Business Case: Scaler - Clustering

Problem Statement

Scaler is an online tech-versity offering intensive computer science & Data Science courses through live classes delivered by tech leaders and subject matter experts. The meticulously structured program enhances the skills of software professionals by offering a modern curriculum with exposure to the latest technologies. It is a product by InterviewBit.

You are working as a data scientist with the analytics vertical of Scaler, focused on profiling the best companies and job positions to work for from the Scaler database. You are provided with the information for a segment of learners and tasked to cluster them on the basis of their job profile, company, and other features. Ideally, these clusters should have similar characteristics.

Data Dictionary:

- 'Unnamed 0'- Index of the dataset
- Email_hash- Anonymised Personal Identifiable Information (PII)
- Company_hash- Current employer of the learner
- orgyear- Employment start date
- CTC- Current CTC
- Job_position- Job profile in the company
- CTC_updated_year: Year in which CTC got updated (Yearly increments, Promotions)

Concept Used:

Manual Clustering Unsupervised Clustering - K- means, Hierarchical Clustering

Additional views

We will begin by importing data and performing basic data exploration, cleaning, imputation, and data aggregation. We will then create a few new features to allow us to group learners into meaningful logical groups (manual clustering). We will attempt to answer a few business queries using the derived features, and also perform uni-variate/bi-variate analyses on the features. In the last section, we will perform unsupervised clustering using KMeans and hierarchical clustering algorithms. We will begin by pre-processing data to make is suitable for clustering, checking data-sets clustering tendency, using elbow method to determine number of clusters, and finally we will evaluate and visualize clusters. At the end, we will attempt to define characteristics of the identified clusters and report business recommendations.

Solution

Basic Data Exploration and Pre-processing

In [1]:
```
import re
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.impute import KNNImputer
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

from scipy.stats import percentileofscore
from pyclustertend import hopkins

#set seaborn theme
sns.set_theme(style="whitegrid", palette="deep")
```

In [3]:
```
df = pd.read_csv('scaler_clustering.csv')
df.drop('Unnamed: 0', axis=1, inplace=True)
df.head()
```

Out[3]:

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

In [4]:
```
origdf = df.copy()
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   company_hash      205799 non-null  object
 1   email_hash        205843 non-null  object
```

```
 2   orgyear           205757 non-null  float64
 3   ctc               205843 non-null  int64
 4   job_position      153281 non-null  object
 5   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(1), object(3)
memory usage: 9.4+ MB
```

In [6]:
```
#remove duplicate rows if any
df = df.drop_duplicates()
df.shape
```

Out[6]:
```
(205810, 6)
```

Observations: Total 33 duplicate records removed.

In [7]:
```
#check missing values
df.isna().agg(['sum', 'mean']).rename({'sum': 'count', 'mean': 'fraction of total'}).T
```

Out[7]:

	count	fraction of total
company_hash	44.0	0.000214
email_hash	0.0	0.000000
orgyear	86.0	0.000418
ctc	0.0	0.000000
job_position	52547.0	0.255318
ctc_updated_year	0.0	0.000000

In [8]:
```
df.describe(include='O')
```

Out[8]:

	company_hash	email_hash	job_position
count	205766	205810	153263
unique	37299	153443	1017
top	nvnv wgzohrnvzwj otqcxwto	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	Backend Engineer
freq	8337	10	43546

In [8]:
```
df.describe()
```

Out[8]:

	orgyear	ctc	ctc_updated_year
count	205724.000000	2.058100e+05	205810.000000
mean	2014.882284	2.271854e+06	2019.628279
std	63.576199	1.180185e+07	1.325188
min	0.000000	2.000000e+00	2015.000000

	orgyear	ctc	ctc_updated_year
25%	2013.000000	5.300000e+05	2019.000000
50%	2016.000000	9.500000e+05	2020.000000
75%	2018.000000	1.700000e+06	2021.000000
max	20165.000000	1.000150e+09	2021.000000

In [9]:
```
np.unique(df[~df['orgyear'].isna()]['orgyear'].astype('int'))
```

Out[9]:
```
array([    0,     1,     2,     3,     4,     5,     6,    38,    83,
          91,   200,   201,   206,   208,   209,  1900,  1970,  1971,
        1972,  1973,  1976,  1977,  1979,  1981,  1982,  1984,  1985,
        1986,  1987,  1988,  1989,  1990,  1991,  1992,  1993,  1994,
        1995,  1996,  1997,  1998,  1999,  2000,  2001,  2002,  2003,
        2004,  2005,  2006,  2007,  2008,  2009,  2010,  2011,  2012,
        2013,  2014,  2015,  2016,  2017,  2018,  2019,  2020,  2021,
        2022,  2023,  2024,  2025,  2026,  2027,  2028,  2029,  2031,
        2101,  2106,  2107,  2204, 20165])
```

Observations:

- We observe that ctc_updated_year ranges from 2015 to 2021. Since the highest ctc updated year is 2021, we can take 2022 as the curryear for all calculations.
- orgyear value seems more inconsistent. it has Nan values, zero values, and some values much higher than possible year values. Needs more examination.

checking consistency of 'ctc_updated_year'

In [2]:
```
curryear = 2022
```

In [9]:
```
#convert ctc_updated_year to int
df['ctc_updated_year'].astype('int')

df['ctc_updated_year'].value_counts()
```

Out[9]:
```
2019.0    68665
2021.0    64975
2020.0    49435
2017.0     7561
2018.0     6746
2016.0     5501
2015.0     2927
Name: ctc_updated_year, dtype: int64
```

observations: ctc_updated_year value for all records appear in reasonable range. We assume that ctc_updated_year value for each record is accurate. Next, we check the value of 'orgyear'.

checking consistency of 'orgyear'

We can validate orgyear using the combination of the following rules.

1. Considering 50-52 years as the maximum career span (or at-least likely work experience range of Scaler learners), all records with 'orgyear' < 1971 are potential suspects.

2. It's reasonable to expect 'orgyear' to be less than or equal to 'ctc_updated_year'. All records where (orgyear > cct_updated_year) are error suspects. In the absence any additional data about the nature of the errors, ctc_updated_year provides the best estimate for orgyear. We will simply set orgyear to their ctc_updated_year value.

In [10]:
```
# 1. For all records with orgyear < 1970, set their orgyear to Nan and impute with KNN
mask = df['orgyear'] < 1970
print(df[mask]['orgyear'].value_counts())

#set the orgyear value to Nan for the records above. We will impute them with knn later
df.loc[mask, 'orgyear'] = np.NAN
```

```
0.0       17
3.0        6
91.0       3
2.0        3
5.0        2
1.0        2
6.0        2
208.0      1
209.0      1
206.0      1
4.0        1
83.0       1
38.0       1
1900.0     1
201.0      1
200.0      1
Name: orgyear, dtype: int64
```

In [11]:
```
#find all records where orgyear > ctc_updated_year
mask = df['orgyear'] > df['ctc_updated_year']
print(f'impacted records : {df[mask].shape[0]}')

#set orgyear to ctc_updatged_year (which is our best estimate for correct orgyear)
df.loc[mask, 'orgyear'] = df.loc[mask, 'ctc_updated_year']
```

```
impacted records : 8840
```

In [12]:
```
df.agg({'orgyear': ['min', 'max']})
```

Out[12]:

	orgyear
min	1970.0
max	2021.0

Analyze email_hash, company_hash, and job_position

In [13]:
```
#check unique values
for col in ['email_hash', 'company_hash', 'job_position']:
    print(f'unique no of {col}: {df[col].nunique()}')
```

```
unique no of email_hash: 153443
unique no of company_hash: 37299
unique no of job_position: 1017
```

Observations:

1. All **email_hash** values appear to share the same formatting and appear to be system generated 64 char values. So we can assume them to be consistent and error free.

2. **company_hash** values, however, appear with inconsistent formatting and length. It is possible that it was generated by masking the actual names using a specific algorithm. Similarly, **job_position**, may have been manually entered by learners (or sourced from disparate data sources) and therefore more likely be inconsistent and error-prone.

In this case-study, we will sanitize all 'job_positions' and 'company_hash' columns by converting to lower case, trimming them, and removing any non-alphanumeric characters.

Sanitize job_positions

We can convert job_positions to lower case, trim it, and remove any non-alpha-numeric characters (including whitespace) from it in order to make it more consistent.

In [14]:
```
def remove_sp_char(x):
    return re.sub(r'[^a-zA-Z0-9]*', '', x) if (type(x) is str) else x

def sanitize_string_col(df, col):
    #convert to lower case and trim
    df[col] = df[col].str.lower().str.strip()

    #remove any special characters and also space
    df[col] = df[col].transform(remove_sp_char)

sanitize_string_col(df, 'job_position')
print(f"no of unique job_positions: {df['job_position'].nunique()}")
```

```
no of unique job_positions: 873
```

In [15]:
```
jpvc = df['job_position'].value_counts()
jpvc
```

Out[15]:
```
backendengineer                 43546
fullstackengineer               25982
other                           18072
frontendengineer                10418
engineeringleadership            6870
                                ...
riskinvestigator                    1
x                                   1
machinelearningengineerintern       1
thirdpartyprovider                  1
```

```
azuredatafactory                    1
Name: job_position, Length: 873, dtype: int64
```

Observation: On careful examination, we notice that most of the job_positions have very few associated learners. To reduce noise, we only keep those job positions which have at-least 50 learners associated to it. For the remaining low frequency job positions, we will set them to 'NAN' and impute later with KNN.

In [16]:
```
#set non-frequent job_positions to NAN
whitelisted_jp = jpvc[jpvc > 50].index.values
df.loc[~df['job_position'].isin(whitelisted_jp), 'job_position'] = np.NAN
print(f'no of unique job_position : {df["job_position"].nunique()}')
```

```
no of unique job_position : 25
```

Sanitize company_hash

In [17]:
```
sanitize_string_col(df, 'company_hash')
df['company_hash'].nunique()
```

Out[17]:
```
36584
```

Removing inconsistent records

We expect each learner to be employed with **atmost one employer at a given time**. Thus, if we take the combination of *{email_hash, company_hash, orgyear, ctc_updated_year}*, we expect unique records. However, there are several instances in the dataset, as shown below, where there are multiple records for the given combination, often differing just in job-positions values. This is clearly inconsistent. In such cases, we will select only one record (one with highest ctc) per given combination and discard the remaining records.

In [18]:
```
df.groupby(['company_hash','email_hash', 'orgyear', 'ctc_updated_year'])['job_position'
```

Out[18]:
```
1    129104
0     26006
2      9138
3       823
4        61
5         5
6         2
Name: job_position, dtype: int64
```

In [19]:
```
df2 = df.copy()
#add temp column to keep track of original index of rows to be deleted.
df['row_index'] = df.index.values

rows_with_no_missing_values = np.all(~df[['email_hash', 'company_hash', 'orgyear', 'ctc

dellist = []
def markfordel(d):
    #for each group, we only wish to keep one row.
    #We keep the first row with non empty job position or else the first row
    if(d.shape[0] > 1):
```

```
        row_to_keep = d['job_position'].first_valid_index()

        if(type(row_to_keep) != int):
            row_to_keep = 0

        for i in range(1, d.shape[0]):
            if(i != row_to_keep):
                dellist.append(d.iloc[i]['row_index'])

df[rows_with_no_missing_values].groupby(['email_hash', 'company_hash', 'orgyear', 'ctc_
df.drop(labels='row_index', axis=1, inplace=True)
```

In [20]:
```
#drop inconsistent rows.
df.drop(index=dellist, inplace=True)
```

In [21]:
```
#reset index
df.reset_index(drop=True, inplace=True)
```

In [22]:
```
print(f'No of inconsistent rows removed = {df2.shape[0] - df.shape[0]}')
```

```
No of inconsistent rows removed = 40497
```

'email_hash' frequency analysis

Imp Note: Since we are doing this analysis after removing a lot of inconsistent rows, we are likely to see less noisy data. If we perform the same analysis on the original dataset, we will see different results.

In [23]:
```
def show_email_frequency(df, grpbycols=['email_hash']):
    emailgroups = df.groupby(grpbycols)['email_hash'].count()
    emailgroups = pd.merge(emailgroups.value_counts(), emailgroups.value_counts(normali
    emailgroups.columns = ['email freq', 'count', 'count %']
    return emailgroups
```

In [24]:
```
#overall email_hash frequency distribution
show_email_frequency(df, ['email_hash'])
```

Out[24]:

	email freq	count	count %
0	1	141575	0.922655
1	2	11866	0.077332
2	3	2	0.000013

Observations: We see that around 92.2% of the email_hash occur once. Around 7.7% email_hash occur twice. Only 12 email_has values occur more than 2 times.

Next we perform similar analysis for combinations {email_hash, company_hash}.

In [25]:
```
# email_hash frequency distribution by {email_hash, company_hash}
show_email_frequency(df, ['email_hash', 'company_hash'])
```

Out[25]:

	email freq	count	count %
0	1	155207	0.968603
1	2	5031	0.031397

Observations: When we consider the combination {email_hash, company_hash}, around 96.8% email_has occur once, and 3.1% email_hash occur twice. Only 7 email_hash values occur 3 or 4 times.

This means that overall 96.8% of Scaler learners have been employed with single company (based on their data shared with Scaler). This implies that a large fraction of Scaler learners are professionals in their early career years, students who have just got placements, or with a lesser likelihood, experienced professionals who have stayed loyal to their first company.

We can check additional combinations as shown below.

In [26]:
```
# email_hash frequency distribution by {email_hash, company_hash, job_position}
show_email_frequency(df, ['email_hash', 'company_hash', 'job_position'])
```

Out[26]:

	email freq	count	count %
0	1	123531	0.986086
1	2	1743	0.013914

In [27]:
```
# email_hash frequency distribution by {email_hash, company_hash, job_position, ctc_upd
show_email_frequency(df, ['email_hash', 'company_hash', 'job_position', 'ctc_updated_ye
```

Out[27]:

	email freq	count	count %
0	1	126919	0.999614
1	2	49	0.000386

In [28]:
```
# email_hash frequency distribution by {email_hash, company_hash, job_position, ctc_upd
show_email_frequency(df, ['email_hash', 'company_hash', 'job_position', 'ctc_updated_ye
```

Out[28]:

	email freq	count	count %
0	1	126971	0.999819
1	2	23	0.000181

Missing value imputation

In [29]:
```
df.isna().sum()
```

Out[29]:
```
company_hash           44
email_hash              0
orgyear               130
ctc                     0
job_position        38283
ctc_updated_year        0
dtype: int64
```

Imputation strategy:

We see that 'company_hash', 'orgyear', and 'job_position' columns have missing values. 'Company_hash' and 'job_position' are categorical columns while 'orgyear' can be treated as a numeric column. We also notice that there are only 44 company_hash values missing, while 38283 missing values for job_position. Missing value count for orgyear is 130. We impute missing values in stages and separately for all three features as described below.

We first standard scale all numeric features.

1. **company_hash**: We will use knn imputer from sklearn library to find the nearest neighbour (k=1) and use its company value. In order to find neighbors, we will use these features: *orgyear, ctc, job_position, ctc_updated_year*. sklearn KNN library implementation requires all input columns to be numeric, so we must encode all categorical columns. We one-hot-encode 'job_position' column (this is practical as after cleanup, we have 25 distinct job positions). We choose label encoding to convert 'company_hash' to numeric value. This should okay as long as we do not try to aggregate 'company_hash' value in some way. Since we choose k=1, we will just take the nearest neighbors 'company_hash' value.

2. **job_position**: We impute this feature in two stages: local imputation and global imputation. First, for all companies having at-least 5 learners, we try to impute missing values by considering peers from the same company. Finally, we impute remaining missing values using global knn imputation.

3. **orgyear**: We impute this using knn imputer (k=1).

Finally, we rescale data back.

imputation helper class

In [30]:
```
df_bk = df.copy()
```

In [ ]:
```
#df = df_bk.copy()
```

In [31]:
```
class Impute_Helper:

    def __init__(self):
        self.enc_map = {}
```

```
    #ignores nan values.
    def labelencode(self, df, col_name, new_col_name):
        mask = ~df[col_name].isna()
        le = LabelEncoder()
        df.loc[mask, new_col_name] = le.fit_transform(df[mask][col_name])
        self.enc_map[f'{col_name}_le'] = le

    def labeldecode(self, vals, col_name):
        ret, key = None, f'{col_name}_le'
        if(key in self.enc_map):
            le = self.enc_map[key]
            ret = le.inverse_transform(vals)
        return ret

    def onehotencode(self, df, col_name):
        ohe = OneHotEncoder(handle_unknown='ignore', drop='first', sparse=False)
        self.enc_map[f'{col_name}_ohe'] = ohe
        vals = ohe.fit_transform(df[col_name])
        return pd.DataFrame(data=vals, columns=ohe.categories_[0][0:vals.shape[1]])

    def onehotdecode(self, vals, col_name):
        ret, key = None, f'{col_name}_ohe'
        if(key in self.enc_map):
            ohe = self.enc_map[key]
            ret = ohe.inverse_transform(vals)
        return ret
```

In [32]:

```
#scale all numeric features before imputation
scaler = StandardScaler()
scale_cols = ['orgyear', 'ctc', 'ctc_updated_year']
scaled_vals = scaler.fit_transform(df[scale_cols])
df.loc[:,scale_cols] = scaled_vals
```

In [33]:

```
df.head()
```

Out[33]:

	company_hash	email_hash	orgyear	ctc	
0	atrgxnntxzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	0.263209	-0.105788	
1	qtrxvzwtxzegwgbbrxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	0.733879	-0.156774	fulls
2	ojzwnvwnxwvx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	0.027874	-0.035193	bacl
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	0.498544	-0.137164	bacl
4	qxensqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	0.498544	-0.082256	fulls

impute 'company_hash'

In [34]:

```
df2 = df.copy()
imph = Impute_Helper()

#Label encode company_hash
imph.labelencode(df2, 'company_hash', 'company_hash_enc')
```

```
#onehotencode job positions
jp_ohe_df = imph.onehotencode(df, ['job_position'])
df2 = pd.merge(df2, jp_ohe_df, left_index=True, right_index=True)

#run knnimputer (k=1) to impute company_hash with nearest neighbor's company_hash value
knnimp = KNNImputer(n_neighbors = 1)
collist = df2.columns[~df2.columns.isin(['company_hash', 'email_hash', 'job_position'])
imputed_vals = knnimp.fit_transform(df2[collist])

# replace original company_hash column with new imputed column after decoding the resul
df['company_hash'] = imph.labeldecode(imputed_vals[:, collist.index('company_hash_enc')
```

In [36]:
```
df.isna().sum()
```

Out[36]:
```
company_hash            0
email_hash              0
orgyear               130
ctc                     0
job_position        38283
ctc_updated_year        0
dtype: int64
```

impute 'job_position'

In [38]:
```
#Find companys with atleast 30 learners.
cvc = df['company_hash'].value_counts()
popular_comp = cvc[cvc >= 5].index.values
```

In [42]:
```
imputed_vals = None
decoded_vals = None

def impute_jp(df, selection_mask):

    global imputed_vals
    global decoded_vals

    df_tmp = df.loc[selection_mask].copy()

    imph = Impute_Helper()
    imph.labelencode(df_tmp, 'job_position', 'job_position_enc')

    #dataframe collist for knn imputation
    collist = df_tmp.columns[~df_tmp.columns.isin(['company_hash', 'email_hash', 'job_p

    #run knnimputer (k=1) to impute job_position with nearest neighbor's (from the same
    knnimp = KNNImputer(n_neighbors = 1)
    imputed_vals = knnimp.fit_transform(df_tmp[collist])

    #ensure no features were dropped
    if(imputed_vals.shape[1] == len(collist)):

        #decode imputed values and copy back to the original frame
        df.loc[selection_mask, 'job_position'] = imph.labeldecode(imputed_vals[:, colli
        #df.loc[selection_mask, 'orgyear'] = imputed_vals[:, collist.index('orgyear')]
```

In [43]:
```
#local imputation within each company
for comp in popular_comp:
     mask = df['company_hash'] == comp
     impute_jp(df, mask)
```

In [44]:
```
df.isna().sum()
```

Out[44]:
```
company_hash          0
email_hash            0
orgyear             130
ctc                   0
job_position       7955
ctc_updated_year      0
dtype: int64
```

Note: We see that number of missing values for job_positions have reduced from to 38299 7955.
We now impute remaining values with global KNN (k=1) imputation (that is across companies).

In [45]:
```
#global imputation
impute_jp(df, [True]*df.shape[0])
```

In [46]:
```
df.isna().sum()
```

Out[46]:
```
company_hash          0
email_hash            0
orgyear             130
```

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

impute 'orgyear'

In [47]:
```
df2 = df.copy()
imph = Impute_Helper()

#onehotencode job positions
jp_ohe_df = imph.onehotencode(df, ['job_position'])
df2 = pd.merge(df2, jp_ohe_df, left_index=True, right_index=True)

#run knnimputer (k=1) to impute org_year with nearest neighbor's org_year value.
knnimp = KNNImputer(n_neighbors = 1)
collist = df2.columns[~df2.columns.isin(['company_hash', 'email_hash', 'job_position'])]
imputed_vals = knnimp.fit_transform(df2[collist])

# replace original orgyear column with new imputed column
df['orgyear'] = imputed_vals[:, collist.index('orgyear')]
```

In [48]:
```
df.isna().sum()
```

Out[48]:
```
company_hash         0
email_hash           0
orgyear              0
ctc                  0
job_position         0
ctc_updated_year     0
dtype: int64
```

In [49]:
```
# rescale all values
df.loc[:, scale_cols] = scaler.inverse_transform(df[scale_cols])
df.head()
```

Out[49]:

	company_hash	email_hash	orgyear	ctc	j
0	atrgxnntxzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000.0	
1	qtrxvzwtxzegwgbbrxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999.0	fulls
2	ojzwnvwnxwvx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000000.0	back
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700000.0	back
4	qxensqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400000.0	fulls

In [50]:
```
df.describe()
```

Out[50]:

	orgyear	ctc	ctc_updated_year
count	165313.000000	1.653130e+05	165313.000000
mean	2014.880935	2.448667e+06	2019.475099

	orgyear	ctc	ctc_updated_year
std	4.250720	1.274879e+07	1.345293
min	1970.000000	2.000000e+00	2015.000000
25%	2013.000000	5.500000e+05	2019.000000
50%	2016.000000	9.600000e+05	2020.000000
75%	2018.000000	1.700000e+06	2021.000000
max	2021.000000	1.000150e+09	2021.000000

In [51]:
```
df_imputed = df.copy()
```

In [52]:
```
df_imputed.to_csv('imputed_data.csv', index_label='index')
```

In [2]:
```
#df = pd.read_csv('imputed_data.csv', index_col='index')
```

Feature engineering, analysis, and data aggregation

Adding 'experience' feature

Each row in the given represents an employment record of a learner at a specific company, job position, with a specific CTC pay. Usually an employee may work in that position (all the above parameters staying the same) for more than a year. In such cases, there could be two ways to compute experience of a learner in that job.

1. Calculating the experience *at the start* the employment in that specific job.
2. Calculating the experience *including the duration* of that specific job.

For instance, consider the following records for employee 'Bob'. Assume that the current year is 2022. The last two columns show experience calculated using both the approaches.

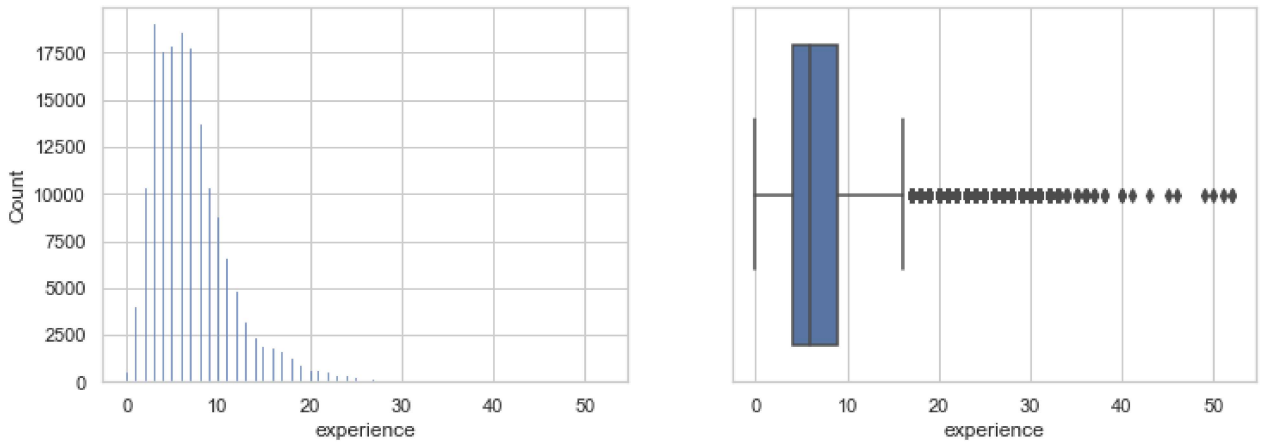
Employee	Company	orgyear	ctc	ctc_updated_year	experience(at job_start)	experience(at job_end till curryear)
Bob	Footech	2015	100000	2015	0	2
Bob	Footech	2015	120000	2017	2	3
Bob	Barsoft	2018	140000	2018	3	5
Bob	Barsoft	2018	180000	2020	5	7

In this case study, we will take the second approach for calculating job experience (considering job_end or curr_year for ongoing job). The primary reason is that with approach 2, when we take the maximum experience value, it coincides nicely with the learner's total work experience (this will be useful when we aggregate data at learner's level for clustering).

In [53]:
```
class workexperience:
    def __init__(self, curryear):
        self.curryear = curryear

    def compute_experience(self, df):
        career_start = df['orgyear'].min()
        job_end_year = df['ctc_updated_year'].shift(-1).fillna(self.curryear)
        df['experience'] = job_end_year - career_start

        return df

we = workexperience(curryear)

df['experience'] = 0

email_count = df.groupby('email_hash')['email_hash'].count()
multi_email_list = email_count[email_count > 1].index.values
emails_with_multiple_records = df['email_hash'].isin(multi_email_list)

#update experience for singular emails first
df.loc[~emails_with_multiple_records, 'experience'] = curryear - df[~emails_with_multip

#update emails with multiple rows.
df.loc[emails_with_multiple_records, 'experience'] = df.sort_values(by=['orgyear', 'ctc
```

```
C:\Users\chins\AppData\Local\Temp/ipykernel_15864/95244597.py:24: UserWarning: Boolean S
eries key will be reindexed to match DataFrame index.
  df.loc[emails_with_multiple_records, 'experience'] = df.sort_values(by=['orgyear', 'ct
c_updated_year'], ascending=[True, True])[emails_with_multiple_records].groupby('email_h
ash').apply(we.compute_experience)
```

In [54]:
```
fig, ax = plt.subplots(1,2, figsize=(12,4))
sns.histplot(df['experience'], ax=ax[0])
sns.boxplot(x=df['experience'], ax=ax[1])
plt.show()
```



In [55]:
```
#defect outliers using simple IQR
def findoutliers(arr):
    q3 = np.percentile(arr, 75)
    q1 = np.percentile(arr, 25)
    iqr = q3-q1
    ulim = q3 + 1.5*iqr
```

```
    llim = q1 - 1.5*iqr
    return pd.Series([True if((ele > ulim) or (ele < llim)) else False for ele in arr])

#outliers_mask = findoutliers(df['experience'])
#df[outliers_mask].shape[0]
df[df['experience'] > 25].shape
```

Out[55]:
```
(535, 7)
```

Observation: The distribution of 'experience' is somewhat right skewed with several outliers on the right tail. From the histpgram, experience value 25 looks like a good cut off point for cluster analysis.

In [56]:
```
#delete outlier rows
df.drop(df[df['experience'] > 25].index, inplace=True)
df = df.reset_index(drop=True)
```

In [ ]:
```

```

create 'experience_level' categorical feature

For more meaningful analysis, we can create 'experience_level' categorical feature from 'experience'.

In [57]:
```
df['experience_level'] = pd.cut(df['experience'], bins=[-1, 0, 2, 5, 9, 12, 15, 20, 30,
df['experience_level'] = df['experience_level'].astype('category')
```

In [58]:
```
df['experience_level'].value_counts()
```

Out[58]:
```
6-9      60298
3-5      54299
10-12    20173
1-2      14258
12-15     7363
16-20     6059
21-30     1845
0          483
30+          0
Name: experience_level, dtype: int64
```

In [59]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164778 entries, 0 to 164777
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   company_hash      164778 non-null  object
 1   email_hash        164778 non-null  object
 2   orgyear           164778 non-null  float64
 3   ctc               164778 non-null  float64
 4   job_position      164778 non-null  object
 5   ctc_updated_year  164778 non-null  float64
 6   experience        164778 non-null  float64
 7   experience_level  164778 non-null  category
```

```
dtypes: category(1), float64(4), object(3)
memory usage: 9.0+ MB
```

Analyzing 'ctc'

In [60]:
```
df['ctc'].describe()
```

Out[60]:
```
count    1.647780e+05
mean     2.442119e+06
std      1.273094e+07
min      2.000000e+00
25%      5.500000e+05
50%      9.600000e+05
75%      1.700000e+06
max      1.000150e+09
Name: ctc, dtype: float64
```

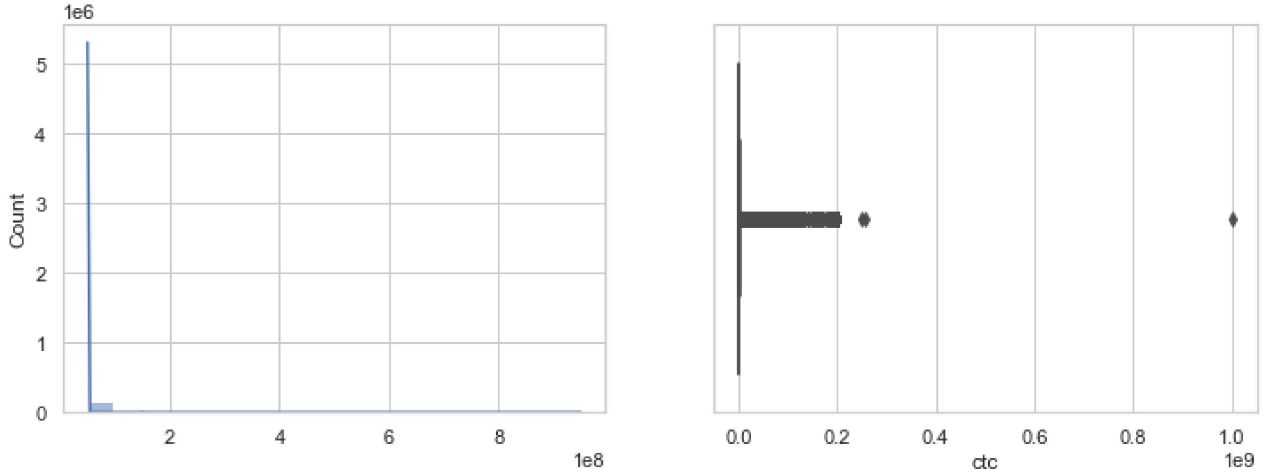
In [61]:
```
np.percentile(df['ctc'], range(5,100,5))
```

Out[61]:
```
array([ 200000.,  340000.,  400000.,  480000.,  550000.,  600000.,
        700000.,  780000.,  850000.,  960000., 1050000., 1200000.,
       1340000., 1500000., 1700000., 2000000., 2300000., 2850000.,
       3900000.])
```

In [62]:
```
#check range for [3,97] percentile
np.percentile(df['ctc'], [3,97])
```

Out[62]:
```
array([ 100000., 5000000.])
```

In [63]:
```
ctc_vals = np.array(list(map(lambda x:np.round((x.left + x.right)/2), list(pd.cut(df['c
fig, ax = plt.subplots(1,2, figsize=(12,4))
sns.histplot(ctc_vals, kde=True, ax=ax[0])
sns.boxplot(x=df['ctc'], ax=ax[1])
plt.show()
```

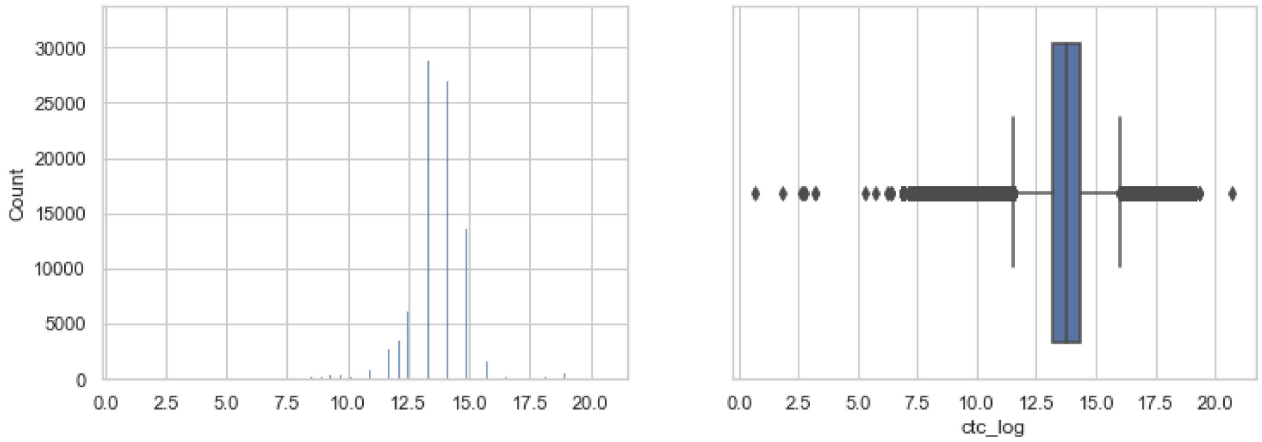


Observation: As we can see, 'ctc' has a highly positively skewed distribution with a lot of outliers. 'ctc' values range from 2 to 1000150000 (1 billion), with mean at 2448667 (24.5 lacs), and median at 960000 (9.6 lacs). Since distance-based clustering algorithms are sensitive to outliers and are

generally adversely impacted by highly skewed data, we will first log transform CTC and then remove outliers.

create 'ctc_log' feature

In [69]:
```
df['ctc_log'] = np.log(df['ctc'])
ctc_vals = np.array(list(map(lambda x:np.round((x.left + x.right)/2, 2), list(pd.cut(df
fig, ax = plt.subplots(1,2, figsize=(12,4))
sns.histplot(ctc_vals, ax=ax[0])
sns.boxplot(x=df['ctc_log'], ax=ax[1])
plt.show()
```



In [78]:
```
q3 = np.percentile(df['ctc_log'], 75)
q1 = np.percentile(df['ctc_log'], 25)
iqr = q3-q1
ulim = q3 + 1.5*iqr
llim = q1 - 1.5*iqr

print(f'Inliers ctc range: ctc_log ({llim}, {ulim}) or ctc ({np.exp(llim)}, {np.exp(uli
```

```
Inliers ctc range: ctc_log (11.524975679481969, 16.03883668675313) or ctc (101212.311085
78993, 9238006.621620089)
```

In [76]:
```
outlier_mask = ((df['ctc_log'] < llim) | (df['ctc_log'] > ulim))
df[outlier_mask].shape
```

Out[76]:
```
(7451, 9)
```

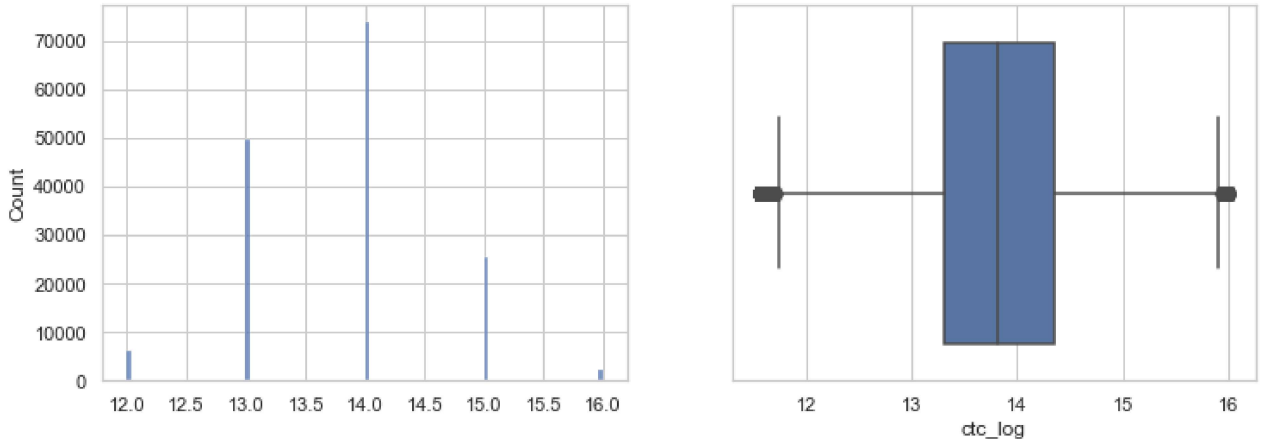
In [79]:
```
#remove outliers
df.drop(df[outlier_mask].index, inplace=True)
df = df.reset_index(drop=True)
```

In [139…
```
# bottom_3_ctc = np.percentile(df['ctc'], 3)
# top_3_ctc = np.percentile(df['ctc'], 97)

# inlier_mask = ((df['ctc'] >= bottom_3_ctc) & (df['ctc'] <= top_3_ctc))

# #remove outliers
```

```
# df.drop(df[~inlier_mask].index, inplace=True)
# df = df.reset_index(drop=True)
```

In [82]:

```
ctc_vals = np.array(list(map(lambda x:np.round((x.left + x.right)/2), list(pd.cut(df['c
fig, ax = plt.subplots(1,2, figsize=(12,4))
sns.histplot(ctc_vals, ax=ax[0])
sns.boxplot(x=df['ctc_log'], ax=ax[1])
plt.show()
```



observation: : The distribution looks better after log transforming ctc and removing outliers.

Calculate CTC stats by company, job position, and experience

To make comparisons more meaningful, we will use 'experience_level' rather than 'experience'

In [83]:

```
def perc_factory(n):
    def percentile_(x):
        return np.percentile(x, n)
    return percentile_

perc33 = perc_factory(33)
perc66 = perc_factory(66)

ctc_by_c_jp_exp = df.groupby(['company_hash', 'job_position', 'experience_level']).agg(
ctc_by_c_jp_exp.columns = ['ctc_count', 'ctc_min', 'ctc_max', 'ctc_mean', 'ctc_median',
ctc_by_c_jp_exp = ctc_by_c_jp_exp.reset_index()
ctc_by_c_jp_exp.sort_values(by='ctc_count', ascending=False).head()
```

Out[83]:

	company_hash	job_position	experience_level	ctc_count	ctc_min	ctc_max	
2769545	nvnvwgzohrnvzwjotqcxwto	backendengineer	3-5	777	300000.0	8000000.0	5.2
5402046	vbvkgz	backendengineer	6-9	610	120000.0	9000000.0	2.6
1175303	eqtoytq	other	3-5	576	180000.0	8000000.0	9.8
2769653	nvnvwgzohrnvzwjotqcxwto	other	3-5	505	190000.0	2930000.0	4.9
6900851	xzegojo	fullstackengineer	3-5	496	200000.0	3960000.0	6.0

Calculate CTC stats by company and job position

In [84]:
```
ctc_by_c_jp = df.groupby(['company_hash', 'job_position']).agg({'ctc': ['count', 'min',
ctc_by_c_jp.columns = ['ctc_count', 'ctc_min', 'ctc_max', 'ctc_mean', 'ctc_median', 'pe
ctc_by_c_jp = ctc_by_c_jp.reset_index()
ctc_by_c_jp.sort_values(by='ctc_count', ascending=False).head()
```

Out[84]:

	company_hash	job_position	ctc_count	ctc_min	ctc_max	ctc_mean	ctc_med
19967	nvnvwgzohrnvzwjotqcxwto	backendengineer	1380	150000.0	8500000.0	6.175984e+05	4600(
38769	vbvkgz	backendengineer	1271	110000.0	9000000.0	2.565029e+06	24000(
19979	nvnvwgzohrnvzwjotqcxwto	other	1063	110000.0	3000000.0	5.099073e+05	4000(
19976	nvnvwgzohrnvzwjotqcxwto	fullstackengineer	964	160000.0	7150000.0	6.493454e+05	5500(
49583	xzegojo	other	921	140000.0	6800000.0	5.510716e+05	4200(

Calculate CTC stats by company

In [85]:
```
ctc_by_c = df.groupby(['company_hash']).agg({'ctc': ['count', 'min', 'max', 'mean', 'me
ctc_by_c.columns = ['ctc_count', 'ctc_min', 'ctc_max', 'ctc_mean', 'ctc_median', 'perc3
ctc_by_c = ctc_by_c.reset_index()
ctc_by_c['company_tier_perc'] = ctc_by_c['ctc_median'].transform(lambda c: np.round((pe
ctc_by_c.sort_values(by='ctc_count', ascending=False).head()
```

Out[85]:

	company_hash	ctc_count	ctc_min	ctc_max	ctc_mean	ctc_median	perc33	
12309	nvnvwgzohrnvzwjotqcxwto	5664	103000.0	9000000.0	6.108456e+05	459999.0	400000.0	
30670	xzegojo	3748	105000.0	9000000.0	6.482141e+05	500000.0	405000.0	
24009	vbvkgz	2608	105000.0	9200000.0	2.220623e+06	2000000.0	1370000.0	2
27354	wgszxkvzn	2266	102000.0	8900000.0	7.386892e+05	580000.0	450000.0	
32674	zgnvuurxwvmrtvwwghzn	2157	105000.0	9000000.0	8.639282e+05	600000.0	400000.0	

In [ ]:
```
```

In [86]:
```
df_feat_engg1 = df.copy()
```

In [87]:
```
df_feat_engg1.to_csv('feat_engg_1.csv',index_label='index')
```

In [292…
```
#df = pd.read_csv('feat_engg_1.csv', index_col = 'index')
```

Add ctc(lacs) for display purpose

In [88]:
```
df['ctc(lacs)'] = df['ctc'] / 100000
```

create 'job_position_tier'

We can assign each job_position tier value of {1,2,3} based on their median CTC value. Companies with top 33% CTC values are considered tier1 job_positions, bottom 33% are considered tier 3 job_positions, and the remaining ones are tier2 job_positions. We will also create 'job_position_tier_perc' to denote the actual percentile rank of that job_position with regards to CTC.

In [90]:
```
def get_quantile_fn(col_name, perc33_val, perc66_val):

    def compute_quantile(r):

        if r[col_name] >= perc33_val and r[col_name] <= perc66_val:
            return 2
        elif r[col_name] > perc66_val:
            return 1
        else:
            return 3

    return compute_quantile

ctc_by_jp = df.groupby('job_position')['ctc'].median().reset_index()

ctc_median_perc33 = np.percentile(ctc_by_jp['ctc'], 33)
ctc_median_perc66 = np.percentile(ctc_by_jp['ctc'], 66)

compute_jp_tier = get_quantile_fn('ctc', ctc_median_perc33, ctc_median_perc66)

ctc_by_jp['job_position_tier'] = ctc_by_jp.apply(compute_jp_tier, axis=1)
ctc_by_jp['job_position_tier_perc'] = ctc_by_jp['ctc'].transform(lambda c: np.round((pe

df2 = df.copy()
df2 = df2.reset_index().merge(ctc_by_jp, on=['job_position'], how="left").set_index('in
df['job_position_tier'] = df2['job_position_tier']
df['job_position_tier_perc'] = df2['job_position_tier_perc']
```

In [91]:
```
df[['job_position', 'job_position_tier',  'job_position_tier_perc']].drop_duplicates().
```

Out[91]:

	job_position	job_position_tier	job_position_tier_perc
0	backendarchitect	1	1.00
1	engineeringleadership	1	0.96
2	programmanager	1	0.92
3	productmanager	1	0.88
4	seniorsoftwareengineer	1	0.84
5	securityleadership	2	0.72
6	researchengineers	2	0.72
7	cofounder	2	0.72
8	datascientist	2	0.72

	job_position	job_position_tier	job_position_tier_perc
9	backendengineer	2	0.72
10	productdesigner	2	0.60
11	devopsengineer	2	0.56
12	fullstackengineer	2	0.52
13	androidengineer	2	0.44
14	frontendengineer	2	0.44
15	iosengineer	2	0.44
16	releaseengineer	2	0.36
17	engineeringintern	3	0.32
18	sdet	3	0.28
19	databaseadministrator	3	0.24
20	qaengineer	3	0.18
21	dataanalyst	3	0.18
22	other	3	0.12
23	supportengineer	3	0.08
24	noncoder	3	0.04

Create 'company_tier' and 'company_scale' features

1. **company_tier** - We can divide all companies based on their meadian CTC value into 3 tiers. Top 33% highest paying companies are given tier1 value, Bottom 33% are given tier 3, and remaining are tier2.

2. **company_scale** - based on the number of employed learners, we can divide the companies into three scales. Companies employing 1-5 learners are scale 3, 6-100 learners having scale 2, and having >100 learners as scale 1.

In [92]:
```
ctc_median_perc33 = np.percentile(ctc_by_c['ctc_median'], 33)
ctc_median_perc66 = np.percentile(ctc_by_c['ctc_median'], 66)

company_scale_lim1 = 5
company_scale_lim2 = 100

compute_company_tier = get_quantile_fn('ctc_median', ctc_median_perc33, ctc_median_perc
compute_company_scale = get_quantile_fn('ctc_count', company_scale_lim1, company_scale_

ctc_by_c['company_tier'] = ctc_by_c.apply(compute_company_tier, axis=1)
ctc_by_c['company_scale'] = ctc_by_c.apply(compute_company_scale, axis=1)

df2 = df.copy()
df2 = df2.reset_index().merge(ctc_by_c, on=['company_hash'], how="left").set_index('ind
df['company_tier'] = df2['company_tier']
```

```
df['company_scale'] = df2['company_scale']
df['company_tier_perc'] = df2['company_tier_perc']
```

In [93]:
```
np.round(df.groupby(['company_tier']).agg({'company_hash':'nunique', 'ctc(lacs)': ['mea
```

Out[93]:

	company_hash	ctc(lacs)		experience
	nunique	mean	median	mean
company_tier				
1	11447	19.88	16.8	8.00
2	12385	9.81	8.0	6.49
3	9984	5.48	4.3	5.47

Create features for learner's relative performance.

We will create the following categories of features to denote a learner's performance/standing among his/her peers using ctc as the evaluation metric.

1. **'designation'** - Possible values are {1,2,3}. To calculate a learner's designation, we first find all peers of the learners **who are working at the same company, job position, and fall in the same experience level bucket**. Divide these peers into 3 groups (or designations) - 1,2, and 3. Top 33% are assigned designation 1, bottom 33% are assigned designation 3, and remaining learners are assigned designation 2.

2. **'class'** - Possible values are {1,2,3}. To calculate a learner's class, we first find all peers of the learners **who are working at the same company and in the same job position (regardless of their experience level)**. Divide these peers into 3 groups (or classes) - 1,2, and 3. Top 33% are assigned class 1, bottom 33% are assigned class 3, and remaining learners are assigned class 2.

3. **'tier'** - Possible values are {1,2,3}. To calculate a learner's tier, we first find all peers of the learners **who are working at the same company (regardless of their job designation or experience level)**. Divide these peers into 3 groups (or tiers) - 1,2, and 3. Top 33% are assigned tier 1, bottom 33% are assigned tier 3, and remaining learners are assigned tier 2.

Imp Note

1. We will also compute 'desination_perc', 'class_perc', and 'tier_perc' features denoting the actual percentile rank for that learner among his/her peer group.

2. When the peer group size is one (either because of lack of data or we only have one learner from that company/job_position), we try to estimate 'XXXX_perc' value by taking average of 'job_position_tier_perc' and 'company_tier_perc' values.

In [94]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 157327 entries, 0 to 157326
Data columns (total 15 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   company_hash            157327 non-null  object
 1   email_hash              157327 non-null  object
 2   orgyear                 157327 non-null  float64
 3   ctc                     157327 non-null  float64
 4   job_position            157327 non-null  object
 5   ctc_updated_year        157327 non-null  float64
 6   experience              157327 non-null  float64
 7   experience_level        157327 non-null  category
 8   ctc_log                 157327 non-null  float64
 9   ctc(lacs)               157327 non-null  float64
 10  job_position_tier       157327 non-null  int64
 11  job_position_tier_perc  157327 non-null  float64
 12  company_tier            157327 non-null  int64
 13  company_scale           157327 non-null  int64
 14  company_tier_perc       157327 non-null  float64
dtypes: category(1), float64(8), int64(3), object(3)
memory usage: 17.0+ MB
```

In [95]:

```
def compute_quantiles(r):

    if r['ctc'] >= r['perc33'] and r['ctc'] <= r['perc66']:
        return 2
    elif r['ctc'] > r['perc66']:
        return 1
    else:
        return 3

#merge df2 with right dataframe and retain df2's index. This will allow us to assign ne
df2 = df_feat_engg1.copy()
df2 = df2.reset_index().merge(ctc_by_c_jp_exp, on=['company_hash', 'job_position', 'exp
df['designation'] = df2.apply(compute_quantiles, axis=1)

df2 = df_feat_engg1.copy()
df2 = df2.reset_index().merge(ctc_by_c_jp, on=['company_hash', 'job_position'], how="le
df['class'] = df2.apply(compute_quantiles, axis=1)

df2 = df_feat_engg1.copy()
df2 = df2.reset_index().merge(ctc_by_c, on=['company_hash'], how="left").set_index('ind
df['tier'] = df2.apply(compute_quantiles, axis=1)
```

In [96]:

```
def get_compute_percentile_fn(col_name, col_to_compare='ctc'):
    def compute_percentile(df):
        peer_ctc = df[col_to_compare].values.copy()
        if len(peer_ctc) == 1:
            df[col_name] = -1 #impute later
        else:
            df[col_name] = np.round(df[col_to_compare].transform(lambda c: percentileof

        return df

    return compute_percentile

df = df.groupby(['company_hash', 'job_position', 'experience_level']).apply(get_compute
```

```
df = df.groupby(['company_hash', 'job_position']).apply(get_compute_percentile_fn('clas
df = df.groupby(['company_hash']).apply(get_compute_percentile_fn('tier_perc'))
```

In [97]:

```
#impute values
def impute_values_from_hierarchy(ser):
    if(ser['tier_perc'] == -1):
        ser['tier_perc'] = np.mean([ser['job_position_tier_perc'], ser['company_tier_pe
    if(ser['class_perc'] == -1):
        ser['class_perc'] = ser['tier_perc']
    if(ser['designation_perc'] == -1):
        ser['designation_perc'] = ser['tier_perc']

    return ser

df = df.apply(impute_values_from_hierarchy, axis=1)
```

In [98]:

```
np.round(df.groupby(['designation']).agg({'ctc(lacs)': ['mean', 'median'], 'experience'
```

Out[98]:

	ctc(lacs)		experience
	mean	median	mean
designation			
1	19.52	15.4	6.63
2	12.35	9.0	7.22
3	8.54	6.6	6.49

In [99]:

```
np.round(df.groupby(['class']).agg({'ctc(lacs)': ['mean', 'median'], 'experience': 'mea
```

Out[99]:

	ctc(lacs)		experience
	mean	median	mean
class			
1	20.80	17.0	7.93
2	11.85	9.0	6.98
3	7.79	6.0	5.75

In [100…

```
np.round(df.groupby(['tier']).agg({'ctc(lacs)': ['mean', 'median'], 'experience': 'mean
```

Out[100…

	ctc(lacs)		experience
	mean	median	mean
tier			
1	22.00	18.3	8.51
2	11.34	9.0	6.76

	ctc(lacs)		experience
	mean	median	mean
tier			
3	6.76	5.6	5.50

In [101…
```
df_feat_engg2 = df.copy()
```

In [102…
```
df_feat_engg2.to_csv('feat_engg2.csv', index_label='index')
```

In [162…
```
#df = pd.read_csv('feat_engg2.csv', index_col = 'index')
```

In [103…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 157327 entries, 0 to 157326
Data columns (total 21 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   company_hash            157327 non-null  object
 1   email_hash              157327 non-null  object
 2   orgyear                 157327 non-null  float64
 3   ctc                     157327 non-null  float64
 4   job_position            157327 non-null  object
 5   ctc_updated_year        157327 non-null  float64
 6   experience              157327 non-null  float64
 7   experience_level        157327 non-null  object
 8   ctc_log                 157327 non-null  float64
 9   ctc(lacs)               157327 non-null  float64
 10  job_position_tier       157327 non-null  int64
 11  job_position_tier_perc  157327 non-null  float64
 12  company_tier            157327 non-null  int64
 13  company_scale           157327 non-null  int64
 14  company_tier_perc       157327 non-null  float64
 15  designation             157327 non-null  int64
 16  class                   157327 non-null  int64
 17  tier                    157327 non-null  int64
 18  designation_perc        157327 non-null  float64
 19  class_perc              157327 non-null  float64
 20  tier_perc               157327 non-null  float64
dtypes: float64(11), int64(6), object(4)
memory usage: 30.4+ MB
```

aggregate data at learner level

Now that we have created features from the all available data, before we cluster learners, we will aggregate data at learner level. For each learner, we now pick his current (last job with highest ctc_updated_year/orgyear) job details including 'ctc', 'company_hash', experience details, designation, class, and tier. We can discard orgyear/crtc_updated_year details as the absolute year details will not be meaningful.

In [104…
```
#sort dataframe by newest jobs to oldest jobs, and group by email_hash
df = df.sort_values(by=['ctc_updated_year', 'orgyear'], ascending=[False, False]).group
    'ctc': 'first',
    'ctc_log': 'first',

    'experience': 'first',
    'experience_level': 'first',

    'designation': 'first',
    'class': 'first',
    'tier': 'first',
    'job_position_tier': 'first',
    'company_tier': 'first',
    #'company_scale': 'first',

    'designation_perc': 'first',
    'class_perc': 'first',
    'tier_perc': 'first',
    'job_position_tier_perc': 'first',
    'company_tier_perc': 'first',

    'company_hash': 'first',
    'job_position': 'first'
}).reset_index()
```

In [105…
```
df_agg = df.copy()
```

In [106…
```
df_agg.to_csv('learners_agg.csv', index_label='index')
```

In [410…
```
#df = pd.read_csv('learners_agg.csv' ,index_col='index')
```

In [411…
```
#Add ctc(lacs) column for better display
df['ctc(lacs)'] = np.round(df['ctc'] / 100000, 2)
```

In [412…
```
df.head()
```

Out[412…

index	email_hash	ctc	ctc_log	experience	experience_
0	00003288036a44374976948c327f246fdbdf0778546904...	3500000.0	15.068274	10.0	1
1	0000aaa0e6b61f7636af1954b43d294484cd151c9b3cf6...	250000.0	12.429216	9.0	
2	0000d58fbc18012bf6fa2605a7b0357d126ee69bc41032...	1300000.0	14.077875	3.0	
3	000120d0c8aa304fcf12ab4b85e21feb80a342cfea03d4...	2000000.0	14.508658	18.0	1
4	00014d71a389170e668ba96ae8e1f9d991591acc899025...	3400000.0	15.039286	13.0	1

Univariate and Bivariate analysis

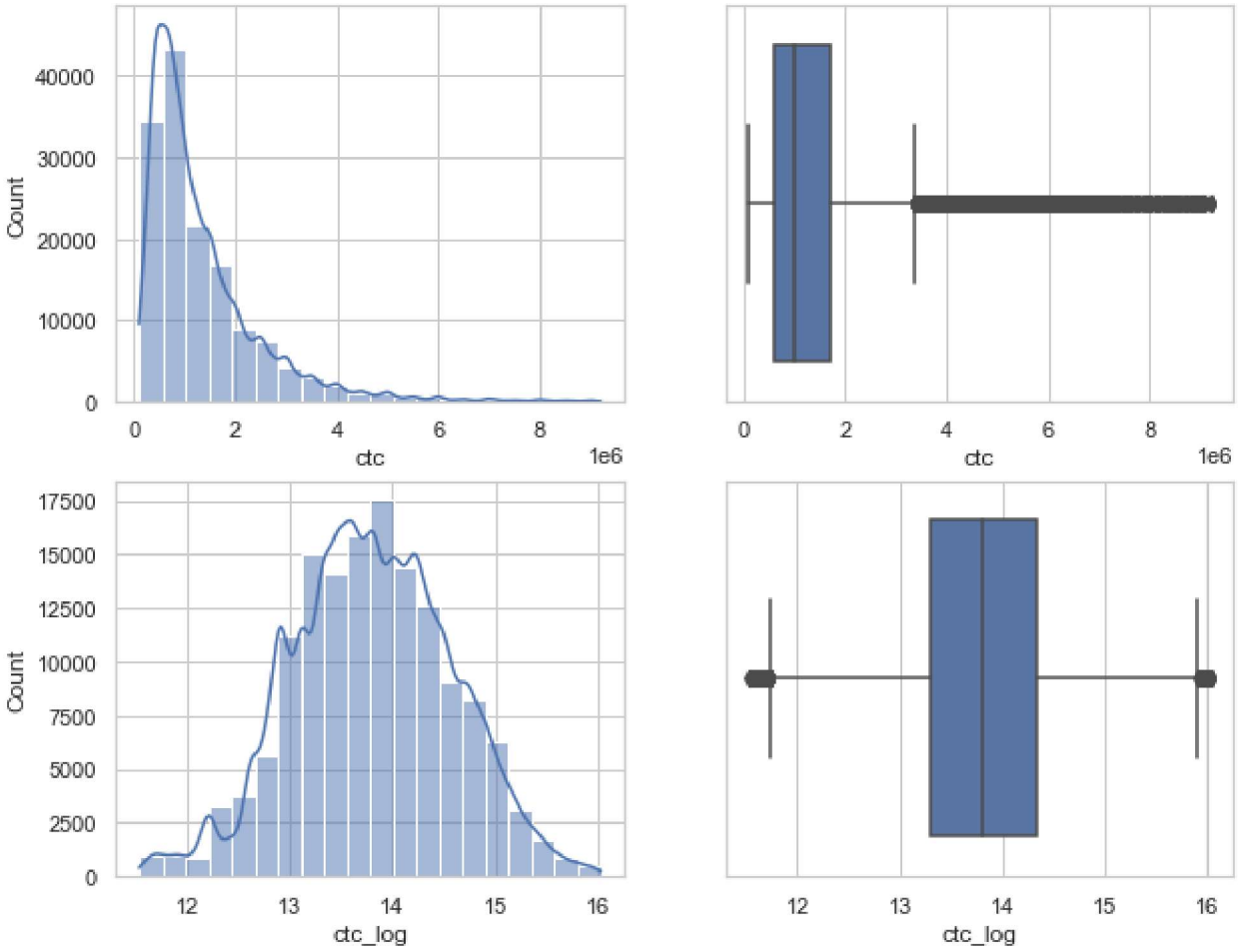
In [318…

```
#Common utility functions
def showpercent(ax, total):
    for p in ax.patches:
            percent = '{:.1f}%'.format(100 * p.get_height()/total)
            xpos = p.get_x() + p.get_width()/2
            ypos = p.get_height()
            ax.annotate(percent, (xpos, ypos),ha='center', va='bottom')

def showpercent_with_hue(ax, hue_levels, x_levels):

    heights_arr = np.array([p.get_height() for p in ax.patches])
    heights = pd.Series([p.get_height() for p in ax.patches]).fillna(0).values.reshape(
    percents = np.round((heights * 100) / np.sum(heights, axis=0)+0.001,1)
    perclist = percents.flatten(order='C') #flatten in column-major (F-style) order

    for i in range(len(ax.patches)):
        p = ax.patches[i]
        percent = f'{perclist[i]}%'
        xpos = p.get_x() + p.get_width()/2
        ypos = p.get_height()

        ax.annotate(percent, (xpos, ypos),ha='center', va='bottom')
```

'ctc' and 'ctc_log'

In [163…

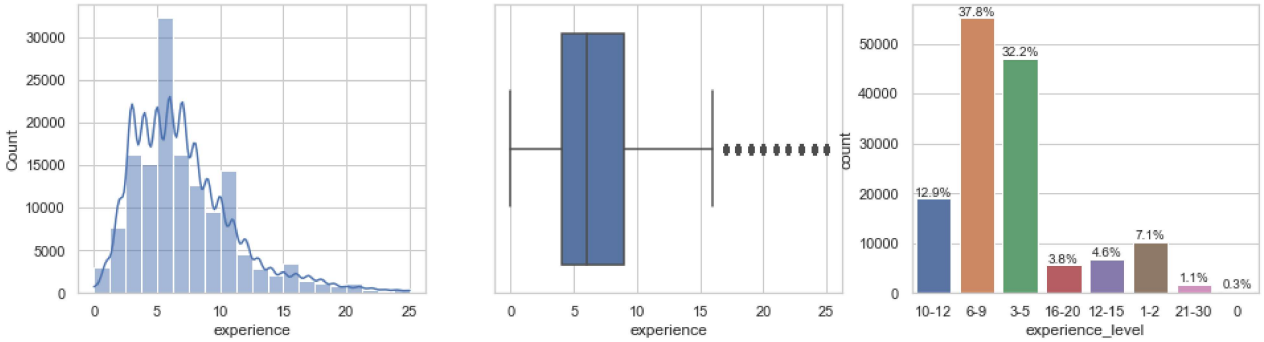
```
fig, ax = plt.subplots(2,2, figsize=(10,8))

sns.histplot(x=df['ctc'], bins=20, kde=True, ax=ax[0][0])
sns.boxplot(x=df['ctc'], ax=ax[0][1])

sns.histplot(x=df['ctc_log'], bins=20, kde=True, ax=ax[1][0])
sns.boxplot(x=df['ctc_log'], ax=ax[1][1])

plt.show()
```



experience and experience_level

In [413…

```
fig, ax = plt.subplots(1,3, figsize=(16,4))

sns.histplot(x=df['experience'], bins=20, kde=True, ax=ax[0])
sns.boxplot(x=df['experience'], ax=ax[1])
sns.countplot(x=df['experience_level'], ax=ax[2])
showpercent(ax[2], df.shape[0])

plt.show()
```



experience_level and average CTC

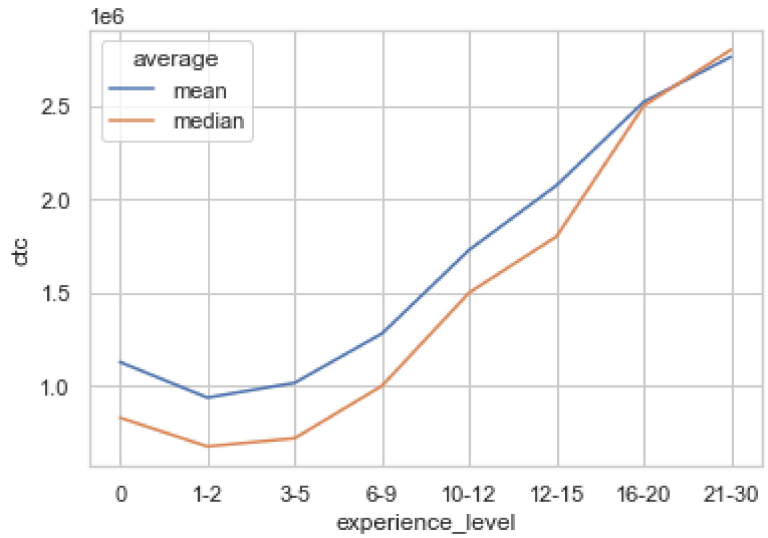
In [309…

```
data = df.groupby('experience_level')['ctc'].agg(['mean', 'median']).reset_index()
data['exp_left_edge'] = data['experience_level'].transform(lambda x: int(x[:x.index('-'
```

```
data = data.sort_values(by='exp_left_edge')
data = data.set_index('experience_level')[['mean', 'median']].stack().reset_index().ren

sns.lineplot(x='experience_level', y='ctc', hue='average', data=data)
```

Out[309…

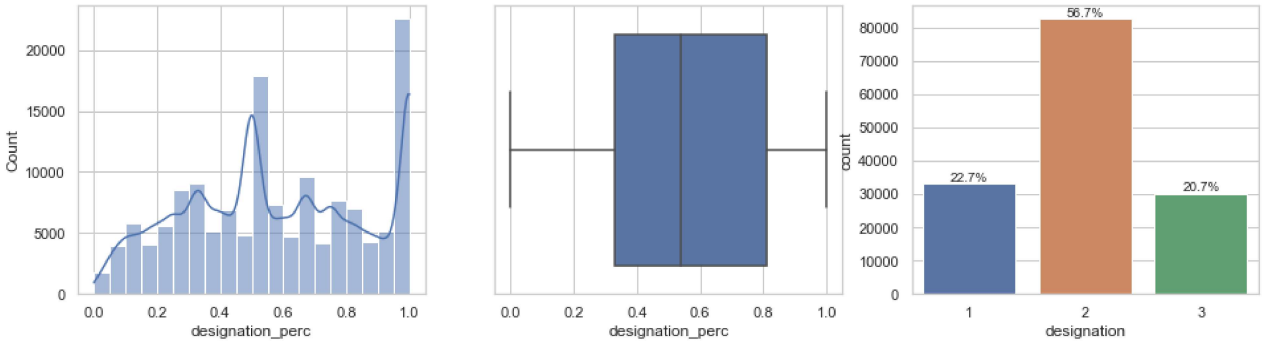
```
<AxesSubplot:xlabel='experience_level', ylabel='ctc'>
```



designation and designation_perc

In [414…

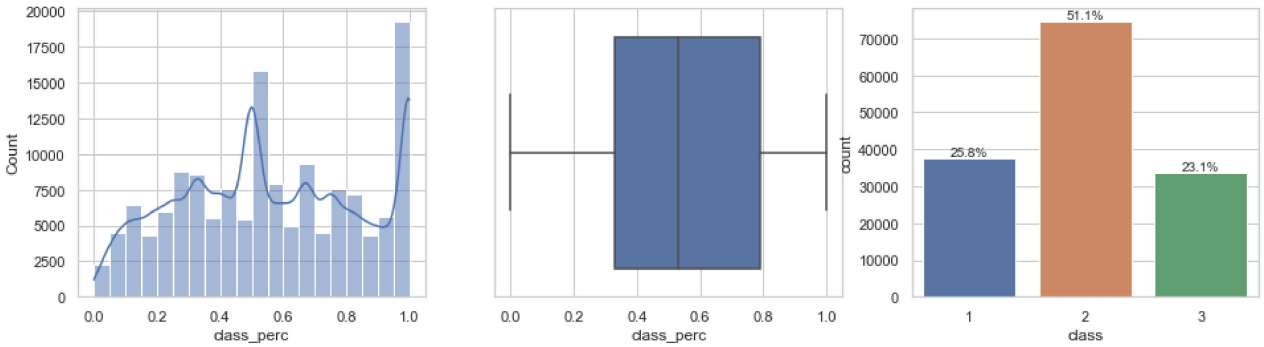
```
fig, ax = plt.subplots(1,3, figsize=(16,4))

sns.histplot(x=df['designation_perc'], bins=20, kde=True, ax=ax[0])
sns.boxplot(x=df['designation_perc'], ax=ax[1])
sns.countplot(x=df['designation'], ax=ax[2])
showpercent(ax[2], df.shape[0])

plt.show()
```



class_perc and class

In [415…

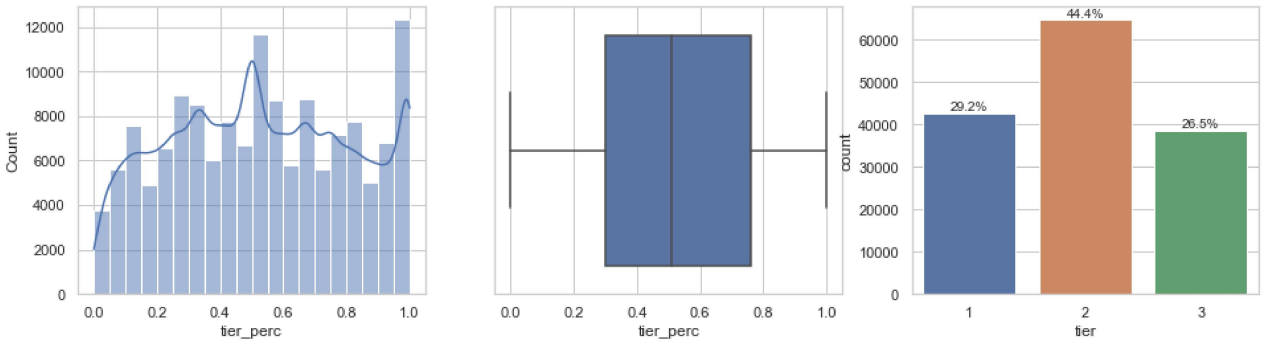
```
fig, ax = plt.subplots(1,3, figsize=(16,4))

sns.histplot(x=df['class_perc'], bins=20, kde=True, ax=ax[0])
sns.boxplot(x=df['class_perc'], ax=ax[1])
sns.countplot(x=df['class'], ax=ax[2])
showpercent(ax[2], df.shape[0])

plt.show()
```



tier_perc and tier

In [416...

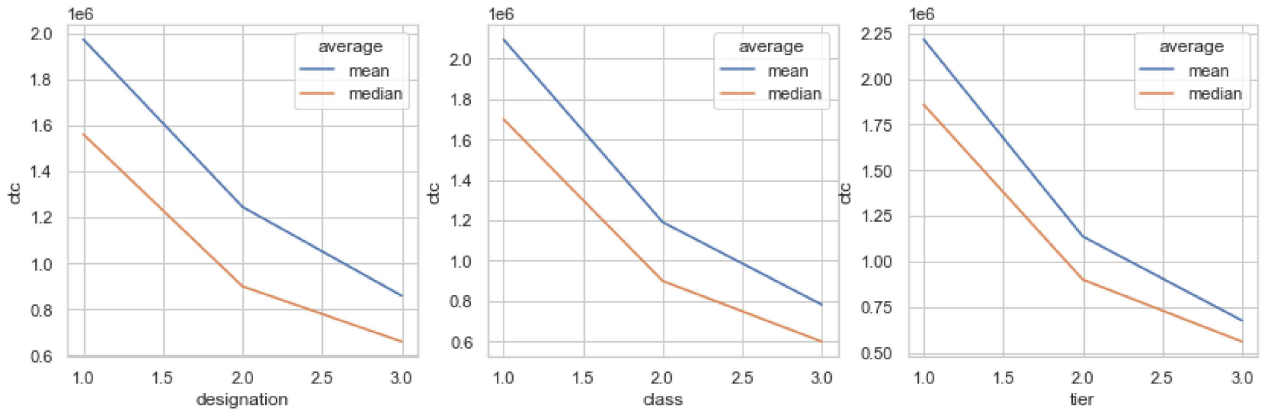
```
fig, ax = plt.subplots(1,3, figsize=(16,4))

sns.histplot(x=df['tier_perc'], bins=20, kde=True, ax=ax[0])
sns.boxplot(x=df['tier_perc'], ax=ax[1])
sns.countplot(x=df['tier'], ax=ax[2])
showpercent(ax[2], df.shape[0])

plt.show()
```



designation, class, and tier and their average CTC

In [314...

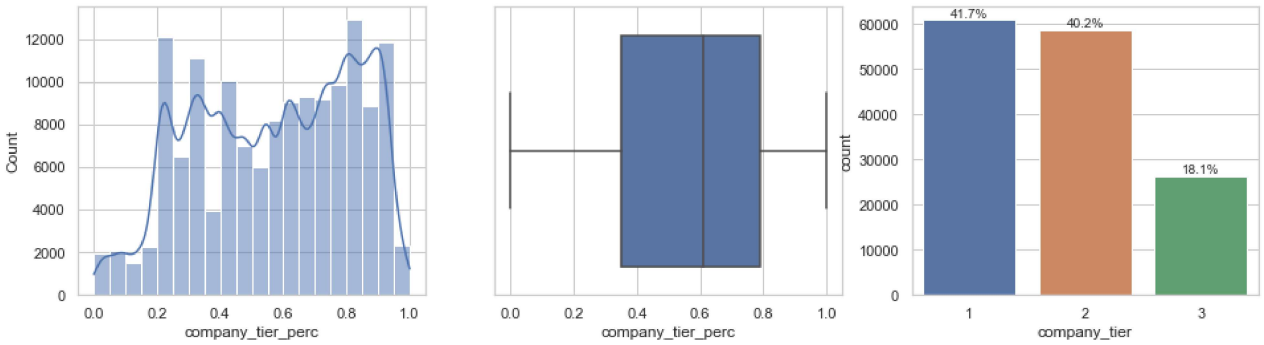
```
cols = ['designation', 'class', 'tier']
fig, ax = plt.subplots(1, 3, figsize=(14, 4))

for i, col in enumerate(cols):
    data = df.groupby(col)['ctc'].agg(['mean', 'median']).reset_index()
    data = data.sort_values(by=col)
    data = data.set_index(col)[['mean', 'median']].stack().reset_index().rename(columns

    sns.lineplot(x=col, y='ctc', hue='average', data=data, ax=ax[i])
```



company_tier_perc and company_tier

In [417…

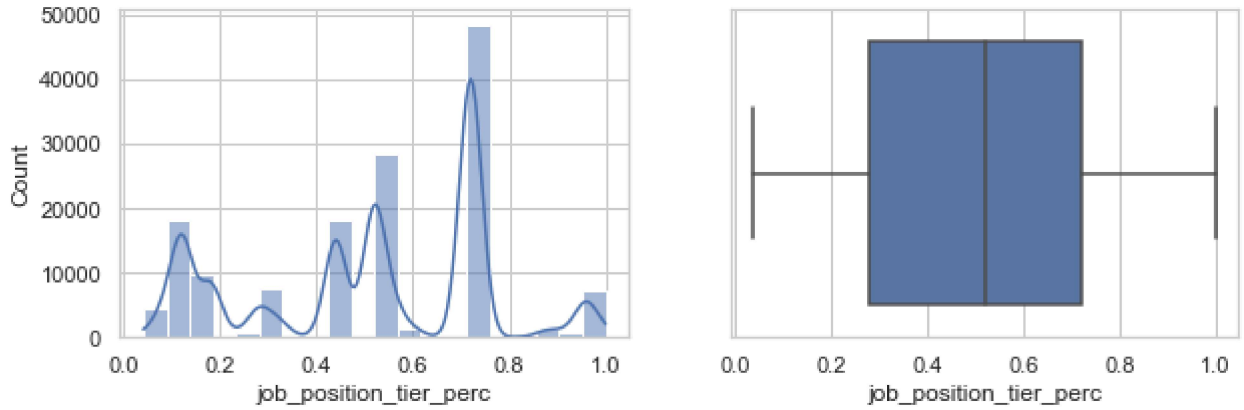
```
fig, ax = plt.subplots(1,3, figsize=(16,4))

sns.histplot(x=df['company_tier_perc'], bins=20, kde=True, ax=ax[0])
sns.boxplot(x=df['company_tier_perc'], ax=ax[1])
sns.countplot(x=df['company_tier'], ax=ax[2])
showpercent(ax[2], df.shape[0])

plt.show()
```



job_position_tier

In [183…

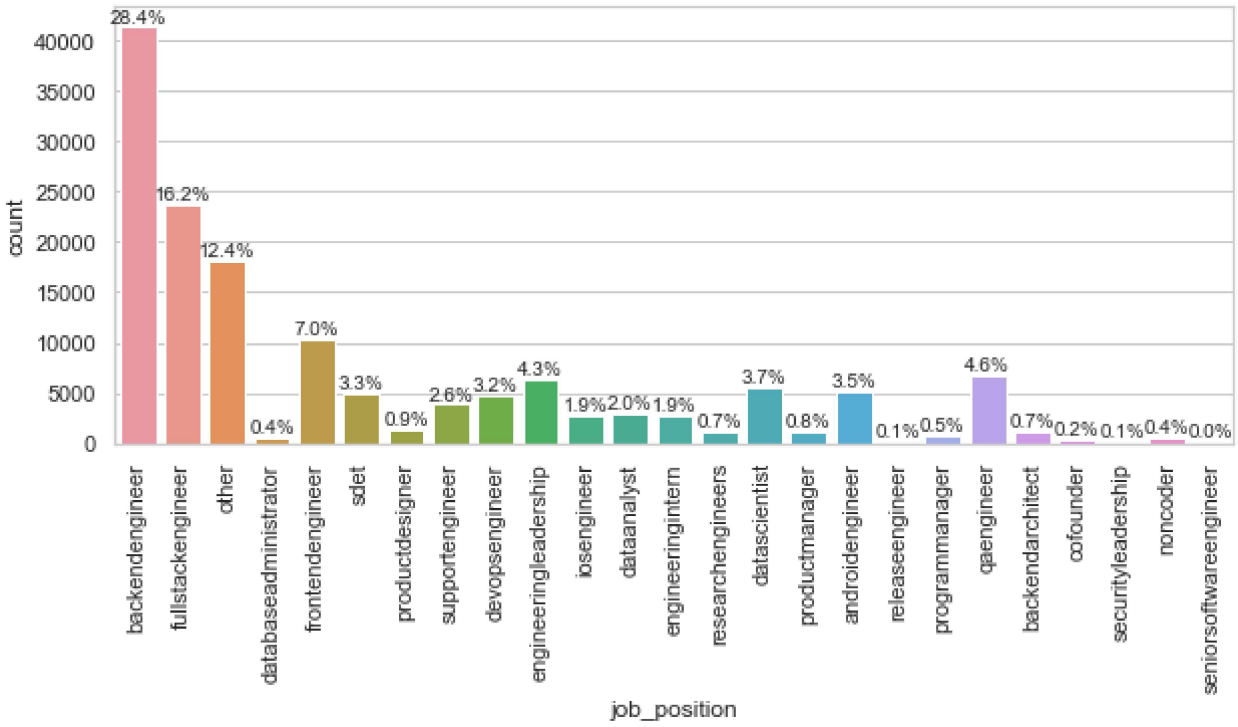
```
fig, ax = plt.subplots(1,2, figsize=(10,3))
sns.histplot(x=df['job_position_tier_perc'], bins=20, kde=True, ax=ax[0])
sns.boxplot(x=df['job_position_tier_perc'], ax=ax[1])
plt.show()
```



job_position

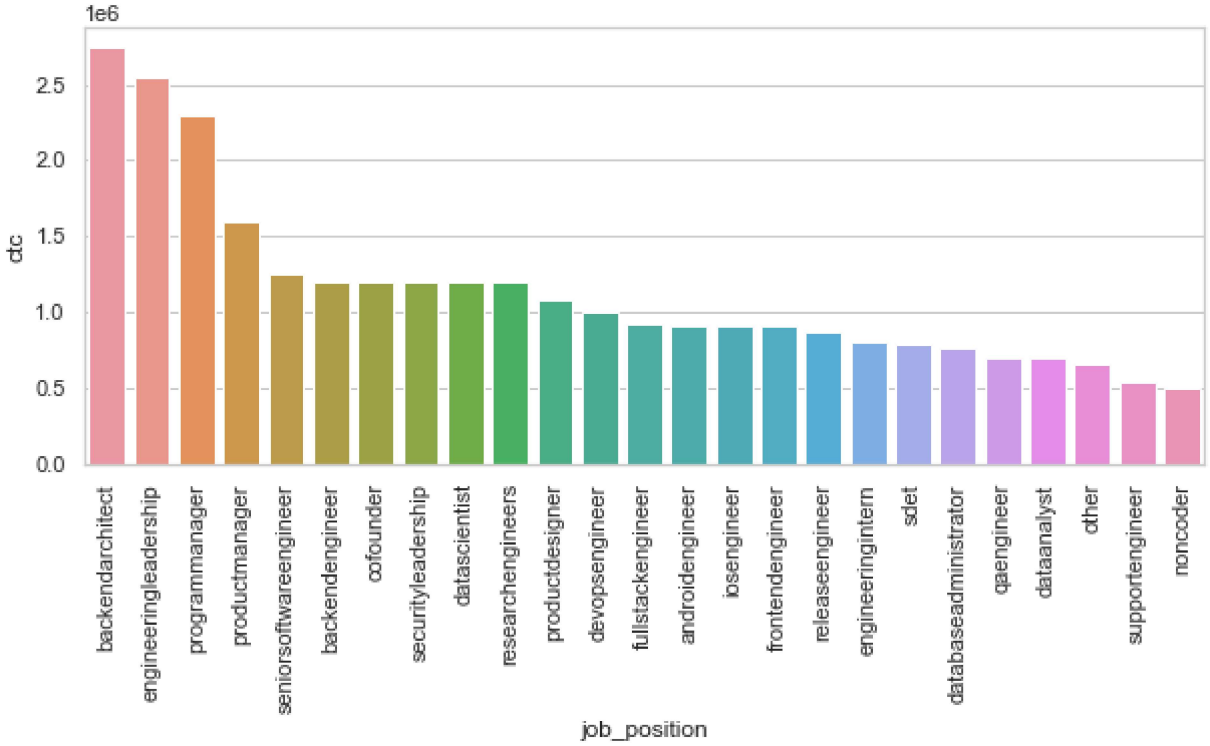
In [421…

```
fig, ax = plt.subplots(1,1, figsize=(10,4))
sns.countplot(x=df['job_position'], ax=ax)
showpercent(ax, df.shape[0])
plt.xticks(rotation = 90)
plt.show()
```



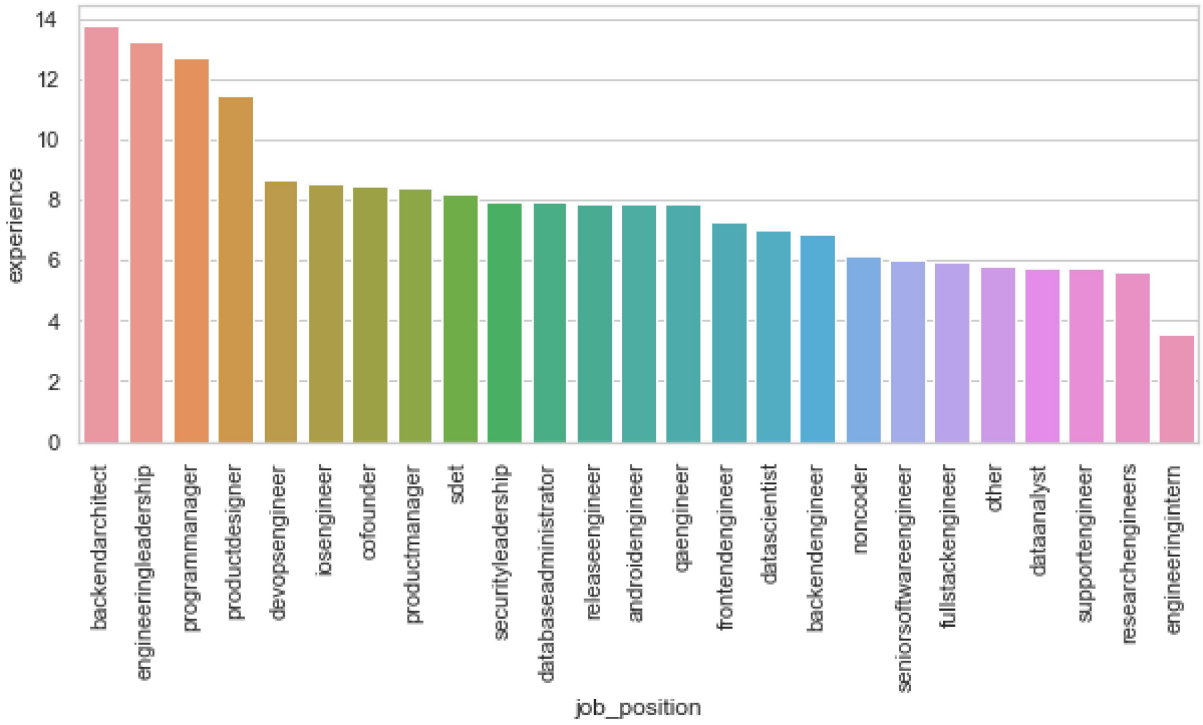
In [423…

```
data = ctc_by_jp.sort_values(by='ctc', ascending=False)
x = data['job_position']
y = data['ctc']

fig, ax = plt.subplots(1,1, figsize=(10,4))
sns.barplot(x=x, y=y, ax=ax)
plt.xticks(rotation = 90)
plt.show()
```



In [428…

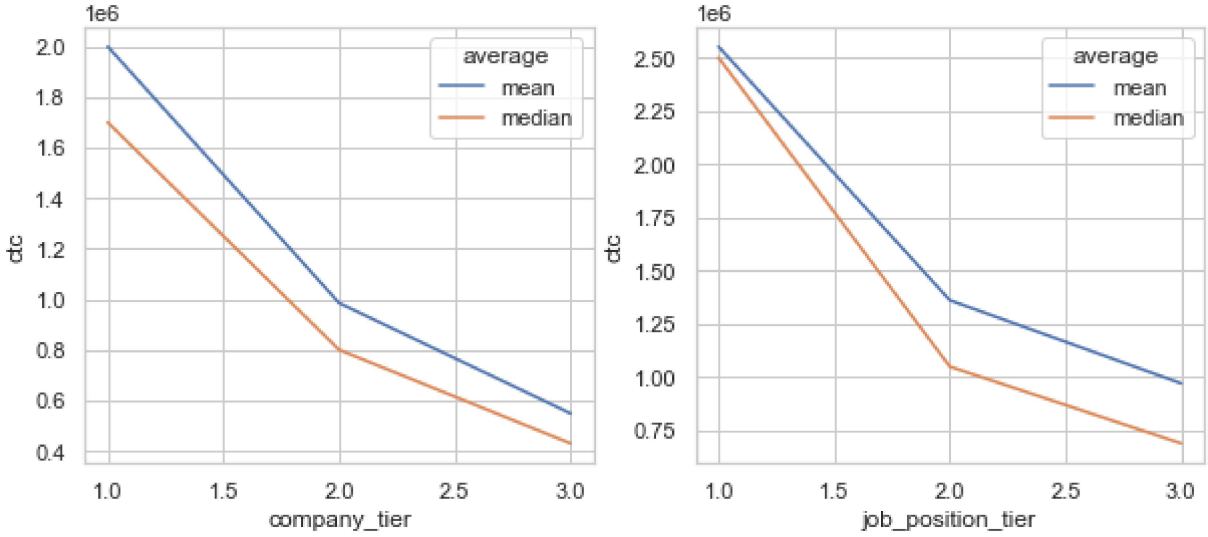
```
data = df.groupby('job_position')['experience'].mean().sort_values(ascending=False).res

x = data['job_position']
y = data['experience']

fig, ax = plt.subplots(1,1, figsize=(10,4))
sns.barplot(x=x, y=y, ax=ax)
plt.xticks(rotation = 90)
plt.show()
```



company_tier, job_position_tier and average ctc

In [316…

```
cols = ['company_tier', 'job_position_tier']
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

for i, col in enumerate(cols):
    data = df.groupby(col)['ctc'].agg(['mean', 'median']).reset_index()
    data = data.sort_values(by=col)
    data = data.set_index(col)[['mean', 'median']].stack().reset_index().rename(columns

    sns.lineplot(x=col, y='ctc', hue='average', data=data, ax=ax[i])
```



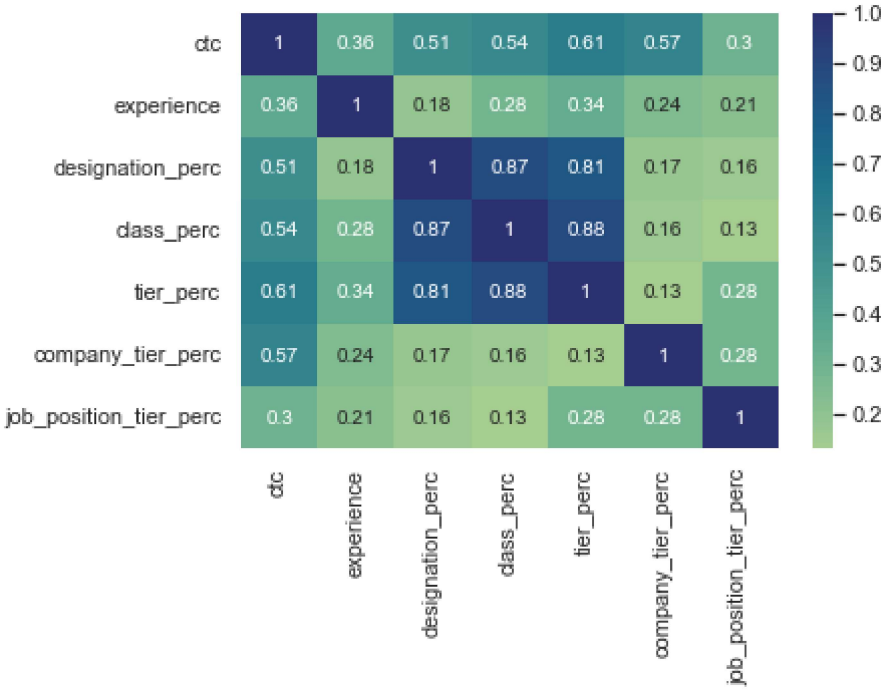
Correlation among continuous features

In [317…

```
data = np.round(df[['ctc', 'experience', 'designation_perc', 'class_perc', 'tier_perc',
sns.heatmap(data, cmap="crest", annot=True)
```

Out[317…

```
<AxesSubplot:>
```



Observations:

1. Distribution of ctc is positively skewed, however, ctc_log distribution is symmetrical. For clustering, we will use ctc_log
2. In general, mean and median income for learners increase with experience, the exception being 1-2 years experience range where average income dips from 0-1 experience group. This is somewhat unexpected.
3. For designation, class, and tier features, as the level increase from 1 to 3, their mean/median CTC drops considerably. This is entirely expected because of the way we computed these features.
4. Similarly, for company_tier, and job_position_tier features, as the level increase from 1 to 3, their mean/median CTC drops considerably.
5. Top 5 lucative job_positions are 'backend architect', 'engineering leadership', 'program manager', 'product manager', and 'senior software enginner'. Least lucrative job_positions include 'non-coders', 'support engineers', 'data analyst', and 'qa engineers'. However, there is a catch. Most of the highest paying job_positions also have highest average experience, similary, most of the low paying job_positions are also associated with low experience.
6. There is a strong positive correlation among designation, class, and tier percentile features.
7. designation, class, tier, and company_tier are all moderately positivly correlated with CTC.
8. There is a weak positive correlation between experience and CTC.

Manual clustering

Based on the features derived in the previous section, we can create manual clusters of learners as shown below.

Clusters based on 'designation'

In [109…
```
np.round(df.groupby(['designation']).agg({'email_hash':'nunique', 'ctc(lacs)': ['mean',
```

Out[109…

	email_hash	ctc(lacs)		experience
	nunique	mean	median	mean
designation				
1	33064	19.72	15.6	6.82
2	82669	12.45	9.0	7.42
3	30185	8.59	6.6	6.67

Interpretation:

Designation 1 are highest paid learners among their peers with their average CTC being significantly higher than the rest of their peers; Their average experience, on the other hand, is only marginally more than designation 3 employees and lower than that of designation 2 employees. Thus designation 1 learners are the brightest and highest performing learners. Designation 2 learners are relatively lesser paid learners with average experience relatively higher than the other groups. Thus they represent average learners with potential to grow. Designation 3 employees are paid the lowest among their peers, possibly an indication of weaker performance or lack of necessary skills.

Clusters based on 'class'

In [110…
```
np.round(df.groupby(['class']).agg({'email_hash': 'nunique','ctc(lacs)': ['mean', 'medi
```

Out[110…

	email_hash	ctc(lacs)		experience
	nunique	mean	median	mean
class				
1	37667	20.95	17.0	8.13
2	74543	11.91	9.0	7.16
3	33708	7.83	6.0	5.95

Interpretation:

Class1 learners consists of higher experience (average experience of 8.1 years) professional with significantly higher average pay than their peers working in the same company and job_position. Class2 learners have relatively lesser average experience (7.1 years) and lower average pay than their class1 peers. Class3 learners are lowest paid but also has relatively lowest experience.

Clusters based on 'tier'

In [111…
```
np.round(df.groupby(['tier']).agg({'email_hash': 'count', 'ctc(lacs)': ['mean', 'median
```

Out[111…

	email_hash	ctc(lacs)		experience
	count	mean	median	mean
tier				
1	42571	22.18	18.6	8.74
2	64741	11.36	9.0	6.93
3	38606	6.76	5.6	5.69

Interpretation:

Tier1 learners are highest paid employees in their company and also happen to be most experienced employees. Tier2 employees are paid relatively lesser than tier1 peers but also have lower exprience. Tier3 group consists of learners who are paid the lowest in their company and have the lowest average experience.

Clusters based on 'company_tier'

In [112…
```
np.round(df.groupby(['company_tier']).agg({'email_hash': 'nunique', 'ctc(lacs)': ['mean
```

Out[112…

	email_hash	ctc(lacs)		experience
	nunique	mean	median	mean
company_tier				
1	60878	20.00	17.0	8.20
2	58672	9.86	8.0	6.69
3	26368	5.48	4.3	5.64

Interpretation: Learners can be clustered into 3 groups based on the tier of their companywith Tier1 companies offering the highest mean pay and tier3 offering the lowest mean pay.

Insights from Manual clustering

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

In [113…
```
def compute_rank(d):
    d['rank'] = np.array(range(1, d.shape[0]+1))
    return d

tier1_learners = df[df['tier'] == 1].copy()
ranked_learners = tier1_learners.sort_values(by=['ctc'], ascending=[False]).groupby('co
top_10 = ranked_learners[ranked_learners['rank'] <= 10]
top_10.sort_values(by=['company_hash', 'rank'], ascending=[False, False])[['company_has
```

Out[113...

	company_hash	email_hash	rank	ctc	experience
71283	zz	7d4588453bc463b39db8c77ef0f856957fc42f5d54cae4...	1	1370000.0	9.0
135071	zycoxzaxv	ed0a1202c31bdee343662f5517fc467fb8b96ccaf8e3eb...	1	600000.0	9.0
32299	zxztrtvuo	3879b9a1e356ed20363fffd6871207eb908b38c864a2db...	10	1500000.0	5.0
13259	zxztrtvuo	16c227291d7c4f151b52599cc15e1ddd6f7e12a694753c...	9	1780000.0	7.0
29442	zxztrtvuo	3385dc93ba44f4f1cc237ef4f8e057dab2f693d8961b64...	8	1800000.0	9.0
...	...	...	...	...	...
28458	1bs	31db7b806a82aac024462d4c97e70eed918bf8c3193b7c...	1	3750000.0	6.0
28964	159ogrhnxgzo	32b7af6a22321bae56df8b23ed279c1ec83098b2d42690...	1	800000.0	4.0
12872	123ongqto	160c93562f03a86198438f7b0ba964d1031e7b8c13c9d2...	1	2560000.0	6.0
98035	10nxbto	abf1baa6163b89982e1a66ea5eca355109946e5030f342...	1	500000.0	3.0
128693	01ojztqsj	e1e15fada844f35fcc33927343d0c80f55526b87c40eee...	1	830000.0	11.0

20127 rows × 5 columns

2. Top 10 employees of data science in Amazon / TCS etc earning more than their peers - Class 1

Note: we will take 'zvz' company_hash in our solution.

In [114...

```
learners = df[(df['class'] == 1) & (df['company_hash'] == 'zvz') & (df['job_position']
learners.sort_values(by='ctc', ascending=False)[['email_hash', 'ctc', 'experience', 'cl
```

Out[114...

	email_hash	ctc	experience	class	designation
24023	29d87536746f4ce38d9404695b4f0686c07fd9864d30b1...	8360000.0	9.0	1	1
23342	28ab43a2ffc7954f01657b9a366a7eac703a4573388f73...	4500000.0	4.0	1	1
54825	601750bf98bc9ecac24f0c1cf2f74f4cdd7e159666a778...	4000000.0	4.0	1	1
70124	7b48a8e29d116bb1ed6c2728da80c20b426c5041570286...	3200000.0	4.0	1	1
90766	9f46b2ea1ed7e21fbdc6c92207e9c71c59a3c9ca435b2f...	3000000.0	1.0	1	1
118219	cf83c09b40f463a93d7c66f6de83c84a3dbea0a7df0652...	2800000.0	3.0	1	1
116684	cccd80969d18917f68b1d38cdbd474e8111091e73b1fbf...	2800000.0	2.0	1	1
101609	b2356a9d3645fea65d5b8da3c7333dcdda01dac326618c...	2800000.0	2.0	1	1
145630	ff72ad9805c95aa037370c3a69b266fd99295a6ce005ec...	2800000.0	2.0	1	1
32830	3962a72ca309a1c3cb70d31505aaa0a57ece5240dc25ba...	2100000.0	1.0	1	1

3. Bottom 10 employees of data science in Amazon / TCS etc earning less than their peers - Class 3

In [115...

```
learners = df[(df['class'] == 3) & (df['company_hash'] == 'zvz') & (df['job_position']
learners.sort_values(by='ctc', ascending=True)[['email_hash', 'ctc', 'experience', 'cla
```

Out[115...

	email_hash	ctc	experience	class	designation	tier
83455	926cb078906d1416b04d68df05c641642f939a613a8c33...	107000.0	7.0	3	3	3
72310	7f01767cc729d905608caf3eb7d6d143e71099b2541cd2...	125000.0	9.0	3	3	3
21624	25a5a06c9c1a539f0ca0126b5988e9e0f51c4bcf31f070...	300000.0	1.0	3	3	3
78706	8a479eb07313169f961435fb2371ae3c4a41ebc34399b9...	450000.0	3.0	3	3	3
86368	9789527614586b84f9227f94152d4b6e793b4560a5e122...	450000.0	3.0	3	3	3
98459	acb372c4754c5eea79b36cbedca284eb32d6b3197fa76a...	500000.0	1.0	3	3	3
94817	a663d2ad8d23cb4f31d51883b390bb6361803469ef62bf...	530000.0	2.0	3	3	3
78687	8a3db5a732b242357fbc5540a8b5b0a03b9a69c6e4ca55...	600000.0	3.0	3	3	3
96850	a9e714eb43900f8c3e3d79c2a8452609d8d2e72196afd9...	600000.0	3.0	3	3	3
41373	48b32fa78c9928807111223efd6801f94dceec569ec22f...	600000.0	1.0	3	3	3

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

In [116...

```
def compute_rank(d):
    d['rank'] = np.array(range(1, d.shape[0]+1))
    return d

tier3_learners = df[df['tier'] == 3].copy()
ranked_learners = tier3_learners.sort_values(by=['ctc'], ascending=[True]).groupby('com
top_10 = ranked_learners[ranked_learners['rank'] <= 10]
top_10.sort_values(by=['company_hash', 'rank'], ascending=[False, True])[['company_hash
```

Out[116...

	company_hash	email_hash	rank	ctc	exp
122267	zz	d6923a6f81c7b36615d9f14349fe01aec442029b2c502f...	1	500000.0	
54051	zxzvnxgzvrxzonqhbtzno	5ece45aea666b6252a7dd88f3da824efd44dab8c28f990...	1	650000.0	
69170	zxztrtvuo	798f312433ee4125ee5dd5887250701eb533249e8fbad6...	1	400000.0	
116896	zxztrtvuo	cd2e14f599f7749ac2be21c9ee219f10e46df8bac74b6d...	2	400000.0	
143788	zxztrtvuo	fc27f95d1dff67864b5a46f7417c3231997d9ba893c3ac...	3	400000.0	
...	...	...	...	...	
61008	1bs	6b01808bba4c2d50258b068274232251620630cb252a9c...	9	900000.0	
12614	1bs	158e0bba9db45a85e623c23b5c66d16e5daadcc8d0e4b4...	10	900000.0	
30037	159ogrhnxgzo	34899d76060c1f9885967e7b07557104a93d67690d5378...	1	500000.0	
84939	10nxbto	951189c83134ac5394c85c66d0e96f2a0a242f3d99b87d...	1	400000.0	
73766	01ojztqsj	819789ff4068fd5c8facf8a5074cdd2e1ff989c95ae02c...	1	270000.0	

18483 rows × 5 columns

5. Top 10 employees in Amazon - X department - having 6-9 years of experience earning more than their peers

Let's assume Amazon is 'zvz' and X is data science deparment.

In [117…

```
def compute_rank(d):
    d['rank'] = np.array(range(1, d.shape[0]+1))
    return d

learners = df[(df['designation'] == 1) & (df['company_hash'] == 'zvz') & (df['experienc
learners.sort_values(by='ctc', ascending=False)[['email_hash', 'ctc', 'experience', 'cl
```

Out[117…

	email_hash	ctc	experience	class	designation
24023	29d87536746f4ce38d9404695b4f0686c07fd9864d30b1...	8360000.0	9.0	1	1
13122	16866399ee80cf82988927b8bae1d07123a7f66406f33f...	5850000.0	6.0	1	1
69704	7a868cd8d5217ff916ace5b7b991bddd89691f4857cece...	5700000.0	6.0	1	1
37351	41850e929e44dc8dc5bb034f30a49d93d7c923a34411b1...	4300000.0	7.0	1	1
41045	4815e84eec35712692738728d4e467348117e404f0d89f...	3200000.0	7.0	1	1
72612	7f8bb5b4a4529ba1d2806406bd2377eabbc9916a546671...	3150000.0	6.0	1	1
144981	fe4186d26ddedc817814cbb9c95a1ccf9663324b3cefc4...	3100000.0	6.0	1	1
110512	c1f6cdd226c3e2d0c8953f3697dc9186fd1f0765905d30...	3000000.0	7.0	1	1
3017	0542c602b6ffe3d8e702dfcbc2d560752f433498f26732...	3000000.0	6.0	1	1
137896	f1e75824ef2041c3c63d2ea75d4bf93ea8d854b11e8c06...	2830000.0	7.0	1	1

6. Top 10 companies (based on their CTC)

In [118…

```
#By median CTC
ctc_by_c.sort_values(by=['ctc_median', 'ctc_count'], ascending=[False, False]).head(10)
```

Out[118…

	company_hash	ctc_count	ctc_min	ctc_max	ctc_mean	ctc_median	perc33	per
13531	ogstnxhov	1	9100000.0	9100000.0	9100000.0	9100000.0	9100000.0	91000(
1143	avnvbvzvstbtzn	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000(
2588	btqwtatomtzkqa	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000(
6286	ftrrztoo	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000(
7570	hmovrrxvzwt	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000(
8051	ic21ntwyzgrgsxto	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000(
9162	lmyhznnqvzougqnxzw	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000(
16165	oxzsvugqttdwyvzst	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000(

	company_hash	ctc_count	ctc_min	ctc_max	ctc_mean	ctc_median	perc33	per
16695	pqtntpxzntqzvnxgzvr	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
17849	qtrxvzwtlxguvjbtznmvzp	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000

In [119…
```
#By mean CTC
ctc_by_c.sort_values(by=['ctc_mean', 'ctc_count'], ascending=[False, False]).head(10)
```

Out[119…

	company_hash	ctc_count	ctc_min	ctc_max	ctc_mean	ctc_median	perc33	per
13531	ogstnxhov	1	9100000.0	9100000.0	9100000.0	9100000.0	9100000.0	91000
1143	avnvbvzvstbtzn	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
2588	btqwtatomtzkqa	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
6286	ftrrztoo	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
7570	hmovrrxvzwt	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
8051	ic21ntwyzgrgsxto	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
9162	lmyhznnqvzougqnxzw	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
16165	oxzsvugqttdwyvzst	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
16695	pqtntpxzntqzvnxgzvr	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000
17849	qtrxvzwtlxguvjbtznmvzp	1	9000000.0	9000000.0	9000000.0	9000000.0	9000000.0	90000

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

In [120…
```
ctc_by_c_jp.sort_values(by=['company_hash', 'ctc_median'], ascending=[False, False]).gr
```

Out[120…

	company_hash	job_position	ctc_count	ctc_min	ctc_max	ctc_mean	ctc_m
55081	zzgato	fullstackengineer	1	130000.0	130000.0	130000.0	130
55080	zzbztdnstzvacxogqjucnrna	fullstackengineer	1	600000.0	600000.0	600000.0	600
55079	zz	other	2	500000.0	1370000.0	935000.0	935
55078	zyvzwtwgzohrnxzstzsxzttqo	frontendengineer	1	940000.0	940000.0	940000.0	940
55077	zyvzwtfggaqtztfvreqvzwyxogqyi	frontendengineer	1	900000.0	900000.0	900000.0	900
...	...	...	...	...	...	...	
4	1	other	1	250000.0	250000.0	250000.0	250
3	05mzexzytvrnyuqxcvntrxbxnta	backendengineer	1	1100000.0	1100000.0	1100000.0	1100
2	01ojztqsj	frontendengineer	1	830000.0	830000.0	830000.0	830
1	01ojztqsj	androidengineer	1	270000.0	270000.0	270000.0	270
0	0000	other	1	300000.0	300000.0	300000.0	300

41682 rows × 9 columns

Unsupervised learning: Clustering

In this section, we will use KMeans and Agglomerative Hierarchical clustering algorithms to identify clusters of learners in our data-set. We use the following features as input. We will standardize all features before running algorithms.

1. ctc_log - log transformed ctc
2. experience
3. designation_perc, class_perc, tier_perc - indicates a learner's performance/standing by percentile rank
4. company_tier_perc - company's standing in terms of ctc
6. job_position_tier_perc - job_position's tier in terms of median CTC

No of Clusters - We will use Elbow method for KMeans.

Cluster evaluation: We will use silhouette score to evaluate clusters obtained from different algorithms. For KMeans, we will also report WCSS score.

cluster visualization: We will use PCA/TSNE for visualizing clusters in lower dimensional space (2d or 3d)

Sample size: For KMeans, we will use the entire data-set. For Hierarchical clustering and for visualizing clusters, we will use a smaller sample.

Cluster interpretation: We will plot graphs of various features for the given clusters to identify core characteristics for each cluster.

Data prepration for clustering

In [320…

```
df_final = df.copy()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145918 entries, 0 to 145917
Data columns (total 18 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   email_hash             145918 non-null  object 
 1   ctc                    145918 non-null  float64
 2   ctc_log                145918 non-null  float64
 3   experience             145918 non-null  float64
 4   experience_level       145918 non-null  object 
 5   designation            145918 non-null  int64  
 6   class                  145918 non-null  int64  
 7   tier                   145918 non-null  int64  
 8   job_position_tier      145918 non-null  int64  
 9   company_tier           145918 non-null  int64  
```

```
 10  designation_perc        145918 non-null  float64
 11  class_perc              145918 non-null  float64
 12  tier_perc               145918 non-null  float64
 13  job_position_tier_perc  145918 non-null  float64
 14  company_tier_perc       145918 non-null  float64
 15  company_hash            145918 non-null  object
 16  job_position            145918 non-null  object
 17  ctc(lacs)               145918 non-null  float64
dtypes: float64(9), int64(5), object(4)
memory usage: 20.0+ MB
```

In [322…

```
#Remove unnecessary features for clustering
df.drop(['email_hash', 'ctc(lacs)', 'ctc', 'company_hash', 'job_position', 'experience_
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145918 entries, 0 to 145917
Data columns (total 7 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   ctc_log                 145918 non-null  float64
 1   experience              145918 non-null  float64
 2   designation_perc        145918 non-null  float64
 3   class_perc              145918 non-null  float64
 4   tier_perc               145918 non-null  float64
 5   job_position_tier_perc  145918 non-null  float64
 6   company_tier_perc       145918 non-null  float64
dtypes: float64(7)
memory usage: 7.8 MB
```

In [323…

```
### scaling data
scaler = StandardScaler()
cols_to_scale = df.columns
scaled_vals = scaler.fit_transform(df[cols_to_scale])
df.loc[:,cols_to_scale] = scaled_vals
```

In [324…

```
df.describe()
```

Out[324…

	ctc_log	experience	designation_perc	class_perc	tier_perc	job_position_tier_p
count	1.459180e+05	1.459180e+05	1.459180e+05	1.459180e+05	1.459180e+05	1.459180e+
mean	3.992962e-15	-7.601236e-17	3.087241e-16	1.811441e-16	-1.934152e-16	3.605353e-
std	1.000003e+00	1.000003e+00	1.000003e+00	1.000003e+00	1.000003e+00	1.000003e+
min	-2.905403e+00	-1.733243e+00	-2.011232e+00	-1.957953e+00	-1.867207e+00	-1.871351e+
25%	-6.341463e-01	-7.608342e-01	-8.531569e-01	-7.992261e-01	-8.061129e-01	-9.103637e-
50%	1.880839e-02	-2.746299e-01	-1.162002e-01	-9.696748e-02	-6.334693e-02	5.062342e-
75%	6.970755e-01	4.546766e-01	8.313155e-01	8.159688e-01	8.208983e-01	8.514460e-
max	2.856858e+00	4.344311e+00	1.498086e+00	1.553340e+00	1.669774e+00	1.972598e+

Checking clustering tendency

We will use hopkins statistical test to assess the clusterability of our dataset. A score between 0 and 1, a score around 0.5 express no clusterability and a score tending to 0 express a high cluster tendency.

In [325…

```
#!pip install pyclustertend --user

hopkins(df, df.shape[0])
```

Out[325…

```
0.11104170931685044
```

Observation: We got a score close to 0, therefore, out dataset is clusterable.

Define helper functions

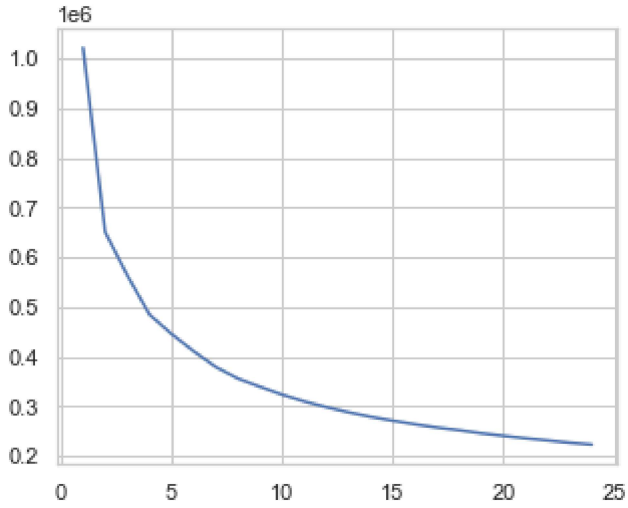
In [364…

```
#Define helper functions

def plot_elbow(clusters_range, sse):
    fig, ax = plt.subplots(1,1, figsize=(5,4))
    sns.lineplot(x=list(clusters_range), y=sse, ax=ax)
    plt.show()

def evaluate_wcss(df, cluster_range=range(1,10), max_ter=100):
    sse = []
    for k in cluster_range:
        kmeans = KMeans(n_clusters=k, max_iter=100).fit(df)
        sse.append(kmeans.inertia_) # Inertia: Sum of distances of samples to their clo
    return sse

def visualize_pca(X, y=None, ndim=3):
    pca = PCA(n_components=ndim)
    pca.fit(X)
    pca_df = pd.DataFrame(pca.fit_transform(X))
    pca_df['cluster'] = y

    fig=None
    if ndim == 3:
        fig = px.scatter_3d(pca_df, x=0, y=1, z=2, color='cluster', width=600, height=6
    else:
        fig = px.scatter(pca_df, x=0, y=1, color='cluster', width=600, height=600)
    fig.show()

def visualize_tsne(X, y=None, perplexity=200, method='exact'):

    tsne_val = TSNE(n_components=2, random_state=0, init='random', learning_rate='auto'
    tsne_df = np.round(pd.DataFrame(tsne_val),2)
    fig = px.scatter(tsne_df, x=0, y=1, color=y, width=600, height=600)
    fig.show()


def cal_silhouette_score(df, labels, metric='euclidean'):
    return silhouette_score(df, labels, metric=metric)
```

Determining no of clusters using Elbow criterion

In [329…
```
def try_elbow(df):
    cluster_range = range(1,25)
    sse_list = []
    for rep in range(1):
        df_sample = df#.sample(n=10000)
        sse_list.append(evaluate_wcss(df_sample, cluster_range))

    avg_sse = np.array(sse_list).mean(axis=0)
    plot_elbow(cluster_range, avg_sse)
```

In [330…
```
try_elbow(df)
```



Observations: Based on the Elbow graph, **either 4, 5, 6 or 7** appear to be the points representing bent elbow. We can further check their Silhouette score to decide number of clusters.

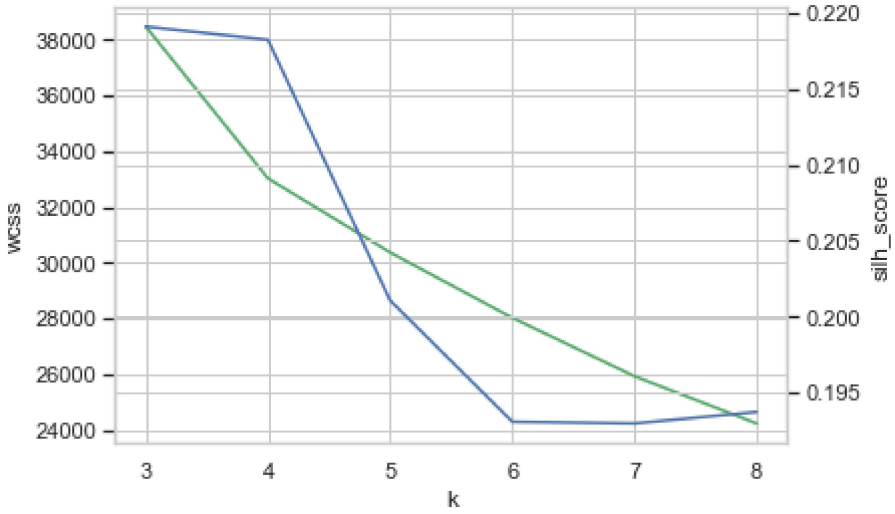
In [331…
```
#take a sample
df_sample = df.sample(n=10000)
```

In [348…
```
k_vals = [3,4,5,6,7,8]
res = []
for k in k_vals:
    km_model = KMeans(n_clusters=k, max_iter=1000).fit(df_sample)
    res.append((k, km_model.inertia_, cal_silhouette_score(df_sample, km_model.labels_)
```

In [357…
```
res = np.array(res)
res_df = pd.DataFrame({'k': res[:,0], 'wcss': res[:,1], 'silh_score': res[:,2]})
```

In [350…
```
sns.lineplot(y=res_df['wcss'], x=res_df['k'], color="g")
ax2 = plt.twinx()
sns.lineplot(y=res_df['silh_score'], x=res_df['k'], color="b")
```

Out[350…
```
<AxesSubplot:xlabel='k', ylabel='silh_score'>
```

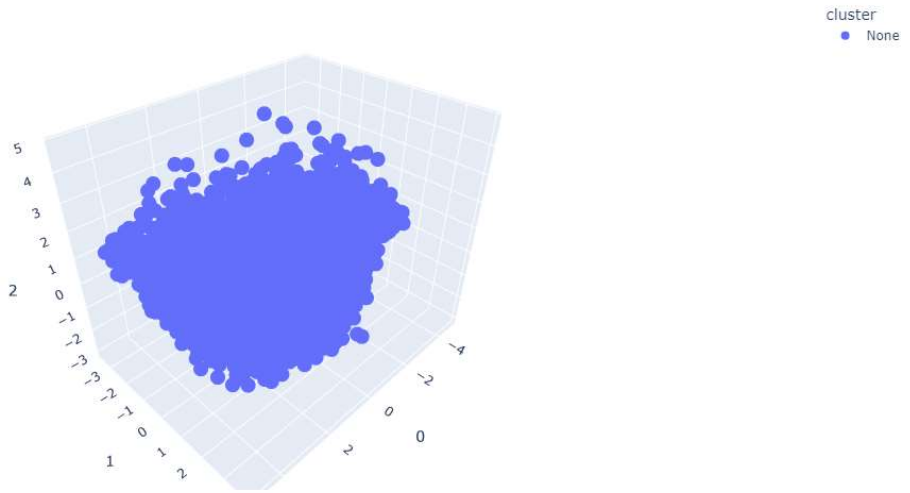


Observations: TODO

Visualizing sample data - PCA and TSNE

In [432…

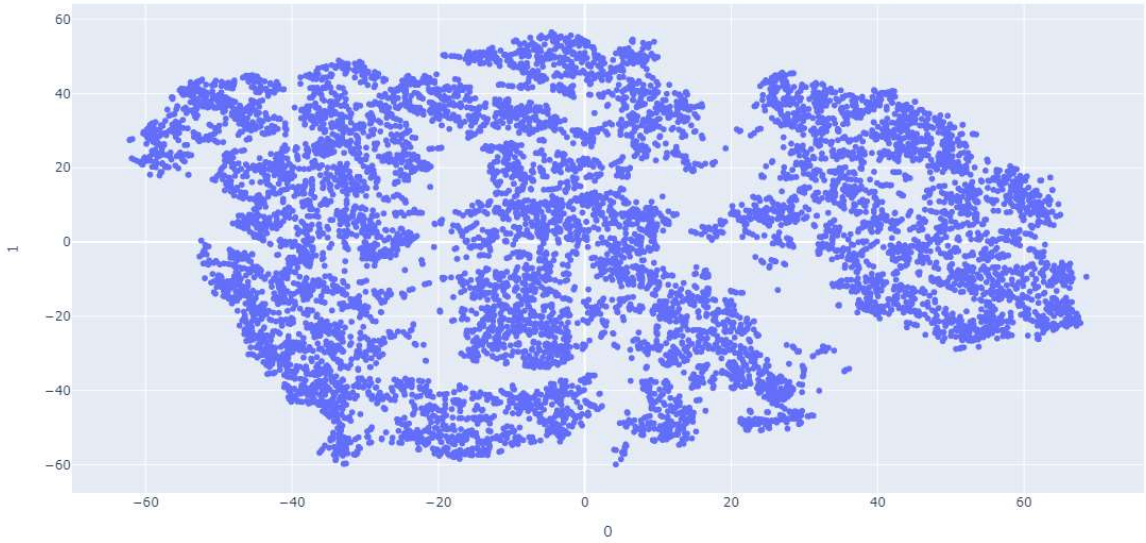
```
visualize_pca(df_sample, ndim=3)
```



observation: 3D PCA visualization isn't revealing any clusters. We can check TSNE visualization next.

In [433…

```
visualize_tsne(df_sample, perplexity=70, method='barnes_hut')
```



Observations: TSNE is revealing a few potential clusters in the data. However they do not appear to be well separated. We will choose k=5 and proceed with kmeans clustering.

KMeans clustering (k=5)

In [383…

```
km_5_sample = KMeans(n_clusters=5, max_iter=1000)
km_5 = KMeans(n_clusters=5, max_iter=1000)

km_vals_sample = km_5_sample.fit_transform(df_sample)
km_vals = km_5.fit_transform(df)
```

check WCSS and Silhouette scores

In [384…

```
silh_score_sample = cal_silhouette_score(df_sample, km_5_sample.labels_)
```

In [386…

```
silh_score = cal_silhouette_score(df, km_5.labels_)
```

In [387…

```
print(f'For sample: WCSS = {km_5_sample.inertia_}, Silhouette score: {silh_score_sample
```

```
For sample: WCSS = 30370.786594598867, Silhouette score: 0.20111120557751513
```

In [388…

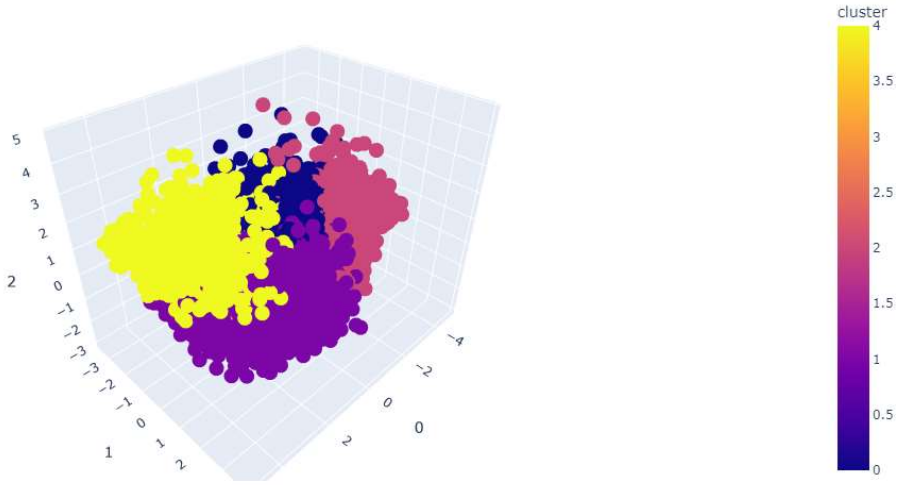
```
print(f'For full dataset: WCSS = {km_5.inertia_}, Silhouette score: {silh_score}')
```

```
For full dataset: WCSS = 445549.58783793234, Silhouette score: 0.2009007398782452
```

Visualize clusters - PCA and TSNE

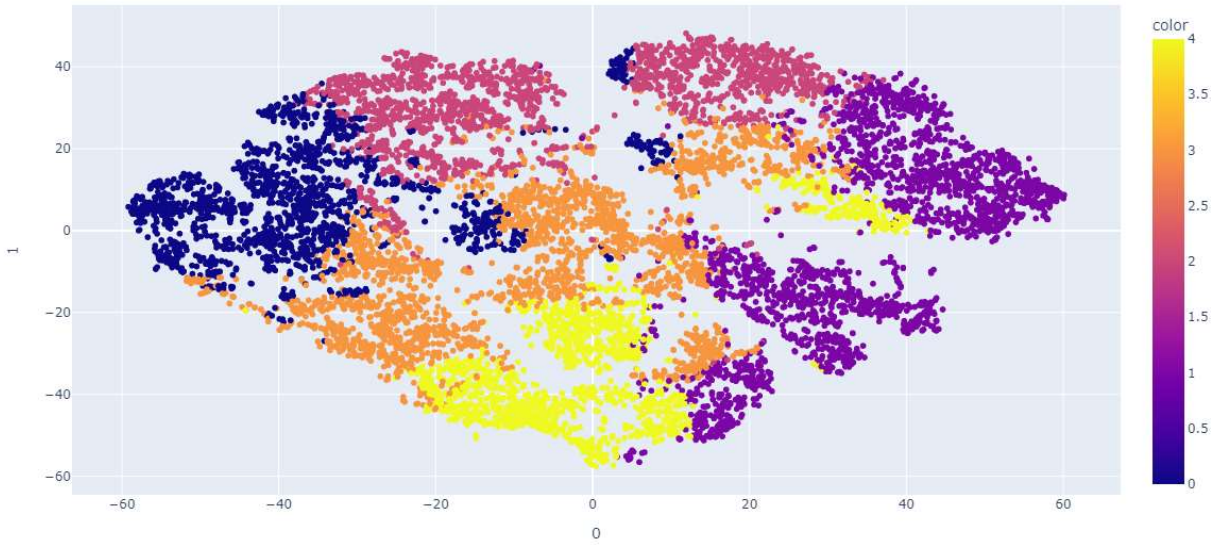
In [434…

```
visualize_pca(df_sample, y=km_5_sample.labels_, ndim=3)
```



In [435…

```
visualize_tsne(df_sample, y=km_5_sample.labels_, perplexity=100, method='barnes_hut')
```



Observation: - The PCA visualization shows cohesive clusters although they are not far apart. TSNE graph, on the other hand, doesn't show clusters well. The clusters appear fragmented and containing multiple sub clusters.

Agglomerative Hierarchical clustering

In [394…

```
hc = AgglomerativeClustering(n_clusters=5)
hc.fit(df_sample)
```

Out[394…

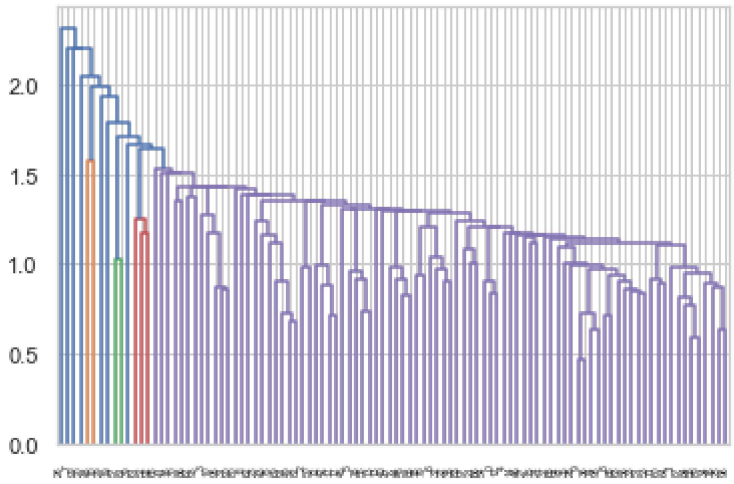
```
AgglomerativeClustering(n_clusters=5)
```

visualize dendrogram (for smaller sample)

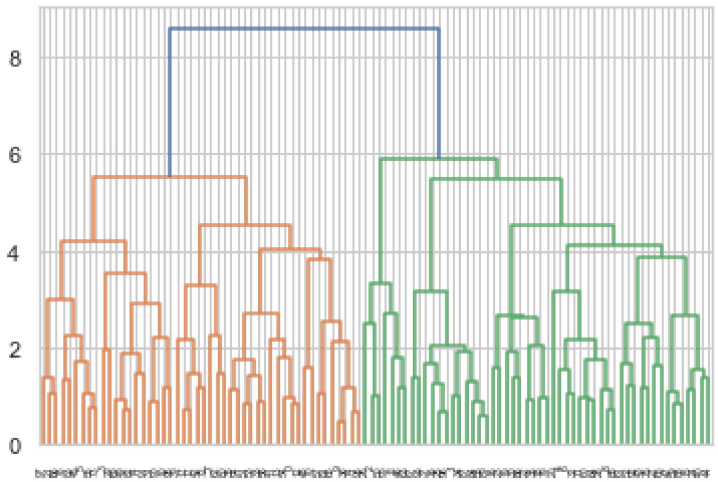
In [401…

```
from scipy.cluster import hierarchy

df_sample2 = df.sample(100) #take a smaller sample for dendrogram visualization

methods =  ['single', 'complete', 'average', 'ward', 'weighted', 'centroid' ]

for method in methods:
    Z = hierarchy.linkage(df_sample2, method)
    ret = hierarchy.dendrogram(Z, leaf_rotation=90)
    print(f'method = {method}')
    plt.show()
```

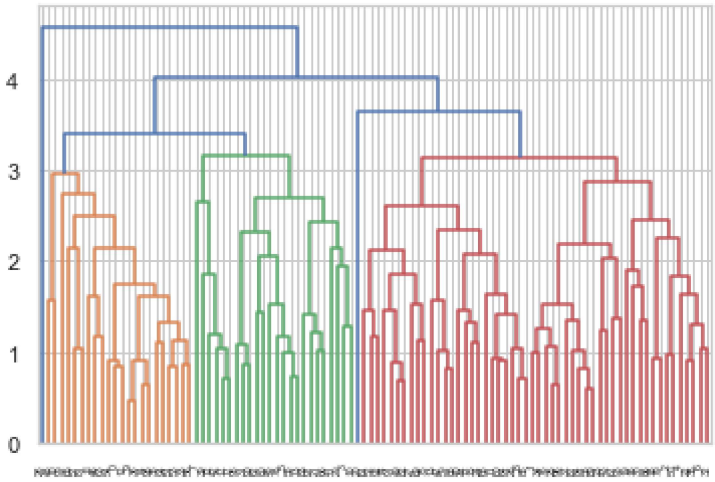
method = single



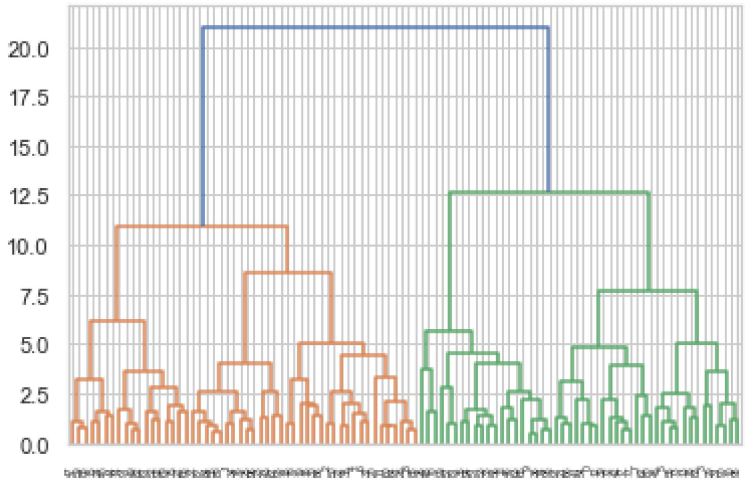
method = complete



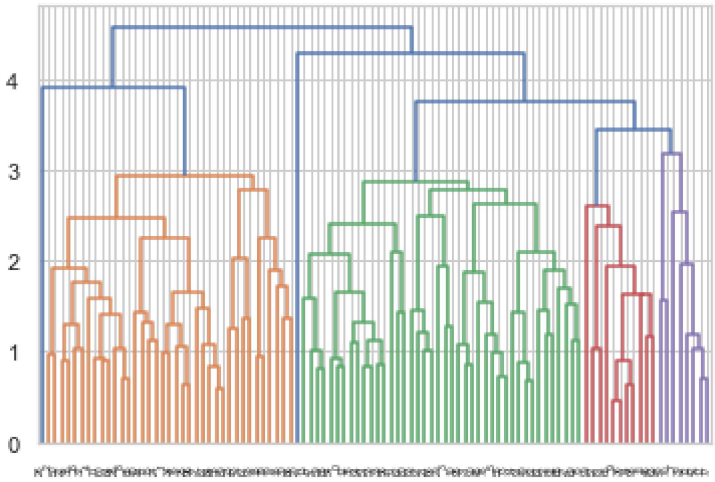
method = average



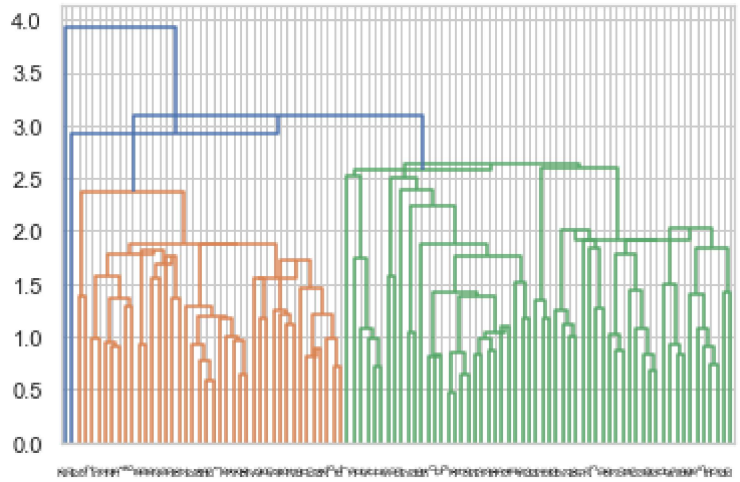
method = ward



method = weighted



method = centroid

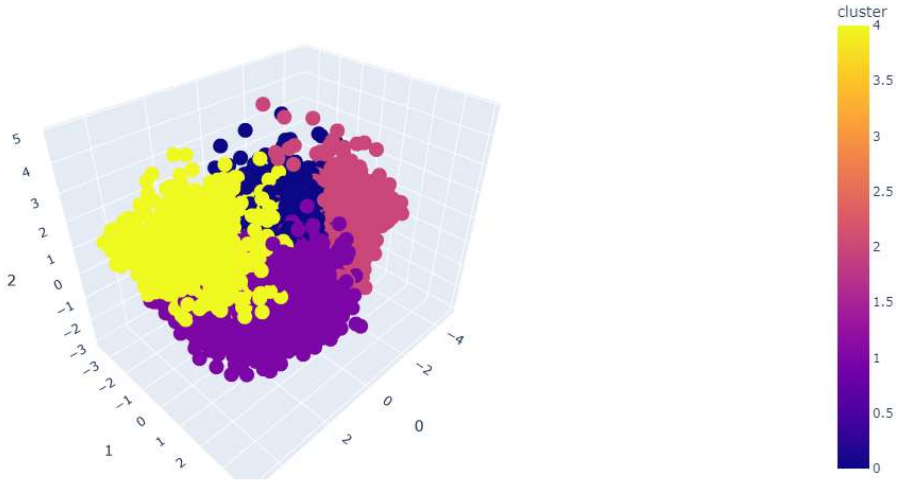


Observations: If we look at dendrogram for ward linkage, 4 appears like a reasonable cluster number estimate. However since we have taken a very small sample (100 points), we will continue with k=5.

Visulize PCA and TSNE

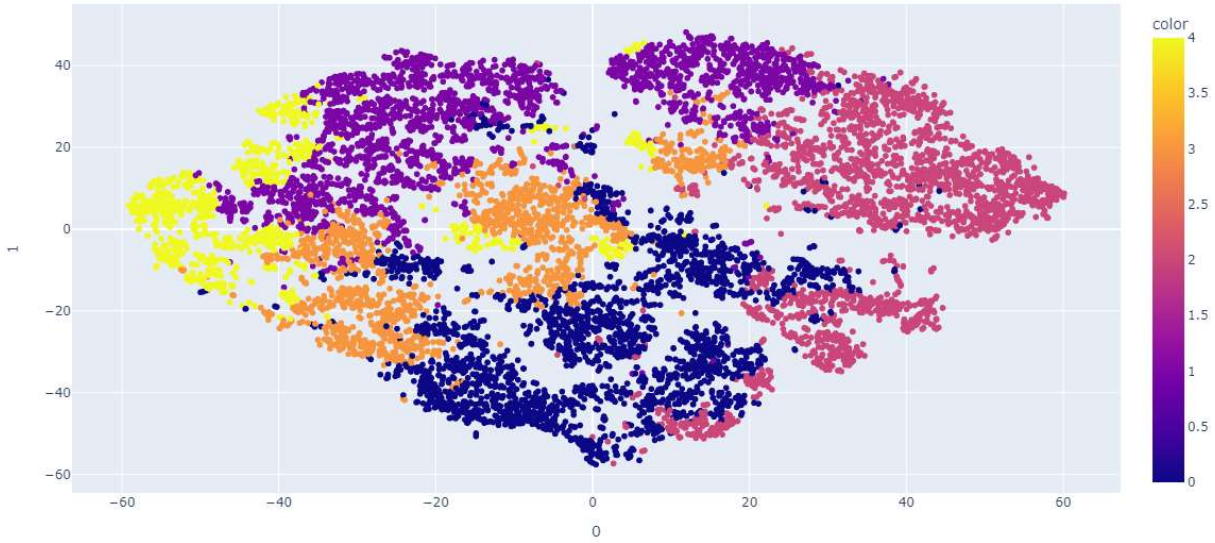
In [436…

```
visualize_pca(df_sample, hc.labels_)
```



In [437…

```
visualize_tsne(df_sample, y=hc.labels_, perplexity=100, method='barnes_hut')
```



Silhouette score

In [405…

```
#calculate silhouette_score for df_sample clustering
hc_sh_sample_score = cal_silhouette_score(df_sample, hc.labels_)
```

In [406…

```
print(f'Agglomerative clustering with sample size {df_sample.shape[0]} => silhoutte sco
```

```
Agglomerative clustering with sample size 10000 => silhoutte score: 0.1395721201261587
```

Cluster interpretation

In this section, we will attempt to understand characteristic of various clusters discovered by KMeans algorithm using various plots and summary stats.

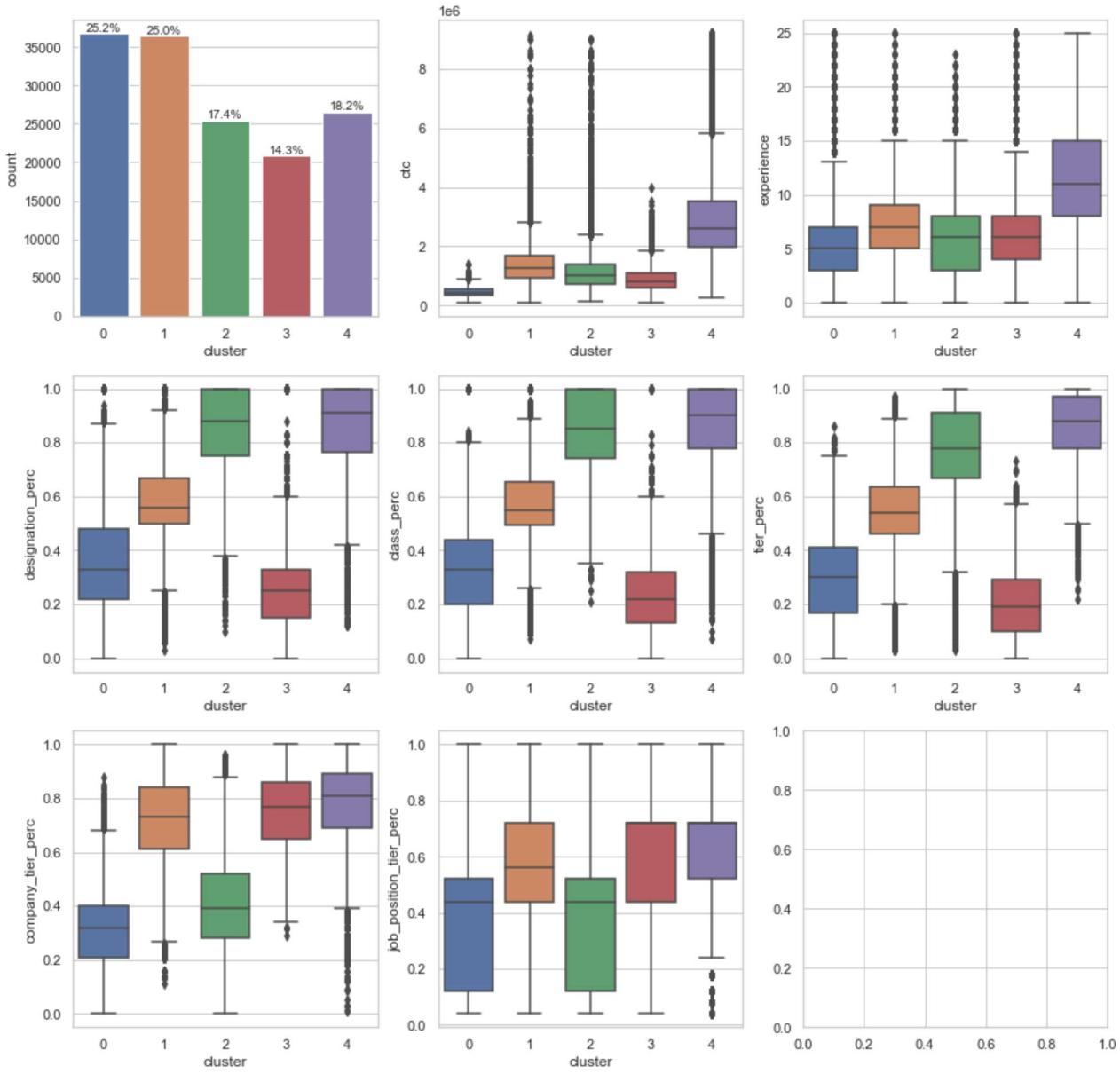
In [407…

```
def plot_cluster_graphs(df, labels):

    df2 = df.copy()
    df2['cluster'] = labels

    for col in ['cluster']:
        df2[col] = df2[col].astype('category')

    fig, ax = plt.subplots(3,3,figsize=(15,15))

    sns.countplot(x='cluster', data=df2, ax=ax[0][0])
    showpercent(ax[0][0], total=df2.shape[0])

    sns.boxplot(y='ctc', x='cluster', data=df2, ax=ax[0][1])
    sns.boxplot(y='experience', x='cluster', data=df2, ax=ax[0][2])

    sns.boxplot(y='designation_perc', x='cluster', data=df2, ax=ax[1][0])
    sns.boxplot(y='class_perc', x='cluster', data=df2, ax=ax[1][1])
    sns.boxplot(y='tier_perc', x='cluster', data=df2, ax=ax[1][2])
```

```
    sns.boxplot(y='company_tier_perc', x='cluster', data=df2, ax=ax[2][0])
    sns.boxplot(y='job_position_tier_perc', x='cluster', data=df2, ax=ax[2][1])

    plt.show()

def show_cluster_stats(df2, labels):

    df2['cluster'] = labels

    cluster_summary = df2.groupby('cluster').agg({
        'ctc': ['mean', 'median'],
        'experience': ['mean'],
        'designation_perc': ['mean'],
        'class_perc': ['mean'],
        'tier_perc': ['mean'],
        'company_tier_perc': ['mean'],
        'job_position_tier_perc': ['mean']
    })

    cluster_summary['ctc'] = cluster_summary['ctc']/100000
    cluster_summary = cluster_summary.rename(columns={'ctc': 'ctc(lacs)'})

    return np.round(cluster_summary.T, 2)
```

In [408…

```
plot_cluster_graphs(df_final, km_5.labels_)
show_cluster_stats(df_final, km_5.labels_)
```



Out[408...

	cluster	0	1	2	3	4
ctc(lacs)	**mean**	4.51	14.18	11.71	8.68	29.43
	median	4.30	12.51	10.00	8.00	26.00
experience	**mean**	5.32	7.24	5.90	6.16	11.44
designation_perc	**mean**	0.34	0.58	0.85	0.26	0.86
class_perc	**mean**	0.32	0.56	0.84	0.23	0.87
tier_perc	**mean**	0.29	0.54	0.77	0.20	0.86
company_tier_perc	**mean**	0.31	0.71	0.41	0.75	0.77
job_position_tier_perc	**mean**	0.37	0.56	0.40	0.56	0.69

Insights:

On close examination, we can note the following characteristics for each of the five clusters identified.

Cluster	cluster name	Characteristics
cluster-0	Underpaid Low performers (lower tier companies/job_positions)	Low performing (lower designation, class), under paid (mean CTC 4.5 lacs, tier percentile 0.29), Junior professionals (average experience 5.3 years). Majority work in lower tier company and less lucrative job_positions
cluster-1	Average paid average performers (high tier companies/job_positions)	Above average performing (above average designation, class percentile ranks), Mid level professionals (average experience 7.24 years), moderately paid (average ctc 14.18 lacs, tier percentile 0.54). Majority work in tier 1 companies and in lucrative job_positions.
cluster-2	High potential learners (lower tier companies/job_positions)	Competitive (high designation, class percentile ranks), well paid (average CTC 11.7 lacs, tier percentile rank 0.77). Majority work in lower/mid tier companies and less lucrative job_positions.
cluster-3	under paid Low performers (in high tier companies/job_positions)	Low performing (lower designation, class), under paid (mean CTC 8.68 lacs, tier percentile rank 0.20), Junior/Mid level professionals (average experience 6.16 years). Majority work in tier 1 companies and/or in lucrative job positions.
cluster-4	well paid High performers (high tier companies/job_positions)	High performing (high designation, class), highest paid (mean CTC 29.43 lacs, tier percentile rank 0.86), Mid/Senior professionals (average experience 11.44 years). Majority work in tier 1 companies and in lucrative job_positions.

Recommendations

1. Underpaid low performers (lower tier companies): These learners are underpaid low performers in their current job. They are usually employed at lower tier companies, often in less lucrative roles. With appropriate up-skilling, they may potentially get a huge CTC increase either by better performance, by switching to better roles in the same company, or by switching to tier 1 companies. Scaler should pursue these learners primarily focusing on the potential for landing higher paying jobs.

2. Underpaid low performers (high tier companies): These learners are underpaid low performers working in tier 1 companies. After appropriate upskilling, they have potential to fetch higher paying jobs in their companies/other top tier companies. These group of learners are also likely to land higher paying jobs after upskilling, and hence Scaler should pursue them focusing on potential for landing higher ctc jobs.

3. High potential performers (lower tier companies) These group represents high potential learners working in low/medium tier companies. With appropriate up-skilling, they have potential to crack high paying lucrative jobs in tier 1 companies. Scaler should pursue them and focus on potential to land higher paying as well as challenging jobs in top tier companies.

4. Average paid average performers (high tier companies): These learners are average paid average performers working in high tier companies and lucrative job_positions. With up-skilling, they can potentially fetch higher paying jobs, better roles. These set of learners are mostly mid-tier professionals, who may also be interested in career transition.

5. Highest paid high performers (high tier companies): These learners are high performing and highly paid mid/senior level employees who mostly work in top tier companies and high paying

job_positions. For these group, the primary motivation could be gaining knowledge and transitioning into different roles.