B565-Data Mining
Homework #4

Due on Thursday, March 2, 2023, 08:00 p.m.

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

Student Name

March 5, 2023

k-means Algorithm in Theory

This part is provided to help you implement k-means clustering algorithm.

1: **ALGORITHM** `k-means`
2: **INPUT** (data Δ, distance $d : \Delta^2 \to \mathbb{R}_{\geq 0}$, centoid number k, threshold τ)
3: **OUTPUT** (Set of centoids $\{c_1, c_2, \ldots, c_k\}$)
4:
5: $\star\star\star$ $Dom(\Delta)$ denotes domain of data.
6:
7: $\star\star\star$ Assume centroid is structure $c = (v \in DOM(\Delta), B \subseteq \Delta)$
8: $\star\star\star$ $c.v$ is the centroid value and $c.B$ is the set of nearest points.
9: $\star\star\star$ c^i means centroid at i^{th} iteration.
10:
11: $i = 0$
12: $\star\star\star$ Initialize Centroids
13: **for** $j = 1, k$ **do**
14: $\quad c_j^i.v \leftarrow random(Dom(\Delta))$
15: $\quad c_j^i.B \leftarrow \emptyset$
16: **end for**
17:
18: **repeat**
19: $\quad i \leftarrow i + 1$
20: $\quad \star\star\star$ Assign data point to *nearest* centroid
21: $\quad$ **for** $\delta \in \Delta$ **do**
22: $\quad\quad c_j^i.B \leftarrow c.B \cup \{\delta\}$, where $\min_{c_j^i}\{d(\delta, c_j^i.v)\}$
23: $\quad$ **end for**
24: $\quad$ **for** $j = 1, k$ **do**
25: $\quad\quad \star\star\star$ Get size of centroid
26: $\quad\quad n \leftarrow |c_j^i.B|$
27: $\quad\quad \star\star\star$ Update centroid with average
28: $\quad\quad c_j^i.v \leftarrow (1/n)\sum_{\delta \in c_j^i.B} \delta$
29: $\quad\quad \star\star\star$ Remove data from centroid
30: $\quad\quad c_j^i.B \leftarrow \emptyset$
31: $\quad$ **end for**
32: $\quad \star\star\star$ Calculate scalar product (abuse notation and structure slightly)
33: $\quad \star\star\star$ See notes
34: **until** $((1/k)\sum_{j=1}^{k} ||c_j^{i-1} - c_j^i||) < \tau$
35: **return** $(\{c_1^i, c_2^i, \ldots, c_k^i\})$

k-means on a tiny data set.

Here are the inputs:

$$\Delta = \{(2,5),(1,5),(22,55),(42,12),(15,16)\} \quad (1)$$

$$d((x_1,y_1),(x_2,y_2)) = [(x_1-x_2)^2 + (y_1-y_2)^2)]^{1/2} \quad (2)$$

$$k = 2 \quad (3)$$

$$\tau = 10 \quad (4)$$

Observe that $Dom(\Delta) = \mathbb{R}^2$. We now work through k-means. We ignore the uninformative assignments. We remind the reader that T means transpose.

1: $i \leftarrow 0$

2: $\star\star\star$ Randomly assign value to first centroid.
3: $c_1^0.v \leftarrow random(Dom(\Delta)) = (16, 19)$
4: $\star\star\star$ Randomly assign value to second centroid.
5: $c_2^0.v \leftarrow random(Dom(\Delta)) = (2, 5)$
6: $i \leftarrow i + 1$
7: $\star\star\star$ Associate each datum with nearest centroid
8: $c_1^1.B = \{(22, 55), (42, 12), (15, 16)\}$
9: $c_2^1.B = \{(2, 5), (1, 5)\}$
10: $\star\star\star$ Update centroids
11: $c_1^1.v \leftarrow (26.3, 27.7) = (1/3)((22, 55) + (42, 12) + (15, 16))$
12: $c_2^1.v \leftarrow (1.5, 5) = (1/2)((2, 5) + (1, 5))$
13: $\star\star\star$ The convergence condition is split over the next few lines to explicitly show the calculations
14: $(1/k)\sum_{j=1}^{k} ||c_j^{i-1} - c_j^i|| = (1/2)(||c_1^0 - c_1^1|| + ||c_2^0 - c_2^1||) = (1/2)(||\binom{2}{5} - \binom{1.5}{5}|| + ||\binom{16}{19} - \binom{26.3}{27.7}||)$
15: $= (1/2)[(\binom{.5}{0}^{\mathsf{T}}\binom{.5}{0})^{(1/2)} + (\binom{-9.7}{-8.7}^{\mathsf{T}}\binom{-9.7}{-8.7})^{(1/2)})] = (1/2)(\sqrt{.5} + \sqrt{169.7}) \sim (1/2)(13.7) = 6.9$
16: Since the threshold is met $(6.9 < 10)$, k-means stops, returning $\{(26.3, 27.7), (1.5, 5)\}$

Problem 1

For this problem we are going to use a diabetes data set collected from many US hospitals on the purpose of analyzing the factors causing readmission of diabetic patients. You can download the data from here [link]. The web-page comes with a downloadable link along with the description of the data.
Answer the following questions [**20 points**]:

1. Briefly describe this data set–what is its purpose? How should it be used? What are the kinds of data it's using?

Discussion of data

Answer here...

The dataset contains information on patients with diabetes who were admitted to 130 hospitals in the United States between 1999 and 2008. The primary purpose of the dataset is to explore features that affect readmission rates for diabetic patients.The dataset can be used for various purposes, including exploring the relationship between various features like patient condition,labtest,medications and readmission rates, developing predictive models for readmission, and evaluating the quality of care provided by hospitals. The dataset can be used to identify the relationship between various parameters and can be used by Doctors, researchers to identify the mistakes or areas of improvements or what better could have been done , so that patient is not readmitted. Also, identify some pattern or cluster in the data given all the features or identify the main features affecting the readmission. The different data types that the dataset includes are integer and object data. The dataset's features include patient demographics, admission type and source, diagnosis and procedure codes, lab tests, medications, length of stay,etc. The target variable is whether or not the patient was readmitted to the hospital within 30 days of discharge. Overall there are 101766 entries of data and 50 features/columns including the target variable.

2. Using *R/Python*, show code that answers the following questions:
   (a) How many entries are in the data set?

R/Python script

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

```
# Sample Python Script With Highlighting
import pandas as pd
df= pd.read_csv('dataset_diabetes/diabetic_data.csv')
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 50 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   encounter_id              101766 non-null  int64
 1   patient_nbr               101766 non-null  int64
 2   race                      101766 non-null  object
```

```
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  object
 5   weight                    101766 non-null  object
 6   admission_type_id         101766 non-null  int64
 7   discharge_disposition_id  101766 non-null  int64
 8   admission_source_id       101766 non-null  int64
 9   time_in_hospital          101766 non-null  int64
 10  payer_code                101766 non-null  object
 11  medical_specialty         101766 non-null  object
 12  num_lab_procedures        101766 non-null  int64
 13  num_procedures            101766 non-null  int64
 14  num_medications           101766 non-null  int64
 15  number_outpatient         101766 non-null  int64
 16  number_emergency          101766 non-null  int64
 17  number_inpatient          101766 non-null  int64
 18  diag_1                    101766 non-null  object
 19  diag_2                    101766 non-null  object
 20  diag_3                    101766 non-null  object
 21  number_diagnoses          101766 non-null  int64
 22  max_glu_serum             101766 non-null  object
 23  A1Cresult                 101766 non-null  object
 24  metformin                 101766 non-null  object
 25  repaglinide               101766 non-null  object
 26  nateglinide               101766 non-null  object
 27  chlorpropamide            101766 non-null  object
 28  glimepiride               101766 non-null  object
 29  acetohexamide             101766 non-null  object
 30  glipizide                 101766 non-null  object
 31  glyburide                 101766 non-null  object
 32  tolbutamide               101766 non-null  object
 33  pioglitazone              101766 non-null  object
 34  rosiglitazone             101766 non-null  object
 35  acarbose                  101766 non-null  object
 36  miglitol                  101766 non-null  object
 37  troglitazone              101766 non-null  object
 38  tolazamide                101766 non-null  object
 39  examide                   101766 non-null  object
 40  citoglipton               101766 non-null  object
 41  insulin                   101766 non-null  object
 42  glyburide-metformin       101766 non-null  object
 43  glipizide-metformin       101766 non-null  object
 44  glimepiride-pioglitazone  101766 non-null  object
 45  metformin-rosiglitazone   101766 non-null  object
 46  metformin-pioglitazone    101766 non-null  object
 47  change                    101766 non-null  object
 48  diabetesMed               101766 non-null  object
 49  readmitted                101766 non-null  object
dtypes: int64(13), object(37)
memory usage: 38.8+ MB

df.shape

(101766, 50)
```

There are 101766 rows/entries and 50 columns

(b) How many unknown or missing data are in the data set?

R/Python script

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

```
# Sample Python Script With Highlighting
import numpy as np
df.replace({'?':np.nan},inplace=True)
df.head()

encounter_id   patient_nbr    race      gender    age  weight    admission_type_id   dischar
0    2278392    8222157   Caucasian      Female    [0-10)    NaN  6    25   1    1    ...  No
1    149190     55629189  Caucasian      Female    [10-20)   NaN  1    1    7    3    ...  No
2    64410      86047875  AfricanAmerican     Female    [20-30)   NaN  1    1    7    2    ..
3    500364     82442376  Caucasian      Male      [30-40)   NaN  1    1    7    2    ...  No
4    16680      42519267  Caucasian      Male      [40-50)   NaN  1    1    7    1    ...  No

5 rows    50 columns

null_feature=[i for i in df.columns if df[i].isnull().sum()>=1]
print('Null features {} \n'.format(null_feature))
print('Feature \t null_count \t not_null_count')
for i in null_feature:
    print('{} \t {} \t\t {}'.format(i,df[i].isnull().sum(),df[i].count()))

Null features ['race', 'weight', 'payer_code', 'medical_specialty', 'diag_1', 'diag_2', 'dia

Feature    null_count     not_null_count
race       2273           99493
weight     98569          3197
payer_code      40256          61510
medical_specialty    49949          51817
diag_1     21        101745
diag_2     358            101408
diag_3     1423           100343

#Dropping columns with count of null values
#around the count of not null values. As they
#dont provide significant information.
df.drop(['weight','payer_code','medical_specialty'],axis=1,inplace=True)
```

There are no null entries as such in data. But some of the columns contains ? Replacing question mark with NAN. After doing that we can see the results and now this missing data needs to be handled. $'race','weight','payer_code','medical_specialty','diag_1','diag_2','diag_3$ Columns contains null data. Dropping the columns where the count of null data is almost around 50 percent as they dont provide significant information.

(c) Create histograms for attributes {age, num_lab_procedures, num_medications}. Label the plots properly. Discuss the distribution of values *e.g.*, are uniform, skewed, normal. Place images

of these histograms into the document. Show the Python or R code that you used below and discussion below that.

R/Python script

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

```python
# Sample Python Script With Highlighting
sns.histplot(df['age'],bins=10)
plt.xticks(rotation=90)
plt.title('Age Class vs the count ')
plt.show()

##In the graph, at x axis we see the range. Lets convert the range to numbers by taking
##mean of each range
df['age'] = df['age'].replace({'[0-10)': 5, '[10-20)': 15, '[20-30)': 25, '[30-40)': 35, '[4

sns.histplot(df['age'],bins=10)
plt.xticks(rotation=90)
plt.title('Age Class vs the count ')
plt.show()

sns.histplot(df['num_lab_procedures'],bins=10,kde=True)
plt.xticks(rotation=90)
plt.title('Number of Lab procedures and their count ')
plt.show()

sns.histplot(df['num_medications'],kde=True,bins=5)
plt.xticks(rotation=90)
plt.title('Number of Medications and their count ')
plt.show()
```

Discussion of Findings

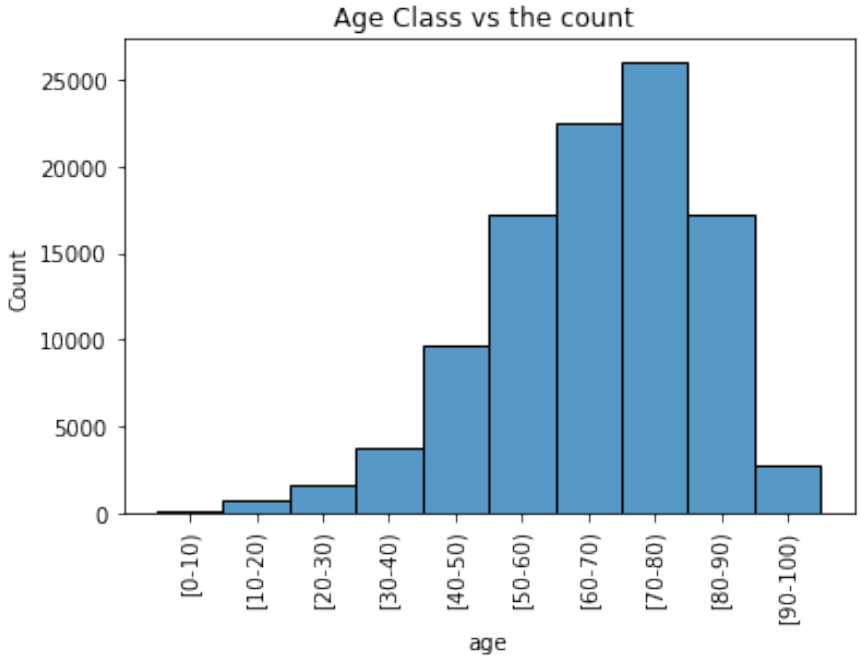
Answer here

The histogram plot of the age feature suggests that the data is skewed towards higher age, while the histogram plot of the `num_lab_procedures` feature suggests that the data is skewed and bimodal. Furthermore, the histogram plot of the `num_medication` feature indicates that the data is normally distributed and right-tailed.

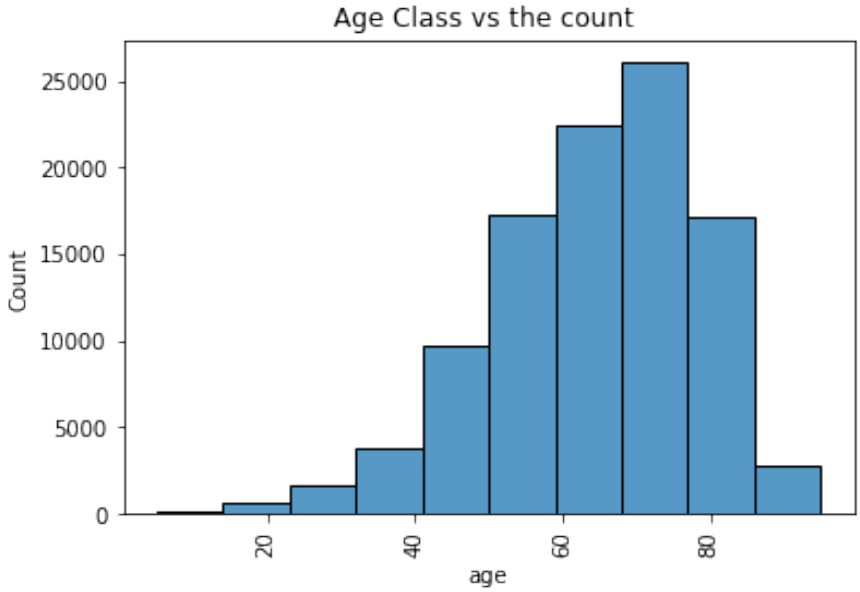
...

Plots

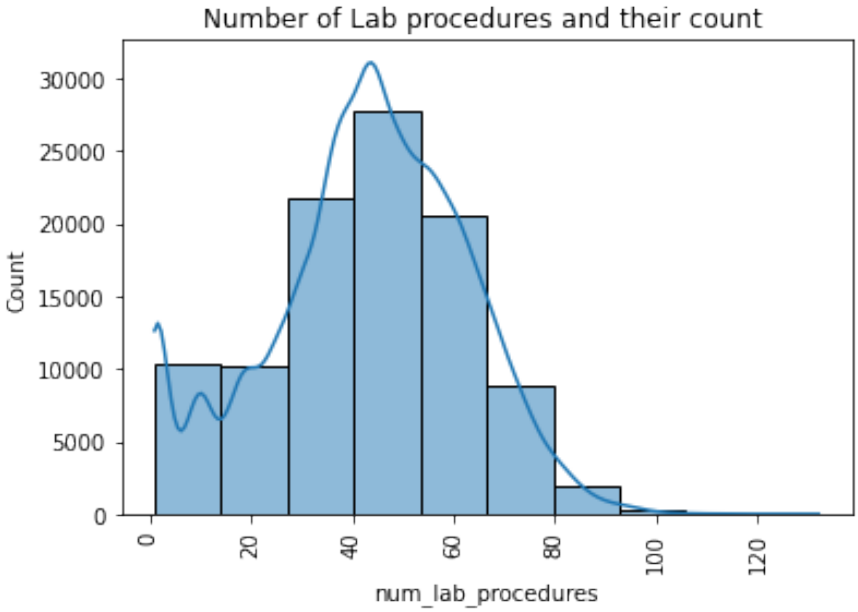
Place images here with suitable captions.


Age Class vs the count
Count
25000
20000
15000
10000
5000
0
[0-10)
[10-20)
[20-30)
[30-40)
[40-50)
[50-60)
[60-70)
[70-80)
[80-90)
[90-100)
age

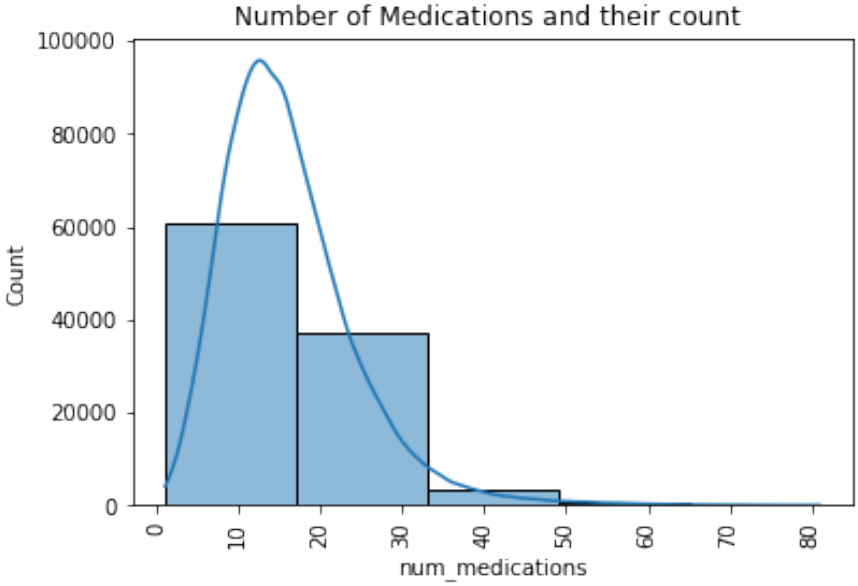
(5)


Age Class vs the count
Count
25000
20000
15000
10000
5000
0
20
40
60
80
age

(6)



(7)



(8)

3. Make a scatter plots of [`time_in_hospital`, `num_medications`] and [`num_medications`, `num_lab_procedures`] variables and color the data points with the class variable [`readmitted`]. Discuss the plots, i.e., do you observe any relationships between variables? Show the Python or R code that you used below and discussion below that.

R/Python script

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

```
# Sample Python Script With Highlighting


sns.scatterplot(data=df, x='time_in_hospital', y='num_medications',hue='readmitted')
plt.xticks(rotation=90)
plt.title('Number of Medications Vs time_in_hospital ')
plt.show()

##Changing the readmitted column by making the target variable as binary class having two values
##For No-0 and for yes i.e >30 or < 30 - 1

df['readmitted'] = df['readmitted'].replace({'>30':1,'<30':1,'NO':0})

sns.scatterplot(data=df, x='time_in_hospital', y='num_medications',hue='readmitted')
plt.xticks(rotation=90)
plt.title('Number of Medications Vs time_in_hospital ')
plt.show()

sns.scatterplot(data=df, x='num_medications', y='num_lab_procedures', hue='readmitted')
plt.xticks(rotation=90)
plt.title('Number of Medications Vs Number of Lab procedures ')
plt.show()

df[['time_in_hospital', 'num_medications','num_lab_procedures','readmitted']].corr()

time_in_hospital    num_medications     num_lab_procedures  readmitted
time_in_hospital    1.000000  0.466135  0.318450  0.051289
num_medications     0.466135  1.000000  0.268161  0.046772
num_lab_procedures  0.318450  0.268161  1.000000  0.039253
readmitted     0.051289  0.046772  0.039253  1.000000
```

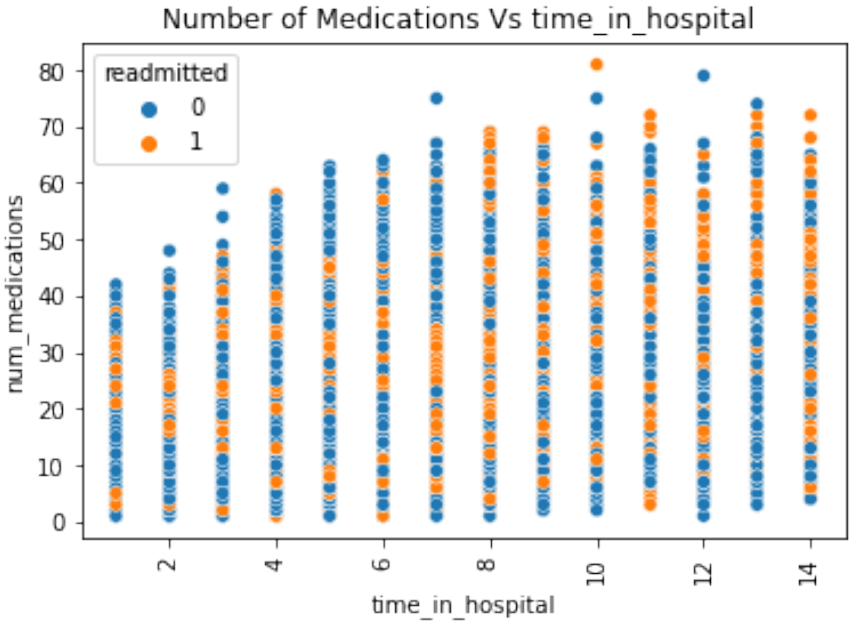
Discussion of Findings

Answer here...

