US Accidents (2016 - 2023): A Countrywide Traffic Accident Dataset

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. You can download data here [Dataset (https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents)](https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents)

#	Attribute	Description	Nullable
1	ID	This is a unique identifier of the accident record.	No
2	Severity	Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).	No
3	Start_Time	Shows start time of the accident in local time zone.	No
4	End_Time	Shows end time of the accident in local time zone. End time here refers to when the impact of accident on traffic flow was dismissed.	No
5	Start_Lat	Shows latitude in GPS coordinate of the start point.	No
6	Start_Lng	Shows longitude in GPS coordinate of the start point.	No
7	End_Lat	Shows latitude in GPS coordinate of the end point.	Yes
8	End_Lng	Shows longitude in GPS coordinate of the end point.	Yes
9	Distance(mi)	The length of the road extent affected by the accident.	No
10	Description	Shows natural language description of the accident.	No
11	Number	Shows the street number in address field.	Yes
12	Street	Shows the street name in address field.	Yes
13	Side	Shows the relative side of the street (Right/Left) in address field.	Yes
14	City	Shows the city in address field.	Yes
15	County	Shows the county in address field.	Yes
16	State	Shows the state in address field.	Yes
17	Zipcode	Shows the zipcode in address field.	Yes
18	Country	Shows the country in address field.	Yes
19	Timezone	Shows timezone based on the location of the accident (eastern, central, etc.).	Yes
20	Airport_Code	Denotes an airport-based weather station which is the closest one to location of the accident.	Yes
21	Weather_Timestamp	Shows the time-stamp of weather observation record (in local time).	Yes
22	Temperature(F)	Shows the temperature (in Fahrenheit).	Yes
23	Wind_Chill(F)	Shows the wind chill (in Fahrenheit).	Yes
24	Humidity(%)	Shows the humidity (in percentage).	Yes
25	Pressure(in)	Shows the air pressure (in inches).	Yes
26	Visibility(mi)	Shows visibility (in miles).	Yes
27	Wind_Direction	Shows wind direction.	Yes

#	Attribute	Description	Nullable
28	Wind_Speed(mph)	Shows wind speed (in miles per hour).	Yes
29	Precipitation(in)	Shows precipitation amount in inches, if there is any.	Yes
30	Weather_Condition	Shows the weather condition (rain, snow, thunderstorm, fog, etc.)	Yes
31	Amenity	A POI (https://wiki.openstreetmap.org/wiki/Points_of_interest) annotation which indicates presence of amenity (https://wiki.openstreetmap.org/wiki/Key:amenity) in a nearby location.	No
32	Bump	A POI annotation which indicates presence of speed bump or hump in a nearby location.	No
33	Crossing	A POI annotation which indicates presence of crossing (https://wiki.openstreetmap.org/wiki/Key:crossing) in a nearby location.	No
34	Give_Way	A POI annotation which indicates presence of give_way (https://wiki.openstreetmap.org/wiki/Tag:highway%3Dgive_way) in a nearby location.	No
35	Junction	A POI annotation which indicates presence of junction (https://wiki.openstreetmap.org/wiki/Key:junction) in a nearby location.	No
36	No_Exit	A POI annotation which indicates presence of no_exit (https://wiki.openstreetmap.org/wiki/Key:noexit) in a nearby location.	No
37	Railway	A POI annotation which indicates presence of railway (https://wiki.openstreetmap.org/wiki/Key:railway) in a nearby location.	No
38	Roundabout	A POI annotation which indicates presence of roundabout (https://wiki.openstreetmap.org/wiki/Tag:junction%3Droundabout) in a nearby location.	No
39	Station	A POI annotation which indicates presence of station (https://wiki.openstreetmap.org/wiki/Key:station) in a nearby location.	No
40	Stop	A POI annotation which indicates presence of stop (https://wiki.openstreetmap.org/wiki/Key:stop) in a nearby location.	No
41	Traffic_Calming	A POI annotation which indicates presence of traffic_calming (https://wiki.openstreetmap.org/wiki/Key:traffic_calming) in a nearby location.	No
42	Traffic_Signal	A POI annotation which indicates presence of traffic_signal (https://wiki.openstreetmap.org/wiki/Tag:highway%3Dtraffic_signals) in a nearby loction.	No
43	Turning_Loop	A POI annotation which indicates presence of turning_loop (https://wiki.openstreetmap.org/wiki/Tag:highway%3Dturning_loop) in a nearby location.	No
44	Sunrise_Sunset	Shows the period of day (i.e. day or night) based on sunrise/sunset.	Yes
45	Civil_Twilight	Shows the period of day (i.e. day or night) based on civil twilight (https://en.wikipedia.org/wiki/Twilight#Civil_twilight).	Yes
46	Nautical_Twilight	Shows the period of day (i.e. day or night) based on nautical twilight (https://en.wikipedia.org/wiki/Twilight#Nautical_twilight).	Yes
47	Astronomical_Twilight	Shows the period of day (i.e. day or night) based on astronomical twilight	Yes

Entrée [1]:
```
import pandas as pd
import numpy as np
import missingno as msno
from sqlalchemy import create_engine
import pymysql
```

Entrée [2]:
```
data = pd.read_csv("Data/US_Accidents_March23.csv")
print(data.shape)
data.columns
```

```
(7728394, 46)
```

Out[2]:
```
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')
```

Entrée [3]:
```
pd.set_option('display.max_columns', None)
data.head()
```

Out[3]:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)
0	A-1	Source2	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	NaN	NaN	0.01
1	A-2	Source2	2	2016-02-08 06:07:59	2016-02-08 06:37:59	39.928059	-82.831184	NaN	NaN	0.01
2	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	NaN	NaN	0.01
3	A-4	Source2	3	2016-02-08 07:23:34	2016-02-08 07:53:34	39.747753	-84.205582	NaN	NaN	0.01
4	A-5	Source2	2	2016-02-08 07:39:07	2016-02-08 08:09:07	39.627781	-84.188354	NaN	NaN	0.01

