B565-Data Mining
Homework #5

Due on Friday, March 12, 2023, 08:00 p.m.

Instructor: Dr. H. Kurban, Head TA: Md R. Kabir

Jash Shah

March 12, 2023

PCA Algorithm

This part is provided to help you implement Principal Component Analysis.

1: **ALGORITHM** `PCA`
2: **INPUT** Δ: $n \times m$ data matrix of rank r, d: the number of new dimensions where $d \leq r$.
3: **OUTPUT** $\hat{\Delta}$: d dimensional representation of Δ.
4: %% mean centering - $\tilde{\Delta}$ is the centered data matrix.
5: %% I denotes $n \times n$ identity matrix, $e = (1, ..., 1) \in \Re^n$.
6: $\tilde{\Delta} \leftarrow \left(I - \frac{ee^t}{n}\right)\Delta$.
7: %% Compute the SVD of $\tilde{\Delta}$. $\sigma_1 \geq ... \geq \sigma_r > 0$ are the strictly positive singular values of $\tilde{\Delta}$.
8: %% $u_1, ..., u_r$ and $v_1, ..., v_r$ are the corresponding left and right singular vectors respectively.
9: $\tilde{\Delta} \leftarrow \sum_{i=1}^{r} \sigma_i u_i v_i^t$
10: $\hat{\Delta}\left[\sigma_1 u_1 | ... | \sigma_d u_d\right] = \begin{bmatrix} \hat{\delta}_1^t \\ \vdots \\ \hat{\delta}_n^t \end{bmatrix}$

Problem 1

Implement Principal Component Analysis algorithm (PCA) and run your program over RNA-Seq (HiSeq) PANCAN data set (the data contains the gene expressions of patients having 5 different types of tumor: BRCA, KIRC, COAD, LUAD and PRAD which we will be using as our data labels) to answer the following questions. **Data Preparation:** Similar to any other data mining problem, before feeding your data to the clustering algorithms, you will have to perform appropriate data cleaning, feature engineering, and feature selection on this dataset **[25 pt.]**

1. Perform PCA over the RNA-Seq data set and make a scatter plot of PC1 and PC2 (the first two principal components). Are PC1 and PC2 linearly correlated? How much of the variance can be explained by PC1 and PC2?

   - **Dataset**
   - The dataset includes expression values for 20,531 genes across 800 cancer samples, which are classified into 5 different cancer types. The data was obtained from The Cancer Genome Atlas (TCGA) project, which is a collaborative effort to comprehensively characterize the genomic and molecular landscape of cancer.
   - There is a csv called labels.csv indicating the type of cancer for each 800 Data Points
   - Usage of Dataset
   - This dataset can be used for various analyses related to cancer biology, such as identifying biomarkers for different cancer types, studying the molecular mechanisms underlying cancer development and progression, and developing diagnostic and prognostic tools for cancer patients.

R or Python script

```
# Sample R Script With Highlighting
```

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

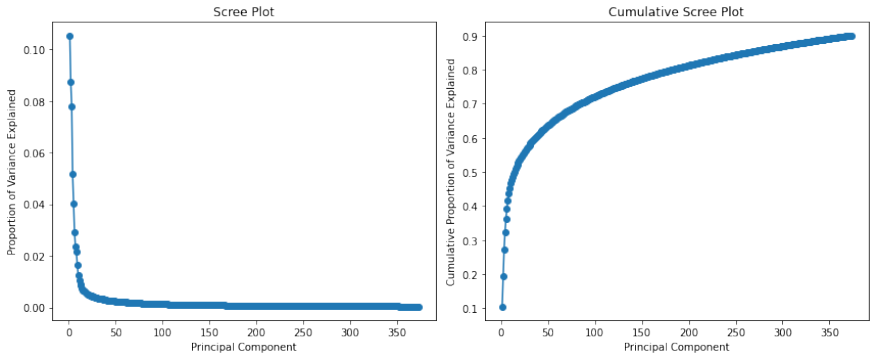
Problem 1 continued on next page...

```
import seaborn as sns

df_pancan=pd.read_csv('pancan.csv')

import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler

def covariance(input_dataframe):
    '''
    This function takse input as a standardized dataframe
    '''
    input_dataframe_mean =
    input_dataframe.swifter.apply(np.mean, axis=0)
    input_dataframe_centered= input_dataframe-
    input_dataframe_mean
    with tqdm(total=input_dataframe.shape[1], desc="Calculating
    Covariance Matrix") as pbar:
        cov_matrix=np.cov(input_dataframe.T)
        pbar.update()
    return cov_matrix,input_dataframe_centered

def principal_component_analysis(input_dataframe):
    '''
    This function takes input_dataframe,stadndardizes it and
    number of components as the number of components required by
    PC
    '''
    scaler = StandardScaler()
    input_dataframe_scaled
    =pd.DataFrame(scaler.fit_transform(input_dataframe))
    #Calling the covriance function
    covariance_matrix,input_dataframe_centered=covariance(input_d
    ataframe_scaled)
    #Calculates Covariance Matirx
    eigen_values,eigen_vectors=np.linalg.eig(covariance_matrix)
    #Calculates Eigen Values and Eigen Vectors
    sorted_indices=np.argsort(eigen_values)
    #Sort the elements in descending order
    sorted_indices=sorted_indices[::-1]


    explained_variances = eigen_values / np.sum(eigen_values)

    variance_explained_ratios =
    pd.DataFrame(explained_variances[sorted_indices], columns=
    ["variance_explained_ratio"])
    variance_explained_ratios["cumulative_variance_explained_rati
    o"] =
    variance_explained_ratios["variance_explained_ratio"].cumsum(
    )

    #Find the number of components that explain 90% of variance
```

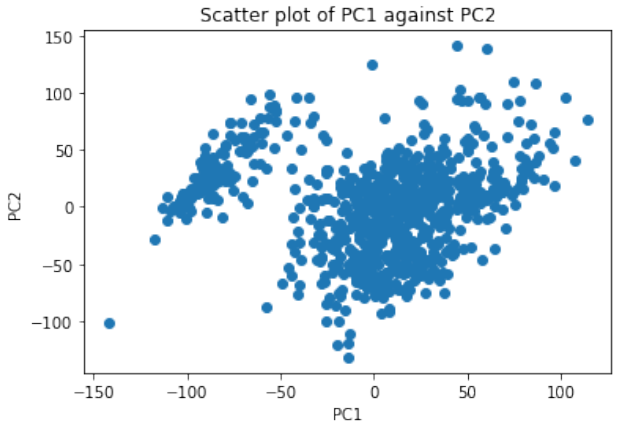
```
    number_of_components =
    variance_explained_ratios["cumulative_variance_explained_rati
o"][variance_explained_ratios["cumulative_variance_explained_rati
o"] <= 0.90].count() + 1

    print("Number of Principal components explain 90% of
    variance are {}".format(number_of_components))




    #Taking Top Eigen Values and Top Eigen Vectors
    top_eigen_values_indices=sorted_indices[:number_of_components
    ]
    top_eigen_vectors=eigen_vectors[:,top_eigen_values_indices]

     #Variance Calculations Plot
    explained_variances = eigen_values/np.sum(eigen_values)
    variance_explained =
    pd.DataFrame(eigen_values[top_eigen_values_indices] /
    sum(eigen_values))
    variance_explained['PC_Feature']=top_eigen_values_indices

    variance_explained_plot=pd.Series(eigen_values[top_eigen_valu
    es_indices] / sum(eigen_values))


    #Cumulative Variance Plot
    cumulative_variance_explained =
    np.cumsum(variance_explained_plot)
    cumulative_variance_explained_plot =
    pd.Series(cumulative_variance_explained)



    #Projecting Principal Components
    principal_components=input_dataframe_centered.dot(top_eigen_v
    ectors)
    principal_components.columns=[f'PC{i+1}' for i in
    range(number_of_components)]




    #Calculate the loadings
    loadings =
    pd.DataFrame(top_eigen_vectors,index=input_dataframe.columns)

    df_principal_components=pd.DataFrame(principal_components,
    columns=[f'PC{i+1}' for i in range(number_of_components)])
    #PLotting the graph
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
    ax[0].plot(np.arange(1,
```

```
    number_of_components+1),variance_explained_plot, 'o-')
    ax[0].set_xlabel('Principal Component')
    ax[0].set_ylabel('Proportion of Variance Explained')
    ax[0].set_title('Scree Plot')


    ax[1].plot(np.arange(1,
    number_of_components+1),cumulative_variance_explained_plot,
    'o-')

    ax[1].set_xlabel('Principal Component')
    ax[1].set_ylabel('Cumulative Proportion of Variance
    Explained')
    ax[1].set_title('Cumulative Scree Plot')
    plt.tight_layout()
    plt.show()

    #Correlation between PC1 and PC2

    plt.scatter(principal_components['PC1'],
    principal_components['PC2'])
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title('Scatter plot of PC1 against PC2')
    plt.show()

    principal_components_temp=principal_components[['PC1','PC2']]
    corr_matrix = principal_components_temp.corr()
    print('Correlation matrix:')
    print(corr_matrix)

    total_variance_explained=cumulative_variance_explained_plot[1
    ]
    print("The total variance explained by first two PC's is
    {}".format(total_variance_explained))

    return
    variance_explained,loadings,principal_components,cumulative_v
    ariance_explained

    '''
    Calling the function
    variance_explained,loadings,principal_components,cumulative_v
    ariance_explained=principal_component_analysis(df_pancan)
    '''
```

Plot/s



(1)



(2)

```
Correlation matrix:
              PC1           PC2
PC1  1.000000e+00 -1.923637e-15
PC2 -1.923637e-15  1.000000e+00
The total variance explained by first two PC's is (0.1929401381992425+0j)
```

(3)

Discussion of Experiments

- Number of Principal components explain 90 percent of variance are 373
- The above code uses Eigen Values and Eigen Vectors decomposition and Variance is explained by Eigen Values.
- **Scatter Plot Discussion between PC1,PC2**
- The Scatter plot shows no discernible pattern or trend in the relationship between the two PC's being plotted. The points appear to be randomly distributed across the plot, with no clear relationship between the x and y axes.
- Specifically, the points do not form a linear or curved shape, and there should be no apparent clustering or grouping of points in any particular region of the plot.
- From the image of Correlation Coefficent's we can see the value is extremely close to zero i.e -1.923637e-15

- **Hence we conclude No, PC1 (principal component 1) and PC2 (principal component 2) of PCA (principal component analysis) are not correlated**
- **The total variance explained by first two PC's is (0.1929401381992425+0j) accounting to 19.2 percent**

2. There are three methods to pick the set of principle components: (1) In the plot where the curve bends; (2) Add the percentage variance until total 75% is reached $(70-90\%)$ (3) Use the components whose variance is at least one. Show the components selected in the RNA-Seq data set data if each of these is used.

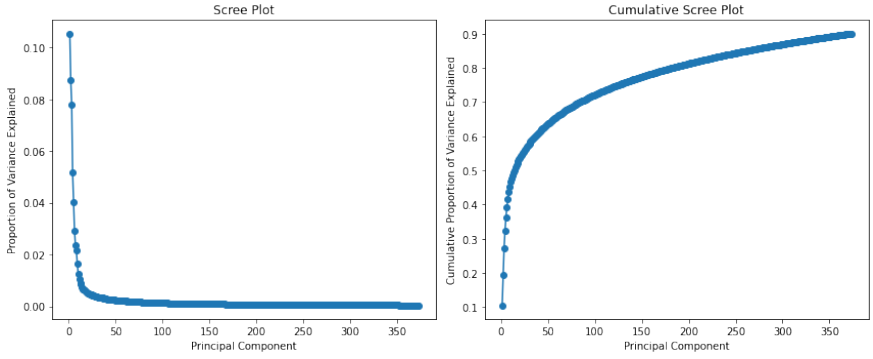
R or Python script

```
# Sample R Script With Highlighting
```

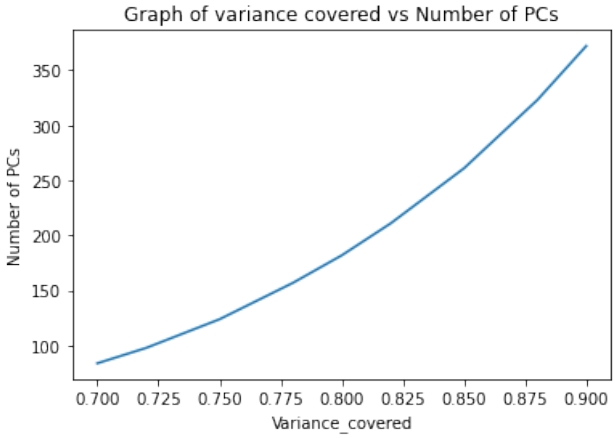
```
# Sample Python Script With Highlighting
#Method 2 When cumulative Variance Reaches 75 percent
variance_explained_method=variance_explained.copy()
variance_explained_method['cumulative_variance']=variance_explain
ed[0].cumsum()
variance_explained_method['cumulative_variance']=variance_explain
ed_method['cumulative_variance'].swifter.apply(lambda x:x.real)

def variance_method(cumulative_variance_dataframe,threshold):
    '''
    Input requires a dataframe having cumulative variance and a
    threshold
    Threshold values must be between 0.70-0.95
    '''
    return
    threshold,len(variance_explained_method[variance_explained_me
    thod['cumulative_variance']<=threshold])

threshold=[0.70,0.72,0.75,0.78,0.80,0.82,0.85,0.88,0.90]
pc_dict={}
for i in threshold:
    threshold,number_of_pcs=variance_method(variance_explained_me
    thod,i)
    pc_dict[threshold]=number_of_pcs
pc_dict
'''
{0.7: 84,
 0.72: 98,
 0.75: 124,
 0.78: 157,
 0.8: 182,
 0.82: 211,
 0.85: 261,
 0.88: 323,
 0.9: 372}
 '''

 #Method 3 Variance is at least 1
```

```
 from sklearn.preprocessing import StandardScaler
input_dataframe=df_pancan
scaler = StandardScaler()
input_dataframe_scaled
=pd.DataFrame(scaler.fit_transform(input_dataframe))
covariance_matrix,input_dataframe_centered=covariance(input_dataf
rame_scaled)
eigen_values,eigen_vectors=np.linalg.eig(covariance_matrix)

eigen_values_greater=[x for x in eigen_values if x.real>1]
print("The number of Principal Components having variance
greater than 1 is {}".format(len(eigen_values_greater)))

'''
The number of Principal Components having variance greater than 1 is 800
'''
```

Plot/s



(4)



(5)

Discussion of Experiments

- **Method 1 From the graph**

- From the graphs we can see the Scree Plot bends between 30 to 100 and the cumulative Scree Plot Also suggests the same.
- **So to check where my curve bends the number of Principal Components will be between 30 to 100**
- **Method 2 Taking 75 Percent of variance**
- From the graph of threshold we can check the number of different Principal Components corresponding to the threshold value mentioned.
- The output is given as
  0.7: 84, 0.72: 98, 0.75: 124, 0.78: 157, 0.8: 182, 0.82: 211, 0.85: 261, 0.88: 323, 0.9: 372
- **Method 3 Selecting Number of Components by variance greater than or equal to 1**
- After applying the filter to select those eigen values having greater than 1 we get **800 Principal Components**
- **Discussion about Methods Used**
  - The choice of the best method for selecting the number of principal components depends on the specific dataset and analysis goals.
  - Each method has its own strengths and weaknesses, and the best method for a given problem may vary depending on the data and the research questions.
  - **Method 1** involves visual inspection of the scree plot to identify the elbow or bend in the curve where the rate of decrease in the eigenvalues levels off. This method is easy to use and interpret, but can be subjective and may not work well for datasets with noisy or complex structures.
  - **Method 2** involves selecting a threshold percentage of the total variance explained by the principal components. This method provides a more objective criterion for selecting the number of components and can be useful for datasets with clear patterns of variation, but may not work well for datasets with more subtle or complex patterns.
  - **Method 3** involves retaining only those components whose variance is at least one. This method can be useful for datasets where the variance of the principal components provides a clear criterion for selecting the number of components, but may not work well for datasets with more subtle or complex patterns of variation.
  - Overall, the choice of the best method for selecting the number of principal components should be based on a combination of statistical criteria and domain expertise. It is often a good idea to explore multiple methods and compare the results to identify the best approach for a given problem.

3. Observe & discuss the loadings in PCA? (e.g., how are principal components and original variables related?)

```
loadings_real=loadings.applymap(lambda x:x.real)
'''
The below function shows the most important atttributes in a PC
'''
loadings_list=[]
for col in loadings_real.columns:
    indices=[]
    indices_positive=loadings_real.index[loadings_real[col] >
    0.04].tolist()
    indices_negative=loadings_real.index[loadings_real[col]
```

```
    <-0.04].tolist()
    indices=indices_positive+indices_negative
    loadings_list.append(['PC'+str(col),indices])
max_abs=pd.DataFrame(loadings_real.abs().idxmax())
max_abs_val_counts=pd.DataFrame(max_abs[0].value_counts())
plt.plot(max_abs_val_counts[max_abs_val_counts[0]>=2])
plt.xticks(rotation=90)
plt.show()
```

Discussion of Experiments

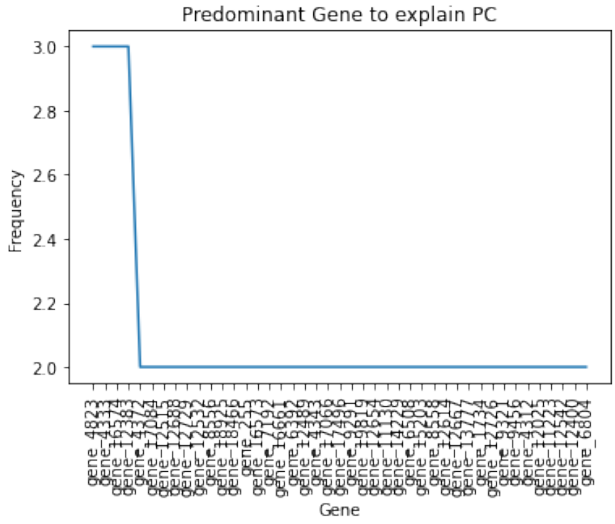
Explanation of Loadings

- Suppose we have a dataset with three variables: x1, x2, and x3. We want to perform PCA to reduce the dimensionality of the data. After performing PCA, we obtain two principal components, PC1 and PC2, which explain 60 percent and 30 percent of the variance in the data, respectively.
- The loadings for PC1 and PC2 can be calculated by multiplying the standardized data matrix by the eigenvectors of the covariance matrix. The resulting loadings matrix will have two rows (one for PC1 and one for PC2) and three columns (one for each original variable).
- For example, the loadings for PC1 might look something like this:
  x1: 0.5
  x2: -0.3
  x3: 0.8
- The loadings show the contribution of each original variable to PC1. We can see that x3 has the highest loading on PC1, followed by x1, indicating that they are strongly associated with this component. On the other hand, x2 has a negative loading on PC1, indicating that it is negatively associated with this component.

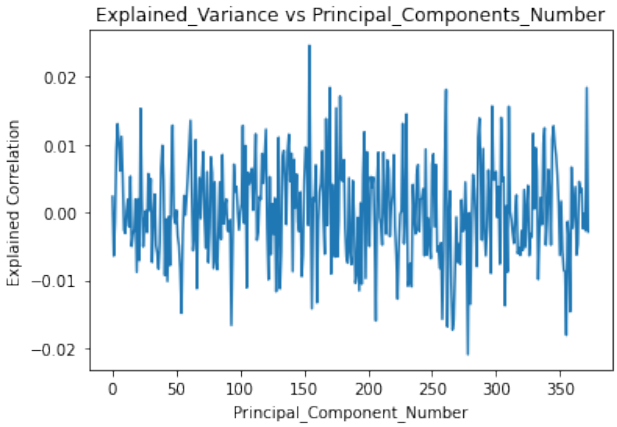
  Explanation of Code

  - Have kept a threshold of 0.04 to filter out those genes which have maximum contribution to the Principal Components
  - In the second part, have taken the genes that contribute the max to a principal components and have plotted some of them occuring more than 2 times,Can be seen in graphs below.
  - In Conclusion,These genes are most important in explaining the Dataset.
  - Also have taken a particular row of Loadings Dataframe and for a particular gene calculated the line plot
  - There is a noisy pattern with spikes in middle.
  - Have also Plotted the Loadings of PC1 vs PC2 and we can observe.
  - The scatter plot of PC1 and PC2 is shown in the above figure. Contribution of every variable to the formation of PCA is ranging from -0.02 to 0.02.
  - The values seems randomly spread out forming a cluster and even the values are 0, showing no contribution. There are variables which are not contributing to the Principal components.

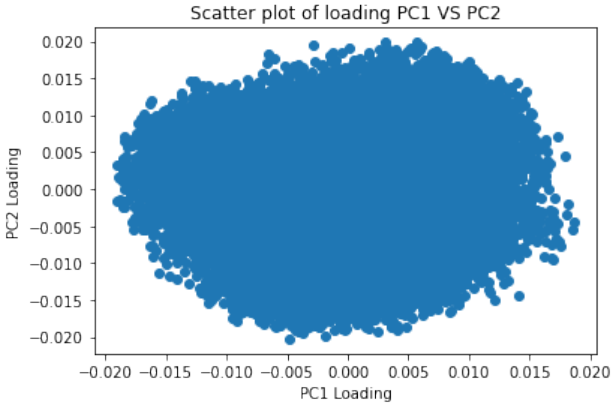
Plot/s



(6)



(7)



(8)

4. Perform dimensionality reduction over the RNA-Seq data set with PCA. Keep 90% of variance after PCA and reduce the RNA-Seq data set and call this data Δ_R. Cluster Δ_R using your k-means program

from previous assignment and report the total error rates for $k = 5$ and 20 runs. Plots are generally a good way to convey complex ideas quickly, i.e., box plots, whisker plots. Discuss your results, i.e.,Did clustering get better after PCA?

R or Python script

```
# Sample R Script With Highlighting
```

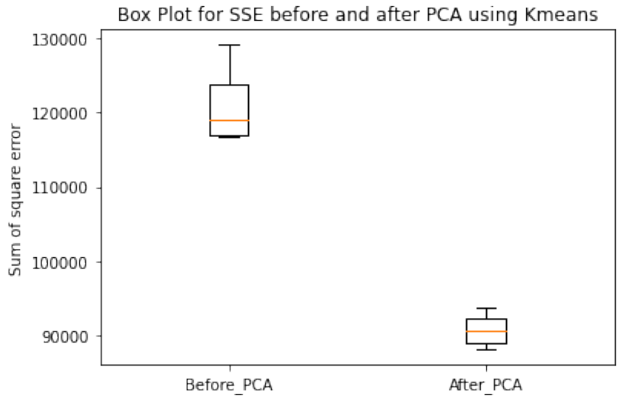
```
#Performing K means Clustering on the above data from previous
#Assignment
#Running K means before PCA on raw Data
import numpy as np
import swifter
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import cdist
import time


def get_random_centroids(input_dataframe,no_of_clusters):
    list_of_centroids = []
    for cluster in range(no_of_clusters):
        random_centroid = input_dataframe.swifter.apply(lambda
        x:float(x.sample()))
        list_of_centroids.append(random_centroid)

    centroid_df=pd.concat(list_of_centroids,axis=1)
    centroid_df.index.name='Cluster_Assigned'
    return centroid_df

def get_labels(input_dataframe,centroid_df):
    euclidean_distances = centroid_df.swifter.apply(lambda x:
    np.sqrt(((input_dataframe - x) ** 2).sum(axis=1)))
    return pd.DataFrame(euclidean_distances.idxmin(axis=1))


def get_new_centroids(df_clustered_label,input_dataframe):

    df_original_label_join=input_dataframe.join(df_clustered_lab
    el)
    df_original_label_join.rename(columns=
    {0:'Cluster_Assigned'},inplace=True)
    new_centroids=df_original_label_join.groupby('Cluster_Assign
    ed').mean()
    return new_centroids.T


def
kmeans_llyod(input_dataframe,no_of_clusters,threshold,no_of_iter
ations):
    start_time=time.time()
    iteration=0
    initial_centroid=get_random_centroids(input_dataframe,no_of_
    clusters)
```

```
    initial_centroid_column_list=initial_centroid.columns.to_lis
    t()

    while True:

        df_cluster_label=get_labels(input_dataframe,initial_cent
        roid)
        df_new_centroids=get_new_centroids(df_cluster_label,inpu
        t_dataframe)
        new_list_of_columns=df_new_centroids.columns.to_list()
        initial_set_columns = set(initial_centroid_column_list)
        new_set_columns = set(new_list_of_columns)
        missing_columns = initial_set_columns - new_set_columns
        for col in missing_columns:
            df_new_centroids[col]=initial_centroid[col]

        from scipy.spatial.distance import euclidean
        scalar_product =
        [euclidean(initial_centroid[col],df_new_centroids[col])
        for col in initial_centroid.columns]

        threshold_calculated=float(sum(scalar_product))/no_of_cl
        usters

        iteration+=1

        if threshold_calculated<threshold:
            print("The input Threshold was
            {}".format(threshold))
            print("The calculated threshold is
            {}".format(threshold_calculated))

        if iteration>no_of_iterations:
            print("Limit for iterations has exceeded")

        if threshold_calculated<threshold or
        iteration>no_of_iterations:
            sum_of_square_error=sum_of_square_error_function(df_

            cluster_label,input_dataframe,df_new_centroids,no_of
            _clusters)
            df_cluster_label_copy=df_cluster_label.copy()
            df_cluster_label_copy.rename(columns=
            {0:'Cluster_Assigned'},inplace=True)
            labels=df_cluster_label_copy['Cluster_Assigned'].to_
            list()
            silheoutte_score=silheoutte_score_Kmeans(input_dataf
            rame,labels)
            chs_score=Calinski_Harbaz_score_Kmeans(input_datafra
            me,labels)
            end_time=time.time()
            return
            df_new_centroids,sum_of_square_error,silheoutte_scor
```

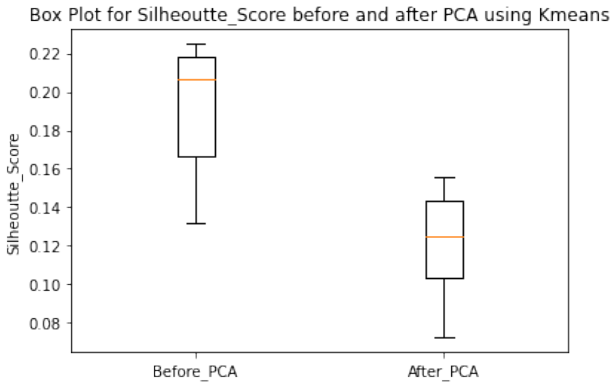
```
            e,chs_score,end_time-start_time
            break
        else:
            initial_centroid= df_new_centroids


def
sum_of_square_error_function(df_cluster_label,input_dataframe,df
_new_centroids,no_of_clusters):
    df_data_label=input_dataframe.join(df_cluster_label)
    df_data_label.rename(columns=
    {0:'Cluster_Assigned'},inplace=True)
    total_error=[]
    for cluster in range(no_of_clusters):
        df_data_label_cluster=df_data_label[df_data_label['Clust
        er_Assigned']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Cluste
        r_Assigned',axis=1)
        centroids=pd.DataFrame(df_new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids
        .T,metric='euclidean')
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

def silheoutte_score_Kmeans(input_dataframe,labels):
    from sklearn.metrics import silhouette_score
    silhouette_avg = silhouette_score(input_dataframe, labels)
    return silhouette_avg

def Calinski_Harbaz_score_Kmeans(input_dataframe,labels):
    from sklearn.metrics import calinski_harabasz_score
    chs=calinski_harabasz_score(input_dataframe,labels)
    return chs

#Running K means after PCA
principal_components_real_part=principal_components.copy()
principal_components_real_part=principal_components_real_part.applymap(lambda x:x.real)
error_values=[]
for no_of_experiments in range(1,21):
    final_centroids,sum_of_squared_error,sil_score,chs_score,run
    _time=kmeans_llyod(principal_components_real_part,5,10,100)
    error_values.append([5,no_of_experiments,sum_of_squared_erro
    r,sil_score,chs_score,run_time])
error_values_df_PCA= pd.DataFrame(error_values,columns=
['No_of_Clusters', 'Iteration
Number','Sum_of_squared_Errors','Silheoutte_Score','Chs_score','
run_time'])
#Plotting Graphs
combined_df =
pd.concat([error_values_before,error_values_df_PCA],
keys=['Before_PCA','After_PCA'])
plt.boxplot([combined_df.loc['Before_PCA']
['Sum_of_squared_Errors'], combined_df.loc['After_PCA']
```

```
['Sum_of_squared_Errors']])
plt.xticks([1, 2], ['Before_PCA', 'After_PCA'])
plt.ylabel('Sum of square error')
plt.title('Box Plot for SSE before and after PCA using Kmeans')
plt.show()

combined_df =
pd.concat([error_values_before,error_values_df_PCA],
keys=['Before_PCA','After_PCA'])
plt.boxplot([combined_df.loc['Before_PCA']['Silheoutte_Score'],
combined_df.loc['After_PCA']['Silheoutte_Score']])
plt.xticks([1, 2], ['Before_PCA', 'After_PCA'])
plt.ylabel('Silheoutte_Score')
plt.title('Box Plot for Silheoutte_Score before and after PCA
using Kmeans')
plt.show()

combined_df =
pd.concat([error_values_before,error_values_df_PCA],
keys=['Before_PCA','After_PCA'])
plt.boxplot([combined_df.loc['Before_PCA']['run_time'],
combined_df.loc['After_PCA']['run_time']])
plt.xticks([1, 2], ['Before_PCA', 'After_PCA'])
plt.ylabel('run_time')
plt.title('Box Plot for run_time before and after PCA using
Kmeans')
plt.show()
```

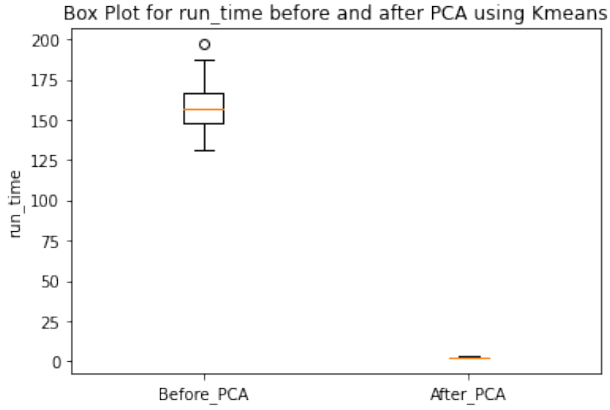
Plot/s



(9)



(10)



(11)

Discussion of Experiments

- **The error rate used in Algorithms is Within Sum of Squared Errors**
- **Reason why we cant use target variable error**
  - The target variable error takes into account the target labels,however here we have 5 target variables and when we perform label encoding on those variables,it is not making logical sense
  - Label encoding assigns the target variable randomly and the error class balances and imbalances will be depedent and statistically insignificant.
- **K means Algorithm**
- Have Ran K means algorithm on the entire dataset before PCA
- Have ran K means algorithm on dataset after applying PCA
- The number of columns after performing dimensionality reduction are 373
- Have plotted graphs for Before and After PCA and we can infer.
- **Within Sum of Square Errors**
- The WSS reduced after PCA and can be shown from the box plot

- The median of WSS is lower for Clustering after PCA
- **Silheoutte Analysis**
- Silhouette analysis is a method used to evaluate the quality of clustering results by measuring how well each data point fits within its assigned cluster. The analysis provides a silhouette score for each data point, which indicates the degree of similarity of that point to its own cluster compared to other clusters.
- The silhouette score ranges from -1 to 1, where a score of -1 indicates that the data point is assigned to the wrong cluster, a score of 0 indicates that the data point is near the decision boundary between two clusters, and a score of 1 indicates that the data point is well- clustered and belongs to its own cluster.
- We can infer there is a slight decrease in the silheoutte Score which indicates there might be **Loss of Information** after performing PCA
- We can increase the covered Variance to 97 percent or more and increase the number of PC's to increase the silheoutte score
- **Run Time** From the graphs it's clearly visible the Run time decreases after applying PCA because the number of dimensions reduce.

Problem 2

Randomly choose 50 points from the RNA-Seq data set (call this data set RNA-Seq$_{50}$) and perform hierarchical clustering. You are allowed to use R/Python packages for this question (Ignore the class variable while performing hierarchical clustering.) **[25 pt.]**

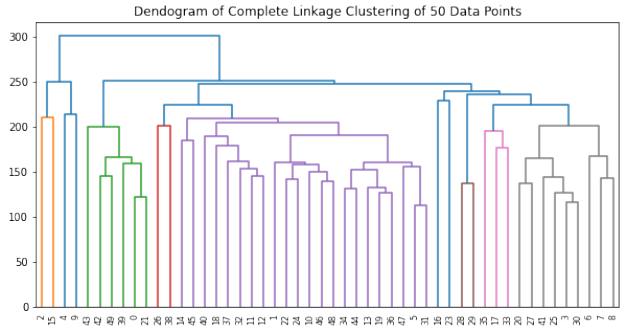
1. Using hierarchical clustering with complete linkage cluster RNA-Seq$_{50}$. Give the dendrogram.