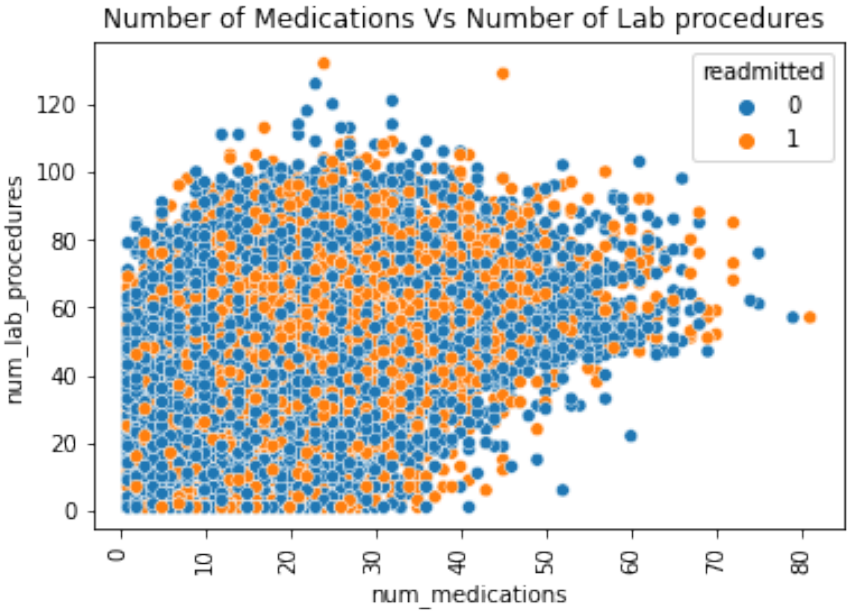
The scatter plot of `time_in_hospital` and `num_medications` shows a positive relationship between the two variables, which is also supported by the correlation matrix. The scatter plot also suggests that the readmitted variable does not have a strong relationship with either variable.
Similarly, the scatter plot of `num_medications` and `num_lab_procedures` shows a weak positive relationship between the two variables, which is also supported by the correlation matrix. The scatter plot also suggests that the readmitted variable does not have a strong relationship with either variable.

Plots

Place images here with suitable captions.



(9)



(10)

Problem 2

The pseudo-code for k-means and a running example of k-means on a small data set are provided above. Answer the following questions [**10 points**]:

2.1 Does k-means always converge? Given your answer, a bound on the iterate must be included. How is its value determined?

No, kmeans donot converge always to global minima. The k-means algorithm is a popular clustering algorithm that aims to minimize the sum of squared distances between data points and their assigned centroids. However, it may converge to a local minimum rather than the global minimum due to its greedy nature. There can be various other reasons for the same such as noise or outliers in data or high dimensionality in data or poor centroid initialization. Yes, bound on the iterate must be included to have a balance between computational efficiency and convergence accuracy. The algorithm should stop sometime and not continue indefinitely. To perform the clustering, the algorithm assigns each data point to the closest cluster centroid and recalculates the centroids based on the new assignments. The stopping condition can be the maximum number of iterations or when error becomes constant and same centroids are generated. In our example it is τ, when the scalar product between clusters is less than τ , the k means stops or the stopping conditon is met.The choice of the stopping criterion is very important and is determined by user i.e it is a user defined parameter. Various factors can be considered while deciding the value of stopping condition such as time to run the code, computational limitations, expected accuracy,etc. The k-means algorithm is sensitive to the initial choice of centroids, and different initializations may result in different local optima. Therefore, it is recommended to run the algorithm multiple times with different initializations and choose the best clustering based on the stopping criterion and any domain knowledge about the expected number of cluster or stopping condition.

2.2 What is the run-time of this algorithm?

The time complexity is $O(n * d * k * i)$
Here
n: the number of data points
d: the number of dimensions
k: the number of clusters
i: the number of iterations till stopping condition is met

Problem 3

Implement Lloyd's algorithm for k-means (see algorithm k-means above) in R or *Python* and call this program C_k. As you present your code explain your protocol for [**20 points**]

1. initializing centroids
2. maintaining k centroids
3. deciding ties
4. stopping criteria

R/Python Code

subsectionR/Python script

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

```
# Sample Python Script With Highlighting
import pandas as pd
import numpy as np
```

```
import swifter
from scipy.spatial.distance import euclidean


def initialize_centroids(df, k):
    """
    Function to initialize random centroids from dataset.
    Input:
        - df: pandas dataframe with the data
        - k: integer number of clusters
    Output:
        - temp_df: pandas dataframe with the centroids as columns and index as label
    """
    centroids = []
    for i in range(k):
        centroids.append(df.apply(lambda x: float(x.sample())))
        # Take a random sample from each column to create a centroid
    centroids = pd.concat(centroids, axis=1)
    centroids.index.name = 'Label'

    return centroids


def assign_labels(df, centroids):
    """
    Function to calculate the closest centroid label for each row in a dataframe.
    Input:
        - df: pandas dataframe with the data
        - centroids: pandas dataframe with the centroids as columns and index as label
    Output:
        - distances.idxmin(axis=1): pandas series with the label of the closest centroid for each rou
    """
    distances = centroids.swifter.apply(lambda x: np.sqrt(((df - x) ** 2).sum(axis=1)))
    # Calculate the Euclidean distance between each row in df and each centroid
    return distances.idxmin(axis=1)
    # Get the index of the minimum distance, which corresponds to the label of the closest centroid


def new_centroids(df_label, df1):
    """
    Function to calculate the new centroids based on the current labels of the rows.
    Input:
        - df_label: pandas series with the label
        of the closest centroid for each row in df1
        - df1: pandas dataframe with the data
    Output:
        - new_centroids.T: pandas dataframe with
        the new centroids as columns and index as feature name
    """
    joined_df = df1.join(df_label)
    joined_df.rename(columns={0: 'Label'}, inplace=True)
    # Rename the column with the label
    # Calculate the mean of the rows with the same label
```

```
    return joined_df.groupby('Label').mean().T
    # Transpose the dataframe to have the new centroids as columns and index as feature name


def kmeans_lyod(df1, k, tou):
    """
    Function to run the K-means Lloyd algorithm.
    Input:
        - df1: pandas dataframe with the data
        - k: integer number of clusters
        - tou: float tolerance level to stop the algorithm
    Output:
        - centroids: pandas dataframe with the final centroids as columns and index as label
    """

    centroids = initialize_centroids(df1, k) # Initialize random centroids
    initial_list_of_columns = centroids.columns.to_list()
    iteration = 0
    while True:
        # Assign labels to current centroids
        df_label = assign_labels(df1, centroids)
        df_label = pd.DataFrame(df_label)
        # Calculate new centroids
        df_new_centroids = new_centroids(df_label, df1)
        new_list_of_columns = df_new_centroids.columns.to_list()
        # Keep the number of clusters the same i.e maintain same k
        for i in initial_list_of_columns:
            if i not in new_list_of_columns:
                df_new_centroids[i] = centroids[i]
        # Calculate tao
        distance = []
        for col in centroids.columns:
            col_distance = euclidean(centroids[col], df_new_centroids[col])
            distance.append(col_distance)
        tao_calculated=sum(distance)/k #Used the formula provided for calculating Tao


        if iteration>100:
            print("Iteration exceeded")

        if tao_calculated<tou or iteration >100:
        #if the convergence is met, kmeans will stop  or else if the convergence is never met, after
            return df_new_centroids
            break                                   # otherwise indefinite loop
        else:
            centroids= df_new_centroids
            # In case we need more iterations, the centroids calculated at this step acts as input
        iteration+=1
```

Discussion of Initialization of Centroids

Answer here...

Initializing the centroids marks the first step. There can be multiple ways to initialize the centroids such as Random selection from the overall search space. Random selection from the domain of the data. We have initialized the centroids by using domain approach. We have randomly selected the cluster centroid from the domain. Value of eachco ordinate of the centroid is taken randomly from the domain of that feature which contributes to that feature. For example if the domain values of a feature is between 1 and 100, we have randomly chosen a value between 1 and 100. This helps in initializing the centroid co ordinate value whihc is close to the actual dataset.

Discussion of Maintaining k Centroids

Answer here...

In kmeans we need to maintain the k centroids at each and every iteration.There are many reasons we may get an empty cluster.Various affecting factors are: Sometimes due to the random initialization of clusters Quality of Data Choice of number of clusters It is possible that in a particular iteration, no data point is mapped to a cluster. We may lose out on that cluster. So to avoid that we have used the co ordinates of previous iteration to maintain same number of clusters throughout. But if this continues in many iterations than it can be an indication of merging the nearest clusters with empty cluter or changing the number of cluster i.e k value whihc is decided in the start.

Discussion of Deciding Ties

Answer here...

K means is distance based greedy algorithm. It calculates the distance between each data point and various clusters. The minimum distance is selected, but there is high chance that there is tie between minimum distance of cluster and datapoints. There could be various techniques to handle such scenario, which include: Random tie breaking - Any cluster out of the cluters at same distance is selected randomly. Least assigned Cluster- In case 2 cluster are into tie break, we can check the datapoints assigned to each clusters, and whichever has low data points that can be selected. This helps in removing imbalance between clusters Highest assigned cluster- This is opposite of least assigned cluster method, here the cluster which is assigned more datapoints is selected. As from the previous iteration, it has high probability to get selected to maintain the number of points assigned. Selection of both the techniques highly depends upon use case, domain knowledge and expected results.

Discussion of Stopping Criteria

Answer here...

Deciding the stopping criteria is an important step in kmeans algorithm. There are various approaches to stop kmeans, which includes: 1) Maximum iterations are reached 2) The cluster centroids are not changing 3) Convergence or global minima is reached 4) The sum of square error becomes constant over the iterations 5) The magnitude of clusters (scalar product) becomes less than a threshold (τ). We have used the last approach. Here the distance between current centroid and previous centroid for consequtive iterations is calculated , squared. This is done for all the clusters and the distance is added and finally divided by k. When this value becomes less than threshold i.e τ the k means algorithm stops.

Problem 4

In this question, you are asked to run your program, C_k, against the Diabetes data set from Problem 1 and New York Times Comments data set [link](use the file `nyt-comments-2020.csv` as your data set). Upon

stopping, you will calculate the quality of the centroids and of the partition. For each centroid c_i, form two counts:

$$y_i \leftarrow \sum_{\delta \in c_i.B} [\delta.C = \text{“y”}], \quad \text{readmitted/selected}$$
$$n_i \leftarrow \sum_{\delta \in c_i.B} [\delta.C == \text{“n”}], \quad \text{not readmitted/selected}$$

where $[x = y]$ returns 1 if True, 0 otherwise. For example, $[2 = 3] + [0 = 0] + [34 = 34] = 2$
The centroid c_i is classified as readmitted/selected if $y_i > n_i$ and not readmitted/selected otherwise. We can now calculate a simple error rate. Assume c_i is good. Then the error is:

$$error(c_i) = \frac{n_i}{n_i + y_i}$$

We can find the total error rate easily:

$$Error(\{c_1, c_2, \ldots, c_k\}) = \sum_{i=1}^{k} error(c_i)$$

Report the total error rates for $k = 2, \ldots 5$ for 20 runs each for both data sets. Presenting the results that are easily understandable. Plots are generally a good way to convey complex ideas quickly, i.e., box plot. Discuss your results.
Note: The error calculation method mentioned above is generalized for both data sets where the first data set asks if a patient was readmitted or not and the second data set asks if a comment was selected by editor or not.
Data preparation: Like any other data mining problem, before feeding your data to the clustering algorithms, you will have to perform data cleaning, feature engineering, and feature selection on both data sets. For the second data set (NY Times Comments data set), you will be using `commentBody` column as your input feature (you have to build a corpus and perform an appropriate vector-space representation technique) [**20 points**].

R/Python script

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

```
# Sample Python Script With Highlighting
#EDA Diabetes Dataset

df.columns

'''
Index(['encounter_id', 'patient_nbr', 'race', 'gender', 'age',
       'admission_type_id', 'discharge_disposition_id', 'admission_source_id',
       'time_in_hospital', 'num_lab_procedures', 'num_procedures',
       'num_medications', 'number_outpatient', 'number_emergency',
       'number_inpatient', 'diag_1', 'diag_2', 'diag_3', 'number_diagnoses',
       'max_glu_serum', 'A1Cresult', 'metformin', 'repaglinide', 'nateglinide',
       'chlorpropamide', 'glimepiride', 'acetohexamide', 'glipizide',
       'glyburide', 'tolbutamide', 'pioglitazone', 'rosiglitazone', 'acarbose',
       'miglitol', 'troglitazone', 'tolazamide', 'examide', 'citoglipton',
       'insulin', 'glyburide-metformin', 'glipizide-metformin',
       'glimepiride-pioglitazone', 'metformin-rosiglitazone',
```

```
       'metformin-pioglitazone', 'change', 'diabetesMed', 'readmitted'],
      dtype='object')
'''


for i in df.select_dtypes(include=['int', 'float']).columns.to_list():
    print('The numeric feature is {} \n The value counts are {}'.format(i,df[i].value_counts()) )

'''
The numeric feature is encounter_id
 The value counts are 2278392      1
190792044    1
190790070    1
190789722    1
190786806    1
            ..
106665324    1
106657776    1
106644876    1
106644474    1
443867222    1
Name: encounter_id, Length: 101766, dtype: int64
The numeric feature is patient_nbr
 The value counts are 88785891     40
43140906     28
1660293      23
88227540     23
23199021     23
            ..
11005362      1
98252496      1
1019673       1
13396320      1
175429310     1
Name: patient_nbr, Length: 71518, dtype: int64
The numeric feature is age
 The value counts are 75    26068
65    22483
55    17256
85    17197
45     9685
35     3775
95     2793
25     1657
15      691
5       161
Name: age, dtype: int64
The numeric feature is admission_type_id
 The value counts are 1    53990
3    18869
2    18480
6     5291
5     4785
```

```
8      320
7       21
4       10
Name: admission_type_id, dtype: int64
The numeric feature is discharge_disposition_id
 The value counts are 1     60234
3     13954
6     12902
18     3691
2      2128
22     1993
11     1642
5      1184
25      989
4       815
7       623
23      412
13      399
14      372
28      139
8       108
15       63
24       48
9        21
17       14
16       11
19        8
10        6
27        5
12        3
20        2
Name: discharge_disposition_id, dtype: int64
The numeric feature is admission_source_id
 The value counts are 7     57494
1     29565
17     6781
4      3187
6      2264
2      1104
5       855
3       187
20      161
9       125
8        16
22       12
10        8
14        2
11        2
25        2
13        1
Name: admission_source_id, dtype: int64
The numeric feature is time_in_hospital
 The value counts are 3     17756
```

```
2     17224
1     14208
4     13924
5      9966
6      7539
7      5859
8      4391
9      3002
10     2342
11     1855
12     1448
13     1210
14     1042
Name: time_in_hospital, dtype: int64
The numeric feature is num_lab_procedures
 The value counts are 1      3208
43     2804
44     2496
45     2376
38     2213
       ...
120       1
132       1
121       1
126       1
118       1
Name: num_lab_procedures, Length: 118, dtype: int64
The numeric feature is num_procedures
 The value counts are 0    46652
1    20742
2    12717
3     9443
6     4954
4     4180
5     3078
Name: num_procedures, dtype: int64
The numeric feature is num_medications
 The value counts are 13    6086
12    6004
11    5795
15    5792
14    5707
      ...
70       2
75       2
81       1
79       1
74       1
Name: num_medications, Length: 75, dtype: int64
The numeric feature is number_outpatient
 The value counts are 0     85027
1      8547
2      3594
```

```
3       2042
4       1099
5        533
6        303
7        155
8         98
9         83
10        57
11        42
13        31
12        30
14        28
15        20
16        15
17         8
21         7
20         7
18         5
22         5
19         3
27         3
24         3
26         2
23         2
25         2
33         2
35         2
36         2
29         2
34         1
39         1
42         1
28         1
37         1
38         1
40         1
Name: number_outpatient, dtype: int64
The numeric feature is number_emergency
 The value counts are 0     90383
1       7677
2       2042
3        725
4        374
5        192
6         94
7         73
8         50
10        34
9         33
11        23
13        12
12        10
22         6
```

```
16        5
18        5
19        4
20        4
15        3
14        3
25        2
21        2
28        1
42        1
46        1
76        1
37        1
64        1
63        1
54        1
24        1
29        1
Name: number_emergency, dtype: int64
The numeric feature is number_inpatient
 The value counts are 0     67630
1     19521
2      7566
3      3411
4      1622
5       812
6       480
7       268
8       151
9       111
10       61
11       49
12       34
13       20
14       10
15        9
16        6
19        2
17        1
21        1
18        1
Name: number_inpatient, dtype: int64
The numeric feature is number_diagnoses
 The value counts are 9     49474
5     11393
8     10616
7     10393
6     10161
4      5537
3      2835
2      1023
1       219
16       45
```

```
10       17
13       16
11       11
15       10
12        9
14        7
Name: number_diagnoses, dtype: int64
The numeric feature is readmitted
 The value counts are 0    54864
1    46902
Name: readmitted, dtype: int64


Here encounter_id, patient_nbr are the unique ids and wont help in clustering as they contain
unique data and we cant forms groups. We can drop them

'''

df.drop(columns=['encounter_id','patient_nbr'],inplace=True,axis=1)

print(df.select_dtypes(include=['object']).columns.to_list())

'''
['race', 'gender', 'diag_1', 'diag_2', 'diag_3', 'max_glu_serum', 'A1Cresult', 'metformin', 'repaglin
'glyburide', 'tolbutamide', 'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone',
'citoglipton', 'insulin', 'glyburide-metformin', 'glipizide-metformin', 'glimepiride-pioglitazone',

Object columns include majority of medications.
'''



meds=['max_glu_serum', 'A1Cresult', 'metformin', 'repaglinide', 'nateglinide', 'chlorpropamide',
      'glimepiride', 'acetohexamide', 'glipizide', 'glyburide', 'tolbutamide',
      'pioglitazone', 'rosiglitazone',
      'acarbose', 'miglitol', 'troglitazone',
      'tolazamide', 'examide', 'citoglipton', 'insulin', 'glyburide-metformin',
      'glipizide-metformin', 'glimepiride-
      pioglitazone', 'metformin-rosiglitazone', 'metformin-pioglitazone']
df_value_counts=pd.DataFrame()
for i in meds:
    value_counts=df[i].value_counts()
    percent = []
    for j in value_counts.index:
        percent.append(value_counts[j] *100/ len(df))
    ## PErcentage dataframe to store the feature, its unique values, the count
    and the percentage
    df_temp=pd.DataFrame({'Feature':i,'Value': value_counts.index, 'Count': value_counts.values, 'Per
    df_value_counts=pd.concat([df_value_counts,df_temp],ignore_index=True)
df_value_counts.head(80)

'''
Feature   Value     Count     Percentage
0    max_glu_serum  None      96420     94.746772
```

```
1    max_glu_serum  Norm      2597      2.551933
2    max_glu_serum  >200      1485      1.459230
3    max_glu_serum  >300      1264      1.242065
4    A1Cresult      None      84748     83.277322
...  ...  ...  ...  ...
74   glimepiride-pioglitazone      Steady    1    0.000983
75   metformin-rosiglitazone  No   101764    99.998035
76   metformin-rosiglitazone  Steady    2    0.001965
77   metformin-pioglitazone   No   101765    99.999017
78   metformin-pioglitazone   Steady    1    0.000983

79 rows    4 columns

From the above dataframe it can be seen that some of the features have data of almost one type.
To make clusters we require data of all types in a column as that will help identifying
differences or similarity in the data
If all the data is of one type, it is not useful
for clustering. Hence removing columns with Percentage > 95.

'''



skewed_data=df_value_counts[df_value_counts['Percentage']>95]['Feature'].to_list()
df.drop(columns=skewed_data,inplace=True)

'''
Performing label encoding on the categorical columns
'''


label_encoded_columns=[]
for i in df.select_dtypes(include=['object']).columns.to_list():
    if i not in skewed_data:
        label_encoded_columns.append(i)
## The columns remaining after all the EDA that are to be label encoded

df_cleaned_dia=df.copy()
## creating the copy of data before performing label encoding and one hot encoding
one_hot = pd.get_dummies(df_cleaned_dia[['gender','race']])
label_encoded_columns.remove('diag_1')
label_encoded_columns.remove('diag_2')
label_encoded_columns.remove('diag_3')
label_encoded_columns.remove('gender')
label_encoded_columns.remove('race')
# combine the one-hot encoded columns with the original dataframe
df_cleaned_dia = pd.concat([df_cleaned_dia, one_hot], axis=1)
df_cleaned_dia.drop(columns=['diag_1','diag_2','diag_3','gender','race'],inplace=True)
##
df_cleaned_dia[label_encoded_columns]=df_cleaned_
dia[label_encoded_columns].swifter.apply(LabelEncoder().fit_transform)


'''
```

```
Have performed one hot encoding for race and gender as the columns are not ordinal.
For rest of the columns performed label encoding
. The column diag1,diag2, diag3 are been dropped.
These 3 columns are categorical columns and in k means the categorical columns are not highly
preferred for clustering. Also, these columns
contains around 700 unique classes, so intotal more than 2100 columns will become the part of
dataset, this will cause the clustering around
diag1,diag2, diag3 and rest columns will get shadowed.
There is high chance of model getting overfit due to these categorical data and will increase
the dimensions and the curse of dimensionality comes into picture.Hence dropping these columns.
'''

df_cleaned_dia.shape

'''
(101766, 32)

'''




import time
def initialize_centroids(df, k):
    """
    Function to initialize random centroids from dataset.
    Input:
       - df: pandas dataframe with the data
        - k: integer number of clusters
    Output:
        - temp_df: pandas dataframe with the centroids as columns and index as label
    """
    centroids = []
    for i in range(k):
        centroids.append(df.apply(lambda x: float(x.sample())))
        # Take a random sample from each column to create a centroid
    centroids = pd.concat(centroids, axis=1)
    centroids.index.name = 'Label'

    return centroids


def assign_labels(df, centroids):
    """
    Function to calculate the closest centroid label for each row in a dataframe.
    Input:
        - df: pandas dataframe with the data
        - centroids: pandas dataframe with the centroids as columns and index as label
    Output:
        - distances.idxmin(axis=1): pandas
        series with the label of the closest centroid for each row in df
    """
    distances = centroids.swifter.apply(lambda x: np.sqrt(((df - x) ** 2).sum(axis=1)))
```

```
    # Calculate the Euclidean distance between each row in df and each centroid
    return distances.idxmin(axis=1) # Get the
    index of the minimum distance, which corresponds to the label of the closest centroid


def new_centroids(df_label, df1):
    """
    Function to calculate the new centroids based on the current labels of the rows.
    Input:
        - df_label: pandas series with the label
        of the closest centroid for each row in df1
        - df1: pandas dataframe with the data
    Output:
        - new_centroids.T: pandas dataframe with
        the new centroids as columns and index as feature name
    """
    joined_df = df1.join(df_label)
    joined_df.rename(columns={0: 'Label'}, inplace=True)
    # Rename the column with the label
    # Calculate the mean of the rows with the same label
    return joined_df.groupby('Label').mean().T
    # Transpose the dataframe to have the new centroids as columns and index as feature name




def error_clusters(df_new_centroids,df1,df_label):
    """
    Calculate the error rate of each cluster.

    Args:
    - df_label (pandas.DataFrame): the label of
    the nearest centroid for each data point.
    - df1 (pandas.DataFrame): the dataset.
    - df_new_centroids (pandas.DataFrame): The
    new centroids computed in the current iteration.

    Returns:
    - error_rate (float): the total error rate
    of all clusters.
    """



    #Calculate mean value
    mean_centroid=df1.groupby('readmitted').mean(
    ).reset_index()
    # Transpose the new centroids dataframe and reset the index
    new_centroids= df_new_centroids.T
    # Get the columns of the data dataframe
    columns = df1.columns

    sse = []
    # Compute the distance between each data
```

```
    point and its assigned centroid
    for i in range(len(new_centroids)):
    #### centroid
        s=[]
        for j in range(len(mean_centroid)):
        ### mean centroid
        # Compute the distance between each data
        point and its assigned centroid
            distance = np.sum(np.square(mean_centroid[mean_c
            entroid['readmitted']==j][columns] - new_centroids.iloc[i][columns]),
            axis=1)
            s.append(distance.iloc[0])
        sse.append(s)
    ## key  is the cluster number and value is the merged value

    merge_label=pd.DataFrame(sse).idxmin(axis=1).to_dict()
    ## Merging cluster based on the target variable
    df_label[0]=df_label[0].replace(merge_label)

    df1 = df1.join(df_label) # add the label column to the dataset
    df1.rename(columns={0: 'Label'}, inplace=True) # rename the label column
    error_list = []
    for i in df1['Label'].value_counts().index:
        df_cluster = df1[df1['Label'] == i] #
        filter the dataset to include only the data points in the current cluster
        y =
        len(df_cluster[df_cluster['readmitted']
        == 1]) # count the number of data points
        in the current cluster that were readmitted
        n = len(df_cluster[df_cluster['readmitted']
        == 0]) # count the number of data points in the current cluster that were not
        readmitted
        if y == 0 and n == 0:
            error = 0
        else:
            error = n / (n + y) # calculate the error rate of the current cluster
        error_list.append(error)
    return round(sum(error_list),4)


def sum_of_square_error(new_centroids, data, labels):
    """
    Computes the sum of squared errors between
    the data points and their assigned centroids.

    Args:
    new_centroids (DataFrame): The new centroids computed in the current iteration.
    data (DataFrame): The input data points.
    labels (DataFrame): The labels assigned to each data point.

    Returns:
    The sum of squared errors.
    """
```

```
    # Transpose the new centroids dataframe and
    reset the index
    new_centroids = new_centroids.T.reset_index()
    # Get the columns of the data dataframe
    columns = data.columns
    # Join the data dataframe and the labels dataframe
    data = data.join(labels)
    # Rename the '0' column of the labels dataframe to 'Label'
    data.rename(columns={0:'Label'}, inplace=True)
    sse = []
    # Compute the distance between each data
    point and its assigned centroid
    for i in range(len(new_centroids)):
        distance =
        np.sum(np.square(data[data['Label']==i]
        [columns] - new_centroids.iloc[i][columns]), axis=1)
        sse.append(sum(distance))
    # Return the sum of squared errors
    return sum(sse)

def kmeans_lyod_with_error(df1, k, tou):
    """
    Function to run the K-means Lloyd algorithm.
    Input:
        - df1: pandas dataframe with the data
        - k: integer number of clusters
        - tou: float tolerance level to stop the algorithm
    Output:
        - centroids: pandas dataframe with the
        final centroids as columns and index as label
    """
    start_time=time.time()
    centroids = initialize_centroids(df1, k)
    # Initialize random centroids
    initial_list_of_columns = centroids.columns.to_list()
    iteration = 0
    while True:
        # Assign labels to current centroids
        df_label = assign_labels(df1, centroids)
        df_label = pd.DataFrame(df_label)
        # Calculate new centroids
        df_new_centroids = new_centroids(df_label, df1)
        new_list_of_columns = df_new_centroids.columns.to_list()
        # Keep the number of clusters the same i.e maintain same k
        for i in initial_list_of_columns:
            if i not in new_list_of_columns:
                df_new_centroids[i] = centroids[i]
        # Calculate tao
        distance = []
        for col in centroids.columns:
            col_distance =
            euclidean(centroids[col], df_new_centroids[col])
            distance.append(col_distance)
```

```
        tao_calculated=sum(distance)/k #Used the formula provided for calculating Tao
        sse =
        sum_of_square_error(df_new_centroids, df1, df_label)
        #error=error_clusters(df_label,df1,k)
        end_time= time.time()
        if iteration>100:
            error=error_clusters(df_new_centroids,df1,df_label)
            print("Iteration exceeded")

            return error, sse,end_time-start_time
            break

        if tao_calculated<tou or iteration >100:
        #if the convergence is met, kmeans will stop  or else if the convergence is
        never met, after 100 iteration code will stop
            error=error_clusters(df_new_centroids,df1,df_label)
            return error, sse,end_time-start_time
            break                                   # otherwise indefinite loop
        else:
            centroids= df_new_centroids # In
            case we need more iterations, the
            centroids calculated at this step acts as input
        iteration+=1


error_matrix_diab=[]
for i in range(2,6):
    for j in range(1,21):
        error,sse,run_time=kmeans_lyod_with_error(df_cleaned_dia,i,10)

        error_matrix_diab.append([i,j,error,sse,run_time])
error_df_diab= pd.DataFrame(error_matrix_diab,columns=
['number_of_cluster', 'iteration', 'error','sse','run_time'])
error_df_diab

'''
number_of_cluster   iteration      error     sse  run_time
0    2    1    1.0691    5.696255e+07   2.029050
1    2    2    1.0849    5.365373e+07   2.117899
2    2    3    1.0949    5.491554e+07   1.994823
3    2    4    1.0988    5.544659e+07   2.155761
4    2    5    1.0819    5.413536e+07   2.040962
...  ...  ...  ...  ...  ...
75   5    16   2.7250    3.783901e+07   2.028614
76   5    17   2.7254    3.225760e+07   3.838290
77   5    18   2.6982    3.359654e+07   3.796923
78   5    19   2.7071    3.280613e+07   3.839000
79   5    20   2.7028    4.462725e+07   2.021136

80 rows    5 columns

'''
```

```
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
x=error_df_diab['number_of_cluster'].value_counts().index
y1=error_df_diab.groupby(['number_of_cluster']).mean()['error']
ax1.plot(x, y1, color='tab:blue')
ax1.set_xlabel('Number of clusters')
ax1.set_ylabel('Error', color='tab:blue')

ax2 = ax1.twinx()
y2 = error_df_diab.groupby(['number_of_cluster']).mean()['sse']
ax2.plot(x, y2, color='tab:orange')
ax2.set_ylabel('SSE', color='tab:orange')
plt.title('Error and SSE (mean)for each cluster tao as convergence')
plt.xticks(range(2, 6))
plt.show()



'''
The first y axis is to plot error using formula
provided and the second y axis is to show sum of square error .
The values for y axis have been averaged out for each cluster.
The x axis is the number of clusters i.e 2,3,4,5.
'''

import seaborn as sns
plt.figure(figsize=(6, 10))
sns.boxplot(x=error_df_diab['number_of_cluster'],y=error_df_diab['error'])
plt.title('Box Plot of runtime for K means llyod')
plt.show()


import seaborn as sns
sns.boxplot(x=error_df_diab['number_of_cluster'],y=error_df_diab['run_time'])
plt.title('Box Plot of runtime for K means llyod')
plt.show()



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
import swifter
```

```
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')

# Load the data
comments_df = pd.read_csv('nyt-comments-
2020.csv', low_memory=False)

# Preprocessing function
def preprocess(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove non-alphabetic characters and punctuations
    text = re.sub(r'[^a-zA-Z\s]', '',
    text.translate(str.maketrans('', '', string.punctuation)))
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]
    # Stem the tokens
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(token) for token in tokens]
    # Join the tokens back into a string
    text = ' '.join(tokens)
    return text

# Apply preprocessing to the comment body column using Swifter for faster processing
comments_df['cleaned_comment'] = comments_df['commentBody'].swifter.apply(lambda x:preprocess(x))
# Save the cleaned data and bag of words matrix to CSV files
comments_df.to_csv('cleaned_data.csv', index=False)




comments_df = pd.read_csv('cleaned_data.csv', low_memory=False)
# New york Dataset

# Remove rows with missing values

comments_df=comments_df[['cleaned_comment','editorsSelection']]
comments_df['cleaned_comment'].replace({'':np.nan},inplace=True)
comments_df.dropna(inplace=True,axis=0)
comments_df.shape
'''
(4985131, 2)
'''

# # Create a bag of words matrix using CountVectorizer with a minimum document
```

```
frequency of 2.5%

vectorizer = CountVectorizer(min_df=int(0.025 * len(comments_df)))
bag_of_words_matrix = vectorizer.fit_transform(tqdm(comments_df['cleane
d_comment']))
bag_of_words_df =
pd.DataFrame.sparse.from_spmatrix(bag_of_words_matrix, columns=vectorizer.get_feature_names())

# Save bag of words matrix to CSV file
bag_of_words_df.to_csv('bag_of_words.csv', index=False)
bag_of_words_df.shape

'''
(4985131, 208)
'''



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
import swifter

bag_of_words_df=pd.read_csv('bag_of_words.csv')
comments_df=pd.read_csv('cleaned_data.csv')

bag_of_words_df=bag_of_words_df.join(comments_df['editorsSelection'].replace({True:1,False:0}))

bag_of_words_df.columns

'''
Index(['actual', 'administr', 'agre', 'allow', 'alreadi', 'also', 'alway',
       'america', 'american', 'anoth',
       ...
       'without', 'wonder', 'word', 'work', 'world', 'would', 'ye', 'year',
       'yet', 'editorsSelection'],
      dtype='object', length=209)
'''



import time
from scipy.spatial.distance import euclidean
def initialize_centroids(df, k):
    """
```

```
    Function to initialize random centroids from dataset.
    Input:
        - df: pandas dataframe with the data
        - k: integer number of clusters
    Output:
        - temp_df: pandas dataframe with the centroids as columns and index as label
    """
    centroids = []
    for i in range(k):
        centroids.append(df.apply(lambda x:
        float(x.sample()))) # Take a random
        sample from each column to create a centroid
    centroids = pd.concat(centroids, axis=1)
    centroids.index.name = 'Label'

    return centroids


def assign_labels(df, centroids):
    """
    Function to calculate the closest centroid label for each row in a dataframe.
    Input:
        - df: pandas dataframe with the data
        - centroids: pandas dataframe with the centroids as columns and index as label
    Output:
        - distances.idxmin(axis=1): pandas series with the label of the closest centroid for each rou
    """
    distances = centroids.swifter.apply(lambda
    x: np.sqrt(((df - x) ** 2).sum(axis=1)))
    # Calculate the Euclidean distance between each row in df and each centroid
    return distances.idxmin(axis=1)
    # Get the index of the minimum distance, which corresponds to the label of the
    closest centroid


def new_centroids(df_label, df1):
    """
    Function to calculate the new centroids based on the current labels of the rows.
    Input:
        - df_label: pandas series with the label of the closest centroid for each row in
        df1
        - df1: pandas dataframe with the data
    Output:
        - new_centroids.T: pandas dataframe with
        the new centroids as columns and index as feature name
    """
    joined_df = df1.join(df_label)
    joined_df.rename(columns={0: 'Label'}, inplace=True) # Rename the column with the
    label
    # Calculate the mean of the rows with the same label
    return joined_df.groupby('Label').mean().T
    # Transpose the dataframe to have the new centroids as columns and index as feature
    name
```

```
def error_clusters(df_new_centroids,df1,df_label):
    """
    Calculate the error rate of each cluster.

    Args:
    - df_label (pandas.DataFrame): the label of the nearest centroid for each data point.
    - df1 (pandas.DataFrame): the dataset.
    - df_new_centroids (pandas.DataFrame): The
    new centroids computed in the current iteration.

    Returns:
    - error_rate (float): the total error rate of all clusters.
    """



    #Calculate mean value
    mean_centroid=df1.groupby('editorsSelection').mean().reset_index()
    # Transpose the new centroids dataframe and reset the index
    new_centroids= df_new_centroids.T
    # Get the columns of the data dataframe
    columns = df1.columns

    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):   #### centroid
        s=[]
        for j in range(len(mean_centroid)): ### mean centroid
        # Compute the distance between each data point and its assigned centroid
            distance = np.sum(np.square(mean_centroid[mean_c
            entroid['editorsSelection']==j][columns] - new_centroids.iloc[i]
            [columns]), axis=1)
            s.append(distance.iloc[0])
        sse.append(s)
    ## key  is the cluster number and value is the merged value

    merge_label=pd.DataFrame(sse).idxmin(axis=1).to_dict()
    ## Merging cluster based on the target variable
    df_label[0]=df_label[0].replace(merge_label)

    df1 = df1.join(df_label) # add the label column to the dataset
    df1.rename(columns={0: 'Label'}, inplace=True) # rename the label column
    error_list = []
    for i in df1['Label'].value_counts().index:
        df_cluster = df1[df1['Label'] == i]
        # filter the dataset to include only the data points in the current cluster
        y = len(df_cluster[df_cluster['editorsSelection'] == 1])
        # count the number of data points in the current cluster that were readmitted
        n = len(df_cluster[df_cluster['editorsSelection'] == 0])
        # count the number of data points in the current cluster that were not readmitted
        if y == 0 and n == 0:
```

```
            error = 0
        else:
            error = n / (n + y)
            # calculate the error rate of the current cluster
        error_list.append(error)
    return round(sum(error_list),4)


def sum_of_square_error(new_centroids, data, labels):
    """
    Computes the sum of squared errors between the data points and their assigned centroids.

    Args:
    new_centroids (DataFrame): The new centroids computed in the current iteration.
    data (DataFrame): The input data points.
    labels (DataFrame): The labels assigned to each data point.

    Returns:
    The sum of squared errors.
    """
    # Transpose the new centroids dataframe and reset the index
    new_centroids = new_centroids.T.reset_index()
    # Get the columns of the data dataframe
    columns = data.columns
    # Join the data dataframe and the labels dataframe
    data = data.join(labels)
    # Rename the '0' column of the labels dataframe to 'Label'
    data.rename(columns={0:'Label'}, inplace=True)
    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):
        distance =
        np.sum(np.square(data[data['Label']==i][columns] - new_centroids.iloc[i]
        [columns]), axis=1)
        sse.append(sum(distance))
    # Return the sum of squared errors
    return np.nansum(sse)

def kmeans_lyod_with_error(df1, k, tou):
    """
    Function to run the K-means Lloyd algorithm.
    Input:
        - df1: pandas dataframe with the data
        - k: integer number of clusters
        - tou: float tolerance level to stop the algorithm
    Output:
        - centroids: pandas dataframe with the
        final centroids as columns and index as label
    """
    start_time=time.time()
    centroids = initialize_centroids(df1, k) # Initialize random centroids
    initial_list_of_columns = centroids.columns.to_list()
    iteration = 0
```

```
    while True:
        # Assign labels to current centroids
        df_label = assign_labels(df1, centroids)
        df_label = pd.DataFrame(df_label)
        # Calculate new centroids
        df_new_centroids =
        new_centroids(df_label, df1)
        new_list_of_columns =
        df_new_centroids.columns.to_list()
        # Keep the number of clusters the same i.e maintain same k
        for i in initial_list_of_columns:
            if i not in new_list_of_columns:
                df_new_centroids[i] = centroids[i]
        # Calculate tao
        distance = []
        for col in centroids.columns:
            col_distance = euclidean(centroids[col],
            df_new_centroids[col])
            distance.append(col_distance)
        tao_calculated=sum(distance)/k #Used the
        formula provided for calculating Tao
        sse =
        sum_of_square_error(df_new_centroids, df1, df_label)
        #error=error_clusters(df_label,df1,k)
        end_time= time.time()
        if iteration>100:
            error=error_clusters(df_new_centroids,df1,df_label)
            print("Iteration exceeded")

            return error, sse,end_time-start_time
            break

        if tao_calculated<tou or iteration >100:   #if the convergence is met, kmeans
        will stop  or else if the convergence is never met, after 100 iteration code will stop
            error=error_clusters(df_new_centroids,df1,df_label)
            return error, sse,end_time-start_time
            break                                  # otherwise indefinite loop
        else:
            centroids= df_new_centroids # In
            case we need more iterations, the centroids calculated at this step
            acts as input
        iteration+=1



error_matrix_ny=[]
for i in range(2,6):
    for j in range(1,21):
        error,sse,run_time=kmeans_lyod_with_error(bag_of_words_df,i,10)
        error_matrix_ny.append([i,j,error,sse,run_time])
error_df_ny= pd.DataFrame(error_matrix_ny,columns=['number_of_cluster', 'iteration', 'error','sse','r
error_df_ny.to_csv('kmeans_llyod_ny.csv')
```

```
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
x=error_df_ny['number_of_cluster'].value_counts().index
y1=error_df_ny.groupby(['number_of_cluster']).mean()['error']
ax1.plot(x, y1, color='tab:blue')
ax1.set_xlabel('Number of clusters')
ax1.set_ylabel('Error', color='tab:blue')

ax2 = ax1.twinx()
y2 =
error_df_ny.groupby(['number_of_cluster']).mean()['sse']
ax2.plot(x, y2, color='tab:orange')
ax2.set_ylabel('SSE', color='tab:orange')
plt.title('Error and SSE (mean)for each cluster tao as convergence for new york data')
plt.xticks(range(2, 6))
plt.show()


import seaborn as sns
plt.figure(figsize=(6, 10))
sns.boxplot(x=error_df_ny['number_of_cluster'],y=error_df_ny['error'])
plt.title('Box Plot of error for K means llyod for new york data')
plt.show()


import seaborn as sns
sns.boxplot(x=error_df_ny['number_of_cluster'],y=error_df_ny['run_time'])
plt.title('Box Plot of runtime for K means llyod
for new york data')
plt.show()
```

Discussion of Findings

Answer here...

I have implemented the kmeans llyod algorithm for number of clusters ranging from 2 to 5. The method involves initializing k centroids randomly. Then assigning clusters to each datapoint. Centroid is calculated by taking mean if datapoints and this serve as new centroids. Have handled scenarios to maintain k centroids always and handling ties,etc. After each iteration, calculated the tao i.e magnitude of scalar product between the new cluster and old cluster. If the tao is below certain threshold, then the algorithm converges and these are the final sets of centroids. Here the data for diabetes and new york comments had a target variable each having two target classes. So to calculate the error we need to merge the classes. Have merged the clusters into 2 as per the class of target variable by calculating distance metric. Whichever distance is minimum,that distance from cluster to target variable is selected for merging clusters such that 2 classes are remaining. Have ran this algorithm for diabetes data set and new york dataset of comments. Have performed required EDA on datasets.

Diabetes Dataset

From the line plot below for error and sum of square error with number of clusters it can be seen that error graph balances out with increase in the number of clusters i.e from 2 to 5. The error is decreasing with

increase in number of clusters. Even the sum of square error shows a decrease with increase in the number of clusters. From the box plot for error wrt number of clusters, shows the median error for each cluster shows a gradual decrease. The box plot of runtime shows that with increase in number of clusters the median run time also increases with increase in number of clusters

New York Dataset

To represent the Comment Body column as a vector, I utilized Natural Language Processing techniques. Firstly, I performed pre-processing steps to clean the data, including removing stop words, stemming words to their root form, removing URLs, punctuation, non-alphanumeric characters, and digits. I also converted the text to lowercase to standardize the data.

Next, I employed the CountVectorizer method to convert the text into an appropriate vector representation. CountVectorizer is a powerful tool that focuses on the frequency of words in a document. Unlike TF-IDF, which considers the frequency of words across all documents in the dataset, CountVectorizer is better suited to capture the specificity of each document, making it a better choice for clustering similar documents together.

CountVectorizer is also simpler and faster than TF-IDF, as it only counts the number of occurrences of each word in a document. This makes it more suitable for processing large amounts of text data, as complex calculations are not required.

One of the advantages of using CountVectorizer is that it produces integer counts for each word, which are easier to interpret and use in clustering algorithms. This makes it a more efficient method for identifying patterns in text data.

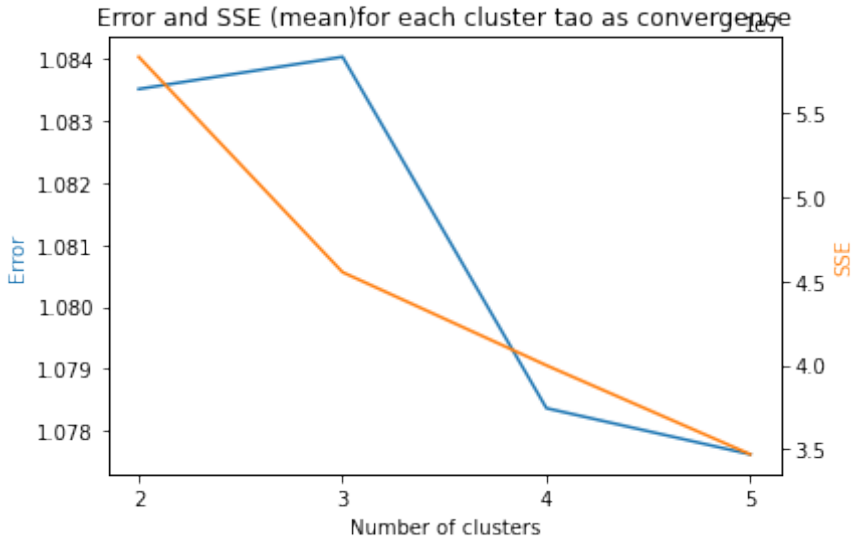
In summary, to represent the Comment Body column as a vector, I utilized NLP techniques and CountVectorizer. By doing so, I was able to capture the unique characteristics of each document, making it easier to identify patterns and cluster similar documents together. CountVectorizer's simplicity and efficiency make it a better choice for processing large amounts of text data than other methods such as TF-IDF.

From the below line graph for error and sum of square error vs number of clusters it can be seen that error is remaining almost constant over the number of clusters. One of the reasons for this could be that as this dataset was comment body, after pre processing the dataset became sparse as the columns are mostly the words. So the column values are mostly 0 or 1 and hence the distance is almost same. We have used distance metric to merge and calculate error. If the data is sparse, meaning that most of the columns contain mostly 0s, then the k-means algorithm may not be able to form distinct clusters. In this case, the error may remain constant regardless of the number of clusters chosen. Also it is possible that homogeneous clusters may be formed. Binary features can have a tendency to form homogeneous clusters. In other words, some clusters may contain mostly 1s and others may contain mostly 0s. In this case, the k-means algorithm may not be able to form distinct clusters, and the error may remain constant regardless of the number of clusters chosen. Even the box plot for error shows the same ,remains constant with increase in number of clusters. The Sum of square error decrease with increase in number of clusters as shown in the line graph. The box plot for run time shows that the run time increases with increase in number of clusters.

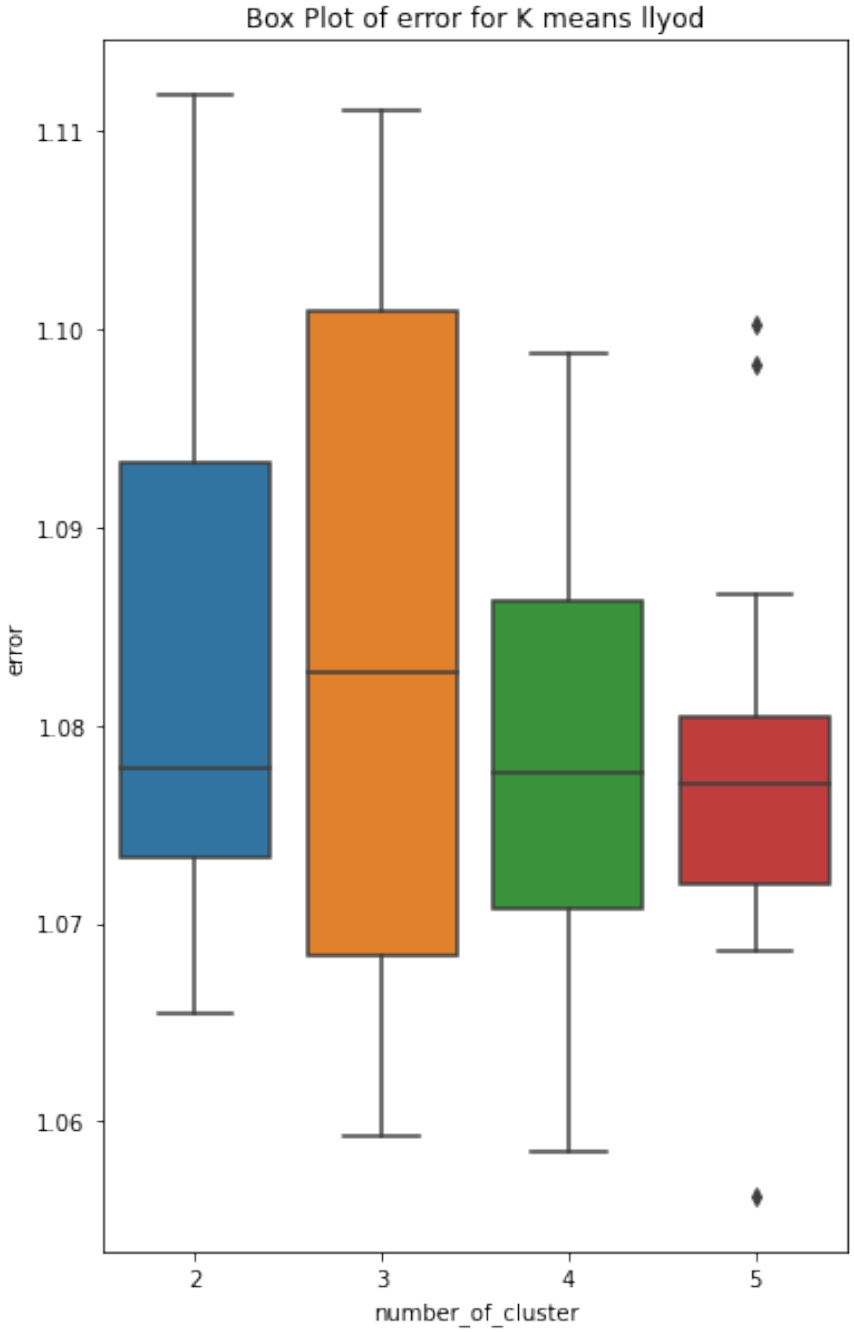
Plots

Place images here with suitable captions.

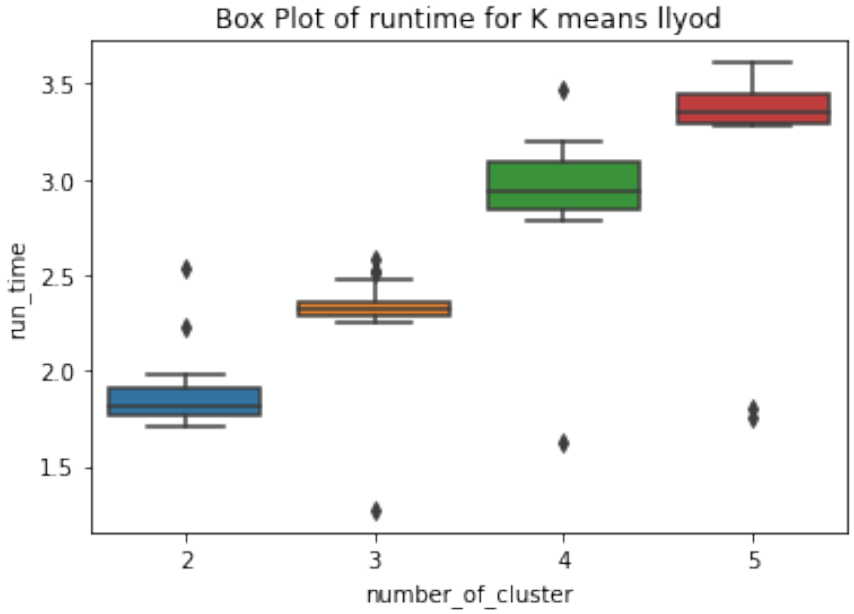
Plots for Diabetes dataset



(11)

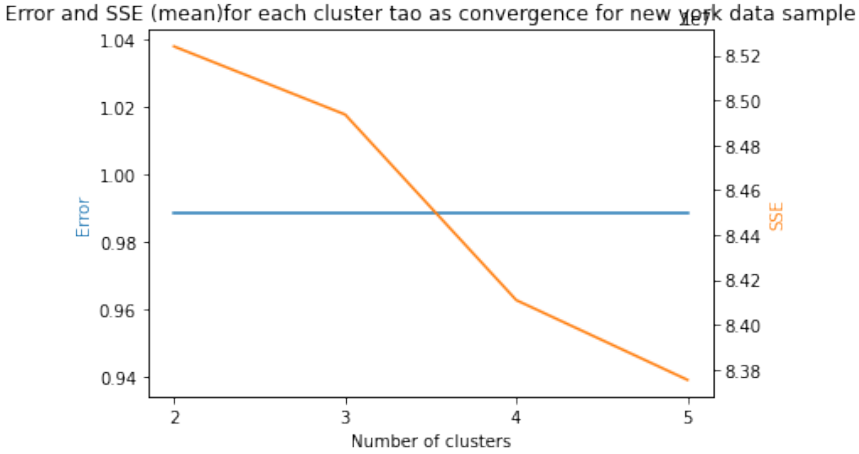

Box Plot of error for K means llyod
error
1.11
1.10
1.09
1.08
1.07
1.06
2
3
4
5
number_of_cluster

(12)

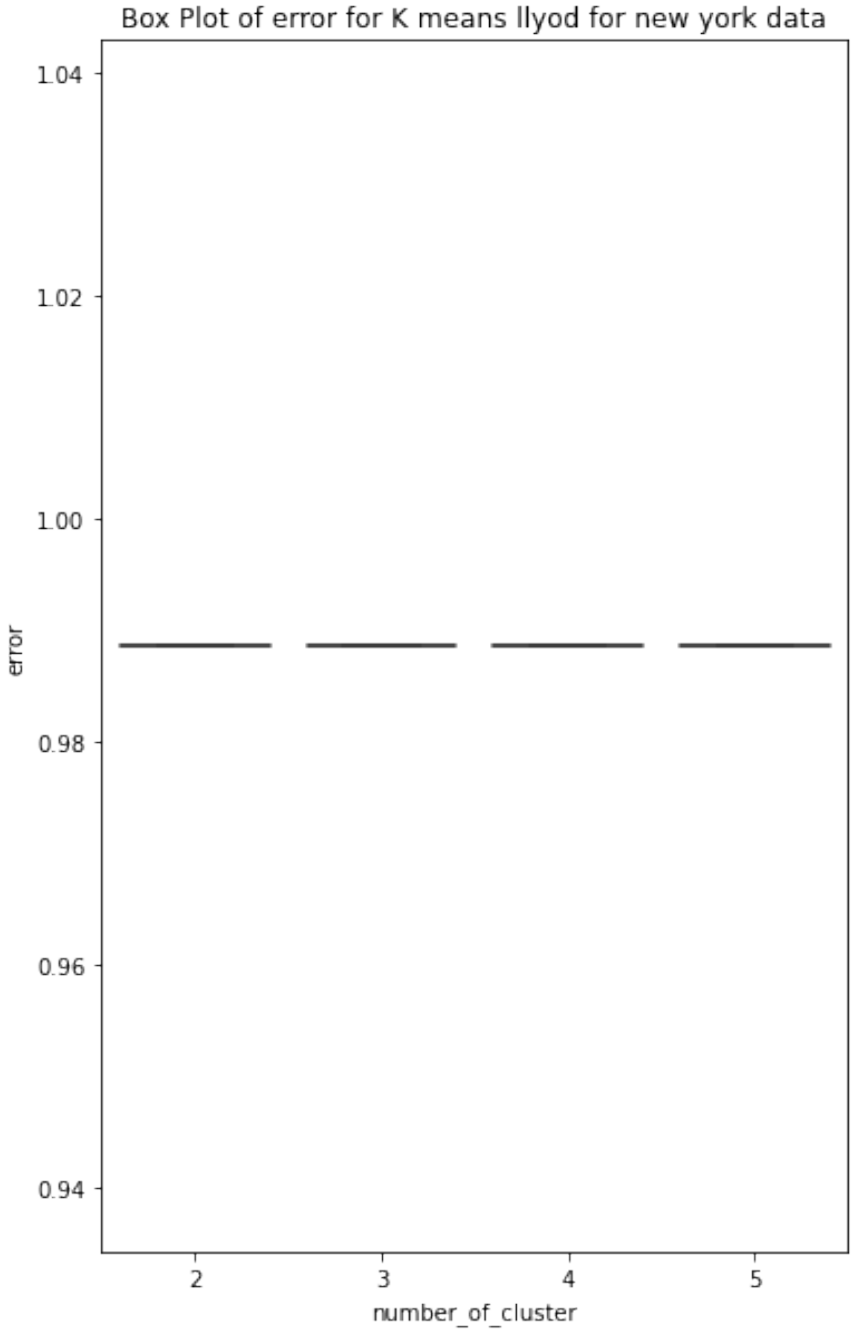


(13)

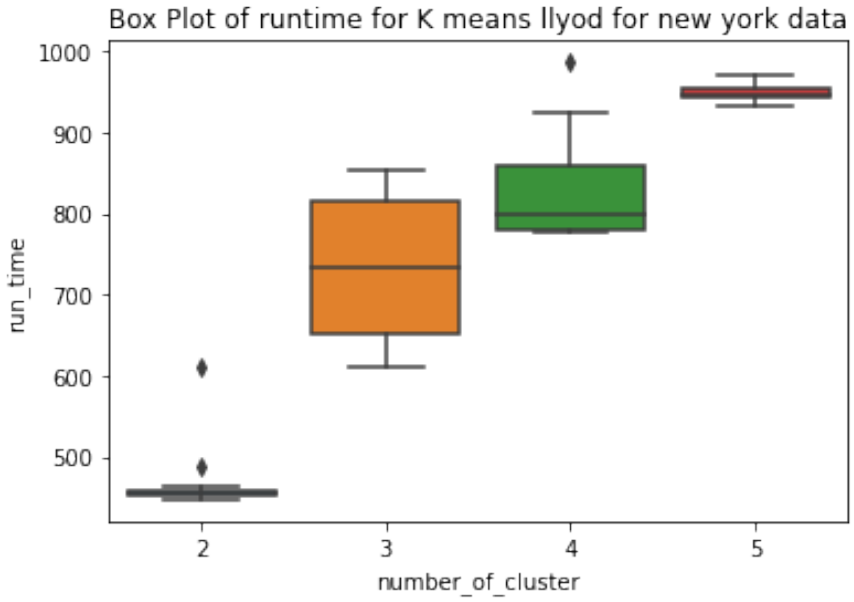
Plots for New York Times Comment dataset



(14)


Box Plot of error for K means llyod for new york data
1.04
1.02
1.00
0.98
0.96
0.94
error
2
3
4
5
number_of_cluster

(15)



(16)

Problem 5

The k-means algorithm provided above stops when centroids become stable (Line 34). In theory, k-means converges once SSE is minimized

$$SSE = \sum_{j}^{k} \sum_{x \in c_j.B} ||\mathbf{x} - c_j.v||_2^2$$

In this question, you are asked to use SSE as stopping criterion. Run your program, $C_{k_{SSE}}$, against the Diabetes and New York Times Comments data sets. Report the total error rates for $k = 2, \ldots 5$ for 20 runs each for both data sets. Moreover, compare k-means and kmeans++'s run time for $k = 2, \ldots 5$ for 20 runs using both data sets. Presenting the results that are easily understandable. Plots are generally a good way to convey complex ideas quickly, i.e., box plot. Discuss your results [**20 points**].

R/Python script

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

```python
# Sample Python Script With Highlighting


import time
def initialize_centroids(df, k):
    """
    Function to initialize random centroids from dataset.
    Input:
        - df: pandas dataframe with the data
        - k: integer number of clusters
```

```
    Output:
        - temp_df: pandas dataframe with the centroids as columns and index as label
    """
    centroids = []
    for i in range(k):
        centroids.append(df.apply(lambda x: float(x.sample())))
        # Take a random sample from each column to create a centroid
    centroids = pd.concat(centroids, axis=1)
    centroids.index.name = 'Label'

    return centroids


def assign_labels(df, centroids):
    """
    Function to calculate the closest centroid
    label for each row in a dataframe.
    Input:
        - df: pandas dataframe with the data
        - centroids: pandas dataframe with the centroids as columns and index as label
    Output:
        - distances.idxmin(axis=1): pandas
        series with the label of the closest centroid for each row in df
    """
    distances = centroids.swifter.apply(lambda x: np.sqrt(((df - x) ** 2).sum(axis=1)))
    # Calculate the Euclidean distance between each row in df and each centroid
    return distances.idxmin(axis=1)
    # Get the index of the minimum distance, which
    corresponds to the label of the closest centroid


def new_centroids(df_label, df1):
    """
    Function to calculate the new centroids based on the current labels of the rows.
    Input:
        - df_label: pandas series with the label
        of the closest centroid for each row in df1
        - df1: pandas dataframe with the data
    Output:
        - new_centroids.T: pandas dataframe with the new centroids as columns and index as feature na
    """
    joined_df = df1.join(df_label)
    joined_df.rename(columns={0: 'Label'}, inplace=True)
    # Rename the column with the label
    # Calculate the mean of the rows with the same label
    return joined_df.groupby('Label').mean().T
    # Transpose the dataframe to have the new centroids as columns and index as feature name



def
error_clusters(df_new_centroids,df1,df_label):
    """
```

```
    Calculate the error rate of each cluster.

    Args:
    - df_label (pandas.DataFrame): the label of the nearest centroid for each data point.
    - df1 (pandas.DataFrame): the dataset.
    - df_new_centroids (pandas.DataFrame): The
    new centroids computed in the current iteration.

    Returns:
    - error_rate (float): the total error rate
    of all clusters.
    """


    #Calculate mean value
    mean_centroid=df1.groupby('readmitted').mean(

    ).reset_index()
    # Transpose the new centroids dataframe and
    reset the index
    new_centroids= df_new_centroids.T
    # Get the columns of the data dataframe
    columns = df1.columns

    sse = []
    # Compute the distance between each data
    point and its assigned centroid
    for i in range(len(new_centroids)):
    #### centroid
        s=[]
        for j in range(len(mean_centroid)):
        ### mean centroid
        # Compute the distance between each data point and its assigned centroid
            distance = np.sum(np.square(mean_centroid[mean_c
            entroid['readmitted']==j][columns] -
            new_centroids.iloc[i][columns]),
            axis=1)
            s.append(distance.iloc[0])
        sse.append(s)
    ## key  is the cluster number and value is the merged value
    merge_label=pd.DataFrame(sse).idxmin(axis=1).
    to_dict()
    ## Merging cluster based on the target variable
    df_label[0]=df_label[0].replace(merge_label)

    df1 = df1.join(df_label)
    # add the label column to the dataset
    df1.rename(columns={0: 'Label'}, inplace=True) # rename the label column
    error_list = []
    for i in df1['Label'].value_counts().index:
        df_cluster = df1[df1['Label'] == i]
        # filter the dataset to include only the data points in the current cluster
        y =
```

```
        len(df_cluster[df_cluster['readmitted'] == 1])
        # count the number of data points in the current cluster that were readmitted
        n =
        len(df_cluster[df_cluster['readmitted'] == 0])
        # count the number of data points in the current cluster that were not readmitted
        if y == 0 and n == 0:
            error = 0
        else:
            error = n / (n + y) # calculate the error rate of the current cluster
        error_list.append(error)
    return round(sum(error_list),4)




def sum_of_square_error(new_centroids, data, labels):
    """
    Computes the sum of squared errors between the data points and their assigned centroids.

    Args:
    new_centroids (DataFrame): The new centroids computed in the current iteration.
    data (DataFrame): The input data points.
    labels (DataFrame): The labels assigned to each data point.

    Returns:
    The sum of squared errors.
    """
    # Transpose the new centroids dataframe and reset the index
    new_centroids = new_centroids.T.reset_index()
    # Get the columns of the data dataframe
    columns = data.columns
    # Join the data dataframe and the labels dataframe
    data = data.join(labels)
    # Rename the '0' column of the labels dataframe to 'Label'
    data.rename(columns={0:'Label'}, inplace=True)
    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):
        distance = np.sum(np.square(data[data['Label']==i]
        [columns] - new_centroids.iloc[i][columns]), axis=1)
        sse.append(sum(distance))
    # Return the sum of squared errors
    return sum(sse)



def kmeans_lloyd_sse_convergence(df1, k,sse_threshold):

    """
    Computes the k-means clustering algorithm
    until the sum of squared errors (SSE) decreases by less than threshold,
    or the maximum number of iterations is reached.
```

```
    Args:
    df1 (df1Frame): The input data points.
    k (int): The number of clusters.
    sse_threshold (int): The threshold for sse percent reduction
    Returns:
    The error,sse, sse_initial,percent,end_time-start_time, iteration
    """



    # Get the current time
    start_time = time.time()
    # Set the initial iteration number to 0
    iteration = 0
    # Generate random initial centroids
    centroids = initialize_centroids(df1, k)
    # Get the columns of the centroids df1frame
    initial_list_of_columns = centroids.columns.to_list()
    # Assign labels to the initial centroids
    labels_i = assign_labels(df1, centroids)
    # Convert the labels to a df1frame
    df_labels_i = pd.DataFrame(labels_i)
    # Compute the initial SSE
    sse_initial = sum_of_square_error(centroids, df1, df_labels_i)

    while True:
        # Compute new centroids based on the current labels

        df_new_centroids = new_centroids(df_labels_i, df1)
        new_list_of_columns = df_new_centroids.columns.to_list()
        # Keep the number of clusters the same
        for i in initial_list_of_columns:
            if i not in new_list_of_columns:
                df_new_centroids[i] = centroids[i]
        # Assign labels based on the new centroids
        labels = assign_labels(df1, df_new_centroids)
        # Convert the labels to a df1frame
        df_labels = pd.DataFrame(labels)
        # Compute the new SSE
        sse = sum_of_square_error(df_new_centroids, df1, df_labels)
        percent = 100*(sse_initial - sse) / sse_initial
        end_time = time.time()
        # Check if the maximum number of iterations has been reached
        if iteration > 100:
            print("Maximum number of iterations exceeded.")

            error=error_clusters(df_new_centroids,df1,df_labels)
            return error,sse,
            sse_initial,percent,end_time-start_time, iteration
            break

        if sse_initial > sse and percent > sse_threshold :
```

```
            error=error_clusters(df_new_centroids
            ,df1,df_labels)
            return error,sse,
            sse_initial,percent,end_time-start_time, iteration
            break
        else:
            centroids = df_new_centroids
            labels_i = df_labels
            sse_initial = sse


        iteration += 1


error_matrix_sse=[]
for i in range(2,6):
    for j in range(1,21):
        error,sse, sse_initial,percent,run_time, iteration=kmeans_lloyd_sse_convergence(df
        _cleaned_dia,i,10)
        error_matrix_sse.append([i,j,error,sse,
        sse_initial,percent,run_time, iteration])
error_df_sse=
pd.DataFrame(error_matrix_sse,columns=
['number_of_cluster', 'repetition','error','sse','sse_initial','percent
','run_time','iteration'])
error_df_sse



import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
x=error_df_sse['number_of_cluster'].value_counts().index
y1=error_df_sse.groupby(['number_of_cluster']).mean()['error']
ax1.plot(x, y1, color='tab:blue')
ax1.set_xlabel('Number of clusters')
ax1.set_ylabel('Error', color='tab:blue')

ax2 = ax1.twinx()
y2 = error_df_sse.groupby(['number_of_cluster']).mean()['sse']
ax2.plot(x, y2, color='tab:orange')
ax2.set_ylabel('SSE', color='tab:orange')
plt.title('Error and SSE (mean)for each cluster SSE as convergence')
plt.xticks(range(2, 6))
plt.show()


import seaborn as sns
plt.figure(figsize=(6, 10))
sns.boxplot(x=error_df_sse['number_of_cluster'],y=error_df_sse['error'])
plt.title('Box Plot of error for K means llyod SSE')
plt.show()
```

```
import seaborn as sns
sns.boxplot(x=error_df_sse['number_of_cluster'],y=error_df_sse['run_time'])
plt.title('Box Plot of runtime for K means with SSE')
plt.show()


error_df_diab['algo']='kmeans'
error_df_sse['algo']='kmeans_sse'
error_df_plus_diab['algo']='kmeans_++'

run_time_diab=pd.DataFrame()
run_time_diab=pd.concat( [
error_df_diab[['algo','number_of_cluster',
'iteration', 'run_time']],
    error_df_sse[['algo','number_of_cluster',
    'iteration', 'run_time']],
    error_df_plus_diab[['algo','number_of_cluster
    ', 'iteration', 'run_time']]
                        ],ignore_index=True )

import seaborn as sns

fig, ax = plt.subplots(figsize=(8,6))

sns.boxplot(x='number_of_cluster', y='run_time', hue='algo',
data=run_time_diab[run_time_diab['algo'].isin (['kmeans','kmeans_sse'])],ax=ax);
plt.title('Box Plot of run time for K means and K means with SSE')
plt.show()




import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
import swifter

bag_of_words_df=pd.read_csv('bag_of_words.csv')
comments_df=pd.read_csv('cleaned_data.csv')

bag_of_words_df=bag_of_words_df.join(comments_df[
'editorsSelection'].replace({True:1,False:0}))
bag_of_words_df.columns
```

```
import time
from scipy.spatial.distance import euclidean

def initialize_centroids(df, k):
    """
    Function to initialize random centroids from
    dataset.
    Input:
        - df: pandas dataframe with the data
        - k: integer number of clusters
    Output:
        - temp_df: pandas dataframe with the centroids as columns and index as label
    """
    centroids = []
    for i in range(k):
        centroids.append(df.apply(lambda x: float(x.sample())))
        # Take a random sample from each column to create a centroid
    centroids = pd.concat(centroids, axis=1)
    centroids.index.name = 'Label'

    return centroids


def assign_labels(df, centroids):
    """
    Function to calculate the closest centroid label for each row in a dataframe.
    Input:
        - df: pandas dataframe with the data
        - centroids: pandas dataframe with the centroids as columns and index as label
    Output:
        - distances.idxmin(axis=1): pandas series with the label of the closest centroid for each rou
    """
    distances = centroids.swifter.apply(lambda x: np.sqrt(((df - x) ** 2).sum(axis=1)))
    # Calculate the Euclidean distance between each row in df and each centroid
    return distances.idxmin(axis=1)
    # Get the index of the minimum distance, which corresponds to the label of the
    closest centroid


def new_centroids(df_label, df1):
    """
    Function to calculate the new centroids based on the current labels of the rows.
    Input:
        - df_label: pandas series with the label
        of the closest centroid for each row in df1
        - df1: pandas dataframe with the data
    Output:
        - new_centroids.T: pandas dataframe with
        the new centroids as columns and index as feature name
    """
    joined_df = df1.join(df_label)
    joined_df.rename(columns={0: 'Label'},
```

```
    inplace=True)
    # Rename the column with the label
    # Calculate the mean of the rows with the same label
    return joined_df.groupby('Label').mean().T
    # Transpose the dataframe to have the new
    centroids as columns and index as feature name




def error_clusters(df_new_centroids,df1,df_label):
    """
    Calculate the error rate of each cluster.

    Args:
    - df_label (pandas.DataFrame): the label of
    the nearest centroid for each data point.
    - df1 (pandas.DataFrame): the dataset.
    - df_new_centroids (pandas.DataFrame): The
    new centroids computed in the current iteration.

    Returns:
    - error_rate (float): the total error rate of all clusters.
    """



    #Calculate mean value
    mean_centroid=df1.groupby('editorsSelection')
    .mean().reset_index()
    # Transpose the new centroids dataframe and
    reset the index
    new_centroids= df_new_centroids.T
    # Get the columns of the data dataframe
    columns = df1.columns

    sse = []
    # Compute the distance between each data
    point and its assigned centroid
    for i in range(len(new_centroids)):   #### centroid
        s=[]
        for j in range(len(mean_centroid)): ### mean centroid
        # Compute the distance between each data point and its assigned centroid
            distance = np.sum(np.square(mean_centroid[mean_c
            entroid['editorsSelection']==j][columns] - new_centroids.iloc[i]
            [columns]), axis=1)
            s.append(distance.iloc[0])
        sse.append(s)
    ## key  is the cluster number and value is the merged value
    merge_label=pd.DataFrame(sse).idxmin(axis=1).
    to_dict()
    ## Merging cluster based on the target variable
    df_label[0]=df_label[0].replace(merge_label)
```

```
    df1 = df1.join(df_label) # add the label column to the dataset
    df1.rename(columns={0: 'Label'}, inplace=True) # rename the label column
    error_list = []
    for i in df1['Label'].value_counts().index:
        df_cluster = df1[df1['Label'] == i]
        # filter the dataset to include only the data points in the current cluster
        y = len(df_cluster[df_cluster['editorsSelection'] == 1]) # count the number of data points i
        n=len(df_cluster[df_cluster['editorsSelection'] == 0])
        # count the number of data points in the current cluster that were not readmitted
        if y == 0 and n == 0:
            error = 0
        else:
            error = n / (n + y)
            # calculate the error rate of the current cluster
        error_list.append(error)
    return round(sum(error_list),4)




def sum_of_square_error(new_centroids, data, labels):
    """
    Computes the sum of squared errors between
    the data points and their assigned centroids.

    Args:
    new_centroids (DataFrame): The new centroids computed in the current iteration.
    data (DataFrame): The input data points.
    labels (DataFrame): The labels assigned to each data point.

    Returns:
    The sum of squared errors.
    """
    # Transpose the new centroids dataframe and reset the index
    new_centroids = new_centroids.T.reset_index()
    # Get the columns of the data dataframe
    columns = data.columns
    # Join the data dataframe and the labels
    dataframe
    data = data.join(labels)
    # Rename the '0' column of the labels
    dataframe to 'Label'
    data.rename(columns={0:'Label'},
    inplace=True)
    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):
        distance =
        np.sum(np.square(data[data['Label']==i][columns] - new_centroids.iloc[i]
        [columns]), axis=1)
        sse.append(sum(distance))
```

```
    # Return the sum of squared errors
    return np.nansum(sse)


def kmeans_lloyd_sse_convergence(df1, k,sse_threshold):

    """
    Computes the k-means clustering algorithm
    until the sum of squared errors (SSE) decreases by less than threshold,
    or the maximum number of iterations is reached.

    Args:
    df1 (df1Frame): The input data points.
    k (int): The number of clusters.
    sse_threshold (int): The threshold for sse percent reduction
    Returns:
    The error,sse, sse_initial,percent,end_time-start_time, iteration
    """


    # Get the current time
    start_time = time.time()
    # Set the initial iteration number to 0
    iteration = 0
    # Generate random initial centroids
    centroids = initialize_centroids(df1, k)
    # Get the columns of the centroids df1frame
    initial_list_of_columns =
    centroids.columns.to_list()
    # Assign labels to the initial centroids
    labels_i = assign_labels(df1, centroids)
    # Convert the labels to a df1frame
    df_labels_i = pd.DataFrame(labels_i)
    # Compute the initial SSE
    sse_initial = sum_of_square_error(centroids, df1, df_labels_i)

    while True:
        # Compute new centroids based on the current labels

        df_new_centroids =
        new_centroids(df_labels_i, df1)
        new_list_of_columns =
        df_new_centroids.columns.to_list()
        # Keep the number of clusters the same
        for i in initial_list_of_columns:
            if i not in new_list_of_columns:

                df_new_centroids[i] = centroids[i]
        # Assign labels based on the new centroids
        labels = assign_labels(df1,
        df_new_centroids)
        # Convert the labels to a df1frame
```

```
        df_labels = pd.DataFrame(labels)
        # Compute the new SSE
        sse =
        sum_of_square_error(df_new_centroids, df1, df_labels)
        percent = 100*(sse_initial - sse) /
        sse_initial
        end_time = time.time()
        # Check if the maximum number of iterations has been reached
        if iteration > 100:
            print("Maximum number of iterations exceeded.")

            error=error_clusters(df_new_centroids,df1,df_labels)
            return error,sse,
            sse_initial,percent,end_time-start_time, iteration
            break

        if sse_initial > sse and percent > sse_threshold :
            error=error_clusters(df_new_centroids
            ,df1,df_labels)
            return error,sse,
            sse_initial,percent,end_time-start_time, iteration
            break
        else:
            centroids = df_new_centroids
            labels_i = df_labels
            sse_initial = sse


        iteration += 1



error_matrix_sse=[]
for i in range(2,6):
    for j in range(1,21):
        error,sse, sse_initial,percent,run_time, iteration=kmeans_lloyd_sse_convergence(bag_of_words
        error_matrix_sse.append([i,j,error,sse, sse_initial,percent,run_time, iteration])
error_df_sse_ny= pd.DataFrame(error_matrix_sse,columns=
['number_of_cluster', 'repetition','error','sse','sse_initial','percent
','run_time','iteration'])
error_df_sse_ny.to_csv('kmeans_llyod_sse.csv')


import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
x=error_df_sse_ny['number_of_cluster'].value_counts().index
y1=error_df_sse_ny.groupby(['number_of_cluster']).mean()['error']
ax1.plot(x, y1, color='tab:blue')
ax1.set_xlabel('Number of clusters')
ax1.set_ylabel('Error', color='tab:blue')

ax2 = ax1.twinx()
y2 = error_df_sse_ny.groupby(['number_of_cluster']).mean()['sse']
```

```
ax2.plot(x, y2, color='tab:orange')
ax2.set_ylabel('SSE', color='tab:orange')
plt.title('Error and SSE (mean)for each cluster SSE as convergence for new york data')
plt.xticks(range(2, 6))
plt.show()



import seaborn as sns
plt.figure(figsize=(6, 10))
sns.boxplot(x=error_df_sse_ny['number_of_cluster'],y=error_df_sse_ny['error'])
plt.title('Box Plot of error for K means llyod SSE for new york data')
plt.show()


import seaborn as sns
sns.boxplot(x=error_df_sse_ny['number_of_cluster'],y=error_df_sse_ny['run_time'])
plt.title('Box Plot of runtime for K means with SSE for new york data')
plt.show()
```

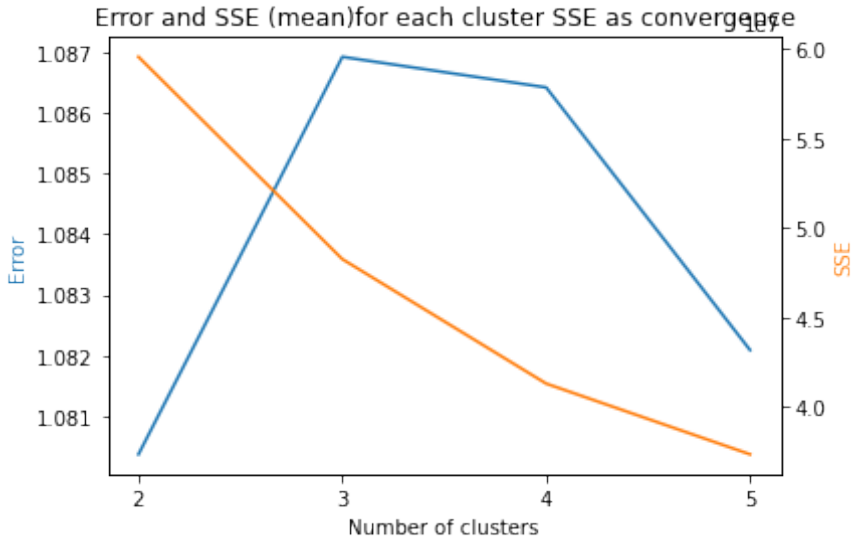
Discussion of Findings

Answer here...
I have implemented the kmeans llyod algorithm with sse convergence for number of clusters ranging from 2 to 5. The method is similar to kmeans llyod with different convergence criteria. Here sum of square error (SSE) is used at the stopping criteria. When the SSE becomes constant or the new iteration reduces the SSE by certain threshold, then the algorithm stops. The threshold of SSE reduction percentage is user defined.Rest all the functions are similar to kmeans Llyods. According to me, SSE convergence can be a better metric to evaluate the quality of clusters. Improving the cohesion among the datapoints assigned to same cluster and when it reaches a certain threshold the algorithm converges. Have ran this algorithm for diabetes data set and new york dataset sample of comments. Have performed required EDA on datasets.
Diabetes Dataset
From the line plot below for error and sum of square error with number of clusters it can be seen that error graph balances out with increase in the number of clusters i.e from 2 to 5. The error is decreasing with increase in number of clusters. Firstly there is a small increase and then the error deceases with increase in clusters.The sum of square error shows a decrease with increase in the number of clusters. From the box plot for error wrt number of clusters, shows the median error for each cluster shows a gradual decrease after a small rise initially. The box plot of runtime shows that with increase in number of clusters the median run time also increases with increase in number of clusters. When the runtime of kmeans llyos is compared to runtime of kmeans with sse convergence it can be seen that for less number of clusters the median run time of kmeans llyod is more that that of kmeans with SSE convergence i.e it takes more time to converge. But when number of clusters increases i.e becomes 5, the median time to converge for kmeans with sse convergence is more than that of kmeans llyod. When number of clusters become more run time of kmeans lyods is less than that of sse converged kmeans.
New York Dataset
From the below line graph for error and sum of square error vs number of clusters it can be seen that error is remaining almost constant over the number of clusters. One of the reasons for this could be that as this dataset was comment body, after pre processing the dataset became sparse as the columns are mostly the words. So the column values are mostly 0 or 1 and hence the distance is almost same. We have used distance metric to merge and calculate error. If the data is sparse, meaning that most of the columns contain mostly

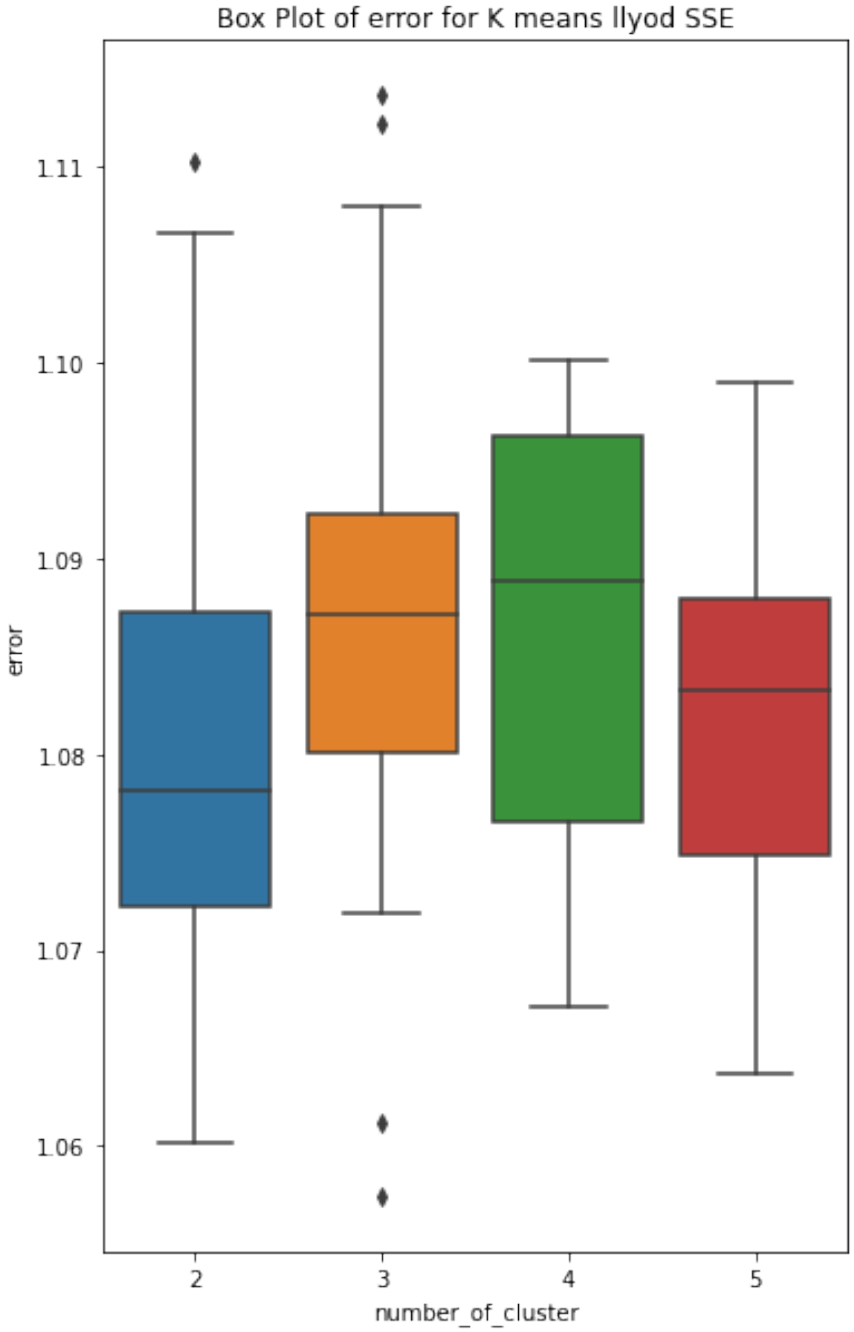
0s, then the k-means algorithm may not be able to form distinct clusters. In this case, the error may remain constant regardless of the number of clusters chosen. Also it is possible that homogeneous clusters may be formed.Binary features can have a tendency to form homogeneous clusters. In other words, some clusters may contain mostly 1s and others may contain mostly 0s. In this case, the k-means algorithm may not be able to form distinct clusters, and the error may remain constant regardless of the number of clusters chosen. Even the box plot for error shows the same ,remains constant with increase in number of clusters. The Sum of square error decrease with increase in number of clusters as shown in the line graph. The box plot for run time shows that the run time increases with increase in number of clusters. The box plot of kmeans sse convergence compared with kmeans llyod, it can be seen that the median run time for kmeans sse is more than that of kmeans llyod.

Plots

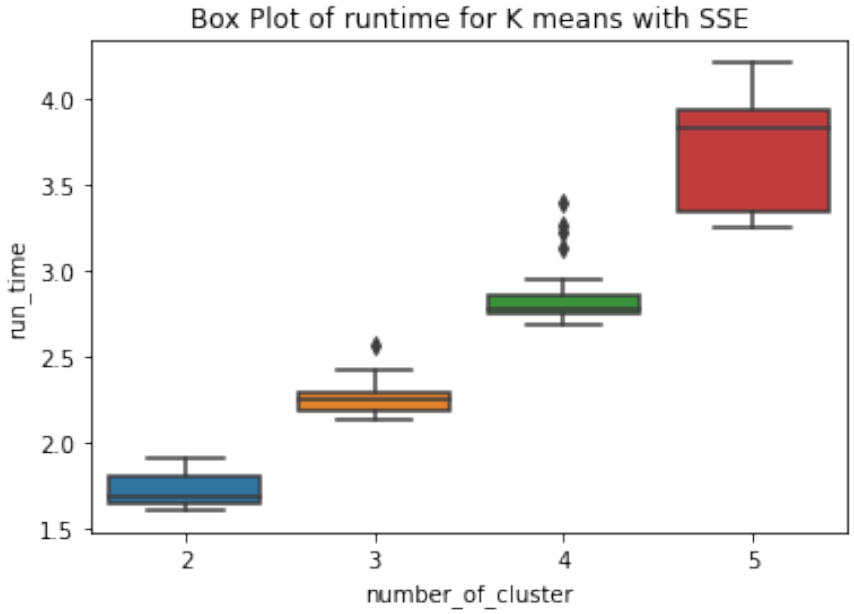
Place images here with suitable captions. Plots for Diabetes Dataset



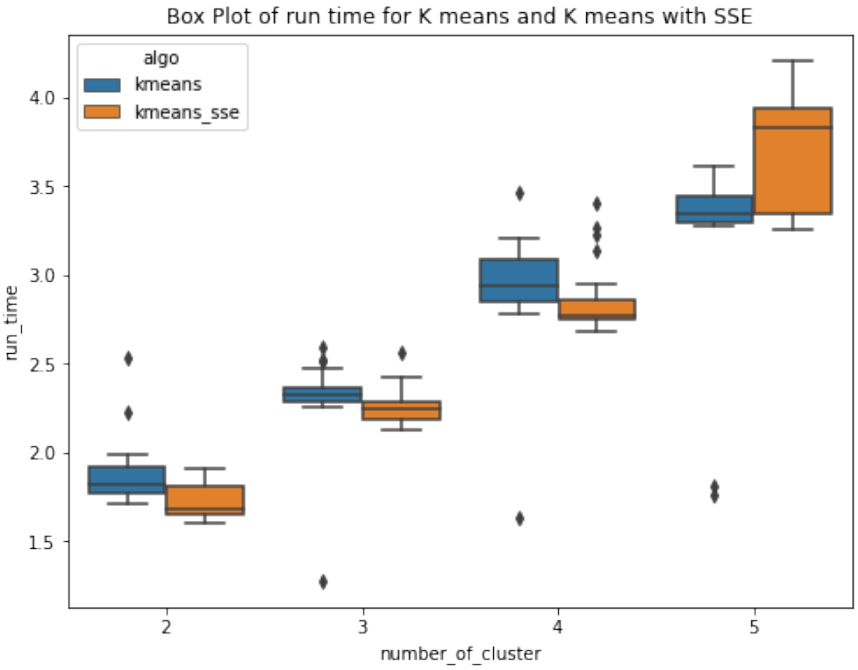
(17)


Box Plot of error for K means llyod SSE
error
1.11
1.10
1.09
1.08
1.07
1.06
2
3
4
5
number_of_cluster

(18)

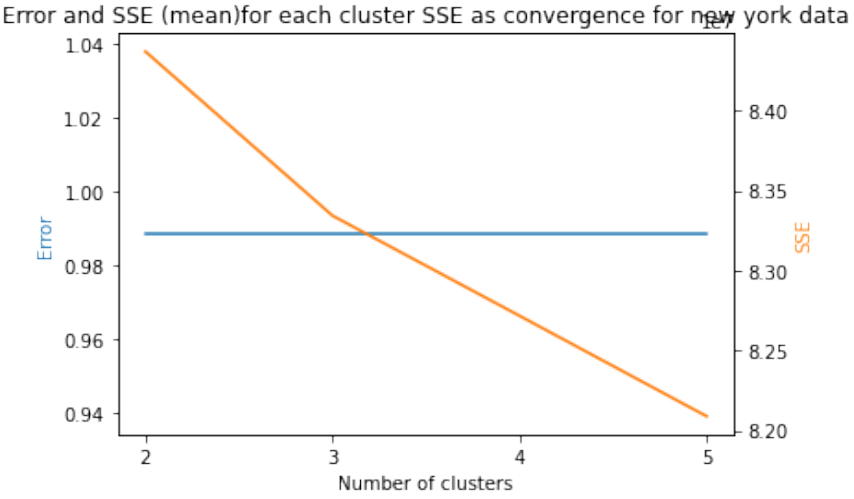


(19)

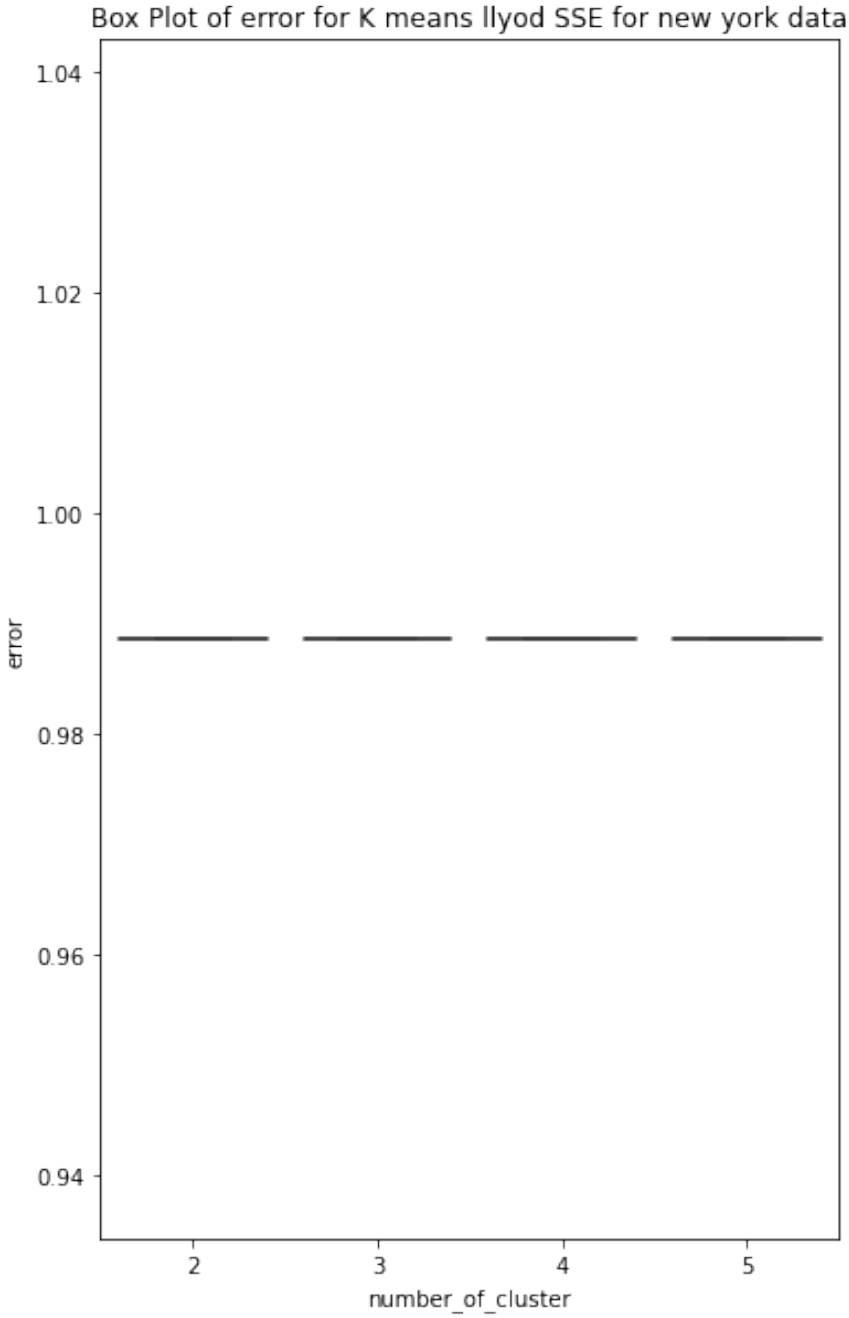


(20)

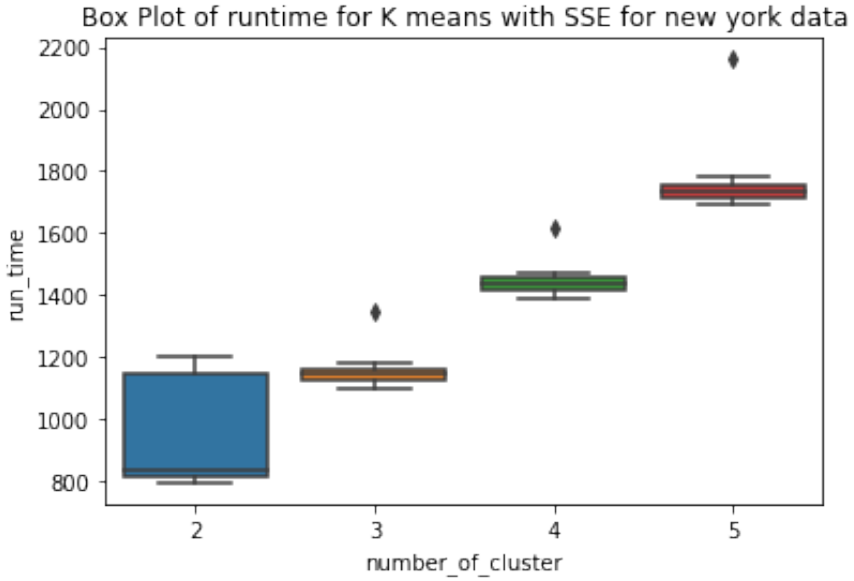
Plots for New York Times Comment dataset


Error and SSE (mean)for each cluster SSE as convergence for new york data
1e7
1.04
1.02
1.00
0.98
0.96
0.94
Error
8.40
8.35
8.30
8.25
8.20
SSE
2
3
4
5
Number of clusters

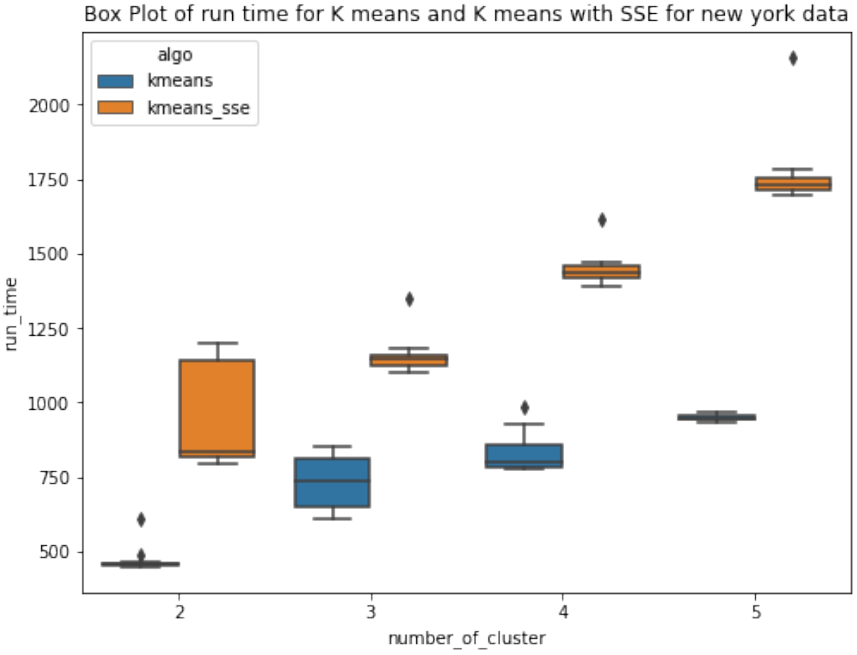
(21)


Box Plot of error for K means llyod SSE for new york data
1.04
1.02
1.00
0.98
0.96
0.94
error
2
3
4
5
number_of_cluster

(22)



(23)



(24)

Problem 6

Traditional k-means initialization is based on choosing values from a uniform distribution. In this question, you are asked to improve k-means through initialization. k-means ++ is an extended k-means clustering algorithm and induces non-uniform distributions over the data that serve as the initial centroids. Read the paper and discuss the idea in a paragraph. Implement this idea to improve your k-means program. Run your program, C_{k++}, against the Diabetes and New York Times Comments data sets. Report the total error rates for $k = 2, \ldots 5$ for 20 runs each for both data sets. Moreover, compare C_k, $C_{k_{SSE}}$ and C_{k++}'s run time

for $k = 2, \ldots 5$ for 20 runs using both data sets. Presenting the results that are easily understandable. Plots are generally a good way to convey complex ideas quickly, i.e., box plot. Discuss your results [**20 points**].

R/Python script

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

```python
# Sample Python Script With Highlighting

import pandas as pd
import numpy as np
import swifter
import time
def initialize_centroids_plus(df,k):
    """
    Function to calculate the random centroid
    using kmeans ++ technique.
    Input:
        - df: pandas dataframe with the data
        - k: number of clusters
    Output:
        - centroid.T: pandas dataframe with all centroids initialized
    """
    #Initialize random centroids from dataset
    centroid = []
    centroid.append(df.apply(lambda x: float(x.sample())))
    centroid=pd.DataFrame(centroid)
    column=centroid.columns.to_list()
    ##Randomly created first centroid from the domain of each column in the dataframe

    for i in range(1,k):
    ## The above for loop is for generating k-1
    clusters as first random cluster is already generated
        distance=pd.DataFrame()
    ## Creating dataframe of distance. This will
    store the distance of each datapoint from each cluster
        for j in range(len(centroid)):
    ##This for loop is for finding distance from each centroid

            a=pd.DataFrame([np.sqrt(np.sum(np.squ
            are(df[column] - centroid.iloc[j][column]), axis=1))]).T


            distance=pd.concat([distance,a],axis=1)
    ## Distance_min stores the minimum distance of each point from all the centroids
        distance_min=distance.min(axis=1)
    ##Calculates probability for each datapoint
        probability = distance_min /
        distance_min.sum()
    ## Selecting the next centroid based on the
    probability which is proportional to find square distance
        new_centroid = pd.DataFrame(df.iloc[np.random.choice(len(df),p=probability)]).T
```

```
        centroid=pd.concat([centroid,new_centroid
        ],ignore_index=True)
        centroid.index.name='Label'
    ## Concatenated the centroid dataframe with
    new centroid and loop continues until all k centroids are initialized randomly
    return centroid.T



def assign_labels(df, centroids):
    """
    Function to calculate the closest centroid label for each row in a dataframe.
    Input:
        - df: pandas dataframe with the data
        - centroids: pandas dataframe with the centroids as columns and index as label
    Output:
        - distances.idxmin(axis=1): pandas series with the label of the closest centroid for each rou
    """
    distances = centroids.swifter.apply(lambda
    x: np.sqrt(((df - x) ** 2).sum(axis=1))) #
    Calculate the Euclidean distance between each row in df and each centroid
    return distances.idxmin(axis=1) # Get the
    index of the minimum distance, which
    corresponds to the label of the closest centroid


def new_centroids(df_label, df1):
    """
    Function to calculate the new centroids based on the current labels of the rows.
    Input:
        - df_label: pandas series with the label
        of the closest centroid for each row in df1
        - df1: pandas dataframe with the data
    Output:
        - new_centroids.T: pandas dataframe with
        the new centroids as columns and index as feature name
    """
    joined_df = df1.join(df_label)
    joined_df.rename(columns={0: 'Label'},
    inplace=True) # Rename the column with the label
    # Calculate the mean of the rows with the same label
    return joined_df.groupby('Label').mean().T #
    Transpose the dataframe to have the new centroids as columns and index as feature
    name




def
error_clusters(df_new_centroids,df1,df_label):
    """
    Calculate the error rate of each cluster.
```

```
    Args:
    - df_label (pandas.DataFrame): the label of the nearest centroid for each data point.
    - df1 (pandas.DataFrame): the dataset.
    - df_new_centroids (pandas.DataFrame): The
    new centroids computed in the current iteration.

    Returns:
    - error_rate (float): the total error rate of all clusters.
    """


    #Calculate mean value

    mean_centroid=df1.groupby('readmitted').mean().reset_index()
    # Transpose the new centroids dataframe and reset the index
    new_centroids= df_new_centroids.T
    # Get the columns of the data dataframe
    columns = df1.columns

    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):   #### centroid
        s=[]
        for j in range(len(mean_centroid)): ### mean centroid
        # Compute the distance between each data point and its assigned centroid
            distance = np.sum(np.square(mean_centroid[mean_c
            entroid['readmitted']==j][columns] - new_centroids.iloc[i][columns]),
            axis=1)
            s.append(distance.iloc[0])
        sse.append(s)
    ## key  is the cluster number and value is the merged value
    merge_label=pd.DataFrame(sse).idxmin(axis=1).to_dict()
    ## Merging cluster based on the target variable
    df_label[0]=df_label[0].replace(merge_label)

    df1 = df1.join(df_label) # add the label
    column to the dataset
    df1.rename(columns={0: 'Label'},
    inplace=True) # rename the label column
    error_list = []
    for i in df1['Label'].value_counts().index:
        df_cluster = df1[df1['Label'] == i]
        # filter the dataset to include only the
        data points in the current cluster
        y =
        len(df_cluster[df_cluster['readmitted'] == 1])
        # count the number of data points in the current cluster that were readmitted
        n = len(df_cluster[df_cluster['readmitted'] == 0])
        # count the number of data points in the current cluster that were not readmitted
        if y == 0 and n == 0:
            error = 0
        else:
```

```
            error = n / (n + y)
            # calculate the error rate of the current cluster
        error_list.append(error)
    return round(sum(error_list),4)


def sum_of_square_error(new_centroids, data, labels):
    """
    Computes the sum of squared errors between the data points and their assigned centroids.

    Args:
    new_centroids (DataFrame): The new centroids computed in the current iteration.
    data (DataFrame): The input data points.
    labels (DataFrame): The labels assigned to each data point.

    Returns:
    The sum of squared errors.
    """
    # Transpose the new centroids dataframe and reset the index
    new_centroids = new_centroids.T.reset_index()
    # Get the columns of the data dataframe
    columns = data.columns
    # Join the data dataframe and the labels dataframe
    data = data.join(labels)
    # Rename the '0' column of the labels dataframe to 'Label'
    data.rename(columns={0:'Label'}, inplace=True)
    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):
        distance =
        np.sum(np.square(data[data['Label']==i][columns] - new_centroids.iloc[i]
        [columns]), axis=1)
        sse.append(sum(distance))
    # Return the sum of squared errors
    return sum(sse)



def kmeans_plus_plus(df1,k,tou):
    """
    Function to run the K-means plus plus algorithm.
    Input:
        - df1: pandas dataframe with the data
        - k: integer number of clusters
        - tou: float tolerance level to stop the algorithm

    """
    start_time=time.time()
    iteration=0
    centroids=initialize_centroids_plus(df1,k)
    initial_list_of_columns=centroids.columns.to_
    list()
```

```
    while True:


        #Assigning labels to randomly generated centroids
        df_label=assign_labels(df1,centroids)
        df_label=pd.DataFrame(df_label)
        #Calculating new centroids

        df_new_centroids=new_centroids(df_label,d
        f1)
        new_list_of_columns=df_new_centroids.colu
        mns.to_list()
        #Keeping the number of clusters same
        for i in initial_list_of_columns:
            if i not in new_list_of_columns:
                df_new_centroids[i]=centroids[i]

        #Calculate tao
        distance = []
        for col in centroids.columns:
            col_distance = euclidean(centroids[col],
            df_new_centroids[col])
            distance.append(col_distance)
        tao_calculated=float(sum(distance))/k #Used the formula provided for calculating Tao




        end_time=time.time()
        if iteration>100:
            print("Iteration exceeded")
            error=error_clusters(df_new_centroids,df1,df_label)
            sse=
            sum_of_square_error(df_new_centroids,
            df1,df_label)
            return error,sse,end_time-start_time

        if tao_calculated<tou or iteration >100:
        #if the convergence is met, kmeans will
        stop  or else if the convergence is
        never met, after 100 iteration code will stop

            error=error_clusters(df_new_centroids,df1,df_label)
            # otherwise indefinite loop
            sse= sum_of_square_error(df_new_centroids,df1,df_label)

            return error,sse,end_time-start_time
            break
        else:
            centroids= df_new_centroids
            # In case we need more iterations,
            the centroids calculated at this step acts as input
```

```
        iteration+=1

error_matrix_plus_diab=[]
for i in range(2,6):
    for j in range(1,21):
        error,sse,run_time=kmeans_plus_plus(df_cl
        eaned_dia,i,10)
        error_matrix_plus_diab.append([i,j,error,
        sse,run_time])
error_df_plus_diab=
pd.DataFrame(error_matrix_plus_diab,columns=['number_of_cluster', 'iteration',
'error','sse','run_time'])

import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
x=error_df_plus_diab['number_of_cluster'].value_c
ounts().index
y1=error_df_plus_diab.groupby(['number_of_cluster
']).mean()['error']
ax1.plot(x, y1, color='tab:blue')
ax1.set_xlabel('Number of clusters')
ax1.set_ylabel('Error', color='tab:blue')

ax2 = ax1.twinx()
y2 = error_df_plus_diab.groupby(['number_of_cluster']).mean()['sse']
ax2.plot(x, y2, color='tab:orange')
ax2.set_ylabel('SSE', color='tab:orange')
plt.title('Error and SSE (mean)for each cluster kmeans ++')
plt.xticks(range(2, 6))
plt.show()


import seaborn as sns
plt.figure(figsize=(6, 10))
sns.boxplot(x=error_df_plus_diab['number_of_clust
er'],y=error_df_plus_diab['error'])
plt.title('Box Plot of error for K means ++')
plt.show()

import seaborn as sns
sns.boxplot(x=error_df_plus_diab['number_of_clust
er'],y=error_df_plus_diab['run_time'])
plt.title('Box Plot of runtime for K means ++')
plt.show()


import seaborn as sns

fig, ax = plt.subplots(figsize=(12, 15))

sns.boxplot(x='number_of_cluster', y='run_time', hue='algo',
            data=run_time_diab,ax=ax);
```

```
plt.title('Box Plot of run time for K means, K means with SSE,K means ++')
plt.show()




import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm

bag_of_words_df=pd.read_csv('bag_of_words.csv')
comments_df=pd.read_csv('cleaned_data.csv')

bag_of_words_df=bag_of_words_df.join(comments_df[
'editorsSelection'].replace({True:1,False:0}))

bag_of_words_df.columns

'''
Index(['actual', 'administr', 'agre', 'allow', 'alreadi', 'also', 'alway',
       'america', 'american', 'anoth',
       ...
       'without', 'wonder', 'word', 'work', 'world', 'would', 'ye', 'year',
       'yet', 'editorsSelection'],
      dtype='object', length=209)
'''



import pandas as pd
import numpy as np
import swifter
import time

from scipy.spatial.distance import euclidean

def initialize_centroids_plus(df,k):
    """
    Function to calculate the random centroid
    using kmeans ++ technique.
    Input:
        - df: pandas dataframe with the data
        - k: number of clusters
    Output:
        - centroid.T: pandas dataframe with all
```

```
        centroids initialized
    """
    #Initialize random centroids from dataset
    centroid = []
    centroid.append(df.apply(lambda x: float(x.sample())))
    centroid=pd.DataFrame(centroid)
    column=centroid.columns.to_list()
    ##Randomly created first centroid from the domain of each column in the dataframe

    for i in range(1,k):
    ## The above for loop is for generating k-1
    clusters as first random cluster is already generated
        distance=pd.DataFrame()
    ## Creating dataframe of distance. This will
    store the distance of each datapoint from each cluster
        for j in range(len(centroid)):
    ##This for loop is for finding distance from each centroid
            a=pd.DataFrame([np.sqrt(np.sum(np.squ
            are(df[column] - centroid.iloc[j][column]), axis=1))]).T

            distance=pd.concat([distance,a],axis=1)
    ## Distance_min stores the minimum distance of each point from all the centroids
        distance_min=distance.min(axis=1)
    ##Calculates probability for each datapoint
        probability = distance_min /
        distance_min.sum()
    ## Selecting the next centroid based on the
    probability which is proportional to find square distance
        new_centroid = pd.DataFrame(df.iloc[np.random.choice(len
        (df),p=probability)]).T

        centroid=pd.concat([centroid,new_centroid
        ],ignore_index=True)
        centroid.index.name='Label'
    ## Concatenated the centroid dataframe with
    new centroid and loop continues until all k centroids are initialized randomly
    return centroid.T



def assign_labels(df, centroids):
    """
    Function to calculate the closest centroid
    label for each row in a dataframe.
    Input:
        - df: pandas dataframe with the data
        - centroids: pandas dataframe with the
        centroids as columns and index as label
    Output:
        - distances.idxmin(axis=1): pandas
        series with the label of the closest
        centroid for each row in df
    """
```

```
    distances = centroids.swifter.apply(lambda
    x: np.sqrt(((df - x) ** 2).sum(axis=1)))
    #    Calculate the Euclidean distance between each row in df and each centroid
    return distances.idxmin(axis=1)
    # Get the index of the minimum distance, which corresponds to the label of the
    closest centroid


def new_centroids(df_label, df1):
    """
    Function to calculate the new centroids based on the current labels of the rows.
    Input:
        - df_label: pandas series with the label
        of the closest centroid for each row in df1
        - df1: pandas dataframe with the data
    Output:
        - new_centroids.T: pandas dataframe with
        the new centroids as columns and index as feature name
    """
    joined_df = df1.join(df_label)
    joined_df.rename(columns={0: 'Label'},
    inplace=True) # Rename the column with the label
    # Calculate the mean of the rows with the same label
    return joined_df.groupby('Label').mean().T
    # Transpose the dataframe to have the new centroids as columns and index as feature
    name




def error_clusters(df_new_centroids,df1,df_label):
    """

    Calculate the error rate of each cluster.

    Args:
    - df_label (pandas.DataFrame): the label of the nearest centroid for each data point.
    - df1 (pandas.DataFrame): the dataset.
    - df_new_centroids (pandas.DataFrame): The
    new centroids computed in the current iteration.

    Returns:
    - error_rate (float): the total error rate of all clusters.
    """



    #Calculate mean value
    mean_centroid=df1.groupby('editorsSelection')
    .mean().reset_index()
    # Transpose the new centroids dataframe and
    reset the index
    new_centroids= df_new_centroids.T
```

```
    # Get the columns of the data dataframe
    columns = df1.columns

    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):   ####
    centroid
        s=[]
        for j in range(len(mean_centroid)): ### mean centroid
        # Compute the distance between each data point and its assigned centroid
            distance =
            np.sum(np.square(mean_centroid[mean_c
            entroid['editorsSelection']==j]
            [columns] - new_centroids.iloc[i][columns]), axis=1)
            s.append(distance.iloc[0])
        sse.append(s)
    ## key  is the cluster number and value is the merged value
    merge_label=pd.DataFrame(sse).idxmin(axis=1).
    to_dict()
    ## Merging cluster based on the target variable
    df_label[0]=df_label[0].replace(merge_label)

    df1 = df1.join(df_label) # add the label column to the dataset
    df1.rename(columns={0: 'Label'}, inplace=True) # rename the label column
    error_list = []
    for i in df1['Label'].value_counts().index:
        df_cluster = df1[df1['Label'] == i]
        # filter the dataset to include only the data points in the current cluster
        y = len(df_cluster[df_cluster['editorsSelection'] == 1])
        # count the number of data points in the current cluster that were readmitted
        n = len(df_cluster[df_cluster['editorsSelection'] == 0])
        # count the number of data points in the current cluster that were not readmitted
        if y == 0 and n == 0:
            error = 0
        else:
            error = n / (n + y)
            # calculate the error rate of the current cluster
        error_list.append(error)
    return round(sum(error_list),4)


def sum_of_square_error(new_centroids, data, labels):
    """
    Computes the sum of squared errors between the data points and their assigned centroids.

    Args:
    new_centroids (DataFrame): The new centroids computed in the current iteration.
    data (DataFrame): The input data points.
    labels (DataFrame): The labels assigned to each data point.

    Returns:
    The sum of squared errors.
    """
```

```
    # Transpose the new centroids dataframe and reset the index
    new_centroids = new_centroids.T.reset_index()
    # Get the columns of the data dataframe
    columns = data.columns
    # Join the data dataframe and the labels dataframe
    data = data.join(labels)
    # Rename the '0' column of the labels dataframe to 'Label'
    data.rename(columns={0:'Label'}, inplace=True)
    sse = []
    # Compute the distance between each data point and its assigned centroid
    for i in range(len(new_centroids)):
        distance = np.sum(np.square(data[data['Label']==i]
        [columns] - new_centroids.iloc[i][columns]), axis=1)
        sse.append(sum(distance))
    # Return the sum of squared errors
    return np.nansum(sse)



def kmeans_plus_plus(df1,k,tou):
    """
    Function to run the K-means plus plus algorithm.
    Input:
        - df1: pandas dataframe with the data
        - k: integer number of clusters
        - tou: float tolerance level to stop the algorithm

    """
    start_time=time.time()
    iteration=0
    centroids=initialize_centroids_plus(df1,k)
    initial_list_of_columns=centroids.columns.to_list()


    while True:



        #Assigning labels to randomly generated centroids
        df_label=assign_labels(df1,centroids)
        df_label=pd.DataFrame(df_label)
        #Calculating new centroids

        df_new_centroids=new_centroids(df_label,df1)

        new_list_of_columns=df_new_centroids.colu
        mns.to_list()
        #Keeping the number of clusters same
        for i in initial_list_of_columns:
            if i not in new_list_of_columns:
                df_new_centroids[i]=centroids[i]

        #Calculate tao
        distance = []
```

```
        for col in centroids.columns:
            col_distance =
            euclidean(centroids[col], df_new_centroids[col])

            distance.append(col_distance)
        tao_calculated=float(sum(distance))/k #Used the formula provided for calculating Tao




        end_time=time.time()
        if iteration>100:
            print("Iteration exceeded")

            error=error_clusters(df_new_centroids
            ,df1,df_label)
            sse= sum_of_square_error(df_new_centroids,
            df1,df_label)
            return error,sse,end_time-start_time

        if tao_calculated<tou or iteration >100:   #if the convergence is met, kmeans
        will stop  or else if the convergence is never met, after 100 iteration code will stop
            error=error_clusters(df_new_centroids
            ,df1,df_label)      # otherwise indefinite loop
            sse= sum_of_square_error(df_new_centroids,
            df1,df_label)
            return error,sse,end_time-start_time
            break
        else:
            centroids= df_new_centroids  # In
            case we need more iterations, the centroids calculated at this step
            acts as input

        iteration+=1


error_matrix_plus_ny=[]
for i in range(2,6):
    for j in range(1,21):
        error,sse,run_time=kmeans_plus_plus(bag_o
        f_words_df,i,10)
        error_matrix_plus_ny.append([i,j,error,ss
        e,run_time])
error_df_plus_ny= pd.DataFrame(error_matrix_plus_ny,columns=
['number_of_cluster', 'iteration', 'error','sse','run_time'])
error_df_plus_ny.to_csv('kmeans_++.csv')




import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
```

```
x=error_df_plus_ny['number_of_cluster'].value_counts().index
y1=error_df_plus_ny.groupby(['number_of_cluster']).mean()['error']
ax1.plot(x, y1, color='tab:blue')
ax1.set_xlabel('Number of clusters')
ax1.set_ylabel('Error', color='tab:blue')

ax2 = ax1.twinx()
y2 = error_df_plus_ny.groupby(['number_of_cluster']).mean()['sse']
ax2.plot(x, y2, color='tab:orange')
ax2.set_ylabel('SSE', color='tab:orange')
plt.title('Error and SSE (mean)for each cluster kmeans ++ for new york data')
plt.xticks(range(2, 6))
plt.show()


import seaborn as sns
plt.figure(figsize=(6, 10))
sns.boxplot(x=error_df_plus_ny['number_of_cluster'],y=error_df_plus_ny['error'])
plt.title('Box Plot of error for K means ++ for new york data')
plt.show()


import seaborn as sns
sns.boxplot(x=error_df_plus_ny['number_of_cluster'],y=error_df_plus_ny['run_time'])
plt.title('Box Plot of runtime for K means ++ for new york data')
plt.show()
```

Discussion of Findings

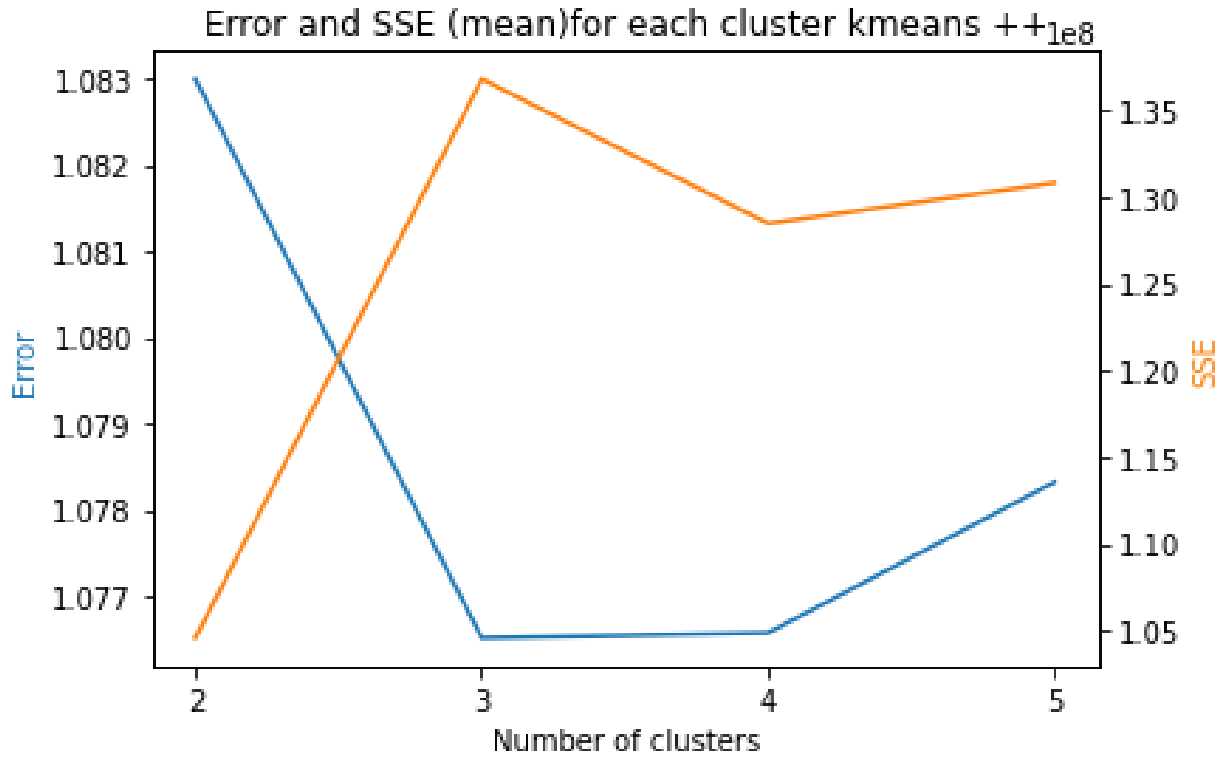
Answer here...
I have implemented the kmeans plus plus algorithm for number of clusters ranging from 2 to 5. The method is similar to kmeans llyod with different centroid initialization. Here the first centroid is chosen randomly from the domain of data. For rest k-1 centroids, we follow a different iterative approach. The second cluster is initialized based on the first centroid. The third centroid requires data of first two centroids. The methodolgy includes,selecting the first centroid randomly from domain of data. Then second cluster is at the farthest distance from that cluster. In this way iteratively we initialize k centroids. As the centroid initialization is not random, it is expected that the inter cluster distance will be more and intra cluster distance bewteen points and centroid will be less.
The K-means++ algorithm selects the centroids in following way
Step1: Choosing the first centroid at random from the data points.
Step2: For each remaining data point, computing its distance to the nearest centroid that has already been chosen.
Step3: Selecting the next centroid randomly from the remaining data points, with probability proportional to the squared distance to the nearest centroid. Repeating steps 2-3 until all K centroids have been chosen.
The main idea behind K-means++ initialization is to select centroids that are well spread out across the data points. By selecting the next centroid from the remaining data points with a probability that is proportional to the squared distance to the nearest centroid, K-means++ initialization ensures that data points that are far away from existing centroids are more likely to be selected as new centroids. Hence the sum of square error is expected to be less. Rest the stopping conditions and other steps are similar to that of kmeans llyods
Have ran this algorithm for diabetes data set and new york dataset sample of comments. Have performed required EDA on datasets.

Diabetes Dataset
From the line plot below for error and sum of square error with number of clusters it can be seen that error graph balances out with increase in the number of clusters i.e from 2 to 5. The error is decreasing with increase in number of clusters. At end there is slight increase in the error.The sum of square error shows an initial small increase with increase in the number of clusters and then decreases with increase in clusters. From the box plot for error wrt number of clusters, shows the median error for each cluster shows a gradual decrease . The box plot of runtime shows that with increase in number of clusters the median run time also increases with increase in number of clusters.
When the runtime of kmeans plus plus is compared to runtime of kmeans with sse convergence and kmeans llyod it can be seen that kmeans plus plus have more median running time than other two variations.
New York Dataset
From the below line graph for error and sum of square error vs number of clusters it can be seen that error is remaining almost constant over the number of clusters. One of the reasons for this could be that as this dataset was comment body, after pre processing the dataset became sparse as the columns are mostly the words. So the column values are mostly 0 or 1 and hence the distance is almost same. We have used distance metric to merge and calculate error. If the data is sparse, meaning that most of the columns contain mostly 0s, then the k-means algorithm may not be able to form distinct clusters. In this case, the error may remain constant regardless of the number of clusters chosen. Also it is possible that homogeneous clusters may be formed.Binary features can have a tendency to form homogeneous clusters. In other words, some clusters may contain mostly 1s and others may contain mostly 0s. In this case, the k-means algorithm may not be able to form distinct clusters, and the error may remain constant regardless of the number of clusters chosen. Even the box plot for error shows the same ,remains constant with increase in number of clusters. The Sum of square error decrease with increase in number of clusters as shown in the line graph. The box plot for run time shows that the run time increases with increase in number of clusters. The run time of box plot of kmeans plus plus is compared with k means sse convergence and kmeans llyod, it can be seen that the median run time for kmeans ++ is comparable to the kmeans llyod. The meadian run time of k means with sse convergence is more than other two variations, showing it takes more time to converge if we use sum of threshold as stopping condition. The run time of kmeans Plus Plus increases with increase in number of clusters as compared to kmeans llyods. After k=3 the run time of kmeans Plus Plus is more than that of kmeans llyod as the centroid initialization is done in series for this method, hence the time increases with increase in number of clusters. K-means++ initialization can help improve the cluster quality as the initialization is not complete random. kmeans plus plus is more expensive, so depending upon the computing power we can decide to use kmeans plus plus or not.

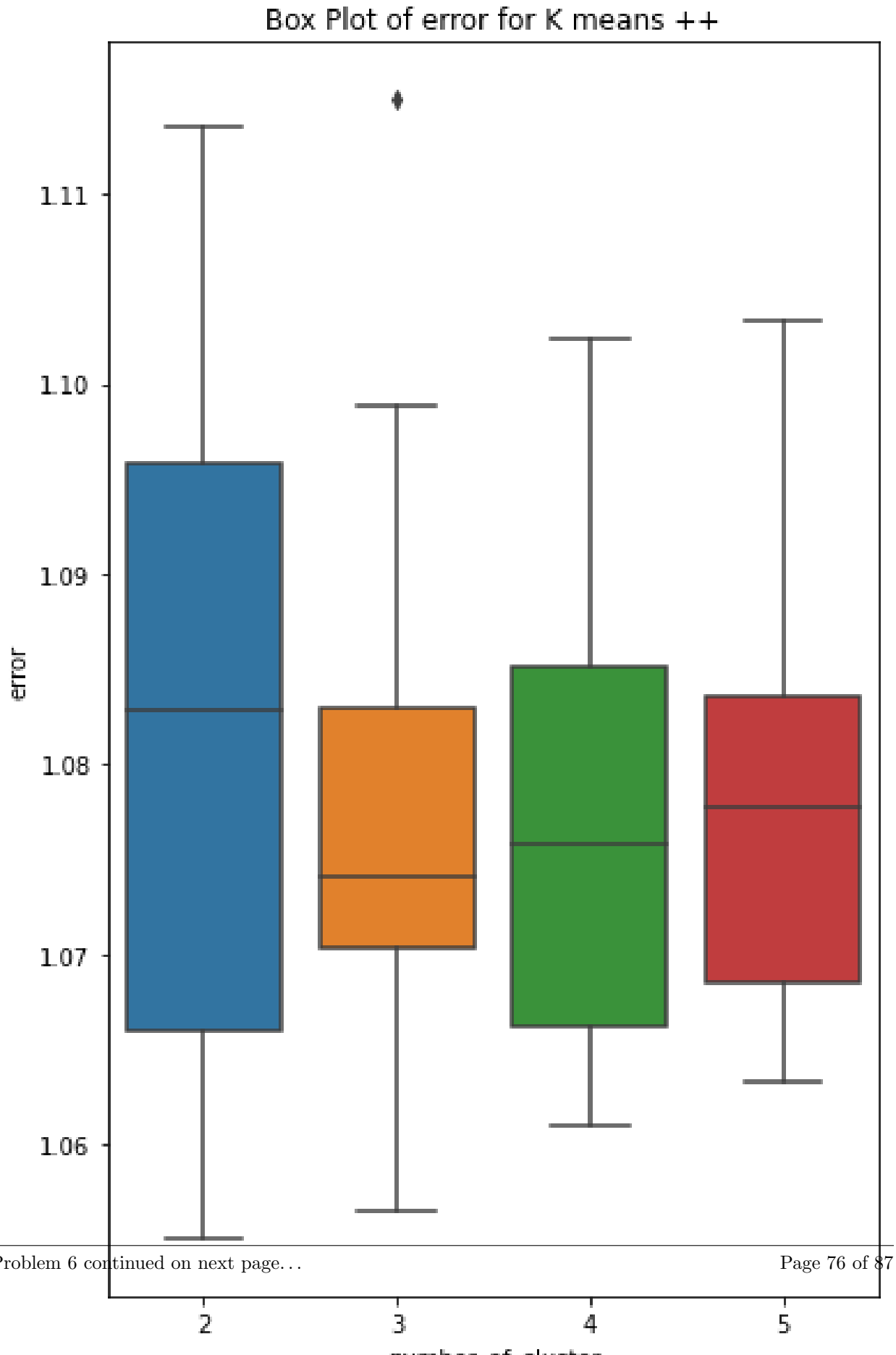
Plots

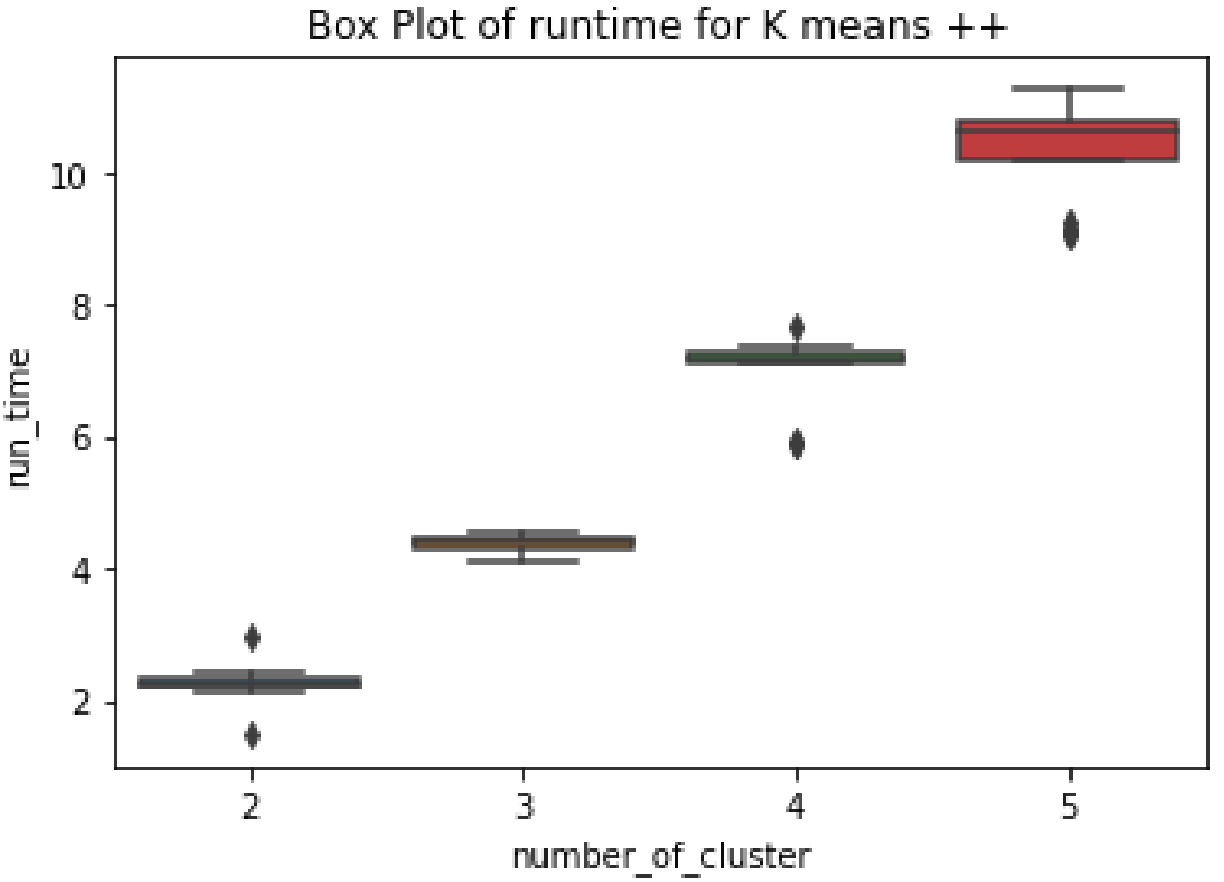
Place images here with suitable captions.

Plots for Diabetes Dataset

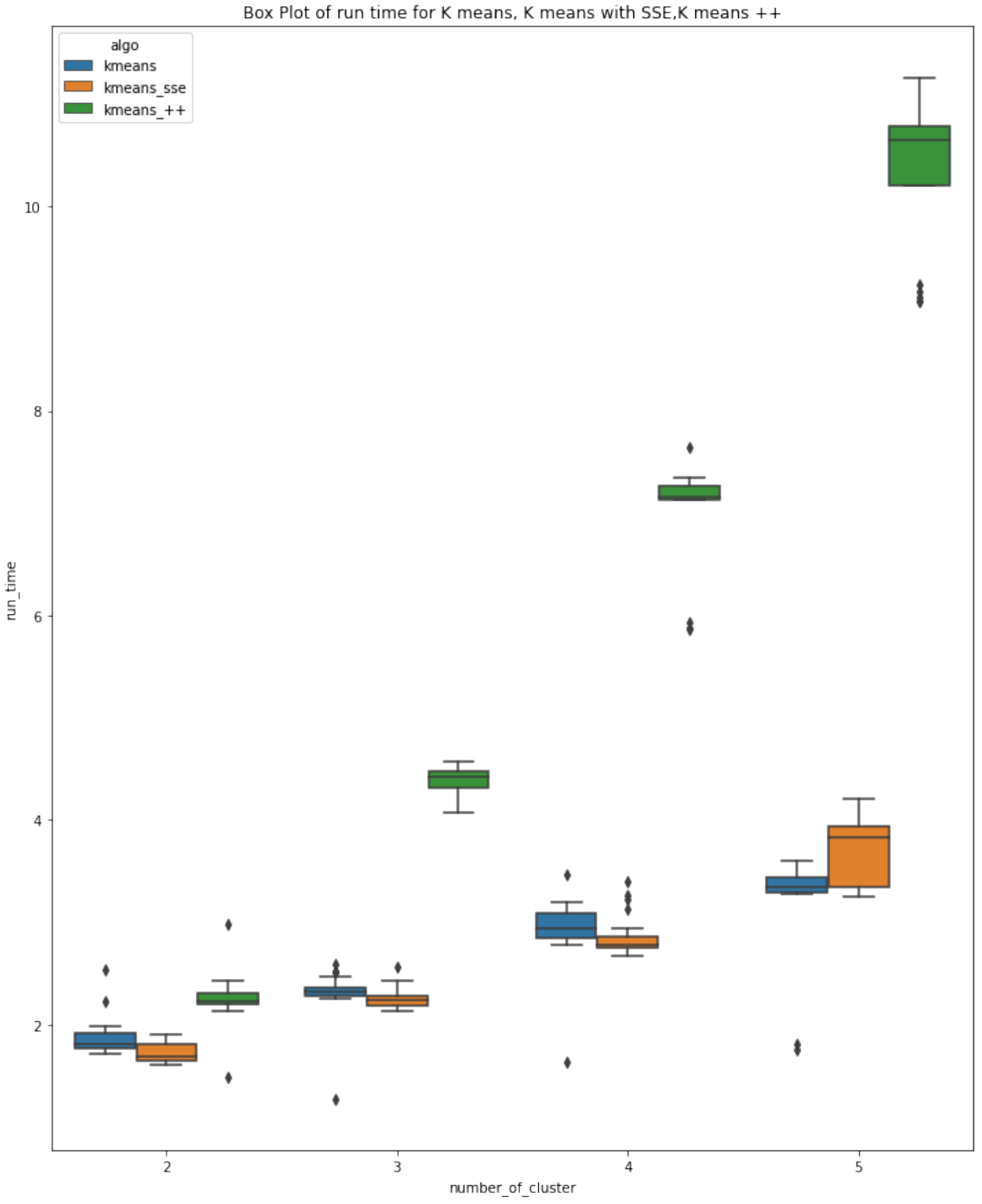


(25)


Box Plot of error for K means ++
error
1.11
1.10
1.09
1.08
1.07
1.06
2
3
4
5

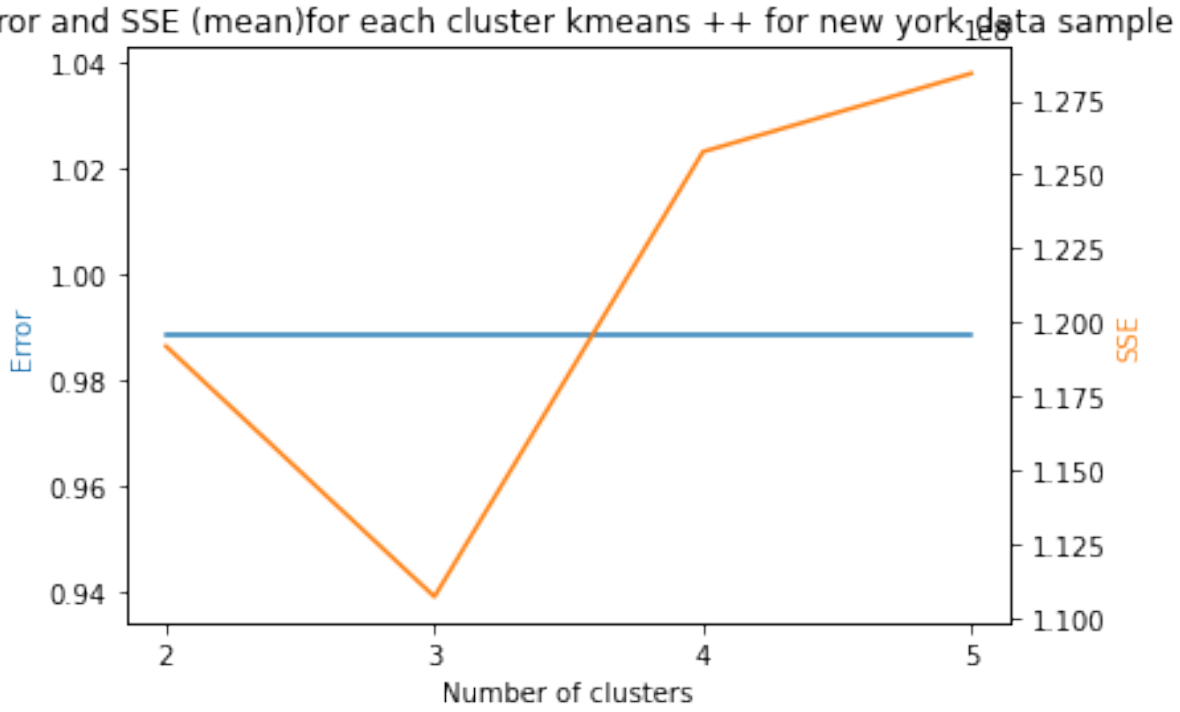

Box Plot of runtime for K means ++
run_time
10
8
6
4
2
2
3
4
5
number_of_cluster

(27)

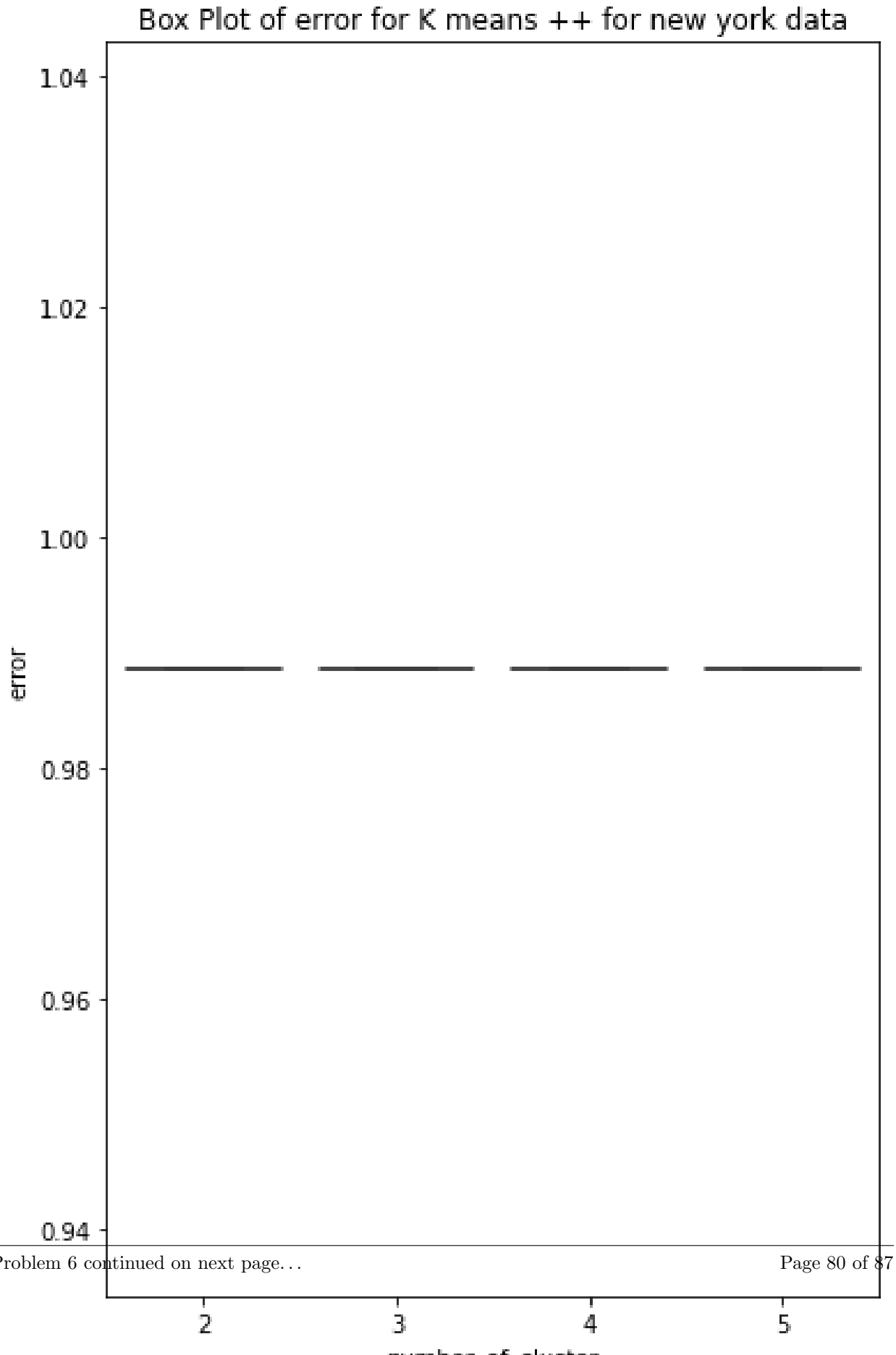


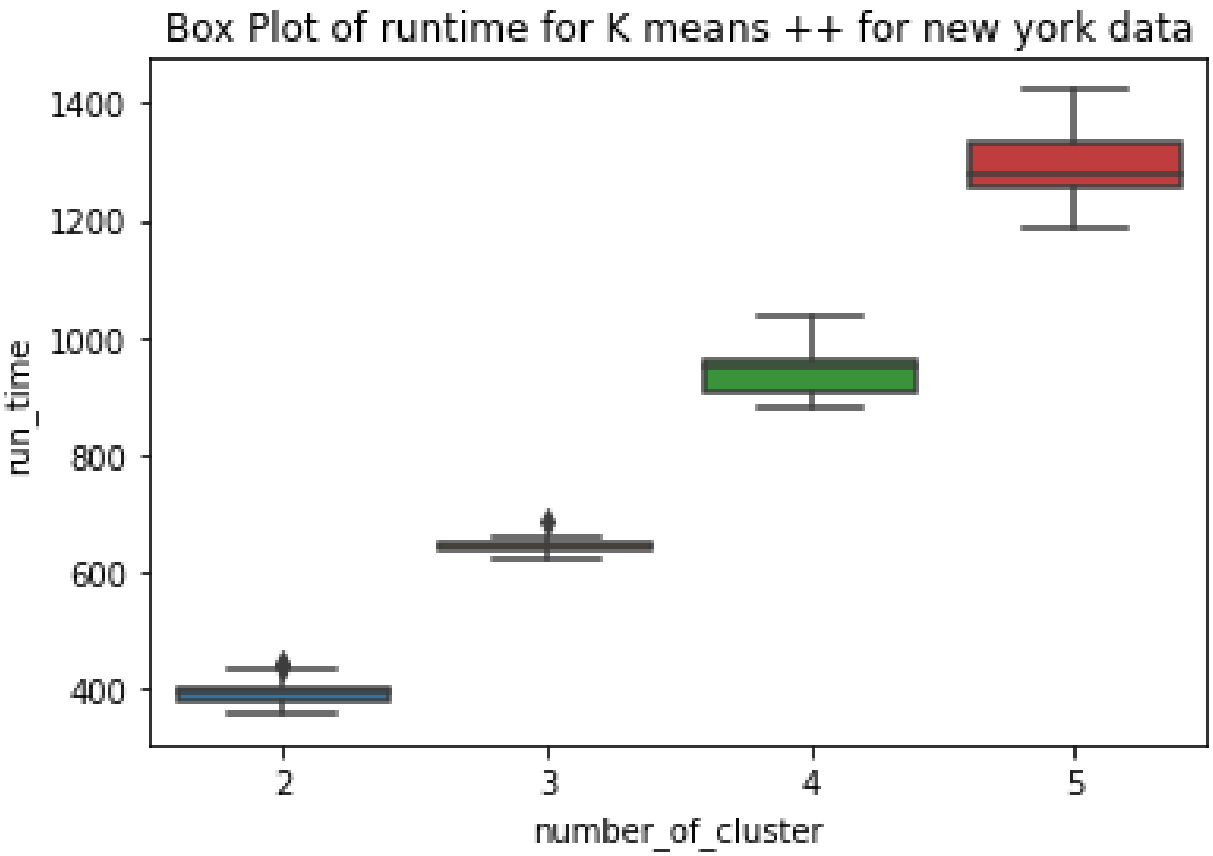
(28)

Plots for New York Times Comment dataset

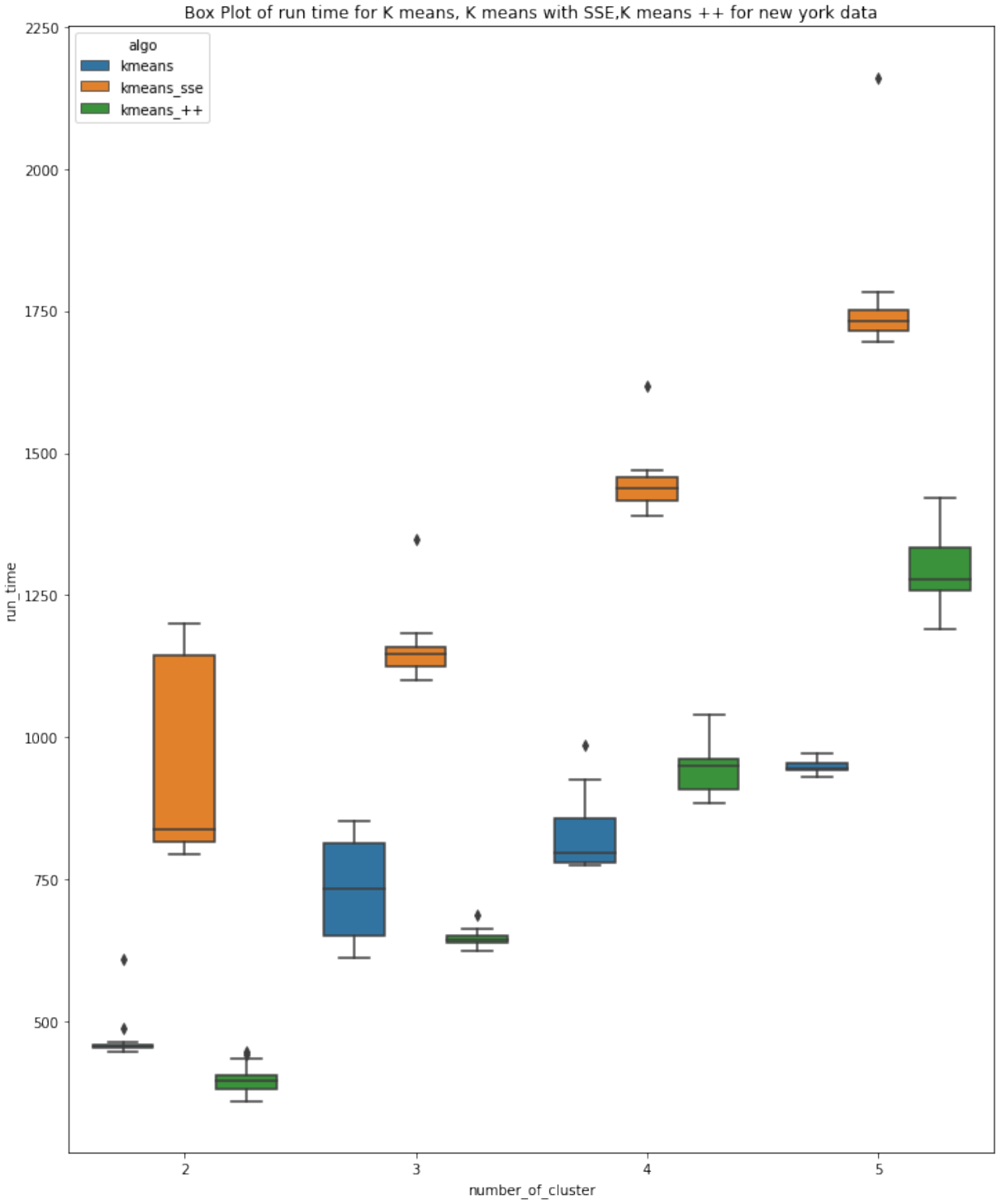

Error and SSE (mean)for each cluster kmeans ++ for new york data sample
1e8
1.04
1.02
1.00
0.98
0.96
0.94
Error
1.275
1.250
1.225
1.200
1.175
1.150
1.125
1.100
SSE
2
3
4
5
Number of clusters

(29)


Box Plot of error for K means ++ for new york data
1.04
1.02
1.00
0.98
0.96
0.94
error
2
3
4
5


Box Plot of runtime for K means ++ for new york data
run_time
1400
1200
1000
800
600
400
2
3
4
5
number_of_cluster

(31)


Box Plot of run time for K means, K means with SSE,K means ++ for new york data
algo
kmeans
kmeans_sse
kmeans_++
run_time
number_of_cluster
2250
2000
1750
1500
1250
1000
750
500
2
3
4
5

(32)

Problem 7

In this question, you are asked to make use of the *R/Python* libraries for k-means. The elbow technique is used to determine optimal cluster number. Find the optimal cluster number for the Diabetes and New York Times Comments data sets using elbow method (for $2 \leq k \leq 15$). Provide plots that show the total SSE for each k. Discuss your results [**20 points**].

R/Python script

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

```
# Sample Python Script With Highlighting
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Define a range of k values to test
k_range = range(2, 16)

# Create an empty list to hold the Sum of
Squared Distances (SSD) for each k value
ssd_values = []

# Calculate SSD for each k value and append it to ssd_values
for k in k_range:
    # Create a KMeans object with the current k value
    kmeans = KMeans(n_clusters=k, random_state=42)
    # Fit the KMeans object to the data
    kmeans.fit(df_cleaned_dia)
    # Append the SSD value to ssd_values
    ssd_values.append(kmeans.inertia_)

# Plot the elbow curve to find the optimal k value
plt.plot(k_range, ssd_values)
plt.title('Elbow Curve')
plt.xlabel('Number of Clusters ')
plt.ylabel('Sum of Squared Distances ')
plt.show()


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
```

```
from tqdm import tqdm
import swifter

bag_of_words_df=pd.read_csv('bag_of_words.csv')
bag_of_words_df.fillna(0,inplace=True)
comments_df=pd.read_csv('cleaned_data.csv')

bag_of_words_df=bag_of_words_df.join(comments_df['editorsSelection'].replace({True:1,False:0}))

bag_of_words_df.columns

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Define a range of k values to test
k_range = range(2, 16)

# Create an empty list to hold the Sum of Squared Distances (SSD) for each k value
ssd_values = []

# Calculate SSD for each k value and append it to ssd_values
for k in k_range:
    # Create a KMeans object with the current k
    value
    kmeans = KMeans(n_clusters=k,
    random_state=42)
    # Fit the KMeans object to the data
    kmeans.fit(bag_of_words_df)
    # Append the SSD value to ssd_values
    ssd_values.append(kmeans.inertia_)

# Plot the elbow curve to find the optimal k value
plt.plot(k_range, ssd_values)
plt.title('Elbow Curve')
plt.xlabel('Number of Clusters ')
plt.ylabel('Sum of Squared Distances ')
plt.show()
```

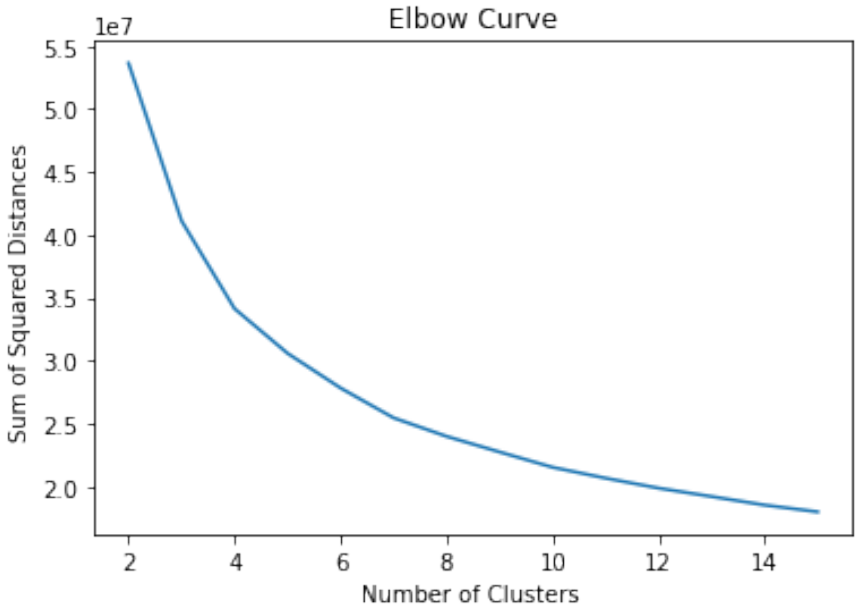
Discussion of Findings

Answer here...
For the diabetes dataset, the elbow plot shows a decrease in error with an increase in the number of clusters. We can see that at k=4, the sse decreased as compared to that of k=2. So we can select k=4 as optimal number of cluster. But if we see further k, the slope from k=10 is becoming constant. From k=4 till 10 there is decrease and from k=10 the slope almost remains constant. Depending upon the use case and information loss that can be accepted we can select k. In this graph, selecting k=10 and then slope becomes constant so there will be hardly any information loss as compared to that of k. So the optimal value can be 10. On similar grounds, for new york dataset, the optimal value of k starts from 4 as we can see elbow there. But the sse is decreasing linearly and after 12 it seems constant. Hence k=12 can be the optimal value. The elbow

method is a common technique used to determine the optimal number of clusters. However, it is possible elbow curve does not provide the correct or interpretable output. Similar to the new york dataset. In such cases, we can use domain knowledge or techniques like Silhouette analysis. Domain knowledge plays a vital role here. The elbow method can be used as one of the method to determine number of clusters coupled with domain knowledge.

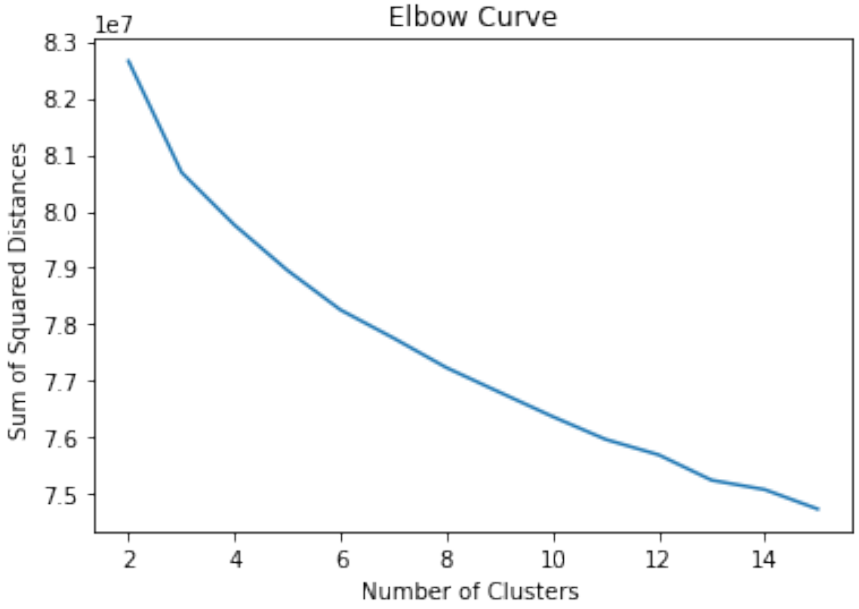
Plots

Place images here with suitable captions.
Plot for Diabetes Dataset



(33)

Plots for New York Times Comment dataset


Elbow Curve
1e7
8.3
8.2
8.1
8.0
7.9
7.8
7.7
7.6
7.5
Sum of Squared Distances
2
4
6
8
10
12
14
Number of Clusters

(34)

Extra credit

This part is optional.

1. Ball-k-means calculates the distance of a data point from the centroid to find the annular region in which the data point resides. The annular region helps determine which neighbor centroids should be included in distance computations. This improves the run-time over earlier approaches by avoiding expensive computations. Run your program, $C_{k_{ball}}$, against the Diabetes and New York Times Comments data sets. Report the total error rates for $k = 2, \ldots 5$ for 20 runs each for both data sets. Moreover, compare C_k, $C_{k_{SSE}}$, C_{k++} and $C_{k_{ball}}$'s run time for $k = 2, \ldots 5$ for 20 runs using both data sets. Presenting the results that are easily understandable. Plots are generally a good way to convey complex ideas quickly, i.e., box plot. Discuss your results [**30 points**].

R/Python script

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

```python
# Sample Python Script With Highlighting
```

Discussion of Findings

Answer here...

Plots

Place images here with suitable captions.

2. The student who has the fastest implementation of all four clustering algorithms will receive extra 20 points [**20 points**].

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-hw4-everything.pdf` which you will get after compiling your .tex file.

2. A .zip file with the name `yourname-hw4.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.