# Business Case: Scaler – Clustering

**Problem statement :**

Scaler is an online tech-versity offering intensive computer science & Data Science courses through live classes delivered by tech leaders and subject matter experts. We need to cluster the sample segment of student data on the basis of their job profile, company and other features.

**Exploratory Data Analysis**

Q) Shape of the data set
Ans)

```
#shape of the data set
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (205843, 6)
```

Insights: There are a total of 2,05,843 rows in the data set.There are 6 features that are present for each of the records.

Q) What are the features and their data types
Ans)

```
[9] #features and their data types
    print("\nInformation about the dataset:\n", df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   company_hash      205799 non-null  object
 1   email_hash        205843 non-null  object
 2   orgyear           205757 non-null  float64
 3   ctc               205843 non-null  int64
 4   job_position      153279 non-null  object
 5   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(1), object(3)
memory usage: 9.4+ MB
```

Insights : There are 6 columns present in the data set. Three of them are object datatypes. Two of them are float data type and one integer data type.

Q) What are the statistical summary for the numerical attributes.
Ans)

```
#statistical summary
print("\nStatistical summary of numerical attributes:\n", df.describe())
```

```
Statistical summary of numerical attributes:
              orgyear           ctc  ctc_updated_year
count  205757.000000  2.058430e+05     205843.000000
mean     2014.882750  2.271685e+06       2019.628231
std        63.571115  1.180091e+07          1.325104
min         0.000000  2.000000e+00       2015.000000
25%      2013.000000  5.300000e+05       2019.000000
50%      2016.000000  9.500000e+05       2020.000000
75%      2018.000000  1.700000e+06       2021.000000
max     20165.000000  1.000150e+09       2021.000000
```

Insights: In this orgyear, ctc_updated_year can be overlooked since statistical summary for joining year of the candidates or the year where ctc got updated doesn't make sense. Hence we can focus on the CTC for the summary

Q) Which features have the missing values.
Ans)

```
#missing values
print("\n missing values):\n", df.isnull().sum())
```

```
 missing values):
 company_hash           44
email_hash              0
orgyear                86
ctc                     0
job_position        52564
ctc_updated_year        0
dtype: int64
```

Insights : There are around 52,564 cases of missing values in job_position feature. Along with company_hash and orgyear which are miniscule when compared to the missing values in job_position as well as the total number of records in the data set.

Q) Identify if there are emails that are unique / multiple records and their frequency.
Ans)

```
[25] # Check unique emails and frequency
     email_counts = df['email_hash'].value_counts()
     print("Total emails:", email_counts.shape[0])
     print("Unique Email counts : " ,  email_counts[email_counts == 1].shape[0])
     print("Count of Emails with multiple records:\n",  email_counts[email_counts > 1].shape)
     print("Emails with multiple records:\n",  email_counts[email_counts > 1])
```

```
Total emails: 153443
Unique Email counts :  112227
Count of Emails with multiple records:
 (41216,)
Emails with multiple records:
 email_hash
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee     9
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf93246d4192a89d8065     8
                                                                    ..
2001ec1f394b0e783e8368ebda4f913e98b4bf876a307d08c6ab9c90d6cf0069     2
e0580056b68a2566c4714afc0a0c4f04eec881fbb49bb62e542101dc7647315e     2
5e03a50d13d475e1ebdf82008abd5e1dc06a62f6e8df25c0fac4659fa67cad52     2
6b9d65aae5c59e401294f7c652e20f29c74e47d76877720736ae492b01774a08     2
b1e44894a7d09a75652cfa26c641e206f7fa8c58c7c1a10eca269b350404daef     2
Name: count, Length: 41216, dtype: int64
```

Insights : There are total of 153443 email ids in the dataset. There are total of 112227 with a count of 41216 emails with multiple records. When checked the data set, there are emails that have a count of 10 to 2.

Q) Convert the categorical columns to category data type.
Ans)

```
# Convert categorical columns to 'category' dtype
categorical_cols = ['email_hash', 'company_hash', 'job_position']
for col in categorical_cols:
    df[col] = df[col].astype('category')

print("\nUpdated data types:\n", df.dtypes)
```

```
Updated data types:
 company_hash        category
email_hash          category
orgyear              float64
ctc                    int64
job_position        category
ctc_updated_year     float64
dtype: object
```

Insights : The features / columns email_hash, company_hash and job_position into category type. This will help us to create an orderedness since there are many unique values that are repeated across the features of the data set.

Q) Remove the special characters from the features
Ans)

```
# Data Cleaning

# Remove special characters from Company_hash and Job_position using regex
df['company_hash'] = df['company_hash'].astype(str).apply(lambda x: re.sub('[^A-Za-z0-9 ]+', '', x))
df['job_position'] = df['job_position'].astype(str).apply(lambda x: re.sub('[^A-Za-z0-9 ]+', '', x))
```

Q) Check for duplicates and remove them
Ans)

```
# Check and drop duplicates
print("Duplicates:", df.duplicated().sum())
df = df.drop_duplicates()
```

```
Duplicates: 34
```

Insights : There are around 34 duplicates records in the data set. And all of them are dropped from the master data set.

Q) Imputations to be done for the features with missing values.

Ans)

```
[177] # updating the nan column with new string Missing since nan is the most frequenetly occuring value.
      df['job_position'] = df['job_position'].replace('nan','Missing')
```

```
[179] # --- Missing Value Imputation ---Impute missing values (if any) - For numeric columns use KNN Imputer
      num_cols = ['orgyear']
      imputer = KNNImputer(n_neighbors=5)
      df[num_cols] = imputer.fit_transform(df[num_cols])

      cat_missing = ['job_position','company_hash']

      freq_imputer = SimpleImputer(strategy = 'most_frequent') # mode
      for col in cat_missing:
          df[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(df[col])))
```

Insights : There are missing values present in the features.of which the most is in job_position. The most frequently occurring value for job_position is nan hence we cannot use SimpleImputer. Hence A new value is created as Missing and we replace all nan with value Missing. For column orgyear we use KNNimputation since it's a numerical column

Q) Update the feature orgyear and ctc_updated_year data type

Ans)

```
#orgyear column has year with values less / more than 3 digits. so replacing those items with the mode values
df['orgyear'] = df['orgyear'].astype(int)
df['orgyear'] = df['orgyear'].apply(
    lambda x: df['orgyear'].mode()[0] if len(str(x)) <= 3 else x
)
df['orgyear'] = df['orgyear'].apply(
    lambda x: df['orgyear'].mode()[0] if len(str(x)) >= 5 else x
)
```

```
[181] # Since  the data set has orgyear value more than 2025 - current year, updating those values.
      def correct_orgyear(year):
          if year > 2025:
              return 2000 + year % 10 # assuming that those values are years in 2000s
          return year

      # Apply the correction function to the 'orgyear' column
      df['orgyear'] = df['orgyear'].apply(correct_orgyear)
```

```
# data type change from float to int - ctc_updated_year
df['ctc_updated_year'] = df['ctc_updated_year'].astype(int)
df['ctc_updated_year'].unique()
```

```
array([2020, 2019, 2021, 2017, 2016, 2015, 2018])
```

Insights : Though the orgyear was imputed, the data is not completely accurate. Upon investigation it is found that there are values like 0, 2 3, etc i.e. single and double digits for year. As well as there are value which are more than the current year which doesn't make any sense. Hence first the float data type is converted to int for easier manipulation. And then if the number of digits are 3 or less, and also if the number of digits are more than 4, then we update the value with most frequent value of the feature – 2018. Now there are cases where the digits are 4 but its more than the current year. Those cases are considered to be typo scenario and they are updated as a year in the 2000s using the login in the function correct_orgyear. E.g. 2107 will be updated as 2007

The data type for ctc_updated_year is changed from float to Int. Upon investigation, we can understand that there is no discrepancy like orgyear for the values marked up

**FEATURE ENGINEERING**

Q) Create a new column named years_of_experience

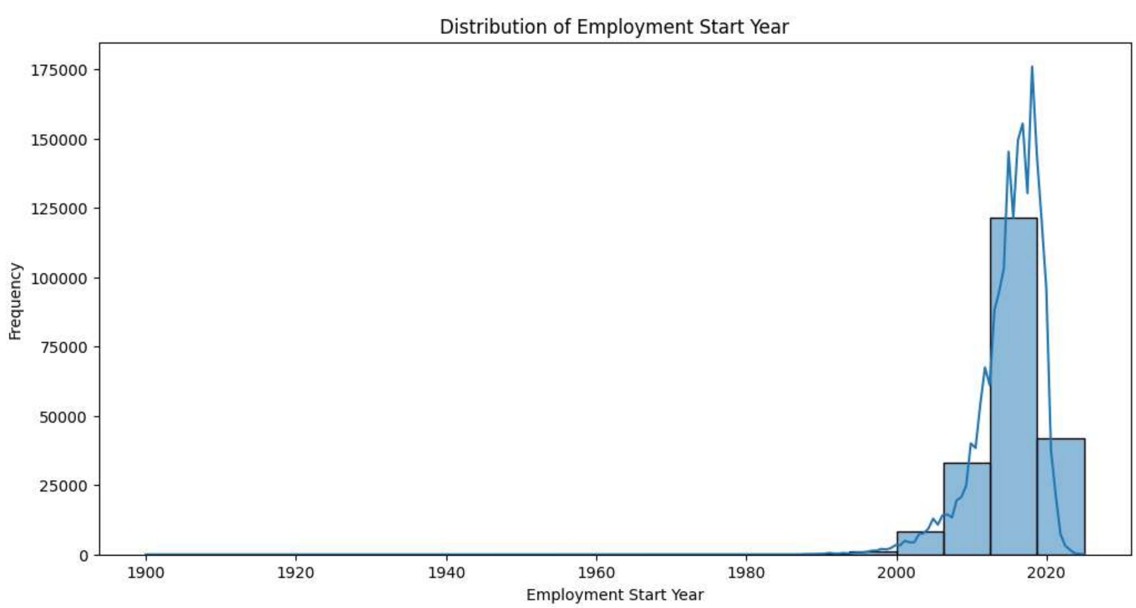
Ans)

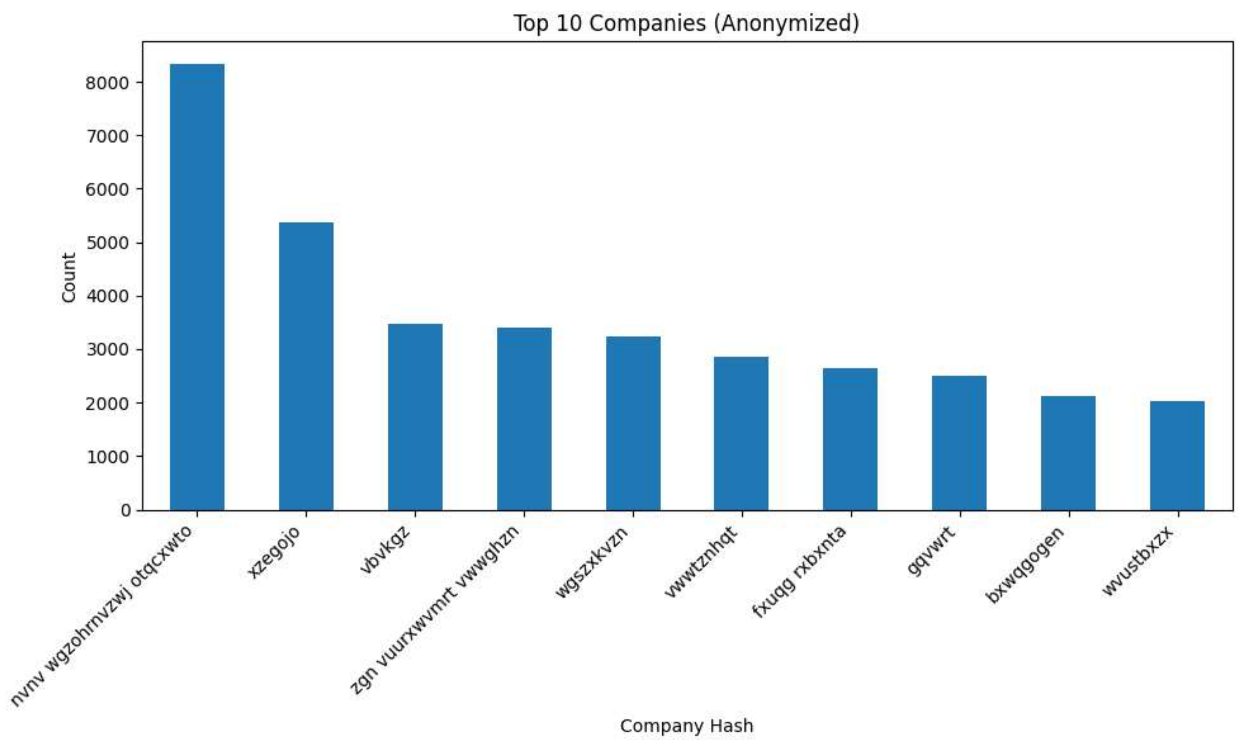
```
# --- Creating 'Years of Experience' Feature ---
current_year =  datetime.now().year # As per the problem context

df['years_of_experience'] = current_year - df['orgyear']
```

Insights : a new variable is created as current_year which will host the value of current year – 2025 – which is used to subtract from the joining date as to deduct the years_of_experience

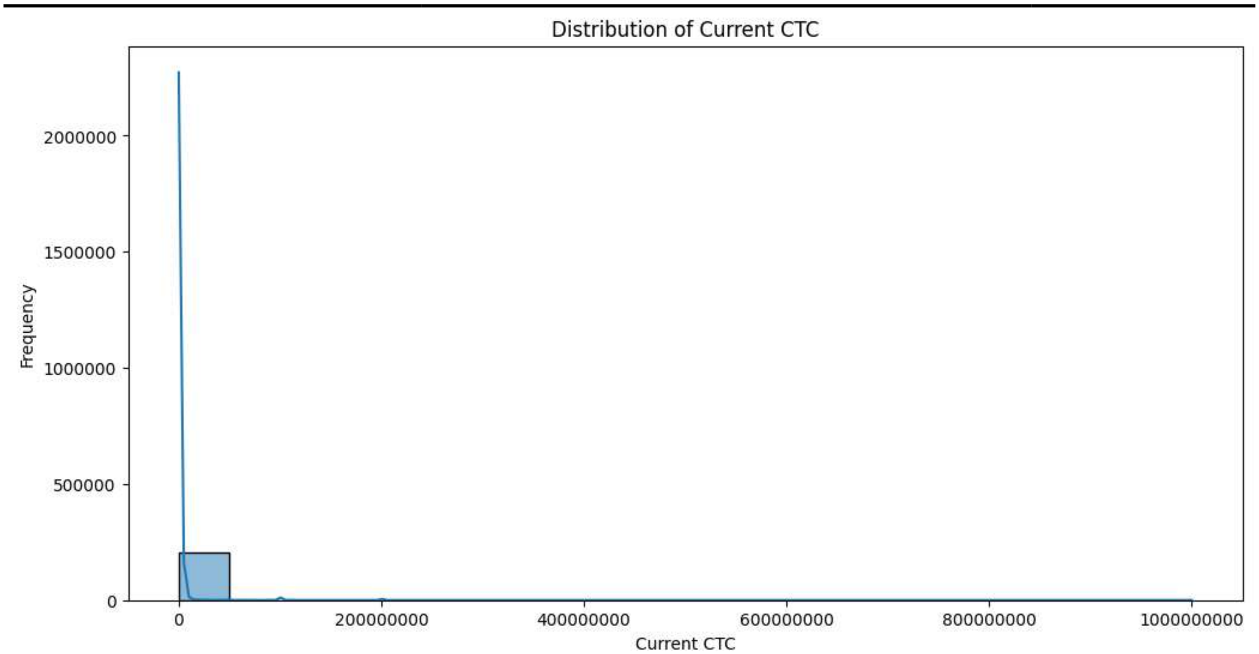
## Univariate Analysis

```
#Univariate analysis
plt.figure(figsize=(12, 6))
sns.histplot(df['orgyear'].dropna(), bins=20, kde=True)
plt.title('Distribution of Employment Start Year')
plt.xlabel('Employment Start Year')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(12, 6))
plt.ticklabel_format(style='plain')
sns.histplot(df['ctc'].dropna(), bins=20, kde=True)
plt.title('Distribution of Current CTC')
plt.xlabel('Current CTC')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(10, 6))
df['job_position'].value_counts().nlargest(10).plot(kind='bar')
plt.title('Top 10 Job Positions')
plt.xlabel('Job Position')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6))
df['company_hash'].value_counts().nlargest(10).plot(kind='bar')
plt.title('Top 10 Companies (Anonymized)')
plt.xlabel('Company Hash')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

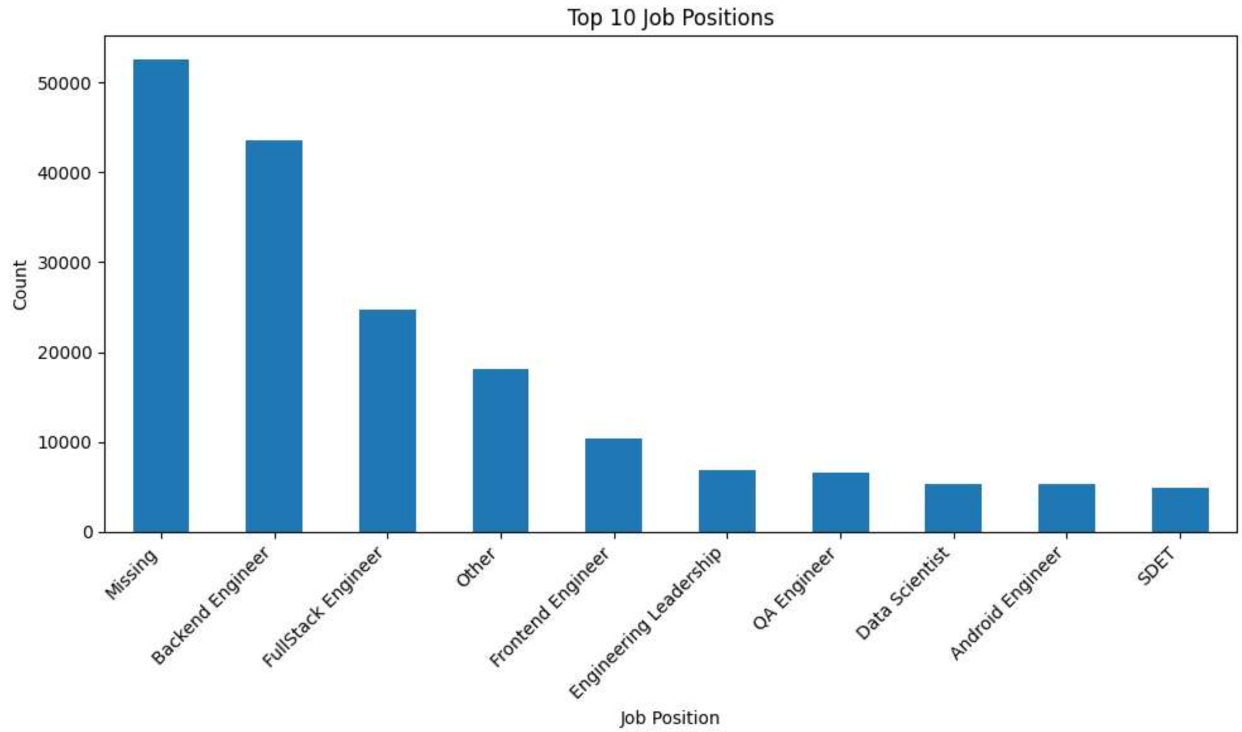




Top 10 Companies (Anonymized)
Count
8000
7000
6000
5000
4000
3000
2000
1000
0
nvnv wgzohrnvzwj otqcxwto
xzegojo
vbvkgz
zgn vuurxwvmrt vwwghzn
wgszxkvzn
vwwtznhqt
fxuqg rxbxnta
gqvwrt
bxwqgogen
wvustbxzx
Company Hash

Distribution of Current CTC
Frequency
2000000
1500000
1000000
500000
0
0
200000000
400000000
600000000
800000000
1000000000
Current CTC




Insights : In case of employment start year, The vast majority of learners in the dataset have relatively recent employment start years, primarily within the last two decades (2000s and 2010s). The years between approximately 2015 and 2020 seem to represent a period with a particularly high number of employment. The drop-off after 2020 might suggest that the data was collected relatively recently, and fewer learners in the dataset would have employment start years beyond that point. It could also reflect a change in the rate of new employment among the learners in the dataset. Absence of data before 2000 can be considered as a limitation of the historical data.

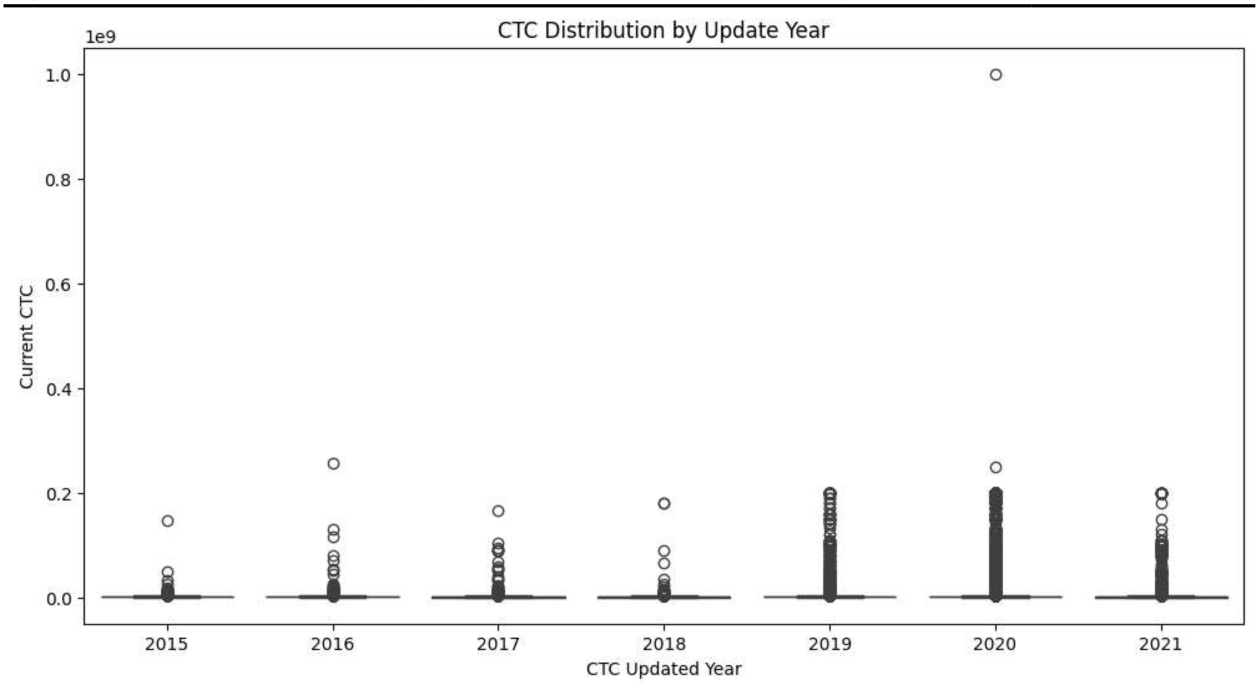
In case of Current CTC, The distribution is highly right-skewed. This means that the majority of the data points are concentrated on the left side. A significant number of learners have relatively low current CTCs. The presence of data points with extremely high CTCs strongly suggests the presence of outliers or a very small number of individuals in exceptionally high-paying roles. These are less frequent than those with lower CTCs.

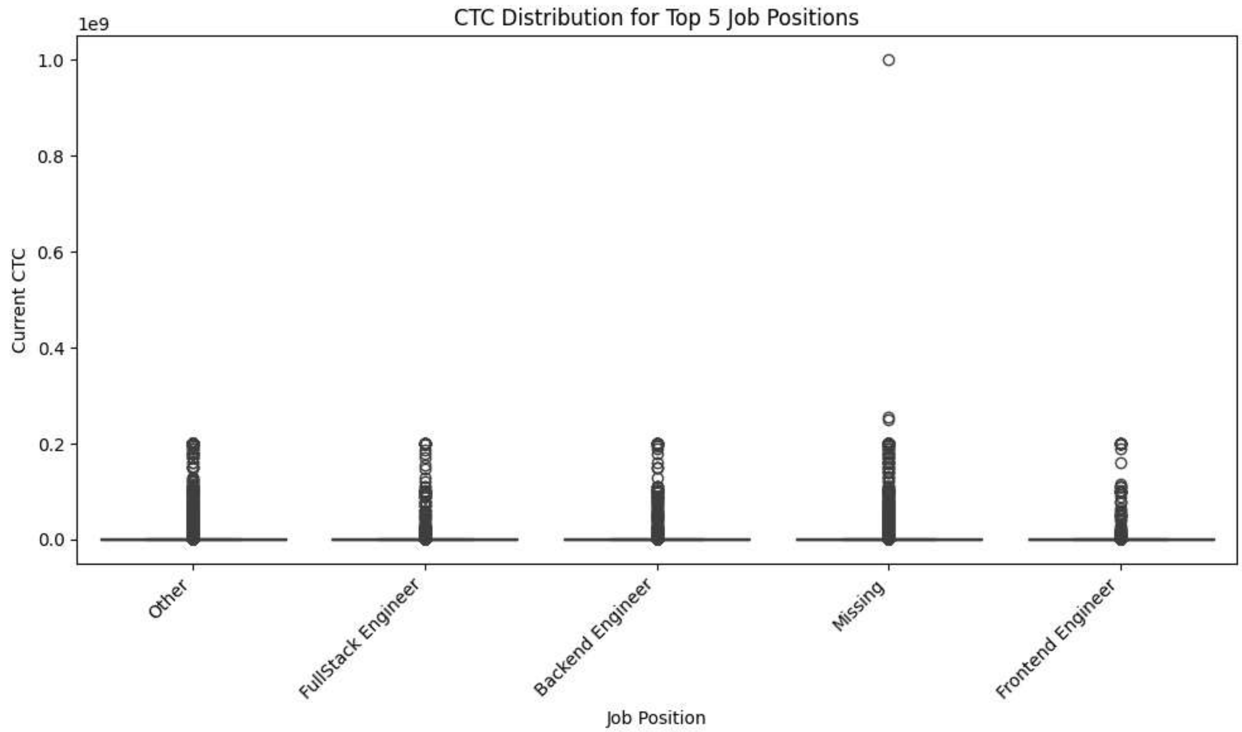
In case of job_position, the bar chart reveals that the "Missing" job position is the most frequent, highlighting a significant amount of missing data in this feature. Among the rest of the job positions, "Backend Engineer" and "FullStack Engineer" are the most common, followed by an "Other" category and then more specific tech roles with lower frequencies.

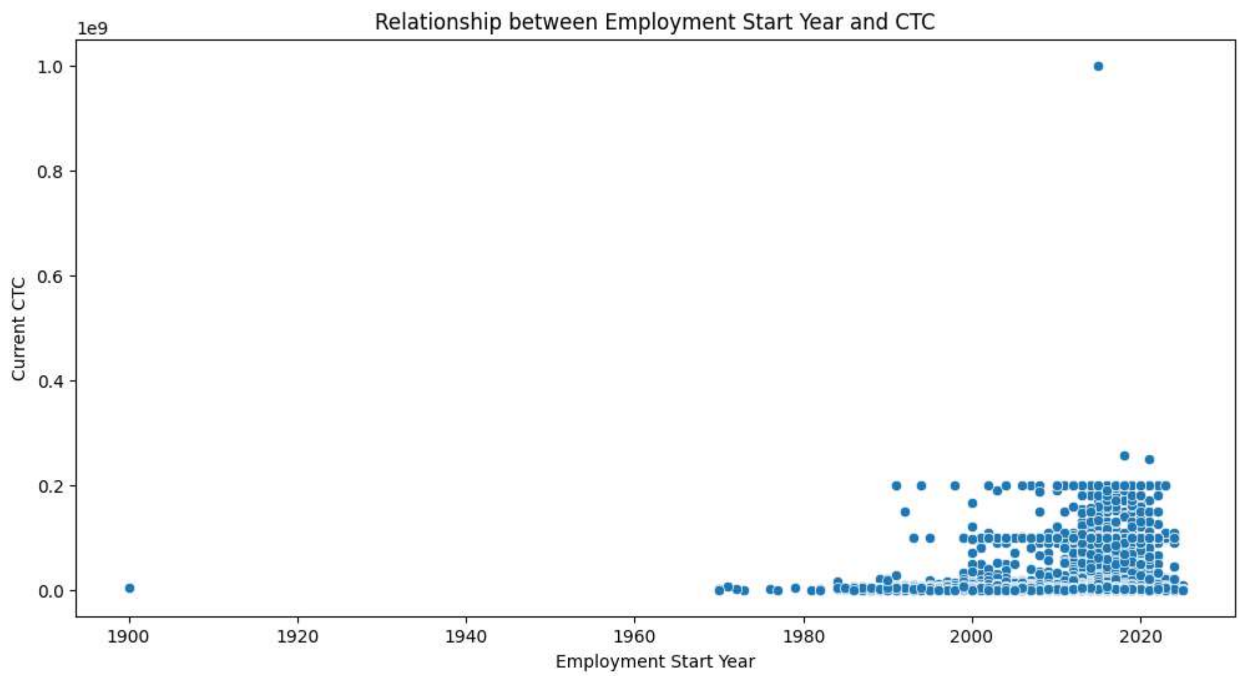
In case of company_hash, The company with the hash ‘nvmv wgzohnmzvj otqckwto’ has a significantly higher number of learners (over 8,000) compared to all other companies in the top 10. This suggests that a large proportion of the learners are currently employed at this particular company. The company with the hash ‘xzegojo’ is the second most represented, with a count of around 5,400 learners. This is considerably less than the top company but still a significant number. The companies with hashes ‘vbvkgz’, ‘zgn vuurxwv mrt vwwghzn’, and ‘wgszxkvzn’ have a relatively similar number of learners, ranging from approximately 3,200 to 3,500. The chart shows an uneven distribution of learners across the top 10 companies, with one company having a clear majority, followed by a step down to the second company, and then a more gradual decrease among the rest.

## Bivariate Analysis

```
# --- Bivariate Analysis ---
plt.figure(figsize=(12, 6))
sns.boxplot(x='ctc_updated_year', y='ctc', data=df.dropna())
plt.title('CTC Distribution by Update Year')
plt.xlabel('CTC Updated Year')
plt.ylabel('Current CTC')
plt.show()

# Since 'Company_hash' has many unique values, let's look at the relationship between top job positions and CTC.
top_jobs = df['job_position'].value_counts().nlargest(5).index
subset_df = df[df['job_position'].isin(top_jobs)].dropna(subset=['ctc'])
plt.figure(figsize=(10, 6))
sns.boxplot(x='job_position', y='ctc', data=subset_df)
plt.title('CTC Distribution for Top 5 Job Positions')
plt.xlabel('Job Position')
plt.ylabel('Current CTC')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
# Comment: Compare the CTC ranges for the most common job positions.

# Similarly, we can explore the relationship between 'orgyear' and 'CTC'.
plt.figure(figsize=(12, 6))
sns.scatterplot(x='orgyear', y='ctc', data=df.dropna())
plt.title('Relationship between Employment Start Year and CTC')
plt.xlabel('Employment Start Year')
plt.ylabel('Current CTC')
plt.show()
```






Insights:
In case of ctc v/s ctc_updated_year, The median CTC (the base line ) appears to be relatively stable across the different update years, generally hovering in the lower range of the CTC values. There might be a slight upward trend in the median CTC over the years, though not significant.

The IQR seems to increase slightly in the more recent years (2019, 2020, 2021) compared to the earlier years. This suggests a wider range of CTC values among learners whose CTC was updated more recently. There are presence of outliers all across the data. There is a particularly extreme outlier in the year 2020, with a CTC value approaching 1 billion. This single data point is far beyond the rest of the distribution for that year. While the median remains relatively stable, the increasing spread and the presence of higher outliers in recent years could indicate that while the typical CTC hasn't changed drastically, there might be more learners reaching higher salary levels in more recent updates.

In case of job_position v/s ctc, the box plot reveals differences in the 'Current CTC' distributions among the top 5 job positions. "FullStack" and "Backend Engineers" tend to have higher and more variable salaries, while "Frontend Engineers" and the "Other" category have lower medians. The "Missing" job position shows a wide salary range with extreme outliers.

In case of ctc v/s orgyear, scatter plot shows that while learners with more recent employment start years tend to have a wider range and generally higher 'Current CTC', the relationship is also influenced by other factors. The presence of outliers, particularly high earners in recent times, is also evident. There is low CTC in the early employment years, and as we towards more recent employment years, there is an increase in magnitude of CTC.

Q) Standardization & Encoding

```python
# --- Label Encoding for Categorical Features (for clustering later) ---
label_encoders = {}
categorical_cols_for_encoding = ['company_hash', 'job_position']
for col in categorical_cols_for_encoding:
    le = LabelEncoder()
    df[col + '_encoded'] = le.fit_transform(df[col])
    label_encoders[col] = le

print("\nSample of encoded categorical features:\n", df[['company_hash', 'company_hash_encoded', 'job_position', 'job_position_encoded']].head())

# --- Standardization for Numerical Features (for clustering later) ---
numerical_cols_for_scaling = ['ctc', 'years_of_experience']
scaler = StandardScaler()
for col in numerical_cols_for_scaling:
    # Reshape the column using values.reshape(-1, 1)
    df[col + '_scaled'] = scaler.fit_transform(df[[col]])
print("\nSample of scaled numerical features:\n", df[['ctc', 'ctc_scaled', 'years_of_experience', 'years_of_experience_scaled']].head())
```

```
Sample of encoded categorical features:
                 company_hash  company_hash_encoded        job_position  \
0             atrgxnnt xzaxv                   969               Other
1  qtrxvzwt xzegwgbb rxbxnta                 19730  FullStack Engineer
2               ojzwnvwnxw vx                 15512    Backend Engineer
3                   ngpgutaxv                 12108    Backend Engineer
4                  qxen sqghu                 20226  FullStack Engineer

   job_position_encoded
0                   455
1                   289
2                   138
3                   138
4                   289

Sample of scaled numerical features:
       ctc  ctc_scaled  years_of_experience  years_of_experience_scaled
0  1100000   -0.099295                    9                   -0.208606
1   449999   -0.154371                    7                   -0.680044
2  2000000   -0.023036                   10                    0.027113
3   700000   -0.133188                    8                   -0.444325
4  1400000   -0.073875                    8                   -0.444325
```

Insights: we have done standardization and encoding for the features ctc, years_of_experience company_hash and job_position. Label_encoding has been done for the categorical columns and each of the entries are given a unique encoded value. The scaling process has transformed the 'CTC' and 'Years of Experience' columns to have a mean of approximately 0 and a standard deviation of approximately 1.As these are successful encoded and standardized they are ready for clustering algorithms

## Manual Clustering

```python
# --- Manual Clustering and CTC Analysis ---
def analyze_ctc_group(group):
    return group['ctc'].agg(['mean', 'median', 'max', 'min', 'count'])

# --- 1. Company, Job Position, Years of Experience Level ---
grouped_co_job_exp = df.groupby(['company_hash', 'job_position', 'years_of_experience'])
ctc_summary_co_job_exp = grouped_co_job_exp.apply(analyze_ctc_group).reset_index()
ctc_summary_co_job_exp.rename(columns={'mean': 'avg_ctc_co_job_exp', 'median': 'median_ctc_co_job_exp'}, inplace=True)

df_merged_co_job_exp = pd.merge(df, ctc_summary_co_job_exp, on=['company_hash', 'job_position', 'years_of_experience'], how='left')

def designation_flag(row):
    if row['ctc'] > row['avg_ctc_co_job_exp']:
        return 1
    elif row['ctc'] < row['median_ctc_co_job_exp']:
        return 3
    else:
        return 2

df_merged_co_job_exp['Designation'] = df_merged_co_job_exp.apply(designation_flag, axis=1)
print("\nSample with Designation Flag:\n", df_merged_co_job_exp[['company_hash', 'job_position', 'years_of_experience', 'ctc', 'avg_ctc_co_job_exp',
                                                                'median_ctc_co_job_exp', 'Designation']].head())

print("\nDesignation Flag Value Counts:\n", df_merged_co_job_exp['Designation'].value_counts())
```

```
<ipython-input-26-a97aec29f3ed>:7: DeprecationWarning: DataFrameGroupBy.apply operated on the g
  ctc_summary_co_job_exp = grouped_co_job_exp.apply(analyze_ctc_group).reset_index()

Sample with Designation Flag:
                 company_hash        job_position  years_of_experience  \
0             atrgxnnt xzaxv               Other                    9
1  qtrxvzwt xzegwgbb rxbxnta  FullStack Engineer                    7
2              ojzwnvwnxw vx    Backend Engineer                   10
3                  ngpgutaxv    Backend Engineer                    8
4                 qxen sqghu  FullStack Engineer                    8

       ctc  avg_ctc_co_job_exp  median_ctc_co_job_exp  Designation
0  1100000        1.100000e+06              1100000.0            2
1   449999        7.742856e+05               750000.0            3
2  2000000        2.000000e+06              2000000.0            2
3   700000        1.037500e+06               950000.0            3
4  1400000        1.400000e+06              1400000.0            2

Designation Flag Value Counts:
 Designation
2    119531
3     47716
1     38562
Name: count, dtype: int64
```

Insights: The designation  2 appears 119,531 times. This is the most frequent value, suggesting that a large number of learners have a CTC that falls around the average/median for their specific company-job-experience group.
The designation   3 appears 47,716 times. This likely represents learners whose CTC is below the average/median for their group.
The designation  1 appears 38,652 times. This likely represents learners whose CTC is above the average/median for their group.
The high count for Designation 2 indicates that most learners CTC is close to the central tendency of their peer group defined by company, job, and experience. There are still substantial numbers of learners whose CTC is either above (Designation 1) or below (Designation 3) the average/median for their group, suggesting potential high and low performers or differences in compensation structures within those small segments. Hence we can summarise as The distribution shows that most learners are around the central tendency, but a significant portion deviates on both the higher and lower ends.

```python
# --- 2. Company & Job Position Level ---
grouped_co_job = df.groupby(['company_hash', 'job_position'])
ctc_summary_co_job = grouped_co_job.apply(analyze_ctc_group).reset_index()
ctc_summary_co_job.rename(columns={'mean': 'avg_ctc_co_job', 'median': 'median_ctc_co_job'}, inplace=True)

df_merged_co_job = pd.merge(df_merged_co_job_exp, ctc_summary_co_job, on=['company_hash', 'job_position'], how='left')

def class_flag(row):
    if row['ctc'] > row['avg_ctc_co_job']:
        return 1
    elif row['ctc'] < row['median_ctc_co_job']:
        return 3
    else:
        return 2

df_merged_co_job['Class'] = df_merged_co_job.apply(class_flag, axis=1)
print("\nSample with Class Flag:\n", df_merged_co_job[['company_hash', 'job_position', 'ctc', 'avg_ctc_co_job', 'median_ctc_co_job', 'Class']].head())

print("\nClass Flag Value Counts:\n", df_merged_co_job['Class'].value_counts())
```

```
<ipython-input-28-08398003fe5e>:3: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns
  ctc_summary_co_job = grouped_co_job.apply(analyze_ctc_group).reset_index()

Sample with Class Flag:
                company_hash        job_position      ctc  avg_ctc_co_job  \
0            atrgxnnt xzaxv               Other  1100000    1.085000e+06
1  qtrxvzwt xzegwgbb rxbxnta  FullStack Engineer   449999    5.514409e+06
2              ojzwnvwnxw vx    Backend Engineer  2000000    2.000000e+06
3                  ngpgutaxv    Backend Engineer   700000    1.159240e+06
4                 qxen sqghu  FullStack Engineer  1400000    1.054000e+06

   median_ctc_co_job  Class
0          1085000.0      1
1           875000.0      3
2          2000000.0      2
3          1050000.0      3
4          1400000.0      1

Class Flag Value Counts:
 Class
2    88515
3    70590
1    46704
Name: count, dtype: int64
```

Insights: Class 2 appears 88,515 times. This is the most frequent value, suggesting that a large number of learners have a CTC that falls around the average/median for their specific company and job position.

Class 3 appears 70,590 times. This likely represents learners whose CTC is below the average/median for their company and job position.

Class 1 appears 46,704 times. This likely represents learners whose CTC is above the average/median for their company and job position.

Similar to the 'Designation' flag, the most frequent value (2) indicates that a significant portion of learners have CTCs close to the central tendency of their company-job group. The counts for 'Class' 1 and 3 are also considerable, indicating a significant number of learners earning more or less than their peers at the company-job level.

```python
# --- 3. Company Level ---
grouped_co = df.groupby(['company_hash'])
ctc_summary_co = grouped_co.apply(analyze_ctc_group).reset_index()
ctc_summary_co.rename(columns={'mean': 'avg_ctc_co', 'median': 'median_ctc_co'}, inplace=True)

df_merged_co = pd.merge(df_merged_co_job, ctc_summary_co, on=['company_hash'], how='left')

def tier_flag(row):
    if row['ctc'] > row['avg_ctc_co']:
        return 1
    elif row['ctc'] < row['median_ctc_co']:
        return 3
    else:
        return 2

df_merged_co['Tier'] = df_merged_co.apply(tier_flag, axis=1)
print("\nSample with Tier Flag:\n", df_merged_co[['company_hash', 'ctc', 'avg_ctc_co', 'median_ctc_co', 'Tier']].head())

print("\nTier Flag Value Counts:\n", df_merged_co['Tier'].value_counts())
```

```
<ipython-input-30-5bd9789d32fa>:3: DeprecationWarning: DataFrameGroupBy.apply operated on the g
  ctc_summary_co = grouped_co.apply(analyze_ctc_group).reset_index()

Sample with Tier Flag:
                 company_hash      ctc    avg_ctc_co  median_ctc_co  Tier
0             atrgxnnt xzaxv  1100000  1.087778e+06      1070000.0     1
1  qtrxvzwt xzegwgbb rxbxnta   449999  2.296193e+06       900000.0     3
2              ojzwnvwnxw vx  2000000  2.000000e+06      2000000.0     2
3                  ngpgutaxv   700000  1.452014e+06      1100000.0     3
4                 qxen sqghu  1400000  1.418667e+06      1550000.0     3

Tier Flag Value Counts:
 Tier
3    83424
2    75232
1    47153
Name: count, dtype: int64
```

Insights: In this scenario,
Tier 3 appears 83,424 times. This is the most frequent value, suggesting that a large number of learners have a CTC that falls below the average/median for their company.
Tier 2 appears 75,232 times. This is the second most frequent value, indicating that a substantial number of learners have a CTC that is around the average/median for their company.
Tier 1 appears 47,153 times. This is the least frequent value, representing learners whose CTC is above the average/median for their company.
The most frequent value is 3 (below average/median), suggesting that within companies, there are more learners with CTCs on the lower end of the group compared to those significantly above the average/median. The least frequent value is 1 (above average/median), indicating that, at a company-wide level, fewer employees earn significantly more than their peers. Hence The distribution suggests that more learners tend to have CTCs below the company-wide average/median, with fewer individuals earning significantly more. This flag provides the broadest perspective on relative compensation within organizations.

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

```python
# Top 10 employees earning more than most in company (Tier 1)
top_tier1 = df_merged_co[df_merged_co['Tier'] == 1].sort_values('ctc', ascending=False).head(10)
print("Top 10 employees (Tier 1):")
print(top_tier1[['email_hash', 'company_hash', 'job_position', 'ctc', 'Tier']])
```

```
Top 10 employees (Tier 1):
                                               email_hash  \
117626  5b4bed51797140db4ed52018a979db1e34cee49e27b488...
22387   94970774b1cf64e61cf30fb6541cd27fdb31b220cda54b...
22286   3e7804b3aef9f10977903287530bb816dcde2d98e87bf3...
22185   97d25613e7bc3f47c87492d311f77232c105e4bc9ce642...
36253   1bdd2d3f1509045bd303e67882df623b0f892d0509b6e8...
22089   f4e874b3329098fdb3de47a83e1b41b2f5f4b873e148dd...
19712   59316048d113539202325e05af9b66620255ba84eab635...
126190  0c9c37269bd373ef507df0bc1bb318787fd895c858b74e...
10401   74f506e2567fb54995842894d2021582effbcde027d8e3...
99280   2744c7f42fd4d492fa66cb2ba5168921c444dc8611ffa2...

                     company_hash            job_position        ctc  Tier
117626               wxowg ojontbo                 Missing  255555555     1
22387                           zv  Engineering Leadership  200000000     1
22286                  exqonoghqwt            Data Analyst  200000000     1
22185   nvnv wgzohrnvzwj otqcxwto        Support Engineer  200000000     1
36253             ywr ntwyzgrgsxto        Android Engineer  200000000     1
22089                 xqgz bghznvxz                   Other  200000000     1
19712   nvnv wgzohrnvzwj otqcxwto            Data Analyst  200000000     1
126190                      sggsrt                 Missing  200000000     1
10401              mtwngz axwpxzogz             QA Engineer  200000000     1
99280                     bxwqgogen        Android Engineer  200000000     1
```

Insights : Since we don't have the employee details other than email_hash, the same is displayed in the result. WE can see that Engineering leadership, data analyst are some of the job position who are earning more than most in the company.

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

```python
# Top 10 data science employees earning more than peers (Class 1)
ds_top_class1 = df_merged_co_job[(df_merged_co_job['job_position'].str.contains('data', case=False, na=False)) & (df_merged_co_job['Class'] == 1)]
top_10_ds_class1 = ds_top_class1.groupby('company_hash').apply(lambda x: x.sort_values(by = 'ctc', ascending=False).head(10)).reset_index(drop=True).sort_values('ctc', ascending=False).head(1
print("Top 10 Data Science employees per company (Class 1):")
print(top_10_ds_class1[['email_hash', 'company_hash', 'job_position', 'ctc', 'Class']])
```

```
Top 10 Data Science employees per company (Class 1):
                                             email_hash  \
1637  268a5aa92f0b6d0c675fc9cc1e300eb0c5930a3a139a23...
689   f5b2a30853a67e1703249db6003884d7e1ae69e0c03aa0...
605   979d02840c45c1d5790306130a0977aab05f2bd2679687...
633   655da5cd99f1ba4ad249dade5039b914023484fb7f3959...
851   35d4845547c5d2e0c2eadc197c97c678035bceb5fddd2d...
1300  9ce2995b2221fe627e861daea9d0603872cce8cc128390...
1252  2f9a4241053f76b2f8c50ea593a90586d38b3f0e08c141...
374   6d4a5d19e889596252b038ee0409510aec8c0b32007fb9...
1290  aad581a532f319c76c6e73937572feed9867d5ee2f1093...
350   89f343bf01094accb8b0b2c799499daf6bf881321db2e4...

                   company_hash    job_position        ctc  Class
1637                       zgzt  Data Scientist  200000000      1
689                ogwxn szqvrt    Data Analyst  200000000      1
605              ntrtutqegqbvzwt    Data Analyst  200000000      1
633   nvnv wgzohrnvzwj otqcxwto    Data Analyst  200000000      1
851                         qmo    Data Analyst  200000000      1
1300                   wgzahtzn    Data Analyst  200000000      1
1252                  vwwtznhqt    Data Analyst  200000000      1
374                       gnytq    Data Analyst  200000000      1
1290                  wgszxkvzn    Data Analyst  200000000      1
350               fxuqg rxbxnta    Data Analyst  200000000      1
```

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

```
# Bottom 10 data science employees earning less than peers (Class 3)
ds_bottom_class3 = df_merged_co_job[(df_merged_co_job['job_position'].str.contains('data ', case=False, na=False)) & (df_merged_co_job['Class'] == 3)]
bottom_10_ds_class3 = ds_bottom_class3.groupby('company_hash').apply(lambda x: x.sort_values(by='ctc').tail(10)).reset_index(drop=True).sort_values('ctc', ascending=False).tail(10)
print("Bottom 10 Data Science employees per company (Class 3):")
print(bottom_10_ds_class3[['email_hash', 'company_hash', 'job_position', 'ctc', 'Class']])
```

```
Bottom 10 Data Science employees per company (Class 3):
                                             email_hash  \
1102  c5b586cc2d3b9e783e76763f274c6fbb05e7fabb12fbcc...
324   ab2dc9db23c3104f0b6b3dbd4cdd5bfb9e5829b8b7943d...
716   bd9c04a574090e05b366a81cdb2f3f565d0c60fa8b1647...
1666  aeb32d3e07a73c021c6ad75a3eebef4bedc726109d853b...
915   13fca3a2e659a7641ac165c4e649947398233b309c6495...
189   690f6fdab1ab7514a6a9325ebd6cfe910dbf12d46b6fde...
1568  648975bbd733a9949d715ba66d2712d0c01ace6e046c9a...
974   8001bc017fbe95541d23f5780c3edb988b7d9b2225e39e...
487   4af25f1052f845426450ad6d96e78338c3b913e3cd4539...
991   4c029c8afc9c245b4300d08f2cc0ccde425aa1a620debe...

                         company_hash    job_position   ctc  Class
1102                 uhmrxwxo ovuxtzn    Data Analyst  7500      3
324         exznqhon ogrhnxgzo ucn rna  Data Scientist  7200      3
716                           onhatzn  Data Scientist  6000      3
1666                      zthonvq xzw    Data Analyst  5000      3
915             rvntzncxtf vzvrjnxwo    Data Analyst  5000      3
189              bxyhu wgbbhzxwvnxgz  Data Scientist  4000      3
1568                      yn btaxv rna  Data Scientist  4000      3
974   srgmvrtast  xzntrrxstzwt ge nyxzso  Data Scientist  4000      3
487                             ihxpq  Data Scientist  4000      3
991                         stzj rvmo  Data Scientist  3500      3
```

Q) Bottom 10 employees (earning less than most of the employees in the company)- Tier 3
Ans)

```
# Bottom 10 employees earning less than most in company (Tier 3)
bottom_tier3 = df_merged_co[df_merged_co['Tier'] == 3].sort_values('ctc').head(10)
print("Bottom 10 employees (Tier 3):")
print(bottom_tier3[['email_hash', 'company_hash', 'job_position', 'ctc', 'Tier']])
```

```
Bottom 10 employees (Tier 3):
                                               email_hash  \
118226  f2b58aeed3c074652de2cfd3c0717a5d21d6fbcf342a78...
114157  23ad96d6b6f1ecf554a52f6e9b61677c7d73d8a409a143...
184918  b8a0bb340583936b5a7923947e9aec21add5ebc50cd60b...
183776  75357254a31f133e2d3870057922feddeba82b88056a07...
116938  f7e5e788676100d7c4146740ada9e2f8974defc01f571d...
166375  c411a6917058b50f44d7c62751be9b232155b23211de4c...
171173  80ba0259f9f59034c4927cf3bd38dc9ce2eb60ff18135b...
150664  9af3dca6c9d705d8d42585ccfce2627f00e1629130d14e...
99417   b995d7a2ae5c6f8497762ce04dc5c04ad6ec734d70802a...
147787  299f764fcae62f331f3c5eb1b451e7107302ded46e2a71...

                          company_hash          job_position   ctc  Tier
118226              zgpxv wgqugqvnxgz                 Other     6     3
114157                       grd sqghu               Missing    14     3
184918                        buyvoxo               Missing    15     3
183776                         tbxao               Missing    16     3
116938  urvj svbto24d7 uqxcvnt rxbxnta               Missing   200     3
166375                        rxnbho7  Engineering Leadership   300     3
171173                     xzegqbvnxwv      Backend Engineer   600     3
150664                        xzegojo               Missing   600     3
99417         nvnv wgzohrnvzwj otqcxwto     FullStack Engineer   600     3
147787  tznqvjz tahwvnxgz ntwyzgrgsxto     FullStack Engineer  1000     3
```

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

Ans)

```python
# Top 10 employees in each company in X department with 5/6/7 years experience earning more than peers (Tier 1)
years_exp_filter = [5, 6, 7]
top_exp_tier = df_merged_co[(df_merged_co['years_of_experience'].isin(years_exp_filter)) & (df_merged_co['Tier'] == 1)]
top_exp_tier_grouped = top_exp_tier.groupby(['company_hash', 'job_position']).apply(lambda x: x.sort_values(by='ctc',ascending=False).head(10)).reset_index(drop=True).sort_values('ctc', asce
print("Top 10 employees in each company & department with 5/6/7 years experience (Tier 1):")
print(top_exp_tier_grouped[['email_hash', 'company_hash', 'job_position', 'years_of_experience', 'ctc', 'Tier']])
```

```
Top 10 employees in each company & department with 5/6/7 years experience (Tier 1):
                                             email_hash         company_hash  \
6688  5b4bed51797140db4ed52018a979db1e34cee49e27b488...         wxowg ojontbo
4015  431c610cffb5f699476173431bb1f47a51bcc680407e44...          qgmtqn mgowy
5069  48b00207f75dd25ca9d518103e2ddc3c9a9706e51ae393...  uqvbvnx ntwyzgrgsxto
1458  71d7605911c92225343efc7e8aa1a81b60b5ed81796318...         fxuqg rxbxnta
1459  1b95e7ba0ee82100ca5a034239fa0203a1bec14280b82a...         fxuqg rxbxnta
1556  68aa38470922a03f6022280b2a13c6f5ab6a717f70c77a...                 gnytq
5520  8dfe6251bd4ec533f02ddceb98b3dcebb9550ccd4ef2e6...                vbvkgz
7924  59361208b0af18838c3240d4f7a02f6aad20ed93f9a73e...                   zvz
6771  431c610cffb5f699476173431bb1f47a51bcc680407e44...         xb  v onhatzn
3413  3c453dd102ae47a4ed1841be352213fad363d0944177e9...         otre tburgjta

           job_position  years_of_experience        ctc  Tier
6688            Missing                    7  255555555     1
4015            Missing                    5  200000000     1
5069            Missing                    5  200000000     1
1458   Frontend Engineer                   7  200000000     1
1459  FullStack Engineer                   7  200000000     1
1556            Missing                    5  200000000     1
5520              Other                    7  200000000     1
7924            Missing                    5  200000000     1
6771  FullStack Engineer                   5  200000000     1
3413    Backend Engineer                   5  200000000     1
```

Q) Top 10 companies (based on their CTC)

Ans)

```python
# Top 10 companies based on average CTC
top_companies = df_merged_co.sort_values('avg_ctc_co', ascending=False).head(10)
print("Top 10 companies based on average CTC:")
print(top_companies)
```

```
52823   57cd2c7b9703fae9ee2e543d48dc8445cff9a36b33349e...      2014
32673   c1988a101573e8c3ce2667d33427579285237b7fbfe77f...      2016
2960    f958792fd46b8a4453d0c2f95512bcc6aa7e0108bcf047...      2016
103063  ab8048ef33901acedee6673ad5168672f4d69507984a1e...      2021
106     996aef9bba62bd99d6cb8e8c112c0ec8096b203ae50b97...      2017
6794    2323f80afa6f3c809ac468997c1cf1ea8572d06bd8904e...      2017
691     dfdb45fb9631b9064a94be87a27a621068530ac1f3807c...      2017
34562   a7ff95d399e2822b866066d08467f99711e3894ba79b96...      2017

               ctc       job_position  ctc_updated_year  years_of_experience  \
72824   1000150000            Missing              2020                   10
3301     250000000            Missing              2020                    4
52823    200000000            Missing              2020                   11
32673    200000000              Other              2020                    9
2960     200000000            Missing              2020                    9
103063   200000000  Frontend Engineer              2020                    4
106      200000000   Support Engineer              2020                    8
6794     200000000              Other              2020                    8
691      200000000              Other              2020                    8
34562    200000000              Other              2020                    8

        company_hash_encoded  job_position_encoded  ctc_scaled  ...  \
72824                  30494                   422   84.552726  ...
3301                    1218                   422   20.990629  ...
52823                  16064                   422   16.754003  ...
32673                  31008                   455   16.754003  ...
2960                   33590                   422   16.754003  ...
103063                 18506                   285   16.754003  ...
106                    17542                   859   16.754003  ...
6794                   29925                   455   16.754003  ...
691                    16660                   455   16.754003  ...
34562                  14619                   455   16.754003  ...

               max_y         min_y  count_y  Class    avg_ctc_co  \
72824   1.000150e+09  1.000150e+09      1.0      2  1.000150e+09
3301    2.500000e+08  2.500000e+08      1.0      2  2.500000e+08
52823   2.000000e+08  2.000000e+08      1.0      2  2.000000e+08
32673   2.000000e+08  2.000000e+08      1.0      2  2.000000e+08
2960    2.000000e+08  2.000000e+08      1.0      2  2.000000e+08
103063  2.000000e+08  2.000000e+08      1.0      2  2.000000e+08
106     2.000000e+08  2.000000e+08      1.0      2  2.000000e+08
6794    2.000000e+08  2.000000e+08      1.0      2  2.000000e+08
691     2.000000e+08  2.000000e+08      1.0      2  2.000000e+08
34562   2.000000e+08  2.000000e+08      1.0      2  2.000000e+08

        median_ctc_co           max           min  count  Tier
72824    1.000150e+09  1.000150e+09  1.000150e+09    1.0     2
3301     2.500000e+08  2.500000e+08  2.500000e+08    1.0     2
52823    2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
32673    2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
2960     2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
103063   2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
106      2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
6794     2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
691      2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
34562    2.000000e+08  2.000000e+08  2.000000e+08    1.0     2
```

Q) Top 2 positions in every company (based on their CTC)
Ans)

```
# Top 2 positions in every company based on average CTC
pos_ctc = df.groupby(['company_hash', 'job_position'])['ctc'].mean().reset_index()
top2_positions = pos_ctc.groupby('company_hash').apply(lambda x: x.sort_values(by='ctc',ascending=False).head(2)).reset_index(drop=True)
print("Top 2 positions in every company based on average CTC:")
print(top2_positions)
```

```
Top 2 positions in every company based on average CTC:
                       company_hash        job_position        ctc
0                                 0             Missing   100000.0
1                                 0               Other   100000.0
2                              0000               Other  1150000.0
3                       01 ojztqsj   Frontend Engineer   830000.0
4                       01 ojztqsj    Android Engineer   270000.0
...                             ...                 ...        ...
50254                            zz             Missing   500000.0
50255  zzb ztdnstz vacxogqj ucn rna             Missing  3000000.0
50256  zzb ztdnstz vacxogqj ucn rna  FullStack Engineer   600000.0
50257                        zzgato             Missing  1800000.0
50258                        zzzbzb               Other   720000.0
```

## **Unsupervised Learning – Clustering**

Q) Checking clustering tendency
Ans)

```
[47] #  Data preprocessing for Unsupervised Clustering

     # Select features for clustering
     features = ['company_hash', 'job_position', 'years_of_experience', 'ctc']

     # Encoding categorical variables
     le_company = LabelEncoder()
     le_job = LabelEncoder()

     df['company_enc'] = le_company.fit_transform(df['company_hash'])
     df['job_enc'] = le_job.fit_transform(df['job_position'])

     X = df[['company_enc', 'job_enc', 'years_of_experience', 'ctc']]
     X.head(10)
```

| | company_enc | job_enc | years_of_experience | ctc |
|---|---|---|---|---|
| 0 | 969 | 455 | 9 | 1100000 |
| 1 | 19730 | 289 | 7 | 449999 |
| 2 | 15512 | 138 | 10 | 2000000 |
| 3 | 12108 | 138 | 8 | 700000 |
| 4 | 20226 | 289 | 8 | 1400000 |
| 5 | 35570 | 289 | 7 | 700000 |
| 6 | 10162 | 289 | 7 | 1500000 |
| 7 | 29158 | 138 | 6 | 400000 |
| 8 | 25405 | 422 | 5 | 450000 |
| 9 | 33129 | 422 | 6 | 360000 |

```
[48] # Standardize features
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
```

```
# Check clustering tendency (Hopkins statistic)


from sklearn.neighbors import NearestNeighbors
import random

def hopkins(X):
    d = X.shape[1]
    n = len(X)  # rows
    m = int(0.1 * n)  # sample size 10%
    nbrs = NearestNeighbors(n_neighbors=1).fit(X)
    rand_X = np.random.uniform(np.min(X, axis=0), np.max(X, axis=0), (m, d))
    ujd = []
    wjd = []
    for j in range(m):
        u_dist, _ = nbrs.kneighbors([rand_X[j]], 2, return_distance=True)
        ujd.append(u_dist[0][1])
        w_dist, _ = nbrs.kneighbors([X[random.randint(0, n-1)]], 2, return_distance=True)
        wjd.append(w_dist[0][1])
    H = sum(ujd) / (sum(ujd) + sum(wjd))
    return H

hopkins_stat = hopkins(X_scaled)
print(f"Hopkins statistic: {hopkins_stat:.3f}")
# If close to 1, data is clusterable
```

```
Hopkins statistic: 0.999
```

Insights : Here we are checking the clustering tendency by using the method of Hopkins statistics. We encode the categorical columns and then use the standardized version for hopkisn statistics. Here the value is shown as 0.999. Any value for Hopkins statistics more than 0.5 implies that the dataset is clusterable.

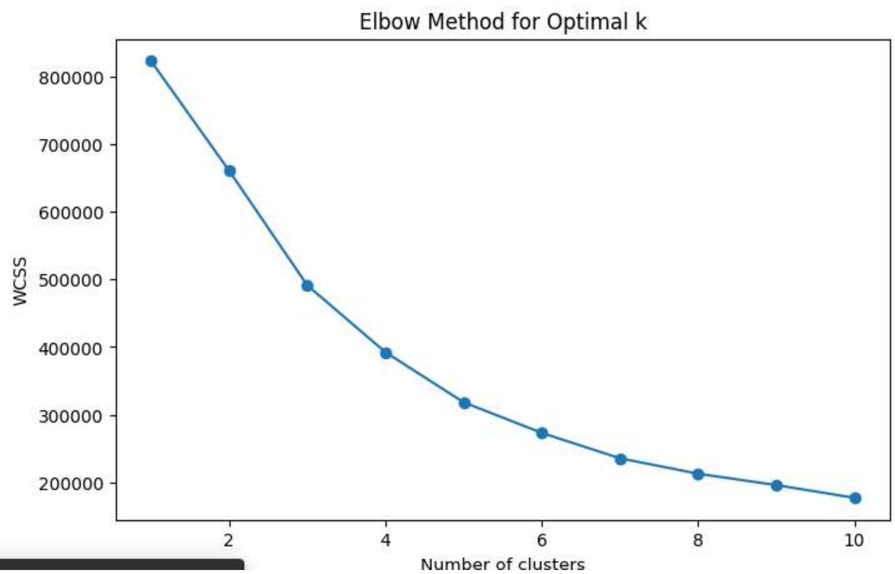
Q) Elbow method

Ans)

Insights : - Elbow plot suggests optimal clusters around k=4 for KMeans.

As the number of clusters increases from 1 to around 3, the WCSS decreases sharply. This indicates that adding more clusters significantly reduces the variance within each cluster. Beyond k = 4, the decrease in WCSS becomes less pronounced, indicating that adding more clusters is not substantially improving the homogeneity of the clusters. Choosing a k beyond this point might lead to overfitting, where you are creating clusters that are too specific to the data and may not generalize well to unseen data.

```
[50] # Elbow method to find optimal k for KMeans
     wcss = []
     for i in range(1, 11):
         kmeans = KMeans(n_clusters=i, random_state=42)
         kmeans.fit(X_scaled)
         wcss.append(kmeans.inertia_)

     plt.figure(figsize=(8,5))
     plt.plot(range(1, 11), wcss, marker='o')
     plt.title('Elbow Method for Optimal k')
     plt.xlabel('Number of clusters')
     plt.ylabel('WCSS')
     plt.show()
```




Q) K-means clustering
Ans)

```
[52] # K-means clustering with chosen k (say k=4 based on elbow)
     k = 4
     kmeans = KMeans(n_clusters=k, random_state=42)
     df['KMeans_Cluster'] = kmeans.fit_predict(X_scaled)

     # Cluster profile summary
     print(df.groupby('KMeans_Cluster')[['ctc', 'years_of_experience']].mean())
```

```
                         ctc  years_of_experience
KMeans_Cluster
0               1.277489e+06             8.574136
1               2.228639e+06            17.439703
2               1.351530e+08             9.548653
3               1.235777e+06             8.511711
```

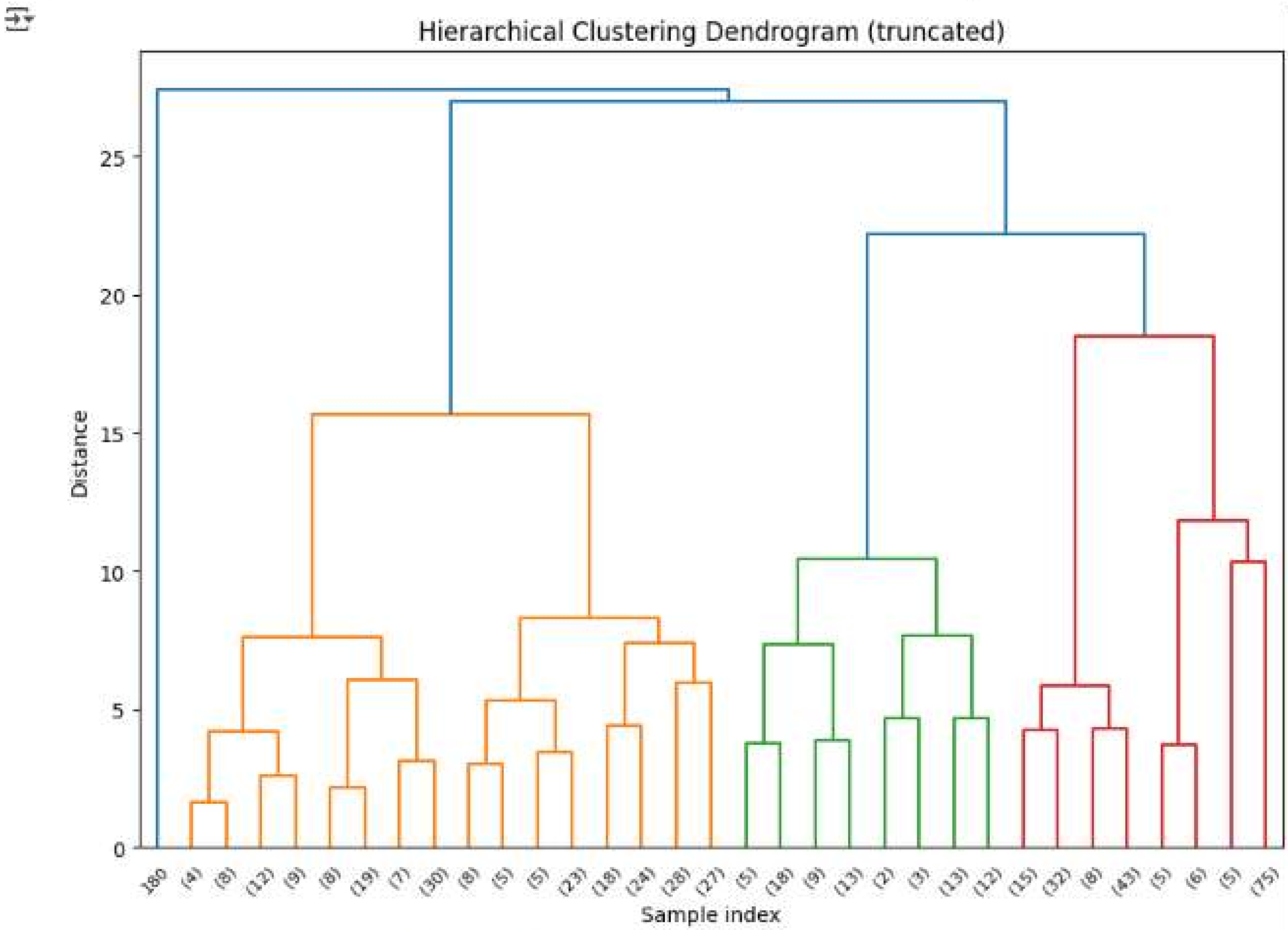
Insights: - KMeans clustering groups learners with similar company, job, experience, and CTC. - The K-Means algorithm has partitioned the learners into four distinct groups based on the features used for clustering. These clusters exhibit different average characteristics in terms of CTC and years of experience:

- **Cluster 1:** Represents high-CTC, high-experience individuals (Senior/Experienced).
- **Cluster 2:** Represents extremely high-CTC individuals with moderate experience (Potential Outliers/Highly Specialized).
- **Cluster 0 and Cluster 3:** Represent groups with moderate CTC and moderate experience (Mid-level). These two clusters might be separated based on other features used in clustering (e.g., company, job role)

Q) Hierarchical clustering

Ans)

```python
# Hierarchical clustering (on a sample if large)
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

sample_df = df.sample(n=500, random_state=42)
sample_X = sample_df[['company_enc', 'job_enc', 'years_of_experience', 'ctc']]
sample_X_scaled = scaler.fit_transform(sample_X)

linked = linkage(sample_X_scaled, method='ward')

plt.figure(figsize=(10, 7))
dendrogram(linked, truncate_mode='level', p=5)
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.show()

# Assign cluster labels from hierarchical clustering (e.g., 4 clusters)
sample_df['Hier_Cluster'] = fcluster(linked, 4, criterion='maxclust')

print(sample_df.groupby('Hier_Cluster')[['ctc', 'years_of_experience']].mean())
```




| Hier_Cluster | ctc | years_of_experience |
|---|---|---|
| 1 | 1.128447e+06 | 9.004255 |
| 2 | 2.087867e+06 | 17.746667 |
| 3 | 1.390753e+06 | 8.455026 |
| 4 | 6.060000e+07 | 10.000000 |

Insights : The dendrogram illustrates the hierarchical relationships between data points (learners). Each leaf at the bottom represents an individual data point. As you move up the dendrogram, branches merge at different distances, forming clusters.
The vertical axis represents the distance (or dissimilarity) at which clusters are merged. Longer vertical lines indicate that clusters merged at a greater distance, implying they were less similar before merging
Visually, we can infer four main branches that are relatively distinct before merging at higher distances. The leftmost large branch (blue at the top, merging orange) represents a significant cluster. The next major branch (orange) represents another cluster. The middle branch (green) forms a third cluster. The rightmost major branch (red) forms a fourth cluster.

Learners in cluster 1 have a relatively moderate average CTC and moderate years of experience. Learners in cluster 2 have a higher average CTC and significantly higher average years of experience compared to the other clusters, suggesting a more senior group.  Learners in cluster 3 have an extremely high average CTC compared to all other clusters, while their average years of experience is moderate. This likely identifies a group of very high-earning individuals who may or may not be the most experienced. Learners in cluster 4 also have a very high average CTC (though lower than Cluster 3) and a moderate level of experience. This represents another group of high-earning individuals

**Comparison with K-Means Results:**

Hierarchical clustering builds a hierarchy, and the final clusters depend on the chosen cut-off, while K-Means directly partitions the data into a pre-defined number of clusters. The profiles of the clusters obtained from Hierarchical Clustering show some similarities to the K-Means results:

- **Cluster 2 (Hierarchical) and Cluster 1 (K-Means):** Both identify a high-CTC, high-experience group.
- **Cluster 3 and 4 (Hierarchical) and Cluster 2 (K-Means):** Both identify groups with very high CTC and moderate experience, although Hierarchical Clustering seems to have separated them into two distinct high-earning groups.
- **Cluster 1 (Hierarchical) and Cluster 0/3 (K-Means):** Both identify groups with moderate CTC and moderate experience.

**Recommendations :**

*Targeted Content and Curriculum Development:*

- Cluster Profiling - Understand the distinct profiles of learners within each cluster based on their job positions, companies, and compensation levels. Tailor course content and marketing materials to resonate with the specific needs and aspirations of each group.
- Specialized Tracks - If clusters represent distinct career paths (e.g., high-paying senior roles vs. entry-level positions in specific domains), consider developing specialized learning tracks or modules that cater to the upskilling requirements of these segments.

*Personalized Career Guidance*

- Benchmarking - Allow learners to benchmark their profiles (job position, company, experience, CTC) against the characteristics of different clusters. This can provide insights into potential career trajectories and salary expectations.
- Mentorship Pairing - Consider pairing learners within similar clusters or connecting those in aspirational clusters with mentors who have successfully transitioned into those groups.

*Company and Job Role Insights for Scaler:*

- High-Growth Areas - Identify clusters with high average CTC and strong representation of specific job roles or companies. This can highlight areas where Scaler's training is particularly effective or where there is high market demand.
- Underperforming Segments - Analyze clusters with lower average CTC or less successful career transitions. Investigate potential reasons and consider curriculum adjustments or additional support to improve outcomes for these learners.
- Partnerships- Focus on building relationships with companies that are prominent in the high-performing clusters to potentially create hiring pipelines for Scaler graduates.

*Marketing and Outreach Strategies*

- Segmented Marketing - Develop targeted marketing campaigns that highlight the success stories and career outcomes of learners within specific clusters. Use language and channels that are most relevant to each segment.
- Employer Branding - Showcase the types of companies that hire Scaler graduates from different clusters to attract prospective learners interested in those organizations.

*Course Feature Enhancement*

- Skill Gap Analysis - Analyze the skills and backgrounds of learners in different clusters to identify potential skill gaps that Scaler's curriculum could address more effectively.
- Alumni Network - Leverage the cluster information to build a more targeted and effective alumni network, connecting learners with shared experiences and career paths.