R or Python script

```r
# Sample R Script With Highlighting
```

```python
# Sample Python Script With Highlighting
#Randomly selecting 50 Data Points
df_pancan_sample=df_pancan.sample(n=50)
df_pancan_sample.shape
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_pancan_sample_scaled =pd.DataFrame(scaler.fit_transform(df_pancan_sample))

heirarchial_clustering=linkage(df_pancan_sample_scaled,method='co
mplete')

# Plot a dendrogram of the hierarchical clustering
fig=plt.figure(figsize=(10, 5))
```

```
dendogram=dendrogram(heirarchial_clustering)

# display the dendrogram

plt.xticks(rotation=90)
plt.title('Dendogram of Complete Linkage Clustering of 50 Data Points')
# show the plot
plt.show()
```



(12)

2. Cut the dendrogram at a height that results in 5 distinct clusters. Calculate the error-rate.

R or Python script

```
# Sample R Script With Highlighting
```

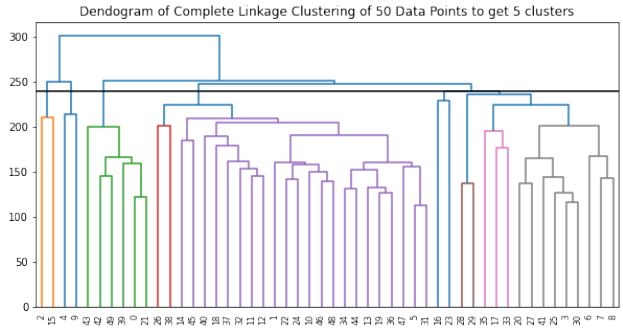
```
# Sample Python Script With Highlighting

distance_threshold = heirarchial_clustering[-5, 2]
from scipy.cluster.hierarchy import fcluster
labels = fcluster(heirarchial_clustering, distance_threshold,
criterion='distance')
print(labels)
'''
Output
[3 4 1 5 2 4 5 5 5 2 4 4 4 4 4 1 5 5 4 4 5 3 4 5 4 5 4 5 5 5 5 4
4 5 4 5 4
 4 4 3 4 5 3 3 4 4 4 4 4 3]

 '''

 '''Code for Calculating Error Rate Withing Sum of Square Errors
 '''
 distance_threshold = heirarchial_clustering[-5, 2]
from scipy.cluster.hierarchy import fcluster
labels = fcluster(heirarchial_clustering, distance_threshold, criterion='distance')
def heirarchial_wss(input_dataframe,labels_array,no_of_clusters):
    input_dataframe_clustered=input_dataframe.copy()
    input_dataframe_clustered['Labels']=labels_array
    new_centroids=input_dataframe_clustered.groupby('Labels').mea
    n()
```

```
    new_centroids=new_centroids.T
    total_error=[]
    for cluster in range(1,no_of_clusters+1):
        df_data_label_cluster=input_dataframe_clustered[input_dat
        aframe_clustered['Labels']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Labels'
        ,axis=1)
        centroids=pd.DataFrame(new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids.
        T,metric='euclidean')
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

heirarchial_wss(df_pancan_sample,labels,5)
#Performing PCA on reduced Dataset
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler

def covariance(input_dataframe):
    '''
    This function takse input as a standardized dataframe
    '''
    input_dataframe_mean =
    input_dataframe.swifter.apply(np.mean, axis=0)
    input_dataframe_centered= input_dataframe-
    input_dataframe_mean
    with tqdm(total=input_dataframe.shape[1], desc="Calculating
    Covariance Matrix") as pbar:
        cov_matrix=np.cov(input_dataframe.T)
        pbar.update()
    return cov_matrix,input_dataframe_centered

def principal_component_analysis(input_dataframe):
    '''
    This function takes input_dataframe,stadndardizes it and
    number of components as the number of components required
    by PC
    '''
    scaler = StandardScaler()
    input_dataframe_scaled
    =pd.DataFrame(scaler.fit_transform(input_dataframe))
    #Calling the covriance function
    covariance_matrix,input_dataframe_centered=covariance(input_
    dataframe_scaled)
    #Calculates Covariance Matirx
    eigen_values,eigen_vectors=np.linalg.eig(covariance_matrix)
    #Calculates Eigen Values and Eigen Vectors
    sorted_indices=np.argsort(eigen_values)
    #Sort the elements in descending order
    sorted_indices=sorted_indices[::-1]
```

```
    explained_variances = eigen_values / np.sum(eigen_values)
    variance_explained_ratios =
    pd.DataFrame(explained_variances[sorted_indices], columns=
    ["variance_explained_ratio"])
    variance_explained_ratios["cumulative_variance_explained_rat
    io"] =
    variance_explained_ratios["variance_explained_ratio"].cumsum()
    #Find the number of components that explain 90% of variance
    number_of_components =
    variance_explained_ratios["cumulative_variance_explained_rat
    io"][variance_explained_ratios["cumulative_variance_explaine
    d_ratio"] <= 0.90].count() + 1
    print(variance_explained_ratios)
    print("Number of Principal components explain 90% of
    variance are {}".format(number_of_components))




    #Taking Top Eigen Values and Top Eigen Vectors
    top_eigen_values_indices=sorted_indices[:number_of_component
    s]
    top_eigen_vectors=eigen_vectors[:,top_eigen_values_indices]

     #Variance Calculations Plot
    explained_variances = eigen_values/np.sum(eigen_values)
    variance_explained =
    pd.DataFrame(eigen_values[top_eigen_values_indices] /
    sum(eigen_values))

    variance_explained['PC_Feature']=top_eigen_values_indices

    variance_explained_plot=pd.Series(eigen_values[top_eigen_val
    ues_indices] / sum(eigen_values))


    #Cumulative Variance Plot
    cumulative_variance_explained =
    np.cumsum(variance_explained_plot)
    cumulative_variance_explained_plot =
    pd.Series(cumulative_variance_explained)



    #Projecting Principal Components
    principal_components=input_dataframe_centered.dot(top_eigen_
    vectors)
    principal_components.columns=[f'PC{i+1}' for i in
    range(number_of_components)]



    #Calculate the loadings
```

```
    loadings =
    pd.DataFrame(top_eigen_vectors,index=input_dataframe.columns
    )

    df_principal_components=pd.DataFrame(principal_components,
    columns=[f'PC{i+1}' for i in range(number_of_components)])
    #PLotting the graph
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
    ax[0].plot(np.arange(1,
    number_of_components+1),variance_explained_plot, 'o-')
    ax[0].set_xlabel('Principal Component')
    ax[0].set_ylabel('Proportion of Variance Explained')
    ax[0].set_title('Scree Plot')


    ax[1].plot(np.arange(1,
    number_of_components+1),cumulative_variance_explained_plot,
    'o-')
    ax[1].set_xlabel('Principal Component')
    ax[1].set_ylabel('Cumulative Proportion of Variance
    Explained')
    ax[1].set_title('Cumulative Scree Plot')
    plt.tight_layout()
    plt.show()

    #Correlation between PC1 and PC2

    plt.scatter(principal_components['PC1'],
    principal_components['PC2'])

    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title('Scatter plot of PC1 against PC2')
    plt.show()


    principal_components_temp=principal_components[['PC1','PC2']
    ]
    corr_matrix = principal_components_temp.corr()
    print('Correlation matrix:')
    print(corr_matrix)

    total_variance_explained=cumulative_variance_explained_plot[
    1]
    print("The total variance explained by first two PC's is
    {}".format(total_variance_explained))

    return
    variance_explained,loadings,principal_components,cumulative_
    variance_explained
```

Discussion of Experiments



(13)

- **Method to Get Exactly Five Clusters**
- In the code above, heirarchialclustering variable stores the linkage matrix generated by hierarchical clustering.
- The linkage matrix has n-1 rows and 4 columns, where n is the number of data points. Each row in the linkage matrix represents a cluster merger, with the first two columns specifying the indices of the clusters being merged, the third column representing the distance between these clusters, and the fourth column indicating the size of the newly formed cluster.
- In hierarchical clustering, the number of clusters is determined by the distance threshold used to form clusters.
- In the code above, the distance threshold that results in exactly five clusters is obtained by accessing the distance value corresponding to the fourth last row of the linkage matrix heirarchialclustring, using the index notation heirarchialclustering[-5,2].
- The reason why this works is because the linkage matrix heirarchialclustering stores information about the merges that occur during hierarchical clustering in a specific order. The rows of the matrix are sorted in increasing order of the distances between the clusters being merged.
- To obtain a specific number of clusters, we need to identify the distance threshold that will result in the desired number of clusters. We can do this by looking at the dendogram and finding the vertical line that intersects with the desired number of clusters. Alternatively, we can use the linkage matrix and count the number of rows that will result in the desired number of clusters.
- In this case, the [-5, 2] index notation accesses the distance value of the fourth last row of the linkage matrix. This distance value represents the distance between the clusters being merged that will result in five clusters. Therefore, using this distance threshold to form the final clusters will result in exactly five clusters.
- **Error Rate**
- Have Calculated the Within Sum of Square Errors for 50 Data points and have taken the root to Normalize the value is **7839.589**

3. First, perform PCA on RNA-Seq$_{50}$ (Keep 90% of variance). Then hierarchically cluster the reduced data using complete linkage and Euclidean distance. Report the dendrogram.

R or Python script

```
# Sample R Script With Highlighting
```

```
# Sample Python Script With Highlighting
#Performing PCA
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler

def covariance(input_dataframe):
    '''
    This function takse input as a standardized dataframe
    '''
    input_dataframe_mean =
    input_dataframe.swifter.apply(np.mean, axis=0)
    input_dataframe_centered= input_dataframe-\
    input_dataframe_mean
    with tqdm(total=input_dataframe.shape[1], desc="Calculating
    Covariance Matrix") as pbar:
        cov_matrix=np.cov(input_dataframe.T)
        pbar.update()
    return cov_matrix,input_dataframe_centered

def principal_component_analysis(input_dataframe):
    '''
    This function takes input_dataframe,stadndardizes it and
    number of components as the number of components required
    by PC
    '''
    scaler = StandardScaler()
    input_dataframe_scaled
    =pd.DataFrame(scaler.fit_transform(input_dataframe))
    #Calling the covriance function
    covariance_matrix,input_dataframe_centered=covariance(input_
    dataframe_scaled)
    #Calculates Covariance Matirx
    eigen_values,eigen_vectors=np.linalg.eig(covariance_matrix)
    #Calculates Eigen Values and Eigen Vectors
    sorted_indices=np.argsort(eigen_values)
    #Sort the elements in descending order
    sorted_indices=sorted_indices[::-1]


    explained_variances = eigen_values / np.sum(eigen_values)
    variance_explained_ratios =
    pd.DataFrame(explained_variances[sorted_indices], columns=
    ["variance_explained_ratio"])
    variance_explained_ratios["cumulative_variance_explained_rat
    io"] =

    variance_explained_ratios["variance_explained_ratio"].cumsum
    ()
    #Find the number of components that explain 90% of variance

    number_of_components =
    variance_explained_ratios["cumulative_variance_explained_rat
```

