

Scaler Clustering Case Study

Gautam Naik (gautamnaik1994@gmail.com)

Github link: <https://github.com/gautamnaik1994/ScalerJobDataClusteringCaseStudy>

[render](#) [nbviewer](#) [Open in Colab](#)

Business Problem

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

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

We will be using following techniques to cluster the data:

- Hopkins Test to check clustering tendency
- Manual Clustering
- Unsupervised Clustering - K-means, Hierarchical Clustering

After this post cluster analysis will be done to understand the clusters and their characteristics. Polar plots will be used to compare the clusters characteristics. UMAP and PCA will be used to visualize the clusters in 2D space.

Dataset columns

Column Name	Description
'Unnamed 0'	Index of the dataset
Email_hash	Anonymised Personal Identifiable Information (PII)
Company_hash	This represents an anonymized identifier for the company, which is the current employer of the learner.
orgyear	Employment start date
CTC	Current CTC
Job_position	Job profile in the company
CTC_updated_year	Year in which CTC got updated (Yearly increments, Promotions)

```
In [1]: import pandas as pd
import numpy as np
import polars as pl
import duckdb as db
import re

import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="whitegrid")

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

import warnings
warnings.filterwarnings("ignore")

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

from collections import Counter
sns.set_theme(style="whitegrid")
```

Intel(R) Extension for Scikit-learn* enabled (<https://github.com/intel/scikit-learn-intelex>)
2024-09-26 10:38:54.305954: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

Data Sanity Check

```
In [204...]: df=pl.read_csv("../data/raw/data.csv", has_header=True)

In [5]: df.head()

Out[5]: shape: (5, 7)
      company_hash          email_hash  orgyear      ctc    job_position  ctc_updated_year
0       atrgxnnnt xzaxv"   "6de0a4417d18ab14334c3f43397fc1...  2016.0  1100000        "Other"      2020.0
1   "qtrxvzwt xzegwgbb rxbxnta"  "b0aaaf1ac138b53cb6e039ba2c3d660...  2018.0  4499999  "FullStack Engineer"      2019.0
2       "ojzwnvwnxw vx"   "4860c670bcd48fb96c02a4b0ae3608...  2015.0  2000000  "Backend Engineer"      2020.0
3       "ngpgutaxv"     "efffdede7a2e7c2af664c8a31d93463...  2017.0  7000000  "Backend Engineer"      2019.0
4       "qxen sqghu"   "6fff54e709262f55cb999a1c1db8436...  2017.0  1400000  "FullStack Engineer"      2019.0
```

statistic	company_hash			email_hash	orgyear	ctc	job_position	ctc_updated_year
	str	f64	str		str	f64	str	f64
"count"	205843.0	"205799"		"205843"	205757.0	205843.0	"153281"	205843.0
"null_count"	0.0	"44"		"0"	86.0	0.0	"52562"	0.0
"mean"	103273.941786	null		null	2014.88275	2.2717e6	null	2019.628231
"std"	59741.306484	null		null	63.571115	1.1801e7	null	1.325104
"min"	0.0	"0"		"00003288036a44374976948c327f24..."	0.0	2.0	"SDE 2"	2015.0
"25%"	51519.0	null		null	2013.0	530000.0	null	2019.0
"50%"	103151.0	null		null	2016.0	950000.0	null	2020.0
"75%"	154993.0	null		null	2018.0	1.7e6	null	2021.0
"max"	206922.0	"zzzbzb"		"fffffa3eb3575f43b86d986911463dc..."	20165.0	1.0002e9	"technology analyst"	2021.0

Observations

- From above table we can see that 2 lakh entries are there in the dataset.
- There are 44 missing values in company hash and 86 missing values in orgyear.
- The minimum value of ctc is 2.0 which can be an error or an outlier.
- The minimum value of orgyear is 0.0 which is not possible.
- The max value of orgyear is 20165.0 which is not possible.

In [206]: `df["email_hash"].unique().count(), df["company_hash"].unique().count()`

Out[206]: (153443, 37299)

In [7]: `df.null_count()`

Out[7]: shape: (1, 7)

company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year	
u32	u32	u32	u32	u32	u32	
0	44	0	86	0	52562	0

Data Cleaning and Preprocessing

In [8]: `df.filter(df["orgyear"].is_null())`

Out[8]: shape: (86, 7)

company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year	
i64	str	str	f64	i64	str	f64
2333 "xgmgn ntwyzgrgsxto ucn rna"	"c737ceb66c7f0ce37c2fce087003aa..."	null	170000	"Other"	2020.0	
2562 "tj"	"25edac17c77f6f0edeafb86f7a7844..."	null	860000	"Data Analyst"	2020.0	
3122 "ft tdwtr"	"c402eba160abf4e5b5f72af775fc98..."	null	600000	"Support Engineer"	2020.0	
3365 "fyxntyvn lq"	"38bd913564fa983cd4fb7799e4027d..."	null	800000	null	2021.0	
4010 "nxamxn ogrhnxgzo"	"f9c1d0e623046d445a705bc39007f0..."	null	900000	null	2020.0	
...
179152 "ohbg rgsxw"	"60c2f0c5b719160991e78a7c00b709..."	null	3600000	null	2020.0	
179445 "vr bhrrv sqghu"	"87febfbadc6689a29e33ff30d7cd40..."	null	600000	"Android Engineer"	2019.0	
182509 "fgqpxzs vo v eqtrrvzwtq"	"bf814503a271c8832095892856e176..."	null	1200000	"Product Designer"	2019.0	
195074 "sgavaaj"	"638ffe59e3f96d6c2e182979680d46..."	null	1800000	"Frontend Engineer"	2018.0	
206515 "vhngsqxa"	"3fa8de870da01d863abba8eb6a8ae3..."	null	2400000	null	2020.0	

Mean and Mode Imputation

In [15]: `df=df.with_columns(pl.col("company_hash").fill_null(df["company_hash"].max()), pl.col("orgyear").fill_null(df["orgyear"].mean()).cast(pl.Int64),)`

Out[15]: shape: (205_843, 7)

company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year	
i64	str	str	i64	i64	str	f64
0 "atrxnnt xzaxv"	"6de0a4417d18ab14334c3f43397fc1..."	2016	1100000	"Other"	2020.0	
1 "qtrxvzwt xzegwgb rxbxnta"	"b0aaaf1ac138b53cb6e039ba2c3d660..."	2018	449999	"FullStack Engineer"	2019.0	
2 "ojzwnvwnxw vx"	"4860c670bcd48fb96c02a4b0ae3608..."	2015	2000000	"Backend Engineer"	2020.0	
3 "ngpgutaxv"	"efffdede7a2e7c2af664c8a31d93463..."	2017	700000	"Backend Engineer"	2019.0	
4 "qxen sqghu"	"6ff54e709262f55cb999a1c1db8436..."	2017	1400000	"FullStack Engineer"	2019.0	
...
206918 "vuurt xzw"	"70027b728c8ee901fe979533ed94ff..."	2008	220000	null	2019.0	
206919 "husqvawgb"	"7f7292ffad724ebbe9ca860f515245..."	2017	500000	null	2020.0	
206920 "wwwgrxnt"	"cb25cc7304e9a24facda7f5567c792..."	2021	700000	null	2021.0	
206921 "zgn vuurxwmrt"	"fb46a1a2752f5f652ce634f6178d05..."	2019	5100000	null	2019.0	
206922 "bgqsvz onvzrtj"	"0bcfc1d05f2e8dc4147743a131aa7..."	2014	1240000	null	2016.0	

In [91]: `df=df.drop_nulls(subset=["job_position"])
df = df.drop("")
df=df.unique()
df = df.with_columns(
 df["orgyear"].cast(pl.Int32),
 df["ctc_updated_year"].cast(pl.Int32)
)`

Unique email and company values

In [92]: `df["email_hash"].unique().count(), df["company_hash"].unique().count()`

Out[92]: (133146, 34203)

Fixing the job position column

In [93]: `df = df.with_columns(pl.col("job_position").fill_null(pl.list("unknown")))
df = df.with_columns(df["job_position"].str.to_lowercase())
df = df.with_columns(df["job_position"].str.replace("-", " "))
df = df.with_columns(df["job_position"].str.replace(r"[\^a-zA-Z0-9]", ""))
df = df.with_columns(df["job_position"].str.replace("\\" , ""))`

```

df = df.with_columns(df["job_position"].str.replace("\(", " "))
df = df.with_columns_seq(
    df["job_position"]
    .str.replace("i1", "2")
    .str.replace(r" i\b", " 1")
    .str.replace("iii", "3")
    # .str.replace(r"\", " 1")
    # .str.replace(r"\(b|w+)\b", r"\{1\} \1")
    .str.replace("sr ", "senior ")
    .str.replace("jr ", "junior ")
)

df = df.with_columns_seq(
    df["job_position"]
    .str.replace("front end", "frontend")
    .str.replace("back end", "backend")
    .str.replace("full stack", "fullstack")
    .str.replace("sde2", "sde 2")
    .str.replace("sde3", "sde 3")
    .str.replace("se4", "se 4")
    .str.replace("2i", "2")
    .str.replace(r"\br \d\b", "rd")
    .str.replace(r"enginner|engineers|engineer|engineet|eengineer", "engineer")
    .str.replace(r"engg\b", "engineer")
    .str.replace("applications", "application")
    .str.replace(r"dev\b", "developer")
    .str.replace(r"devloper|develloper", "developer")
    .str.replace("consultant", "consultant")
    .str.replace(r"others\b", "other")
    .str.replace(" of ", " ")
    .str.replace("tech ", "technical ")
    .str.replace_all(r"development(\w+)", r"development ${1}")
    # .str.replace(r"\b\d+\b", "unknown")
    # .str.replace(r"\b\w\b", "unknown")
    # .str.replace(r"\.", "unknown")
    # replace empty strings with unknown
    # .fill_null("unknown")
)

df = df.with_columns(df["job_position"].str.replace(" ", ""))
df = df.with_columns(df["job_position"].str.strip_chars())

```

Spellings Mistakes Correction

```

In [94]: words = df["job_position"].str.split(" ").explode()
WORDS = Counter(words.to_list())

In [95]: # https://norvig.com/spell-correct.html

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

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

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

def known(words):
    "The 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))

```

```

In [96]: def correct_words(sentence):
    return " ".join([correction(word) for word in sentence.split()])

def clean_jobs(word):
    if len(word) <= 1:
        return "unknown"
    try:
        float(word)
        return "unknown"
    except ValueError:
        return word

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

```

```

In [97]: df = df.with_columns(pl.col('job_position').map_elements(correct_words).alias("job_position"))
df = df.with_columns(pl.col('job_position').map_elements(clean_jobs).alias("job_position"))
df = df.with_columns(pl.col('job_position').map_elements(replace_words).alias("job_position"))

```

```
In [98]: df=df.unique()
```

Org Year Fix

```
In [99]: df.filter(pl.col("orgyear") < 1970).sort("orgyear", descending=True)
```

Out[99]: shape: (26, 6)

company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
str	str	i32	i64	str	i32
"ovst xznvwnn ogenfvqt ucn rna"	"3da3c964efe7a5db0f63e91be25391...	1900	3170000	"engineering leadership"	2019
"wgvxnqxm ntwy exzntwy ctzhqht"	"a9393ceb6f438669a12e2f5632d515...	201	3500000	"product manager"	2017
"wxowg cxatg ntwyzgrgsxto xzaxv..."	"c3cce99fc54361b5c213f8043505d2...	200	3000000	"engineering leadership"	2019
"vzshrvq atcqrgutq"	"1978da71c14333352d051bfb605490...	91	400000	"devops engineer"	2021
"bvzyvnnvz voogwxvnto"	"e725ad631cdc4c57a354f59c98b644...	83	730000	"backend engineer"	2019
...
"wyvqstugxzn"	"a1da4f131e2feea24f6d64c6651972...	0	1850000	"backend engineer"	2019
"wyvqstugxzn"	"a1da4f131e2feea24f6d64c6651972...	0	1850000	"fullstack engineer"	2019
"cvrhjeogrhnxgzo"	"a184e9709d7a9adfc1e2d500770e92...	0	1300000	"backend engineer"	2020
"rgsfvqa"	"b511b17a1aaaa822c6b5d6a04498ff...	0	1600000	"frontend engineer"	2021
"bxznb xzw"	"5bbebf13d3b3edc497bc5e24d1ab54...	0	700000	"fullstack engineer"	2019

```
In [100... df=df.with_columns(
    pl.when(pl.col("orgyear") < 1970).then(pl.col("ctc_updated_year")-1).otherwise(pl.col("orgyear")).alias("orgyear"))
)
```

```
In [101... df.filter(df["orgyear"]> df["ctc_updated_year"]).sort("orgyear", descending=True)
```

Out[101... shape: (5_016, 6)

company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
str	str	i32	i64	str	i32
"lu bgqsgz"	"799dff77b331bfa04cf005935acf7...	2204	2160000	"data scientist"	2018
"hufgqp"	"e4705330da833368780aae561afc93...	2031	1100000	"fullstack engineer"	2019
"hufgqp"	"e4705330da833368780aae561afc93...	2031	1100000	"backend engineer"	2019
"hufgqp"	"e4705330da833368780aae561afc93...	2031	1100000	"other"	2019
"phrxkv"	"3394674bb6bb1de6289e931853fa0b...	2031	1500000	"backend engineer"	2020
...
"mvqwrvjo wgqgqvnt mvzpxzs"	"f2da8af7e11da5d6c7e73670fbe8e0...	2016	1100000	"pune"	2015
"bgmonvw"	"648e830d20e1053c54e8bb2a82f96f...	2016	900000	"fullstack engineer"	2015
"nvnv wzohrnvwj otqcxwto"	"cd681b4485116e1a46c144f2d87ebc...	2016	420000	"backend engineer"	2015
"svrvdtogrhnxgzo"	"c6789faa51e7f32f19908baaeb5cb1...	2016	1100000	"backend engineer"	2015
"ovu"	"b652f5720ca7489ea586203e1b24fd...	2016	930000	"fullstack engineer"	2015

Observations

- From above table we can see that around 8k candidates have org_year greater than ctc updated year which is not possible.
- Need to fix this by subtracting 1 from ctc updated year.

```
In [102... df = df.with_columns(
    orgyear=pl.when(pl.col("orgyear") > pl.col("ctc_updated_year"))
        .then(pl.col("ctc_updated_year") - 1)
        .otherwise(pl.col("orgyear"))
)
```

CTC Fix

```
In [103... df.filter(df["ctc"] <= 100)
```

Out[103... shape: (3, 6)

company_hash	email_hash	orgyear	ctc	job_position	ctc_updated_year
str	str	i32	i64	str	i32
"xzntqcxtfmxn"	"3505b02549ebe2c95840ac6f0a3556...	2014	2	"backend engineer"	2019
"uqvpqnxn voogwxvnto"	"8786759b95d673466e94f62f1b15e4...	2020	24	"other"	2020
"ftm ongqt"	"512f761579fb116e215cabc9821c7f...	2016	25	"android engineer"	2018

Observations

- From above table it is fair to assume that CTC is in lakhs.
- These rows CTCs should be multiplied by 100000.

```
In [104... df = df.with_columns(
    ctc=pl.when(pl.col("ctc") < 100)
        .then(pl.col("ctc") * 100000)
        .otherwise(pl.col("ctc"))
)
```

```
In [105... position_count_df=df.groupby("job_position").agg(pl.count("job_position").alias("count")).sort("count", descending=True)
position_count_df
```

Out[105... shape: (813, 2)

job_position	count
str	u32
"backend engineer"	43532
"fullstack engineer"	25873
"other"	18060
"frontend engineer"	10413
"engineering leadership"	6867
...	...
"professional services engineer"	1
"networking"	1
"member technical staff level 2"	1
"senior chat associate"	1
"software engineer iv"	1

Using top position from each company to fill unknown values of job position

```
In [108... # filtered_df = df.filter(
#     (pl.col("job_position") != 'unknown') & (pl.col("company_hash").is_not_null())
# )
# grouped_df = filtered_df.groupby(["company_hash", "job_position"]).agg(
```

```

#     pl.count().alias("count")
# )
# ranked_df = grouped_df.with_columns(
#     pl.col("count").rank("min").over("company_hash").reverse().alias("rank")
# )
# top_position_df = ranked_df.filter(pl.col("rank") == 1).select("company_hash", "job_position").sort("company_hash").unique()

top_position_df = db.sql("""
    with cte as (
        select company_hash, job_position, count(*) as cnt,
               row_number() over (partition by company_hash order by count(*) desc) as rn
        from df
        where job_position != 'unknown'
          and company_hash in (
              select distinct company_hash
              from df
              where job_position = 'unknown'
                and company_hash is not null
            )
        group by company_hash, job_position
    )
    select company_hash, job_position
    from cte
    where rn = 1
""").pl()

```

```
In [109... df=df.join(top_position_df, on="company_hash", how="left").unique()
df = df.with_columns(pl.col("job_position_right").fill_null(pl.lit("unknown")))
df= df.with_columns(
    pl.when(pl.col("job_position") == "unknown"
        .then(pl.col("job_position_right"))
        .otherwise(pl.col("job_position")).alias("job_position")
    )
df = df.drop(["job_position_right"])

```

```
In [110... df["job_position"].value_counts(sort=True)
```

```
Out[110... shape: (813, 2)
```

job_position	count
"backend engineer"	43534
"fullstack engineer"	25872
"other"	18063
"frontend engineer"	10413
"engineering leadership"	6867
...	...
"senior software engineer"	1
"any technical"	1
"senior webmethods developer"	1
"senior technical manager"	1
"oracle apps technical consulta..."	1

```
In [111... df.columns
```

```
Out[111... ['company_hash',
'email_hash',
'orgyear',
'ctc',
'job_position',
'ctc_updated_year']
```

```
In [117... # df_unique=db.sql("""
#     with cte as (
#         select company_hash, job_position, email_hash, orgyear, ctc, ctc_updated_year,
#             row_number() over (partition by email_hash order by orgyear desc, ctc_updated_year desc, ctc desc) as rn
#         from df
#     )
#     select company_hash, job_position, email_hash, orgyear, ctc, ctc_updated_year from cte where rn = 1
# """).pl()
```

```
In [42]: df=df.unique()
```

```
In [119... df.write_parquet("../data/cleaned/data_duplicates.parquet")
df_unique.write_parquet("../data/cleaned/data_de_duplicates.parquet")
```

Feature Engineering

```
In [35]: df=pl.read_parquet("../data/cleaned/data_duplicates.parquet")
```

Creating Years of Experience Column

```
In [37]: df["orgyear"].max(), df["ctc_updated_year"].max()
```

```
Out[37]: (2021, 2021)
```

Using 2022 as the current year as 2021 is the last year in the dataset.

```
In [38]: df=df.with_columns(
    yoe = 2022 - pl.col("orgyear")
)
```

Extracting unique candidates from the dataset because of duplicate values

```
In [39]: df=db.sql("""
    with cte as (
        select company_hash, job_position, email_hash, orgyear, ctc, ctc_updated_year, yoe,
               row_number() over (partition by email_hash order by orgyear desc, ctc_updated_year desc, ctc desc) as rn,
               -- avg(ctc) over (partition by email_hash) as avg_ctc,
               count(*) over (partition by email_hash) as num_jobs,
               (max(ctc) over (partition by email_hash) - min(ctc) over (partition by email_hash))/min(ctc) over (partition by email_hash) as growth
        from df
    )
    select * from cte where rn = 1 order by num_jobs desc
""").pl().drop(["rn"])
```

```
In [45]: df["yoe"].min(), df["yoe"].max()
```

```
Out[45]: (1, 52)
```

```
In [46]: df=df.with_columns(
    ctc_to_exp_ratio = pl.col("ctc") / (pl.col("yoe")*12*1000),
)
```

```
In [47]: df
```

company_hash	job_position	email_hash	orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio
str	str	str	i32	i64	i32	i32	i64	f64	f64
"wgcxvb ntwygrgsxto"	"data analyst"	"3e5e49daa5527a6d5a33599b238bf...	2018	1130000	2021	4	9	0.041475	23.541667
"ihvrwgbb"	"devops engineer"	"6842660273f70e9aa239026ba33bfe..."	2017	2400000	2020	5	9	0.2	40.0
"nyt a t oyvf sqghu"	"sdet"	"c0eb129061675da412b0d0eb15871dd..."	2013	2300000	2020	9	8	0.513158	21.296296
"oxej ntwygrgsxto rxbxnta"	"other"	"bbace3cc586400bbc65765bc6a16b7..."	2018	720000	2020	4	8	0.090909	15.0
"cvrhtgbtznhb"	"other"	"298528ce3160cc761e4dc37a07337e..."	2018	720000	2020	4	8	0.028571	15.0
...
"so rvnm"	"devops engineer"	"6d5acda3aa50c201f07b60ae58209a..."	2014	800000	2019	8	1	0.0	8.333333
"kxrxxzsg"	"android engineer"	"6d5f2d2389d92e2dee3dbc13d7ad2c..."	2014	2500000	2020	8	1	0.0	26.041667
"r3 ntwygrgsxto"	"other"	"6d60bc429c1a5a542a3db72ea45e70..."	2017	530000	2020	5	1	0.0	8.833333
"vrnqvz vqxwtzn tqonfyxrt ntwy..."	"backend engineer"	"6d6d29d229ecc859d2238f5b756022..."	2013	1250000	2019	9	1	0.0	11.574074
"qxcxsg"	"other"	"6d8712d2fe8732a1e8829644fb7a62..."	2017	360000	2020	5	1	0.0	6.0

Creating individual groups for manual clustering

```
In [48]: g1 = df.groupby(["company_hash", "job_position", "yoel"]).agg(
    pl.mean("ctc").alias("mean_ctc"),
    pl.median("ctc").alias("median_ctc"),
    pl.max("ctc").alias("max_ctc"),
    pl.min("ctc").alias("min_ctc"),
    pl.count("ctc").alias("count"),
).sort("count", descending=True)
g1

g2 = df.groupby(["company_hash", "job_position"]).agg(
    pl.mean("ctc").alias("mean_ctc"),
    pl.median("ctc").alias("median_ctc"),
    pl.max("ctc").alias("max_ctc"),
    pl.min("ctc").alias("min_ctc"),
    pl.count("ctc").alias("count"),
).sort(["count"], descending=True)
g2

g3 = df.groupby(["company_hash"]).agg(
    pl.mean("ctc").alias("mean_ctc"),
    pl.median("ctc").alias("median_ctc"),
    pl.max("ctc").alias("max_ctc"),
    pl.min("ctc").alias("min_ctc"),
    pl.count("ctc").alias("count"),
).sort("count", descending=True)
g3
```

Out[48]: shape: (84_222, 8)

company_hash	job_position	yoel	mean_ctc	median_ctc	max_ctc	min_ctc	count
str	str	i32	f64	f64	i64	i64	u32
"eqtoytq"	"other"	3	1.6407e6	610000.0	200000000	180000	244
"nvnv wgzohrnvwj otqcxwto"	"backend engineer"	4	817240.504425	420000.0	50000000	3360	226
"nvnv wgzohrnvwj otqcxwto"	"backend engineer"	3	1.0756e6	400000.0	100000000	70000	217
"nvnv wgzohrnvwj otqcxwto"	"other"	3	895606.620853	366000.0	100000000	100000	211
"xzegojo"	"other"	4	514633.118343	400000.0	3960000	13000	169
...
"nbmgmxrt"	"backend engineer"	13	2000.0	2000.0	2000	2000	1
"wtrtqvgzt sbmy"	"backend engineer"	8	49000.0	49000.0	49000	49000	1
"ntpojontbo"	"backend engineer"	13	1.5e6	1.5e6	1500000	1500000	1
"nvnv wrxi"	"ios engineer"	8	1.1e6	1.1e6	1100000	1100000	1
"ugrjwgb"	"backend architect"	18	4e6	4e6	4000000	4000000	1

Out[48]: shape: (54_625, 7)

company_hash	job_position	mean_ctc	median_ctc	max_ctc	min_ctc	count
str	str	f64	f64	i64	i64	u32
"nvnv wgzohrnvwj otqcxwto"	"backend engineer"	1.3063e6	480000.0	200000000	600	1107
"vbvkcz"	"backend engineer"	2.9504e6	2.4e6	100000000	10000	944
"nvnv wgzohrnvwj otqcxwto"	"other"	2.4063e6	400000.0	200000000	3500	855
"gqvwr"	"backend engineer"	1.7073e6	1.3e6	70000000	7500	746
"xzegojo"	"other"	1.8482e6	430000.0	200000000	4000	704
...
"sqvbtq xn otqcxwto uqxcvnt ..."	"data scientist"	900000.0	900000.0	900000	900000	1
"qvaxwvrgf"	"engineering leadership"	2e6	2e6	2000000	2000000	1
"wtqbvnx"	"engineering leadership"	120000.0	120000.0	120000	120000	1
"mgppbjoygf"	"engineering leadership"	1.65e6	1.65e6	1650000	1650000	1
"nqvctrtd"	"backend engineer"	1.22e6	1.22e6	1220000	1220000	1

Out[48]: shape: (33_493, 6)

company_hash	mean_ctc	median_ctc	max_ctc	min_ctc	count
str	f64	f64	i64	i64	u32
"nvnv wgzohrnvwj otqcxwto"	1.9812e6	459999.0	200000000	600	4314
"xzegojo"	1.4058e6	500000.0	200000000	3250	2859
"vbvkcz"	3.4831e6	2e6	200000000	2000	2093
"zgn vuurxwvmrt vwwghzn"	3.2739e6	600000.0	199800000	1000	1867
"wgszxkvzn"	1.9492e6	600000.0	200000000	3800	1836
...
"mtnogr ogenfvqt xzaxv ucn rna"	600000.0	600000.0	600000	600000	1
"ogenvst xzegqbvnxg ntwygrgsj..."	780000.0	780000.0	780000	780000	1
"v1 etzwt uqgahwn wgbuvzj ucnrn..."	229999.0	229999.0	229999	229999	1
"xowygrvq tahwvnxg otqcxwto uq..."	1.2e6	1.2e6	1200000	1200000	1
"wyvbxgzo sqghu"	530000.0	530000.0	530000	530000	1

Joining the groups with the main dataset

```
In [49]: df = db.sql("""
    select
        df.*,
        case
            when g1.mean_ctc < df.ctc then 1
            when g1.mean_ctc > df.ctc then 3
            else 2
        end as class_flag
    from df
        join g1 on df.company_hash = g1.company_hash
        and df.job_position = g1.job_position
        and df.yoe = g1.yoe
    ....).pl()

df = db.sql("""
    select
        df.*,
        case
            when g2.mean_ctc < df.ctc then 1
            when g2.mean_ctc > df.ctc then 3
            else 2
        end as designation_flag
    from df
        join g2 on df.company_hash = g2.company_hash
        and df.job_position = g2.job_position
    ....).pl()

df = db.sql("""
    select
        df.*,
        case
            when g3.mean_ctc < df.ctc then 1
            when g3.mean_ctc > df.ctc then 3
            else 2
        end as tier_flag
    from df
        join g3 on df.company_hash = g3.company_hash
    ....).pl()
```

Extracting unique words from job position

```
In [50]: words = df["job_position"].str.split(" ").explode()
WORDS = Counter(words.to_list())
```

Generating new columns based on job position

```
In [52]: df = df.with_columns(
    is_other=pl.col("job_position").str.contains(r"other|unknown"),
    is_developer=pl.col("job_position").str.contains(
        r"developer|software|engineer|programmer|dev|development|backend|frontend|fullstack|web|mobile|app|ios|android|qa|automation|testing|sde|architect|database|cloud|security|devops|a
    is_tech=pl.col("job_position").str.contains(
        r"engineer|developer|cto|application|aiml|fullstack|scientist|designer|frontend|data|engineering|system|data|qa|principal|security|android|sdet|devops|ios|architect|database|softw
    is_management=pl.col("job_position").str.contains(
        r"manager|director|head|vp|president|lead|executive|founder|consulting|leadership|product|administrator|co|project|consultant|team",
    is_sales=pl.col("job_position").str.contains(
        r"sales|business|marketing|account|customer|service|support|growth|strategy|operations|commercial|client|relation|partner|engagement|solution|consultant|executive|representative|m
)
df = df.with_columns(
    is_non_coder=pl.col("job_position").str.contains(
        r"non coder") | pl.col("is_tech") == False
)
```

```
In [53]: df["ctc"].quantile(0.95), df["ctc"].quantile(0.75), df["ctc"].quantile(0.50), df["ctc"].quantile(0.25), df["ctc"].quantile(0.10)
```

```
Out[53]: (4000000.0, 1700000.0, 1000000.0, 560000.0, 340000.0)
```

Binning the CTC

```
In [54]: df=df.with_columns(
    ctc_bin=pl.when(pl.col("ctc") > df["ctc"].quantile(0.95)).then(pl.lit("very_high"))
    .when(pl.col("ctc") > df["ctc"].quantile(0.75)).then(pl.lit("high"))
    .when(pl.col("ctc") > df["ctc"].quantile(0.50)).then(pl.lit("medium"))
    .when(pl.col("ctc") > df["ctc"].quantile(0.25)).then(pl.lit("low"))
    .otherwise(pl.lit("very_low"))
)
```

```
In [55]: df
```

company_hash	job_position	email_hash	orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	class_flag	designation_flag	tier_flag	is_other	is_developer	is_tech
str	str	str	i32	i64	i32	i32	i64	f64	f64	i32	i32	i32	bool	bool	
"vzgnytq wgbuvzj"	"data analyst"	"d6f5821ad2117ac11a4edc969289f4..."	2014	950000	2021	8	1	0.0	9.895833	2	2	3	false	false	
"qvkgqnyxzp xzw"	"backend engineer"	"c84a661fa821151b56547aa673fb0f..."	2015	420000	2016	7	1	0.0	5.0	2	3	3	false	true	
"pzgxnn"	"data scientist"	"a629e1802d753d774cb8ab03a3982e..."	2017	1800000	2019	5	1	0.0	30.0	2	2	3	false	false	
"vrsgfgqpo"	"fullstack engineer"	"b40fb8e24fc9370f64c3db55d411b0..."	2017	700000	2021	5	1	0.0	11.666667	3	1	1	false	true	
"bxwqgogen"	"backend engineer"	"9ae13f1f76dc03d915af49199a8823..."	2013	1000000	2016	9	1	0.0	9.259259	3	3	3	false	true	
...	
"mtonxztn xzavv ucen rna"	"sdet"	"a00f6c259d54e8ad9a72788a3aeb40..."	2014	600000	2019	8	1	0.0	6.25	2	2	2	false	true	
"cxov"	"support engineer"	"a0a88e2d6604342419f3dccc8ef163..."	2016	220000	2019	6	1	0.0	3.055556	3	3	3	false	true	
"ahzzyhbmj"	"other"	"a1f54c0a32322b9aa3ebeafa458b69..."	2005	2500000	2021	17	1	0.0	12.254902	2	1	1	true	false	
"ovbohzs qtotvqwy xzonxnhnt zgx..."	"android engineer"	"b4f61fcf39b7306c68344a3128a802..."	2019	1050000	2020	3	1	0.0	29.166667	3	3	3	false	true	
"ovbohzs qtotvqwy xzonxnhnt zgx..."	"android engineer"	"b6c5afc012cce9e3c6282a0cc0f33c..."	2019	1200000	2021	3	1	0.0	33.333333	1	3	3	false	true	

```
In [56]: df.write_parquet("../data/processed/data.parquet")
```

EDA

```
In [119]: df = pl.read_parquet("../data/processed/data.parquet")
```

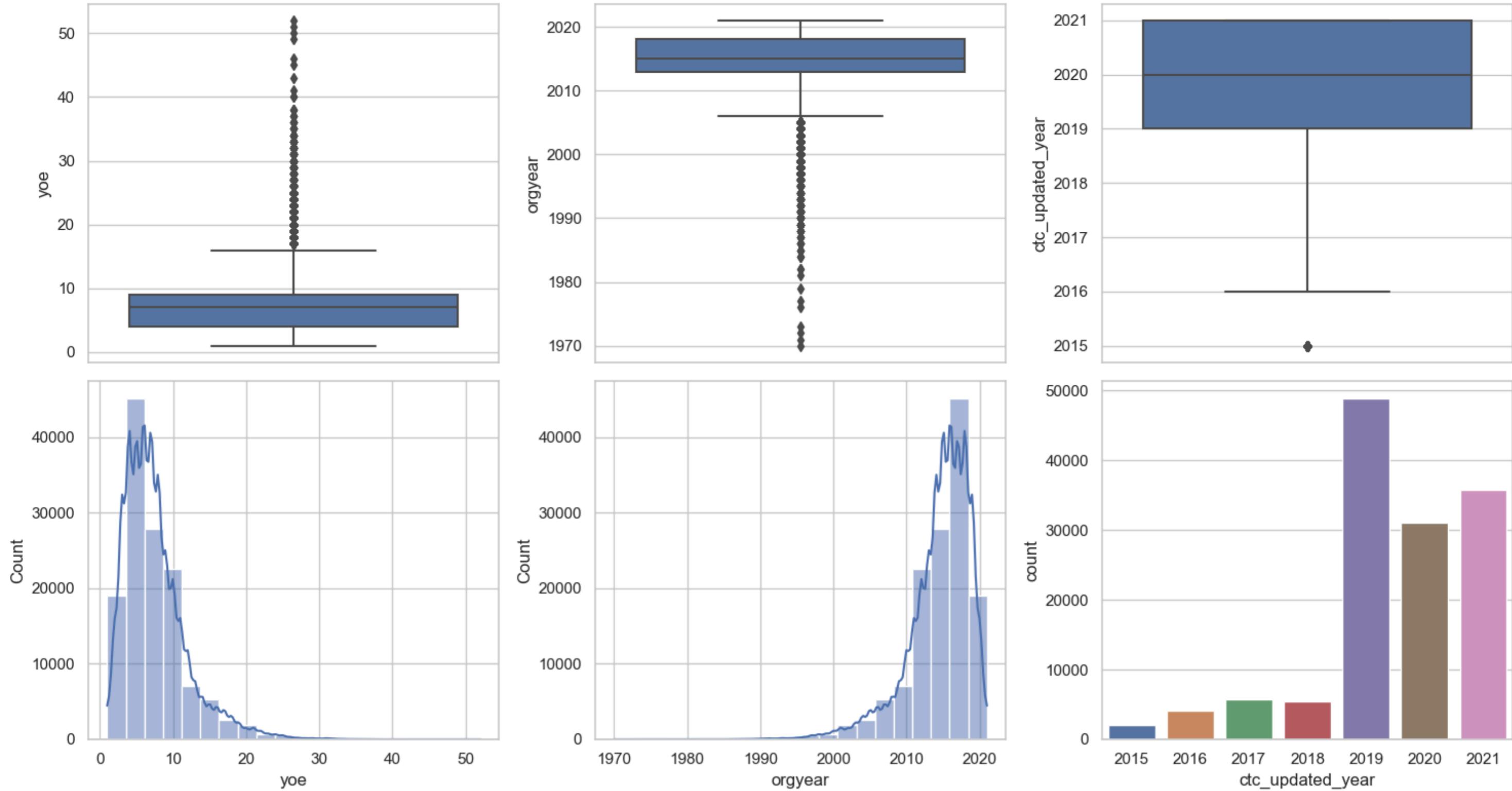
```
In [57]: fig, ax = plt.subplots(2,3, figsize=(15, 8))
sns.boxplot(y=df["yoe"], ax=ax[0,0]);
sns.boxplot(y=df["orgyear"], ax=ax[0,1]);
sns.boxplot(y=df["ctc_updated_year"], ax=ax[0,2]);
```

```

sns.histplot(data=df.to_pandas(), x="yoe", kde=True, bins=20, ax=ax[1,0]);
sns.histplot(data=df.to_pandas(), x="orgyear", kde=True, bins=20, ax=ax[1,1]);
sns.countplot(data=df.to_pandas(), x="ctc_updated_year", ax=ax[1,2]);

# plt.suptitle("Boxplot of Years of Experience, Organization Year and CTC Updated Year");
plt.tight_layout();

```



Observations

- Above plots show distribution of CTC updated year, Years of Experience and joining year
- Most of the candidates have under 10 years of experience.
- Majority of the CTC has been updated from 2019 to 2021.
- Most of the candidates joined after 2010

```
In [145...]: db.sql("""
    select distinct job_position, ctc from df where (job_position like '%intern%' or job_position like '%trainee%') order by ctc desc limit 10
""").pl()
```

Out[145...]: shape: (10, 2)

job_position	ctc
str	i64
"engineering intern"	200000000
"engineering intern"	155000000
"engineering intern"	100310000
"engineering intern"	100000000
"engineering intern"	80000000
"engineering intern"	20000000
"engineering intern"	19700000
"engineering intern"	14100000
"engineering intern"	12000000
"engineering intern"	11500000

Observations

- From above table we can see that some job position like intern have extremely high CTC which is not possible.

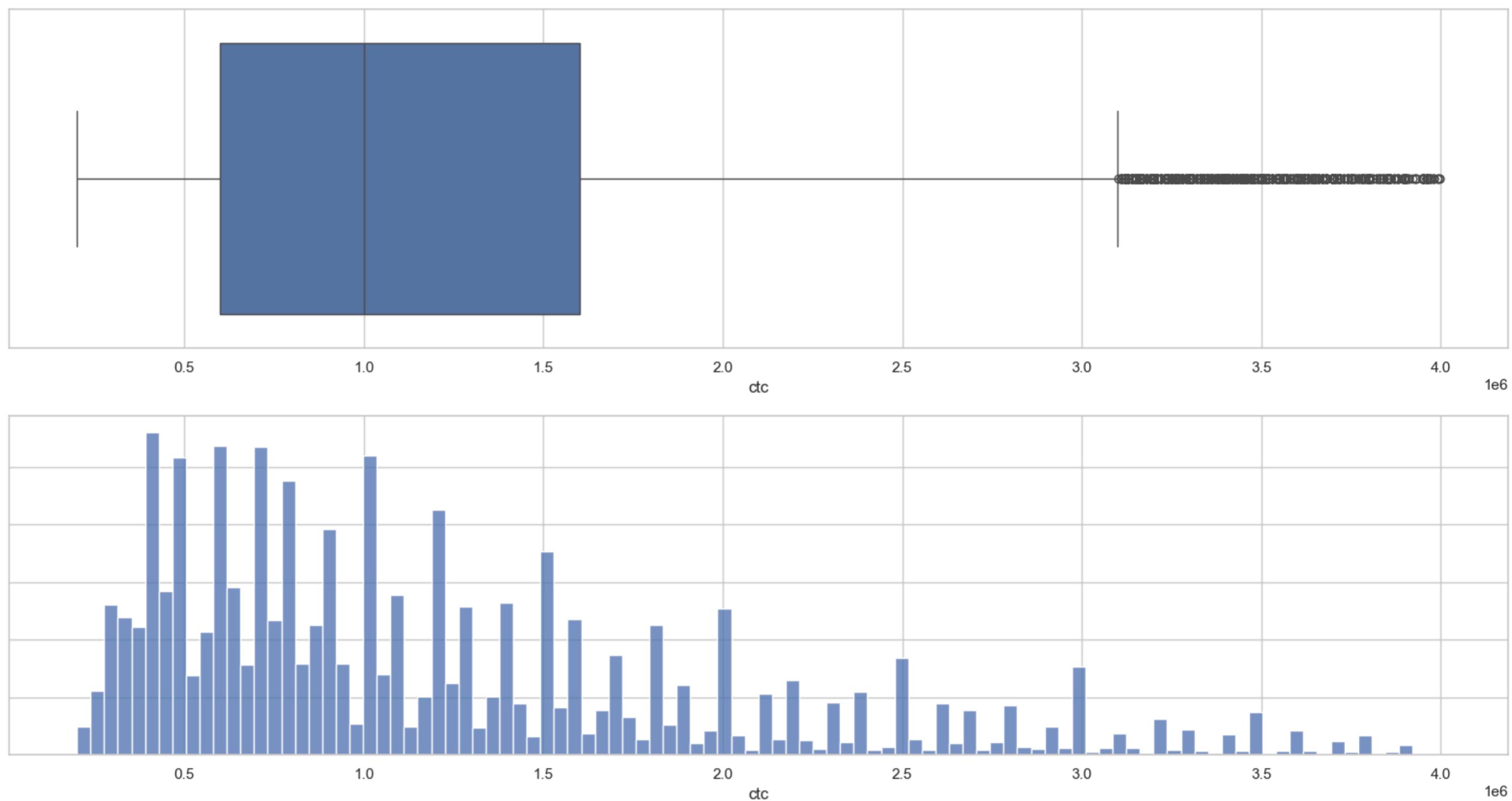
```
In [120...]: df["ctc"].quantile(0.95), df["ctc"].quantile(0.05)
```

Out[120...]: (4000000.0, 200000.0)

```
In [146...]: ctc_filtered_df = df.filter((df["ctc"] < df["ctc"].quantile(0.95)) & (df["ctc"] > df["ctc"].quantile(0.05)))
```

```
In [147...]: fig, ax = plt.subplots(2,1, figsize=(20, 10))
sns.boxplot(x=ctc_filtered_df["ctc"], ax=ax[0]);
sns.histplot(x=ctc_filtered_df["ctc"], ax=ax[1], bins=100);
plt.suptitle("Boxplot and Histogram of CTC");
```

Boxplot and Histogram of CTC



Observations

- From above table we can see that the most of the CTCs are between 6 to 16 lakhs.

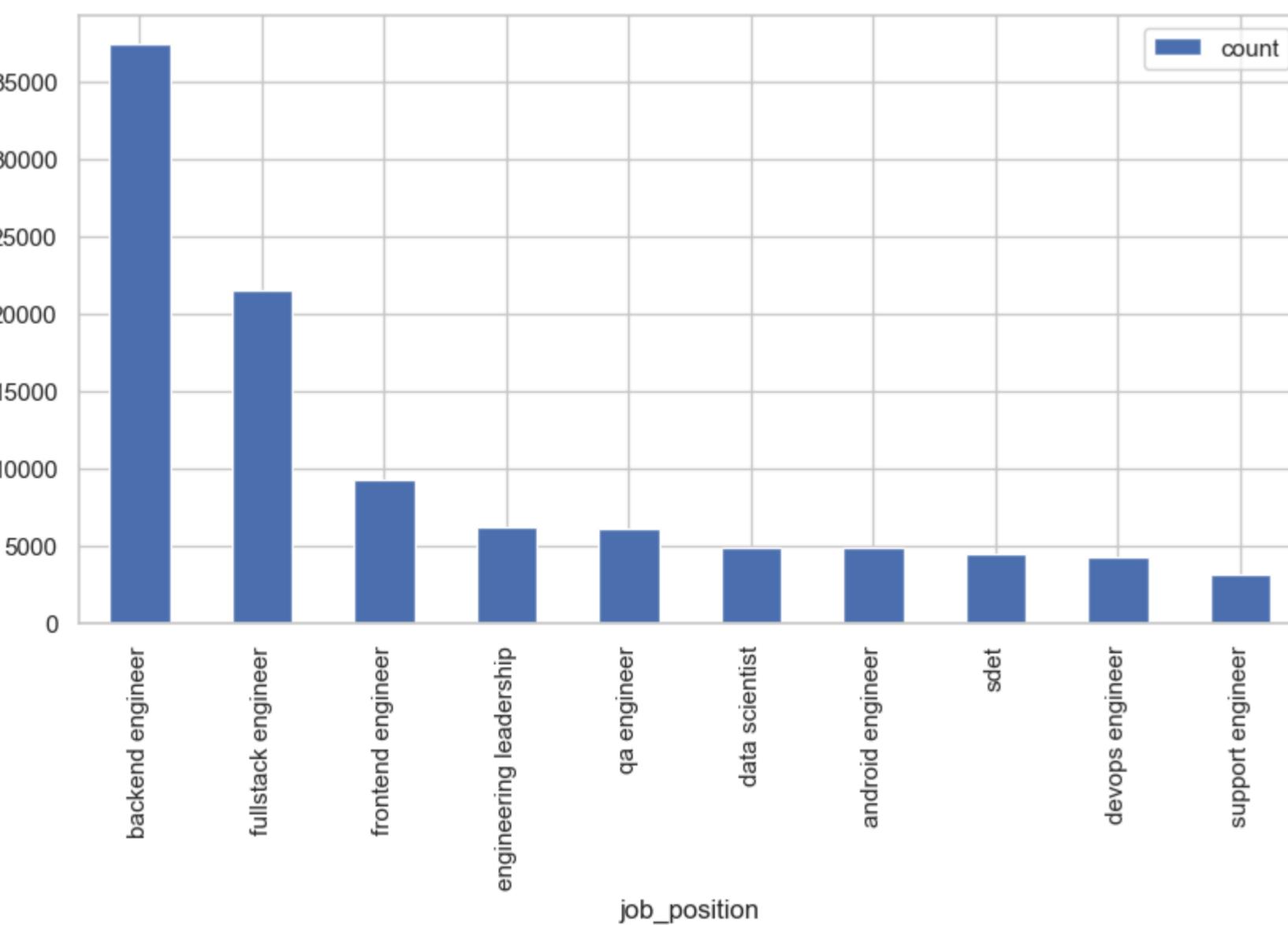
Mean CTC

```
In [148]: ctc_filtered_df["ctc"].mean(), ctc_filtered_df["ctc"].median()
Out[148]: (1220181.536710552, 1000000.0)
```

Observations

The mean CTC is around 12 lakhs and median is around 10 lakhs.

```
In [87]: df.filter(df["job_position"]!="other")["job_position"].value_counts(sort=True).head(10).to_pandas().set_index("job_position").plot(kind="bar", figsize=(10, 5));
```

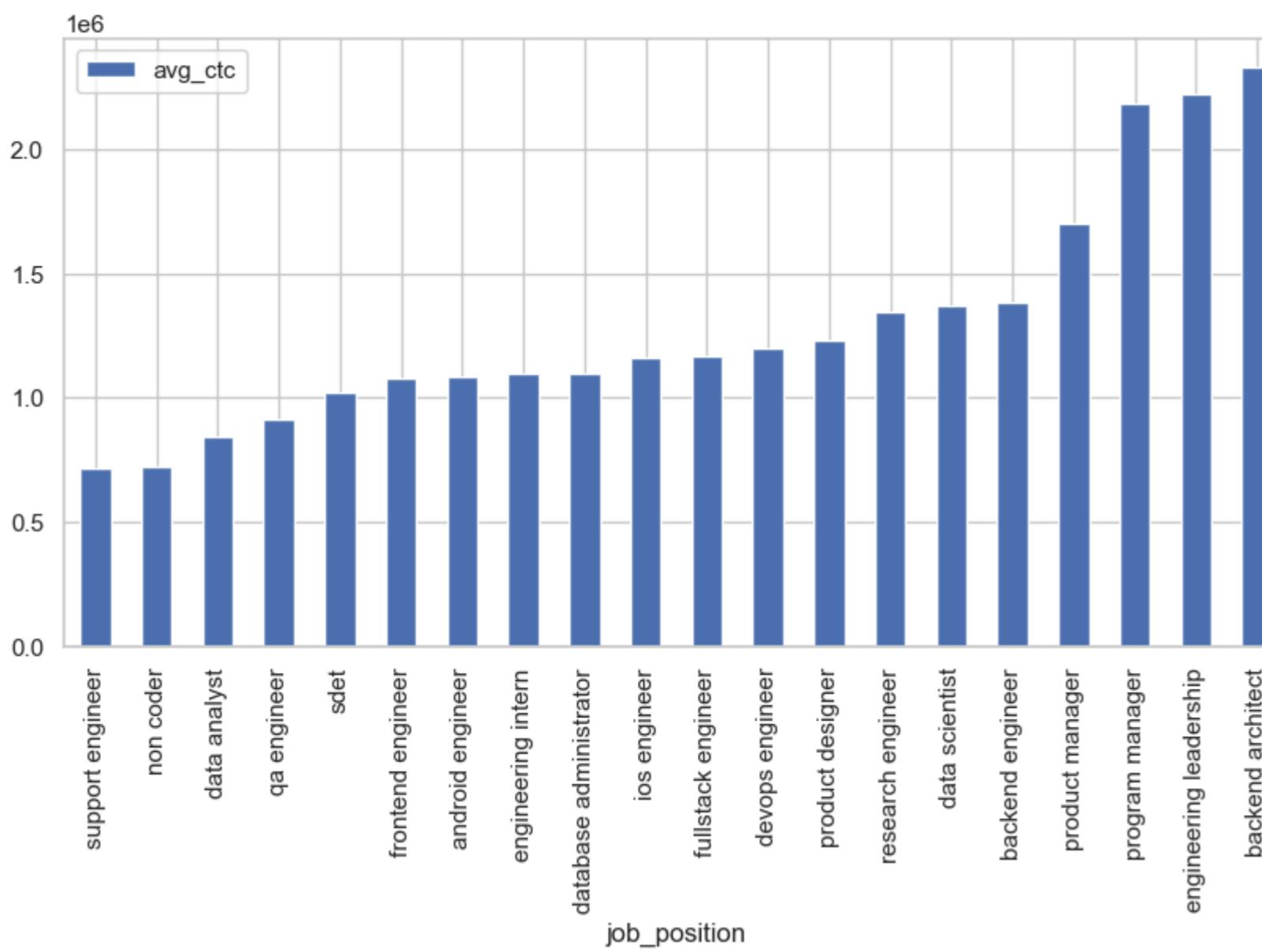


Observations

- From above plot we can see that most the candidates belong to web developement domain

Average CTC of top 20 job positions

```
In [68]: db.sql("""
    select job_position, avg(ctc) as avg_ctc from ctc_filtered_df where job_position <> 'other' group by job_position order by count(*) desc limit 20
""").to_df().sort_values(by="avg_ctc").set_index("job_position").plot(kind="bar", figsize=(10, 5));
```



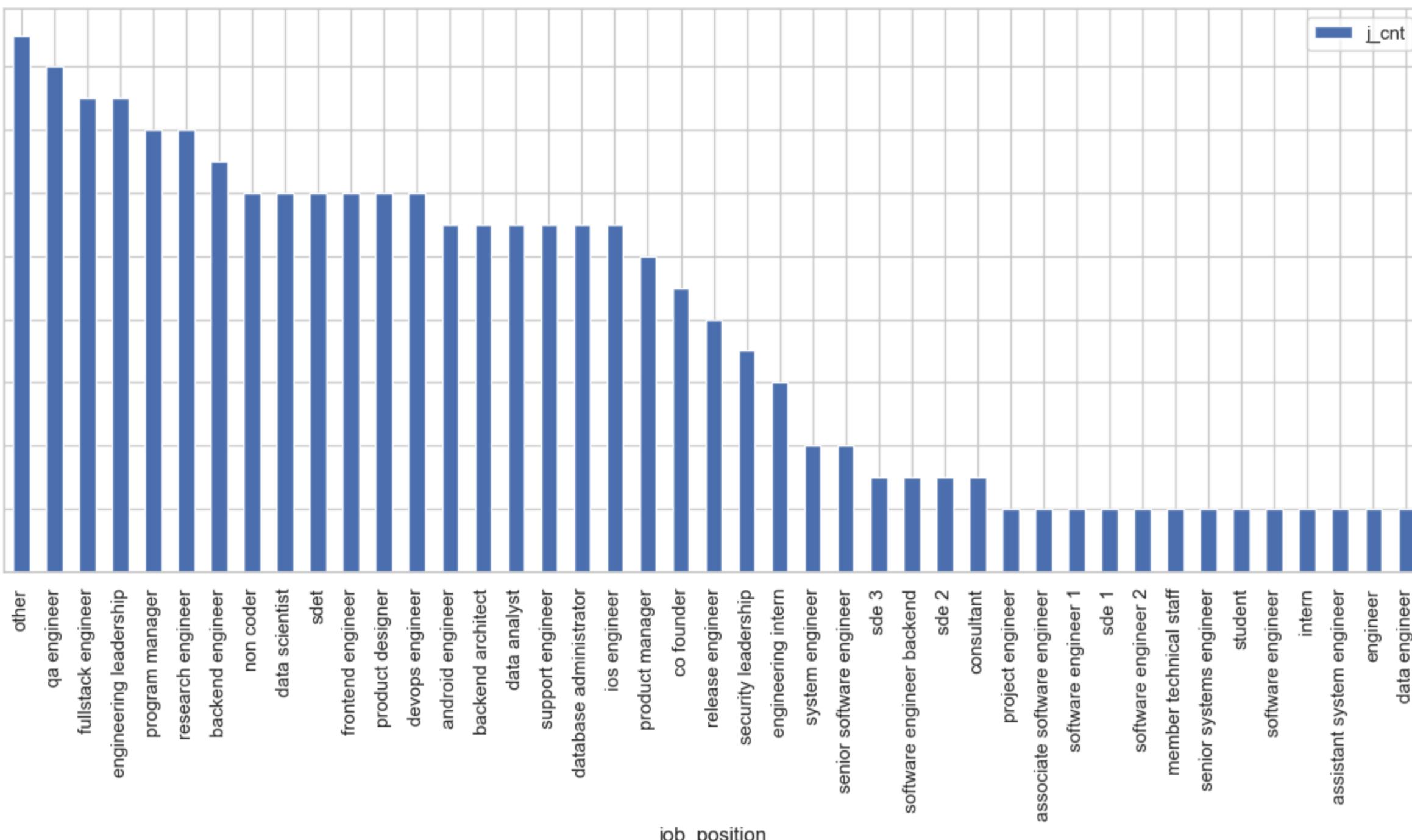
Observations

- Above is the plot of average ctc for top 20 popular job positions.
- We can see that backend architect has the highest average ctc.

Check which job position has decreasing CTC with increase in experience

```
In [29]: neg_salary = db.sql("""
    WITH salary_diff AS (
        SELECT *,
            LAST(avg_ctc) OVER (PARTITION BY job_position ORDER BY yoe) -
            AVG(avg_ctc) OVER (PARTITION BY job_position) AS diff
        FROM (
            SELECT job_position, yoe, ROUND(AVG(ctc)) AS avg_ctc, COUNT(*) AS cnt
            FROM ctc_filtered_df
            GROUP BY yoe, job_position
            HAVING cnt > 1
        )
    )
    SELECT job_position, COUNT(*) AS j_cnt
    FROM salary_diff
    WHERE diff < 0
    GROUP BY job_position
    HAVING j_cnt > 1
    ORDER BY j_cnt DESC
""").to_df()
```

```
In [31]: neg_salary.set_index("job_position").plot(kind="bar", figsize=(15, 6));
```



Observations

From above analysis we can approximately say that QA engineer has decreasing trend in CTC over the years.

```
In [44]: # db.sql("""
#     with cte as (
#         select job_position, ctc_updated_year, round(avg(ctc)) avg_ctc, yoe from df group by ctc_updated_year, job_position, yoe having count(*) > 1
#     )
#     select distinct job_position, yoe,
#         first(avg_ctc) over (partition by job_position, yoe order by ctc_updated_year) as first_avg_ctc,
#         last(avg_ctc) over (partition by job_position, yoe order by ctc_updated_year) as last_avg_ctc
#
#         from cte where yoe > 1 order by job_position, yoe
# """)
```

Comparison of Average CTC of Data Scientist and Other Job Positions

```
In [176...]: db.sql("""
    WITH ds_ctc AS (
        SELECT
```

```

        company_hash,
        ROUND(AVG(ctc)) AS avg_ds_ctc
    FROM df
    WHERE job_position = 'data scientist'
    GROUP BY company_hash
),
cte2 AS (
    SELECT
        DISTINCT df.company_hash,
        ROUND(AVG(ctc)) AS avg_ctc,
        avg_ds_ctc
    FROM df
    JOIN ds_ctc ON df.company_hash = ds_ctc.company_hash
    WHERE job_position <> 'data scientist'
    GROUP BY df.company_hash, avg_ctc
    ORDER BY df.company_hash
)
SELECT *,  

CASE  

    WHEN avg_ctc > avg_ds_ctc THEN 'Higher'  

    WHEN avg_ctc < avg_ds_ctc THEN 'Lower'  

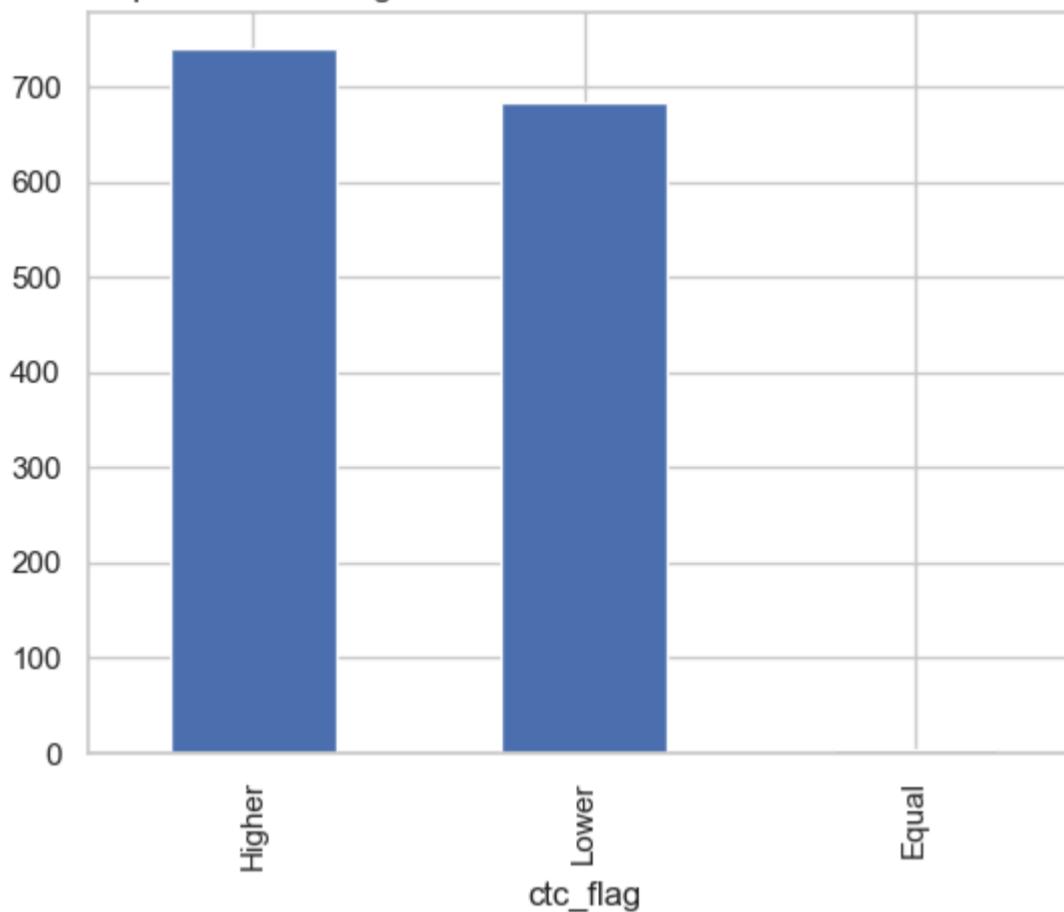
    ELSE 'Equal'  

END AS ctc_flag
FROM cte2
""").to_df().value_counts("ctc_flag").plot(kind="bar");

plt.title("Comparison of Average CTC of Data Scientist and Other Job Positions");

```

Comparison of Average CTC of Data Scientist and Other Job Positions

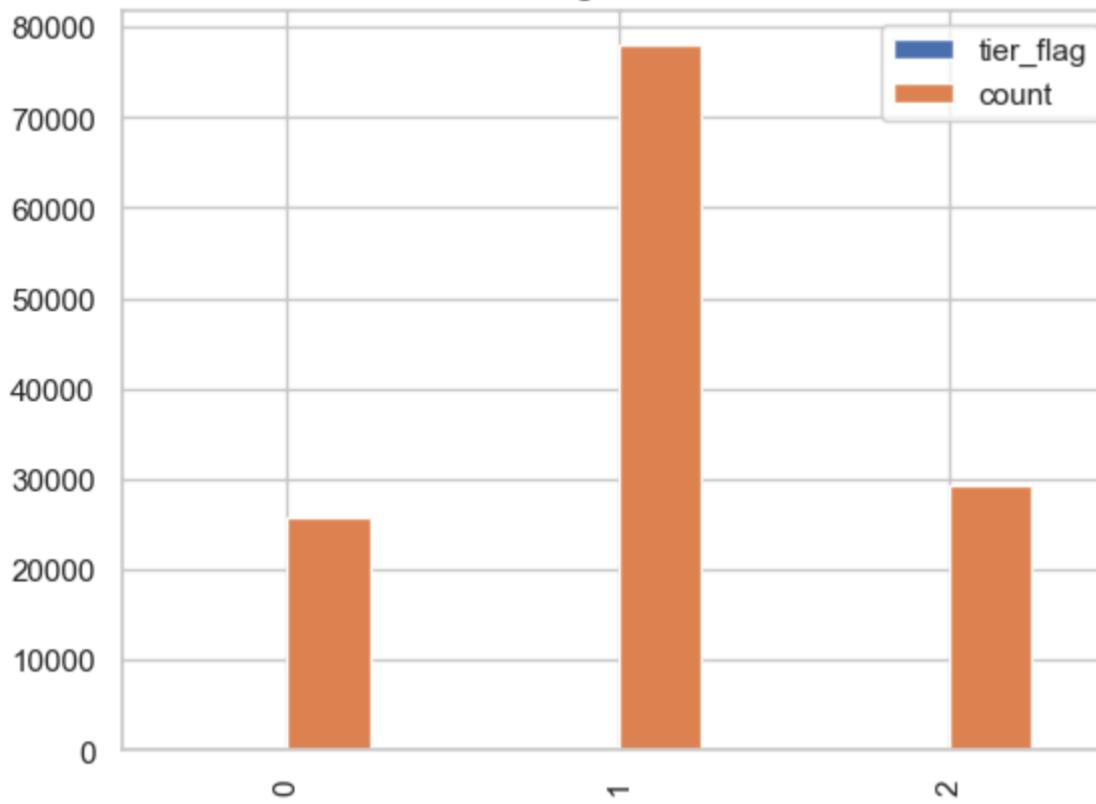


Observations

In majority of the cases Data Scientist has higher average CTC than other job positions. But the gap is not that significant.

```
In [180...]: df["tier_flag"].value_counts().to_pandas().plot(kind="bar");
plt.title("Tier Flag Distribution");
```

Tier Flag Distribution



Observations

- Majority of the candidates belong to tier 1 flag, indicating that these candidates have higher than average CTC.

Top company with highest Tier 1 candidates

```
In [187...]: db.sql("""
select company_hash, count(*) count from df where tier_flag=1 group by company_hash, tier_flag order by count(*) desc limit 10
""").pl()
```

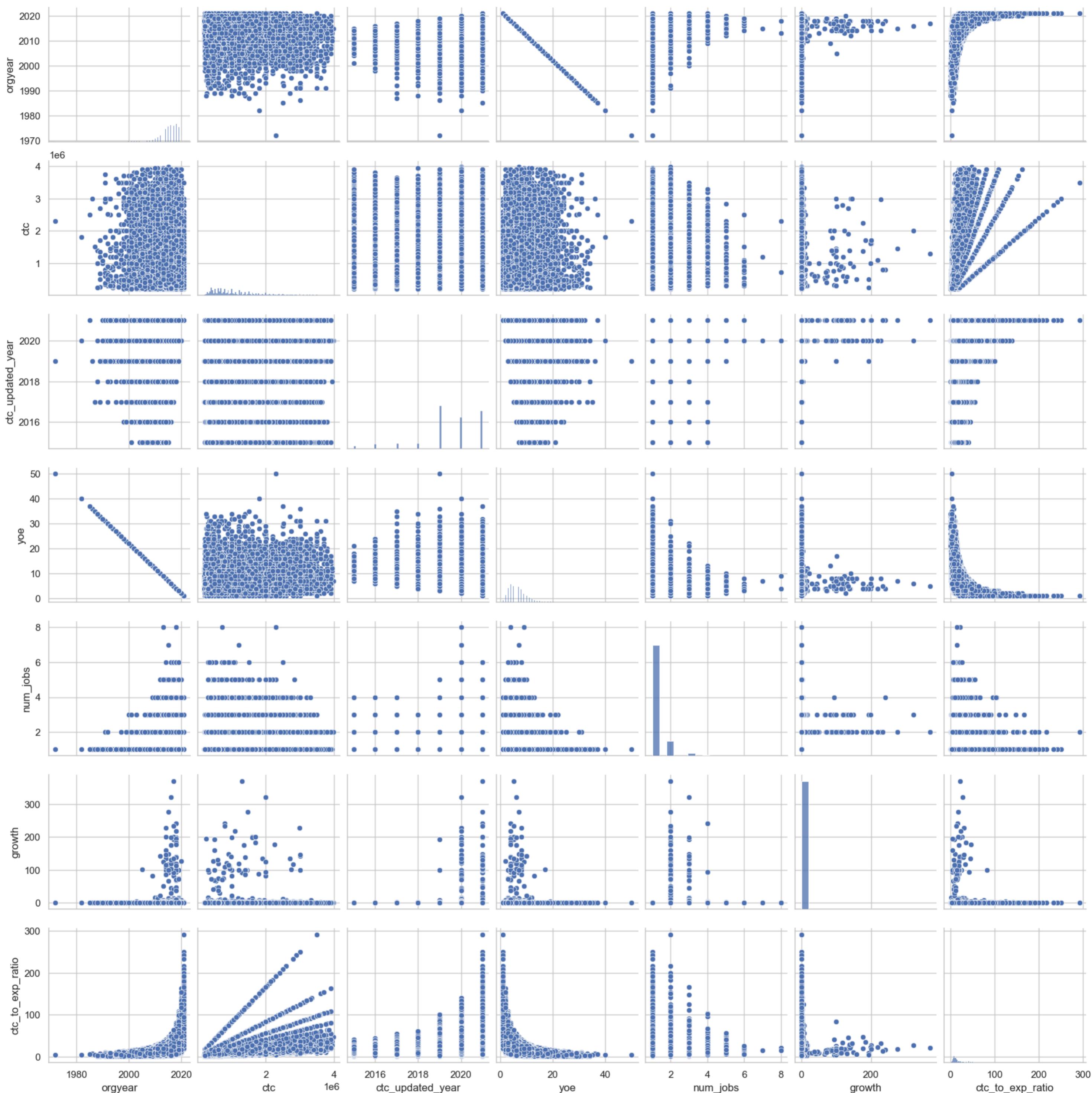
Out[187...]: shape: (10, 2)

company_hash	count
str	i64
"gqvwrt"	532
"vbvkz"	504
"bxwqgogen"	397
"uvjnb"	249
"vagmt"	248
"wxowg"	198
"fvrqvqn rvmo"	192
"ovbohzs qa xzonxnhnt xzaxv mv..."	184
"zvsqqg"	181
"xzegojo"	180

Observations

- Above is the list of top companies with highest tier 1 candidates.

```
In [203...]
cols=[
    'orgyear',
    'ctc',
    'ctc_updated_year',
    'yoe',
    'num_jobs',
    'growth',
    'ctc_to_exp_ratio',
]
sns.pairplot(ctc_filtered_df.filter(ctc_filtered_df["tier_flag"]==3).select(cols).to_pandas());
```

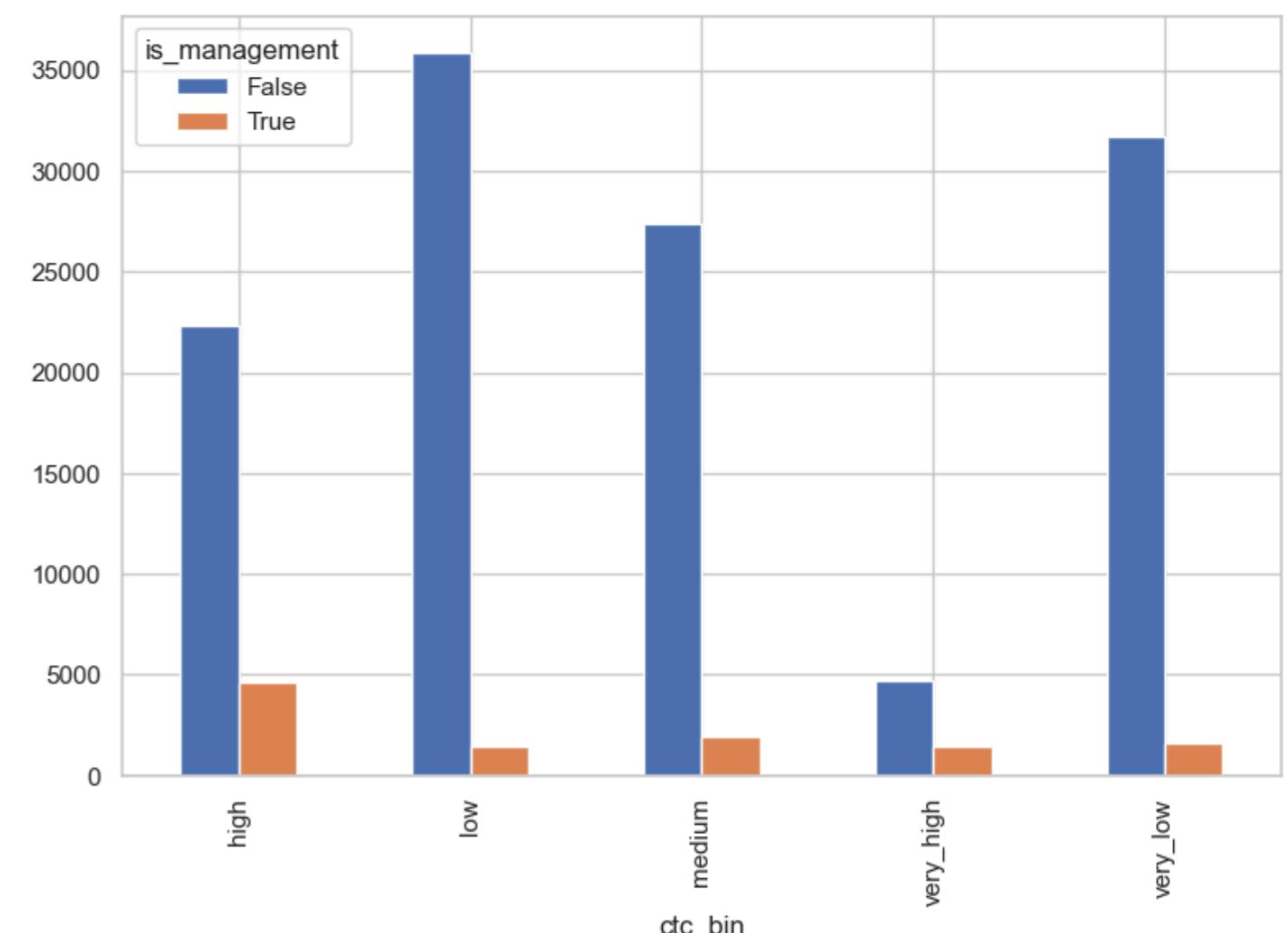
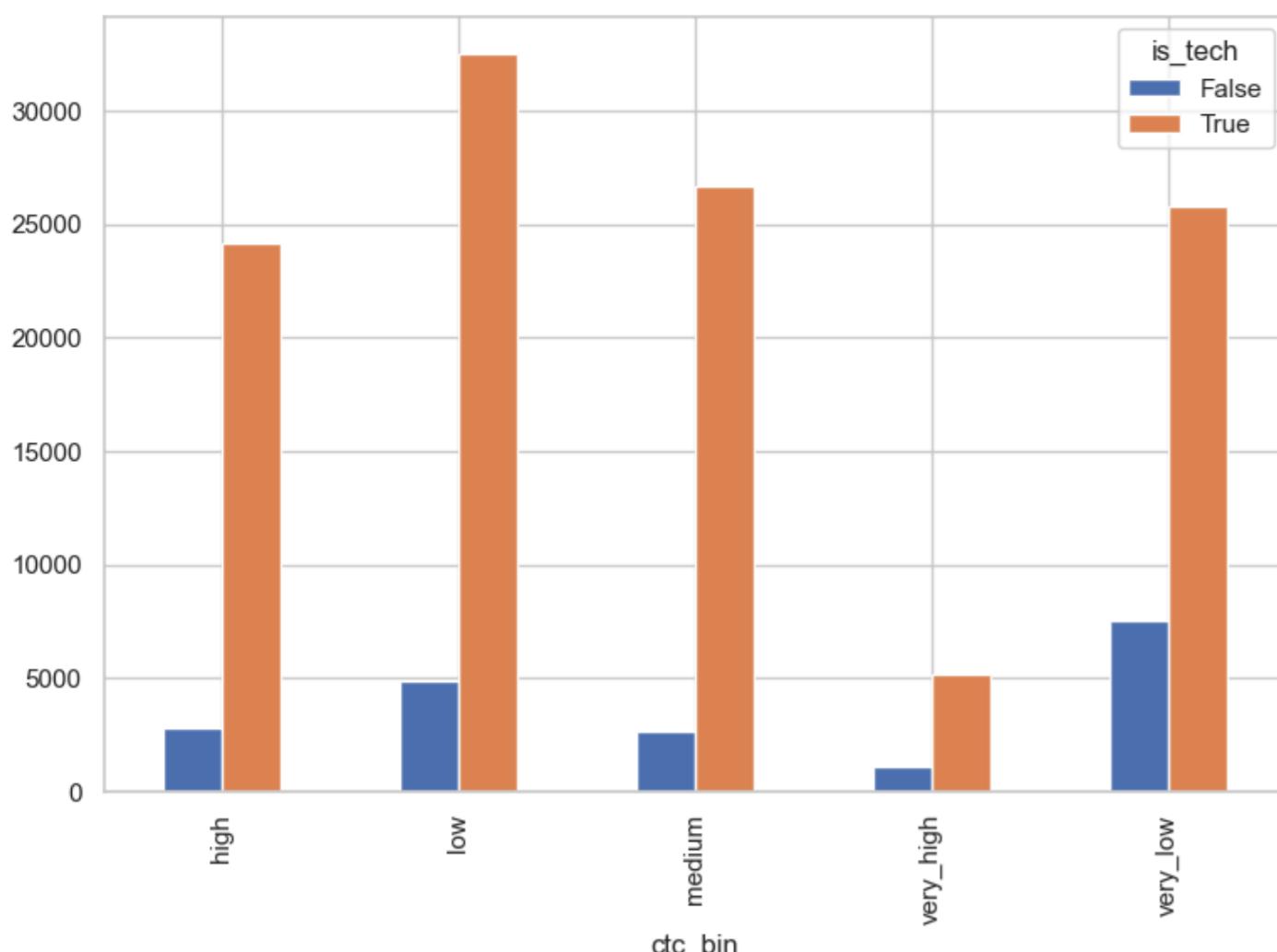


Observations

- We can see that majority of Tier 3 candidates do not switch companies.
- Most of the candidates joined during 2020 but there are employees who have joined before 2000's
- There is huge variance in ctc with respect to experience

```
In [182...]
fig, ax = plt.subplots(1,2, figsize=(20, 6))
db.sql("""
    select ctc_bin, is_tech, count(*) as count from df group by ctc_bin, is_tech
""").to_df().pivot(index="ctc_bin", columns="is_tech", values="count").plot(kind="bar", ax=ax[0]);

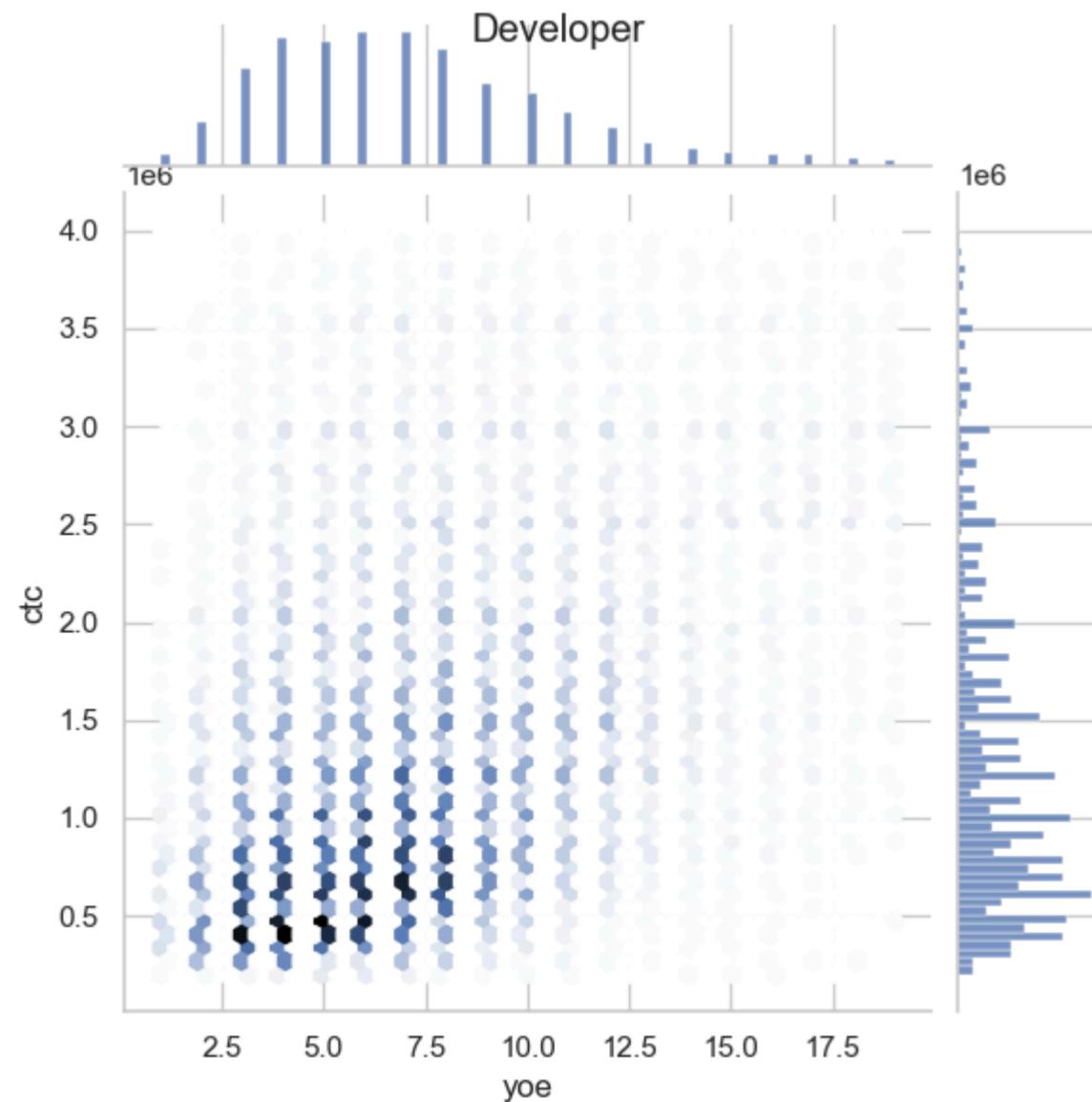
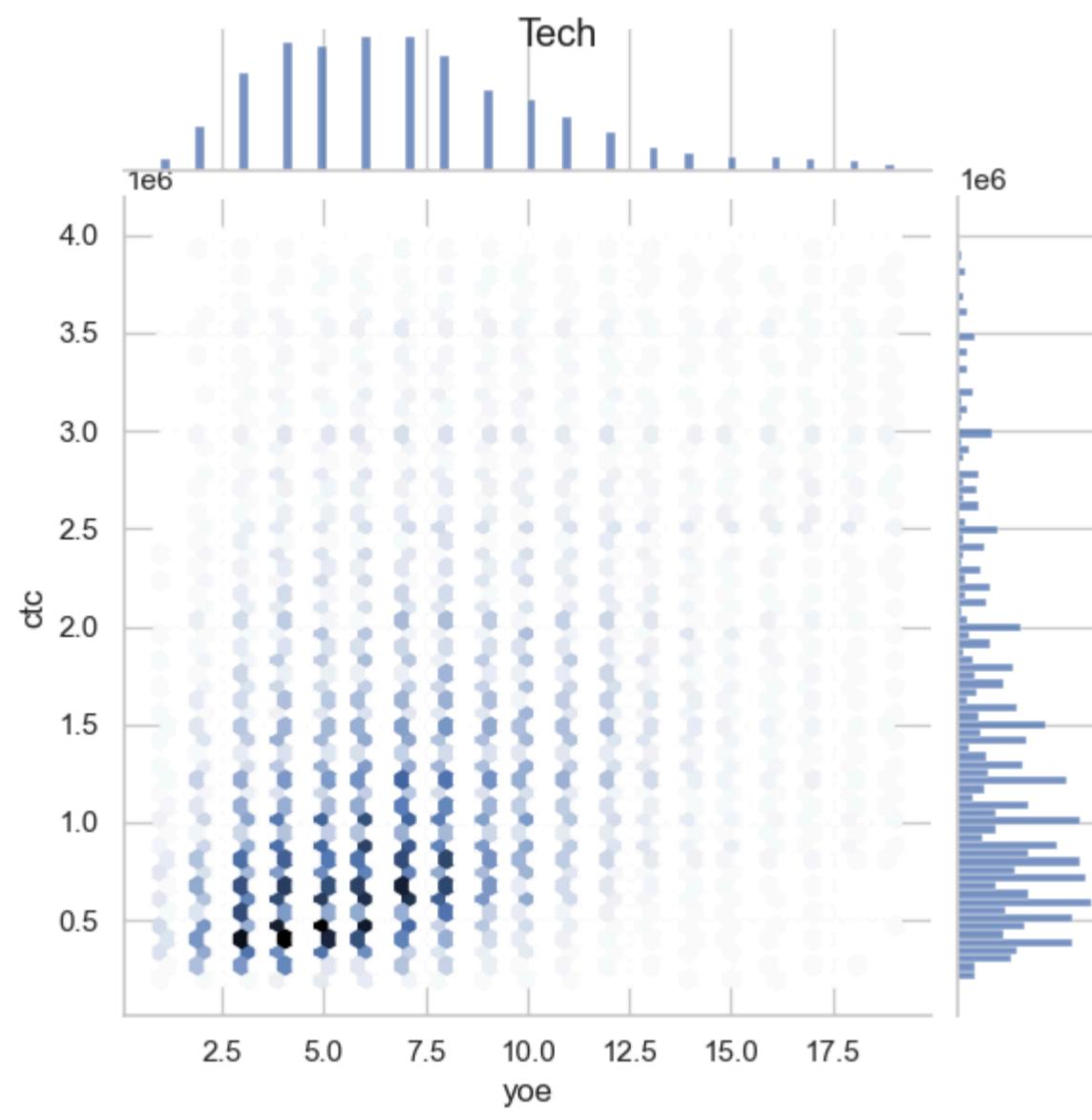
db.sql("""
    select ctc_bin, is_management, count(*) as count from df group by ctc_bin, is_management
""").to_df().pivot(index="ctc_bin", columns="is_management", values="count").plot(kind="bar", ax=ax[1]);
```

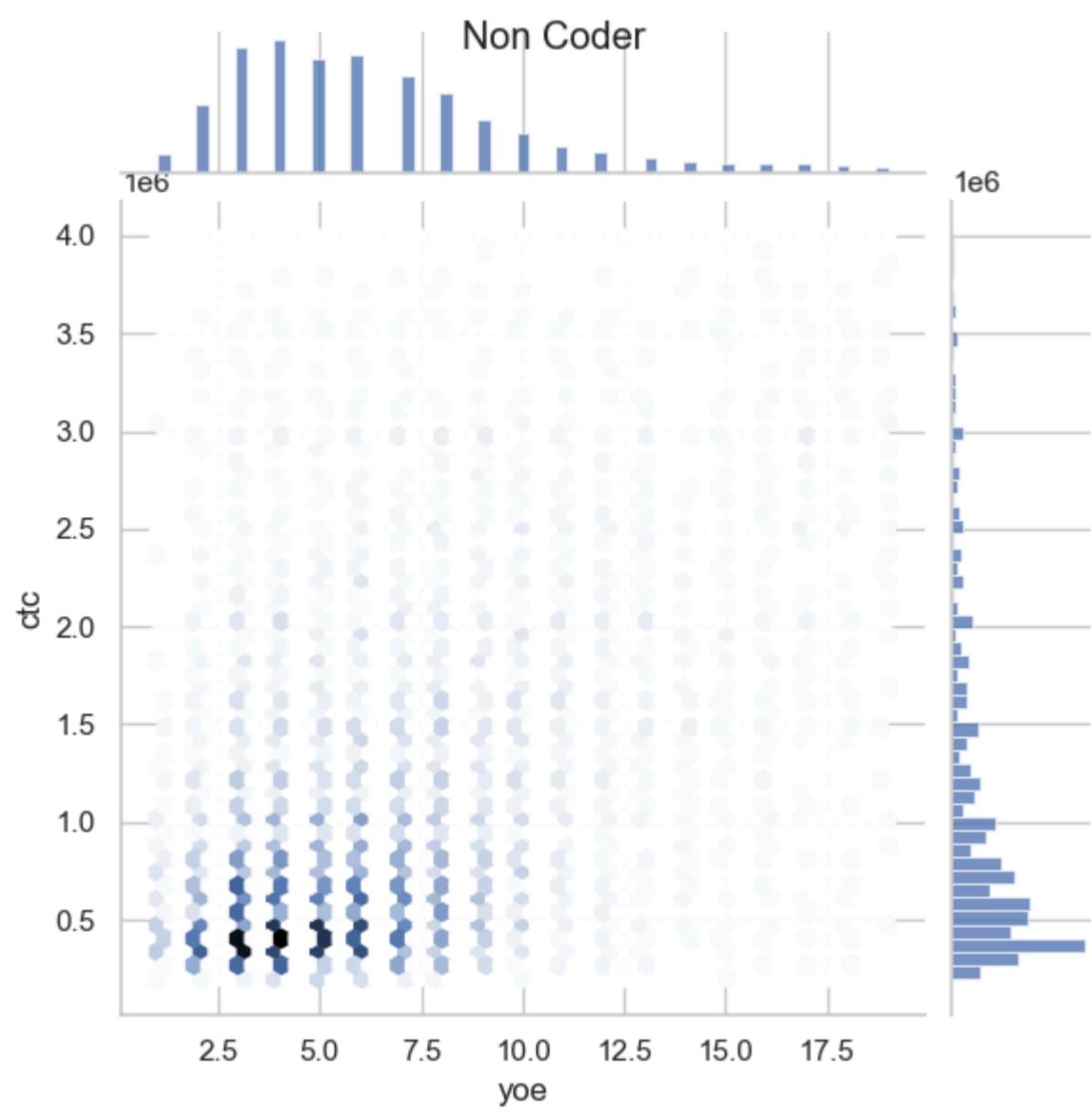
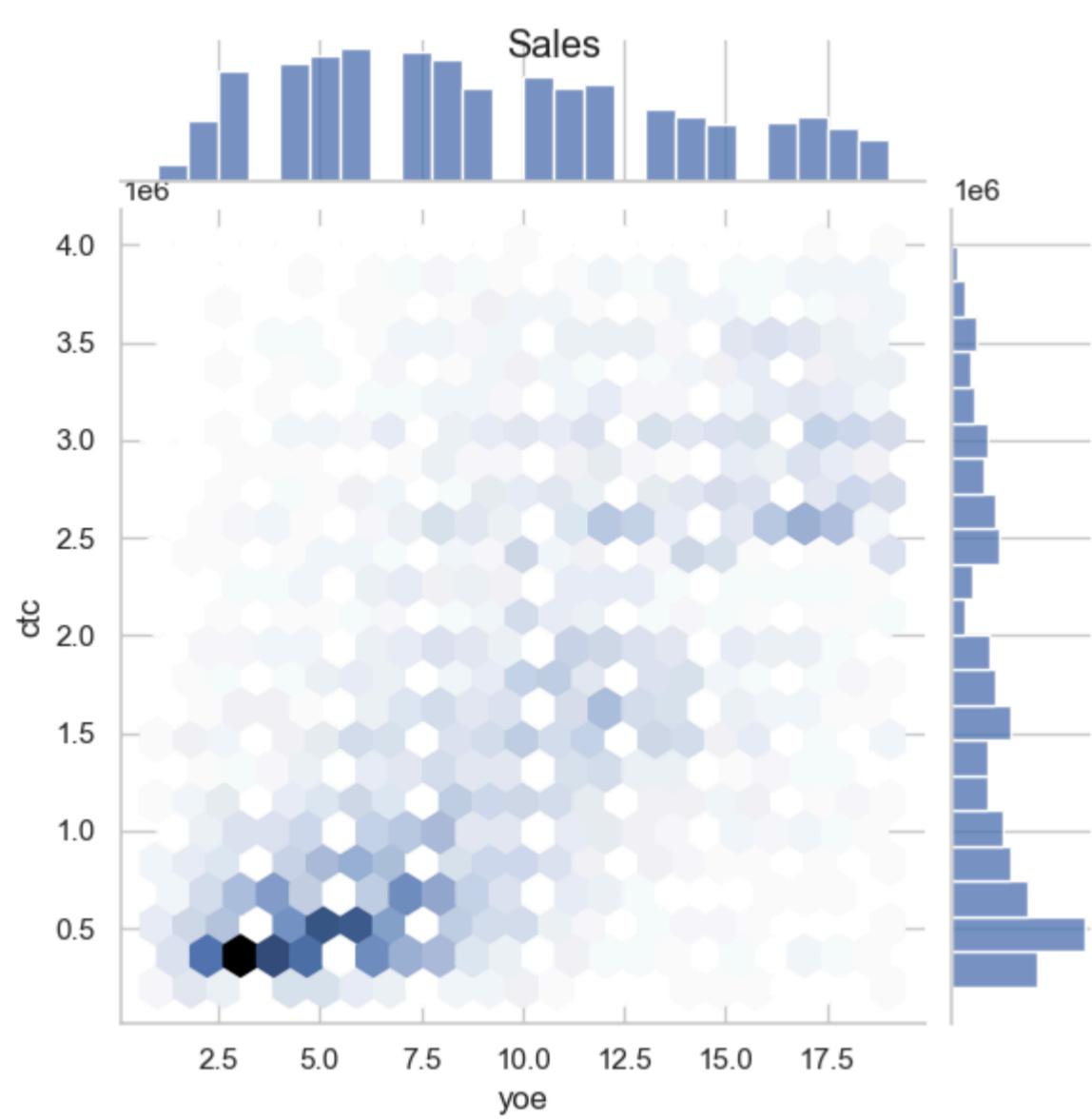
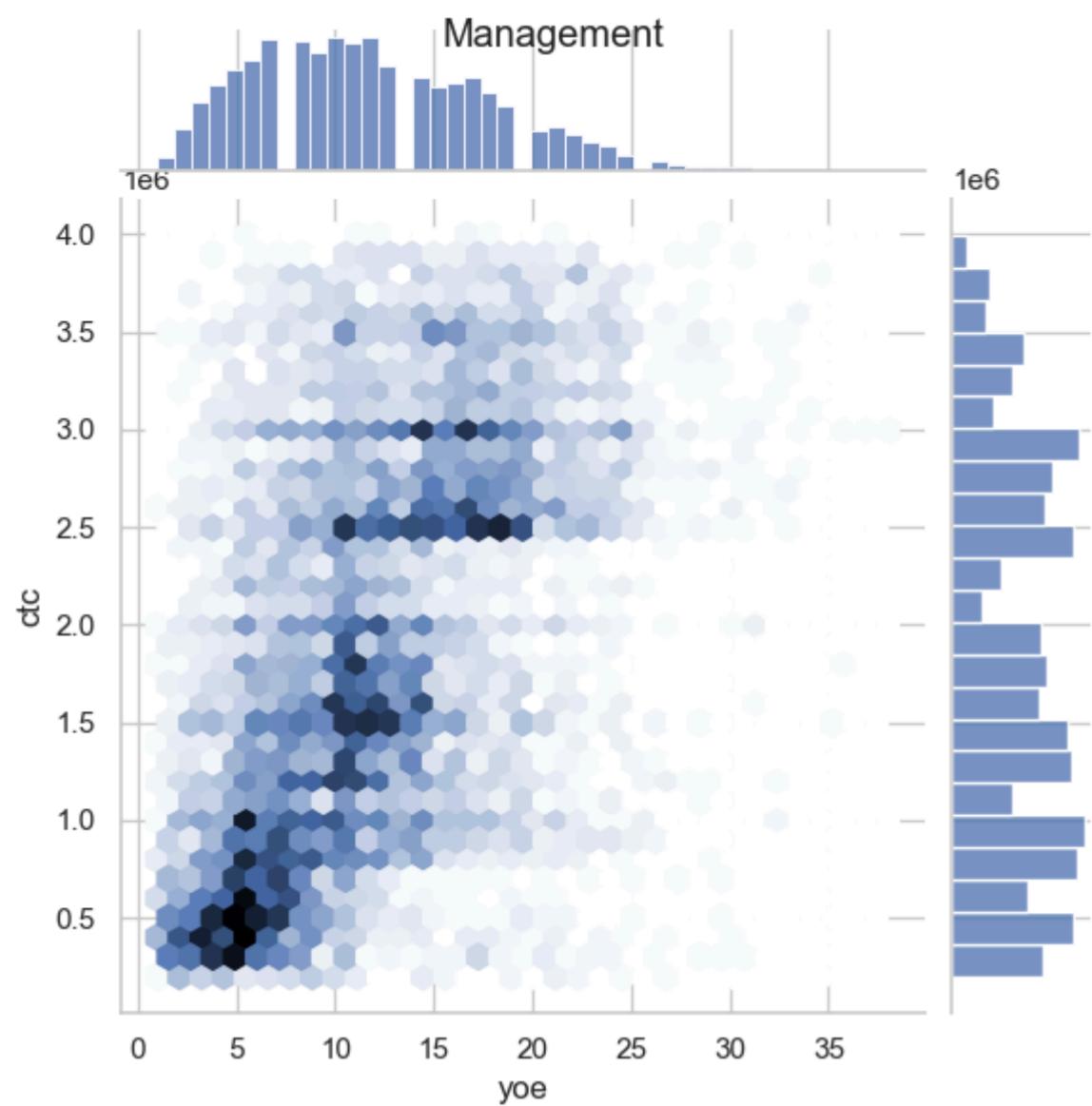


```
In [73]: tech_df = ctc_filtered_df.filter((ctc_filtered_df["is_tech"]) & (ctc_filtered_df["yoe"] < 20))
developer_df = ctc_filtered_df.filter((ctc_filtered_df["is_developer"]) & (ctc_filtered_df["yoe"] < 20))
management_df = ctc_filtered_df.filter((ctc_filtered_df["is_management"]) & (ctc_filtered_df["yoe"] < 40))
sales_df = ctc_filtered_df.filter((ctc_filtered_df["is_sales"]) & (ctc_filtered_df["yoe"] < 20))
non_coder_df = ctc_filtered_df.filter((ctc_filtered_df["is_non_coder"]) & (ctc_filtered_df["yoe"] < 20))
```

```
In [74]: # fig, ax = plt.subplots(2,3, figsize=(20, 10))

sns.jointplot(tech_df["yoe"], tech_df["ctc"], kind="hex");
plt.suptitle("Tech");
sns.jointplot(developer_df["yoe"], developer_df["ctc"], kind="hex");
plt.suptitle("Developer");
sns.jointplot(management_df["yoe"], management_df["ctc"], kind="hex");
plt.suptitle("Management");
sns.jointplot(sales_df["yoe"], sales_df["ctc"], kind="hex");
plt.suptitle("Sales");
sns.jointplot(non_coder_df["yoe"], non_coder_df["ctc"], kind="hex");
plt.suptitle("Non Coder");
```





Observations

- Above plots shows the distribution of CTC for different job positions.
- We can also see that sales ctc decreases after 15 years of experience.
- Majority of the sales have less than 5 lakhs CTC.
- Developers have wide range of salary for different experience levels.

Manual Clustering

```
In [47]: df=pl.read_parquet("../data/processed/data.parquet")
```

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

```
In [49]: df.filter(pl.col("tier_flag") == 1).sort("ctc", descending=True).limit(10)
```

Out[49]: shape: (10, 20)

company_hash		job_position		email_hash		orgyear		ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	class_flag	designation_flag	tier_flag	is_other	is_developer
str	str	str	str	i32	i64	i32	i32	i64	f64	f64	i32	i32	i32	i32	i32	bool	bool	
"qtrxvzwt xzegwgb rbxnta"	"data analyst"	"f1b31a501f6b7fd6edae9e7e883bf6...	2017	200000000	2020	5	1	0.0	3333.333333	2	1	1	false	false				
"otre tburgjta"	"android engineer"	"f63e63a8cc3db37c89a49f0498d073...	2015	200000000	2020	7	1	0.0	2380.952381	1	1	1	false	true				
"nvnv sqghu"	"other"	"3a5fc9df81bcc49b520771ae2fcb9...	2013	200000000	2020	9	1	0.0	1851.851852	2	2	1	true	false				
"xb v onhatzn"	"fullstack engineer"	"431c610cffb5f699476173431bb1f4...	2018	200000000	2019	4	1	0.0	4166.666667	1	1	1	false	true				
"ftrro evqsg"	"support engineer"	"54bafe5fc688d31915438560bd4e94...	2015	200000000	2020	7	1	0.0	2380.952381	1	1	1	false	true				
"vrnqvz"	"other"	"643caeb3a1b6e1cbba8cb026833609...	2014	200000000	2020	8	1	0.0	2083.333333	1	1	1	true	false				
"nvrta"	"support engineer"	"6b3c275643094c663a51e84196fd71...	2016	200000000	2020	6	1	0.0	2777.777778	2	2	1	false	true				
"otre tburgjta"	"other"	"4f83774dfe4b97bfd0836463f3a573...	2020	200000000	2020	2	1	0.0	8333.333333	1	1	1	true	false				
"nvnv wgzohrnvwj otqcxwto"	"support engineer"	"199146d59076620fa919a1b4e51030...	2017	200000000	2020	5	1	0.0	3333.333333	1	1	1	false	true				
"tvngz"	"engineering intern"	"331f1c3d933482a7d0bfe778e8d4e8...	2013	200000000	2020	9	1	0.0	1851.851852	2	2	1	false	true				

Top 10 employees of data science in each company earning more than their peers - Class 1

```
In [51]: db.sql("""  
select * from df where job_position like 'data %' and class_flag = 1 order by ctc desc limit 10  
""").pl()
```

Out[51]: shape: (10, 20)

company_hash		job_position		email_hash		orgyear		ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	class_flag	designation_flag	tier_flag	is_other	is_developer
str	str	str	str	i32	i64	i32	i32	i64	f64	f64	i32	i32	i32	i32	bool	bool		
"zgzt"	"data scientist"	"268a5aa92f0b6d0c675fc9cc1e300e...	2019	200000000	2020	3	2	0.0	5555.555556	1	1	1	false	false				
"nvnv wgzohrnvwj otqcxwto"	"data analyst"	"655da5cd99f1ba4ad249dade5039b9...	2017	200000000	2020	5	1	0.0	3333.333333	1	1	1	false	false				
"wgzahtzn"	"data analyst"	"9ce2995b2221fe627e861dae9d060...	2016	200000000	2020	6	1	0.0	2777.777778	1	1	1	false	false				
"nvnv wgzohrnvwj otqcxwto"	"data analyst"	"59316048d113539202325e05af9b66...	2015	200000000	2020	7	1	0.0	2380.952381	1	1	1	false	false				
"vwwtzhqt"	"data analyst"	"2f9a4241053f76b2f8c50ea593a905...	2016	200000000	2019	6	1	0.0	2777.777778	1	1	1	false	false				
"wgszxkvzn"	"data analyst"	"aad581a532f319c76c6e73937572fe...	2014	200000000	2020	8	1	0.0	2083.333333	1	1	1	false	false				
"gnytq"	"data analyst"	"6d4a5d19e889596252b038ee040951...	2017	200000000	2020	5	1	0.0	3333.333333	1	1	1	false	false				
"ntwy bvyxzaq"	"data analyst"	"7a723f5b71698674b79bd2195c3bb5...	2019	200000000	2021	3	1	0.0	5555.555556	1	1	1	false	false				
"fxuqq rxbxnta"	"data analyst"	"89f343bf01094accb8b0b2c799499d...	2017	200000000	2020	5	1	0.0	3333.333333	1	1	1	false	false				
"ytqt ntwyzgrgsxto"	"data analyst"	"98a90272cbba6e6e9ca94981824f34...	2015	200000000	2020	7	1	0.0	2380.952381	1	1	1	false	false				

Bottom 10 employees of data science in each company earning less than their peers - Class 3

```
In [52]: db.sql("""  
select * from df where job_position like 'data %' and class_flag = 3 order by ctc asc limit 10  
""").pl()
```

Out[52]: shape: (10, 20)

company_hash		job_position		email_hash		orgyear		ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	class_flag	designation_flag	tier_flag	is_other	is_developer
str	str	str	str	i32	i64	i32	i32	i64	f64	f64	i32	i32	i32	i32	bool	bool		
"wgszxkvzn"	"data analyst"	"abf69e786daa23f50d3142b653235e...	2014	4300	2019	8	1	0.0	0.044792	3	3	3	false	false				
"nvnv wgzohrnvwj otqcxwto"	"data analyst"	"00cd57017317c2beea73de68b4835d...	2012	7000	2019	10	1	0.0	0.058333	3	3	3	false	false				
"nvnv wgzohrnvwj otqcxwto"	"data scientist"	"3175d03fd4618eb293d6f5a1d13d42...	2020	7500	2020	2	1	0.0	0.3125	3	3	3	false	false				
"vqxosrgmvr"	"data scientist"	"3675f79c7e05de96ccf189c818b84b...	2015	8800	2019	7	1	0.0	0.104762	3	3	3	false	false				
"sggsrt"	"data scientist"	"fb64af615420e06d46a1965f59068b...	2018	10000	2021	4	1	0.0	0.208333	3	3	3	false	false				
"ytfrtnn uvwpvqa tzntquqxt"	"data scientist"	"8274b3188470cd1c4914e7face4901...	2018	10000	2019	4	1	0.0	0.208333	3	3	3	false	false				
"ahzyhbmj"	"data scientist"	"d32c344816921ae7be310ba7630c6e...	2016	31000	2019	6	1	0.0	0.430556	3	3	3	false	false				
"zvz"	"data scientist"	"d277c90e2b97fb51657936b66b68ee...	2015	46500	2016	7	1	0.0	0.553571	3	3	3	false	false				
"tqxwoogz"	"data scientist"	"d8b6c25488ee764c6dbd20c56ad8dc...	2017	49000	2017	5	1	0.0	0.816667	3	3	3	false	false				
"xzntr wgqgqvnxgz"	"data scientist"	"6344f65bdb7dc5da9abb8bdb0f90b9...	2015	60000	2017	7	1	0.0	0.714286	3	3	3	false	false				

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

```
In [53]: df.filter(pl.col("tier_flag") == 3).sort("ctc", descending=False).limit(10)
```

Out[53]: shape: (10, 20)

company_hash job_position		email_hash	orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	class_flag	designation_flag	tier_flag	is_other	is_developer	is_te
str	str	str	i32	i64	i32	i32	i64	f64	f64	i32	i32	i32	bool	bool	bc
"gjg"	"fullstack engineer"	"b995d7a2ae5c6f8497762ce04dc5c0..."	2018	600	2021	4	2	2749.0	0.0125	3	3	3	false	true	tri
"nvnwgzohrnvwjotqcxwto"	"backend engineer"	"80ba0259f9f59034c4927cf3bd38dc..."	2012	600	2017	10	1	0.0	0.005	3	3	3	false	true	tri
"tznqtuqtzhqbtaxv"	"fullstack engineer"	"5281828ed60da057750fca673995fe..."	2001	1000	2018	21	1	0.0	0.003968	2	3	3	false	true	tri
"bgqsvzonvzrtj"	"backend engineer"	"78eaccdfacf1b3d73037a6e86451d..."	2011	1000	2020	11	1	0.0	0.007576	3	3	3	false	true	tri
"evwtmggp"	"fullstack engineer"	"e59b4841a8316f58090839eba1787d..."	2013	1000	2020	9	1	0.0	0.009259	2	3	3	false	true	tri
"sgrabzovwyoo"	"engineering intern"	"aaca628b0b240afc71c34e7d592243..."	2018	1000	2019	4	1	0.0	0.020833	3	3	3	false	true	tri
"onhatzn"	"engineering leadership"	"56fba328d9e0187923d92f680369ae..."	2016	1000	2020	6	1	0.0	0.013889	2	3	3	false	true	tri
"gnytqo"	"other"	"f8574b672112433b60b8ca5178cc9..."	2018	1000	2020	4	1	0.0	0.020833	3	3	3	true	false	fa
"sggsrt"	"co founder"	"5756870d895deca920251df2377dad..."	1973	1000	2020	49	1	0.0	0.001701	2	2	3	false	false	fa
"tuxw"	"fullstack engineer"	"fdd4a762ae373cfa10fa2bf6dc995c..."	2013	1000	2019	9	1	0.0	0.009259	3	3	3	false	true	tri

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

```
In [63]: db.sql("""
    WITH cte AS (
        SELECT *,
            ROW_NUMBER() OVER (PARTITION BY company_hash ORDER BY ctc DESC) AS rn
        FROM df
        WHERE tier_flag = 1 AND designation_flag = 1 AND yoe BETWEEN 5 AND 7
    )
    SELECT *
    FROM cte
    WHERE rn <= 10
    ORDER BY company_hash, job_position, ctc DESC
""").pl()
```

Out[63]: shape: (3_988, 21)

company_hash job_position		email_hash	orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	class_flag	designation_flag	tier_flag	is_other	is_developer	is_te
str	str	str	i32	i64	i32	i32	i64	f64	f64	i32	i32	i32	bool	bool	bc
"1bs"	"backend engineer"	"31db7b806a82aac024462d4c97e70e..."	2016	3750000	2020	6	1	0.0	52.083333	1	1	1	false	true	tri
"1bs"	"backend engineer"	"579d9c719cc05b885297256c122585..."	2015	2930000	2019	7	1	0.0	34.880952	1	1	1	false	true	tri
"1bs"	"backend engineer"	"9c02076a74a2b8a64a6e003fa0a2e4..."	2016	2320000	2020	6	1	0.0	32.222222	3	1	1	false	true	tri
"1bs"	"backend engineer"	"38dfe791fc911da418b67aa989a6aa..."	2015	2000000	2019	7	1	0.0	23.809524	1	1	1	false	true	tri
"1bs"	"backend engineer"	"c97fd1612080086b898e440529c863..."	2015	1800000	2019	7	1	0.0	21.428571	3	1	1	false	true	tri
...
"zxzlvvvqn"	"backend engineer"	"1ec1cf024ba4147f65cd8324ca5f4..."	2017	2260000	2019	5	1	0.0	37.666667	1	1	1	false	true	tri
"zxzlvvvqn"	"backend engineer"	"486122d99d9df25e478a87ff7d21cb..."	2015	2000000	2019	7	1	0.0	23.809524	3	1	1	false	true	tri
"zxzlvvvqn"	"fullstack engineer"	"e2377e7ee0d53d2e3a45b9687fdc9c..."	2017	2300000	2020	5	1	0.0	38.333333	1	1	1	false	true	tri
"zxztrtvuo"	"backend engineer"	"303aab779b6daea19317b430f600b0..."	2015	11950000	2020	7	1	0.0	142.261905	1	1	1	false	true	tri
"zxztrtvuo"	"fullstack engineer"	"3879b9a1e356ed20363ffffd6871207..."	2017	1500000	2019	5	1	0.0	25.0	2	1	1	false	true	tri

Top 10 companies (based on their CTC)

```
In [65]: db.sql("""
    select company_hash, avg(ctc) as avg_ctc from df group by company_hash order by avg_ctc desc limit 10
""").pl()
```

Out[65]: shape: (10, 2)

company_hash	avg_ctc
str	f64
"ztnowqxmto"	2e8
"nco rgxonxwo otqcxwtxbxnta"	2e8
"neny"	2e8
"rvmayx rxetonjrt rxbxnta"	2e8
"bjnqvy tztqsj xzav ucn rna"	2e8
"exatrxo wqquqqvnt ogrhnxgzo"	2e8
"xzntrrxstzwt bvzugftq otqcxwto..."	2e8
"ztnwrgha ojontbo uqxcvnt rxbxn..."	2e8
"pyxcqvl vhngbmxrto"	2e8
"xwhmmt ogrhnxgzo"	2e8

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

```
In [86]: db.sql("""
    WITH cte AS (
        SELECT *,
            ROW_NUMBER() OVER (PARTITION BY company_hash ORDER BY ctc DESC) AS rn
        FROM df
    )
    select company_hash, job_position, ctc from cte where rn <=2 order by company_hash desc, ctc desc
""").pl()
```

```
Out[86]: shape: (41_443, 3)
```

company_hash	job_position	ctc
str	str	i64
"zzb ztdnstz vacxogaj ucn rna"	"fullstack engineer"	600000
"zz"	"other"	1370000
"zyvzwt wgzohrnxs tszxztqo"	"frontend engineer"	940000
"zyuw rxbxnta"	"frontend engineer"	2400000
"zyco xzaxv"	"other"	600000
...
"05mz exzytvry uqxvcnt rxbxnta"	"backend engineer"	1100000
"01 ojztqsj"	"frontend engineer"	830000
"01 ojztqsj"	"android engineer"	270000
"0000"	"other"	300000
"0"	"other"	100000

Plotting Manual Clusters

```
In [46]: df=df.drop(["email_hash","is_tech"])
df = df.with_columns(
    ctc_bin=pl.when(pl.col("ctc_bin") == "very_high").then(5)
    .when(pl.col("ctc_bin") == "high").then(4)
    .when(pl.col("ctc_bin") == "medium").then(3)
    .when(pl.col("ctc_bin") == "low").then(2)
    .otherwise(1)
)
job_frequency_df = df.groupby("job_position").agg(pl.count().alias("job_pos_frequency"))
df = df.join(job_frequency_df, on="job_position", how="left")
df=df.drop(["job_position"])
df=df.rename({"job_pos_frequency": "job_position"})
company_frequency_df = df.groupby("company_hash").agg(pl.count().alias("company_frequency"))
df = df.join(company_frequency_df, on="company_hash", how="left")
df=df.drop(["company_hash"])
df=df.rename({"company_frequency": "company_hash"})
```

```
In [80]: df = df.with_columns(
    ctc=(df["ctc"] - df["ctc"].mean()) / df["ctc"].std(),
    yoe=(df["yoe"] - df["yoe"].mean()) / df["yoe"].std(),
    orgyear=(df["orgyear"] - df["orgyear"].mean()) / df["orgyear"].std(),
    ctc_updated_year=(df["ctc_updated_year"] - df["ctc_updated_year"].mean()) / df["ctc_updated_year"].std(),
    job_position=(df["job_position"] - df["job_position"].mean()) / df["job_position"].std(),
    ctc_bin=(df["ctc_bin"] - df["ctc_bin"].mean()) / df["ctc_bin"].std(),
    ctc_to_exp_ratio=(df["ctc_to_exp_ratio"] - df["ctc_to_exp_ratio"].mean()) / df["ctc_to_exp_ratio"].std(),
    growth=(df["growth"] - df["growth"].mean()) / df["growth"].std(),
    num_jobs=(df["num_jobs"] - df["num_jobs"].mean()) / df["num_jobs"].std(),
    company_hash=(df["company_hash"] - df["company_hash"].mean()) / df["company_hash"].std(),
)
df = df.with_columns(
    is_other=pl.col("is_other").cast(pl.Int32),
    is_developer=pl.col("is_developer").cast(pl.Int32),
    is_management=pl.col("is_management").cast(pl.Int32),
    is_non_coder=pl.col("is_non_coder").cast(pl.Int32),
    is_sales=pl.col("is_sales").cast(pl.Int32),
)
```

```
In [85]: df = df.unique()
df
```

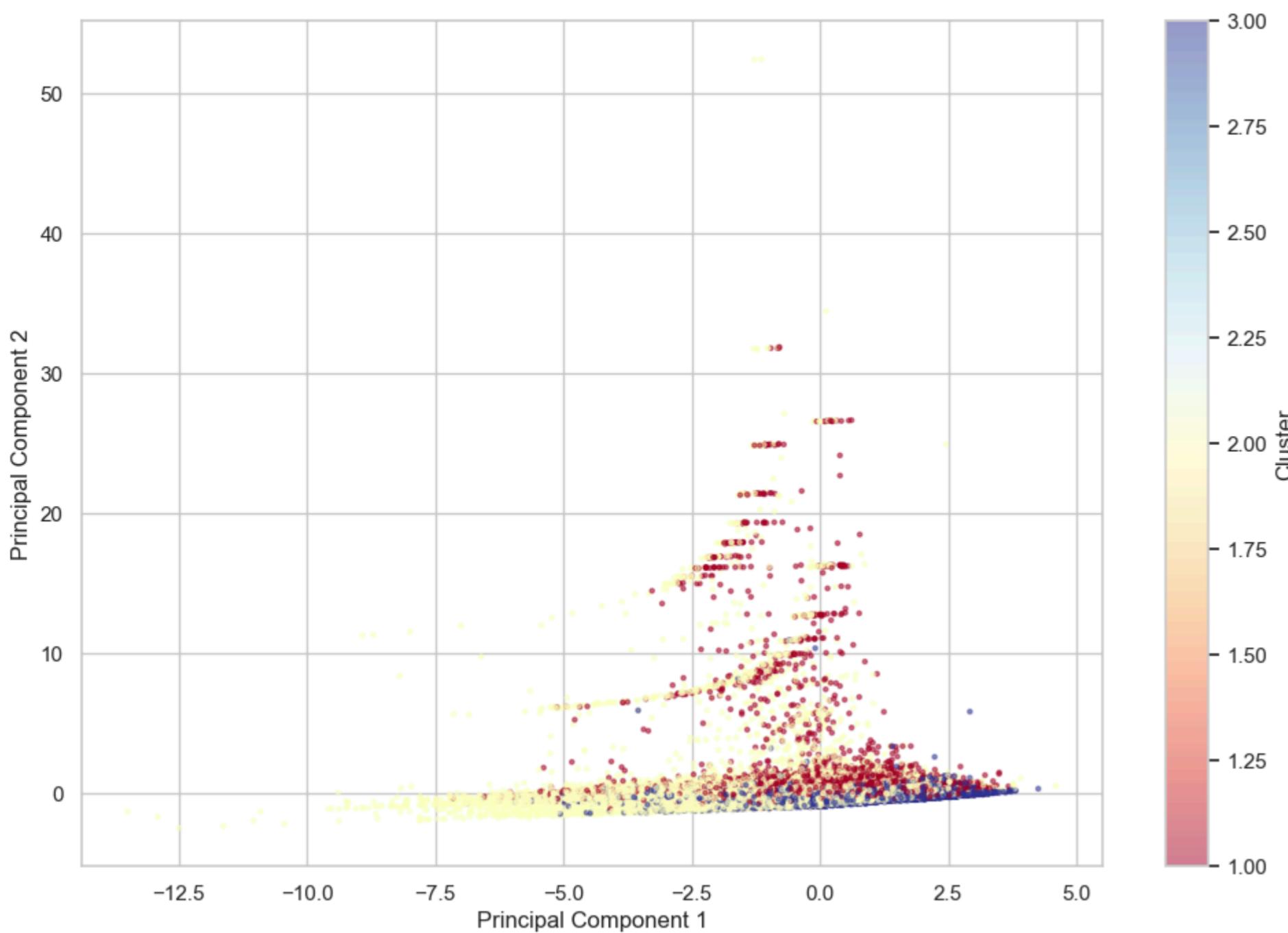
```
Out[85]: shape: (119_771, 18)
```

orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	class_flag	designation_flag	tier_flag	is_other	is_developer	is_management	is_sales	is_non_coder	ctc_bin	job_	
f64	f64	f64	f64	f64	f64	f64	i32	i32	i32	i32	i32	i32	i32	i32	i32	f64	
1.27299	-0.069908	0.3798	-1.27299	1.926407	-0.013919	0.106019	1	1	3	0	1	0	0	0	0	0.40634	-C
0.808337	-0.149511	-0.363565	-0.808337	-0.339155	-0.013919	-0.094719	3	3	3	0	0	0	0	0	0	-0.428769	-
0.57601	-0.161451	-0.363565	-0.57601	-0.339155	-0.013919	-0.113248	3	3	3	0	1	0	0	0	0	-1.263878	C
-1.050274	-0.149511	-0.363565	1.050274	-0.339155	-0.013919	-0.125601	2	2	1	0	1	1	1	1	0	-0.428769	-0
0.111358	-0.139958	-0.363565	-0.111358	-0.339155	-0.013919	-0.109278	3	1	3	0	1	0	0	0	0	-0.428769	-0
...	
0.57601	-0.14155	1.123165	-0.57601	-0.339155	-0.013919	-0.097807	2	3	3	0	1	0	0	0	0	-0.428769	1
-1.2826	-0.081849	-0.363565	1.2826	-0.339155	-0.013919	-0.106597	2	2	1	0	0	1	0	0	0	0.40634	-1
-0.120969	-0.149511	1.123165	0.120969	1.926407	-0.01104	-0.117881	2	1	3	1	0	0	0	0	1	-0.428769	
-4.53517	0.184819	-0.363565	4.53517	-0.339155	-0.013919	-0.08614	2	1	1	0	1	1	1	1	0	2.076557	-0
0.111358	-0.085829	-0.363565	-0.111358	-0.339155	-0.013919	-0.079277	2	3	1	0	1	0	0	0	0	0.40634	1

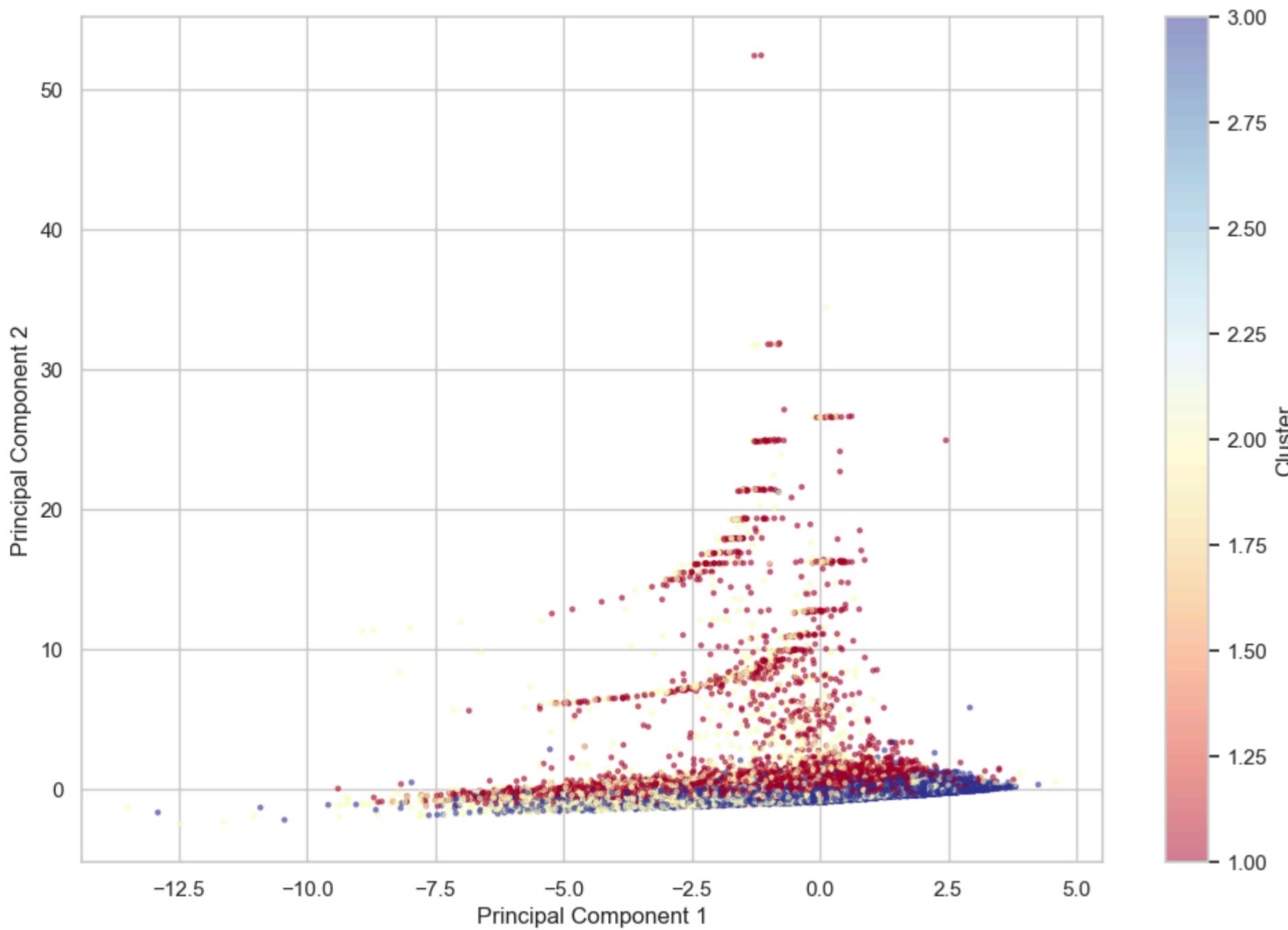
```
In [86]: X=df.to_pandas()
```

```
In [89]: X_embedded_pca = PCA(n_components=2).fit_transform(X.drop(["class_flag","designation_flag","tier_flag"], axis=1))
```

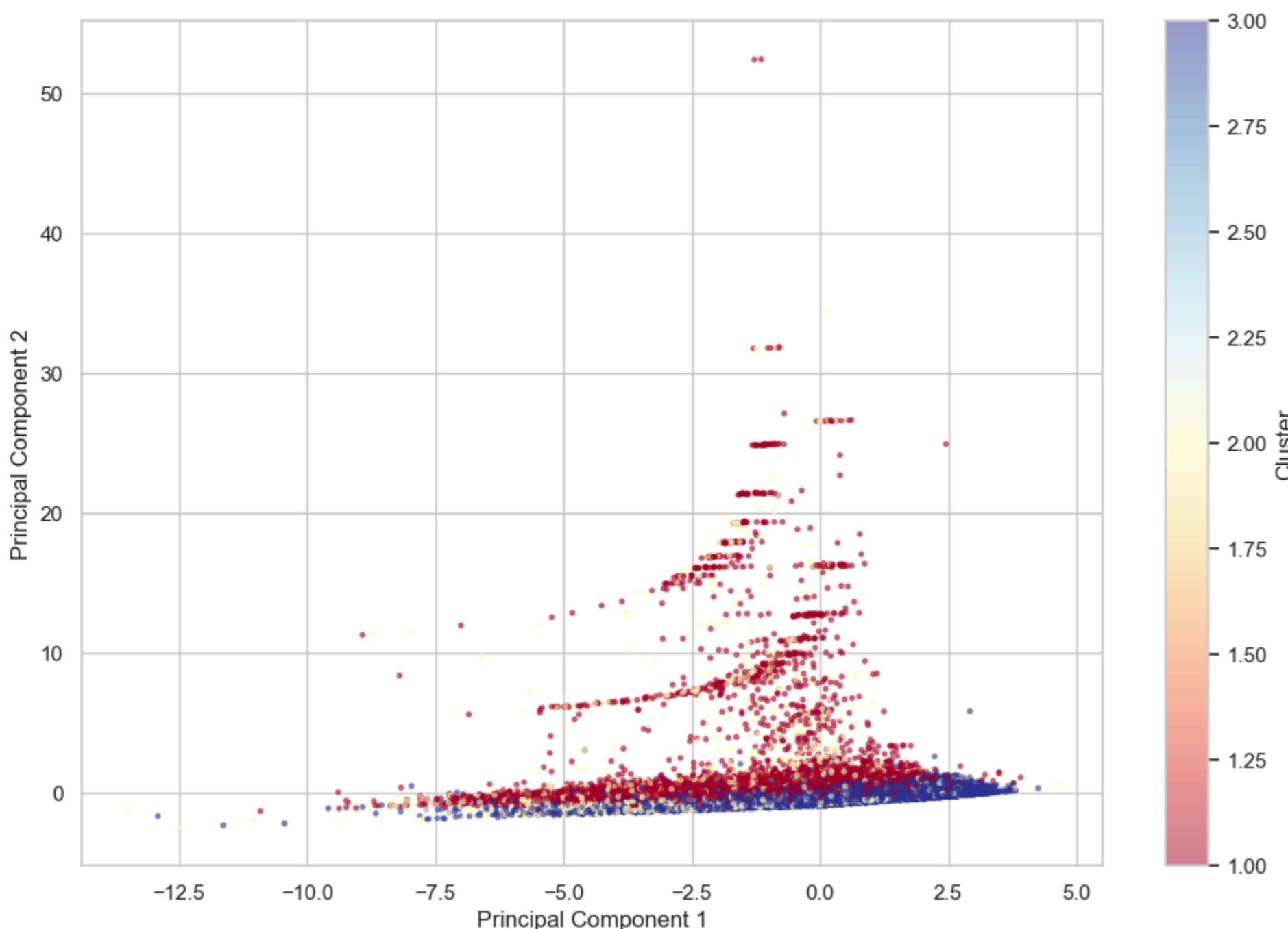
```
In [90]: plt.figure(figsize=(12, 8))
scatter = plt.scatter(X_embedded_pca[:, 0], X_embedded_pca[:, 1], c=X["class_flag"], s=5, cmap='RdYlBu', alpha=0.5)
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show();
```



```
In [91]: plt.figure(figsize=(12, 8))
scatter = plt.scatter(X_embedded_pca[:, 0], X_embedded_pca[:, 1], c=X["designation_flag"] , s=5, cmap='RdYlBu', alpha=0.5)
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show();
```



```
In [92]: plt.figure(figsize=(12, 8))
scatter = plt.scatter(X_embedded_pca[:, 0], X_embedded_pca[:, 1], c=X["tier_flag"] , s=5, cmap='RdYlBu', alpha=0.5)
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show();
```



Standardization & Encoding

```
In [87]: df=pl.read_parquet("../data/processed/data.parquet")
In [88]: df=df.drop(["email_hash", "class_flag", "designation_flag", "tier_flag", "company_hash", "is_tech"])
In [89]: df
Out[89]: shape: (133_146, 14)
          job_position orgyear      ctc  ctc_updated_year  yoe  num_jobs  growth  ctc_to_exp_ratio  is_other  is_developer  is_management  is_sales  is_non_coder  ctc_bin
          str        i32       i64        i32    i32       i64     f64        f64        bool        bool        bool        bool        bool        bool        str
  "data analyst"  2014  950000  2021      8      1  0.0  9.895833  false  false  false  false  false  false  "low"
"backend engineer"  2015  420000  2016      7      1  0.0  5.0  false  true  false  false  false  false  "very_low"
  "data scientist"  2017  1800000 2019      5      1  0.0  30.0  false  false  false  false  false  false  "high"
"fullstack engineer"  2017  700000  2021      5      1  0.0  11.666667  false  true  false  false  false  false  "low"
"backend engineer"  2013  1000000 2016      9      1  0.0  9.259259  false  true  false  false  false  false  "low"
...
  "sdet"  2014  600000  2019      8      1  0.0  6.25  false  true  false  false  false  false  "low"
"support engineer"  2016  220000  2019      6      1  0.0  3.055556  false  true  false  true  false  false  "very_low"
  "other"  2005  2500000 2021     17      1  0.0  12.254902  true  false  false  false  true  true  "high"
"android engineer"  2019  1050000 2020      3      1  0.0  29.166667  false  true  false  false  false  false  "medium"
"android engineer"  2019  1200000 2021      3      1  0.0  33.333333  false  true  false  false  false  false  "medium"
```

```
In [90]: df = df.with_columns(
  ctc_bin=pl.when(pl.col("ctc_bin") == "very_high").then(5)
  .when(pl.col("ctc_bin") == "high").then(4)
  .when(pl.col("ctc_bin") == "medium").then(3)
  .when(pl.col("ctc_bin") == "low").then(2)
  .otherwise(1)
)
```

```
In [91]: job_frequency_df = df.groupby("job_position").agg(pl.count().alias("job_pos_frequency"))
df = df.join(job_frequency_df, on="job_position", how="left")
df=df.drop(["job_position"])
df=df.rename({"job_pos_frequency": "job_position"})
```

```
In [92]: # company_frequency_df = df.groupby("company_hash").agg(pl.count().alias("company_frequency"))
# df = df.join(company_frequency_df, on="company_hash", how="left")
# df=df.drop(["company_hash"])
# df=df.rename({"company_frequency": "company_hash"})
```

```
In [93]: df
Out[93]: shape: (133_146, 14)
          orgyear      ctc  ctc_updated_year  yoe  num_jobs  growth  ctc_to_exp_ratio  is_other  is_developer  is_management  is_sales  is_non_coder  ctc_bin  job_position
          i32       i64        i32    i32       i64     f64        f64        bool        bool        bool        bool        bool        bool        i32       u32
  2014  950000  2021      8      1  0.0  9.895833  false  false  false  false  false  false  2  2574
  2015  420000  2016      7      1  0.0  5.0  false  true  false  false  false  false  1  37353
  2017  1800000 2019      5      1  0.0  30.0  false  false  false  false  false  false  4  4876
  2017  700000  2021      5      1  0.0  11.666667  false  true  false  false  false  false  2  21504
  2013  1000000 2016      9      1  0.0  9.259259  false  true  false  false  false  false  2  37353
...
  2014  600000  2019      8      1  0.0  6.25  false  true  false  false  false  false  2  4541
  2016  220000  2019      6      1  0.0  3.055556  false  true  false  true  false  false  1  3158
  2005  2500000 2021     17      1  0.0  12.254902  true  false  false  false  false  true  4  15663
  2019  1050000 2020      3      1  0.0  29.166667  false  true  false  false  false  false  3  4876
  2019  1200000 2021      3      1  0.0  33.333333  false  true  false  false  false  false  3  4876
```

```
In [94]: # standardize the data
df = df.with_columns(
  ctc=(df["ctc"] - df["ctc"].mean()) / df["ctc"].std(),
  yoe=(df["yoe"] - df["yoe"].mean()) / df["yoe"].std(),
  orgyear=(df["orgyear"] - df["orgyear"].mean()) / df["orgyear"].std(),
  ctc_updated_year=(df["ctc_updated_year"] - df["ctc_updated_year"].mean()) / df["ctc_updated_year"].std(),
  job_position=(df["job_position"] - df["job_position"].mean()) / df["job_position"].std(),
  ctc_bin=(df["ctc_bin"] - df["ctc_bin"].mean()) / df["ctc_bin"].std(),
  # class_flag=(df["class_flag"] - df["class_flag"].mean()) / df["class_flag"].std(),
```

```

# designation_flag=(df["designation_flag"] - df["designation_flag"].mean()) / df["designation_flag"].std(),
# tier_flag=(df["tier_flag"] - df["tier_flag"].mean()) / df["tier_flag"].std(),
ctc_to_exp_ratio=(df["ctc_to_exp_ratio"] - df["ctc_to_exp_ratio"].mean()) / df["ctc_to_exp_ratio"].std(),
growth=(df["growth"] - df["growth"].mean()) / df["growth"].std(),
num_jobs=(df["num_jobs"] - df["num_jobs"].mean()) / df["num_jobs"].std(),
# company_hash=(df["company_hash"] - df["company_hash"].mean()) / df["company_hash"].std(),
)

df = df.with_columns(
    is_other=pl.col("is_other").cast(pl.Int32),
    # is_tech=pl.col("is_tech").cast(pl.Int32),
    is_developer=pl.col("is_developer").cast(pl.Int32),
    is_management=pl.col("is_management").cast(pl.Int32),
    is_non_coder=pl.col("is_non_coder").cast(pl.Int32),
    is_sales=pl.col("is_sales").cast(pl.Int32),
)

```

In [95]: `df=df.unique()`

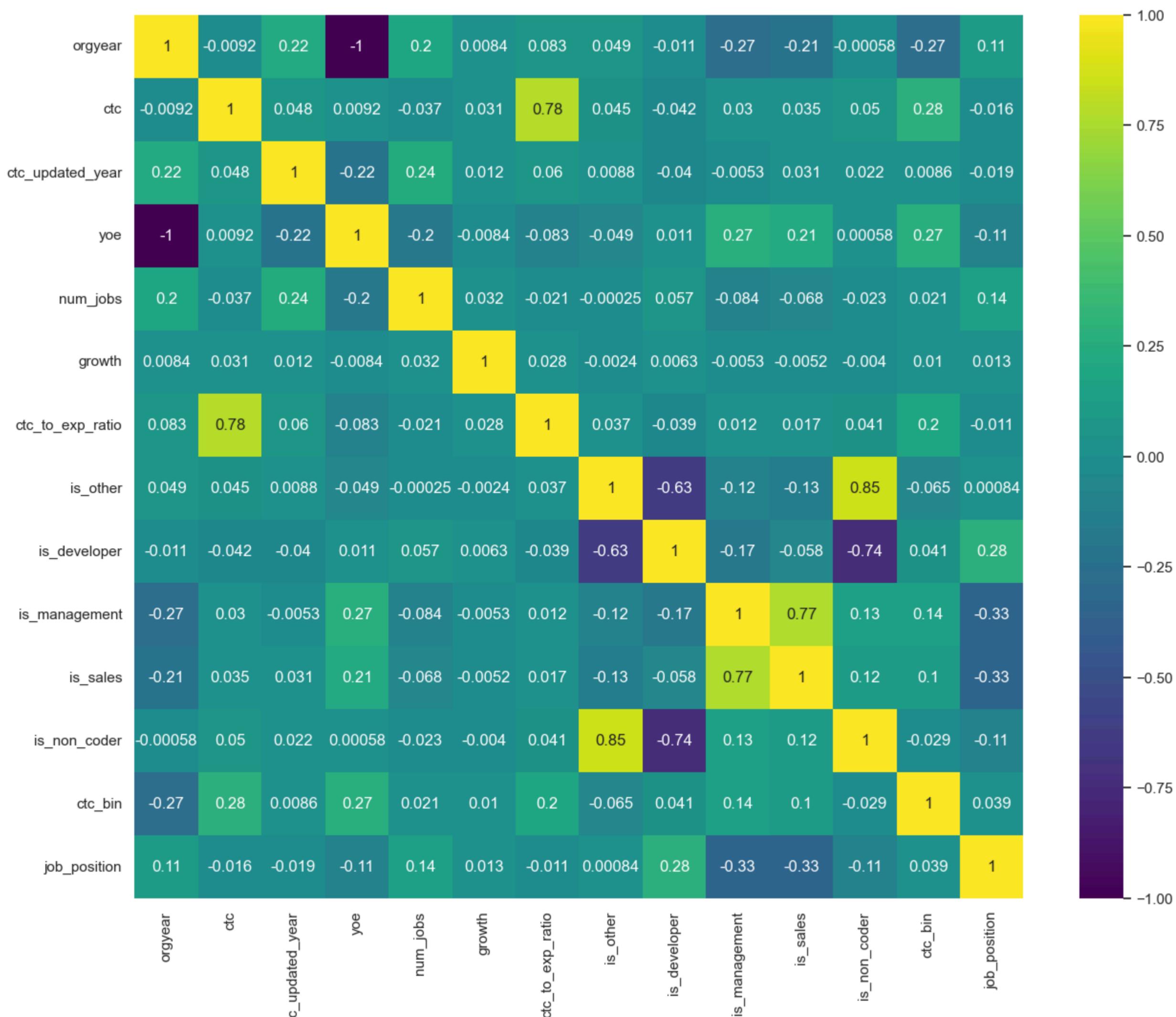
In [96]: `df`

Out[96]: `shape: (68_185, 14)`

orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	is_other	is_developer	is_management	is_sales	is_non_coder	ctc_bin	job_position
f64	f64	f64	f64	f64	f64	f64	i32	i32	i32	i32	i32	f64	f64
-0.120969	0.160938	1.123165	0.120969	-0.339155	-0.013919	0.032673	0	1	1	1	0	2.076557	-0.853597
-0.120969	-0.142346	-0.363565	0.120969	-0.339155	-0.013919	-0.114406	0	1	0	0	0	-0.428769	1.420651
-1.514927	0.280341	1.123165	1.514927	-0.339155	-0.013919	-0.008688	0	1	0	0	0	2.076557	-1.231019
0.808337	-0.149511	1.123165	-0.808337	-0.339155	-0.013919	-0.094719	0	0	1	1	1	-0.428769	-1.284216
0.808337	-0.181352	1.123165	-0.808337	1.926407	0.018472	-0.125601	0	1	0	0	0	-1.263878	1.420651
...
-1.747253	-0.13359	-1.106929	1.747253	-0.339155	-0.013919	-0.124572	0	1	0	0	0	-0.428769	-0.862877
1.040663	-0.159063	0.3798	-1.040663	-0.339155	-0.013919	-0.09163	0	1	0	0	0	-1.263878	-0.994628
0.57601	-0.159461	-0.363565	-0.57601	-0.339155	-0.013919	-0.111704	0	1	0	0	0	-1.263878	1.420651
0.808337	0.086749	0.3798	-0.808337	4.191969	-0.013919	0.134431	0	0	0	0	0	1.241448	-1.120752
0.111358	0.037555	1.123165	-0.111358	-0.339155	-0.013919	-0.010894	0	1	0	0	0	1.241448	0.262518

In [97]: `corr=df.to_pandas().corr()`

In [98]: `plt.figure(figsize=(15, 12))
sns.heatmap(corr, cmap="viridis", annot=True);`



Remove outliers using Local Outlier Factor

In [63]: `clf = LocalOutlierFactor(n_neighbors=20, contamination=0.05).fit_predict(df.to_pandas())`

In [64]: `df = df.with_columns(is_inlier=clf)
df["is_inlier"].value_counts()
df.filter(df["is_inlier"] == 1)`

```
Out[64]: shape: (2, 2)
is_inlier  count
i64      u32
-1      3410
1      64775
```

```
In [65]: X = df.to_pandas().drop(["is_inlier"], axis=1)
```

```
In [219... df.write_parquet("../data/processed/data_scaled.parquet")
X.to_parquet("../data/processed/X.parquet")
```

```
In [99]: df=pl.read_parquet("../data/processed/data_scaled.parquet")
X=pd.read_parquet("../data/processed/X.parquet")
```

Unsupervised Learning

Hopkins Test to check the cluster tendency

```
In [220... def hopkins_statistic(X, sample_size=0.1):
    n = X.shape[0]
    m = int(sample_size * n)

    random_indices = np.random.choice(np.arange(n), size=m, replace=False)
    X_sample = X.iloc[random_indices]

    X_min = np.min(X, axis=0)
    X_max = np.max(X, axis=0)
    X_uniform_random = np.random.uniform(X_min, X_max, (m, X.shape[1]))

    nbrs = NearestNeighbors(n_neighbors=1).fit(X)

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

    hopkins_stat = w_distances / (u_distances + w_distances)

    return hopkins_stat
```

```
In [221... hopkins_stat = hopkins_statistic(X)
print(f"Hopkins Statistic: {hopkins_stat}")
```

Hopkins Statistic: 1.0

Observations

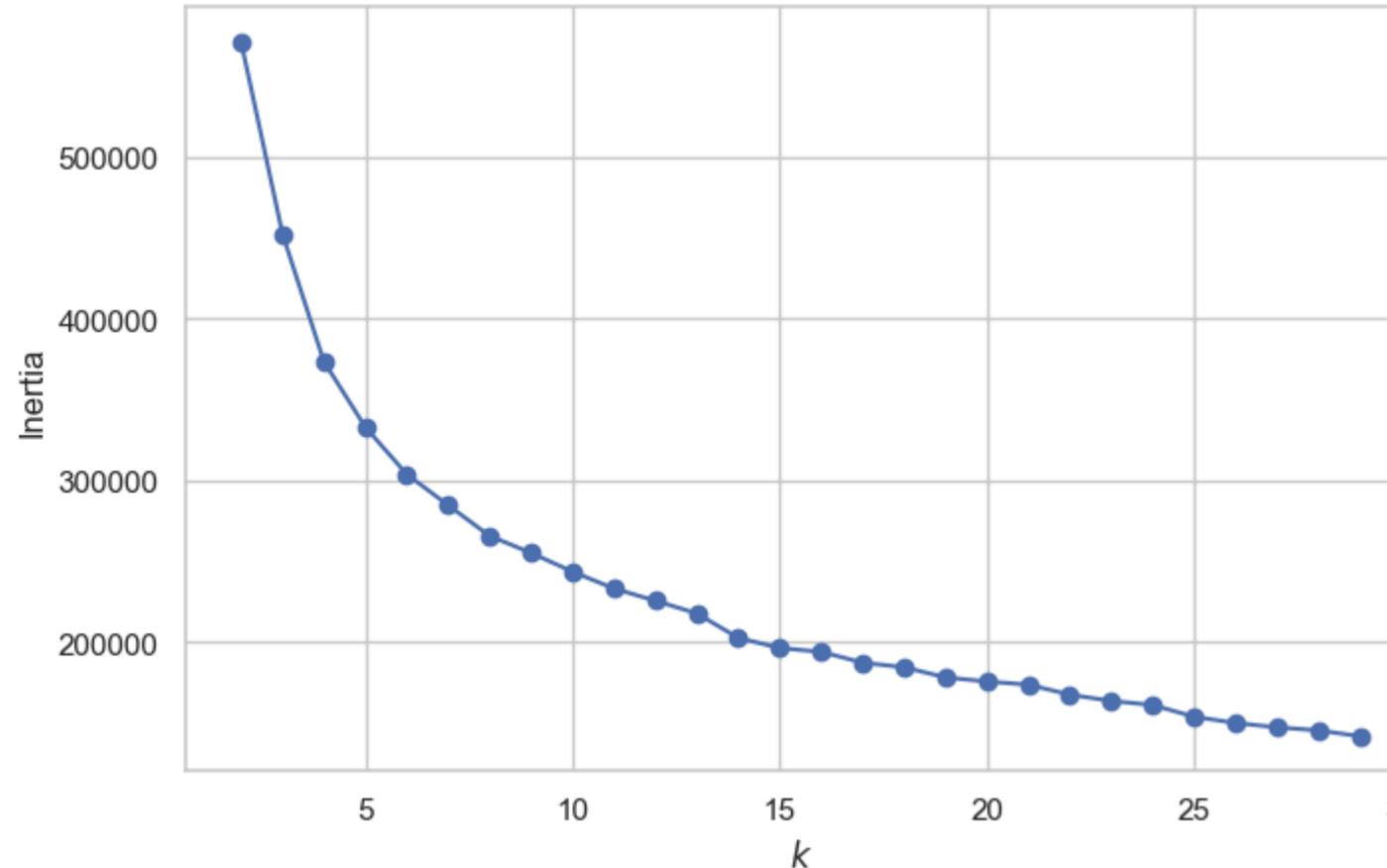
From Hopkins test we can see that the data is not suitable for clustering

K-Means Clustering

```
In [100... kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(X)
                     for k in range(2, 30, 1)]

inertias = [model.inertia_ for model in kmeans_per_k]

plt.figure(figsize=(8, 5))
plt.plot(range(2, 30, 1), inertias, "-o")
plt.xlabel("$k$")
plt.ylabel("Inertia")
plt.show();
```



Observations

- We will use K=5 for K-means clustering due to the presence of slight elbow

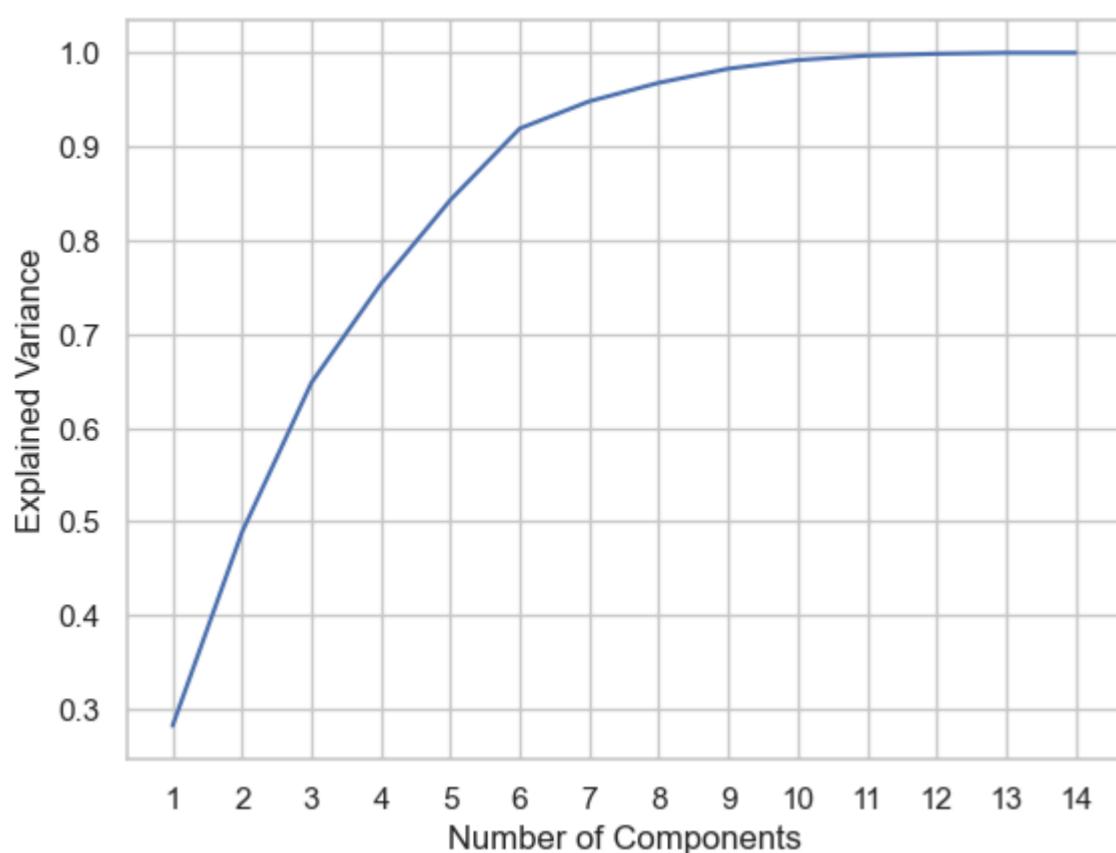
Using PCA to reduce the dimensions

```
In [101... pca_explained_variance = PCA().fit(X).explained_variance_ratio_.cumsum()
pca_explained_variance
```

```
Out[101... array([0.28207368, 0.48866291, 0.64811405, 0.75366195, 0.84304101,
   0.91929074, 0.94826786, 0.96800794, 0.98302944, 0.99222173,
   0.9969669 , 0.99893238, 1.        , 1.        ])
```

```
In [102... sns.lineplot(x=range(1, len(pca_explained_variance) + 1), y=pca_explained_variance)
plt.xlabel("Number of Components")
plt.ylabel("Explained Variance")

plt.xticks(range(1, len(pca_explained_variance) + 1))
plt.show();
```



Observations

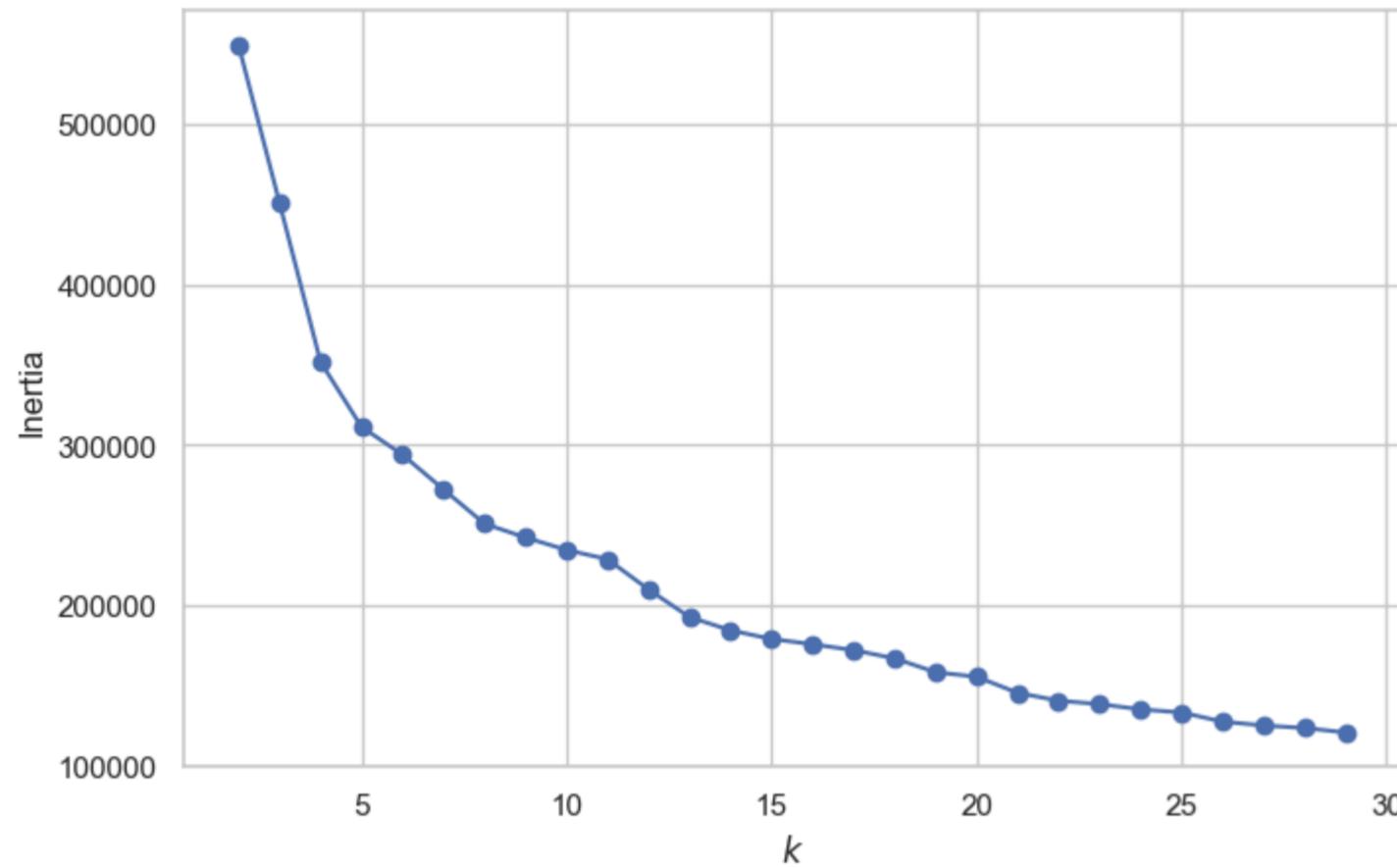
- We will use 8 components as it explains 95% of the variance.

```
In [103...]
X_embedded_pca = PCA(n_components=8).fit_transform(X)

kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(X_embedded_pca)
                 for k in range(2, 30, 1)]

inertias = [model.inertia_ for model in kmeans_per_k]

plt.figure(figsize=(8, 5))
plt.plot(range(2, 30, 1), inertias, "-o")
plt.xlabel("$k$")
plt.ylabel("Inertia")
plt.show();
```



Observations

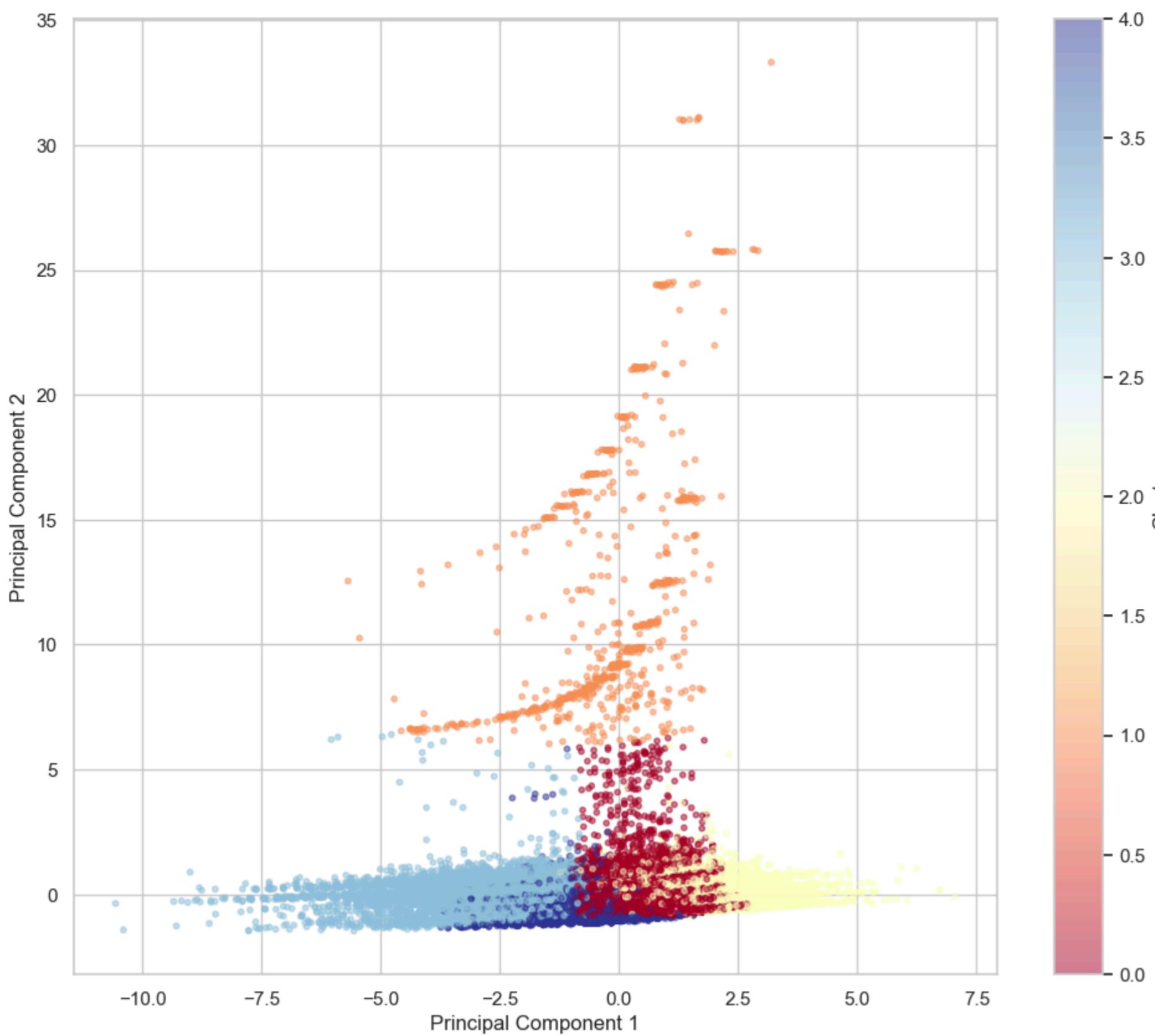
From above plot we are not able to find the optimal number of clusters. But since there seems to be a slight elbow at 5, we can consider 5 clusters.

```
In [104...]
kmeans = KMeans(n_clusters=5, random_state=42)
y_kmeans = kmeans.fit_predict(X_embedded_pca)
df=df.with_columns(
    y_kmeans=y_kmeans
)

In [113...]
df
```

Out[113...]																
orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	is_other	is_developer	is_management	is_sales	is_non_coder	ctc_bin	job_position	is_inlier	y_kmeans	
f64	f64	f64	f64	f64	f64	f64	i32	i32	i32	i32	i32	f64	f64	i64	i32	
0.343684	-0.182148	1.123165	-0.343684	-0.339155	-0.013919	-0.131263	0	1	0	0	0	-1.263878	-0.952538	1	0	
0.808337	-0.085829	-1.106929	-0.808337	-0.339155	-0.013919	-0.032953	0	0	0	0	0	0.40634	-1.120752	1	0	
-0.817948	0.105216	1.123165	0.817948	-0.339155	-0.013919	-0.034357	0	1	0	0	0	1.241448	-0.632186	1	3	
-0.120969	-0.038068	0.3798	0.120969	-0.339155	-0.013919	-0.063836	1	0	0	0	1	1.241448	-0.1643	1	0	
-2.908885	0.129097	-0.363565	2.908885	-0.339155	-0.013919	-0.077733	0	1	0	0	0	2.076557	-1.231019	1	3	
...	
-0.353295	-0.101749	-0.363565	0.353295	1.926407	-0.013919	-0.099866	0	1	0	0	0	0.40634	-0.977017	1	2	
0.808337	-0.135182	-0.363565	-0.808337	-0.339155	-0.013919	-0.080821	0	1	0	1	0	-0.428769	-1.078077	1	0	
-2.676559	0.081336	1.123165	2.676559	-0.339155	-0.013919	-0.084153	1	0	0	0	1	1.241448	-0.1643	1	3	
0.111358	0.033574	0.3798	-0.111358	-0.339155	-0.013919	-0.0131	0	1	0	0	0	1.241448	1.420651	1	0	
0.808337	-0.118466	-0.363565	-0.808337	-0.339155	-0.013919	-0.064608	1	0	0	0	1	-0.428769	-0.1643	1	0	

```
In [227...]
plt.figure(figsize=(12, 10))
scatter=plt.scatter(X_embedded_pca[:, 0], X_embedded_pca[:, 1], c=y_kmeans, s=10, cmap='RdYlBu', alpha=0.5 )
# add legend
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show();
```



Observations

- From above clustering plot, we can see there is similarity between plot generated by manual clustering and k-means clustering.
- We can see that the clusters 1 is far from other clusters

```
In [105...]: def create_polar_plot(data, features, cluster_label):
    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[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):
    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):
        data = np.concatenate((data, [data[0]]))
        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()
```

Cluster analysis

```
In [106...]: cluster_means = df.groupby("y_kmeans").mean().to_pandas().drop(["y_kmeans", "is_inlier"], axis=1)

scaler = MinMaxScaler()
normalized_means = scaler.fit_transform(cluster_means)

In [107...]: cluster_means
```

	orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	is_other	is_developer	is_management	is_sales	is_non_coder	ctc_bin	job_position
0	-1.918170	0.018687	-0.138136	1.918170	-0.209353	-0.010262	-0.084150	0.076208	0.757585	0.296871	0.272362	0.144535	0.806881	-0.483044
1	0.376624	-0.069102	0.302020	-0.376624	-0.339155	-0.013919	-0.038127	0.111660	0.740836	0.095240	0.111131	0.146267	-0.081313	-0.290874
2	-0.327516	-0.089385	-1.855684	0.327516	-0.216063	-0.012841	-0.092348	0.107654	0.827551	0.059655	0.057636	0.127111	-0.045168	0.197273
3	0.390229	-0.069585	0.562197	-0.390229	2.549371	0.065804	-0.046173	0.100049	0.828030	0.047318	0.067492	0.114401	0.185543	0.132658
4	0.120952	9.461221	0.444488	-0.120952	-0.258959	-0.005299	7.738631	0.231563	0.597345	0.168142	0.196165	0.299410	2.076557	-0.383857

```
In [108...]: cluster_means.columns.to_list()
```

```

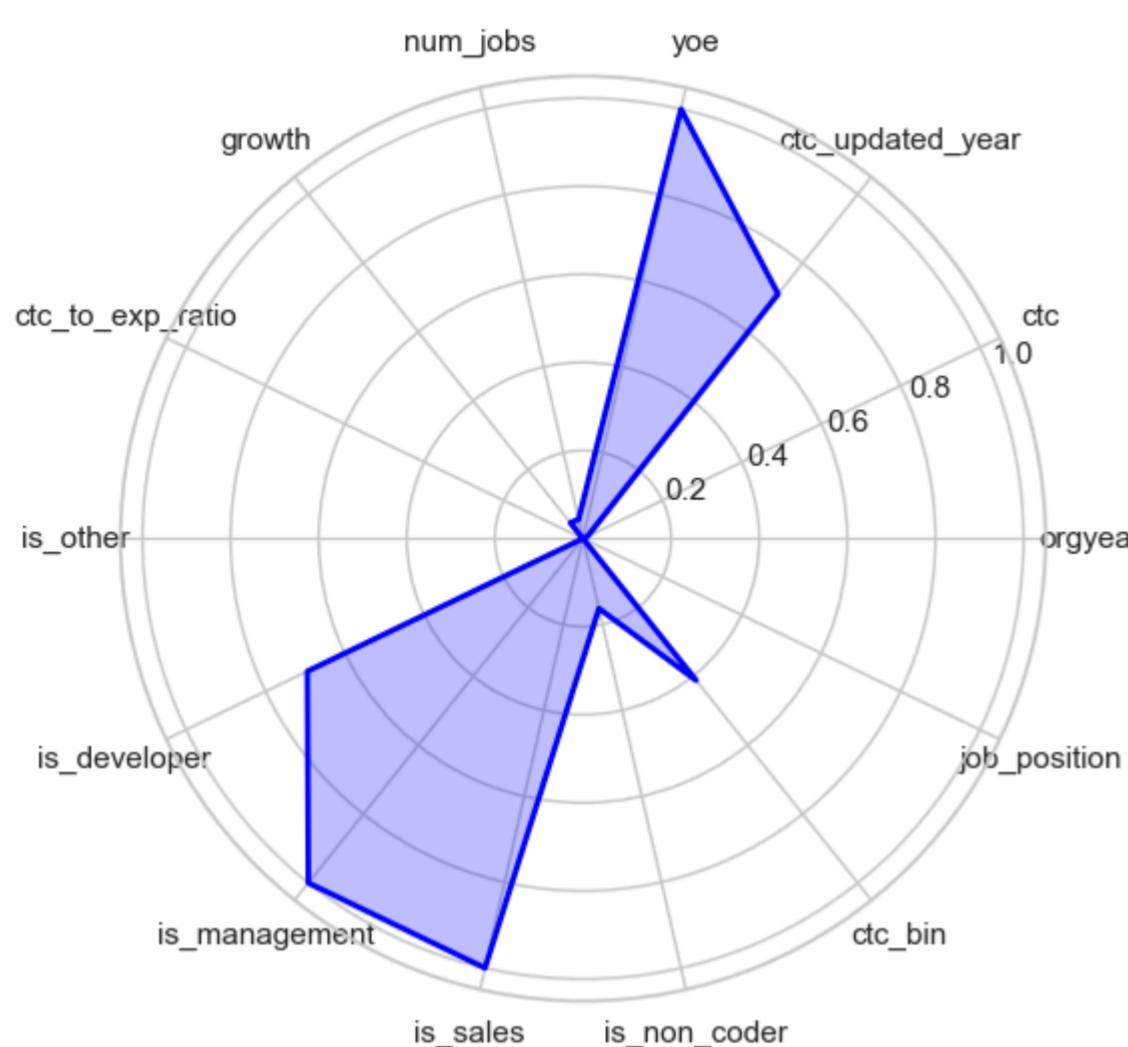
Out[108]: ['orgyear',
 'ctc',
 'ctc_updated_year',
 'yoe',
 'num_jobs',
 'growth',
 'ctc_to_exp_ratio',
 'is_other',
 'is_developer',
 'is_management',
 'is_sales',
 'is_non_coder',
 'ctc_bin',
 'job_position']

In [114]: features = cluster_means.columns.to_list()

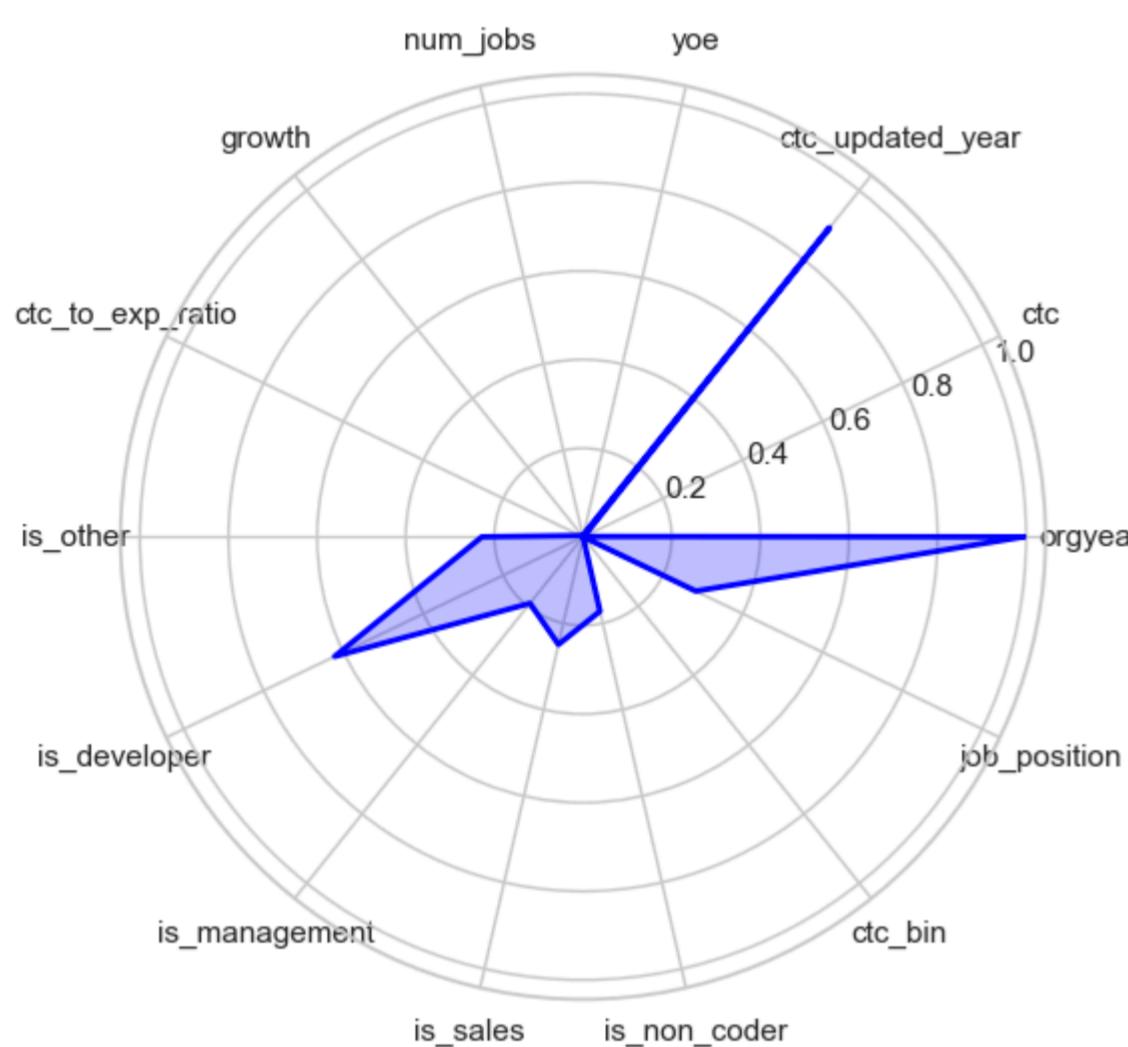
In [115]: for cluster in range(cluster_means.shape[0]):
    create_polar_plot(normalized_means[cluster], features, cluster)

```

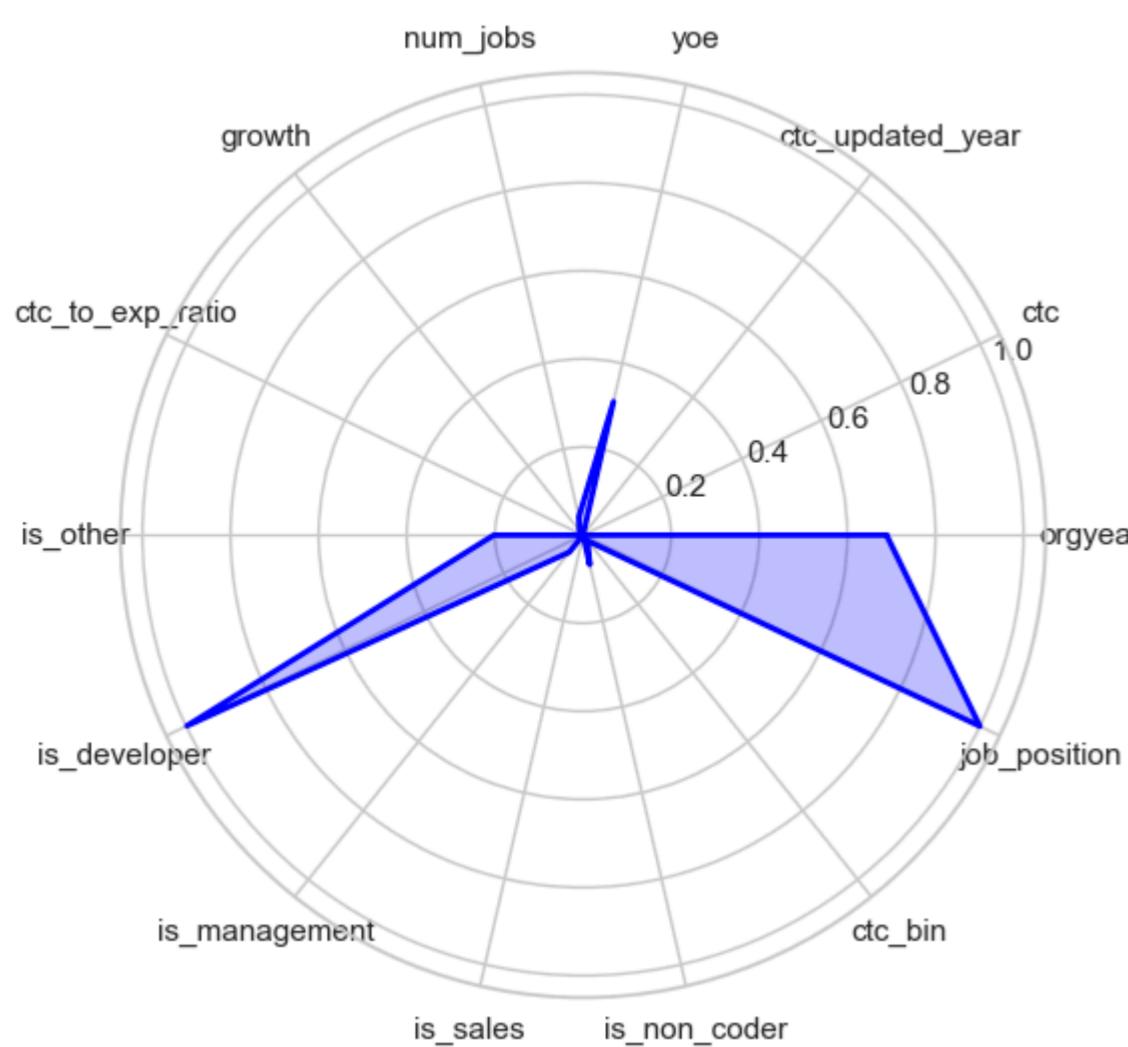
Cluster 0



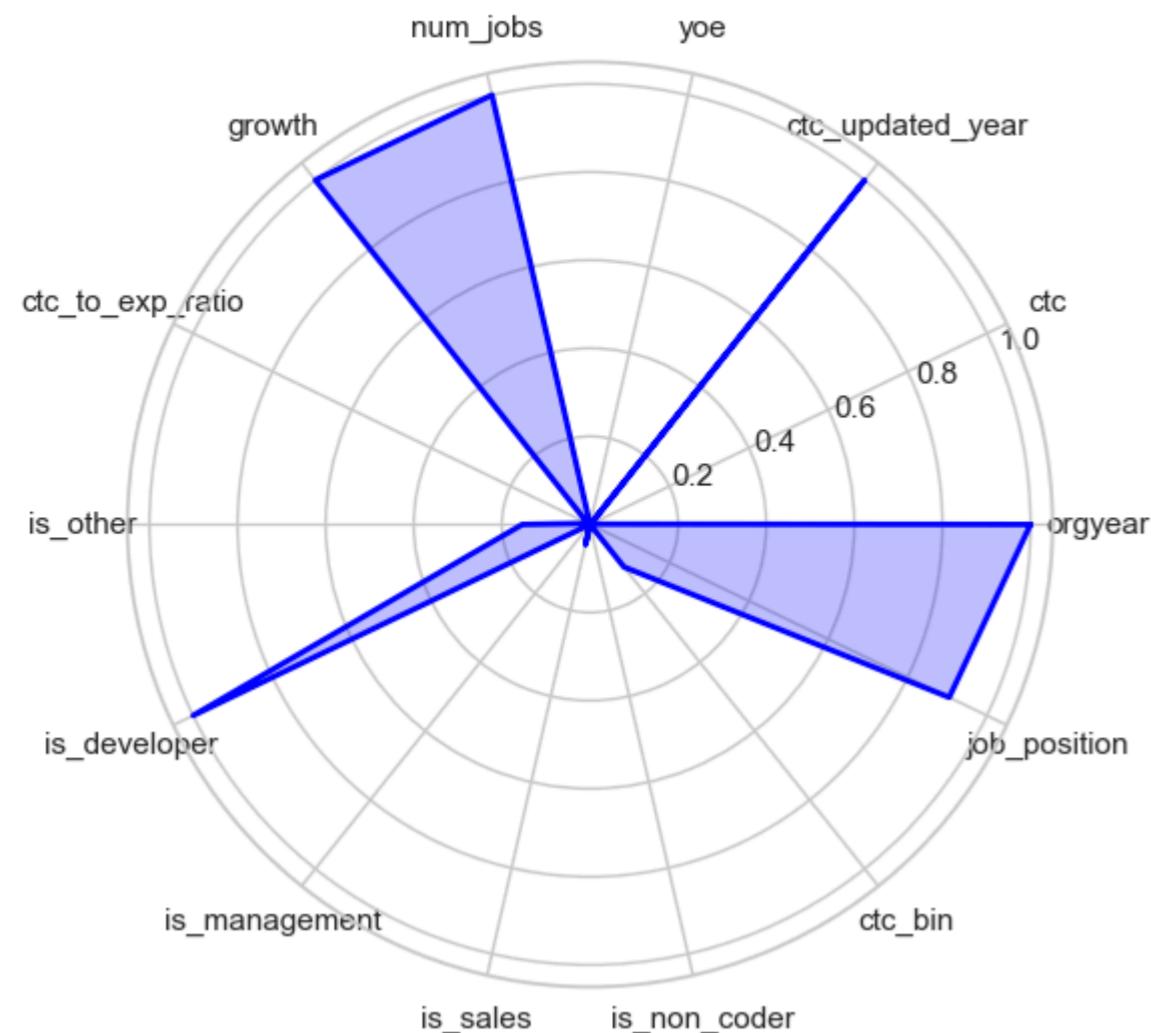
Cluster 1



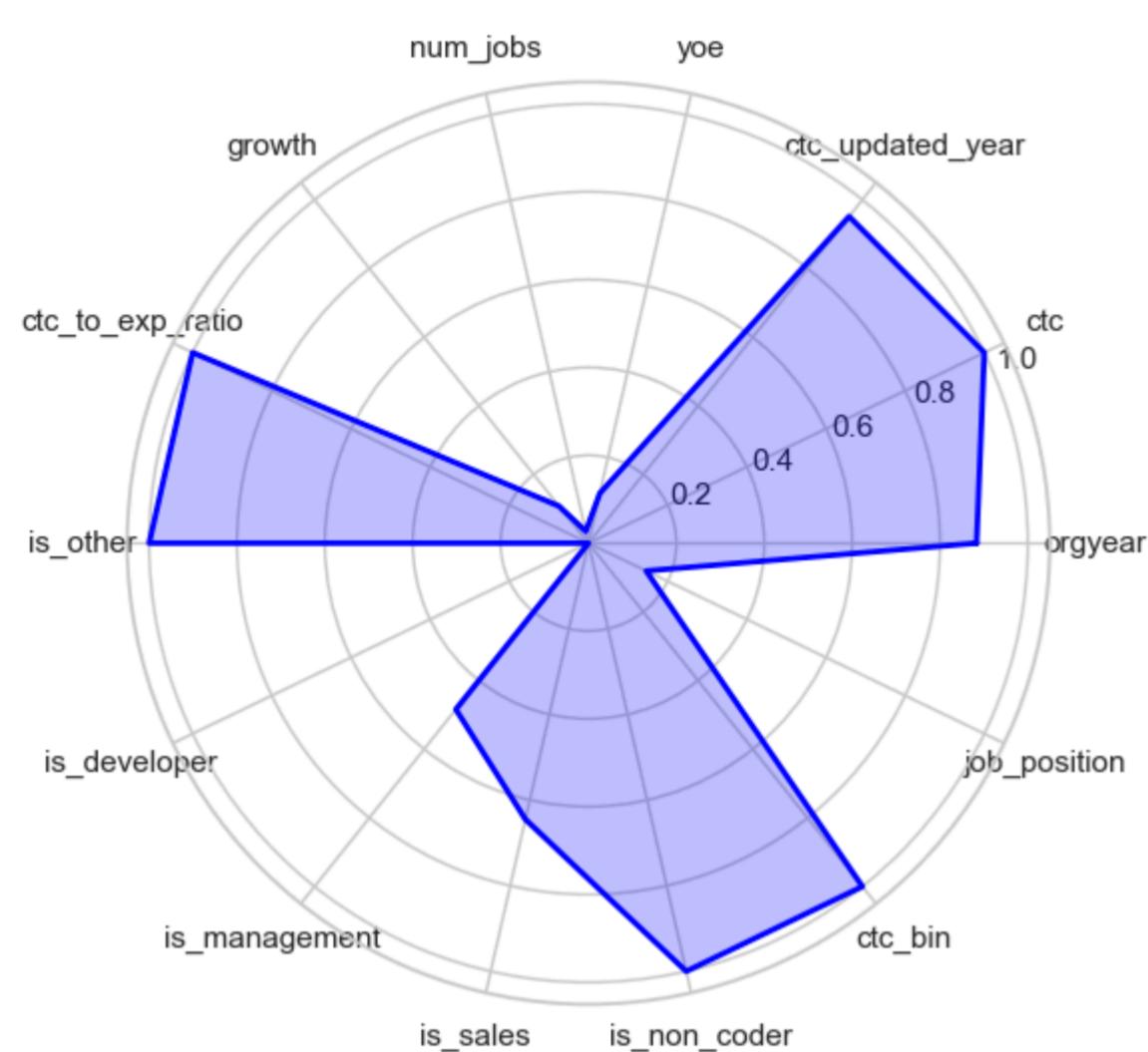
Cluster 2



Cluster 3



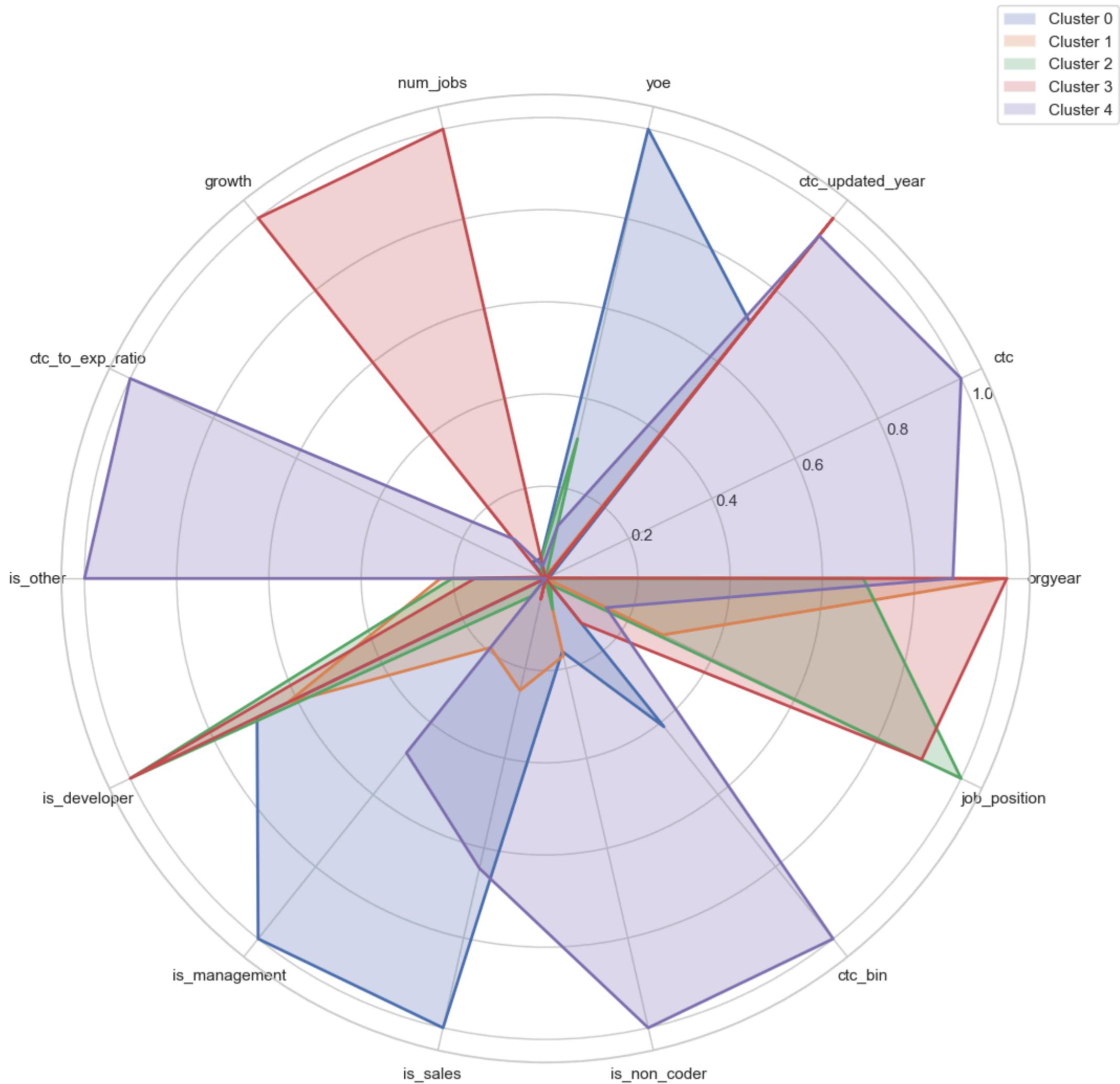
Cluster 4



Combined Polar Plot

```
In [111]: cluster_labels = df["y_kmeans"].unique().to_list()
create_combined_polar_plot(normalized_means, features, cluster_labels)
```

Cluster Comparison



Cluster 0

Candidates in this cluster belong to the management and sales category and have higher experience.

Cluster 1

Candidates in this cluster have lower experience. Also they belong developers category and have joined recently

Cluster 2

Candidates in this cluster belong to developer category and have lower experience

Cluster 3

Candidates in this cluster belong to developer category and have switched jobs multiple times

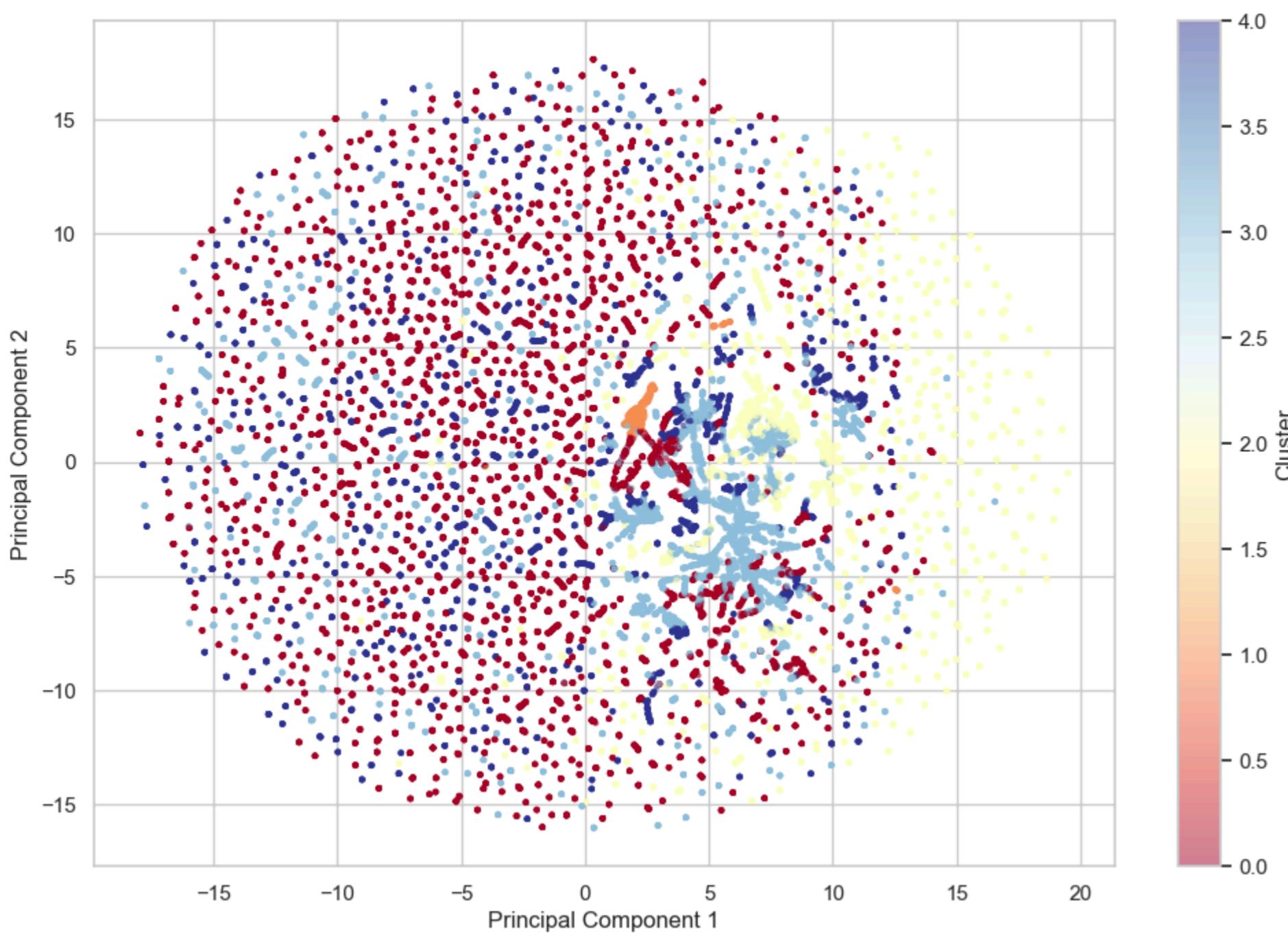
Cluster 4

Candidates in this cluster belong to non coder category and have higher ctc

Plotting clusters using UMAP

```
In [29]: X_embedded_tsne = umap.UMAP().fit_transform(X)
```

```
In [237... plt.figure(figsize=(12, 8))
scatter = plt.scatter(X_embedded_tsne[:, 0], X_embedded_tsne[:, 1], c=y_kmeans, s=5, cmap='RdYlBu', alpha=0.5)
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show();
```



Hierarchical Clustering

```
In [3]: X_embedded_pca = PCA(n_components=8).fit_transform(X)
In [4]: sample = pd.DataFrame(X_embedded_pca).sample(frac=0.9, random_state=42)
In [5]: sample_hc = X.loc[sample.index]
In [6]: sample_hc
```

	orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	is_other	is_developer	is_management	is_sales	is_non_coder	ctc_bin	job_position
22086	0.343684	-0.173391	-0.363565	-0.343684	-0.339155	-0.013919	-0.125601	0	1	0	0	0	-1.263878	-0.862877
34847	0.576010	-0.117670	1.123165	-0.576010	-0.339155	-0.013919	-0.079277	0	0	1	1	0	-0.428769	-1.269674
40754	0.576010	-0.196794	-0.363565	-0.576010	-0.339155	-0.013919	-0.140672	1	0	0	0	1	-1.263878	-0.164300
30258	-0.120969	-0.180078	-2.593659	0.120969	-0.339155	-0.013919	-0.132704	0	1	0	0	0	-1.263878	-0.632186
8006	0.808337	-0.154287	1.123165	-0.808337	1.926407	-0.013919	-0.099351	1	0	0	0	1	-1.263878	-0.164300
...
30453	0.343684	-0.128814	-0.363565	-0.343684	-0.339155	-0.013919	-0.096777	0	1	0	0	0	-0.428769	-1.121263
59128	0.808337	-0.141550	1.123165	-0.808337	-0.339155	-0.013919	-0.086998	0	1	0	0	0	-0.428769	-0.952538
30257	-0.120969	-0.074685	-2.593659	0.120969	-0.339155	-0.013919	-0.081593	0	0	1	1	1	0.406340	-1.230215
35595	0.343684	-0.189312	-1.106929	-0.343684	-0.339155	-0.013919	-0.135896	0	1	0	0	0	-1.263878	0.262518
22164	0.576010	-0.046028	1.123165	-0.576010	-0.339155	-0.013919	-0.023688	0	1	0	0	0	1.241448	-0.862877

58298 rows × 14 columns

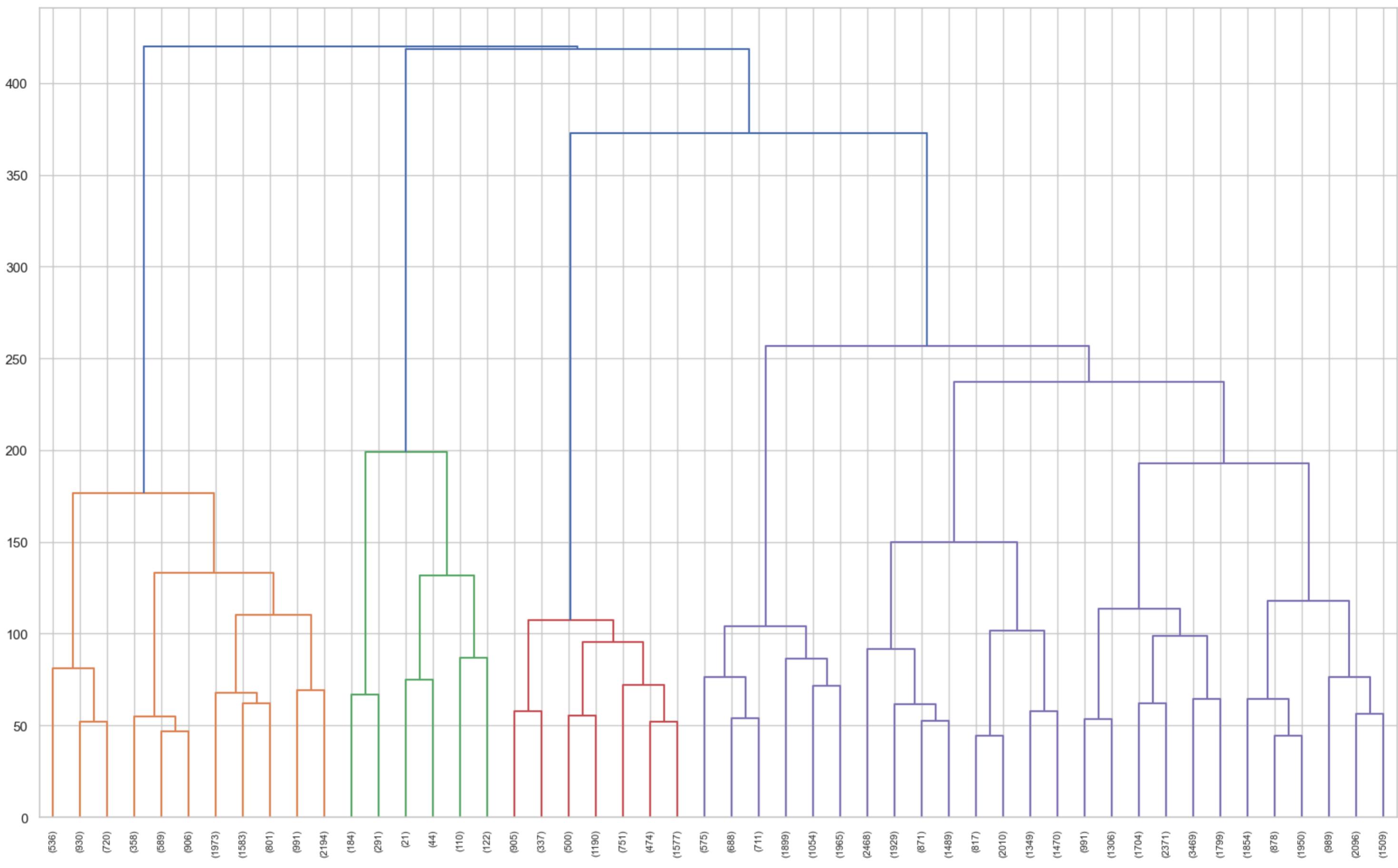
Build Linkage Matrix

```
In [7]: %%time
Z = fastcluster.linkage(sample, method='ward')
CPU times: user 3min 33s, sys: 9min 3s, total: 12min 36s
Wall time: 22min 46s
In [13]: import pickle
# with open("../models/hierarchical_clustering.pkl", "wb") as f:
#     pickle.dump(Z, f)

with open("../models/hierarchical_clustering.pkl", "rb") as f:
    Z = pickle.load(f)
```

Plot Dendrogram

```
In [9]: fig, ax = plt.subplots(figsize=(20, 12))
dendrogram(Z, truncate_mode='lastp', p=50);
```



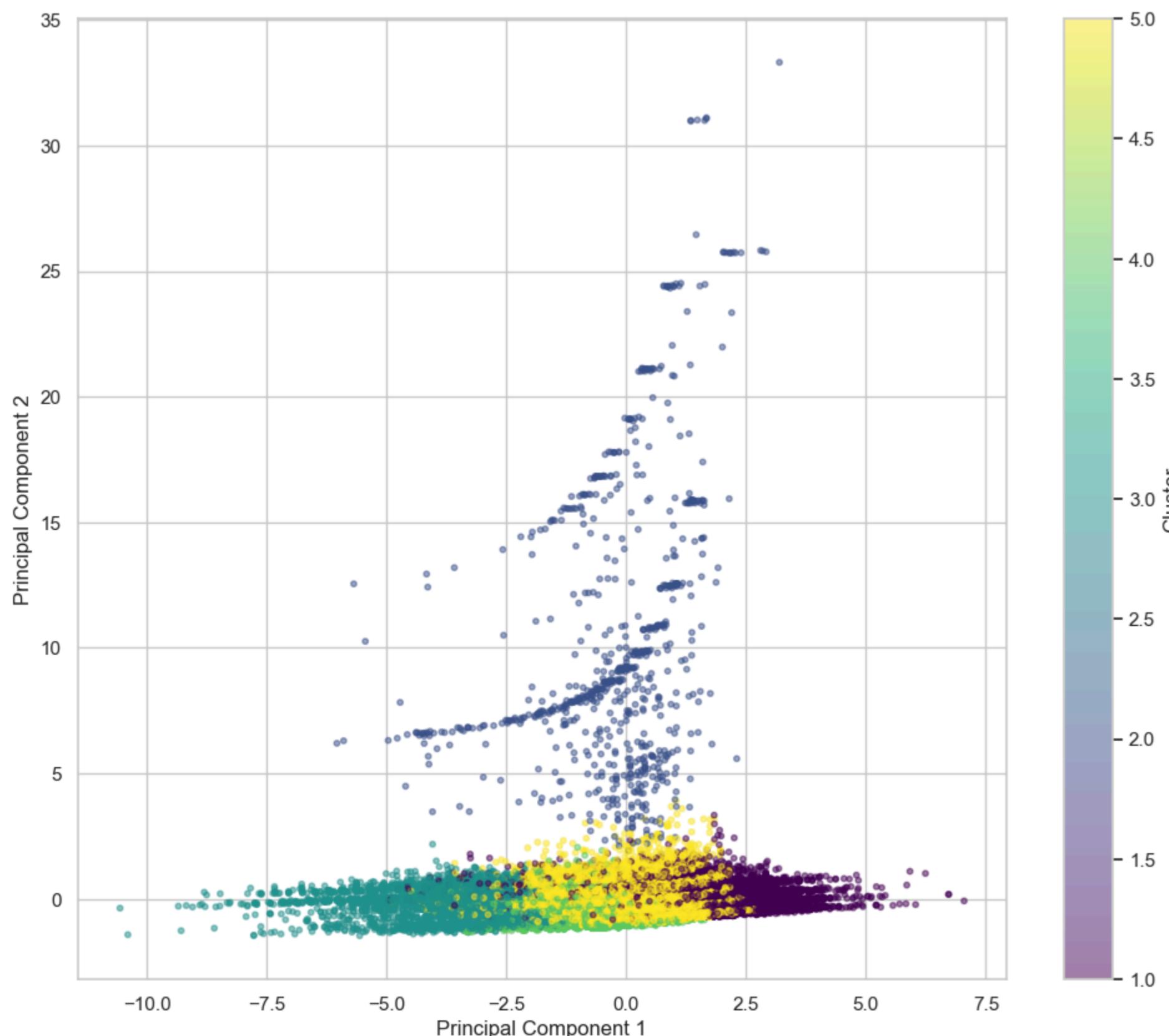
Observations

- We will cut the dendrogram at 250 which will give us 5 clusters.

```
In [14]: sample_hc["cluster"] = fcluster(Z, 5, criterion='maxclust')
```

Plotting Hierarchical Clusters

```
In [12]: plt.figure(figsize=(12, 10))
scatter=plt.scatter(sample.iloc[:, 0], sample.iloc[:, 1], c=sample_hc["cluster"], s=10, cmap='viridis', alpha=0.5 )
# add legend
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show();
```



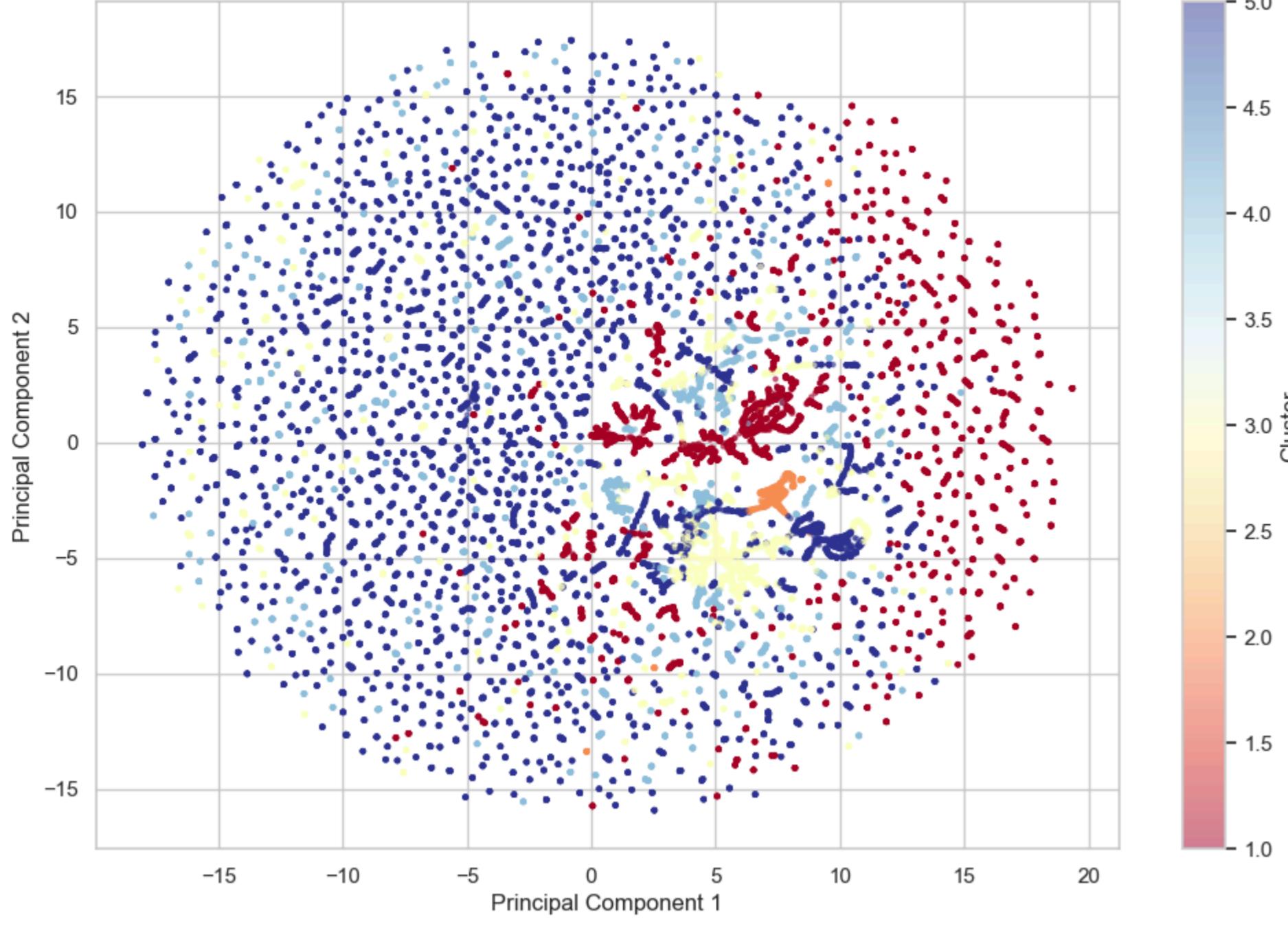
Observations

- From above plot we can see that the clusters are not well defined.
- Cluster 1

Plotting clusters using UMAP

```
In [30]: X_embedded_tsne = umap.UMAP().fit_transform(sample)
```

```
In [31]: plt.figure(figsize=(12, 8))
scatter = plt.scatter(X_embedded_tsne[:, 0], X_embedded_tsne[:, 1], c=sample_hc["cluster"], s=5, cmap='RdYlBu', alpha=0.5)
plt.colorbar(scatter, label='Cluster')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show();
```



Cluster Analysis

```
In [18]: cluster_means = sample_hc.groupby("cluster").mean()
scaler = MinMaxScaler()
normalized_means = scaler.fit_transform(cluster_means)
features = cluster_means.columns.to_list()
```

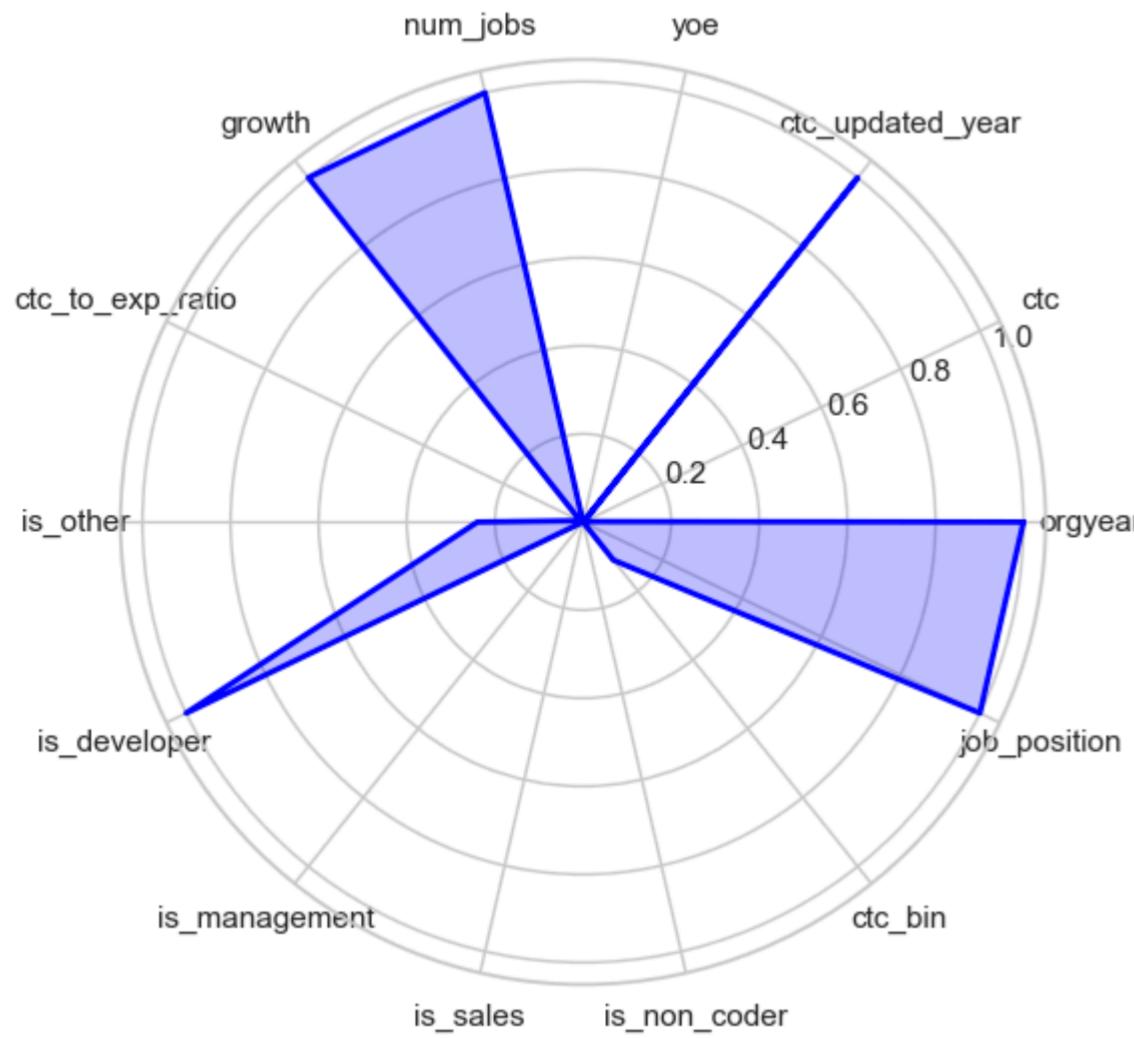
```
In [23]: cluster_means
```

```
Out[23]:
```

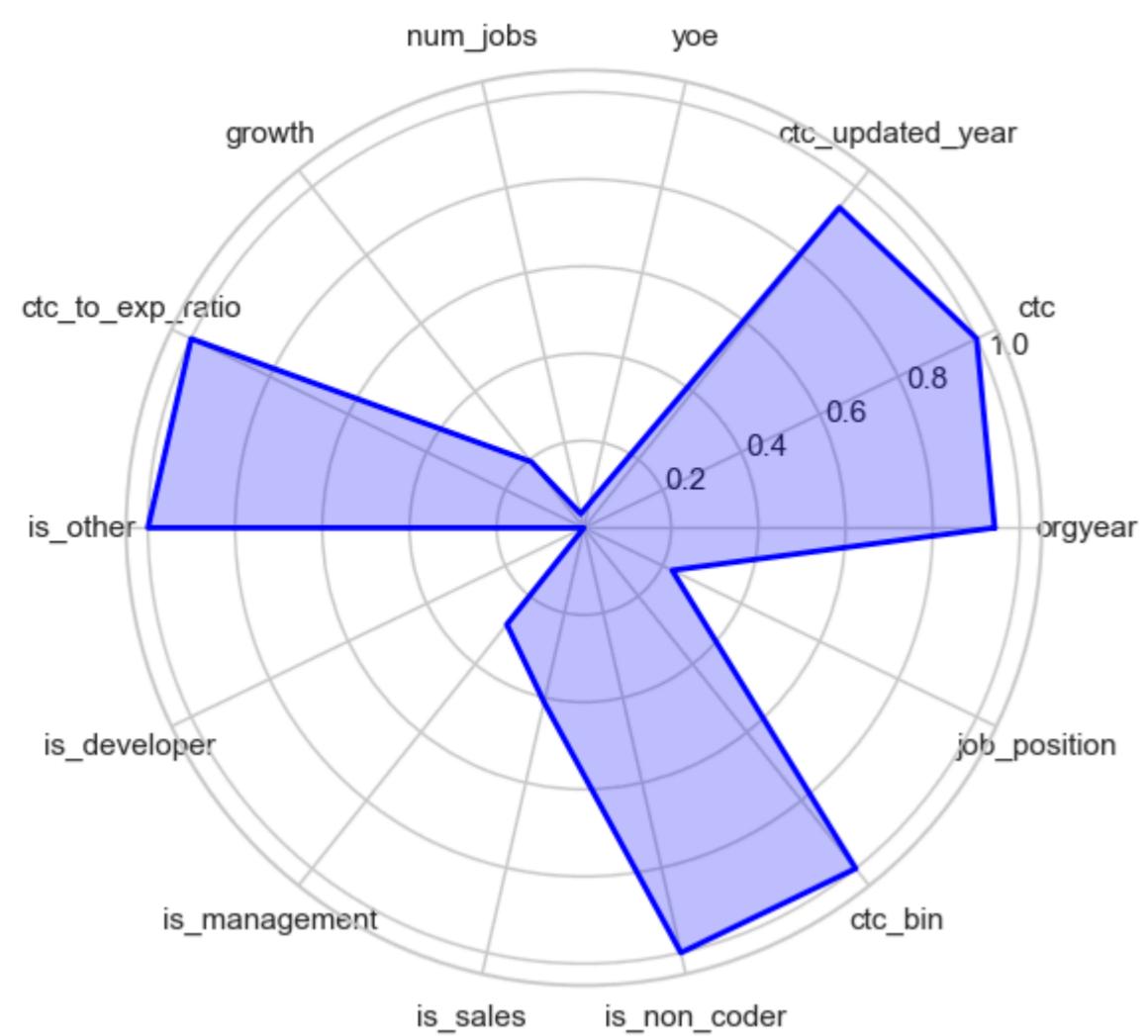
	orgyear	ctc	ctc_updated_year	yoe	num_jobs	growth	ctc_to_exp_ratio	is_other	is_developer	is_management	is_sales	is_non_coder	ctc_bin	job_position
cluster														
1	0.273551	-0.064851	0.564342	-0.273551	2.519354	0.064651	-0.049069	0.099042	0.826008	0.060357	0.079009	0.116657	0.227506	0.105622
2	0.109853	8.087376	0.400021	-0.109853	-0.245246	0.001319	6.560484	0.303109	0.553109	0.148964	0.181347	0.356218	2.076557	-0.348436
3	-2.559585	0.003628	-0.363954	2.559585	-0.322561	-0.013919	-0.097548	0.034705	0.783572	0.372864	0.328392	0.117021	0.700973	-0.480073
4	-0.310585	-0.086293	-2.179157	0.310585	-0.138963	-0.010967	-0.090778	0.165264	0.722867	0.102438	0.094312	0.200087	-0.002248	-0.330149
5	0.073830	-0.075520	0.141434	-0.073830	-0.339155	-0.013919	-0.060270	0.098652	0.769381	0.101474	0.114109	0.131727	0.049653	-0.172190

```
In [22]: for cluster in range(cluster_means.shape[0]):
    create_polar_plot(normalized_means[cluster], features, cluster+1)
```

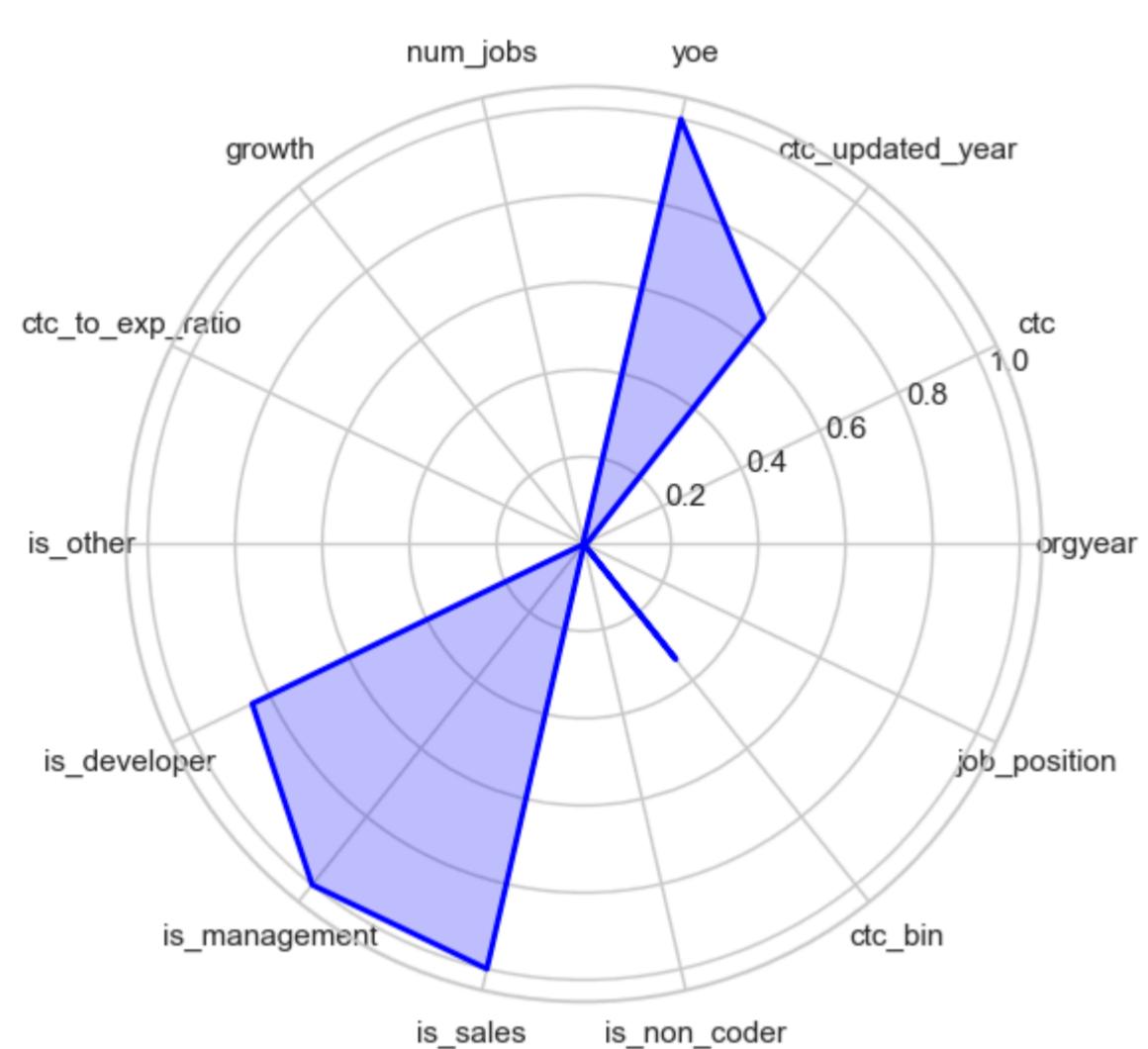
Cluster 1



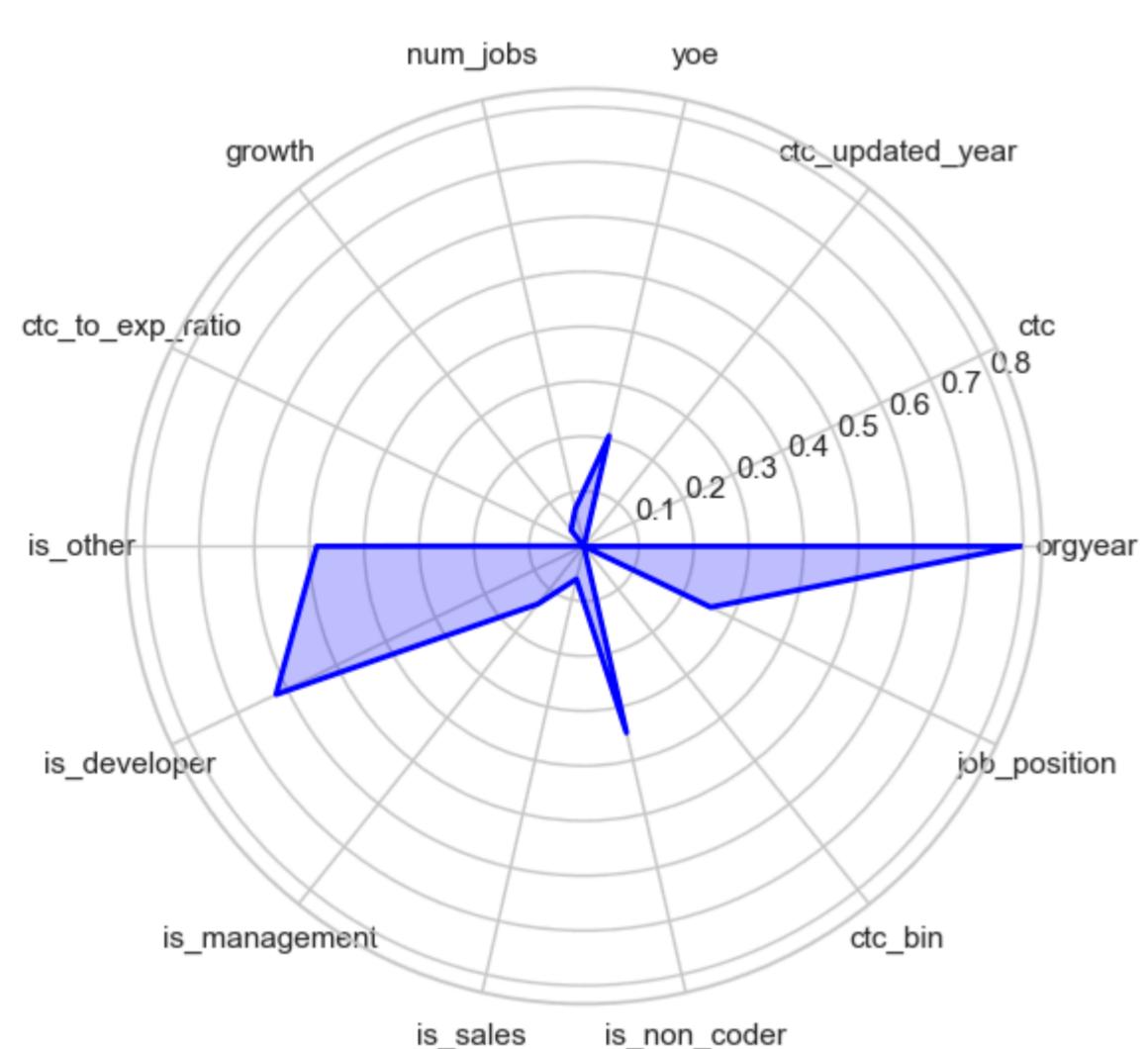
Cluster 2



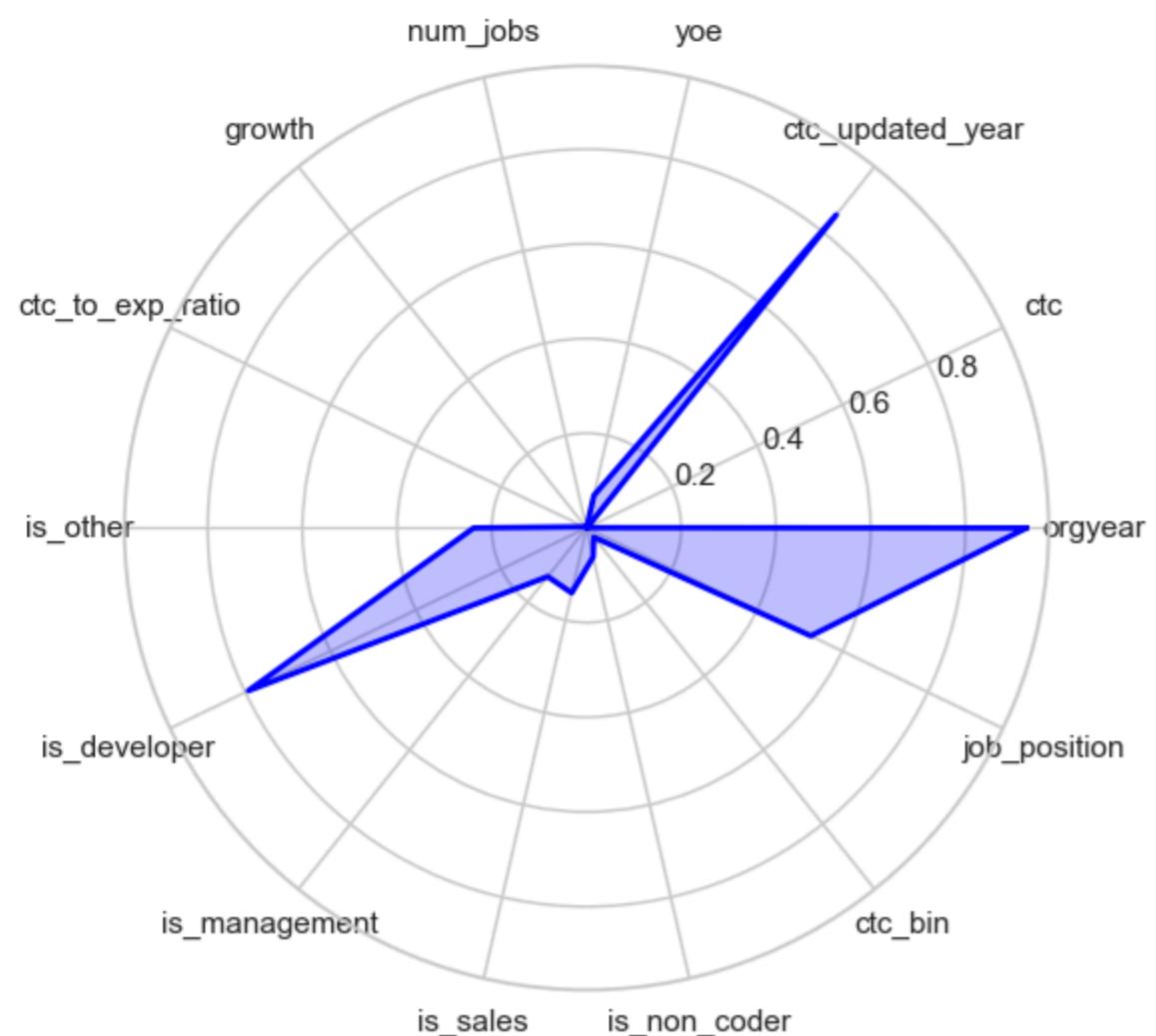
Cluster 3



Cluster 4



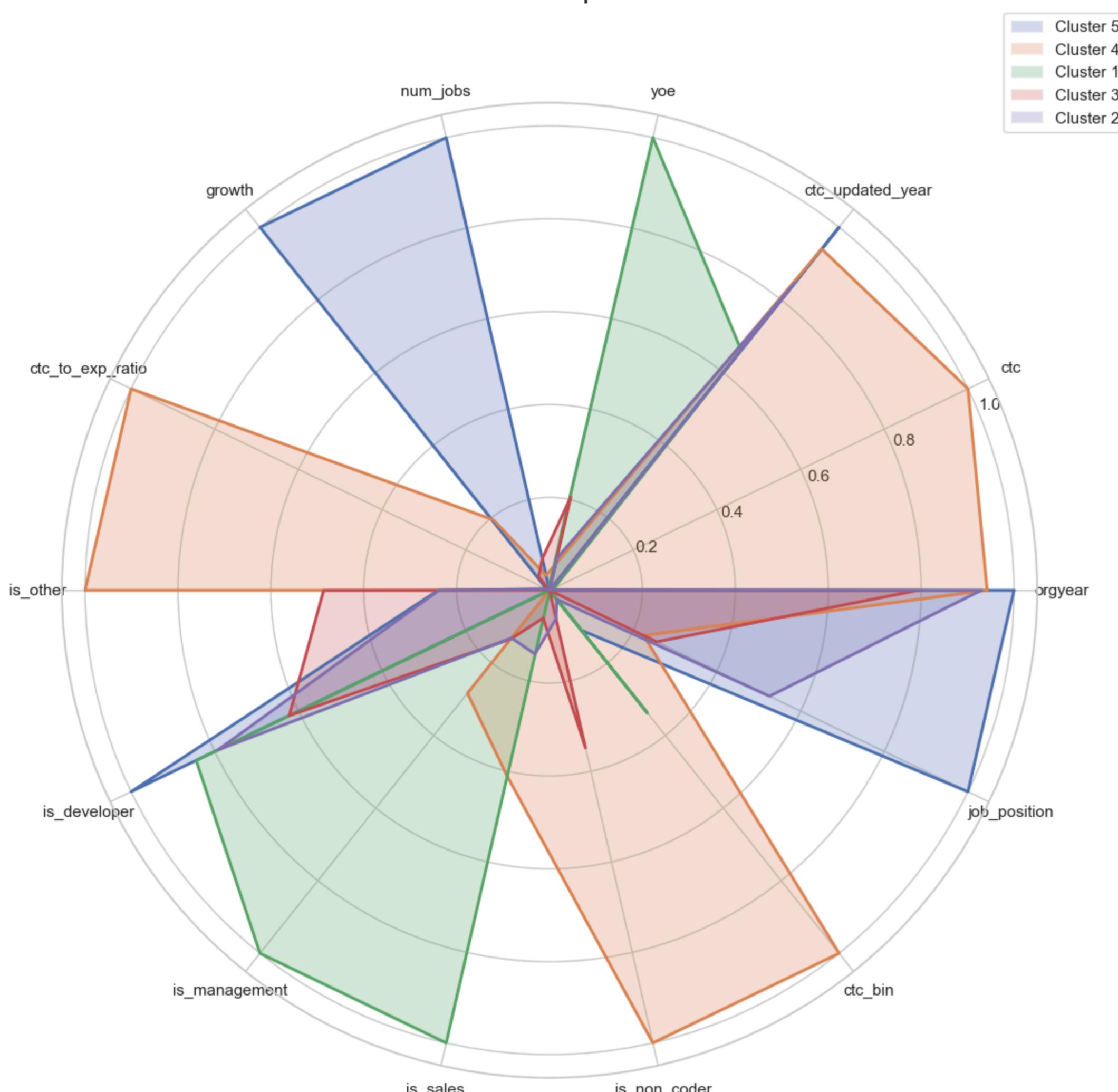
Cluster 5



Combined Polar Plot

```
In [19]: cluster_labels = sample_hc["cluster"].unique()
create_combined_polar_plot(normalized_means, features, cluster_labels)
```

Cluster Comparison



Cluster 1

Candidates in this cluster belong to developers category, and have higher number of job switches.

Cluster 2

Candidates in this cluster belong to the non coder category with high ctc

Cluster 3

Candidates in this cluster have higher experience and belong to management category along with technical skills.

Cluster 4

Candidates in this cluster have lower experience. Also they belong developers category

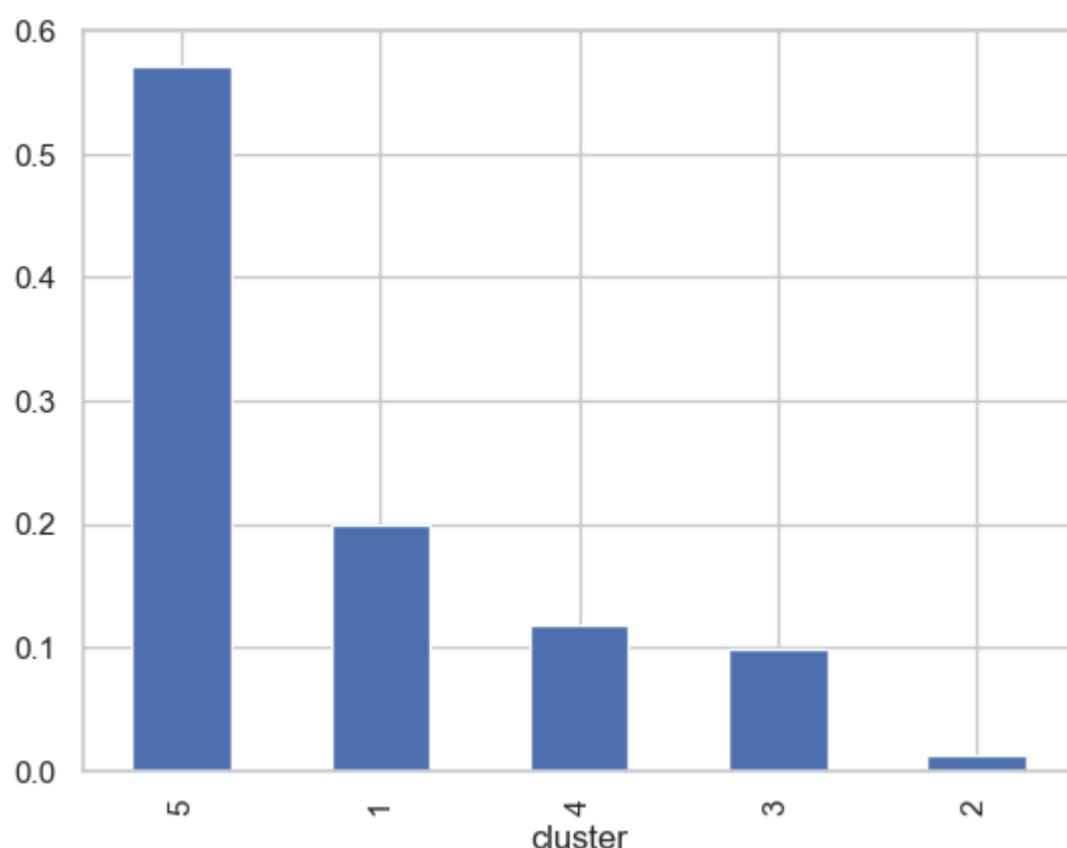
Cluster 5

Candidates in this cluster have recently got hike and belong to developer category

```
In [15]: sample_hc.to_parquet("../data/processed/sample_hc.parquet")
```

```
In [116... sample_hc = pd.read_parquet("../data/processed/sample_hc.parquet")
```

```
In [36]: sample_hc["cluster"].value_counts(normalize=True).plot(kind="bar");
```

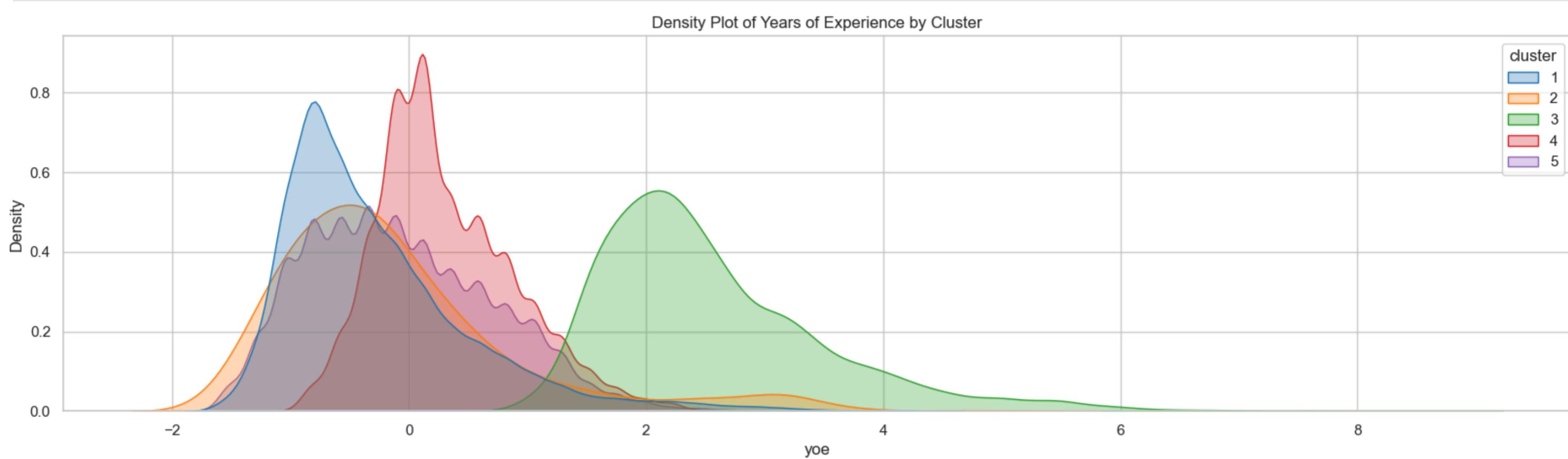


Observations

- From above plot we can see that cluster 5 containing developers has the highest count and cluster 2 containing non coders has the lowest count.
- More than 50% of the candidates belong to cluster 5.

Year of experience density plot for each cluster

```
In [50]: plt.figure(figsize=(20, 5))
sns.kdeplot(data=sample_hc, x="yoe", hue="cluster", cbar=True, fill=True, common_norm=False, alpha=0.3, palette="tab10");
plt.title("Density Plot of Years of Experience by Cluster");
```



Observations

- From above plot we can see that Cluster 3 containing management employees have higher experience

Insights and Recommendations

- There are 153443 candidates 37299 companies in the dataset
- Most of the candidates have under 10 years of experience.
- The majority of the CTC has been updated from 2019 to 2021.
- Most of the candidates joined after 2010
- The job position of "engineering intern" has unusually very high CTC
- Majority of candidates have CTCs between 6 to 16 lakhs
- Most of the candidates belong to the Web development domain specifically backend development
- The mean CTC is around 12 lakhs and the median is around 10 lakhs.
- The backend architect has the highest average CTC.
- QA engineer has decreasing trend in CTC over the years as experience has increased
- In the majority of the cases Data Scientist has higher average CTC than other job positions. But the gap is not that significant.
- Most of the candidates in the dataset belong to the tier 1 flag, indicating that these candidates have higher than average CTC.
- We can see that most Tier 3 candidates do not switch companies.
- There is a huge variance in CTC concerning experience for tier 3 candidates.
- We can also see that sales CTC decreases after 15 years of experience.

The majority of the sales have less than 5 lakhs CTC.

- Developers have a wide range of salaries for different experience levels.
- There are 5 clusters in the dataset.

The majority of the candidates belong to cluster 5 which contains developers amounting to more than 50% of the candidates.

Cluster Insights

Manual Clustering

- Manual clustering and unsupervised clustering had the same cluster shapes when plotted in 2d.

Kmeans Clustering

- Cluster 0**

Candidates in this cluster belong to the management and sales category and have higher experience.

- Cluster 1**

Candidates in this cluster have less experience. Also, they belong developers category and have joined recently.

- **Cluster 2**

Candidates in this cluster belong to the developer category and have less experience

- **Cluster 3**

Candidates in this belong to the developer category and have switched jobs multiple times.

- **Cluster 4**

Candidates in this cluster belong to non-coder category and have higher ctc

Hierarchical Clustering

- **Cluster 1**

Candidates in this cluster belong to the developers category and have a higher number of job switches.

- **Cluster 2**

Candidates in this cluster belong to the non-coder category with high ctc

- **Cluster 3**

Candidates in this cluster have higher experience and belong to the management category along with technical skills.

- **Cluster 4**

Candidates in this cluster have less experience. Also, they belong developers category.

- **Cluster 5**

Candidates in this cluster have recently got a hike and belong to the developer category

Recommendation wrt to candidates

- The company should focus on these candidates as they are the majority.
- The company should push junior candidates towards backend development roles as they have the highest average CTC
- The company should push mid-level candidates towards management roles.
- Alternatively, if mid-level candidates are very good at technical skills, they can be pushed towards technical management roles.
- There are very few non-coders in the dataset which can be pushed to intern roles

Clustering Algorithm Comparison

- Hierarchical clustering seems to be a better fit for this dataset as it is able to capture the clusters better than K-means clustering. The only caveat is that it is slower than K-means clustering.
- If the dataset is large, then K-means clustering is a better fit as it is faster than Hierarchical clustering.

Data Collection Recommendations

- The company has to capture data properly. The CTC is incorrect where candidates have put single-digit CTCs assuming the field to be in lakhs. This has to be corrected by adding validation checks.
- The job position field has to be corrected as there were many spelling mistakes, and in some cases, there were Pune, Jharkhand and Zomato as job positions which is incorrect.
- This can be fixed by providing a dropdown list of job positions to the candidates along with a text field.