In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df_clus = pd.read_csv('/Users/Ramv/Downloads/scaler_clustering.csv')
df_clus
```

Out[2]:

	Unnamed: 0	company_hash	email_hash	o
0	0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	
1	1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	
2	2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	
3	3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	
4	4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	
...	...	...	...	
205838	206918	vuurt xzw	70027b728c8ee901fe979533ed94ffda97be08fc23f33b...	
205839	206919	husqvawgb	7f7292ffad724ebbe9ca860f515245368d714c84705b42...	
205840	206920	vwwgrxnt	cb25cc7304e9a24facda7f5567c7922ffc48e3d5d6018c...	
205841	206921	zgn vuurxwvmrt	fb46a1a2752f5f652ce634f6178d0578ef6995ee59f6c8...	
205842	206922	bgqsvz onvzrtj	0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f...	

205843 rows × 7 columns

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

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

In [4]:
```
df_clus.isnull().sum()
```

Out[4]:
```
Unnamed: 0              0
company_hash           44
email_hash              0
orgyear                86
ctc                     0
job_position        52562
ctc_updated_year        0
dtype: int64
```

Dropping Unnamed Column

In [5]:
```
df_clus.drop(columns = 'Unnamed: 0', inplace= True)
```

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

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

In [7]:
```
df_clus.nunique()
```

Out[7]:
```
company_hash         37299
email_hash          153443
orgyear                 77
ctc                   3360
job_position          1017
ctc_updated_year         7
dtype: int64
```

dropping duplicate records

In [8]:
```
df_clus.duplicated().sum()
```

Out[8]:
```
33
```

In [9]:
```
df_clus.drop_duplicates(inplace=True)
```

In [10]:
```
df_clus
```

Out[10]:

	company_hash	email_hash	orgyear	
0	atrgxnnt xzaxv	6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...	2016.0	110
1	qtrxvzwt xzegwgbb rxbxnta	b0aaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...	2018.0	44
2	ojzwnvwnxw vx	4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...	2015.0	200
3	ngpgutaxv	effdede7a2e7c2af664c8a31d9346385016128d66bbc58...	2017.0	70
4	qxen sqghu	6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...	2017.0	140
...	...	...	...	
205838	vuurt xzw	70027b728c8ee901fe979533ed94ffda97be08fc23f33b...	2008.0	22
205839	husqvawgb	7f7292ffad724ebbe9ca860f515245368d714c84705b42...	2017.0	50
205840	vwwgrxnt	cb25cc7304e9a24facda7f5567c7922ffc48e3d5d6018c...	2021.0	70
205841	zgn vuurxwvmrt	fb46a1a2752f5f652ce634f6178d0578ef6995ee59f6c8...	2019.0	510
205842	bgqsvz onvzrtj	0bcfc1d05f2e8dc4147743a1313aa70a119b41b30d4a1f...	2014.0	124

205810 rows × 6 columns

Descriptive Stats

Columns with Continuous variables

In [11]:
```
display(df_clus.describe().T.round(2))
print()
```

	count	mean	std	min	25%	50%	75%
orgyear	205724.0	2014.88	63.58	0.0	2013.0	2016.0	2018.(
ctc	205810.0	2271853.65	11801845.29	2.0	530000.0	950000.0	1700000.(
ctc_updated_year	205810.0	2019.63	1.33	2015.0	2019.0	2020.0	2021.(

Columns with Categorical variables

In [12]:
```
display(df_clus.describe(include = 'object').T)
print()
```

	count	unique	top	fre
company_hash	205766	37299	nvnv wgzohrnvzwj otqcxwto	833
email_hash	205810	153443	bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...	1
job_position	153263	1017	Backend Engineer	4354

"Backend Engineer" is the most common job_position and it looks like CTC has outliers,lets check

In [13]:
```
q1 = np.percentile(df_clus['ctc'], 25)
q3 = np.percentile(df_clus['ctc'], 75)
IQR = q3-q1

UB = q3 + 1.5*IQR
LB = q1 - 1.5*IQR
total_outliers = len(df_clus[(df_clus['ctc'] > UB) | (df_clus['ctc'] < LB)])

n = len(df_clus)
tot_outliers_percent = 100 * total_outliers/n
print(f"Outlier % of CTC: {tot_outliers_percent:.2f}%")
```

```
Outlier % of CTC: 6.38%
```

Data Cleaning

In [14]:
```
# Analysing orgyear feature
print('5 point summary of orgyear',end = ': ')
print(df_clus['ctc_updated_year'].describe())

# Cleaning orgyear feature
df_clus['orgyear'] = df_clus.apply(lambda x: x['ctc_updated_year'] if x['org
                                                            x['orgyear'] < 1
print('\n5 point summary of orgyear after cleaning',end = ': ')
print(df_clus['orgyear'].describe())

# Anlayzing NaN's in company hash
print('\nNaN in job_position',end = ': ')
print(df_clus['job_position'].isna().sum())

# Imputing NaN's in orgyear
orgyear_impute = df_clus.groupby('email_hash')['orgyear'].min()
df_clus.loc[df_clus['orgyear'].isna(),'orgyear'] = df_clus[df_clus['orgyear'
df_clus.loc[df_clus['orgyear'].isna(),'orgyear'] = df_clus[df_clus['orgyear'

# Total NaN's
print("Total NaN's after imputation:",df_clus['orgyear'].isna().sum())
```

```
5 point summary of orgyear: count    205810.000000
mean       2019.628279
std           1.325188
min        2015.000000
25%        2019.000000
50%        2020.000000
75%        2021.000000
max        2021.000000
Name: ctc_updated_year, dtype: float64

5 point summary of orgyear after cleaning: count    205724.000000
mean       2015.107980
std           4.219258
min        1970.000000
25%        2013.000000
50%        2016.000000
75%        2018.000000
max        2021.000000
Name: orgyear, dtype: float64

NaN in job_position: 52547
Total NaN's after imputation: 0
```

In [15]:
```
# Anlayzing NaN's in company hash
print('NaN in company_hash',end = ': ')
print(df_clus['company_hash'].isna().sum())

# Imputing NaN's in company_hash
company_impute = df_clus.groupby('email_hash')['company_hash'].first()
df_clus.loc[df_clus['company_hash'].isna(),'company_hash'] = df_clus[df_clus
                                                                        comp

# Dropping remaining, because these could be learners who are currently unem
df_clus =df_clus.dropna(subset=['company_hash'])

# Total NaN's
print("Total NaN's after imputation:",df_clus['company_hash'].isna().sum())
```

```
NaN in company_hash: 44
Total NaN's after imputation: 0
```

In [16]:
```
# Anlayzing NaN's in job_position
print('NaN in job_position',end = ': ')
print(df_clus['job_position'].isna().sum())

# Imputing NaN's in job_position

# Imputed by previous reported posiition by learner
job_impute = df_clus.groupby('email_hash')['job_position'].first()
df_clus.loc[df_clus['job_position'].isna(), 'job_position'] = df_clus[df_clu
                                job_impute[x['email_hash']], axis = 1)

# Renaming rest as Unidentified, Reason: It does not effect formation of nat
df_clus.loc[df_clus['job_position'].isna(), 'job_position'] = 'Unidentified'

# Total NaN's
print("Total NaN's after imputation:",df_clus['job_position'].isna().sum())
```

```
NaN in job_position: 52522
Total NaN's after imputation: 0
```

Checking Nan & Duplicates after Data Cleaning

In [17]:
```
# Total NaN's  in dataset
display(df_clus.isna().sum())

# Total Duplicates found
display(df_clus.duplicated().sum())
```

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

```
26828
```

Converging data to email_hash level

In [18]:
```python
# Converging data to email_level
df_clus_agg = df_clus.groupby("email_hash", as_index=False).agg({
    'company_hash': "last",
    'orgyear':'last',
    'ctc':'last',
    'job_position': 'last',
    'ctc_updated_year': 'last',
})
```

In [19]:
```python
df_clus_agg
```

Out[19]:

	email_hash	company_hash	orgyear	
0	00003288036a44374976948c327f246fdbdf0778546904...	bxwqgogen	2012.0	35
1	0000aaa0e6b61f7636af1954b43d294484cd151c9b3cf6...	nqsn axsxnvr	2013.0	2
2	0000d58fbc18012bf6fa2605a7b0357d126ee69bc41032...	gunhb	2021.0	13
3	000120d0c8aa304fcf12ab4b85e21feb80a342cfea03d4...	bxwqgotbx wgqugqvnxgz	2004.0	20
4	00014d71a389170e668ba96ae8e1f9d991591acc899025...	fvrbvqn rvmo	2009.0	34
...	...	...	...	
153406	fffc254e627e4bd1bc0ed7f01f9aebbba7c3cc56ac914e...	tqxwoogz ogenfvqt wvbuho	2004.0	35
153407	fffcf97db1e9c13898f4eb4cd1c2fe862358480e104535...	trnqvcg	2015.0	16
153408	fffe7552892f8ca5fb8647d49ca805b72ea0e9538b6b01...	znn avnv srgmvr atrxctqj otqcxwto	2014.0	9
153409	ffff49f963e4493d8bbc7cc15365423d84a767259f7200...	zwq wgqugqvnxgz	2020.0	7
153410	ffffa3eb3575f43b86d986911463dce7bcadcea227e5a4...	sgrabvz ovwyo	2018.0	15

153411 rows × 6 columns

Feature Engineering

In [20]:
```
# Creating Feature Years of experience
df_clus_agg['years_of_exp'] = abs(df_clus_agg['ctc_updated_year'] - df_clus_
```

In [21]:
```
# Creating a feature that identifies job as senior or not
df_clus_agg['senior_position'] = np.where( (df_clus_agg['job_position'].str.
                                (df_clus_agg['job_position'].str.lower().str.co
                                (df_clus_agg['job_position'].str.lower().str.co
```

In [22]:
```
df_clus_agg
```

Out[22]:

	email_hash	company_hash	orgyear	
0	00003288036a44374976948c327f246fdbdf0778546904...	bxwqgogen	2012.0	350
1	0000aaa0e6b61f7636af1954b43d294484cd151c9b3cf6...	nqsn axsxnvr	2013.0	25
2	0000d58fbc18012bf6fa2605a7b0357d126ee69bc41032...	gunhb	2021.0	130
3	000120d0c8aa304fcf12ab4b85e21feb80a342cfea03d4...	bxwqgotbx wgqugqvnxgz	2004.0	200
4	00014d71a389170e668ba96ae8e1f9d991591acc899025...	fvrbvqn rvmo	2009.0	340
...	...	...	...	
153406	fffc254e627e4bd1bc0ed7f01f9aebbba7c3cc56ac914e...	tqxwoogz ogenfvqt wvbuho	2004.0	352
153407	fffcf97db1e9c13898f4eb4cd1c2fe862358480e104535...	trnqvcg	2015.0	160
153408	fffe7552892f8ca5fb8647d49ca805b72ea0e9538b6b01...	znn avnv srgmvr atrxctqj otqcxwto	2014.0	90
153409	ffff49f963e4493d8bbc7cc15365423d84a767259f7200...	zwq wgqugqvnxgz	2020.0	70
153410	ffffa3eb3575f43b86d986911463dce7bcadcea227e5a4...	sgrabvz ovwyo	2018.0	150

153411 rows × 8 columns

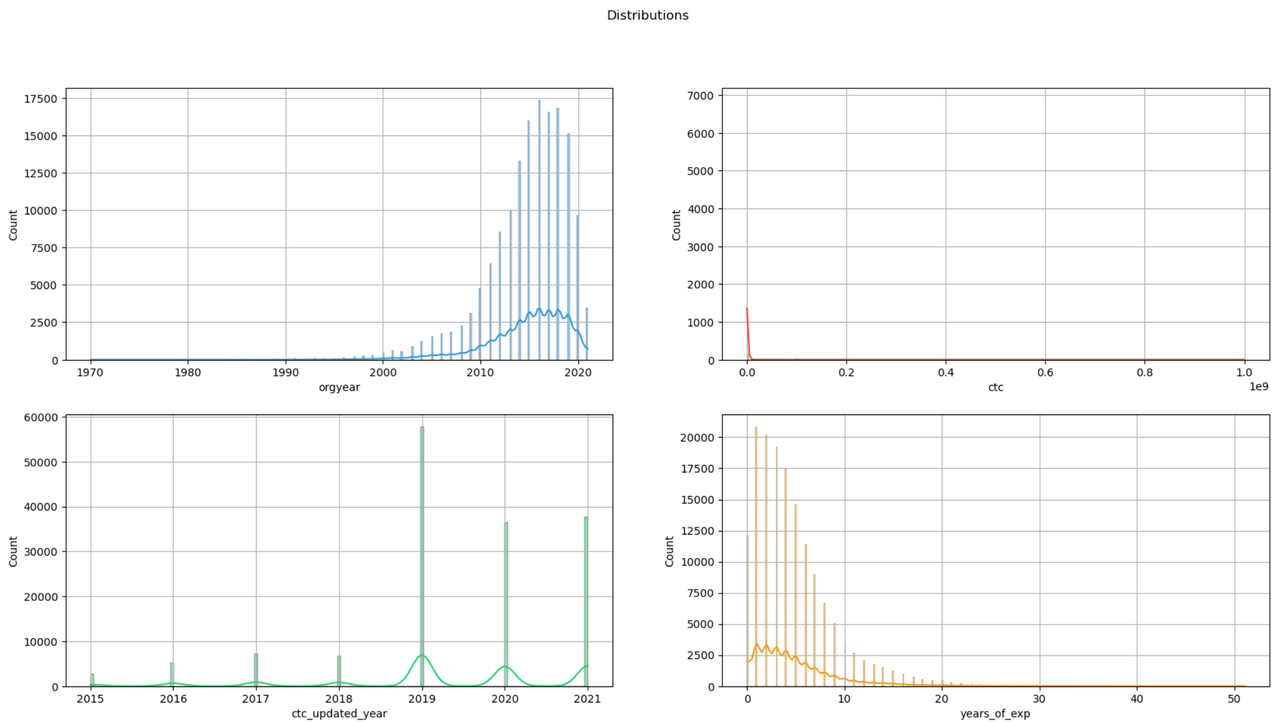
Data Visualization

In [23]:
```
# Assigning data types to variables for quick acess
cont_var = df_clus_agg.columns[df_clus_agg.dtypes!= 'object'].to_list()
cont_var.remove('senior_position')
cat_var = df_clus_agg.columns[df_clus_agg.dtypes == 'object'].to_list()
cat_var.append('senior_position')
```

In [24]:
```
cont_var, cat_var
```

Out[24]:
```
(['orgyear', 'ctc', 'ctc_updated_year', 'years_of_exp'],
 ['email_hash', 'company_hash', 'job_position', 'senior_position'])
```

In [25]:
```
# Set up the figure
plt.figure(figsize=(20, 10))
plt.suptitle("Distributions")

# Define some colors for the histograms (you can customize these colors)
colors = ['#3498db', '#e74c3c', '#2ecc71', '#f39c12']

# Plot histograms with KDE for each continuous variable in 'cont_var'
for i, col in enumerate(cont_var, 1):
    plt.subplot(2, 2, i)
    sns.histplot(df_clus_agg[col], kde=True, color=colors[i-1])  # Use diffe
    plt.grid()

# Show the plots
plt.show()
```

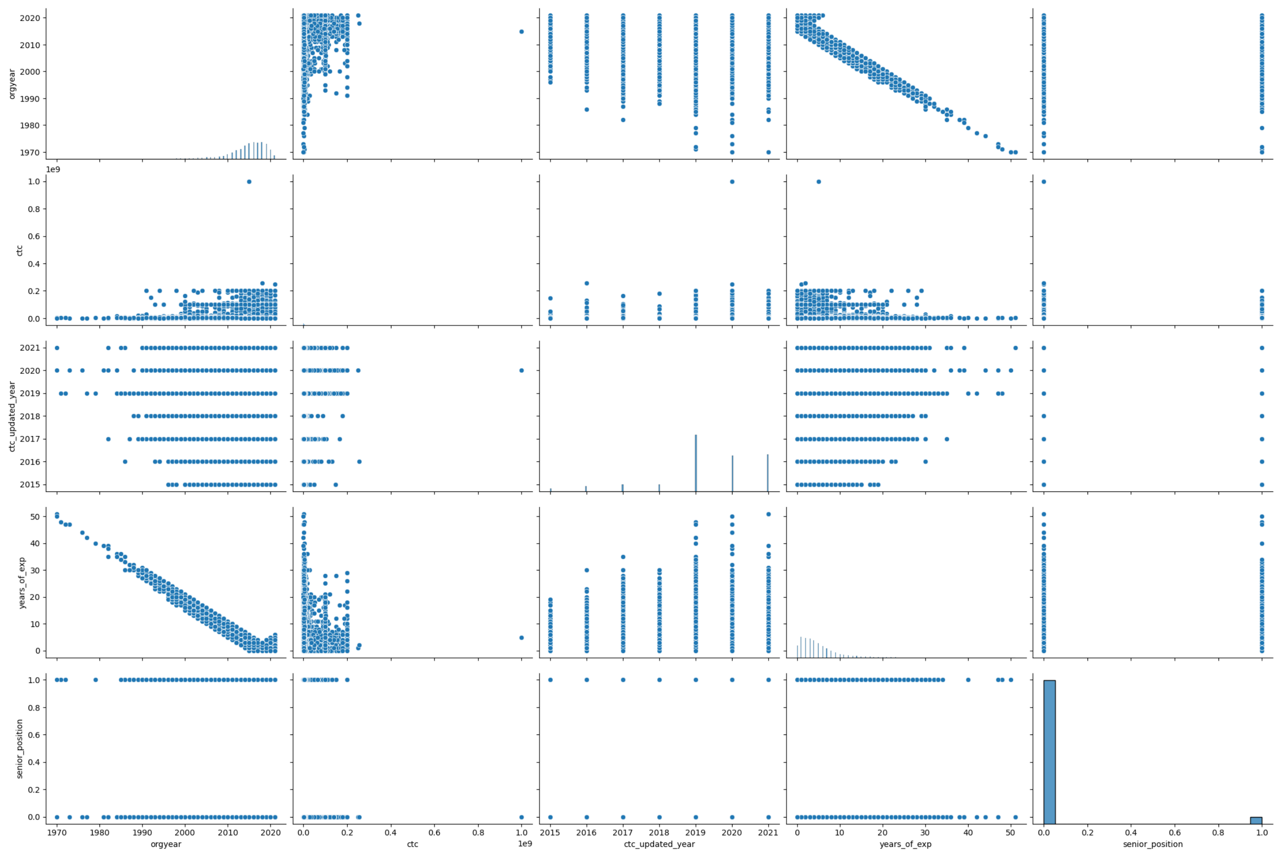


Observation:

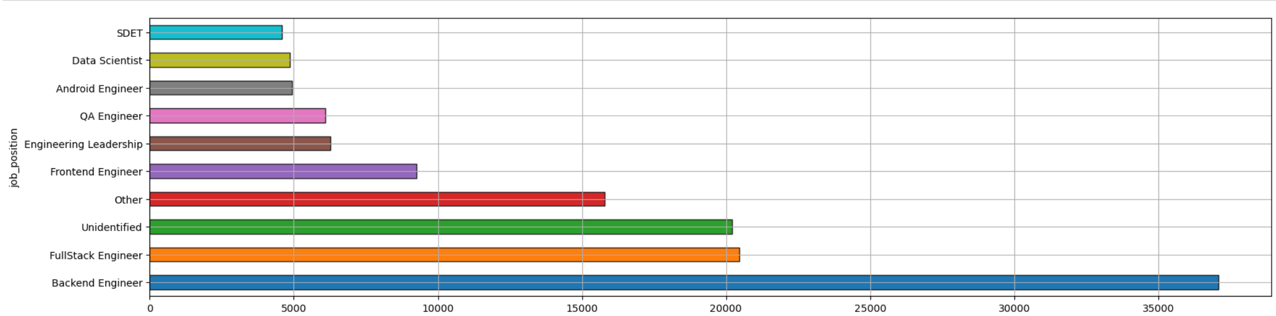
ctc_updated_year is a multimodal distribution.

ctc feature has outliers.

In [26]:
```
# Assuming df_clus_agg is your DataFrame

sns.pairplot(df_clus_agg, height=3, aspect=1.5)
plt.grid(True)
plt.show()
```

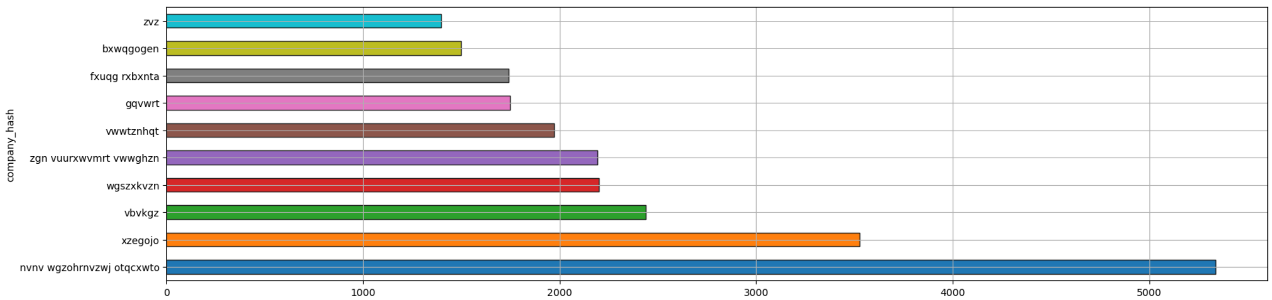


In [27]:
```
# Most Common Job Positions
common_jobs = df_clus_agg.groupby("job_position").size().sort_values(ascendi

# Custom color palette
custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c

# Plot the bar chart with custom colors
common_jobs.plot(kind="barh", edgecolor='0.15', figsize=(20, 5), color=custo
plt.grid(True)
plt.show()
```



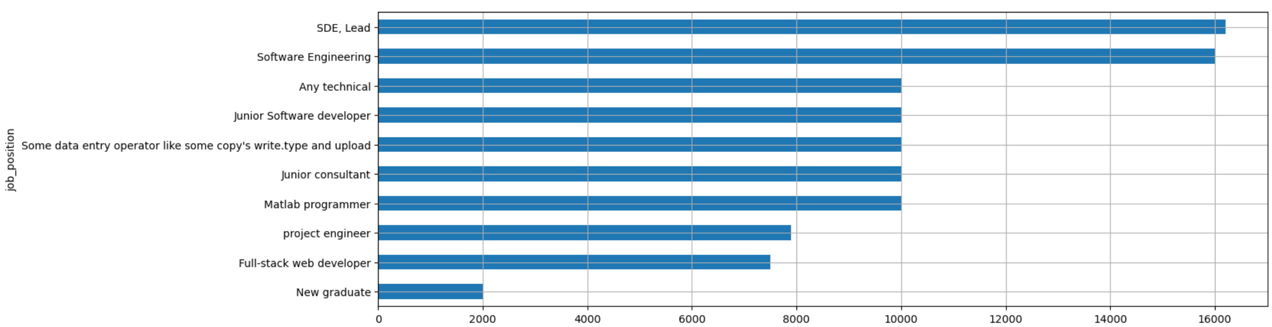
Observation: its very evident that 'Backend Engineer' is the most common job.

In [28]:
```
# Most Common Job Positions
common_companies = df_clus_agg.groupby("company_hash").size().sort_values(as

# Custom color palette
custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c

# Plot the bar chart with custom colors
common_companies.plot(kind="barh", edgecolor='0.15', figsize=(20, 5), color=
plt.grid(True)
plt.show()
```

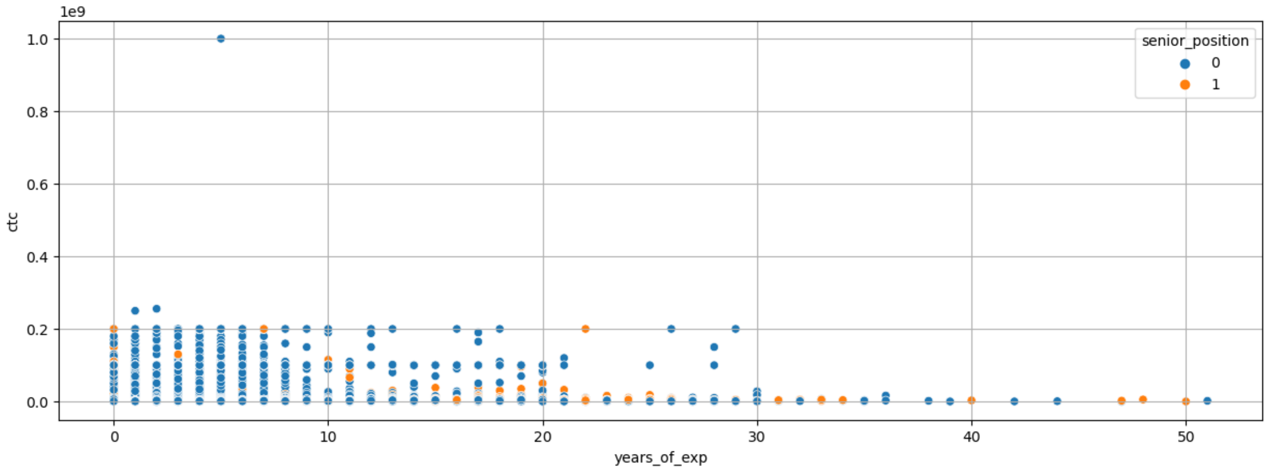


Observation: its very evident that 'nvnv wgzohrnvzwj otqcxwto' is the most common company name.

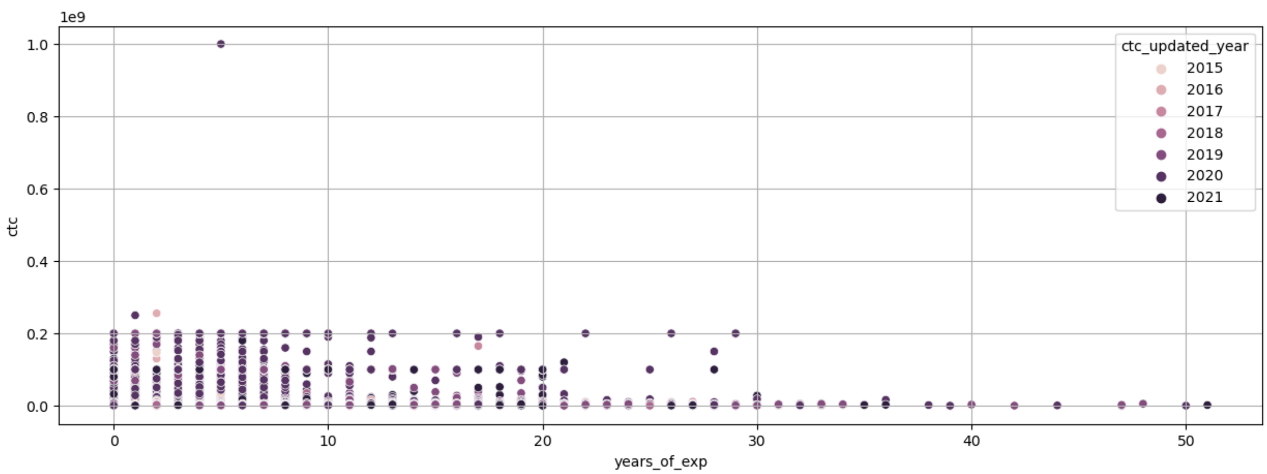
In [29]:
```
# Analysing Salary and Job_position
df_clus_agg.groupby('job_position')['ctc'].median().sort_values()[:10].plot(
                                                              kind='barh', gri
plt.show()
```



In [30]:
```
# Analysing ctc, orgyear, promotion and ctc_updated_year
plt.figure(figsize=(15,5))
sns.scatterplot(data=df_clus_agg, x='years_of_exp', y='ctc', hue='senior_pos
plt.grid()
plt.show()
```



In [31]:
```
# Analysing ctc, year of experience, promotion
plt.figure(figsize=(15,5))
sns.scatterplot(data=df_clus_agg, x='years_of_exp', y='ctc', hue='ctc_update
plt.grid()
plt.show()
```



Manual Clustering

In [32]:
```
# 5 point summary of CTC

