Step 0: Importing and preprocessing

Importing the data

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import silhouette_score

# Dosyayı oku
file_path = 'combined_simulated_data.csv'
data = pd.read_csv(file_path)
```

Date-Time range and defining example_plate, sure_limiti

In [ ]:
```
# Örnek plaka numarası ve süre limiti
example_plate = '72ELF409'
sure_limiti = 20


# Convert the necessary columns to datetime
data['tespitBaslangicZamani'] = pd.to_datetime(data['tespitBaslangicZamani'])
data['tespitBitisZamani'] = pd.to_datetime(data['tespitBitisZamani'])

# Define the time range for filtering
tarih_baslangic = pd.Timestamp('2024-12-06 18:30:00')
tarih_bitis = pd.Timestamp('2024-12-06 23:06:50')

# Filter the data based on the given time range
filtered_data = data[(data['tespitBaslangicZamani'] >= tarih_baslangic) & (data['tespitBaslangicZamani'] <= tarih_bit

filtered_data
```

Out[ ]:

	sensorNo	tespitTipi	deploymentid	tespitBaslangicZamani	alarmBaslangicZamani	alarmBitisZamani	raporlandimi	tes
0	69.0	50	0	2024-12-06 19:14:48	2024-12-06 19:16:48	2024-12-06 19:20:48	True	
1	69.0	50	0	2024-12-06 19:30:48	2024-12-06 19:32:48	2024-12-06 19:39:48	True	
2	69.0	50	0	2024-12-06 19:46:48	2024-12-06 19:50:48	2024-12-06 19:51:48	False	
3	69.0	50	0	2024-12-06 20:02:48	2024-12-06 20:04:48	2024-12-06 20:12:48	False	
4	69.0	50	0	2024-12-06 20:18:48	2024-12-06 20:21:48	2024-12-06 20:27:48	True	
...	...	...	...	...	...	...	...	
9990189	641.0	50	0	2024-12-06 22:04:16	2024-12-06 22:07:16	2024-12-06 22:12:16	True	
9991624	63.0	50	0	2024-12-06 22:39:52	2024-12-06 22:41:52	2024-12-06 22:46:52	False	
9992346	300.0	50	0	2024-12-06 19:05:30	2024-12-06 19:10:30	2024-12-06 19:11:30	False	
9994937	713.0	50	0	2024-12-06 18:47:26	2024-12-06 18:52:26	2024-12-06 18:57:26	True	
9996946	1031.0	50	0	2024-12-06 19:28:29	2024-12-06 19:31:29	2024-12-06 19:36:29	True	

5506 rows × 22 columns

Step 1: Clustering

In [ ]:
```
# Elbow yöntemiyle optimum küme sayısını belirleme
def find_optimal_clusters(data,min_k, max_k,kn):
    iters = range(min_k, max_k,kn)
    sse = []
    for k in iters:
        sse.append(MiniBatchKMeans(n_clusters=k, batch_size=100000).fit(data).inertia_)



    return iters[sse.index(min(sse))]

optimal_clusters=find_optimal_clusters(filtered_data[['tespitEnlem', 'tespitBoylam']],45 , 70,2)


# MiniBatchKMeans kümeleme
```

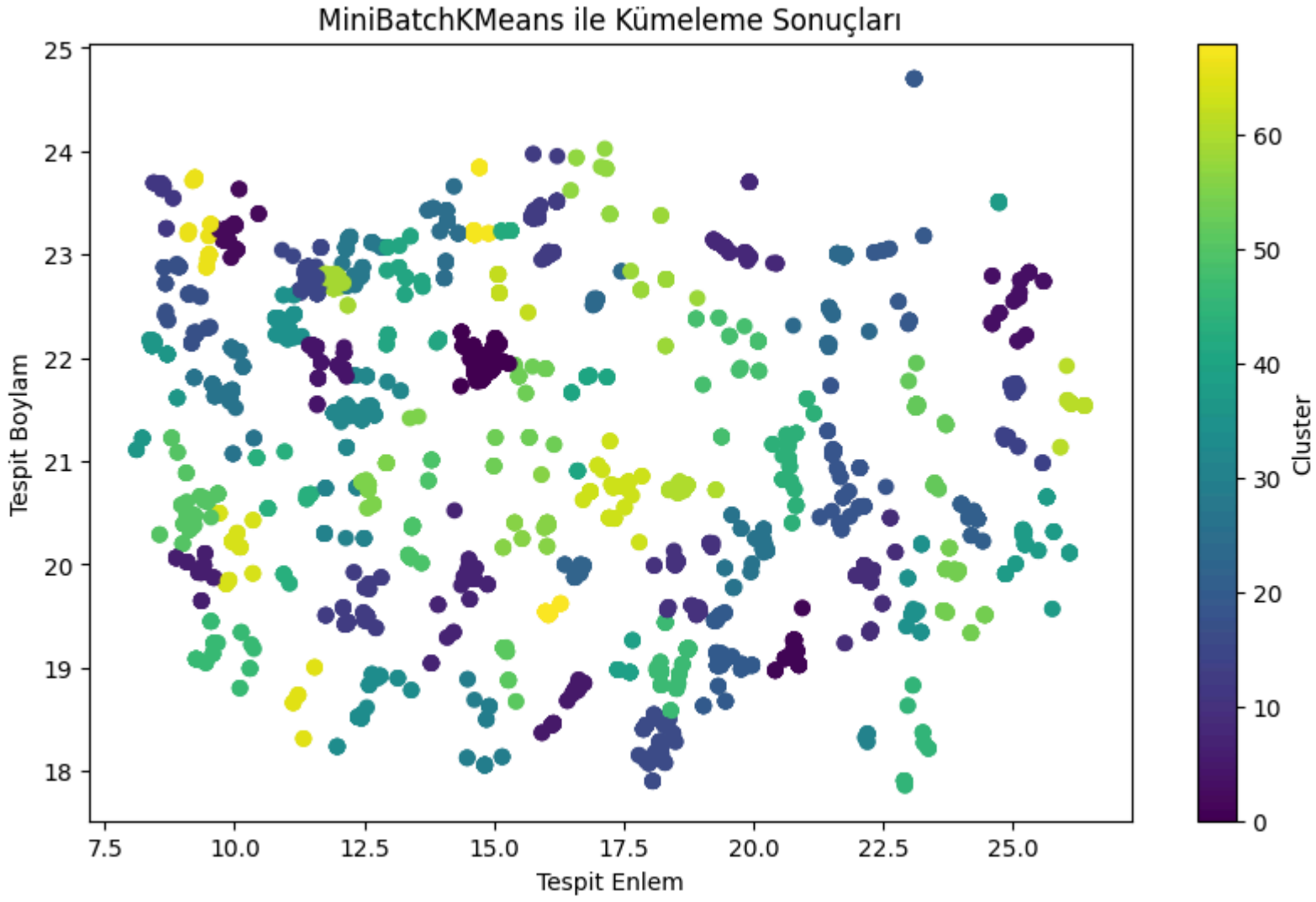
```
mini_batch_kmeans = MiniBatchKMeans(n_clusters=optimal_clusters, batch_size=100000)
filtered_data['cluster'] = mini_batch_kmeans.fit_predict(filtered_data[['tespitEnlem', 'tespitBoylam']])



# Kümeleme sonuçlarını görselleştirme
plt.figure(figsize=(10, 6))
plt.scatter(filtered_data['tespitEnlem'], filtered_data['tespitBoylam'], c=filtered_data['cluster'], cmap='viridis')
plt.xlabel('Tespit Enlem')
plt.ylabel('Tespit Boylam')
plt.title('MiniBatchKMeans ile Kümeleme Sonuçları')
plt.colorbar(label='Cluster')
plt.show()
```

```
C:\Users\xxtra\AppData\Local\Temp\ipykernel_19756\493759139.py:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  filtered_data['cluster'] = mini_batch_kmeans.fit_predict(filtered_data[['tespitEnlem', 'tespitBoylam']])
```



Step 2 Trigger: Looks for cars that are suspicious

In [ ]:

```
from datetime import timedelta


def konvoy_analizi(df_filtered, plaka, sure_limiti):

    # 3. plaka_data = df_filtered içinde belirtilen plaka'nın verileri.
    plaka_data = df_filtered[df_filtered['plaka'] == plaka]

    # 4. konvoy_plakalari boş setini oluştur.
    konvoy_plakalari = set()

    # 5. aday_plakalar = df_filtered'daki tüm farklı plakalar - belirtilen plaka.
    aday_plakalar = set(df_filtered['plaka'].unique()) - {plaka}


    sensor_dizisi = plaka_data[['sensorNo', 'unitNo']].drop_duplicates().to_dict('records')

    # HER sensor için sensor_dizisi içinde
    for sensor in sensor_dizisi:
        sensor_no, unit_no = sensor['sensorNo'], sensor['unitNo']

        # sensor_data = plaka_data'daki bu sensördeki geçiş zamanları
        sensor_data = plaka_data[(plaka_data['sensorNo'] == sensor_no) & (plaka_data['unitNo'] == unit_no)]

        # HER tespit_zamani için sensor_data içinde
        for tespit_zamani in sensor_data['tespitBaslangicZamani']:
            baslangic_zamani = tespit_zamani - timedelta(minutes=sure_limiti)
            bitis_zamani = tespit_zamani + timedelta(minutes=sure_limiti)

            # FILTRELE aday_plakalar, bu zaman aralığında ve sensörde geçenler
            aday_gecenler = df_filtered[(df_filtered['tespitBaslangicZamani'] >= baslangic_zamani) &
                                        (df_filtered['tespitBaslangicZamani'] <= bitis_zamani) &
```

```
                                        (df_filtered['sensorNo'] == sensor_no) &
                                        (df_filtered['unitNo'] == unit_no) &
                                        (df_filtered['plaka'].isin(aday_plakalar))]['plaka']

            # konvoy_plakalari'na aday_gecenler'i ekle
            konvoy_plakalari.update(aday_gecenler)

    # DÖNDÜR konvoy_plakalari listesi
    return list(konvoy_plakalari)


df=filtered_data


sure_limiti = 10

konvoy_plakalari_ = konvoy_analizi(df, example_plate, sure_limiti)

print("Konvoy analizi sonuçları :", konvoy_plakalari_)
```

```
Konvoy analizi sonuçları : ['46ER9613', '61FZ1281', '52D9123', '73LC1156']
```

Step 3 Mathmatical Modeling: Vector repersentation of the path of the cars

In [ ]:
```
example_plate_location=(example_plate,filtered_data[filtered_data['plaka'] == example_plate].sort_values('tespitBasla
example_plate_location
```

Out[ ]:
```
('72ELF409', 12.131768999999998, 19.421847, 19.379293, 19.063026)
```

In [ ]:
```
first_last_measurement=[]
for a in konvoy_plakalari_:
    first_last_measurement.append((a,filtered_data[filtered_data['plaka'] == a].sort_values('tespitBaslangicZamani').
first_last_measurement
```

Out[ ]:
```
[('46ER9613', 12.131768999999998, 19.421847, 19.879578, 22.966309000000003),
 ('61FZ1281', 12.131768999999998, 19.421847, 19.879578, 22.966309000000003),
 ('52D9123', 12.131768999999998, 19.421847, 19.879578, 22.966309000000003),
 ('73LC1156', 12.131768999999998, 19.421847, 19.879578, 22.966309000000003)]
```

In [ ]:
```
import math
from math import radians, cos, sin, sqrt, atan2
def analyze_vectors(x1, y1, x2, y2, x11, y11, x22, y22):
    # Vektörleri oluştur
    vector1 = np.array([x2 - x1, y2 - y1])
    vector2 = np.array([x22 - x11, y22 - y11])

    # Vektörlerin büyüklüğünü hesapla
    vector1_magnitude = np.linalg.norm(vector1)
    vector2_magnitude = np.linalg.norm(vector2)

    # Vektörlerin büyüklüğünün sıfır olup olmadığını kontrol et
    if vector1_magnitude == 0 or vector2_magnitude == 0:
        return "One of the cars has zero dislocation in given time", "Cannot determine intersection status"

    # Vektörlerin dot ürününü hesapla
    dot_product = np.dot(vector1, vector2)

    # Açı hesapla (radyan cinsinden)
    cos_theta = dot_product / (vector1_magnitude * vector2_magnitude)

    # cos_theta'nın aralığını kontrol et
    if cos_theta < -1 or cos_theta > 1:
        cos_theta = max(-1, min(1, cos_theta))

    # Açı derece cinsinden
    theta_radians = np.arccos(cos_theta)
    theta_degrees = np.degrees(theta_radians)

    # Vektörlerin yönlerini kontrol et
    direction1 = vector1 / vector1_magnitude
    direction2 = vector2 / vector2_magnitude

    # İki vektörün yön vektörlerinin cross ürünü
    cross_product = np.cross(direction1, direction2)

    # İki vektörün zamanla birbirlerine yaklaşma veya uzaklaşma durumunu kontrol et
    relative_velocity = vector2 - vector1
    relative_position = np.array([x11 - x1, y11 - y1])
    approaching = np.dot(relative_velocity, relative_position) < 0

    if np.isclose(cross_product, 0):
        intersection_status = "Cars are parallel or anti-parallel"
    else:
        if approaching:
            intersection_status = "Cars are approaching each other"
        else:
            intersection_status = "Cars are moving away from each other"

    return theta_degrees, intersection_status
```

```
def haversine(lat1, lon1, lat2, lon2):
    # Dünya'nın yarıçapı (kilometre cinsinden)
    R = 6371.0

    # Koordinatları radyan cinsine çevir
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    # Farkları hesapla
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    # Haversine formülü
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c

    return distance


path_angles=[]
for a in first_last_measurement:
    c,d=analyze_vectors(example_plate_location[1],example_plate_location[2],example_plate_location[3],example_plate_l
    baslangicuzaklik=haversine(example_plate_location[1],example_plate_location[2],a[1],a[2])
    bitisuzakik=haversine(example_plate_location[3],example_plate_location[4],a[3],a[4])
    path_angles.append([a[0],c,d,baslangicuzaklik,bitisuzakik])
```

Step 4 Feature Extraction

In [ ]:

```
def ortak_cluster_ve_sensor_sayisi(example_plate, plakalar, data):
    example_clusters = data[data['plaka'] == example_plate]['cluster'].unique()
    example_sensors = data[data['plaka'] == example_plate]['sensorNo'].unique()
    ortak_sayilar = []

    for plaka in plakalar:
        if plaka == example_plate:
            continue
        plaka_clusters = data[data['plaka'] == plaka]['cluster'].unique()
        plaka_sensors = data[data['plaka'] == plaka]['sensorNo'].unique()

        ortak_cluster = np.intersect1d(example_clusters, plaka_clusters)
        ortak_sensor = np.intersect1d(example_sensors, plaka_sensors)

        ortak_sayilar.append([plaka, len(ortak_cluster), len(ortak_sensor)])

    return ortak_sayilar


sayımlar = ortak_cluster_ve_sensor_sayisi(example_plate, konvoy_plakalari_, filtered_data)
for a in range(0,len(path_angles)):
    path_angles[a].append(sayımlar[a][1])
    path_angles[a].append(sayımlar[a][2])
```

In [ ]:

```
from geopy.distance import geodesic

def birliktealınanyollar(example_plate, plakalar_listesi, konvoy_df):
    # example_plate ile listedeki her bir plakanın ortak geçtikleri sensörlerden ilki ile sonu arasındaki uzaklığı he
    uzakliklar = []

    for plaka in plakalar_listesi:
        # example_plate'in ve plaka'nın geçtiği ortak sensörler
        example_plate_sensors = konvoy_df[konvoy_df['plaka'] == example_plate][['sensorNo', 'unitNo']].drop_duplicate
        plaka_sensors = konvoy_df[konvoy_df['plaka'] == plaka][['sensorNo', 'unitNo']].drop_duplicates()

        ortak_sensors = pd.merge(example_plate_sensors, plaka_sensors, on=['sensorNo', 'unitNo'])

        if not ortak_sensors.empty:
            # İlk ve son sensörlerin koordinatlarını al
            ilk_sensor_coords = konvoy_df[(konvoy_df['sensorNo'] == ortak_sensors.iloc[0]['sensorNo']) &
                                          (konvoy_df['unitNo'] == ortak_sensors.iloc[0]['unitNo'])].iloc[0][['tespitE

            son_sensor_coords = konvoy_df[(konvoy_df['sensorNo'] == ortak_sensors.iloc[-1]['sensorNo']) &
                                          (konvoy_df['unitNo'] == ortak_sensors.iloc[-1]['unitNo'])].iloc[0][['tespit

            # Uzaklığı hesapla
            uzaklik = geodesic((ilk_sensor_coords['tespitEnlem'], ilk_sensor_coords['tespitBoylam']),
                               (son_sensor_coords['tespitEnlem'], son_sensor_coords['tespitBoylam'])).kilometers

            uzakliklar.append((plaka, uzaklik))

    return uzakliklar



uzaklıklar = birliktealınanyollar(example_plate, konvoy_plakalari_, filtered_data)
```

```
sadeceuzaklıklar= [a[1] for a in uzaklıklar]
mean_sadeceuzaklıklar=np.mean(sadeceuzaklıklar)
std_sadeceuzaklıklar=np.std(sadeceuzaklıklar)
max_sadeceuzaklıklar=max(sadeceuzaklıklar)

for a in range(0,len(path_angles)):
    path_angles[a].append(uzaklıklar[a][1])
```

In [ ]:
```
for a in path_angles:
    print("Plaka: ",a[0], " Açı: ",a[1], " Durum: ",a[2], " Başlangıç Uzaklık: ",a[3], " Bitiş Uzaklık: ",a[4], " Ort
```

```
Plaka:  46ER9613  Açı:  27.417481250440638  Durum:  Cars are moving away from each other  Başlangıç Uzaklık:  0.0  Bi
tiş Uzaklık:  412.558868094133  Ortak Cluster Sayısı:  4  Ortak Sensor Sayısı:  4  Ortak Alınan Yol Uzaklığı:  937.40
84514152216
Plaka:  61FZ1281  Açı:  27.417481250440638  Durum:  Cars are moving away from each other  Başlangıç Uzaklık:  0.0  Bi
tiş Uzaklık:  412.558868094133  Ortak Cluster Sayısı:  4  Ortak Sensor Sayısı:  4  Ortak Alınan Yol Uzaklığı:  937.40
84514152216
Plaka:  52D9123  Açı:  27.417481250440638  Durum:  Cars are moving away from each other  Başlangıç Uzaklık:  0.0  Bit
iş Uzaklık:  412.558868094133  Ortak Cluster Sayısı:  4  Ortak Sensor Sayısı:  4  Ortak Alınan Yol Uzaklığı:  937.408
4514152216
Plaka:  73LC1156  Açı:  27.417481250440638  Durum:  Cars are moving away from each other  Başlangıç Uzaklık:  0.0  Bi
tiş Uzaklık:  412.558868094133  Ortak Cluster Sayısı:  4  Ortak Sensor Sayısı:  4  Ortak Alınan Yol Uzaklığı:  937.40
84514152216
```

Step 5: Scoring and Ranking

In [ ]:
```
def gaussian(x, mu, sigma):
    return (1/(sigma * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - mu) / sigma) ** 2)

scores=[]
for a in path_angles:
    score=0

    if a[2]=="Cars are approaching each other":
        score = 1/a[1]
    elif a[2] == "Cars are moving away from each other":
        score = 1/(-a[1])
    elif a[2] == "Cars are parallel or anti-parallel":
        score = 0

    score+=a[5]*7
    score+=a[6]*10
    score+=10/(a[3]+1)
    score+=10/(a[4]+1)
    score+=a[7]*5/max_sadeceuzaklıklar
    scores.append([a[0],score])

scores.sort(key=lambda x: x[1], reverse=True)
scores
```

Out[ ]:
```
[['46ER9613', 82.98770727393773],
 ['61FZ1281', 82.98770727393773],
 ['52D9123', 82.98770727393773],
 ['73LC1156', 82.98770727393773]]
```

The numbers used in scoring are arbitrary and need to be optimized via a learning algorithm with labeled data to make the scoring more accurate. The scoring and ranking step is just a proof of concept.