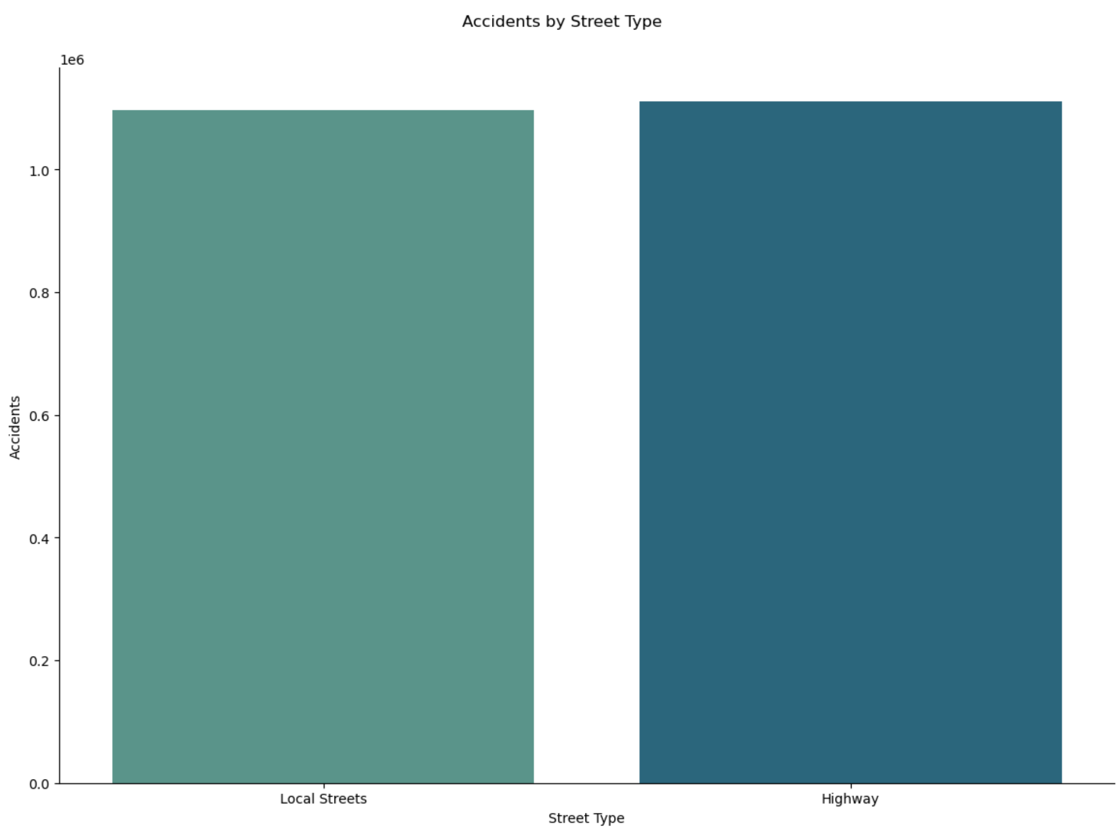
3.0.1 Clean Street Values

[13]:
```
# split highways from local roads
def str_type(text):
```

```
    if '-' in text or 'Fwy'in text or 'Expy' in text or 'Highway'in text or␣
 ↪'Hwy'in text :
        result = 'Highway'
    else:
        result = 'Local Streets'
    return result

df['Street'] = df['Street'].apply(str_type)
```

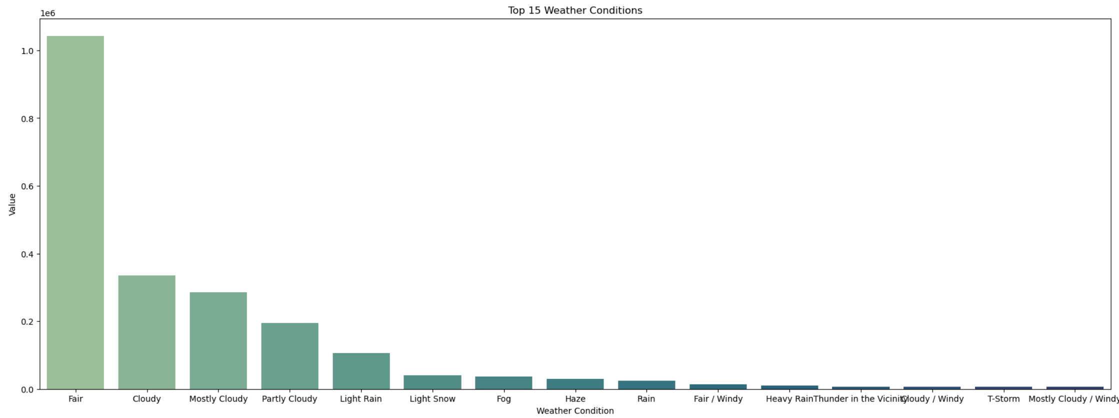
[14]:
```
print(df.Street.value_counts())
h = sns.catplot(x='Street',data=df, kind='count', height=8.27, aspect=11.7/8.
 ↪27, palette='crest')
h.fig.suptitle('Accidents by Street Type', y=1.03)
h.set(ylabel='Accidents', xlabel='Street Type')
plt.show()
```

```
Highway          1110765
Local Streets    1096560
Name: Street, dtype: int64
```



3.0.2 Clean Weather Conditions

[15]:
```
# top 15 weather conditions of Accidents
counts = df["Weather_Condition"].value_counts()[:15]
plt.figure(figsize=(23,8))
sns.barplot(counts.index, counts.values, palette='crest')
plt.title("Top 15 Weather Conditions")
plt.xlabel("Weather Condition")
plt.ylabel("Value")
plt.show()
```



[16]:
```
# look at all values in weather condition
weather_list = df['Weather_Condition'].values.tolist()
list(set(weather_list))
```

[16]:
```
['Light Drizzle',
 'Sand / Dust Whirls Nearby',
 'Light Freezing Fog',
 'Blowing Sand',
 'Heavy Drizzle',
 'Thunder in the Vicinity',
 'Drizzle / Windy',
 'Freezing Rain',
 'Heavy Snow / Windy',
 'Thunder',
 'Snow and Sleet / Windy',
 'Snow and Sleet',
 'Sleet / Windy',
 'Partly Cloudy / Windy',
 'Light Freezing Rain / Windy',
 'Light Blowing Snow',
 'Heavy T-Storm / Windy',
 'T-Storm / Windy',
```

```
'Thunder / Wintry Mix / Windy',
'Heavy T-Storm',
'Heavy Sleet',
'Light Snow and Sleet / Windy',
'Light Snow with Thunder',
'Sand / Dust Whirlwinds / Windy',
'Blowing Snow Nearby',
'Drizzle',
'Heavy Rain Shower / Windy',
'Light Ice Pellets',
'Heavy Rain Shower',
'Scattered Clouds',
'Small Hail',
'Widespread Dust',
'Light Rain Showers',
'Widespread Dust / Windy',
'Snow Grains',
'Tornado',
'Wintry Mix',
'Light Sleet',
'Heavy Ice Pellets',
'Wintry Mix / Windy',
'Heavy Snow with Thunder',
'Haze / Windy',
'Cloudy / Windy',
'Light Rain',
'Light Rain with Thunder',
'Thunder / Wintry Mix',
'Rain Shower',
'Blowing Dust',
'Heavy Rain / Windy',
'Snow / Windy',
'Patches of Fog',
'Mist',
'Patches of Fog / Windy',
'Freezing Drizzle',
'Light Freezing Drizzle',
'Fog / Windy',
'Sand / Dust Whirlwinds',
'Thunder and Hail',
'Ice Pellets',
'Rain',
'Light Rain / Windy',
'Squalls / Windy',
'Partial Fog',
'Sand / Windy',
'Light Snow / Windy',
```

```
'Fair',
'Light Snow Shower',
'Fair / Windy',
'Fog',
'Hail',
'Blowing Dust / Windy',
'Light Thunderstorms and Rain',
'Smoke',
'Squalls',
'Thunder and Hail / Windy',
'Light Rain Shower / Windy',
'Heavy Thunderstorms and Snow',
'Thunder / Windy',
'Light Drizzle / Windy',
'Cloudy',
'Blowing Snow / Windy',
'Drifting Snow',
'Overcast',
'Mostly Cloudy / Windy',
'T-Storm',
'Showers in the Vicinity',
'Light Freezing Rain',
'Drizzle and Fog',
'Light Thunderstorms and Snow',
'Heavy Thunderstorms with Small Hail',
'Thunderstorms and Rain',
'Snow',
'Heavy Freezing Rain',
'Snow and Thunder / Windy',
'Light Sleet / Windy',
'Light Snow',
'Light Snow and Sleet',
'Shallow Fog',
'Heavy Freezing Drizzle',
'Partly Cloudy',
'Heavy Snow',
'Clear',
'Freezing Rain / Windy',
'Smoke / Windy',
'Sleet',
'Heavy Rain',
'Blowing Snow',
'Heavy Thunderstorms and Rain',
'Rain / Windy',
'Haze',
'Light Rain Shower',
'N/A Precipitation',
```

```
 'Thunderstorm',
 'Mostly Cloudy']
```

[17]:
```
weather_dict = {
    'Heavy Rain Shower / Windy':'Rain',
 'Blowing Dust':'Fog',
 'Cloudy / Windy':'Cloudy',
 'Shallow Fog':'Fog',
 'Mostly Cloudy':'Fog',
 'Fog':'Fog',
 'Light Snow and Sleet':'Ice',
 'Snow / Windy':'Snow',
 'Heavy Freezing Rain':'Ice',
 'Snow':'Snow',
 'Light Thunderstorms and Snow':'Snow',
 'Thunder and Hail':'Ice',
 'Smoke / Windy':'Fog',
 'Light Snow with Thunder':'Snow',
 'Light Sleet / Windy':'Ice',
 'Drizzle':'Rain',
 'Light Blowing Snow':'Snow',
 'Heavy Snow with Thunder':'Snow',
 'Showers in the Vicinity':'Cloudy',
 'T-Storm / Windy':'Thunderstorm',
 'Light Freezing Rain':'Ice',
 'Mist':'Rain',
 'Squalls':'Fog',
 'Thunderstorms and Rain':'Thunderstorm',
 'Light Rain Shower / Windy':'Rain',
 'Thunderstorm':'Thunderstorm',
 'Partial Fog':'Fog',
 'Sand / Dust Whirls Nearby':'Fog',
 'Heavy T-Storm':'Thunderstorm',
 'Drizzle and Fog':'Fog',
 'Widespread Dust':'Fog',
 'Tornado':'Thunderstorm',
 'Freezing Rain / Windy':'Ice',
 'Squalls / Windy':'Fog',
 'Thunder / Windy':'Thunderstorm',
 'Heavy Snow / Windy':'Snow',
 'Clear':'Clear',
 'Scattered Clouds':'Cloudy',
 'Mostly Cloudy / Windy':'Cloudy',
 'Light Rain Showers':'Rain',
 'Light Freezing Fog':'Fog',
 'Drifting Snow':'Snow',
 'Sleet / Windy':'Ice',
```

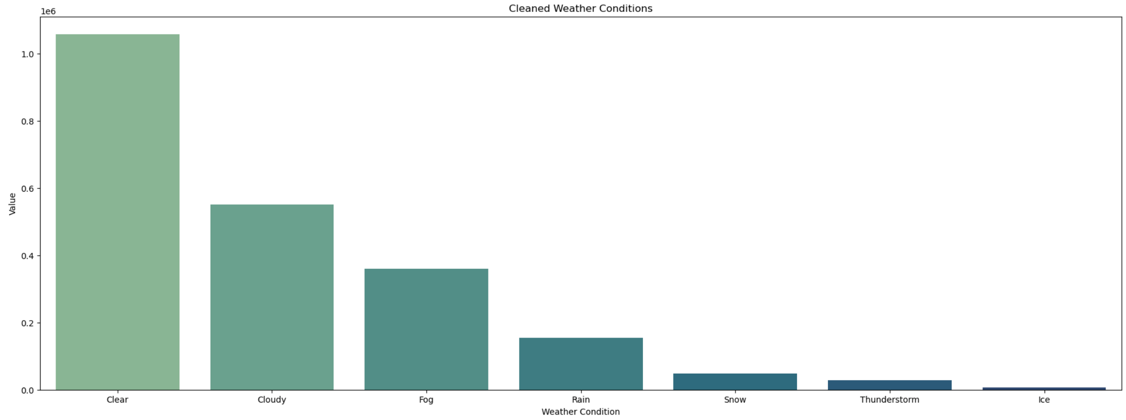
```
'Sand / Windy':'Fog',
'Freezing Drizzle':'Ice',
'Light Ice Pellets':'Ice',
'Light Rain':'Rain',
'Cloudy':'Cloudy',
'Snow and Thunder / Windy':'Snow',
'Blowing Snow':'Snow',
'Heavy Rain':'Rain',
'Light Snow and Sleet / Windy':'Ice',
'Heavy Ice Pellets':'Ice',
'Light Rain with Thunder':'Thunderstorm',
'Freezing Rain':'Ice',
'Partly Cloudy':'Cloudy',
'Snow Grains':'Snow',
'Thunder':'Thunderstorm',
'Sand / Dust Whirlwinds / Windy':'Fog',
'Widespread Dust / Windy':'Fog',
'Light Thunderstorms and Rain':'Thunderstorm',
'Small Hail':'Ice',
'Light Rain Shower':'Rain',
'Fair / Windy':'Clear',
'Heavy Thunderstorms with Small Hail':'Ice',
'Blowing Dust / Windy':'Fog',
'Patches of Fog / Windy':'Fog',
'Blowing Sand':'Fog',
'Sand / Dust Whirlwinds':'Fog',
'Heavy T-Storm / Windy':'Thunderstorm',
'T-Storm':'Thunderstorm',
'Rain Shower':'Rain',
'Light Snow Shower':'Snow',
'Sleet':'Ice',
'N/A Precipitation':'Ice',
'Light Drizzle / Windy':'Rain',
'Light Freezing Rain / Windy':'Ice',
'Rain / Windy':'Rain',
'Thunder / Wintry Mix':'Ice',
'Fog / Windy':'Fog',
'Smoke':'Fog',
'Overcast':'Cloudy',
'Heavy Sleet':'Ice',
'Light Drizzle':'Rain',
'Light Snow / Windy':'Snow',
'Heavy Rain Shower':'Rain',
'Haze':'Fog',
'Snow and Sleet':'Ice',
'Light Rain / Windy':'Rain',
'Thunder / Wintry Mix / Windy':'Ice',
```

```
 'Haze / Windy':'Fog',
 'Hail':'Ice',
 'Rain':'Rain',
 'Blowing Snow / Windy':'Snow',
 'Thunder in the Vicinity':'Thunderstorm',
 'Patches of Fog':'Fog',
 'Wintry Mix / Windy':'Ice',
 'Fair':'Clear',
 'Light Sleet':'Ice',
 'Light Freezing Drizzle':'Ice',
 'Heavy Thunderstorms and Rain':'Thunderstorm',
 'Heavy Drizzle':'Rain',
 'Snow and Sleet / Windy':'Ice',
 'Thunder and Hail / Windy':'Thunderstorm',
 'Partly Cloudy / Windy':'Cloudy',
 'Ice Pellets':'Ice',
 'Blowing Snow Nearby':'Snow',
 'Wintry Mix':'Ice',
 'Heavy Rain / Windy':'Rain',
 'Drizzle / Windy':'Rain',
 'Light Snow':'Snow',
 'Heavy Freezing Drizzle':'Ice',
 'Heavy Snow':'Snow',
 'Heavy Thunderstorms and Snow':'Snow'


}

df['Weather_Condition'] = df.Weather_Condition.map(weather_dict)
# updated weather values
df.Weather_Condition.value_counts()
```

[17]:
```
Clear           1057044
Cloudy           551255
Fog              360506
Rain             154345
Snow              48989
Thunderstorm      28681
Ice                6505
Name: Weather_Condition, dtype: int64
```

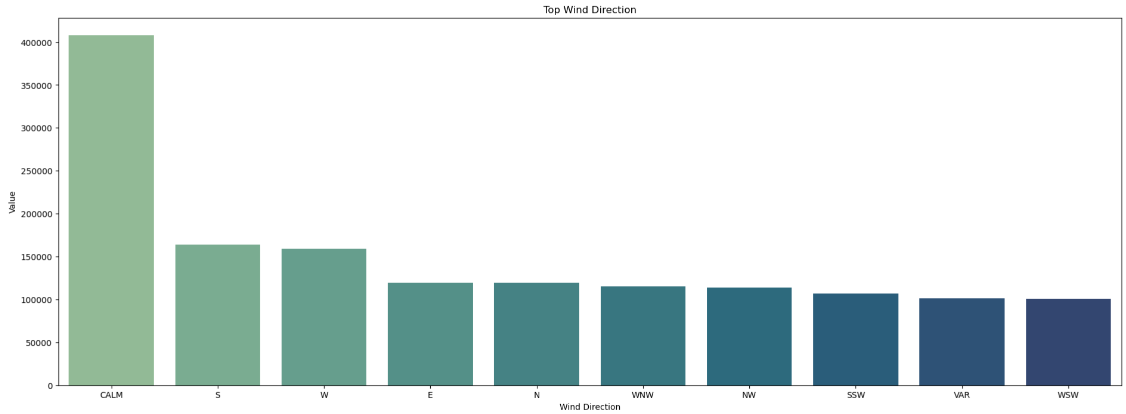
[18]:
```
counts = df['Weather_Condition'].value_counts()
# Cleaned weather conditions of Accidents
plt.figure(figsize=(23,8))
sns.barplot(counts.index, counts.values, palette='crest')
plt.title("Cleaned Weather Conditions")
plt.xlabel("Weather Condition")
```

```
plt.ylabel("Value")
plt.show()
```



3.0.3 Clean Wind Direction

[19]:
```
# top 15 weather conditions of Accidents
counts = df["Wind_Direction"].value_counts()[:10]
plt.figure(figsize=(23,8))
sns.barplot(counts.index, counts.values, palette='crest')
plt.title("Top Wind Direction")
plt.xlabel("Wind Direction")
plt.ylabel("Value")
plt.show()
```



[20]:
```
# all values in wind list
wind_list = df['Wind_Direction'].values.tolist()
list(set(wind_list))
```

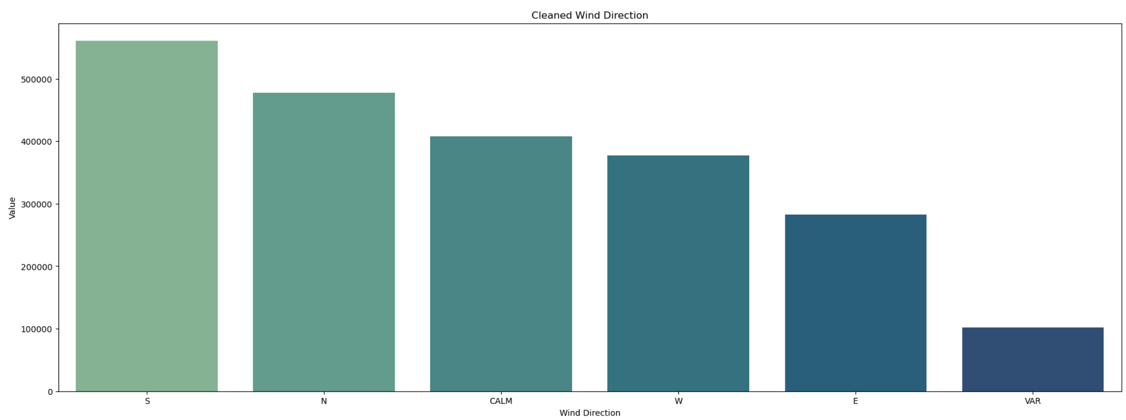
[20]: ['ESE',
 'NW',
 'South',
 'ENE',
 'SSE',
 'WNW',
 'VAR',
 'SW',
 'N',
 'S',
 'Variable',
 'SSW',
 'CALM',
 'West',
 'East',
 'W',
 'SE',
 'E',
 'NNW',
 'NNE',
 'NE',
 'North',
 'WSW']

[21]:
```
wind_dict = {
    'SSE':'S',
 'NNE':'N',
 'WNW':'W',
 'ENE':'E',
 'SE':'S',
 'N':'N',
 'VAR':'VAR',
 'NE':'N',
 'CALM':'CALM',
 'Variable':'VAR',
 'West':'W',
 'ESE':'E',
 'SW':'S',
 'South':'S',
 'WSW':'W',
 'SSW':'S',
 'North':'N',
 'S':'S',
 'NNW':'N',
 'NW':'N',
 'East':'E',
 'E':'E',
```

```
 'W':'W'
}
df['Wind_Direction'] = df.Wind_Direction.map(wind_dict)
df['Wind_Direction'].value_counts()
```

[21]:
```
S       560706
N       477241
CALM    408032
W       376976
E       282818
VAR     101552
Name: Wind_Direction, dtype: int64
```

[22]:
```
# cleaned wind direction
counts = df["Wind_Direction"].value_counts()
plt.figure(figsize=(23,8))
sns.barplot(counts.index, counts.values, palette='crest')
plt.title("Cleaned Wind Direction")
plt.xlabel("Wind Direction")
plt.ylabel("Value")
plt.show()
```

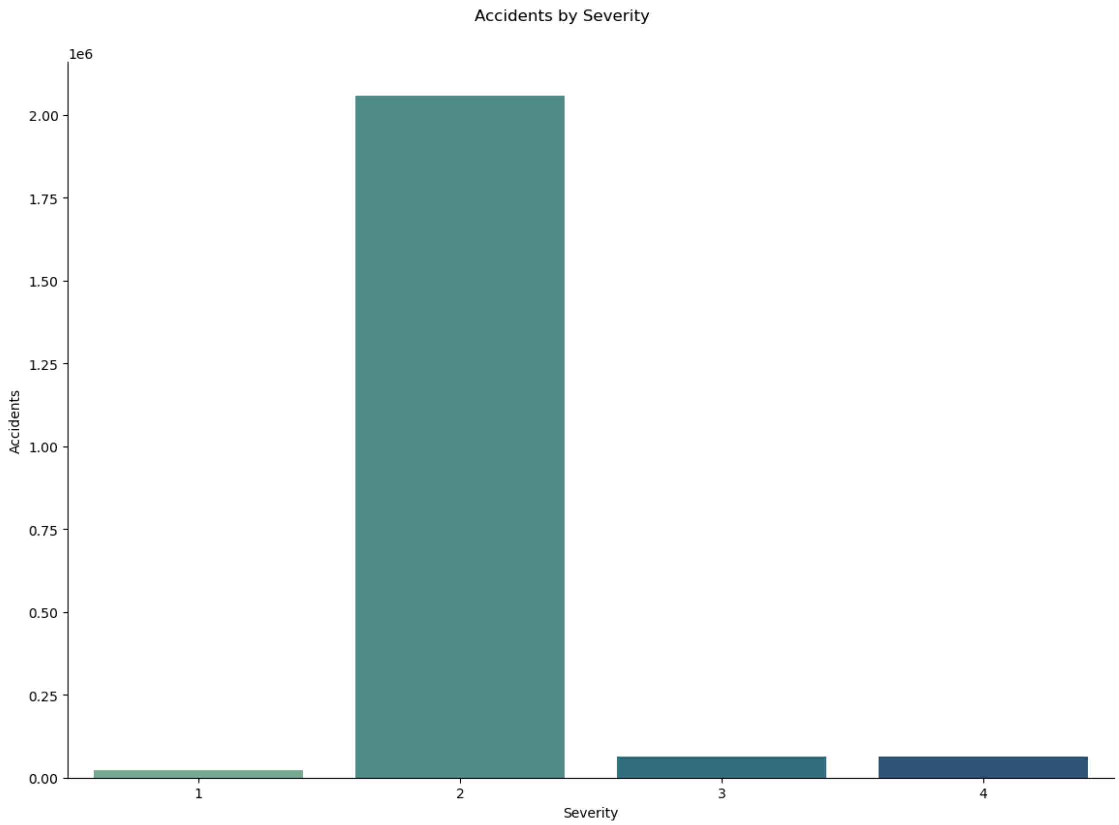


4 Explore Data

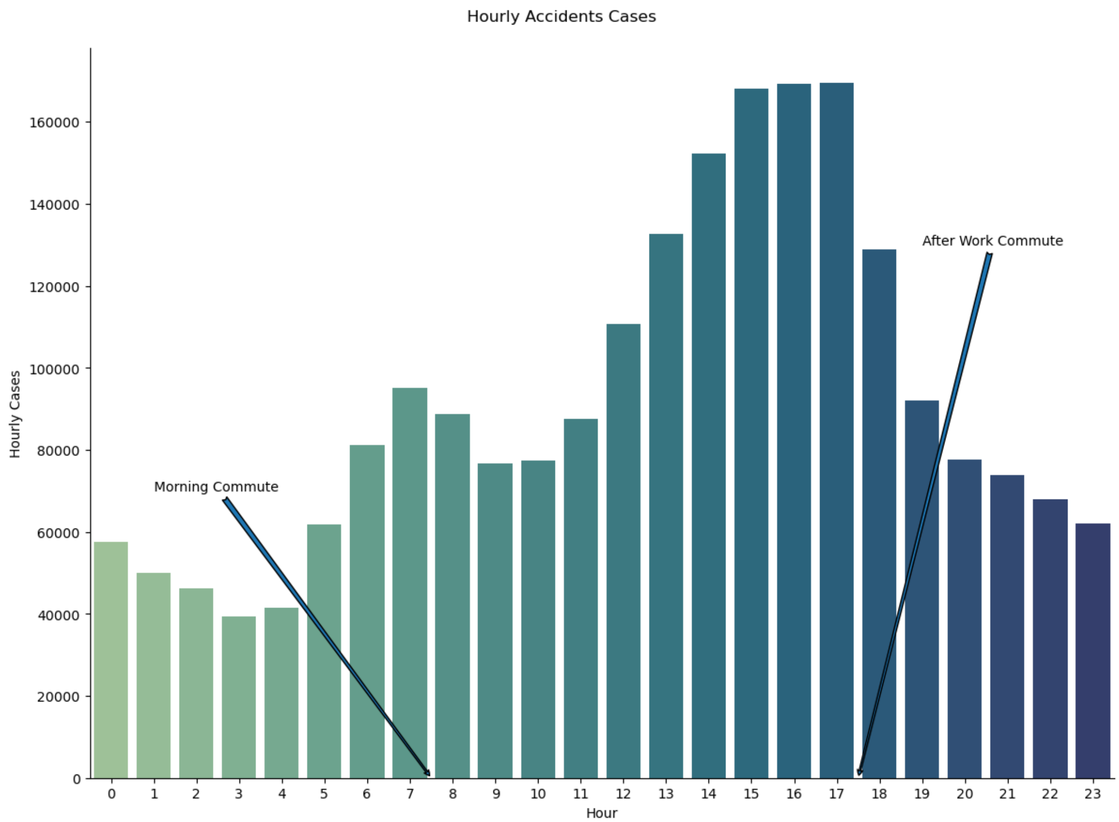
[23]:
```
# severe accidents
print('Number of Accidents per Severity level:')
print(df.Severity.value_counts())
h = sns.catplot(x='Severity',data=df, kind='count', height=8.27, aspect=11.7/8.
 ↪27, palette='crest')
h.fig.suptitle('Accidents by Severity', y=1.03)
h.set(ylabel='Accidents', xlabel='Severity')
```

```
plt.show()
```

```
Number of Accidents per Severity level:
2    2057075
3      64588
4      62106
1      23556
Name: Severity, dtype: int64
```

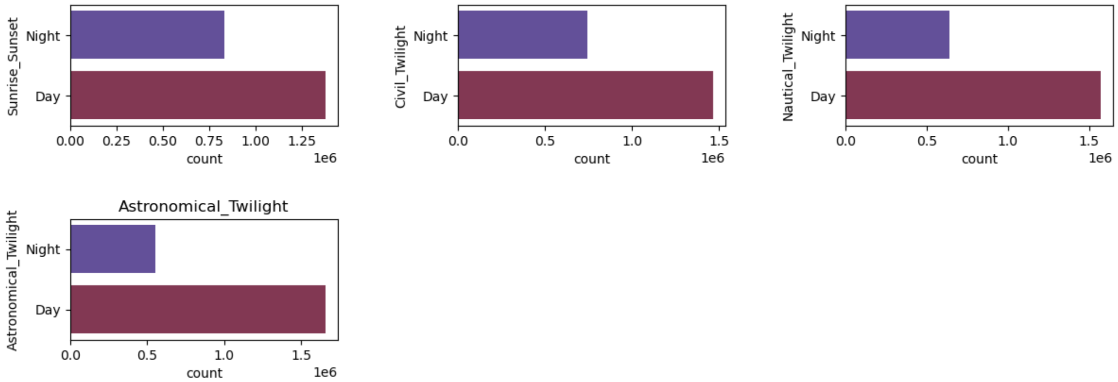


[24]:
```
# accidents by hour
# point out standard morning and night rush
h = sns.catplot(x='Hour',data=df, kind='count', height=8.27, aspect=11.7/8.27,␣
 ↪palette='crest')
h.fig.suptitle('Hourly Accidents Cases', y=1.03)
h.set(ylabel='Hourly Cases', xlabel='Hour')
plt.annotate('Morning Commute', xy=(7.5,0), xytext=(1, 70000),␣
 ↪arrowprops={'arrowstyle':'fancy'})
plt.annotate('After Work Commute', xy=(17.5,0), xytext=(19, 130000),␣
 ↪arrowprops={'arrowstyle':'fancy'})
plt.show()
```

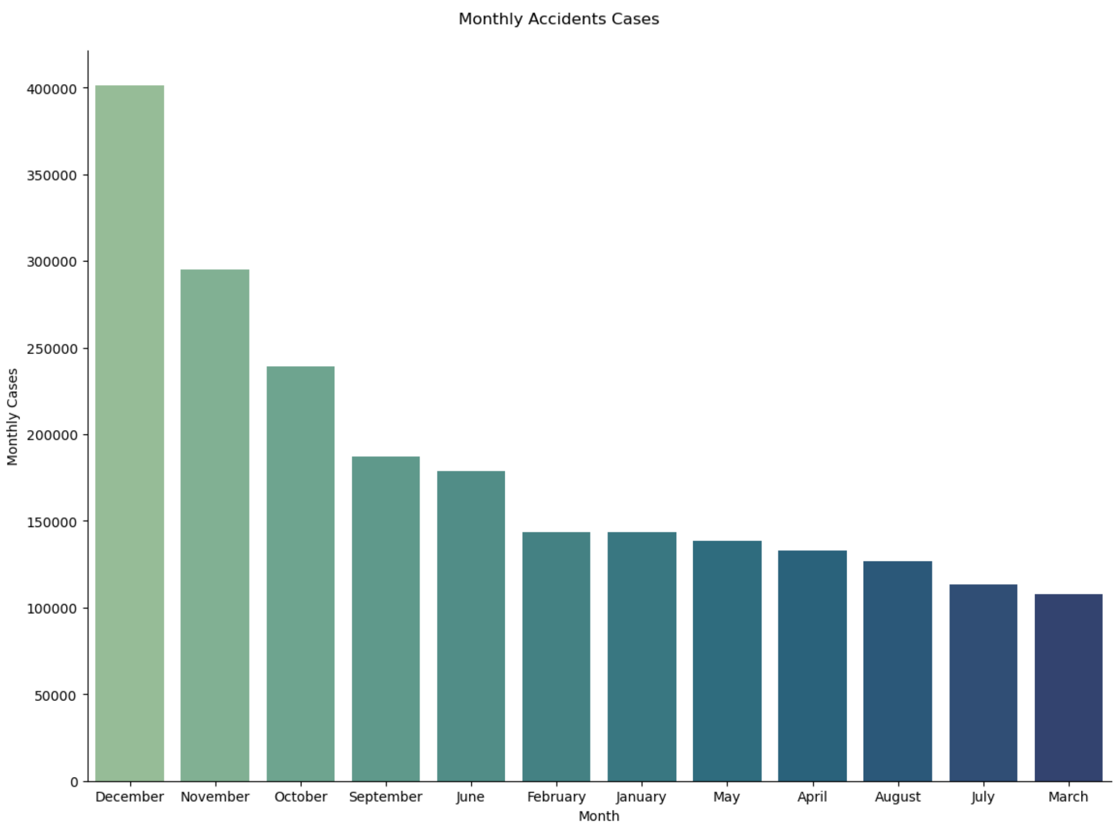


[25]:

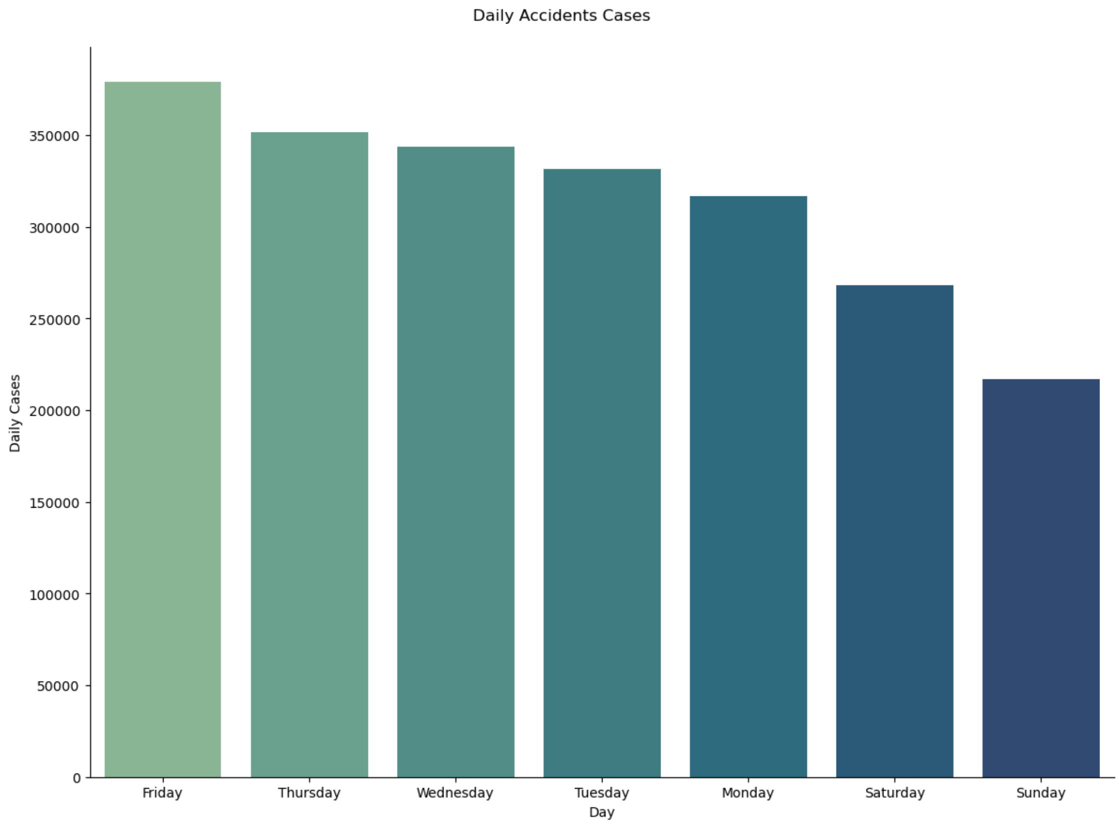
```
time_of_day =␣
↪df[['Sunrise_Sunset','Civil_Twilight','Nautical_Twilight','Astronomical_Twilight']]

fig = plt.figure(figsize = (12,22))
for i, c in enumerate(time_of_day):
    plt.subplot(10,3,i+1)
    ax = sns.countplot(y = c, data = df, palette='twilight')
    fig.tight_layout(h_pad=4, w_pad=4)

plt.title(c)
plt.show()
```



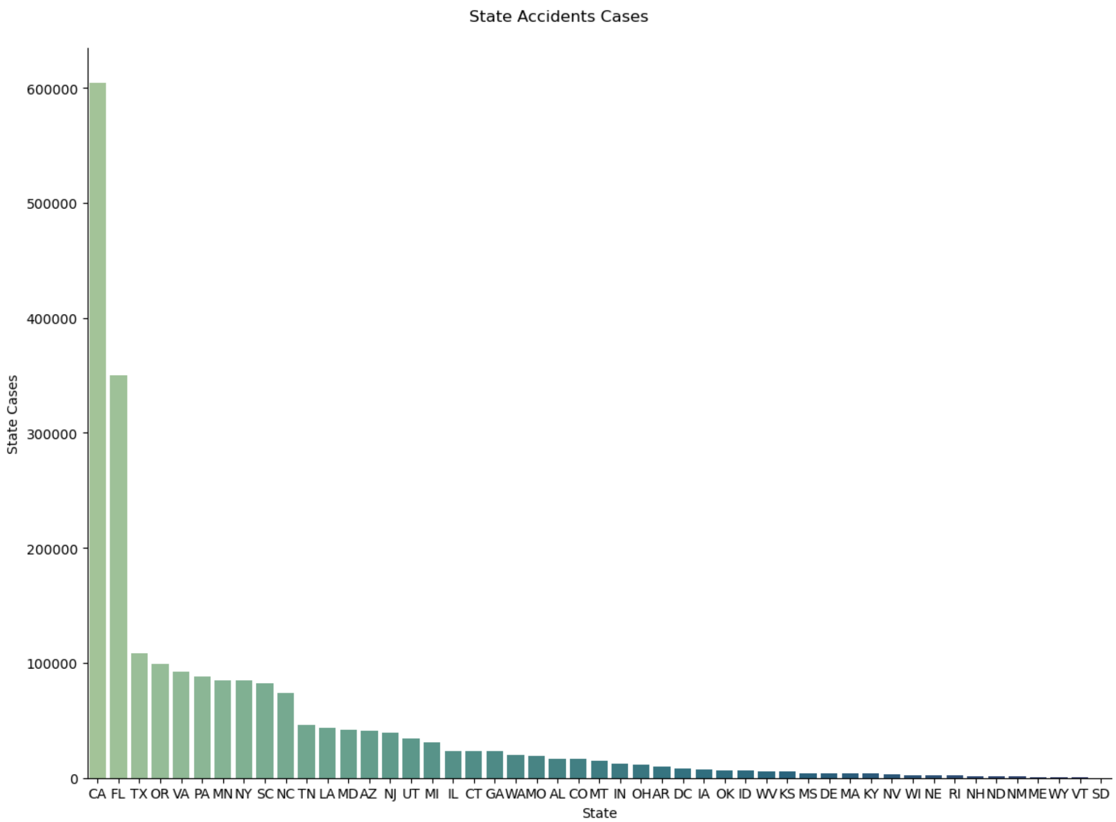
[26]:
```
# monthly accidents, decending
h = sns.catplot(x='Month',data=df, kind='count', order=df.Month.value_counts().
 ↪index, height=8.27, aspect=11.7/8.27, palette='crest')
h.fig.suptitle('Monthly Accidents Cases', y=1.03)
h.set(ylabel='Monthly Cases', xlabel='Month')
plt.show()
```



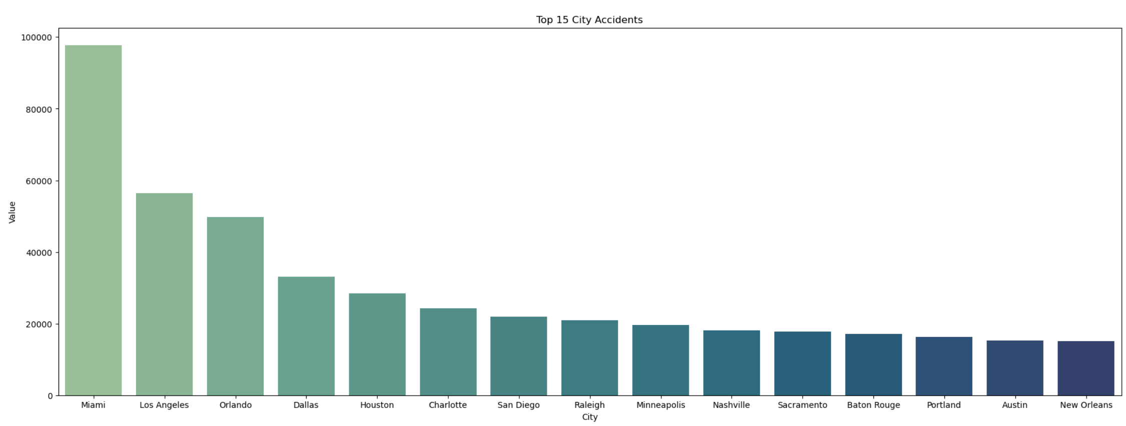
[27]:
```python
# daily accidents descending
h = sns.catplot(x='Day',data=df, kind='count', order=df.Day.value_counts().
 ↪index, height=8.27, aspect=11.7/8.27, palette='crest')
h.fig.suptitle('Daily Accidents Cases', y=1.03)
h.set(ylabel='Daily Cases', xlabel='Day')
plt.show()
```



[28]:
```python
# descending accidents by state
h = sns.catplot(x='State',data=df, kind='count', order=df.State.value_counts().
 ↪index, height=8.27, aspect=11.7/8.27, palette='crest')
h.fig.suptitle('State Accidents Cases', y=1.03)
h.set(ylabel='State Cases', xlabel='State')
plt.show()
```

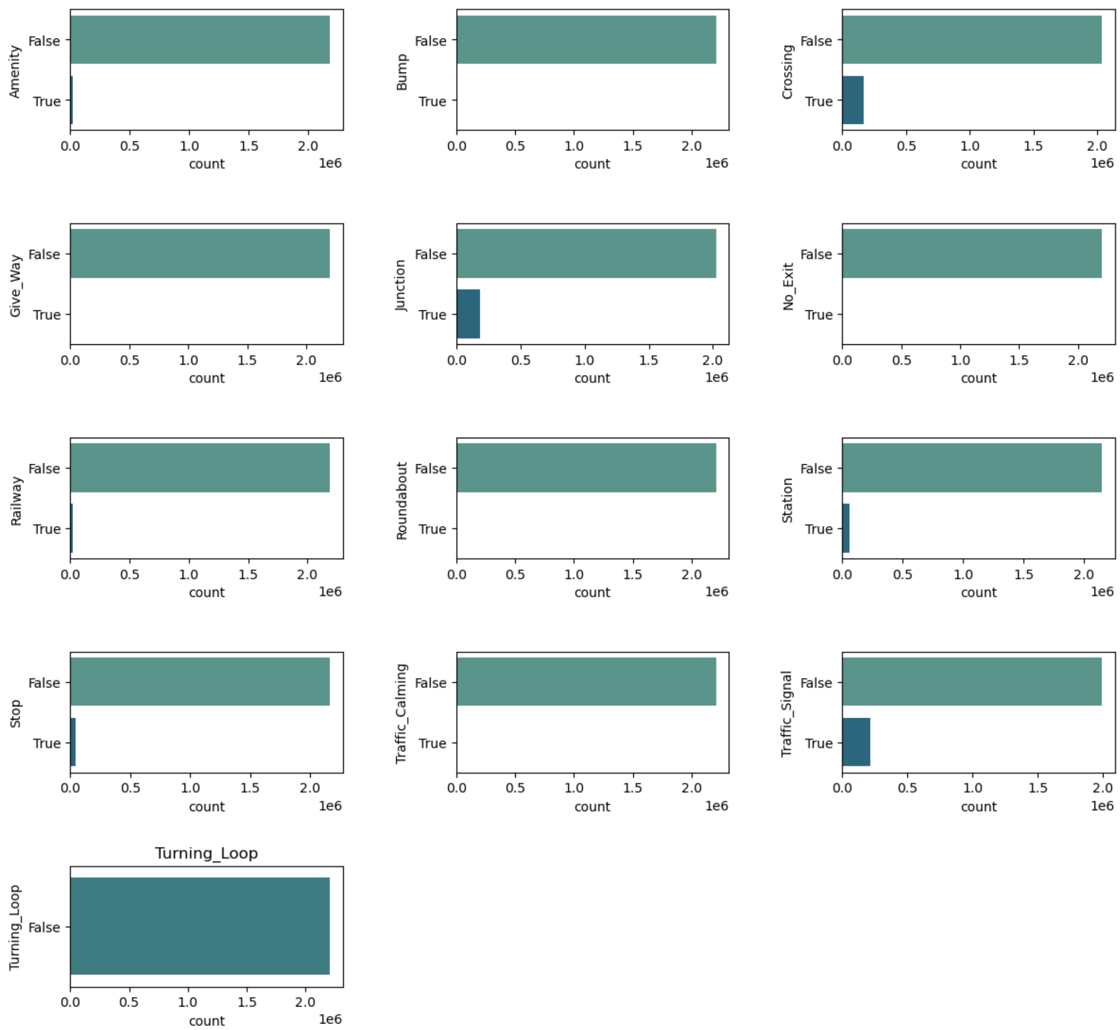



```
[29]: # descending accidents by city
counts = df["City"].value_counts()[:15]
plt.figure(figsize=(23,8))
sns.barplot(counts.index, counts.values, palette='crest')
plt.title("Top 15 City Accidents")
plt.xlabel("City")
plt.ylabel("Value")
plt.show()
```

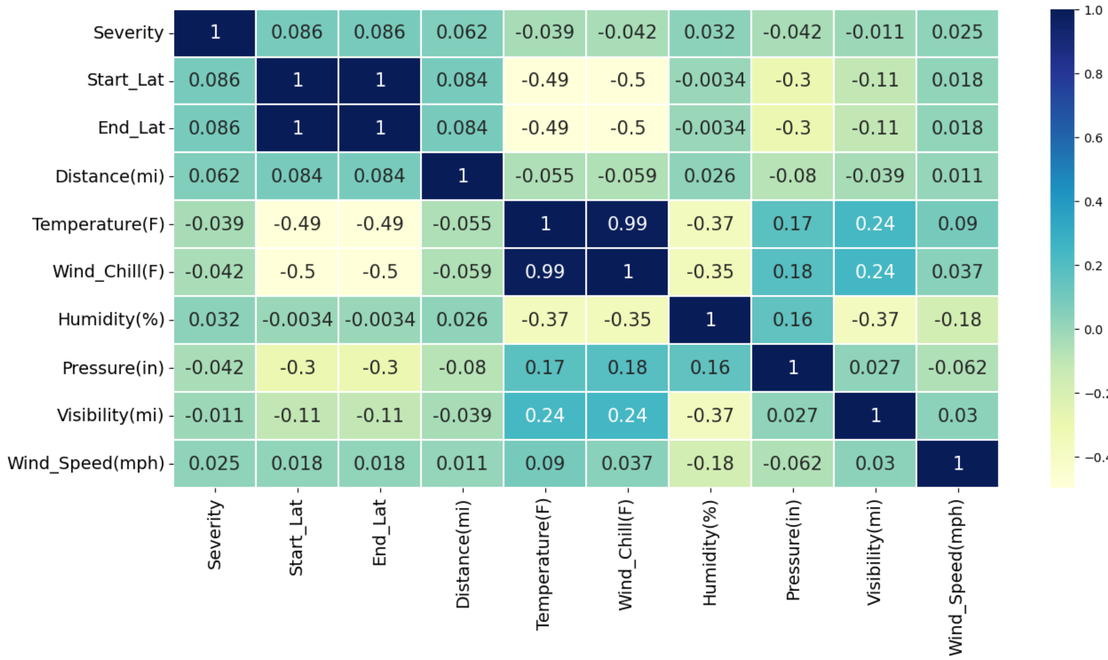


[30]:

```
road_conditions = df[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit',
                      'Railway','Roundabout','Station','Stop','Traffic_Calming',
                      'Traffic_Signal','Turning_Loop']]
fig = plt.figure(figsize = (12,22))
for i, c in enumerate(road_conditions):
    plt.subplot(10,3,i+1)
    ax = sns.countplot(y = c, data = df, palette='crest')
    fig.tight_layout(h_pad=4, w_pad=4)

plt.title(c)
plt.show()
```



[31]:
```
# heat map of accidents and severity by weather conditions
fig=sns.
  ↪heatmap(df[['Severity','Start_Lat','End_Lat','Distance(mi)','Temperature(F)','Wind_Chill(F)
  ↪corr(),annot=True,cmap='YlGnBu',linewidths=0.2,annot_kws={'size':15})
fig=plt.gcf()
fig.set_size_inches(15,7)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



	Severity	Start_Lat	End_Lat	Distance(mi)	Temperature(F)	Wind_Chill(F)	Humidity(%)	Pressure(in)	Visibility(mi)	Wind_Speed(mph)
Severity	1	0.086	0.086	0.062	-0.039	-0.042	0.032	-0.042	-0.011	0.025
Start_Lat	0.086	1	1	0.084	-0.49	-0.5	-0.0034	-0.3	-0.11	0.018
End_Lat	0.086	1	1	0.084	-0.49	-0.5	-0.0034	-0.3	-0.11	0.018
Distance(mi)	0.062	0.084	0.084	1	-0.055	-0.059	0.026	-0.08	-0.039	0.011
Temperature(F)	-0.039	-0.49	-0.49	-0.055	1	0.99	-0.37	0.17	0.24	0.09
Wind_Chill(F)	-0.042	-0.5	-0.5	-0.059	0.99	1	-0.35	0.18	0.24	0.037
Humidity(%)	0.032	-0.0034	-0.0034	0.026	-0.37	-0.35	1	0.16	-0.37	-0.18
Pressure(in)	-0.042	-0.3	-0.3	-0.08	0.17	0.18	0.16	1	0.027	-0.062
Visibility(mi)	-0.011	-0.11	-0.11	-0.039	0.24	0.24	-0.37	0.027	1	0.03
Wind_Speed(mph)	0.025	0.018	0.018	0.011	0.09	0.037	-0.18	-0.062	0.03	1

[32]:
```
# drop columns that aren't needed. Focused moslty on time of day and weather
df = df.drop(['ID','Start_Time', 'End_Time', 'Start_Lng', 'End_Lng',␣
  ↪'Start_Lat','End_Lat','Distance(mi)',
                ␣
  ↪'Description','Number','City','County','State','Zipcode','Country','Timezone',
                 'Airport_Code','Weather_Timestamp',
                 'Humidity(%)','Pressure(in)','Year','Month', 'Side'
                ],axis=1)
df.head()
```

[32]:
```
   Severity         Street  Temperature(F)  Wind_Chill(F)  Visibility(mi)  \
0         3  Local Streets            42.1           36.1            10.0
4         3        Highway            37.0           29.8            10.0
7         2        Highway            33.1           30.0             0.5
9         2  Local Streets            32.0           28.7             0.5
```

```
10         2       Highway            33.8           29.6             3.0

   Wind_Direction  Wind_Speed(mph)  Precipitation(in) Weather_Condition  \
0               S             10.4               0.00              Rain
4               W             10.4               0.01              Rain
7               S              3.5               0.08              Snow
9               W              3.5               0.05              Snow
10              N              4.6               0.03              Snow

    Amenity  ...  Stop  Traffic_Calming  Traffic_Signal  Turning_Loop  \
0     False  ...  False           False           False         False
4     False  ...  False           False           False         False
7     False  ...  False           False           False         False
9     False  ...  False           False           False         False
10    False  ...  False           False           False         False

    Sunrise_Sunset  Civil_Twilight  Nautical_Twilight  Astronomical_Twilight  \
0            Night           Night              Night                  Night
4              Day             Day                Day                    Day
7              Day             Day                Day                    Day
9              Day             Day                Day                    Day
10             Day             Day                Day                    Day

    Hour     Day
0      0  Monday
4      7  Monday
7     11  Monday
9     15  Monday
10    15  Monday

[5 rows x 28 columns]
```

[33]:

```python
# look at continuous types
print('Continuous Features')
df.select_dtypes(include='number').describe().T
```

```
Continuous Features
```

[33]:

```
                       count       mean        std   min   25%   50%   75%  \
Severity           2207325.0   2.074862   0.383241   1.0   2.0   2.0   2.0
Temperature(F)     2207325.0  61.838083  18.561719 -33.0  50.0  64.0  76.0
Wind_Chill(F)      2207325.0  60.716421  20.518191 -50.1  50.0  64.0  76.0
Visibility(mi)     2207325.0   9.046080   2.610565   0.0  10.0  10.0  10.0
Wind_Speed(mph)    2207325.0   7.152117   5.517968   0.0   3.0   7.0  10.0
Precipitation(in)  2207325.0   0.005705   0.058258   0.0   0.0   0.0   0.0
Hour               2207325.0  12.870165   5.927165   0.0   8.0  14.0  17.0
```

```
                         max
Severity                 4.0
Temperature(F)         196.0
Wind_Chill(F)          196.0
Visibility(mi)         100.0
Wind_Speed(mph)       1087.0
Precipitation(in)       24.0
Hour                    23.0
```

[34]:
```
# look at categorical types
print('Categorical Features')
df.select_dtypes(include='object').describe().T
```

```
Categorical Features
```

[34]:

	count	unique	top	freq
Street	2207325	2	Highway	1110765
Wind_Direction	2207325	6	S	560706
Weather_Condition	2207325	7	Clear	1057044
Sunrise_Sunset	2207325	2	Day	1374753
Civil_Twilight	2207325	2	Day	1463403
Nautical_Twilight	2207325	2	Day	1567600
Astronomical_Twilight	2207325	2	Day	1656262
Day	2207325	7	Friday	379067

5 Preprocess Data

[35]:
```
# copy for model
df_model = df.copy()
```

[36]:
```
# Boolean columns
bin_cols = ['Amenity', 'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit',␣
 ↪'Railway', 'Roundabout',
            'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal',␣
 ↪'Turning_Loop']


# change to objects to be split for encoding
for c in bin_cols:
    df_model[c] = df_model[c].astype('object')
```

[37]:
```
df_model = pd.get_dummies(df_model, drop_first=True)
```

[38]:
```
# only care about critical (4) severe accidents
def label(i):
    return 1 if i == 4 else 0
```

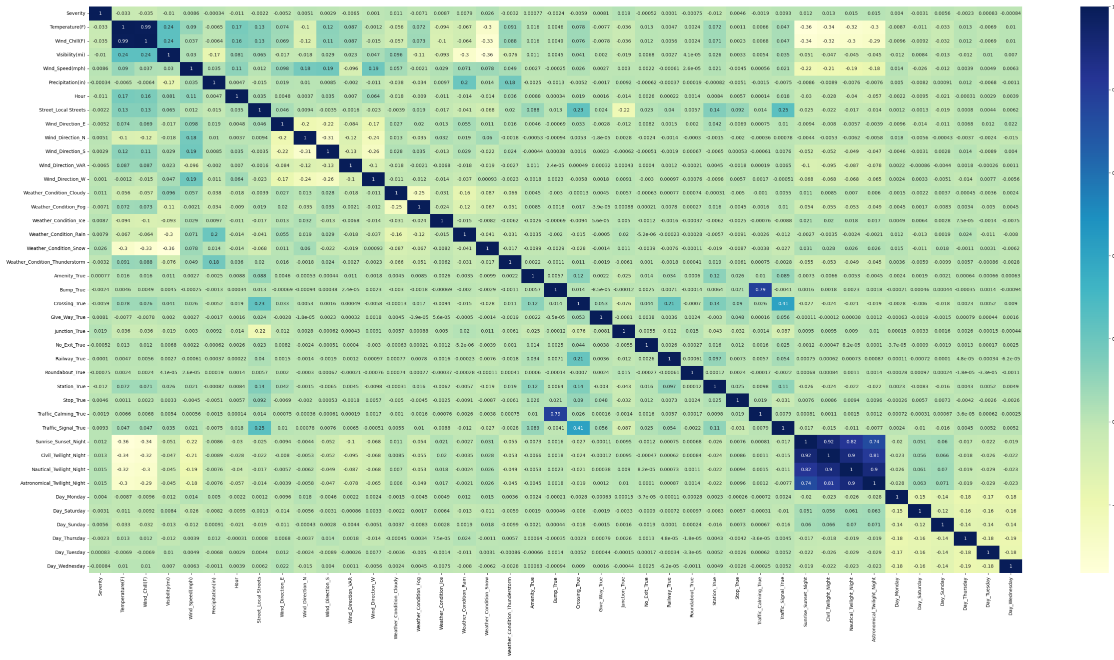
```
df_model['Severity'] = df_model.Severity.apply(label)
```

[39]:
```
# drop any null, just in case
df_model = df_model.dropna()

# new encoded data
df_model.head()
```

[39]:
```
    Severity  Temperature(F)  Wind_Chill(F)  Visibility(mi)  Wind_Speed(mph)  \
0          0            42.1           36.1            10.0             10.4
4          0            37.0           29.8            10.0             10.4
7          0            33.1           30.0             0.5              3.5
9          0            32.0           28.7             0.5              3.5
10         0            33.8           29.6             3.0              4.6

    Precipitation(in)  Hour  Street_Local Streets  Wind_Direction_E  \
0                0.00     0                     1                 0
4                0.01     7                     0                 0
7                0.08    11                     0                 0
9                0.05    15                     1                 0
10               0.03    15                     0                 0

    Wind_Direction_N  …  Sunrise_Sunset_Night  Civil_Twilight_Night  \
0                  0  …                     1                     1
4                  0  …                     0                     0
7                  0  …                     0                     0
9                  0  …                     0                     0
10                 1  …                     0                     0

    Nautical_Twilight_Night  Astronomical_Twilight_Night  Day_Monday  \
0                         1                            1           1
4                         0                            0           1
7                         0                            0           1
9                         0                            0           1
10                        0                            0           1

    Day_Saturday  Day_Sunday  Day_Thursday  Day_Tuesday  Day_Wednesday
0              0           0             0            0              0
4              0           0             0            0              0
7              0           0             0            0              0
9              0           0             0            0              0
10             0           0             0            0              0

[5 rows x 41 columns]
```

[40]:
```
# show correlation of all values
plt.figure(figsize=(45,22))
```

```
sns.heatmap(df_model.corr(),annot=True,cmap='YlGnBu')
plt.show()
```



[41]:
```
# to confirm everything is encoded before analysis
df_model.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2207325 entries, 0 to 2845341
Data columns (total 41 columns):
 #   Column                         Dtype
---  ------                         -----
 0   Severity                       int64
 1   Temperature(F)                 float64
 2   Wind_Chill(F)                  float64
 3   Visibility(mi)                 float64
 4   Wind_Speed(mph)                float64
 5   Precipitation(in)              float64
 6   Hour                           int64
 7   Street_Local Streets           uint8
 8   Wind_Direction_E               uint8
 9   Wind_Direction_N               uint8
 10  Wind_Direction_S               uint8
 11  Wind_Direction_VAR             uint8
 12  Wind_Direction_W               uint8
 13  Weather_Condition_Cloudy       uint8
 14  Weather_Condition_Fog          uint8
```

```
 15   Weather_Condition_Ice            uint8
 16   Weather_Condition_Rain           uint8
 17   Weather_Condition_Snow           uint8
 18   Weather_Condition_Thunderstorm   uint8
 19   Amenity_True                     uint8
 20   Bump_True                        uint8
 21   Crossing_True                    uint8
 22   Give_Way_True                    uint8
 23   Junction_True                    uint8
 24   No_Exit_True                     uint8
 25   Railway_True                     uint8
 26   Roundabout_True                  uint8
 27   Station_True                     uint8
 28   Stop_True                        uint8
 29   Traffic_Calming_True             uint8
 30   Traffic_Signal_True              uint8
 31   Sunrise_Sunset_Night             uint8
 32   Civil_Twilight_Night             uint8
 33   Nautical_Twilight_Night          uint8
 34   Astronomical_Twilight_Night      uint8
 35   Day_Monday                       uint8
 36   Day_Saturday                     uint8
 37   Day_Sunday                       uint8
 38   Day_Thursday                     uint8
 39   Day_Tuesday                      uint8
 40   Day_Wednesday                    uint8
dtypes: float64(5), int64(2), uint8(34)
memory usage: 206.3 MB
```

[42]:
```
# target varaible
y = df_model['Severity']
#X variables
X = df_model.drop(['Severity'], axis = 1)
```

[43]:
```
# to balance the data
os = SMOTE(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
columns = X_train.columns
os_data_X,os_data_y=os.fit_resample(X_train, y_train)
os_data_X = pd.DataFrame(data=os_data_X,columns=columns )
os_data_y= pd.DataFrame(data=os_data_y,columns=['Severity'])

# Check the numbers of  data to make sure balanced
print("Length of Oversampled Data: ",len(os_data_X))
print("Number of No Accidents in Oversampled Data:
 ↪",len(os_data_y[os_data_y['Severity']==0]))
```

```python
print("Number of Accidents",len(os_data_y[os_data_y['Severity']==1]))
print("Proportion of No Severe Accidents in Oversampled Data:
 ↪",len(os_data_y[os_data_y['Severity']==0])/len(os_data_X))
print("Proportion of Severe Accidents in Oversampled Data:
 ↪",len(os_data_y[os_data_y['Severity']==1])/len(os_data_X))
```

```
Length of Oversampled Data:  3003352
Number of No Accidents in Oversampled Data: 1501676
Number of Accidents 1501676
Proportion of No Severe Accidents in Oversampled Data: 0.5
Proportion of Severe Accidents in Oversampled Data: 0.5
```

[44]:
```python
# to search for most valuable features
df_vars=df_model.columns.values.tolist()
y=['Severity']
X=[i for i in df_vars if i not in y]


logreg = LogisticRegression()

# 42 total variables
# select 30 variables important to model and then add them to an array,␣
 ↪predictors
rfe = RFE(logreg, n_features_to_select=30)
rfe = rfe.fit(os_data_X, os_data_y.values.ravel())
predictors=[]
print('The following predictors are selected:')
for i in range(os_data_X.shape[1]):
    if rfe.support_[i] == True:
        predictors.append(os_data_X.columns[i])
        print('Column: %d, Rank: %.3f, Feature %s' % (i, rfe.ranking_[i],␣
 ↪os_data_X.columns[i]))
```

```
The following predictors are selected:
Column: 0, Rank: 1.000, Feature Temperature(F)
Column: 1, Rank: 1.000, Feature Wind_Chill(F)
Column: 3, Rank: 1.000, Feature Wind_Speed(mph)
Column: 6, Rank: 1.000, Feature Street_Local Streets
Column: 7, Rank: 1.000, Feature Wind_Direction_E
Column: 8, Rank: 1.000, Feature Wind_Direction_N
Column: 9, Rank: 1.000, Feature Wind_Direction_S
Column: 10, Rank: 1.000, Feature Wind_Direction_VAR
Column: 11, Rank: 1.000, Feature Wind_Direction_W
Column: 12, Rank: 1.000, Feature Weather_Condition_Cloudy
Column: 13, Rank: 1.000, Feature Weather_Condition_Fog
Column: 14, Rank: 1.000, Feature Weather_Condition_Ice
Column: 15, Rank: 1.000, Feature Weather_Condition_Rain
```

```
Column: 17, Rank: 1.000, Feature Weather_Condition_Thunderstorm
Column: 18, Rank: 1.000, Feature Amenity_True
Column: 20, Rank: 1.000, Feature Crossing_True
Column: 21, Rank: 1.000, Feature Give_Way_True
Column: 22, Rank: 1.000, Feature Junction_True
Column: 24, Rank: 1.000, Feature Railway_True
Column: 26, Rank: 1.000, Feature Station_True
Column: 27, Rank: 1.000, Feature Stop_True
Column: 29, Rank: 1.000, Feature Traffic_Signal_True
Column: 30, Rank: 1.000, Feature Sunrise_Sunset_Night
Column: 32, Rank: 1.000, Feature Nautical_Twilight_Night
Column: 34, Rank: 1.000, Feature Day_Monday
Column: 35, Rank: 1.000, Feature Day_Saturday
Column: 36, Rank: 1.000, Feature Day_Sunday
Column: 37, Rank: 1.000, Feature Day_Thursday
Column: 38, Rank: 1.000, Feature Day_Tuesday
Column: 39, Rank: 1.000, Feature Day_Wednesday
```

[45]:
```
# Change X to most important columns
# Add constant
# Y is ReAdmis
X = os_data_X[predictors]
y = os_data_y['Severity']

frames = [y, X]
df_model = pd.concat(frames, axis = 1)
```

[46]:
```
#normalize data. Range of 0 to 1 to help with analysis
normalized_data = (df_model-df_model.min())/(df_model.max()-df_model.min())



# split X and y variables
X = normalized_data.drop(['Severity'], axis=1)
y = normalized_data['Severity']
Xc = sm.add_constant(X,1)
```

6 Analysis

[47]:
```
# run model
model = sm.Logit(y,Xc)
result = model.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.478087
         Iterations 7
                           Logit Regression Results
```

```
==============================================================================
Dep. Variable:               Severity   No. Observations:              3003352
Model:                          Logit   Df Residuals:                  3003321
Method:                           MLE   Df Model:                           30
Date:                Sun, 06 Nov 2022   Pseudo R-squ.:                  0.3103
Time:                        13:21:39   Log-Likelihood:            -1.4359e+06
converged:                       True   LL-Null:                   -2.0818e+06
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
==================
                                  coef    std err          z      P>|z|
[0.025      0.975]
--------------------------------------------------------------------------------
------------------
const                           3.3966      0.011    317.532      0.000
3.376       3.418
Temperature(F)                  6.9915      0.187     37.300      0.000
6.624       7.359
Wind_Chill(F)                  -8.0262      0.180    -44.486      0.000
-8.380      -7.673
Wind_Speed(mph)                87.8033      0.408    215.058      0.000
87.003      88.604
Street_Local Streets           -0.4501      0.003   -145.120      0.000
-0.456      -0.444
Wind_Direction_E               -2.0222      0.006   -335.640      0.000
-2.034      -2.010
Wind_Direction_N               -1.8372      0.005   -379.772      0.000
-1.847      -1.828
Wind_Direction_S               -1.5900      0.005   -348.213      0.000
-1.599      -1.581
Wind_Direction_VAR             -1.8892      0.009   -216.191      0.000
-1.906      -1.872
Wind_Direction_W               -2.0630      0.005   -381.756      0.000
-2.074      -2.052
Weather_Condition_Cloudy       -0.7576      0.004   -202.229      0.000
-0.765      -0.750
Weather_Condition_Fog          -1.1242      0.005   -228.583      0.000
-1.134      -1.115
Weather_Condition_Ice          -1.2340      0.032    -38.057      0.000
-1.298      -1.170
Weather_Condition_Rain         -0.9780      0.007   -143.416      0.000
-0.991      -0.965
Weather_Condition_Thunderstorm    -2.1518      0.024    -91.154      0.000
-2.198      -2.106
Amenity_True                   -0.9813      0.027    -36.607      0.000
-1.034      -0.929
Crossing_True                  -0.7180      0.009    -80.285      0.000
-0.735      -0.700
```

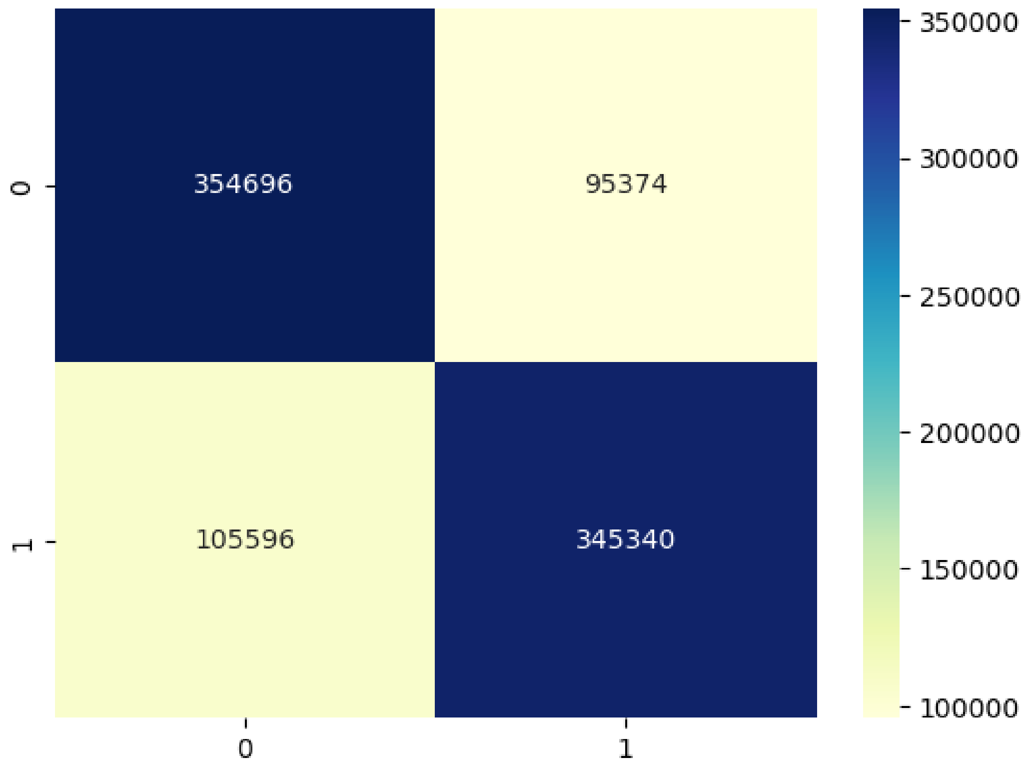
```
Give_Way_True                              -1.0891      0.055    -19.662      0.000
-1.198      -0.981
Junction_True                              -1.2052      0.007   -175.539      0.000
-1.219      -1.192
Railway_True                               -1.1273      0.034    -33.170      0.000
-1.194      -1.061
Station_True                               -1.6166      0.020    -82.887      0.000
-1.655      -1.578
Stop_True                                  -1.4002      0.018    -75.855      0.000
-1.436      -1.364
Traffic_Signal_True                        -0.4566      0.007    -65.423      0.000
-0.470      -0.443
Sunrise_Sunset_Night                       -0.3146      0.006    -55.973      0.000
-0.326      -0.304
Nautical_Twilight_Night                     0.1741      0.006     29.782      0.000
0.163       0.186
Day_Monday                                 -1.9591      0.005   -398.540      0.000
-1.969      -1.949
Day_Saturday                               -2.2379      0.006   -401.308      0.000
-2.249      -2.227
Day_Sunday                                 -2.1427      0.006   -361.735      0.000
-2.154      -2.131
Day_Thursday                               -2.0734      0.005   -426.713      0.000
-2.083      -2.064
Day_Tuesday                                -2.0293      0.005   -415.280      0.000
-2.039      -2.020
Day_Wednesday                              -2.0610      0.005   -422.624      0.000
-2.071      -2.051
================================================================================
==================
```

[48]:

```
# Run a test against the predicted outcome versus true outcome
X_train, X_test, y_train, y_test = train_test_split(Xc, y, test_size=0.3,␣
↪random_state=0)
lgr = LogisticRegression()
lgr.fit(X_train, y_train)
predicted = lgr.predict(X_test)
expected = y_test

# Confusion matrix to show accuracy of test
matrix = pd.DataFrame(confusion_matrix(y_true=expected, y_pred=predicted),
                     index=range(2), columns=range(2))
axes = sns.heatmap(matrix, annot=True, cmap='YlGnBu', fmt='g')


correct = sum(np.diagonal(matrix))
total = matrix.values.sum()
```

```
incorrect = total - correct

print('Correct predictions: {} ({:.0%})'.format(correct, correct/total))
print('Incorrect predictions: {} ({:.0%})'.format(incorrect, incorrect/total))
```

```
Correct predictions: 700036 (78%)
Incorrect predictions: 200970 (22%)
```



[49]:
```
# Classification report to show accuracy of model
print(classification_report(y_test, predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.77      0.79      0.78    450070
         1.0       0.78      0.77      0.77    450936

    accuracy                           0.78    901006
   macro avg       0.78      0.78      0.78    901006
weighted avg       0.78      0.78      0.78    901006
```

[50]:
```
# ROC to show accuracy of model
logit_roc_auc = roc_auc_score(y_test, lgr.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, lgr.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (AUC = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
print('AUC Score:',round(logit_roc_auc*100,0),'%')
plt.show()
```

```
AUC Score: 78.0 %
```