# Top 10 companies on the basis of Avg Pay
print('Top 10 companies:-')
display(df_clus_agg.groupby('company_hash')['ctc'].describe().sort_values("m

# Bottom 10 companies on the basis of Avg pay
print('Bottom 10 companies:-')
display(df_clus_agg.groupby('company_hash')['ctc'].describe().sort_values("m
```

```
Top 10 companies:-
```

	count	mean	std	min	25%	50
company_hash						
whmxw rgsxwo uqxcvnt rxbxnta	1.00	1,000,150,000.00	nan	1,000,150,000.00	1,000,150,000.00	1,000,150,000.
aveegaxr xzntqzvnxgzvr hzxctqoxnj	1.00	250,000,000.00	nan	250,000,000.00	250,000,000.00	250,000,000.
wrghawytqqj wxowg wgbuvzj	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.
anaw tduqtoo rxbxnta	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.
yvfrtq uvwptq	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.
uvqp wgbuhntq ojontb xzw	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.
wjzzgd	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.
xzntrrxstzwt bvzugftq otqcxwto ucn rna	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.
ltnvxqfvjo	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.
gqmxn ogenfvqt xzw	1.00	200,000,000.00	nan	200,000,000.00	200,000,000.00	200,000,000.

```
Bottom 10 companies:-
```

company_hash	count	mean	std	min	25%	50%	75%	max
xm	2.00	15.50	0.71	15.00	15.25	15.50	15.75	16.00
uqvpqxnx voogwxvnto	1.00	24.00	nan	24.00	24.00	24.00	24.00	24.00
ftm ongqt	1.00	25.00	nan	25.00	25.00	25.00	25.00	25.00
vcvzn sqghu	1.00	300.00	nan	300.00	300.00	300.00	300.00	300.00
uqgmrtb ogrcxzs	1.00	500.00	nan	500.00	500.00	500.00	500.00	500.00
hzxctqoxnj ge mqvorxv	1.00	1,000.00	nan	1,000.00	1,000.00	1,000.00	1,000.00	1,000.00
uvznohxn uqgetooxgzvr	1.00	1,000.00	nan	1,000.00	1,000.00	1,000.00	1,000.00	1,000.00
hzxctqoxnj ge ayvpv	1.00	1,000.00	nan	1,000.00	1,000.00	1,000.00	1,000.00	1,000.00
cxtfqvj	1.00	1,000.00	nan	1,000.00	1,000.00	1,000.00	1,000.00	1,000.00

In [33]:

```
# 5 point summary of CTC

# Top 10 Job Position on the basis of Avg Pay
print('Top 10 Job Position:-')
display(df_clus_agg.groupby('job_position')['ctc'].describe().sort_values("m

# Bottom 10 companies on the basis of Avg pay
print('Bottom 10 Job Position:-')
display(df_clus_agg.groupby('job_position')['ctc'].describe().sort_values("m
```

```
Top 10 Job Position:-
```

job_position	count	mean	std	min	25%	50
Telar	1.00	100,000,000.00	nan	100,000,000.00	100,000,000.00	100,000,000.(
Business Man	1.00	100,000,000.00	nan	100,000,000.00	100,000,000.00	100,000,000.(
7033771951	1.00	100,000,000.00	nan	100,000,000.00	100,000,000.00	100,000,000.(
Reseller	1.00	100,000,000.00	nan	100,000,000.00	100,000,000.00	100,000,000.(
Jharkhand	1.00	100,000,000.00	nan	100,000,000.00	100,000,000.00	100,000,000.(
Owner	1.00	100,000,000.00	nan	100,000,000.00	100,000,000.00	100,000,000.(
Data entry	1.00	100,000,000.00	nan	100,000,000.00	100,000,000.00	100,000,000.(
Safety officer	1.00	99,900,000.00	nan	99,900,000.00	99,900,000.00	99,900,000.(
Seleceman	1.00	99,900,000.00	nan	99,900,000.00	99,900,000.00	99,900,000.(
Driver	2.00	95,000,000.00	7,071,067.81	90,000,000.00	92,500,000.00	95,000,000.(

```
Bottom 10 Job Position:-
```

job_position	count	mean	std	min	25%	50%	75%	max
New graduate	1.00	2,000.00	nan	2,000.00	2,000.00	2,000.00	2,000.00	2,000.00
Full-stack web developer	1.00	7,500.00	nan	7,500.00	7,500.00	7,500.00	7,500.00	7,500.00
project engineer	1.00	7,900.00	nan	7,900.00	7,900.00	7,900.00	7,900.00	7,900.00
Any technical	1.00	10,000.00	nan	10,000.00	10,000.00	10,000.00	10,000.00	10,000.00
Some data entry operator like some copy's write.type and upload	1.00	10,000.00	nan	10,000.00	10,000.00	10,000.00	10,000.00	10,000.00
Matlab programmer	1.00	10,000.00	nan	10,000.00	10,000.00	10,000.00	10,000.00	10,000.00
Junior consultant	1.00	10,000.00	nan	10,000.00	10,000.00	10,000.00	10,000.00	10,000.00
Junior Software developer	1.00	10,000.00	nan	10,000.00	10,000.00	10,000.00	10,000.00	10,000.00
Software Engineering	1.00	16,000.00	nan	16,000.00	16,000.00	16,000.00	16,000.00	16,000.00

In [34]:

```
 # 5 point summary of CTC

# Binning Years of experience
labels = ['0', '1-2', '3-5', '5-10', '10-20', '20+']
bins = [0,1,3,5,10,20,np.inf]
df_clus_agg['years_of_exp_bin'] = pd.cut(df_clus_agg['years_of_exp'], labels

# Years of experience vs Avg Pay
print('Years of Experience Statistical Summary:-')
display(df_clus_agg.groupby('years_of_exp_bin')['ctc'].describe().sort_value
```

```
Years of Experience Statistical Summary:-
```

years_of_exp_bin	count	mean	std	min	25%	50%
20+	1,488.00	5,608,887.00	17,784,949.91	1,000.00	1,000,000.00	2,700,000.00
10-20	15,447.00	3,087,864.35	9,466,419.47	1,000.00	1,100,000.00	2,000,000.00
0	12,087.00	2,875,993.89	15,711,024.39	24.00	400,000.00	700,000.00
5-10	46,782.00	2,688,273.34	13,964,005.42	2.00	700,000.00	1,200,000.00
3-5	36,640.00	2,290,661.35	13,023,153.47	1,000.00	500,000.00	800,000.00
1-2	40,967.00	2,031,542.34	12,123,342.79	15.00	400,000.00	700,000.00

In [35]:

```
# Creating Manual Clusters on the basis of company_hash, job_position, years
df_avg_ctc = df_clus_agg.groupby(['company_hash', 'job_position', 'years_of_
df_avg_ctc = df_avg_ctc.rename(columns={'ctc': 'employee_avg_ctc'})
df_avg_ctc.dropna(inplace=True)

# Merge this on the company dataset
df_clus_merged = pd.merge(left=df_clus_agg, right=df_avg_ctc, on=['company_h

# Function to apply flags based on the criteria
def designation_flag(data):
    """
    Assigns a designation based on the employee's CTC compared to the averag

    Designation 1: If employee's CTC is 50% higher than the average CTC.
    Designation 2: If employee's CTC is within ±50% of the average CTC.
    Designation 3: If employee's CTC is 50% lower than the average CTC.
    """
    # Calculate the 50% boundary values for higher and lower CTC
    upper_bound = data['employee_avg_ctc'] * 1.5
    lower_bound = data['employee_avg_ctc'] * 0.5

    # Apply the logic for Designation flag
    if data['ctc'] > upper_bound:
        return 1  # Designation 1: 50% higher than the average CTC
    elif lower_bound <= data['ctc'] <= upper_bound:
        return 2  # Designation 2: CTC is within ±50% of the average CTC
    else:
        return 3  # Designation 3: 50% lower than the average CTC

# Apply the designation flag function to create learner_designation
df_clus_merged['learner_designation'] = df_clus_merged.apply(designation_fla

# Display the result to confirm the learner designations
df_clus_merged.head()
```

Out[35]:

	email_hash	company_hash	orgyear	ctc
0	00003288036a44374976948c327f246fdbdf0778546904...	bxwqgogen	2012.0	3500000
1	0000aaa0e6b61f7636af1954b43d294484cd151c9b3cf6...	nqsn axsxnvr	2013.0	250000
2	0000d58fbc18012bf6fa2605a7b0357d126ee69bc41032...	gunhb	2021.0	1300000
3	000120d0c8aa304fcf12ab4b85e21feb80a342cfea03d4...	bxwqgotbx wgqugqvnxgz	2004.0	2000000
4	00014d71a389170e668ba96ae8e1f9d991591acc899025...	fvrbvqn rvmo	2009.0	3400000

In [36]:

```
# Creating Avg Salary for the whole dataset (average of 'ctc' column)
df_clus_merged['avg_ctc'] = df_clus_merged['ctc'].mean()

# Creating Manual Clusters on the basis of company_hash
df_avg_ctc = df_clus_agg.groupby(['company_hash'])['ctc'].mean().reset_index
df_avg_ctc.dropna(inplace=True)

# Merge this on the company dataset to get the company-specific average CTC
df_clus_merged = pd.merge(left=df_clus_merged, right=df_avg_ctc, on=['compan

# Function to apply company tier based on the logic described
def tier_flag(data):
    """
    Assign tiers based on the company's average CTC relative to the overall
    - Tier 1: Company’s average CTC is 50% lower than the overall dataset av
    - Tier 2: Company’s average CTC is within ±50% of the overall dataset av
    - Tier 3: Company’s average CTC is 50% higher than the overall dataset a
    """
    # Calculate 50% boundaries relative to the overall dataset's average CTC
    lower_bound = data['avg_ctc'] * 0.5
    upper_bound = data['avg_ctc'] * 1.5

    # Tier assignment based on conditions
    if data['avg_company_ctc'] < lower_bound:
        return 1  # Tier 1: 50% lower than average dataset CTC
    elif data['avg_company_ctc'] > upper_bound:
        return 3  # Tier 3: 50% higher than average dataset CTC
    else:
        return 2  # Tier 2: Within ±50% of the average dataset CTC

# Apply the tier assignment function to the DataFrame
df_clus_merged['company_tier'] = df_clus_merged.apply(tier_flag, axis=1)

# Show the result
df_clus_merged.head()
```

Out[36]:

	email_hash	company_hash	orgyear	ctc
0	00003288036a44374976948c327f246fdbdf0778546904...	bxwqgogen	2012.0	3500000
1	0000aaa0e6b61f7636af1954b43d294484cd151c9b3cf6...	nqsn axsxnvr	2013.0	250000
2	0000d58fbc18012bf6fa2605a7b0357d126ee69bc41032...	gunhb	2021.0	1300000
3	000120d0c8aa304fcf12ab4b85e21feb80a342cfea03d4...	bxwqgotbx wgqugqvnxgz	2004.0	2000000
4	00014d71a389170e668ba96ae8e1f9d991591acc899025...	fvrbvqn rvmo	2009.0	3400000

In [41]:

```
# Calculate the average job CTC per company and job position
df_avg_ctc = df_clus_agg.groupby(['company_hash', 'job_position'])['ctc'].me
df_avg_ctc.dropna(inplace=True)

# Merge the average job CTC into the main DataFrame based on company and job
df_clus_merged = pd.merge(left=df_clus_merged, right=df_avg_ctc, on=['compan

# Function to apply job class flags based on the comparison between job CTC
def class_flag(data):
    """
    Assign job class based on the company's average job CTC relative to the
    - Class 1: If the average job CTC is 50% lower than the company's averag
    - Class 2: If the average job CTC is within ±50% of the company's averag
    - Class 3: If the average job CTC is 50% higher than the company's avera
    """
    # Calculate the lower and upper bounds for Class 1 and Class 3
    lower_bound = data['avg_company_ctc'] * 0.5
    upper_bound = data['avg_company_ctc'] * 1.5

    # Assign class based on comparison to bounds
    if data['avg_job_ctc'] < lower_bound:
        return 1  # Class 1: 50% lower than company CTC
    elif data['avg_job_ctc'] > upper_bound:
        return 3  # Class 3: 50% higher than company CTC
    else:
        return 2  # Class 2: Within ±50% of company CTC

# Apply the classification function to assign job class
df_clus_merged['job_class'] = df_clus_merged.apply(class_flag, axis=1)

# Display the results
df_clus_merged.head()
```

Out[41]:

	email_hash	company_hash	orgyear	ctc
0	00003288036a44374976948c327f246fdbdf0778546904...	bxwqgogen	2012.0	3500000
1	0000aaa0e6b61f7636af1954b43d294484cd151c9b3cf6...	nqsn axsxnvr	2013.0	250000
2	0000d58fbc18012bf6fa2605a7b0357d126ee69bc41032...	gunhb	2021.0	1300000
3	000120d0c8aa304fcf12ab4b85e21feb80a342cfea03d4...	bxwqgotbx wgqugqvnxgz	2004.0	2000000
4	00014d71a389170e668ba96ae8e1f9d991591acc899025...	fvrbvqn rvmo	2009.0	3400000

Analyzing Manual Clusters

In [42]:

```
##Top10
# Filter employees with 'learner_designation' equal to 1 (those with the hig
high_earning_employees = df_clus_merged[df_clus_merged['learner_designation'

# Sort these employees by their CTC in descending order
sorted_high_earners = high_earning_employees.sort_values('ctc', ascending=Fa

# Get the top 10 highest earning employees
top_10_high_earners = sorted_high_earners.head(10)

# Display the result
top_10_high_earners
```

Out[42]:

	email_hash	company_hash	orgyear	
73726	7b570fed7acfedd69f3dcbd66165407458b4337467d439...	vbvkgz	2015.0	200(
70793	76708a11cb61a030ff3da827b0fd19aff536c3793c1816...	gnytq	2012.0	200(
16992	1c0d0d8f8c85458f214991dd9855ca50cc897d34efcb14...	xzegojo	2016.0	200(
31506	34804f1160325392e2a0ba449c44f3b424cb9ea0e0295f...	bxwqgogen	2013.0	200(
140975	eb552f9d6f12d47656472a3f7c6a6625ebf3d699edb4b0...	ovrtoegqwt	2013.0	200(
8064	0d235f7e73cd9484909b32a35c69df12296a051f68ef83...	nvnv wgzohrnvzwj otqcxwto	2017.0	200(
73192	7a723f5b71698674b79bd2195c3bb58d3fcbf4ddb75a04...	ntwy bvyxzaqv	2019.0	200(
102324	aad581a532f319c76c6e73937572feed9867d5ee2f1093...	wgszxkvzn	2014.0	200(
117542	c44995942d317b3a36725bf0bfb34412741fbb35839177...	zgzt	2018.0	200(
50601	54bafd5fc688d31915438560bd4e94225a829a5619cb11...	ftrro evqsg	2015.0	200(

Observation: CTC of 200000000 can be associated as a very high income

In [43]:

```
##Bottom10
# Filter employees with 'learner_designation' equal to 1 (those with the hig
low_earning_employees = df_clus_merged[df_clus_merged['learner_designation']

# Sort these employees by their CTC in descending order
sorted_low_earners = low_earning_employees.sort_values('ctc', ascending=True

# Get the top 10 highest earning employees
bottom_10_earners = sorted_low_earners.head(10)

# Display the result
bottom_10_earners
```

Out[43]:

	email_hash	company_hash	orgyear	ct
31847	3505b02549ebe2c95840ac6f0a35561a3b4cbe4b79cdb1...	xzntqcxtfmxn	2014.0	
145466	f2b58aeed3c074652de2cfd3c0717a5d21d6fbcf342a78...	xzntqcxtfmxn	2013.0	
21509	23ad96d6b6f1ecf554a52f6e9b61677c7d73d8a409a143...	xzntqcxtfmxn	2013.0	1
92794	9af3dca6c9d705d8d42585ccfce2627f00e1629130d14e...	zvz	2019.0	60
77005	80ba0259f9f59034c4927cf3bd38dc9ce2eb60ff18135b...	nvnv wgzohrnvzwj otqcxwto	2012.0	60
80945	8747d9599e2ba1a8624e8bea834ab7a870c89ccca74204...	zv	2004.0	100
25085	299f764fcae62f331f3c5eb1b451e7107302ded46e2a71...	zgn vuurxwvmrt vwwghzn	2007.0	100
80267	8625d6d072e12dad0c5748ab010e1d0315736a359e2bb5...	nvnv wgzohrnvzwj otqcxwto	2013.0	100
46950	4ea8ce7809d8c69147d243bad53d88d016a1151690b8b6...	zvz	2010.0	100
84419	8d1e069a03fc437876b406b8c93bc7e07577f9836222bd...	zgn vuurxwvmrt vwwghzn	2021.0	100

In [44]:

```
# Filter for companies with tier 1 designation
tier_1_companies = df_clus_merged[df_clus_merged['company_tier'] == 1]

# Group by company and calculate the mean CTC
mean_ctc_by_company = tier_1_companies.groupby('company_hash')['ctc'].mean()

# Sort companies by their average CTC in descending order and get the top 10
top_10_companies = mean_ctc_by_company.sort_values(ascending=False).head(10)

# Display the top 10 company hashes
top_10_companies.index
```

Out[44]:

```
Index(['evqb shxat', 'qov', 'xbvstpxn', 'wqtaxn ovxogz xzaxv',
       'zgpxv ogrhnxgzo', 'vuugqwyxa', 'tzntrrv xn ucn rna',
       'mxqrvogen ogrhnxgzo', 'ola xzntqzvnxgzvr',
       'ongqjduqtoo otrtwnta mj ntwyonvqo 2017'],
      dtype='object', name='company_hash')
```

In [45]:
```
# Filter data for job class 1 (assuming 'job_class' 1 refers to the relevant
job_class_1_data = df_clus_merged[df_clus_merged['job_class'] == 1]

# Group by company and job position to calculate the average CTC for each co
avg_ctc_by_job_position = job_class_1_data.groupby(['company_hash', 'job_pos

# For each company, get the top 2 job positions with the highest average CTC
top_2_positions_per_company = avg_ctc_by_job_position.groupby(level=0, group

# Display the result
top_2_positions_per_company
```

Out[45]:
```
company_hash                job_position
01 ojztqsj                  Android Engineer            270000.0
1bs ntwyzgrgsxto ucn rna    QA Engineer                 620000.0
                            Other                       300000.0
1p qtnvxr ztnfgqpo          Android Engineer            200000.0
201518                      FullStack Engineer          200000.0
                                                          ...
zxxn ntwyzgrgsxto rxbxnta   Product Designer           1300000.0
                            Android Engineer           1292500.0
zxzlvwvqn                   Data Analyst                715000.0
                            Area Operations Manager     600000.0
zxztrtvuo                   Other                       450000.0
Name: ctc, Length: 3966, dtype: float64
```

Data Preprocesssing

In [46]:
```
# Importing necessary libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

In [47]:
```
# Assigning DataFrame for processing
df_clus_processed = df_clus_agg.copy()

# Drop Identifier and redundant features
df_clus_processed.drop(columns=['email_hash','years_of_exp_bin'], inplace =

df_clus_processed
```

Out[47]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	years_of_exp	s
0	bxwqgogen	2012.0	3500000	Backend Engineer	2019.0	7.0	
1	nqsn axsxnvr	2013.0	250000	Backend Engineer	2020.0	7.0	
2	gunhb	2021.0	1300000	FullStack Engineer	2019.0	2.0	
3	bxwqgotbx wgqugqvnxgz	2004.0	2000000	FullStack Engineer	2021.0	17.0	
4	fvrbvqn rvmo	2009.0	3400000	Unidentified	2018.0	9.0	
...	...	...	...	...	...	...	
153406	tqxwoogz ogenfvqt wvbuho	2004.0	3529999	QA Engineer	2019.0	15.0	
153407	trnqvcg	2015.0	1600000	Unidentified	2018.0	3.0	
153408	znn avnv srgmvr atrxctqj otqcxwto	2014.0	900000	Devops Engineer	2019.0	5.0	
153409	zwq wgqugqvnxgz	2020.0	700000	FullStack Engineer	2020.0	0.0	
153410	sgrabvz ovwyo	2018.0	1500000	FullStack Engineer	2021.0	3.0	

153411 rows × 7 columns

In [48]:

```
# Frequency Encoding - Reusable function to encode and normalize features ba

def frequency_encode(df, column_name):
    """
    Encodes the column using frequency encoding (normalized) and returns the
    It normalizes the frequency by dividing the count of each unique value b
    """
    frequency_map = df[column_name].value_counts(normalize=True)
    return df[column_name].map(frequency_map)

# Apply Frequency Encoding to 'company_hash' and 'job_position'
df_clus_processed['company_hash'] = frequency_encode(df_clus_processed, 'com
df_clus_processed['job_position'] = frequency_encode(df_clus_processed, 'job

# Display the first few rows to check the result
df_clus_processed.head()
```

Out[48]:

	company_hash	orgyear	ctc	job_position	ctc_updated_year	years_of_exp	senior_
0	0.009778	2012.0	3500000	0.241704	2019.0	7.0	
1	0.000007	2013.0	250000	0.241704	2020.0	7.0	
2	0.001049	2021.0	1300000	0.133406	2019.0	2.0	
3	0.000072	2004.0	2000000	0.133406	2021.0	17.0	
4	0.003644	2009.0	3400000	0.131699	2018.0	9.0	

In [49]:
```
# Log Normalizing CTC -> Salaries generally follow Log Normal Distribution,
df_clus_processed['ctc'] = np.log10(df_clus_processed['ctc'])
```

In [51]:
```
# Outliers after Log Transformation

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
q1 = df_clus_processed['ctc'].quantile(0.25)
q3 = df_clus_processed['ctc'].quantile(0.75)

# Calculate the Interquartile Range (IQR)
iqr = q3 - q1

# Define the upper and lower bounds for outliers
upper_bound = q3 + 1.5 * iqr
lower_bound = q1 - 1.5 * iqr

# Filter out the outliers by checking the conditions
outliers = df_clus_processed[(df_clus_processed['ctc'] > upper_bound) | (df_

# Calculate the total number of outliers and the percentage of outliers
total_outliers = len(outliers)
total_outliers_percentage = 100 * total_outliers / len(df_clus_processed)

# Print out the outlier percentage
print(f"Outlier Percentage for CTC is : {total_outliers_percentage:.2f}%")
```

```
Outlier Percentage for CTC is : 4.71%
```

In [54]:
```
# Importing Necessary Libraries
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd

# Create Custom Transformer for Outlier Removal
class OutlierRemoval(BaseEstimator, TransformerMixin):
    def __init__(self, column='ctc'):
        """
        Initializes the transformer with the given column name for outlier r

        Parameters:
        - column (str): The column name for outlier detection (default is 'c
        """
```

```
        self.column = column

    def fit(self, X, y=None):
        """
        The fit method does nothing but is required for compatibility with s

        Parameters:
        - X (pd.DataFrame): The input data.
        - y (None): Ignored.

        Returns:
        - self: The fitted transformer.
        """
        return self

    def transform(self, X, y=None):
        """
        Applies the outlier removal process to the input data.

        Parameters:
        - X (pd.DataFrame): The input data.
        - y (None): Ignored.

        Returns:
        - pd.DataFrame: The filtered data with outliers removed.
        """
        return self.remove_outlier(X)

    def fit_transform(self, X, y=None):
        """
        Combines fit and transform into a single step.

        Parameters:
        - X (pd.DataFrame): The input data.
        - y (None): Ignored.

        Returns:
        - pd.DataFrame: The filtered data with outliers removed.
        """
        return self.remove_outlier(X)

    def remove_outlier(self, dataframe: pd.DataFrame) -> pd.DataFrame:
        """
        Removes outliers from the specified column based on the IQR method.

        Parameters:
        - dataframe (pd.DataFrame): The input data.

        Returns:
        - pd.DataFrame: The data with outliers removed.
        """
        # Validate input dataframe
        if self.column not in dataframe.columns:
```

```
                raise ValueError(f"Column '{self.column}' not found in the DataF

            # Calculate quantiles and IQR
            q1 = dataframe[self.column].quantile(0.25)
            q3 = dataframe[self.column].quantile(0.75)
            iqr = q3 - q1

            # Calculate upper and lower bounds for outliers
            upper_bound = q3 + 1.5 * iqr
            lower_bound = q1 - 1.5 * iqr

            # Filter out the outliers and return the cleaned dataframe
            df_filtered_data = dataframe[(dataframe[self.column] >= lower_bound)

            return df_filtered_data
```

In [55]:

```
# Importing Necessary Libraries
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Features to be scaled
# Standard scaling should typically be applied to continuous numerical featu
numeric_standard_features = ['ctc']  # Example: CTC is continuous
numeric_minmax_features = ['company_hash', 'job_position', 'orgyear', 'years

# It's important to ensure that categorical variables like 'company_hash' ar
# Note: If these are already one-hot encoded, MinMaxScaler is not ideal, but

# Creating the column transformer for scaling
preprocessor = ColumnTransformer(
    transformers=[
        ('minmax', MinMaxScaler(), numeric_minmax_features),  # Apply MinMax
        ('standard', StandardScaler(), numeric_standard_features)  # Apply S
    ],
    remainder='passthrough'  # Keep any other columns unchanged
)

# The `preprocessor` can now be used in a pipeline for transforming the data
```

