

데이터 과학

Group 9

# Term Project

prof / 조풍진

202135708 구준서

202135732 김상준

202235045 박선인

202235105 이하은

202135592 한웅재

# 목차

## CONTENTS

01

### 프로젝트 소개 (Introduction)

프로젝트 주제 및 목적 소개

02

### 데이터셋 개요 (Overview)

데이터 출처 및 선정 이유/ 피처설명

03

### 탐색적 데이터 분석 (EDA)

통계적 특성 분석 / 특성 간 상관관계

04

### 전처리 (Preprocessing)

결측치 및 이상치 처리/스케일링/인코딩

05

### 모델링 (Modeling)

적용 머신러닝 알고리즘/ 하이퍼 파라미터

06

### 학습 경험 (Learning Experience)

프로젝트를 통해 얻은 주요  
인사이트

# 01 프로젝트 소개 (Introduction)

Data Science  
TERM PROJECT

## 분석 데이터

미국 구인공고 웹사이트 Glassdoor에서  
추출한 데이터과학 직업 관련 데이터



### 분석 배경

구직자와 기업 모두에게  
정량적이고 해석 가능한 급여  
분석 필요



### 분석 목표

피쳐 상관관계 해석  
타겟 피쳐 예측  
클러스터링 추천시스템



### 분석 방법

Glassdoor 데이터셋 기반  
EDA → 전처리 → 모델 학습 →  
성능 평가 및 변수 중요도 해석

## 프로젝트

### 목표

산업별 또는 평점별 분포 분석, 급여 정규화 및 클러스터링,  
회귀모델 등의 후속 분석을 통해  
분석 목적에 맞는 결과를 얻고자 합니다.

## 02 데이터셋 개요 (Overview)

출처: Kaggle – Data Science Jobs and Salaries (2024)

<https://www.kaggle.com/datasets/fahadrehman07/data-science-jobs-and-salary-glassdoor/>

선정 이유: 실제 기업 기반의 급여 및 특성 데이터

다양한 기업 속성(산업, 규모, 위치 등) 포함

급여 예측과 기업 특성 분석에 적합한 구조

개요: 956행 x 15열

직책, 예상 급여, 회사 평점, 위치 등 주요 회사 정보 및  
구직 관련 필수적인 정보

### 주요 피처 설명

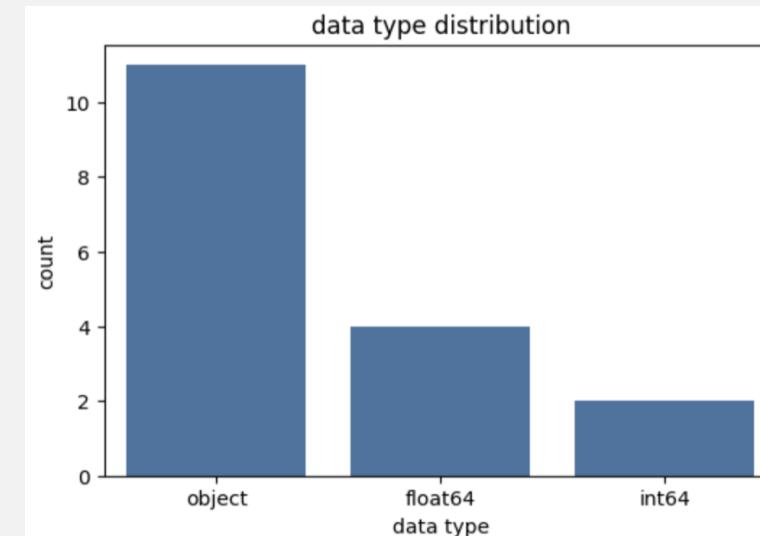
Feature	Description
Job Title	직책명
Salary Estimate	회사가 제공하는 직무에 대한 예상 급여
Job Description	직무 설명
Rating	회사 평가
Company Name	회사명
Location	작업 위치
Headquarters	회사 본사
Size	회사의 직원 수
Founded	회사가 설립된 연도
Type of ownership	민간, 공공, 정부 및 비영리 조직과 같은 소유권 유형
Industry	Aerospace, Energy 등 회사가 서비스를 제공하는 산업 유형
Sector	산업(에너지), 부문(석유, 가스)와 같이 어떤 유형의 서비스를 제공하는지
Revenue	회사의 총 수익
Competitors	회사 경쟁사

# 03 탐색적 데이터 분석 (EDA)

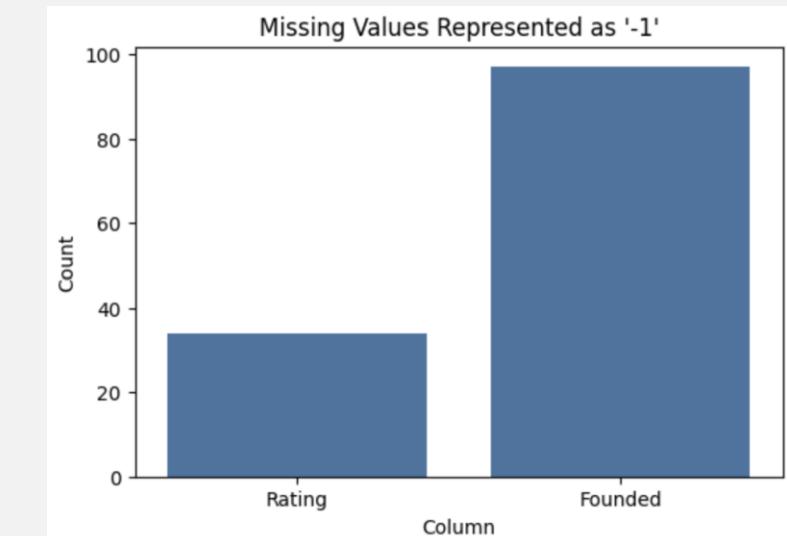
## 통계적 특성

항목	상세 내용
데이터 타입	Object: 12 Int 64: 2 Float 64 : 1
결측치	명시된 null x 수치형 변수에 대해 -1값으로 결측을 표현한 dirty data가 존재 -> 인위적인 drop을 통해 결측값을 추가
고유값	범주형 변수의 고유값 종류가 많음
범주형 변수	차원의 수가 매우 높고 정제되지 않은 형태 -> 다양한 전처리 기법 적용

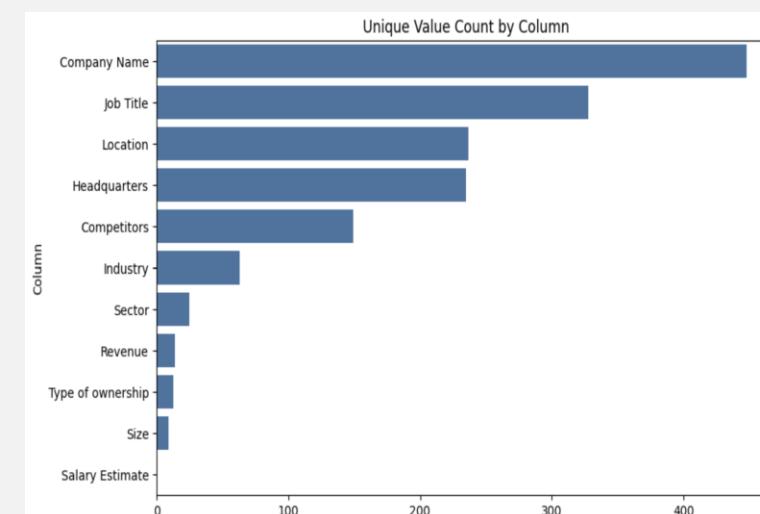
[ 데이터 타입 분포 ]



[ 결측치 현황 ]



[ 고유값 개수 ]



[ 범주형 데이터 ]

Salary Estimate	Job Description
-1	Estimated salary
\$86K-\$143K (Glassdoor est.)	Description of job
Other (736)	77%
596	unique values
\$53K-\$91K (Glassdoor est.)	Data Scientist Location: Albuquerque, NM Education Required: Bachelor's degree required, preferably ...

# 03 탐색적 데이터 분석(EDA)

## Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("glassdoor_jobs.csv")

# 데이터 기본 정보 확인
df.info()
df.describe()

# 결측치 확인
print(df.isnull().sum())

# 고유값 개수 확인
print(df.unique())

# 수치형 컬럼 상관관계 분석
numeric_df = df.select_dtypes(include=['int64', 'float64'])
correlation_matrix = numeric_df.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='rocket', fmt=".2f")
plt.title('Numeric Feature Correlation Matrix')
plt.tight_layout()
plt.show()
```

## [데이터 기본정보]

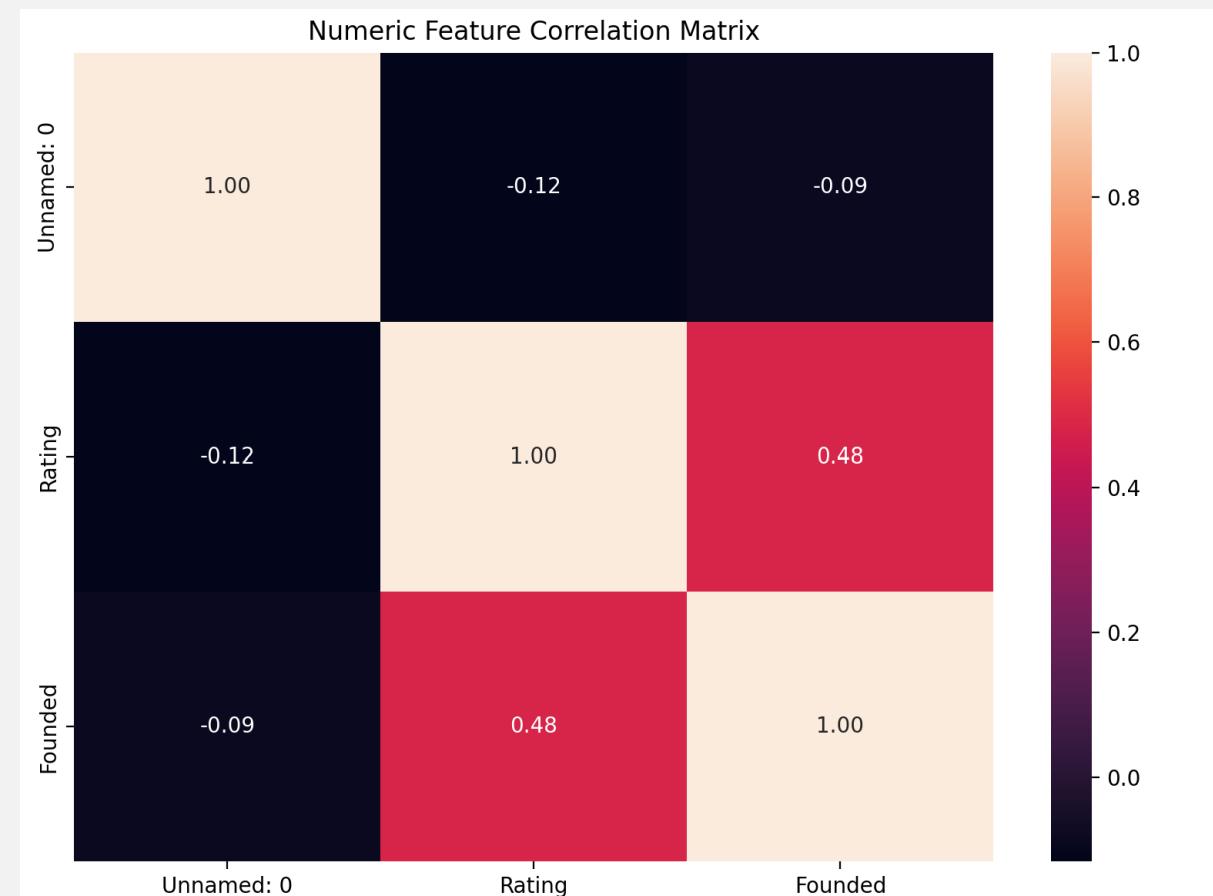
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 956 entries, 0 to 955
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Unnamed: 0        956 non-null    int64  
 1   Job Title         956 non-null    object  
 2   Salary Estimate   956 non-null    object  
 3   Job Description   956 non-null    object  
 4   Rating            956 non-null    float64 
 5   Company Name      956 non-null    object  
 6   Location          956 non-null    object  
 7   Headquarters      956 non-null    object  
 8   Size               956 non-null    object  
 9   Founded            956 non-null    int64  
 10  Type of ownership 956 non-null    object  
 11  Industry           956 non-null    object  
 12  Sector              956 non-null    object  
 13  Revenue             956 non-null    object  
 14  Competitors         956 non-null    object  
dtypes: float64(1), int64(2), object(12)
memory usage: 112.2+ KB
dtype: int64
```

## [고유값]

Unnamed: 0	956
Job Title	328
Salary Estimate	417
Job Description	596
Rating	32
Company Name	448
Location	237
Headquarters	235
Size	9
Founded	109
Type of ownership	13
Industry	63
Sector	25
Revenue	14
Competitors	149

Unnamed: 0	0
Job Title	0
Salary Estimate	0
Job Description	0
Rating	0
Company Name	0
Location	0
Headquarters	0
Size	0
Founded	0
Type of ownership	0
Industry	0
Sector	0
Revenue	0
Competitors	0

## [결측치]



## [수치형 변수 상관관계]

# 04 전처리 (Preprocessing)

## 피처별 전처리 방식

Feature	Description	예시	고유값 수	
Job Title	직책명	Healthcare Data Scientist	328	비닝 필요
Salary Estimate	회사가 제공하는 직무에 대한 예상 급여	\$53K-\$91K (Glassdoor est.)	417	최소 최대 평균 계산
Rating	회사 평가	3.8	-	결측 처리
Location	작업 위치	Albuquerque, NM	237	비닝, 본사근무여부
Headquarters	회사 본사	Goleta, CA	235	
Size	회사의 직원 수	501 to 1000 employees	9	평균 계산
Founded	회사가 설립된 연도	1973	-	회사 연령 계산
Type of ownership	민간, 공공, 정부 및 비영리 조직과 같은 소유권 유형	Company - Private	13	비닝 필요
Industry	Aerospace, Energy 등 회사가 서비스를 제공하는 산업 유형	Banks & Credit Unions	63	상위개념 묶어서 비닝
Sector	산업(에너지), 부문(석유, 가스)와 같이 어떤 유형의 서비스를 제공하는지	Finance	25	
Revenue	회사의 총 수익	\$50 to \$100 million (USD)	14	결측 많음, 최빈, 모델링

# 04 전처리 (Preprocessing)

## Job Title & Revenue code

```
import pandas as pd
import numpy as np
import re
# 데이터 로드
df = pd.read_csv('glassdoor_jobs.csv')
# Job Title 전처리
# 직무 도메인 분류 및 직급 분류
domain_keywords = {
    'Data Science': ['data scientist', 'data science'],
    'Data Engineering': ['data engineer', 'big data engineer', 'etl', 'data platform', 'data systems'],
    'Data Analysis': ['data analyst', 'business intelligence', 'analytics'],
    'Machine Learning': ['machine learning', 'ml engineer', 'deep learning', 'ai', 'artificial intelligence', 'nlp'],
    'Research': ['research scientist', 'research', 'r&d'],
    'Medical/Healthcare': ['medical', 'clinical', 'healthcare', 'pharmacovigilance', 'lab scientist', 'laboratory'],
    'Marketing': ['marketing', 'digital marketing', 'ecommerce'],
    'Software Engineering': ['software engineer', 'developer', 'devops', 'programmer'],
    'Business': ['business', 'consultant', 'account', 'revenue', 'risk', 'insurance', 'manager', 'director', 'executive', 'officer']
}
seniority_keywords = {
    'Chief/Head': ['chief', 'head', 'director', 'officer', 'vp', 'vice president', 'president', 'cxo'],
    'Principal': ['principal', 'distinguished'],
    'Leader': ['leader', 'lead'], # lead 수정 사항 반영
    'Manager': ['manager'],
    'Senior': ['senior', 'sr.', 'sr ', 'sr'],
    'Junior': ['junior', 'jr.', 'jr ', 'associate', 'assistant', 'trainee', 'intern', 'entry'],
    'Staff': ['staff']
}
def classify_domain(title):
    title_lower = str(title).lower()
    for domain, keywords in domain_keywords.items():
        if any(k in title_lower for k in keywords):
            return domain
    return 'Others'
def classify_seniority(title):
    title_lower = str(title).lower()
    for level, keywords in seniority_keywords.items():
        if any(k in title_lower for k in keywords):
            return level
    return 'MidLevel'
df['Job Title'] = df['Job Title'].apply(lambda x: f"{classify_domain(x)} - {classify_seniority(x)}")
# Job Title 열 분리
split_cols = df['Job Title'].str.split(' - ', expand=True)
df['Job Title'] = split_cols[0]
df['Position'] = split_cols[1]
# 오디널 인코딩 (직급)
ordinal_map = {
    'Junior': 1,
    'MidLevel': 2,
    'Staff': 3,
    'Senior': 4,
    'Manager': 5,
    'Leader': 6,
    'Principal': 7
} df['PositionEncoded'] = df['Position'].map(ordinal_map)

# Revenue 전처리 (회귀 모델을 사용해서 revenue 결측치 대체)
# 
def revenue_to_numeric(revenue):
    if isinstance(revenue, str):
        revenue = revenue.lower()
        match = re.match(r'W$(Wd+W.?Wd*)Ws*toWs*W$(Wd+W.?Wd*)Ws*(million|billion)', revenue)
        if match:
            start, end, unit = match.groups()
            start = float(start)
            end = float(end)
            multiplier = 1_000_000 if unit == 'million' else 1_000_000_000
            return int(((start + end) / 2) * multiplier)
        match = re.match(r'W$(Wd+W.?Wd*)W+Ws*billion', revenue)
        if match:
            return int(float(match.group(1)) * 1_000_000_000)
        if 'less than $1 million' in revenue:
            return 500_000
        return np.nan
    df['Revenue'] = df['Revenue'].apply(revenue_to_numeric)

from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor
# Revenue 기준으로 train/predict 나누기
df_train = df[df['Revenue'].notnull()]
df_predict = df[df['Revenue'].isnull()]
X_train = df_train.drop(columns=['Revenue'])
y_train = df_train['Revenue']
X_predict = df_predict.drop(columns=['Revenue'])

# 범주형 및 수치형 열 구분(범주형은 One-Hot Encoding 사용)
categorical_features = X_train.select_dtypes(include=['object']).columns.tolist()
numerical_features = X_train.select_dtypes(include=['float64', 'int']).columns.tolist()

# 전처리 (Revenue를 예측하기 위해서 나머지 열 데이터의 결측치를 평균과 최빈값으로 가정)
preprocessor = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='mean'), numerical_features),
        ('cat', Pipeline([
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
        ]), categorical_features)
    ]
)

# Revenue 결측치를 replace하기 위해 모델 구성 (회귀 사용)
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(random_state=42))
])

# 학습 및 예측
model_pipeline.fit(X_train, y_train)
predicted_revenue = model_pipeline.predict(X_predict)

# 예측값 원본 데이터에 반영
df_result = df.copy()
df_result.loc[df_result['Revenue'].isnull(), 'Revenue'] = predicted_revenue

# 최종으로 Revenue 결측치가 없는지 확인하기 위해 결과 출력
missing_revenue = df_result['Revenue'].isnull().sum()
print(f"Revenue 열의 결측치 개수: {missing_revenue}")
```

