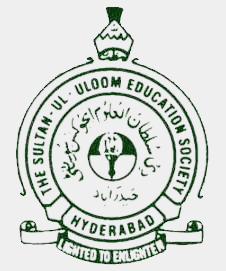
**MUFFAKHAM JAH COLLEGE OF ENGINEERING TECHNOLOGY**

(Affiliated to Osmania University)

Mount Pleasant, 8-2-249, Road No. 3, Banjara Hills, Hyderabad-34.



**DEPARTMENT OF INFORMATION TECHNOLOGY**

***CERTIFICATE***

This is to certify that the Mini Project work titled “**Clustering Geolocation Data Intelligently**” is a bonafide work prescribed by the Osmania University for B.E III/IV VIth semester during the academic year 2019-2020 carried out by **Ayush Singh (1604-17-737-022), Pranav Hindupur (1604-17-737-004) and Samarth Shiram Shetty(1604-17-737-022).**

Course Coordinator Head-ITD

Mr. Prasad Dr Mousmi Ajay Chaurasia

A

Mini Project Report

On

**Clustering Geolocation Data Intelligently**

By

**Ayush Singh (1604-17-737-022)**

**Pranav Hindupur (1604-17-737-004)**

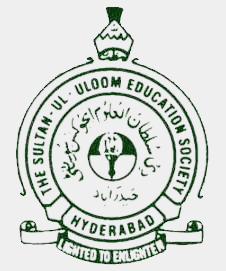
**Samarth Shiram Shetty(1604-17-737-022)**

Of

III/IV B.E. Sem-VI (IT-A)

Under the Guidance of

Mr. Prasad.



DEPARTMENT OF INFORMATION TECHNOLOGY

**MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY** (Affiliated to Osmania University)

Mount Pleasant, 8-2-249, Road No. 3, Banjara Hills, Hyderabad-34.

**ACKNOWLEDGEMENT**

The austerity and satisfaction that one gets on completing a project cannot be fulfilled without mentioning the people who made it possible with gratitude.

We are grateful to the almighty God who helped us all the way throughout the project and also has molded us into what we are today. We express our sincere thanks to our parents who encouraged us always to achieve our goals.

We offer our sincere thanks to Muffakham Jah College of Engineering and Technology for allowing us to do our mini project in their esteemed institution.

We show gratitude to **Dr. K. N. Krishnan sir, M.Tech, PhD** and **Principal** for having providing all the facilities and support. We would like to thank **Mrs. Mousmi Ajay Chaurasia** (**Head of the Department, Information Technology)** for her expert guidance and encouragement at various levels of the project.

We are Thankful to our guide **Prasad** **Sir, Assistant Professor of the Department** for his sustained inspiring Guidance and cooperation throughout the process of this project report.

We express our deep sense of gratitude and thanks to all the **Teaching** and **Non-Teaching Staff** of our college who stood with us and helped us to make it a successful venture.

**ABSTRACT**

PURPOSE OF THE SYSTEM:

The problem statement here is a geo cluster has to be formed for a TAXI SERVICE, where a taxi rank is the place of operation of a taxi and also its storage. Hence this application must find a geo cluster of the appropriate places where taxis can set up for maximum profit through the given data set.

EXISTING SYSTEM:

Geo clustering basically disperses its nodes to several physical locations and is unaware of the physical distance between the nodes. Geo clustering protects data and IT resources and is used in market research and pattern recognition amongst the other applications.

PROPOSED SYSTEM:

The application with the help of the given data set, coded in Python 3, with the help of Pandas library is executed on the Jupyter Notebook, an IDE to help find the most appropriate cluster, for taxi banks with the help of K-Means algorithm, DBSCAN, and HDBSCAN.

SYSTEM CONFIGURATION:

Hardware requirements:

* Processer: Any Updated Processer
* Ram: Min 1 GB
* Hard Disk: Min 100 GB

Software requirements:

* Operating System: Windows family
* Technology: Python 3, and Panda Library.
* Web Technologies: Jupyter Notebook

**CONTENTS**

1. **Introduction**
2. **Literature survey**
3. **Software Requirement Specification**
4. **System Design**
   1. **System Architecture**
5. **Implementation**
6. **Testing**
7. **Screenshots**
8. **Conclusion**
9. **Future Enhancement**

**CHAPTER 1**

**INTRODUCTION**

Geo clustering is [computer clustering](https://susedefines.suse.com/definition/computer-cluster/) over geographically dispersed sites. A basic cluster is a group of independent computers called nodes, usually housed in the same physical location, that work together to run a common set of applications. The nodes are physically connected by network and storage infrastructure and logically connected by clustering software. Unlike a basic cluster, a geo cluster disperses its nodes to several different physical locations.