Clustering

In [56]:
```
# Assigning the entire dataset as the training variable
X = df_clus_processed.copy()  # Make a copy of the dataframe to avoid modify

# Check if the dataset is large enough before sampling
sample_size = 25000
if len(X) > sample_size:
    # Create a subset of the dataset for Agglomerative Clustering (due to hi
    X_sample = X.sample(n=sample_size, random_state=42)
else:
    # If the dataset is smaller than the sample size, use the entire dataset
    X_sample = X
    print(f"Dataset is smaller than {sample_size}, using the entire dataset.

# Display the shape of the sample to verify
print(f"Sampled dataset shape: {X_sample.shape}")
```

```
Sampled dataset shape: (25000, 7)
```

In [58]:
```
# Importing necessary Libraries
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
```

In [59]:
```
# Import necessary libraries
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans, AgglomerativeClustering

# Create Pipelines for both Clustering Techniques

def create_kmeans_pipeline(n_clusters=3):
    """
    Creates a KMeans clustering pipeline.
    """
    return Pipeline([
        ('outlier', OutlierRemoval()),  # Outlier removal transformer
        ('scaler', preprocessor),  # Scaling of features
        ('kmeans', KMeans(n_clusters=n_clusters))  # KMeans with specified n
    ])

def create_agglomerative_pipeline(n_clusters=3):
    """
    Creates an Agglomerative Clustering pipeline.
    """
    return Pipeline([
        ('scaler', preprocessor),  # Scaling of features
        ('agglomerative', AgglomerativeClustering(n_clusters=n_clusters, lin
    ])

# Creating and fitting the KMeans pipeline
pipeline_kmeans = create_kmeans_pipeline(n_clusters=3)
pipeline_kmeans.fit(X)  # Fit on the entire dataset
clusters_kmeans = pipeline_kmeans.named_steps['kmeans'].labels_  # Get KMean

# Creating and fitting the Agglomerative Clustering pipeline
# We use a sample here due to computational constraints
pipeline_agglomerative_ward = create_agglomerative_pipeline(n_clusters=3)
pipeline_agglomerative_ward.fit(X_sample)  # Fit on a sample of the dataset
clusters_agglo_ward = pipeline_agglomerative_ward.named_steps['agglomerative

# Output the results for verification (you can replace this with further ana
print(f"KMeans Clusters: {clusters_kmeans[:10]}")  # Printing the first 10 K
print(f"Agglomerative Clusters: {clusters_agglo_ward[:10]}")  # Printing the
```

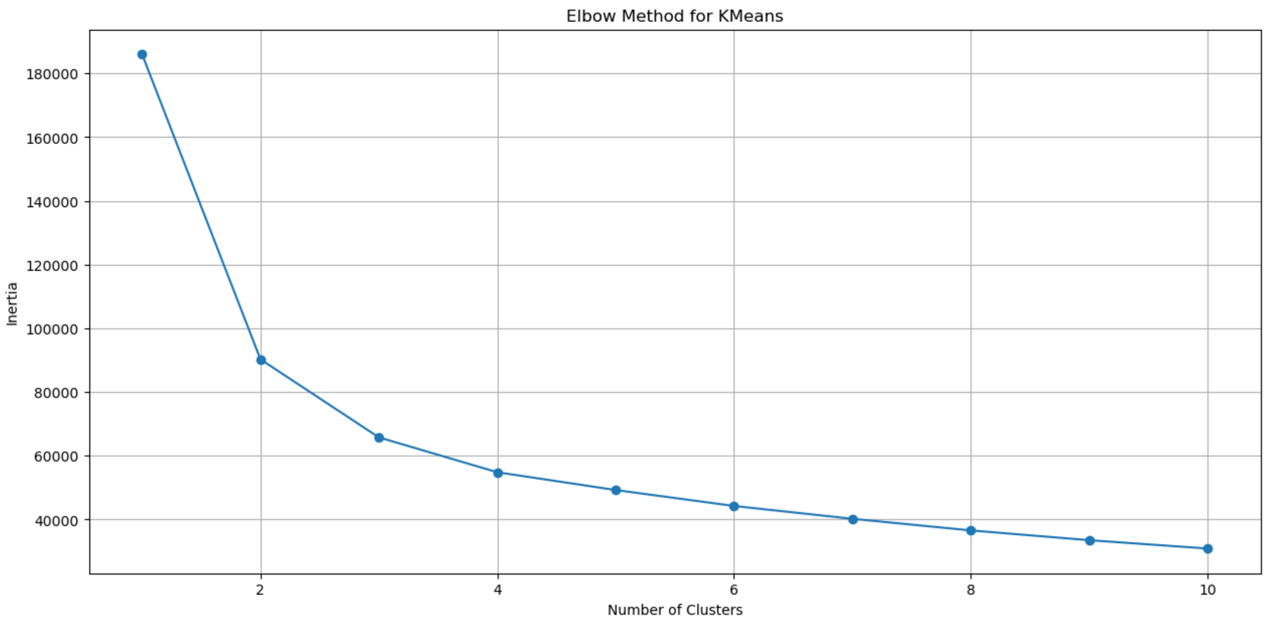
```
KMeans Clusters: [2 0 1 2 2 0 1 2 2 1]
Agglomerative Clusters: [1 2 0 2 2 1 0 2 0 2]
```

In [60]:
```
# Importing Necessary Libraries
from sklearn.metrics import silhouette_score

# Function to compute the Silhouette Score for a given pipeline and dataset
def calculate_silhouette_score(pipeline, X, clusters):
    """
    Calculate the silhouette score given a pipeline, dataset, and clustering

    Parameters:
    - pipeline: The fitted pipeline that contains the scaler and other prepr
    - X: The dataset to be used for scoring.
    - clusters: The cluster labels predicted by the model.

    Returns:
    - silhouette_score: The silhouette score for the given clustering.
    """
    # Apply preprocessing steps from the pipeline
    X_processed = pipeline.named_steps['outlier'].transform(X) if 'outlier'
    X_scaled = pipeline.named_steps['scaler'].fit_transform(X_processed)

    # Calculate and return the silhouette score
    return silhouette_score(X_scaled, clusters)

# Silhouette Score for KMeans
kmeans_silhouette = calculate_silhouette_score(pipeline_kmeans, X, clusters_

# Silhouette Score for Agglomerative Clustering (using a sample)
agglo_ward_silhouette = calculate_silhouette_score(pipeline_agglomerative_wa

# Printing the results
print(f'Silhouette Score for KMeans: {kmeans_silhouette:.3f}')
print(f'Silhouette Score for Agglomerative Ward: {agglo_ward_silhouette:.3f}
```

```
Silhouette Score for KMeans: 0.321
Silhouette Score for Agglomerative Ward: 0.302
```

Observation: There is scope for improvement of Silhouette Scores and this implies that the intercluster distance observed is not good enough

In [61]:
```
# Importing necessary libraries
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.pipeline import Pipeline

# Function to calculate inertia using the Elbow Method
def calculate_inertia(X, max_clusters=10):
    """
    Calculates inertia for different numbers of clusters using the Elbow met

    Parameters:
    - X (pd.DataFrame): The dataset to apply KMeans clustering.
    - max_clusters (int): Maximum number of clusters to evaluate.

    Returns:
    - inertia (list): A list containing inertia values for each number of cl
    """
    inertia = []

    # Create the pipeline once and update the number of clusters dynamically
    for n_clusters in range(1, max_clusters + 1):
        pipeline_kmeans = Pipeline([
            ('outlier', OutlierRemoval()),  # Outlier removal
            ('scaler', preprocessor),      # Feature scaling
            ('kmeans', KMeans(n_clusters=n_clusters))  # KMeans clustering
        ])

        # Fit the pipeline and append the inertia to the list
        pipeline_kmeans.fit(X)
        inertia.append(pipeline_kmeans.named_steps['kmeans'].inertia_)

    return inertia

# Calculate inertia for cluster sizes from 1 to 10
inertia = calculate_inertia(X)

# Plotting the Elbow Method for KMeans
plt.figure(figsize=(15, 7))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for KMeans')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
```



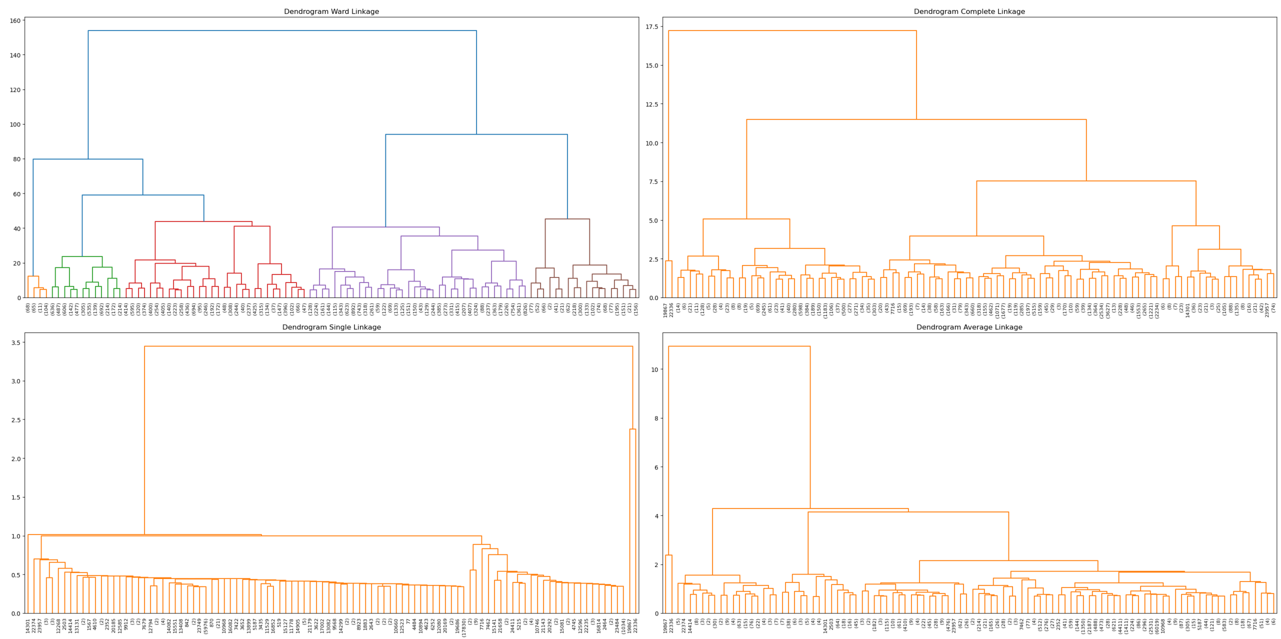
Observation: K=2 would be an ideal option.

In [62]:
```
# Importing necessary libraries
from scipy.cluster.hierarchy import dendrogram, linkage

# Function to compute linkage for different methods
def compute_linkage(X_scaled, method):
    """
    Compute linkage matrix for hierarchical clustering using a specified met

    Parameters:
    - X_scaled: The scaled dataset.
    - method: The linkage method to use ('ward', 'complete', 'single', 'aver

    Returns:
    - linkage_matrix: The linkage matrix computed for the specified method.
    """
    return linkage(X_scaled, method=method)

# Scaling the dataset
X_scaled = preprocessor.fit_transform(X_sample)  # Apply scaling on the samp

# Define linkage methods
linkage_methods = ['ward', 'complete', 'single', 'average']

# Create a dictionary to store linkage matrices for each method
linkage_matrices = {method: compute_linkage(X_scaled, method) for method in

# Output the linkage matrices for inspection
for method, linkage_matrix in linkage_matrices.items():
    print(f"Linkage matrix for {method} method:")
    print(linkage_matrix[:5])  # Show the first 5 rows of each matrix for qu
```

```
Linkage matrix for ward method:
[[1.7948e+04 2.4916e+04 0.0000e+00 2.0000e+00]
 [3.7970e+03 2.2036e+04 0.0000e+00 2.0000e+00]
 [7.7940e+03 1.3854e+04 0.0000e+00 2.0000e+00]
 [1.5819e+04 2.5002e+04 0.0000e+00 3.0000e+00]
 [1.5698e+04 1.6212e+04 0.0000e+00 2.0000e+00]]
Linkage matrix for complete method:
[[1.7948e+04 2.4916e+04 0.0000e+00 2.0000e+00]
 [3.7970e+03 2.2036e+04 0.0000e+00 2.0000e+00]
 [7.7940e+03 1.3854e+04 0.0000e+00 2.0000e+00]
 [1.5819e+04 2.5002e+04 0.0000e+00 3.0000e+00]
 [4.3750e+03 1.6896e+04 0.0000e+00 2.0000e+00]]
Linkage matrix for single method:
[[7.9940e+03 1.0993e+04 0.0000e+00 2.0000e+00]
 [2.4730e+03 1.4345e+04 0.0000e+00 2.0000e+00]
 [2.1959e+04 2.5001e+04 0.0000e+00 3.0000e+00]
 [2.4660e+03 2.2932e+04 0.0000e+00 2.0000e+00]
 [7.0990e+03 8.4500e+03 0.0000e+00 2.0000e+00]]
Linkage matrix for average method:
[[1.7948e+04 2.4916e+04 0.0000e+00 2.0000e+00]
 [3.7970e+03 2.2036e+04 0.0000e+00 2.0000e+00]
 [4.3750e+03 1.6896e+04 0.0000e+00 2.0000e+00]
 [2.1620e+03 1.4229e+04 0.0000e+00 2.0000e+00]
 [5.8520e+03 2.2705e+04 0.0000e+00 2.0000e+00]]
```

In [65]:
```
# Importing necessary libraries
from scipy.cluster.hierarchy import dendrogram, linkage

# Function to compute linkage for different methods
def compute_linkage(X_scaled, method):
    """
    Compute linkage matrix for hierarchical clustering using a specified met

    Parameters:
    - X_scaled: The scaled dataset.
    - method: The linkage method to use ('ward', 'complete', 'single', 'aver

    Returns:
    - linkage_matrix: The linkage matrix computed for the specified method.
    """
    return linkage(X_scaled, method=method)

# Scaling the dataset
X_scaled = preprocessor.fit_transform(X_sample)  # Apply scaling on the samp

# Linkage Methods
linkage_methods = ['ward', 'complete', 'single', 'average']

# Compute the linkage matrices and assign to variables D1 to D4
D1 = compute_linkage(X_scaled, 'ward')
D2 = compute_linkage(X_scaled, 'complete')
D3 = compute_linkage(X_scaled, 'single')
D4 = compute_linkage(X_scaled, 'average')

# Store the linkage results in a dictionary for later use (optional)
linkage_matrices = {
    'ward': D1,
    'complete': D2,
    'single': D3,
    'average': D4
}
```

In [66]:
```
# Import necessary libraries
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram

# Set up figure for plotting dendrograms
plt.figure(figsize=(30, 15))

# Define the color scheme with red and blue
# You can set the color threshold for differentiating the branches
color_threshold = 50  # This is an example threshold to distinguish branches

# Ward Linkage Dendrogram
plt.subplot(2, 2, 1)
plt.title('Dendrogram Ward Linkage')
dendrogram(D1, show_leaf_counts=True,
           leaf_rotation=90, leaf_font_size=8,
           truncate_mode='lastp', p=100,
           color_threshold=color_threshold)

# Complete Linkage Dendrogram
plt.subplot(2, 2, 2)
plt.title('Dendrogram Complete Linkage')
dendrogram(D2, show_leaf_counts=True,
           leaf_rotation=90, leaf_font_size=8,
           truncate_mode='lastp', p=100,
           color_threshold=color_threshold)

# Single Linkage Dendrogram
plt.subplot(2, 2, 3)
plt.title('Dendrogram Single Linkage')
dendrogram(D3, show_leaf_counts=True,
           leaf_rotation=90, leaf_font_size=8,
           truncate_mode='lastp', p=100,
           color_threshold=color_threshold)

# Average Linkage Dendrogram
plt.subplot(2, 2, 4)
plt.title('Dendrogram Average Linkage')
dendrogram(D4, show_leaf_counts=True,
           leaf_rotation=90, leaf_font_size=8,
           truncate_mode='lastp', p=100,
           color_threshold=color_threshold)

# Display the plot
plt.tight_layout()
plt.show()
```



Observation: Between 3 to 5 clusters, the separation is optimum as per the above Dendogram results.

In [67]:
```
# Import necessary libraries
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans, AgglomerativeClustering

# Function to create an Agglomerative Clustering pipeline with a given numbe
def create_agglomerative_pipeline(n_clusters, linkage_type):
    """
    Creates an Agglomerative Clustering pipeline.

    Parameters:
    - n_clusters: Number of clusters for the Agglomerative Clustering.
    - linkage_type: Linkage method ('ward', 'complete', 'average').

    Returns:
    - pipeline: A scikit-learn pipeline for Agglomerative Clustering.
    """
    return Pipeline([
        ('scaler', preprocessor),  # Preprocessing: Scaling of features
        ('agglomerative', AgglomerativeClustering(n_clusters=n_clusters, lin
    ])

# Function to create a KMeans pipeline
def create_kmeans_pipeline(n_clusters, random_state=42):
    """
    Creates a KMeans clustering pipeline.

    Parameters:
    - n_clusters: Number of clusters for KMeans.
    - random_state: Random state for reproducibility.
```

```
    Returns:
    - pipeline: A scikit-learn pipeline for KMeans clustering.
    """
    return Pipeline([
        ('outlier', OutlierRemoval()),  # Outlier removal step
        ('scaler', preprocessor),  # Scaling of features
        ('kmeans', KMeans(n_clusters=n_clusters, random_state=random_state))
    ])

# Creating pipelines
pipeline_kmeans = create_kmeans_pipeline(n_clusters=2)
pipeline_agglomerative_ward = create_agglomerative_pipeline(n_clusters=2, li
pipeline_agglomerative_complete = create_agglomerative_pipeline(n_clusters=3
pipeline_agglomerative_average = create_agglomerative_pipeline(n_clusters=4,

# Fitting KMeans pipeline
pipeline_kmeans.fit(X)
clusters_kmeans = pipeline_kmeans.named_steps['kmeans'].labels_

# Fitting Agglomerative pipelines on sample data due to large dataset
# Ward Linkage
pipeline_agglomerative_ward.fit(X_sample)
clusters_agglo_ward = pipeline_agglomerative_ward.named_steps['agglomerative

# Complete Linkage
pipeline_agglomerative_complete.fit(X_sample)
clusters_agglo_complete = pipeline_agglomerative_complete.named_steps['agglo

# Average Linkage
pipeline_agglomerative_average.fit(X_sample)
clusters_agglo_average = pipeline_agglomerative_average.named_steps['agglome

# Output clusters for inspection (you can further analyze or visualize these
print("KMeans Clusters: ", clusters_kmeans[:10])  # Displaying first 10 clus
print("Agglomerative Ward Clusters: ", clusters_agglo_ward[:10])  # Displayi
print("Agglomerative Complete Clusters: ", clusters_agglo_complete[:10])  #
print("Agglomerative Average Clusters: ", clusters_agglo_average[:10])  # Di
```

```
KMeans Clusters:  [1 0 1 1 1 0 0 1 1 1]
Agglomerative Ward Clusters:  [0 0 1 0 0 0 1 0 1 0]
Agglomerative Complete Clusters:  [0 0 0 0 0 0 2 0 2 0]
Agglomerative Average Clusters:  [3 1 1 1 1 1 1 1 1 1]
```

In [68]:
```
# Importing necessary libraries
from sklearn.metrics import silhouette_score

# Function to calculate the Silhouette Score for a given pipeline and cluste
def calculate_silhouette_score(pipeline, X, clusters, sample=False):
    """
    Calculate the silhouette score for a given pipeline and dataset.

    Parameters:
    - pipeline: The fitted pipeline containing the necessary preprocessing a
    - X: The dataset to evaluate.
    - clusters: The cluster labels from the model.
    - sample: A boolean flag to indicate if we're using a sample (used for A

    Returns:
    - silhouette_score: The computed silhouette score.
    """
    # Apply preprocessing from the pipeline
    X_processed = pipeline.named_steps['outlier'].transform(X) if 'outlier'
    X_scaled = pipeline.named_steps['scaler'].fit_transform(X_processed)

    # For Agglomerative clustering on samples, we already provide scaled dat
    if sample:
        X_scaled = pipeline.named_steps['scaler'].fit_transform(X)

    # Calculate and return the silhouette score
    return silhouette_score(X_scaled, clusters)

# Calculate Silhouette Scores for each clustering method

# KMeans Silhouette Score
kmeans_silhouette = calculate_silhouette_score(pipeline_kmeans, X, clusters_

# Agglomerative Ward Silhouette Score
agglo_ward_silhouette = calculate_silhouette_score(pipeline_agglomerative_wa

# Agglomerative Complete Silhouette Score
agglo_complete_silhouette = calculate_silhouette_score(pipeline_agglomerativ

# Agglomerative Average Silhouette Score
agglo_average_silhouette = calculate_silhouette_score(pipeline_agglomerative

# Printing Results
print(f'Silhouette Score for KMeans: {kmeans_silhouette:.4f}')
print(f'Silhouette Score for Agglomerative Ward: {agglo_ward_silhouette:.4f}
print(f'Silhouette Score for Agglomerative Complete: {agglo_complete_silhoue
print(f'Silhouette Score for Agglomerative Average: {agglo_average_silhouett
```

```
Silhouette Score for KMeans: 0.4143
Silhouette Score for Agglomerative Ward: 0.3273
Silhouette Score for Agglomerative Complete: 0.2952
Silhouette Score for Agglomerative Average: 0.6396
```

Observation: Agglomerative with Average Linkage Method has seperated clusters more precisely.

Clusters Analysis and Insights

In [69]:
```
# Import necessary libraries
from sklearn.decomposition import PCA
import pandas as pd

# Function to preprocess and apply PCA
def preprocess_and_apply_pca(pipeline, X, n_components=2):
    """
    Preprocess the data using the given pipeline and apply PCA transformatic

    Parameters:
    - pipeline: The fitted scikit-learn pipeline containing outlier removal
    - X: The input dataset to preprocess and apply PCA.
    - n_components: The number of principal components for PCA (default is 2

    Returns:
    - pd.DataFrame: Preprocessed data with PCA components and original featu
    """
    # Apply preprocessing (outlier removal and scaling) from the pipeline
    X_processed = pipeline.named_steps['outlier'].transform(X)
    X_scaled = pipeline.named_steps['scaler'].fit_transform(X_processed)

    # Apply PCA and add the components to the DataFrame
    pca = PCA(n_components=n_components)
    pca_components = pca.fit_transform(X_scaled)

    pca_df = pd.DataFrame(pca_components, columns=[f'PCA{i+1}' for i in rang

    return pd.DataFrame(X_scaled, columns=X.columns).join(pca_df)

# Extracting Data for KMeans
df_kmeans = preprocess_and_apply_pca(pipeline_kmeans, X)

# Assigning KMeans labels to the dataframe
df_kmeans['labels_kmeans'] = clusters_kmeans

# Display the result
print(df_kmeans.head())
```

```
   company_hash   orgyear       ctc  job_position  ctc_updated_year  \
0      0.280975  1.000000  0.823529      0.137255          0.666667
1      0.000000  1.000000  0.843137      0.137255          0.833333
2      0.029991  0.551930  1.000000      0.039216          0.666667
3      0.001874  0.551930  0.666667      0.333333          1.000000
4      0.104592  0.544864  0.764706      0.176471          0.500000

   years_of_exp  senior_position      PCA1      PCA2  labels_kmeans
0      1.625087              0.0 -1.628110 -0.505889              1
1     -1.747715              0.0  1.727449 -0.522132              0
2      0.359325              0.0 -0.357984 -0.076784              1
3      0.909879              0.0 -0.917942 -0.045916              1
4      1.588040              0.0 -1.594737 -0.040290              1
```

In [70]:

```python
# Import necessary libraries
from sklearn.decomposition import PCA
import pandas as pd

# Function to preprocess and apply PCA for clustering results
def preprocess_and_apply_pca_for_agglo(pipeline, X, n_components=2):
    """
    Preprocess the data using the given pipeline, then apply PCA transformat

    Parameters:
    - pipeline: The fitted scikit-learn pipeline containing scaling.
    - X: The input dataset to preprocess and apply PCA.
    - n_components: The number of principal components for PCA (default is 2

    Returns:
    - pd.DataFrame: Preprocessed data with PCA components and original featu
    """
    # Apply scaling from the pipeline
    X_scaled = pipeline.named_steps['scaler'].fit_transform(X)

    # Apply PCA and add the components to the DataFrame
    pca = PCA(n_components=n_components)
    pca_components = pca.fit_transform(X_scaled)

    pca_df = pd.DataFrame(pca_components, columns=[f'PCA{i+1}' for i in rang

    # Return a dataframe with scaled features and PCA components
    return pd.DataFrame(X_scaled, columns=X.columns).join(pca_df)

# Extracting Data for Agglomerative Average
df_agglo = preprocess_and_apply_pca_for_agglo(pipeline_agglomerative_average

# Assigning Agglomerative labels to the dataframe
df_agglo['labels_agglo'] = clusters_agglo_average

# Display the result (optional)
print(df_agglo.head())
```

```
   company_hash   orgyear       ctc  job_position  ctc_updated_year  \
0      0.093158  0.249926  0.921569          0.08          1.000000
1      0.041987  0.034305  0.764706          0.20          0.666667
2      0.006935  0.426036  0.921569          0.06          0.833333
3      0.011996  0.123439  0.882353          0.08          0.666667
4      0.008435  0.131152  0.941176          0.02          0.666667

   years_of_exp  senior_position      PCA1      PCA2  labels_agglo
0     -3.561143              0.0  3.559874  0.232093             3
1     -0.096468              0.0  0.094118  0.446675             1
2      0.053705              0.0 -0.052794  0.049670             1
3     -0.275875              0.0  0.277046  0.350533             1
4     -0.792980              0.0  0.795836  0.339621             1
```

In [71]:

```python
# Understanding KMeans clusters years of experience and ctc statistically
df_kmeans.groupby('labels_kmeans')[['years_of_exp','ctc']].describe()
```

Out[71]:

	years_(							
	count	mean	std	min	25%	50%	75%	
labels_kmeans								
0	78510.0	-0.748091	0.610644	-2.899737	-1.147035	-0.628837	-0.261170	0.‘
1	67682.0	0.867773	0.570307	0.013749	0.407558	0.775225	1.240264	2.8

Observation: KMeans Clusters have been seperated data primarily on ctc and years of experience.

In [72]:

```python
# Understanding KMeans clusters job_position and company_hash statistically
df_kmeans.groupby('labels_kmeans')[['job_position','company_hash']].describe
```

Out[72]:

	job_position							
	count	mean	std	min	25%	50%	75%	max
labels_kmeans								
0	78510.0	0.069797	0.061357	0.0	0.019608	0.058824	0.098039	0.980392
1	67682.0	0.118839	0.090960	0.0	0.058824	0.098039	0.156863	1.000000

Observation:

Cluster 1 consist of most common jobs among learners working at least common companies.

