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

```
/tmp/ipykernel_48614/1109543917.py:1: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

EDA

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

Out[ ]:

	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 [ ]:
```python
df.shape
```

Out[ ]: (205843, 7)

In [ ]:
```python
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      153279 non-null  object
 6   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(2), object(3)
memory usage: 11.0+ MB
```

In [ ]:
```python
df.drop(columns=["Unnamed: 0"], axis=1, inplace=True)
```

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

Out[ ]:

	orgyear	ctc	ctc_updated_year
count	205757.000000	2.058430e+05	205843.000000
mean	2014.882750	2.271685e+06	2019.628231
std	63.571115	1.180091e+07	1.325104
min	0.000000	2.000000e+00	2015.000000
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 [ ]:
```python
df['orgyear'] = df['orgyear'].astype("object")
df["ctc_updated_year"] = df["ctc_updated_year"].astype("object")
```

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

Out[ ]:

	ctc
count	2.058430e+05
mean	2.271685e+06
std	1.180091e+07
min	2.000000e+00
25%	5.300000e+05
50%	9.500000e+05
75%	1.700000e+06
max	1.000150e+09

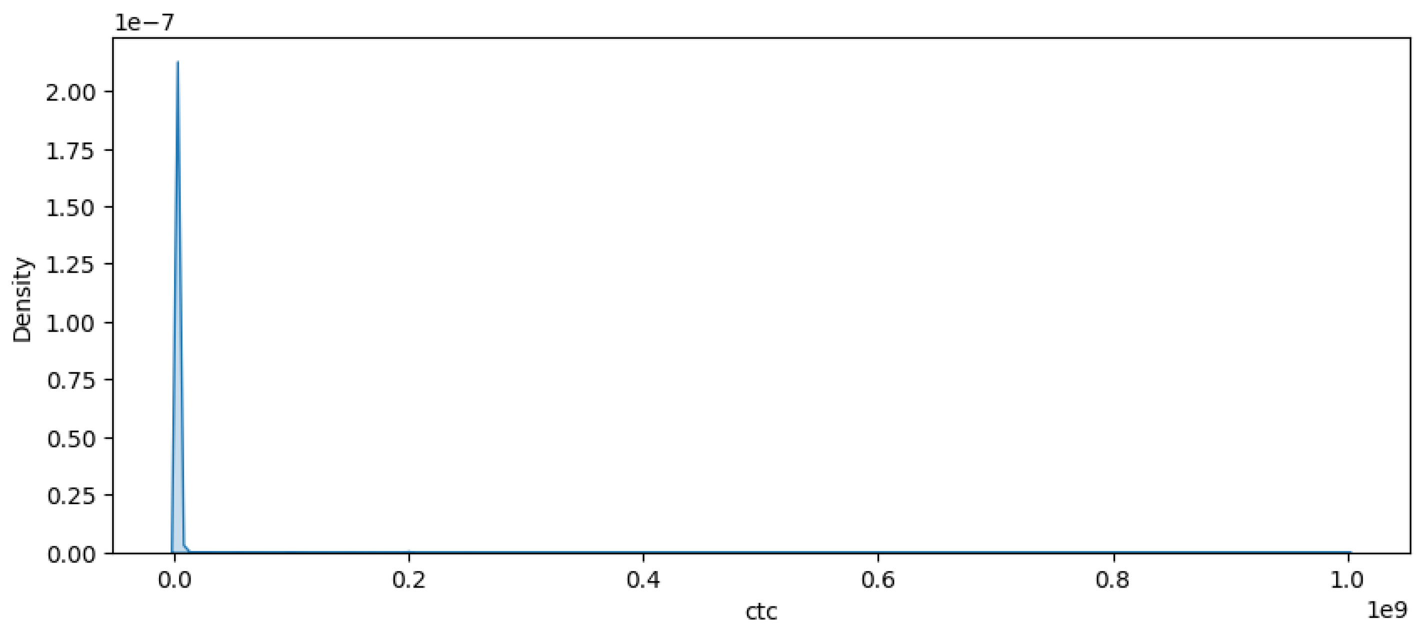
In [ ]:
```python
df.describe(include="object")
```

Out[ ]:

	company_hash	email_hash	orgyear	job_position	ctc_updated_year
count	205799	205843	205757.0	153279	205843.0
unique	37299	153443	77.0	1016	7.0
top	nvnv wgzohrnvzwj otqcxwto	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	Backend Engineer	2019.0
freq	8337	10	25256.0	43554	68688.0

In [ ]:
```python
fig=plt.figure(figsize=(10,4)) # width*height

# Create a KDE plot with filtered data
plt.subplot(1,1,1)
sns.kdeplot(df["ctc"], fill=True)
```

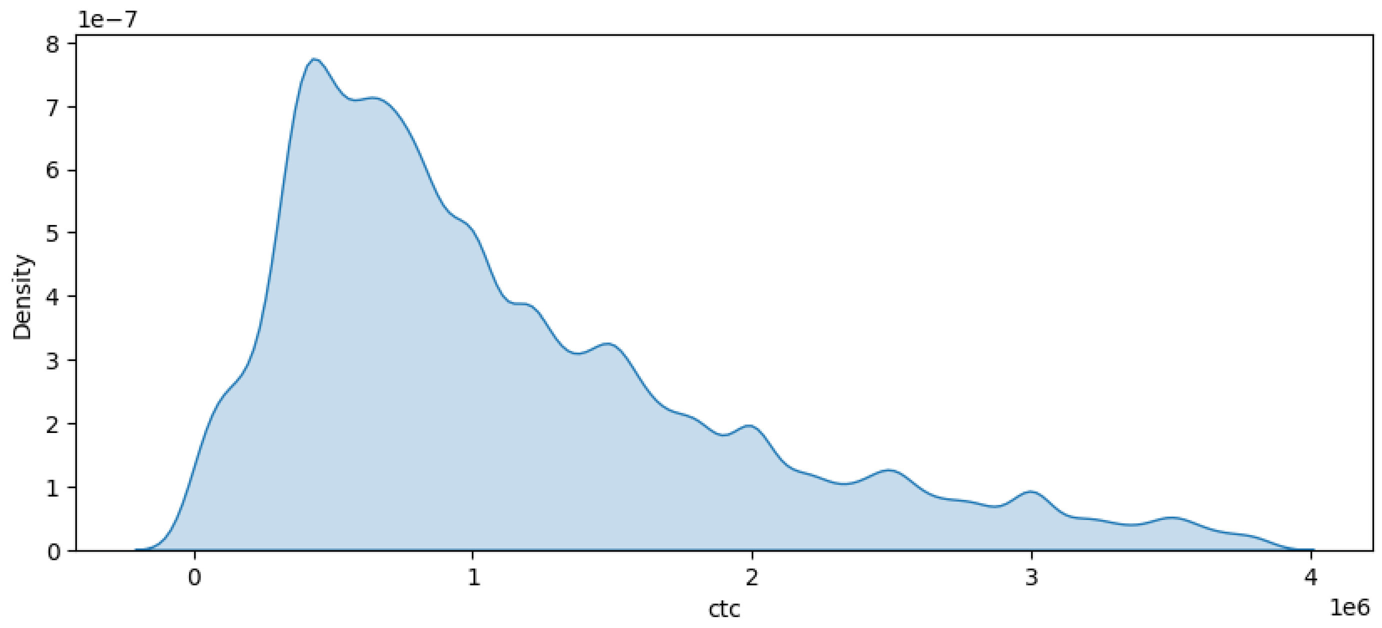
Out[ ]: <Axes: xlabel='ctc', ylabel='Density'>



- CTC seems to have outliers, we will be handling the same while working further.
- note: outliers doesn't mean that they are noise, in case of income we may have people with expemtional income.

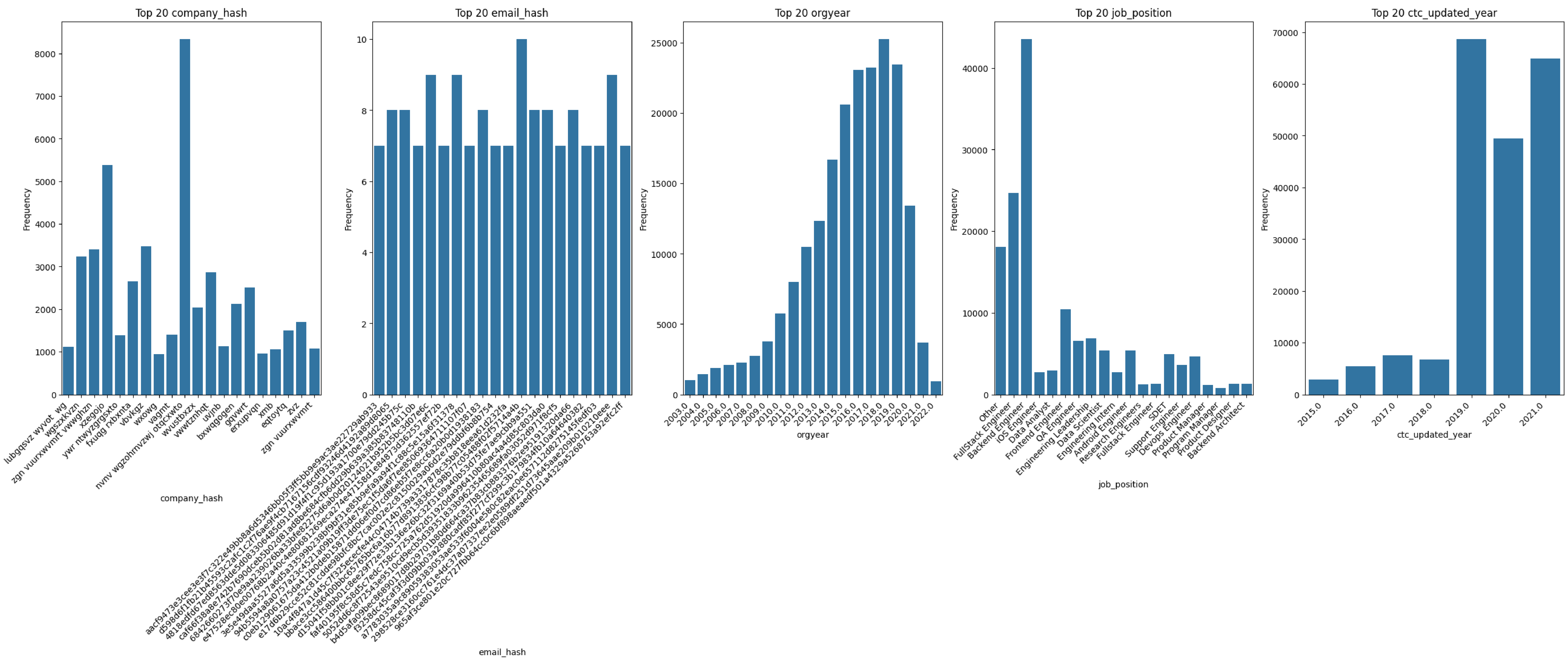
In [ ]:
```python
fig=plt.figure(figsize=(10,4)) # width*height

# Get the data
data = df['ctc']

# Set a range to exclude outliers (adjust as needed)
lower_limit = np.percentile(data, 0)
upper_limit = np.percentile(data, 95)

# Filter the data to exclude outliers
filtered_data = data[(data >= lower_limit) & (data <= upper_limit)]

# Create a KDE plot with filtered data
plt.subplot(1,1,1)
sns.kdeplot(filtered_data, fill=True)
```

Out[ ]: <Axes: xlabel='ctc', ylabel='Density'>



- From the CTC marked in Density graph it seems that most of the Employees enrolled with us seems to be in 5,00,000-6,00,000 job bracket.

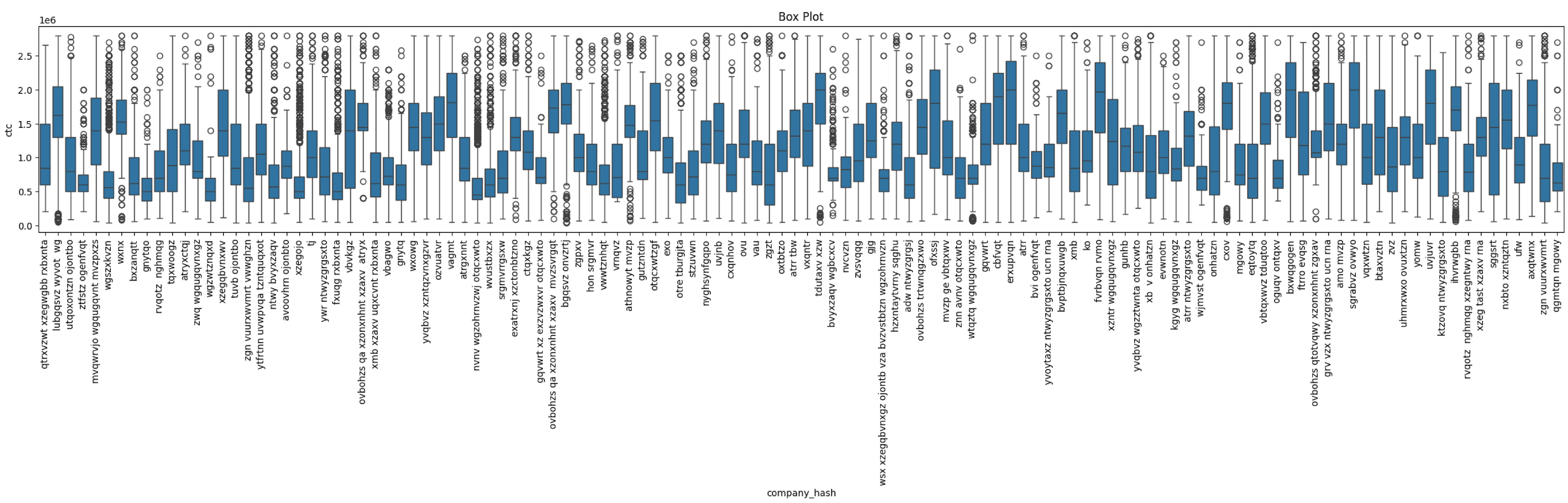
In [ ]:
```python
fig=plt.figure(figsize=(32,8)) # width*height

for ind_num,col_name in enumerate(df.describe(include='object')):

    # get top n categories
    top_n = 20
    top_categories = df[col_name].value_counts().nlargest(top_n).index

    # filter the df
    df_top_n = df[df[col_name].isin(top_categories)]

    # Create a countplot
    plt.subplot(1,5,ind_num+1)
    sns.countplot(x=col_name, data=df_top_n)
    plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better visibility

    # Set labels and title
    plt.xlabel(col_name)
    plt.ylabel('Frequency')
    plt.title(f'Top {top_n} {col_name}')
```



- Seems we have dubplicate entries for candidates in dataset having multiple years of experience.
- Last CTC update year was in 2021.

In [ ]:
```python
# Categorical Vs Numerical (CTC)

fig=plt.figure(figsize=(100,4))

# Set a range to exclude outliers (adjust as needed)
lower_limit = np.percentile(df['ctc'], 1)
upper_limit = np.percentile(df['ctc'], 90)

# Filter the data to exclude outliers
filtered_data = df[(df['ctc'] >= lower_limit) & (df['ctc'] <= upper_limit)]

# get top n categories
top_n = 120
top_categories = filtered_data["company_hash"].value_counts().nlargest(top_n).index

# filter the df
df_top_n = filtered_data[filtered_data["company_hash"].isin(top_categories)]

plt.subplot(1,3,1)
sns.boxplot(y='ctc', x='company_hash', data=df_top_n)

plt.xticks(rotation=90, ha='center')  # Rotate x-axis labels for better visibility

# Set labels and title
plt.xlabel("company_hash")
plt.ylabel('ctc')
plt.title(f'Box Plot')
```

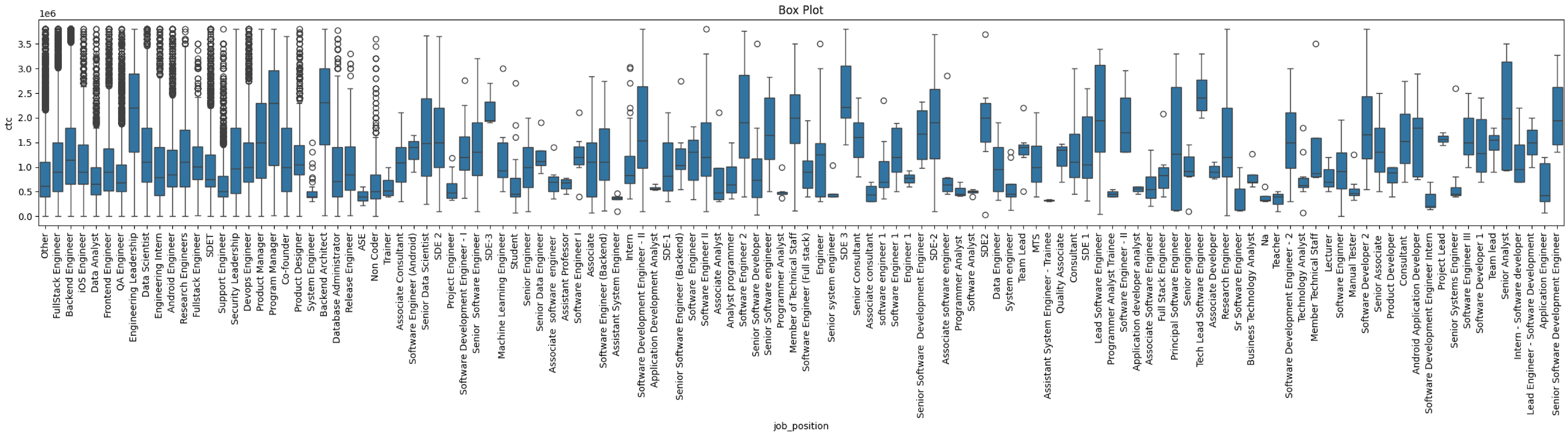
Out[ ]:
```
Text(0.5, 1.0, 'Box Plot')
```



In [ ]:
```python
fig=plt.figure(figsize=(100,4)) # width*height

# Set a range to exclude outliers (adjust as needed)
lower_limit = np.percentile(df['ctc'], 0)
upper_limit = np.percentile(df['ctc'], 95)

# Filter the data to exclude outliers
filtered_data = df[(df['ctc'] >= lower_limit) & (df['ctc'] <= upper_limit)]
```

```python
# get top n categories
top_n = 120
top_categories = filtered_data["job_position"].value_counts().nlargest(top_n).index

# filter the df
df_top_n = filtered_data[filtered_data["job_position"].isin(top_categories)]

plt.subplot(1,3,1)
sns.boxplot(y='ctc', x='job_position', data=df_top_n)

plt.xticks(rotation=90, ha='center')  # Rotate x-axis labels for better visibility

# Set labels and title
plt.xlabel("job_position")
plt.ylabel('ctc')
plt.title(f'Box Plot')
```

Out[ ]: Text(0.5, 1.0, 'Box Plot')

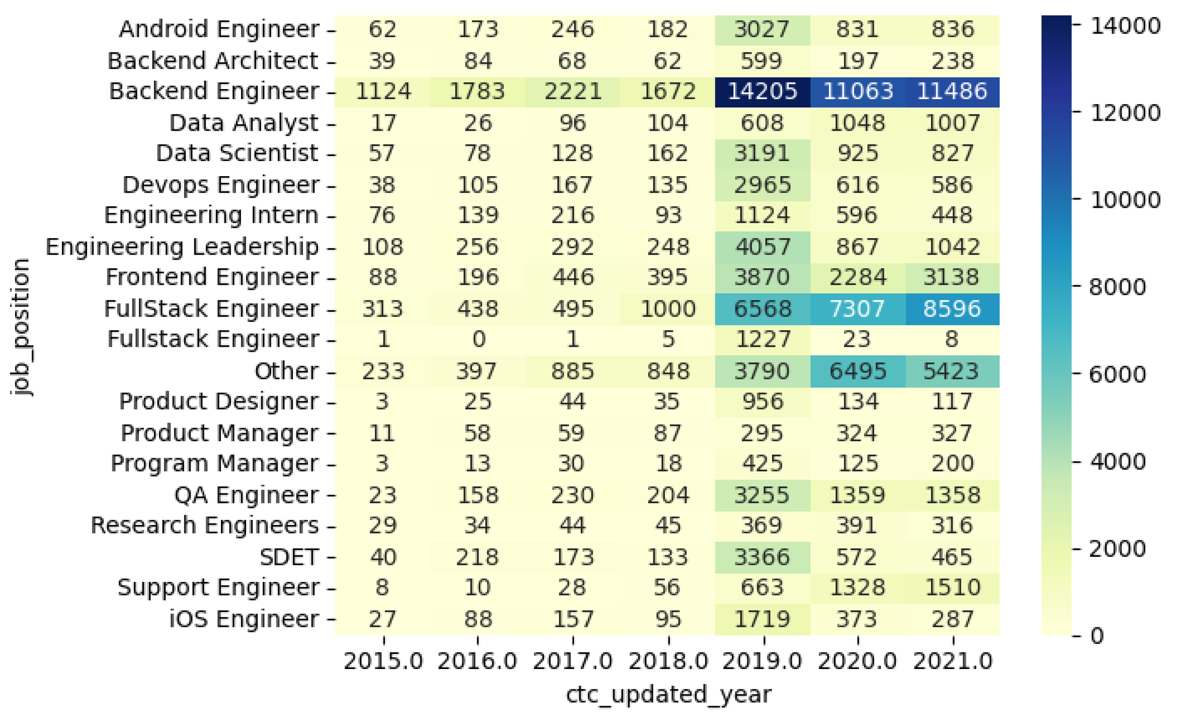


- Profiles seems to earn more irrespective of which company they work for includes:

1. SDE-3
2. Tech Lead Software Engineer

In [ ]:
```python
# Category Vs Category

# get top n categories
top_n = 20
top_categories = df["job_position"].value_counts().nlargest(top_n).index

# filter the df
df_top_n = df[df["job_position"].isin(top_categories)]

cross_tab = pd.crosstab(df_top_n["job_position"], df_top_n["ctc_updated_year"])
sns.heatmap(cross_tab,annot=True, cmap="YlGnBu", fmt='d', cbar=True)
```

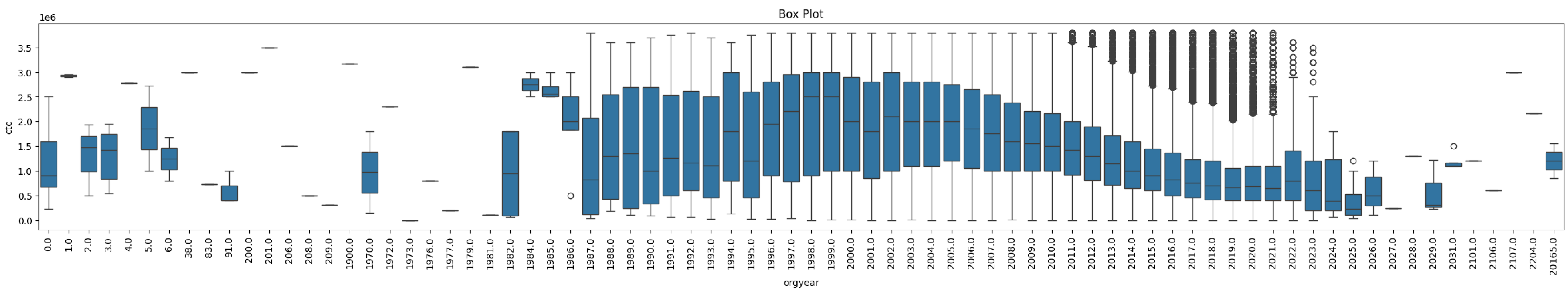
Out[ ]: <Axes: xlabel='ctc_updated_year', ylabel='job_position'>



- Full Stack Engineers and Backend Engineer have higher chances of getting Appraisal or Salary hike as compare to other Profiles

In [ ]:
```python
fig=plt.figure(figsize=(100,4)) # width*height

# Set a range to exclude outliers (adjust as needed)
lower_limit = np.percentile(df['ctc'], 0)
upper_limit = np.percentile(df['ctc'], 95)

# Filter the data to exclude outliers
filtered_data = df[(df['ctc'] >= lower_limit) & (df['ctc'] <= upper_limit)]

# get top n categories
top_n = 120
top_categories = filtered_data["orgyear"].value_counts().nlargest(top_n).index

# filter the df
df_top_n = filtered_data[filtered_data["orgyear"].isin(top_categories)]

plt.subplot(1,3,1)
sns.boxplot(y='ctc', x='orgyear', data=df_top_n)

plt.xticks(rotation=90, ha='center')  # Rotate x-axis labels for better visibility

# Set labels and title
plt.xlabel("orgyear")
plt.ylabel('ctc')
plt.title(f'Box Plot')
```

Out[ ]: Text(0.5, 1.0, 'Box Plot')



- From above graph we have a lot of noise data, we'll be filter that later at pre-processing stage

In [ ]:
```python
df['email_hash'].value_counts()[:10]
```

Out[ ]:
```
email_hash
bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b    10
3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94f1c88c5e15a6f31378     9
298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee     9
6842660273f70e9aa239026ba33bfe82275d6ab0d20124021b952b5bc3d07e6c     9
d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf93246d4192a89d8065     8
faf40195f8c58d5c7edc758cc725a762d51920da996410b80ac4a4d85c803da0     8
b4d5afa09bec8689017d8b29701b80d664ca37b83cb883376b2e95191320da66     8
d15041f58bb01c8ee29f72e33b136e26bc32f3169a40b53d75fe7ae9cbb9a551     8
4818edfd67ed8563dde5d083306485d91d19f4f1c95d193a1700e79dd245b75c     8
c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7e8cc6a20b0d1938183     8
Name: count, dtype: int64
```

In [ ]:
```python
df[df['email_hash']=="bbace3cc586400bbc65765bc6a16b77d8913836cfc98b77c05488f02f5714a4b"]
```

Out[ ]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
24109	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	NaN	2020.0
45984	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	Support Engineer	2020.0
72315	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	Other	2020.0
102915	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	FullStack Engineer	2020.0
117764	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	720000	Data Analyst	2020.0
121483	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	Other	2019.0
124476	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	Support Engineer	2019.0
144479	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	FullStack Engineer	2019.0
152801	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	Devops Engineer	2019.0
159835	oxej ntwyzgrgsxto rxbxnta	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	2018.0	660000	NaN	2019.0

- from above analysis we can check that candidate have multiple entries based on job_positions

In [ ]:

```python
df[df['email_hash']=="d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf93246d4192a89d8065"]
```

Out[ ]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
4401	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	300000	NaN	2020.0
11331	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	300000	Data Scientist	2020.0
22412	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	300000	Frontend Engineer	2020.0
81028	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	400000	Other	2021.0
90782	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	400000	Backend Engineer	2021.0
92949	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	400000	Data Scientist	2021.0
107425	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	400000	NaN	2021.0
132398	nvnv wgzohrnvzwj otqcxwto	d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...	2018.0	400000	Frontend Engineer	2021.0

In [ ]:

```python
df[df['email_hash']=="298528ce3160cc761e4dc37a07337ee2e0589df251d73645aae209b010210eee"]
```

Out[ ]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
65909	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	720000	Backend Engineer	2020.0
72799	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	720000	Research Engineers	2020.0
82099	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	720000	Other	2020.0
93495	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	720000	NaN	2020.0
93783	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	720000	Data Scientist	2020.0
190903	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	700000	Other	2020.0
191498	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	700000	Research Engineers	2020.0
196685	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	700000	Data Scientist	2020.0
201587	cvrhtbgbtznhb	298528ce3160cc761e4dc37a07337ee2e0589df251d736...	2018.0	700000	Backend Engineer	2020.0

In [ ]:

```python
df_cnt = df['email_hash'].value_counts().reset_index()
df_cnt[df_cnt["count"]==2]['email_hash'].tolist()[:5]
```

Out[ ]:

```
['d3e27dfa3240546390161c8f8e7be3ea5cd8f47dbf7152c9df81f48dc841742e',
 '2dc0dd508944f55ff448d30a5e660174896c0de2d150e68941cf750bc5f1aa4a',
 '702b13ba2005b5dbe91e0b7a9e8c09b3d71bb84e468379e79fbb71c7615cfaaa',
 'd7df6bd598c376ae391518a835780f0bfac770b010c664ff3ba9fde097077ec6',
 '59f5ea9240dc8e6ba6790a2697c3e9b1605320851b816d7dd2211c102c6f21a6']
```

In [ ]:

```python
df[df['email_hash']=="5559de74dd698c1ebc14d9653272d0c612970bc5ba206d6704394ccaab18c3cc"]
```

Out[ ]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
34281	qtamrvwpnqtt	5559de74dd698c1ebc14d9653272d0c612970bc5ba206d...	2019.0	600000	NaN	2021.0
184547	ztnbtaowgb	5559de74dd698c1ebc14d9653272d0c612970bc5ba206d...	2019.0	340000	FullStack Engineer	2020.0

In [ ]:

```python
df_cnt = df['email_hash'].value_counts().reset_index()
df_cnt[df_cnt["count"]==3]['email_hash'].tolist()[:5]
```

Out[ ]:

```
['7f9fd9949a7a90e322f3a1e72fb1eba7a1ae670778fe90d4a3a804ab87d7ae6a',
 'bf846924f9ebac3e7bd906316d7da0c0f16aab5b34d0d405c921917f4edd544c',
 '2538e1d0a89523f2a4a112e03b8d93e1b9f6a5d05a73fb2bc0cbe0bcb7ec395f',
 '8c7702516ea7c543e8f7758226a996c3bfbb6c889c432c27d1e9df6d34bcd8a8',
 '0d8e3ae92ca1e52184bb396d9ad92bbbbd6568bc579274ce9d2723ba380c8451']
```

In [ ]:

```python
df[df['email_hash']=="6576d1f1a561eb06bce501ab7ece60baaee8d529902f16f12496a18a4287ff64"]
```

Out[ ]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
83611	vbvkgz	6576d1f1a561eb06bce501ab7ece60baaee8d529902f16...	2015.0	499999	Other	2020.0
96777	vbvkgz	6576d1f1a561eb06bce501ab7ece60baaee8d529902f16...	2015.0	499999	Data Analyst	2020.0
162676	vbvkgz	6576d1f1a561eb06bce501ab7ece60baaee8d529902f16...	2015.0	499999	NaN	2020.0

In [ ]:

```python
df[df['email_hash']=="f6327cc669826a2b55dae0cbb69066a8e5e549320b73a6b0ce5cc9d5cc61c5ed"]
```

Out[ ]:

	company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
41961	vbtqxwvz tduqtoo	f6327cc669826a2b55dae0cbb69066a8e5e549320b73a6...	2014.0	2000000	NaN	2021.0
56796	vbtqxwvz tduqtoo	f6327cc669826a2b55dae0cbb69066a8e5e549320b73a6...	2014.0	2000000	Engineering Leadership	2021.0
67497	vbtqxwvz tduqtoo	f6327cc669826a2b55dae0cbb69066a8e5e549320b73a6...	2014.0	2000000	FullStack Engineer	2021.0

In [ ]:

```python
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  object
 3   ctc               205843 non-null  int64
 4   job_position      153279 non-null  object
 5   ctc_updated_year  205843 non-null  object
dtypes: int64(1), object(5)
memory usage: 9.4+ MB
```

In [ ]:

```python
missing_value = round((df.job_position.isnull().sum()/df.shape[0])*100 , 2)
missing_value
```

Out[ ]: 25.54

- 25% of the values in Job Description seems missing and upon analyzing the feature doesn't seems reliable. It is better to remove the feature while moving forward.

In [ ]:

```python
df.drop(columns=["job_position"], axis=1, inplace=True)
```

In [ ]:

```python
df.head()
```

Out[ ]:

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

In [ ]:

```python
# tbv_diff_pattern : based on box plot
Q1 = df["ctc"].quantile(0.01)
Q2 = df["ctc"].quantile(0.15)
Q3 = df["ctc"].quantile(0.85)
Q4 = df["ctc"].quantile(0.99)

print(df['ctc'].min(), Q1, Q2, Q3, Q4, df['ctc'].max())
```

```
2 37000.0 400000.0 2300000.0 12600000.0 1000150000
```

- I seems the spread for CTC ranges from 37,000 to 1,26,00,000 which is huge margin and we have to be specific while handling the specific CTC ranges.

In [ ]:

Data Preprocessing

In [ ]:

```python
df.head()
```

Out[ ]:

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

In [ ]:

```python
df.shape
```

Out[ ]: (205843, 5)

In [ ]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 5 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  object
 3   ctc               205843 non-null  int64
 4   ctc_updated_year  205843 non-null  object
dtypes: int64(1), object(4)
memory usage: 7.9+ MB
```

In [ ]:
```
df["orgyear"].value_counts()
```

Out[ ]:
```
orgyear
2018.0    25256
2019.0    23427
2017.0    23239
2016.0    23043
2015.0    20610
          ...
4.0           1
1900.0        1
1971.0        1
201.0         1
200.0         1
Name: count, Length: 77, dtype: int64
```

- Points to consider while filtering orgyear

1. We have total experience values it seems in this column as well.
2. Some noise as well with values like 201, 200 etc
3. May contain negative values also.

In [ ]:
```
df["orgyear"].max()
```

Out[ ]:
```
20165.0
```

In [ ]:
```
df["orgyear"].min()
```

Out[ ]:
```
0.0
```

In [ ]:
```
df["year_exp"] = df["orgyear"].apply(lambda x: (2024-x) if (2024-x)<50 else x)
df.head()
```

Out[ ]:

	company_hash	email_hash	orgyear	ctc	ctc_updated_year	year_exp
0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000	2020.0	8.0
1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999	2019.0	6.0
2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000000	2020.0	9.0
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700000	2019.0	7.0
4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400000	2019.0	7.0

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

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

In [ ]:
```
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  object
 3   ctc               205843 non-null  int64
 4   ctc_updated_year  205843 non-null  object
 5   year_exp          205757 non-null  float64
dtypes: float64(1), int64(1), object(4)
memory usage: 9.4+ MB
```

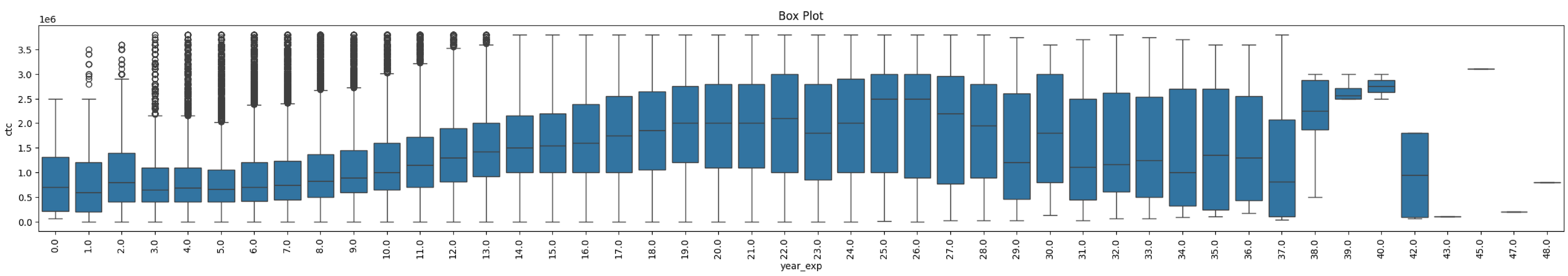
In [ ]:
```
df['year_exp'] = df['year_exp'].apply(lambda x: x if x<=50 and x>=0 else np.nan)
```

In [ ]:
```
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  object
 3   ctc               205843 non-null  int64
 4   ctc_updated_year  205843 non-null  object
 5   year_exp          205699 non-null  float64
dtypes: float64(1), int64(1), object(4)
memory usage: 9.4+ MB
```

In [ ]:
```
fig=plt.figure(figsize=(100,4)) # width*height

# Set a range to exclude outliers (adjust as needed)
lower_limit = np.percentile(df['ctc'], 0)
upper_limit = np.percentile(df['ctc'], 95)

# Filter the data to exclude outliers
filtered_data = df[(df['ctc'] >= lower_limit) & (df['ctc'] <= upper_limit)]

# get top n categories
top_n = 120
top_categories = filtered_data["year_exp"].value_counts().nlargest(top_n).index

# filter the df
df_top_n = filtered_data[filtered_data["year_exp"].isin(top_categories)]

plt.subplot(1,3,1)
sns.boxplot(y='ctc', x='year_exp', data=df_top_n)

plt.xticks(rotation=90, ha='center')  # Rotate x-axis labels for better visibility

# Set labels and title
plt.xlabel("year_exp")
plt.ylabel('ctc')
plt.title(f'Box Plot')
```

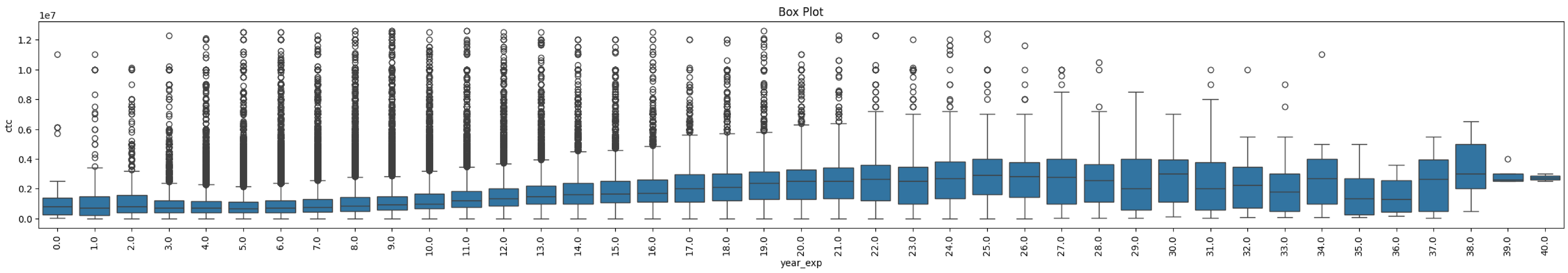
Out[ ]:
```
Text(0.5, 1.0, 'Box Plot')
```



- CTC seems to have noise after 40, so we'll be filtering the values till 40 years of exp.

In [ ]:
```
df['year_exp'] = df['year_exp'].apply(lambda x: x if x<=40 and x>=0 else np.nan)

fig=plt.figure(figsize=(100,4)) # width*height

# Set a range to exclude outliers (adjust as needed)
lower_limit = np.percentile(df['ctc'], 0)
upper_limit = np.percentile(df['ctc'], 99)

# Filter the data to exclude outliers
filtered_data = df[(df['ctc'] >= lower_limit) & (df['ctc'] <= upper_limit)]

# get top n categories
top_n = 120
top_categories = filtered_data["year_exp"].value_counts().nlargest(top_n).index

# filter the df
df_top_n = filtered_data[filtered_data["year_exp"].isin(top_categories)]

plt.subplot(1,3,1)
sns.boxplot(y='ctc', x='year_exp', data=df_top_n)

plt.xticks(rotation=90, ha='center')  # Rotate x-axis labels for better visibility

# Set labels and title
plt.xlabel("year_exp")
plt.ylabel('ctc')
plt.title(f'Box Plot')
```

Out[ ]:
```
Text(0.5, 1.0, 'Box Plot')
```



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

Out[ ]:

	company_hash	email_hash	orgyear	ctc	ctc_updated_year	year_exp
0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	1100000	2020.0	8.0
1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	449999	2019.0	6.0
2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	2000000	2020.0	9.0
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	700000	2019.0	7.0
4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	1400000	2019.0	7.0

- email_hash and orgyear doesn't seems to be of any use for us

In [ ]:
```python
df.drop(columns=["orgyear"], inplace=True, axis=1)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205843 entries, 0 to 205842
Data columns (total 5 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   company_hash      205799 non-null  object 
 1   email_hash        205843 non-null  object 
 2   ctc               205843 non-null  int64  
 3   ctc_updated_year  205843 non-null  object 
 4   year_exp          205691 non-null  float64
dtypes: float64(1), int64(1), object(3)
memory usage: 7.9+ MB
```

In [ ]:
```python
missing_value = round((df.year_exp.isnull().sum()/df.shape[0])*100 , 2)
print(f"year_exp missing value percentage: {missing_value}")

missing_value = round((df.company_hash.isnull().sum()/df.shape[0])*100 , 2)
print(f"company_hash missing value percentage: {missing_value}")
```

```
year_exp missing value percentage: 0.07
company_hash missing value percentage: 0.02
```

- Data seems missing let's merge and see the changes

In [ ]:
```python
df.drop_duplicates(inplace=True)
```

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

Out[ ]: (166861, 5)

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

Out[ ]:

	company_hash	email_hash	ctc	ctc_updated_year	year_exp
0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	1100000	2020.0	8.0
1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	449999	2019.0	6.0
2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2000000	2020.0	9.0
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	700000	2019.0	7.0
4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	1400000	2019.0	7.0

In [ ]:
```python
df.email_hash.value_counts()
```

Out[ ]:
```
email_hash
c986600ef19093ce70837408516acac9570566a4b29b554cfb6b744ffbe697d6    3
db84980ad197f8eff08b14a3442ff57f6374ea780f2587b310aac54b6c32ee3a    3
e5960ec01a207bfa6b83d5576f3d66f98b95f2e200f250303027c95395e4bad9    2
607910ba90b77a948e9255d472b9281ce90e10bc2a3089fede923c0eb421d039    2
1ea4e620e2d1f02c6c546c1d263582badacdb4b783d4cb8660901d3940f7ac3b    2
                                                                   ..
611b800e5d0f4caae6ade0c0392d27590987e77237dc430a2dd7c95d8bb82fa4    1
14d7bfecc80ded3532b8cf37e5b20989448b5d3fe387862b0f295d3117056a5e    1
2fbd7838b0002973720ba4bf6775f07b63cf88373571273c350aaf4652f835fa    1
8fffb1aea12ae1dcf3ee1cfe2934117af26df3e989dae5b0580aaed826a46c8d    1
badb0e8acad3be3bddfee92367413ec947bbb1029826ffc3becaea02593a10f8    1
Name: count, Length: 153443, dtype: int64
```

In [ ]:
```python
df[df["email_hash"]=="c986600ef19093ce70837408516acac9570566a4b29b554cfb6b744ffbe697d6"]
```

Out[ ]:

	company_hash	email_hash	ctc	ctc_updated_year	year_exp
8695	oyxc ozvd uqxcvnt rxbxnta	c986600ef19093ce70837408516acac9570566a4b29b55...	1800000	2021.0	11.0
61154	uvqrt	c986600ef19093ce70837408516acac9570566a4b29b55...	3500000	2021.0	15.0
98347	oyxc ozvd uqxcvnt rxbxnta	c986600ef19093ce70837408516acac9570566a4b29b55...	1800000	2019.0	11.0

In [ ]:
```python
# Find the index of the rows with the maximum 'Year' within each group
idx = df.groupby(['email_hash','company_hash'])['ctc_updated_year'].idxmax()

# Use the index to select the corresponding rows from the original DataFrame
df = df.loc[idx]

# Reset index for the final DataFrame
df.reset_index(drop=True, inplace=True)
```

In [ ]:
```python
df[df["email_hash"]=="c986600ef19093ce70837408516acac9570566a4b29b554cfb6b744ffbe697d6"]
```

Out[ ]:

	company_hash	email_hash	ctc	ctc_updated_year	year_exp
126055	oyxc ozvd uqxcvnt rxbxnta	c986600ef19093ce70837408516acac9570566a4b29b55...	1800000	2021.0	11.0
126056	uvqrt	c986600ef19093ce70837408516acac9570566a4b29b55...	3500000	2021.0	15.0

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

Out[ ]: (160273, 5)

In [ ]:
```python
df.email_hash.value_counts()
```

Out[ ]:
```
email_hash
db84980ad197f8eff08b14a3442ff57f6374ea780f2587b310aac54b6c32ee3a    3
e49d643e06c85681ee3b6feff020134f63c92b17e637283d44854412eb2c95fb    2
9c20e7a5e0b46a4350327978c130282184c2b39772405c5f5c9ab7e1e03b69e8    2
c88c616fb855fd35009d643fc4d9d91b4d53d363057fdc7fd071717c811296e6    2
e496afa17c48da3980dbd8330caf89ec01bd4d4849d17ad87a2b98abb699d172    2
                                                                   ..
57972fe710544a37de4ed56a1ac6f242d25d3f91cac5a4b787b840bea2c6fe30    1
57978d4cf69f3aa592a7a9a6eb5aecd0f5aa25005937befb846d16a6f6d4ae66    1
57983cc12ab513f649544370ad98a61a83e64a228a8d8b1a77c73cfc51b0b2e7    1
5798a70eb4780ddb0087a3c11ff7f6d7e23731ebf92dddeb3b7ca0b4f5c6df2d    1
5794bec6e3ee46ffdb464cd55aa9b34a1e5bc3fc05e52cd58570861bed8b8a88    1
Name: count, Length: 153411, dtype: int64
```

In [ ]:
```python
df[df["email_hash"]=="db84980ad197f8eff08b14a3442ff57f6374ea780f2587b310aac54b6c32ee3a"]
```

Out[ ]:

	company_hash	email_hash	ctc	ctc_updated_year	year_exp
137367	vqwotqct	db84980ad197f8eff08b14a3442ff57f6374ea780f2587...	700000	2021.0	4.0
137368	vqwotqct xzaxv ogrhnxgzo rxbxnta	db84980ad197f8eff08b14a3442ff57f6374ea780f2587...	700000	2021.0	4.0
137369	zgn vuurxwvmrt	db84980ad197f8eff08b14a3442ff57f6374ea780f2587...	400000	2019.0	4.0

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160273 entries, 0 to 160272
Data columns (total 5 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   company_hash      160273 non-null  object 
 1   email_hash        160273 non-null  object 
 2   ctc               160273 non-null  int64  
 3   ctc_updated_year  160273 non-null  object 
 4   year_exp          160134 non-null  float64
dtypes: float64(1), int64(1), object(3)
memory usage: 6.1+ MB
```

In [ ]:
```python
missing_value = round((df.year_exp.isnull().sum()/df.shape[0])*100 , 2)
print(f"year_exp missing value percentage: {missing_value}")
```

```
year_exp missing value percentage: 0.09
```

- only .09% percent values seems to be missing we can drop the specific rows / perform imputing on that dataset.

In [ ]:
```
## Missing value treatment

from sklearn.impute import KNNImputer

numeric_columns = ["ctc", "year_exp"]

# Extract the numeric part of the DataFrame
df_numeric = df[numeric_columns]

# Impute missing values in the numeric part using KNNImputer
knn_imputer = KNNImputer(n_neighbors=20)
df_numeric_imputed = pd.DataFrame(knn_imputer.fit_transform(df_numeric), columns=numeric_columns)

# Combine the imputed numeric part with the original categorical part
df = pd.concat([df.drop(columns=numeric_columns), df_numeric_imputed], axis=1)
df.isnull().sum()
```

Out[ ]:
```
company_hash        0
email_hash          0
ctc_updated_year    0
ctc                 0
year_exp            0
dtype: int64
```

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

Out[ ]:
```
(160273, 5)
```

In [ ]:
```
df.drop(columns=["email_hash"], axis=1, inplace=True)
```

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

Out[ ]:

	company_hash	ctc_updated_year	ctc	year_exp
0	bxwqgogen	2019.0	3500000.0	12.0
1	nqsn axsxnvr	2020.0	250000.0	11.0
2	gunhb	2019.0	1300000.0	3.0
3	bxwqgotbx wgqugqvnxgz	2021.0	2000000.0	20.0
4	fvrbvqn rvmo	2018.0	3400000.0	15.0

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160273 entries, 0 to 160272
Data columns (total 4 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   company_hash      160273 non-null  object
 1   ctc_updated_year  160273 non-null  object
 2   ctc               160273 non-null  float64
 3   year_exp          160273 non-null  float64
dtypes: float64(2), object(2)
memory usage: 4.9+ MB
```

In [ ]:
```
## Outlier Treatment
'''
Based on our previous understanding we will be capping the values at 1% i.e. below 37,000 and above 1,26,00,000
'''

# Set upper and lower thresholds for capping (adjust as needed)
upper_threshold = 12600000
lower_threshold = 37000

# Apply capping to remove outliers
df['ctc'].clip(lower=lower_threshold, upper=upper_threshold, inplace=True)

df.shape
```

```
/tmp/ipykernel_48614/1913124230.py:11: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['ctc'].clip(lower=lower_threshold, upper=upper_threshold, inplace=True)
```

Out[ ]:
```
(160273, 4)
```

- Doesn't seems there exist any outlier after we did merging and filter the noise.

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

Out[ ]:

	company_hash	ctc_updated_year	ctc	year_exp
0	bxwqgogen	2019.0	3500000.0	12.0
1	nqsn axsxnvr	2020.0	250000.0	11.0
2	gunhb	2019.0	1300000.0	3.0
3	bxwqgotbx wgqugqvnxgz	2021.0	2000000.0	20.0
4	fvrbvqn rvmo	2018.0	3400000.0	15.0

In [ ]:
```
## Feature Engineering
'''
1. Calculated Years of experience using 'orgyear' features.
2. Based on Job Profile the data seems fine for clustering and Job Description seems not reliable as per understanding.
3. CTC_updated year: we will be considering the feature as categorical value.
4. Implementation of regex for the usecase doen't seems to be a good idea here.
'''

# Define bin edges and labels
bins = [0, 500000, 1500000, 3000000, float('inf')]  # 'inf' represents infinity, covering values greater than 120
bin_labels = ['Low', 'Medium', 'High', 'Very High']

# Create a new column 'Bins' and assign the bin labels
df['salary_bin'] = pd.cut(df['ctc'], bins=bins, labels=bin_labels, right=False)

# Define bin edges and labels
bins = [0, 5, 10, 50]  # 'inf' represents infinity, covering values greater than 120
bin_labels = ['Low', 'Medium', 'High']

# Create a new column 'Bins' and assign the bin labels
df['exp_bin'] = pd.cut(df['year_exp'], bins=bins, labels=bin_labels, right=False)
```

In [ ]:
```
# Define conditions and corresponding labels
conditions = [
    (df['ctc_updated_year'] == 2021),
    (df['ctc_updated_year'] == 2020),
    ((df['ctc_updated_year'] != 2020) & (df['ctc_updated_year'] != 2021))
]
labels = ['2021', '2020', 'other']
df['ctc_updated'] = np.select(conditions, labels, default='Other')
```

In [ ]:
```
df['ctc_updated'].value_counts()
```

Out[ ]:
```
ctc_updated
other    78368
2021     42742
2020     39163
Name: count, dtype: int64
```

In [ ]:
```
df.drop(columns=["ctc",'year_exp','ctc_updated_year'], axis=1, inplace=True)
```

In [ ]:
```
df.drop_duplicates(inplace=True)
```

In [ ]:
```
df.describe(include="all")
```

Out[ ]:

	company_hash	salary_bin	exp_bin	ctc_updated
count	66898	66898	66898	66898
unique	37299	4	3	3
top	bxwqgogen	Medium	Medium	other
freq	36	30519	33570	32612

In [ ]:
```
tmpdf = df.company_hash.value_counts().reset_index()
```

In [ ]:
```
for i in range(1,36):
    print(i, tmpdf[tmpdf["count"]==i].shape[0])
```

```
1 29137
2 3707
3 1418
4 748
5 487
6 346
7 271
8 172
9 129
10 135
11 103
12 99
13 70
14 70
15 37
16 45
17 39
18 41
19 38
20 37
21 26
22 20
23 17
24 13
25 10
26 10
27 15
28 10
29 10
30 5
31 7
32 6
33 7
34 3
35 5
```

In [ ]:
```
# Count the occurrences of each category
category_counts = df['company_hash'].value_counts()

# Identify categories with count 1
single_occurrence_categories = category_counts[category_counts == 1].index

# Replace single-occurrence categories with 'Other'
df['company_hash'] = df['company_hash'].replace(single_occurrence_categories, 'Other')
```

In [ ]:
```
df.drop_duplicates(inplace=True)
```

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

Out[ ]:
```
(37797, 4)
```

In [ ]:
```
df.company_hash.value_counts()
```

Out[ ]:
```
company_hash
xzegojo                            36
bxwqgogen                          36
xmb                                36
Other                              36
vbvkgz                             36
                                   ..
hzxcvqxtnj                          2
btnnrtqngrtag xzntqzvnxgzvr xzw     2
hubw tzntquqxoto                    2
wvubvnqxd ntwyzgrgsj                2
x3 wgzohrnxzs                       2
Name: count, Length: 8163, dtype: int64
```

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

Out[ ]:

	company_hash	salary_bin	exp_bin	ctc_updated
count	37797	37797	37797	37797
unique	8163	4	3	3
top	xzegojo	Medium	Medium	other
freq	36	16522	18909	16806

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

Out[ ]:
```
Index(['company_hash', 'salary_bin', 'exp_bin', 'ctc_updated'], dtype='object')
```

In [ ]:
```
## Data preparation and Scaling

# OHE
df = pd.get_dummies(df,columns=['salary_bin', 'exp_bin', 'ctc_updated'], dtype=int)
```

In [ ]:
```
## Target Encoding
import category_encoders as ce
target_encoder = ce.TargetEncoder(cols=['company_hash']) # Create a TargetEncoder instance
df = target_encoder.fit_transform(df, df.index)  # Fit and transform the DataFrame with target encoding, using index as a dummy target

# Standard Scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Reshape the data to fit the scaler (required for a single column)
df["company_hash"] = scaler.fit_transform(df["company_hash"].values.reshape(-1, 1))
```

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

Out[ ]:

	company_hash	salary_bin_Low	salary_bin_Medium	salary_bin_High	salary_bin_Very High	exp_bin_Low	exp_bin_Medium	exp_bin_High	ctc_updated_2020	ctc_updated_2021	ctc_updated_other
0	0.166218	0	0	0	1	0	0	1	0	0	1
1	0.000000	1	0	0	0	0	0	1	1	0	0
2	0.491054	0	1	0	0	1	0	0	0	0	1
3	0.788863	0	0	1	0	0	0	1	0	1	0
4	0.351813	0	0	0	1	0	0	1	0	0	1

In [ ]:
```
df.to_csv("dataset/phase2_df.csv", index=False)
```

In [ ]:

In [ ]: