Exploratory Data Analysis (EDA) on US Accidents Dataset

Step 1: Importing the required libraries

In [43]:

```
# Essential libraries for EDA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import scipy
```

Step 2: Loading the Data

In [3]:

```
#Load dataset
df = pd.read_csv('us-accidents-sample-1M.csv')
print(df)
```

```
               ID   Source  Severity           Start_Time  \
0             A-3  Source2         2  2016-02-08 06:49:27
1             A-6  Source2         3  2016-02-08 07:44:26
2             A-7  Source2         2  2016-02-08 07:59:35
3            A-15  Source2         2  2016-02-08 08:39:43
4            A-39  Source2         2  2016-02-09 05:17:08
...           ...      ...       ...                  ...
999995  A-7777735  Source1         4  2019-08-23 13:39:48
999996  A-7777747  Source1         2  2019-08-23 16:26:06
999997  A-7777748  Source1         4  2019-08-23 16:51:29
999998  A-7777750  Source1         2  2019-08-23 17:10:58
999999  A-7777758  Source1         2  2019-08-23 19:11:30

                   End_Time  Start_Lat   Start_Lng   End_Lat    End_Lng  \
0       2016-02-08 07:19:27  39.063148  -84.032608       NaN        NaN
1       2016-02-08 08:14:26  40.100590  -82.925194       NaN        NaN
2       2016-02-08 08:29:35  39.758274  -84.230507       NaN        NaN
3       2016-02-08 09:09:43  39.972038  -82.913521       NaN        NaN
4       2016-02-09 05:47:08  39.782578  -84.178688       NaN        NaN
...                     ...        ...         ...       ...        ...
999995  2019-08-23 14:05:33  33.687300 -117.890190  33.68599 -117.88626
999996  2019-08-23 16:54:36  34.030470 -117.598170  34.03050 -117.58860
999997  2019-08-23 17:21:02  33.779130 -117.887980  33.77991 -117.89086
999998  2019-08-23 17:40:28  33.850800 -117.843650  33.85075 -117.83745
999999  2019-08-23 19:38:23  32.766960 -117.148060  32.76555 -117.15363

        Distance(mi)  ... Roundabout Station   Stop Traffic_Calming  \
0              0.010  ...      False   False  False           False
1              0.010  ...      False   False  False           False
2              0.000  ...      False   False  False           False
3              0.010  ...      False   False  False           False
4              0.010  ...      False   False  False           False
...              ...  ...        ...     ...    ...             ...
999995         0.243  ...      False   False  False           False
999996         0.548  ...      False   False  False           False
999997         0.174  ...      False   False  False           False
999998         0.356  ...      False   False  False           False
999999         0.338  ...      False   False  False           False

       Traffic_Signal Turning_Loop Sunrise_Sunset Civil_Twilight  \
0                True        False          Night          Night
1               False        False            Day            Day
2               False        False            Day            Day
3                True        False            Day            Day
4               False        False          Night          Night
...               ...          ...            ...            ...
999995          False        False            Day            Day
999996          False        False            Day            Day
999997          False        False            Day            Day
999998          False        False            Day            Day
999999          False        False            Day            Day

       Nautical_Twilight Astronomical_Twilight
0                    Day                   Day
```

```
1                    Day                 Day
2                    Day                 Day
3                    Day                 Day
4                  Night               Night
...                  ...                 ...
999995               Day                 Day
999996               Day                 Day
999997               Day                 Day
999998               Day                 Day
999999               Day                 Day

[1000000 rows x 46 columns]
```

In [7]:
```python
#Initial inspection
print("Dataset shape: ", df.shape)
print("\nFirst 5 rows: ")
df.head()
```

```
Dataset shape:  (1000000, 46)

First 5 rows:
```

Out[7]:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_La
0	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	NaI
1	A-6	Source2	3	2016-02-08 07:44:26	2016-02-08 08:14:26	40.100590	-82.925194	NaI
2	A-7	Source2	2	2016-02-08 07:59:35	2016-02-08 08:29:35	39.758274	-84.230507	NaI
3	A-15	Source2	2	2016-02-08 08:39:43	2016-02-08 09:09:43	39.972038	-82.913521	NaI
4	A-39	Source2	2	2016-02-09 05:17:08	2016-02-09 05:47:08	39.782578	-84.178688	NaI

5 rows × 46 columns

Step 3: Initial Data Exploration

In [8]:
```python
# 1. Get the information about dataset
print("Data types and non-null counts: ")
print(df.info())
```

```
Data types and non-null counts:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 46 columns):
 #   Column                 Non-Null Count    Dtype  
---  ------                 --------------    -----  
 0   ID                     1000000 non-null  object 
 1   Source                 1000000 non-null  object 
 2   Severity               1000000 non-null  int64  
 3   Start_Time             1000000 non-null  object 
 4   End_Time               1000000 non-null  object 
 5   Start_Lat              1000000 non-null  float64
 6   Start_Lng              1000000 non-null  float64
 7   End_Lat                560122 non-null   float64
 8   End_Lng                560122 non-null   float64
 9   Distance(mi)           1000000 non-null  float64
 10  Description            999999 non-null   object 
 11  Street                 998532 non-null   object 
 12  City                   999970 non-null   object 
 13  County                 1000000 non-null  object 
 14  State                  1000000 non-null  object 
 15  Zipcode                999745 non-null   object 
 16  Country                1000000 non-null  object 
 17  Timezone               998983 non-null   object 
 18  Airport_Code           997089 non-null   object 
 19  Weather_Timestamp      984486 non-null   object 
 20  Temperature(F)         978905 non-null   float64
 21  Wind_Chill(F)          741428 non-null   float64
 22  Humidity(%)            977637 non-null   float64
 23  Pressure(in)           981869 non-null   float64
 24  Visibility(mi)         977124 non-null   float64
 25  Wind_Direction         977275 non-null   object 
 26  Wind_Speed(mph)        926201 non-null   float64
 27  Precipitation(in)      715363 non-null   float64
 28  Weather_Condition      977581 non-null   object 
 29  Amenity                1000000 non-null  bool   
 30  Bump                   1000000 non-null  bool   
 31  Crossing               1000000 non-null  bool   
 32  Give_Way               1000000 non-null  bool   
 33  Junction               1000000 non-null  bool   
 34  No_Exit                1000000 non-null  bool   
 35  Railway                1000000 non-null  bool   
 36  Roundabout             1000000 non-null  bool   
 37  Station                1000000 non-null  bool   
 38  Stop                   1000000 non-null  bool   
 39  Traffic_Calming        1000000 non-null  bool   
 40  Traffic_Signal         1000000 non-null  bool   
 41  Turning_Loop           1000000 non-null  bool   
 42  Sunrise_Sunset         996947 non-null   object 
 43  Civil_Twilight         996947 non-null   object 
 44  Nautical_Twilight      996947 non-null   object 
 45  Astronomical_Twilight  996947 non-null   object 
dtypes: bool(13), float64(12), int64(1), object(20)
memory usage: 264.2+ MB
```

None

In [10]:
```
# 2. Get the statistical information
print("Statistical summary: ")
display(df.describe())  # display function for better readability
```

Statistical summary:

	Severity	Start_Lat	Start_Lng	End_Lat	End_
count	1000000.000000	1000000.000000	1000000.000000	560122.000000	560122.00(
mean	2.212728	36.210188	-94.697790	36.268519	-95.70ɛ
std	0.488370	5.078645	17.388088	5.275009	18.095
min	1.000000	24.555269	-124.497567	24.569978	-124.497
25%	2.000000	33.406755	-117.213699	33.462100	-117.74(
50%	2.000000	35.831512	-87.778374	36.187096	-88.022
75%	2.000000	40.096065	-80.351738	40.191142	-80.24¢
max	4.000000	49.000759	-67.403551	49.000760	-67.403

