



College of Professional Studies

Northeastern University San Jose

MPS Analytics

Course: ALY6110 – Data Management and Big Data

Assignment:

Module 6 Final Project - Report

Submitted on:

June 30th, 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 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 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 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	20	Temperature	163853
1	Source	0	21	Wind_Chill	1999019
2	Severity	0	22	Humidity	174144
3	Start_Time	0	23	Pressure	140679
4	End_Time	0	24	Visibility	177098
5	Start_Lat	0	25	Wind_Direction	175206
6	Start_Lng	0	26	Wind_Speed	571233
7	End_Lat	3402762	27	Precipitation	2203586
8	End_Lng	3402762	28	Weather_Condition	173459
9	Distance	0	29	Amenity	0
10	Description	5	30	Bump	0
11	Street	10869	31	Crossing	0
12	City	253	32	Give_Way	0
13	County	0	33	Junction	0
14	State	0	34	No_Exit	0
15	Zipcode	1915	35	Railway	0
16	Country	0	36	Roundabout	0
17	Timezone	7808	37	Station	0
18	Airport_Code	22635	38	Stop	0
19	Weather_Timestamp	120228	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 to 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

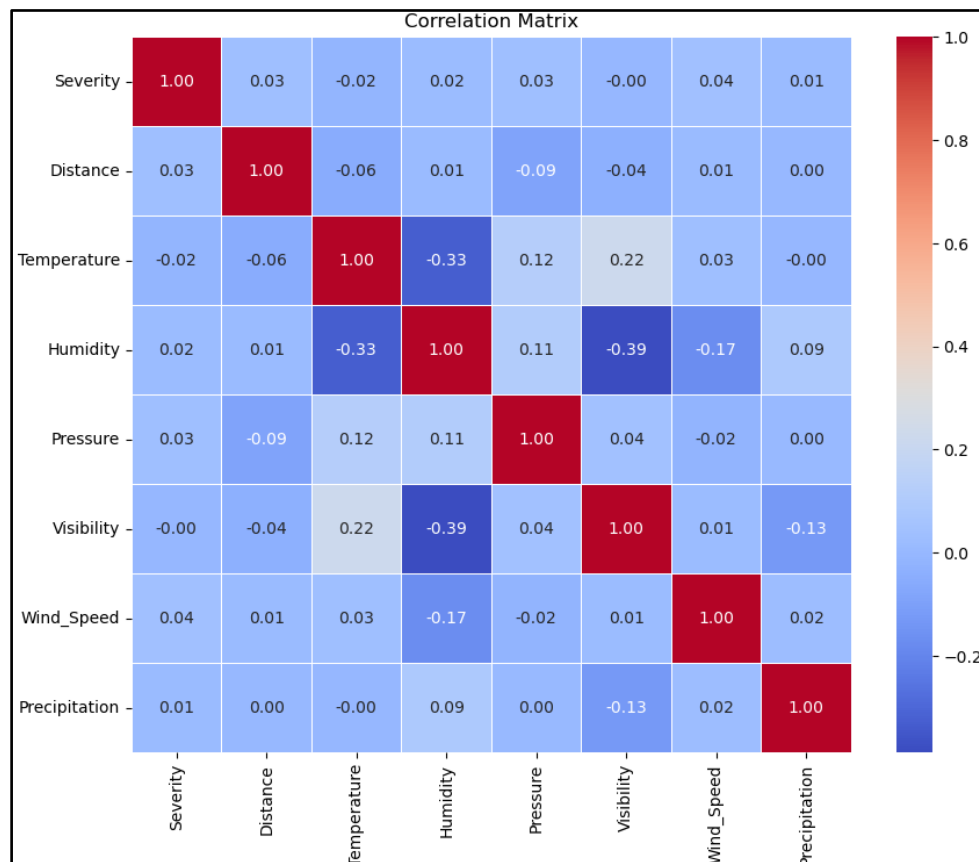
summary ility	Severity Wind_Speed	Distance Precipitation	Temperature	Humidity	Pressure	Visib
count	7061839	7061839	7061839	7061839	7061839	70
max	99.0	99.952	99.9	99.0	9.9	98.0
mean	2.202510422568399	0.5686278966535171	61.86914963651785	64.41364154577866	29.51408396169856	9.1069603031
min	0.0	0.0	-0.0	1.0	0.0	0.0
stddev	0.48026413786068817	1.763447440866646	19.0415499151011	22.758765030268385	1.0152123045265642	2.64188181619

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.

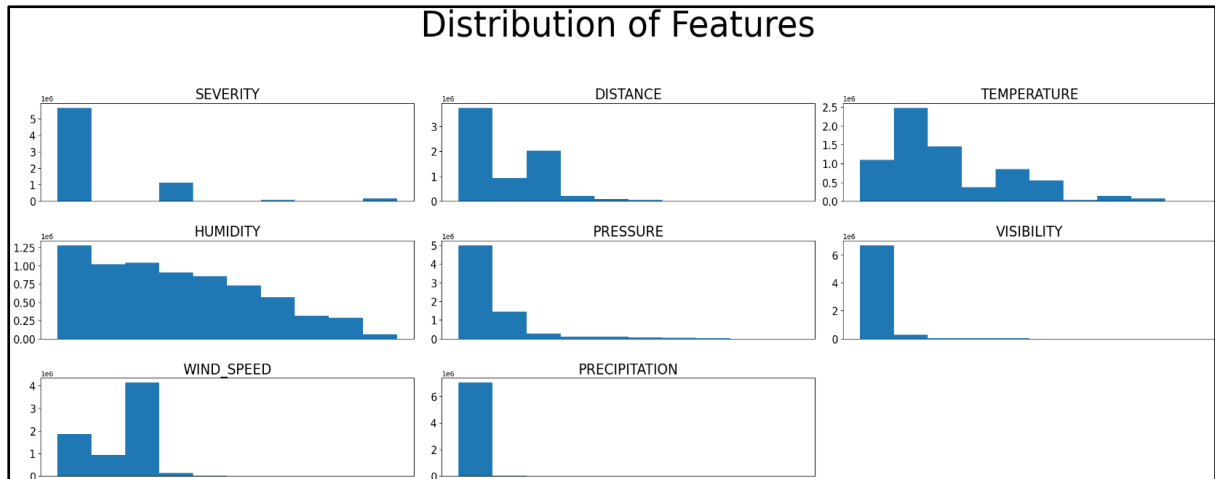
The key statistical information for the numerical columns in the DataFrame is as follows:

- "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.

BASIC EXPLORATORY DATA ANALYSIS

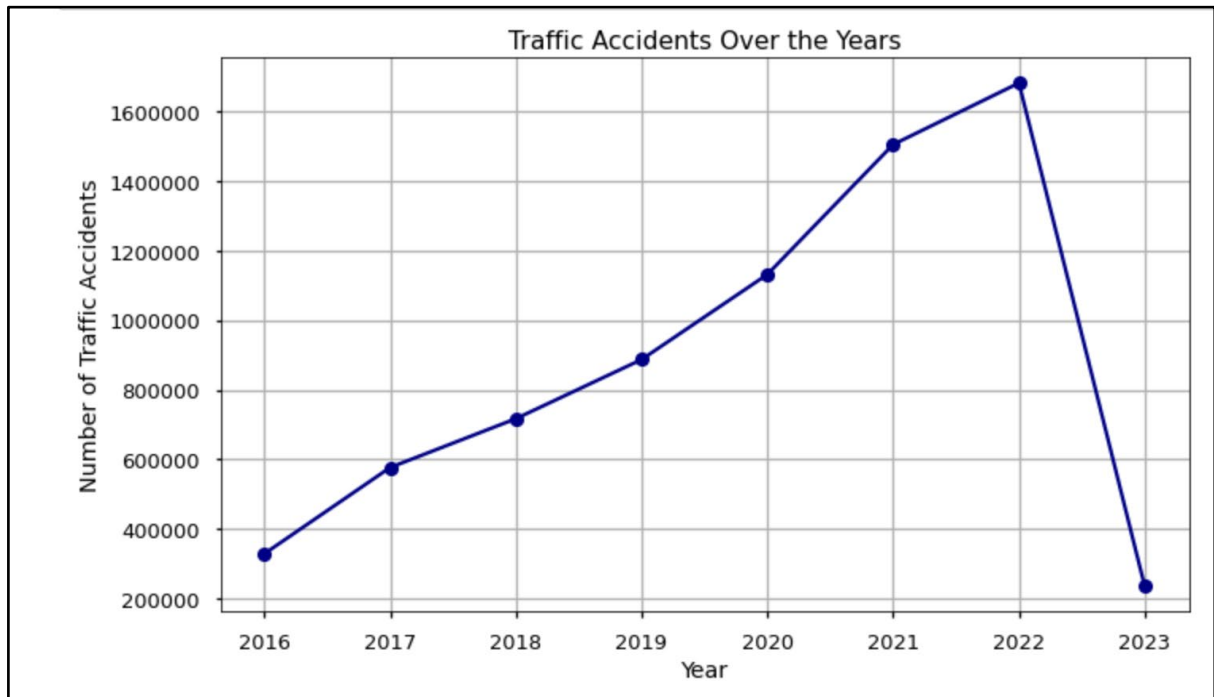


- There is a moderate negative correlation (-0.331) between humidity and temperature. This suggests that as humidity increases, temperature tends to decrease. It indicates an inverse relationship between these two variables.
- There is a moderate negative correlation (-0.387) between visibility and humidity. 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.



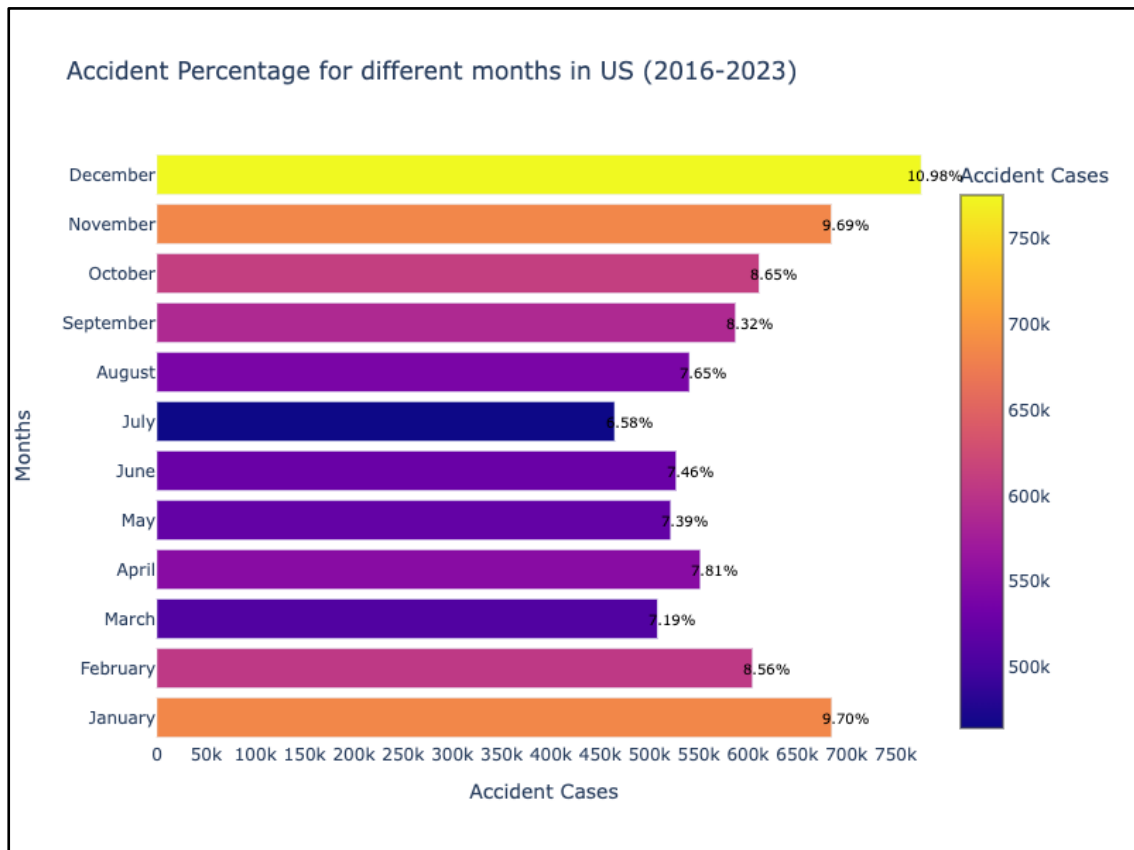
Analyzing multiple histograms for the numerical variables in the dataset provided us insights into their distributions and patterns. For example, the severity variable is skewed to the right, meaning that there are more accidents with a severity of 1 or 2 than there are accidents with a severity of 4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

TIME ANALYSIS



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.

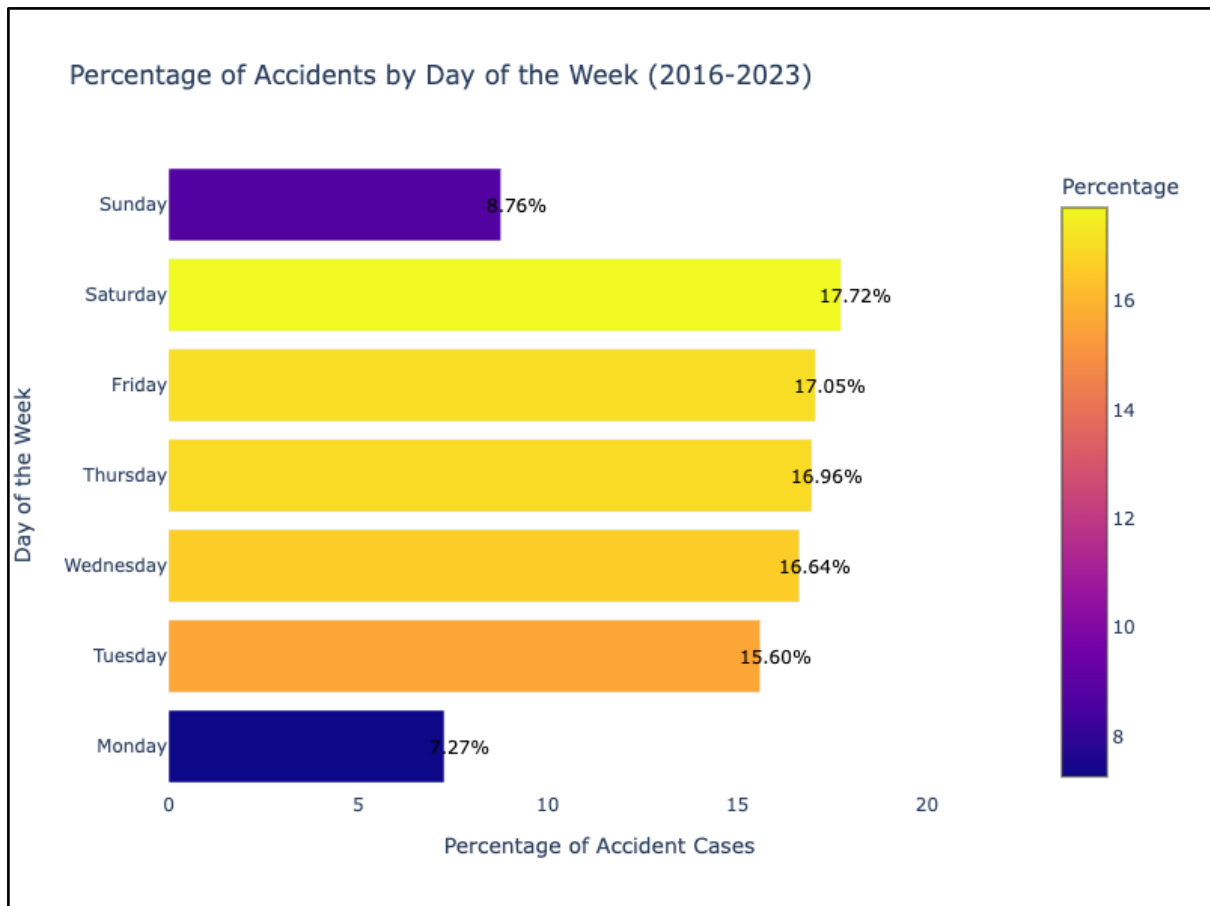


The above bargraph shows the percentage distribution of accidents by month in the US for the period 2016 - 2023 and below are some significant insights –

Peak in Accident Cases: The highest bar is observed in the month of December with **10.98%** followed by January with **9.70 %**, indicating that December had the highest number of recorded accidents during the given time frame. This could be attributed to various factors such as weather conditions, holiday travel, or other seasonal influences.

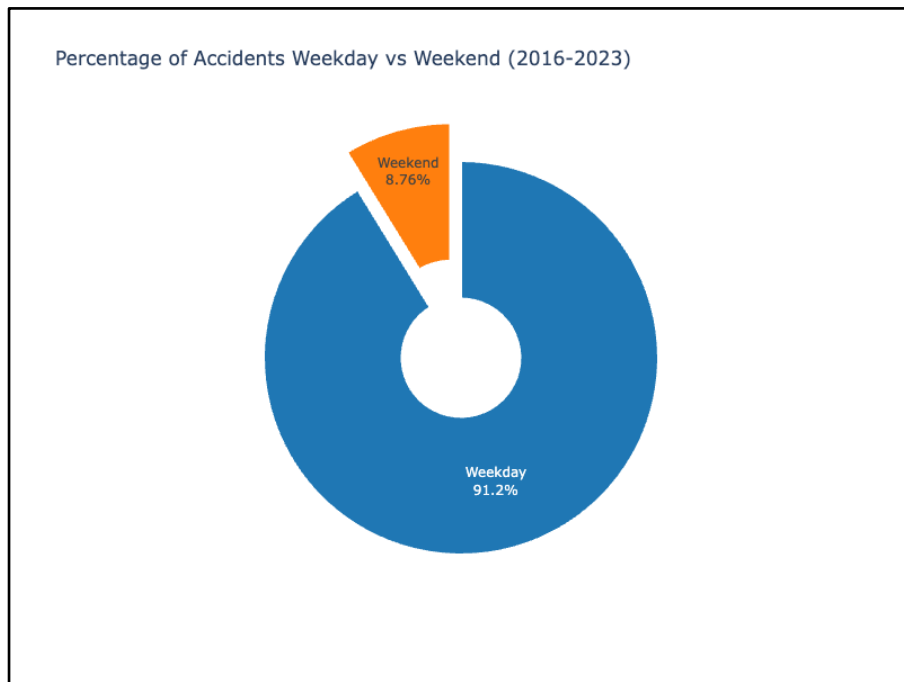
Consistent Accident Rates: From the graph, we can see that the accident rates are relatively consistent across the March ,April , May & June, with slight variations. This suggests that there are no significant seasonal patterns in the occurrence of accidents.

The lowest accidents rates are observed to be in the month of July with 6.58%



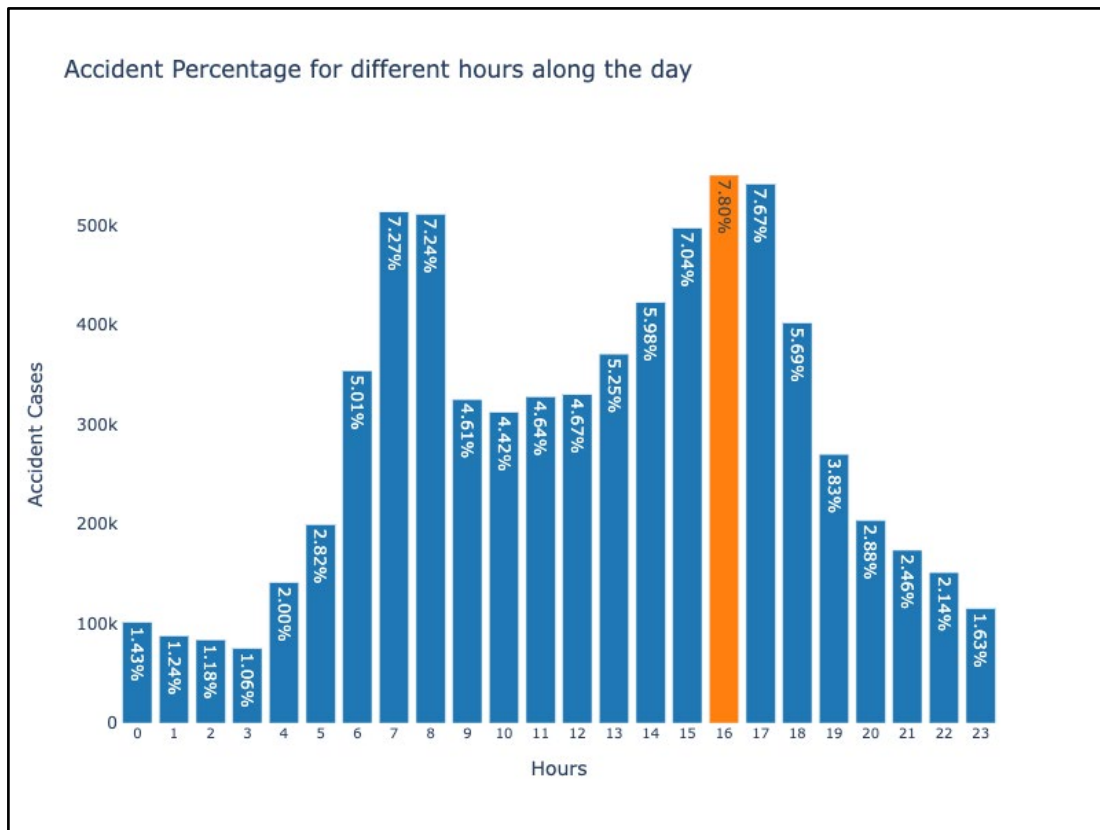
The above horizontal bar graph depicts the percentage of accidents by day of the week and following are the insights that can be observed:

- The graph shows that towards the end of the week - Thursday , Friday & Saturday have a increasing percentage of accidents compared to other days with the highest being on Friday. This suggests that these days experience higher traffic volume probably due to Weekend or potentially more hazardous driving conditions.
- The percentage of accidents decreases on Sunday and Monday having the lowest accident rates. This could be attributed to reduced commuting traffic, fewer work-related trips, or more cautious driving behavior during leisure time.



The above 3D donut chart depicts the distribution of accidents for Weekdays Vs Weekends and below are few insights –

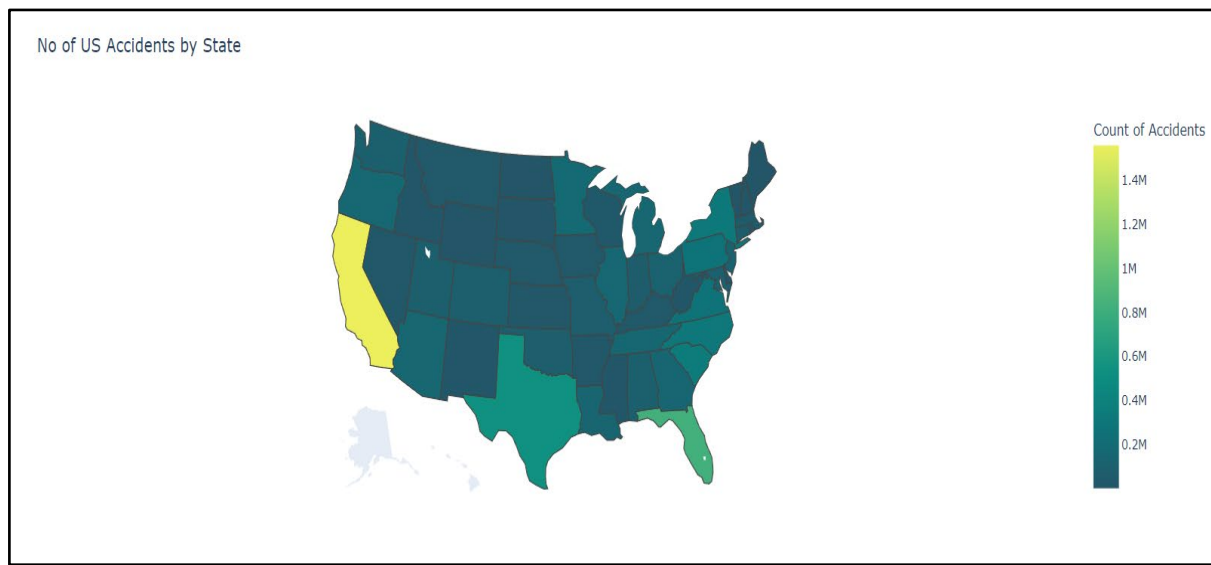
The graph shows that weekends, specifically Saturday and Sunday, have a lower percentage (only **8.76%**) of accidents compared to weekdays (**91.2%**). This suggests that weekends generally have fewer accidents, possibly due to reduced traffic volume or different driving patterns.



The above graph shows the percentage distribution of accidents for each hour of the day and below are some significant insights -

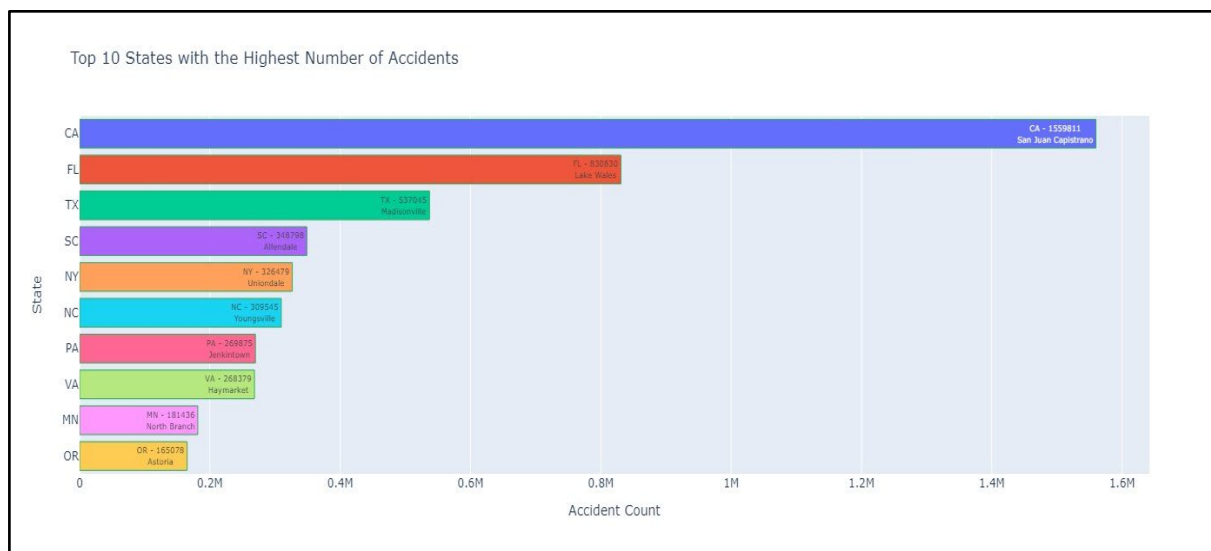
- Each bar represents an hour, ranging from 0 (midnight) to 23 (11 PM). The height of each bar represents the number of accident cases reported during that specific hour.
- Around 24% of the accidents occurred in between 6:00AM to 9:00AM with peak hours being 7AM and 8 AM.
- In evening, around 28% of the accidents occurred in between 3:00PM to 6:00PM.
- The hour with the highest percentage or the most-deadliest accident hour is 4:00PM implies the Evening Office-Returning Hours followed by 5:00 PM

LOCATION ANALYSIS



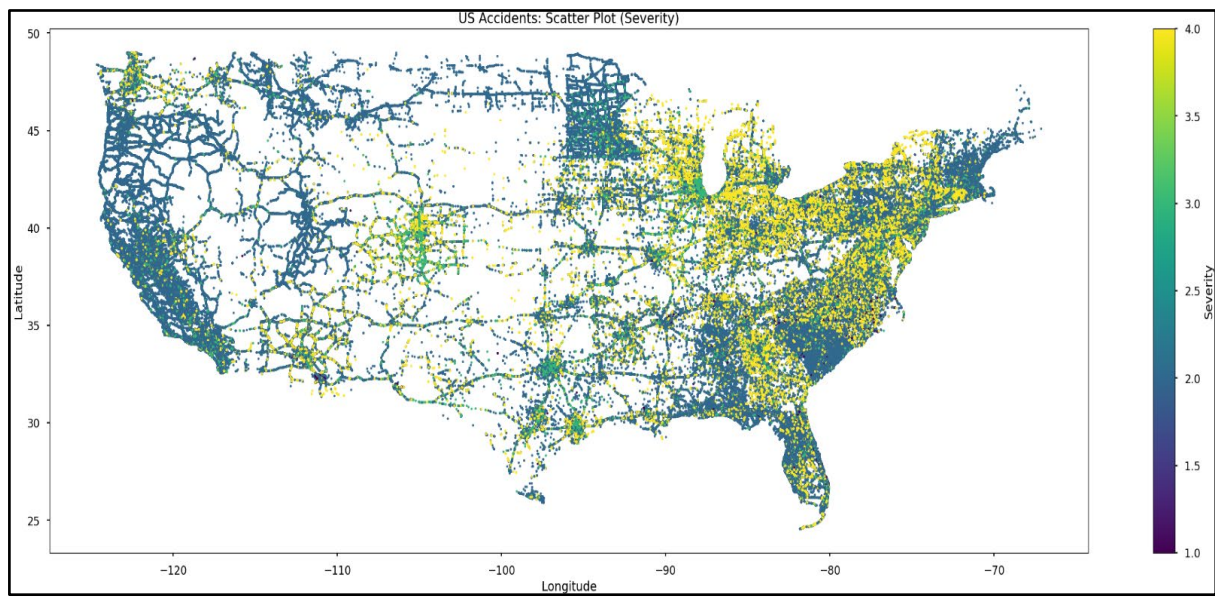
This map shows the number of accidents that occurred in each state and helps in identifying states with accident rates, allowing for geographical analysis and comparisons.

It can be observed 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.

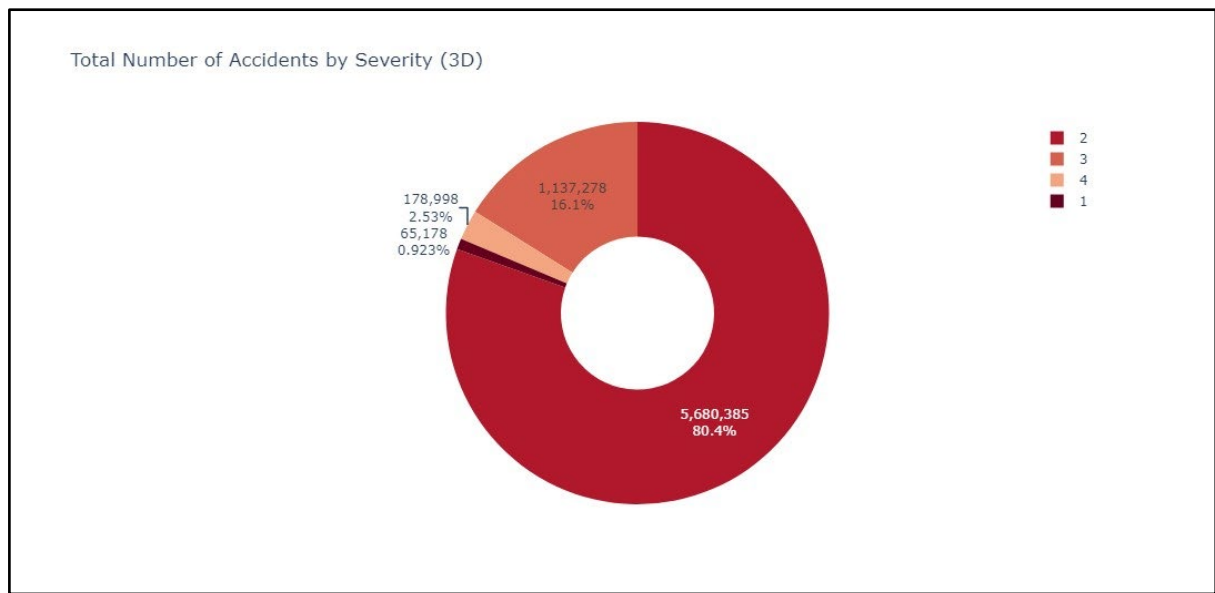


In the above barplot, we can see the cities with the most accidents in that particular state. San Juan Capistrano tops the list of accident-prone cities in California.

SEVERITY ANALYSIS

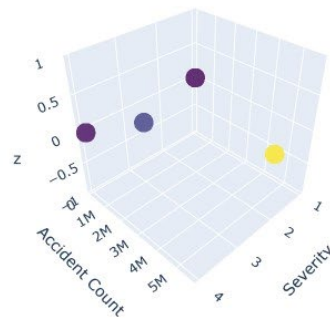


Here, we can see the distribution of accidents in the US based on their severity. One thing to notice is that coastal areas tend to have a higher concentration of accidents compared to non-coastal areas.

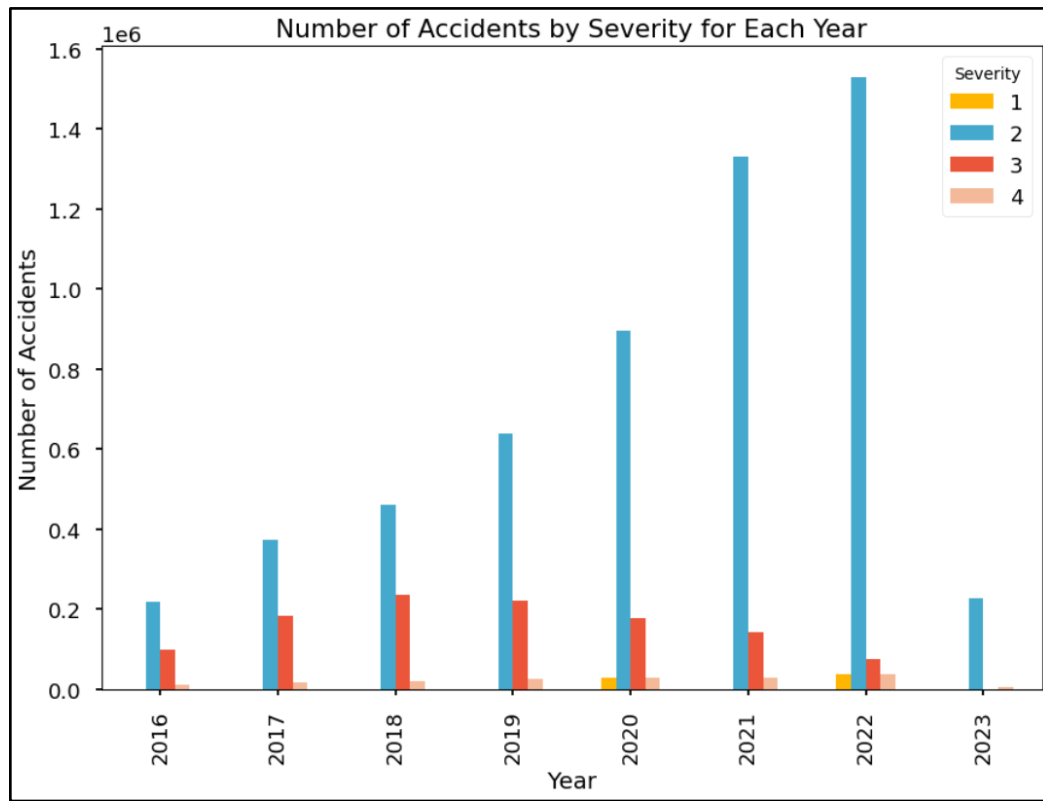


The above donut chart provides a clear overview of the distribution of accident severities in the dataset. Out of all the recorded road accidents in the dataset, approximately 80% of them resulted in a moderate impact, which is classified by Severity-2. Approximately 2.53% of road accidents resulted in a significantly severe impact, categorized as Severity-4.

Total Number of Accidents by Severity (3D Scatter Plot)



The data used for the scatter plot is derived from a table with two columns: "Severity" and "Accident_Count." Each row in the table represents a specific severity level and its corresponding count of accidents. The severity levels range from 1 to 4. In the 3D scatter plot, the severity levels are plotted on the x-axis, while the y-axis represents the count of accidents for each severity level. Each data point in the scatter plot is represented by a marker. The size of the markers is set to 10, and their color is determined by the accident count, following the "Viridis" color scale. The opacity of the markers is set to 0.8. To provide additional context, each marker in the scatter plot displays a text label representing the severity level it corresponds to. The plot is titled "Total Number of Accidents by Severity (3D Scatter Plot)," with the x-axis labeled as "Severity" and the y-axis labeled as "Accident Count." By analyzing the 3D scatter plot, one can gain insights into the relationship between severity levels and the corresponding count of accidents. It enables the identification of severity levels associated with high or low accident counts and facilitates the exploration of any patterns or trends within the data.



By examining the graph, we can draw several conclusions:

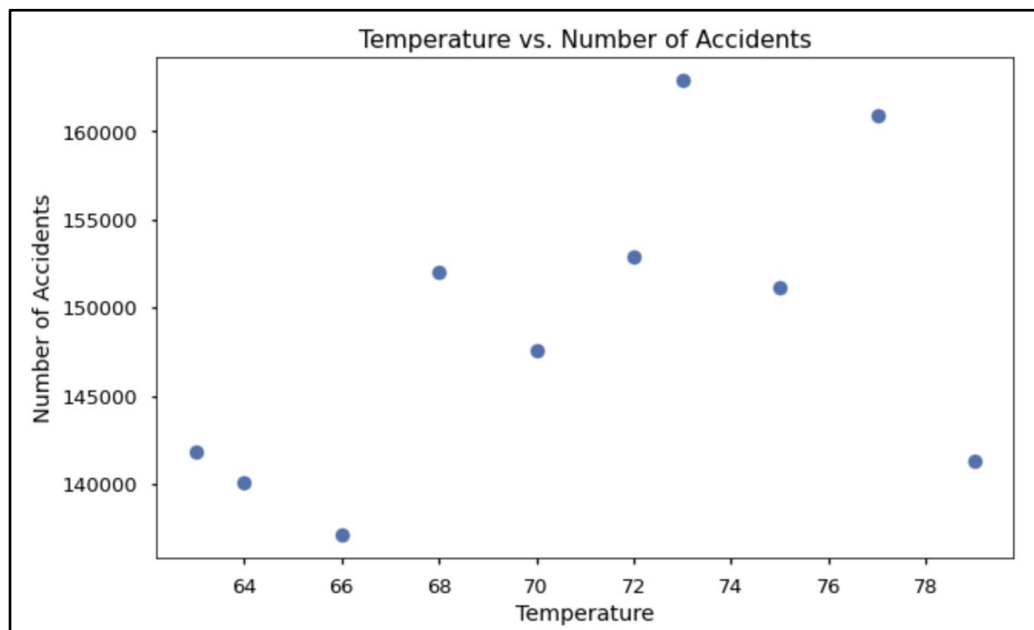
- The number of accidents tends to fluctuate across the years, indicating variations in road safety conditions.
- Severity 2 accidents consistently appear to be the most common, as they are consistently increasing across the years, except for a sudden drop in 2023.
- Severity 1 and Severity 4 accidents appear to have less frequent occurrences compared to Severity 2 and Severity 3 accidents.

DRIVING FACTORS ANALYSIS



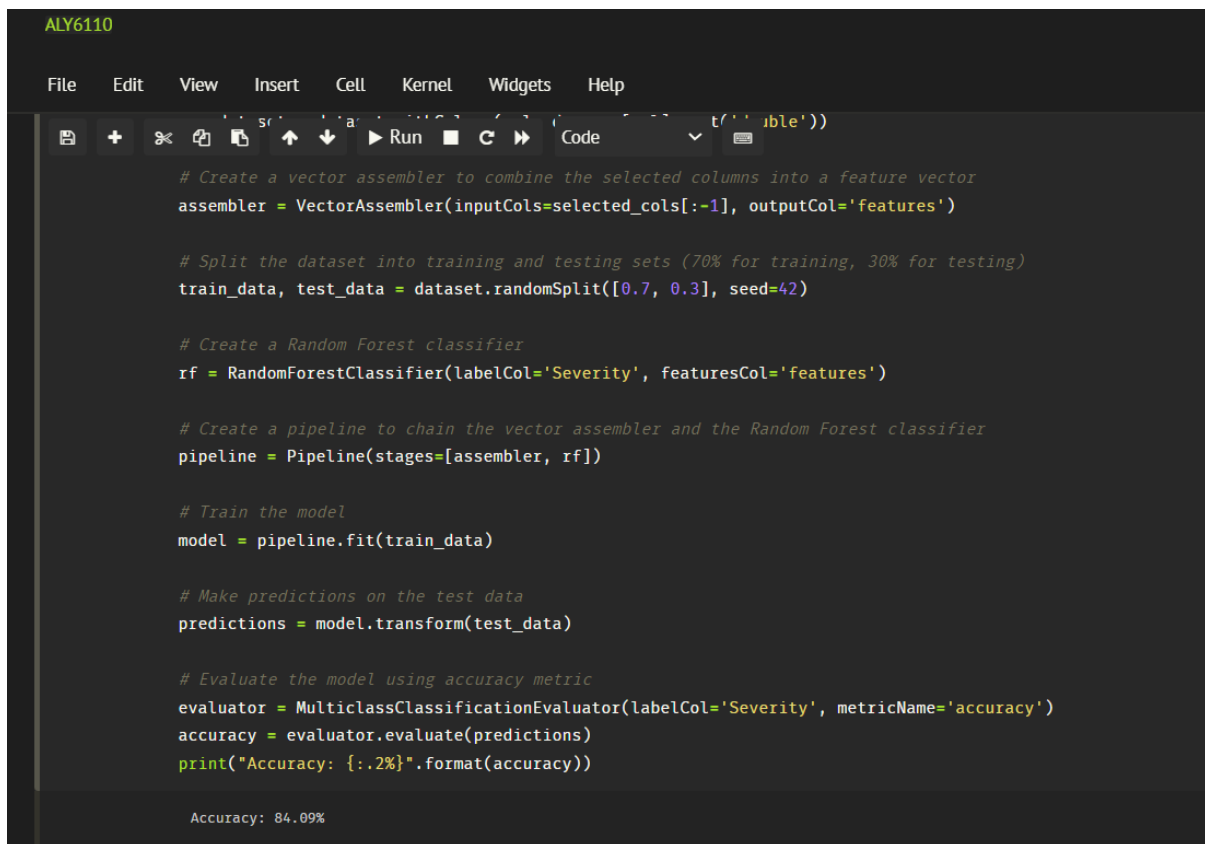
In the generated grid of pie charts, it can be observed that for all the driving factors, the category "False" is the majority. This implies that in the analyzed dataset, the occurrence of driving factors such as Amenity, Bump, Crossing, Give_Way, Junction, No_Exit, Railway, Roundabout, Station, Stop, Traffic_Calming, and Traffic_Signal is generally low. The pie

charts for each column visually represent the distribution of categories, with the "False" slice consistently occupying a larger portion compared to the "True" slice. This indicates that the majority of locations in the dataset do not have the corresponding driving factor present. This finding could potentially suggest that the analyzed dataset predominantly consists of locations without these driving factors. It may also indicate that the presence of these driving factors is relatively rare or less common in the context of the dataset under consideration. This information provides valuable insights into the prevalence and distribution of driving factors, highlighting the importance of focusing on the cases where the driving factors are present to understand their impact on various aspects related to transportation and road safety.



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 73 degrees Fahrenheit.

RANDOM FOREST MODEL



```
ALY6110

File Edit View Insert Cell Kernel Widgets Help

# Create a vector assembler to combine the selected columns into a feature vector
assembler = VectorAssembler(inputCols=selected_cols[:-1], outputCol='features')

# Split the dataset into training and testing sets (70% for training, 30% for testing)
train_data, test_data = dataset.randomSplit([0.7, 0.3], seed=42)

# Create a Random Forest classifier
rf = RandomForestClassifier(labelCol='Severity', featuresCol='features')

# Create a pipeline to chain the vector assembler and the Random Forest classifier
pipeline = Pipeline(stages=[assembler, rf])

# Train the model
model = pipeline.fit(train_data)

# Make predictions on the test data
predictions = model.transform(test_data)

# Evaluate the model using accuracy metric
evaluator = MulticlassClassificationEvaluator(labelCol='Severity', metricName='accuracy')
accuracy = evaluator.evaluate(predictions)
print("Accuracy: {:.2%}".format(accuracy))

Accuracy: 84.09%
```

We have applied the Random Forest model to predict the severity of the accidents. We selected columns like latitude, longitude, temperature, wind chill, humidity, pressure, visibility, wind speed, and the target variable, severity. In this, we selected 'Severity' column as the target variable and the remaining columns were combined into a feature vector using a VectorAssembler.

The dataset was split randomly into training and testing sets using a 70% training and 30% testing ratio.

Based on these results, the trained Random Forest classifier achieved an accuracy of **84.09%** in predicting the severity of incidents based on the selected features. This accuracy score indicates a relatively high level of correctness in the model's predictions.

CONCLUSION

The problem of traffic accidents in the United States is a significant concern, with a rising number of accidents and fatalities each year. To address this issue, a comprehensive analysis of the US Accidents dataset was conducted. The dataset, sourced from Kaggle, contains detailed information about traffic accidents across the country from February 2016 to March 2023.

The analysis focused on answering several key business questions related to traffic accidents, such as identifying the key factors contributing to accidents in specific cities or states, finding effective ways to mitigate accidents, examining the relationship between accident severity and factors like visibility, temperature, and precipitation, identifying states with the highest number of accidents, understanding the impact of weather on accidents, analyzing accident occurrence by time of the year, week, day of the week, and time of day, and investigating the relationship between accident severity and the presence of infrastructural features like traffic lights and pedestrian crossings.

The analysis utilized various methodologies, including data cleaning and preprocessing, feature engineering, descriptive analysis, correlation analysis, data visualization, temporal analysis, and geographical analysis. These methodologies provided a holistic understanding of the dataset and helped derive meaningful insights.

The data cleaning and preprocessing stage involved handling missing values, renaming columns for clarity, and converting the Start_time column to a timestamp format. Descriptive analysis provided statistical measures for numerical columns, offering insights into their distribution and variability. Correlation analysis revealed relationships between variables, such as the inverse relationship between humidity and temperature. Data visualization techniques, including histograms and line graphs, helped visualize the distributions and patterns of accidents.

Temporal analysis revealed trends over the years, with a general increase in accidents until 2022, followed by a sudden drop in 2023. Analysis by month showed a peak in accidents during December, potentially influenced by weather conditions and holiday travel. Analysis by day of the week indicated higher accident rates on Thursday, Friday, and Saturday, possibly due to increased traffic volume. Analysis by hour of the day highlighted peak accident occurrences during morning and evening rush hours.

Geographical analysis identified California, Texas, and Florida as the states with the highest number of accidents, likely due to their population and traffic congestion. Within California, San Juan Capistrano was identified as the city with the most accidents.

Severity analysis revealed that most accidents had a severity level of 2, followed by severity levels 3 and 1. The severity levels showed fluctuations over the years, with severity 2 consistently being the most common.

Driving factors analysis indicated that the occurrence of driving factors such as crossings, junctions, and traffic signals was relatively low in the dataset, suggesting their rarity or absence in most locations.

The analysis also included a Random Forest model to predict accident severity based on various features. The model achieved an accuracy of 84.09% in predicting severity, indicating a high level of correctness in its predictions.

Overall, the analysis of the US Accidents dataset provided valuable insights into the factors contributing to traffic accidents, patterns and trends of accidents over time, the impact of weather on accidents, and the distribution of accidents across different states and cities. These insights can inform policymakers, law enforcement agencies, and organizations working towards road safety in developing effective strategies to reduce the number of accidents and ensure public safety on the nation's roadways.

REFERENCES

US Accidents (2016 - 2023). (2023, May 28). Kaggle.

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

Downloads | Apache Spark. (n.d.).

<https://spark.apache.org/downloads.html>

Shafique, A. (2021, December 9). Exploratory Data Analysis using Pyspark

Dataframe in Python. *Medium*.

<https://medium.com/@aieeshashafique/exploratory-data-analysis-using-pyspark-dataframe-in-python-bd55c02a2852>

Mondal, S. (2021). An Introduction to Data Analysis using Spark SQL. *Analytics*

Vidhya. <https://www.analyticsvidhya.com/blog/2021/08/an-introduction-to-data-analysis-using-spark-sql/>

APPENDIX

```
import pandas as pd
import numpy as n
import matplotlib.pyplot as plt
```

Data Management & Big Data

```

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)

### TRAFFIC ACCIDENTS OVER THE YEARS -- NORMAL LINEGRAPH ##
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()

### TRAFFIC ACCIDENTS OVER THE YEARS -- PLOTLY LINEGRAPH ##
import plotly.graph_objects as go
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=results_pd["Year"],
    y=results_pd["Accidents"],
    mode='lines+markers',
    name='Number of Traffic Accidents',
    line=dict(color='darkblue', width=2),
    marker=dict(symbol='circle', size=6, color='darkblue')
))

fig.update_layout(
    title="Traffic Accidents Over the Years",
    xaxis=dict(title="Year"),
    yaxis=dict(title="Number of Traffic Accidents"),
    showlegend=True,
    plot_bgcolor='rgba(0, 0, 0, 0)' # Set plot background color to transparent
)

fig.show()

## SCATTERPLOT - TEMPERATURE VS NUMBER OF ACCIDENTS ##
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.withColumnn("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()

### ARCHIT's CODE ###

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()

## TRAFFIC ACCIDENTS BY MONTH ##

from pyspark.sql.functions import month
from pyspark.sql.types import StringType
import calendar

# Compute the count of cases for each month
month_df =
df_pyspark_clean.select(month('Start_Time').alias('Month')).groupBy('Month').count()

# Sort the DataFrame by the 'Month' column in ascending order
month_df = month_df.orderBy('Month')

# Create a UDF to map month numbers to month names
month_names = [calendar.month_name[i] for i in range(1, 13)]
month_names_udf = F.udf(lambda month_num: month_names[month_num - 1],
StringType())

```

```
# Apply the UDF to convert month numbers to month names
month_df = month_df.withColumn('Month', month_names_udf('Month'))
```

```
import pandas as pd
import plotly.graph_objects as go
import calendar
```

```
# Convert the PySpark DataFrame to Pandas DataFrame for visualization
month_df_pd = month_df.toPandas()
```

```
# Define the colorscale
colorscale = 'plasma'
```

```
# Create the figure and configure the layout
fig = go.Figure()
```

```
# Add the bar trace
fig.add_trace(go.Bar(
    x=month_df_pd['count'],
    y=month_df_pd['Month'],
    orientation='h',
    marker=dict(
        color=month_df_pd['count'],
        colorscale=colorscale,
        cmin=month_df_pd['count'].min(),
        cmax=month_df_pd['count'].max(),
        colorbar=dict(
            title='Accident Cases'
        )
    )
))
```

```
# Configure the text annotations
total = month_df_pd['count'].sum()
for i, v in enumerate(month_df_pd['count']):
    fig.add_annotation(
        x=v,
        y=month_df_pd['Month'][i],
        text=f'{v/total*100:.2f}%',
        showarrow=False,
        font=dict(
            size=10,
            color='black' # Adjust the font color for better visibility
        )
    )
```

```

    ),
    align='left',
    xshift=10
)

# Configure the layout
fig.update_layout(
    title='\n Accident Percentage\nfor different months in US (2016-2023)\n',
    xaxis_title='\nAccident Cases\n',
    yaxis_title='\nMonths\n',
    xaxis=dict(
        tickmode='linear',
        dtick=50000, # Adjust the tick interval to avoid cluttering
        range=[0, month_df_pd['count'].max() + 10000] # Adjust the x-axis range
    ),
    plot_bgcolor='white',
    showlegend=False,
    height=600, # Increase figure height for better visibility
    width=800 # Increase figure width for better visibility
)

# Show the figure
fig.show()

## TRAFFIC ACCIDENTS - WEEK ANALYSIS ##

import pyspark.sql.functions as F
import pandas as pd
import plotly.graph_objects as go
import calendar

# Compute the count of cases for each day
day_df =
df_pyspark_clean.select(F.dayofweek('Start_Time').alias('Day')).groupBy('Day').count()

# Map day numbers to day names
day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

# Convert 'Day' column to a categorical type with desired order
day_df = day_df.withColumn('Day', F.expr(f'cast(Day as string)').cast('string'))
day_df = day_df.withColumn('Day', F.when(F.col('Day') == '1', day_names[0])
    .when(F.col('Day') == '2', day_names[1])
    .when(F.col('Day') == '3', day_names[2])
    .when(F.col('Day') == '4', day_names[3])
    .when(F.col('Day') == '5', day_names[4])

```

```

        .when(F.col('Day') == '6', day_names[5])
        .when(F.col('Day') == '7', day_names[6]))

# Sort the DataFrame by the index of day names
day_df = day_df.toPandas().set_index('Day').loc[day_names].reset_index()

# Convert the PySpark DataFrame to Pandas DataFrame for visualization
day_df_pd = pd.DataFrame(day_df)

# Compute the total number of accident cases
total_cases = day_df_pd['count'].sum()

# Compute the percentage of accidents for each day
day_df_pd['Percentage'] = (day_df_pd['count'] / total_cases) * 100

# Define the colorscale
colorscale = 'plasma'

# Create the figure and configure the layout
fig = go.Figure()

# Add the bar trace
fig.add_trace(go.Bar(
    x=day_df_pd['Percentage'],
    y=day_df_pd['Day'],
    orientation='h',
    marker=dict(
        color=day_df_pd['Percentage'],
        colorscale=colorscale,
        cmin=day_df_pd['Percentage'].min(),
        cmax=day_df_pd['Percentage'].max(),
        colorbar=dict(
            title='Percentage'
        )
    )
))

# Configure the text annotations
for i, row in day_df_pd.iterrows():
    fig.add_annotation(
        x=row['Percentage'],
        y=row['Day'],
        text=f'{row["Percentage"]:.2f}%',
        showarrow=False,
        font=dict(
            size=12,

```

```

        color='black' # Adjust the font color for better visibility
    ),
    align='left',
    xshift=10
)

# Configure the layout
fig.update_layout(
    title='\nPercentage of Accidents\nby Day of the Week (2016-2023)\n',
    xaxis_title='\nPercentage of Accident Cases\n',
    yaxis_title='\nDay of the Week\n',
    xaxis=dict(
        tickmode='linear',
        dtick=5, # Adjust the tick interval for the x-axis
        range=[0, day_df_pd['Percentage'].max() + 5] # Adjust the x-axis range
    ),
    plot_bgcolor='white',
    showlegend=False,
    height=600, # Increase figure height for better visibility
    width=800 # Increase figure width for better visibility
)

# Show the figure
fig.show()

#### WEEKDAY VS WEEKEND ANALYSIS ####

import pandas as pd
import plotly.graph_objects as go
import pyspark.sql.functions as F

# Compute the count of cases for weekdays and weekends
weekday_df =
df_pyspark_clean.select(F.dayofweek('Start_Time').alias('Day')).groupBy('Day').count().filter
(F.col('Day') < 7)
weekend_df =
df_pyspark_clean.select(F.dayofweek('Start_Time').alias('Day')).groupBy('Day').count().filter
(F.col('Day') >= 7)

# Compute the total number of weekday and weekend cases
total_weekday_cases = weekday_df.agg(F.sum('count')).collect()[0][0]
total_weekend_cases = weekend_df.agg(F.sum('count')).collect()[0][0]

# Compute the percentage of weekday and weekend cases

```

```

weekday_percentage = (total_weekday_cases / (total_weekday_cases +
total_weekend_cases)) * 100
weekend_percentage = (total_weekend_cases / (total_weekday_cases +
total_weekend_cases)) * 100

# Create the DataFrame for the pie chart
pie_df = pd.DataFrame({'Type': ['Weekday', 'Weekend'], 'Percentage': [weekday_percentage,
weekend_percentage]})

# Create the pie chart
fig = go.Figure(data=[go.Pie(
    labels=pie_df['Type'],
    values=pie_df['Percentage'],
    pull=[0, 0.2], # Pull the "Weekend" slice to create an exploded effect
    textinfo='label+percent', # Display only the percentage values on the labels
    textfont=dict(size=12),
    marker=dict(
        colors=['#1f77b4', '#ff7f0e'],
        line=dict(color='ffffff', width=2)
    ),
    hole=0.3 # Create a donut shape by setting the hole size
)])

# Configure the layout
fig.update_layout(
    title='\nPercentage of Accidents\nWeekday vs Weekend (2016-2023)\n',
    showlegend=False,
    height=600, # Increase figure height for better visibility
    width=800 # Increase figure width for better visibility
)

# Show the figure
fig.show()

```

TRAFFIC ACCIDENTS - HOURLY ANALYSIS

```

import pandas as pd
import pyspark.sql.functions as F

# Compute the count of cases for each hour
hour_df =
df_pyspark_clean.select(F.hour('Start_Time').alias('Hours')).groupBy('Hours').count()

# Convert the PySpark DataFrame to Pandas DataFrame
hour_df_pd = hour_df.toPandas()

```



```

# Rename the columns
hour_df_pd = hour_df_pd.rename(columns={'Hours': 'Hours', 'count': 'Cases'})

# Sort the DataFrame by 'Hours' column
hour_df_pd = hour_df_pd.sort_values('Hours')

## TRAFFIC ACCIDENTS - HOURLY ANALYSIS - BARPLOT ##

import pandas as pd
import plotly.graph_objects as go
import pyspark.sql.functions as F

# Compute the count of cases for each hour
hour_df =
df_pyspark_clean.select(F.hour('Start_Time').alias('Hours')).groupBy('Hours').count()

# Convert the PySpark DataFrame to Pandas DataFrame
hour_df_pd = hour_df.toPandas()

# Rename the columns
hour_df_pd = hour_df_pd.rename(columns={'Hours': 'Hours', 'count': 'Cases'})

# Calculate the total number of cases
total_cases = hour_df_pd['Cases'].sum()

# Calculate the percentage for each hour
hour_df_pd['Percentage'] = (hour_df_pd['Cases'] / total_cases) * 100

# Find the hour with the highest percentage
max_percentage_hour = hour_df_pd['Percentage'].idxmax()

# Create a color list for the bars
colors = ['#1f77b4'] * 24
colors[max_percentage_hour] = '#ff7f0e' # Set a different color for the bar with the highest
percentage

# Create the bar chart
fig = go.Figure(data=[go.Bar(
    x=hour_df_pd['Hours'],
    y=hour_df_pd['Cases'],
    text=hour_df_pd['Percentage'].apply(lambda x: f'{x:.2f}%'),
    textposition='auto', # Display the text on the bars
    marker=dict(color=colors) # Set the bar colors
    ## marker_color=sns.color_palette() # Set the bar colors using the Seaborn palette

```

```
))
```

```
# Configure the layout
```

```
fig.update_layout(  
    title='\nAccident Percentage\nfor different hours along the day\n',  
    xaxis=dict(  
        tickmode='linear',  
        tick0=0, # Start the tick from 0  
        dtick=1, # Set the tick interval to 1  
        range=[-0.5, 23.5], # Set the x-axis range from -0.5 to 23.5  
        tickfont=dict(size=10) # Adjust the tick font size  
    ),  
    xaxis_title='\nHours\n',  
    yaxis_title='\nAccident Cases\n',  
    plot_bgcolor='white',  
    showlegend=False,  
    height=600,  
    width=800  
)
```

```
# Show the figure
```

```
fig.show()
```

```
## TRAFFIC ACCIDENTS - HOURLY ANALYSIS - AREAPLOT ##
```

```
import pandas as pd
```

```
import plotly.graph_objects as go
```

```
import pyspark.sql.functions as F
```

```
# Compute the count of cases for each hour
```

```
hour_df =
```

```
df_pyspark_clean.select(F.hour('Start_Time').alias('Hours')).groupBy('Hours').count()
```

```
# Convert the PySpark DataFrame to Pandas DataFrame
```

```
hour_df_pd = hour_df.toPandas()
```

```
# Rename the columns
```

```
hour_df_pd = hour_df_pd.rename(columns={'Hours': 'Hours', 'count': 'Cases'})
```

```
# Sort the DataFrame by the 'Hours' column in ascending order
```

```
hour_df_pd = hour_df_pd.sort_values('Hours')
```

```
# Create the line plot
```

```
fig = go.Figure(data=go.Scatter(
```

```

        x=hour_df_pd['Hours'],
        y=hour_df_pd['Cases'],
        mode='lines',
        line=dict(color='#05ffda', width=2),
        fill='tozeroy' # Fill the area below the line
    ))

# Configure the layout
fig.update_layout(
    title='\nAccident Distribution \nby Hour of the Day\n',
    xaxis=dict(
        tickmode='linear',
        tick0=0, # Start the tick from 0
        dtick=1, # Set the tick interval to 1
        range=[0, 23], # Set the x-axis range from 0 to 23
        tickfont=dict(size=10) # Adjust the tick font size
    ),
    xaxis_title='\nHours\n',
    yaxis_title='\nAccident Cases\n',
    plot_bgcolor='white',
    showlegend=False,
    height=600,
    width=800
)

# Show the figure
fig.show()

```

NIKSHITA UPDATED

TRAFFIC ACCIDENTS BY SEVERITY - 3D PIE CHART

```

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()

```

TRAFFIC ACCIDENTS BY SEVERITY - BARPLOT

```

# Group the data by Year and Severity, and count the number of accidents
grouped_df = df_pyspark_clean.groupBy('Year', 'Severity').count().orderBy('Year')

# Convert the Spark DataFrame to Pandas DataFrame for plotting
pandas_df = grouped_df.toPandas()

# Pivot the data to have Severity values as columns
pivoted_df = pandas_df.pivot(index='Year', columns='Severity', values='count')

colors = ["#FFB600", "#44A9CC", "#EB563A", "#F4B998"]
pivoted_df.plot(kind='bar', stacked=False, color=colors)

plt.xlabel('Year')

```

```
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents by Severity for Each Year')
plt.legend(title='Severity')
```

NO OF TRAFFIC ACCIDENTS BY STATE - MAP

```
import plotly.graph_objects as go
```

```
# Perform the count aggregation using PySpark
state_count = df_pyspark_clean.groupBy('State').count().toPandas()
```

```
fig = go.Figure(data=go.Choropleth(
    locations=state_count['State'],
    z=state_count['count'].astype(float),
    locationmode='USA-states',
    colorscale='aggrnyl',
    colorbar_title="Count of Accidents",
))
```

```
fig.update_layout(
    title_text='No of US Accidents by State',
    geo_scope='usa',
)
```

```
fig.show()
```

NO OF TRAFFIC ACCIDENTS BY SEVERITY - DENSITY MAP

```
lat_lng_severity_df = df_pyspark_clean.select(
    col("Start_Lng").cast("float").alias("Longitude"),
    col("Start_Lat").cast("float").alias("Latitude"),
    col("Severity")
)
```

```
# Step 5: Convert to Pandas DataFrame
pandas_df = lat_lng_severity_df.toPandas()
```

```
# Step 6: Create scatter plot
plt.figure(figsize=(30, 10))
plt.scatter(pandas_df["Longitude"], pandas_df["Latitude"],
    c=pandas_df["Severity"].astype(int), cmap="viridis", s=5)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("US Accidents: Scatter Plot (Severity)")
plt.colorbar(label="Severity")
plt.show()
```

```
## ARCHIT UPDATED ##
```

```
## TRAFFIC ACCIDENTS FOR TOP 10 STATES WITH THEIR RESPECTIVE CITIES  
##
```

```
import plotly.graph_objects as go  
import plotly.colors as colors
```

```
# Convert the PySpark DataFrame to Pandas DataFrame  
state_counts_pd = df_pyspark_clean.groupby('State', 'City').count().toPandas()
```

```
# Group by state and get the count for each city  
state_city_counts = state_counts_pd.groupby('State')['count'].sum().reset_index()
```

```
# Select the top 10 states  
top_10_states = state_city_counts.nlargest(10, 'count')
```

```
# Create a diverging color scale  
color_scale = colors.diverging.RdYlBu
```

```
# Create the horizontal bar chart with color shade  
fig = go.Figure()
```

```
# Add bars to the chart  
for i, row in top_10_states.iterrows():  
    bar_text = f'{row["State"]}' -  
{row["count"]} <br> {state_counts_pd.loc[state_counts_pd["State"] == row["State"],  
'City'].values[0]}"  
    fig.add_trace(go.Bar(  
        x=[row["count"]],  
        y=[row["State"]],  
        orientation='h',  
        text=[bar_text],  
        hovertemplate='%{text}',  
        marker=dict(  
            color=row["count"],  
            colorscale=color_scale,  
            line=dict(color='rgba(50, 171, 96, 1.0)', width=1)  
        ),  
        textposition='inside'  
    ))
```

```
# Update layout properties  
fig.update_layout(  

```

```

        title='Top 10 States with the Highest Number of Accidents',
        xaxis_title='Accident Count',
        yaxis_title='State',
        yaxis=dict(autorange='reversed'),
        showlegend=False
    )

# Show the horizontal bar chart
fig.show()

## TOTAL NUMBER OF ACCIDENTS BY SEVERITY (3D SCATTER PLOT)
import plotly.graph_objects as go

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

# Create the 3D scatter plot
fig = go.Figure(data=[go.Scatter3d(
    x=severity_counts_pd['Severity'],
    y=severity_counts_pd['Accident_Count'],
    z=[0] * len(severity_counts_pd), # Z-values set to 0
    mode='markers',
    marker=dict(
        size=10,
        color=severity_counts_pd['Accident_Count'],
        colorscale='Viridis',
        opacity=0.8
    ),
    text=severity_counts_pd['Severity']
)])

# Update layout properties
fig.update_layout(title='Total Number of Accidents by Severity (3D Scatter Plot)',
                  scene=dict(xaxis_title='Severity', yaxis_title='Accident_Count'))

# Show the 3D scatter plot
fig.show()

```

TOTAL NUMBER OF ACCIDENTS BY SEVERITY (3D DONUT PIE CHART)

```
import plotly.express as px
```

```

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

# Create the 3D plot
fig = px.pie(severity_counts_pd, values='Accident_Count', names='Severity',
             title='Total Number of Accidents by Severity (3D)',
             hover_data=['Accident_Count'], labels={'Accident_Count': 'Count'})

# Add 3D effect to the pie chart
fig.update_traces(hole=0.4, hoverinfo='label+percent+name', textinfo='value+percent',
                  textfont_size=12, marker=dict(colors=px.colors.sequential.RdBu))

# Show the 3D plot
fig.show()

import matplotlib.pyplot as plt
from pyspark.sql.functions import count

# Create a 4x3 grid of subplots
fig, axes = plt.subplots(4, 3, figsize=(25, 25))

# List of columns to plot
columns = ['Amenity', 'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway',
           'Roundabout', 'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal']

# Calculate the count of each category for each column across all years
category_counts = []
for column in columns:
    counts = df_pyspark_clean.groupBy(column).agg(count('*').alias('count'))
    category_counts.append(counts)

# Generate pie charts for each column and place them in the corresponding subplot
for i, column in enumerate(columns):
    row = i // 3 # Calculate the row index
    col = i % 3 # Calculate the column index

    # Create a pie chart for the current column with data from all years
    ax = axes[row, col]
    labels = [str(label[column]) for label in category_counts[i].collect()]
    counts = [count['count'] for count in category_counts[i].collect()]
    ax.pie(counts, labels=labels, autopct='%1.1f%%', colors=["mediumorchid",
"limegreen"])

    # Set the title for the subplot
    ax.set_title(column, fontweight="bold")

```



```

# Adjust the layout
plt.tight_layout()
plt.show()

## WORDCLOUD - SEVERITY4 for DESCRIPTION ###

import matplotlib as mt
import matplotlib.pyplot as plt
import numpy as np
from wordcloud import WordCloud, STOPWORDS
from PIL import Image
from pyspark.sql.functions import col

severity4=df_pyspark_clean[df_pyspark_clean['Severity']==4]

# Convert PySpark DataFrame to Pandas DataFrame
df = severity4.select(col("Description")).toPandas()

# Creating variable with data for this Word Cloud
text = ' '.join(df['Description'].tolist())

# Creating Mask for Word Cloud
path = 'C:/Users/14086/Downloads/'
mask = np.array(Image.open(path + 'car.jpg'))
stopwords = set(STOPWORDS)

# Word Cloud Creation
wc = WordCloud(width=800, height=600, mask=mask, random_state=101,
               min_font_size=1.5, stopwords=stopwords, background_color="white",
               scale=3, max_words=200, collocations=False)

# Generate Word Cloud
wc.generate(str(text))

# Show
fig=plt.figure(figsize=(30,10))
plt.ylim(0,1300)
plt.xlim(0,7500)
plt.gca().invert_yaxis()
plt.axis("off")
plt.title('Accidents: Severity 4', fontdict={'fontsize': 20, 'fontweight': 'bold'})
plt.imshow(wc, interpolation='bilinear', aspect='auto')

plt.show()

```

```

## RANDOM FOREST MODEL - WORKING ##

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select the relevant columns for training
selected_cols = ['Start_Lat', 'Start_Lng', 'Temperature', 'Wind_Chill', 'Humidity', 'Pressure',
'Visibility', 'Wind_Speed', 'Severity']

# Drop rows with missing values in selected columns
dataset = df_pyspark_clean.dropna(subset=selected_cols)

# Convert string columns to numeric type
for col in selected_cols:
    dataset = dataset.withColumn(col, dataset[col].cast('double'))

# Create a vector assembler to combine the selected columns into a feature vector
assembler = VectorAssembler(inputCols=selected_cols[:-1], outputCol='features')

# Split the dataset into training and testing sets (70% for training, 30% for testing)
train_data, test_data = dataset.randomSplit([0.7, 0.3], seed=42)

# Create a Random Forest classifier
rf = RandomForestClassifier(labelCol='Severity', featuresCol='features')

# Create a pipeline to chain the vector assembler and the Random Forest classifier
pipeline = Pipeline(stages=[assembler, rf])

# Train the model
model = pipeline.fit(train_data)

# Make predictions on the test data
predictions = model.transform(test_data)

# Evaluate the model using accuracy metric
evaluator = MulticlassClassificationEvaluator(labelCol='Severity', metricName='accuracy')
accuracy = evaluator.evaluate(predictions)
print("Accuracy: {:.2%}".format(accuracy))

```