US ACCIDENTS ANALYSIS

CS-GY 6513 Spring 2022

Team Members

Akshat Shah(as15098)
Ansh Desai(asd9717)
Jay Sarvaiya(js12178)
Mohit Pursnani(mp5578)

Table Of Contents

1. Problem Statement and Motivation

Reducing traffic accidents is an essential public safety challenge all over the world; therefore, accident analysis has been a subject of much research in recent decades. Each year the number of accidents is increasing rapidly all over the world. There have found to be various factors which contribute to the causing of accidents. So, for our project we have tried to analyse the US accident data from 49 states and find out possible factors cause these accidents.
The analysis includes number of accidents by year, number of accidents by state, number of accidents by city, number of accidents by month, day and hour, factors responsible of the accidents like weather, wind flow, temperature, location, etc.

We have also used Machine Learning (MLlib) to figure out which factors have most correlation to actual accidents and also to predict the severity of accidents caused all over US. This can help various government authorities to reduce the number of accidents caused.

2. Why is our problem a big data problem?

“Big data” is high-volume, velocity, and variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making.” It is an unstructured data that exceeds the processing complexity of conventional database architectures. Since our dataset collects the accidents information from all over the US, and the number of accidents will never become nil. It will keep on getting cumulated over years which leads to increase in Volume and Velocity of the data. For this reason, we will need a real time streaming architecture so that we do not need to change the data once a new data is added. Hence to accumulate this aspect as well, our project is a big data problem.

3. Dataset Description

The dataset that we have used is a countrywide car accident dataset, which covers 49 states of the USA. The accident data are collected from February 2016 to Dec 2021, using multiple APIs that provide streaming traffic incident (or event) data. These APIs broadcast traffic data captured by a variety of entities, such as the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks. Currently, there are about 2.8 million accident records in this dataset.
You can view the entire data from here.

```
root
 |-- ID: string (nullable = true)
 |-- Severity: integer (nullable = true)
 |-- Start_Time: string (nullable = true)
 |-- End_Time: string (nullable = true)
 |-- Start_Lat: double (nullable = true)
 |-- Start_Lng: double (nullable = true)
 |-- End_Lat: double (nullable = true)
 |-- End_Lng: double (nullable = true)
 |-- Distance(mi): double (nullable = true)
 |-- Description: string (nullable = true)
 |-- Street: string (nullable = true)
 |-- Side: 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): double (nullable = true)
 |-- Wind_Chill(F): double (nullable = true)
 |-- Humidity(%): double (nullable = true)
 |-- Pressure(in): double (nullable = true)
 |-- Visibility(mi): double (nullable = true)
 |-- Wind_Direction: string (nullable = true)
 |-- Wind_Speed(mph): double (nullable = true)
 |-- Precipitation(in): double (nullable = true)
 |-- Weather_Condition: string (nullable = true)
 |-- Amenity: boolean (nullable = true)
 |-- Bump: boolean (nullable = true)
 |-- Crossing: boolean (nullable = true)
 |-- Give_Way: boolean (nullable = true)
 |-- Junction: boolean (nullable = true)
 |-- No_Exit: boolean (nullable = true)
 |-- Railway: boolean (nullable = true)
 |-- Roundabout: boolean (nullable = true)
 |-- Station: boolean (nullable = true)
 |-- Stop: boolean (nullable = true)
 |-- Traffic_Calming: boolean (nullable = true)
 |-- Traffic_Signal: boolean (nullable = true)
 |-- Turning_Loop: boolean (nullable = true)
 |-- Sunrise_Sunset: string (nullable = true)
 |-- Civil_Twilight: string (nullable = true)
 |-- Nautical_Twilight: string (nullable = true)
 |-- Astronomical_Twilight: string (nullable = true)
```

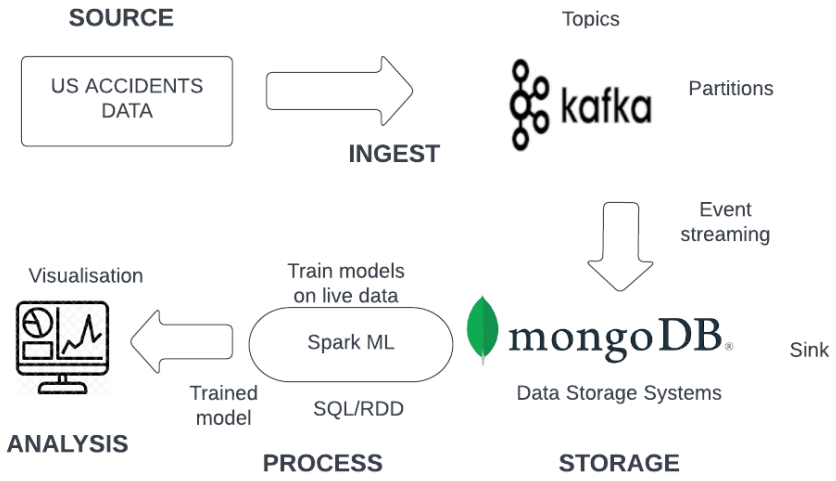
Brief introduction about attributes in our dataset