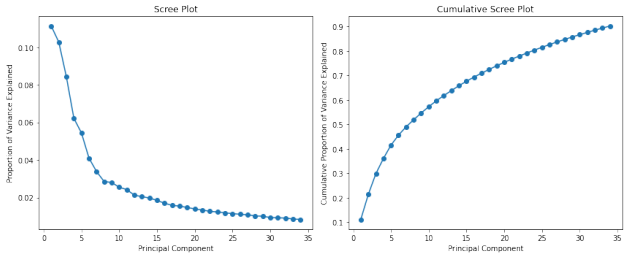
```
io"]
[variance_explained_ratios["cumulative_variance_explained_ra
tio"] <= 0.90].count() + 1
print(variance_explained_ratios)
print("Number of Principal components explain 90% of
variance are {}".format(number_of_components))



#Taking Top Eigen Values and Top Eigen Vectors
top_eigen_values_indices=sorted_indices[:number_of_component
s]
top_eigen_vectors=eigen_vectors[:,top_eigen_values_indices]

 #Variance Calculations Plot
explained_variances = eigen_values/np.sum(eigen_values)
variance_explained =
pd.DataFrame(eigen_values[top_eigen_values_indices] /
sum(eigen_values))
variance_explained['PC_Feature']=top_eigen_values_indices
variance_explained_plot=pd.Series(eigen_values[top_eigen_val
ues_indices] / sum(eigen_values))


#Cumulative Variance Plot
cumulative_variance_explained =
np.cumsum(variance_explained_plot)
cumulative_variance_explained_plot =
pd.Series(cumulative_variance_explained)



#Projecting Principal Components
principal_components=input_dataframe_centered.dot(top_eigen_
vectors)
principal_components.columns=[f'PC{i+1}' for i in
range(number_of_components)]



#Calculate the loadings
loadings =
pd.DataFrame(top_eigen_vectors,index=input_dataframe.columns
)

df_principal_components=pd.DataFrame(principal_components,
columns=[f'PC{i+1}' for i in range(number_of_components)])
#PLotting the graph
fig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax[0].plot(np.arange(1,
number_of_components+1),variance_explained_plot, 'o-')
ax[0].set_xlabel('Principal Component')
```

```
    ax[0].set_ylabel('Proportion of Variance Explained')
    ax[0].set_title('Scree Plot')


    ax[1].plot(np.arange(1,
    number_of_components+1),cumulative_variance_explained_plot,
    'o-')
    ax[1].set_xlabel('Principal Component')
    ax[1].set_ylabel('Cumulative Proportion of Variance
    Explained')
    ax[1].set_title('Cumulative Scree Plot')
    plt.tight_layout()
    plt.show()

    #Correlation between PC1 and PC2

    plt.scatter(principal_components['PC1'],
    principal_components['PC2'])
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title('Scatter plot of PC1 against PC2')
    plt.show()

    principal_components_temp=principal_components[['PC1','PC2']
    ]
    corr_matrix = principal_components_temp.corr()
    print('Correlation matrix:')
    print(corr_matrix)

    total_variance_explained=cumulative_variance_explained_plot[
    1]
    print("The total variance explained by first two PC's is
    {}".format(total_variance_explained))

    return
    variance_explained,loadings,principal_components,cumulative_
    variance_explained

variance_explained_50,loadings_50,principal_components_50,cumula
tive_variance_explained_50=principal_component_analysis(df_pancan_sample)

#Heirarchial clustering on reduced Dataset
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
import matplotlib.pyplot as plt

heirarchial_clustering=linkage(principal_components_50,method='c
omplete')



# plot a dendrogram of the hierarchical clustering
fig=plt.figure(figsize=(10, 5))
```

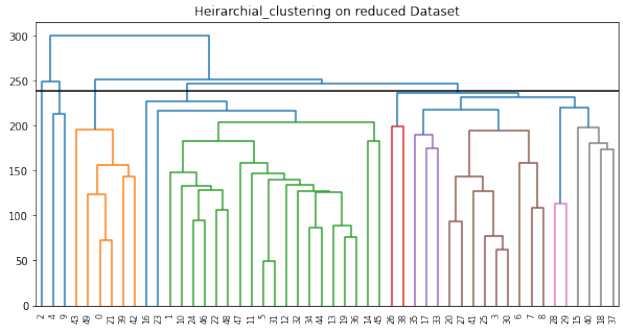
```
dendogram=dendrogram(heirarchial_clustering)

# display the dendrogram

plt.xticks(rotation=90)
plt.axhline(y=distance_threshold, c='k')
plt.title('Heirarchial_clustering on reduced Dataset')
# show the plot
plt.show()

distance_threshold = heirarchial_clustering[-5, 2]
from scipy.cluster.hierarchy import fcluster
labels = fcluster(heirarchial_clustering, distance_threshold,
criterion='distance')
print(labels)

def complex_euclidean_distance(u, v):
    return np.sqrt(np.sum(np.abs(u - v) ** 2))

def
heirarchial_wss_complex(input_dataframe,labels_array,no_of_clust
ers):
    input_dataframe_clustered=input_dataframe.copy()
    input_dataframe_clustered['Labels']=labels_array
    new_centroids=input_dataframe_clustered.groupby('Labels').me
    an()
    new_centroids=new_centroids.T
    total_error=[]
    for cluster in range(1,no_of_clusters+1):
        df_data_label_cluster=input_dataframe_clustered[input_da
        taframe_clustered['Labels']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Labels
        ',axis=1)
        centroids=pd.DataFrame(new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids
        .T,metric=complex_euclidean_distance)
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)
```

Plot/s

Number of Principal components explain 90 percent of variance are 34



(14)



(15)

4. Cut the dendrogram at a height that results in 5 distinct clusters. Give the error-rate. Discuss your findings, i.e., how did PCA affect hierarchical clustering results?

R or Python script

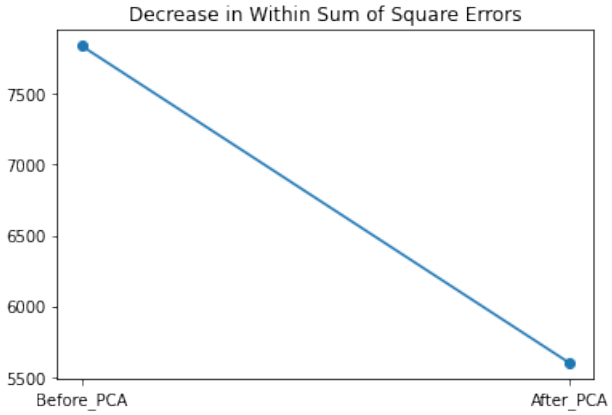
```
# Sample R Script With Highlighting
```

```
def complex_euclidean_distance(u, v):
    return np.sqrt(np.sum(np.abs(u - v) ** 2))

def
heirarchial_wss_complex(input_dataframe,labels_array,no_of_clust
ers):
    input_dataframe_clustered=input_dataframe.copy()
    input_dataframe_clustered['Labels']=labels_array
    new_centroids=input_dataframe_clustered.groupby('Labels').me
    an()
    new_centroids=new_centroids.T
    total_error=[]
    for cluster in range(1,no_of_clusters+1):
        df_data_label_cluster=input_dataframe_clustered[input_da
        taframe_clustered['Labels']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Labels
        ',axis=1)
        centroids=pd.DataFrame(new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids
        .T,metric=complex_euclidean_distance)
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

heirarchial_wss_complex(principal_components_50,labels,5)
```

Discussion of Experiments

- After applying Heirarchial clustering on reduced dataset from PCA we can see the Square root of Within Sum of Square Errors decreases.
- PCA is used to reduce the number of variables that are used to calculate the distance between observations, which can in turn decrease the within sum of square errors.

- When performing hierarchical clustering, the distance between two observations is typically calculated based on the values of all the variables in the dataset. However, if the dataset contains many variables, some of which may be highly correlated, then including all of these variables in the distance calculation can lead to redundant information and an overemphasis on certain features.
- By using PCA to reduce the number of variables used in the distance calculation, we can focus on the most important features of the data and eliminate the effects of correlated variables. This can lead to a more accurate clustering solution and a decrease in the within sum of square errors.
- Thus after applying PCA we get the Normalized Sum of Square errors as **5600.074**



(16)

Problem 3

Run your k-means clustering program from previous assignment for 20 runs and hierarchical clustering with three different linkage techniques over the RNA-Seq data set. Compare those 4 different clustering algorithms for $k = 5$ using appropriate cluster validity techniques, i.e., internal, external or relative indices. Plots are generally a good way to convey complex ideas quickly, i.e., box plots, whisker plots. Discuss your results, i.e., Did clustering improve after PCA? What algorithm performs best before and after PCA? **[25 pt.]**

R or Python script

```
# Sample R Script With Highlighting
```

```
def complex_euclidean_distance(u, v):
    return np.sqrt(np.sum(np.abs(u - v) ** 2))
def
heirarchial_wss(input_dataframe,labels_array,no_of_clusters):
    input_dataframe_clustered=input_dataframe.copy()
    input_dataframe_clustered['Labels']=labels_array
    new_centroids=input_dataframe_clustered.groupby('Labels').me
    an()
    new_centroids=new_centroids.T
    total_error=[]
    for cluster in range(1,no_of_clusters+1):
        df_data_label_cluster=input_dataframe_clustered[input_da
```

```
        taframe_clustered['Labels']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Labels
       ',axis=1)
        centroids=pd.DataFrame(new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids
        .T,metric=complex_euclidean_distance)
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

def silheoutte_score(input_dataframe,labels):
    from sklearn.metrics import silhouette_score
    silhouette_avg = silhouette_score(input_dataframe, labels)
    return silhouette_avg

def Calinski_Harbaz_score(input_dataframe,labels):
    from sklearn.metrics import calinski_harabasz_score
    chs=calinski_harabasz_score(input_dataframe,labels)
    return chs
def heirarchial_code_run(linkage_type):
    import pandas as pd
    from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import StandardScaler
    from scipy.cluster.hierarchy import fcluster
    scaler = StandardScaler()
    df_pancan_sample_scaled =pd.DataFrame(scaler.fit_transform(df_pancan))
    heirarchial_clustering=linkage(df_pancan_sample_scaled,method=linkage_type)
    #For exactly Five Clusters
    distance_threshold = heirarchial_clustering[-5, 2]
    labels = fcluster(heirarchial_clustering, distance_threshold, criterion='distance')
    error=heirarchial_wss(df_pancan,labels,5)
    silhouette_avg=silheoutte_score(df_pancan,labels)
    chs=Calinski_Harbaz_score(df_pancan,labels)
    return error,linkage_type,silhouette_avg,chs

    error_dict={}
silheoutte_dict={}
chs_dict={}
linkage_type=['complete','ward','single','average']
for i in linkage_type:
    error,linkage_type,silheotte_score,chs=heirarchial_code_run(str(i))
    error_dict[i]=error
    silheoutte_dict[i]=silheotte_score
    chs_dict[i]=chs
print(error_dict)
print(silheoutte_dict)
print(chs_dict)

error_df=pd.DataFrame([error_dict])
error_df['Method']='Before_PCA'
silheoutte_df=pd.DataFrame([silheoutte_dict])
silheoutte_df['Method']='Before_PCA'
chs_df=pd.DataFrame([chs_dict])
```

```
chs_df['Method']='Before_PCA'

#Applying PCA to the dataset
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler

def covariance(input_dataframe):
    '''
    This function takse input as a standardized dataframe
    '''
    input_dataframe_mean = input_dataframe.swifter.apply(np.mean, axis=0)
    input_dataframe_centered= input_dataframe-input_dataframe_mean
    with tqdm(total=input_dataframe.shape[1], desc="Calculating
    Covariance Matrix") as pbar:
        cov_matrix=np.cov(input_dataframe.T)
        pbar.update()
    return cov_matrix,input_dataframe_centered

def principal_component_analysis(input_dataframe):
    '''
    This function takes input_dataframe,stadndardizes it and

    number of components as the number of components required
    by PC
    '''
    scaler = StandardScaler()
    input_dataframe_scaled =pd.DataFrame(scaler.fit_transform(input_dataframe))
    #Calling the covriance function
    covariance_matrix,input_dataframe_centered=covariance(input_dataframe_scaled)
    #Calculates Covariance Matirx
    eigen_values,eigen_vectors=np.linalg.eig(covariance_matrix)
    #Calculates Eigen Values and Eigen Vectors
    sorted_indices=np.argsort(eigen_values)
    #Sort the elements in descending order
    sorted_indices=sorted_indices[::-1]


    explained_variances = eigen_values / np.sum(eigen_values)

    variance_explained_ratios =
    pd.DataFrame(explained_variances[sorted_indices], columns=
    ["variance_explained_ratio"])
    variance_explained_ratios["cumulative_variance_explained_rat
    io"] =
    variance_explained_ratios["variance_explained_ratio"].cumsum
    ()

    #Find the number of components that explain 90% of variance
    number_of_components =
    variance_explained_ratios["cumulative_variance_explained_rat
    io"][variance_explained_ratios["cumulative_variance_explaine
    d_ratio"] <= 0.99].count() + 1
```

```
    print("Number of Principal components explain 99% of
    variance are {}".format(number_of_components))



    #Taking Top Eigen Values and Top Eigen Vectors
    top_eigen_values_indices=sorted_indices[:number_of_component
    s]
    top_eigen_vectors=eigen_vectors[:,top_eigen_values_indices]

     #Variance Calculations Plot
    explained_variances = eigen_values/np.sum(eigen_values)
    variance_explained =
    pd.DataFrame(eigen_values[top_eigen_values_indices] /
    sum(eigen_values))
    variance_explained['PC_Feature']=top_eigen_values_indices
    variance_explained_plot=pd.Series(eigen_values[top_eigen_val
    ues_indices] / sum(eigen_values))


    #Cumulative Variance Plot
    cumulative_variance_explained =
    np.cumsum(variance_explained_plot)
    cumulative_variance_explained_plot =
    pd.Series(cumulative_variance_explained)



    #Projecting Principal Components
    principal_components=input_dataframe_centered.dot(top_eigen_vectors)
    principal_components.columns=[f'PC{i+1}' for i in range(number_of_components)]



    #Calculate the loadings
    loadings = pd.DataFrame(top_eigen_vectors,index=input_dataframe.columns)

    df_principal_components=pd.DataFrame(principal_components,
    columns=[f'PC{i+1}' for i in range(number_of_components)])
    #PLotting the graph
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
    ax[0].plot(np.arange(1,
    number_of_components+1),variance_explained_plot, 'o-')
    ax[0].set_xlabel('Principal Component')
    ax[0].set_ylabel('Proportion of Variance Explained')
    ax[0].set_title('Scree Plot')


    ax[1].plot(np.arange(1,
    number_of_components+1),cumulative_variance_explained_plot,
```

```
    'o-')
    ax[1].set_xlabel('Principal Component')
    ax[1].set_ylabel('Cumulative Proportion of Variance Explained')
    ax[1].set_title('Cumulative Scree Plot')
    plt.tight_layout()
    plt.show()

    #Correlation between PC1 and PC2

    plt.scatter(principal_components['PC1'], principal_components['PC2'])
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title('Scatter plot of PC1 against PC2')
    plt.show()

    principal_components_temp=principal_components[['PC1','PC2']]
    corr_matrix = principal_components_temp.corr()
    print('Correlation matrix:')
    print(corr_matrix)

    total_variance_explained=cumulative_variance_explained_plot[
    1]
    print("The total variance explained by first two PC's is
    {}".format(total_variance_explained))

    return
    variance_explained,loadings,principal_components,cumulative_
    variance_explained
#Heierarchial Clustering after PCA

def heirarchial_code_run_after_PCA(linkage_type):
    import pandas as pd
    from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import StandardScaler
    from scipy.cluster.hierarchy import fcluster
    scaler = StandardScaler()
    principal_components_real=principal_components.swifter.apply
    (np.real)
    principal_components_scaled
    =pd.DataFrame(scaler.fit_transform(principal_components_real
    ))
    heirarchial_clustering=linkage(principal_components_scaled,m
    ethod=linkage_type)
    distance_threshold = heirarchial_clustering[-5, 2]
    labels = fcluster(heirarchial_clustering, distance_threshold, criterion='distance')
    error=heirarchial_wss(principal_components,labels,5)
    silhouette_avg=silheoutte_score(principal_components_real,labels)
    chs=Calinski_Harbaz_score(principal_components_real,labels)
    return error,linkage_type,silhouette_avg,chs

    error_dict_PCA={}
silheoutte_dict_PCA={}
```

```
chs_dict_PCA={}
linkage_type=['complete','ward','single','average']
for i in linkage_type:
    error,linkage_type,silheotte_score,chs=heirarchial_code_run_after_PCA(str(i))
    error_dict_PCA[i]=error
    silheoutte_dict_PCA[i]=silheotte_score
    chs_dict_PCA[i]=chs
print(error_dict_PCA)
print(silheoutte_dict_PCA)
print(chs_dict_PCA)

error_df_PCA=pd.DataFrame([error_dict_PCA])
error_df_PCA['Method']='After_PCA'
silheoutte_df_PCA=pd.DataFrame([silheoutte_dict_PCA])
silheoutte_df_PCA['Method']='After_PCA'
chs_df_PCA=pd.DataFrame([chs_dict_PCA])
chs_df_PCA['Method']='After_PCA'

#Plotting the graphs
error_final_df=pd.concat([error_df,error_df_PCA])
error_final_df
df_melt = pd.melt(error_final_df, id_vars=['Method'],
var_name='Clustering Method', value_name='Within Sum of Squares')
sns.set_style('whitegrid')
plt.figure(figsize=(6,4))
ax = sns.barplot(x='Clustering Method', y='Within Sum of Squares',
hue='Method', data=df_melt)
ax.set_xlabel('Clustering Method')
ax.set_ylabel('Within Sum of Squares')
ax.set_title('Clustering Results of Heirarchial for SSE')
plt.show()

silheoutte_final_df=pd.concat([silheoutte_df,silheoutte_df_PCA])
df_melt = pd.melt(silheoutte_final_df, id_vars=['Method'],
var_name='Clustering Method', value_name='Within Sum of
Squares')
sns.set_style('whitegrid')
plt.figure(figsize=(6,4))
ax = sns.barplot(x='Clustering Method', y='Within Sum of
Squares', hue='Method', data=df_melt)
ax.set_xlabel('Clustering Method')
ax.set_ylabel('Silheoutte_Scores')
ax.set_title('Clustering Results of Heirarchial Silheoutte_Scores')
plt.show()

chs_final_df=pd.concat([chs_df,chs_df_PCA])
df_melt = pd.melt(chs_final_df, id_vars=['Method'],
var_name='Clustering Method', value_name='Within Sum of
Squares')
sns.set_style('whitegrid')
ax = sns.barplot(x='Clustering Method', y='Within Sum of
Squares', hue='Method', data=df_melt)
ax.set_xlabel('Clustering Method')
```

```
ax.set_ylabel('CHS_Scores')
ax.set_title('Clustering Results of Heirarchial Celinski-
Harabasaz Score')
plt.show()

#Running K means on entire dataset
import numpy as np
import swifter
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import cdist
import time


def get_random_centroids(input_dataframe,no_of_clusters):
    list_of_centroids = []
    for cluster in range(no_of_clusters):
        random_centroid = input_dataframe.swifter.apply(lambda x:float(x.sample()))
        list_of_centroids.append(random_centroid)

    centroid_df=pd.concat(list_of_centroids,axis=1)
    centroid_df.index.name='Cluster_Assigned'
    return centroid_df

def get_labels(input_dataframe,centroid_df):
    euclidean_distances = centroid_df.swifter.apply(lambda x:
    np.sqrt(((input_dataframe - x) ** 2).sum(axis=1)))
    return pd.DataFrame(euclidean_distances.idxmin(axis=1))


def get_new_centroids(df_clustered_label,input_dataframe):
    df_original_label_join=input_dataframe.join(df_clustered_lab
    el)
    df_original_label_join.rename(columns=
    {0:'Cluster_Assigned'},inplace=True)
    new_centroids=df_original_label_join.groupby('Cluster_Assign
    ed').mean()
    return new_centroids.T


def
kmeans_llyod(input_dataframe,no_of_clusters,threshold,no_of_iter
ations):
    start_time=time.time()
    iteration=0
    initial_centroid=get_random_centroids(input_dataframe,no_of_
    clusters)
    initial_centroid_column_list=initial_centroid.columns.to_lis
    t()

    while True:

        df_cluster_label=get_labels(input_dataframe,initial_cent
        roid)
```

```
        df_new_centroids=get_new_centroids(df_cluster_label,inpu
        t_dataframe)
        new_list_of_columns=df_new_centroids.columns.to_list()
        initial_set_columns = set(initial_centroid_column_list)
        new_set_columns = set(new_list_of_columns)
        missing_columns = initial_set_columns - new_set_columns
        for col in missing_columns:
            df_new_centroids[col]=initial_centroid[col]

        from scipy.spatial.distance import euclidean
        scalar_product =
        [euclidean(initial_centroid[col],df_new_centroids[col])
        for col in initial_centroid.columns]
        threshold_calculated=float(sum(scalar_product))/no_of_cl
        usters

        iteration+=1

        if threshold_calculated<threshold:
            print("The input Threshold was
            {}".format(threshold))
            print("The calculated threshold is
            {}".format(threshold_calculated))

        if iteration>no_of_iterations:
            print("Limit for iterations has exceeded")

        if threshold_calculated<threshold or iteration>no_of_iterations:
            sum_of_square_error=sum_of_square_error_function(df_
            cluster_label,input_dataframe,df_new_centroids,no_of
            _clusters)
            df_cluster_label_copy=df_cluster_label.copy()
            df_cluster_label_copy.rename(columns=
            {0:'Cluster_Assigned'},inplace=True)
            labels=df_cluster_label_copy['Cluster_Assigned'].to_list()
            silheoutte_score=silheoutte_score_Kmeans(input_dataframe,labels)
            chs_score=Calinski_Harbaz_score_Kmeans(input_dataframe,labels)
            end_time=time.time()
            return
            df_new_centroids,sum_of_square_error,silheoutte_score,chs_score,end_
            time-start_time
            break
        else:
            initial_centroid= df_new_centroids


def
sum_of_square_error_function(df_cluster_label,input_dataframe,df_new_centroids,n
o_of_clusters):
    df_data_label=input_dataframe.join(df_cluster_label)
    df_data_label.rename(columns={0:'Cluster_Assigned'},inplace=True)
    total_error=[]
    for cluster in range(no_of_clusters):
```

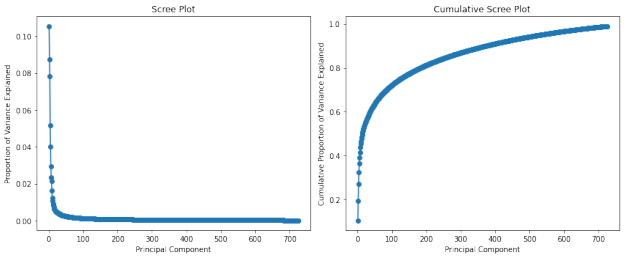
```
        df_data_label_cluster=df_data_label[df_data_label['Cluster_Assigned']==c
        luster]
        df_data_label_cluster=df_data_label_cluster.drop('Cluster_Assigned',axis
        =1)
        centroids=pd.DataFrame(df_new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids.T,metric='eucli
        dean')
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

def silheoutte_score_Kmeans(input_dataframe,labels):
    from sklearn.metrics import silhouette_score
    silhouette_avg = silhouette_score(input_dataframe, labels)
    return silhouette_avg

def Calinski_Harbaz_score_Kmeans(input_dataframe,labels):
    from sklearn.metrics import calinski_harabasz_score
    chs=calinski_harabasz_score(input_dataframe,labels)
    return chs
error_values=[]
for no_of_experiments in range(1,21):
    final_centroids,sum_of_squared_error,sil_score,chs_score,run_time=kmeans_lly
    od(df_pancan,5,10,100)
    error_values.append([5,no_of_experiments,sum_of_squared_error,sil_score,chs_
    score,run_time])
error_values_df= pd.DataFrame(error_values,columns=['No_of_Clusters',
'Iteration
Number','Sum_of_squared_Errors','Silheoutte_Score','Chs_score','run_time'])

#After Applying PCA
principal_components_real_part=principal_components.copy()
principal_components_real_part=principal_components_real_part.applymap(lambda
x:x.real)
error_values=[]
for no_of_experiments in range(1,21):
    final_centroids,sum_of_squared_error,sil_score,chs_score,run_time=kmeans_lly
    od(principal_components_real_part,5,10,100)
    error_values.append([5,no_of_experiments,sum_of_squared_error,sil_score,chs_
    score,run_time])
error_values_df_PCA= pd.DataFrame(error_values,columns=['No_of_Clusters',
'Iteration
Number','Sum_of_squared_Errors','Silheoutte_Score','Chs_score','run_time'])

#PLOTTING GRAPHS
combined_df = pd.concat([error_values_df,error_values_df_PCA],
keys=['Before_PCA','After_PCA'])
plt.boxplot([combined_df.loc['Before_PCA']['Sum_of_squared_Errors'],
combined_df.loc['After_PCA']['Sum_of_squared_Errors']])
plt.xticks([1, 2], ['Before_PCA', 'After_PCA'])
plt.ylabel('Sum of square error')
plt.title('Box Plot for SSE before and after PCA using Kmeans')
plt.show()
```

```
combined_df = pd.concat([error_values_df,error_values_df_PCA],
keys=['Before_PCA','After_PCA'])
plt.boxplot([combined_df.loc['Before_PCA']['Silheoutte_Score'],
combined_df.loc['After_PCA']['Silheoutte_Score']])

plt.xticks([1, 2], ['Before_PCA', 'After_PCA'])
plt.ylabel('Silheoutte_Score')
plt.title('Box Plot for Silheoutte_Score before and after PCA using Kmeans')
plt.show()

combined_df = pd.concat([error_values_df,error_values_df_PCA],
keys=['Before_PCA','After_PCA'])
plt.boxplot([combined_df.loc['Before_PCA']['run_time'],
combined_df.loc['After_PCA']['run_time']])
plt.xticks([1, 2], ['Before_PCA', 'After_PCA'])
plt.ylabel('run_time')
plt.title('Box Plot for run_time before and after PCA using Kmeans')
plt.show()

#Comparing all algorithms

k_means_entire=pd.concat([error_values_df,error_values_df_PCA])
k_means_entire.fillna('After_PCA',inplace=True)
k_means_stats=k_means_entire.groupby('Method').mean().reset_index()
k_means_stats['Algorithm']='K means'
k_means_stats=k_means_stats[['Method','Sum_of_squared_Errors','Algorithm']]
df_melt_error = pd.melt(error_final_df, id_vars=['Method'],
var_name='Algorithm', value_name='Sum_of_squared_Errors')
error_combined_cluster=pd.concat([df_melt_error,k_means_stats])
plt.figure(figsize=(6,6))
sns.set_style('whitegrid')
ax = sns.barplot(x='Algorithm', y='Sum_of_squared_Errors', hue='Method',
data=error_combined_cluster)
ax.set_xlabel('Clustering Method')
ax.set_ylabel('Sum_of_Squared_Errors')
ax.set_title('Clustering Results of all methods for SSE')
plt.show()

k_means_entire=pd.concat([error_values_df,error_values_df_PCA])
k_means_entire.fillna('After_PCA',inplace=True)
k_means_stats=k_means_entire.groupby('Method').mean().reset_index()
k_means_stats['Algorithm']='K means'
k_means_stats=k_means_stats[['Method','Silheoutte_Score','Algorithm']]
df_melt_error = pd.melt(silheoutte_final_df, id_vars=['Method'],
var_name='Algorithm', value_name='Silheoutte_Score')
error_combined_cluster=pd.concat([df_melt_error,k_means_stats])
plt.figure(figsize=(6,6))
sns.set_style('whitegrid')
ax = sns.barplot(x='Algorithm', y='Silheoutte_Score', hue='Method',
data=error_combined_cluster)
ax.set_xlabel('Clustering Method')
ax.set_ylabel('Silheoutte_Score')
ax.set_title('Clustering Results of Silheoutte Score for all')
```

```
plt.show()
k_means_entire=pd.concat([error_values_df,error_values_df_PCA])
k_means_entire.fillna('After_PCA',inplace=True)
k_means_stats=k_means_entire.groupby('Method').mean().reset_index()
k_means_stats['Algorithm']='K means'
k_means_stats=k_means_stats[['Method','Chs_score','Algorithm']]
df_melt_error = pd.melt(chs_final_df, id_vars=['Method'], var_name='Algorithm',
value_name='Chs_score')
error_combined_cluster=pd.concat([df_melt_error,k_means_stats])
plt.figure(figsize=(6,6))
sns.set_style('whitegrid')
ax = sns.barplot(x='Algorithm', y='Chs_score', hue='Method',
data=error_combined_cluster)
ax.set_xlabel('Clustering Method')
ax.set_ylabel('Chs_score')
ax.set_title('Clustering Results of CHS Score for All')
plt.show()
#End
```

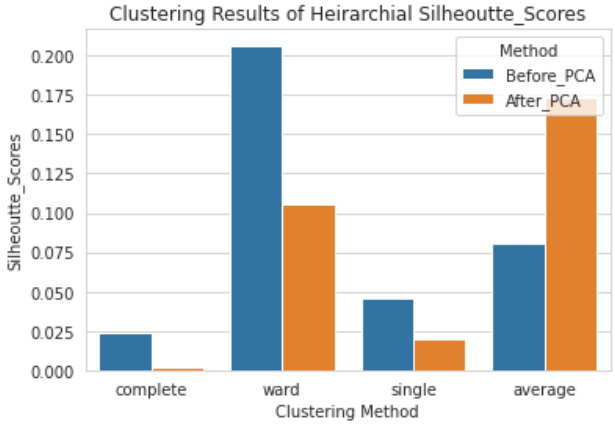
Plot/s

Number of Principal components explain 99 percent of variance are 725

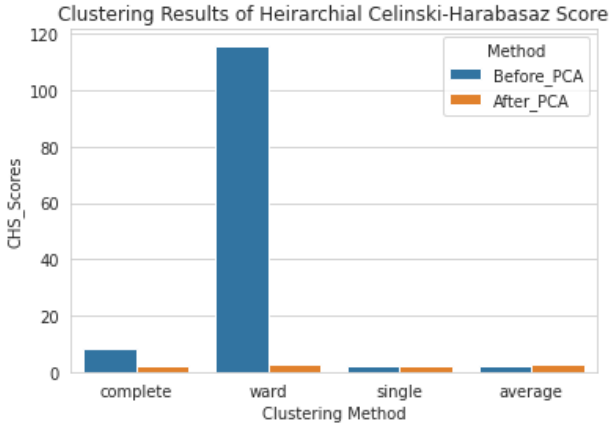


(17)

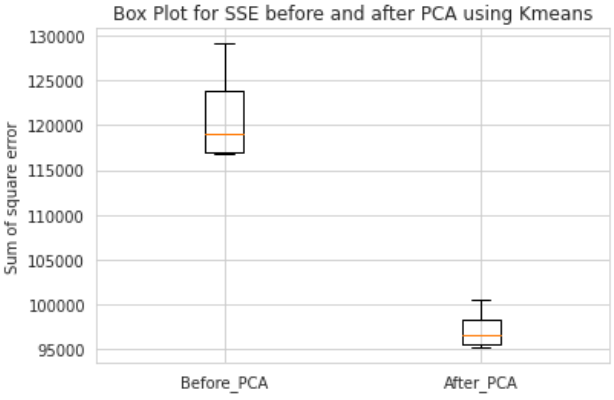
(18)


Clustering Results of Heirarchial Silheoutte_Scores
Method
Before_PCA
After_PCA
Silheoutte_Scores
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
complete
ward
single
average
Clustering Method

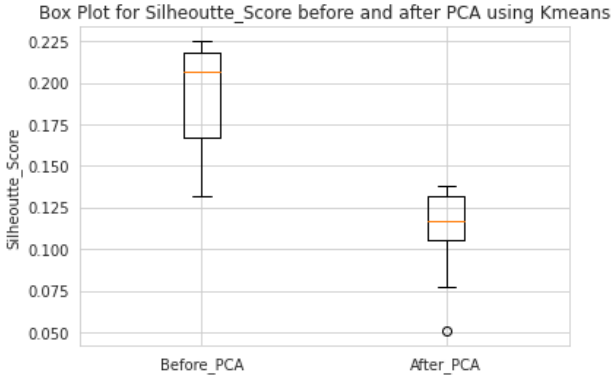
(19)


Clustering Results of Heirarchial Celinski-Harabasaz Score
Method
Before_PCA
After_PCA
CHS_Scores
120
100
80
60
40
20
0
complete
ward
single
average
Clustering Method

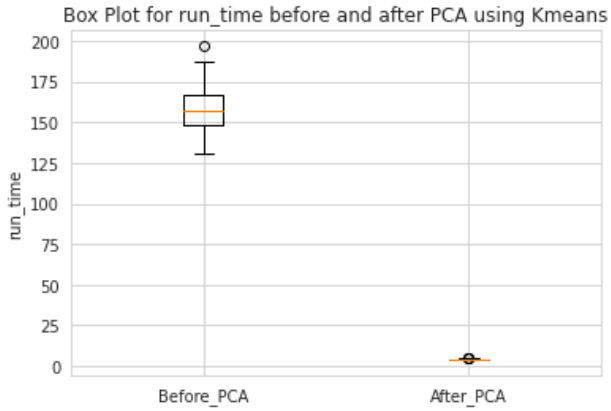
(20)


Box Plot for SSE before and after PCA using Kmeans
Sum of square error
130000
125000
120000
115000
110000
105000
100000
95000
Before_PCA
After_PCA

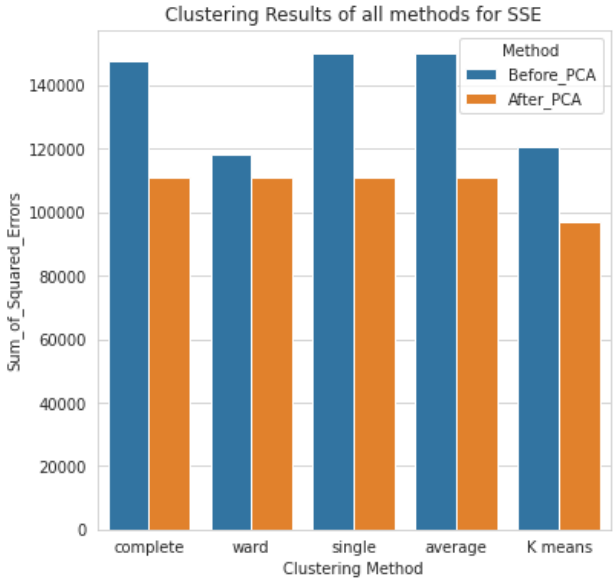
(21)


Box Plot for Silheoutte_Score before and after PCA using Kmeans
Silheoutte_Score
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
Before_PCA
After_PCA

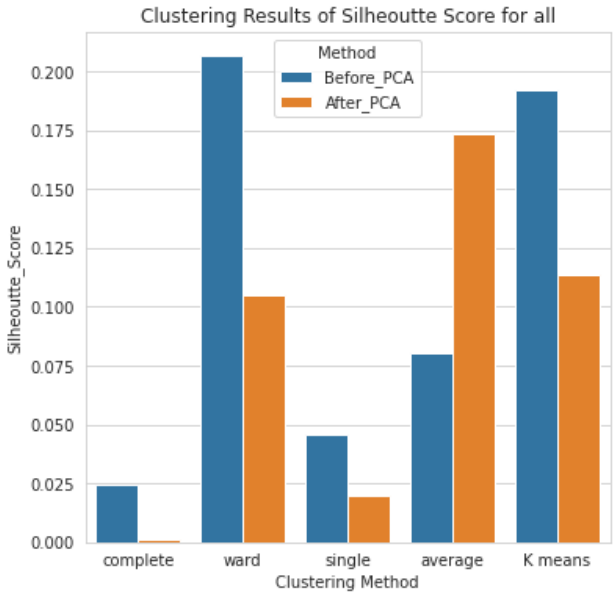
(22)


Box Plot for run_time before and after PCA using Kmeans
run_time
200
175
150
125
100
75
50
25
0
Before_PCA
After_PCA

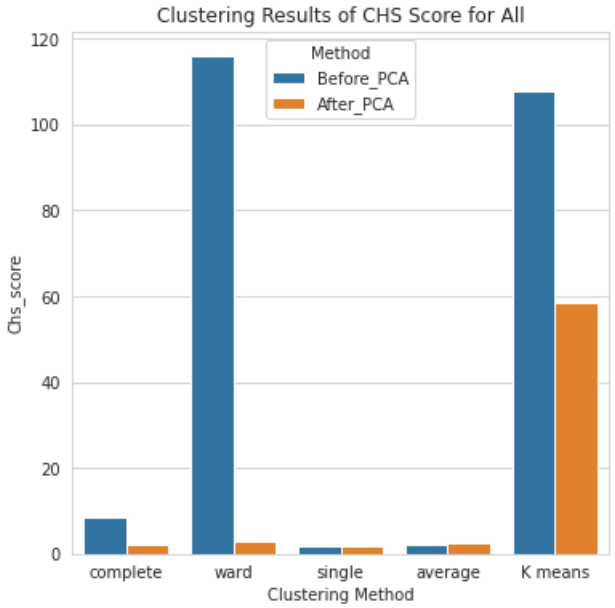
(23)


Clustering Results of all methods for SSE
Method
Before_PCA
After_PCA
Sum_of_Squared_Errors
140000
120000
100000
80000
60000
40000
20000
0
complete
ward
single
average
K means
Clustering Method

(24)



(25)



(26)

Discussion of Experiments

- **Indices Used**
- Silhouette score is a metric used to evaluate the quality of clustering results in unsupervised learning. It measures how well each data point in a cluster is separated from other clusters based on a given distance metric. It ranges from -1 to 1, with higher values indicating better clustering.
- **The Silhouette score** ranges from -1 to 1, where a score of -1 indicates that the data point is probably assigned to the wrong cluster, a score of 0 indicates that the data point is close to the decision boundary between two clusters, and a score of 1 indicates that the data point is well separated from other clusters.

- In clustering analysis, the Silhouette score is used to evaluate the quality of the clustering results and to select the optimal number of clusters. A higher Silhouette score indicates better clustering quality, while a lower score indicates that the clusters are not well separated.
- **The Calinski-Harabasz score** is a metric used to evaluate the quality of clustering results in unsupervised learning. It measures the ratio of the between-cluster dispersion and the within-cluster dispersion, and is also known as the Variance Ratio Criterion. A higher Calinski-Harabasz score indicates better clustering quality. The Calinski-Harabasz score is a metric used to evaluate the quality of clustering results in unsupervised learning. It measures the ratio of the between- cluster dispersion and the within-cluster dispersion, and is also known as the Variance Ratio Criterion. A higher Calinski-Harabasz score indicates better clustering quality.
- For calculating the Sum of Squared Errors have taken the Real part because after projecting the datapoints can go in complex planes.
- **Explanation of Indices**
- **Internal methods**: Internal methods evaluate the quality of clustering based on the data alone, without reference to external information. The most commonly used internal method for hierarchical clustering is the Calinski-Harabasz score, which measures the ratio of between-cluster variance to within-cluster variance. The higher the score, the better the clustering.
- **Relative methods**: Relative methods compare the quality of clustering between different clustering solutions. The most commonly used relative method for hierarchical clustering is the Silhouette score, which measures the average distance between each data point and its own cluster compared to other clusters. The higher the Silhouette score, the better the clustering.

Results From the experiment

- From the first graph we can conclude by keeping 99 percent variance the count of Principal Components are 725
- For Clustering results of Heirarchial Clustering using 4 different methods namely Ward,Avergae,Single and Complete and have plotted Sum of Square Errors for all the methods used as shown in second Graph
- The within Sum of Square errors has decreased in every method after applying PCA
- The Ward Method has the least sum of square errors.
- For other metric like Silheoutte and Celenski-Harabasaz score we can conclude that Ward method has the highest ration suggesting the Clustering was efficient in Ward's Method and has good cluster separation.
- However the value of Statistic isnt that high.
- After Applying PCA we see a decrease in those values suggesting there might be loss of some information which was required for Clustering
- The above conclusions can be drawn from graph 2,3 and 4
- Graphs 5,6,7 Shows the effect of PCA on K means Algorithm
- Grapg 5 Proves a decrease in Sum of Square Error in K means after applying PCA
- Run time drops drastically as we reduce the number of computations

- Thus K means performs better after PCA
- Graphs 7,8,9 draws overall comparison of all clustering algorithms.
- Based on the within sum of square error graph, the average method of hierarchical clustering has the highest bar length, while K means and Ward has the smallest before PCA. Therefore, Ward and K means performed well before applying PCA. After applying PCA, K means had the smallest error compared to Ward and other hierarchical clustering methods. Therefore, for the given dataset, K means can be used as the technique for clustering, with within sum of square error as the metric.
- For both datasets, the silhouette coefficient for Ward method is the maximum compared to other methods. The Calinski index is the maximum for Ward clustering and minimum for clustering using single linkage method.
- Different validation techniques show that Ward and K means performed best when the within sum of square error validation was used.
- It depends on the use case and the given problem statement to select a particular validation technique. Each technique has its pros and cons.
- **Clustering improved after performing PCA, resulting in reduced error rates**

Problem 4

The leader algorithm [1] represents each cluster using a point, known as a *leader*, and assigns each point to the cluster corresponding to the closest leader, unless this distance is above a user-specified threshold. In that case, the point becomes the leader in a new cluster **[10 pt.]**.

(a) What are the advantages and disadvantages of the leader algorithm as compared to K-means?

- **Explanation of Algorithm**
- Hartigan's leader algorithm is a hierarchical clustering algorithm that was proposed by John A. Hartigan in 1975.
- This algorithm is a divisive approach to clustering, meaning it begins with all data points in a single cluster and recursively splits them into smaller clusters until a stopping criterion is met.
- To begin, the algorithm selects a leader point from the data set at random, which serves as the initial center of the first cluster.
- Next, the algorithm calculates the distance between each point and the leader, and assigns each point to the closest cluster.
- The algorithm then selects a new leader for each cluster, which is the point with the smallest sum of distances to all the other points in the cluster.
- This process is repeated until a stopping criterion is met, such as a pre-defined number of clusters or a threshold for the distance between leaders of two clusters.
- **Advantages of Hartingan's Algorithm over K means**
- With reference from Research Paper Hartigan's K-Means Versus Lloyd's K-Means – Is It Time for a Change?
- Few Notable Advantages of Hartingan's Leader Algorithm are:
  - **Reduction of distance calculations**: In k-means clustering, distance calculations are performed for each data point with respect to all the cluster centers at every iteration.

  - This can be a computationally expensive step, particularly for large datasets or when the number of clusters is high.
  - In contrast, Hartigan's leader algorithm reduces the number of distance calculations required by only comparing each data point to a single leader point, rather than all the cluster centers. This reduces the computational complexity of the algorithm.
  - Also K means does a addiitonal computational step of Voronoi Diagram which is not required for Leader's Algorithms
- Few Disadvantages of Hartingan's Algorithm over K means
  - **Better Suited for Complex Clusters** K means is better suited for complex clusters
  - **Simplicity** K means is a simpler algorithm and generally used in real life datasets with optimisation parameters
  - **Sensitive to Leader** Hartingan's Leader Algorithm is sensitive to the first leader Selection

(b) Suggest ways in which the leader algorithm might be improved.

- Leader's Algorithm can be improved in these ways
- **Selection of Initial Leader** Since Leader's Algorithm is extremely sensitive to the Initial Leader Data point, We can use a better heuristic as we did in K means (k Means ++ Initialization)
- **Adaptive number of leader points**: The number of leader points is an important hyper parameter in Hartigan's algorithm, and it can affect the quality of the clustering results. An adaptive method that adjusts the number of leader points during the clustering process can improve the algorithm's robustness and flexibility
- **Parallelization** Since we calculate the distance from the leader to every other point we can parallelize the distance calculation to make it computationally better.
- **K means ++ Initialization** K means ++ Initialization can help in selecting the leader points to reduce the Within Sum of Square Errors after clustering and increases the inter Clusters distance.

Problem 5

Clusters of documents can be summarized by finding the top terms (words) for the documents in the cluster, e.g., by taking the most frequent k terms, where k is a constant, say 10, or by taking all terms that occur more frequently than a specified threshold. Suppose that K-means is used to find clusters of both documents and words for a document data set **[10 pt.]**

(a) How might a set of term clusters defined by the top terms in a documents cluster differ from the word clusters found by clustering the terms with K-means?

- The set of term clusters defined by the top terms in a documents cluster may differ from the word clusters found by clustering the terms with K-means.
- This is because the two approaches have different goals and operate at different levels of abstraction.
- The set of term clusters defined by the top terms in a documents cluster focuses on summarizing the content of a group of related documents by identifying the most important and frequent terms in the cluster.
- In contrast, K-means clustering of terms seeks to find groups of terms that have similar patterns of co-occurrence across the entire document corpus, regardless of whether they appear frequently or infrequently in any one document.

- Thus, while the former approach is more focused on the specific content of a given set of documents, the latter approach captures more general patterns of term co-occurrence across the entire corpus. **This can be illustrated with an example**
  - Let's say we have a large corpus of online product reviews from various categories such as electronics, clothing, and appliances.
  - We want to cluster both the documents (reviews) and the terms (words) in the corpus to identify patterns and themes in the reviews.
  - If we use the approach of finding the top terms in each document cluster, we might identify the most frequent words that appear in reviews of a particular product category.
  - For example, in the electronics category, we might find that terms such as "battery life", "screen resolution", and "user interface" are most frequently mentioned in reviews, whereas in the clothing category, terms such as "fit", "comfort", and "style" might be more common.
  - On the other hand, if we use K-means clustering of terms, we might identify groups of words that tend to co-occur frequently in reviews across all product categories.
  - For example, we might find that terms such as "price", "quality", and "customer service" tend to cluster together, suggesting that consumers often mention these factors together in their product reviews regardless of the category.

(b) How could term clustering be used to define clusters of documents?

- Term clustering is a method used to define clusters of documents by identifying groups of documents that share similar vocabulary. This can be accomplished using natural language processing (NLP) techniques such as creating a co-occurrence matrix of terms in the corpus.
- To calculate the frequency of terms we can use the Count Vectorizer technique to build a count vectorizer matrix
- The co-occurrence matrix represents the frequency of term co-occurrences across all documents in the corpus.
- A clustering algorithm such as K-means or hierarchical clustering can then be applied to group terms that frequently co-occur together into clusters.
- After the term clusters are defined, each document in the corpus can be represented as a vector of weights that indicates the frequency of terms in each cluster within the document. These document vectors can be used as input for another clustering algorithm to group similar documents together based on the similarity of their term cluster weights.
- So we perform Clustering twice to group the frequently occuring words and to group those frequently occuring words together

Submission

You must use LaTeX to turn in your assignments. Please submit the following two files via Canvas:

1. A .pdf with the name `yourname-hw5-everything.pdf` which you will get after compiling your .tex file.
2. A .zip file with the name `yourname-hw5.zip` which should contain your .tex, .pdf, codes(.py, .ipynb, .R, or .Rmd), and a README file. The README file should contain information about dependencies and how to run your codes.

References

[1] J. Hartigan. *Clustering Algorithms.* John Wiley and Sons, New York, 1975.