ISTE.470.601 - Data Mining and Exploration

Fall 2024

Dr. Khalil Al Hussaeni

Final Project Report

28/11/2024

Khaled Al Dasouki

Masa AlGhrawi

Table of Contents

Introduction **4**

Approach **5**

Overview 5

Dataset 5

Pre-Processing **6**

Removing Incident ID's 6

Translating Incident Names 7

Splitting Incident Names 8

Standardizing Severity 9

Removing Unwanted Incidents 10

Splitting Incident Time 11

Standardizing Date 11

Filling Missing Severities 11

Dealing with noise and outliers 12

Visualizing the dataset **13**

Plotting the Incidents to Create a Map-like Model 13

Traffic Incident Distribution by Month 14

Distribution of Updated Severities 16

Experimental Analysis **17**

Predicting Missing Severities 17

Method 1: Using a Categorical Imputer 17

Method 2: Using Logistic Regression 18

Method 3: Enhancing Logistic Regression 19

Method 4: Creating a Random Forest 20

Outlier Detection Using Density Based Clustering 21

Using DBSCAN to Identify Outliers 21
Removing Outliers 24
Using DBSCAN to Identify Incident Clusters 24
Future Forecast for Incident Count By Month 25
Filling in the Month of May 2024 26
Creating Forecasts for Upcoming Months 28
Correlation Analysis 37
Identifying Incident Hotspots 38
Incidents Involving Animals 38
Incidents Involving Reviews and Drifting 39
Incidents Caused By Traffic Jams 40
Conclusion 41
Reference List 43

Introduction

The United Arab Emirates is known for its mission of building a safe environment through safety laws, innovations, and technologies. Yet, traffic incidents remain a common concern, especially in the emirate of Dubai, due to its large and dense population [1] even after constant efforts and long-term plans.Authorities have repeatedly taken action to combat traffic incidents, such as using AI for monitoring and reporting, planning a new metro line, as well as promoting the use of public transportation. Officials also implemented many laws to prevent accidents, such as speed limiting, avoiding distractions while driving, preventing driving under influence, and constantly having patrolling policemen. Undoubtedly, the efforts are paying off, with death rates dropping by over 90% across the past 2 decades, reaching a record low of 1.6 deaths per 100,000 people and surpassing the originally targeted rate of 2 deaths [2]. However, even after such measures, Dubai may sometimes still surpass over 10,000 incidents a month, and while many of them may not cause deaths, it is still a concern to be aware of, potentially risking health issues, psychological distress, financial troubles, and more.

Our project aims to use open-data provided by Dubai Police about traffic incident records to try and discover causes, relationships and solutions for this issue, while also trying to predict information about traffic incidents in the future to better identify what needs to be focused and improved. The main difficulty of predicting incidents is simply the infinite number of factors or variables that play a part, with some factors affecting a single car such as malfunctions or human error, and others being purely environmental such as storms and natural traffic congestions. The lack of data for many of these factors also has an impact on the difficulty of finding a solution. For example, if a car crash was to cause traffic across a highway, that could cause a chain reaction of multiple crashes over the day, yet discovering that a month later from basic records could prove difficult.

Our target goals include:

- Predicting missing severities
- Finding outliers in the dataset
- Predicting the traffic incidents of future months
- Identifying any correlation between attributes
- Identifying hotspots of interesting incident types

Using classification, clustering, outlier detection, and association techniques, we will be attempting to achieve our objectives and extract knowledge from our dataset.

Approach

Overview

We will be using a mixture of different data mining techniques including:

- Logistic Regression
- Linear interpolation
- Density Based Clustering
- Seasonal Autoregressive Integrated Moving Average
- Random Forests
- Categorical Imputation
- Correlation Matrices

With these techniques we hope to complete the objectives stated in the introduction.

Dataset

Our main dataset was acquired through the Dubai Police Open Data available on Dubai Pulse. The dataset describes 62,173 traffic incidents recorded in Dubai between from March 6th 2024 all the way up to October 23rd of the same year, with some gaps in between, most notably the lack of records between April 30th and June 10th. It's important to note that this data is updated daily with new records being added, however to stay consistent we decided to stick with the same dataset since we started working with it to maintain consistency.

The 5 attributes in the dataset are as follows:

- **acci_id:** a system-generated unique positive number used to identify the incident
- **acci_time:** a datetime object representing both the date and local time when the incident was registered by the system. We will be assuming that this time is accurate enough to consider the real time of the incident, ruling out any possible delays or differences between the incident happening and it being registered.
- **acci_name:** an Arabic description of the incident describing the general event and its severity, for example a severe collision between two vehicles.
- **acci_x:** latitude coordinates of the incidents location
- **acci_y:** longitude coordinates of the incidents location

Attribute	Unique Values Count	Sample Unique Values
acci_id	63142	[4649976584, 4649982133, 4649983700, 4649983943, 4649985299]
acci_time	62879	['23/10/2024 20:19:22', '23/10/2024 20:27:57', '23/10/2024 20:30:51', '23/10/2024 20:31:12', '23/10/2024 20:32:10']
acci_name	90	['صدم دراجة نارية - بليغ', 'صدم عمود - بسيط', 'الوقوف خلف المركبات (دبل بارك) - بسيط', 'اصطدام بين عدة مركبات - بسيط', 'اصطدام بين مركبتين - بسيط']
acci_x	49724	[25.28443999, 25.03105999, 24.97398999, 25.27101999, 25.23349999]
acci_y	50051	[55.38680999, 55.14942, 55.20736999, 55.42988, 55.3135]

Accidents and incidents are terms often used interchangeably, but they have distinct meanings in safety contexts. An accident refers to an unplanned event that results in damage, injury, or loss, typically with more serious consequences. On the other hand, an incident encompasses a broader range of events, including those with no immediate harm but the potential for future risk [3]. For our project, we use the term incident to include both simple and serious occurrences as defined by our dataset. While we acknowledge that these terms are often used interchangeably, our usage emphasizes the inclusivity of all event severities within the scope of analysis.

Pre-Processing

Removing Incident ID’s

Our first decision was to remove the *acci_id* attribute for our purposes since some algorithms such as information gain based decision trees may result in overfit models or inaccurate results.

```
#Read CSV data into panda dataframe
df = pd.read_csv("Traffic_Incidents.csv")

df1 = df.drop("acci_id",axis=1)
```

```
#Write to new CSV
df1.to_csv("Traffic_Incidents_1.csv", encoding='utf-8', index=False)
```

Translating Incident Names

As mentioned before, the *acci_name* attribute in the dataset was recorded in Arabic, and we made the choice to start by translating the values in this attribute to English so that we can identify the proper steps we should take for the rest of pre-processing and the overall project.

To translate, we considered using Google Translate's API, however it required creating developer accounts that would have set limits for the number of translation requests, and as we didn't want to get locked by the limit, we tried to find another solution. We identified a Python library called *googletrans*, which allows developers to use Google Translate without having to use the API directly, which fits our needs.

We used Pandas to read the dataset, collect unique *acci_name* values, and send a translation request through the *googletrans.Translator()* object. Once the requests were processed, we used Panda's *dataframe.replace* method to swap the Arabic values with the translated English values. We now had values such as "Collision between two vehicles - simple" filling our *acci_name* dimension.

```
def translate(to_translate):
    """Translates all the values from the passed list into a English

    Args:
        to_translate (list): list of values to be translated

    Returns:
        translated: list of translated values
    """
    #Initilize translator from googletrans library (whihc uses Google
Translate API)
    translator = Translator()

    translated = []
    counter = 1

    for s in to_translate:
        print(f'Translating {counter}/{len(to_translate)}')
        counter +=1
        #Translate the text and append it to result list
```

```
        translation = translator.translate(s,src='ar',target='en')
        translated.append(translation.text)
    return translated

#Read CSV data into panda dataframe
df = pd.read_csv("Traffic_Incidents_1.csv")

#Get list of unique values in acci_name column
original = list(df.acci_name.unique())

#Replace the original values in the dataframe with the translated values
using the translate function
print("Translating dataframe...")
df1 = df.replace(original,translate(original))

#Write to new CSV file
print("Writing to csv file...")
df1.to_csv("Traffic_Incidents_2.csv", encoding='utf-8', index=False)
```

Splitting Incident Names

The *acci_name* attribute served 2 purposes, containing both a description of the incident as well as a severity description, which could prove troublesome during the data mining part of the project as it will be less accurate for algorithms to identify relationships between each part of the attribute separately.

We solved this by splitting *acci_name* into *acci_desc* and *acci_severity* using Pandas using hyphens (“-”) as the delimiter as per the *description - severity* format common across the dataset.

```
#Read CSV data into panda dataframe
df = pd.read_csv("Traffic_Incidents_2.csv")

#Split column based on '-' delimeter
df[['acci_desc','acci_severity']] = df.acci_name.str.split('-
',expand=True)

#Remove old column
df1 = df.drop("acci_name",axis=1)

#Strip the description and severity
df1['acci_desc'] = df1['acci_desc'].str.strip()
```

```
df1['acci_severity'] = df1['acci_severity'].str.strip()

#Write to new CSV
df1.to_csv("Traffic_Incidents_3.csv", encoding='utf-8', index=False)
```

Standardizing Severity

We noticed that there was some inconsistency caused during the translation process of the dataset, as originally the severity was described as either simple or serious using 2 corresponding Arabic words, however our dataset now had 5 unique severity values being:

- “Serious”
- “Simple”
- Nan
- “Eloquent”
- “severe”

Ignoring the Nan value, we had synonymous severities that would be noisy if left unfixed, with those being “Severe” and “Eloquent”. We therefore used Pandas to change both of those severities into “Serious” with the aim of transforming our domain into a binary representation. However, we still had the Nan values to deal with.

```
#Read CSV data into panda dataframe
df = pd.read_csv("Traffic_Incidents_3.csv")

#Get list of unique values in acci_severity column
original = list(df.acci_severity.unique())

#Change all values that aren't 'simple' or 'serious' into 'serious'. Keep
missing severities the same.
original.remove('simple')
original.remove('serious')
original.remove(nan)

df1 = df.replace(original,'serious')

#Write to new CSV file
df1.to_csv("Traffic_Incidents_4.csv", encoding='utf-8', index=False)
```

Removing Unwanted Incidents

When exploring our data, we discovered that there are 3414 entries that did not have a severity assigned to them.

```
#checking how many rows have missing severity
df = pd.read_csv("Traffic_Incidents_3.csv")

#Check how many rows have emprt severities (True means no severity)
df1 = df['acci_severity'].isnull()
count = df1[df1 == True].value_counts()[True]
print(f"There are {count} inccidents with missing severities.")

#Check the descriptions of the missing severity incidents
null_rows = df[df.isnull().any(axis=1)]
print(null_rows["acci_desc"].value_counts())
```

The *acci_desc* and number of instances with missing severities can be seen in the table below.