Attributes	Description
ID	This is a unique identifier of the accident record.
Severity	Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).
Start_Time	Shows start time of the accident in local time zone.
End_Time	Shows end time of the accident in local time zone. End time here refers to when the impact of accident on traffic flow was dismissed.
Start_Lat	Shows latitude in GPS coordinate of the start point.
Start_Lng	Shows longitude in GPS coordinate of the start point.
End_Lat	Shows latitude in GPS coordinate of the end point.
End_Lng	Shows longitude in GPS coordinate of the end point.
Distance(mi)	The length of the road extent affected by the accident.
Description	Shows natural language description of the accident.
Number	Shows the street number in address field.
Street	Shows the street name in address field.
Side	Shows the relative side of the street (Right/Left) in address field.
City	Shows the city in address field.
County	Shows the county in address field.
State	Shows the state in address field.
Zipcode	Shows the zipcode in address field.
Country	Shows the country in address field.
Timezone	Shows timezone based on the location of the accident (eastern, central, etc.).
Airport_Code	Denotes an airport-based weather station which is the closest one to location of the accident.
Weather_Timestamp	Shows the time-stamp of weather observation record (in local time).
Temperature(F)	Shows the temperature (in Fahrenheit).
Wind_Chill(F)	Shows the wind chill (in Fahrenheit).
Humidity(%)	Shows the humidity (in percentage).
Pressure(in)	Shows the air pressure (in inches).
Visibility(mi)	Shows visibility (in miles).
Wind_Direction	Shows wind direction.
Wind_Speed(mph)	Shows wind speed (in miles per hour).
Precipitation(in)	Shows precipitation amount in inches, if there is any.
Weather_Condition	Shows the weather condition (rain, snow, thunderstorm, fog, etc.)
Amenity	A POI annotation which indicates presence of amenity in a nearby location.
Bump	A POI annotation which indicates presence of speed bump or hump in a nearby location.
Crossing	A POI annotation which indicates presence of crossing in a nearby location.
Give_Way	A POI annotation which indicates presence of give_way in a nearby location.
Junction	A POI annotation which indicates presence of junction in a nearby location.
No_Exit	A POI annotation which indicates presence of no_exit in a nearby location.
Railway	A POI annotation which indicates presence of railway in a nearby location.

Roundabout	A POI annotation which indicates presence of roundabout in a nearby location.
Station	A POI annotation which indicates presence of station in a nearby location.
Stop	A POI annotation which indicates presence of stop in a nearby location.
Traffic_Calming	A POI annotation which indicates presence of traffic_calming in a nearby location.
Traffic_Signal	A POI annotation which indicates presence of traffic_signal in a nearby location.
Turning_Loop	A POI annotation which indicates presence of turning_loop in a nearby location.
Sunrise_Sunset	Shows the period of day (i.e., day or night) based on sunrise/sunset.
Civil_Twilight	Shows the period of day (i.e., day or night) based on civil twilight.
Nautical_Twilight	Shows the period of day (i.e., day or night) based on nautical twilight.
Astronomical_Twilight	Shows the period of day (i.e., day or night) based on astronomical twilight.

4. Architecture

The following proposed architecture was used for building reliable and scalable real-time ETL pipeline which conforms to big data principles. Streaming data integration is the foundation for leveraging streaming analytics. In many data centres, different type of servers generates large amount of data (events, Event in this case is timestamp for one data point) in real-time. There is always a need to process these data in real-time and generate insights which will be used by the server/data centre monitoring people, and they must track these server's status regularly and find the resolution in case of issues occurring, for better server stability. Since the data is huge and coming in real-time, we need to choose the right architecture with scalable storage and computation frameworks/technologies. Hence, we want to use scalable and reliable technology stack which can process and give results in real-time.



The components used in the above architecture and their use is described below:

1. **Data Source:**
   We get input data from accidents dataset that is publicly available here. The dataset that we have used is a countrywide car accident dataset, which covers 49 states of the USA. The accident data are collected from February 2016 to Dec 2021, using multiple APIs that provide streaming traffic incident (or event) data. These APIs broadcast traffic data captured by a variety of entities, such as the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks. Currently, there are about 2.8 million accident records in this dataset.

2. **Apache Kafka:**
   It's most used to create real-time streaming data pipelines and applications that adapt to changing data streams. It mixes communications, storage, and stream processing to enable for both historical and real-time data storage and analysis. Our input data is delivered to Apache Kafka, as indicated in the architecture diagram. We send the data from accidents dataset to Apache Kafka using duration between two accidents as the speed of data stream. There are two components, namely producer and consumer. The producer generates data to be consumed by consumer and send it to the sink (MongoDB).

3. Apache Spark

Apache Spark is an integral part of Spark core API to perform real-time data analytics. It allows us to build a scalable, high-throughput, and fault-tolerant streaming application of live data streams. Spark Streaming supports the processing of real-time data from various input sources and storing the processed data to various output sinks. We utilize SparkML to perform severity prediction and analyse the accidents that occur using different machine learning models.

4. Database:

With MongoDB, applications do not need to create the database and collection before they start writing data. These objects are created automatically upon first arrival of data into MongoDB. However, a time-series collection type needs to be created first before you start writing data. To make it easy to ingest time-series data into MongoDB from Kafka, these collection options are exposed as sink parameters and the time-series collection is created by the connector if it doesn't already exist. We define our data storage as sink so that subscriber can send the streaming data and access it.

5. Data Visualization

We perform exploratory data analysis that help us perform data pre-processing and machine learning model building. We utilize methods from open-source libraries such as seaborn, matplotlib, SparkML, etc. We have three different sections of exploratory data analysis based on factors such as weather, location, and time. We have built several inferences through this analysis and performed severity prediction for the real-time data. Finally, using folium library we plot markers on location of accident and identify hotspots to recognize severity of the accidents.

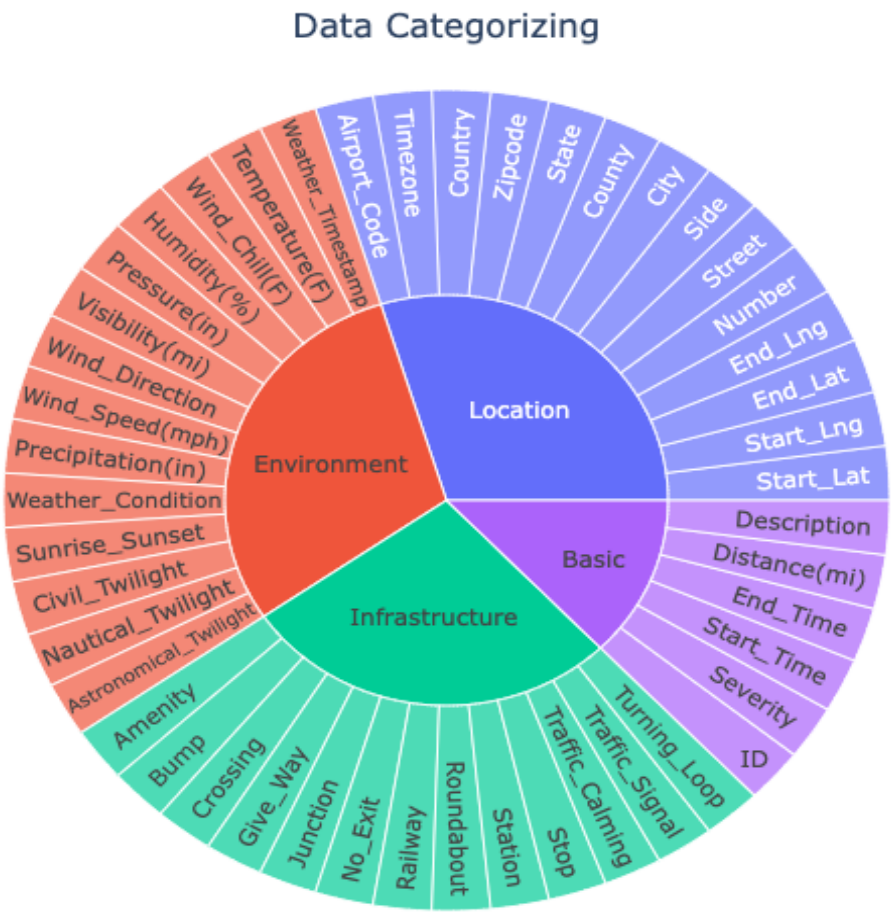
5. Data Preprocessing

Preprocessing is one of the most important steps to obtaining efficient data for analysis and prediction. Preprocessing is done here to check if the data is in the right format. We checked for missing values and removed unwanted information which was not useful in the dataset. For instance, in the dataset, the column Number had more than 60% Nulls so we dropped that column. so that the computation process was made smoother. These entire steps account for data cleaning

Analysis with the help of EDA

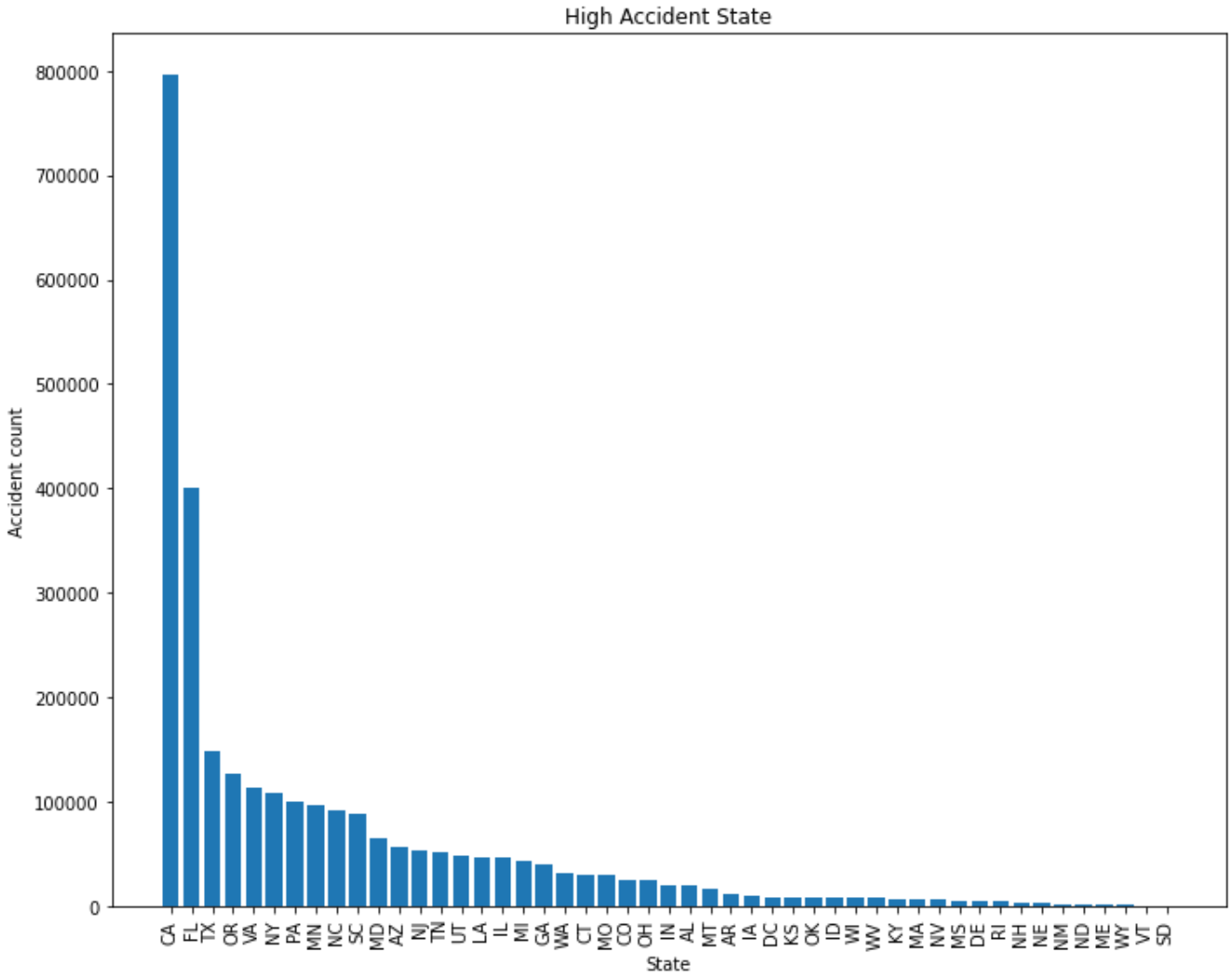
Categorizing the data

Our data set had data similar and could be categorized into different sections. The categories generated are Location, Environment, Infrastructure, and Basic. The image below has the detailed categorization



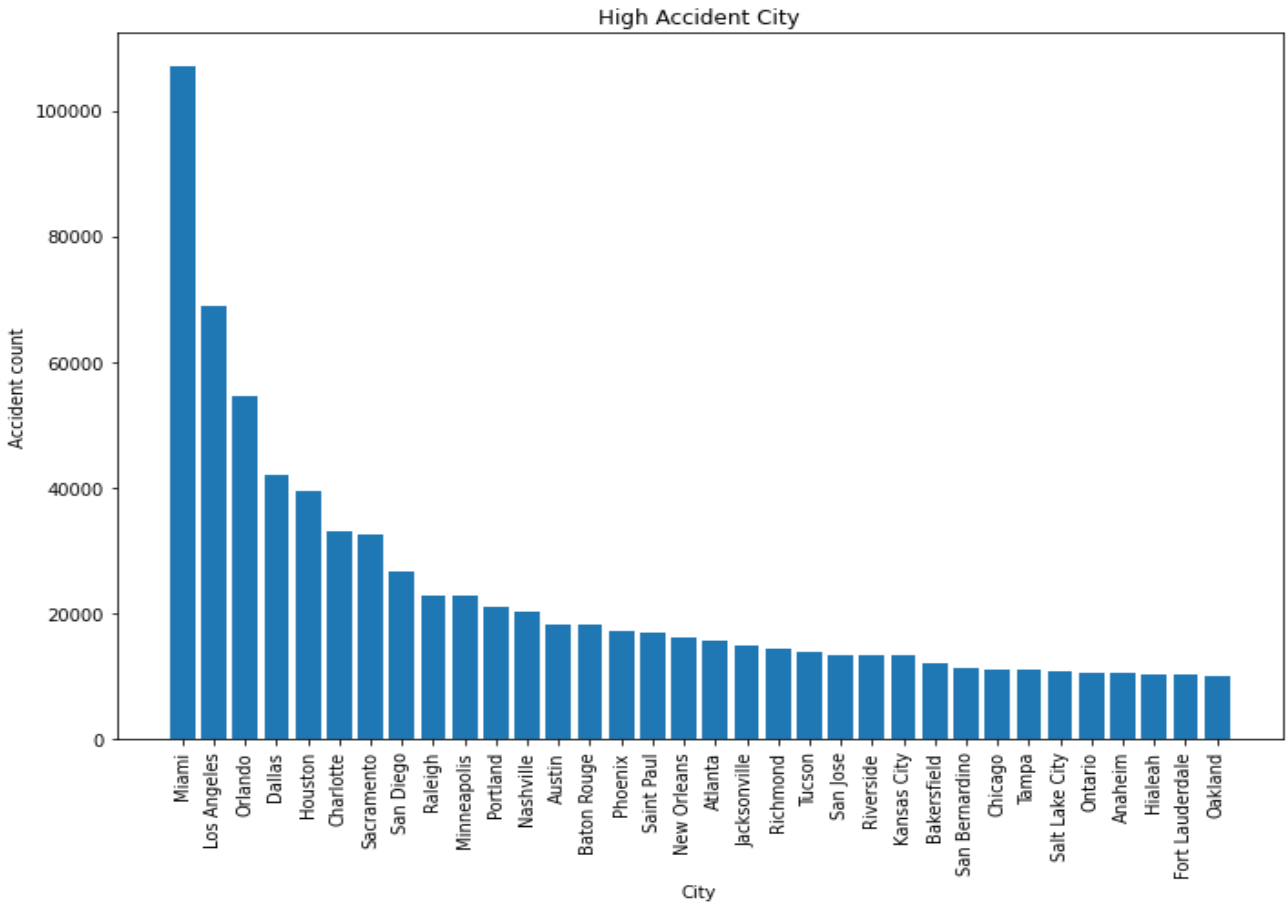
Analysis as per States

Since our Dataset is only based in the US we tried to find out if any particular State has a higher accident count each year relative to other States and later tried to find out any patterns if they existed for that particular State. As we can see in the figure below California has a much higher accident count compared to other States.



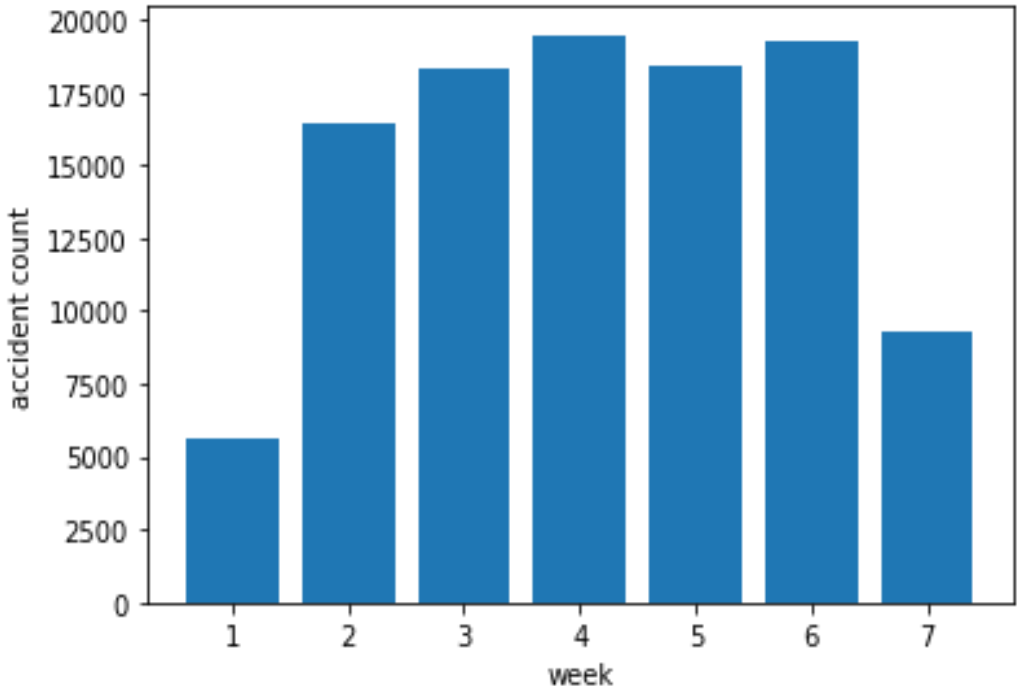
Furthering The Analysis of City

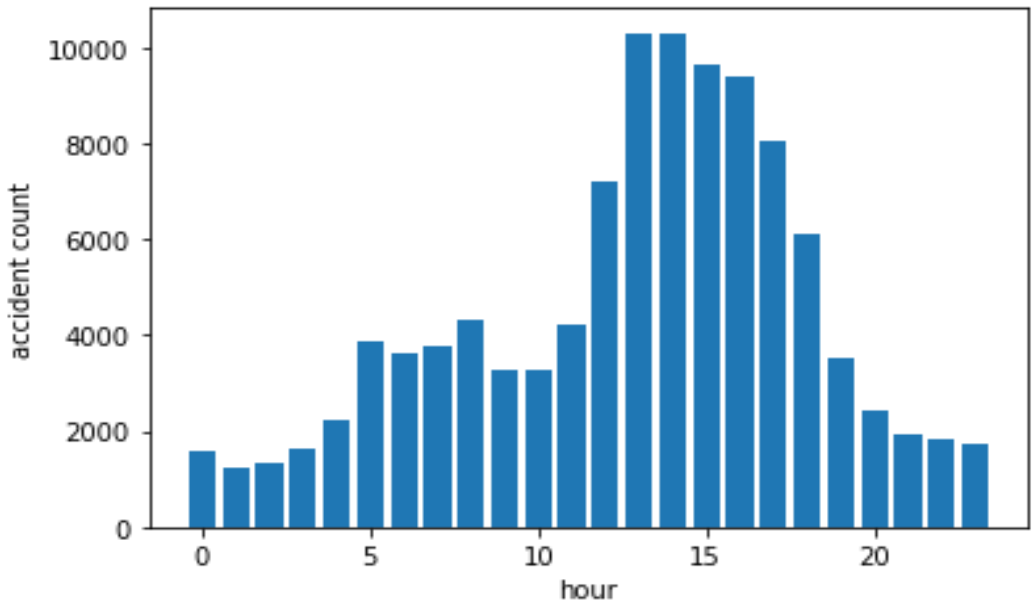
As in the above diagram we observed that California State has the highest accident count. We tried to find if the city-wise most frequent accident count belonged to California or not. In other words, we tried to find if the City wise and State wise accident counts were directly proportional or not. As we can see in the figure below the top 5 states that contributed did appear in the top 5 City wise accident count but after that city wises accident count was randomized.



Analysing Miami City

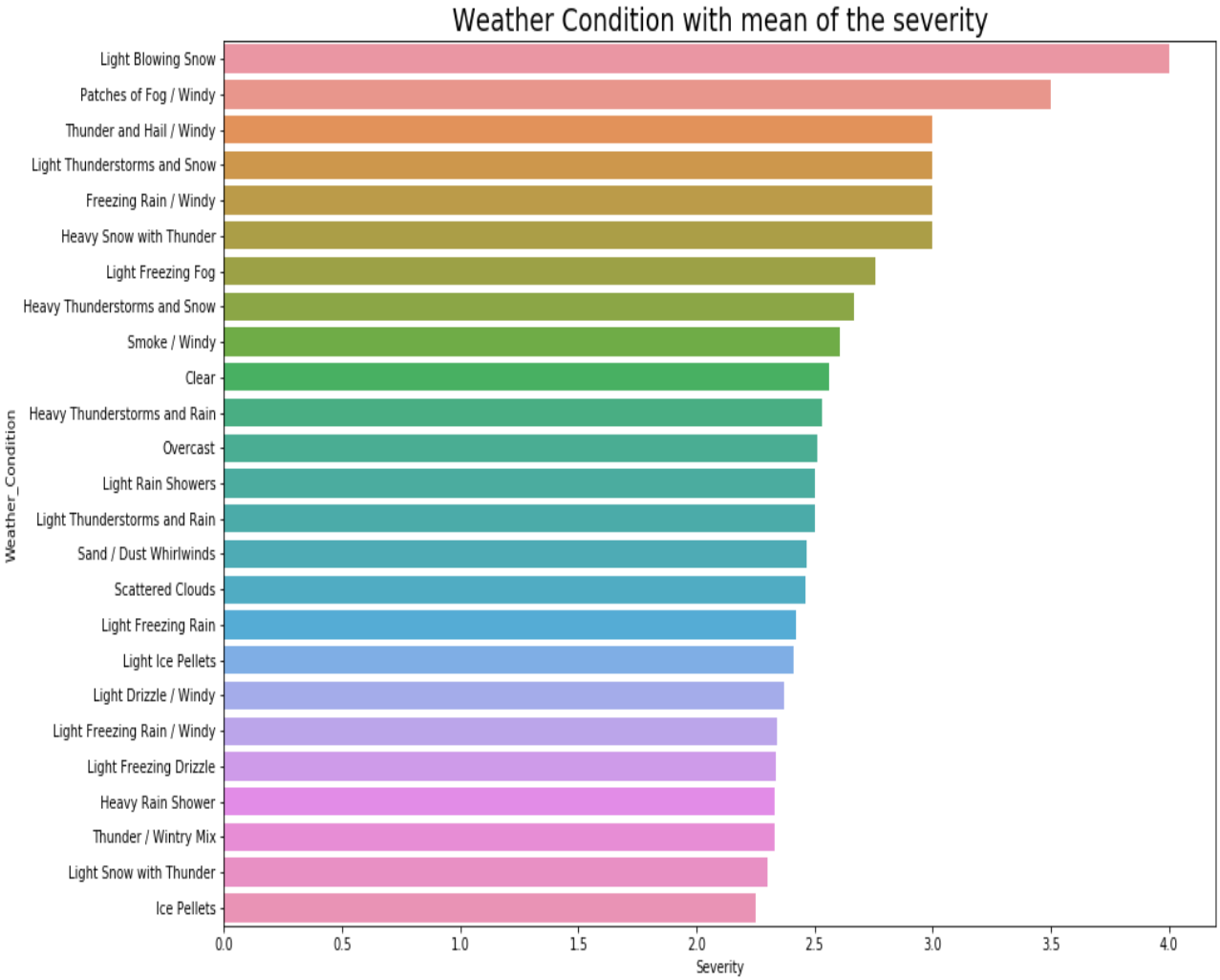
We tried to find out if there were weekdays during which accidents were more frequent or not. And further, then we check to see during which hours the days the accident were more frequent. As we can see in the figure below Wednesday through Friday and 10 am to 8 pm observed heavier accident count.





