The rendered version is at `/pdf_notebooks/02-US_data_analysis.pdf`

In [1]:
```
import pandas as pd
import numpy as np
import re
import math
```

In [2]:
```
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from plotly.subplots import make_subplots
```

In this notebook, we analyse the US Traffic Accident dataset to derive insights and select features for predictive models.

1. Load Data

In [3]:
```
%%time
data = pd.read_csv("data/US_Accidents_March23_Clean.csv")
data.shape
```

```
CPU times: total: 22.2 s
Wall time: 25.1 s
```

Out[3]:
```
(6818003, 40)
```

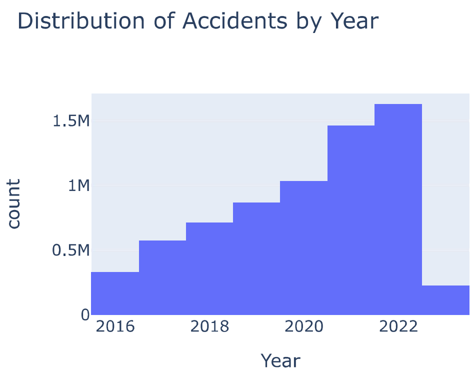
The dataset we have contains close to 7 millions rows and has a size of 1.5Go (or more). In this analysis and beyond, we are focusing solely on the years 2021 and 2022 to ensure that our insights and predictive models are based on the most recent and relevant data available.

- **Recent and Relevant Data:** The years 2021 and 2022 would be more relevant to leverage the most current insights into what influences impacting accident severity.
- **Higher Data Volume:** Despite comprising a smaller portion of the dataset (43%), data from the years 2021 and 2022 offer a substantial amount of recorded accidents, ensuring robust analysis and modeling.
- **Accuracy in Predictions:** By analyzing recent years, we aim to produce predictive models that accurately reflect present-day accident trends and conditions, enhancing the reliability of our forecasts.
- **Resource Optimization:** Prioritizing these years optimizes our resources (less data to process) by concentrating efforts on data that is more likely to yield actionable insights.

In [4]:
```
print(f'Portion of data for 2021 to 2022: {100*data[(data["Year"] == 2021) | (data["Year"] == 2022)].shape[0] / data.shape[0]:.2f}%')
print(f'Portion of data for oher years: {100*data[(data["Year"] != 2021) & (data["Year"] != 2022)].shape[0] / data.shape[0]:.2f}%')
```

```
Portion of data for 2021 to 2022: 45.21%
Portion of data for oher years: 54.79%
```

In [5]:
```
fig = px.histogram(data, x='Year', title='Distribution of Accidents by Year')
fig.update_layout(width=450, height=350)
fig.show(config={'staticPlot': True})
```



In [6]:
```
data = data[(data["Year"] == 2021) | (data["Year"] == 2022)].copy()
data.shape
```

Out[6]:
```
(3082687, 40)
```

The dataset is also heavily imbalanced. Accidents of `Severity` 2 make up over 80% of all data. We could downsample this class to have closer to the other ones. This would allow the analysis to be more effectibe.

In [7]:
```
downsampled_count = int(data["Severity"].value_counts().sort_values(ascending=False).iloc[1] * 3.0)
downsampled_count
```

Out[7]:
```
637410
```

In [8]:

```
df_majority_downsampled = data[data["Severity"] == 2].sample(n=downsampled_count, random_state=42)
df_rest = data[data["Severity"] != 2]
df_balanced = pd.concat([df_majority_downsampled, df_rest])
```

In [9]:

```
df_balanced.shape
```

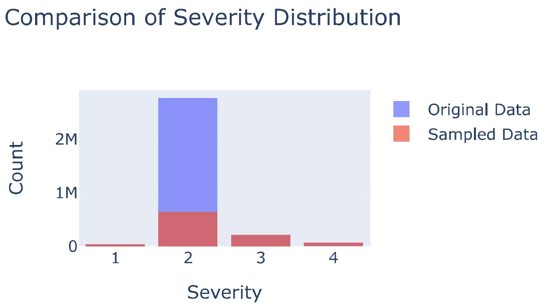
Out[9]:

```
(949682, 40)
```

We are down to about 940000 rows which is more manageable

In [10]:

```
fig = go.Figure()
fig.add_trace(go.Histogram(x=data['Severity'], opacity=0.7, name='Original Data'))
fig.add_trace(go.Histogram(x=df_balanced['Severity'], opacity=0.7, name='Sampled Data'))
fig.update_layout(
    height=300, width=450,
    title='Comparison of Severity Distribution',
    xaxis_title='Severity', yaxis_title='Count',
    barmode='overlay', bargap=0.1, bargroupgap=0.1,
    xaxis=dict(tickmode='linear',  tick0=min(data['Severity']), dtick=1)
)
# Show the plot
fig.show(config={'staticPlot': True})
```



In [11]:

```
df = df_balanced.copy()
```

In [12]:

```
del(data)
del(df_balanced)
```

We fix the Datetime datatypes

In [13]:

```
df["Start_Time"] = pd.to_datetime(df["Start_Time"])
df["End_Time"] = pd.to_datetime(df["End_Time"])
df["Date"] = pd.to_datetime(df["Date"])
```

Descriptive Analysis

In [14]:

```
numerical_vars = df.select_dtypes(include=['number']).columns.tolist()
boolean_vars = df.select_dtypes(include=['bool']).columns.tolist()
categorical_vars = df.select_dtypes(include=['object','category']).columns.tolist()
datetime_vars = df.select_dtypes(include=['datetime']).columns.tolist()
```

1. Univariate Analysis

Numerical and ordinal categorical variables

- **Hourly Patterns:** Peak accident times are during the late afternoon (16:00 - 17:00), likely due to the evening rush hour. The early morning hours (2:00 - 5:00) have the fewest accidents.
- **Daily Patterns:** Accidents are evenly spread across the days of the month, with minor fluctuations. This indicates no specific days are particularly prone to accidents.
- **Weekly Patterns:** Weekdays see a higher number of accidents compared to weekends. Fridays have the highest number of accidents, possibly due to end-of-week fatigue and increased travel. Sundays have the fewest, suggesting reduced traffic.
- **Monthly Patterns:** December has the highest number of accidents, possibly due to winter weather and holiday travel. October has the lowest, which might be attributed to milder weather.
- **Weather Conditions:** Most accidents occur under clear and cloudy conditions, with fewer accidents in severe weather conditions like snowstorms and thunderstorms. The mean temperature during accidents is 63°F, indicating accidents occur across a wide range of temperatures. The average visibility is 9 miles, and wind speeds are generally low (mean of 7.38 mph). However, there are extreme values, indicating occasional severe conditions.
- **Distance and Duration:** The median accident duration is approximately 78 minutes, with a wide range of durations indicating variability in accident severity and response times. The average distance affected by an accident is relatively short (0.73 miles), with most area affected being at or near the accident location.

- **Traffic Features:** Traffic signals, crossings, and junctions are common at accident sites. Notably, a significant portion of accidents occur at night (30.20%) and on highways (32.63%), suggesting these conditions require special attention for safety improvements.
- **State-Level Insights:** California and Florida have the highest number of accidents, reflecting their large populations and extensive road networks. States like Wyoming and Vermont have significantly fewer accidents, likely due to smaller populations and less traffic.

In [15]:

```
len(numerical_vars)
```

Out[15]:

```
16
```

In [21]:

```
df[numerical_vars].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 949682 entries, 1032100 to 6114234
Data columns (total 16 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Severity           949682 non-null  int64  
 1   Start_Lat          949682 non-null  float64
 2   Start_Lng          949682 non-null  float64
 3   Distance(mi)       949682 non-null  float64
 4   Temperature(F)     949682 non-null  float64
 5   Humidity(%)        949682 non-null  float64
 6   Pressure(in)       949682 non-null  float64
 7   Visibility(mi)     949682 non-null  float64
 8   Wind_Speed(mph)    949682 non-null  float64
 9   Precipitation(in)  949682 non-null  float64
 10  Duration(min)      949682 non-null  float64
 11  Hour               949682 non-null  int64  
 12  Day                949682 non-null  int64  
 13  Day_of_Week        949682 non-null  int64  
 14  Month              949682 non-null  int64  
 15  Year               949682 non-null  int64  
dtypes: float64(10), int64(6)
memory usage: 123.2 MB
```

In [16]:

```
df["Severity"].value_counts()
```

Out[16]:

```
2    637410
3    212470
4     63948
1     35854
Name: Severity, dtype: int64
```

In [17]:

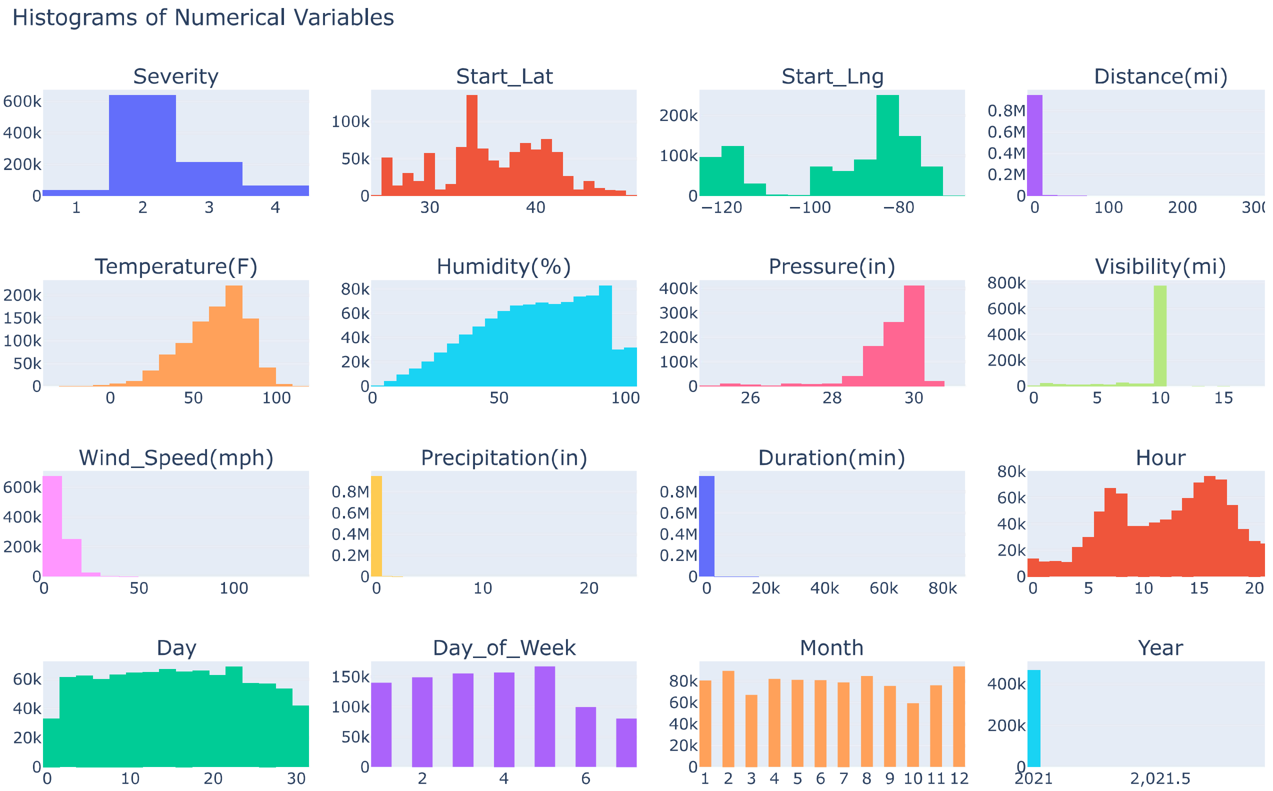
```
df[numerical_vars].describe()
```

Out[17]:

	Severity	Start_Lat	Start_Lng	Distance(mi)	Temperature(F)	Humidity(%)	Pressure(in)	Visibility(mi)	Wind_Speed(mph)	Precipitation(in)
count	949682.000000	949682.000000	949682.000000	949682.000000	949682.000000	949682.000000	949682.000000	949682.000000	949682.000000	949682.000000
mean	2.320646	36.026573	-92.618611	0.731070	63.089140	64.277052	29.451392	9.113224	7.375663	0.005913
std	0.654227	5.148736	16.859640	2.029884	19.140466	22.585382	0.812348	2.297177	5.383343	0.053254
min	1.000000	24.571531	-124.539056	0.000000	-38.000000	1.000000	25.000000	0.000000	0.000000	0.000000
25%	2.000000	32.972778	-111.892227	0.000000	51.000000	48.000000	29.210000	10.000000	3.000000	0.000000
50%	2.000000	35.825008	-85.285594	0.119000	66.000000	66.000000	29.680000	10.000000	7.000000	0.000000
75%	3.000000	40.103453	-80.169367	0.704000	77.000000	83.000000	29.950000	10.000000	10.000000	0.000000
max	4.000000	49.000504	-67.709053	336.570007	117.000000	100.000000	30.760000	20.000000	132.000000	23.970000

In [18]:

```
fig = make_subplots(rows=4, cols=4, subplot_titles=numerical_vars)
for i, col in enumerate(numerical_vars):
    row = i // 4 + 1
    col_pos = i % 4 + 1
    fig.add_trace(
        go.Histogram(x=df[col], nbinsx=25 if col != 'Severity' else 4, showlegend=False),
        row=row, col=col_pos
    )

# Update layout
fig.update_layout(height=700, width=1120, title_text="Histograms of Numerical Variables")
fig.update_xaxes(tickvals=[1, 2, 3, 4], row=1, col=1)
fig.update_xaxes(tickvals=list(range(1, 13)), row=4, col=3)

# Show plot
fig.show(config={'staticPlot': True})
```



Boolean variables

In [19]:

```
len(boolean_vars)
```

Out[19]:

```
15
```

In [20]:

```
df[boolean_vars].info()
```

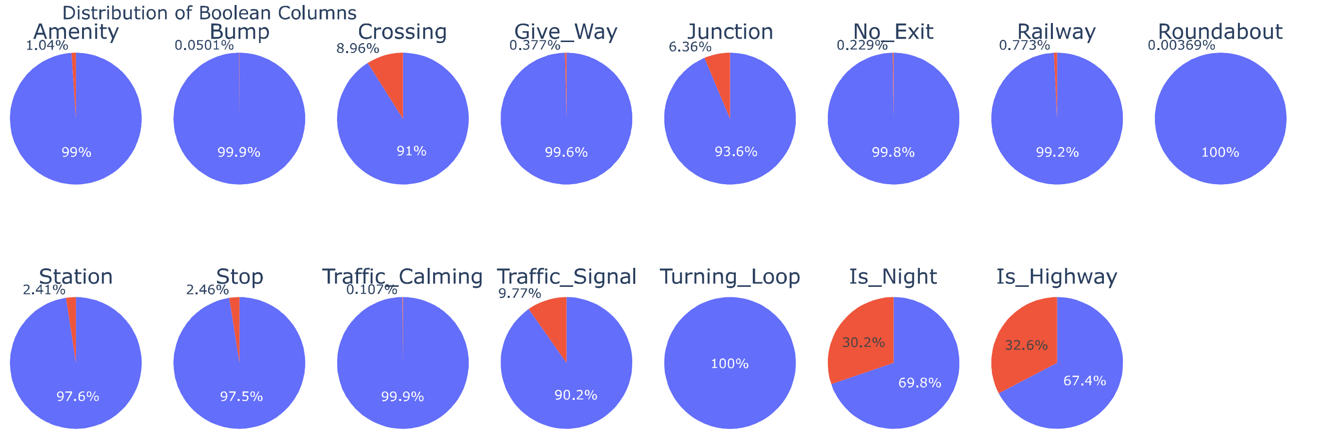
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 949682 entries, 1032100 to 6114234
Data columns (total 15 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   Amenity          949682 non-null  bool
 1   Bump             949682 non-null  bool
 2   Crossing         949682 non-null  bool
 3   Give_Way         949682 non-null  bool
 4   Junction         949682 non-null  bool
 5   No_Exit          949682 non-null  bool
 6   Railway          949682 non-null  bool
 7   Roundabout       949682 non-null  bool
 8   Station          949682 non-null  bool
 9   Stop             949682 non-null  bool
 10  Traffic_Calming  949682 non-null  bool
 11  Traffic_Signal   949682 non-null  bool
 12  Turning_Loop     949682 non-null  bool
 13  Is_Night         949682 non-null  bool
 14  Is_Highway       949682 non-null  bool
dtypes: bool(15)
memory usage: 20.8 MB
```

In [22]:

```
true_counts = df[boolean_vars].sum()
```

In [23]:

```
num_plots = len(boolean_vars)
num_cols = 8
num_rows = math.ceil(num_plots / num_cols)

# Create Plotly figure with subplots
fig = make_subplots(rows=num_rows, cols=num_cols, subplot_titles=boolean_vars,
                    specs=[[{'type':'pie'}]*num_cols]*num_rows)
# Populate subplots with pie charts
for i, column in enumerate(boolean_vars):
    row = i // num_cols + 1  # Plotly subplots start from row 1
    col = i % num_cols + 1   # Plotly subplots start from col 1
    counts = df[column].value_counts()
    fig.add_trace(
        go.Pie(labels=counts.index, values=counts, textinfo='percent', sort=False),
        row=row, col=col
    )
    fig.update_layout(
        title=f"{column}",
        font=dict(size=10),
        margin=dict(l=10, r=10, t=40, b=10),  # Adjust margins for better layout
        showlegend=False
    )
# Update layout and show figure
fig.update_layout(
    title='Distribution of Boolean Columns',
    height=350, width=1000,
    template='plotly_white',
)

fig.show(config={'staticPlot': True})
```



Categorical variables

Some variables may not be very useful as they are so we would use their transformed version or new variables extracted from them.

In [24]:

```
categorical_vars.remove("Street")
categorical_vars.remove("Weather_Condition")
```

In [25]:

```
len(categorical_vars)
```

Out[25]:

```
4
```

In [26]:

```
df[categorical_vars].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 949682 entries, 1032100 to 6114234
Data columns (total 4 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   City              949682 non-null  object
 1   County            949682 non-null  object
 2   State             949682 non-null  object
 3   Weather_Category  949682 non-null  object
dtypes: object(4)
memory usage: 36.2+ MB
```

In [27]:

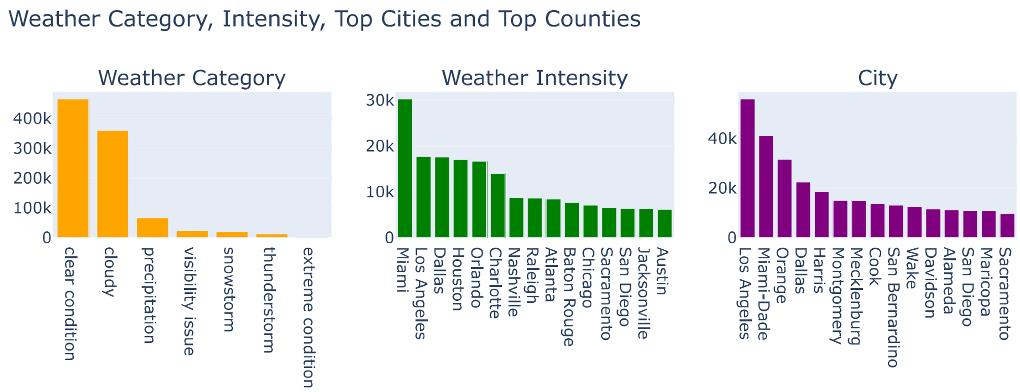
```
print("Unique values counts")
for col in categorical_vars:
    print(f"{col}: {df[col].unique().shape[0]}")
```

```
Unique values counts
City: 9325
County: 1548
State: 49
Weather_Category: 7
```

In [28]:

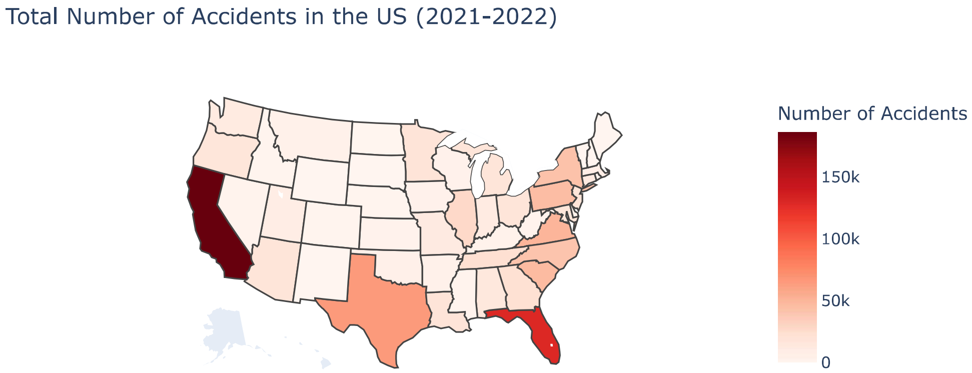
```
# Create a subplot for Weather Category, Weather Intensity, and City
fig = make_subplots(rows=1, cols=3, subplot_titles=['Weather Category', 'Weather Intensity', 'City', 'County'])

# Plot Weather Category (bar plot)
weather_cat_counts = df['Weather_Category'].value_counts()
fig.add_trace(go.Bar(x=weather_cat_counts.index, y=weather_cat_counts.values, marker_color='orange'), row=1, col=1)

# Plot City (bar plot)
city_counts = df['City'].value_counts().nlargest(15)
fig.add_trace(go.Bar(x=city_counts.index, y=city_counts.values, marker_color='green'), row=1, col=2)

# Plot County (bar plot)
county_counts = df['County'].value_counts().nlargest(15)
```

```
fig.add_trace(go.Bar(x=county_counts.index, y=county_counts.values, marker_color='purple'), row=1, col=3)

fig.update_layout(title='Weather Category, Intensity, Top Cities and Top Counties',
                  height=340, width=900, showlegend=False)

fig.show(config={'staticPlot': True})
```



In [30]:

```
# Plot State (choropleth map)
state_counts = df['State'].value_counts().reset_index()
state_counts.columns = ['State', 'Counts']

# Create choropleth map for State
fig = go.Figure(data=go.Choropleth(
    locations=state_counts['State'],
    z=state_counts['Counts'],
    locationmode='USA-states',
    colorscale='Reds',
    colorbar_title='Number of Accidents'
))

fig.update_layout(
    title_text='Total Number of Accidents in the US (2021-2022)',
    geo=dict(scope='usa', projection_type='albers usa'),
    height=400, width=800, showlegend=True, barmode='group',
)

# Show the plot
fig.show(config={'staticPlot': True})
```



In [ ]: