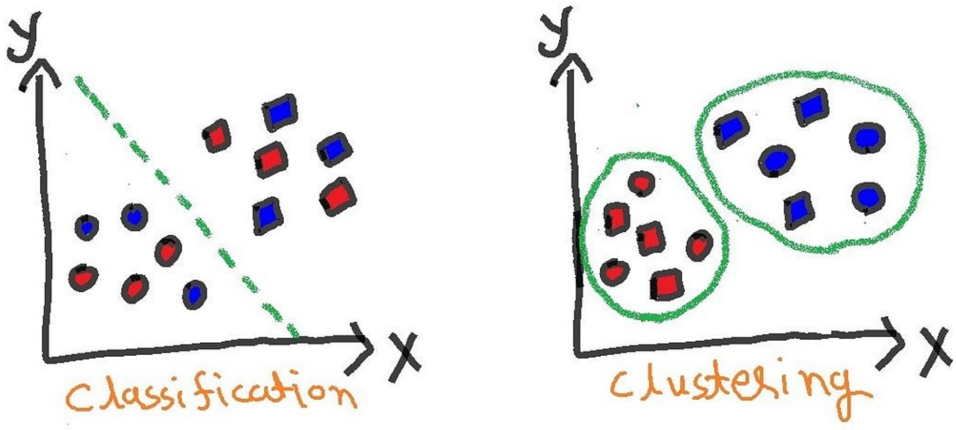


Clustering in Machine Learning (ML) is an unsupervised learning technique used to group similar data points into clusters, where points in the same cluster are more similar to each other than to those in other clusters. It’s a fundamental method for understanding the underlying structure in a dataset without predefined labels.

How Clustering Works

1. The algorithm takes a dataset with no labels.
2. It measures the similarity or distance between data points.
3. It groups data points into clusters based on a defined similarity metric.

Introduction

Scaler is an online tech-versity offering intensive computer science & Data Science courses through live classes delivered by tech leaders and subject matter experts. The meticulously structured program enhances the skills of software professionals by offering a modern curriculum with exposure to the latest technologies. It is a product by InterviewBit.

Working as a data scientist with the analytics vertical of Scaler, focused on profiling the best companies and job positions to work for from the Scaler database. Provided with the information for a segment of learners and tasked to cluster them on the basis of their job profile, company, and other features. Ideally, these clusters should have similar characteristics.

Problem statement

Assuming you're a data scientist at Scaler, you're tasked with the responsibility of analyzing the dataset to profile the best companies and job positions from Scaler's database. Your primary goal is to execute clustering techniques, evaluate the coherence of your clusters, and provide actionable insights for enhanced learner profiling and course tailoring.

Know the data

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')


from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
```

In [2]:
```
#importing the dataset
df = pd.read_csv("scaler_clustering.csv")
```

```
df.head()
```

Out[2]:

	Unnamed: 0	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_yea
0	0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000	Other	2020
1	1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999	FullStack Engineer	2019
2	2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000000	Backend Engineer	2020
3	3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700000	Backend Engineer	2019
4	4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400000	FullStack Engineer	2019

In [3]:

```
#last few rows
df.tail()
```

Out[3]:

	Unnamed: 0	company_hash	email_hash	orgyear	ctc	job_position	ctc_update
205838	206918	vuurt xzw	70027b728c8ee901fe979533ed94ffda97be08fc23f33b...	2008.0	220000	NaN	
205839	206919	husqvawgb	7f7292ffad724ebbe9ca860f515245368d714c84705b42...	2017.0	500000	NaN	
205840	206920	vwwgrxnt	cb25cc7304e9a24facda7f5567c7922ffc48e3d5d6018c...	2021.0	700000	NaN	
205841	206921	zgn vuurxwvmrt	fb46a1a2752f5f652ce634f6178d0578ef6995ee59f6c8...	2019.0	5100000	NaN	
205842	206922	bgqsvz onvzrtj	0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f...	2014.0	1240000	NaN	

In [4]:

```
#columns
df.columns
```

Out[4]:

```
Index(['Unnamed: 0', 'company_hash', 'email_hash', 'orgyear', 'ctc',
       'job_position', 'ctc_updated_year'],
      dtype='object')
```

In [5]:

```
#basic information about the columns
df.info()
```

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

Data Dictionary

Feature	Description
'Unnamed 0'	Index of the dataset.
Email_hash	Anonymised Personal Identifiable Information (PII).
Company_hash	This represents an anonymized identifier for the company, which is the current employer of the learner..
orgyear	Employment start date.
CTC	Current CTC.
Job_position	Job profile in the company.
CTC_updated_year	Year in which CTC got updated (Yearly increments, Promotions).

In [6]:

```
#shape of the dataset
df.shape
```

Out[6]:

```
(205843, 7)
```

In [7]:
```
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns in the dataset")
```
There are 205843 rows and 7 columns in the dataset

In [8]:
```
df.describe(exclude = ['float64','int64']).T
```
Out[8]:

	count	unique	top	freq
company_hash	205799	37299	nvnv wgzohrnvzwj otqcxwto	8337
email_hash	205843	153443	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	10
job_position	153279	1016	Backend Engineer	43554

Backend Engineer is the most frequent job position in our dataset

In [9]:
```
#missing values
df.isnull().sum()
```
Out[9]:

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

dtype: int64

In [10]:
```
#duplicates
df.duplicated().sum()
```
Out[10]: 0

There are no duplicates in the dataset

In [ ]:

□Data Preprocessing

- Remove Special characters using Regex
- Checking Duplicates and Treatment
- Checking Null Values and Treatment using KNN imputation for Numerical attributes
- Data Cleaning
- Feature Engineering
- Checking Outliers and Treatment using Capping

In [11]:
```
#creating a copy of the dataset for further Data Processing
df1 = df.copy()
```

Removing Special characters using Regex

In [12]:
```
import re  #regular expressions

#creating a function to remove special characters
def remove_special_char(s) :
  if not isinstance(s, str) :
    s = str(s)                #Ensures the input s is converted to a string if it is not already a string.
  return re.sub('[^A-Za-z0-9 ]+', '',s)


#applying this funtion to the 'company_hash' column
df1['company_hash'] = df1['company_hash'].apply(remove_special_char)
```

In [13]:
```
df1['company_hash'].head()
```

Out[13]:

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

dtype: object

Duplicate checking

- Based on entire dataset
- Based on email_hash column (inorder to ensure uniqueness of learner's data)

In [14]:
```
#on entire dataset
df1.duplicated().any()
```

Out[14]: False

In [15]:
```
#based on email_hash
df1.duplicated(subset = ['email_hash']).any()
```

Out[15]: True

In [16]:
```
df1['email_hash'].value_counts()
```

Out[16]:

email_hash	count
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b	10
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c	9
298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee	9
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378	9
b4d5afa09bec8689017d8b29701b80d664ca37b83cb883376b2e95191320da66	8
...	...
bb2fe5e655ada7f7b7ac4a614db0b9c560e796bdfcaa4e5367e69eedfea93876	1
d6cdef97e759dbf1b7522babccbbbd5f164a75d1b4139e02c945958720f1ed79	1
700d1190c17aaa3f2dd9070e47a4c042ecd9205333545dbfaee0f85644d00306	1
c2a1c9e4b9f4e1ed7d889ee4560102c1e2235b2c1a0e59cea95a6fe55c658407	1
0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f7e738a6a87d3712c31	1

153443 rows × 1 columns

dtype: int64

In [17]:
```
display(df1[df1['email_hash'] == 'bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b'])
```

	Unnamed: 0	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_
24109	24129	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	NaN	20
45984	46038	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	Support Engineer	20
72315	72415	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	Other	20
102915	103145	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	FullStack Engineer	20
117764	118076	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	Data Analyst	20
121483	121825	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	Other	20
124476	124840	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	Support Engineer	20
144479	145021	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	FullStack Engineer	20
152801	153402	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	Devops Engineer	20
159835	160472	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	NaN	20

Insights

- Apparently for single email hash id there exists multiple rows with same joining dates and company but different job positions, this couldn't be possible.
- There are duplicate entries in the email_hash column

In [18]:
```
#to remove those rows
combined_duplicates = df[df.duplicated(subset = ['email_hash'], keep = False)]
#number of duplicates based on the email_hash column

print(f"Number of combined duplicates: {combined_duplicates.shape[0]}")
```

```
Number of combined duplicates: 93616
```

In [19]:
```
#few rows are
print(combined_duplicates.head())
```

```
   Unnamed: 0                          company_hash  \
0           0                        atrgxnnt xzaxv
1           1              qtrxvzwt xzegwgbb rxbxnta
2           2                         ojzwnvwnxw vx
4           4                            qxen sqghu
5           5  yvuuxrj hzbvqqxta bvqptnxzs ucn rna

                                        email_hash  orgyear      ctc  \
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...   2016.0  1100000
1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...   2018.0   449999
2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...   2015.0  2000000
4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...   2017.0  1400000
5  18f2c4aa2ac9dd3ae8ff74f32d30413f5165565b90d8f2...   2018.0   700000

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

In [20]:
```
#removing those rows
df1 = df1.drop_duplicates(subset = ['email_hash'], keep = 'last')
```

In [21]:
```
df1.shape
```

Out[21]: (153443, 7)

- The number of rows has decreased
- Removed Duplicates based on Email_hash
- Multiple individuals can be associated with the same company, so using company_hash alone may not ensure the uniqueness of learners
- Could have considered both Email and Company for duplicate. This would have ensured that we don't remove valid records where the same individual is associated with multiple companies. But with limited information currently going ahead with removing duplicates based on Email_hash to ensure uniqueness of each learner's data

In [ ]:

Unique values in each feature

In [22]:
```
#categorical columns
cat_cols = df1.select_dtypes(include = 'object').columns
cat_cols
```

Out[22]:
```
Index(['company_hash', 'email_hash', 'job_position'], dtype='object')
```

In [23]:
```
for i in cat_cols :
  print(f"There are {df1[i].nunique()} unique values in {i}")
  print(f"Value count in the {i} column are :-\n {df1[i].value_counts()}")
  print()
  print("-"*100)
```

```
There are 36366 unique values in company_hash
Value count in the company_hash column are :-
 company_hash
nvnv wgzohrnvzwj otqcxwto         5336
xzegojo                           3526
vbvkgz                            2440
wgszxkvzn                         2199
zgn vuurxwvmrt vwwghzn            2192
                                  ...
bgovbmtt                             1
wrxd wvuxnvr otqcxwto ucn rna        1
vxs mhoxztoo ogrhnxgzo ucn rna       1
uhroho                               1
bvptbjnqxu td vbvkgz                 1
Name: count, Length: 36366, dtype: int64

----------------------------------------------------------------------------------------------------
There are 153443 unique values in email_hash
Value count in the email_hash column are :-
 email_hash
effdede7a2e7c2af664c8a31d9346385016128d66bbc58a44274d5d6876dfec7    1
d8e8d73114617d98f7b647d6a2943983564978c3999509d98d6d5142714c7958    1
0a2c6b808187b21a9ab6b27f2365dc315cd4f64c5c908f12e387303a340dbd9a    1
77e5e9c3b29bef911b71f1ec28753029aad23172f0f6fb5ffdbfb47e086f9148    1
9ec7bc44fb8497e552087e27b3f264773c11f91e67a615d17be2a6112dad8743    1
                                                                   ..
a92418070ae61c6a53ac35b2f5748e95a0a7e5ee823cee3f4f3cf5ce82702bf8    1
65a8bb70616d56b0c8a57c229716cee6e8ee9bd690cc26203693858175313449    1
2b166c2eefe21566e54403538a39e65851cfdd64a51a3bdf1f48476d0a5ce11e    1
28992538aebfd55c662be0ef06c7a5ec32d85de4e260ef12ddddcf43066d6e29    1
0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f7e738a6a87d3712c31    1
Name: count, Length: 153443, dtype: int64

----------------------------------------------------------------------------------------------------
There are 651 unique values in job_position
Value count in the job_position column are :-
 job_position
Backend Engineer                 33154
FullStack Engineer               17460
Other                            13747
Frontend Engineer                 8154
Engineering Leadership            5987
                                 ...
system software engineer             1
Pop engineer                         1
Senior Web Developer                 1
Messenger come driver                1
Android Application developer        1
Name: count, Length: 651, dtype: int64

----------------------------------------------------------------------------------------------------
```

Insights

- Since company and email are hashed so not much information can be inferred. However, we can conveniently identify contribution of each element to the Feature
- Since email_hash is unique for every learner so it is correctly showing frequency as 1 for each email
- Backend Engineer is the most frequent with 33154 value counts followed by FullStack Engineer and Other

In [24]:
```
#Numerical columns
num_cols = df1.select_dtypes(include = "number").columns
num_cols
```

Out[24]:
```
Index(['Unnamed: 0', 'orgyear', 'ctc', 'ctc_updated_year'], dtype='object')
```

In [25]:
```
for i in num_cols :
  print(f"There are {df1[i].nunique()} unique values in {i}")
  print(f"Value count in the {i} column are :-\n {df1[i].value_counts(normalize = True)}")
  print()
  print("-"*100)
```

```
There are 153443 unique values in Unnamed: 0
Value count in the Unnamed: 0 column are :-
 Unnamed: 0
3         0.000007
150149    0.000007
150151    0.000007
150152    0.000007
150153    0.000007
            ...
82865     0.000007
82866     0.000007
82867     0.000007
82868     0.000007
206922    0.000007
Name: proportion, Length: 153443, dtype: float64

----------------------------------------------------------------------------------------------------
There are 76 unique values in orgyear
Value count in the orgyear column are :-
 orgyear
2016.0    0.112993
2018.0    0.109661
2017.0    0.107985
2015.0    0.104249
2019.0    0.098022
            ...
2106.0    0.000007
1973.0    0.000007
209.0     0.000007
208.0     0.000007
200.0     0.000007
Name: proportion, Length: 76, dtype: float64

----------------------------------------------------------------------------------------------------
There are 3299 unique values in ctc
Value count in the ctc column are :-
 ctc
600000     0.036300
1000000    0.033830
400000     0.032422
800000     0.030637
500000     0.030448
             ...
449000     0.000007
1386000    0.000007
2301000    0.000007
1023000    0.000007
3327000    0.000007
Name: proportion, Length: 3299, dtype: float64

----------------------------------------------------------------------------------------------------
There are 7 unique values in ctc_updated_year
Value count in the ctc_updated_year column are :-
 ctc_updated_year
2019.0    0.376166
2021.0    0.245335
2020.0    0.237652
2017.0    0.046825
2018.0    0.043228
2016.0    0.033452
2015.0    0.017342
Name: proportion, dtype: float64

----------------------------------------------------------------------------------------------------
```

The 'Unnamed: 0' columns doesn't give us any information, so we can drop it

In [26]:
```
#dropping column
df1 = df1.drop('Unnamed: 0', axis = 1)
```

Insights

- ctc_updated_year to be converted to datetime datatype for further analysis
- orgyear is the starting year of employment. We could identify many invalid entries like 200,208,2107 ..which are not valid years. This column will undergo treatment
- Maximum Learners have got CTC of 6 Lac followed by 10 Lac and 4 Lac

In [27]:
```
#converting some columns to datetime
df1["ctc_updated_year"] = pd.to_datetime(df1["ctc_updated_year"], format = "%Y")
df1["ctc_updated_year"] = df1["ctc_updated_year"].dt.year
```

In [ ]:

Converting the feature "orgyear" to datetime

- This columns represents the year the learner began employment at the current company.
- We could identify many invalid entries like 200,208,2107 ..which are not valid years.

In [28]:
```
# Define the range of valid years (e.g., 1900 to 2023)
valid_years = range(1900, 2024)

# Replace invalid years with NaN
df1['orgyear'] = df1['orgyear'].apply(lambda x: x if x in valid_years else np.nan)

# Convert valid years to datetime
df1['orgyear'] = pd.to_datetime(df1['orgyear'].dropna().astype(int), format='%Y')
df1['orgyear'] = df1['orgyear'].dt.year
df1['orgyear'] = df1['orgyear'].astype('Int64')
```

- We changed the column into an 'Int64' datatype
- Converted the column to the nullable integer type (Int64) to accommodate NaN values and maintain compatibility with pandas.

In [29]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 153443 entries, 3 to 205842
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   company_hash      153443 non-null  object
 1   email_hash        153443 non-null  object
 2   orgyear           153257 non-null  Int64 
 3   ctc               153443 non-null  int64 
 4   job_position      119252 non-null  object
 5   ctc_updated_year  153443 non-null  int32 
dtypes: Int64(1), int32(1), int64(1), object(3)
memory usage: 7.8+ MB
```

Missing value treatment (using KNN)

- For Numerical values we can use KNN Imputation

In [30]:
```
#missing values
df1.isna().sum()
```

Out[30]:

	0
company_hash	0
email_hash	0
orgyear	186
ctc	0
job_position	34191
ctc_updated_year	0

dtype: int64

In [31]:
```
#Imputing missing values in job_position column with 'unknown' since it is categorical column
```

```
df1['job_position'].fillna('unknown', inplace = True)
```

In [32]:
```
#impututing missing values in the 'orgyear' using KNNImputer(numerical variable)
from sklearn.impute import KNNImputer

#creating instance
knn_imp = KNNImputer(n_neighbors=3)

num_col = ['orgyear']
df1[num_col] = knn_imp.fit_transform(df1[num_col])
```

In [33]:
```
#converting 'orgyear' back to int
df1['orgyear'] = df1['orgyear'].astype(int)
```

In [34]:
```
df1.isna().sum()
```

Out[34]:

	0
company_hash	0
email_hash	0
orgyear	0
ctc	0
job_position	0
ctc_updated_year	0

dtype: int64

- There are no more missing values
- For job_position, filled missing values with the string 'unknown'. This ensures that the analysis considers these entries as a separate category rather than skewing the distribution of existing categories. This approach maintains the integrity of the categorical data while addressing missing values in a straightforward and non-biased manner.
- Applied KNN imputation to fill missing values in the orgyear column. This leverages the relationships between existing data points to predict the missing values.

Feature Engineering

- Using the 'orgyear' to compute Years of Experience(YOE) by subtracting from the current year. This can provide better insights than just the starting year of employment

In [35]:
```
from datetime import datetime

#getting the curent year
current_year = datetime.now().year

#creating new feature 'YOE'
df1['YOE'] = current_year - df1['orgyear']
```

In [36]:
```
df1.head()
```

Out[36]:

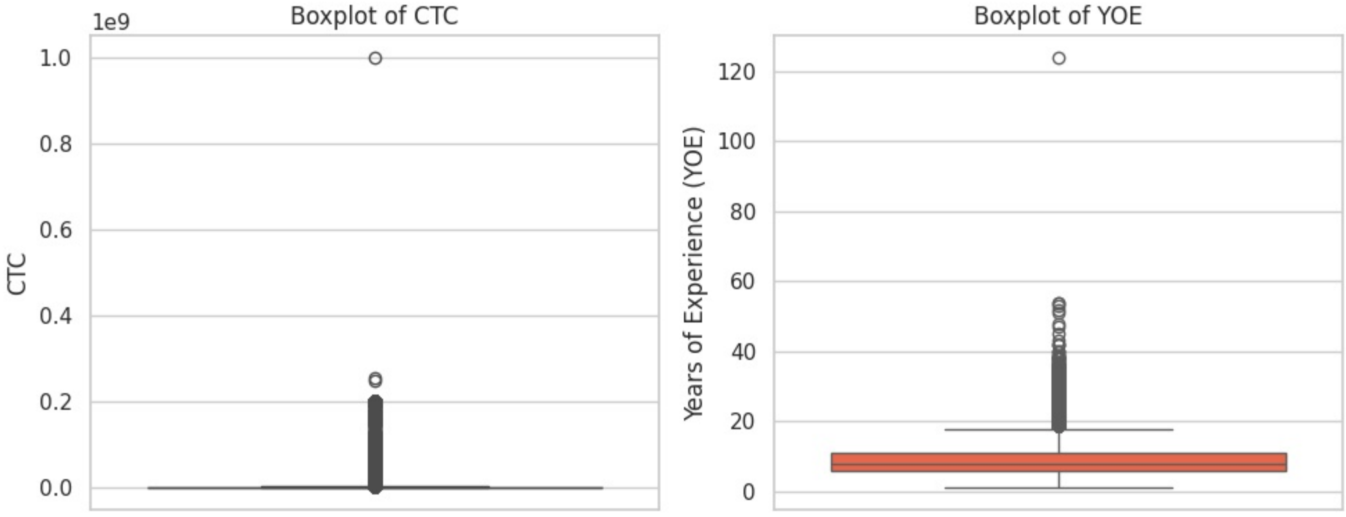
	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year	YOE
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	700000	Backend Engineer	2019	7
7	vwwtznhqt ntwyzgrgsj	756d35a7f6bb8ffeaffc8fcca9ddbb78e7450fa0de2be0...	2019	400000	Backend Engineer	2019	5
9	xrbhd	b2dc928f4c22a9860b4a427efb8ab761e1ce0015fba1a5...	2019	360000	unknown	2019	5
13	wgszxkvzn	134cc4a76a119493d523f1855a3b7106f64287455d5cd4...	2016	440000	Data Analyst	2020	8
14	xznhxn	ebcaf397ef5084e05889a6e9a0c3f96a5c8fb0b16749ce...	2016	440000	Backend Engineer	2019	8

Outlier detection and treatment

In [37]:
```
df2=df1.copy()
```

In [38]:
```
# Set the style of seaborn
sns.set(style="whitegrid")

# Create a figure with two subplots for CTC and YOE
```

```
fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize=(10, 4))

# Boxplot for CTC
sns.boxplot(data=df2, y='ctc', ax=ax[0])
ax[0].set_title('Boxplot of CTC')
ax[0].set_ylabel('CTC')

# Boxplot for YOE
sns.boxplot(data=df2, y='YOE', ax=ax[1],color='#FF5733')
ax[1].set_title('Boxplot of YOE')
ax[1].set_ylabel('Years of Experience (YOE)')

# Show the plots
plt.tight_layout()
plt.show()
```



- We can clearly observe outliers in ctc and YOE
- Outliers can significantly impact the performance and results of clustering algorithms like K-means and hierarchical clustering

Outlier treatment

- Using capping method
- This approach reduces the impact of extreme outliers without completely removing data points.

In [39]:
```
# Calculate upper bound for ctc using 99th percentile
ctc_upper_bound = df2['ctc'].quantile(0.99)

# Apply clipping to ctc column
df2['ctc_capped'] = np.clip(df2['ctc'], df2['ctc'].min(), ctc_upper_bound)

# Calculate upper bound for YOE using 99th percentile
yoe_upper_bound = df2['YOE'].quantile(0.99)

# Apply clipping to YOE column
df2['YOE_capped'] = np.clip(df2['YOE'], df2['YOE'].min(), yoe_upper_bound)
```

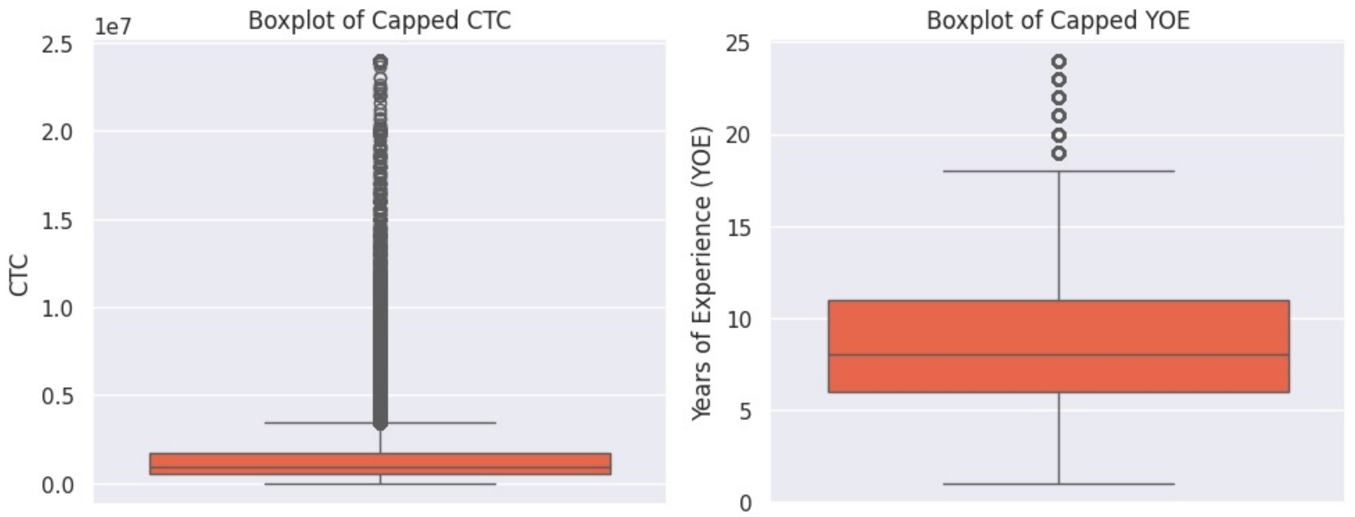
- The 99th percentile value of the ctc column is calculated. This value represents the threshold below which 99% of the data falls.
- Extreme outliers (the top 1% of values) are identified for capping.
- np.clip Function: Limits each value in the ctc column to lie within a specified range.
  - Lower Bound: df2['ctc'].min() ensures the smallest ctc value remains unchanged.
  - Upper Bound: ctc_upper_bound caps values exceeding the 99th percentile.
- The results are stored in a new column, ctc_capped, preserving the original ctc column for comparison

Similarly we did for the 'YOE' column

After clipping

In [40]:
```
# Set the style of seaborn
sns.set(style="darkgrid")

# Create a figure with two subplots for CTC and YOE
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

# Boxplot for CTC post Capping
```

```
sns.boxplot(data=df2, y='ctc_capped', ax=ax[0],color='#FF5733')
ax[0].set_title('Boxplot of Capped CTC')
ax[0].set_ylabel('CTC')

# Boxplot for YOE post Capping
sns.boxplot(data=df2, y='YOE_capped', ax=ax[1],color='#FF5733')
ax[1].set_title('Boxplot of Capped YOE')
ax[1].set_ylabel('Years of Experience (YOE)')

# Show the plots
plt.tight_layout()
plt.show()
```



- We can observe that extreme outliers have been treated with capping method.
- Setting the upper percentile to 99% is a way to include most of the data points while excluding the extreme 1% of outliers that are far from the rest of the data.
- This approach ensured that the majority of data remains intact, while the extreme values that could significantly impact the clustering results are capped.(1e9 range changed to 1e7)
- Created two new capped columns while keeping the original columns if needed to refer further

In [41]:
```
df2.head()
```

Out[41]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year	YOE	ctc
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	700000	Backend Engineer	2019	7	
7	vwwtznhqt ntwyzgrgsj	756d35a7f6bb8ffeaffc8fcca9ddbb78e7450fa0de2be0...	2019	400000	Backend Engineer	2019	5	
9	xrbhd	b2dc928f4c22a9860b4a427efb8ab761e1ce0015fba1a5...	2019	360000	unknown	2019	5	
13	wgszxkvzn	134cc4a76a119493d523f1855a3b7106f64287455d5cd4...	2016	440000	Data Analyst	2020	8	
14	xznhxn	ebcaf397ef5084e05889a6e9a0c3f96a5c8fb0b16749ce...	2016	440000	Backend Engineer	2019	8	

In [42]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 153443 entries, 3 to 205842
Data columns (total 9 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   company_hash      153443 non-null  object
 1   email_hash        153443 non-null  object
 2   orgyear           153443 non-null  int64 
 3   ctc               153443 non-null  int64 
 4   job_position      153443 non-null  object
 5   ctc_updated_year  153443 non-null  int32 
 6   YOE               153443 non-null  int64 
 7   ctc_capped        153443 non-null  int64 
 8   YOE_capped        153443 non-null  int64 
dtypes: int32(1), int64(5), object(3)
memory usage: 11.1+ MB
```

In [ ]:

Manual Clustering

Steps :

- Creating Designation Flag & Insights
- Creating Class Flag & Insights
- Creating Tier Flag & Insights

In [43]:
```
#creating a copy
df3 = df2.copy()
```

We can now drop the original 'ctc' and 'YOE' columns since we have new capped columns 'ctc_capped' and 'YOE_capped'

- Analysis will be more consistent and robust when performed on a dataset where extreme values have been controlled or standardized through capping.
- Capped Feature preserves the integrity of the dataset by retaining most data points while adjusting extreme values. This ensures that the analysis reflects the general trends and patterns in the data without being overly influenced by outliers.

In [44]:
```
df3.drop(columns = ['ctc', 'YOE'], inplace = True)
```

In [45]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 153443 entries, 3 to 205842
Data columns (total 7 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   company_hash      153443 non-null  object
 1   email_hash        153443 non-null  object
 2   orgyear           153443 non-null  int64 
 3   job_position      153443 non-null  object
 4   ctc_updated_year  153443 non-null  int32 
 5   ctc_capped        153443 non-null  int64 
 6   YOE_capped        153443 non-null  int64 
dtypes: int32(1), int64(3), object(3)
memory usage: 8.8+ MB
```

In [ ]:

Creating Flags:

- Designation Flag: CTC on the basis of Company, Job Position and Years of Experience
- Class Flag: CTC On the basis of Company and Job Position
- Tier Flag: CTC On the basis of Company

In [46]:
```
# Step 1:
#Group by company, job position, and years of experience (Designation)
grouped_summary_designation = df3.groupby(['company_hash', 'job_position', 'YOE_capped'])['ctc_capped'].agg(['m
grouped_summary_designation.rename(columns={'mean': 'mean_designation'}, inplace=True)

# Group by company and years of experience (Class)
grouped_summary_class = df3.groupby(['company_hash', 'job_position'])['ctc_capped'].agg(['mean', 'median', 'max
grouped_summary_class.rename(columns={'mean': 'mean_class'}, inplace=True)

# Group by company level only (Tier)
grouped_summary_tier = df3.groupby(['company_hash'])['ctc_capped'].agg(['mean', 'median', 'max', 'min', 'count'
grouped_summary_tier.rename(columns={'mean': 'mean_tier'}, inplace=True)

# Step 2:
#Merge only the mean values with the original dataset
df3 = df3.merge(grouped_summary_designation[['company_hash', 'job_position', 'YOE_capped', 'mean_designation']]
df3 = df3.merge(grouped_summary_class[['company_hash', 'job_position', 'mean_class']], on=['company_hash', 'job_
df3 = df3.merge(grouped_summary_tier[['company_hash', 'mean_tier']], on=['company_hash'])

# Step 3:
#Create flags for Designation, Class, and Tier based on mean values
def designation_flag(row):
    if row['ctc_capped'] > row['mean_designation']:
        return 3
    elif row['ctc_capped'] == row['mean_designation']:
        return 2
    else:
        return 1

def class_flag(row):
    if row['ctc_capped'] > row['mean_class']:
        return 3
    elif row['ctc_capped'] == row['mean_class']:
```

```python
        return 2
    else:
        return 1

def tier_flag(row):
    if row['ctc_capped'] > row['mean_tier']:
        return 3
    elif row['ctc_capped'] == row['mean_tier']:
        return 2
    else:
        return 1

df3['Designation_Flag'] = df3.apply(designation_flag, axis=1)
df3['Class_Flag'] = df3.apply(class_flag, axis=1)
df3['Tier_Flag'] = df3.apply(tier_flag, axis=1)

# Step 4:
#Check if columns exist before attempting to drop them
columns_to_drop = ['mean_class', 'mean_designation', 'mean_tier']
df3.drop(columns=[col for col in columns_to_drop if col in df3.columns], inplace=True)
```

Comparison Logic:

- If ctc_capped is greater than the mean (mean_designation, mean_class, or mean_tier), the function returns a flag value of 3.
- If ctc_capped is equal to the mean, the function returns a flag value of 2.
- If ctc_capped is less than the mean, the function returns a flag value of 1.

In [47]:
```python
grouped_summary_designation.head()
```

Out[47]:

	company_hash	job_position	YOE_capped	mean_designation	median	max	min	count
0	0	Other	4	100000.0	100000.0	100000	100000	1
1	0000	Other	7	300000.0	300000.0	300000	300000	1
2	01 ojztqsj	Android Engineer	8	270000.0	270000.0	270000	270000	1
3	01 ojztqsj	Frontend Engineer	13	830000.0	830000.0	830000	830000	1
4	05mz exzytvrny uqxcvnt rxbxnta	Backend Engineer	5	1100000.0	1100000.0	1100000	1100000	1

In [48]:
```python
grouped_summary_class.head()
```

Out[48]:

	company_hash	job_position	mean_class	median	max	min	count
0	0	Other	100000.0	100000.0	100000	100000	1
1	0000	Other	300000.0	300000.0	300000	300000	1
2	01 ojztqsj	Android Engineer	270000.0	270000.0	270000	270000	1
3	01 ojztqsj	Frontend Engineer	830000.0	830000.0	830000	830000	1
4	05mz exzytvrny uqxcvnt rxbxnta	Backend Engineer	1100000.0	1100000.0	1100000	1100000	1

In [49]:
```python
grouped_summary_tier.head()
```

Out[49]:

	company_hash	mean_tier	median	max	min	count
0	0	100000.0	100000.0	100000	100000	1
1	0000	300000.0	300000.0	300000	300000	1
2	01 ojztqsj	550000.0	550000.0	830000	270000	2
3	05mz exzytvrny uqxcvnt rxbxnta	1100000.0	1100000.0	1100000	1100000	1
4	1	175000.0	175000.0	250000	100000	2

In [50]:
```python
df3.head()
```

Out[50]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped	YOE_
0	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	Backend Engineer	2019	700000	
1	vwwtznhqt ntwyzgrgsj	756d35a7f6bb8ffeaffc8fcca9ddbb78e7450fa0de2be0...	2019	Backend Engineer	2019	400000	
2	xrbhd	b2dc928f4c22a9860b4a427efb8ab761e1ce0015fba1a5...	2019	unknown	2019	360000	
3	wgszxkvzn	134cc4a76a119493d523f1855a3b7106f64287455d5cd4...	2016	Data Analyst	2020	440000	
4	xznhxn	ebcaf397ef5084e05889a6e9a0c3f96a5c8fb0b16749ce...	2016	Backend Engineer	2019	440000	

We have created three new features (flags)

Let us explore some informations derived from these features

Designation flag exploration

Top 10 Employees with Designation Flag 1 (Earning More Than Most of Their Peers with Same Job Position and Experience)

In [51]:
```
top_10_designation1 = df3[df3['Designation_Flag'] == 1].nlargest(10, 'ctc_capped')
top_10_designation1
```

Out[51]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped
5984	ofxssj	e6b830e44ae282c86a370685a6e3bb3aa82ec995eec5db...	2014	Other	2020	11200000
141146	vbvkgz ftm otqcxwto	0932dc8d855953b2ac63c8046c9fb33f7f554174b6c2fe...	2013	Backend Engineer	2019	11200000
124808	sqvm	ed3b3231ac4758173e68bcde8eac3842497e153d9d1832...	2015	Engineering Leadership	2019	9200000
141694	hmtq	9885423385b89dd905f1df74a1d6e71906ccccd915c7e4...	2013	Engineering Leadership	2020	8500000
71127	xzntr wgqugqvnxgz	9aa54ea5c7e0b2567cc43718bd6516f3cfefb5622b6e2b...	2015	Other	2021	8400000
51532	fvrbvqn rvmo	9adf861294aa69336409395a5474ce6f9ffbfd38594ed4...	2010	Backend Architect	2019	8100000
149217	vba	f9530fc2d3629fc9a04c7e4e2ea6b8ddbe03eb3a97caff...	2003	Engineering Leadership	2020	8100000
2890	eqttwyvqst	28dc7d414a336ebfecf691f1db3b9cdc95b58ffede1107...	2005	Engineering Leadership	2020	7300000
40534	gnytq	4f4f4bac863dc79205345fd614a4e4cd4c99718533c60d...	2017	Data Analyst	2019	7300000
115313	sggsrt	97f2289a59953b4e94f8d2436f6edf621b9a359d919bbc...	2019	FullStack Engineer	2020	7300000

- The 'nlargest' function selects the top 10 rows from the filtered subset, based on the values in the 'ctc_capped' column.
- Similarly we can use 'nsmallest' function to select bottom top 10 rows based on 'ctc_capped' column

Bottom 10 Employees with Designation Flag 3 (Earning Less Than Most of Their Peers with Same Job Position and Experience)

In [52]:
```
bottom_10_designation3 = df3[df3['Designation_Flag'] == 3].nsmallest(10, 'ctc_capped')
bottom_10_designation3
```

Out[52]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped
73575	xzntqcxtfmxn	23ad96d6b6f1ecf554a52f6e9b61677c7d73d8a409a143...	2013	unknown	2018	14
43232	xz rgwg	66573ebeb4fcfc496d2af1548a18a62ec3a48dae59d1cc...	2016	Backend Engineer	2016	16000
59186	xmtd	792ac1d3daa5bc5fef39e3d61e0722cce004a0b81966b1...	2016	FullStack Engineer	2021	27000
47719	wgbgag	87f95061ed13da965818fded3d19249bc6d88de3b73ff2...	2014	Backend Engineer	2017	36000
94108	kvrgqv sqghu	0b1eeb6d24629a06d29fcd410c02d0f1f2577a0a050c54...	2017	Backend Engineer	2020	40000
65176	ogwxn szqvrt	38e8416bc59782b9fb60b144657130662ec8dab8094a41...	2018	Data Scientist	2021	55000
124414	cxkqn	718ad268d9c671de079ff1c55f93e91a2d06928243ad29...	2011	Backend Engineer	2020	55000
112825	zvnxgzvr wgrrtst ge xqtrvza	fb10b6e7b4fcc82e96f5a591146046c0988c23cccb8269...	2019	Other	2019	60000
146490	wtqz	217504679c19c4738eb44eacb651c80432d3a3801f62a5...	2014	FullStack Engineer	2019	65000
56743	jvzatd	2f31b0f7d87048f22a9a6eb33526325d0b3f470185652b...	2019	Backend Engineer	2020	70000

Top 10 Employees in Each Company with Designation Flag 1

In [53]:
```
top_10_each_company_designa1 = df3[df3['Designation_Flag'] == 1].groupby(['company_hash']).apply(lambda x : x.n
top_10_each_company_designa1
```

Out[53]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped	Y
0	1bs	9c02076a74a2b8a64a6e003fa0a2e4115fc717dacb3585...	2016	Backend Engineer	2020	2320000	
1	1bs	38dfe791fc911da418b67aa989a6aa7f00b8c680c6d4e1...	2015	Backend Engineer	2019	2000000	
2	1bs	c97fd1612080086b898e440529c86325ae8ddf2e9a0b60...	2015	Backend Engineer	2019	1800000	
3	1bs	7c6f711001cae257c36a621abb0b6ffa249b3d92240ee4...	2014	unknown	2017	1500000	
4	1bs	bde68bd40e5bf94d4af39e89c6fe8af4b0926e4286de55...	2017	unknown	2021	1300000	
...	...	...	...	...	...	...	
8457	zxztrtvuo	b5628c03989a151f60c89e726351817c3a62078e7c70de...	2016	FullStack Engineer	2019	575000	
8458	zxztrtvuo	41367fd92cd85ecfa2e2ce76f4ff94cde287b95df93871...	2018	Frontend Engineer	2019	570000	
8459	zxztrtvuo	f09524b67053af24c9e446c0dd4d861cf053470ceaf0c9...	2020	unknown	2020	550000	
8460	zxztrtvuo	73ed57fdb578ccb723d176b1624bb29b0e840e89ab4230...	2019	Backend Engineer	2021	520000	
8461	zxztrtvuo	f861d9f1bfee791938d90e9ad91069220eec8664b32fea...	2019	unknown	2020	500000	

8462 rows × 10 columns

In [ ]:

Class flag exploration

Top 10 employees of FullStack Engineer in each company earning more than their peers - Class 1

In [54]:
```
top_10_fs_class1 = df3[(df3['job_position'] == 'FullStack Engineer') & (df3['Class_Flag'] == 1)].groupby(['compa
                                                                                                   lambda x: x.nla

top_10_fs_class1
```

Out[54]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped	Y
0	1bs	4ccdf10738e25d4f5ac6b85572ca7454453e17c5b1091b...	2019	FullStack Engineer	2021	1350000	
1	1bs	55824c4e7df3af153fdfe867c15a599a6e86432c33f7c6...	2018	FullStack Engineer	2019	1300000	
2	1bs ntwyzgrgsxto ucn rna	4c1e4fa4b2a7ef873e1f2b7104790a2b85aa51cae54585...	2016	FullStack Engineer	2019	900000	
3	1bs ntwyzgrgsxto ucn rna	31d074dc51e6fabd2a235c23a3d9ae0e3702cf78f270e9...	2018	FullStack Engineer	2020	800000	
4	2017	03b2ac96f3c199bcf9a5b4176d63750cd522cc315537a2...	2015	FullStack Engineer	2019	380000	
...	...	...	...	...	...	...	
4004	zxzlvwvqn	e2377e7ee0d53d2e3a45b9687fdc9c08b136b1dc470806...	2017	FullStack Engineer	2020	2300000	
4005	zxzlvwvqn	e38914706e3522ee5773627abe091edd8c6596b8519a80...	2017	FullStack Engineer	2019	900000	
4006	zxztrtvuo	af742fa47c46fa167ddfaf9c22a12a31cff23717582daa...	2018	FullStack Engineer	2020	710000	
4007	zxztrtvuo	b5628c03989a151f60c89e726351817c3a62078e7c70de...	2016	FullStack Engineer	2019	575000	
4008	zxztrtvuo	0228801807a4911ebde807b5f88a273a51d92b25e6c160...	2019	FullStack Engineer	2021	500000	

4009 rows × 10 columns

Bottom 10 Employees of FullStack Engineer in Each Company Earning Less Than Their Peers - Class 3

In [55]:

```
bottom_10_fs_class1 = df3[(df3['job_position'] == 'FullStack Engineer') & (df3['Class_Flag'] == 3)].groupby(['c
                                                                                                      lambda x: x.nsma

bottom_10_fs_class1
```

Out[55]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped	Y
0	1bs	a58fadbfbc00c007dfe6e5d5891f2dda013eb5cc66552a...	2014	FullStack Engineer	2019	1600000	
1	1bs ntwyzgrgsxto ucn rna	70ba4ee689ae53a942d5a9dffe2ceae1d776ca5736e69e...	2015	FullStack Engineer	2020	2800000	
2	2017	59e55425c5c878bc984e046f7664ca70e4d0df93bb21f0...	2016	FullStack Engineer	2018	500000	
3	247 xrvm	e959c3dae7a03c57d6bf03d299e623be9f7e736184788b...	2008	FullStack Engineer	2018	2500000	
4	247vx	f8b27f9ca749c05db8ed076d13534413b63f2a2185234d...	2014	FullStack Engineer	2020	1500000	
...	...	...	...	...	...	...	
3226	zxxn ntwyzgrgsxto rxbxnta	58e652d3e06d4228be0a8ac9ef8228928628299d93795f...	2014	FullStack Engineer	2020	24000000	
3227	zxzlvwvqn	9002b19d0e582e7a807b96851505b9937bf8b696eaaa50...	2016	FullStack Engineer	2021	4650000	
3228	zxztrtvuo	650fd4e2b40bbc033df1c93c07f9b778ce8aa5d98e8292...	2016	FullStack Engineer	2019	923000	
3229	zxztrtvuo	077a6b1aa5195410e497d0fb91fe2627db85d9b9879ec7...	2016	FullStack Engineer	2020	1200000	
3230	zxztrtvuo	3879b9a1e356ed20363fffd6871207eb908b38c864a2db...	2017	FullStack Engineer	2019	1500000	

3231 rows × 10 columns

In [ ]:

Tier flag Exploration

Top 10 Employees Earning More Than Most of the Employees in the Company - Tier 1

In [56]:
```
top_10_tier1 = df3[df3['Tier_Flag'] == 1].nlargest(10, 'ctc_capped')
top_10_tier1
```

Out[56]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped	Y
63104	mvmjrgz ytvrny	c5e7360dd9c5dd31b9b4927ccccc2f3be8f6f6a5a84963...	2015	Backend Engineer	2020	17000000	
69295	aggqavoy	68f1fea4dbfb7ae2209664b93d5f57fb86912dbe516b37...	2018	Backend Engineer	2020	13500000	
3393	ho mvzp	7ffb1e475e90f5bcb65de6664f24820a0049992f50cddd...	2017	Engineering Leadership	2020	12000000	
13674	fvqsvbxzs	299864b7e8f632bfd7079ac1f97a18371f413dfb06a2dd...	2006	Devops Engineer	2020	10000000	
82719	bvqptnxzs	a53d6b54b56d30daedbfaf860cbdbbb6cc376c60832c57...	2020	Product Designer	2021	10000000	
74545	wvqttb	01a83f323a2e7dfe7561157dce0b3dd718d68511127512...	2012	Backend Engineer	2019	7200000	
50199	zxbmrt ongqvst	b6c269b356f1f7fd8d0aa23957f42d832a1de3d6c58ed3...	2006	Engineering Leadership	2021	7100000	
68395	wvqttb	0485990d28fdbb10e494793b31dd97f94c326a93c07a2d...	2014	Data Scientist	2020	7000000	
70600	zvnxgzvr vhonqvrxv mvzp	2ddbc233754a1bf09fa7e92d61a5fb8fd46f3fe7908318...	2000	Engineering Leadership	2020	7000000	
75025	bsb qtogqno xzntqzvnxgzvr	420388fd953332be671e1b0761f9af06d323382d075ecf...	2017	FullStack Engineer	2018	7000000	

The given list shows Top 10 employees details earning more than most of the employees of the company

Top 10 companies according to their CTC

- ie; list of Top 10 Companies with highest CTC

In [57]:
```
top_10_companies = df3.groupby(['company_hash'])['ctc_capped'].mean().nlargest(10).reset_index()
top_10_companies
```

Out[57]:

	company_hash	ctc_capped
0	2jghqaggq mrxav1 hzxctqoxnj	24000000.0
1	32255407428	24000000.0
2	3ow ogrhnxgz	24000000.0
3	99 mvkvq	24000000.0
4	agbtonxiht	24000000.0
5	aggovz mgmwvn xzaxv uqxcvnt rxbxnta	24000000.0
6	agyv tdnqvwg	24000000.0
7	ajzvbxnt vootno bvzvstbtzn	24000000.0
8	ajzvbxw oxszvr	24000000.0
9	al	24000000.0

Top 2 Positions in Every Company Based on Their CTC

In [58]:
```
top_2_postions_per_company = df3.groupby(['company_hash', 'job_position'])['ctc_capped'].mean().reset_index()
#to get top 2 positions
top_2_postions_per_company = top_2_postions_per_company.groupby('company_hash').apply(lambda x : x.nlargest(2,
top_2_postions_per_company
```

Out[58]:

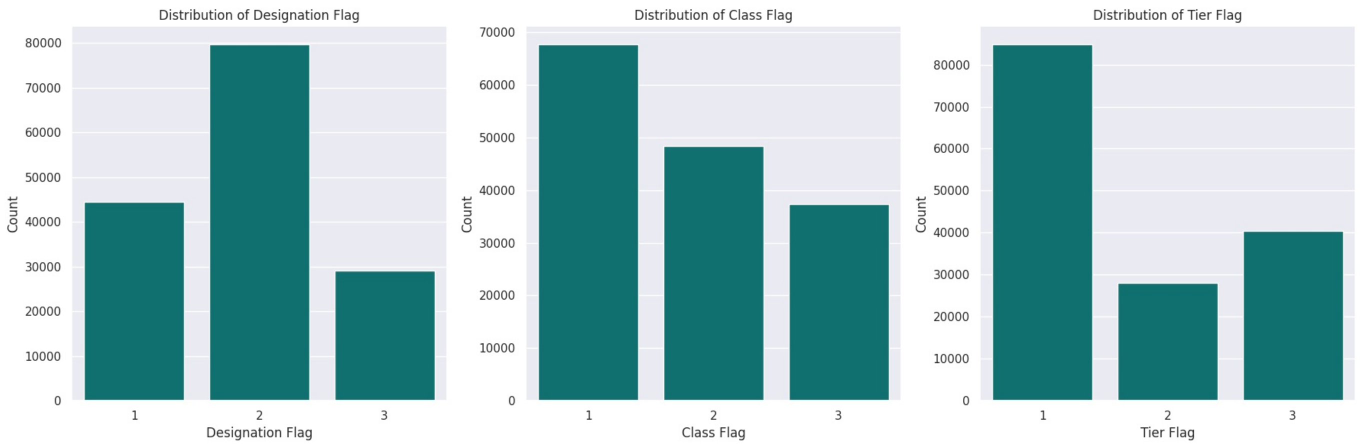
	company_hash	job_position	ctc_capped
0	0	Other	100000.0
1	0000	Other	300000.0
2	01 ojztqsj	Frontend Engineer	830000.0
3	01 ojztqsj	Android Engineer	270000.0
4	05mz exzytvrny uqxcvnt rxbxnta	Backend Engineer	1100000.0
...	...	...	...
44121	zyvzwt wgzohrnxzs tzsxzttqo	Frontend Engineer	940000.0
44122	zz	Other	1370000.0
44123	zz	unknown	500000.0
44124	zzb ztdnstz vacxogqj ucn rna	unknown	600000.0
44125	zzgato	unknown	130000.0

44126 rows × 3 columns

Let us see the distribution analysis of the flags

In [59]:

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot the distribution of Designation_Flag
sns.countplot(x='Designation_Flag', data=df3, ax=axes[0],color='teal')
axes[0].set_title('Distribution of Designation Flag')
axes[0].set_xlabel('Designation Flag')
axes[0].set_ylabel('Count')

# Plot the distribution of Class_Flag
sns.countplot(x='Class_Flag', data=df3, ax=axes[1],color='teal')
axes[1].set_title('Distribution of Class Flag')
axes[1].set_xlabel('Class Flag')
axes[1].set_ylabel('Count')

# Plot the distribution of Tier_Flag
sns.countplot(x='Tier_Flag', data=df3, ax=axes[2],color='teal')
axes[2].set_title('Distribution of Tier Flag')
axes[2].set_xlabel('Tier Flag')
axes[2].set_ylabel('Count')

# Display the plots
plt.tight_layout()
plt.show()
```



Class Flag distribution looks more balanced as compared to Designation and Tier Flag

In [ ]:

Q. Discuss the distribution of learners based on the Tier flag:

1. Which companies dominate in Tier 1 and why might this be the case?

In [60]:

```
tier_1 = df3[df3['Tier_Flag'] == 1]

#learners count in Tier 1 in each company
tier_1_company_count = tier_1['company_hash'].value_counts().reset_index()
tier_1_company_count.columns = ['company_hash', 'count']
```

```
#top 10 companies are
tier_1_company_count.head(10)
```

Out[60]:

	company_hash	count
0	nvnv wgzohrnvzwj otqcxwto	4642
1	xzegojo	2947
2	zgn vuurxwvmrt vwwghzn	1804
3	wgszxkvzn	1783
4	vwwtznhqt	1660
5	vbvkgz	1564
6	fxuqg rxbxnta	1513
7	gqvwrt	1136
8	wvustbxzx	1039
9	zvz	983

Companies Dominating in Tier 1

- Common Factors: Companies dominating Tier 1 might have a large number of entry-level positions or companies that offer lower-than-average compensation.
- Possible Reasons: Large enterprises with many junior or mid-level positions. Companies in traditional industries or smaller firms with limited budgets.

2. Are there any notable patterns or insights when comparing learners from Tier 3 across different companies?

In [61]:

```
tier_3 = df3[df3['Tier_Flag'] == 3]

#learners count in Tier 3 in each company
tier_3_company_count = tier_3['company_hash'].value_counts().reset_index()
tier_3_company_count.columns = ['company_hash', 'count']

#top 10 companies are
tier_3_company_count.head(10)
```

Out[61]:

	company_hash	count
0	vbvkgz	876
1	nvnv wgzohrnvzwj otqcxwto	694
2	gqvwrt	611
3	bxwqgogen	592
4	xzegojo	579
5	zvz	416
6	wgszxkvzn	416
7	zgn vuurxwvmrt vwwghzn	388
8	vagmt	366
9	wvustbxzx	336

Patterns in Tier 3 Across Different Companies

- High CTC Companies: Companies with a high number of Tier 3 learners might be in tech, finance, or other high-paying sectors.
- Career Progression: These companies might offer better career progression and compensation growth.
- Retention Strategy: Higher compensation could be a strategy to retain top talent.

In [ ]:

Summary statistics

In [62]:

```
designation_summary = df3.groupby('Designation_Flag')['ctc_capped'].describe()
designation_summary
```

Out[62]:

Designation_Flag	count	mean	std	min	25%	50%	75%	max
1	44536.0	8.964837e+05	7.123766e+05	2.0	400000.0	697000.0	1200000.0	11200000.0
2	79743.0	1.606745e+06	2.846606e+06	15.0	550000.0	950000.0	1700000.0	24000000.0
3	29164.0	2.508838e+06	3.605452e+06	14.0	1000000.0	1600000.0	2650000.0	24000000.0

In [63]:

```
class_summary = df3.groupby('Class_Flag')['ctc_capped'].describe()
class_summary
```

Out[63]:

Class_Flag	count	mean	std	min	25%	50%	75%	max
1	67733.0	9.048573e+05	6.766357e+05	2.0	450000.0	710000.0	1200000.0	20000000.0
2	48358.0	1.520279e+06	3.011210e+06	24.0	490000.0	810000.0	1500000.0	24000000.0
3	37352.0	2.848947e+06	3.778324e+06	16.0	1200000.0	1900000.0	3000000.0	24000000.0

In [64]:

```
tier_summary = df3.groupby('Tier_Flag')['ctc_capped'].describe()
tier_summary
```

Out[64]:

Tier_Flag	count	mean	std	min	25%	50%	75%	max
1	84894.0	8.710126e+05	5.787411e+05	2.0	450000.0	730000.0	1150000.0	17000000.0
2	28069.0	1.491308e+06	3.280566e+06	24.0	400000.0	730000.0	1310000.0	24000000.0
3	40480.0	3.098245e+06	3.936230e+06	16.0	1400000.0	2100000.0	3200000.0	24000000.0

- Mean CTC in all the categories and under each flag is similar
- Maximum CTC in flags 2 and 3 of all the categories is same

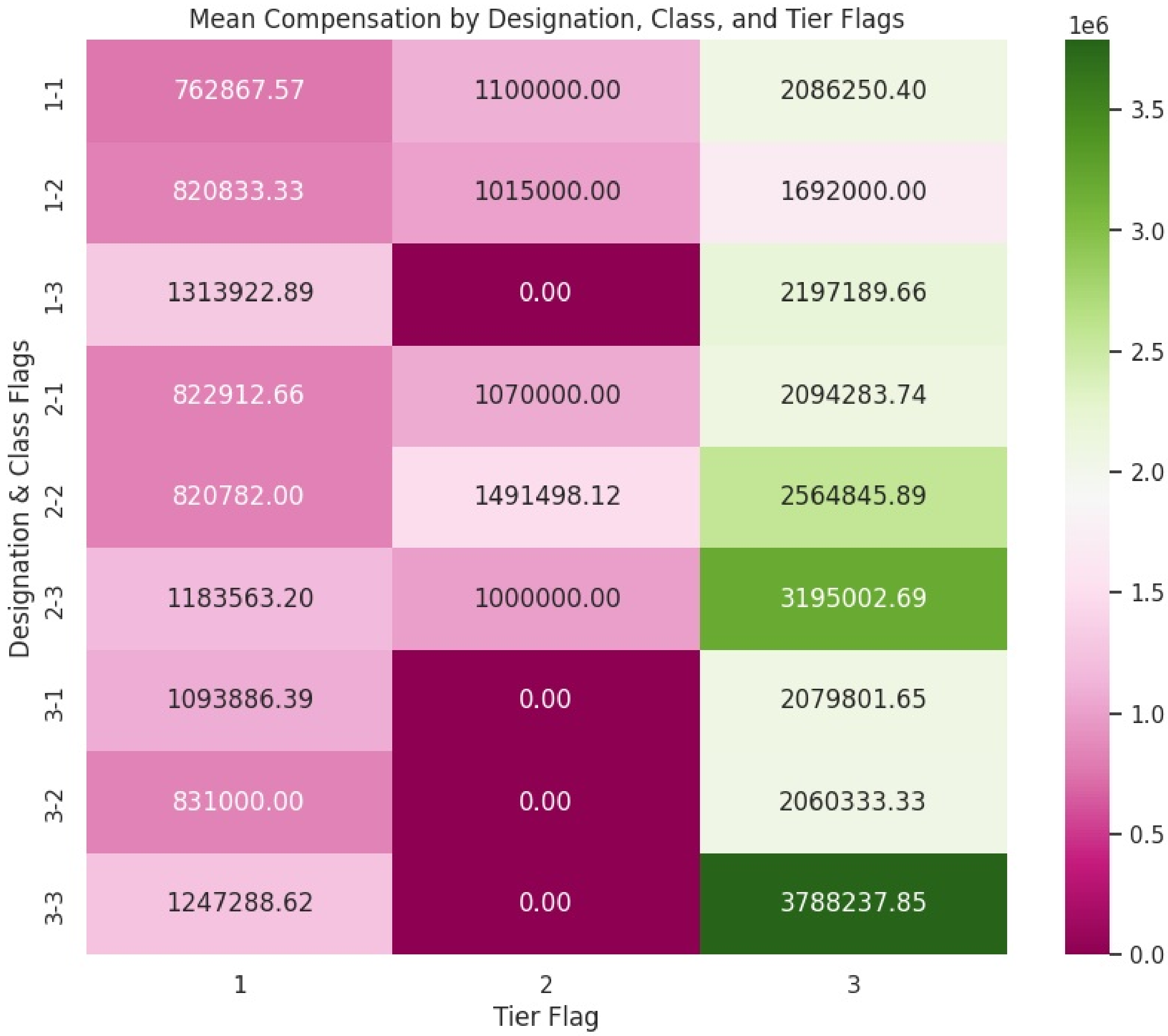
In [ ]:

```

```

Visualizing Mean Compensation

In [65]:

```
# Mean compensation by flags
mean_compensation_flags = df3.groupby(['Designation_Flag', 'Class_Flag', 'Tier_Flag'])['ctc_capped'].mean().uns

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(mean_compensation_flags, annot=True, cmap='PiYG', fmt='.2f')
plt.title('Mean Compensation by Designation, Class, and Tier Flags')
plt.xlabel('Tier Flag')
plt.ylabel('Designation & Class Flags')
plt.show()
```



- Mean CTC in Designation Flag 3 , Class Flag 3 and Tier Flag 3 are highly correlated
- Followed by mean CTC of D2, C3 and T3
- Tier 2 ctc is not correlated to D3 and any of the flag of Class

In [ ]:

Examining the relationship between these flags can reveal if there are patterns or dependencies among them.

For that we can use crosstabs

In [66]:

```
#Designation vs Class
designation_class_ct = pd.crosstab(df3['Designation_Flag'], df3['Class_Flag'], normalize='index')*100

#Designation vs Tier
designation_tier_ct = pd.crosstab(df3['Designation_Flag'], df3['Tier_Flag'], normalize='index')*100

#Class vs Tier
class_tier_ct = pd.crosstab(df3['Class_Flag'], df3['Tier_Flag'], normalize='index')*100

print("Cross-Tabulation between Designation Flag and Class Flag:\n", designation_class_ct)
print("\nCross-Tabulation between Designation Flag and Tier Flag:\n", designation_tier_ct)
print("\nCross-Tabulation between Class Flag and Tier Flag:\n", class_tier_ct)
```

```
Cross-Tabulation between Designation Flag and Class Flag:
 Class_Flag                1          2          3
Designation_Flag
1                 91.189150   0.042662   8.768188
2                 21.136651  60.594660  18.268688
3                 35.200933   0.065149  64.733919

Cross-Tabulation between Designation Flag and Tier Flag:
 Tier_Flag                 1          2          3
Designation_Flag
1                 91.696605   0.006736   8.296659
2                 38.407133  35.195566  26.397301
3                 46.046496   0.000000  53.953504

Cross-Tabulation between Class Flag and Tier Flag:
 Tier_Flag           1          2          3
Class_Flag
1           93.678118   0.010335   6.311547
2           24.171802  58.023491  17.804707
3           26.113729   0.008032  73.878239
```

Cross-Tabulation between Designation_Flag and Class_Flag:

- Designation_Flag 1: 91.2% of these employees are also in Class_Flag 1, indicating a strong overlap where lower designations coincide with lower class levels.
- Designation_Flag 2: Majority (60.6%) are in Class_Flag 2, meaning median designation levels align with median class levels.
- Designation_Flag 3: 64.7% are in Class_Flag 3, showing higher designations are often associated with higher class levels.

Cross-Tabulation between Designation_Flag and Tier_Flag:

- Designation_Flag 1: 91.7% are in Tier_Flag 1, showing low designation levels are mostly in lower tier companies.
- Designation_Flag 2: Distribution is more spread with notable percentages in all tiers.
- Designation_Flag 3: 53.9% in Tier_Flag 3, indicating higher designations are more common in higher tier companies.

Cross-Tabulation between Class_Flag and Tier_Flag:

- Class_Flag 1: 93.7% are in Tier_Flag 1, indicating lower class levels are predominantly in lower tier companies.
- Class_Flag 2: 58% in Tier_Flag 2, showing median class levels align with median tier companies.
- Class_Flag 3: 73.9% in Tier_Flag 3, indicating higher class levels are mostly in higher tier companies.

Insights

1. Alignment within Categories: There is a noticeable alignment within the categories where higher flags in one dimension (e.g., Designation_Flag) often coincide with higher flags in another dimension (e.g., Class_Flag and Tier_Flag). This suggests that performance, role importance, and company tier are interconnected.

2. Disparities at Median Levels: Employees with median flags (Flag 2) in one category tend to have a more spread distribution across other categories. This indicates that employees at the median level in terms of designation, class, or tier are not strictly confined to the median level in the other categories.

3. Low-End and High-End Correlation: Employees at the low end (Flag 1) in one category are predominantly at the low end in others, and similarly for the high end (Flag 3). This can be used to target interventions or identify opportunities for improvement for lower-tier employees.

In [ ]:

EDA (Exploratory Data Analysis)

- Univariate analysis
- Bivariate analysis
- Statistical Summary

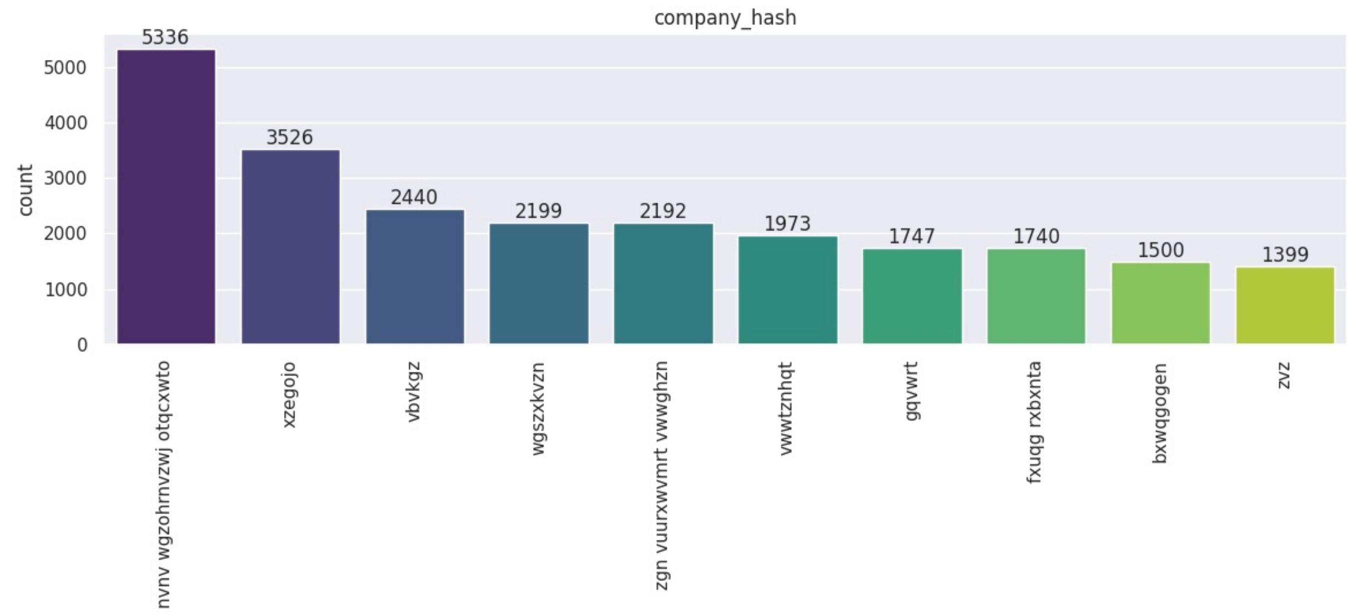
In [67]:
```
df3.info()
```

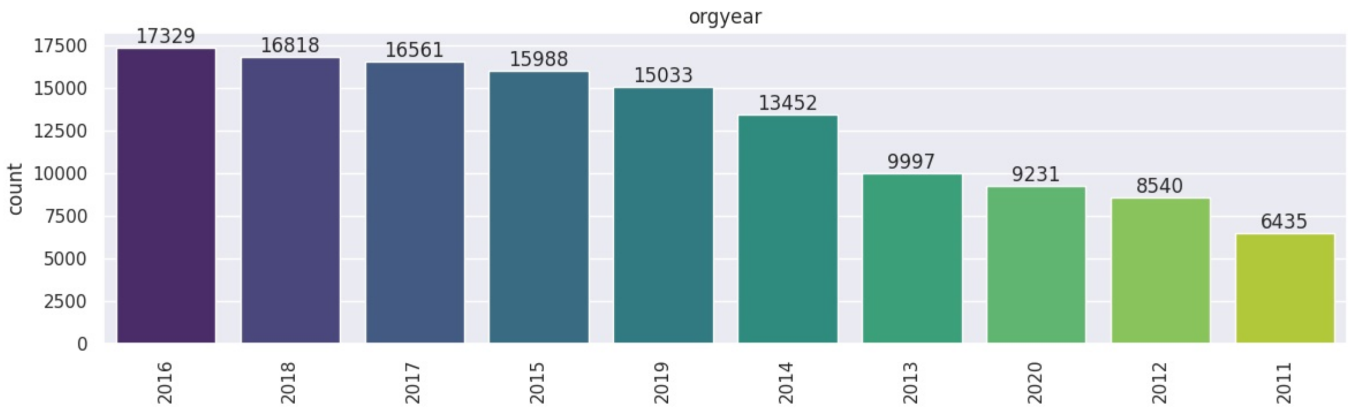
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153443 entries, 0 to 153442
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   company_hash       153443 non-null  object
 1   email_hash         153443 non-null  object
 2   orgyear            153443 non-null  int64
 3   job_position       153443 non-null  object
 4   ctc_updated_year   153443 non-null  int32
 5   ctc_capped         153443 non-null  int64
 6   YOE_capped         153443 non-null  int64
 7   Designation_Flag   153443 non-null  int64
 8   Class_Flag         153443 non-null  int64
 9   Tier_Flag          153443 non-null  int64
dtypes: int32(1), int64(6), object(3)
memory usage: 11.1+ MB
```

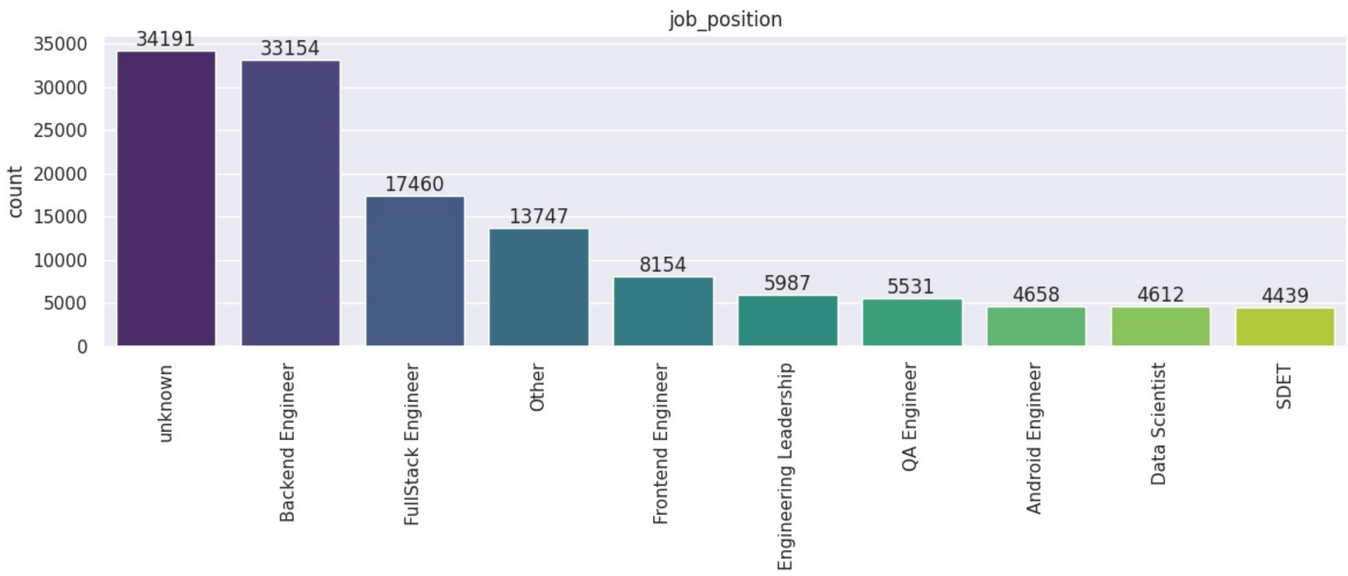
In [68]:
```
cat_cols = ['company_hash', 'orgyear', 'job_position']

num_cols = ['ctc_capped', 'YOE_capped']
```

Univariate Analysis

In [69]:
```
#For categorical features
plt.figure(figsize=(12, 16))

i = 1
for col in cat_cols:
    # Get the top 10 values for the column
    top_10 = df3[col].value_counts().nlargest(10)
    top_10_index = top_10.index

    ax = plt.subplot(3, 1, i)
    sns.countplot(x=df3[col], order=top_10_index, palette = 'viridis')
    for j in ax.containers:
      ax.bar_label(j)
    plt.title(f'{col}')
    if i <= 3:
        plt.xticks(rotation=90)

    ax.set_xlabel('')
    i += 1

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







Insights

- We can easily find top 10 companies in terms of count in the dataset
- Top job position is 'unknown' followed by 'Backend Engineer' and 'FullStack Engineer'
- Most of the employees started working in the year 2016 followed by 2018 and 2017

In [ ]:

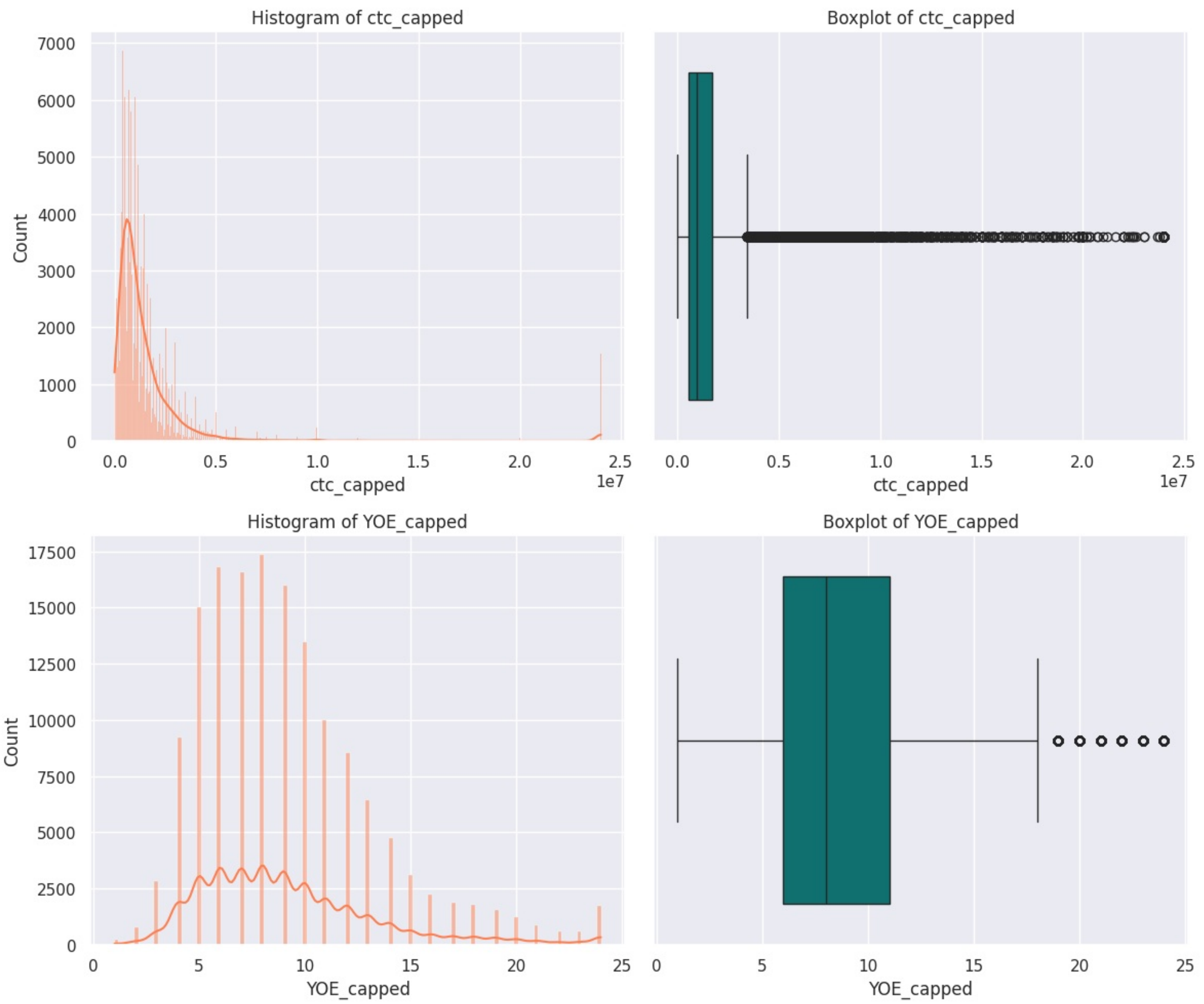
```

```

Now we can create histograms and boxplots to analyze the numerical features.

In [70]:

```
#for numerical features
```

```
plt.figure(figsize = (12,10))

#creating histograms and boxplots using a for loop
for i, col in enumerate(num_cols) :
  #Histogram
  ax1 = plt.subplot(2, 2, 2*i + 1)
  sns.histplot(df3[col], kde = True, color = 'coral')
  plt.title(f"Histogram of {col}")

  #Boxplot
  ax2 = plt.subplot(2, 2, 2*i + 2)
  sns.boxplot(x=df3[col], color='teal')
  plt.title(f'Boxplot of {col}')


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



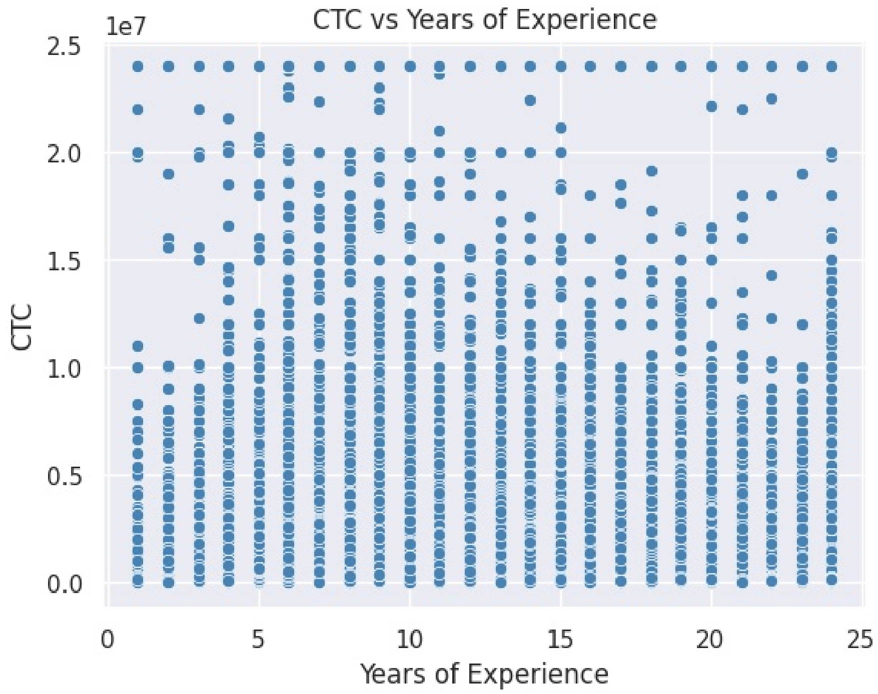
Insights

- Distribution of CTC is right skewed
- Most of the ctc is around 10 Lac
- Distribution of YOE is almost normal with most of the YOE lying around 6-9 years

Bivariate Analysis

CTC vs Years of Experience

In [71]:
```
#using scatterplot
sns.scatterplot(x = 'YOE_capped', y = 'ctc_capped', data = df3, color = 'steelblue')
plt.title('CTC vs Years of Experience')
plt.xlabel('Years of Experience')
plt.ylabel('CTC')
plt.show()
```



Insights

- There is no linear relationship or any specific pattern between Years of Experience and CTC

Years of Experience vs Avg. CTC

Q. Is it always true that with an increase in years of experience, the CTC increases? Provide a case where this isn't true.

In [72]:
```
#we can use a lineplot
#avg ctc per year of experience
avg_ctc_per_yoe = df3.groupby('YOE_capped')['ctc_capped'].mean().reset_index()

#linieplot
sns.lineplot(x = 'YOE_capped', y = 'ctc_capped', data = avg_ctc_per_yoe,marker = 'o', color = 'teal')
plt.title('Average CTC vs Years of Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Average CTC')
plt.show()
```

Insights

- Avg. CTC is decreasing from 1 to 5 years of Experience. There might be a slight decrease in CTC with increasing experience, possibly due to industry-specific factors or career shifts.
- From 5 to 23 years it is showing natural increase in CTC, then again a drop from 23 to 24 years

In [ ]:

Q. What is the average CTC of learners across different job positions?

In [73]:
```
#group by job positions and calculate avg ctc
avg_ctc_per_job = df3.groupby('job_position')['ctc_capped'].mean().reset_index()

#renaming
avg_ctc_per_job.columns = ['job_positon', 'avg_ctc']

#sorting the ctc in descending order
avg_ctc_per_job = avg_ctc_per_job.sort_values(by = 'avg_ctc', ascending = False)

avg_ctc_per_job
```

Out[73]:

	job_positon	avg_ctc
372	Safety officer	24000000.0
342	Reseller	24000000.0
288	Owner	24000000.0
593	Telar	24000000.0
218	Jharkhand	24000000.0
...	...	...
24	Any technical	10000.0
257	Matlab programmer	10000.0
641	project engineer	7900.0
189	Full-stack web developer	7500.0
273	New graduate	2000.0

652 rows × 2 columns

Q.For a given company, how does the average CTC of a Data Scientist compare with other roles?

In [74]:
```
#first we filter out the companies with Data scientist role
data_scientist_df = df3[df3['job_position'] == 'Data Scientist']

#companies which provide data scientist roles
comp_with_ds = data_scientist_df['company_hash'].unique()

#result
print("Number of companies with Data Scientist job positioin is",len(comp_with_ds))
print("The companies are :")
print(comp_with_ds)
```

```
Number of companies with Data Scientist job positioin is 2533
The companies are :
['ihvznuyx' 'tqxwoogz' 'vrsgzgd ucn rna' ... 'ohbjvs xzoxsyno rrw'
 'yjhzavx bgmxo' 'wgbuzgcv wgznqvwn']
```

In [75]:
```
#comparing the ctc's of data scientist with others

def compare_ctc(comp_hash) :
  #filter for the given company hash
  df_company = df3[df3['company_hash'] == comp_hash]

  #avg ctc for the data scientist in the company
  ds_avg_ctc = df_company[df_company['job_position'] == 'Data Scientist']['ctc_capped'].mean()

  #avg ctc for other roles
  other_avg_ctc = df_company[df_company['job_position'] != 'Data Scientist']['ctc_capped'].mean()

  print(f"Average CTC of Data Scientist in {comp_hash} is {ds_avg_ctc}")
  print(f"Average CTC of other roles in {comp_hash} is {other_avg_ctc}")

  #comparison
  if not pd.isna(ds_avg_ctc) and not pd.isna(other_avg_ctc) :    #non null
    if ds_avg_ctc > other_avg_ctc :
      result = f"Data Scientist has higher average CTC in {comp_hash}"
    elif ds_avg_ctc < other_avg_ctc :
      result = f"Other roles have higher average CTC in {comp_hash}"
    else :
      result = f"Average CTC of Data Scientist and other roles are same in {comp_hash}"
  else :
    result = f"Data Scientist data not available for {comp_hash}"
  return result
```

```
compare_ctc("ihvznuyx")
```

```
Average CTC of Data Scientist in ihvznuyx is 953333.3333333334
Average CTC of other roles in ihvznuyx is 900000.0
```

Out[75]:

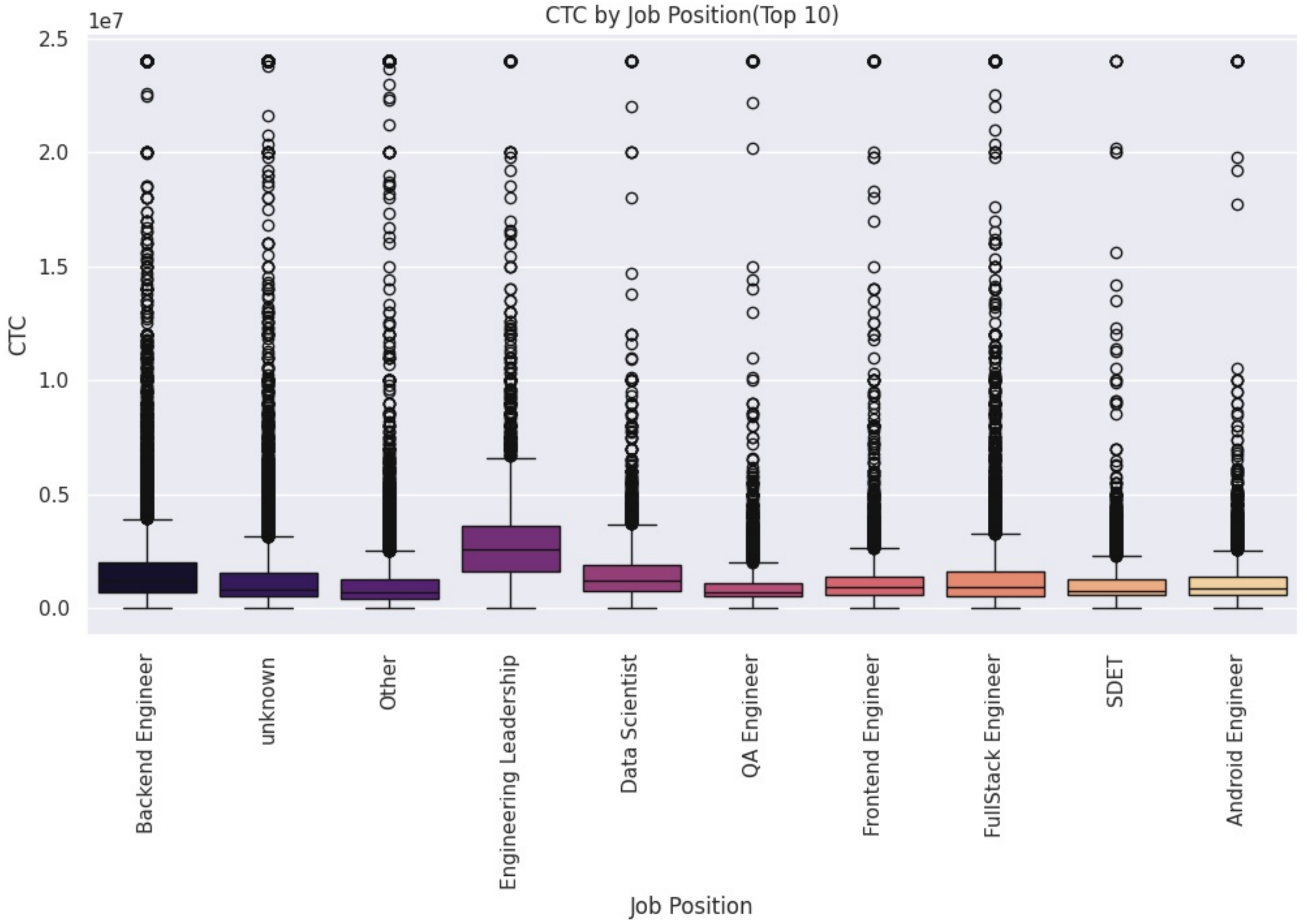
```
'Data Scientist has higher average CTC in ihvznuyx'
```

Likewise we can compare any information for any given company or all the companies

CTC by Job Position

In [76]:

```
#getting top 10 job according to count
top_10_job_positions = df3['job_position'].value_counts().nlargest(10).index

#only including the top 10 job positions
df_top_10_job = df3[df3['job_position'].isin(top_10_job_positions)]

#using boxplot to visualize
plt.figure(figsize = (12,6))
sns.boxplot(data = df_top_10_job, x = 'job_position', y = 'ctc_capped', palette = 'magma')
plt.xticks(rotation = 90)
plt.title('CTC by Job Position(Top 10)')
plt.xlabel('Job Position')
plt.ylabel('CTC')
plt.show()
```



Insights

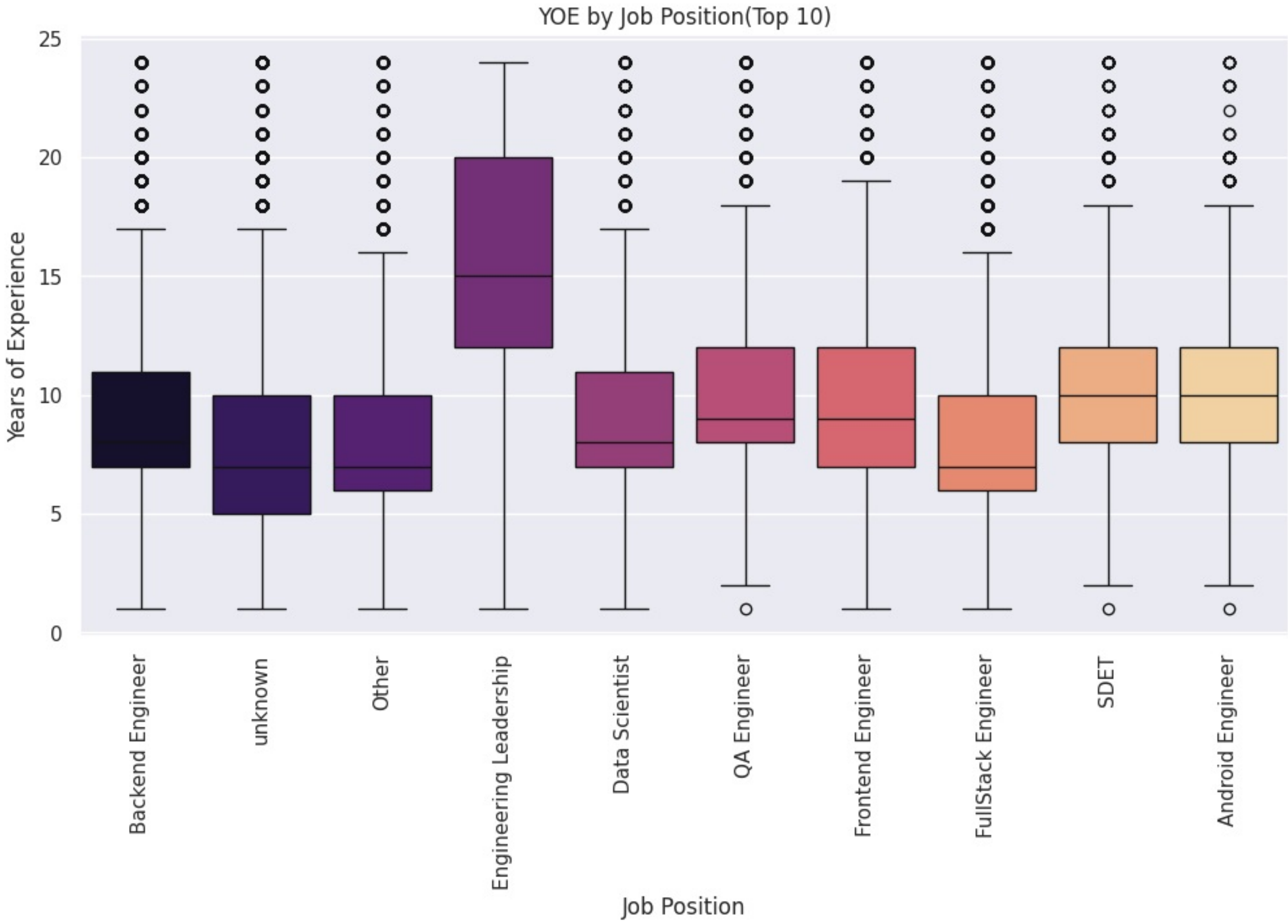
- CTC is highest for Engineering Leadership followed by Backend Engineer and Data Scientist

Years of Experience by Job Position

In [77]:

```
#getting top 10 job according to count
top_10_job_positions = df3['job_position'].value_counts().nlargest(10).index

#only including the top 10 job positions
df_top_10_job = df3[df3['job_position'].isin(top_10_job_positions)]

#using boxplot to visualize
plt.figure(figsize = (12,6))
sns.boxplot(data = df_top_10_job, x = 'job_position', y = 'YOE_capped', palette = 'magma')
plt.xticks(rotation = 90)
plt.title('YOE by Job Position(Top 10)')
plt.xlabel('Job Position')
```

```
plt.ylabel('Years of Experience')
plt.show()
```



Insights

- Years of Experience is highest for Engineering Leadership followed by Android Engineer and SDET

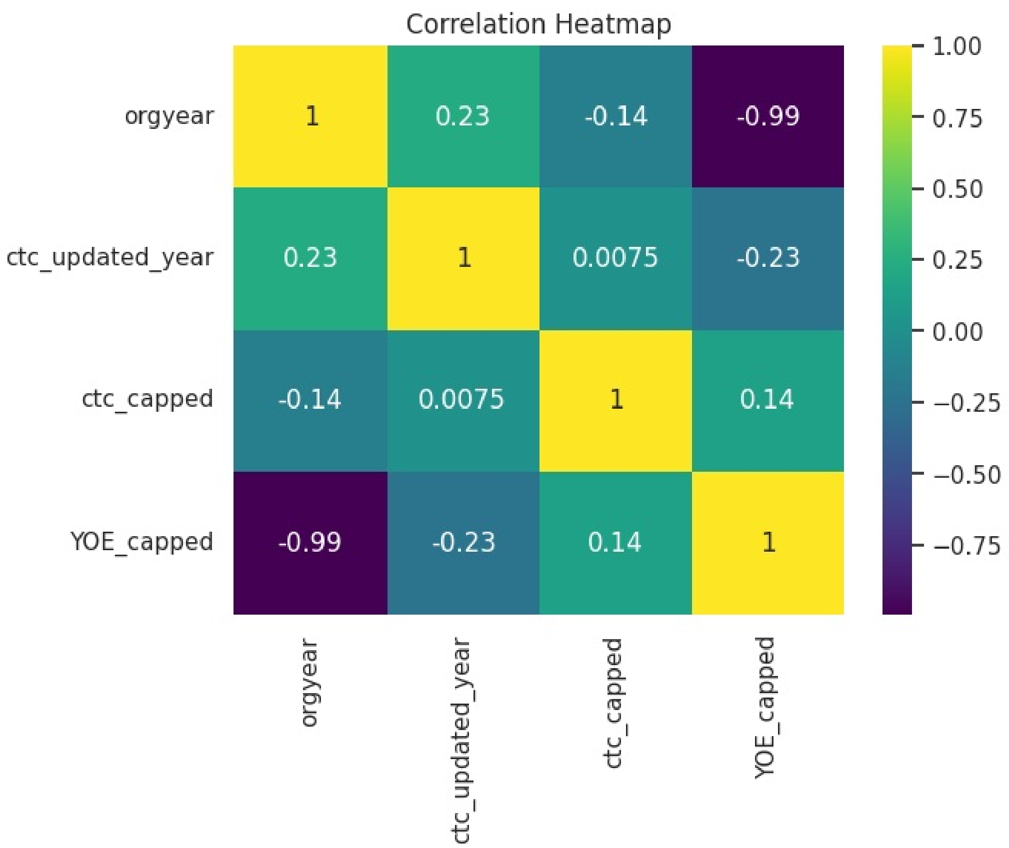
Correlation Heatmap

In [78]:
```
cols = ['orgyear', 'ctc_updated_year', 'ctc_capped', 'YOE_capped']
col_corr = df3[cols].corr()
col_corr
```

Out[78]:

	orgyear	ctc_updated_year	ctc_capped	YOE_capped
orgyear	1.000000	0.225324	-0.142172	-0.992170
ctc_updated_year	0.225324	1.000000	0.007543	-0.232092
ctc_capped	-0.142172	0.007543	1.000000	0.143054
YOE_capped	-0.992170	-0.232092	0.143054	1.000000

In [79]:
```
#Heatmap
sns.heatmap(col_corr, annot = True, cmap = 'viridis')
plt.title("Correlation Heatmap")
plt.show()
```



Insights

- orgyear and ctc_updated_year shown weak positive correlation
- Years of Experience and orgyear show strong negative correlation
- Years of Experience and CTC show weak positive correaltion

In [ ]:

Statistical summary

In [80]:
```
df2.head()
```

Out[80]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year	YOE	ctc
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	700000	Backend Engineer	2019	7	
7	vwwtznhqt ntwyzgrgsj	756d35a7f6bb8ffeaffc8fcca9ddbb78e7450fa0de2be0...	2019	400000	Backend Engineer	2019	5	
9	xrbhd	b2dc928f4c22a9860b4a427efb8ab761e1ce0015fba1a5...	2019	360000	unknown	2019	5	
13	wgszxkvzn	134cc4a76a119493d523f1855a3b7106f64287455d5cd4...	2016	440000	Data Analyst	2020	8	
14	xznhxn	ebcaf397ef5084e05889a6e9a0c3f96a5c8fb0b16749ce...	2016	440000	Backend Engineer	2019	8	

In [81]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 153443 entries, 3 to 205842
Data columns (total 9 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   company_hash      153443 non-null  object
 1   email_hash        153443 non-null  object
 2   orgyear           153443 non-null  int64 
 3   ctc               153443 non-null  int64 
 4   job_position      153443 non-null  object
 5   ctc_updated_year  153443 non-null  int32 
 6   YOE               153443 non-null  int64 
 7   ctc_capped        153443 non-null  int64 
 8   YOE_capped        153443 non-null  int64 
dtypes: int32(1), int64(5), object(3)
memory usage: 11.1+ MB
```

In [82]:
```
df2.describe(include = 'object')
```

Out[82]:

	company_hash	email_hash	job_position
count	153443	153443	153443
unique	36366	153443	652
top	nvnv wgzohrnvzwj otqcxwto	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	unknown
freq	5336	1	34191

In [83]:
```
df2.describe()
```

Out[83]:

	orgyear	ctc	ctc_updated_year	YOE	ctc_capped	YOE_capped
count	153443.000000	1.534430e+05	153443.00000	153443.000000	1.534430e+05	153443.000000
mean	2014.811467	2.501398e+06	2019.42172	9.188533	1.572051e+06	9.153744
std	4.369586	1.307523e+07	1.36023	4.369586	2.670005e+06	4.212774
min	1900.000000	2.000000e+00	2015.00000	1.000000	2.000000e+00	1.000000
25%	2013.000000	5.500000e+05	2019.00000	6.000000	5.500000e+05	6.000000
50%	2016.000000	9.500000e+05	2019.00000	8.000000	9.500000e+05	8.000000
75%	2018.000000	1.700000e+06	2020.00000	11.000000	1.700000e+06	11.000000
max	2023.000000	1.000150e+09	2021.00000	124.000000	2.400000e+07	24.000000

Insights

- Dataset have got 36366 unique companies
- There are 153443 unique learners
- And 652 unique job positions
- Minimum year of Employment starting date is 1900 and maximum is 2023
- Minimum CTC is 2 and maximum 2.4 cr after capping
- Minimum Years of experience is 1 and maximum is 24 after capping

In [ ]:
```

```

⚙Data Processing for Unsupervised Learning

- Feature Engineering
- Encoding
- Log Transformation
- Scaling

Creating a new dataset by removing the flags since it was only useful for manual clustering

In [84]:
```
df4 = df3.drop(['Designation_Flag', 'Class_Flag', 'Tier_Flag'], axis = 1)
```

In [85]:
```
df4.head()
```

Out[85]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped	YOE_
0	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	Backend Engineer	2019	700000	
1	vwwtznhqt ntwyzgrgsj	756d35a7f6bb8ffeaffc8fcca9ddbb78e7450fa0de2be0...	2019	Backend Engineer	2019	400000	
2	xrbhd	b2dc928f4c22a9860b4a427efb8ab761e1ce0015fba1a5...	2019	unknown	2019	360000	
3	wgszxkvzn	134cc4a76a119493d523f1855a3b7106f64287455d5cd4...	2016	Data Analyst	2020	440000	
4	xznhxn	ebcaf397ef5084e05889a6e9a0c3f96a5c8fb0b16749ce...	2016	Backend Engineer	2019	440000	

Feature Engineering

Creating a new feature 'no_of_ctc_update' signifying the number of times CTC got updated of a learner which is derived from frequency of email_hash in the dataset

In [86]:
```
#copy of df
dfcopy = df.copy()
```

```
In [87]: #frequency of email hashes in df
email_hash_freq = dfcopy['email_hash'].value_counts().reset_index()
email_hash_freq.columns = ['email_hash', 'no_of_ctc_update']

#merging this with df4
df4_merged = pd.merge(df4, email_hash_freq, on='email_hash', how='left')
```

```
In [88]: df4_merged.head()
```

Out[88]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year	ctc_capped	YOE_
0	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	Backend Engineer	2019	700000	
1	vwwtznhqt ntwyzgrgsj	756d35a7f6bb8ffeaffc8fcca9ddbb78e7450fa0de2be0...	2019	Backend Engineer	2019	400000	
2	xrbhd	b2dc928f4c22a9860b4a427efb8ab761e1ce0015fba1a5...	2019	unknown	2019	360000	
3	wgszxkvzn	134cc4a76a119493d523f1855a3b7106f64287455d5cd4...	2016	Data Analyst	2020	440000	
4	xznhxn	ebcaf397ef5084e05889a6e9a0c3f96a5c8fb0b16749ce...	2016	Backend Engineer	2019	440000	

Removing followng columns:

- email_hash: It is unique for each row and do not provide useful information for clustering
- orgyear: We have got Years of Experience derived from this feature which is more relevant then just the year of joining for clustering algorithm
- ctc_updated_year: We have derived a feature no. of ctc update signifying number of times ctc got updated of a learner which is more relevant to clustering algorithm than mere year as a timeline or int

```
In [89]: df4_merged = df4_merged.drop(['email_hash', 'orgyear', 'ctc_updated_year'], axis = 1)
```

```
In [90]: df4_merged.head()
```

Out[90]:

	company_hash	job_position	ctc_capped	YOE_capped	no_of_ctc_update
0	ngpgutaxv	Backend Engineer	700000	7	1
1	vwwtznhqt ntwyzgrgsj	Backend Engineer	400000	5	1
2	xrbhd	unknown	360000	5	1
3	wgszxkvzn	Data Analyst	440000	8	1
4	xznhxn	Backend Engineer	440000	8	1

```
In [91]: df5 = df4_merged.copy()
```

Encoding

Encoding categorical columns

- Here we are going to use frequency encoding
- Frequency Encoding replaces each categorical value with its frequency in the dataset. A good compromise between simplicity and capturing categorical variable importance.

```
In [92]: # Frequency encoding for company_hash
company_hash_freq = df4_merged['company_hash'].value_counts().to_dict()
df4_merged['company_hash_encoded'] = df4_merged['company_hash'].map(company_hash_freq)

# Frequency encoding for job_position
job_position_freq = df4_merged['job_position'].value_counts().to_dict()
df4_merged['job_position_encoded'] = df4_merged['job_position'].map(job_position_freq)
```

.to_dict():

- Converts the resulting pandas Series into a Python dictionary where:
  - Keys: Unique values in the company_hash column.
  - Values: Their respective frequencies.

We can drop 'company_hash' and 'job_postiton' from df4_merged

```
In [93]: df4_merged = df4_merged.drop(['company_hash', 'job_position'], axis = 1)
```

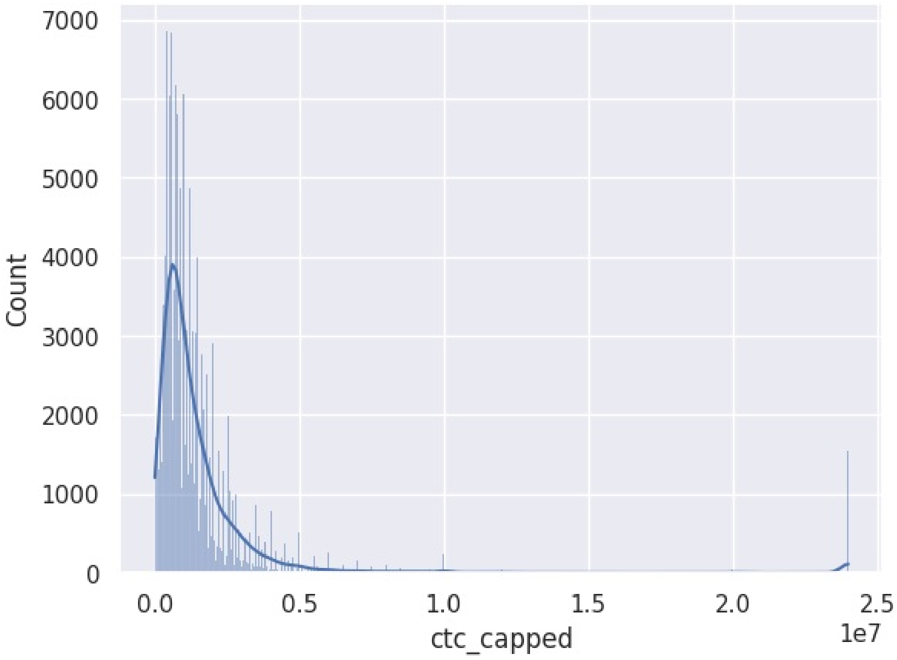
In [94]:
```
df4_merged.head()
```

Out[94]:

	ctc_capped	YOE_capped	no_of_ctc_update	company_hash_encoded	job_position_encoded
0	700000	7	1	53	33154
1	400000	5	1	15	33154
2	360000	5	1	1	34191
3	440000	8	1	2199	2222
4	440000	8	1	202	33154

Log transformation

In [95]:
```
sns.histplot(df4_merged['ctc_capped'], kde = True)
plt.show()
```



- We can see that the 'ctc_capped' column is Right Skewed
- Since skewness can affect performance of clustering algorithms, we can apply Log Transformation on ctc_capped column.

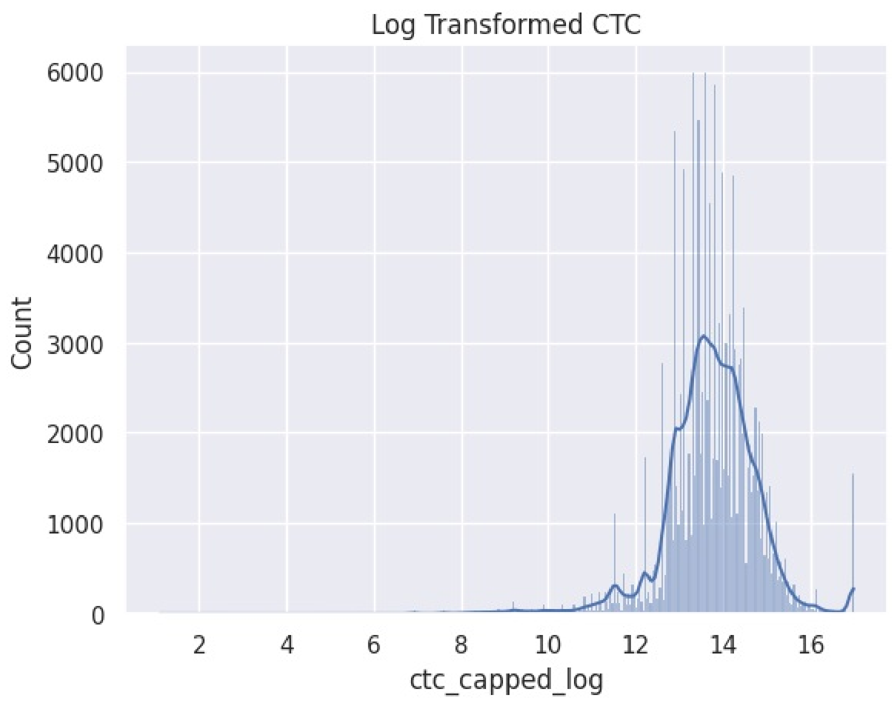
In [96]:
```
df4_merged['ctc_capped_log'] = np.log1p(df4_merged['ctc_capped'])

#dropping the 'ctc_capped' column
df4_merged = df4_merged.drop(['ctc_capped'], axis = 1)
```

In [97]:
```
df4_merged.head()
```

Out[97]:

	YOE_capped	no_of_ctc_update	company_hash_encoded	job_position_encoded	ctc_capped_log
0	7	1	53	33154	13.458837
1	5	1	15	33154	12.899222
2	5	1	1	34191	12.793862
3	8	1	2199	2222	12.994532
4	8	1	202	33154	12.994532

In [98]:
```
sns.histplot(df4_merged['ctc_capped_log'], kde = True)
plt.title('Log Transformed CTC')
plt.show()
```



The feature now shows Normal Distribution

Standard Scaling

It transforms data such that the mean of each feature becomes 0 and the standard deviation becomes 1.

In [99]:
```
from sklearn.preprocessing import StandardScaler
```

In [100…
```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the data
scaled_features = scaler.fit_transform(df4_merged[['YOE_capped', 'no_of_ctc_update', 'company_hash_encoded', 'j

# Convert the scaled features back to a DataFrame
df_scaled = pd.DataFrame(scaled_features, columns=['YOE_capped', 'no_of_ctc_update', 'company_hash_encoded', 'j
```

In [101…
```
df_scaled.head()
```

Out[101…

	YOE_capped	no_of_ctc_update	company_hash_encoded	job_position_encoded	ctc_capped_log
0	-0.511243	-0.530558	-0.453490	1.025669	-0.271471
1	-0.985991	-0.530558	-0.485592	1.025669	-0.810550
2	-0.985991	-0.530558	-0.497419	1.103940	-0.912044
3	-0.273869	-0.530558	1.359455	-1.309018	-0.718737
4	-0.273869	-0.530558	-0.327614	1.025669	-0.718737

All the features has been scaled

In [ ]:
```
```

Model Building

- K-means clustering
- Hierarchical clustering

Checking Clustering tendency - Hopkins statistics

In [102…
```
from sklearn.neighbors import NearestNeighbors
```

In [103…
```
#function to calculate hopkins statistics
def hopkins_statistic(X) :
  X = np.array(X)
  n, d = X.shape    #number of datapoints and dimensions
  m = int(0.1*n)    #Subset size(10% of datapoints)
```

```
  nbrs = NearestNeighbors(n_neighbors=1).fit(X)
  rand_X = np.random.random((m, d)) * np.amax(X, axis = 0)
  u_distances, _ = nbrs.kneighbors(rand_X, 2, return_distance = True)

  w_distances, _ = nbrs.kneighbors(X[np.random.choice(n, m, replace = False)], 2, return_distance = True)

  u_distances = u_distances[:, 1]
  w_distances = w_distances[:, 1]

  H = (np.sum(u_distances) / (np.sum(u_distances) + np.sum(w_distances)))
  return H

hopkins_score = hopkins_statistic(df_scaled)
print("Hopkins Statistic:", hopkins_score)
```

```
Hopkins Statistic: 0.9828651279356919
```

- The value is very close to 1, which means that the dataset has a very strong clustering structure.
- It is likely to form well defined clusters

Elbow Method- To select optimal number of clusters

Inertia

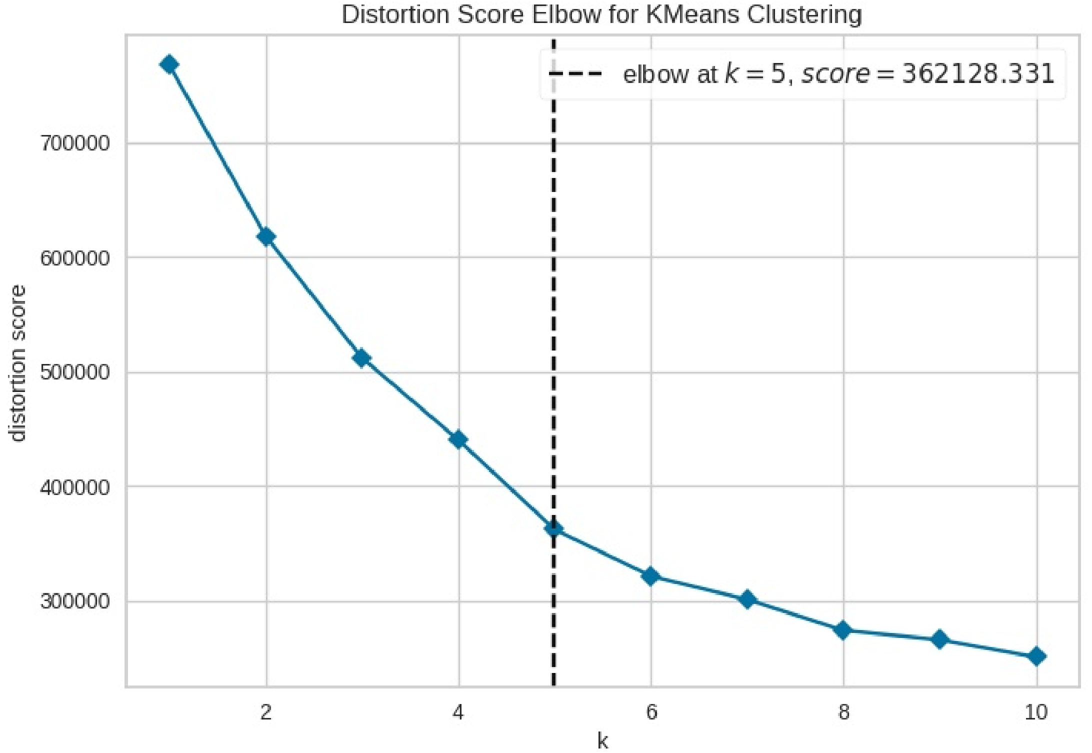
Within Cluster Sum of Squares(WCSS)

- This metric measures how tightly the clusters are packed.
- Lower inertia values indicate better-defined clusters.

In [104…
```
from sklearn.cluster import KMeans
```

In [105…
```
#importing elbow
from yellowbrick.cluster import KElbowVisualizer
```

In [106…
```
#initializing
model = KMeans()

# k is the range of number of clusters
visualizer = KElbowVisualizer(model, k = (1,11), timings = False)
#fitting
visualizer.fit(df_scaled)
visualizer.show()
```



Out[106…
```
<Axes: title={'center': 'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>
```

The elbow point suggests that 5 clusters is a good choice for our data. This is where the inertia starts to decrease at a slower rate, indicating that additional clusters beyond this point don't significantly improve the clustering quality.

K-Means Clustering

In [107…

```
optimal_clusters = 5  # Set the optimal number of clusters as found above
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
kmeans.fit(df_scaled)

# Adding cluster labels to the DataFrame
df5['kmeans_cluster'] = kmeans.labels_
```

In [108…

```
df5.head()
```

Out[108…

	company_hash	job_position	ctc_capped	YOE_capped	no_of_ctc_update	kmeans_cluster
0	ngpgutaxv	Backend Engineer	700000	7	1	3
1	vwwtznhqt ntwyzgrgsj	Backend Engineer	400000	5	1	3
2	xrbhd	unknown	360000	5	1	3
3	wgszxkvzn	Data Analyst	440000	8	1	0
4	xznhxn	Backend Engineer	440000	8	1	3

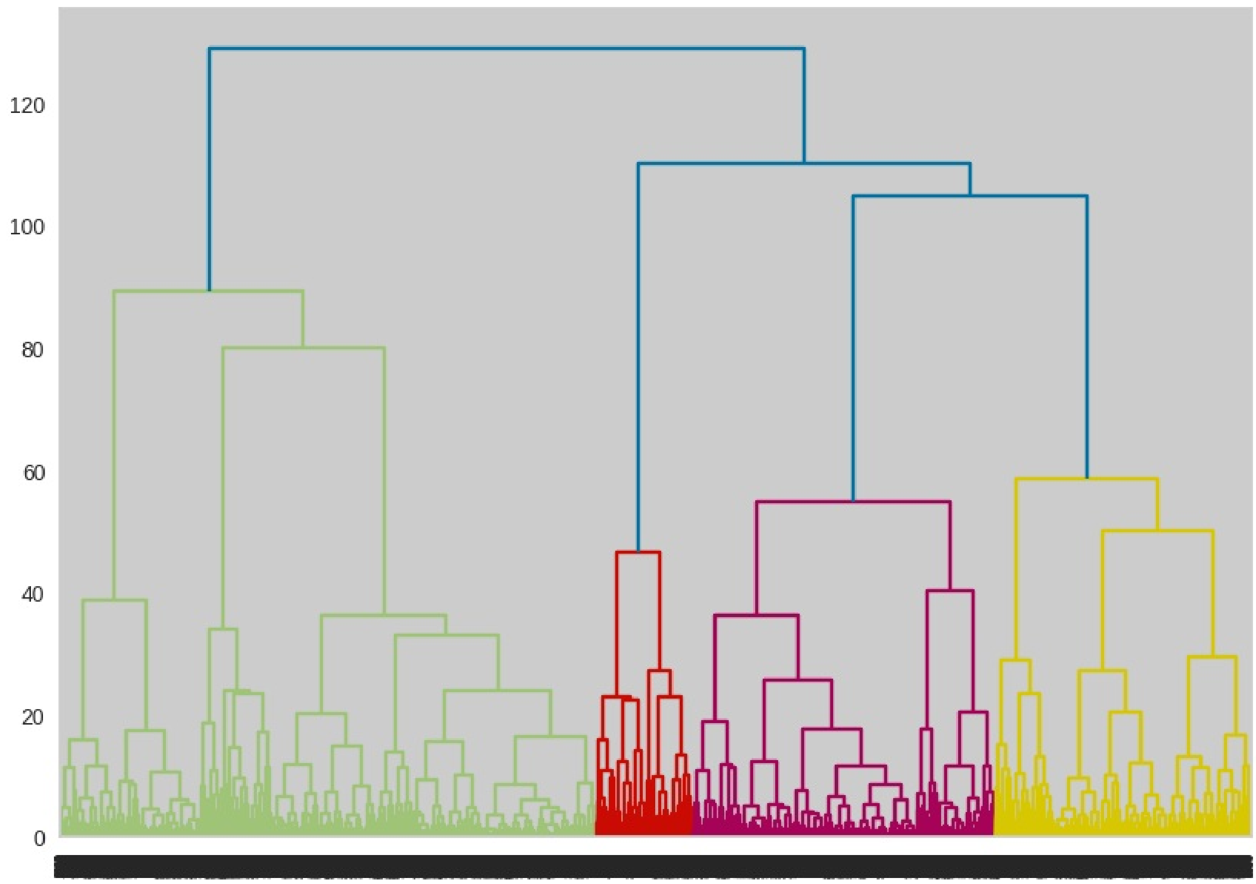
Hierarchical Clustering

In [109…

```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
```

In [110…

```
# Sample a subset of the data
df_sampled = df_scaled.sample(n=10000, random_state=42)

# Perform hierarchical clustering
Z = linkage(df_sampled, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.show()
```



- Used Representative subset of data to avoid running out of memory.
- Dendogram is showing 4 different colored branches at the end representing 4 clusters

Elbow Method suggests 5 clusters and Dendogram are suggesting 4 clusters for the given dataset

Evaluation of K-means Clustering

Within-Cluster Sum of Squares (WCSS)

The Within-Cluster Sum of Squares (WCSS) is a measure of the compactness of the clusters formed by the K-means algorithm. It represents the sum of squared distances between each data point and its corresponding cluster centroid. A lower WCSS value indicates tighter clusters, meaning that the data points within each cluster are closer to their respective centroid.

In [111…
```
#evaluating
wcss = kmeans.inertia_
print(f'Within-Cluster Sum of Squares (WCSS): {wcss}')
```

```
Within-Cluster Sum of Squares (WCSS): 362131.63471221185
```

WCSS Value Consistency: The WCSS value remains consistent at 362131.63471221185 for k=5. This value represents the total within-cluster variance for the five clusters formed by K-means.

Optimal Number of Clusters:

- The elbow method helps identify the optimal number of clusters by plotting WCSS values for different k values and looking for a point where the decrease in WCSS slows down.
- If k=5 is identified as the elbow point, it suggests that adding more clusters beyond this number does not significantly reduce the WCSS, indicating diminishing returns in terms of cluster compactness.

Between-Cluster Sum of Squares (BCSS)

This value represents the total squared distance between each cluster centroid and the overall mean of the data, weighted by the number of points in each cluster. A higher BCSS indicates that the cluster centroids are far from the overall mean, suggesting well-separated clusters.

In [112…
```
#calculation
# Assuming df_scaled is your scaled dataframe
df_scaled_copy = df_scaled.copy()

# Adding cluster labels to the DataFrame
df_scaled_copy['kmeans_cluster'] = kmeans.labels_

# Between-Cluster Sum of Squares (BCSS)
def calculate_bcss(df, kmeans):
    cluster_centers = kmeans.cluster_centers_
    overall_mean = df.drop(columns='kmeans_cluster').mean(axis=0)
    bcss = 0
    for i, center in enumerate(cluster_centers):
        size = len(df[df['kmeans_cluster'] == i])
        bcss += size * np.sum((center - overall_mean) ** 2)
    return bcss

bcss = calculate_bcss(df_scaled_copy, kmeans)
print(f'Between-Cluster Sum of Squares (BCSS): {bcss}')
```

```
Between-Cluster Sum of Squares (BCSS): 405340.05670200626
```

Insights

- High BCSS and Low WCSS
- The combination of a relatively high BCSS and a relatively low WCSS is desirable. It means that the clusters are well-separated and compact.

Visual Inspection- PCA

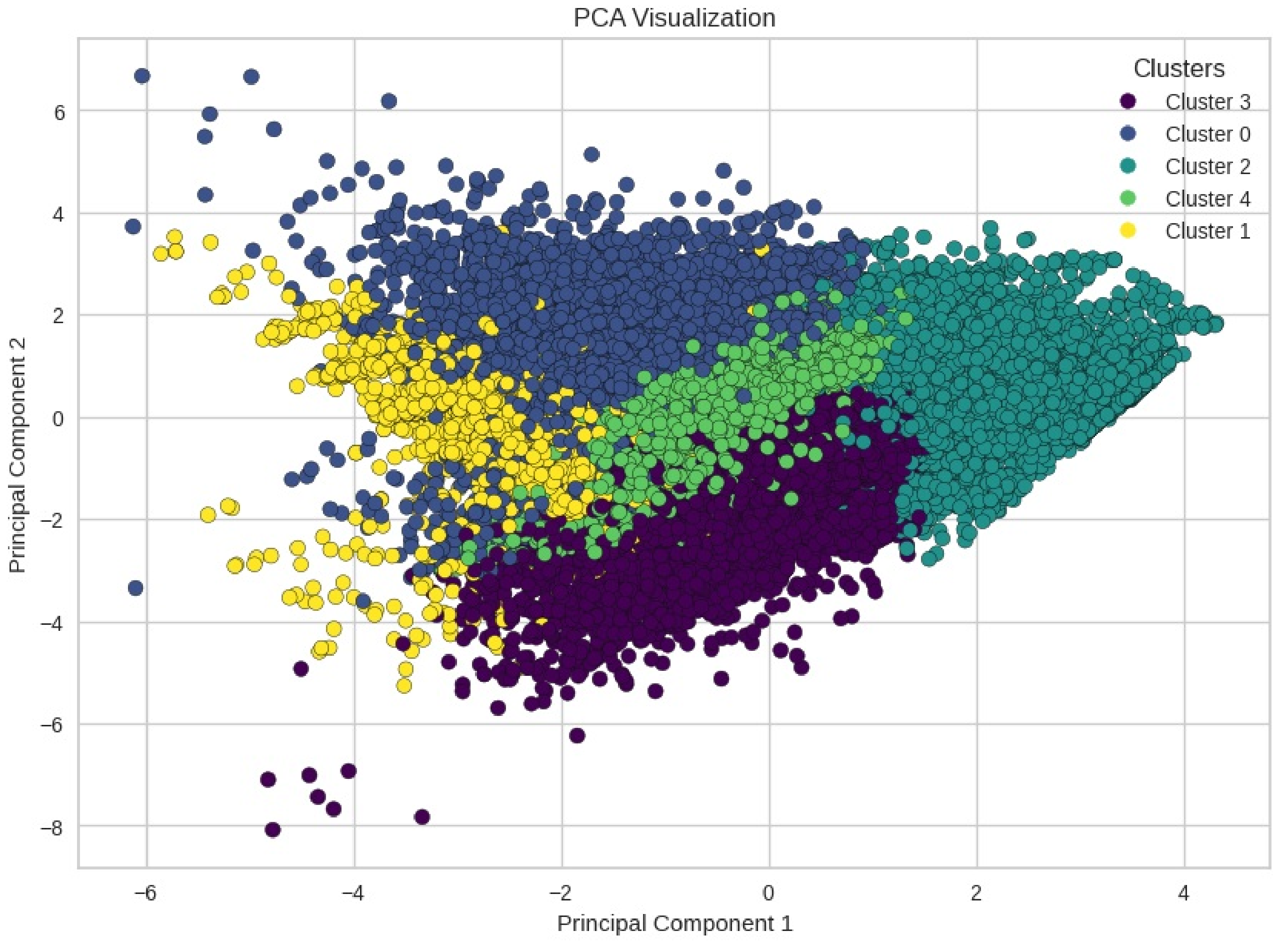
Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving as much variability (information) as possible. It helps simplify complex datasets and makes them easier to analyze and visualize.

In [113…
```
from sklearn.decomposition import PCA
```

In [114…
```
#visualization using PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(df_scaled_copy.drop(columns = 'kmeans_cluster'))

pca_result
```

Out[114…
```
array([[-0.42826287,  0.00388176],
       [-0.91106897, -0.45175586],
       [-0.97833403, -0.48143337],
       ...,
       [-1.04502574, -0.12673735],
       [-0.32062709,  1.25731839],
       [ 0.05629082,  0.54285537]])
```

In [115…
```
# Creating the components
df_scaled_copy['pca_one'] = pca_result[:, 0]
df_scaled_copy['pca_two'] = pca_result[:, 1]

# Get unique cluster labels
unique_clusters = df_scaled_copy['kmeans_cluster'].unique()

# Visualization with Legends
plt.figure(figsize=(10, 7))
scatter = plt.scatter(
    data=df_scaled_copy,
    x='pca_one',
    y='pca_two',
    c=df_scaled_copy['kmeans_cluster'],
    cmap='viridis',
    marker='o',
    edgecolor='k',
    s=50
)

# Adding Legend
plt.legend(
    handles=scatter.legend_elements()[0],
    labels=[f"Cluster {int(cluster)}" for cluster in unique_clusters],
    title="Clusters",
    loc='best'
)

# Add titles and labels
plt.title('PCA Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.show()
```



We can see that there are 5 clusters

In [ ]:

Cluster Profile and Characteristics

Cluster size and Distribution

We name the 5 clusters as cluster0, cluster1, cluster2, cluster3 and cluster4

In [116…

```
#cluster size
cluster_sizes = df5['kmeans_cluster'].value_counts().sort_index()
cluster_sizes
```

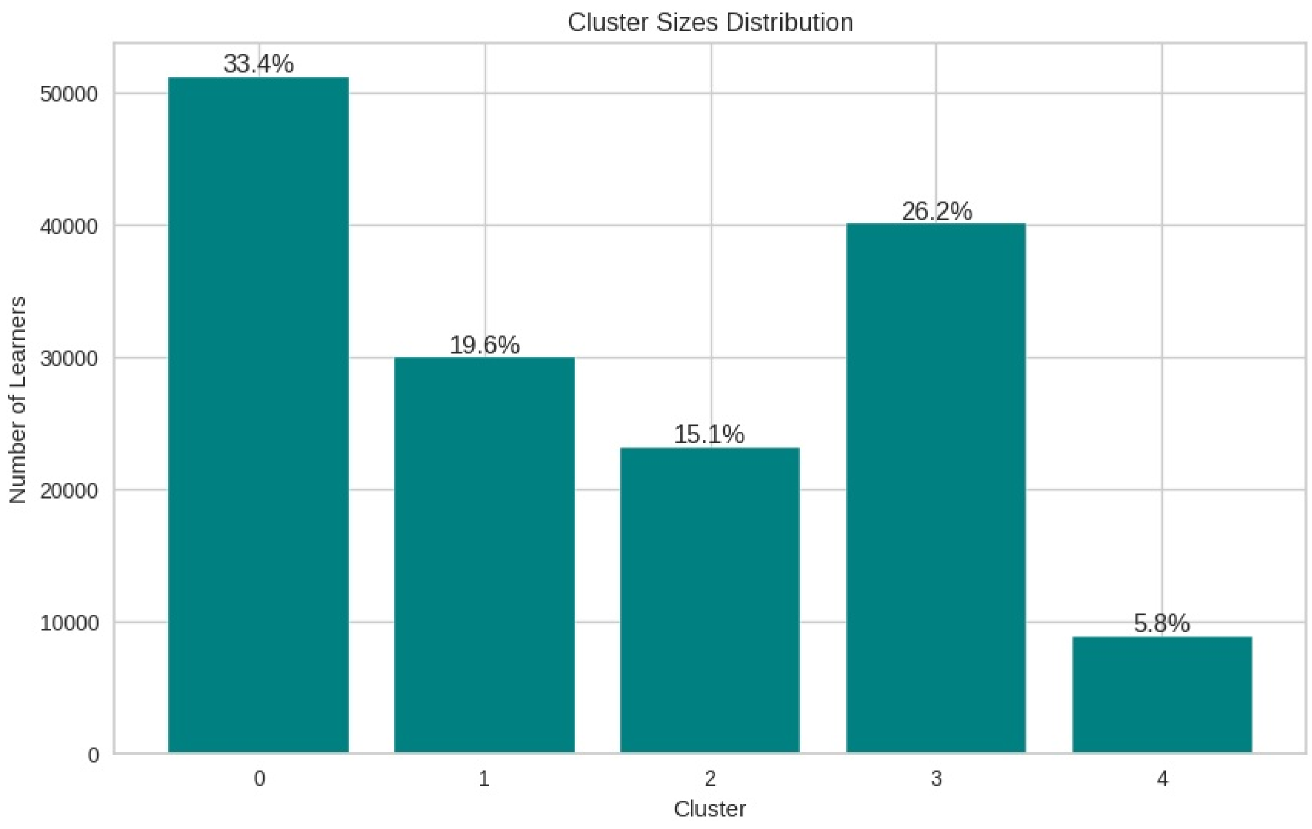
Out[116…

	count
kmeans_cluster	
0	51233
1	30007
2	23193
3	40150
4	8860

dtype: int64

In [117…

```
#distribution
# Assuming cluster sizes are stored in a dictionary
cluster_sizes = {0: 51233, 1: 30007, 2: 23193, 3: 40150, 4: 8860}

# Calculate the total number of learners
total_learners = sum(cluster_sizes.values())

# Create a bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(cluster_sizes.keys(), cluster_sizes.values(), color='teal')

# Add percentage labels above the bars
for bar in bars:
    height = bar.get_height()
    percentage = (height / total_learners) * 100
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{percentage:.1f}%', ha='center', va='bottom')

# Add labels and title
plt.xlabel('Cluster')
plt.ylabel('Number of Learners')
plt.title('Cluster Sizes Distribution')
plt.show()
```



The clustering analysis resulted in 5 distinct clusters with the following sizes:

- Cluster 0: 51,223 learners (33.4%)
- Cluster 1: 30,007 learners (19.6%)
- Cluster 2: 23,193 learners (15.1%)
- Cluster 3: 40,150 learners (26.2%)
- Cluster 4: 8,860 learners (5.8%)

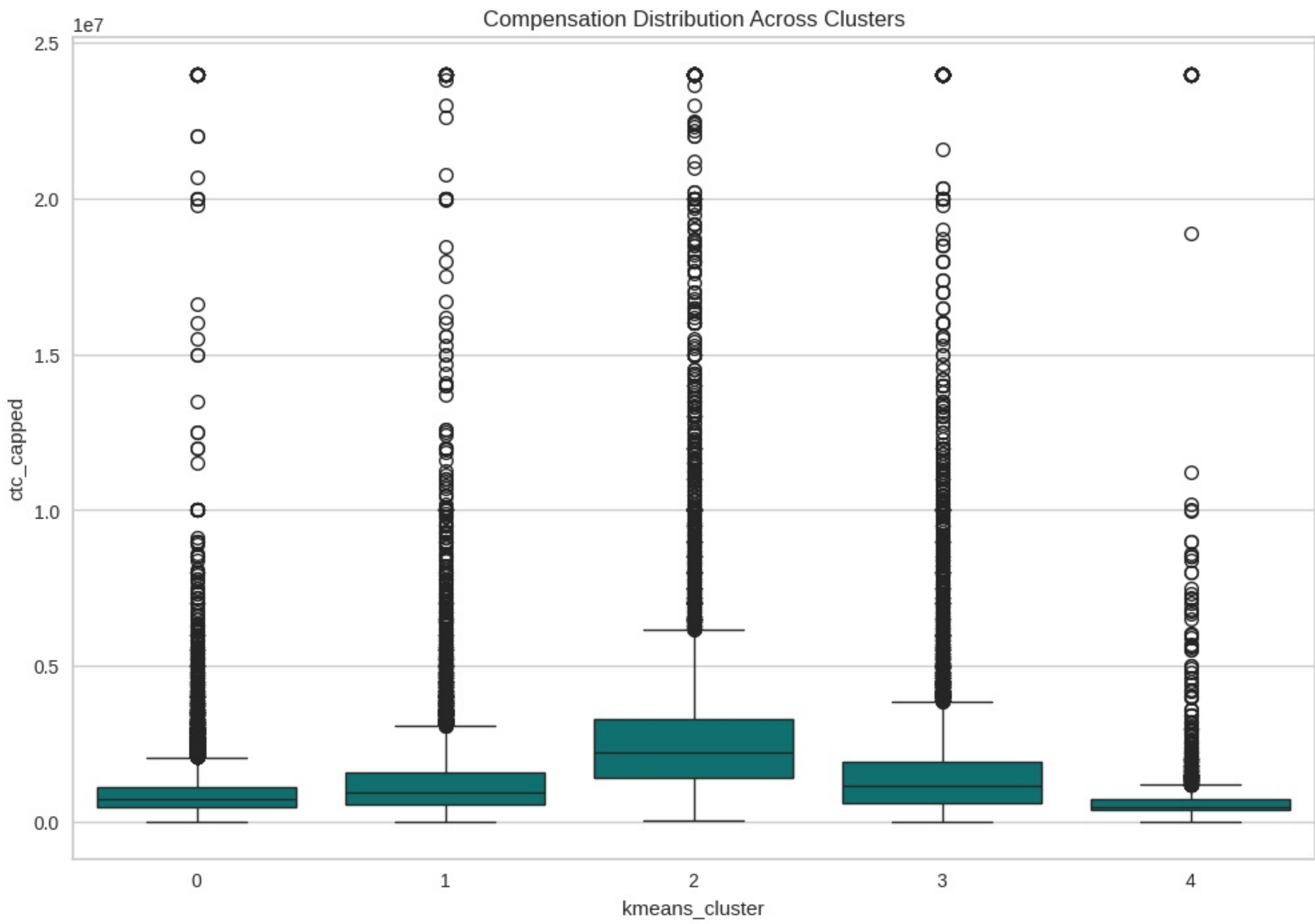
This distribution indicates that Cluster 0 is the largest segment, representing a significant portion of our learner base.

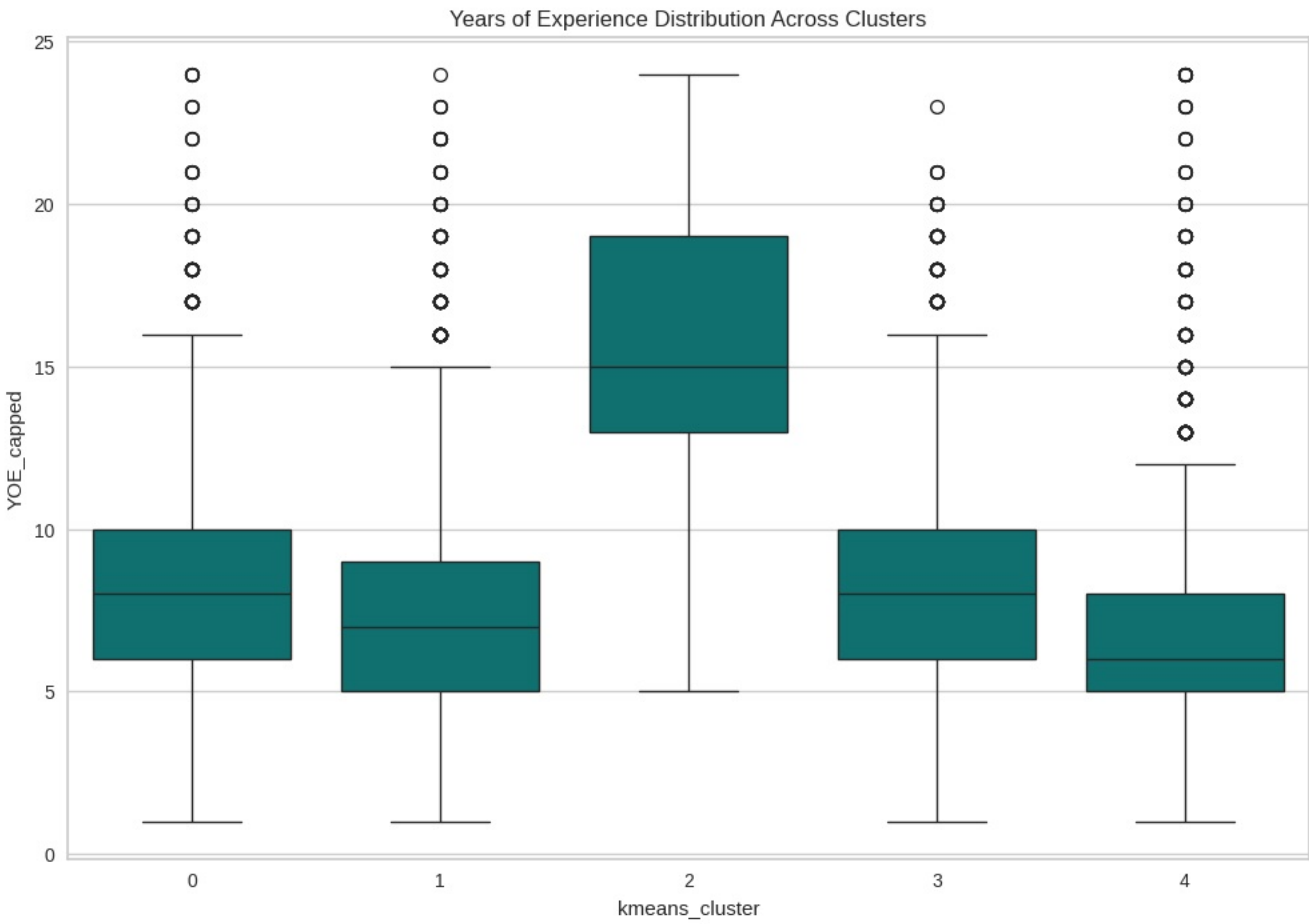
In [ ]:

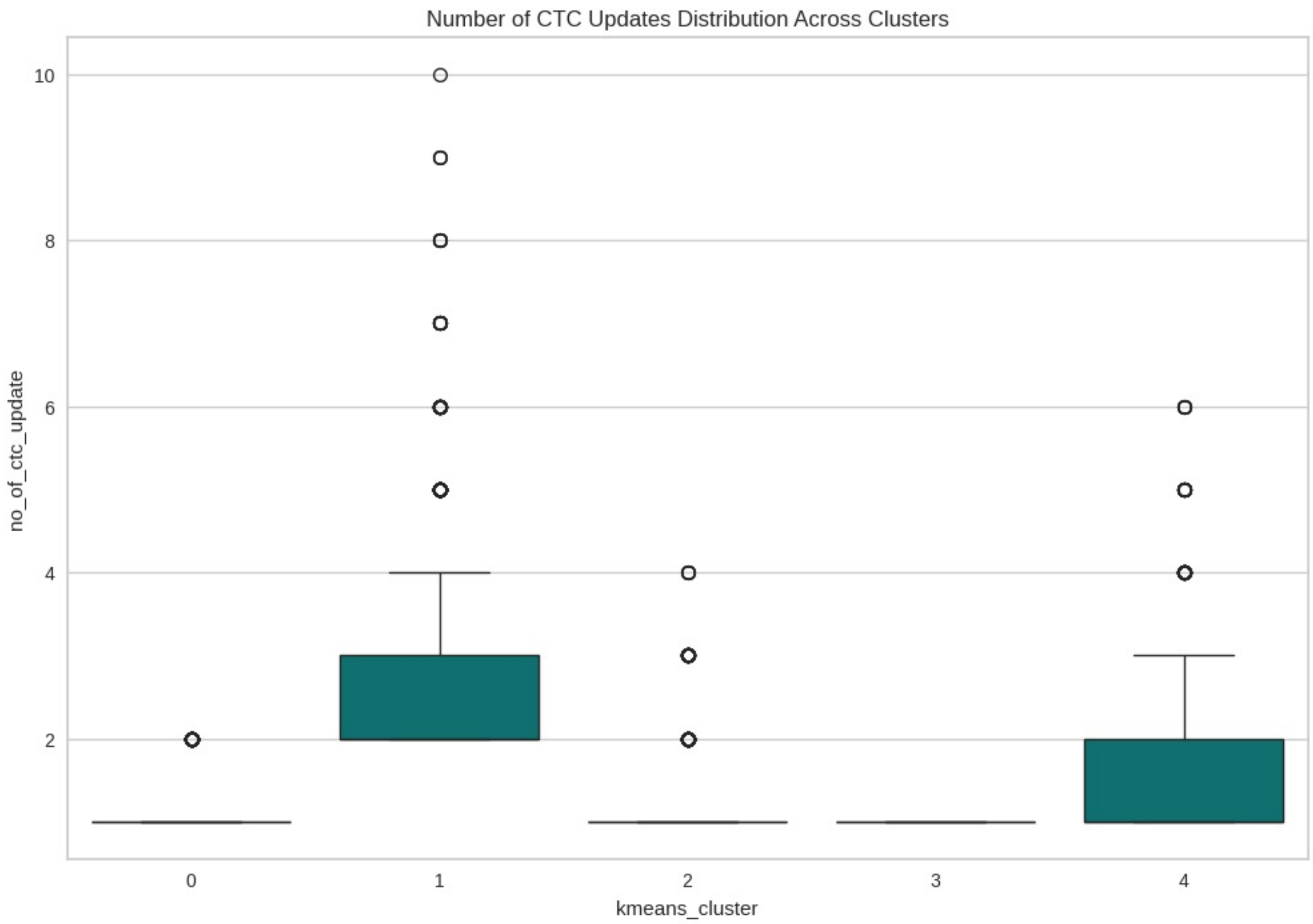
CTC / Years of Exp / CTC_updates distribution across Clusters

In [118…

```
plt.figure(figsize=(12, 8))
sns.boxplot(x='kmeans_cluster', y='ctc_capped', data=df5,color='teal')
plt.title('Compensation Distribution Across Clusters')
plt.show()

plt.figure(figsize=(12, 8))
sns.boxplot(x='kmeans_cluster', y='YOE_capped', data=df5,color='teal')
plt.title('Years of Experience Distribution Across Clusters')
plt.show()

plt.figure(figsize=(12, 8))
sns.boxplot(x='kmeans_cluster', y='no_of_ctc_update', data=df5, color='teal')
plt.title('Number of CTC Updates Distribution Across Clusters')
plt.show()
```


Compensation Distribution Across Clusters
1e7
2.5
2.0
1.5
1.0
0.5
0.0
ctc_capped
0
1
2
3
4
kmeans_cluster

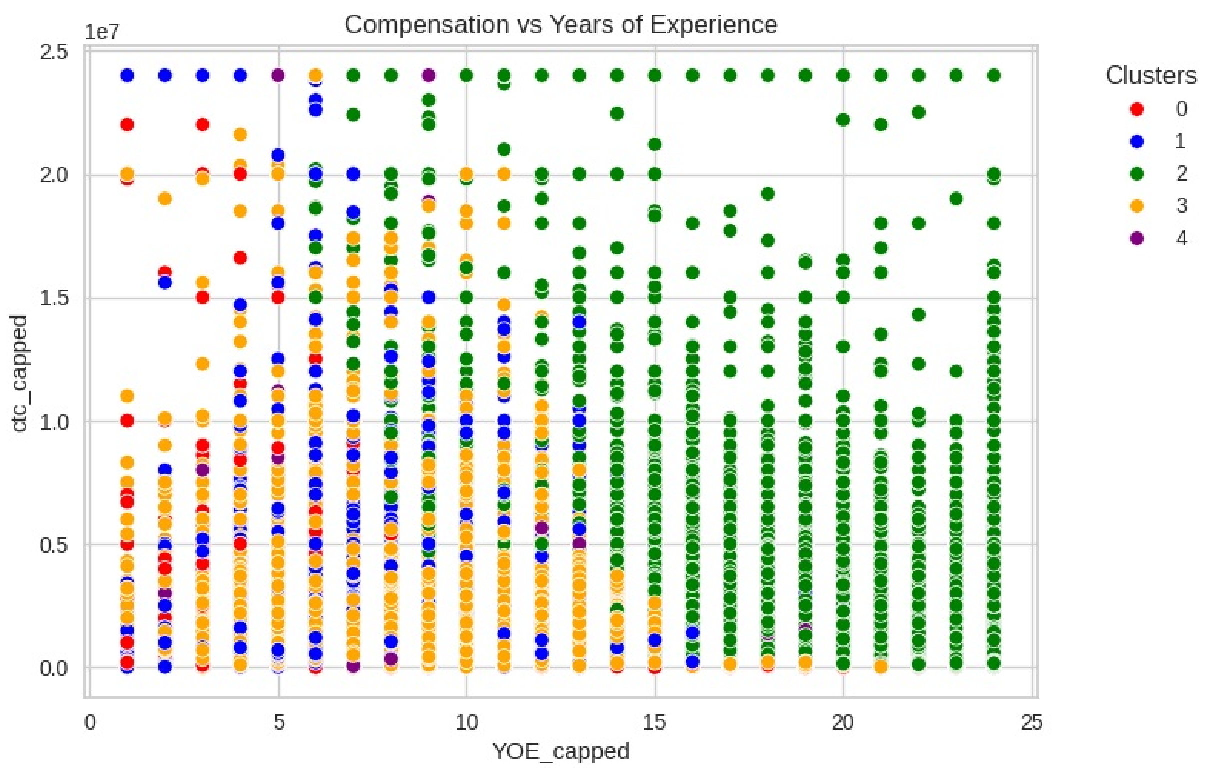

Years of Experience Distribution Across Clusters
25
20
15
10
5
0
YOE_capped
0
1
2
3
4
kmeans_cluster



- Compensation is high for cluster 2 followed by cluster 3
- Years of Experience is highest for cluster 2 followed by 0 and 3
- CTC_updates is high for cluster 1 followed by 4
- Compensation and Years of Exp is relatively higher for cluster 2

In [119…

```
#custom colors
custom_palette = ["red", 'blue', 'green', 'orange', 'purple']

#creating a scatterplot
sns.scatterplot(data = df5, x = 'YOE_capped', y = 'ctc_capped', hue = 'kmeans_cluster', palette = custom_palett
plt.title('Compensation vs Years of Experience')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', title='Clusters')
plt.show()
```



- Cluster 2 has relatively higher years of experience and compensation which was reflected from above box plots too
- Cluster 0 has lesser years of experience and w.r.t cluster 2 and most of the compensation is lower

Cluster Profiling

In [120…

```python
# Select only numeric columns for aggregation
numeric_columns = ['ctc_capped', 'YOE_capped', 'no_of_ctc_update']

# Calculate mean values for each cluster
cluster_averages = df5.groupby('kmeans_cluster')[numeric_columns].mean()

# Display the average values for each cluste
print(cluster_averages)
```

```
                  ctc_capped  YOE_capped  no_of_ctc_update
kmeans_cluster
0               9.197125e+05    8.047255          1.091172
1               1.310774e+06    7.611591          2.335655
2               3.445561e+06   16.030613          1.122149
3               1.684767e+06    8.215243          1.000000
4               8.139841e+05    7.026185          1.543679
```

In [121…

```python
from collections import Counter

# Function to get the most common job positions and companies in each cluster
def get_common_entries(df, cluster_label, column_name, top_n=3):
    cluster_data = df[df['kmeans_cluster'] == cluster_label]
    most_common_entries = Counter(cluster_data[column_name]).most_common(top_n)
    return most_common_entries

# Get profiles for each cluster
cluster_profiles = {}

for cluster in range(5):
    job_positions = get_common_entries(df5, cluster, 'job_position')
    companies = get_common_entries(df5, cluster, 'company_hash')

    cluster_profiles[cluster] = {
        'average_ctc': cluster_averages.loc[cluster, 'ctc_capped'],
        'average_yoe': cluster_averages.loc[cluster, 'YOE_capped'],
        'average_ctc_updates': cluster_averages.loc[cluster, 'no_of_ctc_update'],
        'common_job_positions': job_positions,
        'common_companies': companies
    }

# Display the profiles
for cluster, profile in cluster_profiles.items():
    print(f"Cluster {cluster}:")
    print(f"  Average Compensation (CTC): {profile['average_ctc']}")
    print(f"  Average Years of Experience: {profile['average_yoe']} years")
    print(f"  Average Number of CTC Updates: {profile['average_ctc_updates']}")
    print("  Common Job Positions:")
    for job, count in profile['common_job_positions']:
        print(f"    - {job}: {count} occurrences")
    print("  Common Companies:")
    for company, count in profile['common_companies']:
        print(f"    - {company}: {count} occurrences")
    print()
```

```
Cluster 0:
  Average Compensation (CTC): 919712.4957546893
  Average Years of Experience: 8.047254699119708 years
  Average Number of CTC Updates: 1.091171705736537
  Common Job Positions:
    - FullStack Engineer: 8540 occurrences
    - Other: 8013 occurrences
    - Frontend Engineer: 5328 occurrences
  Common Companies:
    - wgszxkvzn: 862 occurrences
    - vwwtznhqt: 711 occurrences
    - zgn vuurxwvmrt vwwghzn: 701 occurrences

Cluster 1:
  Average Compensation (CTC): 1310773.8996900723
  Average Years of Experience: 7.611590628853268 years
  Average Number of CTC Updates: 2.3356550138301064
  Common Job Positions:
    - unknown: 12848 occurrences
    - Backend Engineer: 7466 occurrences
    - FullStack Engineer: 5085 occurrences
  Common Companies:
    - zgn vuurxwvmrt vwwghzn: 771 occurrences
    - wgszxkvzn: 673 occurrences
    - vwwtznhqt: 604 occurrences

Cluster 2:
  Average Compensation (CTC): 3445561.4628982884
  Average Years of Experience: 16.03061268486181 years
  Average Number of CTC Updates: 1.122148924244384
  Common Job Positions:
    - Engineering Leadership: 4583 occurrences
    - FullStack Engineer: 1951 occurrences
    - Other: 1892 occurrences
  Common Companies:
    - gqvwrt: 341 occurrences
    - bxwqgogen: 293 occurrences
    - lubgqsvz wyvot  wg: 251 occurrences

Cluster 3:
  Average Compensation (CTC): 1684766.825180573
  Average Years of Experience: 8.21524283935243 years
  Average Number of CTC Updates: 1.0
  Common Job Positions:
    - Backend Engineer: 22411 occurrences
    - unknown: 16829 occurrences
    - FullStack Engineer: 825 occurrences
  Common Companies:
    - vbvkgz: 1116 occurrences
    - zgn vuurxwvmrt: 803 occurrences
    - zvz: 678 occurrences

Cluster 4:
  Average Compensation (CTC): 813984.141309255
  Average Years of Experience: 7.026185101580135 years
  Average Number of CTC Updates: 1.5436794582392777
  Common Job Positions:
    - unknown: 2938 occurrences
    - Backend Engineer: 1476 occurrences
    - Other: 1329 occurrences
  Common Companies:
    - nvnv wgzohrnvzwj otqcxwto: 5335 occurrences
    - xzegojo: 3392 occurrences
    - vbvkgz: 100 occurrences
```

In [ ]:

Q.Do the clusters formed align or differ significantly from the manual clustering efforts? If so, in what way?

In [122…
```python
df8 = df3.copy()
```

In [123…
```python
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
```

In [124…
```python
kmeans = KMeans(n_clusters = 5)
df8['Cluster'] = kmeans.fit_predict(df8[['ctc_capped', 'YOE_capped', 'Designation_Flag', 'Class_Flag', 'Tier_Fl


# Create a unique identifier for each combination of manual flags
df8['Manual_Cluster'] = df8['Designation_Flag'].astype(str) + df8['Class_Flag'].astype(str) + df8['Tier_Flag'].a

# Convert the combined string to a categorical variable and then to integer codes
```

```
df8['Manual_Cluster'] = pd.Categorical(df8['Manual_Cluster']).codes
```

- This code creates a manual cluster identifier based on a combination of three flags: Designation_Flag, Class_Flag, and Tier_Flag.
- First code Purpose:
  - Each unique combination of the three flags is treated as a separate category.
  - Example:
    - If Designation_Flag = 1, Class_Flag = 2, Tier_Flag = 3, the combined string will be "123".
    - Another row with Designation_Flag = 2, Class_Flag = 1, Tier_Flag = 3 will result in "213".
- Second code Purpose
  - Converts the combined strings (e.g., "123", "213") into categorical data.
  - Assigns a unique integer to each category:
    - For example:
      - "123" → 0
      - "213" → 1
      - "321" → 2
- The .codes attribute extracts these integers.

In [125…

```
# Adjusted Rand Index
ari = adjusted_rand_score(df8['Manual_Cluster'], df8['Cluster'])

# Normalized Mutual Information
nmi = normalized_mutual_info_score(df8['Manual_Cluster'], df8['Cluster'])

print(f'Adjusted Rand Index: {ari}')
print(f'Normalized Mutual Information: {nmi}')
```

```
Adjusted Rand Index: 0.09607474104818903
Normalized Mutual Information: 0.1266739128812287
```

Summary

The values of Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) provide insights into how similar the manual clustering is to the clusters obtained through unsupervised clustering:

Adjusted Rand Index (ARI):

Value: 0.095

Interpretation:

- ARI ranges from -1 to 1, where 1 indicates perfect agreement between the two clusterings, 0 indicates random agreement, and negative values indicate less than random agreement. An ARI of 0.095 suggests that there is a low level of agreement between the manual clustering and the unsupervised clustering. This means the clusters formed by the two methods are not very similar.

Normalized Mutual Information (NMI):

Value: 0.126

Interpretation:

- NMI ranges from 0 to 1, where 1 indicates perfect agreement and 0 indicates no agreement. An NMI of 0.126 indicates a low level of shared information between the manual clustering and the unsupervised clustering. This also suggests that the clusters formed by the two methods do not align well.

Implications:

Low Similarity:

- Both ARI and NMI values are quite low, indicating that the clusters formed by your manual clustering (using flags) and the clusters formed by the unsupervised method (e.g., KMeans) are significantly different.

Potential Reasons:

- This difference could be due to various factors, such as the criteria used for manual clustering not capturing the underlying structure of the data as effectively as the unsupervised method, or the unsupervised method uncovering patterns not evident through the manual criteria.

In [ ]:

Insights

Cluster 0:

- Individuals in this cluster have slightly more experience on average compared to Cluster 1.
- The average compensation is lower than Cluster 1.
- FullStack and Frontend Engineers are prevalent.
- The "Other" category indicates a diverse range of job positions.
- The companies overlap with those in Cluster 1, suggesting similar employer bases.

Cluster 1:

- This cluster consists of individuals with mid-level experience (around 7.6 years).
- They have a moderately high average compensation.
- Backend and FullStack Engineers are prominent roles apart from 'unknown'.
- Common employers include companies like zgn vuurxwvmrt, vwwghzn and wgszxkvzn.

Cluster 2:

- This cluster represents highly experienced professionals with significantly high compensation.
- The predominant role is in Engineering Leadership, indicating senior positions.
- The diversity in job positions (FullStack Engineer and Other) suggests a variety of responsibilities even among senior staff.
- The companies are distinct from those in other clusters, likely top-tier employers or specialized firms.

Cluster 3:

- This cluster has individuals with high compensation and above-average experience.
- Backend Engineers dominate this cluster, indicating a specialized skill set.
- There's a significant number of "unknown" job positions.
- Key employers include vbvkgz and zgn vuurxwvmrt.

Cluster 4:

- Individuals in this cluster have lower compensation and slightly less experience compared to other clusters.
- Backend Engineers are common, but there's also a significant "Other" category.
- The most frequent employers are nvnv wgzohrnvzwj otqcxwto and xzegojo, which are distinct from those in other clusters.

In [ ]:

```

```

Central Tendencies (Mean/Median) of Features

By examining the mean and median of features within each cluster, we gain a deeper understanding of the dominant characteristics:

Cluster 0:

- Similar experience to Cluster 1 but with lower compensation.
- High prevalence of FullStack and Frontend Engineers.

Cluster 1:

- Mid-level experience and moderately high compensation.
- Diverse job positions, predominantly in tech roles.

Cluster 2:

- Very high compensation and extensive experience.
- Leadership roles dominate, with a focus on senior positions.

Cluster 3:

- High compensation and specialized in Backend Engineering.
- Above-average experience, indicating skilled professionals.

Cluster 4:

- Lower compensation and slightly less experience.
- Backend Engineers are common, with significant data inconsistencies in job positions.

In [ ]:

```

```

1. What percentage of users fall into the largest cluster?

Ans: 33% of the learners fall into largest cluster 0.

2. Comment on the characteristics that differentiate the primary clusters from each other.

Ans:

Compensation:

- Cluster 2 has the highest average CTC, followed by Cluster 3, Cluster 1, Cluster 0, and Cluster 4.

Experience:

- Cluster 3 members have the most experience, significantly higher than other clusters.

Job Positions:

```
- Cluster 2 is dominated by leadership roles.

- Cluster 3 mainly comprises backend engineers.

- Cluster 0 has a mix of FullStack and Frontend engineers.

- Cluster 1 and Cluster 4 have a varied mix of job positions.
```

CTC Updates:

- Cluster 1 has the highest number of CTC updates, indicating more job movement or salary revisions.
- Cluster 2 and Cluster 3 have fewer CTC updates, indicating stability in roles.

3. Is it always true that with an increase in years of experience, the CTC increases? Provide a case where this isn't true.

Ans: No its not always true as shown in the plot earlier above. Avg. CTC is decreasing from 1 to 5 years of Experience. There might be a slight decrease in CTC with increasing experience, possibly due to industry-specific factors or career shifts.

4. Name a job position that is commonly considered entry-level but has a few learners with unusually high CTCs in the dataset.

Ans: Associate is the job position considered as entry level but maximum CTC going beyond set threshold

5. What is the average CTC of learners across different job positions?

Ans: job_position average_ctc

372 - Safety officer - 24000000.0

342 - Reseller - 24000000.0

288 - Owner - 24000000.0

593 - Telar - 24000000.0

.. ..

24 - Any technical - 10000.0

257 - Matlab programmer - 10000.0

641 - project engineer - 7900.0

189 - Full-stack web developer - 7500.0

273 - New graduate - 2000.0

[652 rows x 2 columns]

6. For a given company, how does the average CTC of a Data Scientist compare with other roles?

Ans:

- Average CTC of Data Scientist in ihvznuyx is 953333.3333333334 Average CTC of other roles in ihvznuyx is 900000.0

```
Data Scientist has higher average CTC in ihvznuyx
```

Similarly we can find for any company or all the companies.

7. Discuss the distribution of learners based on the Tier flag:

- Which companies dominate in Tier 1 and why might this be the case?
- Are there any notable patterns or insights when comparing learners from Tier 3 across different companies?

Ans:

- Companies Dominating in Tier 1

  Common Factors: Companies dominating Tier 1 might have a large number of entry-level positions or companies that offer lower-than-average compensation.

  Possible Reasons: Large enterprises with many junior or mid-level positions. Companies in traditional industries or smaller firms with limited budgets.

- Patterns in Tier 3 Across Different Companies

  High CTC Companies: Companies with a high number of Tier 3 learners might be in tech, finance, or other high-paying sectors.

  Career Progression: These companies might offer better career progression and compensation growth.

  Retention Strategy: Higher compensation could be a strategy to retain top talent.

8. After performing unsupervised clustering:

- How many clusters have been identified using the Elbow method?
- Do the clusters formed align or differ significantly from the manual clustering efforts? If so, in what way?

Ans:

5 clusters were identified using Elbow method. It differs w.r.t no. of clusters and tried statistical comparison as below.

The values of Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) provide insights into how similar the manual clustering is to the clusters obtained through unsupervised clustering:

Adjusted Rand Index (ARI):

Value: 0.095

Interpretation: ARI ranges from -1 to 1, where 1 indicates perfect agreement between the two clusterings, 0 indicates random agreement, and negative values indicate less than random agreement. An ARI of 0.096 suggests that there is a low level of agreement between the manual clustering and the unsupervised clustering. This means the clusters formed by the two methods are not very similar.

Normalized Mutual Information (NMI):

Value: 0.126

Interpretation: NMI ranges from 0 to 1, where 1 indicates perfect agreement and 0 indicates no agreement. An NMI of 0.125 indicates a low level of shared information between the manual clustering and the unsupervised clustering. This also suggests that the clusters formed by the two methods do not align well.

Implications:

Low Similarity: Both ARI and NMI values are quite low, indicating that the clusters formed by your manual clustering (using flags) and the clusters formed by the unsupervised method (e.g., KMeans) are significantly different.

Potential Reasons: This difference could be due to various factors, such as the criteria used for manual clustering not capturing the underlying structure of the data as effectively as the unsupervised method, or the unsupervised method uncovering patterns not evident through the manual criteria.

9. From the Hierarchical Clustering results:

- Are there any clear hierarchies or patterns formed that could suggest the different levels of seniority or roles within a company?
- How does the dendrogram representation correlate with the 'Years of Experience' feature?

Ans:

From the detailed analysis in previous section following is the summarized answer.

Cluster 0: Similar experience to Cluster 1 but with lower compensation. High prevalence of FullStack and Frontend Engineers.

Cluster 1: Mid-level experience and moderately high compensation. Diverse job positions, predominantly in tech roles.

Cluster 2: Very high compensation and extensive experience. Leadership roles dominate, with a focus on senior positions.

Cluster 3: High compensation and specialized in Backend Engineering. Above-average experience, indicating skilled professionals.

Cluster 4: Lower compensation and slightly less experience. Backend Engineers are common, with significant data inconsistencies in job positions

In [ ]:

Recommendations

Cluster 0:

- Average Compensation (CTC): ₹919712.49
- Average Years of Experience: 8.04 years
- Common Job Positions: FullStack Engineer, Frontend Engineer
- Common Companies: wgszxkvzn, vwwtznhqt

Recommendations:

1. Increase Purchase Frequency:

- Strategy: Introduce micro-credentials or nano-degrees for specific frontend and fullstack technologies. Offer bundle discounts for multiple courses.
- Example: "Frontend Developer Toolkit" package including courses on React, Angular, and Vue.js.

2. Retention Strategies:

- Strategy: Develop a points-based loyalty program where learners earn points for completing courses, which can be redeemed for additional courses or exclusive content.
- Example: Points system where 100 points can be redeemed for a free advanced course.

3. Targeted Marketing:

- Strategy: Highlight course bundles for fullstack and frontend technologies. Promote content that addresses common challenges and trends in these fields.
- Example: Blog posts and webinars on "The Future of Frontend Development".

Cluster 1:

- Average Compensation (CTC): ₹ 1310773.89
- Average Years of Experience: 7.61 years
- Common Job Positions: Backend Engineer, FullStack Engineer
- Common Companies: zgn vuurxwvmrt vwwghzn, wgszxkvzn

Recommendations:

1. Increase Purchase Frequency:

- Strategy: Offer advanced courses or certifications that build on existing skills. Introduce subscription-based learning models with new content released periodically to encourage ongoing participation.
- Example: Monthly or quarterly advanced backend or fullstack engineering workshops.

2. Retention Strategies:

- Strategy: Implement a mentorship program connecting mid-level professionals with more experienced mentors. Offer personalized career coaching sessions to help them navigate career growth.
- Example: Bi-monthly career coaching and mentoring sessions.

3. Targeted Marketing:

- Strategy: Focus marketing efforts on backend and fullstack engineering courses. Highlight success stories and case studies from learners in similar roles.
- Example: Email campaigns featuring testimonials from successful backend and fullstack engineers.

Cluster 2:

- Average Compensation (CTC): ₹3445561.46
- Average Years of Experience: 16.03 years
- Common Job Positions: Engineering Leadership, FullStack Engineer
- Common Companies: gqvwrt, bxwqgogen

Recommendations:

1. Increase Purchase Frequency:

- Strategy: Introduce executive education programs and leadership workshops tailored for senior professionals. Offer continuous

learning subscriptions for leadership content.
- Example: Annual subscription to "Executive Leadership Series".

2. Retention Strategies:

- Strategy: Implement an executive coaching program. Provide access to exclusive leadership forums and roundtable discussions with industry leaders.
- Example: Monthly executive coaching sessions and leadership roundtables.

3. Targeted Marketing:

- Strategy: Focus on leadership development programs and high-impact management courses. Highlight the ROI of these programs through case studies and testimonials.
- Example: Marketing campaigns showcasing leaders who achieved significant career milestones after completing Scaler’s leadership programs.

Cluster 3:

- Average Compensation (CTC): ₹1684766.82
- Average Years of Experience: 8.21 years
- Common Job Positions: Backend Engineer, FullStack Engineer
- Common Companies: vbvkgz, zgn vuurxwvmrt

Recommendations:

1. Increase Purchase Frequency:

- Strategy: Offer specialization tracks in advanced backend technologies, such as microservices, cloud computing, and big data. Create exclusive content available only to frequent learners.
- Example: "Mastering Microservices" specialization track.

2. Retention Strategies:

- Strategy: Provide access to exclusive webinars, industry talks, and networking events. Create a premium membership tier with added benefits.
- Example: Premium membership that includes quarterly industry webinars and access to an exclusive online community.

3. Targeted Marketing:

- Strategy: Emphasize advanced backend engineering content in marketing materials. Showcase the career advancement of learners who have completed these tracks.
- Example: Success stories of learners who transitioned to senior backend roles after completing advanced courses.

Cluster 4:

- Average Compensation (CTC): ₹813984.14
- Average Years of Experience: 7.03 years
- Common Job Positions: Backend Engineer, Other
- Common Companies: nvnv wgzohrnvzwj otqcxwto, xzegojo

Recommendations:

1. Increase Purchase Frequency:

- Strategy: Offer foundational and intermediate courses in various backend technologies. Provide frequent learners with incentives such as discounts on advanced courses.
- Example: Discounted rates for advanced courses upon completion of foundational courses.

2. Retention Strategies:

- Strategy: Develop a comprehensive career pathing tool that helps learners identify and achieve their career goals. Implement a regular feedback loop to improve course offerings based on learner input.
- Example: Personalized career pathing tool and quarterly feedback surveys with actionable improvements.

3. Targeted Marketing:

- Strategy: Promote a wide range of backend engineering courses, emphasizing career growth and skill enhancement. Use targeted ads on platforms frequented by mid-level professionals.
- Example: Ads on LinkedIn targeting backend engineers looking for career advancement.

Overall Recommendations

- Personalized Learning Paths: Implement personalized learning paths based on the cluster profiles. Utilize data to recommend courses that align with individual career goals and current industry trends.
- Exclusive Content and Membership Tiers: Develop exclusive content and membership tiers for high-value clusters, providing advanced learning opportunities and industry insights.
- Loyalty Programs and Incentives: Create loyalty programs to encourage continued learning and engagement. Offer incentives such as discounts, exclusive access, and career coaching.
- Targeted Marketing Campaigns: Design marketing campaigns that address the specific needs and preferences of each cluster. Use success stories, testimonials, and case studies to highlight the benefits of Scaler's programs.

By implementing these recommendations, Scaler can enhance its engagement with learners, improve retention rates, and maximize the ROI of its educational programs.

In [ ]:

```

```

Loading [MathJax]/extensions/Safe.js