Analysing to see which weather conditions contribute to more accidents

As we can see in the figure below light blowing Snow, Thunder and hail and patches of fog with windy are the top 3 dangerous weather conditions.



The main Inferences derived from our EDA are:

1. There are a total of 11681 cities all over the US
2. Miami has the highest number of accidents
3. Wednesday Thursday and Friday are the peak days on which accidents occur
4. Accidents occur most between 10 am to 8 pm.
5. California has the highest number of accidents
6. Freezing rain with windy, light blowing Snow, and patches of fog with windy are the top 3 dangerous weather conditions.

6. Severity Prediction:

Before we use models to do prediction, we need to perform Feature Engineering and Feature Selection.
In our dataset, the Start_time and End_time which we are using to calculate prediction are of type String so we will convert the datatype to timestamp so that we can get hour, month values easily and calculate duration from those. This is as shown below:

Feature Engineering and Feature Selection

```
']: df_preprocessed = df_preprocessed.withColumn("Start_Time",to_timestamp(col("Start_Time")))
    df_preprocessed = df_preprocessed.withColumn("End_Time",to_timestamp(col("End_Time")))
```

After converting it to timestamp column type, we created new columns Duration, Month and Hour in the dataset from the timestamp fields. This will help us in our understanding of the time frame of the accident.

```
9]: timeFmt = "yyyy-MM-dd HH:mm:ss"
    timeDiff = (unix_timestamp('End_Time', format=timeFmt)- unix_timestamp('Start_Time', format=timeFmt))/3600
    df_preprocessed = df_preprocessed.withColumn("Duration", timeDiff)

    df_preprocessed = df_preprocessed.withColumn('Month',month(df_preprocessed.Start_Time))
    import pyspark.sql.functions as F
    from pyspark.sql.types import IntegerType

    hour = udf(lambda x: x.hour, IntegerType())
    df_preprocessed = df_preprocessed.withColumn("Hour", hour("Start_Time"))
```

Once we have done this, our next task is to do one hot encoding of the feature columns which we have done as shown below:

```
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^SSE", "S"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^SW", "S"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^NW", "N"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^Calm", "C"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^WSW", "W"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^ENE", "E"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^NE", "N"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^South", "S"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^NNW", "N"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^SSW", "S"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^SE", "S"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^East", "E"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^WNW", "W"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^NNE", "N"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^West", "W"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^VAR", "V"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^CALM", "C"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^ESE", "E"))
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^Variable", "V")
df_preprocessed  = df_preprocessed.withColumn("Wind_Direction", regexp_replace(col("Wind_Direction"), "^North", "N"))
```

```
df_preprocessed = df_preprocessed.withColumn('Weather_Condition',
                                             when(col('Weather_Condition') == 'Fair' | col('Weather_Condition') == 'Cle
                                                  col('Weather_Condition') == 'Overcast' | col('Weather_Condition') ==
                                                  col('Weather_Condition') == 'Smoke|Fog|Mist|Haze' | col('Weather_Cond
                                                  col('Weather_Condition') == 'Thunderstorms|T-Storm' | col('Weather_Co
                                                  col('Weather_Condition') == 'Hail|Ice Pellets' |  col('Weather_Condit
                                                  col('Weather_Condition') == 'Dust'| col('Weather_Condition') == 'Torn
```

```
'Weather_Condition',
when(col('Weather_Condition') == 'Fair' | col('Weather_Condition') == 'Clear' | col('Weather_Condition') == 'Cloudy'
     col('Weather_Condition') == 'Overcast' | col('Weather_Condition') == 'Snow|Wintry|Sleet' |
     col('Weather_Condition') == 'Smoke|Fog|Mist|Haze' | col('Weather_Condition') == 'Rain|Drizzle|Showers'
     col('Weather_Condition') == 'Thunderstorms|T-Storm' | col('Weather_Condition') == 'Windy|Squalls' |
     col('Weather_Condition') == 'Hail|Ice Pellets' |  col('Weather_Condition') == 'Thunder' |
     col('Weather_Condition') == 'Dust'| col('Weather_Condition') == 'Tornado', 1).otherwise(0))


label_encoding_features = ['Amenity','Bump', 'Crossing', 'Give_Way',
                           'Junction', 'No_Exit', 'Railway','Roundabout', 'Station', 'Stop', 'Traffic_Calming',
                          'Traffic_Signal','Turning_Loop']

# Label Encoding
for feature in label_encoding_features:
    df_preprocessed = df_preprocessed.withColumn(feature, when(col(feature), 1).otherwise(0))

label_encoding_features_2 = ['Sunrise_Sunset','Side']

for feature in label_encoding_features_2:
    indexer = StringIndexer(inputCol=feature, outputCol=feature+"1")
    df_preprocessed = indexer.fit(df_preprocessed).transform(df_preprocessed)
```

Once we did Feature Engineering, we will select relevant features from the dataset and then apply different models:

```
relevant_features = ['Month','Hour','Duration', 'Severity',
                     'Start_Lat', 'Start_Lng','Side1','Temperature(F)', 'Humidity(%)',
                     'Pressure(in)', 'Visibility(mi)', 'Wind_Speed(mph)', 'Amenity','Bump',
                     'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway','Roundabout', 'Station',
                     'Stop', 'Traffic_Calming', 'Traffic_Signal','Turning_Loop', 'Sunrise_Sunset1']
data_modelling_df = df_preprocessed.select(relevant_features)
```

To do Severity Prediction we will run Logistic Regression, Random Forest and Decision Tree but before doing that we will split the dataset into Train (70%) and Test (30%) dataset.

```
train, test = data_modelling_df.randomSplit([0.7, 0.3], seed = 2018)
features = relevant_features
features.remove("Severity")
assembler = VectorAssembler(inputCols=features, outputCol='features')
train = assembler.transform(train)
test = assembler.transform(test)
```

i. Logistic Regression:

Logistic Regression Learning Phase

```
log_reg_severity = LogisticRegression(featuresCol= 'features',labelCol='Severity')
log_reg = log_reg_severity.fit(train)
```

Logistic Regression Generalization Phase

```
pred_logistic = log_reg.transform(test)
```

Logistic Regression Model Accuracy in the Generalization Phase:

```
evaluator = MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(pred_logistic)
print("Accuracy %g"%  (accuracy))
```

```
Accuracy 0.931781
```

ii. Random Forest:

Random Forest Learning Phase

```
rf = RandomForestClassifier(labelCol='Severity', featuresCol='features', numTrees=50)
pred_rf = rf.fit(train)
```

Random Forest Generalization Phase

```
pred_rf_transformed = pred_rf.transform(test)
```

Random Forest Model Accuracy in the Generalization Phase:

```
evaluator = MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="prediction", metricName="accuracy")
rf_accuracy = evaluator.evaluate(pred_rf_transformed)
print("Accuracy %g"%  (rf_accuracy))
```

```
Accuracy 0.931795
```

iii. Decision Tree:

Decision Tree Learning Phase:

```
dt = DecisionTreeClassifier(labelCol="Severity", featuresCol="features")
dt_model = dt.fit(train)
```

Decision Tree Generalization Phase

```
prediction_dt = dt_model.transform(test)
```

Decision Tree Model Accuracy in the Generalization Phase:

```
evaluator = MulticlassClassificationEvaluator(labelCol="Severity", predictionCol="prediction", metricName="accuracy")
dt_accuracy = evaluator.evaluate(prediction_dt)
print("Accuracy %g"%  (dt_accuracy))
```

```
Accuracy 0.936358
```

Overall Accuracy of Different Models:

Model	Accuracy
Logistic Regression	0.931781
Random Forest	0.931795
Decision Tree	0.936358

We can see from the accuracy that the decision tree performed slightly better than all other models.

Severity analysis hotspots and markers indicating the level of severity on location of accident: