Scaler is an online tech-university offering intensive Computer Science and Data Science courses through live classes conducted by industry experts and tech leaders. Designed to upskill software professionals, Scaler's meticulously structured programs focus on a modern curriculum that aligns with the latest industry trends and technologies. The platform also provides personalized mentorship and career support, helping learners excel in their careers.

Scaler is a product of **InterviewBit**, a renowned platform that specializes in coding preparation and job placements, aimed at bridging the gap between academic learning and industry expectations.

Dataset:

Dataset Link: **scaler_kmeans.csv**
Data Dictionary:

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

Concept Used:

- Manual Clustering
- Unsupervised Clustering - K- means, Hierarchical Clustering

Content:

1. **Data Preprocessing**
2. **Exploratory Data Analysis..**
3. **Manual Clustering.**
    a. **Insights from Manual Clustering.**
4. **Model implementation.**
    a. **K Means clustering.**
    b. **Agglomerative Clustering**
5. **Insights & Recommendations**

1. Data Preprocessing.

```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
from yellowbrick.cluster import KElbowVisualizer
plt.rcParams["figure.figsize"] = (12, 10)
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as mse
```

```
url =
"https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002
/856/original/scaler_clustering.csv"
df_raw = pd.read_csv(url)
df_raw.sample(10)
```

	Unnamed: 0	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
123511	123870	stzuvwn	d192c98f3018d3d354e592022fa22122360ca8851f0be9...	2008.0	2000000	Backend Engineer	2021.0
101366	101590	gwnqg xzw	f8de80889d474107e36209e87c6b73cc0068b024b67133...	2019.0	550000	Backend Engineer	2020.0
162487	163174	vqxkgzv onvnt hzxctqoxnj	0e7332ec8881a98b9f044a0bce3fddb95bcf01ae8629aa...	2017.0	865000	Backend Engineer	2016.0
88977	89131	qmo qgjvr mvzp ge owgnrvza	515be1d90b2656375e37a99c1e3531d9db7b95ad64536e...	2011.0	1850000	Backend Engineer	2019.0
119146	119460	myvqvn utnqgrthb wgqugqvnxgz rxbxnta	91407545a9a5911e8b6998acf116d4a9ab6cb9acc97ccb...	2017.0	1800000	FullStack Engineer	2020.0
70262	70360	bvygmv ytvrnywvqt ucn rna	748840fbf28a4f3caea59b6f54b577ad607fe139c1daee...	2019.0	400000	FullStack Engineer	2020.0
56820	56892	nyvrto	7399a04bda6a5dfd339173b4ff76df3e94706fb06a4a89...	2019.0	1000000	NaN	2021.0
60830	60911	oao ogenuqg ucn rna	7a94e60575ab8fe5fcdb63084770eaee4fc7892cd511e3...	2016.0	229999	Other	2018.0
98937	99143	bxwqgogen	fb0c758bce0d2b58d192fb53386f823191e96bbc152298...	2016.0	2380000	NaN	2020.0
115962	116267	vcvjv xzw	b4be3f38bf88fd3cba3fc101fb34efc6fa345be25d232c...	2008.0	2750000	FullStack Engineer	2019.0

```python
print(df_raw.shape)
print(df_raw.info())
```

```
(205843, 7)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 7 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        205843 non-null  int64  
 1   company_hash      205799 non-null  object 
 2   email_hash        205843 non-null  object 
 3   orgyear           205757 non-null  float64
 4   ctc               205843 non-null  int64  
 5   job_position      153279 non-null  object 
 6   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(2), object(3)
memory usage: 11.0+ MB
None
```

```python
print(df_raw.isnull().sum())
```

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

```python
print(f"Shape of dataset: {df_raw.shape}")
print(f"Data Types:\n{df_raw.dtypes}")
```

```
Shape of dataset: (205843, 7)
Data Types:
Unnamed: 0             int64
company_hash          object
email_hash            object
orgyear              float64
ctc                    int64
job_position          object
ctc_updated_year     float64
dtype: object
```

```python
print("Unique Email Hashes:", df_raw['email_hash'].nunique())
print("Unique Company Hashes:", df_raw['company_hash'].nunique())
```

```
Unique Email Hashes: 153443
Unique Company Hashes: 37299
```

```python
# Summary statistics
print(df_raw.describe())
```

```
         Unnamed: 0       orgyear           ctc  ctc_updated_year
count  205843.000000  205757.000000  2.058430e+05     205843.000000
mean   103273.941786    2014.882750  2.271685e+06       2019.628231
std     59741.306484      63.571115  1.180091e+07          1.325104
min         0.000000       0.000000  2.000000e+00       2015.000000
25%     51518.500000    2013.000000  5.300000e+05       2019.000000
50%    103151.000000    2016.000000  9.500000e+05       2020.000000
75%    154992.500000    2018.000000  1.700000e+06       2021.000000
max    206922.000000   20165.000000  1.000150e+09       2021.000000
```

```python
# Check duplicates
duplicates = df_raw.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")
```

```
Number of duplicate rows: 0
```

```python
import re
df_raw['company_hash'] = df_raw['company_hash'].apply(lambda x:
re.sub('[^A-Za-z0-9]+', '', str(x)) if pd.notnull(x) else x)

df_raw['job_position'] = df_raw['job_position'].apply(lambda x:
re.sub('[^A-Za-z0-9 ]+', '', str(x)) if pd.notnull(x) else x)

df_raw['email_hash'] = df_raw['email_hash'].apply(lambda x:
re.sub('[^A-Za-z0-9]+', '', str(x)) if pd.notnull(x) else x)

print(df_raw[['company_hash', 'job_position',
'email_hash']].head())
```

```
                company_hash        job_position  \
0              atrgxnntxzaxv               Other
1  qtrxvzwtxzegwgbbrxbxnta   FullStack Engineer
2              ojzwnvwnxwvx     Backend Engineer
3                 ngpgutaxv     Backend Engineer
4                 qxensqghu   FullStack Engineer

                                          email_hash
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...
1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...
2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...
3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58...
4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...
```

```python
# Check missing values
missing_values = df_raw.isnull().sum()
print(f"Missing values:\n{missing_values}")
```

```
Missing values:
Unnamed: 0              0
company_hash           44
email_hash              0
orgyear                86
ctc                     0
job_position        52564
ctc_updated_year        0
dtype: int64
```

```python
df_raw['company_hash'] = df_raw['company_hash'].fillna('Unknown')
df_raw['job_position'] = df_raw['job_position'].fillna('Unknown')

# Display missing values again
print(df_raw.isnull().sum())
```

```
Unnamed: 0           0
company_hash         0
email_hash           0
orgyear             86
ctc                  0
job_position         0
ctc_updated_year     0
dtype: int64
```

```python
# Impute missing values using KNN imputation
numerical_columns = ['ctc', 'orgyear', 'ctc_updated_year']  #
Replace with relevant columns
imputer = KNNImputer(n_neighbors=5)
df_raw[numerical_columns] =
imputer.fit_transform(df_raw[numerical_columns])
```

```python
# Check for missing values
df_raw.isnull().sum()
```

	0
Unnamed: 0	0
company_hash	0
email_hash	0
orgyear	0
ctc	0
job_position	0
ctc_updated_year	0

dtype: int64

```python
# Add 'Years of Experience' column
current_year = 2025
df_raw['Years_of_Experience'] = current_year -
df_raw['orgyear'].astype(int)
```

```python
df_raw = df_raw.rename(columns={'Unnamed: 0':
'employee_id','company_hash': 'company','email_hash' :
'email_id','orgyear' : 'Start_year'})
```

```python
df = df_raw[['employee_id', 'company', 'email_id', 'Start_year',
'ctc', 'job_position', 'ctc_updated_year', 'Years_of_Experience']]
df.head()
```

	employee_id	company	email_id	Start_year	ctc	job_position	ctc_updated_year	Years_of_Experience
0	0	atrgxnntxzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000.0	Other	2020.0	9
1	1	qtrxvzwtxzegwgbbrxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999.0	FullStack Engineer	2019.0	7
2	2	ojzwnvwnxwvx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000000.0	Backend Engineer	2020.0	10
3	3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700000.0	Backend Engineer	2019.0	8
4	4	qxensqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400000.0	FullStack Engineer	2019.0	8

```python
sorted(df['Start_year'].fillna(0).astype(int).unique())
```

```python
df = df[~df['Start_year'].isin([0,
 1,
 2,
 3,
 4,
 5,
 6,
 38,
 83,
 91,
```

```
 200,
 201,
 206,
 208,
 209,
 1900, 2023,
 2024,
 2025,
 2026,
 2027,
 2028,
 2029,
 2031,
 2101,
 2106,
 2107,
 2204,
 20165]))]
```

```
# Removing future years
df = df[~(df['Years_of_Experience']<0)]
```

```
sorted(df['Start_year'].fillna(0).astype(int).unique())
```

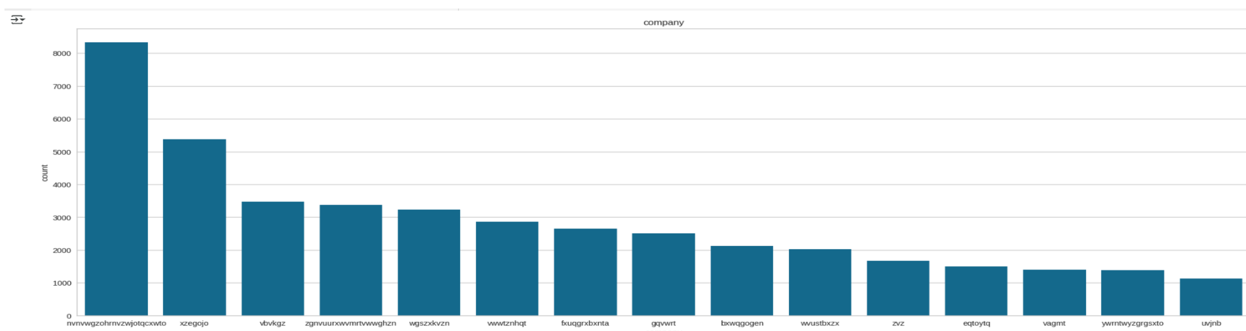
```
[1970,
 1971,
 1972,
 1973,
 1976,
 1977,
 1979,
 1981,
 1982,
 1984,
 1985,
 1986,
 1987,
 1988,
 1989,
 1990,
 1991,
 1992,
 1993,
 1994,
 1995,
 1996,
 1997,
 1998,
 1999,
 2000,
 2001,
 2002,
 2003,
 2004,
 2005,
 2006,
 2007,
 2008,
 2009,
 2010,
 2011,
 2012,
 2013,
 2014,
 2015,
 2016,
 2017,
 2018,
 2019,
 2020,
 2021,
 2022]
```

Removing future years, as this case is impossible to happen, also removing single digit years.