Geo clustering protects data and IT resources from location-related disasters such as fires, floods, electrical outages and malicious damage. The cluster nodes are separated geographically and synchronously mirrored between sites. A geo cluster is unaware of the physical distance between its nodes. In a geographically dispersed cluster, the public and private network interfaces must exist in the same network segment and the cluster nodes must share the same IP subnet for [failover](https://susedefines.suse.com/definition/failover/) purposes.

**CHAPTER 2**

**LITERATURE SURVEY**

The taxi banks, help the mode of operation of taxis, as these are hubs where the highest frequency of taxis are hired and also has place to park the taxis when not in use.

These taxi banks are very convenient, for both the passengers and taxi drivers; as they act as hubs. Identifying the points/clusters, where taxi banks can be useful for both passengers and taxi drivers.

Various studies reveal, these hubs can be determined using K-Means Algorithm, and various others for better business.

Hence, this code, utilizes the K-Means algorithm, and further improves on it, using the DBSCAN&HDBSCAN to reduce the no. of outliers and increase the potential nodes, for better analysis to determine taxi banks.

**CHAPTER 3**

**SYSTEM ANALYSIS**

Following python modules/functions have been used in the project:-

1. matplotlib for plots and charts visualization of the outcomes.
2. Pandas for storing and manipulating data.
3. Numpy for its use in data-manipulation.
4. hdbscan and DBSCAN for spatial-clusterings (hierarchichal).
5. sklearn functionalities like Kmeans and silhouette\_score with KneighboursClassifier.
6. folium for maps and co-ordinates visualization.

K-Means Clustering Algorithm:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

[AndreyBu](https://www.liveedu.tv/andreybu/REaxr-machine-learning-model-python-sklearn-kera/), who has more than 5 years of machine learning experience and currently teaches people his skills, says that “the objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset.”

A cluster refers to a collection of data points aggregated together because of certain similarities.

You’ll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.

In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

The ‘means’ in the K-means refers to averaging of the data; that is, finding the centroid.

# **How the K-means algorithm works**

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids

It halts creating and optimizing clusters when either:

* The centroids have stabilized — there is no change in their values because the clustering has been successful.
* The defined number of iterations has been achieved.

DBSCAN:

Clusters are dense regions in the data space, separated by regions of the lower density of points. The ***DBSCAN algorithm*** is based on this intuitive notion of “clusters” and “noise”. The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.

Partitioning methods (K-means, PAM clustering) and hierarchical clustering work for finding spherical-shaped clusters or convex clusters. In other words, they are suitable only for compact and well-separated clusters. Moreover, they are also severely affected by the presence of noise and outliers in the data.

Real life data may contain irregularities, like –  
i) Clusters can be of arbitrary shape such as those shown in the figure below.  
ii) Data may contain noise.

HDBSCAN:

HDBSCAN uses a density-based approach which makes few implicit assumptions about the clusters. It is a non-parametric method that looks for a cluster hierarchy shaped by the multivariate modes of the underlying distribution. Rather than looking for clusters with a particular shape, it looks for regions of the data that are denser than the surrounding space. The mental image you can use is trying to separate the islands from the sea or mountains from its valleys.

OUTLIERS:

Outliers are single, mostly isolated data points that are far from the rest of the data. If you do not have outliers, outlier detection can hurt your data by removing small clusters or removing only a part of a scattered noise.

**CHAPTER 4**

**SOFTWARE REQUIREMENT SPECIFICATIONS**

**SOFTWARE AND HARDWARE REQUIREMENT SPECIFICATIONS**

Hardware requirements:

* Processer: Any Updated Processer
* Ram: Min 1 GB
* Hard Disk: Min 100 GB

Software requirements:

* Operating System: Windows family
* Technology : Python 3, and Panda Library.
* Web Technologies: Jupyter Notebook

**CHAPTER 5**

**SYSTEM DESIGN**

The code is divided into 8 parts, of 7 tasks.

The First part of the code determines the importing of various files from Pandas and also loads the colors going to be used in the clusters, in their hexadecimal form.

The 7 tasks of the codes are as follows:

Task 1: Exploratory Data Analysis

Task 2: Visualizing Geographical Data

Task 3: Clustering Strength / Performance Metric

Task 4: K-Means Clustering

Task 5: DBSCAN

Task 6: HDBSCAN

Task 7: Addressing Outliers

Task 1: Exploratory Data Analysis

Here the given data set is analyzed, and the longitude and latitude of the various datasets are fed to the systems.

Task 2: Visualizing Geographical Data

Here the clusters are predicted on the fed longitudes and latitudes, these predictory clusters on then placed on the map of the respective longitude and latitudes, for better analysis.

Task 3: Clustering Strength / Performance Metric

The prediction of clusters are measured here, by calculating the silhouette-score. The higher the score, the more accurate the cluster is. This score is between 0 and 1.

Task 4: K-Means Clustering

This determines how the clusters run in an interactive way. It doesn’t know how the clusters work. It starts with random values, which after every iteration, result in a more efficient cluster. Till the most efficient cluster is reached. This is done by assigning a centroid at random to different clusters, and the calculations are carried out, and re-assigned to a more efficient cluster node.

Task 5: DBSCAN

Stands for Density Based Spatial Cluster of Application Noise. Which provides an improvement over the K-Means method. The number of outliers is still high though.

Task 6: HDBSCAN

Stands for Hierarchy Density Based Spatial Cluster of Application Noise. Which provides an improvement over the DBSCAN. The number of outliers is reduced and also the cluster are improved and increased.

Task 7: Addressing Outliers

It addresses which cluster a point to be set to based on the pre-determined parameter.

**CHAPTER 6**

**IMPLEMENTATION**

**Loading of required Libraries and Colors to be used for clusters:**

**import** **matplotlib** %matplotlib inline %config InlineBackend.figure\_format = 'svg'

**import** **matplotlib.pyplot** **as** **plt** plt.style.use('ggplot')

**import** **pandas** **as** **pd** **import** **numpy** **as** **np** **from** **tqdm**

**import** tqdm **from** **sklearn.cluster**

**import** KMeans, DBSCAN **from** **sklearn.metrics**

**import** silhouette\_score **from** **sklearn.datasets**

**import** make\_blobs **from** **sklearn.neighbors**

**import** KNeighborsClassifier **from** **ipywidgets**

**import** interactive **from** **collections**

**import** defaultdict

**import** **hdbscan**

**import** **folium**

**import** **re** cols = ['#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231', '#911eb4', '#46f0f0', '#f032e6', '#bcf60c', '#fabebe', '#008080', '#e6beff', '#9a6324', '#fffac8', '#800000', '#aaffc3', '#808000', '#ffd8b1', '#000075', '#808080']\*10

**Task 1: Exploratory Data Analysis**

df = pd.read\_csv('Data/taxi\_data.csv')

df.head()

df.duplicated(subset=['LON', 'LAT']).values.any()

df.isna().values.any()

print(f'Before dropping NaNs and dupes**\t**:**\t**df.shape = **{df.shape}**')

df.dropna(inplace=**True**)

df.drop\_duplicates(subset=['LON', 'LAT'], keep='first', inplace=**True**)

print(f'After dropping NaNs and dupes**\t**:**\t**df.shape = **{df.shape}**')

Before dropping NaNs and dupes : df.shape = (838, 3)

After dropping NaNs and dupes : df.shape = (823, 3)

df.head()

X = np.array(df[['LON', 'LAT']], dtype='float64')

plt.scatter(X[:,0], X[:,1], alpha=0.2, s=50)

<matplotlib.collections.PathCollection at 0x1ec3d427208>

**Task 2: Visualizing Geographical Data**

m = folium.Map(location=[df.LAT.mean(), df.LON.mean()], zoom\_start=9,

tiles='Stamen Toner')

**for** \_, row **in** df.iterrows():

folium.CircleMarker(

location=[row.LAT, row.LON],

radius=5,

popup=re.sub(r'[^a-zA-Z ]+', '', row.NAME),

color='#1787FE',

fill=**True**,

fill\_colour='#1787FE'

).add\_to(m)

**Task 3: Clustering Strength / Performance Metric**

X\_blobs, \_ = make\_blobs(n\_samples=1000, centers=10, n\_features=2,

cluster\_std=0.5, random\_state=4)

plt.scatter(X\_blobs[:,0], X\_blobs[:,1], alpha=0.2)

<matplotlib.collections.PathCollection at 0x1ec3dcdb788>

class\_predictions = np.load('Data/sample\_clusters.npy')

unique\_clusters = np.unique(class\_predictions)

**for** unique\_cluster **in** unique\_clusters:

X = X\_blobs[class\_predictions==unique\_cluster]

plt.scatter(X[:,0], X[:,1], alpha=0.2, c=cols[unique\_cluster])

silhouette\_score(X\_blobs, class\_predictions)

class\_predictions = np.load('Data/sample\_clusters\_improved.npy')

unique\_clusters = np.unique(class\_predictions)

**for** unique\_cluster **in** unique\_clusters:

X = X\_blobs[class\_predictions==unique\_cluster]

plt.scatter(X[:,0], X[:,1], alpha=0.2, c=cols[unique\_cluster])

silhouette\_score(X\_blobs, class\_predictions)

**Task 4: K-Means Clustering**

X\_blobs, \_ = make\_blobs(n\_samples=1000, centers=50,

n\_features=2, cluster\_std=1, random\_state=4)

data = defaultdict(dict)

**for** x **in** range(1,21):

model = KMeans(n\_clusters=3, random\_state=17,

max\_iter=x, n\_init=1).fit(X\_blobs)

data[x]['class\_predictions'] = model.predict(X\_blobs)

data[x]['centroids'] = model.cluster\_centers\_

data[x]['unique\_classes'] = np.unique(class\_predictions)

**def** f(x):

class\_predictions = data[x]['class\_predictions']

centroids = data[x]['centroids']

unique\_classes = data[x]['unique\_classes']

**for** unique\_class **in** unique\_classes:

plt.scatter(X\_blobs[class\_predictions==unique\_class][:,0],

X\_blobs[class\_predictions==unique\_class][:,1],

alpha=0.3, c=cols[unique\_class])

plt.scatter(centroids[:,0], centroids[:,1], s=200, c='#000000', marker='v')

plt.ylim([-15,15]); plt.xlim([-15,15])

plt.title('How K-Means Clusters')

interactive\_plot = interactive(f, x=(1, 20))

output = interactive\_plot.children[-1]

output.layout.height = '350px'

interactive\_plot

interactive(children=(IntSlider(value=10, description='x', max=20, min=1), Output(layout=Layout(height='350px'…

X = np.array(df[['LON', 'LAT']], dtype='float64')

k = 70

model = KMeans(n\_clusters=k, random\_state=17).fit(X)

class\_predictions = model.predict(X)

df[f'CLUSTER\_kmeans**{k}**'] = class\_predictions

df.head()

**def** create\_map(df, cluster\_column):

m = folium.Map(location=[df.LAT.mean(), df.LON.mean()], zoom\_start=9, tiles='Stamen Toner')

**for** \_, row **in** df.iterrows():

**if** row[cluster\_column] == -1:

cluster\_colour = '#000000'

**else**:

cluster\_colour = cols[row[cluster\_column]]

folium.CircleMarker(

location= [row['LAT'], row['LON']],

radius=5,

popup= row[cluster\_column],

color=cluster\_colour,

fill=**True**,

fill\_color=cluster\_colour

).add\_to(m)

**return** m

m = create\_map(df, 'CLUSTER\_kmeans70')

print(f'K=**{k}**')

print(f'Silhouette Score: {silhouette\_score(X, class\_predictions)}')

m.save('kmeans\_70.html')

K=70

Silhouette Score: 0.6527069281188838

best\_silhouette, best\_k = -1, 0

**for** k **in** tqdm(range(2, 100)):

model = KMeans(n\_clusters=k, random\_state=1).fit(X)

class\_predictions = model.predict(X)

curr\_silhouette = silhouette\_score(X, class\_predictions)

**if** curr\_silhouette > best\_silhouette:

best\_k = k

best\_silhouette = curr\_silhouette

print(f'K=**{best\_k}**')

print(f'Silhouette Score: **{best\_silhouette}**')

**Task 5: DBSCAN**

Density-Based Spatial Clustering of Applications with Noise

*# code for indexing out certain values*

dummy = np.array([-1, -1, -1, 2, 3, 4, 5, -1])

new = np.array([(counter+2)\*x **if** x==-1 **else** x **for** counter, x **in** enumerate(dummy)])

model = DBSCAN(eps=0.01, min\_samples=5).fit(X)

class\_predictions = model.labels\_

df['CLUSTERS\_DBSCAN'] = class\_predictions

m = create\_map(df, 'CLUSTERS\_DBSCAN')

print(f'Number of clusters found: {len(np.unique(class\_predictions))}')

print(f'Number of outliers found: {len(class\_predictions[class\_predictions==-1])}')

print(f'Silhouette ignoring outliers: {silhouette\_score(X[class\_predictions!=-1], class\_predictions[class\_predictions!=-1])}')

no\_outliers = 0

no\_outliers = np.array([(counter+2)\*x **if** x==-1 **else** x **for** counter, x **in** enumerate(class\_predictions)])

print(f'Silhouette outliers as singletons: {silhouette\_score(X, no\_outliers)}')

**Task 6: HDBSCAN**

Hierarchical DBSCAN

model = hdbscan.HDBSCAN(min\_cluster\_size=5, min\_samples=2,

cluster\_selection\_epsilon=0.01)

