Data Collection

Pandas:

Pandas is one of the tools in Machine Learning which is used for data cleaning and analysis. It has features which are used for exploring, cleaning, transforming and visualizing from data.

The data is taken in the form of csv file. Hence we read_csv() to read the data.

Data Cleaning

The process of Extracting the features, (and dealing with different kinds of values as well as NaN values) is known as Data Cleaning.

Dropna() is function which is use to remove nan values in case any.

Data Exploration and Visualisation

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.

Pairplot():The Seaborn Pairplot allows us to plot pairwise relationships between variables within a dataset. This creates a nice visualisation and helps us understand the data by summarising a large amount of data in a single figure. This is essential when we are exploring our dataset and trying to become familiar with it.

Boxplot of Dataset

NumPy is a popular library for numerical operations in Python.

pandas is a powerful library for data manipulation and analysis.

`matplotlib.pyplot` is a module within the Matplotlib library that provides a collection of functions and classes for creating various types of data visualizations, such as line plots, scatter plots, bar charts, histograms

Creating a Boxplot:

- `ax = d.boxplot(figsize=(16,6))`: This code creates a boxplot from a DataFrame or Series named 'd' and assigns it to the variable 'ax'. A boxplot is a graphical representation of the distribution of a dataset, showing the median, quartiles, and potential outliers.
- `figsize=(16, 6)`: This argument specifies the size of the plot in inches, with a width of 16 inches and a height of 6 inches.

Setting X-axis Tick Labels:

- `ax.set_xticklabels(ax.get_xticklabels(), rotation=30)`: This line sets the tick labels on the x-axis of the boxplot. It uses `ax.get_xticklabels()` to retrieve the current tick labels and then applies a rotation of 30 degrees to them. This rotation is typically used to make long labels on the x-axis more readable when they might overlap.

the shape() method is the actual number of elements that represent the value of the dimension of the object.

Run KMeans Clustering on the data

1. **For Data Manipulation**:
   - `import numpy as np`: Imports the NumPy library with the alias 'np'. NumPy is used for numerical operations and working with arrays and matrices.
   - `import pandas as pd`: Imports the pandas library with the alias 'pd'. pandas is a powerful library for data manipulation and analysis, particularly for working with structured data in tabular form.
2. **For Plotting**:

- `import matplotlib.pyplot as plt`: Imports the pyplot module from Matplotlib with the alias 'plt'. Matplotlib is a popular library for creating various types of data visualizations, such as charts, graphs, and plots.
- `import seaborn as sns`: Imports the seaborn library with the alias 'sns'. seaborn is built on top of Matplotlib and provides a high-level interface for creating more visually appealing and informative statistical graphics.

3. **For Geospatial Data**:
   - `import folium`: Imports the Folium library, which is used for creating interactive and visually appealing maps. Folium is often used in data visualization projects that involve geographic data.
   - `import geopy`: Imports the geopy library, which provides tools for geocoding (converting addresses to coordinates) and calculating distances between coordinates.

4.For Machine Learning:

- `from sklearn import preprocessing, cluster`: Imports the preprocessing and cluster modules from scikit-learn (sklearn). scikit-learn is a machine learning library that provides a wide range of tools for tasks such as data preprocessing, clustering, classification, and regression.

5.For Deep Learning:

- `import minisom`: Imports the minisom library, which stands for "Minimalistic Self-Organizing Maps." Self-organizing maps are a type of neural network used for unsupervised learning and data clustering.

The model is fitted to the data ('X' contains the selected features) and the inertia (a measure of within-cluster variance) is computed and stored in the 'distortions' list.

Plotting the Elbow Curve:

- The code creates a plot showing the distortion values for different values of 'k'.
- It adds a vertical dashed line at the identified optimal 'k' value to help visualize the elbow point.
- The plot is labeled and displayed.

The Elbow Method is a common technique for determining the appropriate number of clusters in K-Means clustering. It looks for

the point on the plot where adding more clusters does not significantly reduce the within-cluster variance (distortion) anymore, which is often indicative of a reasonable cluster count for the data. This value of 'k' is typically chosen as the optimal number of clusters for further analysis.

Get Geolocational Data

In [ ]:

1. `from pandas.io.json import json_normalize`:
    - Imports the `json_normalize` function from the pandas library. This function is used to convert nested JSON data into a tabular format, making it easier to work with.
2. `import folium`:
    - Imports the Folium library, which is used for creating interactive and visually appealing maps in Python.
3. `from geopy.geocoders import Nominatim`:
    - Imports the `Nominatim` geocoder from the geopy library. This geocoder is used for converting between addresses and geographic coordinates (latitude and longitude).
4. **Foursquare API Credentials**:
    - `CLIENT_ID`: This variable holds your Foursquare API client ID, which is required for making authenticated requests to the Foursquare API.
    - `CLIENT_SECRET`: This variable holds your Foursquare API client secret, which is also required for authentication.
    5. `VERSION`:
        - Sets the version of the Foursquare API that you want to use. In this case, it's set to '20200316'.
    6. `LIMIT`:
        - Specifies a limit for the number of results returned in Foursquare API queries. In this code, it's set to 10,000, indicating that you want to retrieve up to 10,000 results per query.

These credentials and settings are typically used when interacting with the Foursquare API to retrieve information about venues, locations, and other geographic data.

```
venues = results['response']['groups'][0]['items']
nearby_venues = json_normalize(venues)
```

1. `results['response']['groups'][0]['items']`:
    - This expression navigates through the JSON structure to access the list of items representing nearby venues. Here's a breakdown:
        - `results['response']`: Accesses the 'response' key in the JSON, which typically contains the response data from the Foursquare API.
        - `['groups'][0]`: Navigates into the first group (often the "Recommended Places" group) within the response.
        - `['items']`: Retrieves the list of items within this group, where each item represents a nearby venue.
2. `json_normalize(venues)`:
    - This function, `json_normalize`, is used to convert the list of venue items (previously extracted) into a flat DataFrame or table-like structure. This is often done because the data retrieved from APIs like Foursquare may be nested or hierarchical JSON, and normalizing it makes it easier to work with as a tabular data structure.

In summary, this code extracts the list of nearby venues from the Foursquare API response and converts it into a structured DataFrame for further analysis or visualization. The resulting `nearby_venues` DataFrame would contain information about the venues, such as their names, categories, locations, and other details.

Adding two more Columns Restaurant and Others

1. Restaurant: Number of Restaurant in the radius of 20 km
2. others:Number of Gyms, Parks,etc in the radius of 20 km

This code appears to be using the Foursquare API to explore venues near specific geographic coordinates (latitude and longitude) and then categorize these venues into two groups: restaurants and others. Here's a detailed explanation of each part of the code:

1. **Loop Through Nearby Venues**:
   - The code uses a `for` loop to iterate through each pair of latitude (`lat`) and longitude (`long`) from the 'nearby_venues' DataFrame. These coordinates represent the locations of nearby venues.
2. **Foursquare API Request**:
   - Inside the loop, a URL is constructed for making an API request to Foursquare using the provided credentials (`CLIENT_ID` and `CLIENT_SECRET`). This request is made to the 'venues/explore' endpoint.
   - The URL includes parameters such as the version (`VERSION`), the latitude and longitude (`lat` and `long`) of the current venue, a radius of 1000 meters, and a limit of 100 venues to retrieve.
3. **API Response Processing**:
   - The API response (`res`) is obtained by sending an HTTP GET request to the constructed URL. The response is expected to be in JSON format.
   - The code then extracts the list of venue items from the response using `res['response']['groups'][0]['items']`.
4. **Normalizing Venue Data**:
   - The venue items are normalized into a DataFrame called 'nearby_venue' using `json_normalize`. This step flattens the nested JSON data, making it easier to work with.
5. **Categorizing Venues**:
   - The code extracts the 'venue.categories' column from 'nearby_venue', which contains information about the categories of each venue.
   - It then iterates through this list and checks whether each category belongs to the 'food' category by examining the 'icon' information in the category data. The presence of 'food' in the 'icon' prefix suggests that the venue is a restaurant.
   - The code counts the number of restaurants (`co`) and the number of other types of venues (`len(g) - co`) within the explored area.
6. **Appending Counts to Lists**:
   - The counts of restaurants (`co`) and other venues (`len(g) - co`) are appended to two separate lists: 'resta' and 'oth', respectively.
7. **Adding Columns to 'nearby_venues' DataFrame**:
   - Finally, the code adds two new columns to the 'nearby_venues' DataFrame: 'restaurant' and 'others'. These columns contain the counts of restaurants and other venues in the vicinity of each explored location.

8. **Resulting DataFrame**:
    - The 'nearby_venues' DataFrame now contains information about the original venues along with counts of nearby restaurants and other types of venues for each location.

This code essentially uses the Foursquare API to analyze the types of venues (restaurants and others) near a list of coordinates, providing insights into the surrounding amenities and helping categorize the venues based on their type.