2. Exploratory Data Analysis.

Univariate Analysis.

```
categroical_columns = ['company','job_position','Start_year','ctc_updated_year']
numerical_columns = ['ctc','Years_of_Experience']
for i in categroical_columns:
    tmp = df.copy()
    tmp['count'] = 1
    tmp =
tmp.groupby(i).sum()['count'].reset_index().sort_values('count',ascending=False).h
ead(15)
    plt.figure(figsize=(25,8))
    sns.barplot(data=tmp,y='count',x=i).set(title=i)
    plt.show()
```



Insights:

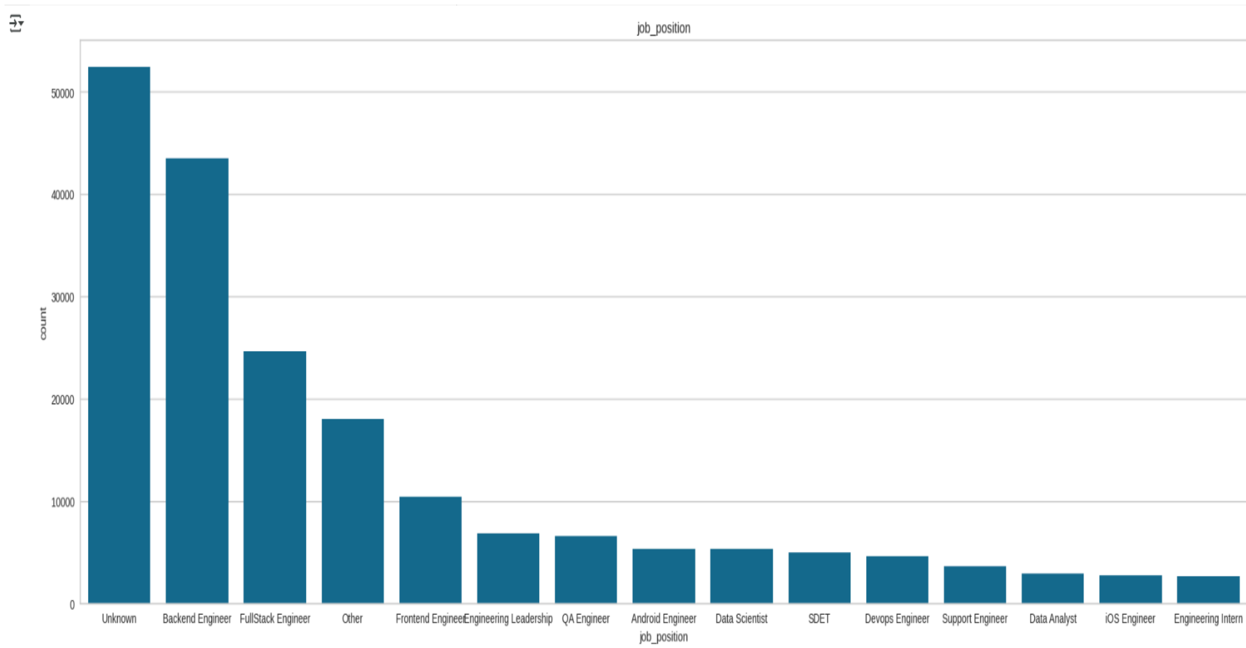
Top Companies by CTC Count:

- "nvnvwgzohrnvzwjoqcxwto" leads with over 80000 CTC counts, followed by "xzegojo" with more than 50000, and "vbvkgz" with over 30000.

Companies with Lower CTC Counts:

- "unjnb" has fewer than 20000 CTC counts, and "ywrntwzgrgsxto" is just below 20000, indicating fewer high-paying positions.

Implications: Companies like "nvnvwgzohrnvzwjoqcxwto" and "xzegojo" offer higher CTCs, while "unjnb" and "ywrntwzgrgsxto" have lower CTC distributions.



Insights:

Unknown Job Position:

- The "Unknown" category dominates with over 50,000 CTC counts, likely due to missing labels in the dataset.

Top Job Positions by CTC Count:

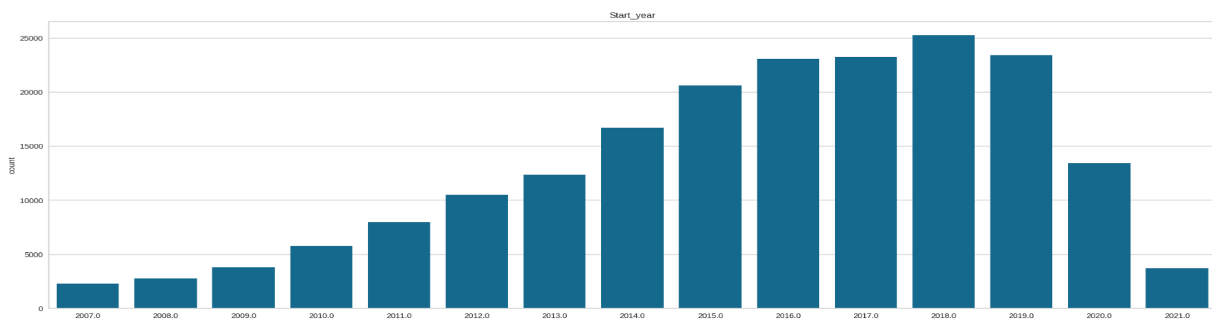
- **Back End Developer** ranks as the top job position with more than 45,000 CTC counts, making it the most prevalent role after "Unknown."
- **Full Stack Enginee**r follows with over 25,000 CTC counts.
- **Other job roles** collectively account for more than 18,000 counts.

Mid-tier Job Positions:

- **Front End Developer** has over 10,000 CTC counts.

Least Common Job Positions:

- **Engineer Intern** has the lowest count at around 5,000, followed by **iOS Engineer** with 8,000 and **Data Analyst** with fewer than 10,000 CTC counts.

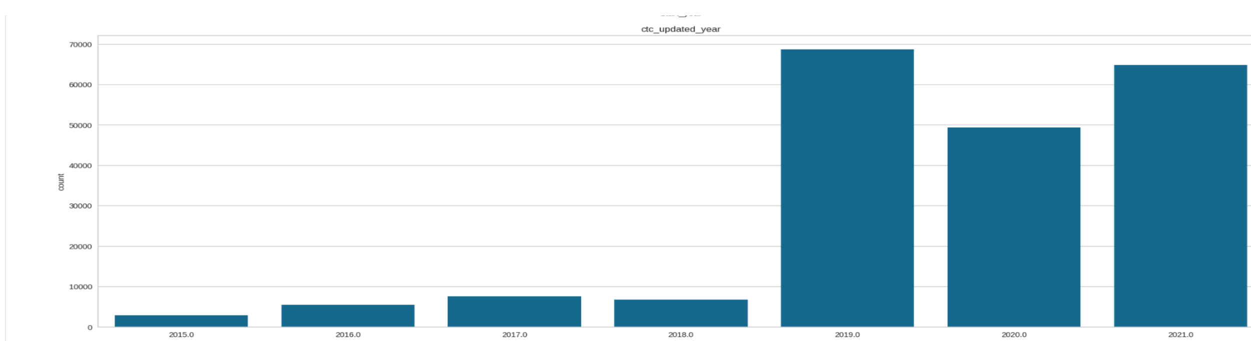


Top Start Years by CTC Count:

- 2018 has the highest CTC count, with more than 25,000, indicating a peak in hiring or compensation during this year.
- 2019 and 2017 are tied for second place, each with 20,000 counts.
- 2016 follows with around 19,000 counts, marking it as the third-highest year for employee start dates.

Lower Start Years:

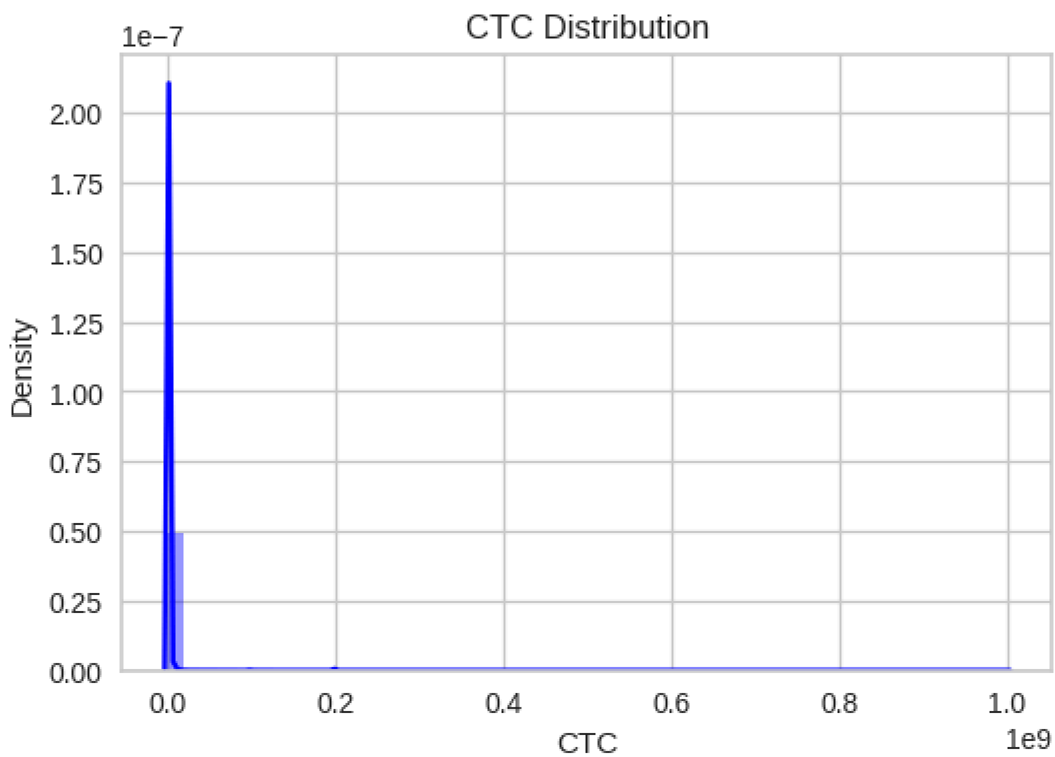
- 2007 has the lowest CTC count, with only 4,000, suggesting fewer employees started their job in that year with higher compensation.
- 2008 and 2009 follow with 5,000 and 6,000 counts, respectively, indicating a gradual increase in employee start dates and CTCs as the years progress.



Insights:

- The highest CTC updates occurred in 2019 with over 70,000 counts, followed by 2021 and 2020, which saw a dip likely due to COVID-19. The lowest counts were in 2015 (6,000) and 2016 (8,000).

```
plt.figure(figsize=(6, 4))
sns.distplot(df['ctc'], kde=True, color='blue')
plt.title('CTC Distribution')
plt.xlabel('CTC')
plt.ylabel('Density')
plt.show()
```



The plot displays a wide range of values, making comparisons challenging. Scaling the column might help to normalize the data and enhance visualization clarity.

Outlier Removal using IQR:

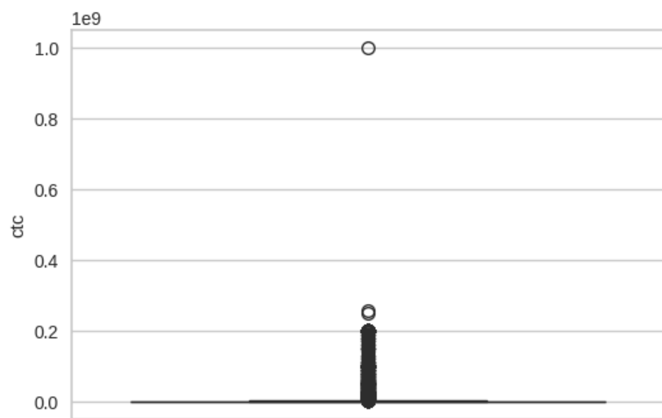
```python
outlier_removed = df.copy()

# Print initial shape
print(f"Initial Shape: {outlier_removed.shape}")

# Columns to check for outliers
cols = ['ctc']  # Specify one or more columns

# Calculate the Interquartile Range (IQR)
Q1 = outlier_removed[cols].quantile(0.25)  # 25th percentile
Q3 = outlier_removed[cols].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1

# Filter the dataset by removing outliers
outlier = outlier_removed[~((outlier_removed[cols] < (Q1 - 1.5 *
IQR)) | (outlier_removed[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]

# Print final shape
print(f"Shape after removing outliers: {outlier_removed.shape}")
```

```
Initial Shape: (205461, 8)
Shape after removing outliers: (205461, 8)
```

Insights:

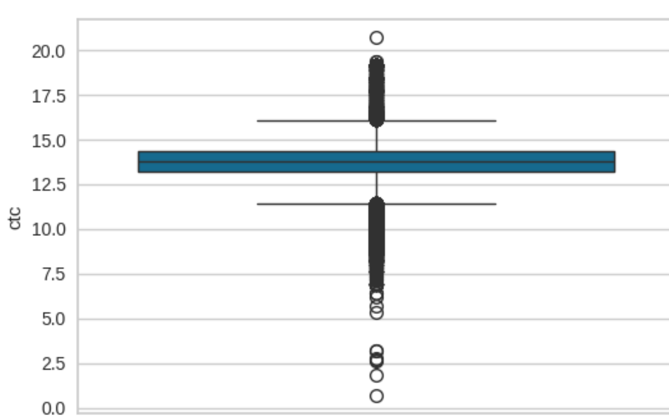
- Outlier removal using the IQR method did not impact the dataset, as the shape remained unchanged:
  Initial Shape: (205,461, 8)
  Shape after removing outliers: (205,461, 8)
- This suggests there were no extreme values identified as outliers.

```
plt.figure(figsize=(6, 4))
out = outlier_removed['ctc']
sns.boxplot(out)
plt.show()
```

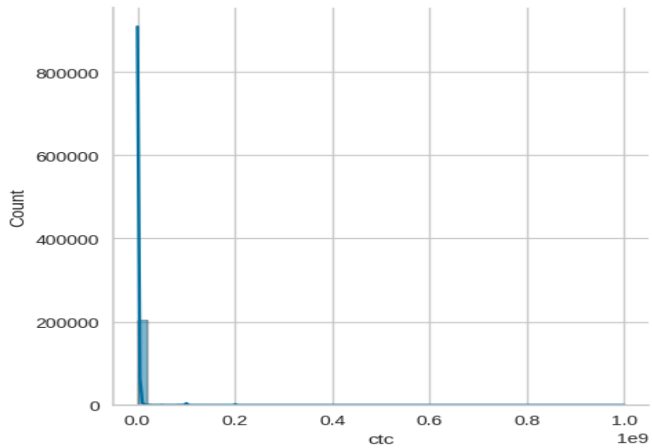


Checking by converting the removed outliers to log

```
plt.figure(figsize=(6, 4))
out = np.log(outlier_removed['ctc'])
sns.boxplot(out)
plt.show()
```

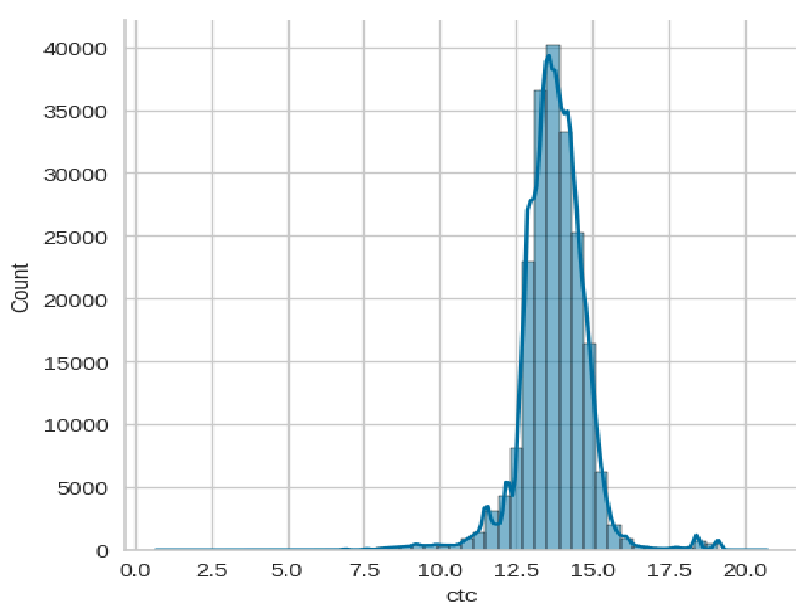


```
out = outlier_removed['ctc']
sns.displot(out,kde=True,bins=50)
plt.show()
```

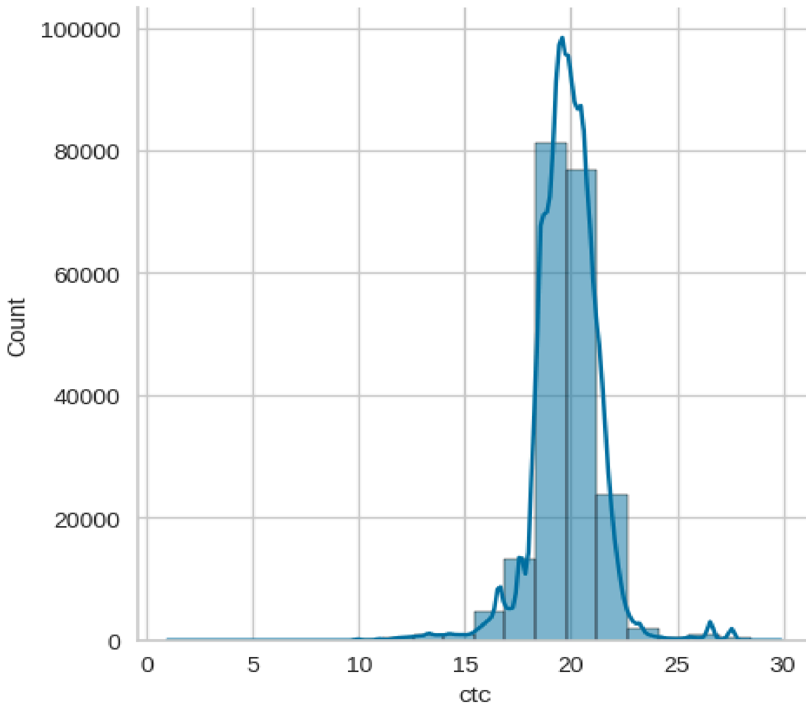


Checking by converting the removed outliers to log

```
out = np.log(outlier_removed['ctc'])
sns.displot(out,kde=True,bins=50)
plt.show()
```



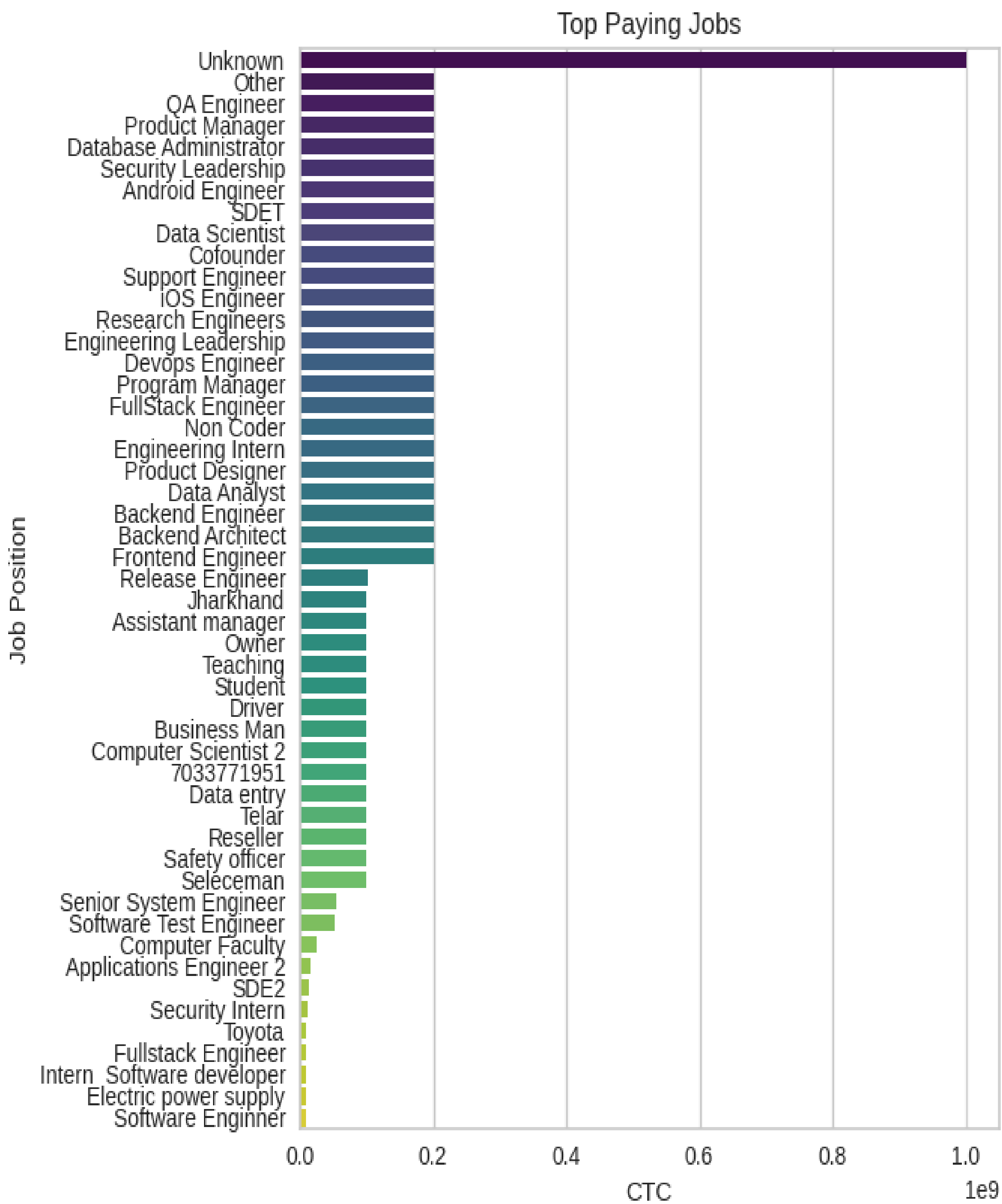
```
out2 = np.log2(outlier_removed['ctc'])
sns.displot(out2,kde=True,bins=20)
plt.show()
```

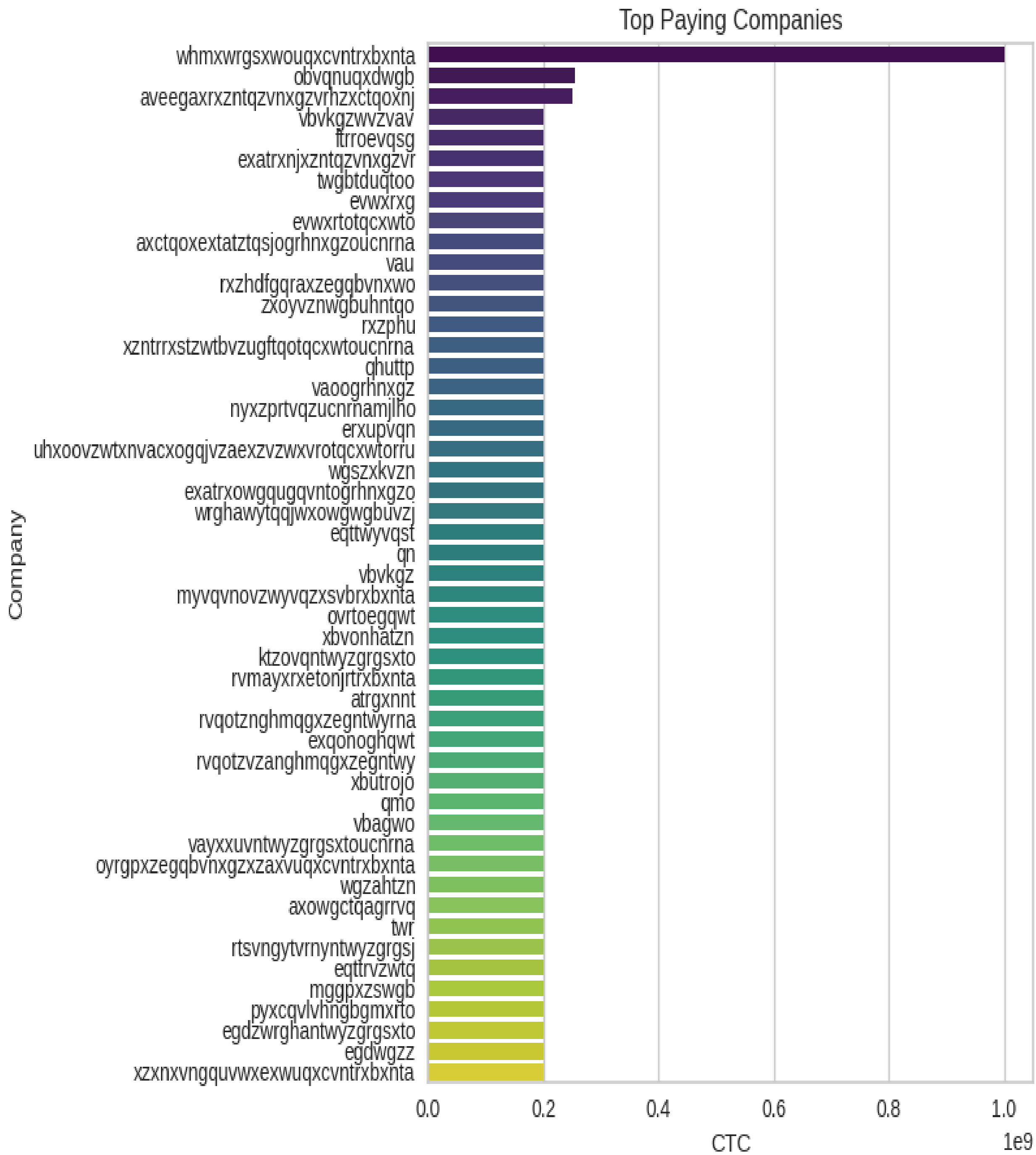


Multivariate Anyalisis:

```
# top-paying jobs
tmp_jobs = df.copy()
tmp_jobs =
tmp_jobs.groupby(['job_position']).max()['ctc'].reset_index().sort_
values('ctc', ascending=False).head(50)

#top-paying companies
```

```
tmp_companies = df.copy()
tmp_companies =
tmp_companies.groupby(['company']).max()['ctc'].reset_index().sort_
values('ctc', ascending=False).head(50)

plt.figure(figsize=(15, 7))

# Plot for Top Paying Jobs
plt.subplot(1, 2, 1)
sns.barplot(data=tmp_jobs, x='ctc', y='job_position',
palette='viridis').set(title="Top Paying Jobs")
plt.xlabel('CTC')
plt.ylabel('Job Position')

# Plot for Top Paying Companies
plt.subplot(1, 2, 2)
sns.barplot(data=tmp_companies, x='ctc', y='company',
palette='viridis').set(title="Top Paying Companies")
plt.xlabel('CTC')
plt.ylabel('Company')

# Display the plots
plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()

# List the top job positions and companies
top_job_positions = list(tmp_jobs['job_position'])
top_companies = list(tmp_companies['company'])

print("Top Job Positions:")
print(top_job_positions)

print("Top Companies:")
print(top_companies)
```


Top Paying Jobs
Unknown
Other
QA Engineer
Product Manager
Database Administrator
Security Leadership
Android Engineer
SDET
Data Scientist
Cofounder
Support Engineer
iOS Engineer
Research Engineers
Engineering Leadership
Devops Engineer
Program Manager
FullStack Engineer
Non Coder
Engineering Intern
Product Designer
Data Analyst
Backend Engineer
Backend Architect
Frontend Engineer
Release Engineer
Jharkhand
Assistant manager
Owner
Teaching
Student
Driver
Business Man
Computer Scientist 2
7033771951
Data entry
Telar
Reseller
Safety officer
Seleceman
Senior System Engineer
Software Test Engineer
Computer Faculty
Applications Engineer 2
SDE2
Security Intern
Toyota
Fullstack Engineer
Intern Software developer
Electric power supply
Software Enginner
Job Position
0.0
0.2
0.4
0.6
0.8
1.0
CTC
1e9


Top Paying Companies
Company
CTC
1e9
0.0
0.2
0.4
0.6
0.8
1.0
whmxwrgsxwouqxcvntrxbxnta
obvqnuqxdwgb
aveegaxrxzntqzvnxgzvrhzxctqoxnj
vbvkgzwvzvav
ftrroevqsg
exatrxnjxzntqzvnxgzvr
twgbtduqtoo
evwxrxg
evwxrtotqcxwto
axctqoxextatztqsjogrhnxgzoucnrna
vau
rxzhdfgqraxzegqbvnxwo
zxoyvznwgbuhntqo
rxzphu
xzntrrxstzwtbvzugftqotqcxwtoucnrna
qhuttp
vaoogrhnxgz
nyxzprtvqzucnrnamjlho
erxupvqn
uhxoovzwtxnvacxogqjvzaexzvzwxvrotqcxwtorru
wgszxkvzn
exatrxowgqugqvntogrhnxgzo
wrghawytqqjwxowgwgbuvzj
eqttwyvqst
qn
vbvkgz
myvqvnovzwyvqzxsvbrxbxnta
ovrtoegqwt
xbvonhatzn
ktzovqntwyzgrgsxto
rvmayxrxetonjrtrxbxnta
atrgxnnt
rvqotznghmqgxzegntwyrna
exqonoghqwt
rvqotzvzanghmqgxzegntwy
xbutrojo
qmo
vbagwo
vayxxuvntwyzgrgsxtoucnrna
oyrgpxzegqbvnxgzxzaxvuqxcvntrxbxnta
wgzahtzn
axowgctqagrrvq
twr
rtsvngytvrnyntwyzgrgsj
eqttrvzwtq
mggpxzswgb
pyxcqvlvhngbgmxrto
egdzwrghantwyzgrgsxto
egdwgzz
xzxnxvngquvwxexwuqxcvntrxbxnta

3. Manual Clustering.

```
# Calculate the 5-point summary of CTC for each combination of
Company, Job Position, and Years of Experience
ctc_summery =
df_outlier_removed.groupby(['company','job_position','Years_of_Expe
rience'])['ctc'].describe()
ctc_summery = ctc_summery.reset_index()
ctc_summery
```

	company	job_position	Years_of_Experience	count	mean	std	min	25%	50%	75%	max
0	0	Other	5	1.0	100000.0	NaN	100000.0	100000.0	100000.0	100000.0	100000.0
1	0	Unknown	5	1.0	100000.0	NaN	100000.0	100000.0	100000.0	100000.0	100000.0
2	0000	Other	8	1.0	300000.0	NaN	300000.0	300000.0	300000.0	300000.0	300000.0
3	01ojztqsj	Android Engineer	9	1.0	270000.0	NaN	270000.0	270000.0	270000.0	270000.0	270000.0
4	01ojztqsj	Frontend Engineer	14	1.0	830000.0	NaN	830000.0	830000.0	830000.0	830000.0	830000.0
...	...	...	...	...	...	...	...	...	...	...	...
112923	zz	Unknown	16	1.0	500000.0	NaN	500000.0	500000.0	500000.0	500000.0	500000.0
112924	zzbztdnstzvacxogqjucnrna	FullStack Engineer	8	1.0	600000.0	NaN	600000.0	600000.0	600000.0	600000.0	600000.0
112925	zzbztdnstzvacxogqjucnrna	Unknown	8	1.0	600000.0	NaN	600000.0	600000.0	600000.0	600000.0	600000.0
112926	zzgato	Unknown	11	1.0	130000.0	NaN	130000.0	130000.0	130000.0	130000.0	130000.0
112927	zzzbzb	Other	35	1.0	720000.0	NaN	720000.0	720000.0	720000.0	720000.0	720000.0

112928 rows × 11 columns

Insights:

- **Consistency in Counts:** Each record corresponds to a single entry (`count = 1`), meaning there is no aggregation across records.
- **Job Positions and Companies:**The data includes various job positions like "Other," "Unknown," "Android Engineer," "Frontend Engineer," etc., across multiple companies.
- **CTC (Mean and Spread):** Each record has a unique CTC value for `mean`, `min`, `max`, and quartiles, indicating that there is no variability (`std = NaN`). For example: An Android Engineer at 01ojztqsj with 9 years of experience has a fixed CTC of 270,000. A Frontend Engineer at the same company with 14 years of experience has a higher CTC of 830,000.
- **"Unknown" and "Other"** Job Positions: These categories are frequent, potentially representing missing or generalized job titles.
- **Highest CTC:** The highest recorded CTC is 830,000 for a Frontend Engineer with 14 years of experience.
- **Years of Experience:** The range of experience spans from 5 years to 35 years, with a noticeable correlation between higher experience and higher CTC.
- **Data Quality:** The lack of variability within records (e.g., `std = NaN` and identical quartiles) indicates the data might represent predefined salary bands or manually entered values.

```python
# Create the 'designation' flag based on CTC comparison with
average department CTC
def set_designation_flag(row):
    if row['ctc'] < row['avg_ctc_dept']:
        return 1  # Below average
    elif row['ctc'] == row['avg_ctc_dept']:
        return 2  # Equal to average
    else:
        return 3  # Above average

df['designation'] = df.apply(set_designation_flag, axis=1)

# Calculate the average CTC by company, job position, and years of
experience for 'Class' flag
ctc_avg_by_class = df.groupby(['company', 'job_position',
'Years_of_Experience'])['ctc'].mean().reset_index()
ctc_avg_by_class = ctc_avg_by_class.rename(columns={'ctc':
'avg_ctc_class'})

df = df.merge(ctc_avg_by_class, on=['company', 'job_position',
'Years_of_Experience'], how='left')

# Create the 'Class' flag based on CTC comparison with average
company & job position CTC
def set_class_flag(row):
    if row['ctc'] < row['avg_ctc_class']:
        return 1  # Below average
    elif row['ctc'] == row['avg_ctc_class']:
        return 2  # Equal to average
    else:
        return 3  # Above average

df['Class'] = df.apply(set_class_flag, axis=1)
```

```python
# Calculate the average CTC by company and years of experience for 'Tier' flag
ctc_avg_by_tier = df.groupby(['company', 'Years_of_Experience'])['ctc'].mean().reset_index()
ctc_avg_by_tier = ctc_avg_by_tier.rename(columns={'ctc': 'avg_ctc_tier'})

# Merge the company-level average CTC back into the original dataset
df = df.merge(ctc_avg_by_tier, on=['company', 'Years_of_Experience'], how='left')

# Create the 'Tier' flag based on CTC comparison with average company CTC
def set_tier_flag(row):
    if row['ctc'] < row['avg_ctc_tier']:
        return 1  # Below average
    elif row['ctc'] == row['avg_ctc_tier']:
        return 2  # Equal to average
    else:
        return 3  # Above average
df['Tier'] = df.apply(set_tier_flag, axis=1)
df
```

employee_id	company	email_id	Start_year	ctc	job_position	ctc_updated_year	Years_of_Experience	avg_ctc_dept	designation	avg_ctc_class	Class	avg_ctc_tier	Tier
0	atrgxnntxzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000.0	Other	2020.0	9	1.100000e+06	2	1.100000e+06	2	1.100000e+06	2
1	qtrxvzwtxzegwgbbrxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999.0	FullStack Engineer	2019.0	7	7.373787e+05	1	7.742856e+05	1	7.373787e+05	1
2	ojzwnvwnxwvx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000000.0	Backend Engineer	2020.0	10	2.000000e+06	2	2.000000e+06	2	2.000000e+06	2
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700000.0	Backend Engineer	2019.0	8	1.322632e+06	1	1.158571e+06	1	1.322632e+06	1
4	qxensqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400000.0	FullStack Engineer	2019.0	8	1.400000e+06	2	1.400000e+06	2	1.400000e+06	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
206918	vuurtxzw	70027b728c8ee901fe979533ed94ffda97be08fc23f33b...	2008.0	220000.0	Unknown	2019.0	17	2.200000e+05	2	2.200000e+05	2	2.200000e+05	2
206919	husqvawgb	7f7292ffad724ebbe9ca860f515245368d714c84705b42...	2017.0	500000.0	Unknown	2020.0	8	1.085882e+06	1	1.150000e+06	1	1.085882e+06	1
206920	vwwgrxnt	cb25cc7304e9a24facda7f5567c7922ffc48e3d5d6018c...	2021.0	700000.0	Unknown	2021.0	4	6.571429e+05	3	6.666667e+05	3	6.571429e+05	3
206921	zgnvuurxwvmrt	fb46a1a2752f5f652ce634f6178d0578ef6995ee59f6c8...	2019.0	5100000.0	Unknown	2019.0	6	5.891461e+06	1	5.920732e+06	1	5.891461e+06	1
206922	bgqsvzonvzrtj	0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f...	2014.0	1240000.0	Unknown	2016.0	11	2.012674e+06	1	1.693333e+06	1	2.012674e+06	1

rows × 14 columns

1. Top 10 employees(earning more than most of the employees in the company)Tier 1.

```
top_10_tier_1 = df[df['Tier'] == 1].groupby('company').apply(lambda
x: x.nlargest(10, 'ctc')).reset_index(drop=True)
top_10_tier_1 = top_10_tier_1.head(10)
print(top_10_tier_1[['employee_id', 'company', 'job_position',
'ctc', 'Tier']])
```

```
   employee_id        company            job_position        ctc  Tier
0        13280        10nxbto       FullStack Engineer   410000.0     1
1       100004        10nxbto        Frontend Engineer   400000.0     1
2       109189        10nxbto       FullStack Engineer   400000.0     1
3        11130  159ogrhnxgzo                    Other   500000.0     1
4        15387  159ogrhnxgzo                  Unknown   500000.0     1
5        62158            1bs        Backend Engineer   1800000.0     1
6        93599            1bs        Product Designer   1730000.0     1
7       115696            1bs        Backend Engineer   1600000.0     1
8       203546            1bs  Engineering Leadership   1600000.0     1
9        51815            1bs                   Other   1500000.0     1
<ipython-input-195-c7e81b0e96fb>:1: DeprecationWarning: DataFrameGroupBy.appl
  top_10_tier_1 = df[df['Tier'] == 1].groupby('company').apply(lambda x: x.nl
```

Insights:

- **Top Earners Across Companies:** Employees in Tier 1 have significantly higher CTCs compared to their peers.For example, employees in companies like 10nxbto and 159ogrhnxgzo earn between 400,000 and 500,000, while in 1bs, top earners make much higher amounts, ranging from 1,500,000 to 1,700,000.
- **Notable Job Positions:** High-earning positions include FullStack Engineer, Frontend Engineer, Backend Engineer, Product Designer, and Engineering Leadership.
- **Company Overview:** 1bs stands out with multiple employees earning above 1 million, including Backend Engineers, Product Designers, and Engineering Leadership. 10nxbto and 159ogrhnxgzo have a few top earners, but the CTCs are generally lower than those at 1bs.
- **High Salary Profiles:** Backend Engineers and Engineering Leadership roles in 1bs lead with CTCs around 1,600,000 to 1,800,000. Frontend Engineer and FullStack Engineer positions at 10nxbto also make it to the top, with 400,000 to 410,000 CTCs.

2. Top 10 employees of data science in each company earning more than their peers - Class 1.

```
top_10_class_1_data_science = df[(df['job_position'] == 'Data Scientist') & (df['Class'] == 1)]
top_10_class_1_data_science =
top_10_class_1_data_science.groupby('company').apply(lambda x:
x.nlargest(10, 'ctc')).reset_index(drop=True)
top_10_class_1_data_science = top_10_class_1_data_science.head(10)
top_10_class_1_data_science =
top_10_class_1_data_science[['employee_id','job_position', 'ctc',
'Class']]
top_10_class_1_data_science
```

	employee_id	job_position	ctc	Class
0	106613	Data Scientist	1200000.0	1
1	161190	Data Scientist	880000.0	1
2	80279	Data Scientist	800000.0	1
3	122140	Data Scientist	700000.0	1
4	54042	Data Scientist	850000.0	1
5	107906	Data Scientist	1550000.0	1
6	178842	Data Scientist	1250000.0	1
7	144082	Data Scientist	1200000.0	1
8	200229	Data Scientist	1200000.0	1
9	176859	Data Scientist	1190000.0	1

- **Top Earners in Data Science:** The highest earning Data Scientist in this group is 107906, with a CTC of 1,550,000, followed by others earning between 880,000 and 1,200,000.
- **Consistent High Earnings:** Many employees earn around 1,200,000 or more, showing that these individuals are well-compensated compared to their peers in the same role.
- **Wide Salary Range:** CTCs for these Data Scientists range from 700,000 to 1,550,000, suggesting varying levels of experience or company-specific salary structures.
- **Role Consistency:** All top earners share the same job position of Data Scientist, but their salaries differ, likely reflecting experience, seniority, or performance within the company.

Notable High Earners: 107906 stands out with the highest salary, while others like 161190, 54042, and 80279 also earn significantly more than most of their peers in similar roles.

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

```
bottom_10_class_3_data_science = df[(df['job_position'] == 'Data Scientist') & (df['Class'] == 3)]
bottom_10_class_3_data_science = bottom_10_class_3_data_science.groupby('company').apply(lambda x: x.nsmallest(10, 'ctc')).reset_index(drop=True)
bottom_10_class_3_data_science = bottom_10_class_3_data_science.head(10)
bottom_10_class_3_data_science = bottom_10_class_3_data_science[['employee_id', 'job_position', 'ctc', 'Class']]
bottom_10_class_3_data_science
```

```
<ipython-input-226-f6b1ce9d95ff>:2: DeprecationWarning
  bottom_10_class_3_data_science = bottom_10_class_3_d
```

	employee_id	job_position	ctc	Class
0	165530	Data Scientist	1800000.0	3
1	74350	Data Scientist	850000.0	3
2	36316	Data Scientist	1220000.0	3
3	11561	Data Scientist	1680000.0	3
4	206408	Data Scientist	1100000.0	3
5	90369	Data Scientist	1150000.0	3
6	154852	Data Scientist	1150000.0	3
7	65298	Data Scientist	1160000.0	3
8	99386	Data Scientist	1310000.0	3
9	89719	Data Scientist	1400000.0	3

Insights:

- **Salary Range:** Employees earn between 850,000 and 1,800,000, with the lowest being 74350 at 850,000**.**
- **High-Earning Outliers:** Despite high salaries, they are the lowest earners compared to their peers in similar roles within the company.
- **Inconsistent Earnings:** The wide salary range indicates varying experience levels, but they still fall below the average compared to other Data Scientists.

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

```
# Filter out rows where job_position is 'Others' or 'Unknown'
df_filtered1 = df[~df['job_position'].isin(['Other', 'Unknown'])]

bottom_10_tier_3 = df_filtered1[df_filtered1['Tier'] ==
3].groupby('company').apply(lambda x: x.nsmallest(10,
'ctc')).reset_index(drop=True)
bottom_10_tier_3 = bottom_10_tier_3.head(10)
bottom_10_tier_3 = bottom_10_tier_3[['employee_id', 'job_position',
'ctc', 'Tier']]
bottom_10_tier_3
```

```
<ipython-input-229-a15e0f105c12>:4: DeprecationWarning:
  bottom_10_tier_3 = df_filtered1[df_filtered1['Tier']
```

	employee_id	job_position	ctc	Tier
0	140181	Data Scientist	1100000.0	3
1	175721	Backend Engineer	1100000.0	3
2	63841	Backend Engineer	1200000.0	3
3	38633	Backend Engineer	1300000.0	3
4	154247	FullStack Engineer	1300000.0	3
5	151872	Backend Engineer	1350000.0	3
6	176160	FullStack Engineer	1350000.0	3
7	55160	FullStack Engineer	1400000.0	3
8	175772	Backend Engineer	1400000.0	3
9	20642	FullStack Engineer	1600000.0	3

Insights :

- **Salary Range:** Employees earn between 1,100,000 and 1,600,000, with the highest salary being 1,600,000 (for FullStack Engineers).
- **Job Position Distribution:** Backend Engineers and FullStack Engineers dominate this group, with Backend Engineers having a slightly lower salary range.
- **Earnings Comparison:** Despite earning in the range of 1,100,000 to 1,600,000, these employees are still among the lower earners within their respective companies.

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

```
df_10_by_experience = df[(df['job_position'] == 'Data Scientist') &
(df['Years_of_Experience'].isin([5, 6, 7]))]
df_10_by_experience =
df_10_by_experience.groupby('company').apply(lambda x:
x.nlargest(10, 'ctc')).reset_index(drop=True)
df_10_by_experience = df_10_by_experience.head(10)
df_10_by_experience = df_10_by_experience[['employee_id',
'company', 'job_position', 'ctc', 'Years_of_Experience']]
df_10_by_experience
```

```
<ipython-input-224-164722f52b97>:2: DeprecationWarning: DataFrameGroupBy.app
  df_10_by_experience = df_10_by_experience.groupby('company').apply(lambda :
```

	employee_id	company	job_position	ctc	Years_of_Experience
0	117081	10dvxrtvqzxzs	Data Scientist	400000.0	5
1	140181	1bs	Data Scientist	1100000.0	7
2	125001	2020	Data Scientist	2100000.0	5
3	136321	247vx	Data Scientist	1100000.0	7
4	165530	3pntwyzgrgsxto	Data Scientist	1800000.0	7
5	106613	3pntwyzgrgsxto	Data Scientist	1200000.0	7
6	160549	3pntwyzgrgsxto	Data Scientist	1000000.0	6
7	144259	3rgi	Data Scientist	800000.0	7
8	195788	Unknown	Data Scientist	2000000.0	6
9	99735	aaw	Data Scientist	88555.0	7

Insights:

- **Experience Range:** All employees have 5 to 7 years of experience, with the majority at 7 years.
- **CTC Distribution:** Salaries range from 88,555 to 2,100,000, indicating significant variation based on company and position.
- **High Earners:** The highest CTC is 2,100,000 for an employee with 5 years of experience at company `2020`.
- **Key Companies:** Companies like `3pntwyzgrgsxto` have multiple top earners, reflecting a strong investment in their Data Science department.
- **Notable Variance:** The lowest salary, 88,555, is an outlier compared to others in the same tier, suggesting potential discrepancies or differing job roles.

6. Top 10 companies (based on their CTC).

```
top_10_companies_by_ctc =
df.groupby('company')['ctc'].mean().sort_values(ascending=False).he
ad(10).reset_index()
top_10_companies_by_ctc
```

	company	ctc
0	whmxwrgsxwouqxcvntrxbxnta	1.000150e+09
1	aveegaxrxzntqzvnxgzvrhzxctqoxnj	2.500000e+08
2	ehlxonh	2.000000e+08
3	vayxxuvntwyzgrgsxtoucnrna	2.000000e+08
4	axctqoxextatztqsjogrhnxgzoucnrna	2.000000e+08
5	ofvbxcxctpvzvzavxzonxnhntgeowxtzwtvzantwyzgrgsj	2.000000e+08
6	touxqxnntwyzgrgsxtoucnrna	2.000000e+08
7	nwjgzrxzt	2.000000e+08
8	mvpyntqnqvaxzs	2.000000e+08
9	qn	2.000000e+08

Insights:

- **Highest CTC:** The company `whmxwrgsxwouqxcvntrxbxnta` tops the list with a remarkable 1 billion CTC, far exceeding others.
- **Close Competition:** The remaining companies have a uniform CTC of 200 million, except for the second-ranked company, `aveegaxrxzntqzvnxgzvrhzxctqoxnj`, with 250 million CTC.
- **Industry Standouts:** These companies represent the top-tier earners, reflecting significant investments in their workforce.
- **Notable Disparity:** The highest CTC (1 billion) is 4x greater than the next competitor, highlighting a standout performer in compensation practices.

7. Top 2 positions in every company (based on their CTC)

```
df_filtered = df[~df['job_position'].isin(['Other', 'Unknown'])]
top_2_positions = df_filtered.groupby(['company',
'job_position']).apply(lambda x: x.nlargest(2,
'ctc')).reset_index(drop=True)
top_2_positions =
top_2_positions.groupby('company').head(2).reset_index(drop=True)
top_2_positions = top_2_positions[['company', 'job_position',
'ctc']]
top_2_positions
```

```
<ipython-input-216-9eda17178f1c>:5: DeprecationWarning: DataFrameG
  top_2_positions = df_filtered.groupby(['company', 'job_position'
```

	company	job_position	ctc
0	01ojztqsj	Android Engineer	270000.0
1	01ojztqsj	Frontend Engineer	830000.0
2	05mzexzytvrnyuqxcvntrxbxnta	Backend Engineer	1100000.0
3	10	Backend Engineer	450000.0
4	1000uqgltwn	Frontend Engineer	600000.0
...	...	...	...
39163	zxzvnxgzvrxzonqhbtzno	FullStack Engineer	1350000.0
39164	zxzvzxjvsqghu	Engineering Leadership	1180000.0
39165	zyuwrxbxnta	Frontend Engineer	2400000.0
39166	zyvzwtwgzohrnxzstzsxzttqo	Frontend Engineer	940000.0
39167	zzbztdnstzvacxogqjucnrna	FullStack Engineer	600000.0

39168 rows × 3 columns

Insights:

- **Diverse Roles Across Companies:** The top 2 positions in each company vary, with roles like Android Engineer, Frontend Engineer, Backend Engineer, and FullStack Engineer frequently.
- **High CTC Variations:** Significant differences in CTC are observed across companies, e.g., `zyuwrxbxnta` offers 2.4 million for a Frontend Engineer, while others, such as `01ojztqsj`, offer 830,000 for the same role.Frontend Engineer roles frequently emerge in the top 2 positions across various companies, reflecting strong demand.
- **Backend and FullStack Demand:** Backend Engineers and FullStack Engineers also feature prominently, showcasing their critical importance in tech firms.
- **Variety of Industries Represented:** Companies ranging from `01ojztqsj` to `zzbztdnstzvacxogqjucnrna` highlight widespread competition for top talent across sectors.

4. Model implementation.

Data processing for Unsupervised clustering - Label encoding/ One- hot encoding, Standardization of data

```
data = df.drop(columns = ['avg_ctc_dept','designation',
'avg_ctc_class', 'Class', 'avg_ctc_tier', 'Tier'])
data
```

```
data['ctc_log'] = np.log2(data['ctc'])
```

```
drop_cols = ['email_id','employee_id']
for i in drop_cols:
    try:
        data.drop([i],axis=1,inplace=True)
    except:
        print('no')
```

```
df_model = data.copy()
df_model
```

Code cell output actions

	company	Start_year	ctc	job_position	ctc_updated_year	Years_of_Experience	ctc_log
0	atrgxnntxzaxv	2016.0	1100000.0	Other	2020.0	9	20.069072
1	qtrxvzwtxzegwgbbrxbxnta	2018.0	449999.0	FullStack Engineer	2019.0	7	18.779562
2	ojzwnvwnxwvx	2015.0	2000000.0	Backend Engineer	2020.0	10	20.931569
3	ngpgutaxv	2017.0	700000.0	Backend Engineer	2019.0	8	19.416995
4	qxensqghu	2017.0	1400000.0	FullStack Engineer	2019.0	8	20.416995
...	...	...	...	...	...	...	...
205456	vuurtxzw	2008.0	220000.0	Unknown	2019.0	17	17.747144
205457	husqvawgb	2017.0	500000.0	Unknown	2020.0	8	18.931569
205458	vwwgrxnt	2021.0	700000.0	Unknown	2021.0	4	19.416995
205459	zgnvuurxwvmrt	2019.0	5100000.0	Unknown	2019.0	6	22.282066
205460	bgqsvzonvzrtj	2014.0	1240000.0	Unknown	2016.0	11	20.241909

205461 rows × 7 columns

```python
le = LabelEncoder()

# Encode company and job_position
df_model['company_encoded'] = le.fit_transform(data['company'])
df_model['job_position_encoded'] =
le.fit_transform(data['job_position'])
df_model = df_model.drop(['company', 'job_position'], axis=1)
```

	Start_year	ctc	ctc_updated_year	Years_of_Experience	ctc_log	company_encoded	job_position_encoded
0	2016.0	1100000.0	2020.0	9	20.069072	948	452
1	2018.0	449999.0	2019.0	7	18.779562	19307	288
2	2015.0	2000000.0	2020.0	10	20.931569	15174	138
3	2017.0	700000.0	2019.0	8	19.416995	11854	138
4	2017.0	1400000.0	2019.0	8	20.416995	19803	288
...	...	...	...	...	...	...	...
205456	2008.0	220000.0	2019.0	17	17.747144	28145	939
205457	2017.0	500000.0	2020.0	8	18.931569	8335	939
205458	2021.0	700000.0	2021.0	4	19.416995	28478	939
205459	2019.0	5100000.0	2019.0	6	22.282066	35274	939
205460	2014.0	1240000.0	2016.0	11	20.241909	2113	939

205461 rows × 7 columns

```python
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df_model[['Start_year',
'ctc', 'Years_of_Experience', 'ctc_log', 'ctc_updated_year']])

# Combine scaled features with encoded columns
processed_data = pd.DataFrame(scaled_features,
columns=['Start_year', 'ctc', 'Years_of_Experience', 'ctc_log',
'ctc_updated_year'])
processed_data['company_encoded'] = df_model['company_encoded']
processed_data['job_position_encoded'] =
df_model['job_position_encoded']
```

```
processed_data
```

	Start_year	ctc	Years_of_Experience	ctc_log	ctc_updated_year	company_encoded	job_position_encoded
0	0.211792	-0.098704	-0.211832	0.156579	0.281656	948	452
1	0.685111	-0.154160	-0.685151	-0.691854	-0.472813	19307	288
2	-0.024867	-0.021919	0.024827	0.724059	0.281656	15174	138
3	0.448452	-0.132831	-0.448492	-0.272455	-0.472813	11854	138
4	0.448452	-0.073109	-0.448492	0.385496	-0.472813	19803	288
...	...	...	...	...	...	...	...
205456	-1.681483	-0.173783	1.681442	-1.371135	-0.472813	28145	939
205457	0.448452	-0.149894	-0.448492	-0.591842	0.281656	8335	939
205458	1.395090	-0.132831	-1.395129	-0.272455	1.036124	28478	939
205459	0.921771	0.242564	-0.921811	1.612620	-0.472813	35274	939
205460	-0.261527	-0.086760	0.261486	0.270297	-2.736219	2113	939

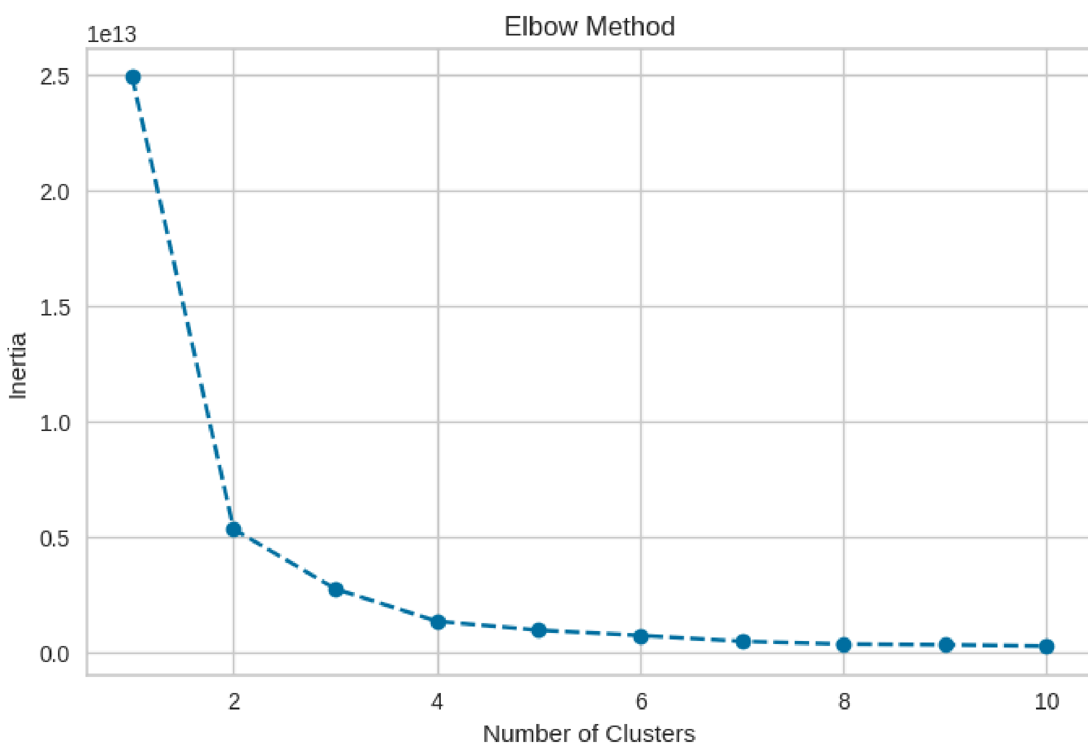
205461 rows × 7 columns

```
def hopkins(X):
    d = X.shape[1]
    n = len(X) // 10
    nbrs = NearestNeighbors(n_neighbors=1).fit(X)
    rand_X = uniform(X.min(axis=0), X.max(axis=0), (n, d))
    u_distances, _ = nbrs.kneighbors(rand_X)
    x_distances, _ = nbrs.kneighbors(sample(list(X), n))
    u_sum = sum(u_distances)
    x_sum = sum(x_distances)
    return u_sum / (u_sum + x_sum)
print(f"Hopkins statistic: {hopkins(processed_data.values)}")
```

```
Hopkins statistic: [1.]
```

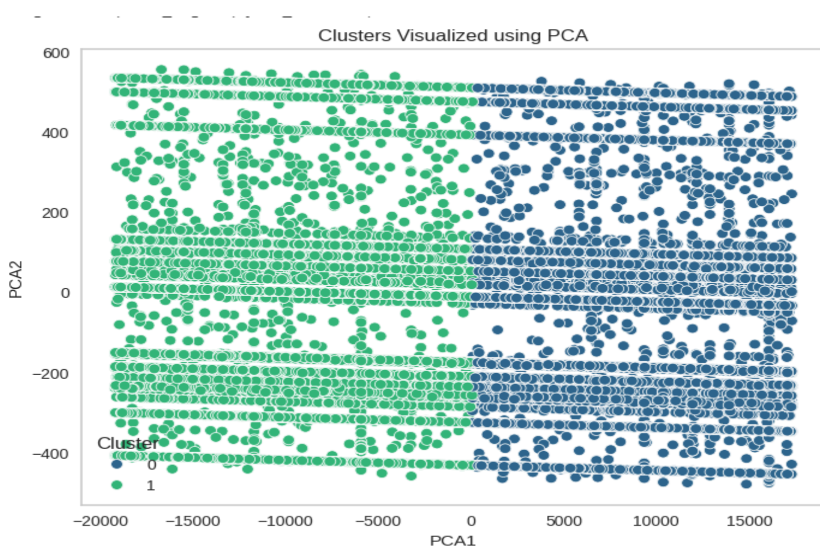
A Hopkins statistic of `1.0` indicates that data has perfect clustering tendency, meaning the data points are highly structured and form natural clusters. This is an excellent starting point for clustering since it suggests the presence of meaningful groupings in the dataset.

```python
# Elbow method
inertia = []
range_clusters = range(1, 11)
for k in range_clusters:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(processed_data)
    inertia.append(kmeans.inertia_)
# Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(range_clusters, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```

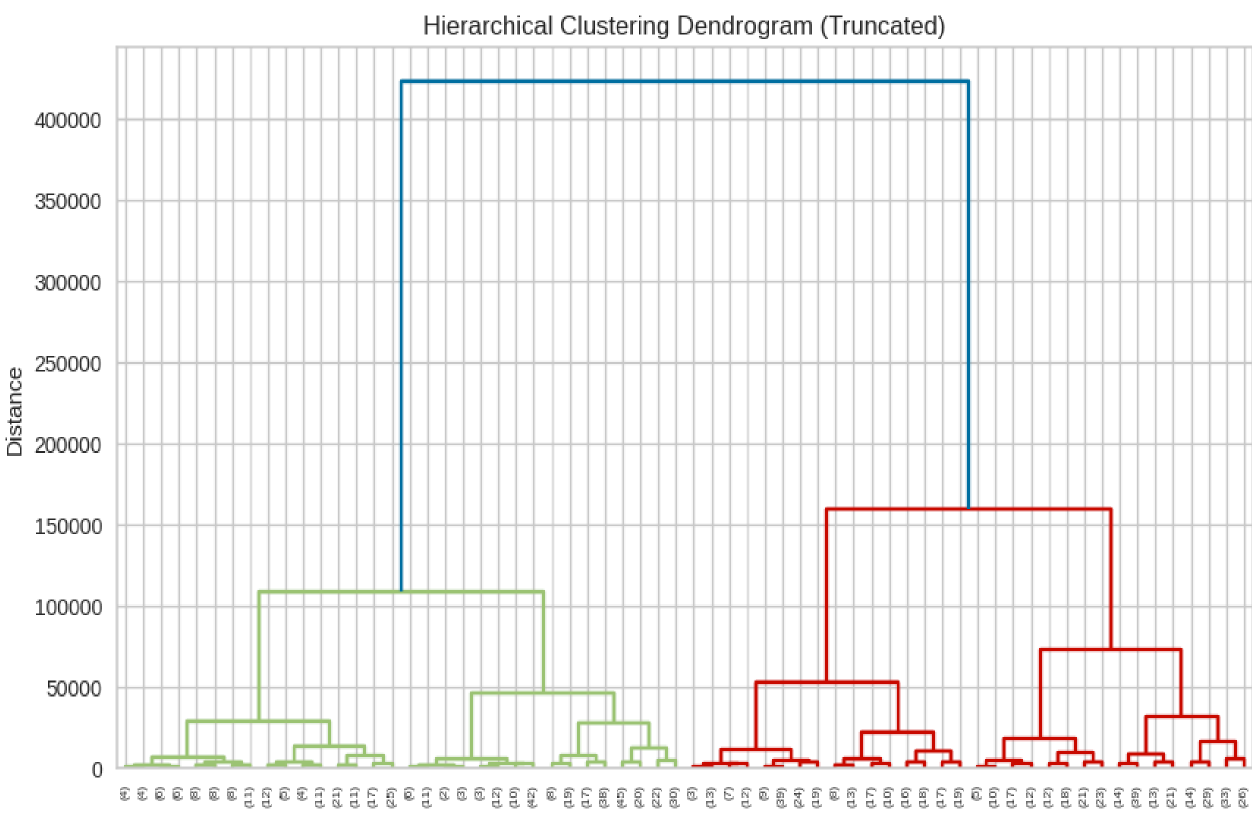


The elbow method converges when the number of clusters are 2.

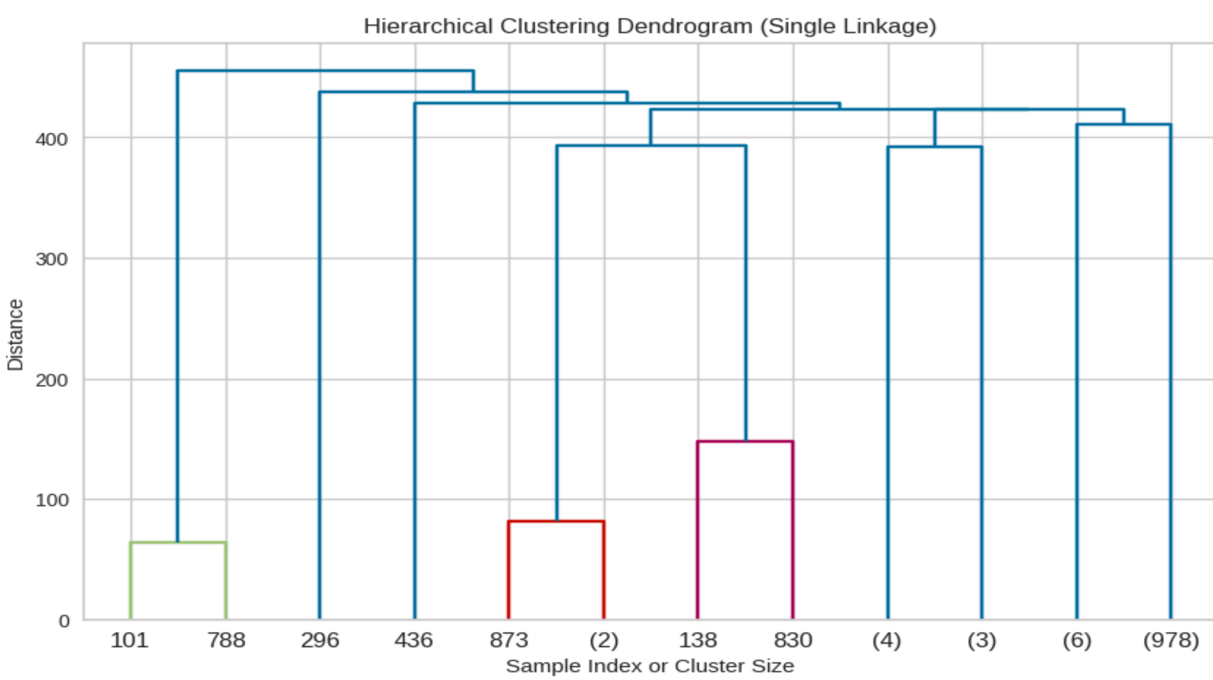
```
# K-means Clustering
optimal_k = 2
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(processed_data)
# Add cluster labels to the original dataframe
df['Cluster'] = clusters
# Visualizing cluster centers (2D plot using PCA)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(processed_data)
df['PCA1'] = pca_data[:, 0]
df['PCA2'] = pca_data[:, 1]
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PCA1', y='PCA2', hue='Cluster',
palette='viridis', s=50)
plt.title('Clusters Visualized using PCA')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.legend(title='Cluster')
plt.grid()
plt.show()
```

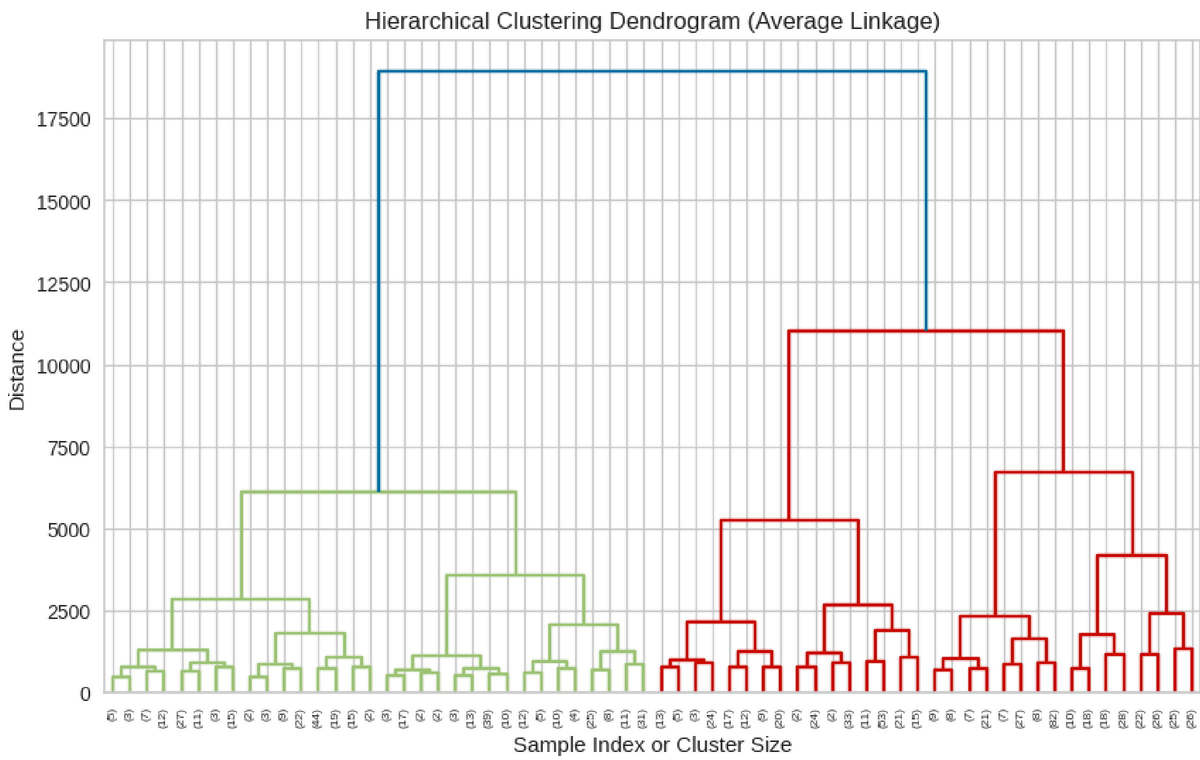


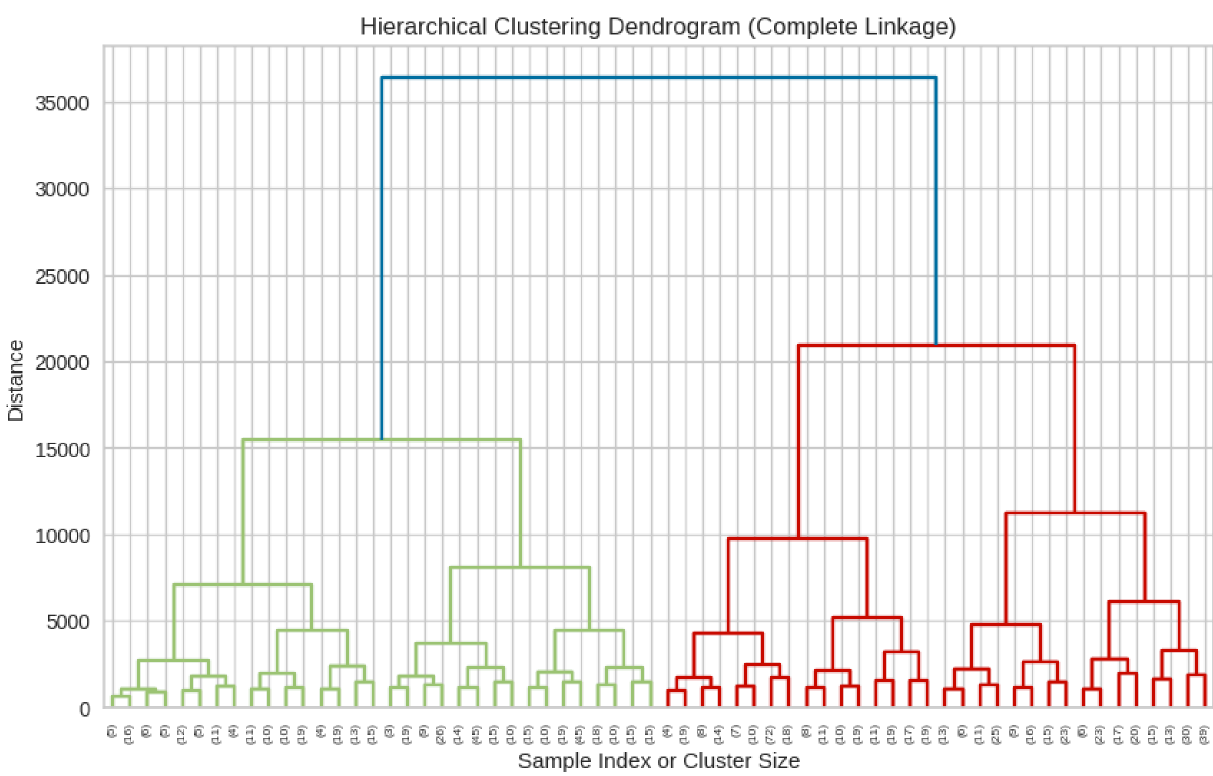
```python
# Hierarchical Clustering
sample_data = processed_data[:1000]  # Use a subset for performance
# Generate the linkage matrix
linkage_matrix = linkage(sample_data, method='ward')
# Plot the dendrogram
plt.figure(figsize=(10, 6))
dendrogram(linkage_matrix, truncate_mode='level', p=5)
plt.title('Hierarchical Clustering Dendrogram (Truncated)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
# Perform Agglomerative Clustering
agg_clustering = AgglomerativeClustering(n_clusters=optimal_k,
linkage='ward')
agg_clusters = agg_clustering.fit_predict(sample_data)
```



```
# Sample the data (use a smaller sample for visualization if needed)
sample_data = processed_data[:1000]  # Use the first 1000 rows for
simplicity
# Function to plot dendrograms
def plot_dendrogram(linkage_matrix, method):
    plt.figure(figsize=(10, 6))
    dendrogram(linkage_matrix, truncate_mode='level', p=5)  # Truncate for
better visualization
    plt.title(f'Hierarchical Clustering Dendrogram ({method} Linkage)')
    plt.xlabel('Sample Index or Cluster Size')
    plt.ylabel('Distance')
    plt.show()
# Single Linkage
single_linkage = linkage(sample_data, method='single')
plot_dendrogram(single_linkage, 'Single')
# Average Linkage
average_linkage = linkage(sample_data, method='average')
plot_dendrogram(average_linkage, 'Average')
# Complete Linkage
complete_linkage = linkage(sample_data, method='complete')
plot_dendrogram(complete_linkage, 'Complete')
```




Hierarchical Clustering Dendrogram (Average Linkage)
Distance
Sample Index or Cluster Size


Hierarchical Clustering Dendrogram (Complete Linkage)
Distance
Sample Index or Cluster Size

5. Insights & Recommendations:

1. **Distinct Employee Clusters Based on Compensation**
    - Employees were segmented into clusters based on their CTC. The clustering identified top earners (Tier 1), average earners, and bottom earners (Tier 3) in each job role and company. This helps reveal the salary disparity and align compensation strategies across departments and companies.
2. **Job Positions with High and Low CTC**
    - Certain job positions, like Backend Engineers, Data Scientists, and FullStack Engineers, consistently appeared in the top-tier earning clusters, whereas other roles, such as Android Engineers, had comparatively lower representation in the high-earning clusters.
3. **Experience Level Insights**
    - Employees with 5–7 years of experience in certain companies, especially in the Data Science department, are among the highest earners in their clusters. This shows a strong correlation between experience level and compensation.
4. **Company-Wise Salary Trends**
    - Specific companies like *whmxwrgsxwouqxcvntrxbxnta* and *aveegaxrxzntqzvnxgzvrhzxctqoxnj* are paying significantly higher salaries overall compared to others. These companies are potential leaders in talent acquisition and retention through competitive compensation packages.
5. **Disparity in Salary Across Companies**
    - Significant CTC gaps were observed across companies, even within similar job roles. For example, the CTC for Data Scientists ranged widely between companies, highlighting potential inconsistencies in market alignment.
6. **Cluster Identification for Low-Earning Employees**
    - Bottom-tier employees are distributed across various roles and companies. These clusters can help identify employees who may require upskilling, reskilling, or better alignment with organizational goals.
7. **Top Roles and Companies**
    - Specific roles like Engineering Leadership and Product Design often rank among the highest-paid positions across companies. These roles may be critical for driving innovation and strategy.

Recommendation:

1. **Review and Standardize Compensation Packages**
   - Address salary disparities across companies for similar roles by aligning compensation with industry standards. This will ensure fairness and help retain talent.
2. **Focus on Retaining Top Earners**
   - For Tier 1 employees, implement retention strategies like performance-based bonuses, stock options, or career development opportunities. These employees are likely high contributors to organizational success.
3. **Upskill and Reskill Low Earners**
   - Identify employees in Tier 3 clusters and provide targeted training programs to improve their productivity and align them with higher-value roles. This will help move them to higher salary clusters.
4. **Leverage Insights for Hiring Strategies**
   - Use clustering insights to identify companies or roles that pay competitively and benchmark them for recruitment strategies. Offer market-aligned salaries to attract top talent.
5. **Reward Experience Strategically**
   - Employees with 5–7 years of experience, especially in technical roles like Data Science, should be prioritized for promotions, leadership training, and high-value projects to maximize their contribution.
6. **Tailored Employee Benefits by Cluster**
   - Design benefits packages based on clusters. For instance, Tier 1 employees might value equity plans, while Tier 3 employees could benefit more from subsidized skill development programs.
7. **Encourage Transparency in Salary Bands**
   - Publish salary bands for each role and department within the organization to improve transparency and ensure employees are aware of growth opportunities.
8. **Invest in High-Paying Departments**
   - Departments with top-tier salaries (e.g., Engineering Leadership, Product Design) may require sustained investment to maintain competitive advantages. Consider prioritizing innovation and support in these areas.

By addressing these areas, the organization can strengthen its talent strategy, improve employee satisfaction, and enhance competitiveness in the market.