## 코드 설명

- 직무/직급 분류: Job Title에서 도메인과 직급 추출 및 인코딩
- Revenue 결측치: RandomForest 회기모델 사용

# 04 전처리 (Preprocessing)

## Salary & Size & Founded code

```
# -----  
# Salary Estimate 전처리  
# -----  
def extract_salary(s):  
    if pd.isnull(s) or '-1' in s:  
        return np.nan, np.nan, np.nan  
  
    s = s.lower()  
    s = s.replace('employer provided salary:', '')  
    s = s.replace('per hour', '')  
    s = re.sub(r'W(.?W)', ' ', s) # 괄호 안 제거  
    s = s.replace('$', '').replace('k', '').strip()  
  
    try:  
        min_sal, max_sal = map(int, s.split('-'))  
        avg_sal = (min_sal + max_sal) / 2  
        return min_sal, max_sal, avg_sal  
    except:  
        return np.nan, np.nan, np.nan  
  
# Apply to Salary Estimate  
df[["min_salary", "max_salary", "avg_salary"]] = df["Salary Estimate"].apply(  
    lambda x: pd.Series(extract_salary(x)))  
  
# -----  
# Size 전처리 (원하는 그룹 기준 적용)  
# -----  
def map_size(size_str):  
    if pd.isnull(size_str) or size_str in ['-1', 'Unknown']:  
        return np.nan  
    size_str = size_str.strip()  
    if size_str in ['0 to 50 employees', '51 to 200 employees']:  
        return 'Small'  
    elif size_str in ['201 to 500 employees', '501 to 1000 employees']:  
        return 'Medium'  
    elif size_str in ['1001 to 5000 employees', '5001 to 10000 employees']:  
        return 'Large'  
    elif size_str == '10000+ employees':  
        return 'Very Large'  
    else:  
        return np.nan  
  
df["Size_cleaned"] = df["Size"].apply(map_size)  
  
# -----  
# Founded 전처리 (회사 나이 계산)  
# -----  
df["Founded_cleaned"] = df["Founded"].apply(lambda x: x if x > 1800 else np.nan)  
df["Company_age"] = df["Founded_cleaned"].apply(lambda x: 2025 - x if pd.notnull(x) else np.nan)
```

### 코드 설명

- 최소·최대·평균 연봉 추출
- 직원 수 범주로 그룹화  
(Small ~ Very Large)
- 설립연도로 회사 나이 계산

# 04 전처리 (Preprocessing)

## Location & Headquarters code

```
# Location & Headquarters 전처리
# 임의로 drop해서 missing data 만들기
np.random.seed(42)

# location 컬럼에서 임의의 15개 인덱스를 골라 nan으로 설정
location_missing_idx = np.random.choice(df.index, size=15, replace=False)
df.loc[location_missing_idx, 'Location'] = np.nan

# headquarters 컬럼에서도 임의의 15개 인덱스를 골라 nan으로 설정
headquarters_missing_idx = np.random.choice(df.index, size=15, replace=False)
df.loc[headquarters_missing_idx, 'Headquarters'] = np.nan

#dirty data 정제
def clean_location_col(col):
    return (col
            .str.strip() # 공백 제거
            .str.lower() # 소문자 통일
            .str.replace(' ', ' ', regex=False) # 쉼표 양쪽 공백 제거
            .str.replace(' ', ' ', regex=False)) # 쉼표 뒤에만 공백 붙이기

df['Location'] = clean_location_col(df['Location'])
df['Headquarters'] = clean_location_col(df['Headquarters'])

# missing data 채우기 (최빈값 사용)
# 각 컬럼의 최빈값 계산
mode_location = df['Location'].mode()[0]
mode_headquarters = df['Headquarters'].mode()[0]

# 결측치 채우기
df['Location'] = df['Location'].fillna(mode_location)
df['Headquarters'] = df['Headquarters'].fillna(mode_headquarters)

# 본사근무 여부 피처 생성 (state 기준)
# location에서 state 추출
df['Location_state'] = df['Location'].apply(lambda x: x.split(',')[1].strip() if ',' in x else 'Unknown')

# headquarters에서 state 추출
df['Headquarters_state'] = df['Headquarters'].apply(lambda x: x.split(',')[1].strip() if ',' in x else 'Unknown')

# state 기준으로 binning (상위 6개 + 나머지는 'Other'로)
top5_location_state = df['Location_state'].value_counts().nlargest(6).index.tolist()
top5_headquarters_state = df['Headquarters_state'].value_counts().nlargest(6).index.tolist()

df['Location_state_binned'] = df['Location_state'].apply(lambda x: x if x in top5_location_state else 'Other')
df['Headquarters_state_binned'] = df['Headquarters_state'].apply(lambda x: x if x in top5_headquarters_state else 'Other')

# location과 headquarters가 같으면 본사 근무 (Yes), 아니면 No
df['works_at_headquarters'] = df.apply(
    lambda row: 'Yes' if row['Location'] == row['Headquarters'] else 'No', axis=1
)

# Binning: region 기준으로 전처리
region_map = {
    'Northeast': ['ny', 'nj', 'ma', 'pa', 'ct', 'ri', 'vt', 'nh', 'me'],
    'Midwest': ['il', 'oh', 'mi', 'wi', 'in', 'mo', 'mn', 'ia', 'ks', 'ne'],
    'South': ['tx', 'fl', 'ga', 'nc', 'va', 'tn', 'al', 'sc', 'la', 'ky', 'ok', 'ms', 'ar'],
    'West': ['ca', 'wa', 'or', 'co', 'az', 'nv', 'ut', 'nm', 'id', 'mt', 'wy', 'ak', 'hi']
}

def map_state_to_region(state):
    for region, states in region_map.items():
        if state in states:
            return region
    return 'Other' # 알 수 없거나 희귀한 state

# location_state, headquarters_state → region
df['Location_region'] = df['Location_state'].apply(map_state_to_region)
df['Headquarters_region'] = df['Headquarters_state'].apply(map_state_to_region)
```

## 코드 설명

- 결측치 임의로 생성 후 정제  
-> 최빈값으로 대체
- 상위 6개 주만 추출해 그룹화
- 주를 4개 권역으로 맵핑
- 본사 근무 여부:  
**Location=Headquarters-> "Yes"**

# 04 전처리 (Preprocessing)

## Industry & Sector code

```
# _____  
# Industry & Sector 전처리  
# _____  
  
# "-1" 표기를 NaN으로 변환  
df.replace({'Industry': {'-1': np.nan},  
           'Sector' : {'-1': np.nan}},  
          inplace=True)  
  
# 회사별 Mode 계산, 회사별로 존재하는 공고의 최빈값.  
ind_mode_by_co = (  
    df.groupby('Company Name')['Industry']  
        .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan))  
sec_mode_by_co = (  
    df.groupby('Company Name')['Sector']  
        .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan))  
  
# 회사별 Mode로 결측치 채우기 (기존 컬럼 덮어쓰기)  
df['Industry'] = df.apply(  
    lambda r: ind_mode_by_co[r['Company Name']]  
        if pd.isna(r['Industry']) else r['Industry'],  
    axis=1)  
df['Sector'] = df.apply(  
    lambda r: sec_mode_by_co[r['Company Name']]  
        if pd.isna(r['Sector']) else r['Sector'],  
    axis=1)  
  
# 남은 NaN → 전역 Mode(최빈값)로 채우기, 만약 회사별 공고가 없을 경우 모든 공고의 최빈값으로 채움  
global_ind_mode = df['Industry'].mode()[0]  
global_sec_mode = df['Sector'].mode()[0]  
df.fillna({'Industry': global_ind_mode,  
           'Sector' : global_sec_mode},  
          inplace=True)
```

```
# 6) Sector를 6개 카테고리로 맵핑 (기존 컬럼 덮어쓰기)  
sector_mapping = {  
    # 1. 금융 · 컨설팅  
    'Finance': 'Finance & Consulting',  
    'Insurance': 'Finance & Consulting',  
    'Accounting & Legal': 'Finance & Consulting',  
    'Business Services': 'Finance & Consulting',  
    # 2. IT · 테크  
    'Information Technology': 'Technology',  
    'Telecommunications': 'Technology',  
    # 3. 헬스케어 · 교육  
    'Health Care': 'Healthcare & Education',  
    'Biotech & Pharmaceuticals': 'Healthcare & Education',  
    'Education': 'Healthcare & Education',  
    # 4. 소비재 · 리테일  
    'Retail': 'Consumer & Retail',  
    'Consumer Services': 'Consumer & Retail',  
    # 5. 산업 · 에너지  
    'Manufacturing': 'Industrial & Energy',  
    'Oil, Gas, Energy & Utilities': 'Industrial & Energy',  
    'Mining & Metals': 'Industrial & Energy',  
    'Construction, Repair & Maintenance': 'Industrial & Energy',  
    'Aerospace & Defense': 'Industrial & Energy',  
    'Transportation & Logistics': 'Industrial & Energy',  
    # 6. 기타 서비스  
    'Media': 'Other Services',  
    'Real Estate': 'Other Services',  
    'Government': 'Other Services',  
    'Non-Profit': 'Other Services',  
    'Arts, Entertainment & Recreation': 'Other Services',  
    'Agriculture & Forestry': 'Other Services',  
    'Travel & Tourism': 'Other Services'  
}  
  
df['Sector'] = df['Sector'].map(sector_mapping).fillna('Other Services')
```

## 코드 설명

- 결측치 처리: '-1'을 NaN으로 변환 후 회사별 최빈값 → 전체 최빈값 순으로 채움
- 카테고리 통합: 'Sector' 값을 6개 산업군으로 그룹화
- 결측 여부 및 통합 결과 출력

# 04 전처리 (Preprocessing)

## Type of Ownership & Rating code

```

# _____
# Type of Ownership 전처리
# _____

# 최빈값 계산 (결측 대체용)
most_common_ownership = df['Type of ownership'].mode()[0]

# 전처리 함수 정의
def classify_ownership(value):
    # - "-1", "Unknown", "Other Organization"은 실제 기업 구조 파악 불가
    # - 단순히 제거하거나 Others로 둘 경우 분석/모델링에서 해석성 및 예측 안정성 저하
    # - 최빈값으로 대체하여 데이터 누락 없이 편입
    if value in ["-1", "Unknown", "Other Organization"]:
        value = most_common_ownership

    # Private 그룹
    # - "Company - Private": 사기업
    # - "Subsidiary or Business Segment": 대부분 상위 사기업 소속
    # - "Contract": 사기업 중심의 고용 형태
    # - "Private Practice / Firm": 소규모 독립 사기업으로 판단
    # 따라서 이들은 "Private"으로 통합
    if value in [
        "Company - Private", "Subsidiary or Business Segment",
        "Contract", "Private Practice / Firm"
    ]:
        return "Private"

    # Public 그룹
    # - "Company - Public": 상장 대기업 또는 주식 공개 기업
    # 독립적인 경제적 규모와 해석을 위해 그룹 유지
    elif value == "Company - Public":
        return "Public"

    # Public Service 그룹
    # - "Nonprofit Organization": 공익 목적
    # - "Government": 공공 부문
    # - "Hospital": 보건 서비스
    # - "College / University", "School / School District": 교육 기관
    # 모두 공공 목적 중심으로, 유사한 고용 특성 고려
    elif value in [
        "Nonprofit Organization", "Government",
        "Hospital", "College / University", "School / School District"
    ]:
        return "Public Service"

    # 새 피처 생성
    df['Ownership_Grouped'] = df['Type of ownership'].apply(classify_ownership)

```

```

# _____
# Rating 전처리
# _____

# -1.0을 결측치로 간주하고 평균값으로 대체, 분포가 왜곡되지 않았으므로 안전
df['Rating'] = df['Rating'].replace(-1.0, np.nan)
rating_mean = df['Rating'].mean(skipna=True)
df['Rating'] = df['Rating'].fillna(rating_mean)

```

### 코드 설명

- 결측값을 최빈값으로 대체
- 기업 소유 형태를 3개 그룹으로 단순화  
(Private, Public, Public Service )
- Rating: -1을 결측 처리 후 평균값으로 대체

# 04 전처리 (Preprocessing)

## Final result

```
# -----  
# 사용할 피쳐 리스트 정의 (미사용 피쳐는 주석 처리)  
  
final_features = [  
    'Job_Title', # ✓ Position 분리 후 직무명  
    'Position_Encoded', # ✓ 오디널 인코딩된 직급  
    # 'Position', # → Position_Encoded로 대체됨  
  
    'min_salary', # ✓ 수치형 최저 연봉  
    'max_salary', # ✓ 수치형 최고 연봉  
    'avg_salary', # ✓ 수치형 평균 연봉  
    # 'Salary_Estimate', # → 위 세 변수로 분해됨  
  
    'Rating', # ✓ 결측 보완된 수치형 평점  
  
    'Location_state_binned', # ✓ 정제된 지역 (상위 6개 + 기타)  
    'Headquarters_state_binned', # ✓ 정제된 본사 지역 (상위 6개 + 기타)  
    'works_at_headquarters', # ✓ 본사 근무 여부 (Yes/No)  
    #'Location_region', # → 정제된 지역 (상위 6개 + 기타)  
    #'Headquarters_region', # → 정제된 본사 지역 (상위 6개 + 기타)  
    #'Location', # → 정제 후 위 컬럼들로 분해됨  
    #'Headquarters', # → 동일  
    #'Location_state', # → 동일  
    #'Headquarters_state', # → 동일  
  
    'Size_cleaned', # ✓ 정제된 회사 규모  
    # 'Size', # → Size_cleaned로 대체됨  
  
    'Company_age', # ✓ 회사 나이 (2025 - Founded)  
    # 'Founded_cleaned', # → 위로 대체됨  
    # 'Founded', # → 동일  
  
    'Ownership_Grouped', # ✓ 정제된 소속 그룹  
    # 'Type_of_ownership', # → Ownership_Grouped로 대체됨  
  
    'Industry', # ✓ 산업군 (세분화된 업종)  
    'Sector', # ✓ 6개 대분류 부문 그룹  
  
    #--! 결측 처리 모델링 필요!--  
    'Revenue', # ✓ 수치형 매출 규모  
]  
  
df[final_features].to_csv("glassdoor_cleaned.csv", index=False)  
print("전처리된 주요 피쳐만 포함하여 저장 완료.")
```

## 코드 설명

- 모델링에 사용할 주요 피쳐 리스트 정의
- 불필요하거나 중복된 원본 컬럼은 주석 처리
- 정제·가공된 컬럼만 피처로 데이터프레임을 저장

# 05 모델링 (Modeling)

## 회귀 모델을 통한 기업의 규모, 업종, 소유 형태, 위치 등과 급여의 상관관계 분석

```
# 라이브러리 import
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, KFold, GridSearchCV, RandomizedSearchCV, cross_val_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 데이터 로드 및 'avg_salary' 결측치 제거
df = pd.read_csv("glassdoor_cleaned_final.csv")
df = df.dropna(subset=['avg_salary'])

"""# 2. 데이터 탐색 (EDA)"""
# 각 범주형 변수에 따른 평균 급여 출력
print(df.groupby('Size_cleaned')['avg_salary'].mean())
print(df.groupby('Ownership_Grouped')['avg_salary'].mean())
print(df.groupby('Industry')['avg_salary'].mean().sort_values(ascending=False).head(10))

# 수치형 변수 간의 상관관계 시각화
sns.heatmap(df[['Company_age', 'Rating', 'avg_salary']].corr(), annot=True)
plt.title("수치형 변수 간 상관관계")
plt.show()

# 기업 규모별 급여 분포 시각화
sns.boxplot(data=df, x='Size_cleaned', y='avg_salary')
plt.xticks(rotation=45)
plt.title("기업 규모별 급여 분포")
plt.show()

# 본사 위치에 따른 급여 비교 (상위 10개 위치 기준)
top_states = df['Headquarters_state_binned'].value_counts().head(10).index
sns.boxplot(data=df[df['Headquarters_state_binned'].isin(top_states)], x='Headquarters_state_binned', y='avg_salary')
plt.xticks(rotation=45)
plt.title("본사 위치에 따른 급여 분포")
plt.show()

# 산업별 기업 나이와 급여의 관계 (다변량 시각화)
sns.lmplot(data=df, x='Company_age', y='avg_salary', hue='Sector', fit_reg=False)
plt.title("기업 연령과 급여 - 산업별 분포")
plt.show()

"""# 3. 기초 피처 엔지니어링
결측치 처리, 스케일링 및 인코딩"""

# 분석에 사용할 feature 및 target 설정
features = ['Size_cleaned', 'Ownership_Grouped', 'Industry', 'Sector',
            'Location_state_binned', 'Headquarters_state_binned',
            'works_at_headquarters', 'Company_age', 'Rating']
target = 'avg_salary'

X = df[features]
y = df[target]

# 수치형, 범주형 변수 구분
numeric_features = ['Company_age', 'Rating']
categorical_features = list(set(features) - set(numeric_features))

# 수치형 변수 전처리: 결측치 평균 대체 후 표준화
num_imputer = SimpleImputer(strategy='mean')
num_scaler = StandardScaler()
X_num = num_scaler.fit_transform(num_imputer.fit_transform(X[numeric_features]))

# 범주형 변수 전처리: 결측치 최빈값 대체 후 one-hot 인코딩
cat_imputer = SimpleImputer(strategy='most_frequent')
X_cat_imputed = cat_imputer.fit_transform(X[categorical_features])
ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
X_cat = ohe.fit_transform(X_cat_imputed)

# 수치형 + 범주형 병합
from numpy import hstack
X_processed = hstack([X_num, X_cat])
```

```
"""# 4. 회기 모델 학습
- Random Forest
- Xgboost
- Gradient boosting

회기모델 학습 및 평가 함수 정의 (kfold cross validation : 평가지표는 mae, mse, rscore 활용)
"""

# 모델 성능 평가 함수 정의 (KFold 적용, MAE, MSE, RMSE, R2 계산)
def evaluate_model(model, X, y):
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    mae = -cross_val_score(model, X, y, cv=kf, scoring='neg_mean_absolute_error').mean()
    mse = -cross_val_score(model, X, y, cv=kf, scoring='neg_mean_squared_error').mean()
    rmse = np.sqrt(mse)
    r2 = cross_val_score(model, X, y, cv=kf, scoring='r2').mean()
    return mae, mse, rmse, r2

# 사용 모델 정의
models = {
    'RandomForest': RandomForestRegressor(random_state=42),
    'XGBoost': XGBRegressor(random_state=42, verbosity=0),
    'GradientBoosting': GradientBoostingRegressor(random_state=42)
}

# 각 모델에 대해 성능 평가
for name, model in models.items():
    mae, mse, rmse, r2 = evaluate_model(model, X_processed, y)
    print(f"{name} -> MAE: {mae:.2f}, MSE: {mse:.2f}, RMSE: {rmse:.2f}, R2: {r2:.2f}")

"""# 5. 하이퍼파라미터 튜닝
- gridsearch
- randomsearch

# randomforest 하이퍼파라미터 튜닝 (GridSearchCV)
rfr = RandomForestRegressor(random_state=42)
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
}
gs = GridSearchCV(rfr, param_grid, cv=5, scoring='neg_mean_absolute_error')
gs.fit(X_processed, y)
print("Best RF Params:", gs.best_params_)

# XGBoost 하이퍼파라미터 튜닝 (RandomizedSearchCV)
xgb = XGBRegressor(random_state=42, verbosity=0)
rnd_params = {
    'n_estimators': [50, 100],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}
rs = RandomizedSearchCV(xgb, rnd_params, n_iter=5, cv=5, scoring='neg_mean_squared_error')
rs.fit(X_processed, y)
print("Best XGB Params:", rs.best_params_)
```

```
"""# 6. 상관관계 해석
특성 중요도, 기업 특성(규모, 업종, 소유 형태, 위치 등)과 급여와의 연관성 시각화
"""

# 최적 하이퍼파라미터로 모델 학습 후 feature importance 확인
final_model = RandomForestRegressor(**gs.best_params_, random_state=42)
final_model.fit(X_processed, y)

# 특성 중요도 데이터프레임 생성
importances = final_model.feature_importances_
feature_names = numeric_features + list(ohe.get_feature_names_out(categorical_features))
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# 중요 변수 상위 15개 시각화
sns.barplot(data=importance_df.head(15), x='Importance', y='Feature')
plt.title("급여에 영향을 미치는 주요 기업 특성")
plt.show()
```

