Scaler - Clustering

About Scaler

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.

Problem Statement

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**

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import plotly.express as px
import plotly.graph_objects as go


import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
df=pd.read_csv("scaler_clustering.csv")
df.head()
```

Out[2]:

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

In [3]:
```
df.shape
```

Out[3]:
```
(205843, 7)
```

The dataset has 205843 rows and 7 columns.

In [4]:
```
# check for the data types of values provided and see if any columns has the missing values.
df.info()
```

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

- There is missing values in some features.

In [5]:
```
df = df.drop('Unnamed: 0',axis=1)
```

In [6]:
```
df.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
orgyear	205757.0	2.014883e+03	6.357112e+01	0.0	2013.0	2016.0	2018.0	2.016500e+04
ctc	205843.0	2.271685e+06	1.180091e+07	2.0	530000.0	950000.0	1700000.0	1.000150e+09
ctc_updated_year	205843.0	2.019628e+03	1.325104e+00	2015.0	2019.0	2020.0	2021.0	2.021000e+03

In [7]:
```
df.describe(include ='object').T
```

Out[7]:

	count	unique	top	freq
company_hash	205799	37299	nvnv wgzohrnvzwj otqcxwto	8337
email_hash	205843	153443	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	10
job_position	153281	1017	Backend Engineer	43554

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

Out[8]:
```
Index(['company_hash', 'email_hash', 'orgyear', 'ctc', 'job_position',
       'ctc_updated_year'],
      dtype='object')
```

In [9]:
```
for x in df.columns:
    print("\033[1m" + "Feature : " + x  + "\033[0m")
    print('-'*30)
    print(' > Count of unique values :',df[x].nunique())
    print(' > Percentage of missing values :',np.around(df[x].isna().sum()*100/len(df[x]),4))
    print("__"*40)
```

```
Feature : company_hash
------------------------------
 > Count of unique values : 37299
 > Percentage of missing values : 0.0214
________________________________________________________________________________
Feature : email_hash
------------------------------
 > Count of unique values : 153443
 > Percentage of missing values : 0.0
________________________________________________________________________________
Feature : orgyear
------------------------------
 > Count of unique values : 77
 > Percentage of missing values : 0.0418
________________________________________________________________________________
Feature : ctc
------------------------------
 > Count of unique values : 3360
 > Percentage of missing values : 0.0
________________________________________________________________________________
Feature : job_position
------------------------------
 > Count of unique values : 1017
 > Percentage of missing values : 25.535
________________________________________________________________________________
Feature : ctc_updated_year
------------------------------
 > Count of unique values : 7
 > Percentage of missing values : 0.0
________________________________________________________________________________
```

In [10]:
```
for x in df.columns:
    print("\033[1m" + "Feature : " + x  + "\033[0m")
    print(df[x].value_counts())
    print(100 * "_")
```

Feature : company_hash

```
nvnv wgzohrnvzwj otqcxwto         8337
xzegojo                           5381
vbvkgz                            3481
zgn vuurxwvmrt vwwghzn            3411
wgszxkvzn                         3240
                                  ...
onvqmhwpo                            1
bvsxw ogenfvqt uqxcvnt rxbxnta       1
agsbv ojontbo                        1
vnnhzt xzegwgb                       1
bvptbjnqxu td vbvkgz                 1
Name: company_hash, Length: 37299, dtype: int64
```

Feature : email_hash

```
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee     9
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
b4d5afa09bec8689017d8b29701b80d664ca37b83cb883376b2e95191320da66     8
                                                                    ..
bb2fe5e655ada7f7b7ac4a614db0b9c560e796bdfcaa4e5367e69eedfea93876     1
d6cdef97e759dbf1b7522babccbbbd5f164a75d1b4139e02c945958720f1ed79     1
700d1190c17aaa3f2dd9070e47a4c042ecd9205333545dbfaee0f85644d00306     1
c2a1c9e4b9f4e1ed7d889ee4560102c1e2235b2c1a0e59cea95a6fe55c658407     1
0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f7e738a6a87d3712c31     1
Name: email_hash, Length: 153443, dtype: int64
```

Feature : orgyear

```
2018.0     25256
2019.0     23427
2017.0     23239
2016.0     23043
2015.0     20610
2014.0     16696
2020.0     13431
2013.0     12351
2012.0     10493
2011.0      7970
2010.0      5751
2009.0      3777
2021.0      3670
2008.0      2728
2007.0      2257
2006.0      2075
2005.0      1873
2004.0      1455
2003.0      1018
2022.0       911
2001.0       713
2002.0       685
2000.0       495
1999.0       340
1998.0       279
2023.0       252
1997.0       234
1996.0       134
1995.0        94
1991.0        79
1993.0        74
1994.0        65
1992.0        47
2024.0        43
1990.0        38
1989.0        22
0.0           17
2025.0        13
1988.0        10
2026.0         9
1986.0         8
1987.0         6
3.0            6
1985.0         5
2031.0         5
2029.0         5
1982.0         4
2028.0         4
1984.0         3
91.0           3
2.0            3
20165.0        2
1970.0         2
6.0            2
5.0            2
1.0            2
1979.0         1
83.0           1
209.0          1
2204.0         1
1977.0         1
1900.0         1
201.0          1
38.0           1
4.0            1
1971.0         1
206.0          1
1976.0         1
2027.0         1
1981.0         1
1973.0         1
```

```
2106.0         1
2107.0         1
1972.0         1
2101.0         1
208.0          1
200.0          1
Name: orgyear, dtype: int64
```

```
Feature : ctc
600000     7832
400000     7598
1000000    7581
500000     7242
800000     6752
           ...
1916000       1
5340000       1
2305000       1
4225000       1
3327000       1
Name: ctc, Length: 3360, dtype: int64
```

```
Feature : job_position
Backend Engineer                  43554
FullStack Engineer                24717
Other                             18071
Frontend Engineer                 10417
Engineering Leadership             6870
                                  ...
ayS                                   1
Principal Product Engineer            1
Senior Director of Engineering        1
Seller Support Associate              1
Android Application developer         1
Name: job_position, Length: 1017, dtype: int64
```

```
Feature : ctc_updated_year
2019.0    68688
2021.0    64976
2020.0    49444
2017.0     7561
2018.0     6746
2016.0     5501
2015.0     2927
Name: ctc_updated_year, dtype: int64
```

Checking missing values

In [11]:
```
# Check for the missing values in dataset
null_total = df.isna().sum().sort_values(ascending = False)
null_percent =((df.isnull().sum()/df.isnull().count())*100).sort_values(ascending = False)
data_type =df.dtypes
null = pd.concat([null_total,null_percent.round(3),df.dtypes],axis=1,keys=['Missing in Total','Missing in Percent','Data type'])
null
```

Out[11]:

	Missing in Total	Missing in Percent	Data type
job_position	52562	25.535	object
orgyear	86	0.042	float64
company_hash	44	0.021	object
email_hash	0	0.000	object
ctc	0	0.000	int64
ctc_updated_year	0	0.000	float64

- job_position 25% data is missing which can be replaced with "Others".
- company_hash is a categorical column so let's fill it with string "others"
- The missing values in orgyear can be imputed by mean/knn imputation.

Checking duplicate values

In [12]:
```
# Check for the duplicate values in dataset
df.duplicated().sum()
```

Out[12]: 33

In [13]:
```
# let's drop duplicate values from dataframe
df = df.drop_duplicates(keep='last')
```

Checking unique emails and frequency of occurrence of the same email hash in the data

In [14]:
```
# Finding count of emails in dataset.
print('Count of total emails :',len(df["email_hash"]))
print('Count of unique emails :',df["email_hash"].nunique())
print('Count of duplicate emails :',df["email_hash"].duplicated().sum())
```

```
Count of total emails : 205810
Count of unique emails : 153443
Count of duplicate emails : 52367
```

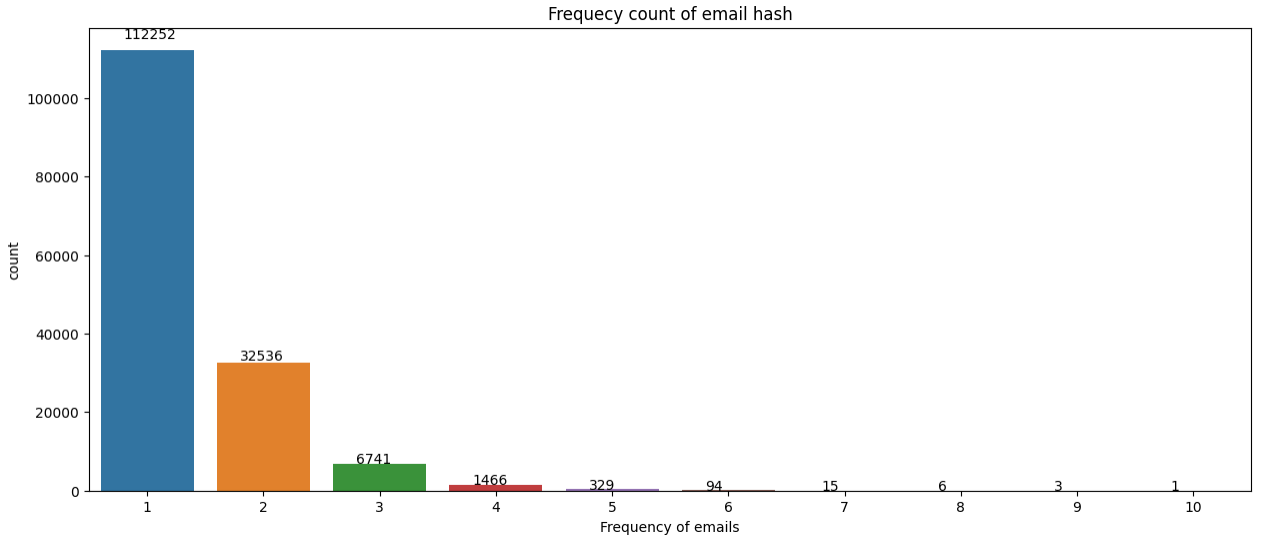
- There are 1,53,443 unique email ids present in the data and remaining are duplicates.

In [15]:
```
# Finding frequency of occurrence of the same email hash in the data.
df.email_hash.value_counts()
```

Out[15]:
```
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee     9
c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7e8cc6a20b0d1938183     8
                                                                    ..
ff9cc70bdbe86b3929ed54254034eb4508a62745d106664b5356e2aea4e73a8b     1
9148cf557e1df4894984d8c8e5384b16d9d023959c0d58ec677612700d0ae068     1
c1fde9d634fbcab672b6c2c88fa1da49d1149242eed895d88c8aee0d597dfe08     1
049d9f3080422ff65b4259f39078dab4feb89c8325441ab26a5bb27a97310efd     1
0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f7e738a6a87d3712c31     1
Name: email_hash, Length: 153443, dtype: int64
```

The maximum repeatation of the email count is 10.

In [16]:
```
plt.figure(figsize=(15,6))
ax = sns.countplot(x=df['email_hash'].value_counts())
for p in ax.patches:
    ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.2, p.get_height()*1.025))
ax.set_title("Frequecy count of email hash")
ax.set_xlabel("Frequency of emails")
plt.show()
```



In [17]:
```
df = df.drop('email_hash',axis=1)
```

Analysis of Continuous Features - ctc, ctc_updated_year, orgyear

In [18]:
```
df = df[df['orgyear'] <= 2023]
```

In [19]:
```
df.describe().T
```

Out[19]:

	count	mean	std	min	25%	50%	75%	max
orgyear	205638.0	2.014699e+03	2.897050e+01	0.0	2013.0	2016.0	2018.0	2.023000e+03
ctc	205638.0	2.267567e+06	1.178232e+07	2.0	530000.0	950000.0	1700000.0	1.000150e+09
ctc_updated_year	205638.0	2.019628e+03	1.325287e+00	2015.0	2019.0	2020.0	2021.0	2.021000e+03

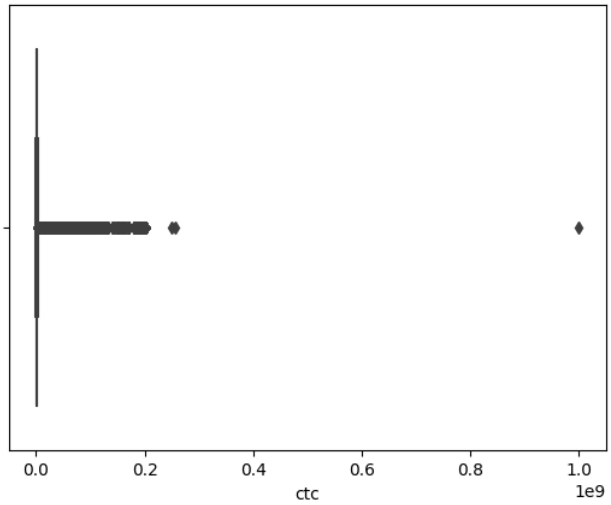
1) ctc

In [20]:
```
df.ctc.describe()
```

Out[20]:
```
count    2.056380e+05
mean     2.267567e+06
std      1.178232e+07
min      2.000000e+00
25%      5.300000e+05
50%      9.500000e+05
75%      1.700000e+06
max      1.000150e+09
Name: ctc, dtype: float64
```

In [21]:
```
sns.boxplot(x='ctc',data = df)
```

Out[21]:
```
<Axes: xlabel='ctc'>
```

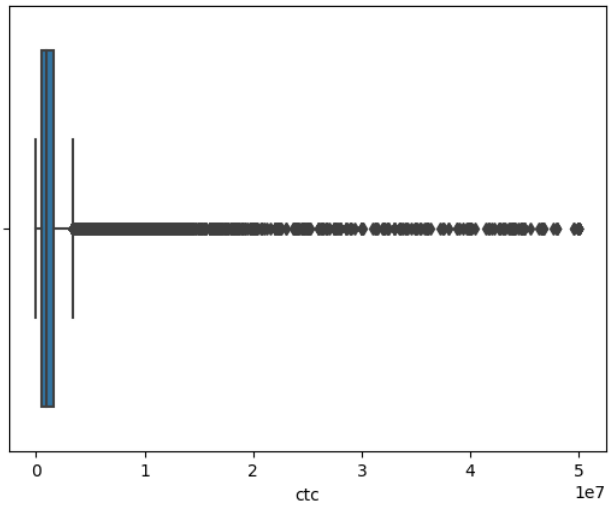


- The range of CTC is too large. There are outliers present.

In [22]:
```
df = df.drop(df[df['ctc']>50000000].index)
```

In [23]:
```
sns.boxplot(x='ctc',data = df)
```

Out[23]:
```
<Axes: xlabel='ctc'>
```



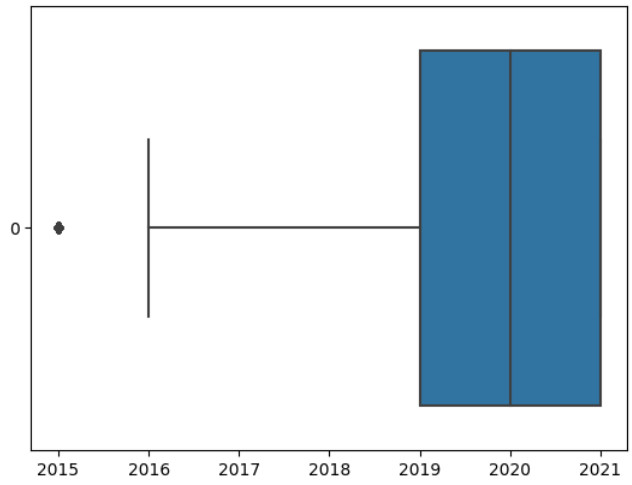
2) ctc_updated_year

In [24]:
```
df.ctc_updated_year.describe().round(2)
```

Out[24]:
```
count    204193.00
mean       2019.63
std           1.33
min        2015.00
25%        2019.00
50%        2020.00
75%        2021.00
max        2021.00
Name: ctc_updated_year, dtype: float64
```

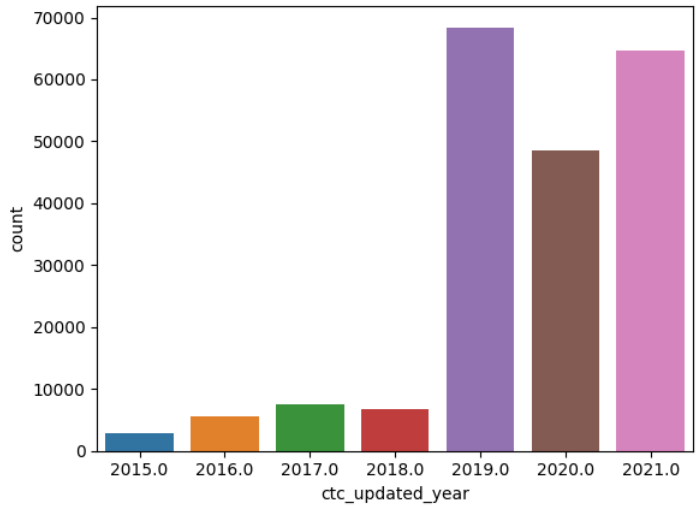
In [25]:
```
sns.boxplot(df['ctc_updated_year'],orient = 'h')
```

Out[25]:
```
<Axes: >
```



In [26]:
```
sns.countplot(x=df['ctc_updated_year'])
```

Out[26]:
```
<Axes: xlabel='ctc_updated_year', ylabel='count'>
```



Most of the CTC revisions are done in years 2019 and 2021.

3) org_year

In [27]:
```
df.orgyear = np.where(df.orgyear.isnull(), df.ctc_updated_year, df.orgyear)
```

In [28]:
```
# In some cases orgyear is more than ctc_updated_year.Lets consider ctc_updated_year is latest and hence we need to replace those orgyear values wit
df.loc[(df['orgyear'] > df['ctc_updated_year'])]
```

Out[28]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year
8	utqoxontzn ojontbo	2020.0	450000	NaN	2019.0
17	puxn	2020.0	1400000	NaN	2019.0
40	rvqotz nghmqg	2021.0	2500000	Other	2020.0
48	rgfto wgbuvzxto xzw	2020.0	3010000	NaN	2019.0
56	axztqg xzzgcvnxgz ucn rna	2020.0	700000	Frontend Engineer	2019.0
...	...	...	...	...	...
205796	zgn vuurxwvmrt	2021.0	19800000	NaN	2019.0
205798	ztw wgqugqvnxgz	2020.0	800000	NaN	2019.0
205801	zgn vuurxwvmrt	2019.0	1800000	NaN	2016.0
205804	myvqvn trtwnqgzxwo rxbxnta	2022.0	1000000	NaN	2020.0
205834	wyvqntq wgbbhzxwvnxgzo	2020.0	100000	NaN	2019.0

8619 rows × 5 columns

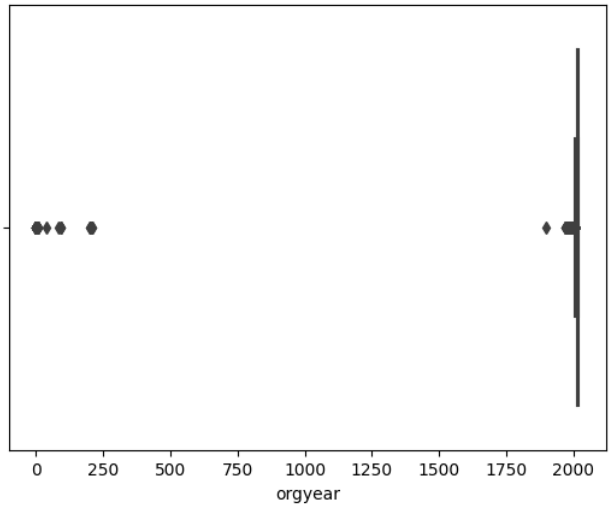
In [29]:
```
df.orgyear = np.where(df['orgyear']>df['ctc_updated_year'], df.ctc_updated_year, df.orgyear)
```

In [30]:
```
df.orgyear.isnull().sum()
```

Out[30]:
```
0
```

In [31]:
```
sns.boxplot(x='orgyear',data=df)
```

Out[31]:
```
<Axes: xlabel='orgyear'>
```

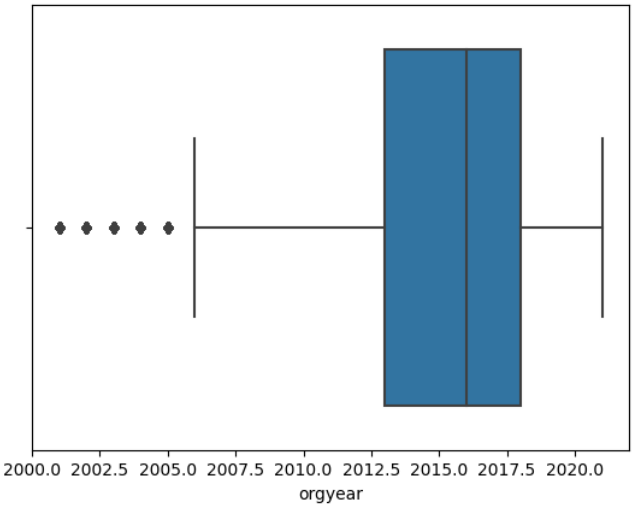


orgyear has many outliers.We limit the range of employee joinied year from 2000 to 2023 only.

In [32]:
```
# df.orgyear = np.where(df['orgyear']<1970, df.ctc_updated_year, df.orgyear) # considering 50 years of age
df['orgyear'] = df['orgyear'].clip(lower=df.orgyear.quantile(0.01), upper=df.orgyear.quantile(1.00))
```

In [33]:
```
sns.boxplot(x='orgyear',data=df)
```

Out[33]:
```
<Axes: xlabel='orgyear'>
```

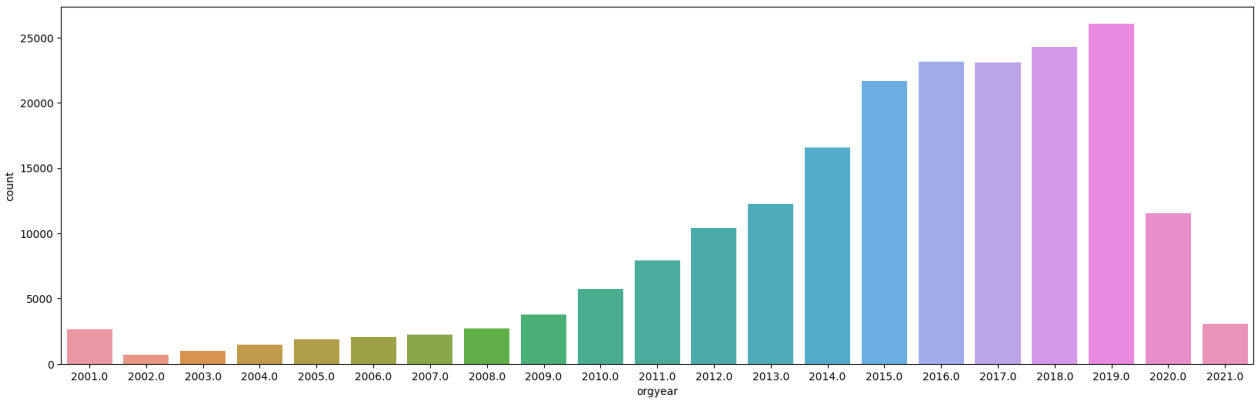


In [34]:
```
df.orgyear.describe().T
```

Out[34]:
```
count    204193.000000
mean       2015.092045
std           4.019617
min        2001.000000
25%        2013.000000
50%        2016.000000
75%        2018.000000
max        2021.000000
Name: orgyear, dtype: float64
```

In [35]:
```
plt.figure(figsize=(20,6))
sns.countplot(x=df['orgyear'])
```

Out[35]:
```
<Axes: xlabel='orgyear', ylabel='count'>
```



Most of the learners have joined their respective companies from 2015 onwards.

In [36]:
```
df['orgyear'] =  df['orgyear'].astype("int")
df['ctc_updated_year'] =  df['ctc_updated_year'].astype("int")
```

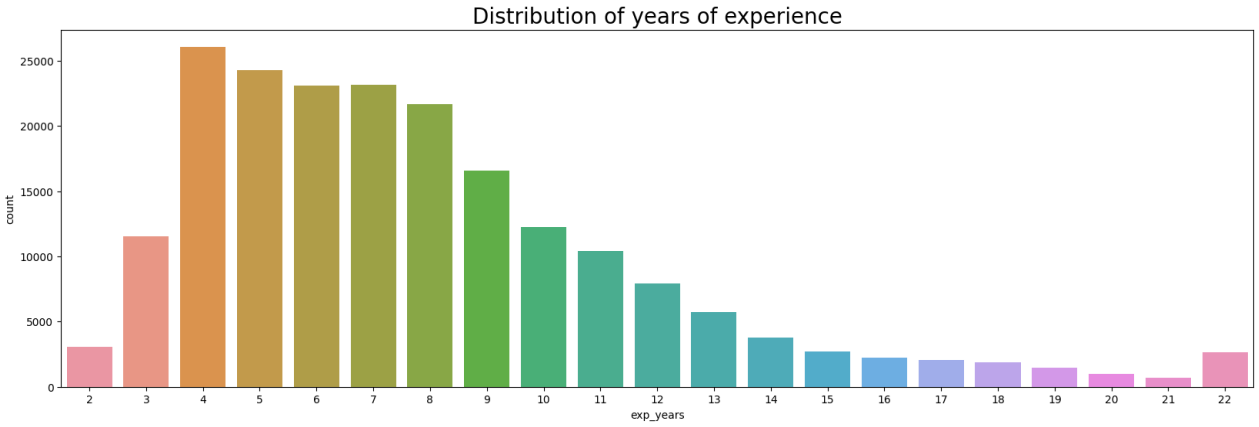
Creating new feature column 'exp_years' from orgyear

In [37]:
```
from datetime import datetime
df['exp_years']=datetime.now().year - df['orgyear']
```

In [38]:
```
df['exp_years'].describe().round(2).T
```

Out[38]:
```
count    204193.00
mean          7.91
std           4.02
min           2.00
25%           5.00
50%           7.00
75%          10.00
max          22.00
Name: exp_years, dtype: float64
```

In [39]:
```
plt.figure(figsize=(20,6))
df['exp_years'].value_counts().sort_values(ascending = False)[:15]
sns.countplot(x='exp_years',data=df)
plt.title('Distribution of years of experience',fontsize = 20)
plt.xticks(rotation=0)
plt.show()
```

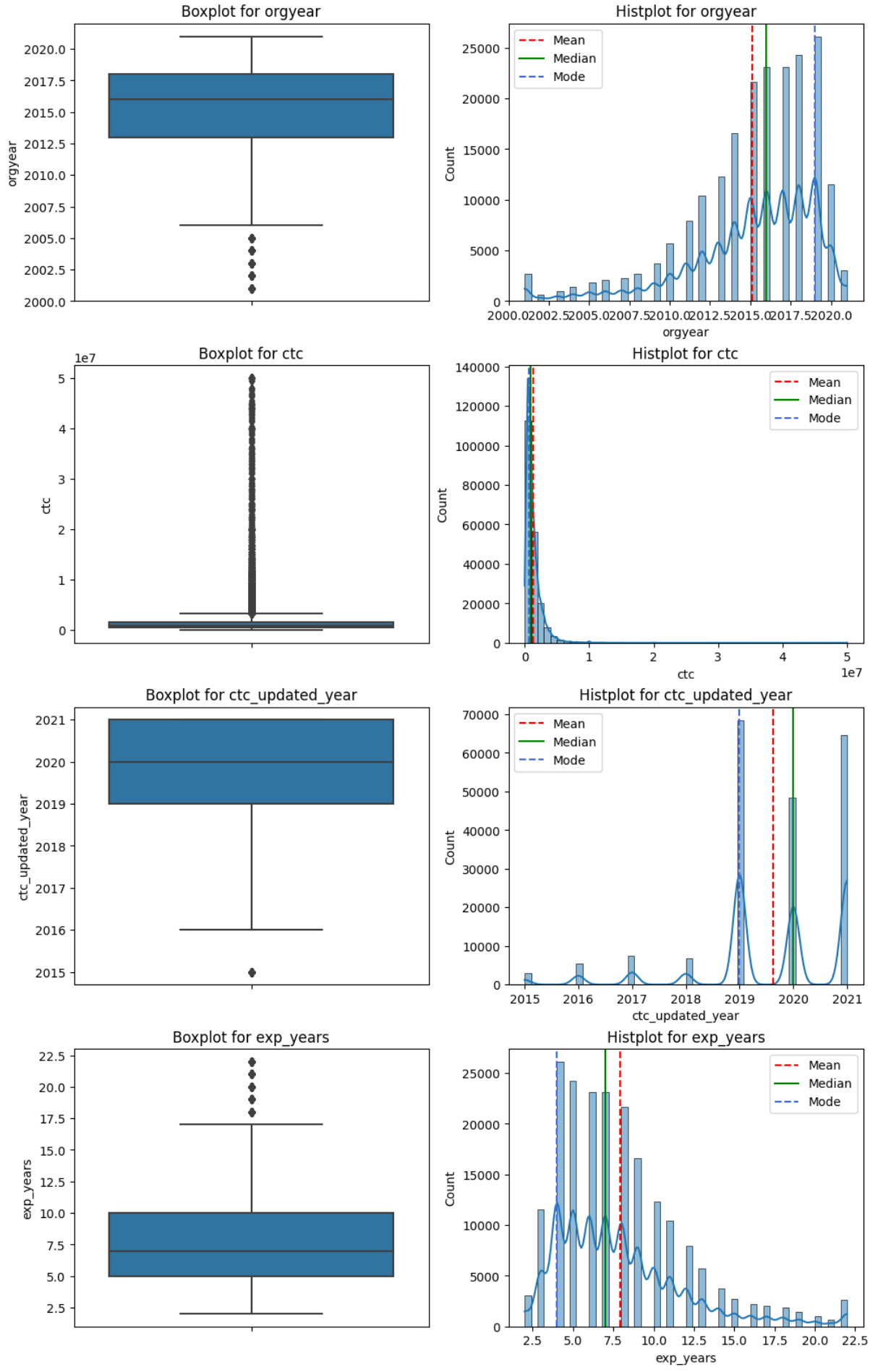


A large group of learners have experience between 4 years and 9 years.

In [40]:
```
def univariate_analysis(data, features, type_of_feature, width = 10, height = 4):
 '''
 This function is to perform univariate analysis
 For Categorical features: Count Plot and Pie Chart
 For Continuous : Boxplot and Histogram
 Parameters:
 data (pd.DataFrame): Data under analysis
 features (list): List of continuous/categorical features
 type_of_feature (string): continuous or categorical
 width (int): width of 1 axes
 height (int): height of 1 axes

 '''
 n_features = len(features)
 figsize = (width, height*n_features)
 fig, ax = plt.subplots(nrows=n_features, ncols=2, figsize=figsize, constrained_layout=True)
 if n_features == 1:
    ax = [ax]
 if type_of_feature == 'continuous':
    for i in range(n_features):
        f = features[i]
        ax[i][0].set_title(f"Boxplot for {f}")
        sns.boxplot(y = data[f], ax=ax[i][0],orient = 'h')
        ax[i][0].set_ylabel(f)
        sns.histplot(data[f], ax=ax[i][1], kde=True, bins=50)
        ax[i][1].axvline(data[f].mean(), color='r', linestyle='--', label="Mean")
        ax[i][1].axvline(data[f].median(), color='g', linestyle='-', label="Median")
        ax[i][1].axvline(data[f].mode()[0], color='royalblue', linestyle='--', label="Mode")
        ax[i][1].set_title(f"Histplot for {f}")
        ax[i][1].legend()
 elif type_of_feature == 'categorical':
    for i in range(n_features):
        f = features[i]
        counts = df[f].value_counts()
        ax[i][0].set_title(f"Count Plot for {f}")
        counts.plot.bar(ax=ax[i][0])
        ax[i][0].set_ylabel("Counts")
        ax[i][0].set_xlabel(f)
        ax[i][1].set_title(f"Pie chart for {f}")
        counts.plot.pie(autopct='%0.0f%%', ax=ax[i][1])
 fig.suptitle(f"Univariate analysis of {type_of_feature} features", fontweight="bold")
 plt.show()
```

In [41]:
```
univariate_analysis(df, features=['orgyear', 'ctc', 'ctc_updated_year', 'exp_years'], type_of_feature = 'continuous')
```

Univariate analysis of continuous features



Analysis of Categorical features - job_position , company_hash

Using Regex to remove special characters from the dataset

In [42]:
```
def regex_string(string):
    newstring= re.sub('[^A-Za-z ]+', '', string).lower().strip()
    return newstring
```

1) job_position

In [43]:
```
df.job_position=df.job_position.apply(lambda x: regex_string(str(x)))
```

In [44]:
```
avg_sal = df[["job_position","ctc"]].groupby("job_position").mean().reset_index().sort_values("ctc",ascending= False)
avg_sal["ctc"] = round(avg_sal.ctc,2)
avg_sal.head()
```

Out[44]:

	job_position	ctc
152	computer faculty	24000000.0
527	security intern	11000000.0
817	toyota	10000000.0
204	electric power supply	10000000.0
420	phd student	8400000.0

In [45]:
```
# Encoding the job_position using the average salary of each position
avg_sal.loc[avg_sal["job_position"]=="backend engineer"]
```

Out[45]:

	job_position	ctc
112	backend engineer	1556636.72

In [46]:
```
x = avg_sal.job_position.to_list()
y = avg_sal.ctc.to_list()
job_sal_dict = dict(zip(x,y))
joblist = df.job_position.to_list()
jobsal = []
for jobs in joblist:
 if pd.isnull(jobs):
    jobsal.append(None) # Replace missing values by None
 else :
    jobsal.append(job_sal_dict[jobs])

df["job_position_encoded"] = jobsal
```

In [47]:
```
# Imputing the missing data in job_position_encoded feature
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors = 25)
job_pos_impute = imputer.fit_transform(np.array(df.job_position_encoded).reshape(-1,1))
df["job_position_encoded"] = np.round(job_pos_impute,2)
# Filling the missing values in job_position by 'Other'

df.loc[df['job_position']=='nan', 'job_position']=np.nan
df["job_position"]=df["job_position"].fillna("other")
```

In [48]:
```
df["job_position"].value_counts()[:20]
```

Out[48]:
```
other                     69877
backend engineer          43358
fullstack engineer        25833
frontend engineer         10353
engineering leadership     6805
qa engineer                6538
data scientist             5345
android engineer           5325
sdet                       4963
devops engineer            4596
support engineer           3538
data analyst               2840
ios engineer               2732
engineering intern         2672
product designer           1305
backend architect          1282
research engineers         1221
product manager            1136
program manager             803
non coder                   586
Name: job_position, dtype: int64
```

2) company_hash

In [49]:
```
df.company_hash=df.company_hash.apply(lambda x: regex_string(str(x)))
```

In [50]:
```
df.company_hash.fillna('others',inplace=True)
df.loc[df.groupby('company_hash')['ctc'].transform('count') <= 5, 'company_hash'] = 'other'
```

In [51]:
```
# Removing rows where company or job_position is not available
df=df[ ~((df['company_hash']=='') | (df['job_position']==''))]
df.drop(columns=['job_position_encoded'],inplace = True)
```

Outlier Treatment

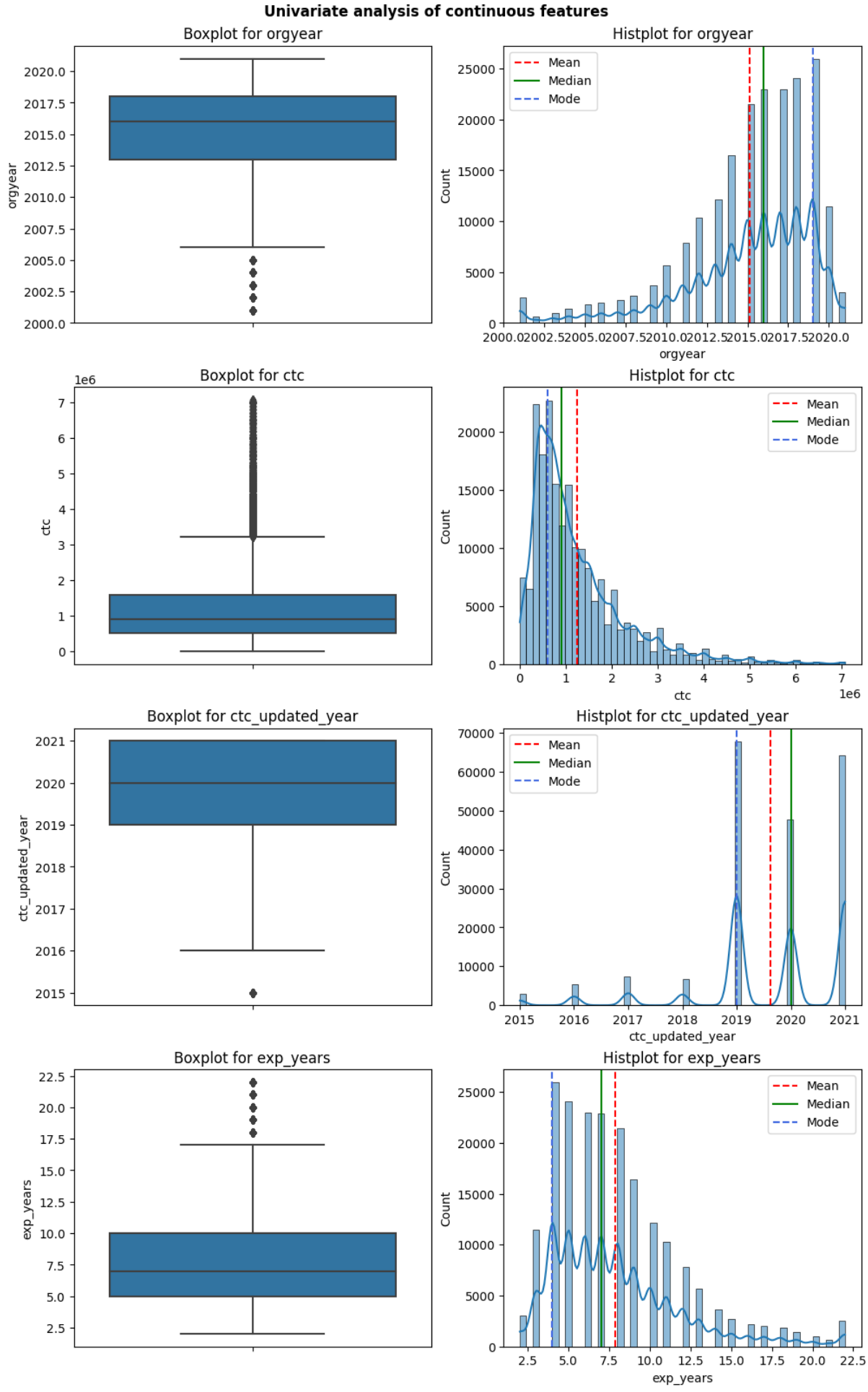
In [52]:
```
# function for outlier treatment
def remove_outliers(data, features, method='iqr'):
 '''
 Removes outliers based on requested method.

 Parameters:
 data (pandas.DataFrame): Data under analysis
 features (list): List of features for outlier removal
 method (string): Accepts two values "iqr", "z-score"
 Returns:
 data (pandas.Series): Cleaned Data
 outliers (pandas.Series):
 '''
 intial_shape = data.shape
 outliers = pd.DataFrame()
 if method == 'iqr':
    for f in features:
        q1 = data[f].quantile(0.25)
        q3 = data[f].quantile(0.75)
        iqr = q3-q1
        f_outliers = data.loc[(data[f]<q1-1.5*iqr) | (data[f]>q3+1.5*iqr)]
        outliers = outliers.append(f_outliers)
        data.drop(f_outliers.index, inplace=True)
 elif method == 'z-score':
    for f in features:
        mean = data[f].mean()
        std = data[f].std()
        f_outliers = data.loc[((data[f]-mean)/std<-3) | ((data[f]-mean)/std>3)]
        outliers = outliers.append(f_outliers)
        data.drop(f_outliers.index, inplace=True)
 print(f'{outliers.shape[0]*100/intial_shape[0]} % of data detected as outlier.')
 return data, outliers
```

In [53]:
```
#Let's remove these outliers
df_new, outliers = remove_outliers(data=df,features=['ctc'], method='z-score')
cols = df.columns
df[cols] = df_new[cols]
```

```
0.9676534281878668 % of data detected as outlier.
```

Univariate analysis after removing outliers

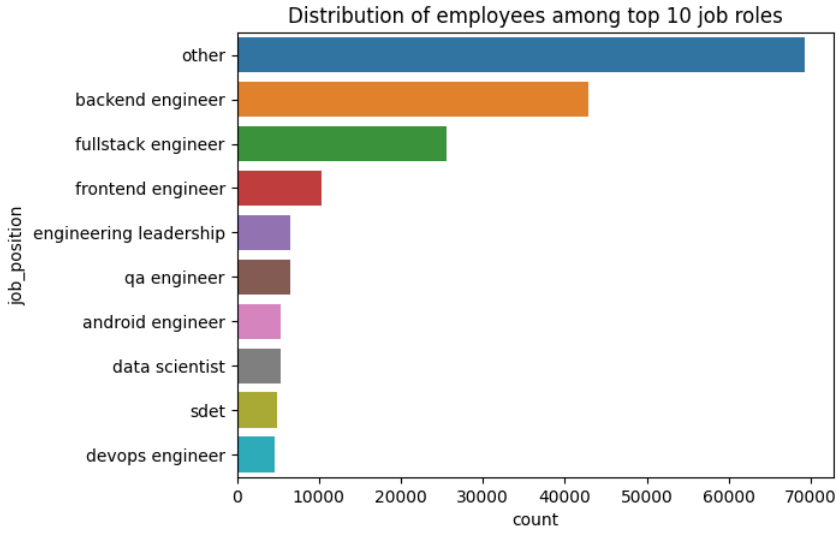
In [54]:
```
univariate_analysis(df_new, features=['orgyear', 'ctc', 'ctc_updated_year', 'exp_years'], type_of_feature = 'continuous')
```



In [55]:
```
Jobpsn = pd.DataFrame(df['job_position'].value_counts())
Jobpsn.reset_index(inplace= True)
Jobpsn.rename(columns = {'index':'job_position', 'job_position':'count'}, inplace = True)
Jobpsn = Jobpsn[:10]
```

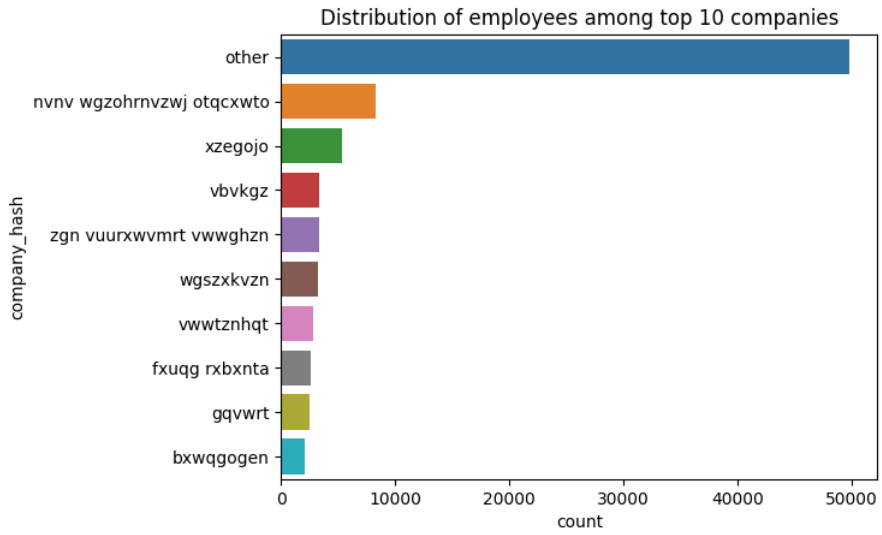
```
plt.title('Distribution of employees among top 10 job roles',fontsize = 12)
sns.barplot(y= Jobpsn['job_position'],x= Jobpsn['count'], orient='h')
```

Out[55]: <Axes: title={'center': 'Distribution of employees among top 10 job roles'}, xlabel='count', ylabel='job_position'>



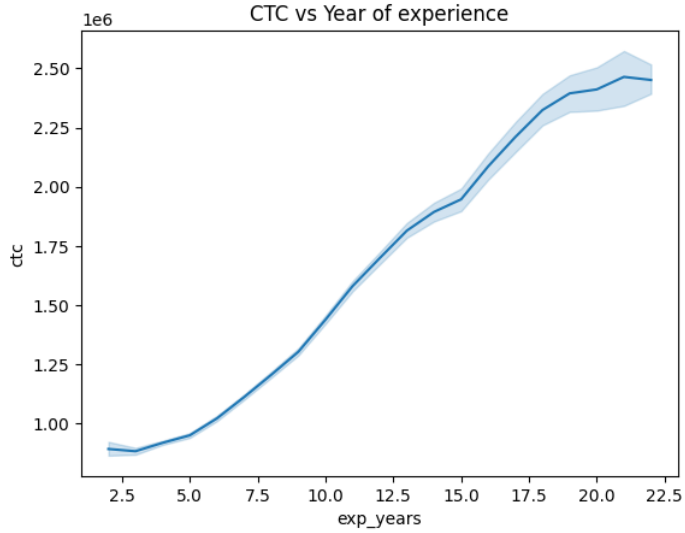
In [56]:
```
cmpy = pd.DataFrame(df['company_hash'].value_counts())
cmpy.reset_index(inplace= True)
cmpy.rename(columns = {'index':'company_hash', 'company_hash':'count'}, inplace = True)
cmpy = cmpy[:10]
plt.title('Distribution of employees among top 10 companies',fontsize=12)
sns.barplot(y= cmpy['company_hash'],x= cmpy['count'], orient='h')
```

Out[56]: <Axes: title={'center': 'Distribution of employees among top 10 companies'}, xlabel='count', ylabel='company_hash'>



In [57]:
```
sns.lineplot(x=df['exp_years'], y=df['ctc'])
plt.title('CTC vs Year of experience')
```

Out[57]: Text(0.5, 1.0, 'CTC vs Year of experience')



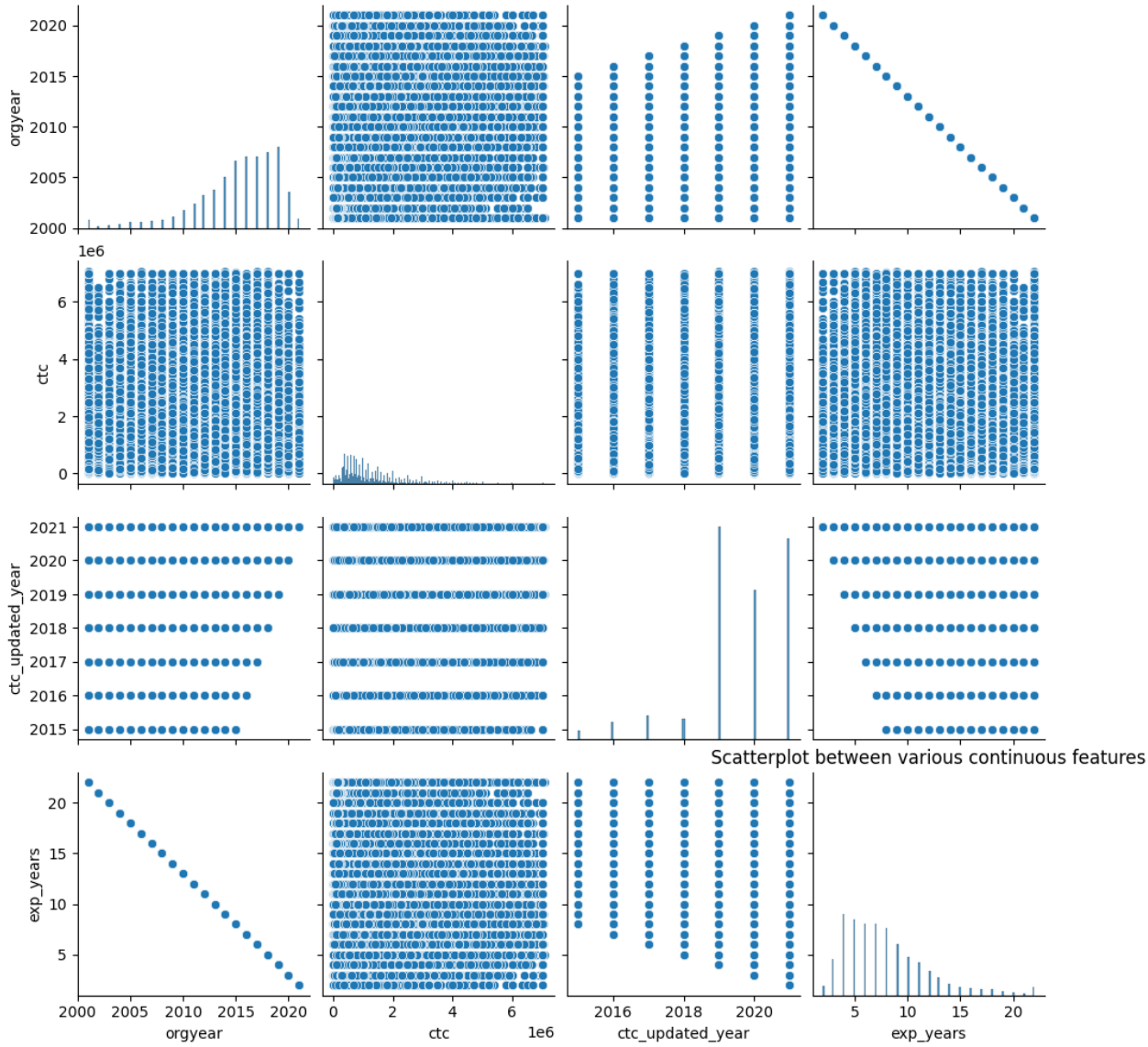
There is rise in CTC among employees with respect to their years of experience.

Bivariate Analysis

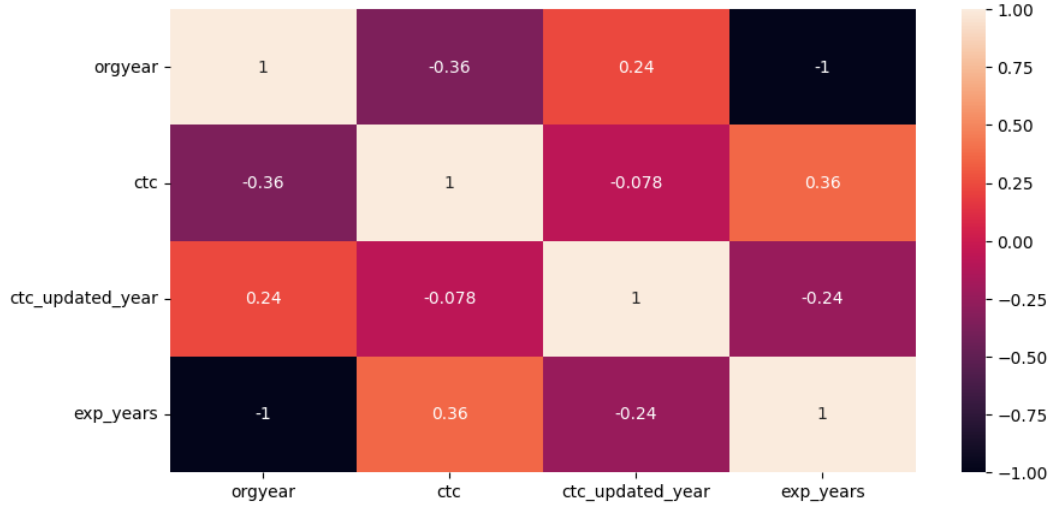
In [58]:
```
plt.figure(figsize=(12,10))
sns.pairplot(df)
plt.title('Scatterplot between various continuous features')
```

Out[58]: Text(0.5, 1.0, 'Scatterplot between various continuous features')

<Figure size 1200x1000 with 0 Axes>



In [59]:
```
plt.figure(figsize=(10,5))
ax = sns.heatmap(df.corr(), annot=True)
```



- Years of experience and orgyear are negatively correlated.
- CTC is positively correlated with years of experience.

Manual Clustering

-- Based on learner's company, job position and years of experience.

-- Creation of new features Tier , Class , Designation based on different levels of grouping wrt CTC

-- 3 clusters are created for each newly created features based on average CTC

Summary of CTC on the basis of Company, Job Position, Years of Experience

In [60]:
```
d1=df.groupby(['exp_years','job_position','company_hash'])['ctc'].describe().round(2)
d1.head(10)
```

Out[60]:

			count	mean	std	min	25%	50%	75%	max
exp_years	job_position	company_hash								
2	analyst programmer	other	1.0	360000.0	NaN	360000.0	360000.0	360000.0	360000.0	360000.0
	android engineer	bgztj cxtf	1.0	1900000.0	NaN	1900000.0	1900000.0	1900000.0	1900000.0	1900000.0
		bhnhvr bgmxrt	1.0	1000000.0	NaN	1000000.0	1000000.0	1000000.0	1000000.0	1000000.0
		egqa bgngq wgbuvzj	1.0	1100000.0	NaN	1100000.0	1100000.0	1100000.0	1100000.0	1100000.0
		evwtmggp	1.0	100000.0	NaN	100000.0	100000.0	100000.0	100000.0	100000.0
		grv vzx ntwyzgrgsxto ucn rna	1.0	400000.0	NaN	400000.0	400000.0	400000.0	400000.0	400000.0
		other	5.0	412000.0	313878.96	100000.0	180000.0	400000.0	480000.0	900000.0
		otre tburgjta	1.0	150000.0	NaN	150000.0	150000.0	150000.0	150000.0	150000.0
		ptrrngz ntwy ogrhnxgzo	1.0	300000.0	NaN	300000.0	300000.0	300000.0	300000.0	300000.0
		qvaxojo wgqugqvnxgz	1.0	1150000.0	NaN	1150000.0	1150000.0	1150000.0	1150000.0	1150000.0

Creating **Designation** Flag & Insights

- Analysis at Company, Job position and Years of experience level
- Creating Designation flag on the basis of salary they are getting in their respective companybased on years of experience.

In [61]:
```
df_merged1=df.merge(d1, on=['exp_years','job_position','company_hash'], how='left')
df_merged1.sort_values(['exp_years','job_position','company_hash']).head(10)
```

Out[61]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	count	mean	std	min	25%	50%	75%
33133	other	2021	360000	analyst programmer	2021	2	1.0	360000.0	NaN	360000.0	360000.0	360000.0	360000.0
184516	bgztj cxtf	2021	1900000	android engineer	2021	2	1.0	1900000.0	NaN	1900000.0	1900000.0	1900000.0	1900000.0
177964	bhnhvr bgmxrt	2021	1000000	android engineer	2021	2	1.0	1000000.0	NaN	1000000.0	1000000.0	1000000.0	1000000.0
158690	egqa bgngq wgbuvzj	2021	1100000	android engineer	2021	2	1.0	1100000.0	NaN	1100000.0	1100000.0	1100000.0	1100000.0
27082	evwtmggp	2021	100000	android engineer	2021	2	1.0	100000.0	NaN	100000.0	100000.0	100000.0	100000.0
72258	grv vzx ntwyzgrgsxto ucn rna	2021	400000	android engineer	2021	2	1.0	400000.0	NaN	400000.0	400000.0	400000.0	400000.0
4875	other	2021	480000	android engineer	2021	2	5.0	412000.0	313878.96	100000.0	180000.0	400000.0	480000.0
40314	other	2021	100000	android engineer	2021	2	5.0	412000.0	313878.96	100000.0	180000.0	400000.0	480000.0
43063	other	2021	400000	android engineer	2021	2	5.0	412000.0	313878.96	100000.0	180000.0	400000.0	480000.0
128324	other	2021	900000	android engineer	2021	2	5.0	412000.0	313878.96	100000.0	180000.0	400000.0	480000.0

In [62]:
```
def designation(x,x_50,x_75):
    if x >= x_75:
        return 1
    elif x >= x_50 and x <= x_75:
        return 2
    elif x < x_50:
        return 3
```

In [63]:
```
df_merged1['designation'] =df_merged1.apply(lambda x: designation(x['ctc'],x['50%'],x['75%']), axis=1)
df_merged1.head()
```

Out[63]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	count	mean	std	min	25%	50%	75%	
0	atrgxnnt xzaxv	2016	1100000	other	2020	7	1.0	1100000.00	NaN	1100000.0	1100000.0	1100000.0	1100000.0	1100
1	qtrxvzwt xzegwgbb rxbxnta	2018	449999	fullstack engineer	2019	5	7.0	774285.57	250922.32	449999.0	610000.0	750000.0	900000.0	1200
2	other	2015	2000000	backend engineer	2020	8	963.0	1097089.19	955809.23	1000.0	500000.0	900000.0	1430000.0	700(
3	ngpgutaxv	2017	700000	backend engineer	2019	6	6.0	1218333.33	408187.05	700000.0	937500.0	1205000.0	1502500.0	175(
4	qxen sqghu	2017	1400000	fullstack engineer	2019	6	1.0	1400000.00	NaN	1400000.0	1400000.0	1400000.0	1400000.0	140(

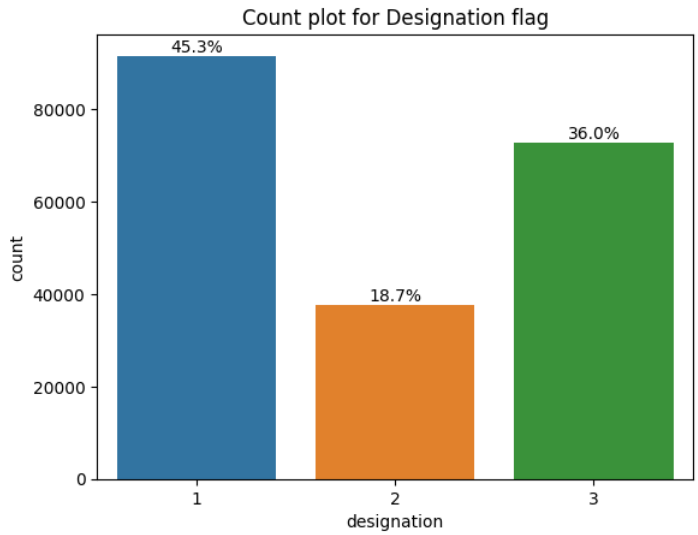
In [64]:
```
df_merged1['designation'].value_counts()
```

Out[64]:
```
1    91634
3    72750
2    37743
Name: designation, dtype: int64
```

In [65]:
```
df_merged1.designation.value_counts(normalize=True) * 100
```

Out[65]:
```
1    45.334864
3    35.992223
2    18.672914
Name: designation, dtype: float64
```

In [66]:
```
graph = sns.countplot(x=df_merged1['designation'])
for c in graph.containers:
    labels = [f'{h/df_merged1.designation.count()*100:0.1f}%' if (h := v.get_height()) > 0 else '' for v in c]
    graph.bar_label(c, labels=labels, label_type='edge', fontsize = 10)
plt.title("Count plot for Designation flag")
plt.show()
```



Observations :

- 36 % employees belongs to designation flag 3 who earns less than average salary in thier company's department having same years of experience.
- 18.7 % employees belongs to designation flag 2 who earns between average salary and 75 percentile in thier companies department with same year of experience.
- 45.3 % employees earns more than 75% in their companies deaprtment with same years of experience.

Creating **Class** Flag & Insights

- Analysis at Company & Job Position level.

> We divide the job positions by ctc
>
> - Job positions with ctc greater than 15% of the average ctc in company department is categorised as class 1
> - Job positions with ctc within (+-)15% of the average ctc is categorised as class 2
> - Job positions with ctc less than 15% of the average ctc in company department is categorised as class 3

Summary of CTC on the basis of Company and Job Position

In [67]:
```
d2 =  df.groupby(['company_hash' ,'job_position'])['ctc'].describe().round(2)
df_merged2=df.merge(d2, on=['job_position','company_hash'], how='left')
df_merged2.sort_values(['job_position','company_hash']).head(10)
```

Out[67]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	count	mean	std	min	25%	50%	75%
28049	other	2014	500000	a group chat application	2021	9	1.0	500000.0	NaN	500000.0	500000.0	500000.0	500000.0
81490	other	2018	500000	abap developer	2021	5	1.0	500000.0	NaN	500000.0	500000.0	500000.0	500000.0
77461	zgn vuurxwvmrt vwwghzn	2012	2000000	account	2021	11	1.0	2000000.0	NaN	2000000.0	2000000.0	2000000.0	2000000.0
34445	other	2015	500000	administrative clerk	2021	8	1.0	500000.0	NaN	500000.0	500000.0	500000.0	500000.0
3974	other	2020	380000	administrator	2020	3	2.0	1940000.0	2206173.16	380000.0	1160000.0	1940000.0	2720000.0
34297	other	2007	3500000	administrator	2021	16	2.0	1940000.0	2206173.16	380000.0	1160000.0	1940000.0	2720000.0
33657	exo	2018	360000	advisor	2021	5	1.0	360000.0	NaN	360000.0	360000.0	360000.0	360000.0
79472	uxnztj mgfto	2002	1650000	advisory consultant uiux expert	2019	21	1.0	1650000.0	NaN	1650000.0	1650000.0	1650000.0	1650000.0
68156	other	2016	1290000	advisory software engineer	2019	7	1.0	1290000.0	NaN	1290000.0	1290000.0	1290000.0	1290000.0
72203	xmb	2014	680000	advisory system analyat	2021	9	1.0	680000.0	NaN	680000.0	680000.0	680000.0	680000.0

In [68]:
```
def Class(x,x_50,x_75):
    if x >= x_75:
        return 1
    elif x >= x_50 and x <= x_75:
        return 2
    elif x < x_50:
        return 3
```

In [69]:
```
df_merged2['Class'] =  df_merged2.apply(lambda x: Class(x['ctc'],x['50%'],x['75%']), axis=1)
df_merged2.head()
```

Out[69]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	count	mean	std	min	25%	50%	75%	
0	atrgxnnt xzaxv	2016	1100000	other	2020	7	2.0	1085000.00	21213.20	1070000.0	1077500.0	1085000.0	1092500.0	110
1	qtrxvzwt xzegwgbb rxbxnta	2018	449999	fullstack engineer	2019	5	25.0	988199.96	487499.79	300000.0	600000.0	850000.0	1380000.0	200
2	other	2015	2000000	backend engineer	2020	8	7821.0	1098427.31	984591.46	1000.0	430000.0	880000.0	1500000.0	700
3	ngpgutaxv	2017	700000	backend engineer	2019	6	25.0	1500000.00	677212.42	520000.0	1050000.0	1540000.0	1800000.0	350
4	qxen sqghu	2017	1400000	fullstack engineer	2019	6	3.0	846666.67	480138.87	540000.0	570000.0	600000.0	1000000.0	140

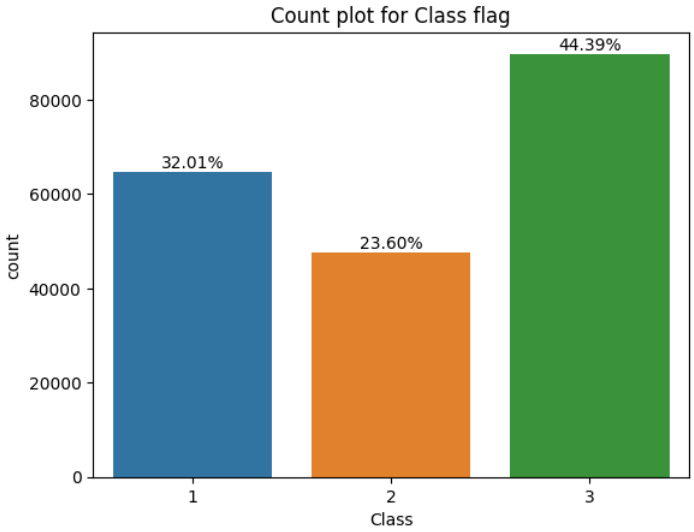
In [70]:
```
df_merged2['Class'].value_counts()
```

Out[70]:
```
3    89725
1    64698
2    47704
Name: Class, dtype: int64
```

In [71]:
```
df_merged2['Class'].value_counts(normalize=True) * 100
```

Out[71]:
```
3    44.390408
1    32.008589
2    23.601003
Name: Class, dtype: float64
```

In [72]:
```
graph = sns.countplot(x=df_merged2['Class'])
for c in graph.containers:
    labels = [f'{h/df_merged2.Class.count()*100:0.2f}%' if (h := v.get_height()) > 0 else '' for v in c]
    graph.bar_label(c, labels=labels, label_type='edge', fontsize = 10)
plt.title("Count plot for Class flag")
plt.show()
```



Observations :

- 44.39 % employees belongs to class 3 who earns less than average salary in thier companies department.
- 23.6 % employees earns between average and 75% salary of thier comapnies department with class flag as 2
- 32 % employees belongs to class 1 who earns more than 75 percentile of employees in thier companies department.

Creating **Tier** Flag & Insights

- Analysis at Company level.

> We are clustering based on ctc of each individual learner.
>
> - The learners whose ctc is greater than 15% of the average ctc of same job position is assigned tier 1.
> - The learners whose ctc is within (+-)15% of the average ctc of same job position is assigned tier 2.
> - The learners whose ctc is less than 15% of the average ctc of same job position is assigned tier 3.

Summary of CTC on the basis of Company

In [73]:
```
d3 =  df.groupby(['company_hash'])['ctc'].describe().round(2)
df_merged3=df.merge(d3, on=['company_hash'], how='left')
df_merged3.sort_values(['company_hash']).head(10)
```

Out[73]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	count	mean	std	min	25%	50%	75%	
149107	a ntwyzgrgsxto	2014	710000	android engineer	2019	9	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
193224	a ntwyzgrgsxto	2016	500000	frontend engineer	2019	7	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
70898	a ntwyzgrgsxto	2006	4000000	other	2021	17	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
93189	a ntwyzgrgsxto	2006	4000000	other	2021	17	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
8193	a ntwyzgrgsxto	2014	600000	backend engineer	2020	9	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
31334	a ntwyzgrgsxto	2016	500000	other	2021	7	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
4351	a ntwyzgrgsxto	2013	1350000	other	2020	10	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
168723	a ntwyzgrgsxto	2006	3150000	other	2019	17	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
47945	a ntwyzgrgsxto	2021	350000	fullstack engineer	2021	2	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(
67257	a ntwyzgrgsxto	2013	1350000	backend engineer	2020	10	16.0	1234687.5	1277246.35	350000.0	500000.0	600000.0	1350000.0	4(

In [74]:
```
def Tier(x,x_50,x_75):
    if x >= x_75:
        return 1
    elif x >= x_50 and x <= x_75:
        return 2
    elif x < x_50:
        return 3
```

In [75]:
```
df_merged3['tier'] =df_merged3.apply(lambda x: Tier(x['ctc'],x['50%'],x['75%']), axis=1)
df_merged3.head()
```

Out[75]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	count	mean	std	min	25%	50%	75%	
0	atrgxnnt xzaxv	2016	1100000	other	2020	7	9.0	1115666.67	458111.89	500000.0	800000.0	1070000.0	1500000.0	177
1	qtrxvzwt xzegwgbb rxbxnta	2018	449999	fullstack engineer	2019	5	423.0	1182644.90	909017.93	10000.0	600000.0	900000.0	1640000.0	700
2	other	2015	2000000	backend engineer	2020	8	49777.0	1040251.93	961919.66	15.0	420000.0	750000.0	1300000.0	707
3	ngpgutaxv	2017	700000	backend engineer	2019	6	70.0	1713928.57	967769.32	200000.0	1100000.0	1400000.0	2000000.0	470
4	qxen sqghu	2017	1400000	fullstack engineer	2019	6	6.0	940000.00	389871.77	540000.0	625000.0	850000.0	1300000.0	140

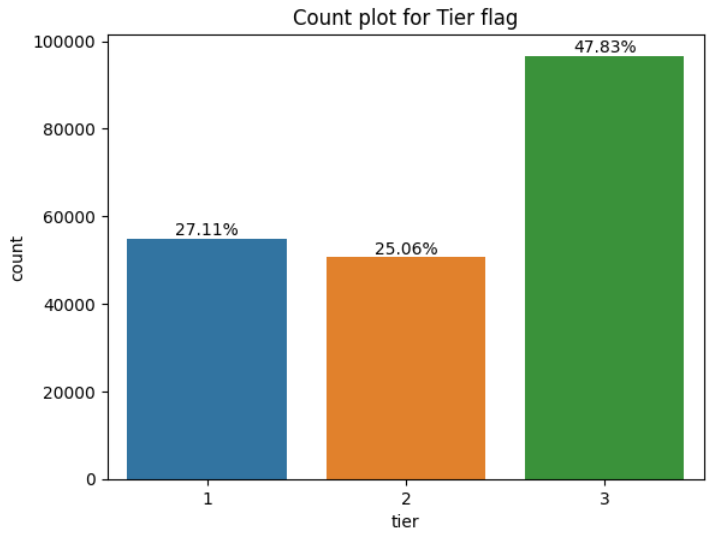
In [76]:
```
df_merged3['tier'].value_counts()
```

Out[76]:
```
3    96684
1    54788
2    50655
Name: tier, dtype: int64
```

In [77]:
```
df_merged3['tier'].value_counts(normalize=True) * 100
```

Out[77]:
```
3    47.833293
1    27.105731
2    25.060977
Name: tier, dtype: float64
```

In [78]:
```
graph = sns.countplot(x=df_merged3['tier'])
for c in graph.containers:
    labels = [f'{h/df_merged3.tier.count()*100:0.2f}%' if (h := v.get_height()) > 0 else '' for v in c]
    graph.bar_label(c, labels=labels, label_type='edge', fontsize = 10)
plt.title("Count plot for Tier flag")
plt.show()
```



Observations :

- 27.11 % employees belongs to Tier 1 companies
- 25% of employees are from Tier 2 companies
- 47.70% employees belongs to Tier 3 companies

In [79]:
```
df_merged1.drop(columns=['count','mean','std','min','25%','50%','75%','max'],inplace=True)
df_merged2.drop(columns=['count','mean','std','min','25%','50%','75%','max'],inplace=True)
df_merged3.drop(columns=['count','mean','std','min','25%','50%','75%','max'],inplace=True)
```

In [80]:
```
df_merged = df_merged2.merge(df_merged1, on=['company_hash','orgyear','ctc','job_position','exp_years','ctc_updated_year'])
df_final=df_merged.merge(df_merged3, on=['company_hash','orgyear','ctc','job_position','exp_years','ctc_updated_year'])
```

In [81]:
```
df_final.rename(columns = {'Class':'class'}, inplace = True)
df_final.drop_duplicates(keep='first',inplace = True)
df_final.head()
```

Out[81]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
0	atrgxnnt xzaxv	2016	1100000	other	2020	7	1	1	2
1	qtrxvzwt xzegwgbb rxbxnta	2018	449999	fullstack engineer	2019	5	3	3	3
2	other	2015	2000000	backend engineer	2020	8	1	1	1
345	ngpgutaxv	2017	700000	backend engineer	2019	6	3	3	3
346	qxen sqghu	2017	1400000	fullstack engineer	2019	6	1	1	1

Bivariate Analysis

In [82]:
```
df_final.shape
```

Out[82]: (156490, 9)

In [83]:
```
pd.crosstab(df_final['class'], df_final['designation'], normalize= True)
```

Out[83]:

class \ designation	1	2	3
1	0.319151	0.029420	0.011847
2	0.098313	0.073724	0.047581
3	0.100000	0.048418	0.271545

In [84]:
```
pd.crosstab(df_final['class'], df_final['tier'], normalize= True)
```

Out[84]:

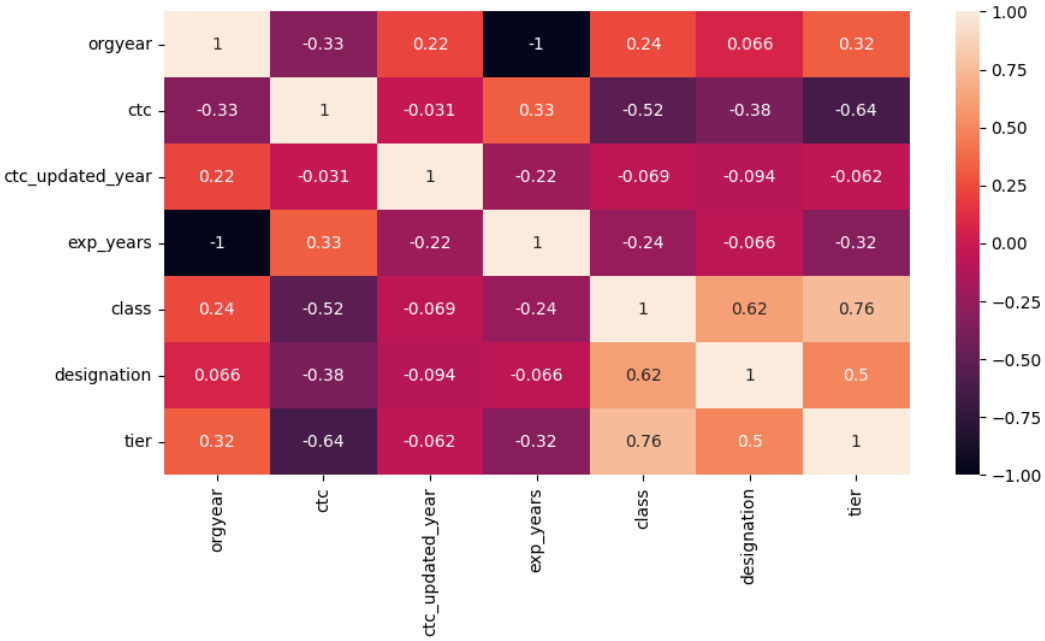
class \ tier	1	2	3
1	0.259557	0.061876	0.038987
2	0.028372	0.146329	0.044917
3	0.011943	0.038386	0.369634

In [85]:
```
pd.crosstab(df_final['designation'], df_final['tier'], normalize= True)
```

Out[85]:

designation \ tier	1	2	3
1	0.250323	0.127356	0.139785
2	0.031178	0.068049	0.052336
3	0.018372	0.051185	0.261416

In [86]:
```
plt.figure(figsize=(10,5))
ax = sns.heatmap(df_final.corr(), annot=True)
```



Questionaire based on Manual Clustering

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

In [87]:
```
df_final[(df_final['tier']==1) ].nlargest(n=10, columns=['ctc'])
```

Out[87]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
23977272	other	2001	7070000	program manager	2021	22	1	1	1
23978688	other	2001	7070000	other	2021	22	1	1	1
24468007	ko	2018	7070000	fullstack engineer	2019	5	1	1	1
24543955	vbvkgz	2017	7050000	fullstack engineer	2021	6	1	1	1
24610406	lhon axvr rxbxnta	2014	7050000	other	2019	9	1	1	1
24632378	lxg urvnegqbo rxbxnta	2006	7040000	backend architect	2020	17	1	1	1
24605004	bxwqgogen	2015	7030000	backend engineer	2021	8	1	1	1
24575986	sggsrt	2015	7020000	backend engineer	2019	8	1	1	1
6429768	other	2020	7000000	other	2020	3	1	1	1
6693532	xzntr wgqugqvnxgz	2014	7000000	other	2020	9	1	1	1

All top 10 is having above 7000000 CTC and majority of employees are fullstack/ backend engineer.

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

-- Assuming that company hash: bxwqgogen corresponds to Amazon

In [88]:
```
df_final[(df_final['job_position'].str.contains('(?i)data scientist')) &  (df_final['company_hash']=='bxwqgogen') & (df_final['class']==1) ].nlarges
```

Out[88]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
24515306	bxwqgogen	2016	6000000	data scientist	2021	7	1	1	1
24585816	bxwqgogen	2011	4500000	data scientist	2018	12	1	1	1
24604649	bxwqgogen	2016	4500000	data scientist	2020	7	1	1	1
24589523	bxwqgogen	2020	4330000	data scientist	2020	3	1	1	1
24558000	bxwqgogen	2016	3700000	data scientist	2019	7	1	2	1
24120955	bxwqgogen	2013	3600000	data scientist	2021	10	1	1	1
24608850	bxwqgogen	2015	3500000	data scientist	2020	8	1	1	1
24635732	bxwqgogen	2011	3500000	data scientist	2020	12	1	3	1
24615135	bxwqgogen	2008	3440000	data scientist	2019	15	1	1	2

2b. Bottom 10 employees of data science in Amazon / TCS etc earning less than their peers - Class 1

In [89]:
```
df_final[(df_final['job_position'].str.contains('(?i)data scientist')) &  (df_final['company_hash']=='bxwqgogen')& (df_final['class']==1) ].nsmalles
```

Out[89]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
24615135	bxwqgogen	2008	3440000	data scientist	2019	15	1	1	2
24608850	bxwqgogen	2015	3500000	data scientist	2020	8	1	1	1
24635732	bxwqgogen	2011	3500000	data scientist	2020	12	1	3	1
24120955	bxwqgogen	2013	3600000	data scientist	2021	10	1	1	1
24558000	bxwqgogen	2016	3700000	data scientist	2019	7	1	2	1
24589523	bxwqgogen	2020	4330000	data scientist	2020	3	1	1	1
24585816	bxwqgogen	2011	4500000	data scientist	2018	12	1	1	1
24604649	bxwqgogen	2016	4500000	data scientist	2020	7	1	1	1
24515306	bxwqgogen	2016	6000000	data scientist	2021	7	1	1	1

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

-- Assuming that company hash: bxwqgogen corresponds to Amazon

In [90]:
```
df_final[(df_final['job_position'].str.contains('(?i)data scientist')) &  (df_final['company_hash']=='bxwqgogen')& (df_final['class']==3) ].nsmalles
```

Out[90]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
24626750	bxwqgogen	2016	94000	data scientist	2019	7	3	3	3
24539687	bxwqgogen	2012	160000	data scientist	2015	11	3	3	3
21271933	bxwqgogen	2019	200000	data scientist	2019	4	3	3	3
24479408	bxwqgogen	2013	200000	data scientist	2020	10	3	3	3
24560680	bxwqgogen	2009	300000	data scientist	2016	14	3	1	3
24610957	bxwqgogen	2012	380000	data scientist	2019	11	3	1	3
24551158	bxwqgogen	2019	500000	data scientist	2021	4	3	3	3
24579252	bxwqgogen	2018	610000	data scientist	2019	5	3	3	3
24553253	bxwqgogen	2016	700000	data scientist	2016	7	3	3	3
24560210	bxwqgogen	2016	700000	data scientist	2017	7	3	3	3

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

-- Assuming that company hash: bxwqgogen corresponds to Amazon

In [91]:
```
df_final[(df_final['job_position'].str.contains('(?i)data scientist')) &  (df_final['company_hash']=='bxwqgogen')& (df_final['class']==3) ].nsmalles
```

Out[91]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
24626750	bxwqgogen	2016	94000	data scientist	2019	7	3	3	3
24539687	bxwqgogen	2012	160000	data scientist	2015	11	3	3	3
21271933	bxwqgogen	2019	200000	data scientist	2019	4	3	3	3
24479408	bxwqgogen	2013	200000	data scientist	2020	10	3	3	3
24560680	bxwqgogen	2009	300000	data scientist	2016	14	3	1	3
24610957	bxwqgogen	2012	380000	data scientist	2019	11	3	1	3
24551158	bxwqgogen	2019	500000	data scientist	2021	4	3	3	3
24579252	bxwqgogen	2018	610000	data scientist	2019	5	3	3	3
24553253	bxwqgogen	2016	700000	data scientist	2016	7	3	3	3
24560210	bxwqgogen	2016	700000	data scientist	2017	7	3	3	3

- In Class 1 highest CTC is 6000000 whereas in class 3 It is 8600000 which is very less compare to 1st one.
- In Class 1 10th employee CTC is 3000000 whereas in class 3 It is 2495000.
- Average CTC for data scientist in class-1 is 6020000 whereas in class-3 3643500, which is around 40% less.

4. Bottom 10 employees (earning less than most of the employees in the company) - Tier3

In [92]:
```
df_final[(df_final['tier']==3) ].nsmallest(n=10, columns=['ctc'])
```

Out[92]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
24564567	xzntqcxtfmxn	2014	2	backend engineer	2019	9	3	3	3
24533030	xzntqcxtfmxn	2013	6	other	2018	10	3	3	3
24523375	xzntqcxtfmxn	2013	14	other	2018	10	3	1	3
24623429	other	2016	15	other	2018	7	3	3	3
24622233	other	2018	16	other	2018	5	3	3	3
24080396	other	2020	24	other	2020	3	3	3	3
24446209	other	2016	25	android engineer	2018	7	3	3	3
24529127	other	2021	200	other	2021	2	3	3	3
24604114	other	2013	300	database administrator	2019	10	3	3	3
24382332	other	2018	500	cofounder	2019	5	3	3	3

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

In [93]:
```
df_final[(df_final['tier']==1) & (df_final['class']==1) & (df_final['designation']==1) &  (df_final['company_hash']=='bxwqgogen') & (df_final['exp_y
```

Out[93]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
24626475	bxwqgogen	2016	6500000	backend engineer	2021	7	1	1	1
21959845	bxwqgogen	2017	6200000	backend engineer	2020	6	1	1	1
21993817	bxwqgogen	2017	6000000	backend engineer	2020	6	1	1	1
24515306	bxwqgogen	2016	6000000	data scientist	2021	7	1	1	1
24573699	bxwqgogen	2017	5900000	other	2019	6	1	1	1
24530515	bxwqgogen	2016	5700000	software engineer full stack	2021	7	1	1	1
4047415	bxwqgogen	2018	5600000	fullstack engineer	2020	5	1	1	1
24631248	bxwqgogen	2016	5520000	backend engineer	2020	7	1	1	1
24533380	bxwqgogen	2017	5500000	fullstack engineer	2021	6	1	1	1
24543723	bxwqgogen	2016	5400000	backend engineer	2020	7	1	1	1

6. Top 10 companies (based on their CTC)

In [94]:
```
d3.sort_values(['50%','count'], ascending=False).head(10)
```

Out[94]:

company_hash	count	mean	std	min	25%	50%	75%	max
bgngqi	6.0	4133333.33	720185.16	2700000.0	4200000.0	4500000.0	4500000.0	4500000.0
vruyvsqtu otwhqxnxto	22.0	4009090.91	1368057.05	1600000.0	3250000.0	4325000.0	4550000.0	6000000.0
qhmqxp xzw	33.0	3854545.45	1613668.36	800000.0	3200000.0	4000000.0	5100000.0	7000000.0
zgctroyxu	6.0	3866666.67	637704.22	2600000.0	4000000.0	4000000.0	4225000.0	4300000.0
ngftq qtotvqwy wvuxnvr	45.0	3546533.33	1665411.89	207000.0	2600000.0	3900000.0	4600000.0	7000000.0
avnvdh	11.0	3178181.73	1156363.53	1500000.0	2004999.5	3900000.0	4175000.0	4300000.0
ovvcz	6.0	3466666.67	1508199.81	1500000.0	2375000.0	3800000.0	4100000.0	5600000.0
axsxnvr yvqmgq	6.0	3318333.33	1404854.68	600000.0	3362500.0	3700000.0	4000000.0	4560000.0
xzoxatcxtf ntwyzgrgsxto xzw	10.0	3268000.00	1083038.93	1560000.0	2350000.0	3600000.0	3950000.0	4600000.0
viq wvuxnvr bvzvstbtzn	9.0	3466666.67	708872.34	2500000.0	2900000.0	3600000.0	3700000.0	4500000.0

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

In [95]:
```
sorted_df = df_final.sort_values(['company_hash', 'ctc'], ascending=[True, False])
df_ = sorted_df.groupby('company_hash').head(2)
df_.head(10)
```

Out[95]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	exp_years	class	designation	tier
24276253	a ntwyzgrgsxto	2006	4000000	other	2021	17	1	1	1
24610029	a ntwyzgrgsxto	2006	3150000	other	2019	17	2	3	1
24261068	aaqxctz avnv owxtzwto vzvrjnxwo ucn rna	2012	3600000	other	2018	11	1	1	1
24550336	aaqxctz avnv owxtzwto vzvrjnxwo ucn rna	2015	1400000	backend engineer	2015	8	1	1	1
24592269	adw ntwyzgrgsj	2012	5000000	engineering leadership	2019	11	1	1	1
24639637	adw ntwyzgrgsj	2012	5000000	product manager	2019	11	1	1	1
24555659	adw ntwyzgrgsxto	2003	4000000	other	2019	20	1	1	1
24615283	adw ntwyzgrgsxto	2001	3600000	engineering leadership	2019	22	1	1	1
23488134	agdutq	2015	2500000	backend architect	2019	8	1	1	1
24376458	agdutq	2015	2500000	backend engineer	2019	8	1	1	1

Unsupervised Learning - Clustering

Data processing for Unsupervised clustering

In [96]:
```
df_final=df_final.drop(['orgyear','ctc_updated_year'],axis=1)
# df_final.drop_duplicates(keep='first',inplace = True)
df_final.head()
```

Out[96]:

	company_hash	ctc	job_position	exp_years	class	designation	tier
0	atrgxnnt xzaxv	1100000	other	7	1	1	2
1	qtrxvzwt xzegwgbb rxbxnta	449999	fullstack engineer	5	3	3	3
2	other	2000000	backend engineer	8	1	1	1
345	ngpgutaxv	700000	backend engineer	6	3	3	3
346	qxen sqghu	1400000	fullstack engineer	6	1	1	1

Target Encoding / One Hot Encoding

-- For categorical features company_hash and job_position

In [97]:
```
from sklearn.preprocessing import OneHotEncoder
from category_encoders import TargetEncoder

cat_df=df_final.select_dtypes('object')
targ_enc = TargetEncoder()
cat_df = targ_enc.fit_transform(cat_df, df_final['ctc'])
cat_df.head()
```

Out[97]:

	company_hash	job_position
0	1.302111e+06	1.278593e+06
1	1.218111e+06	1.275749e+06
2	1.197725e+06	1.512525e+06
345	1.726185e+06	1.512525e+06
346	1.280265e+06	1.275749e+06

Standardization of Data

-- For numerical columns

In [98]:
```
num_df = df_final.select_dtypes('int')
data = num_df.join(cat_df)
data.head()
```

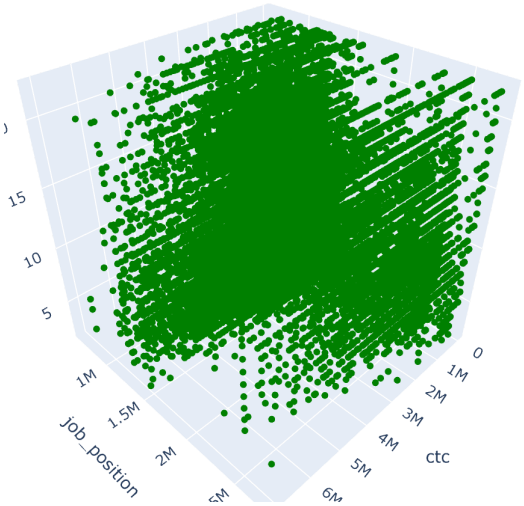
Out[98]:

	ctc	exp_years	class	designation	tier	company_hash	job_position
0	1100000	7	1	1	2	1.302111e+06	1.278593e+06
1	449999	5	3	3	3	1.218111e+06	1.275749e+06
2	2000000	8	1	1	1	1.197725e+06	1.512525e+06
345	700000	6	3	3	3	1.726185e+06	1.512525e+06
346	1400000	6	1	1	1	1.280265e+06	1.275749e+06

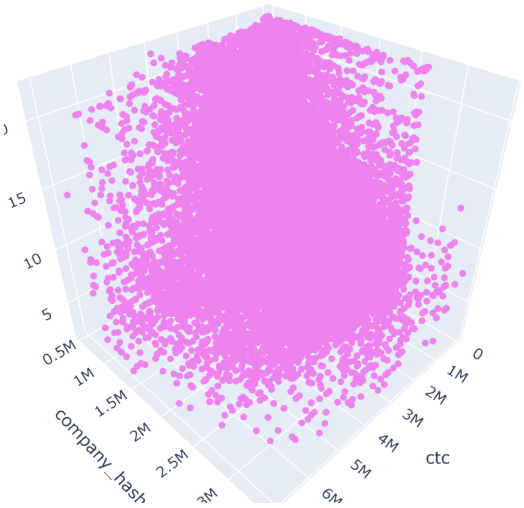
In [99]:
```
data.shape
```

Out[99]:
```
(156490, 7)
```

In [100]:
```
fig = px.scatter_3d(data, x='ctc', y='job_position', z='exp_years', width=600, height=600)
fig.update_traces(marker=dict(size=2,color = "green"), selector=dict(mode='markers'))
fig.show()
```



In [101]:
```
fig = px.scatter_3d(data, x='ctc', y='company_hash', z='exp_years',  width=600, height=600)
fig.update_traces(marker=dict(size=2,color='violet'), selector=dict(mode='markers'))
fig.show()
```



In [102]:
```
from sklearn.preprocessing import  MinMaxScaler
scaler =  MinMaxScaler()

data_scaled = scaler.fit_transform(data)
data_scaled = pd.DataFrame(data_scaled)
data_scaled.columns =data.columns
pd.DataFrame(data_scaled).describe().T
```

Out[102]:

	count	mean	std	min	25%	50%	75%	max
ctc	156490.0	0.192952	0.156260	0.0	0.084865	0.145686	0.254597	1.0
exp_years	156490.0	0.313881	0.204372	0.0	0.150000	0.250000	0.400000	1.0
class	156490.0	0.529772	0.440693	0.0	0.000000	0.500000	1.000000	1.0
designation	156490.0	0.406754	0.451017	0.0	0.000000	0.000000	1.000000	1.0
tier	156490.0	0.576832	0.427142	0.0	0.000000	0.500000	1.000000	1.0
company_hash	156490.0	0.292193	0.170073	0.0	0.189602	0.231749	0.370495	1.0
job_position	156490.0	0.350302	0.150002	0.0	0.308398	0.309729	0.419195	1.0

Checking clustering tendency by Hopkins Clustering Tendency Test

- The **Hopkins Clustering Tendency Test** is a way to assess the cluster tendency of a data set by measuring the probability that a given data set is generated by a uniform data distribution.
- The Hopkins statistic, is a statistic which gives a value which indicates the cluster tendency i.e, how well the data can be clustered.
- The null and the alternative hypotheses are defined as follow:
  - Null hypothesis: the data set X is uniformly distributed (i.e., no meaningful clusters)
  - Alternative hypothesis: the data set X is not uniformly distributed (i.e., contains meaningful clusters)
- Therefore, we can interpret Hopkins' statistic in the following manner:
  - If the value is between {0.01, ...,0.3}, the data is regularly spaced.
  - If the value is around 0.5, it is random.
  - If the value is between {0.7, ..., 0.99}, it has a high tendency to cluster.
- We can conduct the Hopkins Statistic test iteratively, using 0.5 as the threshold to reject the alternative hypothesis.
  - If Hopkins statistic (H) < 0.5, then it is unlikely that dataset D has statistically significant clusters.
  - If the value of Hopkins statistic (H) is close to 1, then we can reject the null hypothesis and conclude that the dataset D is significantly a clusterable data.

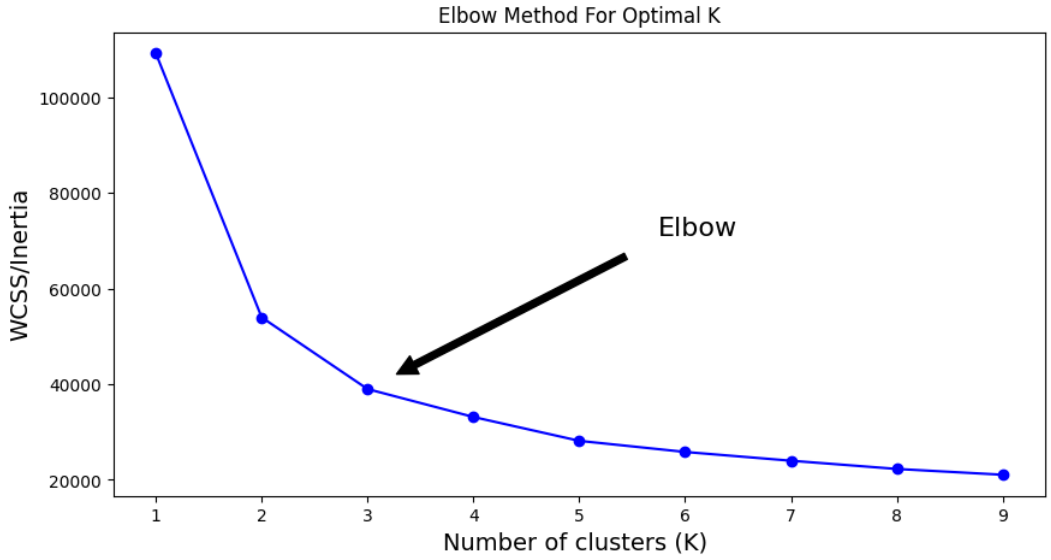
In [103]:
```
from pyclustertend import hopkins
print('H statistic :',hopkins(data_scaled,data_scaled.shape[0]))
```

```
H statistic : 0.031130160950253984
```

- After checking Hopkins Clustering Tendency Test we got 0.031 which is small than 0.5
- Hopkins statistic (H) < 0.5, which means it is unlikely that dataset D has statistically significant clusters.We can conclude data is uniformly distributed.
- But we proceed to check the clustering algorithms.

Elbow method to get optimal number of clusters (K)

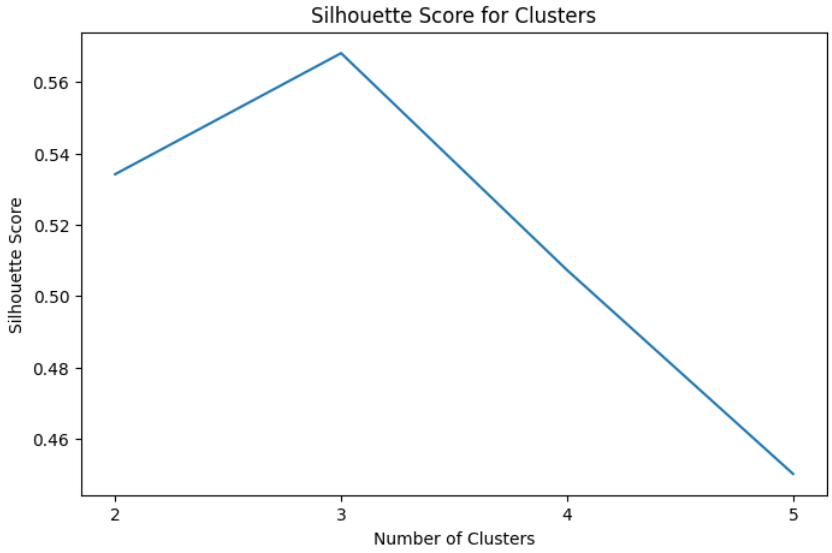
In [104]:
```
from sklearn.cluster import KMeans
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(data_scaled) for k in range(1,10)]
inertias = [model.inertia_ for model in kmeans_per_k]

plt.figure(figsize=(10,5))
plt.plot(range(1,10),inertias,"bo-")
plt.xlabel('Number of clusters (K) ',fontsize=14)
plt.ylabel('WCSS/Inertia',fontsize=14)
plt.title('Elbow Method For Optimal K')
plt.annotate('Elbow',
 xy=(3,inertias[2]),
 xytext=(0.55,0.55),
 textcoords='figure fraction',
 fontsize=16,
 arrowprops=dict(facecolor='black',
 shrink=0.1))
plt.show()
```



From elbow method, we get inflaction point at 3 which represents the optimal number of clusters to be 3.

Silhouette score

In [140]:
```
# silhoutte score calculation
from sklearn.metrics import silhouette_score
silhouette_scores = []
for k in range(2, 6):
    kmeans = KMeans(n_clusters=k,random_state = 20)
    kmeans.fit(data_scaled)
    score = silhouette_score(data_scaled, kmeans.labels_)
    silhouette_scores.append(score)

#plotting silhouette scores
plt.figure(figsize = (8,5))
plt.plot(range(2, 6), silhouette_scores)
plt.xticks(range(2, 6))
plt.title("Silhouette Score for Clusters")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.show()
```



The silhouette score is high when there are 3 clusters. So we choose to divide our data into 3 clusters.

KMeans Clustering with optimal K

-- From elbow method, we get optimal number of clusters = 3

In [141]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 3, random_state = 20)
kmeans.fit(data_scaled)
data_scaled['cluster_label']=kmeans.fit_predict(data_scaled)
```

In [142]:
```
kmeans.cluster_centers_
```

Out[142]:
```
array([[1.06979112e-01, 2.75875241e-01, 9.37800681e-01, 9.44566569e-01,
        9.16475973e-01, 2.92182371e-01, 3.42373623e-01, 8.34887715e-14],
       [3.14571414e-01, 3.92377887e-01, 4.13474023e-02, 9.59347512e-02,
        8.83089809e-02, 2.85085214e-01, 3.72220875e-01, 2.00000000e+00],
       [1.52373801e-01, 2.66737929e-01, 6.17517492e-01, 1.29232530e-01,
        7.46652846e-01, 3.00599690e-01, 3.33920576e-01, 1.00000000e+00]])
```

In [143]:
```
len(kmeans.labels_)
```

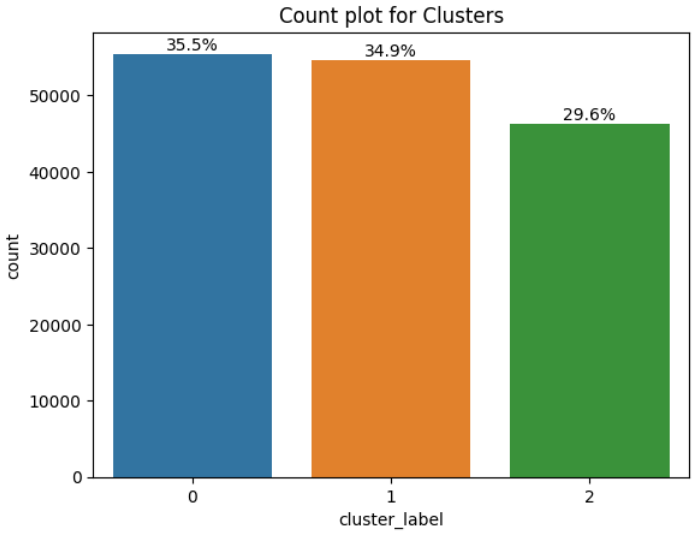
Out[143]:
```
156490
```

In [144]:
```
print(data_scaled["cluster_label"].value_counts())
print("--"*40)
print(data_scaled['cluster_label'].value_counts(normalize=True)*100)
```

```
0    55499
1    54683
2    46308
Name: cluster_label, dtype: int64
--------------------------------------------------------------------------------
0    35.464886
1    34.943447
2    29.591667
Name: cluster_label, dtype: float64
```

In [145]:
```
cluster_df = pd.DataFrame(df_final, columns=df_final.columns)
cluster_df['cluster_label'] = kmeans.labels_
```

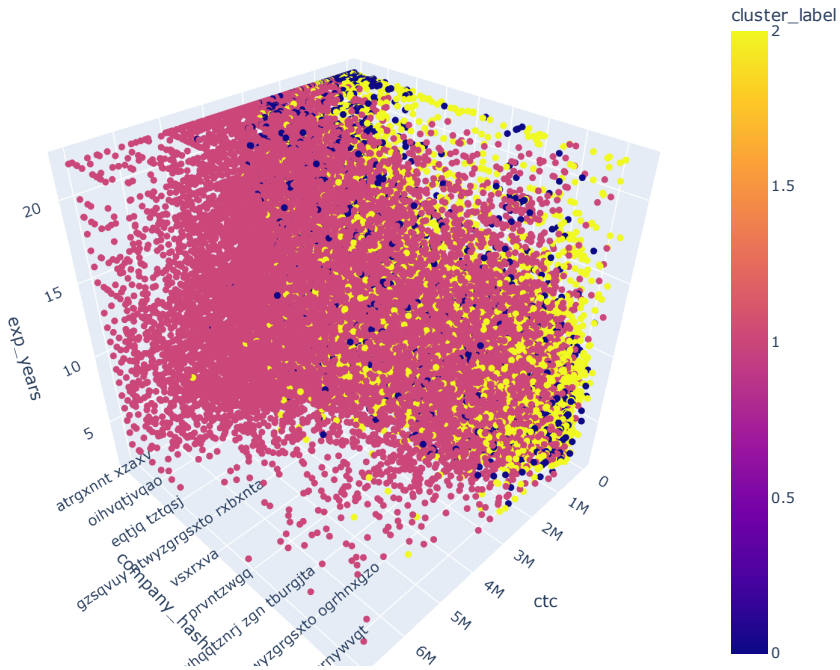
In [146]:
```
graph = sns.countplot(x=cluster_df["cluster_label"])
for c in graph.containers:
    labels = [f'{h/cluster_df["cluster_label"].count()*100:0.1f}%' if (h := v.get_height()) > 0 else '' for v in c]
    graph.bar_label(c, labels=labels, label_type='edge', fontsize = 10)
plt.title("Count plot for Clusters")
plt.show()
```



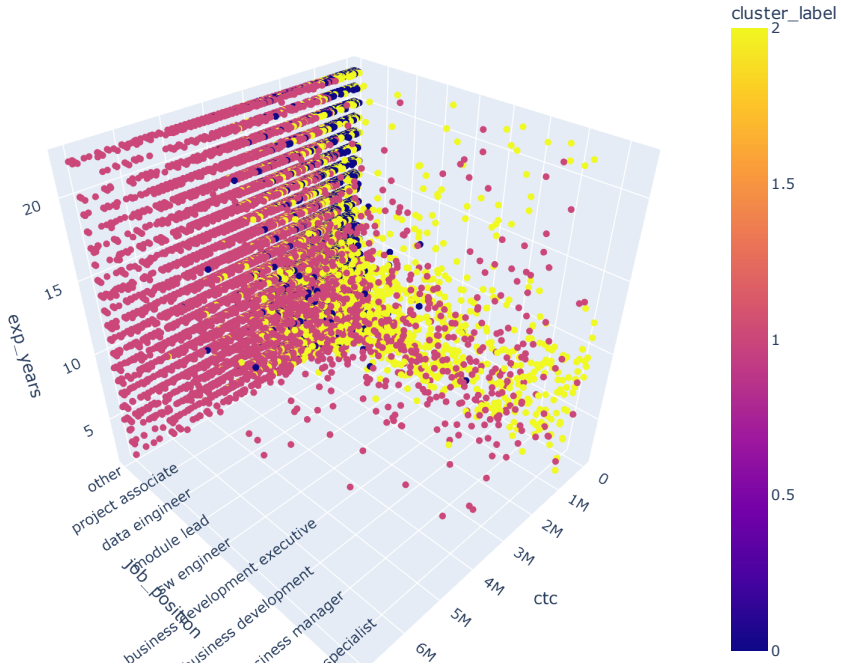
35.5% of data belongs to cluster 0, 29.6% of data belongs to cluster 1 and 34.9% of data belongs to cluster 2

Cluster Analysis by Visualization

In [155]:
```
fig = px.scatter_3d(cluster_df, x='ctc', y='company_hash', z='exp_years', color='cluster_label', width=800, height=700)
fig.update_traces(marker=dict(size=2), selector=dict(mode='markers'))
fig.show()
```

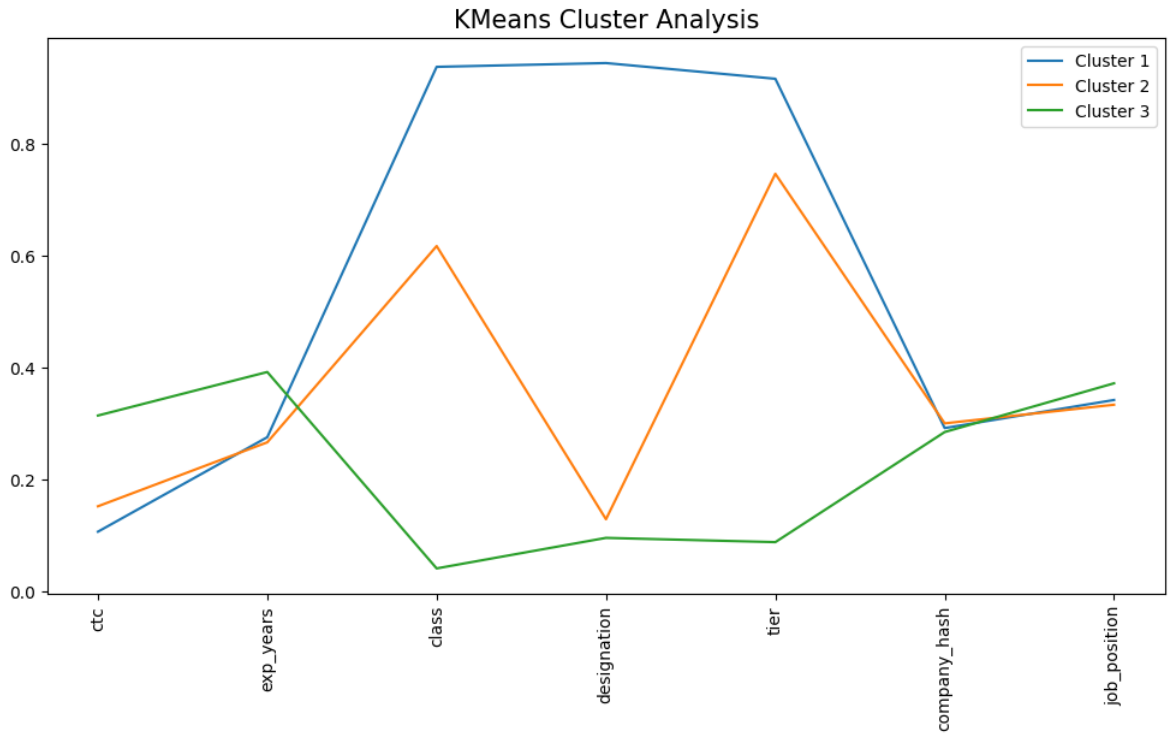


In [158]:
```
fig = px.scatter_3d(cluster_df, x='ctc', y='job_position', z='exp_years', color='cluster_label', width=800, height=700)
fig.update_traces(marker=dict(size=2), selector=dict(mode='markers'))
fig.show()
```



In [161]:
```
#Plot a line graph to see the characteristics of the clusters
y_pred = kmeans.fit_predict(data_scaled)
data_scaled['cluster_label'] = pd.Series(y_pred, index=data_scaled.index)
clustered_df = data_scaled.groupby('cluster_label').mean()
labels = ['Cluster 1', 'Cluster 2', 'Cluster 3']

plt.figure(figsize=(12,6))
plt.plot(clustered_df.T, label=labels)
plt.title('KMeans Cluster Analysis',fontsize=15)
plt.xticks(rotation=90)
plt.legend(labels)
```

Out[161]: <matplotlib.legend.Legend at 0x18009abdaf0>



Observation : Each cluster has different mean at different feature with K-means clustering

K-means++ Clustering

In [162]:
```
from sklearn.cluster import KMeans
kmeans_plus = KMeans(n_clusters = 3,init='k-means++',verbose=0, random_state = 20)
kmeans_plus.fit(data_scaled)
data_scaled['cluster_label']=kmeans_plus.fit_predict(data_scaled)
```

In [163]:
```
kmeans_plus.cluster_centers_
```

Out[163]:
```
array([[1.06979112e-01, 2.75875241e-01, 9.37800681e-01, 9.44566569e-01,
        9.16475973e-01, 2.92182371e-01, 3.42373623e-01, 8.34887715e-14],
       [3.14571414e-01, 3.92377887e-01, 4.13474023e-02, 9.59347512e-02,
        8.83089809e-02, 2.85085214e-01, 3.72220875e-01, 2.00000000e+00],
       [1.52373801e-01, 2.66737929e-01, 6.17517492e-01, 1.29232530e-01,
        7.46652846e-01, 3.00599690e-01, 3.33920576e-01, 1.00000000e+00]])
```

In [164]:
```
len(kmeans_plus.labels_)
```

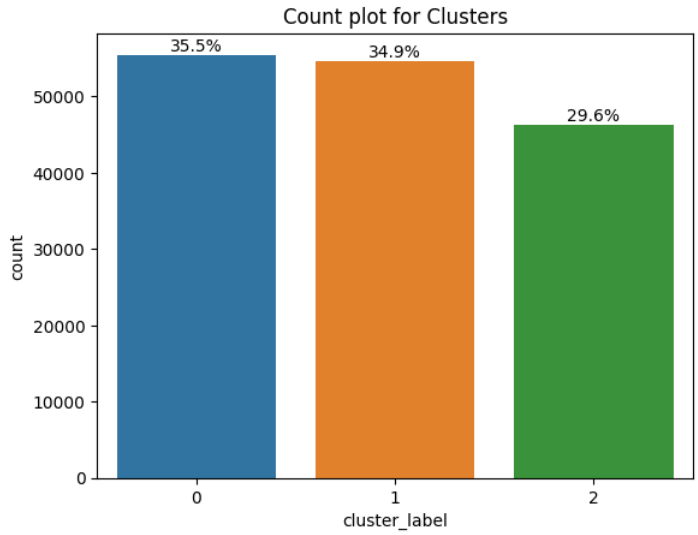
Out[164]:
```
156490
```

In [165]:
```
print(data_scaled["cluster_label"].value_counts())
print("--"*40)
print(data_scaled['cluster_label'].value_counts(normalize=True)*100)
```

```
0    55499
1    54683
2    46308
Name: cluster_label, dtype: int64
--------------------------------------------------------------------------------
0    35.464886
1    34.943447
2    29.591667
Name: cluster_label, dtype: float64
```

In [166]:
```
cluster_df1 = pd.DataFrame(df_final, columns=df_final.columns)
cluster_df1['cluster_label'] = kmeans_plus.labels_
```

In [167]:
```
graph = sns.countplot(x=cluster_df1["cluster_label"])
for c in graph.containers:
    labels = [f'{h/cluster_df1["cluster_label"].count()*100:0.1f}%' if (h := v.get_height()) > 0 else '' for v in c]
    graph.bar_label(c, labels=labels, label_type='edge', fontsize = 10)
plt.title("Count plot for Clusters")
plt.show()
```



Hierarchical clustering

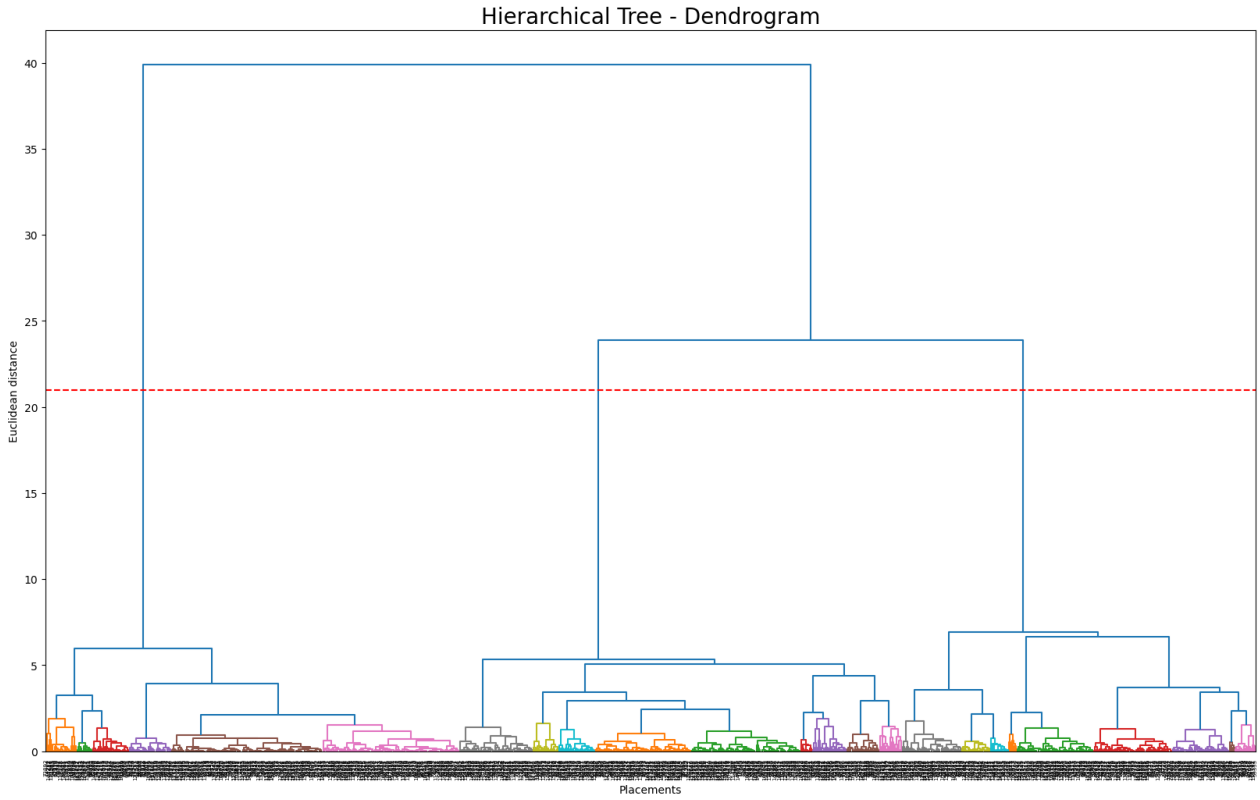
In [168]:
```
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt

sample = data_scaled.sample(1000)
sample.drop(columns='company_hash',inplace=True)
Z = sch.linkage(sample, method='ward',metric="euclidean")
```

Dendrogram

- A dendrogram is a branching diagram that represents the relationships of similarity among a group of entities.
- The main use of a dendrogram is to work out the best way to allocate objects to clusters.

In [169]:
```
# Plotting Dendrogram
fig, ax = plt.subplots(figsize=(20, 12))
sch.dendrogram(Z, labels=sample.index, ax=ax, color_threshold=2)
plt.xticks(rotation=90)
plt.title('Hierarchical Tree - Dendrogram',fontsize=20)
plt.xlabel('Placements')
plt.axhline(y = 21, color = 'r', linestyle = "--")
ax.set_ylabel('Euclidean distance')
```

Out[169]:
```
Text(0, 0.5, 'Euclidean distance')
```



Observations :

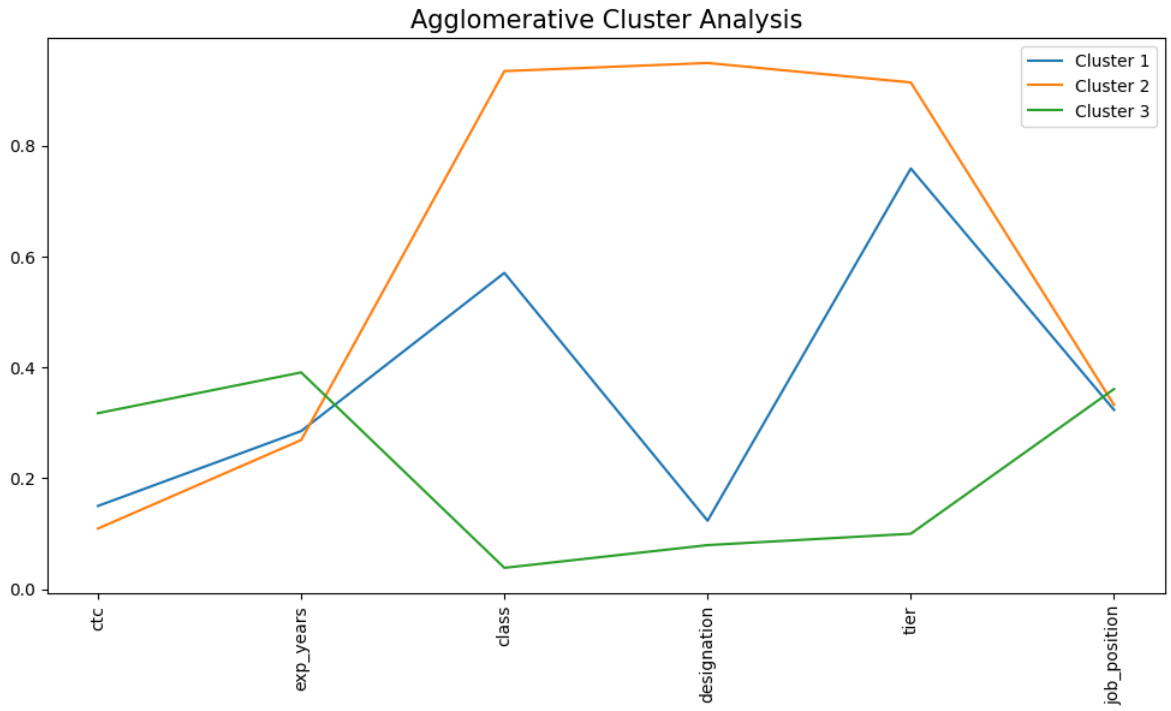
- Height of the dendogram is around 42 units,
- Threshold for dendrogram would be around 21 units, which crosses 3 vertical lines.
- This indicates that 3 seems a good indication of the number of clusters that have the most distance between them.

Agglomerative Clustering

In [170]:
```
# import hierarchical clustering libraries
from sklearn.cluster import AgglomerativeClustering

# create clusters
hc = AgglomerativeClustering(n_clusters= 3, affinity = 'euclidean', linkage = 'ward')
y_pred = hc.fit_predict(sample)
```

In [171]:
```
#Plot a line graph to see the characteristics of the clusters
sample['cluster_label'] = pd.Series(y_pred, index=sample.index)
clustered_df = sample.groupby('cluster_label').mean()
labels = ['Cluster 1', 'Cluster 2', 'Cluster 3']

plt.figure(figsize=(12,6))
plt.plot(clustered_df.T, label=labels)
plt.title('Agglomerative Cluster Analysis', fontsize = 15)
plt.xticks(rotation=90)
plt.legend(labels)
```

Out[171]:
```
<matplotlib.legend.Legend at 0x18009ec1cd0>
```

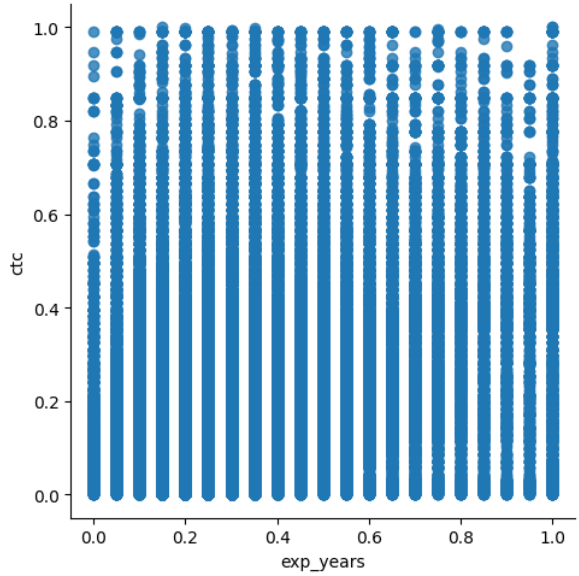


Observation : Each cluster has different mean at different feature with Agglomerative Clustering.

K-means Cluster Analysis using Visualization

K-means clustering on ctc and exp_years with 3 clusters

In [172]:
```
ctc_yoe_df = data_scaled[['ctc','exp_years']]
sns.lmplot(y='ctc',x='exp_years',data=ctc_yoe_df,fit_reg=False, legend=True)
plt.show()
```



In [173]:
```
kmeans=KMeans(n_clusters=3, random_state= 42)
kmeans.fit(ctc_yoe_df)
pred=kmeans.predict(ctc_yoe_df)
ctc_yoe_df["cluster_label"]=pd.DataFrame(pred,columns=["cluster_label"])
ctc_yoe_df.head()
```

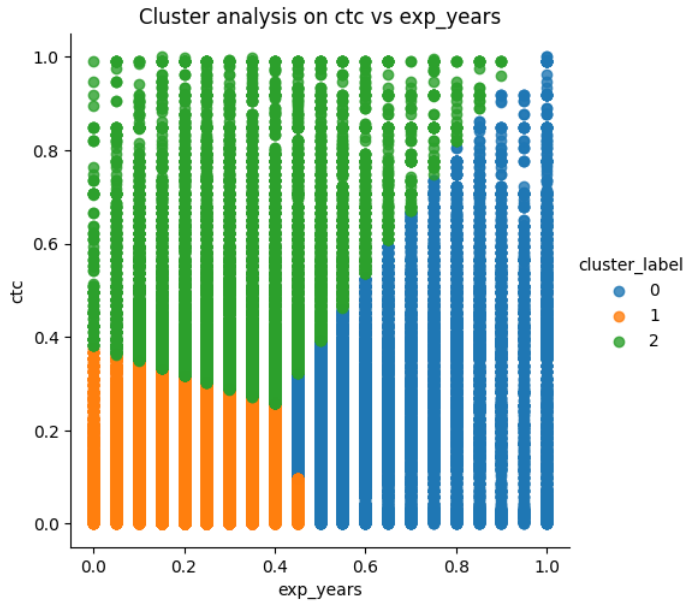
Out[173]:

	ctc	exp_years	cluster_label
0	0.155587	0.25	1
1	0.063649	0.15	1
2	0.282885	0.30	1
3	0.099010	0.20	1
4	0.198020	0.20	1

In [174]:
```
ctc_yoe_df["cluster_label"].value_counts()
```

Out[174]:
```
1    105245
0     31433
2     19812
Name: cluster_label, dtype: int64
```

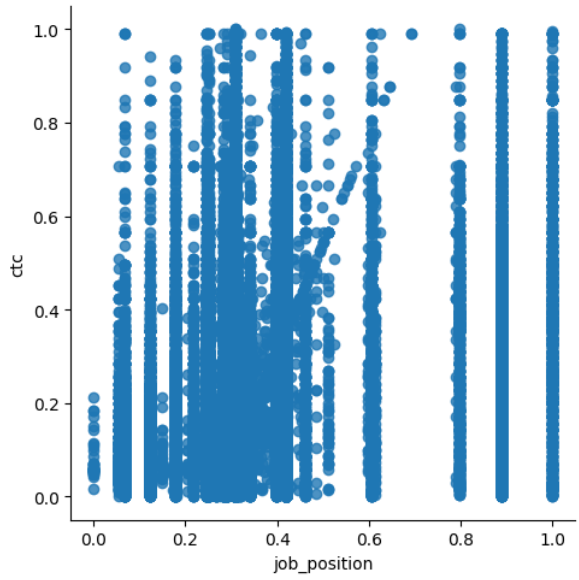
In [175]:
```
sns.lmplot(y='ctc',x='exp_years',data=ctc_yoe_df,fit_reg=False, hue='cluster_label',legend=True)
plt.title('Cluster analysis on ctc vs exp_years',fontsize=12)
plt.show()
```



- In green cluster we can find the candidate with high year of experience and it's ctc range is low to high
- In orange clustering we can observe that years of experence is low and the ctc is also low which is ideally the valid cluster we can say.
- In blue part some exceptions we can see and that is year of experience is low but their ctc is high.
- these three important clusters we can figure out from ctc and Years of experience data points

K-means clustering on ctc and job_position with 3 clusters

In [176]:
```
ctc_job_df = data_scaled[['ctc','job_position']]
sns.lmplot(y='ctc',x='job_position',data=ctc_job_df,fit_reg=False, legend=True)
plt.show()
```



In [177]:
```
kmeans=KMeans(n_clusters=3, random_state= 42)
kmeans.fit(ctc_job_df)
pred=kmeans.predict(ctc_job_df)
ctc_job_df["cluster_label"]=pd.DataFrame(pred,columns=["cluster_label"])
ctc_job_df.head()
```

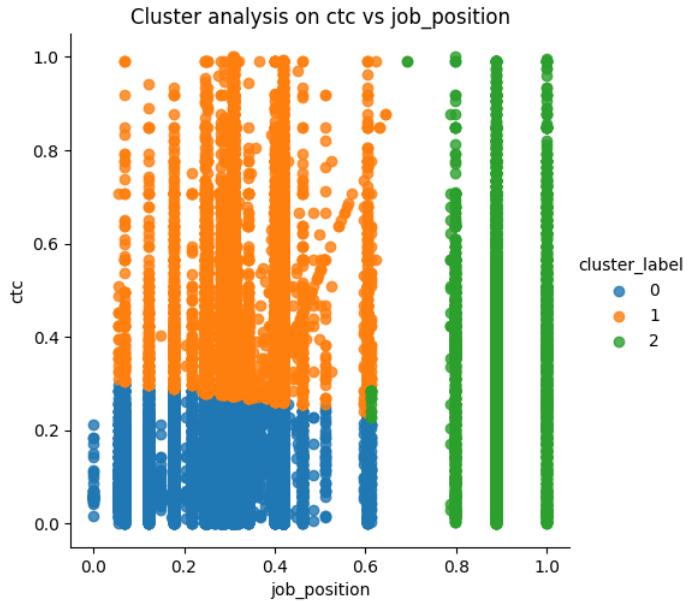
Out[177]:

	ctc	job_position	cluster_label
0	0.155587	0.309729	0
1	0.063649	0.308398	0
2	0.282885	0.419195	1
3	0.099010	0.419195	0
4	0.198020	0.308398	0

In [178]:
```
ctc_job_df["cluster_label"].value_counts()
```

Out[178]:
```
0    117971
1     30663
2      7856
Name: cluster_label, dtype: int64
```

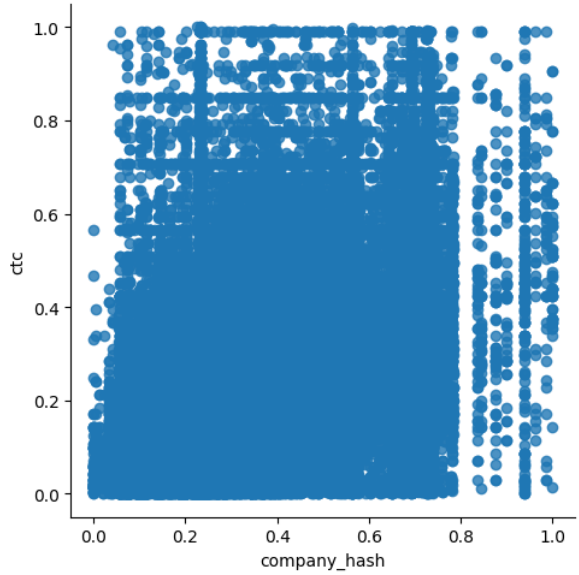
In [179]:
```
sns.lmplot(y='ctc',x='job_position',data=ctc_job_df,fit_reg=False, hue='cluster_label',legend=True)
plt.title('Cluster analysis on ctc vs job_position',fontsize=12)
plt.show()
```



- Manual clustering and Unsupervised clustering giving the almost same result on the clustering of ctc and job_position
- Blue clusters job_position with ctc less than average
- Orange clusters gives the job_position has ctc in range fron 50 to 75% approx
- Green cluster gives the job_position has higher ctc

K-means clustering on ctc and company_hash with 3 clusters

In [180]:
```
ctc_cmpy_df = data_scaled[['ctc','company_hash']]
sns.lmplot(y='ctc',x='company_hash',data=ctc_cmpy_df,fit_reg=False, legend=True)
plt.show()
```



In [181]:
```
kmeans=KMeans(n_clusters=3, random_state= 42)
kmeans.fit(ctc_cmpy_df)
```

```
pred=kmeans.predict(ctc_cmpy_df)
ctc_cmpy_df["cluster_label"]=pd.DataFrame(pred,columns=["cluster_label"])
ctc_cmpy_df.head()
```

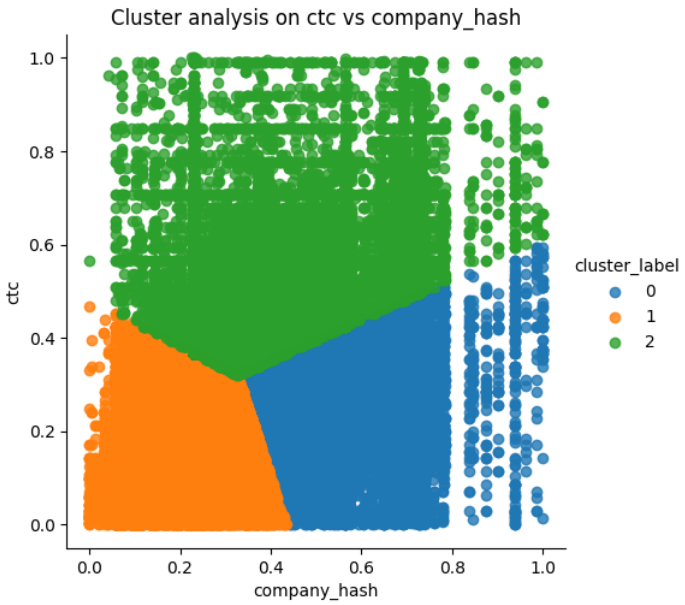
Out[181]:

	ctc	company_hash	cluster_label
0	0.155587	0.267707	1
1	0.063649	0.238772	1
2	0.282885	0.231749	1
3	0.099010	0.413787	0
4	0.198020	0.260182	1

In [182]:
```
ctc_cmpy_df["cluster_label"].value_counts()
```

Out[182]:
```
1    109828
0     29401
2     17261
Name: cluster_label, dtype: int64
```

In [183]:
```
sns.lmplot(y='ctc',x='company_hash',data=ctc_cmpy_df,fit_reg=False, hue='cluster_label',legend=True)
plt.title('Cluster analysis on ctc vs company_hash',fontsize=12)
plt.show()
```



- Manual clustering and Unsupervised clustering giving the almost same result on the clustering of ctc and company_hash
- Orange clusters company_hash with ctc less than average
- Green clusters gives the company-hash has ctc in range fron 50 to 75% approx
- Blue cluster gives the company_hash has higher ctc

Conclusions

- Around 4 % of employees are working with company having company hash "nvnv wgzohrnvzwj otqcxwto" and most of the employees joined scaler having work experience between 4 to 9 years.
- Around 25% employees working in 'Others' Job position which is null and around 21% working as backend engineers and 12 % as fullstack engineers. This may suggest that employees belongs to Others may joined Scaler for Academy as well as for DSML track.Most of the employees working as fullstack or backend engineers have been joined for upskilling in Academy track.
- Top 10 Job position in which scaler students are working : backend engineer, others, fullstack engineer,frontend engineer, engineering leadership, qa engineer, data scientist, sdet, devops engineer roles.
- 33.4 % employees belongs to designation flag 3 who earns less than average salary in their company's department having same years of experience. 45.3 % employees belongs to designation flag 3 who earns between average salary and 75 percentile in thier comapnies department with same year of experience and remaining 21.3 % employees earns more than 75% in their companies department with same years of experience.
- Top rating company 1bs, with job position pays highest with 3 years of work experience, Backend engineers with designation flag 1 earns around 21 Lakh as their median salary while employees working in engineering leadership earns around 39 lakh as thier median asalary with designation flag 1.Data scientists with designation flag 1 earns 20 lakh as thier median salary.
- 43.53 % employees belongs to class 3 who earns less than average salary in their companies department. 32.05 % employees earns between average and 75% salary of their companies department with class flag as 2 and 24.42 % employees earn more than 75 percentile of employees in their companies department.
- Top 10 Job Positions earning more than other job positions in their respective comapanies: Employees belong to company with company hash 'vbvkgz' who are backend engineers and other job positions earns around 42L as their median salary, so vbvkgz can be a tier 1 company.

Business Insights and Recommendations

- Based on the clustering analysis, we have identified several distinct groups of learners based on their profile characteristics and salary levels. This can help the business to better understand the diversity of their workforce and tailor their HR policies and compensation packages accordingly.
- We can see that there are certain companies and job positions that tend to pay higher salaries compared to others. This could be due to various factors such as the company's size, industry, location, and competitive landscape. By identifying these trends, the business can develop strategies to attract and retain top talent in these high-paying roles.
- The clustering results can also reveal patterns in terms of the learners' skills, experience, and education levels. By understanding these trends, the business can identify skill gaps and areas for training and development, as well as target their recruiting efforts more effectively.
- Scaler could focus on pushing the learners who have level 3 job positions to higher positions of level 2 or level 1 through referrals to get their learners for desired position with better salaries.
- Scaler can improvise their business relationship with companies that offer better salary package and work culture to their employees and encourage their learners to get into these companies by organising better placement campaigns.
- Learners with very high years of experience can be identified and encouraged to share their prefessional paths to the younger learners.
- Based on the clustering results, the business could consider revising their compensation packages to better align with the market rates and industry standards. This could involve conducting salary surveys, benchmarking against similar companies, and offering more competitive pay and benefits to attract and retain top talent.
- The business could also explore opportunities to upskill and reskill their workforce, based on the insights from the clustering analysis. This could involve offering training programs, mentoring, and other professional development opportunities to help learners acquire the skills and knowledge needed to succeed in their roles.
- In order to effectively align candidates with their future roles, Scaler need to places a strong emphasis on individuals with at least 1+ years of experience in their respective fields. By doing so, Scaler will be able to better match candidates with the right opportunities and pave the way for long-term success in the industry.
- One of the ways Scaler can use the cluster is to develop their courses for high-earning positions such as Engineers and Data Scientists, who are in high demand and command top salaries.
- Scaler can benefit from Tier 2 professionals who are already performing well and looking to advance in their careers. By offering shorter duration courses, Scaler can help them accelerate their career progression.
- Finally, the business could consider using the insights from the clustering analysis to improve their recruitment and retention strategies. By targeting high-potential learners with tailored messages and incentives, and by offering competitive salaries and growth opportunities, the business can create a more engaged and motivated workforce that is better equipped to drive business success.

By Mrudula A P
