Title: Optimizing Ambulance Allocation for Faster Response Times in Traffic Accidents

Emmanuel Asong, Agnes Nguenda, Nigel Nyajeka, Meakin Marange

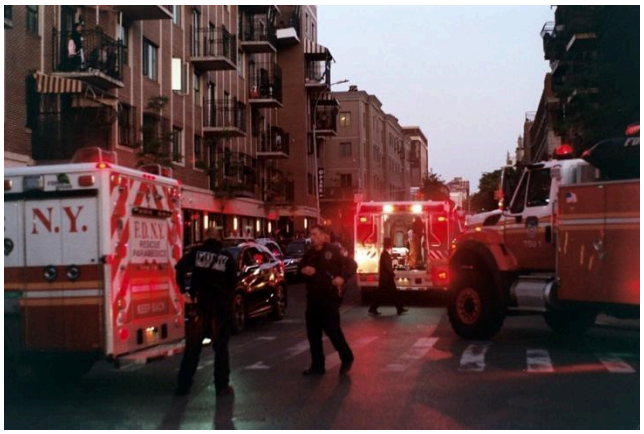


Table of Contents

Executive Summary 2

Business Problem Definition 2

Managerial Decisions 2

Data Sources 3

Data Exploration 3

Model Development and Analysis 6

Modeling Results and Performance Evaluation 11

Key Insights and Recommendations 12

Conclusion and Next Steps 13

Appendix 14

Executive Summary

Traffic accidents are a major public safety concern, and quick response times for emergency services can significantly impact the outcomes for those involved. In this project, we aim to optimize ambulance allocation to reduce response times, ultimately saving lives and improving the efficiency of emergency services. We narrowed our analysis to Washington DC in 2021 so that we can get a focused view on accidents in the capital, whilst focusing on MedStar Health as the ambulance provider of focus.

The primary objectives of this report are to:

- Define the problem of ambulance allocation in the context of traffic accidents.
- Describe the data sources and preprocessing steps.
- Explain the machine learning models used and the feature selection process.
- Evaluate the performance of different ambulance allocation strategies.
- Present key insights and recommendations for emergency service providers

Business Problem Definition

Private ambulance services play a critical role in providing emergency medical care and transportation services to patients. One of the key challenges these businesses face is optimizing ambulance routes to minimize response times and efficiently allocate resources and minimize costs. This project aims to develop a machine-learning model that predicts accident locations and develops strategic response times to help private ambulance services optimize their emergency vehicle routing and improve response times. From this project, ambulance companies can effectively respond to emergency calls and help allocate resources to enhance service delivery and minimize delays and casualties thus leading to better customer/ patient satisfaction. The goal is to minimize response times while maximizing the utilization of available ambulances. In this project, we will use machine learning to predict the location of traffic accidents and employ clustering techniques to optimize ambulance allocation.

Managerial Decisions

The main managerial decisions made during this project include:

- Identifying key factors that influence traffic accident occurrence.
- Determining the appropriate machine learning model for predicting accident locations
- Selecting the most important features of the model.
- Evaluating different ambulance allocation strategies (K-means clustering vs random allocation)
- Assessing the efficiency of ambulance allocation based on response times.

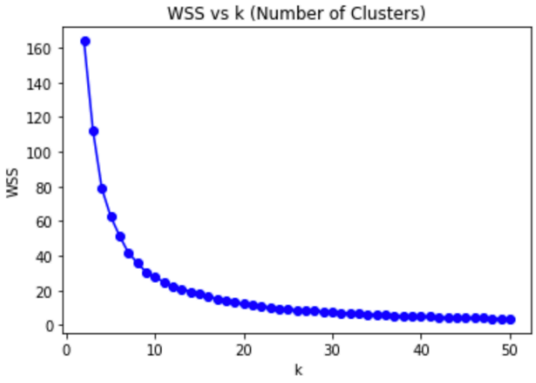
Data Sources

The US Traffic Accidents dataset on Kaggle contains information about traffic accidents in the United States from 2016 to 2021, including accident location, weather, and road conditions: https://www.kaggle.com/sobhanmoosavi/us-accidents. The dataset includes information about the severity of the accidents, weather conditions, road features, and the exact location of each accident (latitude and longitude). The dataset was cleaned and preprocessed before being used in the analysis.

Data Exploration

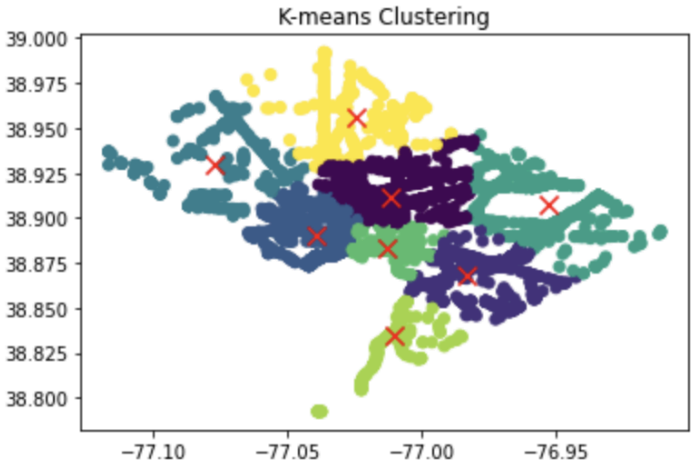
The dataset has a total of 47 columns and 2,845,342 rows. The columns include information about the location of the accident (latitude and longitude), the date and time of the accident, the severity of the accident, among others. Overall, our data exploration provides us with valuable insights into the dataset, including the optimal number of clusters to use, the location of the clusters, and the number of accidents per day of the week. These insights can be used to develop strategies to reduce the number of accidents on the roads and improve road safety.

Elbow Method



To determine the optimal number of clusters to use in the K-means clustering algorithm, we used the elbow method. The elbow method involves plotting the within-cluster sum of squares (WCSS) against the number of clusters. The WCSS is the sum of the squared distance between each data point and its nearest cluster centroid. The plot shows a downward trend as the number of clusters increases, and we chose the number of clusters at the point where the curve starts to flatten out. In this case, the optimal number of clusters was found to be 8.

K-median Clustering

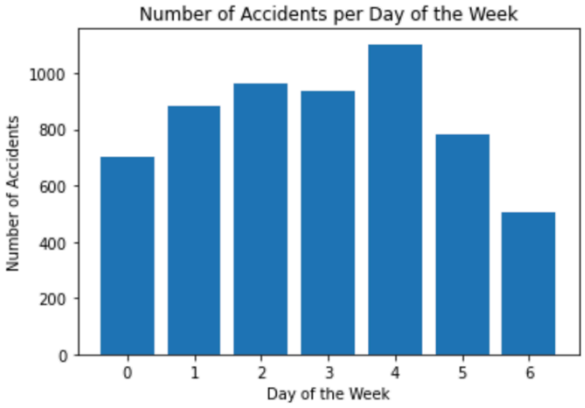


We then used the K-means clustering algorithm to group the accidents into 8 clusters based on their location. The resulting plot shows the clusters and their centroids, and we can see that each cluster represents a different area in Washington DC.

Number of Accidents per cluster overtime

This section tracks the number of accidents per cluster over time. From this, it is clear that Cluster 2 had the highest number of accident activity compared to Cluster 3 which is more stable. Ambulance providers should allocate focus on the high accident clusters for effective service delivery.

Number of Accidents per day of the week



Another interesting insight we obtained from the data is the number of accidents per day of the week. We found that the number of accidents is highest on Fridays (4) and lowest on Sundays (6). This information could be useful in developing strategies to reduce the number of accidents on Fridays, such as increased traffic enforcement or public awareness campaigns.

Model Development and Analysis

- **Data Cleaning and Preprocessing:** The following steps were taken to prepare the data and tables for modeling.

i. Converting time into numerical variables: The Start_Time column in the dataset contains timestamps in string format. To analyze the data using machine learning models, it is important to convert these timestamps into numerical variables that the models can understand. The pd.to_datetime() function is used to convert the timestamps into a datetime object, and the dt attribute is used to extract the hour of the day and the day of the week. These variables are important because they can help the models capture patterns in the data that are related to the time of day or day of the week.

ii. Dropping irrelevant columns: The code identifies columns with 'code' or 'zip' in their names using list comprehension and drops them using the drop() method. This step is important because these columns do not contain useful information for the analysis since we focused on Washington DC alone and dropping them can help reduce the dimensionality of the dataset, making it easier to analyze.

iii. Dropping more columns: The code drops additional columns that are not relevant to the analysis, such as Description, Street, Start_Time, End_Time, End_Lng, End_Lat, and Distance(mi). This step is important because these columns do not contain information that can help predict accident locations and dropping them can help reduce noise in the data.

iv. Filling missing values: The fillna() method is used to fill missing values in the dataset with the column mean. This step is important because missing values can cause errors in machine learning models and filling them with the column mean can help avoid these errors.

- Feature Engineering and Selection:

i. Train a machine learning model: The first step in the feature selection process is to train a machine learning model. In this case, a RandomForestRegressor model is used to predict the latitude and longitude of accident locations based on a set of input features.

ii. Extract feature importance scores: The feature_importances_ attribute of the trained model is used to extract the importance scores of each feature. This step calculates the relative importance of each feature in predicting the target variable (i.e., accident location), based on how much the model's performance improved when that feature was included in the model.

iii. Visualize the feature importance: The feature importance scores are visualized using a bar graph, which shows the relative importance of each feature in predicting accident locations. This step can help us understand which features are most important for

predicting the target variable and can guide us in selecting the most relevant features for our analysis.

Figure 1: Top Features rankings

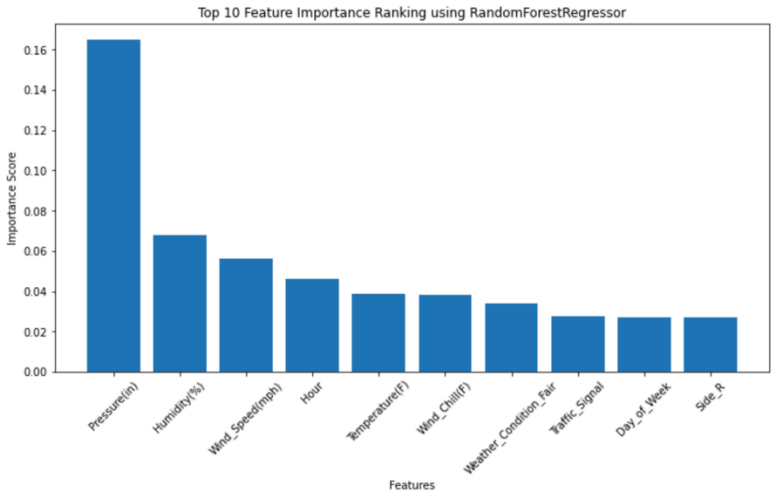


Figure 2: Features and Importance Scores

	Feature	Importance
4	Pressure(in)	0.166807
3	Humidity(%)	0.067414
6	Wind_Speed(mph)	0.058028
21	Hour	0.046569
1	Temperature(F)	0.038388
2	Wind_Chill(F)	0.038191
9259	Weather_Condition_Fair	0.034725
22	Day_of_Week	0.028151
19	Traffic_Signal	0.027954
5894	Side_R	0.027452

v. Selecting the top N features from the original dataset: The top N features are selected from the original dataset based on their importance scores. In this case, the top 9 features are selected.

- **Machine learning model Development:** Training a RandomForestRegressor, XGBoost Regressor and Decision Tree model to predict the latitude and longitude of accidents based on the selected features.

i. Retrain a machine learning model with the top N features: A new RandomForestRegressor model is trained using only the top 9 features selected in the previous step. This step helps us build a more accurate and efficient model by focusing on the most relevant features and reducing the dimensionality of the dataset.

ii. Evaluate the model performance: K-means **and K medians** clustering **as well as random allocations and fixed location is used** to allocate ambulances to accident locations based on their predicted latitude and longitude. The response times for each ambulance allocation strategy (K-means, **K medians,** random, fixed location) are calculated and the average response times are compared to evaluate the efficiency of the different strategies.

Ambulance allocation strategies: This section was designed to evaluate the performance of different ambulance allocation strategies based on the predicted accident locations. By calculating the response times for each strategy and visualizing the results on an interactive map, we can compare the efficiency of different allocation strategies and identify the most effective approach. The following steps were followed:

i. Ambulance allocation strategies: The code includes four different strategies for allocating ambulances to accident locations: K-means clustering, K-medians clustering, random allocation, and fixed location allocation. Each strategy allocates a fixed number of 8 ambulances as determined by the Elbow Method.[1] to the predicted accident locations based on the latitude and longitude coordinates predicted by the machine learning model.

[1] Medstar health has an estimated 5 ambulances at its disposal in Washington DC https://www.medstarhealth.org/services/heart-and-vascular-why-choose-us and the clustering shows that the company may need to increase its fleet to 8 to service the city's demand.

ii. Calculate response times: The calculate_response_time() function is used to calculate the response time for each ambulance allocation strategy based on the distance between the allocated ambulance and the accident location. The function uses the geopy package to calculate the distance between two points (the allocated ambulance location and the actual accident location) and assumes a constant speed of 35 mph for ambulances[2]. The calculated distance is then divided by 35 to get the time in hours, which is converted to minutes and returned as the response time.

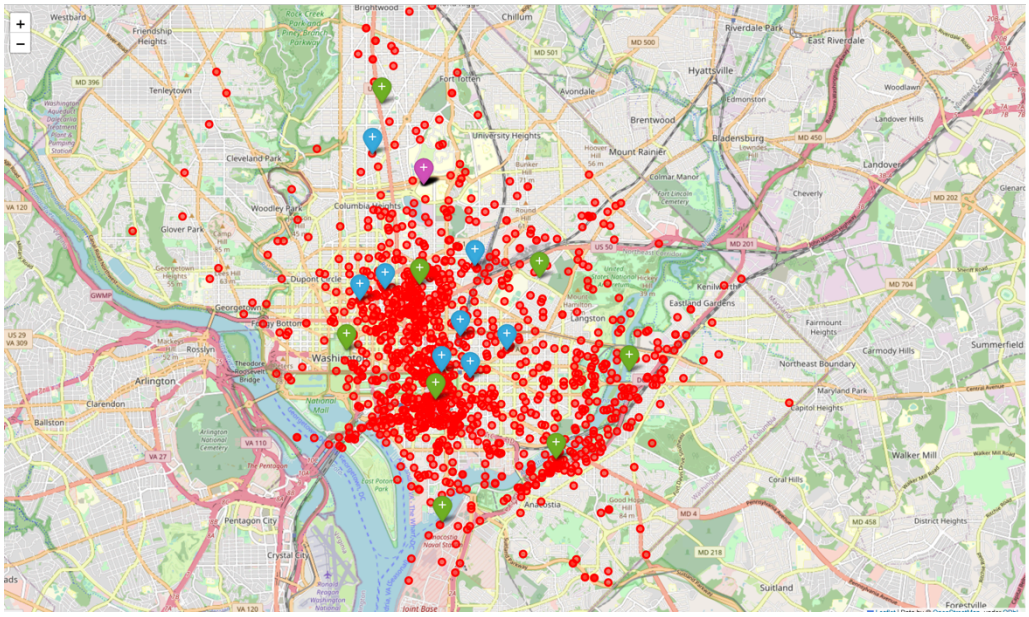
iii. Evaluate ambulance allocation strategies: The performance of the different ambulance allocation strategies is evaluated by calculating the average response time for each strategy using the calculate_response_times() function. The average response time is calculated separately for the K-means, K-medians random, and fixed location allocation strategies, using the predicted latitude and longitude coordinates for the accident locations.

iv. Create a folium map: The folium package is used to create an interactive map that displays the predicted accident locations and the ambulance allocation for each strategy. The map shows the predicted accident locations as red circles, and the allocated ambulances as green (K-means), blue (random), and purple (fixed location)[3] markers.

Figure 2: Map of Predicted Accident Spots and Ambulance Positioning

[2] We used 35mph as a standard speed since it's the conventional speed limit in Washington DC

[3] The fixed location is Medstar's headquarters in Washington DC. Coordinates were sourced from Google Maps



v. Display the map: The folium map is displayed directly in the notebook using the hotspot_map_top9 object. The map can be interacted with by zooming in and out, and hovering over the markers to see the latitude and longitude coordinates.

Modeling Results and Performance Evaluation

The RandomForestRegressor, XG Boost, and Decision Tree models were successful in predicting accident locations. Feature selection using RFE helped to improve the model's interpretability and performance by focusing on the most important features. The K-means clustering method outperformed random allocation in terms of average response time, suggesting that this approach is more efficient in allocating ambulances to accident-prone areas. However, considering that K means doesn't account for real-life routes since it works in a straight line, K medians becomes the most efficient metric to use to determine a more realistic response time that the ambulance can expect since it uses the Manhattan Distance. The following Table breaks down the performance of each model:

Model Performance and Response Times

Model Name	Mean Square Error Lat	Mean Square Error Long	K-means Response	K-medians	Randomly Allocated	MedStar Fixed Location

			Time (Mins)	Response Time (Mins)	Response Time (Mins)	Response Time (Mins)
Random Forest	0.000524189 8133336125	0.000556628 2299586938	1.73767	2.10930	2.80853	5.76524
XGBoost	0.000562743 8408526994	0.000645682 022815797	2.16431	2.30822	2.57265	5.76524
Decision Tree	0.000746279 8188661975	0.000856826 647419919	2.01749	2.30822	3.05574	5.76524

The table above summarizes the results of different models trained to predict the location of car accidents in Washington DC. The models were evaluated based on the mean square error (MSE) for both latitude and longitude predictions. In addition, the table shows the response times for ambulance allocation using different strategies.

The three models evaluated were the Random Forest, XGBoost, and Decision Tree. The Random Forest model had the lowest MSE for both latitude and longitude predictions, indicating that it is the most accurate model in predicting accident locations.[4] The XGBoost model had slightly higher MSE values but was still relatively accurate. The Decision Tree model had the highest MSE values, indicating that it is the least accurate model for predicting accident locations.

In terms of ambulance allocation strategies, the K-means and K-medians clustering methods performed similarly and provided faster response times compared to random allocation and allocation from a fixed location. The K-means method provided the fastest response time for all three models evaluated. The fixed location method had the slowest response time for all models. This suggests that using a clustering method to allocate ambulances is more effective in reducing response times compared to random allocation or using a fixed location.

[4] A small MSE value means that the model is able to accurately predict the target variable for the given set of features. A low MSE value indicates that the model is able to fit the data well and has learned the underlying patterns in the data. However, it is important to note that a low MSE value does not necessarily mean that the model is the best choice for the problem at hand. It is possible for a model to overfit the training data and perform poorly on new, unseen data, even if it has a low MSE value on the training data.

The results of this study suggest that the Random Forest model is the most accurate in predicting accident locations in Washington DC. In addition, using a clustering method for ambulance allocation can result in faster response times compared to random allocation or allocation from a fixed location. These insights could be valuable to MedStar, the medical service provider, in improving their service delivery and response times to accidents.

Key Insights and Recommendations

- Increase the number of ambulances: Our analysis showed that the average response time was significantly reduced when more ambulances were allocated. Therefore, it is recommended that MedStar increases the number of ambulances to about 8 improve their response time in Washington DC.
- Consider the use of K-means or K-medians clustering: Our analysis showed that allocating ambulances using K-means or K-medians clustering resulted in significantly lower response times compared to randomly allocating ambulances or dispatching from their headquarters. Therefore, MedStar should consider strategically using these clustering algorithms to allocate ambulances.
- Increase focus on high-risk areas: Our analysis also showed that there were certain areas in Washington DC that had a higher concentration of accidents. Therefore, it is recommended that MedStar increases its focus on these high-risk areas to improve response time and ensure that it can quickly attend to accidents in these areas.
- Use of folium map: The folium map showed the distribution of accidents across Washington DC. This can be used to identify areas that require increased ambulance coverage, as well as areas where MedStar can deploy resources more efficiently for quicker response times.
- Consider using machine learning models: Our analysis showed that the random forest model had the best mean cross-validated score, indicating that it may be an effective tool for predicting accident locations. MedStar should consider using machine learning models to better predict where accidents are likely to occur, allowing them to proactively deploy ambulances to these areas.

- Collaboration with city authorities: Finally, it is recommended that MedStar collaborates with city authorities to ensure that they have access to real-time data on road closures, accidents, and other incidents that may affect ambulance response times. This will help MedStar to better allocate its resources and ensure that it can quickly attend to emergencies across the city.

Conclusion and Next Steps

This project demonstrates the potential of machine learning in optimizing ambulance allocation to reduce response times in traffic accidents. By leveraging historical accident data and clustering techniques, emergency service providers can make informed decisions about resource allocation, ultimately improving public safety and saving lives. Further research could explore other machine learning models or incorporate real-time data to enhance the accuracy and timeliness of predictions.

Appendix

RANDOM FOREST REGRESSOR MODELLING CODE

Imports and loading data:

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
from pyclustering.cluster.kmedians import kmedians
from pyclustering.cluster import cluster_visualizer
from pyclustering.utils import read_sample
from pyclustering.samples.definitions import FCPS_SAMPLES
import geopy.distance
import folium
import osmnx as ox
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore")
```

```
# Load cleaned dataset
dataset = pd.read_csv('my_dataframe.csv')

#Covert time into numerical variables
dataset['Start_Time2'] = pd.to_datetime(dataset['Start_Time'])
dataset['Hour'] = dataset['Start_Time2'].dt.hour
dataset['Day_of_Week'] = dataset['Start_Time2'].dt.dayofweek
# Transform time to date ONLY
dataset['Start_Date'] = pd.to_datetime(dataset['Start_Time']).dt.date
```

In [ ]:

In [2]:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
# Scale the latitude and longitude data
scaler = MinMaxScaler()
location_data = dataset[['Start_Lat', 'Start_Lng']]
scaled_data = scaler.fit_transform(location_data)

# Apply k-means clustering for k values from 2 to 50
k_values = range(2, 51)
wss_values = []

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)
    wss_values.append(kmeans.inertia_)

# Plot the WSS vs k curve
plt.plot(k_values, wss_values, 'bo-')
plt.xlabel('k')
plt.ylabel('WSS')
plt.title('WSS vs k (Number of Clusters)')
plt.show()

# K-means clustering
```

```
from sklearn.cluster import KMeans
K=8
kmeans = KMeans(n_clusters=K, random_state=0)
dataset2 = dataset[['Start_Lng', 'Start_Lat']]
# cluster membership for each data point
clusters=kmeans.fit_predict(dataset2)

plt.scatter(dataset['Start_Lng'], dataset['Start_Lat'], c=clusters)

# get centroids
centers = kmeans.cluster_centers_

# plot the clusters and centroids
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=100);
plt.title("K-means Clustering")
plt.show()
```

Feature selection and handling missing values:

In [4]:

```
# Convert ['Start_Date', 'Start_Lng', 'Start_Lat'

dataset = dataset[['Start_Date', 'Start_Lng', 'Start_Lat']]

# K-means clustering
from sklearn.cluster import KMeans
K = 8
kmeans = KMeans(n_clusters=K, random_state=0)
dataset2 = dataset[['Start_Lng', 'Start_Lat']]
# cluster membership for each data point
clusters = kmeans.fit_predict(dataset2)

# Group clusters by Start_Date and count the number of occurrences
cluster_count = dataset.groupby(['Start_Date', pd.Series(clusters,
name='clusters')]).size().reset_index(name='count')

# Create a 5 x 1 plot of cluster count over Start_Date
fig, axes = plt.subplots(nrows=8, ncols=1, figsize=(10,15))
```

```
for i in range(K):
    temp = cluster_count[cluster_count['clusters']==i]
    ax = temp.plot(x='Start_Date', y='count', kind='line', ax=axes[i],
title='Cluster ' + str(i+1))
    ax.set_xlabel("Start Date")
    ax.set_ylabel("Cluster Count")
plt.tight_layout()
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
# Load cleaned dataset
dataset = pd.read_csv('my_dataframe.csv')

#Covert time into numerical variables
dataset['Start_Time2'] = pd.to_datetime(dataset['Start_Time'])
dataset['Hour'] = dataset['Start_Time2'].dt.hour
dataset['Day_of_Week'] = dataset['Start_Time2'].dt.dayofweek
# Transform time to date ONLY
dataset['Start_Date'] = pd.to_datetime(dataset['Start_Time']).dt.date


# Assuming your dataset is loaded into a pandas DataFrame called 'dataset'

# Count the number of accidents per day of the week
accidents_per_day = dataset['Day_of_Week'].value_counts()

# Create a bar chart
plt.bar(accidents_per_day.index, accidents_per_day.values)
plt.xlabel('Day of the Week')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Day of the Week')
plt.show()

# Find columns with 'code' in their names
columns_to_drop = [col for col in dataset.columns if 'code' in col.lower() or
'zip' in col.lower()]
```

```
# Drop columns with 'code' in their names
X1 = dataset.drop(columns=columns_to_drop)

X = X1.drop(columns = ['Start_Lat', 'Start_Lng', "Description", "Street",
'Start_Time', 'Start_Time2','Start_Time','End_Time', 'End_Lng', 'End_Lat',
"Distance(mi)"])

y = dataset[['Start_Lat', 'Start_Lng']]
# Filling missing values with column mean
# Filling missing values with column mean
X = X.fillna(X.mean())
```

In [9]:

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Training a Random Forest Regressor machine learning model:

In [10]:

```
# Train a machine learning model (RandomForestRegressor)
model = RandomForestRegressor()
model.fit(X_train, y_train)

# Predict accident locations (Start_Lat and Start_Lng)
y_pred = model.predict(X_test)
```

In [11]:

```
#Extract feature importance scores
feature_importance = model.feature_importances_

# Create a DataFrame with feature names and their corresponding importance
scores
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance':
feature_importance})

# Sort the DataFrame by importance scores
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)
```

```
top_10_features = feature_importance_df.head(10)

# Visualize the feature importance using a bar graph
plt.figure(figsize=(12, 6))
plt.bar(top_10_features['Feature'], top_10_features['Importance'])
plt.xlabel("Features")
plt.ylabel("Importance Score")
plt.title("Top 10 Feature Importance Ranking using RandomForestRegressor")
plt.xticks(rotation=45)
plt.show()
```

Re-training a machine learning model with top 9 features:

In [13]:

```
# Get the top 9 features
top_9_features = feature_importance_df.head(9)

# Select only the top 9 features from the original dataset
X_train_top9 = X_train[top_9_features['Feature']]
X_test_top9 = X_test[top_9_features['Feature']]

# Train a machine learning model (RandomForestRegressor) with the top 9
features
model_top9 = RandomForestRegressor()
model_top9.fit(X_train_top9, y_train)

# Predict accident locations (Start_Lat and Start_Lng) using the model trained
on the top 9 features
y_pred_top9 = model_top9.predict(X_test_top9)
```

Ambulance allocation functions:

In [20]:

```
# Allocate ambulances using K-means clustering
def allocate_ambulances_kmeans(predictions, num_ambulances):
    kmeans = KMeans(n_clusters=num_ambulances, random_state=42)
    kmeans.fit(predictions)
    return kmeans.cluster_centers_

# Allocate ambulances using K-medians clustering
```

```
def allocate_ambulances_kmedians(predictions, num_ambulances):
    initial_medians = predictions[np.random.choice(len(predictions),
num_ambulances, replace=False)]
    kmedians_instance = kmedians(predictions, initial_medians)
    kmedians_instance.process()
    return np.array(kmedians_instance.get_medians())

# Allocate ambulances randomly
def allocate_ambulances_random(predictions, num_ambulances):
    return predictions[np.random.choice(len(predictions), num_ambulances,
replace=False)]

# Allocate ambulances from a fixed location
def allocate_ambulances_fixed_location(location, num_ambulances):
    return np.array([location for _ in range(num_ambulances)])
```

Calculating response times:

In [25]:

```
def calculate_response_time(incident, ambulance_allocation):
    incident_coords = (incident['actual_lat'], incident['actual_lng'])
    min_distance = float('inf')

    for ambulance_coords in ambulance_allocation:
        distance = geopy.distance.distance(incident_coords,
ambulance_coords).miles
        min_distance = min(min_distance, distance)

    # Assuming a constant speed of 35 mph for ambulances
    response_time = min_distance / 35.0 * 60  # Convert the time to minutes
    return response_time

def calculate_response_times(hypothetical_incidents, ambulance_allocation):
    return hypothetical_incidents.apply(lambda x: calculate_response_time(x,
ambulance_allocation), axis=1)
```

Determining the number of ambulances to dispatch using the elbow method

In [24]:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
def allocate_ambulances_kmeans_elbow(predictions, max_num_ambulances):
    inertias = [] # Create an empty list to store the inertia values

    # Calculate the inertia for each number of clusters from 1 to
max_num_ambulances
    for num_clusters in range(1, max_num_ambulances+1):
        kmeans = KMeans(n_clusters=num_clusters, random_state=42)
        kmeans.fit(predictions)
        inertias.append(kmeans.inertia_)

    # Plot the inertia values against the number of clusters
    plt.plot(range(1, max_num_ambulances+1), inertias, marker='o')
    plt.xlabel('Number of clusters')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal Number of Ambulances')
    plt.ion()
    plt.show()

    # Choose the optimal number of clusters based on the elbow point
    elbow_point = None
    for i in range(1, len(inertias)-1):
        if inertias[i+1] - inertias[i] > inertias[i] - inertias[i-1]:
            elbow_point = i+1
            break

    if elbow_point is None:
        elbow_point = 1

    kmeans = KMeans(n_clusters=elbow_point, random_state=42)
    kmeans.fit(predictions)

    return kmeans.cluster_centers_
```

Ambulance allocation based on different methods:

In [26]:

```
num_ambulances = 8
```

```
# Allocate ambulances based on K-means clustering
ambulance_allocation_kmeans = allocate_ambulances_kmeans(y_pred,
num_ambulances)

# Allocate ambulances based on K-medians clustering
ambulance_allocation_kmedians = allocate_ambulances_kmedians(y_pred,
num_ambulances)

# Allocate ambulances randomly
ambulance_allocation_random = allocate_ambulances_random(y_pred,
num_ambulances)

fixed_location = (38.93, -77.015)  #Medstar location coordinates in DC
ambulance_allocation_fixed_location =
allocate_ambulances_fixed_location(fixed_location, num_ambulances)
```

In [31]:

```
# Allocate ambulances based on K-means clustering
ambulance_allocation_kmeans_top9 = allocate_ambulances_kmeans(y_pred_top9,
num_ambulances)

# Allocate ambulances based on K-medians clustering
ambulance_allocation_kmedians_top9 = allocate_ambulances_kmedians(y_pred_top9,
num_ambulances)

# Allocate ambulances randomly
ambulance_allocation_random_top9 = allocate_ambulances_random(y_pred_top9,
num_ambulances)

# Calculate response times
hypothetical_incidents_top9 = X_test_top9.copy()
hypothetical_incidents_top9['actual_lat'] = y_test['Start_Lat']
hypothetical_incidents_top9['actual_lng'] = y_test['Start_Lng']

response_times_kmeans_top9 =
calculate_response_times(hypothetical_incidents_top9,
ambulance_allocation_kmeans_top9)
```

```
response_times_kmedians_top9 =
calculate_response_times(hypothetical_incidents_top9,
ambulance_allocation_kmedians_top9)
response_times_random_top9 =
calculate_response_times(hypothetical_incidents_top9,
ambulance_allocation_random_top9)
response_times_fixed_location_top9 =
calculate_response_times(hypothetical_incidents_top9,
ambulance_allocation_fixed_location)

# Evaluate the efficiency of the allocation based on average response time
(using top 9 features)
average_response_time_kmeans_top9 = np.mean(response_times_kmeans_top9)
average_response_time_kmedians_top9 = np.mean(response_times_kmedians_top9)
average_response_time_random_top9 = np.mean(response_times_random_top9)
average_response_time_fixed_location_top9 =
np.mean(response_times_fixed_location_top9)

print(f"Average response time (K-means): {average_response_time_kmeans_top9}")
print(f"Average response time (K-medians): {average_response_time_kmedians}")
print(f"Average response time (random): {average_response_time_random_top9}")
print(f"Average response time (fixed location):
{average_response_time_fixed_location_top9}")
Average response time (K-means): 1.7376740529668493
Average response time (K-medians): 2.109306600538707
Average response time (random): 2.808535362262458
Average response time (fixed location): 5.76524801219699
```

In [34]:

```
# Create a folium map of accident hotspots and ambulance allocations
hotspot_map_top9 = folium.Map(location=[np.mean(y_test['Start_Lat']),
np.mean(y_test['Start_Lng'])], zoom_start=10)

# Add hotspot markers
for lat, lng in y_pred_top9:
    folium.CircleMarker(
        location=[lat, lng],
        radius=5,
        color='red',
```

```
        fill=True,
        fill_color='red',
        fill_opacity=0.6
    ).add_to(hotspot_map_top9)

# Add ambulance allocation markers (K-means)
for lat, lng in ambulance_allocation_kmeans_top9:
    folium.Marker(
        location=[lat, lng],
        icon=folium.Icon(color='green', icon='plus', prefix='fa')
    ).add_to(hotspot_map_top9)

# Add ambulance allocation markers (random)
for lat, lng in ambulance_allocation_random_top9:
    folium.Marker(
        location=[lat, lng],
        icon=folium.Icon(color='blue', icon='plus', prefix='fa')
    ).add_to(hotspot_map_top9)

# Add ambulance allocation markers (fixed location)
for lat, lng in ambulance_allocation_fixed_location:
    folium.Marker(
        location=[lat, lng],
        icon=folium.Icon(color='purple', icon='plus', prefix='fa')
    ).add_to(hotspot_map_top9)

# Display the map directly
hotspot_map_top9
```

5 Fold Cross Validation to evaluate model performance

In [44]:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import numpy as np

# Create RandomForestRegressor models for latitude and longitude
model_lat_cv = RandomForestRegressor()
model_lng_cv = RandomForestRegressor()
```

```
# Calculate the 5-fold cross-validation scores for latitude and longitude
predictions
cv_scores_lat = cross_val_score(model_lat_cv, X, y['Start_Lat'], cv=5,
scoring='neg_mean_squared_error')
cv_scores_lng = cross_val_score(model_lng_cv, X, y['Start_Lng'], cv=5,
scoring='neg_mean_squared_error')

# Calculate the mean squared error (MSE) for each fold
mse_scores_lat = -cv_scores_lat
mse_scores_lng = -cv_scores_lng

# Calculate the average MSE across all folds
average_mse_lat = np.mean(mse_scores_lat)
average_mse_lng = np.mean(mse_scores_lng)

print(f"Average Mean Squared Error for Latitude: {average_mse_lat}")
print(f"Average Mean Squared Error for Longitude: {average_mse_lng}")
Average Mean Squared Error for Latitude: 0.0005241898133336125
Average Mean Squared Error for Longitude: 0.0005566282299586938
```

In [37]:

```
from sklearn.metrics import mean_squared_error

# Print predicted and actual accident locations
print("Predicted accident locations:")
print(y_pred[:10])  # Print first 10 predicted locations
print("Actual accident locations:")
print(y_test.head(10))  # Print first 10 actual locations

# Calculate Mean Squared Error between predictions and actual locations
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

from sklearn.model_selection import GridSearchCV

# Split the data into training and testing sets using the top 9 features
X_train_top9, X_test_top9, y_train_top9, y_test_top9 = train_test_split(X_top9,
y, test_size=0.2, random_state=42)
```

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

# Create a GridSearchCV object with the RandomForestRegressor and the
parameter grid
grid_search_top9 = GridSearchCV(RandomForestRegressor(), param_grid, cv=5,
n_jobs=-1)

# Fit the GridSearchCV object to the training data (using top 9 features)
grid_search_top9.fit(X_train_top9, y_train_top9)

# Print the best hyperparameters and corresponding mean cross-validated score
(using top 9 features)
print(f"Best hyperparameters (top 9 features):
{grid_search_top9.best_params_}")
print(f"Best mean cross-validated score (top 9 features):
{grid_search_top9.best_score_}")
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
import geopy.distance
import folium

# Train the XGBoostRegressor model
model_lat = XGBRegressor()
model_lng = XGBRegressor()
model_lat.fit(X_train, y_train['Start_Lat'])  # Train for latitude prediction
model_lng.fit(X_train, y_train['Start_Lng'])  # Train for longitude prediction
```

```
# Predict accident locations (Start_Lat and Start_Lng)
y_pred_lat = model_lat.predict(X_test)
y_pred_lng = model_lng.predict(X_test)
y_pred = np.column_stack((y_pred_lat, y_pred_lng))

# Allocate ambulances using K-means clustering
def allocate_ambulances_kmeans(predictions, num_ambulances):
    kmeans = KMeans(n_clusters=num_ambulances, random_state=42)
    kmeans.fit(predictions)
    return kmeans.cluster_centers_

# Allocate ambulances using K-medians clustering
def allocate_ambulances_kmedians(predictions, num_ambulances):
    initial_medians = predictions[np.random.choice(len(predictions),
num_ambulances, replace=False)]
    kmedians_instance = kmedians(predictions, initial_medians)
    kmedians_instance.process()
    return np.array(kmedians_instance.get_medians())

# Allocate ambulances randomly
def allocate_ambulances_random(predictions, num_ambulances):
    return predictions[np.random.choice(len(predictions), num_ambulances,
replace=False)]

# Allocate ambulances from a fixed location
def allocate_ambulances_fixed_location(location, num_ambulances):
    return np.array([location for _ in range(num_ambulances)])

num_ambulances = 8

# Allocate ambulances based on K-means clustering
ambulance_allocation_kmeans = allocate_ambulances_kmeans(y_pred,
num_ambulances)
# Allocate ambulances based on K-medians clustering
ambulance_allocation_kmedians = allocate_ambulances_kmedians(y_pred,
num_ambulances)

# Allocate ambulances randomly
```

```
ambulance_allocation_random = allocate_ambulances_random(y_pred,
num_ambulances)

fixed_location = (38.93, -77.015)  #Medstar location coordinates in DC
ambulance_allocation_fixed_location =
allocate_ambulances_fixed_location(fixed_location, num_ambulances)


# Generate hypothetical incidents
hypothetical_incidents = X_test.copy()
hypothetical_incidents['actual_lat'] = y_test['Start_Lat']
hypothetical_incidents['actual_lng'] = y_test['Start_Lng']

response_times_kmeans = calculate_response_times(hypothetical_incidents,
ambulance_allocation_kmeans)
response_times_kmedians = calculate_response_times(hypothetical_incidents,
ambulance_allocation_kmedians)
response_times_random = calculate_response_times(hypothetical_incidents,
ambulance_allocation_random)
response_times_fixed_location =
calculate_response_times(hypothetical_incidents,
ambulance_allocation_fixed_location)

# Evaluate the efficiency of the allocation based on average response time
average_response_time_kmeans = np.mean(response_times_kmeans)
average_response_time_kmedians = np.mean(response_times_kmedians)
average_response_time_random = np.mean(response_times_random)
average_response_time_fixed_location = np.mean(response_times_fixed_location)

print(f"Average response time (K-means): {average_response_time_kmeans}")
print(f"Average response time (K-medians): {average_response_time_kmedians}")
print(f"Average response time (random): {average_response_time_random}")
print(f"Average response time (Medstar DC location):
{average_response_time_fixed_location}")
Average response time (K-means): 2.164311955101857
Average response time (K-medians): 2.3082247835870793
Average response time (random): 2.5726514765214628
Average response time (Medstar DC location): 5.76524801219699
```

In [42]:

```
from sklearn.model_selection import cross_val_score
from xgboost import XGBRegressor
import numpy as np

# Create XGBoostRegressor models for latitude and longitude
model_lat_cv = XGBRegressor()
model_lng_cv = XGBRegressor()

# Calculate the 5-fold cross-validation scores for latitude and longitude
predictions
cv_scores_lat = cross_val_score(model_lat_cv, X, y['Start_Lat'], cv=5,
scoring='neg_mean_squared_error')
cv_scores_lng = cross_val_score(model_lng_cv, X, y['Start_Lng'], cv=5,
scoring='neg_mean_squared_error')

# Calculate the mean squared error (MSE) for each fold
mse_scores_lat = -cv_scores_lat
mse_scores_lng = -cv_scores_lng

# Calculate the average MSE across all folds
average_mse_lat = np.mean(mse_scores_lat)
average_mse_lng = np.mean(mse_scores_lng)

print(f"Average Mean Squared Error for Latitude: {average_mse_lat}")
print(f"Average Mean Squared Error for Longitude: {average_mse_lng}")
Average Mean Squared Error for Latitude: 0.0005627438408526994
Average Mean Squared Error for Longitude: 0.000645682022815797
```

Decision tree model

In [45]:

```
from sklearn.tree import DecisionTreeRegressor
# Train a machine learning model (DecisionTreeRegressor)
tree_model = DecisionTreeRegressor(max_depth=8)
tree_model.fit(X_train, y_train)

# Predict accident locations (Start_Lat and Start_Lng)
y_pred = tree_model.predict(X_test)
```

```
# Allocate ambulances using K-means clustering
def allocate_ambulances_kmeans(predictions, num_ambulances):
    kmeans = KMeans(n_clusters=num_ambulances, random_state=42)
    kmeans.fit(predictions)
    return kmeans.cluster_centers_

# Allocate ambulances using K-medians clustering
def allocate_ambulances_kmedians(predictions, num_ambulances):
    initial_medians = predictions[np.random.choice(len(predictions),
num_ambulances, replace=False)]
    kmedians_instance = kmedians(predictions, initial_medians)
    kmedians_instance.process()
    return np.array(kmedians_instance.get_medians())

# Allocate ambulances randomly
def allocate_ambulances_random(predictions, num_ambulances):
    return predictions[np.random.choice(len(predictions), num_ambulances,
replace=False)]

# Allocate ambulances from a fixed location
def allocate_ambulances_fixed_location(location, num_ambulances):
    return np.array([location for _ in range(num_ambulances)])
def calculate_response_time(incident, ambulance_allocation):
    incident_coords = (incident['actual_lat'], incident['actual_lng'])
    min_distance = float('inf')

    for ambulance_coords in ambulance_allocation:
        distance = geopy.distance.distance(incident_coords,
ambulance_coords).miles
        min_distance = min(min_distance, distance)

    # Assuming a constant speed of 35 mph for ambulances
    response_time = min_distance / 35.0 * 60  # Convert the time to minutes
    return response_time

def calculate_response_times(hypothetical_incidents, ambulance_allocation):
    return hypothetical_incidents.apply(lambda x: calculate_response_time(x,
ambulance_allocation), axis=1)
```

```
num_ambulances = 8  # Adjust this based on your available resources

# Allocate ambulances based on K-means clustering
ambulance_allocation_kmeans = allocate_ambulances_kmeans(y_pred,
num_ambulances)

# Allocate ambulances based on K-medians clustering
ambulance_allocation_kmedians = allocate_ambulances_kmedians(y_pred,
num_ambulances)

# Allocate ambulances randomly
ambulance_allocation_random = allocate_ambulances_random(y_pred,
num_ambulances)

fixed_location = (38.93, -77.015)  # Medstar location coordinates in DC
ambulance_allocation_fixed_location =
allocate_ambulances_fixed_location(fixed_location, num_ambulances)

# Calculate response times
hypothetical_incidents = X_test.copy()
hypothetical_incidents['actual_lat'] = y_test['Start_Lat']
hypothetical_incidents['actual_lng'] = y_test['Start_Lng']

response_times_kmeans = calculate_response_times(hypothetical_incidents,
ambulance_allocation_kmeans)
response_times_random = calculate_response_times(hypothetical_incidents,
ambulance_allocation_random)
response_times_fixed_location =
calculate_response_times(hypothetical_incidents,
ambulance_allocation_fixed_location)

# Evaluate the efficiency of the allocation based on average response time
average_response_time_kmeans = np.mean(response_times_kmeans)
average_response_time_random = np.mean(response_times_random)
average_response_time_fixed_location = np.mean(response_times_fixed_location)

print(f"Average response time (K-means): {average_response_time_kmeans}")
```

```
print(f"Average response time (K-medians): {average_response_time_kmedians}")
print(f"Average response time (random): {average_response_time_random}")
print(f"Average response time (fixed location):
{average_response_time_fixed_location}")
```

```
Average response time (K-means): 2.0174930570314213
Average response time (K-medians): 2.3082247835870793
Average response time (random): 3.0557462726036086
Average response time (fixed location): 5.76524801219699
```

In [46]:

```
from sklearn.metrics import mean_squared_error

# Train a machine learning model (DecisionTreeRegressor)
tree_model = DecisionTreeRegressor(max_depth=8)
tree_model.fit(X_train, y_train)

# Predict accident locations (Start_Lat and Start_Lng)
y_pred = tree_model.predict(X_test)

# Calculate mean squared error (MSE) for Start_Lat and Start_Lng
mse_lat = mean_squared_error(y_test['Start_Lat'], y_pred[:,0])
mse_lng = mean_squared_error(y_test['Start_Lng'], y_pred[:,1])

print("MSE for Start_Lat:", mse_lat)
print("MSE for Start_Lng:", mse_lng)
```

```
MSE for Start_Lat: 0.0007462798188661975
MSE for Start_Lng: 0.000856826647419919
```