```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path =
'https://raw.githubusercontent.com/gagan-iitb/DataAnalyticsAndVisualiz
ation/refs/heads/main/Lab-W25/dataset/Person_Data.xlsx' # Update with
the correct file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
import pandas as pd

# Step 1: Shift non-NaN values to the left
for index, row in df.iterrows():
    non_nan_values = row.dropna().values
    num_nans = row.isna().sum()
    df.iloc[index] = list(non_nan_values) + [None] * num_nans

# Step 2: Sort each row alphabetically
for index, row in df.iterrows():
    sorted_values = sorted(row.dropna().astype(str).values)
    num_nans = row.isna().sum()
    df.iloc[index] = list(sorted_values) + [None] * num_nans

# Step 3: Sort the rows by the number of non-NaN values (most filled
at the top)
df['num_non_nan'] = df.notna().sum(axis=1)
df.sort_values(by='num_non_nan', ascending=False, inplace=True)
df.drop(columns=['num_non_nan'], inplace=True)

# Step 4: Drop columns where all values are NaN
df.dropna(axis=1, how='all', inplace=True)

# The dataframe is now cleaned, sorted by row and column
alphabetically
df.head()
```

```
<ipython-input-5-ef650feb5360>:13: FutureWarning: Setting an item of
incompatible dtype is deprecated and will raise an error in a future
version of pandas. Value '1.0' has dtype incompatible with float64,
please explicitly cast to a compatible dtype first.
  df.iloc[index] = list(sorted_values) + [None] * num_nans
```

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 49,\n  \"fields\": [\n
{\n      \"column\": \"Sno\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 48,\n
```

```
\"samples\": [\n          \"19 years old\",\n          \"39.0\",\n
\"11.0\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Person\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 44,\n        \"samples\": [\n          \"55
years\",\n          \"21 Years Old\",\n          \"5.4 feet\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Attributes\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 47,\n        \"samples\": [\n
\"Helpful\",\n          \"Sudarshan\",\n          \"9.0\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Unnamed: 3\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 43,\n        \"samples\": [\n
\"Helpful\",\n          \"Awareness\",\n          \"Humorous\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Unnamed: 4\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 43,\n        \"samples\": [\n
\"friendly\",\n          \"Atheletic Figure\",\n          \"Akash\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Unnamed: 5\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 44,\n        \"samples\": [\n
\"old\",\n          \"Color: Brown\",\n          \"E\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Unnamed: 6\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 41,\n        \"samples\": [\n          \"Doing
Engineering\",\n          \"hard working\",\n          \"caring\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Unnamed: 7\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n
\"netflix\",\n          \"Short tempered\",\n          \"helping\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Unnamed: 8\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 34,\n        \"samples\": [\n
\"joyful\",\n          \"mature\",\n          \"Suresh\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Unnamed: 9\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 28,\n        \"samples\": [\n
\"Male\",\n          \"Very Arogent\",\n          \"hardworking\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Unnamed: 10\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
```

\"num_unique_values\": 19,\n   \"samples\": [\n
\"Glasses: True\",\n   \"mental\",\n   \"sports\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Unnamed: 11\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 16,\n   \"samples\": [\n   \"H\",\n
\"loving\",\n   \"Slim\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Unnamed: 12\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 14,\n   \"samples\": [\n   \"Skin
Color: Brown\",\n   \"Understanding\",\n   \"H Color:
Black\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Unnamed: 13\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 9,\n   \"samples\":
[\n   \"weak communication\",\n   \"naughty\",\n
\"Voilent\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Unnamed: 14\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 7,\n   \"samples\":
[\n   \"Job: Asstt Professor\",\n   \"painter\",\n
\"Writer\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Unnamed: 15\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 5,\n   \"samples\":
[\n   \"pretty\",\n   \"support\",\n
\"simple\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Unnamed: 16\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   \"sister\",\n   \"Skin Type: Dry\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Unnamed: 17\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n
\"small\",\n   \"Skintone: Pale\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Unnamed: 18\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n
\"Working: True\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df"}

```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt

text_data = df.iloc[:, 2:].values.flatten()
```

```
text_data = [str(i).strip() for i in text_data if
str(i).strip().lower() not in [None, 'none', 'nan', '', 'kg', 'cm',
'animal', 'low', 'upma', 'sister', '']]
text_data = [i.lower().replace('kg',"") for i in text_data]
text_data = [i.lower().replace('cm',"") for i in text_data]
text_data = [i.strip() for i in text_data if len(i.strip()) > 1]
# text_data = [i for i in text_data if i.isalpha()]


# Join the filtered data into a single text corpus
text_corpus = " ".join(text_data)

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(text_corpus)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```


```
import numpy as np
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA

# KMeans Class Definition (Already Provided)
```

```
class KMeans:
    def __init__(self, k=3, max_iters=100, tol=1e-4):
        self.k = k  # Number of clusters
        self.max_iters = max_iters  # Maximum number of iterations
        self.tol = tol  # Tolerance for convergence
        self.centroids = None  # Centroids of the clusters
        self.labels = None  # Labels for each point (cluster
assignments)

    def fit(self, X):
        np.random.seed(42)  # Set seed for reproducibility
        random_indices = np.random.choice(X.shape[0], self.k,
replace=False)  # Randomly select k points
        self.centroids = X[random_indices]  # Set initial centroids

        for _ in range(self.max_iters):
            labels = self._assign_labels(X)  # Step 2: Assign points
to nearest centroid
            new_centroids = self._compute_centroids(X, labels)  # Step
3: Update centroids

            # Step 4: Check for convergence (if centroids do not
change much)
            centroid_shift = np.sum((new_centroids - self.centroids)
** 2)
            if centroid_shift < self.tol:
                print(f"Converged after {_} iterations.")
                break

            self.centroids = new_centroids  # Update centroids for
next iteration

        self.labels = labels  # Final cluster assignments

    def _assign_labels(self, X):
        """Assign each point to the nearest centroid."""
        distances = np.linalg.norm(X[:, np.newaxis] - self.centroids,
axis=2)
        return np.argmin(distances, axis=1)

    def _compute_centroids(self, X, labels):
        """Compute new centroids as the mean of points in each
cluster."""
        centroids = np.zeros((self.k, X.shape[1]))
        for i in range(self.k):
            centroids[i] = X[labels == i].mean(axis=0)
        return centroids

    def predict(self, X):
        """Predict the cluster labels for new data points."""
```

```
        return self._assign_labels(X)

    def get_centroids(self):
        """Return the final centroids."""
        return self.centroids

import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from gensim.models import Word2Vec
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans  # Importing KMeans

# Download necessary resources
nltk.download('stopwords')

# Assuming df is already defined. Example: df =
pd.read_csv("your_data.csv")

# Text Preprocessing
text_data = df.iloc[:, 2:].fillna("").astype(str).apply(lambda x: "
".join(x), axis=1).tolist()

custom_stopwords = {"nan", "years", "kg", "sister", "nonvegetarian",
"upma", "cm"}

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = text.split()  # Split text into words based on whitespace
    all_stopwords =
set(stopwords.words('english')).union(custom_stopwords)
    words = [word for word in words if word not in all_stopwords and
len(word) > 1]
    return words

# Filter out empty texts
preprocessed_text = [preprocess_text(text) for text in text_data if
text.strip()]

# Train Word2Vec Model
model = Word2Vec(sentences=preprocessed_text, vector_size=100,
window=5, min_count=1, workers=4)

# Generate Document Vectors
```

```
document_vectors = []
cleaned_corpus = []

for document in preprocessed_text:
    word_vectors = [model.wv[word] for word in document if word in
model.wv]
    if word_vectors:
        document_vectors.append(np.mean(word_vectors, axis=0))
        cleaned_corpus.append(document)
    else:
        document_vectors.append(np.zeros(model.vector_size))
        cleaned_corpus.append([])

X = np.array(document_vectors)

# Apply KMeans Clustering
k = 3
kmeans = KMeans(n_clusters=k, max_iter=100, tol=1e-4)  # Corrected to
use n_clusters instead of k
kmeans.fit(X)

# Function to get top words per cluster
def get_top_words(centroid, model, top_n=10):
    words = list(model.wv.index_to_key)
    word_vectors = np.array([model.wv[word] for word in words])
    similarities = cosine_similarity([centroid], word_vectors)[0]
    top_indices = similarities.argsort()[-top_n:][::-1]
    return [words[i] for i in top_indices]

# Generate Word Clouds for Each Cluster
fig, axes = plt.subplots(1, k, figsize=(15, 5))

for i in range(k):
    cluster_indices = np.where(kmeans.labels_ == i)[0]  # Corrected to
use labels_ instead of labels

    cluster_words = []
    for index in cluster_indices:
        cluster_words.extend(cleaned_corpus[index])

    word_freq = " ".join(cluster_words)

    wordcloud = WordCloud(width=400, height=400,
background_color="white").generate(word_freq)

    axes[i].imshow(wordcloud, interpolation="bilinear")
    axes[i].axis("off")
    axes[i].set_title(f"Cluster {i+1}")

plt.show()
```

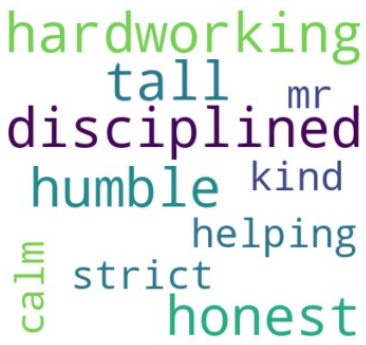
```
# PCA for Cluster Visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.figure(figsize=(8, 6))
for i in range(k):
    plt.scatter(X_pca[kmeans.labels_ == i, 0], X_pca[kmeans.labels_ ==
i, 1], label=f'Cluster {i+1}')

plt.legend()
plt.title("KMeans Cluster Visualization (PCA)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

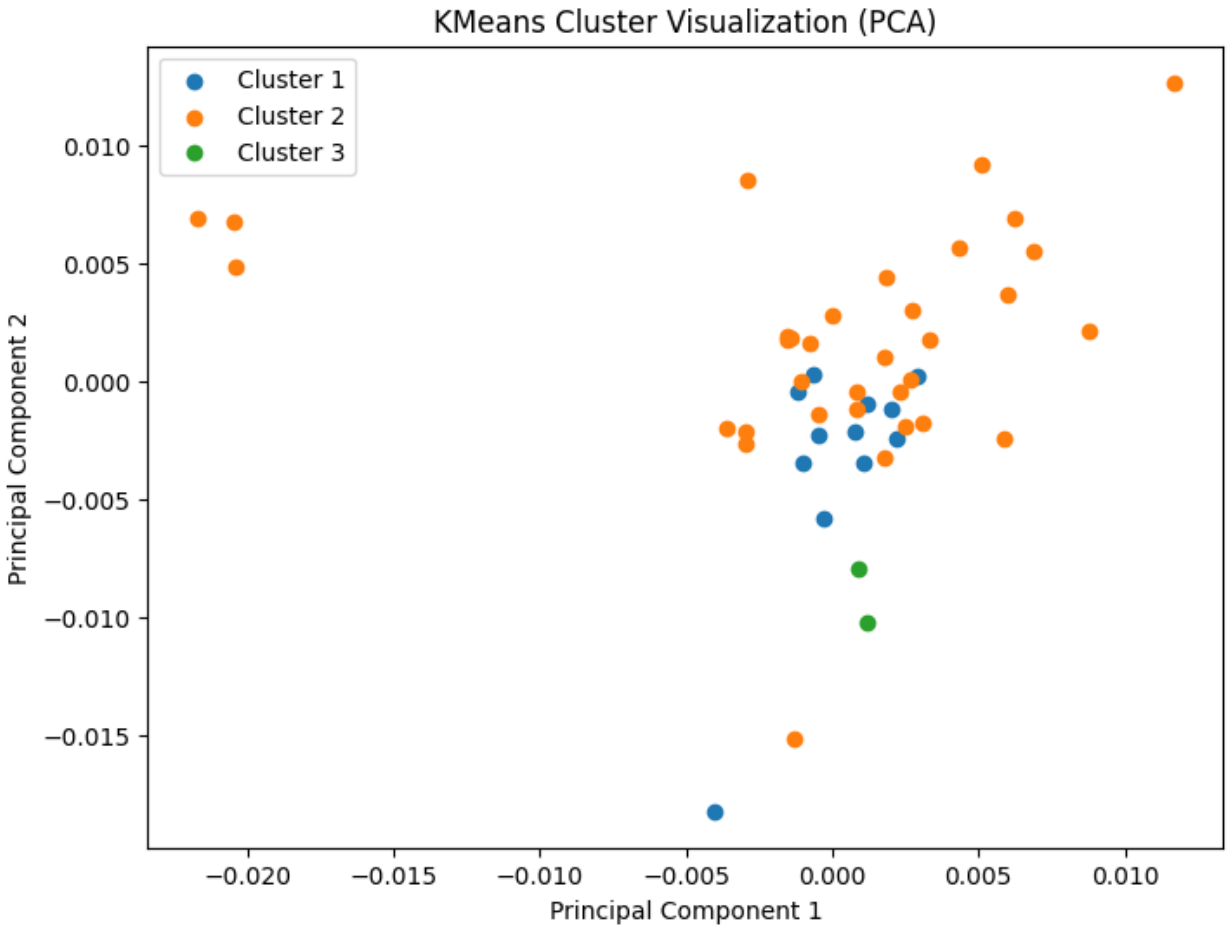
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Cluster 1

Cluster 2

Cluster 3






```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import jaccard_score
from itertools import combinations


# Set of custom words to exclude
excluded_words = {
    "nan", "hobbies", "feet", "kg", "married", "vijayawada", "ap",
"job", "new", "age",
    "non-vegetarian", "glasses", "govt", "bindi", "thing", "song",
"sister", "eye", "cm",
    "those", "upma", "type", "who", "makes", "the", "to", "in", "and",
"is", "not", "of",
    "telugu", "emp", "years"
}

# Clean and structure the data as a dictionary {Person: Set of
Attributes}
df_clean =
```

```
df.drop(columns=["Sno"]).set_index("Person").stack().reset_index(drop=
True, level=1)
df_clean = df_clean.groupby("Person").apply(lambda x: set(val for val
in x if val not in excluded_words and val==val))  # Remove nan and
stopwords


# Function to calculate Jaccard distance
def calculate_jaccard_distance(set1, set2):
    intersection_count = len(set1.intersection(set2))
    union_count = len(set1.union(set2))
    return 1 - (intersection_count / union_count) if union_count != 0
else 1  # Handle empty sets

# Choose two people for comparison
person_a = "Rohit"
person_b = "C"

set_a = df_clean.get(person_a, set())
set_b = df_clean.get(person_b, set())

# Calculate the Jaccard distance
jaccard_result = calculate_jaccard_distance(set_a, set_b)

# Display the results
print(f"Attributes of {person_a}: {set_a}")
print(f"Attributes of {person_b}: {set_b}")
print(f"Jaccard Distance between {person_a} and {person_b}:
{jaccard_result:.4f}")


# Calculate pairwise Jaccard distances for all persons
distance_list = []
for (person_a, set_a), (person_b, set_b) in
combinations(df_clean.items(), 2):
    jaccard_result = calculate_jaccard_distance(set_a, set_b)
    distance_list.append(jaccard_result)

# Define Jaccard distance thresholds (from 0 to 1 in steps of 0.05)
threshold_range = np.arange(0, 1.05, 0.05)

# Plot the distribution of Jaccard distances
plt.hist(distance_list, bins=20, edgecolor='black')  # Adjusted bins
for better representation
plt.xlabel('Jaccard Distance')
plt.ylabel('Frequency of Pairs')
plt.title('Distribution of Jaccard Distances Between Pairs of People')
plt.grid(True)
plt.show()
```

```
Attributes of Rohit: set()
Attributes of C: {'selfless', 'Good Memory', 'reasonable', 'Helping',
'relational', 'Smart', 'simple', 'agrees to everything', 'Supportive',
'Understanding', 'no emotions', 'Funny', 'Rich', 'Compromising'}
Jaccard Distance between Rohit and C: 1.0000
```