In [13]:
```
#for both numerical and catagorical columns
display(df.describe(include='all').transpose())
```

	count	unique	top	freq	mean	
ID	1000000	1000000	A-3	1	NaN	
Source	1000000	3	Source1	560122	NaN	
Severity	1000000.0	NaN	NaN	NaN	2.212728	0.48
Start_Time	1000000	952419	2021-01-26 16:16:13	27	NaN	
End_Time	1000000	976257	2017-05-15 15:22:55	12	NaN	
Start_Lat	1000000.0	NaN	NaN	NaN	36.210188	5.078
Start_Lng	1000000.0	NaN	NaN	NaN	-94.69779	17.388
End_Lat	560122.0	NaN	NaN	NaN	36.268519	5.275
End_Lng	560122.0	NaN	NaN	NaN	-95.708158	18.095
Distance(mi)	1000000.0	NaN	NaN	NaN	0.562804	1.78
Description	999999	747382	A crash has occurred causing no to minimum del...	1271	NaN	
Street	998532	130147	I-95 N	10226	NaN	
City	999970	10808	Miami	24353	NaN	
County	1000000	1705	Los Angeles	68263	NaN	
State	1000000	49	CA	225116	NaN	
Zipcode	999745	217017	91761	1480	NaN	
Country	1000000	1	US	1000000	NaN	
Timezone	998983	4	US/Eastern	463163	NaN	
Airport_Code	997089	1964	KCQT	15205	NaN	
Weather_Timestamp	984486	387171	2022-03-13 01:53:00	159	NaN	
Temperature(F)	978905.0	NaN	NaN	NaN	61.657002	19.016
Wind_Chill(F)	741428.0	NaN	NaN	NaN	58.22479	22.385
Humidity(%)	977637.0	NaN	NaN	NaN	64.846766	22.813
Pressure(in)	981869.0	NaN	NaN	NaN	29.537666	1.005
Visibility(mi)	977124.0	NaN	NaN	NaN	9.087955	2.702
Wind_Direction	977275	24	CALM	124367	NaN	

	count	unique	top	freq	mean	
Wind_Speed(mph)	926201.0	NaN	NaN	NaN	7.690723	5.451
Precipitation(in)	715363.0	NaN	NaN	NaN	0.008318	0.103
Weather_Condition	977581	115	Fair	330917	NaN	
Amenity	1000000	2	False	987520	NaN	
Bump	1000000	2	False	999567	NaN	
Crossing	1000000	2	False	886756	NaN	
Give_Way	1000000	2	False	995343	NaN	
Junction	1000000	2	False	926173	NaN	
No_Exit	1000000	2	False	997461	NaN	
Railway	1000000	2	False	991231	NaN	
Roundabout	1000000	2	False	999976	NaN	
Station	1000000	2	False	973764	NaN	
Stop	1000000	2	False	972094	NaN	
Traffic_Calming	1000000	2	False	999040	NaN	
Traffic_Signal	1000000	2	False	852050	NaN	
Turning_Loop	1000000	1	False	1000000	NaN	
Sunrise_Sunset	996947	2	Day	690447	NaN	
Civil_Twilight	996947	2	Day	737088	NaN	
Nautical_Twilight	996947	2	Day	786350	NaN	
Astronomical_Twilight	996947	2	Day	825558	NaN	

In [21]:
```
# 3. Checking missing values
missing = df.isnull().sum().sort_values(ascending = False)
missing_percentage = (missing / len(df)) * 100
```

In [28]:
```
missing_df = pd.concat([missing, missing_percentage], keys=['No of missing val
display(missing_df[missing_df['Missing values %']>0])
```

	No of missing values	Missing values %
End_Lat	439878	43.9878
End_Lng	439878	43.9878
Precipitation(in)	284637	28.4637
Wind_Chill(F)	258572	25.8572
Wind_Speed(mph)	73799	7.3799
Visibility(mi)	22876	2.2876
Wind_Direction	22725	2.2725
Weather_Condition	22419	2.2419
Humidity(%)	22363	2.2363
Temperature(F)	21095	2.1095
Pressure(in)	18131	1.8131
Weather_Timestamp	15514	1.5514
Nautical_Twilight	3053	0.3053
Civil_Twilight	3053	0.3053
Sunrise_Sunset	3053	0.3053
Astronomical_Twilight	3053	0.3053
Airport_Code	2911	0.2911
Street	1468	0.1468
Timezone	1017	0.1017
Zipcode	255	0.0255
City	30	0.0030
Description	1	0.0001

What I explored :

-- Which columns have significant missing data?

-- What are the data types of each column?

-- For numeric columns: min/max/mean values that might reveal outliers

-- For categorical columns: number of unique values and their distribution

Step 4: Data Cleaning (Based on Initial Exploration)

After seeing the initial output, I need to:

-- Handle missing values(if required)

-- Convert data types if needed

-- Address any data quality issues

1. Handle missing values

In [55]:
```
# create copy of the dataframe
df_clean = df.copy()
```

In [56]:
```
# Too many missings in end coordinates(43%), so I'm going to drop the columns.
df_clean.drop(columns=['End_Lat', 'End_Lng'], axis= 1, inplace = True)
```

In [57]:
```
# For Weather data columns
weather_cols= ['Temperature(F)', 'Humidity(%)', 'Pressure(in)', 'Visibility(mi
for i in weather_cols:
    df_clean[i].fillna(df_clean[i].median(), inplace=True)  # Sometimes weathe
```

```
C:\Users\sanka\AppData\Local\Temp\ipykernel_4944\2343568762.py:4: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df_clean[i].fillna(df_clean[i].median(), inplace=True)  # Sometimes weather d
ata has outliers so I use median imputations
```

In [58]:
```
df_clean.isna().sum().sort_values(ascending=False)
```

Out[58]:
```
Precipitation(in)        284637
Wind_Chill(F)            258572
Wind_Direction            22725
Weather_Condition         22419
Weather_Timestamp         15514
Astronomical_Twilight      3053
Nautical_Twilight          3053
Civil_Twilight             3053
Sunrise_Sunset             3053
Airport_Code               2911
Street                     1468
Timezone                   1017
Zipcode                     255
City                         30
Description                   1
Bump                          0
Roundabout                    0
Station                       0
Traffic_Calming               0
Turning_Loop                  0
Railway                       0
Stop                          0
No_Exit                       0
Junction                      0
Give_Way                      0
Crossing                      0
Traffic_Signal                0
ID                            0
Amenity                       0
Wind_Speed(mph)               0
Source                        0
Pressure(in)                  0
Humidity(%)                   0
Temperature(F)                0
Country                       0
State                         0
County                        0
Distance(mi)                  0
Start_Lng                     0
Start_Lat                     0
End_Time                      0
Start_Time                    0
Severity                      0
Visibility(mi)                0
dtype: int64
```

In [59]:
```
# For categorical weather data
df_clean['Weather_Condition'].fillna('Unknown', inplace=True)
df_clean['Wind_Direction'].fillna('Unknown', inplace=True)
```

```
C:\Users\sanka\AppData\Local\Temp\ipykernel_4944\527200815.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df_clean['Weather_Condition'].fillna('Unknown', inplace=True)
C:\Users\sanka\AppData\Local\Temp\ipykernel_4944\527200815.py:3: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df_clean['Wind_Direction'].fillna('Unknown', inplace=True)
```

In [60]:
```
df_clean['Precipitation(in)'].unique()[:10]
```

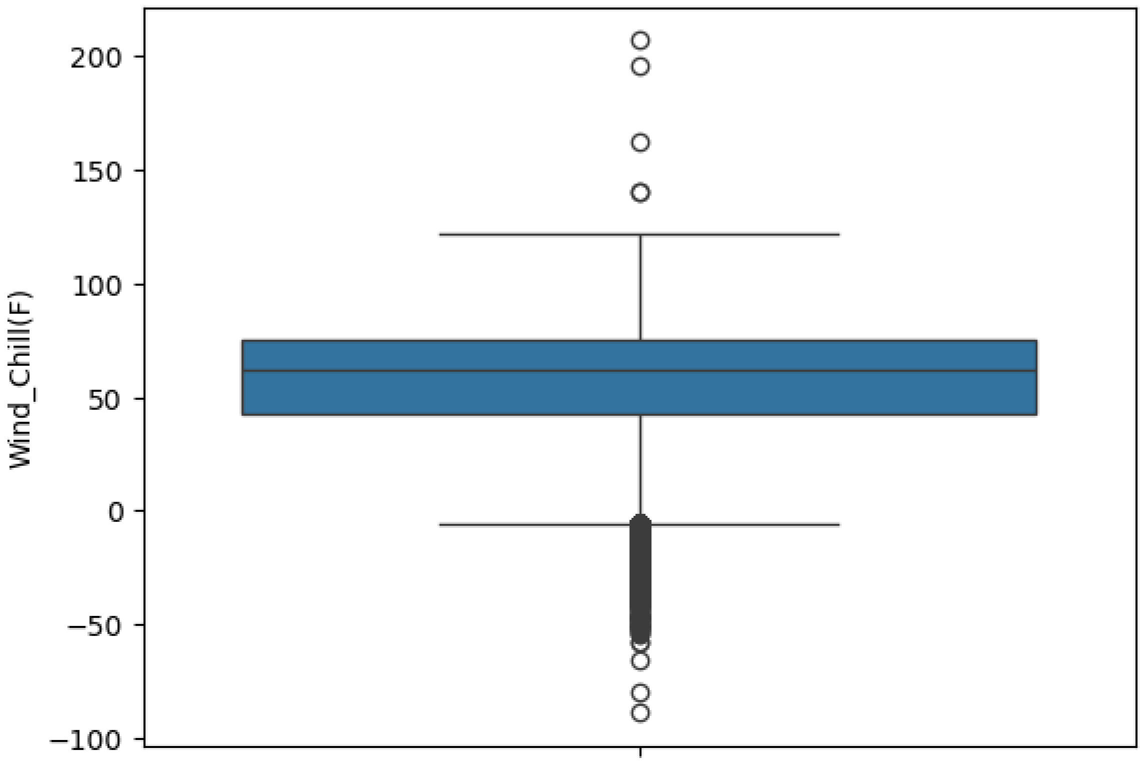
Out[60]:
```
array([ nan, 0.03, 0.02, 0.  , 0.01, 0.05, 0.22, 0.16, 0.08, 0.06])
```

In [61]:
```
# Precipitation values are missing a lot (28%), So I create a label for the va
df_clean['Precipitation_occured'] = df_clean['Precipitation(in)'].apply(lambda
df_clean.drop(columns=['Precipitation(in)'], axis = 1, inplace= True)
```

Checking 'Wind_Chill(F)' has outlier or not ?

In [68]:
```
sns.boxplot(data = df_clean, y = df_clean['Wind_Chill(F)'])
```

Out[68]:
```
<Axes: ylabel='Wind_Chill(F)'>
```



In [75]:

```
Q1 = df_clean['Wind_Chill(F)'].quantile(.25)
Q2 = df_clean['Wind_Chill(F)'].quantile(.50)
Q3 = df_clean['Wind_Chill(F)'].quantile(.75)

IQR = Q3 - Q1

lowerfence = Q1 - (1.5 * IQR)
upperfence = Q3 + (1.5 * IQR)

print('Q1: ', Q1)
print('Q2: ', Q2)
print('Q3: ', Q3)

print('IQR: ', IQR)

print('lowerfence: ', lowerfence)
print('upperfence: ', upperfence)

print('Min: ', df['Wind_Chill(F)'].min())
print('Max: ', df['Wind_Chill(F)'].max())

print('median: ', df_clean['Wind_Chill(F)'].median())
```

```
Q1:  42.6
Q2:  62.0
Q3:  75.0
IQR:  32.4
lowerfence:  -5.999999999999993
upperfence:  123.6
Min:  -89.0
Max:  207.0
median:  62.0
```

Outliers detected - So, I imputing the logic of Wind_Chill = Temperature(F) - (Wind_Speed(mph) * 0.7)

derive it from temperature when missing

In [77]:
```
def calculate_wind_chill(row):
    # Rule1: Only calculate if Wind_Chill is missing and Temperature exists
    if pd.isna(row['Wind_Chill(F)']) and not pd.isna(row['Temperature(F)']):
        # Rule2: Apply wind chill formula only if wind speed > 3 mph
        if row['Wind_Speed(mph)'] > 3:
            return row['Temperature(F)'] - (row['Wind_Speed(mph)'] * 0.7)   #S

        # Rule3: If wind speed <= 3 mph, wind chill = temperature
        return row['Temperature(F)']
    # Rule 4: Return original value if Wind_Chill isn't missing
    return row['Wind_Chill(F)']

df_clean['Wind_Chill(F)']= df_clean.apply(calculate_wind_chill, axis = 1)
```

In [85]:
```
# Other columns with minimal missingness (<2%)
df_clean.dropna(subset= ['Street', 'City', 'Zipcode', 'Timezone', 'Weather_Tim
```

2. Convert Data Types

In [84]:
```
df_clean.head()
```

Out[84]:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	Distanc
0	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	
1	A-6	Source2	3	2016-02-08 07:44:26	2016-02-08 08:14:26	40.100590	-82.925194	
2	A-7	Source2	2	2016-02-08 07:59:35	2016-02-08 08:29:35	39.758274	-84.230507	
3	A-15	Source2	2	2016-02-08 08:39:43	2016-02-08 09:09:43	39.972038	-82.913521	
4	A-39	Source2	2	2016-02-09 05:17:08	2016-02-09 05:47:08	39.782578	-84.178688	

5 rows × 44 columns

In [ ]:

```
# Convert timestamps
df_clean['Start_Time'] = pd.to_datetime(df_clean['Start_Time'], format='mixed'
df_clean['End_Time'] = pd.to_datetime(df_clean['End_Time'], format='mixed')
df_clean['Weather_Timestamp'] = pd.to_datetime(df_clean['Weather_Timestamp'],
```

In [93]:

```
# Clean text fields
df_clean['Description'] = df_clean['Description'].str.strip()
df_clean['City'] = df_clean['City'].str.strip()
```

In [94]:

```
# Check remaining missing values
print("Remaining missing values:")
print(df_clean.isnull().sum()[df_clean.isnull().sum() > 0])
```

```
Remaining missing values:
Series([], dtype: int64)
```

In [ ]:

```
#Save the cleaned file
df_clean.to_csv('US_Accidents_clean.csv', index=False)
```

Step 5: Univariate Analysis

In [2]:

```
#load the clean dataset
df_clean = pd.read_csv('US_Accidents_clean.csv')
display(df_clean)
```

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_L
0	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.0326
1	A-6	Source2	3	2016-02-08 07:44:26	2016-02-08 08:14:26	40.100590	-82.9251
2	A-7	Source2	2	2016-02-08 07:59:35	2016-02-08 08:29:35	39.758274	-84.2305
3	A-15	Source2	2	2016-02-08 08:39:43	2016-02-08 09:09:43	39.972038	-82.9135
4	A-39	Source2	2	2016-02-09 05:17:08	2016-02-09 05:47:08	39.782578	-84.1786
...	...	...	...	...	...	...	
980383	A-7777735	Source1	4	2019-08-23 13:39:48	2019-08-23 14:05:33	33.687300	-117.8901
980384	A-7777747	Source1	2	2019-08-23 16:26:06	2019-08-23 16:54:36	34.030470	-117.5981
980385	A-7777748	Source1	4	2019-08-23 16:51:29	2019-08-23 17:21:02	33.779130	-117.8879
980386	A-7777750	Source1	2	2019-08-23 17:10:58	2019-08-23 17:40:28	33.850800	-117.8436
980387	A-7777758	Source1	2	2019-08-23 19:11:30	2019-08-23 19:38:23	32.766960	-117.1480

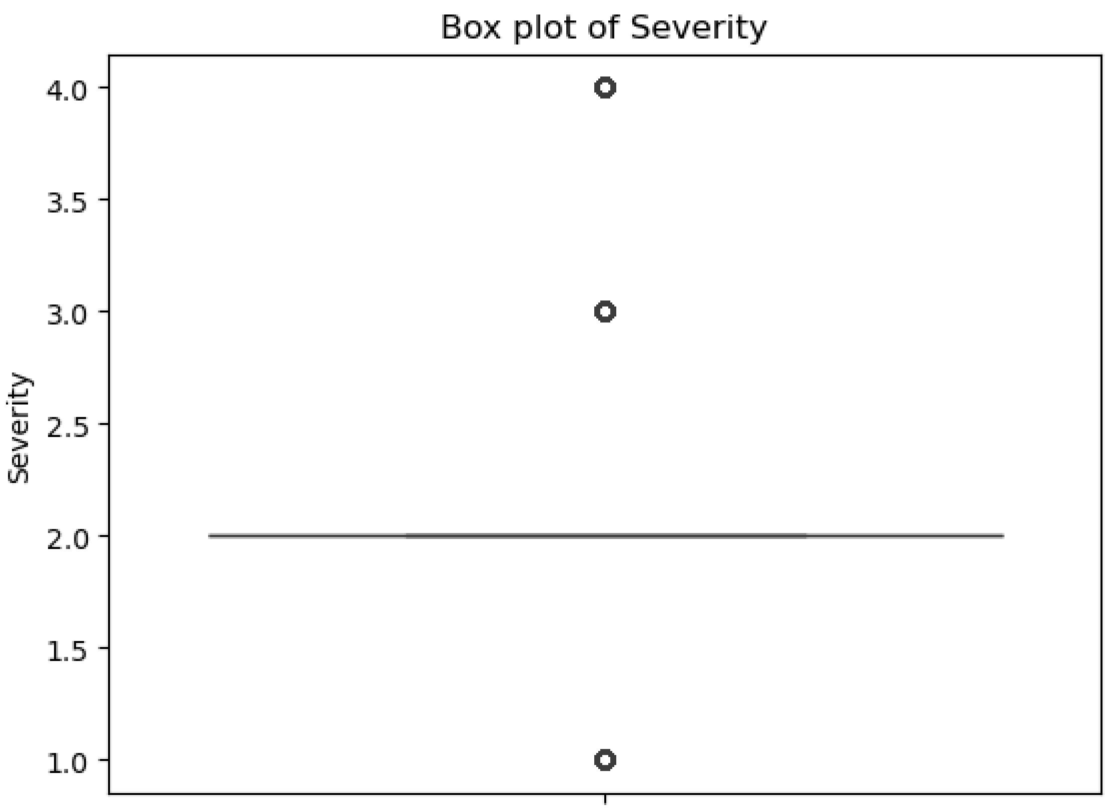
980388 rows × 44 columns

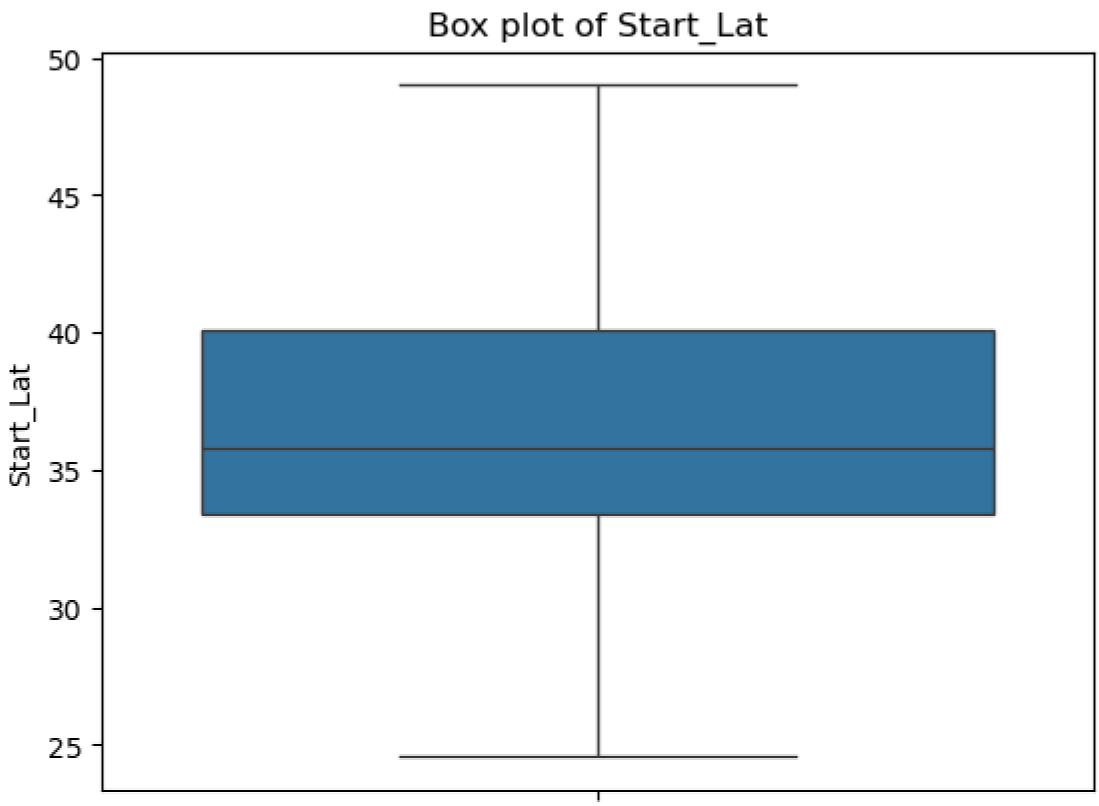
In [105…

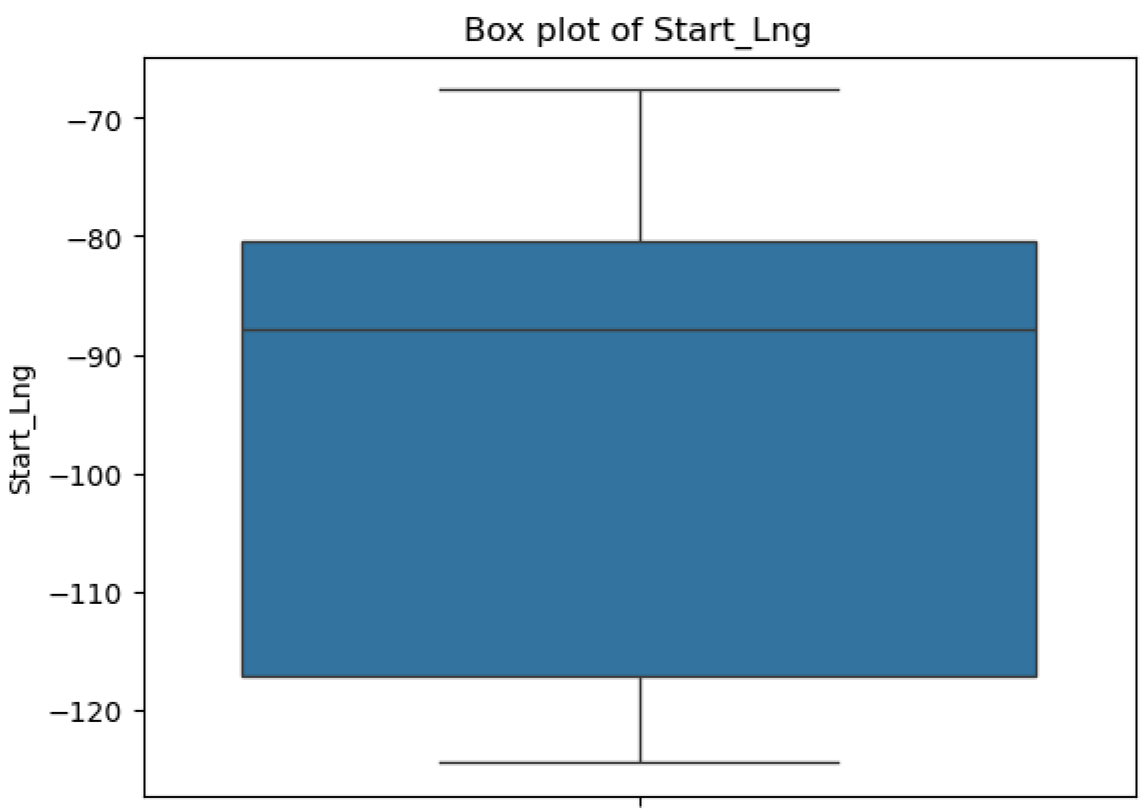
```
# For numerical variables
num_cols = df_clean.select_dtypes(include= ['int64', 'float64']).columns
for col in num_cols[:5]:
    #print(col)
    #sns.histplot(data = df_clean, x = col, kde= True)
    fig = px.histogram(data_frame=df_clean, x=col, title= f'Distribution of {c
    fig.show()
```

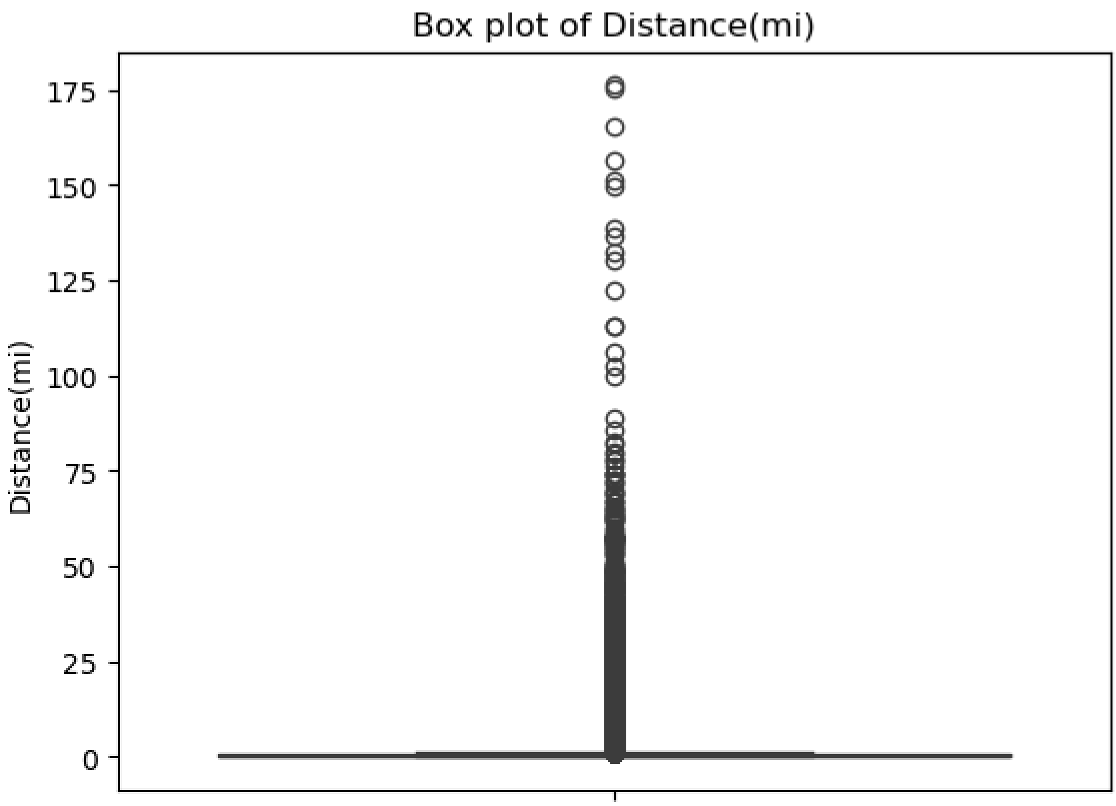
In [108…

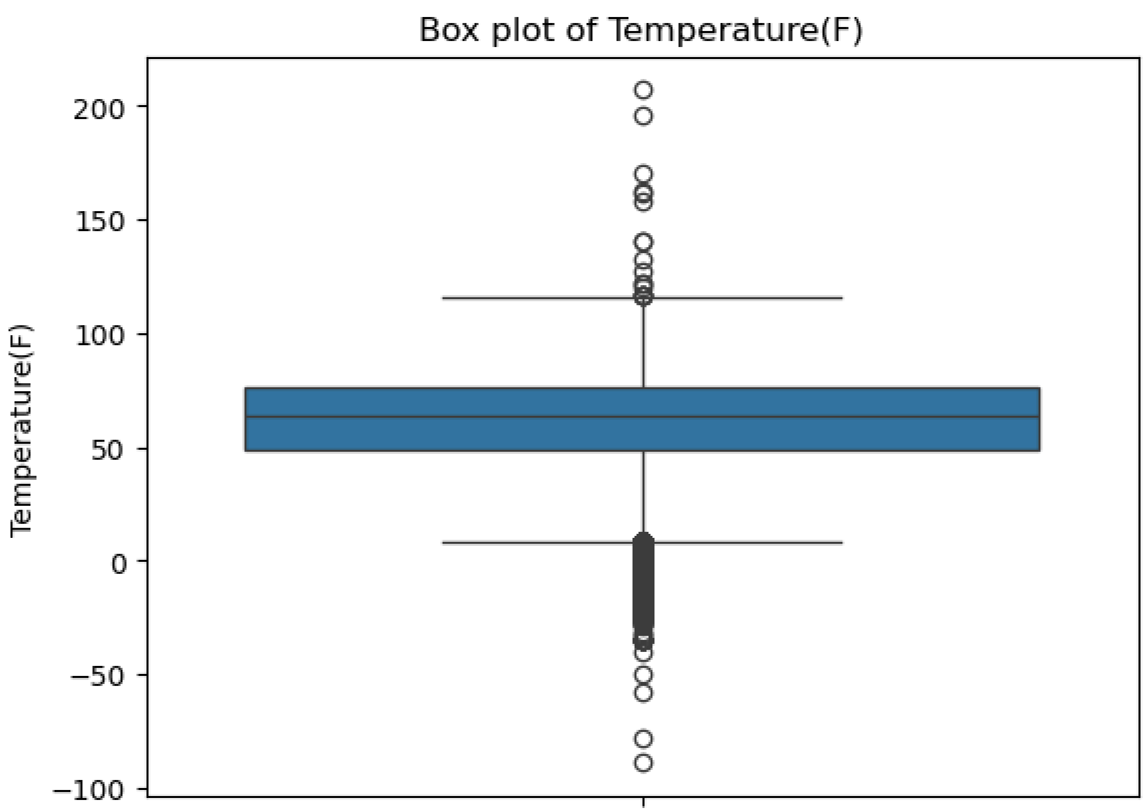
```
# Boxplot to check outliers
for col in num_cols[:5]:
    sns.boxplot(data= df_clean, y = col)
    plt.title(f"Box plot of {col}")
    plt.show()
```




Box plot of Start_Lat
Start_Lat
50
45
40
35
30
25


Box plot of Start_Lng
Start_Lng
−70
−80
−90
−100
−110
−120



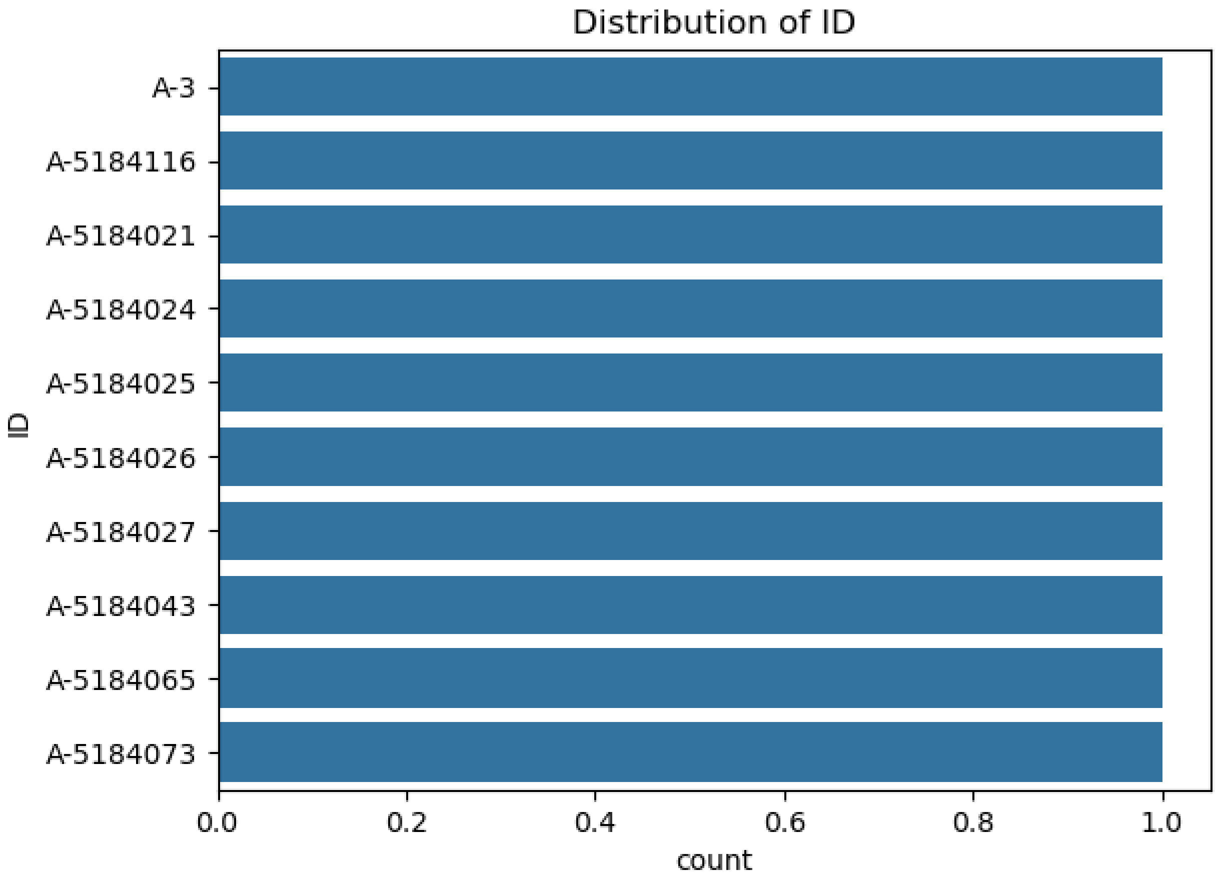


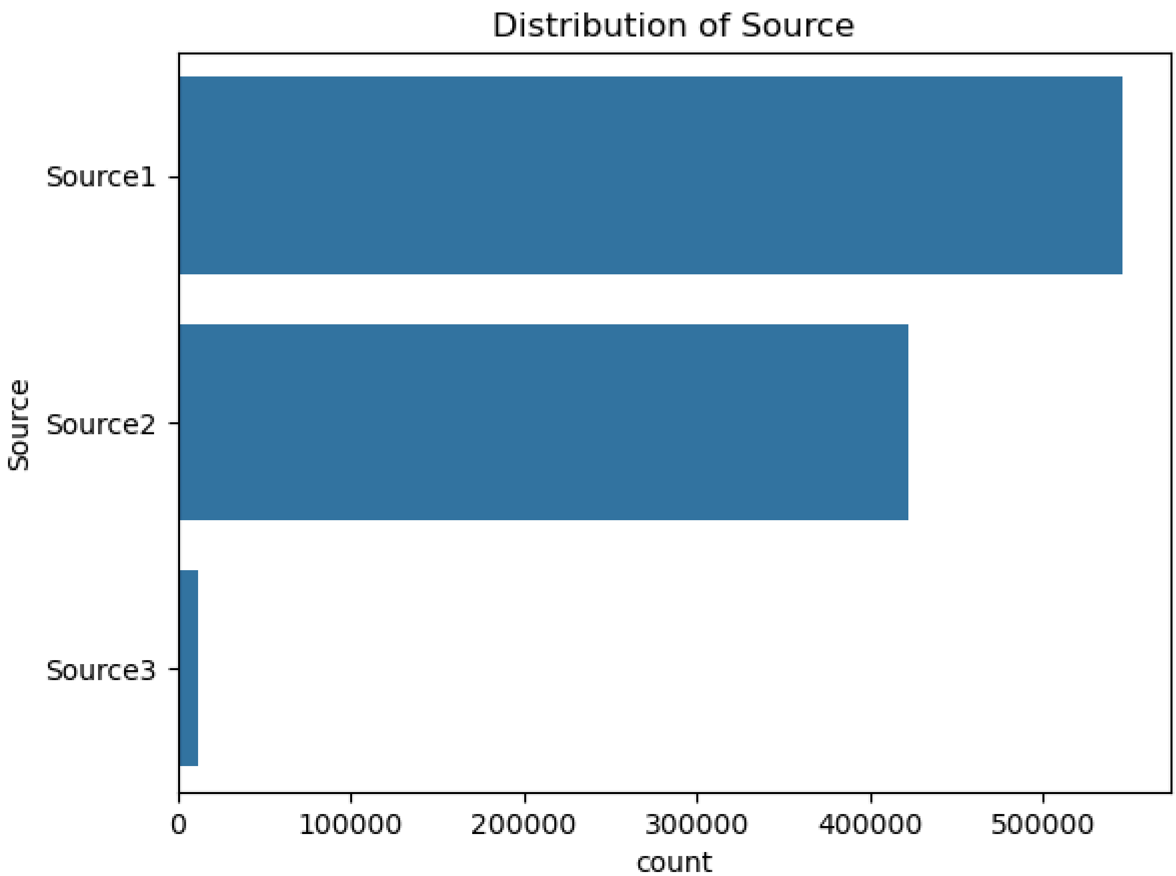
In [3]:
```
df_clean.info()
```

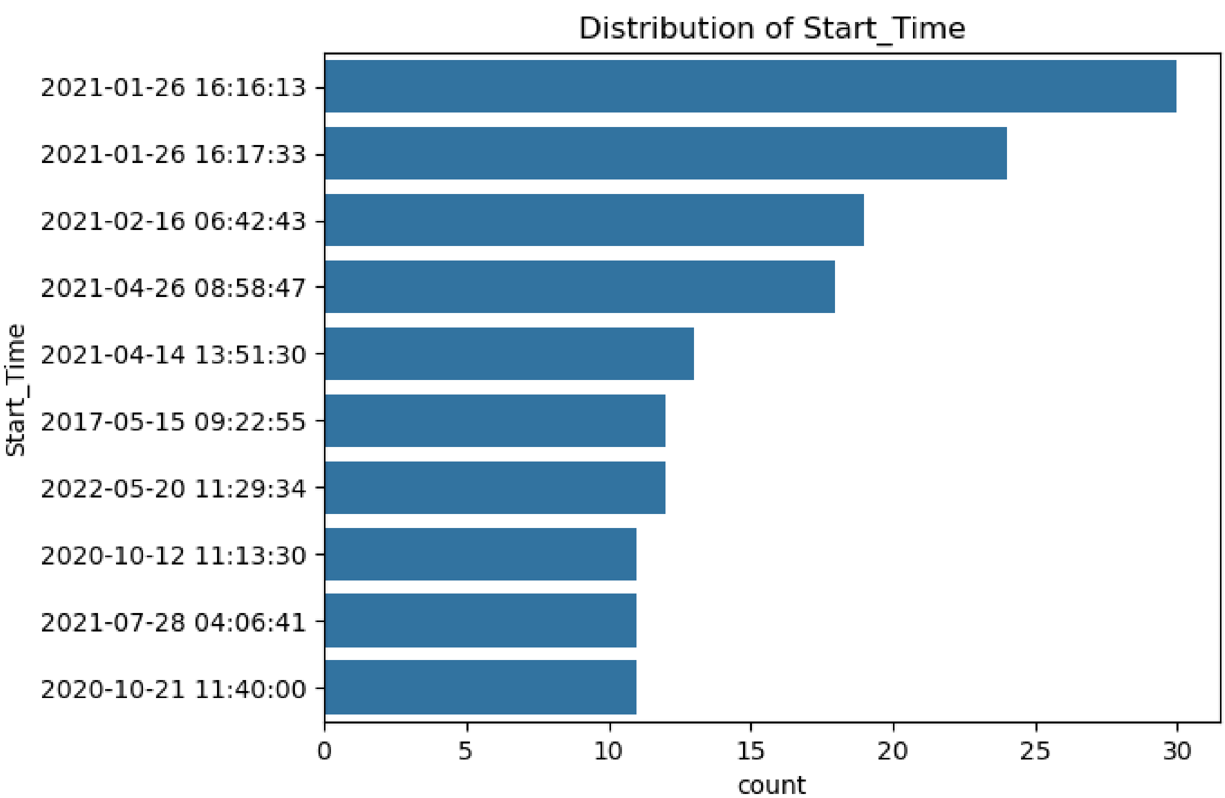
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 980388 entries, 0 to 980387
Data columns (total 44 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   ID                     980388 non-null  object 
 1   Source                 980388 non-null  object 
 2   Severity               980388 non-null  int64  
 3   Start_Time             980388 non-null  object 
 4   End_Time               980388 non-null  object 
 5   Start_Lat              980388 non-null  float64
 6   Start_Lng              980388 non-null  float64
 7   Distance(mi)           980388 non-null  float64
 8   Description            980388 non-null  object 
 9   Street                 980388 non-null  object 
 10  City                   980388 non-null  object 
 11  County                 980388 non-null  object 
 12  State                  980388 non-null  object 
 13  Zipcode                980388 non-null  object 
 14  Country                980388 non-null  object 
 15  Timezone               980388 non-null  object 
 16  Airport_Code           980388 non-null  object 
 17  Weather_Timestamp      980388 non-null  object 
 18  Temperature(F)         980388 non-null  float64
 19  Wind_Chill(F)          980388 non-null  float64
 20  Humidity(%)            980388 non-null  float64
 21  Pressure(in)           980388 non-null  float64
 22  Visibility(mi)         980388 non-null  float64
 23  Wind_Direction         980388 non-null  object 
 24  Wind_Speed(mph)        980388 non-null  float64
 25  Weather_Condition      980388 non-null  object 
 26  Amenity                980388 non-null  bool   
 27  Bump                   980388 non-null  bool   
 28  Crossing               980388 non-null  bool   
 29  Give_Way               980388 non-null  bool   
