import pandas as pd

from pandas import json\_normalize

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

import folium as f

import json

import requests

# Load the dataset

full\_dataset = pd.read\_csv("food\_coded.csv")

# Select relevant columns

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

print(full\_dataset.head())

print(full\_dataset.info())

#print(full\_dataset.describe())

missing\_values\_count = relevent.isnull().sum()

print(missing\_values\_count)

# how many total missing values do we have?

total\_cells = np.product(relevent.shape)

total\_missing = missing\_values\_count.sum()

# percent of data that is missing

percent\_missing = (total\_missing/total\_cells) \* 100

print(percent\_missing)

#As percent missing is less than 5%,we are dropping rows containing missing values.

relevent\_with\_no\_na=relevent.dropna(axis=0)

print(relevent\_with\_no\_na.shape[0])

# Boxplot

fig = plt.figure(figsize=(10, 5))

ax = sns.boxplot(data=relevent\_with\_no\_na, linewidth=2)

plt.show()

# Foursquare API Credentials

CLIENT\_ID = 'YVXU3P2WSRWZAFQPVSSO3HHUVWZ5TIMEW1NNKENYURXHMMKD'

CLIENT\_SECRET = 'Q3ABL5LX1B2AXBEOTM3AIL3NM2DYAJ2JEYJNFMPHYWZJRXHR'

VERSION = '20200604'

LIMIT = 200

# Example location

latitude = 17.50

longitude = 78.48

# Foursquare API request

search\_query = 'Apartment'

radius = 100000

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

CLIENT\_ID, CLIENT\_SECRET, latitude, longitude, VERSION, search\_query, radius, LIMIT)

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

# Extract venue information

venues = results['response']['venues']

dataframe = pd.json\_normalize(venues)

filtered\_columns = ['name', 'categories'] + [col for col in dataframe.columns if col.startswith('location.')] + ['id']

dataframe\_filtered = dataframe.loc[:, filtered\_columns]

# Function to extract category of the venue

def get\_category\_type(row):

try:

categories\_list = row['categories']

except:

categories\_list = row['venue.categories']

if len(categories\_list) == 0:

return None

else:

return categories\_list[0]['name']

# Apply the function to filter the category for each row

dataframe\_filtered['categories'] = dataframe\_filtered.apply(get\_category\_type, axis=1)

dataframe\_filtered.columns = [column.split('.')[-1] for column in dataframe\_filtered.columns]

dataframe\_filtered.drop([4, 17, 18], axis=0, inplace=True)

dataframe\_filtered.drop(['cc', 'country', 'state', 'city'], axis=1, inplace=True)

# Map visualization

map\_bang = f.Map(location=[latitude,longitude], zoom\_start=12)

locations = f.map.FeatureGroup()

latitudes = list(dataframe\_filtered.lat)

longitudes = list(dataframe\_filtered.lng)

labels = list(dataframe\_filtered.name)

for lat, lng, label in zip(latitudes, longitudes, labels):

f.Marker([lat, lng], popup=label).add\_to(map\_bang)

# add incidents to map

map\_bang.add\_child(locations)

map\_bang

df\_evaluate=dataframe\_filtered[['lat','lng']]

RestList=[]

latitudes = list(dataframe\_filtered.lat)

longitudes = list( dataframe\_filtered.lng)

for lat, lng in zip(latitudes, longitudes):

radius = 100000

latitude=lat#Query for the apartment location in question

longitude=lng

url = 'https://api.foursquare.com/v2/venues/search?client\_id={}&client\_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.format(CLIENT\_ID, CLIENT\_SECRET, latitude, longitude, VERSION, search\_query, radius, LIMIT)

search\_query = 'Restaurant' #Search for any food related locations

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

# assign relevant part of JSON to venues

venues = results['response']['venues']

# tranform venues into a dataframe

dataframe2 = json\_normalize(venues)

filtered\_columns = ['name', 'categories'] + [col for col in dataframe2.columns if col.startswith('location.')] + ['id']

dataframe\_filtered2 = dataframe2.loc[:, filtered\_columns]

# filter the category for each row

dataframe\_filtered2['categories'] = dataframe\_filtered2.apply(get\_category\_type, axis=1)

# clean column names by keeping only last term

dataframe\_filtered2.columns = [column.split('.')[-1] for column in dataframe\_filtered2.columns]

RestList.append(dataframe\_filtered2['categories'].count())

df\_evaluate['Restaurants']=RestList

FruitList=[]

latitudes = list(dataframe\_filtered.lat)

longitudes = list( dataframe\_filtered.lng)

for lat, lng in zip(latitudes, longitudes):

radius = 100000

latitude=lat#Query for the apartment location in question

longitude=lng

url = 'https://api.foursquare.com/v2/venues/search?client\_id={}&client\_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.format(CLIENT\_ID, CLIENT\_SECRET, latitude, longitude, VERSION, search\_query, radius, LIMIT)

search\_query = 'Fruit' #Search for any food related locations

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

# assign relevant part of JSON to venues

venues = results['response']['venues']

# tranform venues into a dataframe

dataframe2 = json\_normalize(venues)

filtered\_columns = ['name', 'categories'] + [col for col in dataframe2.columns if col.startswith('location.')] + ['id']

dataframe\_filtered2 = dataframe2.loc[:, filtered\_columns]

# filter the category for each row

dataframe\_filtered2['categories'] = dataframe\_filtered2.apply(get\_category\_type, axis=1)

# clean column names by keeping only last term

dataframe\_filtered2.columns = [column.split('.')[-1] for column in dataframe\_filtered2.columns]

FruitList.append(dataframe\_filtered2['categories'].count())

df\_evaluate['Fruits,Vegetables,Groceries']=FruitList

# Elbow Method for determining the optimal number of clusters (k)

inertia\_values = []

possible\_k\_values = range(1, 11)

for k in possible\_k\_values:

kmeans = KMeans(n\_clusters=k, random\_state=0)

kmeans.fit(df\_evaluate[['lat', 'lng']])

inertia\_values.append(kmeans.inertia\_)

# Plot the elbow curve

plt.plot(possible\_k\_values, inertia\_values, marker='o')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Within-Cluster Sum of Squares (Inertia)')

plt.show()

# Choose the optimal k based on the elbow plot (manually)

kclusters = 3 # Adjust this based on the elbow in the plot

# run k-means clustering

kmeans = KMeans(n\_clusters=kclusters, random\_state=0).fit(df\_evaluate)

df\_evaluate['Cluster']=kmeans.labels\_

df\_evaluate['Cluster']=df\_evaluate['Cluster'].apply(str)

df\_evaluate.head(10)

# calculate silhouette score

sil\_score = silhouette\_score(df\_evaluate[['lat', 'lng']], df\_evaluate['Cluster'])

print(f"Silhouette Score: {sil\_score}")

#define coordinates of the college

map\_bang=f.Map(location=[latitude,longitude],zoom\_start=12)

# instantiate a feature group for the incidents in the dataframe

locations = f.map.FeatureGroup()

# set color scheme for the clusters

def color\_producer(cluster):

if cluster=='0':

return 'green'

elif cluster=='1':

return 'orange'

else:

return 'red'

latitudes = list(df\_evaluate.lat)

longitudes = list(df\_evaluate.lng)

labels = list(df\_evaluate.Cluster)

names=list(dataframe\_filtered.name)

for lat, lng, label,names in zip(latitudes, longitudes, labels,names):

f.CircleMarker(

[lat,lng],

fill=True,

fill\_opacity=1,

popup=f.Popup(names, max\_width = 300),

radius=5,

color=color\_producer(label)

).add\_to(map\_bang)

# add locations to map

map\_bang.add\_child(locations)

# Save the map as an HTML file

map\_bang.save("map\_for\_kmeans.html")