VANETs using BIRCH

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

datos = pd.read_csv('/content/VehicleData.csv')
pd.set_option('display.max_columns', 12)
datos.head(10)
#displaying the basic format of the dataset used in the first 10 lines
```

	latitude	longitude	target_speed	way_maxspeed	speed_osrm	elevation	...	elevation_diff	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11
0	50.518304	14.971893	1.776536	90	4.375000	242.0	...	0.0	NaN	NaN	NaN	NaN
1	50.518305	14.971879	1.776536	90	4.375000	242.0	...	0.0	NaN	NaN	NaN	NaN
2	50.518305	14.971865	1.776536	90	4.375000	242.0	...	0.0	NaN	NaN	NaN	NaN
3	50.518306	14.971851	1.776536	90	4.375000	242.0	...	0.0	NaN	NaN	NaN	NaN
4	50.518306	14.971837	2.056782	90	4.339816	242.0	...	0.0	NaN	NaN	NaN	NaN
5	50.518307	14.971823	2.683833	90	4.261092	242.0	...	0.0	NaN	NaN	NaN	NaN
6	50.518308	14.971809	3.310884	90	4.182368	242.0	...	0.0	NaN	NaN	NaN	NaN
7	50.518308	14.971795	3.937934	90	4.103644	242.0	...	0.0	NaN	NaN	NaN	NaN
8	50.518309	14.971780	4.564985	90	4.024920	242.0	...	0.0	NaN	NaN	NaN	NaN
9	50.518309	14.971766	4.753176	90	4.039720	242.0	...	0.0	NaN	NaN	NaN	NaN

10 rows × 13 columns

```
"""datos = datos.drop(['Unnamed: 6'], axis = 1)
datos = datos.drop(['Unnamed: 7'], axis = 1)
datos = datos.drop(['Unnamed: 8'], axis = 1)
datos = datos.drop(['Unnamed: 9'], axis = 1)
datos = datos.drop(['Unnamed: 10'], axis = 1)
datos = datos.drop(['Unnamed: 11'], axis = 1)"""
datos = datos.drop(['Unnamed: 12'], axis = 1)
datos.info()
#this code will drop the column named 'Unnamed: 12' from the DataFrame datos and then print the information about the DataFrame using the info() method.
```

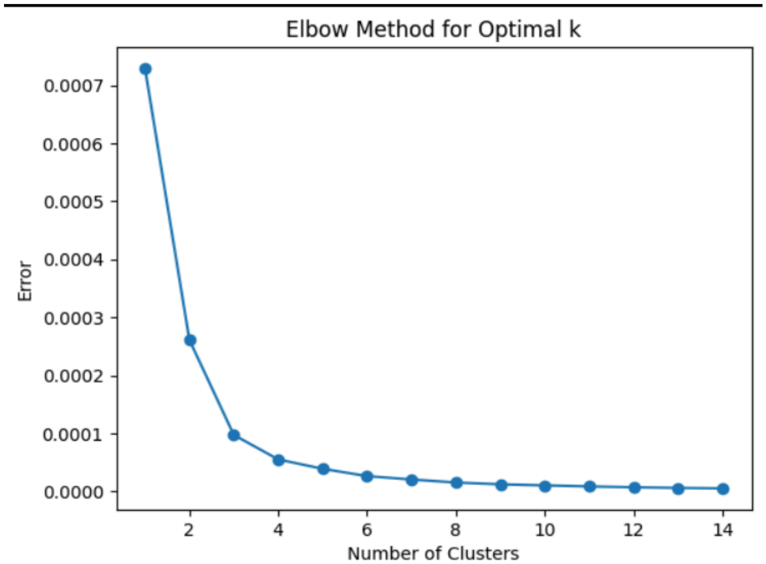
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 534 entries, 0 to 533
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   latitude        534 non-null    float64
 1   longitude       534 non-null    float64
 2   target_speed    534 non-null    float64
 3   way_maxspeed    534 non-null    int64  
 4   speed_osrm      534 non-null    float64
 5   elevation       534 non-null    float64
 6   azimuth_diff    534 non-null    float64
 7   elevation_diff  534 non-null    float64
 8   Unnamed: 8      0 non-null      float64
 9   Unnamed: 9      0 non-null      float64
 10  Unnamed: 10     0 non-null      float64
 11  Unnamed: 11     1 non-null      object 
dtypes: float64(10), int64(1), object(1)
memory usage: 50.2+ KB
```

datos.describe()

	latitude	longitude	target_speed	way_maxspeed	speed_osrm	elevation
count	534.000000	534.000000	534.000000	534.000000	534.000000	534.000000
mean	50.518982	14.972706	9.405788	66.029963	8.923809	239.122303
std	0.000894	0.000755	4.559061	19.620390	2.903118	2.023365
min	50.518083	14.971105	0.481069	50.000000	0.185747	236.000000
25%	50.518203	14.971963	5.608784	50.000000	6.785714	237.000000
50%	50.518526	14.973183	8.464671	50.000000	10.792773	239.000000
75%	50.519723	14.973262	14.612338	90.000000	11.153846	241.000000
max	50.520917	14.973398	15.263918	90.000000	12.571429	242.000000

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

rango_clusters = range(1, 15)
error_clusters = []

# Selecting columns for clustering (assuming 'datos' contains your data)
columns_for_clustering = datos[['latitude', 'longitude']]

# Iterate over different numbers of clusters
for num_clusters in rango_clusters:
    clusters = KMeans(n_clusters=num_clusters)
    clusters.fit(columns_for_clustering)
    error_clusters.append(clusters.inertia_)  # Sum of squared distances to closest cluster center

# Create a dataframe to store number of clusters and corresponding errors
clusters_df = pd.DataFrame({"num_clusters": rango_clusters, "error_clusters": error_clusters})
```

```python
# Plotting the elbow method to determine the optimal number of clusters
plt.plot(clusters_df.num_clusters, clusters_df.error_clusters, marker="o")
plt.xlabel('Number of Clusters')
plt.ylabel('Error')
plt.title('Elbow Method for Optimal k')
plt.show()
```

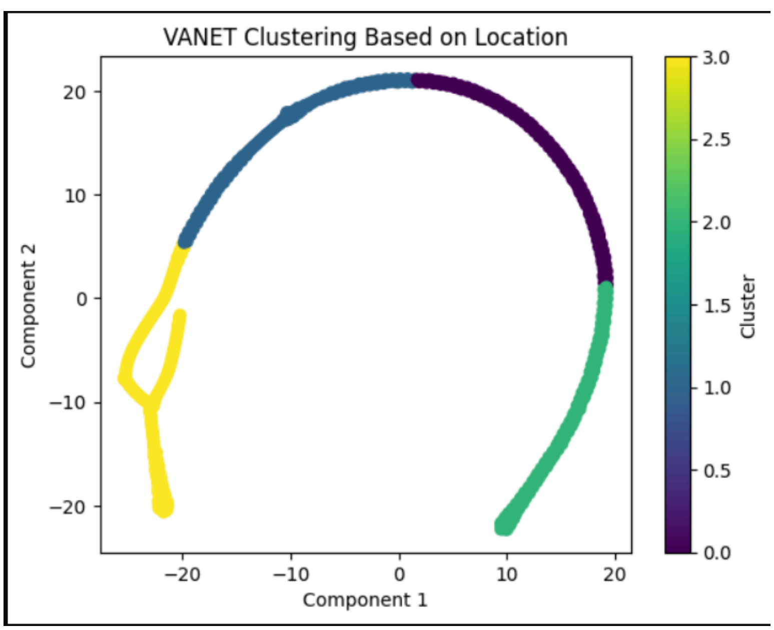



```python
from sklearn.cluster import KMeans
kmeans1 = KMeans(n_clusters=2, init='k-means++', n_init=10, max_iter=300, tol=0.0001,
verbose=0, random_state=None, copy_x=True, algorithm='lloyd')
kY = kmeans1.fit_predict(datos)

# Creating a two-dimensional embedding of the data
from sklearn.manifold import TSNE

# Creating a TSNE object with specified parameters
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=4000, random_state=2)

# Applying t-SNE to your scaled data to generate a two-dimensional embedding
Y = tsne.fit_transform(datos)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 534 samples in 0.001s...
[t-SNE] Computed neighbors for 534 samples in 0.017s...
[t-SNE] Computed conditional probabilities for sample 534 / 534
[t-SNE] Mean sigma: 0.647550
[t-SNE] KL divergence after 250 iterations with early exaggeration: 42.796524
[t-SNE] KL divergence after 1700 iterations: 0.110132
```

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

```
from sklearn.manifold import TSNE

# 'datos' contains VANET data
# Selecting relevant features for clustering (latitude and longitude of vehicle positions)
vanet_features = datos[['latitude', 'longitude']]

# Creating a TSNE object with 2 components (for a 2D projection)
tsne = TSNE(n_components=2, random_state=0)

# Applying t-SNE to your VANET data
vanet_data_projected = tsne.fit_transform(vanet_features)

# Creating a KMeans object with the desired number of clusters
kmeans = KMeans(n_clusters=4, random_state=0)

# Fitting the KMeans model to the VANET data
cluster_labels = kmeans.fit_predict(vanet_data_projected)

# Plotting the clusters
plt.scatter(vanet_data_projected[:, 0], vanet_data_projected[:, 1], c=cluster_labels,
cmap='viridis')
plt.title('VANET Clustering Based on Location')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.colorbar(label='Cluster')
plt.show()
```



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

# Assuming 'vanet_data' contains your VANET data
# Modify the code accordingly based on your VANET data structure and features
# Selecting relevant features for clustering
vanet_features = datos[['latitude', 'longitude']]

# Creating a TSNE object with 2 components (for a 2D projection)
tsne = TSNE(n_components=2, random_state=0)

# Applying t-SNE to your VANET data
vanet_data_projected = tsne.fit_transform(vanet_features)

# Creating a KMeans object with the desired number of clusters
kmeans = KMeans(n_clusters=4, random_state=0)

# Fitting the KMeans model to the VANET data
cluster_labels = kmeans.fit_predict(vanet_data_projected)

# Creating subplots
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(12, 8))

# Plotting modeled clusters by K-means
ax1.scatter(vanet_data_projected[:, 0], vanet_data_projected[:, 1], c=cluster_labels, cmap='jet',
edgecolor='None', alpha=0.35)
ax1.set_title('Clusters by K-means')

# Plotting actual clusters (if available)
# Replace 'datos['diagnosis']' with the actual cluster labels in your VANET data
# ax2.scatter(vanet_data_projected[:, 0], vanet_data_projected[:, 1],
c=vanet_data['actual_cluster_labels'], cmap='jet', edgecolor='None', alpha=0.35)
# ax2.set_title('Actual Clusters')

# Setting common labels
for ax in (ax1, ax2):
   ax.set_xlabel('Component 1')
   ax.set_ylabel('Component 2')

# Displaying the plot
plt.show()
```