In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

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.

1. Unnamed 0: The index of the dataset.
2. Email_hash: An anonymized identifier representing the email of the learner.
3. Company_hash: An anonymized identifier indicating the current employer of the learner.
4. orgyear: Represents the year the learner began employment at the current company.
5. CTC: Current Compensation to the Company (CTC) of the learner.
6. Job_position: Represents the job profile or role of the learner within their company.
7. CTC_updated_year: The year in which the learner's CTC was most recently updated. This could be due to yearly increments, promotions, or other factors.

In [2]:
```
df = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/8
df
```

Out[2]:

	Unnamed: 0	company_hash	email_hash	orgyear	
0	0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100
1	1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449
2	2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000
3	3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700
4	4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400
...	...	...	...	...	
205838	206918	vuurt xzw	70027b728c8ee901fe979533ed94ffda97be08fc23f33b...	2008.0	220
205839	206919	husqvawgb	7f7292ffad724ebbe9ca860f515245368d714c84705b42...	2017.0	500
205840	206920	vwwgrxnt	cb25cc7304e9a24facda7f5567c7922ffc48e3d5d6018c...	2021.0	700
205841	206921	zgn vuurxwvmrt	fb46a1a2752f5f652ce634f6178d0578ef6995ee59f6c8...	2019.0	5100
205842	206922	bgqsvz onvzrtj	0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f...	2014.0	1240

205843 rows × 7 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 7 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        205843 non-null  int64  
 1   company_hash      205799 non-null  object 
 2   email_hash        205843 non-null  object 
 3   orgyear           205757 non-null  float64
 4   ctc               205843 non-null  int64  
 5   job_position      153281 non-null  object 
 6   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(2), object(3)
memory usage: 11.0+ MB
```

In [4]:
```
df.dtypes
```

Out[4]:
```
Unnamed: 0            int64
company_hash         object
email_hash           object
orgyear             float64
ctc                   int64
job_position         object
ctc_updated_year    float64
dtype: object
```

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

Out[5]:

	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

In [6]:
```
df.shape
```

Out[6]:
```
(205843, 7)
```

Rename 1 column

In [7]:
```
df.rename(columns = {"Unnamed: 0": "id"}, inplace=True)
```

In [8]:
```
df.dtypes
```

Out[8]:
```
id                    int64
company_hash         object
email_hash           object
orgyear             float64
ctc                   int64
job_position         object
ctc_updated_year    float64
dtype: object
```

checking for duplicated rows

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

checking for null values

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
id                      0
company_hash           44
email_hash              0
orgyear                86
ctc                     0
job_position        52562
ctc_updated_year        0
dtype: int64
```

Handling Missing Values

In [11]:
```
# drop rows with missing values
df.dropna(subset=["company_hash"], inplace=True)
```

In [12]:
```
# now all 44 rowss with missing vavalues has been deleted
df.shape
```

Out[12]:
```
(205799, 7)
```

In [13]:
```
df['orgyear'].value_counts()
```

Out[13]:
```
2018.0    25247
2019.0    23420
2017.0    23234
2016.0    23042
2015.0    20606
          ...  
2107.0        1
1972.0        1
2101.0        1
208.0         1
200.0         1
Name: orgyear, Length: 77, dtype: int64
```

In [14]:
```
df['ctc_updated_year'].value_counts()
```

Out[14]:
```
2019.0    68676
2021.0    64961
2020.0    49436
2017.0     7559
2018.0     6742
2016.0     5498
2015.0     2927
Name: ctc_updated_year, dtype: int64
```

In [15]:
```
from sklearn.impute import KNNImputer

# Specify the columns you want to impute using k-NN
columns_to_impute = ['orgyear']

# Initialize KNNImputer with k=5 (you can adjust the value of k as needed)
imputer = KNNImputer(n_neighbors=5)

# Fit and transform the data with KNN imputer
df[columns_to_impute] = imputer.fit_transform(df[columns_to_impute])
```

```
# Display summary of missing values after imputation
print("\nMissing value counts after imputation:\n", df.isnull().sum())
```

```
Missing value counts after imputation:
 id                      0
company_hash          0
email_hash            0
orgyear               0
ctc                   0
job_position      52531
ctc_updated_year      0
dtype: int64
```

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
id                    0
company_hash          0
email_hash            0
orgyear               0
ctc                   0
job_position      52531
ctc_updated_year      0
dtype: int64
```

In [17]:
```
# Fill missing values in job_position column with 'Not Specified'
df['job_position'].fillna('Not Specified', inplace=True)
```

In [18]:
```
df.isnull().sum()
```

Out[18]:
```
id                  0
company_hash        0
email_hash          0
orgyear             0
ctc                 0
job_position        0
ctc_updated_year    0
dtype: int64
```

In [19]:
```
df.dtypes
```

Out[19]:
```
id                    int64
company_hash         object
email_hash           object
orgyear             float64
ctc                   int64
job_position         object
ctc_updated_year    float64
dtype: object
```

Convert Columns to Integer

In [20]:
```
# Convert orgyear to integer type
df['orgyear'] = df['orgyear'].astype(int)
```

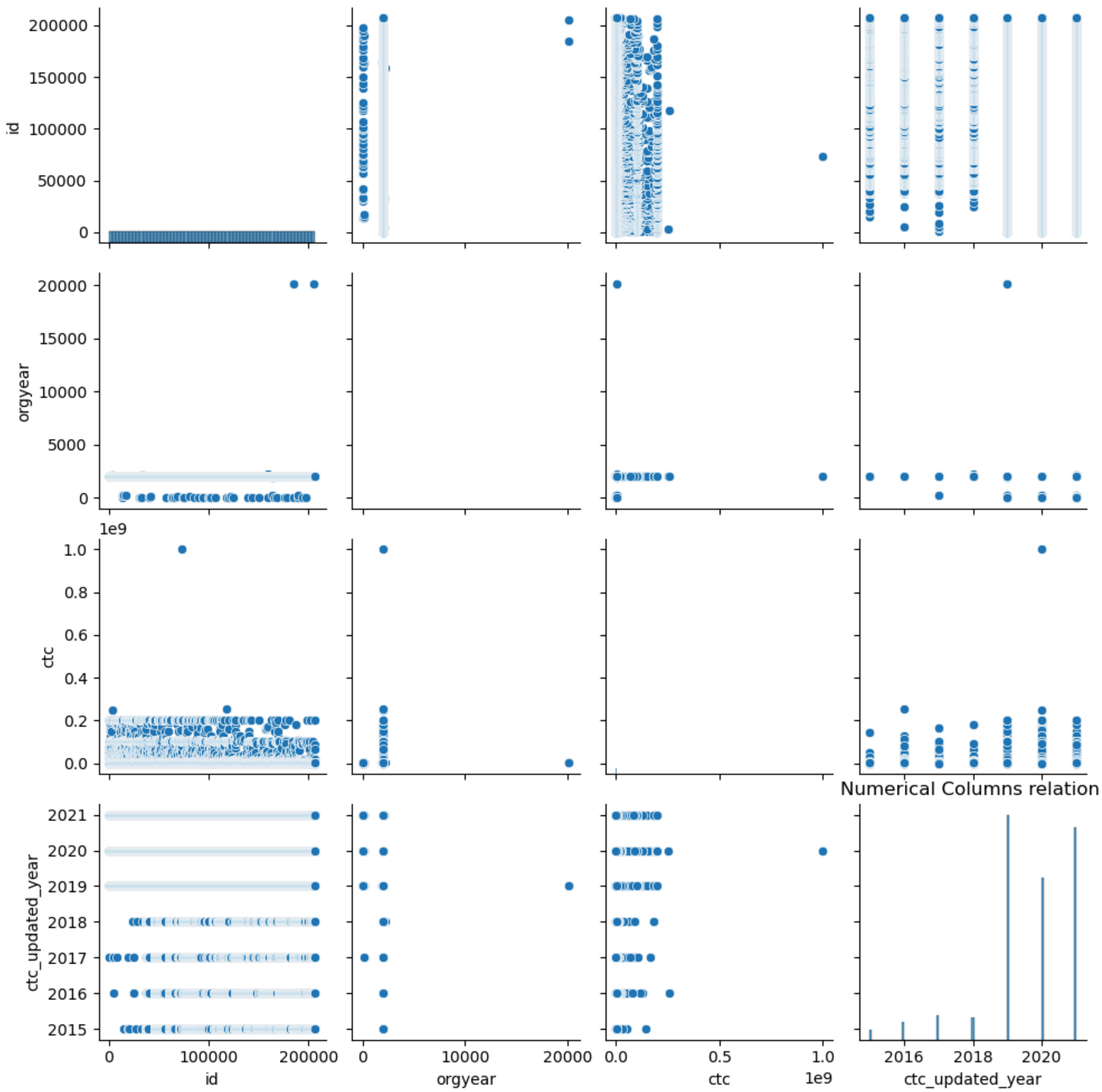
In [21]:
```
# Convert orgyear to integer type
df['ctc_updated_year'] = df['ctc_updated_year'].astype(int)
```

In [22]:
```
df.dtypes
```

Out[22]:
```
id                    int64
company_hash         object
email_hash           object
orgyear               int32
ctc                   int64
job_position         object
ctc_updated_year      int32
dtype: object
```

Visualize all Numerical Columns

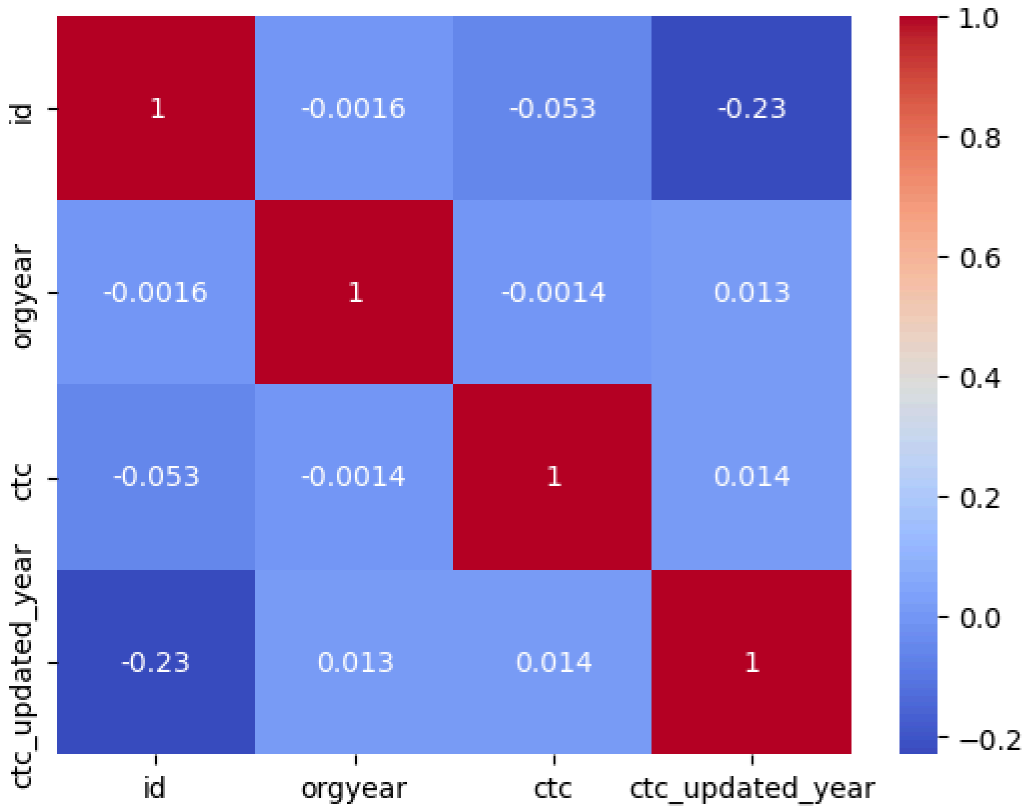
In [23]:
```
sns.pairplot(data=df )
plt.title("Numerical Columns relation ")
plt.show()
```



CTC values are very small.

