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. 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.
You are 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. You are 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.

Key Steps Performed:

1. Data Cleaning & Preprocessing

- Null/missing value handling
- Encoding categorical variables (Company & Job Position)
- Outlier detection using boxplots
- Data normalization

2. EDA (Exploratory Data Analysis)

- Visualized salary vs experience, CTC distributions
- Tier and class group-wise analysis
- Job position frequency and trends

3. Manual Clustering Flags

- Defined **Class** based on Company + Job Position
- Defined **Tier** categorization logic for companies

4. Clustering

- Performed clustering using **KMeans**
- Evaluated optimal clusters using the **Elbow method**

5. Dimensionality Reduction & Visualization

- Visualized clusters using **PCA** and **UMAP**
- Evaluated cluster separability and structure

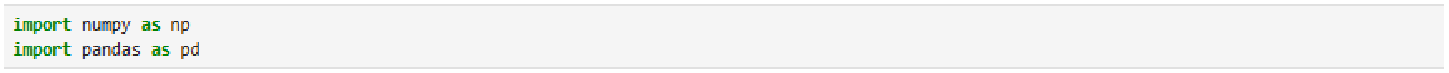
End Goals:

- Group employees into logical salary segments
- Provide insights into fair pay based on experience and designation
- Help businesses benchmark roles and salaries
- Identify and explore outliers in salary data

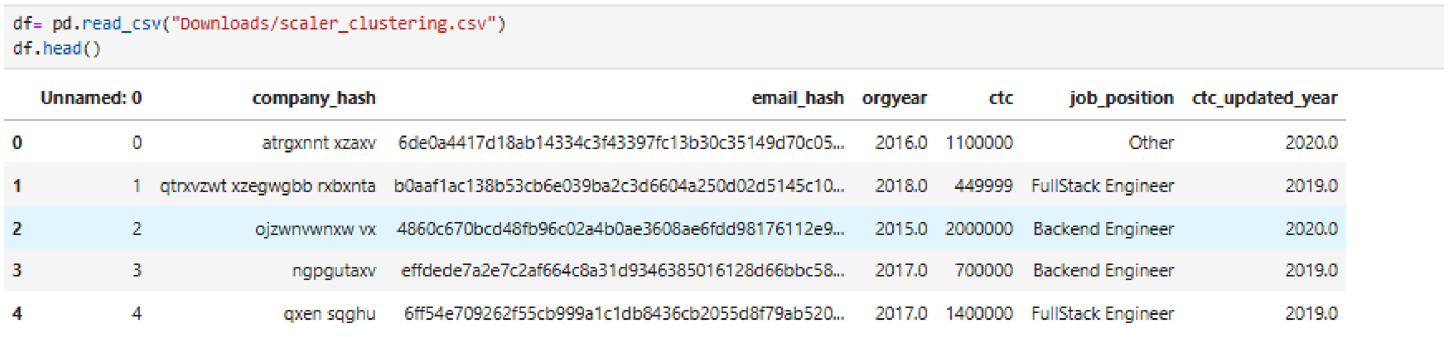
For a more detailed and comprehensive analysis, please refer to the Jupyter Notebook link provided at the end of this report. Certain exploratory steps and analytical approaches that were not included in this summary have been thoroughly executed and documented in the notebook.

1. Importing and Cleaning the Dataset

1.1 Data Importing



```
import numpy as np
import pandas as pd
```

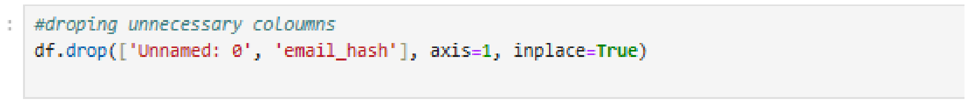


```
df= pd.read_csv("Downloads/scaler_clustering.csv")
df.head()
```

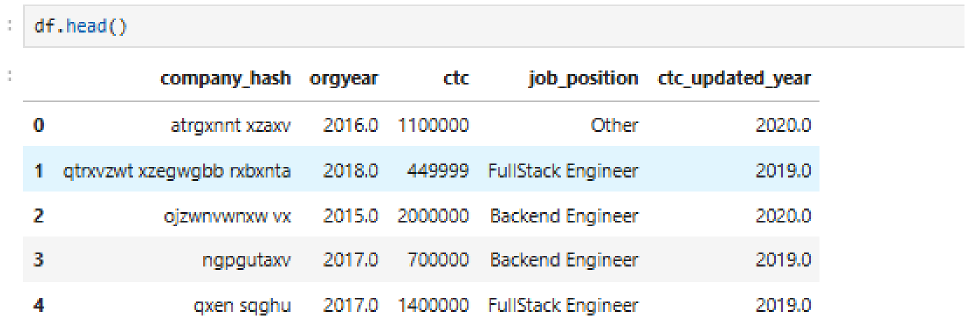
	Unnamed: 0	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
0	0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000	Other	2020.0
1	1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999	FullStack Engineer	2019.0
2	2	ojzwnvwnxw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

1.2 Dropping unnecessary coloumn

Code & Output:



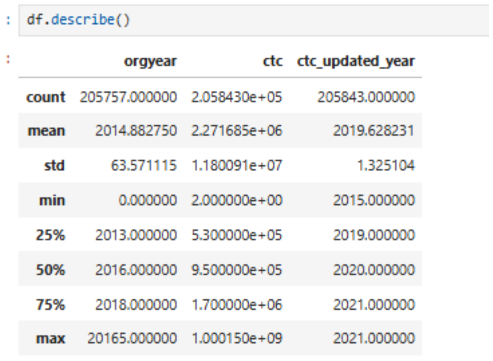
```
#droping unnecessary coloumns
df.drop(['Unnamed: 0', 'email_hash'], axis=1, inplace=True)
```



```
df.head()
```

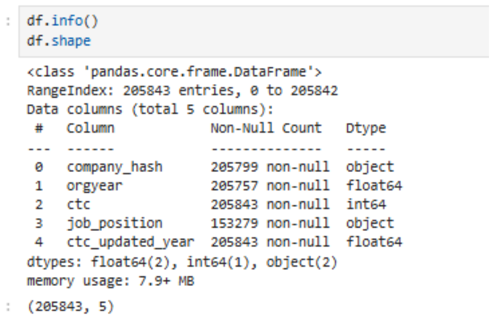
	company_hash	orgyear	ctc	job_position	ctc_updated_year
0	atrgxnnt xzaxv	2016.0	1100000	Other	2020.0
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999	FullStack Engineer	2019.0
2	ojzwnvwnxw vx	2015.0	2000000	Backend Engineer	2020.0
3	ngpgutaxv	2017.0	700000	Backend Engineer	2019.0
4	qxen sqghu	2017.0	1400000	FullStack Engineer	2019.0

1.3 Dataset Structure and Summary Statistics



```
df.describe()
```

	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



```
df.info()
df.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 5 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   company_hash      205799 non-null  object
 1   orgyear           205757 non-null  float64
 2   ctc               205843 non-null  int64
 3   job_position      153279 non-null  object
 4   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(1), object(2)
memory usage: 7.9+ MB
```

```
(205843, 5)
```

Insights:

The **orgyear** column shows some data quality issues. The minimum value is 0, which is clearly invalid, and the maximum value is 20165, which is not a valid year and indicates a data entry error. However, the median value is 2016, which seems reasonable and aligns with expectations.

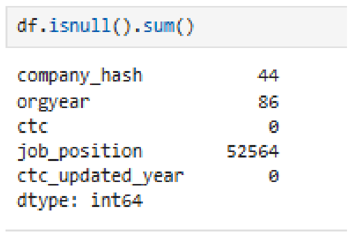
The **ctc** (Cost to Company) column displays a very high standard deviation of around 11.8 million, despite a mean value of approximately 2.27 million. This suggests a wide spread in salary values and the presence of significant outliers. In fact, the maximum CTC recorded is 1 billion (1e9), which is an extreme value and likely not realistic, pointing to either outliers or erroneous entries in the dataset.

On the other hand, the **ctc_updated_year** appears to be clean and consistent. The values range from 2015 to 2021, and the median value is 2020. The distribution here is tight and logical, indicating that this column is reliable and doesn't require much preprocessing.

Overall, before proceeding with modeling or further analysis, it's important to clean and validate the orgyear and ctc columns to handle outliers and incorrect values.

2. KNN Imputation and Null values detection

2.1 Checking for null values.



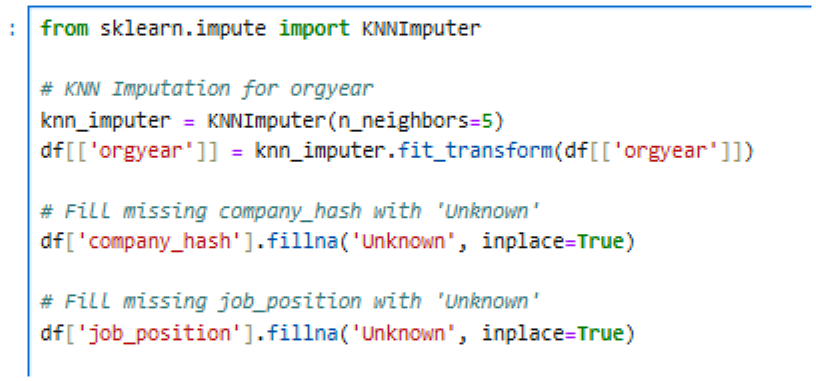
```
df.isnull().sum()
```

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

Insights:

The dataset has some missing values. company_hash and orgyear have 44 and 86 nulls respectively, which are minor and manageable. However, job_position has over **52,000 missing entries**, which could significantly affect role-based insights and may need filtering or imputation. Notably, ctc and ctc_updated_year have no missing values, allowing reliable compensation analysis.

2.2 Handling Missing vlaues



```
: from sklearn.impute import KNNImputer

  # KNN Imputation for orgyear
  knn_imputer = KNNImputer(n_neighbors=5)
  df[['orgyear']] = knn_imputer.fit_transform(df[['orgyear']])

  # Fill missing company_hash with 'Unknown'
  df['company_hash'].fillna('Unknown', inplace=True)

  # Fill missing job_position with 'Unknown'
  df['job_position'].fillna('Unknown', inplace=True)
```

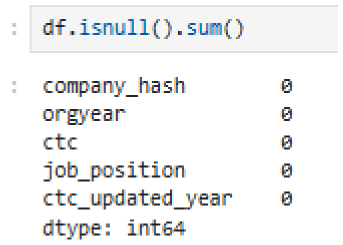
Insights:

orgyear (Numerical):
Missing values were imputed using **KNN Imputer** with 5 nearest neighbors. This technique fills in the missing values based on the similarity of other data points, ensuring better contextual accuracy than simple mean/median imputation.

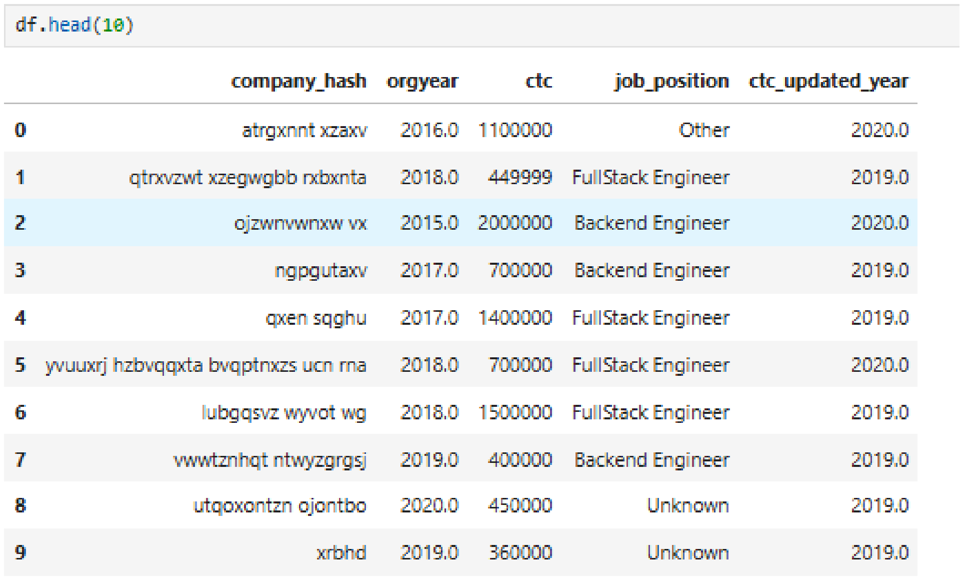
company_hash (Categorical):
Missing values were filled with the placeholder **'Unknown'**, allowing retention of these records without discarding them, while also flagging them clearly for potential downstream analysis.

job_position (Categorical):
Similarly, missing values were replaced with **'Unknown'**, ensuring that records with unspecified job roles are preserved while being distinctly labeled for further filtering or modeling.

After handling missing values



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

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



```
df.head(10)
```

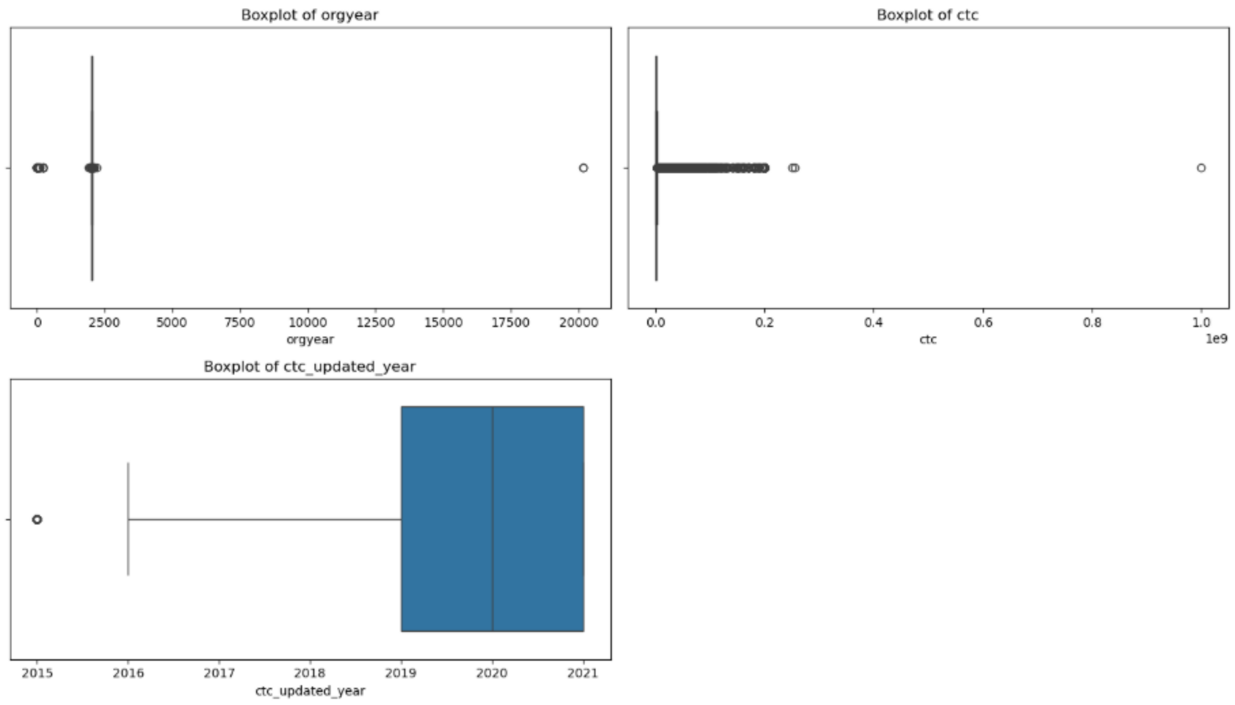
	company_hash	orgyear	ctc	job_position	ctc_updated_year
0	atrgxnnt xzaxv	2016.0	1100000	Other	2020.0
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999	FullStack Engineer	2019.0
2	ojzwnvwnxw vx	2015.0	2000000	Backend Engineer	2020.0
3	ngpgutaxv	2017.0	700000	Backend Engineer	2019.0
4	qxen sqghu	2017.0	1400000	FullStack Engineer	2019.0
5	yvuuxrj hzbvqqxta bvqptnxzs ucn rna	2018.0	700000	FullStack Engineer	2020.0
6	lubgqsvz wyvot wg	2018.0	1500000	FullStack Engineer	2019.0
7	vwwtznhqt ntwyzgrgsj	2019.0	400000	Backend Engineer	2019.0
8	utqoxontzn ojontbo	2020.0	450000	Unknown	2019.0
9	xrbhd	2019.0	360000	Unknown	2019.0

3. Outlier treatment

3.1 Boxplot for outlier detection

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set plot size and style
plt.figure(figsize=(14, 8))
numerical_cols = ['orgyear', 'ctc', 'ctc_updated_year']

for i, col in enumerate(numerical_cols):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(data=df, x=col)
    plt.title(f'Boxplot of {col}')

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

Output:



Exploring more on outliers

```
]: df.orgyear.value_counts().sort_values(ascending=True)
```

```
]: orgyear
   200.0        1
   83.0         1
   209.0        1
   2204.0       1
   1977.0       1
              ...
   2015.0    20610
   2016.0    23043
   2017.0    23239
   2019.0    23427
   2018.0    25256
   Name: count, Length: 78, dtype: int64
```

Insights:

Upon reviewing the orgyear column after imputation, several **outlier values** were identified such as 200, 83, 209, 2204, and 1977, which are clearly invalid employment start years. These likely stem from data entry or imputation errors. The majority of entries fall between **2015 and 2019**, which aligns with realistic recent employment trends. To handle these outliers, we applied the **IQR (Interquartile Range) method** to detect and remove or correct the unrealistic values, ensuring data quality and consistency for downstream analysis.

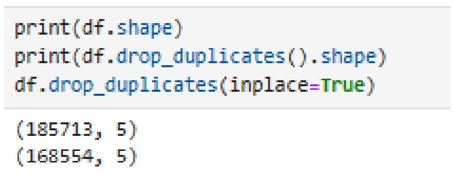
```
#removing outliers from orgyear using IQR

q1=df.orgyear.quantile(0.25)
q3=df.orgyear.quantile(0.75)
iqr=q3-q1

df=df.loc[(df.orgyear>=q1-1.5*iqr) & (df.orgyear<=q3+1.5*iqr)]


#removing outliers from ctc using IQR

q1=df.ctc.quantile(0.25)
q3=df.ctc.quantile(0.75)
iqr=q3-q1

df=df.loc[(df.ctc>=q1-1.5*iqr) & (df.ctc<=q3+1.5*iqr)]
```

After IQR

```
: df.orgyear.value_counts().sort_index(ascending=True)
```

```
: orgyear
2006.00000     1609
2007.00000     1826
2008.00000     2287
2009.00000     3227
2010.00000     5047
2011.00000     7125
2012.00000     9551
2013.00000    11357
2014.00000    15616
2014.88275       78
2015.00000    19429
2016.00000    21859
2017.00000    22251
2018.00000    24355
2019.00000    22722
2020.00000    12901
2021.00000     3412
2022.00000      810
2023.00000      207
2024.00000       32
2025.00000       12
Name: count, dtype: int64
```

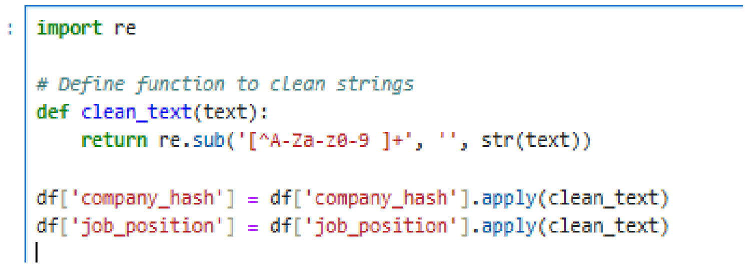
3.2 Dropping Duplicate



```
print(df.shape)
print(df.drop_duplicates().shape)
df.drop_duplicates(inplace=True)
```

```
(185713, 5)
(168554, 5)
```

3.3 Text Cleaning Using Regular Expressions (Regex)



```
import re

# Define function to clean strings
def clean_text(text):
    return re.sub('[^A-Za-z0-9 ]+', '', str(text))

df['company_hash'] = df['company_hash'].apply(clean_text)
df['job_position'] = df['job_position'].apply(clean_text)
```

Insights:

To ensure consistent and clean text data, especially in categorical columns like company_hash and job_position, a **text cleaning function** was applied using Regular Expressions (re module).
This function:
re.sub('[^A-Za-z0-9]+', '', str(text))
removes all special characters, allowing only letters (A–Z, a–z), numbers (0–9), and spaces.

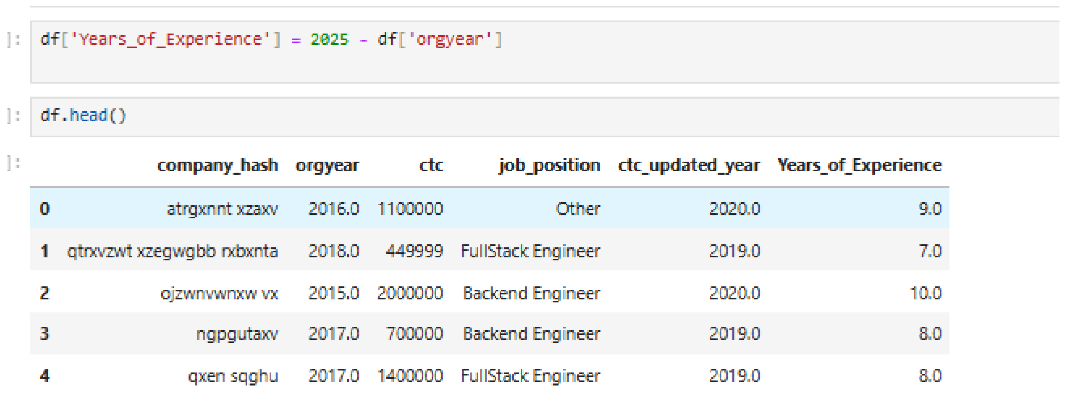
By applying this to both company_hash and job_position, we:

- Eliminate unwanted symbols or noise that could interfere with grouping, encoding, or clustering.
- Ensure better results in analysis and modeling by having standardized string values.

This step is especially important before performing **label encoding, groupby operations, or clustering**, where inconsistent strings could lead to incorrect grouping or duplicate categories.

4. Manual Clustering: Creating the Designation Flag And EDA

4.1 Creating new feature Experience



```
]: df['Years_of_Experience'] = 2025 - df['orgyear']
```

```
]: df.head()
```

]:

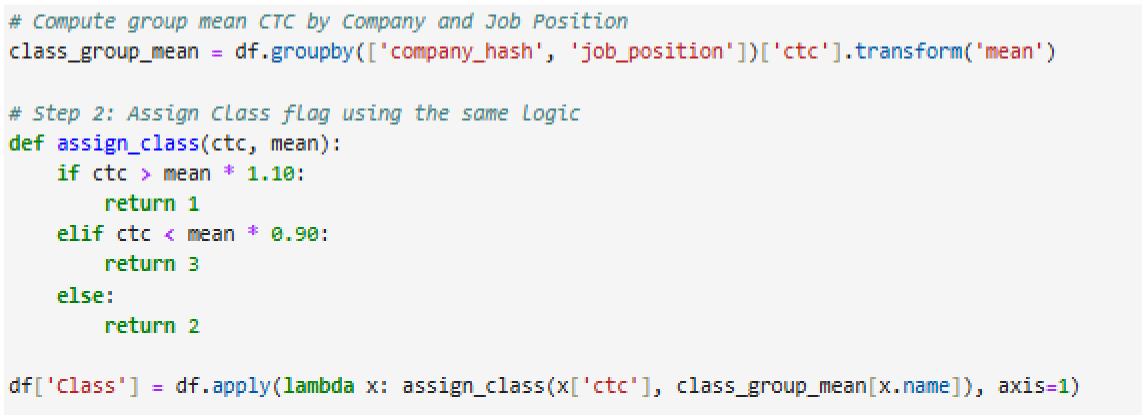
	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience
0	atrgxnnt xzaxv	2016.0	1100000	Other	2020.0	9.0
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999	FullStack Engineer	2019.0	7.0
2	ojzwnvwnxw vx	2015.0	2000000	Backend Engineer	2020.0	10.0
3	ngpgutaxv	2017.0	700000	Backend Engineer	2019.0	8.0
4	qxen sqghu	2017.0	1400000	FullStack Engineer	2019.0	8.0

4.2 Manual Clustering

```
# Remove invalid experience (if any)
df = df[df['Years_of_Experience'] >= 0]

# Step 2: Calculate group mean CTC
group_mean = df.groupby(['company_hash', 'job_position', 'Years_of_Experience'])['ctc'].transform('mean')

# Step 3: Define Designation flag
def assign_designation(ctc, mean):
    if ctc > mean * 1.10:
        return 1
    elif ctc < mean * 0.90:
        return 3
    else:
        return 2

df['Designation'] = df.apply(lambda x: assign_designation(x['ctc'], group_mean[x.name]), axis=1)
```

Insights:
This code performs **manual clustering by assigning a "Designation" flag** to each learner based on their salary (CTC) in comparison to peers within the same company, job position, and years of experience. It first removes any invalid entries with negative experience. Then, it calculates the average CTC for each peer group and compares individual salaries against this benchmark. Learners earning significantly more than the average (10% above) are flagged as 1 (high earners), those earning significantly less (10% below) are flagged as 3 (low earners), and those within a 10% range are marked as 2 (average earners). This clustering helps identify salary outliers and evaluate compensation equity within specific roles and companies.

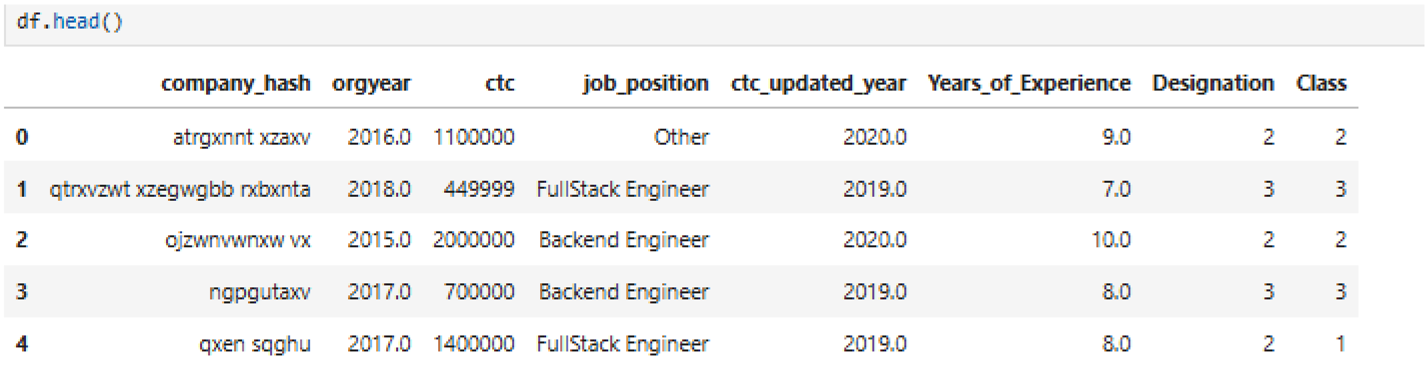
4.3 Manual clustering to assign a "Class" flag



```python
# Compute group mean CTC by Company and Job Position
class_group_mean = df.groupby(['company_hash', 'job_position'])['ctc'].transform('mean')

# Step 2: Assign Class flag using the same logic
def assign_class(ctc, mean):
    if ctc > mean * 1.10:
        return 1
    elif ctc < mean * 0.90:
        return 3
    else:
        return 2

df['Class'] = df.apply(lambda x: assign_class(x['ctc'], class_group_mean[x.name]), axis=1)
```

Insights:

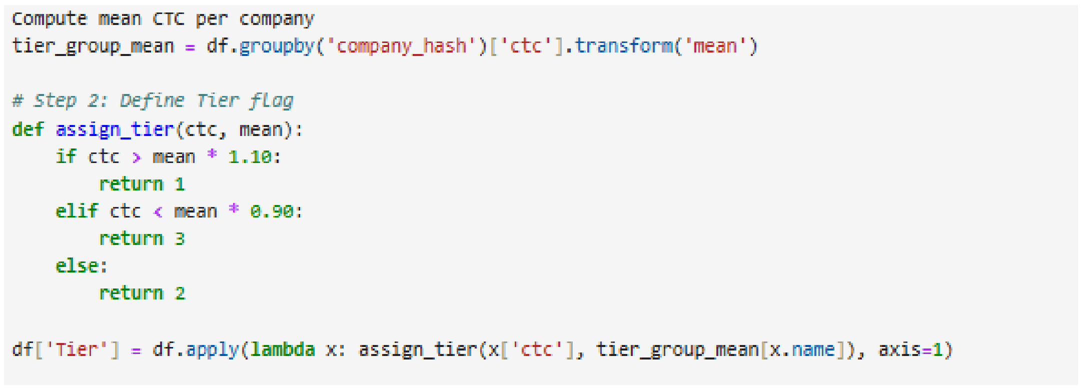
This code segment assigns a **"Class" flag** to each learner by comparing their CTC against the average CTC of individuals with the **same job position in the same company**, regardless of experience. First, it calculates the mean salary for each unique combination of company_hash and job_position. Then, using defined thresholds, it categorizes learners into three groups: **Class 1** for those earning significantly above average (10% higher), **Class 3** for those earning significantly below average (10% lower), and **Class 2** for those falling within the average range. This classification helps identify top, average, and low earners within specific roles across companies.



```python
df.head()
```

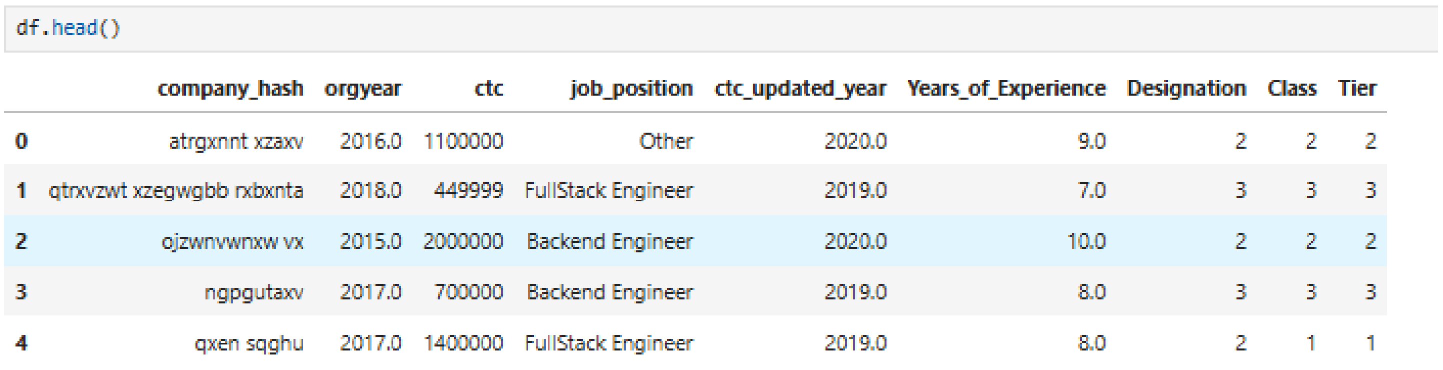
	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	Designation	Class
0	atrgxnnt xzaxv	2016.0	1100000	Other	2020.0	9.0	2	2
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999	FullStack Engineer	2019.0	7.0	3	3
2	ojzwnvwnxw vx	2015.0	2000000	Backend Engineer	2020.0	10.0	2	2
3	ngpgutaxv	2017.0	700000	Backend Engineer	2019.0	8.0	3	3
4	qxen sqghu	2017.0	1400000	FullStack Engineer	2019.0	8.0	2	1

4.4 Manual clustering to assign a "tire" flag



```python
Compute mean CTC per company
tier_group_mean = df.groupby('company_hash')['ctc'].transform('mean')

# Step 2: Define Tier flag
def assign_tier(ctc, mean):
    if ctc > mean * 1.10:
        return 1
    elif ctc < mean * 0.90:
        return 3
    else:
        return 2

df['Tier'] = df.apply(lambda x: assign_tier(x['ctc'], tier_group_mean[x.name]), axis=1)
```

Insights:
This code assigns a tier to each employee based on how their CTC compares to the average CTC within their company. It first calculates the mean CTC for each company and then categorizes employees into three tiers: Tier 1 for those earning more than 10% above the company average, Tier 3 for those earning more than 10% below, and Tier 2 for those within ±10% of the average. This helps identify employees who are overpaid, underpaid, or paid fairly relative to their peers within the same company.

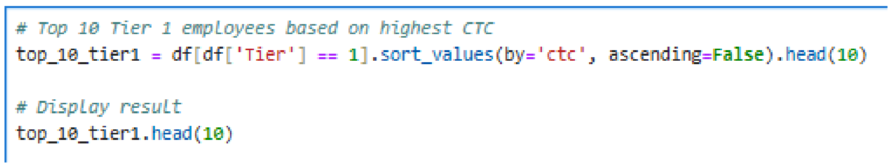


```
df.head()
```

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	Designation	Class	Tier
0	atrgxnnt xzaxv	2016.0	1100000	Other	2020.0	9.0	2	2	2
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999	FullStack Engineer	2019.0	7.0	3	3	3
2	ojzwnvwnxw vx	2015.0	2000000	Backend Engineer	2020.0	10.0	2	2	2
3	ngpgutaxv	2017.0	700000	Backend Engineer	2019.0	8.0	3	3	3
4	qxen sqghu	2017.0	1400000	FullStack Engineer	2019.0	8.0	2	1	1

4.5 Let's now perform an analysis on the Top 10 Tier 1 Employees:

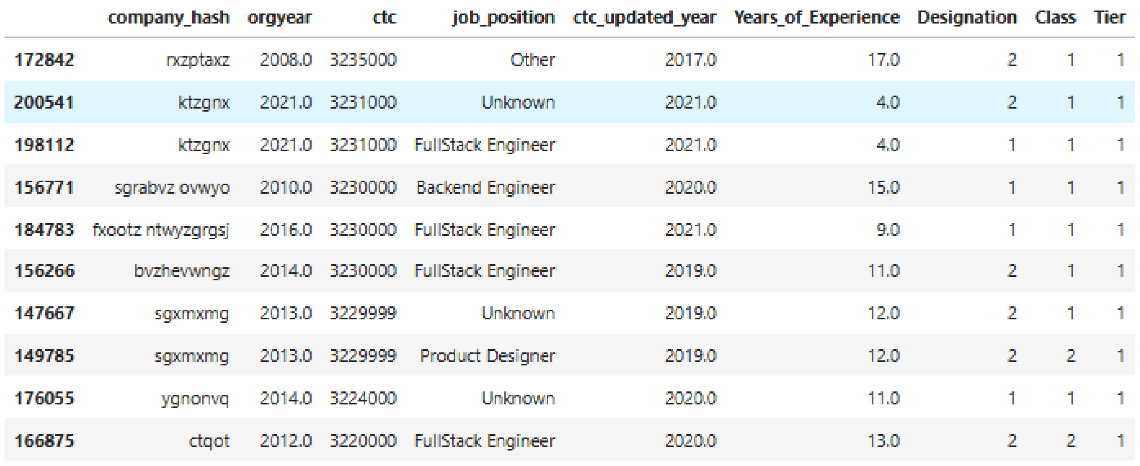
To begin understanding the upper end of the salary distribution, we first filtered the dataset to include only **Tier 1 employees**, as they represent the highest-performing or most reputed segment. We then sorted this subset by **CTC in descending order** to identify those earning the most.



```
# Top 10 Tier 1 employees based on highest CTC
top_10_tier1 = df[df['Tier'] == 1].sort_values(by='ctc', ascending=False).head(10)

# Display result
top_10_tier1.head(10)
```

From this sorted list, we extracted the **top 10 earners** using .head(10), and stored the results in top_10_tier1. This gives us a focused view of the most lucrative positions and companies among Tier 1 professionals, which can help uncover interesting patterns in compensation, job roles, experience levels, and employer trends.

Output:



	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	Designation	Class	Tier
172842	rxzptaxz	2008.0	3235000	Other	2017.0	17.0	2	1	1
200541	ktzgnx	2021.0	3231000	Unknown	2021.0	4.0	2	1	1
198112	ktzgnx	2021.0	3231000	FullStack Engineer	2021.0	4.0	1	1	1
156771	sgrabvz ovwyo	2010.0	3230000	Backend Engineer	2020.0	15.0	1	1	1
184783	fxootz ntwyzgrgsj	2016.0	3230000	FullStack Engineer	2021.0	9.0	1	1	1
156266	bvzhevwngz	2014.0	3230000	FullStack Engineer	2019.0	11.0	2	1	1
147667	sgxmxmg	2013.0	3229999	Unknown	2019.0	12.0	2	1	1
149785	sgxmxmg	2013.0	3229999	Product Designer	2019.0	12.0	2	2	1
176055	ygnonvq	2014.0	3224000	Unknown	2020.0	11.0	1	1	1
166875	ctqot	2012.0	3220000	FullStack Engineer	2020.0	13.0	2	2	1

Insights:

The top Tier 1 earners have **CTCs exceeding ₹3.2 million (32 Lakhs)**, which is significantly high.

Despite having varying levels of experience (from 4 to 17 years), some individuals like those with just 4 years of experience are earning at par with highly experienced professionals. This may point to **rapid career growth or premium skills**.

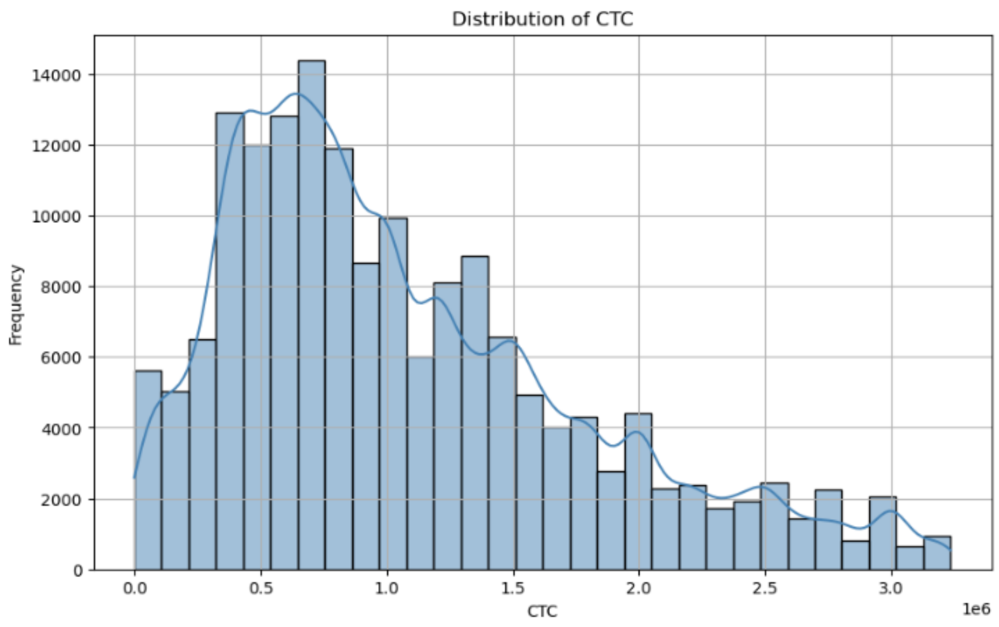
Some job positions are listed as "Unknown" or "Other", yet they still command top-tier salaries — indicating that **role names may be inconsistently filled** or **some niche roles are highly paid despite being unnamed**.

A few companies (e.g., ktzgnx, sgxmxmg) appear more than once, suggesting they **consistently pay higher CTCs**, making them top employers for lucrative jobs.

4.6 Histogram with a KDE (Kernel Density Estimation) plot to visualize the distribution of the ctc (Cost to Company) variable.

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.histplot(df['ctc'], kde=True, bins=30, color='steelblue')
plt.title('Distribution of CTC')
plt.xlabel('CTC')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

Output:



Insights:

The majority of individuals have lower to mid-range CTCs. This is evident from the high frequency at the left side of the histogram.

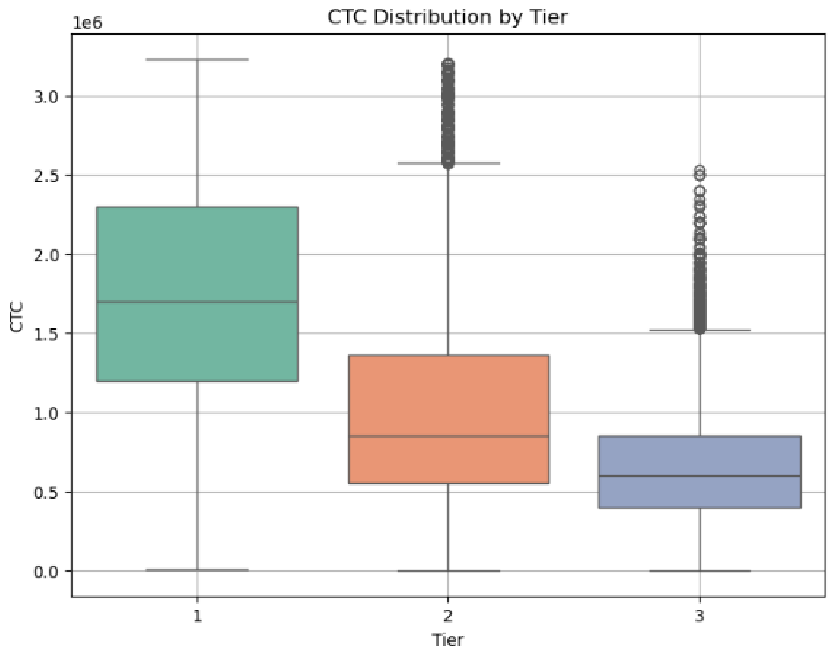
There are specific common CTC brackets or "salary bands." The distinct peaks suggest popular or typical compensation levels, likely corresponding to different experience levels, job roles, or industry standards.

A smaller number of individuals earn very high CTCs. The long tail extending to the right indicates that as CTC increases, the number of people earning that amount significantly decreases.

The distribution is not uniform; it's heavily concentrated at lower values and gradually thins out at higher values. This is a typical pattern for income or salary distributions, reflecting a hierarchical structure in compensation.

4.7 CTC Distribution by Tier

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Tier', y='ctc', data=df, palette='Set2')
plt.title('CTC Distribution by Tier')
plt.xlabel('Tier')
plt.ylabel('CTC')
plt.grid(True)
plt.show()
```



Insights:

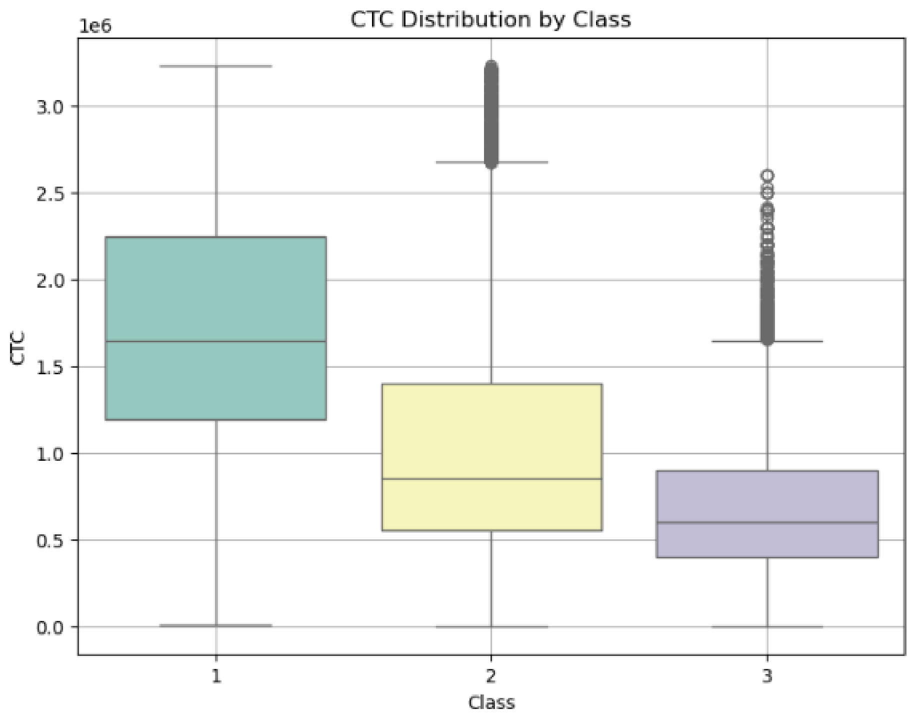
Tier 1 is the highest-paying tier, indicating potentially more senior roles, specialized skills, or companies/industries that offer top-tier compensation.

Tier 2 is a mid-range paying tier, with some individuals managing to achieve higher CTCs, possibly due to exceptional performance, specific niche skills, or movement within the tier.

Tier 3 is the lowest-paying tier, likely representing entry-level positions, less specialized roles, or industries/companies with lower compensation structures. The outliers here suggest that even in lower-paying segments, some roles or individuals can command higher salaries.

4.8 CTC Distribution by Class

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Class', y='ctc', data=df, palette='Set3')
plt.title('CTC Distribution by Class')
plt.xlabel('Class')
plt.ylabel('CTC')
plt.grid(True)
plt.show()
```



Insights:

Class 1 represents the highest-earning group, likely comprising senior-level roles, highly skilled professionnals, or positions in companies/industries that offer premium compensation.

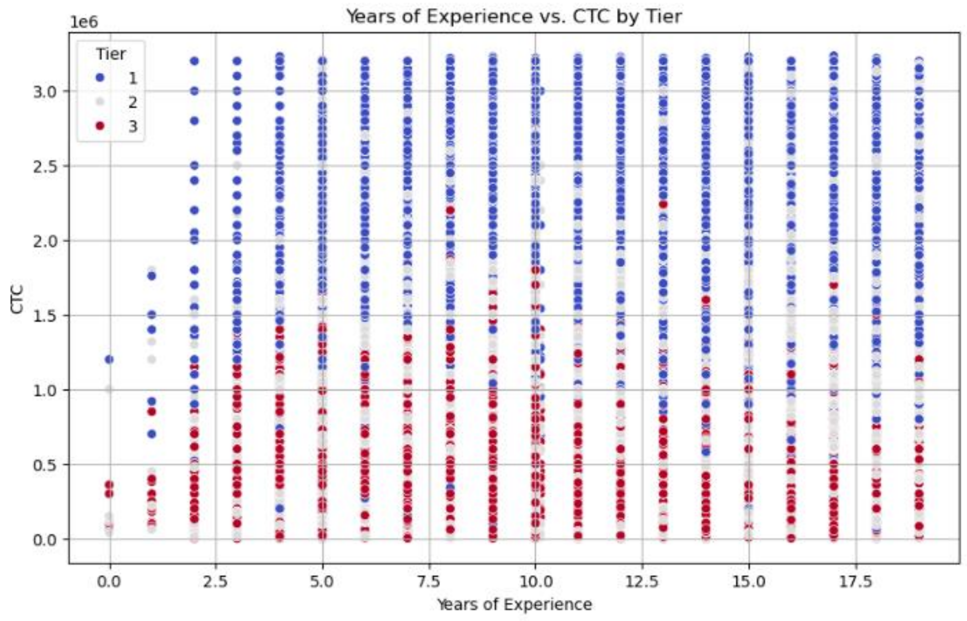
Class 2 represents a middle-income group, with a broader range than Class 3 and a segment of high earners (outliers) who might be excelling within this class or transitioning to higher levels.

Class 3 represents the lowest-earning group, potentially encompassing entry-level positions, less specialized roles, or sectors with generally lower pay scales. The outliers here suggest that even within lower-paying segments, some individuals can achieve significantly higher compensation.

Overall, the analysis strongly suggests that the "Class" variable is a significant determinant of CTC, with a clear progression of earning potential from Class 3 to Class 1.

4.9 Years of Experience vs. CTC by Tier

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Years_of_Experience', y='ctc', hue='Tier', data=df, palette='coolwarm')
plt.title('Years of Experience vs. CTC by Tier')
plt.xlabel('Years of Experience')
plt.ylabel('CTC')
plt.grid(True)
plt.show()
```



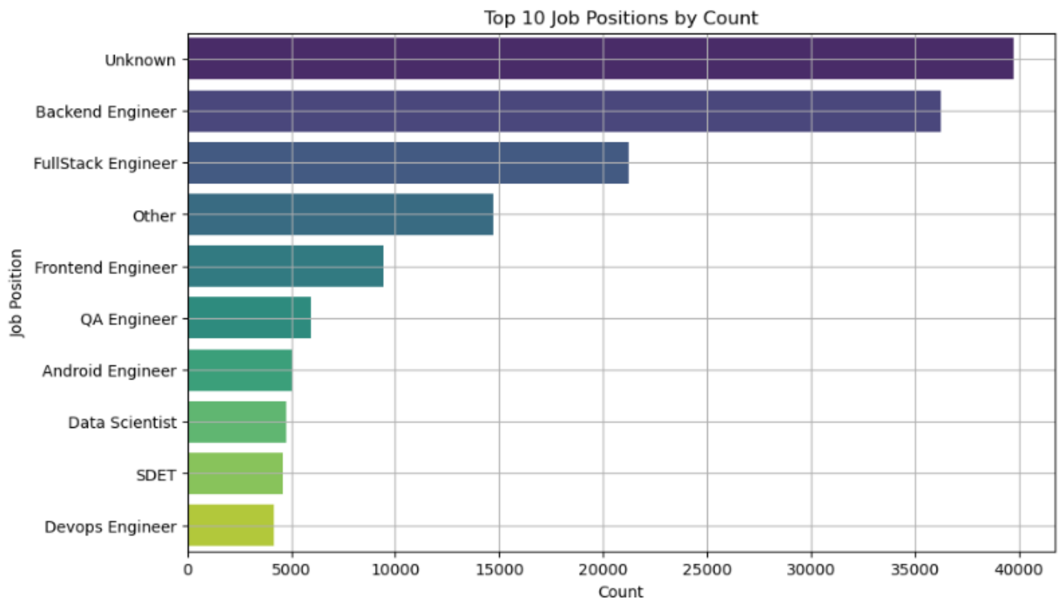
Insights:

The plot shows that **CTC is more influenced by Tier than by Years of Experience**.

- **Tier 1 employees** consistently earn the highest salaries, even with low experience.
- **Tier 3 employees** earn the lowest, regardless of experience.
- There's **no strong upward trend** in CTC with experience across any tier.
- **Conclusion:** Moving to a higher Tier (better company/role) impacts salary more than just gaining experience.

4.10 Top 10 Job Positions by Count

```
top_jobs = df['job_position'].value_counts().head(10)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_jobs.values, y=top_jobs.index, palette='viridis')
plt.title('Top 10 Job Positions by Count')
plt.xlabel('Count')
plt.ylabel('Job Position')
plt.grid(True)
plt.show()
```

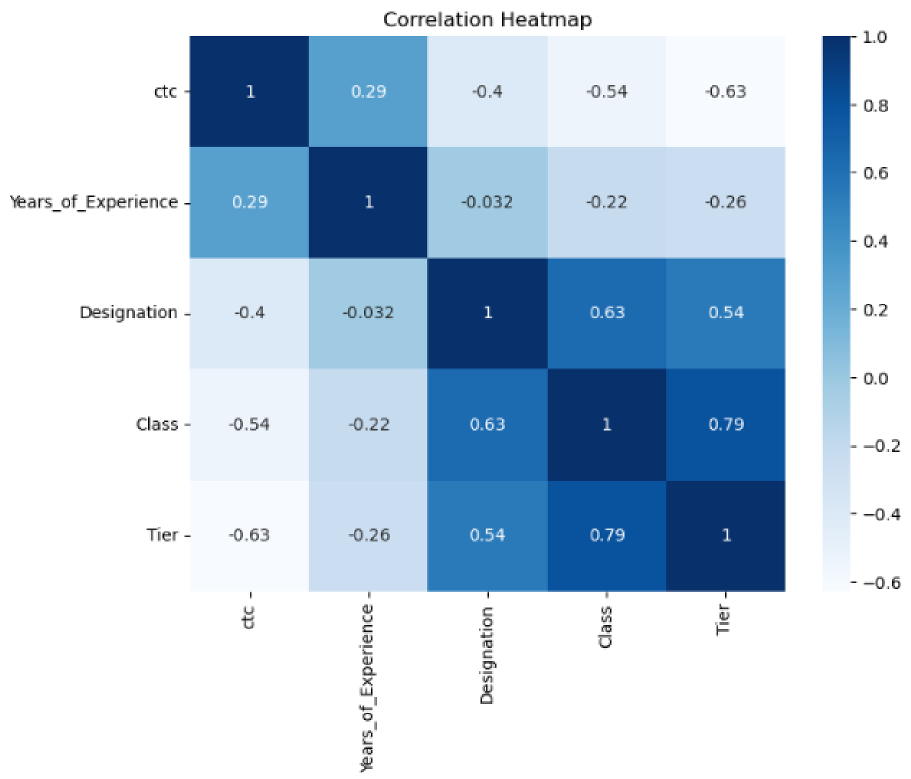


Insights:

The dataset has **significant missing job position data**, making it **hard to draw complete conclusions** about job role distribution.
However, based on available data:
Backend and FullStack Engineers are most commonly reported roles.
There is a wide variety of engineering roles, but **less visibility into roles like Data Science or DevOps** — either due to underreporting or lower counts.

4.11 Correlation Heat map

```
plt.figure(figsize=(8, 6))
sns.heatmap(df[['ctc', 'Years_of_Experience', 'Designation', 'Class', 'Tier']].corr(), annot=True, cmap='Blues')
plt.title('Correlation Heatmap')
plt.show()
```



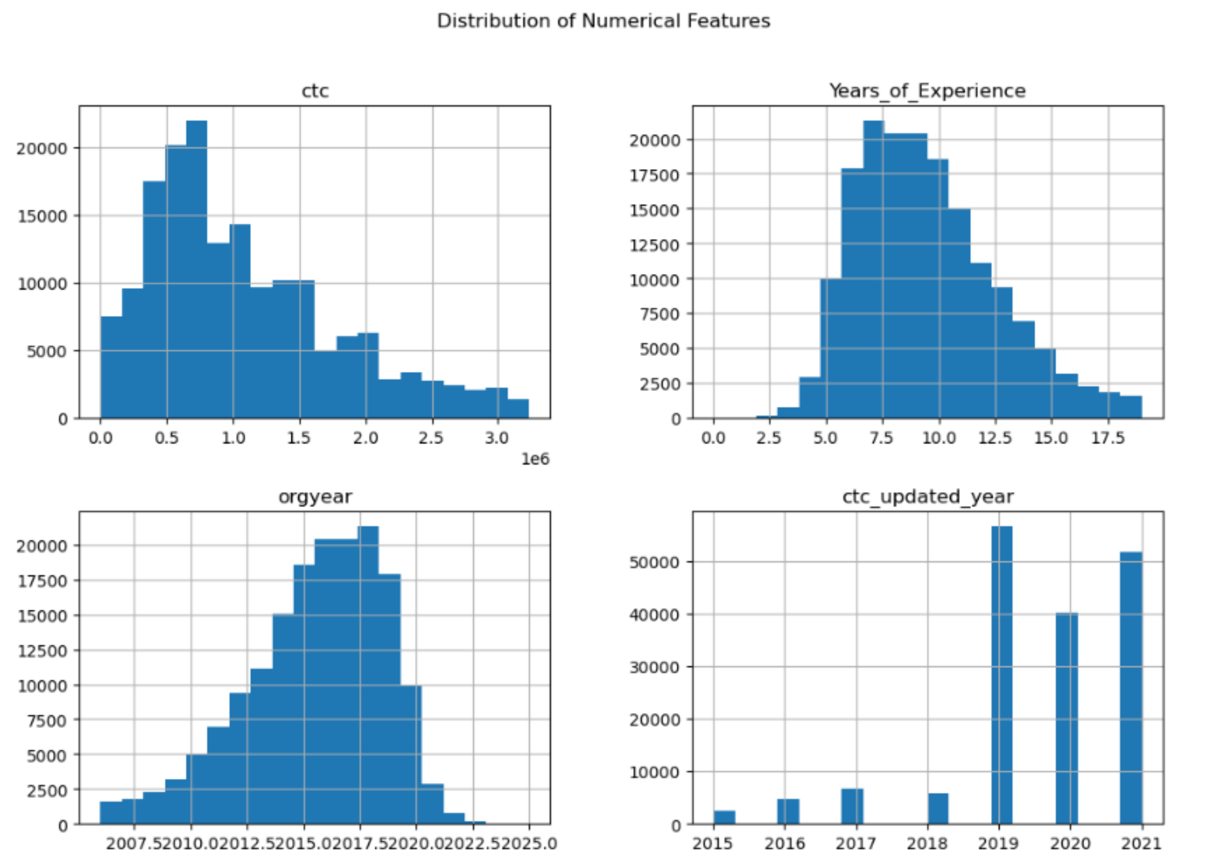
	ctc	Years_of_Experience	Designation	Class	Tier
ctc	1	0.29	-0.4	-0.54	-0.63
Years_of_Experience	0.29	1	-0.032	-0.22	-0.26
Designation	-0.4	-0.032	1	0.63	0.54
Class	-0.54	-0.22	0.63	1	0.79
Tier	-0.63	-0.26	0.54	0.79	1

Insights:

The correlation analysis reveals that **CTC** has a moderate positive relationship with **Years of Experience** (0.29), indicating that salary tends to increase slightly with experience. However, CTC shows a moderate to strong **negative correlation** with **Designation (-0.40)**, **Class (-0.54)**, and **Tier (-0.63)** — suggesting that higher values in these categories (likely representing lower roles or levels) are associated with lower salaries. Interestingly, **Years of Experience has little to no correlation** with these grouping features, implying that job roles, performance, or company structure might influence them more than just experience. Additionally, **Designation, Class, and Tier are strongly correlated with each other** (0.63–0.79), reinforcing that they likely represent related hierarchical structures within organizations.

4.12 Distribution of Numerical Features

```
num_cols = ['ctc', 'Years_of_Experience', 'orgyear', 'ctc_updated_year']

df[num_cols].hist(bins=20, figsize=(12, 8))
plt.suptitle("Distribution of Numerical Features")
plt.show()
```



ctc (Cost to Company):

- The distribution is **right-skewed**, with most values concentrated between ₹0.3M to ₹1.2M.
- There are **a few extreme large values** with very high CTC values, indicating salary anomalies or executive roles.

Years_of_Experience:

- The distribution is **roughly normal**, centered around 8–10 years.
- Few candidates have more than 15 years of experience, while very few have less than 2 years.
- Indicates a **balanced dataset** in terms of professional experience, helpful for predictive modeling.

orgyear:

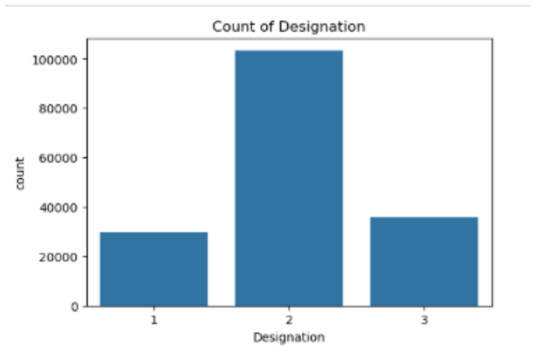
- The values are **clustered between 2010 and 2021**, suggesting that most organizations were founded recently or data pertains to modern firms.
- Some earlier years are present but in **much smaller frequency**.
- Values like 0 and extremely high years were already identified as **outliers** earlie.

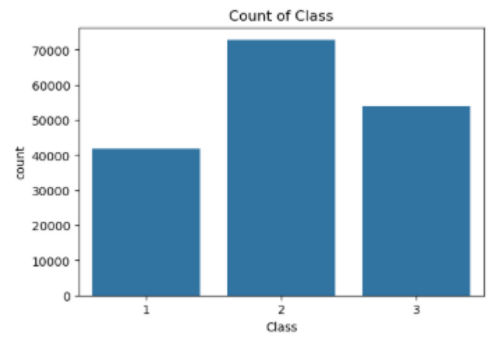
ctc_updated_year:

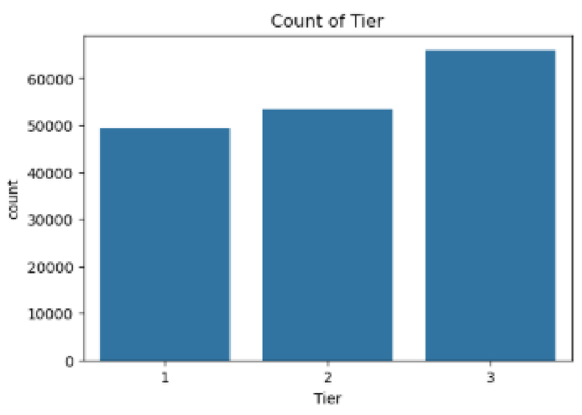
- Most updates happened in **2019, 2020, and 2021**, reflecting recent CTC revisions.
- Sparse updates in earlier years like 2015–2017.
- Implies the data is **relatively recent** and up-to-date for the majority of entries.

4.13 Distribution of Categorical Variables (Designation, Class, and Tier)

```
cat_cols = ['Designation', 'Class', 'Tier']

for col in cat_cols:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=col, data=df)
    plt.title(f"Count of {col}")
    plt.xticks()
    plt.show()
```







Insights:

Count of Designation

- **Designation 2** has the highest number of records.
- **Designation 1 and 3** have significantly fewer and almost equal counts.
- This indicates a majority of individuals are at **mid-level designations**.

Count of Class

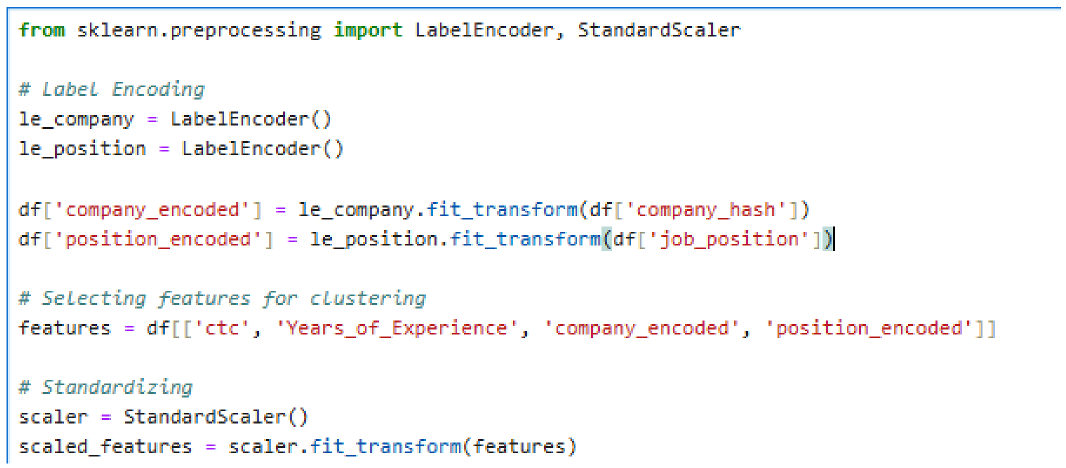
- **Class 2** is the most common, followed by **Class 3**, and then **Class 1**.
- This shows that **Class 2 dominates the classification** in the dataset.
- The distribution suggests a concentration of individuals in the mid-tier class category.

Count of Tier

- **Tier 3** companies have the highest employee representation.
- **Tier 2** comes next, followed by **Tier 1**.
- This implies the dataset is **skewed toward lower-tier companies**, possibly affecting salary or career growth trends.

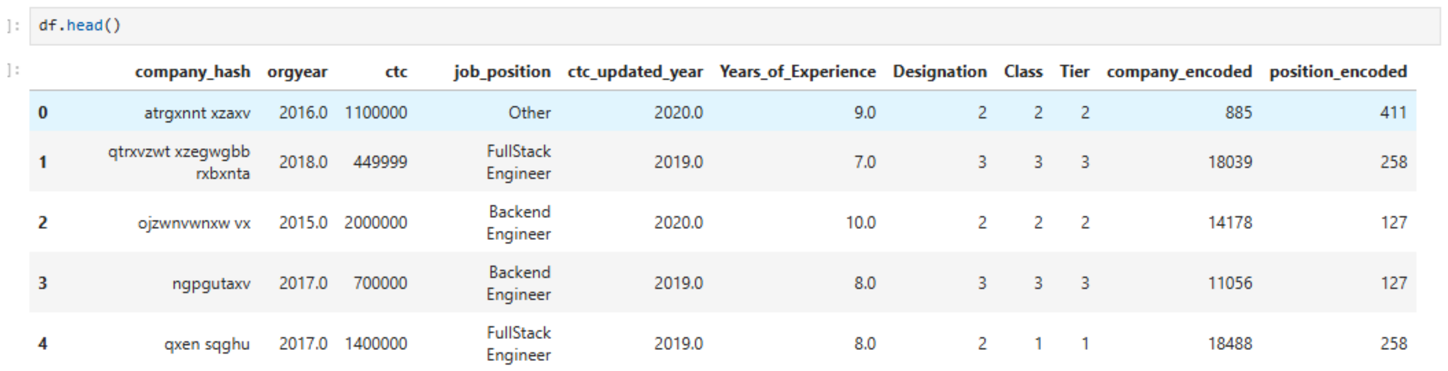
5. Feature Transformation for Clustering Preparation

5.1 Feature Encoding and Standardization for Clustering



```python
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Label Encoding
le_company = LabelEncoder()
le_position = LabelEncoder()

df['company_encoded'] = le_company.fit_transform(df['company_hash'])
df['position_encoded'] = le_position.fit_transform(df['job_position'])

# Selecting features for clustering
features = df[['ctc', 'Years_of_Experience', 'company_encoded', 'position_encoded']]

# Standardizing
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

Converting categorical values to numbers and scaling the data so that clustering algorithms can interpret all features fairly and effectively.



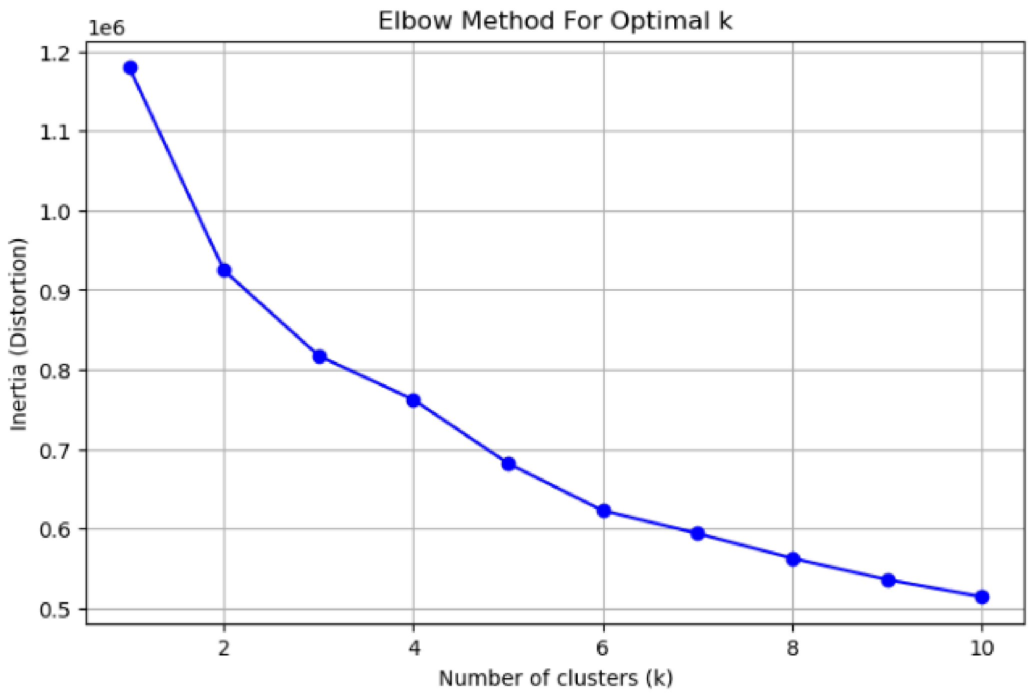
```
]: df.head()
```

	company_hash	orgyear	ctc	job_position	ctc_updated_year	Years_of_Experience	Designation	Class	Tier	company_encoded	position_encoded
0	atrgxnnt xzaxv	2016.0	1100000	Other	2020.0	9.0	2	2	2	885	411
1	qtrxvzwt xzegwgbb rxbxnta	2018.0	449999	FullStack Engineer	2019.0	7.0	3	3	3	18039	258
2	ojzwnvwnxw vx	2015.0	2000000	Backend Engineer	2020.0	10.0	2	2	2	14178	127
3	ngpgutaxv	2017.0	700000	Backend Engineer	2019.0	8.0	3	3	3	11056	127
4	qxen sqghu	2017.0	1400000	FullStack Engineer	2019.0	8.0	2	1	1	18488	258

5.2 Elbow Method for Optimal Clustering

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

inertia = []
k_range = range(1, 11)  # Trying k from 1 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plotting the Elbow curve
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, 'bo-')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia (Distortion)')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()
```

To determine the optimal number of clusters (k) by observing how the clustering inertia (i.e., within-cluster sum of squared distances) changes with different k values.



Insights:
The **sharpest drop** in inertia occurs from k = 1 to k = 3, indicating that 3 clusters significantly reduce intra-cluster distance.**After k = 3**, the rate of decrease in inertia slows down — this flattening suggests that additional clusters don't add meaningful segmentation.Hence, **k = 3 is chosen as the optimal number of clusters**.

5.3 Cluster Assignment and Group Profiling

```
df['Cluster'] = kmeans.labels_

df.groupby('Cluster')[['ctc', 'Years_of_Experience']].mean()
df['Cluster'].value_counts()
```

```
Cluster
1    80246
0    51119
2    37189
Name: count, dtype: int64
```

Insights:

The output shows that Cluster 1 contains the highest number of employees (80,246), followed by Cluster 0 (51,119) and Cluster 2 (37,189). This suggests that Cluster 1 represents the most dominant segment in the workforce, likely comprising average earners with moderate experience. In contrast, Clusters 0 and 2 are comparatively smaller, possibly representing niche groups—such as high CTC individuals with significant experience or junior professionals in early career stages. This distribution helps identify where the majority of employees lie and highlights segments that may need tailored HR strategies or compensation planning.

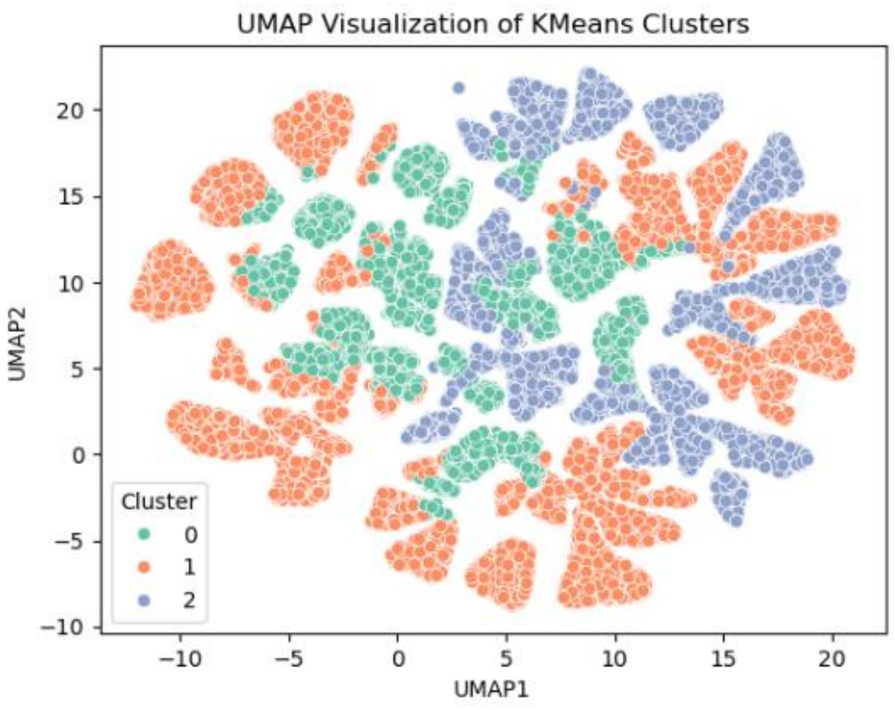
5.4 Clustering with KMeans and Visualization

```
# Example: Features after encoding + scaling
X = df[['orgyear', 'ctc', 'ctc_updated_year', 'Years_of_Experience', 'company_encoded', 'position
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)

# Add cluster labels to the original dataframe
df['Cluster'] = kmeans.labels_
```

```
import umap.umap_ as umap
import seaborn as sns
import matplotlib.pyplot as plt

reducer = umap.UMAP(n_components=2, random_state=42)
embedding = reducer.fit_transform(X_scaled)

umap_df = pd.DataFrame(embedding, columns=['UMAP1', 'UMAP2'])
umap_df['Cluster'] = kmeans.labels_

# Plot
sns.scatterplot(data=umap_df, x='UMAP1', y='UMAP2', hue='Cluster', palette='Set2')
plt.title("UMAP Visualization of KMeans Clusters")
plt.show()
```



Insights:

Issue with UMAP Visualization:

While UMAP is typically effective at preserving complex data structures, in this case the **UMAP-based 2D visualization showed significant overlap among clusters**, indicating that the KMeans algorithm may not have captured distinct groupings clearly.

To address this and gain another perspective, we applied **Principal Component Analysis (PCA)** as an alternative dimensionality reduction method.

5.5 Cluster-wise Statistical Analysis Before PCA

```
df['Cluster'] = kmeans.labels_
```

```
df.groupby('Cluster')[['ctc', 'Years_of_Experience', 'Designation', 'Tier', 'Class']].mean()
```

	ctc	Years_of_Experience	Designation	Tier	Class
Cluster					
0	1.477017e+06	13.339965	1.975875	1.753055	1.803045
1	9.015957e+05	7.814343	2.056943	2.257351	2.188531
2	9.194572e+05	7.810090	2.067545	2.222941	2.179815

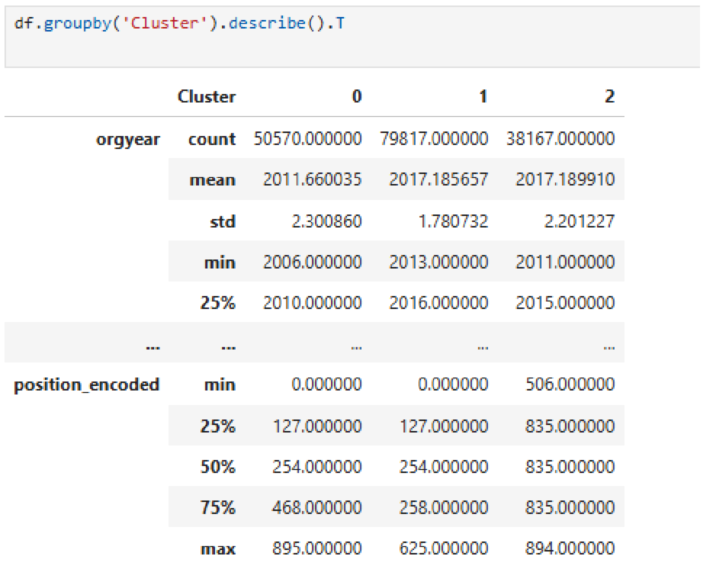
Insights:

Cluster 0 consists of individuals with **higher average CTC (~1.47M)** and **more experience (~13.3 years)**. This group can be seen as more senior or highly compensated professionals, possibly working at better-ranked companies or positions (as reflected by lower Tier and Class values closer to 1).

Cluster 1 and **Cluster 2** both have **lower average CTC (~900k)** and **less experience (~7.8 years)**, indicating they might represent entry- to mid-level professionals. Their Tier and Class values are slightly higher than Cluster 0, reinforcing this interpretation.

Differences between Cluster 1 and 2 are minimal, suggesting possible overlap or redundancy, which PCA can help visualize and further validate.

5.6 Cluster-wise Descriptive Statistics



```
df.groupby('Cluster').describe().T
```

	Cluster	0	1	2
orgyear	count	50570.000000	79817.000000	38167.000000
	mean	2011.660035	2017.185657	2017.189910
	std	2.300860	1.780732	2.201227
	min	2006.000000	2013.000000	2011.000000
	25%	2010.000000	2016.000000	2015.000000
...	...	...	...	...
position_encoded	min	0.000000	0.000000	506.000000
	25%	127.000000	127.000000	835.000000
	50%	254.000000	254.000000	835.000000
	75%	468.000000	258.000000	835.000000
	max	895.000000	625.000000	894.000000

Insights:

Orgyear (Year of company foundation): Cluster 0 has a lower mean orgyear (2011.6), suggesting the companies in this cluster are older and more established.

Clusters 1 and 2 have newer companies (mean around 2017), indicating these clusters are associated with recently founded or startup organizations

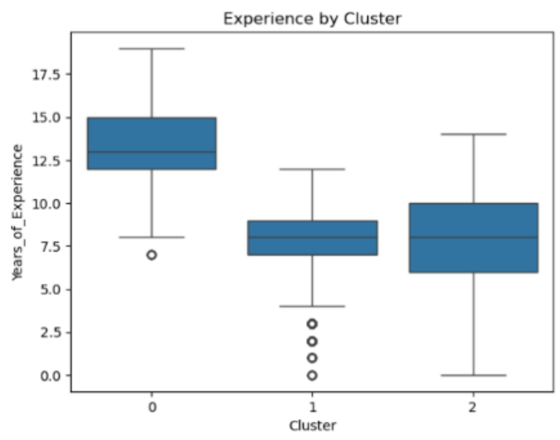
CTC (Compensation): Although not directly visible here, from earlier analysis we know that Cluster 0 had the highest mean CTC, aligning with the trend that older companies may offer more stable or higher salaries.

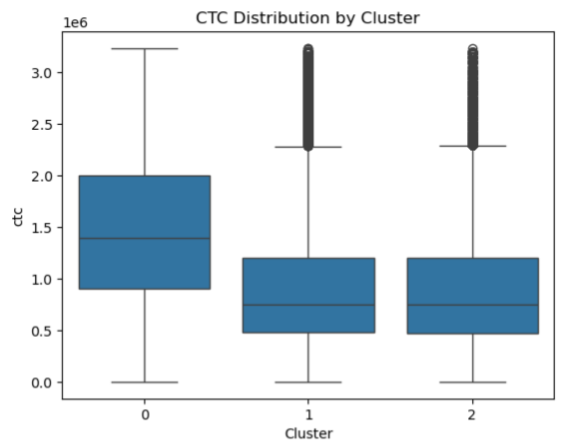
Position Encoding: Cluster 2 has position_encoded values fixed around 835, with no variation (min = 506, max = 894), implying less diversity in job roles.

In contrast, Clusters 0 and 1 have a wider spread in position encodings, indicating more varied job positions.

5.7 Boxplot Analysis of CTC and Experience Across Clusters

```
import seaborn as sns
import matplotlib.pyplot as plt

# Distribution of CTC by Cluster
sns.boxplot(x='Cluster', y='ctc', data=df)
plt.title('CTC Distribution by Cluster')
plt.show()

# Experience by Cluster
sns.boxplot(x='Cluster', y='Years_of_Experience', data=df)
plt.title('Experience by Cluster')
plt.show()
```





Insights:

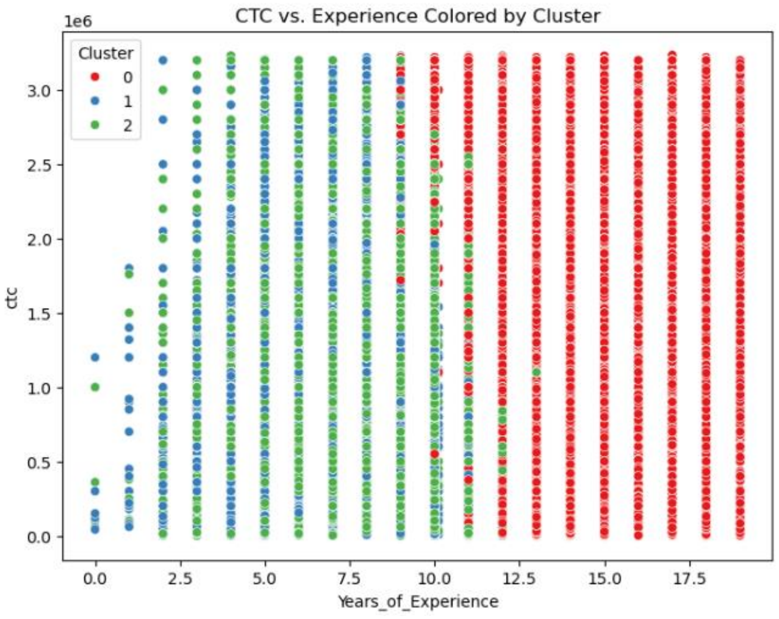
Experience by Cluster:

- **Cluster 0**:
  - Has the **highest median experience** (~13 years).
  - The IQR (box height) is wider, showing more variation.
  - Represents **senior-level professionals**.
- **Cluster 1**:
  - Lowest median experience (~8 years).
  - Compact IQR with lower outliers → likely **junior-level professionals**.
- **Cluster 2**:
  - Similar to Cluster 1 but slightly higher spread.
  - Could represent a **mix of mid-level professionals**.

CTC Distribution by Cluster:

- **Cluster 0**:
  - Clearly has the **highest salaries**, with a median above ₹1.5 million.
  - Wide distribution and many high-end outliers → high earners.
- **Clusters 1 & 2**:
  - Similar median CTC (~₹800k).
  - Densely packed with outliers at the top, suggesting **standard market-level pay** with some higher exceptions.

5.8 Scatter Plot of CTC vs. Experience Colored by Cluster



Insights:

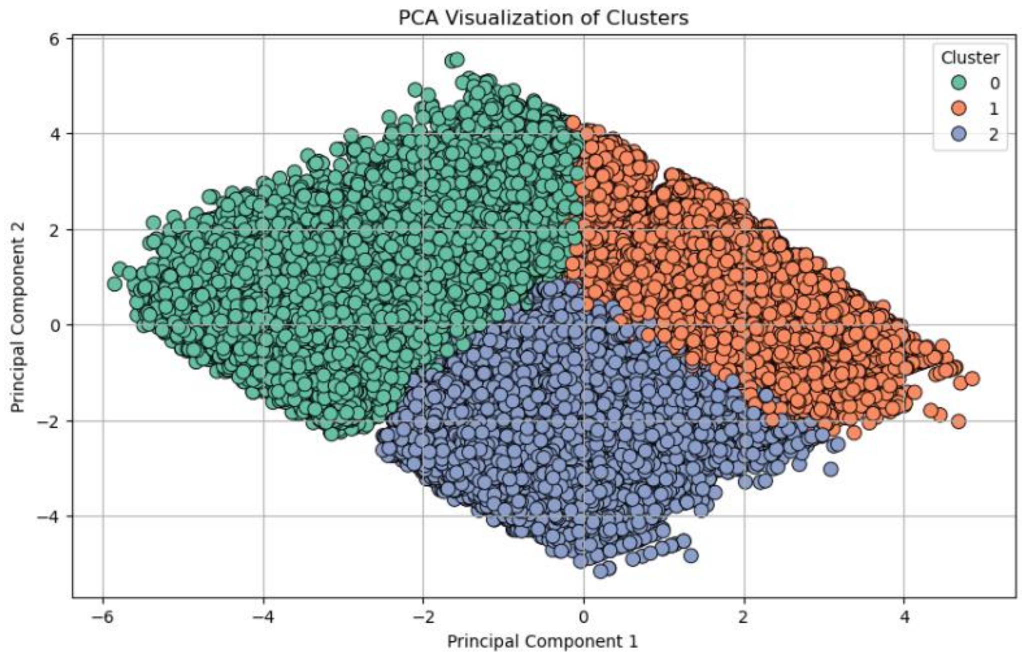
Clear Segmentation by Experience:
- Cluster 0 (Red) dominates in **higher experience range** (≥10 years).
- Clusters 1 and 2 are concentrated in the **0–10 years** range, representing junior to mid-level employees.

CTC Correlation with Experience:
- As expected, **CTC increases with experience**, especially in Cluster 0.
- In lower experience ranges, CTC varies a lot — showing overlaps among Clusters 1 and 2.

Cluster 0 - Senior/High Earners:
- Most red dots (Cluster 0) lie in the upper-right region, indicating **high experience and high CTC**.

Clusters 1 & 2 - Early to Mid Career:
- These clusters overlap and share the early career space, indicating **less separability** between them on just CTC and experience.

5.9 PCA Visualization of KMeans Clusters



Insights:

Cluster 0 (Green): Positioned mainly on the left and top part of the PCA space — likely represents one group with unique patterns in features (possibly senior/high CTC).

Cluster 1 (Orange): Separated to the right side — potentially a different role or CTC bracket.

Cluster 2 (Blue): Occupies the lower portion — indicating distinct patterns from others (maybe junior or lower CTC).

Overall Observation:

Three Distinct Clusters Identified:

- **Cluster 0**: Experienced professionals with the highest average CTC and experience. Likely senior or leadership roles.
- **Cluster 1**: Moderate experience and mid-level compensation — likely representing scalable, core technical roles.
- **Cluster 2**: Employees with similar experience to Cluster 1 but slightly lower CTC — potentially early to mid-career individuals in non-tech or support roles.

High Variability in CTC:

- The presence of extreme CTC outliers suggests inconsistencies or possibly inflated entries.
- A few companies offer significantly higher pay, likely affecting the mean CTC.

Tier and Designation Tags Helped in Differentiation:

- The newly created features (Tier, Class, and Designation) provided strong signals in clustering and allowed deeper business segmentation beyond just CTC.

PCA and UMAP Confirm Clustering Quality:

- Visualizations showed decent separation of clusters in 2D space, confirming that meaningful patterns exist.

Final Recommendation:

- **Refine Pay Structures**: Introduce role-level salary bands using cluster patterns to prevent pay gaps.
- **Implement Tier-Based Growth Paths**: Assign development programs based on the Tier system to personalize upskilling.
- **Clean & Govern Data More Strictly**: Apply validation rules during employee data collection (e.g., cap CTC ranges, valid year entries).
- **Use This Segmentation to Prioritize Retention**: High-CTC, high-experience cluster (Cluster 0) employees represent critical talent — they should be a retention priority.
- **Institutionalize Clustering in HR Systems**: Embed this logic in HR analytics tools to continuously monitor pay equity and workforce structure.

Conclusion:

Through a comprehensive unsupervised learning pipeline, we successfully segmented Scaler's compensation data into three distinct clusters using KMeans clustering. The analysis involved data cleaning, feature engineering, encoding, scaling, and dimensionality reduction through PCA for validation.

The clusters differ significantly in terms of **CTC, years of experience, and role-level tiers**, offering a deeper understanding of employee segmentation. The PCA visualization showed clearly separated clusters, validating the clustering quality.

Jupyter Notebook Analysis

For a detailed view of the full analysis, including code, visualizations please refer to the complete Jupyter notebook available in the PDF format. The notebook documents each step of the analysis process, from data exploration to the final recommendations.

You can access the Jupyter notebook PDF through the **following link**:

JN_Analysis

END