Run K Means clustering on the dataset, with the optimal K value using Elbow Method

This code performs the "Elbow Method" to determine the optimal number of clusters (k) for a K-Means clustering algorithm using latitude and longitude coordinates from the 'nearby_venues' DataFrame. Here's a short explanation of each part:

1. **Feature Selection**:
    - `f=['venue.location.lat','venue.location.lng']`: Defines a list 'f' containing the names of two columns, 'venue.location.lat' (latitude) and 'venue.location.lng' (longitude), which will be used as features for clustering.
2. **K-Means Clustering**:
    - The code iterates over different values of 'k' (number of clusters) from 1 to 'max_k' (which is set to 10).
    - For each 'k', it initializes a K-Means clustering model with parameters such as 'k-means++' initialization, maximum iterations, and random seed.
    - The model is fitted to the latitude and longitude data ('X'), and the inertia (a measure of within-cluster variance) is computed and stored in the 'distortions' list.
3. **Finding the Optimal K**:
    - The code calculates the second derivative of the distortion values and identifies the value of 'k' where this second derivative is the lowest. This point is often referred to as the "elbow point," indicating a reasonable number of clusters.
4. **Plotting the Elbow Curve**:
    - The code creates a plot showing the distortion values for different values of 'k'.
    - It adds a vertical dashed line at the identified optimal 'k' value (the elbow point) to help visualize the ideal cluster count.
    - The plot is labeled and displayed.

The Elbow Method is a common technique used to determine the appropriate number of clusters in K-Means clustering. It helps identify a value of 'k' that represents a trade-off between capturing meaningful clusters and minimizing within-cluster variance. The identified 'k' can guide the clustering process for further analysis or segmentation of geographical data.

```
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)
```

This code retrieves the latitude and longitude coordinates of a given city, in this case, "Hyderabad," using the Geopy library's Nominatim geocoder. Here's a short explanation:

1. **City Name**:
    - `city = "Hyderabad"`: Specifies the name of the city you want to obtain geographic coordinates for, in this case, "Hyderabad."
2. **Geolocation Lookup**:
    - `locator = geopy.geocoders.Nominatim(user_agent="MyCoder")`: Initializes a Nominatim geocoder with a user-agent string. The user-agent string helps identify the source of the geocoding request.
3. **Location Retrieval**:
    - `location = locator.geocode(city)`: Uses the geocoder to look up the geographic coordinates (latitude and longitude) of the specified city. The result is stored in the 'location' variable.

1. **Printing Location**:
    - `print(location)`: Prints the geocoded location, which typically includes the city name, state, country, and geographic coordinates.
2. **Extracting Latitude and Longitude**:
    - `location = [location.latitude, location.longitude]`: Extracts the latitude and longitude coordinates from the 'location' object and stores them in a list called 'location'. This list now contains the [latitude, longitude] pair.
3. **Printing Latitude and Longitude**:
    - `print("[lat, long]:", location)`: Prints the latitude and longitude coordinates of the city in the format "[lat, long]" to the console.

In summary, this code uses Geopy's Nominatim geocoder to find the geographic coordinates of the specified city ("Hyderabad") and then prints both the complete location information and the latitude and longitude coordinates of that city.

Plot the clustered locations on a map

This code appears to create an interactive map using the Folium library in Python. The map displays location markers with different colors and sizes based on certain data attributes. Here's a short explanation of each part:

1. **Variable Assignments**:
   - `x, y = "lat", "long"`: Defines the latitude and longitude column names as "lat" and "long," respectively.
   - `color = "restaurant"`: Specifies the column name ("restaurant") used to determine the color of the location markers on the map.
   - `size = "others"`: Specifies the column name ("others") used to determine the size of the location markers on the map.
   - `popup = "venue.location.formattedAddress"`: Specifies the column name ("venue.location.formattedAddress") containing the information to be displayed in pop-up windows when markers are clicked.
   - `data = n.copy()`: Creates a copy of the DataFrame "n" and assigns it to the variable "data."
2. **Color Assignment**:
   - `lst_colors` and `lst_elements` are used to define a color palette for the markers based on unique values in the "restaurant" column.
3. **Size Scaling**:
   - The code scales the values in the "others" column to determine the marker sizes using the `preprocessing.MinMaxScaler`. The scaled values are stored in a new column called "size."
4. **Map Initialization**:
   - `map_ = folium.Map(location=location, tiles="cartodbpositron", zoom_start=11)`: Initializes a Folium map centered at the location specified by the "location" variable. It sets the map tiles to "cartodbpositron" and the initial zoom level to 11.
5. **Adding Location Markers**:
   - `data.apply(...)`: Iterates through the rows of the "data" DataFrame and adds circular markers to the map using Folium's `CircleMarker` function. The location, pop-up content, radius (size), and color of each marker are determined based on the data in each row.
6. **Plotting the Map**:

- `map_`: Finally, the map is displayed.

This code essentially creates an interactive map with location markers that vary in color and size based on the data attributes "restaurant" and "others." When you click on a marker, a pop-up window displays information from the "venue.location.formattedAddress" column. This kind of map visualization is often used to provide an overview of geographic data and highlight specific attributes of locations.