CLUSTERING GEO LOCATION DATA INTELLIGENTLY IN PYTHON

We have taxi rank locations. Our task is to determine where we can build our service centers/stations to accomodate the taxis in the region

Simple way : Make clusters of data and build a service ~~centre~~/station at centroid of each clusters.

Import Libraries and set colours

TASK 1 : Exploratory Data Analysis → Removing NULLS and Duplicates.

```
df = pd.read_csv('Data/taxi_data.csv')
df.head()
```

Duplicate

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

(Column name of duplicate value we need to check)

(returns true if any is true.)

Is none or nulls

```
df.isna().values.any()
df.shape() → Before dropping data
df.dropna(inplace=True)
```

→ Make changes to same data instead of returning new data

```
df.drop_duplicates(subset=['LON','LAT'], keep='first', inplace=True)
```

df.shape() → After dropping

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

X → array only of LON & LAT.

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

X[:,0] → everything from first column

X[:,1] → everything from second column

alpha=0.2 → makes everything light for better view in dense conditions.

s=50 → size of circle

TASK -2 VISUALISE GEOGRAPHICAL DATA

We need to do it so we can zoom in data to properly find clusters.

Point where our map should start.

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

tiles → colour scheme

m → It will print only the map. We need to populate it. The way we do is iteratively add all rows in data frame.

for _, row in df.iterrows():

we use underscore because iterrows sends row number and then data of row we do not need the number and hence we ignore it.

Making circle for all rows.

__/__/____

```
folium.CircleMarker(
    location = [row.LAT, row.LOW],
    radius = 5,
    popup = re.sub('[^a-zA-Z ]+', '', row.Name),
    color = '#1787FE'
    fill = True,
    fill_color = '#1787FE'
).add_to(m)
```

what will popup when we click circle → popup

space

add to map

m → Now the map contains circle for each taxi

Because we have not actually cleaned the strings in row name it is possible there might be some form of escapes that cause problem when you convert the python code to javascript or html file.

Way to Deal → Take row name and only keep characters and spaces. One way to do that is to use regex.

re.sub('[^a-zA-Z]+', '', row.name)

re.sub → substitute

'[^a-zA-Z]+' → reg exp starts; Anything Except these will be replaced by

'' → ,

row.name

TASK-3 Clustering Strength / Performance Metric

Get dummy data to work with

X_blobs, _ = make_blobs(n_samples = 1000,
array ↙ ↳ to ignore ↳ total samples

10 Clusters → centers = 10,
X & Y values → n_features = 2,
standard deviation → cluster_std = 0.5,
random_state = 4)

plt.scatter(X_blobs[:,0], X_blobs[:,1], alpha=0.2)

Algo to cluster

class_predictions = np.load('Data/Sample_clusters.npy')

class_predictions → value of each cluster number

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])

array of colours before task 1.

↓ → It gives score from -1 to +1, Better Score better clusters.

silhouette_score(X_blobs, class_predictions)
Data ↗ ↖ Cluster Labels

If we use sample_clusters_improved.npy we get better score.

K Means algo starts with random values. and can result in different.

__/__/____

TASK-4 K-MEANS CLUSTERING

```
X = np.array(df[['LON','LAT']], dtype = float64)
K = 70   # any number not defined here we can have no. of service station we can actually build.
```

fitting to model → `model = KMeans(n_clusters = k, random_state = 17).fit(X)`

predicting our clusters → `class_predictions = model.predict(X)`

moving → `df[f'cluster_kmeans{k}'] = class_predictions`

new column in our data

`df.head()` - viewing the new column.

new data-frame ↙ column we care about.

```
def create_map(df, cluster_column):
                    starting location
    m = folium.Map(location=[df.LAT.mean(),
                             df.LON.mean()],
                   starting zoom
                   zoom_start = 9,
                   our colour filler = 'Stamen Jones')
                   scheme.

    for _, row in df.iterrows():
        cluster_colours = cols[row[cluster_column]]
                          ↑
                          Colour according to cluster label.
```

__/__/____

folium.CircleMaker(
locations = [row['LAT'], row['LON']],
radius = 5,
popup = row[cluster - column],
colour = cluster - ~~column~~ colour,
fill = True,
fill_color = cluster - colour
).add_to(m).

What will show → in popup → the cluster of label.

↓ defined on prev page.

return m.

The new column name.

m = create_map(df, CLUSTER_kmeans, 70)

m → print map

Calculate best silhouette score.

~~best~~ best_silhouette, best_k = -1, 0

↑ ↑

Initial values → value -1 0

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

2-100 clusters → model = K-means(n_clusters = k
random_state = 1)
.fit(x)

class_predictions = model.predict(x)

Current score → cur_silhouette = silhouette_score(x, class_predictions)

If cur_silhouette > best_silhouette:
~~that~~ best_k = k
best_silhouette = cur_silhouette

```
print (best_k)
print (best_silhouette)
```

TASK-5 DBSCAN

Density Based Spatial Clustering of Applications with Noise.

K-means doesn't takes in account density. like more clusters in dense regions and less on outside but here we know in city centers there will be dense ~~clusters~~ points and hence more clusters

dummy with -1 as noise

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

make each noise individual clusters

```
new = np.array([(counter+2) if x==-1 else x for counter, x in enumerate(dummy)])
dummy = [-2,-3,-4,2,3,4,5,-6)
```

radius around each point

min samples in clusters.

```
model = DBSCAN(eps=0.01, min_samples=5).fit(x)
class_predictions = model.labels_   # predict won't work.
df['Clusters_DBScan'] = class_predictions
m = create_map(df, 'CLUSTERS_DBSCAN')
```

TASK-6 HDBSCAN

Hierarchical → ~~Hybrid~~ DBSCAN

DBSCAN have global understanding of density in dataset. This means it can't necessarily pickup different kinds of Density within a specific dataset.

HDBSCAN will try to learn multiple types of densities

DBSCAN takes out outliers. That is not a downside but given our data set problem we want each point to be assigned a cluster.

```
model = hdbscan.HDBSCAN(min_cluster_size=5,
                        min_samples=2,
                        cluster_selection_epsilon=
                        0.01)

class_predictions = model.fit_predict(x)
df['CLUSTER_HDBSCAN'] = class_predictions

m = create_map(df, 'CLUSTER_HDBSCAN')
```

TASK → ADDRESSING OUTLIERS

K-NN → K nearest neighbour

```
classifier = KNeighborsClassifier(n_neighbors=1)

df_train = df[df.CLUSTER_HDBSCAN != -1]   Identified Clusters
df_predict = df[df.CLUSTER_HDBSCAN == -1]   Outliers

X_train = np.array(df_train[['LON', 'LAT']], dtype='float64')
y_train = np.array
X_predict = np.array(df_predict[['LON', 'LAT']], dtype='float64')

classifier.fit(X_train, y_train) → fit the model
predictions = classifier.predict(X_predict) → predict values

df['CLUSTER_hybrid'] = df['CLUSTER_HDBSCAN']

df.loc[df.CLUSTER_HDBSCAN == -1, 'CLUSTER_hybrid'] =
```

Copy values to new column

where it is -1

Replace cluster hybrid / column with predictions

predictions

```
m = create_map(df['CLUSTER_hybrid'])
```