```
In [24]: correlation_matrix = df.corr()
         sns.heatmap(correlation_matrix, annot= True, cmap='coolwarm')
         plt.show()
```

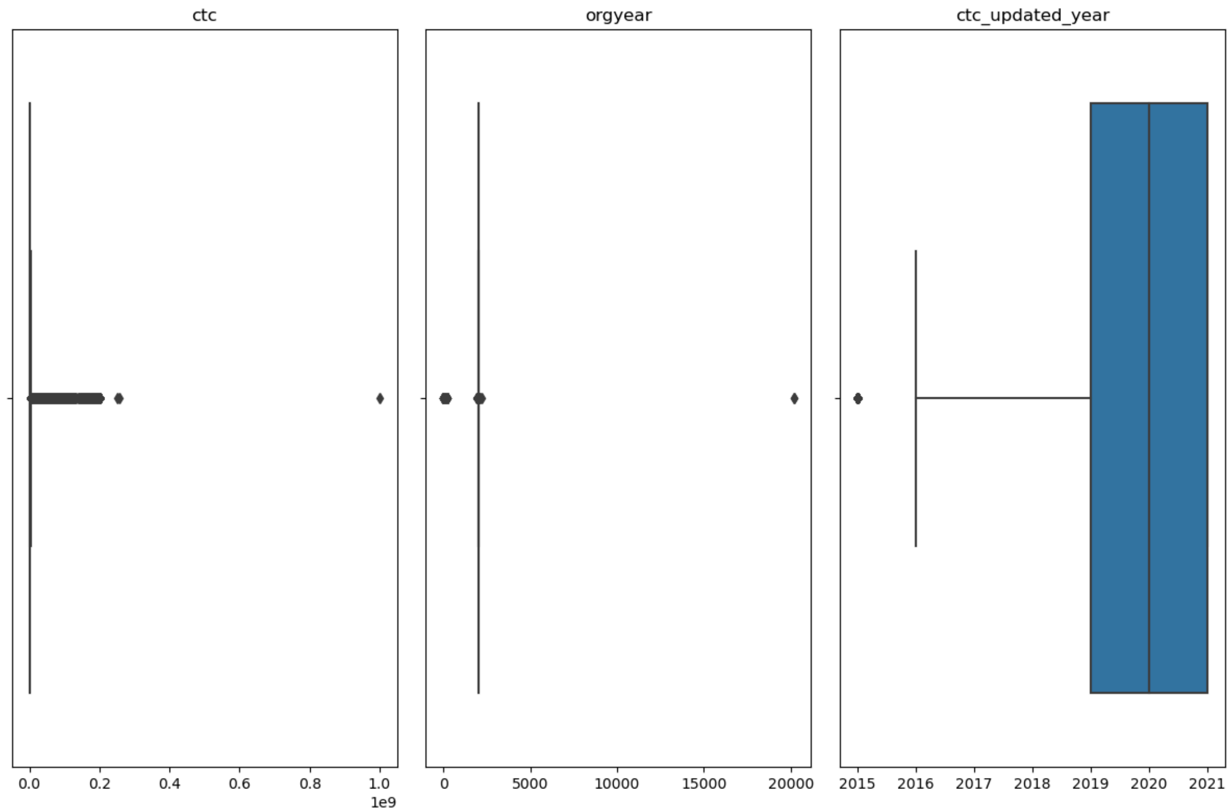
```
C:\Users\harsh\AppData\Local\Temp\ipykernel_2416\2630758536.py:1: FutureWarning: The
default value of numeric_only in DataFrame.corr is deprecated. In a future version, i
t will default to False. Select only valid columns or specify the value of numeric_on
ly to silence this warning.
  correlation_matrix = df.corr()
```



There is no correlation between all numerical columns

Outlier Detection

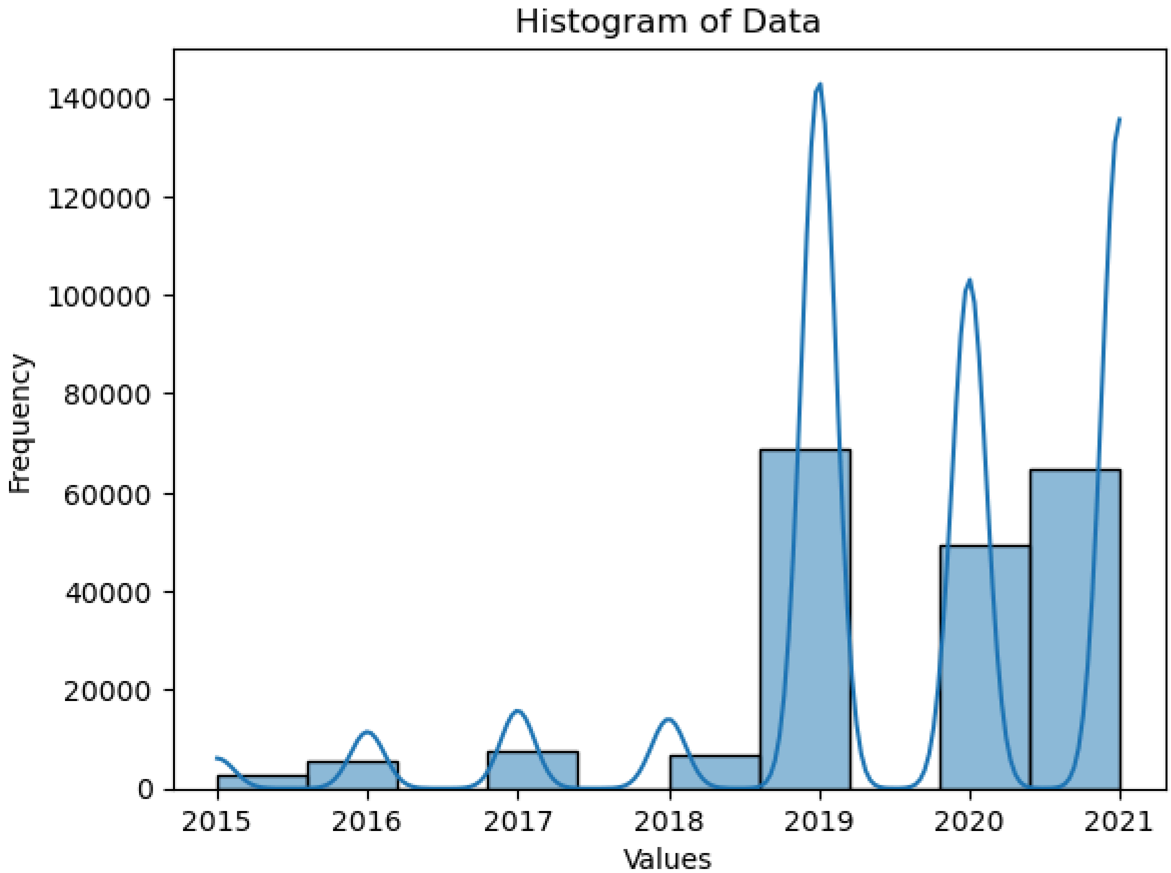
```
In [25]: # Select columns for box plot
         columns_for_boxplot = [
            'ctc', 'orgyear', 'ctc_updated_year']