 30  Junction               980388 non-null  bool   
 31  No_Exit                980388 non-null  bool   
 32  Railway                980388 non-null  bool   
 33  Roundabout             980388 non-null  bool   
 34  Station                980388 non-null  bool   
 35  Stop                   980388 non-null  bool   
 36  Traffic_Calming        980388 non-null  bool   
 37  Traffic_Signal         980388 non-null  bool   
 38  Turning_Loop           980388 non-null  bool   
 39  Sunrise_Sunset         980388 non-null  object 
 40  Civil_Twilight         980388 non-null  object 
 41  Nautical_Twilight      980388 non-null  object 
 42  Astronomical_Twilight  980388 non-null  object 
 43  Precipitation_occured  980388 non-null  int64  
dtypes: bool(13), float64(9), int64(2), object(20)
memory usage: 244.0+ MB
```

In [4]:
```
# For categorical variables
cat_cols = df_clean.select_dtypes(include = ['object']).columns
```

```python
for col in cat_cols[:3]:
    sns.countplot(data=df_clean, y=col, order=df_clean[col].value_counts().inc
    plt.title(f"Distribution of {col}")
    plt.show()
```




Distribution of Source
Source1
Source2
Source3
Source
0
100000
200000
300000
400000
500000
count


Distribution of Start_Time
2021-01-26 16:16:13
2021-01-26 16:17:33
2021-02-16 06:42:43
2021-04-26 08:58:47
2021-04-14 13:51:30
2017-05-15 09:22:55
2022-05-20 11:29:34
2020-10-12 11:13:30
2021-07-28 04:06:41
2020-10-21 11:40:00
Start_Time
0
5
10
15
20
25
30
count

Step 6: Time Series Analysis

In [3]:
```
print(df_clean['Start_Time'].dtype)
```

```
object
```

In [3]:
```
# Extract time components
df_clean['Start_Time'] = pd.to_datetime(df_clean['Start_Time'])

df_clean['year'] = df_clean['Start_Time'].dt.year
df_clean['Month'] = df_clean['Start_Time'].dt.month
df_clean['Day'] = df_clean['Start_Time'].dt.day
df_clean['Hour'] = df_clean['Start_Time'].dt.hour
df_clean['DayOfWeek'] = df_clean['Start_Time'].dt.dayofweek  # Monday=0, Sunda
```

In [4]:
```
print(df_clean['Start_Time'].dtype)
```

```
datetime64[ns]
```

In [5]:
```
#Save the cleaned file
df_clean.to_csv('US_Accidents_clean.csv', index=False)
```

In [ ]:
```
for i in ['year', 'Month', 'Day', 'Hour', 'DayOfWeek']:
    fig = px.histogram(
        df_clean,
        x=i,
        title=f'Accidents by {i}',
        nbins=len(df_clean[i].unique()),
        text_auto = True,
        color= i,
        color_discrete_sequence=px.colors.qualitative.Pastel
    ).update_layout(bargap = 0.1)  # One bin per year
    fig.show()
```

In [16]:
```
# plot by state
fig = px.bar(df_clean['State'].value_counts()[:10], title= 'Accidents by state
fig.show()
```

Questions from insights

1. How do accidents vary by time of day?

In [12]:
```
fig = px.histogram(data_frame= df_clean,
                   x = 'Hour',
                   nbins= len(df_clean['Hour'].unique()),
                   title= "Accidents vary by time of the day",
                   text_auto= True).update_layout(bargap = 0.1)
fig.show()
```

2. Which days of the week have most accidents?

In [14]:
```
fig = px.histogram(data_frame= df_clean,
                   x = 'DayOfWeek',
                   nbins= len(df_clean['DayOfWeek'].unique()),
                   title= "Accidents by Day of Week",
                   text_auto= True
                   ).update_layout(bargap = 0.1)
fig.show()
```

3. How does accident severity distribute?

In [15]:
```
df_clean['Severity'].unique()
```

Out[15]:
```
array([2, 3, 1, 4])
```

In [22]:
```
fig = px.histogram(data_frame= df_clean,
                   x = 'Severity',
                   #nbins= len(df_clean['Severity'].unique()),
                   title= "Distribution of Accident Severity Levels",
                   text_auto= True,
                   color_discrete_sequence=['Red']
                   ).update_layout(bargap = 0.1)
fig.show()
```

4. Which states have the highest accident frequency?

In [23]:
```
df_clean['State'].unique()[:10]
```

Out[23]:
```
array(['OH', 'CA', 'FL', 'GA', 'SC', 'NE', 'IA', 'MO', 'IL', 'WI'],
      dtype=object)
```

In [25]:
```
fig = px.bar(df_clean['State'].value_counts()[:10],
                   title= "Accidents by state",
                   text_auto= True
                   ).update_layout(bargap = 0.1)
fig.show()
```

5. What's the relationship between weather conditions and severity?

In [31]:
```
top_weather_df = df_clean['Weather_Condition'].value_counts().head(10).index
weather_df = df_clean[df_clean['Weather_Condition'].isin(top_weather_df)]

display(top_weather_df)
display(weather_df.head())
```

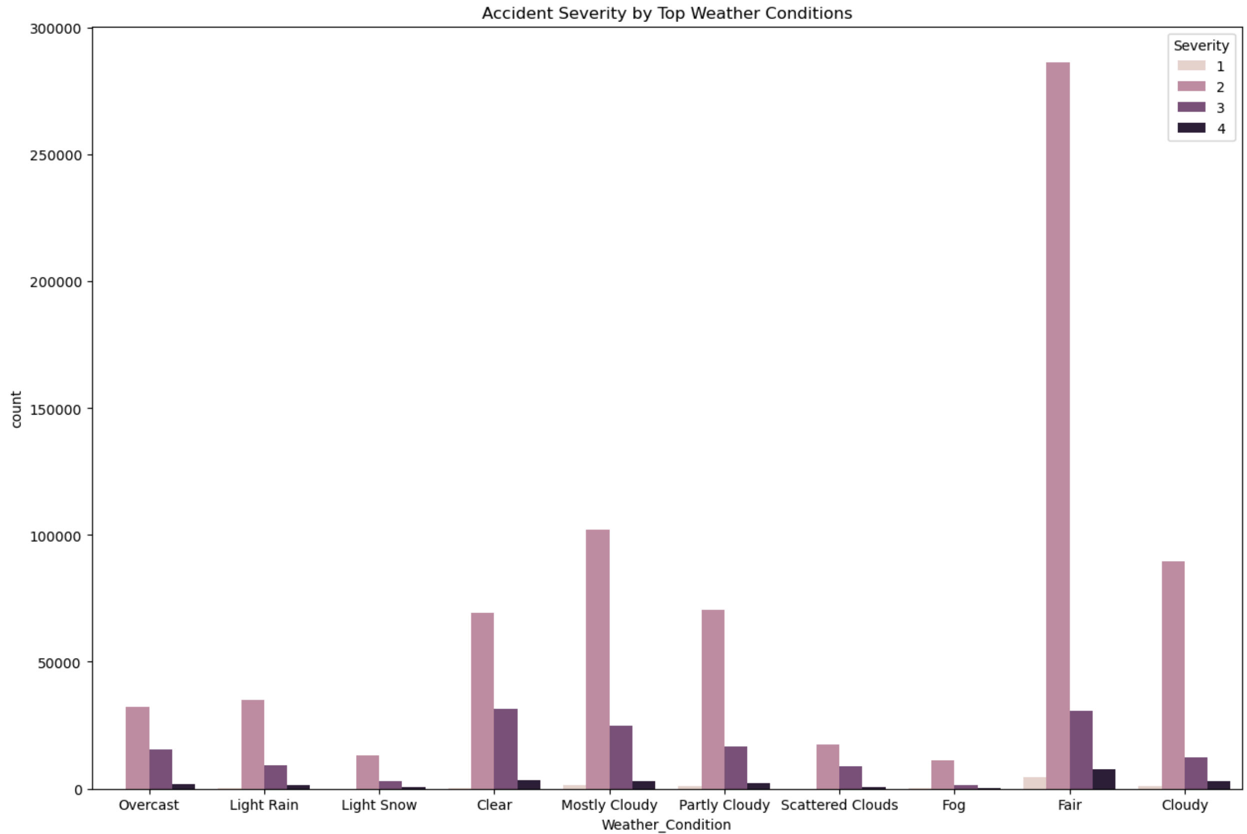
```
Index(['Fair', 'Mostly Cloudy', 'Cloudy', 'Clear', 'Partly Cloudy', 'Overcast',
       'Light Rain', 'Scattered Clouds', 'Light Snow', 'Fog'],
      dtype='object', name='Weather_Condition')
```

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	Distance
0	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	
1	A-6	Source2	3	2016-02-08 07:44:26	2016-02-08 08:14:26	40.100590	-82.925194	
2	A-7	Source2	2	2016-02-08 07:59:35	2016-02-08 08:29:35	39.758274	-84.230507	
3	A-15	Source2	2	2016-02-08 08:39:43	2016-02-08 09:09:43	39.972038	-82.913521	
4	A-39	Source2	2	2016-02-09 05:17:08	2016-02-09 05:47:08	39.782578	-84.178688	

5 rows × 49 columns

In [38]:

```
plt.figure(figsize=(15, 10))
sns.countplot(data = weather_df, x = weather_df['Weather_Condition'], hue = we
plt.title('Accident Severity by Top Weather Conditions')
plt.show()
```



6. What's the temperature distribution during accidents?

In [39]:
```
df_clean['Temperature(F)'].unique()[0:10]
```

Out[39]:
```
array([36. , 37.9, 34. , 37.4, 22.8, 25. , 26.1, 24.8, 27. , 15.1])
```

In [46]:
```
fig = px.histogram(data_frame= df_clean,
                   x = df_clean['Temperature(F)'],
                   nbins = 50,
                   title= "Temperature Distribution During Accidents",
                   text_auto= True
                   ).update_layout(bargap = 0.1)
fig.show()
```

7. How do accidents vary by light conditions?

In [50]:
```
df_clean['Sunrise_Sunset'].unique()
```

Out[50]:
```
array(['Night', 'Day'], dtype=object)
```

In [59]:
```
fig = px.histogram(data_frame= df_clean,
                   x = df_clean['Severity'],
                   color= df_clean['Sunrise_Sunset'],
                   barmode= 'group',
                   title= "Accidents by Daylight vs Night",
```

```
                    text_auto= True
                    )
fig.show()
```

Conclusion

-- Most accidents happen during the day, fewer accidents happen late at night, maybe because fewer cars are on the road.

-- Accidents mostly occur when temperatures are between 50°F and 80°F – normal driving weather.

-- California, Florida, and Texas have the highest number of accidents – probably because they have more people and traffic.

-- Accidents are more common on weekdays, especially on Fridays. Fewer happen on weekends.

-- Accidents peak during morning (7-8 AM) and evening (4–6 PM) rush hours, when people go to and return from work.

In [ ]:

```
```