1. Data Preparation

Delete unused columns

Entrée [4]:
```
data.drop(columns=['Country','Timezone','Source', 'ID',
                   'Description','Street','County','Zipcode','Airport_Code',
                   'End_Lat', 'End_Lng', 'Astronomical_Twilight', 'Civil_Twilight',
                   'Nautical_Twilight','Weather_Timestamp'],
          inplace=True)
data.columns
```

Out[4]:
```
Index(['Severity', 'Start_Time', 'End_Time', 'Start_Lat', 'Start_Lng',
       'Distance(mi)', 'City', 'State', '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'],
      dtype='object')
```

Data description

Entrée [5]:
```
data.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	
Severity	7728394.0	2.212384	0.487531	1.000000	2.000000	2.000000	2.000000	
Start_Lat	7728394.0	36.201195	5.076079	24.554800	33.399631	35.823974	40.084959	4
Start_Lng	7728394.0	-94.702545	17.391756	-124.623833	-117.219396	-87.766616	-80.353676	-
Distance(mi)	7728394.0	0.561842	1.776811	0.000000	0.000000	0.030000	0.464000	4
Temperature(F)	7564541.0	61.663286	19.013653	-89.000000	49.000000	64.000000	76.000000	2
Wind_Chill(F)	5729375.0	58.251048	22.389832	-89.000000	43.000000	62.000000	75.000000	2
Humidity(%)	7554250.0	64.831041	22.820968	1.000000	48.000000	67.000000	84.000000	1
Pressure(in)	7587715.0	29.538986	1.006190	0.000000	29.370000	29.860000	30.030000	
Visibility(mi)	7551296.0	9.090376	2.688316	0.000000	10.000000	10.000000	10.000000	1
Wind_Speed(mph)	7157161.0	7.685490	5.424983	0.000000	4.600000	7.000000	10.400000	10
Precipitation(in)	5524808.0	0.008407	0.110225	0.000000	0.000000	0.000000	0.000000	

Convert Objects dates to Timestamp

Entrée [6]:
```
# this function convert date columns from Objects to timestamp
def convert_date_columns(columns):
    for column in columns:
        data[column] = pd.to_datetime(data[column])
    return data
convert_date_columns(['Start_Time','End_Time'])
data[['Start_Time','End_Time']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7728394 entries, 0 to 7728393
Data columns (total 2 columns):
 #   Column      Dtype         
---  ------      -----         
 0   Start_Time  datetime64[ns]
 1   End_Time    datetime64[ns]
dtypes: datetime64[ns](2)
memory usage: 117.9 MB
```

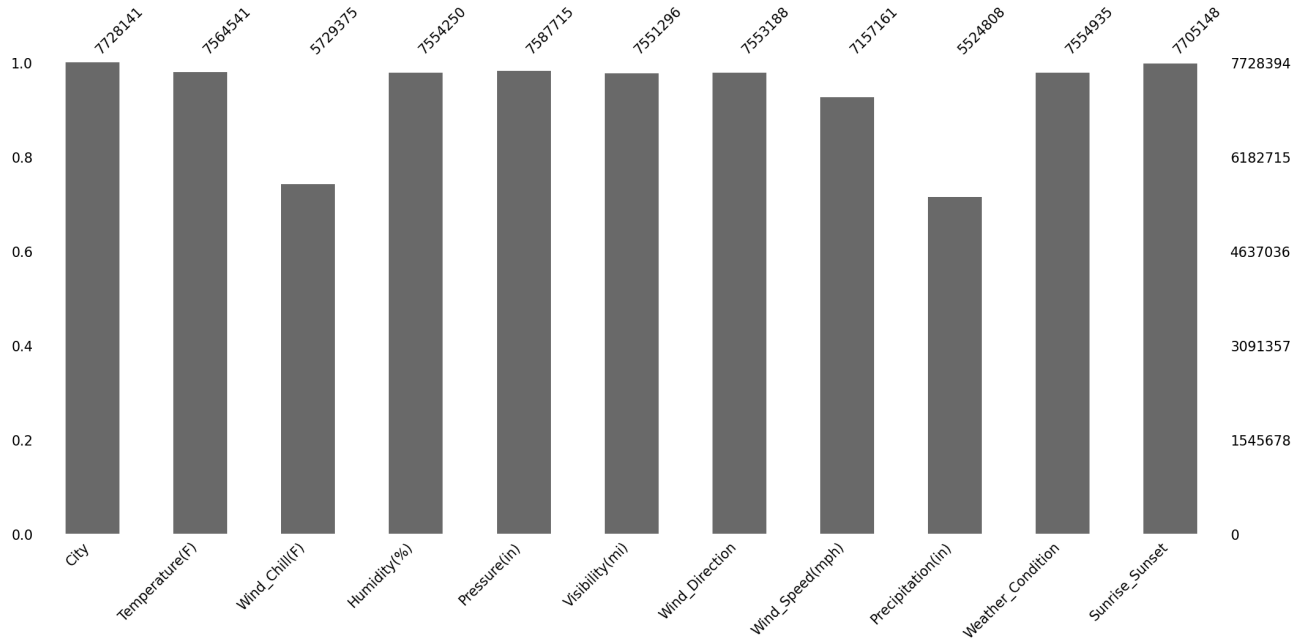
Dealing with missing data

Entrée [7]:

```
from pprint import pprint
pprint('*'*68)
pprint('# of Rows: {0[0]}    ***    # of Columns : {0[1]}'.format(data.shape))
pprint('*'*68)
data_info = pd.DataFrame(data.dtypes.reset_index()).rename(columns = {0 : 'Type'}).se

data_info = pd.concat([data_info,data.isnull().sum()], axis=1).rename(columns = {0 :
data_info = pd.concat([data_info,(data.isnull().mean()*100).round(2)], axis=1).rename

pprint(data_info)
pprint('*'*68)
```

```
'********************************************************************'
'# of Rows: 7728394    ***    # of Columns : 31'
'********************************************************************'
                           Type  Missing Values  Missing Percentage
Severity                  int64               0                0.00
Start_Time        datetime64[ns]               0                0.00
End_Time          datetime64[ns]               0                0.00
Start_Lat                float64               0                0.00
Start_Lng                float64               0                0.00
Distance(mi)             float64               0                0.00
City                      object             253                0.00
State                     object               0                0.00
Temperature(F)           float64          163853                2.12
Wind_Chill(F)            float64         1999019               25.87
Humidity(%)              float64          174144                2.25
Pressure(in)             float64          140679                1.82
Visibility(mi)           float64          177098                2.29
Wind_Direction            object          175206                2.27
Wind_Speed(mph)          float64          571233                7.39
Precipitation(in)        float64         2203586               28.51
Weather_Condition         object          173459                2.24
Amenity                     bool               0                0.00
Bump                        bool               0                0.00
Crossing                    bool               0                0.00
Give_Way                    bool               0                0.00
Junction                    bool               0                0.00
No_Exit                     bool               0                0.00
Railway                     bool               0                0.00
Roundabout                  bool               0                0.00
Station                     bool               0                0.00
Stop                        bool               0                0.00
Traffic_Calming             bool               0                0.00
Traffic_Signal              bool               0                0.00
Turning_Loop                bool               0                0.00
Sunrise_Sunset            object           23246                0.30
'********************************************************************'
```

Entrée [8]:
```
null_cols = [i for i in data.columns if data[i].isnull().any()]
msno.bar(data[null_cols]);
```



Replace Null Numerical values with mean

Entrée [9]:
```
def replace_with_mean(cols):
    for col in cols:
        mean = data[col].mean()
        print(col + ' mean : ' , mean)
        data[col].replace(np.nan, mean , inplace = True)

numerical_cols = ['Temperature(F)','Wind_Chill(F)','Humidity(%)','Pressure(in)',
          'Wind_Speed(mph)','Precipitation(in)','Visibility(mi)']
replace_with_mean(numerical_cols)
data[numerical_cols].isnull().sum()
```

```
Temperature(F) mean :  61.663285809412194
Wind_Chill(F) mean :  58.25104839532788
Humidity(%) mean :  64.83104146672403
Pressure(in) mean :  29.538985607660777
Wind_Speed(mph) mean :  7.68548959567956
Precipitation(in) mean :  0.00840720980710487
Visibility(mi) mean :  9.090376447963306
```

Out[9]:
```
Temperature(F)       0
Wind_Chill(F)        0
Humidity(%)          0
Pressure(in)         0
Wind_Speed(mph)      0
Precipitation(in)    0
Visibility(mi)       0
dtype: int64
```

Convert Temperature and Wind_Chill from Fahrenheit to Celsius

Entrée [10]:
```
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32) * 5/9
    return celsius

data['Temperature(C)'] = data['Temperature(F)'].apply(fahrenheit_to_celsius)
data['Wind_Chill(C)'] = data['Wind_Chill(F)'].apply(fahrenheit_to_celsius)
data[['Temperature(C)','Wind_Chill(C)']].head(1)
```

Out[10]:

	Temperature(C)	Wind_Chill(C)
0	2.722222	14.583916

Convert Precipitation from inches to millimeters

Entrée [11]:
```
data['Precipitation(mm)'] = data['Precipitation(in)'].apply(lambda x : x * 25.4)
data[['Precipitation(mm)']].head(1)
```

Out[11]:

	Precipitation(mm)
0	0.508

Convert Pressure from inches of mercury to Pascal

Entrée [12]:
```
data['Pressure(Pa)'] = data['Pressure(in)'].apply(lambda x : x * 3386.39)
data[['Pressure(Pa)']].head(1)
```

Out[12]:

	Pressure(Pa)
0	100508.0552

Convert Distance from miles to metres

Entrée [13]:
```
data['Distance(m)'] = data['Distance(mi)'].apply(lambda x : x * 1609.34)
data[['Distance(m)']].head(1)
```

Out[13]:

	Distance(m)
0	16.0934

Convert Visibility from miles to KM

Entrée [14]:
```
data['Visibility(Km)'] = data['Visibility(mi)'].apply(lambda x: x * 1.60934)
data[['Visibility(Km)']].head(1)
```

Out[14]:

	Visibility(Km)
0	16.0934

Entrée [15]:
```
### Delete replaced columns
data.drop(columns=['Temperature(F)','Wind_Chill(F)', 'Visibility(mi)',
                   'Precipitation(in)', 'Pressure(in)', 'Distance(mi)'], inplace = Tru
```

Entrée [16]:
```
data.columns
```

Out[16]:
```
Index(['Severity', 'Start_Time', 'End_Time', 'Start_Lat', 'Start_Lng', 'City',
       'State', 'Humidity(%)', 'Wind_Direction', 'Wind_Speed(mph)',
       'Weather_Condition', 'Amenity', 'Bump', 'Crossing', 'Give_Way',
       'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station', 'Stop',
       'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop', 'Sunrise_Sunset',
       'Temperature(C)', 'Wind_Chill(C)', 'Precipitation(mm)', 'Pressure(Pa)',
       'Distance(m)', 'Visibility(Km)'],
      dtype='object')
```

Replace non numerical data with most frequent value

Entrée [17]:
```
def replace_with_frequent_value(cols):
    for col in cols:
        frq_value = data[col].value_counts().idxmax()
        print(frq_value + ' frequent value : ' , frq_value)
        data[col].replace(np.nan, frq_value , inplace = True)

no_numerical_cols = ['Wind_Direction', 'Weather_Condition', 'Sunrise_Sunset']
replace_with_frequent_value(no_numerical_cols)
data[no_numerical_cols].isnull().sum()
```

```
CALM frequent value :  CALM
Fair frequent value :  Fair
Day frequent value :  Day
```

Out[17]:
```
Wind_Direction       0
Weather_Condition    0
Sunrise_Sunset       0
dtype: int64
```

Delete null values for City and Weather_timestamp

Entrée [18]:
```
print('All data shape: ', data.shape)
print('Shape of data with null city',data.loc[data['City'].isnull()].shape)
data.dropna(subset = ['City'], inplace = True)
print('All data shape: ', data.shape)
```

```
All data shape:  (7728394, 31)
Shape of data with null city (253, 31)
All data shape:  (7728141, 31)
```

Entrée [19]:
```
from pprint import pprint
pprint('*'*68)
pprint('-' * 10 + ' Data Description after cleansing ' + '-' * 10 )
pprint('*'*68)
pprint('# of Rows: {0[0]}    ***    # of Columns : {0[1]}'.format(data.shape))
pprint('*'*68)
data_info = pd.DataFrame(data.dtypes.reset_index()).rename(columns = {0 : 'Type'}).se

data_info = pd.concat([data_info,data.isnull().sum()], axis=1).rename(columns = {0 :
data_info = pd.concat([data_info,(data.isnull().mean()*100).round(2)], axis=1).rename

pprint(data_info)
pprint('*'*68)
```

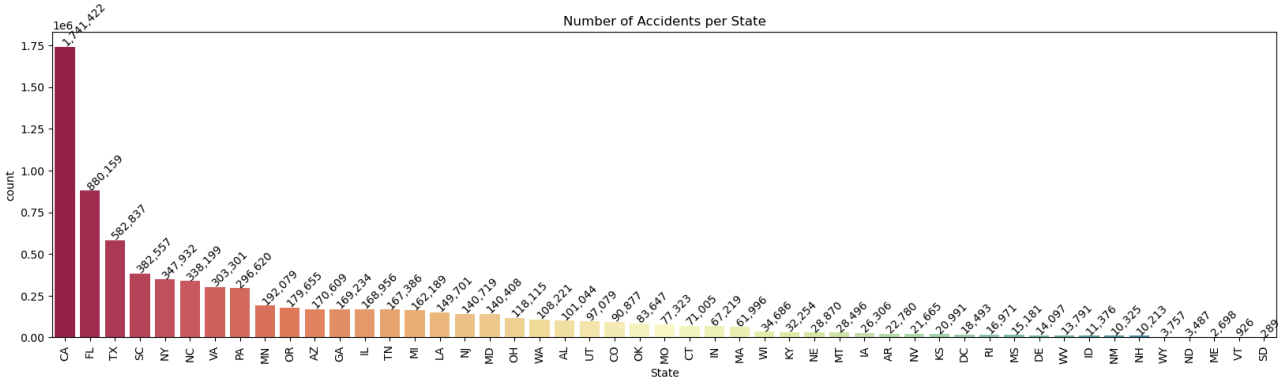
```
'********************************************************************'
'---------- Data Description after cleansing ----------'
'********************************************************************'
'# of Rows: 7728141    ***    # of Columns : 31'
'********************************************************************'
                           Type  Missing Values  Missing Percentage
Severity                  int64               0                 0.0
Start_Time        datetime64[ns]               0                 0.0
End_Time          datetime64[ns]               0                 0.0
Start_Lat                float64               0                 0.0
Start_Lng                float64               0                 0.0
City                      object               0                 0.0
State                     object               0                 0.0
Humidity(%)              float64               0                 0.0
Wind_Direction            object               0                 0.0
Wind_Speed(mph)          float64               0                 0.0
Weather_Condition         object               0                 0.0
Amenity                     bool               0                 0.0
Bump                        bool               0                 0.0
Crossing                    bool               0                 0.0
Give_Way                    bool               0                 0.0
Junction                    bool               0                 0.0
No_Exit                     bool               0                 0.0
Railway                     bool               0                 0.0
Roundabout                  bool               0                 0.0
Station                     bool               0                 0.0
Stop                        bool               0                 0.0
Traffic_Calming             bool               0                 0.0
Traffic_Signal              bool               0                 0.0
Turning_Loop                bool               0                 0.0
Sunrise_Sunset            object               0                 0.0
Temperature(C)           float64               0                 0.0
Wind_Chill(C)            float64               0                 0.0
Precipitation(mm)        float64               0                 0.0
Pressure(Pa)             float64               0                 0.0
Distance(m)              float64               0                 0.0
Visibility(Km)           float64               0                 0.0
'********************************************************************'
```

Data Visualization

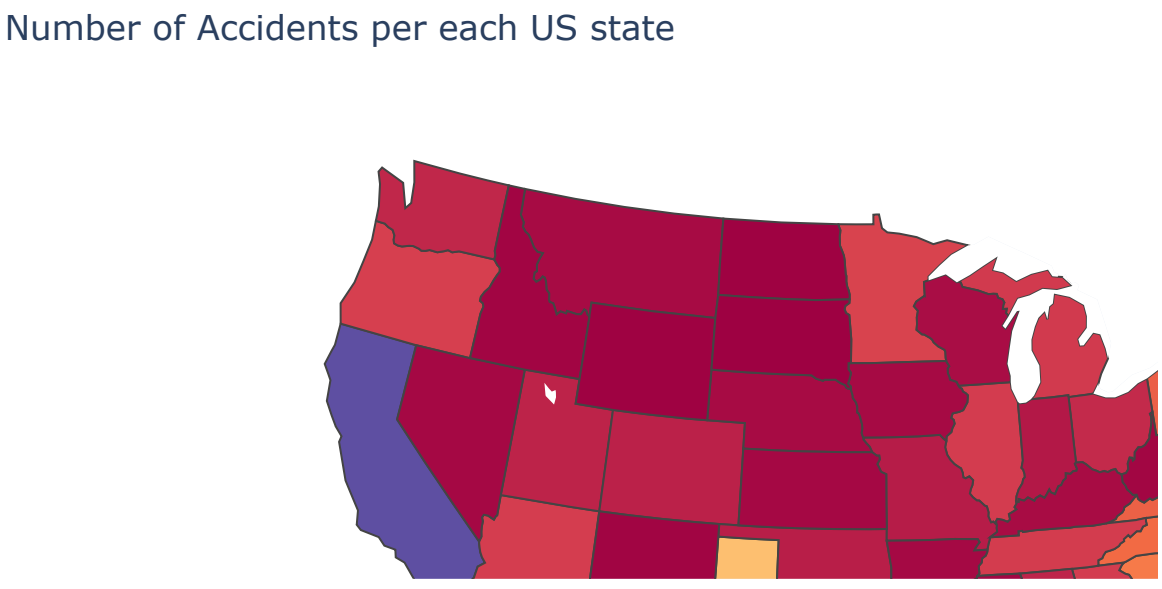
Entrée [20]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
%matplotlib inline

## function to show numbers inside figures
def show_numbers_in_figure(rotation):
    for i in ax.patches:
        count = '{:,.0f}'.format(i.get_height())
        x = i.get_x()+i.get_width()-0.60
        y = i.get_height()+5000
        ax.annotate(count, (x, y), rotation=rotation)
```

Visualize states

Entrée [21]:
```
state_names = data['State'].value_counts().index
fig, ax = plt.subplots(figsize = (20,5))
state_fig = sns.countplot(x="State", data=data, orient = 'v',
                          palette = "Spectral", order = state_names)
state_fig.set_xticklabels(state_fig.get_xticklabels(), rotation=90)
show_numbers_in_figure(rotation=45)
state_fig.set_title("Number of Accidents per State");
```

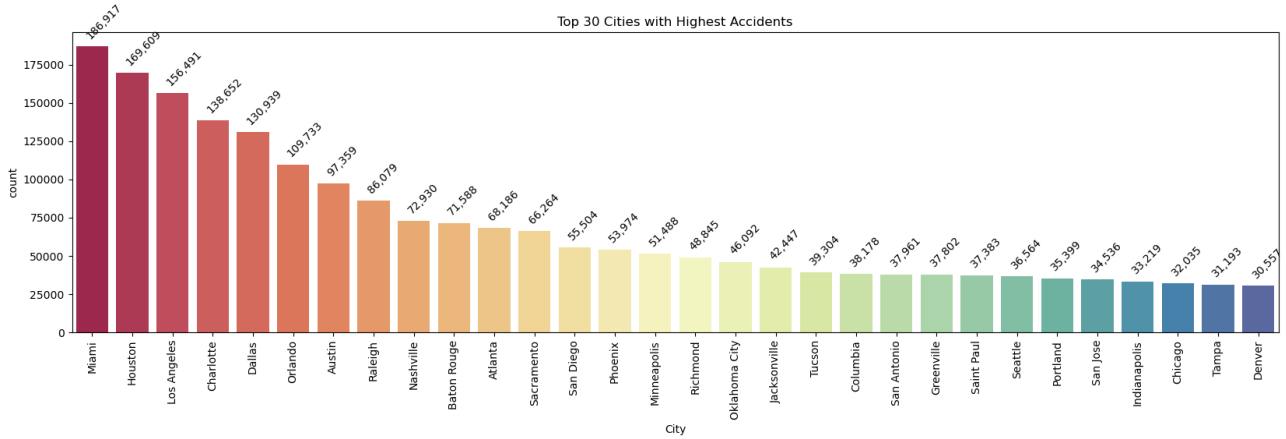


Entrée [22]:
```
states = data["State"].value_counts()
fig = go.Figure(data=go.Choropleth(locations=states.index, z=states.values.astype(flo
                                   locationmode="USA-states", colorscale="Spectral"))
fig.update_layout(title_text="Number of Accidents per each US state", geo_scope="usa"
fig.show()
```



Visualize number of accidents per City

Entrée [23]:
```
## visualize top 100 cities
cities_top_30 = data.City.value_counts().iloc[:30].index

fig, ax = plt.subplots(figsize = (20,5))
cities = sns.countplot(x="City", data = data, order = cities_top_30, orient = 'v', pa
cities.set_title("Top 30 Cities with Highest Accidents")
cities.set_xticklabels(cities.get_xticklabels(), rotation=90)
show_numbers_in_figure(rotation=45)
plt.show()
```



Create new columns for Data options

Entrée [24]:
```
data['Month'] = data['Start_Time'].dt.month
data['Year'] = data['Start_Time'].dt.year
data['Week_Number'] = data['Start_Time'].dt.strftime('%W')
data['Day'] = data['Start_Time'].dt.weekday
```

Use Month names and Day names instead of numbers

Entrée [25]:
```
conditions = [data["Month"] == i for i in range(1,13)]

month_names = [
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
]

data['Month'] = np.select(conditions, month_names)

######

conditions = [data["Day"] == i for i in range(7)]

days_of_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", '


data['Day'] = np.select(conditions, days_of_week)
```

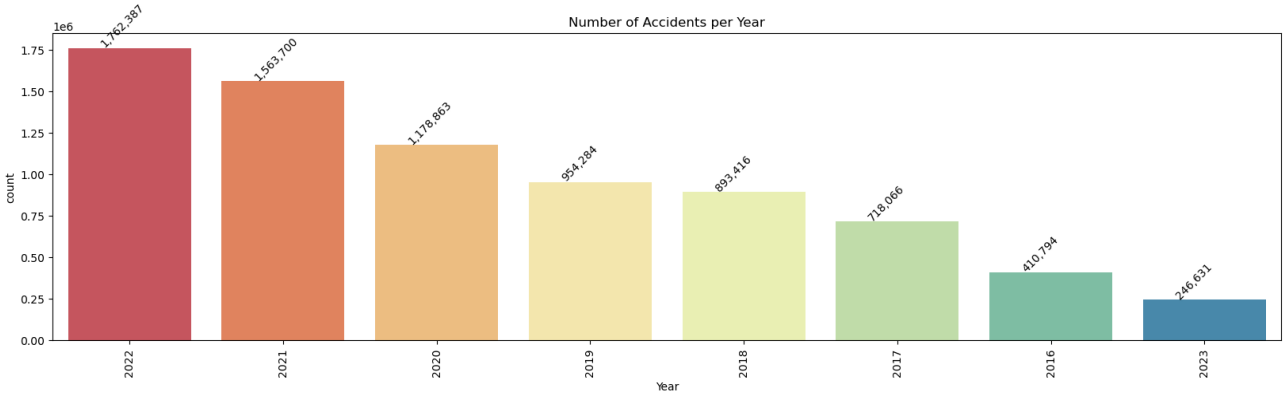
Create a season column

Entrée [26]:
```
conditions = [
                data["Month"].isin(["December", "January","February"]),
                data["Month"].isin(["March", "April", "May"]),
                data["Month"].isin(["June","July", "August"]),
                data["Month"].isin(["September", "October", "November"])
              ]
seasons = [
                "Winter",
                "Spring",
                "Summer",
                "Automn"
              ]
data["Season"] = np.select(conditions, seasons)
data["Season"].unique()
```

Out[26]:
```
array(['Winter', 'Spring', 'Summer', 'Automn'], dtype=object)
```

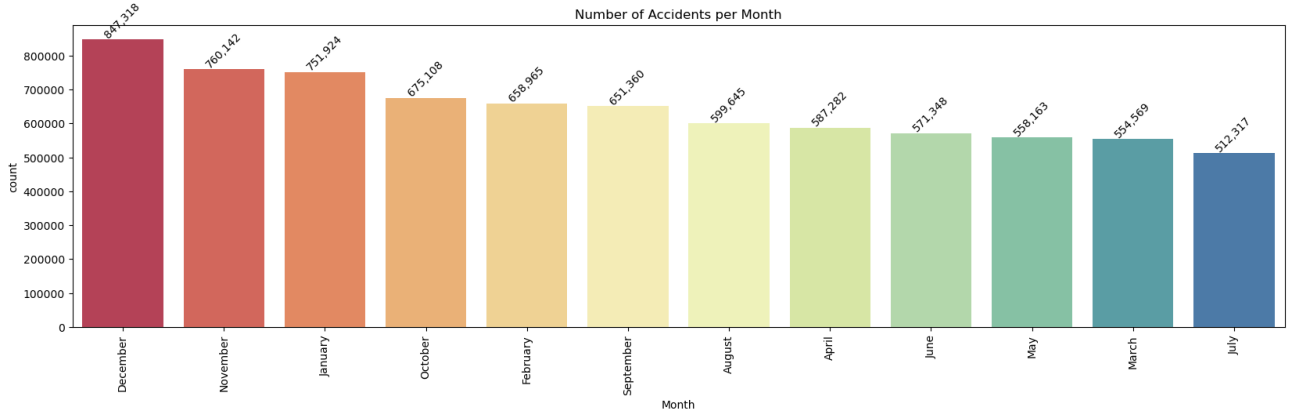
Show Number of Accidents per Year

Entrée [27]:
```
fig, ax = plt.subplots(figsize = (20,5))
years = sns.countplot(x="Year", data=data, order= data.Year.value_counts().index,
                      orient = 'v', palette = "Spectral")
years.set_title("Number of Accidents per Year")
years.set_xticklabels(years.get_xticklabels(), rotation=90)
show_numbers_in_figure(rotation=45)
plt.show()
```



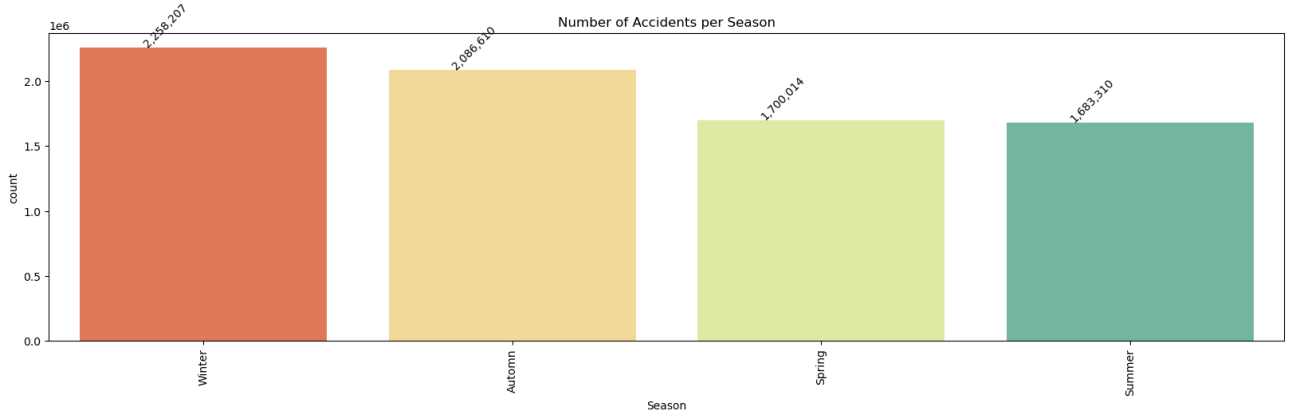
Show Number of Accidents per Month

Entrée [28]:
```
fig, ax = plt.subplots(figsize = (20,5))
months = sns.countplot(x="Month", data=data, order= data.Month.value_counts().index,
                       orient = 'v', palette = "Spectral")
months.set_title("Number of Accidents per Month")
months.set_xticklabels(months.get_xticklabels(), rotation=90)
show_numbers_in_figure(rotation=45)
plt.show()
```



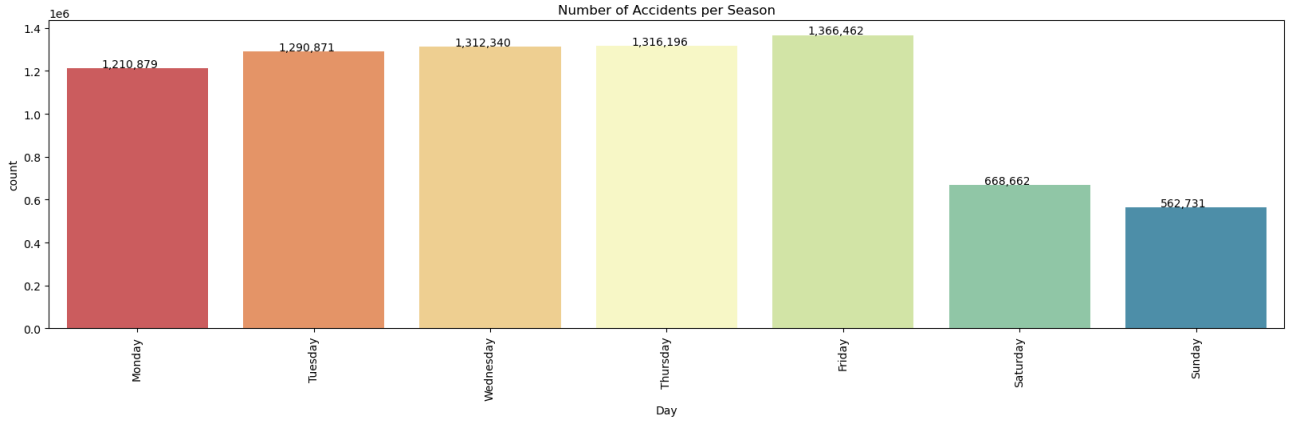
Show Number of Accidents per Season

Entrée [29]:
```
fig, ax = plt.subplots(figsize = (20,5))
season = sns.countplot(x="Season", data=data, order= data.Season.value_counts().index
                       orient = 'v', palette = "Spectral")
season.set_title("Number of Accidents per Season")
season.set_xticklabels(season.get_xticklabels(), rotation=90)
show_numbers_in_figure(rotation=45)
plt.show()
```



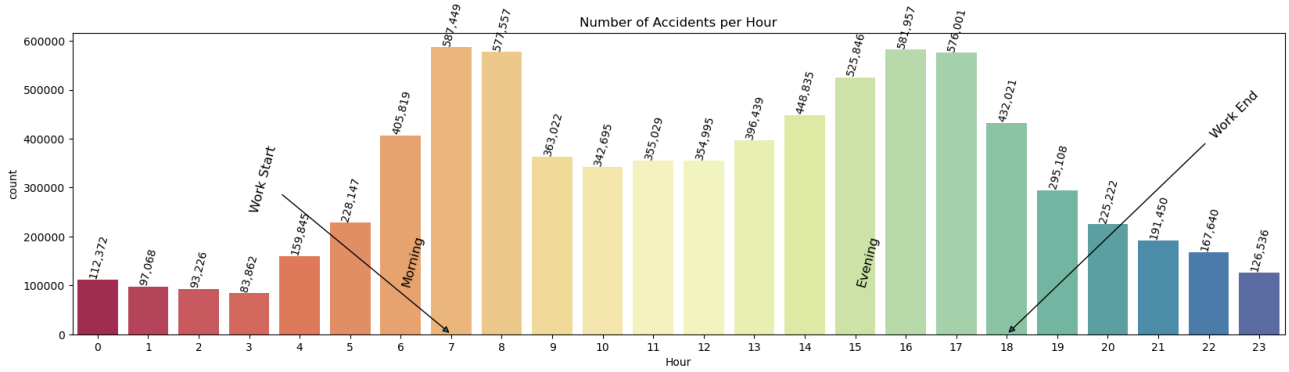
Show Number of Accidents per Day

Entrée [30]:
```
fig, ax = plt.subplots(figsize = (20,5))
week_day = sns.countplot(x="Day", data=data,orient = 'v', palette = "Spectral")
week_day.set_title("Number of Accidents per Season")
week_day.set_xticklabels(week_day.get_xticklabels(), rotation=90)
show_numbers_in_figure(rotation=0)
plt.show()
```



Check number of accidents per Hour to have insights of Working hours and off hours

Entrée [31]:
```
data['Hour'] = data['Start_Time'].dt.hour
fig, ax = plt.subplots(figsize = (20,5))
hours = sns.countplot(x="Hour", data=data, orient = 'v', palette = "Spectral")
hours.set_title("Number of Accidents per Hour")
plt.annotate('Morning',xy=(6,100000), fontsize=12, rotation=75)
plt.annotate('Evening',xy=(15,100000), fontsize=12, rotation=75)
plt.annotate('Work Start',xy=(7,0),xytext=(3,250000),arrowprops={'arrowstyle':'-|>'},
             fontsize=12, rotation=75)
plt.annotate('Work End',xy=(18,0),xytext=(22,400000),arrowprops={'arrowstyle':'-|>'},
             fontsize=12, rotation=45)
show_numbers_in_figure(rotation=75)
plt.show()
```



Split Data into different parts and Create differenet sources for it

Split data based on years and create for each part a different set (MySQL, PostgreSQL, Excel, CSV ...)

Entrée [32]:
```
data.head()
```

Out[32]:

	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	City	State	Humidity(%)	Wind_Directi
0	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	Dayton	OH	91.0	Ca
1	2	2016-02-08 06:07:59	2016-02-08 06:37:59	39.928059	-82.831184	Reynoldsburg	OH	100.0	Ca
2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	Williamsburg	OH	100.0	S
3	3	2016-02-08 07:23:34	2016-02-08 07:53:34	39.747753	-84.205582	Dayton	OH	96.0	S
4	2	2016-02-08 07:39:07	2016-02-08 08:09:07	39.627781	-84.188354	Dayton	OH	89.0	S

Entrée [33]:
```
data.columns
```

Out[33]:
```
Index(['Severity', 'Start_Time', 'End_Time', 'Start_Lat', 'Start_Lng', 'City',
       'State', 'Humidity(%)', 'Wind_Direction', 'Wind_Speed(mph)',
       'Weather_Condition', 'Amenity', 'Bump', 'Crossing', 'Give_Way',
       'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station', 'Stop',
       'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop', 'Sunrise_Sunset',
       'Temperature(C)', 'Wind_Chill(C)', 'Precipitation(mm)', 'Pressure(Pa)',
       'Distance(m)', 'Visibility(Km)', 'Month', 'Year', 'Week_Number', 'Day',
       'Season', 'Hour'],
      dtype='object')
```

Entrée [34]:
```
data.Year.unique()
```

Out[34]:
```
array([2016, 2017, 2022, 2021, 2020, 2019, 2018, 2023], dtype=int64)
```

Save 2016 data into mysql

Entrée [35]:
```
data_2016 = data.loc[data['Year'] == 2016]
print('Number of accidents in 2016: ', data_2016.shape)
```

```
Number of accidents in 2016:  (410794, 37)
```

Entrée [36]:
```
engine = create_engine('mysql+pymysql://root:@localhost/usa_accidents_2016')
cnx = engine.connect()
data_2016.to_sql("usa_accidents", cnx, if_exists='replace', index=None)
cnx.close()
```

Save 2017 data into Excel

Entrée [37]:
```
data_2017 = data.loc[data['Year'] == 2017]
print('Number of accidents in 2017: ', data_2017.shape)
```

```
Number of accidents in 2017:  (718066, 37)
```

Entrée [38]:
```
data_2017.to_excel('Data/USA_Accidents_2017.xlsx', sheet_name='data', index=None)
```

Save 2018 data into json

Entrée [39]:
```
data_2018 = data.loc[data['Year'] == 2018]
print('Number of accidents in 2018: ', data_2018.shape)
```

```
Number of accidents in 2018:  (893416, 37)
```

Entrée [40]:
```
data_2018.to_json('Data/USA_Accidents_2018.json')
```

Save 2019 data to excel

Entrée [41]:
```
data_2019 = data.loc[data['Year'] == 2019]
print('Number of accidents in 2019: ', data_2019.shape)
```

```
Number of accidents in 2019:  (954284, 37)
```

Entrée [42]:
```
data_2019.to_excel('Data/USA_Accidents_2019.xlsx', sheet_name = 'data', index = None)
```

Save 2020 and 2021 data to postgresql

Entrée [43]:
```
data_2020 = data.loc[data['Year'] == 2020]
print('Number of accidents in 2020: ', data_2020.shape)
```

```
Number of accidents in 2020:  (1178863, 37)
```

Entrée [45]:
```
engine = create_engine('postgresql://postgres:password@localhost:5432/usa_accidents_2
data_2020.to_sql('usa_accidents', engine,if_exists='replace', index=None)
```

Out[45]:
```
863
```

Entrée [46]:
```
data_2021 = data.loc[data['Year'] == 2021]
print('Number of accidents in 2021: ', data_2021.shape)
```

```
Number of accidents in 2021:  (1563700, 37)
```

Entrée [47]:
```
engine1 = create_engine('postgresql://postgres:password@localhost:5432/usa_accidents_
data_2021.to_sql('usa_accidents', engine1,if_exists='replace')
```

Out[47]:
```
700
```

Save 2022 data to csv

Entrée [48]:
```
data_2022 = data.loc[data['Year'] == 2022]
print('Number of accidents in 2022: ', data_2022.shape)
```

```
Number of accidents in 2022:  (1762387, 37)
```

Entrée [49]:
```
data_2022.to_csv('Data/USA_Accidents_2022.csv', sep=',', index=None)
```

Save 2023 data to mysql

Entrée [52]:
```
data_2023 = data.loc[data['Year'] == 2023]
print('Number of accidents in 2023: ', data_2023.shape)
```

```
Number of accidents in 2023:  (246631, 37)
```

Entrée [53]:
```
engine = create_engine('mysql+pymysql://root:@localhost/usa_accidents_2023')
cnx = engine.connect()
data_2023.to_sql("usa_accidents", cnx, if_exists='replace')
cnx.close()
```