Cluster Analysis of Scaler Learners

Problem Statement:

- Cluster Scaler learners based on job profile, company, and other features to identify groups with similar characteristics, enabling the recommendation of optimal job positions and companies for data science professionals.

Dataset: Scaler database (segment of learners)

Goal: Identify meaningful clusters to inform career development and industry insights.

In [34]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("C:/Users/asus/Downloads/scaler_clustering.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	Unnamed: 0	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
0	0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000	Other	2020.0
1	1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999	FullStack Engineer	2019.0
2	2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000000	Backend Engineer	2020.0
3	3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700000	Backend Engineer	2019.0
4	4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400000	FullStack Engineer	2019.0

Step 1: Importing the Dataset and Basic Exploratory Data Analysis (EDA)

In [2]:
```
df.shape
```

Out[2]:
```
(205843, 7)
```

In [3]:
```
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      153281 non-null  object 
 6   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(2), object(3)
memory usage: 11.0+ MB
```

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
Unnamed: 0              0
company_hash           44
email_hash              0
orgyear                86
ctc                     0
job_position        52562
ctc_updated_year        0
dtype: int64
```

In [17]:
```
df['email_hash'].value_counts().head()
```

Out[17]:
```
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee     9
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
b4d5afa09bec8689017d8b29701b80d664ca37b83cb883376b2e95191320da66     8
Name: email_hash, dtype: int64
```

In [29]:
```
df.describe()
```

Out[29]:

	orgyear	ctc	ctc_updated_year
count	192184.000000	1.921840e+05	192184.000000
mean	2014.823305	2.329745e+06	2019.568138
std	65.622824	1.206217e+07	1.333585
min	0.000000	2.000000e+00	2015.000000
25%	2013.000000	5.450000e+05	2019.000000
50%	2016.000000	9.500000e+05	2020.000000
75%	2018.000000	1.700000e+06	2021.000000
max	20165.000000	1.000150e+09	2021.000000

In [31]:
```
df.describe(include='object').T
```

Out[31]:

	count	unique	top	freq
company_hash	192184	37299	nvnv wgzohrnvzwj otqcxwto	7877
email_hash	192184	153443	6842660273f70e9aa239026ba33bfe82275d6ab0d20124...	9
job_position	192184	1006	Backend Engineer	66494

Insights:

- The dataset contains anonymized columns: Unnamed 0, Email_hash, Company_hash, orgyear, CTC, Job_position, and CTC_updated_year.
- Initial inspection reveals the structure of the data and the presence of missing values.
- Checking the frequency of Email_hash will help in identifying unique learners and any potential issues with duplicate entries.

Step 2: Handling Missing Values

In [35]:
```
df.drop('Unnamed: 0', axis = 1, inplace=True)
```

Imputation Plan

- **Numerical Columns:** using KNN imputer, which helps in filling these values based on the nearest neighbors.
- **Object type Columns:**

  job_position
- This represents the job profile in the company. Missing values in this column can be challenging to handle because job positions can be diverse. We can fill missing values using the mode within groups of related data (e.g., based on company_hash)

  company_hash
- This represents an anonymized identifier for the company, which is the current employer of the learner. Missing values in this column can be filled using the mode

In [36]:
```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5)

# Selecting numeric columns for imputation
numeric_cols = df.select_dtypes(include=[np.number]).columns

df[numeric_cols] = imputer.fit_transform(df[numeric_cols])

df.isnull().sum()
```

Out[36]:
```
company_hash           44
email_hash              0
orgyear                 0
ctc                     0
job_position        52562
ctc_updated_year        0
dtype: int64
```

In [37]:
```
# Fill missing 'company_hash' with mode
company_hash_mode = df['company_hash'].mode()[0]
df['company_hash'].fillna(company_hash_mode, inplace=True)

# Fill missing 'job_position' within groups of 'company_hash' with mode
df['job_position'] = df.groupby('company_hash')['job_position'].apply(lambda x: x.fillna(x.mode()[0] if not x.mod

df.isnull().sum()
```

Out[37]:
```
company_hash        0
email_hash          0
orgyear             0
ctc                 0
job_position        0
ctc_updated_year    0
dtype: int64
```

Insights:

- The company_hash missing values are filled with the most frequent company identifier, which is a reasonable approach given the anonymized nature of the data.
- The job_position missing values are filled within groups of company_hash using the most frequent job position within each company, ensuring consistency and retaining important information.

Step 3: Cleaning Data

In [38]:
```
import re

# Function to remove special characters
def clean_text(text):
    return re.sub('[^A-Za-z0-9 ]+', '', text)

# Applying the function to the job_position column
df['job_position'] = df['job_position'].apply(clean_text)
```

Insights:

- Cleaning the Job_position column to remove any special characters which ensures consistency in data.

In [39]:
```
# Checking for duplicates
duplicates = df.duplicated().sum()
print(f'Number of duplicate rows: {duplicates}')

# Drop duplicates
df = df.drop_duplicates()
```

```
Number of duplicate rows: 13659
```

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
0
```

Insights:

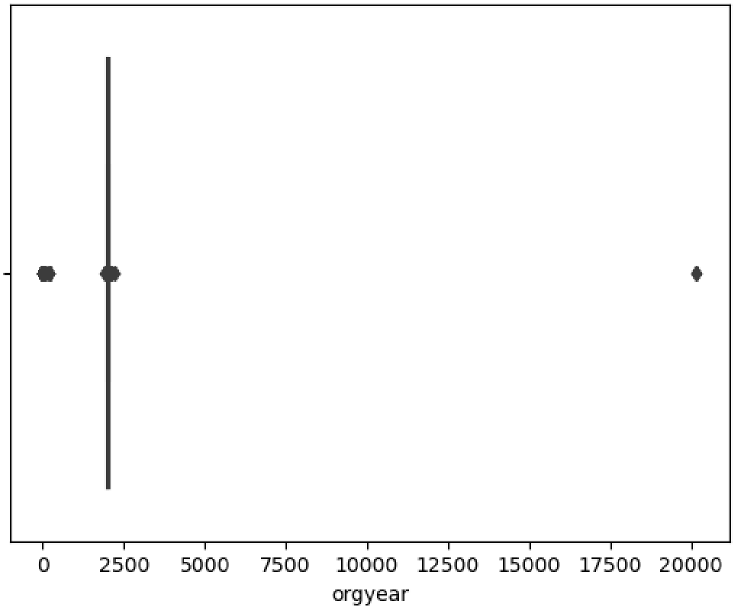
- Identified 13659 rows of duplicate entries. Removed them to maintain data integrity.

Step 4: Anomaly Detection

I have noticed some anomalies in the 'orgyear' column. Year is more than 2024 and way less than 1900 and even there are 0s as values

In [28]:
```
sns.boxplot(df['orgyear'])
```

Out[28]: <AxesSubplot:xlabel='orgyear'>



In [138]:
```
df[df['orgyear'] < 1980].head()
```

Out[138]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
3908	sggsrt	5756870d895deca920251df2377dad261084904a4f9d10...	1973.0	1000.0	Cofounder	2020.0
13424	9xntwyzgrgsj	854ff163ded87211b944dfcaebdcf9e8efa45defc9582f...	0.0	700000.0	Unknown	2021.0
13698	oxtbtzo	4a64fdec422e657b175d5dd914b91e0df7c78ec7716bfe...	208.0	500000.0	Backend Engineer	2020.0
15323	nvnv wgzohrnvzwj otqcxwto nwo	437fa88cd652351931ef679e6b074aa91acb384ef193dd...	209.0	300000.0	Other	2021.0
17139	sgxmxmg	6db474dae5093f975e43697cd77ac5a486248c26235778...	206.0	1500000.0	Backend Engineer	2021.0

In [139]:
```
df[df['orgyear'] > 2024].head()
```

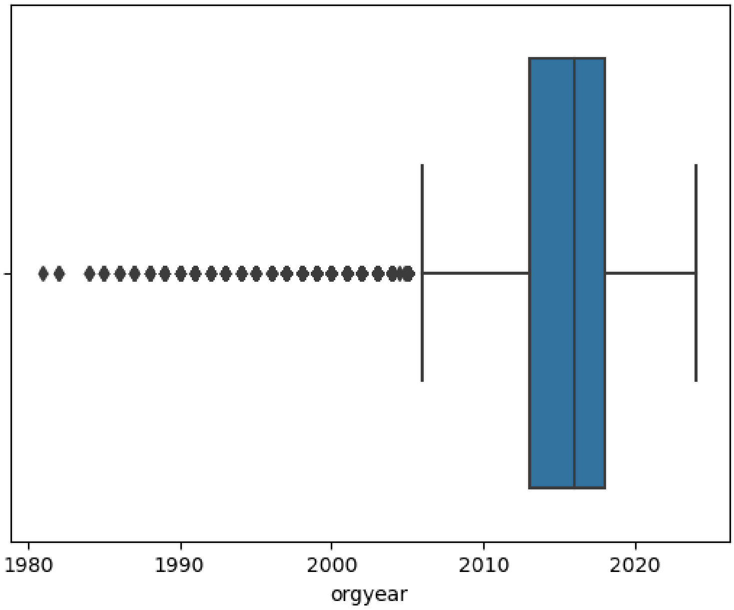
Out[139]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
2211	phrxkv	3394674bb6bb1de6289e931853fa0bd131c811e0054a92...	2031.0	1500000.0	Backend Engineer	2020.0
3651	wgszxkvzn	2cc6bae4e52677d27ce3fca38d7a01ecbe537e1dc1c48d...	2106.0	600000.0	Other	2021.0
10076	xzegojo	4c171381270155fb87b885f89cd71ca37ebbb8fd9da58b...	2025.0	360000.0	Other	2020.0
11081	exqon vacvznvst uqxcvnt rxbxnta	d6df76c2b61fa3a068e4e3812be12a58f86f78a31fe888...	2029.0	310000.0	Other	2020.0
19920	zgn vuurxwvmrt vwwghzn	6aa38b497c73367a7dd6eafb95bdd5b07cca83ed14c588...	2026.0	500000.0	Backend Engineer	2021.0

In [40]:
```
# Calculate median 'orgyear'
median_orgyear = df['orgyear'].median()

# Replace future and unreasonable 'orgyear' values with the median
df.loc[df['orgyear'] > 2024, 'orgyear'] = median_orgyear
df.loc[df['orgyear'] < 1980, 'orgyear'] = median_orgyear
```

In [76]:
```
sns.boxplot(df['orgyear'])
```

Out[76]: <AxesSubplot:xlabel='orgyear'>



Insights

- There were a lot of rows in the column "orgyear" where the values aere either more than 2024 or way less. A person can not have 100 years of experience. Hence, Hardcoded by taking a threshold of 1980 and 2024, rest of the values, replaced with median value

Step 5: Adding New Features

In [41]:
```
# Creating new Features Now

df['Years_of_Experience'] = 2024 - df['orgyear']

# Feature engineering: 'experience_level' based on 'Years_of_Experience'
df['experience_level'] = pd.cut(df['Years_of_Experience'],
                                bins=[-1, 0, 3, 7, 15, np.inf],
                                labels=['Fresher', 'Junior', 'Mid', 'Senior', 'Expert'])
```

In [43]:
```
df['experience_level'].value_counts()
```

Out[43]:
```
Senior     95034
Mid        78452
Expert     14090
Junior      4566
Fresher       42
Name: experience_level, dtype: int64
```

In [143]:
```
95034+78452+14090+4566+42
```

Out[143]: 192184

In [140]:
```
df.shape
```

Out[140]: (192184, 8)

In [44]:
```
df[df['experience_level'].isnull()]
```

Out[44]:

company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level

Insights:

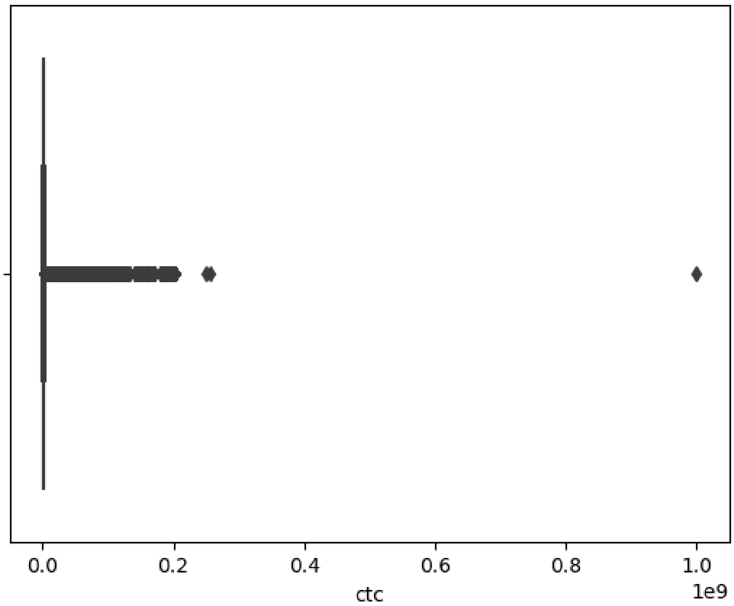
- Initially, there were rows with Years_of_Experience as 0, leading to NaN values for experience_level.
- By including a separate category for 0 years of experience, ensuring all rows have valid experience_level values.
- The experience_level distribution now includes all rows, ensuring consistency and completeness in the dataset.

In [ ]:

Step 6: Univariate Analysis

In [75]:
```
# boxplot of variable 'ctc'
sns.boxplot(df['ctc'])
```

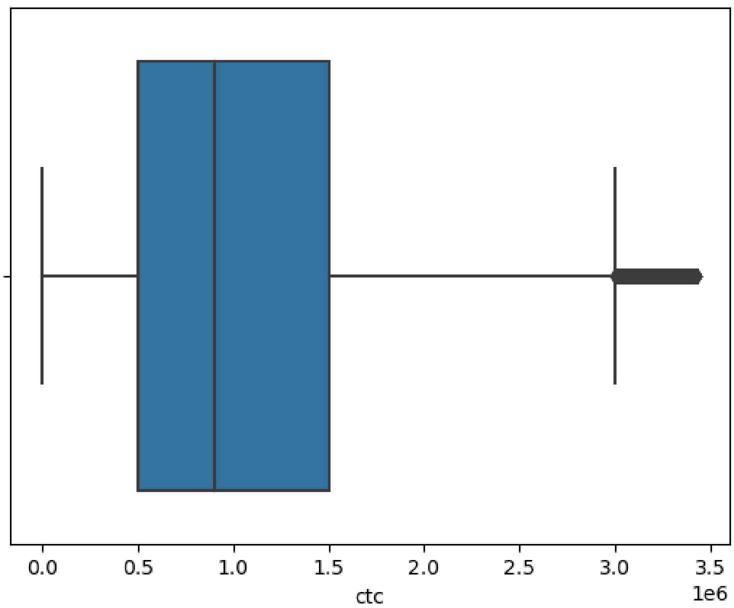
Out[75]: <AxesSubplot:xlabel='ctc'>



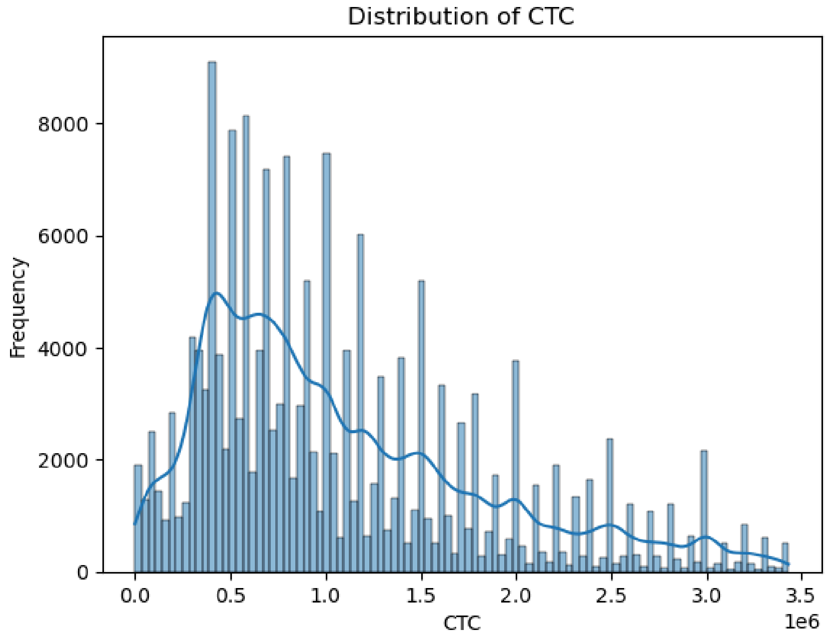
In [45]:
```
# Removing outlier
Q1 = df['ctc'].quantile(0.25)
Q3 = df['ctc'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[(df['ctc'] >= lower_bound) & (df['ctc'] <= upper_bound)]
```

In [81]:
```
sns.boxplot(df['ctc'])
```

Out[81]: <AxesSubplot:xlabel='ctc'>



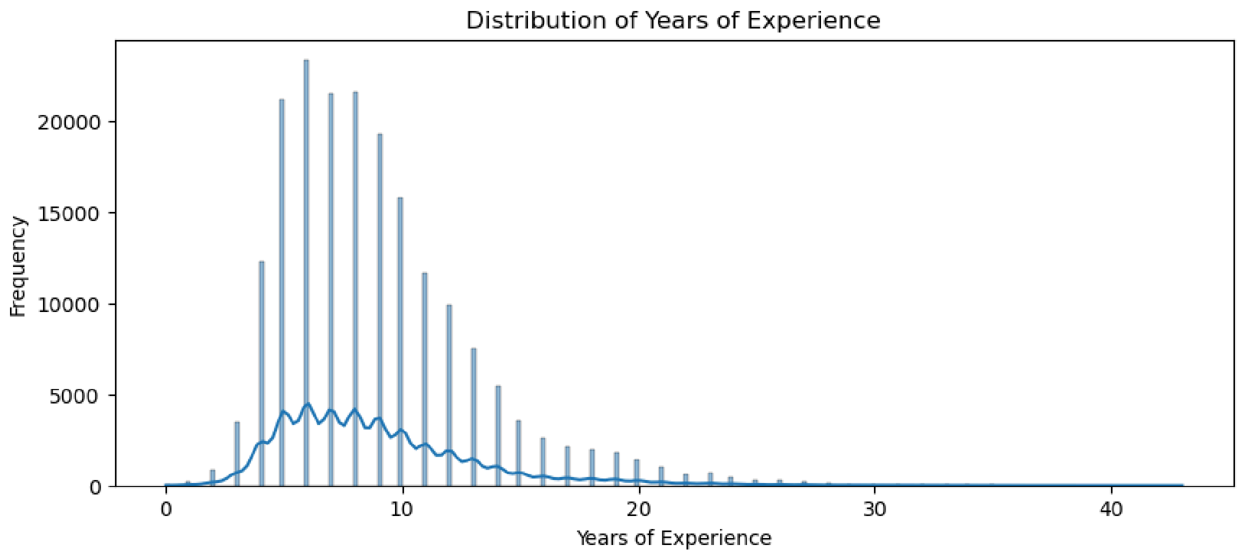
In [78]:
```
sns.histplot(df['ctc'], kde=True)
plt.title('Distribution of CTC')
plt.xlabel('CTC')
plt.ylabel('Frequency')
plt.show()
```



Insights

- Right-Skewed Distribution: Indicates that most employees have lower salaries, while a few have very high salaries.
- Log Transformation: Helps in normalizing the data, making it more suitable for clustering. However, KMeans and Heirarchial Clusterings are robust to non-normality. So not transforming it.
- Capping Outliers: Reduces the impact of extreme values, ensuring that they do not distort the clustering results.

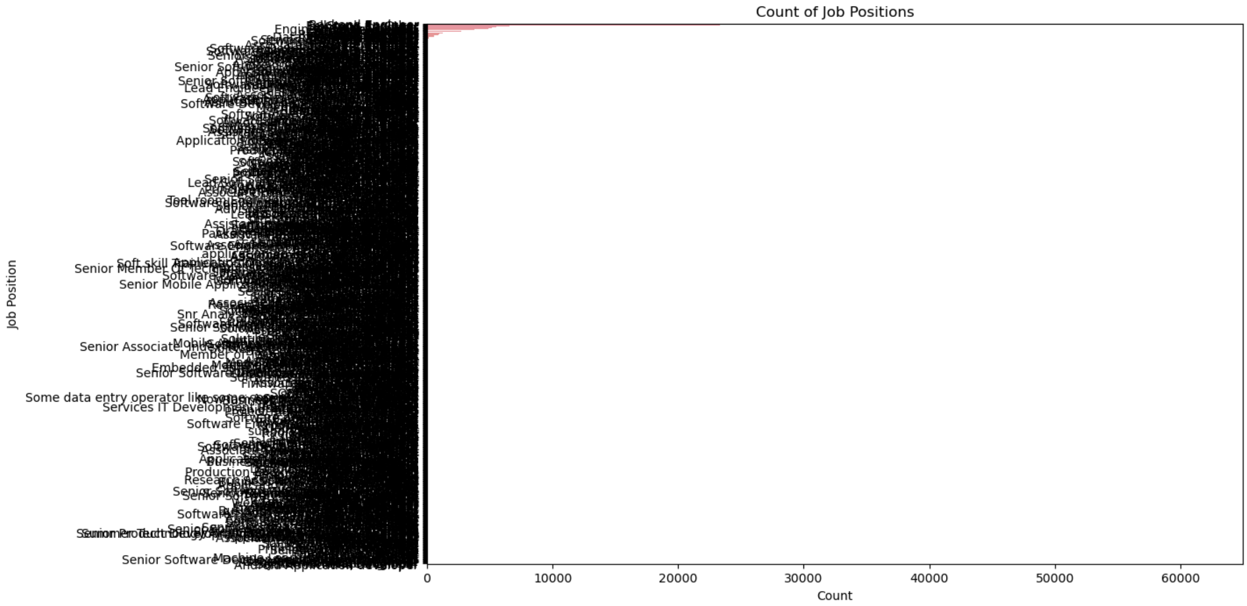
In [58]:
```
plt.figure(figsize=(10, 4))
sns.histplot(df['Years_of_Experience'], kde=True)
plt.title('Distribution of Years of Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Frequency')
plt.show()
```



Insights

- Again, the distribution is not so normal(almost normal). But not transforming as the clustering techniques can handle it.

In [79]:
```
plt.figure(figsize=(12, 8))
sns.countplot(y=df['job_position'], order=df['job_position'].value_counts().index)
plt.title('Count of Job Positions')
plt.xlabel('Count')
plt.ylabel('Job Position')
plt.show()
```



Handling High Cardinality and Imbalance in Categorical Data

Grouping Rare Categories:

- Grouping less frequent job positions into a single category (e.g., "Other") can help reduce the dimensionality and complexity.

Later

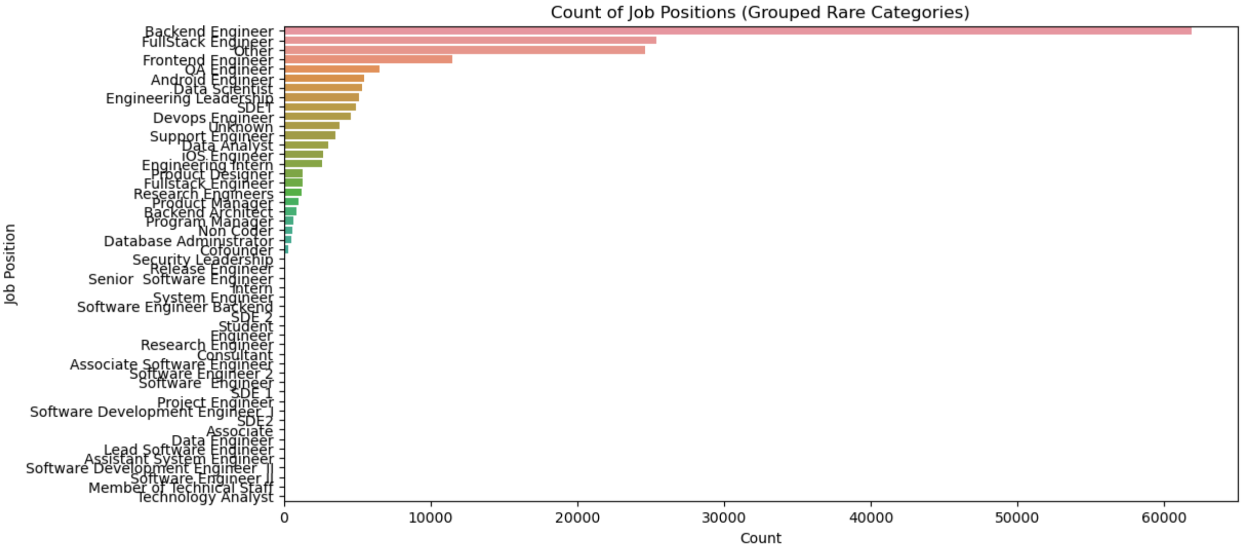
Encoding Categorical Variables:

For clustering, categorical variables need to be encoded numerically. Common methods include one-hot encoding, frequency encoding, or target encoding.

Dimensionality Reduction:

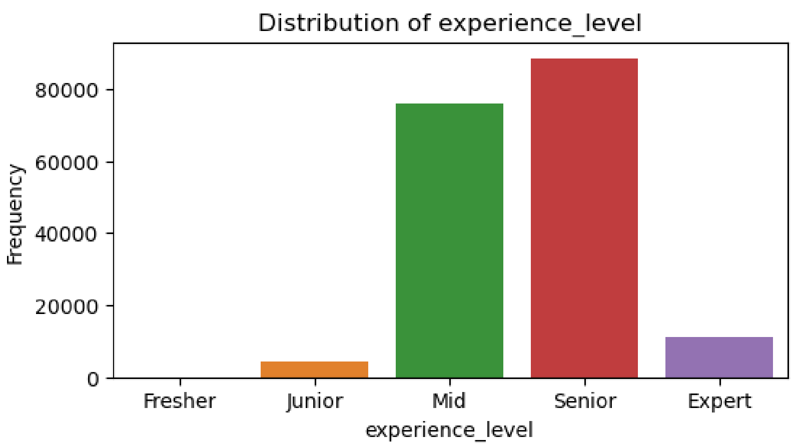
Techniques like PCA (Principal Component Analysis) can be applied after encoding to reduce the feature space, making it more manageable for clustering.

In [46]:
```
# Threshold for rare categories: renaming job_positions as 'Other' which has frequency<10
threshold = 10

# Group rare categories
job_position_counts = df['job_position'].value_counts()
rare_positions = job_position_counts[job_position_counts < threshold].index

df['job_position'] = df['job_position'].apply(lambda x: 'Other' if x in rare_positions else x)

# Visualize the updated job positions distribution
plt.figure(figsize=(12, 6))
sns.countplot(y=df['job_position'], order=df['job_position'].value_counts().index)
plt.title('Count of Job Positions (Grouped Rare Categories)')
plt.xlabel('Count')
plt.ylabel('Job Position')
plt.show()
```



In [110]:

In [26]:
```
plt.figure(figsize=(6, 3))
sns.countplot(df['experience_level'])
plt.title('Distribution of experience_level')
plt.xlabel('experience_level')
plt.ylabel('Frequency')
plt.show()
```



Insights

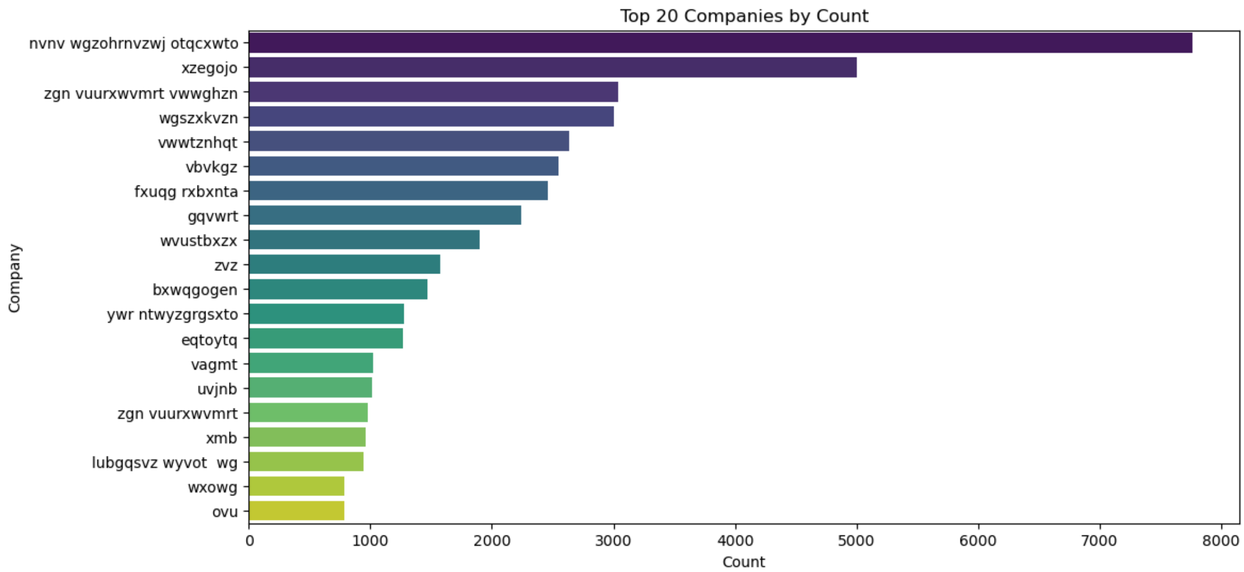
Dominance of Senior and Mid-Level Experience:

The majority of the dataset comprises individuals with Senior and Mid levels of experience.

In [ ]:

showing the graph of top_20 companies

In [27]:
```
# Visualize the top 20 companies by count
top_n = 20
top_companies = df['company_hash'].value_counts().nlargest(top_n)

plt.figure(figsize=(12, 6))
sns.barplot(y=top_companies.index, x=top_companies.values, palette="viridis")
plt.title(f'Top {top_n} Companies by Count')
plt.xlabel('Count')
plt.ylabel('Company')
plt.show()
```



In [ ]:

Dropping the column 'email_hash' as it is a personal identifier and it does not make any impact in the analysis or clustering

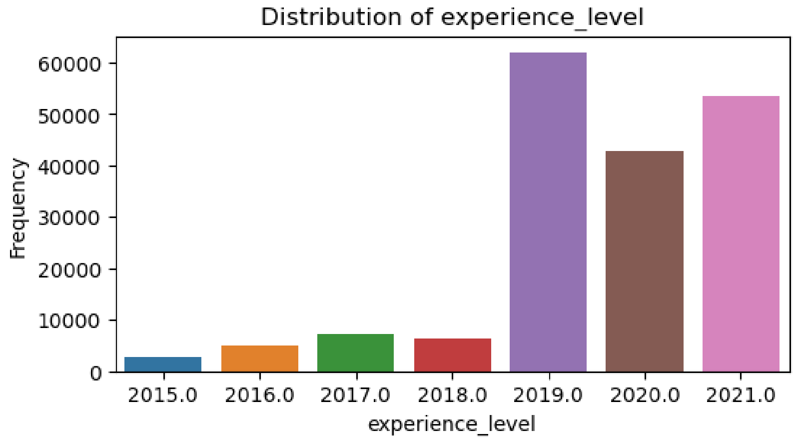
In [47]:
```
df = df.drop('email_hash', axis=1)
```

In [16]:
```
df.head()
```

Out[16]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level
0	atrgxnnt xzaxv	2016.0	1100000.0	Other	2020.0	8.0	Senior
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999.0	FullStack Engineer	2019.0	6.0	Mid
2	ojzwnvwnxw vx	2015.0	2000000.0	Backend Engineer	2020.0	9.0	Senior
3	ngpgutaxv	2017.0	700000.0	Backend Engineer	2019.0	7.0	Mid
4	qxen sqghu	2017.0	1400000.0	FullStack Engineer	2019.0	7.0	Mid

In [30]:
```
plt.figure(figsize=(6, 3))
sns.countplot(df['ctc_updated_year'])
plt.title('Distribution of experience_level')
plt.xlabel('experience_level')
plt.ylabel('Frequency')
plt.show()
```



Insights

The majority of the dataset comprises the years 2019, 2020, 2021 when the employees got a raise.

Step 7: Multivariate Analysis

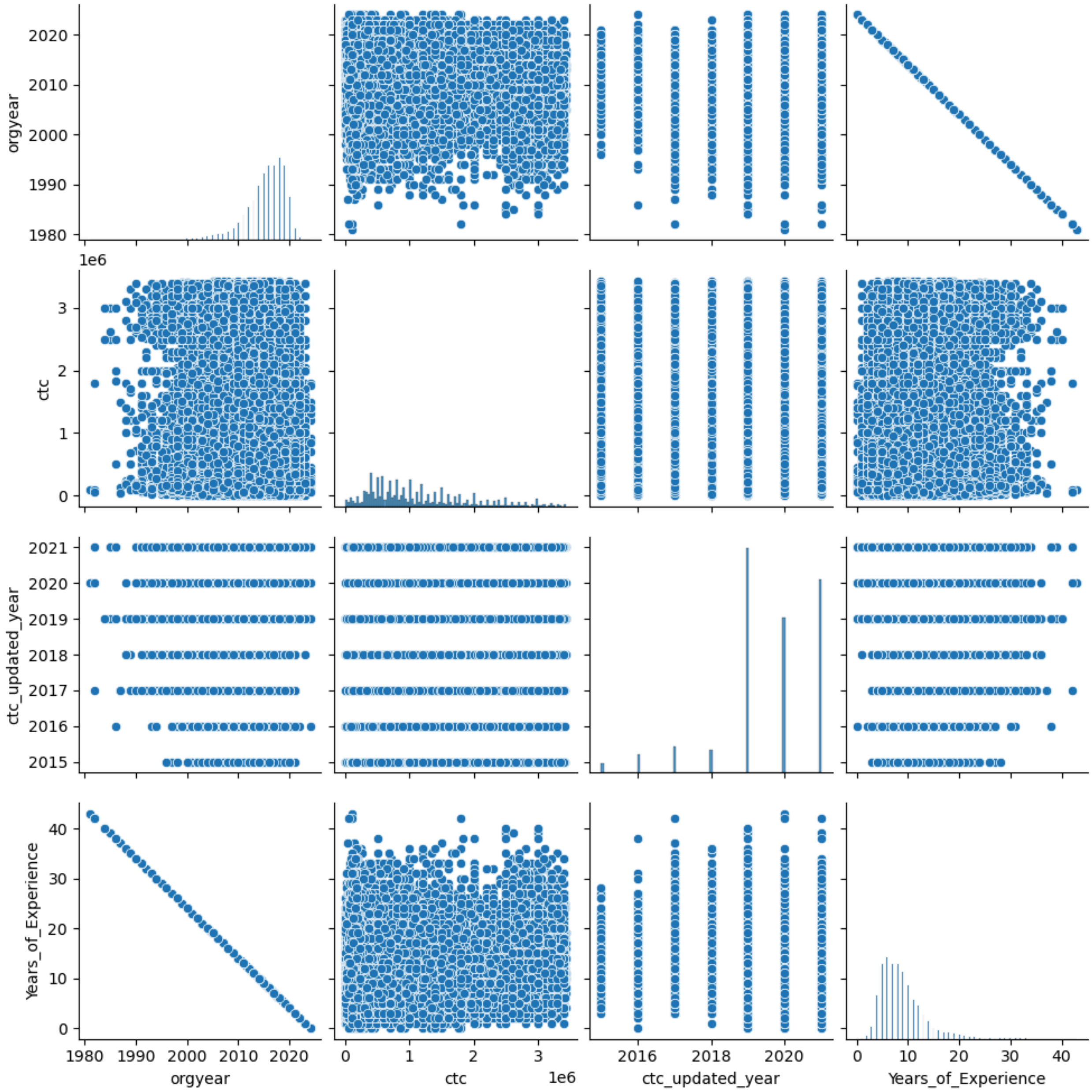
In [48]:
```
df.head()
```

Out[48]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level
0	atrgxnnt xzaxv	2016.0	1100000.0	Other	2020.0	8.0	Senior
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999.0	FullStack Engineer	2019.0	6.0	Mid
2	ojzwnvwnxw vx	2015.0	2000000.0	Backend Engineer	2020.0	9.0	Senior
3	ngpgutaxv	2017.0	700000.0	Backend Engineer	2019.0	7.0	Mid
4	qxen sqghu	2017.0	1400000.0	FullStack Engineer	2019.0	7.0	Mid

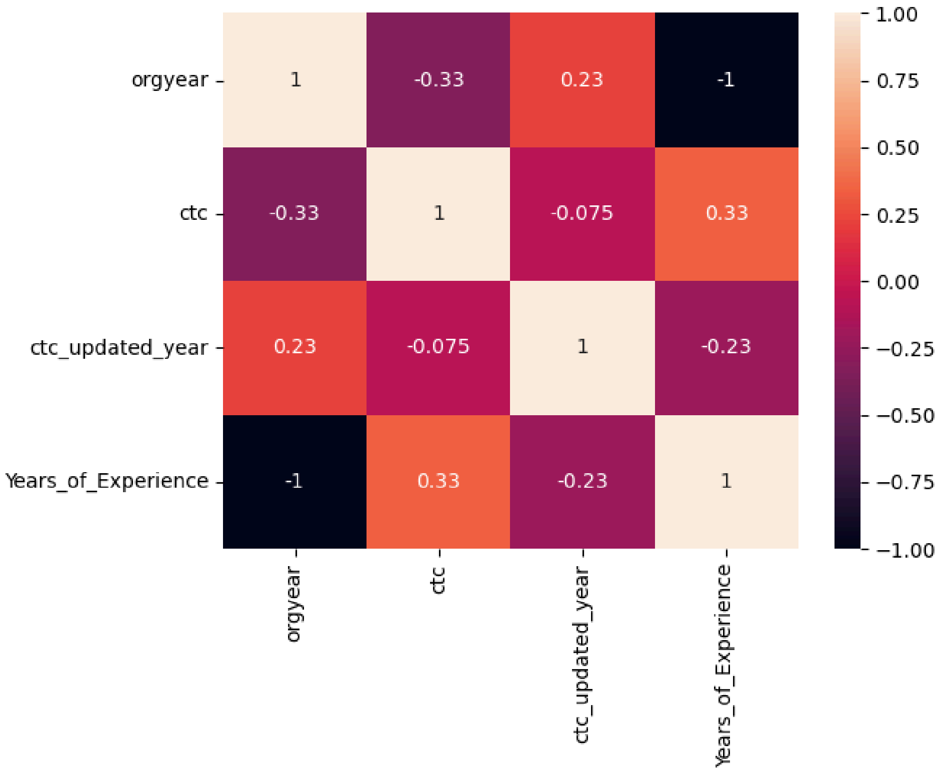
In [149]:
```
sns.pairplot(df)
```

Out[149]: <seaborn.axisgrid.PairGrid at 0x1e90518bb50>



In [160]:
```
sns.heatmap(df.corr(), annot=True)
```

Out[160]: <AxesSubplot:>



Insights from Pairplot & heatmap

CTC vs. Years_of_Experience: We would expect to see a positive correlation between ctc and Years_of_Experience. However, there is no strong correlation. Later I might use PCA for Feature Selection

Years_of_Experience vs. orgyear: I can drop the column 'orgyear' as I have already created a new column 'Years_of_Experience'

CTC vs. orgyear: There is Scattered Distribution and no strong direct correlation, but a pattern could emerge showing that individuals with recent orgyear (newer employees) might have lower ctc due to fewer years of experience.

Step 8: Manual Clustering

In [86]:
```
data = df.copy()
data
```

Out[86]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level
0	atrgxnnt xzaxv	2016.0	1100000.0	Other	2020.0	8.0	Senior
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999.0	FullStack Engineer	2019.0	6.0	Mid
2	ojzwnvwnxw vx	2015.0	2000000.0	Backend Engineer	2020.0	9.0	Senior
3	ngpgutaxv	2017.0	700000.0	Backend Engineer	2019.0	7.0	Mid
4	qxen sqghu	2017.0	1400000.0	FullStack Engineer	2019.0	7.0	Mid
...	...	...	...	...	...	...	...
205837	zgn vuurxwvmrt	2021.0	800000.0	Frontend Engineer	2021.0	3.0	Junior
205838	vuurt xzw	2008.0	220000.0	Unknown	2019.0	16.0	Expert
205839	husqvawgb	2017.0	500000.0	Other	2020.0	7.0	Mid
205840	vwwgrxnt	2021.0	700000.0	Backend Engineer	2021.0	3.0	Junior
205842	bgqsvz onvzrtj	2014.0	1240000.0	Backend Engineer	2016.0	10.0	Senior

179517 rows × 7 columns

5-point summary for Company, Job Position, and Years of Experience

In [87]:
```
# Calculating 5-point summary for Company, Job Position, and Years of Experience
summary = data.groupby(['company_hash', 'job_position', 'Years_of_Experience'])['ctc'].agg(['mean', 'median', 'ma
summary.columns = ['company_hash', 'job_position', 'Years_of_Experience', 'mean_ctc', 'median_ctc', 'max_ctc', 'm
summary.head(20)
```

Out[87]:

	company_hash	job_position	Years_of_Experience	mean_ctc	median_ctc	max_ctc	min_ctc	count_ctc
0	0	Other	4.0	1.000000e+05	100000.0	100000.0	100000.0	1
1	0000	Other	7.0	3.000000e+05	300000.0	300000.0	300000.0	1
2	01 ojztqsj	Android Engineer	8.0	2.700000e+05	270000.0	270000.0	270000.0	1
3	01 ojztqsj	Frontend Engineer	13.0	8.300000e+05	830000.0	830000.0	830000.0	1
4	05mz exzytvrny uqxcvnt rxbxnta	Backend Engineer	5.0	1.100000e+06	1100000.0	1100000.0	1100000.0	1
5	1	Other	2.0	2.500000e+05	250000.0	250000.0	250000.0	1
6	1	Other	7.0	1.000000e+05	100000.0	100000.0	100000.0	1
7	1 axsxnvro	Backend Engineer	6.0	3.500000e+05	350000.0	350000.0	350000.0	1
8	1 jtvq	Backend Engineer	6.0	1.180000e+06	1180000.0	1700000.0	660000.0	2
9	10	Backend Engineer	31.0	4.500000e+05	450000.0	450000.0	450000.0	1
10	10 axsxnvr ahmvx rgzagz	Android Engineer	13.0	1.300000e+06	1300000.0	1300000.0	1300000.0	1
11	1000uqgltwn	Frontend Engineer	5.0	6.000000e+05	600000.0	600000.0	600000.0	1
12	1001 vuuo	Frontend Engineer	9.0	1.650000e+06	1650000.0	1650000.0	1650000.0	1
13	100uxzo	Engineering Intern	6.0	9.000000e+05	900000.0	900000.0	900000.0	1
14	103 onhaxgo ucn rna	Frontend Engineer	10.0	3.200000e+05	320000.0	320000.0	320000.0	1
15	10dvx rtvqzxzs	Data Scientist	4.0	4.000000e+05	400000.0	400000.0	400000.0	1
16	10ev xzaxv ucn rna	Data Analyst	8.0	8.800000e+05	880000.0	880000.0	880000.0	1
17	10hu	Engineering Leadership	11.0	6.600000e+04	66000.0	66000.0	66000.0	1
18	10nxbto	Frontend Engineer	5.0	4.000000e+05	400000.0	400000.0	400000.0	1
19	10nxbto	FullStack Engineer	5.0	4.366667e+05	410000.0	500000.0	400000.0	3

In [88]:
```
# Merging the summary with the original dataset
data = pd.merge(data, summary, on=['company_hash', 'job_position', 'Years_of_Experience'], how='left')
data.head()
```

Out[88]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	mean_ctc	median_ctc	m
0	atrgxnnt xzaxv	2016.0	1100000.0	Other	2020.0	8.0	Senior	1.100000e+06	1100000.0	11(
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999.0	FullStack Engineer	2019.0	6.0	Mid	7.742856e+05	750000.0	12(
2	ojzwnvwnxw vx	2015.0	2000000.0	Backend Engineer	2020.0	9.0	Senior	2.000000e+06	2000000.0	20(
3	ngpgutaxv	2017.0	700000.0	Backend Engineer	2019.0	7.0	Mid	1.455833e+06	1255000.0	31(
4	qxen sqghu	2017.0	1400000.0	FullStack Engineer	2019.0	7.0	Mid	1.400000e+06	1400000.0	14(

flag showing learners with CTC greater than the Average of their Company's department having same Years of Experience

In [89]:
```
# Creating Designation Flag
data['designation_flag'] = data.apply(lambda x: 1 if x['ctc'] > x['mean_ctc'] else 2 if x['ctc'] == x['mean_ctc']
```

Class clustering at Company & Job Position level AND creating Class_flag

In [90]:
```
# Creating Class Flag based on company and job position
class_summary = data.groupby(['company_hash', 'job_position']).ctc.agg(['mean', 'median', 'max', 'min', 'count'])
class_summary.columns = ['company_hash', 'job_position', 'class_mean_ctc', 'class_median_ctc', 'class_max_ctc', '

data = pd.merge(data, class_summary, on=['company_hash', 'job_position'], how='left')

data['class_flag'] = data.apply(lambda x: 1 if x['ctc'] > x['class_mean_ctc'] else 2 if x['ctc'] == x['class_mean
```

Repeating the same analysis at the Company level. Naming that Teir_flag

In [91]:
```
# Creating Tier Flag based on company level
tier_summary = data.groupby('company_hash').ctc.agg(['mean', 'median', 'max', 'min', 'count']).reset_index()
tier_summary.columns = ['company_hash', 'tier_mean_ctc', 'tier_median_ctc', 'tier_max_ctc', 'tier_min_ctc', 'tier

data = pd.merge(data, tier_summary, on='company_hash', how='left')

data['tier_flag'] = data.apply(lambda x: 1 if x['ctc'] > x['tier_mean_ctc'] else 2 if x['ctc'] == x['tier_mean_ct
```

In [92]:
```
data.head()
```

Out[92]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	mean_ctc	median_ctc	m
0	atrgxnnt xzaxv	2016.0	1100000.0	Other	2020.0	8.0	Senior	1.100000e+06	1100000.0	110
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999.0	FullStack Engineer	2019.0	6.0	Mid	7.742856e+05	750000.0	120
2	ojzwnvwnxw vx	2015.0	2000000.0	Backend Engineer	2020.0	9.0	Senior	2.000000e+06	2000000.0	200
3	ngpgutaxv	2017.0	700000.0	Backend Engineer	2019.0	7.0	Mid	1.455833e+06	1255000.0	316
4	qxen sqghu	2017.0	1400000.0	FullStack Engineer	2019.0	7.0	Mid	1.400000e+06	1400000.0	140

5 rows × 25 columns

Based on the manual clustering done so far, answering few questions like:

- Top 10 employees (earning more than most of the employees in the company) - Tier 1
- Top 10 employees of data science in each company earning more than their peers - Class 1
- Bottom 10 employees of data science in each company earning less than their peers - Class 3
- Bottom 10 employees (earning less than most of the employees in the company)- Tier 3
- Top 10 employees in each company - X department - having 5/6/7 years of experience earning more than their peers - Tier X
- Top 10 companies (based on their CTC)
- Top 2 positions in every company (based on their CTC)

In [93]:
```
# Top 10 employees - Tier 1
top_10_employees_tier1 = data[data['tier_flag'] == 1].sort_values(by='ctc', ascending=False).head(10)
top_10_employees_tier1
```

Out[93]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	mean_ctc	median_ct
162730	atrr ntwyzgrgsxto	2017.0	3430000.0	Backend Engineer	2019.0	7.0	Mid	1.093857e+06	1050000.(
70179	yvqbvz wgzztwnta otqcxwto	2013.0	3430000.0	Frontend Engineer	2019.0	11.0	Senior	1.592400e+06	1002000.(
164301	zhnvzxd	2016.0	3430000.0	Backend Engineer	2020.0	8.0	Senior	2.810000e+06	2750000.(
108503	ftrro evqsg	2020.0	3430000.0	FullStack Engineer	2019.0	4.0	Mid	2.230000e+06	2060000.(
39359	utqoxontzn ojontbo	2014.0	3430000.0	Backend Engineer	2021.0	10.0	Senior	1.324615e+06	1240000.(
171356	nvxontwy	2010.0	3429999.0	Frontend Engineer	2019.0	14.0	Senior	3.429999e+06	3429999.(
174183	zgcvqnxo	2016.0	3429999.0	Data Scientist	2019.0	8.0	Senior	3.415000e+06	3414999.!
115381	yvwptqqvzp	2010.0	3429999.0	Devops Engineer	2018.0	14.0	Senior	3.429999e+06	3429999.(
160604	nvxontwy	2010.0	3429999.0	Backend Architect	2019.0	14.0	Senior	3.429999e+06	3429999.(
128594	tqxwoogz	2008.0	3429999.0	Data Scientist	2019.0	16.0	Expert	2.915000e+06	2914999.!

10 rows × 25 columns

In [94]:
```
# Top 10 employees of data science in each company earning more than their peers - Class 1

# Step 1: Filter for Data Science Employees
data_scientists_df = data[data['job_position'] == 'Data Scientist']

# Step 2: Filter for Class 1 Employees
class1_data_scientists_df = data_scientists_df[data_scientists_df['class_flag'] == 1]

# Step 3: Group by Company and Get Top 10 within each company
top_10_employees_class1 = class1_data_scientists_df.groupby('company_hash').apply(lambda x: x.nlargest(10, 'ctc')

top_10_employees_class1
```

Out[94]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	mean_ctc	median_ctc	m
0	1bs	2018.0	1100000.0	Data Scientist	2021.0	6.0	Mid	1100000.0	1100000.0	11(
1	2020	2020.0	2700000.0	Data Scientist	2019.0	4.0	Mid	1444000.0	1000000.0	27(
2	2020	2020.0	2100000.0	Data Scientist	2019.0	4.0	Mid	1444000.0	1000000.0	27(
3	247vx	2010.0	2600000.0	Data Scientist	2015.0	14.0	Senior	2600000.0	2600000.0	26(
4	247vx	2008.0	2500000.0	Data Scientist	2019.0	16.0	Expert	2500000.0	2500000.0	25(
...	...	...	...	...	...	...	...	...	...	
1381	zxwt xzntqvwnxct ogrhnxgzo	2013.0	1130000.0	Data Scientist	2019.0	11.0	Senior	1130000.0	1130000.0	11:
1382	zxxn ntwyzgrgsxto	2012.0	2200000.0	Data Scientist	2019.0	12.0	Senior	2200000.0	2200000.0	22(
1383	zxxn ntwyzgrgsxto rxbxnta	2015.0	1500000.0	Data Scientist	2020.0	9.0	Senior	1500000.0	1500000.0	15(
1384	zxxn ntwyzgrgsxto rxbxnta	2014.0	1200000.0	Data Scientist	2021.0	10.0	Senior	1200000.0	1200000.0	12(
1385	zxztrtvuo	2014.0	2250000.0	Data Scientist	2021.0	10.0	Senior	2250000.0	2250000.0	22!

1386 rows × 25 columns

In [97]:
```
# Bottom 10 employees of data science in each company earning less than their peers - Class 3

# Filter for Class 3 Employees
class3_data_scientists_df = data_scientists_df[data_scientists_df['class_flag'] == 3]

# Step 3: Group by Company and Get Top 10 within each company
top_10_employees_class3 = class3_data_scientists_df.groupby('company_hash').apply(lambda x: x.nsmallest(10, 'ctc'

top_10_employees_class3
```

Out[97]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	mean_ctc	median_ctc	m
0	1bs	1994.0	800000.0	Data Scientist	2019.0	30.0	Expert	800000.0	800000.0	8(
1	2020	2020.0	700000.0	Data Scientist	2020.0	4.0	Mid	1444000.0	1000000.0	27(
2	2020	2020.0	720000.0	Data Scientist	2019.0	4.0	Mid	1444000.0	1000000.0	27(
3	2020	2020.0	1000000.0	Data Scientist	2019.0	4.0	Mid	1444000.0	1000000.0	27(
4	247vx	2002.0	1440000.0	Data Scientist	2019.0	22.0	Expert	1440000.0	1440000.0	144
...	...	...	...	...	...	...	...	...	...	
1513	zxxn ntwyzgrgsxto	2018.0	1500000.0	Data Scientist	2019.0	6.0	Mid	1500000.0	1500000.0	15(
1514	zxxn ntwyzgrgsxto rxbxnta	2012.0	800000.0	Data Scientist	2021.0	12.0	Senior	800000.0	800000.0	8(
1515	zxztrtvuo	2019.0	1250000.0	Data Scientist	2021.0	5.0	Mid	1250000.0	1250000.0	12
1516	zxztrtvuo	2018.0	1370000.0	Data Scientist	2019.0	6.0	Mid	1370000.0	1370000.0	13
1517	zxztrtvuo	2017.0	1400000.0	Data Scientist	2019.0	7.0	Mid	1400000.0	1400000.0	14(

1518 rows × 25 columns

In [96]:
```
# Bottom 10 employees (earning less than most of the employees in the company)- Tier 3

tier3_emp = data[data['tier_flag'] == 3]

bottom_10_employees_tier3 = tier3_emp.groupby('company_hash').apply(lambda x: x.nsmallest(10, 'ctc')).reset_index

bottom_10_employees_tier3
```

Out[96]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	mean_ctc	median_ctc
0	01 ojztqsj	2016.0	270000.0	Android Engineer	2019.0	8.0	Senior	2.700000e+05	270000.0
1	1	2017.0	100000.0	Other	2020.0	7.0	Mid	1.000000e+05	100000.0
2	1 jtvq	2018.0	660000.0	Backend Engineer	2019.0	6.0	Mid	1.180000e+06	1180000.0
3	10nxbto	2019.0	400000.0	Frontend Engineer	2020.0	5.0	Mid	4.000000e+05	400000.0
4	10nxbto	2019.0	400000.0	FullStack Engineer	2020.0	5.0	Mid	4.366667e+05	410000.0
...	...	...	...	...	...	...	...	...	...
28645	zxztrtvuo	2020.0	450000.0	Backend Engineer	2020.0	4.0	Mid	5.888889e+05	450000.0
28646	zxztrtvuo	2019.0	450000.0	Backend Engineer	2020.0	5.0	Mid	6.370000e+05	500000.0
28647	zxztrtvuo	2020.0	450000.0	Backend Engineer	2019.0	4.0	Mid	5.888889e+05	450000.0
28648	zyco xzaxv	2013.0	500000.0	Other	2019.0	11.0	Senior	5.500000e+05	550000.0
28649	zz	2009.0	500000.0	Other	2021.0	15.0	Senior	5.000000e+05	500000.0

28650 rows × 25 columns

In [98]:
```
# Top 10 companies (based on their CTC)
top_10_companies = data.groupby('company_hash')['ctc'].mean().sort_values(ascending=False).head(10)
top_10_companies
```

Out[98]:
```
company_hash
mvzp ge vbtqxwv  yjatqvmva                         3429999.0
btqwtato mtzk qtotvqwy vza atctrgubtzn wtzntq      3429999.0
mvqwrvjo ntwyzgrgsj wtzntq                         3420000.0
st xzahonqxvr                                      3410000.0
ntwymrht ogenfvqto ucn rna                         3400000.0
phrn wgobtnxwo ucn rna                             3400000.0
wgatzvnxgz xzzgcvnxgz rvmowzxr                     3400000.0
mzjbtrrgz ntwyzgrgsxto                             3400000.0
nfgagnotctz                                        3400000.0
vjrv ztnfgqpo                                      3400000.0
Name: ctc, dtype: float64
```

In [99]:
```
# Top 2 positions in every company (based on their CTC)
top_2_positions = data.groupby(['company_hash', 'job_position'])['ctc'].mean().reset_index().sort_values(by=['com
top_2_positions
```

Out[99]:

	company_hash	job_position	ctc
0	0	Other	100000.0
1	0000	Other	300000.0
3	01 ojztqsj	Frontend Engineer	830000.0
2	01 ojztqsj	Android Engineer	270000.0
4	05mz exzytvrny uqxcvnt rxbxnta	Backend Engineer	1100000.0
...	...	...	...
58018	zyvzwt wgzohrnxzs tzsxzttqo	Frontend Engineer	940000.0
58019	zz	Other	935000.0
58020	zzb ztdnstz vacxogqj ucn rna	FullStack Engineer	600000.0
58021	zzgato	Unknown	130000.0
58022	zzzbzb	Other	720000.0

43516 rows × 3 columns

In [ ]:

Step 9: Data processing for Unsupervised clustering

Encoding to convert into numerical

1. One-Hot Encoding This is a straightforward approach where each category is represented by a binary vector. One-hot encoding is beneficial when there are a reasonable number of categories.

Advantages: 1)Does not assume any ordinal relationship between categories. 2)Well-suited for algorithms that do not require numerical order, like tree-based methods.

Disadvantages: Can lead to a high-dimensional feature space if the number of categories is large.

2. Frequency Encoding This method replaces each category with the frequency of its occurrence. It's useful when dealing with high cardinality and can help to maintain the order of categories based on their frequency.

Advantages: 1) Reduces dimensionality compared to one-hot encoding. 2) Can capture some information about the distribution of categories. **Disadvantages:** May not perform well if the distribution of categories is skewed.

Approach for Clustering

For clustering, it's often best to balance simplicity and the ability to capture useful information. Therefore, frequency encoding is a good choice as it reduces dimensionality and captures the distribution of categories without introducing high-dimensional data.

In [49]:
```
freq_encoding = df['job_position'].value_counts().to_dict()
df['job_position'] = df['job_position'].map(freq_encoding)

df['job_position'].unique()
```

Out[49]:
```
array([24648, 25412, 61879,  3795,  2715,  3019, 11462,  6544,  5142,
        5314,  2625,  5497,  1202,  1276,  4912,  3545,   130,  4543,
         985,  1285,   649,   329,    39,   901,   526,   117,   583,
          24,    16,    15,    51,    22,    14,    11,    42,    17,
          10,    21,    18,    12,    19], dtype=int64)
```

Encoding of the variable 'experience_level'

- Since there is a clear ordering in the categories, using Ordinal Encoder

In [50]:
```
from sklearn.preprocessing import OrdinalEncoder

experience_level_order = ['Fresher', 'Junior', 'Mid', 'Senior', 'Expert']

# Create an OrdinalEncoder object with the specified order
ordinal_encoder = OrdinalEncoder(categories=[experience_level_order])

# Fit and transform the experience_level column
df['experience_level'] = ordinal_encoder.fit_transform(df[['experience_level']])
```

Encoding:

- For 'company_hash', one-hot encoding is impractical due to the high number of unique values.
- Frequency Encoding, which is more suitable for high cardinality categorical variables.

Absolute Frequency Encoder

In [51]:
```
# Absolute Frequency Encoding for company_hash
company_freq_abs = df['company_hash'].value_counts()
df['company_hash'] = df['company_hash'].map(company_freq_abs)
```

In [52]:
```
df.head()
```

Out[52]:

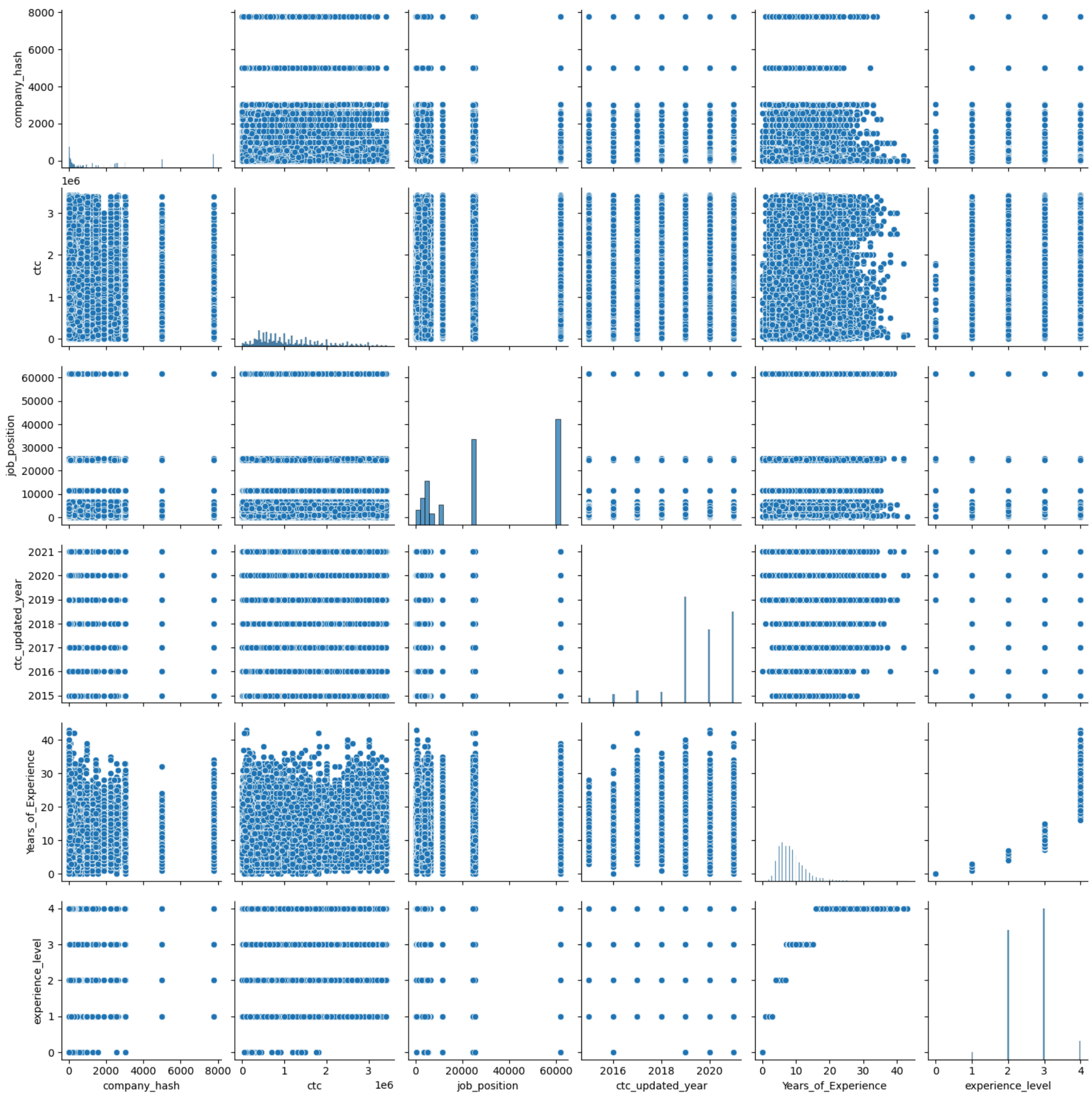
	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level
0	9	2016.0	1100000.0	24648	2020.0	8.0	3.0
1	386	2018.0	449999.0	25412	2019.0	6.0	2.0
2	1	2015.0	2000000.0	61879	2020.0	9.0	3.0
3	63	2017.0	700000.0	61879	2019.0	7.0	2.0
4	6	2017.0	1400000.0	25412	2019.0	7.0	2.0

Dropping the column 'orgyear'

In [83]:
```
df = df.drop('orgyear', axis=1)
```

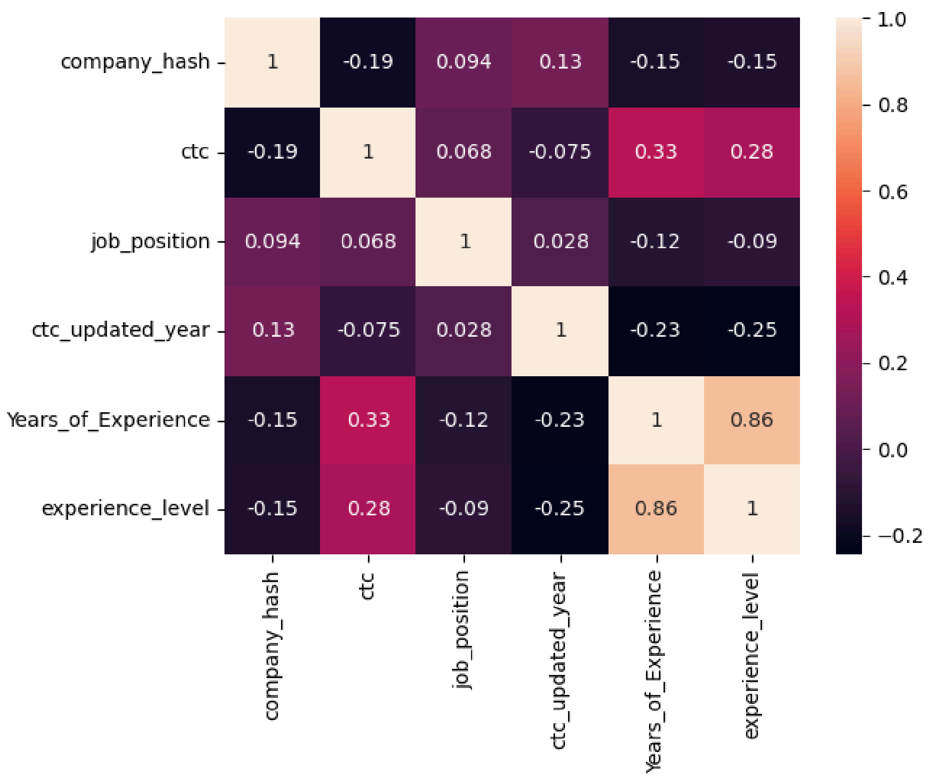
In [23]:
```
sns.pairplot(df)
```

Out[23]: <seaborn.axisgrid.PairGrid at 0x24b211e2250>



In [24]:
```
sns.heatmap(df.corr(), annot=True)
```

Out[24]: <AxesSubplot:>



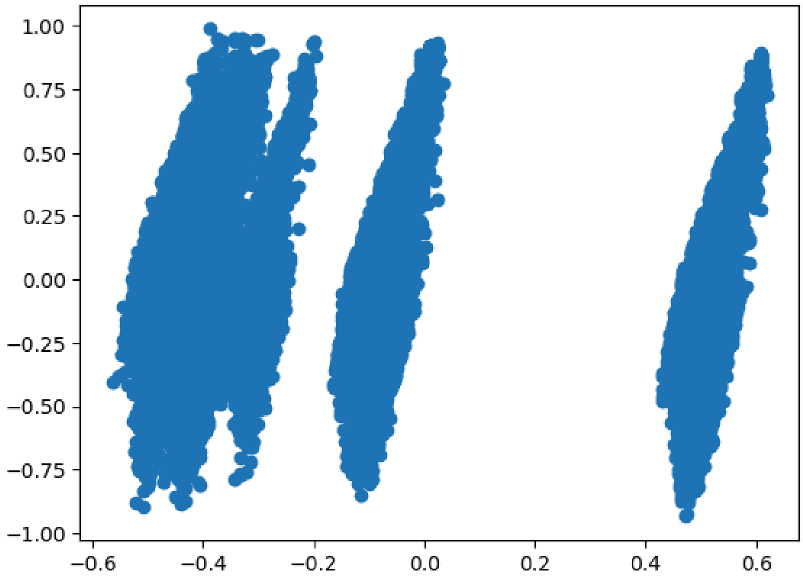
Feature Scaling

- K-Means is a distance-based algorithm. Because of that, it's really important to perform feature scaling (normalize, standardize, or choose any other option in which the distance has some comparable meaning for all the columns).
- For our use case, we can use MinMaxScaler instead of StandardScaler, transforming the feature values to fall within the bounded intervals (min and max), rather than making them to fall around mean as 0 with standard deviation as 1 (StandardScaler).
- MinMaxScaler is an excellent tool for this purpose. MinMaxScaler scales all the data features in the range [0, 1] or else in the range [-1, 1] if there are negative values in the dataset. This scaling compresses all the inliers in the narrow range [0, 0.005].

In [98]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize the scaler
scaler = MinMaxScaler()

# Fit and transform the numerical columns
df_scaled = scaler.fit_transform(df)
```

PCA-Visualizing the data in 2D

In [105]:
```
from sklearn.decomposition import PCA

pca = PCA(2)

components = pca.fit_transform(df_scaled)

x_1 = components[:,0]
x_2 = components[:,1]

plt.scatter(x_1,x_2)
plt.show()
```



Step 10: Clustering/Modelling: KMeans

In [99]:
```
from sklearn.cluster import KMeans

k = 4 ## arbitrary value
kmeans = KMeans(n_clusters=k,random_state = 42)
y_pred = kmeans.fit_predict(df_scaled)
```

In [65]:
```
y_pred
```

Out[65]:
```
array([0, 0, 3, ..., 0, 3, 3])
```

In [55]:
```
##coordinates of the cluster centers
kmeans.cluster_centers_
```

Out[55]:
```
array([[0.06950653, 0.84465443, 0.21986774, 0.26944572, 0.85925864,
        0.15534557, 0.56162664],
       [0.86356422, 0.83994411, 0.16922323, 0.54682923, 0.84219647,
        0.16005589, 0.57951028],
       [0.0338892 , 0.7251896 , 0.42043297, 0.15369722, 0.63801958,
        0.2748104 , 0.77112923],
       [0.07058946, 0.80128308, 0.36348448, 1.        , 0.75247317,
        0.19871692, 0.64549856]])
```

PCA visualization of KMeans = 4 clusters

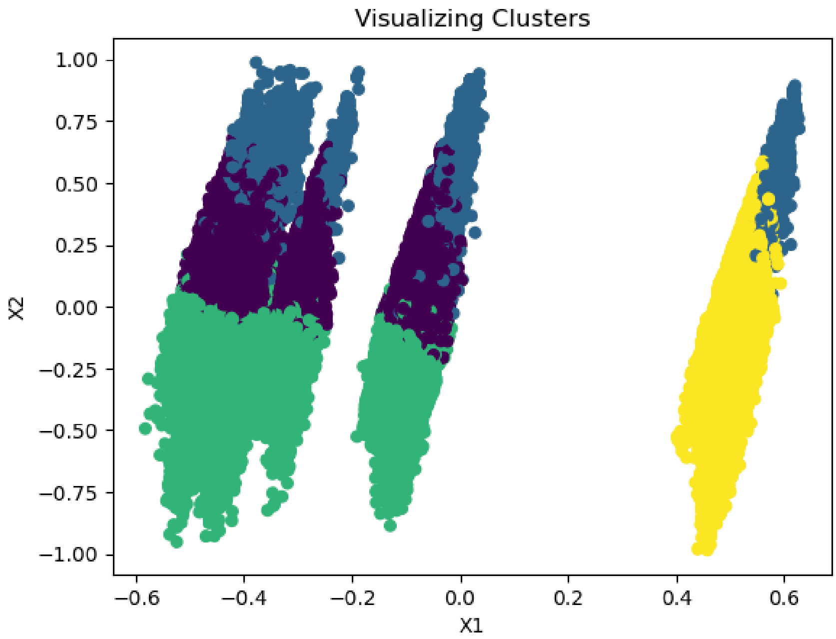
In [85]:
```
from sklearn.decomposition import PCA

pca = PCA(2)

components_pca = pca.fit_transform(df_scaled)
```

In [86]:
```
clusters = pd.DataFrame(components_pca, columns=['X1', 'X2'])
clusters['label'] = kmeans.labels_
clusters.head()
```

Out[86]:

	X1	X2	label
0	-0.103054	-0.056599	0
1	-0.081938	0.104864	0
2	0.501230	-0.255227	3
3	0.499815	-0.014849	3
4	-0.078332	-0.079219	0

In [87]:
```
def viz_clusters(clusters):
    plt.scatter(clusters['X1'], clusters['X2'], c=clusters['label'], s = 30)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title('Visualizing Clusters')

viz_clusters(clusters)
```



In [70]:
```
import warnings
warnings.filterwarnings('ignore')
```

- **There is some distinction between clusters, but making sense out of this is a bit hard from this plot.**

Let's try t-SNE

Visualizing clusters - tSNE

In [89]:
```
from sklearn.manifold import TSNE

tsne = TSNE(2)

components_tsne = tsne.fit_transform(df_scaled)
```

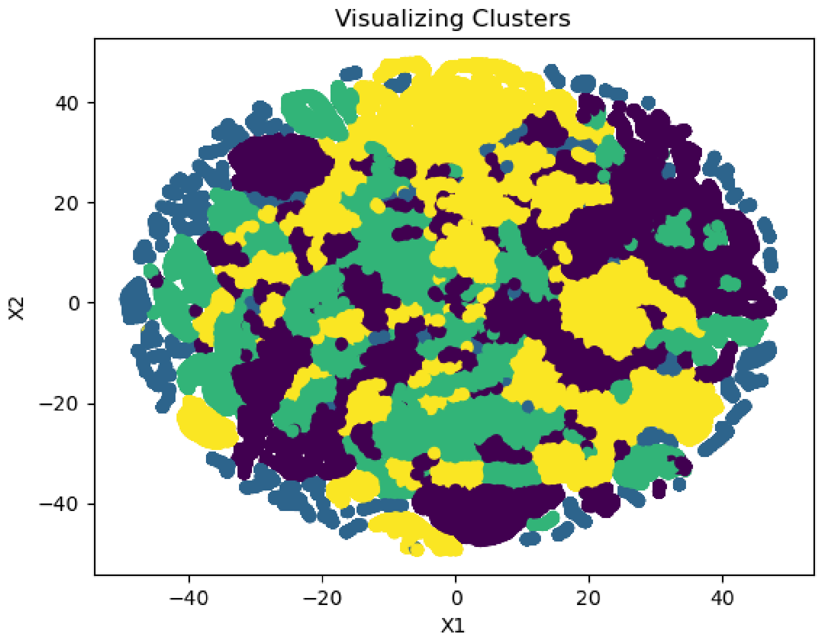
In [90]:
```
clusters_tsne = pd.DataFrame(components_tsne, columns=['X1', 'X2'])
clusters_tsne['label'] = kmeans.labels_
clusters_tsne.head()
```

Out[90]:

	X1	X2	label
0	40.965763	-10.084338	0
1	20.837851	15.890117	0
2	4.014632	11.704596	3
3	3.927227	26.723278	3
4	35.704269	15.749392	0

In [ ]:

In [91]:
```
viz_clusters(clusters_tsne)
```



- **It's even harder to distinct the clusters**

A better alternative would be a line polar plot from plotly library - useful for visualizing multi-dimensional data

- Group the customers by labels and calculate mean for all the features.
- Melt the data to have features on rows along with their corresponding mean values

In [100]:
```
clusters_df = pd.DataFrame(df_scaled, columns=df.columns)
clusters_df['label'] = kmeans.labels_
clusters_df.head()
```

Out[100]:

	company_hash	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	label
0	0.001031	0.320699	0.398229	0.833333	0.186047	0.75	3
1	0.049601	0.131195	0.410577	0.666667	0.139535	0.50	3
2	0.000000	0.583090	1.000000	0.833333	0.209302	0.75	1
3	0.007988	0.204081	1.000000	0.666667	0.162791	0.50	1
4	0.000644	0.408163	0.410577	0.666667	0.162791	0.50	3

In [101]:
```
polar = clusters_df.groupby("label").mean().reset_index()
polar = pd.melt(polar, id_vars=["label"])
polar.head()
```

Out[101]:

	label	variable	value
0	0	company_hash	0.033501
1	1	company_hash	0.070529
2	2	company_hash	0.863252
3	3	company_hash	0.069854
4	0	ctc	0.426191

In [102]:
```
import plotly.express as px

"""
  'polar' : customer dataset we are using
  'r' :  mean values for each feature which will be connected using lines
  'theta' : variables where each of the feature will have an angle and
            color will be based on the label of the clusters.
"""
fig = px.line_polar(polar, r="value", theta="variable", color="label", line_close=True,height=700,width=800)
fig.show()
```

Insights

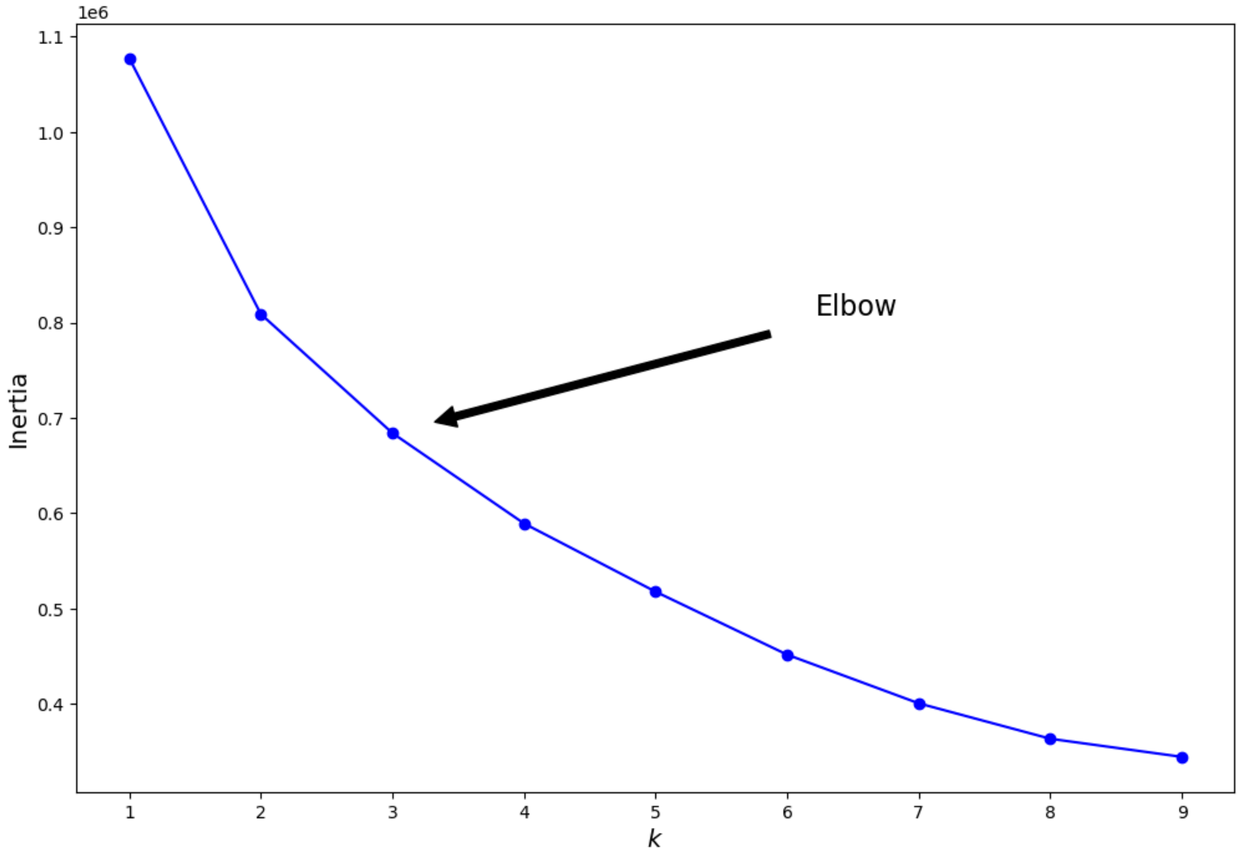
- **Polar plot is read and interpreted radially**
  - values increase as we move away from the center, showing the influence of a feature on that label.
  - green(2) and purple(3) overlap on all the features except one.

Looking at this plot, we have different customer segments:

- As this data has a mostly categorical features, the polar plot would not make a very good sense. However, we can clearly see the segmentation of clusters.

In [103]:
```
# Inertia = Within Cluster Sum of Squares
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(df_scaled)
                for k in range(1, 10)]

inertias = [model.inertia_ for model in kmeans_per_k]
```

In [131]:
```
plt.figure(figsize=(12, 8))
plt.plot(range(1, 10), inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.annotate('Elbow',
             xy=(3, inertias[2]),
             xytext=(0.55, 0.55),
             textcoords='figure fraction',
             fontsize=16,
             arrowprops=dict(facecolor='black', shrink=0.1)
            )
plt.show()
```



In [ ]:

In [132]:
```
from sklearn.metrics import silhouette_score

k = 4 ## arbitrary value
kmeans = KMeans(n_clusters=k,random_state = 42)
kmeans.fit(df_scaled)
```

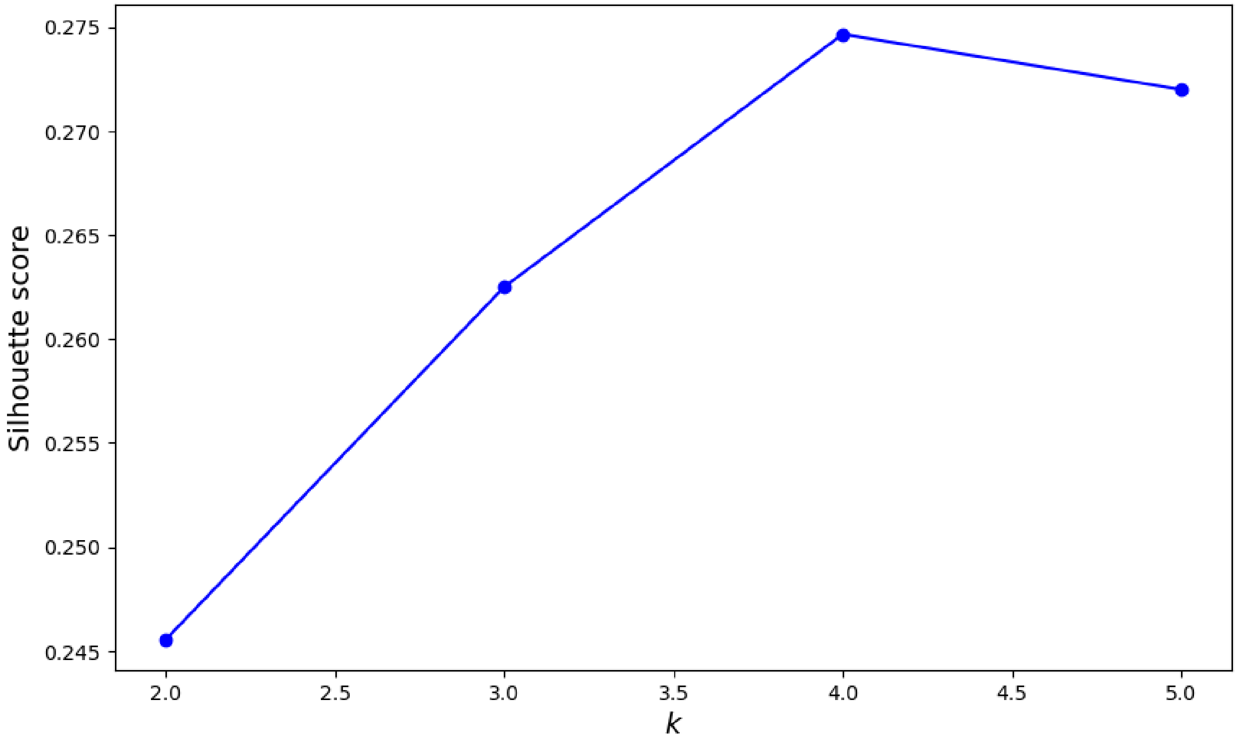
Out[132]: KMeans(n_clusters=4, random_state=42)

In [135]:
```
## silhouette score for 4 clusters
silhouette_score(df_scaled, kmeans.labels_)
```

Out[135]: 0.27465347321981937

In [139]:
```
## plot for different values of K
kmeans_per_k_for_sil = [KMeans(n_clusters=k, random_state=42).fit(df_scaled)
                for k in range(2, 6)]

silhouette_scores = [silhouette_score(df_scaled, model.labels_)
                     for model in kmeans_per_k_for_sil]
```

In [140]:
```
plt.figure(figsize=(10, 6))
plt.plot(range(2, 6), silhouette_scores, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
plt.show()
```



- We should pick 3 or 4 because after 4 there is a significant drop in the scores.
- According to Elbow curve I wanted to pick 3, but according to Silhouette we can pick 4.

Performing Agglomerative Clustering

In [106]:
```
sample_size = int(0.1 * df_scaled.shape[0])  # 10% of the dataset

# Sample the data
np.random.seed(42)
sample_indices = np.random.choice(df_scaled.shape[0], sample_size, replace=False)
df_sampled = df_scaled[sample_indices]
```

In [107]:
```
df_sampled.shape
```

Out[107]:
```
(17951, 6)
```

In [108]:
```
# import hierarchical clustering libraries
import scipy.cluster.hierarchy as sch

Z = sch.linkage(df_sampled, method='ward') #linkage = ward
```

In [109]:
```
Z
```

Out[109]:
```
array([[1.00000000e+00, 1.09590000e+04, 0.00000000e+00, 2.00000000e+00],
       [1.42440000e+04, 1.71890000e+04, 0.00000000e+00, 2.00000000e+00],
       [1.35420000e+04, 1.64730000e+04, 0.00000000e+00, 2.00000000e+00],
       ...,
       [3.58920000e+04, 3.58960000e+04, 2.77409171e+01, 1.10420000e+04],
       [3.58930000e+04, 3.58970000e+04, 4.13171984e+01, 6.90900000e+03],
       [3.58980000e+04, 3.58990000e+04, 6.71626985e+01, 1.79510000e+04]])
```

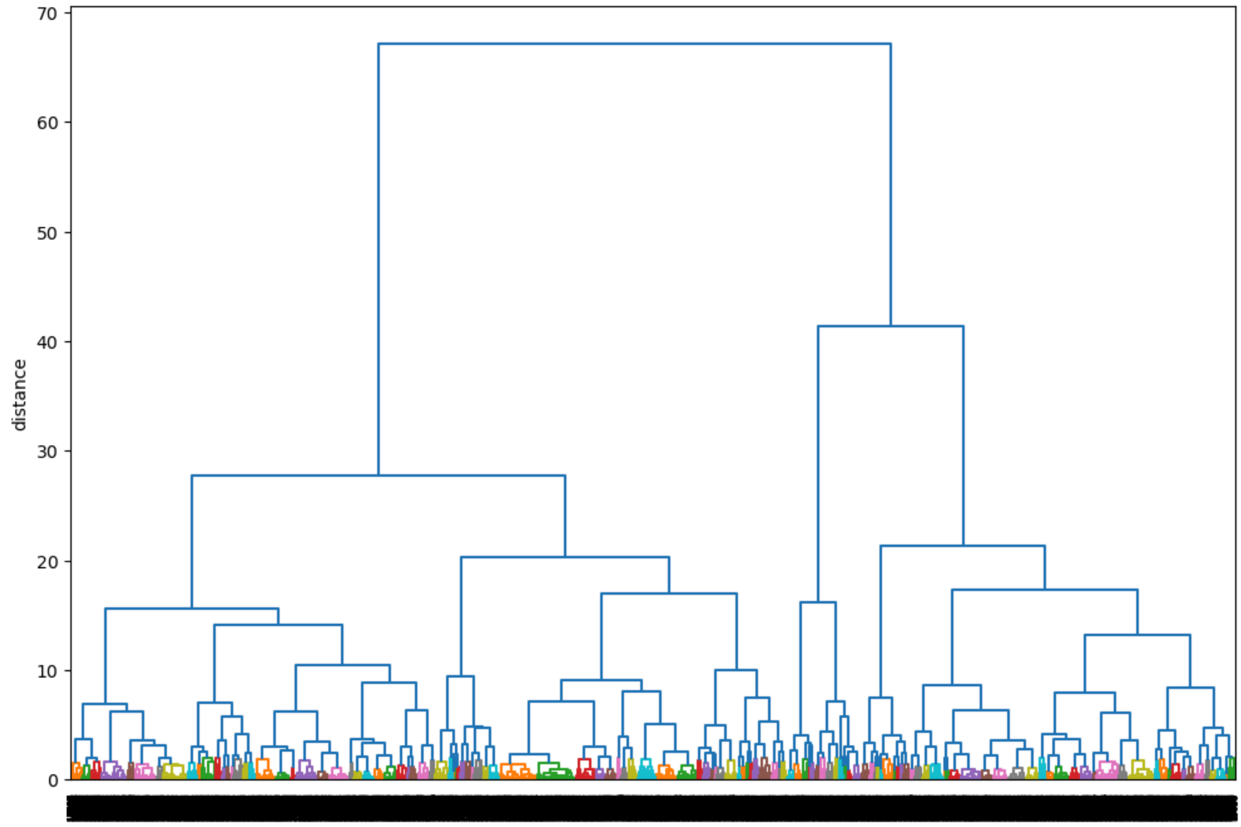
- The first 2 columns are cluster name.
- 3rd column is the distance between them
- 4th column tell you no of data points inside that cluster

In [110]:
```
Z.shape
```

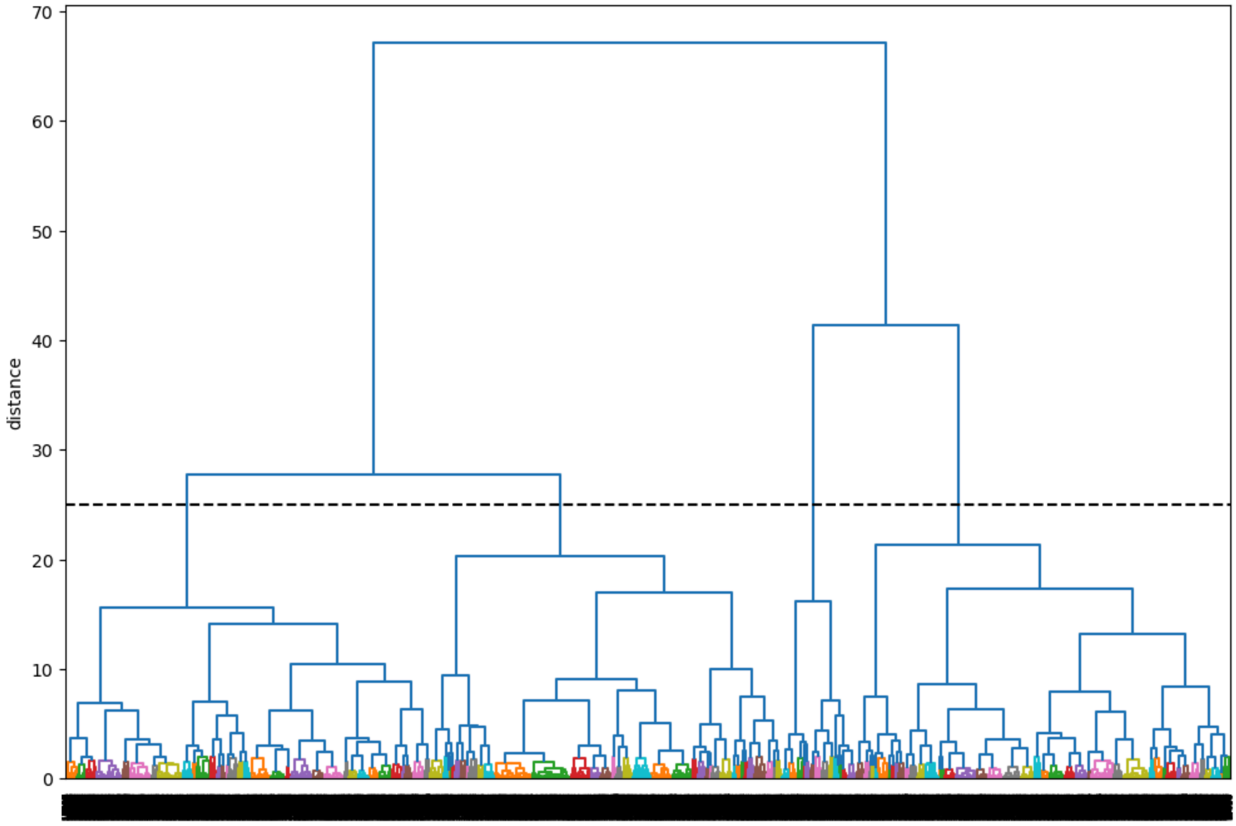
Out[110]:
```
(17950, 4)
```

In [111]:
```
df_sampled = pd.DataFrame(df_sampled, columns = df.columns)
```

In [112]:
```
fig, ax = plt.subplots(figsize=(12, 8))
sch.dendrogram(Z, labels=df_sampled.index, ax=ax, color_threshold=2)
plt.xticks(rotation=90)
ax.set_ylabel('distance')
plt.show()
```



In [113]:
```
fig, ax = plt.subplots(figsize=(12, 8))
sch.dendrogram(Z, labels=df_sampled.index, ax=ax, color_threshold=2)
plt.xticks(rotation=90)
plt.axhline(y=25, color='k', linestyle='--')
ax.set_ylabel('distance')
plt.show()
```



By taking a threshold value of 25 on the Y_axis (Distance/dissimilarity), the appropriate number of clusters is 4

Performing Agglomerative Clustering with 4 clusters

In [119]:
```
# import hierarchical clustering libraries
from sklearn.cluster import AgglomerativeClustering


# create clusters
hr_clust = AgglomerativeClustering(n_clusters=4, affinity = 'euclidean', linkage = 'ward')
y_pred = hr_clust.fit_predict(df_sampled)
```

In [120]:
```
y_pred
```

Out[120]:
```
array([2, 1, 2, ..., 1, 2, 0], dtype=int64)
```

In [121]:
```
df_sampled = pd.DataFrame(df_sampled, columns = df.columns)
df_sampled['Y_Predicted'] = y_pred
```

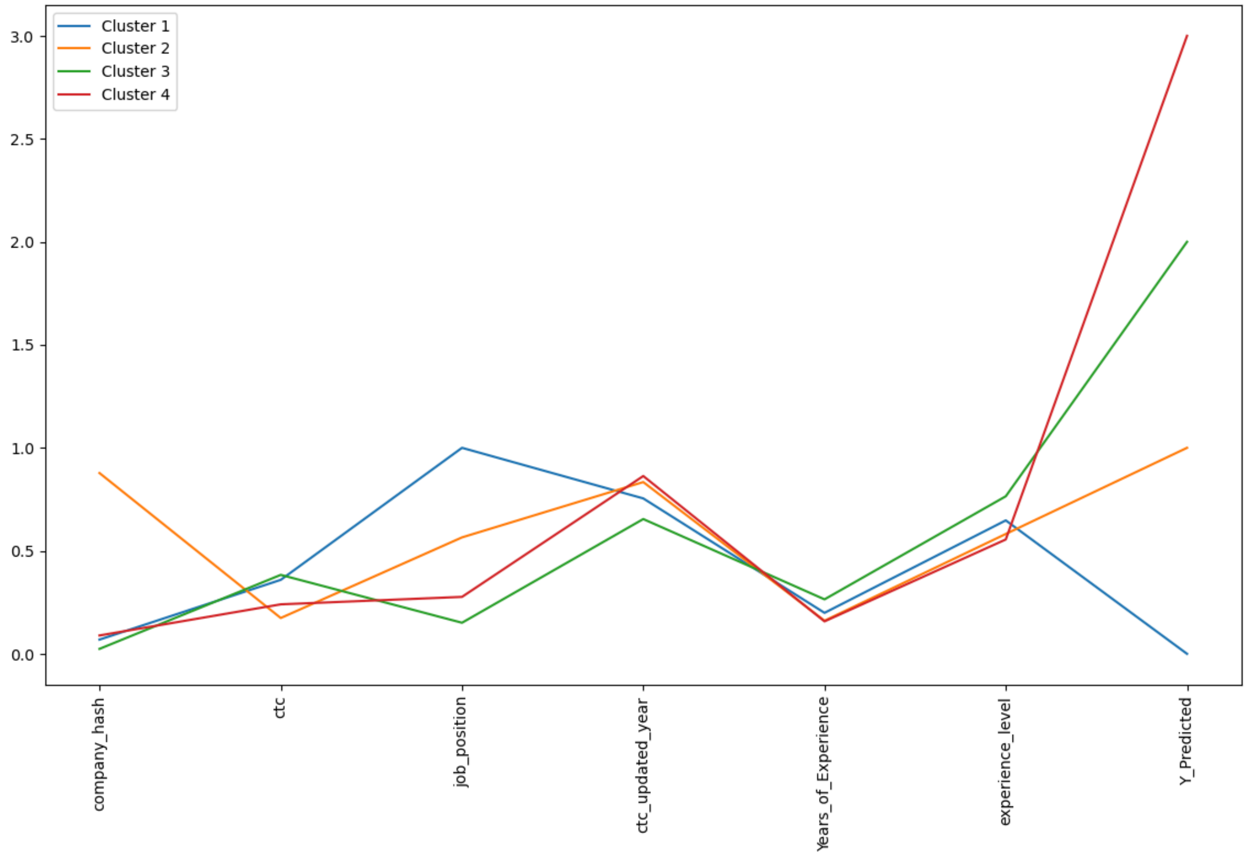
In [124]:
```
df_sampled.head()
```

Out[124]:

	company_hash	ctc	job_position	ctc_updated_year	Years_of_Experience	experience_level	Y_Predicted
0	0.001031	0.580174	0.082949	0.666667	0.372093	1.00	2
1	1.000000	0.104956	1.000000	0.833333	0.116279	0.50	1
2	0.015331	0.396501	0.043721	0.666667	0.279070	0.75	2
3	0.000902	0.481049	0.398229	0.833333	0.255814	0.75	3
4	0.000000	0.087463	0.398229	0.666667	0.186047	0.75	2

In [126]:
```
#Plot a line graph to see the characteristics of the clusters
df_sampled['label'] = pd.Series(y_pred, index=df_sampled.index)

clustered_df = df_sampled.groupby('label').mean()

labels = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4']

plt.figure(figsize=(14,8))
plt.plot(clustered_df.T, label=labels)
plt.xticks(rotation=90)
plt.legend(labels)
```

Out[126]:
```
<matplotlib.legend.Legend at 0x24b530687f0>
```



Looking at this, What characteristics do we find unique in each of these learned cluster?

- **Cluster 1** - Learners with high ctc, recently updated ctc, and good experience_level
- **Cluster 2** - Learners with lowest ctc, even after most recently updated ctc
- **Cluster 3** - Learners with highest ctc
- **Cluster 4** - Learners with mid range ctc, low Years_of_experience and esperience_level

This way, with the help of Hierarchical Clustering, we can draw conclusions on how different data points are grouped into different clusters, and also get information about the features of the dataset based on which the grouping is done.

In [128]:
```
from sklearn.metrics import silhouette_score

# Hierarchical clustering
def hierarchical_clustering_silhouette(df_sampled, n_clusters):
    clustering = AgglomerativeClustering(n_clusters=n_clusters)
    labels = clustering.fit_predict(df_sampled)
    silhouette_avg = silhouette_score(df_sampled, labels)
    return silhouette_avg

# Evaluate silhouette score for different numbers of clusters
for n_clusters in range(2, 6):
    silhouette_avg = hierarchical_clustering_silhouette(df_sampled, n_clusters)
    print(f"For n_clusters = {n_clusters}, the average silhouette_score is : {silhouette_avg}")
```

```
For n_clusters = 2, the average silhouette_score is : 0.7086807266184053
For n_clusters = 3, the average silhouette_score is : 0.6796063614945664
For n_clusters = 4, the average silhouette_score is : 0.7003990507158994
For n_clusters = 5, the average silhouette_score is : 0.5907632501157801
```

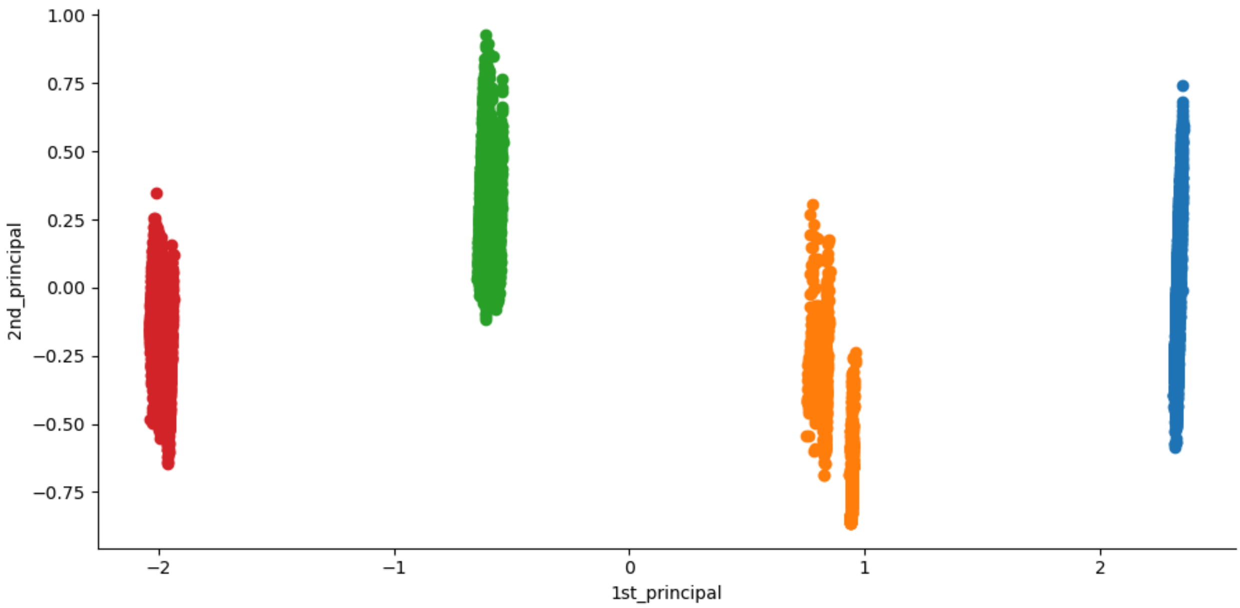
Observation :

- 42 & 4 cluster is giving us the best silhouette score

PCA Visualisation of Agglomerative Clustering

In [130]:
```
# PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(df_sampled)

# Attaching labels for data points
labels = df_sampled['Y_Predicted']

# Create DataFrame
pca_df = pd.DataFrame(data=np.vstack((pca_data.T, labels)).T, columns=("1st_principal", "2nd_principal", "label"))
```

In [131]:
```
# Visualization
g = sns.FacetGrid(pca_df, hue="label",height=5 ,aspect= 2)
g.map(plt.scatter, '1st_principal', '2nd_principal')
plt.show()
```



In [ ]:

Recommendations

1. No so much focus neede on learners in Cluster 4 & 5 as they have the highest CTC and experience_level indicating high-value employees who can take care of themselves in finding jobs.
2. Provide additional training and development opportunities for employees in Cluster 3 to improve their performance and CTC.
3. Identify the key factors contributing to high CTC in Cluster 4 & 5 and replicate these practices across other clusters.
4. For top companies, implement targeted retention strategies to maintain competitive advantage.
5. Review compensation packages for the bottom 10 employees in Cluster 3 to ensure alignment with industry

In [ ]: