Scaler_Clustering_Case_Study

December 1, 2024

1 Problem Statement

As a data scientist working at Scaler focused on profiling the best companies and job profiles to work for, the task is to cluster a segment of learners based on their job profiles, companies and other features. Manual as well as Unsupervised algorithm based clustering need to be performed to reveal insights from the data.

1.1 Data Loading and basic data analysis

[199]:
```
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.sparse import hstack
from scipy.sparse import csr_matrix
import scipy.cluster.hierarchy as sch
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import TargetEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import KNNImputer
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.manifold import TSNE
```

[200]:
```
# loading data from url
data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/
 ↪000/002/856/original/scaler_clustering.csv")
```

[201]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 7 columns):
 #   Column            Non-Null Count   Dtype
```

```
---  ------            --------------   -----
 0   Unnamed: 0        205843 non-null  int64
 1   company_hash      205799 non-null  object
 2   email_hash        205843 non-null  object
 3   orgyear           205757 non-null  float64
 4   ctc               205843 non-null  int64
 5   job_position      153279 non-null  object
 6   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(2), object(3)
memory usage: 11.0+ MB
```

[202]:
```
data.head()
```

[202]:
```
   Unnamed: 0               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  1100000
1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...   2018.0   449999
2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...   2015.0  2000000
3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58...   2017.0   700000
4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...   2017.0  1400000

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

"Unnamed: 0" seem irrelevant for this case study for dropping it

[203]:
```
data = data.drop(columns = ["Unnamed: 0"])
```

[204]:
```
data.head()
```

[204]:
```
                company_hash  \
0             atrgxnnt xzaxv
1  qtrxvzwt xzegwgbb rxbxnta
2              ojzwnvwnxw vx
3                  ngpgutaxv
4                 qxen sqghu

                                        email_hash  orgyear      ctc  \
```

```
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05…   2016.0  1100000
1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10…   2018.0   449999
2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9…   2015.0  2000000
3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58…   2017.0   700000
4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520…   2017.0  1400000

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

[205]:
```
data.info()
```

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

Cleaning unwanted characters (if there exists any) from the "email_hash" using RegEx. Cleaning from "company_hash" is done after handling its missing entries.

[206]:
```
def clean(str_to_clean):
  if str_to_clean is np.nan:
    return str_to_clean
  else:
    return re.sub('[^A-Za-z0-9 ]+', '', str_to_clean)
```

[207]:
```
data["email_hash"] = data["email_hash"].apply(clean)
```

[208]:
```
print("Number of unique email hashes in the data - ",data["email_hash"].
↪nunique())
```

```
Number of unique email hashes in the data -  153443
```

[209]:
```
data["email_hash"].value_counts().head()
```

[209]:
```
email_hash
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee     9
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
b4d5afa09bec8689017d8b29701b80d664ca37b83cb883376b2e95191320da66     8
Name: count, dtype: int64
```

[210]:
```
data["company_hash"] = data["company_hash"].apply(clean)
```

[211]:
```
print("Number of unique company hashes in the data - ",data["company_hash"].
↪nunique())
```

```
Number of unique company hashes in the data -  37299
```

[212]:
```
data["company_hash"].value_counts().head()
```

[212]:
```
company_hash
nvnv wgzohrnvzwj otqcxwto    8337
xzegojo                      5381
vbvkgz                       3481
zgn vuurxwvmrt vwwghzn       3411
wgszxkvzn                    3240
Name: count, dtype: int64
```

1.1.1 Handling illogical values in "orgyear" column

[213]:
```
data["ctc_updated_year"].describe()
```

[213]:
```
count    205843.000000
mean       2019.628231
std           1.325104
min        2015.000000
25%        2019.000000
50%        2020.000000
75%        2021.000000
max        2021.000000
Name: ctc_updated_year, dtype: float64
```

[214]:
```
data["orgyear"].describe()
```

[214]:
```
count    205757.000000
mean       2014.882750
std          63.571115
min           0.000000
25%        2013.000000
50%        2016.000000
```

```
75%        2018.000000
max       20165.000000
Name: orgyear, dtype: float64
```

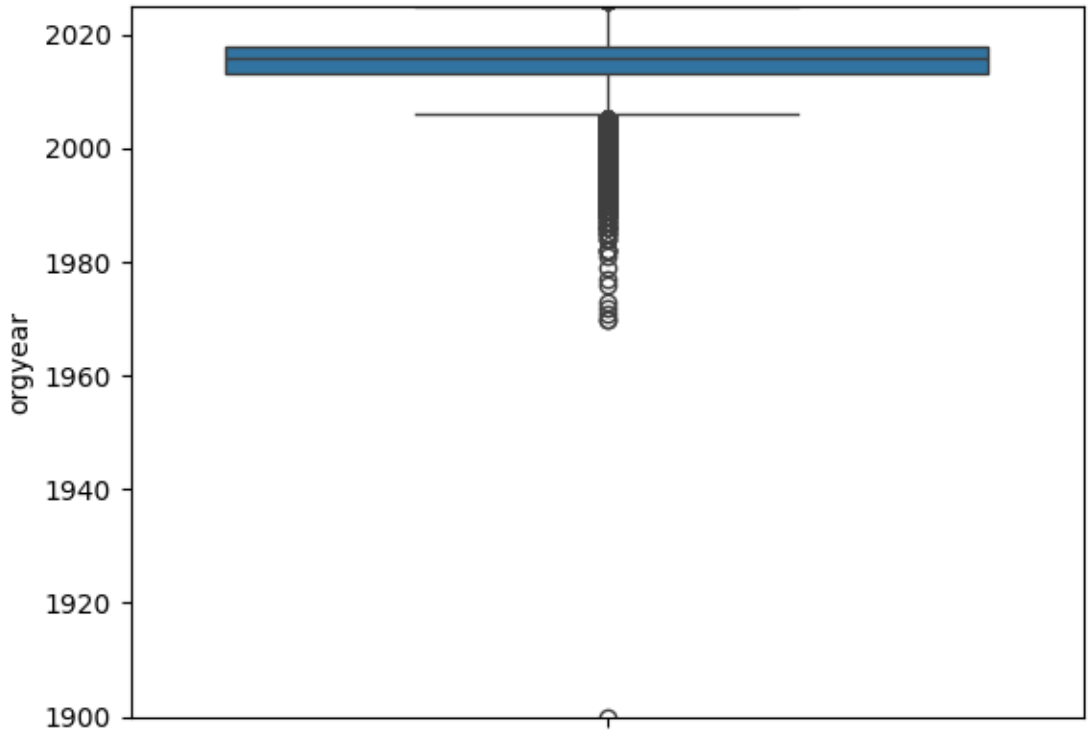
[215]:
```
data["orgyear"].nunique()
```

[215]: 77

[216]:
```
print(data["orgyear"].nunique(), data["orgyear"].min(), data['orgyear'].max())
```

```
77 0.0 20165.0
```

[217]:
```
sns.boxplot(data = data, y = "orgyear")
plt.ylim(1900, 2025)
plt.show()
```



[218]:
```
# finding the number of entries where 'orgyear' is less than 1960 or greater␣
↪than 2024
data[((data["orgyear"] < 1960.0) | (data["orgyear"] >= 2024.0))].shape[0]
```

[218]: 130

[219]:
```
# finding the number of entries where 'orgyear' is greater than␣
↪'ctc_updated_year'
data[data["orgyear"] > data["ctc_updated_year"]].shape[0]
```

[219]: 8847

It looks like there are a lot of spurious entries in the "orgyear" column.A spurious entry in 'orgyear' column has value either less than 1960 or greater than 2024 or it is greater than the corresponding entry in 'ctc_updated_year' column.

We can either drop these rows or fill them up with their corresponding entries of 'ctc_updated_year' column. We choose the second option, i.e., for any spurious entry in 'orgyear' column, we fill it with the corresponding entry in 'ctc_updated_year' column

[220]:
```
spurious_cond = (data['orgyear'] < 1960.0) | (data['orgyear'] > 2024.0) |␣
↪(data['orgyear'] > data['ctc_updated_year'])
spurious_indices = data[spurious_cond].index.tolist()

data.loc[spurious_indices, 'orgyear'] = data.loc[spurious_indices,␣
↪'ctc_updated_year']
```

[221]:
```
print(data["orgyear"].nunique(), data["orgyear"].min(), data['orgyear'].max())
```

```
47 1970.0 2021.0
```

1.1.2 Handling Null/Missing values

[222]:
```
print("Number of missing values in each column below :- \n")
print(data.isna().sum())
```

```
Number of missing values in each column below :-

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

Since number of rows with missing values in "company_hash" and "orgyear" are just 44 and 86 respectively out of 205843 rows, we drop them

[223]:
```
# dropping rows from the data where is there is missing entry in ...
# ..'company_hash' and 'orgyear'
data.dropna(subset = ["company_hash", "orgyear"], inplace = True)
```

[224]:
```
print(data.isna().sum())
```

```
company_hash            0
email_hash              0
orgyear                 0
ctc                     0
job_position        52509
ctc_updated_year        0
dtype: int64
```

[225]:
```
data["job_position"].value_counts(dropna = False, normalize = True)
```

[225]:
```
job_position
NaN                                                 0.255254
Backend Engineer                                    0.211654
FullStack Engineer                                  0.120099
Other                                               0.087782
Frontend Engineer                                   0.050614
                                                        …
Compliance auditor                                  0.000005
91                                                  0.000005
Senior Software Development Engineer (Backend)      0.000005
Messenger come driver                               0.000005
Android Application developer                       0.000005
Name: proportion, Length: 1017, dtype: float64
```

Roughly 25% of the entries in 'job_position' column are missing.

We will use KNN Imputation to fill the missing values. We will use only the numerical columns('orgyear', 'ctc' and 'ctc_updated_year') for computing similarities and ignore 'company_hash' and 'email_hash'. Row with missing value is assigned the modal value of 'job_position' of its k nearest neighbours.

[226]:
```
temp_data = data.copy()
```

While KNNImputation, we ignore the "email_hash" column

[227]:
```
temp_data.drop(columns=["email_hash", "company_hash"], inplace = True)
```

[228]:
```
temp_data.head()
```

[228]:

	orgyear	ctc	job_position	ctc_updated_year
0	2016.0	1100000	Other	2020.0
1	2018.0	449999	FullStack Engineer	2019.0
2	2015.0	2000000	Backend Engineer	2020.0
3	2017.0	700000	Backend Engineer	2019.0
4	2017.0	1400000	FullStack Engineer	2019.0

We first standardize the numerical columns (orgyear, ctc and ctc_updated_year)

[229]:
```
temp_data["orgyear"] = (temp_data["orgyear"] - temp_data["orgyear"].mean())/
 ↪temp_data["orgyear"].std()
temp_data["ctc"] = (temp_data["ctc"] - temp_data["ctc"].mean())/
 ↪temp_data["ctc"].std()
temp_data["ctc_updated_year"] = (temp_data["ctc_updated_year"] -␣
 ↪temp_data["ctc_updated_year"].mean())/temp_data["ctc_updated_year"].std()
```

[230]:
```
temp_data.head()
```

[230]:

	orgyear	ctc	job_position	ctc_updated_year
0	0.224105	-0.099255	Other	0.280439
1	0.702070	-0.154335	FullStack Engineer	-0.474202
2	-0.014877	-0.022989	Backend Engineer	0.280439
3	0.463088	-0.133150	Backend Engineer	-0.474202
4	0.463088	-0.073833	FullStack Engineer	-0.474202

[231]:
```
# Below function is used for missing value imputation in 'job_position' column..
 ↪.
# For each row with missing entry in 'job_position', we find its K nearest␣
 ↪neighbours..
# among the samples with NO missing value and then use the mode/majority vote␣
 ↪of ..
# neighbours. Ties are broken randomnly.

def knn_impute(df, missing_col_name, k = 5):
  # below dataframe cosists of rows with missing values of 'job_position'
  missing_data = df[df[missing_col_name].isna()]
  # below dataframe cosists of rows which have no missing entry in␣
 ↪'job_position'
  non_missing_data = df[~df[missing_col_name].isna()]

  lab_encoder = LabelEncoder()
  lab_encoder.fit(non_missing_data[missing_col_name])
  y_labels = lab_encoder.transform(non_missing_data[missing_col_name])

  non_missing_X = non_missing_data.drop(columns = [missing_col_name])

  knn_classifier = KNeighborsClassifier(n_neighbors = k)
  knn_classifier.fit(non_missing_X, y_labels)

  missing_X = missing_data.drop(columns = missing_col_name)
  neigh_dist, neigh_ind = knn_classifier.kneighbors(missing_X)

  num_rows_missing = neigh_ind.shape[0]
  for i in range(num_rows_missing):
    missing_og_ind_in_data = missing_data.index[i]
    k_neighbours_series = non_missing_data.iloc[neigh_ind[i]][missing_col_name]
```

```
    df.loc[missing_og_ind_in_data, missing_col_name] = k_neighbours_series.
  ↪mode()[0]

  return df
```

[232]:
```
temp_data_imputed = knn_impute(temp_data, 'job_position', 5)
```

[233]:
```
data_imputed = data.copy()
data_imputed["job_position"] = temp_data_imputed["job_position"]
```

1.1.3 Checking and deleting duplicated rows from the data

[234]:
```
# checking if there are duplicate rows
print("Number of duplicated rows in the data after imputation - ", data_imputed.
  ↪duplicated().sum())
```

```
Number of duplicated rows in the data after imputation -  8723
```

[235]:
```
data_dup_dropped = data_imputed.drop_duplicates()
```

[236]:
```
data_dup_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 196990 entries, 0 to 205842
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   company_hash      196990 non-null  object
 1   email_hash        196990 non-null  object
 2   orgyear           196990 non-null  float64
 3   ctc               196990 non-null  int64
 4   job_position      196990 non-null  object
 5   ctc_updated_year  196990 non-null  float64
dtypes: float64(2), int64(1), object(3)
memory usage: 10.5+ MB
```

[237]:
```
data_dup_dropped["email_hash"].value_counts().head()
```

[237]:
```
email_hash
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7e8cc6a20b0d1938183     8
b4d5afa09bec8689017d8b29701b80d664ca37b83cb883376b2e95191320da66     8
Name: count, dtype: int64
```

As we can see, there are some email hashes which occur more than once even after duplicate row removal.

This can happen because in hashing, two objects can map to the same hash value so we will assume that each of the emails mapping to the same hash are distinct after duplicate row removal.

1.1.4 Basic Feature Engineering

We add a new feature called 'years_of_experience' which is the number of years the employee has been working in the company since his joining.

[238]:
```
data_dup_dropped['years_of_experience'] = 2024.0 - data_dup_dropped["orgyear"]
```

```
<ipython-input-238-59eaef3f7614>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dup_dropped['years_of_experience'] = 2024.0 - data_dup_dropped["orgyear"]
```

1.2 Exploratory Data Analysis and Visualization

Generating Descriptive statistics of the columns of dataset below. The stats generated below are from the cleaned and imputed dataset.