## 분석 목표

- 1. 주요 피쳐들의 상관관계 해석
- 2. 여러가지 회귀 예측 모델을 통해
- 기업 특성별 급여 영향도 파악

# 05 모델링 (Modeling)

## 회귀 모델을 통한 기업의 규모, 업종, 소유 형태, 위치 등과 급여의 상관관계 분석

```
# 데이터 로드 및 'avg_salary' 결측치 제거
df = pd.read_csv("glassdoor_cleaned_final.csv")
df = df.dropna(subset=['avg_salary'])

"""# 2. 데이터 탐색 (EDA)"""

# 각 범주형 변수에 따른 평균 급여 출력
print(df.groupby('Size_cleaned')['avg_salary'].mean())
print(df.groupby('Ownership_Grouped')['avg_salary'].mean())
print(df.groupby('Industry')['avg_salary'].mean().sort_values(ascending=False).head(10))

# 수치형 변수 간의 상관관계 시각화
sns.heatmap(df[['Company_age', 'Rating', 'avg_salary']].corr(), annot=True)
plt.title("수치형 변수 간 상관관계")
plt.show()

# 기업 규모별 급여 분포 시각화
sns.boxplot(data=df, x='Size_cleaned', y='avg_salary')
plt.xticks(rotation=45)
plt.title("기업 규모별 급여 분포")
plt.show()

# 본사 위치에 따른 급여 분포 비교 (상위 10개 위치 기준)
top_states = df['Headquarters_state_binned'].value_counts().head(10).index
sns.boxplot(data=df[df['Headquarters_state_binned'].isin(top_states)],
            x='Headquarters_state_binned', y='avg_salary')
plt.xticks(rotation=45)
plt.title("본사 위치에 따른 급여 분포")
plt.show()

# 산업별 기업 나이와 급여의 관계 (다변량 시각화)
sns.lmplot(data=df, x='Company_age', y='avg_salary', hue='Sector', fit_reg=False)
plt.title("기업 연령과 급여 - 산업별 분포")
plt.show()
```

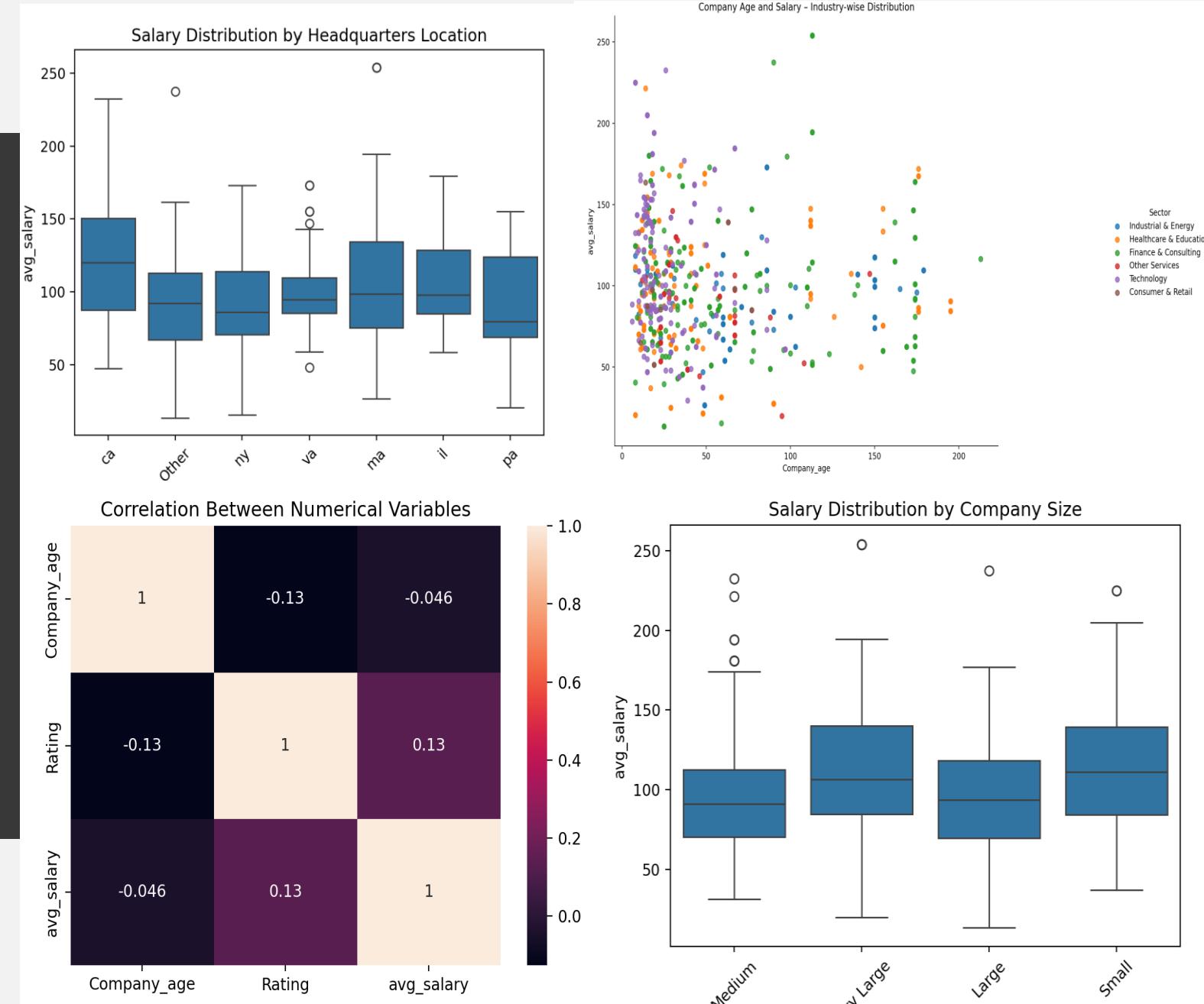
### 데이터 탐색(EDA)

- 범주형 변수별 평균 급여 분석
- 수치형 변수 간 상관관계 분석
- 급여 분포 시각화
- 산업군별 기업 나이 vs 급여 관계 분석

# 05 모델링 (Modeling)

## 회귀 모델을 통한 기업의 규모, 업종, 소유 형태, 위치 등과 급여의 상관관계 분석

```
Size_cleaned
Large      94.139381
Medium     95.179283
Small      111.351064
Very Large 112.230769
Name: avg_salary, dtype: float64
Ownership_Grouped
Private    102.177060
Public     110.893782
Public Service 73.845000
Name: avg_salary, dtype: float64
Industry
Other Retail Stores      163.500000
Motion Picture Production & Distribution 146.000000
Financial Analytics & Research 145.125000
Health, Beauty, & Fitness 139.500000
Telecommunications Services 131.500000
Brokerage Services        129.000000
Auctions & Galleries       128.000000
Internet                 123.810345
Investment Banking & Asset Management 118.400000
TV Broadcast & Cable Networks 117.750000
Name: avg_salary, dtype: float64
```



### 분석 결과

- 기업 규모가 클수록 평균 급여가 높은 경향
- 소유 형태에 따라 급여 차이 존재
- 산업군별 급여 편차 큼
- Rating, avg\_salary ->약한 양의 상관관계
- Company\_age, 급여와 큰 상관관계 없음
- 본사 위치별 급여 차이 존재
- 산업군별로 기업 나이와 급여의 관계가 다르게 나타남

# 05 모델링 (Modeling)

회귀 모델을 통한 기업의 규모, 업종, 소유 형태, 위치 등과 급여의 상관관계 분석

```
# 수치형, 범주형 변수 구분
numerical_features = ['Company_age', 'Revenue', 'Employees', 'Industry', 'Ownership', 'Location']
categorical_features = list(set(features) - set(numerical_features))

# 수치형 변수 전처리: 결측치 평균
num_imputer = SimpleImputer(strategy='mean')
num_scaler = StandardScaler()
X_num = num_scaler.fit_transform(X[numerical_features])

# 범주형 변수 전처리: 결측치 최빈값
cat_imputer = SimpleImputer(strategy='most_frequent')
X_cat_imputed = cat_imputer.fit_transform(X[categorical_features])
ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
X_cat = ohe.fit_transform(X_cat_imputed)

# 수치형 + 범주형 병합
from numpy import hstack
X_processed = hstack([X_num, X_cat])
```

범주형 변수 희소성 문제  
→ 범주 수가 많아 고차원 희소 행렬이 생성되었고, binning과  
handle\_unknown='ignore' 옵션으로 안정화함.

## 전처리 (Preprocessing)

- 수치형 변수: 결측치 평균/표준화(StandardScaler)
- 범주형 변수: 결측치 최빈값/ 원핫 인코딩
- numpy.hstack으로 가로 방향으로 결합  
→ 모델 입력 행렬 X\_processed 생성

# 05 모델링 (Modeling)

회귀 모델을 통한 기업의 규모, 업종, 소유 형태, 위치 등과 급여의 상관관계 분석

```
# 모델 성능 평가 함수 정의 (KFold 적용, MAE, MSE, RMSE, R2 계산)
def evaluate_model(model, X, y):
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    mae = -cross_val_score(model, X, y, cv=kf, scoring='neg_mean_absolute_error').mean()
    mse = -cross_val_score(model, X, y, cv=kf, scoring='neg_mean_squared_error').mean()
    rmse = np.sqrt(mse)
    r2 = cross_val_score(model, X, y, cv=kf, scoring='r2').mean()
    return mae, mse, rmse, r2

# 사용 모델 정의
models = {
    'RandomForest': RandomForestRegressor(random_state=42),
    'XGBoost': XGBRegressor(random_state=42, verbosity=0),
    'GradientBoosting': GradientBoostingRegressor(random_state=42)
}

# 각 모델에 대해 성능 평가
for name, model in models.items():
    mae, mse, rmse, r2 = evaluate_model(model, X_processed, y)
    print(f"{name} → MAE: {mae:.2f}, MSE: {mse:.2f}, RMSE: {rmse:.2f}, R2: {r2:.2f}")

RandomForest → MAE: 18.48, MSE: 685.38, RMSE: 26.18, R2: 0.54
XGBoost → MAE: 17.55, MSE: 728.32, RMSE: 26.99, R2: 0.51
GradientBoosting → MAE: 23.07, MSE: 896.22, RMSE: 29.94, R2: 0.40
```

## 회귀 모델 학습 및 성능 평가

- **RandomForest** → 가장 높은 설명력과 낮은 오차 (가장 우수한 성능)
- **XGBoost** → MAE는 가장 낮고 MSE/RMSE가 높아 전체적인 안정성은 다소 떨어짐
- **GradientBoosting** → 모든 지표에서 가장 낮은 성능 (데이터셋에 적합하지 않은 모델로 판단)

# 05 모델링 (Modeling)

회귀 모델을 통한 기업의 규모, 업종, 소유 형태, 위치 등과 급여의 상관관계 분석

```
# randomforest 하이퍼파라미터 튜닝 (GridSearchCV)
rfr = RandomForestRegressor(random_state=42)
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
}
gs = GridSearchCV(rfr, param_grid, cv=5, scoring='neg_mean_absolute_error')
gs.fit(X_processed, y)
print("Best RF Params:", gs.best_params_)

# XGBoost 하이퍼파라미터 튜닝 (RandomizedSearchCV)
xgb = XGBRegressor(random_state=42, verbosity=0)
rnd_params = {
    'n_estimators': [50, 100],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}
rs = RandomizedSearchCV(xgb, rnd_params, n_iter=5, cv=5, scoring='neg_mean_squared_error')
rs.fit(X_processed, y)
print("Best XGB Params:", rs.best_params_)

Best RF Params: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Best XGB Params: {'n_estimators': 100, 'max_depth': 7, 'learning_rate': 0.05}
```

## 하이퍼 파라미터 튜닝 및 최적의 파라미터 탐색

- 모델 모두 **n\_estimators=100** -> best
- RandomForest** : **max\_depth=None / 분할 최소 샘플 수= 2** -> best
- XGBoost** : **max\_depth=7/ 학습률 0.05-> best**

# 05 모델링 (Modeling)

## 회귀 모델을 통한 기업의 규모, 업종, 소유 형태, 위치 등과 급여의 상관관계 분석

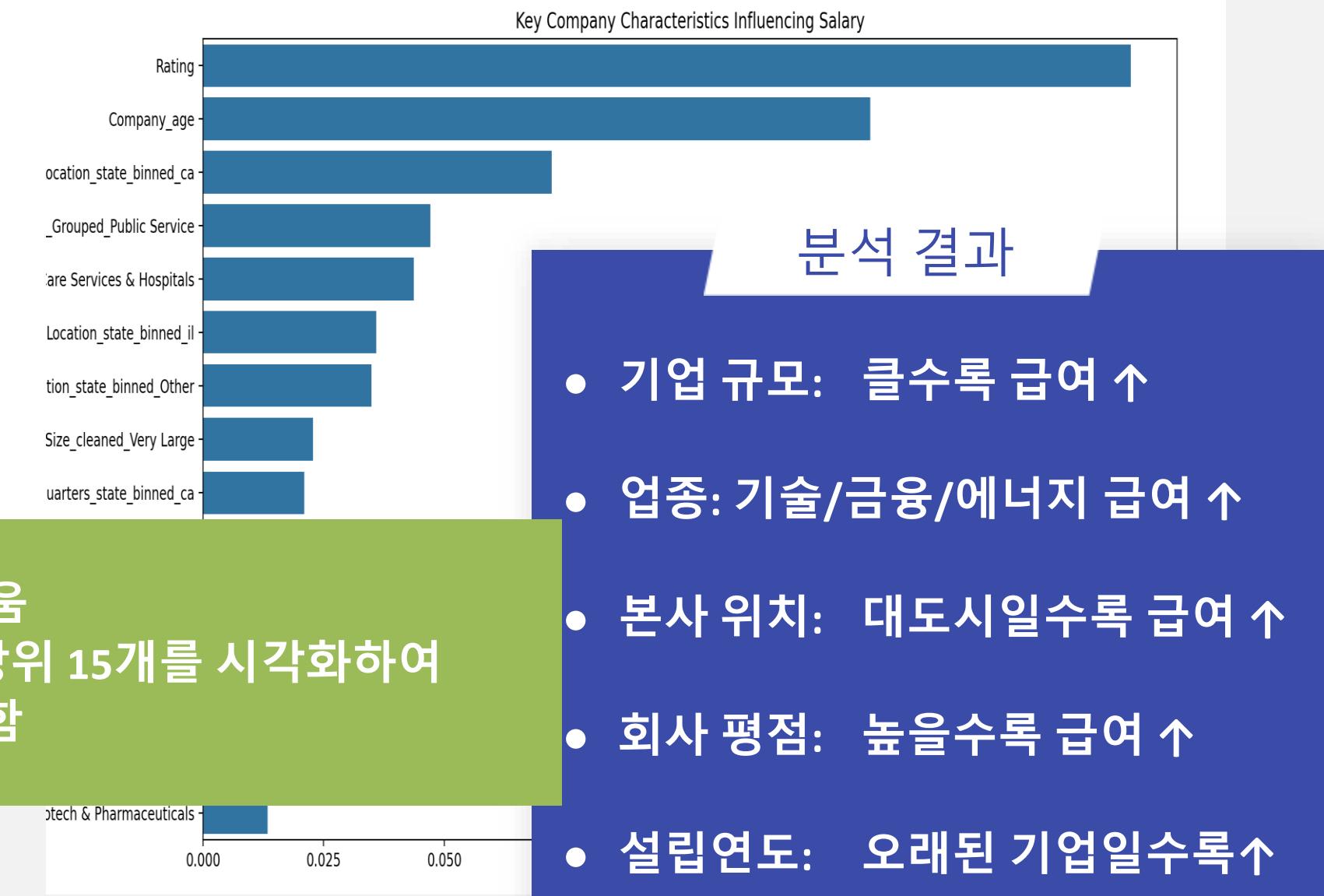
```
'''# 6. 상관관계 해석
특성 중요도, 기업 특성(규모, 업종, 소유 형태, 위치 등)과 급여와의 연관성 시각화'''
```

```
# 최적 하이퍼파라미터로 모델 학습 후 feature importance 확인
final_model = RandomForestRegressor(**gs.best_params_, random_state=42)
final_model.fit(X_processed, y)
```

```
# 특성 중요도 데이터프레임 생성
importances = final_model.feature_importances_
feature_names = numeric_features + list(ohe.get_feature_names_out(categorical_features))
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)
```

```
# 중요 변수 상위 15개 시각화
sns.barplot(data=importance_df.head(15))
plt.title("급여에 영향을 미치는 주요")
plt.show()
```

특성 중요도 프레임을 이용한 결과 시각화



모델 해석 어려움  
feature\_importances\_로 영향 변수 상위 15개를 시각화하여  
설명력을 확보함

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
...  
    === Top-10 Feature Importances - Classifier ===  
  
...    Revenue          0.101989  
    Rating           0.095649  
    Company_age     0.095461  
    Position_Encoded 0.072271  
    Job Title_Data Analysis 0.040462  
    Job Title_Data Science   0.040288  
    Location_state_binned_Other 0.020764  
    Job Title_Data Engineering 0.019078  
    works_at_headquarters_No 0.018749  
    works_at_headquarters_Yes 0.018242  
    dtype: float64  
  
...  
    === Top-10 Feature Importances - Regressor (Trim 모델) ===  
  
...    Position_Encoded 0.100333  
    Company_age      0.083643  
    Job Title_Data Analysis 0.081592  
    Revenue          0.077708  
    Rating           0.073872  
    Job Title_Data Science   0.059857  
    Location_state_binned_Other 0.024444  
    Location_state_binned_ca 0.023014  
    Ownership_Grouped_Public 0.018639  
    Job Title_Medical/Healthcare 0.017235  
    dtype: float64
```

### 분석 목표

1. 급여 밴드 분류 (Classification)
2. 평균 급여 정량 예측 (Regression)
3. 급여 결정 요인 해석

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
# 3) 숫자형 vs 범주형 분리
numeric_cols = df[missing_cols].select_dtypes(include=['number']).columns.tolist()
# Position_Encoded는 ordinal 범주형으로 처리하기 위해 숫자형 리스트에서 제외
numeric_cols = [col for col in numeric_cols if col != 'Position_Encoded']

categorical_cols = df[missing_cols].select_dtypes(include=['object', 'category']).columns.tolist()
# Position_Encoded를 범주형 처리 대상에 추가
if 'Position_Encoded' not in categorical_cols:
    categorical_cols.append('Position_Encoded')

# 4) 숫자형은 평균의 반올림 값으로 채우기
for col in numeric_cols:
    mean_val = df[col].mean()
    fill_val = int(round(mean_val))
    df[col].fillna(fill_val, inplace=True)
    print(f"Filled numeric {col} with rounded mean={fill_val}")

# 5) 범주형은 최빈값으로 채우기
for col in categorical_cols:
    mode_val = df[col].mode()[0]
    df[col].fillna(mode_val, inplace=True)
    print(f"Filled categorical {col} with mode={mode_val}")

# 6) 검증: 모든 결측치가 없어졌는지 확인
print("\n남은 결측치 개수:", df.isnull().sum())

df.to_csv('glassdoor_cleaned_final_filled.csv', index=False)
# 7) 최종 데이터 저장
```

### 전처리 (Preprocessing)

- 수치형과 범주형 분리: 수치형 컬럼에서 **Position\_Encoded** 제외
- 수치형 변수: 각 평균값을 반올림
- 범주형 변수: 각 컬럼의 최빈값

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
df = pd.read_csv('glassdoor_cleaned_final_filled.csv')
df['SalaryBand'] = pd.qcut(df['avg_salary'], q=3, labels=['Low','Mid','High'])

X_full = df.drop(columns=['min_salary','avg_salary','max_salary','SalaryBand'])
y_clf = df['SalaryBand']
y_reg = df['avg_salary']
y_log = np.log1p(y_reg) # 로그 타깃

num_cols = X_full.select_dtypes(include=['int64','float64']).columns.tolist()
cat_cols = X_full.select_dtypes(include=['object','category','bool']).columns.tolist()

pre = ColumnTransformer([
    ('num', StandardScaler(), num_cols),
    ('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), cat_cols),
])
df[col].fillna(mode_val, inplace=True)

Filled numeric min_salary with rounded mean=74
Filled numeric max_salary with rounded mean=127
Filled numeric avg_salary with rounded mean=101
Filled numeric Company_age with rounded mean=46
Filled categorical Size_cleaned with mode='Medium'
Filled categorical Position_Encoded with mode='2.0'

남은 결측치 개수:
Job Title 0
Position_Encoded 0
min_salary 0
max_salary 0
avg_salary 0
Rating 0
Location_state_binned 0
Headquarters_state_binned 0
works_at_headquarters 0
Size_cleaned 0
Company_age 0
Ownership_Grouped 0
Industry 0
Sector 0
Revenue 0
dtype: int64
```

급여 분위수로 SalaryBand 생성(Low/Mid/High)  
StandardScaler + OneHotEncoder(ignore) -> ColumnTransformer로 구성

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
cv10_clf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
cv10_reg = KFold(n_splits=10, shuffle=True, random_state=42)

# =====
# 1. 분류 - 클래스 가중치 수동 조정
# =====
clf_weight_pipe = Pipeline([
    ('pre', pre),
    ('clf', RandomForestClassifier(
        n_estimators=1000, max_features='sqrt',
        class_weight={'Low':1, 'Mid':1, 'High':1}, # 수동 가중
        random_state=42
    ))
])

acc = accuracy_score(
    y_clf,
    cross_val_predict(clf_weight_pipe, X_full, y_clf, cv=cv10_clf)
)
cm = confusion_matrix(
    y_clf,
    cross_val_predict(clf_weight_pipe, X_full, y_clf, cv=cv10_clf),
    labels=['Low','Mid','High']
)
print("\n[Classification] Accuracy (w/ custom weights):", round(acc, 3))
print("Confusion Matrix:\n", cm)
print(classification_report(y_clf, cross_val_predict(clf_weight_pipe, X_full, y_clf, cv=cv10_clf)))
```

```
[Classification] Accuracy (w/ custom weights): 0.812
Confusion Matrix:
[[259 32 29]
 [ 41 244 32]
 [ 23 23 273]]
           precision    recall   f1-score   support
High          0.82     0.86     0.84      319
Low          0.80     0.81     0.81      320
Mid          0.82     0.77     0.79      317
accuracy       0.81      --      0.81      956
macro avg     0.81     0.81     0.81      956
weighted avg  0.81     0.81     0.81      956
```

### 클래스 가중치 수동 조절 (분류)

- **RandomForestClassifier(10-fold Stratified CV)**
- **클래스 가중치 실험**  
-> 기본 가중치(1:1:1) 유지 (급여 분위수 분류)

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
# _____  
# 2. 회귀 - 원 타깃 vs. 로그 타깃, 하이퍼파라미터 Grid 확장  
# _____  
reg_pipe = Pipeline([  
    ('pre', pre),  
    ('reg', RandomForestRegressor(random_state=42))  
])  
reg_grid = {  
    'reg_n_estimators': [300, 600, 1000],  
    'reg_max_depth' : [None, 20],  
    'reg_max_features': ['sqrt', 0.6],  
}  
# (a) 원 타깃  
gs_raw = GridSearchCV(  
    reg_pipe, reg_grid,  
    cv=cv10_reg,  
    scoring='neg_root_mean_squared_error',  
    n_jobs=-1  
)  
gs_raw.fit(X_full, y_reg)  
  
rmse_raw = -gs_raw.best_score_  
best_raw_model = gs_raw.best_estimator_  
  
# 원 타깃에 대한 R2 (cross-validated)  
r2_raw_scores = cross_val_score(  
    best_raw_model, X_full, y_reg,  
    cv=cv10_reg, scoring='r2', n_jobs=-1  
)  
r2_raw_mean = r2_raw_scores.mean()  
  
print("\n[Regression] Raw target best RMSE: {:.2f}, R2(mean): {:.3f}".format(rmse_raw, r2_raw_mean))  
print("        Raw target best params:", gs_raw.best_params_)
```

### 1. 원 타깃 (회귀)

- 그리드 서치 (`n_estimators: 300·600·1000, max_depth: None·20, max_features: 'sqrt'·0.6`)  
→ 총  $3 \times 2 \times 2$  조합
- 최적 파라미터: `n_estimators=300, max_features='sqrt', max_depth=None`
- 성능 (CV 기준): RMSE 18.71K,  $R^2$  0.69

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
# (b) 로그 타깃
gs_log = GridSearchCV(
    reg_pipe, reg_grid,
    cv=cv10_reg,
    scoring='neg_root_mean_squared_error',
    n_jobs=-1
)
gs_log.fit(X_full, y_log)

rmse_log = -gs_log.best_score_
best_log_model = gs_log.best_estimator_

# 로그 모델에 대한 R2 (cross-validated, 로그 스케일)
r2_log_scores = cross_val_score(
    best_log_model, X_full, y_log,
    cv=cv10_reg, scoring='r2', n_jobs=-1
)
r2_log_mean = r2_log_scores.mean()

print("[Regression] Log target best log-RMSE: {:.3f}, R2(log mean): {:.3f}".format(rmse_log, r2_log_mean))
print("          Log target best params:", gs_log.best_params_)

# 로그 모델을 역변환해 실제 RMSE 및 R2 환산
y_pred_log = np.expm1(
    cross_val_predict(best_log_model, X_full, y_log, cv=cv10_reg)
)
mse_log_back = mean_squared_error(y_log, y_pred_log)
rmse_log_back = np.sqrt(mse_log_back)

# 로그 모델(back-transform) 예측값에 대한 R2
r2_log_back = r2_score(y_log, y_pred_log)

print("      ↳ RMSE (back-transform): {:.2f}, R2(back-transform): {:.3f}".format(rmse_log_back, r2_log_back))
```

### 2. 로그 변환 타깃 (회귀) - log1p(avg\_salary)

- **타깃을 log1p(avg\_salary)로 변환**  
→ 원 타깃과 동일한 그리드를 탐색
- **로그 변환에서는 R<sup>2</sup> 0.70**
- **역변환 시 RMSE가 19.26 K로 악화**

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
# _____  
# 3. 회귀 - 극단치(outlier) 제거 후 재학습  
#   (상·하위 2% 컷 기준; 필요 시 다른 임계값으로 수정)  
# _____  
  
low_q, high_q = y_reg.quantile([0.02, 0.98])  
mask = (y_reg >= low_q) & (y_reg <= high_q)  
  
X_trim = X_full.loc[mask]  
y_trim = y_reg.loc[mask]  
  
trim_pipe = Pipeline([  
    ('pre', pre),  
    ('reg', RandomForestRegressor(  
        n_estimators=gs_raw.best_params_['reg_n_estimators'],  
        max_depth=gs_raw.best_params_['reg_max_depth'],  
        max_features=gs_raw.best_params_['reg_max_features'],  
        random_state=42))  
])  
  
# 트리밍된 데이터로 cross_val_predict를 사용해 예측  
y_pred_trim = cross_val_predict(trim_pipe, X_trim, y_trim, cv=cv10_reg)  
mse_trim = mean_squared_error(y_trim, y_pred_trim)  
rmse_trim = np.sqrt(mse_trim)  
  
# 트리밍 후 R2 (cross-validated)  
r2_trim_scores = cross_val_score(  
    trim_pipe, X_trim, y_trim,  
    cv=cv10_reg, scoring='r2', n_jobs=-1  
)  
r2_trim_mean = r2_trim_scores.mean()  
  
print("\n[Regression] RMSE after trimming 2% tails: {:.2f}, R2(trim mean): {:.3f}".format(rmse_trim, r2_trim_mean))
```

### 3. 극단치(상·하위 2 %) 제거(trim) 전처리 후 재학습

- 전처리: 급여 분포의 하위 2 %(≤ ≈ 30 K\$)와 상위 2 %(≥ ≈ 200 K\$) 샘플을 제외  
->극단치로 인한 모델 분산을 완화
- 학습 알고리즘:  
이전과 동일한 **RandomForestRegressor**  
(n = 300, max\_features = v, max\_depth = None)/  
10-fold CV를 수행

# 05 모델링 (Modeling)

분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석  
및 예측

```
[Regression] Raw target best RMSE: 18.71, R2(mean): 0.695
    Raw target best params: {'reg__max_depth': None, 'reg__max_features': 'sqrt', 'reg__n_estimators': 300}
[Regression] Log target best log-RMSE: 0.204, R2(log mean): 0.700
    Log target best params: {'reg__max_depth': None, 'reg__max_features': 'sqrt', 'reg__n_estimators': 300}
↳ RMSE (back-transform): 19.26, R2(back-transform): 0.683

[Regression] RMSE after trimming 2% tails: 15.96, R2(trim mean): 0.687
```

Trim 적용 결과: 10-Fold RMSE ≈ 15.96K → 전체 데이터 기준 모델 대비 약 16% 개선  
 $R^2$ : 0.69로 다른 회귀 실험들과 유사

-> 'Trim 2% 전처리 + Random Forest 회귀' 조합을 최종 회귀 선정

# 05 모델링 (Modeling)

분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
# =====
# 4. 잔차 플롯 비교
# =====

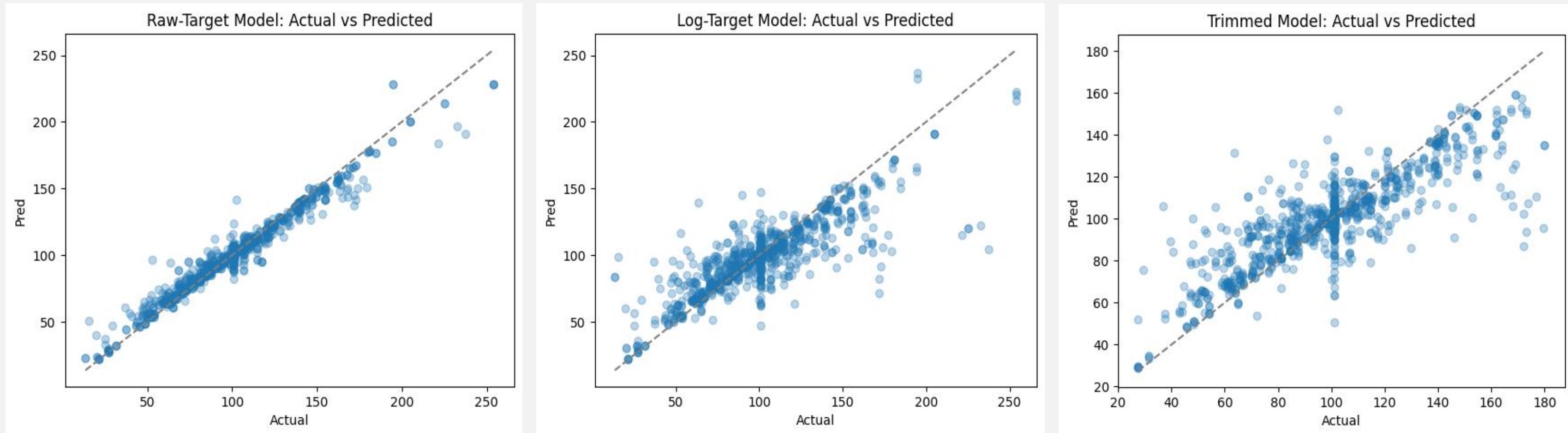
plt.figure()
plt.scatter(y_reg, gs_raw.best_estimator_.predict(X_full), alpha=0.3)
plt.title('Raw-Target Model: Actual vs Predicted'); plt.xlabel('Actual'); plt.ylabel('Pred')
plt.plot([y_reg.min(), y_reg.max()], [y_reg.min(), y_reg.max()], '--', c='gray')

plt.figure()
plt.scatter(y_reg, y_pred_log, alpha=0.3)
plt.title('Log-Target Model: Actual vs Predicted'); plt.xlabel('Actual'); plt.ylabel('Pred')
plt.plot([y_reg.min(), y_reg.max()], [y_reg.min(), y_reg.max()], '--', c='gray')

plt.figure()
plt.scatter(y_trim, cross_val_predict(trim_pipe, X_trim, y_trim, cv=cv10_reg), alpha=0.3)
plt.title('Trimmed Model: Actual vs Predicted'); plt.xlabel('Actual'); plt.ylabel('Pred')
plt.plot([y_trim.min(), y_trim.max()], [y_trim.min(), y_trim.max()], '--', c='gray')
plt.show()
```

# 05 모델링 (Modeling)

분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석  
및 예측



# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
# =====
# 1. 분류 - 클래스 가중치 1:3:1
# =====
clf_weight_pipe = Pipeline([
    ('pre', pre),
    ('clf', RandomForestClassifier(
        n_estimators=1000, max_features='sqrt',
        class_weight={'Low':1, 'Mid':3, 'High':1}, # 수동 가중
        random_state=42
    ))
])

acc = accuracy_score(
    y_clf,
    cross_val_predict(clf_weight_pipe, X_full, y_clf, cv=cv10_clf)
)

cm = confusion_matrix(
    y_clf,
    cross_val_predict(clf_weight_pipe, X_full, y_clf, cv=cv10_clf),
    labels=['Low', 'Mid', 'High']
)

print("\n[Classification] Accuracy (w/ custom weights):", round(acc, 3))
print("Confusion Matrix:\n", cm)
print(classification_report(y_clf, cross_val_predict(clf_weight_pipe, X_full, y_clf, cv=cv10_clf)))
```

[Classification] Accuracy (w/ custom weights): 0.804  
Confusion Matrix:  
[[253 39 28]  
 [ 36 246 35]  
 [ 23 26 270]]

	precision	recall	f1-score	support
High	0.81	0.85	0.83	319
Low	0.81	0.79	0.80	320
Mid	0.79	0.78	0.78	317
accuracy			0.80	956
macro avg	0.80	0.80	0.80	956
weighted avg	0.80	0.80	0.80	956

● 분류 - 클래스 가중치 수동 조정

# 05 모델링 (Modeling)

## 분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석 및 예측

```
# 5-1) 분류기 파이프라인 전체 데이터로 최종 적합
clf_weight_pipe.fit(X_full, y_clf)

# 5-2) 회귀 파이프라인 Trim(±2 %) 데이터로 최종 적합
trim_pipe.fit(X_trim, y_trim)

# 5-3) 전처리기에 들어간 최종 feature 이름 추출
preproc = clf_weight_pipe.named_steps['pre']           # 두 파이프라인 공통
num_names = preproc.transformers_[0][2]                # numeric feature list
cat_names = preproc.named_transformers_['cat']\
|   |   |   |   .get_feature_names_out(preproc.transformers_[1][2])
feature_names = list(num_names) + list(cat_names)

# 5-4) importance 값 가져오기
fi_clf = clf_weight_pipe.named_steps['clf'].feature_importances_
fi_reg = trim_pipe.named_steps['reg'].feature_importances_

# 5-5) Top-10 시리즈 생성
import pandas as pd
top10_clf = (pd.Series(fi_clf, index=feature_names)
|   |   |   |   .sort_values(ascending=False)
|   |   |   |   .head(10))
top10_reg = (pd.Series(fi_reg, index=feature_names)
|   |   |   |   .sort_values(ascending=False)
|   |   |   |   .head(10))

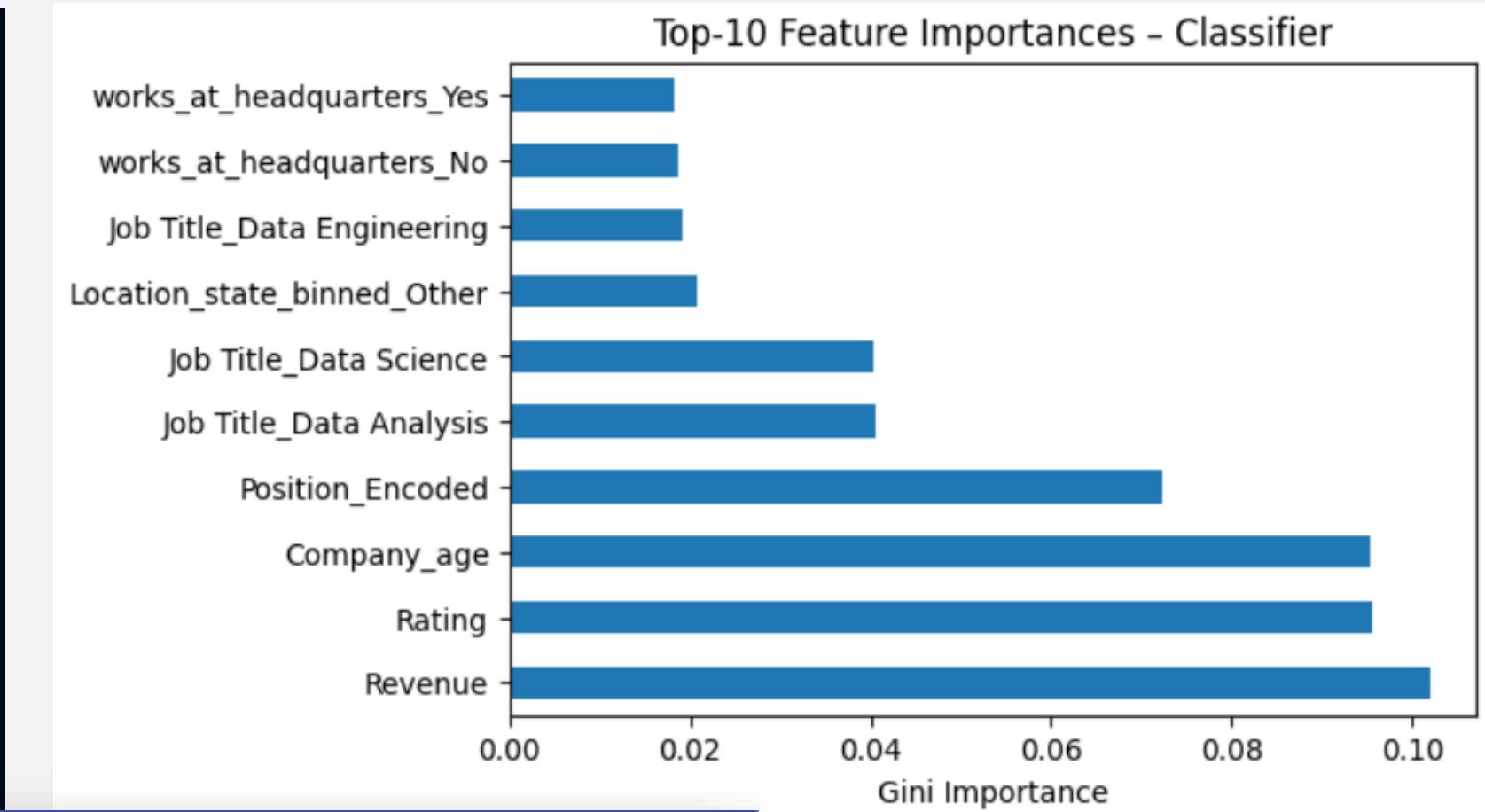
print("\n== Top-10 Feature Importances [] Classifier ===")
display(top10_clf)
```

# 05 모델링 (Modeling)

분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석  
및 예측

```
...  
...  
==== Top-10 Feature Importances - Classifier ====  
...  
Revenue 0.101989  
Rating 0.095649  
Company_age 0.095461  
Position_Encoded 0.072271  
Job Title_Data Analysis 0.040462  
Job Title_Data Science 0.040288  
Location_state_binned_Other 0.020764  
Job Title_Data Engineering 0.019078  
works_at_headquarters_No 0.018749  
works_at_headquarters_Yes 0.018242  
dtype: float64
```

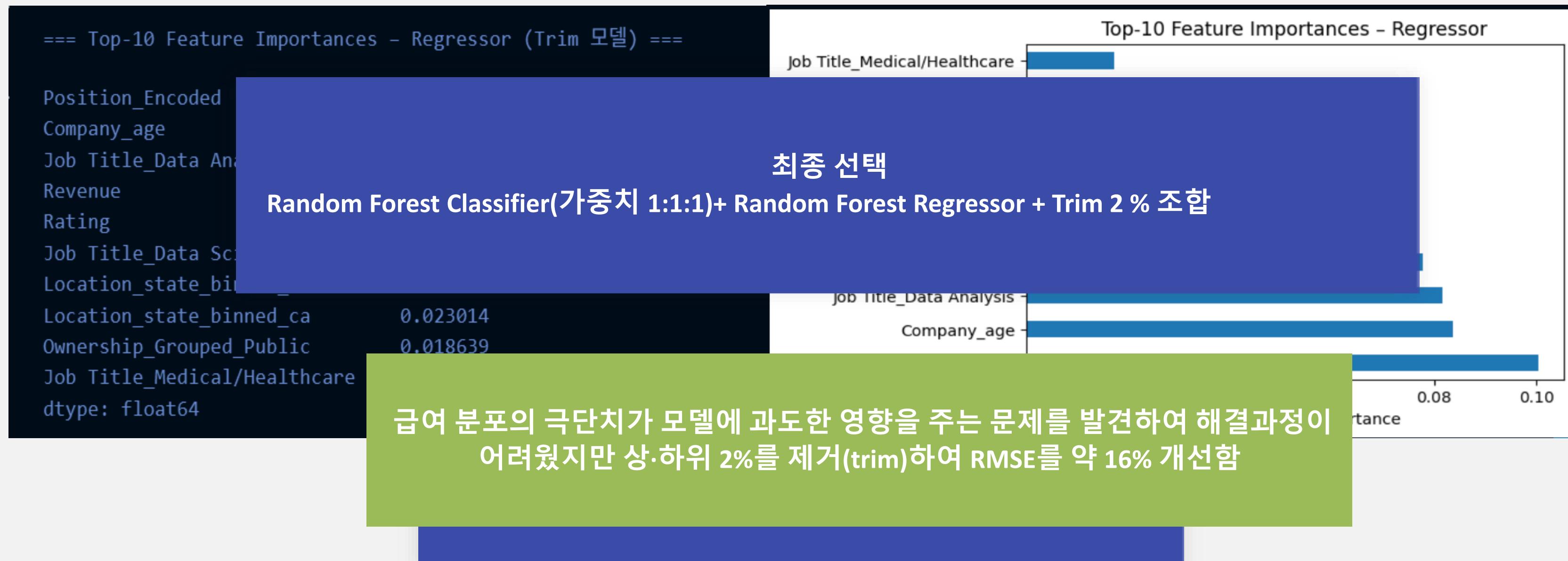
분류



- Accuracy 0.812, Macro-F1 0.811, Mid Recall 0.770
- 전체 성능은 81 %대에서 안정적

# 05 모델링 (Modeling)

분류, 회귀를 활용한 기업 특성과 데이터 사이언스 직군 연봉의 다층적 관계 분석  
및 예측



# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측

예측

목표

1. 회귀 모델을 통한 “연봉” 예측
2. 회귀 모델로 기업 “평점” 예측
3. 평점 예측을 기반으로 “선호 vs 비선호” 회사 분류

회귀 목표

구직자의 정보만 있으면 연봉 예측

Job Title	Position_Rating	Location_s	Headquarter	works_at_h	Size_cleaned	Company_Ownership	Industry	Sector	Revenue	avg_salary
Data Scier2	3.8	Other	ca	No	Medium	52	Private	Aerospace	Industrial & Energy	75000000
Data Scier2	3.4	Other	Other	No	Very Large	41	Private	Health Care	Healthcare & Education	3.5E+9
Data Scier2	4.8	Other	Other	Yes	Medium	15	Private	Security S	Finance & Consulting	3E+8
Data Scier2	3.8	Other	Other	Yes	Large	60	Public Ser	Energy	Industrial & Energy	8.115E+8

분류 목표

이 회사는 다른 구직자들이 선호하는 회사일까?

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 연봉예측

## 사용 모듈

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import RepeatedKFold, cross_validate
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV
from IPython.display import display
```

## 전처리

```
numeric_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean'))])

# categorical pipeline
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))])

categorical_features = [
    feature for feature in df.columns if df[feature].dtype == 'category']

categorical_pipeline.set_params(
    imputer__strategy='most_frequent',
    encoder__handle_unknown='ignore')

categorical_pipeline.fit(categorical_features)

categorical_pipeline.named_steps['imputer'].fit(categorical_features)
categorical_pipeline.named_steps['encoder'].fit(categorical_features)

categorical_pipeline.named_steps['imputer'].transform(categorical_features)
categorical_pipeline.named_steps['encoder'].transform(categorical_features)
```

avg\_salary에 결측치가 있었으나 이를 평균 등으로 채우면 모델 학습에 오류가 발생할 수 있어, 해당 행들을 제거해 타깃 왜곡을 방지

## 코드 설명

- 1차 전처리는 공통  
2차 전처리는 각 모델에 적합하게 적용
- 2차 전처리 :  
수치형 - 평균값 대체  
범주형 - 최빈값 대체
- 범주형은 one hot encoding

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 연봉예측

## 모델 종류

```
# Linear
lin_importance = (pd.DataFrame({
    'Feature': get_feature_names(lin_pipeline),
    'Importance': np.abs(lin_pipeline.named_steps['regressor'].coef_)
}).sort_values('Importance', ascending=False).head(20))
```

avg\_salary가 min\_salary와 max\_salary의 평균이라  
선형 회귀 모델이 이 변수들에만 과도하게 의존해 과적합이 발생했고,  
두 변수를 제거해 일반화 성능을 확보했다.

```
# GradientBoosting
gbr_importance = (pd.DataFrame({
    'Feature': get_feature_names(gbr_pipeline),
    'Importance': gbr_pipeline.named_steps['regressor'].feature_importances_
}).sort_values('Importance', ascending=False).head(20))
```

## 코드 설명

- 단순 한모델을 정하는 것이 아니라 3개를 비교 후 베스트 모델 선택
- 선형과 비선형 모델의 성능 비교 목적

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 연봉예측

모델 성능 비교

```
def evaluate(pipe):
    cv_results = cross_validate(pipe, X_train, y_train,
                                scoring=scoring, cv=cv, n_jobs=-1, return_train_score=False)
    return {
        'R2_mean': np.mean(cv_results['test_R2']),
        'RMSE_mean': -np.mean(cv_results['test_RMSE']),
        'MAE_mean': -np.mean(cv_results['test_MAE'])
    }
```

Model Evaluation Metrics				
	Model	R2_mean	RMSE_mean	MAE_mean
2	GradientBoostingRegressor	0.696054	20.733721	11.896198
1	RandomForestRegressor	0.668284	21.680130	14.107624
0	Linear Regression	-0.004144	38.407432	30.233839

## 코드 설명

- 모델 평가 기준은  $R^2$ , RMSE, MAE 복합적 고려
- GradientBoostingRegressor 69%로 가장 성능 우수

! 여기서 드는 의문 !

성능이 더 좋은 파라미터 조합은 없을까?

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 연봉예측

하이퍼 파라미터

```
rf_param_grid = {
    'regressor__n_estimators': [50, 100, 150, 200, 250],
    'regressor__max_depth': [None, 5, 10, 15, 20],
    'regressor__min_samples_split': [2, 5, 10],
    'regressor__max_features': ['sqrt', 'log2',1.0]}

gbr_param_grid = {
    'regressor__n_estimators': [100, 200, 300, 400],
    'regressor__learning_rate': [0.01, 0.05, 0.1, 0.2],
    'regressor__max_depth': [3, 5, 7, 9],
    'regressor__subsample': [0.6, 0.8, 1.0]}

def hyperparameter_tuning(pipeline, param_grid):
    search = RandomizedSearchCV(
        pipeline, param_grid, n_iter=50, # 50개 샘플링
        scoring='r2', cv=cv, n_jobs=-1 )
    search.fit(X_train, y_train)
    return search
```

## 코드 설명

- **RandomForest**  
225가지 중 랜덤으로 50개 선택
- **GradientBoosting**  
192가지 중 랜덤으로 50개 선택
- **Linear Regression**  
학습 과정에서 튜닝할 파라미터  
적절하지 않음

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 연봉예측

하이퍼 파라미터 결과 - RandomForestRegressor

Rank	Mean Test Score	n_estimators	max_depth	min_samples_split	max_features
1	0.674366	50	1	2	sqrt
2	0.666768	50	1	2	log2
3	0.666469	100	20	2	sqrt
4	0.665067	50	20	2	sqrt
5	0.662206	150			

Q. 연봉과 가장 연관된 특성들은?

하이퍼 파라미터 결과 - GradientBoostingRegressor

Rank	Mean Test Score	n_estimators	learning_rate	max_depth	subsample
1	0.704506	400	0.05	7	0.6
2	0.703699	300	0.05	9	0.6
3	0.700908	300	0.1	9	0.8
4	0.700798	100	0.1	9	0.8
5	0.69443	400	0.1	9	0.6

코드 설명

- GradientBoost 최적 조합 특징

학습률 낮음 (learning\_rate ↓)

트리 개수 많음 (n\_estimators ↑)

일부 데이터만 사용  
(subsample < 1.0)

트리 깊이 낮음 (max\_depth ↓)

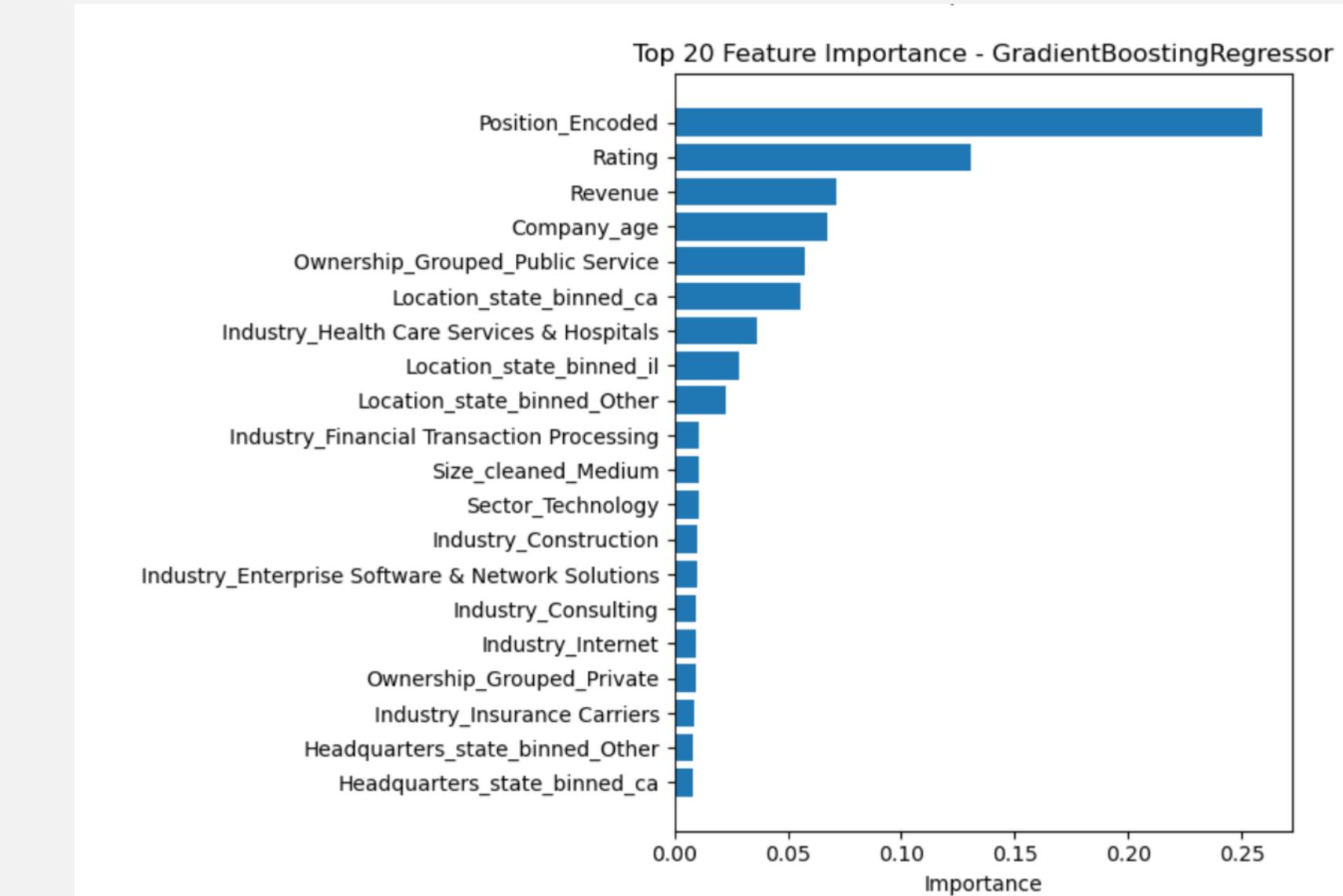
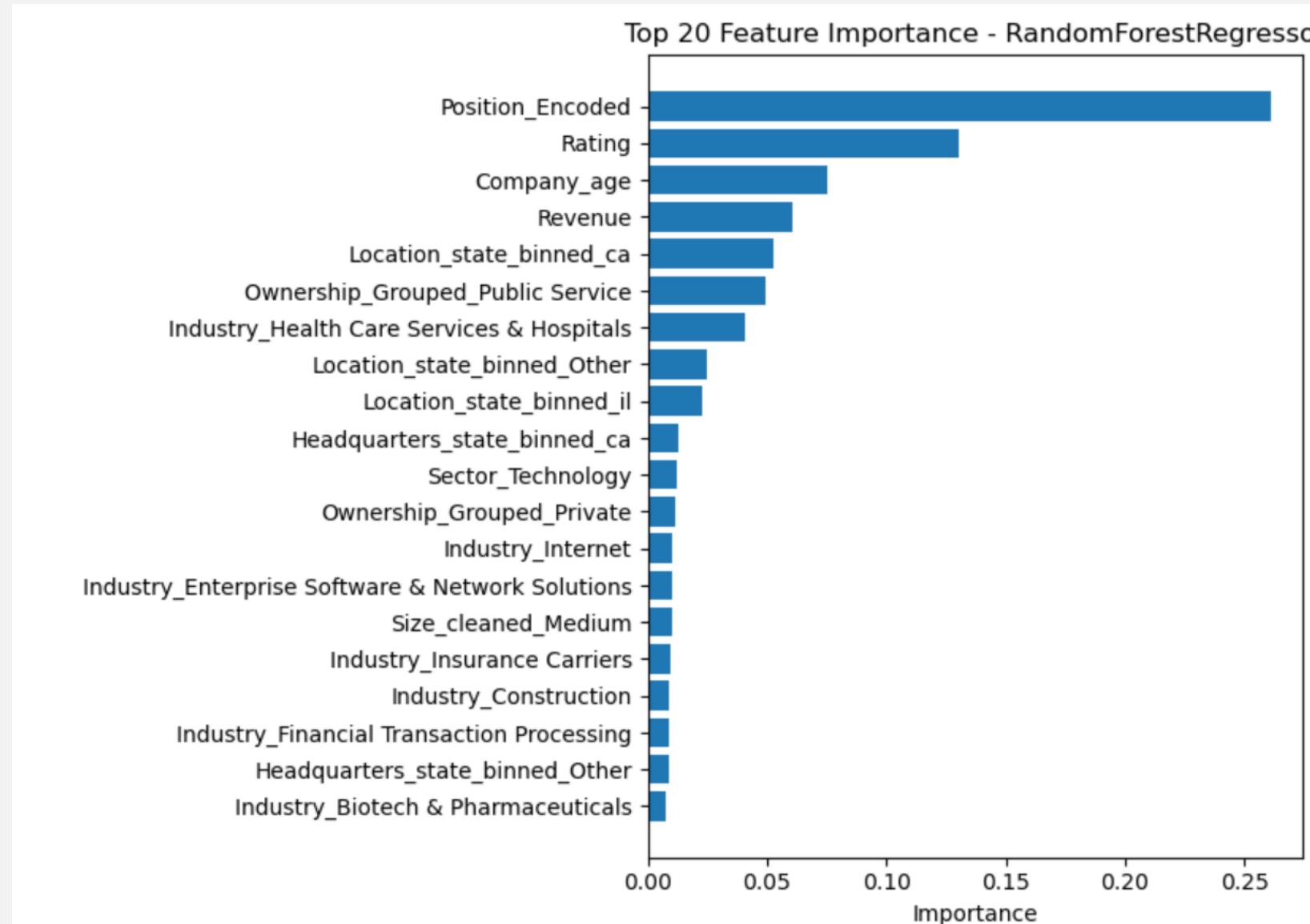
최종 결론

GradientBoost가 70%의 정확도로  
모든 모델 중 가장 성능이 우수.

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 연봉예측

연봉과 feature의 연관성



연봉은 직무에 따라 가장 크게 달라지며,  
평판이 좋은 오래된 회사일수록 연봉도 높다.

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 기업 평점 예측

## 사용 모듈

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, r2_score, mean_squared_error
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

    numeric_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', StandardScaler())
    ])
    categorical_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])
    preprocessor = ColumnTransformer([
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

범주형 데이터를 수치형으로 변환하는 과정이  
복잡했으나, Pipeline과 ColumnTransformer로  
전처리를 자동화하고 구조화함

전처리

## 코드 설명

- 1차 전처리는 공통  
2차 전처리는 각 모델에  
적합하게 적용
- 2차 전처리 :  
수치형- StandardScaler 표준화  
범주형- OneHotEncoder 인코딩

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 기업 평점 예측

```
test_sizes = [0.2, 0.3, 0.4, 0.5]
linreg_results = []
reg_models = []

for ts in test_sizes:
    X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
        X_processed, y_reg, test_size=ts, random_state=42)
    reg = LinearRegression()
    reg.fit(X_train_r, y_train_r)
    y_pred_r = reg.predict(X_test_r)
    r2 = r2_score(y_test_r, y_pred_r)
    rmse = np.sqrt(mean_squared_error(y_test_r, y_pred_r))
    linreg_results.append((ts, r2, rmse))
    reg_models.append((ts, y_test_r, y_pred_r))
```

모델 - 회귀(선형 회귀)

```
max_depths = [3, 4, 5, 6, 7]
dt_results = []

X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(
    X_processed, y_cls, test_size=0.2, random_state=42)
for d in max_depths:
    clf = DecisionTreeClassifier(max_depth=d, random_state=42)
    clf.fit(X_train_c, y_train_c)
    y_pred_c = clf.predict(X_test_c)
    acc = accuracy_score(y_test_c, y_pred_c)
    dt_results.append((d, acc))
```

모델 - 분류 (의사결정 트리)

## 코드 설명

- 회귀 모델(기업 평점 예측)
- 분류 모델(좋은 기업 판별)
- 하이퍼 파라미터  
최고의 성능 조합을 찾기
- 단순 기업 평가에서 벗어나  
좋은 기업인지 알아보기 위해  
2가지 방식 복합 사용

# 05 모델링 (Modeling) - 회귀 & 분류

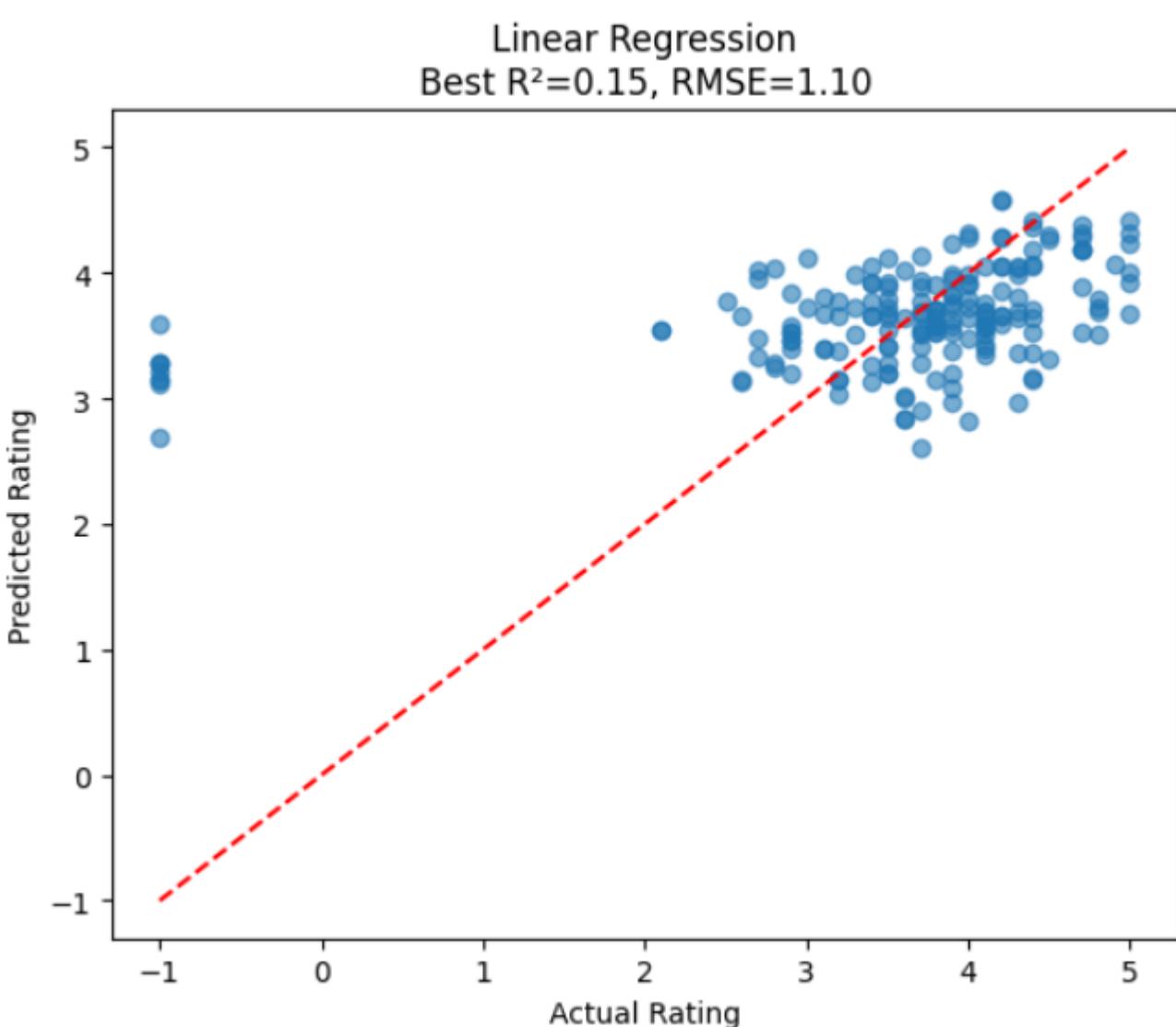
회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 기업 평점 예측

모델 성능 비교 - 회귀

```
# 최적 모델 및 결과 시각화
best_ts, best_r2, best_rmse = max(linreg_results, key=lambda x: x[1])
_, best_y_test_r, best_y_pred_r = max(reg_models, key=lambda x: r2_score(x[1], x[2]))
best_depth, best_acc = max(dt_results, key=lambda x: x[1])

# 최종 결정 트리 모델과 교차 검증
clf_best = DecisionTreeClassifier(max_depth=best_depth)
clf_best.fit(X_train_c, y_train_c)
y_pred_best = clf_best.predict(X_test_c)
cm = confusion_matrix(y_test_c, y_pred_best)
cv = StratifiedKFold(n_splits=5, shuffle=True,
cv_scores = cross_val_score(clf_best, X_process)

# 결과 시각화
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
```



## 코드 설명

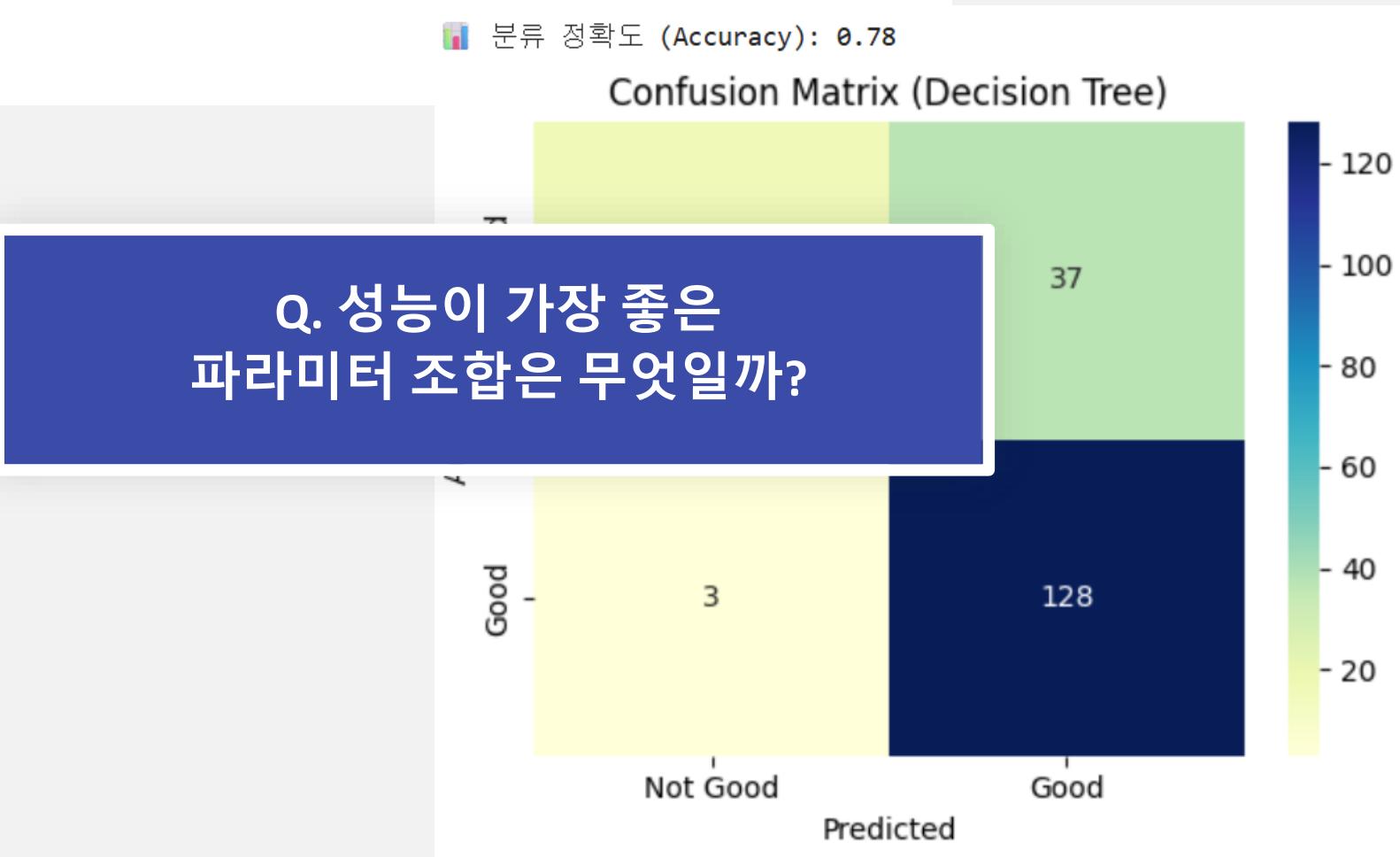
- 모델 평가 기준은  $R^2$ , RMSE 복합적 고려
- 선형 회귀 예측 결과와 실제 값 산점도 시각화
- 기대 효과: 평점이 없는 기업의 예상 평점 추정 가능

# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 기업 평점 예측

모델 성능 비교 - 분류

```
# 혼동 행렬 시각화
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu',
            xticklabels=["Not Good", "Good"], yticklabels=["Not Good", "Good"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title(f"Confusion Matrix (Decision Tree, max_depth={best_depth})")
plt.tight_layout()
plt.show()
```



## 코드 설명

- 분류 모델의 예측 정확도 79%
- 기대 효과:  
구직자가 여러 특성을 바탕으로  
'좋은 회사'를 사전 예측 가능

# 05 모델링 (Modeling) - 회귀 & 분류

## 회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 기업 평점 예측

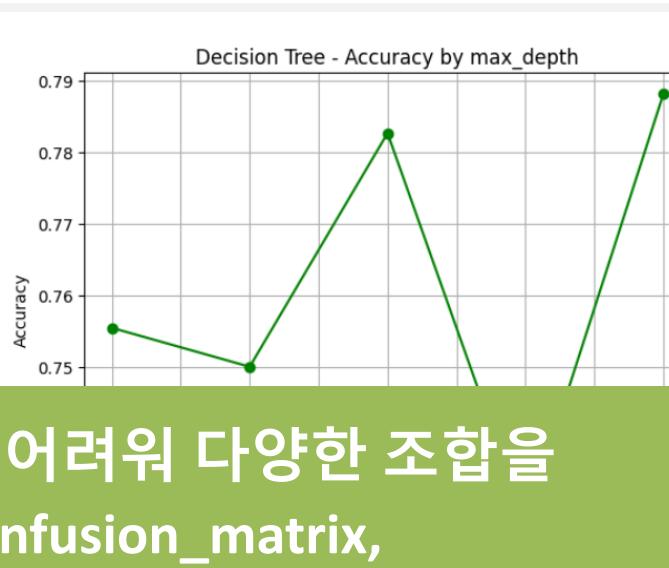
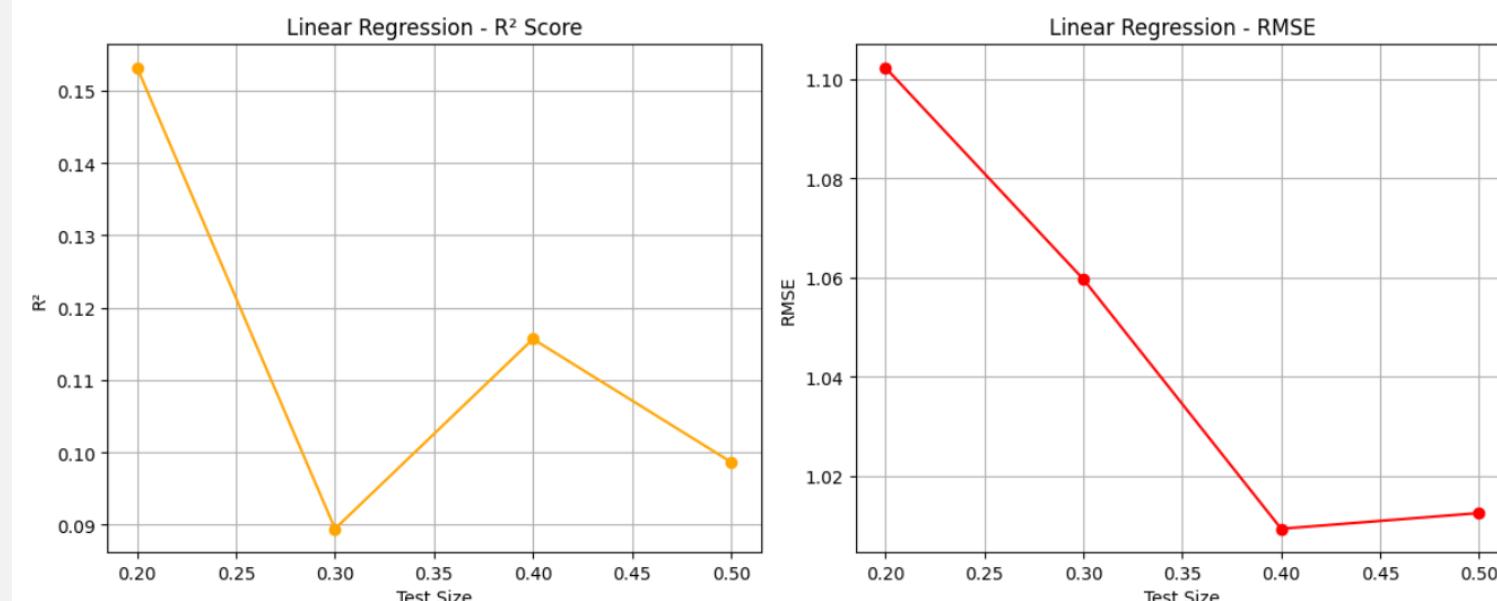
### 하이퍼 파라미터

```
# (1)
# test_sizes = [0.2, 0.3, 0.4, 0.5]
# ts_vreg_models = []
# axes[0]
# axes[1]for ts in test_sizes:
#     X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
#         X_processed, y_reg, test_size=ts, random_state=42)
#     reg = LinearRegression()
#     reg.fit(X_train_r, y_train_r)
#     y_pred_r = reg.predict(X_test_r)
#     r2 = r2_score(y_test_r, y_pred_r)
#     rmse = np.sqrt(mean_squared_error(y_test_r, y_pred_r))
#     linreg_results.append((ts, r2, rmse))
#     reg_models.append((ts, y_test_r, y_pred_r))
# axes[0, 1].grid(True)
```

### 모델 - 회귀(선형 회귀)

```
# (2)
# max_depths = [3, 4, 5, 6, 7]
# dep_dt_results = []
# axes[2]
# axes[3]X_train_c, X_test_c, y_train_c, y
# axes[3]X_processed, y_cls, test_size
# axes[4]for d in max_depths:
#     clf = DecisionTreeClassifier(
#         max_depth=d)
#     clf.fit(X_train_c, y_train_c)
#     y_pred_c = clf.predict(X_test_c)
#     acc = accuracy_score(y_test_c, y_pred_c)
#     dt_results.append((d, acc))
```

단일 파라미터로는 성능 평가가 어려워 다양한 조합을  
실험하고,  $R^2$ , RMSE, Accuracy, confusion\_matrix,  
cross\_val\_score 등을 시각화해 모델 성능과 안정성을  
종합 비교



### 결과 설명

- 회귀 모델
- test\_size = 0.2일 때  $R^2 = 0.15$ ,  
RMSE = 1.10 (최적)
- 분류 모델
- max\_depth = 7일 때  
Accuracy = 0.79 (최적)

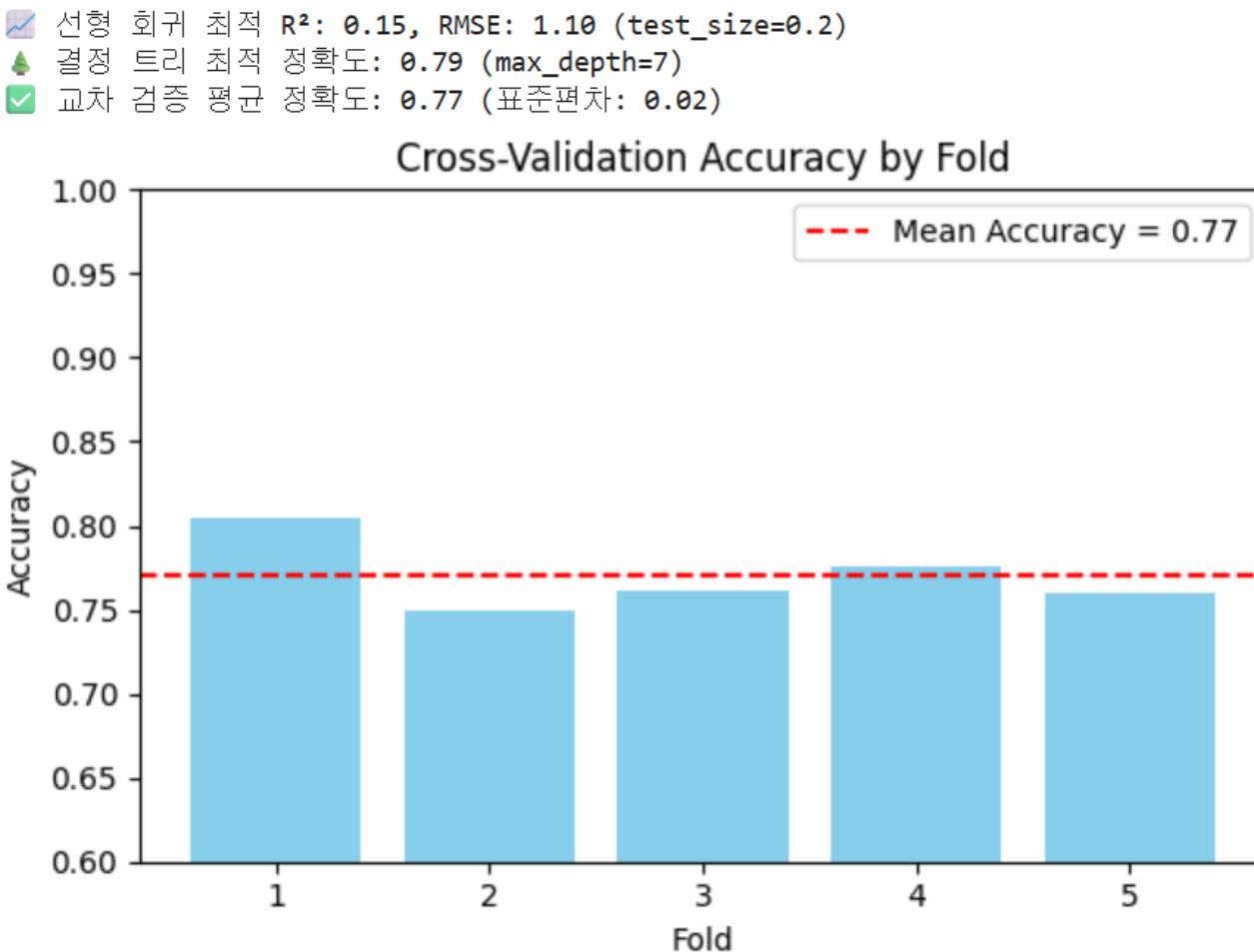
# 05 모델링 (Modeling) - 회귀 & 분류

회귀분석과 분류 기법을 활용해 각각 연봉과 기업평점 예측 - 기업 평점 예측

## 모델 성능 최종 분석

```
print(f"\n선형 회귀 최적 R2: {best_r2:.2f}, RMSE: {best_rmse:.2f} (test_size={best_ts})")
print(f"결정 트리 최적 정확도: {best_acc:.2f} (max_depth={best_depth})")
print(f"교차 검증 평균 정확도: {cv_scores.mean():.2f} (표준편차: {cv_scores.std():.2f})")

plt.figure(figsize=(6, 4))
plt.bar(range(1, 6), cv_scores, color='skyblue')
plt.axhline(y=cv_scores.mean(), color='red', linestyle='--',
            label=f"Mean Accuracy = {cv_scores.mean():.2f}")
plt.title("Cross-Validation Accuracy by Fold")
plt.xlabel("Fold")
plt.ylabel("Accuracy")
plt.ylim(0.6, 1.0)
plt.legend()
plt.tight_layout()
plt.show()
```

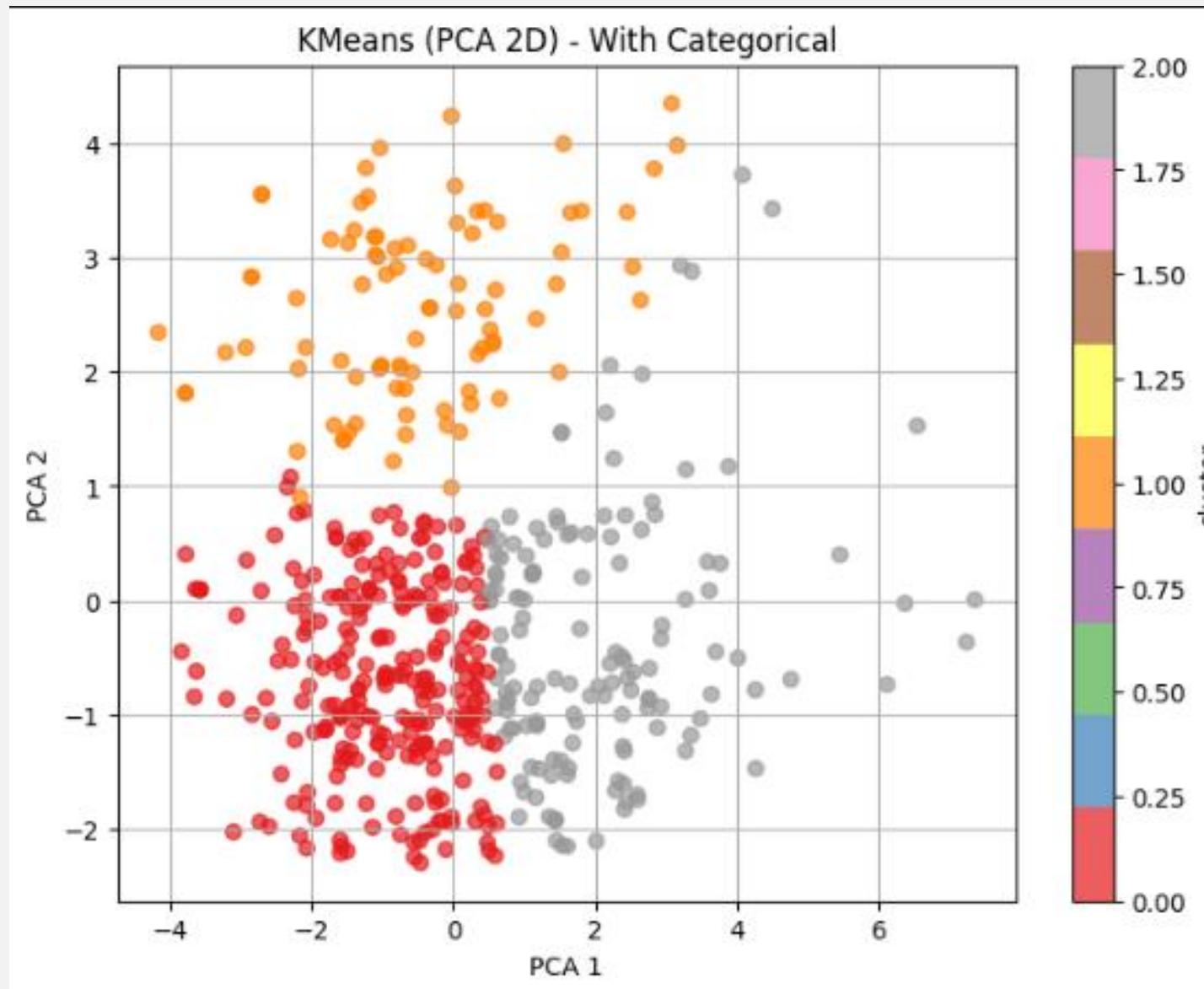


## 결과 분석

- 목표: 모델의 일반화 성능 확인
- 데이터를 5등분하여 반복적으로 학습 및 검증
- 평균 정확도: 0.77
- 평균 정확도 높고 표준편차가 낮아 모델 성능이 일정, 안정적

# 05 모델링 (Modeling)

## 클러스터링을 통한 공고 추천 시스템



### 분석 목표

- Glassdoor 채용공고 데이터를 바탕으로 Kmeans 클러스터링을 적용하여 유사한 직무군을 군집화
- 구직자의 입력값과 유사한 공고를 추천하는 시스템을 구현

가장 가까운 클러스터: cluster#0

해당 클러스터에서 가장 가까운 공고 Top 10:

Company Name	Job Title	Position_Encoded
34	Esri	2.0
434	Mentor Graphics	2.0
31	Juniper Networks	2.0

# 05 모델링 (Modeling)

## Feature Selection

```
# 시나리오 정의
scenarios = {
    'Numeric Only': numeric_cols,
    'With Categorical': numeric_cols + ordered_cols + categorical_cols
}
```

- 수치형: 'min\_salary', 'max\_salary', 'avg\_salary'
- 순서형: 'Position\_Encoded', 'Size\_cleaned'
- 범주형: 'Job Title', 'Location\_state\_binned'

### 분석 목표

- 1. 클러스터링을 수행하므로 수치형 데이터를 활용
- 2. 실험적으로, 순서형 데이터와 일부 범주형 데이터를 활용한  
트리플릿 그룹

### Feature Selection의 어려움

어떤 피처를 쓸지 판단이 어려웠고,

전체 파이프라인에서 다양한 조합으로  
시나리오를 분리하는 아이디어 적용

(별에 영향이 있을 것으로 판단)

	Company Name	Job Title	Position_Encoded	Location_state_binned	Sector	avg_salary	Company_age	Rating
34	Esri	Data Science	2.0	ca	Technology	107.0	56.0	3.5

# 05 모델링 (Modeling)

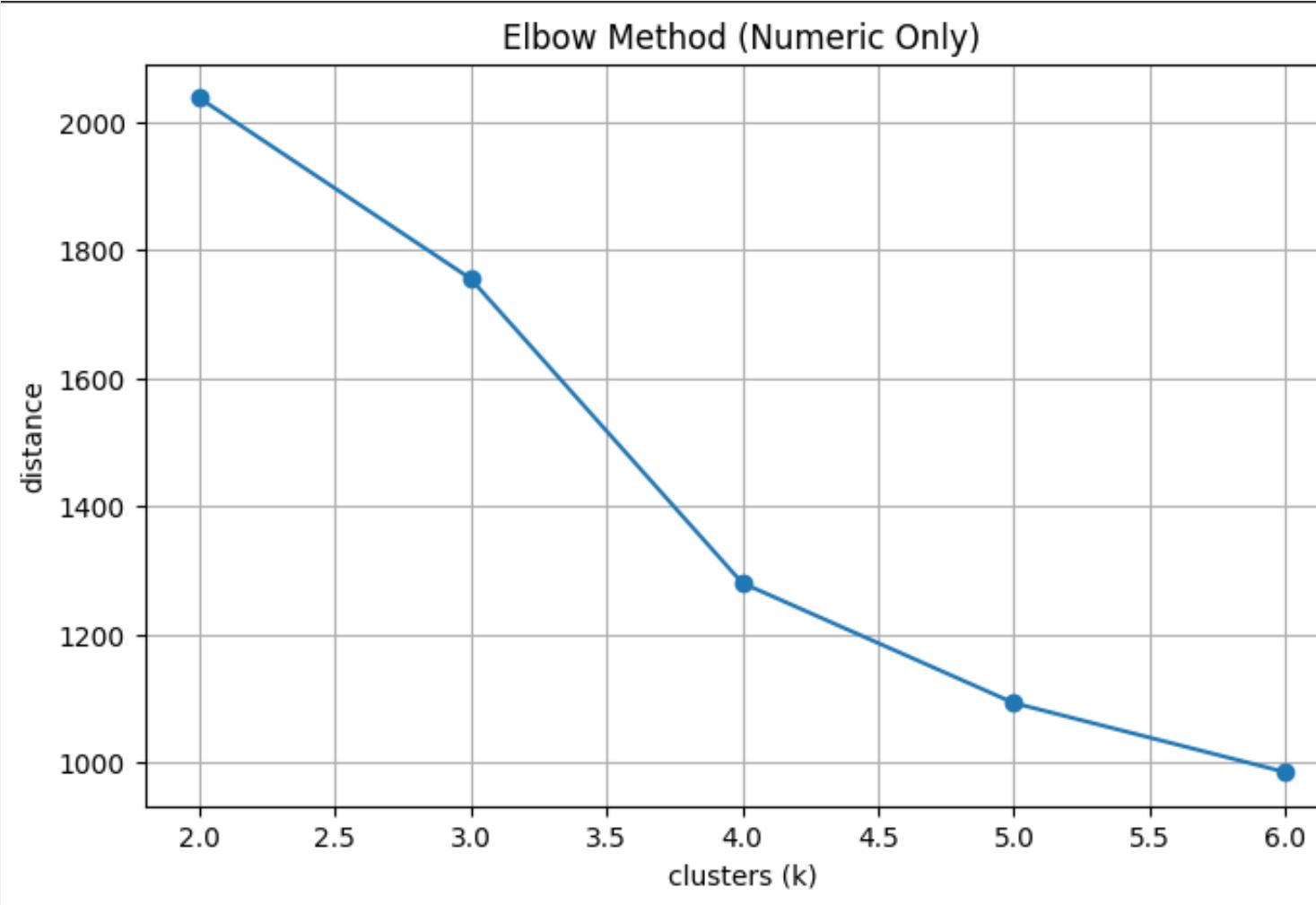
```
# 원-핫 인코딩  
df_encoded_cat = pd.get_dummies(df_filtered[cat_cols], drop_first=True) if cat_cols else pd.DataFrame(index=df_filtered.index)  
  
# 스케일링  
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(df_filtered[num_cols + ord_cols])
```

- 1. 범주형 데이터들에 대해 원핫인코딩 적용
- 2. 나머지 Feature들에 Standard Scailing 적용

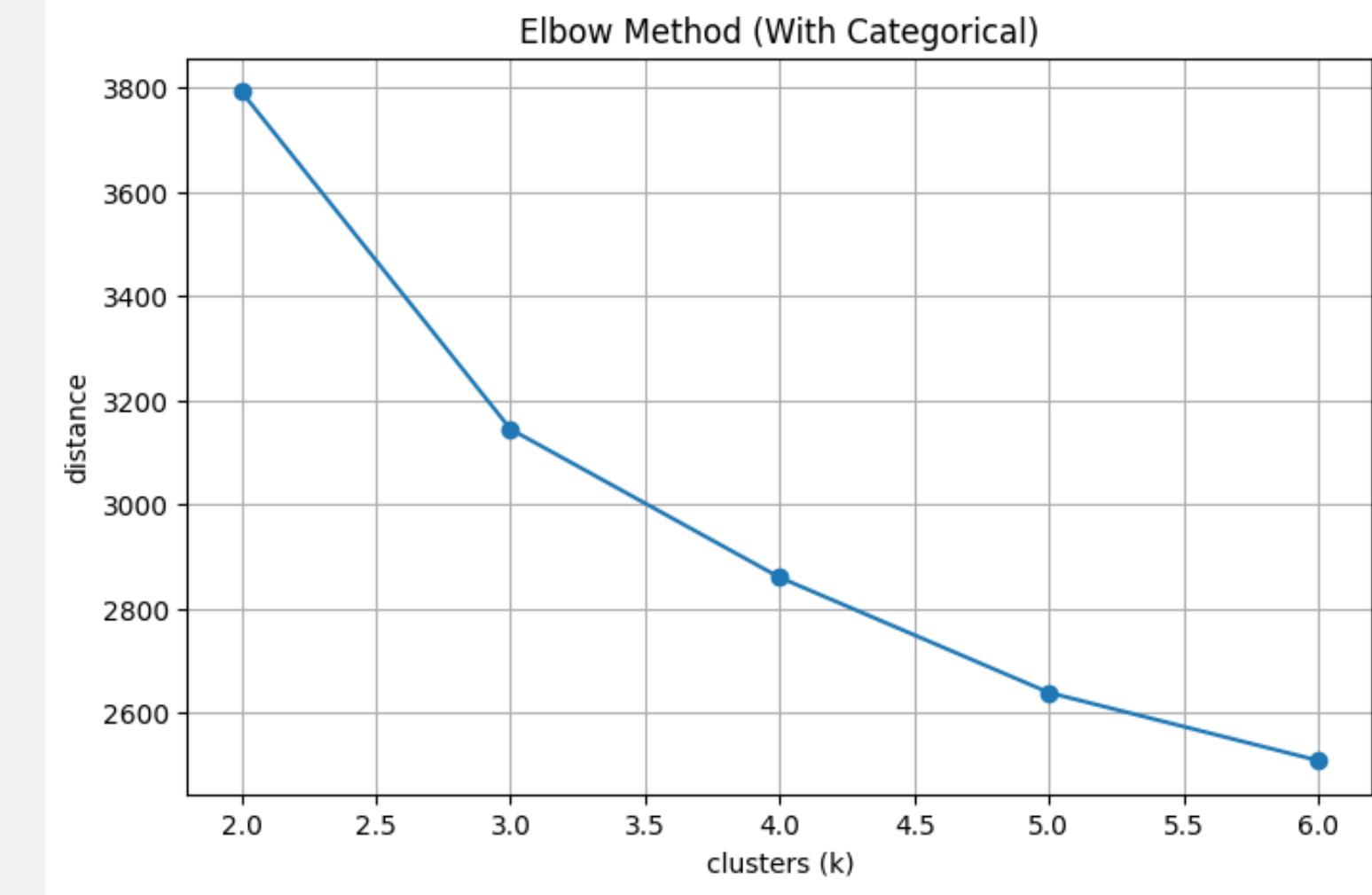
```
# Elbow Method  
inertias = []  
for k in range(2, 7):  
    kmeans = KMeans(n_clusters=k, random_state=42)  
    kmeans.fit(X_final)  
    inertias.append(kmeans.inertia_)
```

- Elbow Method를 통해 최적 K값 탐색

# 05 모델링 (Modeling)



- 시나리오별 최적 K값 확인 및 적용
- Kmeans 학습 수행



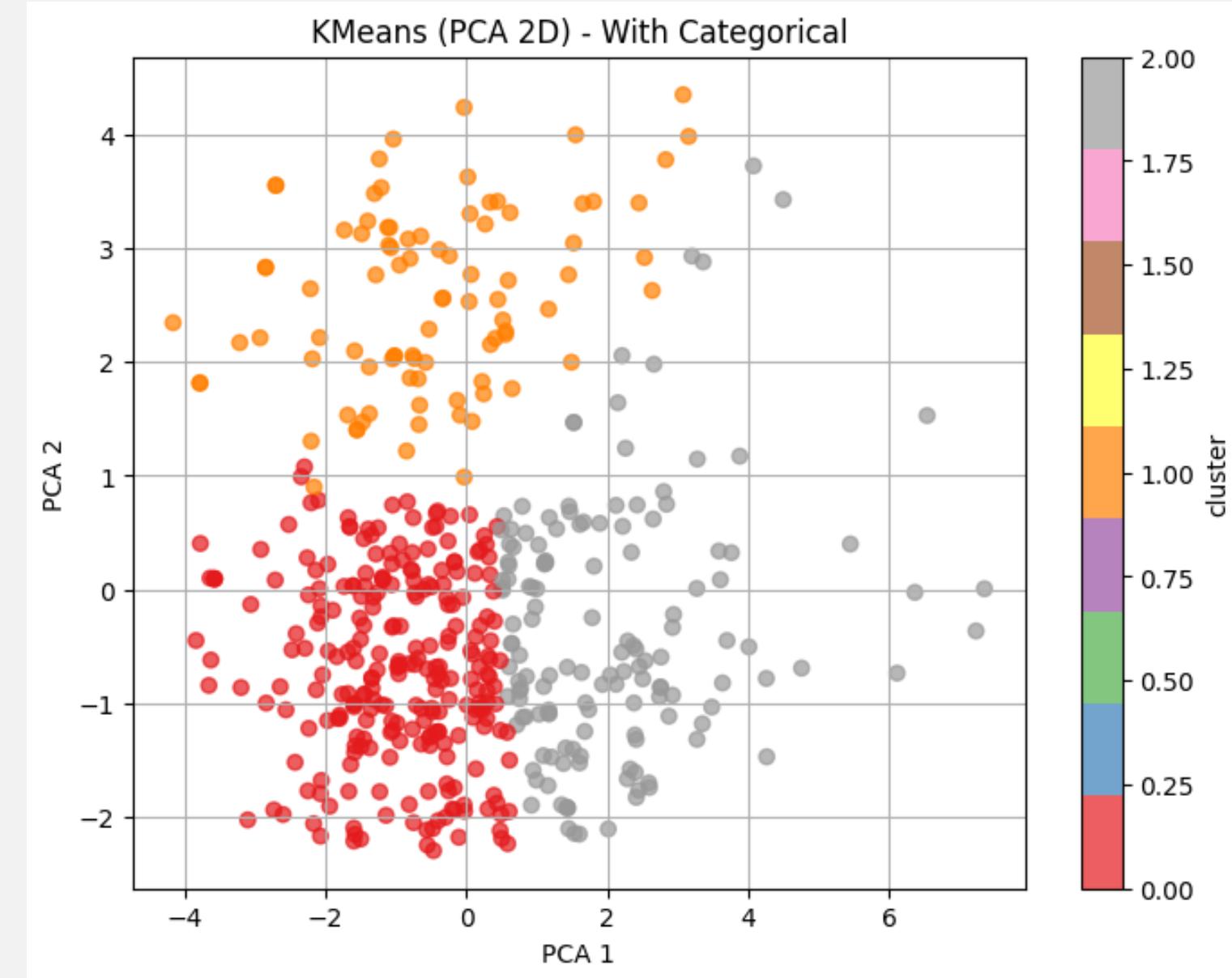
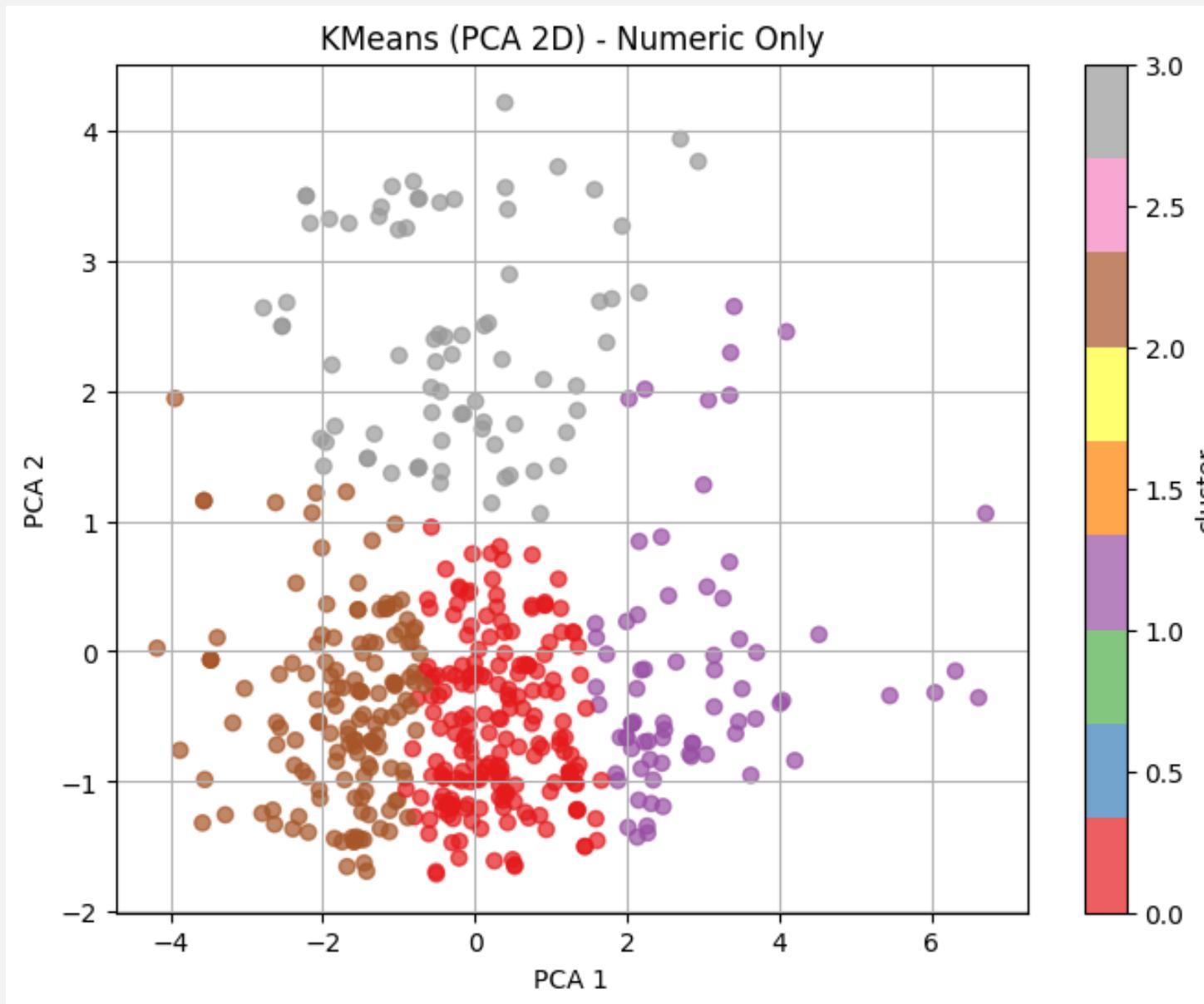
```
# KMeans 클러스터링 (k=3)
# 시나리오별 k값 설정
if scenario_name == 'Numeric Only':
    k_value = 4
elif scenario_name == 'With Categorical':
    k_value = 3

kmeans = KMeans(n_clusters=k_value, random_state=42)
df['cluster'] = kmeans.fit_predict(X_final)
```

# 05 모델링 (Modeling)

```
# PCA 2차원 시각화
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_final)
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['cluster'], cmap='Set1', alpha=0.7)
```

- PCA(2차원)을 통해 시각화



# 05 모델링 (Modeling)

```
# 클러스터 품질 평가 지표  
sil_score = silhouette_score(X_final, df['cluster']) # silhouette score  
db_index = davies_bouldin_score(X_final, df['cluster']) # davies-bouldin index
```

- Silhouette Score
- Davies-Bouldin Index

## • Davies-Bouldin Index

S: 클러스터 내 point – centroid 평균 거리

$$DB = \frac{1}{k} \sum_{i=1}^k \max_{j \neq i} \left( \frac{S_i + S_j}{M_{ij}} \right)$$

M: i와 j의 중심 거리

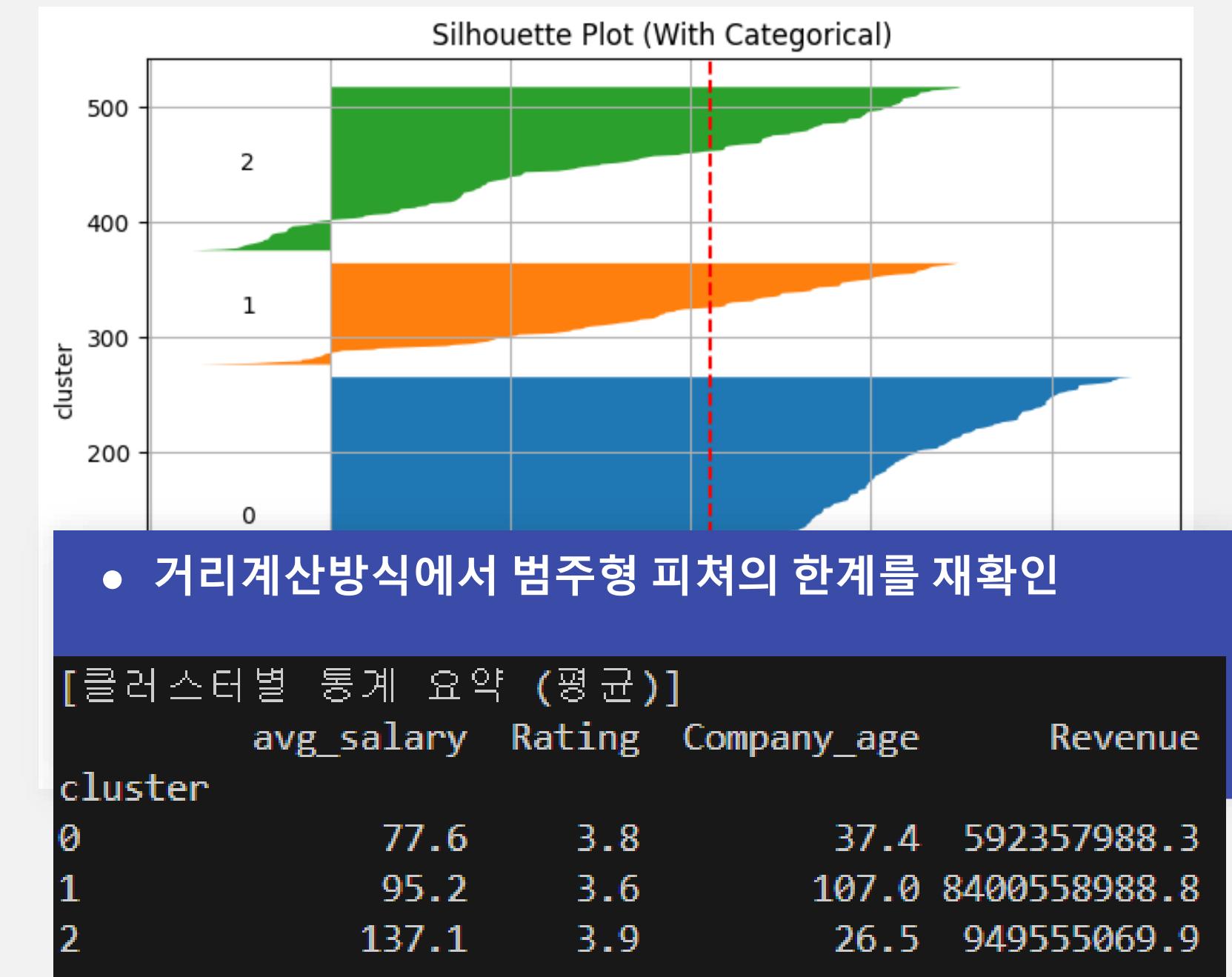
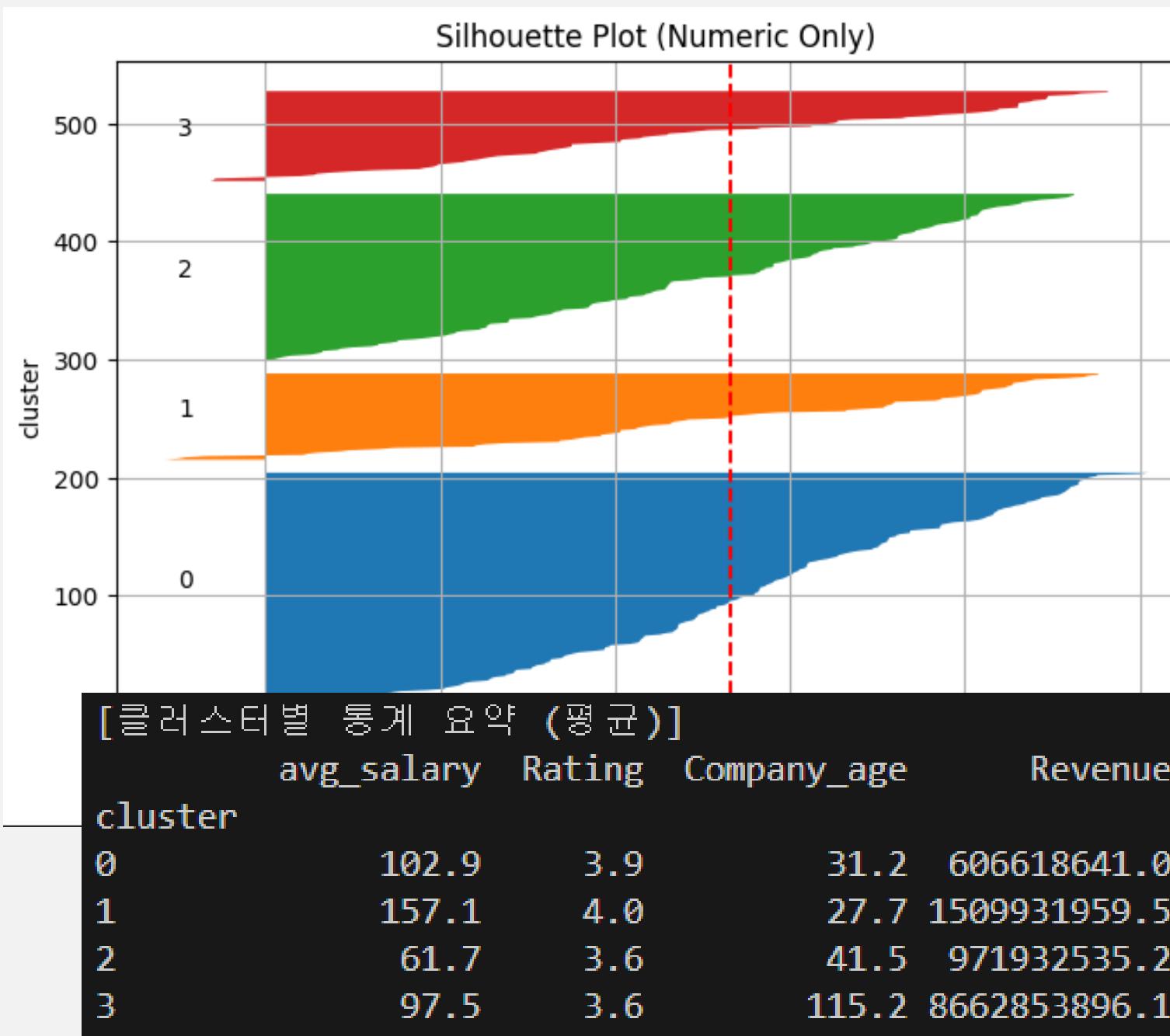
- i와 가장 유사한 j를 찾아 그 유사도를 구함
- 모든 i에 대해 반복하고 평균을 계산 => mean(최악)
- 최솟값 0(이상적)

DBI : 군집간 분리도와 내부 응집도 비율 최대값들의 평균  
SS : 각 샘플의 군집 내 일치도 vs 타 군집과의 거리차이

# 05 모델링 (Modeling)

```
# 클러스터 품질 평가 지표
sil_score = silhouette_score(X_final, df['cluster']) # silhouette score
db_index = davies_bouldin_score(X_final, df['cluster']) # davies-bouldin index
```

- Silhouette Score
- Davies-Bouldin Index



# 05 모델링 (Modeling)

- 시나리오별 사용자 입력

- 연봉과 평점 등을 기준으로(수치형)

```
# 사용자 입력 (시나리오별)
if scenario_name == "Numeric Only":
    input_data = {
        'min_salary': 80, #단위: 1000달러
        'max_salary': 120,
        'avg_salary': 100,
        'Rating': 4.0,
        #dummy 변수는 평균값으로 대체
        'Company_age': df['Company_age'].mean(),
        'Revenue': df['Revenue'].mean()
    }
```

- 직무, 위치, 분야 등 설정 (통합형)

```
else:
    input_data = [
        'Job Title': 'Data Science',
        'Location_state_binned': 'ca',
        'Sector': 'Technology',
        'Position_Encoded': 2.0,
        #dummy 변수는 평균값으로 대체
        'Size_cleaned': df['Size_cleaned'].mean(),
        'min_salary': df['min_salary'].mean(),
        'max_salary': df['max_salary'].mean(),
        'avg_salary': df['avg_salary'].mean(),
        'Rating': df['Rating'].mean(),
        'Company_age': df['Company_age'].mean(),
        'Revenue': df['Revenue'].mean()
    ]
```

- ‘입력없음’의 개념을 평균값으로 처리하여 편향을 최소화

# 05 모델링 (Modeling)

- 연봉과 평점 등을 기준으로(수치형)

```
# 사용자 입력 (시나리오별)
if scenario_name == "Numeric Only":
    input_data = {
        'min_salary': 80, #단위: 1000달러
        'max_salary': 120,
        'avg_salary': 100,
        'Rating': 4.0,
```

가장 가까운 클러스터: cluster#0

해당 클러스터에서 가장 가까운 공고 Top 10:

	Company	Name	Job Title	Position_Encoded	Location_state_binned	Sector	avg_salary	Company_age	Rating
253	ManTech	Data Engineering		2.0	va	Finance & Consulting	92.0	57.0	4.1
143	PNNL	Data Science		4.0	Other	Industrial & Energy	98.5	60.0	3.8
434	Mentor Graphics	Data Science		2.0	ca	Technology	107.0	44.0	4.1
108	Autodesk	Data Engineering		2.0	ca	Technology	108.0	43.0	4.0
140	PatientPoint	Data Science		6.0	Other	Finance & Consulting	100.0	38.0	3.8
263	SciPlay	Data Engineering		2.0	Other	Other Services	107.0	27.0	4.0
153	Legal & General America	Data Science		2.0	Other	Finance & Consulting	107.0	44.0	3.8
257	CyberCoders	Data Science		2.0	va	Finance & Consulting	100.5	26.0	4.2
100	CapTech	Data Engineering		2.0	Other	Technology	95.5	28.0	3.9
180	BioMarin Pharmaceutical	Data Engineering		2.0	ca	Healthcare & Education	94.5	28.0	3.8

# 05 모델링 (Modeling)

## • 직무, 위치, 분야 등 설정 (통합형)

```
else:  
    input_data = [  
        'Job Title': 'Data Science',  
        'Location_state_binned': 'ca',  
        'Sector': 'Technology',  
        'Position_Encoded': 2.0,
```

가장 가까운 클러스터: cluster#0

해당 클러스터에서 가장 가까운 공고 Top 10:

	Company Name	Job Title	Position_Encoded	Location_state_binned	Sector	avg_salary	Company_age	Rating
34	Esri	Data Science	2.0	ca	Technology	107.0	56.0	3.5
434	Mentor Graphics	Data Science	2.0	ca	Technology	107.0	44.0	4.1
31	Juniper Networks	Data Science	2.0	ca	Technology	121.0	29.0	3.8
386	Brillient	Data Science	2.0	Other	Technology	86.0	19.0	3.7
33	Clarity Insights	Data Analysis	2.0	ca	Technology	106.0	17.0	4.2
332	Esri	Data Science	2.0	ca	Technology	68.5	56.0	3.5
38	Sauce Labs	Data Analysis	2.0	ca	Technology	88.0	17.0	4.2
81	goTRG	Data Science	2.0	Other	Technology	85.5	17.0	4.2
108	Autodesk	Data Engineering	2.0	ca	Technology	108.0	43.0	4.0
473	SoftBank Robotics	Data Science	2.0	ca	Industrial & Energy	103.0	20.0	3.8

## 최종 결론

### 1. 회귀모델을 통한 기업 특성과 급여 간 상관관계 분석

→ Random Forest가 가장 높은 설명력을 보였고, 기업의 규모, 본사 위치, 산업군이 avg\_salary에 큰 영향을 미침을 확인함

### 2. 기업 특성과 연봉 간의 다층적 관계 분석 및 예측 (회귀+분류)

→ 기업 평점, 업종, 위치 등의 변수가 연봉 예측에 지속적으로 영향력 있게 작용함을 확인함

### 3. 연봉 예측 시스템 구축

→ 주요 영향 변수는 회사 위치, 직무 Position, 회사 규모임을 확인함

### 4. 기업 평점 예측 (회귀 + 분류)

→ 회사 규모, 업종, 소유형태 등 비재무적 특성이 평점에 주요한 영향이 있음을 확인함

### 5. 클러스터링을 통한 공고 추천 시스템

→ 특정 클러스터는 '고연봉+대도시+상위직급' 공고가 집중됨을 파악하고, 사용자별 맞춤 추천 방향 도출 가능함을 확인

## 프로젝트를 통해 얻은 인사이트

### 구준서

클러스터링을 통해  
연봉과 직무 특성별  
그룹을 나누는  
과정에서 데이터  
기반 맞춤형 추천의  
가능성을 실감했다.

### 김상준

다양한 회귀 모델  
실험을 통해 가장  
성능이 좋은 조합을  
찾는 과정이  
흥미로웠고, 피처  
선택의 중요성을  
체감했다.

### 박선인

기업 평점 예측에서  
비재무 정보가  
중요한 인사이트가  
된 점이 흥미로웠고,  
분류 모델의 활용  
가능성을 확인할 수  
있었다.

### 이하은

다양한 전처리  
기법과 모델 비교를  
통해 기업 특성이  
연봉에 어떻게  
영향을 주는지  
정량적으로 파악할  
수 있어 의미 있었다.

### 한웅재

회귀·분류를 함께  
실험하며 데이터  
분포와 전처리에  
따라 성능이  
달라지는 경험이<sup>1</sup>  
인상 깊었고, 실제  
적용의 중요성을  
느꼈다.

### 최종 결론

- 데이터 전처리의 중요성을 체감하며, 결측치·이상치 처리가 모델 성능에 큰 영향을 미친다는 점을 배웠다.
- 여러 회귀 및 앙상블 모델 실험을 통해 하이퍼파라미터 튜닝과 교차 검증의 중요성을 경험했다.
- 실제 데이터를 다루면서 예측 정확도보다 분석의 흐름과 해석의 일관성이 더 중요함을 느꼈다.
- 전체 과정을 통해 데이터과학은 기술보다 사고력과 해석력이 핵심임을 깊이 깨달았다.

For more info, visit

[https://github.com/jsk4581/DS\\_jobs\\_glassdoor\\_Analysis](https://github.com/jsk4581/DS_jobs_glassdoor_Analysis)

데이터 과학

Group 9

감사합니다!