US Accidents Exploratory Data Analysis

Description This is a countrywide car accident dataset that covers 49 states of the USA. The accident data were collected from February 2016 to March 2023, using multiple APIs that provide streaming traffic incident (or event) data. These APIs broadcast traffic data captured by various entities, including the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road networks. The dataset currently contains approximately 7.7 Million accident records. For more information about this dataset, please visit here.

TODO - TALK ABOUT EDA TODO - TALK ABOUT DATASET (SOURCES,WHAT IT CONTAINS,HOW IT WILL USEFUL)

KAGGLE, INFORMATION ABOUT ACCIDENTS, CAN BE USEFUL TO PREVENT ACCIDENTS Mention that this does not contain data about New York

Importing the important python libraries

In [24]:

```
# Importing the important python libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

DATA PREPARATION AND CLEANING

In [25]:

```
df = pd.read_csv(r"C:\Users\AYUSH\Downloads\archive (1)\US_Accidents_March23.csv")
```

In [26]:

```
df
```

Out [26]:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	...	Roundabout	Station	Stop
0	A-1	Source2	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	NaN	NaN	0.010	...	False	False	False
1	A-2	Source2	2	2016-02-08 06:07:59	2016-02-08 06:37:59	39.928059	-82.831184	NaN	NaN	0.010	...	False	False	False
2	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	NaN	NaN	0.010	...	False	False	False
3	A-4	Source2	3	2016-02-08 07:23:34	2016-02-08 07:53:34	39.747753	-84.205582	NaN	NaN	0.010	...	False	False	False
4	A-5	Source2	2	2016-02-08 07:39:07	2016-02-08 08:09:07	39.627781	-84.188354	NaN	NaN	0.010	...	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7728389	A-7777757	Source1	2	2019-08-23 18:03:25	2019-08-23 18:32:01	34.002480	-117.379360	33.99888	-117.37094	0.543	...	False	False	False
7728390	A-7777758	Source1	2	2019-08-23 19:11:30	2019-08-23 19:38:23	32.766960	-117.148060	32.76555	-117.15363	0.338	...	False	False	False
7728391	A-7777759	Source1	2	2019-08-23 19:00:21	2019-08-23 19:28:49	33.775450	-117.847790	33.77740	-117.85727	0.561	...	False	False	False
7728392	A-7777760	Source1	2	2019-08-23 19:00:21	2019-08-23 19:29:42	33.992460	-118.403020	33.98311	-118.39565	0.772	...	False	False	False
7728393	A-7777761	Source1	2	2019-08-23 18:52:06	2019-08-23 19:21:31	34.133930	-117.230920	34.13736	-117.23934	0.537	...	False	False	False

7728394 rows × 46 columns

In [27]:

```
df.columns
```

Out [27]:

```
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 [28]:

```python
# Overall Information about the Data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7728394 entries, 0 to 7728393
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: 2.0+ GB
```

In [29]:

```python
# Descriptive Statistics of all the numerical columns
df.describe()
```

Out [29]:

	Severity	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	Temperature(F)	Wind_Chill(F)	Humidity(%)	Pressure
count	7.728394e+06	7.728394e+06	7.728394e+06	4.325632e+06	4.325632e+06	7.728394e+06	7.564541e+06	5.729375e+06	7.554250e+06	7.587715e+0
mean	2.212384e+00	3.620119e+01	-9.470255e+01	3.626183e+01	-9.572557e+01	5.618423e-01	6.166329e+01	5.825105e+01	6.483104e+01	2.953899e+
std	4.875313e-01	5.076079e+00	1.739176e+01	5.272905e+00	1.810793e+01	1.776811e+00	1.901365e+01	2.238983e+01	2.282097e+01	1.006190e+0
min	1.000000e+00	2.455480e+01	-1.246238e+02	2.456601e+01	-1.245457e+02	0.000000e+00	-8.900000e+01	-8.900000e+01	1.000000e+00	0.000000e+
25%	2.000000e+00	3.339963e+01	-1.172194e+02	3.346207e+01	-1.177543e+02	0.000000e+00	4.900000e+01	4.300000e+01	4.800000e+01	2.937000e+
50%	2.000000e+00	3.582397e+01	-8.776662e+01	3.618349e+01	-8.802789e+01	3.000000e-02	6.400000e+01	6.200000e+01	6.700000e+01	2.986000e+
75%	2.000000e+00	4.008496e+01	-8.035368e+01	4.017892e+01	-8.024709e+01	4.640000e-01	7.600000e+01	7.500000e+01	8.400000e+01	3.003000e+
max	4.000000e+00	4.900220e+01	-6.711317e+01	4.907500e+01	-6.710924e+01	4.417500e+02	2.070000e+02	2.070000e+02	1.000000e+02	5.863000e+

In [30]:

```python
numerics = ['int16','int32','int64','float16','float32','float64']
numeric_df = df.select_dtypes(include=numerics)
len(numeric_df.columns)
```

Out [30]:

```
13
```

In [31]:

```python
# Missing entry
df.isna().sum()
```

Out [31]:

```
ID                             0
Source                         0
Severity                       0
Start_Time                     0
End_Time                       0
Start_Lat                      0
Start_Lng                      0
End_Lat                  3402762
End_Lng                  3402762
Distance(mi)                   0
Description                    5
Street                     10869
City                         253
County                         0
```

```
State                          0
Zipcode                     1915
Country                        0
Timezone                    7808
Airport_Code               22635
Weather_Timestamp         120228
Temperature(F)            163853
Wind_Chill(F)            1999019
Humidity(%)               174144
Pressure(in)              140679
Visibility(mi)            177098
Wind_Direction            175206
Wind_Speed(mph)           571233
Precipitation(in)        2203586
Weather_Condition         173459
Amenity                        0
Bump                           0
Crossing                       0
Give_Way                       0
Junction                       0
No_Exit                        0
Railway                        0
Roundabout                     0
Station                        0
Stop                           0
Traffic_Calming                0
Traffic_Signal                 0
Turning_Loop                   0
Sunrise_Sunset             23246
Civil_Twilight             23246
Nautical_Twilight          23246
Astronomical_Twilight      23246
dtype: int64
```

In [32]:

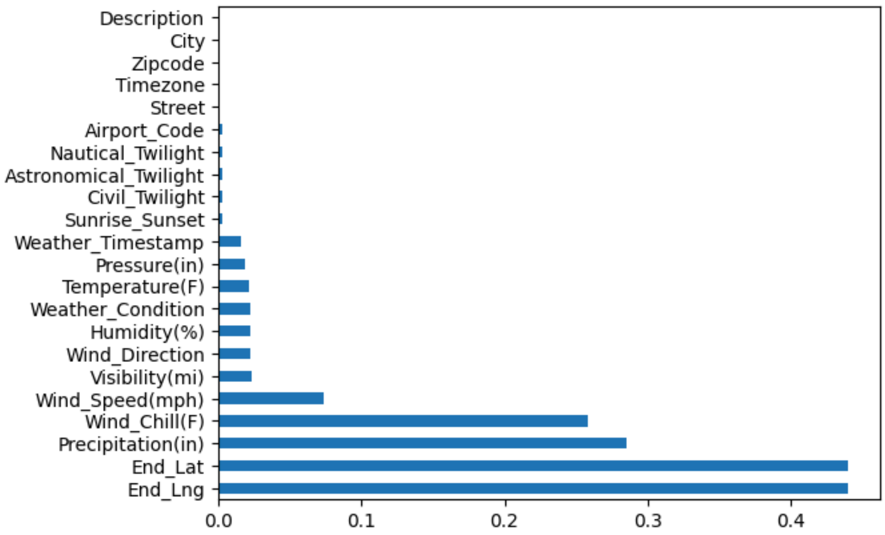
```
#percentage of missing values per columns
missing_df = df.isna().sum().sort_values(ascending=False)/len(df)
```

In [33]:

```
missing_df[missing_df!=0].plot(kind='barh')
```

Out [33]:

```
<Axes: >
```



In [34]:

```
percentage_miss = missing_df[missing_df != 0]
percentage_miss
```

Out [34]:

```
End_Lng                  4.402935e-01
End_Lat                  4.402935e-01
Precipitation(in)        2.851286e-01
Wind_Chill(F)            2.586590e-01
Wind_Speed(mph)          7.391355e-02
Visibility(mi)           2.291524e-02
Wind_Direction           2.267043e-02
Humidity(%)              2.253301e-02
Weather_Condition        2.244438e-02
Temperature(F)           2.120143e-02
Pressure(in)             1.820288e-02
Weather_Timestamp        1.555666e-02
Sunrise_Sunset           3.007869e-03
Civil_Twilight           3.007869e-03
Astronomical_Twilight    3.007869e-03
Nautical_Twilight        3.007869e-03
Airport_Code             2.928810e-03
Street                   1.406372e-03
Timezone                 1.010300e-03
Zipcode                  2.477876e-04
City                     3.273643e-05
Description              6.469649e-07
dtype: float64
```

remove columns that are not useful

In [35]:

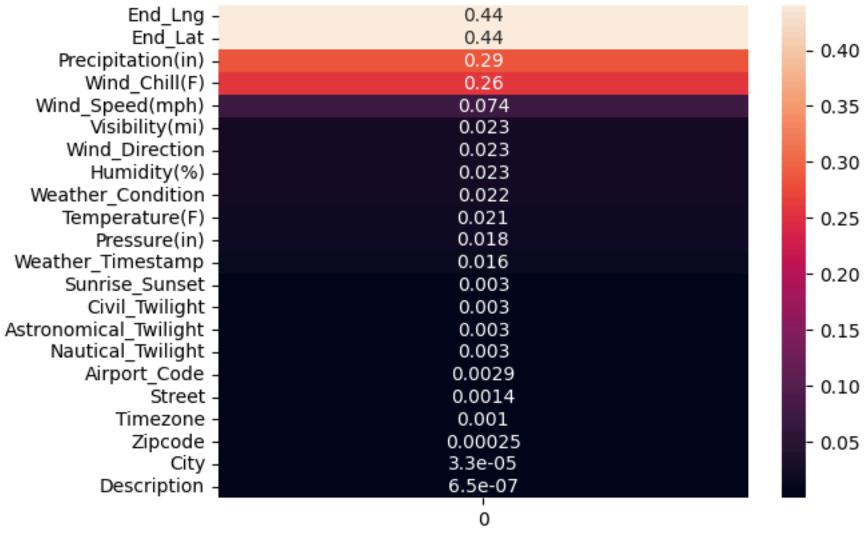
```
# Converting Series to Dataframe
percentage_miss = percentage_miss.to_frame()
```

In [36]:

```
# Heatmap Visualization
sns.heatmap(percentage_miss,annot=True)
```

Out [36]:

```
<Axes: >
```



In [37]:

```
# # We will exclude the columns that contain more than 5% of missing values, as they are not app

cols = [var for var in df.columns if df[var].isnull().mean()>0.05]
cols
```

Out [37]:

```
['End_Lat', 'End_Lng', 'Wind_Chill(F)', 'Wind_Speed(mph)', 'Precipitation(in)']
```

In [38]:

```
# Unnecessary columns removal
df.drop(columns = cols,inplace = True)
```

Exploratory Data And Visualisation

columns we will analyse

1. city
2. start time
3. start lat,start lng
4. state
5. temperature
6. weather condition

In [39]:

```
df.columns
```

Out [39]:

```
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)', 'Humidity(%)', 'Pressure(in)',
       'Visibility(mi)', 'Wind_Direction', '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')
```

City

In [40]:

```
cities = df.City.nunique()
cities
```

Out [40]:

```
13678
```

In [41]:

```
cities_by_accident = df.City.value_counts()
cities_by_accident
```

Out [41]:

```
City
Miami            186917
Houston          169609
Los Angeles      156491
Charlotte        138652
Dallas           130939
                  ...
Rapid River           1
Cat Spring            1
Glenwood City         1
Downing               1
Marfa                 1
Name: count, Length: 13678, dtype: int64
```

In [42]:
```
cities_by_accident[:10]
```

Out [42]:
```
City
Miami          186917
Houston        169609
Los Angeles    156491
Charlotte      138652
Dallas         130939
Orlando        109733
Austin          97359
Raleigh         86079
Nashville       72930
Baton Rouge     71588
Name: count, dtype: int64
```

In [43]:
```
"New York" in df.City
```

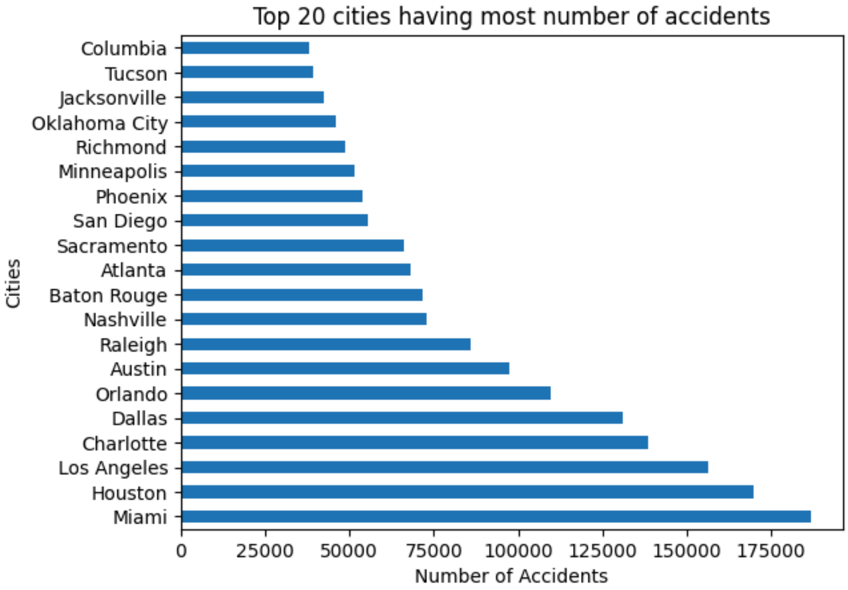
Out [43]:
```
False
```

In [44]:
```
"NY" in df.State
```

Out [44]:
```
False
```

In [45]:
```
# Plot
cities_by_accident[:20].plot(kind="barh",title='Top 20 cities having most number of accidents',x
```

Out [45]:
```
<Axes: title={'center': 'Top 20 cities having most number of accidents'}, xlabel='Number of Accidents',
ylabel='Cities'>
```

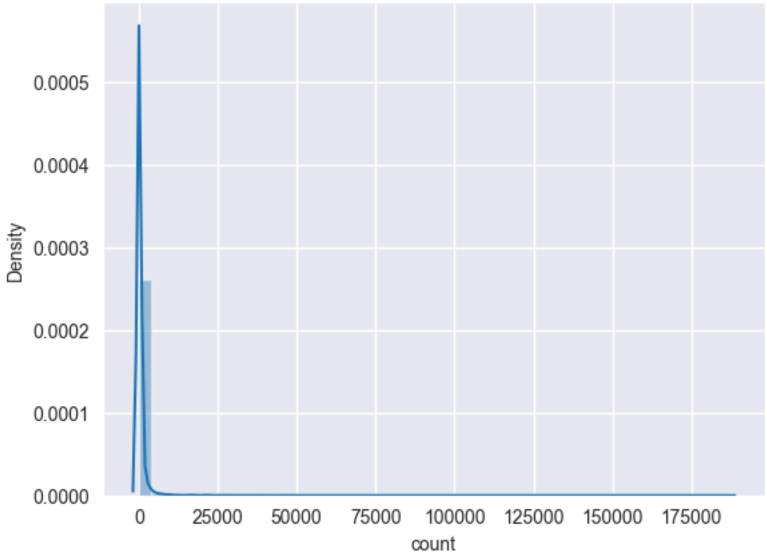


In [46]:
```
sns.set_style("darkgrid")
```

In [47]:
```
sns.distplot(cities_by_accident)
```

```
C:\Users\AYUSH\AppData\Local\Temp\ipykernel_54680\3405282844.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(cities_by_accident)
```

Out [47]:
```
<Axes: xlabel='count', ylabel='Density'>
```



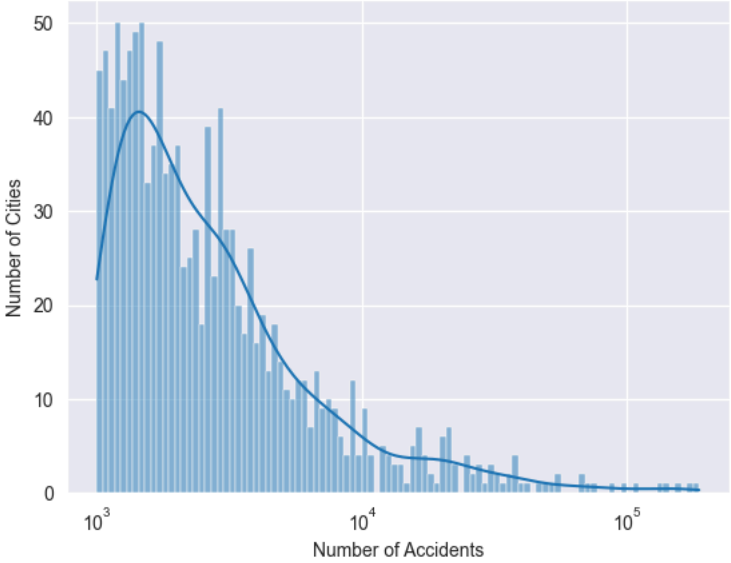
In [48]:
```
high_accident_cities = cities_by_accident[cities_by_accident >= 1000]
low_accident_cities = cities_by_accident[cities_by_accident < 1000]
```

In [49]:
```
len(high_accident_cities)/len(cities_by_accident)
```

Out [49]:
```
0.08904810644831115
```

In [51]:
```
# Distribution II(Accidents are high)
sns.histplot(high_accident_cities,bins=100,log_scale=True,kde=True,).set(xlabel='Number of Accid
```

Out [51]:
```
[Text(0.5, 0, 'Number of Accidents'), Text(0, 0.5, 'Number of Cities')]
```



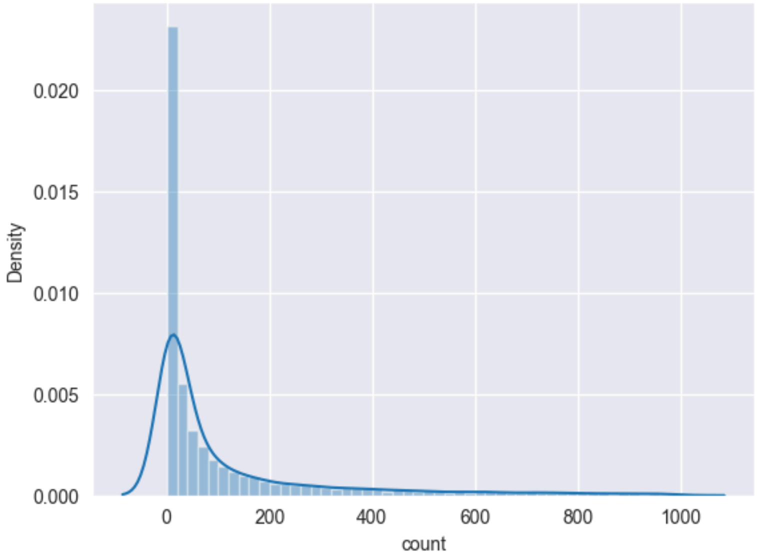
In [52]:
```
cities_by_accident[cities_by_accident==1]
```

Out [52]:
```
City
American Fork-Pleasant Grove    1
Waldoboro                       1
Kinsley                         1
Killona                         1
Jeanerette                      1
                               ..
Rapid River                     1
Cat Spring                      1
Glenwood City                   1
Downing                         1
Marfa                           1
Name: count, Length: 1023, dtype: int64
```

In [53]:
```
sns.distplot(low_accident_cities)
```

```
C:\Users\AYUSH\AppData\Local\Temp\ipykernel_54680\469555131.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(low_accident_cities)
```

Out [53]:
```
<Axes: xlabel='count', ylabel='Density'>
```

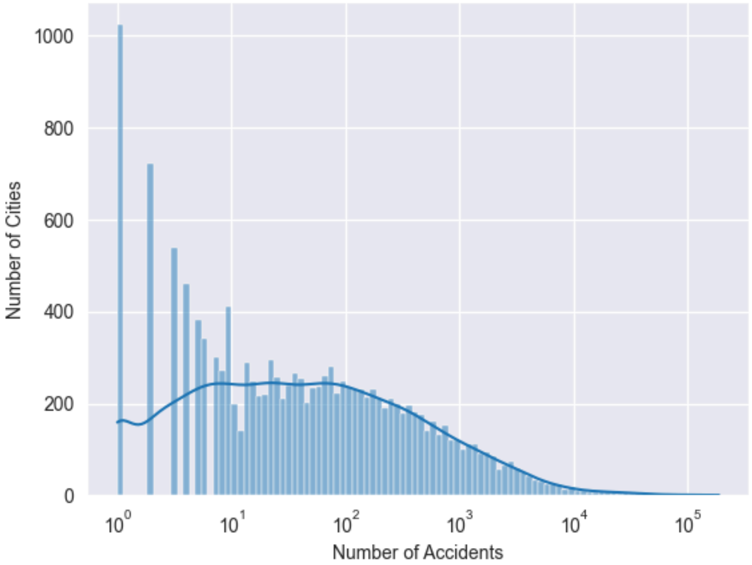


In [54]:
```
# Number of Accidents reported in New York
(df[df['City']== 'New York'].shape[0])
```

Out [54]:
```
21699
```

In [58]:
```
# Distribution I(Accidents-City Analysis)
sns.set_style('darkgrid')
sns.histplot(cities_by_accident,bins=100,log_scale=True,kde=True,).set(xlabel='Number of Acciden
```

Out [58]:
```
[Text(0.5, 0, 'Number of Accidents'), Text(0, 0.5, 'Number of Cities')]
```



