

1. Introduction

❓ What is Scaler?

Scaler is a cutting-edge online tech university (tech-versity) designed to upskill software and data science professionals. It offers rigorous, live learning programs delivered by seasoned industry experts and leaders, focusing on modern curriculums aligned with the latest technological advancements. Scaler empowers learners to bridge skill gaps, advance their careers, and gain exposure to real-world industry scenarios through personalized mentorship and a structured educational approach.

💡 Issue at Hand

Scaler caters to a diverse learner base with varying job roles, industry backgrounds, and levels of experience. This diversity presents challenges in profiling learners effectively to deliver a personalized learning experience. Identifying and understanding these profiles through data-driven clustering methods is critical for Scaler to enhance learner engagement, retention, and satisfaction. Without a precise approach, Scaler risks providing generalized solutions that may not address individual learner needs effectively.

🎯 Objective

- Leverage clustering techniques to categorize learners into meaningful profiles based on their job roles, companies, experience, and salary structures.
- Analyze these profiles to uncover actionable insights that will aid in tailoring Scaler's curriculum, mentorship programs, and overall learner journey.
- Provide recommendations to improve learner outcomes and satisfaction while boosting Scaler's impact as an ed-tech leader.

🎯 Key Goal

- Develop a data-driven, personalized approach to enhance learner experiences by clustering individuals with similar characteristics.
- Enable Scaler to refine its course offerings, mentorship programs, and marketing strategies by understanding distinct learner groups and their professional trajectories.
- Support Scaler in becoming a leader in learner-centric education by addressing unique skill gaps and aligning learning paths with industry trends and demands.

📋 Features of the dataset:

| Feature | Description |
|------------------|---|
| Unnamed: 0 | The index of the Dataset |
| company_hash | An anonymized identifier indicating the current employer of the learner. |
| email_hash | An anonymized identifier representing the email of the learner. |
| orgyear | Represents the year the learner began employment at the current company. |
| ctc | Current Compensation to the Company (CTC) of the learner. |
| job_position | Represents the job profile or role of the learner within their company. |
| ctc_updated_year | The year in which the learner's CTC was most recently updated. This could be due to yearly increments, promotions, or other factors. |

2. Exploratory Data Analysis

In [145]: # Importing Libraries Needed

```
import pandas as pd
import numpy as np
import seaborn as sns
sns.set_style(style='whitegrid')
from scipy import stats
import matplotlib.pyplot as plt

import re

import warnings
warnings.filterwarnings("ignore")

# Machine Learning Libraries

from sklearn.impute import KNNImputer
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.neighbors import NearestNeighbors, LocalOutlierFactor
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
```

```

from sklearnex import patch_sklearn
patch_sklearn()

import umap

from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import fastcluster

from IPython.display import clear_output
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from collections import Counter

```

Intel(R) Extension for Scikit-learn* enabled (<https://github.com/intel/scikit-learn-intelex>)

Dataset Link - [Scaler_Clustering](#)

In [146]: # Importing Data Set

```

df = pd.read_csv(r'M:\Business Cases\10 - Scaler\scaler_clustering.csv')
df.head()

```

| | Unnamed: 0 | company_hash | | email_hash | orgyear | ctc | job_position | ctc_updated_year |
|---|---------------|--------------------------|--|------------|---------|---------|--------------------|------------------|
| 0 | 0 | atrgxnnt xzaxv | 6de0a4417d18ab14334c3f43397fc13b30c35149d70c05... | 2016.000 | 1100000 | | Other | 2020.000 |
| 1 | 1 | qtrxvzwt xzegwgbbrxbxnta | b0aaaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10... | 2018.000 | 449999 | | FullStack Engineer | 2019.000 |
| 2 | 2 | ojzwnvwnxw vx | 4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9... | 2015.000 | 2000000 | Backend | Engineer | 2020.000 |
| 3 | 3 | ngpgutaxv | effdede7a2e7c2af664c8a31d9346385016128d66bbc58... | 2017.000 | 700000 | Backend | Engineer | 2019.000 |
| 4 | 4 | qxen sqghu | 6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520... | 2017.000 | 1400000 | | FullStack Engineer | 2019.000 |

In [147]: df.shape

Out[147]: (205843, 7)

🔍 Insights

Number of Rows: 205843

Number of Columns: 7

In [148]: df.drop(['Unnamed: 0'], axis = 1, inplace = True)

In [149]: df.info()

```

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

```

In [150]: # Changing the display format

```

pd.options.display.float_format = '{:.3f}'.format

```

In [151]: df.describe()

| | orgyear | ctc | ctc_updated_year |
|-------|------------|----------------|------------------|
| count | 205757.000 | 205843.000 | 205843.000 |
| mean | 2014.883 | 2271685.042 | 2019.628 |
| std | 63.571 | 11800914.440 | 1.325 |
| min | 0.000 | 2.000 | 2015.000 |
| 25% | 2013.000 | 530000.000 | 2019.000 |
| 50% | 2016.000 | 950000.000 | 2020.000 |
| 75% | 2018.000 | 1700000.000 | 2021.000 |
| max | 20165.000 | 1000150000.000 | 2021.000 |

🔍 Insights

1. The Column `Unnamed: 0` has been dropped, since it won't be impacting the Clustering.
2. There are About 2 Lakh entries in the Dataset.
3. The Minimum Value in `CTC` is **2.0**, this has to either be Error or an Outlier.
4. The Minimum Value in `orgyear` is **0.0**, This must be an Error.
5. The Maximum Value in `orgyear` is **20165**, This must be an Error.

Duplicate Detection

```
In [152]: df.duplicated().value_counts()
```

```
Out[152]: False    205809
True      34
Name: count, dtype: int64
```

Insights

- There are **34** Duplicate entries in the Dataset, Which are to be Dropped.

```
In [153]: # Dropping Duplicate Rows
```

```
df.drop_duplicates(inplace = True)
```

```
In [154]: # Check Duplicate Rows
```

```
df.duplicated().value_counts()
```

```
Out[154]: False    205809
Name: count, dtype: int64
```

Basic Data Cleaning and Exploration

Handling Missing Values in the Data Set

```
In [155]: # Checking for Missing Values
```

```
df.isnull().sum()
```

```
Out[155]: company_hash      44
email_hash          0
orgyear            86
ctc                0
job_position       52548
ctc_updated_year   0
dtype: int64
```

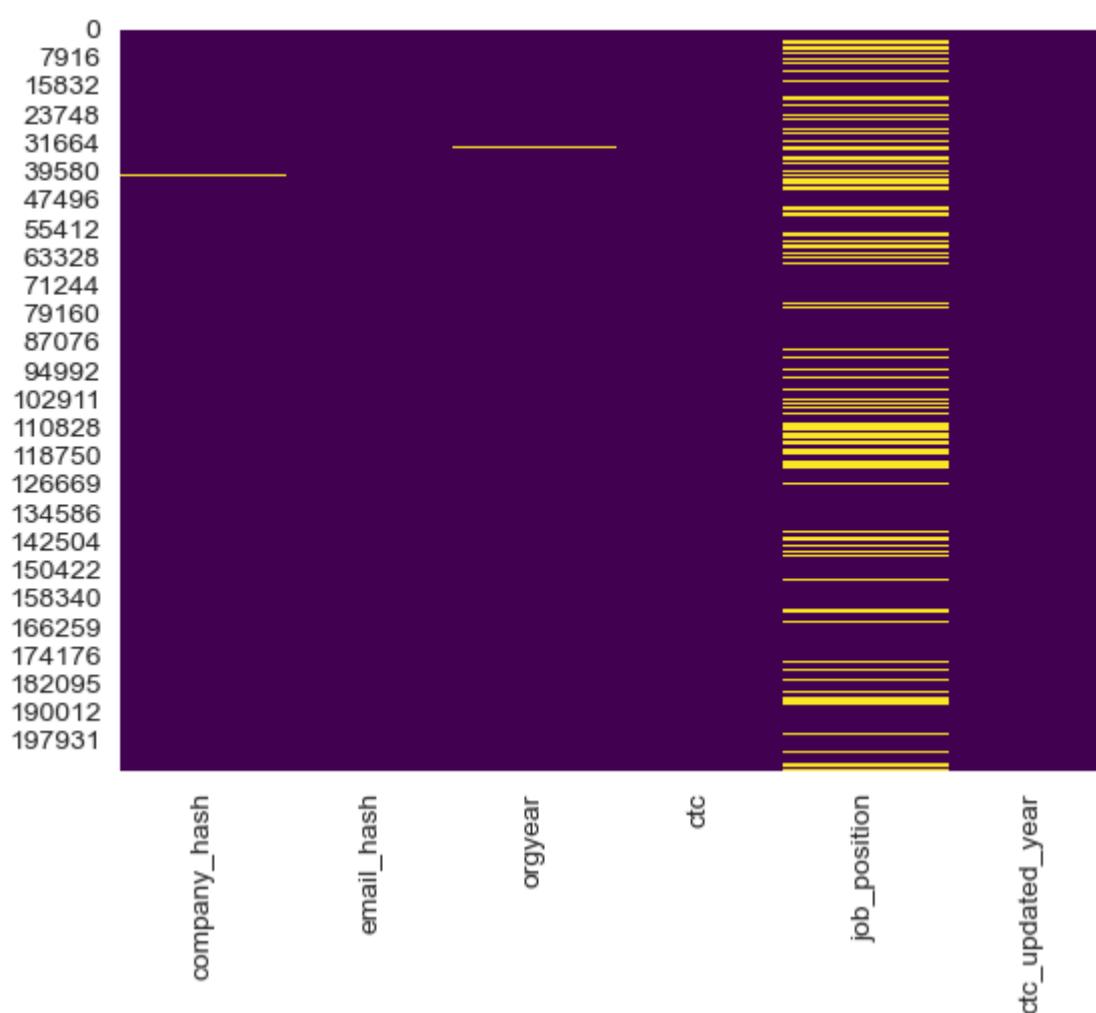
```
In [156]: # Percentage of Null Values
```

```
df.isnull().sum()/df.shape[0]*100
```

```
Out[156]: company_hash      0.021
email_hash          0.000
orgyear            0.042
ctc                0.000
job_position       25.532
ctc_updated_year   0.000
dtype: float64
```

```
In [157]: sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
plt.show()
```

```
Out[157]: <Axes: >
```



🔍 Insights

3 Columns have Missing Values:

- `company_hash` and `orgyear` have **Less than 1%**
- `job_position` has **about 25 %**

Handling Missing Values in `company_hash` and `orgyear` Columns Using Mean/Mode Imputation

```
In [158]: # Impute missing values for `company_hash` using Mode (most frequent value)
mode_company_hash = df["company_hash"].mode()[0] # Get the most frequent value
df["company_hash"] = df["company_hash"].fillna(mode_company_hash)

# Impute missing values for `orgyear` using Mean
mean_orgyear = df["orgyear"].mean() # Calculate the mean value
df["orgyear"] = df["orgyear"].fillna(mean_orgyear)
```

```
In [159]: df.isnull().sum()
```

```
Out[159]: company_hash      0
email_hash        0
orgyear          0
ctc              0
job_position     52548
ctc_updated_year 0
dtype: int64
```

Handling Missing Values in `job_position` Column

```
In [160]: df['job_position'].value_counts(dropna = False)
```

```
Out[160]: job_position
NaN                      52548
Backend Engineer          43546
FullStack Engineer         24711
Other                      18071
Frontend Engineer          10417
...
Compliance auditor          1
91                         1
Senior Software Development Engineer (Backend) 1
Messenger come driver      1
Android Application developer 1
Name: count, Length: 1017, dtype: int64
```

```
In [161]: df['job_position'].nunique()
```

```
Out[161]: 1016
```

```
In [162]: # Replace all types of null-like values with NaN first
df["job_position"].replace(["NaN", "NaN ", "nan", None], pd.NA, inplace=True)

# Fill null values in 'job_position' with "unknown"
df["job_position"] = df["job_position"].fillna("unknown")
```

```

# Convert job_position to lowercase
df["job_position"] = df["job_position"].str.lower()

# Replace hyphens with spaces
df["job_position"] = df["job_position"].str.replace("-", " ", regex=False)

# Remove all non-alphanumeric characters except spaces
df["job_position"] = df["job_position"].str.replace(r"[^a-zA-Z0-9 ]", "", regex=True)

# Remove parentheses
df["job_position"] = df["job_position"].str.replace(r"\)", "", regex=True)
df["job_position"] = df["job_position"].str.replace(r"\(", " ", regex=True)

# Sequential replacements for specific patterns
df["job_position"] = (
    df["job_position"]
        .str.replace("ii", "2", regex=False)
        .str.replace(r" i\b", " 1", regex=True)
        .str.replace("iii", "3", regex=False)
        .str.replace("sr ", "senior ", regex=False)
        .str.replace("jr ", "junior ", regex=False)
)

# Additional sequential replacements for common patterns
df["job_position"] = (
    df["job_position"]
        .str.replace("front end", "frontend", regex=False)
        .str.replace("back end", "backend", regex=False)
        .str.replace("full stack", "fullstack", regex=False)
        .str.replace("sde2", "sde 2", regex=False)
        .str.replace("sde3", "sde 3", regex=False)
        .str.replace("se4", "se 4", regex=False)
        .str.replace("2i", "2", regex=False)
        .str.replace(r"\br d\b", "rd", regex=True)
        .str.replace(r"engineer|engineers|engineer|eingenieur", "engineer", regex=True)
        .str.replace(r"Engeneering", "Engineering", regex=True)
        .str.replace(r"engg\b", "engineer", regex=True)
        .str.replace("applications", "application", regex=False)
        .str.replace(r"dev\b", "developer", regex=True)
        .str.replace(r"devloper|develloper", "developer", regex=True)
        .str.replace("consulant", "consultant", regex=False)
        .str.replace(r"others\b", "other", regex=True)
        .str.replace(" of ", " ", regex=False)
        .str.replace("tech ", "technical ", regex=False)
        .str.replace(r"development(\w+)", r"development \1", regex=True)
)

# Replace double spaces with single space and strip leading/trailing spaces
df["job_position"] = df["job_position"].str.replace("  ", " ", regex=False).str.strip()

```

In [163]: `df.isnull().sum()`

Out[163]:

| | |
|------------------|---|
| company_hash | 0 |
| email_hash | 0 |
| orgyear | 0 |
| ctc | 0 |
| job_position | 0 |
| ctc_updated_year | 0 |
| dtype: int64 | |

In [164]: `df['job_position'].value_counts(dropna = False)`

Out[164]:

| | |
|--|-------|
| job_position | |
| unknown | 52548 |
| backend engineer | 43546 |
| fullstack engineer | 25982 |
| other | 18073 |
| frontend engineer | 10418 |
| ... | |
| phd student | 1 |
| ays | 1 |
| principal product engineer | 1 |
| senior director engineering | 1 |
| azure data factory | 1 |
| Name: count, Length: 841, dtype: int64 | |

In [165]: `df['job_position'].nunique()`

Out[165]: 841

Spell Check

```

# The Following Code is taken from https://norvig.com/spell-correct.html For Spell Checking

# Split job positions into individual words, explode them, and count occurrences
words = df["job_position"].str.split(" ").explode()
WORDS = Counter(words.to_list())

# Define helper functions for spelling correction and word cleaning
def P(word, N=sum(WORDS.values())):
    """Probability of `word`."""

```

```

    return WORDS[word] / N

def correction(word):
    """Most probable spelling correction for a word."""
    return max(candidates(word), key=P)

def candidates(word):
    """Generate possible spelling corrections for a word."""
    return known([word]) or known(edits1(word)) or known(edits2(word)) or [word]

def known(words):
    """Subset of `words` that appear in the dictionary of WORDS."""
    return set(w for w in words if w in WORDS)

def edits1(word):
    """All edits that are one edit away from `word`."""
    letters = 'abcdefghijklmnopqrstuvwxyz'
    splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes = [L + R[1:] for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R) > 1]
    replaces = [L + c + R[1:] for L, R in splits if R for c in letters]
    inserts = [L + c + R for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    """All edits that are two edits away from `word`."""
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))

def correct_words(sentence):
    """Correct all words in a sentence."""
    return " ".join([correction(word) for word in sentence.split()])

def clean_jobs(word):
    """Clean individual words by replacing short or numeric values with 'unknown'."""
    if len(word) <= 1:
        return "unknown"
    try:
        float(word)
        return "unknown"
    except ValueError:
        return word

def replace_words(sentence):
    """Replace specific job-related phrases."""
    unemployed = [
        "na", "no", "no job", "not applicable", "not employed",
        "not working", "null", "none", "now iam not working waiting for job"
    ]
    if sentence.lower() in unemployed:
        return "unemployed"
    else:
        return sentence

# Apply cleaning and corrections to job_position column
df["job_position"] = df["job_position"].apply(replace_words)
df["job_position"] = df["job_position"].apply(correct_words)
df["job_position"] = df["job_position"].apply(clean_jobs)

```

In [167]: `df.isnull().sum()`

Out[167]:

| | |
|------------------|---|
| company_hash | 0 |
| email_hash | 0 |
| orgyear | 0 |
| ctc | 0 |
| job_position | 0 |
| ctc_updated_year | 0 |

dtype: int64

Manually Fixing the Errors in orgyear, ctc, job_position Columns

orgyear

In [168]: `df.info()`

```

<class 'pandas.core.frame.DataFrame'>
Index: 205809 entries, 0 to 205842
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   company_hash     205809 non-null   object 
 1   email_hash       205809 non-null   object 
 2   orgyear          205809 non-null   float64
 3   ctc              205809 non-null   int64  
 4   job_position     205809 non-null   object 
 5   ctc_updated_year 205809 non-null   float64
dtypes: float64(2), int64(1), object(3)
memory usage: 11.0+ MB

```

```
In [169]: # Converting `orgyear` and `ctc_updated_year` to int32 format
df["orgyear"] = df["orgyear"].astype("int32")
df["ctc_updated_year"] = df["ctc_updated_year"].astype("int32")
```

```
In [170]: df.isna().sum()
```

```
Out[170]: company_hash      0
email_hash        0
orgyear          0
ctc              0
job_position     0
ctc_updated_year 0
dtype: int64
```

```
In [171]: df['orgyear'].value_counts().sort_index()
```

```
Out[171]: orgyear
0            17
1             2
2             3
3             6
4             1
..
2101          1
2106          1
2107          1
2204          1
20165         2
Name: count, Length: 77, dtype: int64
```

💡 Insights

- Based on the Above Result, there are Errors in the `orgyear` Column.
- We will fix this by keeping Upper and Lower Limits and Anything that fall beyond these limits will be Replaced using `ctc_updated_year`

```
In [172]: # Lower limit for orgyear
lower_limit_orgyear = 1970

# view orgyear values outside the limits
df[(df['orgyear'] < lower_limit_orgyear)]
```

| Out[172]: | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year |
|-----------|---------------------------------------|---|---------|---------|------------------------|------------------|
| 13424 | 9xntwyzgrgsj | 854ff163ded87211b944dfcaebdcf9e8efa45defc9582f... | 0 | 700000 | unknown | 2021 |
| 13698 | oxbtzoz | 4a64fdec422e657b175d5dd914b91e0df7c78ec7716bfe... | 208 | 500000 | unknown | 2020 |
| 15323 | nvnv wgzohrnvzwj otqcxwto nwo | 437fa88cd652351931ef679e6b074aa91acb384ef193dd... | 209 | 300000 | unknown | 2021 |
| 17139 | sgxmxmg | 6db474dae5093f975e43697cd77ac5a486248c26235778... | 206 | 1500000 | unknown | 2021 |
| 30335 | avqn xzzgcvnxgz | d767ad3012a86dab37a38a106f65d00b85189fe49a3f0f... | 0 | 220000 | unknown | 2020 |
| 32086 | lxg | ae1b500192dc0b6c2d5c69b51f6caf19c630640c0aeb1... | 0 | 600000 | unknown | 2021 |
| 33117 | mvzp ge vbtqxwv | 8ef16126bd9a4691801d2830156dc5528142a45d314593... | 0 | 1200000 | unknown | 2021 |
| 40553 | mqvpto xzaxv rna | a0f794db04d5c13cae6f07c6ce9aee8ff731176cc4d4a1... | 0 | 600000 | unknown | 2021 |
| 41361 | otqcxwtzgf | b60d93faa9a96e2d8362f0b6f16aac79dc484560356ff3... | 91 | 1000000 | unknown | 2021 |
| 56770 | vngo ojzntr | 536e217527d1101538c70b7001d7e28344dba87cd8e246... | 0 | 400000 | unknown | 2021 |
| 62717 | wyvqstugxzn | a1da4f131e2efea24f6d64c66519724aa5c8c5c11a23d9... | 0 | 1850000 | fullstack engineer | 2019 |
| 63629 | bxznb xzw | 5bbebf13d3b3edc497bc5e24d1ab540b66b3d9328d4948... | 0 | 700000 | fullstack engineer | 2019 |
| 66344 | fxkzx ogenfvqt uqxcvnt rxbxnta | 583d48749d8f694951a25ceebe4c0cdebc814a5b6def5b... | 3 | 1800000 | product designer | 2019 |
| 68701 | vzshrvq atcqrgutq | 1978da71c14333352d051fb6054904770b70cecce389d... | 91 | 400000 | devops engineer | 2021 |
| 74313 | ovbohzs trtwngqzxwo | b9fac647e08fe47c6d112466c338e847e82e8f24e16236... | 0 | 1000000 | unknown | 2021 |
| 74945 | fgrntqo prhftq | e0501afab1e9cc00253928e1488701685039437b1780f6... | 3 | 1580000 | engineering leadership | 2019 |
| 80500 | vzshrvq atcqrgutq | 1978da71c14333352d051fb6054904770b70cecce389d... | 91 | 400000 | unknown | 2021 |
| 84882 | hxxtaytvnry sqghu | 82e77fdd3e43e37ec6b805bbfe624d7cd24b37cb0a4317... | 3 | 540000 | qa engineer | 2019 |
| 90049 | vowtzv | 32fb75b1f5c1001ab5e34280641e13d5b91f8cdaa65772... | 2 | 1470000 | frontend engineer | 2019 |
| 94051 | utqoxontzn ojontbo | 9bec1bea9b46f4dcbbb35564e5fef83d1fa52cd2c92b96... | 4 | 2780000 | backend architect | 2019 |
| 94689 | xzentr otwhqxnj bwvett ogenfvqt | 7126263909db5c37b278b05ea56740688062821f452ef6... | 5 | 2720000 | qa engineer | 2019 |
| 99408 | wyvqstugxzn | a1da4f131e2efea24f6d64c66519724aa5c8c5c11a23d9... | 0 | 1850000 | backend engineer | 2019 |
| 100415 | rgsfvqa | b511b17a1aaaa822c6b5d6a04498ff3121808101697407... | 0 | 1600000 | frontend engineer | 2021 |
| 101678 | awm mvzp | 37a5cce01b1e049307d73bbb2138fa54ccb02555c2137a... | 0 | 700000 | unknown | 2021 |
| 106669 | cvrhej ogrhnxgzo | a184e9709d7a9adfc1e2d500770e922a72a584f8dffee3... | 0 | 1300000 | backend engineer | 2020 |
| 117087 | nyghsynfgqpo | 6e793d5dcfe38ed51840bc720ed72e6f6747d57e64ab45... | 0 | 1600000 | unknown | 2019 |
| 119329 | uyxruxo | c0787fd47b636c7cb790a07196a71cf80b9aa5419b0be... | 3 | 1950000 | devops engineer | 2019 |
| 121487 | bvzyvnnvz voogwxvnto | e725ad631cdc4c57a354f59c98b6441f0672c6b7bb8adb... | 83 | 730000 | backend engineer | 2019 |
| 124319 | tr xzeg atoxsz | 3661bbc848e743c2ad62c20fc8a6e0eedebc33e2288b1f... | 3 | 690000 | ios engineer | 2019 |
| 138295 | fvrqvqn rvmo | 73265267f042756723746edd1cedcef322809d50369eea... | 0 | 5700000 | unknown | 2020 |
| 143238 | wxowg | 19bd55263d4e8cb5659b50814dc18b1fbdd1b7e0689e4b... | 5 | 1000000 | backend engineer | 2019 |
| 149468 | tox ogenfvqt | ba78adb8793f4e7257308b724bc84b1b096af03631049b... | 1 | 2950000 | engineering leadership | 2019 |
| 159224 | vrongb | a568e7ea5f84dc0d6f886a943dc71652652de9b00c8c4... | 38 | 3000000 | engineering leadership | 2019 |
| 163444 | ovst xznwww ogenfvqt ucn rna | 3da3c964efe7a5db0f63e91be25391dcaadf24d44cdc59... | 1900 | 3170000 | engineering leadership | 2019 |
| 163455 | wgxznqxmt ntwy exzntwy ctzhqht | a9393ceb6f438669a12e2f5632d515ba2fdb2656c8d727... | 201 | 3500000 | product manager | 2017 |
| 165075 | adw ntwygrgsj | 32a3ee8218df48d1ba56e0e87323a5225c83929d00a6d8... | 3 | 1250000 | devops engineer | 2019 |
| 167456 | xzwnhqt ntwygrgsj | 8b670b30a7588a36cbe0cea33f8c840c3ab25c649d970c... | 2 | 500000 | unknown | 2019 |
| 175205 | vcr | c0118cfea338955b1a4c66292987ab2359ad31e0b49a51... | 6 | 1680000 | data scientist | 2019 |
| 178024 | zthqvrntwyogen | 6212c450f20c14afc8720c33b6b07d667200f7500196db... | 6 | 800000 | data scientist | 2019 |
| 184479 | nqvctrgrp | 3d3a669652938eaf0f9f3c862db3c805a9f55c3961b0c6... | 0 | 2500000 | unknown | 2020 |
| 188483 | gutqv ogrhnxgzo ucn rna zxav | bcb7f27531d7e75a5572bf515fc9b4fb38254b8a77f848... | 2 | 1939999 | backend engineer | 2019 |
| 188672 | wxowg cxatg ntwygrgsxto xzaxv ucn rna | c3cce99fc54361b5c213f8043505d2990c8dfa93669df8... | 200 | 3000000 | engineering leadership | 2019 |
| 193131 | vxqvoxv | 0a5e691a0f8c2c06862ef19d43dc11c22f462f800db26b... | 0 | 800000 | unknown | 2019 |
| 196354 | vaxnjv mxqrw vvuxnvr | 069308440811d578c817c05392f97e8919baac6aa12aa3... | 1 | 2900000 | data scientist | 2019 |

```
In [173]: # Lower limit for orgyear
lower_limit_orgyear = 1970

# Fix `orgyear` values outside the limits
df["orgyear"] = df.apply(
    lambda row: row["ctc_updated_year"] - 1
    if row["orgyear"] < lower_limit_orgyear
    else row["orgyear"]
)
```

```
        else row["orgyear"],
        axis=1
    )
```

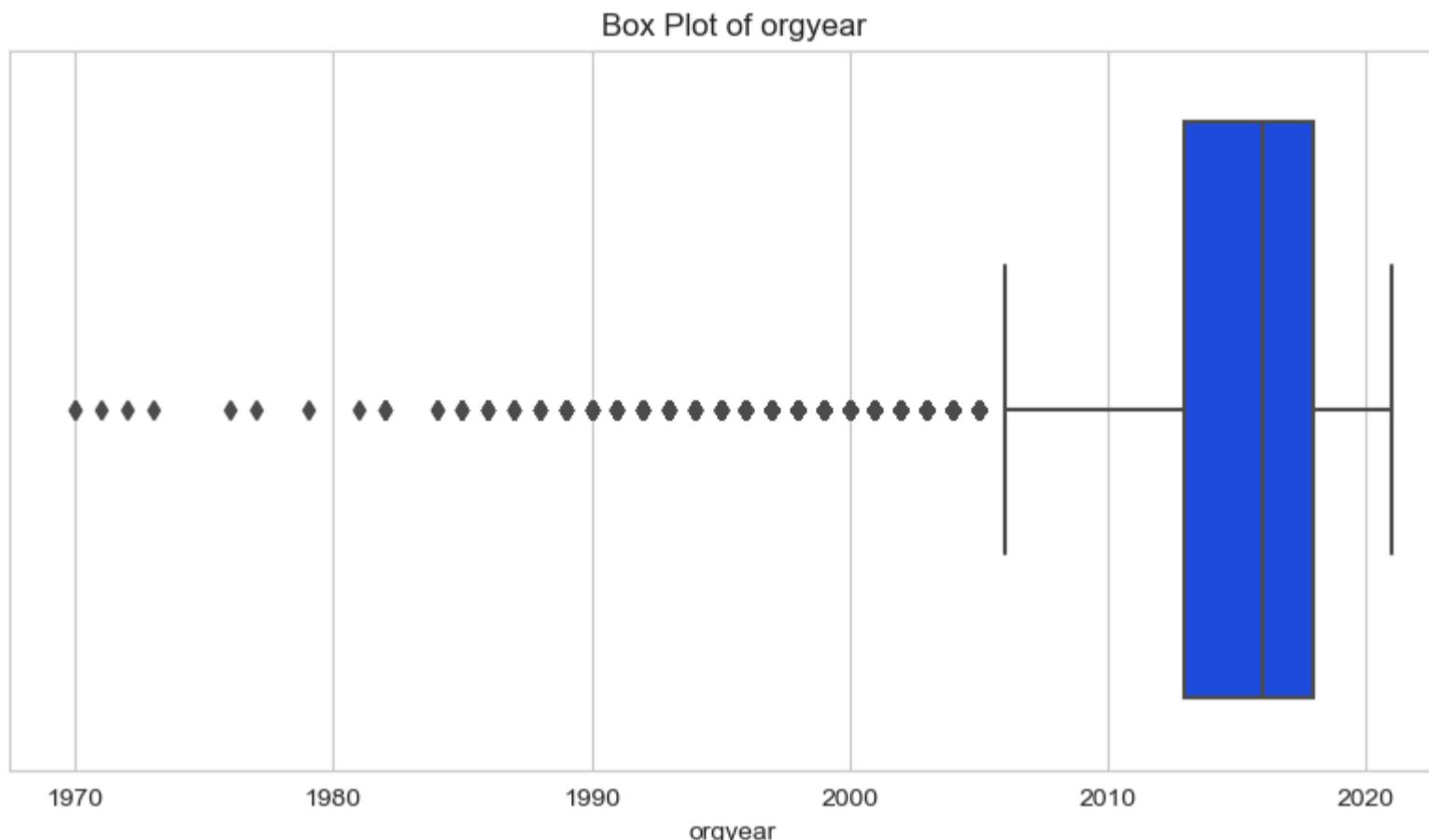
```
In [174]: # Fix `orgyear` values that are greater than `ctc_updated_year`
df["orgyear"] = df.apply(
    lambda row: row["ctc_updated_year"] - 1
    if row["orgyear"] > row["ctc_updated_year"]
    else row["orgyear"],
    axis=1
)
```

```
In [175]: # Box Plot for orgyear
plt.figure(figsize=(10, 5))
sns.boxplot(x=df["orgyear"])
plt.title("Box Plot of orgyear")
plt.show()
```

```
Out[175]: <Figure size 1000x500 with 0 Axes>
```

```
Out[175]: <Axes: xlabel='orgyear'>
```

```
Out[175]: Text(0.5, 1.0, 'Box Plot of orgyear')
```



```
ctc
```

```
In [176]: df['ctc'].value_counts().sort_index()
```

```
Out[176]: ctc
2           1
6           1
14          1
15          1
16          1
...
199990000  2
200000000  378
250000000  1
255555555  1
1000150000 1
Name: count, Length: 3360, dtype: int64
```

```
In [177]: # View rows where ctc is less than or equal to 100
df[df['ctc'] <= 100]
```

```
Out[177]:   company_hash      email_hash  orgyear  ctc  job_position  ctc_updated_year
  54820  uqvpqxnvoogwxvnto  8786759b95d673466e94f62f1b15e4f8c6bd7de6164074...  2020    24        other            2020
  91552       ftm ongqt    512f761579fb116e215cab9821c7f81153f0763e16018...  2016    25  android engineer            2018
 114164      xzntqcxxtfmxn  23ad96d6b6f1ecf554a52f6e9b61677c7d73d8a409a143...  2013    14        unknown            2018
 118236      xzntqcxxtfmxn  f2b58aeed3c074652de2cf3c0717a5d21d6fbef342a78...  2013     6        unknown            2018
 135435      xzntqcxxtfmxn  3505b02549ebe2c95840ac6f0a35561a3b4cbe4b79cdb1...  2014     2  backend engineer            2019
 183804             xm    75357254a31f133e2d3870057922feddeba82b88056a07...  2017    16        unknown            2018
 184946             xm  b8a0bb340583936b5a7923947e9aec21add5ebc50cd60b...  2016    15        unknown            2018
```

🔍 Insights

Considering that the Error in CTC column are Simply in Lakhs, We can either drop them or Multiply them 100000.

```
In [178]: # Multiplying the Value in CTC Column if they are Below 100 by 100000
```

```
df["ctc"] = df["ctc"].apply(lambda x: x * 100000 if x <= 100 else x)
```

```
job_position
```

```
In [179]: # Count of Rows Where Job Position is Unknown  
df[df['job_position'] == 'unknown'].shape[0]
```

```
Out[179]: 52561
```

```
In [180]: df["job_position"].nunique()
```

```
Out[180]: 812
```

```
In [181]: import pandas as pd
```

```
# Step 1: Filter rows where `job_position` is not 'unknown' and `company_hash` is not null  
filtered_df = df[(df["job_position"] != "unknown") & (df["company_hash"].notnull())]  
  
# Step 2: Group by `company_hash` and `job_position` to calculate the count  
grouped_df = filtered_df.groupby(["company_hash", "job_position"]).size().reset_index(name="count")  
  
# Step 3: Rank job positions within each company based on count (highest count first)  
grouped_df["rank"] = grouped_df.groupby("company_hash")["count"].rank(method="first", ascending=False)  
  
# Step 4: Select the top position for each company (`rank` == 1)  
top_position_df = grouped_df[grouped_df["rank"] == 1][["company_hash", "job_position"]]  
  
# Debugging Check 1: Print the shape and inspect top_position_df  
print("Top Position DataFrame shape:", top_position_df.shape)  
print(top_position_df.head())  
  
# Step 5: Merge top positions back into the original DataFrame  
df = df.merge(top_position_df, on="company_hash", how="left", suffixes=("","_top"))  
  
# Debugging Check 2: Identify rows where `job_position_top` is NaN  
print("Rows with no match in top_position_df:", df[df["job_position_top"].isnull()].shape[0])  
  
# Step 6: Replace 'unknown' job positions with the top position from the company  
df["job_position"] = df.apply(  
    lambda row: row["job_position_top"] if row["job_position"] == "unknown" and pd.notnull(row["job_position_top"]) else row["job_position"],  
    axis=1  
)  
  
# Step 7: Drop the temporary column used for merging  
df.drop(columns=["job_position_top"], inplace=True)  
  
# Debugging Check 3: Check if any NaN values remain in `job_position`  
print("Remaining NaN values in job_position:", df["job_position"].isnull().sum())  
  
# Step 8: Drop duplicates if required  
df.drop_duplicates(inplace=True)
```

```
Top Position DataFrame shape: (34222, 2)  
      company_hash    job_position  
0              0        other  
1            0000        other  
2          01 ojztqsj  android engineer  
4  05mz exzytvrny uqxcvnt rxbxnta  backend engineer  
5              1        other  
Rows with no match in top_position_df: 4037  
Remaining NaN values in job_position: 0
```

```
In [182]: # Calculate the mode of the `job_position` column  
job_position_mode = df["job_position"].mode()[0] # The mode() method returns a Series; take the first value  
  
# Fill remaining NaN values in `job_position` with the mode  
df["job_position"].fillna(job_position_mode, inplace=True)  
  
# Verify that no NaN values remain in `job_position`  
print("Remaining NaN values in job_position:", df["job_position"].isnull().sum())
```

```
Remaining NaN values in job_position: 0
```

```
In [183]: df.isnull().sum()
```

```
Out[183]: company_hash      0  
email_hash        0  
orgyear         0  
ctc             0  
job_position     0  
ctc_updated_year 0  
dtype: int64
```

```
In [184]: df["job_position"].value_counts(sort=True)
```

```
Out[184]: job_position
backend engineer          66388
fullstack engineer        28097
other                      24518
frontend engineer          11900
engineering leadership      7034
...
messenger come driver      1
associate technical engineer 1
senior associate platform l1 1
senior web developer       1
azure data factory          1
Name: count, Length: 812, dtype: int64
```

```
In [185]: df["job_position"].nunique()
```

```
Out[185]: 812
```

```
In [186]: # Sort the DataFrame to prioritize records with higher orgyear, ctc_updated_year, and ctc for each email_hash.
df_sorted = df.sort_values(
    by=["email_hash", "orgyear", "ctc_updated_year", "ctc"],
    ascending=[True, False, False, False]
)

# Assign a row number to each record within each email_hash group.
df_sorted["row_number"] = df_sorted.groupby("email_hash").cumcount() + 1

# Filter the DataFrame to keep only the top-ranked record for each email_hash.
df_unique = df_sorted[df_sorted["row_number"] == 1]

# Drop the temporary row_number column.
df_unique = df_unique.drop(columns=["row_number"])


