Ways to manage Employees to satisfy and retain the employees

Name: Soh Hong Yu

Admin Number: P2100775

Class: DAAA/FT/2A/01

Module Code: ST1511 AI and Machine Learning

References (In Harvard format):

1. Medium. 2022. Agglomerative Clustering and Dendrograms—Explained. [online] Available at: https://towardsdatascience.com/agglomerative-clustering-and-dendrograms-explained-29fc12b85f23 [Accessed 12 August 2022].
2. Medium. 2022. K-Means Clustering: How It Works & Finding The Optimum Number Of Clusters In The Data. [online] Available at: https://towardsdatascience.com/k-means-clustering-how-it-works-finding-the-optimum-number-of-clusters-in-the-data-13d18739255c [Accessed 12 August 2022].
3. Spiceworks. 2022. What Is Employee Attrition? Definition, Attrition Rate, Factors, and Reduction Best Practices | Spiceworks. [online] Available at: https://www.spiceworks.com/hr/engagement-retention/articles/what-is-attrition-complete-guide/ [Accessed 12 August 2022].
4. Medium. 2022. How DBSCAN works and why should we use it?. [online] Available at: https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80 [Accessed 12 August 2022].
5. Medium. 2022. Spectral Clustering. [online] Available at: https://towardsdatascience.com/spectral-clustering-aba2640c0d5b [Accessed 12 August 2022].
6. Analytics Vidhya. 2022. Hierarchical Clustering | Hierarchical Clustering Python. [online] Available at: https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering [Accessed 12 August 2022].
7. Medium. 2022. Cheat sheet for implementing 7 methods for selecting the optimal number of clusters in Python. [online] Available at: https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-

for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad [Accessed 12 August 2022].
8. Andrew Wheeler. 2022. PCA does not make sense after one hot encoding. [online] Available at: https://andrewpwheeler.com/2021/06/22/pca-does-not-make-sense-after-one-hot-encoding/ [Accessed 12 August 2022].
9. En.wikipedia.org. 2022. Hopkins statistic - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Hopkins_statistic [Accessed 12 August 2022].
10. Sonrai Analytics. 2022. What is tSNE and when should I use it? - Sonrai Analytics. [online] Available at: https://sonraianalytics.com/what-is-tsne/ [Accessed 12 August 2022].
11. Sartorius. 2022. What Is Principal Component Analysis (PCA) and How It Is Used?. [online] Available at: https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186 [Accessed 12 August 2022].
12. scikit-learn. 2022. Plot Hierarchical Clustering Dendrogram. [online] Available at: https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html#sphx-glr-auto-examples-cluster-plot-agglomerative-dendrogram-py [Accessed 12 August 2022].
13. Medium. 2022. Agglomerative Clustering and Dendrograms—Explained. [online] Available at: https://towardsdatascience.com/agglomerative-clustering-and-dendrograms-explained-29fc12b85f23 [Accessed 12 August 2022].
14. Skillsfuture.gov.sg. 2022. SSG | SkillsFuture Credit. [online] Available at: https://www.skillsfuture.gov.sg/credit [Accessed 12 August 2022].
15. Ludwig, S., 2022. How to Retain Your Best Employees. [online] https://www.uschamber.com/co. Available at: https://www.uschamber.com/co/run/human-resources/how-to-retain-top-employees [Accessed 12 August 2022].

Project Objective

As a HR manager, you want to understand your employees so that appropriate direction can be given to the management to satisfy and retain the employees.

Potential Reasons for leaving the company (Attrition)

1. Attrition due to retirement
2. Voluntary attrition
3. Involuntary attrition

4. Internal attrition
5. Demographic-specific attrition

Initialising Libraries and Variables

In [ ]:
```
# import libraries/packages/functions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import pickle
from pathlib import Path
from sklearn import metrics

from sklearn.decomposition import PCA
```

Loading Datasets

In [ ]:
```
df = pd.read_csv("./Company_Employee.csv", sep=",")
df.head()
```

Out[ ]:

	Age	Gender	BusinessTravel	Job Function	Distance Between Company and Home (KM)	Education (1 is lowest, 5 is highest)	Job Satisfaction (1 is lowest, 4 is highest)	MaritalStatus
0	41	Female	Travel_Rarely	Sales	1	2	4	Single
1	49	Male	Travel_Frequently	Research & Development	8	1	2	Married
2	37	Male	Travel_Rarely	Research & Development	2	2	3	Single
3	33	Female	Travel_Frequently	Research & Development	3	4	3	Married
4	27	Male	Travel_Rarely	Research & Development	2	1	2	Married

Exploratory Data Analysis

We will begin by conducting an exploratory data analysis of the data, to gain a better understanding of the characteristics of the dataset.

This is a dataset collected from a HR department in a company, it contains 1470 data points with 13 columns.

Age: Employee age
Gender: Employee gender
BusinessTravel: Business travel frequency for the employee
Job Function: Department of the employee
Distance Between Company and Home (KM): distance between company and home
Education (1 is lowest, 5 is highest): Education qualification of employee
Job Satisfaction (1 is lowest, 4 is highest): Employee's job satisfaction
MaritalStatus: Marital status of employee
Salary ($): Employee's salary
Performance Rating (1 is lowest, 4 is highest): Employee's performance rating in the company
Work Life Balance (1 is worst, 4 is best): Work life balance rating
Length of Service (Years): How many years the employee works for the company
Resign Status: Is the employee still with the company

Data Exploration

To prevent mutation of our original data, we will make a copy of our data to perform eda on it.

In [ ]:
```
df_eda = df.copy()
```

Descriptive Statistics

In [ ]:
```
df_eda.shape
```

Out[ ]:
```
(1470, 13)
```

There are 1470 rows and 13 columns in the entire data set.

In [ ]:
```
df_eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 13 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Age                                             1470 non-null   int64
 1   Gender                                          1470 non-null   object
 2   BusinessTravel                                  1470 non-null   object
 3   Job Function                                    1470 non-null   object
 4   Distance Between Company and Home (KM)          1470 non-null   int64
 5   Education (1 is lowest, 5 is highest)           1470 non-null   int64
 6   Job Satisfaction (1 is lowest, 4 is highest)    1470 non-null   int64
 7   MaritalStatus                                   1470 non-null   object
 8   Salary ($)                                      1470 non-null   int64
 9   Performance Rating (1 is lowest, 4 is highest)  1470 non-null   int64
 10  Work Life Balance (1 is worst, 4 is best)       1470 non-null   int64
 11  Length of Service (Years)                       1470 non-null   int64
 12  Resign Status                                   1470 non-null   object
dtypes: int64(8), object(5)
memory usage: 149.4+ KB
```

Observations

- The shape of dataset is (1470, 13)
- There are no missing values in the dataset.

Data Information

In [ ]:
```
descriptive_stats = df_eda.describe(include="all").T
descriptive_stats["Proportion of Most Frequent Value"] = (
    descriptive_stats["freq"] / len(df_eda) * 100
)
descriptive_stats.sort_values("Proportion of Most Frequent Value", ascending=False)
```

Out[ ]:

	count	unique	top	freq	mean	std	min	25%	5(
Resign Status	1470	2	No	1233	NaN	NaN	NaN	NaN	Na
BusinessTravel	1470	3	Travel_Rarely	1043	NaN	NaN	NaN	NaN	Na
Job Function	1470	3	Research & Development	961	NaN	NaN	NaN	NaN	Na
Gender	1470	2	Male	882	NaN	NaN	NaN	NaN	Na
MaritalStatus	1470	3	Married	673	NaN	NaN	NaN	NaN	Na
Age	1470.0	NaN	NaN	NaN	36.92381	9.135373	18.0	30.0	3(
Distance Between Company and Home (KM)	1470.0	NaN	NaN	NaN	9.192517	8.106864	1.0	2.0	7
Education (1 is lowest, 5 is highest)	1470.0	NaN	NaN	NaN	2.912925	1.024165	1.0	2.0	3
Job Satisfaction (1 is lowest, 4 is highest)	1470.0	NaN	NaN	NaN	2.728571	1.102846	1.0	2.0	3
Salary ($)	1470.0	NaN	NaN	NaN	6502.931293	4707.956783	1009.0	2911.0	491
Performance Rating (1 is lowest, 4 is highest)	1470.0	NaN	NaN	NaN	3.153741	0.360824	3.0	3.0	3
Work Life Balance (1 is worst, 4 is best)	1470.0	NaN	NaN	NaN	2.761224	0.706476	1.0	2.0	3
Length of Service (Years)	1470.0	NaN	NaN	NaN	7.008163	6.126525	0.0	3.0	5

Pandas-Profiling

Pandas-Profiling is a convenient tool to quickly explore the datasets along with some alerts/warnings about the dataset.

In [ ]:
```
from pandas_profiling import ProfileReport
```

```
prof = ProfileReport(df_eda, explorative=True)
# prof.to_file(output_file='employees.html')
```

In [ ]:
```
# Summary 1
print("Summary")
print(
    f"1. Salary ($) is correlated with Length of Service (Years) = {df_eda.corr()[
)
print(
    f"2. Age is correlated with Length of Service (Years) = {df_eda.corr()['Age'][
)
print(f"3. Salary ($) is correlated with Age = {df_eda.corr()['Salary ($)']['Age']]
print(
    f"4. Number of zeros in Length of Service (Years): {(df_eda['Length of Service
)
```

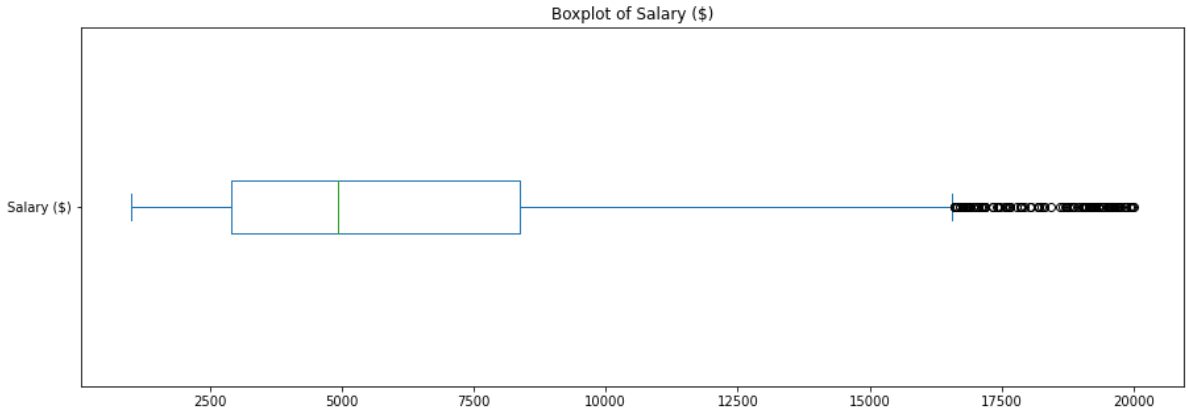
```
Summary
1. Salary ($) is correlated with Length of Service (Years) = 0.5142848257331957
2. Age is correlated with Length of Service (Years) = 0.3113087697450989
3. Salary ($) is correlated with Age = 0.4978545669265801
4. Number of zeros in Length of Service (Years): 44
Percentage of dataset: 2.9931972789115644%
```

Univariate Analysis

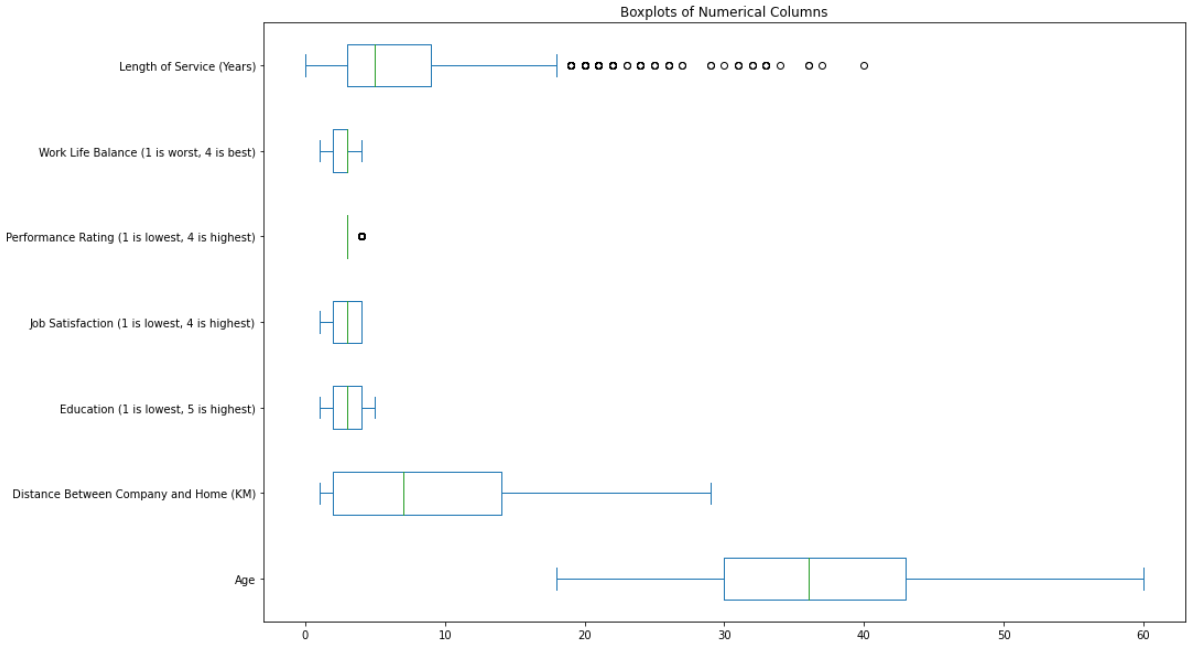
We will begin with a univariate analysis, analysing the distribution of each variable.

In [ ]:
```
continuous = [
    "Age",
    "Distance Between Company and Home (KM)",
    "Education (1 is lowest, 5 is highest)",
    "Job Satisfaction (1 is lowest, 4 is highest)",
    "Salary ($)",
    "Performance Rating (1 is lowest, 4 is highest)",
    "Work Life Balance (1 is worst, 4 is best)",
    "Length of Service (Years)",
]
```

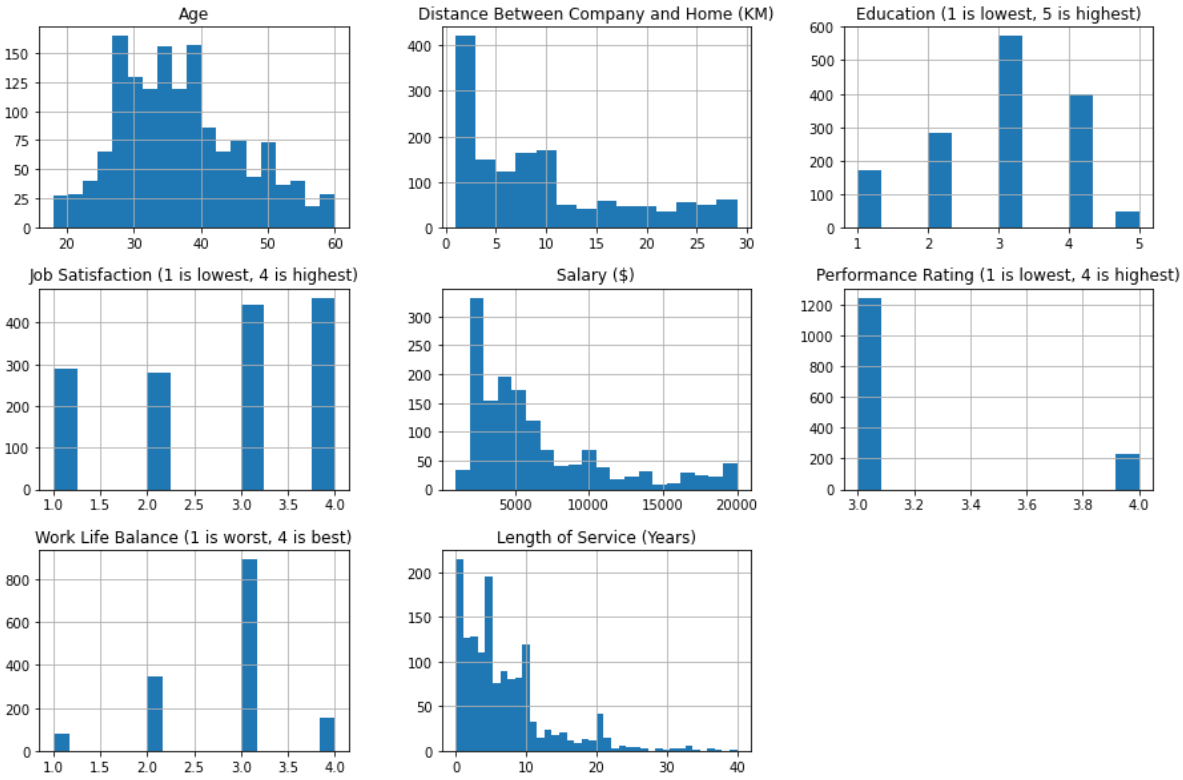
In [ ]:
```
fig, ax = plt.subplots(figsize=(15, 5))
df_eda["Salary ($)"].plot.box(vert=False, ax=ax)
plt.title("Boxplot of Salary ($)")
plt.show()
```



In [ ]:
```
fig, ax = plt.subplots(figsize=(15, 10))
df_eda[continuous].drop("Salary ($)", axis=1).plot.box(vert=False, ax=ax)
plt.title("Boxplots of Numerical Columns")
plt.show()
```



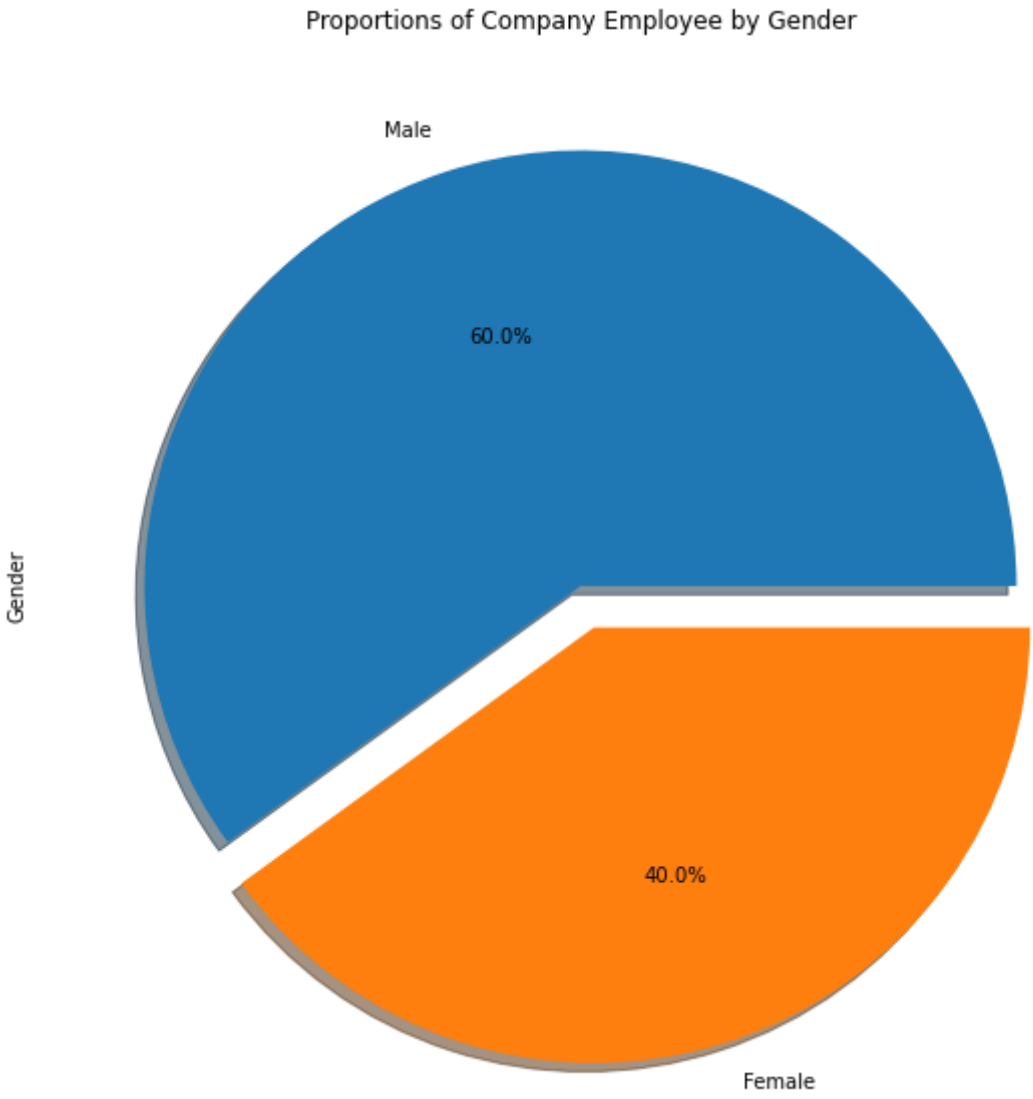
In [ ]:
```
# distribution of numerical features
fig, ax = plt.subplots(figsize=(15, 10))
df_eda[continuous].hist(bins="auto", ax=ax)
plt.show()
```



Observations

- Age seems to have normal distribution

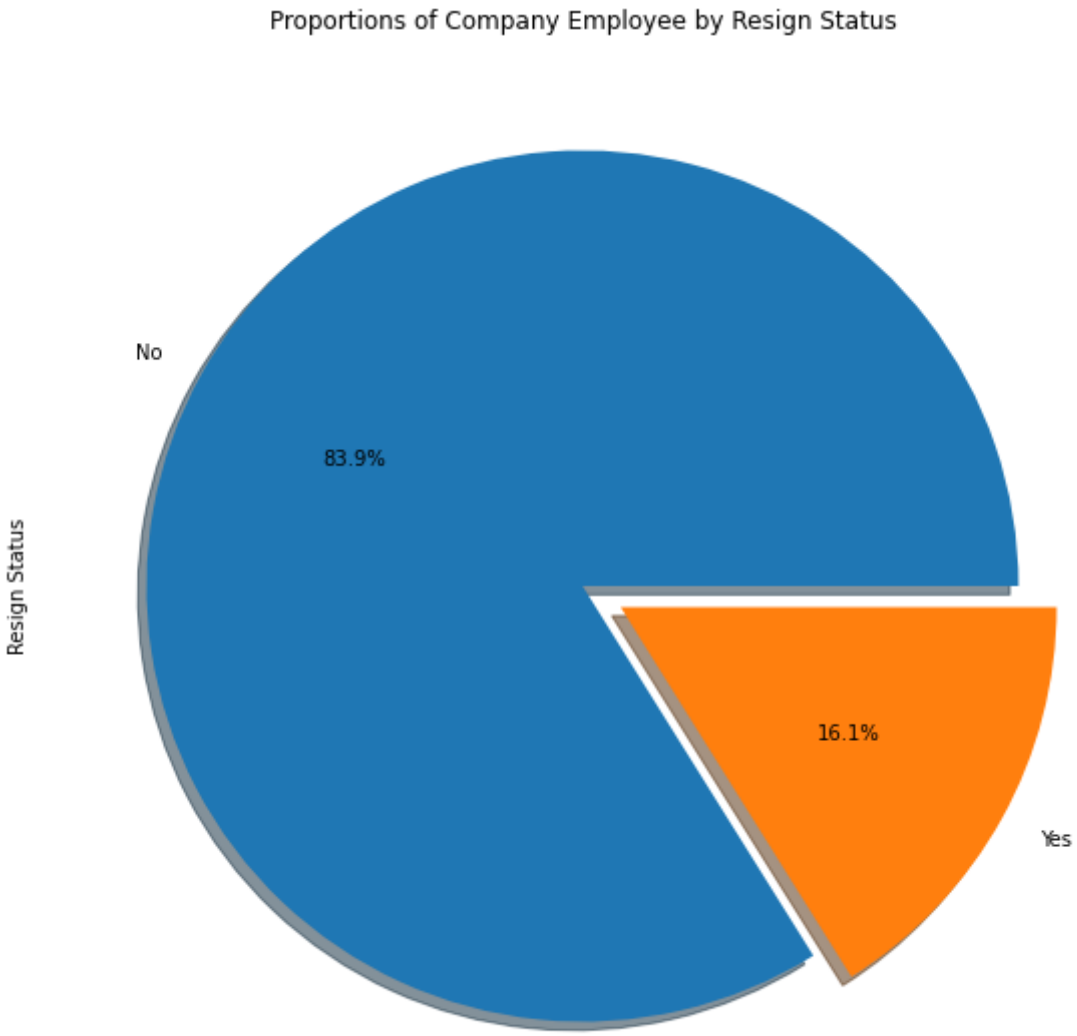
In [ ]:
```
# pie chart on gender
fig, ax = plt.subplots(figsize=(15, 10))
fig.set_facecolor("white")
df_eda["Gender"].value_counts().plot.pie(
    title="Proportions of Company Employee by Gender",
    legend=False,
    autopct="%1.1f%%",
    explode=(0, 0.1),
    shadow=True,
    startangle=0,
)
plt.show()
```



Observations

- Overall, there are more female customers compare to male customers.
- The difference in margin is not so significant to consider the class to be imbalanced.

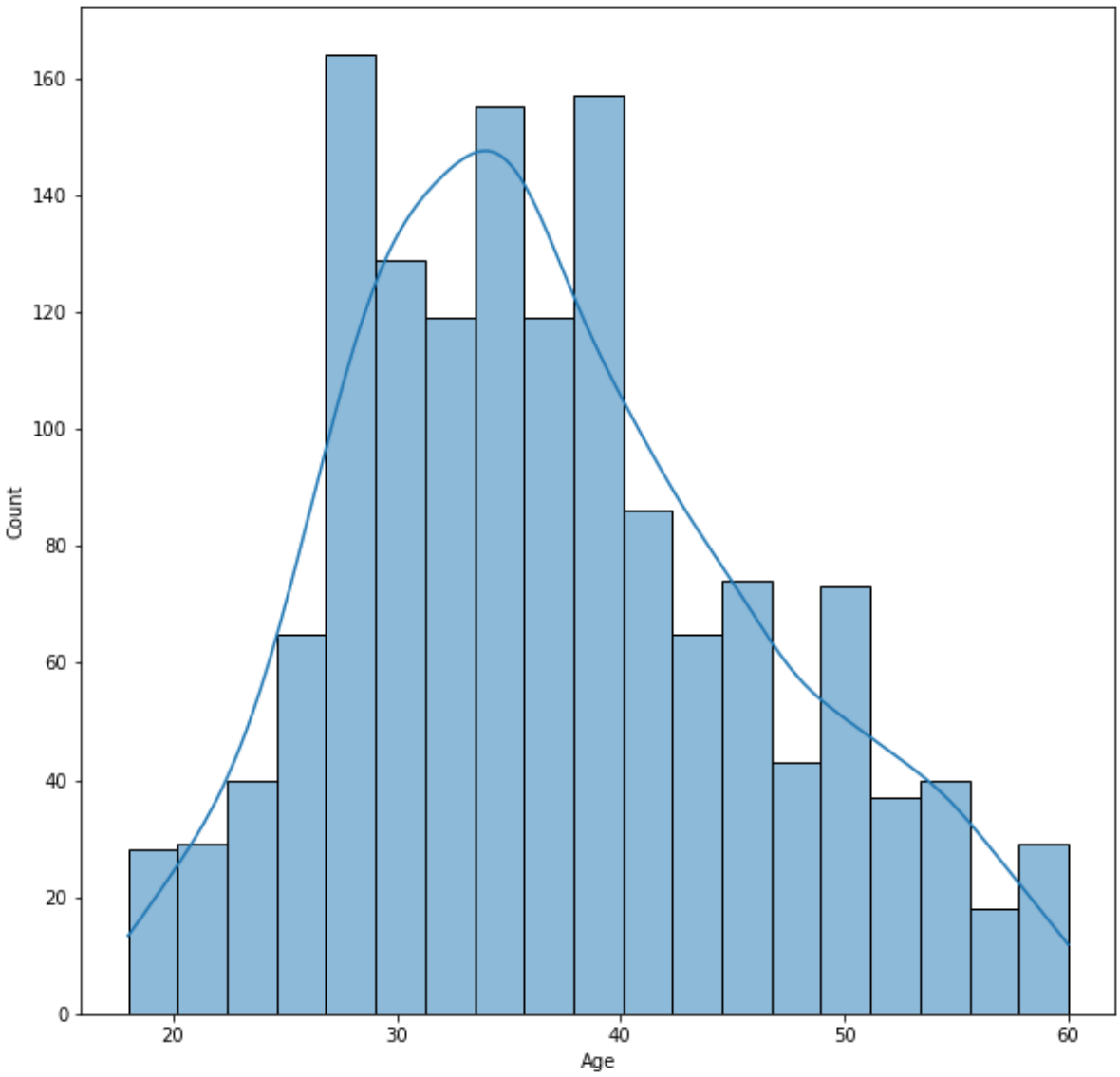
In [ ]:
```
# pie chart on resign status
fig, ax = plt.subplots(figsize=(15, 10))
fig.set_facecolor("white")
df_eda["Resign Status"].value_counts().plot.pie(
    title="Proportions of Company Employee by Resign Status",
    legend=False,
    autopct="%1.1f%%",
    explode=(0, 0.1),
    shadow=True,
    startangle=0,
)
plt.show()
```



Observations

- Overall, there is a minority in people who resign and not stay in the job.
- 84% of the employees in the dataset have not left the company.

In [ ]:
```
plt.figure(figsize=(10, 10))
sns.histplot(df_eda["Age"], kde=True)
plt.show()
```



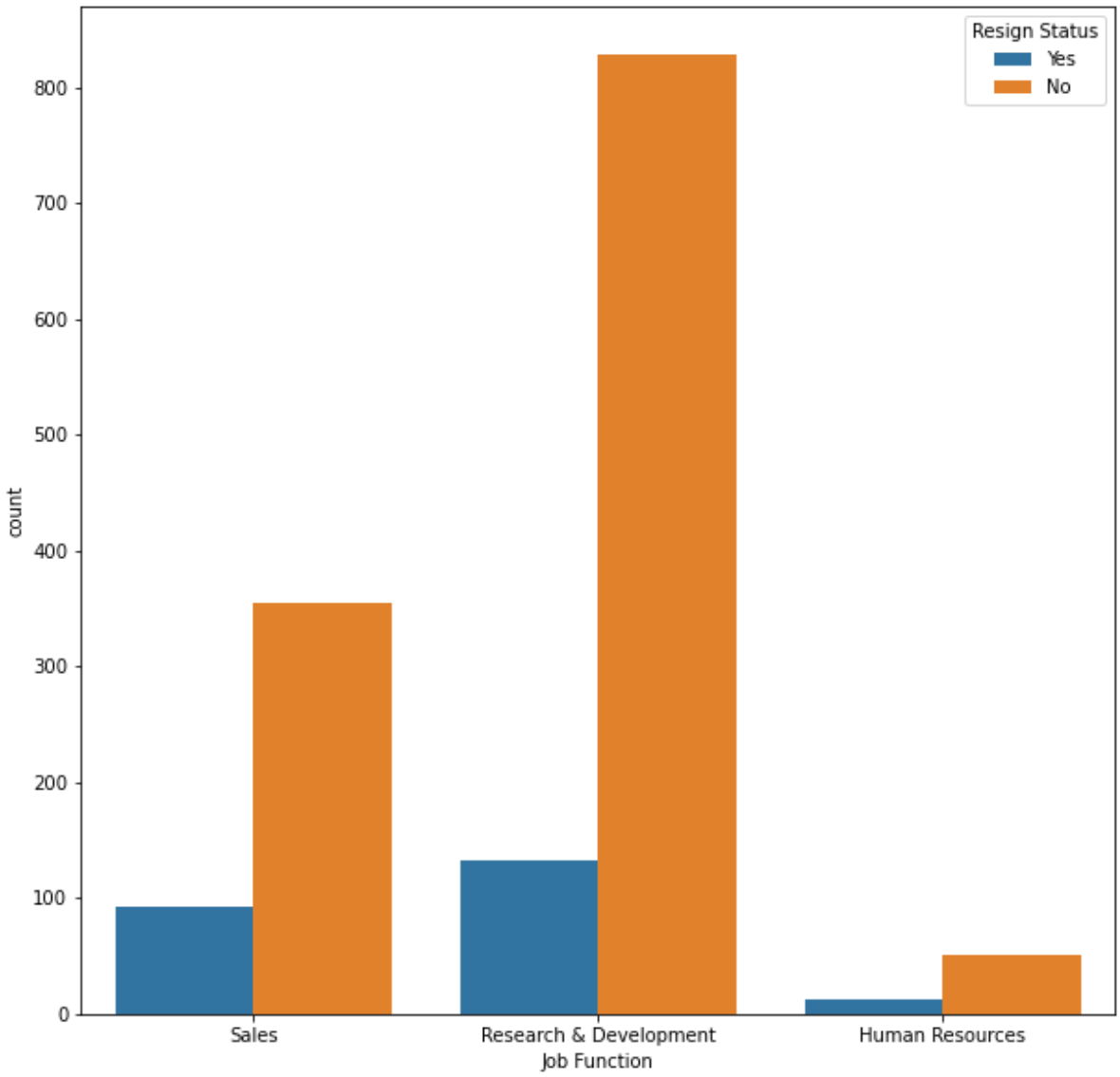
Observations

- Most employees are in their 30s with 35 year old's having the highest count and lowest are people at around the age 60 or less than 20.

Bivariate Analysis

Bivariate Analysis refers to the analysis of two variables. In this section, the key thing we want to find is the linearity between any two variables of this dataset.

In [ ]:
```
plt.figure(figsize=(10, 10))
sns.countplot(data=df_eda, x="Job Function", hue="Resign Status")
plt.show()
```

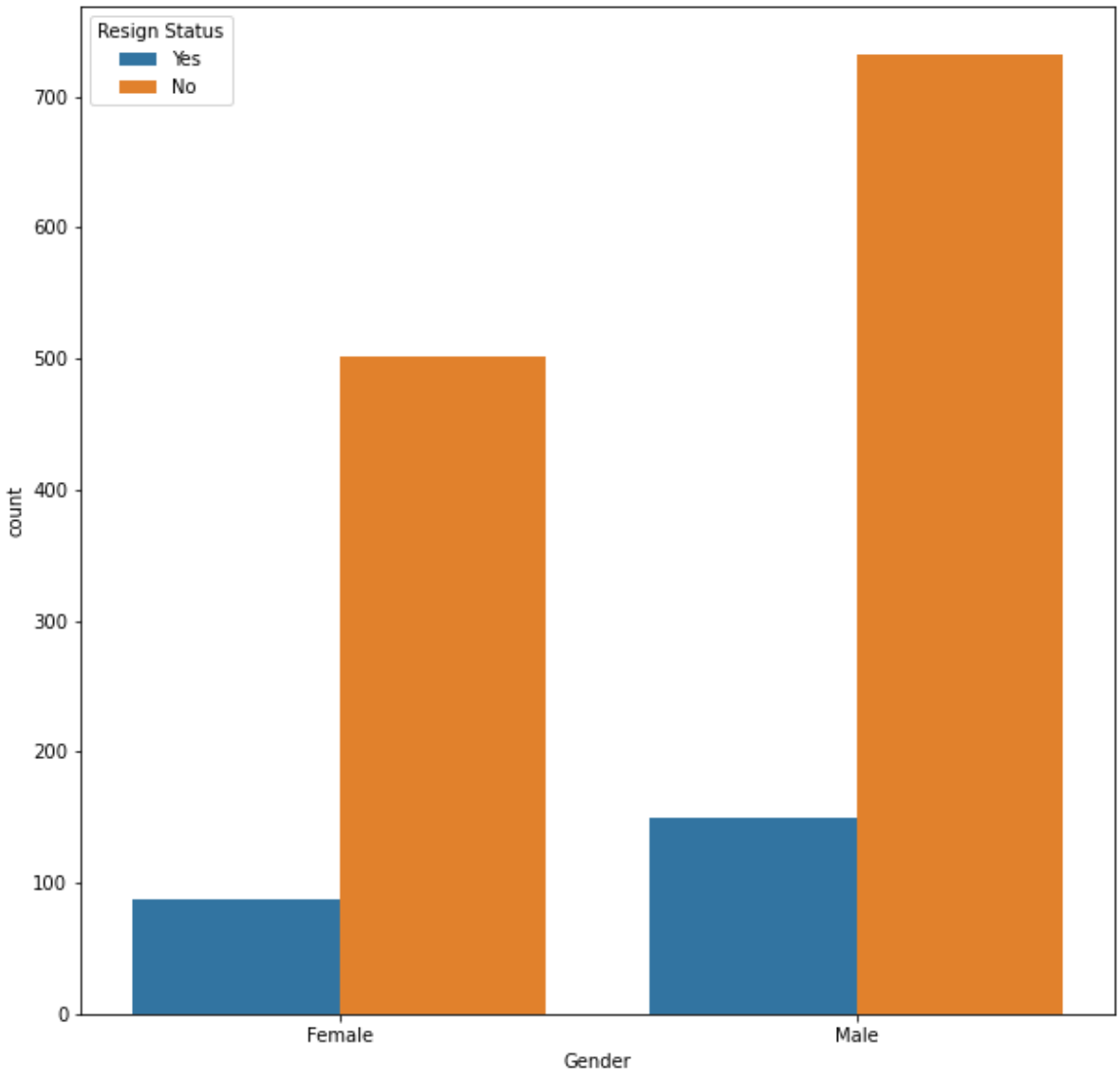


Observations

- Most resignation are from the research & development department only for sales department to come second by a small margin. HUman resources has the least number of resignation. But we need to keep in mind that R&D has a lot more employees than sales and HR.

If we considered percentage of resignation per department, we would see that the HR department has most resignation.

In [ ]:
```
plt.figure(figsize=(10, 10))
sns.countplot(data=df_eda, x="Gender", hue="Resign Status")
plt.show()
```



Observations

- There are more males in the organisation than females, so resignation are higher but slightly. Therefore, gender is not a significant a factor behind resignation status.

Pearson's r Correlations

To check for correlation between the features and target, we will make use Pearson's r correlation coefficient. The Pearson's r correlation is able to measure the linear correlation between the two variables. Furthermore, r is invariant under separate changes in location and scale of the two variables, implying that for a linear function the angle to the x-axis does not affect r.

To calculate r for two variables X and Y, one divides the covariance of X and Y by the product of their standard deviations.

Formula:

$$\rho = \frac{\operatorname{cov}(X, Y)}{\sigma_x \sigma_y} \tag{1}$$

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}} \tag{2}$$

Encode Resign Status

As Pearson's r correlations can only work with qualitative variables, we will be encoding Resign Status.

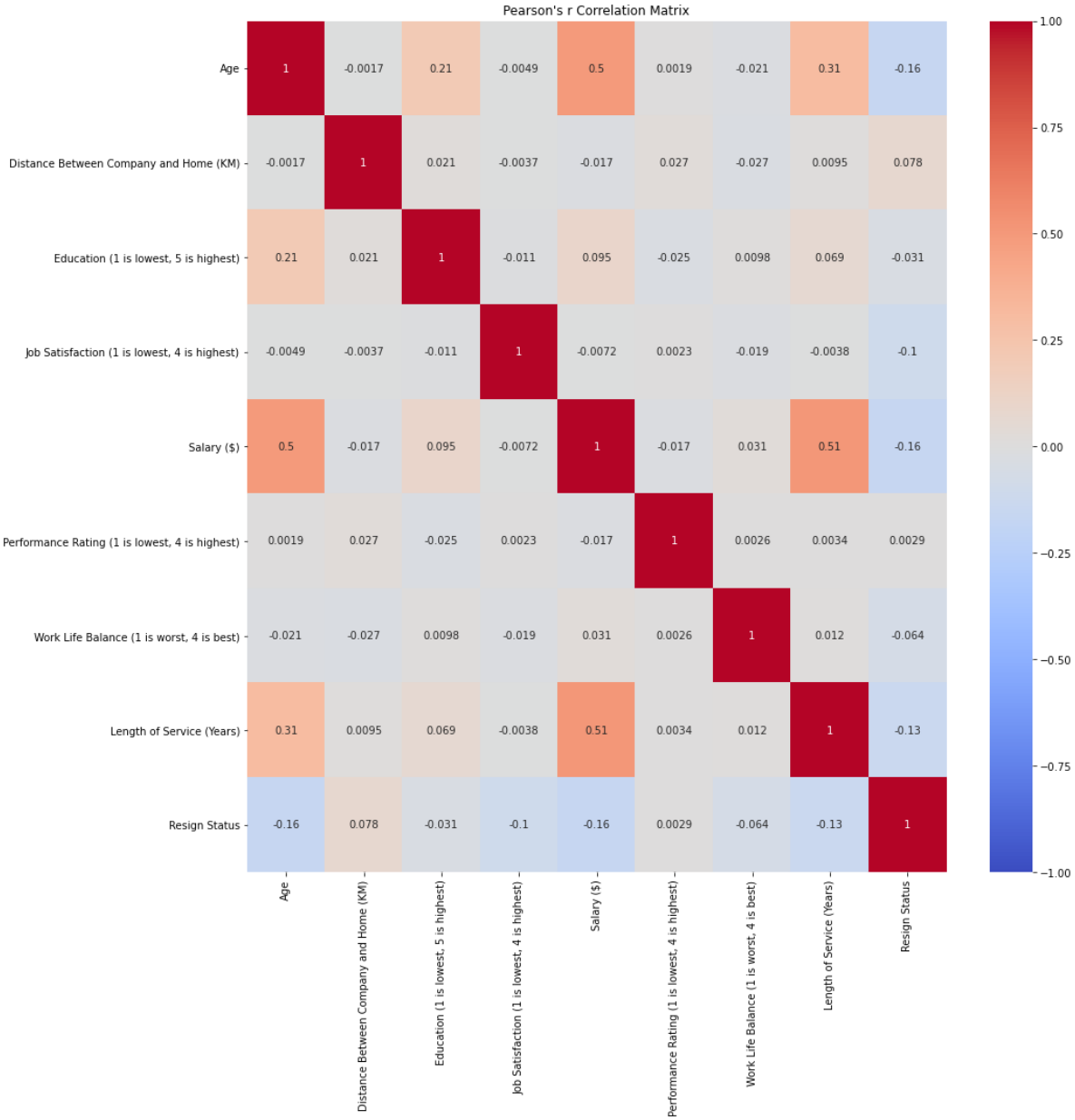
In [ ]:
```
df_eda["Resign Status"] = pd.Series(
    np.where(df_eda["Resign Status"].values == "Yes", 1, 0), df_eda.index
)
df_eda
```

Out[ ]:

	Age	Gender	BusinessTravel	Job Function	Distance Between Company and Home (KM)	Education (1 is lowest, 5 is highest)	Job Satisfaction (1 is lowest, 4 is highest)	MaritalStat
0	41	Female	Travel_Rarely	Sales	1	2	4	Sin
1	49	Male	Travel_Frequently	Research & Development	8	1	2	Marri
2	37	Male	Travel_Rarely	Research & Development	2	2	3	Sin
3	33	Female	Travel_Frequently	Research & Development	3	4	3	Marri
4	27	Male	Travel_Rarely	Research & Development	2	1	2	Marri
...	...	...	...	...	...	...	...	
1465	36	Male	Travel_Frequently	Research & Development	23	2	4	Marri
1466	39	Male	Travel_Rarely	Research & Development	6	1	1	Marri
1467	27	Male	Travel_Rarely	Research & Development	4	3	2	Marri
1468	49	Male	Travel_Frequently	Sales	2	3	2	Marri
1469	34	Male	Travel_Rarely	Research & Development	8	3	3	Marri

1470 rows × 13 columns

In [ ]:
```
fig, ax = plt.subplots(figsize=(15, 15))
# pearson correlation matrix
sns.heatmap(data=df_eda.corr(), annot=True, cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Pearson's r Correlation Matrix")
plt.show()
```

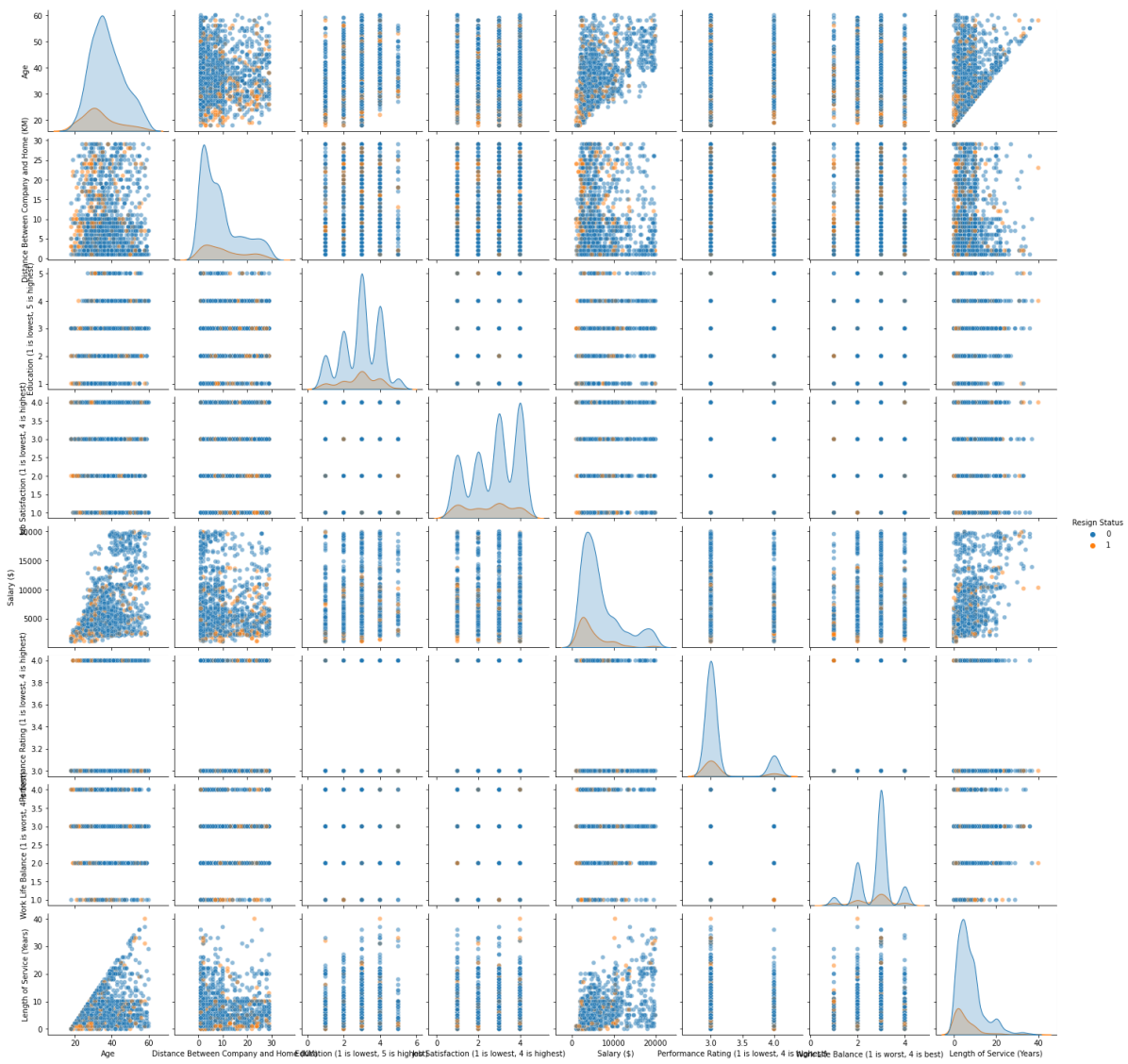


Observations

- Most variables have little to no linear correlation between variables. There is no pattern in the scatterplot between variables.
- Age vs Salary; 0.5 r value indicates there is a moderate positive correlation between age and salary. A "cone" shape pattern that can be seen from the scatterplot which indicates that there is a larger variation of salary when you are older.
- Length of Service vs Salary; 0.51 r value indicates that there is a moderate positive correlation between length of service and salary. A clear cluster in the length of service vs salary where salary is at a range from around $3000 to 7500$ and length of service is at range from 5 to 10 years.
- Age vs Length of Service; 0.31 r value indicates that there is a weak positive correlation between age and length of service. As the older you are, one is more likely to serve in

the company longer.

In [ ]:
```
# pearson correlation matrix
sns.pairplot(data=df_eda, hue="Resign Status", plot_kws={"alpha": 0.5})
plt.show()
```



Observations

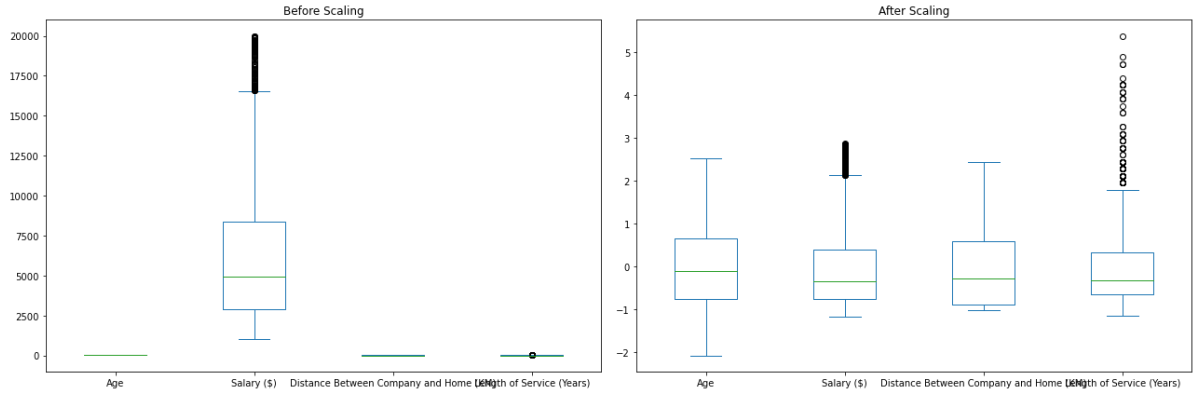
- There are no clear clusters that formed which means some feature selection and engineering will need to be done
- Certain variables as mentioned in the correlation section has a slight linear relationship. One example is Age vs Length of Service

Data Preparation

Before we train the machine learning algorithm on our data, we first need to prepare our data.

In [ ]:
```
from sklearn.preprocessing import StandardScaler

num_cols = [
    "Age",
    "Salary ($)",
    "Distance Between Company and Home (KM)",
    "Length of Service (Years)",
]

scale = StandardScaler()
customer_scale = scale.fit_transform(encode_df[num_cols])
scale_df = encode_df.copy(deep=True)
scale_df[num_cols] = customer_scale
```

In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(18, 6), tight_layout=True)
df[num_cols].plot.box(vert=True, ax=ax[0], title="Before Scaling")
scale_df[num_cols].plot.box(vert=True, ax=ax[1], title="After Scaling")
plt.show()
```



Categorical Encoding

As the our dataset still contains categorical data which is hard for some of sklearn's model to train on, we have to encode the data using one of the encoders.

1. pd.get_dummies()
2. One Hot Encoder
3. Label Encoder
4. Ordinal Encoder

As most of the categorical columns have multiple values, we will use one hot encoder and drop if it is binary to remove additional columns like gender (Male and Female) which can be identified as if you are not male then you are female.

In [ ]:
```
encode_df
```

Out[ ]:

	Age	Distance Between Company and Home (KM)	Education (1 is lowest, 5 is highest)	Job Satisfaction (1 is lowest, 4 is highest)	Salary ($)	Performance Rating (1 is lowest, 4 is highest)	Work Life Balance (1 is worst, 4 is best)	Length of Service (Years)	Gender_Ma
0	41	1	2	4	5993	3	1	6	0
1	49	8	1	2	5130	4	3	10	1
2	37	2	2	3	2090	3	3	0	1
3	33	3	4	3	2909	3	3	8	0
4	27	2	1	2	3468	3	3	2	1
...	...	...	...	...	...	...	...	...	
1465	36	23	2	4	2571	3	3	5	1
1466	39	6	1	1	9991	3	3	7	1
1467	27	4	3	2	6142	4	3	6	1
1468	49	2	3	2	5390	3	2	9	1
1469	34	8	3	3	4404	3	4	4	1

1470 rows × 19 columns

Standardization

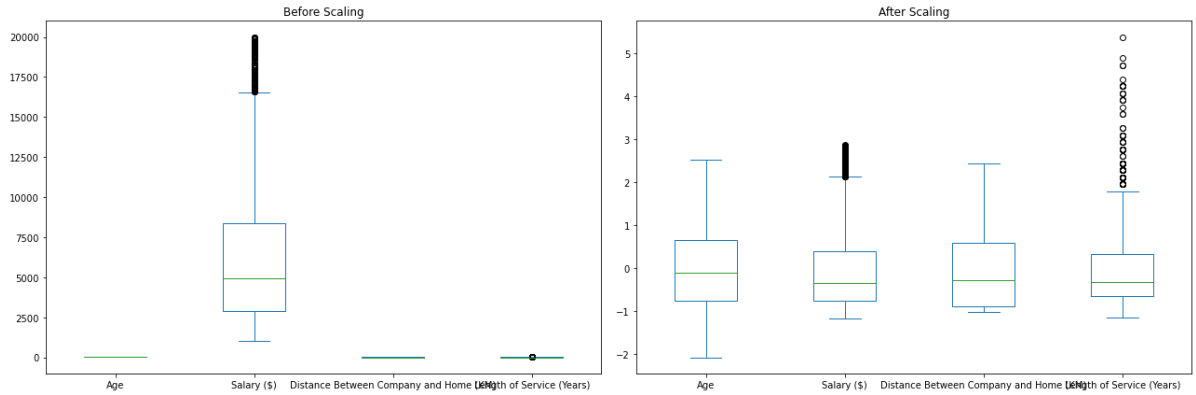
Typical Clustering Models are distance based algorithm, in order to measure similarities and form clusters using distance metrics. By standardization, there would be no statistical bias between features within the learning algorithm. Therefore, standardization is required before building a clustering model.

In this case, I would be using Z-Score Standardization to perform feature scaling. This is to ensure that all of the features would have the mean and variance, following N(0, 1).

In [ ]:
```
cat_cols = [
    "Gender",
    "BusinessTravel",
    "Job Function",
    "MaritalStatus",
    "Resign Status",
]
```

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(drop="if_binary")
```

```
ohe.fit(df[cat_cols])

encode_df = pd.concat(
    [
        df.drop(cat_cols, axis=1).reset_index(drop=True),
        pd.DataFrame(
            ohe.transform(df[cat_cols]).toarray(), columns=ohe.get_feature_names_ou
        ),
    ],
    axis=1,
)
```



In [ ]:
```
scale_df
```

Out[ ]:

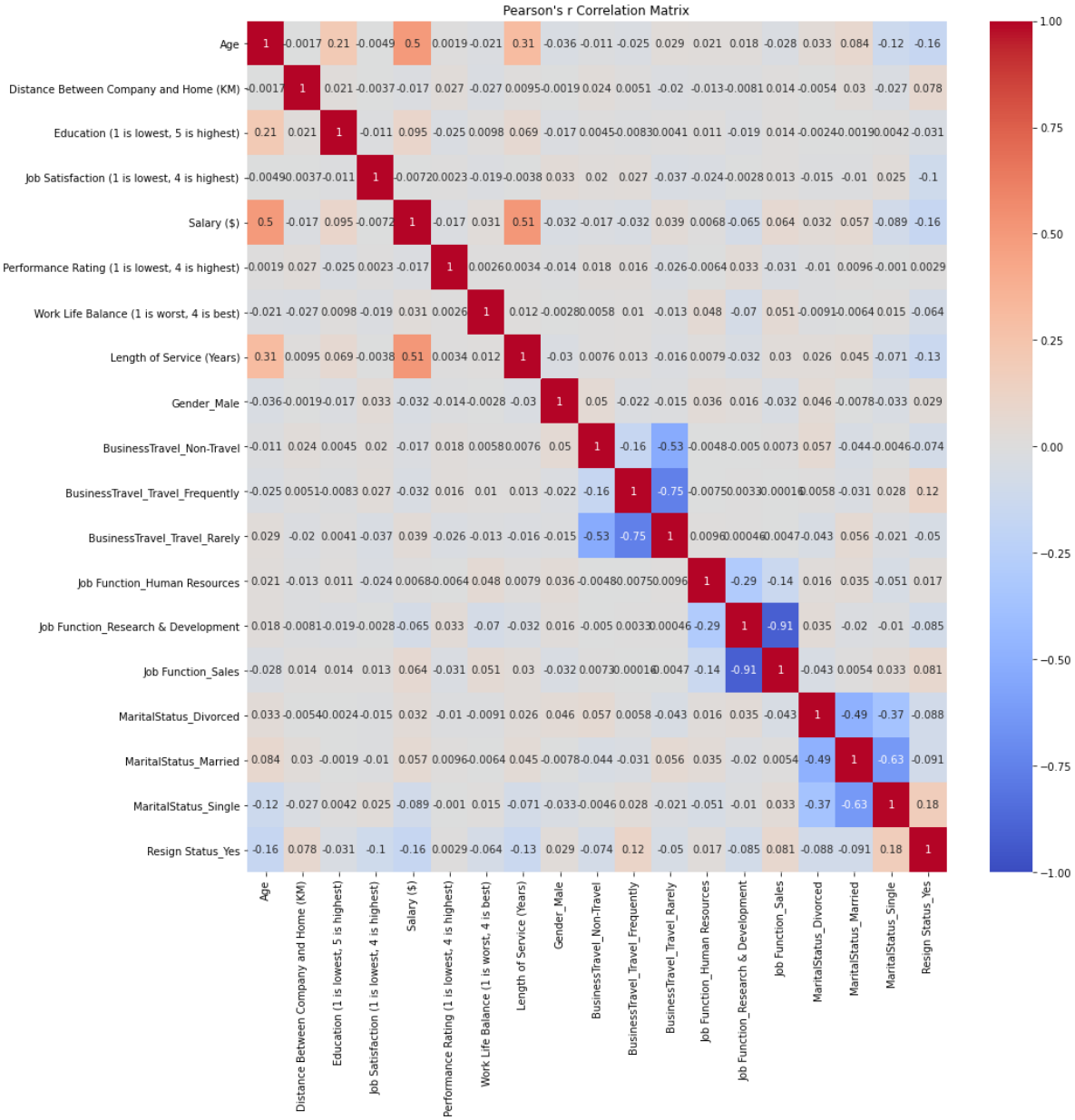
	Age	Distance Between Company and Home (KM)	Education (1 is lowest, 5 is highest)	Job Satisfaction (1 is lowest, 4 is highest)	Salary ($)	Performance Rating (1 is lowest, 4 is highest)	Work Life Balance (1 is worst, 4 is best)	Length of Service (Years)
0	0.446350	-1.010909	2	4	-0.108350	3	1	-0.164613
1	1.322365	-0.147150	1	2	-0.291719	4	3	0.488508
2	0.008343	-0.887515	2	3	-0.937654	3	3	-1.144294
3	-0.429664	-0.764121	4	3	-0.763634	3	3	0.161947
4	-1.086676	-0.887515	1	2	-0.644858	3	3	-0.817734
...	...	...	...	...	...	...	...	...
1465	-0.101159	1.703764	2	4	-0.835451	3	3	-0.327893
1466	0.227347	-0.393938	1	1	0.741140	3	3	-0.001333
1467	-1.086676	-0.640727	3	2	-0.076690	4	3	-0.164613
1468	1.322365	-0.887515	3	2	-0.236474	3	2	0.325228
1469	-0.320163	-0.147150	3	3	-0.445978	3	4	-0.491174

1470 rows × 19 columns

Pearson's r correlation - Encoded & Scaled dataframe

In [ ]:

```
fig, ax = plt.subplots(figsize=(15, 15))
# pearson correlation matrix
sns.heatmap(data=scale_df.corr(), annot=True, cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Pearson's r Correlation Matrix")
plt.show()
```



Observations

- There is no clear correlation between variables except those categorical data which we have hot encoded.
- Job Function Research & Development has a high negative correlation between Job Function Sales as 1 person can only be working at 1 department which therefore caused the strong correlation.

Clustering Tendency

Before we continue to cluster, we want to know if the dataset can be clustered well so that we can interpret the silhouette scores. We will be using Hopkins Statistics.

Hopkins Statistics

It is a hypothesis test where h0 is the data is uniformly randomly distributed.

H_0: Data is uniformly randomly distributed

H_1: Data is not uniformly randomly distributed

A value close to 1 tends to indicate the data is highly clustered, random data will tend to result in values around 0.5, and uniformly distributed data will tend to result in values close to 0.

$$H = \frac{\sum_{i=1}^{m} u_i^d}{\sum_{i=1}^{m} u_i^d + \sum_{i=1}^{m} w_i^d}$$

In [ ]:
```
from sklearn.neighbors import NearestNeighbors
import math
from random import sample
from numpy.random import uniform


def hopkins(X):
    d = X.shape[1]
    n = len(X)  # rows
    m = int(0.1 * n)  # heuristic from article [1]
    neighbour = NearestNeighbors(n_neighbors=1).fit(X.values)

    rand_X = sample(range(0, n, 1), m)

    ujd = []
    wjd = []
    for j in range(0, m):
        u_dist, _ = neighbour.kneighbors(
            uniform(np.amin(X, axis=0), np.amax(X, axis=0), d).reshape(1, -1),
            2,
            return_distance=True,
        )
        ujd.append(u_dist[0][1])
        w_dist, _ = neighbour.kneighbors(
            X.iloc[rand_X[j]].values.reshape(1, -1), 2, return_distance=True
        )
        wjd.append(w_dist[0][1])

    H = sum(ujd) / (sum(ujd) + sum(wjd))
    if math.isnan(H):
        print(ujd, wjd)
        H = 0

    return H


hopkins(scale_df)
```

Out[ ]:
```
0.6430111815128152
```

Observations

- We note that this test is run on all variables of the entire dataset and the test statistics we got is 0.645 which indicates that data has a low tendency to cluster.

Data Preparation

Before a cluster, we need to prepare the data for mutation. Even tho our Hopkins is > 0.5 and it has a lower tendency to cluster. However, the actual data is also close to 0.5 which means the clusters are also randomly displaced. The issue comes when hot encoding, additional columns are created which makes dimension reduction important. There are many methods to reduce the dimension of a dataset.

1. PCA
2. T-SNE
3. Gower Distance
4. UMAP

We will be using some of these methods to help us reduce the dataset.

Principal Component Analysis

Principal component analysis, or PCA, is a statistical procedure that allows you to summarize the information content in large data tables by means of a smaller set of "summary indices" that can be more easily visualized and analysed. In summary it reduces the number of variables in the dataset at expense oof accuracy. However with a smaller dataset, more data can be analysed faster.

However, a constraint is that all data we use PCA to must be quantitative. Even though we have hot encoded the data and it is now quantitative, applying PCA to this variables will make it difficult to reduce the data as it is completely useless.

In [ ]:
```
# Functions returns eigenvalues, % variance explained, cumulative % variance explai


def get_pca_results(data, pca):

    # Dimension indexing
    dimensions = ["PC {}".format(i) for i in range(1, len(pca.components_) + 1)]

    # PCA components
    components = pd.DataFrame(np.round(pca.components_, 4), columns=data.keys())
    components.index = dimensions
```

```
    # PCA eigenvalues
    ev = pca.explained_variance_.reshape(len(pca.components_), 1)
    eigenvalues = pd.DataFrame(np.round(ev, 4), columns=["Eigenvalue"])
    eigenvalues.index = dimensions

    # PCA explained variance
    ratios = pca.explained_variance_ratio_.reshape(len(pca.components_), 1)
    variance_ratios = pd.DataFrame(np.round(ratios, 4), columns=["Explained Varianc
    variance_ratios.index = dimensions

    # PCA cumulative variance explained
    cum_ratios = pca.explained_variance_ratio_.cumsum().reshape(len(pca.components_
    cum_variance_ratios = pd.DataFrame(
        np.round(cum_ratios, 4), columns=["Cumulative Explained Variance"]
    )
    cum_variance_ratios.index = dimensions

    # Return a concatenated DataFrame
    return pd.concat(
        [eigenvalues, variance_ratios, cum_variance_ratios, components], axis=1
    )
```

In [ ]:
```
numCols = [
    "Age",
    "Salary ($)",
    "Distance Between Company and Home (KM)",
    "Length of Service (Years)",
    "Work Life Balance (1 is worst, 4 is best)",
    "Performance Rating (1 is lowest, 4 is highest)",
    "Education (1 is lowest, 5 is highest)",
    "Job Satisfaction (1 is lowest, 4 is highest)",
]
```

In [ ]:
```
pca = PCA().fit(scale_df[numCols])
pca_results = get_pca_results(scale_df[numCols], pca)
pca_results
```

Out[ ]:

	Eigenvalue	Explained Variance	Cumulative Explained Variance	Age	Salary ($)	Distance Between Company and Home (KM)	Length of Service (Years)	Work Life Balance (1 is worst, 4 is best)	Performanc Rating (1 i lowest, 4 i highest
PC 1	1.9429	0.2817	0.2817	0.5475	0.6031	-0.0012	0.5285	0.0078	-0.002
PC 2	1.2170	0.1764	0.4581	-0.0093	-0.0195	0.0221	-0.0234	0.0203	-0.000
PC 3	1.0400	0.1508	0.6089	-0.0890	0.1926	-0.4742	0.2381	0.0213	0.002
PC 4	0.9914	0.1437	0.7527	-0.0648	0.0798	0.8780	0.1735	-0.0310	0.014
PC 5	0.6610	0.0958	0.8485	0.6993	0.0432	0.0351	-0.6474	-0.1096	0.004
PC 6	0.5007	0.0726	0.9211	0.0316	-0.1778	-0.0462	0.1791	-0.9657	0.000
PC 7	0.4144	0.0601	0.9812	0.4449	-0.7474	-0.0172	0.4267	0.2312	0.021
PC 8	0.1299	0.0188	1.0000	-0.0101	0.0163	-0.0112	-0.0082	-0.0038	0.999

A general rule of thumb when doing PCA is to use the cumulative explained variance of 75%. Thus we will be selecting PC1 to PC6.

In [ ]:
```
pca = PCA(n_components=4).fit(scale_df[numCols])
pca_select = pca.transform(scale_df[numCols])
pca_df = pd.DataFrame(pca_select, columns=["PC1", "PC2", "PC3", "PC4"])
pca_df
```

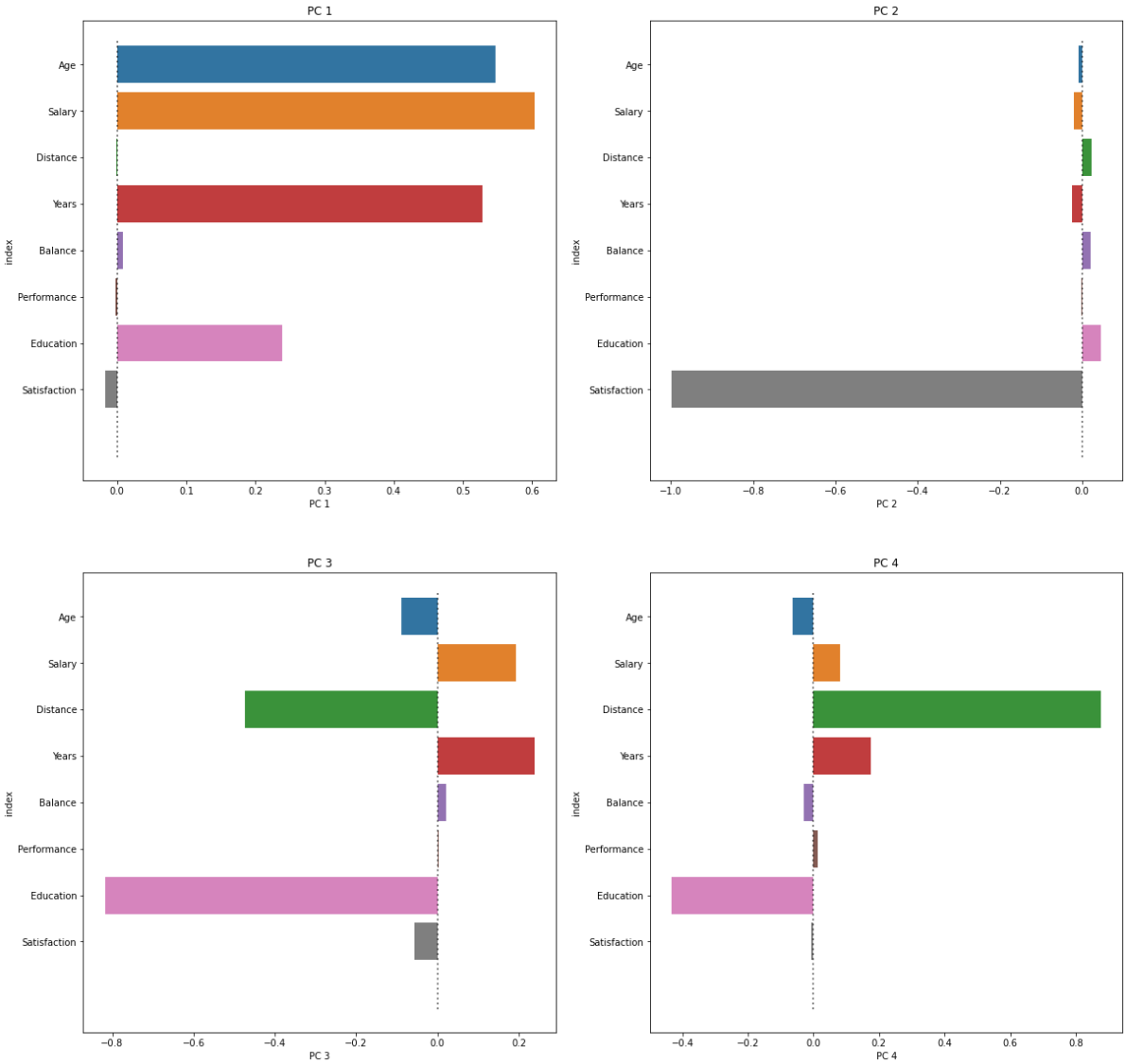
Out[ ]:

	PC1	PC2	PC3	PC4
0	-0.160571	-1.367460	1.016770	-0.514404
1	0.363194	0.620742	1.625870	0.683632
2	-1.384817	-0.283056	0.703497	-0.669379
3	-0.352827	-0.217075	-0.608245	-1.157631
4	-1.855499	0.665239	1.810218	-0.079298
...	...	...	...	...
1465	-0.972792	-1.243835	-0.358107	1.756235
1466	0.148730	1.615767	1.977384	0.527600
1467	-0.693980	0.736522	0.323347	-0.555304
1468	0.782672	0.680882	0.288662	-0.838935
1469	-0.677724	-0.221637	-0.165216	-0.309229

1470 rows × 4 columns

In [ ]:

```
fig, ax = plt.subplots(2, 2, figsize=(20, 20))
for i in range(4):
    row = pd.DataFrame(pca_results.iloc[i, 3:]).reset_index()
    sns.barplot(data=row, y="index", x=f"PC {i+1}", ax=ax[i // 2, i % 2], orient="h
    ax[i // 2, i % 2].set_yticklabels(
        [
            "Age",
            "Salary",
            "Distance",
            "Years",
            "Balance",
            "Performance",
            "Education",
            "Satisfaction",
        ]
    )
    ax[i // 2, i % 2].vlines(
        0, ymin=-0.5, ymax=8.5, linestyle=":", lw=2, color="#00000080"
    )
    ax[i // 2, i % 2].set_title(f"PC {i+1}")
```



Observations

1. In PC1 seems to represent employee information. Age, Salary ($), Length of Service (Years), Education have high positive loading while the others have negligible or low negative loading.

2. In PC2 seems to represent employee job satisfaction as Job Satisfaction has a high negative loading while the others have negligible or low positive loading.

3. In PC3 seems to represent employee job involvement. Age, Distance between company and home, Education, Job Satisfaction have high negative loading. It also show Salary and Length of Service a high positive loading

4. In PC4 seems to represent employee experience in company. Education, Age is not important in the employee's experience that is why it has a high negative loading. Salary, Distance, Years are involved in the experience in company as the longer you

work, the more salary one will get and distance travelling to work is an experience that all employee will experience

t-Distributed Stochastic Neighbour Embedding

t-Distributed Stochastic Neighbour Embedding also known as t-SNE is used to help us visualise our high dimensionality dataset into two dimensions.

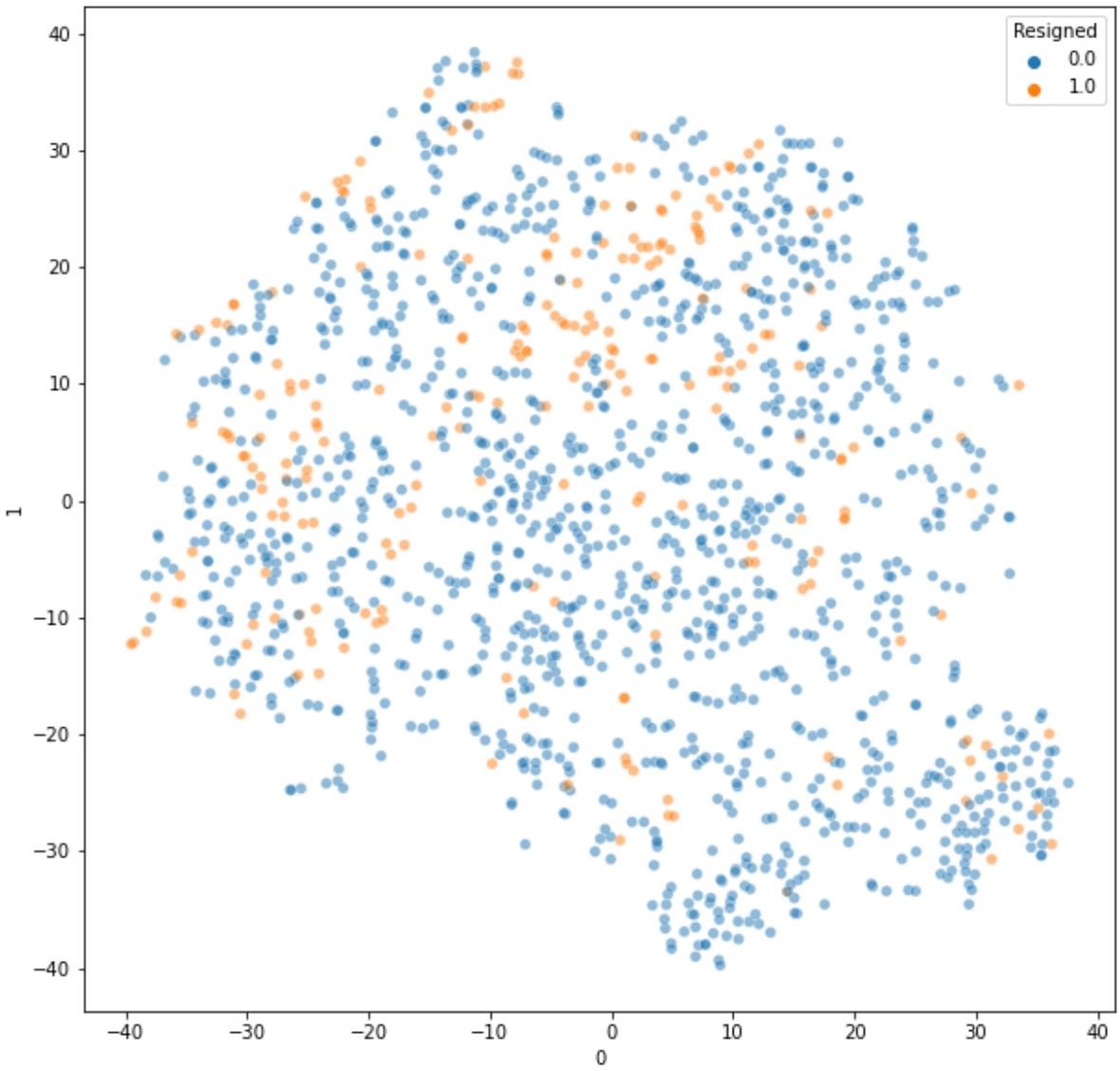
In [ ]:
```
from sklearn.manifold import TSNE

optimal_perplexity = round(np.sqrt(scale_df.shape[0]))
tsne = TSNE(learning_rate=50, perplexity=optimal_perplexity, random_state=42)
tsne_features = pd.DataFrame(tsne.fit_transform(scale_df))
tsne_features
```

Out[ ]:

	0	1
0	2.377712	0.348328
1	28.621067	10.220360
2	3.214035	12.119661
3	-0.090673	2.955988
4	-15.622397	31.196638
...	...	...
1465	-33.304409	-8.083161
1466	16.301046	30.564701
1467	14.501309	25.078405
1468	18.102413	-13.656248
1469	-10.371546	3.231286

1470 rows × 2 columns

In [ ]:
```
demo = tsne_features.copy()
demo["Resigned"] = encode_df.iloc[:, -1]
plt.figure(figsize=(10, 10))
sns.scatterplot(x=demo.iloc[:, 0], y=demo.iloc[:, 1], hue=demo.iloc[:, -1], alpha=0
plt.show()
```



Observations

- There is no clear cluster of resignation which means pure tSNE will not work to form insightful clusters for us to find out why employees are leaving the job

Gower Distance

The Gower distance is a metric that measures the dissimilarity of two items with mixed numeric and non-numeric data. Gower distance is also called Gower dissimilarity. One possible use of Gower distance is with k-means clustering with mixed data because k-means needs the numeric distance between data items.

In [ ]:
```
import gower

gower_df = gower.gower_matrix(scale_df)
```

```
gower_df = pd.DataFrame(gower_df)
gower_df
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	0.587856	0.236131	0.472332	0.430808	0.355914	0.441227	0.506953	0.493063	0.
1	0.587856	0.000000	0.394390	0.200516	0.211874	0.265928	0.259888	0.300551	0.204602	0.
2	0.236131	0.394390	0.000000	0.361794	0.207579	0.211169	0.361319	0.276797	0.296862	0.
3	0.472332	0.200516	0.361794	0.000000	0.233755	0.224188	0.266139	0.408305	0.287199	0.
4	0.430808	0.211874	0.207579	0.233755	0.000000	0.290269	0.212174	0.224015	0.386276	0.
...	...	...	...	...	...	...	...	...	...	
1465	0.514559	0.159035	0.329341	0.142729	0.210694	0.171302	0.365813	0.308859	0.215642	0.
1466	0.440968	0.209149	0.247274	0.266558	0.064758	0.329338	0.208010	0.266217	0.361690	0.
1467	0.475350	0.174735	0.276644	0.262219	0.095381	0.348808	0.145901	0.206614	0.290895	0.
1468	0.399102	0.215069	0.452692	0.236262	0.296492	0.289142	0.365809	0.512989	0.362967	0.
1469	0.428088	0.248002	0.215311	0.208655	0.086679	0.297750	0.224919	0.245532	0.334685	0.

1470 rows × 1470 columns

To help make Gower more understandable, we will make use of a combination of dimensional reduction methods like t-SNE.

In [ ]:
```
optimal_perplexity = round(np.sqrt(gower_df.shape[0]))
gower_tsne = TSNE(learning_rate=50, perplexity=optimal_perplexity, random_state=42)
gower_tsne_features = pd.DataFrame(gower_tsne.fit_transform(gower_df))
gower_tsne_features
```

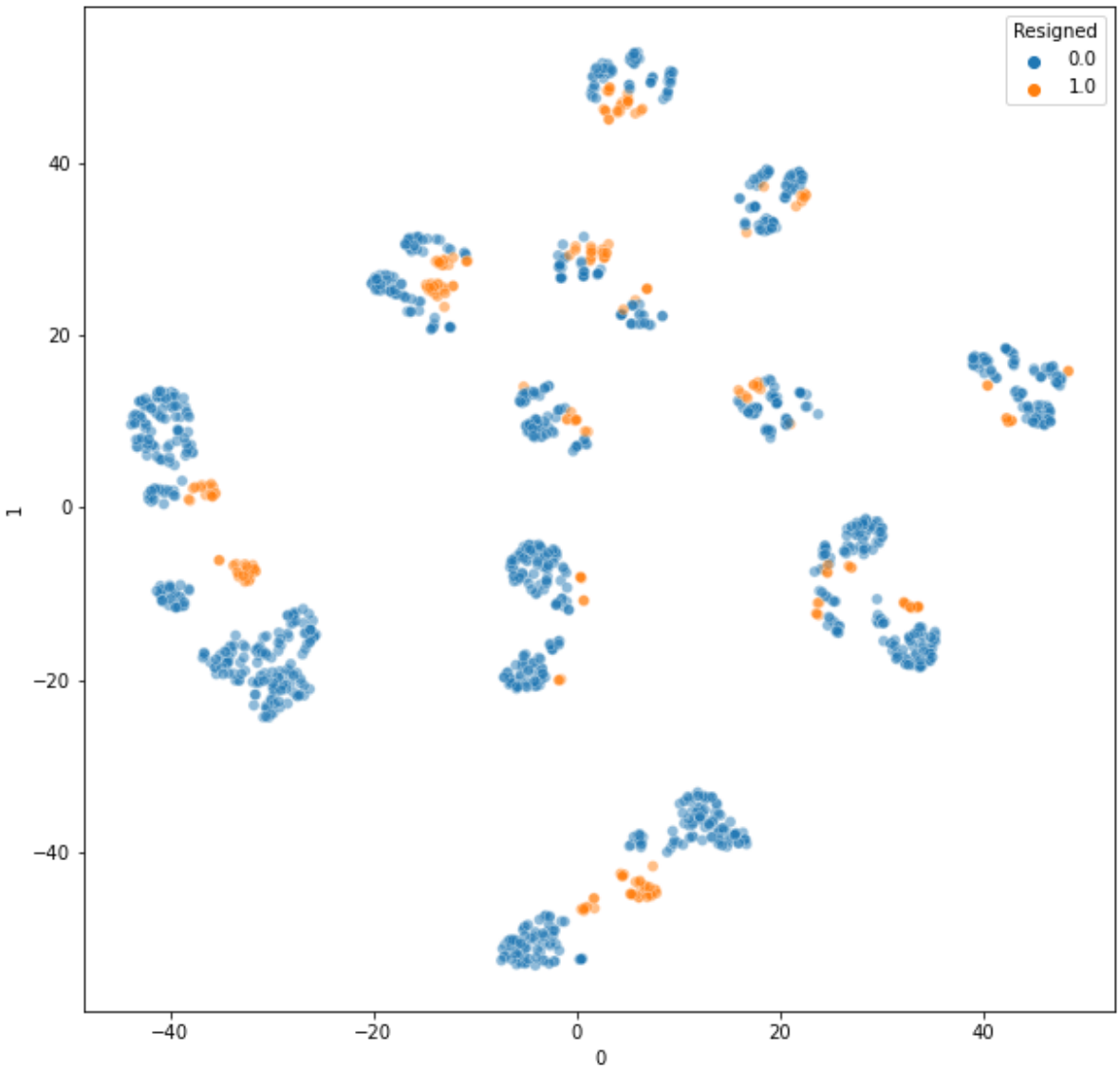
Out[ ]:

	0	1
0	-12.613850	28.187586
1	43.684853	13.263348
2	7.621294	-44.396866
3	47.522755	14.163584
4	-35.464344	-19.310467
...	...	...
1465	44.911102	9.919642
1466	-35.483398	-17.604563
1467	-39.445148	-11.565556
1468	19.676910	12.169048
1469	-30.814178	-20.115576

1470 rows × 2 columns

In [ ]:

```
demo = gower_tsne_features.copy()
demo["Resigned"] = encode_df.iloc[:, -1]
plt.figure(figsize=(10, 10))
sns.scatterplot(x=demo.iloc[:, 0], y=demo.iloc[:, 1], hue=demo.iloc[:, -1], alpha=0
plt.show()
```



Observations

After using t-SNE on the Gower Distance data there are multiple different clusters that has been formed. This seems to be a fairly robust solution with the Resign forming clusters. We will be using this dataset for the clustering models as the cluster points are quite clear

Clustering

After we are done preparing the data and testing for clustering tendency, we will proceed to perform some clustering with some of the most commonly used clustering algorithms. We will be using the following clustering algorithms:

1. K-Means Clustering (Centroid-based)
2. Agglomerative Clustering (Hierarchical-based)
3. Spectral Clustering (Distribution-based)

4. DBScan Clustering (Density-based)

K Means Clustering

K Means clustering is a simple and popular algorithm for clustering data. It is a simple algorithm that follows an iterative approach which attempts to partition the dataset into different "K" number of predefined and non-overlapping clusters.

Choosing Number of k

We will be using using 2 approach to evaluate the quality of clusters with KMeans of different k values.

1. Elbow Method using Inertia
2. Silhouette Analysis

In [ ]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples
from sklearn.metrics import silhouette_score

warnings.filterwarnings("ignore")
inertia = []
score = []
centroid = []
silhouettes_value = []

for k in range(2, 21):
    kmeans = KMeans(n_clusters=k, random_state=1).fit(gower_tsne_features)
    label = kmeans.predict(gower_tsne_features)
    inertia.append(np.sqrt(kmeans.inertia_))
    centroid.append(kmeans.cluster_centers_)
    score.append(silhouette_score(gower_tsne_features, label))
    silhouettes_value.append(silhouette_samples(gower_tsne_features, label))
```

Elbow Method (Inertia)

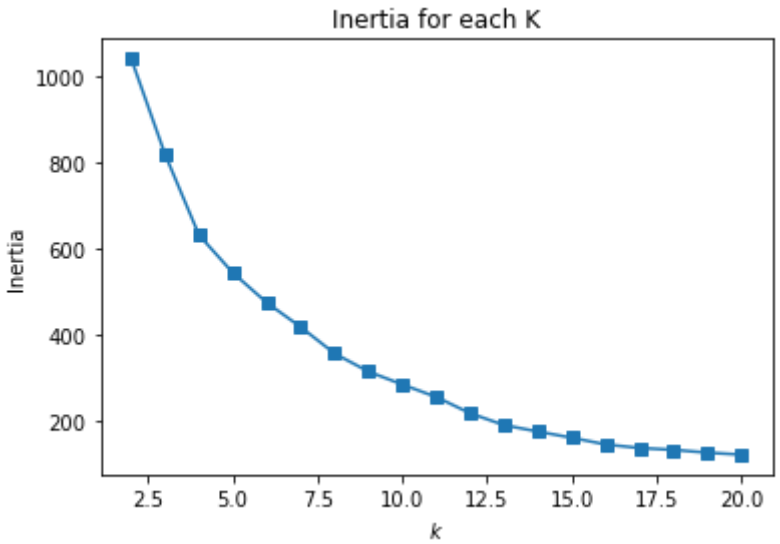
Inertia is the sum of squared distance of samples to their closest cluster center. We would like this number to be as small as possible. But, if we choose K that is equal to the number of samples we will get inertia=0.

$$\text{Inertia} = \sum^{m} *i = 1||x^{(i)} - \mu * c^{(i)}||_2^2$$

- m : Number of Datapoints
- $x^{(i)}$: i-th Datapoints
- $\mu_{c(i)}$: Cluster Centroids for i-th Datapoints

In [ ]:
```
plt.plot(range(2, 21), inertia, marker="s")
plt.title("Inertia for each K")
```

```
plt.xlabel("$k$")
plt.ylabel("Inertia")
plt.show()
```



Observations

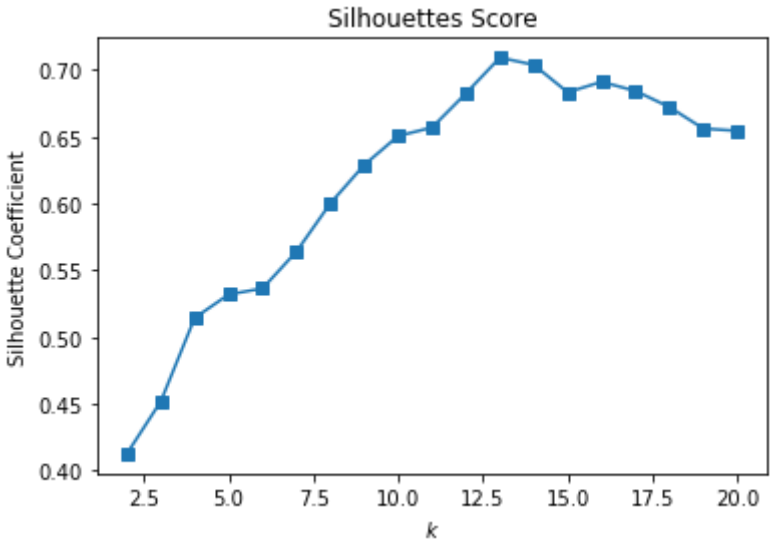
- As expected, as k increases the inertia value will decrease while picking the k value we need to consider the trade off as after a certain point, the improvement in the inertia value is not significant.
- In this case, there is no clear k value to be chosen as there is no clear elbow

Silhouette Analysis

Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighbouring clusters and thus provides a way to assess parameters like number of clusters visually. This measure has a range of [-1, 1].

$$s(o) = \frac{b(o) - a(o)}{max\{a(o), b(o)\}}$$

In [ ]:
```
plt.plot(range(2, 21), score, marker="s")
plt.title("Silhouettes Score")
plt.xlabel("$k$")
plt.ylabel("Silhouette Coefficient")
plt.show()
```



Observations

- Through the analysis of Silhouette score max out at k = 13
- Hence, k = 13 might be a good candidate for the number of clusters generated, we will be performing a more comprehensive Silhouette Analysis.
- A potential range of k could be 11 - 15

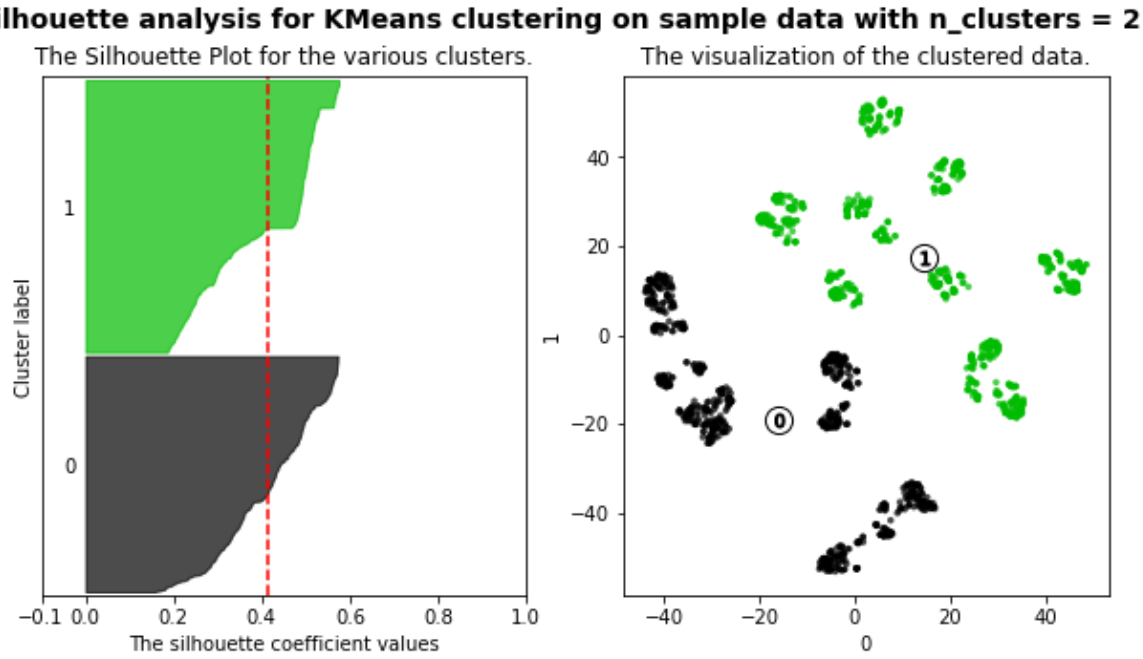
In [ ]:
```
import matplotlib.cm as cm


def silhouette_analysis(n_clusters, X, feature_1_index=0, feature_2_index=1):
    # Perform K-Mean Clustering
    kmean = KMeans(n_clusters=n_clusters, random_state=24)
    cluster_labels = kmean.fit_predict(X)
    cluster_centroids = kmean.cluster_centers_

    # Compute Individual Silhouette Score
    silhouettes_avg = silhouette_score(X, cluster_labels)
    print(
        f"For n_clusters = {n_clusters} The average silhouette score is : {silhouet
    )

    # Compute Average Silhouette Score
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(10, 5)

    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(df) + (n_clusters + 1) * 10])
    y_lower = 10

    # Assign colours for different cluster
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
```

```
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels ==
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )
        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The Silhouette Plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouettes_avg, color="red", linestyle="--")
    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(
        X.iloc[:, feature_1_index],
        X.iloc[:, feature_2_index],
        marker=".",
        s=50,
        lw=0,
        alpha=0.7,
        c=colors,
        edgecolor="k",
    )
    # Draw white circles at cluster centers
    ax2.scatter(
        cluster_centroids[:, feature_1_index],
        cluster_centroids[:, feature_2_index],
        marker="o",
        c="white",
        alpha=1,
        s=200,
        edgecolor="k",
    )

    for i, c in enumerate(cluster_centroids):
        ax2.scatter(
            c[feature_1_index],
            c[feature_2_index],
            marker="$%d$" % i,
            alpha=1,
            s=50,
            edgecolor="k",
        )
```

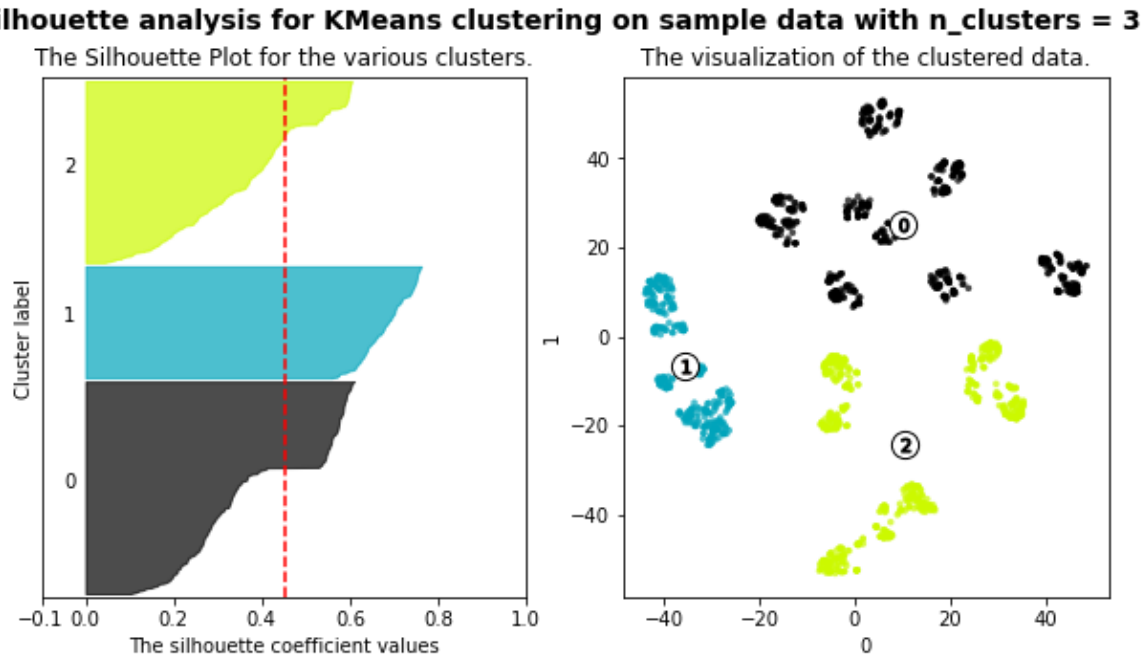
```
    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel(X.columns[feature_1_index])
    ax2.set_ylabel(X.columns[feature_2_index])
    plt.suptitle(
        "Silhouette analysis for KMeans clustering on sample data with n_clusters =
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )
```

In [ ]:
```
warnings.filterwarnings("ignore")
for i, k in enumerate(range(2, 16)):
    silhouette_analysis(n_clusters=k, X=gower_tsne_features)
    plt.show()
```

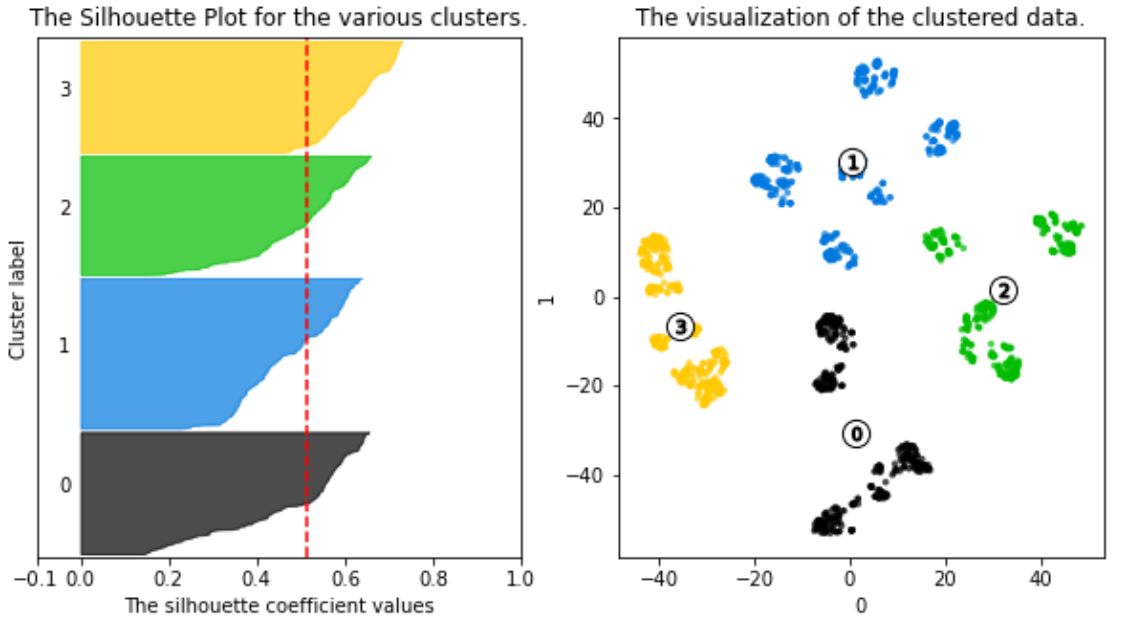
```
For n_clusters = 2 The average silhouette score is : 0.41281065344810486
```



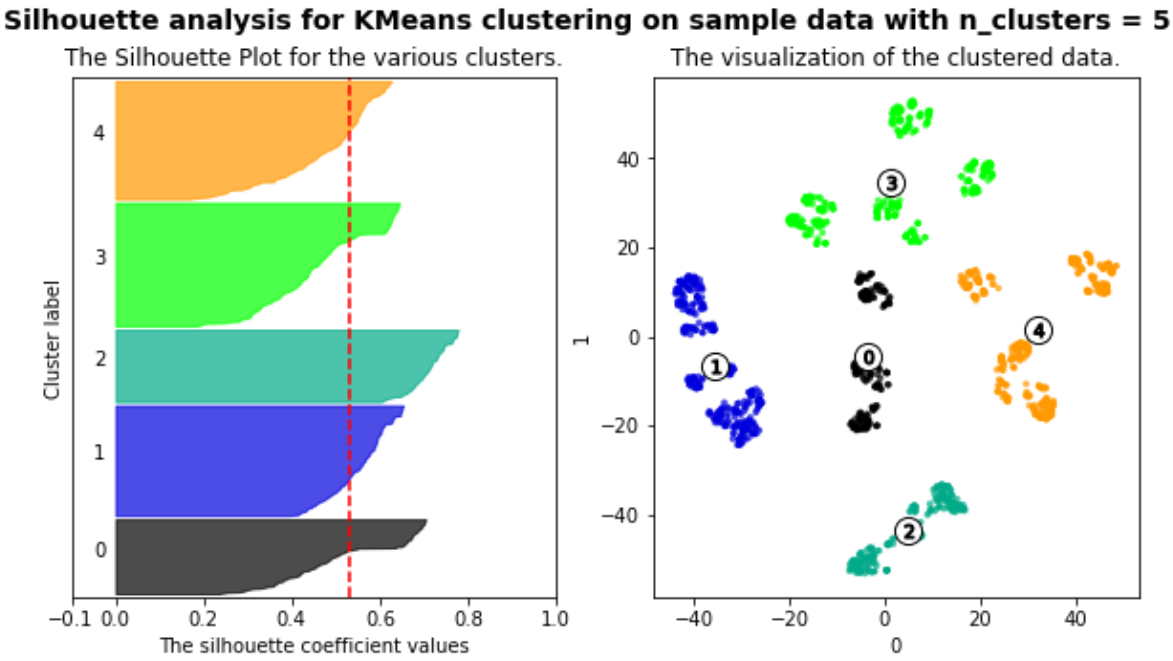

```
For n_clusters = 3 The average silhouette score is : 0.45195943117141724
```



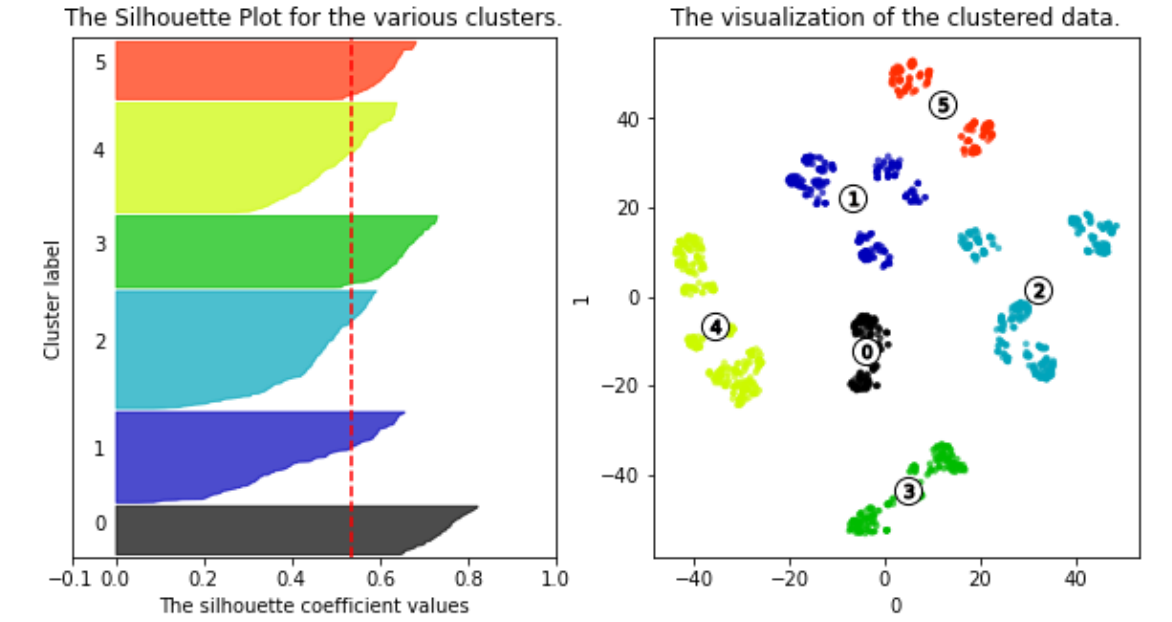
For n_clusters = 4 The average silhouette score is : 0.5144578814506531



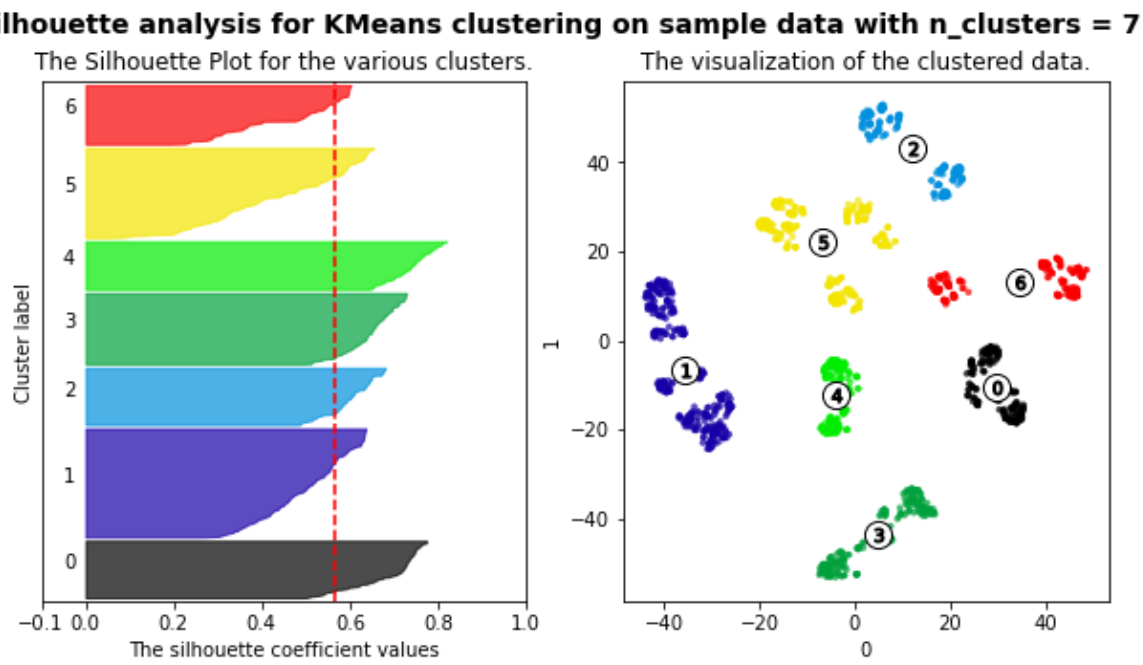
For n_clusters = 5 The average silhouette score is : 0.5319223999977112



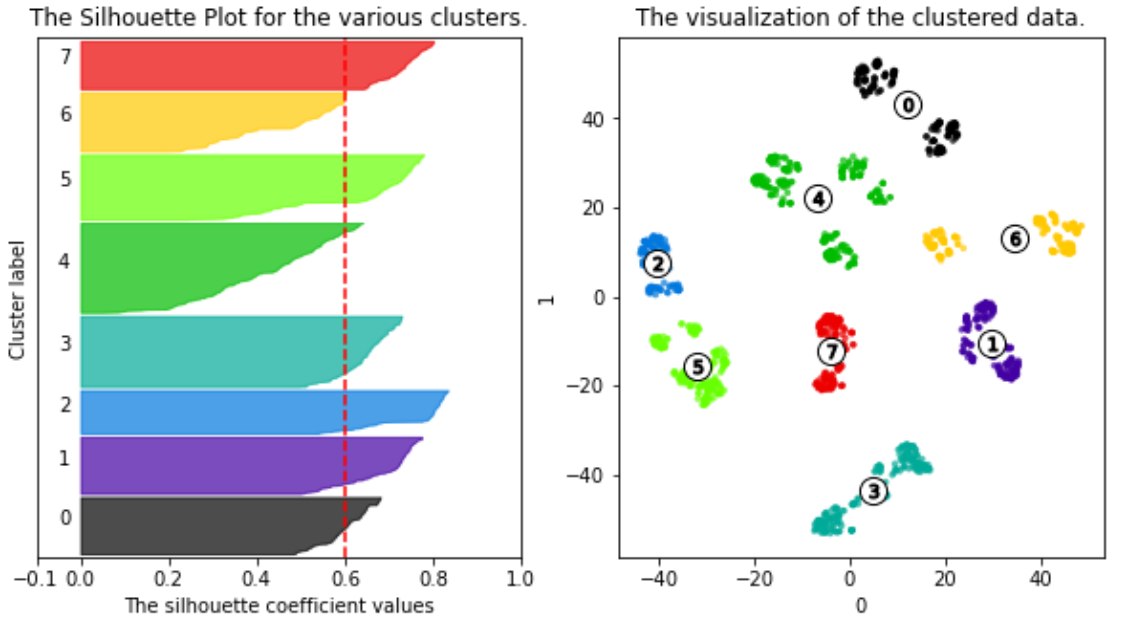
For n_clusters = 6 The average silhouette score is : 0.5364595055580139



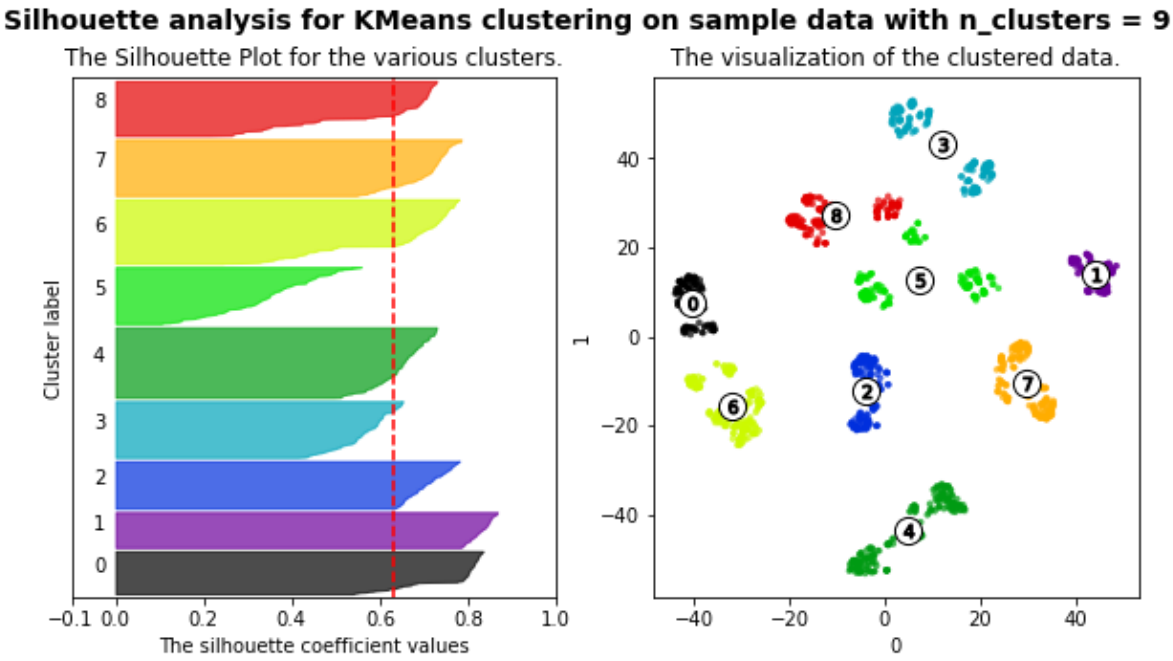
For n_clusters = 7 The average silhouette score is : 0.5640267133712769



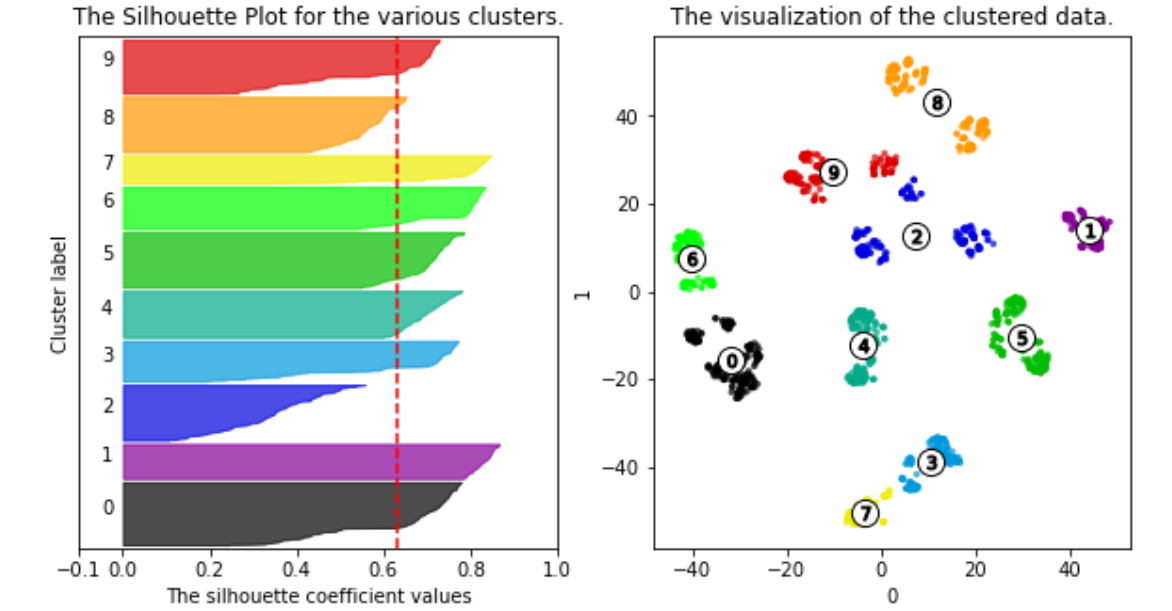
For n_clusters = 8 The average silhouette score is : 0.600136935710907



For n_clusters = 9 The average silhouette score is : 0.6289101839065552

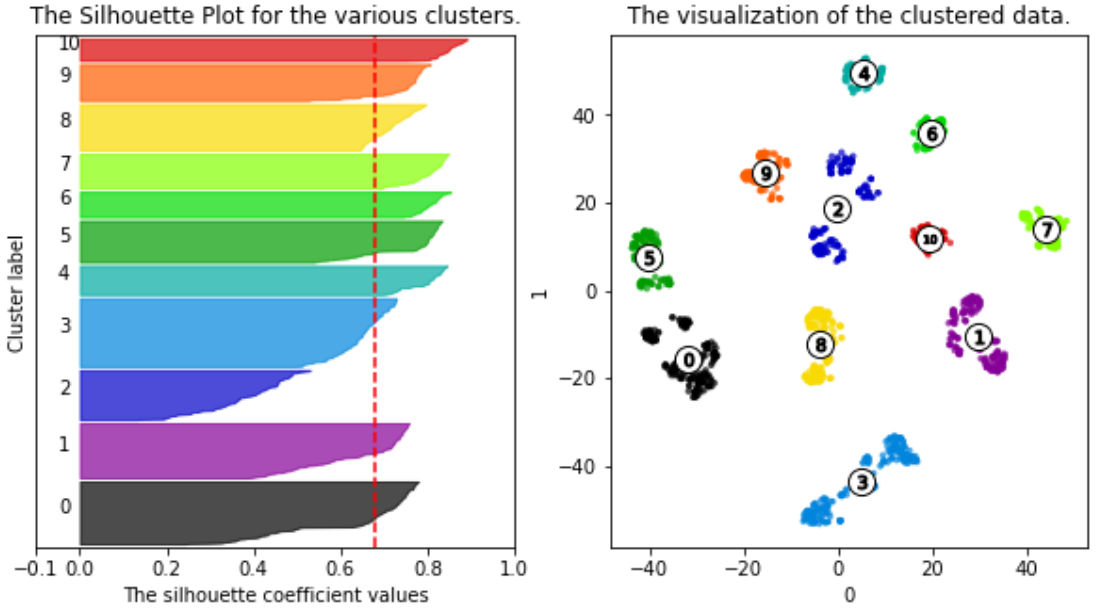


For n_clusters = 10 The average silhouette score is : 0.632358193397522



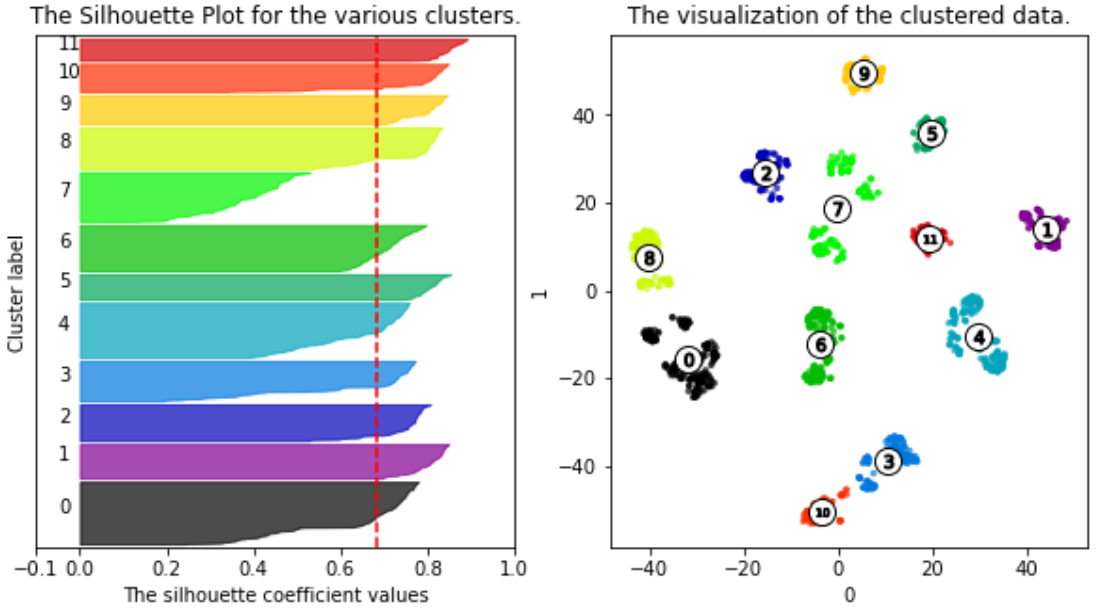
For n_clusters = 11 The average silhouette score is : 0.6787691712379456

Silhouette analysis for KMeans clustering on sample data with n_clusters = 11



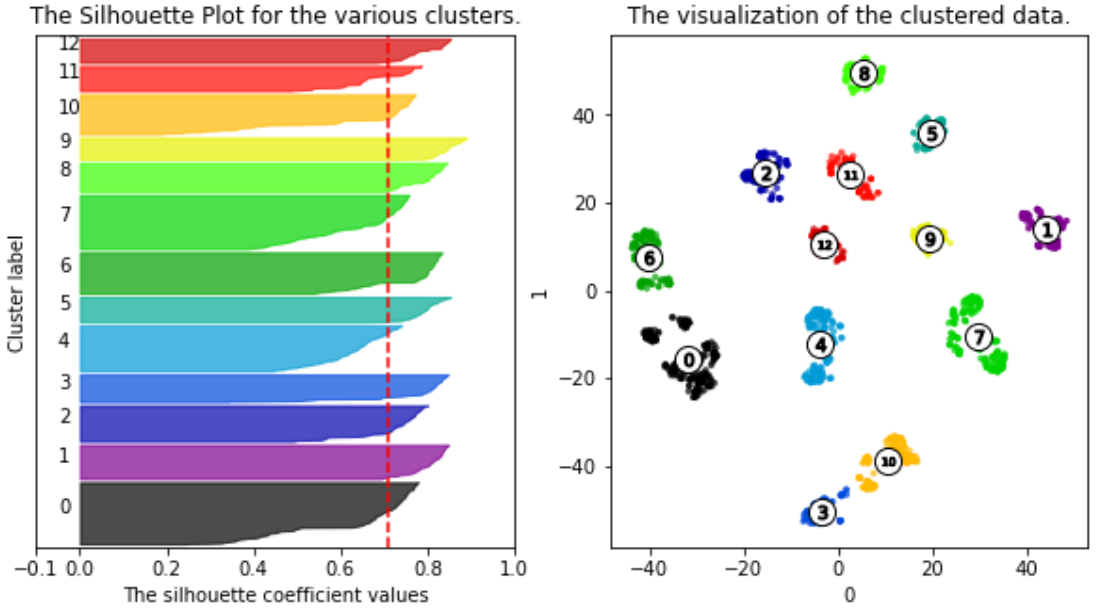
For n_clusters = 12 The average silhouette score is : 0.6824660301208496

Silhouette analysis for KMeans clustering on sample data with n_clusters = 12



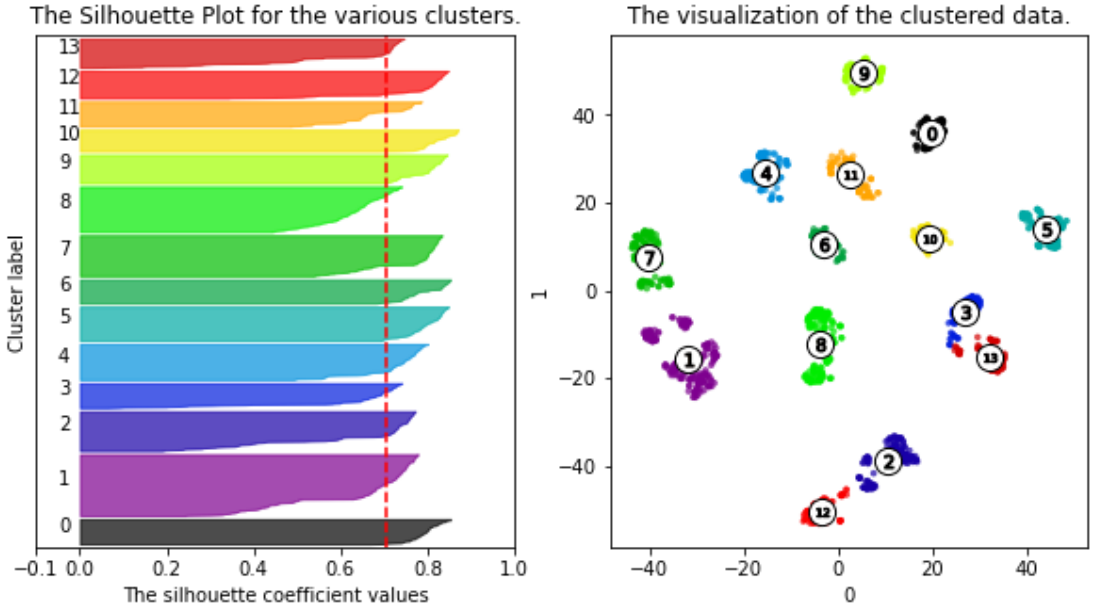
For n_clusters = 13 The average silhouette score is : 0.7092220187187195

Silhouette analysis for KMeans clustering on sample data with n_clusters = 13

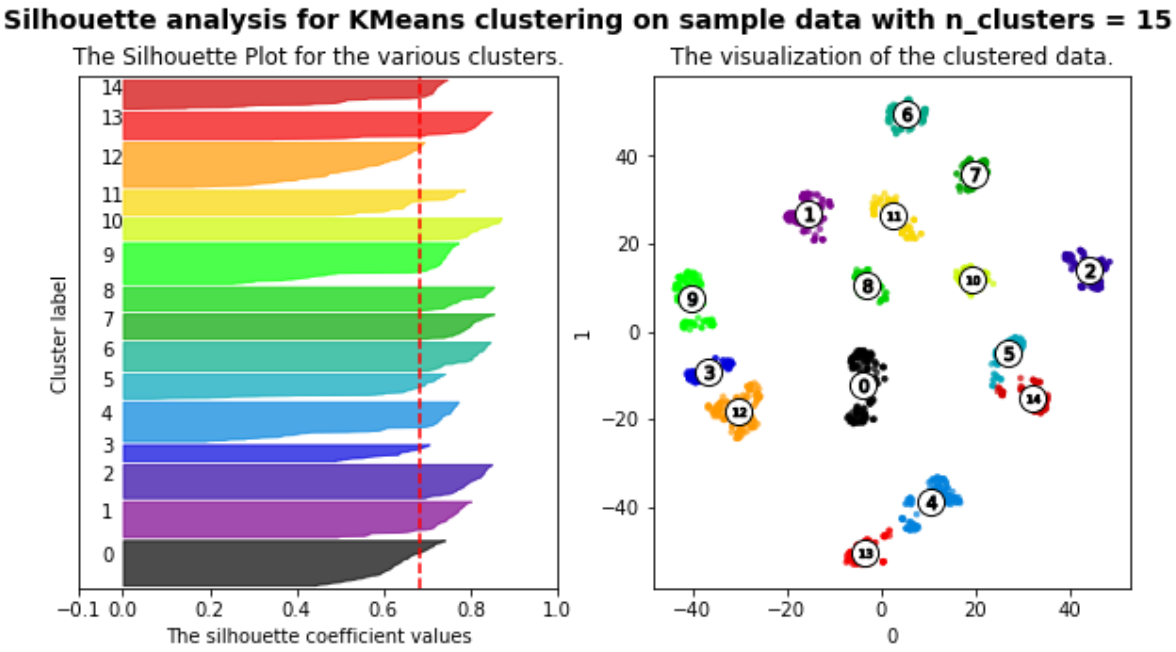


For n_clusters = 14 The average silhouette score is : 0.7038273811340332

Silhouette analysis for KMeans clustering on sample data with n_clusters = 14



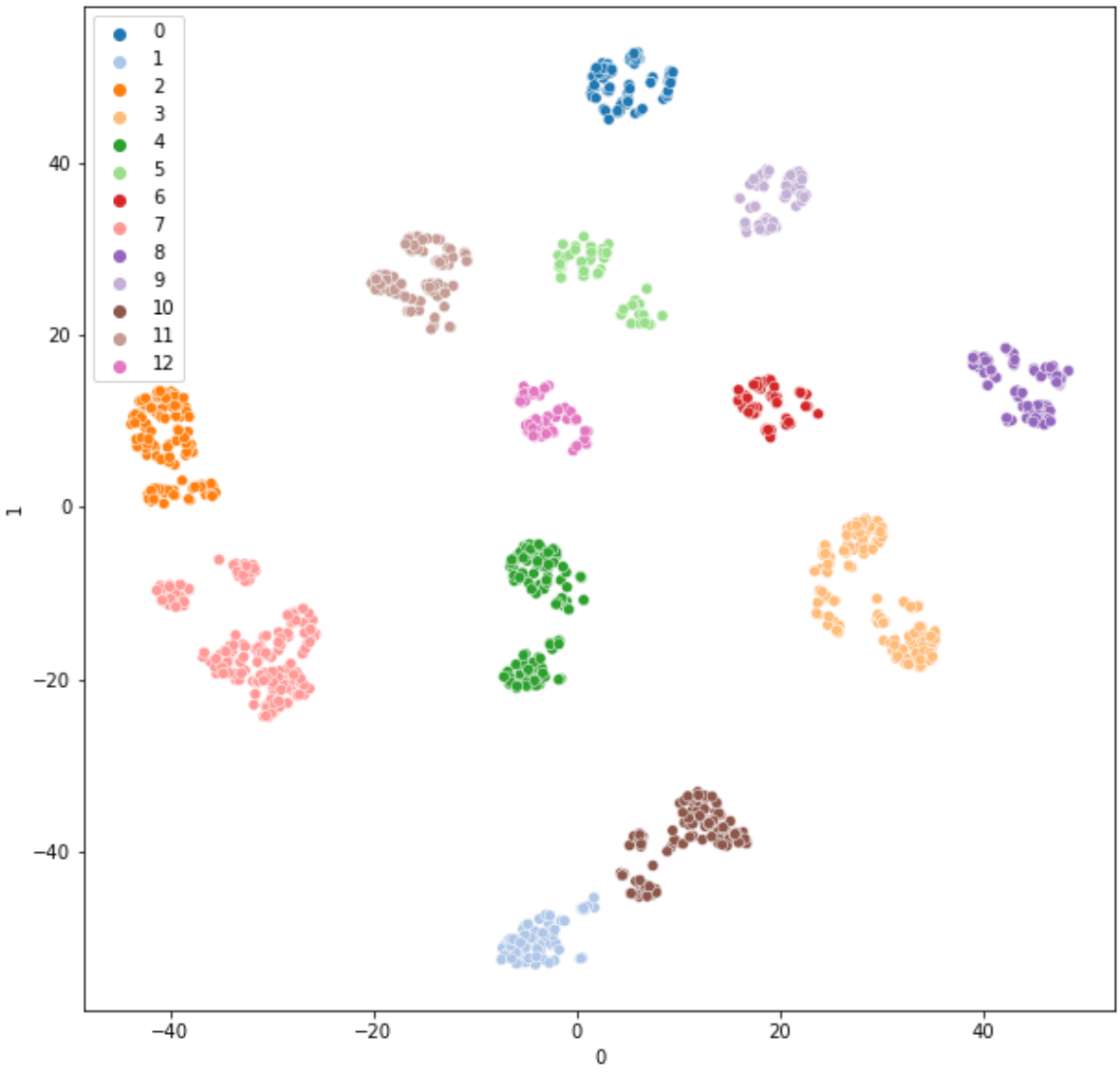
For n_clusters = 15 The average silhouette score is : 0.6832579970359802



Observations

- The best average silhouette score is 0.7092220187187195 belonging to n_clusters = 13.
- Most silhouette have cluster label coefficient values is more than the average silhouette indicated using the red line.
  - n_clusters that have cluster label coefficient values below the red line are [9,10,11,12]
- Another factor to consider if n_clusters should be used is the distribution of the cluster. If distribution is the same then it shows that the the clusters are in the same size. To find the distribution of the cluster we will be comparing the width of the cluster label.
  - n_clusters that does not have the same width are [2,5,6,7,8,11,14,15]
- Clusters that are valid = [1,3,4,13]

In [ ]:
```
kmeans = KMeans(n_clusters=13, random_state=42).fit(gower_tsne_features)
kmeans_label = kmeans.predict(gower_tsne_features)
plt.figure(figsize=(10, 10))
sns.scatterplot(
    x=gower_tsne_features.iloc[:, 0],
    y=gower_tsne_features.iloc[:, 1],
    hue=kmeans_label,
    palette="tab20",
)
plt.show()
```



In [ ]:
```
silhouettes_avg = silhouette_score(gower_tsne_features, kmeans_label)
print(silhouettes_avg)
```

```
0.709222
```

Agglomerative Clustering (Hierarchical)

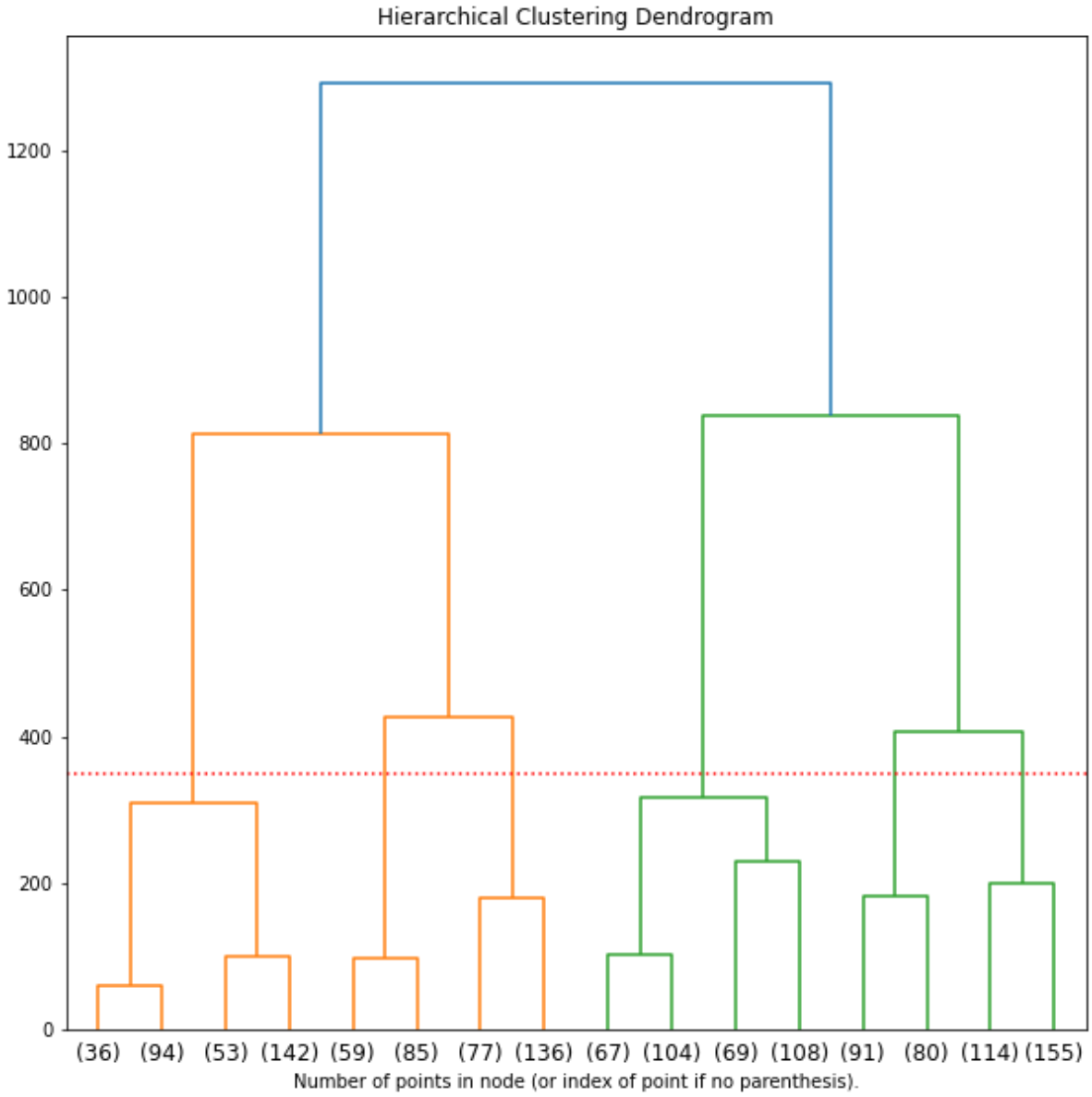
Agglomerative Clustering is a type of hierarchical clustering algorithm. It is an unsupervised machine learning technique that divides the population into several clusters such that data points in the same cluster are more similar and data points in different clusters are dissimilar.

In [ ]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram
```

```
    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

In [ ]:

```
plt.figure(figsize=(10, 10))
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)

model = model.fit(gower_tsne_features)
plt.title("Hierarchical Clustering Dendrogram")

plot_dendrogram(model, truncate_mode="level", p=3)  # Select top 3 level
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.hlines(350, 0, 300, colors="r", linestyle=":")
plt.show()
```

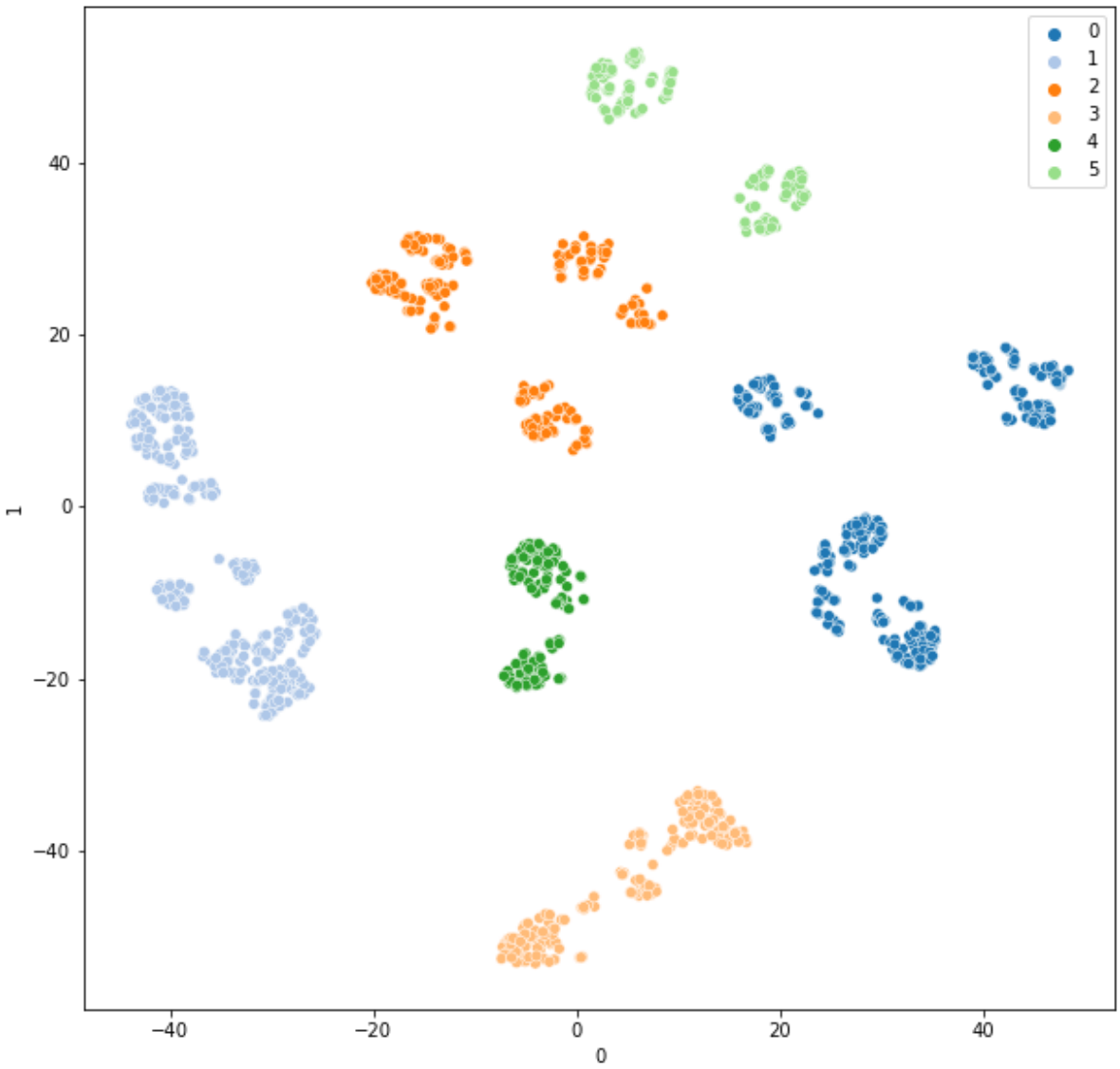


Observations

It seems that the points in the node (102)(168) and (162)(529) is very different due to the large difference in the dendrogram. The taller the edge the bigger the difference, the closeness of the data points are given by the vertical position of the split. From the dendrogram generated, the reasonable number of cluster that would be k = 6 as we look at all the vertical lines. The vertical lines with the maximum distance is the green line and hence we can decide a k = 6 to cut the dendrogram

In [ ]:
```
# Setting Number of Cluster to 6
agg_cluster_6 = AgglomerativeClustering(n_clusters=6)

agg_labels = agg_cluster_6.fit_predict(gower_tsne_features)

plt.figure(figsize=(10, 10))
sns.scatterplot(
```

```
    x=gower_tsne_features.iloc[:, 0],
    y=gower_tsne_features.iloc[:, 1],
    hue=agg_labels,
    palette="tab20",
)
plt.show()
```



In [ ]:
```
silhouette_score(gower_tsne_features, agg_labels)
```

Out[ ]:
```
0.5364595
```

Observations

By comparing the clusters formed using Agglomerative Clustering, we notice that none of the clusters overlap which is a good thing as there is no ambiguity.

We also note that the Agglomerative Clustering looks similar to the KMeans Cluster (n_clusters=6). This is also further proven as the silhouette score for KMeans (n_cluster=6)

and Agglomerative Clustering (n_cluster=6) is the same at 0.5364595

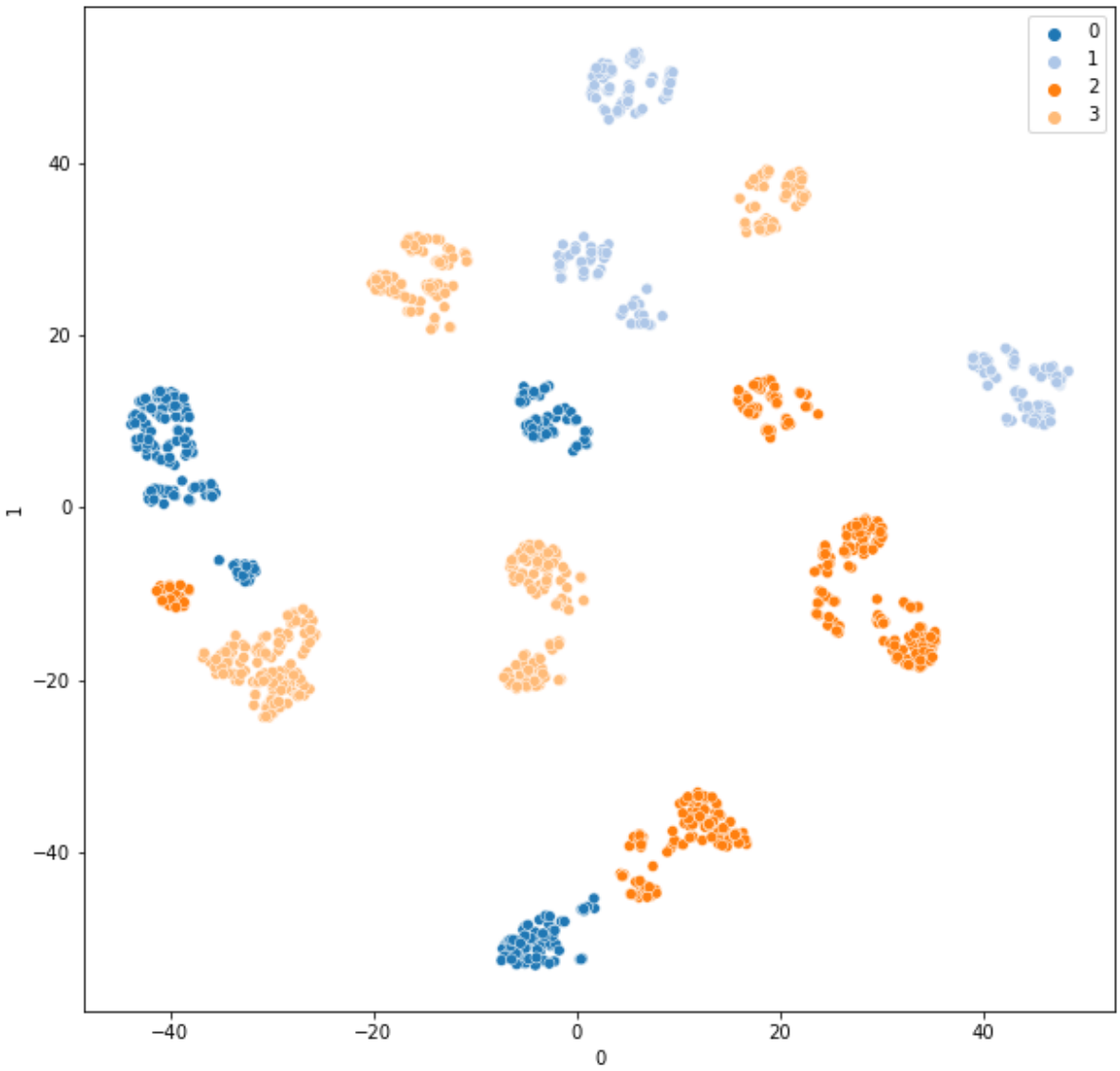
Spectral Clustering

Spectral clustering is a technique with roots in graph theory, where the approach is used to identify communities of nodes in a graph based on the edges connecting them. The method is flexible and allows us to cluster non graph data as well.

Spectral clustering uses information from the eigenvalues (spectrum) of special matrices built from the graph or the data set. We'll learn how to construct these matrices, interpret their spectrum, and use the eigenvectors to assign our data to clusters.

We will be using n_clusters of 4 as it is one of the clusters in KMeans which is valid

In [ ]:
```
from sklearn.cluster import SpectralClustering

spec_cluster_4 = SpectralClustering(
    n_clusters=4, assign_labels="discretize", random_state=0
).fit(gower_tsne_features)

spec_labels = spec_cluster_4.labels_
plt.figure(figsize=(10, 10))
sns.scatterplot(
    x=gower_tsne_features.iloc[:, 0],
    y=gower_tsne_features.iloc[:, 1],
    hue=spec_labels,
    palette="tab20",
)
plt.show()
```



In [ ]:
```
silhouette_score(gower_tsne_features, spec_labels)
```

Out[ ]:
```
0.13540813
```

Observations

The clustering of SpectralClustering has clusters in different points of the scatter plot. There are no overlaps in the clusters which means that the clusters are no ambiguous.

However, the silhouette score states that the mean clusters are indifferent or the distances between the clusters are not significant

DBSCAN

Density-based spatial clustering of applications with noise also known as DBSCAn is a data clustering algorithm. DBSCAN Groups points together that are close by based on a distance

measurement usually Euclidean distance and a minimum point. It also marks outliers as points that are in low density regions.

DBSCAN will require hyperparameter tuning as epsilon and min point to find the optimal value.

But first we will be using DBSCAN's default params of epsilon = 0.5 and min point = 5

In [ ]:
```
from sklearn.cluster import DBSCAN
```

In [ ]:
```
db = DBSCAN(n_jobs=-1).fit(gower_tsne_features)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print("Silhouette Coefficient: %0.3f" % silhouette_score(gower_tsne_features, label
```

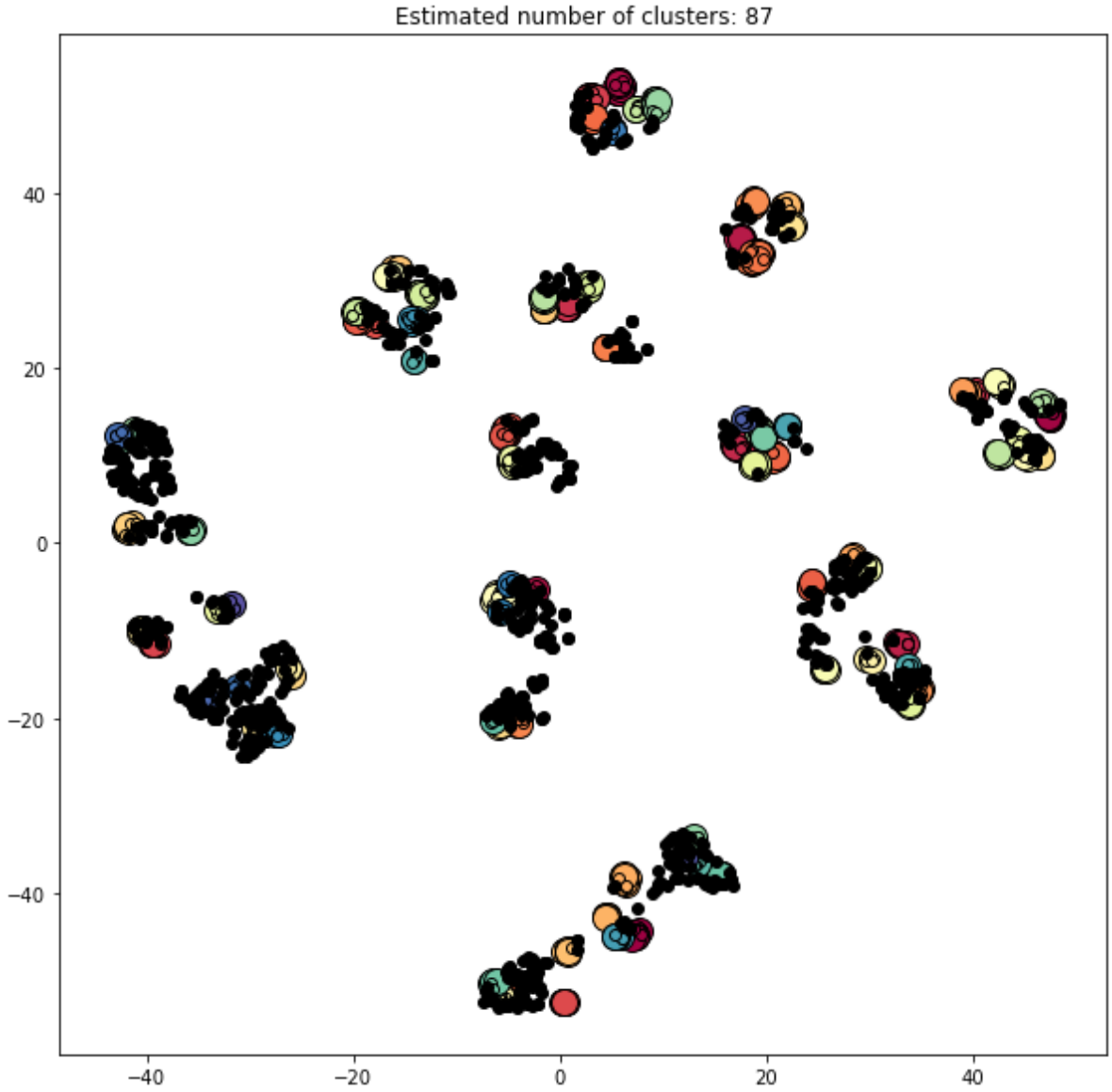
```
Silhouette Coefficient: -0.203
```

Observations

Using the silhouette score, the coefficient is -0.203 which means that the clusters are assigned in the wrong way. This is bad and we will need to change and increase the models parameters. But let us see what is wrong with the clustering.

In [ ]:
```
plt.figure(figsize=(10, 10))

# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = labels == k

    xy = gower_tsne_features[class_member_mask & core_samples_mask]
    plt.plot(
        xy.iloc[:, 0],
        xy.iloc[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )

    xy = gower_tsne_features[class_member_mask & ~core_samples_mask]
    plt.plot(
```

```
        xy.iloc[:, 0],
        xy.iloc[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.title("Estimated number of clusters: %d" % n_clusters_)
plt.show()
```



Observations

The estimated number of clusters is 87 which means that the epsilon value might be too small for the data to form meaningful clusters. We note that the min point value is making those dots that are black considered outliers and might need to be adjusted to make the clusters more meaningful

Hyperparameter tuning

We will be using custom made GridSearchCV to run through the parameters to see which model's parameter will give the best silhouette score.

Parameters that we will be tuning for the DBSCAN:

1. eps - Maximum distance between 2 samples for one to be considered as a neighbouring point
2. min_samples - Minimum number of points around the sample to be considered as a neighbouring point

In [ ]:
```
epsArr = np.arange(0.5, 4.5, 0.5)
min_samplesArr = np.arange(3, 6)

scoreArr = []

for eps in epsArr:
    for minPts in min_samplesArr:
        db = DBSCAN(eps=eps, min_samples=minPts, n_jobs=-1).fit(gower_tsne_features
        core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
        core_samples_mask[db.core_sample_indices_] = True
        labels = db.labels_

        # Number of clusters in labels, ignoring noise if present.
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
        n_noise_ = list(labels).count(-1)
        scoreArr.append(
            [str([eps, minPts]), silhouette_score(gower_tsne_features, labels)]
        )

scoreArr = (
    pd.DataFrame(scoreArr)
    .rename({0: "Params", 1: "Silhouette Score"}, axis=1)
    .sort_values(["Silhouette Score"], ascending=False)
)
scoreArr.head(20)
```

Out[ ]:

	Params	Silhouette Score
20	[3.5, 5]	0.684202
19	[3.5, 4]	0.684202
18	[3.5, 3]	0.684202
17	[3.0, 5]	0.684202
16	[3.0, 4]	0.684202
15	[3.0, 3]	0.684202
23	[4.0, 5]	0.677652
22	[4.0, 4]	0.677652
21	[4.0, 3]	0.677652
13	[2.5, 4]	0.643288
14	[2.5, 5]	0.643288
12	[2.5, 3]	0.643288
11	[2.0, 5]	0.546498
10	[2.0, 4]	0.546498
9	[2.0, 3]	0.546498
7	[1.5, 4]	0.534313
6	[1.5, 3]	0.532940
8	[1.5, 5]	0.523284
4	[1.0, 4]	0.467355
3	[1.0, 3]	0.466555

Observations

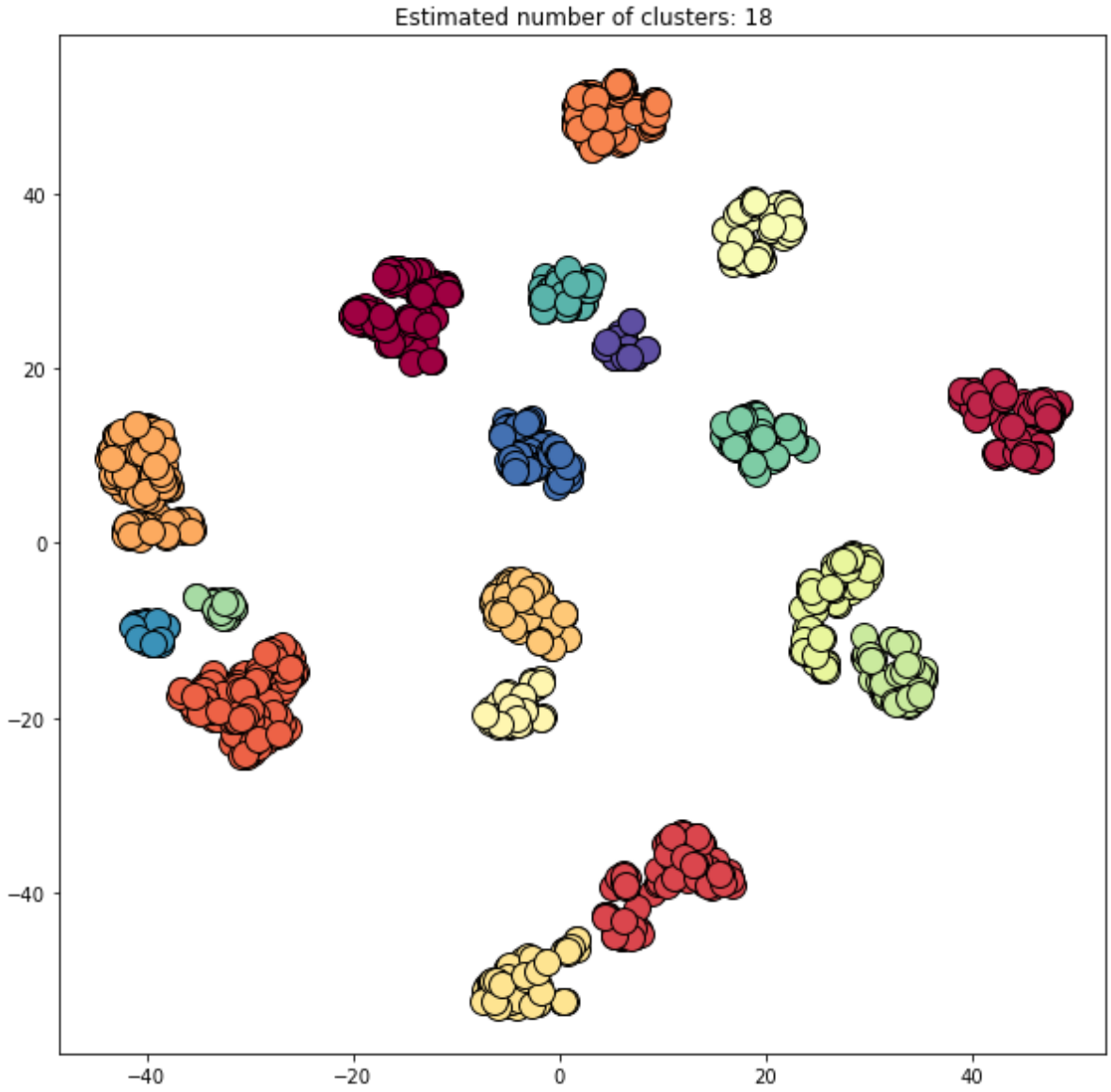
The silhouette score has a max peak which is 0.684202. As such eps = 3 and minPts = 3 will be the capping distance params that we will be using to do clustering. We can also use the eps = 4 and minPts = 4 as the silhouette score is the 2 best score.

In [ ]:

```
db = DBSCAN(eps=3, min_samples=3, n_jobs=-1).fit(gower_tsne_features)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
db_label = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(db_label)) - (1 if -1 in db_label else 0)
n_noise_ = list(db_label).count(-1)

print("Silhouette Coefficient: %0.3f" % silhouette_score(gower_tsne_features, db_la
```

```
plt.figure(figsize=(10, 10))

# Black removed and is used for noise instead.
unique_db_label = set(db_label)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_db_label))]
for k, col in zip(unique_db_label, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = db_label == k

    xy = gower_tsne_features[class_member_mask & core_samples_mask]
    plt.plot(
        xy.iloc[:, 0],
        xy.iloc[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )

    xy = gower_tsne_features[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy.iloc[:, 0],
        xy.iloc[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.title("Estimated number of clusters: %d" % n_clusters_)
plt.show()
```

```
Silhouette Coefficient: 0.684
```



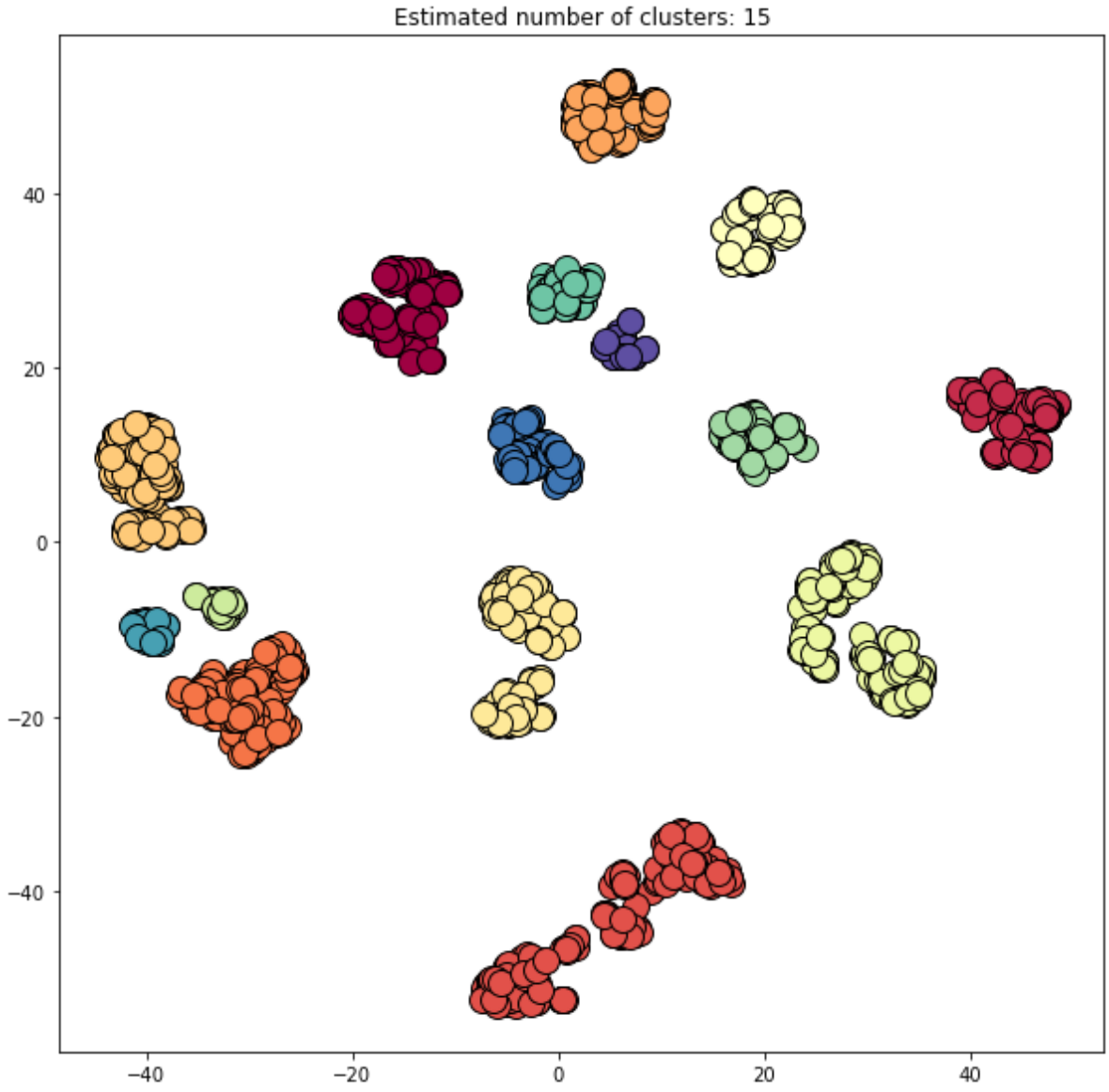
Observations

We observe that DBSCAN is able to form clusters with it's neighbouring points and the clusters are not overlapping each other (DBSCAN will just form it together as a cluster). There are also 18 different clusters that are formed, even small clusters that KMeans and other models are unable to cluster together are formed.

In [ ]:
```
db = DBSCAN(eps=4, min_samples=4, n_jobs=-1).fit(gower_tsne_features)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
db_label = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(db_label)) - (1 if -1 in db_label else 0)
n_noise_ = list(db_label).count(-1)

print("Silhouette Coefficient: %0.3f" % silhouette_score(gower_tsne_features, db_la
```

```
plt.figure(figsize=(10, 10))

# Black removed and is used for noise instead.
unique_db_label = set(db_label)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_db_label))]
for k, col in zip(unique_db_label, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = db_label == k

    xy = gower_tsne_features[class_member_mask & core_samples_mask]
    plt.plot(
        xy.iloc[:, 0],
        xy.iloc[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )

    xy = gower_tsne_features[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy.iloc[:, 0],
        xy.iloc[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.title("Estimated number of clusters: %d" % n_clusters_)
plt.show()
```

```
Silhouette Coefficient: 0.678
```



Observations

We will be using eps = 4 and minPts = 4 as it gives 15 different clusters instead of 18 like eps = 3 and minPts = 3 which makes interpreting the clusters easier

Model Selection

After looking at the different clusters formed, the models that are generally useful is KMeans and DBSCAN as Agglomerative Clustering is the same as KMeans n_clusters = 6 where the distribution based of the silhouette plot is not evenly distributed and Spectral Clustering have a low silhouette score which means that the mean of the clusters are indifferent or the distance between the clusters are insignificant because Spectral Clustering caused clusters to be in many different places in the scatter plot.

However, between DBSCAN and KMeans there is no clear distinguishing factor to differentiate the values and thus we will be see which provides a better cluster interpretation.

Cluster Interpretation

We will first do a bit of renaming of the columns of the dataframe to make our lives of interpreting the columns easier.

In [ ]:
```
cluster_df = encode_df.copy()
cluster_df.rename(
    {
        "Distance Between Company and Home (KM)": "Distance",
        "Education (1 is lowest, 5 is highest)": "Education",
        "Job Satisfaction (1 is lowest, 4 is highest)": "Satisfaction",
        "Salary ($)": "Salary",
        "Performance Rating (1 is lowest, 4 is highest)": "Performance",
        "Work Life Balance (1 is worst, 4 is best)": "Balance",
        "Length of Service (Years)": "Years",
        "Gender_Male": "Male",
        "BusinessTravel_Non-Travel": "Non-Travel",
        "BusinessTravel_Travel_Frequently": "Frequent Travel",
        "BusinessTravel_Travel_Rarely": "Rarely Travel",
        "Job Function_Human Resources": "Human Resource",
        "Job Function_Research & Development": "R&D",
        "Job Function_Sales": "Sales",
        "MaritalStatus_Divorced": "Divorced",
        "MaritalStatus_Married": "Married",
        "MaritalStatus_Single": "Single",
        "Resign Status_Yes": "Resigned",
    },
    axis=1,
    inplace=True,
)
cluster_df
```

Out[ ]:

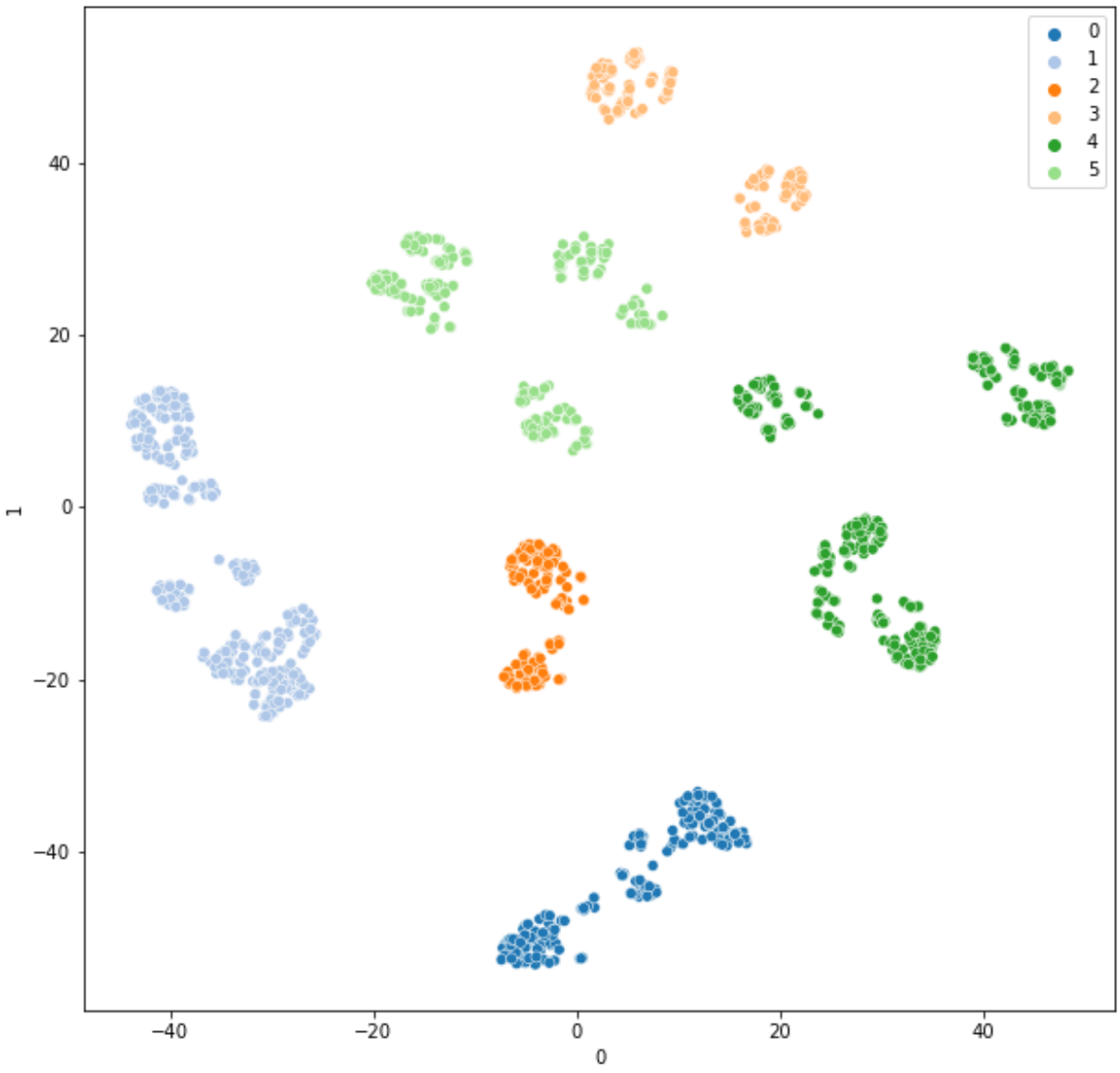
	Age	Distance	Education	Satisfaction	Salary	Performance	Balance	Years	Male	Non-Travel
0	41	1	2	4	5993	3	1	6	0.0	0.0
1	49	8	1	2	5130	4	3	10	1.0	0.0
2	37	2	2	3	2090	3	3	0	1.0	0.0
3	33	3	4	3	2909	3	3	8	0.0	0.0
4	27	2	1	2	3468	3	3	2	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...
1465	36	23	2	4	2571	3	3	5	1.0	0.0
1466	39	6	1	1	9991	3	3	7	1.0	0.0
1467	27	4	3	2	6142	4	3	6	1.0	0.0
1468	49	2	3	2	5390	3	2	9	1.0	0.0
1469	34	8	3	3	4404	3	4	4	1.0	0.0

1470 rows × 19 columns

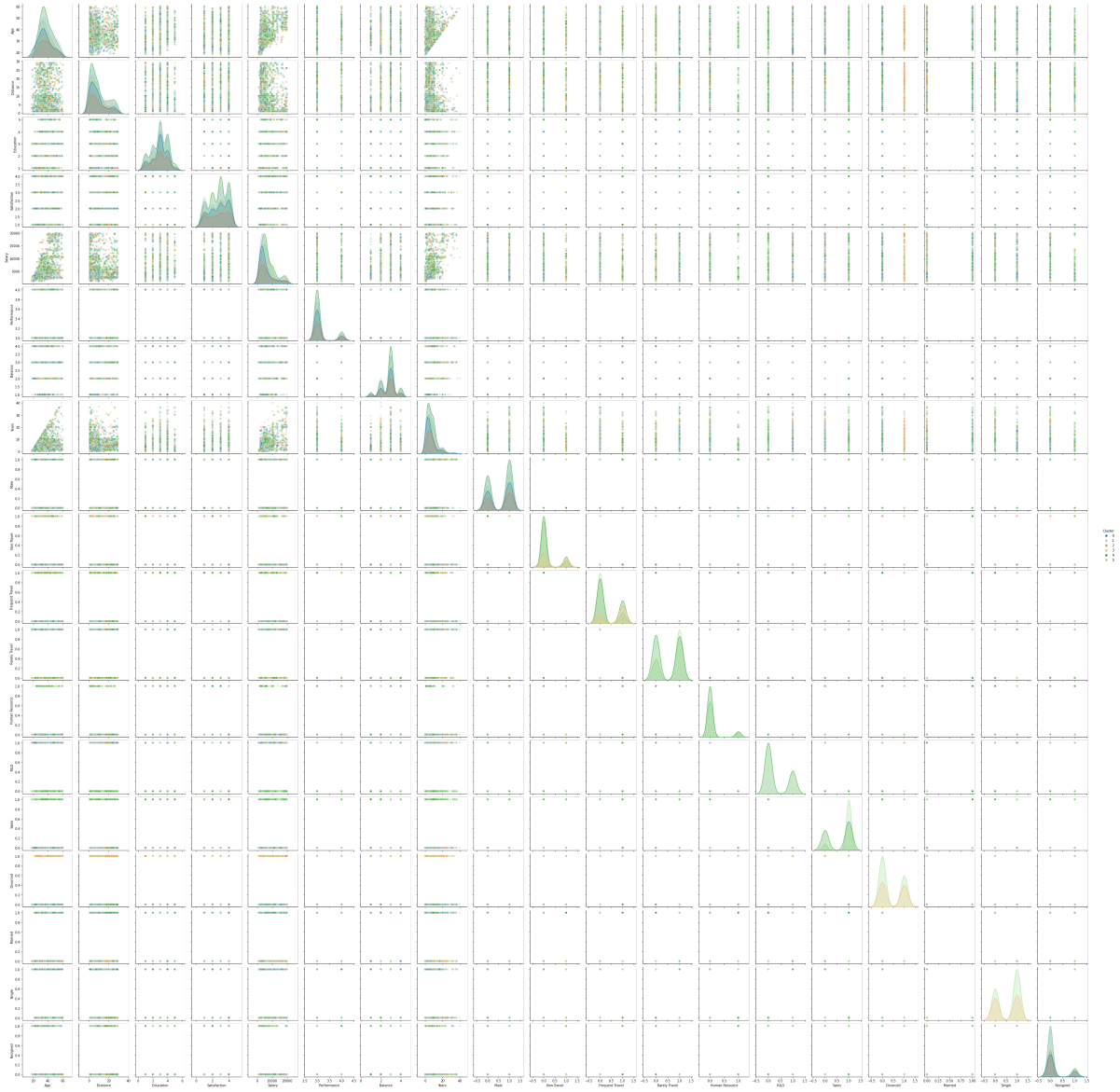
KMeans

In [ ]:

```
kmeans = KMeans(n_clusters=6, random_state=42).fit(gower_tsne_features)
kmeans_label = kmeans.predict(gower_tsne_features)
kmean_df = cluster_df.copy()
kmean_df["Cluster"] = kmeans_label
plt.figure(figsize=(10, 10))
sns.scatterplot(
    x=gower_tsne_features.iloc[:, 0],
    y=gower_tsne_features.iloc[:, 1],
    hue=kmeans_label,
    palette="tab20",
)
plt.show()
```



In [ ]:
```
sns.pairplot(kmean_df, hue="Cluster", palette="tab20", plot_kws=dict(alpha=0.4))
plt.show()
```



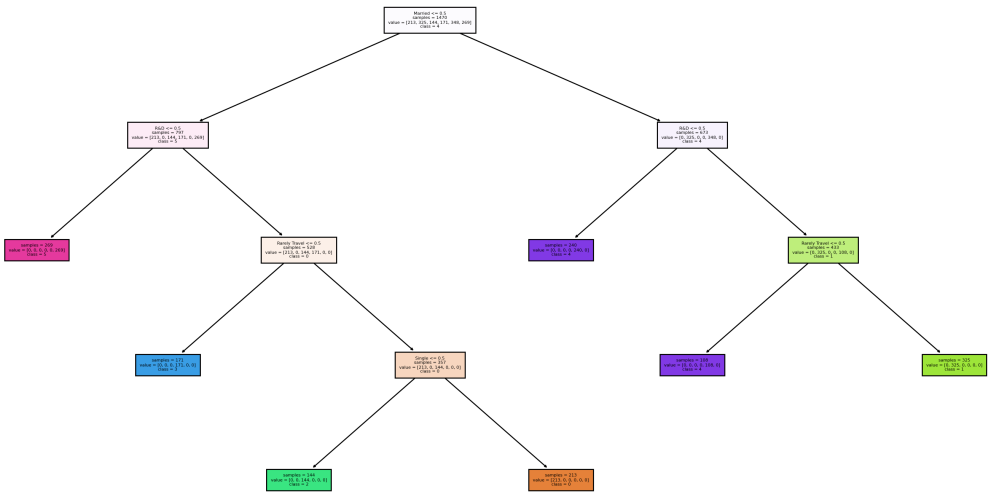
Surrogate Model: Decision Tree

To better quantify the clustering rule, a Decision Tree can be used as a surrogate model for us to better understand the decision rule by which each cluster is associated.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
```

In [ ]:
```
clf = DecisionTreeClassifier(max_depth=5, min_samples_leaf=10)

clf.fit(kmean_df.drop(columns="Cluster"), kmean_df["Cluster"])

fig, ax = plt.subplots(figsize=(20, 10), dpi=200)
plot_tree(
    clf,
    feature_names=kmean_df.drop(columns="Cluster").columns,
    class_names=np.unique(kmean_df["Cluster"].values.astype(str)),
```

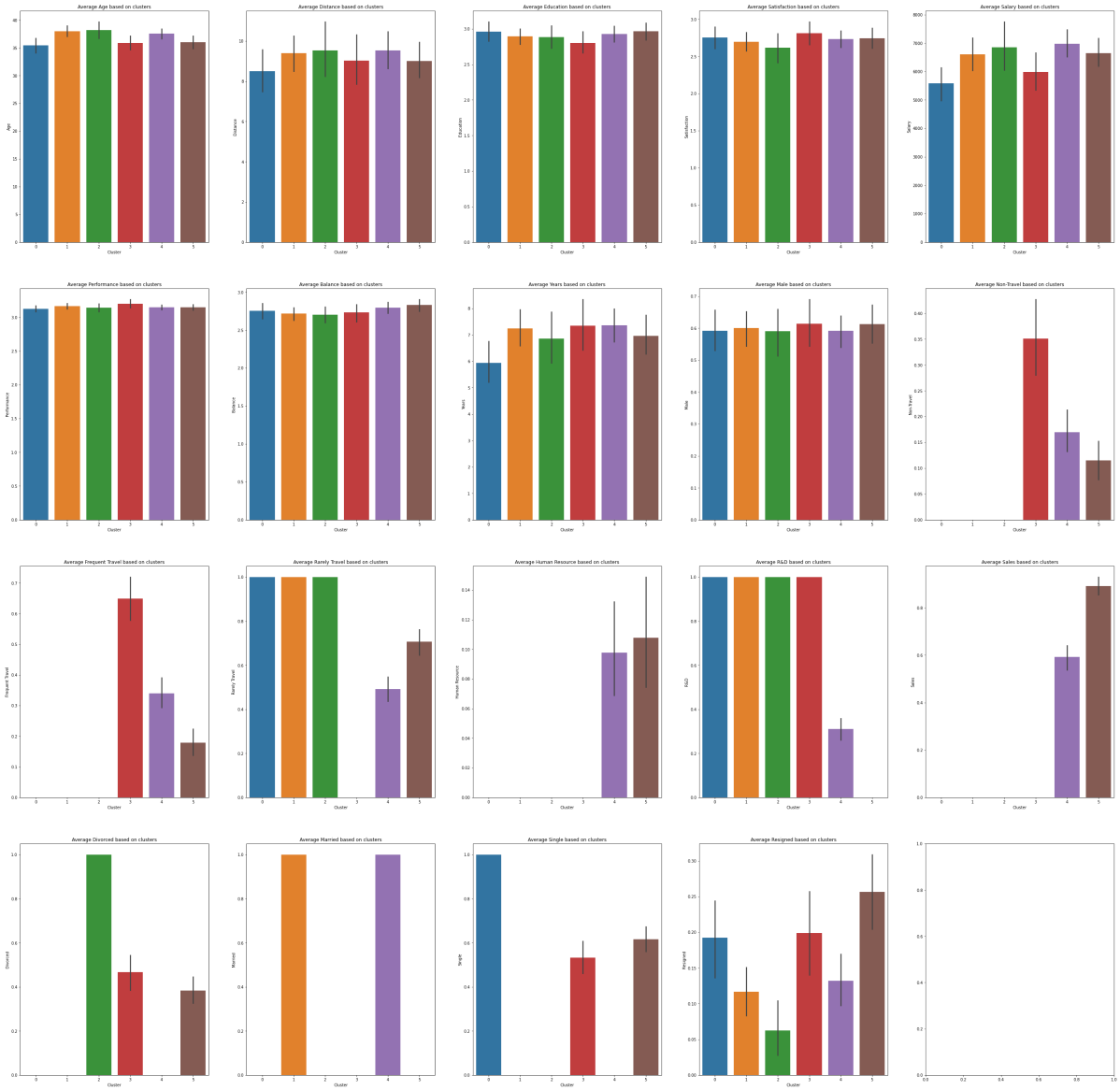
```
    impurity=False,
    filled=True,
    ax=ax,
    fontsize=4,
)
plt.show()
```



Cluster Visualisation with data

We will split the clusters and see the relationship with the variables

In [ ]:
```
fig, ax = plt.subplots(4, 5, figsize=(50, 50))
kmean_data = kmean_df.drop(columns="Cluster")
for i in range(len(kmean_data.columns)):
    sns.barplot(
        data=kmean_df, y=kmean_data.columns[i], x="Cluster", ax=ax[i // 5, i % 5]
    )
    ax[i // 5, i % 5].set_title(f"Average {kmean_data.columns[i]} based on clusters
plt.show()
```



In [ ]:

```
observation = []
for i in range(len(kmean_data.columns)):
    valueCluster = []
    for cluster in range(len(np.unique(kmeans_label))):
        currentClusterDF = kmean_df[kmean_df["Cluster"] == cluster]
        valueCluster.append(np.mean(currentClusterDF.iloc[:, i]))
    observation.append(valueCluster)

observation = pd.DataFrame(observation)
observation["variable"] = kmean_data.columns
observation.rename_axis("Cluster", axis=1, inplace=True)
observation.set_index("variable", inplace=True)
observation
```

Out[ ]:

Cluster variable	0	1	2	3	4	5
Age	35.427230	37.981538	38.215278	35.836257	37.548851	36.022305
Distance	8.502347	9.384615	9.520833	9.029240	9.528736	9.000000
Education	2.962441	2.892308	2.881944	2.801170	2.928161	2.966543
Satisfaction	2.755869	2.698462	2.618056	2.812865	2.732759	2.743494
Salary	5588.309859	6604.372308	6855.187500	5984.380117	6970.459770	6640.825279
Performance	3.126761	3.166154	3.138889	3.198830	3.149425	3.144981
Balance	2.755869	2.716923	2.701389	2.730994	2.793103	2.828996
Years	5.943662	7.249231	6.861111	7.350877	7.362069	6.962825
Male	0.591549	0.600000	0.590278	0.614035	0.591954	0.613383
Non-Travel	0.000000	0.000000	0.000000	0.350877	0.169540	0.115242
Frequent Travel	0.000000	0.000000	0.000000	0.649123	0.339080	0.178439
Rarely Travel	1.000000	1.000000	1.000000	0.000000	0.491379	0.706320
Human Resource	0.000000	0.000000	0.000000	0.000000	0.097701	0.107807
R&D	1.000000	1.000000	1.000000	1.000000	0.310345	0.000000
Sales	0.000000	0.000000	0.000000	0.000000	0.591954	0.892193
Divorced	0.000000	0.000000	1.000000	0.467836	0.000000	0.382900
Married	0.000000	1.000000	0.000000	0.000000	1.000000	0.000000
Single	1.000000	0.000000	0.000000	0.532164	0.000000	0.617100
Resigned	0.192488	0.116923	0.062500	0.198830	0.132184	0.256506

Cluster Interpretation - KMeans

Cluster 0: Young Single R&D Employees

Single R&D Employees that Rarely Travel and is aged around 35 and joined the company for around 5 years are likely to quit the job. This might but likely due to them being younger and wants to explore their own love life and form families before they hit their 40s.

Cluster 1: Middle Aged Married R&D Employees

Married R&D Employees that rarely travels and age is close to 40. They have stayed in the company for 7 years. These employees are unlikely to quit as they have a family and is settling down as a middle man.

Cluster 2: Rich Middle Aged Divorced R&D Employees
Divorced R&D Employees that rarely travels and age is close to 40. They have also say in the company for 7 years and earn approximately $6900. These employees are unlikely to quit as they have lost their partner and would want to focus on their work instead of their love life as they are too old for it and earn a lot of money.

Cluster 3: Male R&D Employees climbing corporate ladder
Majority of the Male R&D Employees that majority travel frequently and are single or divorce are not ready to quit as they are young (aged around 35) are unlikely to Resign as they want to climb the corporate ladder and focus on their own personal life like travel frequently.

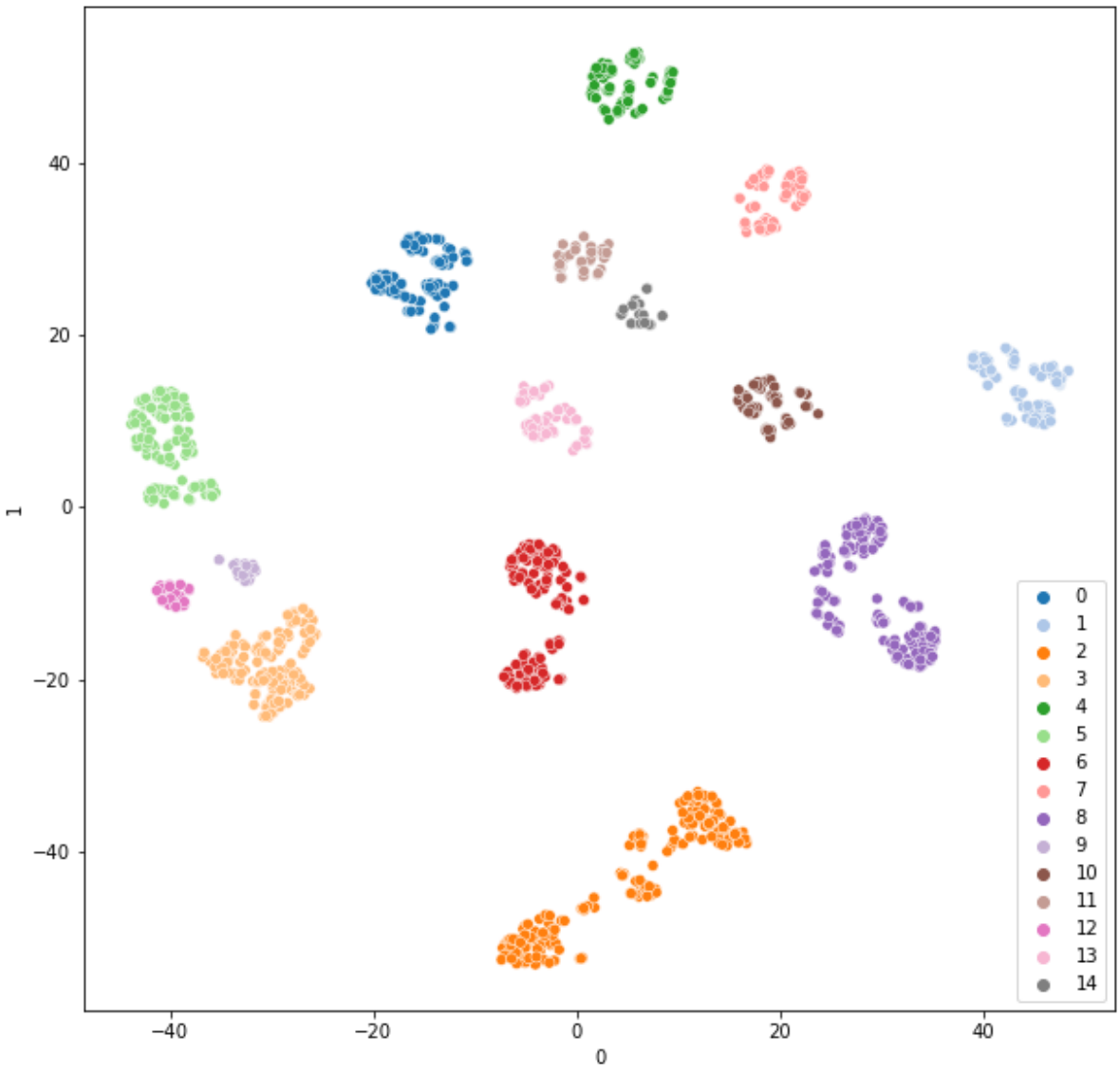
Cluster 4: Rich Married Employees
Married Employees that earns around $6900 and stays 9.5 km away from the company like to maintain at their job. This is likely due to the fact that they earn the most in the company and have a family to feed.

Cluster 5: Single Male Sales Employee
Majority of Single, Male and from the Sales department that rarely travels and live 9 km away from the company are likely to quit as they are quite young and want to work on their love life and work on their own self

DBSCAN

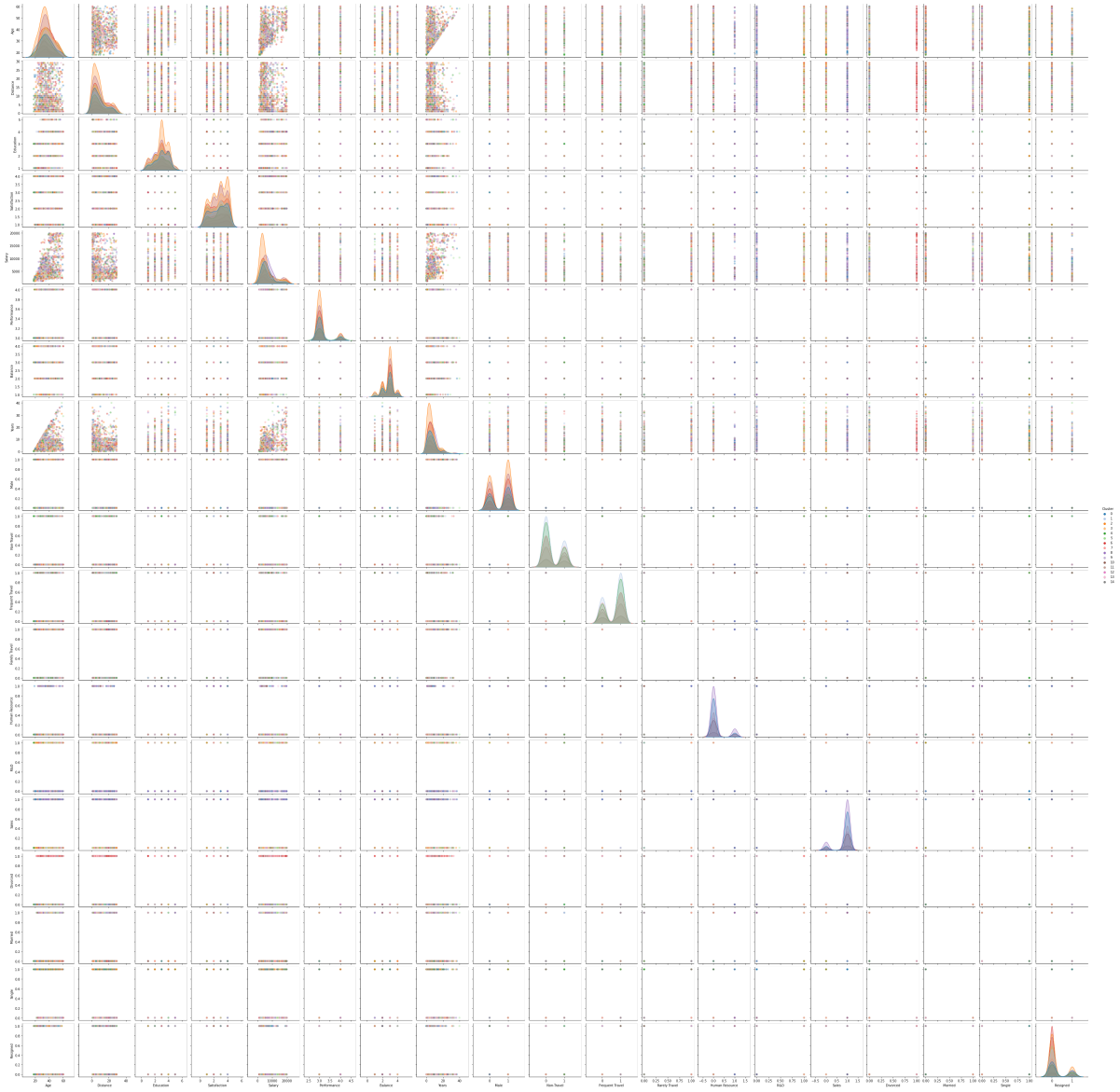
We will be using the DBSCAN Algorithm of eps = 4 and minPts = 4 as the silhouette score is 0.67765224, it give us a total of 15 clusters. Compared

In [ ]:
```
db = DBSCAN(eps=4, min_samples=4, n_jobs=-1).fit(gower_tsne_features)
db_label = db.labels_
db_df = cluster_df.copy()
db_df["Cluster"] = db_label
plt.figure(figsize=(10, 10))
sns.scatterplot(
    x=gower_tsne_features.iloc[:, 0],
    y=gower_tsne_features.iloc[:, 1],
    hue=db_label,
    palette="tab20",
)
plt.show()
silhouette_score(gower_tsne_features, db_label)
```



Out[ ]: 0.67765224

In [ ]:
```
sns.pairplot(db_df, hue="Cluster", palette="tab20", plot_kws=dict(alpha=0.4))
plt.show()
```

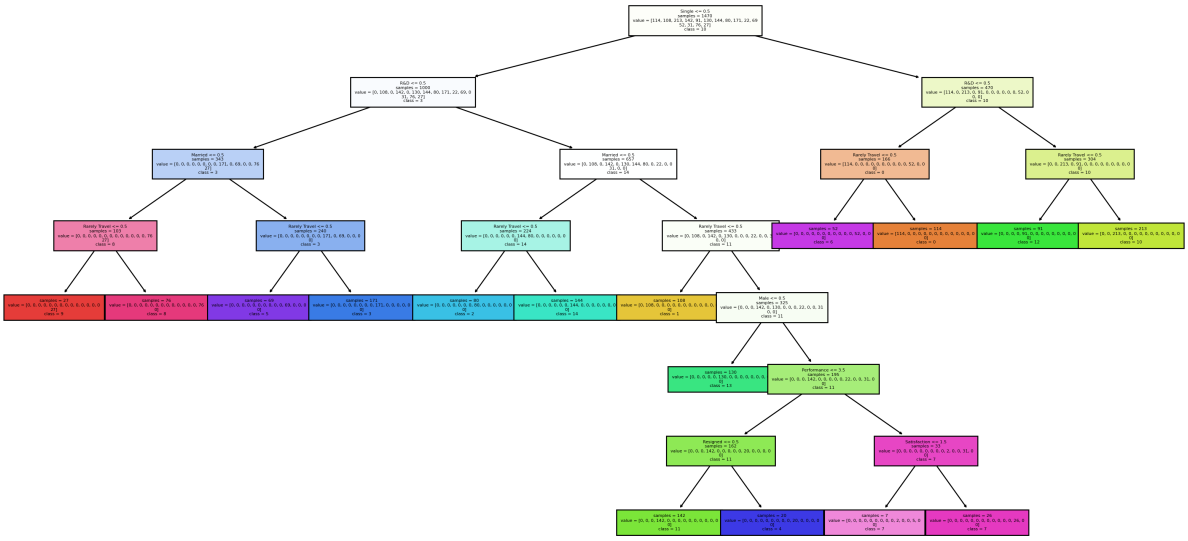


Surrogate Model: Decision Tree

To better quantify the clustering rule, a Decision Tree can be used as a surrogate model for us to better understand the decision rule by which each cluster is associated.

In [ ]:
```
clf = DecisionTreeClassifier(max_depth=None, min_samples_leaf=5)

clf.fit(db_df.drop(columns="Cluster"), db_df["Cluster"])

fig, ax = plt.subplots(figsize=(20, 10), dpi=200)
plot_tree(
    clf,
    feature_names=db_df.drop(columns="Cluster").columns,
    class_names=np.unique(db_df["Cluster"].values.astype(str)),
    impurity=False,
    filled=True,
    ax=ax,
    fontsize=4,
```

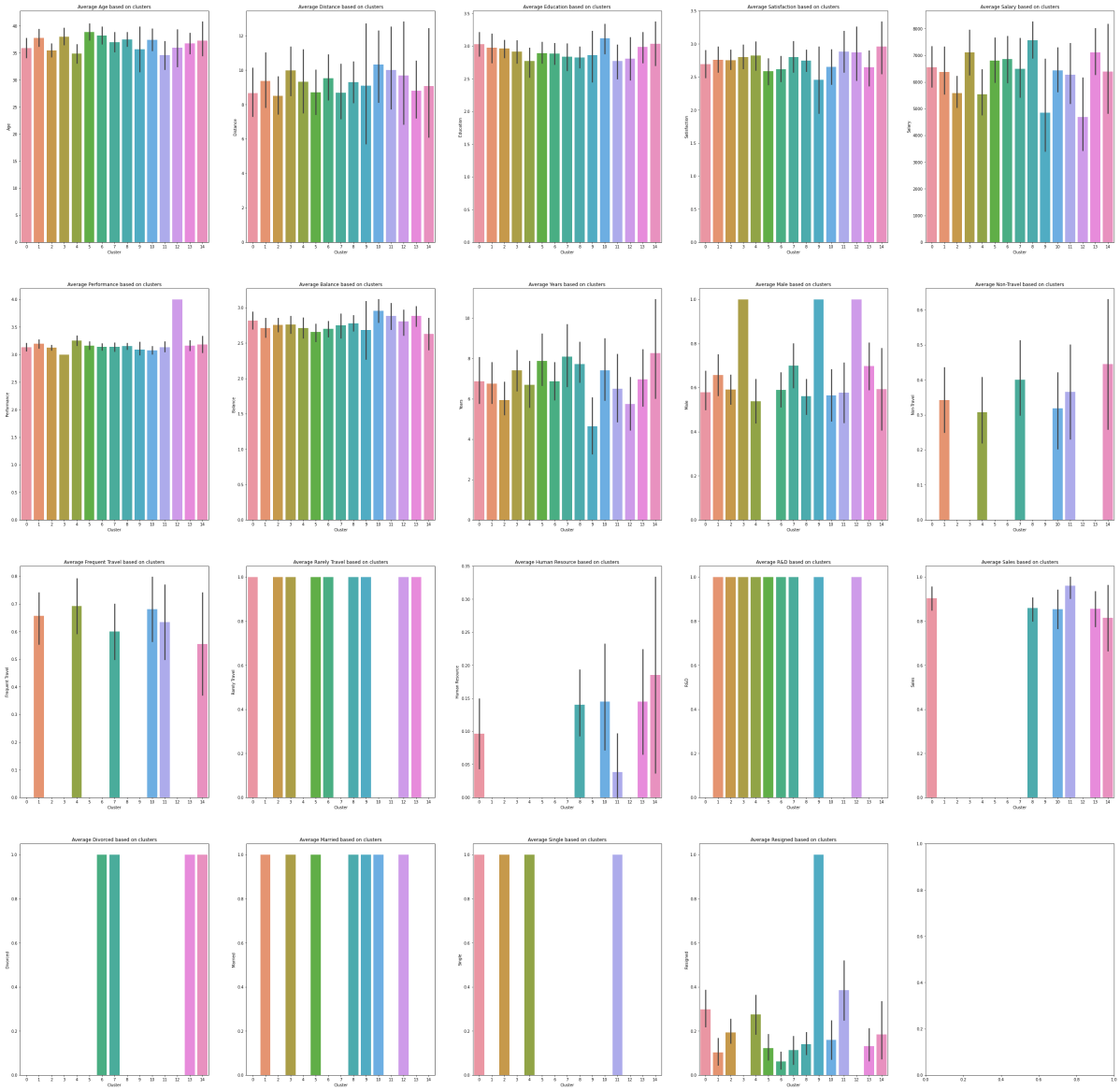
```
)
plt.show()
```



Cluster Visualisation with data

We will split the clusters and see the relationship with the variables

In [ ]:
```
fig, ax = plt.subplots(4, 5, figsize=(50, 50))
db_data = db_df.drop(columns="Cluster")
for i in range(len(db_data.columns)):
    sns.barplot(data=db_df, y=db_data.columns[i], x="Cluster", ax=ax[i // 5, i % 5]
    ax[i // 5, i % 5].set_title(f"Average {db_data.columns[i]} based on clusters")
plt.show()
```



In [ ]:

```
observation = []
for i in range(len(db_data.columns)):
    valueCluster = []
    for cluster in range(len(np.unique(db_label))):
        currentClusterDF = db_df[db_df["Cluster"] == cluster]
        valueCluster.append(np.mean(currentClusterDF.iloc[:, i]))
    observation.append(valueCluster)

observation = pd.DataFrame(observation)
observation["variable"] = db_data.columns
observation.rename_axis("Cluster", axis=1, inplace=True)
observation.set_index("variable", inplace=True)
observation
```

Out[ ]:

Cluster variable	0	1	2	3	4	5	
Age	35.903509	37.750000	35.427230	37.978873	34.846154	38.869231	38
Distance	8.666667	9.370370	8.502347	9.978873	9.329670	8.715385	9
Education	3.026316	2.972222	2.962441	2.915493	2.769231	2.892308	2
Satisfaction	2.692982	2.759259	2.755869	2.802817	2.824176	2.584615	2
Salary	6554.894737	6380.342593	5588.309859	7115.887324	5541.241758	6799.715385	6855
Performance	3.131579	3.194444	3.126761	3.000000	3.252747	3.161538	3
Balance	2.815789	2.712963	2.755869	2.760563	2.714286	2.653846	2
Years	6.868421	6.759259	5.943662	7.415493	6.692308	7.869231	6
Male	0.578947	0.657407	0.591549	1.000000	0.538462	0.000000	0
Non-Travel	0.000000	0.342593	0.000000	0.000000	0.307692	0.000000	0
Frequent Travel	0.000000	0.657407	0.000000	0.000000	0.692308	0.000000	0
Rarely Travel	1.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1
Human Resource	0.096491	0.000000	0.000000	0.000000	0.000000	0.000000	0
R&D	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1
Sales	0.903509	0.000000	0.000000	0.000000	0.000000	0.000000	0
Divorced	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1
Married	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	0
Single	1.000000	0.000000	1.000000	0.000000	1.000000	0.000000	0
Resigned	0.298246	0.101852	0.192488	0.000000	0.274725	0.123077	0

Cluster Interpretation - KMeans

Cluster 0: Young Single Sales Employee that are unsatisfied with the job

Single Rarely Travel Sales employee that are young (<= 35) are likely to quit due to the need to them wanting to make use of their education to do something outside of Sales. Furthermore, they are also not satisfied with the job too.

Cluster 1: Rich Married Male R&D that frequently travel

Married Male R&D that frequently travel that earns more than $6300 are likely to stay in the company as they earn a lot of money. They are unlikely to quit as they have a family to feed.

Cluster 2: Young Single R&D Male that want to climb corporate ladder in a different company

Single R&D Males would likely quit the company as they are young and want to earn more money as they are only paid $5500.

Cluster 3: Rich Married Male R&D Employee

Married Male R&D with high salary that rarely travel stays and are 100% committed to the company because they have a family and is paid enough.

Cluster 4: Young Single R&D Employees that earn very little and have low education

Single R&D that Frequently Travel aged around 34 are likely to leave the company as their salary are very low at around $5500 and get more job opportunity outside of the company. Furthermore, they want to work on themselves as their education is low compared to the others and potentially get married.

Cluster 5: Rich Married Middle Aged R&D Female that performs very well

Middle aged (40) R&D Females that rarely travel are unlikely to quit as their performance are very good and are paid around $6800. They are also married and have a family to feed.

Cluster 6: Rich Divorced Middle Aged R&D Employee

Divorced Middle Aged R&D Employee that rarely travel are unwilling to quit as they earn $6900 and is too old to work on their love life.

Cluster 7: Rich Divorced R&D Employee strong performance

Divorced R&D mostly male employee are unlikely to quit the company as they earn $6500 and have strong performance. Not only that, but also they have worked there for 8 years and climbed the corporate ladder which means quitting would render their hard work useless.

Cluster 8: Rich Married Middle Aged Sales employee

Married Middle Aged Sales employee who earns $7500 is unlikely to quit as they earn so much money and have a family to feed.

Cluster 9: Young Married R&D Male employee that earn less than $5000 and does not perform well and no work life balance

Young Married R&D Male employee that rarely travel and earn less than $5000 are 100% going to resign as they are starting their own families and supporting a family need more money. Furthermore, they are not performing well and could be let go by the company and they felt that there is no work life balance.

Cluster 10: Rich Married Middle Aged Sales employee that love the work life balance

Married and mostly in the sales department employee that frequently travel are unlikely to leave the company as they enjoy living a balanced lifestyle and earns $6400.

Cluster 11: Young Single Sales employee that stay far away from the company
Young Single Sales employee under the age of 35 are likely to quit the company as they live 10 km away from the company making travel time a factor. They are also young and want to make time for themselves as they frequently travel.

Cluster 12: Married R&D employee that performs very well
These employees are 100% committed to the company as they perform very well at their own job which means that they enjoy the job despite the bad pay.

Cluster 13: Rich Divorced Young Sales employee
These employees are unlikely to quit the job as they are paid $7000 and even though they are divorced, they want to help earn money before going back into the dating game.

Cluster 14: Divorced Middle Aged Sales employee no work life balanced
These employees have worked for 8 over years in the company and earn $6000. They are likely to quit the job because they find it hard to balance their work and life.

Summary + Recommendations

As DBSCAN's Clusters give us the most insights in the different clusters, we will be using the DBSCAN data to help and give recommendation to the company.

First, the companies should look out for the following cluster of people

1. Cluster 0: Young Single Sales employees that are unsatisfied with the job (Bored of Sales)
2. Cluster 2: Young Single R&D Male that want to climb the corporate ladder at a different company (Paid too little)
3. Cluster 4: Young Single R&D employee that have very little education and pay (Paid too little and not enough education)
4. Cluster 9: Young Married R&D Male employee that earn less than $5000 and does not perform well and have no work life balance (no work life balance)
5. Cluster 11: Young Single Sales employee stay far away from the company (stay too far)
6. Cluster 14: Divorced Middle Aged Sales employee with no work life balance (no work life balance)

Generally, based on this cluster we can see that the people under 35 are a problem as they do not have a goal in life yet. Pay is also a problem where employees feel that they are undervalued in the company. The company can also give them a more balanced work life by giving them more breaks and allow employees to take leaves.

What can the company do?

1. Management could increase the salary above the average salary or offer excellent benefits like family bonus etc.
    - It will lock employees in and encourage them to stick around for the pay and perks
2. Management can provide external learning opportunities like SkillsFuture to make them upskill themselves so that they do not worry able not having sufficient experience
    - Some employees do the same task over and over again and can get tired of doing the same thing, providing them the opportunity to upskill will make the company more profitable in the long run too
3. Management can give more breaks and allow employees to take more leaves
    - Employees like how flexible it is to work and will start to enjoy work as a whole
4. Management can host workshops and seminars for the employees to encourage and form their dreams and allow them to speak up for themselves
    - This will make employees feel more comfortable and make them feel like work is like home

---

Personal Reflection

The difficulty I found in doing the clustering task is that there were too many features and cleaning and being able to choose which are the best clusters was difficult. Through the assignment I have learnt how to cluster data and find patterns in the data. Dimensional Reduction was also a pain as there were many methods that might have worked better than Gower-TSNE combination that was used in the assignment.