Business Case 9 : Scaler kmeans

Submission by Aditya Vyas

```
# importing all required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import scipy.stats as stats
from matplotlib.gridspec import GridSpec

from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import QuantileTransformer
from sklearn.decomposition import PCA

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import pairwise_distances_argmin_min
from sklearn.metrics.pairwise import euclidean_distances
from joblib import Parallel, delayed

from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

import warnings
warnings.filterwarnings("ignore")



#importing the dataset
df = pd.read_csv('D:/Learning/Scalar -
DataScience/Datasets/scaler_clustering.csv')
```

Section 1 : EDA

```
# checking the shape of the data
df.shape
```

```
(205843, 7)
```

```
# viewing the dataset
df.head()
```

```
   Unnamed: 0               company_hash  \
0           0            atrgxnnt xzaxv
1           1  qtrxvzwt xzegwgbb rxbxnta
2           2              ojzwnvwnxw vx
3           3                  ngpgutaxv
4           4                 qxen sqghu

                                          email_hash  orgyear      ctc
\
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...   2016.0  1100000

1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...   2018.0   449999

2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...   2015.0  2000000

3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58...   2017.0   700000

4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...   2017.0  1400000


         job_position  ctc_updated_year
0               Other            2020.0
1  FullStack Engineer            2019.0
2    Backend Engineer            2020.0
3    Backend Engineer            2019.0
4  FullStack Engineer            2019.0
```

```
# getting some additional info on the features
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 7 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Unnamed: 0        205843 non-null  int64
 1   company_hash      205799 non-null  object
 2   email_hash        205843 non-null  object
 3   orgyear           205757 non-null  float64
 4   ctc               205843 non-null  int64
 5   job_position      153279 non-null  object
 6   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(2), object(3)
memory usage: 11.0+ MB
```

```
df.describe(include = 'all')
```

```
           Unnamed: 0               company_hash  \
count   205843.000000                     205799
unique            NaN                      37299
top               NaN  nvnv wgzohrnvzwj otqcxwto
```

```
freq                 NaN                        8337
mean       103273.941786                         NaN
std         59741.306484                         NaN
min             0.000000                         NaN
25%         51518.500000                         NaN
50%        103151.000000                         NaN
75%        154992.500000                         NaN
max        206922.000000                         NaN

                                                   email_hash
orgyear  \
count                                                  205843
205757.000000
unique                                                 153443
NaN
top     bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...
NaN
freq                                                       10
NaN
mean                                                      NaN
2014.882750
std                                                       NaN
63.571115
min                                                       NaN
0.000000
25%                                                       NaN
2013.000000
50%                                                       NaN
2016.000000
75%                                                       NaN
2018.000000
max                                                       NaN
20165.000000

                 ctc      job_position  ctc_updated_year
count   2.058430e+05            153279     205843.000000
unique           NaN              1016               NaN
top              NaN  Backend Engineer               NaN
freq             NaN             43554               NaN
mean    2.271685e+06               NaN       2019.628231
std     1.180091e+07               NaN          1.325104
min     2.000000e+00               NaN       2015.000000
25%     5.300000e+05               NaN       2019.000000
50%     9.500000e+05               NaN       2020.000000
75%     1.700000e+06               NaN       2021.000000
max     1.000150e+09               NaN       2021.000000
```

```python
# chekcing if there are any NULL values
df.isnull().sum()/df.shape[0]*100
```

```
Unnamed: 0            0.000000
company_hash          0.021376
email_hash            0.000000
orgyear               0.041779
ctc                   0.000000
job_position         25.535967
ctc_updated_year      0.000000
dtype: float64
```

```
# checking if there are any duplicate rows
df.duplicated().sum()
```

```
0
```

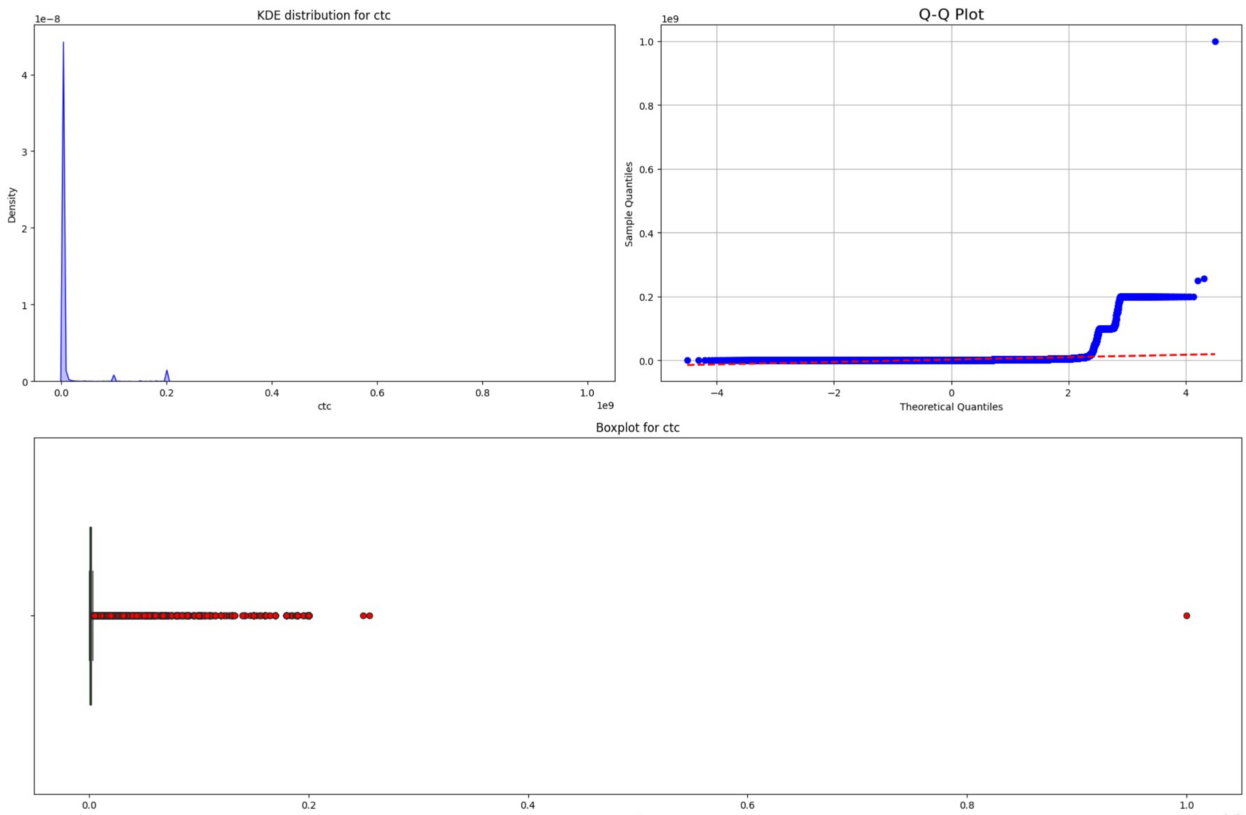
Observation set : 1

1. The first column seems to contain the index, hence can be dropped
2. There are quite a few NULL values, with maximum (~25%) in job_position
3. Unique values in email_hash are almost 75% of the total dataset, we may drop it before final kmeans step as well
4. Unique values in company_hash is ~19%
5. 'orgyear' has min as 0, we will see the distribution and see how to treat it
6. 'orgyear' has max as 20165, we will cap the max at 2024 (latest year)
7. We will apply the same rules on ctc_updated_year

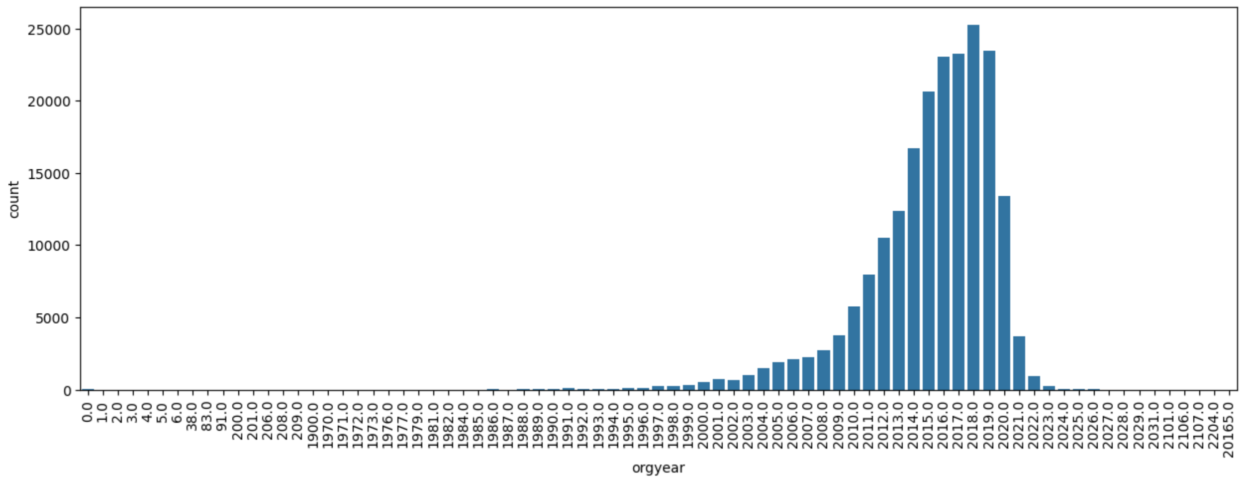
Section 2 : Visualization Part 1

```
def plot_num(data, col):
    fig= plt.figure(figsize=(18,12))
    gs = GridSpec(2,2, figure = fig)
    ax1 = fig.add_subplot(gs[0,0])
    sns.kdeplot(data, ax = ax1,fill=True, color="blue", bw_adjust=0.5)
    ax1.set_title('KDE distribution for {}'.format(col))

    ax2 = fig.add_subplot(gs[0,1])
    stats.probplot(data, dist="norm", plot=ax2)
    ax2.set_title('Q-Q Plot for {}'.format(col))
    ax2.get_lines()[1].set_color('red')
    ax2.get_lines()[1].set_linestyle('--')
    ax2.get_lines()[1].set_linewidth(2)
    ax2.set_title('Q-Q Plot', fontsize=16)
    ax2.set_xlabel('Theoretical Quantiles')
    ax2.set_ylabel('Sample Quantiles')
    ax2.grid(True)
```

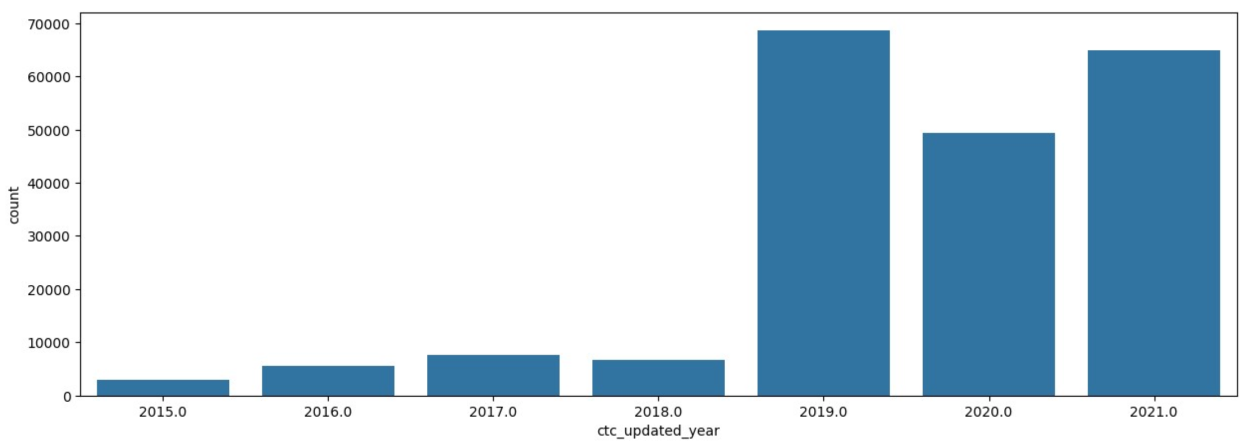
```
    ax3 = fig.add_subplot(gs[1,:])
    sns.boxplot(x = data,ax = ax3, color="green", width=0.5,
flierprops=dict(marker='o', markerfacecolor='red'))
    ax3.set_title('Boxplot for {}'.format(col))
    #axs[1,1].axis('off')
    plt.tight_layout()
    #plt.despine
    plt.show()
    return

plot_num(df['ctc'],'ctc')
```



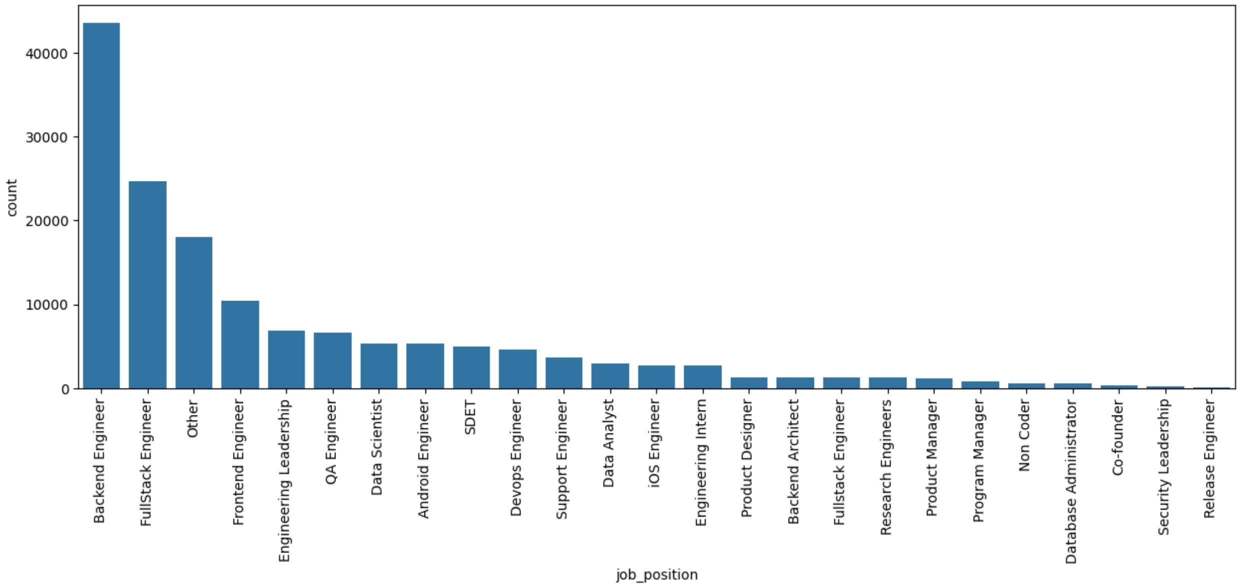

```
plt.figure(figsize = (15,5))
sns.barplot(df['orgyear'].value_counts())
plt.xticks(rotation = 90)
plt.show()
```



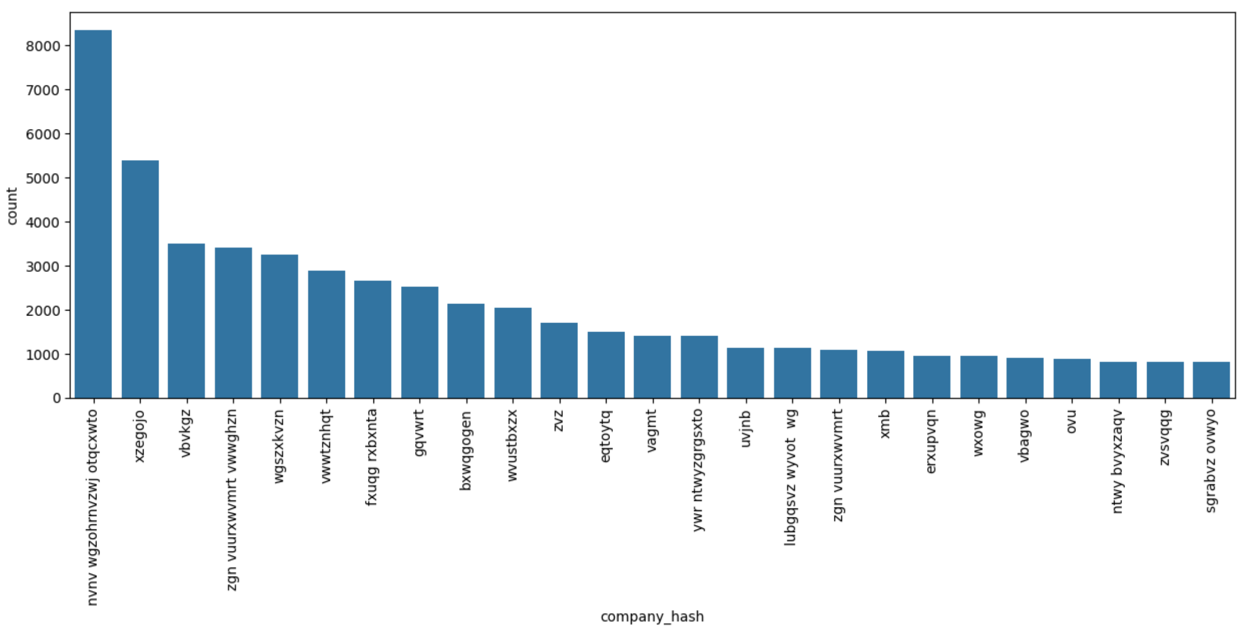

```
plt.figure(figsize = (15,5))
sns.barplot(df['ctc_updated_year'].value_counts())
plt.show()
```



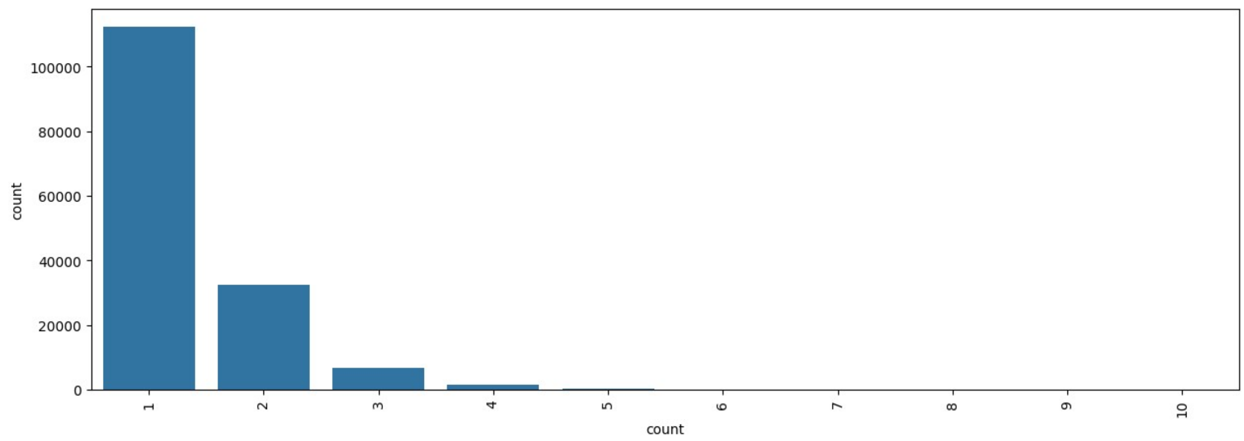

```
plt.figure(figsize = (15,5))
sns.barplot(df['job_position'].value_counts().head(25))
plt.xticks(rotation = 90)
plt.show()
```




```
plt.figure(figsize = (15,5))
sns.barplot(df['company_hash'].value_counts().head(25))
plt.xticks(rotation = 90)
plt.show()
```




```
plt.figure(figsize = (15,5))
sns.barplot(df['email_hash'].value_counts().value_counts())
plt.xticks(rotation = 90)
plt.show()
```



Observation set : 2

1. The first column seems to contain the index, hence can be dropped
2. The ctc column has many outliers and is not normally distributed. We will need to work on this.
3. comp_hash in terms of its distribution seems to be well positioned for aggregation.
4. email_hash again has more than 100000 rows appearing only once. which meant that the aggregation will not result in anything useful, hence we will drop it.
5. ctc seems to have been updated majorly from 2019
6. orgyear have high density in the range of 2015 to 2018
7. orgyear has a wide spread of datapoints which do not look clean. we will have to work on cleaning the orgyear fearure later.

Section 3 : Solving a few business queries

In section we will try to find answers to a few business questions as follows:

1. Top 10 employees (earning more than most of the employees in the company) - Tier 1

   1.1 Top 10 employees of data science in each company earning more than their peers - Class 1

   1.2 Bottom 10 employees of data science in each company earning less than their peers - Class 3

2. Bottom 10 employees (earning less than most of the employees in the company)- Tier 3
3. Top 10 employees in each company - X department - having 5/6/7 years of experience earning more than their peers - Tier X
4. Top 10 companies (based on their CTC)

5. Top 2 positions in every company (based on their CTC)

In the earlier section we saw that the data has many rows with missing values and there are a lot of inconsistencies within the data. However, to answer the business questions, we will drop the missing values and will go ahead with inconsistent data. In the later sections when we have the dataset cleaned and treated, we can again come back run the analysis in this section to see if we are getting different results.

```
# dropping the rows with NULL values
df1 = df.iloc[:,1:].dropna()

# Drop duplicate email_hash, keeping the first occurrence
df1 = df1.drop_duplicates(subset=['email_hash'])

# cleaning the orgyear and ctc_updated_year columns
df1.loc[(df1['orgyear']>df1['ctc_updated_year']),(['orgyear'])] =
df1['ctc_updated_year']

# creating the YoE column for our analysis
df1.loc[:,'YoE'] = df1['ctc_updated_year'] - df1['orgyear']

#checking the newly created feature - YoE
df1['YoE'].describe()
```

```
count    133145.000000
mean          5.273131
std          26.440631
min           0.000000
25%           2.000000
50%           4.000000
75%           7.000000
max        2021.000000
Name: YoE, dtype: float64
```

```
# checking if there are any further inconsistencies
df1.loc[df1['orgyear']>df1['ctc_updated_year']]
```

```
Empty DataFrame
Columns: [company_hash, email_hash, orgyear, ctc, job_position,
ctc_updated_year, YoE]
Index: []
```

```
df1.isnull().sum()
```

```
company_hash        0
email_hash          0
orgyear             0
ctc                 0
job_position        0
ctc_updated_year    0
YoE                 0
dtype: int64
```

```
df1.shape
```

```
(133145, 7)
```

Aggregating and creating groups on multiple levels

To answer the business questions defined earlier, we will need to group the data as described below:

1. Group at comany_hash level for ctc : this will help us find out mean, max, min etc ctc at each or any perticular company. We will suffix the resultant dataframe name with a 'C' (for company) to identify it easily.
2. Group at comany_hash level and job_position for ctc : this will help us find out mean, max, min etc ctc at each or any perticular company also drilled down to job_position in that company. We will suffix the resultant dataframe name with a 'CJ' (for company and job) to identify it easily.
3. Group at comany_hash level, job_position and YoE for ctc : this will help us find out mean, max, min etc ctc at each or any perticular company also filtering down to job_position and then within that the YoE. We will suffix the resultant dataframe name with a 'CJE' (for company, job and experience) to identify it easily.

Following is the process we will follow:

- Step 1 --> Group the data as described above
- Step 2 --> merge the information obtained in the grouped data with the original dataset
- Step 3 --> create new features in the dataset to categorise the rows in 3 classes (Top, Mid, Low). Top being those whose ctc is more than the mean of their group, Mid will have ctc equal to the mean ctc and Low will have ctc lower than the mean ctc.

We will ultimately be adding the below columns to the data set

- 3 new columns with mean ctc at each group level
- 3 new columns to describe category of each employee based on ctc compared to mean of that group

Step 1 : grouping the data

```
# grouping at company_hash level
df_grp_C = df1.groupby(['company_hash']).agg(ctc_mean_C =
('ctc','mean'),
                                             ctc_max_C = ('ctc','max'),
                                             ctc_min_C =
('ctc','min')).reset_index()

# grouping at company and job level
df_grp_CJ =
df1.groupby(['company_hash','job_position']).agg(ctc_mean_CJ =
('ctc','mean'),

ctc_max_CJ = ('ctc','max'),
```

```
ctc_min_CJ = ('ctc','min')).reset_index()

# grouping at company, job and YoE level
df_grp_CJE =
df1.groupby(['company_hash','job_position','YoE']).agg(ctc_mean_CJE =
('ctc','mean'),

ctc_max_CJE = ('ctc','max'),

ctc_min_CJE = ('ctc','min')).reset_index()
```

Step 2 : merging the groups with larger datasets

```
df_merge = df1.merge(df_grp_C[['company_hash', 'ctc_mean_C',
'ctc_max_C','ctc_min_C']],
                     on=['company_hash'],
                     how='left')

df_merge = df_merge.merge(df_grp_CJ[['company_hash', 'job_position',
'ctc_mean_CJ','ctc_max_CJ','ctc_min_CJ']],
                          on=['company_hash', 'job_position'],
                          how='left')

df_merge = df_merge.merge(df_grp_CJE[['company_hash', 'job_position',
'YoE', 'ctc_mean_CJE','ctc_max_CJE','ctc_min_CJE']],
                          on=['company_hash', 'job_position', 'YoE'],
                          how='left')

df_merge.head()

                company_hash  \
0             atrgxnnt xzaxv
1  qtrxvzwt xzegwgbb rxbxnta
2              ojzwnvwnxw vx
3                  ngpgutaxv
4                 qxen sqghu

                                          email_hash  orgyear      ctc
\
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...   2016.0  1100000

1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...   2018.0   449999

2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...   2015.0  2000000

3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58...   2017.0   700000

4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...   2017.0  1400000
```

```
          job_position  ctc_updated_year  YoE    ctc_mean_C  ctc_max_C  \
0                Other            2020.0  4.0  1.115667e+06    1771000

1  FullStack Engineer            2019.0  1.0  2.632682e+06  200000000

2    Backend Engineer            2020.0  5.0  2.000000e+06    2000000

3    Backend Engineer            2019.0  2.0  1.570326e+06    4700000

4  FullStack Engineer            2019.0  2.0  8.850000e+05    1400000


   ctc_min_C   ctc_mean_CJ  ctc_max_CJ  ctc_min_CJ  ctc_mean_CJE  ctc_max_CJE  \
0     500000  1.085000e+06     1100000     1070000  1.085000e+06      1100000
1      10000  9.212499e+05     2000000      300000  4.599995e+05       470000
2    2000000  2.000000e+06     2000000     2000000  2.000000e+06      2000000
3     200000  1.456667e+06     2600000      520000  1.358889e+06      1950000
4     540000  8.466667e+05     1400000      540000  1.000000e+06      1400000

   ctc_min_CJE
0      1070000
1       449999
2      2000000
3       700000
4       600000

# defingin a function to return the category

def get_label(data, col):
    if data['ctc'] > data[col]:
        return 'High'
    if data['ctc'] < data[col]:
        return 'Low'
    else:
        return 'Mid'

df_merge['class_C'] = df_merge.apply(get_label, args=('ctc_mean_C',), axis = 1)
df_merge['class_CJ'] = df_merge.apply(get_label, args=('ctc_mean_CJ',), axis = 1)
df_merge['class_CJE'] = df_merge.apply(get_label, args=('ctc_mean_CJE',), axis = 1)
```

Finding the answers to the questions defined earlier

1. Getting top 10 employees (earning more than most of the employees in the company)

```
df_merge[df_merge['class_C'] == 'High'].sort_values('ctc', ascending =
False).head(10)
```

```
                     company_hash  \
14795             ntrtutqegqbvzwt
71280  ytfrtnn uvwpvqa tzntquqxot
60691                          zv
9249               ntvb wgbuhntqo
54765                   yaew mvzp
29855             xzatstzt tzwxbv
54758                       gnytq
17523  uqxzwxuvr srgmvr xzctongqo
9192    nvnv wgzohrnvzwj otqcxwto
15327               xqgz bghznvxz

                                              email_hash  orgyear
ctc  \
14795  2d2aeac90e34dd5eede9b999578428f2e8c90608ee2160...   2015.0
200000000
71280  c888824e687a535d1bd2486ae28d67e414b21b09cbee61...   2015.0
200000000
60691  8f4e202df3abcd8aa3236a20193de65c2e0ae7c8b3664c...   2020.0
200000000
9249   6739ade7083af0779d5eb4cf40bcb9dce42d314fb234c2...   2015.0
200000000
54765  1ad3d8c2b855e9cbe6bb187e8140a07ed8a8b29d275ae0...   2017.0
200000000
29855  1fe9990319dc839ee1d74ceb8ff599ffa4fda6e9fd5666...   2016.0
200000000
54758  5a22951df9377bc13dab9bd4771ee7d0bbd7e639d2f81b...   2015.0
200000000
17523  9e2c0f18746d1491023938082ad560d01de8a0f58cde93...   2015.0
200000000
9192   0d235f7e73cd9484909b32a35c69df12296a051f68ef83...   2017.0
200000000
15327  f4e874b3329098fdb3de47a83e1b41b2f5f4b873e148dd...   2012.0
200000000

             job_position  ctc_updated_year  YoE    ctc_mean_C
ctc_max_C  \
14795    Support Engineer            2020.0  5.0  1.000206e+07
200000000
71280        Data Analyst            2020.0  5.0  3.046015e+06
200000000
```

```
60691  Engineering Intern            2020.0  0.0  1.048143e+07
200000000
9249   FullStack Engineer            2020.0  5.0  1.887364e+07
200000000
54765               Other            2020.0  3.0  1.437041e+07
200000000
29855   Frontend Engineer            2020.0  4.0  8.384074e+06
200000000
54758   Frontend Engineer            2020.0  5.0  9.771761e+06
200000000
17523               Other            2020.0  5.0  1.015773e+07
200000000
9192                Other            2020.0  3.0  1.976978e+06
200000000
15327               Other            2020.0  8.0  3.395833e+07
200000000

       ctc_min_C   ctc_mean_CJ  ctc_max_CJ  ctc_min_CJ
ctc_mean_CJE  \
14795     100000  2.043100e+07   200000000      280000  1.002400e+08

71280      10000  5.058000e+07   200000000      760000  6.717333e+07

60691       1000  6.703000e+07   200000000      300000  1.001500e+08

9249      400000  6.740667e+07   200000000      420000  2.000000e+08

54765     100000  1.344970e+07   200000000      100000  5.055750e+07

29855     180000  3.385000e+07   200000000      180000  1.004000e+08

54758       2300  5.032700e+07   200000000      200000  1.002500e+08

17523     600000  2.000000e+08   200000000   200000000  2.000000e+08

9192         600  2.496613e+06   200000000        3500  3.266908e+06

15327     380000  2.000000e+08   200000000   200000000  2.000000e+08


       ctc_max_CJE  ctc_min_CJE class_C class_CJ class_CJE
14795    200000000       480000    High     High      High
71280    200000000       760000    High     High      High
60691    200000000       300000    High     High      High
9249     200000000    200000000    High     High       Mid
54765    200000000       720000    High     High      High
29855    200000000       800000    High     High      High
54758    200000000       500000    High     High      High
17523    200000000    200000000    High      Mid       Mid
```

```
9192     200000000       100000     High      High       High
15327    200000000    200000000     High       Mid        Mid
```

1.1. Getting top 10 employees of data science in each company earning more than their peers

We need to get to 10 employees in terms of ctc company wise with job_position similar or close to 'Data Science'. So in addition to what we did in the first problem :

- we will include the job_position column while filtering
- we will use ctc_mean_CJ and class_CJ to solve this problem
- we will also need to identify a few companies to check our results

But, how do we get the rows where the job is 'data science' or partially matching it? Let's try to find out.

```
# checking unique job_position in the dataset
df_merge['job_position'].nunique()

820
```

There are quite a lot of unique job positions, we can not manually find what we need. Lets try to filter with the term 'Data Science'

```
# Filter job descriptions that contain 'data science'
job_filtered_df = df_merge[df_merge['job_position'].str.contains('Data
Science', case=False, na=False)]
job_filtered_df['job_position'].unique()

array(['Data Science Analyst'], dtype=object)

# Filter job descriptions that contain 'data science'
job_filtered_df =
df_merge[df_merge['job_position'].str.contains('Data', case=False,
na=False)]
job_filtered_df['job_position'].unique()

array(['Data Analyst', 'Data Scientist', 'Database Administrator',
       'Senior Data Scientist', 'Intern-data analyst',
       'Senior Data Engineer', 'Data Science Analyst', 'Data
Engineer',
       'Software developer (Data engineer)', 'Data Eingineer',
       'DATA ASSOCIATE', 'Associate Data Scientist',
       'Machine Learning Data Associate ',
       "Some data entry operator like some copy's write.type and
upload",
       'Data Engineer 2', 'Data Scientist II', 'Big data Developer',
       'Data Visualization Engineer', 'Database developer',
       'Associate Data Engineer', 'Cloud Data architect',
```

```
       'Data Warehouse Developer', 'Data entry', 'Data Scientist 2',
       'Data Operations Manager', 'Senior Database Engineer',
       'Data/Product Engineer'], dtype=object)
```

```
# creating a list of all required job positions
job_list = ['Data Scientist', 'Senior Data Scientist', 'Data Science
Analyst',
            'Associate Data Scientist', 'Machine Learning Data
Associate ',
            'Data Scientist II','Data Scientist 2']

df_merge.loc[((df_merge['class_CJ'] == 'High') & (df_merge['class_C']
== 'High')) &
(df_merge['job_position'].isin(job_list))].sort_values('ctc',
ascending = False).head(10)
```

```
                                company_hash  \
676     mqxonrtwgzt v bvyxzaqv sqghu wgbuvzj
21307             ihvaqvnxw xzoxsyno ucn rna
79310                       xzzgcv ogrhnxgzo
99343                          ntwy bvyxzaqv
22198                                zvsvqqg
15097                         bgqsvz onvzrtj
113831                                  wxnx
51222                              bxwqgogen
7189                                  sggsrt
47860                                    zvz

                                               email_hash  orgyear
ctc  \
676     cda8d723438e81185d2ee8c348870a4612eea974cdb2db...   2017.0
200000000
21307   bd222ea783ee372da4e0ad60fdccec0b8f37999a032025...   2015.0
200000000
79310   6b6dd66bae787dd4dd417e1777f8ea5a057257e9019995...   2016.0
100000000
99343   6ad86d120e39db485331f9a0b2b1f15ce2a7bdaee778ab...   2019.0
100000000
22198   15adaeb2eef9c0ee8a0f18e189bf426be390f5d1e911fd...   2021.0
60000000
15097   2bede29959707d8c6f283d98319361c386baa6fa5c8028...   2020.0
50000000
113831  f7b7c771ccdbbca7248002ba83f7a176baa974c2c7bb8f...   2011.0
24200000
51222   599e489c815ba51967965c5d6adefd7a76a99ffaa129bd...   2002.0
22500000
7189    3e290b892b73283b96293c53e4ce4dce2cc6a22399b95c...   2020.0
22000000
47860   80f1ae60373f0ada3b75ce19eb585f8cf112de3cfa6ea7...   2017.0
20000000
```

```
          job_position  ctc_updated_year   YoE    ctc_mean_C  
ctc_max_C  \
676     Data Scientist            2020.0   3.0  7.106909e+06   
200000000   
21307   Data Scientist            2019.0   4.0  3.408333e+07   
200000000   
79310   Data Scientist            2020.0   4.0  1.197778e+07   
100000000   
99343   Data Scientist            2019.0   0.0  3.893317e+06   
200000000   
22198   Data Scientist            2021.0   0.0  1.280034e+06    
60000000   
15097   Data Scientist            2020.0   0.0  2.626905e+06   
100000000   
113831  Data Scientist            2020.0   9.0  2.784877e+06   
100000000   
51222   Data Scientist            2019.0  17.0  3.368807e+06   
200000000   
7189    Data Scientist            2020.0   0.0  6.803418e+06   
200000000   
47860   Data Scientist            2020.0   3.0  2.363139e+06   
200000000   

        ctc_min_C   ctc_mean_CJ  ctc_max_CJ  ctc_min_CJ   ctc_mean_CJE  
\
676        350000  4.113400e+07   200000000      770000  2.000000e+08   

21307      420000  6.743667e+07   200000000      750000  2.000000e+08   

79310      450000  5.040000e+07   100000000      800000  5.040000e+07   

99343       10000  6.768882e+06   100000000      350000  5.080000e+07   

22198        6000  4.428056e+06    60000000      300000  1.557500e+07   

15097        1000  1.851333e+07    50000000     2300000  2.662000e+07   

113831       1500  3.239412e+06    24200000      850000  1.320500e+07   

51222        5000  3.205111e+06    22500000       94000  2.250000e+07   

7189         1000  5.730000e+06    22000000       10000  8.966667e+06   

47860        1000  1.706903e+06    20000000       46500  5.795000e+06   


        ctc_max_CJE  ctc_min_CJE class_C class_CJ class_CJE  
676       200000000    200000000    High     High       Mid  
21307     200000000    200000000    High     High       Mid  
```

```
79310     100000000       800000    High     High      High
99343     100000000      1600000    High     High      High
22198      60000000       300000    High     High      High
15097      50000000      3240000    High     High      High
113831     24200000      2210000    High     High      High
51222      22500000     22500000    High     High       Mid
7189       22000000      1100000    High     High      High
47860      20000000       750000    High     High      High
```

1.2 Bottom 10 employees of data science in each company earning less than their peers

```
df_merge.loc[((df_merge['class_CJ'] == 'Low') & (df_merge['class_C']
== 'High')) &
(df_merge['job_position'].isin(job_list))].sort_values('ctc',
ascending = False).tail(10)
```

```
                                            company_hash  \
93400   qgmtqn mgowy tzsxzttqxzs vza mhoxztoo ogrhnxgzo
52678                                       stztqvr bxrro
2477                                       ihvznxuyx xzw
77883                                   nvnv ntwyzgrgsxto
15415                                             szvzxvx
14098                                 nwo xzzgcvnxgz rvmo
13320                                            zthatoxw
111696                                        pcvznhb xzw
59887                                       vrsgvzvrjnxwo
8475                                  nyt ouvqpo eghzavnxgz

                                               email_hash  orgyear
ctc  \
93400   e07c7b80830b137ea848f24c5b8201c355a1cbf817420b...   2015.0
1000000
52678   98c934d3c9cec8b081f0f98cfa8f6173459f79551b7451...   2016.0
960000
2477    71f8c5b08273ef0d44427dffd1b1d520cf561a0f4bd1b3...   2019.0
940000
77883   ba46cba4c34e23e24ac61febf59d467d6d0e975df064e9...   2017.0
850000
15415   d95f45f714f0d32d04753c69b4685537317b17649270a2...   2018.0
850000
14098   f4abe54f6d28593645d29c32c795e3ba55ee3e7fc71c7d...   2016.0
800000
13320   15a224659521108b95493bafdc20655e78f9f5db733817...   2016.0
800000
111696  47b14d527d3b8042b760ddcf0abfdc2fadc5279573df63...   2016.0
800000
59887   3a9c5ed6900922871a5f91627c0a7f84cba58d31a53038...   2017.0
630000
```

```
8475    a26144267a29d5dae251ef1519f26d178b088f85889686...   2018.0
600000

          job_position  ctc_updated_year  YoE     ctc_mean_C
ctc_max_C  \
93400   Data Scientist            2019.0  4.0  915277.736111
3600000
52678   Data Scientist            2019.0  3.0  959999.800000
1689999
2477    Data Scientist            2020.0  1.0  932941.176471
1480000
77883   Data Scientist            2019.0  2.0  848484.848485
2500000
15415   Data Scientist            2020.0  2.0  800000.000000
1300000
14098   Data Scientist            2020.0  4.0  797000.000000
1200000
13320   Data Scientist            2020.0  4.0  732727.272727
2050000
111696  Data Scientist            2017.0  1.0  699999.750000
1040000
59887   Data Scientist            2019.0  2.0  613333.000000
700000
8475    Data Scientist            2021.0  3.0  324285.714286
650000

        ctc_min_C   ctc_mean_CJ  ctc_max_CJ  ctc_min_CJ  ctc_mean_CJE
\
93400      140000  1.002000e+06     1200000      800000     1000000.0

52678      650000  9.999998e+05     1689999      650000      960000.0

2477       500000  1.035000e+06     1400000      800000      870000.0

77883      300000  1.016667e+06     1700000      500000      850000.0

15415      500000  1.075000e+06     1300000      850000      850000.0

14098      400000  8.333333e+05     1200000      500000      800000.0

13320      360000  1.425000e+06     2050000      800000      800000.0

111696     450000  9.200000e+05     1040000      800000      800000.0

59887      509999  6.650000e+05      700000      630000      665000.0

8475       100000  6.250000e+05      650000      600000      600000.0


        ctc_max_CJE  ctc_min_CJE class_C class_CJ class_CJE
```

```
93400       1000000     1000000    High     Low      Mid
52678        960000      960000    High     Low      Mid
2477         940000      800000    High     Low     High
77883        850000      850000    High     Low      Mid
15415        850000      850000    High     Low      Mid
14098        800000      800000    High     Low      Mid
13320        800000      800000    High     Low      Mid
111696       800000      800000    High     Low      Mid
59887        700000      630000    High     Low      Low
8475         600000      600000    High     Low      Mid
```

2. Bottom 10 employees (earning less than most of the employees in the company)

```
df_merge[df_merge['class_C'] == 'Low'].sort_values('ctc', ascending =
False).tail(10)
```

```
                     company_hash  \
55512          bgngqgrv ogrhnxgzo
71149                     onvqnhu
2913                       sggsrt
117667             mtznrtj ojontbo
105077                kvrgqv sqghu
87165                  cxo wvqttqo
104666                    zhbmqo
66223                         gjg
112454  nvnv wgzohrnvzwj otqcxwto
88813                 xzntqcxtfmxn

                                              email_hash  orgyear
ctc  \
55512   a7894c6d848de3021cfd16b35178cf8f48b10d77aa46dc...   2016.0
1000
71149   d9476096e4e5d6f0b0f6079b0543145f62b43c82478bbc...   2018.0
1000
2913    5756870d895deca920251df2377dad261084904a4f9d10...   1973.0
1000
117667  7c8e0d8194db4deb41cbc9b3b6c428e0f9ab289436638e...   2016.0
1000
105077  ae625c7063c1f8194deadfb28905d5dcc6f9077274a083...   2017.0
1000
87165   daa966561c4087398b3c3b13855ce17adcf5e08dda803f...   2012.0
1000
104666  d926b36fd7c88094c8837323e378671f8354d3fe0dc488...   2011.0
1000
66223   b995d7a2ae5c6f8497762ce04dc5c04ad6ec734d70802a...   2018.0
600
112454  80ba0259f9f59034c4927cf3bd38dc9ce2eb60ff18135b...   2012.0
600
```

```
88813   3505b02549ebe2c95840ac6f0a35561a3b4cbe4b79cdb1...   2014.0
2

              job_position  ctc_updated_year   YoE    ctc_mean_C
ctc_max_C  \
55512     Android Engineer            2020.0   4.0  1.078474e+06
3000000
71149         iOS Engineer            2020.0   2.0  5.479771e+06
199990000
2913            Co-founder            2020.0  47.0  6.803418e+06
200000000
117667  FullStack Engineer            2019.0   3.0  6.919999e+05
2033000
105077    Backend Engineer            2021.0   4.0  1.866667e+04
40000
87165     Android Engineer            2017.0   5.0  1.729683e+07
100000000
104666    Backend Engineer            2019.0   8.0  5.550000e+04
110000
66223   FullStack Engineer            2021.0   3.0  2.000809e+06
100000000
112454    Backend Engineer            2017.0   5.0  1.976978e+06
200000000
88813     Backend Engineer            2019.0   5.0  1.386119e+06
6400000

        ctc_min_C   ctc_mean_CJ  ctc_max_CJ  ctc_min_CJ   ctc_mean_CJE
\
55512        1000  3.670000e+05      600000        1000    1000.000000

71149        1000  9.999550e+07   199990000        1000    1000.000000

2913         1000  1.000000e+03        1000        1000    1000.000000

117667       1000  7.268000e+05     2033000        1000  275500.000000

105077       1000  1.866667e+04       40000        1000    1000.000000

87165        1000  1.000000e+03        1000        1000    1000.000000

104666       1000  1.000000e+03        1000        1000    1000.000000

66223         600  1.386017e+06     2800000         600  883533.333333

112454        600  1.310597e+06   200000000         600  635373.064103

88813           2  7.373685e+05     2400000           2   70001.000000


        ctc_max_CJE  ctc_min_CJE class_C class_CJ class_CJE
```

```
55512          1000         1000     Low      Low       Mid
71149          1000         1000     Low      Low       Mid
2913           1000         1000     Low      Mid       Mid
117667       550000         1000     Low      Low       Low
105077         1000         1000     Low      Low       Mid
87165          1000         1000     Low      Mid       Mid
104666         1000         1000     Low      Mid       Mid
66223       1400000          600     Low      Low       Low
112454      2500000          600     Low      Low       Low
88813        140000            2     Low      Low       Low
```

3. Top 10 employees in each company - X department - having 5/6/7 years of experience earning more than their peers

4. Top 10 companies (based on their CTC)

```
df_grp_C.sort_values('ctc_mean_C', ascending = False).head(10)
```

```
                         company_hash   ctc_mean_C   ctc_max_C
ctc_min_C
25958      vuytrxgz ogenfvqto ucn   rna  200000000.0   200000000
200000000
11912    ntwywg egqbtqrj ntwy wgwpnvxr  200000000.0   200000000
200000000
29952                 xwhmt ogrhnxgzo  200000000.0   200000000
200000000
27623               wo ogen ogrhnxgzo  200000000.0   200000000
200000000
17049             pyxcqvl vhngbgmxrto  200000000.0   200000000
200000000
19355             sgraygbk wgzohrnxzs  200000000.0   200000000
200000000
2146       bjnqvy tztqsj xzaxv ucn rna  200000000.0   200000000
200000000
520              ama uqgltwno rxbxnta  200000000.0   200000000
200000000
10779                            neny  200000000.0   200000000
200000000
10766   nco rgsxonxwo otqcxwto rxbxnta  200000000.0   200000000
200000000
```

5. Top 2 positions in every company (based on their CTC)

```
def get_top_n(grp, n=2):
    return grp.head(n)

df_grp_CJE.sort_values('ctc_max_CJE').groupby('company_hash').apply(ge
t_top_n,

n=2).reset_index(drop = True)[['company_hash',

'job_position']].head(15)
```

```
                     company_hash          job_position
0                               0                 Other
1                            0000                 Other
2                     01 ojztqsj      Android Engineer
3                     01 ojztqsj     Frontend Engineer
4    05mz exzytvrny uqxcvnt rxbxnta   Backend Engineer
5                               1                 Other
6                      1 axsxnvro     Backend Engineer
7                          1 jtvq     Backend Engineer
8                              10     Backend Engineer
9        10 axsxnvr  ahmvx  rgzagz    Android Engineer
10                   1000uqgltwn     Frontend Engineer
11                     1001 vuuo     Frontend Engineer
12                      100uxzo    Engineering Intern
13             103 onhaxgo ucn rna   Frontend Engineer
14                 10dvx rtvqzxzs       Data Scientist
```

Observation set 3

All the questions defined earlier have been answered. We can choose to have the required information from the output of each question. Similarly we may also be able to define new set of question based on our aggregation and get desired information.

Section 4 : Pre-processing the data

In this section we will prepare our data for the kmeans clustering step.

Approach :

- we will drop features which are not required
- we will treat each column for inconsistencies and outliers
- we will scale the features, while ignoring the NULL values
- we will use knnimputer to fill those null values

- finally we compress the data and see what kind of cluster formation do we see beofre kmeans

```
# reminding ourselves how many null values were there in the original dataset
df.isnull().sum()
```

```
Unnamed: 0               0
company_hash            44
email_hash               0
orgyear                 86
ctc                      0
job_position         52564
ctc_updated_year         0
dtype: int64
```

```
# dropping the firt column
df2 = df.iloc[:,1:]
```

Treating orgyear and ctc_updated_year

```
# replacing inconsistent values in org_year with NULL valules
df2['orgyear'] = df2['orgyear'].apply(lambda x : np.nan if x < 1970 else x)
df2['orgyear'] = df2['orgyear'].apply(lambda x : np.nan if x > 2024 else x)

# replacing the inconsistent year values which we found in the earlier sections
df2.loc[df2['orgyear']>df2['ctc_updated_year'],'orgyear'] = np.nan

df2.isnull().sum()
```

```
company_hash            44
email_hash               0
orgyear               8977
ctc                      0
job_position         52564
ctc_updated_year         0
dtype: int64
```

treating comp_hash and job_posittion

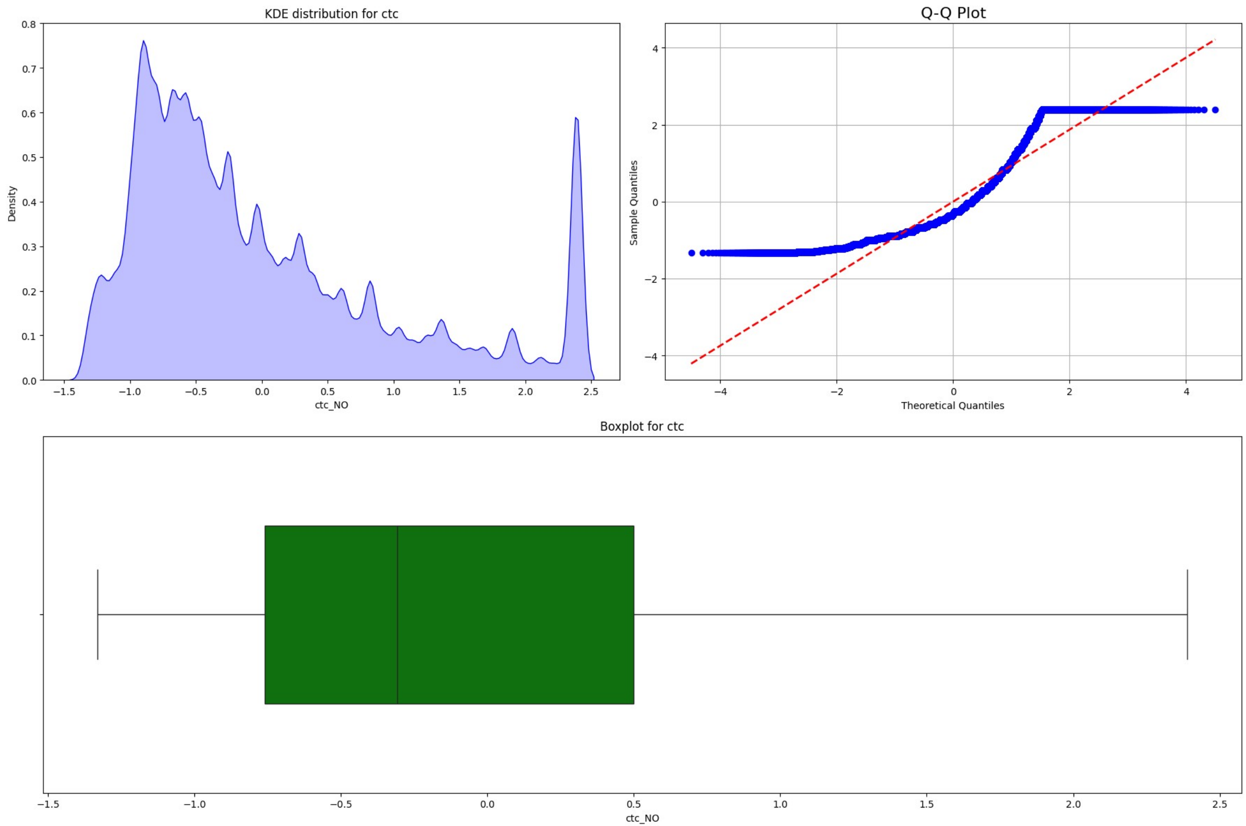
Both the columns have missing values and we plan to use KNNImputer in the next section for treating the missing values. Hence we need to first scale the data while ignoring the missing values. Lets get to it.

```
df2['comp_encoded'] =
df2['company_hash'].map(df2['company_hash'].value_counts())
```

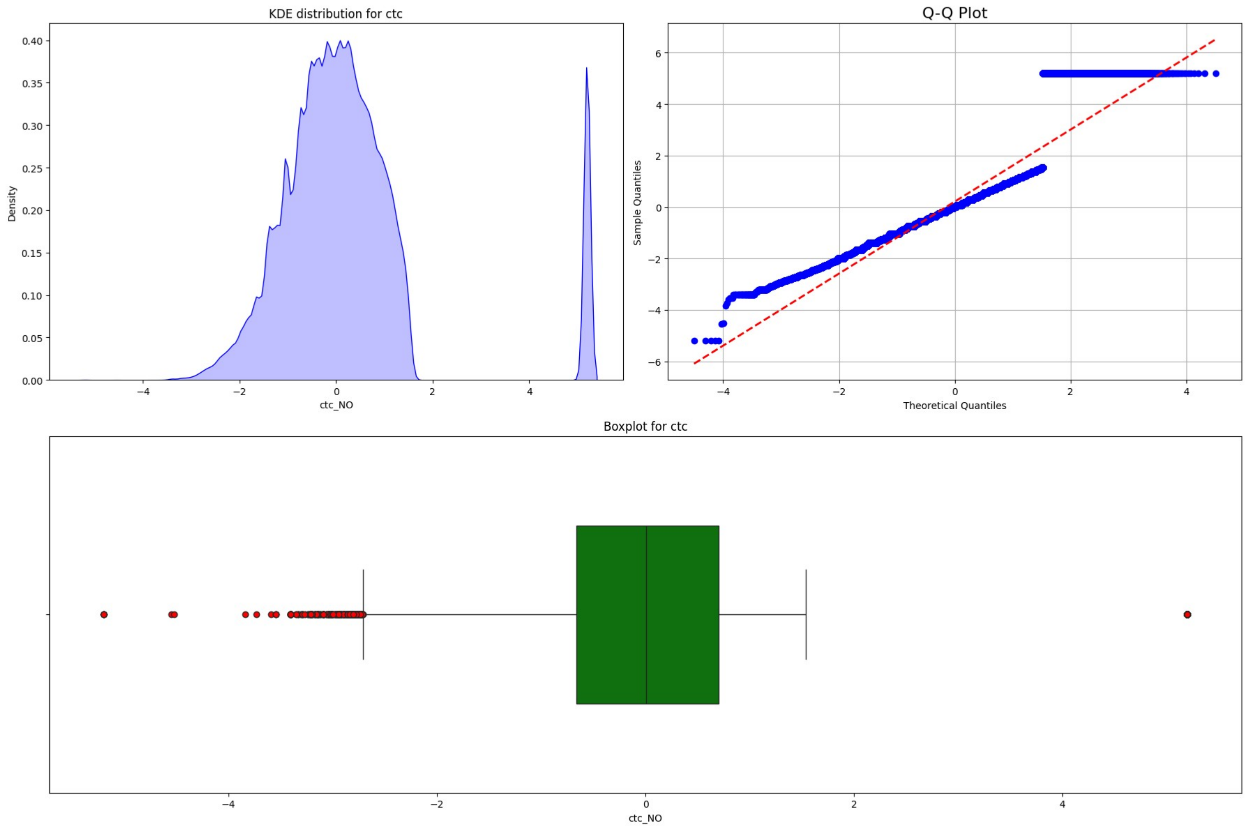
```
df2['job_encoded'] =
df2['job_position'].map(df2['job_position'].value_counts())
```

treating ctc for outliers

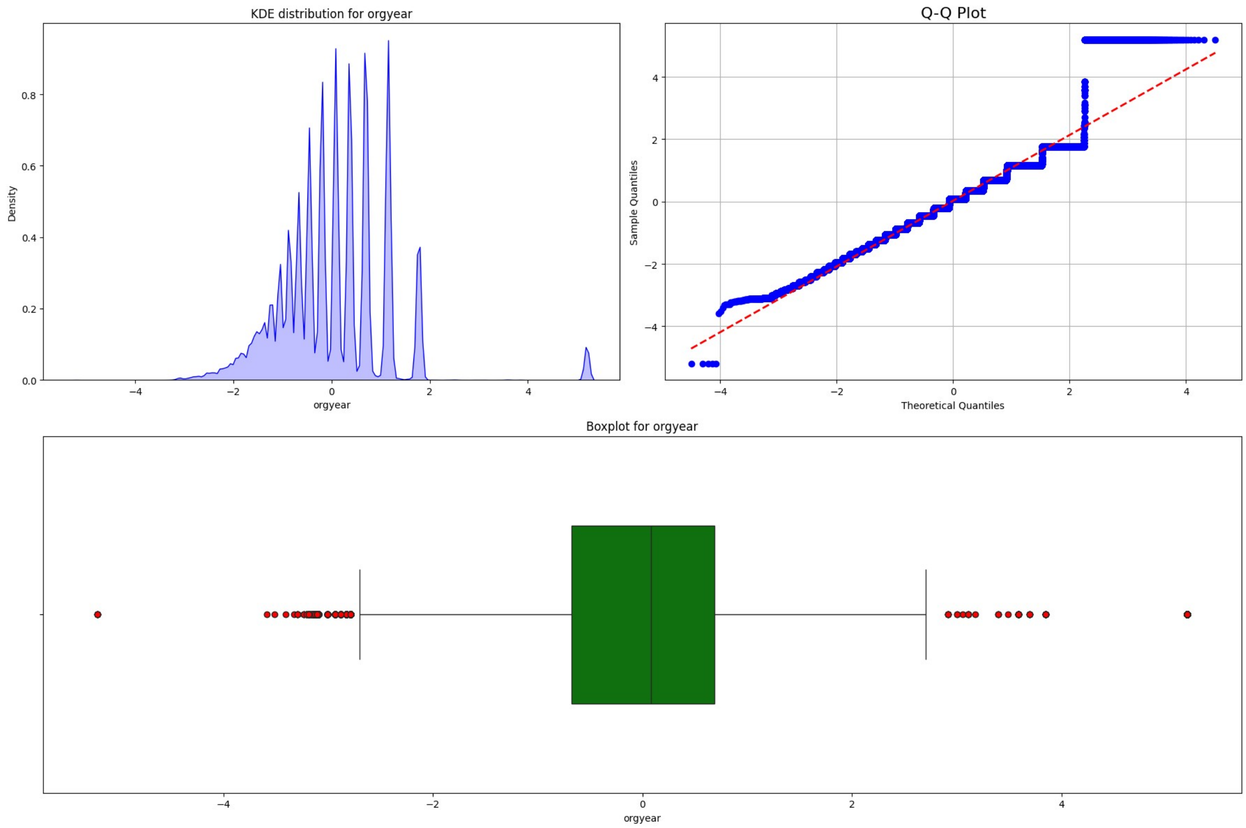
```
# Calculate the Interquartile Range (IQR) for 'ctc'
Q1 = df2['ctc'].quantile(0.25)
Q3 = df2['ctc'].quantile(0.75)
IQR = Q3 - Q1

# Define the upper and lower bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df2['ctc_NO'] = df2['ctc'].apply(lambda x : 0 if x < lower_bound else
(upper_bound if x > upper_bound else x))



df3 = df2.drop(columns =
['ctc','email_hash','job_position','company_hash'])



# Initialize the scaler
scaler = StandardScaler(with_mean=True, with_std = True)

# Function to scale data while ignoring NaN values
def scale_column(column):
    non_nan_mask = ~column.isna()
    column_scaled = column.copy()
    scaled_values =
scaler.fit_transform(column[non_nan_mask].values.reshape(-1,
1)).flatten()
    # Ensure the scaled values have the same dtype as the original
column
    column_scaled[non_nan_mask] = scaled_values.astype(column.dtype)
    return column_scaled

# Apply the scaling function to each column
df_scaled = df3.apply(scale_column, axis=0)



plot_num(df_scaled['ctc_NO'],'ctc')
```

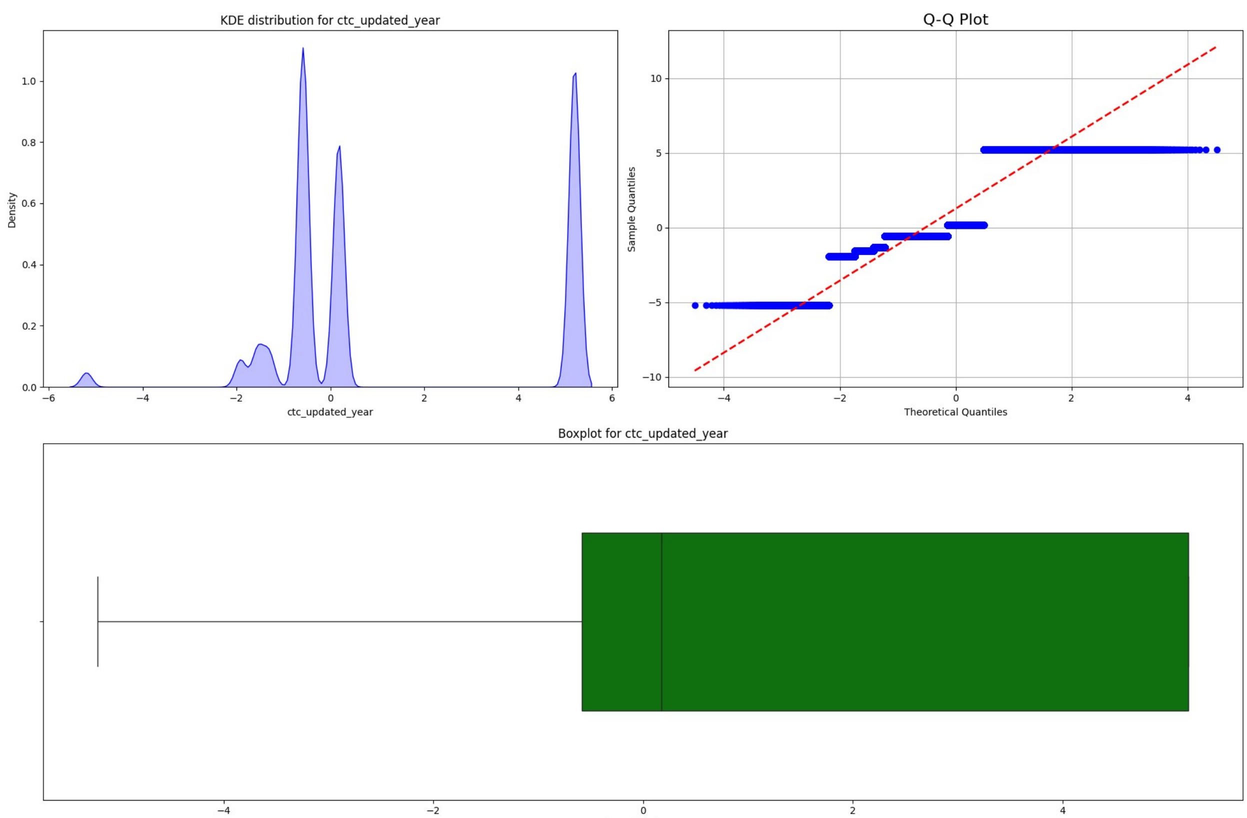


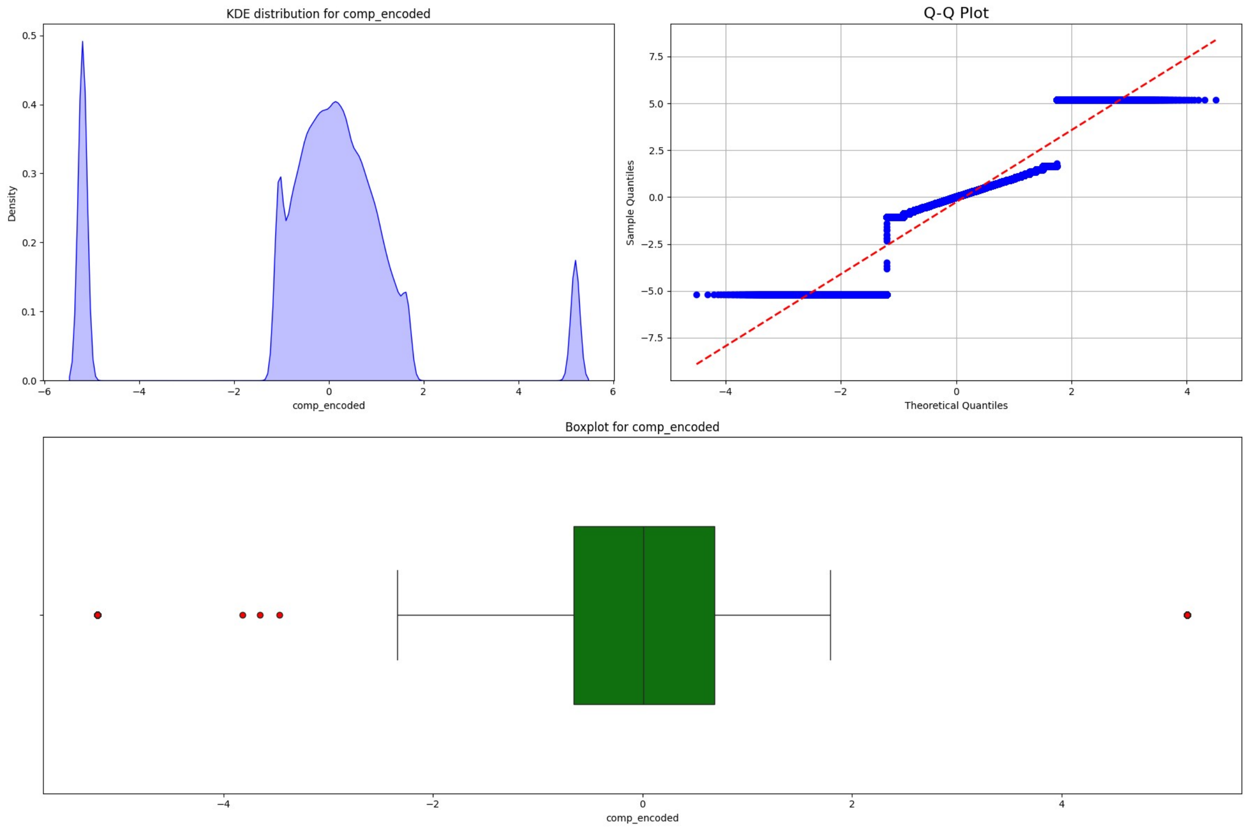

```
scaler = QuantileTransformer(output_distribution='normal')
# Apply the scaling function to each column
df_scaled_QT = df3.apply(scale_column, axis=0)

plot_num(df_scaled_QT['ctc_NO'],'ctc')
```

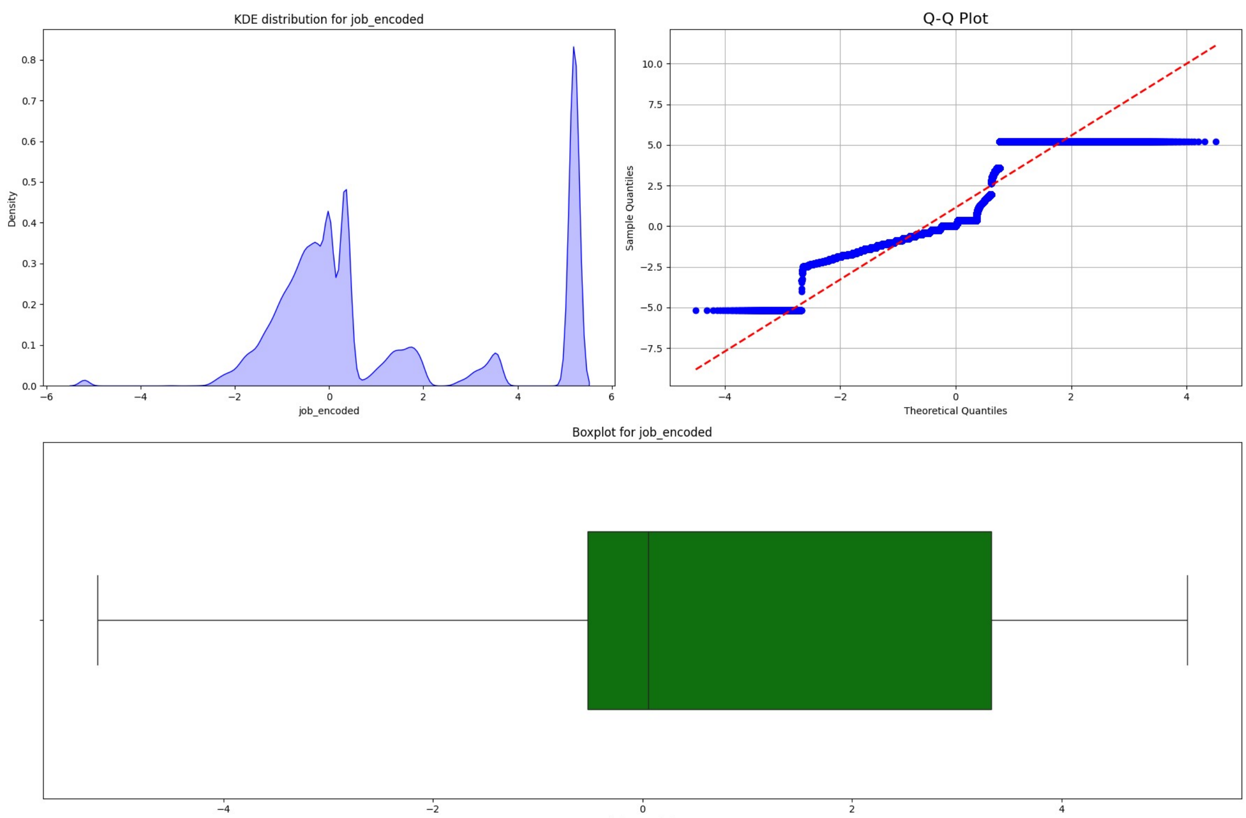


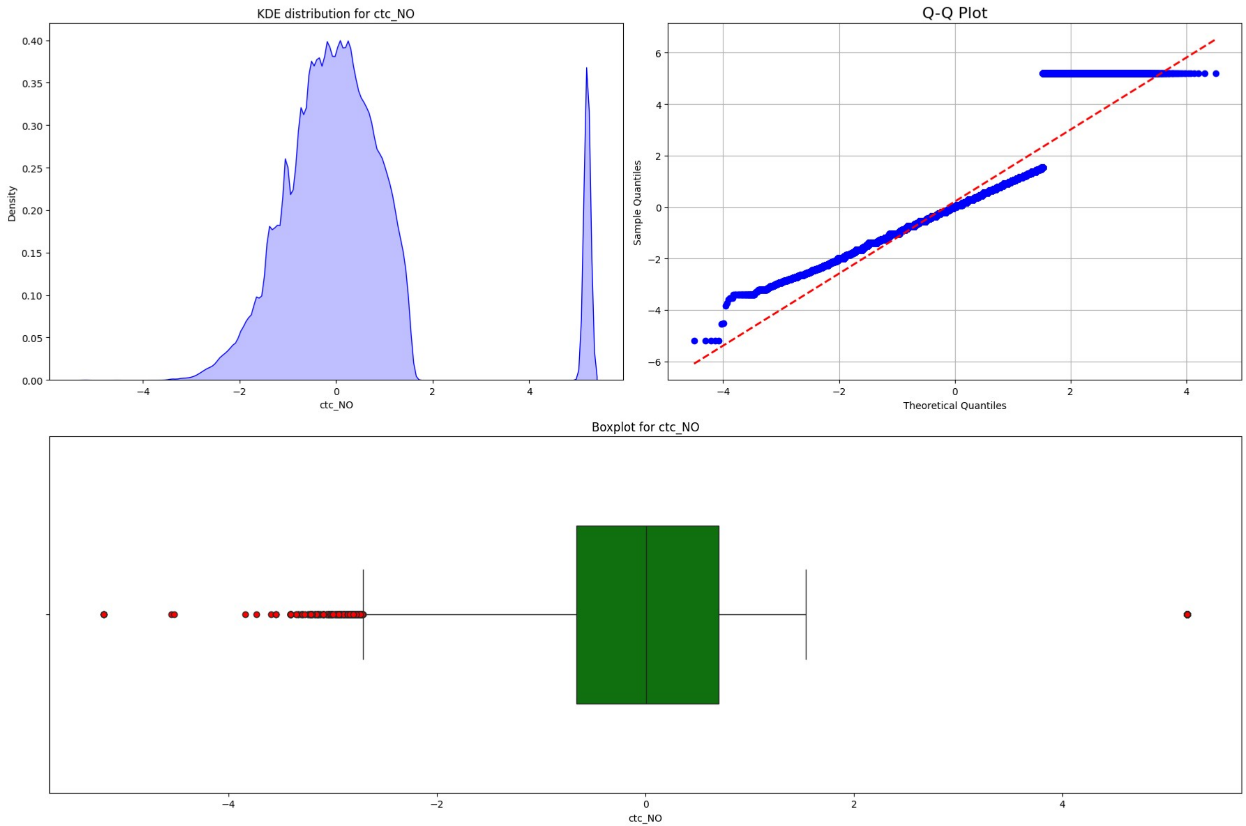

```python
# Create the KNN imputer
knn_imputer = KNNImputer(n_neighbors=3)
# Fit and transform the data
df_imputed_n3 = pd.DataFrame(knn_imputer.fit_transform(df_scaled_QT),
columns=df_scaled_QT.columns)


# Create the KNN imputer
knn_imputer = KNNImputer(n_neighbors=10)
# Fit and transform the data
df_imputed_n10 = pd.DataFrame(knn_imputer.fit_transform(df_scaled_QT),
columns=df_scaled_QT.columns)


for col in df_imputed_n3.columns:
    plot_num(df_imputed_n3[col],col)
```


KDE distribution for orgyear
Density
orgyear
Q-Q Plot
Sample Quantiles
Theoretical Quantiles
Boxplot for orgyear
orgyear

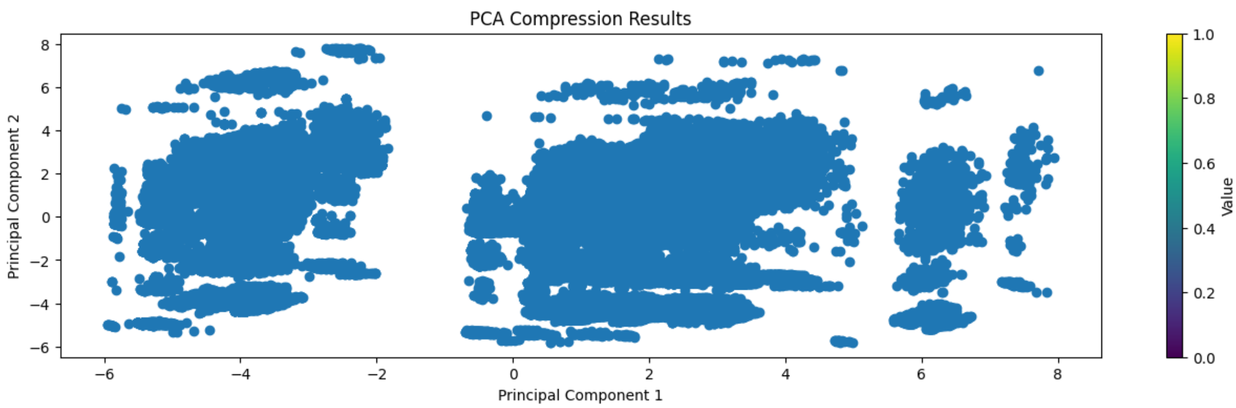

KDE distribution for ctc_updated_year
Density
ctc_updated_year
Q-Q Plot
Sample Quantiles
Theoretical Quantiles
Boxplot for ctc_updated_year
ctc_updated_year


KDE distribution for comp_encoded
Density
comp_encoded
Q-Q Plot
Sample Quantiles
Theoretical Quantiles
Boxplot for comp_encoded
comp_encoded

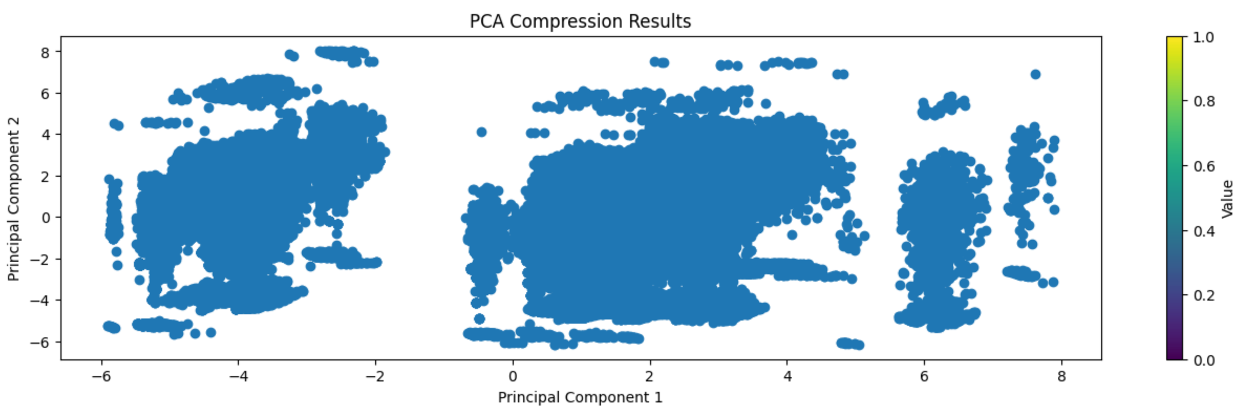

KDE distribution for job_encoded
Density
job_encoded
Q-Q Plot
Sample Quantiles
Theoretical Quantiles
Boxplot for job_encoded
job_encoded




```
def compress_and_plot_pca(data, n_components=2):

    # Applying PCA
    pca = PCA(n_components=n_components)
    pca_data = pca.fit_transform(data)

    # Plotting the results
    plt.figure(figsize=(16, 4))
    plt.scatter(pca_data[:, 0], pca_data[:, 1], cmap='viridis',
marker='o')
    plt.title('PCA Compression Results')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.colorbar(label='Value')
    plt.show()

compress_and_plot_pca(df_imputed_n3)
```




```
compress_and_plot_pca(df_imputed_n10)
```



Observation set 3 :

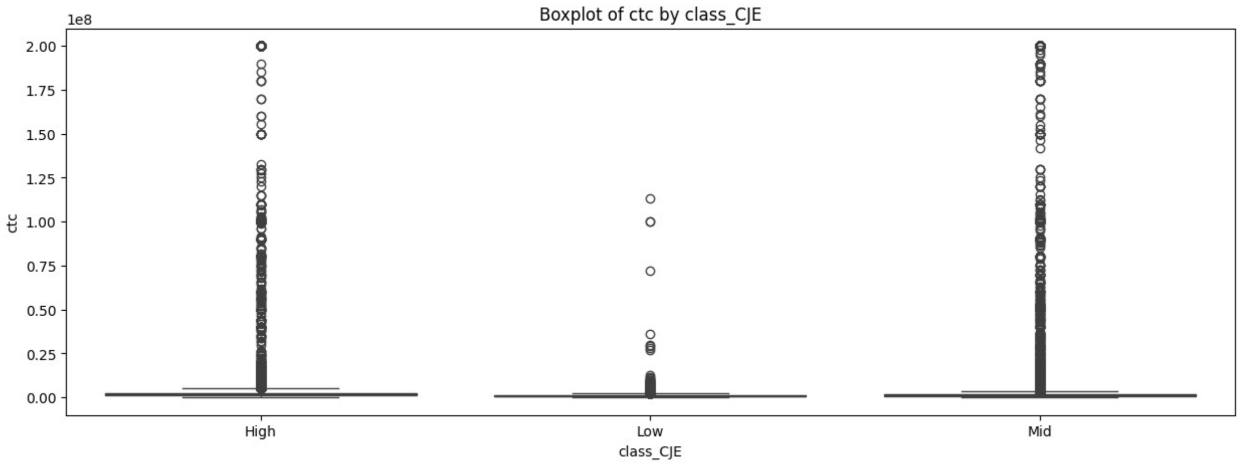
- the distribution of columns post outlier removal and normlization has improved fairly if not entirely
- the cluster formation between different knnimputer approach does not seem to vary and hence we will go with n3 in the later section
- we can see that there are clear clusters visible in the plots above, we will have to later compare and see if kmeans is giving us similar results
- we couls have used other methods for outlier removal and imputation but in the interst of time and complexity lets us proceed with what we have

Section 5 : Visualization Part 2

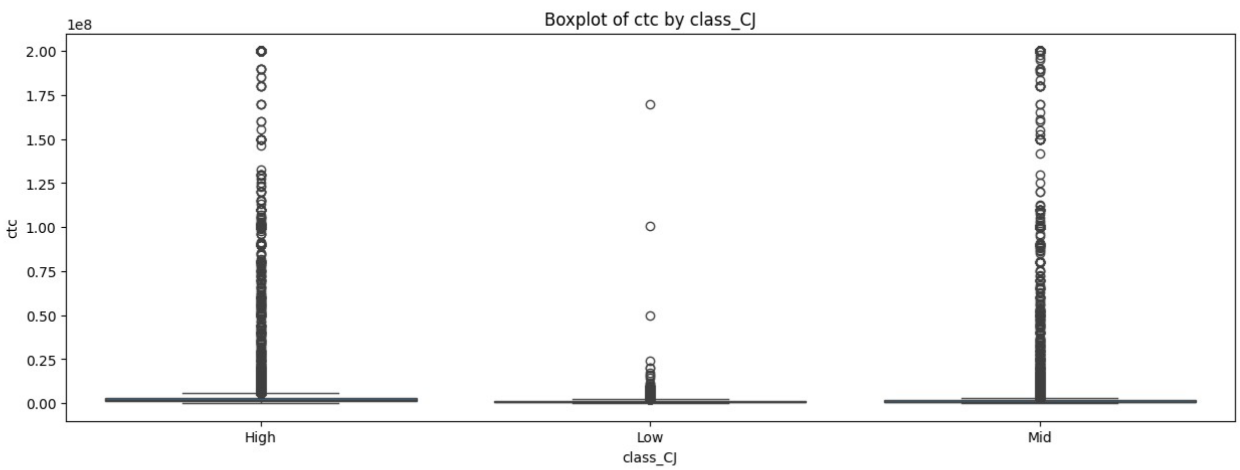
```
df_merge.head()

                  company_hash  \
0              atrgxnnt xzaxv
1  qtrxvzwt xzegwgbb rxbxnta
```

```
2               ojzwnvwnxw vx
3                   ngpgutaxv
4                  qxen sqghu

                                            email_hash  orgyear      ctc
\
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...   2016.0  1100000

1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...   2018.0   449999

2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...   2015.0  2000000

3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58...   2017.0   700000

4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...   2017.0  1400000


         job_position  ctc_updated_year  YoE    ctc_mean_C  ctc_max_C
\
0               Other            2020.0  4.0  1.115667e+06    1771000

1  FullStack Engineer            2019.0  1.0  2.632682e+06  200000000

2    Backend Engineer            2020.0  5.0  2.000000e+06    2000000

3    Backend Engineer            2019.0  2.0  1.570326e+06    4700000

4  FullStack Engineer            2019.0  2.0  8.850000e+05    1400000


   ctc_min_C   ctc_mean_CJ  ctc_max_CJ  ctc_min_CJ  ctc_mean_CJE
ctc_max_CJE  \
0     500000  1.085000e+06     1100000     1070000  1.085000e+06
1100000
1      10000  9.212499e+05     2000000      300000  4.599995e+05
470000
2    2000000  2.000000e+06     2000000     2000000  2.000000e+06
2000000
3     200000  1.456667e+06     2600000      520000  1.358889e+06
1950000
4     540000  8.466667e+05     1400000      540000  1.000000e+06
1400000

   ctc_min_CJE class_C class_CJ class_CJE
0      1070000     Low     High      High
1       449999     Low      Low       Low
2      2000000     Mid      Mid       Mid
3       700000     Low      Low       Low
4       600000    High     High      High
```

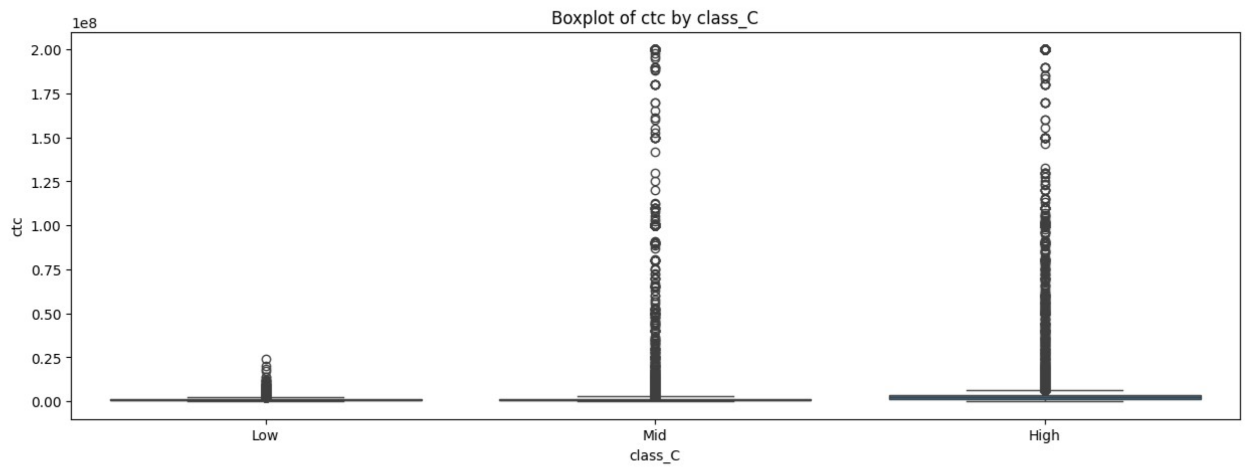
```python
# Boxplot for Numerical Data by Category
def cat_ctc(data, cat):
    plt.figure(figsize=(15, 5))
    sns.boxplot(x=cat, y='ctc', data=data)
    plt.title('Boxplot of ctc by {}'.format(cat))
    plt.show()

cat_ctc(df_merge, 'class_CJE')
```



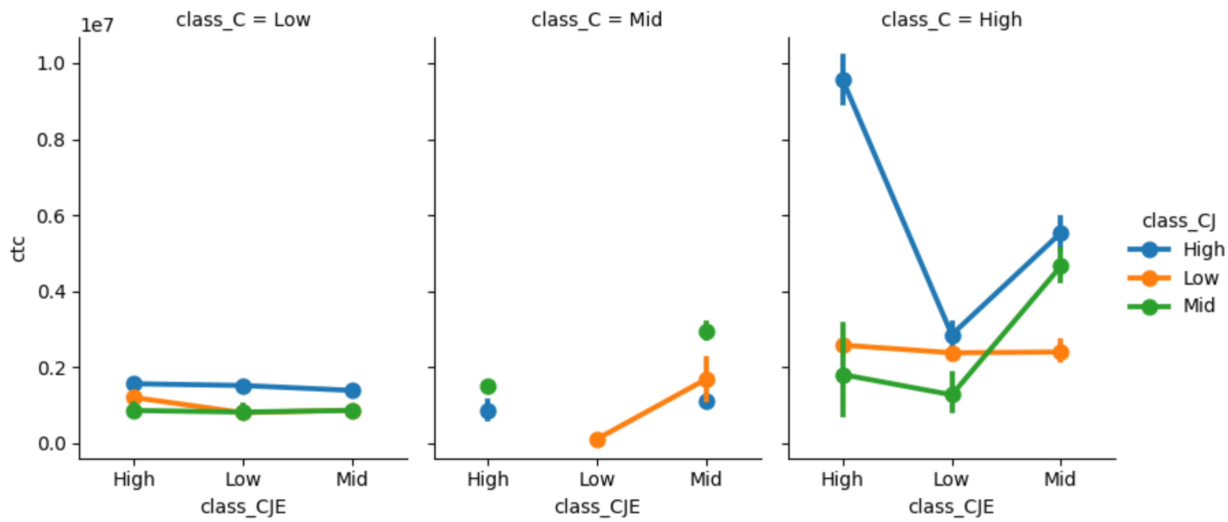

```python
cat_ctc(df_merge, 'class_CJ')
```




```python
cat_ctc(df_merge, 'class_C')
```




```python
# Catplot for Mixed Data
sns.catplot(x='class_CJE', y='ctc', hue='class_CJ', col='class_C',
data=df_merge, kind='point', height=4, aspect=0.7)
plt.show()
```



Section 6 : kmeans clustering

We are now ready with our dataset which has been :

- treted for outiers
- treated for other inconsistencies

- has been scaled
- has been imputed

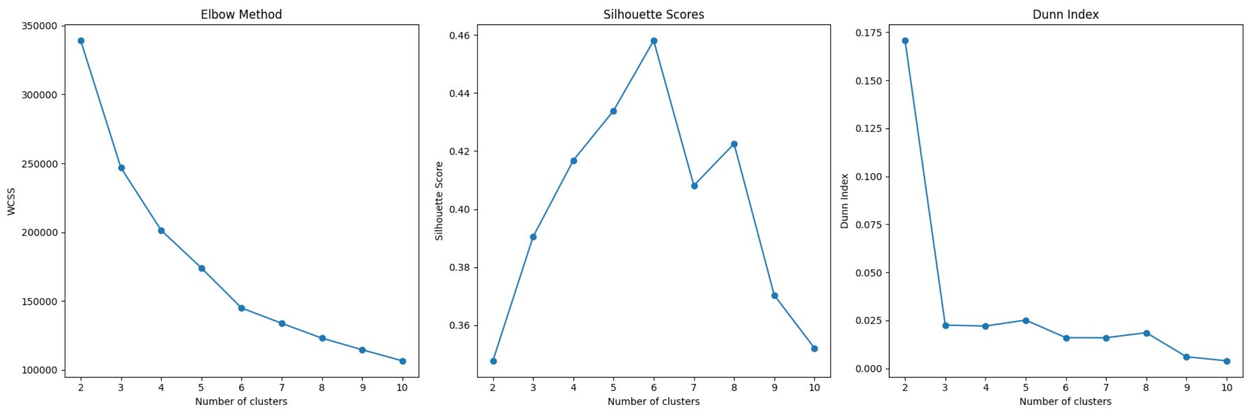
We can now proceed towards implementing kmeans. We will be using the elobow method, silhoutte and dunn index to check the performance of our clustering algorithm. Post this, we will see the clusters being formed for best parameters we receive from elbow method.

Finally, we will also try hierarchical clustering to see how that performs.

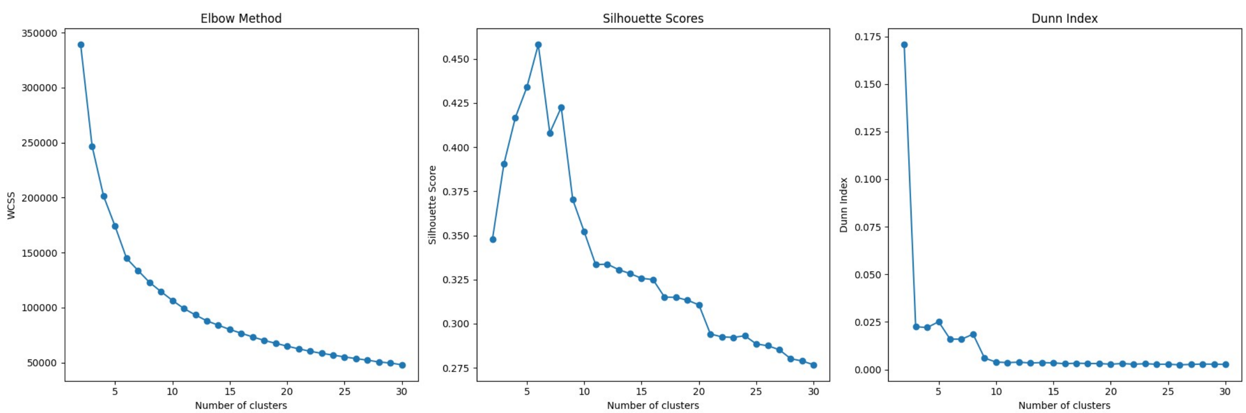
```
def compute_dunn_index(X, labels):
    clusters = np.unique(labels)
    n_clusters = clusters.shape[0]

    inter_cluster_distances = []
    intra_cluster_distances = []

    for i in range(n_clusters):
        cluster_i = X[labels == i]
        for j in range(i + 1, n_clusters):
            cluster_j = X[labels == j]

inter_cluster_distances.append(np.min(euclidean_distances(cluster_i,
cluster_j)))


intra_cluster_distances.append(np.max(euclidean_distances(cluster_i,
cluster_i)))

    dunn_index = np.min(inter_cluster_distances) /
np.max(intra_cluster_distances)
    return dunn_index

def run_kmeans_and_plot_2(data, max_clusters=10, batch_size=25000):
    """
    Runs KMeans clustering on the given data and plots the Elbow
method, Silhouette scores, and Dunn Index.

    Parameters:
    data (pd.DataFrame or np.array): The input data for clustering.
    max_clusters (int): The maximum number of clusters to consider for
the Elbow method, Silhouette scores, and Dunn Index.
    batch_size (int): The size of each batch for processing.

    Returns:
    None
    """
    # Step 1: Scaling the data
    #scaler = StandardScaler()
    #scaled_data = scaler.fit_transform(data)
```

```python
    # helper function to process each batch
    def process_batch(batch, k):
        kmeans = KMeans(n_clusters=k, random_state=42)
        labels = kmeans.fit_predict(batch)
        wcss_value = kmeans.inertia_
        silhouette_avg = silhouette_score(batch, labels)
        dunn_index = compute_dunn_index(batch, labels)
        return wcss_value, silhouette_avg, dunn_index

    # Initialize lists to store the results
    wcss = []  # Within-cluster sum of squares
    silhouette_scores = []
    dunn_indices = []

    # Run KMeans in parallel for different number of clusters and
compute metrics
    for k in range(2, max_clusters + 1):
        results = Parallel(n_jobs=-1)(
            delayed(process_batch)(data[i:i + batch_size], k) for i in
range(0, data.shape[0], batch_size)
        )

        # Aggregate results from all batches
        avg_wcss = np.mean([result[0] for result in results])
        avg_silhouette = np.mean([result[1] for result in results])
        avg_dunn = np.mean([result[2] for result in results])

        wcss.append(avg_wcss)
        silhouette_scores.append(avg_silhouette)
        dunn_indices.append(avg_dunn)

    # Plot the Elbow Method
    plt.figure(figsize=(18, 6))

    plt.subplot(1, 3, 1)
    plt.plot(range(2, max_clusters + 1), wcss, marker='o')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.title('Elbow Method')

    # Plot the Silhouette Scores
    plt.subplot(1, 3, 2)
    plt.plot(range(2, max_clusters + 1), silhouette_scores,
marker='o')
    plt.xlabel('Number of clusters')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Scores')

    # Plot the Dunn Index
```

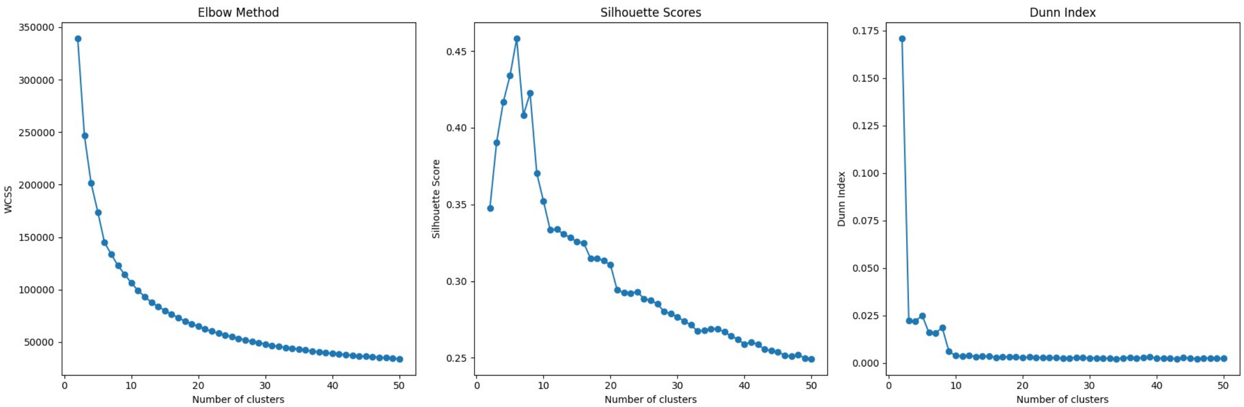
```
    plt.subplot(1, 3, 3)
    plt.plot(range(2, max_clusters + 1), dunn_indices, marker='o')
    plt.xlabel('Number of clusters')
    plt.ylabel('Dunn Index')
    plt.title('Dunn Index')

    plt.tight_layout()
    plt.show()

run_kmeans_and_plot_2(df_imputed_n3)
```




```
run_kmeans_and_plot_2(df_imputed_n3, max_clusters = 30)
```




```
run_kmeans_and_plot_2(df_imputed_n3, max_clusters = 50)
```

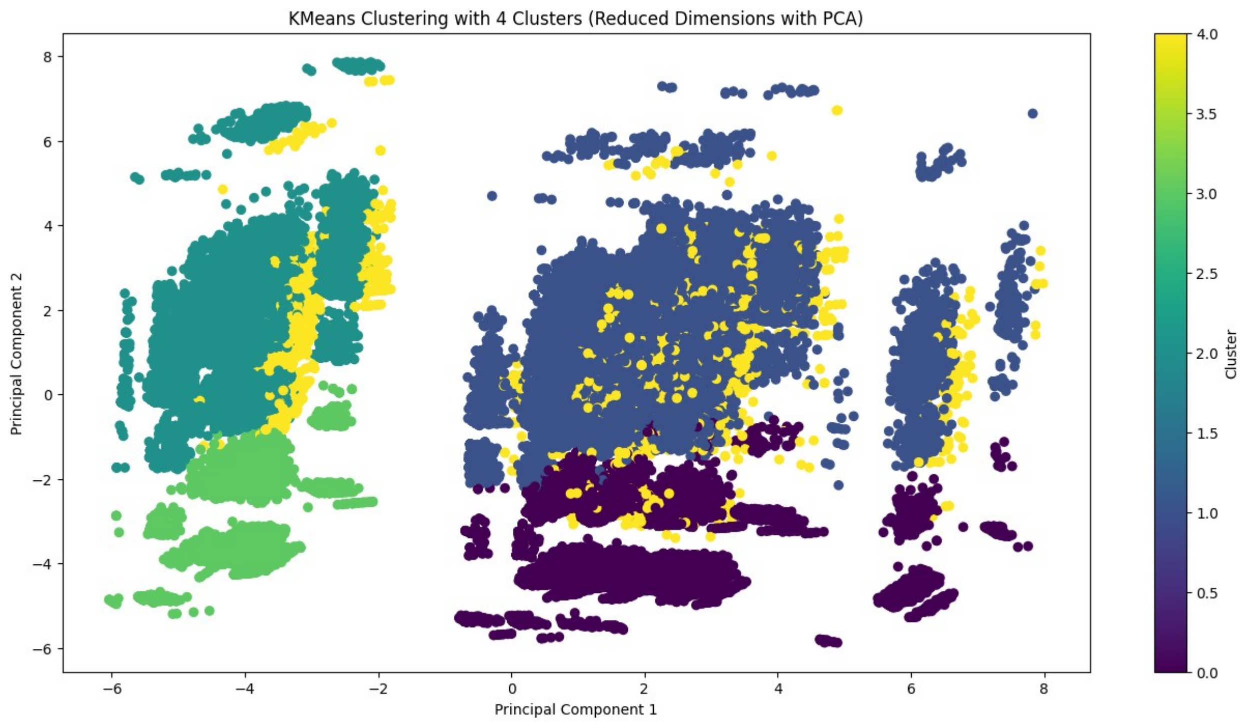


Observations :

- the elbow seems to be forming at 6 clusters
- the SIL score is highest 5 clusters

We will go ahead and re-run the kmens with 5 clusters to see how that performs

```
# Run KMeans with 5 clusters
kmeans = KMeans(n_clusters=5, random_state=42)
labels = kmeans.fit_predict(df_imputed_n3)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
pca_data = pca.fit_transform(df_imputed_n3)

# Plot the results
plt.figure(figsize=(16, 8))
plt.scatter(pca_data[:, 0], pca_data[:, 1], c=labels, cmap='viridis',
marker='o')
plt.title('KMeans Clustering with {} Clusters (Reduced Dimensions with
PCA)'.format(n_clusters))
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

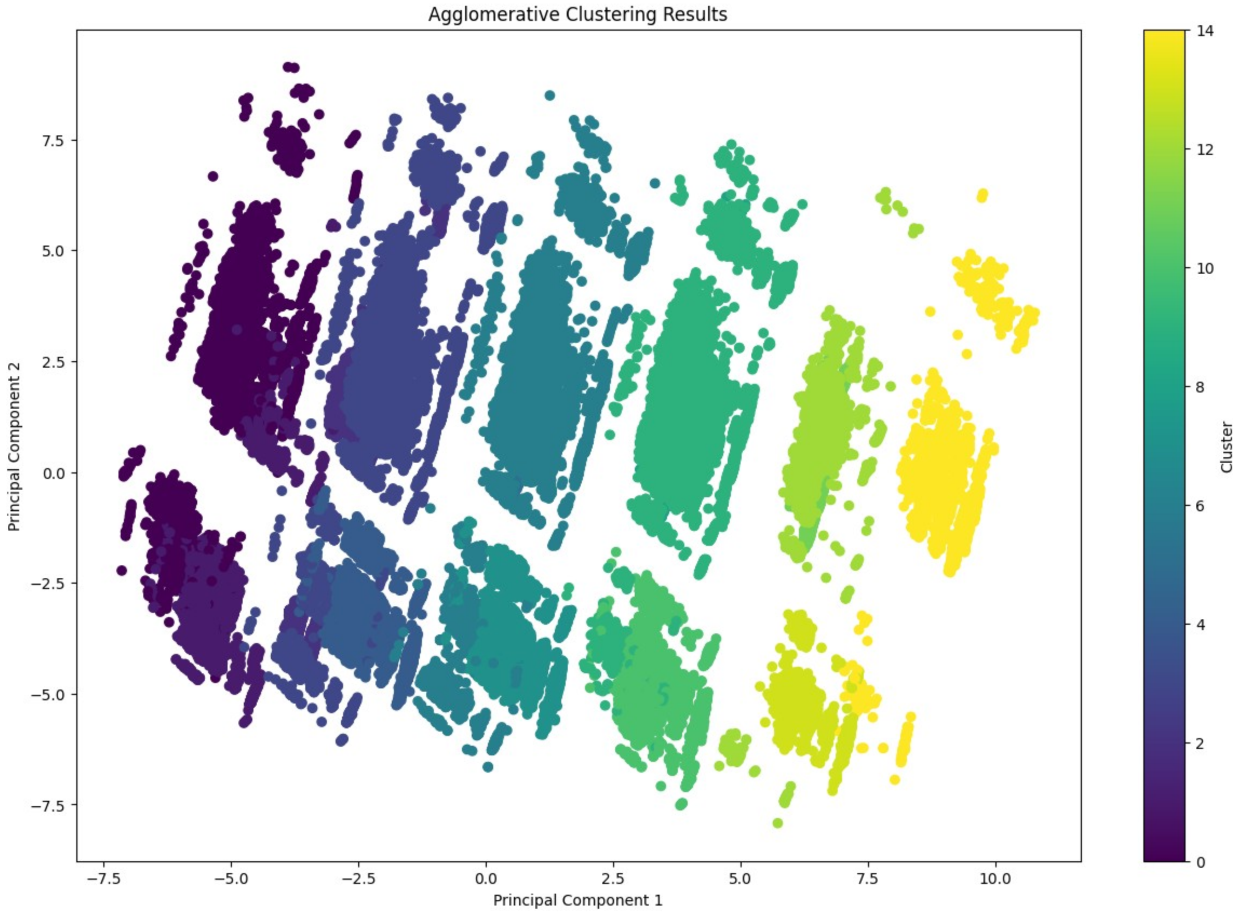


Observations :

We can see that kmeans has done a good job to segregate the clusters. We could have used tSNE instead of PCA to plot the performance.

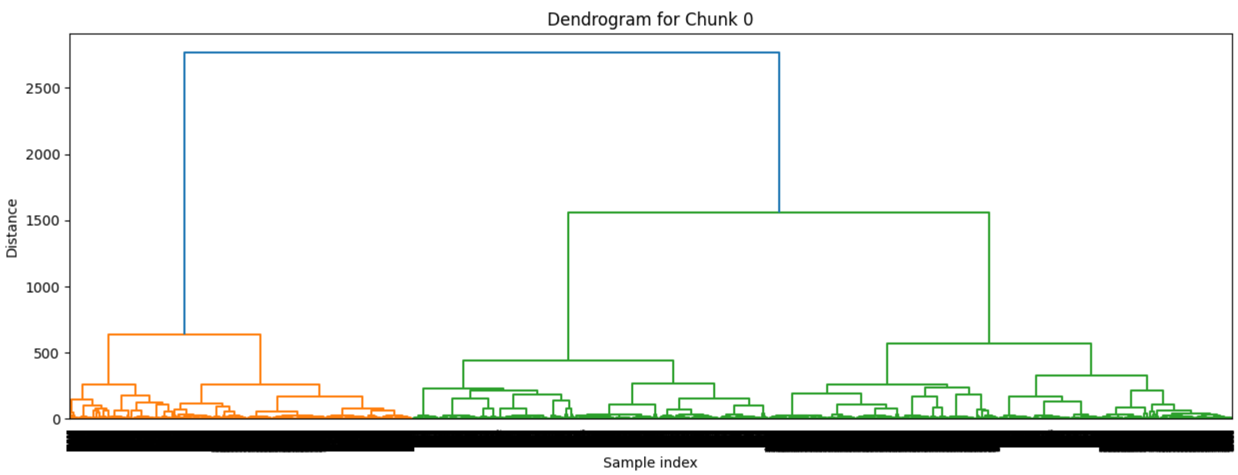
```
def agg_clust(data, n_clusters, batch_size=50000):
    """
    Perform agglomerative clustering on data split into smaller chunks
and aggregate the results.

    Parameters:
    data (pd.DataFrame or np.array): The input data for clustering.
    n_clusters (int): The number of clusters to form.
    batch_size (int): The size of each batch for processing.

    Returns:
    None
    """
    # Step 1: Scale the data
    #scaler = StandardScaler()
    #scaled_data = scaler.fit_transform(data)

    # Step 2: Define helper function to process each batch
    def process_batch(batch):
        hc = AgglomerativeClustering(n_clusters=n_clusters,
metric='euclidean', linkage='ward')
        labels = hc.fit_predict(batch)
        return labels
```

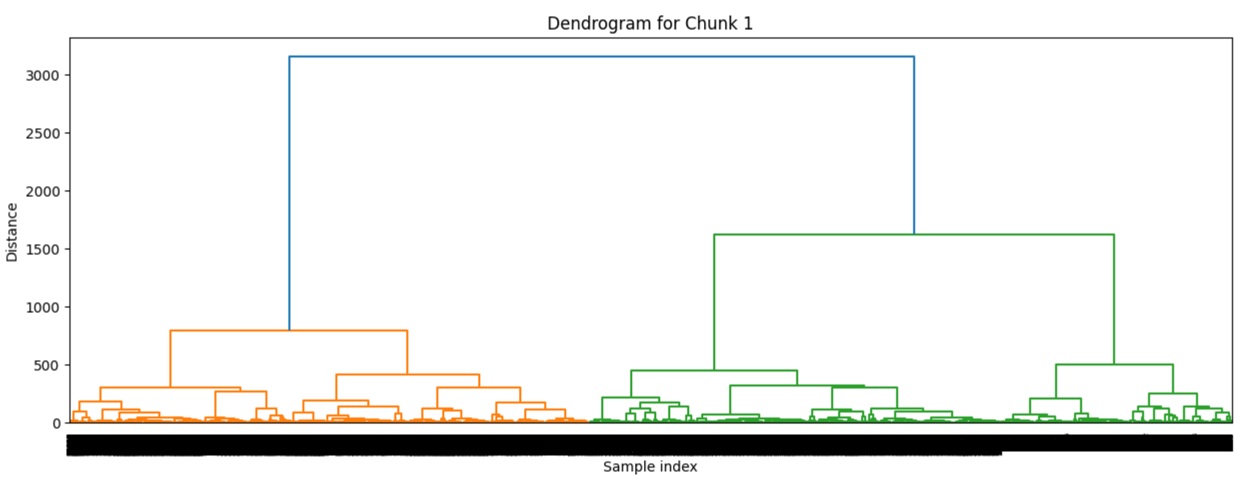
```
    # Initialize an empty array for labels
    labels = np.zeros(data.shape[0], dtype=int)

    # Step 3: Process each batch
    start_idx = 0
    for i in range(0, data.shape[0], batch_size):
        end_idx = min(i + batch_size, data.shape[0])
        batch_labels = process_batch(data[i:end_idx])
        labels[i:end_idx] = batch_labels + start_idx
        start_idx += n_clusters

    # Add cluster labels to the DataFrame
    data['Cluster'] = labels

    # Step 4: Reduce to 2D using PCA
    pca = PCA(n_components=2)
    pca_data = pca.fit_transform(data)

    # Step 5: Plot the results of Agglomerative Clustering
    plt.figure(figsize=(15, 10))
    plt.scatter(pca_data[:, 0], pca_data[:, 1], c=labels,
cmap='viridis', marker='o')
    plt.title('Agglomerative Clustering Results')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.colorbar(label='Cluster')
    plt.show()

agg_clust(df_imputed_n3, n_clusters = 3)
```



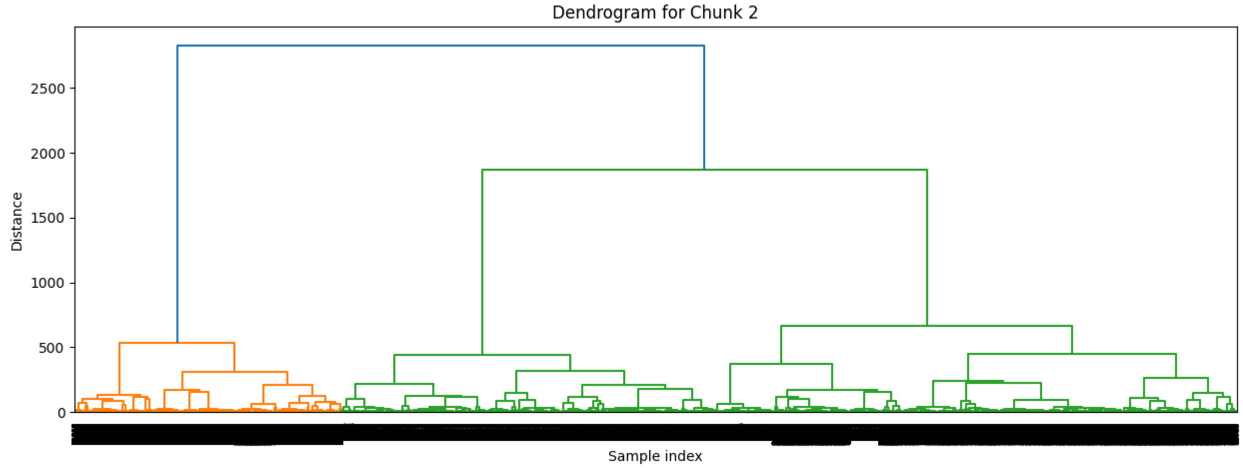

```
def plot_dendrogram_for_chunk(data, chunk_index):
    # Compute the Linkage Matrix
    linked = linkage(data, method='ward')

    # Create a Dendrogram for the chunk
    plt.figure(figsize=(15, 5))
    dendrogram(linked,
               orientation='top',
               distance_sort='ascending',
               show_leaf_counts=True)
    plt.title(f'Dendrogram for Chunk {chunk_index}')
    plt.xlabel('Sample index')
    plt.ylabel('Distance')
    plt.show()

def hierarchical_clustering_with_chunks(data, batch_size=1000):
    # Step 1: Scale the data
    #scaler = StandardScaler()
    #scaled_data = scaler.fit_transform(data)
```

```python
    # Step 2: Process each chunk and create dendrograms
    num_chunks = (data.shape[0] + batch_size - 1) // batch_size  # Calculate number of chunks
    for chunk_index in range(num_chunks):
        start_idx = chunk_index * batch_size
        end_idx = min(start_idx + batch_size, data.shape[0])
        data_chunk = data[start_idx:end_idx]

        # Plot dendrogram for the current chunk
        plot_dendrogram_for_chunk(data_chunk, chunk_index)

hierarchical_clustering_with_chunks(df_imputed_n3, batch_size=25000)
```

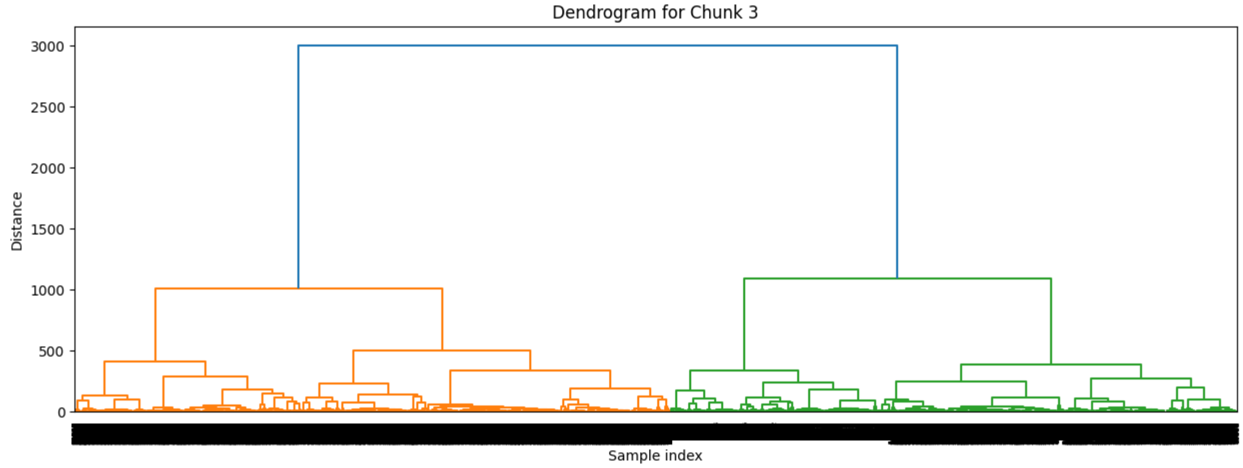




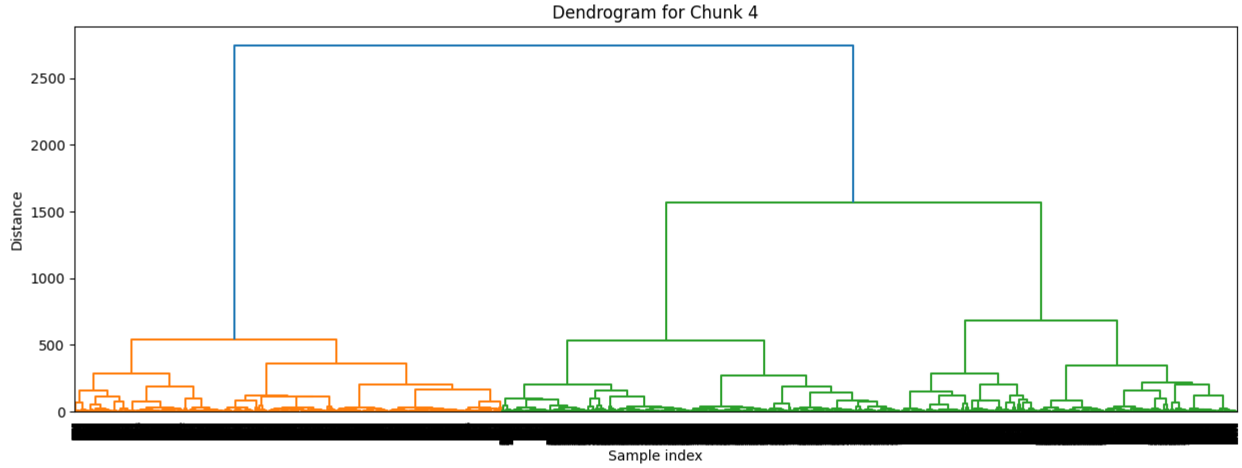
Dendrogram for Chunk 2

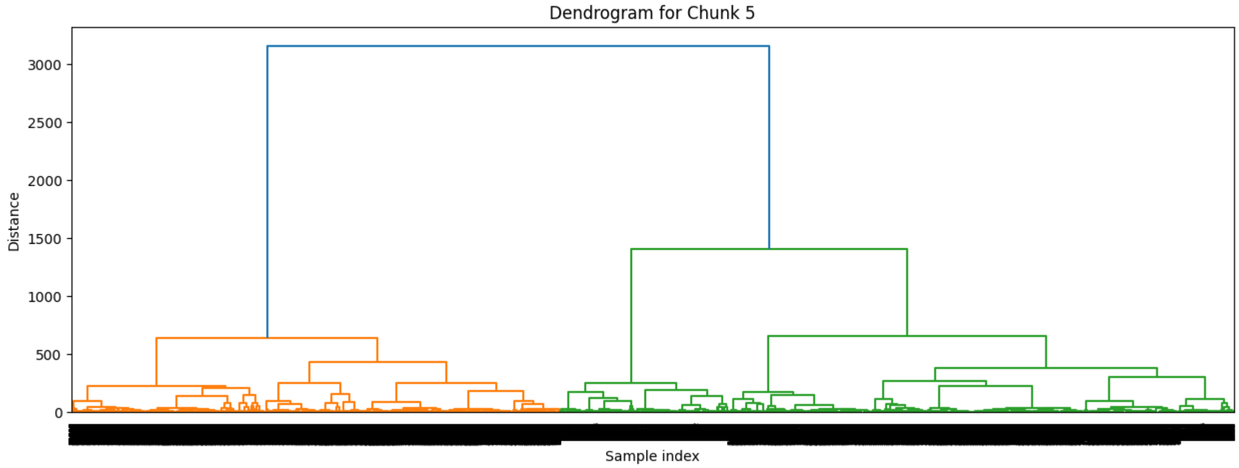


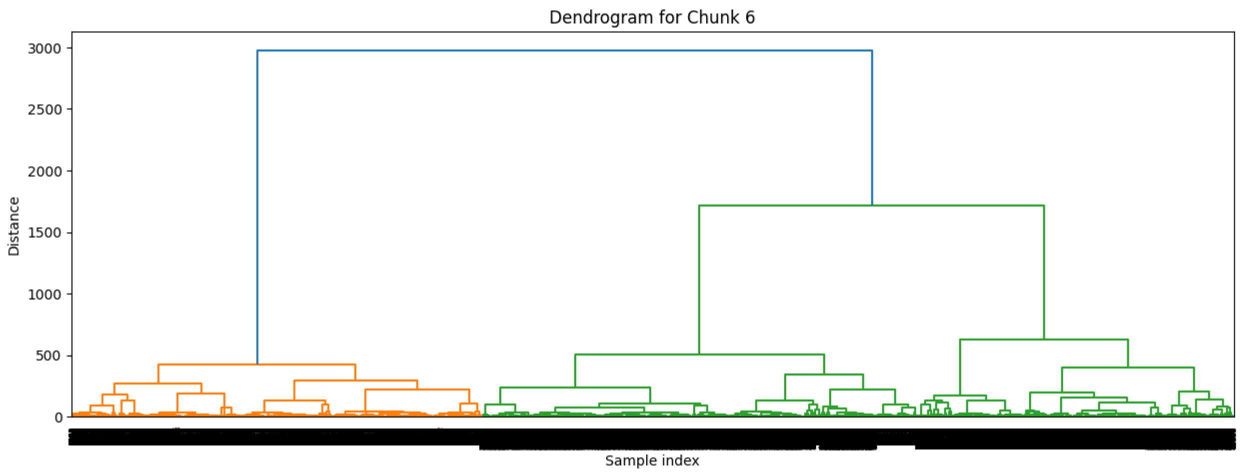
Dendrogram for Chunk 3

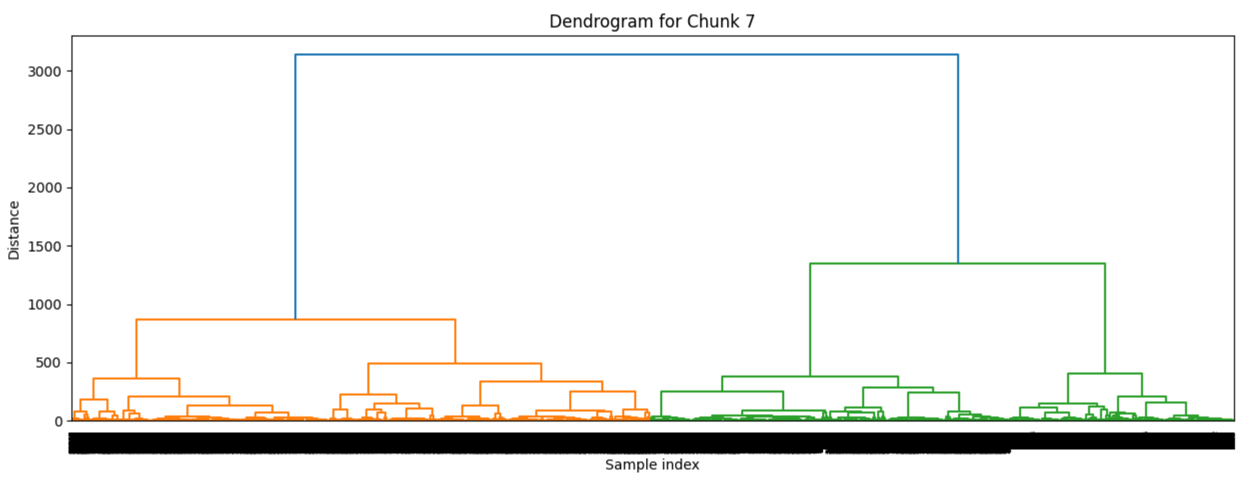


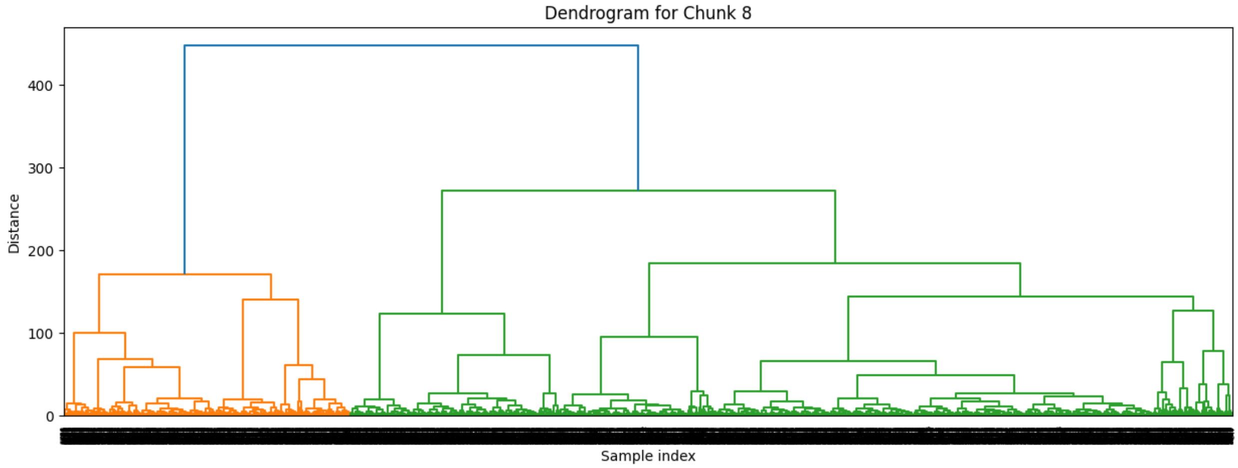
Dendrogram for Chunk 4




Dendrogram for Chunk 5
Distance
0
500
1000
1500
2000
2500
3000
Sample index


Dendrogram for Chunk 6
Distance
0
500
1000
1500
2000
2500
3000
Sample index


Dendrogram for Chunk 7
Distance
0
500
1000
1500
2000
2500
3000
Sample index



Observations :

Since the dataset is quite large, the dendogram is not giving out any usefull infomration. We will have choose a much smaller subset of the data to make some sense out of it. But that would not serve our purpose here and hence not continuing with it.

END