[239]:
```
data_dup_dropped.describe()
```

[239]:

	orgyear	ctc	ctc_updated_year	years_of_experience
count	196990.000000	1.969900e+05	196990.000000	196990.000000
mean	2015.037652	2.293740e+06	2019.592741	8.962348
std	4.185849	1.194456e+07	1.331231	4.185849
min	1970.000000	2.000000e+00	2015.000000	3.000000
25%	2013.000000	5.300000e+05	2019.000000	6.000000
50%	2016.000000	9.500000e+05	2020.000000	8.000000
75%	2018.000000	1.700000e+06	2021.000000	11.000000
max	2021.000000	1.000150e+09	2021.000000	54.000000

[240]:
```
data_dup_dropped.describe(include = "object")
```

[240]:

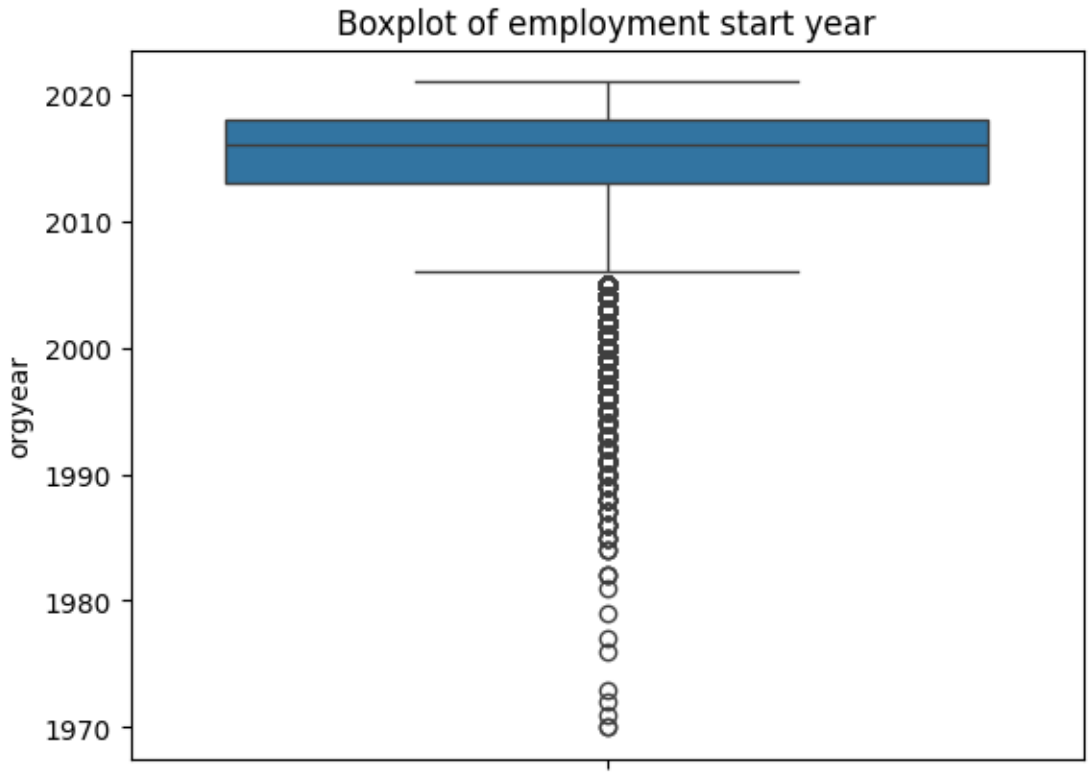
	company_hash	email_hash	job_position
count	196990	196990	196990
unique	37274	153333	1016
top	nvnv wgzohrnvzwj otqcxwto		
freq	7860		

```
top     bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7…  Backend Engineer
freq                                                 10              64101
```

1.2.1 Univariate Analysis

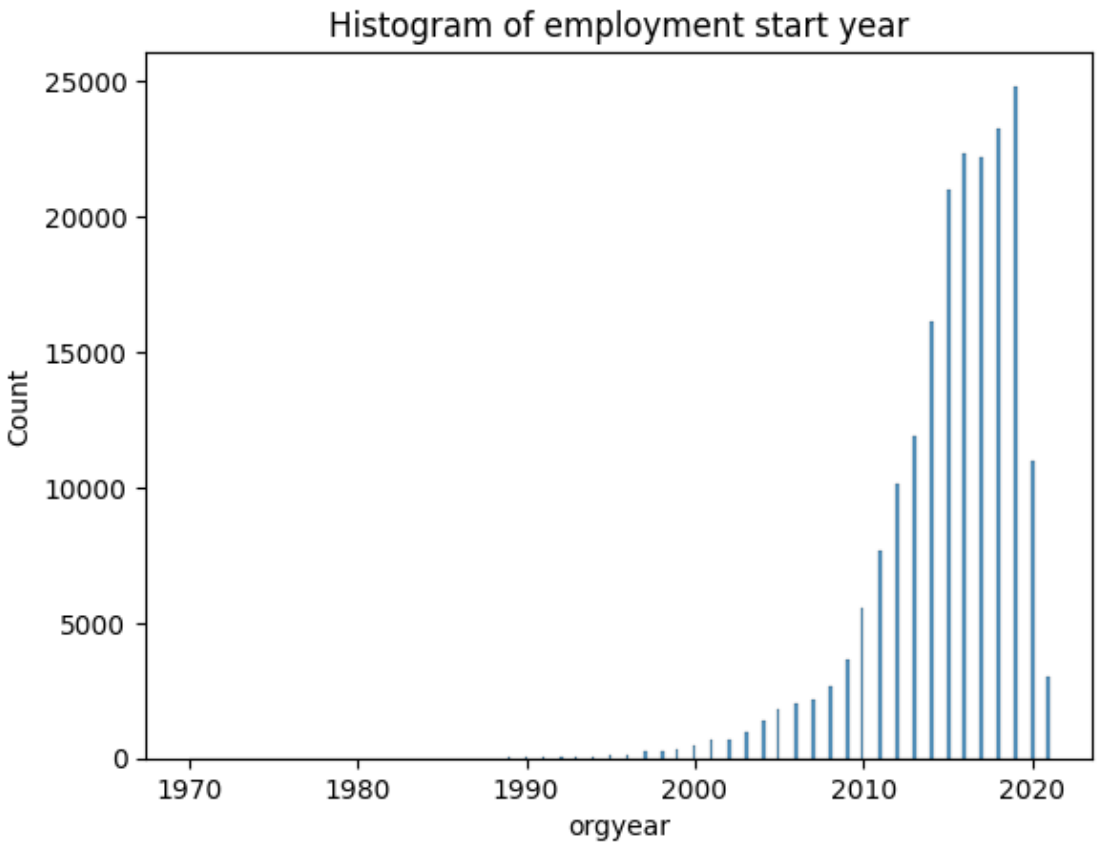
Boxplot of 'orgyear'(employment start date) of the cleaned dataset below

[241]:
```
sns.boxplot(data = data_dup_dropped, y = "orgyear")
plt.title("Boxplot of employment start year")
plt.show()
```



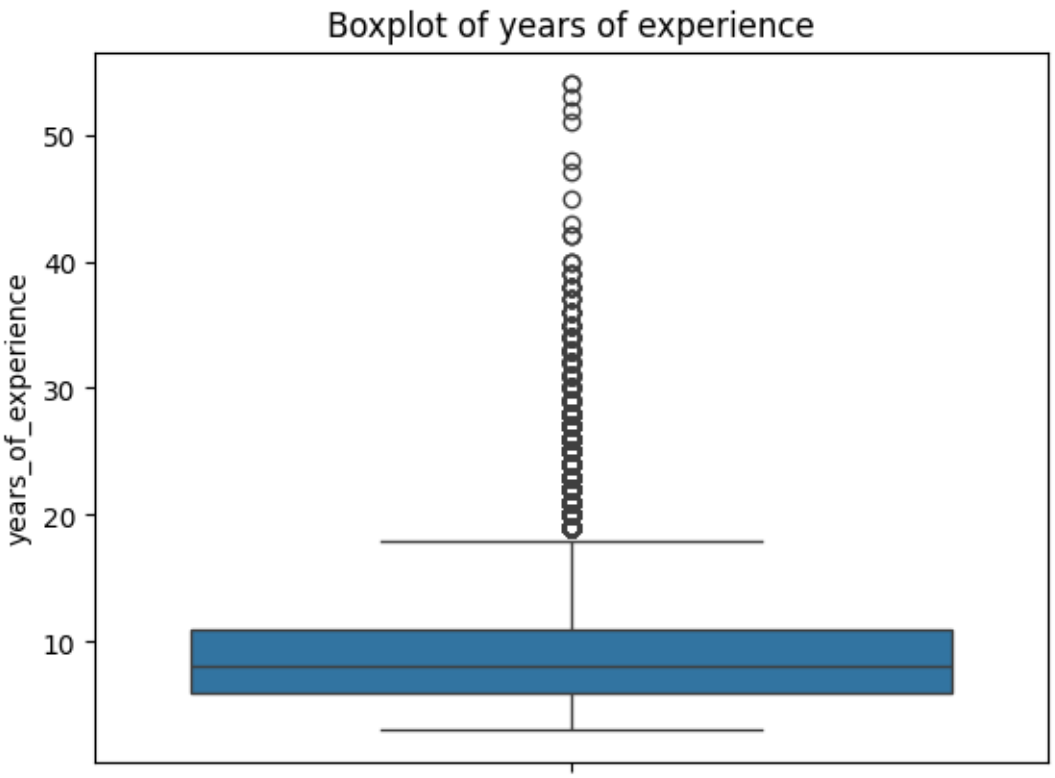
Histogram of 'orgyear' column below

[242]:
```
sns.histplot(data = data_dup_dropped, x = "orgyear")
plt.title("Histogram of employment start year")
plt.show()
```



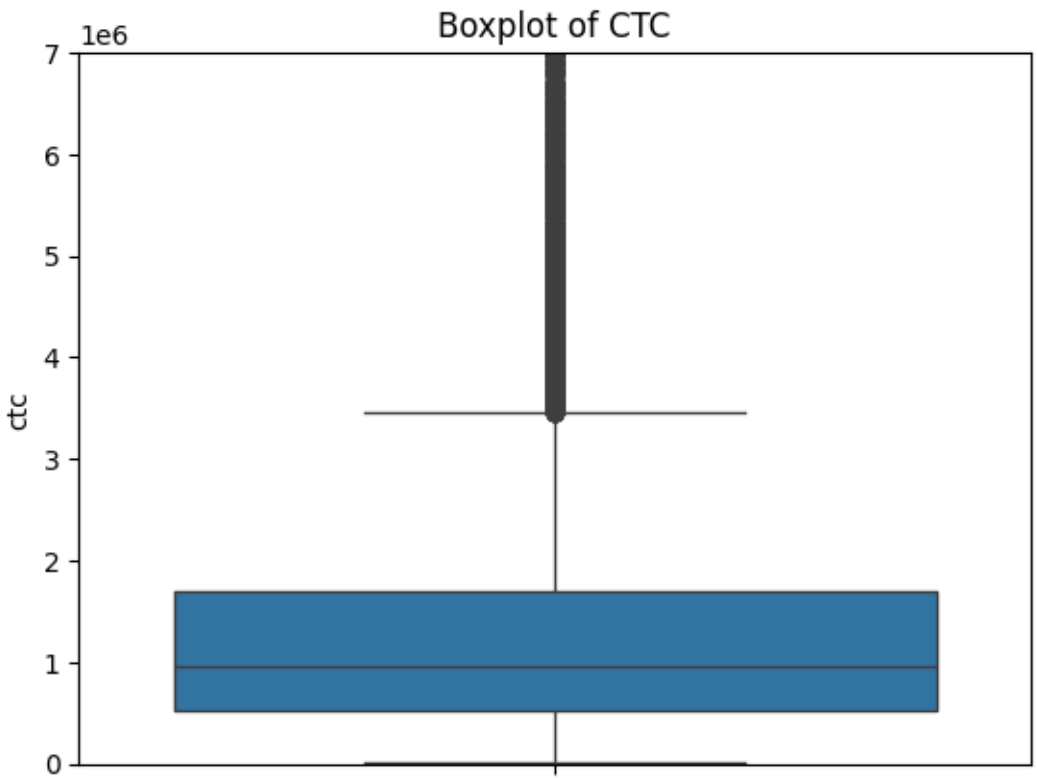
Boxplot of 'Years of Experience'(current year - employement start year) below

[243]:
```
sns.boxplot(data = data_dup_dropped, y = "years_of_experience")
plt.title("Boxplot of years of experience")
plt.show()
```



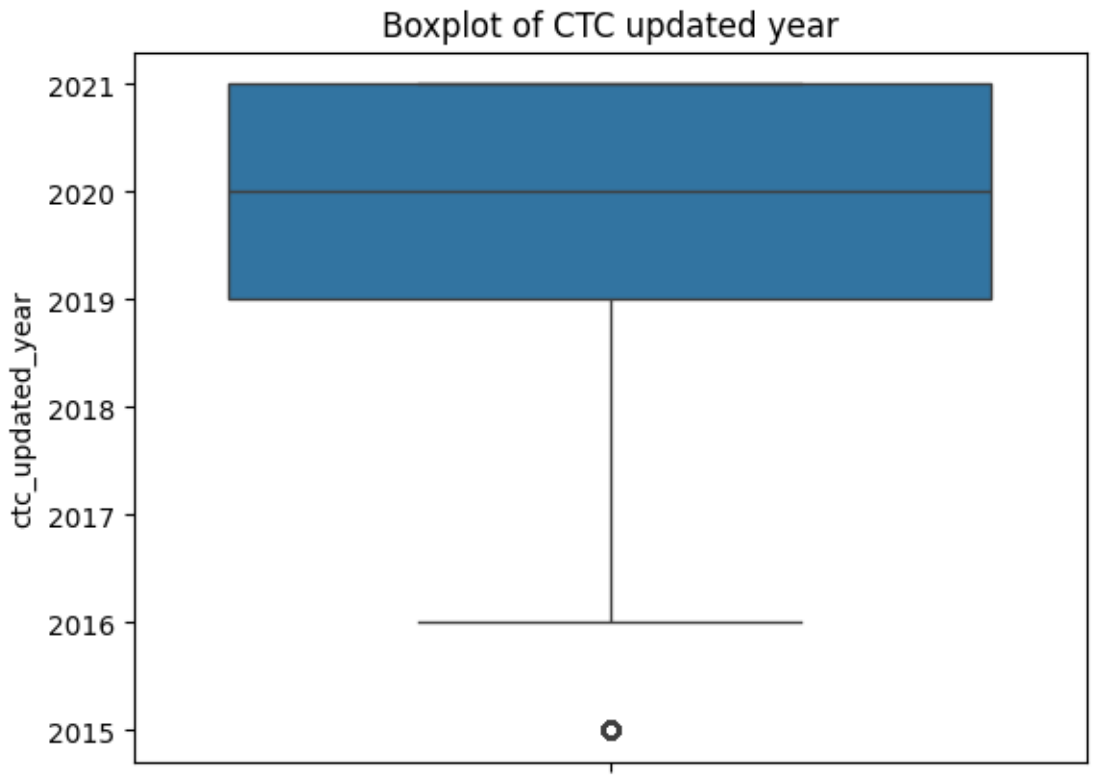
Boxplot of 'ctc' below

[244]:
```
sns.boxplot(data = data_dup_dropped, y = "ctc")
plt.ylim(-10, 7000000)
plt.title("Boxplot of CTC")
plt.show()
```



Boxplot of 'ctc_updated_year' (Year in which CTC got updated) column below

[245]:
```
sns.boxplot(data = data_dup_dropped, y = "ctc_updated_year")
plt.title("Boxplot of CTC updated year")
plt.show()
```



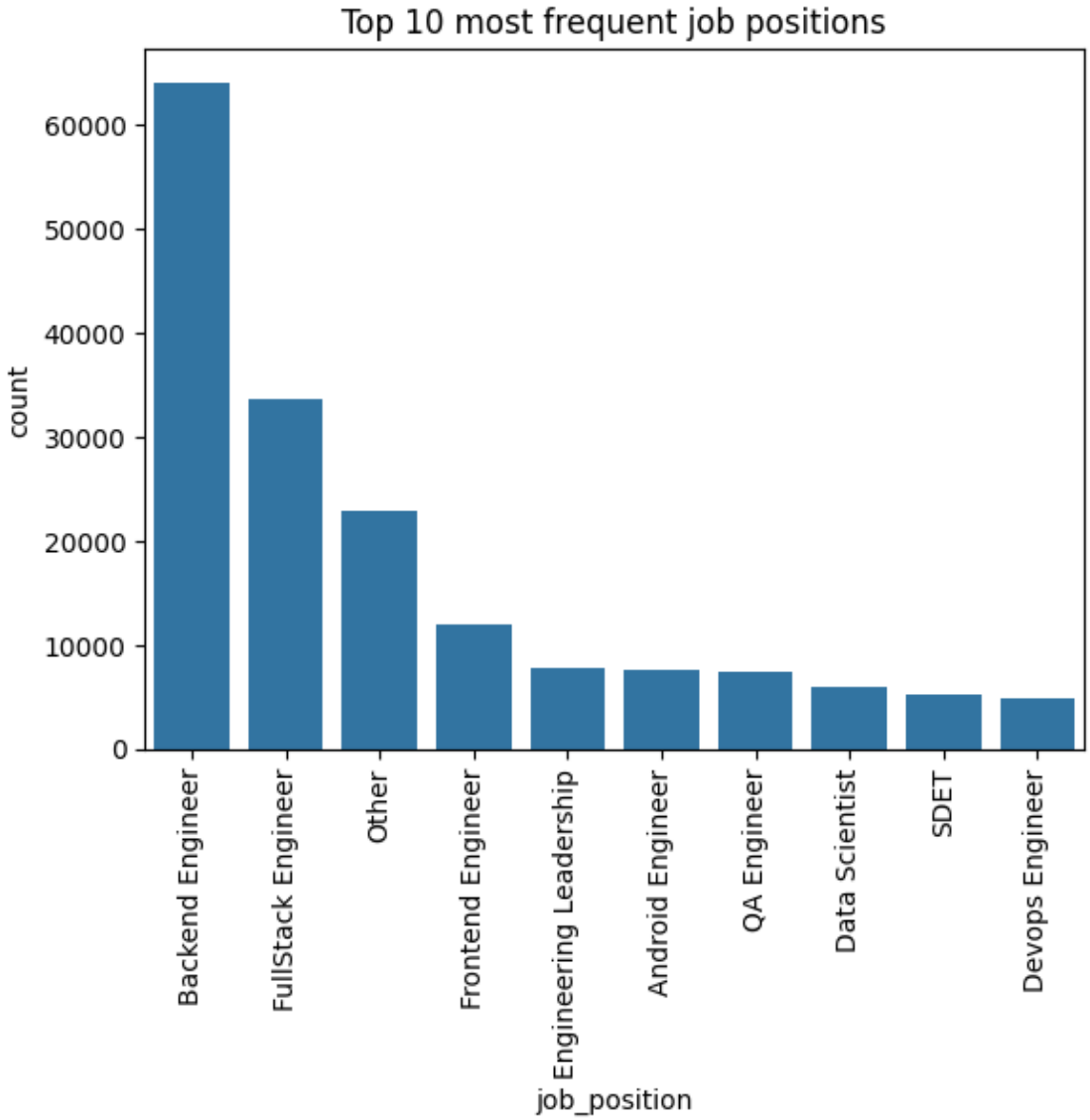
Countplot of top 10 most frequent job positions below

[246]:
```
top_10_frequent_jobs = data_dup_dropped["job_position"].value_counts().head(10)
top_10_frequent_jobs
```

[246]:
```
job_position
Backend Engineer          64101
FullStack Engineer        33524
Other                     22864
Frontend Engineer         11894
Engineering Leadership     7729
Android Engineer           7548
QA Engineer                7324
Data Scientist             5931
SDET                       5139
Devops Engineer            4899
Name: count, dtype: int64
```

[247]:
```
top_10_jobs = top_10_frequent_jobs.index.tolist()
top_10_data = data_dup_dropped[data_dup_dropped["job_position"].
 ↪isin(top_10_jobs)]
```

[248]:
```
sns.countplot(data = top_10_data, x = "job_position", order =␣
↪top_10_data["job_position"].value_counts().index)
plt.xticks(rotation = 90)
plt.title("Top 10 most frequent job positions")
plt.show()
```



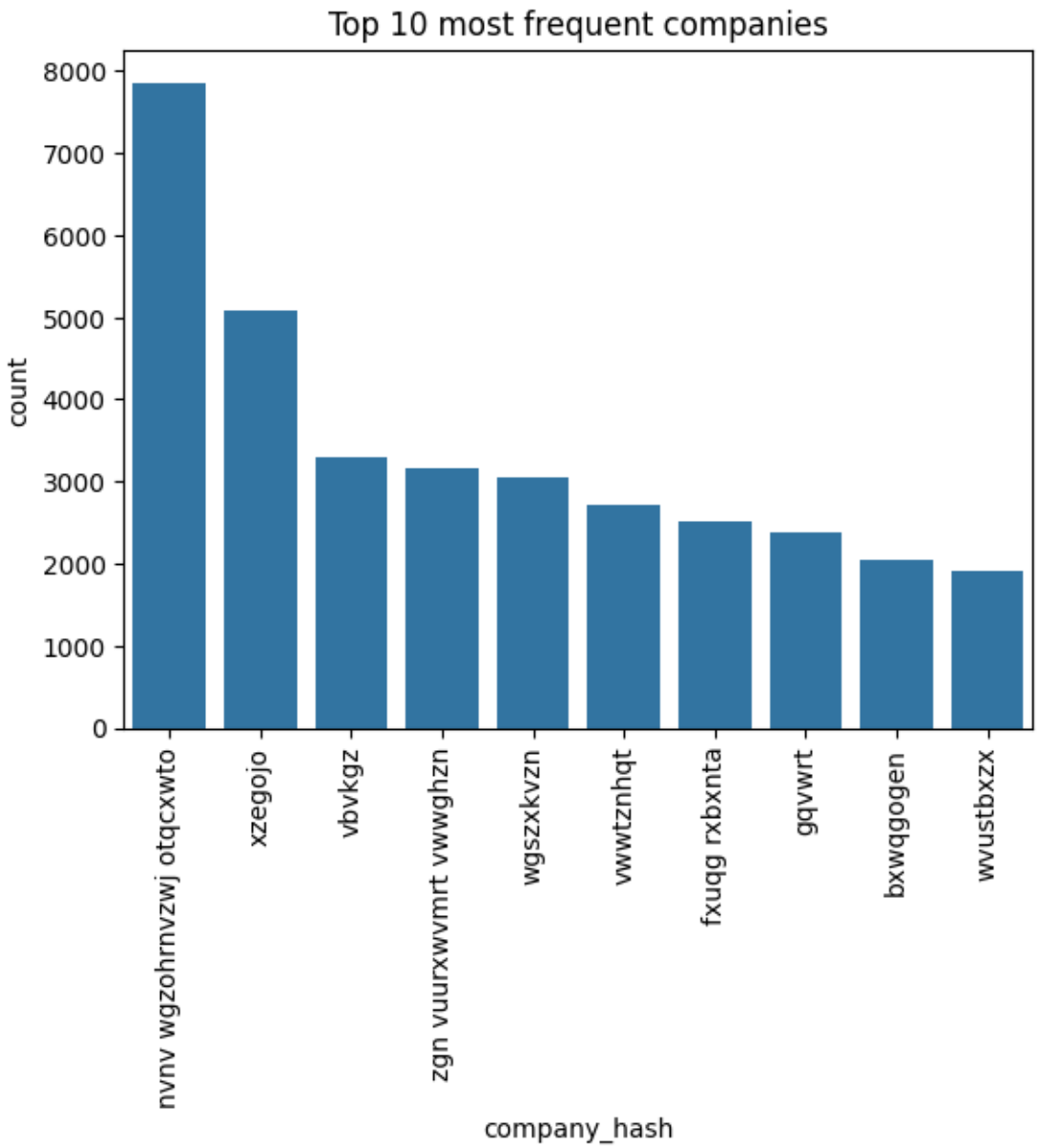
Countplot of top 10 most frequent company hash below

[249]:
```
top_10_frequent_companies = data_dup_dropped["company_hash"].value_counts().
↪head(10)
top_10_frequent_companies
```

[249]:
```
company_hash
nvnv wgzohrnvzwj otqcxwto    7860
xzegojo                      5071
vbvkgz                       3305
zgn vuurxwvmrt vwwghzn       3171
wgszxkvzn                    3045
vwwtznhqt                    2724
fxuqg rxbxnta                2521
gqvwrt                       2379
bxwqgogen                    2047
wvustbxzx                    1910
Name: count, dtype: int64
```

[250]:
```
top_10_companies = top_10_frequent_companies.index.tolist()
top_10_comp_data = data_dup_dropped[data_dup_dropped["company_hash"].
 ↪isin(top_10_companies)]
```

[251]:
```
sns.countplot(data = top_10_comp_data, x = "company_hash", order =␣
 ↪top_10_comp_data["company_hash"].value_counts().index)
plt.xticks(rotation = 90)
plt.title("Top 10 most frequent companies")
plt.show()
```



Investigating 'email_hash' column below

[252]:
```
email_hash_count = data_dup_dropped["email_hash"].value_counts()
email_hash_count[email_hash_count > 1]
```

[252]:
```
email_hash
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7e8cc6a20b0d1938183     8
```

```
b4d5afa09bec8689017d8b29701b80d664ca37b83cb883376b2e95191320da66     8
                                                                    ..
dcc09fa146e4b8ba685e90b19d4791faf6330d1728bf76a07d8449106f65abdc     2
9b536fa5e332dd71470a94771d77acd0ef6948f29b54384276e8914a0257a1ea     2
af9bb0405a2baf3fa904dafa98ae5d633787537b2dfdd8f754ffc4ee47bdf164     2
c677adb11aea2249f4559461c28fe97cfd347907d04370ac2dc9f37bf2c208e9     2
681fd2efa7ad6263fb1b73bc4b620b602b5b18ccd889d482959062212183f51e     2
Name: count, Length: 35420, dtype: int64
```

- 'orgyear'(year of start of employment) column has 25 percentile, median and 75% percentile values as 2013, 2016 and 2018 respectively. Most of the learners in the dataset have employment start date after 2010. There are a few outliers by IQR method. Such outliers have earlier employment start date, that is earlier than 2005.
- 'ctc'(Current CTC) column has 25 percentile, median and 75 percentile values as 530,000 , 950,000 and 1,700,000 respectively. Mean CTC is 2,293,739.63. Maximun CTC is 1,000,150,000. Such a difference between mean and median implies a long right-tailed distribution with high value outliers. This can be seen in the boxplot where there are too many high value outliers by IQR method.
- 'ctc_updated_year'(Year in which CTC got updated) column has 25 percentile, median and 75 percentile values as 2019, 2020 and 2021. Earliest 'ctc_updated_year' is 2015.
- Most frequent job position in the dataset(after cleaning and imputation) is Backend Engineer followed by FullStack Engineer. There are 1016 unique job positions in the dataset.
- There are 37274 companies in the dataset. Most frequent company, that is company where most learners in the dataset work, has the company hash 'nvnv wgzohrnvzwj otqcxwto'.
- There are 153,333 unique email hashes in the dataset. There are 35,420 email hash which occur more than once in the dataset.

After data cleaning, if there exists more than one occurrence of an email hash, we assume that those hashes belong to separate emails and thus, separate learners. This is because two different objects can map to same hash value in hashing.

1.2.2 Bivariate Analysis

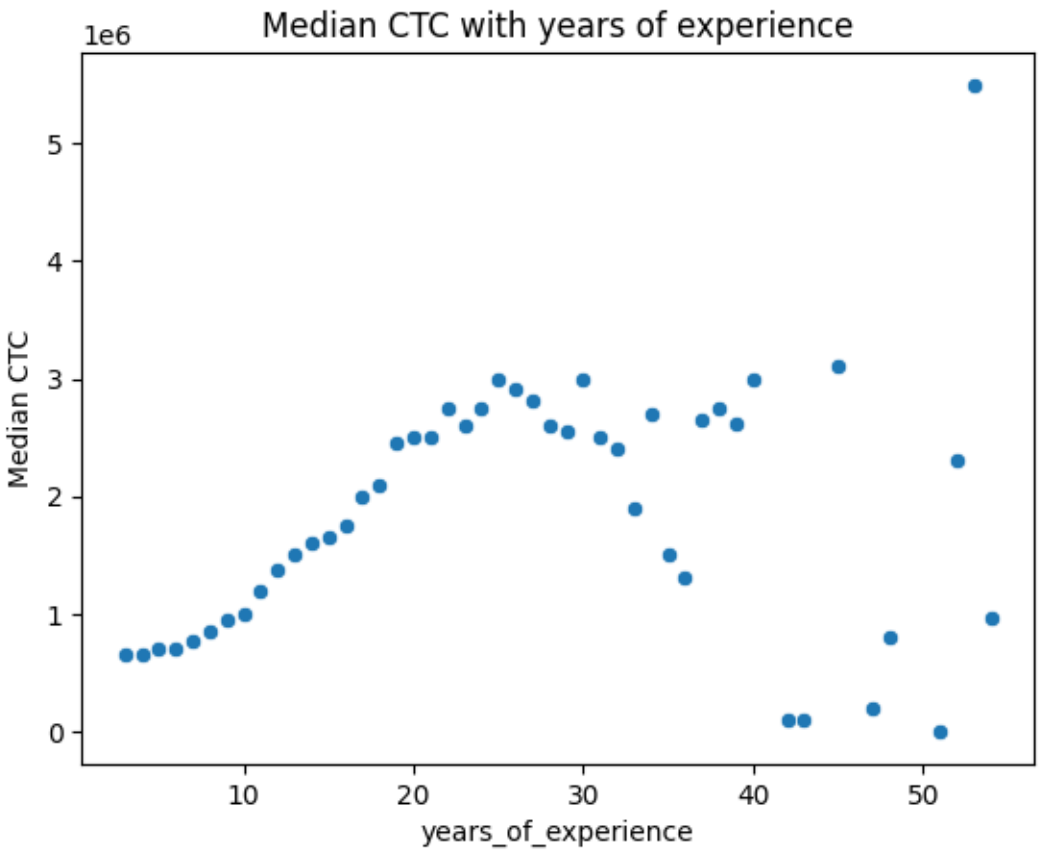
[253]:
```
data_dup_dropped.head()
```

[253]:
```
                company_hash  \
0             atrgxnnt xzaxv
1  qtrxvzwt xzegwgbb rxbxnta
2              ojzwnvwnxw vx
3                  ngpgutaxv
```

```
4                 qxen sqghu

                                          email_hash  orgyear      ctc  \
0  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05…   2016.0  1100000
1  b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10…   2018.0   449999
2  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9…   2015.0  2000000
3  effdede7a2e7c2af664c8a31d9346385016128d66bbc58…   2017.0   700000
4  6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520…   2017.0  1400000

         job_position  ctc_updated_year  years_of_experience
0               Other            2020.0                  8.0
1  FullStack Engineer            2019.0                  6.0
2    Backend Engineer            2020.0                  9.0
3    Backend Engineer            2019.0                  7.0
4  FullStack Engineer            2019.0                  7.0
```

Scatterplot of **Median** CTC with years of experience below

[254]:
```
ctc_with_yoe = data_dup_dropped.groupby("years_of_experience")["ctc"].median()
sns.scatterplot(x = ctc_with_yoe.index, y = ctc_with_yoe.values)
plt.ylabel("Median CTC")
plt.title("Median CTC with years of experience")
plt.show()
```



[255]:
```
# data where 'years_of_experience' > 40 years
data_dup_dropped[data_dup_dropped["years_of_experience"] > 40].sort_values(by =␣
↪"years_of_experience")
```

[255]:

	company_hash	email_hash	orgyear	ctc
18985	vroa	6cd568d2b4cc0584529da9b195dca0201a1e32ff82e3d0…	1982.0	100000
30602	gnytq			
185036	uqxzwxuvr srgmvr xzctongqo			
2199	xzaxvbvqno ehwxzs mhoxztoo bgahrt			
154872	ovmqt wgqugqvnxgz			
98126	djk			
92152	xzonxnhnt ge vtqgzvhnxwvr tzsxzttqxzs			
3908	sggsrt			
32665	st ytvrnywvqt			
97787	wygrvbvzavrvb xzctonbtzn vza exzvzwt wg rna			
31368	vbagwo			
145837	hzxctqoxnj ge zgqny ntdvo			

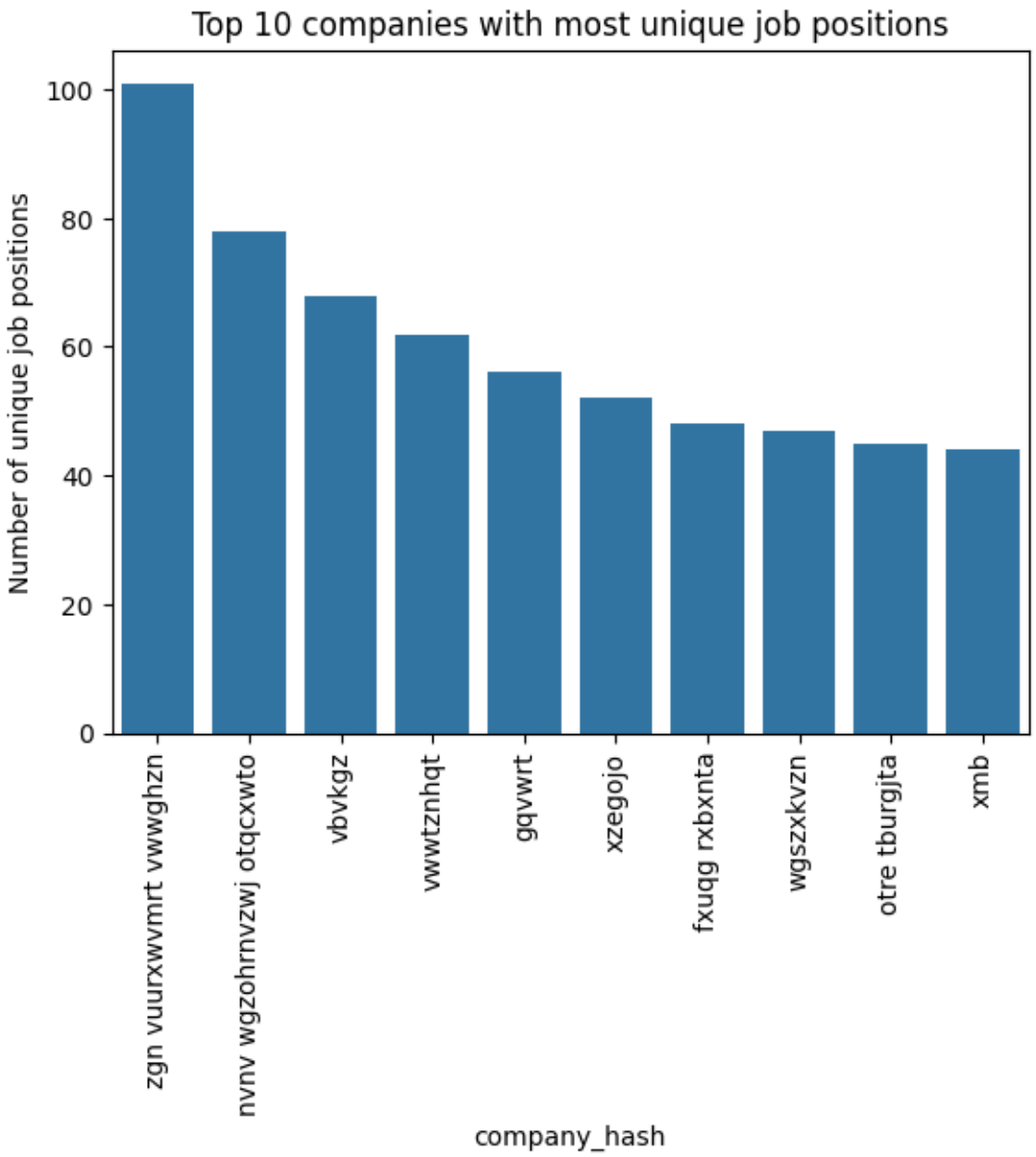
```
30602   da06126be06bf935865f52f7357c9f5fa937bf2290b815…   1982.0  1800000
185036  8855e533898b745ee9d8b2c621aaa017cba868fc86bcc7…   1982.0    60000
2199    e031d2f19dbdb54618078c835e030c384ad8360f7980db…   1981.0   100000
154872  31f887502f0c8b92259cf768adcca5414af7c209127bf7…   1979.0  3100000
98126   b246c9010fe1ef23e8ca01152670edc2728d783df2b307…   1977.0   200000
92152   6959d42a598119fe2ca41dcdc4e770951646cb1015c774…   1976.0   800000
3908    5756870d895deca920251df2377dad261084904a4f9d10…   1973.0     1000
32665   de72685914984cd9e1b0ec13223cd266f3c81d9517f282…   1972.0  2300000
97787   dc573e5ccc7f6d36b259b939f81655454a6e41a0f79fb5…   1971.0  5500000
31368   86dbdeada523d09881aec29ffa56ff63aca56f0278a97e…   1970.0  1800000
145837  e66b927f4ee3bd0d7202bbd35486d23d68555fc03dcd54…   1970.0   140000
```

	job_position	ctc_updated_year	years_of_experience
18985	Other	2021.0	42.0
30602	Other	2020.0	42.0
185036	Backend Engineer	2017.0	42.0
2199	Co-founder	2020.0	43.0
154872	Engineering Leadership	2019.0	45.0
98126	Other	2019.0	47.0
92152	Database Administrator	2020.0	48.0
3908	Co-founder	2020.0	51.0
32665	Engineering Leadership	2019.0	52.0
97787	Engineering Leadership	2019.0	53.0
31368	Engineering Leadership	2021.0	54.0
145837	Engineering Leadership	2020.0	54.0

Top 10 Companies with most unique job positions below

[256]:
```
top_10_comp_job = data_dup_dropped.groupby("company_hash")["job_position"].
↪nunique().sort_values(ascending = False).head(10)
```

[257]:
```
sns.barplot(x = top_10_comp_job.index, y = top_10_comp_job.values)
plt.xticks(rotation = 90)
plt.ylabel("Number of unique job positions")
plt.title("Top 10 companies with most unique job positions")
plt.show()
```



- Median CTC increases with years of experience as expected, however, after 30 years of experience, there is no uniform trend and we see ups and downs. This might be because there are very few learners with greater than 40 years of experience and thus, there is no sufficient data for a trend to emerge among learners with very high years of experience.

Remaining bivariate analysis especially the ones involving CTC will be done in subsequent section

of manual clustering.

1.3 Clustering

1.3.1 Manual Clustering

[258]:
```
# converting dtype of "ctc" column from int to float
data_dup_dropped["ctc"] = data_dup_dropped["ctc"].astype("float64")
```

```
<ipython-input-258-cefe19f4c1b2>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dup_dropped["ctc"] = data_dup_dropped["ctc"].astype("float64")
```

Grouping the dataframe by (company, job_position, years of experience) and then computing the 5 point summar (mean, median, max, min, count) of CTC of different groups below

[259]:
```
# 5 point summary of CTC grouped by (company, job position and years of␣
 ↪experience)
comp_job_years_grouped = data_dup_dropped.groupby(["company_hash",␣
 ↪"job_position", "years_of_experience"])
comp_job_years_ctc_agg = comp_job_years_grouped.agg(mean_ctc = ("ctc", "mean"),␣
 ↪median_ctc = ("ctc", "median")
                                    , max_ctc = ("ctc", "max")
                                    , min_ctc = ("ctc", "min")
                                    , ten_smallest_ctc = ("ctc", lambda x: x.
 ↪nsmallest(10).iloc[-1])
                                    , ten_largest_ctc = ("ctc", lambda x: x.
 ↪nlargest(10).iloc[-1])
                                    , count = ("ctc", "count"))
comp_job_years_ctc_agg.reset_index(inplace = True)
comp_job_years_ctc_agg.tail()
```

[259]:

	company_hash	job_position	years_of_experience	\
108878	zz	Other	15.0	
108879	zzb ztdnstz vacxogqj ucn rna	Backend Engineer	7.0	
108880	zzb ztdnstz vacxogqj ucn rna	FullStack Engineer	7.0	
108881	zzgato	Backend Engineer	10.0	
108882	zzzbzb	Other	34.0	

	mean_ctc	median_ctc	max_ctc	min_ctc	ten_smallest_ctc	\

```
108878  500000.0    500000.0  500000.0  500000.0          500000.0
108879  600000.0    600000.0  600000.0  600000.0          600000.0
108880  600000.0    600000.0  600000.0  600000.0          600000.0
108881  130000.0    130000.0  130000.0  130000.0          130000.0
108882  720000.0    720000.0  720000.0  720000.0          720000.0

        ten_largest_ctc  count
108878         500000.0      1
108879         600000.0      1
108880         600000.0      1
108881         130000.0      1
108882         720000.0      1
```

[260]:
```
# merging the aggregated data with the original datafame
three_level_agg_data = pd.merge(data_dup_dropped, comp_job_years_ctc_agg,
         on = ["company_hash", "job_position", "years_of_experience"])
```

Grouping the dataframe by (company, job_position) and then computing the 5 point summary (mean, median, max, min, count) of CTC of different groups below

[261]:
```
# 5 point summary of CTC grouped by (company, job position)
comp_job_grouped = data_dup_dropped.groupby(["company_hash", "job_position"])
comp_job_ctc_agg = comp_job_grouped.agg(mean_ctc = ("ctc", "mean"), median_ctc␣
 ↪= ("ctc", "median")
                                       , max_ctc = ("ctc", "max")
                                       , min_ctc = ("ctc", "min")
                                       , ten_smallest_ctc = ("ctc", lambda x: x.
 ↪nsmallest(10).iloc[-1])
                                       , ten_largest_ctc = ("ctc", lambda x: x.
 ↪nlargest(10).iloc[-1])
                                       , count = ("ctc", "count"))
comp_job_ctc_agg.reset_index(inplace = True)
comp_job_ctc_agg.tail()
```

[261]:
```
                     company_hash        job_position  mean_ctc  median_ctc  \
69276                          zz               Other  935000.0    935000.0
69277  zzb ztdnstz vacxogqj ucn rna    Backend Engineer  600000.0    600000.0
69278  zzb ztdnstz vacxogqj ucn rna  FullStack Engineer  600000.0    600000.0
69279                        zzgato    Backend Engineer  130000.0    130000.0
69280                        zzzbzb               Other  720000.0    720000.0

        max_ctc   min_ctc  ten_smallest_ctc  ten_largest_ctc  count
69276  1370000.0  500000.0         1370000.0         500000.0      2
69277   600000.0  600000.0          600000.0         600000.0      1
69278   600000.0  600000.0          600000.0         600000.0      1
69279   130000.0  130000.0          130000.0         130000.0      1
69280   720000.0  720000.0          720000.0         720000.0      1
```

[262]:
```
# merging the aggregated data with the original datafame
two_level_agg_data = pd.merge(data_dup_dropped, comp_job_ctc_agg,
         on = ["company_hash", "job_position"])
```

Grouping the dataframe by company and then computing the 5 point summary (mean, median, max, min, count) of CTC of different groups below

[263]:
```
# 5 point summary of CTC grouped by company
job_grouped = data_dup_dropped.groupby("company_hash")
comp_ctc_agg = job_grouped.agg(mean_ctc = ("ctc", "mean"), median_ctc = ("ctc",␣
 ↪"median")
                                    , max_ctc = ("ctc", "max")
                                    , min_ctc = ("ctc", "min")
                                    , ten_smallest_ctc = ("ctc", lambda x: x.
 ↪nsmallest(10).iloc[-1])
                                    , ten_largest_ctc = ("ctc", lambda x: x.
 ↪nlargest(10).iloc[-1])
                                    , count = ("ctc", "count"))
comp_ctc_agg.reset_index(inplace = True)
comp_ctc_agg.tail()
```

[263]:

	company_hash	mean_ctc	median_ctc	max_ctc	min_ctc	ten_smallest_ctc	ten_largest_ctc	count
37269	zyvzwt wgzohrnxzs tzsxzttqo	940000.0	940000.0	940000.0	940000.0	940000.0	940000.0	1
37270	zz	935000.0	935000.0	1370000.0	500000.0	1370000.0	500000.0	2
37271	zzb ztdnstz vacxogqj ucn rna	600000.0	600000.0	600000.0	600000.0	600000.0	600000.0	2
37272	zzgato	130000.0	130000.0	130000.0	130000.0	130000.0	130000.0	1
37273	zzzbzb	720000.0	720000.0	720000.0	720000.0	720000.0	720000.0	1

[264]:
```
# merging the aggregated data with the original datafame
one_level_agg_data = pd.merge(data_dup_dropped, comp_ctc_agg,
         on = "company_hash")
```

We need to create flags for the aggregated datasets. The process of assigning flags is not clear to me from the problem statement so I use my own understanding to assign flags.

- For data aggregated at company level, we need to create a flag named *Tier*.
  - For a company, **The employees with top 10 highest CTC and their CTC being higher than average CTC are Tier 1**. In case there are more than 10 people meeting

this criteria(for eg. many people with same CTC), all of them are assigned Tier 1. Given the criteria, this means that only those companies which have more than 10 employees can have Tier-1 employees.
  - For a company, **The employees with bottom 10 lowest CTC and their CTC being lower than average CTC are Tier 3**. In case there are more than 10 people meeting this criteria(for eg. many people with same CTC), all of them are assigned Tier 3. Given the criteria, this means that only those companies which have more than 10 employees can have Tier-3 employees.
  - Remaining employees of the company get assigned Tier-2.
  - For data aggregated at company and job_position level, we need to name a flag named *Class*.
  - For a specific job_position in a company, **The employees with top 10 highest CTC and their CTC being higher than average CTC are Class 1**. In case there are more than 10 people meeting this criteria(for eg. many people with same CTC), all of them are assigned Class 1. Given the criteria, this means that only those tuples of companies and job_positions which have more than 10 employees can have Class-1 employees.
  - For a specific job_position in a company, **The employees with bottom 10 lowest CTC and their CTC being lower than average CTC are Class 3**. In case there are more than 10 people meeting this criteria(for eg. many people with same CTC), all of them are assigned Class 3. Given the criteria, this means that only those those tuples of companies and job_positions which have more than 10 employees can have Class-3 employees.
  - Remaining employees of the company get assigned Class-2.
- For data aggregated at company, job_position and years_of_experience level, we need to name a flag named *Designation*. The process to assign a particular designation to an employee is similar to the ones above.

