```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.gridspec as gridspec
import seaborn as sns
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import DBSCAN
import plotly.express as px
import umap.plot
from sklearn.manifold import TSNE

###************************ Function Hub ************************###

############################## Subpart 1 ##############################

def cat_plot(data, cat_features):
    # Plot barplots
    fig = plt.figure(figsize=(10,20))
    gs = gridspec.GridSpec(5,2)
    ax = {}

    for ftr, i in zip(cat_features, range(len(cat_features))):
        ax[i] = fig.add_subplot(gs[i])
        ax[i] = sns.countplot(data, x=ftr)
        ax[i].set_xticklabels(ax[i].get_xticklabels()) #, rotation=40, ha="right")
        ax[i].set_xlabel(cat_features[i])
    plt.tight_layout()
    plt.show()


def num_plot(X):
    # Plot histograms
    X.hist(figsize=(20, 15))
    plt.suptitle("Histograms of the Attributes", fontsize=20)
    plt.show()

    # Plot boxplots
    X.boxplot(figsize=(6, 10))
    plt.title("Boxplot of the Attributes")
    plt.xticks(rotation=45)
    plt.show()

    # Plot pairwise scatterplots
    sns.pairplot(X, corner=True)
    plt.suptitle("Pairwise Scatterplots", fontsize=20)
    plt.show()

```

```python
    # Heatmap of cross correlations
    sns.heatmap(X.corr(numeric_only=False))
    plt.title("Heatmap")
    plt.show()


############################## Subpart 2 ##############################

def plot_outliers(outliers):
    fig = plt.figure(figsize=(12, 6))
    gs = gridspec.GridSpec(1,2)
    ax = {}

    ax[0] = fig.add_subplot(gs[0])
    ax[0] = sns.scatterplot(x=range(0,440), y=outliers)
    ax[0].set(xlabel ="Item", ylabel = "Soft-Min score", title ='Soft-Min scores (gamma = 1)')
    ax[0].set_xticks(range(0,440,40))

    ax[1] = fig.add_subplot(gs[1])
    ax[1].set(ylabel = "Soft-Min score", title ='Boxplot of Soft-Min scores (gamma = 1)')
    ax[1].set_xticks([])
    ax[1] = plt.boxplot(outliers)
    min, max = [item.get_ydata()[1] for item in ax[1]['whiskers']]

    plt.show()
    return min, max


############################## Subpart 3 ##############################

def distance_plots(mean_distance, var_distance, gamma_range, outliers):
    fig, axs = plt.subplots(2, 2, figsize=(14, 10))
    axs = axs.flatten()

    for i in range(len(mean_distance)):
        if i in outliers:
            axs[1].plot(gamma_range, mean_distance[i], linewidth=2, c="blue")
        else:
            axs[1].plot(gamma_range, mean_distance[i], linewidth=0.3)

    for i in range(len(var_distance)):
        if i in outliers:
            axs[0].plot(gamma_range, var_distance[i], linewidth=2, c="blue")
        else:
            axs[0].plot(gamma_range, var_distance[i], linewidth=0.3)

    sns.scatterplot(x=gamma_range, y=np.var(mean_distance, axis=0), ax=axs[3])
```

```

    sns.scatterplot(x=gamma_range, y=np.mean(var_distance, axis=0), ax=axs[2])

    axs[0].set_ylabel("MEAN of the relevance distances between components")
    axs[1].set_ylabel("VARIANCE of the relevance distances between components")
    #axs[2].set_ylabel("VARIANCE of the mean relevance distances between components")
    axs[2].set_ylabel("Average MEAN of the relevance distances between components")
    axs[3].set_ylabel("Average VARIANCE of the relevance distances between components")
    for ax in axs:
        ax.set_xlabel("Gamma")

    plt.show()


def attribution_stat_plots(statistic, gamma_range, num_features, outliers, type):
    fig = plt.figure(figsize=(14, 14))
    gs = gridspec.GridSpec(3, 2)
    ax = {}
    for f_ind, feature in enumerate(num_features):
        ax[f_ind] = fig.add_subplot(gs[f_ind])
        ax[f_ind].set(xlabel="Gamma", ylabel="Bootstrap sample " + type + " relevance", title=feature)

        for item in range(statistic.shape[0]):
            if item in outliers:
                ax[f_ind] = plt.plot(gamma_range, statistic[item,:, f_ind], linewidth=2, c="blue")
            else:
                ax[f_ind] = plt.plot(gamma_range, statistic[item,:, f_ind], linewidth=0.3)

    plt.show()


def attribution_variance_means(mean, variance, gamma_range, num_features, outliers, inliers):
    fig = plt.figure(figsize=(18, 18))  # (28,28)
    gs = gridspec.GridSpec(2, 2)
    ax = {}

    ax[0] = fig.add_subplot(gs[0])
    ax[0].set(xlabel="Gamma", ylabel="Average bootstrap sample MEAN for INLIERS")
    # alternative: "Variance of bootstrap sample mean for inliers"
    for f_ind, feature in enumerate(num_features):
        ax[0] = sns.scatterplot(x=gamma_range, y=np.var(mean[:,:,
```

```python
f_ind][inliers], axis=0), legend=num_features)
    ax[0].legend(labels=num_features)

    ax[1] = fig.add_subplot(gs[1])
    ax[1].set(xlabel="Gamma", ylabel="Average bootstrap sample MEAN for OUTLIERS")
    # alternative: "Variance of bootstrap sample mean for inliers"
    for f_ind, feature in enumerate(num_features):
        ax[1] = sns.scatterplot(x=gamma_range, y=np.var(mean[:,:, f_ind][outliers], axis=0))
    ax[1].legend(labels=num_features)

    ax[2] = fig.add_subplot(gs[2])
    ax[2].set(xlabel="Gamma", ylabel="Average bootstrap sample VARIANCE for INLIERS")
    for f_ind, feature in enumerate(num_features):
        ax[2] = sns.scatterplot(x=gamma_range, y=np.mean(variance[:,:,f_ind][inliers], axis=0), legend=num_features)
    ax[2].legend(labels=num_features)

    ax[3] = fig.add_subplot(gs[3])
    ax[3].set(xlabel="Gamma", ylabel="Average bootstrap sample VARIANCE for OUTLIERS")
    for f_ind, feature in enumerate(num_features):
        ax[3] = sns.scatterplot(x=gamma_range, y=np.mean(variance[:,:,f_ind][outliers], axis=0))
    ax[3].legend(labels=num_features)

    plt.show()


def attribution_boxplots(statistic, stat_name):
    fig = plt.figure(figsize=(14, 14))
    gs = gridspec.GridSpec(2, 2)
    ax = {}

    ax[0] = fig.add_subplot(gs[0])
    ax[0].set(ylabel= "Relevance " + stat_name + " over the Bootstrap samples for OUTLIERS", title="Boxplot, gamma = 1")
    ax[0] = statistic.boxplot()
    ax[0].tick_params(axis='x', rotation=45)

    ax[1] = fig.add_subplot(gs[1])
    ax[1].set(title="Boxplot without outlier points")
    ax[1].tick_params(axis='x', rotation=45)
    ax[1] = sns.boxplot(data=statistic, showfliers=False)
    plt.show()


############################## Subpart 4 ##############################

def silhouette_analysis(min_k, max_k, X, Umap = False):
    """ Adapdet from:
```

```
    https://scikit-learn.org/stable/auto_examples/cluster/
    plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py
    mapper_type="UMAP" or "t-SNE"""

    if Umap:
        mapper = umap.UMAP(n_neighbors=4, min_dist=1, n_components=2, metric='euclidean', random_state=42)
    else:
        mapper = TSNE(n_components=2, learning_rate='auto', init='pca', perplexity=30)
    X_embedded = mapper.fit_transform(X)

    range_n_clusters = np.arange(min_k, max_k+1)

    for n_clusters in range_n_clusters:
        # Create a subplot with 1 row and 2 columns
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.set_size_inches(18, 7)

        # The 1st subplot is the silhouette plot
        # The silhouette coefficient can range from -1, 1 but in this example all
        # lie within [-0.1, 1]
        ax1.set_xlim([-0.1, 1])
        # The (n_clusters+1)*10 is for inserting blank space between silhouette
        # plots of individual clusters, to demarcate them clearly.
        ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

        # Initialize the clusterer with n_clusters value and a random generator
        # seed of 0 for reproducibility.

        clusterer = KMeans(n_clusters=n_clusters, init='k-means++', n_init = 100, random_state=42)
        cluster_labels = clusterer.fit_predict(X)

        # The silhouette_score gives the average value for all the samples.
        # This gives a perspective into the density and separation of the formed
        # clusters
        silhouette_avg = silhouette_score(X, cluster_labels)
        print("For n_clusters =", n_clusters,
              "The average silhouette_score is :", silhouette_avg)

        # Compute the silhouette scores for each sample
        sample_silhouette_values = silhouette_samples(X, cluster_labels)

        y_lower = 10
        for i in range(n_clusters):
```

```
            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them
            ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]

            ith_cluster_silhouette_values.sort()

            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i

            color = cm.nipy_spectral(float(i) / n_clusters)
            ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values,
                              facecolor=color, edgecolor=color, alpha=0.7,
            )

            # Label the silhouette plots with their cluster numbers at the middle
            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

            # Compute the new y_lower for next plot
            y_lower = y_upper + 10  # 10 for the 0 samples

            ax1.set_title("The silhouette plot for the various clusters.")
            ax1.set_xlabel("The silhouette coefficient values")
            ax1.set_ylabel("Cluster label")

            # The vertical line for average silhouette score of all the values
            ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

            ax1.set_yticks([])  # Clear the yaxis labels / ticks
            ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

            # 2nd Plot showing the actual clusters formed on within a 2D embedding of choice
            colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
            ax2.scatter(
                X_embedded[:,1], X_embedded[:,0], marker=".", s=30, lw=0, alpha=0.7, c=colors, edgecolor="k"
            )
            ax2.set_title("Data Embedding")
            ax2.set_xlabel("Embedded dimension 2")
            ax2.set_ylabel("Embedded dimension 1")

            if Umap:
                # Labeling the clusters
                centers = clusterer.cluster_centers_
```

```
                # U-Map transformation of the cluster centers
                centers_mapped = mapper.transform(centers)
                # Draw white circles at cluster centers
                ax2.scatter(centers_mapped[:,1], centers_mapped[:,0], marker="o", c="white", alpha=1, s=200,
                            edgecolor="k",
                )

                for i, c in enumerate(centers_mapped):
                    ax2.scatter(c[1], c[0], marker="$%d$" % i, alpha=1, s=50, edgecolor="k")

            plt.suptitle(
                "Silhouette analysis for KMeans clustering on sample data with n_clusters = %d"
                % n_clusters, fontsize=14, fontweight="bold",
            )

    plt.show()


def visualise_kmeans(X, clusterer):
    # Compute UMAP for the visualisation
    umapper = umap.UMAP(n_neighbors=4, min_dist=1, n_components=2, metric='euclidean', random_state=42)
    X_embedded_umap = umapper.fit_transform(X)
    # Compute t-SNE for the visualisation
    mapper = TSNE(n_components=2, learning_rate='auto', init='pca', perplexity=30)
    X_embedded = mapper.fit_transform(X)

    fig = plt.figure(figsize=(18, 9))
    gs = gridspec.GridSpec(1, 2)
    ax = {}

    # UMAP
    ax[0] = fig.add_subplot(gs[0])
    ax[0] = sns.scatterplot(x=X_embedded_umap[:, 1], y=X_embedded_umap[:, 0], hue=clusterer.labels_, palette="tab10")
    ax[0].set_xlabel("UMAP dimension 2")
    ax[0].set_ylabel("UMAP dimension 1")
    ax[0].set_title('UMAP')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # U-Map transformation of the cluster centers
    centers_mapped = umapper.transform(centers)
    # Draw white circles at cluster centers
    ax[0].scatter(centers_mapped[:, 1], centers_mapped[:, 0], marker="o", c="white", s=200, edgecolor="k")
    for i, c in enumerate(centers_mapped):
        ax[0].scatter(c[1], c[0], marker="$%d$" % i, c="white", s=50, edgecolor="k")
```

```python

    # t-SNE
    ax[1] = fig.add_subplot(gs[1])
    ax[1] = sns.scatterplot(x=X_embedded[:, 1], y=X_embedded[:, 0], hue=clusterer.labels_, palette="tab10")
    ax[1].set_xlabel("t-SNE dimension 2")
    ax[1].set_ylabel("t-SNE dimension 1")
    ax[1].set_title('t_SNE')

    plt.show()


def clusters_stats(df, clustering):
    def means(df, clustering):
        plots_per_row = 2
        plots_per_column = math.ceil(len(set(clustering.labels_)) / plots_per_row)
        fig = plt.figure(figsize=(14 * plots_per_row, 4 * plots_per_column))  # (28,28)
        gs = gridspec.GridSpec(plots_per_column, plots_per_row)
        ax = {}

        for cluster in set(clustering.labels_):
            filter = clustering.labels_ == cluster

            n = sum(clustering.labels_ == cluster)
            mean = df[filter].mean()
            sd = df[filter].std()

            ax[cluster] = fig.add_subplot(gs[cluster])
            ax[cluster] = plt.errorbar(df.columns.values.tolist(), mean, sd, marker='o', linestyle='None')
            plt.xlabel('Attribute')
            plt.ylabel('Mean +/- SD')
            plt.title('Cluster ' + str(cluster) + ", size: " + str(n))
            plt.xticks(rotation=45)

        fig.suptitle("Statistics of individual features for the clusters: Mean and Standard Deviation", fontsize=20)
        plt.show()

    def boxplots(df, clustering):
        K = len(set(clustering.labels_))
        fig, ax = plt.subplots(nrows=1, ncols=6, sharex=True, sharey=True, figsize=(28, K))
        cluster_ind = []

        for i, feature in enumerate(df.columns):
            sns.boxplot(ax=ax[i], data=df, x=feature, y=clustering.labels_, orient="h", palette="tab10")
            cluster_ind.append("Cluster " + str(i))
        plt.yticks(range(0, K), cluster_ind)
```

```
        fig.suptitle("Statistics of individual features for the clusters: Boxplots", fontsize=20)

        plt.show()

    means(df, clustering)
    boxplots(df, clustering)

```