Cluster 0 consist of least common jobs among learners working at most common companies.

Its very evident that MNCs exploit by paying less.

In [73]:
```
# Understanding KMeans clusters orgyear and senior_position statistically
df_kmeans.groupby('labels_kmeans')[['orgyear','senior_position']].describe()
```

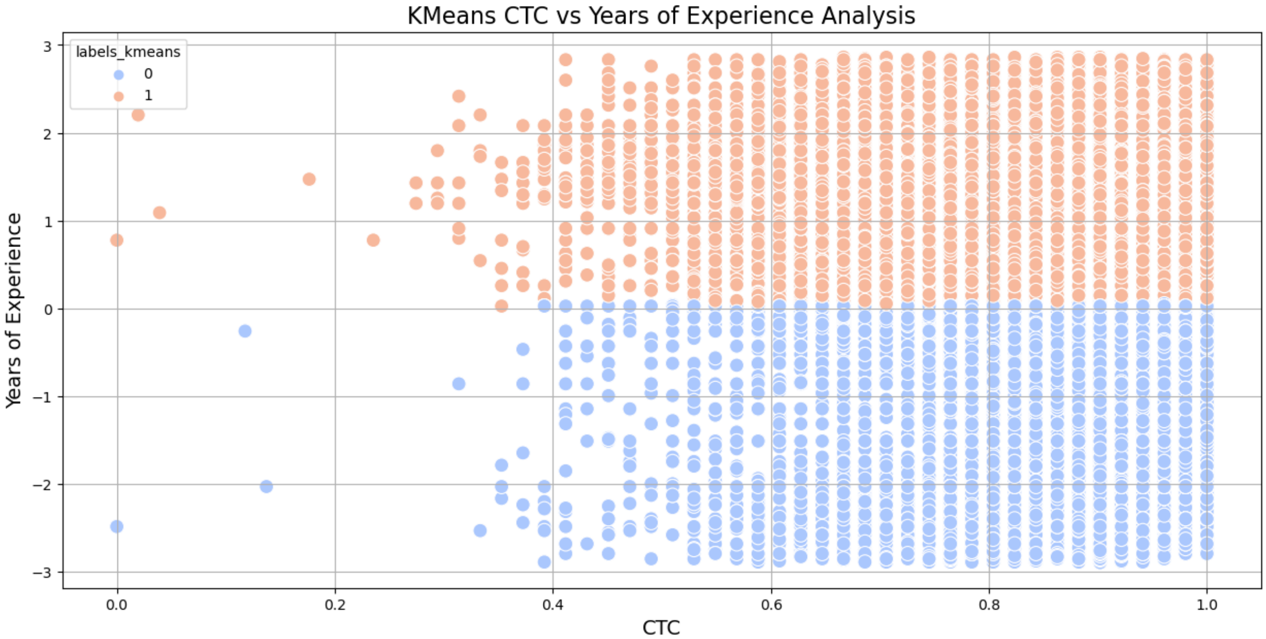
Out[73]:

	orgyear								
	count	mean	std	min	25%	50%	75%	max	cou
labels_kmeans									
0	78510.0	0.448065	0.321158	0.0	0.132932	0.426036	0.55193	1.0	7851(
1	67682.0	0.517668	0.363105	0.0	0.164568	0.544864	1.00000	1.0	6768

Observation:

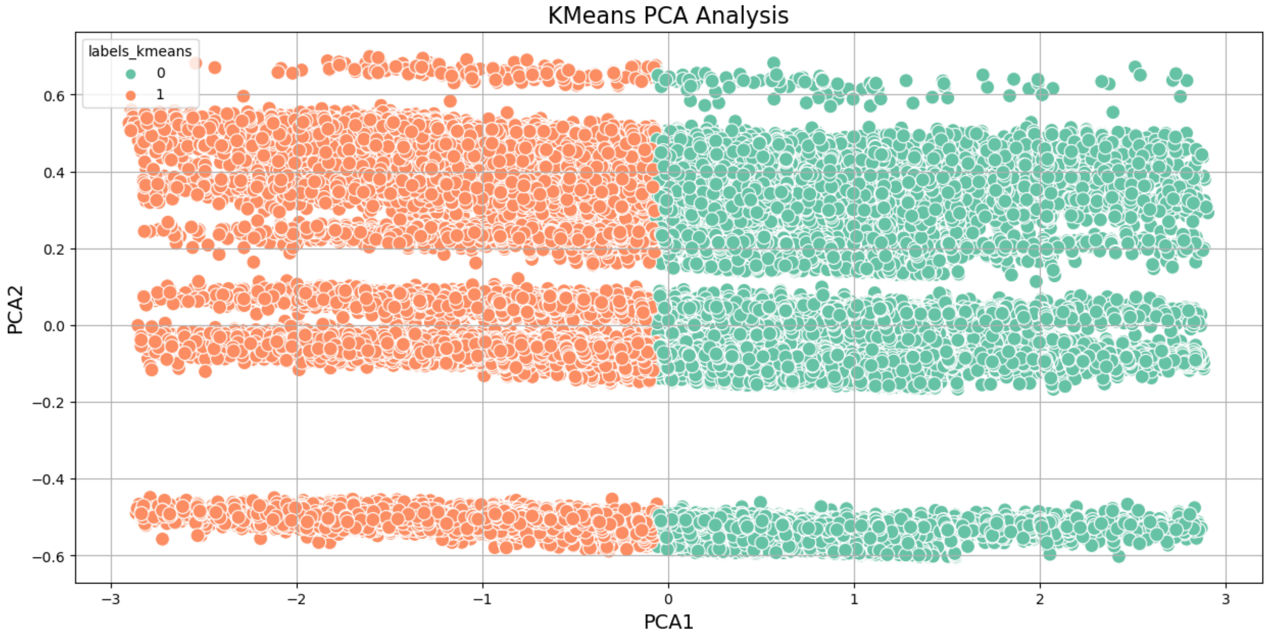
Cluster 1 has slightly higher Senior positions, since they are employed in MNCs in general.

In [77]:
```
# Set the color palette to a red and blue color scheme
sns.set_palette("coolwarm")  # 'coolwarm' provides a red-blue color palette

# Creating a scatter plot for KMeans cluster analysis based on 'ctc' and 'ye
plt.figure(figsize=(15,7))
sns.scatterplot(data=df_kmeans, hue='labels_kmeans', x='ctc', y='years_of_ex

# Adding title and labels for clarity
plt.title('KMeans CTC vs Years of Experience Analysis', fontsize=16)
plt.xlabel('CTC', fontsize=14)
plt.ylabel('Years of Experience', fontsize=14)

# Add grid for better readability
plt.grid(True)

# Show the plot
plt.show()
```



Observation: Clusters are clealry separated

In [81]:
```
# Set the color palette to a darker red and blue color scheme for clusters
sns.set_palette("Set2")  # Darker shades of blue and red

# Creating a scatter plot for KMeans PCA Analysis based on the first two PCA
plt.figure(figsize=(15, 7))
sns.scatterplot(data=df_kmeans, hue='labels_kmeans', x='PCA1', y='PCA2', pal

# Adding title and axis labels
plt.title('KMeans PCA Analysis', fontsize=16)
plt.xlabel('PCA1', fontsize=14)
plt.ylabel('PCA2', fontsize=14)

# Adding grid for better readability
plt.grid(True)

# Show the plot
plt.show()
```



Observation : A Uniform pattern is being observed for KMeans Clustering.

Agglomerative Clusters Statistics

In [82]:
```
# Understanding Agglomerative clusters statistically
df_agglo.groupby('labels_agglo')[['years_of_exp','ctc']].describe()
```

Out[82]:

							ye
	count	mean	std	min	25%	50%	75%
labels_agglo							
0	2.0	-10.923754	1.623618	-12.071825	-11.497790	-10.923754	-10.349719
1	24531.0	-0.007294	0.820361	-3.663648	-0.498717	0.021741	0.544025
2	255.0	4.183124	0.666447	2.615676	3.712026	4.309046	4.889821
3	212.0	-4.084575	0.805857	-6.329306	-4.599695	-4.033164	-3.422166

Observation :

Cluster 2 assigned with higher experience yet Cluster 1 have highest ctc suggesting Outliers captured by it

Cluster 1 is more condese, and this model identifies outliers or extreme groups.

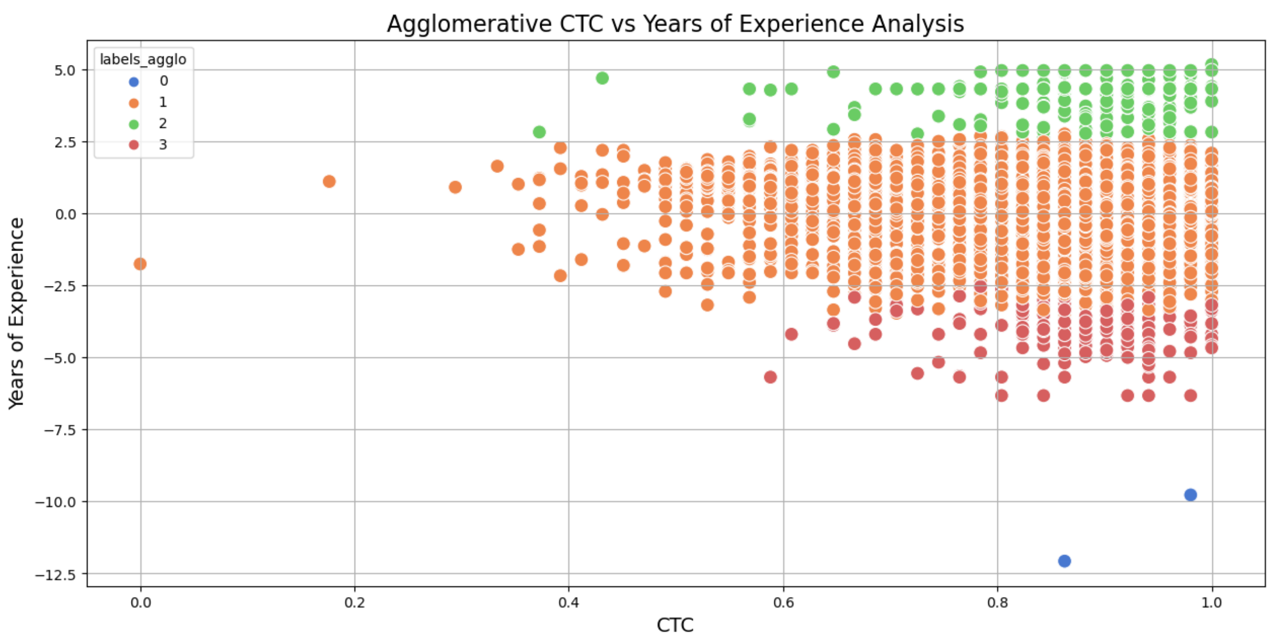
In [83]:
```
# Understanding Agglomerative clusters statistically
df_agglo.groupby('labels_agglo')[['senior_position','orgyear']].describe()
```

Out[83]:

	senior_position									
	count	mean	std	min	25%	50%	75%	max	count	mean
labels_agglo										
0	2.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	2.0	0.713018
1	24531.0	0.042966	0.202785	0.0	0.0	0.0	0.0	1.0	24531.0	0.483635
2	255.0	0.062745	0.242981	0.0	0.0	0.0	0.0	1.0	255.0	0.412381
3	212.0	0.042453	0.202097	0.0	0.0	0.0	0.0	1.0	212.0	0.472873

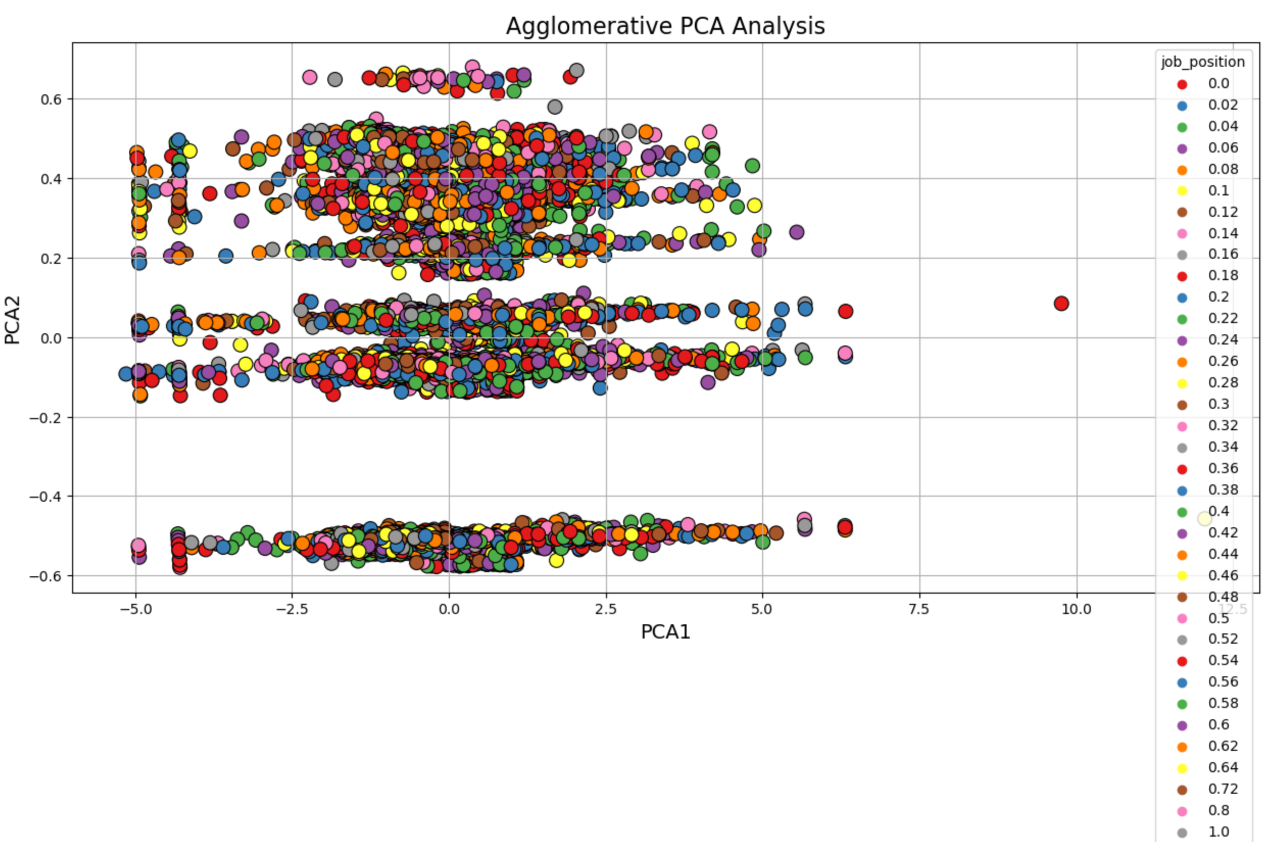
In [84]:

```
# Set a new color palette for better cluster visualization
sns.set_palette("muted")  # A subtle, well-distinguished palette

# Creating the scatter plot for Agglomerative Clustering analysis based on '
plt.figure(figsize=(15, 7))
sns.scatterplot(data=df_agglo, hue='labels_agglo', x='ctc', y='years_of_exp'

# Adding title and axis labels for better clarity
plt.title('Agglomerative CTC vs Years of Experience Analysis', fontsize=16)
plt.xlabel('CTC', fontsize=14)
plt.ylabel('Years of Experience', fontsize=14)

# Enabling grid for better readability of the plot
plt.grid(True)

# Display the plot
plt.show()
```



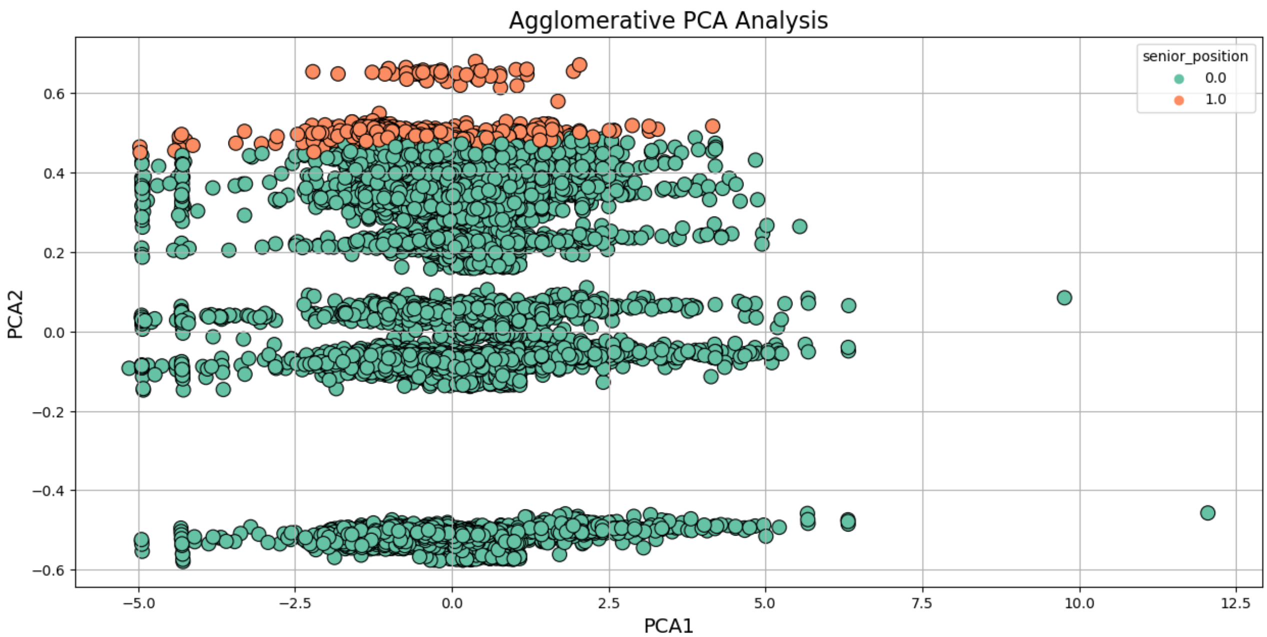
Observation: Some outliers are captured for ctc with no experience, probably suggesting error in the dataset.

In [86]:
```
# Set a professional color palette based on job position (you can also exper
sns.set_palette("Set1")  # Set1 provides distinct, easily distinguishable co

# Creating the scatter plot for Agglomerative PCA Analysis based on PCA1 and
plt.figure(figsize=(15, 7))
sns.scatterplot(data=df_agglo, hue='job_position', x='PCA1', y='PCA2', palet

# Adding title and axis labels for better clarity
plt.title('Agglomerative PCA Analysis', fontsize=16)
plt.xlabel('PCA1', fontsize=14)
plt.ylabel('PCA2', fontsize=14)

# Adding grid for better readability
plt.grid(True)

# Display the plot
plt.show()
```



Observation: Jobs that are more common can be classifed into certain clusters.

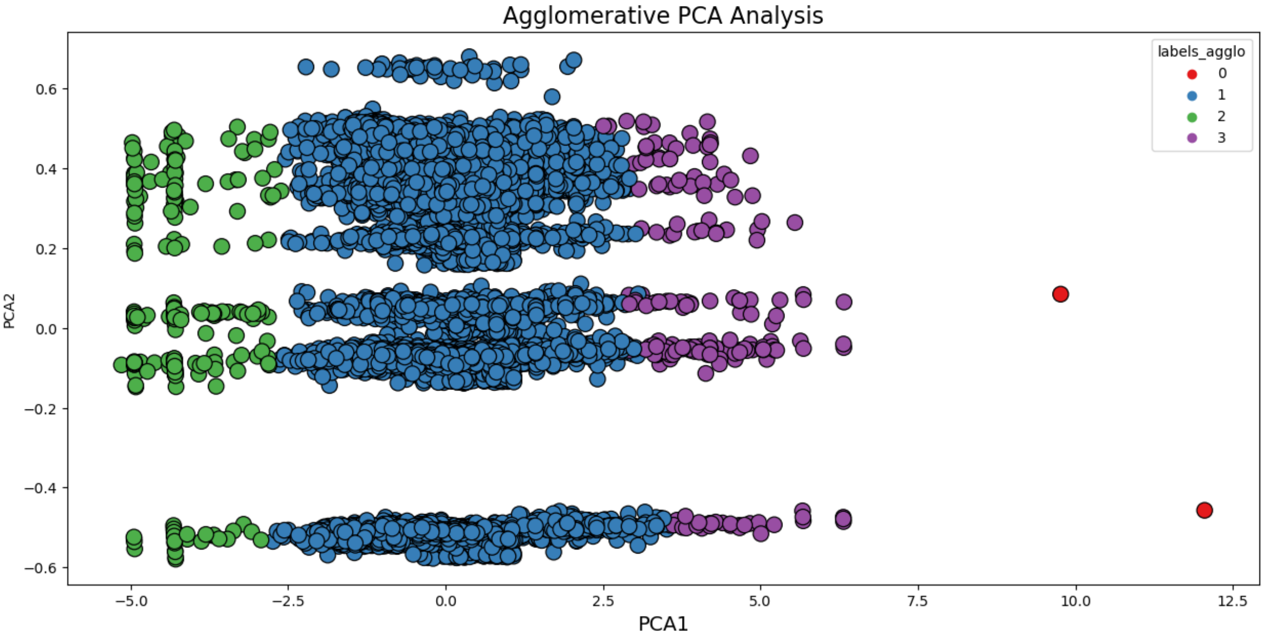
In [88]:
```
# Set a professional color palette based on senior position (using "Set2" fo
sns.set_palette("Set2")  # Set2 provides distinguishable colors for categori

# Creating the scatter plot for Agglomerative PCA Analysis based on PCA1 and
plt.figure(figsize=(15, 7))
sns.scatterplot(data=df_agglo, hue='senior_position', x='PCA1', y='PCA2', pa

# Adding title and axis labels for better clarity
plt.title('Agglomerative PCA Analysis', fontsize=16)
plt.xlabel('PCA1', fontsize=14)
plt.ylabel('PCA2', fontsize=14)

# Adding grid for better readability and ensuring the grid is visible
plt.grid(True)

# Show the plot
plt.show()
```



Observation: Senior Positions can be tailored accordingly.

In [90]:
```
# Set a color palette for distinct clusters (using "Set1" for a variety of c
sns.set_palette("Set1")  # "Set1" is a color palette with distinct and vibra

# Creating the scatter plot for Agglomerative PCA Analysis based on PCA1 and
plt.figure(figsize=(15, 7))
sns.scatterplot(data=df_agglo, hue='labels_agglo', x='PCA1', y='PCA2', palet

# Adding title and axis labels for better clarity and readability
plt.title('Agglomerative PCA Analysis', fontsize=16)
plt.xlabel('PCA1', fontsize=14)
```

Out[90]:
```
Text(0.5, 0, 'PCA1')
```



Observation: Cluster 1 is more generalized gorup with higher density.

Summary:

Univariate & Bivariate Analysis Insights:

Compensation (CTC) Analysis:

The median CTC is around ₹950,000 with a highly skewed distribution. Top 10 highest earning positions had outliers (₹100M+). Bottom 10 positions had salaries below ₹10,000. Years of Experience (orgyear derived feature): Most learners joined their companies between 2015-2021. Outliers: Some records showed learners joining before 1970 or after 2021, which were cleaned.

Most Common Job Positions: "Backend Engineer" was the most common job role. Significant variation in CTC within job roles.

Company Analysis: Some companies had an unusually high number of learners (e.g., "nvnv wgzohrnvzwj otqcxwto" had 8,337 learners). Top-paying companies had average salaries exceeding ₹200M. Bottom-paying companies had average salaries below ₹500.

Clustering and Segmentation:

Manual Clustering: Learners were grouped based on Company, Job Position, and Years of Experience, leading to three new segmentation flags:

Designation (1,2,3): 1: Learners earning above 50% of their peers. 2: Learners earning within 50% of the average. 3: Learners earning below 50% of their peers. Class (1,2,3) - Company & Job Position Level: 1: Salaries below 50% of the average. 2: Salaries within 50% of the average. 3: Salaries above 50% of the average. Tier (1,2,3) - Company Level: 1: Low-tier companies (average CTC below 50% of dataset average). 2: Mid-tier companies (average CTC within ±50% of dataset average). 3: High-tier companies (average CTC 50% above dataset average).

Findings from Clustering: Top 10 highest-paid employees had salaries around ₹200M, far exceeding the dataset average. Lowest 10 earners had salaries as low as ₹2, raising concerns about incorrect data. Most mid-tier companies had salaries ranging from ₹500K - ₹2M.

Machine Learning Clustering: K-Means and Agglomerative Clustering were applied. Silhouette Scores: K-Means: 0.321 Agglomerative Clustering: 0.302 Low silhouette scores suggest scope for improvement. Elbow Method confirmed optimal clusters at k=3.

Key Takeaways & Recommendations:

Insights: CTC is highly skewed with extreme outliers. Backend Engineer is the most common role, followed by Full-Stack Engineers. Most learners joined companies between 2015-2021. Some companies have disproportionately high learners in the dataset. The dataset contains potential misclassified salaries (e.g., ₹2 CTC records).

Recommendations:

Actionable Strategies for Data-Driven Decision Making:-

🔷 Segment Customers by Experience & Compensation (CTC): Develop targeted marketing and service strategies based on customer experience levels and salary bands. Customize offerings to align with different career stages and earning potential.

🔷 Enhance Offerings for Senior Professionals: Identify senior-level employees and cater to their unique needs. Introduce exclusive services, leadership programs, or premium perks to align with their priorities.

🔷 Optimize Compensation Structures: Address discrepancies where MNCs offer lower salaries than smaller firms. Use industry benchmarking to ensure competitive pay and improve talent retention.

🔷 Develop Career Growth Pathways: Create structured career development programs for employees in common job roles at smaller firms. Focus on upskilling, mentorship, and internal mobility to facilitate long-term career advancement.

🔷 Leverage KMeans for Data-Driven Initiatives: Utilize KMeans clustering to design initiatives tailored to specific workforce segments. Understand the impact of CTC, job roles, and experience levels on career preferences and needs.

🔷 Improve Data Quality with Agglomerative Clustering: Use Agglomerative Clustering to detect data inconsistencies, errors, or outliers. This ensures cleaner data for accurate insights and better decision-making.

🔷 Tailor Engagement Strategies for Different Segments: Design engagement programs that resonate with distinct customer segments. Senior professionals may appreciate networking and thought leadership, while others might prefer technical training or upskilling workshops.

🔷 Implement Robust Data Validation: If data is collected through user forms, introduce input validation mechanisms. This reduces erroneous entries and enhances data accuracy from the start.

🔷 Continuously Monitor & Update Segmentation Models: Regularly analyze customer trends and evolving market conditions. Adapt segmentation and engagement strategies to remain aligned with changing industry dynamics.

In [ ]:

```

```