Assigning flag named desgination to (company, job, YOE) aggregated data below

[265]:
```
# assigning flag named desgination to (company, job, YOE) aggregated data

three_level_agg_data["designation"]  = 2
```

[266]:
```
desig_one_condition = ((three_level_agg_data["count"] > 10) &
                       (three_level_agg_data["ctc"] >␣
 ↪three_level_agg_data["mean_ctc"]) &
                       (three_level_agg_data["ctc"] >=␣
 ↪three_level_agg_data["ten_largest_ctc"]))
```

[267]:
```
desig_three_condittion = ((three_level_agg_data["count"] > 10) &
                          (three_level_agg_data["ctc"] <␣
 ↪three_level_agg_data["mean_ctc"]) &
                           (three_level_agg_data["ctc"] <=␣
 ↪three_level_agg_data["ten_smallest_ctc"]))
```

[268]:
```
desig_one_index = three_level_agg_data[desig_one_condition].index
desig_three_index = three_level_agg_data[desig_three_condittion].index
```

[269]:
```
three_level_agg_data.loc[desig_one_index, "designation"] = 1
three_level_agg_data.loc[desig_three_index, "designation"] = 3
```

Assigning flag named Class to (company, job) aggregated data below

[270]:
```
# Assigning flag named Class to (company, job) aggregated data
two_level_agg_data["Class"] = 2
```

[271]:
```
class_one_condition = ((two_level_agg_data["count"] > 10) &
                       (two_level_agg_data["ctc"] >␣
 ↪two_level_agg_data["mean_ctc"]) &
                       (two_level_agg_data["ctc"] >=␣
 ↪two_level_agg_data["ten_largest_ctc"]))
```

[272]:
```
class_three_condittion = ((two_level_agg_data["count"] > 10) &
                          (two_level_agg_data["ctc"] <␣
 ↪two_level_agg_data["mean_ctc"]) &
                           (two_level_agg_data["ctc"] <=␣
 ↪two_level_agg_data["ten_smallest_ctc"]))
```

[273]:
```
class_one_index = two_level_agg_data[class_one_condition].index
class_three_index = two_level_agg_data[class_three_condittion].index
```

[274]:
```
two_level_agg_data.loc[class_one_index, "Class"] = 1
two_level_agg_data.loc[class_three_index, "Class"] = 3
```

Assigning flag named Tier to company aggregated data below

[275]:
```
# Assigning flag named Tier to company aggregated data below

one_level_agg_data["Tier"] = 2
```

[276]:
```
tier_one_condition = ((one_level_agg_data["count"] > 10) &
                      (one_level_agg_data["ctc"] >␣
 ↪one_level_agg_data["mean_ctc"]) &
                      (one_level_agg_data["ctc"] >=␣
 ↪one_level_agg_data["ten_largest_ctc"]))
```

[277]:
```
tier_three_condition = ((one_level_agg_data["count"] > 10) &
                        (one_level_agg_data["ctc"] <␣
 ↪one_level_agg_data["mean_ctc"]) &
                         (one_level_agg_data["ctc"] <=␣
 ↪one_level_agg_data["ten_smallest_ctc"]))
```

[278]:
```
tier_one_index = one_level_agg_data[tier_one_condition].index
tier_three_index = one_level_agg_data[tier_three_condition].index
```

[279]:
```
one_level_agg_data.loc[tier_one_index, "Tier"] = 1
one_level_agg_data.loc[tier_three_index, "Tier"] = 3
```

Now we do investigation using the manual clustering results

Entry level position that has few learners with unusually high CTC in the dataset. We define entry level job positions as those which have learners with the lowest years of experience.

[280]:
```
min_exp = data_dup_dropped["years_of_experience"].min()
print("Minimum years of job experience in the dataset - ", min_exp)
```

```
Minimum years of job experience in the dataset -  3.0
```

[281]:
```
entry_level_data = data_dup_dropped[data_dup_dropped["years_of_experience"] ==␣
↪min_exp]
entry_level_data["job_position"].value_counts().head(10)
```

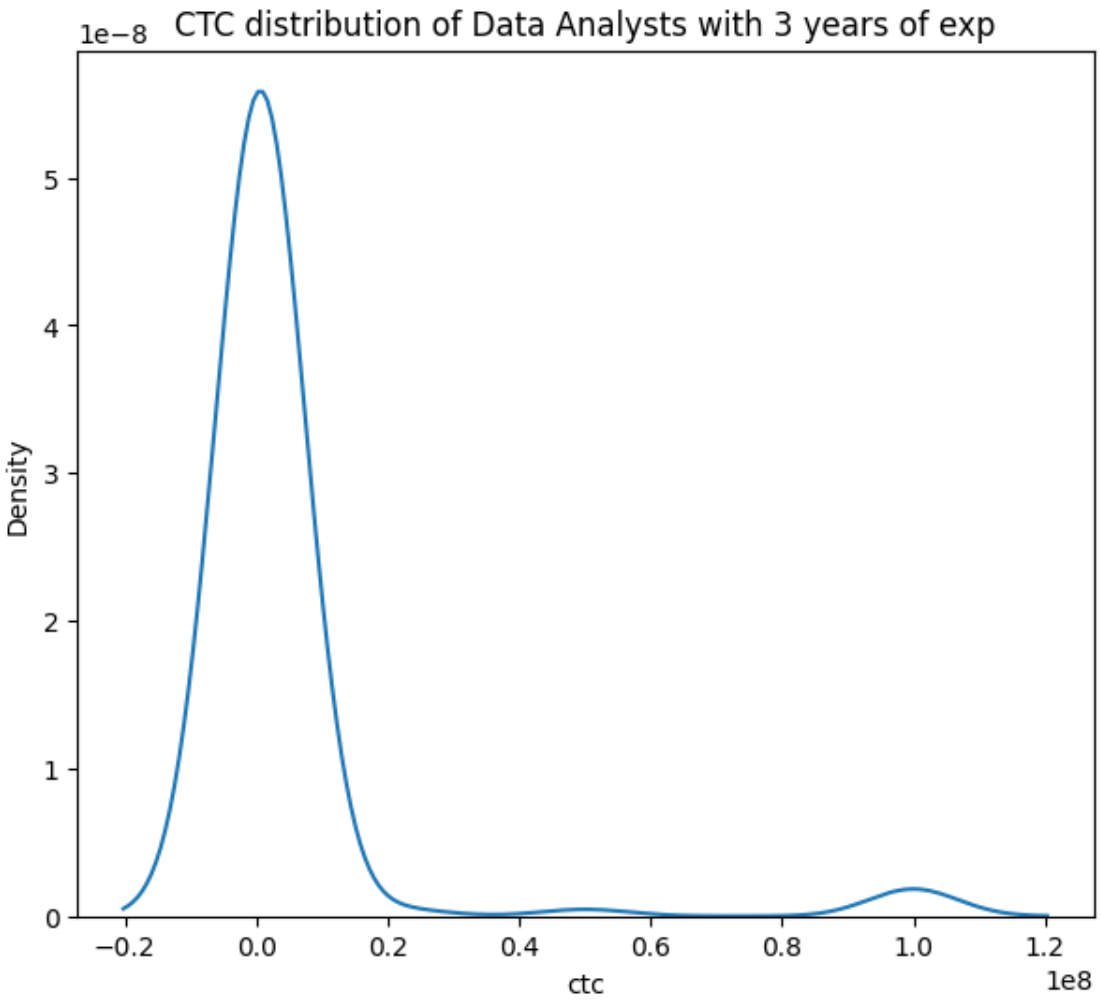
[281]:
```
job_position
Backend Engineer          758
Other                     641
FullStack Engineer        588
Engineering Intern        211
Frontend Engineer         167
Data Analyst              128
Android Engineer           86
Data Scientist             73
Engineering Leadership     55
Support Engineer           55
Name: count, dtype: int64
```

Data Analyst is usually considered an 'entry-level' job. We investigate the CTC distribution among data analyst learners with lowest job experience.

[282]:
```
data_analyst_data = entry_level_data[entry_level_data["job_position"] == "Data␣
↪Analyst"]
data_analyst_data["ctc"].describe()
```

[282]:
```
count    1.280000e+02
mean     4.246625e+06
std      1.790721e+07
min      1.000000e+05
25%      3.000000e+05
50%      4.000000e+05
75%      8.000000e+05
max      1.000000e+08
Name: ctc, dtype: float64
```

[283]:
```
plt.figure(figsize = (7,6))
sns.kdeplot(data = data_analyst_data, x = "ctc")
plt.title("CTC distribution of Data Analysts with 3 years of exp")
plt.show()
```



We see that there are a few data analysts who have unusually high CTC despite just 3 years of experience.

Top 10 job positions with highest mean CTC

[284]:
```
job_grouped = data_dup_dropped.groupby("job_position")
job_agg_data = job_grouped.agg(mean_ctc = ("ctc", "mean"), median_ctc = ("ctc",␣
↪"median")
                                , max_ctc = ("ctc", "max")
                                , min_ctc = ("ctc", "min")
                                , count = ("ctc", "count"))
```

```
job_agg_data.sort_values(by = "mean_ctc", ascending = False).head(10)
```

[284]:

job_position	mean_ctc	median_ctc	max_ctc	min_ctc	count
Telar	100000000.0	100000000.0	100000000.0	100000000.0	1
Data entry	100000000.0	100000000.0	100000000.0	100000000.0	1
Business Man	100000000.0	100000000.0	100000000.0	100000000.0	1
Jharkhand	100000000.0	100000000.0	100000000.0	100000000.0	1
Reseller	100000000.0	100000000.0	100000000.0	100000000.0	1
Computer Scientist 2	100000000.0	100000000.0	100000000.0	100000000.0	1
7033771951	100000000.0	100000000.0	100000000.0	100000000.0	1
Seleceman	99900000.0	99900000.0	99900000.0	99900000.0	1
Safety officer	99900000.0	99900000.0	99900000.0	99900000.0	1
Driver	95000000.0	95000000.0	100000000.0	90000000.0	2

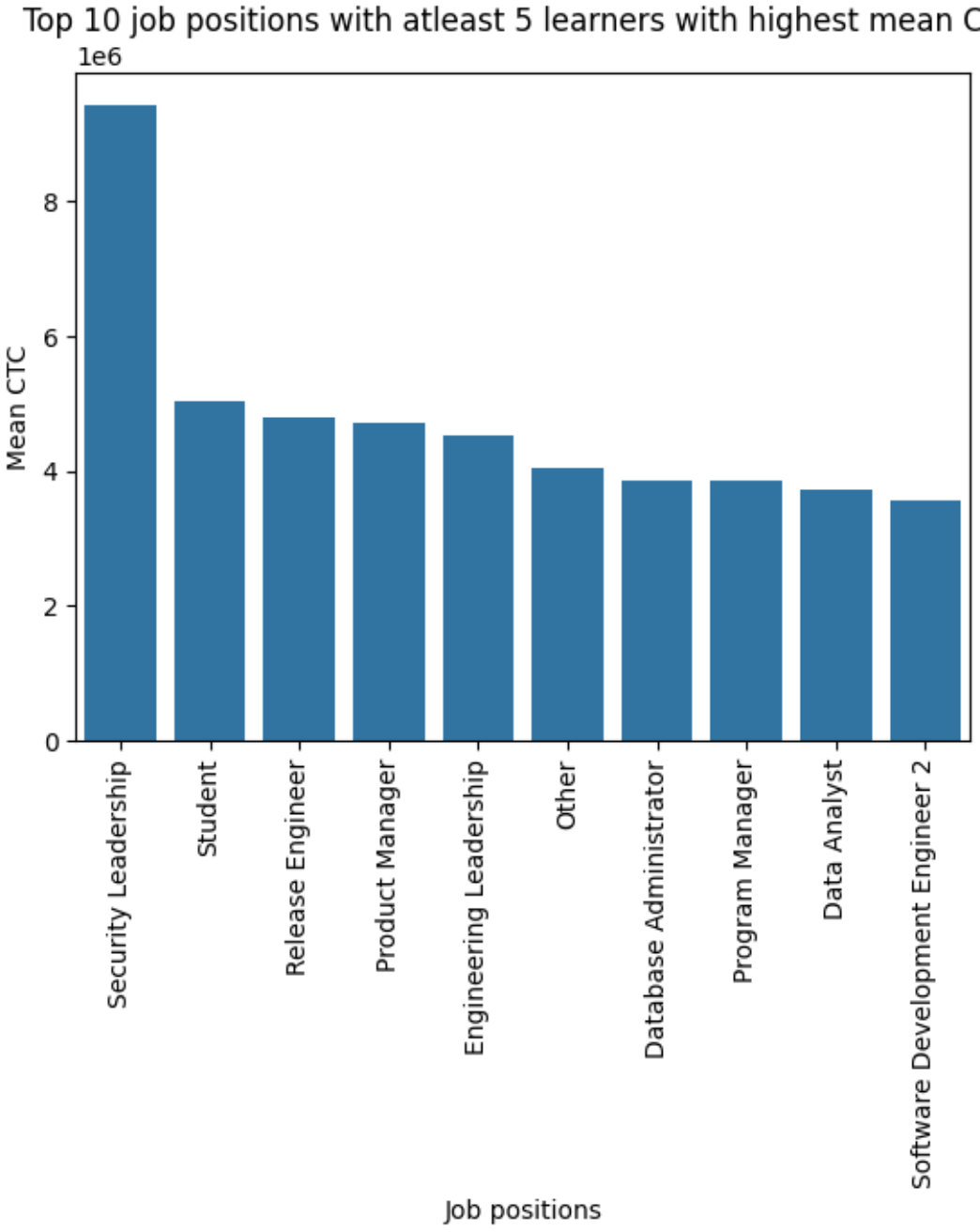
There are single learner jobs in the above so we explore among those jobs **which have atleast 5 learners**.

[285]:
```
top_10_job_pos = job_agg_data[job_agg_data["count"] > 5].sort_values(by =␣
↪"mean_ctc", ascending = False).head(10)
top_10_job_pos[["mean_ctc", "median_ctc", "count"]]
```

[285]:

job_position	mean_ctc	median_ctc	count
Security Leadership	9.430426e+06	1190000.0	148
Student	5.036522e+06	450000.0	23
Release Engineer	4.809151e+06	900000.0	126
Product Manager	4.729641e+06	1715000.0	1172
Engineering Leadership	4.540346e+06	2600000.0	7729
Other	4.039322e+06	600000.0	22864
Database Administrator	3.855444e+06	520000.0	765
Program Manager	3.853453e+06	2520000.0	835
Data Analyst	3.721739e+06	700000.0	3598
Software Development Engineer 2	3.571429e+06	4000000.0	7

[286]:
```
sns.barplot(x = top_10_job_pos.index, y = top_10_job_pos["mean_ctc"])
plt.xticks(rotation = 90)
plt.ylabel("Mean CTC")
plt.xlabel("Job positions")
plt.title("Top 10 job positions with atleast 5 learners with highest mean CTC")
plt.show()
```



Top 10 companies with highest mean CTC We find the Top 10 companies based on their mean CTC

[287]:
```
cols_to_include = ["company_hash", "mean_ctc", "count"]
```

[288]:
```
top_10_comp_data = comp_ctc_agg.sort_values(by = "mean_ctc", ascending =␣
↪False)[cols_to_include].head(10)
top_10_comp_data
```

[288]:

	company_hash	mean_ctc	count
30473	whmxw rgsxwo uqxcvnt rxbxnta	1.000150e+09	1
1217	aveegaxr xzntqzvnxgzvr hzxctqoxnj	2.500000e+08	1
16049	opxrr	2.000000e+08	1
32693	xfgqp ntwyzgrgsxto	2.000000e+08	1
30987	wqxwwrhmo	2.000000e+08	1
13762	nwj gzrxzt	2.000000e+08	1
8825	i wgzztin mhoxztoo ogrhnxgzo ucn rna	2.000000e+08	1
1516	axctqoxexta tztqsj ogrhnxgzo ucn rna	2.000000e+08	1
33193	xtrrxuot ntwyzgrgsxto	2.000000e+08	1
34607	xzxnxvngq uvwxexw uqxcvnt rxbxnta	2.000000e+08	1

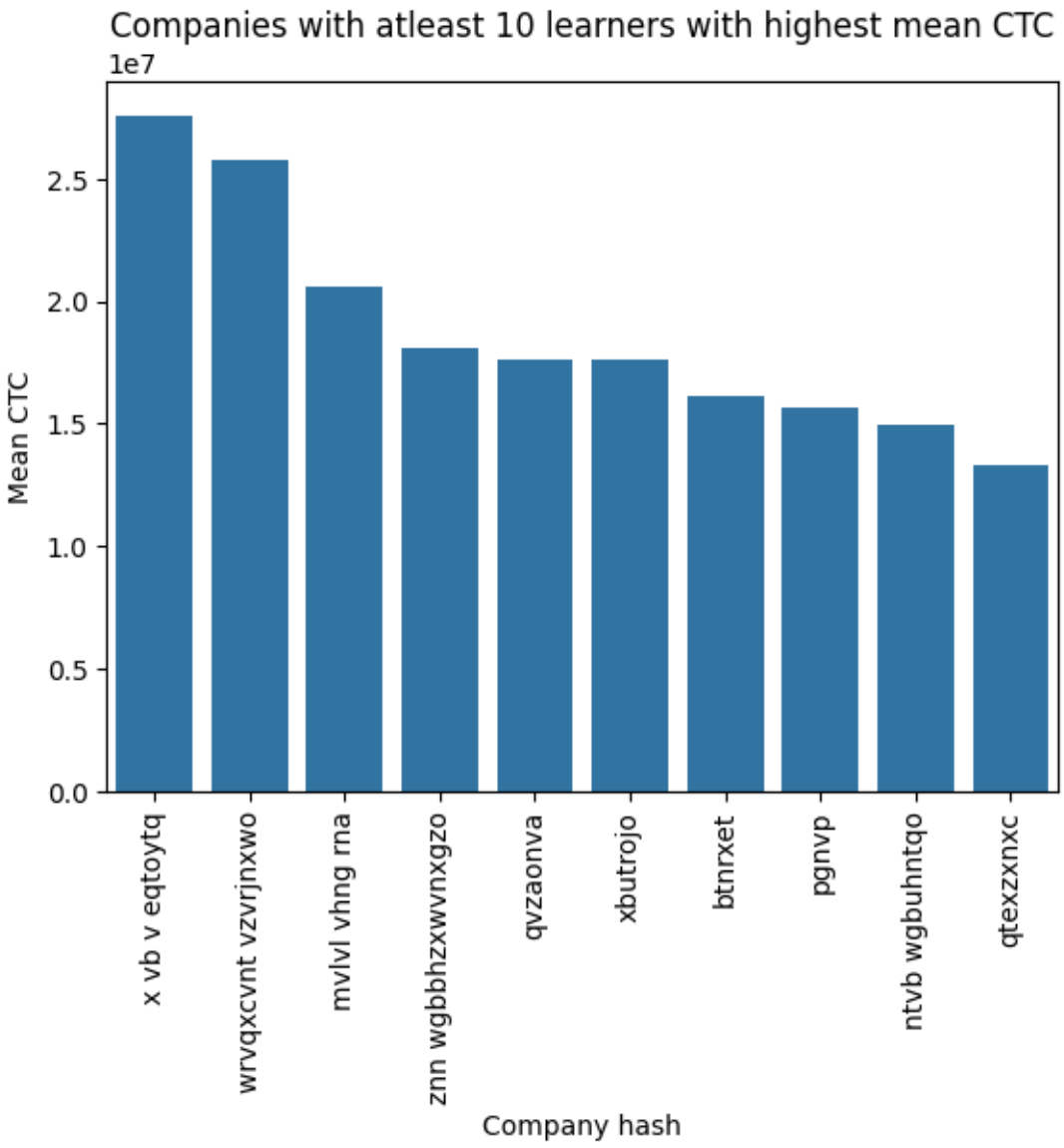
As we see, the above list consists of companies with single employees. For better trend, we include **only those companies which have at least 10 employees**. We find the top 10 such companies with the highest mean CTC

[289]:
```
top_10_comp_data = comp_ctc_agg[comp_ctc_agg["count"] > 10].sort_values(by =␣
↪"mean_ctc", ascending = False)[cols_to_include].head(10)
top_10_comp_data
```

[289]:

	company_hash	mean_ctc	count
32343	x vb v eqtoytq	2.758429e+07	21
31206	wrvqxcvnt vzvrjnxwo	2.578545e+07	11
11425	mvlvl vhng rna	2.062333e+07	15
36251	znn wgbbhzxwvnxgzo	1.811636e+07	11
20142	qvzaonva	1.763333e+07	12
32565	xbutrojo	1.758250e+07	12
2763	btnrxet	1.611613e+07	13
18180	pgnvp	1.565000e+07	12
13050	ntvb wgbuhntqo	1.492214e+07	14
19558	qtexzxnxc	1.330544e+07	16

[290]:
```
sns.barplot(x = top_10_comp_data["company_hash"], y =␣
↪top_10_comp_data["mean_ctc"])
plt.xticks(rotation = 90)
plt.ylabel("Mean CTC")
plt.xlabel("Company hash")
plt.title("Companies with atleast 10 learners with highest mean CTC")
plt.show()
```



Top 10 job positions with highest percentage of Tier-1 learners Tier-1 employees are those employees in a company who are among the top 10 highest CTC earners in the company as well as have CTC greater than average CTC of the company.

Job positions with highest % of Tier-1 learners mean that such job positions consist of high percentage of top earners. We find 10 job positions which have highest percetange of Tier-1 learners

(We only limit ourselves to those jobs which have atleast 5 learners in the dataset)

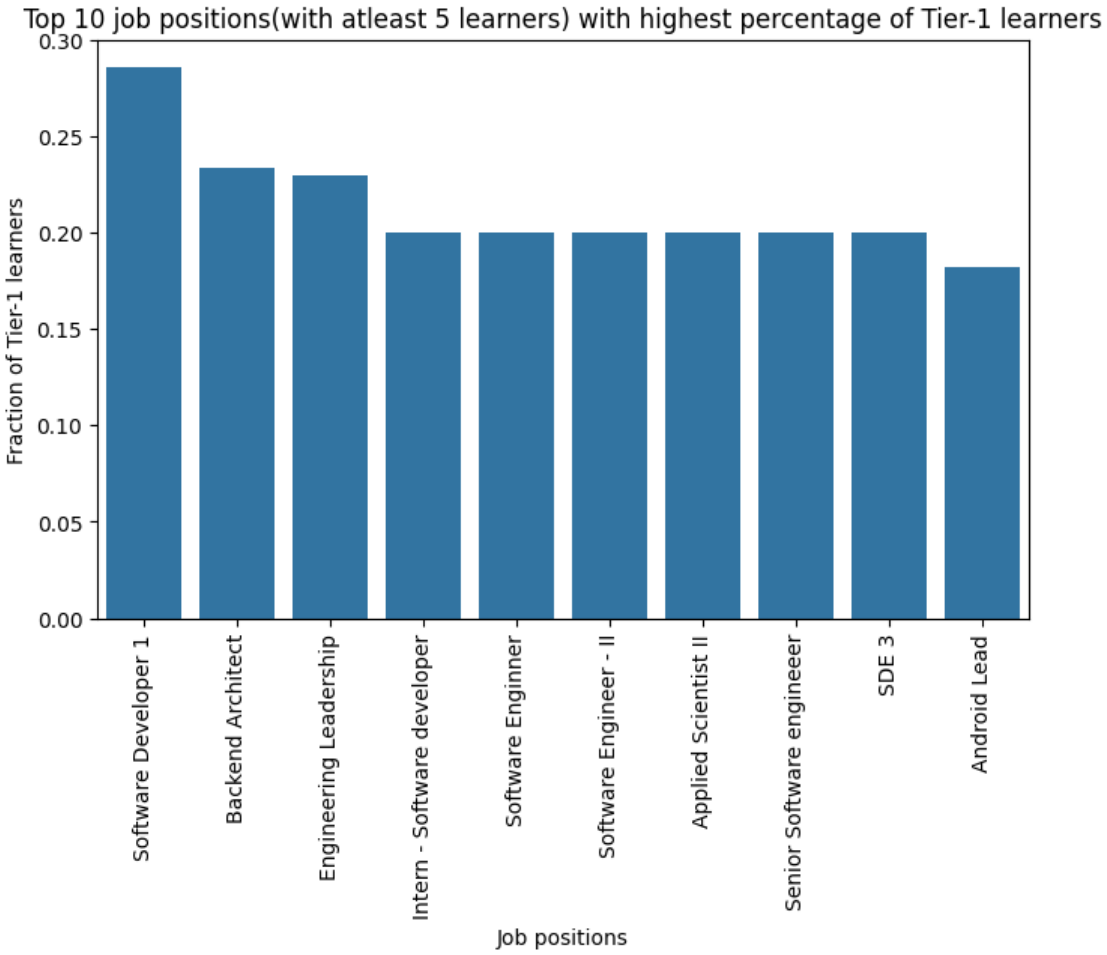
[291]:
```
job_tier_data = one_level_agg_data.groupby("job_position").agg(tier_1_count =␣
↪("Tier", lambda x: x[x == 1].count())
                                                    , job_count = ("Tier", "count"))
```

```
job_tier_data["tier_1_percent"] = job_tier_data["tier_1_count"] /␣
 ↪job_tier_data["job_count"]
job_learners_filter = job_tier_data["job_count"] >= 5
top_ten_job_tier_1 = job_tier_data[job_learners_filter].sort_values(by =␣
 ↪"tier_1_percent", ascending = False).head(10)
top_ten_job_tier_1
```

[291]:

job_position	tier_1_count	job_count	tier_1_percent
Software Developer 1	2	7	0.285714
Backend Architect	397	1701	0.233392
Engineering Leadership	1776	7729	0.229784
Intern - Software developer	1	5	0.200000
Software Enginner	1	5	0.200000
Software Engineer - II	1	5	0.200000
Applied Scientist II	1	5	0.200000
Senior Software engineeer	1	5	0.200000
SDE 3	2	10	0.200000
Android Lead	2	11	0.181818

[292]:
```
plt.figure(figsize = (8,5))
sns.barplot(x = top_ten_job_tier_1.index, y =␣
 ↪top_ten_job_tier_1["tier_1_percent"])
plt.xticks(rotation = 90)
plt.ylabel("Fraction of Tier-1 learners")
plt.xlabel("Job positions")
plt.title("Top 10 job positions(with atleast 5 learners) with highest␣
 ↪percentage of Tier-1 learners")
plt.show()
```



Top 2 job positions with highest mean CTC at various levels of experience We categorize years of job experience in three categories. - Low - Learners with less than or equal to 5 years of experience are considered in 'Low' category. Simply put, here YOE $\in (0, 5]$. - Medium - Learners with years of job experience more than 5 but less than or equal to 15 are considered in 'Medium' Category. Simply put, here YOE $\in (5, 15]$. - High - Leaners with with years of job experience greater than 15 years are considered in 'High' years of job experience categories. Simply put, here YOE > 15.

We then find the Top 2 job positions with highest mean CTC among learners in 'Low', 'Medium' and 'High' category.

Note - We will limit ourselves to only those jobs which have at least 5 learners each.

[293]:
```
data_dup_dropped["yoe_level"] = pd.cut(data_dup_dropped["years_of_experience"],
                                       bins = [0, 5, 15, np.inf], labels =␣
↪["Low", "Medium", "High"])
```

```
<ipython-input-293-0a0d2f82a4a3>:1: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dup_dropped["yoe_level"] =
pd.cut(data_dup_dropped["years_of_experience"],
```

[294]:
```
low_yoe_data = data_dup_dropped[data_dup_dropped["yoe_level"] == "Low"]
medium_yoe_data = data_dup_dropped[data_dup_dropped["yoe_level"] == "Medium"]
high_yoe_data = data_dup_dropped[data_dup_dropped["yoe_level"] == "High"]
```

[295]:
```
job_in_low = low_yoe_data.groupby("job_position").agg(mean_ctc = ("ctc",␣
 ↪"mean"), count = ("ctc", "count"))
job_in_medium = medium_yoe_data.groupby("job_position").agg(mean_ctc = ("ctc",␣
 ↪"mean"), count = ("ctc", "count"))
job_in_high = high_yoe_data.groupby("job_position").agg(mean_ctc = ("ctc",␣
 ↪"mean"), count = ("ctc", "count"))
```

[296]:
```
top_2_job_low = job_in_low[job_in_low["count"] >= 5].sort_values(by =␣
 ↪"mean_ctc", ascending  = False).head(2)
top_2_job_low
```

[296]:

job_position	mean_ctc	count
Engineering Leadership	1.569337e+07	363
Program Manager	9.800000e+06	58

[297]:
```
top_2_job_medium = job_in_medium[job_in_medium["count"] >= 5].sort_values(by =␣
 ↪"mean_ctc", ascending  = False).head(2)
top_2_job_medium
```

[297]:

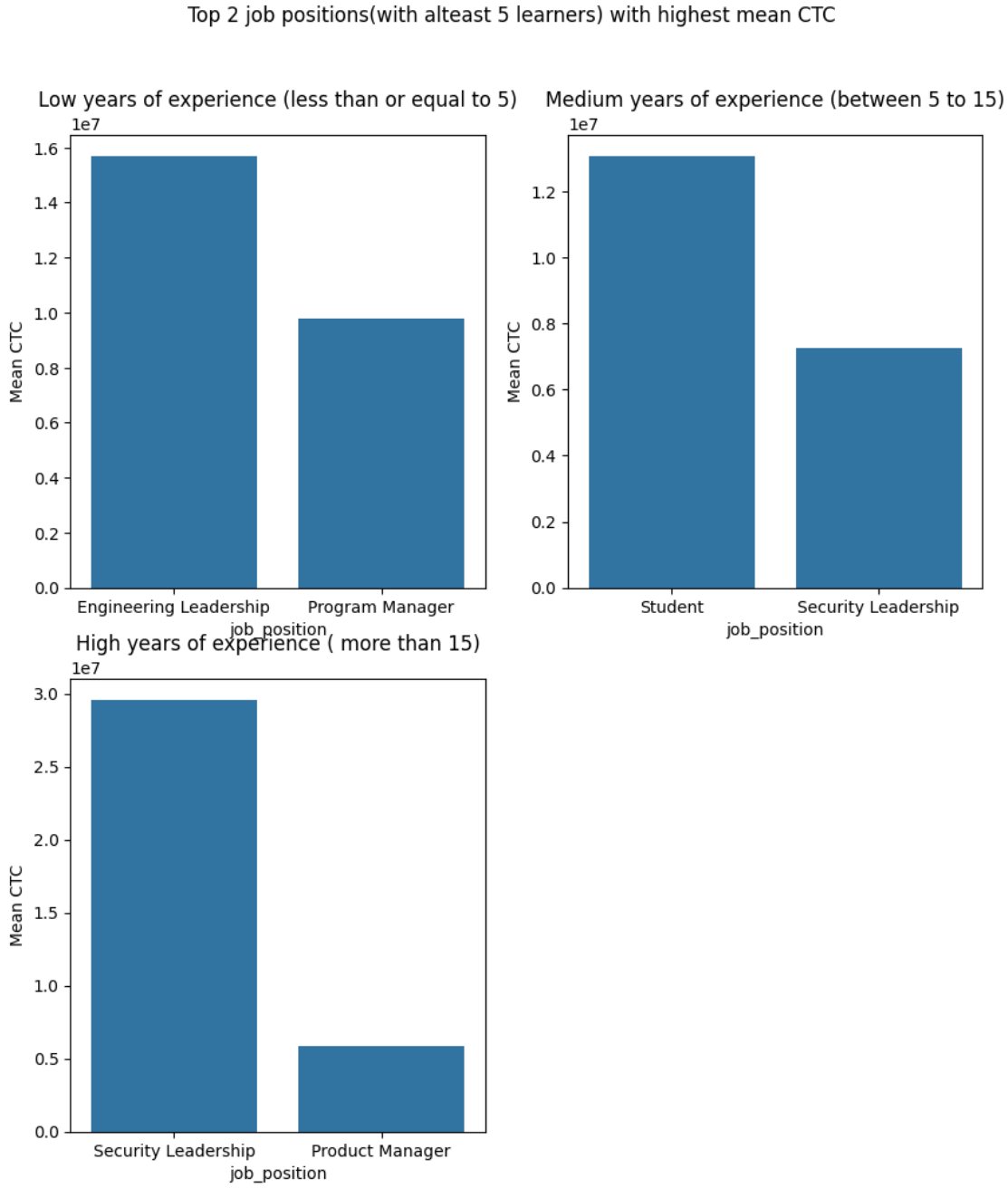
job_position	mean_ctc	count
Student	1.308125e+07	8
Security Leadership	7.248743e+06	101

[298]:
```
top_2_job_high = job_in_high[job_in_high["count"] >= 5].sort_values(by =␣
 ↪"mean_ctc", ascending  = False).head(2)
top_2_job_high
```

[298]:

job_position	mean_ctc	count
Security Leadership	2.956545e+07	22
Product Manager	5.841120e+06	196

[299]:
```
plt.figure(figsize = (10, 11))
plt.subplot(2,2,1)
sns.barplot(x = top_2_job_low.index, y = top_2_job_low["mean_ctc"])
plt.ylabel("Mean CTC")
plt.title("Low years of experience (less than or equal to 5)")

plt.subplot(2,2,2)
sns.barplot(x = top_2_job_medium.index, y = top_2_job_medium["mean_ctc"])
plt.ylabel("Mean CTC")
plt.title("Medium years of experience (between 5 to 15)")

plt.subplot(2,2,3)
sns.barplot(x = top_2_job_high.index, y = top_2_job_high["mean_ctc"])
plt.ylabel("Mean CTC")
plt.title("High years of experience ( more than 15)")

plt.suptitle("Top 2 job positions(with alteast 5 learners) with highest mean␣
 ↪CTC")
plt.show()
```



1.3.2 Unsupervised Clustering

[300]:
```
data_dup_dropped.drop(columns = ["yoe_level"], inplace = True)
```

```
<ipython-input-300-99e18ad387e0>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dup_dropped.drop(columns = ["yoe_level"], inplace = True)
```

[301]:
```
data_dup_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 196990 entries, 0 to 205842
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   company_hash         196990 non-null  object
 1   email_hash           196990 non-null  object
 2   orgyear              196990 non-null  float64
 3   ctc                  196990 non-null  float64
 4   job_position         196990 non-null  object
 5   ctc_updated_year     196990 non-null  float64
 6   years_of_experience  196990 non-null  float64
dtypes: float64(4), object(3)
memory usage: 12.0+ MB
```

We ignore "email_hash" for clustering. We also drop "orgyear" since we already have a similar column "years_of_experience".

[302]:
```
data_to_cluster = data_dup_dropped.drop(columns = ["email_hash", "orgyear"])
```

We will make three separate datasets and run clustering algorithms on each to see how categorical features like "company_hash" and "job_position" impact clustering.

- First dataset will contain one-hot encoded "company_hash", "job_position" and the original numerical variables "ctc", "ctc_updated_year" and "years_of_experience".
- Second dataset will contain one-hot encoded "job_position" and the original numerical variables "ctc", "ctc_updated_year" and "years_of_experience".
- Third dataset will contain only the original numerical variables "ctc", "ctc_updated_year" and "years_of_experience".

[303]:
```
data_num_job_company = data_to_cluster.copy()
```

[304]:
```
data_num_only_job_pos = data_to_cluster.drop(columns = "company_hash")
```

[305]:
```
data_numerical = data_to_cluster.select_dtypes(exclude = "object")
```

[306]:
```
# Standard scaling the original numerical variables
scaler_num = StandardScaler()
X_only_num_scaled = scaler_num.fit_transform(data_numerical)
```

One hot encoding the categorical variables. Note that many of the categories within a variable occur only once. If we were to make binary feature for every such infrequent category then the

number of dimensions will become very large.

So, we aggregate the infrequent categories(that is, the categories which occur only once each in a feature) into a single binary feature.

[307]:
```
# One hot encoding "company_hash" and "job_position"
enc_job_comp = OneHotEncoder(min_frequency=2)
X_job_comp = enc_job_comp.fit_transform(data_num_job_company[["company_hash",␣
↪"job_position"]])
X_job_comp
```

[307]:
```
<196990x13235 sparse matrix of type '<class 'numpy.float64'>'
        with 393980 stored elements in Compressed Sparse Row format>
```

[308]:
```
# One hot encoding "job_position"
enc_job = OneHotEncoder(min_frequency=2)
X_job = enc_job.fit_transform(data_num_only_job_pos[["job_position"]])
X_job
```

[308]:
```
<196990x257 sparse matrix of type '<class 'numpy.float64'>'
        with 196990 stored elements in Compressed Sparse Row format>
```

[309]:
```
# merging the one-hot encoded data with numerical data
X_job_comp_scaled = hstack([X_job_comp, csr_matrix(X_only_num_scaled)])
X_job_scaled = hstack([X_job, csr_matrix(X_only_num_scaled)])
```

[310]:
```
X_job_comp_scaled
```

[310]:
```
<196990x13238 sparse matrix of type '<class 'numpy.float64'>'
        with 984950 stored elements in Compressed Sparse Row format>
```

[311]:
```
X_job_scaled
```

[311]:
```
<196990x260 sparse matrix of type '<class 'numpy.float64'>'
        with 787960 stored elements in Compressed Sparse Row format>
```

K-Means Applying K-Means Clustering

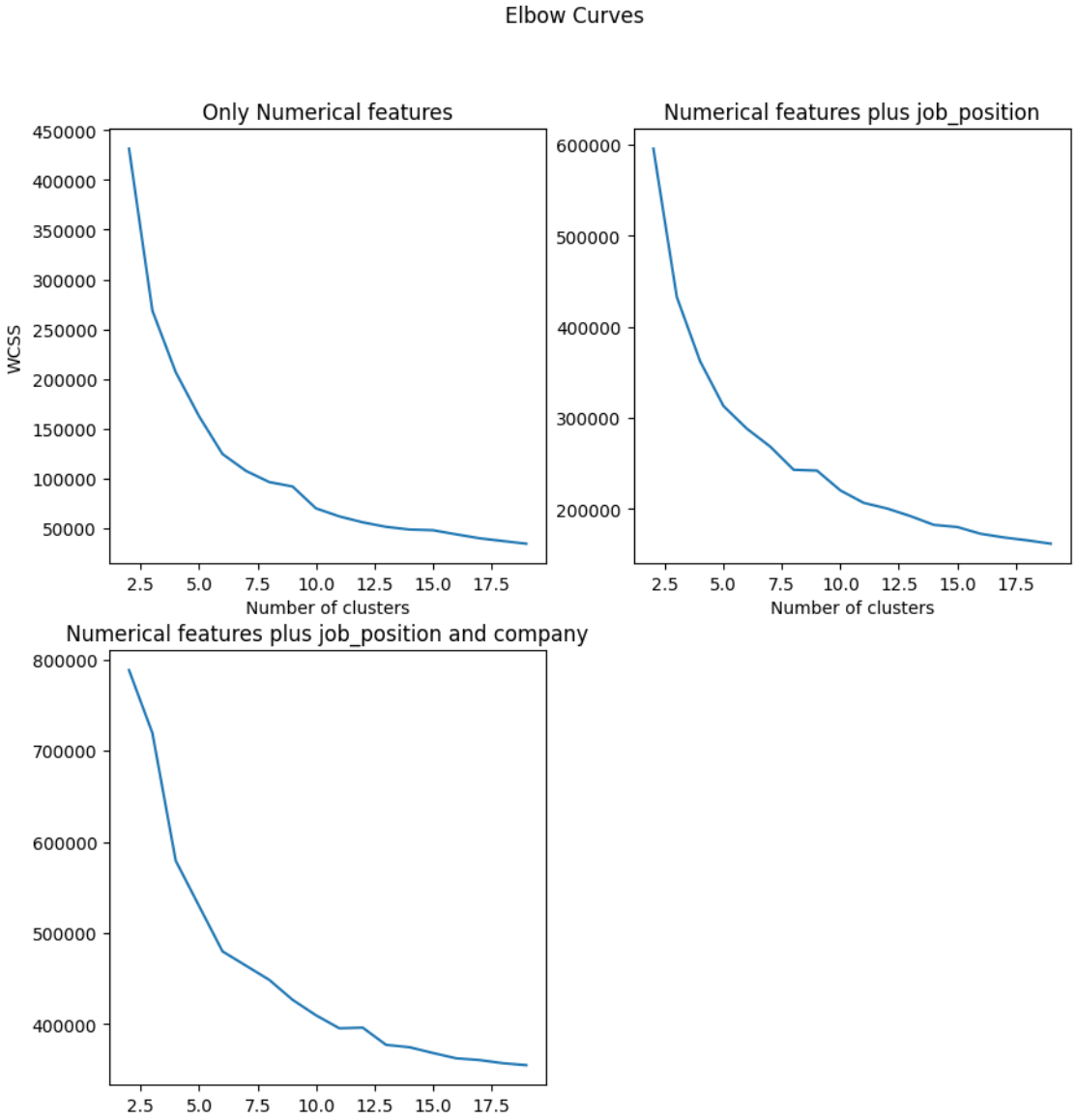
[312]:
```
def apply_kmeans(data, k = 5):
  kmeans = KMeans(n_clusters = k, random_state = 1)
  kmeans.fit(data)
  return kmeans
```

[313]:
```
def calculate_wcss(data):
  wcss_scores = []
  for i in range(2, 20):
    kmeans = apply_kmeans(data, i)
    wcss_scores.append((i, kmeans.inertia_))
```

```
    return wcss_scores
```

[314]:
```
wcss_only_num = calculate_wcss(X_only_num_scaled)
wcss_num_job = calculate_wcss(X_job_scaled)
wcss_num_job_comp = calculate_wcss(X_job_comp_scaled)
```

[315]:
```
plt.figure(figsize = (10, 10))
plt.subplot(2,2,1)
sns.lineplot(x = [x[0] for x in wcss_only_num], y = [x[1] for x in␣
 ↪wcss_only_num])
plt.title("Only Numerical features")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.subplot(2,2,2)
sns.lineplot(x = [x[0] for x in wcss_num_job], y = [x[1] for x in wcss_num_job])
plt.title("Numerical features plus job_position")
plt.xlabel("Number of clusters")
plt.subplot(2,2,3)
sns.lineplot(x = [x[0] for x in wcss_num_job_comp], y = [x[1] for x in␣
 ↪wcss_num_job_comp])
plt.title("Numerical features plus job_position and company")
plt.suptitle("Elbow Curves")
plt.show()
```



After looking at the Elbow curves,

- For the first dataset with "company_hash", "job_position", "ctc", "ctc_updated_year" and "years_of_experience", 8 clusters seem to be ideal.
- For the second dataset with "job_position", "ctc", "ctc_updated_year" and "years_of_experience", 6 clusters seem to be ideal.
- For the third dataset with only the original numerical variables "ctc", "ctc_updated_year" and "years_of_experience", 6 clusters seem to be ideal.

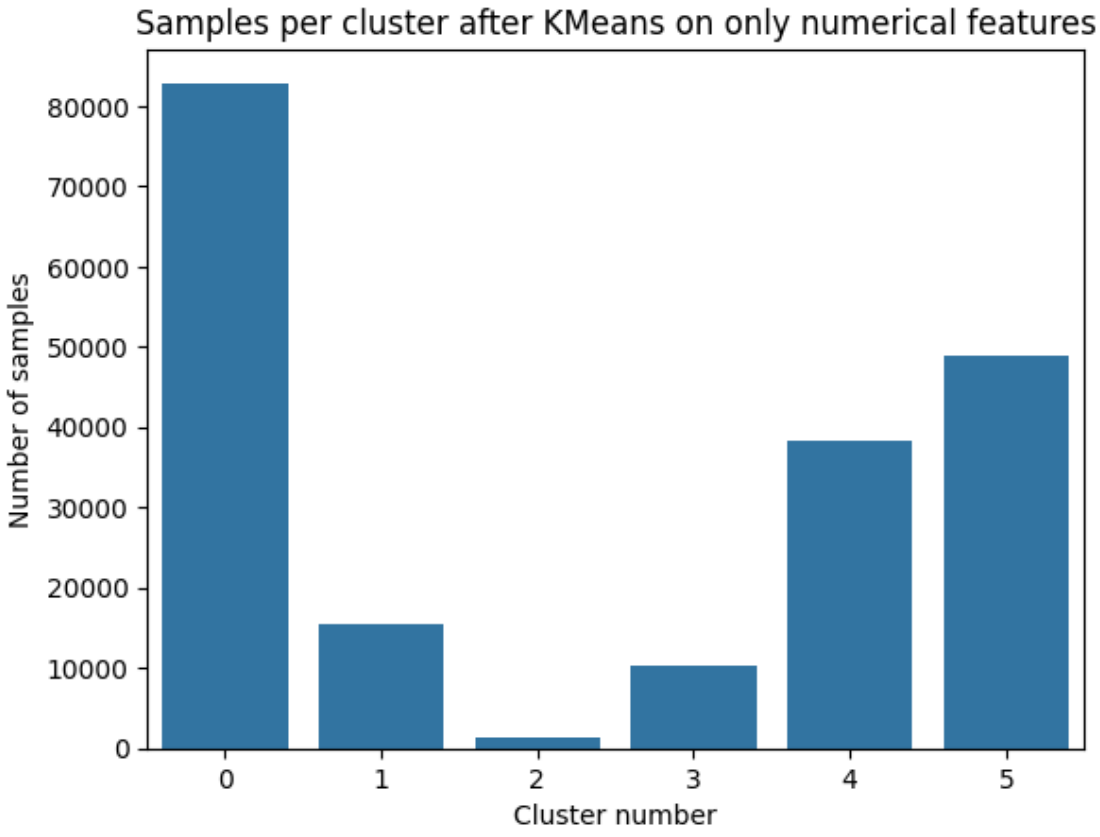
We take the majority vote and go with 6 clusters for each of the dataset.

[316]:
```
num_clusters = 6
```

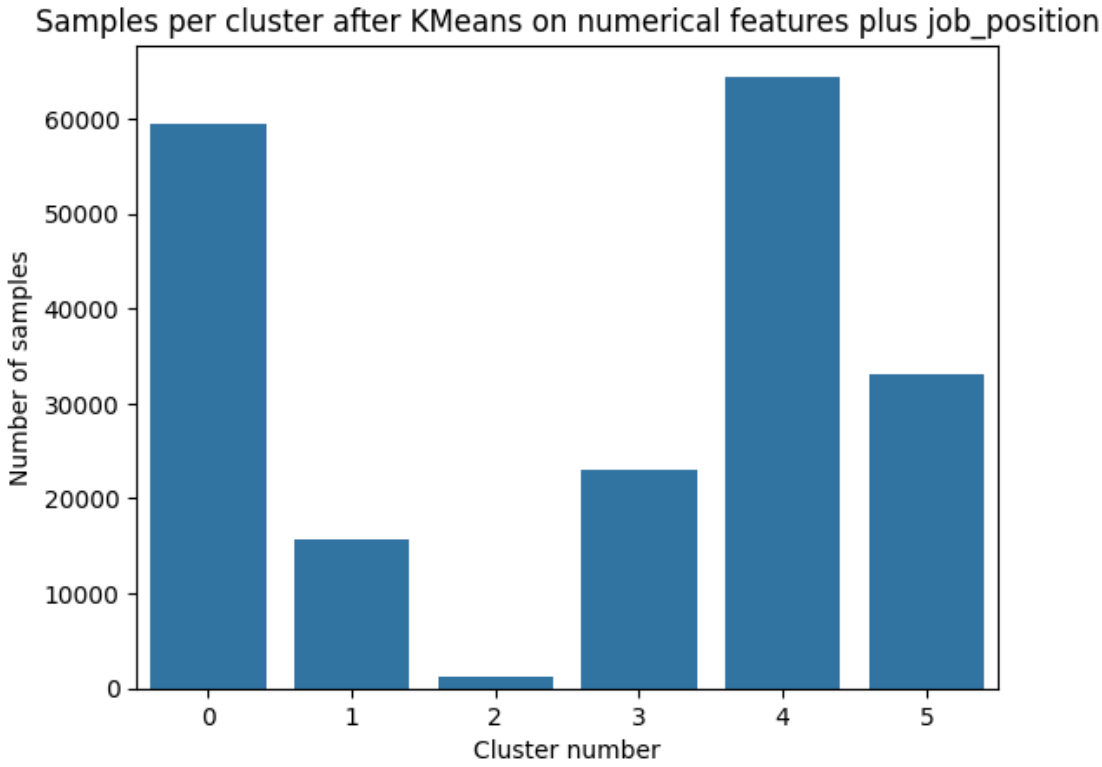
[317]:
```
kmeans_only_num = apply_kmeans(X_only_num_scaled, num_clusters)
labels_only_num = kmeans_only_num.predict(X_only_num_scaled)

kmeans_num_job = apply_kmeans(X_job_scaled, num_clusters)
labels_num_job = kmeans_num_job.predict(X_job_scaled)

kmeans_num_job_comp = apply_kmeans(X_job_comp_scaled, num_clusters)
labels_num_job_comp = kmeans_num_job_comp.predict(X_job_comp_scaled)
```

[318]:
```
data_to_cluster["label_only_num"] = labels_only_num
data_to_cluster["label_num_job"] = labels_num_job
data_to_cluster["label_num_job_comp"] = labels_num_job_comp
```

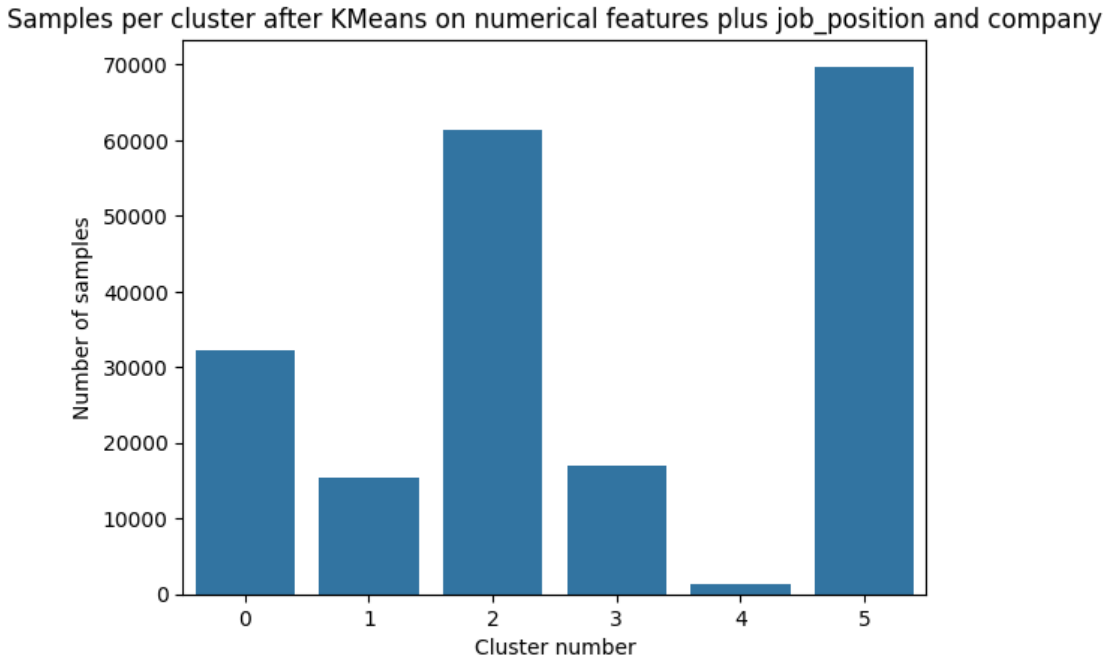
[319]:
```
vc_only_num = data_to_cluster["label_only_num"].value_counts()
sns.barplot(x = vc_only_num.index, y = vc_only_num.values)
plt.title("Samples per cluster after KMeans on only numerical features")
plt.xlabel("Cluster number")
plt.ylabel("Number of samples")
plt.show()
```



[320]:
```
vc_num_job = data_to_cluster["label_num_job"].value_counts()
sns.barplot(x = vc_num_job.index, y = vc_num_job.values)
plt.title("Samples per cluster after KMeans on numerical features plus␣
 ↪job_position")
plt.xlabel("Cluster number")
plt.ylabel("Number of samples")
plt.show()
```



[321]:
```
vc_num_job_comp = data_to_cluster["label_num_job_comp"].value_counts()
sns.barplot(x = vc_num_job_comp.index, y = vc_num_job_comp.values)
plt.title("Samples per cluster after KMeans on numerical features plus␣
 ↪job_position and company")
plt.xlabel("Cluster number")
plt.ylabel("Number of samples")
plt.show()
```



Calculating the mean and median CTC, mean years of experience, mean CTC_updated_year using the KMeans cluster labels done on first dataset (the dataset having features "company_hash", "job_position", "ctc", "ctc_updated_year", "years_of_experience")

[322]:
```
cluster_grouped = data_to_cluster.groupby("label_num_job_comp")
cluster_grouped_agg = cluster_grouped.agg(mean_ctc = ("ctc", "mean"),␣
↪median_ctc = ("ctc", "median")
                                          , mean_yoe = ("years_of_experience",␣
↪"mean")
                                          , mean_ctc_updated_year =␣
↪("ctc_updated_year", "mean")
                                          , modal_job = ("job_position", lambda x:␣
↪x.mode()[0]))
cluster_grouped_agg
```

[322]:

label_num_job_comp	mean_ctc	median_ctc	mean_yoe	mean_ctc_updated_year	modal_job
0	1.769136e+06	1200000.0	10.971210		
1	1.404986e+06	1030000.0	10.564697		
2	1.268084e+06	900000.0	8.409749		
3	2.450772e+06	2000000.0	18.459553		
4	1.349764e+08	100000000.0	8.681126		
5	1.095202e+06	700000.0	5.851990		

```
label_num_job_comp
0                              2020.575125        Backend Engineer
1                              2016.274866        Backend Engineer
2                              2018.916447        Backend Engineer
3                              2019.216402  Engineering Leadership
4                              2020.059361                   Other
5                              2020.545044        Backend Engineer
```

Hierarchical Clustering We perform agglomerative clustering using ward linkage below. Since this clustering takes a lot of time, we only run this algorithm on a random subset of 5000 rows of the dataset.

[323]:
```
# For dendogram, we take a random sample of 5000 rows instead of all the rows
num_rand = 5000
rand_ind = np.random.choice(X_only_num_scaled.shape[0], num_rand, replace =␣
↪False)
X_only_num_scaled_rand = X_only_num_scaled[rand_ind, :]
```

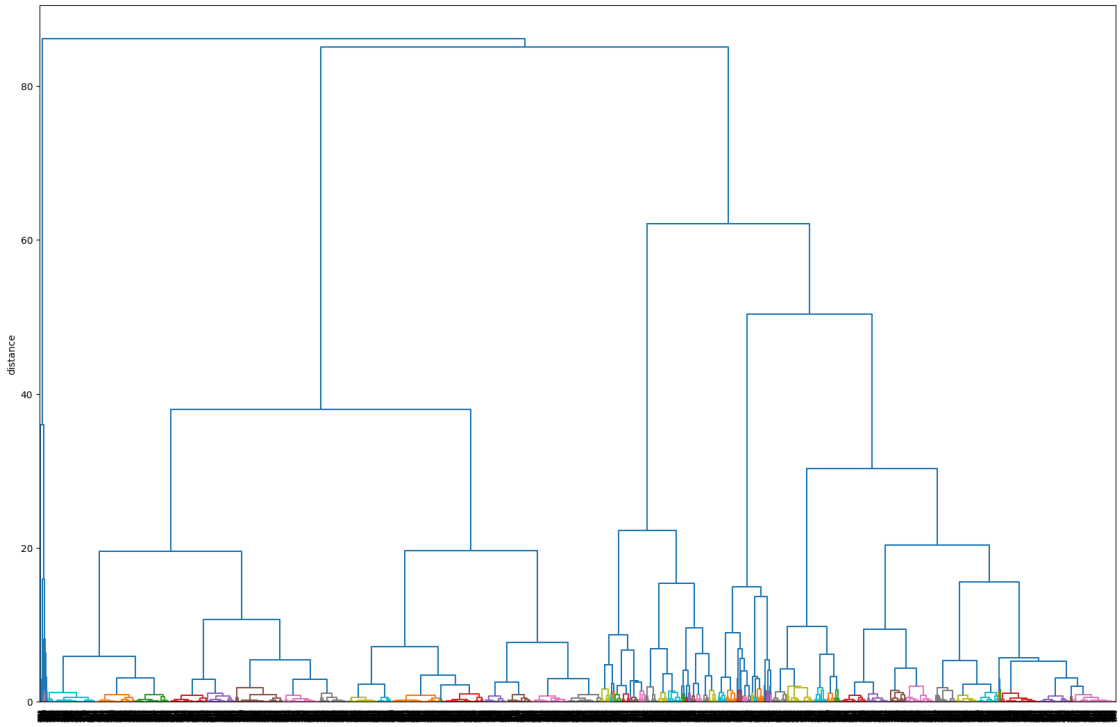
[324]:
```
Z = sch.linkage(X_only_num_scaled_rand, method = "ward")
```

[325]:
```
Z.shape
```

[325]:
```
(4999, 4)
```

[326]:
```
fig, ax = plt.subplots(figsize=(20, 13))
sch.dendrogram(Z, ax=ax, color_threshold=2)
plt.xticks(rotation=90)
ax.set_ylabel('distance')
```

[326]:
```
Text(0, 0.5, 'distance')
```



After seeing the dendogram on the random subset of data, 5 clusters seem to fine.

[327]:
```
agg_clustering = AgglomerativeClustering(n_clusters = 5)
```

[328]:
```
X_job_comp_scaled_rand = X_job_comp_scaled[rand_ind, :]
data_to_cluster_rand = data_to_cluster.iloc[rand_ind, :]
```

[329]:
```
X_job_comp_scaled_rand
```

[329]:
```
<5000x13238 sparse matrix of type '<class 'numpy.float64'>'
        with 25000 stored elements in Compressed Sparse Row format>
```

[330]:
```
labels = agg_clustering.fit_predict(X_job_comp_scaled_rand.toarray())
```

[331]:
```
data_to_cluster_rand["label"] = labels
```

```
<ipython-input-331-dc78f27ab9bc>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_to_cluster_rand["label"] = labels
```

[332]:
```
cluster_rand_hier_grouped = data_to_cluster_rand.groupby("label")
cluster_rand_hier_agg = cluster_rand_hier_grouped.agg(mean_ctc = ("ctc",␣
↪"mean"), median_ctc = ("ctc", "median")
                                        , mean_yoe = ("years_of_experience",␣
↪"mean")
                                        , mean_ctc_updated_year =␣
↪("ctc_updated_year", "mean")
                                        , modal_job = ("job_position", lambda x:␣
↪x.mode()[0]))
cluster_rand_hier_agg
```

[332]:

label	mean_ctc	median_ctc	mean_yoe	mean_ctc_updated_year	modal_job
0	1.165313e+08	100000000.0	8.615385	2019.846154	Other
1	1.069479e+06	760000.0	6.793977	2020.050650	Backend Engineer
2	2.515091e+06	2150000.0	18.491525	2019.254237	Engineering Leadership
3	1.317846e+06	1000000.0	10.208955	2016.350746	Backend Engineer
4	1.583333e+06	1200000.0	10.519608	2019.629902	Backend Engineer

Insights from Clustering

1. We went with $K = 6$ clusters in KMeans and $K = 5$ in Agglomerative Clustering based on Elbow method and dendogram respectively. The agglomerative clustering is run on random sample of 5000 rows as its algorithm has high time complexity.
2. The characteristics of clusters identified in KMeans are very similar to those identified in hierarchical clustering. We describe some characteristics of clusters found by KMeans below

- The most numerous cluster (cluster number 5) has close to 70,000 samples. It is characterized by lowest mean CTC and lowest years of experience among all the clusters. The mean CTC_updated_year for this cluster is also recent compared to the rest. This cluster likely represents the *rookies*, that is those learners who have started their professional careers relatively recently.
- The smallest cluster by size (cluster number 4) contains 1,314 samples. It is characterized by very high mean CTC (this cluster thas the highest mean CTC) but relatively low-medium mean years_of_experience. This is the *Outlier CTC* cluster. More investigation is needed into this cluster to find out if these are outliers or novelties or errors in data entry.
- Cluster number 3, which has roughly 17000 samples, is likely the *Senior Role* Cluster. It is characterized by high CTC (second highest mean CTC among all clusters) and the highest

mean years_of_experience among all clusters. The most frequent job position in the cluster is 'Engineering Leadership'.

- Cluster number 1, which has roughly 15000 samples, is likely the *Professionally Stuck* cluster. This cluster has the earliest mean ctc_updated_year among all the clusters. This means that learners is this cluster haven't had promotions or increments since a long time and have plateued in their positions and companies.

1.4 Recommendations

- Working professionals with characteristics similar to the ones in *Professionally Stuck* cluster have the most need for upskilling. To acquire customers of this category
- Scaler can make video ads showing data and individual testimonials of successfully placed learners of this category.
- Most frequent companies and job positions in this cluster can be identified. Scaler Customer Acquistion team can then find learners at these companies and at these job positions on LinkedIn and contact them.
- For helping the already enrolled learners in *Professionally Stuck* cluster, Scaler can organize master sessions featuring one or two hour sessions from already placed scalerites of this category.
- Scaler can start a (Gen) AI + MBA kind of mixed program which will cater to business professionals want to get into AI-based tech management role.
- The *Outlier CTC* cluster need more investigation. It must be found out if the CTC entries in that cluster are genuine signals or just due to errors in data entry.
- Scaler can tie up with the companies having high mean CTC for placement.