Acci_desc (Incident Description)	Instances with missing severity
Illegal vehicles	3133
One or more people crossing from a place not designated for pedestrian crossing	217
Vehicle fire while driving	64

We decided that out of the above list we will only keep “Vehicle fire while driving” as:

- It’s the only one out of the three that has other records with severities included
- We wanted to focus on incidents that are on the road and related to the vehicles, meaning that the other 2 descriptions did not exactly match our targets

```
#read dataframe
df = pd.read_csv("Traffic_Incidents_4.csv")
#create a set of all incident descriptions with a null severity except
vehicle fire while driving
unwanted = set(null_rows['acci_desc'])
unwanted.remove('Vehicle fire while driving')

#remove the rows with these descriptions
for desc in unwanted:
    df = df[(df['acci_desc'] != desc)]
df.to_csv("Traffic_Incidents_5.csv", encoding='utf-8', index=False)
```

Splitting Incident Time

Another attribute that we deemed worthy of splitting was *acci_time* as it included both the date and time of the incidents. By splitting it we could perform a more accurate analysis of incidents and we would more easily be able to take subsets or, through leveraging the coordinate dimensions, also identify what incidents may have occurred together We used Pandas to split using space as the delimiter, and now had *acci_date* and *acci_time* attributes instead.

```
#Read CSV data into panda dataframe
df = pd.read_csv("Traffic_Incidents_5.csv")

#Split column based on ' ' delimeter
df[['acci_date','acci_time']] = df.acci_time.str.split(' ',expand=True)

#Write to new CSV
df.to_csv("Traffic_Incidents_6.csv", encoding='utf-8', index=False)
```

Standardizing Date

Dealing with date values is difficult when they are not parsed into a datetime object, and while this is done while reading the CSV file into a dataframe object, it's better to convert them and write them into the CSV so that we have a standardized format for them throughout the project

```
#Read CSV data into panda dataframe
df = pd.read_csv("Traffic_Incidents_6.csv")

#Convert the format of acci_date into datetime. This step is not
neccessary but is for standarizing the format of dates
df['acci_date'] = pd.to_datetime(df['acci_date'],dayfirst=True)

#Write to a new CSV file
df.to_csv("Traffic_Incidents_7.csv", encoding='utf-8', index=False)
```

Filling Missing Severities

Once we decided which types of accidents we removed, it was time to replace the records including an empty severity, which is represented by nan in Panda dataframes. We attempted to use multiple methods, which will be discussed in further detail in the experimental analysis section of this report.

Dealing with noise and outliers

While plotting our dataset, we noticed that there are some points that seemed extremely far from the majority, and this meant that we had to perform outlier analysis and identify them to better understand how we should deal with them. This will be covered in detail in the experimental analysis section.

One record that stood out between all the outliers had both its coordinates set to 0, which is far outside of the coordinate range for Dubai, therefore we decided to use the mean of all other coordinates to update it. Note that this snippet uses *Traffic_Incidents_8.csv* which was created while filling missing severities later in experimental analysis.

```
df = pd.read_csv("Traffic_Incidents_8.csv")
# Find the mean of the columns excluding the incorrect record
mean_x = df.loc[df['acci_x'] != 0, 'acci_x'].mean()
mean_y = df.loc[df['acci_y'] != 0, 'acci_y'].mean()

print(f"Calculated coordinates: ({mean_x},{mean_y})")
# Replace 0 values with the calculated means
df.loc[df['acci_x'] == 0, 'acci_x'] = mean_x
df.loc[df['acci_y'] == 0, 'acci_y'] = mean_y
```

This sets the points coordinates to (55.29489228735156, 25.152824680363487). Remember that acci_x and acci_y represent the latitude and longitude respectively, therefore x-axis and y-axis throughout the project are represented by the opposite variable.

Another set of records for incidents in March and April had dates that were labelled with the year 2023, and as we don't have valid proof whether or not this data is correct or not, we decided to keep them and consider them as 2024 records, and this was to maintain a large dataset.

```
filter = df["acci_date"].apply(lambda x: x.replace(year=2024))
df["acci_date"] = np.where(df["acci_date"].dt.year == 2023, filter,
df["acci_date"])
df.to_csv("Traffic_Incidents_11111.csv", encoding='utf-8', index=False)
```

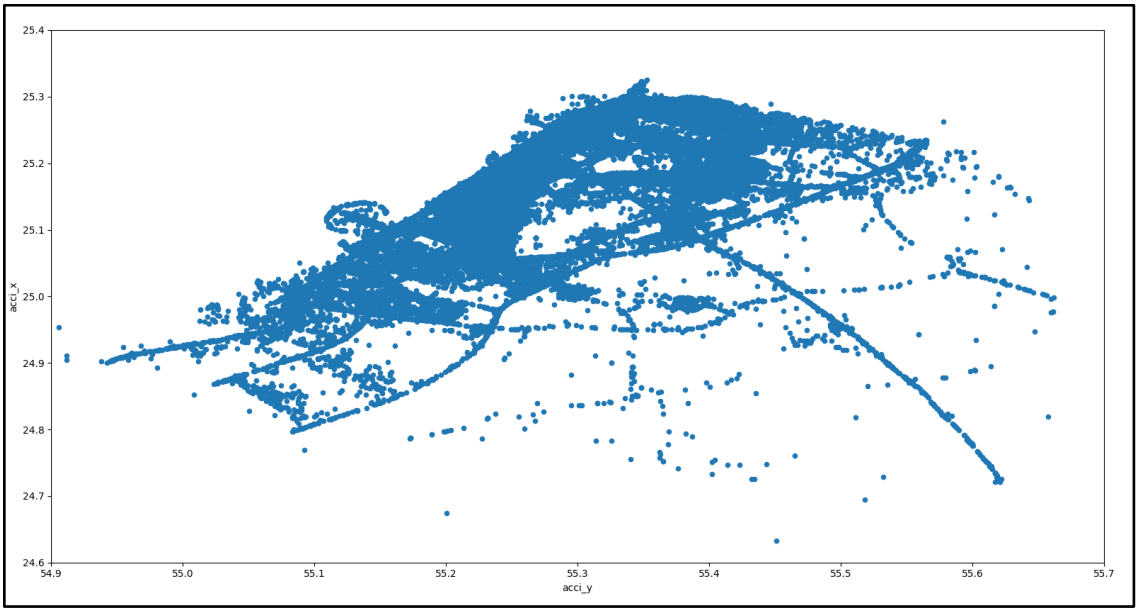
Visualizing the dataset

At this point we had finished pre-processing the data, and now we are able to visualize the dataset we are working with more accurately. Below are some of the diagrams and graphs we created to visualize different attributes and characteristics.

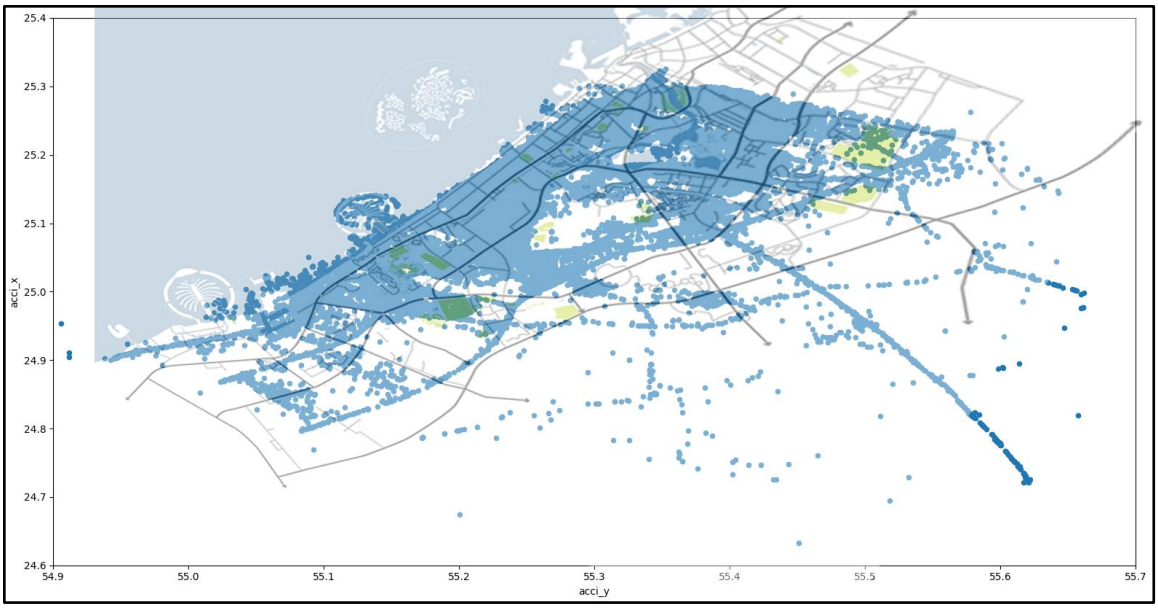
Plotting the Incidents to Create a Map-like Model

Plotting the records using the x and y coordinates allows us to visualize the spatial and geographical spread of the data across the longitude and latitude. To help with the visualization, the plot is limited to the main city of Dubai, which only excludes a couple of outliers (discussed later).

```
df = pd.read_csv("Traffic_Incidents_9.csv")
df.plot(kind = 'scatter', x = 'acci_y', y = 'acci_x', xlim=(54.9, 55.7),
ylim=(24.6, 25.4))
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Traffic Incident Distribution")
plt.savefig("Incident_Distribution_Map.png")
```

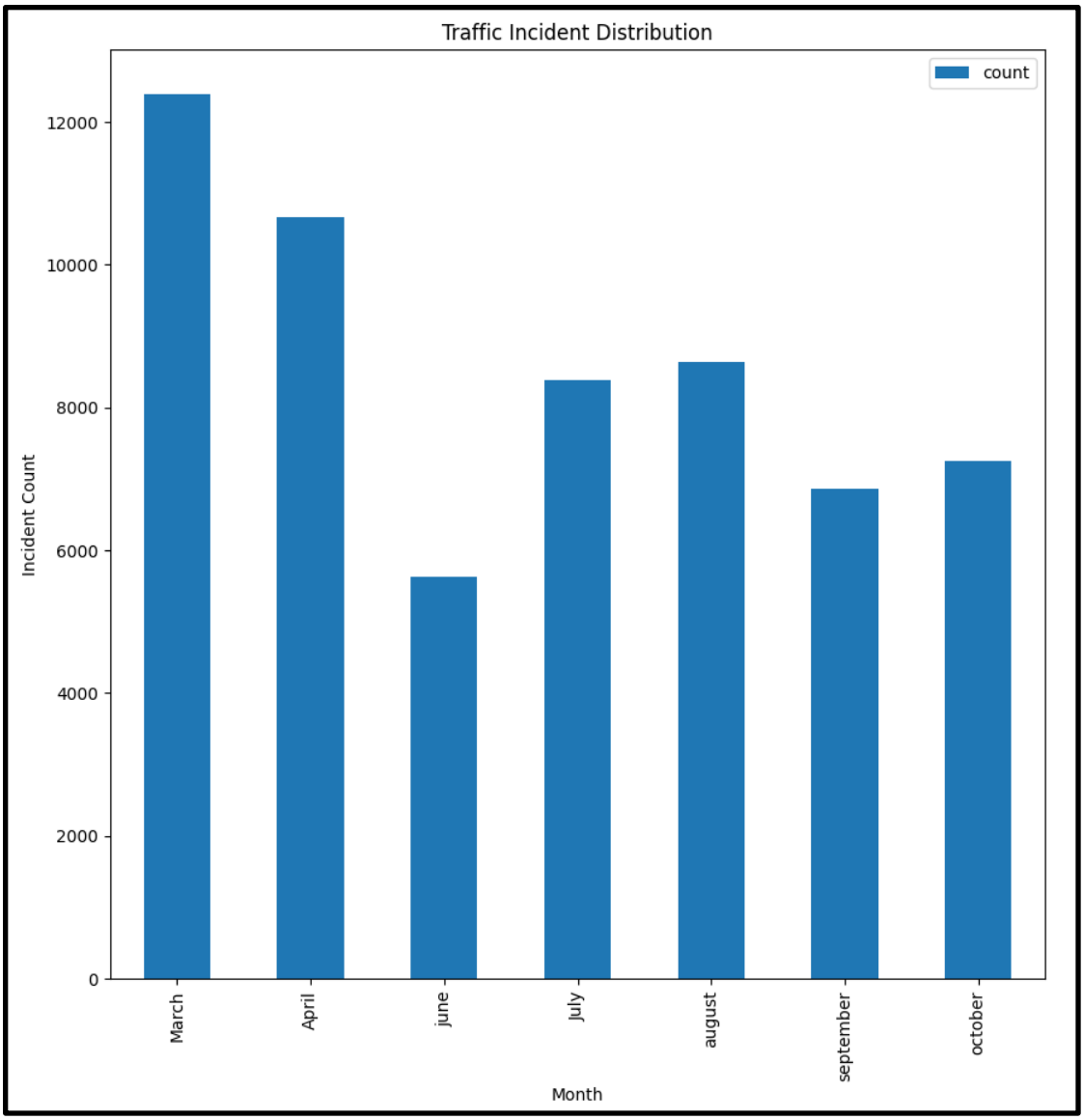


Through this plot we can make out a general image of Dubai and its streets, some areas can also be recognized such as the palm. As a small support we added an outline of Dubai using Photoshop to help connect the similarity. Please note that the map and the plots are not to scale, but just for a visual demonstration.



Traffic Incident Distribution by Month

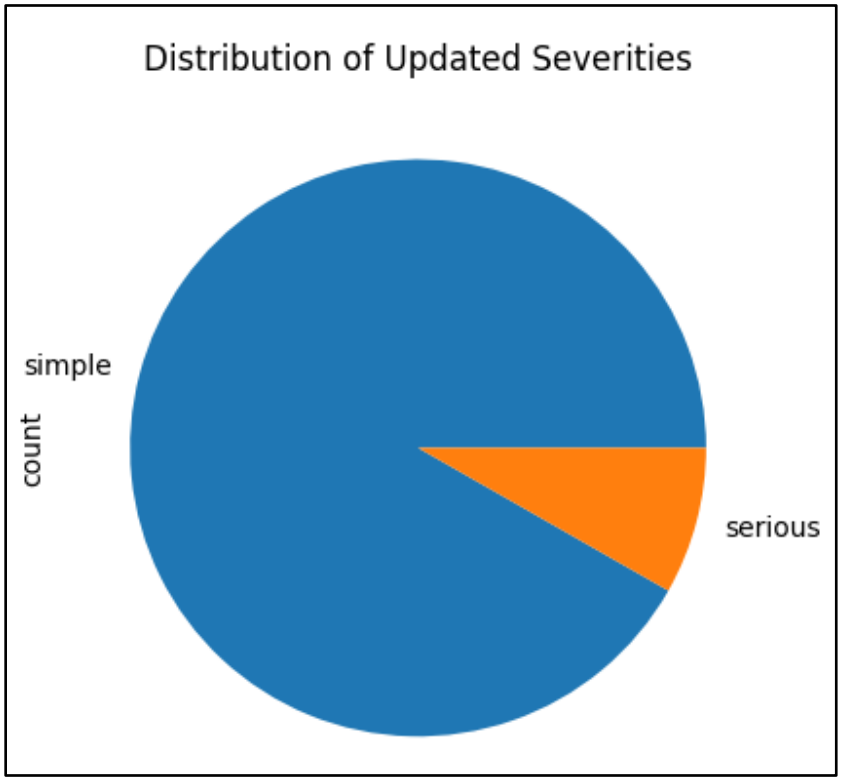
```
df = pd.read_csv("Traffic_Incidents_9.csv",parse_dates=['acci_date'])
df['Month'] = df['acci_date'].dt.month
df.set_index('Month',inplace=True)
grouped = df.groupby('Month').size().to_frame('count').reset_index()
grouped['Month'] = ['March', 'April', "june" , "July", "august",
"september", "october"]

grouped.plot(kind='bar', x='Month', y='count')
plt.xlabel("Month")
plt.ylabel("Incident Count")
plt.title("Traffic Incident Distribution")
plt.savefig("Incident_Distribution_By_Month.png")
```


Traffic Incident Distribution
count
12000
10000
8000
6000
4000
2000
0
Incident Count
March
April
june
July
august
september
october
Month

Distribution of Updated Severities

We found that the probability of a record being simple is 91.86% while the probability of it being severe is only 8.14%, meaning that there was a large skew in our dataset towards simple incidents.

```
df = pd.read_csv("Traffic_Incidents_9.csv")
#visualize the probability
df['acci_severity'].value_counts().plot(kind = 'pie')
plt.title("Distribution of Updated Severities")
```



Experimental Analysis

In this section, we will be discussing the data mining techniques and procedures that we used to achieve our objectives, as well as the decisions that we chose to make.

Predicting Missing Severities

Referring back to the pre-processing section, we identified records that had missing severities and decided to remove most of them from our dataset by checking what incident descriptions align with our criteria of “Incident” and which ones don’t. We concluded on only keeping null records that had the description “Vehicle Fire while driving”.

Our next step was to fill the empty severities in these null tuples. As discussed earlier, our dataset has a much larger quantity of “simple” records as opposed to “serious” ones, and furthermore, the dataset had “Vehicle Fire while driving” records that had severities, but they were all labelled serious. These 2 factors meant that relying on the whole dataset would most likely result in a classification of “simple”, while relying only on the previous records would result in a guaranteed “serious” classification. To combat this, we tried to use multiple classification methods in hopes to identify relationships or associations between the different attributes that would result in a more realistic and balanced classification. Results we find that have a mixture of both classes will be considered good, while results leaning towards 1 class will be considered unideal.

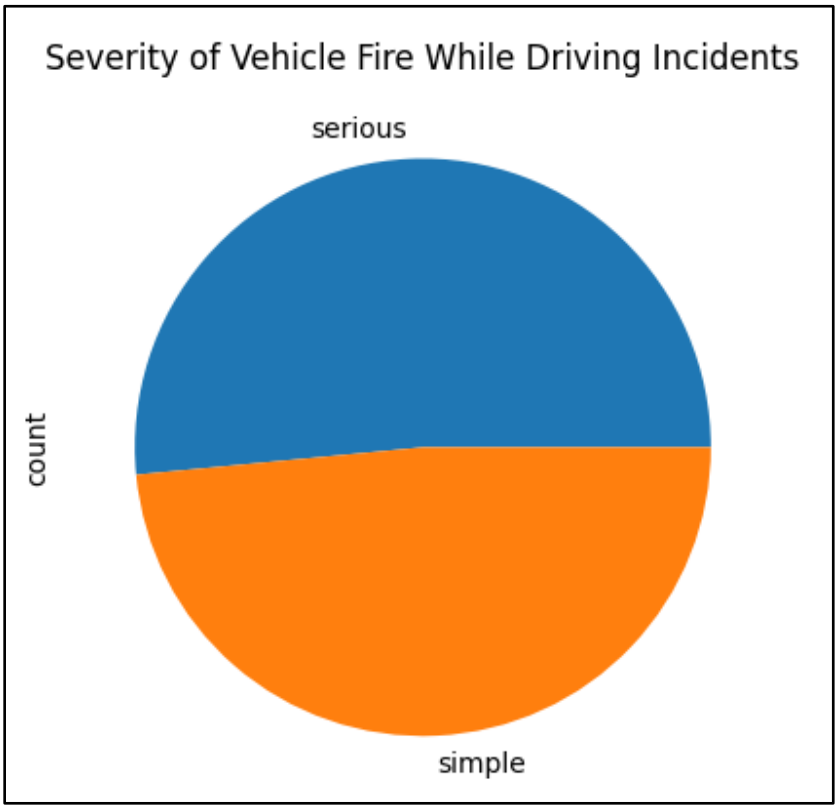
Note that this was done during the pre-processing procedures and therefore deals with the file *Traffic_Incidents_7.csv*.

Method 1: Using a Categorical Imputer

Our first idea was to use a Categorical Imputer, which would take in the dataset and make predictions on the null values within the data. While there are Imputers in sklearn libraries, we needed one that supports categorical values. We found a library called sklearn_pandas that included a categorical imputer model we could use.

```
df =
pd.read_csv("Traffic_Incidents_7.csv",parse_dates=['acci_time','acci_date'
])

imputer = CategoricalImputer()

#feed the dataset to the imputer and it will predict the null severities
predictions = imputer.fit_transform(df['acci_severity'])
df['acci_severity'] = predictions
df.to_csv("Traffic_Incidents_imputer.csv", encoding='utf-8', index=False)
```

However, as expected the predictions all came out to be “simple”, as this imputer relied on all the records in the dataset, which were dominated by these simple incidents.



Method 2: Using Logistic Regression

Logistic regression is a classification technique that’s effectively used with large data sets and binary class labels. It works by performing a linear regression and running it through a logistic (or sigmoid) function that outputs a value between 0 and 1, with anything equal to or above 0.5 being classified as 1, and anything below being classified as 0.

Our biggest challenge and limitation when using logistic regression from sklearn is that it does not support categorical and datetime attributes, meaning that it will only be able to rely on the coordinates for forming a prediction

```
df =
pd.read_csv("Traffic_Incidents_7.csv",parse_dates=['acci_time','acci_date'
])

#split the dataset into training and testing sets
train = df[df['acci_severity'].isnull()==False]
```

```
test = df[df['acci_severity'].isnull()==True]

#drop any column that doesn't contain floats or int values as sklearn
models do not support other types
X_train =
train.drop(['acci_time','acci_severity','acci_desc','acci_date'],axis=1)
X_test =
test.drop(['acci_time','acci_severity','acci_desc','acci_date'],axis=1)
Y_train = train['acci_severity']
Y_test = test['acci_severity']

# fit the model with data to train it
model = LogisticRegression()
model.fit(X_train, Y_train)

#make and print the predictions
predictions = model.predict(X_test)
print(predictions)
```

Once again as expected, all the resulting predictions were simple, without any “Serious” classifications.

Method 3: Enhancing Logistic Regression

As a final attempt to use logistic regression, we attempted to enhance it by converting both date and time into a timestamp value, which is accepted by logistic regression, therefore allowing us to leverage any possible relationship between date & time and the severity of the incidents.

```
df =
pd.read_csv("Traffic_Incidents_7.csv",parse_dates=['acci_time','acci_date'
])

#transform the time and date into floats
df['acci_time'] = df['acci_time'].apply(lambda x: x.timestamp())
df['acci_date'] = df['acci_date'].apply(lambda x: x.timestamp())

#split the dataset into training and testing sets
train = df[df['acci_severity'].isnull()==False]
test = df[df['acci_severity'].isnull()==True]

#drop any column that doesn't contain floats or int values as sklearn
models do not support other types
```

```
X_train = train.drop(['acci_severity','acci_desc'],axis=1)
X_test = test.drop(['acci_severity','acci_desc'],axis=1)
Y_train = train['acci_severity']

# fit the model with data to train it
model = LogisticRegression()
model.fit(X_train, Y_train)

#make and print the predictions
predictions = model.predict(X_test)
print(predictions)
```

However, this did not help balance the predictions and also resulted in only “simple” classifications.

Method 4: Creating a Random Forest

Our final attempt at creating balanced predictions was to use RapidMiner to generate a Random Forest to complete this classification task. Random Forest is an ensemble method that creates multiple decision trees that are trained on random subsets of the data, and then combines the predictions of all of them into a final result, and this makes it robust against overfitting.

Our random forest model was set to have 100 decision trees, a maximum depth of 10, and used the gain ratio criterion. The resulting predictions came out to be all “serious”, with all predictions having a confidence over 0.9. While these predictions are still unideal as they are not varied or balanced, they are more accurate than the previous predictions since they rely on making decisions based on all attributes rather than only numerical values. Having “serious” classes also makes more sense than “simple” considering that all of the classified records with this incident type are serious.

Evaluation metrics of the random forest (considering ‘simple’ is the positive class):
Accuracy: 94.54%
Precision: 94.39%
Recall: 99.99%
F-measure:97.11%

We updated our dataset manually to keep everything updated in the Python code.

```
df = pd.read_csv("Traffic_Incidents_7.csv",)
#fill all null severities with the value serious
df["acci_severity"].fillna("serious", inplace = True)
df.to_csv("Traffic_Incidents_8.csv", encoding='utf-8', index=False)
```

Outlier Detection Using Density Based Clustering

To continue our analysis of this dataset, we wanted to identify and deal with outliers by using clustering algorithms. For evaluating results, we will be considering the number of clusters or outliers generated as well as the inter-cluster similarity based on distance.

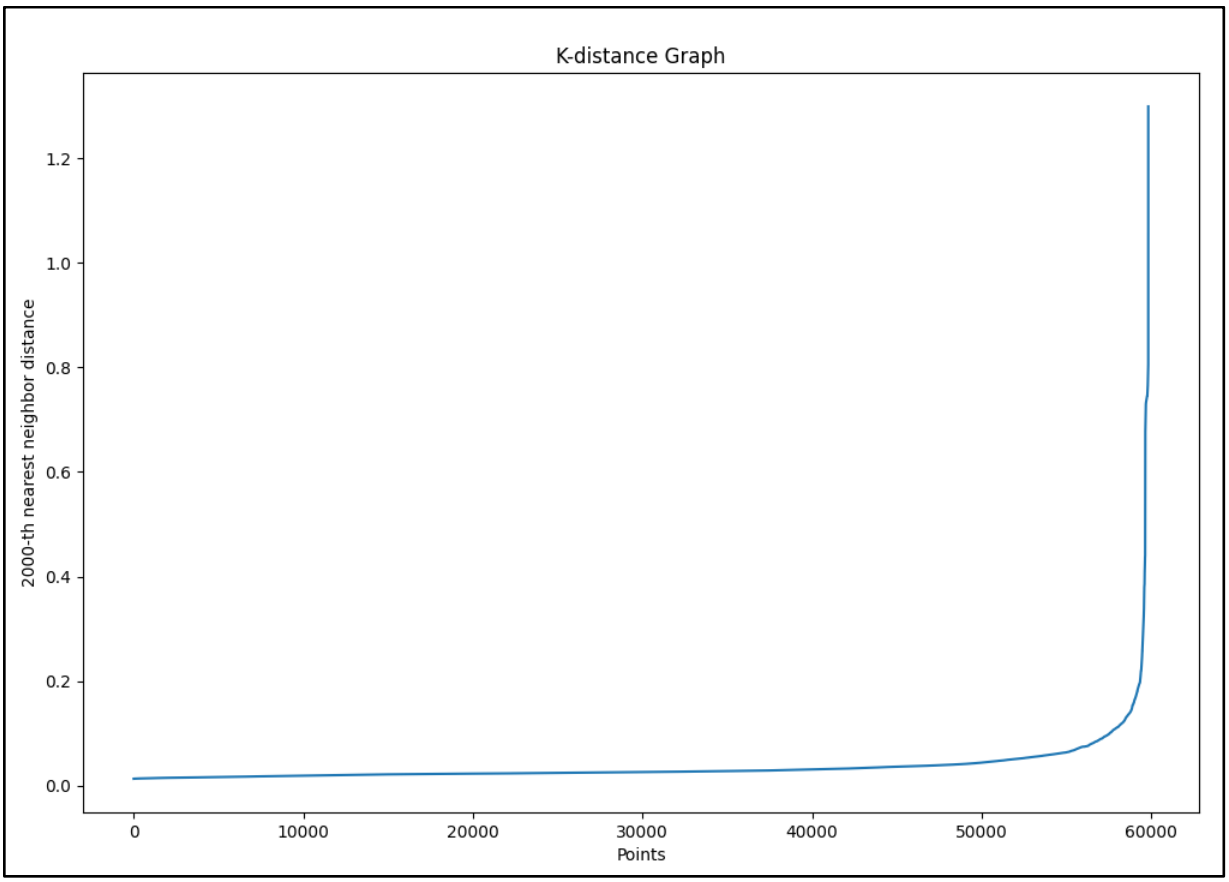
As seen in the visualizations earlier, plotting our points results in very closely placed points that may be difficult to cluster separately, especially considering the data sample and our limitations, including processing time and available hardware resources. This means that using clustering methods like K-means would be unideal, considering the time we will need to test multiple k values and identify a satisfactory result, which will also be affected by the outliers we have yet to deal with.

We decided to proceed using DBSCAN as our clustering technique since it does not require a pre-defined number of clusters and can detect non-convex shaped groups, which could prove helpful in discovering knowledge about the locations of the incidents

Using DBSCAN to Identify Outliers

Before identifying outliers we needed to identify the proper epsilon distance for the algorithm, and to do this we used the elbow method. We attempted to use different values for K and eventually settled on using 2000, as going any further resulted in increased processing time and memory utilization.

```
df = pd.read_csv("Traffic_Incidents_9.csv")
df1=df[['acci_x','acci_y']]

#calculate distance of k'th nearest neighbor
k=2000
neigh = NearestNeighbors(n_neighbors=k)
neigh.fit(df1)
distances, _ = neigh.kneighbors(df1)
#sort the distances
distances = np.sort(distances[:, k-1])
#plot
plt.figure(figsize=(12, 8))
plt.plot(distances)
plt.xlabel('Points')
plt.ylabel(f'{k}-th nearest neighbor distance')
plt.title('K-distance Graph')
plt.savefig("K-distance_graph.png")
```

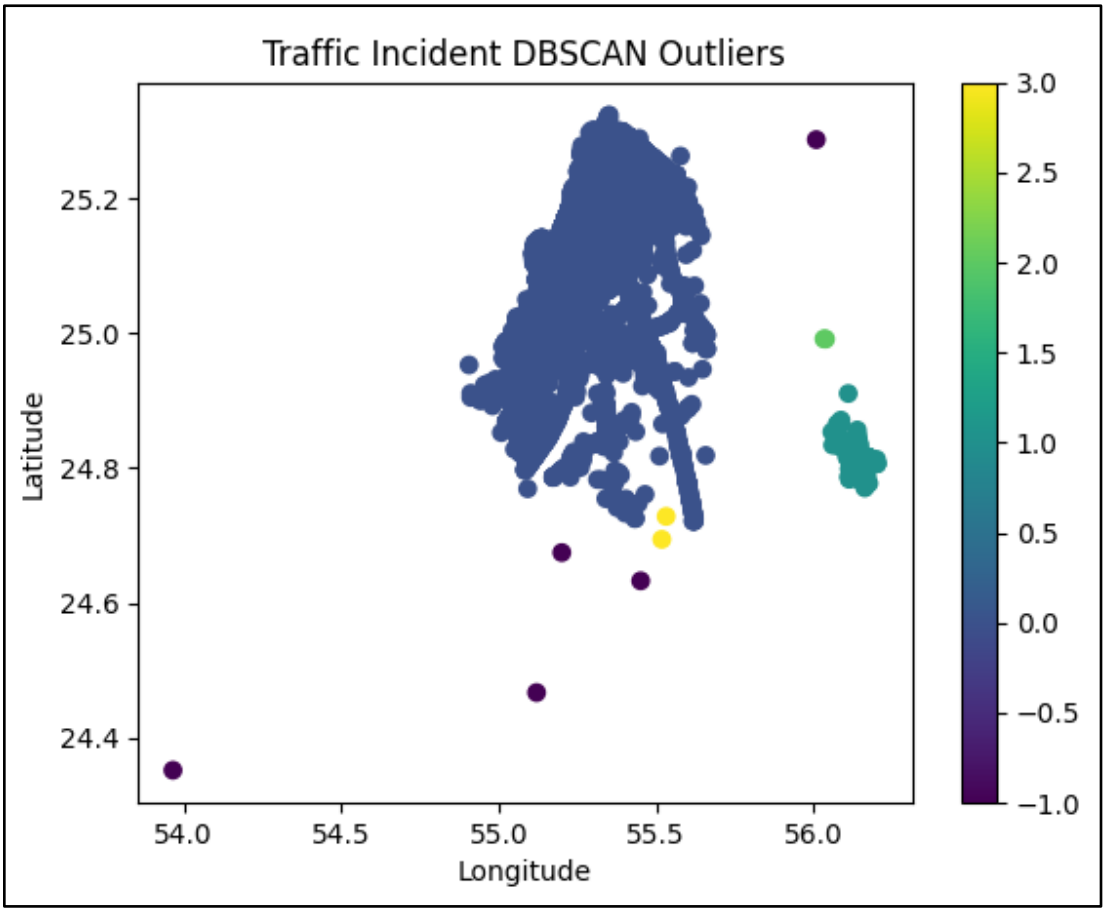


Our graph was clear enough to identify an elbow we can use to test epsilon values (when the plot shows up after running the code we are able to hover over the elbow and see the distance value) and we settled on using 0.1 as our epsilon distance.

Upon attempting to perform DBSCAN clustering with an epsilon distance of 0.1 and a minimum number of samples set to 2, we were unfortunately stopped by not having enough memory, therefore we attempted to decrease the epsilon distance by small amounts. Eventually we were able to run the model with eps=0.07, which gave us the following scatter plot.

```
df = pd.read_csv("Traffic_Incidents_9.csv")
# create the training set
train = df[['acci_x','acci_y']]

# Most satisfactory parameters for outliers based on elbow method
dbscan = DBSCAN(eps=0.07, min_samples=2)
clusters = dbscan.fit_predict(train)

#plot
plt.figure(figsize=(12, 8))
```

```
scatter = plt.scatter(train['acci_y'], train['acci_x'], c=clusters,)
print(f"DBSCAN identified {list(clusters).count(-1)} outliers")
plt.colorbar(scatter)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Traffic Incident DBSCAN Outliers")
plt.savefig("Traffic_Incident_DBSCAN_Outliers.png")
plt.show()
```



Fortunately this was satisfactory for us, as we were able to clearly identify the 5 detected outliers. The diagram clearly shows major clusters and separated outliers, and by further looking into the coordinates of these clusters and outliers we identified the following:

- Most outliers were incidents that occurred in the desert, non-dense areas of the UAE
- The small cluster on the right side of the diagram matches the coordinates of Fujairah, while the outlier on the far bottom left was past Abu Dhabi, indicating that Dubai Police operates, or at least keeps records of, incidents that happen even outside of Dubai.

Assuming the above inferences are correct, it could be beneficial for Dubai Police to create

dedicated stations or units that monitor other emirates rather than having information from other emirates in the same dataset. This also means that a large number of these records in the future could cause a skew in data or statistics, which is unideal for data mining and statisticians, therefore it would be beneficial to remove them from the dataset and instead keep this information in the police departments of the respective emirates.

Removing Outliers

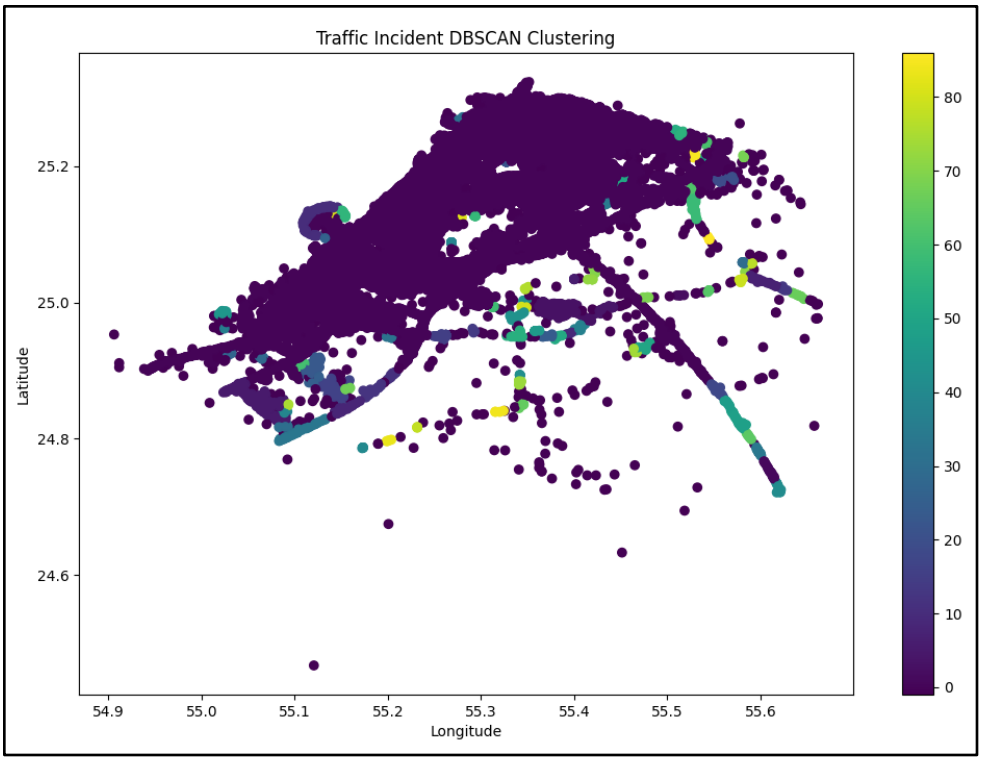
While these records exist in the official dataset, we would like to only focus on incidents occurring in the emirate of Dubai, therefore we will be removing records that have a longitude lower than 54.5 or greater than 56, keeping only outliers that are still present in Dubai, even if it's outside the main city.

```
df = pd.read_csv("Traffic_Incidents_9.csv")
df = df[(df['acci_y'] >= 54.5) & (df['acci_y'] <= 56) ]
df.to_csv("Traffic_Incidents_10.csv", encoding='utf-8', index=False)
```

Using DBSCAN to Identify Incident Clusters

Once all outliers are removed or fixed we can now identify clusters in our data using the longitude and latitude to find any hidden information.

```
df = pd.read_csv("Traffic_Incidents_10.csv")
# create the training set
train = df[['acci_x','acci_y']]

# Most satisfactory parameters based on our testing
dbscan = DBSCAN(eps=0.005, min_samples=4 )
clusters = dbscan.fit_predict(train)
#plot
plt.figure(figsize=(12, 8))
scatter = plt.scatter(train['acci_y'], train['acci_x'], c=clusters)
print(f"DBSCAN identified {len(set(clusters))} clusters")
plt.colorbar(scatter)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Traffic Incident DBSCAN Clustering")
plt.savefig("Traffic_Incident_DBSCAN_Clustering.png")
plt.show()
```



In the scatter plot above we can see that one large cluster exists connecting most of the incidents in the main city areas, with small clusters being formed the further south you move. It's also noticeable that those separated by empty gaps also have their own clusters, this separation could be due to large empty deserts, water bodies, or human-made distances such as that in the palm.

Future Forecast for Incident Count By Month

When starting this project, we wanted to use statistical data posted by Dubai Police on incidents from previous years to fill missing years and predict forecasts for the future, however while working on this project the data seems to have been deleted or hidden away, and we are no longer able to perform yearly forecasts. As an alternative, we decided to predict forecasts for upcoming months.

Our general criteria for our predictions is that they should be reasonable when compared to the current data that we have, meaning that they should not be much higher or much lower. Furthermore these forecasts shouldn't be equal across multiple months, as that would be unrealistic and would mean no change.

Filling in the Month of May 2024

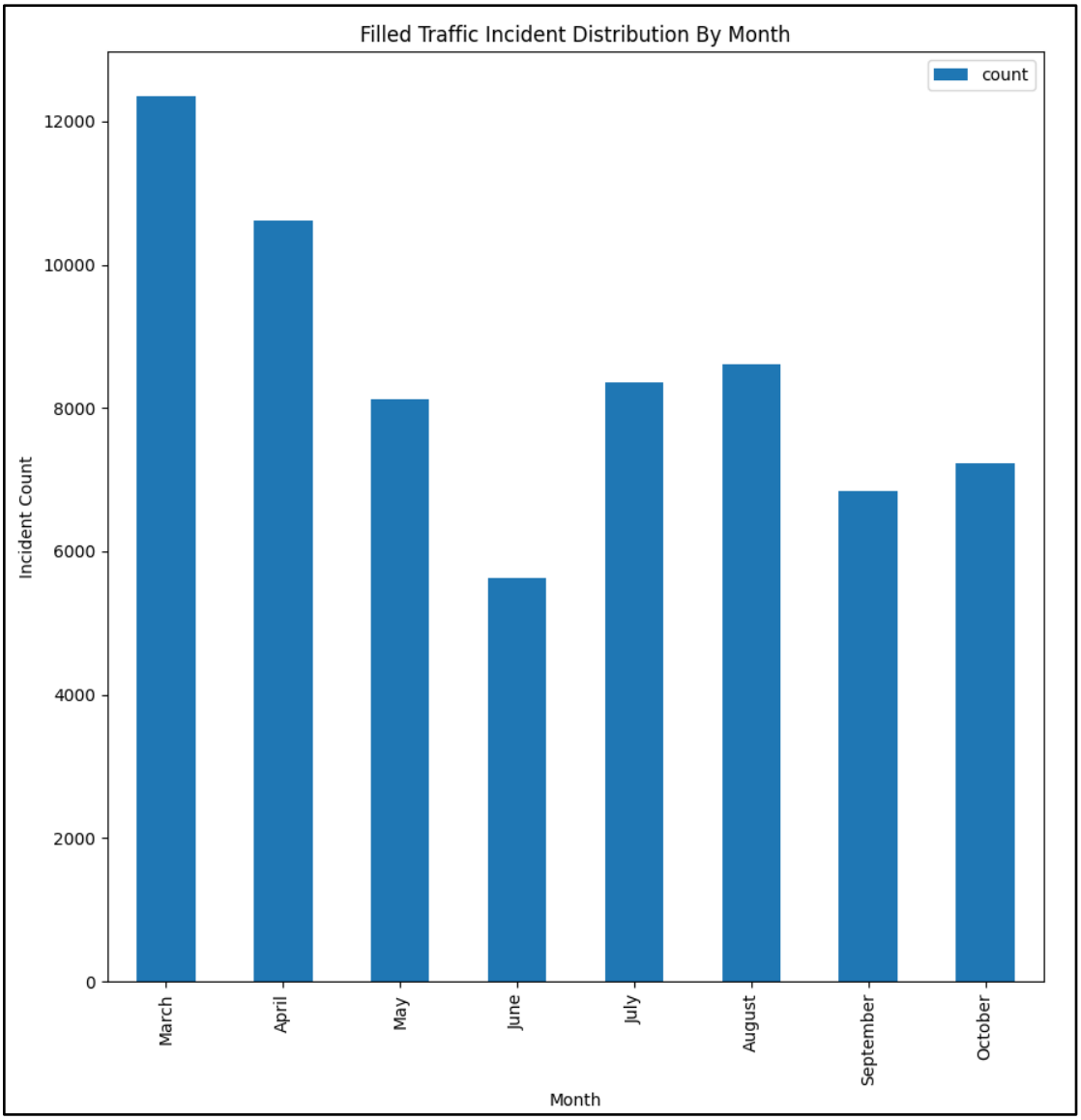
Before predicting future months we can improve our predictions by filling in data for the missing month of May. When considering what technique to use for this prediction, we noticed that there seems to be a somewhat linear line that can be drawn between the months of March, April and June, meaning that a linear prediction model could be accurate enough. We decided to use linear interpolation as it is a simple method that will use the values of April and June to fill in the missing count in between.

```
#Read in dataframe and group into months
df = pd.read_csv("Traffic_Incidents_10.csv",parse_dates=['acci_date'])
df['Month'] = df['acci_date'].dt.month
grouped = df.groupby('Month').size().to_frame('count').reset_index()

#Add a record for the month of May and sort the dataframe by month
grouped = pd.concat([grouped,pd.DataFrame({"Month":[5],'count':[nan]})],
ignore_index=True)
grouped = grouped.sort_values(by='Month')

#Use linear interpolation to predict the number of incidents in the month
of may
grouped = grouped.interpolate(method='linear')
grouped['Month'] = ['March', 'April', 'May', "June" , "July", "August",
"September", "October"]
print(grouped)
#Plot

grouped.plot(kind='bar', x='Month', y='count',figsize=(10, 10))
plt.xlabel("Month")
plt.ylabel("Incident Count")
plt.title("Filled Traffic Incident Distribution By Month")
plt.savefig("Filled_Incident_Distribution_By_Month.png")
plt.show()
grouped.to_csv("Monthly_Traffic_Incidents_Filled.csv", encoding='utf-8',
index=False)
```

The resulting counts can be seen below.

```
Month    count
March  12358.0
April  10624.0
May   8123.5
June   5623.0
```

```
July    8361.0
August    8607.0
September    6838.0
October    7228.0
```



We can see that there's a larger increase in incidents during the first half of the year with a minimum number of incidents occurring during June.

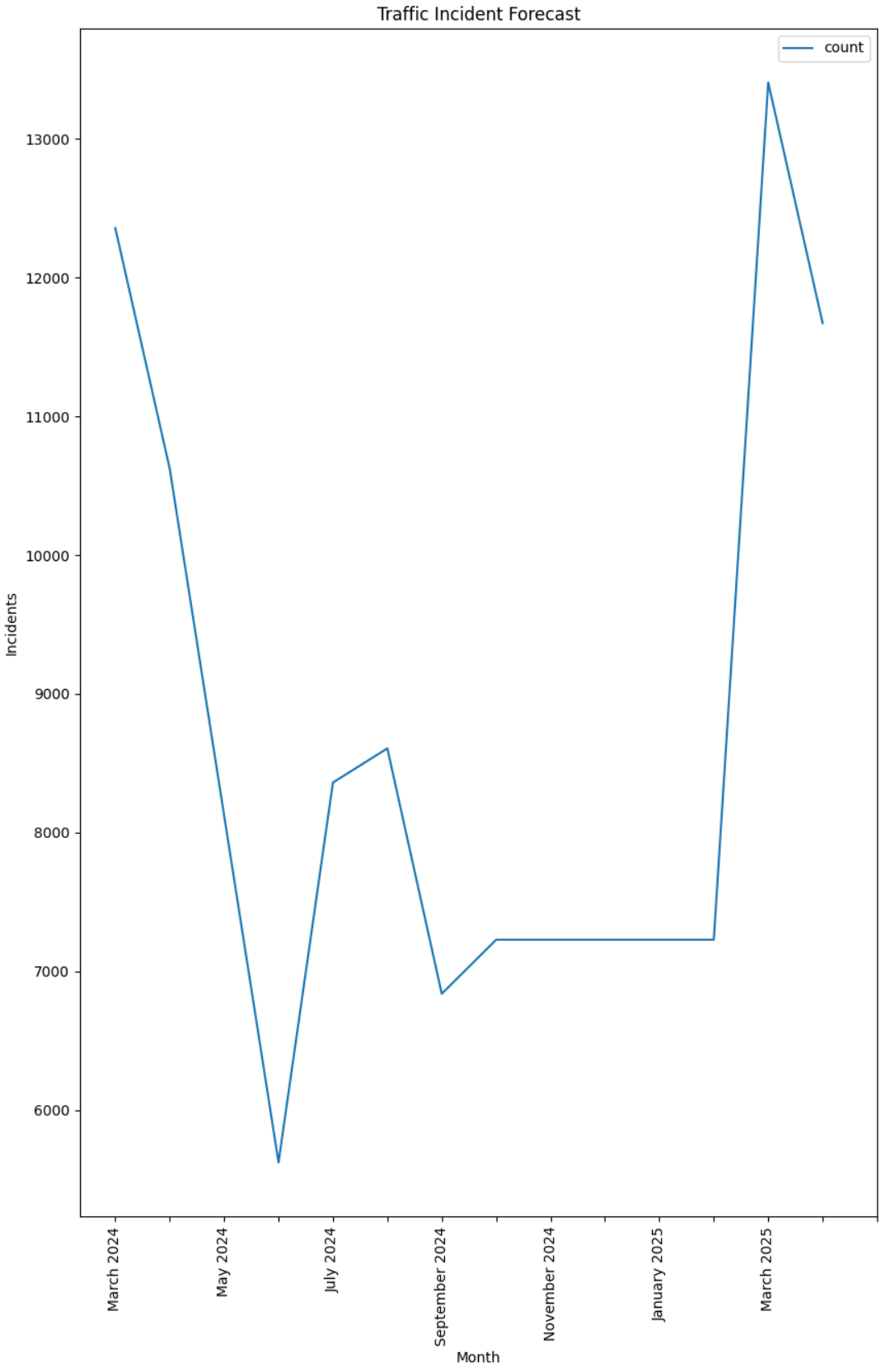
Creating Forecasts for Upcoming Months

Now that we filled in the missing month, we can move onto creating future forecasts for the number of incidents. To do this, we will be using the technique called Autoregressive Integrated Moving Average (SARIMA) which is a model for creating future forecasts based on the Autoregressive Integrated Moving Average (ARIMA) model, with the difference between them being that SARIMA considers seasonal patterns, which is better for our data as we can see patterns being displayed across groups of months.

Using SARIMA requires fitting the model with the seasonality, or how often is considered a season, which for annual data is usually set to 12. SARIMA also assumes that the data is stationary, meaning that its statistical properties remain constant over time, to test this we can use the Dickey-Fuller test. We used the adfuller function with the lag order being Akaike Information Criterion. Once we found that the data is stationary, we used applied SARIMA to create forecasts for the following 6 months.

```
df = pd.read_csv("Traffic_Incidents_10.csv",parse_dates=['acci_date'])
df['Month'] = df['acci_date'].dt.month
grouped = df.groupby('Month').size().to_frame('count').reset_index()
grouped = pd.concat([grouped,pd.DataFrame({"Month":[5],'count':[nan]})],
ignore_index=True)

grouped = grouped.sort_values(by='Month')
grouped = grouped.interpolate(method='linear')
grouped['Month'] = ['March', 'April', 'May', "June" , "July", "August",
"September", "October"]

grouped.reset_index()
#To apply SARIMA the data needs to be stationary
#Use Dickey-Fuller test
def check_stationarity(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    p_value = result[1]
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {p_value}')
    print('Stationary' if p_value < 0.05 else 'Non-Stationary')
check_stationarity(grouped['count'])

# Fitting the SARIMA model
p, d, q = 1, 1, 1
# 12 month seasons
```

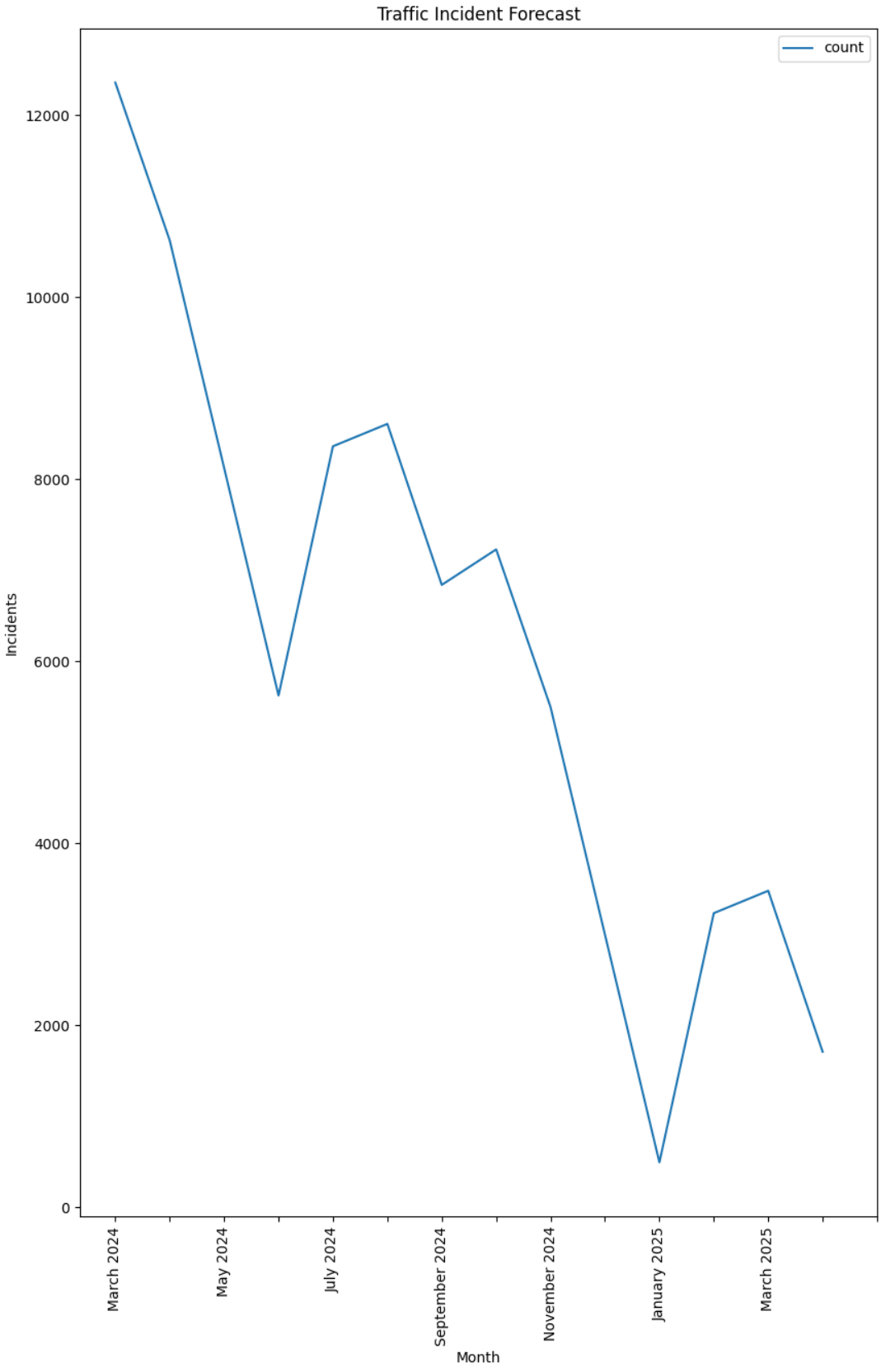
```
P, D, Q, s = 1, 1, 1, 12

model = SARIMAX(grouped['count'], order=(p, d, q), seasonal_order=(P, D,
Q, s))
results = model.fit()

#Set desired forecast period and create the prediction
forecast_periods = 6
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Concatinate the forecast with the grouped dataset
month = 11
for x in forecast_mean:
    new_row = pd.DataFrame({"Month":[month],'count':[x]})
    grouped = pd.concat([grouped,new_row], ignore_index=True)
    month +=1

#Visualize
grouped['Month'] = ["March 2024", 'April 2024', "May 2024","June 2024" ,
"July 2024", "August 2024", "September 2024", "October 2024",'November
2024','December 2024','January 2025','February 2025','March 2025','April
2025']
grouped.plot(kind='line', x='Month', y='count',figsize=(10, 15))
plt.xlabel("Month")
plt.ylabel("Incidents")
plt.title("Traffic Incident Forecast")
plt.legend()
plt.xticks(range(len(grouped['Month']) + 1), rotation=90)
plt.savefig("Monthly_Traffic_Incident_S12_Forecast.png")
plt.show()

grouped.to_csv("Monthly_Traffic_Incidents_S12_Forecast.csv",
encoding='utf-8', index=False)
```


Traffic Incident Forecast
count
13000
12000
11000
10000
9000
8000
7000
6000
Incidents
March 2024
May 2024
July 2024
September 2024
November 2024
January 2025
March 2025
Month

Unfortunately, the model seems to use the count of october as a prediction for the following couple of months before more accurate predictions are made, which is inaccurate. We attributed this to the model being trained using 7 months but considering a 12 month seasonality. Therefore we reattempted the procedure with a 7 month seasonality instead

```
df = pd.read_csv("Traffic_Incidents_10.csv",parse_dates=['acci_date'])
df['Month'] = df['acci_date'].dt.month
grouped = df.groupby('Month').size().to_frame('count').reset_index()
grouped = pd.concat([grouped,pd.DataFrame({"Month":[5],'count':[nan]})],
ignore_index=True)

grouped = grouped.sort_values(by='Month')
grouped = grouped.interpolate(method='linear')
grouped['Month'] = ['March', 'April', 'May', "June" , "July", "August",
"September", "October"]

grouped.reset_index()
#To apply SARIMA the data needs to be stationary
#Use Dickey-Fuller test
def check_stationarity(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    p_value = result[1]
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {p_value}')
    print('Stationary' if p_value < 0.05 else 'Non-Stationary')
check_stationarity(grouped['count'])

# Fitting the SARIMA model
p, d, q = 1, 1, 1
# 7 month seasons
P, D, Q, s = 1, 1, 1, 7

model = SARIMAX(grouped['count'], order=(p, d, q), seasonal_order=(P, D,
Q, s))
results = model.fit()

#Set desired forecast period and create the prediction
forecast_periods = 6
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()
```

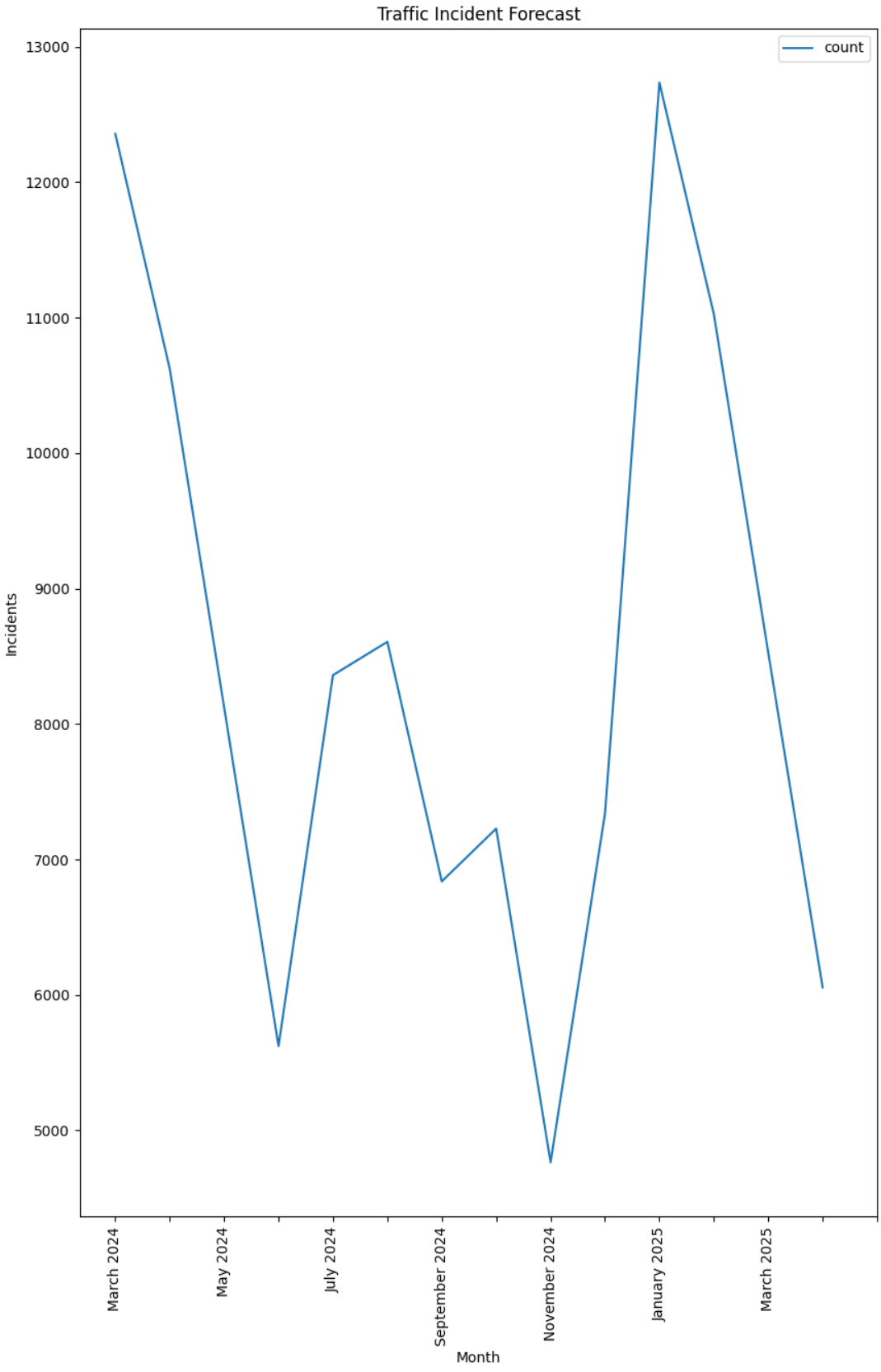
```
# Concatinate the forecast with the grouped dataset
month = 11
for x in forecast_mean:
    new_row = pd.DataFrame({"Month":[month],'count':[x]})
    grouped = pd.concat([grouped,new_row], ignore_index=True)
    month +=1

#Visualize
grouped['Month'] = ["March 2024", 'April 2024', "May 2024","June 2024" ,
"July 2024", "August 2024", "September 2024", "October 2024",'November
2024','December 2024','January 2025','February 2025','March 2025','April
2025']
grouped.plot(kind='line', x='Month', y='count',figsize=(10, 15))
plt.xlabel("Month")
plt.ylabel("Incidents")
plt.title("Traffic Incident Forecast")
plt.legend()
plt.xticks(range(len(grouped['Month']) + 1), rotation=90)
plt.savefig("Monthly_Traffic_Incident_S7_Forecast.png")
plt.show()

grouped.to_csv("Monthly_Traffic_Incidents_S7_Forecast.csv", encoding='utf-
8', index=False)
```


Traffic Incident Forecast
count
12000
10000
8000
6000
4000
2000
0
Incidents
March 2024
May 2024
July 2024
September 2024
November 2024
January 2025
March 2025
Month

This time the predictions were more varied, however they seemed unrealistic with a noticeable sharp decline in the first 3 months and a small rise after that. We ran the experiment once more with a 5 month seasonality.

```
df = pd.read_csv("Traffic_Incidents_10.csv",parse_dates=['acci_date'])
df['Month'] = df['acci_date'].dt.month
grouped = df.groupby('Month').size().to_frame('count').reset_index()
grouped = pd.concat([grouped,pd.DataFrame({"Month":[5],'count':[nan]})],
ignore_index=True)

grouped = grouped.sort_values(by='Month')
grouped = grouped.interpolate(method='linear')
grouped['Month'] = ['March', 'April', 'May', "June" , "July", "August",
"September", "October"]

grouped.reset_index()
#To apply SARIMA the data needs to be stationary
#Use Dickey-Fuller test
def check_stationarity(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    p_value = result[1]
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {p_value}')
    print('Stationary' if p_value < 0.05 else 'Non-Stationary')
check_stationarity(grouped['count'])

# Fitting the SARIMA model
p, d, q = 1, 1, 1
# 5 month seasons
P, D, Q, s = 1, 1, 1, 5

model = SARIMAX(grouped['count'], order=(p, d, q), seasonal_order=(P, D,
Q, s))
results = model.fit()

#Set desired forecast period and create the prediction
forecast_periods = 6
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()
```

```
# Concatinate the forecast with the grouped dataset
month = 11
for x in forecast_mean:
    new_row = pd.DataFrame({"Month":[month],'count':[x]})
    grouped = pd.concat([grouped,new_row], ignore_index=True)
    month +=1

#Visualize
grouped['Month'] = ["March 2024", 'April 2024', "May 2024","June 2024" ,
"July 2024", "August 2024", "September 2024", "October 2024",'November
2024','December 2024','January 2025','February 2025','March 2025','April
2025']
grouped.plot(kind='line', x='Month', y='count',figsize=(10, 15))
plt.xlabel("Month")
plt.ylabel("Incidents")
plt.title("Traffic Incident Forecast")
plt.legend()
plt.xticks(range(len(grouped['Month']) + 1), rotation=90)
plt.savefig("Monthly_Traffic_Incident_S5_Forecast.png")
plt.show()

grouped.to_csv("Monthly_Traffic_Incidents_S5_Forecast.csv", encoding='utf-
8', index=False)
```


Traffic Incident Forecast
count
13000
12000
11000
10000
9000
8000
7000
6000
5000
Incidents
March 2024
May 2024
July 2024
September 2024
November 2024
January 2025
March 2025
Month

We can also evaluate the model using mean absolute error and mean square error, where lower results are more accurate.

```
#Evaluate the model
observed = grouped['count'][-forecast_periods:]
mae = mean_absolute_error(observed, forecast_mean)
mse = mean_squared_error(observed, forecast_mean)
print(f'Mean Absolute errorE: {mae}')
print(f'Mean Squared Error: {mse}')
```

Both of our metrics came out as 0.0, meaning that the model was accurate based on the given data.

This time our forecasts seemed much more accurate, as they had both varied but still remained similar to the original data. We can notice a decline in incidents during the month of November and a massive spike at the start of 2025. Analyzing this data, we can infer that there are more incidents towards the end and start of years, meaning winter season, and this could be attributed to many factors such as storms, rain, events, and the increased number of rushing vehicles on the roads due to schools, universities, and jobs.

Correlation Analysis

We tried to identify any correlation between the different attributes in the dataset by using RapidMiner to build a correlation matrix. The process file is called *Correlation_Analysis.rmp* and the results can be seen below.

Attribut...	acci_ti...	acci_x	acci_y	acci_de...	acci_se...	acci_da...
acci_time	1	0.027	0.010	?	-0.014	?
acci_x	0.027	1	0.578	?	0.013	?
acci_y	0.010	0.578	1	?	-0.008	?
acci_desc	?	?	?	1	?	?
acci_sev...	-0.014	0.013	-0.008	?	1	?
acci_date	?	?	?	?	?	1

There are no huge notable correlations between the attributes, with the only mentionable one being the positive correlation between x and y coordinates, however this is not surprising given the coordinates where Dubai resides.

We also tried creating a correlation matrix in python but without nominal or categorical data, and ended up with similar results

```
df =
pd.read_csv("Traffic_Incidents_10.csv",parse_dates=['acci_date','acci_time
'])
df['acci_time'] = df['acci_time'].apply(lambda x: x.timestamp())
df['acci_date'] = df['acci_date'].apply(lambda x: x.timestamp())
df.drop(columns=['acci_desc','acci_severity'],inplace=True)
corr = df.corr(method = 'pearson')
corr
```

	acci_time	acci_x	acci_y	acci_date
acci_time	1.000000	0.026534	0.010049	-0.004917
acci_x	0.026534	1.000000	0.577530	-0.005427
acci_y	0.010049	0.577530	1.000000	-0.010770
acci_date	-0.004917	-0.005427	-0.010770	1.000000

Identifying Incident Hotspots

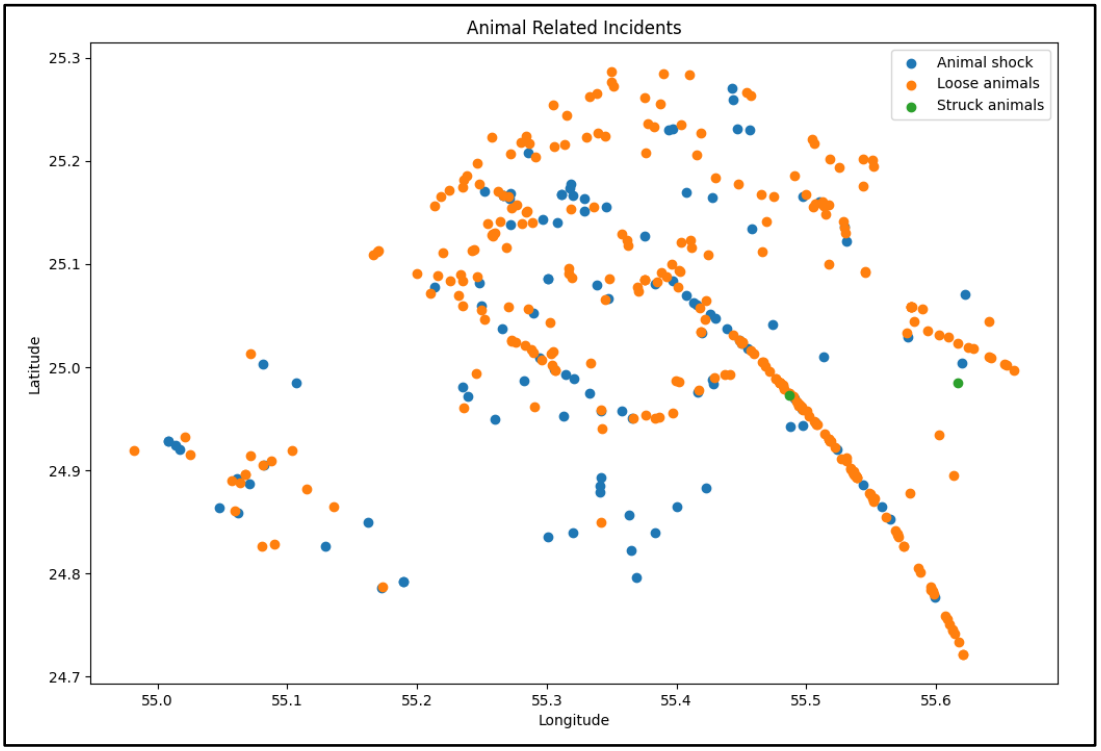
It's important to understand where certain types of incidents are most common, as that helps authorities prepare and take actions to provide a safer environment for drivers. We decided to look through some of the more interesting types of incidents and try to extract knowledge out of them.

Incidents Involving Animals

We noticed that there were several incident descriptions related to animals. Animals may not be a predictable factor for causing incidents, and that means it cannot be completely prevented, yet however animal life and ecosystems are important to care for, and therefore minimizing the harm done to and by them is important.

```
df = pd.read_csv("Traffic_Incidents_10.csv")
df1 = df[df['acci_desc'] == 'Animal shock']
df2 = df[df['acci_desc'] == 'Loose animals on the public road']
df3 = df[df['acci_desc'] == 'An animal was struck']
plt.figure(figsize=(12, 8))
plt.scatter(df1['acci_y'], df1['acci_x'])
plt.scatter(df2['acci_y'], df2['acci_x'])
plt.scatter(df3['acci_y'], df3['acci_x'])
```

```
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Animal Related Incidents")
plt.legend(['Animal shock','Loose animals','Struck animals'])
plt.savefig('Animal_Related_Incidents.png')
plt.show()
```



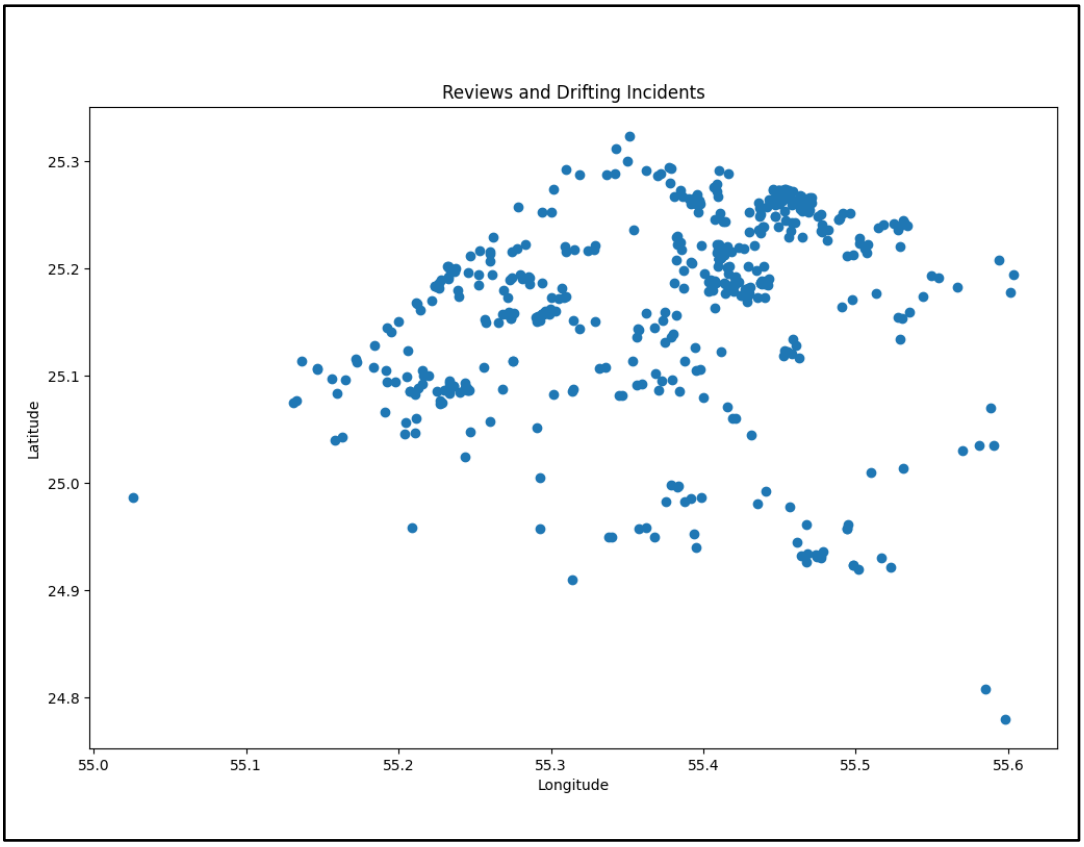
We can see that there's a street containing most of these animal related incidents, using an online map to search up the coordinates, we were able to identify that this is **Al Ain Road**. Authorities can consider setting up some sort of wildlife protection structures in that area to minimize future incidents. Wildlife protection organizations may also play a role in providing support for the wildlife around that area.

Incidents Involving Reviews and Drifting

Dubai has strict rules regarding careless driving behaviors such as drifting, yet they are commonly seen during car meets and group meetups. Even with all the laws and regulations, people do not understand the importance of careful driving and safety precautions, which therefore leads to a large number of incidents which we were able to visualize with the dataset.

```
df = pd.read_csv("Traffic_Incidents_10.csv")
df1 = df[df['acci_desc'] == 'Reviews and drifting']
```

```
plt.figure(figsize=(12, 8))
plt.scatter(df1['acci_y'], df1['acci_x'])
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Reviews and Drifting Incidents")
plt.savefig('Reviews_and_Drifting_Incidents.png')
plt.show()
```



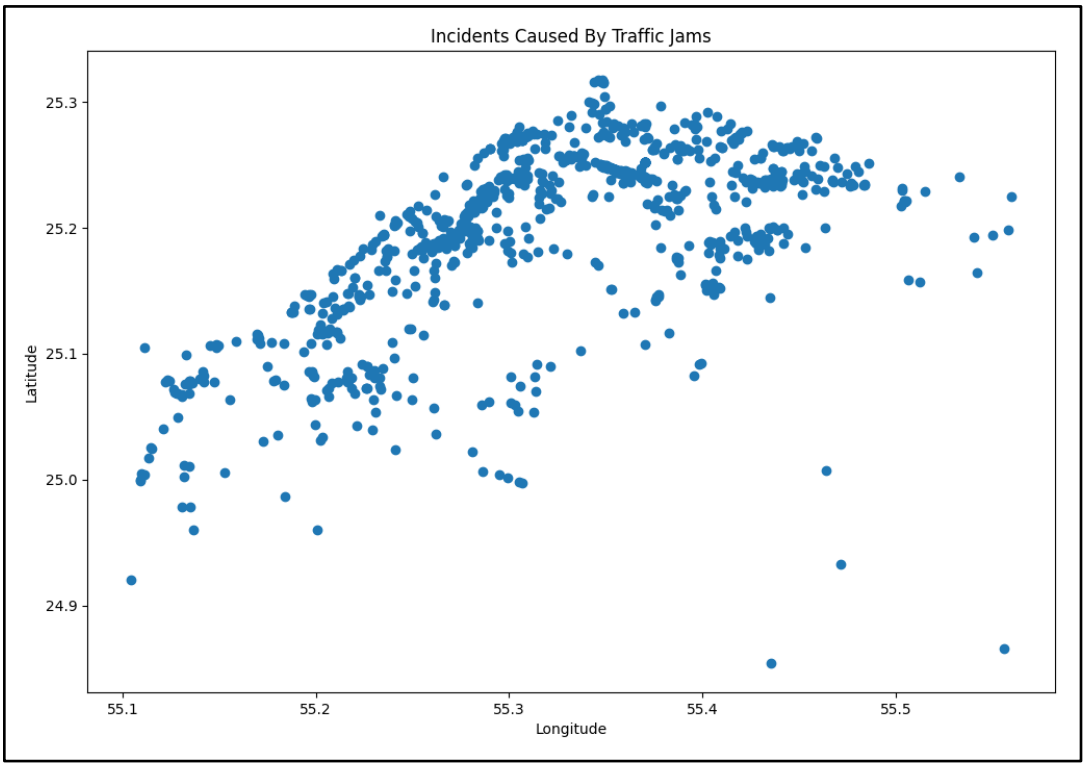
Using the same technique as before we traced the most common hotspots to **Mizhar 1** and **Al Warqa.** Authorities can use this information to scout and patrol common meetup and unsafe spots, which should have their security and surveillance improved.

Incidents Caused By Traffic Jams

Traffic jams were also identified as a larger contributor to traffic incidents in Dubai, therefore we wanted to identify their most common areas.

```
df = pd.read_csv("Traffic_Incidents_10.csv")
df1 = df[df['acci_desc'] == 'Traffic jam']
plt.figure(figsize=(12, 8))
```

```
plt.scatter(df1['acci_y'], df1['acci_x'])
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Incidents Caused By Traffic Jams")
plt.savefig('Incidents_Caused_By_Traffic_Jams.png')
plt.show()
```



As before we found that the majority of these incidents to be on **Sheikh Zayed Road (North)** and specifically around **Dubai Mall** and **Downtown**, this could mean that there is a poor traffic direction or a lack of alternative ways around that area.

Conclusion

With the application of data mining techniques, statistical modeling, and machine learning, even public datasets such as the one we dealt with can be used to uncover critical patterns, correlations, and trends that are not immediately evident. By analyzing these records, authorities can identify hotspots for specific types of incidents, predict periods of higher risk, and assess the impact of external factors such as weather or urban development. Moreover, future forecasts derived from such datasets enable proactive resource allocation, infrastructure planning, and policy-making aimed at reducing traffic-related risks. This study highlights how

data-driven approaches can transform raw information into actionable insights, empowering governments to enhance road safety and improve the overall quality of life for their citizens.

As for the data-mining techniques that we used, we were unfortunately unable to identify too much information from the categorical attributes due to them requiring more advanced models and resources. This project only touches the surface for the amount of knowledge to be found from incident records, and with more samples and recorded attributes, future projects may identify much more valuable information and pave the way for a more safe road experience for everyone.

Reference List

[1] References Tyres Online. (n.d.). **Accidents statistics UAE**. Retrieved from https://www.tyresonline.ae/en/blog/accidents-statistics-uae

[2] Gulf News. (2024). **Accident deaths in Dubai down 93% as RTA, police make roads safer**. Retrieved from https://gulfnews.com/uae/transport/accident-deaths-in-dubai-down-93-as-rta-police-make-roads-safer-1.103464894

[3] OSHA. (n.d.). **Incident vs. accident: What's the difference?** Retrieved from https://www.osha.com/blog/incident-accident-difference