CODE:

import pandas as pd

import numpy as np

import seaborn as sns

data=pd.read\_csv("D:/3rd year/miniproject/food\_coded.csv")

data

data.columns

column=['cook','eating\_out','employment','ethnic\_food', 'exercise','fruit\_day','income','on\_off\_campus','pay\_meal\_out','sports','veggies\_day']

d=data[column]

d

import seaborn as sns

sns.pairplot(d)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

ax=d.boxplot(figsize=(16,6))

ax.set\_xticklabels(ax.get\_xticklabels(),rotation=30)

d.shape

s=d.dropna()

## for data

import numpy as np

import pandas as pd

## for plotting

import matplotlib.pyplot as plt

import seaborn as sns

## for geospatial

!pip install folium

import folium

!pip install geopy

import geopy

## for machine learning

from sklearn import preprocessing, cluster

import scipy

## for deep learning

!pip install minisom

import minisom

rom pandas.io.json import json\_normalize

import folium

from geopy.geocoders import Nominatim

import requests

CLIENT\_ID = "KTCJJ2YZ2143QHEZ2JAQS4FJIO5DLSDO0YN4YBXPMI5NKTEF" # your Foursquare ID

CLIENT\_SECRET = "KNG2LO22BPLHN1E3OAHWLYQ5PQBN14XYZMEMAS0CPJEJKOTR" # your Foursquare Secret

VERSION = '20200316'

LIMIT = 10000

url = 'https://api.foursquare.com/v2/venues/explore?&client\_id={}&client\_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(

CLIENT\_ID,

CLIENT\_SECRET,

VERSION,

17.448372, 78.526957,

30000,

LIMIT)

results = requests.get(url).json()

results

venues = results['response']['groups'][0]['items']

nearby\_venues = json\_normalize(venues)

nearby\_venues

resta=[]

oth=[]

for lat,long in zip(nearby\_venues['venue.location.lat'],nearby\_venues['venue.location.lng']):

url = 'https://api.foursquare.com/v2/venues/explore?&client\_id={}&client\_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(

CLIENT\_ID,

CLIENT\_SECRET,

VERSION,

lat,long,

1000,

100)

res = requests.get(url).json()

venue = res['response']['groups'][0]['items']

nearby\_venue = json\_normalize(venue)

df=nearby\_venue['venue.categories']

g=[]

for i in range(0,df.size):

g.append(df[i][0]['icon']['prefix'].find('food'))

co=0

for i in g:

if i>1:

co+=1

resta.append(co)

oth.append(len(g)-co)

nearby\_venues['restaurant']=resta

nearby\_venues['others']=oth

nearby\_venues

lat=nearby\_venues['venue.location.lat']

long=nearby\_venues['venue.location.lng']

f=['venue.location.lat','venue.location.lng']

X = nearby\_venues[f]

max\_k = 10

## iterations

distortions = []

for i in range(1, max\_k+1):

if len(X) >= i:

model = cluster.KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

model.fit(X)

distortions.append(model.inertia\_)

## best k: the lowest derivative

k = [i\*100 for i in np.diff(distortions,2)].index(min([i\*100 for i

in np.diff(distortions,2)]))

## plot

fig, ax = plt.subplots()

ax.plot(range(1, len(distortions)+1), distortions)

ax.axvline(k, ls='--', color="red", label="k = "+str(k))

ax.set(title='The Elbow Method', xlabel='Number of clusters',

ylabel="Distortion")

ax.legend()

ax.grid(True)

plt.show()

city = "Hyderabad"

## get location

locator = geopy.geocoders.Nominatim(user\_agent="MyCoder")

location = locator.geocode(city)

print(location)

## keep latitude and longitude only

location = [location.latitude, location.longitude]

print("[lat, long]:", location)

nearby\_venues.head()

nearby\_venues.columns

n=nearby\_venues.drop(['referralId', 'reasons.count', 'reasons.items', 'venue.id',

'venue.name',

'venue.location.labeledLatLngs', 'venue.location.distance',

'venue.location.cc',

'venue.categories', 'venue.photos.count', 'venue.photos.groups',

'venue.location.crossStreet', 'venue.location.address','venue.location.city',

'venue.location.state', 'venue.location.crossStreet',

'venue.location.neighborhood', 'venue.venuePage.id',

'venue.location.postalCode','venue.location.country'],axis=1)

n.columns

n

n=n.dropna()

n = n.rename(columns={'venue.location.lat': 'lat', 'venue.location.lng': 'long'})

n

n['venue.location.formattedAddress']

spec\_chars = ["[","]"]

for char in spec\_chars:

n['venue.location.formattedAddress'] = n['venue.location.formattedAddress'].astype(str).str.replace(char, ' ')

n

x, y = "lat", "long"

color = "restaurant"

size = "others"

popup = "venue.location.formattedAddress"

data = n.copy()

## create color column

lst\_colors=["red","green","orange"]

lst\_elements = sorted(list(n[color].unique()))

## create size column (scaled)

scaler = preprocessing.MinMaxScaler(feature\_range=(3,15))

data["size"] = scaler.fit\_transform(

data[size].values.reshape(-1,1)).reshape(-1)

## initialize the map with the starting location

map\_ = folium.Map(location=location, tiles="cartodbpositron",

zoom\_start=11)

## add points

data.apply(lambda row: folium.CircleMarker(

location=[row[x],row[y]],popup=row[popup],

radius=row["size"]).add\_to(map\_), axis=1)

## add html legend

## plot the map

map\_

X = n[["lat","long"]]

max\_k = 10

## iterations

distortions = []

for i in range(1, max\_k+1):

if len(X) >= i:

model = cluster.KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

model.fit(X)

distortions.append(model.inertia\_)

## best k: the lowest derivative

k = [i\*100 for i in np.diff(distortions,2)].index(min([i\*100 for i in np.diff(distortions,2)]))

## plot

fig, ax = plt.subplots()

ax.plot(range(1, len(distortions)+1), distortions)

ax.axvline(k, ls='--', color="red", label="k = "+str(k))

ax.set(title='The Elbow Method', xlabel='Number of clusters',

ylabel="Distortion")

ax.legend()

ax.grid(True)

plt.show()

k = 6

model = cluster.KMeans(n\_clusters=k, init='k-means++')

X = n[["lat","long"]]

## clustering

dtf\_X = X.copy()

dtf\_X["cluster"] = model.fit\_predict(X)

## find real centroids

closest, distances = scipy.cluster.vq.vq(model.cluster\_centers\_,

dtf\_X.drop("cluster", axis=1).values)

dtf\_X["centroids"] = 0

for i in closest:

dtf\_X["centroids"].iloc[i] = 1

## add clustering info to the original dataset

n[["cluster","centroids"]] = dtf\_X[["cluster","centroids"]]

n

## plot

fig, ax = plt.subplots()

sns.scatterplot(x="lat", y="long", data=n,

palette=sns.color\_palette("bright",k),

hue='cluster', size="centroids", size\_order=[1,0],

legend="brief", ax=ax).set\_title('Clustering (k='+str(k)+')')

th\_centroids = model.cluster\_centers\_

ax.scatter(th\_centroids[:,0], th\_centroids[:,1], s=50, c='black',

marker="x")

model = cluster.AffinityPropagation()

k = n["cluster"].nunique()

sns.scatterplot(x="lat", y="long", data=n,

palette=sns.color\_palette("bright",k),

hue='cluster', size="centroids", size\_order=[1,0],

legend="brief").set\_title('Clustering (k='+str(k)+')')

x, y = "lat", "long"

color = "cluster"

size = "restaurant"

popup = "venue.location.formattedAddress"

marker = "centroids"

data = n.copy()

## create color column

lst\_elements = sorted(list(n[color].unique()))

lst\_colors = ['#%06X' % np.random.randint(0, 0xFFFFFF) for i in

range(len(lst\_elements))]

data["color"] = data[color].apply(lambda x:

lst\_colors[lst\_elements.index(x)])

## create size column (scaled)

scaler = preprocessing.MinMaxScaler(feature\_range=(3,15))

data["size"] = scaler.fit\_transform(

data[size].values.reshape(-1,1)).reshape(-1)

## initialize the map with the starting location

map\_ = folium.Map(location=location, tiles="cartodbpositron",

zoom\_start=11)

## add points

data.apply(lambda row: folium.CircleMarker(

location=[row[x],row[y]],

color=row["color"], fill=True,popup=row[popup],

radius=row["size"]).add\_to(map\_), axis=1)

## add html legend

legend\_html = """<div style="position:fixed; bottom:10px; left:10px; border:2px solid black; z-index:9999; font-size:14px;">&nbsp;<b>"""+color+""":</b><br>"""

for i in lst\_elements:

legend\_html = legend\_html+"""&nbsp;<i class="fa fa-circle

fa-1x" style="color:"""+lst\_colors[lst\_elements.index(i)]+"""">

</i>&nbsp;"""+str(i)+"""<br>"""

legend\_html = legend\_html+"""</div>"""

map\_.get\_root().html.add\_child(folium.Element(legend\_html))

## add centroids marker

lst\_elements = sorted(list(n[marker].unique()))

data[data[marker]==1].apply(lambda row:

folium.Marker(location=[row[x],row[y]],

draggable=False, popup=row[popup] ,

icon=folium.Icon(color="black")).add\_to(map\_), axis=1)

## plot the map

map\_