```

```
In [187]: df.head()
```

```
Out[187]:   company_hash           email_hash  orgyear  ctc  job_position  ctc_updated_year
0  atrgxnnnt xzaxv  6de0a4417d18ab14334c3f43397fc13b30c35149d70c05...  2016  1100000        other        2020
1  qtrxvzwt xzegwgbb rxbxnta  b0aaaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10...  2018  449999  fullstack engineer        2019
2  ojzwnvwnxw vx  4860c670bcd48fb96c02a4b0ae3608ae6fdd98176112e9...  2015  2000000  backend engineer        2020
3  nppgutaxv     effdede7a2e7c2af664c8a31d9346385016128d66bbc58...  2017  700000  backend engineer        2019
4  qxen sqghu    6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520...  2017  1400000  fullstack engineer        2019
```

Unique Values

```
In [188]: # Checking Unique Values for Each Column
```

```
for i in df.columns:
    print("Count of Unique Values in ", i, "Column are :-", df[i].nunique())
    print("-" * 80)
```

```
Count of Unique Values in company_hash Column are :- 37299
-----
Count of Unique Values in email_hash Column are :- 153443
-----
Count of Unique Values in orgyear Column are :- 47
-----
Count of Unique Values in ctc Column are :- 3353
-----
Count of Unique Values in job_position Column are :- 812
-----
Count of Unique Values in ctc_updated_year Column are :- 7
-----
```

```
In [189]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 192051 entries, 0 to 205808
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   company_hash     192051 non-null   object 
 1   email_hash       192051 non-null   object 
 2   orgyear          192051 non-null   int64  
 3   ctc              192051 non-null   int64  
 4   job_position     192051 non-null   object 
 5   ctc_updated_year 192051 non-null   int32  
dtypes: int32(1), int64(2), object(3)
memory usage: 9.5+ MB
```

```
In [190]: # Let's Make a Copy of the Data Frame
```

```
df_copy = df.copy()
```

```
In [191]: # Create a DataFrame with unique rows based on email_hash, prioritizing recent and high values
df_unique = (
    df.sort_values(by=["email_hash", "orgyear", "ctc_updated_year", "ctc"], ascending=[True, False, False, False])
)
```

```
    .drop_duplicates(subset="email_hash", keep="first")
)
```

```
In [192]: # Remove duplicate rows from the DataFrame
df = df.drop_duplicates()
```

3. Feature Engineering

```
In [193]: df["orgyear"].max(), df["ctc_updated_year"].max()
```

```
Out[193]: (2021, 2021)
```

NOTE:

- We will use the `Current Year` as **2022** as the last Year in the Data Set is 2021

```
In [194]: # Calculate the Years of Experience (YoE) for each record, keeping 2022 as the Current Year
df["yoe"] = 2022 - df["orgyear"]
```

```
In [195]: df.head()
```

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe |
|---|-----------------------------|--|---------|---------|--------------------|------------------|-----|
| 0 | atrgxnnnt xzaxv | 6de0a4417d18ab14334c3f43397fc13b30c35149d70c05... | 2016 | 1100000 | other | 2020 | 6 |
| 1 | qtrxvzwt xzegwgb bb rxbxnta | b0aaaf1ac138b53cb6e039ba2c3d6604a250d02d5145c10... | 2018 | 449999 | fullstack engineer | 2019 | 4 |
| 2 | ojzwnvwnxw vx | 4860c670bcd48fb96c02a4b0ae3608ae6fd98176112e9... | 2015 | 2000000 | backend engineer | 2020 | 7 |
| 3 | ngpgutaxv | effdede7a2e7c2af664c8a31d9346385016128d66bbc58... | 2017 | 700000 | backend engineer | 2019 | 5 |
| 4 | qxen sqghu | 6ff54e709262f55cb999a1c1db8436cb2055d8f79ab520... | 2017 | 1400000 | fullstack engineer | 2019 | 5 |

```
In [196]: df["yoe"].min(), df["yoe"].max()
```

```
Out[196]: (1, 52)
```

```
In [197]: import pandas as pd
```

```
# Sort the dataframe by the necessary columns: 'email_hash', 'orgyear', 'ctc_updated_year', 'ctc' in descending order
df.sort_values(by=['email_hash', 'orgyear', 'ctc_updated_year', 'ctc'], ascending=[True, False, False, False], inplace=True)

# Add a row number based on the sorted data for each 'email_hash'
df['rn'] = df.groupby('email_hash').cumcount() + 1

# Compute additional columns:
# 1. Calculate the number of jobs per 'email_hash'
df['num_jobs'] = df.groupby('email_hash')['email_hash'].transform('count')

# 2. Calculate the growth as per the formula (max(ctc) - min(ctc)) / min(ctc) for each 'email_hash'
df['growth'] = df.groupby('email_hash')['ctc'].transform(lambda x: (x.max() - x.min()) / x.min())

# Filter the rows where row number is 1 (to mimic 'rn = 1' in SQL)
df = df[df['rn'] == 1]

# Drop the row number column (as per the SQL query to drop 'rn')
df.drop(columns=['rn'], inplace=True)

# Sort the final dataframe by 'num_jobs' in descending order
df.sort_values(by='num_jobs', ascending=False, inplace=True)

# Display the final dataframe
df.head()
```

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth |
|--------|--------------------|---|---------|---------|--------------------|------------------|-----|----------|--------|
| 65909 | cvrhtbgbtzhb | 298528ce3160cc761e4dc37a07337ee2e0589df251d736... | 2018 | 720000 | backend engineer | 2020 | 4 | 9 | 0.029 |
| 10214 | wgcxvb ntwygrgsxto | 3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94... | 2018 | 1130000 | engineering intern | 2021 | 4 | 9 | 0.041 |
| 9857 | ihvrwgbb | 6842660273f70e9aa239026ba33bfe82275d6ab0d20124... | 2017 | 2400000 | qa engineer | 2020 | 5 | 9 | 0.200 |
| 4557 | gqvwr | 4818edfd67ed8563dde5d083306485d91d19f4f1c95d19... | 2015 | 1200000 | backend architect | 2020 | 7 | 8 | 0.000 |
| 133770 | nyt a t oyvf sqghu | c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7... | 2013 | 2300000 | other | 2020 | 9 | 8 | 0.513 |

```
In [198]: # New Feature - CTC to YoE Ratio
```

```
df["ctc_yoe_ratio"] = df["ctc"] / (df["yoe"] * 12 * 1000)

# Display the result
print(df[['company_hash', 'job_position', 'ctc', 'yoe', 'ctc_yoe_ratio']].head())
```

| | company_hash | job_position | ctc | yoe | ctc_yoe_ratio |
|--------|---------------------|--------------------|---------|-----|---------------|
| 65909 | cvrhtbgbtznhb | backend engineer | 720000 | 4 | 15.000 |
| 10214 | wgcxvb ntwyzgrgsxto | engineering intern | 1130000 | 4 | 23.542 |
| 9857 | ihvrwgb | qa engineer | 2400000 | 5 | 40.000 |
| 4557 | gqvwrt | backend architect | 1200000 | 7 | 14.286 |
| 133770 | nyt a t oyvf sqghu | other | 2300000 | 9 | 21.296 |

3.1 - Creating Individual Groups for Manual Clustering

```
In [199]: # Grouping by company, job position, and years of experience,
# to calculate aggregate statistics for 'ctc' such as mean, median, max, min, and count
grouped_by_position_exp = df.groupby(["company_hash", "job_position", "yoe"]).agg(
    mean_ctc_exp=("ctc", "mean"),
    median_ctc_exp=("ctc", "median"),
    max_ctc_exp=("ctc", "max"),
    min_ctc_exp=("ctc", "min"),
    count_exp=('ctc', 'size') # Correct way to get the count
).sort_values("count_exp", ascending=False) # Sorting by 'count_exp' in descending order

# Flatten the multi-index by resetting the index
grouped_by_position_exp = grouped_by_position_exp.reset_index()

grouped_by_position_exp.head(5)

# Grouping by company and job position,
# to calculate aggregate statistics for 'ctc' such as mean, median, max, min, and count
grouped_by_position = df.groupby(["company_hash", "job_position"]).agg(
    mean_ctc_pos=("ctc", "mean"),
    median_ctc_pos=("ctc", "median"),
    max_ctc_pos=("ctc", "max"),
    min_ctc_pos=("ctc", "min"),
    count_pos=('ctc', 'size') # Correct way to get the count
).sort_values("count_pos", ascending=False) # Sorting by 'count_pos' in descending order

# Flatten the multi-index by resetting the index
grouped_by_position = grouped_by_position.reset_index()

grouped_by_position.head(5)

# Grouping by company,
# to calculate aggregate statistics for 'ctc' such as mean, median, max, min, and count
grouped_by_company = df.groupby(["company_hash"]).agg(
    mean_ctc_com=("ctc", "mean"),
    median_ctc_com=("ctc", "median"),
    max_ctc_com=("ctc", "max"),
    min_ctc_com=("ctc", "min"),
    count_com=('ctc', 'size') # Correct way to get the count
).sort_values("count_com", ascending=False) # Sorting by 'count_com' in descending order

# Flatten the multi-index by resetting the index
grouped_by_company = grouped_by_company.reset_index()

grouped_by_company.head(5)
```

```
Out[199]:   company_hash  job_position  yoe  mean_ctc_exp  median_ctc_exp  max_ctc_exp  min_ctc_exp  count_exp
0  nvnv wgzohrnzwj otqcxwto  backend engineer   3  1463465.871  400000.000  160000000     4700      727
1  nvnv wgzohrnzwj otqcxwto  backend engineer   4  1601404.161  430000.000  200000000     3300      541
2           xzegojo          other   3   715774.169  420000.000  72000000     100000     403
3           xzegojo          other   4  2176306.000  435000.000  200000000     13000      366
4  nvnv wgzohrnzwj otqcxwto  backend engineer   2  1475113.092  440000.000  200000000     40000      336
```

```
Out[199]:   company_hash  job_position  mean_ctc_pos  median_ctc_pos  max_ctc_pos  min_ctc_pos  count_pos
0  nvnv wgzohrnzwj otqcxwto  backend engineer  1779334.246  470000.000  200000000       600     2747
1           xzegojo          other  1989436.459  500000.000  200000000      4000     1738
2           vbvkzg  backend engineer  3490234.921  2200000.000  200000000      2290     1520
3  zgn vuurxwvwmrt vwwghzn  backend engineer  1600478.683  600000.000  180000000      1000     1126
4           wgszxkvzn          other  2022947.531  500000.000  200000000      3000     1086
```

```
Out[199]:   company_hash  mean_ctc_com  median_ctc_com  max_ctc_com  min_ctc_com  count_com
0  nvnv wgzohrnzwj otqcxwto  2119574.830  460000.000  200000000       600     5271
1           xzegojo          1620473.456  500000.000  200000000      3250     3455
2           vbvkzg            3745454.345  2000000.000  200000000      2000     2548
3  zgn vuurxwvwmrt vwwghzn  2945297.317  600000.000  199800000      1000     2278
4           wgszxkvzn          2038290.080  600000.000  200000000      3000     2188
```

3.2 - Merging the Groups to the main dataset

3.2.1 - Flagging

```
In [200]: # Merging df with grouped_by_position_exp (g1) and adding 'class_flag' based on comparison with mean_ctc_exp
df = df.merge(grouped_by_position_exp[['company_hash', 'job_position', 'yoe', 'mean_ctc_exp']], 
              on=['company_hash', 'job_position', 'yoe'],
              how='left')

# Applying conditions for 'class_flag'
df['class_flag'] = df.apply(
    lambda row: 1 if row['ctc'] > row['mean_ctc_exp'] else (3 if row['ctc'] < row['mean_ctc_exp'] else 2),
    axis=1
)

# Merging df with grouped_by_position (g2) and adding 'designation_flag' based on comparison with mean_ctc_pos
# Rename 'mean_ctc_pos' column in grouped_by_position to avoid duplication
grouped_by_position = grouped_by_position.rename(columns={'mean_ctc_pos': 'mean_ctc_position'})

df = df.merge(grouped_by_position[['company_hash', 'job_position', 'mean_ctc_position']], 
              on=['company_hash', 'job_position'],
              how='left')

# Applying conditions for 'designation_flag'
df['designation_flag'] = df.apply(
    lambda row: 1 if row['ctc'] > row['mean_ctc_position'] else (3 if row['ctc'] < row['mean_ctc_position'] else 2),
    axis=1
)

# Merging df with grouped_by_company (g3) and adding 'tier_flag' based on comparison with mean_ctc_com
# Rename 'mean_ctc_com' column in grouped_by_company to avoid duplication
grouped_by_company = grouped_by_company.rename(columns={'mean_ctc_com': 'mean_ctc_company'})

df = df.merge(grouped_by_company[['company_hash', 'mean_ctc_company']], 
              on='company_hash',
              how='left')

# Applying conditions for 'tier_flag'
df['tier_flag'] = df.apply(
    lambda row: 1 if row['ctc'] > row['mean_ctc_company'] else (3 if row['ctc'] < row['mean_ctc_company'] else 2),
    axis=1
)
```

```
In [201]: print(grouped_by_position_exp.columns)
```

```
Index(['company_hash', 'job_position', 'yoe', 'mean_ctc_exp', 'median_ctc_exp',
       'max_ctc_exp', 'min_ctc_exp', 'count_exp'],
      dtype='object')
```

```
In [202]: words = df["job_position"].str.split(" ").explode()
WORDS = Counter(words.to_list())
```

```
In [203]: # Total Number of Words
```

```
len(WORDS)
```

```
Out[203]: 391
```

3.3 - Creating New Columns based on Job Positions

```
In [204]: # Creating the 'is_other' column based on job positions containing 'other' or 'unknown'
df['is_other'] = df['job_position'].str.contains(r"other|unknown", case=False, na=False)

# Creating the 'is_developer' column for job positions related to development and programming
df['is_developer'] = df['job_position'].str.contains(
    r"developer|software|engineer|programmer|dev|development|backend|frontend|fullstack|web|mobile|app|ios|android|qa|automation|testing|sde|a",
    case=False, na=False
)

# Creating the 'is_tech' column for tech-related job positions
df['is_tech'] = df['job_position'].str.contains(
    r"engineer|developer|cto|application|aiml|fullstack|scientist|designer|frontend|data|engineering|system|data|qa|principal|security|android",
    case=False, na=False
)

# Creating the 'is_management' column for job positions related to management and Leadership roles
df['is_management'] = df['job_position'].str.contains(
    r"manager|director|head|vp|president|lead|executive|founder|consulting|leadership|product|administrator|co|project|consultant|team",
    case=False, na=False
)

# Creating the 'is_sales' column for sales and business-related roles
df['is_sales'] = df['job_position'].str.contains(
    r"sales|business|marketing|account|customer|service|support|growth|strategy|operations|commercial|client|relation|partner|engagement|solut",
    case=False, na=False
)

# Creating the 'is_non_coder' column, marking positions as non-coders if they do not match 'is_tech'
df['is_non_coder'] = df['job_position'].str.contains(r"non coder", case=False, na=False) | (df['is_tech'] == False)
```

```
In [205]: df.head()
```

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth | ctc_ye |
|---|------------------------|---|---------|---------|--------------------|------------------|-----|----------|--------|--------|
| 0 | cvrhtbgbtzhb | 298528ce3160cc761e4dc37a07337ee2e0589df251d736... | 2018 | 720000 | backend engineer | 2020 | 4 | 9 | 0.029 | |
| 1 | wgcxvb ntwyzgrgsxto | 3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94... | 2018 | 1130000 | engineering intern | 2021 | 4 | 9 | 0.041 | |
| 2 | ihvrwgb | 6842660273f70e9aa239026ba33bfe82275d6ab0d20124... | 2017 | 2400000 | qa engineer | 2020 | 5 | 9 | 0.200 | |
| 3 | gqvwrt | 4818edfd67ed8563dde5d083306485d91d19f4f1c95d19... | 2015 | 1200000 | backend architect | 2020 | 7 | 8 | 0.000 | |
| 4 | nyt a t oyvf sqghu | c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7... | 2013 | 2300000 | other | 2020 | 9 | 8 | 0.513 | |

5 rows × 22 columns

```
# Calculate the quantiles for the 'ctc' column
quantiles = df['ctc'].quantile([0.25, 0.50, 0.75, 0.95])

# Define a function to assign the bins based on the quantiles
def assign_ctc_bin(ctc_value):
    if ctc_value > quantiles[0.95]:
        return 'very_high'
    elif ctc_value > quantiles[0.75]:
        return 'high'
    elif ctc_value > quantiles[0.50]:
        return 'medium'
    elif ctc_value > quantiles[0.25]:
        return 'low'
    else:
        return 'very_low'

# Apply the function to the 'ctc' column
df['ctc_bin'] = df['ctc'].apply(assign_ctc_bin)

# Display the first few rows to verify
df.head()
```

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth | ctc_ye |
|---|------------------------|---|---------|---------|--------------------|------------------|-----|----------|--------|--------|
| 0 | cvrhtbgbtzhb | 298528ce3160cc761e4dc37a07337ee2e0589df251d736... | 2018 | 720000 | backend engineer | 2020 | 4 | 9 | 0.029 | |
| 1 | wgcxvb ntwyzgrgsxto | 3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94... | 2018 | 1130000 | engineering intern | 2021 | 4 | 9 | 0.041 | |
| 2 | ihvrwgb | 6842660273f70e9aa239026ba33bfe82275d6ab0d20124... | 2017 | 2400000 | qa engineer | 2020 | 5 | 9 | 0.200 | |
| 3 | gqvwrt | 4818edfd67ed8563dde5d083306485d91d19f4f1c95d19... | 2015 | 1200000 | backend architect | 2020 | 7 | 8 | 0.000 | |
| 4 | nyt a t oyvf sqghu | c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7... | 2013 | 2300000 | other | 2020 | 9 | 8 | 0.513 | |

5 rows × 23 columns

```
df_FE = df.copy()
```

4. EDA

```
Colors_Palette = ['#313866", "#504099", "#974EC3", "#FE7BE5", "#392467", "#5D3587", "#A367B1", "#FFD1E3", "#F2AFEF", "#C499F3", "#7360DF", "#313866"]

clr_palette = sns.color_palette()
sns.palplot(sns.color_palette(Colors_Palette))

plt.show()
```



4.1 - Univariate Analysis

```
df.columns
```

```
Index(['company_hash', 'email_hash', 'orgyear', 'ctc', 'job_position',
       'ctc_updated_year', 'yoe', 'num_jobs', 'growth', 'ctc_yoe_ratio',
       'mean_ctc_exp', 'class_flag', 'mean_ctc_position', 'designation_flag',
       'mean_ctc_company', 'tier_flag', 'is_other', 'is_developer', 'is_tech',
       'is_management', 'is_sales', 'is_non_coder', 'ctc_bin'],
      dtype='object')
```

In [210]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153443 entries, 0 to 153442
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   company_hash     153443 non-null   object  
 1   email_hash       153443 non-null   object  
 2   orgyear          153443 non-null   int64   
 3   ctc              153443 non-null   int64   
 4   job_position     153443 non-null   object  
 5   ctc_updated_year 153443 non-null   int32   
 6   yoe              153443 non-null   int64   
 7   num_jobs         153443 non-null   int64   
 8   growth            153443 non-null   float64 
 9   ctc_yoe_ratio    153443 non-null   float64 
 10  mean_ctc_exp    153443 non-null   float64 
 11  class_flag       153443 non-null   int64   
 12  mean_ctc_position 153443 non-null   float64 
 13  designation_flag 153443 non-null   int64   
 14  mean_ctc_company 153443 non-null   float64 
 15  tier_flag        153443 non-null   int64   
 16  is_other          153443 non-null   bool    
 17  is_developer      153443 non-null   bool    
 18  is_tech           153443 non-null   bool    
 19  is_management     153443 non-null   bool    
 20  is_sales          153443 non-null   bool    
 21  is_non_coder      153443 non-null   bool    
 22  ctc_bin           153443 non-null   object  
dtypes: bool(6), float64(5), int32(1), int64(7), object(4)
memory usage: 20.2+ MB
```

4.1.1 - Top 20 Job Positions

```
In [211]: def get_top_20_value_records(data, feature_name):
    # Get value counts sorted in descending order and get the top 20
    top_20_values = data[feature_name].value_counts().head(20).index

    # Filter the records where the feature value is one of the top 20
    return data[data[feature_name].isin(top_20_values)]
```

```
In [212]: # Get the top 20 records for 'job_position'
top_20_job_positions = get_top_20_value_records(df_FE, 'job_position')

# Count plot for 'job_position'

plt.figure(figsize=(12, 6))
sns.countplot(y='job_position', data=top_20_job_positions, palette='viridis')
plt.title('Top 20 Job Positions')
plt.xlabel('Count')
plt.ylabel('Job Position')
plt.show()
```

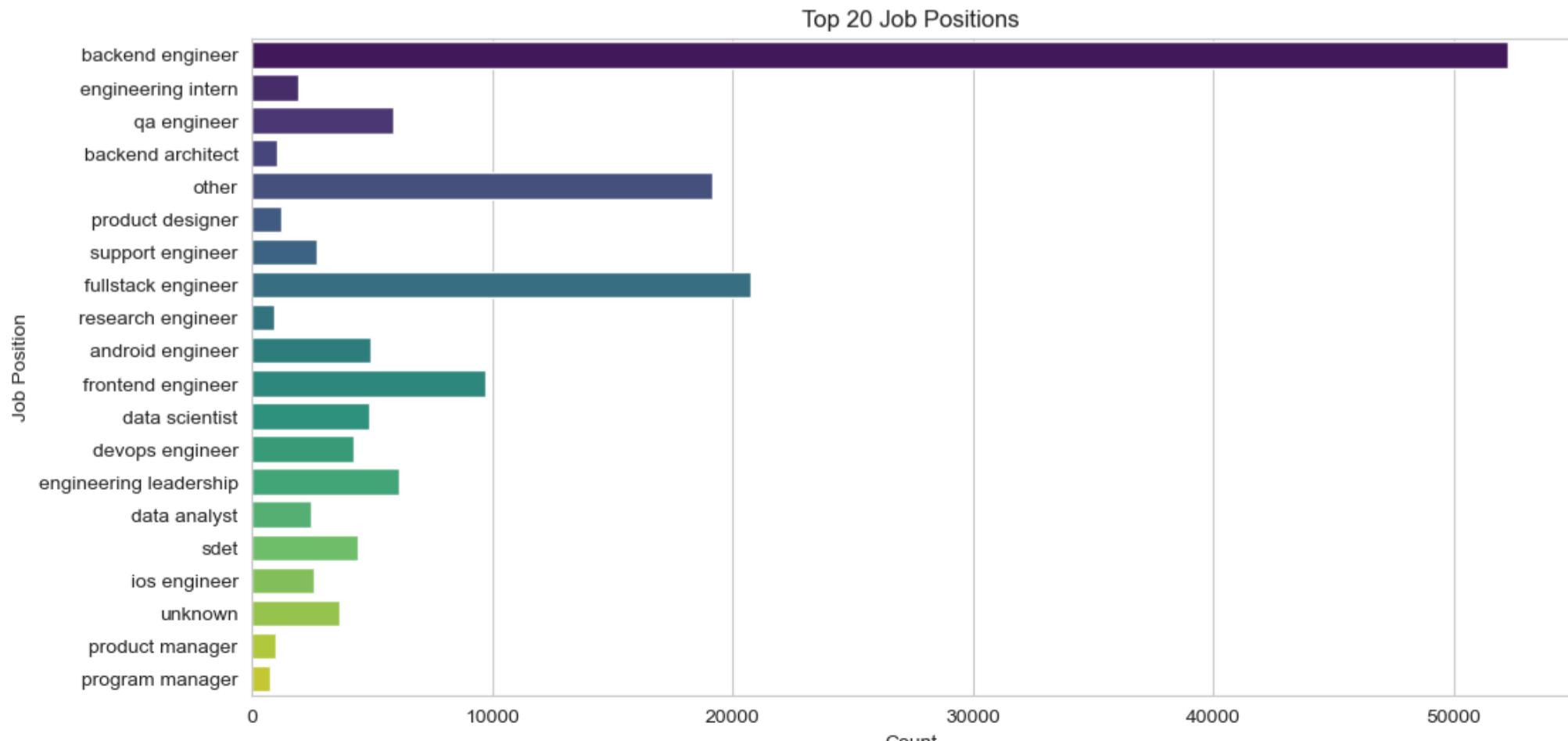
Out[212]: <Figure size 1200x600 with 0 Axes>

Out[212]: <Axes: xlabel='count', ylabel='job_position'>

Out[212]: Text(0.5, 1.0, 'Top 20 Job Positions')

Out[212]: Text(0.5, 0, 'Count')

Out[212]: Text(0, 0.5, 'Job Position')



🔍 Insights

- Most the candidates belong to web developement domain

4.1.2 - Top 20 Companies

```
In [213]: # Get the top 20 records for 'company_hash'  
top_20_companies = get_top_20_value_records(df_FE, 'company_hash')  
  
# Count plot for 'company_hash'  
  
plt.figure(figsize=(12, 6))  
sns.countplot(y='company_hash', data=top_20_companies, palette='viridis')  
plt.title('Top 20 Companies')  
plt.xlabel('Count')  
plt.ylabel('Company Hash')  
plt.show()
```

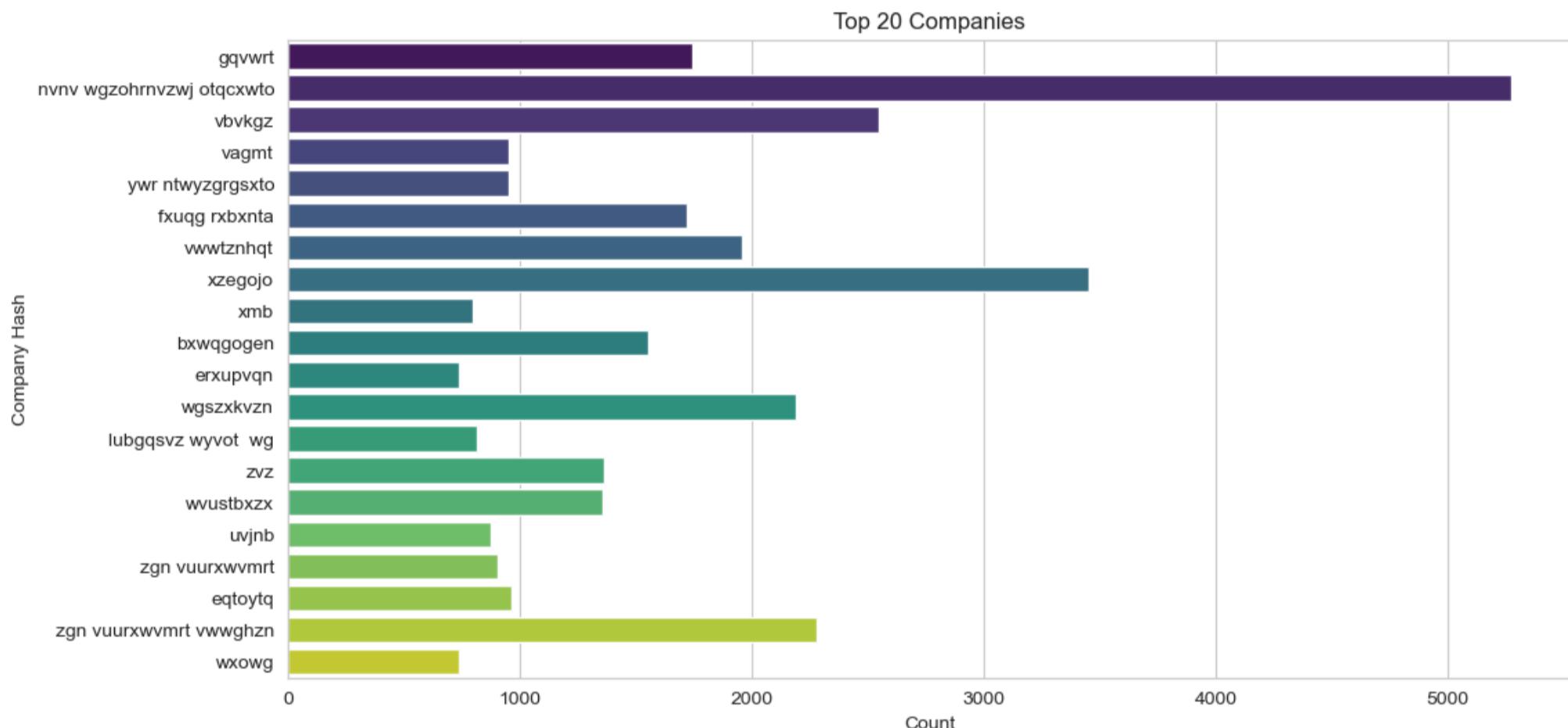
Out[213]: <Figure size 1200x600 with 0 Axes>

Out[213]: <Axes: xlabel='count', ylabel='company_hash'>

Out[213]: Text(0.5, 1.0, 'Top 20 Companies')

Out[213]: Text(0.5, 0, 'Count')

Out[213]: Text(0, 0.5, 'Company Hash')



4.1.3 - Top 20 CTC Packages

```
In [214]: # Get the top 20 records for 'ctc_bin'  
top_20_ctc_bins = get_top_20_value_records(df_FE, 'ctc_bin')  
  
# Count plot for 'ctc_bin'  
  
plt.figure(figsize=(12, 6))  
sns.countplot(y='ctc_bin', data=top_20_ctc_bins, palette='viridis')  
plt.title('Top 20 CTC Bins')  
plt.xlabel('Count')  
plt.ylabel('CTC Bin')  
plt.show()
```

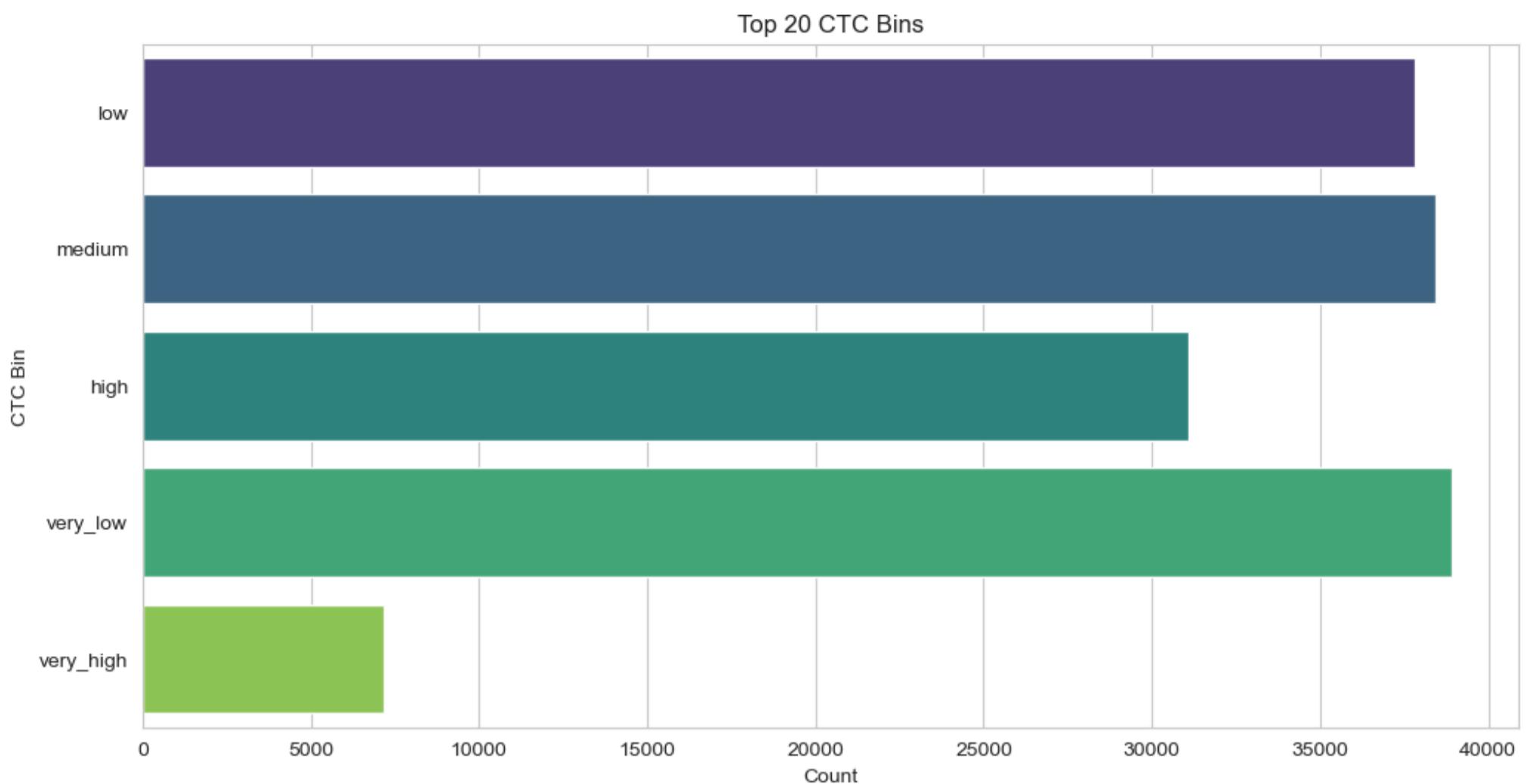
Out[214]: <Figure size 1200x600 with 0 Axes>

Out[214]: <Axes: xlabel='count', ylabel='ctc_bin'>

Out[214]: Text(0.5, 1.0, 'Top 20 CTC Bins')

Out[214]: Text(0.5, 0, 'Count')

Out[214]: Text(0, 0.5, 'CTC Bin')



4.1.4 - Further Visual Analysis

```
In [215]: # Boxplot and Distplot for Selected Columns

# Columns to include in the plots
selected_columns = ['orgyear', 'ctc_updated_year', 'yoe']

# Set plot styles
plt.style.use('default')
plt.style.use('seaborn-v0_8-bright')

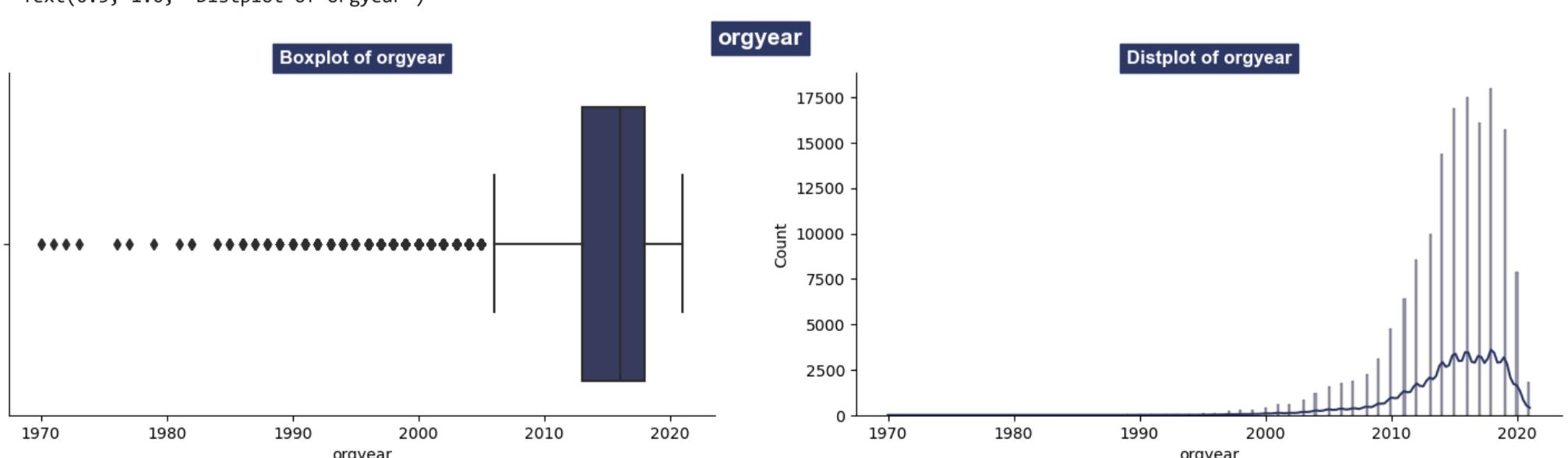
# Loop through the selected columns and create boxplots and distplots
for _, col in enumerate(selected_columns):
    plt.figure(figsize=(18, 4))
    plt.suptitle(f'{col}', fontsize=15, fontfamily='Arial', fontweight='bold', backgroundcolor=Colors_Palette[_], color='w')

    # Boxplot
    plt.subplot(121)
    sns.boxplot(x=df[col], color=Colors_Palette[_])
    plt.title(f'Boxplot of {col}', fontsize=12, fontfamily='Arial', fontweight='bold', backgroundcolor=Colors_Palette[_], color='w')

    # Distplot
    plt.subplot(122)
    sns.histplot(x=df[col], kde=True, color=Colors_Palette[_])
    plt.title(f'Distplot of {col}', fontsize=12, fontfamily='Arial', fontweight='bold', backgroundcolor=Colors_Palette[_], color='w')

sns.despine()
plt.show()
```

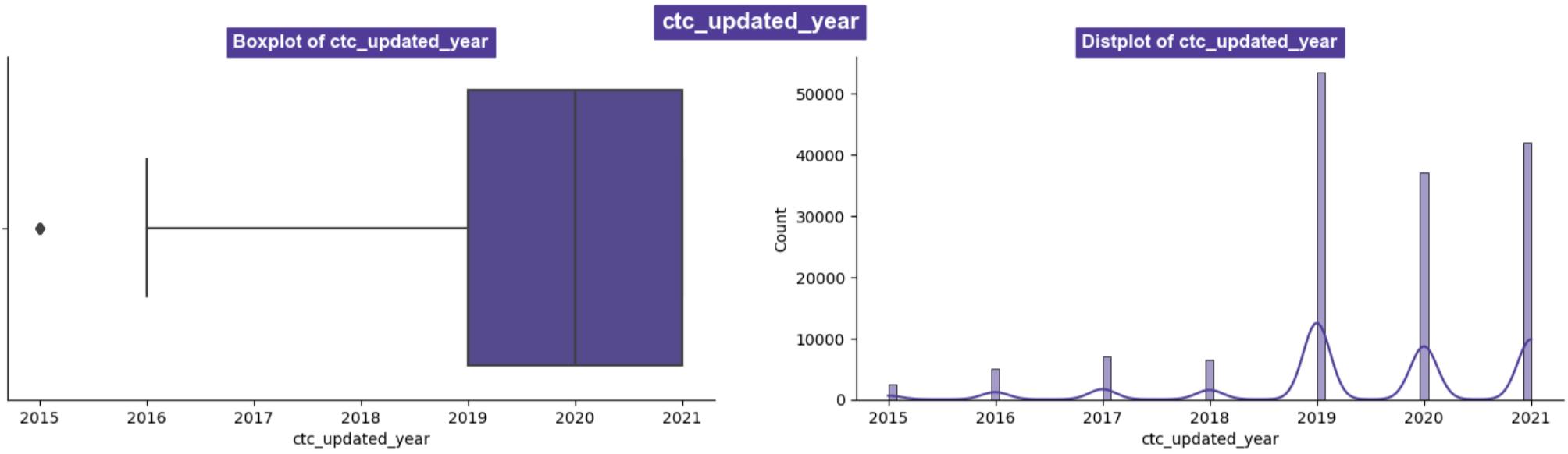
```
Out[215]: <Figure size 1800x400 with 0 Axes>
Out[215]: Text(0.5, 0.98, 'orgyear')
Out[215]: <Axes: >
Out[215]: <Axes: xlabel='orgyear'>
Out[215]: Text(0.5, 1.0, 'Boxplot of orgyear')
Out[215]: <Axes: >
Out[215]: <Axes: xlabel='orgyear', ylabel='Count'>
Out[215]: Text(0.5, 1.0, 'Distplot of orgyear')
```



```

Out[215]: <Figure size 1800x400 with 0 Axes>
Out[215]: Text(0.5, 0.98, 'ctc_updated_year')
Out[215]: <Axes: >
Out[215]: <Axes: xlabel='ctc_updated_year'>
Out[215]: Text(0.5, 1.0, 'Boxplot of ctc_updated_year')
Out[215]: <Axes: >
Out[215]: <Axes: xlabel='ctc_updated_year', ylabel='Count'>
Out[215]: Text(0.5, 1.0, 'Distplot of ctc_updated_year')

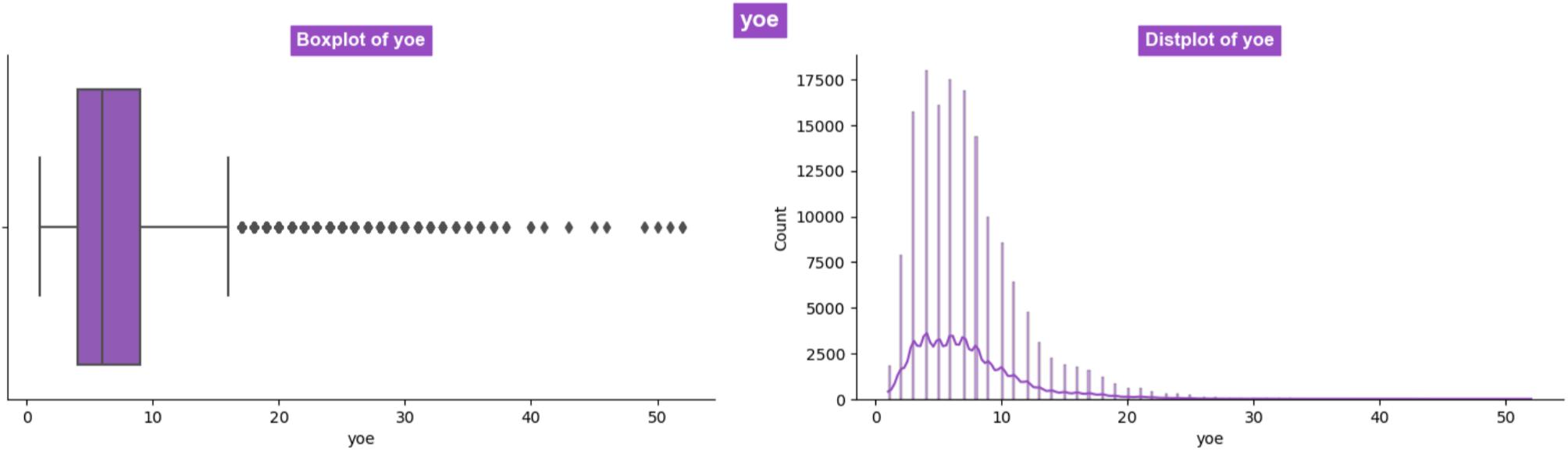
```



```

Out[215]: <Figure size 1800x400 with 0 Axes>
Out[215]: Text(0.5, 0.98, 'yoe')
Out[215]: <Axes: >
Out[215]: <Axes: xlabel='yoe'>
Out[215]: Text(0.5, 1.0, 'Boxplot of yoe')
Out[215]: <Axes: >
Out[215]: <Axes: xlabel='yoe', ylabel='Count'>
Out[215]: Text(0.5, 1.0, 'Distplot of yoe')

```



💡 Insights

- The majority of candidates have less than 10 years of experience.
- Most CTC updates occurred between 2019 and 2021.
- A significant number of candidates joined after 2010.

```

In [216]: import pandas as pd

# Filter rows where 'job_position' contains 'intern' or 'trainee', case insensitive
filtered_df = df[df['job_position'].str.contains('intern|trainee', case=False, na=False)]

# Select distinct 'job_position' and 'ctc' and sort by 'ctc' in descending order
distinct_sorted_df = filtered_df[['job_position', 'ctc']].drop_duplicates().sort_values(by='ctc', ascending=False)

# Limit the result to the top 10 rows
top_10_df = distinct_sorted_df.head(10)

# Display the result
print(top_10_df)

```

```

job_position      ctc
33304  engineering intern  200000000
94233  engineering intern  155000000
9907   engineering intern  100000000
139597  engineering intern  80000000
89184   engineering intern  20000000
36711   engineering intern  19700000
31222   engineering intern  14100000
25934   engineering intern  12000000
36423   engineering intern  11500000
85602    security intern   11000000

```

NOTE:

Job Position like **intern** have Very High CTC which is not Possible.

```
In [217]: df["ctc"].quantile(0.95), df["ctc"].quantile(0.05)
```

```
Out[217]: (4000000.0, 200000.0)
```

```
In [218]: # Filter the DataFrame to retain rows where 'ctc' is within the 5th and 95th percentiles
ctc_filtered_df = df[(df['ctc'] < df['ctc'].quantile(0.95)) & (df['ctc'] > df['ctc'].quantile(0.05))]
```

```
In [219]: # Mean and Median of CTC for the Filtered Data
```

```
mean_ctc = ctc_filtered_df['ctc'].mean()
median_ctc = ctc_filtered_df['ctc'].median()

# Display the results
print(f"Mean CTC: {mean_ctc:.2f}")
print(f"Median CTC: {median_ctc:.2f}")
```

```
Mean CTC: 1215967.07
Median CTC: 1000000.00
```

Insights

- ***Mean Salary*** is 12 Lakhs
- ***Median Salary*** is 10 Lakhs

```
In [220]: # Filter the DataFrame to retain rows where 'job_position' is in the top 10
top_10_job_positions = df['job_position'].value_counts().head(10).index
top_10_positions_df = df[df['job_position'].isin(top_10_job_positions)]
```

```
# Group by 'job_position' and calculate the mean 'ctc'
average_ctc_top_10 = (
    top_10_positions_df
    .groupby('job_position', as_index=False) # Ensure 'job_position' is a column, not an index
    .agg(average_ctc=('ctc', 'mean')) # Calculate the mean of 'ctc'
)
```

```
# Display the results
print(average_ctc_top_10)
```

```
# Optionally sort the results for better readability
average_ctc_top_10 = average_ctc_top_10.sort_values(by='average_ctc', ascending=False)
```

```
# Print sorted results
print("\nSorted Average CTC for Top 10 Job Positions:")
print(average_ctc_top_10)
```

| | job_position | average_ctc |
|---|------------------------|-------------|
| 0 | android engineer | 1540722.520 |
| 1 | backend engineer | 2074889.063 |
| 2 | data scientist | 1935848.904 |
| 3 | devops engineer | 1653288.719 |
| 4 | engineering leadership | 3982217.920 |
| 5 | frontend engineer | 2323741.050 |
| 6 | fullstack engineer | 1978556.112 |
| 7 | other | 4043263.789 |
| 8 | qa engineer | 2027199.074 |
| 9 | sdet | 1268721.121 |

```
Sorted Average CTC for Top 10 Job Positions:
```

| | job_position | average_ctc |
|---|------------------------|-------------|
| 7 | other | 4043263.789 |
| 4 | engineering leadership | 3982217.920 |
| 5 | frontend engineer | 2323741.050 |
| 1 | backend engineer | 2074889.063 |
| 8 | qa engineer | 2027199.074 |
| 6 | fullstack engineer | 1978556.112 |
| 2 | data scientist | 1935848.904 |
| 3 | devops engineer | 1653288.719 |
| 0 | android engineer | 1540722.520 |
| 9 | sdet | 1268721.121 |

```
In [76]: # Box Plot & Histplot for 'ctc' after filtering
```

```
plt.figure(figsize=(10, 12))
plt.suptitle('Distribution of CTC', fontsize=16, fontweight='bold', color='navy')
```

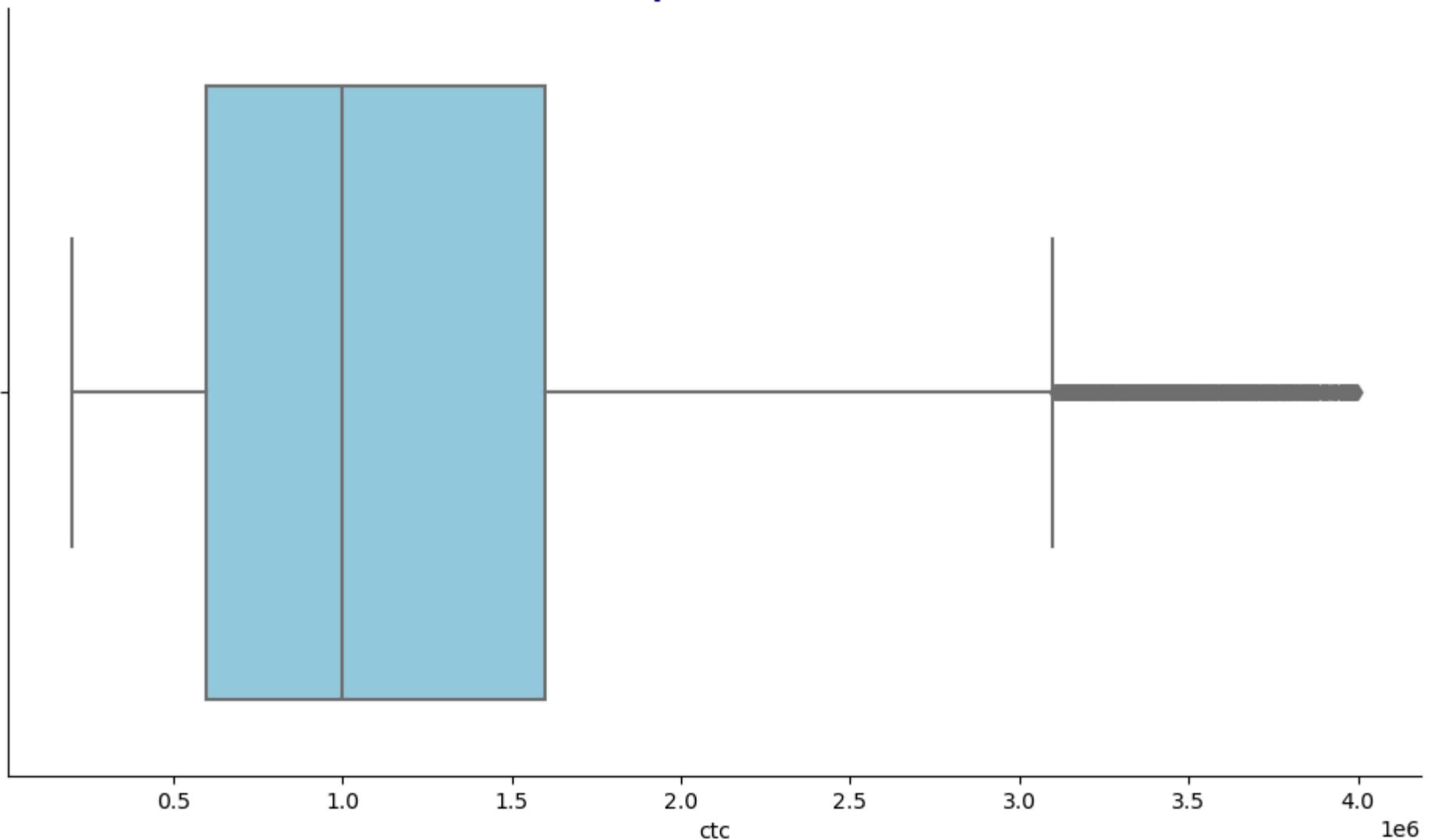
```
# Boxplot
plt.subplot(211)
sns.boxplot(x=ctc_filtered_df['ctc'], color='skyblue')
plt.title('Boxplot of CTC', fontsize=14, fontweight='bold', color='navy')

# Histplot
plt.subplot(212)
sns.histplot(x=ctc_filtered_df['ctc'], kde=True, color='skyblue')
plt.title('Histplot of CTC', fontsize=14, fontweight='bold', color='navy')

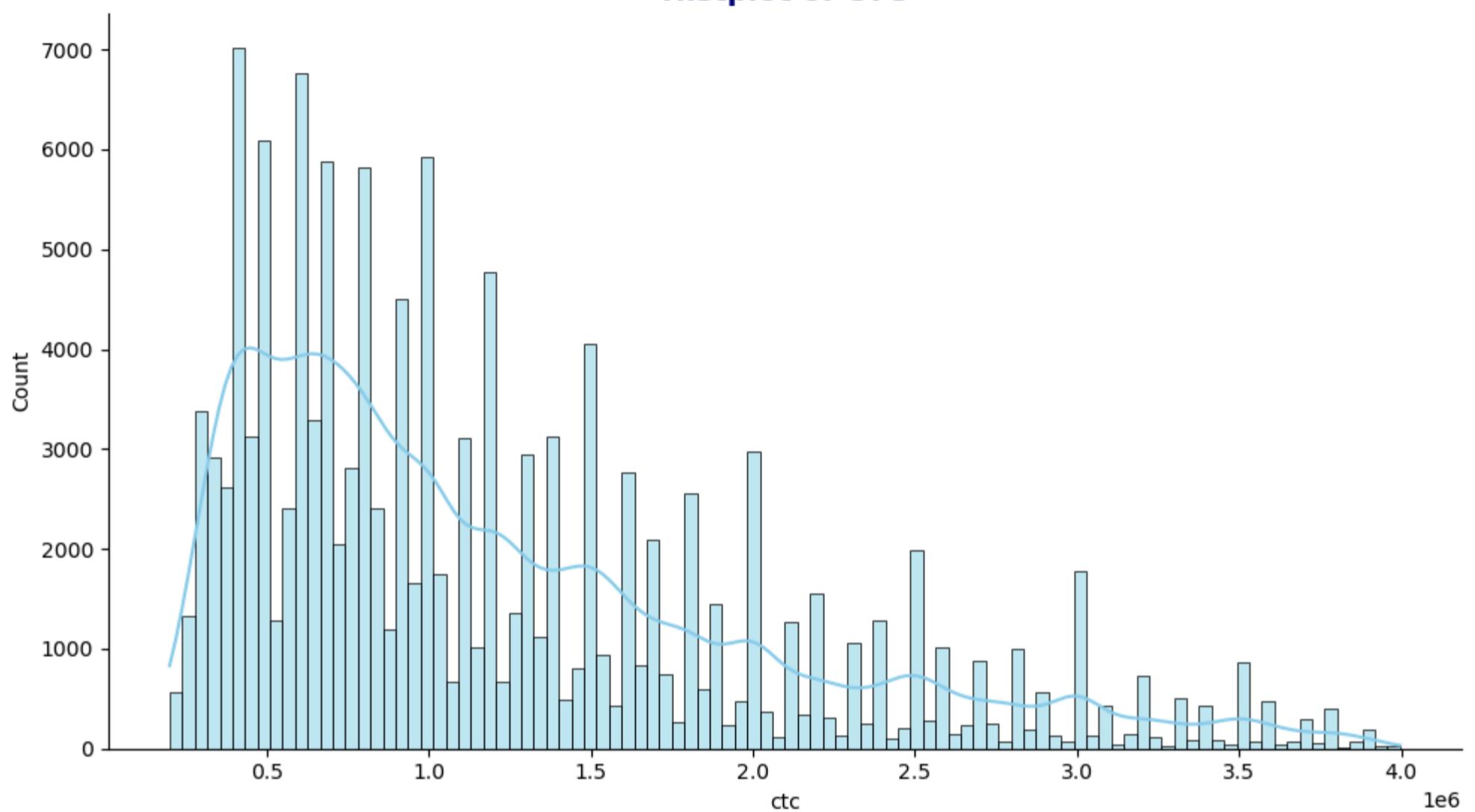
sns.despine()
plt.tight_layout()
plt.show()
```

```
Out[76]: <Figure size 1000x1200 with 0 Axes>
Out[76]: Text(0.5, 0.98, 'Distribution of CTC')
Out[76]: <Axes: >
Out[76]: <Axes: xlabel='ctc'>
Out[76]: Text(0.5, 1.0, 'Boxplot of CTC')
Out[76]: <Axes: >
Out[76]: <Axes: xlabel='ctc', ylabel='Count'>
Out[76]: Text(0.5, 1.0, 'Histplot of CTC')
```

Distribution of CTC Boxplot of CTC



Histplot of CTC



🔍 Insights

- Backend Architect has the Highest Average CTC
- Support Engineer has Lowest Average CTC

4.2 - Bivariate Analysis

4.2.1 - CTC vs Job Position

In [77]: # CTC vs Job Position

```
# Filter out 'other' job positions and calculate the average CTC for each job position
job_position_ctc_summary = (
    ctc_filtered_df[ctc_filtered_df['job_position'] != 'other'] # Exclude 'other' job positions
    .groupby('job_position', as_index=False) # Group by 'job_position'
    .agg(avg_ctc=('ctc', 'mean'), count=('ctc', 'count')) # Calculate average CTC and count
)
```

```

# Select the top 20 job positions by count
top_20_positions_ctc = (
    job_position_ctc_summary
    .sort_values(by='count', ascending=False) # Sort by count in descending order
    .head(20) # Take the top 20
)

# Sort by average CTC for plotting
top_20_positions_ctc_sorted = top_20_positions_ctc.sort_values(by='avg_ctc')

# Set 'job_position' as the index for better visualization
top_20_positions_ctc_sorted.set_index('job_position', inplace=True)

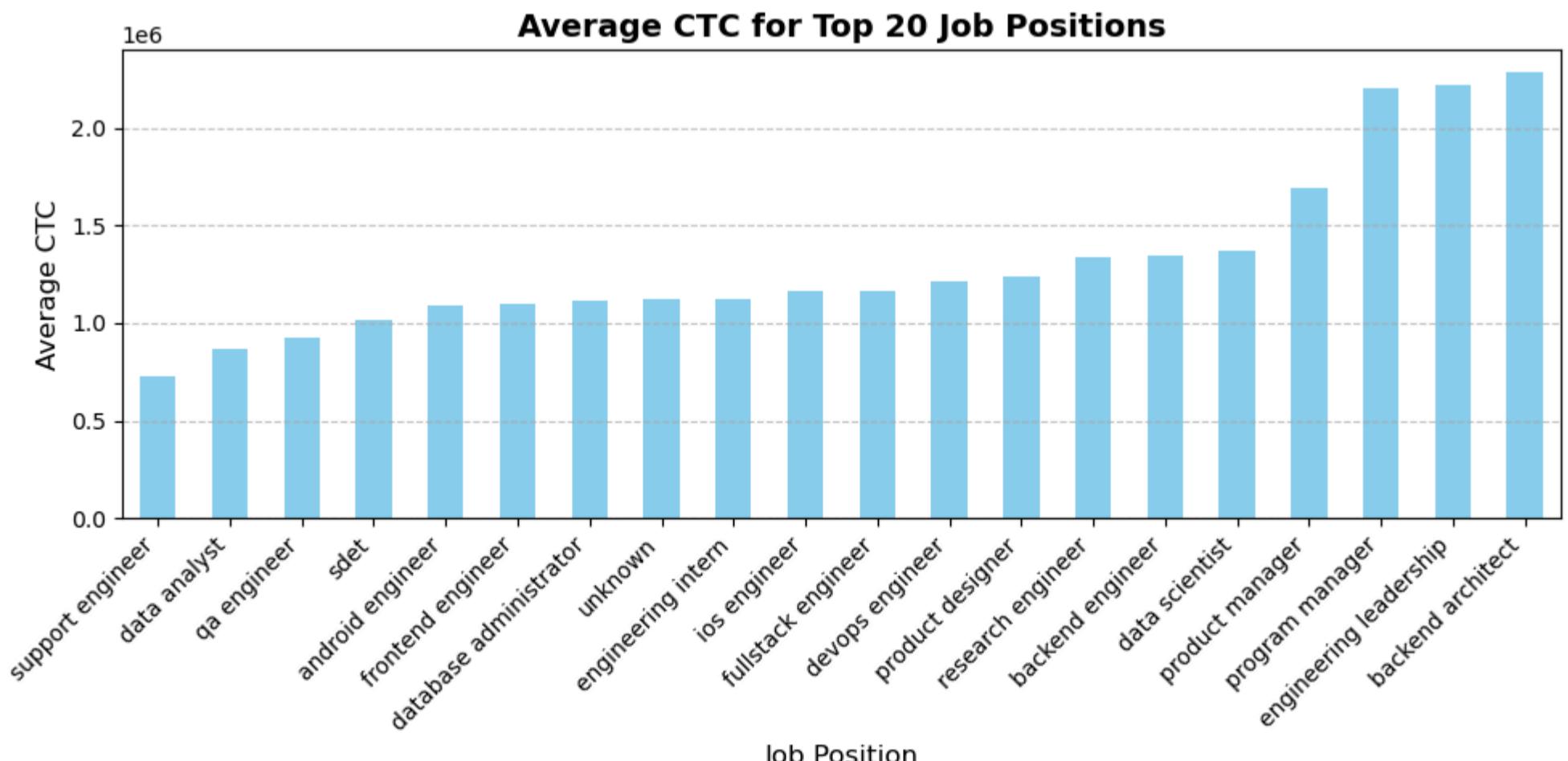
# Plot a bar chart of the average CTC for the top 20 job positions
plt.figure(figsize=(10, 5))
top_20_positions_ctc_sorted['avg_ctc'].plot(kind='bar', color='skyblue')
plt.title('Average CTC for Top 20 Job Positions', fontsize=14, fontweight='bold')
plt.ylabel('Average CTC', fontsize=12)
plt.xlabel('Job Position', fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

```

```

Out[77]: <Figure size 1000x500 with 0 Axes>
Out[77]: <Axes: xlabel='job_position'>
Out[77]: Text(0.5, 1.0, 'Average CTC for Top 20 Job Positions')
Out[77]: Text(0, 0.5, 'Average CTC')
Out[77]: Text(0.5, 0, 'Job Position')
Out[77]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19]),
 [Text(0, 0, 'support engineer'),
  Text(1, 0, 'data analyst'),
  Text(2, 0, 'qa engineer'),
  Text(3, 0, 'sdet'),
  Text(4, 0, 'android engineer'),
  Text(5, 0, 'frontend engineer'),
  Text(6, 0, 'database administrator'),
  Text(7, 0, 'unknown'),
  Text(8, 0, 'engineering intern'),
  Text(9, 0, 'ios engineer'),
  Text(10, 0, 'fullstack engineer'),
  Text(11, 0, 'devops engineer'),
  Text(12, 0, 'product designer'),
  Text(13, 0, 'research engineer'),
  Text(14, 0, 'backend engineer'),
  Text(15, 0, 'data scientist'),
  Text(16, 0, 'product manager'),
  Text(17, 0, 'program manager'),
  Text(18, 0, 'engineering leadership'),
  Text(19, 0, 'backend architect')])

```



4.2.2 - CTC vs Experience

```

In [78]: # CTC vs Years of Experience

# Scatter Plot for CTC vs YOE
plt.figure(figsize=(12, 6))
sns.scatterplot(

```

```

    data=ctc_filtered_df,
    x='yoe', # Years of Experience
    y='ctc', # CTC
    palette='viridis', # Color palette
    alpha=0.7 # Transparency for overlapping points
)

# Add titles and Labels
plt.title('CTC vs YOE (Bivariate Analysis)', fontsize=16, fontweight='bold')
plt.xlabel('Years of Experience (YOE)', fontsize=12)
plt.ylabel('CTC (Cost to Company)', fontsize=12)

# Add grid for better visualization
plt.grid(axis='both', linestyle='--', alpha=0.5)

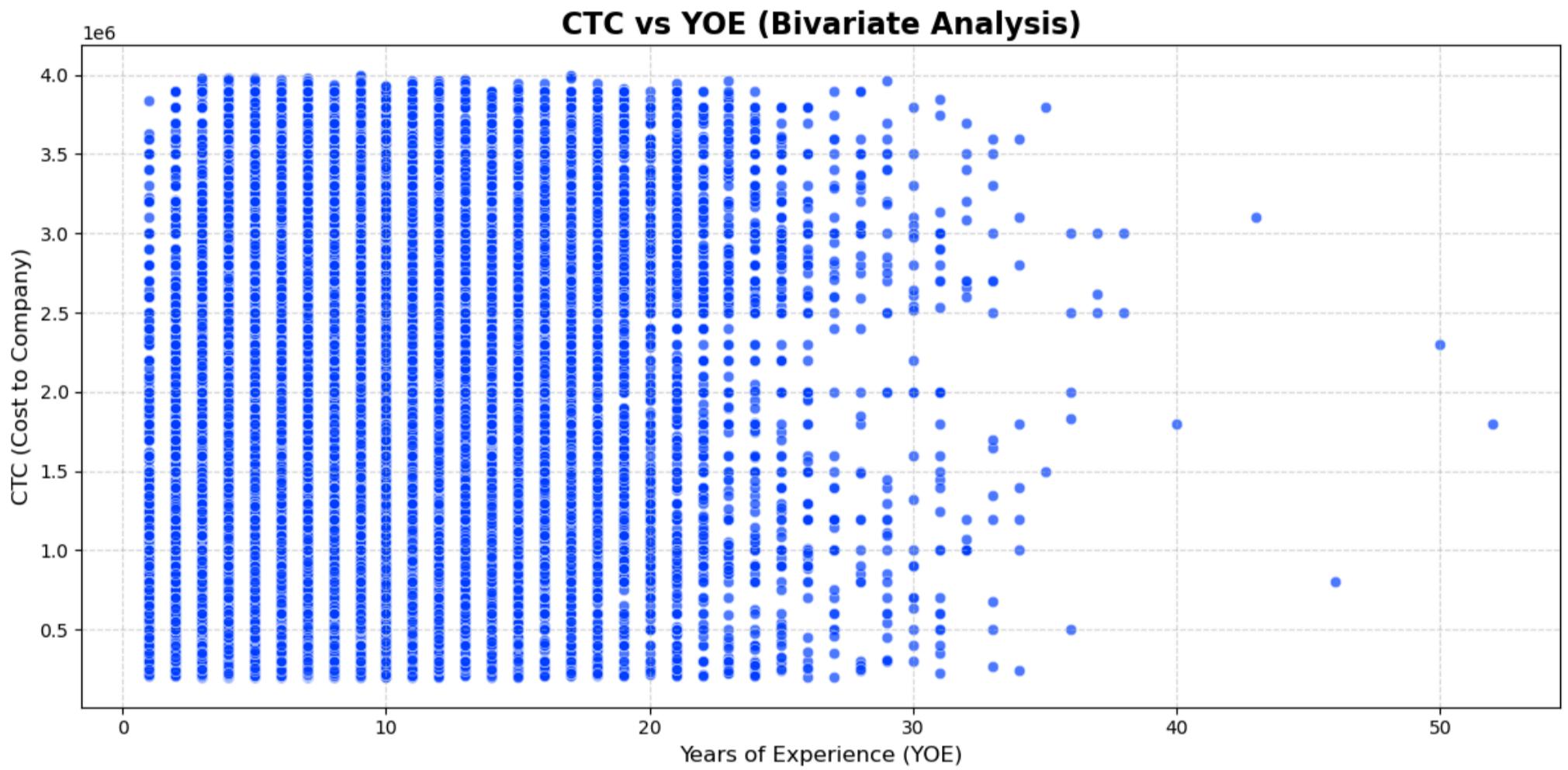
# Improve Layout
plt.tight_layout()
plt.show()

```

```

Out[78]: <Figure size 1200x600 with 0 Axes>
Out[78]: <Axes: xlabel='yoe', ylabel='ctc'>
Out[78]: Text(0.5, 1.0, 'CTC vs YOE (Bivariate Analysis)')
Out[78]: Text(0.5, 0, 'Years of Experience (YOE)')
Out[78]: Text(0, 0.5, 'CTC (Cost to Company)')

```



4.2.3 - Top 10 Companies with Tier 1 Candidates

```

In [79]: # Filtering for Tier 1 candidates and grouping by company_hash
top_tier1_companies = (
    df[df['tier_flag'] == 1]
    .groupby('company_hash')
    .size() # Counting occurrences
    .reset_index(name='count') # Renaming the count column
    .sort_values(by='count', ascending=False) # Sorting by count
    .head(10) # Selecting the top 10 companies
)

# Plotting a count plot
plt.figure(figsize=(10, 6))
sns.barplot(
    data=top_tier1_companies,
    x='count',
    y='company_hash',
    palette='viridis'
)

# Adding titles and Labels
plt.title('Top 10 Companies with Tier 1 Candidates', fontsize=16, fontweight='bold')
plt.xlabel('Number of Tier 1 Candidates', fontsize=12)
plt.ylabel('Company Hash', fontsize=12)

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

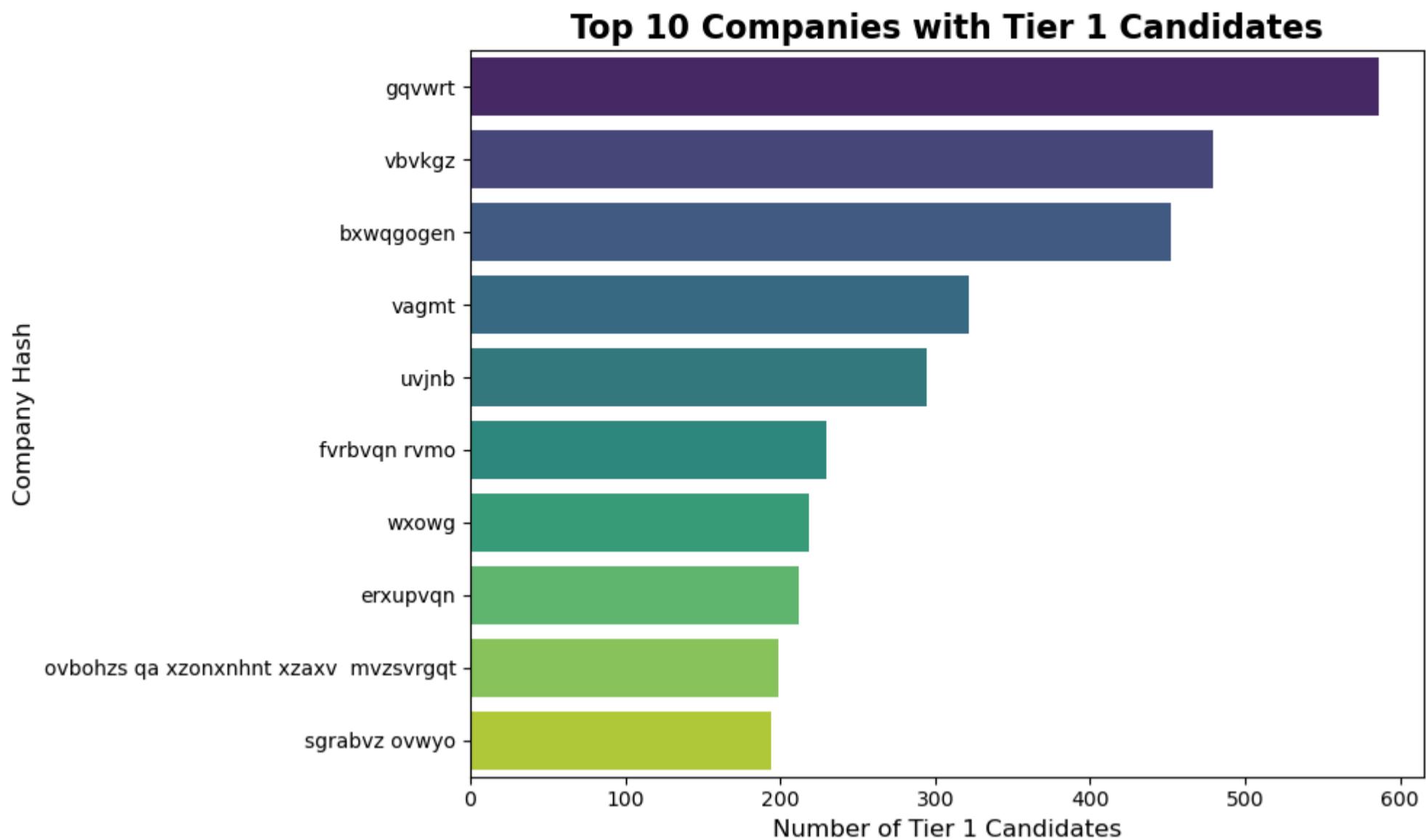
```

```

Out[79]: <Figure size 1000x600 with 0 Axes>
Out[79]: <Axes: xlabel='count', ylabel='company_hash'>

```

```
Out[79]: Text(0.5, 1.0, 'Top 10 Companies with Tier 1 Candidates')
Out[79]: Text(0.5, 0, 'Number of Tier 1 Candidates')
Out[79]: Text(0, 0.5, 'Company Hash')
```



💡 Insights

- Company 'gqvwrt' has close to 600 Tier 1 Candidates

```
In [80]: ctc_filtered_df.columns
```

```
Out[80]: Index(['company_hash', 'email_hash', 'orgyear', 'ctc', 'job_position',
       'ctc_updated_year', 'yoe', 'num_jobs', 'growth', 'ctc_yoe_ratio',
       'mean_ctc_exp', 'class_flag', 'mean_ctc_position', 'designation_flag',
       'mean_ctc_company', 'tier_flag', 'is_other', 'is_developer', 'is_tech',
       'is_management', 'is_sales', 'is_non_coder', 'ctc_bin'],
      dtype='object')
```

```
In [81]: # Selecting relevant columns and filtering for Tier 3 candidates
```

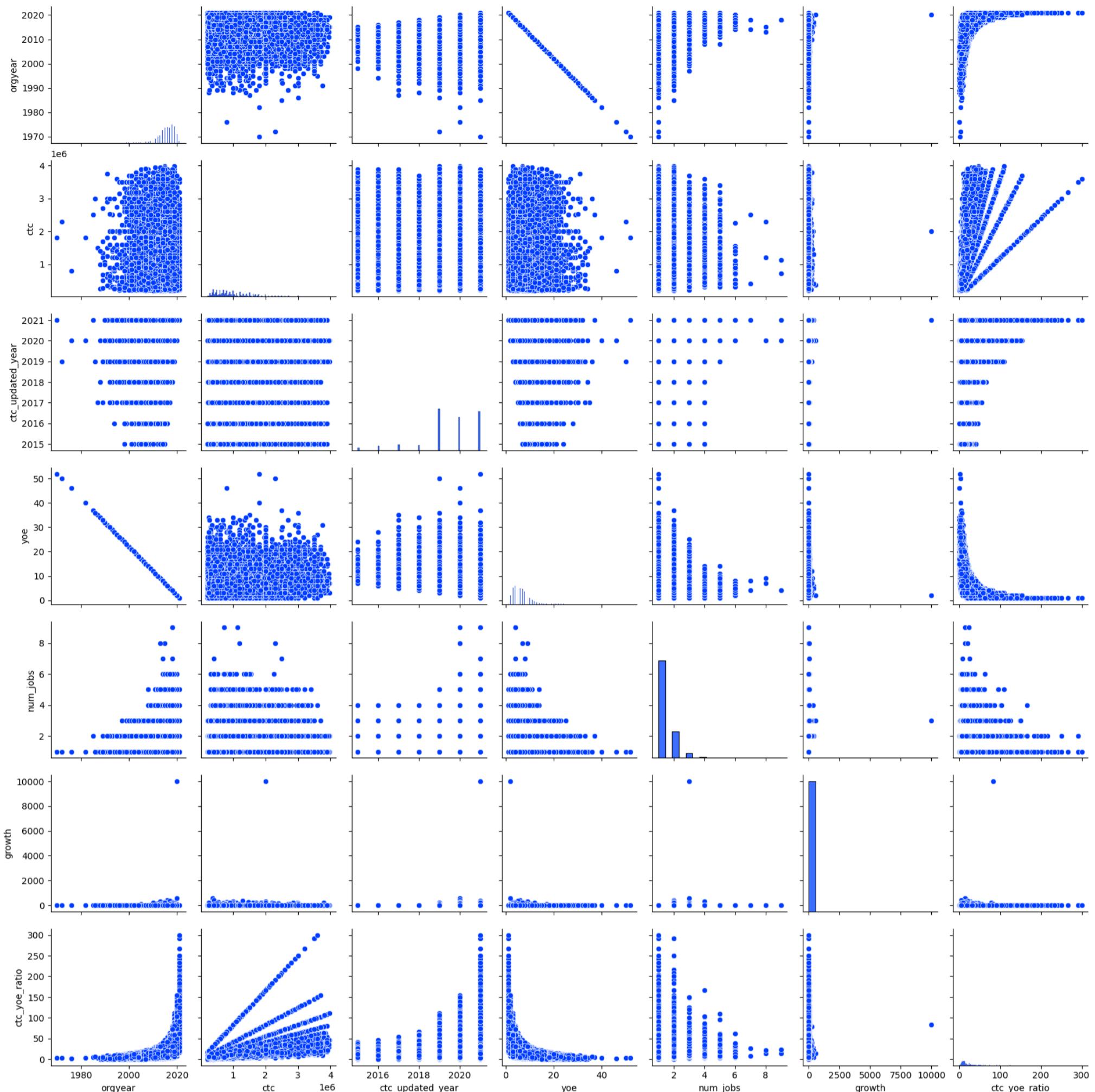
```
cols = [
    'orgyear',
    'ctc',
    'ctc_updated_year',
    'yoe',
    'num_jobs',
    'growth',
    'ctc_yoe_ratio',
]

# Filtering the DataFrame for Tier 3 candidates
tier3_filtered_df = ctc_filtered_df[ctc_filtered_df['tier_flag'] == 3]

# Creating the pairplot
sns.pairplot(tier3_filtered_df[cols])

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

```
Out[81]: <seaborn.axisgrid.PairGrid at 0x2ed1053e2f0>
```



💡 Insights

- The majority of Tier 3 candidates tend to remain with the same company without frequent switches.
- While most candidates joined their organizations around 2020, there are employees with joining dates extending back to before the 2000s.
- A significant variance in CTC is observed relative to years of experience, indicating diverse salary structures.

4.2.4 - CTC vs Categories

```
In [82]: # Define the columns you want to analyze
cols = ['is_developer', 'is_tech', 'is_management', 'is_sales', 'is_non_coder']

# Create subplots for each group comparison
fig, ax = plt.subplots(1, 5, figsize=(20, 6))

# Loop through the columns and create a grouped bar chart for each
for i, col in enumerate(cols):
    # Grouping the data by 'ctc_bin' and the binary column (e.g., 'is_developer', 'is_tech', etc.)
    grouped_data = df.groupby(['ctc_bin', col]).size().reset_index(name='count')

    # Pivot the grouped data to create a DataFrame suitable for a bar plot
    pivot_data = grouped_data.pivot(index='ctc_bin', columns=col, values='count').fillna(0)

    # Plot the data on the appropriate subplot
    pivot_data.plot(kind='bar', ax=ax[i])

    # Set the title for each subplot
    ax[i].set_title(f'CTC Bin vs {col.capitalize()}', fontsize=14, fontweight='bold')
    ax[i].set_ylabel('Count', fontsize=12)
```

```

        ax[i].set_xlabel('CTC Bin', fontsize=12)

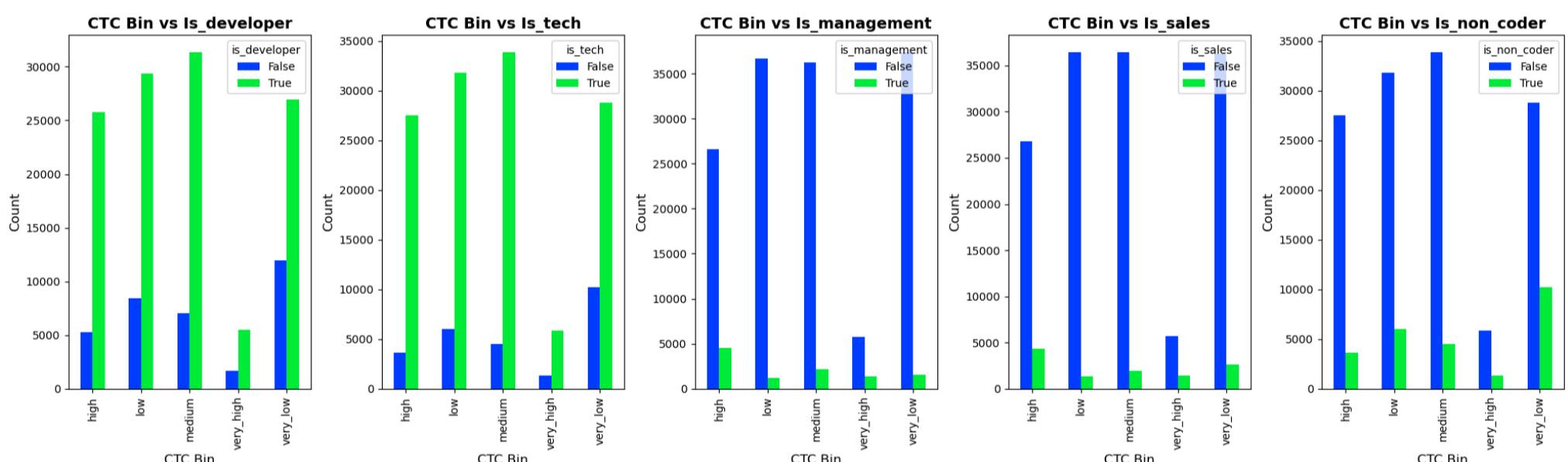
# Adjust layout and display the plots
plt.tight_layout()
plt.show()

```

```

Out[82]: <Axes: xlabel='ctc_bin'>
Out[82]: Text(0.5, 1.0, 'CTC Bin vs Is_developer')
Out[82]: Text(0, 0.5, 'Count')
Out[82]: Text(0.5, 0, 'CTC Bin')
Out[82]: <Axes: xlabel='ctc_bin'>
Out[82]: Text(0.5, 1.0, 'CTC Bin vs Is_tech')
Out[82]: Text(0, 0.5, 'Count')
Out[82]: Text(0.5, 0, 'CTC Bin')
Out[82]: <Axes: xlabel='ctc_bin'>
Out[82]: Text(0.5, 1.0, 'CTC Bin vs Is_management')
Out[82]: Text(0, 0.5, 'Count')
Out[82]: Text(0.5, 0, 'CTC Bin')
Out[82]: <Axes: xlabel='ctc_bin'>
Out[82]: Text(0.5, 1.0, 'CTC Bin vs Is_sales')
Out[82]: Text(0, 0.5, 'Count')
Out[82]: Text(0.5, 0, 'CTC Bin')
Out[82]: <Axes: xlabel='ctc_bin'>
Out[82]: Text(0.5, 1.0, 'CTC Bin vs Is_non_coder')
Out[82]: Text(0, 0.5, 'Count')
Out[82]: Text(0.5, 0, 'CTC Bin')

```



```
In [83]: num_column = ['orgyear', 'ctc','ctc_updated_year', 'yoe', 'num_jobs', 'growth', 'ctc_yoe_ratio', 'mean_ctc_exp', 'mean_ctc_position', 'mean_ctc_company']
```

```
In [84]: df[num_column].corr()
```

| | orgyear | ctc | ctc_updated_year | yoe | num_jobs | growth | ctc_yoe_ratio | mean_ctc_exp | mean_ctc_position | mean_ctc_company | |
|--------------------------|---------|--------|------------------|--------|----------|--------|---------------|--------------|-------------------|------------------|--------|
| orgyear | 1.000 | -0.019 | | 0.273 | -1.000 | 0.142 | 0.006 | 0.069 | -0.023 | -0.010 | -0.002 |
| ctc | -0.019 | 1.000 | | 0.028 | 0.019 | -0.029 | 0.024 | 0.813 | 0.812 | 0.708 | 0.616 |
| ctc_updated_year | 0.273 | 0.028 | | 1.000 | -0.273 | 0.267 | 0.009 | 0.046 | 0.027 | 0.028 | 0.024 |
| yoe | -1.000 | 0.019 | | -0.273 | 1.000 | -0.142 | -0.006 | -0.069 | 0.023 | 0.010 | 0.002 |
| num_jobs | 0.142 | -0.029 | | 0.267 | -0.142 | 1.000 | 0.028 | -0.018 | -0.024 | -0.020 | -0.014 |
| growth | 0.006 | 0.024 | | 0.009 | -0.006 | 0.028 | 1.000 | 0.020 | 0.018 | 0.008 | 0.008 |
| ctc_yoe_ratio | 0.069 | 0.813 | | 0.046 | -0.069 | -0.018 | 0.020 | 1.000 | 0.630 | 0.545 | 0.480 |
| mean_ctc_exp | -0.023 | 0.812 | | 0.027 | 0.023 | -0.024 | 0.018 | 0.630 | 1.000 | 0.872 | 0.759 |
| mean_ctc_position | -0.010 | 0.708 | | 0.028 | 0.010 | -0.020 | 0.008 | 0.545 | 0.872 | 1.000 | 0.870 |
| mean_ctc_company | -0.002 | 0.616 | | 0.024 | 0.002 | -0.014 | 0.008 | 0.480 | 0.759 | 0.870 | 1.000 |

```
In [85]: # Correlation Heatmap
```

```

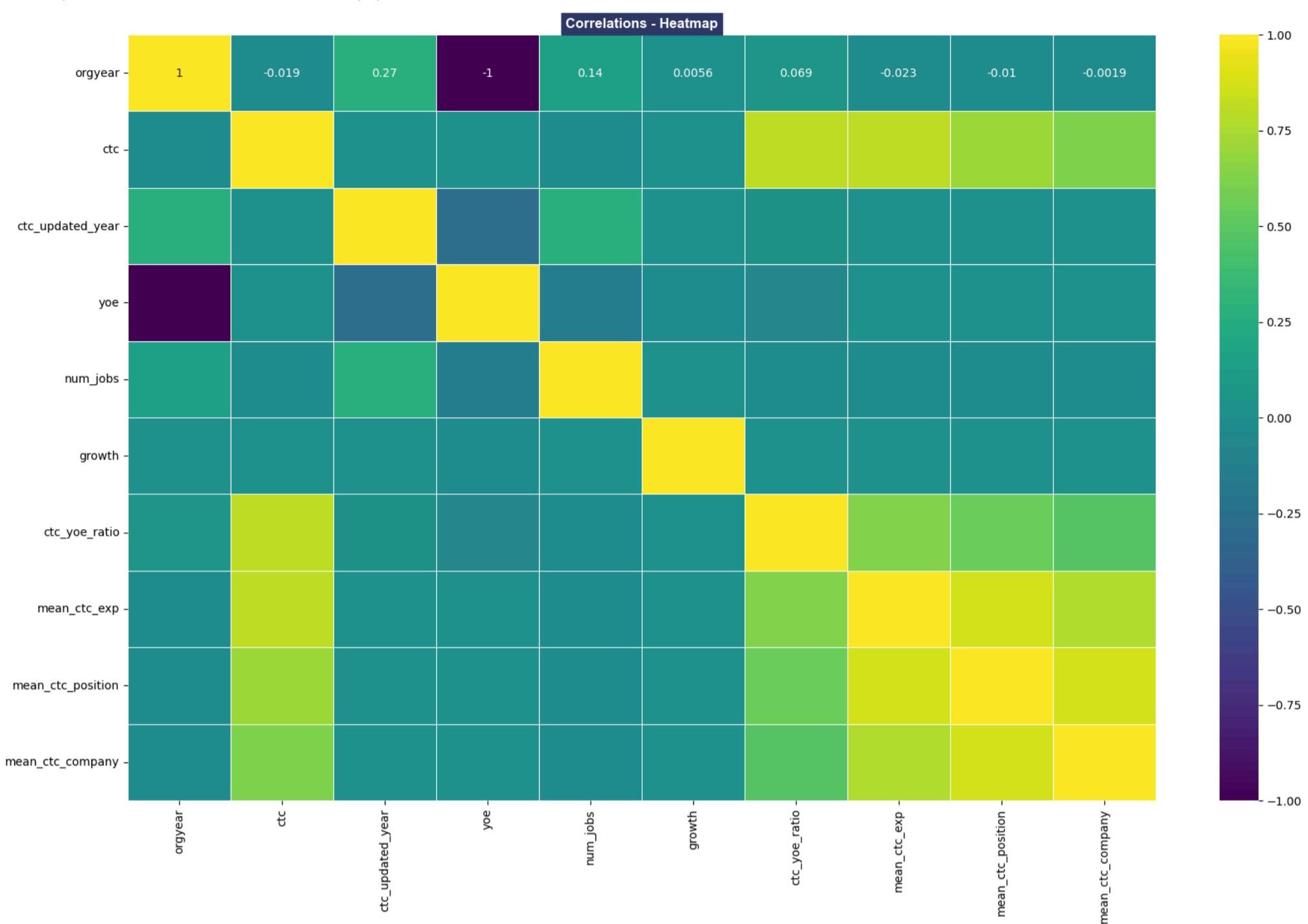
plt.figure(figsize=(20,12))

sns.heatmap(df[num_column].corr(), annot=True, cmap='viridis', linewidth=.5, linecolor='w')
plt.title('Correlations - Heatmap', fontsize=12, fontfamily='Arial', fontweight='bold', backgroundcolor=Colors_Palette[0], color='w')
plt.show()

```

```
Out[85]: <Figure size 2000x1200 with 0 Axes>
```

Out[85]: <Axes: >
Out[85]: Text(0.5, 1.0, 'Correlations - Heatmap')



5. Manual Clustering

Already Performed in Earlier Steps ---> Data Aggregate, Merging the Df, and Flagging

5.1 - Questions Based on the Manual Clustering Done So Far

5.1.1 - Q - Top 10 Employees(Earning more than most of the employees in the Company)

```
In [86]: # Filter rows where 'tier_flag' is 1, sort by 'ctc' in descending order, and get the top 10
top_10_tier_1 = df[df['tier_flag'] == 1].sort_values(by='ctc', ascending=False).head(10)
top_10_tier_1
```

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth |
|--------|--------------------------------------|---|---------|-----------|--------------------|------------------|-----|----------|--------|
| 150985 | obvqnuqxdwgb | 5b4bed51797140db4ed52018a979db1e34cee49e27b488... | 2015 | 255555555 | fullstack engineer | 2016 | 7 | 1 | 0.00 |
| 114573 | grd sqghu | 1f8e216b2328e7764f79dbd66deaf008b409870ae992fe... | 2019 | 200000000 | other | 2020 | 3 | 1 | 0.00 |
| 112228 | psxor | 1b9cb9fd05dd794efc20da0f5e873f1afc92767306166b... | 2017 | 200000000 | other | 2020 | 5 | 1 | 0.00 |
| 112355 | nvnv wgzohrvnzwj otqcxwto | 1ad9ecf5763e6fcc1e7204355c2645e12d3baaf7a7c083... | 2015 | 200000000 | data analyst | 2020 | 7 | 1 | 0.00 |
| 67703 | otre tburgjta | f63e63a8cc3db37c89a49f0498d0731f8c9e2e27943e85... | 2015 | 200000000 | android engineer | 2020 | 7 | 1 | 0.00 |
| 112367 | yaew mvzp | 1ad3d8c2b855e9cbe6bb187e8140a07ed8a8b29d275ae0... | 2017 | 200000000 | other | 2020 | 5 | 1 | 0.00 |
| 136303 | gnytq | 82b6acf155f320e9e70dc6d1b6fd33449dc731f4c07bf... | 2008 | 200000000 | other | 2020 | 14 | 1 | 0.00 |
| 112489 | cxcg bgmxrt xzaxv ucn rna | 1b1a39899354f398255dcbef537ed636bc8ce242fab0ea... | 2018 | 200000000 | other | 2020 | 4 | 1 | 0.00 |
| 135865 | qtrxvzwt lxr xzegwgbbr rxbxnta | 81d9854a6797a2ef69f8a9fa358c332d7da3297163507b... | 2015 | 200000000 | unknown | 2020 | 7 | 1 | 0.00 |
| 112619 | stzuvnn | 1c6bc8b95225bf255f939a64f9d60f84371f16eb621f3a... | 2014 | 200000000 | other | 2020 | 8 | 1 | 0.00 |

10 rows × 23 columns

5.1.2 - Q - Top 10 employees of Data Science in each Company earning more than their peers - Class 1

In [87]:

```
# Filter rows where 'job_position' starts with 'data' and 'class_flag' is 1,
# sort by 'ctc' in descending order, and get the top 10
filtered_data = df[df['job_position'].str.startswith('data') & (df['class_flag'] == 1)]
top_10_DS = filtered_data.sort_values(by='ctc', ascending=False).head(10)
top_10_DS
```

Out[87]:

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth |
|--------|----------------------------------|---|---------|-----------|--------------|------------------|-----|----------|--------|
| 145840 | nvnv wgzohrvzjw otqcxwto | 655da5cd99f1ba4ad249dade5039b914023484fb7f3959... | 2017 | 200000000 | data analyst | 2020 | 5 | 1 | 0.00 |
| 72275 | ytfrtnn uvwpvqa tzntquqxot | c888824e687a535d1bd2486ae28d67e414b21b09cbee61... | 2015 | 200000000 | data analyst | 2020 | 7 | 1 | 0.00 |
| 112355 | nvnv wgzohrvzjw otqcxwto | 1ad9ecf5763e6fcc1e7204355c2645e12d3baaf7a7c083... | 2015 | 200000000 | data analyst | 2020 | 7 | 1 | 0.00 |
| 123241 | ntwy bvyxzaqv | 7a723f5b71698674b79bd2195c3bb58d3fcf4ddb75a04... | 2019 | 200000000 | data analyst | 2021 | 3 | 1 | 0.00 |
| 48160 | vbkvgz | 95c8bdd3d10f967f6fb32f280b0b37fb5db2359918ba... | 2016 | 200000000 | data analyst | 2020 | 6 | 1 | 0.00 |
| 41075 | wgzahtzn | 9ce2995b2221fe627e861dae9d0603872cce8cc128390... | 2016 | 200000000 | data analyst | 2020 | 6 | 1 | 0.00 |
| 124098 | gnytq | 6d4a5d19e889596252b038ee0409510aec8c0b32007fb9... | 2017 | 200000000 | data analyst | 2020 | 5 | 1 | 0.00 |
| 139296 | nvnv wgzohrvzjw otqcxwto | 59316048d113539202325e05af9b66620255ba84eab635... | 2015 | 200000000 | data analyst | 2020 | 7 | 1 | 0.00 |
| 84049 | nvnv wgzohrvzjw otqcxwto | d42cf076c3915454a083788ea22df68a65a294563adfa3... | 2012 | 200000000 | data analyst | 2020 | 10 | 1 | 0.00 |
| 51040 | ytqt ntwyzgrgsxto | 98a90272cbb6e6e9ca94981824f3465f3d34567cb065f... | 2015 | 200000000 | data analyst | 2020 | 7 | 1 | 0.00 |

10 rows × 23 columns

5.1.3 - Q - Bottom 10 employees of Data Science in each Company earning less than their peers - Class 3

In [88]:

```
# Filter rows where 'job_position' starts with 'data' and 'class_flag' is 3,
# sort by 'ctc' in Ascending order, and get the Bottom 10
filtered_data = df[df['job_position'].str.startswith('data') & (df['class_flag'] == 3)]
Bottom_10_DS = filtered_data.sort_values(by='ctc', ascending=True).head(10)
Bottom_10_DS
```

Out[88]:

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth | c |
|--------|--------------------------------|---|---------|-------|------------------------|------------------|-----|----------|--------|---|
| 102215 | nvnv wgzohrvzjw otqcxwto | 3becd3658bc0d426f8867142eb6cbd7e9ca9f43b572794... | 2018 | 3500 | database administrator | 2019 | 4 | 1 | 0.000 | |
| 51390 | wgszxkvzn | abf69e786daa23f50d3142b653235e2c030b3b180b07d2... | 2014 | 4300 | data analyst | 2019 | 8 | 1 | 0.000 | |
| 4717 | onhatzn | bd9c04a574090e05b366a81cdb2f3f565d0c60fa8b1647... | 2018 | 6000 | data scientist | 2019 | 4 | 3 | 0.000 | |
| 89200 | nvnv wgzohrvzjw otqcxwto | 3175d03fd4618eb293d6f5a1d13d42a0c79f68e9acaaa3... | 2020 | 7500 | data scientist | 2020 | 2 | 1 | 0.000 | |
| 99457 | vqxosrgmvr | 3675f79c7e05de96ccf189c818b84b487cb1aa3f6b80e8... | 2015 | 8800 | data scientist | 2019 | 7 | 1 | 0.000 | |
| 61349 | sggsrt | fb64af615420e06d46a1965f59068b34460fb3cbe70541... | 2018 | 10000 | data scientist | 2021 | 4 | 1 | 0.000 | |
| 150728 | zgn vuurxwvmrt wwwghzn | 598b129dd60613cd01a43ea7b946a50a1eb781489feb12... | 2020 | 10000 | database administrator | 2021 | 2 | 1 | 0.000 | |
| 83446 | ahzzyhbmj | d32c344816921ae7be310ba7630c6e267703b2fc4e6a6c... | 2016 | 31000 | data scientist | 2019 | 6 | 1 | 0.000 | |
| 83770 | zvz | d277c90e2b97fb51657936b66b68eee2e52bd4d8d98e1d... | 2015 | 46500 | data scientist | 2016 | 7 | 1 | 0.000 | |
| 77788 | tqxwoogz | d8b6c25488ee764c6dbd20c56ad8dca794b6347d6448e2... | 2017 | 49000 | data scientist | 2017 | 5 | 1 | 0.000 | |

10 rows × 23 columns

5.1.4 - Q - Top 10 Employees in Each Company - X Department - having 5/6/7 Years of Experience earning more than their peers - Tier X

In [89]:

```
# Filter the DataFrame for tier_flag = 1, designation_flag = 1, and yoe between 5 and 7
filtered_df = df[(df['tier_flag'] == 1) &
                  (df['designation_flag'] == 1) &
                  (df['yoe'] >= 5) &
                  (df['yoe'] <= 7)]
```

```

# Sort the DataFrame by company_hash and ctc in descending order,
# and apply the row number logic (group by company_hash and order by ctc desc)
filtered_df['rn'] = filtered_df.sort_values(['company_hash', 'ctc'], ascending=[True, False]) \
    .groupby('company_hash') \
    .cumcount() + 1

# Select top 10 rows for each company (rn <= 10)
top_10_records = filtered_df[filtered_df['rn'] <= 10]

# Sort the final DataFrame by company_hash, job_position, and ctc (desc)
final_df = top_10_records.sort_values(['company_hash', 'job_position', 'ctc'], ascending=[True, True, False])

# Display the final result
final_df

```

Out[89]:

| | company_hash | | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth |
|--------|--------------|---|------------|---------|----------|--------------------|------------------|------|----------|---------|
| 88690 | 1bs | 31db7b806a82aac024462d4c97e70eed918bf8c3193b7c... | | 2016 | 3750000 | backend engineer | | 2020 | 6 | 1 0.000 |
| 138984 | 1bs | 579d9c719cc05b885297256c122585a8f71871f9079c6f... | | 2015 | 2930000 | backend engineer | | 2019 | 7 | 1 0.000 |
| 42573 | 1bs | 9c02076a74a2b8a64a6e003fa0a2e4115fc717dacb3585... | | 2016 | 2320000 | backend engineer | | 2020 | 6 | 1 0.000 |
| 99138 | 1bs | 38dfe791fc911da418b67aa989a6aa7f00b8c680c6d4e1... | | 2015 | 2000000 | backend engineer | | 2019 | 7 | 1 0.000 |
| 72576 | 1bs | c97fd1612080086b898e440529c86325ae8ddf2e9a0b60... | | 2015 | 1800000 | backend engineer | | 2019 | 7 | 1 0.000 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 60073 | zxzlvwvqn | e2377e7ee0d53d2e3a45b9687fdc9c08b136b1dc470806... | | 2017 | 2300000 | fullstack engineer | | 2020 | 5 | 1 0.000 |
| 5710 | zxztrtvuo | 303aab779b6daea19317b430f600b0e8543f4c66ca3251... | | 2015 | 11950000 | backend engineer | | 2020 | 7 | 2 8.700 |
| 10537 | zxztrtvuo | 234f0f52e89b20231f5685551b865a08a5634189e1ffec... | | 2015 | 2500000 | backend engineer | | 2021 | 7 | 2 0.667 |
| 21763 | zxztrtvuo | 16c227291d7c4f151b52599cc15e1ddd6f7e12a694753c... | | 2015 | 1780000 | backend engineer | | 2020 | 7 | 2 0.271 |
| 98282 | zxztrtvuo | 3879b9a1e356ed20363ffd6871207eb908b38c864a2db... | | 2017 | 1500000 | fullstack engineer | | 2019 | 5 | 1 0.000 |

4899 rows × 24 columns

5.1.5 - Q - Top 10 Companies (Based on their CTC)

```

In [90]: # Group the DataFrame by 'company_hash' and calculate the average of 'ctc' for each company
avg_ctc_df = df.groupby('company_hash')['ctc'].mean().reset_index()

# Sort the DataFrame by 'avg_ctc' in descending order and get the top 10 companies
top_10_avg_ctc = avg_ctc_df.sort_values(by='ctc', ascending=False).head(10)

# Display the result
top_10_avg_ctc

```

Out[90]:

| | company_hash | ctc |
|-------|---|----------------|
| 29688 | whmxw rgsxwo uqxcvnt rxbxnta | 1000150000.000 |
| 1189 | aveegaxr xzntqzvnxzvr hzxctqoxnj | 2500000000.000 |
| 14649 | ogzv wgrrtst ge ntwyzgrgsjvzzv hzxctqoxnj | 2000000000.000 |
| 25708 | vayxxuv ntwyzgrgsxto ucn rna | 2000000000.000 |
| 25038 | uvqp wgbuhntq ojontb xzw | 2000000000.000 |
| 8627 | i wgzztin mhoxztoo ogrhnxgzo ucn rna | 2000000000.000 |
| 17103 | oxburjyq ogrhnxgzo rru | 2000000000.000 |
| 4991 | dqq avnv tdwyvzst | 2000000000.000 |
| 592 | anaw tduqtoo rxbxnta | 2000000000.000 |
| 13319 | nrvtn ouvwt xzw | 2000000000.000 |

5.1.6 - Q - Top 2 Positions in Every Company (based on their CTC)

```

In [91]: # First, we sort the DataFrame by 'company_hash' and 'ctc' in descending order to simulate the ranking
df_sorted = df.sort_values(by=['company_hash', 'ctc'], ascending=[True, False])

# Then, we group the DataFrame by 'company_hash' and assign a row number (rank) to each 'job_position' within each company
df_sorted['rn'] = df_sorted.groupby('company_hash').cumcount() + 1

```

```
# Filter the rows where the row number is less than or equal to 2 (i.e., select top 2 for each company)
top_2_ctc_per_company = df_sorted[df_sorted['rn'] <= 2]

# Select the relevant columns and sort the DataFrame by 'company_hash' and 'ctc' in descending order
result = top_2_ctc_per_company[['company_hash', 'job_position', 'ctc']].sort_values(by=['company_hash', 'ctc'], ascending=[False, False])

# Display the result
result
```

Out[91]:

| | company_hash | job_position | ctc |
|--------|-------------------------------|--------------------|---------|
| 84070 | zzgato | unknown | 130000 |
| 71133 | zzb ztdnstz vacxogqj ucn rna | fullstack engineer | 600000 |
| 133579 | zz | other | 1370000 |
| 78987 | zz | other | 500000 |
| 141069 | zyvzwt wgzohrnzs tzsxttqo | frontend engineer | 940000 |
| ... | ... | ... | ... |
| 96750 | 05mz exzytvry uqxcvnt rxbxnta | backend engineer | 1100000 |
| 59912 | 01 ojztqsj | frontend engineer | 830000 |
| 135778 | 01 ojztqsj | android engineer | 270000 |
| 39426 | 0000 | other | 300000 |
| 54190 | 0 | other | 100000 |

45032 rows × 3 columns

6. Standardization and Encoding

Label Encoding Part was already done in Feature Engineering Section --> except for the ctc_bin

In [92]: df_Clustering = df.copy()

In [93]: df.head()

Out[93]:

| | company_hash | email_hash | orgyear | ctc | job_position | ctc_updated_year | yoe | num_jobs | growth | ctc_yc |
|---|------------------------|---|---------|---------|--------------------|------------------|-----|----------|--------|--------|
| 0 | cvrhtgbtzhnb | 298528ce3160cc761e4dc37a07337ee2e0589df251d736... | 2018 | 720000 | backend engineer | 2020 | 4 | 9 | 0.029 | |
| 1 | wgcxvb ntwyzgrgsxto | 3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94... | 2018 | 1130000 | engineering intern | 2021 | 4 | 9 | 0.041 | |
| 2 | ihvrwgb | 6842660273f70e9aa239026ba33bfe82275d6ab0d20124... | 2017 | 2400000 | qa engineer | 2020 | 5 | 9 | 0.200 | |
| 3 | gqvwr | 4818edfd67ed8563dde5d083306485d91d19f4f1c95d19... | 2015 | 1200000 | backend architect | 2020 | 7 | 8 | 0.000 | |
| 4 | nyt a t oyvf sqghu | c0eb129061675da412b0deb15871dd06ef0d7cd86eb5f7... | 2013 | 2300000 | other | 2020 | 9 | 8 | 0.513 | |

5 rows × 23 columns

```
# List of columns to drop
columns_to_drop = ["email_hash", "class_flag", "designation_flag", "tier_flag", "company_hash", "is_tech"]

# Check if the columns exist in the DataFrame
existing_columns = [col for col in columns_to_drop if col in df.columns]

# Drop only the existing columns
df = df.drop(columns=existing_columns)

# Display the updated DataFrame
print(df.head())
```

```

orgyear      ctc      job_position ctc_updated_year   yoe  num_jobs \
0    2018    720000  backend engineer           2020    4      9
1    2018   1130000 engineering intern           2021    4      9
2    2017   2400000          qa engineer           2020    5      9
3    2015   1200000  backend architect           2020    7      8
4    2013   2300000             other            2020    9      8

growth  ctc_yoe_ratio  mean_ctc_exp  mean_ctc_position  mean_ctc_company \
0    0.029        15.000  720000.000       980000.000     747619.048
1    0.041        23.542 1130000.000      1130000.000    1223750.000
2    0.200        40.000 1900000.000      1612857.143   2236561.619
3    0.000        14.286 1200000.000      3269120.000   1700962.689
4    0.513        21.296 2300000.000      3603825.308   4401026.176

is_other  is_developer  is_management  is_sales  is_non_coder  ctc_bin
0    False        True        False      False      False      low
1    False        True        False      False      False  medium
2    False        True        False      False      False    high
3    False        True        False      False      False  medium
4    True        False        False      False      True    high

```

In [95]: # Define a dictionary for mapping ctc_bin values to corresponding numerical values

```

ctc_bin_map = {
    "very_high": 5,
    "high": 4,
    "medium": 3,
    "low": 2
}

# Use pandas' map function to map ctc_bin to numerical values, with a default value of 1 for other categories
df['ctc_bin'] = df['ctc_bin'].map(ctc_bin_map).fillna(1).astype(int)

# Display the updated DataFrame
print(df.head())

```

```

orgyear      ctc      job_position ctc_updated_year   yoe  num_jobs \
0    2018    720000  backend engineer           2020    4      9
1    2018   1130000 engineering intern           2021    4      9
2    2017   2400000          qa engineer           2020    5      9
3    2015   1200000  backend architect           2020    7      8
4    2013   2300000             other            2020    9      8

growth  ctc_yoe_ratio  mean_ctc_exp  mean_ctc_position  mean_ctc_company \
0    0.029        15.000  720000.000       980000.000     747619.048
1    0.041        23.542 1130000.000      1130000.000    1223750.000
2    0.200        40.000 1900000.000      1612857.143   2236561.619
3    0.000        14.286 1200000.000      3269120.000   1700962.689
4    0.513        21.296 2300000.000      3603825.308   4401026.176

is_other  is_developer  is_management  is_sales  is_non_coder  ctc_bin
0    False        True        False      False      False      2
1    False        True        False      False      False      3
2    False        True        False      False      False      4
3    False        True        False      False      False      3
4    True        False        False      False      True      4

```

In [96]: # Step 1: Calculate the job position frequency

```

job_frequency_df = df.groupby("job_position").size().reset_index(name="job_pos_frequency")

# Step 2: Merge the job frequency data back into the original dataframe
df = df.merge(job_frequency_df, on="job_position", how="left")

# Step 3: Drop the original 'job_position' column
df = df.drop(columns=["job_position"])

# Step 4: Rename the 'job_pos_frequency' column to 'job_position'
df = df.rename(columns={"job_pos_frequency": "job_position"})

# Display the resulting dataframe
print(df.head())

```

```

orgyear      ctc  ctc_updated_year   yoe  num_jobs  growth  ctc_yoe_ratio \
0    2018    720000           2020    4      9  0.029        15.000
1    2018   1130000           2021    4      9  0.041        23.542
2    2017   2400000           2020    5      9  0.200        40.000
3    2015   1200000           2020    7      8  0.000        14.286
4    2013   2300000           2020    9      8  0.513        21.296

mean_ctc_exp  mean_ctc_position  mean_ctc_company  is_other  is_developer \
0    720000.000       980000.000      747619.048    False      True
1   1130000.000      1130000.000    1223750.000    False      True
2   1900000.000      1612857.143   2236561.619    False      True
3   1200000.000      3269120.000   1700962.689    False      True
4   2300000.000      3603825.308   4401026.176    True     False

is_management  is_sales  is_non_coder  ctc_bin  job_position
0    False    False      False        2      52262
1    False    False      False        3      1958
2    False    False      False        4      5863
3    False    False      False        3      1028
4    False    False      True        4      19147

```

In [97]: df_enc = df.copy()

6.1 - Standardization

```
In [98]: # Import StandardScaler from sklearn for standardization
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler object, which will scale the features
std_scaler = StandardScaler()

# Fit the StandardScaler to the DataFrame 'df' to calculate the mean and standard deviation
std_scaler.fit(df)

# Transform the data using the fitted scaler
std_df = std_scaler.transform(df)

# Convert the resulting numpy array 'std_df' back into a pandas DataFrame
std_df = pd.DataFrame(std_df, columns=df.columns)

# Display the first 5 rows of the standardized DataFrame
std_df.head()
```

Out[98]:

▼ StandardScaler ⓘ ⓘ

StandardScaler()

```
Out[98]:   orgyear    ctc  ctc_updated_year     yoe  num_jobs  growth  ctc_yoe_ratio  mean_ctc_exp  mean_ctc_position  mean_ctc_company  is_other  is_develope
0      0.768 -0.138           0.372 -0.768     14.047 -0.013       -0.089       -0.170       -0.166        -0.220      -0.418      0.53
1      0.768 -0.106           1.101 -0.768     14.047 -0.013       -0.059       -0.131       -0.150        -0.161      -0.418      0.53
2      0.534 -0.009           0.372 -0.534     14.047 -0.009       -0.001       -0.059       -0.098        -0.035      -0.418      0.53
3      0.067 -0.101           0.372 -0.067     12.234 -0.014       -0.091       -0.124       0.081        -0.102      -0.418      0.53
4     -0.401 -0.017           0.372  0.401     12.234  0.000       -0.067       -0.021       0.117        0.233      2.395     -1.86
```

In [99]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153443 entries, 0 to 153442
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   orgyear          153443 non-null   int64  
 1   ctc              153443 non-null   int64  
 2   ctc_updated_year 153443 non-null   int32  
 3   yoe              153443 non-null   int64  
 4   num_jobs          153443 non-null   int64  
 5   growth            153443 non-null   float64 
 6   ctc_yoe_ratio    153443 non-null   float64 
 7   mean_ctc_exp     153443 non-null   float64 
 8   mean_ctc_position 153443 non-null   float64 
 9   mean_ctc_company 153443 non-null   float64 
 10  is_other          153443 non-null   bool   
 11  is_developer      153443 non-null   bool   
 12  is_management     153443 non-null   bool   
 13  is_sales          153443 non-null   bool   
 14  is_non_coder      153443 non-null   bool   
 15  ctc_bin           153443 non-null   int32  
 16  job_position      153443 non-null   int64  
dtypes: bool(5), float64(5), int32(2), int64(5)
memory usage: 13.6 MB
```

In [100]: # Making std_df as df for further processing

```
DF = std_df.copy()
```

6.1.1 - Visualizing the Standardized Data

```
In [101]: # Heatmap for Correlation Matrix of Standardized Data

plt.figure(figsize=(12, 8))

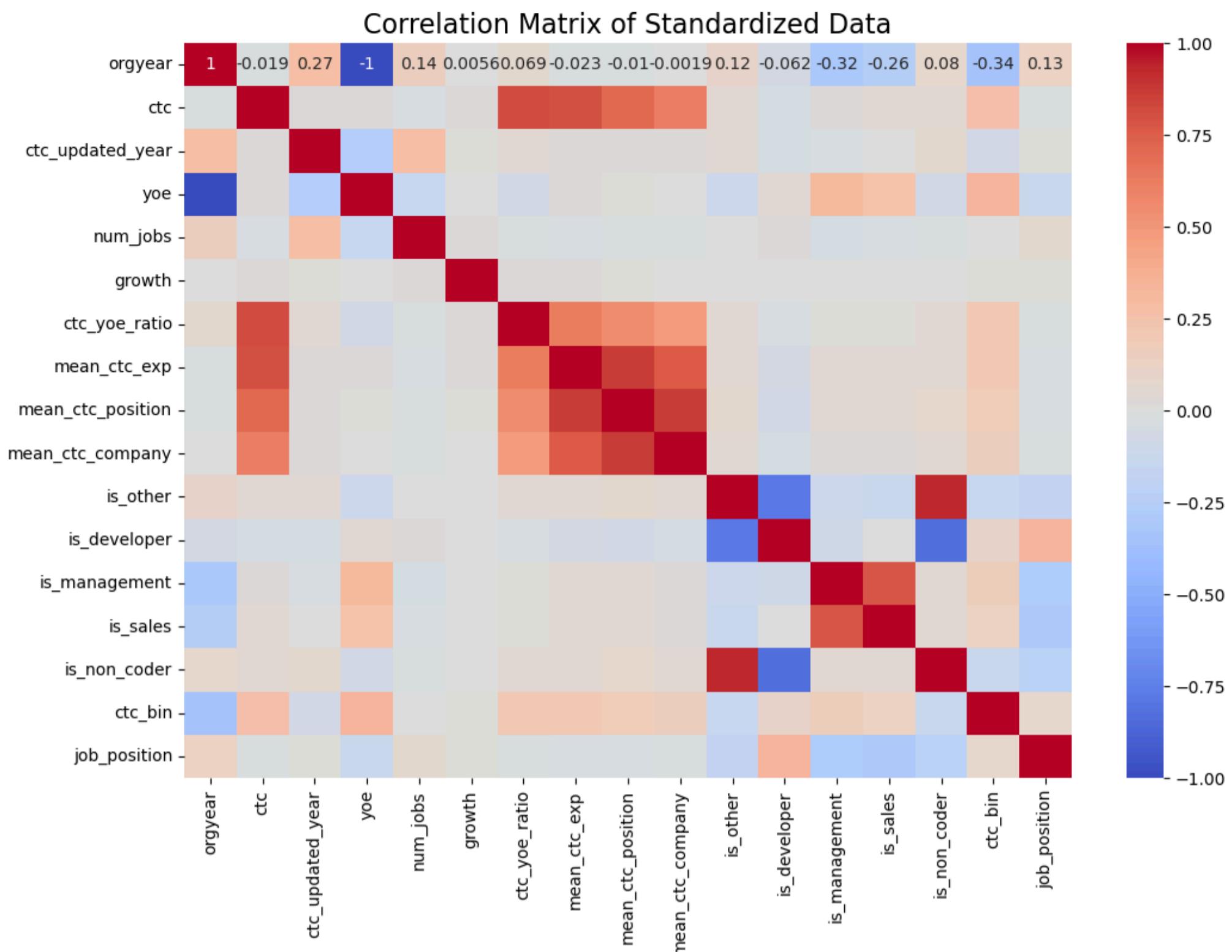
sns.heatmap(DF.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix of Standardized Data", fontsize=16)

plt.show()
```

Out[101]: <Figure size 1200x800 with 0 Axes>

Out[101]: <Axes: >

Out[101]: Text(0.5, 1.0, 'Correlation Matrix of Standardized Data')



7. Unsupervised Learning

In [102]: `DF.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153443 entries, 0 to 153442
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   orgyear     153443 non-null  float64
 1   ctc         153443 non-null  float64
 2   ctc_updated_year  153443 non-null  float64
 3   yoe         153443 non-null  float64
 4   num_jobs    153443 non-null  float64
 5   growth      153443 non-null  float64
 6   ctc_yoe_ratio  153443 non-null  float64
 7   mean_ctc_exp   153443 non-null  float64
 8   mean_ctc_position  153443 non-null  float64
 9   mean_ctc_company  153443 non-null  float64
 10  is_other    153443 non-null  float64
 11  is_developer 153443 non-null  float64
 12  is_management 153443 non-null  float64
 13  is_sales    153443 non-null  float64
 14  is_non_coder 153443 non-null  float64
 15  ctc_bin     153443 non-null  float64
 16  job_position 153443 non-null  float64
dtypes: float64(17)
memory usage: 19.9 MB
```

7.1 - Checking Clustering Tendency

7.1.1 - Hopkins Test for Cluster Tendency Check

The Hopkins statistic is a measure used to evaluate the tendency of a dataset to cluster. It assesses the randomness of data distribution.

The statistic ranges from 0 to 1, with the following interpretations:

- Near 0: The data is uniformly distributed and does not exhibit cluster structure.
- Near 1: The data exhibits strong clustering tendencies.

- Around 0.5: The data distribution is random.

```
In [103]: def hopkins_statistic(df, sample_size=0.1):
    # Ensure all columns in the DataFrame are numeric
    df_numeric = DF.apply(pd.to_numeric, errors='coerce')

    # Number of rows in the dataframe
    n = df_numeric.shape[0]

    # Determine the sample size based on the given proportion
    m = int(sample_size * n)

    # Randomly select indices for sampling
    random_indices = np.random.choice(np.arange(n), size=m, replace=False)
    df_sample = df_numeric.iloc[random_indices]

    # Convert the dataframe to a NumPy array for min/max calculation
    X = df_numeric.to_numpy()

    # Calculate the min and max for each column
    X_min = np.min(X, axis=0)
    X_max = np.max(X, axis=0)

    # Generate random points within the bounds of the data
    X_uniform_random = np.random.uniform(X_min, X_max, (m, X.shape[1]))

    # Fit nearest neighbors model on the data
    nbrs = NearestNeighbors(n_neighbors=1).fit(X)

    # Calculate the distances for the sampled points and the random points
    u_distances, _ = nbrs.kneighbors(df_sample)
    u_distances = u_distances.sum()

    w_distances, _ = nbrs.kneighbors(X_uniform_random)
    w_distances = w_distances.sum()

    # Compute the Hopkins statistic
    hopkins_stat = w_distances / (u_distances + w_distances)

    return hopkins_stat
```

```
In [104]: hopkins_stat = hopkins_statistic(DF, sample_size=0.1)
print("Hopkins Statistic:", hopkins_stat)
```

Hopkins Statistic: 0.999999998724884

The *Hopkins statistic* value is **0.999999998724884**, which is extremely *close to 1*.

This indicates that The Data has **Strong Clustering Tendency & Clustering Methods Are Applicable**

7.2 - K-Means Clustering

```
In [105]: DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153443 entries, 0 to 153442
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   orgyear          153443 non-null   float64
 1   ctc              153443 non-null   float64
 2   ctc_updated_year 153443 non-null   float64
 3   yoe              153443 non-null   float64
 4   num_jobs          153443 non-null   float64
 5   growth            153443 non-null   float64
 6   ctc_yoe_ratio     153443 non-null   float64
 7   mean_ctc_exp      153443 non-null   float64
 8   mean_ctc_position 153443 non-null   float64
 9   mean_ctc_company  153443 non-null   float64
 10  is_other          153443 non-null   float64
 11  is_developer      153443 non-null   float64
 12  is_management     153443 non-null   float64
 13  is_sales          153443 non-null   float64
 14  is_non_coder      153443 non-null   float64
 15  ctc_bin           153443 non-null   float64
 16  job_position      153443 non-null   float64
dtypes: float64(17)
memory usage: 19.9 MB
```

```
In [106]: import os
from sklearn.cluster import KMeans
```

```
# If needed, you can set the number of threads to avoid parallelism issues
os.environ["OMP_NUM_THREADS"] = "1" # Disabling multi-threading

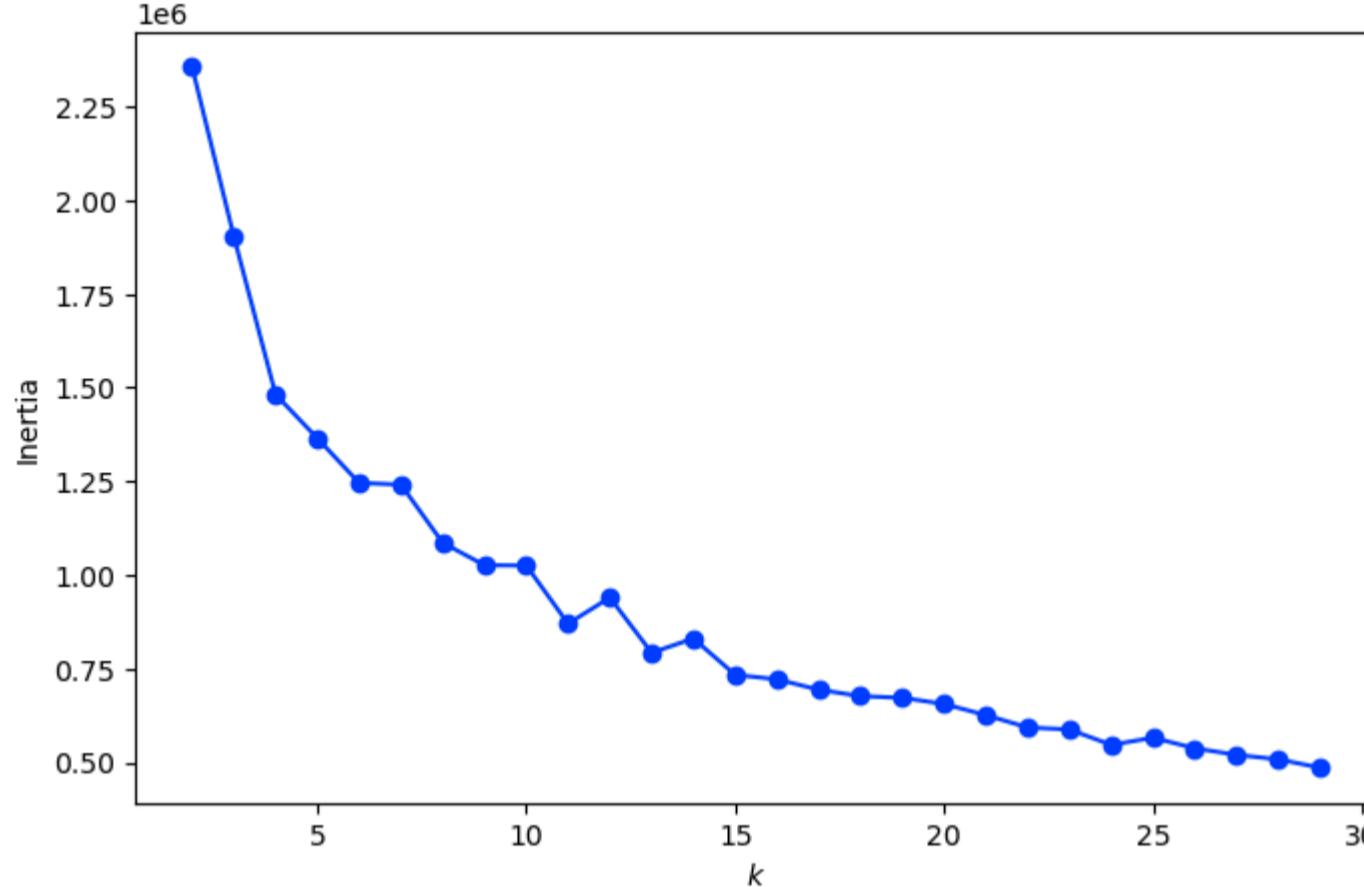
# Assuming df is your DataFrame and is numeric
df_numeric = DF.select_dtypes(include=['float64', 'int64'])

# Fit KMeans models for each value of k and store the models in a list
kmeans_per_k = [KMeans(n_clusters=k, random_state=42, n_init=10).fit(df_numeric) for k in range(2, 30, 1)]
```

```
# Extract the inertia values (within-cluster sum of squared distances) from each model
inertias = [model.inertia_ for model in kmeans_per_k]

# Plot the inertia for each k value to visualize the Elbow method
plt.figure(figsize=(8, 5))
plt.plot(range(2, 30, 1), inertias, "-o")
plt.xlabel("$k$")
plt.ylabel("Inertia")
plt.show()
```

Out[106]: <Figure size 800x500 with 0 Axes>
Out[106]: [`<matplotlib.lines.Line2D at 0x2ed3bb017e0>`]
Out[106]: `Text(0.5, 0, 'k')`
Out[106]: `Text(0, 0.5, 'Inertia')`



We will Use K = 5 for K - Means Clustering due to the presence of Elbow

7.2.1 - Using PCA to reduce the Dimensions

In [107]: # Fit PCA and calculate the cumulative explained variance ratio
`pca = PCA()`
`pca.fit(df_numeric)`
`pca_explained_variance = pca.explained_variance_ratio_.cumsum()`

Output the cumulative explained variance
`pca_explained_variance`

Out[107]: `PCA`
`PCA()`

Out[107]: `array([0.23337373, 0.40859196, 0.56448411, 0.6534014 , 0.71934577, 0.77789144, 0.82638646, 0.8698445 , 0.91103796, 0.94638182, 0.96183536, 0.97527273, 0.98496568, 0.99222017, 0.99719639, 1. , 1.])`

In [108]: # Create a line plot for the cumulative explained variance
`sns.lineplot(x=range(1, len(pca_explained_variance) + 1), y=pca_explained_variance)`

Label the x-axis and y-axis
`plt.xlabel("Number of Components")`
`plt.ylabel("Explained Variance")`

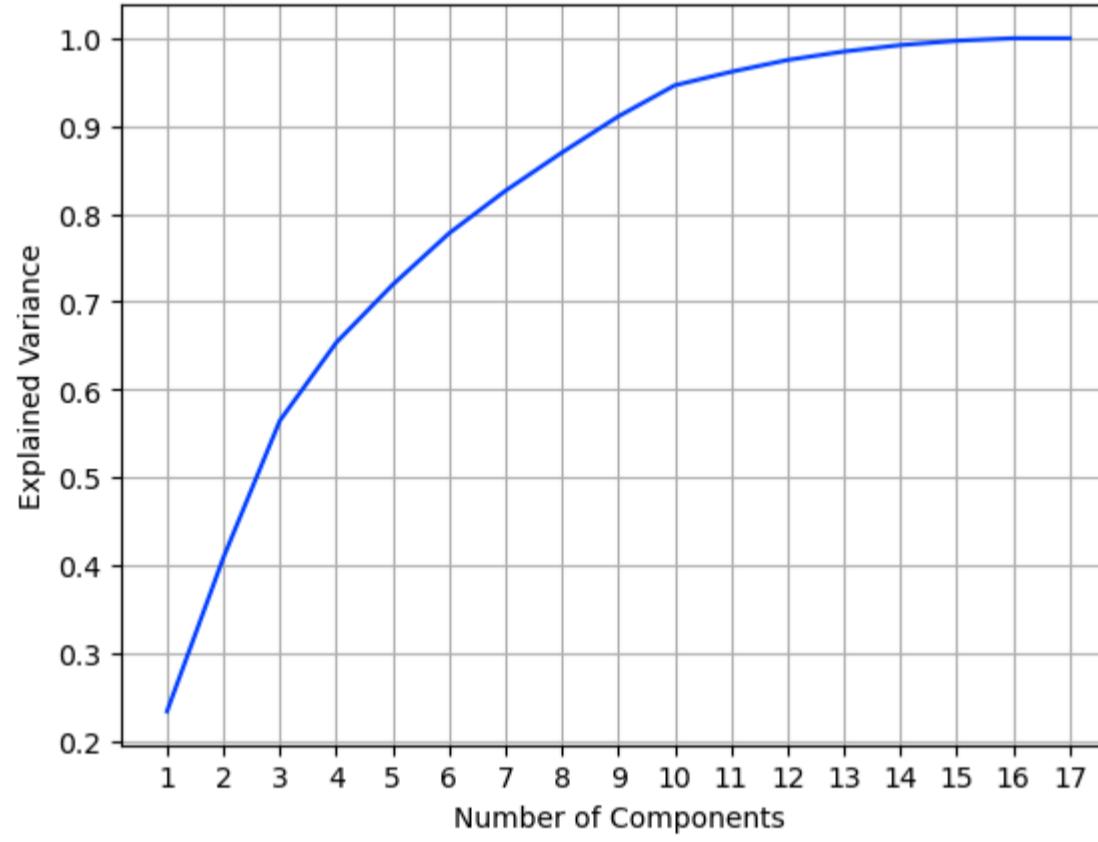
Set the x-ticks to match the number of components
`plt.xticks(range(1, len(pca_explained_variance) + 1))`

Add grid to the plot
`plt.grid(True)`

Display the plot
`plt.show()`

Out[108]: <Axes: >
Out[108]: `Text(0.5, 0, 'Number of Components')`
Out[108]: `Text(0, 0.5, 'Explained Variance')`

```
Out[108]: ([<matplotlib.axis.XTick at 0x2ed3baa6320>,
 <matplotlib.axis.XTick at 0x2ed3baa6740>,
 <matplotlib.axis.XTick at 0x2ed11e3ef50>,
 <matplotlib.axis.XTick at 0x2ed34e6c820>,
 <matplotlib.axis.XTick at 0x2ed34e6e950>,
 <matplotlib.axis.XTick at 0x2ed34e6f130>,
 <matplotlib.axis.XTick at 0x2ed34e6f910>,
 <matplotlib.axis.XTick at 0x2ed34e6f670>,
 <matplotlib.axis.XTick at 0x2ed34e6e2c0>,
 <matplotlib.axis.XTick at 0x2ed34f0c490>,
 <matplotlib.axis.XTick at 0x2ed3bae6350>,
 <matplotlib.axis.XTick at 0x2ed34f0dd50>,
 <matplotlib.axis.XTick at 0x2ed34e3ca00>,
 <matplotlib.axis.XTick at 0x2ed34e3d2d0>,
 <matplotlib.axis.XTick at 0x2ed34e3dab0>,
 <matplotlib.axis.XTick at 0x2ed34e6ca00>,
 <matplotlib.axis.XTick at 0x2ed34e3d090>],
 [Text(1, 0, '1'),
 Text(2, 0, '2'),
 Text(3, 0, '3'),
 Text(4, 0, '4'),
 Text(5, 0, '5'),
 Text(6, 0, '6'),
 Text(7, 0, '7'),
 Text(8, 0, '8'),
 Text(9, 0, '9'),
 Text(10, 0, '10'),
 Text(11, 0, '11'),
 Text(12, 0, '12'),
 Text(13, 0, '13'),
 Text(14, 0, '14'),
 Text(15, 0, '15'),
 Text(16, 0, '16'),
 Text(17, 0, '17')])
```



💡 NOTE:

- We will use 10 Components where the cumulative explained variance exceeds this threshold of 95 %.

```
In [109]: DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153443 entries, 0 to 153442
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   orgyear          153443 non-null   float64
 1   ctc              153443 non-null   float64
 2   ctc_updated_year 153443 non-null   float64
 3   yoe              153443 non-null   float64
 4   num_jobs          153443 non-null   float64
 5   growth            153443 non-null   float64
 6   ctc_yoe_ratio    153443 non-null   float64
 7   mean_ctc_exp     153443 non-null   float64
 8   mean_ctc_position 153443 non-null   float64
 9   mean_ctc_company 153443 non-null   float64
 10  is_other          153443 non-null   float64
 11  is_developer      153443 non-null   float64
 12  is_management     153443 non-null   float64
 13  is_sales          153443 non-null   float64
 14  is_non_coder      153443 non-null   float64
 15  ctc_bin           153443 non-null   float64
 16  job_position      153443 non-null   float64
dtypes: float64(17)
memory usage: 19.9 MB
```

```
In [110]: df_numeric.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153443 entries, 0 to 153442
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   orgyear          153443 non-null   float64
 1   ctc              153443 non-null   float64
 2   ctc_updated_year 153443 non-null   float64
 3   yoe              153443 non-null   float64
 4   num_jobs          153443 non-null   float64
 5   growth            153443 non-null   float64
 6   ctc_yoe_ratio    153443 non-null   float64
 7   mean_ctc_exp     153443 non-null   float64
 8   mean_ctc_position 153443 non-null   float64
 9   mean_ctc_company 153443 non-null   float64
 10  is_other          153443 non-null   float64
 11  is_developer      153443 non-null   float64
 12  is_management     153443 non-null   float64
 13  is_sales          153443 non-null   float64
 14  is_non_coder      153443 non-null   float64
 15  ctc_bin           153443 non-null   float64
 16  job_position      153443 non-null   float64
dtypes: float64(17)
memory usage: 19.9 MB
```

```
In [111]: from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
# Perform PCA with 10 components
X_embedded_pca = PCA(n_components=10).fit_transform(df_numeric)

# Fit KMeans models for each value of k and store the models in a list
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(X_embedded_pca)
                 for k in range(2, 30, 1)]

# Extract the inertia values (within-cluster sum of squared distances) from each model
inertias = [model.inertia_ for model in kmeans_per_k]

# Plot the inertia for each k value to visualize the Elbow method
plt.figure(figsize=(8, 5))
plt.plot(range(2, 30, 1), inertias, "-o")
plt.xlabel("$k$")
plt.ylabel("Inertia")

# Add grid to the plot
plt.grid(True)

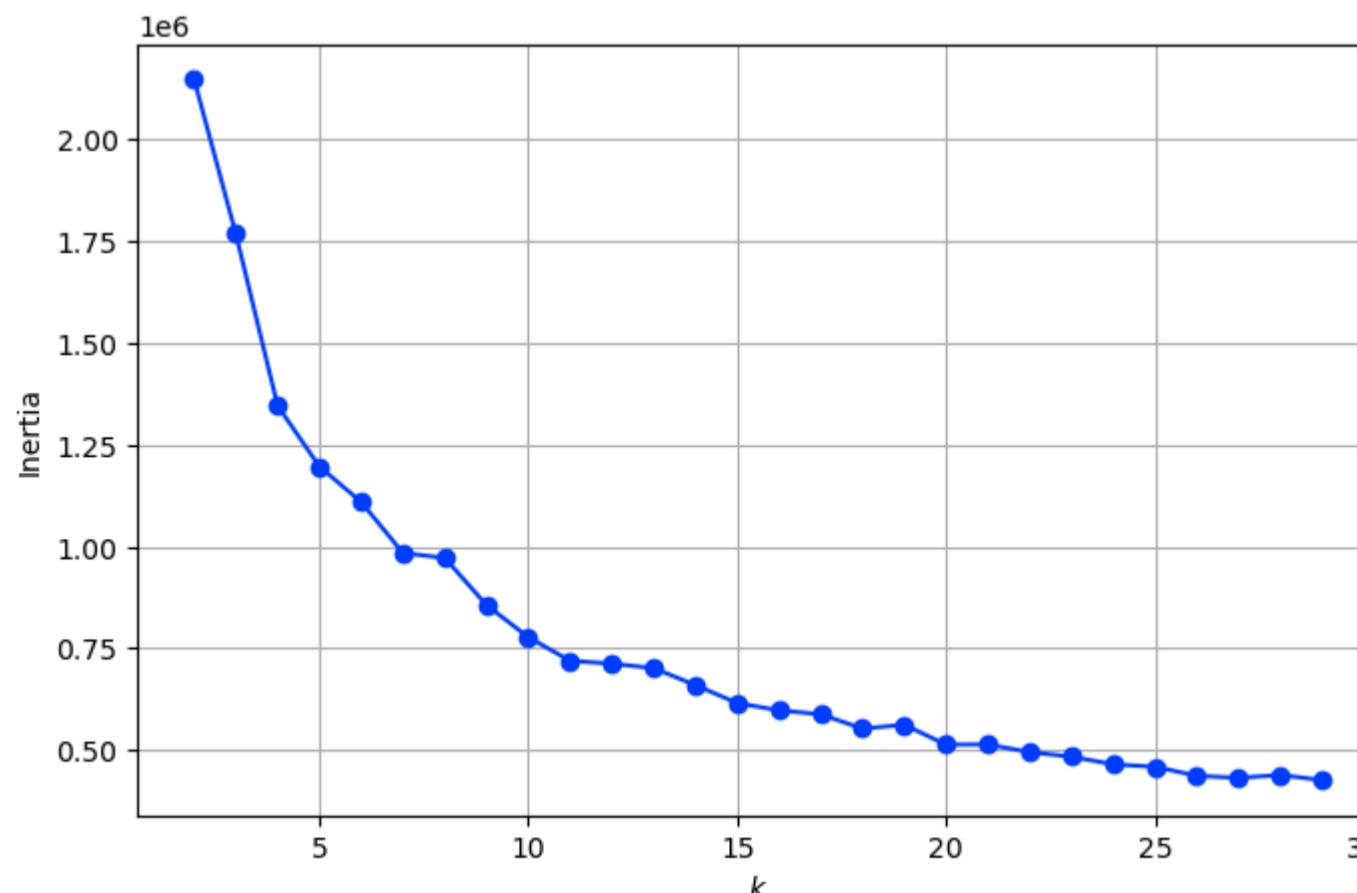
plt.show()
```

```
Out[111]: <Figure size 800x500 with 0 Axes>
```

```
Out[111]: []
```

```
Out[111]: Text(0.5, 0, '$k$')
```

```
Out[111]: Text(0, 0.5, 'Inertia')
```



The Optimal Number of Clusters Seems to be 6, after which the additional components contribute much less.

```
In [112]: from sklearn.cluster import KMeans

# Perform KMeans clustering with 6 clusters
```

```
kmeans = KMeans(n_clusters=6, random_state=42)
y_kmeans = kmeans.fit_predict(X_embedded_pca)

# Add the clustering labels to the DataFrame
DF['y_kmeans'] = y_kmeans
```

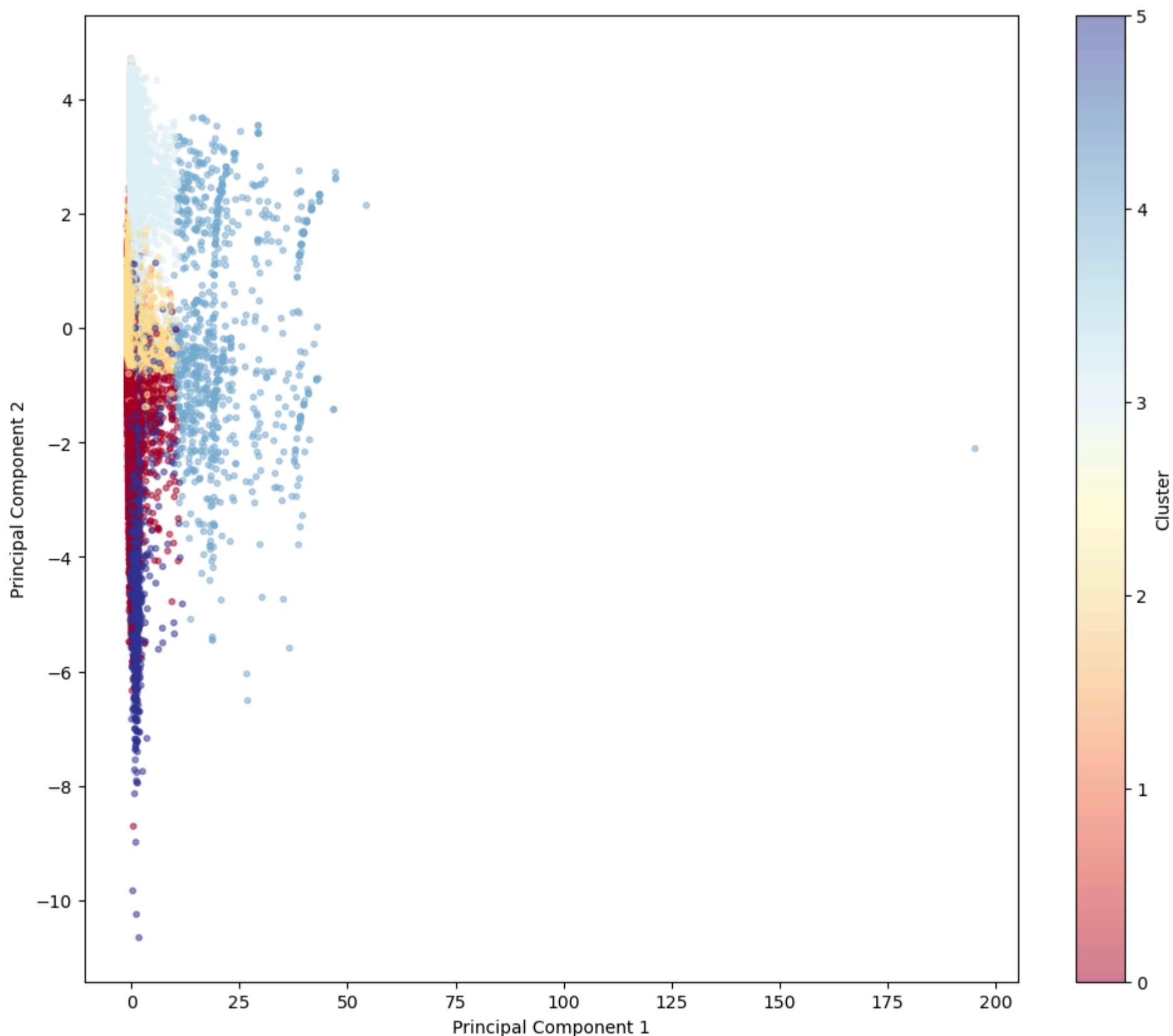
In [113]: `DF.head()`

```
Out[113]:   orgyear    ctc  ctc_updated_year     yoe  num_jobs  growth  ctc_yoe_ratio  mean_ctc_exp  mean_ctc_position  mean_ctc_company  is_other  is_develope
0      0.768 -0.138           0.372 -0.768     14.047 -0.013       -0.089       -0.170       -0.166       -0.220      -0.418      0.53
1      0.768 -0.106           1.101 -0.768     14.047 -0.013       -0.059       -0.131       -0.150      -0.161      -0.418      0.53
2      0.534 -0.009           0.372 -0.534     14.047 -0.009       -0.001       -0.059       -0.098      -0.035      -0.418      0.53
3      0.067 -0.101           0.372 -0.067    12.234 -0.014       -0.091       -0.124       0.081      -0.102      -0.418      0.53
4     -0.401 -0.017           0.372  0.401    12.234  0.000       -0.067       -0.021       0.117      0.233      2.395     -1.86
```

```
In [114]: # Plotting the scatter plot with cluster colors
plt.figure(figsize=(12, 10))
scatter = plt.scatter(X_embedded_pca[:, 0], X_embedded_pca[:, 1], c=DF['y_kmeans'], s=10, cmap='RdYlBu', alpha=0.5)

# Add colorbar and labels
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

```
Out[114]: <Figure size 1200x1000 with 0 Axes>
Out[114]: <matplotlib.colorbar.Colorbar at 0x2ed34eed5d0>
Out[114]: Text(0.5, 0, 'Principal Component 1')
Out[114]: Text(0, 0.5, 'Principal Component 2')
```



7.2.2 - Cluster Analysis

```
In [226]: import pandas as pd
import numpy as np
```

```

import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

def create_polar_plot(data, features, cluster_label):
    """
    Creates a polar plot for a specific cluster, visualizing the means of its features.

    Parameters:
        data (array-like): Feature values for the cluster.
        features (list): List of feature names.
        cluster_label (int): The label or identifier for the cluster.
    """

    # Ensure data is a pandas Series for better handling
    data = pd.Series(data)

    num_vars = len(features)

    # Compute angle of each axis (in radians)
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()

    # The plot is made in a circular (polar) form, so we need to "close the loop"
    data = np.concatenate((data, [data.iloc[0]]))
    angles += angles[:1]

    fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))

    ax.fill(angles, data, color='blue', alpha=0.25)
    ax.plot(angles, data, color='blue', linewidth=2)

    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(features)

    plt.title(f'Cluster {cluster_label}', size=20, color='blue', y=1.1)

    plt.show()

def create_combined_polar_plot(normalized_means, features, cluster_labels):
    """
    Creates a combined polar plot to compare clusters by visualizing the normalized means of their features.

    Parameters:
        normalized_means (DataFrame): DataFrame containing normalized means for each cluster.
        features (list): List of feature names.
        cluster_labels (list): List of cluster labels.
    """

    # Ensure normalized_means is a DataFrame
    if not isinstance(normalized_means, pd.DataFrame):
        normalized_means = pd.DataFrame(normalized_means, columns=features)

    num_vars = len(features)
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
    angles += angles[:1]

    fig, ax = plt.subplots(figsize=(12, 12), subplot_kw=dict(polar=True))

    for i, data in enumerate(normalized_means.values):
        data = np.concatenate((data, [data[0]])) # Close the Loop for the plot
        ax.fill(angles, data, alpha=0.25, label=f'Cluster {cluster_labels[i]}')
        ax.plot(angles, data, linewidth=2)

    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(features)

    plt.title('Cluster Comparison', size=20, y=1.1)
    plt.legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))
    plt.show()

```

```

In [227]: # Group the DataFrame by the 'y_kmeans' column and calculate the mean for each cluster
cluster_means = DF.groupby("y_kmeans").mean().reset_index()

# Drop unnecessary columns ('y_kmeans') from the resulting DataFrame
cluster_means = cluster_means.drop(["y_kmeans", "mean_ctc_exp", "mean_ctc_company", "mean_ctc_position"], axis=1)

# Initialize the MinMaxScaler for normalization
scaler = MinMaxScaler()

# Normalize the cluster means using MinMaxScaler
normalized_means = scaler.fit_transform(cluster_means)

```

```
In [228]: cluster_means
```

| | orgyear | ctc | ctc_updated_year | yoe | num_jobs | growth | ctc_yoe_ratio | is_other | is_developer | is_management | is_sales | is_non_coder | ctc_bin | jol |
|---|---------|--------|------------------|--------|----------|--------|---------------|----------|--------------|---------------|----------|--------------|---------|--------|
| 0 | -0.921 | -0.012 | | -0.782 | 0.921 | -0.345 | -0.012 | -0.073 | -0.415 | 0.416 | -0.274 | -0.285 | -0.445 | 0.533 |
| 1 | 0.461 | -0.084 | | 0.638 | -0.461 | 1.816 | 0.062 | -0.051 | -0.414 | 0.419 | -0.257 | -0.182 | -0.443 | -0.034 |
| 2 | 0.457 | -0.100 | | 0.172 | -0.457 | -0.456 | -0.014 | -0.056 | -0.418 | 0.352 | -0.250 | -0.186 | -0.448 | -0.290 |
| 3 | 0.285 | -0.066 | | 0.139 | -0.285 | -0.019 | -0.008 | -0.045 | 2.370 | -1.861 | -0.273 | -0.286 | 2.230 | -0.386 |
| 4 | 0.203 | 10.922 | | 0.428 | -0.203 | -0.354 | 0.123 | 9.758 | 0.637 | -0.717 | 0.232 | 0.299 | 0.722 | 2.047 |
| 5 | -1.266 | 0.014 | | -0.166 | 1.266 | -0.187 | -0.010 | -0.061 | -0.418 | -0.305 | 3.496 | 3.023 | 0.220 | 0.652 |

```
In [229]: # Convert the columns of the DataFrame to a list
column_list = list(cluster_means.columns)
```

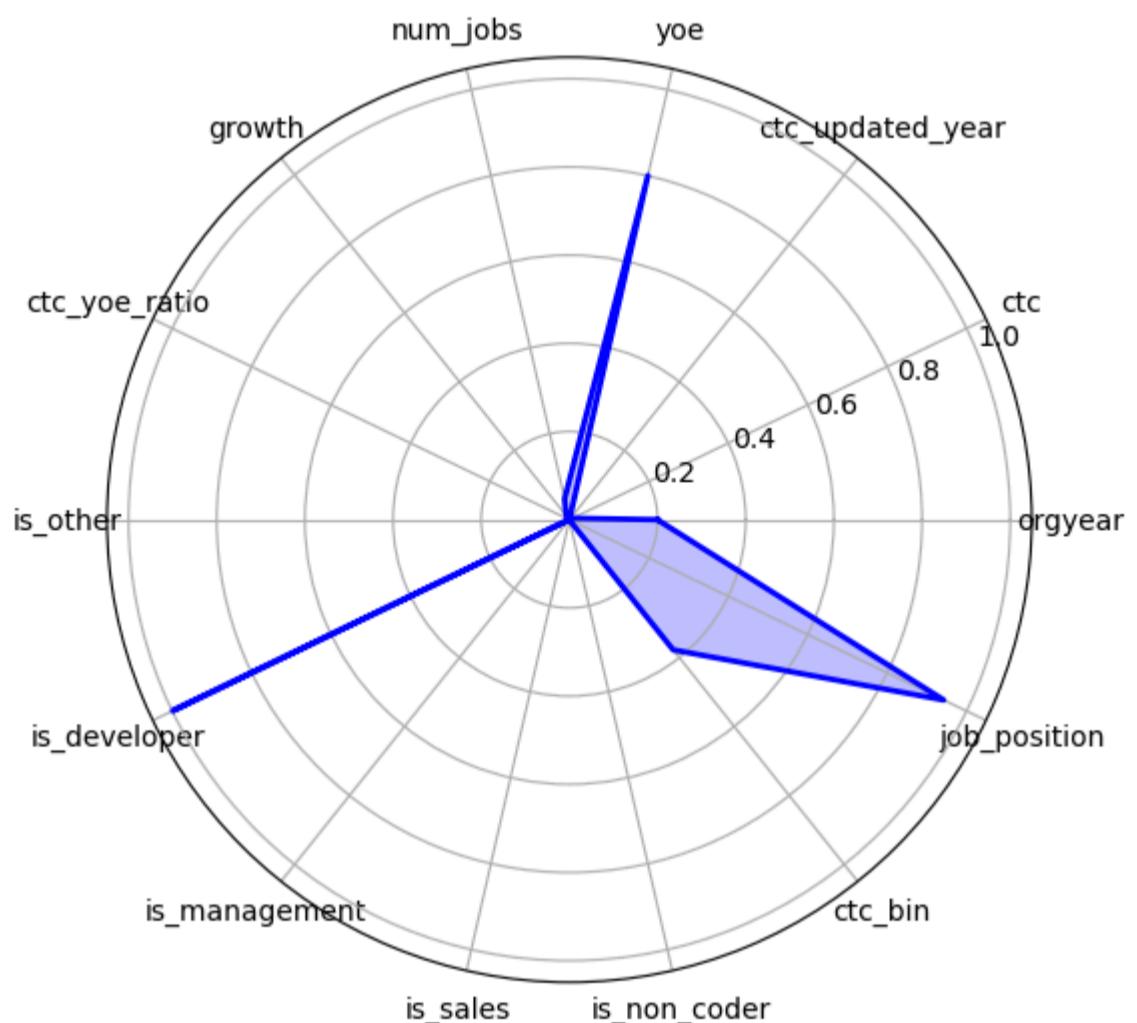
```
In [230]: column_list
```

```
Out[230]: ['orgyear',
 'ctc',
 'ctc_updated_year',
 'yoe',
 'num_jobs',
 'growth',
 'ctc_yoe_ratio',
 'is_other',
 'is_developer',
 'is_management',
 'is_sales',
 'is_non_coder',
 'ctc_bin',
 'job_position']
```

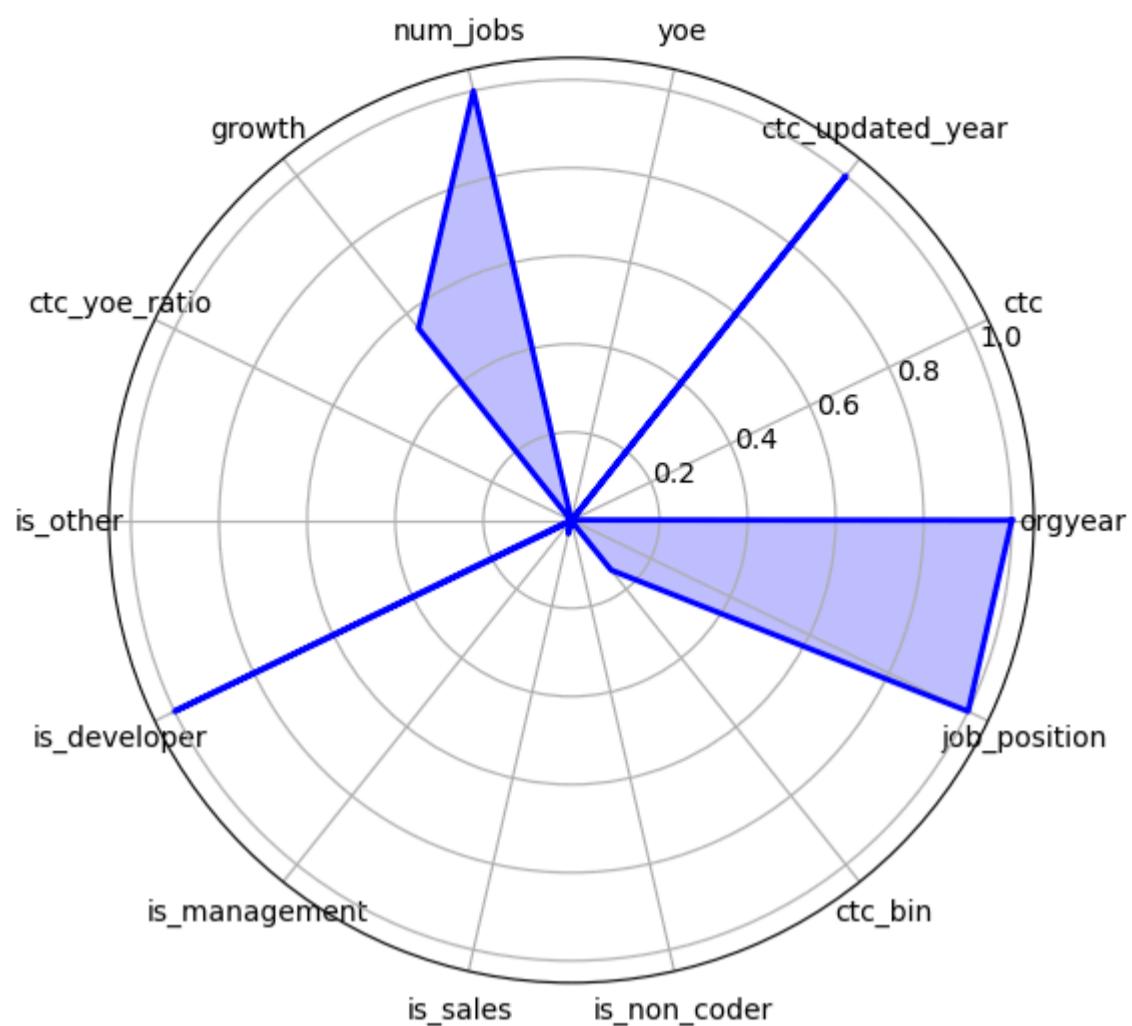
```
In [231]: # Convert the columns of the DataFrame to a list
features = cluster_means.columns.tolist()
```

```
In [232]: for cluster in range(min(cluster_means.shape[0]), normalized_means.shape[0])):
    create_polar_plot(normalized_means[cluster], features, cluster + 1)
```

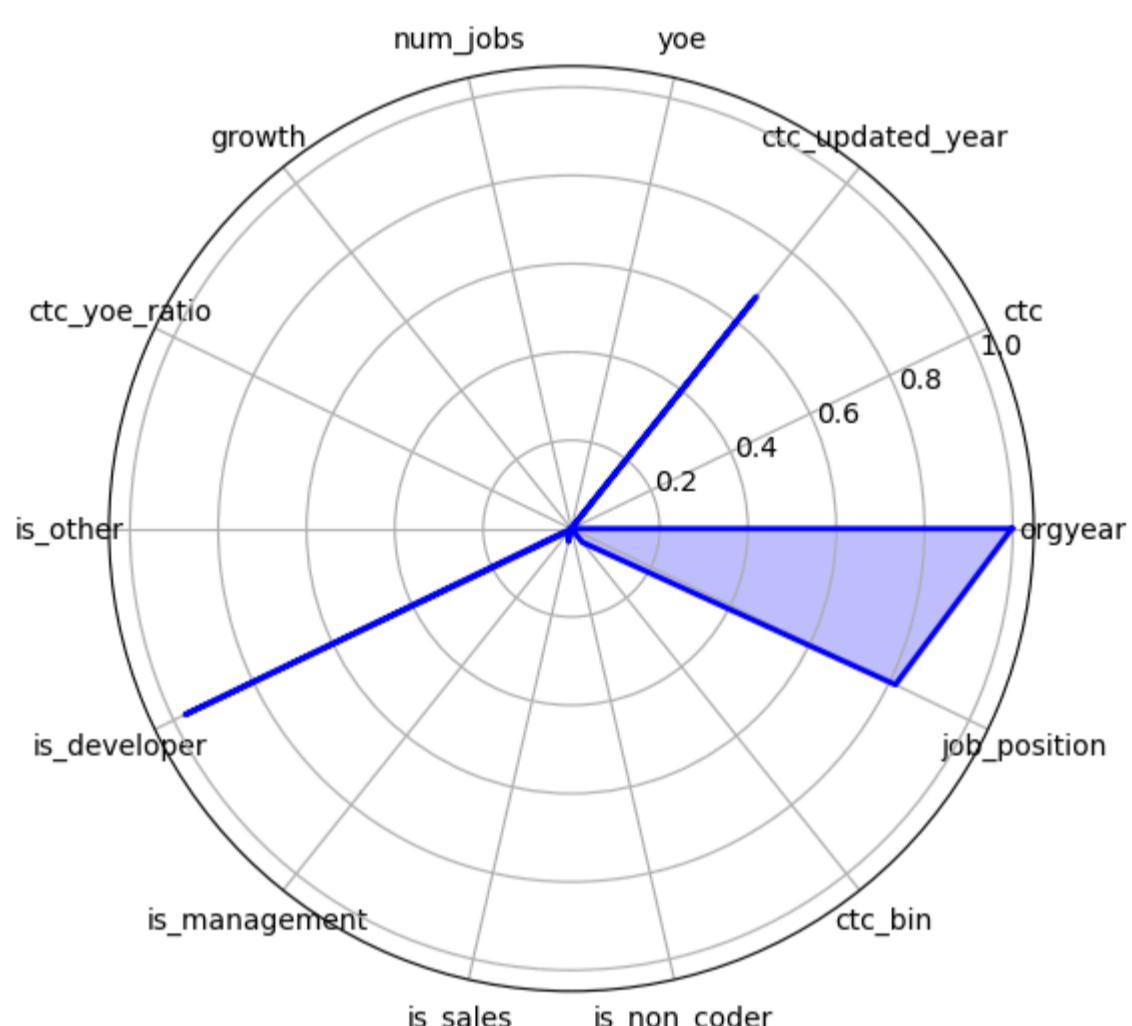
Cluster 1



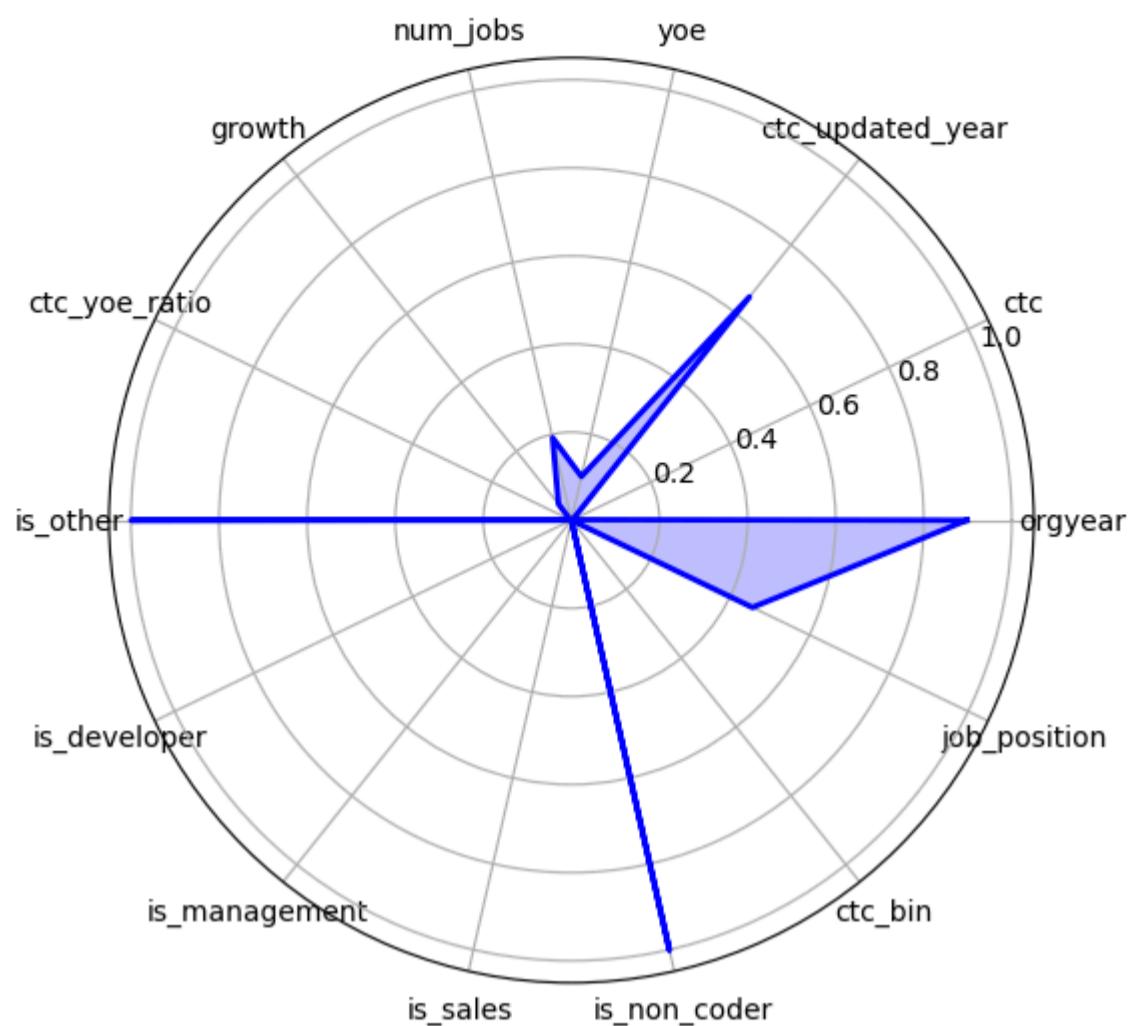
Cluster 2



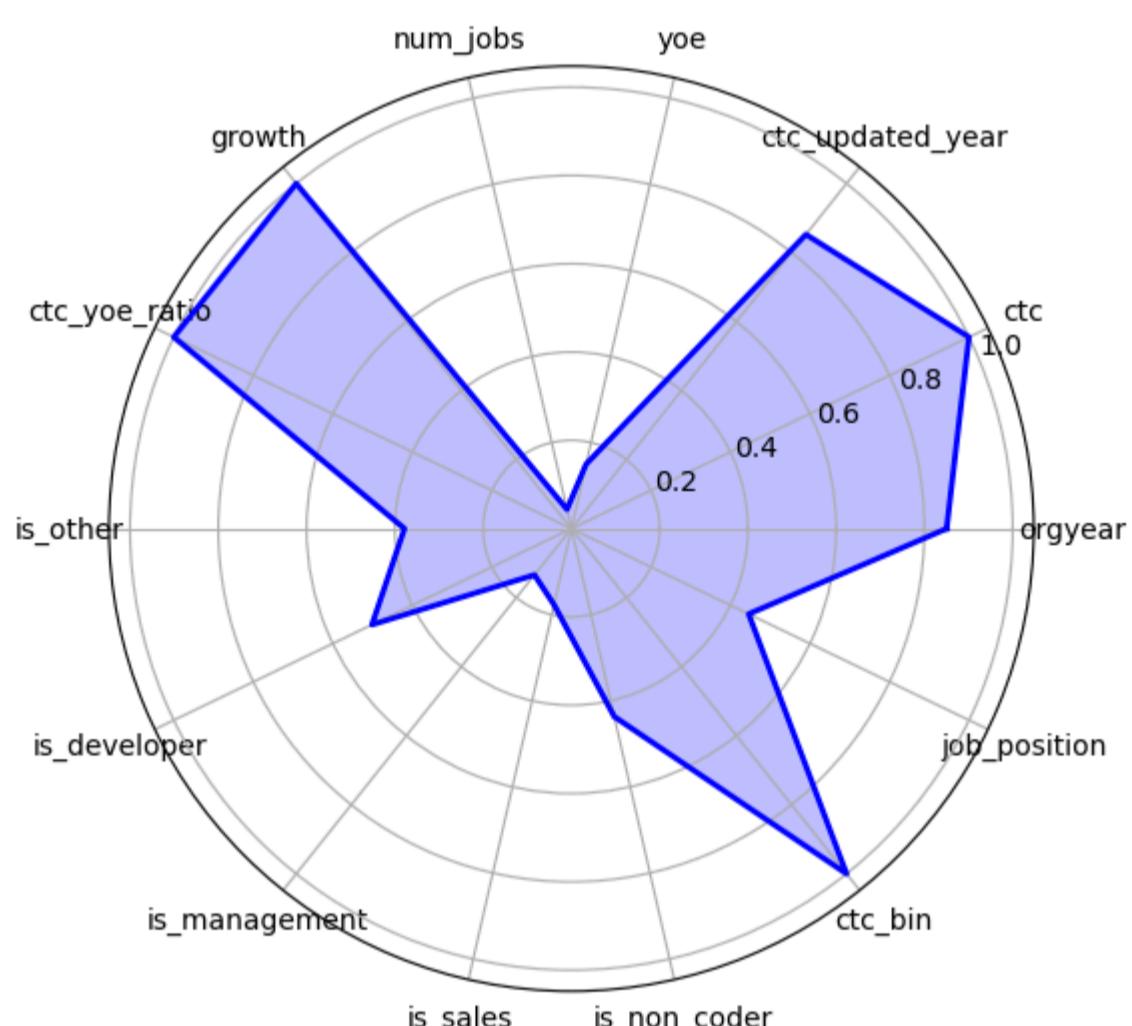
Cluster 3



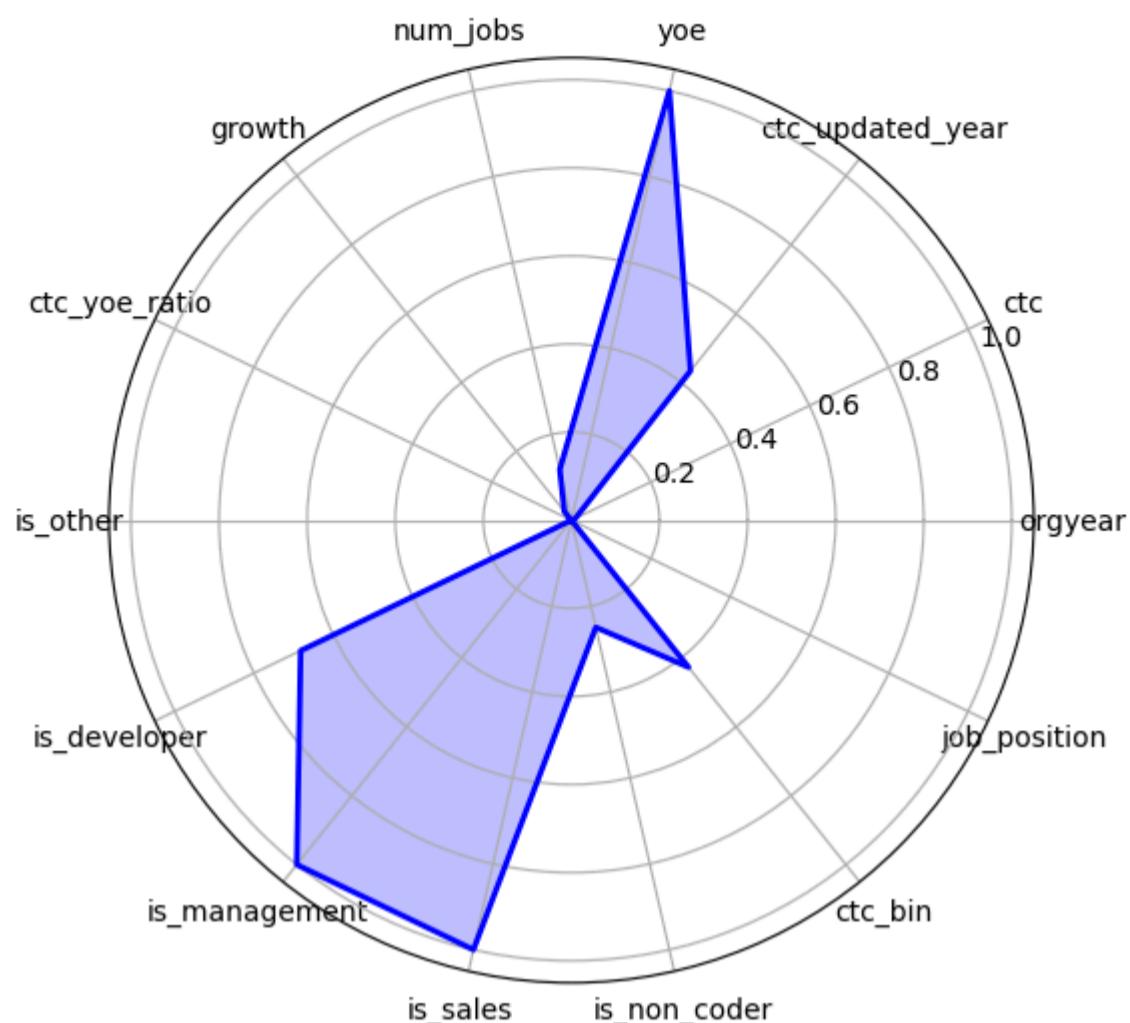
Cluster 4



Cluster 5



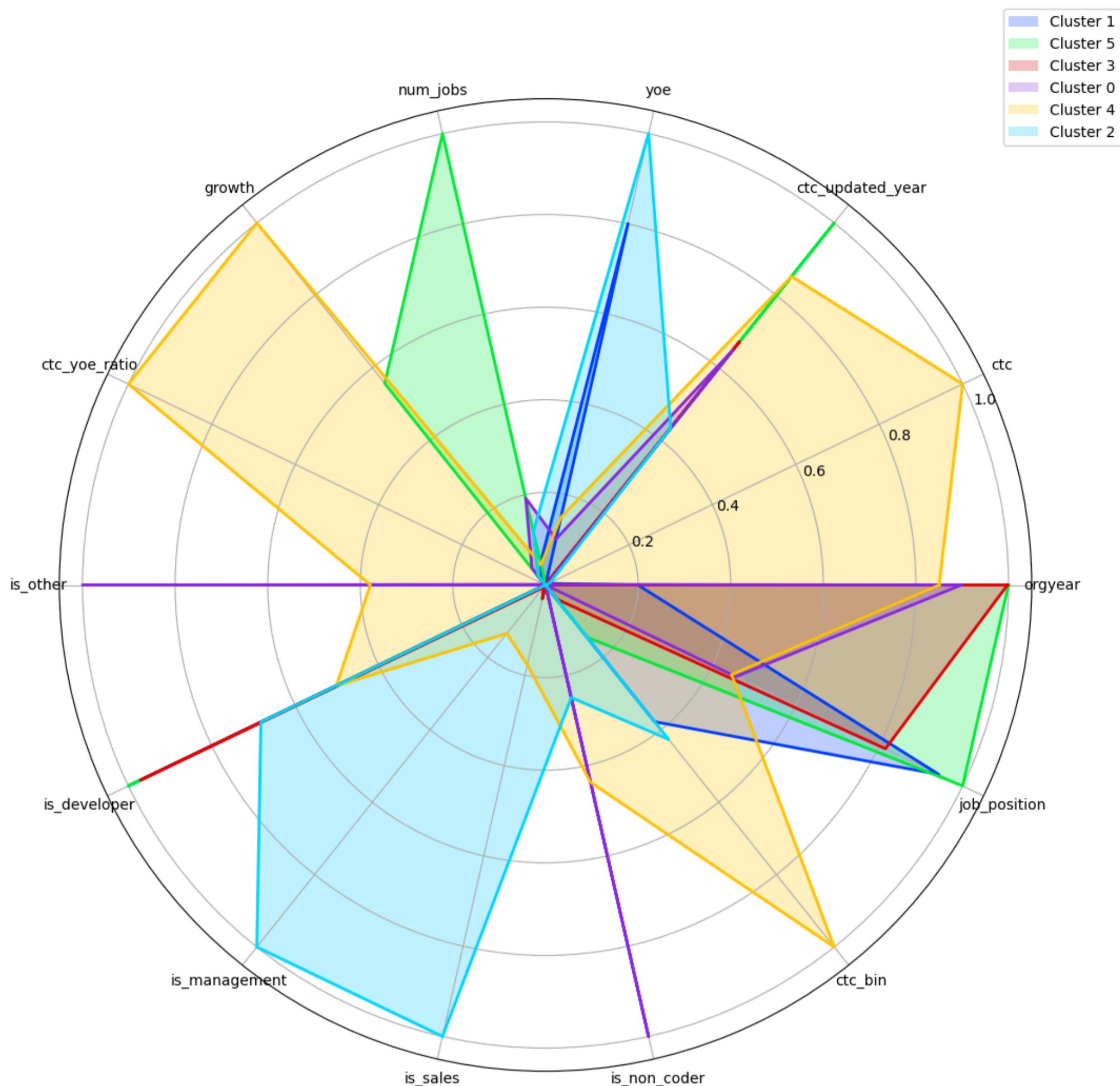
Cluster 6



```
In [233]: # Get the unique cluster labels from the 'y_kmeans' column and convert them to a list
cluster_labels = DF["y_kmeans"].dropna().unique().tolist()

# Create a combined polar plot for all clusters
create_combined_polar_plot(normalized_means, features, cluster_labels)
```

Cluster Comparison



CLUSTER Insights

Cluster - 1 . Underpaid Switchers

(Candidates in this cluster belong to the Developer category, have very good experience, higher job switches, but low CTC.)

Cluster - 2 . Ambitious Movers

(Candidates in this cluster belong to the Developer category, have average growth, higher job switches, and recently got a hike.)

Cluster - 3 . Stagnant Movers

(Candidates in this cluster belong to the Developer category, with no recent hikes and higher job switches.)

Cluster - 4 . Professionally Stuck

(Candidates in this cluster belong to Other and Non-Coder categories, have not gotten any recent hikes, fewer job switches, and appear to be professionally stuck.)

Cluster - 5 . Elite Achievers

(Candidates in this cluster belong to Developer, Non-Coder, and Other categories, have very high growth, very high pay, recent hikes, and fewer job switches.)

Cluster - 6 . Underpaid Veterans

(Candidates in this cluster belong to Management and Sales categories, have very high experience, but are paid less.)

7.3 - Hierarchical Clustering

In [124]: # Drop Mean_ctc_exp, Mean_ctc_company, Mean_ctc_position columns from the df_numeric

```
df_numeric = df_numeric.drop(["mean_ctc_exp", "mean_ctc_company", "mean_ctc_position"], axis=1)
```

```
In [125]: X_embedded_pca = PCA(n_components=10).fit_transform(df_numeric)
```

```
In [126]: # Reduce the sample size to 20% of the original data
sample = pd.DataFrame(X_embedded_pca).sample(frac=0.2, random_state=42) # Adjust frac as needed
sample_hc = df_numeric.loc[sample.index]

# Verify the size of the reduced sample
print(f"Original data size: {len(X_embedded_pca)} rows")
print(f"Reduced sample size: {len(sample)} rows")
```

Original data size: 153443 rows
Reduced sample size: 30689 rows

```
In [127]: sample_hc.shape
```

```
Out[127]: (30689, 14)
```

```
In [128]: sample_hc.head()
```

```
Out[128]:      orgyear    ctc  ctc_updated_year     yoe  num_jobs   growth  ctc_yoe_ratio  is_other  is_developer  is_management  is_sales  is_non_coder  ctc_bin
39677    0.534 -0.124          -0.357 -0.534     -0.456   -0.014       -0.089   -0.418       0.537      -0.275     -0.288     -0.448     -0.451
48859   -0.167 -0.189          -3.274  0.167     -0.456   -0.014       -0.139   -0.418      -1.861      -0.275     -0.288     -0.448     -1.284
52702    0.300 -0.135          -0.357 -0.300     -0.456   -0.014       -0.105   -0.418       0.537      -0.275     -0.288     -0.448     -0.451
69085    1.002 -0.147          1.101 -1.002     -0.456   -0.014       -0.083   -0.418       0.537      -0.275     -0.288     -0.448     -0.451
17088    1.236 -0.167          0.372 -1.236     1.357   -0.014       -0.092   2.395      -1.861      -0.275     -0.288      2.230     -1.284
```

7.3.1 - Building Linkage Matrix

```
In [129]: import time

start_time = time.time()
Z = fastcluster.linkage(sample, method='ward')
end_time = time.time()

print(f"Execution time: {end_time - start_time} seconds")
```

Execution time: 47.29646039009094 seconds

7.3.2 - Plot Dendrogram

```
In [130]: # Plot the dendrogram
fig, ax = plt.subplots(figsize=(20, 12))
dendrogram(
    Z,
    truncate_mode='lastp', # Show the last p merged clusters
    p=100,                # Number of clusters displayed at the bottom
)
plt.title("Dendrogram")
plt.xlabel("Cluster Index")
plt.ylabel("Distance")
plt.show()
```

```
Out[130]: {'icoord': [[5.0, 5.0, 15.0, 15.0],  
[35.0, 35.0, 45.0, 45.0],  
[25.0, 25.0, 40.0, 40.0],  
[10.0, 10.0, 32.5, 32.5],  
[65.0, 65.0, 75.0, 75.0],  
[55.0, 55.0, 70.0, 70.0],  
[95.0, 95.0, 105.0, 105.0],  
[85.0, 85.0, 100.0, 100.0],  
[115.0, 115.0, 125.0, 125.0],  
[135.0, 135.0, 145.0, 145.0],  
[120.0, 120.0, 140.0, 140.0],  
[92.5, 92.5, 130.0, 130.0],  
[62.5, 62.5, 111.25, 111.25],  
[21.25, 21.25, 86.875, 86.875],  
[165.0, 165.0, 175.0, 175.0],  
[155.0, 155.0, 170.0, 170.0],  
[185.0, 185.0, 195.0, 195.0],  
[205.0, 205.0, 215.0, 215.0],  
[225.0, 225.0, 235.0, 235.0],  
[210.0, 210.0, 230.0, 230.0],  
[190.0, 190.0, 220.0, 220.0],  
[162.5, 162.5, 205.0, 205.0],  
[54.0625, 54.0625, 183.75, 183.75],  
[265.0, 265.0, 275.0, 275.0],  
[255.0, 255.0, 270.0, 270.0],  
[245.0, 245.0, 262.5, 262.5],  
[295.0, 295.0, 305.0, 305.0],  
[285.0, 285.0, 300.0, 300.0],  
[325.0, 325.0, 335.0, 335.0],  
[345.0, 345.0, 355.0, 355.0],  
[330.0, 330.0, 350.0, 350.0],  
[315.0, 315.0, 340.0, 340.0],  
[292.5, 292.5, 327.5, 327.5],  
[253.75, 253.75, 310.0, 310.0],  
[365.0, 365.0, 375.0, 375.0],  
[385.0, 385.0, 395.0, 395.0],  
[370.0, 370.0, 390.0, 390.0],  
[415.0, 415.0, 425.0, 425.0],  
[435.0, 435.0, 445.0, 445.0],  
[420.0, 420.0, 440.0, 440.0],  
[405.0, 405.0, 430.0, 430.0],  
[380.0, 380.0, 417.5, 417.5],  
[465.0, 465.0, 475.0, 475.0],  
[455.0, 455.0, 470.0, 470.0],  
[505.0, 505.0, 515.0, 515.0],  
[495.0, 495.0, 510.0, 510.0],  
[485.0, 485.0, 502.5, 502.5],  
[535.0, 535.0, 545.0, 545.0],  
[525.0, 525.0, 540.0, 540.0],  
[565.0, 565.0, 575.0, 575.0],  
[555.0, 555.0, 570.0, 570.0],  
[532.5, 532.5, 562.5, 562.5],  
[493.75, 493.75, 547.5, 547.5],  
[462.5, 462.5, 520.625, 520.625],  
[398.75, 398.75, 491.5625, 491.5625],  
[605.0, 605.0, 615.0, 615.0],  
[595.0, 595.0, 610.0, 610.0],  
[585.0, 585.0, 602.5, 602.5],  
[625.0, 625.0, 635.0, 635.0],  
[593.75, 593.75, 630.0, 630.0],  
[655.0, 655.0, 665.0, 665.0],  
[645.0, 645.0, 660.0, 660.0],  
[675.0, 675.0, 685.0, 685.0],  
[695.0, 695.0, 705.0, 705.0],  
[715.0, 715.0, 725.0, 725.0],  
[700.0, 700.0, 720.0, 720.0],  
[680.0, 680.0, 710.0, 710.0],  
[652.5, 652.5, 695.0, 695.0],  
[745.0, 745.0, 755.0, 755.0],  
[735.0, 735.0, 750.0, 750.0],  
[785.0, 785.0, 795.0, 795.0],  
[775.0, 775.0, 790.0, 790.0],  
[765.0, 765.0, 782.5, 782.5],  
[742.5, 742.5, 773.75, 773.75],  
[673.75, 673.75, 758.125, 758.125],  
[611.875, 611.875, 715.9375, 715.9375],  
[805.0, 805.0, 815.0, 815.0],  
[825.0, 825.0, 835.0, 835.0],  
[810.0, 810.0, 830.0, 830.0],  
[845.0, 845.0, 855.0, 855.0],  
[875.0, 875.0, 885.0, 885.0],  
[865.0, 865.0, 880.0, 880.0],  
[850.0, 850.0, 872.5, 872.5],  
[820.0, 820.0, 861.25, 861.25],  
[895.0, 895.0, 905.0, 905.0],  
[915.0, 915.0, 925.0, 925.0],  
[900.0, 900.0, 920.0, 920.0],  
[935.0, 935.0, 945.0, 945.0],  
[965.0, 965.0, 975.0, 975.0],  
[985.0, 985.0, 995.0, 995.0],  
[970.0, 970.0, 990.0, 990.0],  
[955.0, 955.0, 980.0, 980.0],  
[940.0, 940.0, 967.5, 967.5],
```

[910.0, 910.0, 953.75, 953.75],
[840.625, 840.625, 931.875, 931.875],
[663.90625, 663.90625, 886.25, 886.25],
[445.15625, 445.15625, 775.078125, 775.078125],
[281.875, 281.875, 610.1171875, 610.1171875],
[118.90625, 118.90625, 445.99609375, 445.99609375]],
'dcoord': [[0.0, 25.982713145913223, 25.982713145913223, 0.0],
[0.0, 33.30576524076548, 33.30576524076548, 0.0],
[0.0, 38.770750273322726, 38.770750273322726, 33.30576524076548],
[25.982713145913223,
61.03592656133985,
61.03592656133985,
38.770750273322726],
[0.0, 26.408603882233567, 26.408603882233567, 0.0],
[0.0, 32.46683667432226, 32.46683667432226, 26.408603882233567],
[0.0, 23.651454277506705, 23.651454277506705, 0.0],
[0.0, 24.54819897504214, 24.54819897504214, 23.651454277506705],
[0.0, 27.0389238289269, 27.0389238289269, 0.0],
[0.0, 28.356846883741014, 28.356846883741014, 0.0],
[27.0389238289269,
45.730349556656016,
45.730349556656016,
28.356846883741014],
[24.54819897504214,
52.89297595289976,
52.89297595289976,
45.730349556656016],
[32.46683667432226, 74.97218703936281, 74.97218703936281, 52.89297595289976],
[61.03592656133985, 90.28873786355042, 90.28873786355042, 74.97218703936281],
[0.0, 29.539011164371296, 29.539011164371296, 0.0],
[0.0, 60.918682841270616, 60.918682841270616, 29.539011164371296],
[0.0, 23.25912962405862, 23.25912962405862, 0.0],
[0.0, 22.69125321477098, 22.69125321477098, 0.0],
[0.0, 24.52484480893198, 24.52484480893198, 0.0],
[22.69125321477098, 33.69254695681695, 33.69254695681695, 24.52484480893198],
[23.25912962405862, 67.20513476037856, 67.20513476037856, 33.69254695681695],
[60.918682841270616,
118.62693086200359,
118.62693086200359,
67.20513476037856],
[90.28873786355042,
306.98479547683564,
306.98479547683564,
118.62693086200359],
[0.0, 21.69225960382963, 21.69225960382963, 0.0],
[0.0, 33.28408905224972, 33.28408905224972, 21.69225960382963],
[0.0, 43.910092616107434, 43.910092616107434, 33.28408905224972],
[0.0, 22.587615071447072, 22.587615071447072, 0.0],
[0.0, 34.59132931708419, 34.59132931708419, 22.587615071447072],
[0.0, 21.975737615148912, 21.975737615148912, 0.0],
[0.0, 24.167227064228488, 24.167227064228488, 0.0],
[21.975737615148912,
35.77203050631094,
35.77203050631094,
24.167227064228488],
[0.0, 37.57761643580205, 37.57761643580205, 35.77203050631094],
[34.59132931708419, 73.06985963719568, 73.06985963719568, 37.57761643580205],
[43.910092616107434,
104.32859305637959,
104.32859305637959,
73.06985963719568],
[0.0, 23.618836292533885, 23.618836292533885, 0.0],
[0.0, 27.294130918000782, 27.294130918000782, 0.0],
[23.618836292533885,
40.61676836529461,
40.61676836529461,
27.294130918000782],
[0.0, 23.291733402046756, 23.291733402046756, 0.0],
[0.0, 23.923669350515468, 23.923669350515468, 0.0],
[23.291733402046756,
34.65809039174862,
34.65809039174862,
23.923669350515468],
[0.0, 45.056083856987385, 45.056083856987385, 34.65809039174862],
[40.61676836529461,
82.14549690012821,
82.14549690012821,
45.056083856987385],
[0.0, 34.56823489668652, 34.56823489668652, 0.0],
[0.0, 37.71038379457799, 37.71038379457799, 34.56823489668652],
[0.0, 26.0419989838496, 26.0419989838496, 0.0],
[0.0, 38.04854442839366, 38.04854442839366, 26.0419989838496],
[0.0, 48.21874360123134, 48.21874360123134, 38.04854442839366],
[0.0, 21.830073478028552, 21.830073478028552, 0.0],
[0.0, 23.861745955229885, 23.861745955229885, 21.830073478028552],
[0.0, 23.000650369355228, 23.000650369355228, 0.0],
[0.0, 25.703527328225906, 25.703527328225906, 23.000650369355228],
[23.861745955229885,
53.75146877603288,
53.75146877603288,
25.703527328225906],
[48.21874360123134, 77.33446527659824, 77.33446527659824, 53.75146877603288],
[37.71038379457799, 84.4449660420998, 84.4449660420998, 77.33446527659824],

[82.14549690012821, 165.6094022461787, 165.6094022461787, 84.4449660420998],
[0.0, 25.351868749378717, 25.351868749378717, 0.0],
[0.0, 33.5606337025335, 33.5606337025335, 25.351868749378717],
[0.0, 37.224163749363456, 37.224163749363456, 33.5606337025335],
[0.0, 39.47127703981233, 39.47127703981233, 0.0],
[37.224163749363456,
84.41251848568047,
84.41251848568047,
39.47127703981233],
[0.0, 22.04828600509864, 22.04828600509864, 0.0],
[0.0, 37.05017329691842, 37.05017329691842, 22.04828600509864],
[0.0, 21.876324140057676, 21.876324140057676, 0.0],
[0.0, 24.174947435751132, 24.174947435751132, 0.0],
[0.0, 26.228237645290314, 26.228237645290314, 0.0],
[24.174947435751132,
42.05656634685039,
42.05656634685039,
26.228237645290314],
[21.876324140057676,
52.57401648020495,
52.57401648020495,
42.05656634685039],
[37.05017329691842, 64.48358888540095, 64.48358888540095, 52.57401648020495],
[0.0, 23.68969786724258, 23.68969786724258, 0.0],
[0.0, 31.01191394564313, 31.01191394564313, 23.68969786724258],
[0.0, 22.536407024837278, 22.536407024837278, 0.0],
[0.0, 31.799661834732934, 31.799661834732934, 22.536407024837278],
[0.0, 38.586149042678, 38.586149042678, 31.799661834732934],
[31.01191394564313, 100.07791245219484, 100.07791245219484, 38.586149042678],
[64.48358888540095,
106.57252198079262,
106.57252198079262,
100.07791245219484],
[84.41251848568047,
121.27388118703443,
121.27388118703443,
106.57252198079262],
[0.0, 23.617652345678795, 23.617652345678795, 0.0],
[0.0, 32.81492556622314, 32.81492556622314, 0.0],
[23.617652345678795,
42.10866380327482,
42.10866380327482,
32.81492556622314],
[0.0, 21.971325202194596, 21.971325202194596, 0.0],
[0.0, 23.27325813731938, 23.27325813731938, 0.0],
[0.0, 26.120093181656586, 26.120093181656586, 23.27325813731938],
[21.971325202194596,
50.20930864661433,
50.20930864661433,
26.120093181656586],
[42.10866380327482, 73.0533913619095, 73.0533913619095, 50.20930864661433],
[0.0, 22.845428107715946, 22.845428107715946, 0.0],
[0.0, 28.72581464865713, 28.72581464865713, 0.0],
[22.845428107715946,
44.265542099815924,
44.265542099815924,
28.72581464865713],
[0.0, 26.324076010757036, 26.324076010757036, 0.0],
[0.0, 22.96716991529192, 22.96716991529192, 0.0],
[0.0, 27.003473663811107, 27.003473663811107, 0.0],
[22.96716991529192,
30.483885496706975,
30.483885496706975,
27.003473663811107],
[0.0, 36.35934712716218, 36.35934712716218, 30.483885496706975],
[26.324076010757036,
55.56694304880003,
55.56694304880003,
36.35934712716218],
[44.265542099815924,
90.98590999585197,
90.98590999585197,
55.56694304880003],
[73.0533913619095,
146.66853127135118,
146.66853127135118,
90.98590999585197],
[121.27388118703443,
193.98454297833652,
193.98454297833652,
146.66853127135118],
[165.6094022461787,
215.48635554628922,
215.48635554628922,
193.98454297833652],
[104.32859305637959,
339.68400537039236,
339.68400537039236,
215.48635554628922],
[306.98479547683564,
394.0821420473858,
394.0821420473858,
339.68400537039236]],

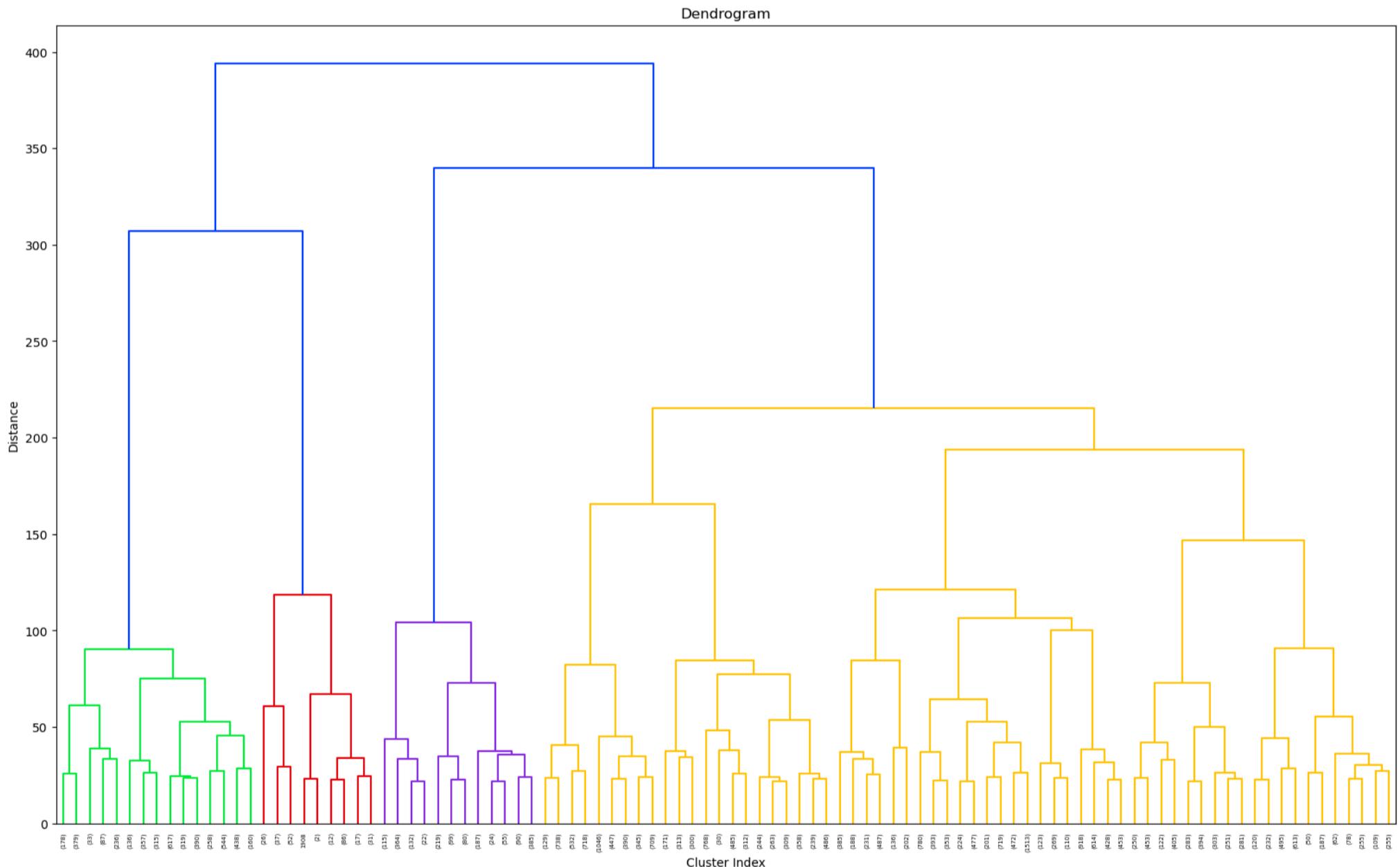
'ivl': ['(178)',
'(379)',
'(33)',
'(87)',
'(236)',
'(136)',
'(357)',
'(315)',
'(617)',
'(319)',
'(390)',
'(258)',
'(544)',
'(438)',
'(160)',
'(26)',
'(37)',
'(52)',
'1908',
'(2)',
'(12)',
'(86)',
'(17)',
'(31)',
'(115)',
'(364)',
'(132)',
'(22)',
'(219)',
'(99)',
'(80)',
'(187)',
'(24)',
'(55)',
'(90)',
'(385)',
'(129)',
'(738)',
'(532)',
'(718)',
'(1046)',
'(447)',
'(390)',
'(345)',
'(709)',
'(171)',
'(313)',
'(300)',
'(768)',
'(30)',
'(485)',
'(312)',
'(244)',
'(263)',
'(309)',
'(358)',
'(239)',
'(486)',
'(385)',
'(188)',
'(231)',
'(487)',
'(136)',
'(202)',
'(780)',
'(393)',
'(353)',
'(224)',
'(477)',
'(201)',
'(719)',
'(472)',
'(1513)',
'(123)',
'(269)',
'(110)',
'(918)',
'(614)',
'(428)',
'(453)',
'(250)',
'(453)',
'(122)',
'(405)',
'(283)',
'(394)',
'(303)',
'(251)',
'(281)',
'(120)',
'(232)',
'(495)',
'(613)',

'(50)',
'(187)',
'(62)',
'(78)',
'(255)',
'(109)',
'(255)'],
'leaves': [61162,
61273,
61255,
61240,
61244,
61208,
61169,
61242,
61210,
61027,
61125,
61203,
61222,
61187,
61252,
61271,
61206,
61253,
1908,
61181,
61171,
61249,
60918,
61110,
61268,
61272,
60949,
61196,
61277,
61096,
61202,
61269,
61081,
61121,
61111,
61233,
61091,
61234,
61180,
61236,
61266,
61159,
61217,
61126,
61237,
61261,
61262,
61267,
61265,
61260,
61229,
61270,
61156,
61188,
61198,
61192,
61136,
61186,
61256,
61241,
61185,
61245,
61250,
61276,
61247,
61161,
61224,
61134,
61189,
61037,
61248,
61123,
61274,
61211,
61190,
61227,
61212,
61215,
61141,
61193,
61100,
61275,
61231,
61263,
61108,
61258,


```
Out[130]: Text(0.5, 1.0, 'Dendrogram')
```

```
Out[130]: Text(0.5, 0, 'Cluster Index')
```

```
Out[130]: Text(0, 0.5, 'Distance')
```



NOTE:

- We will cut the Dendrogram at 250 which will give us 8 Clusters

```
In [131]: sample_hc["cluster"] = fcluster(Z, t=8, criterion="maxclust")
```

7.3.3 - Plotting Hierarchical Clusters

```
In [132]: plt.figure(figsize=(12, 10))
```

```
# Create a scatter plot with the first two principal components
# The points are colored based on the cluster labels from the hierarchical clustering results
scatter = plt.scatter(
    sample.iloc[:, 0], # X-axis: Principal Component 1
    sample.iloc[:, 1], # Y-axis: Principal Component 2
    c=sample_hc["cluster"],
    s=10,
    cmap='viridis',
    alpha=0.5
)
```

```
plt.colorbar(scatter, label='Cluster')
```

```
plt.xlabel("Principal Component 1")
```

```
plt.ylabel("Principal Component 2")
```

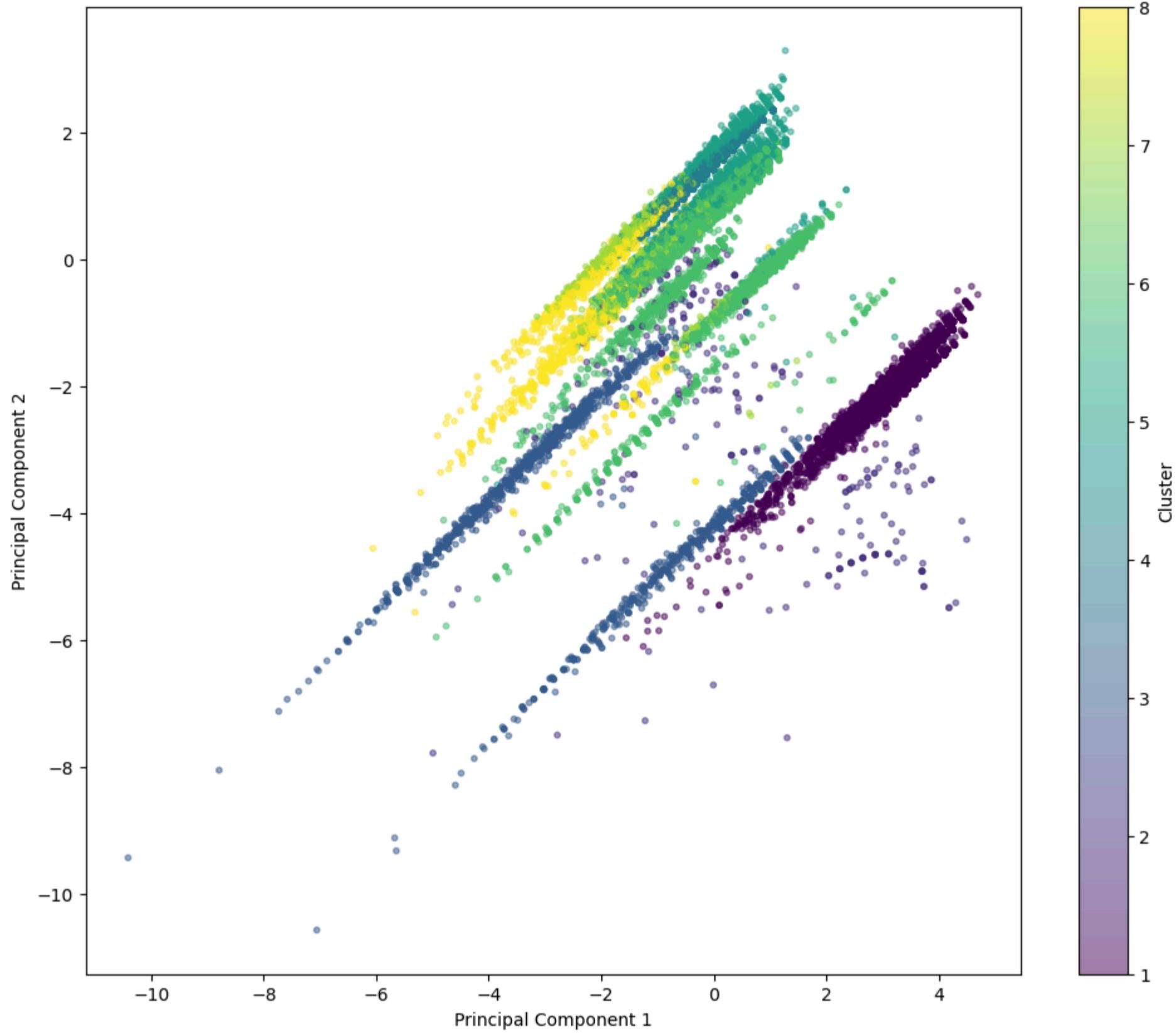
```
plt.show()
```

```
Out[132]: <Figure size 1200x1000 with 0 Axes>
```

```
Out[132]: <matplotlib.colorbar.Colorbar at 0x2ed3e8c2410>
```

```
Out[132]: Text(0.5, 0, 'Principal Component 1')
```

```
Out[132]: Text(0, 0.5, 'Principal Component 2')
```



7.3.4 - Cluster Analysis

```
In [221]: # Group the data by 'cluster' and calculate the mean for each cluster
cluster_means = sample_hc.groupby("cluster").mean()
```

```
# Initialize the MinMaxScaler for normalization
scaler = MinMaxScaler()
```

```
# Apply the MinMaxScaler to normalize the cluster means
normalized_means = scaler.fit_transform(cluster_means)
```

```
# Extract the feature names from the columns of the cluster means DataFrame
features = cluster_means.columns.to_list()
```

```
In [222]: cluster_means
```

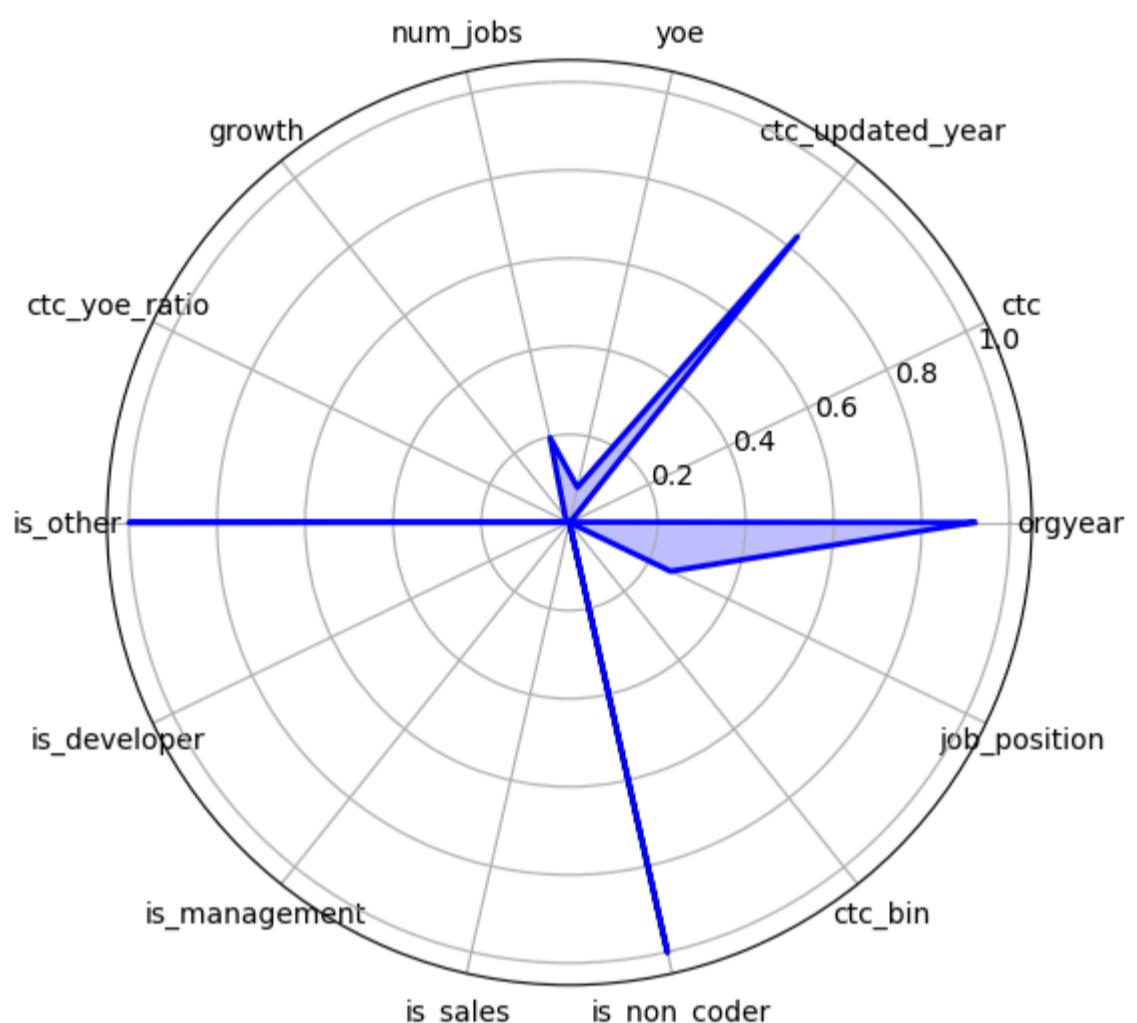
Out[222]:

| cluster | orgyear | ctc | ctc_updated_year | yoe | num_jobs | growth | ctc_yoe_ratio | is_other | is_developer | is_management | is_sales | is_non_coder | ctc_bin |
|---------|---------|--------|------------------|--------|----------|--------|---------------|----------|--------------|---------------|----------|--------------|---------|
| 1 | 0.310 | -0.084 | | 0.160 | -0.310 | -0.013 | -0.009 | -0.066 | 2.395 | -1.861 | -0.275 | -0.288 | 2.230 |
| 2 | 0.233 | 9.884 | | 0.369 | -0.233 | -0.305 | 0.316 | 8.616 | 0.467 | -0.489 | 0.081 | 0.082 | 0.465 |
| 3 | -1.341 | 0.002 | | -0.184 | 1.341 | -0.134 | -0.013 | -0.075 | -0.418 | -0.165 | 3.634 | 3.473 | 0.331 |
| 4 | 0.477 | -0.090 | | 0.298 | -0.477 | -0.456 | -0.014 | -0.055 | -0.418 | 0.537 | -0.275 | -0.288 | -0.448 |
| 5 | 0.439 | -0.088 | | 0.622 | -0.439 | 1.790 | 0.032 | -0.058 | -0.417 | 0.515 | -0.268 | -0.287 | -0.447 |
| 6 | 0.117 | -0.106 | | 0.102 | -0.117 | -0.386 | -0.014 | -0.081 | -0.418 | 0.155 | -0.137 | -0.093 | -0.438 |
| 7 | -0.286 | -0.094 | | -2.100 | 0.286 | -0.366 | -0.014 | -0.096 | -0.418 | 0.490 | -0.275 | -0.288 | -0.445 |
| 8 | -1.572 | -0.025 | | 0.001 | 1.572 | -0.218 | -0.014 | -0.092 | -0.418 | 0.421 | -0.275 | -0.288 | -0.446 |

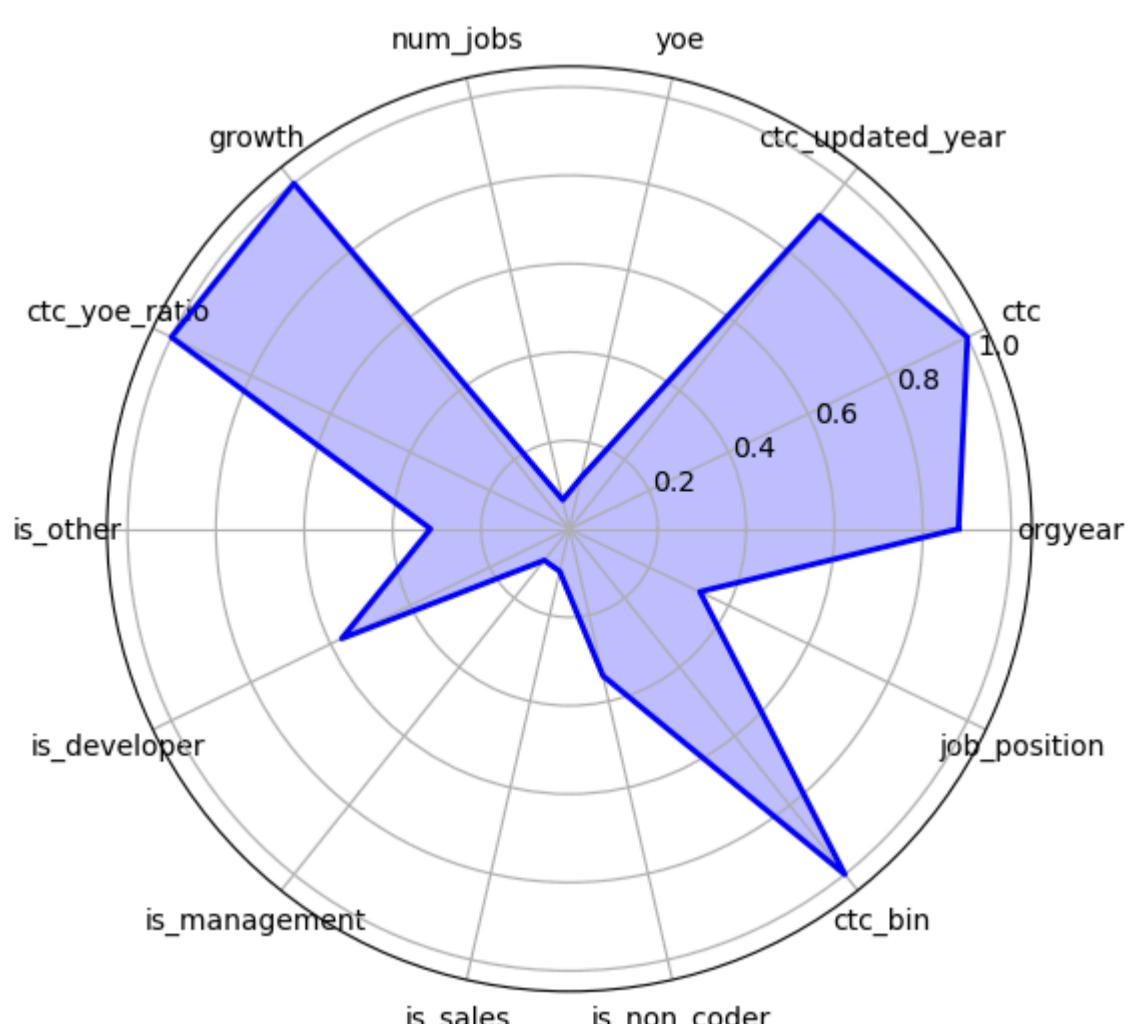
In [223]:

```
for cluster_index in range(min(cluster_means.shape[0], normalized_means.shape[0])):
    create_polar_plot(normalized_means[cluster_index], features, cluster_index + 1)
```

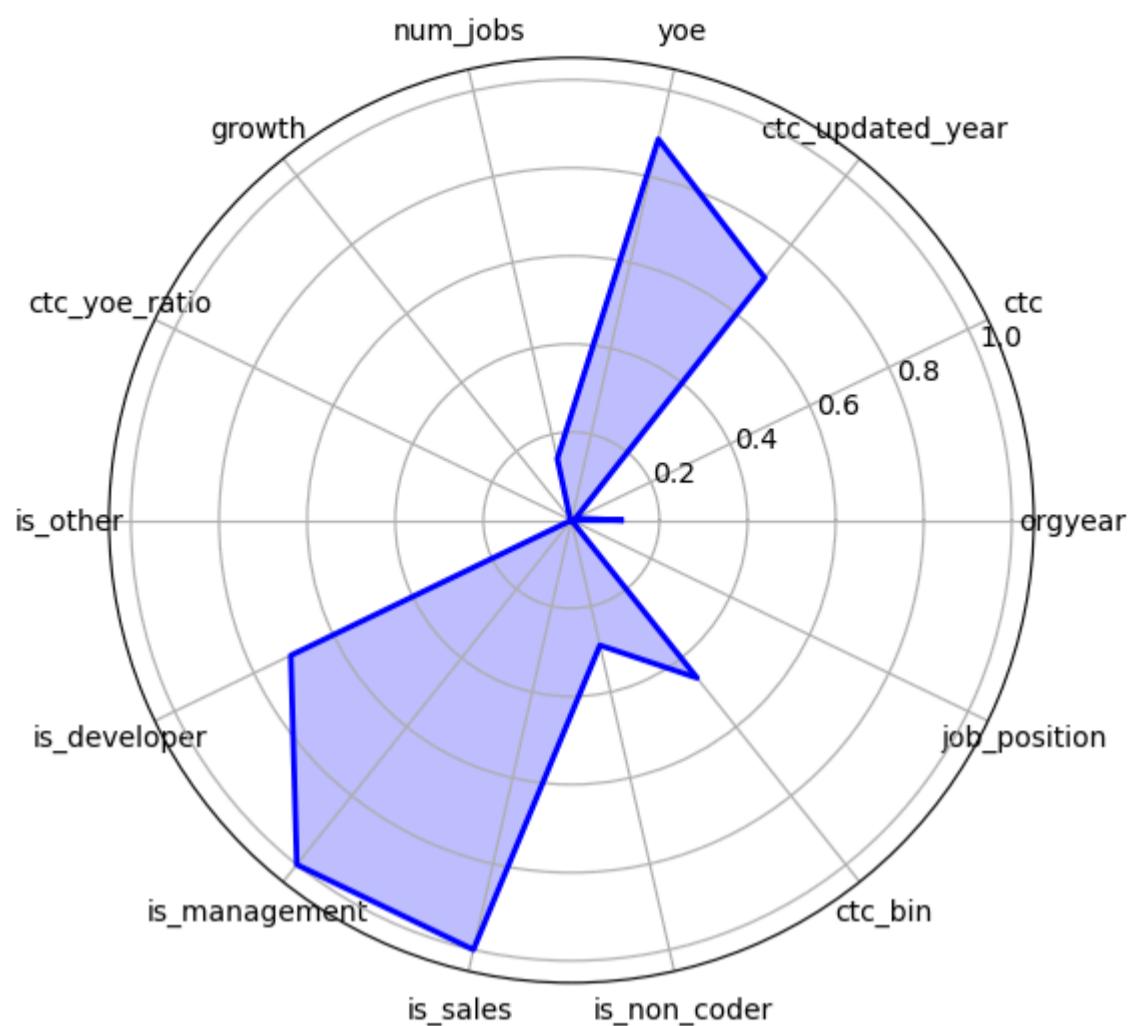
Cluster 1



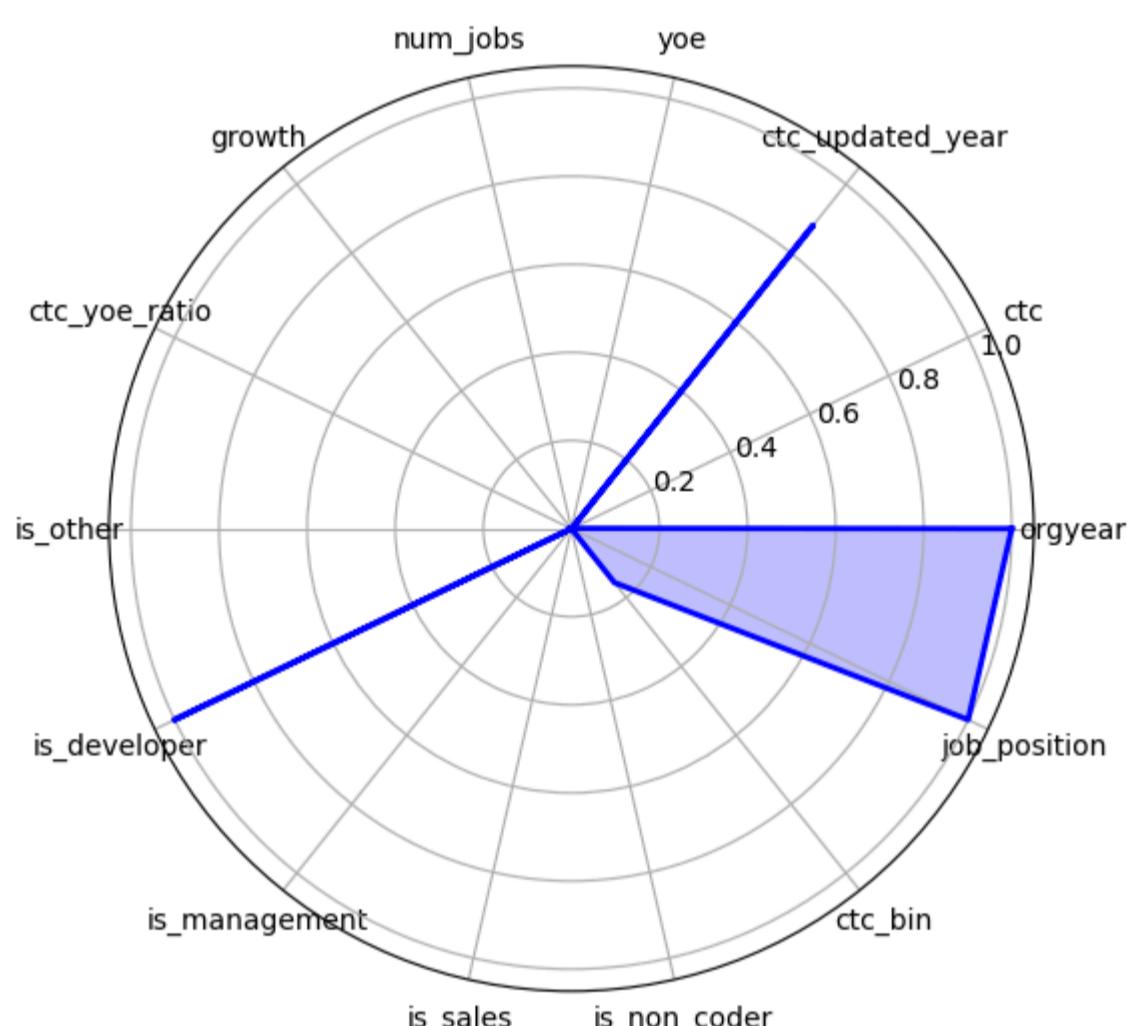
Cluster 2



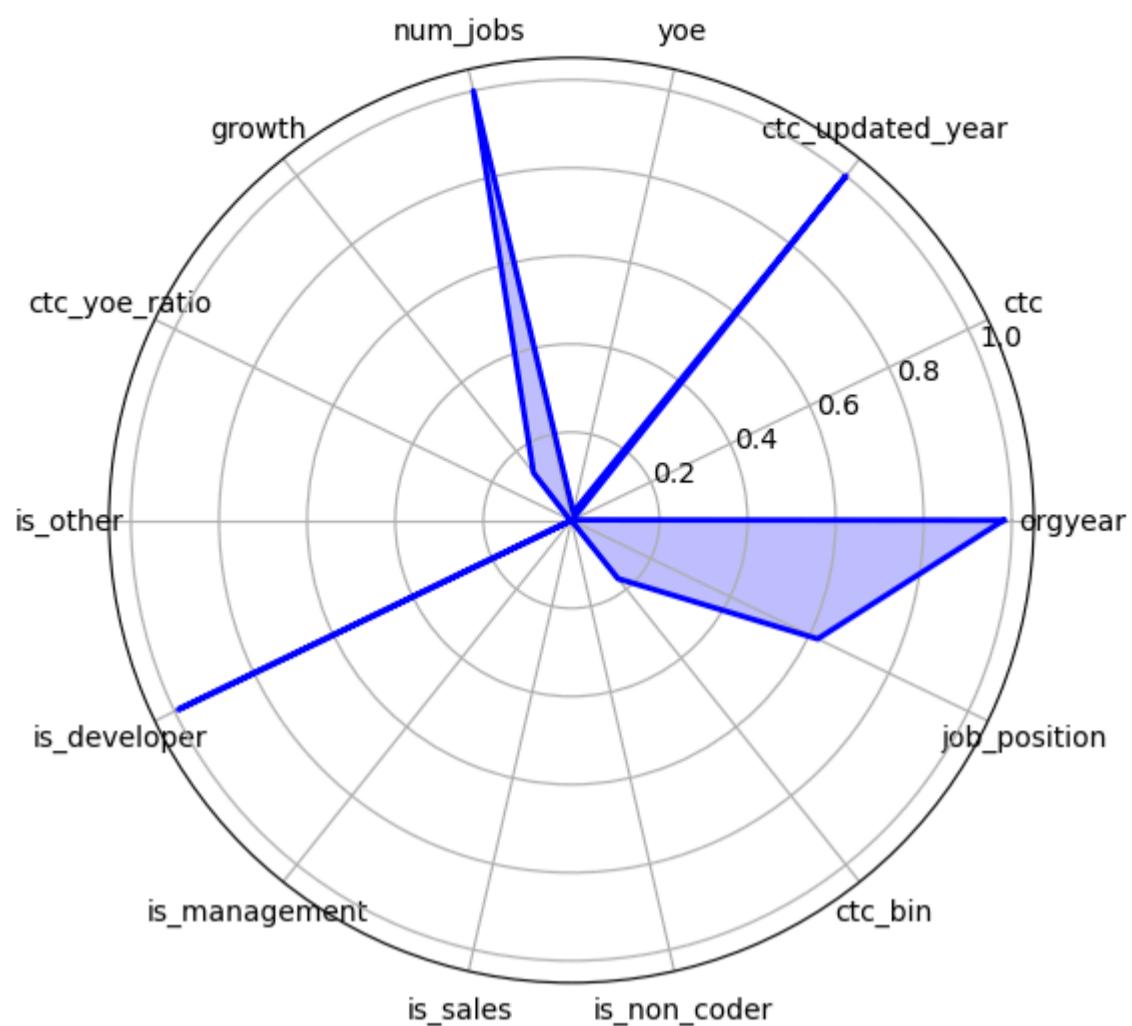
Cluster 3



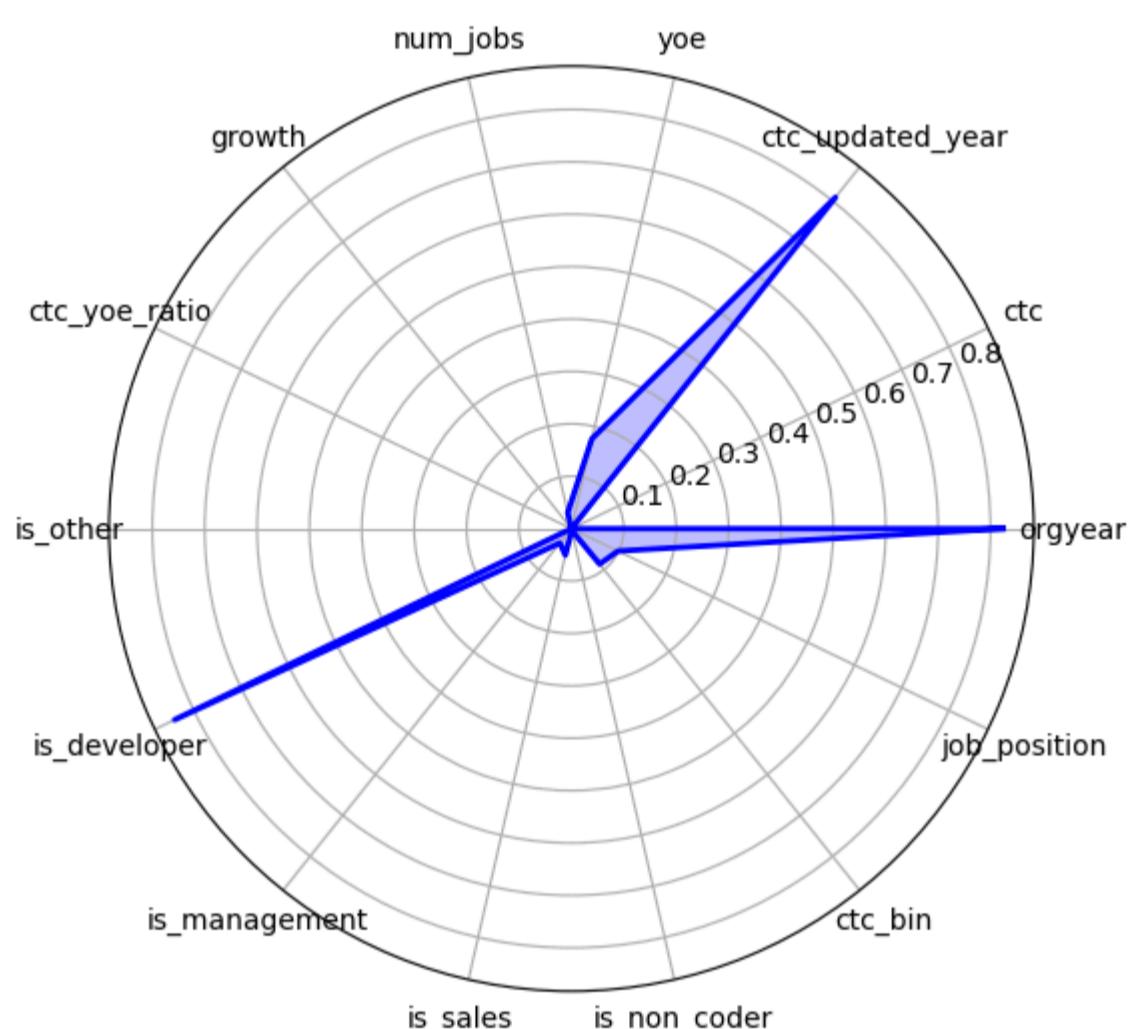
Cluster 4



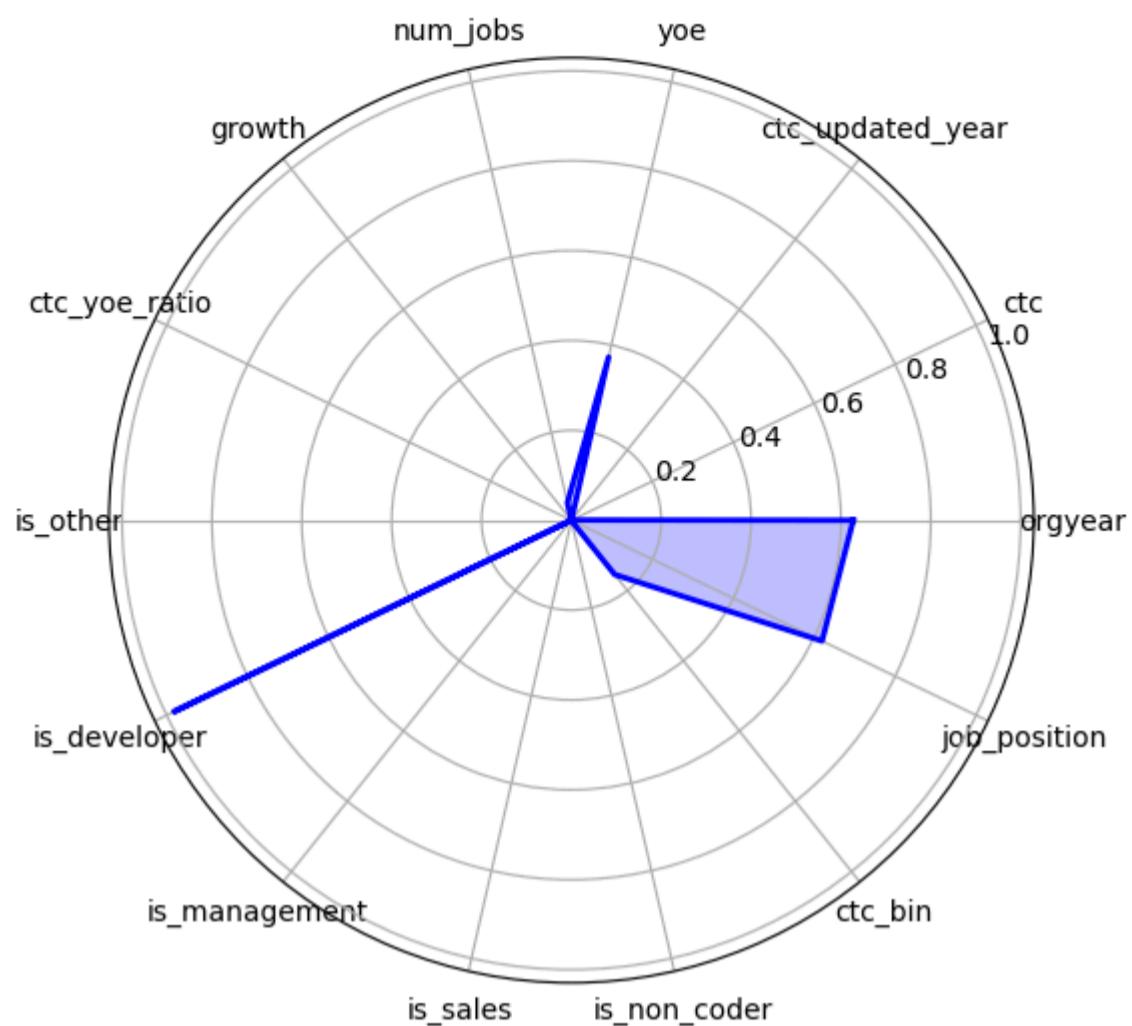
Cluster 5



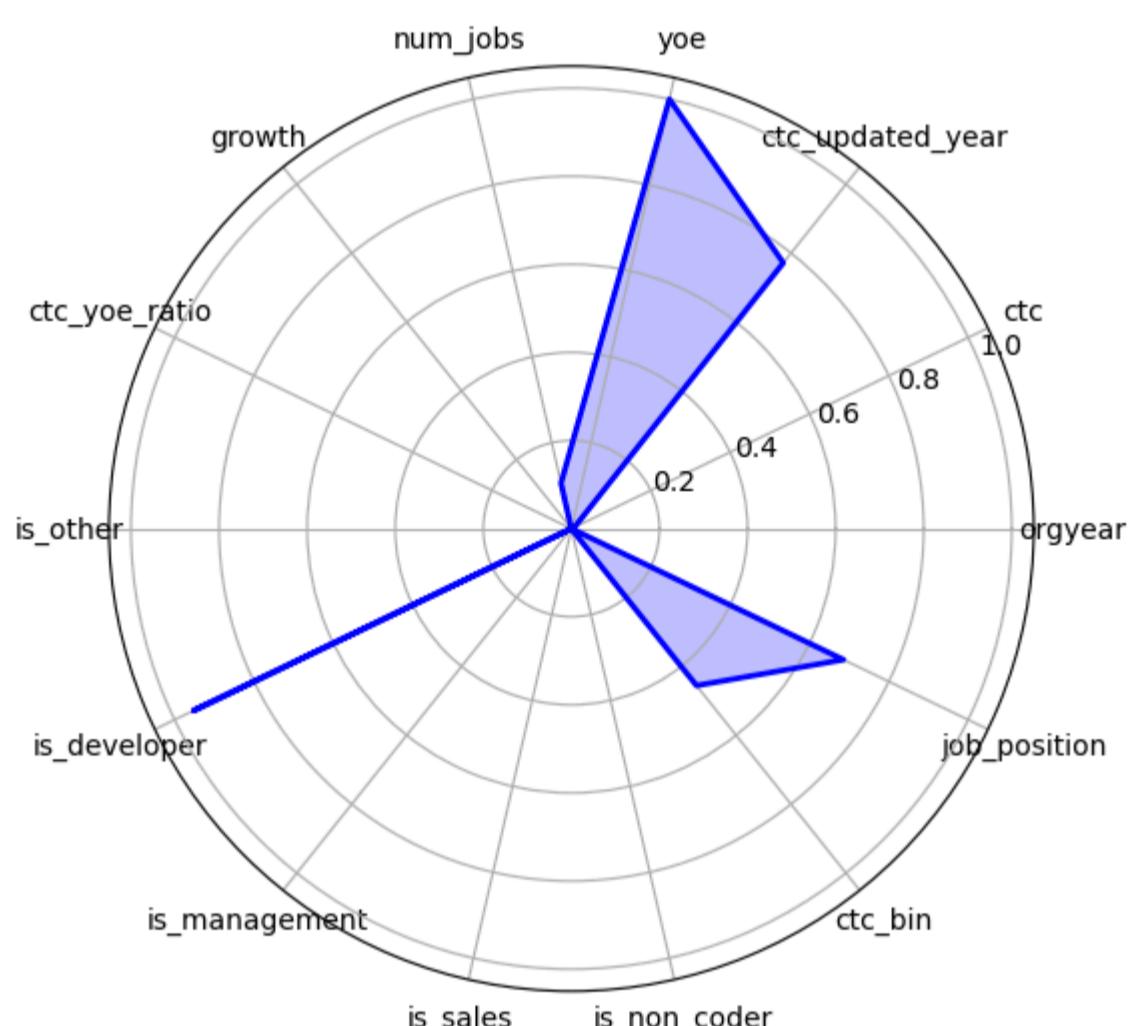
Cluster 6



Cluster 7



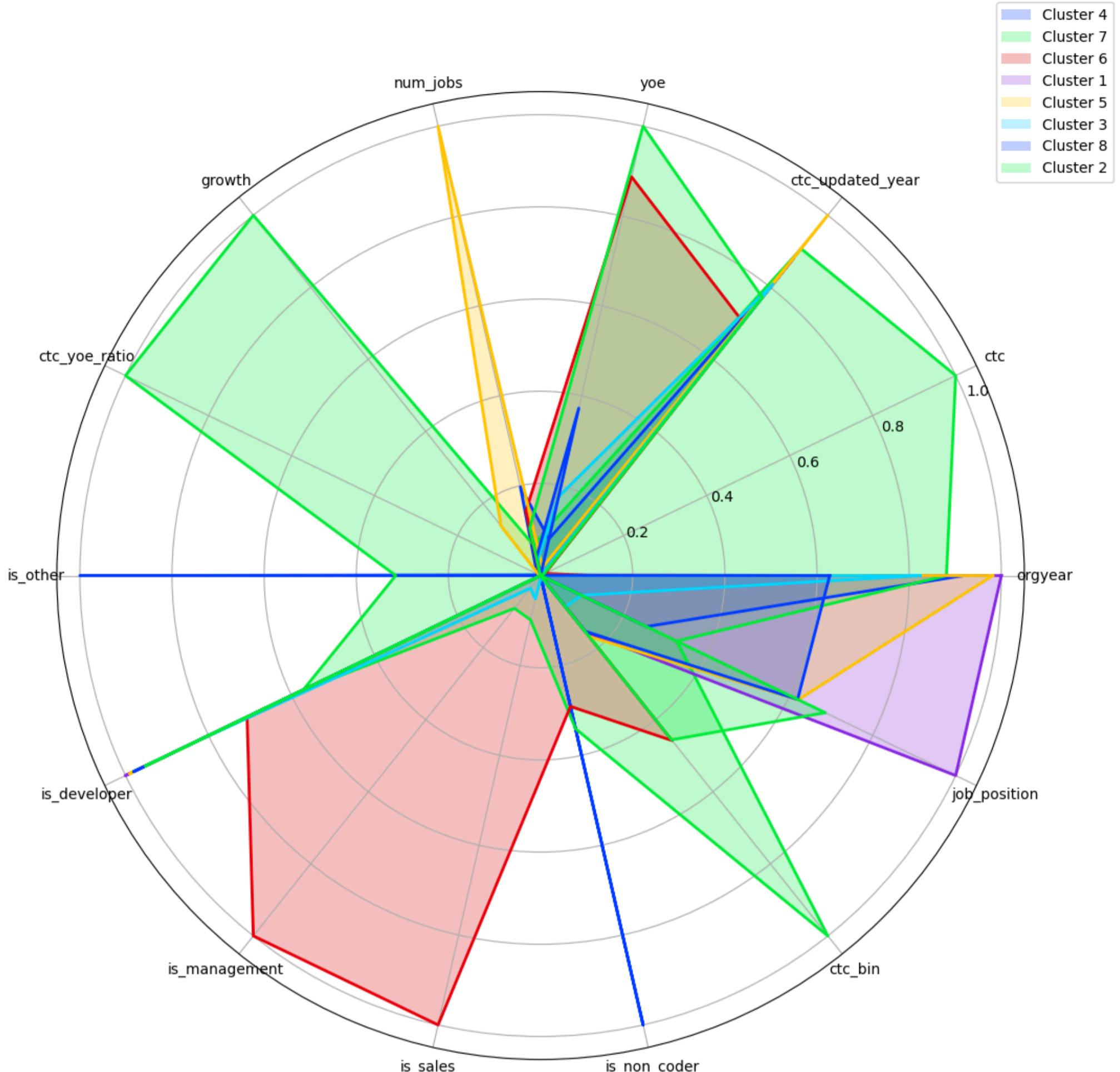
Cluster 8



```
In [224]: # Get the unique labels from the 'cluster' column and convert them to a list
cluster_labels = sample_hc['cluster'].dropna().unique().tolist()

# Create a combined polar plot for all clusters
create_combined_polar_plot(normalized_means, features, cluster_labels)
```

Cluster Comparison



CLUSTER Insights

Cluster - 1 . Stable Transitioners

(Candidates in this cluster belong to Other and Non-Coder categories with a recent hike.)

Cluster - 2 . Elite Achievers

(Candidates in this cluster belong to Developer, Non-Coder, and Other categories, with very high growth, very high pay, recent hikes, and fewer job switches.)

Cluster - 3 . Underpaid Veterans

(Candidates in this cluster belong to Developer, Management, and Sales categories, with very high experience but are paid less.)

Cluster - 4 . Frequent Switchers

(Candidates in this cluster belong to the Developer category with recent hikes and a high number of job switches.)

Cluster - 5 . Growth Seekers

(Candidates in this cluster belong to the Developer category, with average job switches and recent hikes.)

Cluster - 6 . Committed Achievers

(Candidates in this cluster belong to the Developer category, with almost no job switches and recent hikes.)

Cluster - 7 . Upskilling Needed

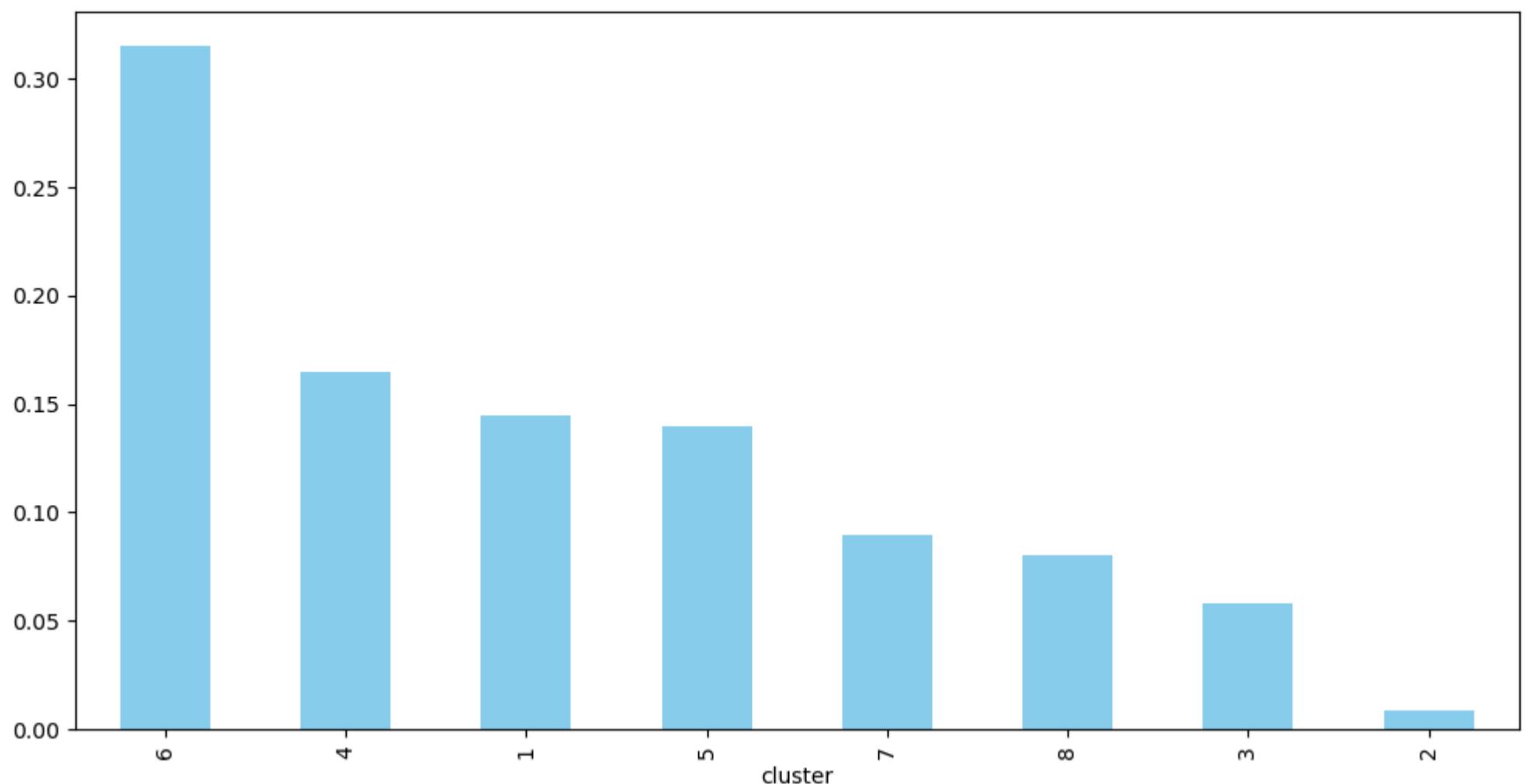
(Candidates in this cluster belong to the Developer category, with average job switches, low experience, no hikes, and a need for new skills.)

Cluster - 8 . Experienced Undervalued

(Candidates in this cluster belong to the Developer category, with very high experience, average job switches, but are paid less.)

```
In [225]: sample_hc['cluster'].value_counts(normalize=True).plot(kind='bar', figsize=(12, 6), color='skyblue')
```

Out[225]: <Axes: xlabel='cluster'>



💡 Insights

- Cluster 6 containing developers has the highest count and cluster 2 containing non coders has the lowest count.

8. Questionnaire Answers

1. What percentage of users fall into the largest cluster?

Answer: Around 34% of learners fall into the largest cluster, 1

2. Comment on the characteristics that differentiate the primary clusters from each other.

Answer: CTC, Years of experience (YOE), CTC to YOE Ratio, Designation, Class and Tier are the most important characteristics that differentiate the clusters.

3. Is it always true that with an increase in years of experience, the CTC increases? Provide a case where this isn't true.

Answer: No, it is not true that CTC increases with increase in experience. The maximum CTC belongs to a learner with 7 years of experience and the minimum CTC belongs to a 29 year experienced learner

4. Name a job position that is commonly considered entry-level but has a few learners with unusually high CTCs in the dataset.

Answer: Data Analysts is usually considered a entry-level job and there is a learner with Data Analyst job position with a CTC higher than that of a learner with engineering leadership job position

5. What is the average CTC of learners across different job positions?

Answer: Average CTC across different job positions is 12 Lakhs (Overall)

Here's a List of Top 5 Jobs with their Average CTC:

- **Frontend Engineer** 23,23,741.050
- **Backend Engineer** 20,74,889.063
- **QA Engineer** 20,27,199.074
- **Fullstack Engineer** 19,78,556.112
- **Data Scientist** 19,35,848.904

6. For a given company, how does the average CTC of a Data Scientist compare with other roles?

Answer: There are around 900+ companies in which more than 50% of the times the average CTC of a Data Scientist is greater than that of other roles

7. Discuss the distribution of learners based on the Tier flag:

- 7.1. Which companies dominate in Tier 1 and why might this be the case?

- **Answer:** The companies which offer Software Developer roles are the companies which dominate Tier 1 as Software Developers are in high demand due to the exponential growth of AI

- 7.2. Are there any notable patterns or insights when comparing learners from Tier 3 across different companies

▪ **Answer:** The learners have a mean of 7 years of experience with CTC being on the lower end.

8. After performing unsupervised clustering:

- 8.1. How many clusters have been identified using the Elbow method?

▪ **Answer:** The elbow method gave two elbows, one at 4 and another at 7

- 8.2. Do the clusters formed align or differ significantly from the manual clustering efforts? If so, in what way

▪ **Answer:** The clusters formed are slightly better than the manual clustering. The manual clustering was able to differentiate between CTCs and years of experience clearly and job_position partially. The clusters formed due to kMeans are able to differentiate CTC, years of experience and job position(encoded in class column) clearly.

9. From the Hierarchical Clustering results:

- 9.1. Are there any clear hierarchies or patterns formed that could suggest the different levels of seniority or roles within a company?

▪ **Answer:** Yes

- 9.2 How does the dendrogram representation correlate with the 'Years of Experience' feature?

▪ **Answer:** We can see a clear pattern that the low number of years of experience are grouped together on the extreme left of the dendrogram, medium number of years of experience are grouped together in the middle and high number of years of experience are grouped together at the extreme right?.s.

9. Actionable Insights and Recommendations

Insights

1. General Dataset Insights

- The dataset comprises **205,843 rows and 7 columns**, indicating substantial data for analysis.
- A majority of the candidates belong to the **web development domain** and have **less than 10 years of experience**.
- Most CTC updates occurred between **2019 and 2021**, reflecting a significant period of salary revisions.
- A notable number of candidates joined after **2010**, with some extending back to pre-2000s, indicating a diverse workforce.
- The **mean salary** is **₹12 Lakhs**, while the **median salary** is **₹10 Lakhs**, signifying a skewed salary distribution influenced by outliers.

2. Tier-Level Insights

- Tier 3 candidates tend to remain **with the same company without frequent switches**, which may reflect limited career growth opportunities.

3. CTC Variance

- A significant variance in CTC relative to years of experience suggests **diverse salary structures**, influenced by job roles, domains, and company policies.

4. Clustering Analysis

- **Hopkins Statistic** of **0.9999** confirms a strong clustering tendency, validating the application of clustering methods.
- Insights from K-Means and Hierarchical clustering provide distinct learner profiles, helping tailor actionable strategies.

K-Means Cluster Insights

1. Underpaid Switchers:

Action: Analyze causes of underpayment (skill gaps, domain shifts) and offer targeted skill enhancement programs.

2. Ambitious Movers:

Action: Highlight career path optimization strategies to sustain growth and reduce unnecessary switching.

3. Stagnant Movers:

Action: Focus on skill upgrades and market-readiness to secure better-paying roles.

4. Professionally Stuck:

Action: Provide intensive upskilling programs and mentorship to reignite career trajectories.

5. Elite Achievers:

Action: Utilize these individuals for testimonials, advanced technical tracks, or leadership programs.

6. Underpaid Veterans:

Action: Design transition pathways for moving into higher-paying, hybrid roles like tech management.

Hierarchical Cluster Insights

1. Stable Transitioners:

Action: Identify opportunities for sustainable growth in roles suited to their backgrounds.

2. Elite Achievers:

Action: Leverage their success stories for marketing and encourage cross-skilling opportunities.

3. Underpaid Veterans: Developers, managers, and sales professionals with high experience but low CTC.

Action: Promote domain transitions or certifications in high-demand skills.

4. Frequent Switchers: Developers with recent hikes and frequent job changes.

Action: Foster skills that emphasize stability and long-term career growth.

5. Growth Seekers: Developers with recent hikes and average job switches.

Action: Encourage specialized certifications to accelerate career growth.

6. Committed Achievers: Developers with recent hikes and minimal job switches.

Action: Reward their commitment through advanced leadership or technical tracks.

7. Upskilling Needed: Developers with low experience, no hikes, and a need for new skills.

Action: Offer foundational and intermediate programs in trending domains.

8. Experienced Undervalued: Developers with high experience but low pay.

Action: Design salary negotiation workshops and advanced courses for their expertise.

Recommendations

1. Targeted Programs for Professionally Stuck and Stagnant Movers:

- Create personalized programs addressing skill gaps and market trends.
- Use success stories of similar learners for video testimonials.

2. Enhanced Outreach and Acquisition:

- Identify and engage with learners in low-paying companies and stagnant job roles through LinkedIn campaigns.

3. High-Value Program Development:

- Introduce niche programs like **Gen AI + MBA**, tailored for transitioning business professionals.
- Expand offerings in software development, data analysis, and leadership skills for junior and mid-level professionals.

4. Course Optimization:

- Utilize cluster insights to create tracks that cater to specific growth areas like technical expertise or role transitions.

5. Corporate Partnerships:

- Collaborate with companies offering high CTC to streamline placements.
- Work with low-paying companies to improve employee growth opportunities.

6. Algorithm Improvements:

- Request detailed learner inputs, such as precise job positions and domains, to refine clustering algorithms.

7. Marketing Strategies:

- Showcase roles like **software development** and **data science** with their potential for high salaries in ads.
- Highlight programs for individuals stuck in stagnant roles, focusing on successful transitions.

8. Investigate Outliers:

- Validate the accuracy of extreme CTC data entries to avoid skewed results.

9. Upskilling Academia:

- Attract students and educators from academia to make them industry-ready and employable with competitive salaries.

10. Retention Support for Learners:

- Organize master sessions featuring placed learners to inspire others in similar clusters.