testaccidentMODEL

November 27, 2023

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import calendar

import datetime as dt
import math
import scipy.stats as stats
import scipy
import matplotlib.pyplot as plt

from matplotlib import ticker
import matplotlib.font_manager as font_manager
from matplotlib.ticker import FuncFormatter
import matplotlib.ticker as ticker
import matplotlib.patches as mpatches
import matplotlib.patheffects as PathEffects


import plotly as pt
from plotly import graph_objs as go
import plotly.express as px
import plotly.figure_factory as ff
from pylab import *

from shapely.geometry import Point
import geopandas as gpd
import geoplot
from geopy.geocoders import Nominatim

import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
import random
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
# nltk.download("stopwords")
```

```
stop_words=stopwords.words("english")
new_stopping_words = stop_words[:len(stop_words)-36]
new_stopping_words.remove("not")
# nltk.download('punkt')
```

[ ]:
```
import statsmodels.api as sm
from scipy.stats import chi2_contingency
from scipy.stats import chi2
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

[ ]:
```
# dataset = pd.read_csv('./dataset/US_Accidents_March23.csv')
dataset = pd.read_csv('./dataset/US_Accidents_March23_sampled_500k.csv')
```

[ ]:
```
dataset.head()
```

[ ]:
```
           ID   Source  Severity                    Start_Time  \
0  A-2047758  Source2         2           2019-06-12 10:10:56
1  A-4694324  Source1         2  2022-12-03 23:37:14.000000000
2  A-5006183  Source1         2  2022-08-20 13:13:00.000000000
3  A-4237356  Source1         2           2022-02-21 17:43:04
4  A-6690583  Source1         2           2020-12-04 01:46:00

                      End_Time  Start_Lat   Start_Lng    End_Lat  \
0           2019-06-12 10:55:58  30.641211  -91.153481        NaN
1  2022-12-04 01:56:53.000000000  38.990562  -77.399070  38.990037
2  2022-08-20 15:22:45.000000000  34.661189 -120.492822  34.661189
3           2022-02-21 19:43:23  43.680592  -92.993317  43.680574
4           2020-12-04 04:13:09  35.395484 -118.985176  35.395476

      End_Lng  Distance(mi)  … Roundabout Station   Stop Traffic_Calming  \
0         NaN         0.000  …      False   False  False           False
1  -77.398282         0.056  …      False   False  False           False
2 -120.492442         0.022  …      False   False  False           False
3  -92.972223         1.054  …      False   False  False           False
4 -118.985995         0.046  …      False   False  False           False

  Traffic_Signal Turning_Loop Sunrise_Sunset Civil_Twilight Nautical_Twilight  \
0           True        False            Day            Day              Day
1          False        False          Night          Night            Night
2           True        False            Day            Day              Day
3          False        False            Day            Day              Day
4          False        False          Night          Night            Night

  Astronomical_Twilight
0                   Day
1                 Night
```

```
2                    Day
3                    Day
4                  Night

[5 rows x 46 columns]
```

[ ]:
```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
numeric_df = dataset.select_dtypes(include=numerics)
print("There are", len(numeric_df.columns), "numeric columns.")
```

```
There are 13 numeric columns.
```

[ ]:
```
missing_percentages = dataset.isna().sum().sort_values(ascending=False) /␣
↪len(dataset)
```

[ ]:
```
missing_percentages.round(5)
```

[ ]:
```
End_Lat                 0.44075
End_Lng                 0.44075
Precipitation(in)       0.28523
Wind_Chill(F)           0.25803
Wind_Speed(mph)         0.07397
Visibility(mi)          0.02258
Wind_Direction          0.02239
Humidity(%)             0.02226
Weather_Condition       0.02220
Temperature(F)          0.02093
Pressure(in)            0.01786
Weather_Timestamp       0.01535
Nautical_Twilight       0.00297
Civil_Twilight          0.00297
Sunrise_Sunset          0.00297
Astronomical_Twilight   0.00297
Airport_Code            0.00289
Street                  0.00138
Timezone                0.00101
Zipcode                 0.00023
City                    0.00004
Description             0.00000
Traffic_Signal          0.00000
Roundabout              0.00000
Station                 0.00000
Stop                    0.00000
Traffic_Calming         0.00000
Country                 0.00000
Turning_Loop            0.00000
No_Exit                 0.00000
End_Time                0.00000
```

```
Start_Time              0.00000
Severity                0.00000
Railway                 0.00000
Crossing                0.00000
Junction                0.00000
Give_Way                0.00000
Bump                    0.00000
Amenity                 0.00000
Start_Lat               0.00000
Start_Lng               0.00000
Distance(mi)            0.00000
Source                  0.00000
County                  0.00000
State                   0.00000
ID                      0.00000
dtype: float64
```

[ ]:
```
dataset.columns
```

[ ]:
```
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')
```

[ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500000 entries, 0 to 499999
Data columns (total 46 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   ID                     500000 non-null  object
 1   Source                 500000 non-null  object
 2   Severity               500000 non-null  int64
 3   Start_Time             500000 non-null  object
 4   End_Time               500000 non-null  object
 5   Start_Lat              500000 non-null  float64
 6   Start_Lng              500000 non-null  float64
 7   End_Lat                279623 non-null  float64
 8   End_Lng                279623 non-null  float64
 9   Distance(mi)           500000 non-null  float64
```

```
 10  Description            499999 non-null  object
 11  Street                 499309 non-null  object
 12  City                   499981 non-null  object
 13  County                 500000 non-null  object
 14  State                  500000 non-null  object
 15  Zipcode                499884 non-null  object
 16  Country                500000 non-null  object
 17  Timezone               499493 non-null  object
 18  Airport_Code           498554 non-null  object
 19  Weather_Timestamp      492326 non-null  object
 20  Temperature(F)         489534 non-null  float64
 21  Wind_Chill(F)          370983 non-null  float64
 22  Humidity(%)            488870 non-null  float64
 23  Pressure(in)           491072 non-null  float64
 24  Visibility(mi)         488709 non-null  float64
 25  Wind_Direction         488803 non-null  object
 26  Wind_Speed(mph)        463013 non-null  float64
 27  Precipitation(in)      357384 non-null  float64
 28  Weather_Condition      488899 non-null  object
 29  Amenity                500000 non-null  bool
 30  Bump                   500000 non-null  bool
 31  Crossing               500000 non-null  bool
 32  Give_Way               500000 non-null  bool
 33  Junction               500000 non-null  bool
 34  No_Exit                500000 non-null  bool
 35  Railway                500000 non-null  bool
 36  Roundabout             500000 non-null  bool
 37  Station                500000 non-null  bool
 38  Stop                   500000 non-null  bool
 39  Traffic_Calming        500000 non-null  bool
 40  Traffic_Signal         500000 non-null  bool
 41  Turning_Loop           500000 non-null  bool
 42  Sunrise_Sunset         498517 non-null  object
 43  Civil_Twilight         498517 non-null  object
 44  Nautical_Twilight      498517 non-null  object
 45  Astronomical_Twilight  498517 non-null  object
dtypes: bool(13), float64(12), int64(1), object(20)
memory usage: 132.1+ MB
```

[ ]:
```
# dataset.isnull().sum()*100/len(dataset)
dataset.shape
```

[ ]:
```
(500000, 46)
```

[ ]:
```
dataset.Source.unique()
```

[ ]:
```
array(['Source2', 'Source1', 'Source3'], dtype=object)
```

[ ]:
```
dataset.describe()
```

[ ]:
```
            Severity      Start_Lat      Start_Lng        End_Lat  \
count  500000.000000  500000.000000  500000.000000  279623.000000
mean        2.212748      36.206421     -94.736583      36.273192
std         0.486661       5.071411      17.405761       5.265333
min         1.000000      24.562117    -124.497420      24.570110
25%         2.000000      33.416823    -117.233047      33.474773
50%         2.000000      35.832147     -87.794365      36.192669
75%         2.000000      40.082443     -80.359601      40.181341
max         4.000000      48.999569     -67.484130      48.998901

             End_Lng   Distance(mi)  Temperature(F)  Wind_Chill(F)  \
count  279623.000000  500000.000000   489534.000000  370983.000000
mean      -95.776553       0.564317       61.646254      58.229028
std        18.120211       1.774872       19.000133      22.352246
min      -124.497419       0.000000      -77.800000     -53.200000
25%      -117.778324       0.000000       49.000000      43.000000
50%       -88.039013       0.029000       64.000000      62.000000
75%       -80.252449       0.465000       76.000000      75.000000
max       -67.484130     193.479996      207.000000     207.000000

         Humidity(%)   Pressure(in)  Visibility(mi)  Wind_Speed(mph)  \
count  488870.000000  491072.000000   488709.000000    463013.000000
mean       64.834921      29.536621        9.091540         7.681347
std        22.826158       1.008666        2.708083         5.431361
min         1.000000       0.120000        0.000000         0.000000
25%        48.000000      29.370000       10.000000         4.600000
50%        67.000000      29.860000       10.000000         7.000000
75%        84.000000      30.030000       10.000000        10.400000
max       100.000000      38.440000      130.000000       822.800000

       Precipitation(in)
count      357384.000000
mean            0.008289
std             0.101865
min             0.000000
25%             0.000000
50%             0.000000
75%             0.000000
max            10.130000
```

[ ]:
```
# remove rows with zero values
# dataset = dataset.dropna()
# missing_values = dataset.isna()
# missing_counts = dataset.isna().sum()
# total_missing = dataset.isna().sum().sum()
```

[ ]:
```
dataset.shape
```

[ ]: (500000, 46)

[ ]:
```
dataset.head()
```

[ ]:
```
           ID   Source  Severity                    Start_Time  \
0  A-2047758  Source2         2           2019-06-12 10:10:56
1  A-4694324  Source1         2  2022-12-03 23:37:14.000000000
2  A-5006183  Source1         2  2022-08-20 13:13:00.000000000
3  A-4237356  Source1         2           2022-02-21 17:43:04
4  A-6690583  Source1         2           2020-12-04 01:46:00

                      End_Time  Start_Lat   Start_Lng    End_Lat  \
0           2019-06-12 10:55:58  30.641211  -91.153481        NaN
1  2022-12-04 01:56:53.000000000  38.990562  -77.399070  38.990037
2  2022-08-20 15:22:45.000000000  34.661189 -120.492822  34.661189
3           2022-02-21 19:43:23  43.680592  -92.993317  43.680574
4           2020-12-04 04:13:09  35.395484 -118.985176  35.395476

      End_Lng  Distance(mi)  … Roundabout Station   Stop Traffic_Calming  \
0         NaN         0.000  …      False   False  False           False
1  -77.398282         0.056  …      False   False  False           False
2 -120.492442         0.022  …      False   False  False           False
3  -92.972223         1.054  …      False   False  False           False
4 -118.985995         0.046  …      False   False  False           False

  Traffic_Signal Turning_Loop Sunrise_Sunset Civil_Twilight Nautical_Twilight  \
0           True        False            Day            Day              Day
1          False        False          Night          Night            Night
2           True        False            Day            Day              Day
3          False        False            Day            Day              Day
4          False        False          Night          Night            Night

  Astronomical_Twilight
0                   Day
1                 Night
2                   Day
3                   Day
4                 Night

[5 rows x 46 columns]
```

[ ]:
```
# Change type from INT to TEXT (betetr classification analysis)
# dataset["Severity"]=dataset["Severity"].apply(lambda x : str(x))

# change format of time from FLOAT to DATE
```

```
dataset['Start_Time'] = pd.to_datetime(dataset["Start_Time"], errors="coerce")
dataset['End_Time'] = pd.to_datetime(dataset["End_Time"], errors="coerce")
# Add new column YEAR, MONTH , DAY and Hour ... to create new features
dataset["Year"]=dataset["Start_Time"].dt.year
dataset["Hour"]=dataset["Start_Time"].dt.hour
dataset["Month"]=dataset["Start_Time"].dt.month
dataset["Day"]=dataset["Start_Time"].dt.day_name()
```

[ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500000 entries, 0 to 499999
Data columns (total 50 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   ID                     500000 non-null  object
 1   Source                 500000 non-null  object
 2   Severity               500000 non-null  int64
 3   Start_Time             500000 non-null  datetime64[ns]
 4   End_Time               500000 non-null  datetime64[ns]
 5   Start_Lat              500000 non-null  float64
 6   Start_Lng              500000 non-null  float64
 7   End_Lat                279623 non-null  float64
 8   End_Lng                279623 non-null  float64
 9   Distance(mi)           500000 non-null  float64
 10  Description            499999 non-null  object
 11  Street                 499309 non-null  object
 12  City                   499981 non-null  object
 13  County                 500000 non-null  object
 14  State                  500000 non-null  object
 15  Zipcode                499884 non-null  object
 16  Country                500000 non-null  object
 17  Timezone               499493 non-null  object
 18  Airport_Code           498554 non-null  object
 19  Weather_Timestamp      492326 non-null  object
 20  Temperature(F)         489534 non-null  float64
 21  Wind_Chill(F)          370983 non-null  float64
 22  Humidity(%)            488870 non-null  float64
 23  Pressure(in)           491072 non-null  float64
 24  Visibility(mi)         488709 non-null  float64
 25  Wind_Direction         488803 non-null  object
 26  Wind_Speed(mph)        463013 non-null  float64
 27  Precipitation(in)      357384 non-null  float64
 28  Weather_Condition      488899 non-null  object
 29  Amenity                500000 non-null  bool
 30  Bump                   500000 non-null  bool
 31  Crossing               500000 non-null  bool
 32  Give_Way               500000 non-null  bool
```

```
 33  Junction                500000 non-null  bool
 34  No_Exit                 500000 non-null  bool
 35  Railway                 500000 non-null  bool
 36  Roundabout              500000 non-null  bool
 37  Station                 500000 non-null  bool
 38  Stop                    500000 non-null  bool
 39  Traffic_Calming         500000 non-null  bool
 40  Traffic_Signal          500000 non-null  bool
 41  Turning_Loop            500000 non-null  bool
 42  Sunrise_Sunset          498517 non-null  object
 43  Civil_Twilight          498517 non-null  object
 44  Nautical_Twilight       498517 non-null  object
 45  Astronomical_Twilight   498517 non-null  object
 46  Year                    500000 non-null  int64
 47  Hour                    500000 non-null  int64
 48  Month                   500000 non-null  int64
 49  Day                     500000 non-null  object
dtypes: bool(13), datetime64[ns](2), float64(12), int64(4), object(19)
memory usage: 147.3+ MB
```

[ ]:
```
dataset.head(5)
```

[ ]:

	ID	Source	Severity	Start_Time	End_Time	\
0	A-2047758	Source2	2	2019-06-12 10:10:56	2019-06-12 10:55:58	
1	A-4694324	Source1	2	2022-12-03 23:37:14	2022-12-04 01:56:53	
2	A-5006183	Source1	2	2022-08-20 13:13:00	2022-08-20 15:22:45	
3	A-4237356	Source1	2	2022-02-21 17:43:04	2022-02-21 19:43:23	
4	A-6690583	Source1	2	2020-12-04 01:46:00	2020-12-04 04:13:09	

	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	…	\
0	30.641211	-91.153481	NaN	NaN	0.000	…	
1	38.990562	-77.399070	38.990037	-77.398282	0.056	…	
2	34.661189	-120.492822	34.661189	-120.492442	0.022	…	
3	43.680592	-92.993317	43.680574	-92.972223	1.054	…	
4	35.395484	-118.985176	35.395476	-118.985995	0.046	…	

	Traffic_Signal	Turning_Loop	Sunrise_Sunset	Civil_Twilight	Nautical_Twilight	\
0	True	False	Day	Day	Day	
1	False	False	Night	Night	Night	
2	True	False	Day	Day	Day	
3	False	False	Day	Day	Day	
4	False	False	Night	Night	Night	

	Astronomical_Twilight	Year	Hour	Month	Day
0	Day	2019	10	6	Wednesday
1	Night	2022	23	12	Saturday
2	Day	2022	13	8	Saturday

```
3                  Day  2022   17     2     Monday
4                Night  2020    1    12     Friday

[5 rows x 50 columns]
```

[ ]:
```
dataset.describe()
```

[ ]:
```
            Severity      Start_Lat      Start_Lng        End_Lat  \
count  500000.000000  500000.000000  500000.000000  279623.000000
mean        2.212748      36.206421     -94.736583      36.273192
std         0.486661       5.071411      17.405761       5.265333
min         1.000000      24.562117    -124.497420      24.570110
25%         2.000000      33.416823    -117.233047      33.474773
50%         2.000000      35.832147     -87.794365      36.192669
75%         2.000000      40.082443     -80.359601      40.181341
max         4.000000      48.999569     -67.484130      48.998901

             End_Lng   Distance(mi)  Temperature(F)  Wind_Chill(F)  \
count  279623.000000  500000.000000   489534.000000  370983.000000
mean      -95.776553       0.564317       61.646254      58.229028
std        18.120211       1.774872       19.000133      22.352246
min      -124.497419       0.000000      -77.800000     -53.200000
25%      -117.778324       0.000000       49.000000      43.000000
50%       -88.039013       0.029000       64.000000      62.000000
75%       -80.252449       0.465000       76.000000      75.000000
max       -67.484130     193.479996      207.000000     207.000000

         Humidity(%)   Pressure(in)  Visibility(mi)  Wind_Speed(mph)  \
count  488870.000000  491072.000000   488709.000000    463013.000000
mean       64.834921      29.536621        9.091540         7.681347
std        22.826158       1.008666        2.708083         5.431361
min         1.000000       0.120000        0.000000         0.000000
25%        48.000000      29.370000       10.000000         4.600000
50%        67.000000      29.860000       10.000000         7.000000
75%        84.000000      30.030000       10.000000        10.400000
max       100.000000      38.440000      130.000000       822.800000

       Precipitation(in)           Year           Hour          Month
count      357384.000000  500000.000000  500000.000000  500000.000000
mean            0.008289    2019.906596      12.321320       6.702528
std             0.101865       1.913944       5.471098       3.639946
min             0.000000    2016.000000       0.000000       1.000000
25%             0.000000    2018.000000       8.000000       3.000000
50%             0.000000    2020.000000      13.000000       7.000000
75%             0.000000    2022.000000      17.000000      10.000000
max            10.130000    2023.000000      23.000000      12.000000
```

[ ]:
```
dataset.Source.unique()
```

[ ]:
```
array(['Source2', 'Source1', 'Source3'], dtype=object)
```

[ ]:
```
import pandas as pd
import plotly.graph_objects as go

# Read in the CSV file

# Group the data by state and count the number of accidents in each state
state_counts = dataset.groupby('State')['ID'].count().
 ↪sort_values(ascending=False)

# Create the plotly bar chart
fig = go.Figure([go.Bar(x=state_counts.index, y=state_counts.values)])

# Set the plot title and axis labels
fig.update_layout(title='US State wise Accidents data',
                  xaxis_title='State',
                  yaxis_title='Number of Accidents')

# Show the plot
fig.show()
```

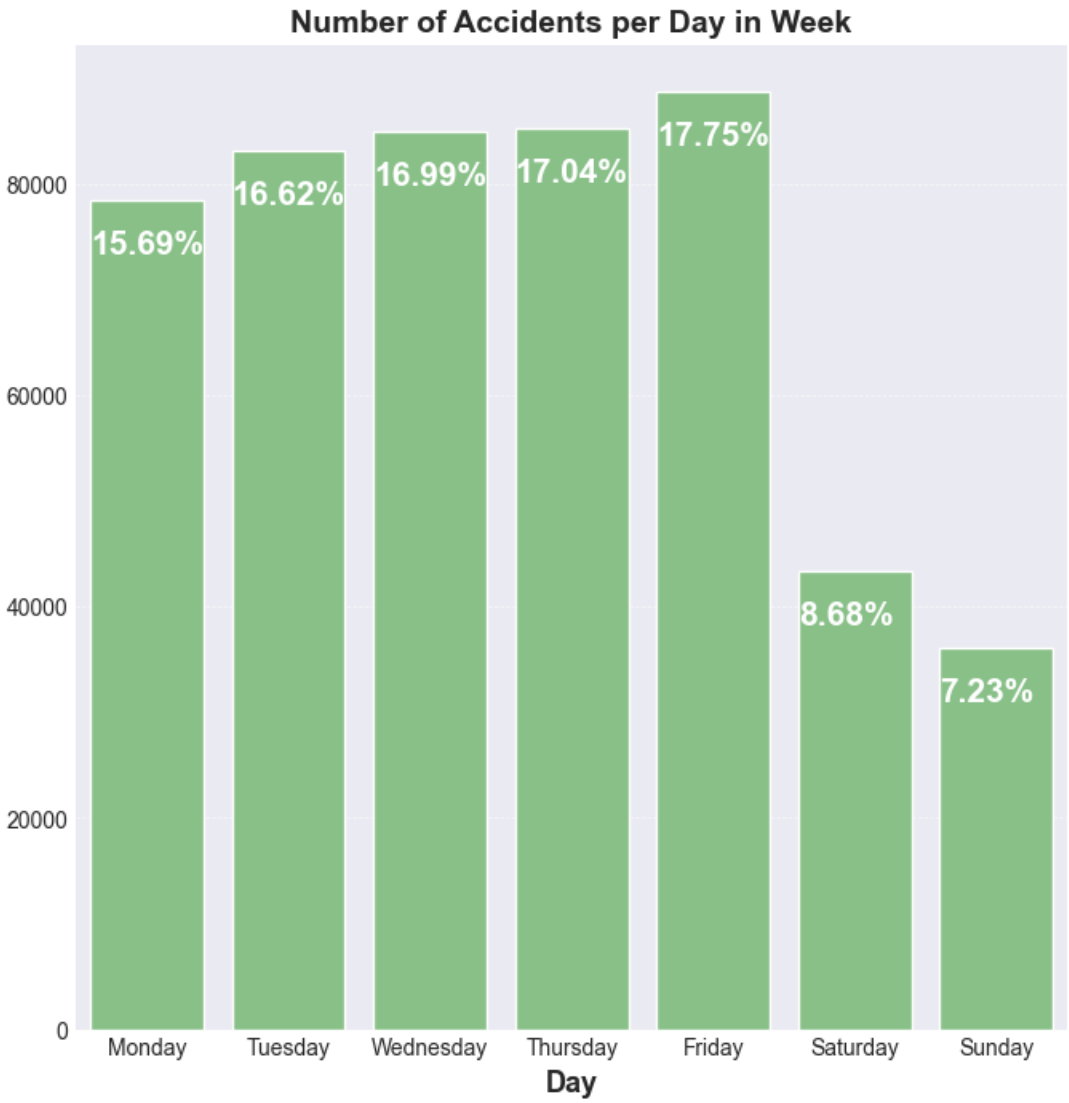
[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
sns.set_style("darkgrid")
```

[ ]:
```
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-
nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight',
'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright',
'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette',
'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted',
'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel',
'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks',
'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

[ ]:
```
plt.style.use('seaborn-v0_8-dark')

custom_palette = sns.color_palette("Accent", 7)

# Change the default color palette to the custom palette
sns.set_palette(custom_palette)
```

```
plt.figure(figsize=(8, 8))
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)

order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',␣
 ↪'Sunday']
sns.countplot(x=dataset["Day"], order=order)

# Calculate the total counts
total_counts = dataset["Day"].value_counts()

plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

plt.xlabel("Day", fontsize=13, fontweight="bold")
plt.ylabel(" ")
plt.title("Number of Accidents per Day in Week", fontsize=14, fontweight="bold")

ax = plt.gca()

for i, bar in enumerate(ax.patches):
    proportion = (total_counts[order[i]]/total_counts.sum()) * 100
    ax.text(
        bar.get_x(),
        bar.get_height()-5000,
        f'{proportion:.2f}%',
        fontsize=15,
        weight='bold',
        color='white'
    )

plt.show()
```

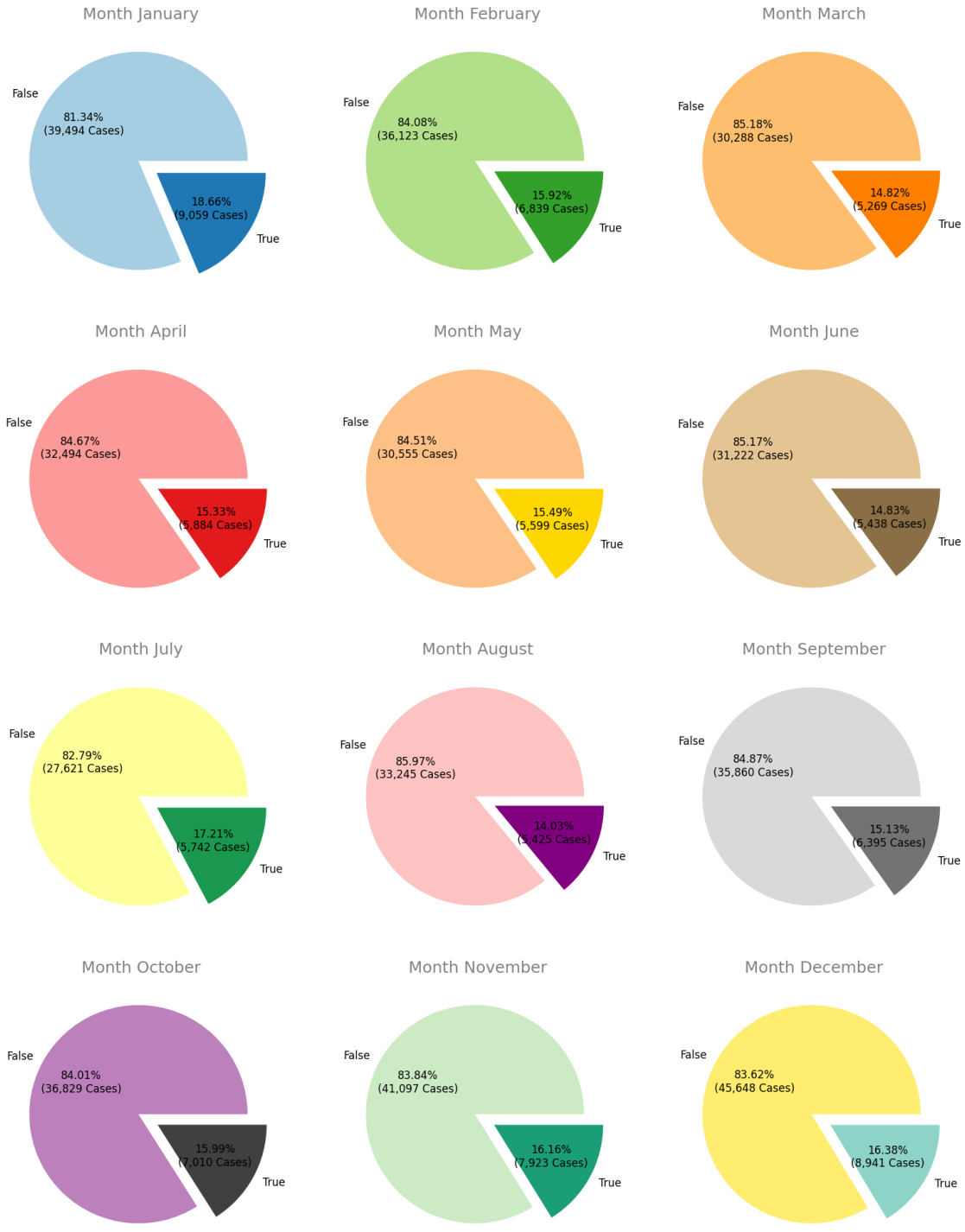


[ ]:
```
sns.reset_defaults()
```

[ ]:
```
dataset["Is_Weekend"] = dataset["Day"].isin(["Saturday", "Sunday"])
```

[ ]:
```
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(16, 20))

month_map = {1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6:␣
 ↪'June', 7: 'July', 8: 'August', 9: 'September', 10: 'October', 11:␣
 ↪'November', 12: 'December'}

colors = [('#A6CEE3', '#1F78B4'), ('#B2DF8A', '#33A02C'), ('#FDBF6F',␣
 ↪'#FF7F00'), ('#FB9A99', '#E31A1C'),
```

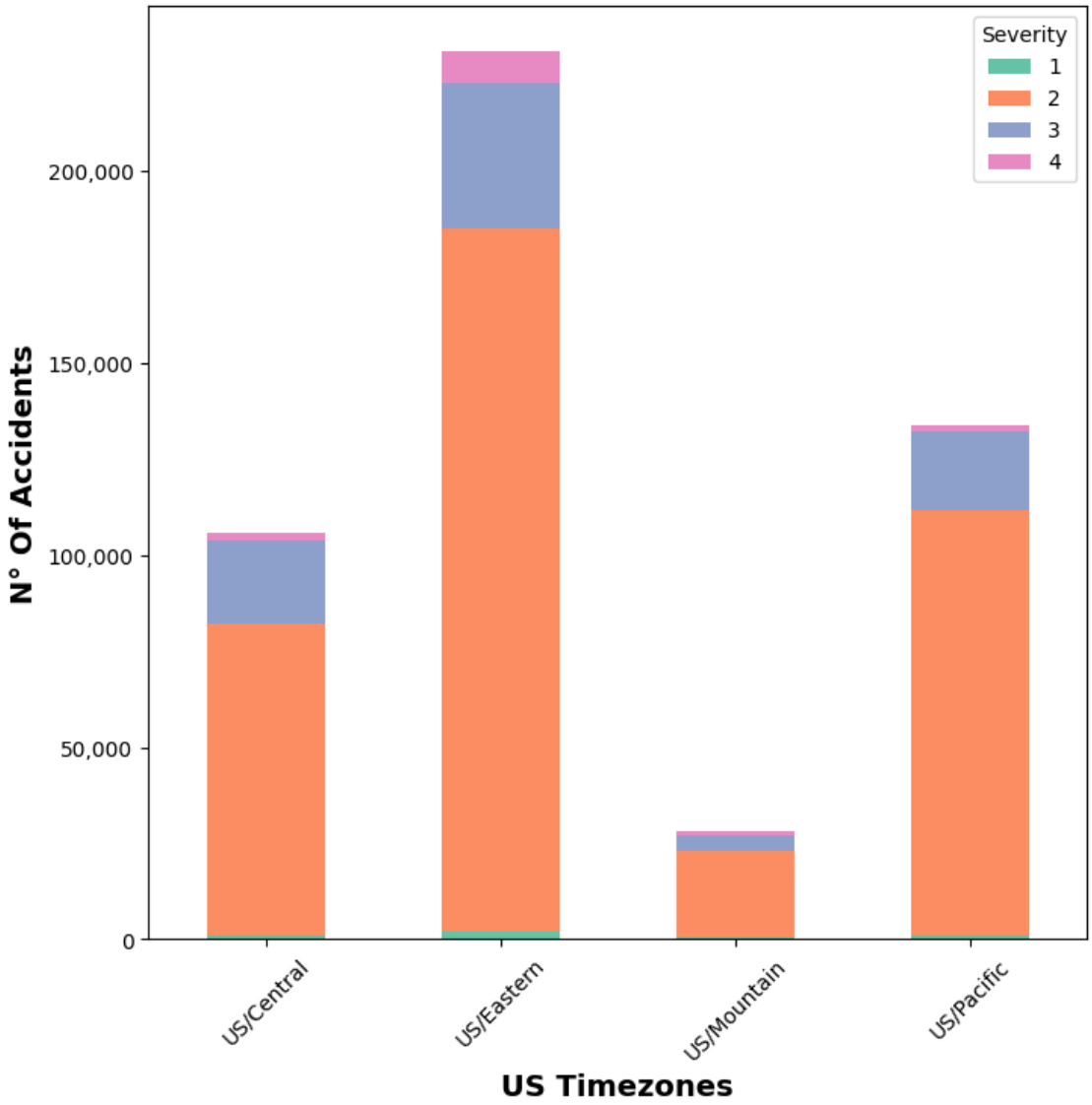
```
          ('#FDC086', '#FFD700'), ('#E5C494', '#8A6E45'), ('#FFFF99',␣
 ↪'#1A9850'), ('#FDC3C3', '#800080'),
          ('#D9D9D9', '#737373'), ('#BC80BD', '#404040'), ('#CCEBC5',␣
 ↪'#1B9E77'), ('#FFED6F', '#8DD3C7')]



count = 0

def func(pct, allvals):
    absolute = int(round(pct / 100 * np.sum(allvals), 2))
    return "{:.2f}%\n({:,d} Cases)".format(pct, absolute)

for i, ax in enumerate(axes.flatten()):
    month = i + 1
    size = list(dataset[dataset["Month"] == month]["Is_Weekend"].value_counts())
    if len(size) != 2:
        size.append(0)

    labels = ['False', 'True']

    ax.pie(size, labels=labels, colors=colors[count],
           autopct=lambda pct: func(pct, size), labeldistance=1.1,
           textprops={'fontsize': 12}, explode=[0, 0.2])

    title = '\n Month {}'.format(month_map[month])

    ax.set_title(title, fontsize=18, color='grey')

    count += 1

plt.tight_layout()
plt.show()
```



[ ]:
```
crosstab=pd.crosstab(dataset["Timezone"],dataset["Severity"])
crosstab
```

[ ]:
```
Severity        1       2      3     4
Timezone
US/Central    783   81312  21737  2180
US/Eastern   2023  183147  37957  8270
US/Mountain   598   22320   4139  1040
US/Pacific    862  110923  20640  1562
```

[ ]:
```
sns.set_palette("Set2")


crosstab.plot(kind="bar",stacked=True,
              figsize=(8,8),
             rot=45,
             fontsize=10
             )
label_font_bold = font_manager.FontProperties(weight='bold')
plt.xlabel("US Timezones",fontsize=14,fontproperties=label_font_bold)
plt.ylabel("N° Of Accidents",fontsize=14,fontproperties=label_font_bold)

formatter = ticker.FuncFormatter(lambda x, pos: '{:,.0f}'.format(x))
plt.gca().yaxis.set_major_formatter(formatter)

plt.show()
```



[ ]:
```
# dataset = pd.read_csv("./US_Accidents_March23.csv")
dataset['Start_Time'] = pd.to_datetime(dataset["Start_Time"], errors="coerce")
dataset['End_Time'] = pd.to_datetime(dataset["End_Time"], errors="coerce")
dataset["Year"]=dataset["Start_Time"].dt.year
dataset["Hour"]=dataset["Start_Time"].dt.hour
dataset["Month"]=dataset["Start_Time"].dt.month
dataset.tail()
```

[ ]:
```
                ID    Source  Severity          Start_Time            End_Time  \
499995  A-6077227  Source1         2 2021-12-15 07:30:00 2021-12-15 07:50:30
499996  A-6323243  Source1         2 2021-12-19 16:25:00 2021-12-19 17:40:37
499997  A-3789256  Source1         2 2022-04-13 19:28:29 2022-04-13 21:33:44
```

```
499998  A-7030381  Source1         3 2020-05-15 17:20:56 2020-05-15 17:50:56
499999  A-5438901  Source1         2 2022-04-02 23:23:13 2022-04-03 00:49:48

        Start_Lat   Start_Lng    End_Lat     End_Lng  Distance(mi)  …  \
499995  45.522510 -123.084104  45.520225 -123.084211         0.158  …
499996  26.702570  -80.111169  26.703141  -80.111133         0.040  …
499997  34.561862 -112.259620  34.566822 -112.267150         0.549  …
499998  38.406680  -78.619310  38.406680  -78.619310         0.000  …
499999  35.069358  -85.234410  35.070505  -85.233836         0.086  …

       Turning_Loop Sunrise_Sunset Civil_Twilight Nautical_Twilight  \
499995        False          Night            Day               Day
499996        False            Day            Day               Day
499997        False          Night          Night               Day
499998        False            Day            Day               Day
499999        False          Night          Night             Night

       Astronomical_Twilight  Year Hour Month        Day Is_Weekend
499995                   Day  2021    7    12  Wednesday      False
499996                   Day  2021   16    12     Sunday       True
499997                   Day  2022   19     4  Wednesday      False
499998                   Day  2020   17     5     Friday      False
499999                 Night  2022   23     4   Saturday       True

[5 rows x 51 columns]
```

[ ]:
```
dataset["State"].unique()
```

[ ]:
```
array(['LA', 'VA', 'CA', 'MN', 'MA', 'OR', 'FL', 'TX', 'IN', 'MT', 'AL',
       'AZ', 'NY', 'NC', 'SC', 'KS', 'MI', 'CO', 'GA', 'MD', 'IL', 'PA',
       'NM', 'NV', 'NE', 'NJ', 'TN', 'UT', 'RI', 'MS', 'IA', 'WA', 'MO',
       'OH', 'OK', 'CT', 'DC', 'AR', 'WV', 'KY', 'WI', 'NH', 'WY', 'DE',
       'VT', 'ID', 'ME', 'ND', 'SD'], dtype=object)
```

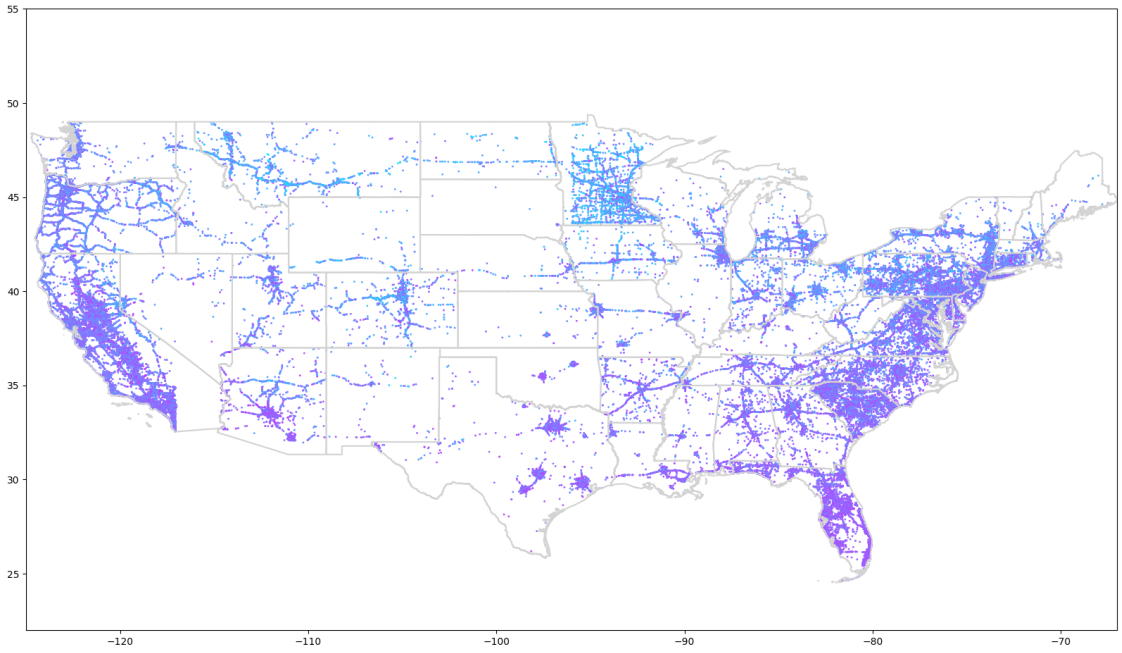
[ ]:
```
# states = gpd.read_file('./dataset/cb_2018_us_state_500k.shp')
states = gpd.read_file('./dataset/map/cb_2018_us_state_500k.shp')
```

[ ]:
```
geometry = [Point(xy) for xy in zip(dataset['Start_Lng'], dataset['Start_Lat'])]
gdf = gpd.GeoDataFrame(dataset, geometry=geometry)
```

[ ]:
```
gdf["Temperature(C)"] = (gdf["Temperature(F)"] - 32) * (5/9)
```

[ ]:
```
fig, ax = plt.subplots(figsize=(20, 20))

# Plot the map of states
states.boundary.plot(ax=ax, color='lightgray')
```

```
# Plot the data points
gdf.dropna().plot(column="Temperature(C)", cmap='cool', markersize=1, ax=ax)

ax.set_xlim([-125,-67])
ax.set_ylim([22,55])

plt.show()
```



[ ]:
```
df_copy=dataset.copy()
```

[ ]:
```
df_copy.shape
```

[ ]:
```
(500000, 51)
```

[ ]:
```
df_copy
```

[ ]:

	ID	Source	Severity	Start_Time	End_Time	\
0	A-2047758	Source2	2	2019-06-12 10:10:56	2019-06-12 10:55:58	
1	A-4694324	Source1	2	2022-12-03 23:37:14	2022-12-04 01:56:53	
2	A-5006183	Source1	2	2022-08-20 13:13:00	2022-08-20 15:22:45	
3	A-4237356	Source1	2	2022-02-21 17:43:04	2022-02-21 19:43:23	
4	A-6690583	Source1	2	2020-12-04 01:46:00	2020-12-04 04:13:09	
…	…	…	…	…	…	
499995	A-6077227	Source1	2	2021-12-15 07:30:00	2021-12-15 07:50:30	
499996	A-6323243	Source1	2	2021-12-19 16:25:00	2021-12-19 17:40:37	
499997	A-3789256	Source1	2	2022-04-13 19:28:29	2022-04-13 21:33:44	

```
499998  A-7030381  Source1         3 2020-05-15 17:20:56 2020-05-15 17:50:56
499999  A-5438901  Source1         2 2022-04-02 23:23:13 2022-04-03 00:49:48

        Start_Lat   Start_Lng    End_Lat     End_Lng  Distance(mi)  …  \
0       30.641211  -91.153481         NaN         NaN         0.000  …
1       38.990562  -77.399070   38.990037  -77.398282         0.056  …
2       34.661189 -120.492822   34.661189 -120.492442         0.022  …
3       43.680592  -92.993317   43.680574  -92.972223         1.054  …
4       35.395484 -118.985176   35.395476 -118.985995         0.046  …
…             …          …          …           …           …  …
499995  45.522510 -123.084104   45.520225 -123.084211         0.158  …
499996  26.702570  -80.111169   26.703141  -80.111133         0.040  …
499997  34.561862 -112.259620   34.566822 -112.267150         0.549  …
499998  38.406680  -78.619310   38.406680  -78.619310         0.000  …
499999  35.069358  -85.234410   35.070505  -85.233836         0.086  …

       Turning_Loop Sunrise_Sunset Civil_Twilight Nautical_Twilight  \
0             False            Day            Day               Day
1             False          Night          Night             Night
2             False            Day            Day               Day
3             False            Day            Day               Day
4             False          Night          Night             Night
…               …              …              …                 …
499995        False          Night            Day               Day
499996        False            Day            Day               Day
499997        False          Night          Night               Day
499998        False            Day            Day               Day
499999        False          Night          Night             Night

       Astronomical_Twilight  Year Hour Month        Day Is_Weekend
0                        Day  2019   10     6  Wednesday      False
1                      Night  2022   23    12   Saturday       True
2                        Day  2022   13     8   Saturday       True
3                        Day  2022   17     2     Monday      False
4                      Night  2020    1    12     Friday      False
…                      …   …  …   …       …          …
499995                   Day  2021    7    12  Wednesday      False
499996                   Day  2021   16    12     Sunday       True
499997                   Day  2022   19     4  Wednesday      False
499998                   Day  2020   17     5     Friday      False
499999                 Night  2022   23     4   Saturday       True

[500000 rows x 51 columns]
```

[ ]:
```
import random
i=0
result_dict = {}
```

```
for year in [2021,2022]:
    result_dict[year] = list(df_copy[df_copy["Year"] == year].index)

for year in [2021,2022]:
    length_df=len(df_copy)
    i+=1
    for j in range(math.floor(length_df*0.02)):
        choice=random.choice(result_dict[year])
        df_copy.loc[choice, 'Sample']='Sample'+str(i)
        result_dict[year].remove(choice)
```

[ ]:
```
contingency_table = pd.crosstab(df_copy["Sample"],df_copy["Severity"])

# Calculate the total sum of the contingency table
totalsum = contingency_table.sum().sum()

# Get row sums (Sum of Samples)
row_sums = contingency_table.sum(axis=1)

# Get column sums (Sum of Severities)
col_sums = contingency_table.sum(axis=0)

# Generate expected values matrix
expected_values = np.outer(row_sums, col_sums) / totalsum

# Get observed values directly from the contingency table
observed_values = contingency_table.values

# Calculate the chi-square statistic
chi_squared_stat = ((observed_values-expected_values)**2/expected_values).sum().
 ↪sum()

# Calculate the degrees of freedom
df_ = (len(row_sums)-1)*(len(col_sums)-1)

# Get p-value from chi-square distribution
p_value = 1 - chi2.cdf(chi_squared_stat, df_)

print("Chi-squared Statistic :", chi_squared_stat)
print("Degrees of Freedom :", df_)
print("P-Value :", p_value)
```

```
Chi-squared Statistic : 426.99363961817687
Degrees of Freedom : 3
P-Value : 0.0
```

[ ]:
```
i=0
result_dict = {}
for Timezone in ['US/Central', 'US/Pacific', 'US/Eastern', 'US/Mountain']:
    result_dict[Timezone] = list(df_copy[df_copy["Timezone"] == Timezone].index)

for Timezone in ['US/Central', 'US/Pacific', 'US/Eastern', 'US/Mountain']:
    length_df=len(df_copy)
    i+=1
    for j in range(math.floor(length_df*0.001)):
        choice=random.choice(result_dict[Timezone])
        df_copy.loc[choice, 'Sample']='Sample'+str(i)
        result_dict[Timezone].remove(choice)
```

[ ]:
```
df_copy.columns
```

[ ]:
```
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', 'Year', 'Hour', 'Month', 'Day', 'Is_Weekend',
       'Sample'],
      dtype='object')
```

[ ]:
```
pd.crosstab(df_copy["Sample"],df_copy["Severity"])
```

[ ]:
```
Severity    1     2     3    4
Sample
Sample1     3  9182  1082  198
Sample2   199  9512   499  251
Sample3     5   390    85   20
Sample4     9   383    82   26
```

[ ]:
```
chi2_contingency(pd.crosstab(df_copy["Sample"],df_copy["Severity"]))
```

[ ]:
```
Chi2ContingencyResult(statistic=548.2673442595317,
pvalue=2.6184418541667544e-112, dof=9, expected_freq=array([[1.03094044e+02,
9.29135068e+03, 8.34298094e+02, 2.36257183e+02],
       [1.03054638e+02, 9.28779928e+03, 8.33979203e+02, 2.36166880e+02],
       [4.92565903e+00, 4.43925021e+02, 3.98613518e+01, 1.12879686e+01],
       [4.92565903e+00, 4.43925021e+02, 3.98613518e+01, 1.12879686e+01]]))
```

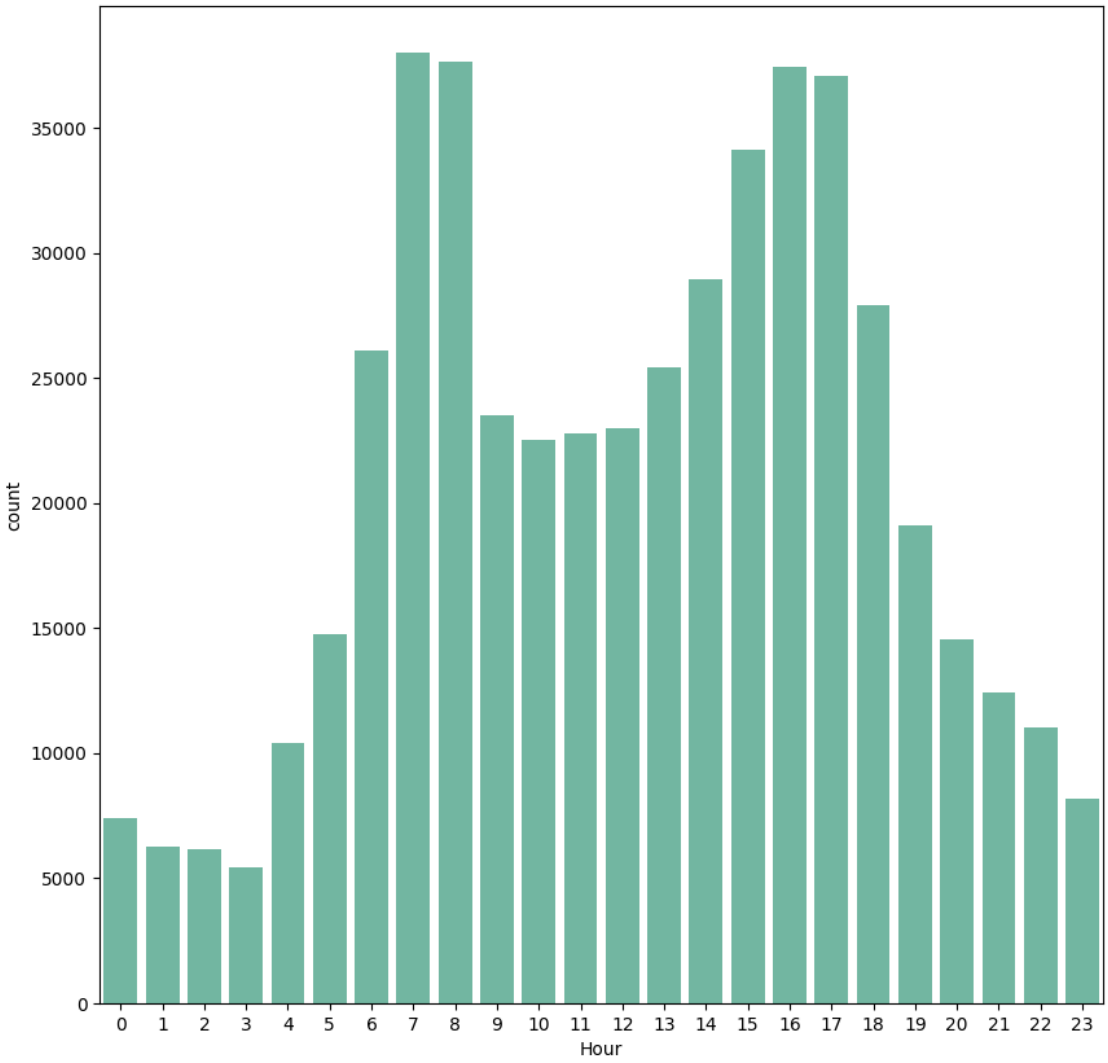
[ ]:
```
chi2_contingency(pd.crosstab(df_copy["Sample"],df_copy["Severity"]).iloc[0:2,])
```

[ ]:
```
Chi2ContingencyResult(statistic=417.2425464254256, pvalue=4.074776032658768e-90,
dof=3, expected_freq=array([[ 101.01930613, 9348.78667686,  790.65110389,
224.54291312],
       [ 100.98069387, 9345.21332314,  790.34889611,  224.45708688]]))
```

[ ]:
```
chi2_contingency(pd.crosstab(df_copy["Sample"],df_copy["Severity"]).
↪iloc[[2,3],])
```

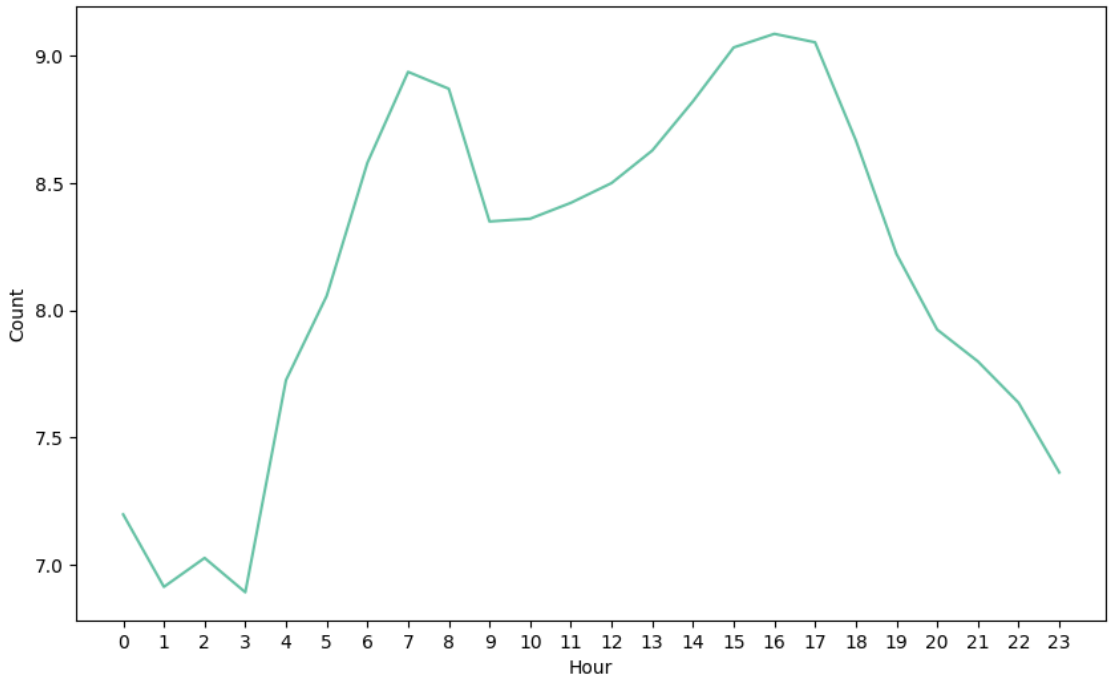
[ ]:
```
Chi2ContingencyResult(statistic=2.04274744605748, pvalue=0.5635818839038944,
dof=3, expected_freq=array([[  7. , 386.5,  83.5,  23. ],
       [  7. , 386.5,  83.5,  23. ]]))
```

[ ]:
```
plt.figure(figsize=(10,10))
sns.countplot(data=df_copy,x="Hour")
plt.show()
```



[ ]:
```
desired_dates = ['2021-03', '2021-04', '2021-06', '2021-07', '2021-08',␣
 ↪'2021-09', '2021-10', '2021-11', '2022-01', '2022-02', '2022-03', '2022-04']

df_copy['Year_Month'] = df_copy['Year'].astype(str) + '-' + df_copy['Month'].
 ↪astype(str).str.zfill(2)

filtered_df = df_copy[df_copy['Year_Month'].isin(desired_dates)]

filtered_df = filtered_df.drop(columns=['Year_Month'])
```

[ ]:
```
# comparative table accidents showing the range of hours within day
hour_counts = filtered_df["Hour"].value_counts().reset_index()
hour_counts.columns = ["Hour", "Count"]
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
plt.xticks(filtered_df["Hour"].unique())

sns.lineplot(data=hour_counts, x="Hour", y=hour_counts["Count"].apply(lambda x:
 ↪math.log(x)))
plt.show()
```

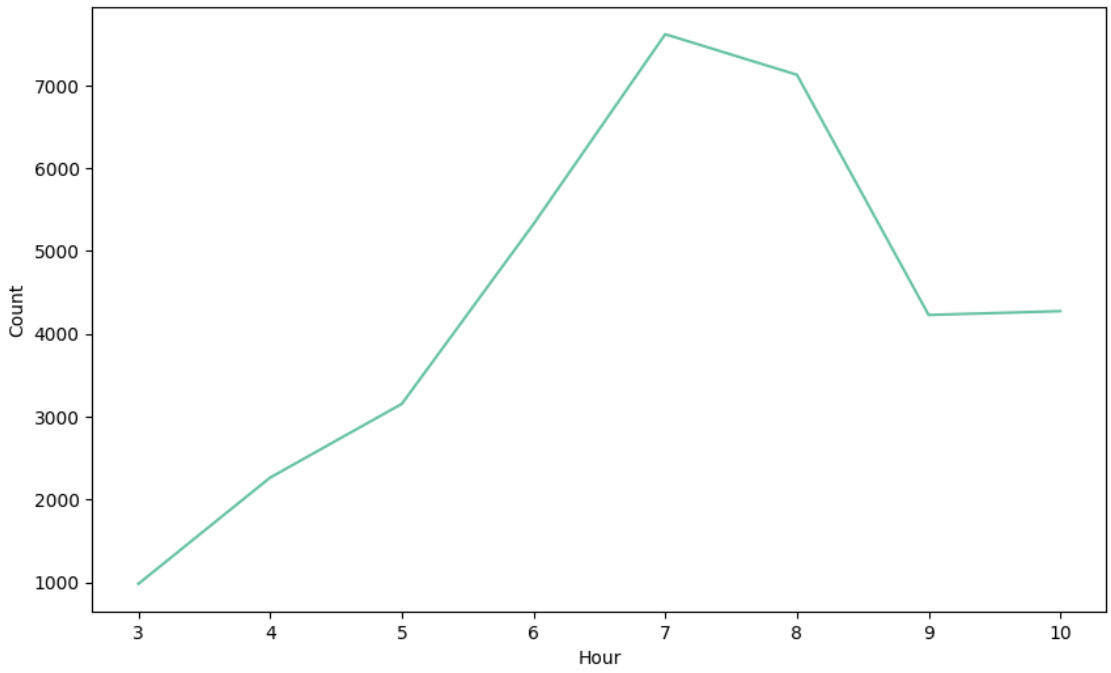


[ ]:
```
hour_counts_day=hour_counts[(hour_counts["Hour"] >= 3) & (hour_counts["Hour"]␣
 ↪<= 10)]
hour_counts_day
```

[ ]:

	Hour	Count
3	7	7617
4	8	7127
8	6	5326
11	10	4274
12	9	4228
14	5	3155
17	4	2265
23	3	983

[ ]:
```
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
plt.xticks(hour_counts_day["Hour"].unique())

sns.lineplot(data=hour_counts_day, x="Hour", y=hour_counts_day["Count"])
plt.show()
```



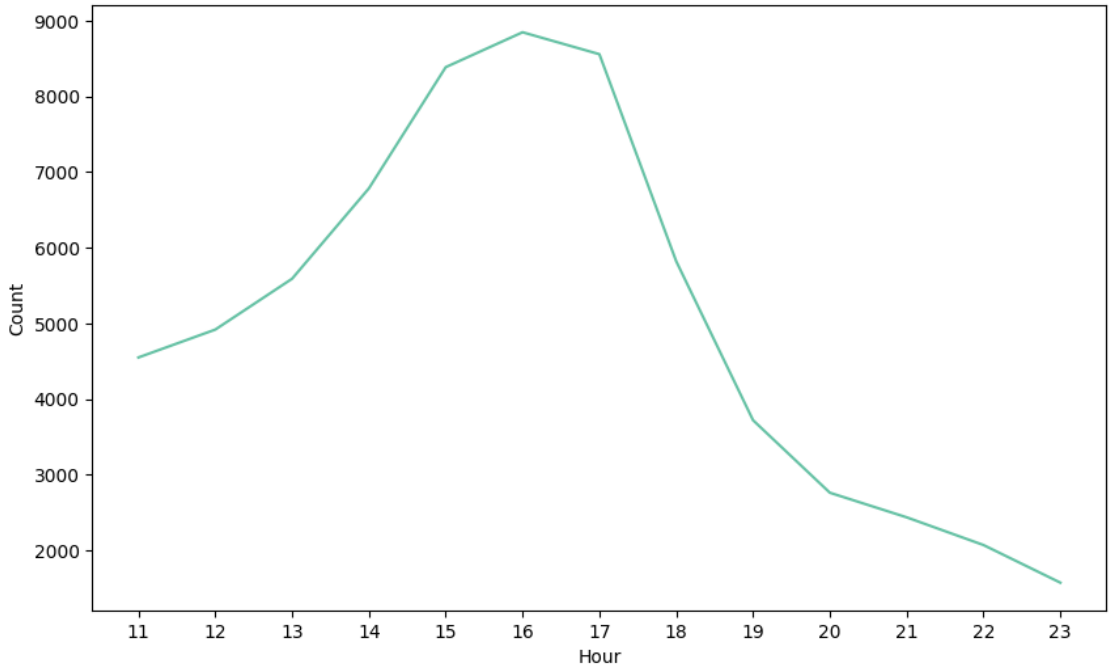
[ ]:
```
hour_counts_night=hour_counts[(hour_counts["Hour"] < 0) | (hour_counts["Hour"]␣
↪> 10)]
hour_counts_night
```

[ ]:

	Hour	Count
0	16	8846
1	17	8556
2	15	8385
5	14	6784

```
6      18    5821
7      13    5590
9      12    4918
10     11    4550
13     19    3721
15     20    2763
16     21    2438
18     22    2072
19     23    1575
```

[ ]:
```
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
plt.xticks(hour_counts_night["Hour"].unique())

sns.lineplot(data=hour_counts_night, x="Hour", y=hour_counts_night["Count"])
plt.show()
```



[ ]:
```
confidence = 0.95
sample_mean = hour_counts_night['Count'].mean()
sample_size = hour_counts_night.shape[0]
standard_error = stats.sem(hour_counts_night['Count'])

ci = stats.t.interval(confidence, df=sample_size-1, loc=sample_mean,␣
 ↪scale=standard_error)

print("Confidence Interval:")
```

```
print(f"Lower Bound: {ci[0]}")
print(f"Upper Bound: {ci[1]}")
result_interval = hour_counts_night[(hour_counts_night["Count"] >= ci[0]) &␣
 ↪(hour_counts_night["Count"] <= ci[1])]
print(f"We are 95% sure that accidents falls in [{result_interval['Hour'].
 ↪min()},{result_interval['Hour'].max()}]")
```

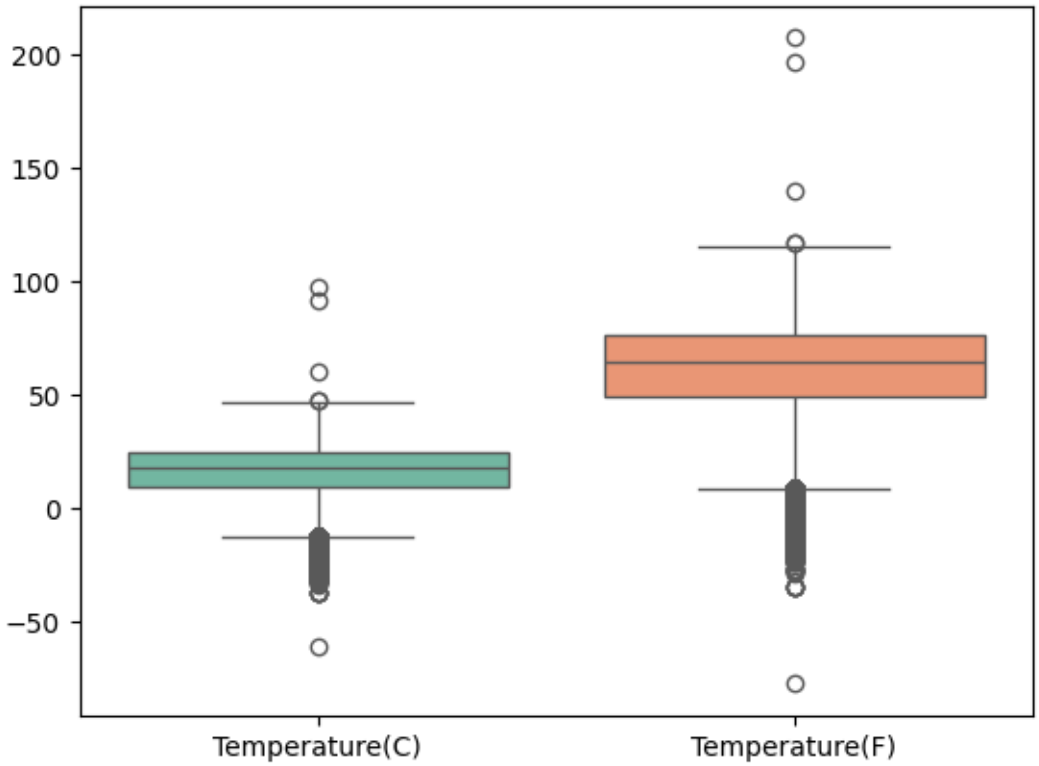
```
Confidence Interval:
Lower Bound: 3550.256685449029
Upper Bound: 6606.512545320202
We are 95% sure that accidents falls in [11,19]
```

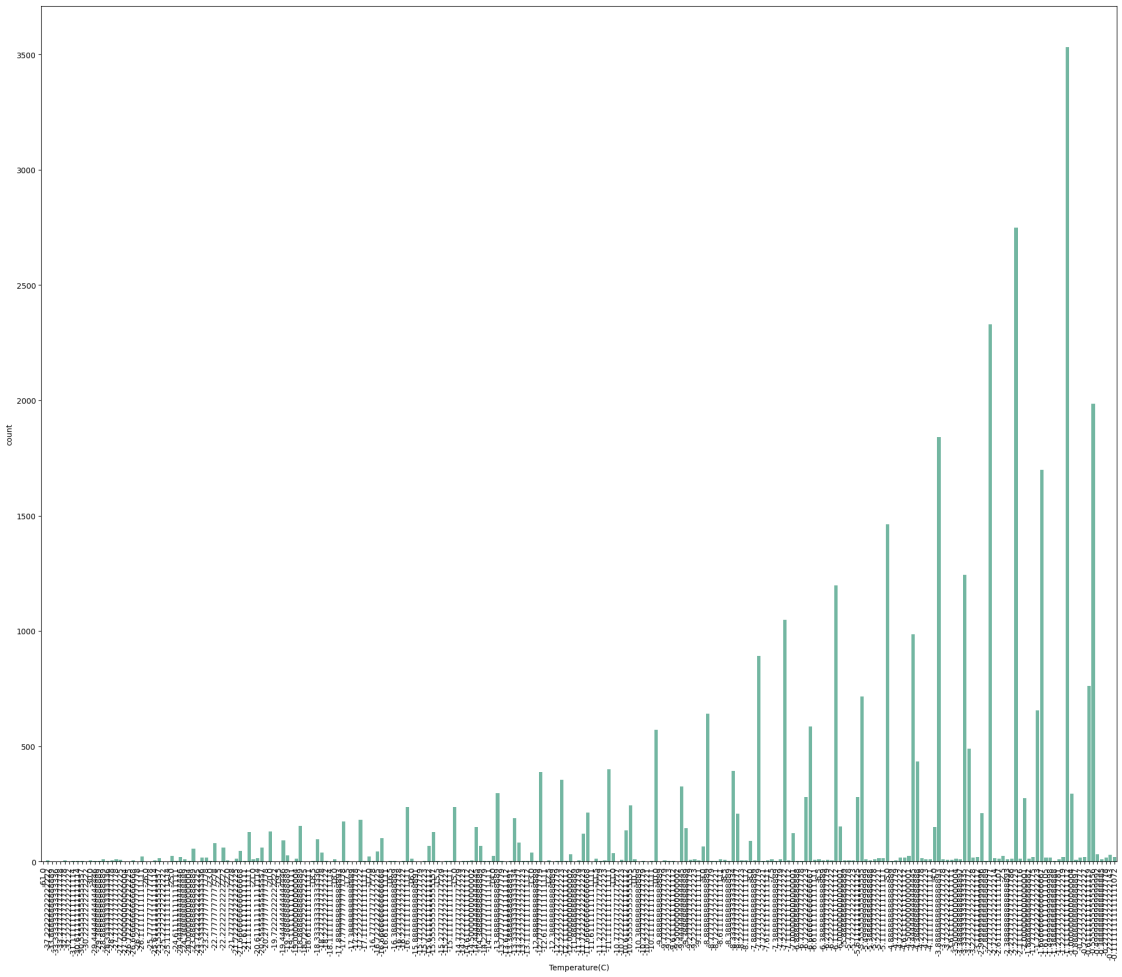
[ ]:
```
df_copy["Temperature(C)"] = (df_copy["Temperature(F)"] - 32) * (5/9)
```

[ ]:
```
sns.boxplot(df_copy[["Temperature(C)", "Temperature(F)"]])
# plt.show()
```

[ ]:
```
<Axes: >
```

[ ]:
```
plt.figure(figsize=(25,20))
sns.countplot(df_copy[df_copy["Temperature(C)"]<0],x="Temperature(C)")
plt.xticks(rotation=90)  # Rotate the x-axis labels by 45 degrees
# plt.tight_layout()
plt.show()
```





[ ]:
```
df_copy=df_copy[(df_copy["Temperature(C)"]<=42) &␣
↪(df_copy["Temperature(C)"]>=-10)]
```

[ ]:
```
temp_counts = df_copy["Temperature(C)"].value_counts().reset_index()
temp_counts.columns = ["Temperature(C)", "Count"]
temp_counts
```

[ ]:

	Temperature(C)	Count
0	25.000000	11090
1	22.777778	11076
2	20.000000	10684
3	22.222222	10217
4	23.888889	10144
..	…	…
541	39.111111	1

```
542       35.388889      1
543       40.500000      1
544       38.722222      1
545       36.611111      1

[546 rows x 2 columns]
```

sns.scatterplot(temp_counts,x="Temperature(C)",y=temp_counts["Count"].apply(lambda x:math.log(x)))

[ ]:
```
sns.scatterplot(temp_counts,x="Temperature(C)",y=temp_counts["Count"].
 ↪apply(lambda x:math.log(x)))
# plt.show
```

[ ]:
```
<Axes: xlabel='Temperature(C)', ylabel='Count'>
```

[ ]:
```
X = temp_counts['Temperature(C)'].values.reshape(-1, 1)
y = temp_counts['Count'].values


# Degree of the polynomial regression
degree = 3  # You can change this to the degree you want (e.g., 3 for cubic␣
 ↪regression)

# Transforming the features to include polynomial terms
poly = PolynomialFeatures(degree=degree)
X_poly = poly.fit_transform(X)

# Fitting the polynomial regression model
model = LinearRegression()
model.fit(X_poly, y)

# Getting the coefficients and intercept of the model
coefficients = model.coef_
intercept = model.intercept_
```

[ ]:
```
X_poly_with_constant = sm.add_constant(X_poly)
model_stats = sm.OLS(y, X_poly_with_constant).fit()
print(model_stats.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.067
Model:                            OLS   Adj. R-squared:                  0.061
Method:                 Least Squares   F-statistic:                     12.89
Date:                Sun, 26 Nov 2023   Prob (F-statistic):           3.82e-08
Time:                        14:05:30   Log-Likelihood:                -4933.1
No. Observations:                 546   AIC:                             9874.
```

```
Df Residuals:                      542   BIC:                             9891.
Df Model:                            3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        590.5922    149.604      3.948      0.000     296.717     884.467
x1            67.3609     18.798      3.583      0.000      30.434     104.288
x2             0.1895      1.653      0.115      0.909      -3.058       3.437
x3            -0.0609      0.034     -1.777      0.076      -0.128       0.006
==============================================================================
Omnibus:                      315.607   Durbin-Watson:                   0.068
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1822.904
Skew:                           2.647   Prob(JB):                         0.00
Kurtosis:                      10.219   Cond. No.                     3.93e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.93e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[ ]:
```
df_copy["Duration"]=df_copy["End_Time"] - df_copy["Start_Time"]
df_copy["Duration"]=df_copy["Duration"].apply(lambda x: x.total_seconds() / 60)
df_copy["Duration"]
```

[ ]:
```
0          45.033333
1         139.650000
2         129.750000
3         120.316667
4         147.150000
              …
499995     20.500000
499996     75.616667
499997    125.250000
499998     30.000000
499999     86.583333
Name: Duration, Length: 483874, dtype: float64
```

[ ]:
```
df_copy=df_copy.loc[df_copy["Duration"] <= 400]
```

[ ]:
```
sns.displot(df_copy["Duration"])
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x269407eb9d0>
```

[ ]:
```
from sklearn.utils import resample
```

```
data = df_copy["Duration"].dropna()

# Perform bootstrap resampling
n_iterations = 5000
bootstrap_means = []
for _ in range(n_iterations):
    resampled_data = resample(data)
    mean = np.mean(resampled_data)
    bootstrap_means.append(mean)

# Calculate the confidence interval using percentiles
confidence_interval = np.percentile(bootstrap_means, [2.5, 97.5])
print(f"The confidence interval using bootstrap resampling is:␣
 ↪{confidence_interval}")
```

```
The confidence interval using bootstrap resampling is: [94.58998886 95.08481176]
```

[ ]:
```
df_copy["Duration"].describe()
```

[ ]:
```
count    470501.000000
mean         94.839852
std          85.185572
min           2.500000
25%          30.000000
50%          72.250000
75%         121.650000
max         400.000000
Name: Duration, dtype: float64
```

[ ]:
```
#searching about duplicated values
df_copy.duplicated().sum()
```

[ ]:
```
0
```

[ ]:
```
df_copy.isnull().sum()
```

[ ]:
```
ID                          0
Source                      0
Severity                    0
Start_Time                  0
End_Time                    0
Start_Lat                   0
Start_Lng                   0
End_Lat                214387
End_Lng                214387
Distance(mi)                0
Description                 1
Street                    650
```

```
City                       17
County                      0
State                       0
Zipcode                     0
Country                     0
Timezone                    0
Airport_Code                0
Weather_Timestamp           0
Temperature(F)              0
Wind_Chill(F)          117884
Humidity(%)               633
Pressure(in)              481
Visibility(mi)           2411
Wind_Direction           2637
Wind_Speed(mph)         27506
Precipitation(in)      131868
Weather_Condition        2314
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           1232
Civil_Twilight           1232
Nautical_Twilight        1232
Astronomical_Twilight    1232
Year                        0
Hour                        0
Month                       0
Day                         0
Is_Weekend                  0
Sample                 450231
Year_Month                  0
Temperature(C)              0
Duration                    0
dtype: int64
```

[ ]:
```
#filling missing values with interpolate method
# limit is Maximum number of consecutive NaNs to fill. Must be greater than 0.
```

```
df_copy.fillna(method='ffill', limit=5, inplace=True)
df_copy.fillna(method='bfill', limit=5, inplace=True)
```

[ ]:
```
df_copy.isnull().sum()
```

[ ]:
```
ID                         0
Source                     0
Severity                   0
Start_Time                 0
End_Time                   0
Start_Lat                  0
Start_Lng                  0
End_Lat                   88
End_Lng                   88
Distance(mi)               0
Description                0
Street                     0
City                       0
County                     0
State                      0
Zipcode                    0
Country                    0
Timezone                   0
Airport_Code               0
Weather_Timestamp          0
Temperature(F)             0
Wind_Chill(F)              0
Humidity(%)                0
Pressure(in)               0
Visibility(mi)             0
Wind_Direction             0
Wind_Speed(mph)            0
Precipitation(in)          0
Weather_Condition          0
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                0
Civil_Twilight                0
Nautical_Twilight             0
Astronomical_Twilight         0
Year                          0
Hour                          0
Month                         0
Day                           0
Is_Weekend                    0
Sample                   289470
Year_Month                    0
Temperature(C)                0
Duration                      0
dtype: int64
```

[ ]:
```
df_copy.drop(columns=['End_Lat', 'End_Lng'],inplace= True)
df_copy.drop(columns=['Sample'],inplace= True)
# df_copy.
  ↪dropna(subset=['Wind_Chill(F)','Wind_Speed(mph)','Precipitation(in)'],inplace␣
  ↪= True)
```

[ ]:
```
df_copy.isnull().sum()
```

[ ]:
```
ID                    0
Source                0
Severity              0
Start_Time            0
End_Time              0
Start_Lat             0
Start_Lng             0
Distance(mi)          0
Description           0
Street                0
City                  0
County                0
State                 0
Zipcode               0
Country               0
Timezone              0
Airport_Code          0
Weather_Timestamp     0
Temperature(F)        0
Wind_Chill(F)         0
Humidity(%)           0
Pressure(in)          0
Visibility(mi)        0
Wind_Direction        0
```

```
Wind_Speed(mph)          0
Precipitation(in)        0
Weather_Condition        0
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           0
Civil_Twilight           0
Nautical_Twilight        0
Astronomical_Twilight    0
Year                     0
Hour                     0
Month                    0
Day                      0
Is_Weekend               0
Year_Month               0
Temperature(C)           0
Duration                 0
dtype: int64
```

[ ]:
```
df_copy.describe()
```

[ ]:
```
            Severity      Start_Lat      Start_Lng   Distance(mi)  \
count  470501.000000  470501.000000  470501.000000  470501.000000
mean        2.215356      36.163513     -94.710503       0.528834
std         0.487005       5.021527      17.436777       1.668964
min         1.000000      24.562117    -124.497420       0.000000
25%         2.000000      33.412563    -117.248543       0.000000
50%         2.000000      35.788753     -87.644279       0.019000
75%         2.000000      40.038849     -80.367728       0.430000
max         4.000000      48.991585     -67.484130     193.479996

       Temperature(F)  Wind_Chill(F)    Humidity(%)   Pressure(in)  \
count   470501.000000  470501.000000  470501.000000  470501.000000
mean        62.283057      59.110685      64.770538      29.546435
std         18.085153      21.070755      22.893141       0.986629
min         14.000000      -9.300000       1.000000       2.990000
```

```
25%         50.000000      43.000000      48.000000      29.380000
50%         64.000000      63.000000      67.000000      29.860000
75%         76.000000      75.000000      84.000000      30.030000
max        107.600000     107.000000     100.000000      38.440000

       Visibility(mi)  Wind_Speed(mph)  Precipitation(in)           Year  \
count   470501.000000    470501.000000      470501.000000  470501.000000
mean         9.105735         7.667757           0.008708    2019.871205
std          2.685973         5.396414           0.112725       1.928460
min          0.000000         0.000000           0.000000    2016.000000
25%         10.000000         4.600000           0.000000    2018.000000
50%         10.000000         7.000000           0.000000    2020.000000
75%         10.000000        10.400000           0.000000    2022.000000
max        130.000000       822.800000          10.130000    2023.000000

                Hour          Month  Temperature(C)       Duration
count  470501.000000  470501.000000   470501.000000  470501.000000
mean       12.387833       6.715799       16.823921      94.839852
std         5.447358       3.610765       10.047307      85.185572
min         0.000000       1.000000      -10.000000       2.500000
25%         8.000000       4.000000       10.000000      30.000000
50%        13.000000       7.000000       17.777778      72.250000
75%        17.000000      10.000000       24.444444     121.650000
max        23.000000      12.000000       42.000000     400.000000
```

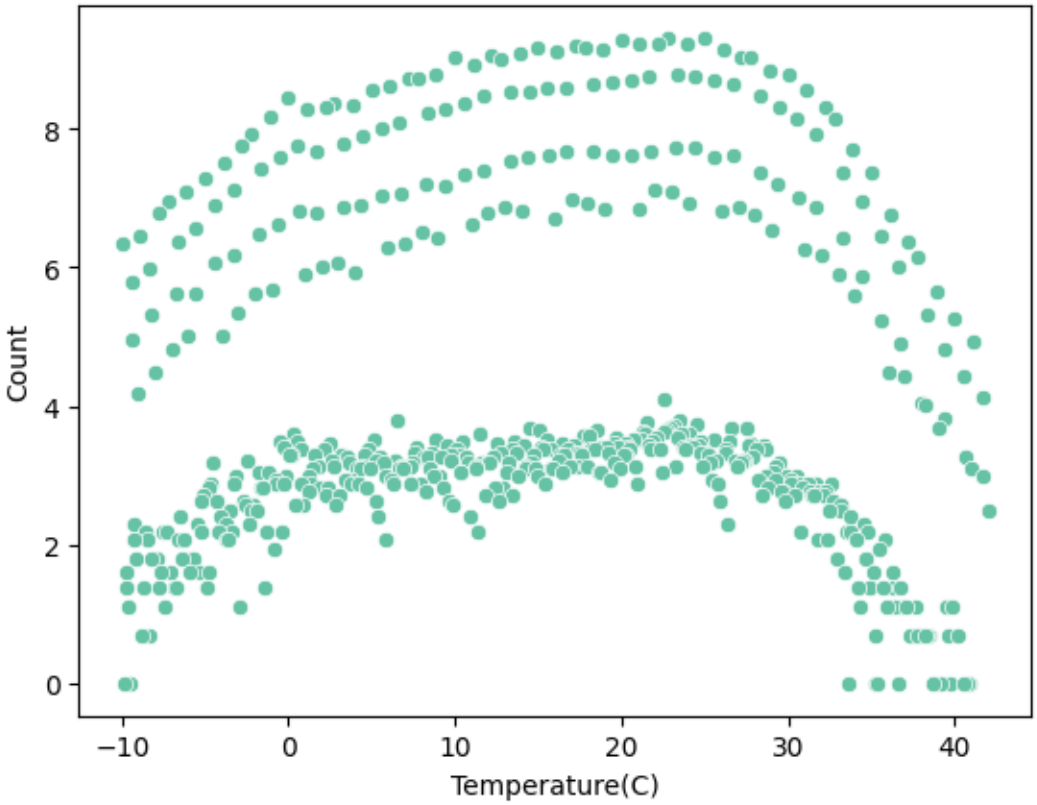
[ ]:
```
OutliersColumns = ["Start_Lat","Start_Lng"]
```

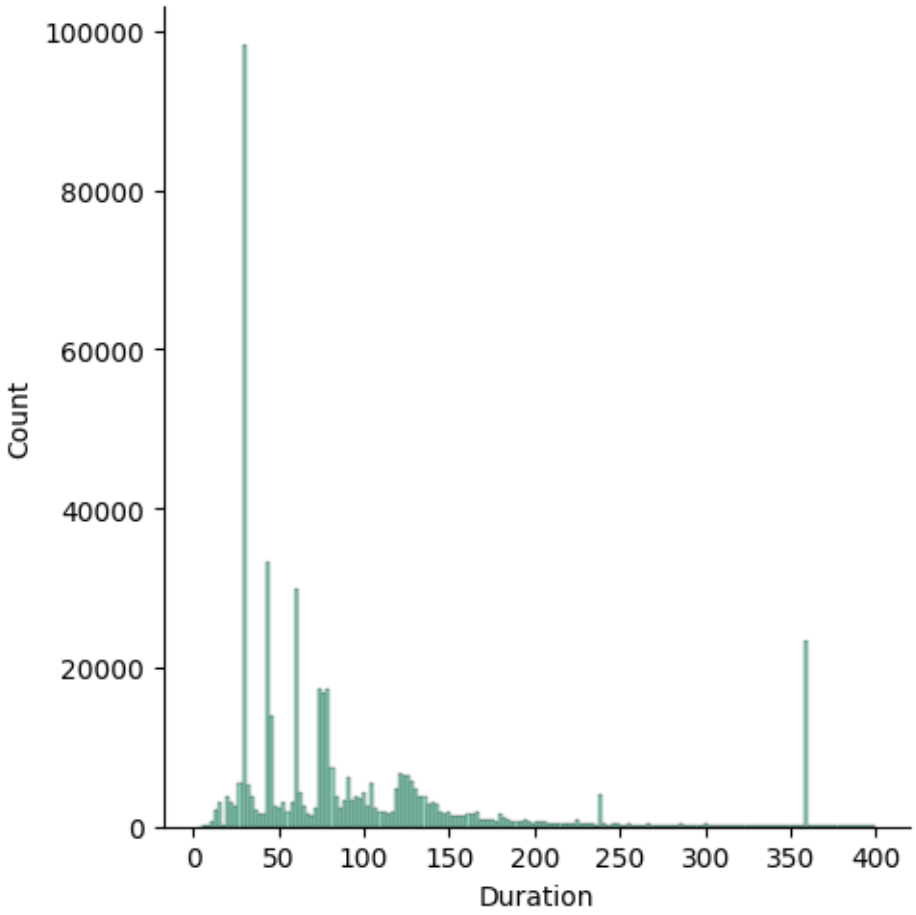
[ ]:
```
# for i in OutliersColumns:
#   # IQR
#   # Calculate the upper and lower limits
#   Q1 = df_copy[i].quantile(0.25)
#   Q3 = df_copy[i].quantile(0.75)
#   IQR = Q3 - Q1
#   lower = Q1 - 1.5*IQR
#   upper = Q3 + 1.5*IQR

#   # Create arrays of Boolean values indicating the outlier rows
#   upper_array = np.where(df_copy[i]>=upper)[0]
#   lower_array = np.where(df_copy[i]<=lower)[0]

#   # Removing the outliers
#   df_copy = df_copy[~df_copy.index.isin(upper_array)]
#   df_copy = df_copy[~df_copy.index.isin(lower_array)]
```

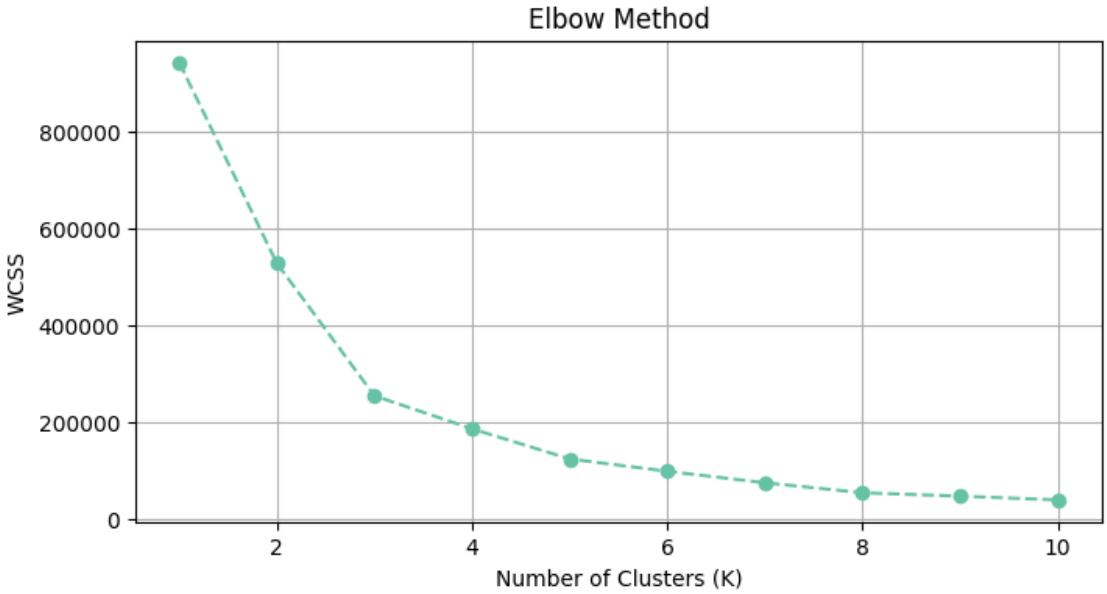
[ ]:
```
from sklearn.preprocessing import StandardScaler
X = df_copy[['Start_Lat', 'Start_Lng']]
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[ ]:
```
from sklearn.cluster import KMeans
wcss = []
max_clusters = 10
for i in range(1, max_clusters + 1):
    kmeans = KMeans(n_clusters=i, random_state=0)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(8, 4))
plt.plot(range(1, max_clusters + 1), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.grid()
plt.show()
```

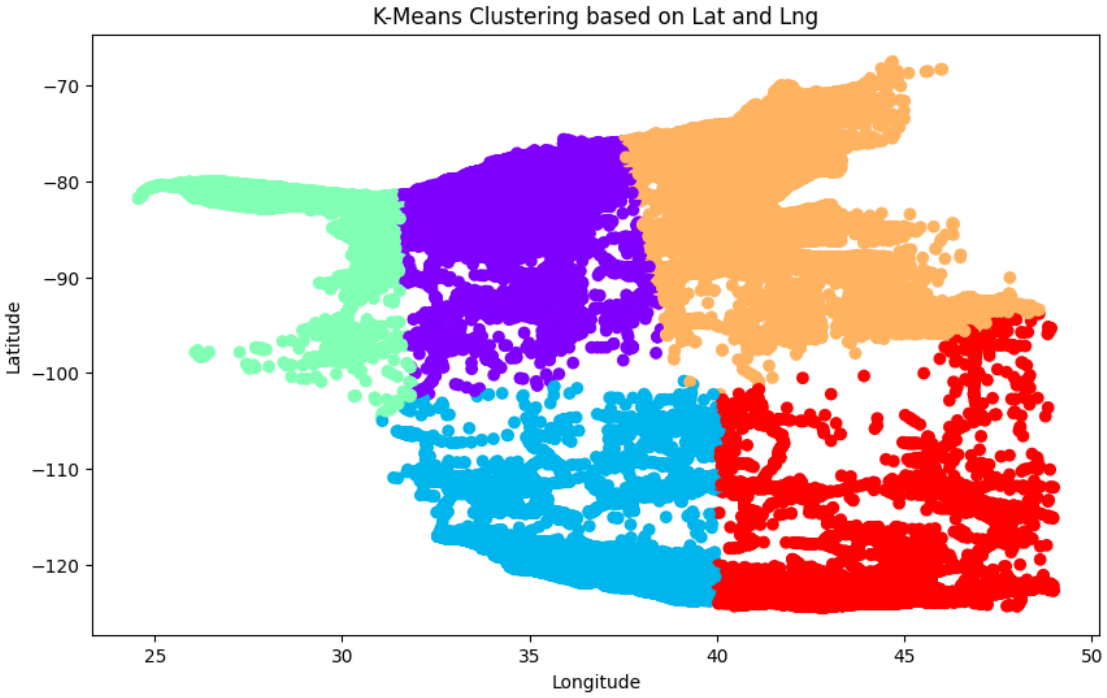



100000
80000
60000
40000
20000
0
Count
0
50
100
150
200
250
300
350
400
Duration


Elbow Method
800000
600000
400000
200000
0
WCSS
2
4
6
8
10
Number of Clusters (K)

[ ]:
```
k = 5
kmeans = KMeans(n_clusters=k, random_state=0)
clusters = kmeans.fit_predict(X_scaled)

df_copy['cluster_LatLng'] = clusters
```

[ ]:
```
plt.figure(figsize=(10, 6))
plt.scatter(df_copy['Start_Lat'], df_copy['Start_Lng'],␣
 ↪c=df_copy['cluster_LatLng'], cmap='rainbow')
plt.title('K-Means Clustering based on Lat and Lng')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



[ ]:
```
df_copy['cluster_LatLng'].unique()
```

[ ]:
```
array([2, 3, 1, 4, 0])
```

[ ]:
```
df_copy.shape
```

[ ]:
```
(470501, 53)
```

[ ]:
```
df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 470501 entries, 0 to 499999
Data columns (total 53 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   ID                     470501 non-null  object
 1   Source                 470501 non-null  object
 2   Severity               470501 non-null  int64
 3   Start_Time             470501 non-null  datetime64[ns]
 4   End_Time               470501 non-null  datetime64[ns]
 5   Start_Lat              470501 non-null  float64
 6   Start_Lng              470501 non-null  float64
 7   Distance(mi)           470501 non-null  float64
 8   Description            470501 non-null  object
 9   Street                 470501 non-null  object
 10  City                   470501 non-null  object
 11  County                 470501 non-null  object
 12  State                  470501 non-null  object
 13  Zipcode                470501 non-null  object
 14  Country                470501 non-null  object
 15  Timezone               470501 non-null  object
 16  Airport_Code           470501 non-null  object
 17  Weather_Timestamp      470501 non-null  object
 18  Temperature(F)         470501 non-null  float64
 19  Wind_Chill(F)          470501 non-null  float64
 20  Humidity(%)            470501 non-null  float64
 21  Pressure(in)           470501 non-null  float64
 22  Visibility(mi)         470501 non-null  float64
 23  Wind_Direction         470501 non-null  object
 24  Wind_Speed(mph)        470501 non-null  float64
 25  Precipitation(in)      470501 non-null  float64
 26  Weather_Condition      470501 non-null  object
 27  Amenity                470501 non-null  bool
 28  Bump                   470501 non-null  bool
 29  Crossing               470501 non-null  bool
 30  Give_Way               470501 non-null  bool
 31  Junction               470501 non-null  bool
 32  No_Exit                470501 non-null  bool
 33  Railway                470501 non-null  bool
 34  Roundabout             470501 non-null  bool
 35  Station                470501 non-null  bool
 36  Stop                   470501 non-null  bool
 37  Traffic_Calming        470501 non-null  bool
 38  Traffic_Signal         470501 non-null  bool
 39  Turning_Loop           470501 non-null  bool
 40  Sunrise_Sunset         470501 non-null  object
 41  Civil_Twilight         470501 non-null  object
 42  Nautical_Twilight      470501 non-null  object
```

```
 43  Astronomical_Twilight  470501 non-null  object
 44  Year                   470501 non-null  int64
 45  Hour                   470501 non-null  int64
 46  Month                  470501 non-null  int64
 47  Day                    470501 non-null  object
 48  Is_Weekend             470501 non-null  bool
 49  Year_Month             470501 non-null  object
 50  Temperature(C)         470501 non-null  float64
 51  Duration               470501 non-null  float64
 52  cluster_LatLng         470501 non-null  int32
dtypes: bool(14), datetime64[ns](2), float64(12), int32(1), int64(4), object(20)
memory usage: 148.1+ MB
```

[ ]:
```
df_numerical = df_copy.select_dtypes(include=['int64','float64'])
```

[ ]:
```
plt.figure(figsize=(19, 8))
sns.set(style="white")
mask = np.triu(df_numerical.corr())
sns.heatmap(data=df_numerical.corr(), annot=True, fmt=".2f", cmap='coolwarm',␣
 ↪mask=mask)
# plt.show
```

[ ]:
```
<Axes: >
```

[ ]:
```
print("Number of rows:", len(df_copy.index))
df_copy.drop_duplicates(inplace=True)
print("Number of rows after dropping duplicates:", len(df_copy.index))
```

```
Number of rows: 470501
Number of rows after dropping duplicates: 470501
```

[ ]:
```
state_counts = df_copy["State"].value_counts()
fig = go.Figure(data=go.Choropleth(locations=state_counts.index, z=state_counts.
 ↪values.astype(float), locationmode="USA-states", colorscale="turbo"))
fig.update_layout(title_text="Number of Accidents by State", geo_scope="usa")
# state_counts
fig.show()
```

[ ]:
```
from pprint import pprint
def sanity_check(df_copy):
    pprint('-'*70)
    pprint('No. of Rows: {0[0]}        No. of Columns : {0[1]}'.format(df_copy.
 ↪shape))
    pprint('-'*70)
    data_profile = pd.DataFrame(df_copy.dtypes.reset_index()).rename(columns =␣
 ↪{'index':'Attribute', 0:'DataType'}).set_index('Attribute')
```

```
    data_profile = pd.concat([data_profile,df_copy.isnull().sum()], axis=1).
 ↪rename(columns = {0 : 'Missing Values'})
    data_profile = pd.concat([data_profile,(df_copy.isnull().mean()*100).
 ↪round(2)], axis=1).rename(columns = {0 : 'Missing %'})
    data_profile = pd.concat([data_profile,df_copy.nunique()], axis=1).
 ↪rename(columns = {0 : 'Unique Values'})

    pprint(data_profile)
    pprint('-'*70)

sanity_check(df_copy)
```

```
'----------------------------------------------------------------------'
'No. of Rows: 470501        No. of Columns : 53'
'----------------------------------------------------------------------'
                            DataType  Missing Values  Missing %  \
ID                            object               0        0.0
Source                        object               0        0.0
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
Distance(mi)                  float64               0        0.0
Description                   object               0        0.0
Street                        object               0        0.0
City                          object               0        0.0
County                        object               0        0.0
State                         object               0        0.0
Zipcode                       object               0        0.0
Country                       object               0        0.0
Timezone                      object               0        0.0
Airport_Code                  object               0        0.0
Weather_Timestamp             object               0        0.0
Temperature(F)                float64               0        0.0
Wind_Chill(F)                 float64               0        0.0
Humidity(%)                   float64               0        0.0
Pressure(in)                  float64               0        0.0
Visibility(mi)                float64               0        0.0
Wind_Direction                object               0        0.0
Wind_Speed(mph)               float64               0        0.0
Precipitation(in)             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
Civil_Twilight	object	0	0.0
Nautical_Twilight	object	0	0.0
Astronomical_Twilight	object	0	0.0
Year	int64	0	0.0
Hour	int64	0	0.0
Month	int64	0	0.0
Day	object	0	0.0
Is_Weekend	bool	0	0.0
Year_Month	object	0	0.0
Temperature(C)	float64	0	0.0
Duration	float64	0	0.0
cluster_LatLng	int32	0	0.0

	Unique Values
ID	470501
Source	3
Severity	4
Start_Time	455567
End_Time	463377
Start_Lat	349628
Start_Lng	350675
Distance(mi)	9996
Description	387700
Street	85225
City	9146
County	1573
State	49
Zipcode	122442
Country	1
Timezone	4
Airport_Code	1839
Weather_Timestamp	248147
Temperature(F)	546
Wind_Chill(F)	535
Humidity(%)	100
Pressure(in)	963
Visibility(mi)	63
Wind_Direction	24

```
Wind_Speed(mph)                  93
Precipitation(in)               174
Weather_Condition               106
Amenity                           2
Bump                              2
Crossing                          2
Give_Way                          2
Junction                          2
No_Exit                           2
Railway                           2
Roundabout                        2
Station                           2
Stop                              2
Traffic_Calming                   2
Traffic_Signal                    2
Turning_Loop                      1
Sunrise_Sunset                    2
Civil_Twilight                    2
Nautical_Twilight                 2
Astronomical_Twilight             2
Year                              8
Hour                             24
Month                            12
Day                               7
Is_Weekend                        2
Year_Month                       86
Temperature(C)                  546
Duration                      18679
cluster_LatLng                    5
'-----------------------------------------------------------------------'
```

[ ]:
```
states = pd.DataFrame(state_counts).reset_index()
states.rename(columns={'index': 'state_code', 'State': 'cases'}, inplace=True)
states = states.sort_values(by='cases', ascending=False)
# states
```

[ ]:
```
# Import
import pandas as pd

# convert the states series into a DataFrame
states = pd.DataFrame(state_counts).reset_index()

# rename the columns
states.rename(columns={'index': 'state_code', 'State': 'cases'}, inplace=True)

# sort the DataFrame by cases (the no. of accidents)
states = states.sort_values(by='cases', ascending=False)
```

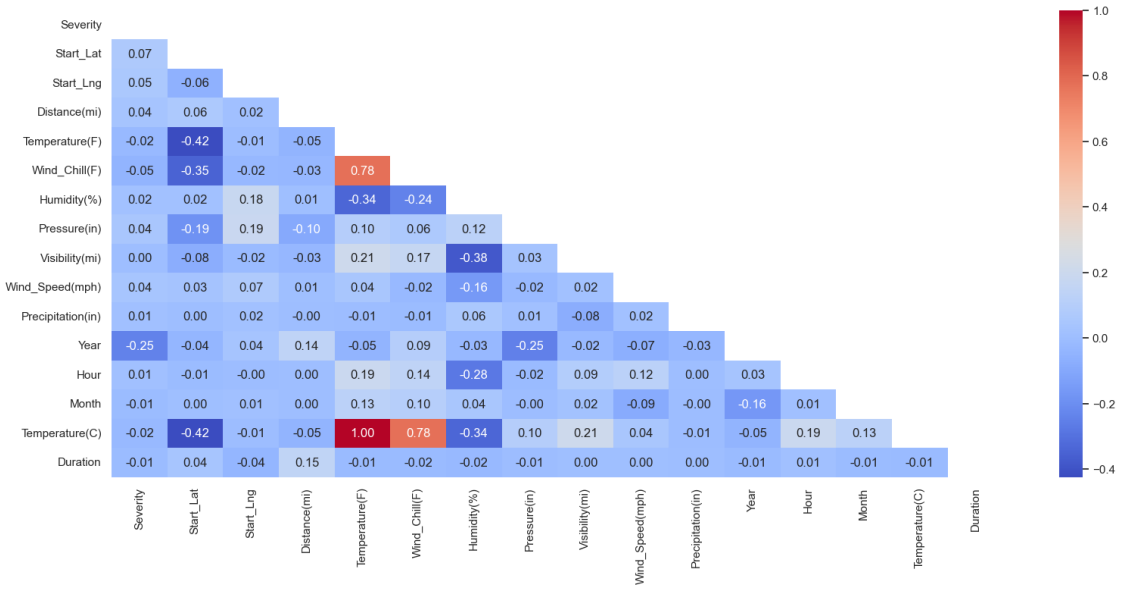
```
# print the sorted DataFrame
states.head()
```

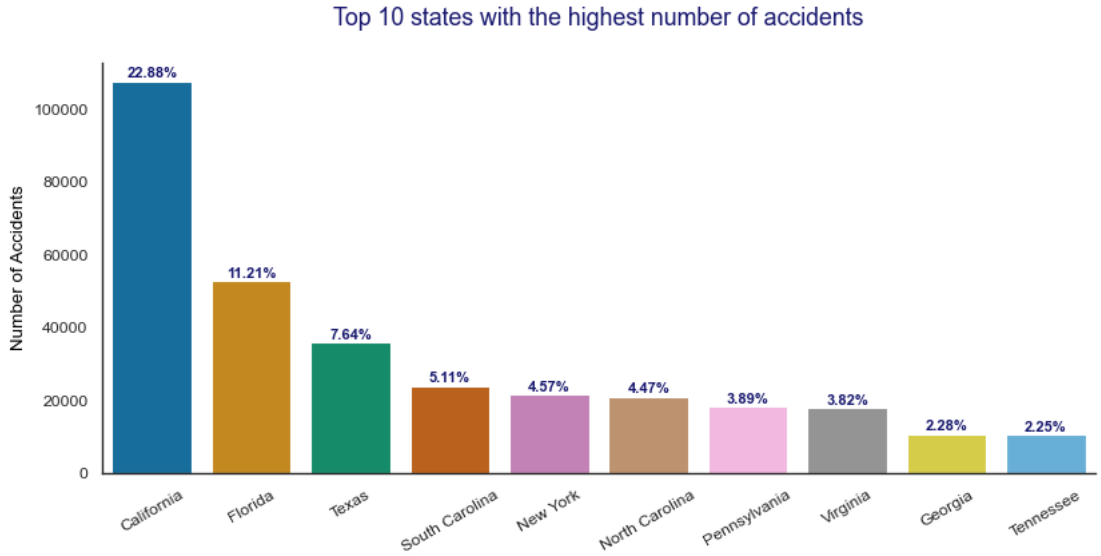
[ ]:
```
  state_code   cases
0         CA  107648
1         FL   52722
2         TX   35959
3         SC   24052
4         NY   21524
```

[ ]:
```
us_states = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA':␣
 ↪'California',
    'CO': 'Colorado', 'CT': 'Connecticut', 'DE': 'Delaware', 'FL': 'Florida',␣
 ↪'GA': 'Georgia',
    'HI': 'Hawaii', 'ID': 'Idaho', 'IL': 'Illinois', 'IN': 'Indiana', 'IA':␣
 ↪'Iowa', 'KS': 'Kansas',
    'KY': 'Kentucky', 'LA': 'Louisiana', 'ME': 'Maine', 'MD': 'Maryland', 'MA':␣
 ↪'Massachusetts',
    'MI': 'Michigan', 'MN': 'Minnesota', 'MS': 'Mississippi', 'MO': 'Missouri',␣
 ↪'MT': 'Montana',
    'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New␣
 ↪Jersey', 'NM': 'New Mexico',
    'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North Dakota', 'OH':␣
 ↪'Ohio', 'OK': 'Oklahoma',
    'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode Island', 'SC': 'South␣
 ↪Carolina',
    'SD': 'South Dakota', 'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT':␣
 ↪'Vermont',
    'VA': 'Virginia', 'WA': 'Washington', 'WV': 'West Virginia', 'WI':␣
 ↪'Wisconsin', 'WY': 'Wyoming'
}

# Add a new column 'State_Name' based on 'State_Code'
states['state'] = states['state_code'].map(us_states)

# Display the updated DataFrame
states.head()
```

[ ]:
```
  state_code   cases           state
0         CA  107648      California
1         FL   52722         Florida
2         TX   35959           Texas
3         SC   24052  South Carolina
4         NY   21524        New York
```

[ ]:
```
fig, ax = plt.subplots(figsize = (12,5), dpi = 80)
sns.set_style('ticks')

top_10 = states[:10]

sns.barplot(x=top_10['state'], y=top_10['cases'], palette='colorblind')

plt.title("Top 10 states with the highest number of accidents\n", fontdict =␣
 ↪{'fontsize':16, 'color':'MidnightBlue'})
plt.ylabel("\nNumber of Accidents", fontdict = {'fontsize':12, 'color':'black'})
plt.xticks(rotation=30)
plt.xlabel(None)

total_accidents = df_copy.shape[0]
for p in ax.patches :
    height = p.get_height()
    ax.text(p.get_x() + p.get_width()/2,
            height + 1000,
            '{:.2f}%'.format(height/total_accidents*100),
            ha = "center",
            fontsize = 10, weight = 'bold', color='MidnightBlue')

# Increase the font size of the axis tick labels
sns.set(rc={'xtick.labelsize': 12, 'ytick.labelsize': 12})

# Customize Y-axis tick labels to show real numbers
def format_func(value, _):
    return f'{value:.0f}'  # Format as whole numbers
ax.yaxis.set_major_formatter(FuncFormatter(format_func))

for i in ['top', 'right']:
    ax.spines[i].set_color('white')
    ax.spines[i].set_linewidth(1.5)

plt.show()
```





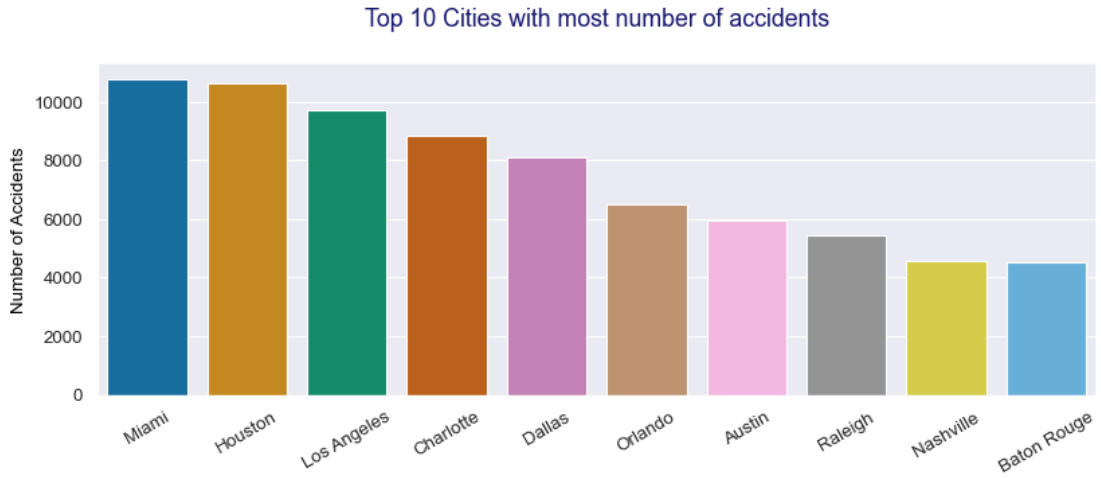
[ ]:
```
cities = pd.DataFrame(df_copy["City"].value_counts()).reset_index().
  ↪sort_values(by='City',ascending=False)
cities = cities.rename(columns={'index':'city','City':'cases'})
cities
```

[ ]:

	city	cases
0	Miami	10779
1	Houston	10627
2	Los Angeles	9708

```
3       Charlotte   8825
4          Dallas   8100
…             …    …
7920     Wellston      1
7919     Holliday      1
7918    East Bank      1
7917     Bushkill      1
9145   Fair Haven      1

[9146 rows x 2 columns]
```

[ ]:
```
fig, ax = plt.subplots(figsize = (12,4), dpi = 80)
sns.set_style('ticks')

sns.barplot(x=cities[:10].city, y=cities[:10].cases, palette='colorblind')
plt.title("Top 10 Cities with most number of accidents\n", fontdict =␣
 ↪{'fontsize':16, 'color':'MidnightBlue'})
plt.ylabel("\nNumber of Accidents", fontdict = {'fontsize':12, 'color':'black'})
plt.xlabel(None)
plt.xticks(rotation=30)

# Increase the font size of the axis tick labels
sns.set(rc={'xtick.labelsize': 12, 'ytick.labelsize': 12})

# Customize Y-axis tick labels to show real numbers
def format_func(value, _):
    return f'{value:.0f}'  # Format as whole numbers
ax.yaxis.set_major_formatter(FuncFormatter(format_func))

for i in ['top', 'right']:
    ax.spines[i].set_color('white')
    ax.spines[i].set_linewidth(1.5)

plt.show()
```

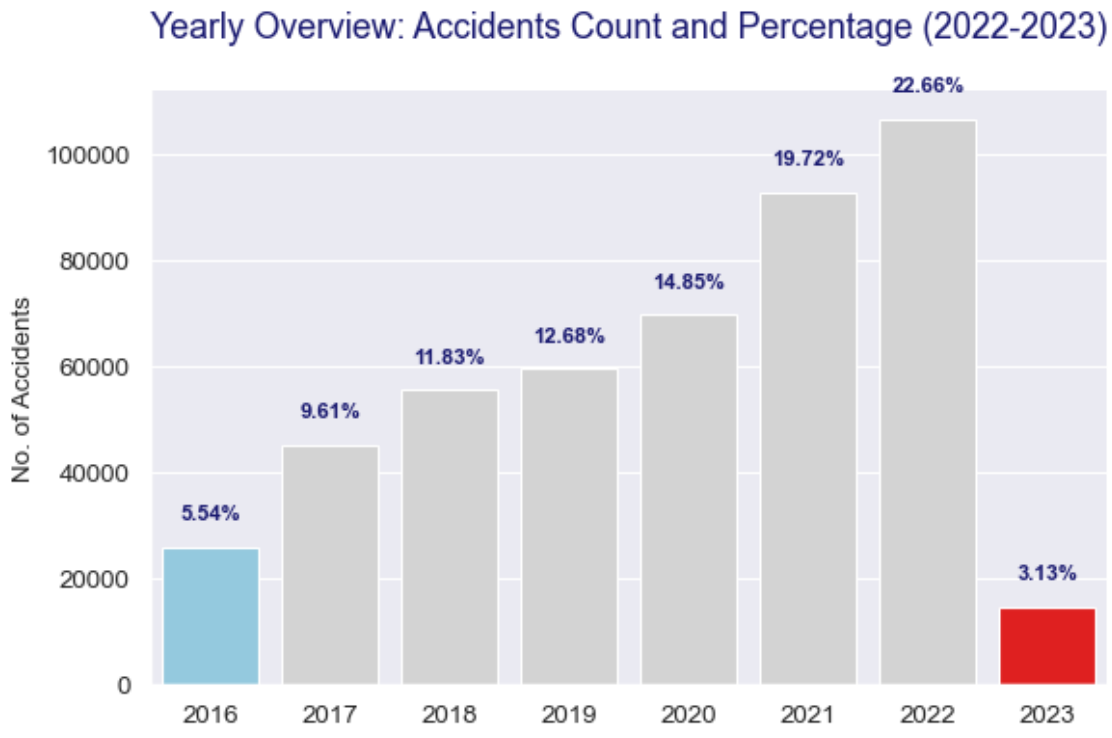


[ ]:
```
# convert the Start_Time and End_Time attributes to datetime
df_copy["Start_Time"] = pd.to_datetime(df_copy["Start_Time"], format="mixed",␣
 ↪errors='coerce', dayfirst=True)
df_copy["End_Time"] = pd.to_datetime(df_copy["End_Time"], format="mixed",␣
 ↪errors='coerce', dayfirst=True)

# Extract year, month, weekday and day
df_copy["Year"] = df_copy["Start_Time"].dt.year
df_copy["Month"] = df_copy["Start_Time"].dt.month
df_copy["Weekday"] = df_copy["Start_Time"].dt.weekday
df_copy["Day"] = df_copy["Start_Time"].dt.day
df_copy["Hour"] = df_copy["Start_Time"].dt.hour
```

[ ]:
```
year_df = pd.DataFrame(df_copy['Year'].value_counts()).reset_index().
 ↪sort_values(by='Year', ascending=True)
year = year_df.rename(columns={'index':'year','Year':'cases'})
```

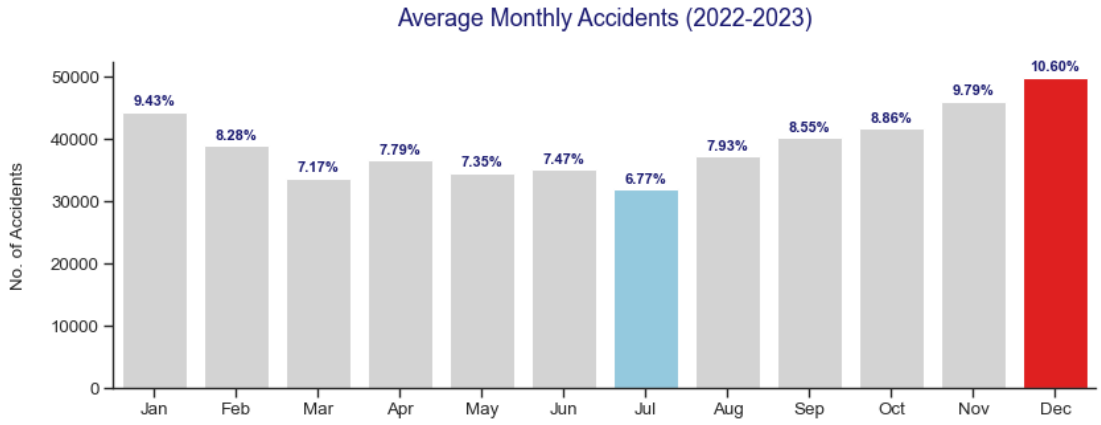
[ ]:
```
fig, ax = plt.subplots(figsize = (8,5), dpi = 80)
sns.set_style('ticks') # style must be one of white, dark, whitegrid, darkgrid,␣
 ↪ticks

# Determine the colors (as before)
colors = ['red' if val == max(year['cases']) else 'skyblue' if val ==␣
 ↪min(year['cases']) else 'lightgrey' for val in year['cases']]

sns.barplot(x=year.year, y=year.cases, palette=colors)
ax.spines[('top')].set_visible(False)
ax.spines[('right')].set_visible(False)
ax.set_xlabel(None)
ax.set_ylabel("No. of Accidents")
```

```
ax.set_title('Yearly Overview: Accidents Count and Percentage (2022-2023)\n',␣
 ↪fontdict = {'fontsize':16 , 'color':'MidnightBlue'})

# Customize Y-axis tick labels to show real numbers
def format_func(value, _):
    return f'{value:.0f}'  # Format as whole numbers
ax.yaxis.set_major_formatter(FuncFormatter(format_func))

for p in ax.patches :
    height = p.get_height()
    ax.text(p.get_x() + p.get_width()/2,
            height + 5000,
            '{:.2f}%'.format(height/total_accidents*100),
            ha = "center",
            fontsize = 10, weight='bold', color='MidnightBlue')

for i in ['top','right']:
    side = ax.spines[i]
    side.set_visible(False)

plt.show()
```



[ ]:
```
month_df = pd.DataFrame(df_copy.Start_Time.dt.month.value_counts()).
 ↪reset_index()
month = month_df.rename(columns={'index':'month#','Start_Time':'cases'}).
 ↪sort_values(by='month#', ascending=True)

# adding month name as a column
month_map = {1:'Jan' , 2:'Feb' , 3:'Mar' , 4:'Apr' , 5:'May' , 6:'Jun', 7:'Jul'␣
 ↪, 8:'Aug', 9:'Sep',10:'Oct' , 11:'Nov' , 12:'Dec'}
month['month_name'] = month['month#'].map(month_map)
```

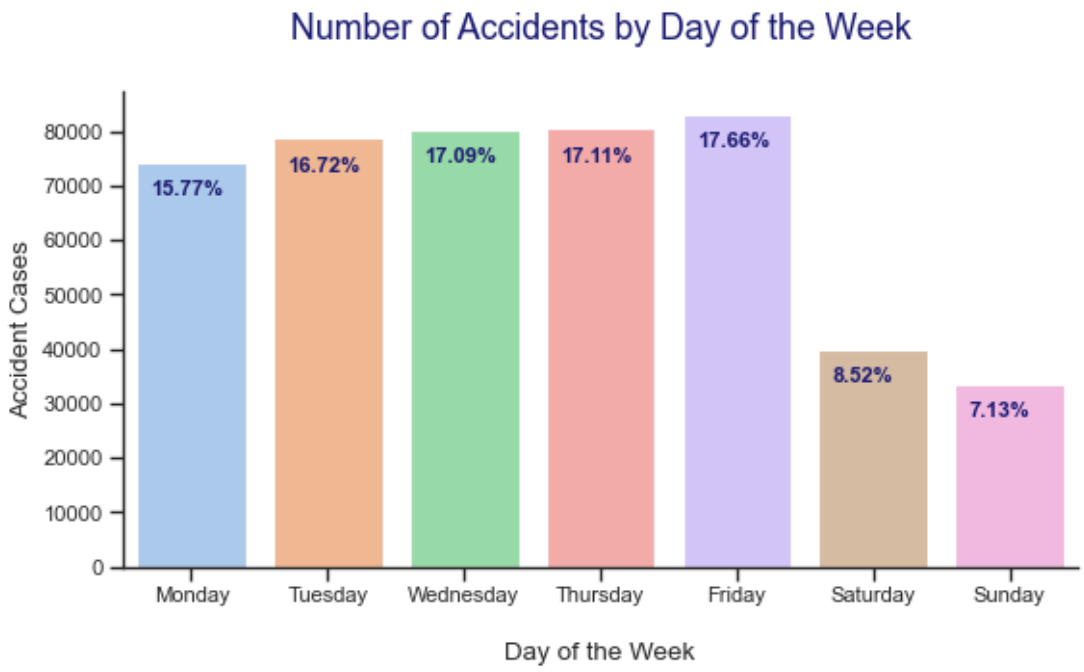
[ ]:
```
fig, ax = plt.subplots(figsize = (12,4), dpi = 80)
sns.set_style('ticks')

# Determine the colors (as before)
colors = ['red' if val == max(month['cases']) else 'skyblue' if val ==␣
 ↪min(month['cases']) else 'lightgrey' for val in month['cases']]

sns.barplot(x=month.month_name, y=month.cases, palette=colors)

ax.set_title('Average Monthly Accidents (2022-2023)\n', fontdict = {'fontsize':
 ↪16 , 'color':'MidnightBlue'})
ax.set_ylabel("\nNo. of Accidents\n", fontsize = 12)
ax.set_xlabel(None)

# Customize Y-axis tick labels to show real numbers
def format_func(value, _):
    return f'{value:.0f}'  # Format as whole numbers
ax.yaxis.set_major_formatter(FuncFormatter(format_func))

for p in ax.patches :
    height = p.get_height()
    ax.text(p.get_x() + p.get_width()/2,
            height + 1000,
            '{:.2f}%'.format(height/total_accidents*100),
            ha = "center",
            fontsize = 10, weight='bold', color='MidnightBlue')
for i in ['top', 'right']:
    side = ax.spines[i]
    side.set_visible(False)
plt.show()
```



[ ]:
```
dow = pd.DataFrame(df_copy['Start_Time'].dt.dayofweek.value_counts()).
 ↪reset_index()
dow = dow.rename(columns={'index':'day_of_week', 'Start_Time':'cases'}).
 ↪sort_values(by='day_of_week')
day_map = {0:'Monday' , 1:'Tuesday' , 2:'Wednesday' , 3:"Thursday" , 4:'Friday'␣
 ↪, 5:"Saturday" , 6:'Sunday'}
dow['weekday'] = dow['day_of_week'].map(day_map)
```

[ ]:
```
fig, ax = plt.subplots(figsize = (8,4), dpi = 80)
sns.set_style('ticks')

ax=sns.barplot(y=dow.cases, x=dow.weekday, palette='pastel')
plt.title('Number of Accidents by Day of the Week\n', size=16,␣
 ↪color='MidnightBlue')
plt.ylabel('\nAccident Cases', fontsize=12)
plt.xlabel('\nDay of the Week', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

total = df_copy.shape[0]
for i in ax.patches:
    ax.text(i.get_x()+0.1, i.get_height()-5000,
    str(round((i.get_height()/total)*100, 2))+'%',
    va = "center", fontsize=10, weight='bold', color='MidnightBlue')

for i in ['top', 'right']:
    side = ax.spines[i]
    side.set_visible(False)

# Customize Y-axis tick labels to show real numbers
def format_func(value, _):
    return f'{value:.0f}'  # Format as whole numbers
```

```
ax.yaxis.set_major_formatter(FuncFormatter(format_func))

plt.show()
```



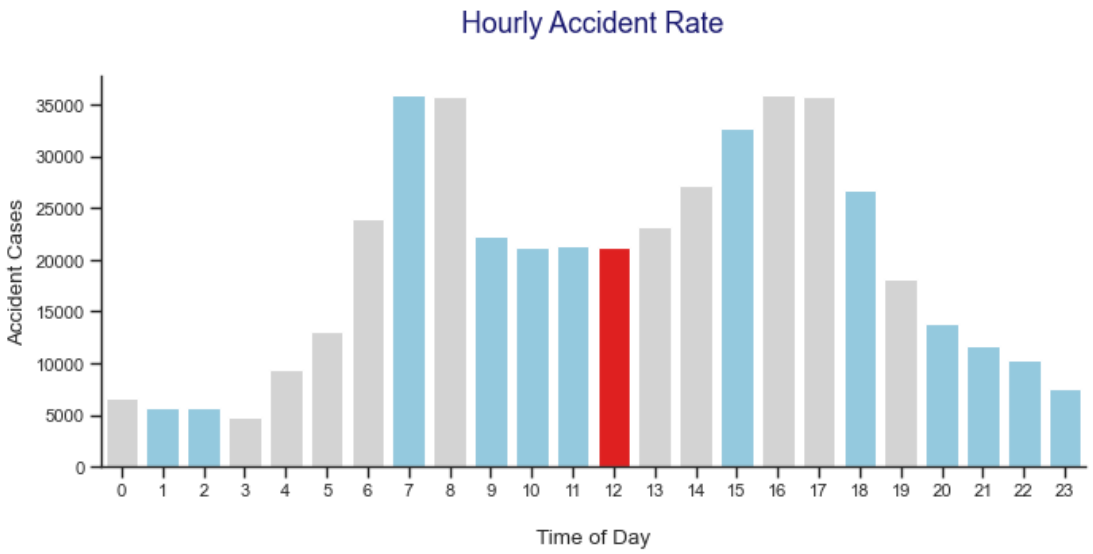
[ ]:
```
hour_of_day = pd.DataFrame(df_copy['Hour'].value_counts()).reset_index().
  ↪rename(columns={'index':'hour','Hour':'cases'})
# hour_of_day.sort_values(by='hour', inplace=True)
hour_of_day
```

[ ]:

	hour	cases
0	16	36025
1	7	35970
2	8	35829
3	17	35786
4	15	32750
5	14	27218
6	18	26759
7	6	24016
8	13	23232
9	9	22214
10	11	21446
11	10	21175
12	12	21145
13	19	18201
14	20	13774

```
15     5  13065
16    21  11696
17    22  10364
18     4   9333
19    23   7601
20     0   6675
21     1   5756
22     2   5666
23     3   4805
```

[ ]:
```
fig, ax = plt.subplots(figsize=(10, 4), dpi=80)
sns.set_style('ticks')

colors = []
for x in hour_of_day['cases']:
    if int(hour_of_day[hour_of_day['cases'] == x]['hour']) <=11:
        if x == max(list(hour_of_day['cases'])[:12]):
            colors.append('red')
        else:
            colors.append('skyblue')
    else:
        if x == max(list(hour_of_day['cases'])[12:]):
            colors.append('red')
        else:
            colors.append('lightgrey')

# Create a bar plot of 'hourly_accident_rate'
sns.barplot(x=hour_of_day.hour, y=hour_of_day.cases, palette=colors)

plt.title('Hourly Accident Rate\n', size=16, color='MidnightBlue')
plt.ylabel('\nAccident Cases', fontsize=12)
plt.xlabel('\nTime of Day', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for i in ['top', 'right']:
    side = ax.spines[i]
    side.set_visible(False)

plt.show()
```



[ ]:
```
print("No. of Weather Conditions:", len(df_copy["Weather_Condition"].unique()))

# To view the complete list of 142 weather descriptions, run the following code
print("\nList of unique weather conditions:", list(df_copy["Weather_Condition"].
  ↪unique()))
```

```
No. of Weather Conditions: 106

List of unique weather conditions: ['Fair', 'Wintry Mix', 'Light Rain',
'Cloudy', 'Partly Cloudy', 'Clear', 'Scattered Clouds', 'Mostly Cloudy', 'Fog',
'Overcast', 'Light Snow', 'T-Storm', 'Thunderstorms and Rain', 'Thunder', 'Light
Rain with Thunder', 'Rain', 'Showers in the Vicinity', 'Mostly Cloudy / Windy',
'Heavy Rain', 'Cloudy / Windy', 'Light Drizzle', 'Heavy T-Storm', 'Light Rain /
Windy', 'Smoke', 'Haze', 'Blowing Dust / Windy', 'N/A Precipitation', 'Thunder
in the Vicinity', 'Snow', 'Heavy Thunderstorms and Rain', 'Shallow Fog', 'Light
Freezing Drizzle', 'Fair / Windy', 'Patches of Fog', 'Light Snow / Windy',
'Thunderstorm', 'Drizzle', 'T-Storm / Windy', 'Partly Cloudy / Windy', 'Heavy
Rain / Windy', 'Mist', 'Light Thunderstorms and Rain', 'Rain / Windy', 'Light
Freezing Rain', 'Heavy Snow', 'Light Ice Pellets', 'Heavy T-Storm / Windy',
'Heavy Drizzle', 'Sleet', 'Light Rain Shower', 'Haze / Windy', 'Snow and Sleet',
'Snow / Windy', 'Fog / Windy', 'Light Freezing Fog', 'Sleet / Windy', 'Light
Sleet', 'Sand / Dust Whirlwinds', 'Squalls / Windy', 'Thunder / Wintry Mix',
'Light Haze', 'Freezing Drizzle', 'Light Hail', 'Heavy Snow / Windy', 'Blowing
Dust', 'Drizzle and Fog', 'Thunder / Windy', 'Small Hail', 'Light Rain Showers',
'Ice Pellets', 'Funnel Cloud', 'Light Freezing Rain / Windy', 'Light Drizzle /
Windy', 'Rain Shower', 'Partial Fog / Windy', 'Hail', 'Snow and Thunder', 'Rain
Showers', 'Heavy Freezing Drizzle', 'Wintry Mix / Windy', 'Light Snow and
Sleet', 'Freezing Rain', 'Light Snow Shower', 'Blowing Snow / Windy', 'Tornado',
'Widespread Dust', 'Light Snow with Thunder', 'Smoke / Windy', 'Widespread Dust
```

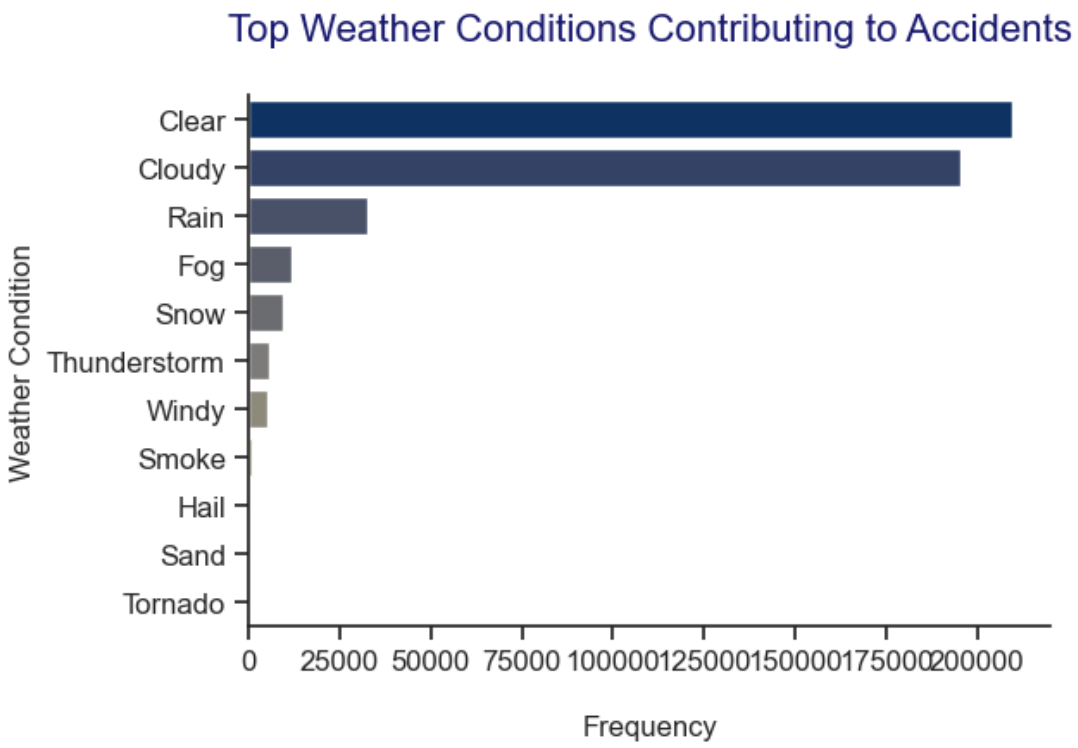
```
/ Windy', 'Light Snow Showers', 'Light Snow and Sleet / Windy', 'Snow and Sleet
/ Windy', 'Blowing Snow', 'Partial Fog', 'Drizzle / Windy', 'Heavy Sleet', 'Snow
Grains', 'Squalls', 'Light Rain Shower / Windy', 'Light Thunderstorms and Snow',
'Light Snow Grains', 'Thunder and Hail', 'Volcanic Ash', 'Mist / Windy', 'Light
Blowing Snow', 'Low Drifting Snow']
```

[ ]:
```
df_copy.loc[df_copy["Weather_Condition"].str.contains("Thunder|T-Storm",␣
 ↪na=False), "Weather_Condition"] = "Thunderstorm"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Snow|Sleet|Wintry",␣
 ↪na=False), "Weather_Condition"] = "Snow"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Rain|Drizzle|Shower",␣
 ↪na=False), "Weather_Condition"] = "Rain"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Wind|Squalls",␣
 ↪na=False), "Weather_Condition"] = "Windy"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Hail|Pellets",␣
 ↪na=False), "Weather_Condition"] = "Hail"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Fair", na=False),␣
 ↪"Weather_Condition"] = "Clear"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Cloud|Overcast",␣
 ↪na=False), "Weather_Condition"] = "Cloudy"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Mist|Haze|Fog",␣
 ↪na=False), "Weather_Condition"] = "Fog"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Sand|Dust", na=False),␣
 ↪"Weather_Condition"] = "Sand"
df_copy.loc[df_copy["Weather_Condition"].str.contains("Smoke|Volcanic Ash",␣
 ↪na=False), "Weather_Condition"] = "Smoke"
df_copy.loc[df_copy["Weather_Condition"].str.contains("N/A Precipitation",␣
 ↪na=False), "Weather_Condition"] = np.nan
```

[ ]:
```
wc = pd.DataFrame(df_copy['Weather_Condition'].value_counts()).reset_index().
 ↪sort_values(by='Weather_Condition', ascending=False)
wc.rename(columns={'index':'weather_condition', 'Weather_Condition':
 ↪'frequency'}, inplace=True)
# wc stands for weather condition
```

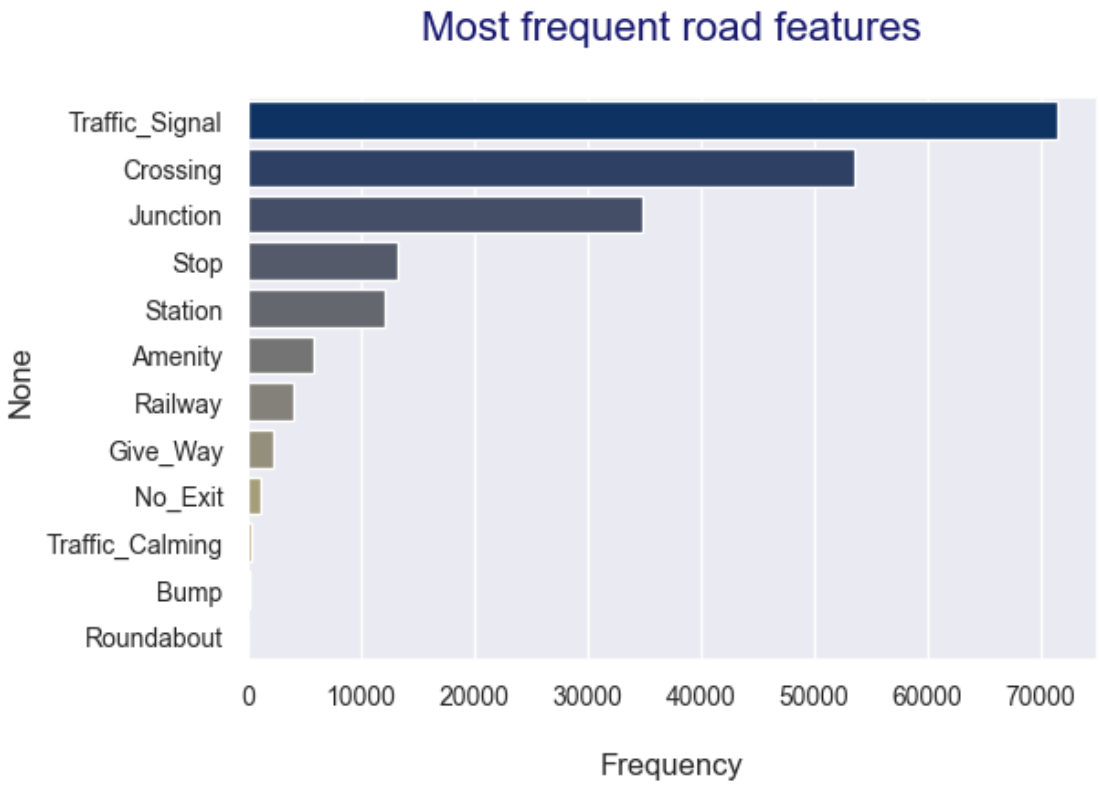
[ ]:
```
# Create a figure and axis
fig, ax = plt.subplots(figsize=(6, 4))
sns.set_style('ticks')
sns.barplot(x='frequency', y='weather_condition', data=wc, palette='cividis',␣
 ↪orient='h')

# Add labels and title
ax.set_xlabel('\nFrequency')
ax.set_ylabel('\nWeather Condition')
ax.set_title('\nTop Weather Conditions Contributing to Accidents\n',␣
 ↪fontsize=16, color='MidnightBlue')
```

```
plt.xticks(rotation=0)  # Adjust the rotation angle of x-axis labels

# Increase the font size of the axis tick labels
sns.set(rc={'xtick.labelsize': 10, 'ytick.labelsize': 10})

# Remove top and right spines
for i in ['top', 'right']:
    ax.spines[i].set_visible(False)

# Show the plot
plt.show()
```



[ ]:
```
road_features = ["Amenity", "Bump", "Crossing", "Give_Way", "Junction",␣
 ↪"No_Exit", "Railway", "Roundabout", "Station", "Stop", "Traffic_Calming",␣
 ↪"Traffic_Signal"]

data = df_copy[road_features].sum().sort_values(ascending=False)

fig, ax = plt.subplots(figsize=(6, 4))
sns.barplot(x=data.values, y=data.index, orient="h", palette='cividis')
plt.title("Most frequent road features\n", fontsize=16, color='MidnightBlue')
```

```
plt.xlabel("\nFrequency")

plt.show()
```



[ ]:
```
import nltk
stop = stopwords.words("english") + ["-"]
```

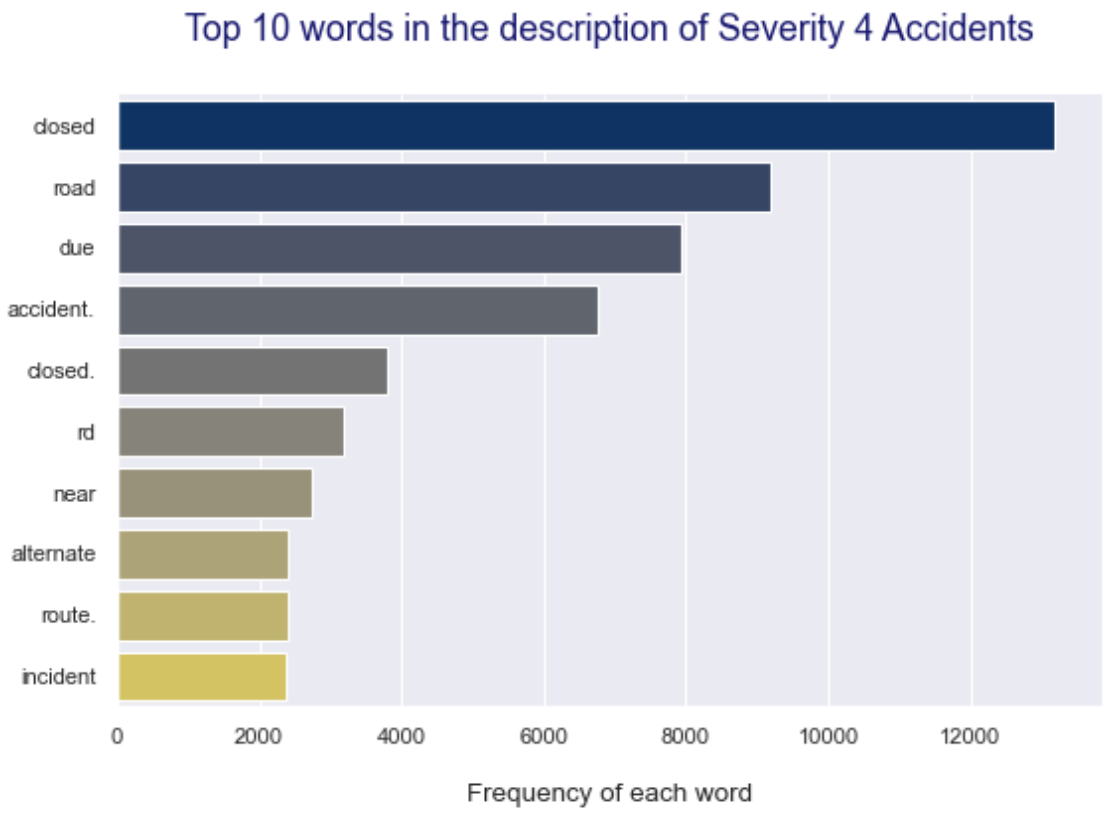
[ ]:
```
description_s4 = df_copy[df_copy["Severity"] == 4]["Description"] # filter the␣
↪data
# Split the description
df_words = description_s4.str.lower().str.split(expand=True).stack()
```

[ ]:
```
# If the word is not in the stopwords list
counts = df_words[~df_words.isin(stop)].value_counts()[:10]
print(counts)
```

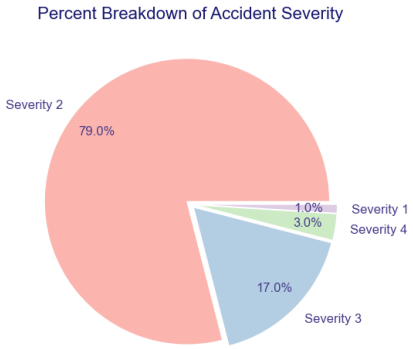
```
closed       13189
road          9188
due           7936
accident.     6758
closed.       3790
rd            3186
```

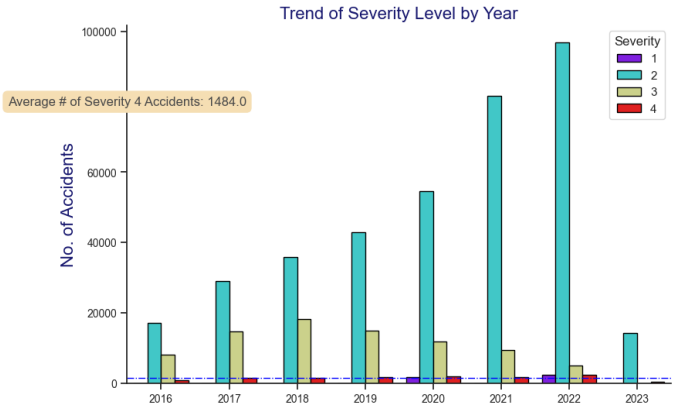
```
near          2730
alternate     2417
route.        2393
incident      2381
dtype: int64
```

[ ]:
```
# visualize the frequencies of the top 10 words in the description
fig, ax = plt.subplots(figsize=(8, 5), dpi=80)
sns.set_style('ticks')
sns.barplot(x=counts.values, y=counts.index, orient="h", palette = "cividis")
# sns.barplot(x=counts.values, y=counts.index, orient="h", palette="viridis")
# sns.barplot(x=counts.values, y=counts.index, orient="h")

ax.set_title("Top 10 words in the description of Severity 4 Accidents\n",␣
 ↪fontsize=16, color='MidnightBlue')
ax.set_xlabel("\nFrequency of each word\n")
ax.set_ylabel(None)

for i in ['top', 'right']:
    side = ax.spines[i]
    side.set_visible(False)

plt.show()
```



[ ]:
```
s4_by_yr = df_copy[df_copy['Severity'] == 4][['Severity','Year']].
 ↪groupby('Year').agg({'Severity': 'count'}).mean().round(0)

fig, (ax1,ax2) = plt.subplots(1,2,figsize=(20,6))

# Calculate the percentage of each severity level
severity = df_copy['Severity'].value_counts(normalize=True).round(2) * 100
severity.plot.pie(autopct = '%1.1f%%' , ax=ax1, colors =sns.
 ↪color_palette(palette='Pastel1'),
                                        pctdistance = 0.8, explode = [.03,.03,.
 ↪03,.03],
                                        textprops = {'fontsize' : 12 , 'color' :
 ↪ 'DarkSlateBlue'},
                                        labels=['Severity 2','Severity 3' ,␣
 ↪'Severity 4' , 'Severity 1'])

ax1.set_title("Percent Breakdown of Accident Severity", fontdict = {'fontsize':
 ↪16 , 'color':'MidnightBlue'} )
ax1.set_ylabel(None)

s = sns.countplot(data=df_copy[['Severity','Year']] , x = 'Year' ,␣
 ↪hue='Severity' , ax=ax2, palette = 'rainbow', edgecolor='black')

ax2.axhline(s4_by_yr[0], color='Blue', linewidth=1, linestyle='dashdot')
ax2.annotate(f"Average # of Severity 4 Accidents: {s4_by_yr[0]}",
            va = 'center', ha='center',
            color='#4a4a4a',
            bbox=dict(boxstyle='round', pad=0.4, facecolor='Wheat',␣
 ↪linewidth=0), xy=(-0.5,80000))

ax2.set_title("Trend of Severity Level by Year", fontdict = {'fontsize':16 ,␣
 ↪'color':'MidnightBlue'} )
ax2.set_ylabel("\nNo. of Accidents", fontdict = {'fontsize':16 , 'color':
 ↪'MidnightBlue'} )
ax2.set_xlabel(None)

for i in ['top', 'right']:
    side = ax2.spines[i]
    side.set_visible(False)

# sns.despine(left=True)
plt.show()
```





[ ]:

[ ]:
```
# %%html
# <iframe title="US Accidents - Accidents Locations Dashboard" width="1140"␣
 ↪height="541.25" src="https://app.powerbi.com/reportEmbed?
 ↪reportId=f7262d0d-05b6-4938-978b-95748f173719&autoAuth=true&ctid=dbd6664d-4eb9-46eb-99d8-5c
 ↪frameborder="0" allowFullScreen="true"></iframe>
```

[ ]:
```
df_copy.columns
```

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

[ ]:
```
df_copy.head(3)
```

[ ]:
```
          ID   Source  Severity          Start_Time            End_Time  \
0  A-2047758  Source2         2 2019-06-12 10:10:56 2019-06-12 10:55:58
1  A-4694324  Source1         2 2022-12-03 23:37:14 2022-12-04 01:56:53
2  A-5006183  Source1         2 2022-08-20 13:13:00 2022-08-20 15:22:45

   Start_Lat   Start_Lng  Distance(mi)  \
0  30.641211  -91.153481         0.000
```

```
1  38.990562  -77.399070         0.056
2  34.661189 -120.492822         0.022

                                          Description            Street  …  \
0  Accident on LA-19 Baker-Zachary Hwy at Lower Z…        Highway 19  …
1  Incident on FOREST RIDGE DR near PEPPERIDGE PL…   Forest Ridge Dr  …
2  Accident on W Central Ave from Floradale Ave t…     Floradale Ave  …

   Year Hour Month Day Is_Weekend Year_Month Temperature(C)    Duration  \
0  2019   10     6  12      False    2019-06      25.000000   45.033333
1  2022   23    12   3       True    2022-12       7.222222  139.650000
2  2022   13     8  20       True    2022-08      20.000000  129.750000

   cluster_LatLng  Weekday
0               2        2
1               3        5
2               1        5

[3 rows x 54 columns]
```

[ ]:
```
df_copy.describe()
```

[ ]:
```
            Severity      Start_Lat      Start_Lng   Distance(mi)  \
count  470501.000000  470501.000000  470501.000000  470501.000000
mean        2.215356      36.163513     -94.710503       0.528834
std         0.487005       5.021527      17.436777       1.668964
min         1.000000      24.562117    -124.497420       0.000000
25%         2.000000      33.412563    -117.248543       0.000000
50%         2.000000      35.788753     -87.644279       0.019000
75%         2.000000      40.038849     -80.367728       0.430000
max         4.000000      48.991585     -67.484130     193.479996

       Temperature(F)  Wind_Chill(F)    Humidity(%)   Pressure(in)  \
count   470501.000000  470501.000000  470501.000000  470501.000000
mean        62.283057      59.110685      64.770538      29.546435
std         18.085153      21.070755      22.893141       0.986629
min         14.000000      -9.300000       1.000000       2.990000
25%         50.000000      43.000000      48.000000      29.380000
50%         64.000000      63.000000      67.000000      29.860000
75%         76.000000      75.000000      84.000000      30.030000
max        107.600000     107.000000     100.000000      38.440000

       Visibility(mi)  Wind_Speed(mph)  Precipitation(in)          Year  \
count   470501.000000    470501.000000      470501.000000  470501.000000
mean         9.105735         7.667757           0.008708    2019.871205
std          2.685973         5.396414           0.112725       1.928460
min          0.000000         0.000000           0.000000    2016.000000
```

```
25%          10.000000          4.600000           0.000000    2018.000000
50%          10.000000          7.000000           0.000000    2020.000000
75%          10.000000         10.400000           0.000000    2022.000000
max         130.000000        822.800000          10.130000    2023.000000

                Hour           Month             Day  Temperature(C)  \
count  470501.000000  470501.000000  470501.000000   470501.000000
mean       12.387833       6.715799      15.729140       16.823921
std         5.447358       3.610765       8.673874       10.047307
min         0.000000       1.000000       1.000000      -10.000000
25%         8.000000       4.000000       8.000000       10.000000
50%        13.000000       7.000000      16.000000       17.777778
75%        17.000000      10.000000      23.000000       24.444444
max        23.000000      12.000000      31.000000       42.000000

            Duration  cluster_LatLng        Weekday
count  470501.000000   470501.000000  470501.000000
mean       94.839852        1.698942       2.582296
std        85.185572        1.243513       1.800927
min         2.500000        0.000000       0.000000
25%        30.000000        1.000000       1.000000
50%        72.250000        2.000000       3.000000
75%       121.650000        3.000000       4.000000
max       400.000000        4.000000       6.000000
```

[ ]:
```
X = df_copy[['Month', 'Weekday', 'Hour', 'Start_Lat', 'Temperature(F)',␣
 ↪'Visibility(mi)', 'Pressure(in)']]
y = df_copy['Severity']
# df['column_name'] = pd.to_numeric(df['column_name'], errors='coerce')
```

[ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 42)
```

[ ]:
```
X_train[['Month', 'Weekday', 'Hour', 'Start_Lat', 'Temperature(F)',␣
 ↪'Visibility(mi)', 'Pressure(in)']].isna().sum()
y_train.isna().sum()
X_test [['Month', 'Weekday', 'Hour', 'Start_Lat', 'Temperature(F)',␣
 ↪'Visibility(mi)', 'Pressure(in)']].isna().sum()
y_train .isna().sum()
```

[ ]:
```
0
```

[ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

```
# Assuming you have your features in X and labels in y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create a GradientBoostingClassifier
model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,␣
 ↪max_depth=3, random_state=42)

# Train the classifier
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.79
              precision    recall  f1-score   support

           1       0.33      0.00      0.00       798
           2       0.79      1.00      0.88     74614
           3       0.77      0.00      0.00     16325
           4       0.00      0.00      0.00      2364

    accuracy                           0.79     94101
   macro avg       0.47      0.25      0.22     94101
weighted avg       0.76      0.79      0.70     94101
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Assuming X contains your features and y contains your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create a random forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training set
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# You can also print a classification report for more detailed metrics
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.79
              precision    recall  f1-score   support

           1       0.40      0.04      0.07       798
           2       0.80      0.98      0.88     74614
           3       0.48      0.09      0.15     16325
           4       0.41      0.05      0.08      2364

    accuracy                           0.79     94101
   macro avg       0.52      0.29      0.30     94101
weighted avg       0.73      0.79      0.73     94101
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Create a logistic regression model
model = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=1000)

# Train the model on the training set
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# You can also print a classification report for more detailed metrics
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.79
              precision    recall  f1-score   support

           1       0.00      0.00      0.00      1249
           2       0.79      1.00      0.89    112054
           3       0.00      0.00      0.00     24287
           4       0.00      0.00      0.00      3561

    accuracy                           0.79    141151
```

```
   macro avg       0.20      0.25      0.22    141151
weighted avg       0.63      0.79      0.70    141151
```

[ ]:
```python
# the list of algorithms for classification as Baselines are
from sklearn.linear_model import LogisticRegression  #drawing a line based on␣
 ↪linear regression but used for classification
from sklearn.ensemble import RandomForestClassifier  #using trees to classify
from sklearn.svm import SVC  #drawing a line based on maximum distance used as␣
 ↪support vector classification

# validate with random_state 42

lr = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=1000)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,␣
 ↪max_depth=3)

models = [lr, rf, gbc]

# perform cross validation using KFold
from sklearn.model_selection import KFold, cross_val_score

kfold = KFold(n_splits = 5, shuffle = True, random_state=42)

for model in models:
    score = cross_val_score(model, X_train, y_train, cv=kfold,␣
 ↪scoring='accuracy')  #f1, recall, precision, accuracy
    print("Sklearn Model: ", model)
    print("Scores: ", score, "- Scores mean: ", score.mean(), "- Scores std: ",␣
 ↪score.std())  #out of 1 ; 1 means perfect accuracy
# #Classification report
# y_pred = model.predict(X_test)
# accuracy = accuracy_score(y_test, y_pred)
# print(f'Accuracy: {accuracy:.2f}')
# # also print a classification report for more detailed metrics
# print(classification_report(y_test, y_pred))
```

```
Sklearn Model:  LogisticRegression(max_iter=1000)
Scores:  [0.792322   0.79129915 0.79274708 0.79161796 0.79167109] - Scores mean:
0.7919314558979809 - Scores std:  0.0005260495495234573
Sklearn Model:  RandomForestClassifier(random_state=42)
Scores:  [0.79278693 0.79144527 0.79123273 0.79152497 0.79002391] - Scores mean:
0.7914027630180659 - Scores std:  0.0008789541155617807
Sklearn Model:  GradientBoostingClassifier()
Scores:  [0.79258767 0.79128587 0.79285335 0.79173751 0.79156482] - Scores mean:
0.7920058448459086 - Scores std:  0.0006069026977779012
```

[ ]:
```
# grid search to find the best version of the selected model

from sklearn.model_selection import GridSearchCV

Best_model = LogisticRegression(multi_class='auto', random_state=142) ␣
 ↪#<----this is the model I choose, after cross validation

param_grid = dict()
param_grid['solver'] = ['newton-cg', 'lbfgs', 'liblinear']

#refit means it will pick the best model, and fit again, so it means grid is␣
 ↪already the best model after this line
grid = GridSearchCV(Best_model, param_grid, scoring="accuracy", cv=kfold,␣
 ↪refit=True, return_train_score=True)
#scoring = f1, recall, precision, accuracy

#fit the grid, which will basically do cross validation across all␣
 ↪combinatiosn, here we only have 3 comb
grid.fit(X_train, y_train)

#print the best parameters and accuracy
print('best parameters -',grid.best_params_)
print('best score Mean -', grid.best_score_)
print('best fitted results', grid.cv_results_)
```

```
best parameters - {'solver': 'liblinear'}
best score Mean - 0.7919314558979809
best fitted results {'mean_fit_time': array([86.58514104,  6.11812353,
19.39373283]), 'std_fit_time': array([ 7.49853372,  0.4708449 , 11.84574211]),
'mean_score_time': array([0.01388288, 0.01184278, 0.02071357]),
'std_score_time': array([0.00325805, 0.0017522 , 0.00682078]), 'param_solver':
masked_array(data=['newton-cg', 'lbfgs', 'liblinear'],
             mask=[False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'solver': 'newton-cg'}, {'solver':
'lbfgs'}, {'solver': 'liblinear'}], 'split0_test_score': array([0.792322,
0.792322, 0.792322]), 'split1_test_score': array([0.79129915, 0.79129915,
0.79129915]), 'split2_test_score': array([0.79274708, 0.79274708, 0.79274708]),
'split3_test_score': array([0.79160468, 0.79159139, 0.79161796]),
'split4_test_score': array([0.79167109, 0.79167109, 0.79167109]),
'mean_test_score': array([0.7919288 , 0.79192614, 0.79193146]),
'std_test_score': array([0.00052766, 0.00052931, 0.00052605]),
'rank_test_score': array([2, 3, 1]), 'split0_train_score': array([0.7918305 ,
0.7918305 , 0.79183382]), 'split1_train_score': array([0.79208621, 0.79208621,
0.79208953]), 'split2_train_score': array([0.79172423, 0.79172423, 0.79172755]),
'split3_train_score': array([0.79200983, 0.79200983, 0.79200983]),
'split4_train_score': array([0.79199323, 0.79199323, 0.79199655]),
```

```
'mean_train_score': array([0.7919288 , 0.7919288 , 0.79193146]),
'std_train_score': array([0.00013191, 0.00013191, 0.00013151])}
```

[ ]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

# Assuming X contains your features and y contains your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create a random forest classifier
rf_classifier = RandomForestClassifier()

# Define the hyperparameter grid to search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the grid search with cross-validation
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid,␣
 ↪cv=3, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model from the grid search
Best_model_rf = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred = Best_model_rf.predict(X_test)

# Evaluate the performance of the best model
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of the Best Model: {accuracy:.2f}')
```

```
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50}
Accuracy of the Best Model: 0.83
```