Start Time

In [59]:
```
df.columns
```

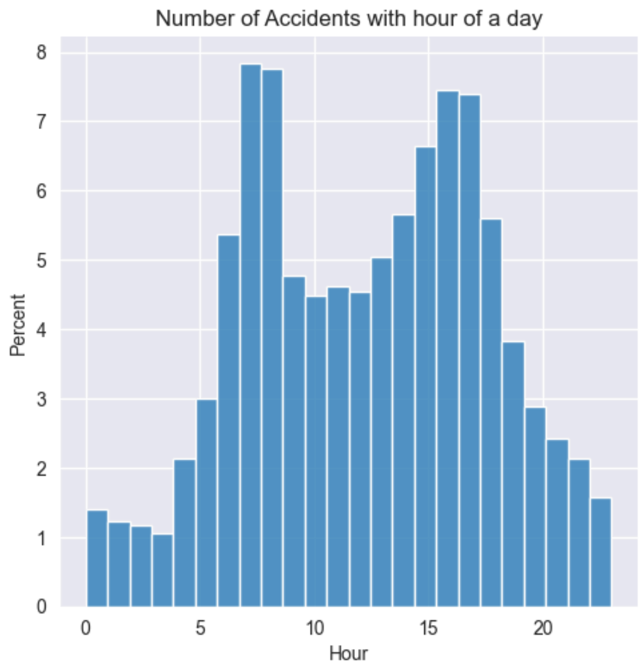
Out [59]:
```
Index(['ID', 'Source', 'Severity', 'Start_Time', 'End_Time', 'Start_Lat',
       'Start_Lng', 'Distance(mi)', 'Description', 'Street', 'City', 'County',
       'State', 'Zipcode', 'Country', 'Timezone', 'Airport_Code',
       'Weather_Timestamp', 'Temperature(F)', 'Humidity(%)', 'Pressure(in)',
       'Visibility(mi)', 'Wind_Direction', '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 [60]:
```
df.Start_Time[0]
```

Out [60]:
```
'2016-02-08 05:46:00'
```

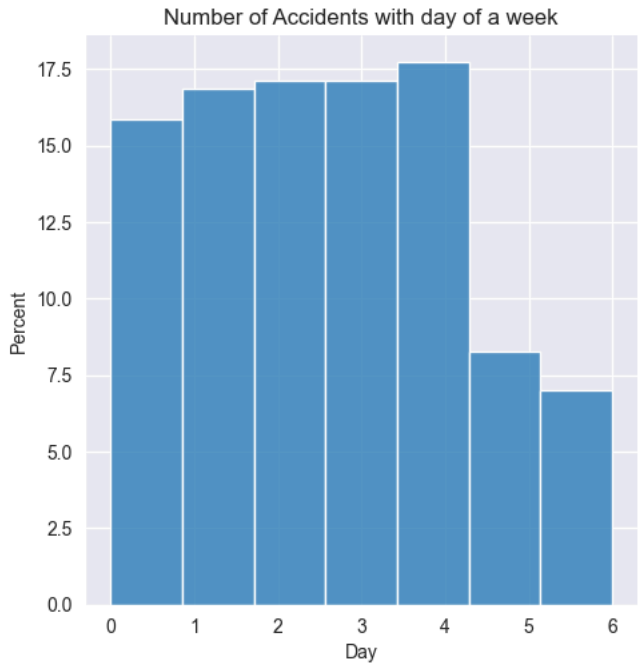
In [61]:
```
# Datatype Conversion
time=df['Start_Time']
time = pd.to_datetime(time, errors='coerce')
```

In [62]:
```
# # Number of Accidents with hour of a day
sns.displot(time.dt.hour, bins=24,stat='percent')
plt.title('Number of Accidents with hour of a day')
plt.xlabel('Hour')
plt.show()
```



High Percentage Of Accidents occur between 6 to 10 am.(Probably,people are in a hurry to get to work) Next Highest percentage is 3 to 6 pm.

In [63]:
```
# Number of Accidents with day of a week
sns.displot(time.dt.dayofweek, bins=7,stat='percent')
plt.title('Number of Accidents with day of a week')
plt.xlabel('Day')
plt.show()
```



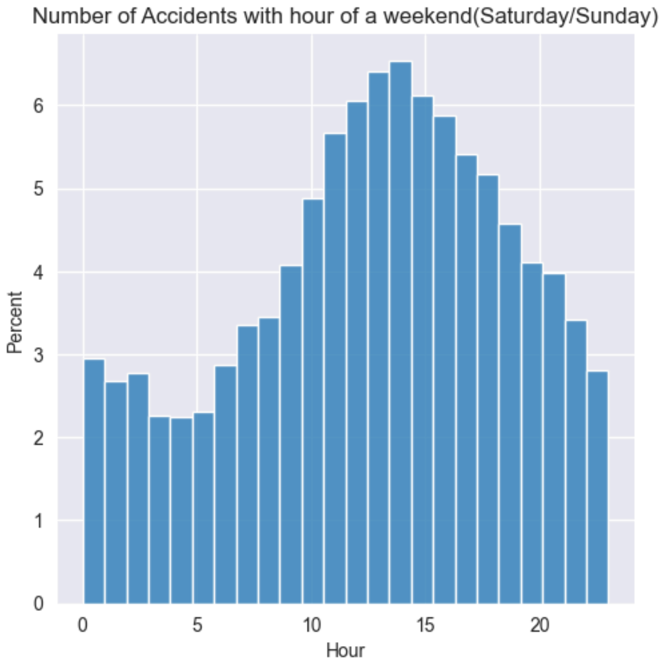
Is the distribution of accidents by hour the same on weekends as on weekdays.

In [64]:
```
# Weekends-Accident Analysis
weekends = time[(time.dt.dayofweek==6) | (time.dt.dayofweek==5)]
weekends
```

Out [64]:
```
129       2016-02-13 11:05:00
130       2016-02-13 11:05:21
131       2016-02-13 11:17:01
132       2016-02-13 11:25:42
133       2016-02-13 12:56:31
                  ...
7726137   2019-08-18 23:24:10
7726177   2019-08-17 03:36:35
7726252   2019-08-18 22:56:56
7726253   2019-08-18 22:56:56
7726292   2019-08-18 22:54:41
Name: Start_Time, Length: 1069268, dtype: datetime64[ns]
```

In [65]:

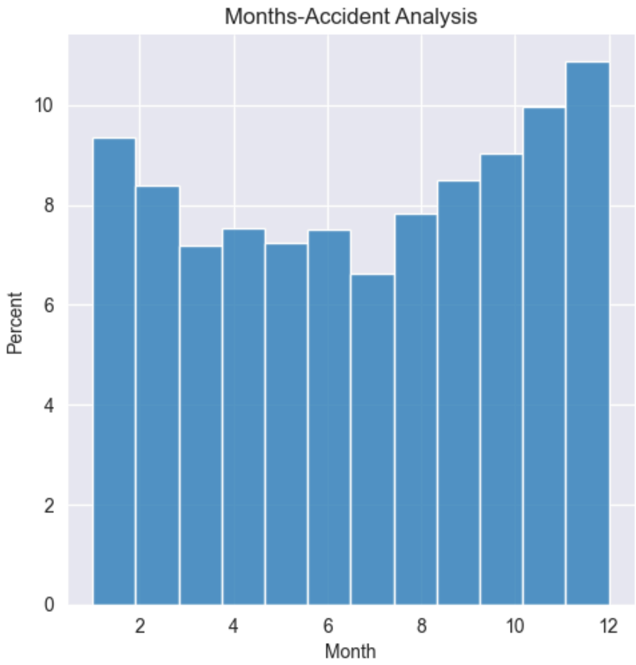
```
# Number of Accidents with hour of a weekend(Saturday/Sunday)
sns.displot(weekends.dt.hour, bins=24,stat='percent')
plt.title('Number of Accidents with hour of a weekend(Saturday/Sunday)')
plt.xlabel('Hour')
plt.show()
```



On weekends,peak occur between 10 am to 3 pm ,unlike weekdays.

In [66]:

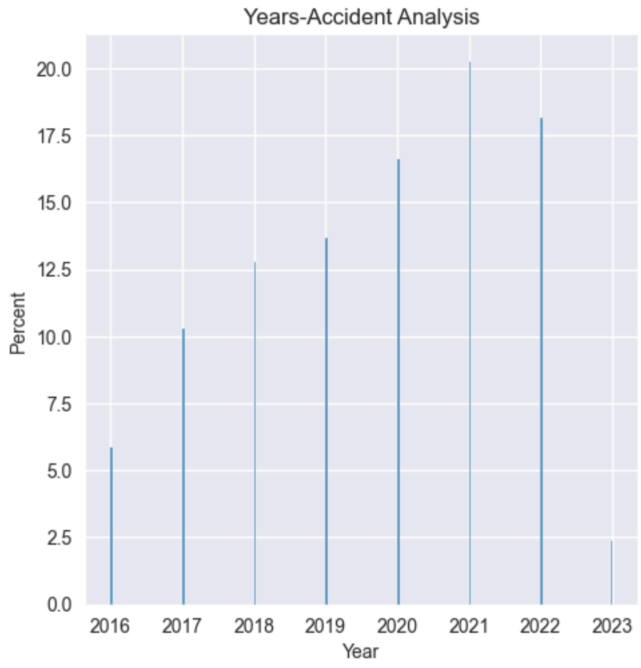
```
# Months-Accident Analysis
sns.displot(time.dt.month, bins=12,stat='percent')
plt.title('Months-Accident Analysis')
plt.xlabel('Month')
plt.show()
```



The majority of accidents tend to occur at the end of the year.

In [67]:

```
# Years-Accident Analysis
sns.displot(time.dt.year,stat='percent')
plt.title('Years-Accident Analysis')
plt.xlabel('Year')
plt.show()
```



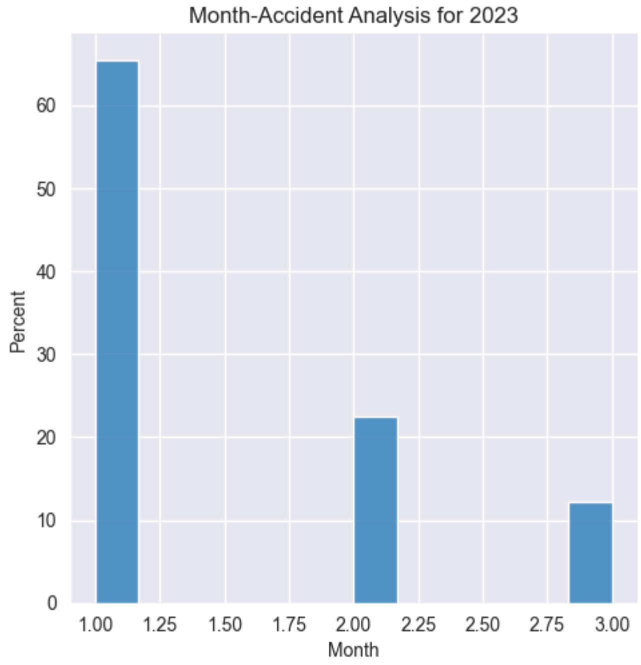
The number of accidents has been increasing each year, yet 2023 recorded the lowest number of accidents. This is highly unusual. We need to examine the data for 2023.

In [68]:

```
#  2023 Analysis

time_2023 = time[time.dt.year==2023]
```

In [69]:

```
# Visualization Plot
sns.displot(time_2023.dt.month,bins = 12,stat='percent')
plt.title('Month-Accident Analysis for 2023')
plt.xlabel('Month')
plt.show()
```



The reason for the seemingly lower number of accidents in 2023 is primarily due to data availability

In [70]:

```
df.Source
```

Out [70]:

```
0          Source2
1          Source2
2          Source2
3          Source2
4          Source2
            ...
7728389    Source1
7728390    Source1
7728391    Source1
7728392    Source1
```

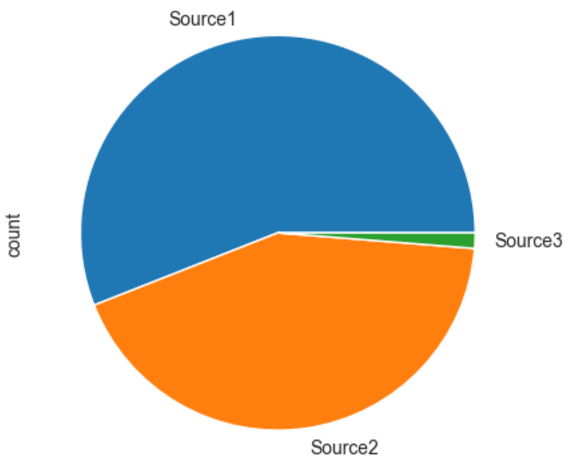
```
7728393    Source1
Name: Source, Length: 7728394, dtype: object
```

In [71]:

```
df.Source.value_counts().plot(kind='pie')
```

Out [71]:

```
<Axes: ylabel='count'>
```



Start Latitude & Longitude

In [72]:

```
df.Start_Lng
```

Out [72]:

```
0          -84.058723
1          -82.831184
2          -84.032608
3          -84.205582
4          -84.188354
              ...
7728389   -117.379360
7728390   -117.148060
7728391   -117.847790
7728392   -118.403020
7728393   -117.230920
Name: Start_Lng, Length: 7728394, dtype: float64
```

In [73]:

```
df.Start_Lat
```

Out [73]:

```
0          39.865147
1          39.928059
2          39.063148
3          39.747753
4          39.627781
             ...
7728389    34.002480
7728390    32.766960
7728391    33.775450
7728392    33.992460
7728393    34.133930
Name: Start_Lat, Length: 7728394, dtype: float64
```

In [74]:

```
# Latitude
lat = df['Start_Lat']
lat
```

Out [74]:

```
0          39.865147
1          39.928059
2          39.063148
3          39.747753
4          39.627781
             ...
7728389    34.002480
7728390    32.766960
7728391    33.775450
7728392    33.992460
7728393    34.133930
Name: Start_Lat, Length: 7728394, dtype: float64
```

In [75]:

```
# Longitude
lon = df['Start_Lng']
lon
```

Out [75]:

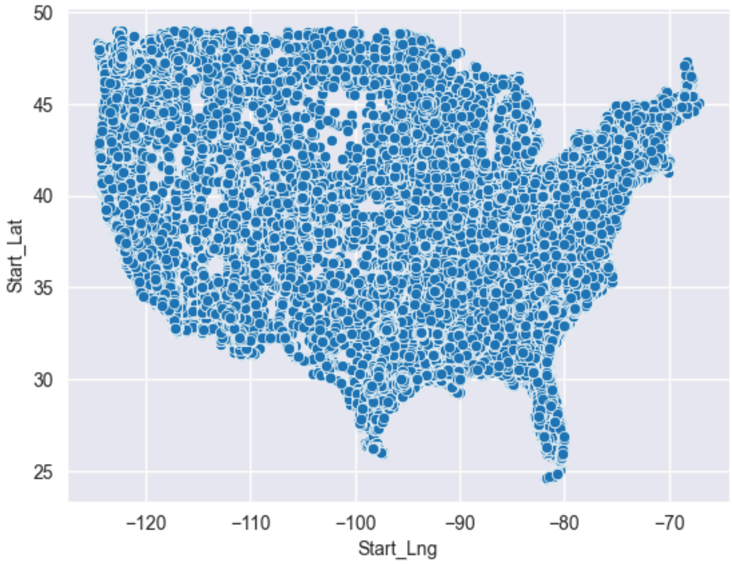
```
0          -84.058723
1          -82.831184
2          -84.032608
3          -84.205582
4          -84.188354
              ...
7728389   -117.379360
7728390   -117.148060
7728391   -117.847790
7728392   -118.403020
7728393   -117.230920
Name: Start_Lng, Length: 7728394, dtype: float64
```

In [76]:

```
# Approximate Map of US
sns.scatterplot(x=df.Start_Lng,y=df.Start_Lat)
```

Out [76]:

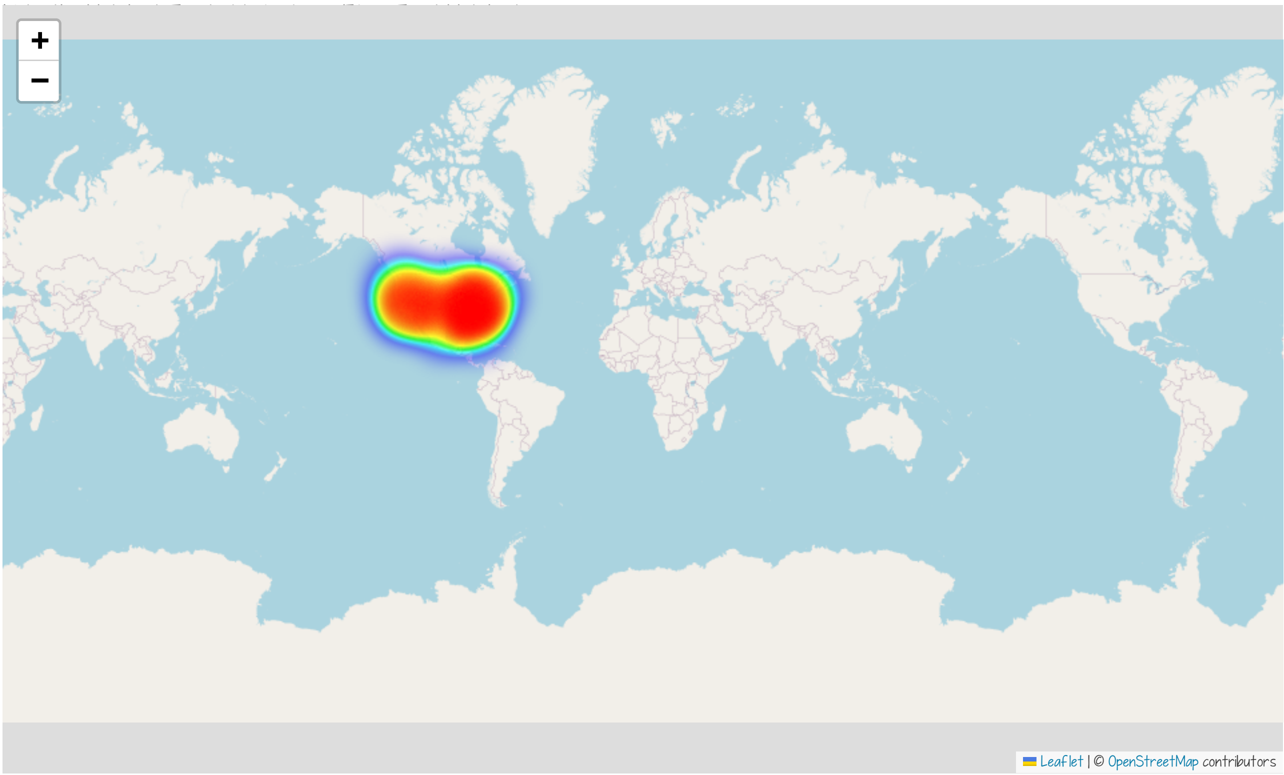
```
<Axes: xlabel='Start_Lng', ylabel='Start_Lat'>
```



In [78]:

```
# Using folium to visualize the location of accidents on the map
import folium

from folium.plugins import HeatMap
lat_lon_pairs = list(zip(list(lat), list(lon)))

map = folium.Map()
HeatMap(lat_lon_pairs).add_to(map)
map
```

Out [78]:



State

In [79]:

```
state = df['State']
```

In [80]:

```
state.nunique()
```

Out [80]:

```
49
```

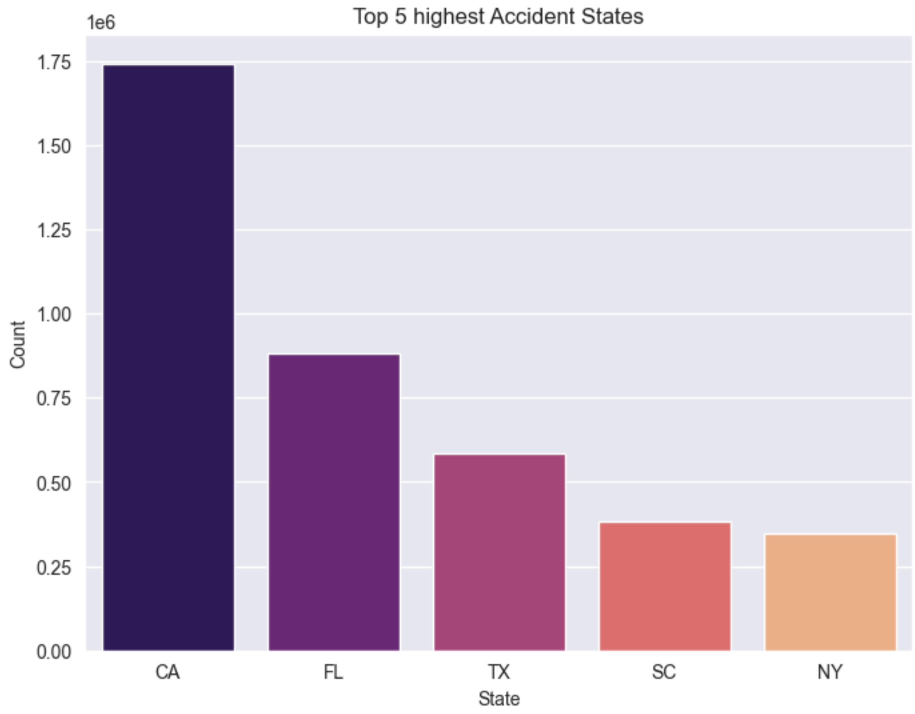
In [81]:

```
# The top 5 states with the highest number of accidents
top_state = state.value_counts().head(5).reset_index()
```

In [82]:

```
# Visualization Plot
plt.figure(figsize=(8, 6))
sns.barplot(x='State', y='count', data=top_state, palette="magma")
plt.title('Top 5 highest Accident States')
plt.xlabel('State')
plt.ylabel('Count')
plt.show()
```

```
C:\Users\AYUSH\AppData\Local\Temp\ipykernel_54680\941678388.py:3: 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.

  sns.barplot(x='State', y='count', data=top_state, palette="magma")
```



Temperature

In [83]:

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

Out [83]:

```
count    7.564541e+06
mean     6.166329e+01
std      1.901365e+01
min     -8.900000e+01
25%      4.900000e+01
50%      6.400000e+01
75%      7.600000e+01
max      2.070000e+02
Name: Temperature(F), dtype: float64
```

In [84]:

```
# Checking the maximum recorded temperature
df[df['Temperature(F)']==207]
```

Out [84]:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street	...	Roundabout
4208142	A-4239885	Source1	2	2023-02-22 19:24:00.000000000	2023-02-22 21:32:35.000000000	34.066720	-117.238034	0.021	Accident on I-10 W from California St (I-10) t...	I-10 W	...	False
4546366	A-4580836	Source1	2	2023-02-22 20:10:00.000000	2023-02-23 00:01:38.000000	34.228800	-117.301024	0.417	Accident on CA-18 from N Sierra Way (N Waterma...	Rim of the World Hwy	...	False
5029338	A-5067455	Source1	4	2023-02-22 19:30:00	2023-02-23 00:00:53	34.228728	-117.251234	0.543	CA-189 is closed from CA-18/Lake Gregory Dr (C...	Lake Gregory Dr	...	False

3 rows × 41 columns

In [85]:

```
temperature = df['Temperature(F)'].value_counts().reset_index()
```

In [86]:

```
temperature = temperature.round(2)
temperature
```

Out [86]:

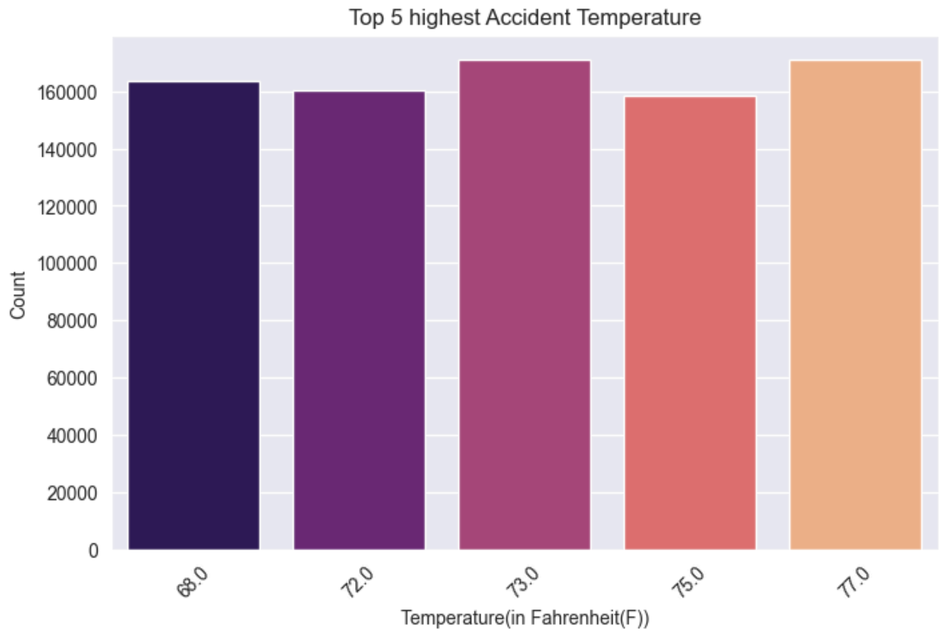
	Temperature(F)	count
0	77.0	170991
1	73.0	170898
2	68.0	163767
3	72.0	160498
4	75.0	158448
...	...	...
855	-32.8	1
856	-9.2	1
857	-17.9	1
858	168.8	1
859	-12.1	1

860 rows × 2 columns

In [87]:

```
plt.figure(figsize=(8, 5))
sns.barplot(x='Temperature(F)', y='count', data=temperature.head(5), palette="magma")
plt.title('Top 5 highest Accident Temperature')
plt.xlabel('Temperature(in Fahrenheit(F))')
plt.ylabel('Count')
plt.xticks(rotation=45,ha='center')
plt.show()
```

```
C:\Users\AYUSH\AppData\Local\Temp\ipykernel_54680\3890819157.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.

  sns.barplot(x='Temperature(F)', y='count', data=temperature.head(5), palette="magma")
```



In [88]:

```
# Checking for data inconsistencies
temperature[temperature['Temperature(F)']> 140].sum()
```

Out [88]:

```
Temperature(F)    2272.6
count               22.0
dtype: float64
```

Weather

In [89]:

```
df['Weather_Condition']
```

Out [89]:

```
0              Light Rain
1              Light Rain
2                Overcast
3           Mostly Cloudy
4           Mostly Cloudy
```

```
                      ...
7728389               Fair
7728390               Fair
7728391      Partly Cloudy
7728392               Fair
7728393               Fair
Name: Weather_Condition, Length: 7728394, dtype: object
```

In [90]:
```
weather = df['Weather_Condition'].value_counts().reset_index()
```

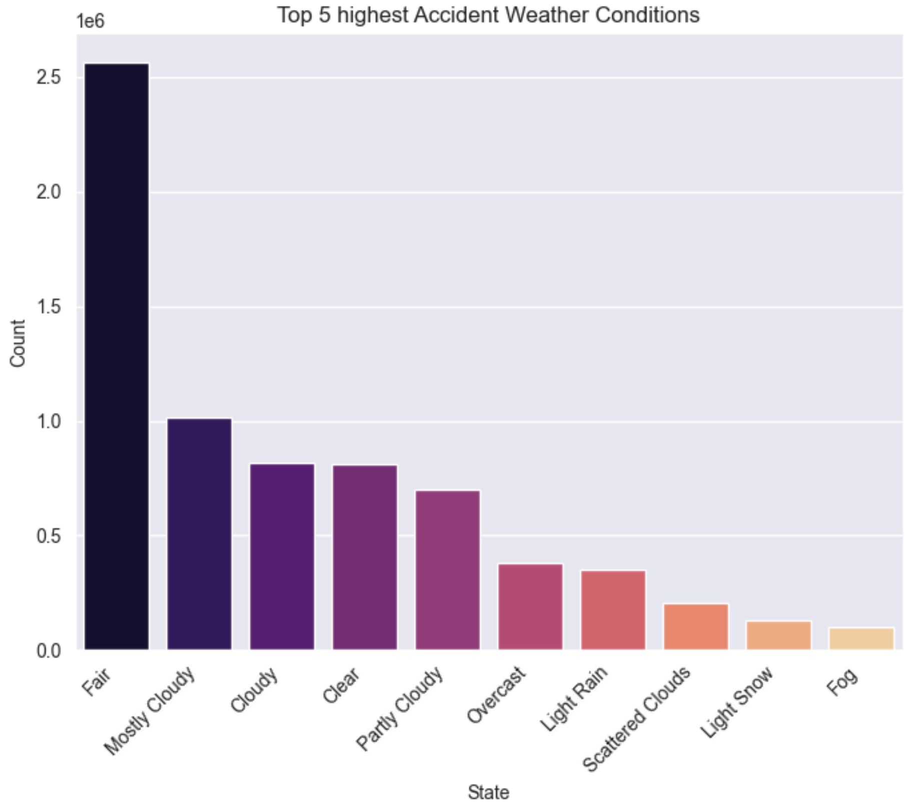
In [91]:
```
# Top 10 Weather Conditions with most number of Accidents
weather = weather.head(10)
weather
```

Out [91]:

	Weather_Condition	count
0	Fair	2560802
1	Mostly Cloudy	1016195
2	Cloudy	817082
3	Clear	808743
4	Partly Cloudy	698972
5	Overcast	382866
6	Light Rain	352957
7	Scattered Clouds	204829
8	Light Snow	128680
9	Fog	99238

In [92]:
```
# Visualization Plot
plt.figure(figsize=(8, 6))
sns.barplot(x='Weather_Condition', y='count', data=weather, palette="magma")
plt.title('Top 5 highest Accident Weather Conditions')
plt.xlabel('State')
plt.ylabel('Count')
plt.xticks(rotation=45,ha='right')
plt.show()
```

```
C:\Users\AYUSH\AppData\Local\Temp\ipykernel_54680\2376085648.py:3: 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.

  sns.barplot(x='Weather_Condition', y='count', data=weather, palette="magma")
```



Impact of Weather on Accident Severity:

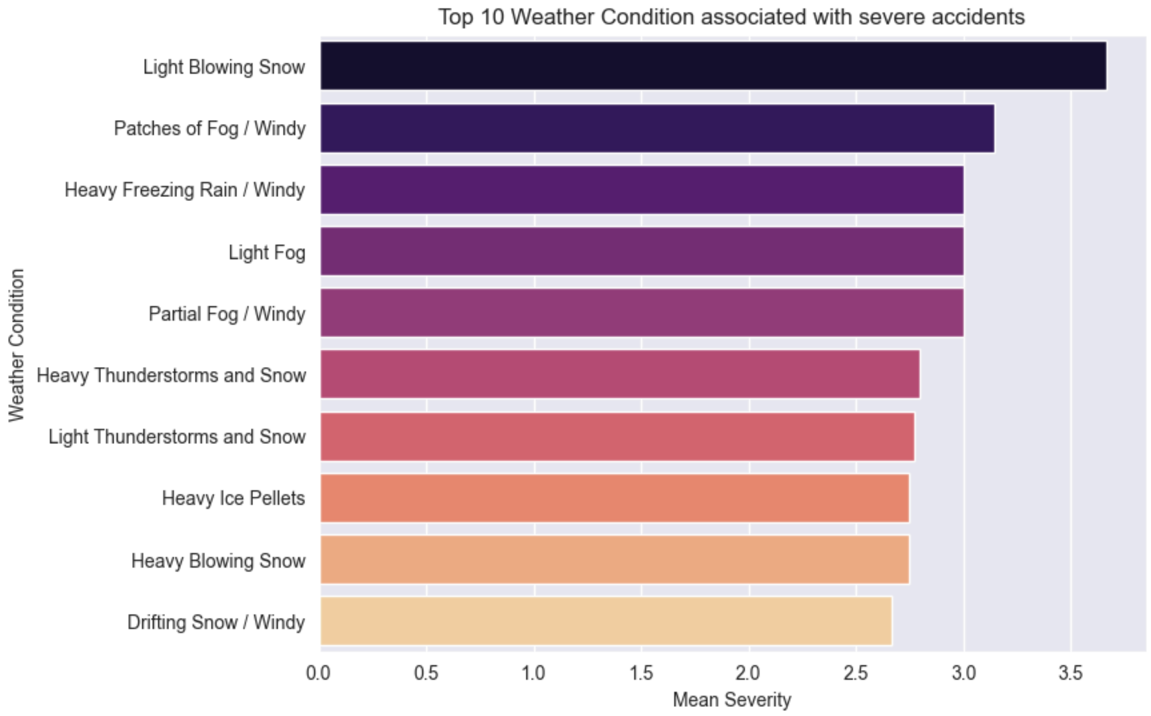
In [93]:

```
# Creating a DataFrame that summarizes the mean severity for each weather condition by grouping.
severity_weather = df.groupby('Weather_Condition')['Severity'].mean().reset_index().sort_values(
```

In [94]:

```
# Visualisation of the Top 10 Weather Conditions that are associated with severe accidents.
plt.figure(figsize=(8, 6))
sns.barplot(x='Severity', y='Weather_Condition', data=severity_weather[0:10], palette="magma")
plt.title('Top 10 Weather Condition associated with severe accidents')
plt.xlabel('Mean Severity')
plt.ylabel('Weather Condition')
plt.show()
```

```
C:\Users\AYUSH\AppData\Local\Temp\ipykernel_54680\3304834884.py:3: 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(x='Severity', y='Weather_Condition', data=severity_weather[0:10], palette="magma")
```



Ask & Answer Questions

1. are there more accidents in warmer and cooler areas?
2. which 5 states have the highest numbers of accidents? how about per capita?
3. does New York show up in the data? if yes, why the count lower if this the most populated city?
4. Among top 100 cities in number of accidents,which states do they belong the most frequently?
5. What time of the day are accidents most frequent in ?
6. which day of the week have the most accidents?
7. which months have the most accidents?
8. what is the trends of accidents over the years (increase/decrease)?
9. when is accidents per unit of traffic the highest.

Summary & Conclusions

1. Approximately 9% of the cities experience more than 1,000 accidents.

2. About 1,023 cities (7%) have reported only one accident, suggesting the presence of potential outliers that may need to be addressed.

3. Miami leads with the highest number of accidents among the top 20 cities, while Columbia has the fewest accidents in this group.

4. New York accounts for only 21,699 accidents, which is approximately 0.28% of the total number of accidents in the US, despite being the most populated city in the country.

5. California, Florida, Texas, South Carolina, and New York emerge as the top 5 states with the highest number of accidents.

6. A high percentage of accidents occur between between 6 AM to 10 AM.

7. On Weekends, high percentage of accidents occur between 12PM to 4 PM with the peak accidents around 2PM.

8. Weekdays show a higher number of accidents compared to weekends.

9. The majority of accidents tend to occur at the end of the year.

10. The dataset used for analysis only includes information up to March 2023, which means it does not encompass the entire year. Consequently, the observed decrease in accidents for 2023 may not accurately represent the true accident rate for the entire year, as data for the subsequent months is missing. Therefore, any conclusions about accident trends in 2023 should be approached with caution, given the incomplete data for that year.

11. The top 10 most common weather conditions during accidents include: Fair, Mostly Cloudy, Cloudy, Clear, Partly Cloudy, Overcast, Light Rain, Scattered Clouds, Light Snow, and Fog.
12. Light Blowing Snow exhibits the highest average severity at 3.67.
13. Partial Fog with Wind, Heavy Thunderstorms and Snow, Light Thunderstorms and Snow, Heavy Ice Pellets, Heavy Blowing Snow, and Drifting Snow with Wind also demonstrate notable average severity levels.
14. Some weather conditions can significantly reduce visibility and create hazardous driving conditions. Heavy Thunderstorms and Snow are characterized by the lowest visibility among all weather conditions.
15. The initial analysis suggests a potential correlation between temperature and accident frequency within the range of 61.66°F and 77°F.
16. The dataset does contain approximately 22 entries that appear to be outliers based on expected temperature ranges.(The maximum reported temperature of US is around 134°F)
17. There is a seasonal variation in accidents, with fewer incidents during the summer and an increasing trend as winter approaches.
18. Accident rates are higher in coastal/bay areas than in inland regions.

Recommendations:

1. Safety Campaigns
    - Focus on cities with over 1,000 accidents.
    - Enhance awareness and enforcement during peak times (12 PM-4 PM weekends, 6 AM-10 AM weekdays).
2. Outlier Review
    - Investigate the 1,023 cities with only one accident for data accuracy.
3. Urban Traffic Management
    - Monitor and improve traffic strategies in major cities like New York.
4. State-Specific Programs
    - Develop safety programs for California, Florida, Texas, South Carolina, and New York.
5. Weather-Related Measures
    - Improve road safety and visibility during severe weather conditions.
    - Enhance road lighting in fog-prone areas.
6. Seasonal Adjustments
    - Increase safety measures for winter; analyze factors for fewer summer accidents.
7. Temperature Analysis
    - Investigate temperature-related accident trends and address outlier data.
8. Coastal Area Safety
    - Implement additional safety measures in coastal and bay areas.
9. Data Completeness
    - Ensure future datasets include full-year data for accurate trend analysis.