         # Draw box plots for selected columns
         plt.figure(figsize=(12, 8))
         for i, col in enumerate(columns_for_boxplot, start=1):
             plt.subplot(1, 3, i)
             sns.boxplot(x=df[col])
             plt.title(col)
             plt.xlabel('')
         plt.tight_layout()
         plt.show()
```



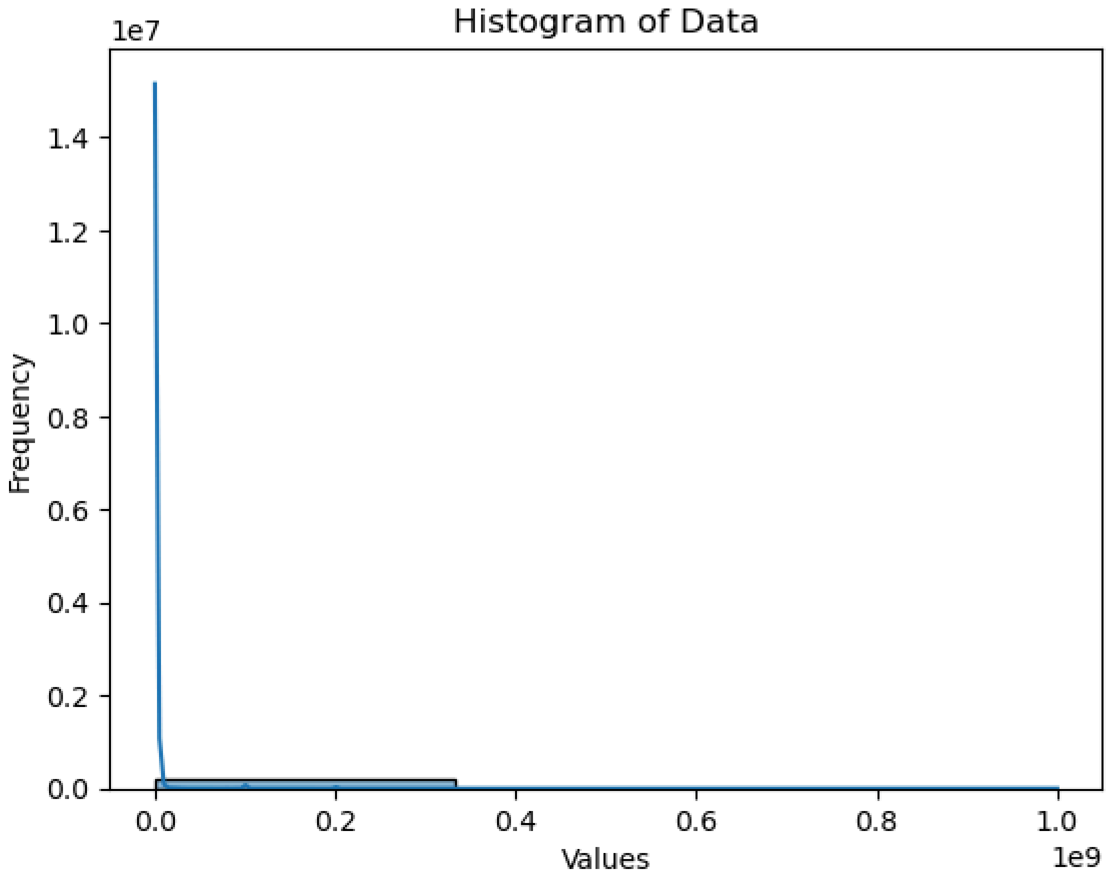
CTC is the only columns where we see most outliers

In [26]:

```
# Plot histogram of the data

sns.histplot(df, x='ctc_updated_year', bins=10, kde=True)
plt.title('Histogram of Data')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

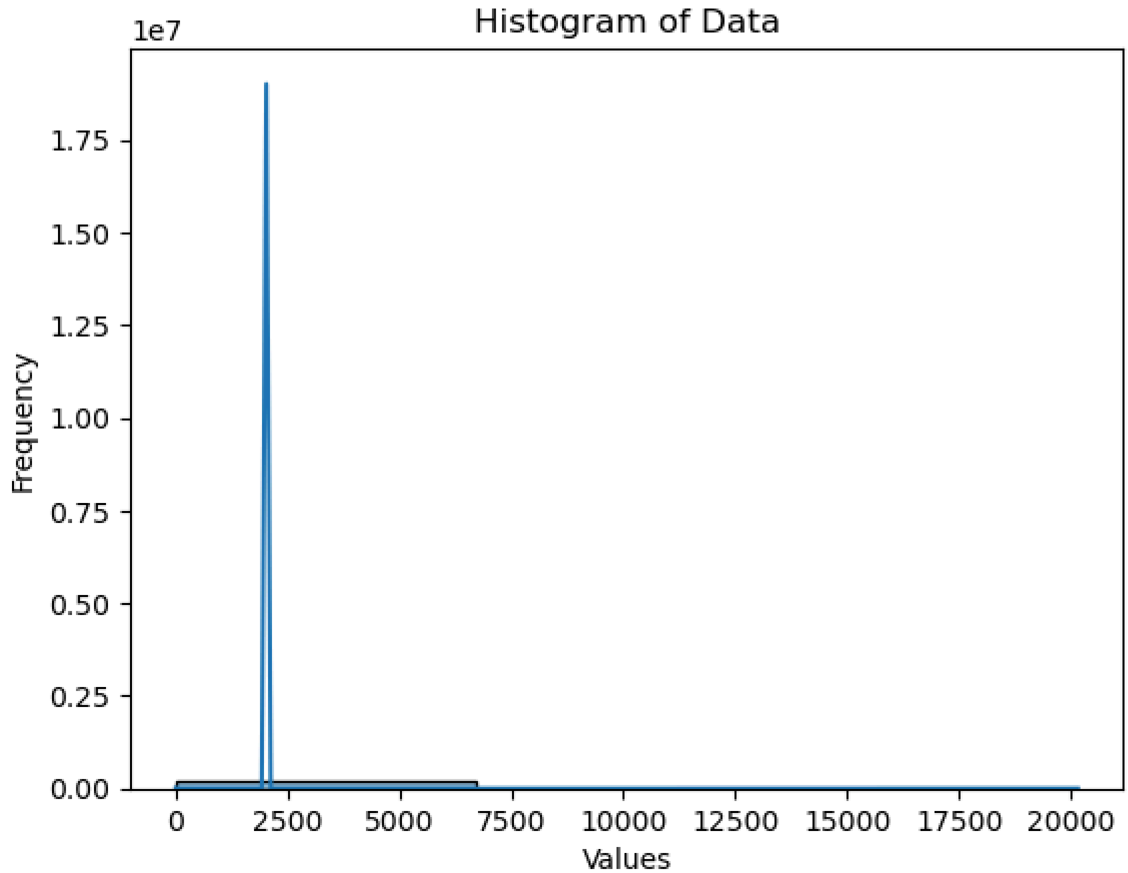


In [27]:
```
# Plot histogram of the data

sns.histplot(df, x='ctc', bins=3, kde=True)
plt.title('Histogram of Data')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```



In [28]:

```
# Plot histogram of the data

sns.histplot(df, x='orgyear', bins=3, kde=True)
plt.title('Histogram of Data')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```



