College of Professional Studies

Northeastern University San Jose

MPS Analytics

Course: ALY6110 – Data Management and Big Data

Assignment:

Module 4 Final Project Basic Analysis

Submitted on:
June 26th, 2023

Submitted to:
Professor: BEHZAD AHMADI

Submitted by:
ARCHIT BARUA
BHAGYASHRI KADAM
NIKSHITA RANGANATHAN

PROBLEM DEFINITION

The problem of traffic accidents in the United States is a significant concern that has been escalating over time. Researchers, law enforcement agencies, government bodies, and related organizations have been actively working to address this issue. The number of traffic accident fatalities each year in the US is alarmingly high, reaching nearly 1.25 million deaths.

This problem affects all states across the country, and the frequency of accidents has been on the rise. The increasing number of accidents poses a threat to public safety, necessitating urgent action to find effective solutions.

To gain valuable insights and address this problem, a dataset related to US traffic accidents would be analyzed. The analysis aims to identify meaningful patterns and provide actionable analysis that can aid in curbing the issue and reducing the number of accidents and fatalities on the nation's roadways.

INTRODUCTION

About the Dataset:

The US Accidents dataset contains information about countrywide traffic accidents that occurred in the United States from February 2016 to March 2023. Each record includes details about the accident, such as the date, time, location, weather conditions etc.

The dataset also includes information about the road conditions, such as the type of road, the number of lanes, and the presence of traffic lights or stop signs. This dataset is a valuable resource for businesses and organizations that are interested in improving road safety.

The dataset has around **7.7M records and 46 columns.**

Data source: The dataset has been sourced from Kaggle:

https://www.kaggle.com/datasets/sobhanmoosavi/us -accidents

The data is continuously being collected from February 2016, using several data providers, including multiple APIs that provide streaming traffic event data.

Objective/Business Questions :

1. What are the key factors contributing to traffic accidents in a specific city/state ?
2. What the most effective ways to mitigate these accidents?
3. What is the relationship between accident severity and factors such as visibility, temperature, and precipitation?

4. Which are the top states that have the highest number of accidents?
5. How does the weather has an impact on the accidents?
6. Which time of the year, week, day of the week and time of day have the highest accident occurrence?
7. Is there a relationship between the severity of accidents and the presence of specific infrastructural features like traffic lights, stop signs, or pedestrian crossings?

Inspiration/Applications of Dataset :

The US-Accidents dataset can be applied to a wide range of applications, including the prediction of car accidents in real time, the analysis of accident hotspots, casualty analysis, the extraction of cause-and-effect rules to forecast car accidents, and the study of how precipitation or other environmental factors affect the likelihood of accidents occurring.

Group Members :

- Archit Barua
- Bhagyashri Kadam
- Nikshita Ranganathan

Methodologies Used:

The study on US traffic accidents used a combination of methodologies and techniques to analyze the data, derive insights, and answer the proposed business questions. Here are the methods utilized throughout the process:

Data Acquisition: The dataset was collected from Kaggle, a popular online platform for data science projects, which provides access to a multitude of datasets.

Data Cleaning and Preprocessing: The raw dataset underwent several cleaning processes to ensure the quality of the data being used. This involved handling missing data, dropping irrelevant columns, renaming columns for better understanding, and replacing certain values for consistency.

Feature Engineering: New features were derived from the existing ones to provide deeper insights. This included extracting the year from the timestamp, converting Start_time to timestamp format, and extracting the month, day of the week, and hour from the Start_time.

Descriptive Analysis: Basic statistical measures like count, mean, standard deviation, minimum, and maximum were calculated for several numerical columns in the dataset. This analysis provided a good understanding of the data distribution and variability.

Correlation Analysis: Correlation matrix was used to determine the relationship between different numerical variables in the dataset. It helped in understanding whether and how these variables are associated with each other.

Data Visualization: Various data visualization techniques were employed to represent the data visually and to understand the patterns, trends, and relationships more intuitively. These included histograms, line graphs, pie charts, and bar plots.

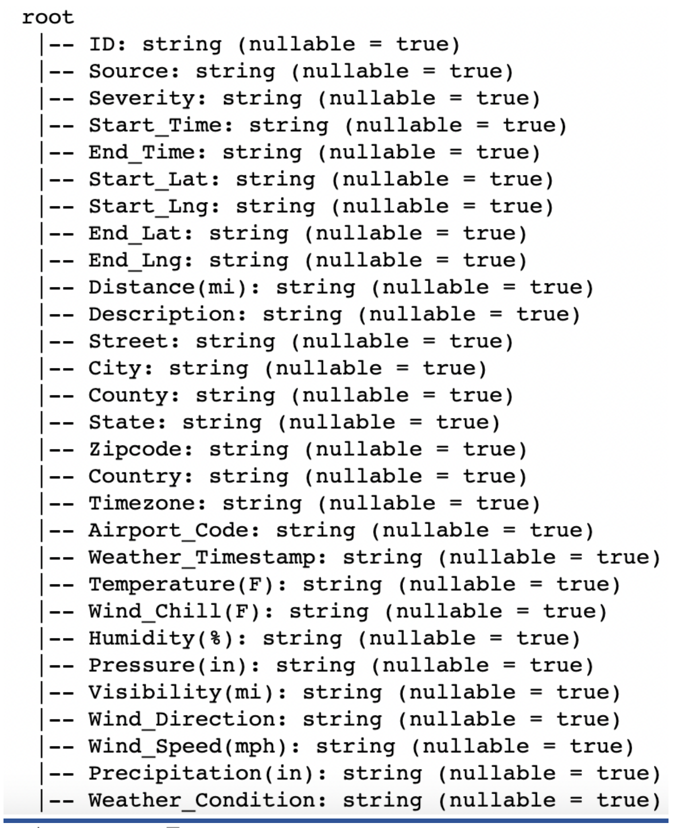
Temporal Analysis: Time-based data analysis was conducted to examine the trends and patterns of accidents over different time periods – years, months, days, and hours.

Geographical Analysis: The data was analyzed based on geographical location to understand which states reported the highest number of accidents. This could help in identifying areas that need more focus in terms of road safety.

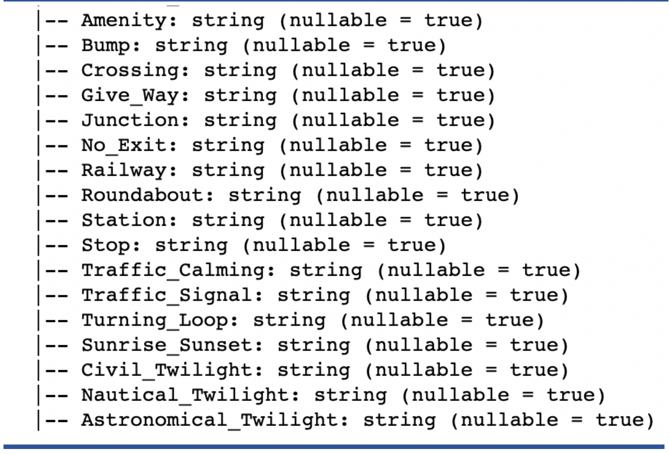
These methodologies were crucial in breaking down the complex problem of traffic accidents into smaller, more manageable segments. The insights derived from these techniques provided a holistic view of the issue and helped answer the business questions accurately.

Understanding the Dataset :

This dataset was has been sourced from the Kaggle platform but originally gathered by using multiple APIs that provide streaming traffic incident (or event) data. These APIs broadcast traffic data captured by various entities, including the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road networks.
Both numerical and categorical data types are present in this dataset. Below are the basic schema of the Raw dataset:



```
root
 |-- ID: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Severity: string (nullable = true)
 |-- Start_Time: string (nullable = true)
 |-- End_Time: string (nullable = true)
 |-- Start_Lat: string (nullable = true)
 |-- Start_Lng: string (nullable = true)
 |-- End_Lat: string (nullable = true)
 |-- End_Lng: string (nullable = true)
 |-- Distance(mi): string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Street: string (nullable = true)
 |-- City: string (nullable = true)
 |-- County: string (nullable = true)
 |-- State: string (nullable = true)
 |-- Zipcode: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Timezone: string (nullable = true)
 |-- Airport_Code: string (nullable = true)
 |-- Weather_Timestamp: string (nullable = true)
 |-- Temperature(F): string (nullable = true)
 |-- Wind_Chill(F): string (nullable = true)
 |-- Humidity(%): string (nullable = true)
 |-- Pressure(in): string (nullable = true)
 |-- Visibility(mi): string (nullable = true)
 |-- Wind_Direction: string (nullable = true)
 |-- Wind_Speed(mph): string (nullable = true)
 |-- Precipitation(in): string (nullable = true)
 |-- Weather_Condition: string (nullable = true)
```



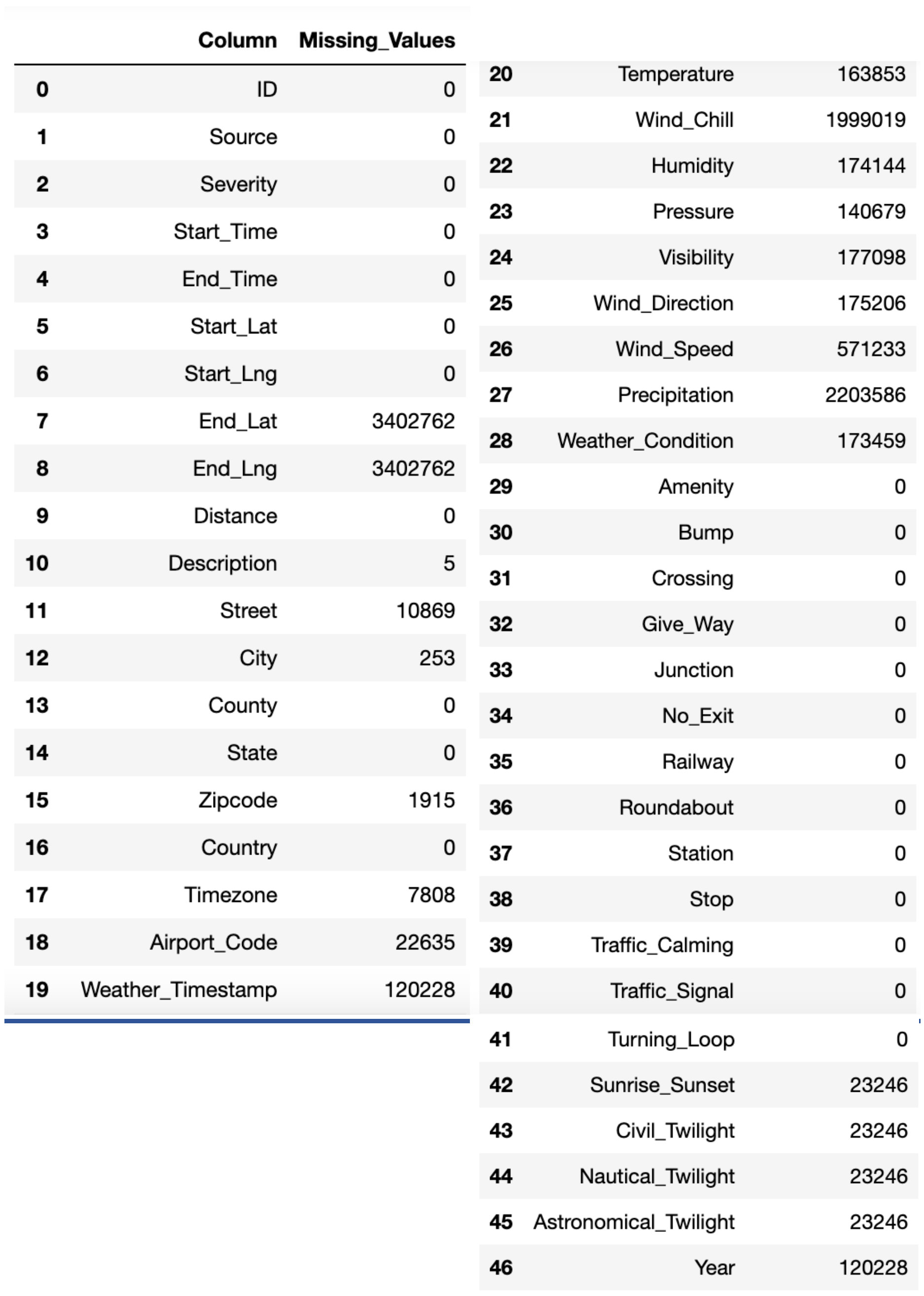
```
 |-- Amenity: string (nullable = true)
 |-- Bump: string (nullable = true)
 |-- Crossing: string (nullable = true)
 |-- Give_Way: string (nullable = true)
 |-- Junction: string (nullable = true)
 |-- No_Exit: string (nullable = true)
 |-- Railway: string (nullable = true)
 |-- Roundabout: string (nullable = true)
 |-- Station: string (nullable = true)
 |-- Stop: string (nullable = true)
 |-- Traffic_Calming: string (nullable = true)
 |-- Traffic_Signal: string (nullable = true)
 |-- Turning_Loop: string (nullable = true)
 |-- Sunrise_Sunset: string (nullable = true)
 |-- Civil_Twilight: string (nullable = true)
 |-- Nautical_Twilight: string (nullable = true)
 |-- Astronomical_Twilight: string (nullable = true)
```

The above output shows the schema of our DataFrame representing traffic accident data. It includes various columns such as ID, Source, Severity, Start_Time, End_Time, Location coordinates (Start_Lat, Start_Lng, End_Lat, End_Lng), Distance, Description, Street, City, County, State, Zipcode, Country, Timezone, Airport_Code, Weather_Timestamp, Weather conditions (Temperature, Wind Chill, Humidity, Pressure, Visibility, Wind Direction, Wind Speed, Precipitation), and various other attributes related to traffic signals, junctions, and daylight conditions.

The schema indicates that the DataFrame contains these columns with their respective data types, all represented as strings. The data appears to be a mix of traffic-related information, geographical details, and weather conditions, potentially useful for analyzing and understanding traffic accidents and their associated factors.

DATA CLEANING & FEATURE ENGINEERING

- **Checking the number of missing values for each Attribute in the dataset**



	Column	Missing_Values
0	ID	0
1	Source	0
2	Severity	0
3	Start_Time	0
4	End_Time	0
5	Start_Lat	0
6	Start_Lng	0
7	End_Lat	3402762
8	End_Lng	3402762
9	Distance	0
10	Description	5
11	Street	10869
12	City	253
13	County	0
14	State	0
15	Zipcode	1915
16	Country	0
17	Timezone	7808
18	Airport_Code	22635
19	Weather_Timestamp	120228
20	Temperature	163853
21	Wind_Chill	1999019
22	Humidity	174144
23	Pressure	140679
24	Visibility	177098
25	Wind_Direction	175206
26	Wind_Speed	571233
27	Precipitation	2203586
28	Weather_Condition	173459
29	Amenity	0
30	Bump	0
31	Crossing	0
32	Give_Way	0
33	Junction	0
34	No_Exit	0
35	Railway	0
36	Roundabout	0
37	Station	0
38	Stop	0
39	Traffic_Calming	0
40	Traffic_Signal	0
41	Turning_Loop	0
42	Sunrise_Sunset	23246
43	Civil_Twilight	23246
44	Nautical_Twilight	23246
45	Astronomical_Twilight	23246
46	Year	120228

From the above output , we can observe that there are lot of missing values for many attributes. We are going it handle these missing values as per our requirements of data analysis of this project.
Following are the actions taken for data cleaning & feature engineering -

- **Renaming columns**
  We have renamed specific columns of our data frame by removing their respective measurement units . The columns include "Visibility(mi)", "Wind_Speed(mph)",

"Distance(mi)", "Temperature(F)", "Wind_Chill(F)", "Humidity(%)", "Pressure(in)", and "Precipitation(in)". Renaming these columns provides more meaningful and concise names for better data interpretation and analysis.

- **Extracting the YEAR from the Weather timestamp column**
  Using PySpark's year function to extract the year from the Weather timestamp column in a DataFrame. It adds a new column named 'Year' to our DataFrame using the withColumn method. This Column will be significant for our analysis of the data.

- **Removing the missing values for the below-mentioned Attributes in the dataset**

```
["City", "Zipcode", "Weather_Condition" ,"Temperature", "Humidity", "Pressure",
 "Visibility", "Wind_Speed", "Year" , "Sunrise_Sunset" ,"Civil_Twilight" ,
 "Nautical_Twilight" , "Astronomical_Twilight" ]
```

Using the dropna method on the PySpark DataFrame - df_pyspark , we have removed rows with missing values for the above-mentioned attributes as handling them would not add any significance to our data analysis of the accidents and allow us to work on a cleaner data thus enabling more accurate and meaningful insights to be derived from the data.

- **Replacing column values**
  The missing values for Precipitation column were replaced with “0” to ensure that data is consistent throughout the column. Replaced the missing values for Description column with “No Comments” to ensure data completeness and to provide clear information that no data was provided for such accidents.
  In a similar manner , we have handled the missing values for Street column by replacing it with “No Street data”.

- **Dropping the ID column**
  As we can see from the dataset, column – **ID** is not required or significant for our analysis, hence we have dropped it.

- **Converting the Start_time column to timestamp format**
  Converting the "Start_Time" column of our dataframe to timestamp format using the `to_timestamp()` function. This transformation is useful as the "Start_Time" column is originally stored as a string or in a different date/time format, and hence we standardized it to a timestamp format. By converting the column to a timestamp, we can perform various time-based operations and analyses on the data, such as filtering, grouping, and calculating time durations.

- **Extracting the Month of year , Day of week and Hour day from Start_time column**
  Using the date_format function , we have extracted the month of the year , day of week and hour of the day from Start_time column. These additional columns provide more granular information about the timing of each accident in the dataset and can be useful for our time-based analysis and insights.

DESCRIPTIVE CHARACTERISTICS OF THE DATASET

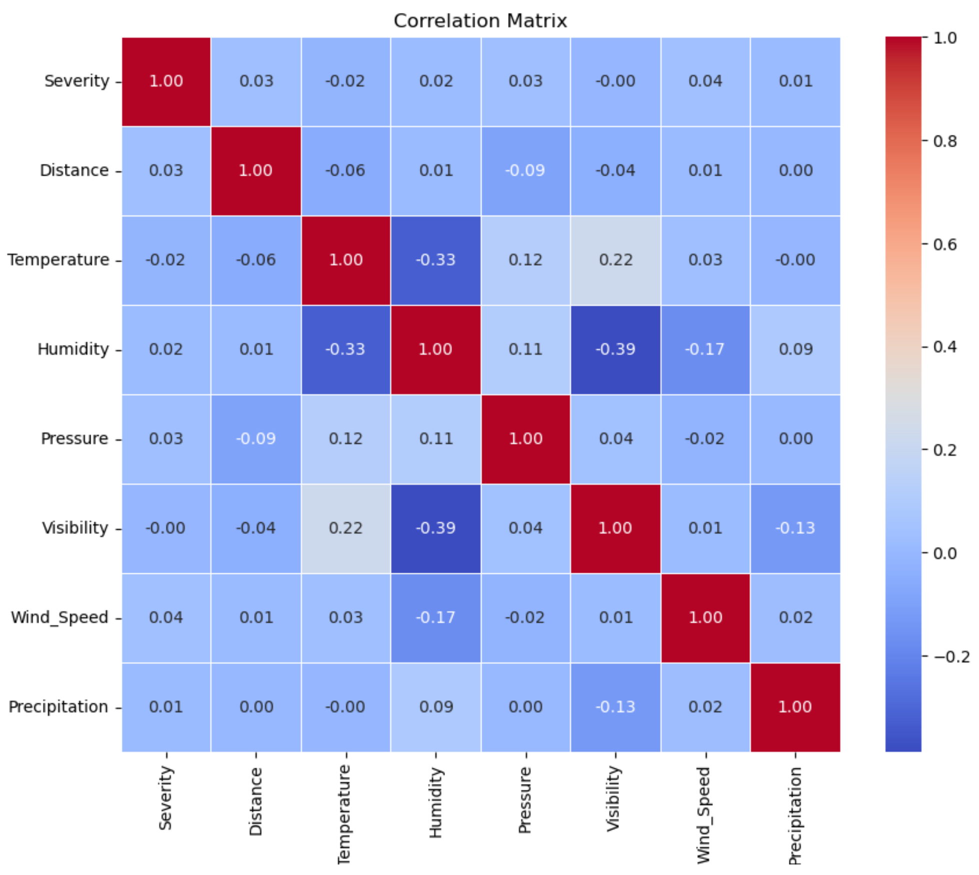
```
+-------+------------------+------------------+-----------------+------------------+------------------+-------------
-----+------------------+-------------------+
|summary|          Severity|          Distance|      Temperature|          Humidity|          Pressure|        Visib
ility|        Wind_Speed|      Precipitation|
+-------+------------------+------------------+-----------------+------------------+------------------+-------------
-----+------------------+-------------------+
|  count|           7061839|           7061839|          7061839|           7061839|           7061839|           70
61839|           7061839|            7061839|
|    max|                 4|            99.952|             99.9|              99.0|               9.9|
98.0|              99.0|               9.99|
|   mean| 2.202510422568399|0.5686278966535171|61.86914963651785| 64.41364154577866| 29.51408396169856| 9.1069603031
16535|    7.691038623226|0.00590666963661951|
|    min|                 1|               0.0|             -0.0|               1.0|               0.0|
0.0|               0.0|                  0|
| stddev|0.48026413786068817| 1.763447440866646| 19.0415499151011|22.758765030268385|1.0152123045265642|2.64188181619
84317|5.4133171640463935|0.07482367052713206|
+-------+------------------+------------------+-----------------+------------------+------------------+-------------
-----+------------------+-------------------+
```

The above displays the statistical information for several numerical columns in the DataFrame. The columns included are "Severity", "Distance", "Temperature", "Humidity", "Pressure", "Visibility", "Wind_Speed", and "Precipitation". The statistical information provides insights such as count, mean, standard deviation, minimum, maximum, and quartiles for each column, giving an overview of the data distribution and variability.

The key statistical information for the numerical columns in the DataFrame is as follows:

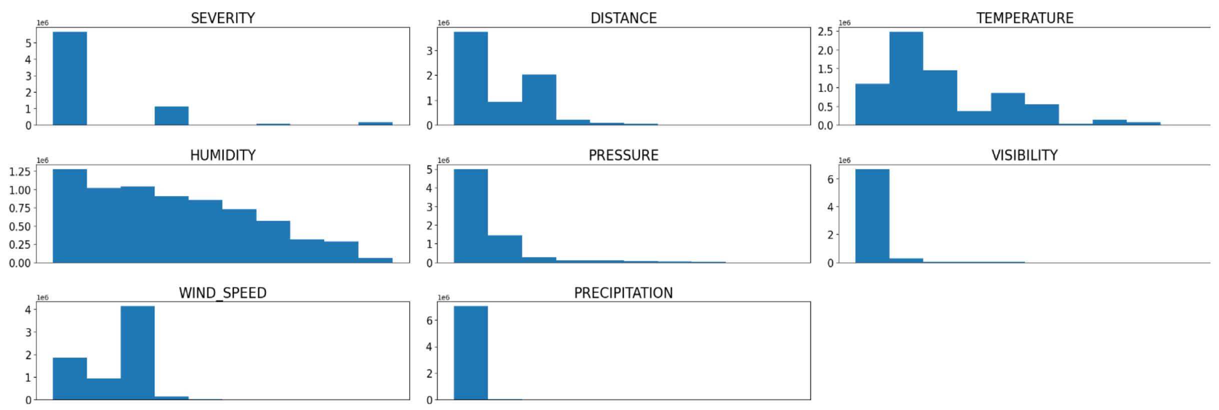
- "Severity": The count is 7,061,839. The mean is approximately 2.20, with a standard deviation of 0.48. The minimum severity is 1, and the maximum severity is 4.
- "Distance": The count is 7,061,839. The mean distance is approximately 0.57, with a standard deviation of 1.76. The minimum distance is 0.0, and the maximum distance is 99.952.
- "Temperature": The count is 7,061,839. The mean temperature is approximately 61.87, with a standard deviation of 19.04. The minimum temperature is -0.0, and the maximum temperature is 99.9.
- "Humidity": The count is 7,061,839. The mean humidity is approximately 64.41, with a standard deviation of 22.76. The minimum humidity is 1.0, and the maximum humidity is 99.0.
- "Pressure": The count is 7,061,839. The mean pressure is approximately 29.51, with a standard deviation of 1.02. The minimum pressure is 0.0, and the maximum pressure is 9.9.
- "Visibility": The count is 7,061,839. The mean visibility is approximately 9.11, with a standard deviation of 2.64. The minimum visibility is 0.0, and the maximum visibility is 98.0.
- "Wind_Speed": The count is 7,061,839. The mean wind speed is approximately 7.69, with a standard deviation of 5.41. The minimum wind speed is 0.0, and the maximum wind speed is 99.0.
- "Precipitation": The count is 7,061,839. The mean precipitation is approximately 0.0059, with a standard deviation of 0.0748. The minimum precipitation is 0, and the maximum precipitation is 9.99.

DATA VISUALIZATIONS



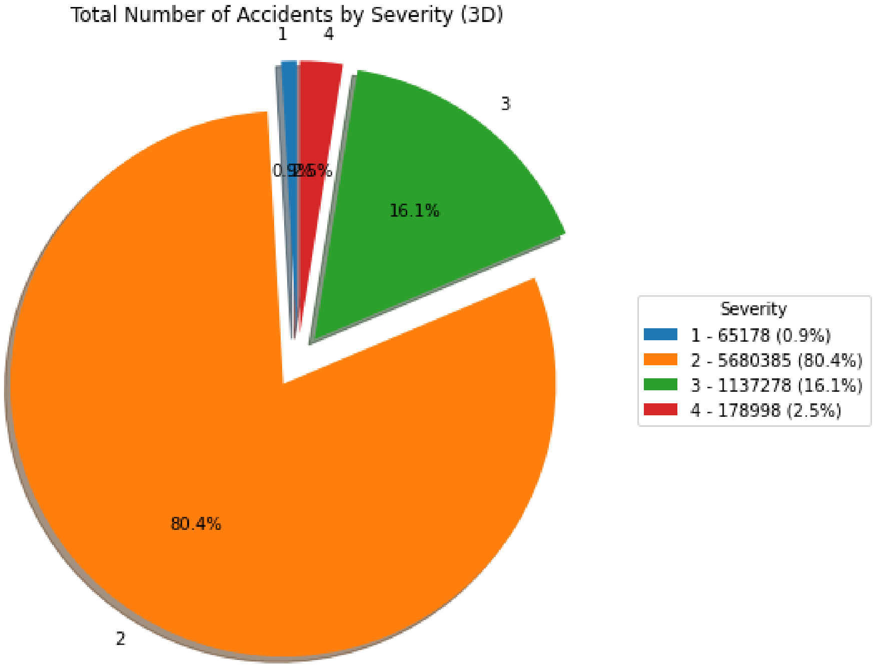
- There is a moderate negative correlation (-0.331) between humidity and temperature. This suggests that as humidity increases, temperature tends to decrease. It indicates an inverse relationship between these two variables.
- There is a moderate negative correlation (-0.387) between visibility and humidity. This indicates that as humidity increases, visibility tends to decrease. It implies that higher humidity levels are associated with reduced visibility conditions.
- There is a relatively moderate positive correlation (0.040) between wind speed and severity. Although the correlation is not very strong, it suggests that higher wind speeds might have a slight positive association with more severe accidents.
- The correlation between Severity and Distance is relatively low (0.033), indicating a weak positive relationship. This suggests that there is only a slight tendency for accidents with higher severity to occur at longer distances.

Distribution of Features

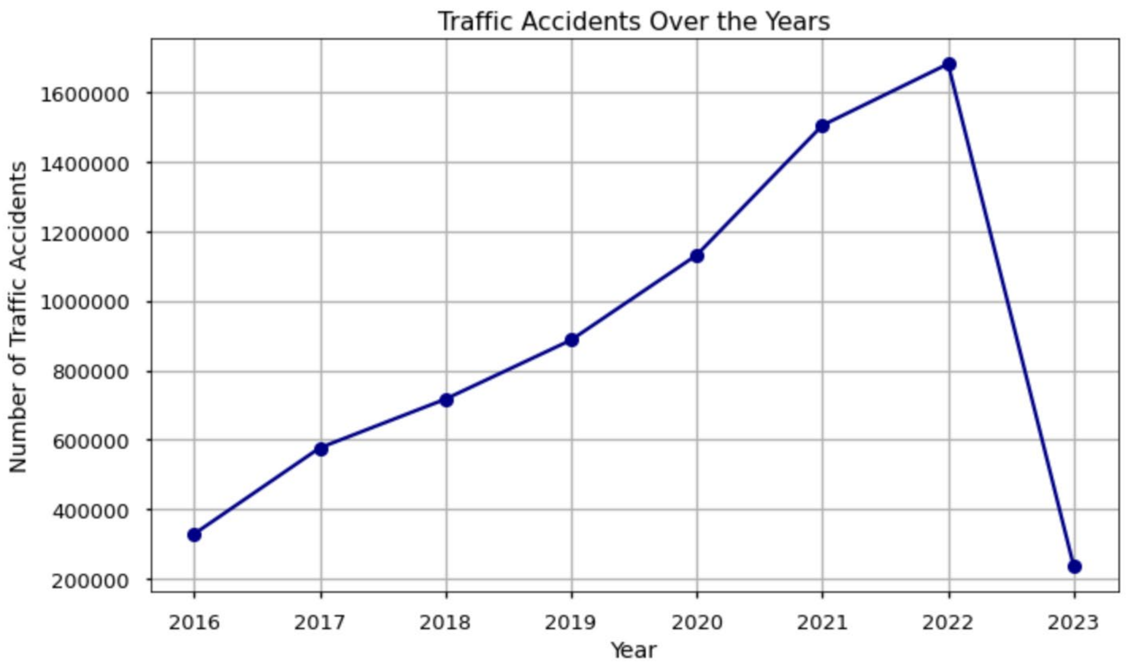


Analyzing multiple histograms for the numerical variables in the dataset provided us insights into their distributions and patterns.For example, the severity variable is skewed to the right, meaning that there are more accidents with a severity of 1 or 2 than there are accidents with a severity of 4 or 5.

Total Number of Accidents by Severity (3D)

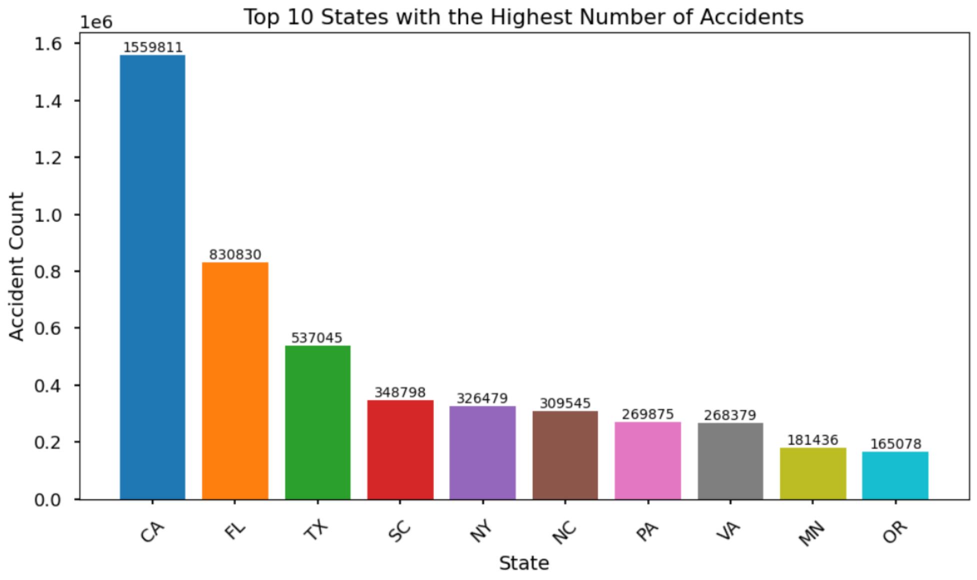


The above pie chart provides a clear overview of the distribution of accident severities in the dataset. Out of all the recorded road accidents in the dataset, approximately 80% of them resulted in a moderate impact, which is classified by Severity-2.

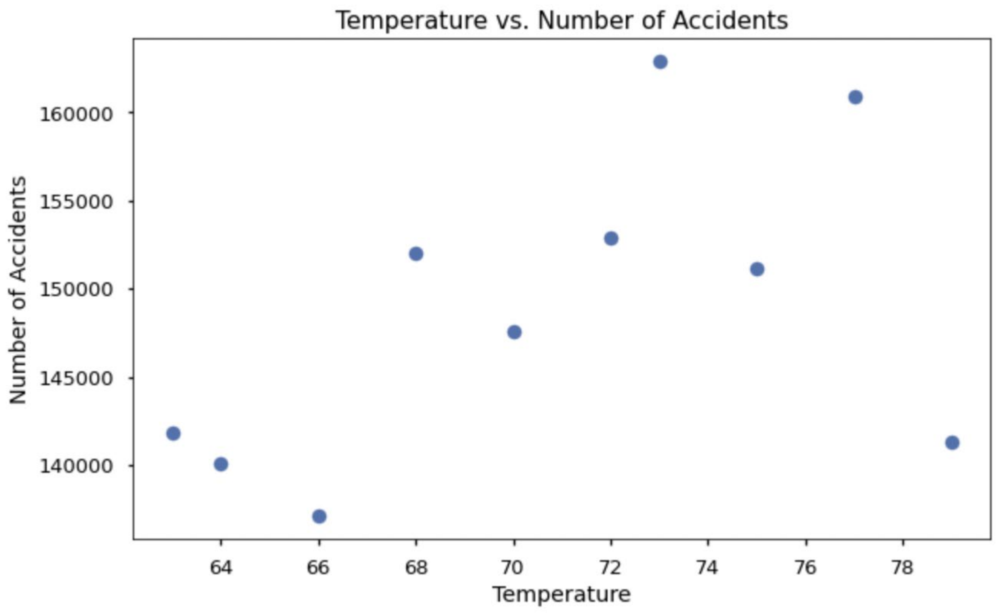


By plotting a line graph between the number of accidents against the years, we observed the following trends:

- Increase in accidents from 2016 to 2022: The line graph shows a general upward trend in the number of accidents over this period.
- Most accidents in 2022: The line reaches its peak in 2022, suggesting that this year had the highest number of recorded accidents during the given time frame. This could be due to various factors such as population growth, increased traffic volume, changes in driving behavior, or improvements in accident reporting and data collection.
- Sudden drop from 2022 to 2023: Following the peak in 2022, the line shows a sudden drop in the number of accidents in 2023.



The graph shows that California has the most accidents, followed by Texas and Florida. These three states are also the most populous states in the United States, so it is not surprising that they have the most accidents. These states also have high levels of traffic congestion, which can be a contributing factors to the high number of accidents.



The graph shows fluctuations in the range of 64-78 degrees Fahrenheit and these fluctuations may be due to a number of factors, including traffic congestion, driver behavior, and weather conditions. The number of accidents peaks at around 73degrees Fahrenheit.

CONCLUSION

The extensive data analysis performed on the US Accidents dataset offers numerous crucial insights into the prevailing problem of road accidents. The investigation focused on a range of contributing factors, such as weather conditions, geographical aspects, and temporal patterns, with the overarching aim of identifying actionable interventions to curb the escalating issue of traffic accidents.

The exploratory analysis clearly showed a marked increase in accidents from 2016 to 2022, with 2022 recording the highest number of accidents. This upward trend in accidents could be attributed to multiple factors, including population growth, changes in driving behavior, traffic congestion, and improved accident reporting. Interestingly, there was a sudden dip in the number of accidents in 2023, warranting further investigation to understand its causes.

States with high population density, namely California, Texas, and Florida, reported the highest number of accidents, which is to be expected given the increased traffic volume in these regions. These findings suggest that tailored interventions in these areas could significantly reduce the overall number of traffic accidents in the country.

Our investigation into the weather impact on accidents shows that visibility and temperature play a considerable role. Specifically, higher humidity levels, which often lead to decreased visibility, were found to correlate with an increased number of accidents. Moreover, most accidents occurred when the temperature was around 73 degrees Fahrenheit, indicating that weather conditions significantly influence road safety.

We also noted a positive correlation, albeit slight, between wind speed and severity of accidents. This correlation suggests that adverse weather conditions, such as high wind speeds, can potentially lead to more severe accidents, thus implying the necessity of comprehensive weather advisories and efficient traffic management during such situations.

The analysis of the temporal aspects revealed that the time of year, week, day, and even the hour can impact the frequency of accidents. This understanding underscores the importance of disseminating timely information about peak accident periods to the public, which could potentially help in accident prevention.

From an infrastructure perspective, the presence of traffic signals, stop signs, and pedestrian crossings were also identified as potential factors impacting the severity of accidents. Therefore, their strategic placement and regular maintenance could prove instrumental in mitigating road accidents.

Lastly, it is worth noting that the dataset had a considerable amount of missing data. Although appropriate data cleaning and feature engineering techniques were employed to address this issue, the presence of such missing data underlines the need for more robust and thorough data collection methods.

In summary, this detailed analysis of the US Accidents dataset serves as a significant stepping stone towards understanding and tackling the rampant issue of road accidents. The insights gleaned from the data underscore the multifaceted nature of the problem and emphasize the need for concerted, multi-pronged efforts in addressing it. These could include enhancing traffic infrastructure, refining data collection methods, leveraging weather data for improved traffic management, and focusing on accident-prone states and periods. The ultimate goal is to foster safer road conditions, thereby saving numerous lives and enhancing public safety.

APPENDIX

```
import pandas as pd
import numpy as n
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql.functions import countDistinct, col, sum, when, coalesce, lit, date_format,
to_timestamp, hour, dayofweek
from pyspark.sql import functions as F
import pandas as pd
import numpy as n
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql.functions import countDistinct, col, sum, when, coalesce, lit, date_format,
to_timestamp, hour, dayofweek
from pyspark.sql import functions as F

# Import Spark Session #
from pyspark.sql import SparkSession

# Start Spark Connection #
spark = SparkSession.builder.appName('ALY6110Project').getOrCreate()
spark

#Load data#
path ='C:/Users/14086/Downloads/accidents/accidents.csv'
df_pyspark=spark.read.csv (path , header=True)

type(df_pyspark)
num_rows = df_pyspark.count()
num_cols = len(df_pyspark.columns)

print("Number of rows: ", num_rows)
print("Number of columns: ", num_cols)
df_pyspark.printSchema()
df_pyspark.show(2)

# checking the head 10 records of the data #
df_pyspark.limit(10).toPandas().T

# Renaming columns #

# Create the mapping dictionary
mapping = {
   "Visibility(mi)": "Visibility",
   "Wind_Speed(mph)": "Wind_Speed",
   "Distance(mi)" :  "Distance",
   "Temperature(F)" : "Temperature",
   "Wind_Chill(F)" :  "Wind_Chill",
   "Humidity(%)" :"Humidity",
   "Pressure(in)" :"Pressure",
    "Visibility(mi)" : "Visibility",
   "Wind_Speed(mph)" : "Wind_Speed",
   "Precipitation(in)" : "Precipitation"

}

# Select and rename columns using the mapping dictionary
df_pyspark = df_pyspark.select([col(c).alias(mapping.get(c, c)) for c in df_pyspark.columns])
# Extracting the year from the timestamp

from pyspark.sql.functions import year

# Assuming 'df' is the DataFrame containing the 'Weather_Timestamp' column
df_pyspark = df_pyspark.withColumn('Year', year(df_pyspark['Weather_Timestamp']))

# Calculate the count of missing values for each column
missing_values = df_pyspark.select(*[sum(col(c).isNull().cast("int")).alias(c) for c in
df_pyspark.columns])

# Convert the result to a Pandas DataFrame
df_missing_values = pd.DataFrame(missing_values.first().asDict().items(),
columns=["Column", "Missing_Values"])
df_missing_values

# Specify the columns with missing values and then remove them
columns_with_missing = ["City", "Zipcode", "Weather_Condition" ,"Temperature",
"Humidity", "Pressure",
                    "Visibility", "Wind_Speed", "Year" , "Sunrise_Sunset" ,"Civil_Twilight"
,"Nautical_Twilight" , "Astronomical_Twilight" ]

# Remove rows with missing values in specified columns
df_pyspark_clean = df_pyspark.dropna(subset=columns_with_missing)

# Replacing missing values for Precipitation with 0#
df_pyspark_clean = df_pyspark_clean.withColumn('Precipitation',
when(df_pyspark_clean.Precipitation.isNull(), 0).otherwise(df_pyspark_clean.Precipitation))

# Replacing missing values for Description with "No comments"#
```

```
df_pyspark_clean = df_pyspark_clean.withColumn('Description',
when(df_pyspark_clean.Description.isNull(), "No
Comments").otherwise(df_pyspark_clean.Description))

# Replacing missing values for street with "No street data"#
df_pyspark_clean = df_pyspark_clean.withColumn('Street',
when(df_pyspark_clean.Street.isNull(), "No Street
data").otherwise(df_pyspark_clean.Street))

# Calculate the count of missing values for each column AFTER CLEANING THE DATA #
missing_values_clean = df_pyspark_clean.select(*[sum(col(c).isNull().cast("int")).alias(c)
for c in df_pyspark_clean.columns])

# Convert the result to a Pandas DataFrame
df_missing_values_clean = pd.DataFrame(missing_values_clean.first().asDict().items(),
columns=["Column", "Missing_Values"])
df_missing_values_clean

# Drop column ID
df_pyspark_clean = df_pyspark_clean.drop("ID")

# Checking total no of records for cleaned dataset#
df_pyspark_clean.count()

# checking the statistical info for each numerical column #
df_pyspark_clean.describe("Severity","Distance","Temperature","Humidity","Pressure","Visi
bility","Wind_Speed","Precipitation").show()

from pyspark.sql.functions import to_timestamp, date_format, col

df_pyspark_clean = df_pyspark_clean.withColumn("Start_Time",
to_timestamp(col("Start_Time")))
df_pyspark_clean = df_pyspark_clean.withColumn("month_of_year",
date_format(col("Start_Time"), "MMMM"))
df_pyspark_clean = df_pyspark_clean.withColumn("day_of_week",
date_format(col("Start_Time"), "EEEE"))
df_pyspark_clean = df_pyspark_clean.withColumn("hour_day",
date_format(col("Start_Time"), "H"))

df_pyspark_clean.printSchema()
# checking the head 10 records of the data #
df_pyspark_clean.limit(10).toPandas().T
from pyspark.sql.functions import year, month, hour, dayofweek

# Subset the data based on the year
data_2016 = df_pyspark_clean.filter(year('Start_Time') == 2016)
```

```
data_2017 = df_pyspark_clean.filter(year('Start_Time') == 2017)
data_2018 = df_pyspark_clean.filter(year('Start_Time') == 2018)
data_2019 = df_pyspark_clean.filter(year('Start_Time') == 2019)
data_2020 = df_pyspark_clean.filter(year('Start_Time') == 2020)
data_2021 = df_pyspark_clean.filter(year('Start_Time') == 2021)
data_2022 = df_pyspark_clean.filter(year('Start_Time') == 2022)
data_2023 = df_pyspark_clean.filter(year('Start_Time') == 2023)

data_2016.limit(10).toPandas().T

# Execute the SQL query for showin the total accidents by Year #
# Register the table as a temporary view

df_pyspark_clean.createOrReplaceTempView("accidents_year")

acc_year = spark.sql("SELECT Year, COUNT(*) AS Accidents FROM accidents_year
WHERE Year IS NOT NULL GROUP BY Year ORDER BY Year")

# Show the results
acc_year.show()

## Bar plot for accidents over the years ##

# Convert the results to a Pandas DataFrame
from matplotlib import ticker
results_pd = acc_year.toPandas()

# Define the "tab10" color palette
color_palette = plt.get_cmap("Accent").colors

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(results_pd["Year"], results_pd["Accidents"], color=color_palette)
plt.xlabel("Year")
plt.ylabel("Number of Traffic Accidents")
plt.title("Traffic Accidents Over the Years")
plt.xticks(results_pd["Year"], results_pd["Year"].astype(str))  # Set x-axis labels as the full
year and rotate them by 45 degrees
plt.ticklabel_format(style='plain', axis='y')  # Display y-axis labels as actual values
plt.tight_layout()
plt.style.use("seaborn-talk")
plt.show()
# Checking different styles for plots ##

print(plt.style.available)
```

```
# Create a line graph
plt.figure(figsize=(10, 6))
plt.plot(results_pd["Year"], results_pd["Accidents"], marker='o', linestyle='-',
color='darkblue')
plt.xlabel("Year")
plt.ylabel("Number of Traffic Accidents")
plt.title("Traffic Accidents Over the Years")
plt.style.use("seaborn-talk")
plt.xticks(results_pd["Year"], results_pd["Year"].astype(str))  # Set x-axis labels as the full
year and rotate them by 45 degrees
plt.ticklabel_format(style='plain', axis='y')  # Display y-axis labels as actual values
plt.grid(True)
plt.tight_layout()
plt.show()
visibility = spark.sql("SELECT Visibility, COUNT(*) AS Accidents\
                    FROM accidents_year \
                    GROUP BY Visibility\
                    ORDER BY Accidents DESC").limit(10)
visibility.show()
## Scatter plot for Visibility vs Accidents ##

import matplotlib.pyplot as plt

# Convert the visibility DataFrame to Pandas for plotting
visibility_pd = visibility.toPandas()

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(visibility_pd['Visibility'], visibility_pd['Accidents'],color='darkblue')
plt.xlabel('Visibility')
plt.ylabel('Number of Accidents')
plt.title('Visibility vs Number of Accidents')
plt.ticklabel_format(style='plain', axis='y')  # Display y-axis labels as actual values
plt.show()

temperature = spark.sql("SELECT Temperature, COUNT(*) AS Accidents\
                    FROM accidents_year \
                    GROUP BY Temperature\
                    ORDER BY Accidents DESC").limit(10)
temperature.show()
import pandas as pd
import matplotlib.pyplot as plt
from pyspark.sql.types import FloatType


# Convert the "Temperature" column to float type
```

```
temperature_accidents = temperature.withColumn("Temperature",
col("Temperature").cast(FloatType()))

# Filter out null values
filtered_data = temperature_accidents.filter(col("Temperature").isNotNull())

# Convert PySpark DataFrame to Pandas DataFrame
pandas_data = filtered_data.toPandas()

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(pandas_data["Temperature"], pandas_data["Accidents"])
plt.xlabel("Temperature")
plt.ylabel("Number of Accidents")
plt.title("Temperature vs. Number of Accidents")
plt.ticklabel_format(style='plain', axis='y')  # Display y-axis labels as actual values
plt.show()

# Selecting the relevant columns for correlation analysis #

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation

selected_columns =
["Severity","Distance","Temperature","Humidity","Pressure","Visibility","Wind_Speed","Pr
ecipitation"]
selected_df = df_pyspark_clean.select(*selected_columns)

numeric_df = selected_df.select([col(column).cast("double").alias(column) for column in
selected_df.columns])

vector_assembler = VectorAssembler(inputCols=numeric_df.columns, outputCol="features")
assembled_df = vector_assembler.transform(numeric_df).select("features")

correlation_matrix = Correlation.corr(assembled_df, "features").head()
correlation_values = correlation_matrix[0].toArray().tolist()
correlation_df = pd.DataFrame(correlation_values, columns=numeric_df.columns,
index=numeric_df.columns)

# correlation matrix
print(correlation_df)
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_df, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

```
## HISTOGRAM ##

# Remove NA values from DataFrame
selected_df = selected_df.dropna()

# Plot distribution of features
fig = plt.figure(figsize=(25, 15))
st = fig.suptitle("Distribution of Features", fontsize=50, verticalalignment="center")
for col, num in zip(selected_df.columns, range(1, 20)):
    ax = fig.add_subplot(5, 3, num)
    ax.hist(selected_df.select(col).toPandas()[col])
    plt.grid(False)
    plt.yticks(fontsize=15)
    plt.title(col.upper(), fontsize=20)
    plt.xticks([])

plt.tight_layout()
st.set_y(0.95)
fig.subplots_adjust(top=0.85, hspace=0.4)
plt.show()

from pyspark.sql.functions import count

# Group the data by state and count the number of accidents
state_counts = df_pyspark_clean.groupBy('State').agg(count('*').alias('Accident_Count'))

# Order the results by state
state_counts = state_counts.orderBy('State')

# Show the result
state_counts.show()
from pyspark.sql.functions import count

# Group the data by state and count the number of accidents
state_counts = df_pyspark_clean.groupBy('State').agg(count('*').alias('Accident_Count'))

# Order the results by the number of accidents in descending order
state_counts = state_counts.orderBy('Accident_Count', ascending=False)

# Show the top 10 states
state_counts.show(10)
import matplotlib.pyplot as plt
import numpy as np

# Convert the PySpark DataFrame to Pandas DataFrame
state_counts_pd = state_counts.toPandas()
```

```
# Select the top 10 states
top_10_states = state_counts_pd.head(10)

# Create a colormap for different colors for each state
colors = plt.cm.get_cmap('tab10', len(top_10_states))

# Create the bar plot
plt.figure(figsize=(10, 6))
bars = plt.bar(top_10_states['State'],
top_10_states['Accident_Count'],color=colors(range(len(top_10_states))))
plt.xlabel('State')
plt.ylabel('Accident Count')
plt.title('Top 10 States with the Highest Number of Accidents')
plt.xticks(rotation=45)
plt.tight_layout()

# Add labels for each bar
for i, bar in enumerate(bars):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), str(int(bar.get_height())),
ha='center', va='bottom')
plt.show()

from pyspark.sql.functions import count

# Group the data by severity and count the number of accidents
severity_counts =
df_pyspark_clean.groupBy('Severity').agg(count('*').alias('Accident_Count'))

# Order the results by severity
severity_counts = severity_counts.orderBy('Severity')

# Show the result
severity_counts.show()

import matplotlib.pyplot as plt

# Convert the PySpark DataFrame to Pandas DataFrame
severity_counts_pd = severity_counts.toPandas()

# Create the 3D pie chart
plt.figure(figsize=(8, 6))
explode = [0.1] * len(severity_counts_pd)  # Optional: explode slices for emphasis

wedges, texts, autotexts = plt.pie(severity_counts_pd['Accident_Count'],
                                   labels=severity_counts_pd['Severity'],
                          autopct='%1.1f%%',  # Display values as percentages on the chart
                          startangle=90,
                          explode=explode,
                          shadow=True)

# Create the legend with labels and values
legend_labels = [f'{label} - {value} ({autotext.get_text()})' for label, value, autotext in
             zip(severity_counts_pd['Severity'], severity_counts_pd['Accident_Count'],
autotexts)]
plt.legend(wedges, legend_labels, loc='center left', bbox_to_anchor=(1, 0.5), title='Severity')

# Adjust the layout
plt.axis('equal')
plt.title('Total Number of Accidents by Severity (3D)')
plt.tight_layout()  # Ensures the legend is displayed correctly
plt.show()

import matplotlib.pyplot as plt

# Convert the PySpark DataFrame to Pandas DataFrame
severity_counts_pd = severity_counts.toPandas()

# Create the 3D pie chart
plt.figure(figsize=(8, 6))
explode = [0.1] * len(severity_counts_pd)  # Optional: explode slices for emphasis

wedges, texts, autotexts = plt.pie(severity_counts_pd['Accident_Count'],
                          labels=severity_counts_pd['Severity'],
                          autopct='%1.1f%%',  # Display values as percentages on the chart
                          startangle=90,
                          explode=explode,
                          shadow=True)

# Create the legend with labels and values
legend_labels = [f'{label} - {value} ({autotext.get_text()})' for label, value, autotext in
             zip(severity_counts_pd['Severity'], severity_counts_pd['Accident_Count'],
autotexts)]
plt.legend(wedges, legend_labels, loc='center left', bbox_to_anchor=(1, 0.5), title='Severity')

# Adjust the layout
plt.axis('equal')
plt.title('Total Number of Accidents by Severity (3D)')
plt.tight_layout()  # Ensures the legend is displayed correctly
plt.show()
```