US Accidents - Exploratory Data Analysis

Import essential libraries

In [67]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import folium
```

Data Preparation and Cleaning

Loading File Using Pandas

In [68]:
```
df = pd.read_csv('us_accidents.csv')
```

In [69]:
```
df.head()
```

Out[69]:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distanc
0	A-1	Source2	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	NaN	NaN	
1	A-2	Source2	2	2016-02-08 06:07:59	2016-02-08 06:37:59	39.928059	-82.831184	NaN	NaN	
2	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	NaN	NaN	
3	A-4	Source2	3	2016-02-08 07:23:34	2016-02-08 07:53:34	39.747753	-84.205582	NaN	NaN	
4	A-5	Source2	2	2016-02-08 07:39:07	2016-02-08 08:09:07	39.627781	-84.188354	NaN	NaN	

5 rows × 46 columns

Look at some basic information about the data & the columns

In [70]:
```
df.columns
```

Out[70]:
```
Index(['ID', 'Source', 'Severity', 'Start_Time', 'End_Time', 'Start_Lat',
       'Start_Lng', 'End_Lat', 'End_Lng', 'Distance(mi)', 'Description',
       'Street', 'City', 'County', 'State', 'Zipcode', 'Country', 'Timezone',
       'Airport_Code', 'Weather_Timestamp', 'Temperature(F)', 'Wind_Chill(F)',
       'Humidity(%)', 'Pressure(in)', 'Visibility(mi)', 'Wind_Direction',
       'Wind_Speed(mph)', 'Precipitation(in)', 'Weather_Condition', 'Amenity',
       'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway',
       'Roundabout', 'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal',
       'Turning_Loop', 'Sunrise_Sunset', 'Civil_Twilight', 'Nautical_Twilight',
       'Astronomical_Twilight'],
      dtype='object')
```

In [71]:
```
len(df.columns)
```

Out[71]:
```
46
```

In [72]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000000 entries, 0 to 2999999
Data columns (total 46 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   ID                     object 
 1   Source                 object 
 2   Severity               int64  
 3   Start_Time             object 
 4   End_Time               object 
 5   Start_Lat              float64
 6   Start_Lng              float64
 7   End_Lat                float64
 8   End_Lng                float64
 9   Distance(mi)           float64
 10  Description            object 
 11  Street                 object 
 12  City                   object 
 13  County                 object 
 14  State                  object 
 15  Zipcode                object 
 16  Country                object 
 17  Timezone               object 
 18  Airport_Code           object 
 19  Weather_Timestamp      object 
 20  Temperature(F)         float64
 21  Wind_Chill(F)          float64
 22  Humidity(%)            float64
 23  Pressure(in)           float64
 24  Visibility(mi)         float64
 25  Wind_Direction         object 
 26  Wind_Speed(mph)        float64
 27  Precipitation(in)      float64
 28  Weather_Condition      object 
 29  Amenity                bool   
 30  Bump                   bool   
 31  Crossing               bool   
 32  Give_Way               bool   
 33  Junction               bool   
 34  No_Exit                bool   
 35  Railway                bool   
 36  Roundabout             bool   
 37  Station                bool   
 38  Stop                   bool   
 39  Traffic_Calming        bool   
 40  Traffic_Signal         bool   
 41  Turning_Loop           bool   
 42  Sunrise_Sunset         object 
 43  Civil_Twilight         object 
 44  Nautical_Twilight      object 
 45  Astronomical_Twilight  object 
dtypes: bool(13), float64(12), int64(1), object(20)
memory usage: 792.5+ MB
```

In [73]:
```
# describe() is used to get the total statistical analysis of all the columns;
df.describe()
```

Out[73]:

	Severity	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	Temperature(
count	3.000000e+06	3.000000e+06	3.000000e+06	0.0	0.0	3.000000e+06	2.950706e+(
mean	2.327517e+00	3.609880e+01	-9.346988e+01	NaN	NaN	2.165676e-01	6.261110e+(
std	5.056328e-01	4.803971e+00	1.639142e+01	NaN	NaN	1.658924e+00	1.839671e+(
min	1.000000e+00	2.455480e+01	-1.245344e+02	NaN	NaN	0.000000e+00	-8.900000e+(
25%	2.000000e+00	3.323096e+01	-1.108751e+02	NaN	NaN	0.000000e+00	5.050000e+(
50%	2.000000e+00	3.539112e+01	-8.727015e+01	NaN	NaN	0.000000e+00	6.490000e+(
75%	3.000000e+00	3.997931e+01	-8.084516e+01	NaN	NaN	0.000000e+00	7.600000e+(
max	4.000000e+00	4.900220e+01	-6.755331e+01	NaN	NaN	4.417500e+02	2.030000e+(

In [74]:

```
# Checking the number of numerical columns present in our dataset
numerics = ['int16' , 'int32' , 'int64', 'float16', 'float32', 'float64']

numeric_df = df.select_dtypes(include = numerics)
print(str(len(numeric_df.columns)) + ' numeric columns')
print()
numeric_df.columns
```

```
13 numeric columns
```

Out[74]:

```
Index(['Severity', 'Start_Lat', 'Start_Lng', 'End_Lat', 'End_Lng',
       'Distance(mi)', 'Temperature(F)', 'Wind_Chill(F)', 'Humidity(%)',
       'Pressure(in)', 'Visibility(mi)', 'Wind_Speed(mph)',
       'Precipitation(in)'],
      dtype='object')
```

Fix Any Missing or Incorrect values

In [75]:

```
# missing values
# total count of columns in the DataFrame df that have at least one missing value.
df.isna().any().sum()
```

Out[75]:

```
22
```

In [76]:

```
df.isna().sum().sort_values(ascending = False)
```

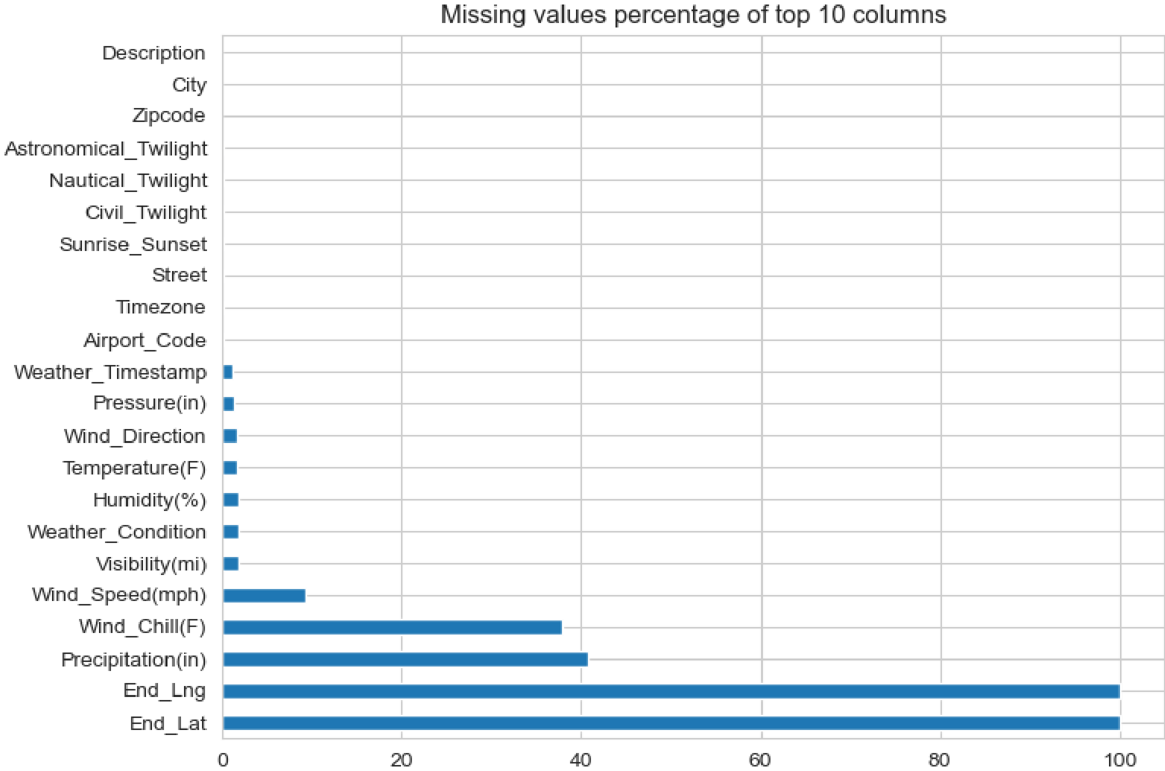
Out[76]:
```
End_Lat                  3000000
End_Lng                  3000000
Precipitation(in)        1225273
Wind_Chill(F)            1134852
Wind_Speed(mph)           278651
Visibility(mi)             56015
Weather_Condition          55200
Humidity(%)                52968
Temperature(F)             49294
Wind_Direction             48501
Pressure(in)               41365
Weather_Timestamp          34172
Airport_Code                5631
Timezone                    2380
Street                      1712
Sunrise_Sunset              1669
Civil_Twilight              1669
Nautical_Twilight           1669
Astronomical_Twilight       1669
Zipcode                      412
City                          56
Description                    5
Country                        0
No_Exit                        0
Severity                       0
Start_Time                     0
End_Time                       0
Turning_Loop                   0
Traffic_Signal                 0
Traffic_Calming                0
Stop                           0
Station                        0
Roundabout                     0
Railway                        0
Give_Way                       0
Junction                       0
Crossing                       0
Bump                           0
Amenity                        0
Start_Lat                      0
Start_Lng                      0
Distance(mi)                   0
Source                         0
County                         0
State                          0
ID                             0
dtype: int64
```

In [77]:
```
# Top 10 clolumns with highest percentage of missing values;
missing_percent = df.isna().sum().sort_values(ascending = False) / len(df) *100
missing_percent[ : 10]
```

Out[77]:
```
End_Lat              100.000000
End_Lng              100.000000
Precipitation(in)     40.842433
Wind_Chill(F)         37.828400
Wind_Speed(mph)        9.288367
Visibility(mi)         1.867167
Weather_Condition      1.840000
Humidity(%)            1.765600
Temperature(F)         1.643133
Wind_Direction         1.616700
dtype: float64
```

In [80]:
```
sns.set_style('whitegrid')
missing_percent[missing_percent != 0].plot(kind = 'barh' , figsize = (8,6))
plt.title("Missing values percentage of top 10 columns")
```

Out[80]:
```
Text(0.5, 1.0, 'Missing values percentage of top 10 columns')
```



In [83]:
```
# Remove columns that have more than 50 percentage of missing values or that are no
```

In [84]:
```
df.drop(columns = ['End_Lng' , 'End_Lat'] , axis = 1 , inplace = True)
```

In [85]:
```
df.columns
```

Out[85]:
```
Index(['ID', 'Source', 'Severity', 'Start_Time', 'End_Time', 'Start_Lat',
       'Start_Lng', 'Distance(mi)', 'Description', 'Street', 'City', 'County',
       'State', 'Zipcode', 'Country', 'Timezone', 'Airport_Code',
       'Weather_Timestamp', 'Temperature(F)', 'Wind_Chill(F)', 'Humidity(%)',
       'Pressure(in)', 'Visibility(mi)', 'Wind_Direction', 'Wind_Speed(mph)',
       'Precipitation(in)', 'Weather_Condition', 'Amenity', 'Bump', 'Crossing',
       'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station',
       'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop',
       'Sunrise_Sunset', 'Civil_Twilight', 'Nautical_Twilight',
       'Astronomical_Twilight'],
      dtype='object')
```

Now impute the missing values

To handle missing data missing or null values in numerical columns of a dataset are filled with appropriate replacement values. Missing values of numerical columns can be filled by mean or median.

In [87]:
```
# Impute missing values for necessary numerical columns:
df["Temperature(F)"] = df["Temperature(F)"].fillna(df["Temperature(F)"].median())
df["Humidity(%)"]=df["Humidity(%)"].fillna(df["Humidity(%)"].median())
```

Missing data of categorical columns can be filled by using mode of that column.

In [88]:
```
# Impute missing values for categorical data:
df["Weather_Condition"]=df["Weather_Condition"].fillna(df["Weather_Condition"].mode
```

In [89]:
```
# Now our data is clean;
```

Exploratory Analysis And Visualization

Columns to be analysed: City Start_Time Start_Lat and Start_Lng Temperature Weather_Condition and severity

City

In [90]:
```
cities = len(df['City'].unique())
```

In [93]:
```
print(f'There are total of {cities} number of cities.')
```

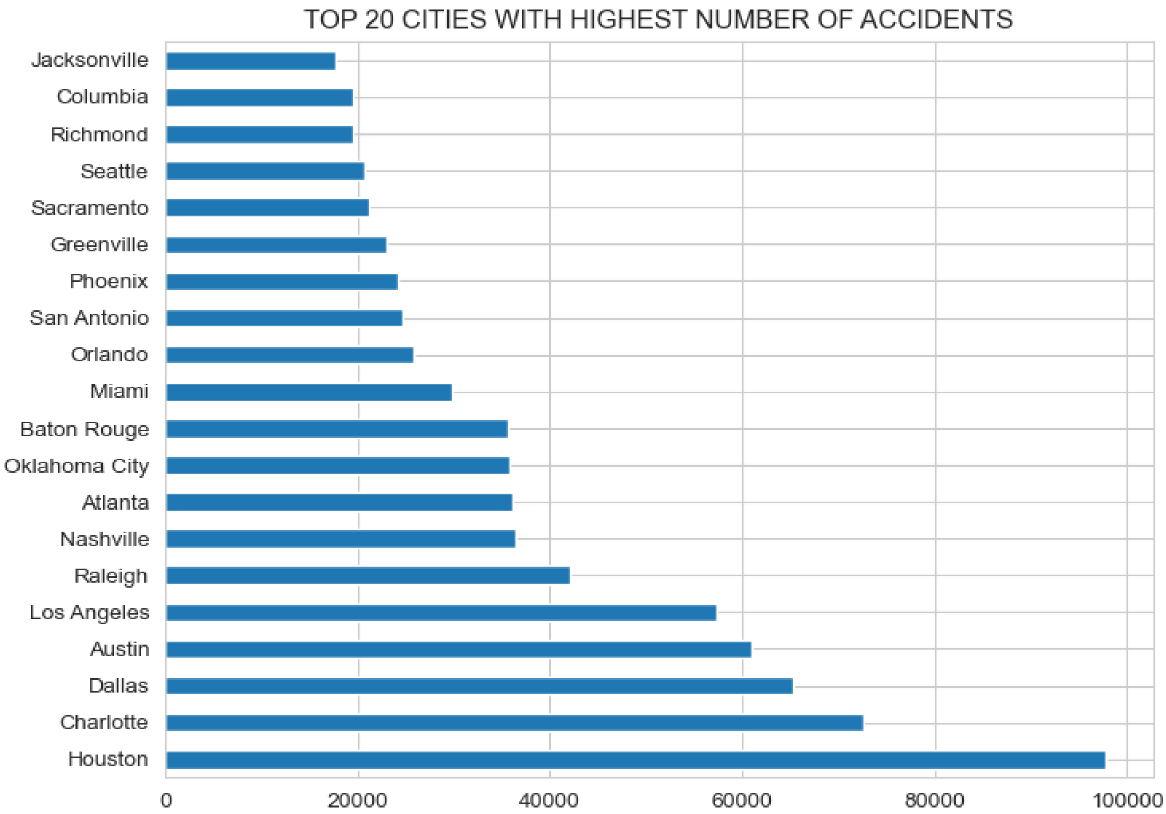
```
There are total of 11086 number of cities.
```

In [94]:
```
# Lets check the cities by accidents
cities_by_accidents = df['City'].value_counts()
```

In [95]:
```
cities_by_accidents_20 = cities_by_accidents[ : 20]
```

In [97]:
```
cities_by_accidents_20.plot(kind = 'barh' , figsize = (8,6))
plt.title("TOP 20 CITIES WITH HIGHEST NUMBER OF ACCIDENTS")
```

Out[97]:
```
Text(0.5, 1.0, 'TOP 20 CITIES WITH HIGHEST NUMBER OF ACCIDENTS')
```



In [98]:
```
# Lets find out the cities with highest and lowest number of accidents
```

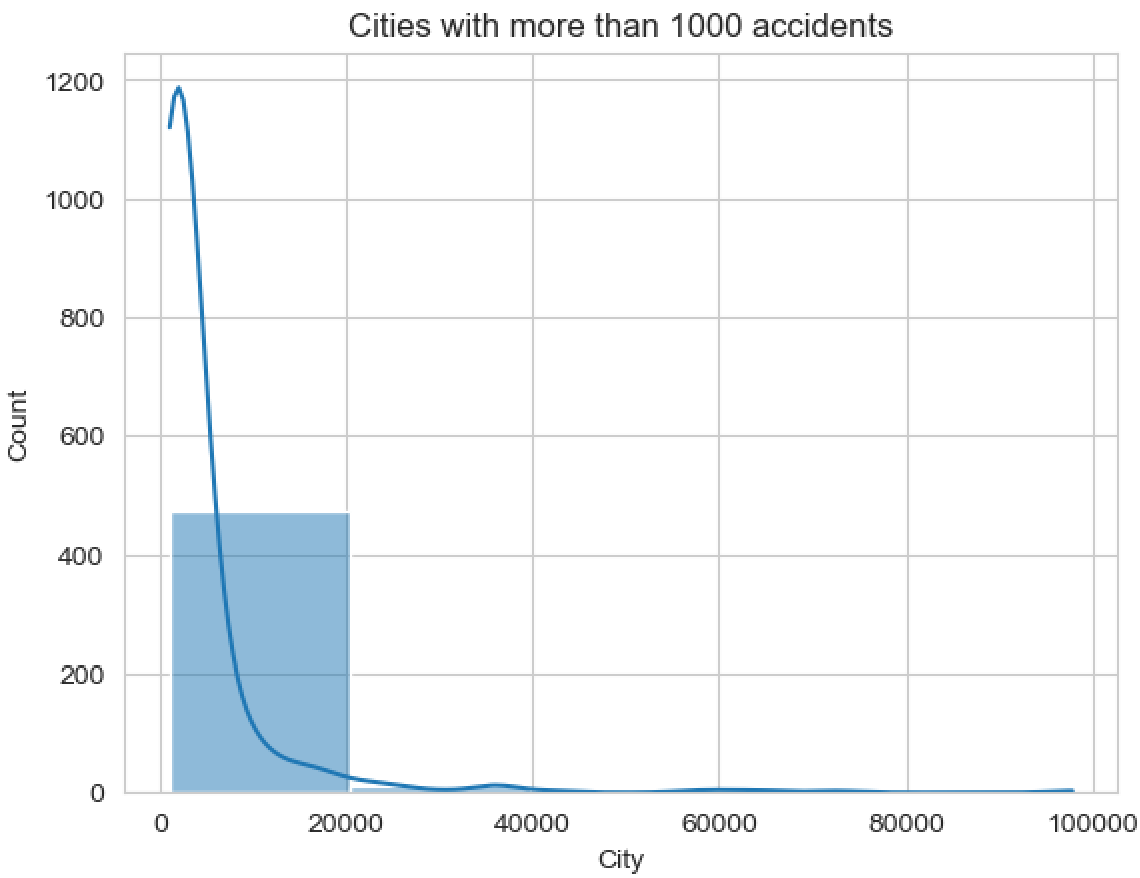
In [99]:
```
high_accident_cities = cities_by_accidents[cities_by_accidents > 1000]
low_accident_cities = cities_by_accidents[cities_by_accidents < 1000]
```

In [103…
```
print("Number of cities with more than 1000 accidents: " + str(len(high_accident_c
print('percentage :' + str(len(high_accident_cities) / cities * 100))
```

```
Number of cities with more than 1000 accidents: 490
percentage :4.4199891755367124
```

In [104…
```
sns.histplot(high_accident_cities , kde = True , bins = 5 )
plt.title('Cities with more than 1000 accidents')
```

Out[104]:
```
Text(0.5, 1.0, 'Cities with more than 1000 accidents')
```



In [107…
```
print("Number of cities with less than 1000 accidents: "+ str(len(low_accident_citi
```

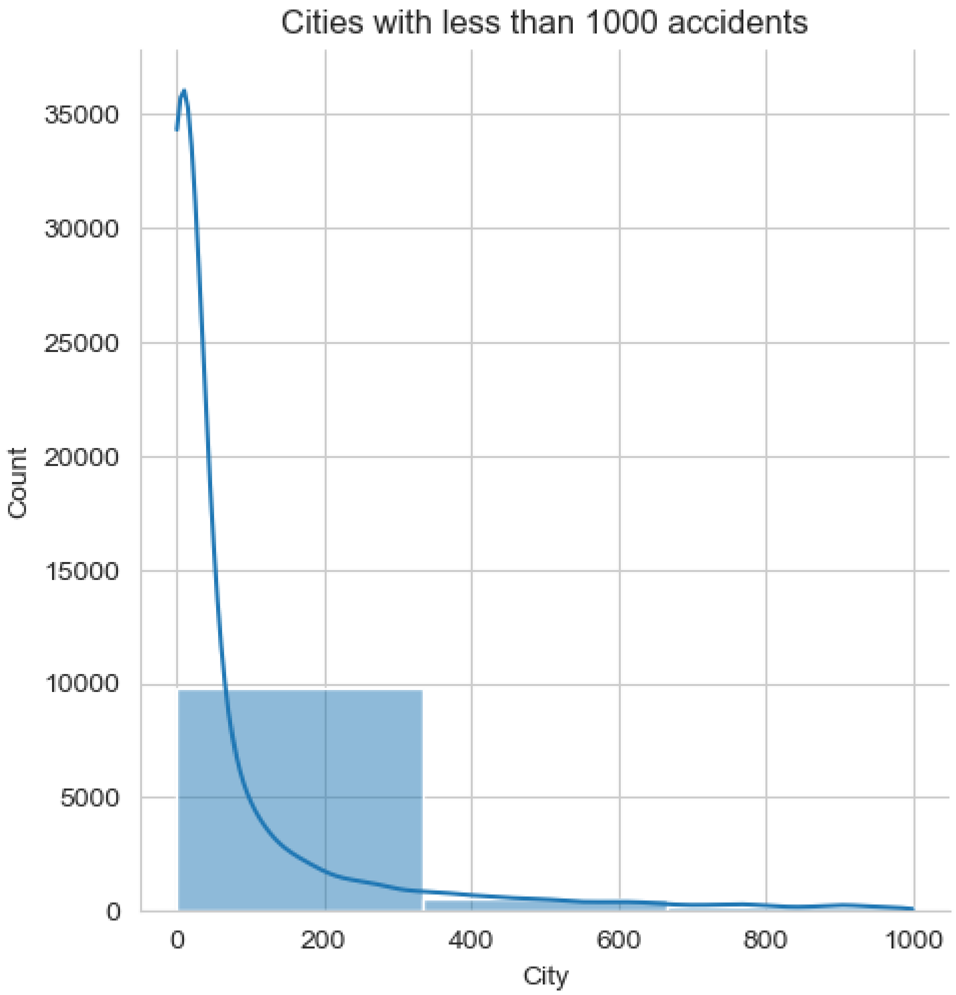
```
Number of cities with less than 1000 accidents: 10594
```

In [108…
```
# percentage of lowest accident cities
print('Percentage: '+str(len(low_accident_cities) / cities * 100))
```

```
Percentage: 95.56197005231823
```

In [109…
```
sns.displot(low_accident_cities, kde = True , bins = 3)
plt.title("Cities with less than 1000 accidents")
```

Out[109]:
```
Text(0.5, 1.0, 'Cities with less than 1000 accidents')
```



In [110…
```
# Cities with one number of accidents;
cities_by_accidents[cities_by_accidents == 1].sum()
```
Out[110]:
```
1594
```

Summary:

Number of accidents per city decreases exponentially. Less than five percent of cities have more than 1000 accidents. Less tham 1000 accidents are recorded for 95% of cities. It seems like over 1500 cities reported only one accident.

Start time

In [111…
```
# Lets analyse start_time column;
df['Start_Time']
```
Out[111]:
```
0          2016-02-08 05:46:00
1          2016-02-08 06:07:59
2          2016-02-08 06:49:27
3          2016-02-08 07:23:34
4          2016-02-08 07:39:07
                  ...
2999995    2018-02-13 14:49:29
2999996    2018-02-13 15:05:31
2999997    2018-02-13 15:21:08
2999998    2018-02-13 15:41:05
2999999    2018-02-13 07:36:21
Name: Start_Time, Length: 3000000, dtype: object
```

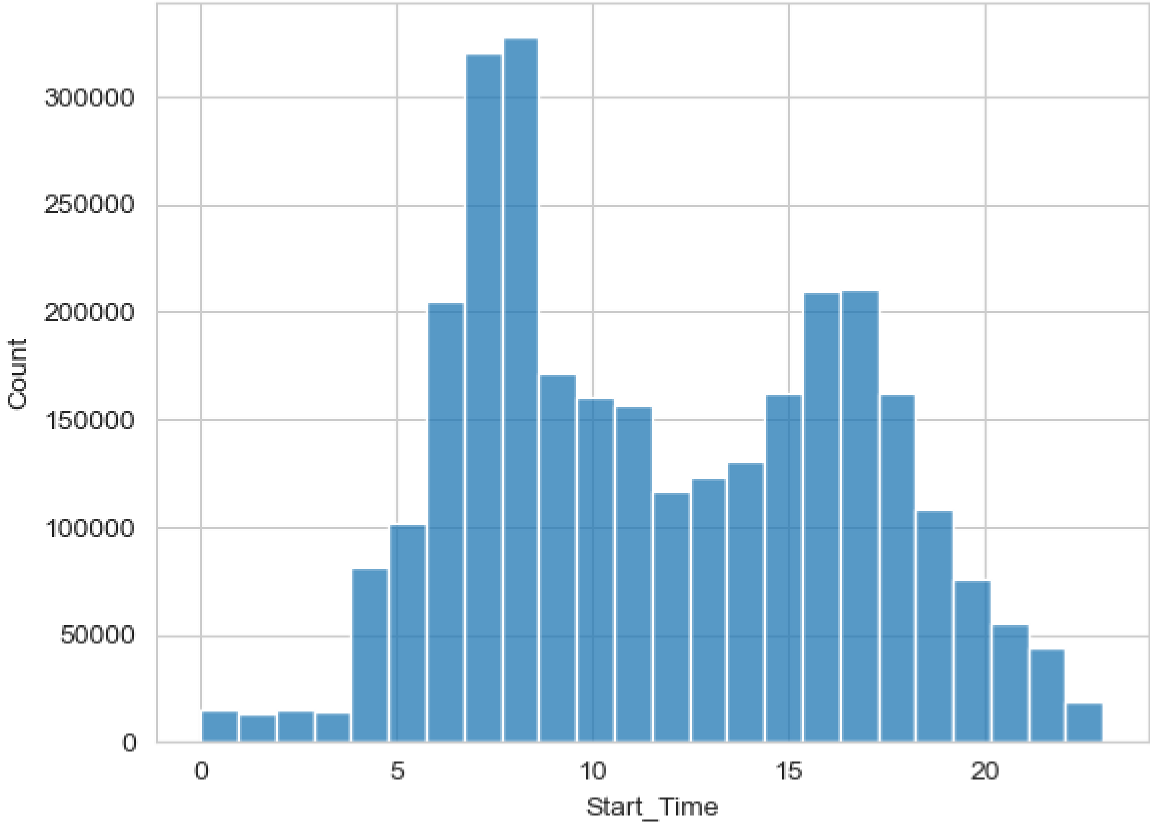
In [112…
```
# start_time column is in string form.
# converting this column into date datatype;
df['Start_Time'] = pd.to_datetime(df['Start_Time'])
```

In [113…
```
df['Start_Time'][0] # Now it is in date form
```

Out[113]:
```
Timestamp('2016-02-08 05:46:00')
```

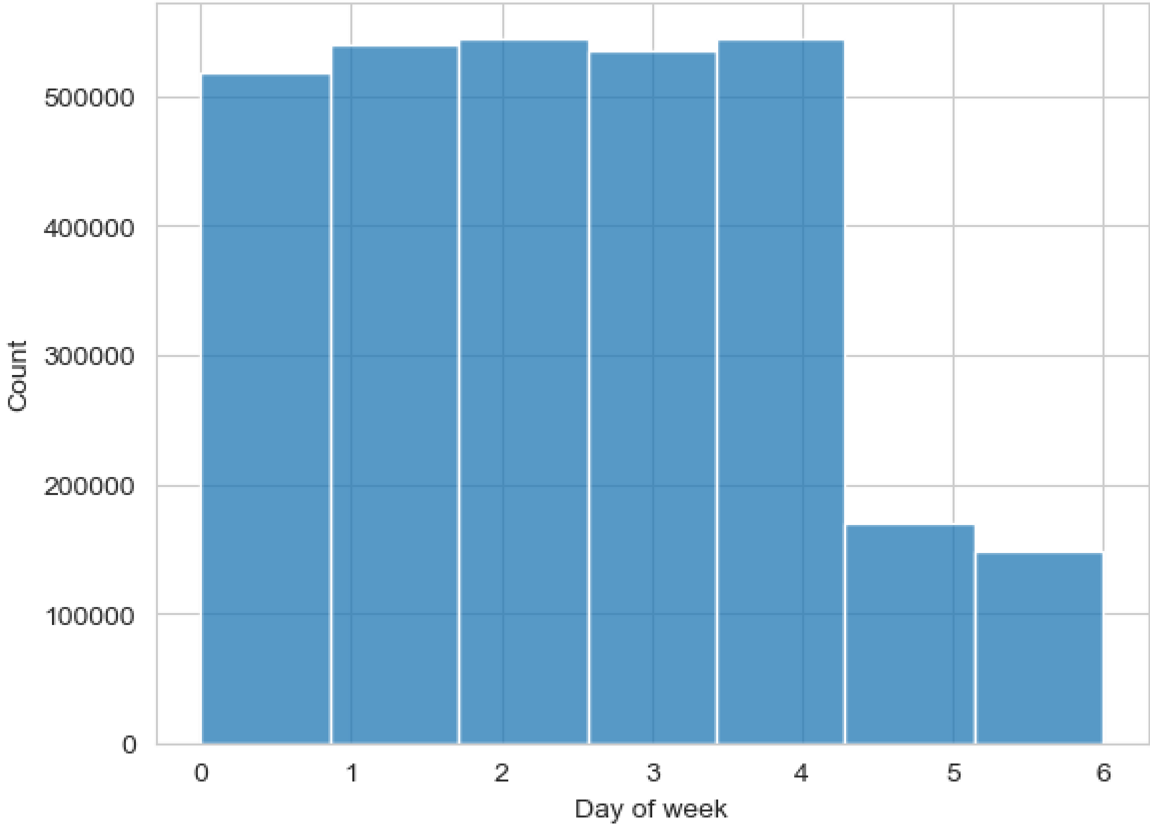
In [114…
```
# Lets check at what time of the day there is high percentage of accidents
sns.set_style('whitegrid')
sns.histplot(df['Start_Time'].dt.hour , bins = 24 , kde = False)
```

Out[114]:
```
<Axes: xlabel='Start_Time', ylabel='Count'>
```



In [115…
```
# check for trend of accidents in weak
sns.histplot(df['Start_Time'].dt.dayofweek , bins = 7)
plt.xlabel("Day of week")
```

Out[115]:
```
Text(0.5, 0, 'Day of week')
```

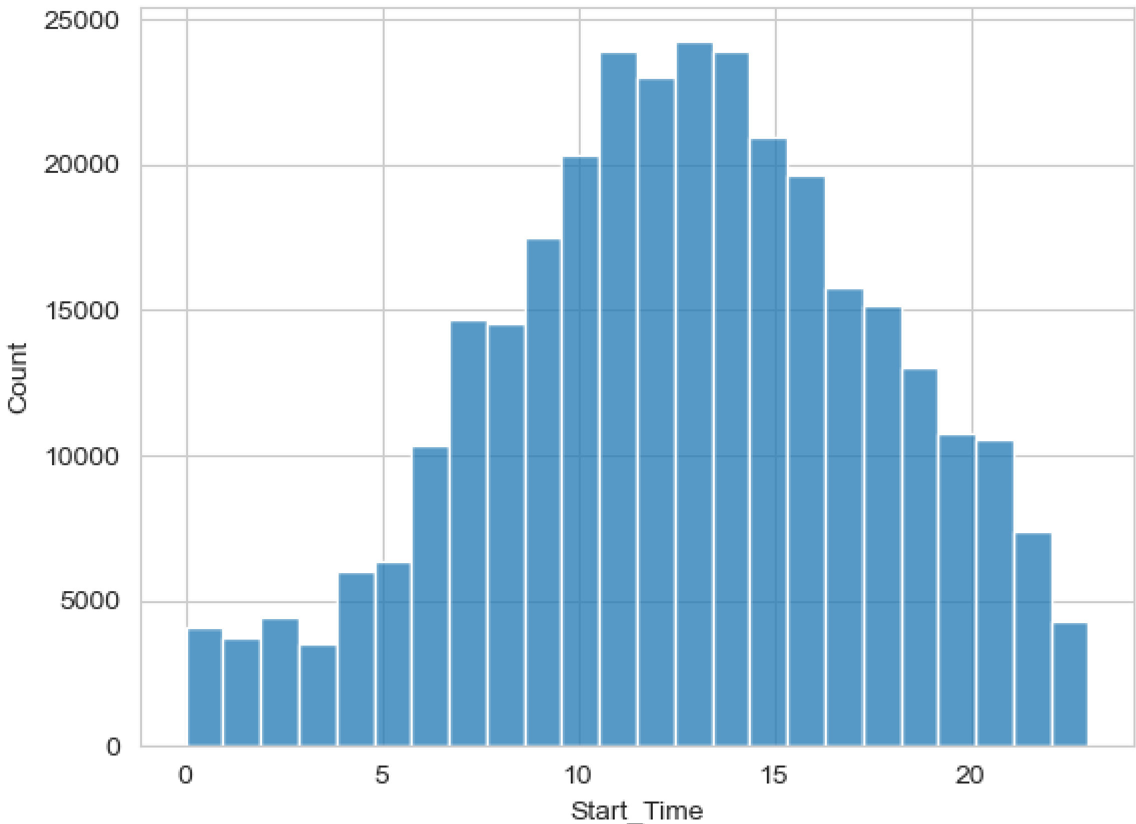


In [116…
```
# lets analyse whether accidents are more prone between 6AM and 10AM on weekends a

weekend_starttime = df[(df['Start_Time'].dt.dayofweek == 5) | (df['Start_Time'].dt
```

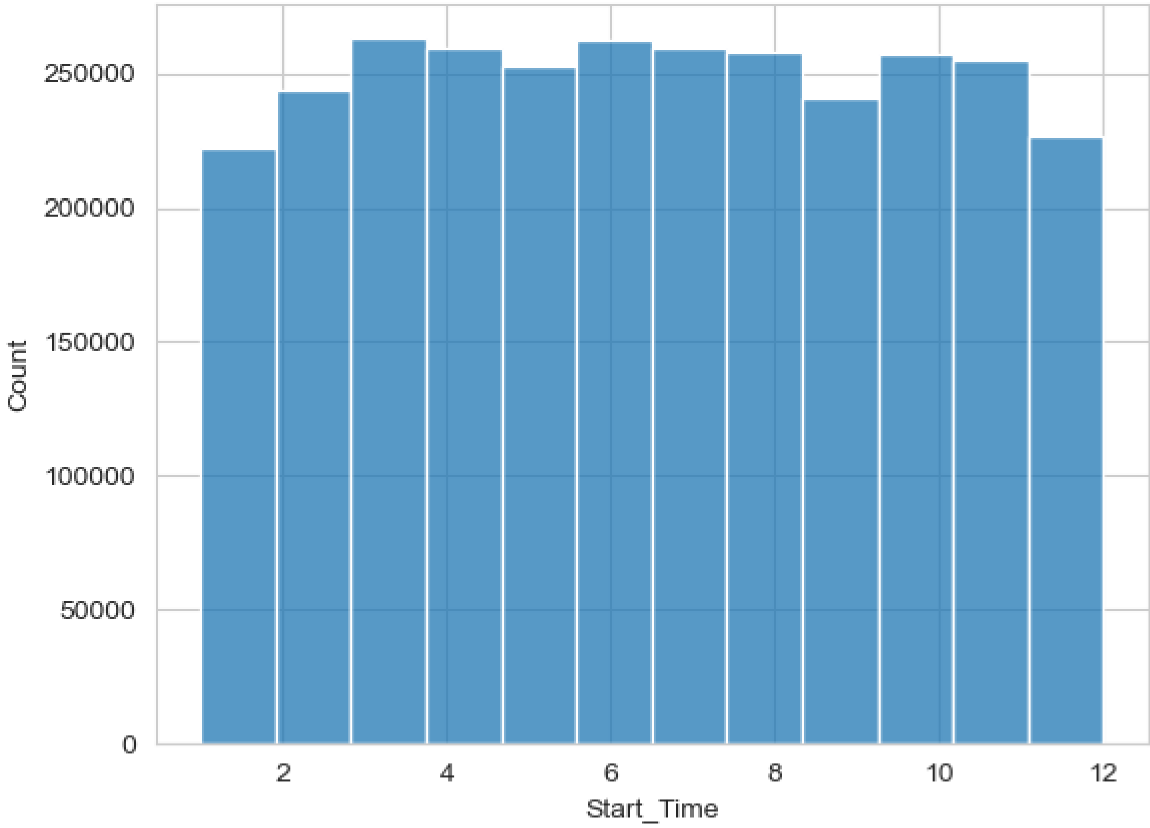
In [117…
```
sns.histplot(weekend_starttime['Start_Time'].dt.hour , bins = 24)
```

Out[117]: <Axes: xlabel='Start_Time', ylabel='Count'>



In [118…
```
# check trend of accidents in month;
sns.histplot(df['Start_Time'].dt.month , bins = 12)
```

Out[118]: <Axes: xlabel='Start_Time', ylabel='Count'>

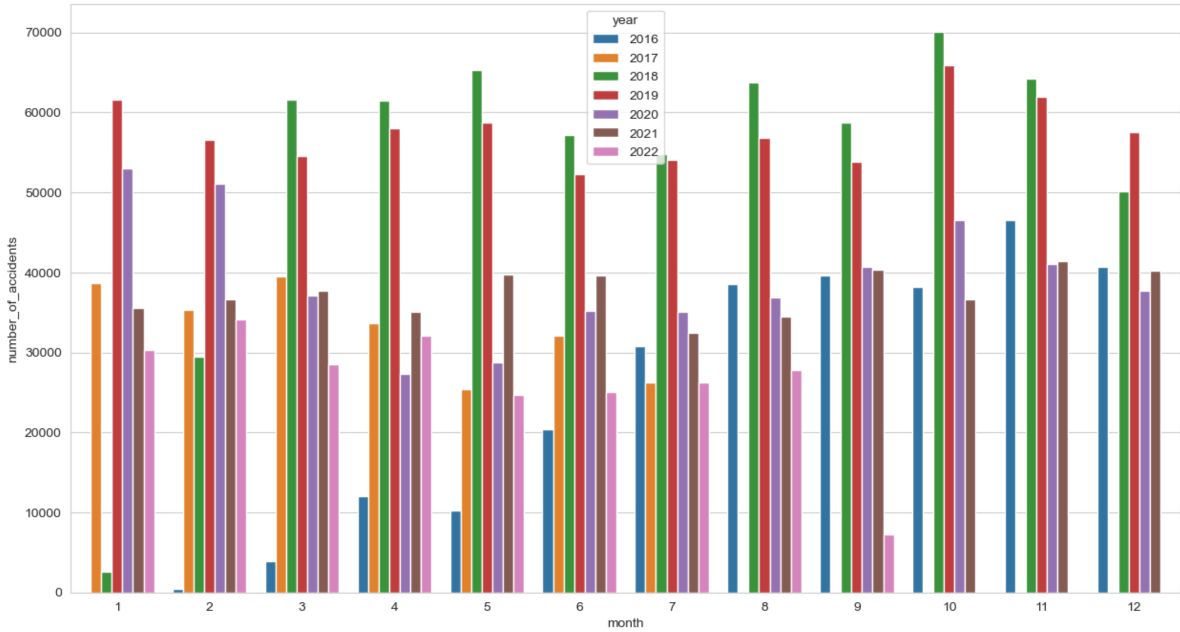


In [119…
```
# Now we will interpret accident rates in every month for each year:
```

In [120…
```
df['Month'] = df['Start_Time'].dt.month
df['Year'] = df['Start_Time'].dt.year
monthly_accidents=df[["Month","Year"]].value_counts().reset_index()
monthly_accidents.columns=["month","year","number_of_accidents"]
```

In [121…
```
plt.figure(figsize=(15,8))
sns.barplot(x="month",y="number_of_accidents",hue="year",data=monthly_accidents)
```

Out[121]: <Axes: xlabel='month', ylabel='number_of_accidents'>



Summary

Most of the accidents are occured between 7AM to 10AM. On weekends it seems like accidents are less. unlike in week days, more accidents in weekends are occuring during afternoon between 11AM and 3PM. I think there is no particular trend in accidents by month in a year. MOre accidents are recorded in the years 2018 and 2019 in most of the months.

Start Latitude and Longitude

In [122…

```
df[['Start_Lat' , 'Start_Lng']]
```

Out[122]:

	Start_Lat	Start_Lng
0	39.865147	-84.058723
1	39.928059	-82.831184
2	39.063148	-84.032608
3	39.747753	-84.205582
4	39.627781	-84.188354
...	...	...
2999995	39.743855	-105.016495
2999996	38.948586	-104.803490
2999997	38.930573	-104.775215
2999998	38.985554	-104.765465
2999999	38.725010	-104.733223

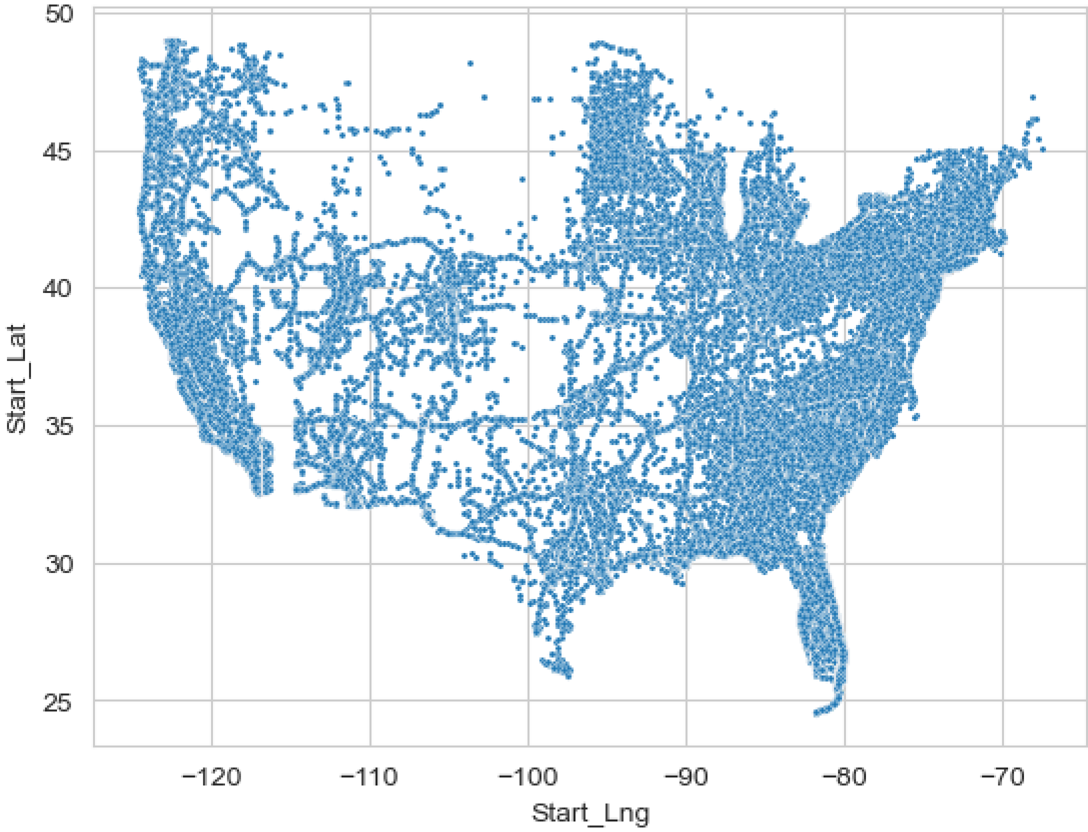
3000000 rows × 2 columns

In [123…

```
sns.scatterplot(data = df , x = 'Start_Lng' , y = 'Start_Lat' , s = 5)
```

Out[123]:

```
<Axes: xlabel='Start_Lng', ylabel='Start_Lat'>
```



Point one marker on plot

In [44]:
```
# Lets try to put it in a map
import folium
folium.Map() # it gives world map
# lets plot one accident in map;
lat, lng = df['Start_Lat'][0] , df['Start_Lat'][0]
map = folium.Map()
marker = folium.Marker((lat, lng))
marker.add_to(map)
map
# Pointing one accident spot in map
```

Out[44]: Make this Notebook Trusted to load map: File -> Trust Notebook

To point multiple markers on map:

In [124…

```
# Only 0.001% of sample is taken to mark multiple points on map:
sample_df1 = df.sample(int(0.0001 * len(df)))
locations = sample_df1[['Start_Lng' , 'Start_Lat']]
location_list = locations.values.tolist()
```

In [125…

```
len(location_list)
```

Out[125]:

```
300
```

In [47]:

```
map = folium.Map()
for x in range(0, len(location_list)):
    marker = folium.Marker(location_list[x])
    marker.add_to(map)
map
```

Out[47]: Make this Notebook Trusted to load map: File -> Trust Notebook

Heatmap of areas where accidents have occured

In [48]:

```
lat_lng = list(zip(list(df['Start_Lat']) , list(df['Start_Lng'])))
```

In [49]:

```
# Lets create a heatmap

from folium.plugins import HeatMap

map = folium.Map()
marker = HeatMap(lat_lng).add_to(map)
map
```

Out[49]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [50]:
```
# Lets create heatmap for sample data
```

In [51]:
```
sample_df = df.sample(int(0.01 * len(df)))
samp_lat_lng = list(zip(list(sample_df['Start_Lat']) , list(sample_df['Start_Lng']
map = folium.Map()
HeatMap(samp_lat_lng).add_to(map)
map
```

Out[51]: Make this Notebook Trusted to load map: File -> Trust Notebook

Severity

In [126…
```
df.columns
```

Out[126]:
```
Index(['ID', 'Source', 'Severity', 'Start_Time', 'End_Time', 'Start_Lat',
       'Start_Lng', 'Distance(mi)', 'Description', 'Street', 'City', 'County',
       'State', 'Zipcode', 'Country', 'Timezone', 'Airport_Code',
       'Weather_Timestamp', 'Temperature(F)', 'Wind_Chill(F)', 'Humidity(%)',
       'Pressure(in)', 'Visibility(mi)', 'Wind_Direction', 'Wind_Speed(mph)',
       'Precipitation(in)', 'Weather_Condition', 'Amenity', 'Bump', 'Crossing',
       'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station',
       'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop',
       'Sunrise_Sunset', 'Civil_Twilight', 'Nautical_Twilight',
       'Astronomical_Twilight', 'Month', 'Year'],
      dtype='object')
```

In [127…
```
df[['Severity', 'Year']]
```

Out[127]:

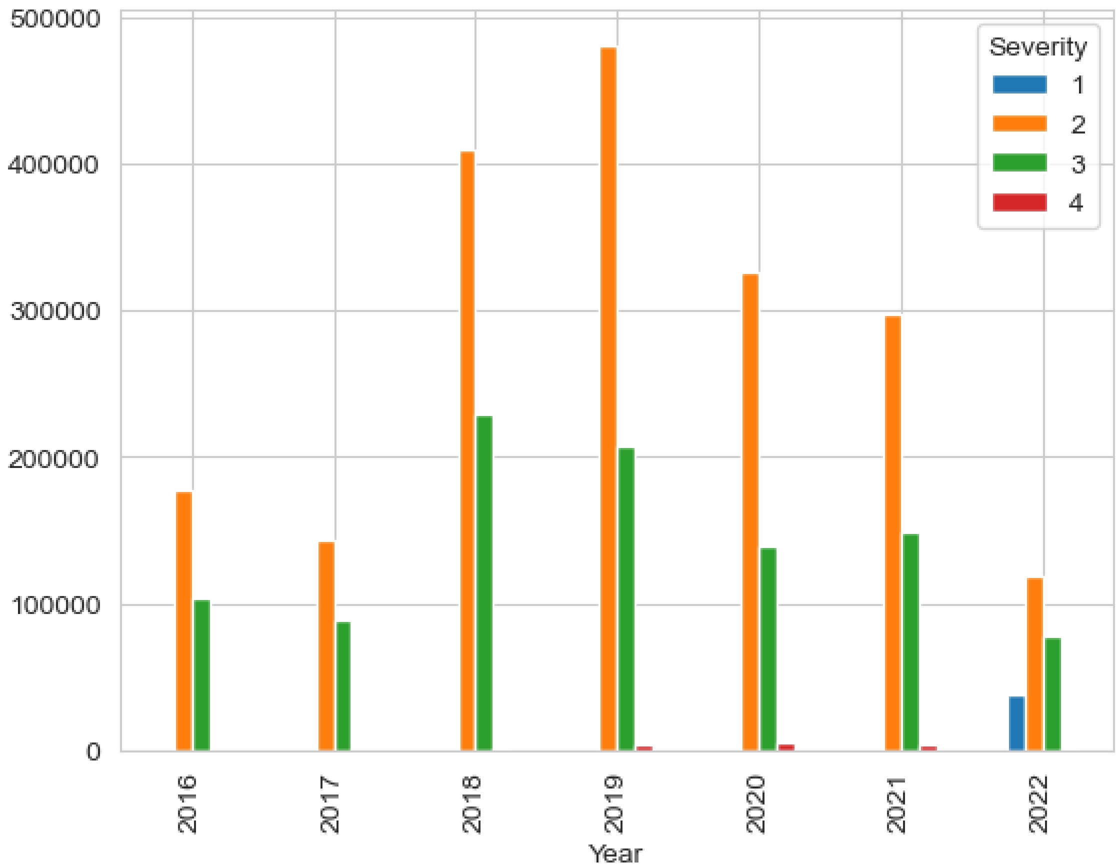
	Severity	Year
0	3	2016
1	2	2016
2	2	2016
3	3	2016
4	2	2016
...	...	...
2999995	3	2018
2999996	2	2018
2999997	2	2018
2999998	2	2018
2999999	2	2018

3000000 rows × 2 columns

Severity of accidents in each year

In [54]:
```
pd.crosstab(df["Year"],df["Severity"]).plot(kind="bar")
```

Out[54]: <Axes: xlabel='Year'>



Summary:
It seems like the trend of severity level 2 is common in all the years

Temperature

In [128…

```
df.columns
```

Out[128]:

```
Index(['ID', 'Source', 'Severity', 'Start_Time', 'End_Time', 'Start_Lat',
       'Start_Lng', 'Distance(mi)', 'Description', 'Street', 'City', 'County',
       'State', 'Zipcode', 'Country', 'Timezone', 'Airport_Code',
       'Weather_Timestamp', 'Temperature(F)', 'Wind_Chill(F)', 'Humidity(%)',
       'Pressure(in)', 'Visibility(mi)', 'Wind_Direction', 'Wind_Speed(mph)',
       'Precipitation(in)', 'Weather_Condition', 'Amenity', 'Bump', 'Crossing',
       'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station',
       'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop',
       'Sunrise_Sunset', 'Civil_Twilight', 'Nautical_Twilight',
       'Astronomical_Twilight', 'Month', 'Year'],
      dtype='object')
```

In [129…

```
df['Temperature(F)'].value_counts()
```

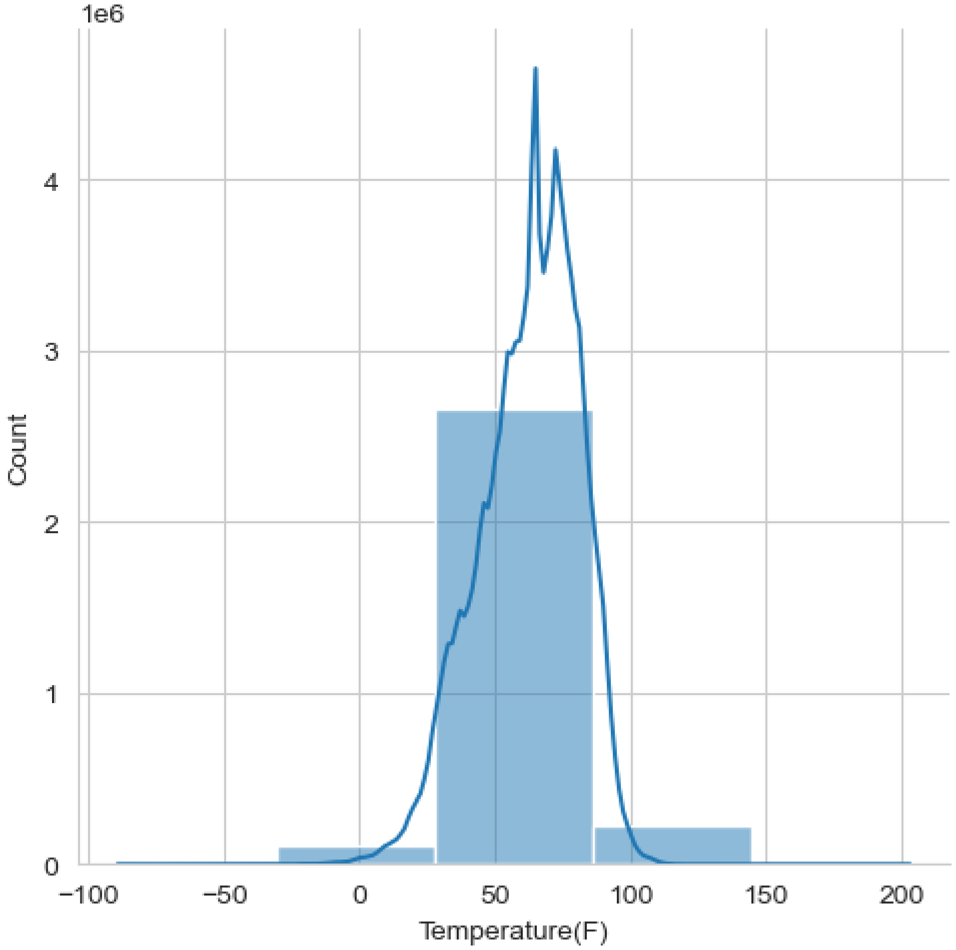
Out[129]:

```
 77.0     69931
 64.9     68738
 73.0     68510
 68.0     67344
 72.0     63930
          ...
-9.6          1
-15.3         1
 116.0        1
-24.9         1
 108.7        1
Name: Temperature(F), Length: 827, dtype: int64
```

In [130…

```
sns.displot(df['Temperature(F)'] , bins = 5, kde = True)
```

Out[130]: <seaborn.axisgrid.FacetGrid at 0x1f8c4e4abc0>

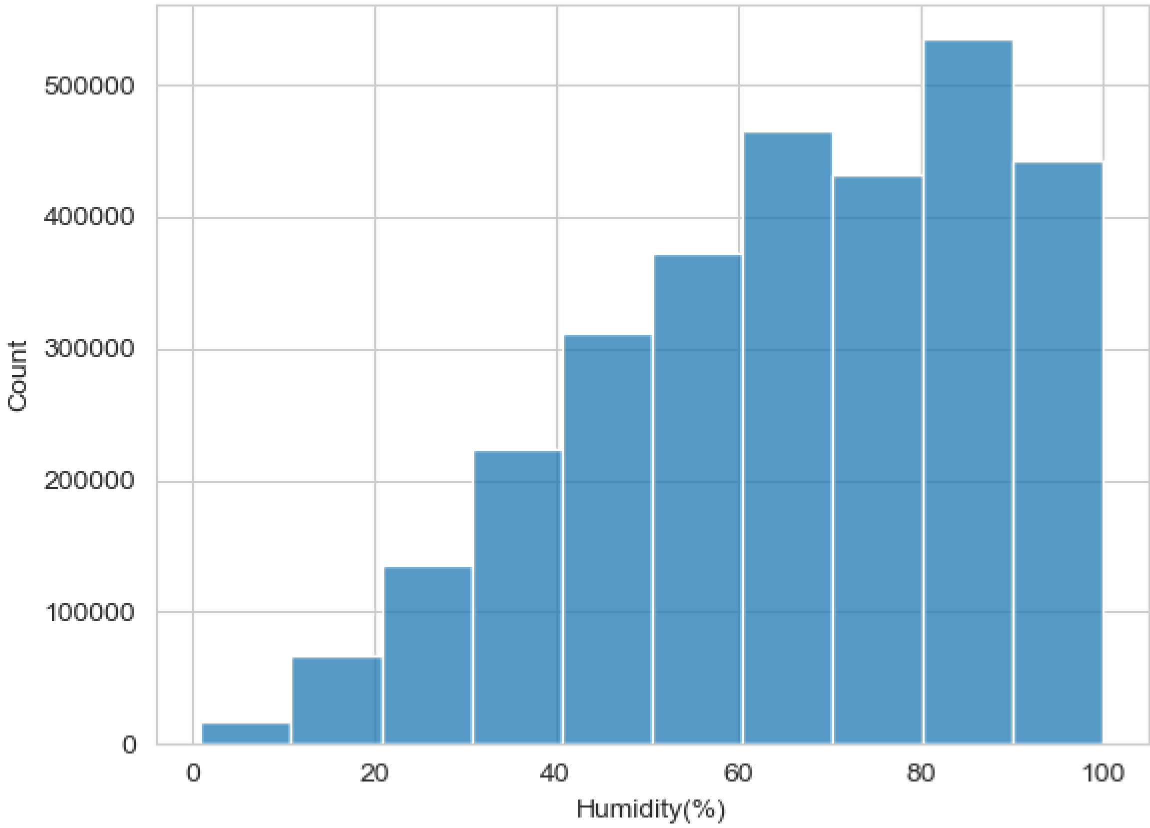


Summary:
More number of accidents occured in the temperatures between 30°F to 50°F.

Humidity

In [131…
```
sns.histplot(df['Humidity(%)'] , bins = 10)
```

Out[131]: <Axes: xlabel='Humidity(%)', ylabel='Count'>



Summary:
There is increasing trend of accidents with increase in the percentage of humidity

Weather condition

In [132…

```
df['Weather_Condition'].unique()
```

Out[132]:
```
array(['Light Rain', 'Overcast', 'Mostly Cloudy', 'Rain', 'Light Snow',
       'Haze', 'Scattered Clouds', 'Partly Cloudy', 'Clear', 'Snow',
       'Light Freezing Drizzle', 'Light Drizzle', 'Fog', 'Shallow Fog',
       'Heavy Rain', 'Light Freezing Rain', 'Cloudy', 'Drizzle', 'Fair',
       'Light Rain Showers', 'Mist', 'Smoke', 'Patches of Fog',
       'Light Freezing Fog', 'Light Haze', 'Light Thunderstorms and Rain',
       'Thunderstorms and Rain', 'Volcanic Ash', 'Blowing Sand',
       'Blowing Dust / Windy', 'Widespread Dust', 'Fair / Windy',
       'Rain Showers', 'Mostly Cloudy / Windy', 'Light Rain / Windy',
       'Hail', 'Heavy Drizzle', 'Showers in the Vicinity', 'Thunderstorm',
       'Light Rain Shower', 'Light Rain with Thunder',
       'Partly Cloudy / Windy', 'Thunder in the Vicinity', 'T-Storm',
       'Heavy Thunderstorms and Rain', 'Thunder', 'Heavy T-Storm',
       'Funnel Cloud', 'Heavy T-Storm / Windy', 'Blowing Snow',
       'Light Thunderstorms and Snow', 'Heavy Snow', 'Low Drifting Snow',
       'Light Ice Pellets', 'Ice Pellets', 'Squalls', 'N/A Precipitation',
       'Cloudy / Windy', 'Light Fog', 'Sand', 'Snow Grains',
       'Snow Showers', 'Heavy Thunderstorms and Snow', 'Rain / Windy',
       'Heavy Rain / Windy', 'Heavy Ice Pellets', 'Light Snow / Windy',
       'Heavy Freezing Rain', 'Small Hail', 'Heavy Rain Showers',
       'Thunder / Windy', 'Drizzle and Fog', 'T-Storm / Windy',
       'Blowing Dust', 'Smoke / Windy', 'Haze / Windy', 'Tornado',
       'Light Drizzle / Windy', 'Widespread Dust / Windy', 'Wintry Mix',
       'Wintry Mix / Windy', 'Light Snow with Thunder', 'Fog / Windy',
       'Snow and Thunder', 'Light Snow Shower', 'Sleet',
       'Light Snow and Sleet', 'Snow / Windy', 'Rain Shower',
       'Snow and Sleet', 'Light Sleet', 'Heavy Snow / Windy',
       'Freezing Drizzle', 'Light Freezing Rain / Windy',
       'Thunder / Wintry Mix', 'Blowing Snow / Windy', 'Freezing Rain',
       'Light Snow and Sleet / Windy', 'Snow and Sleet / Windy',
       'Sleet / Windy', 'Heavy Freezing Rain / Windy', 'Squalls / Windy',
       'Light Rain Shower / Windy', 'Snow and Thunder / Windy',
       'Light Sleet / Windy', 'Sand / Dust Whirlwinds', 'Mist / Windy',
       'Drizzle / Windy', 'Duststorm', 'Sand / Dust Whirls Nearby',
       'Thunder and Hail', 'Heavy Sleet', 'Freezing Rain / Windy',
       'Light Snow Shower / Windy', 'Partial Fog',
       'Thunder / Wintry Mix / Windy', 'Patches of Fog / Windy',
       'Rain and Sleet', 'Light Snow Grains', 'Partial Fog / Windy',
       'Sand / Dust Whirlwinds / Windy', 'Heavy Snow with Thunder',
       'Light Snow Showers', 'Heavy Blowing Snow', 'Light Hail',
       'Heavy Smoke', 'Heavy Thunderstorms with Small Hail',
       'Light Thunderstorm', 'Heavy Freezing Drizzle',
       'Light Blowing Snow', 'Thunderstorms and Snow'], dtype=object)
```

In [133…
```
df['Weather_Condition']
```

Out[133]:
```
0                Light Rain
1                Light Rain
2                  Overcast
3             Mostly Cloudy
4             Mostly Cloudy
                 ...
2999995    Scattered Clouds
2999996               Clear
2999997               Clear
2999998               Clear
2999999               Clear
Name: Weather_Condition, Length: 3000000, dtype: object
```

In [135…
```
df['Weather_Condition'].value_counts().sort_values()
```

Out[135]:
```
Thunderstorms and Snow         1
Light Blowing Snow             1
Heavy Freezing Drizzle         1
Light Thunderstorm             1
Blowing Sand                   1
                           ...
Cloudy                    266898
Partly Cloudy             283188
Mostly Cloudy             414926
Clear                     482722
Fair                      811195
Name: Weather_Condition, Length: 131, dtype: int64
```

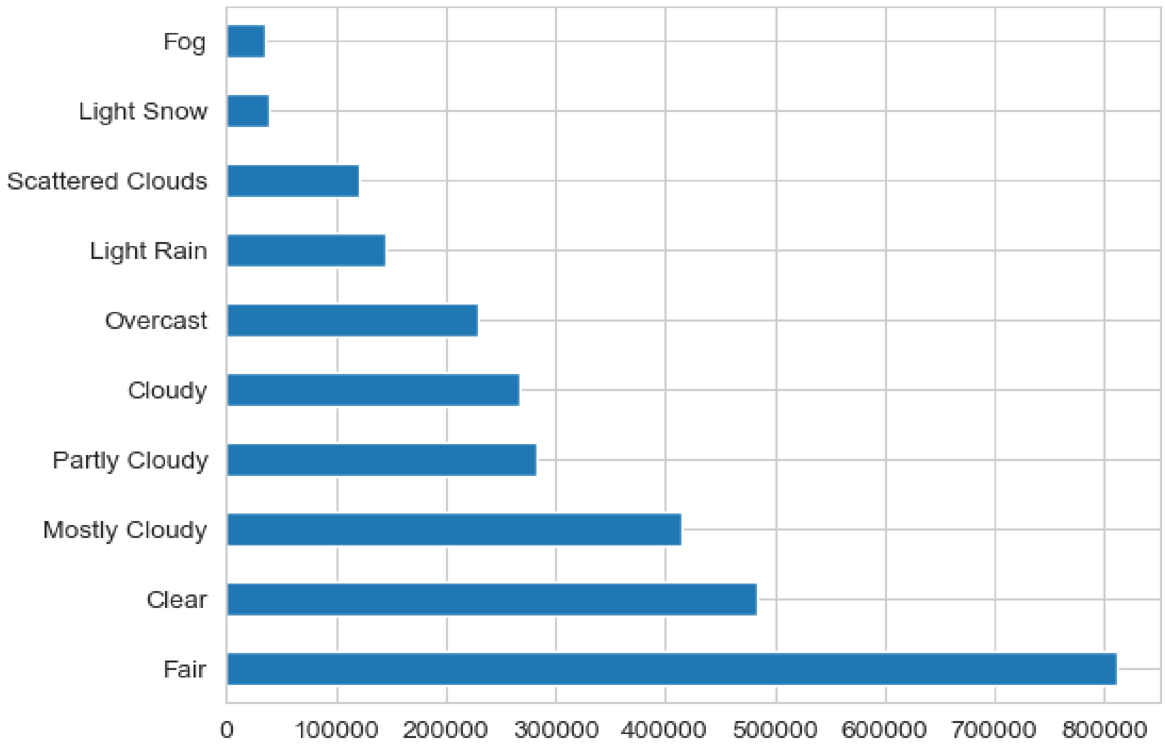
Let's take top 10 weather conditions during time of accident:

In [136…
```
df['Weather_Condition'].value_counts().sort_values(ascending = False)[:10]
```

Out[136]:
```
Fair                811195
Clear               482722
Mostly Cloudy       414926
Partly Cloudy       283188
Cloudy              266898
Overcast            229476
Light Rain          145082
Scattered Clouds    120483
Light Snow           38020
Fog                  35588
Name: Weather_Condition, dtype: int64
```

In [137…
```
weather_top = df['Weather_Condition'].value_counts().sort_values(ascending = False
weather_top.plot(kind = 'barh')
```

Out[137]: <Axes: >



Analyse weather conditions along with the severity of accidents:

In [139…
```
df['Severity'].unique()
```

Out[139]:
```
array([3, 2, 1, 4], dtype=int64)
```

In [138…

```
df[['Weather_Condition' , 'Severity']].value_counts().sort_values(ascending = Fals
```

Out[138]:

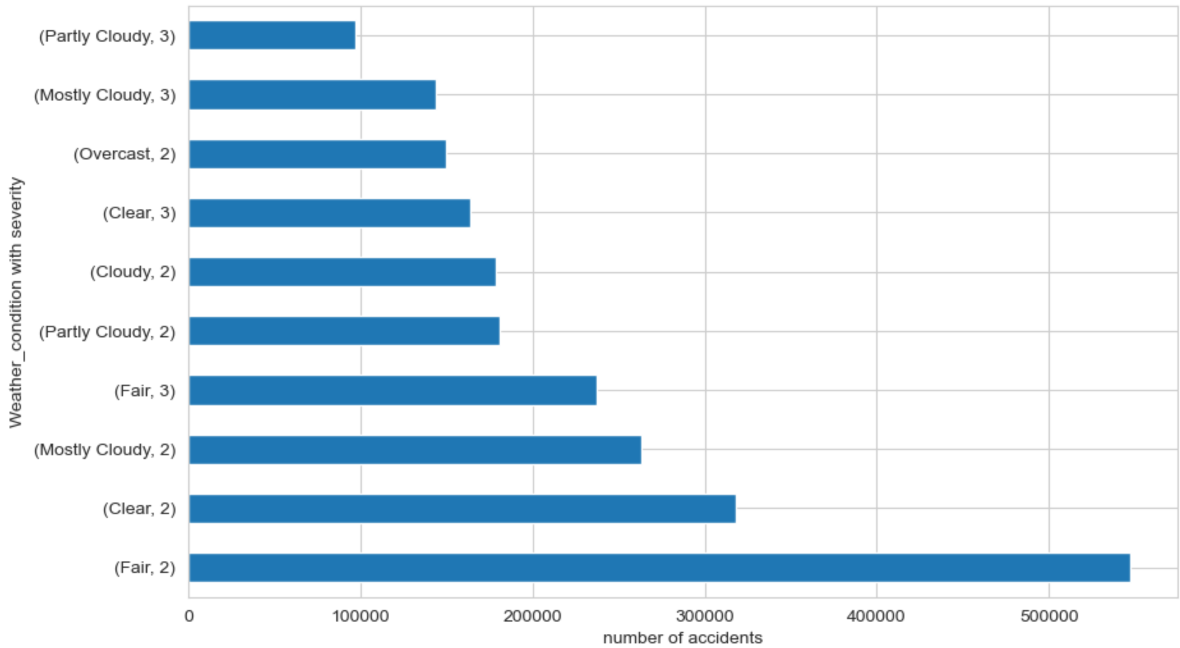
```
Weather_Condition  Severity
Fair               2           547190
Clear              2           318277
Mostly Cloudy      2           262913
Fair               3           237092
Partly Cloudy      2           180872
Cloudy             2           178909
Clear              3           163812
Overcast           2           149542
Mostly Cloudy      3           144098
Partly Cloudy      3            96887
dtype: int64
```

In [63]:

```
top_cond = df[['Weather_Condition' , 'Severity']].value_counts().sort_values(ascen
top_cond.plot(kind = 'barh', figsize = (10,6))
plt.xlabel('number of accidents')
plt.ylabel('Weather_condition with severity')
```

Out[63]:

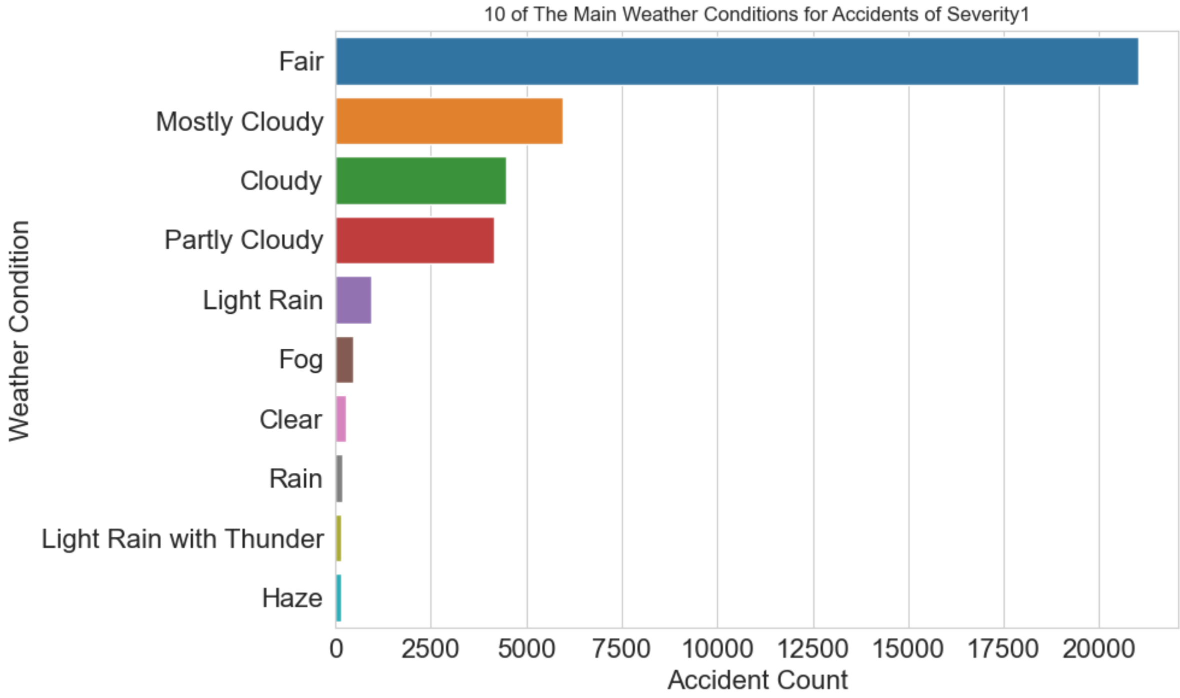
```
Text(0, 0.5, 'Weather_condition with severity')
```

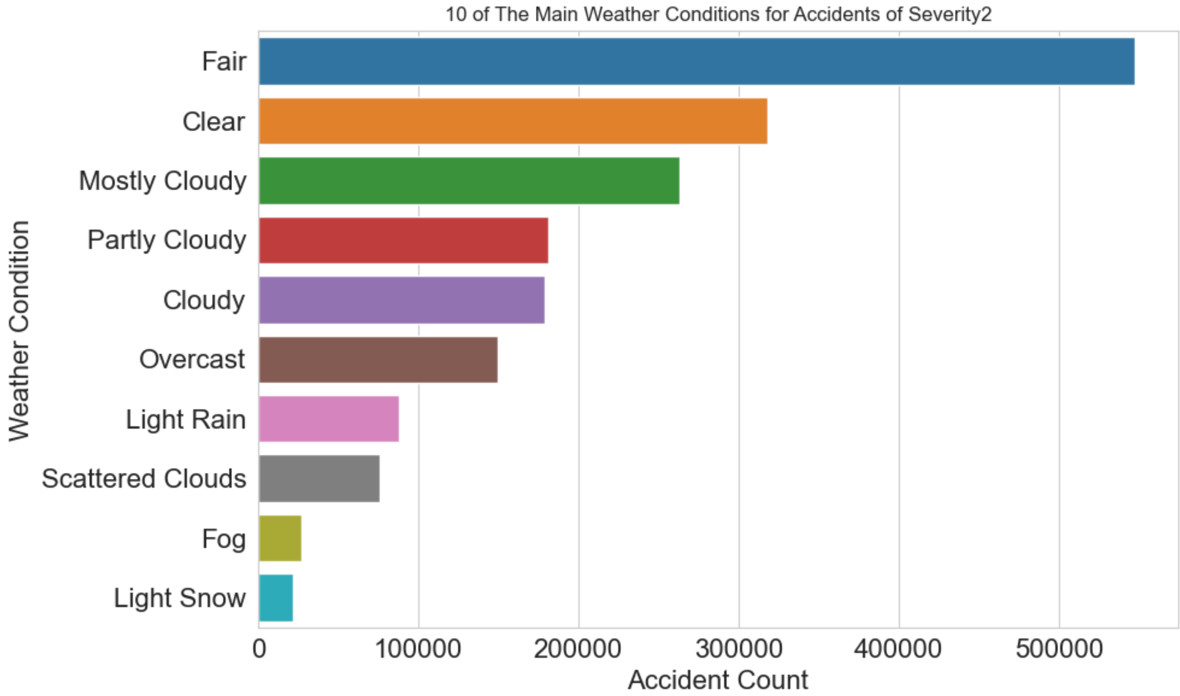


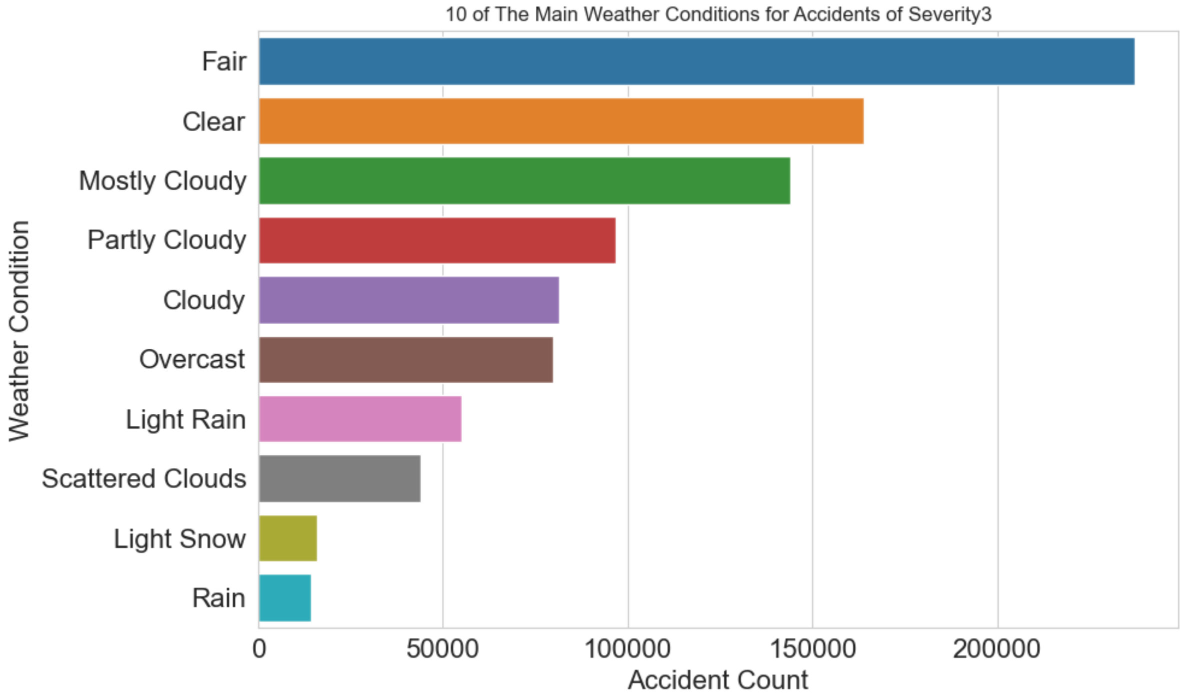
10 of the main weather conditions for accidents at severity 1, 2, 3, 4

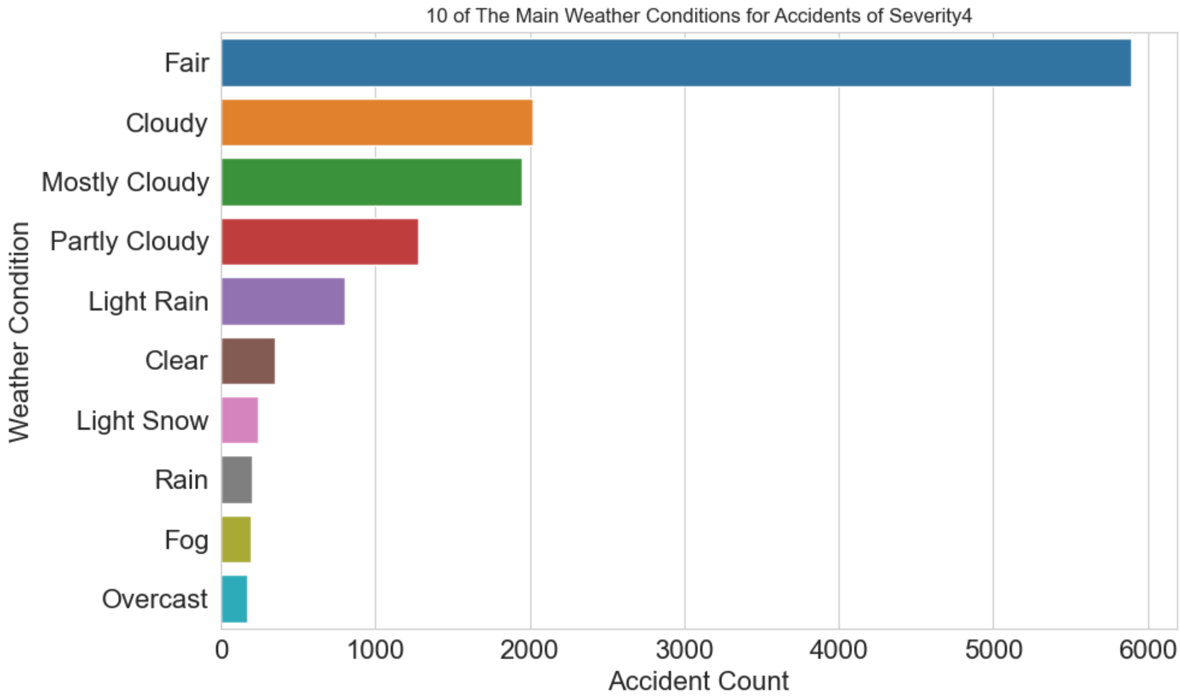
In [65]:

```
for x in range(1,5):
    plt.subplots(figsize = (10,6))
    severity =  df.loc[df['Severity'] == x , ['Weather_Condition']].value_counts()
    severity.columns = ['Weather condition' , 'Number of accidents']
    sns.barplot(y = severity['Weather condition'] , x = severity['Number of accide
    plt.ylabel('Weather Condition',fontsize=16)
    plt.xlabel('Accident Count',fontsize=16)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    plt.title('10 of The Main Weather Conditions for Accidents of Severity'+str(x)
    plt.tight_layout()
```


10 of The Main Weather Conditions for Accidents of Severity1
Weather Condition
Fair
Mostly Cloudy
Cloudy
Partly Cloudy
Light Rain
Fog
Clear
Rain
Light Rain with Thunder
Haze
0
2500
5000
7500
10000
12500
15000
17500
20000
Accident Count


10 of The Main Weather Conditions for Accidents of Severity2
Weather Condition
Fair
Clear
Mostly Cloudy
Partly Cloudy
Cloudy
Overcast
Light Rain
Scattered Clouds
Fog
Light Snow
0
100000
200000
300000
400000
500000
Accident Count





Summary

Most of the accidents have occured in fair weather conditions in all severity levels. The second most common weather condition is clear weather for severity 2 and 3, which is not the case with severity 1 and 4.