*#min\_cluster\_size*

*#min\_samples*

*#cluster\_slection\_epsilon*

class\_predictions = model.fit\_predict(X)

df['CLUSTER\_HDBSCAN'] = class\_predictions

m = create\_map(df, 'CLUSTER\_HDBSCAN')

print(f'Number of clusters found: {len(np.unique(class\_predictions))-1}')

print(f'Number of outliers found: {len(class\_predictions[class\_predictions==-1])}')

print(f'Silhouette ignoring outliers: {silhouette\_score(X[class\_predictions!=-1], class\_predictions[class\_predictions!=-1])}')

no\_outliers = np.array([(counter+2)\*x **if** x==-1 **else** x **for** counter, x **in** enumerate(class\_predictions)])

print(f'Silhouette outliers as singletons: {silhouette\_score(X, no\_outliers)}')

**Task 7: Addressing Outliers**

classifier = KNeighborsClassifier(n\_neighbors=1)

df\_train = df[df.CLUSTER\_HDBSCAN!=-1]

df\_predict = df[df.CLUSTER\_HDBSCAN==-1]

X\_train = np.array(df\_train[['LON', 'LAT']], dtype='float64')

y\_train = np.array(df\_train['CLUSTER\_HDBSCAN'])

X\_predict = np.array(df\_predict[['LON', 'LAT']], dtype='float64')

classifier.fit(X\_train, y\_train)

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=1, p=2,

weights='uniform')

predictions = classifier.predict(X\_predict)

df['CLUSTER\_hybrid'] = df['CLUSTER\_HDBSCAN']

df.loc[df.CLUSTER\_HDBSCAN==-1, 'CLUSTER\_hybrid'] = predictions

m = create\_map(df, 'CLUSTER\_hybrid')

\_predictions = df.CLUSTER\_hybrid

print(f'Number of clusters found: {len(np.unique(class\_predictions))}')

print(f'Silhouette: {silhouette\_score(X, class\_predictions)}')

m.save('hybrid.html')

df['CLUSTER\_hybrid'].value\_counts().plot.hist(bins=70, alpha=0.4,

label='Hybrid')

df['CLUSTER\_kmeans70'].value\_counts().plot.hist(bins=70, alpha=0.4,

label='K-Means (70)')

plt.legend()

plt.title('Comparing Hybrid and K-Means Approaches')

plt.xlabel('Cluster Sizes')

Text(0.5, 0, 'Cluster Sizes')

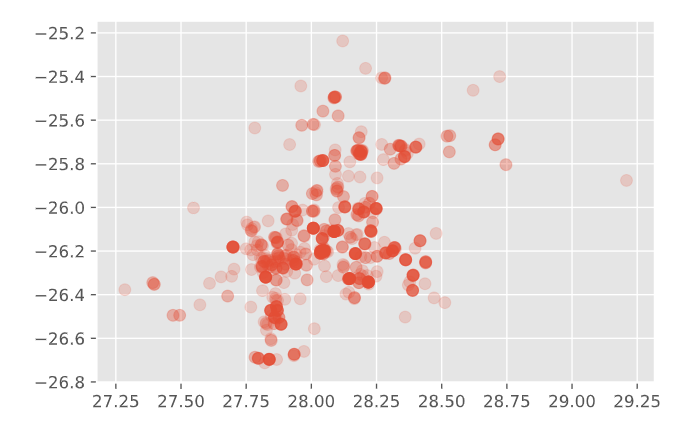
**CHAPTER 7**

**TESTING**

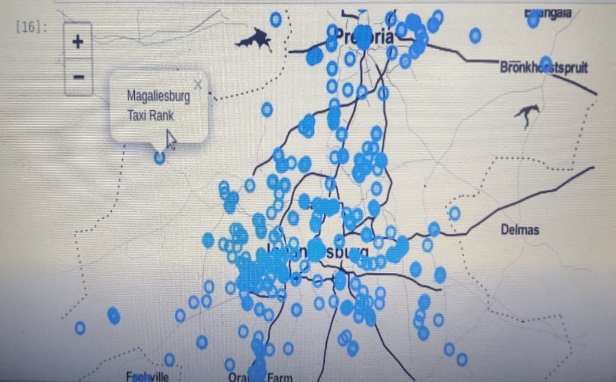
The purpose of testing is to identify the errors in the program.

All the Seven tasks are run individually to generate the clusters in a proper way with the use of the given data set.

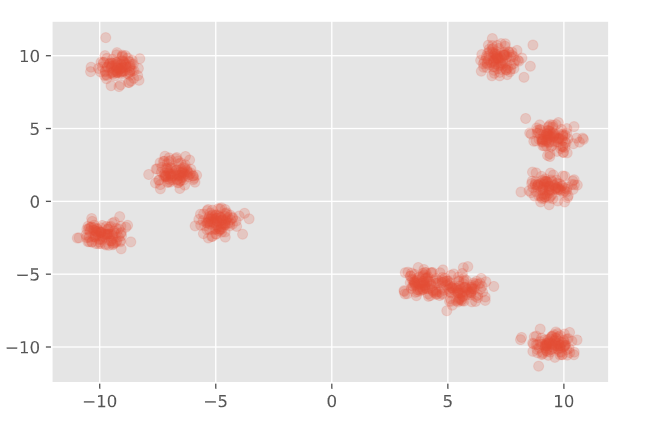
Task 1:

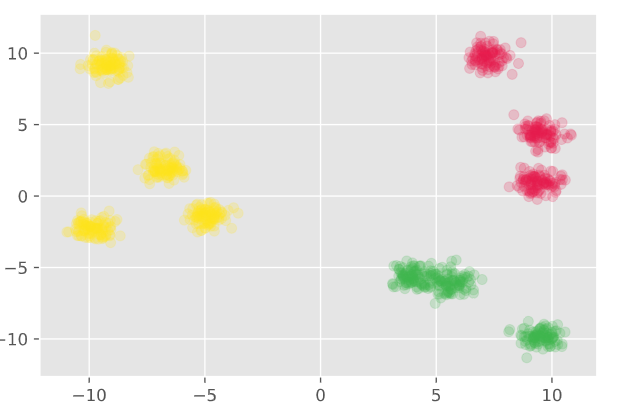


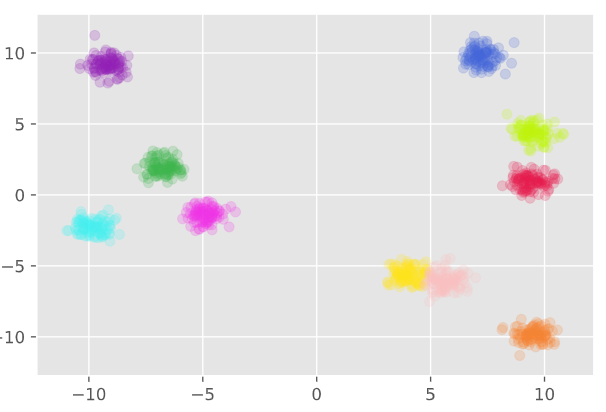
Task 2:



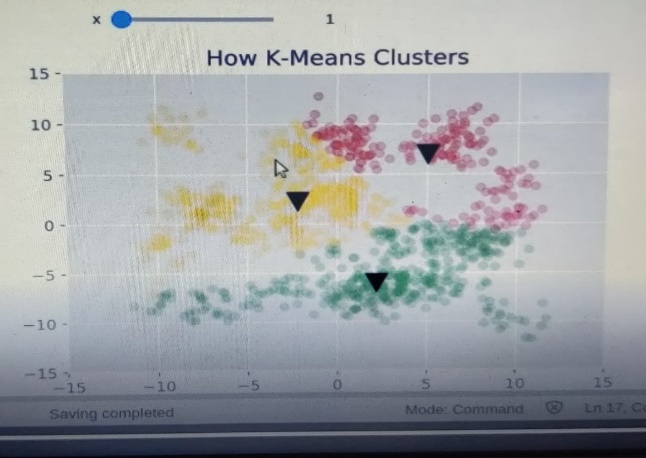
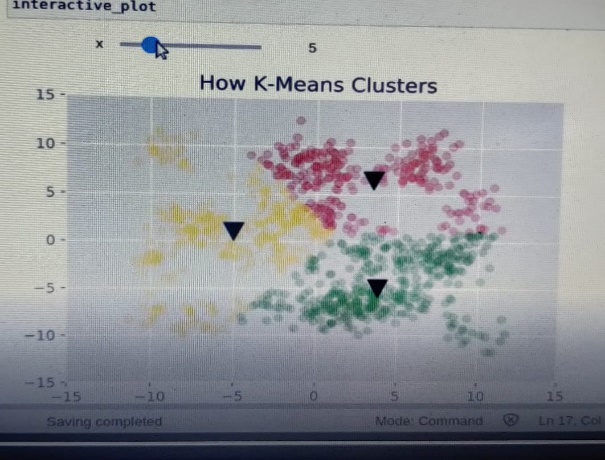
Task 3:



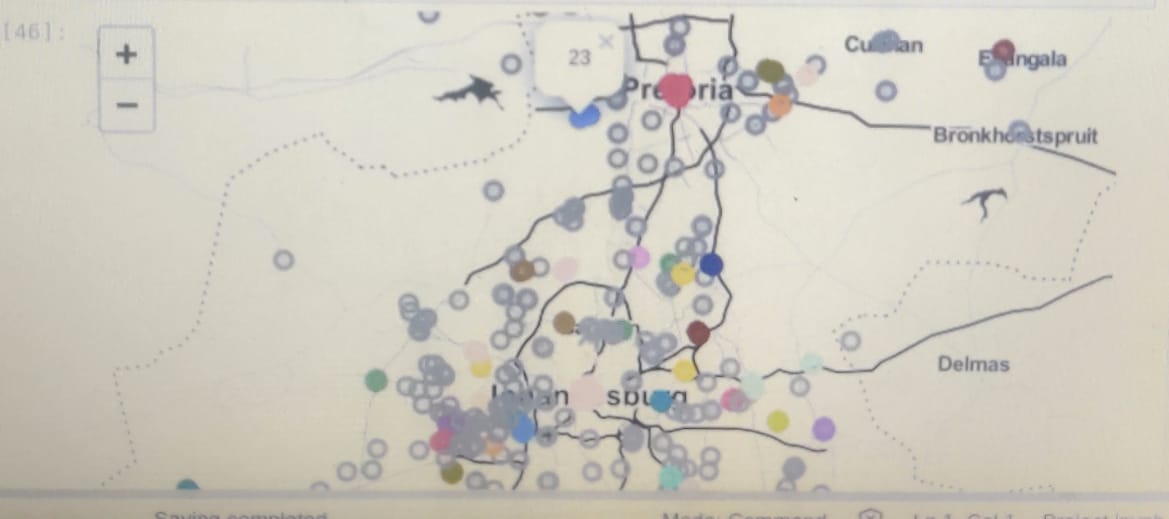
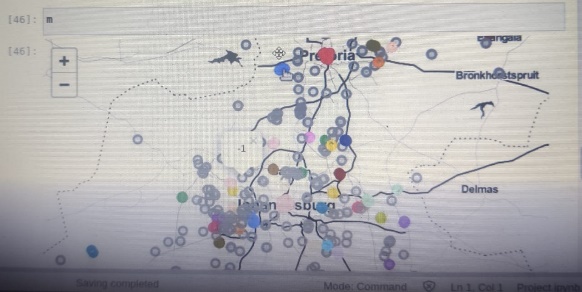


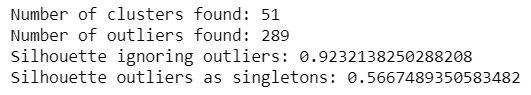


Task 4:

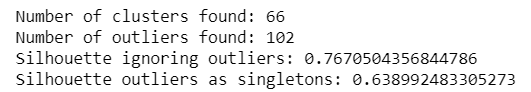
 

Task 5:

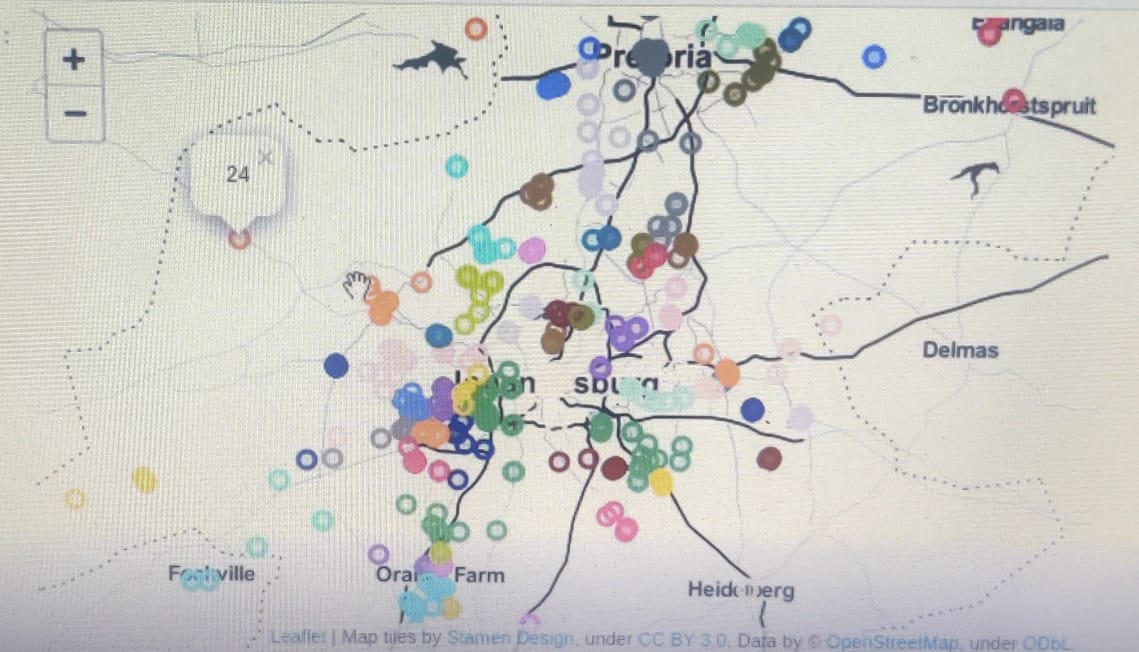
 

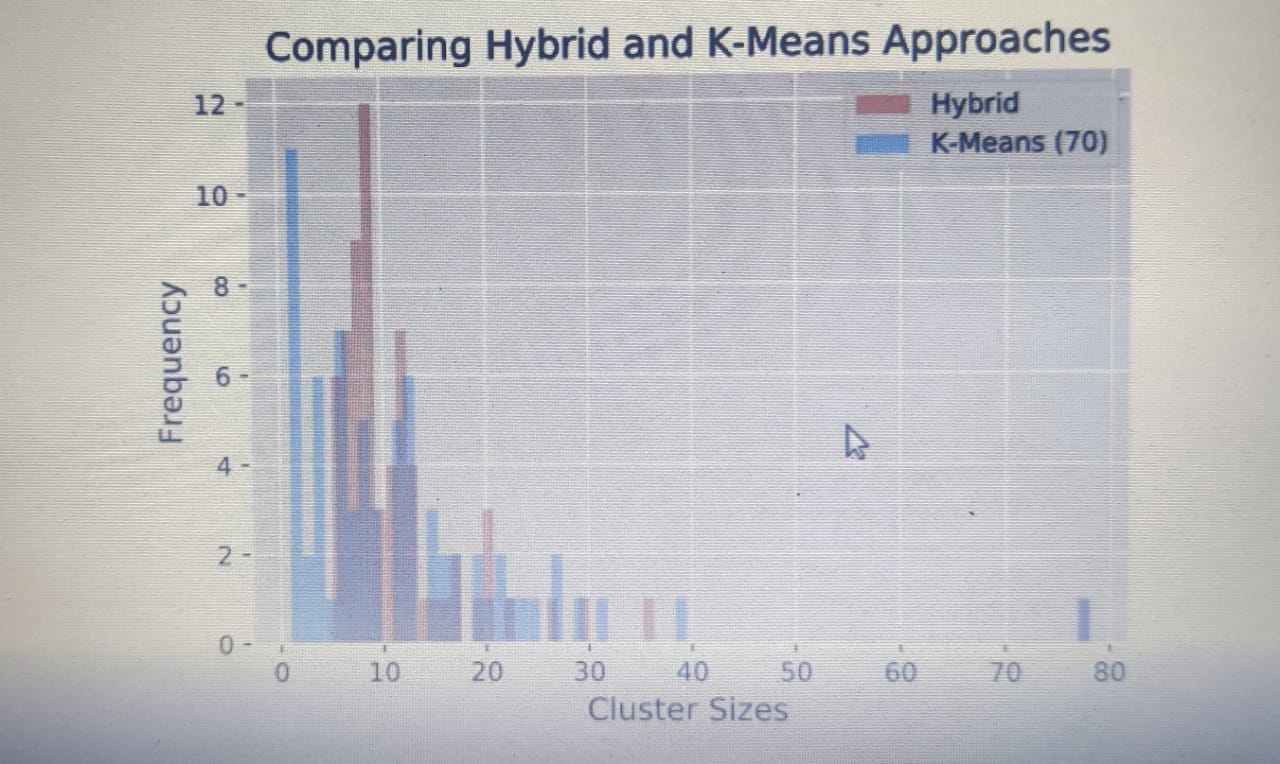


Task 6:



Task 7:





**CHAPTER 9**

**CONCLUSION**

This project used geo-location clustering to help find appropriate taxi banks, where the business or frequency of usage of a taxi is high as well and parking of the taxi is easier. It uses an example data set in Johannesburg, along with the use of K-Means algorithm, DBSCAN, HDBSCAN, coded in Python 3, to form clusters and hence, help the taxi drivers. It is executed on an IDE i.e. Jupyter Notebook.

**CHAPTER 10**

**Future Enhancement**

This Mini Project as of now works on a given data set in Johannesburg, and shows all the clusters of nodes where Taxi Banks can form and hence, increase business. These are the locations of highest traffic and usage of taxis and also place for parking them.

We want to implement this project in the future on real data sets of places in India, to help Taxi drivers in real life to increase their business through this application.

We also intend to improve the usage of the K-Means algorithm and DBSCAN&HDBSCAN for better functionality.