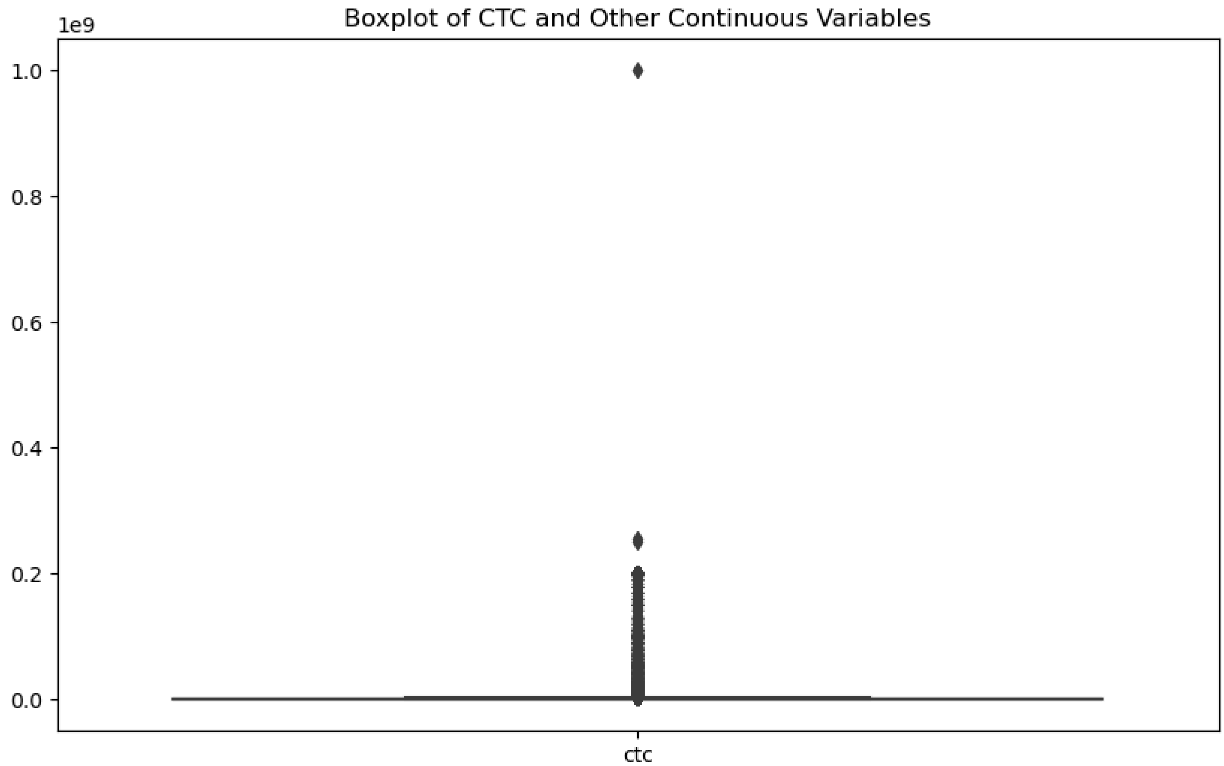
Outlier Treatment for CTC using Log Transformation

In [ ]:

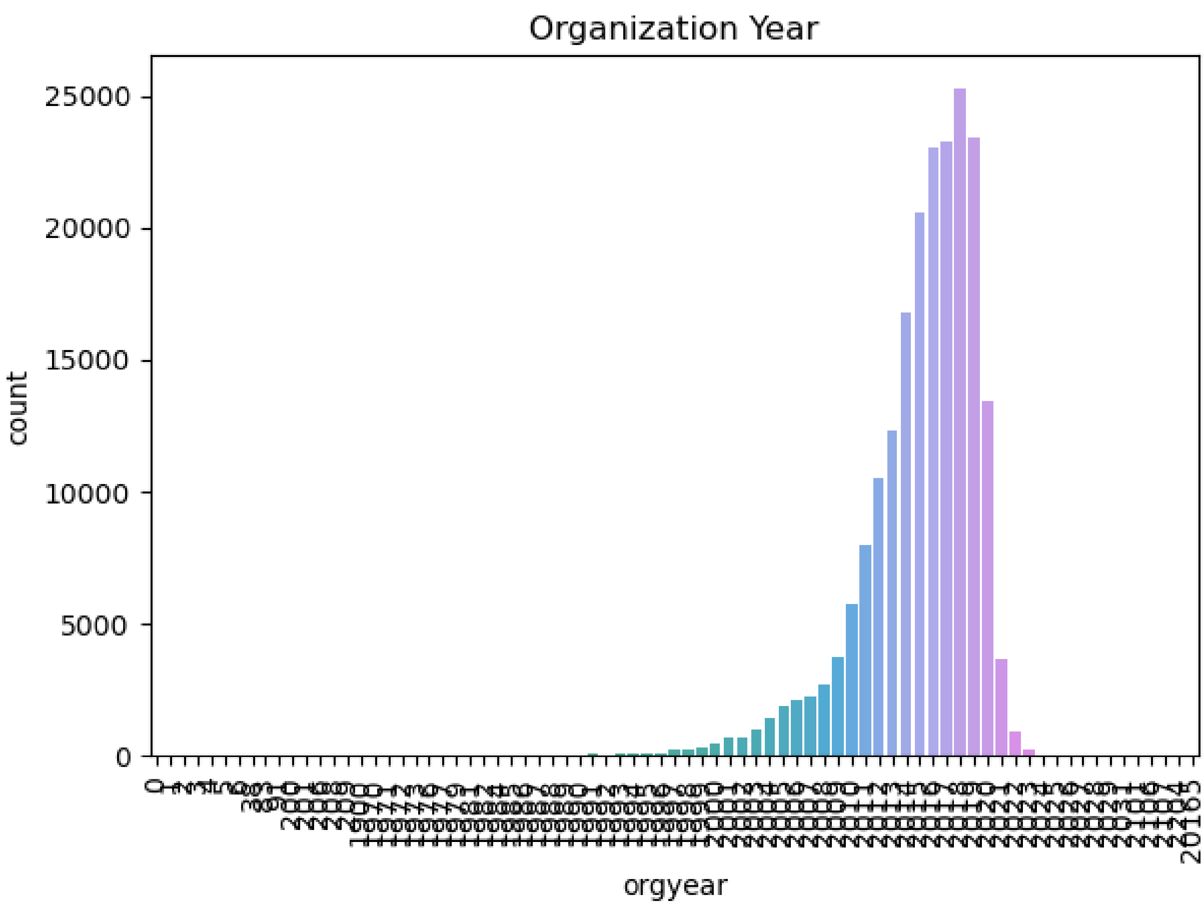
```
```

In [29]:

```
# Plot boxplots to visualize distribution and identify outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[['ctc']])  # Adjust columns as needed
plt.title('Boxplot of CTC and Other Continuous Variables')
plt.show()
```

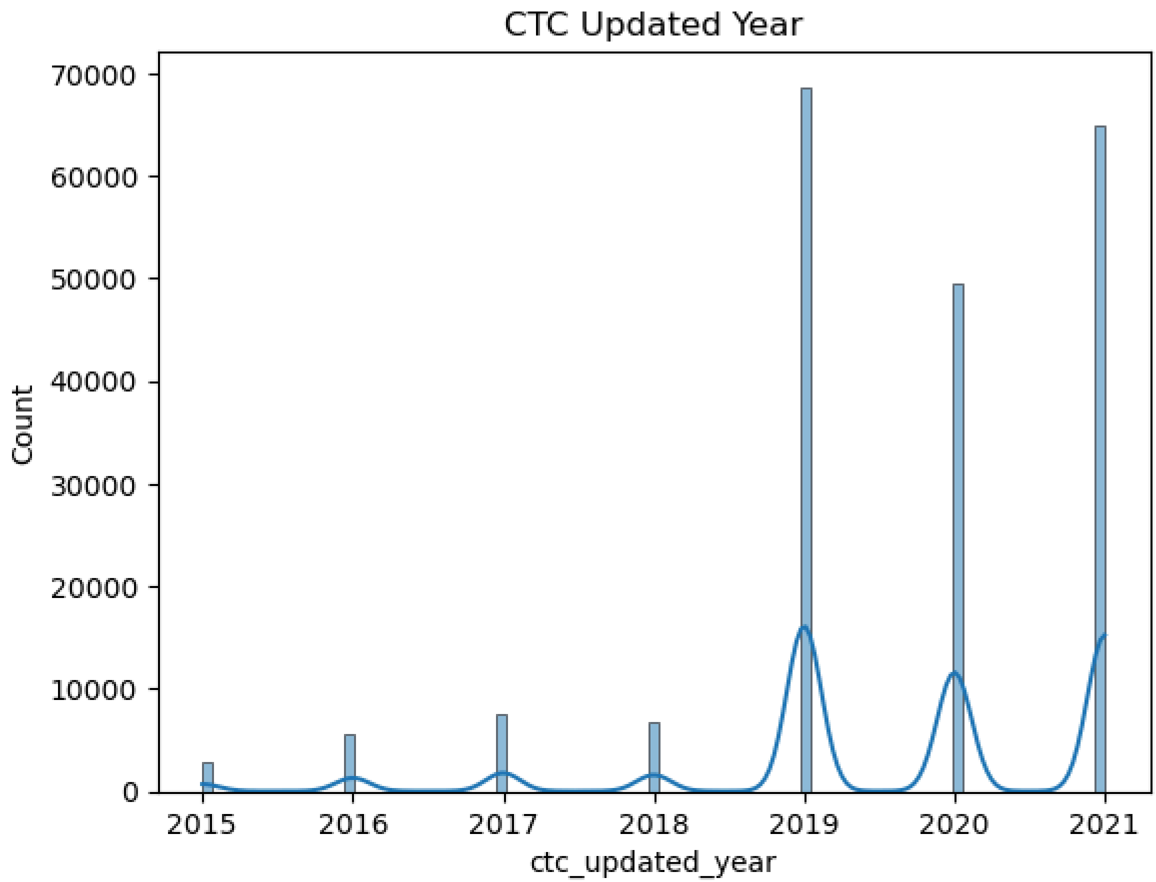


In [30]:
```python
sns.countplot(data= df, x="orgyear")
plt.title("Organization Year")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



In [31]:

```
sns.histplot(data= df, x="ctc_updated_year", kde=True)
plt.title("CTC Updated Year")
plt.show()
```

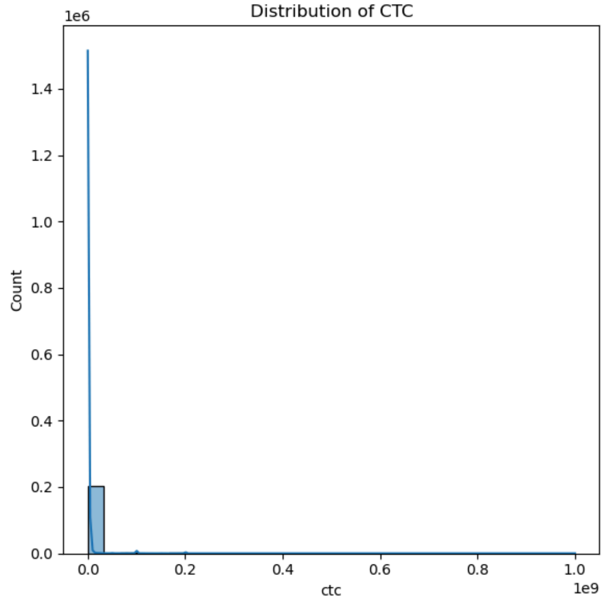


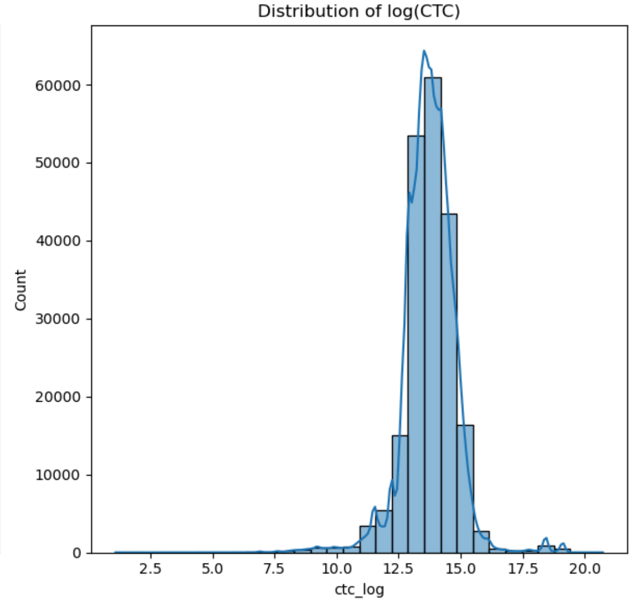
In [32]:

```
# Log transformation of CTC
df['ctc_log'] = np.log1p(df['ctc'])
```

In [33]:

```
# Visualize the distribution of the original 'ctc' and the transformed 'ctc_log'
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(df['ctc'], bins=30, kde=True)
plt.title('Distribution of CTC')

plt.subplot(1, 2, 2)
sns.histplot(df['ctc_log'], bins=30, kde=True)
plt.title('Distribution of log(CTC)')
plt.tight_layout()
plt.show()
```





Feature Engineering

In [34]:
```python
import datetime

# Assuming 'orgyear' is the year of organization establishment
current_year = datetime.datetime.now().year
df['years_of_experience'] = current_year - df['orgyear']
```

In [35]:
```python
df['years_of_experience'].value_counts()
```

Out[35]:
```
 6       25247
 5       23420
 7       23234
 8       23042
 9       20606
         ...
-83          1
 52          1
-77          1
 1816        1
 1824        1
Name: years_of_experience, Length: 77, dtype: int64
```

In [36]:
```python
 df['job_position'].value_counts()
```

Out[36]:
```
Not Specified                      52531
Backend Engineer                   43553
FullStack Engineer                 24714
Other                              18067
Frontend Engineer                  10417
                                   ...
ayS                                    1
Principal Product Engineer             1
Senior Director of Engineering         1
Seller Support Associate               1
Android Application developer          1
Name: job_position, Length: 1018, dtype: int64
```

In [37]:
```
# Example: Create flags for prominent roles
df['is_engineer'] = df['job_position'].apply(lambda x: 1 if 'Engineer' in x else 0)
df['is_manager'] = df['job_position'].apply(lambda x: 1 if 'Manager' in x else 0)
# Add more flags based on specific job titles or categories
```

In [38]:
```
df['is_engineer']
```

Out[38]:
```
0         0
1         1
2         1
3         1
4         1
         ..
205838    0
205839    0
205840    0
205841    0
205842    0
Name: is_engineer, Length: 205799, dtype: int64
```

In [39]:
```
df['got_increment'] = (df['ctc_updated_year'] > (df['ctc_updated_year'].shift(1))).ast
```

In [40]:
```
# Define bins and labels
bins = [0, 500000, 1000000, 1500000, float('inf')]
labels = ['Low', 'Average', 'High', 'Very High']

# Create 'salary_category' column
df['salary_category'] = pd.cut(df['ctc'], bins=bins, labels=labels, right=False)
```

In [41]:
```
# Example: Calculate average salary by job position
average_salary_by_position = df.groupby('job_position')['ctc'].mean().reset_index()
average_salary_by_position.rename(columns={'ctc': 'avg_salary_by_position'}, inplace=T
df = pd.merge(df, average_salary_by_position, on='job_position', how='left')
```

In [73]:
```
df['salary_category'].value_counts()
```

Out[73]:
```
3    63692
1    62068
0    42922
2    37117
Name: salary_category, dtype: int64
```

In [76]:
```
df["avg_salary_by_position"].value_counts()
```

Out[76]:
```
0.019913    52531
0.019695    43553
0.018697    24714
0.039723    18067
0.018358    10417
            ...
0.032981        1
0.005380        1
0.013580        1
0.048481        1
0.006680        1
Name: avg_salary_by_position, Length: 401, dtype: int64
```

In [42]:
```
df.head()
```

Out[42]:

	id	company_hash	email_hash	orgyear	ctc	job_posit
0	0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016	1100000	Ot
1	1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018	449999	FullSt Engir
2	2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015	2000000	Back Engir
3	3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	700000	Back Engir
4	4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017	1400000	FullSt Engir

In [43]:
```
df.tail()
```

Out[43]:

	id	company_hash	email_hash	orgyear	ctc
205794	206918	vuurt xzw	70027b728c8ee901fe979533ed94ffda97be08fc23f33b...	2008	220000
205795	206919	husqvawgb	7f7292ffad724ebbe9ca860f515245368d714c84705b42...	2017	500000
205796	206920	vwwgrxnt	cb25cc7304e9a24facda7f5567c7922ffc48e3d5d6018c...	2021	700000
205797	206921	zgn vuurxwvmrt	fb46a1a2752f5f652ce634f6178d0578ef6995ee59f6c8...	2019	5100000
205798	206922	bgqsvz onvzrtj	0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f...	2014	1240000

In [44]:
```
df.dtypes
```

Out[44]:
```
id                          int64
company_hash               object
email_hash                 object
orgyear                     int32
ctc                         int64
job_position               object
ctc_updated_year            int32
ctc_log                   float64
years_of_experience         int32
is_engineer                 int64
is_manager                  int64
got_increment               int32
salary_category          category
avg_salary_by_position    float64
dtype: object
```

Scaling Numerical Columns

In [45]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numeric_columns = ['ctc', 'ctc_updated_year', 'ctc_log', 'years_of_experience', 'avg_s
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
```

In [46]:
```
df[numeric_columns]
```

Out[46]:

	ctc	ctc_updated_year	ctc_log	years_of_experience	avg_salary_by_position
0	0.001100	0.833333	0.652858	0.900025	0.039723
1	0.000450	0.666667	0.607313	0.899926	0.018697
2	0.002000	0.833333	0.683321	0.900074	0.019695
3	0.000700	0.666667	0.629827	0.899975	0.019695
4	0.001400	0.666667	0.665147	0.899975	0.018697
...	...	...	...	...	...
205794	0.000220	0.666667	0.570848	0.900422	0.019913
205795	0.000500	0.833333	0.612681	0.899975	0.019913
205796	0.000700	1.000000	0.629827	0.899777	0.019913
205797	0.005099	0.666667	0.731021	0.899876	0.019913
205798	0.001240	0.166667	0.658963	0.900124	0.019913

205799 rows × 5 columns

Categorical Encoding

Label Encoding: Convert ordinal categorical variables (salary_category) into numerical format if they have a natural order. One-Hot Encoding: Convert nominal categorical variables (job_position) into binary columns using one-hot encoding.

In [47]:
```
# Label Encoding for ordinal categorical variable (if applicable)
df['salary_category'] = df['salary_category'].cat.codes

# One-Hot Encoding for nominal categorical variable
df = pd.get_dummies(df, columns=['job_position'], prefix='job')
```

In [48]:
```
df.dtypes
```

Out[48]:
```
id                                 int64
company_hash                      object
email_hash                        object
orgyear                            int32
ctc                              float64
                                  ...
job_student                        uint8
job_support escalation engineer    uint8
job_system engineer                uint8
job_system software engineer       uint8
job_technology analyst             uint8
Length: 1031, dtype: object
```

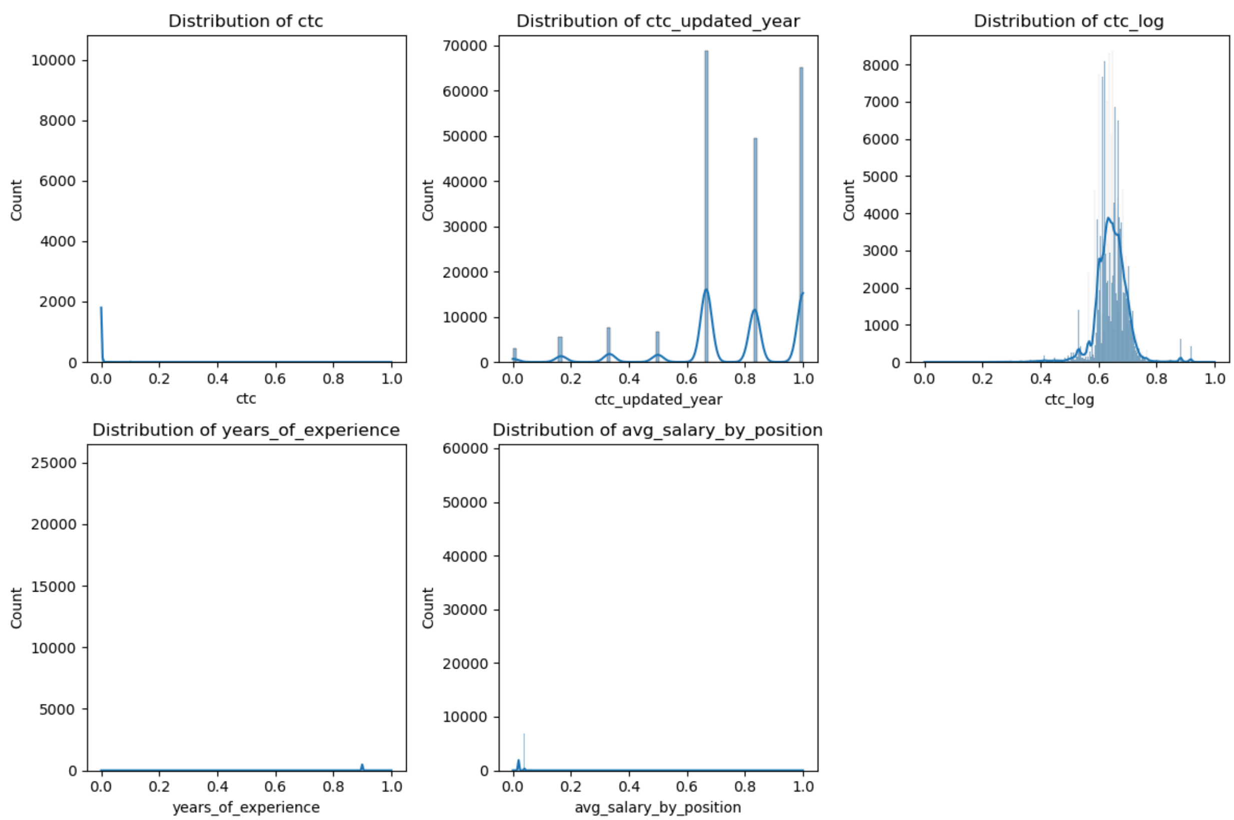
Evaluate Distribution of Newly Created Features

Compute Skewness and Kurtosis

In [49]:
```python
import pandas as pd

# Select relevant numeric features
numeric_features = ['ctc', 'ctc_updated_year', 'ctc_log', 'years_of_experience', 'avg_

# Compute skewness and kurtosis for each numeric feature
skewness_values = {}
kurtosis_values = {}

for feature in numeric_features:
    skewness_values[feature] = df[feature].skew()
    kurtosis_values[feature] = df[feature].kurtosis()

# Display skewness and kurtosis values
skewness_df = pd.DataFrame(skewness_values, index=['Skewness']).transpose()
kurtosis_df = pd.DataFrame(kurtosis_values, index=['Kurtosis']).transpose()

print("Skewness values:")
print(skewness_df)
print("\nKurtosis values:")
print(kurtosis_df)
```

```
Skewness values:
                         Skewness
ctc                     15.972492
ctc_updated_year        -1.182029
ctc_log                 -0.235224
years_of_experience   -219.905070
avg_salary_by_position  37.186778

Kurtosis values:
                            Kurtosis
ctc                       441.113823
ctc_updated_year            1.642902
ctc_log                     6.368954
years_of_experience     64799.070007
avg_salary_by_position   3029.040721
```

In [50]:
```
# Plot histograms for numeric features
plt.figure(figsize=(12, 8))

for i, feature in enumerate(numeric_features, 1):
    plt.subplot(2, 3, i)
    sns.histplot(df[feature], kde=True)
    plt.title(f"Distribution of {feature}")

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



Model Building

Train-Test Split for unsupervised learning Unsupervised Learning: In clustering, we are primarily interested in patterns and relationships within the data without explicit labels or target variables.

Evaluation with Unseen Data: Using a separate evaluation dataset helps assess how well the clustering model generalizes to new, unseen data.

Random State: Setting random_state ensures reproducibility of the split, which is important for consistent results in data analysis.

In [51]:
```
from sklearn.model_selection import train_test_split

# Assuming df_encoded is your preprocessed DataFrame for clustering

# Split the dataset into training (80%) and evaluation (20%) sets
train_data, eval_data = train_test_split(df, test_size=0.2, random_state=42)
```

```
# Extract features (X_train, X_eval) - No labels (unsupervised learning)
X_train = train_data.drop(columns=['id'])  # Assuming 'id' is not a feature for cluste
X_eval = eval_data.drop(columns=['id'])

# Display the shape of training and evaluation datasets
print("Training data shape:", X_train.shape)
print("Evaluation data shape:", X_eval.shape)
```

```
Training data shape: (164639, 1030)
Evaluation data shape: (41160, 1030)
```

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

Out[52]:

	id	company_hash	email_hash	orgyear	ctc	ctc_updat
0	0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016	0.00110	0
1	1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018	0.00045	0
2	2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015	0.00200	0
3	3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017	0.00070	0
4	4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017	0.00140	0

5 rows × 1031 columns

In [53]:
```
# Assuming df is your DataFrame with relevant features for clustering
features_for_clustering = ['ctc_log', 'years_of_experience', 'avg_salary_by_position']
X = df[features_for_clustering]
```

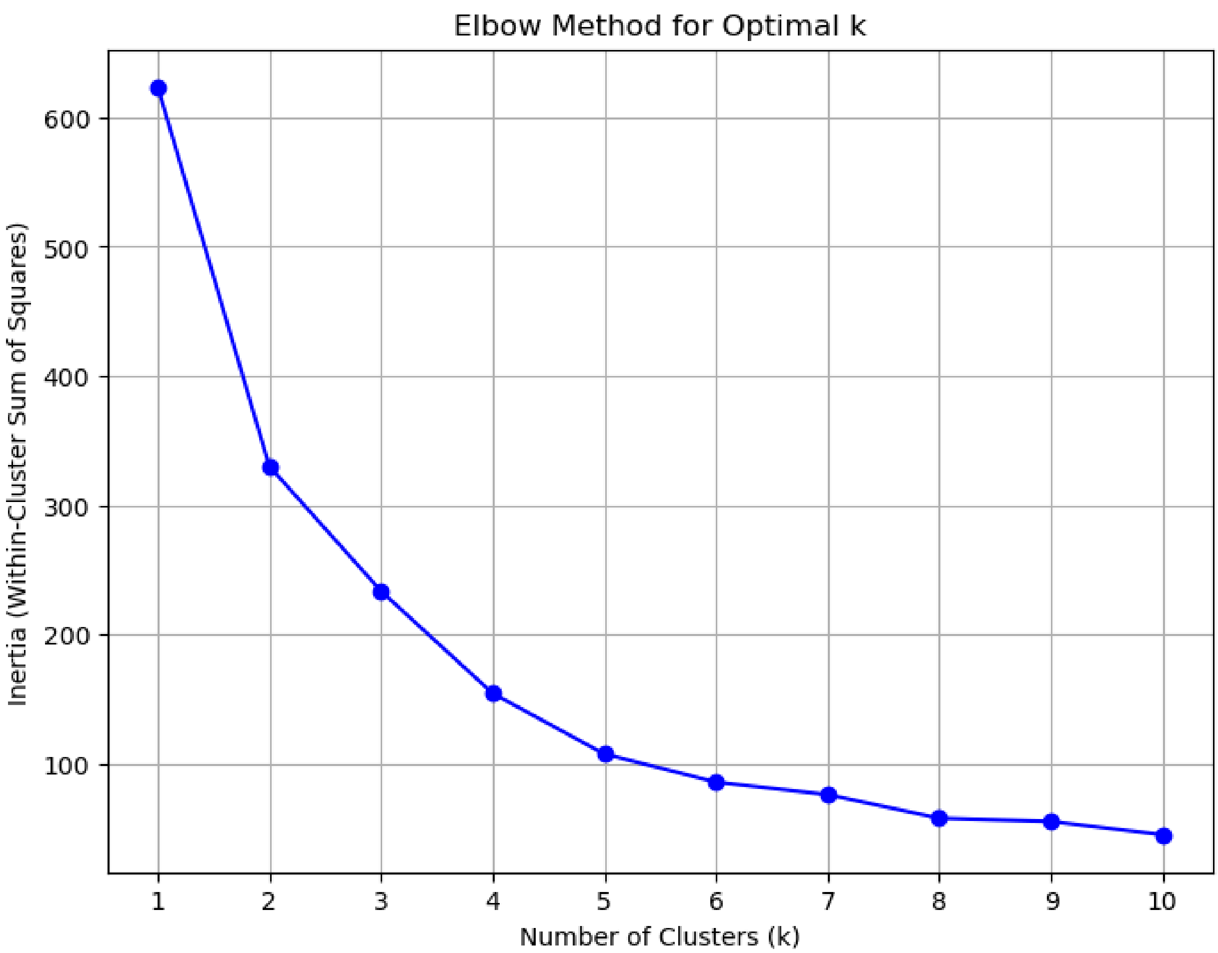
In [57]:
```
X.head()
```

Out[57]:

	ctc_log	years_of_experience	avg_salary_by_position
0	0.652858	0.900025	0.039723
1	0.607313	0.899926	0.018697
2	0.683321	0.900074	0.019695
3	0.629827	0.899975	0.019695
4	0.665147	0.899975	0.018697

In [55]:
```
from sklearn.cluster import KMeans

# Range of cluster numbers (k) to evaluate
k_range = range(1, 11)  # Try cluster numbers from 1 to 10

# List to store inertia values for each k
inertia_values = []

# Iterate over each value of k and fit KMeans model
```

```
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia_values.append(kmeans.inertia_)

# Plotting the Elbow Curve
plt.figure(figsize=(8, 6))
plt.plot(k_range, inertia_values, marker='o', linestyle='-', color='b')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Within-Cluster Sum of Squares)')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```



3 would be optimal number of cluster by elbow method.

In [59]:

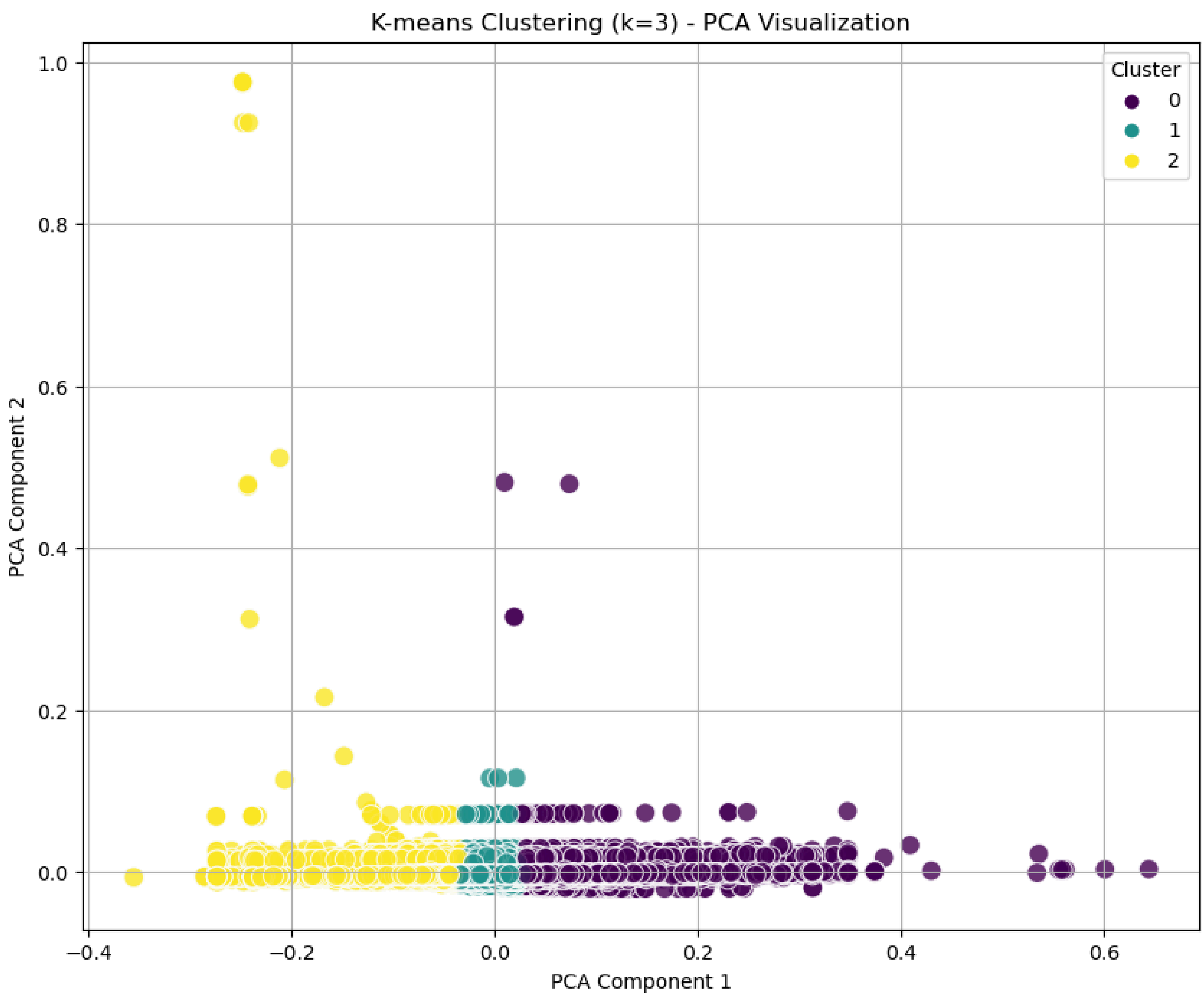
```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Obtain cluster labels for each data point
cluster_labels = kmeans.labels_

# Add cluster labels to the original DataFrame or create a new DataFrame with cluster
df['cluster_label'] = cluster_labels

# Display the DataFrame with cluster labels
print(df.head())
```

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

                                        email_hash  orgyear      ctc  \
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...     2016  0.00110
1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...     2018  0.00045
2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...     2015  0.00200
3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58...     2017  0.00070
4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...     2017  0.00140

   ctc_updated_year   ctc_log  years_of_experience  is_engineer  is_manager  \
0          0.833333  0.652858             0.900025            0           0
1          0.666667  0.607313             0.899926            1           0
2          0.833333  0.683321             0.900074            1           0
3          0.666667  0.629827             0.899975            1           0
4          0.666667  0.665147             0.899975            1           0

   ...  job_software developer - UI  job_software engineer 1  \
0  ...                            0                        0
1  ...                            0                        0
2  ...                            0                        0
3  ...                            0                        0
4  ...                            0                        0

   job_software engineer 2B  job_sr. developer  job_student  \
0                         0                  0            0
1                         0                  0            0
2                         0                  0            0
3                         0                  0            0
4                         0                  0            0

   job_support escalation engineer  job_system engineer  \
0                                0                    0
1                                0                    0
2                                0                    0
3                                0                    0
4                                0                    0

   job_system software engineer  job_technology analyst  cluster_label
0                             0                       0              1
1                             0                       0              0
2                             0                       0              2
3                             0                       0              1
4                             0                       0              1

[5 rows x 1032 columns]
```

In [60]:

```
from sklearn.decomposition import PCA
k = 3  # Example: Optimal number of clusters determined from the Elbow Curve

# Initialize and fit KMeans model with the chosen number of clusters (k)
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)

# Obtain cluster labels for each data point
cluster_labels = kmeans.labels_
```

```
# Add cluster labels to the original DataFrame or create a new DataFrame with cluster
df['cluster_label'] = cluster_labels

# Apply PCA to reduce dimensionality to 2D for visualization
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X)

# Create a DataFrame for visualization with PCA components and cluster labels
df_pca = pd.DataFrame({'PCA1': X_pca[:, 0], 'PCA2': X_pca[:, 1], 'Cluster': cluster_la

# Plot clusters in 2D PCA space
plt.figure(figsize=(10, 8))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster', data=df_pca, palette='viridis', s=1
plt.title('K-means Clustering (k={}) - PCA Visualization'.format(k))
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```



Hierarchical Clustering:

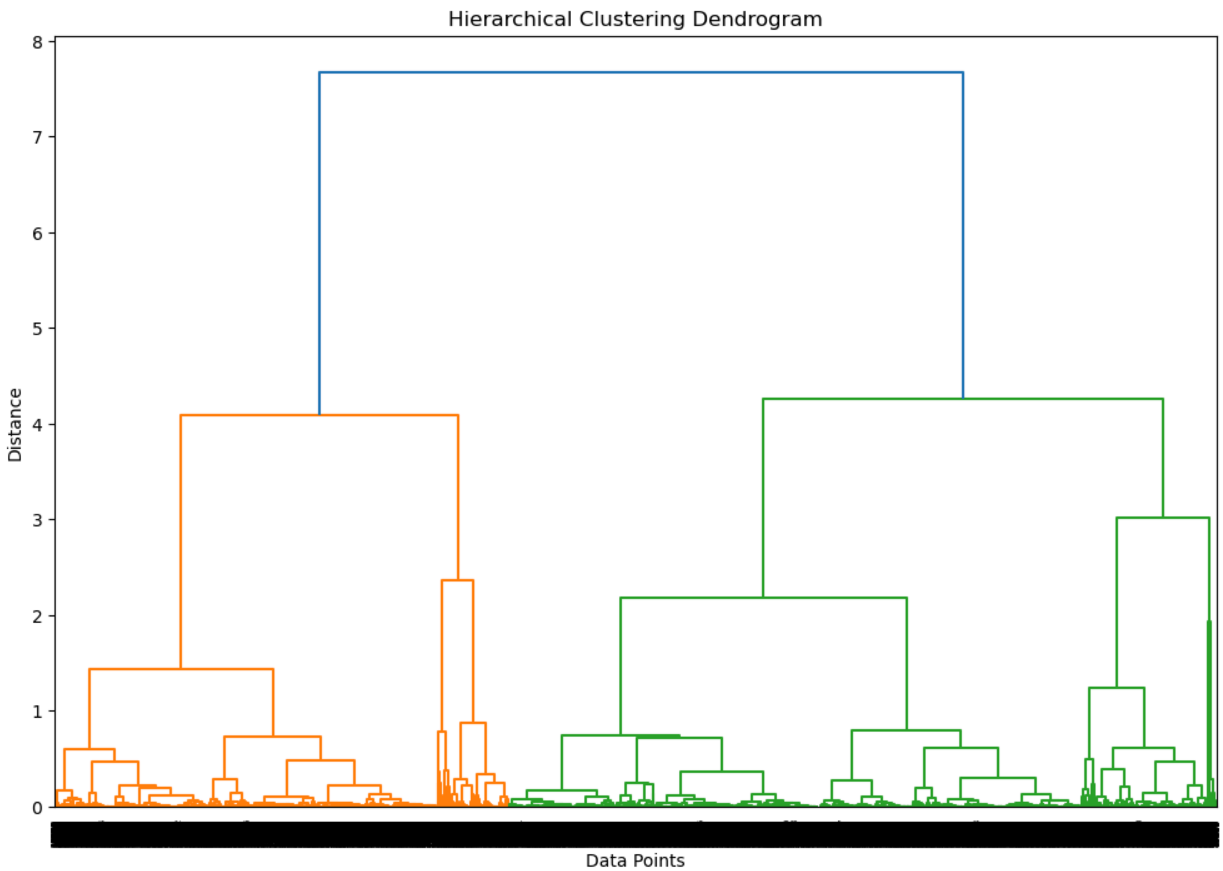
In [63]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster import hierarchy
```

In [66]:

```
# Sample a fraction (e.g., 10%) of the original DataFrame to reduce memory usage
sampled_df = df.sample(frac=0.1, random_state=42)

# Assuming X is your preprocessed and scaled feature matrix for clustering
X = sampled_df[features_for_clustering]

# Choose the desired number of clusters (n_clusters) for hierarchical clustering
n_clusters = 3  # Number of clusters to identify

# Initialize and fit AgglomerativeClustering model with the specified number of cluste
model = AgglomerativeClustering(n_clusters=n_clusters)

# Calculate linkage matrix for hierarchical clustering using sampled data
linkage_matrix = hierarchy.linkage(X, method='ward')  # Use Ward's method for linkage
```

In [68]:

```
# Plot dendrogram for hierarchical clustering using sampled data
plt.figure(figsize=(12, 8))
dendrogram = hierarchy.dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
```



In [ ]:

```
Green Cluster is bigger than orange cluster. AgglomerativeClustering algorithm to clus
your data into a specified number of clusters (n_clusters=3)
```

In [72]:

```
# cluster_counts = pd.Series(cluster_labels).value_counts()

# Determine the largest cluster
largest_cluster_label = cluster_counts.idxmax()
```

```
largest_cluster_size = cluster_counts.max()

# Calculate the percentage of users in the largest cluster
total_users = len(cluster_labels)
percentage_largest_cluster = (largest_cluster_size / total_users) * 100

print(f"Number of users in the largest cluster ({largest_cluster_label}): {largest_clu
print(f"Percentage of users in the largest cluster: {percentage_largest_cluster:.2f}%"
```

```
Number of users in the largest cluster (1): 110128
Percentage of users in the largest cluster: 53.51%
```

Cluster Means

In [71]:

```
# Add cluster labels to the DataFrame
df['cluster_label'] = cluster_labels

# Calculate mean feature values for each cluster
cluster_means = df.groupby('cluster_label').mean()

# Display the cluster characteristics (mean feature values)
print(cluster_means)
```

```
                        id      orgyear       ctc  ctc_updated_year  \
cluster_label
0             92785.712852  2016.399116  0.000334          0.797362
1            103651.918940  2015.315360  0.001038          0.770706
2            114183.535880  2012.110365  0.007475          0.743723

                ctc_log  years_of_experience  is_engineer  is_manager  \
cluster_label
0              0.581647             0.900005     0.473433    0.004852
1              0.646891             0.900059     0.575203    0.005939
2              0.709309             0.900218     0.602882    0.024750

               got_increment  salary_category  ...  \
cluster_label                                  ...
0                   0.143032         0.153446  ...
1                   0.140782         1.677058  ...
2                   0.185039         3.000000  ...

               job_senior software engineer-L2  job_software developer - UI  \
cluster_label
0                                     0.000000                     0.000000
1                                     0.000009                     0.000009
2                                     0.000000                     0.000000

               job_software engineer 1  job_software engineer 2B  \
cluster_label
0                             0.000020                  0.000000
1                             0.000045                  0.000009
2                             0.000022                  0.000000

               job_sr. developer  job_student  \
cluster_label
0                        0.00002     0.000000
1                        0.00000     0.000009
2                        0.00000     0.000022

               job_support escalation engineer  job_system engineer  \
cluster_label
0                                     0.000000              0.00002
1                                     0.000000              0.00000
2                                     0.000022              0.00000

               job_system software engineer  job_technology analyst
cluster_label
0                                  0.000000                 0.00002
1                                  0.000009                 0.00000
2                                  0.000000                 0.00000

[3 rows x 1029 columns]
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_2416\1647051861.py:5: FutureWarning: The
default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future ver
sion, numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  cluster_means = df.groupby('cluster_label').mean()
```

Recommendation

In [ ]:
```
There will be 3 clusters by elbow method.
Green clusters are bigger than orange cluster in Dendogram.
Number of users in the largest cluster (1): 110128
Percentage of users in the largest cluster: 53.51%
In k-means clustering 0 is the group which shows the lagest cluster of purple colour.
New Feature created on the basis low. medium and High salary.
'ctc_log', 'years_of_experience', 'avg_salary_by_position' are numerical features for
```

In [ ]:
```
```

In [ ]:
```
```