

[1]: #Final Project

[6]:

```
import pandas as pd
import numpy as np
```

[8]:

```
# Define file paths (Update these if filenames change)
Employee= pd.read_csv("/Users/rahmasaadawy/Downloads/Final_Project/HR/Employee.csv")
PerformanceRating= pd.read_csv("/Users/rahmasaadawy/Downloads/Final_Project/HR/PerformanceRating.csv")
RatingLevel= pd.read_csv("/Users/rahmasaadawy/Downloads/Final_Project/HR/RatingLevel.csv")
SatisfiedLevel= pd.read_csv("/Users/rahmasaadawy/Downloads/Final_Project/HR/SatisfiedLevel.csv")
EducationLevel= pd.read_csv("/Users/rahmasaadawy/Downloads/Final_Project/HR/EducationLevel.csv")
```

[10...]

```
#Searching for Null values
print(Employee.isnull().sum())
print(PerformanceRating.isnull().sum())
print(RatingLevel.isnull().sum())
print(SatisfiedLevel.isnull().sum())
print(EducationLevel.isnull().sum())
```

```
EmployeeID          0
FirstName          0
LastName           0
Gender             0
Age                0
BusinessTravel     0
Department         0
DistanceFromHome (KM) 0
State              0
Ethnicity          0
Education          0
EducationField      0
JobRole            0
MaritalStatus       0
Salary              0
StockOptionLevel    0
OverTime            0
HireDate            0
Attrition           0
YearsAtCompany      0
YearsInMostRecentRole 0
YearsSinceLastPromotion 0
YearsWithCurrManager 0
dtype: int64
PerformanceID       0
EmployeeID          0
ReviewDate          0
EnvironmentSatisfaction 0
JobSatisfaction     0
RelationshipSatisfaction 0
TrainingOpportunitiesWithinYear 0
TrainingOpportunitiesTaken 0
WorkLifeBalance      0
SelfRating           0
ManagerRating        0
dtype: int64
RatingID            0
RatingLevel          0
dtype: int64
SatisfactionID      0
SatisfactionLevel    0
dtype: int64
EducationLevelID     0
EducationLevel        0
dtype: int64
```



```
[12... #Searching for duplicated values
print(Employee.duplicated().sum())
print(PerformanceRating.duplicated().sum())
print(RatingLevel.duplicated().sum())
print(SatisfiedLevel.duplicated().sum())
print(EducationLevel.duplicated().sum())
```

```
0
0
0
0
0
```

```
[14... # Convert reviewdate to datetime
PerformanceRating['ReviewDate'] = pd.to_datetime(PerformanceRating['ReviewDate'], errors='coerce')

# Confirm the change
print(PerformanceRating.dtypes)
```

```
PerformanceID          object
EmployeeID             object
ReviewDate            datetime64[ns]
EnvironmentSatisfaction    int64
JobSatisfaction        int64
RelationshipSatisfaction    int64
TrainingOpportunitiesWithinYear    int64
TrainingOpportunitiesTaken        int64
WorkLifeBalance         int64
SelfRating              int64
ManagerRating           int64
dtype: object
```

```
[16... #Merging Primary tables
merged_df = pd.merge(Employee, PerformanceRating, on="EmployeeID")

#Merging secondary tables
education_level = pd.DataFrame({
    "EducationLevelID": [1, 2, 3, 4, 5],
    "EducationLevel": ["No Formal Qualifications", "High School", "Bachelors", "Masters", "Doctorate"]
})
merged_df = pd.merge(merged_df, education_level, left_on="Education", right_on="EducationLevelID", how=
```

```
[18...]
# Merge Employee with PerformanceRating (Primary tables)
merged_df = pd.merge(Employee, PerformanceRating, on='EmployeeID', how='left')

# Merge with Satisfaction Levels (EnvironmentSatisfaction)
merged_df = pd.merge(merged_df, SatisfiedLevel, left_on='EnvironmentSatisfaction', right_on='SatisfactionID', how='left')

# Merge with Rating Levels (ManagerRating)
merged_df = pd.merge(merged_df, RatingLevel, left_on='ManagerRating', right_on='RatingID', how='left')

# Merge with Education Levels (Education)
merged_df = pd.merge(merged_df, EducationLevel, left_on='Education', right_on='EducationLevelID', how='left')

# Drop duplicate key columns
merged_df.drop(columns=['SatisfactionID', 'RatingID', 'EducationLevelID'], inplace=True)

# Check merged dataset structure
print(merged_df.info())

# Save the cleaned and merged dataset
merged_df.to_csv("Cleaned_Data.csv", index=False)

#Download the cleansed Data
merged_df.to_csv ("/Users/rahmasaadawy/Downloads/Cleaned_HR_Data.csv")
merged_df.to_excel("/Users/rahmasaadawy/Downloads/Cleaned_HR_Data.xlsx", index=False)

print(f"✅ Cleaned dataset saved successfully to: {""/Users/rahmasaadawy/Downloads/Cleaned_HR_Data.csv"}
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6899 entries, 0 to 6898
Data columns (total 36 columns):

#	Column	Non-Null Count	Dtype
0	EmployeeID	6899	non-null object
1	FirstName	6899	non-null object
2	LastName	6899	non-null object
3	Gender	6899	non-null object
4	Age	6899	non-null int64
5	BusinessTravel	6899	non-null object
6	Department	6899	non-null object
7	DistanceFromHome (KM)	6899	non-null int64
8	State	6899	non-null object
9	Ethnicity	6899	non-null object
10	Education	6899	non-null int64
11	EducationField	6899	non-null object
12	JobRole	6899	non-null object
13	MaritalStatus	6899	non-null object
14	Salary	6899	non-null int64
15	StockOptionLevel	6899	non-null int64
16	Overtime	6899	non-null object
17	HireDate	6899	non-null object
18	Attrition	6899	non-null object
19	YearsAtCompany	6899	non-null int64
20	YearsInMostRecentRole	6899	non-null int64
21	YearsSinceLastPromotion	6899	non-null int64
22	YearsWithCurrManager	6899	non-null int64
23	PerformanceID	6709	non-null object
24	ReviewDate	6709	non-null datetime64[ns]
25	EnvironmentSatisfaction	6709	non-null float64
26	JobSatisfaction	6709	non-null float64
27	RelationshipSatisfaction	6709	non-null float64
28	TrainingOpportunitiesWithinYear	6709	non-null float64
29	TrainingOpportunitiesTaken	6709	non-null float64
30	WorkLifeBalance	6709	non-null float64
31	SelfRating	6709	non-null float64
32	ManagerRating	6709	non-null float64
33	SatisfactionLevel	6709	non-null object
34	RatingLevel	6709	non-null object
35	EducationLevel	6899	non-null object

dtypes: datetime64[ns](1), float64(8), int64(9), object(18)

memory usage: 1.9+ MB

None

✅ Cleaned dataset saved successfully to: /Users/rahmasaadawy/Downloads/Cleaned_HR_Data.csv

```
[20... # Check the structure of the dataset
print(merged_df.info())
print(merged_df.head())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6899 entries, 0 to 6898
Data columns (total 36 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   EmployeeID      6899 non-null   object  
 1   FirstName        6899 non-null   object  
 2   LastName         6899 non-null   object  
 3   Gender           6899 non-null   object  
 4   Age              6899 non-null   int64   
 5   BusinessTravel   6899 non-null   object  
 6   Department       6899 non-null   object  
 7   DistanceFromHome (KM) 6899 non-null   int64   
 8   State            6899 non-null   object  
 9   Ethnicity        6899 non-null   object  
 10  Education        6899 non-null   int64   
 11  EducationField   6899 non-null   object  
 12  JobRole          6899 non-null   object  
 13  MaritalStatus    6899 non-null   object  
 14  Salary           6899 non-null   int64   
 15  StockOptionLevel 6899 non-null   int64   
 16  OverTime         6899 non-null   object  
 17  HireDate         6899 non-null   object  
 18  Attrition        6899 non-null   object  
 19  YearsAtCompany   6899 non-null   int64   
 20  YearsInMostRecentRole 6899 non-null   int64   
 21  YearsSinceLastPromotion 6899 non-null   int64   
 22  YearsWithCurrManager 6899 non-null   int64   
 23  PerformanceID    6709 non-null   object  
 24  ReviewDate       6709 non-null   datetime64[ns] 
 25  EnvironmentSatisfaction 6709 non-null   float64  
 26  JobSatisfaction  6709 non-null   float64  
 27  RelationshipSatisfaction 6709 non-null   float64  
 28  TrainingOpportunitiesWithinYear 6709 non-null   float64  
 29  TrainingOpportunitiesTaken 6709 non-null   float64  
 30  WorkLifeBalance   6709 non-null   float64  
 31  SelfRating        6709 non-null   float64  
 32  ManagerRating     6709 non-null   float64  
 33  SatisfactionLevel 6709 non-null   object  
 34  RatingLevel       6709 non-null   object  
 35  EducationLevel    6899 non-null   object  
dtypes: datetime64[ns](1), float64(8), int64(9), object(18)
memory usage: 1.9+ MB
None
```

EmployeeID	FirstName	LastName	Gender	Age	BusinessTravel	Department	
0	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales
1	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales
2	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales
3	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales
4	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales

DistanceFromHome (KM)	State	Ethnicity	...	JobSatisfaction
0	27	IL	White	3.0
1	27	IL	White	4.0
2	27	IL	White	5.0
3	27	IL	White	3.0
4	27	IL	White	4.0

RelationshipSatisfaction	TrainingOpportunitiesWithinYear
0	2.0
1	5.0
2	4.0
3	2.0
4	2.0

TrainingOpportunitiesTaken	WorkLifeBalance	SelfRating	ManagerRating
0	0.0	4.0	3.0
1	1.0	2.0	3.0
2	0.0	4.0	5.0
3	1.0	3.0	5.0
4	0.0	3.0	4.0

SatisfactionLevel	RatingLevel	EducationLevel
0 Neutral	Meets Expectation	Doctorate
1 Satisfied	Needs Improvement	Doctorate
2 Very Satisfied	Above and Beyond	Doctorate
3 Very Dissatisfied	Exceeds Expectation	Doctorate
4 Neutral	Meets Expectation	Doctorate

[5 rows x 36 columns]

[23...

#####
Ending of Week 1: Build Data Model, Data Cleaning and Preprocessing

```
[25... ##### Week 2: Analysis Questions Phase
[22... import matplotlib.pyplot as plt
import seaborn as sns
[23... # Load the cleaned dataset
cleaned_data_path = "/Users/rahmasaadawy/Downloads/Cleaned_HR_Data.csv"
merged_df = pd.read_csv("/Users/rahmasaadawy/Downloads/Cleaned_HR_Data.csv", index_col=0)
merged_df
```

	EmployeeID	FirstName	LastName	Gender	Age	BusinessTravel	Department	DistanceFromHome (KM)	Sta
0	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales	27	
1	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales	27	
2	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales	27	
3	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales	27	
4	3012-1A41	Leonelle	Simco	Female	30	Some Travel	Sales	27	
...	
6894	467E-977A	Jud	Melanaphy	Male	20	Some Travel	Technology	28	
6895	6FB9-A624	Marc	Calver	Non-Binary	27	Some Travel	Technology	8	
6896	EBF4-5928	Rudolph	MacDearmont	Male	21	Some Travel	Sales	4	
6897	60E6-B1D9	Merill	Agg	Male	21	Some Travel	Technology	7	
6898	84D4-D4C3	Naoma	Hebbard	Female	20	No Travel	Technology	28	

6899 rows × 36 columns

```
[26... # First Category
# Employee Demographics & Salary Analysis
[33... #1.1 Calculate and display the total number of employees
total_employees = merged_df["EmployeeID"].nunique()
print(f"Total Number of Employees: {total_employees}")
Total Number of Employees: 1470
```

```
[35... #1.2 Count unique employees by gender
unique_gender_counts = merged_df.groupby("EmployeeID")["Gender"].first().value_counts()

print("Unique Number of Male and Female Employees:")
print(unique_gender_counts)

Unique Number of Male and Female Employees:
Gender
Female          675
Male           651
Non-Binary      124
Prefer Not To Say   20
Name: count, dtype: int64
```

```
[37... #1.3 Count unique employees by department
unique_department_counts = merged_df.groupby("EmployeeID")["Department"].first().value_counts()

print("Unique Number of Employees in Each Department:")
print(unique_department_counts)

# Visualize the unique number of employees in each department
# Set the plot style
sns.set_style("whitegrid")

# Create a bar plot for the unique employee count by department
plt.figure(figsize=(8, 6))
sns.barplot(x=unique_department_counts.values, y=unique_department_counts.index, palette="muted")

# Add labels and title
plt.title("Unique Number of Employees in Each Department", fontsize=14)
plt.xlabel("Number of Employees", fontsize=12)
plt.ylabel("Department", fontsize=12)

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

Unique Number of Employees in Each Department:

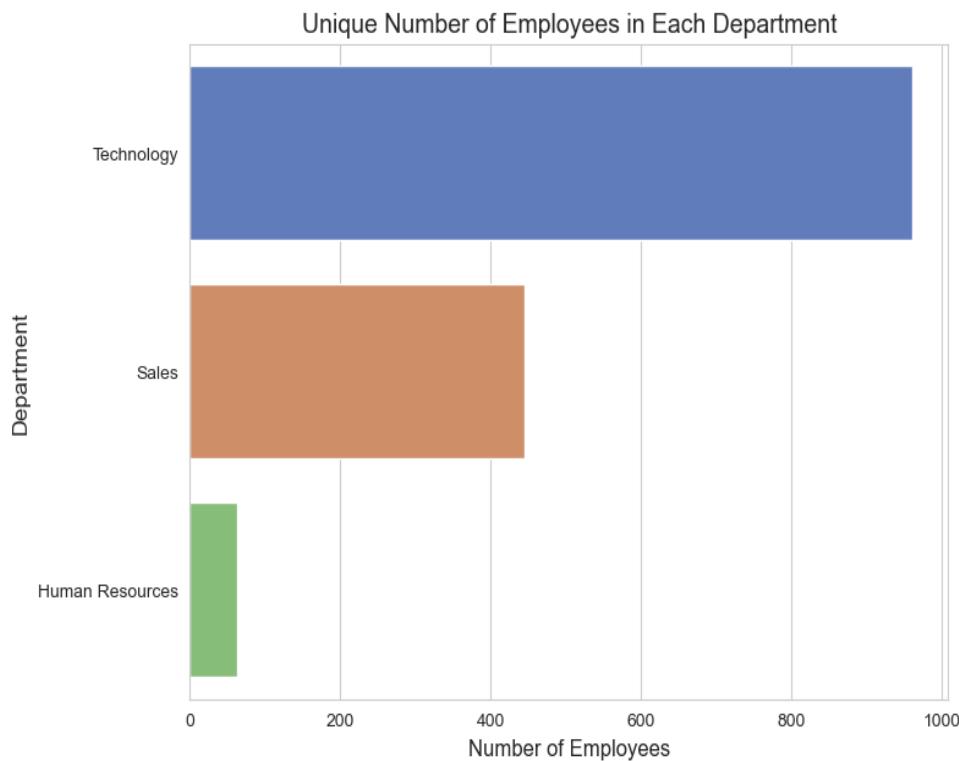
Department	Count
Technology	961
Sales	446
Human Resources	63

Name: count, dtype: int64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw000gn/T/ipykernel_10217/254342271.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=unique_department_counts.values, y=unique_department_counts.index, palette="muted")
```



```
[39... #1.4 Count unique employees by gender within each department
unique_gender_department_counts = merged_df.groupby(["Department", "EmployeeID"])["Gender"].first().reset_index()
gender_by_department = unique_gender_department_counts.groupby(["Department", "Gender"]).size().unstack()

print("Unique Number of Employees by Gender in Each Department:")
print(gender_by_department)

# Visualize gender distribution within each department
plt.figure(figsize=(10, 7))
gender_by_department.plot(kind="bar", stacked=True, colormap="Set2", figsize=(10, 7))

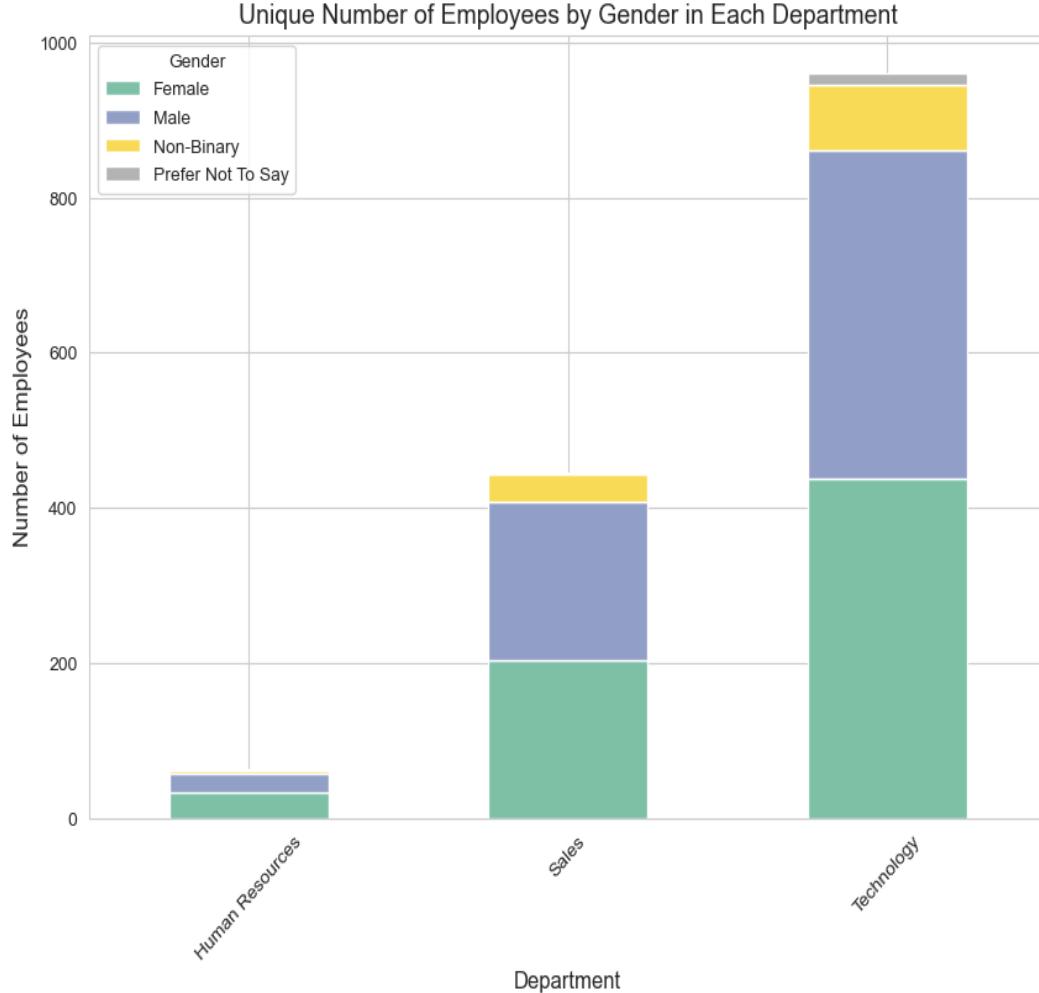
plt.title("Unique Number of Employees by Gender in Each Department", fontsize=14)
plt.xlabel("Department", fontsize=12)
plt.ylabel("Number of Employees", fontsize=12)
plt.xticks(rotation=45)
plt.legend(title="Gender")

plt.show()
```

Unique Number of Employees by Gender in Each Department:

Gender	Female	Male	Non-Binary	Prefer Not To Say
Human Resources	33	24	5	1
Sales	204	204	35	3
Technology	438	423	84	16

<Figure size 1000x700 with 0 Axes>



```
[41... #1.5 Count unique employees by education level
unique_education_counts = merged_df.groupby("EmployeeID")["EducationLevel"].first().value_counts()

print("Unique Number of Employees by Education Level:")
print(unique_education_counts)

# Visualize the distribution of employees by education level
plt.figure(figsize=(8, 6))
sns.barplot(x=unique_education_counts.values, y=unique_education_counts.index, palette="muted")

plt.title("Distribution of Employees by Education Level", fontsize=14)
plt.xlabel("Number of Employees", fontsize=12)
plt.ylabel("Education Level", fontsize=12)

plt.show()
```

Unique Number of Employees by Education Level:

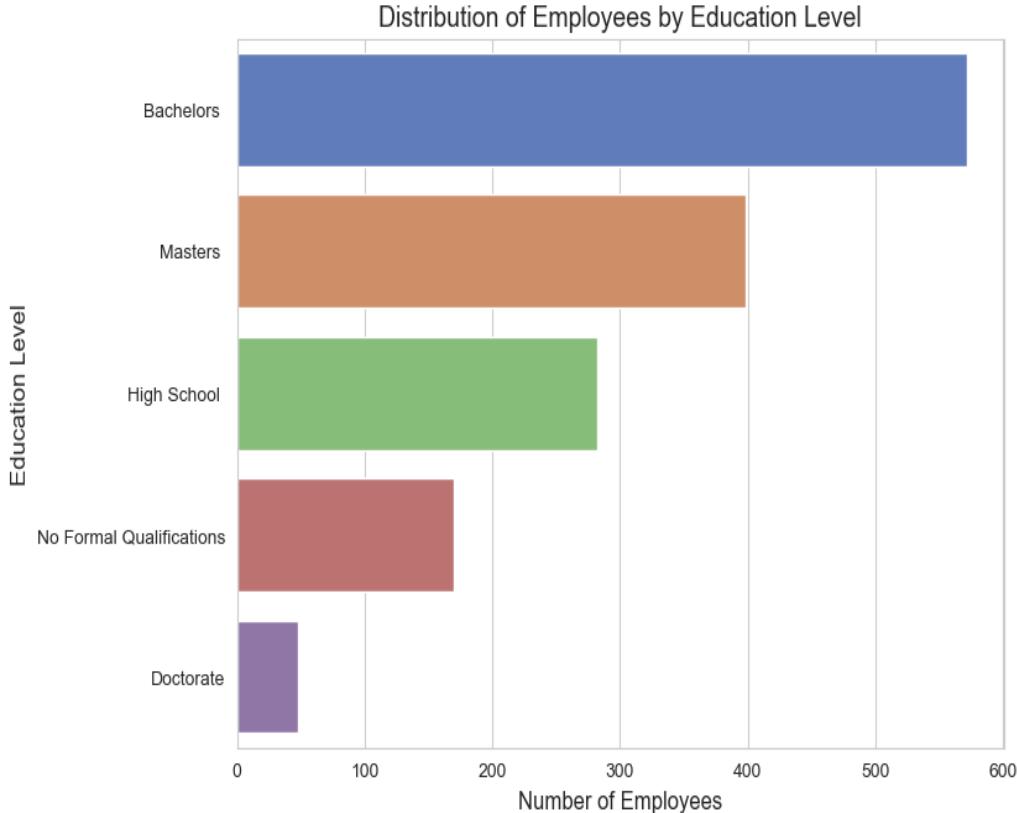
EducationLevel	count
Bachelors	572
Masters	398
High School	282
No Formal Qualifications	170
Doctorate	48

Name: count, dtype: int64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/433170451.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=unique_education_counts.values, y=unique_education_counts.index, palette="muted")
```



```
[43... #1.6 Count unique employees by job role
unique_jobrole_counts = merged_df.groupby("EmployeeID")["JobRole"].first().value_counts()

print("Unique Number of Employees by Job Role:")
print(unique_jobrole_counts)

# Visualize the distribution of employees by job role
plt.figure(figsize=(10, 8))
sns.barplot(x=unique_jobrole_counts.values, y=unique_jobrole_counts.index, palette="Set2")

plt.title("Distribution of Employees by Job Role", fontsize=14)
plt.xlabel("Number of Employees", fontsize=12)
plt.ylabel("Job Role", fontsize=12)

plt.show()
```

Unique Number of Employees by Job Role:

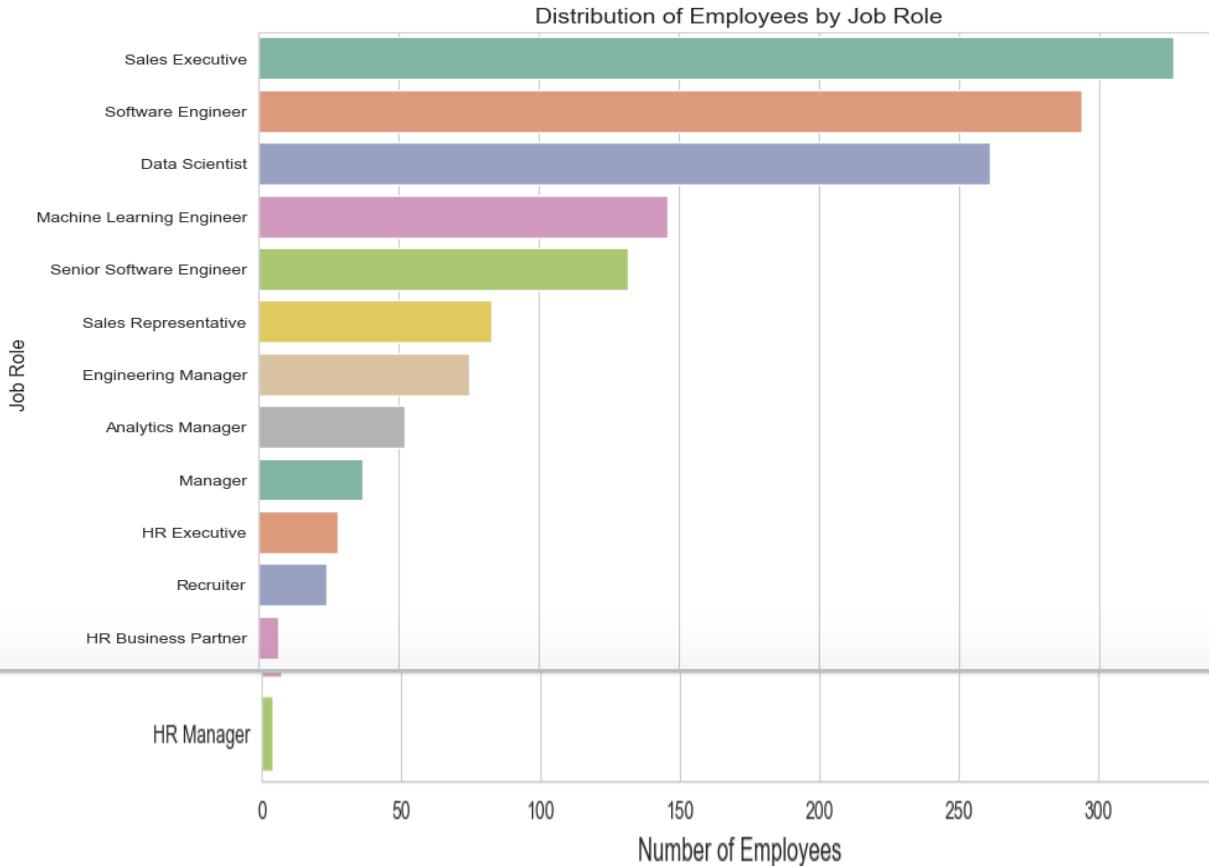
JobRole	Count
Sales Executive	327
Software Engineer	294
Data Scientist	261
Machine Learning Engineer	146
Senior Software Engineer	132
Sales Representative	83
Engineering Manager	75
Analytics Manager	52
Manager	37
HR Executive	28
Recruiter	24
HR Business Partner	7
HR Manager	4

Name: count, dtype: int64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/4202081888.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=unique_jobrole_counts.values, y=unique_jobrole_counts.index, palette="Set2")
```



```
[45... # 2. How does the average salary vary by education level?
avg_salary_by_education = merged_df.groupby("EmployeeID")[["EducationLevel", "Salary"]].first().groupby("EducationLevel")
print("Average Salary by Education Level:")
print(avg_salary_by_education)

# Convert the series to DataFrame
avg_salary_by_education = avg_salary_by_education.reset_index()

# Visualize the average salary by education level
plt.figure(figsize=(8, 6))
sns.barplot(x='Salary', y='EducationLevel', data=avg_salary_by_education, palette="viridis")

plt.title("Average Salary by Education Level", fontsize=14)
plt.xlabel("Average Salary", fontsize=12)
plt.ylabel("Education Level", fontsize=12)

plt.show()
```

Average Salary by Education Level:

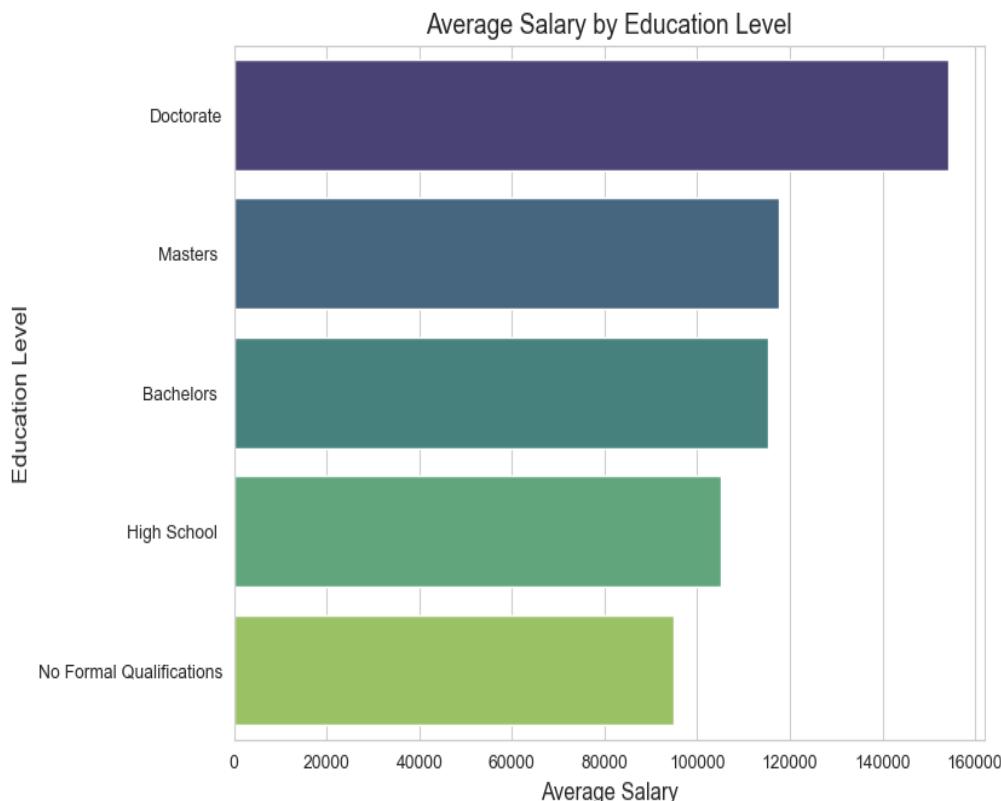
EducationLevel	Average Salary
Doctorate	154268.791667
Masters	117641.057789
Bachelors	115405.430070
High School	105180.535461
No Formal Qualifications	94983.482353

Name: Salary, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/106904704.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x='Salary', y='EducationLevel', data=avg_salary_by_education, palette="viridis")
```



[49...]

```
# 3. Count unique employees by job role
unique_jobrole_counts = merged_df.groupby("EmployeeID")["JobRole"].first().value_counts()

print("Unique Number of Employees by Job Role:")
print(unique_jobrole_counts)

# Visualize the distribution of employees by job role
plt.figure(figsize=(10, 8))
sns.barplot(x=unique_jobrole_counts.values, y=unique_jobrole_counts.index, palette="Set2")

plt.title("Distribution of Employees by Job Role", fontsize=14)
plt.xlabel("Number of Employees", fontsize=12)
plt.ylabel("Job Role", fontsize=12)

plt.show()
```

Unique Number of Employees by Job Role:

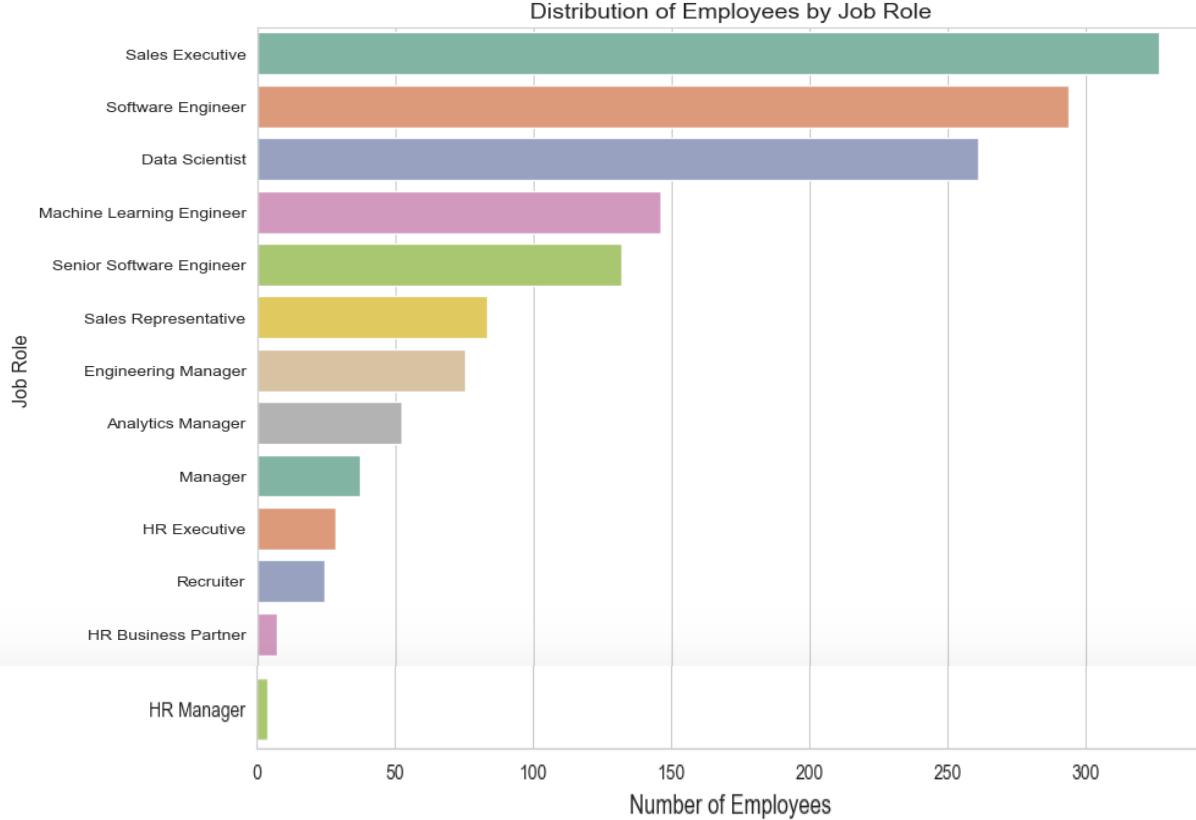
JobRole	Count
Sales Executive	327
Software Engineer	294
Data Scientist	261
Machine Learning Engineer	146
Senior Software Engineer	132
Sales Representative	83
Engineering Manager	75
Analytics Manager	52
Manager	37
HR Executive	28
Recruiter	24
HR Business Partner	7
HR Manager	4

Name: count, dtype: int64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/631614572.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=unique_jobrole_counts.values, y=unique_jobrole_counts.index, palette="Set2")
```



ZC

```
[51... # 3. Calculate the average salary by job role for unique employees
avg_salary_by_jobrole = merged_df.groupby("EmployeeID")[["JobRole", "Salary"]].first().groupby("JobRole")

print("Average Salary by Job Role:")
print(avg_salary_by_jobrole)

# Visualize the average salary by job role
plt.figure(figsize=(12, 8))
sns.barplot(x=avg_salary_by_jobrole.values, y=avg_salary_by_jobrole.index, palette="viridis")

plt.title("Average Salary by Job Role", fontsize=14)
plt.xlabel("Average Salary", fontsize=12)
plt.ylabel("Job Role", fontsize=12)

plt.show()
```

Average Salary by Job Role:

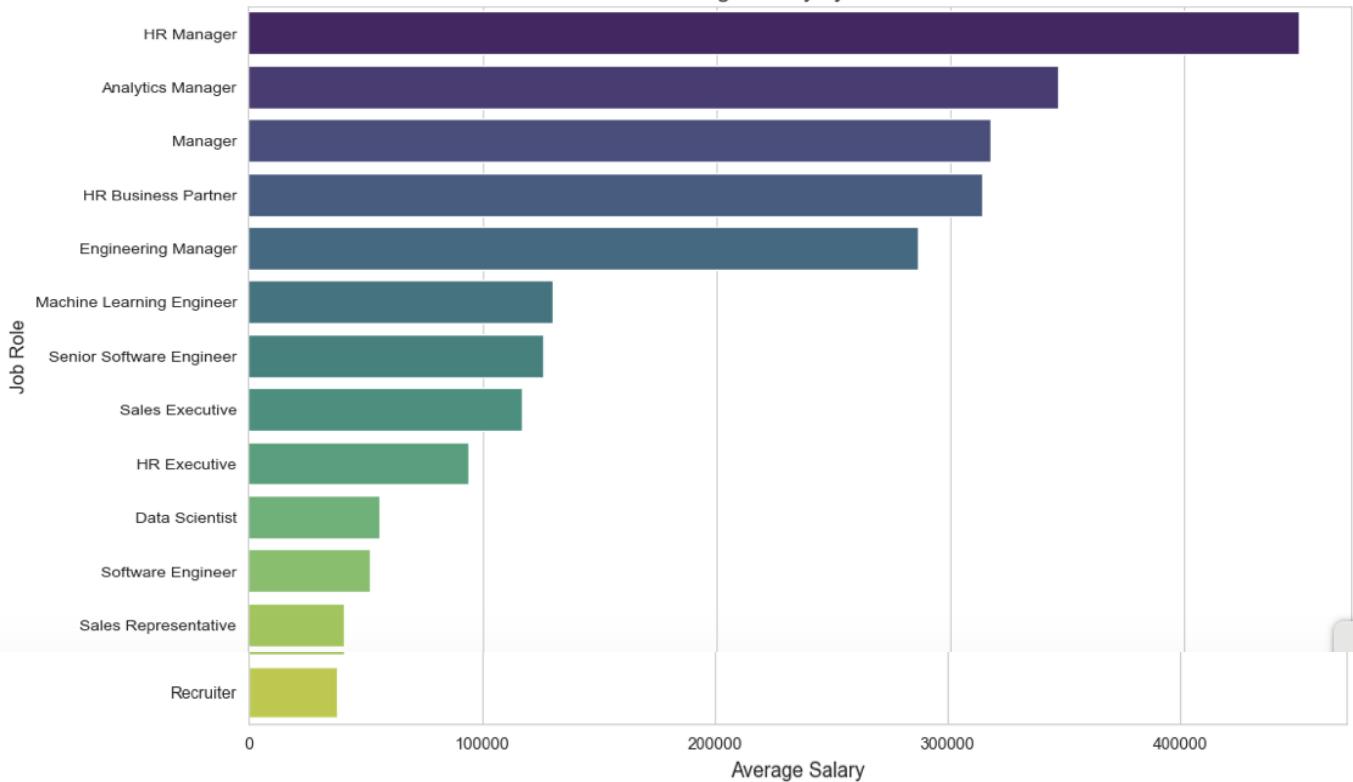
JobRole	
HR Manager	449330.750000
Analytics Manager	346484.230769
Manager	317531.054054
HR Business Partner	314002.428571
Engineering Manager	286258.506667
Machine Learning Engineer	130164.616438
Senior Software Engineer	126161.295455
Sales Executive	117195.538226
HR Executive	94362.321429
Data Scientist	56079.494253
Software Engineer	51967.051020
Sales Representative	40656.421687
Recruiter	37647.500000

Name: Salary, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/952937992.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=avg_salary_by_jobrole.values, y=avg_salary_by_jobrole.index, palette="viridis")  
Average Salary by Job Role
```



```
[53...]
# 4. What is the salary distribution based on years of experience?
# Identify employees who have been promoted (YearsSinceLastPromotion == 0 indicates a recent promotion)
promoted_employees = merged_df[merged_df["YearsSinceLastPromotion"] == 0]

# Calculate the number of promoted employees by job role (unique employees only)
promotion_by_jobrole = promoted_employees.groupby("EmployeeID")["JobRole"].first().value_counts()

# Total unique employees by job role (for comparison)
total_by_jobrole = merged_df.groupby("EmployeeID")["JobRole"].first().value_counts()

# Calculate promotion rate by job role (% of promoted employees per role)
promotion_rate_by_jobrole = (promotion_by_jobrole / total_by_jobrole * 100).fillna(0).sort_values(ascending=True)

print("Promotion Rate by Job Role (%):")
print(promotion_rate_by_jobrole)

# Visualize promotion rate by job role
plt.figure(figsize=(12, 8))
sns.barplot(x=promotion_rate_by_jobrole.values, y=promotion_rate_by_jobrole.index, palette="coolwarm")

plt.title("Promotion Rate by Job Role (%)", fontsize=14)
plt.xlabel("Promotion Rate (%)", fontsize=12)
plt.ylabel("Job Role", fontsize=12)

plt.show()
```

Promotion Rate by Job Role (%):

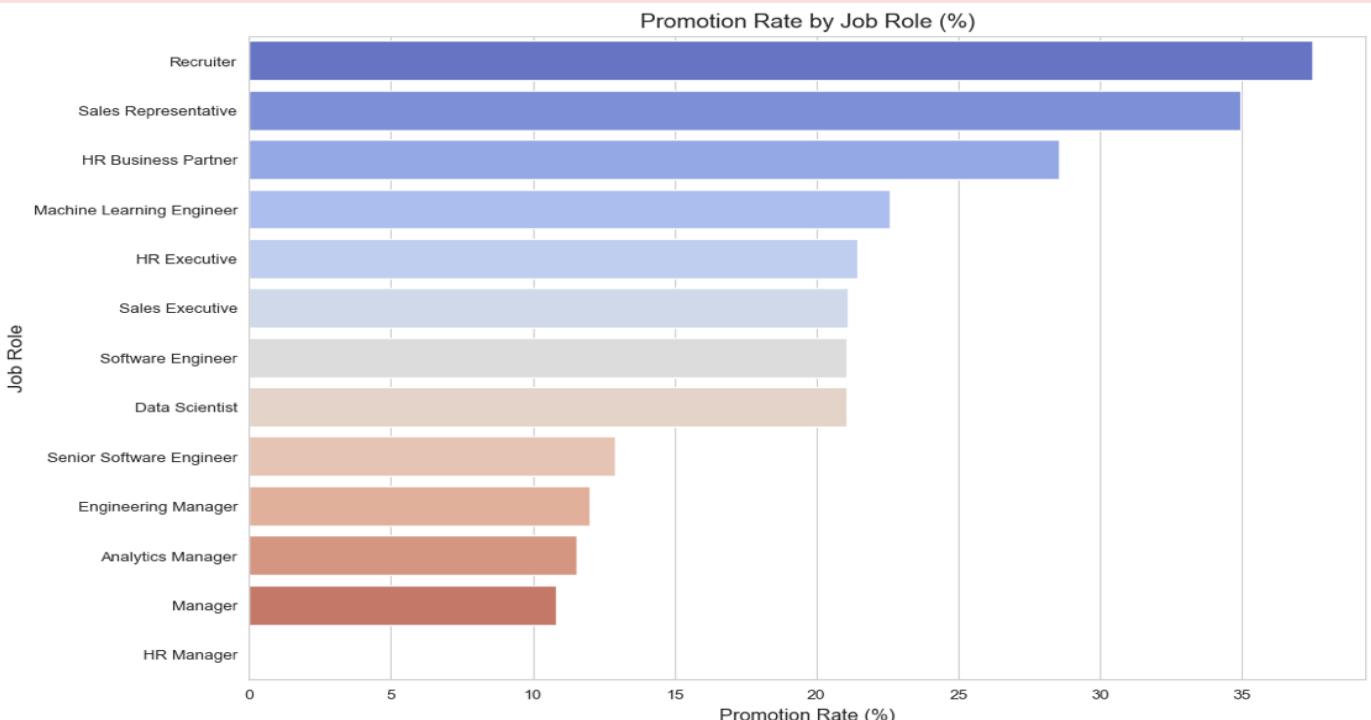
JobRole	Promotion Rate (%)
Recruiter	37.500000
Sales Representative	34.939759
HR Business Partner	28.571429
Machine Learning Engineer	22.602740
HR Executive	21.428571
Sales Executive	21.100917
Software Engineer	21.088435
Data Scientist	21.072797
Senior Software Engineer	12.878788
Engineering Manager	12.000000
Analytics Manager	11.538462
Manager	10.810811
HR Manager	0.000000

Name: count, dtype: float64

```
/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/2626743584.py:19: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=promotion_rate_by_jobrole.values, y=promotion_rate_by_jobrole.index, palette="coolwarm")
```



```
[55... # 5 Calculate the average salary by department for unique employees
avg_salary_by_department = merged_df.groupby("EmployeeID")[["Department", "Salary"]].first().groupby("I
print("Average Salary by Department:")
print(avg_salary_by_department)

# Visualize the average salary by department
plt.figure(figsize=(8, 6))
sns.barplot(x=avg_salary_by_department.values, y=avg_salary_by_department.index, palette="viridis")

plt.title("Average Salary by Department", fontsize=14)
plt.xlabel("Average Salary", fontsize=12)
plt.ylabel("Department", fontsize=12)

plt.show()
```

Average Salary by Department:

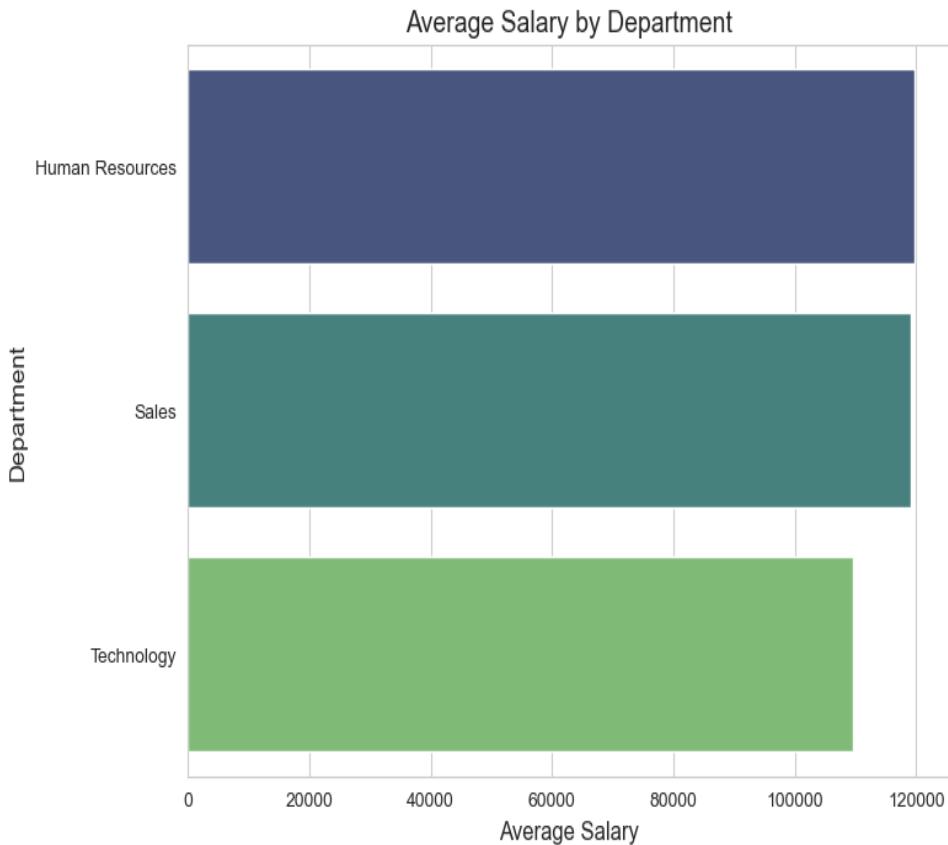
Department	Average Salary
Human Resources	119698.809524
Sales	119117.609865
Technology	109655.122789

Name: Salary, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/3839542891.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=avg_salary_by_department.values, y=avg_salary_by_department.index, palette="viridis")
```



```
[57... # Second Category
```

```
# Employee Satisfaction & Engagement
```

```
[59... # 6. What is the average satisfaction level across different job roles?
```

```
# Average satisfaction by job role
```

```
avg_satisfaction_by_role = merged_df.groupby('JobRole')['JobSatisfaction'].mean().reset_index()  
print(avg_satisfaction_by_role)
```

```
# Visualization
```

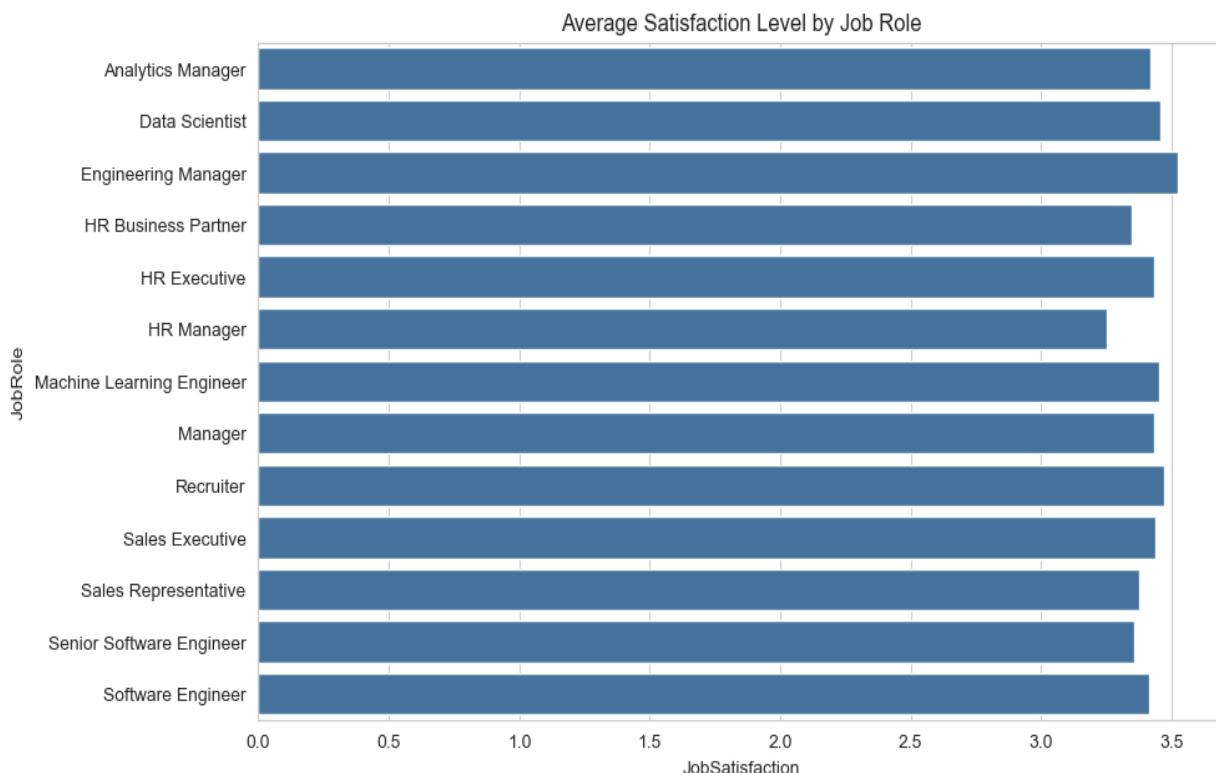
```
plt.figure(figsize=(10, 6))  
sns.barplot(x='JobSatisfaction', y='JobRole', data=avg_satisfaction_by_role, ci=None)  
plt.title("Average Satisfaction Level by Job Role")  
plt.show()
```

	JobRole	JobSatisfaction
0	Analytics Manager	3.418269
1	Data Scientist	3.457353
2	Engineering Manager	3.526490
3	HR Business Partner	3.347826
4	HR Executive	3.434783
5	HR Manager	3.250000
6	Machine Learning Engineer	3.453405
7	Manager	3.435714
8	Recruiter	3.469799
9	Sales Executive	3.435897
10	Sales Representative	3.378323
11	Senior Software Engineer	3.356275
12	Software Engineer	3.413043

```
/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/1379388627.py:8: FutureWarning:
```

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

```
sns.barplot(x='JobSatisfaction', y='JobRole', data=avg_satisfaction_by_role, ci=None)
```



[28...]

```
# 7 Calculate average salary by satisfaction level for unique employees
unique_df = merged_df.drop_duplicates(subset=[ 'EmployeeID' ])
salary_by_satisfaction = unique_df.groupby("EmployeeID")[[ "JobSatisfaction", "Salary"]].first().groupby("JobSatisfaction")

print("Average Salary by Job Satisfaction Level:")
print(salary_by_satisfaction)

# Visualizing the relationship between job satisfaction and salary
plt.figure(figsize=(10, 6))
sns.boxplot(data=merged_df, x="JobSatisfaction", y="Salary", palette="coolwarm")

plt.title("Relationship Between Job Satisfaction and Salary", fontsize=14)
plt.xlabel("Job Satisfaction Level", fontsize=12)
plt.ylabel("Salary", fontsize=12)

plt.show()
```

Average Salary by Job Satisfaction Level:

JobSatisfaction

1.0	120500.333333
2.0	116695.341463
3.0	104914.106312
4.0	113217.044444
5.0	120707.823718

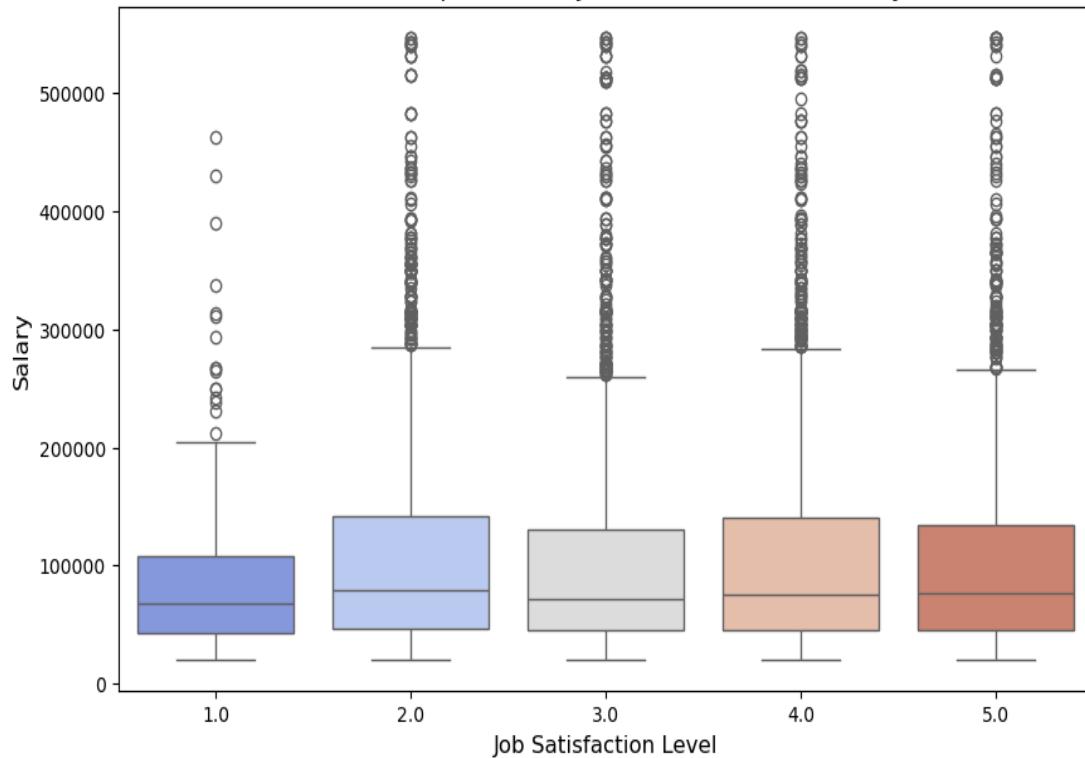
Name: Salary, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_29168/2446902177.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=merged_df, x="JobSatisfaction", y="Salary", palette="coolwarm")
```

Relationship Between Job Satisfaction and Salary



```
[65... # 8. Do employees with higher education levels report higher satisfaction?  
# Average satisfaction by education level  
satisfaction_by_edu = merged_df.groupby('EducationLevel')['JobSatisfaction'].mean().reset_index()  
print(satisfaction_by_edu)
```

	EducationLevel	JobSatisfaction
0	Bachelors	3.440015
1	Doctorate	3.298578
2	High School	3.460400
3	Masters	3.435146
4	No Formal Qualifications	3.377381

```
[69... # 9. Which departments have the most satisfied and least satisfied employees?  
# Average satisfaction by department  
satisfaction_by_dept = merged_df.groupby('Department')['JobSatisfaction'].mean().reset_index()  
print(satisfaction_by_dept.sort_values('JobSatisfaction', ascending=False))
```

	Department	JobSatisfaction
0	Human Resources	3.435644
2	Technology	3.434578
1	Sales	3.422057

```
[47... # 10. Does job role impact satisfaction level?
import pandas as pd

# Function to classify satisfaction levels based on numerical values
def classify_satisfaction(avg_satisfaction):
    rounded_value = round(avg_satisfaction) # Round to the nearest integer
    if rounded_value == 1:
        return 'Very Dissatisfied'
    elif rounded_value == 2:
        return 'Dissatisfied'
    elif rounded_value == 3:
        return 'Neutral'
    elif rounded_value == 4:
        return 'Satisfied'
    elif rounded_value == 5:
        return 'Very Satisfied'
    else:
        return 'Neutral' # Default to 'Neutral' for non-exact values

# Group data by JobRole and calculate the average JobSatisfaction
job_satisfaction_summary = merged_df.groupby("JobRole")["JobSatisfaction"].mean().reset_index()

# Round the average satisfaction value to match SQL behavior
job_satisfaction_summary["AverageSatisfaction"] = job_satisfaction_summary["JobSatisfaction"].round(6)

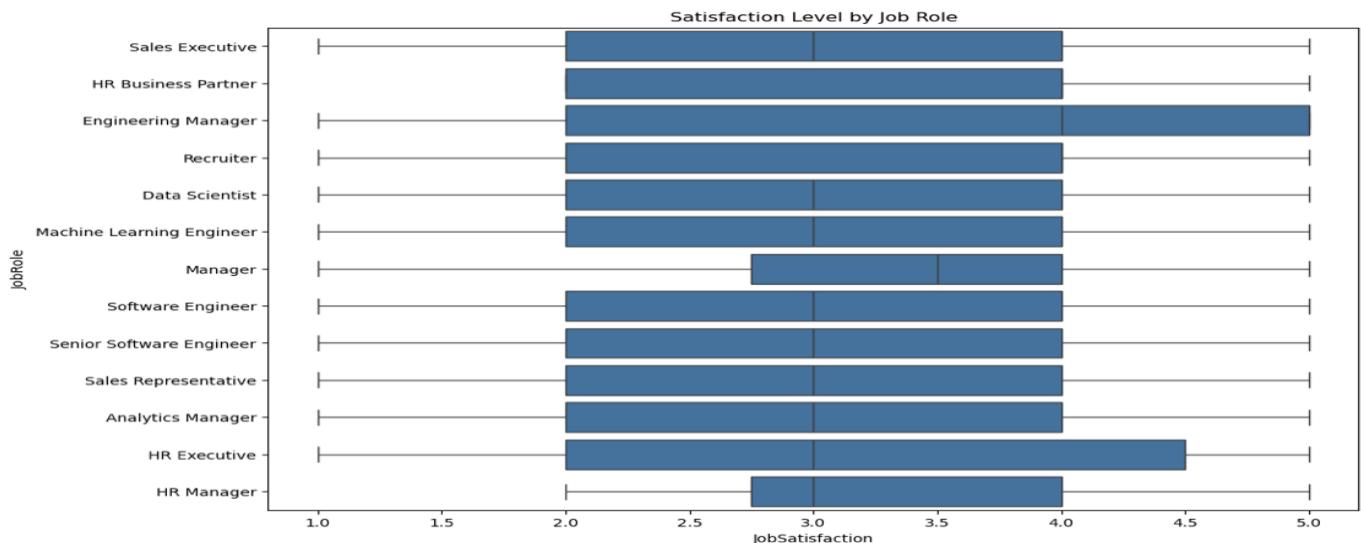
# Apply classification function to categorize satisfaction levels
job_satisfaction_summary["SatisfactionCategory"] = job_satisfaction_summary["AverageSatisfaction"].apply(classify_satisfaction)

# Print results
print("Job Role Impact on Satisfaction Level: ")
for index, row in job_satisfaction_summary.iterrows():
    print(f"\t*{row['JobRole']}: Avg Satisfaction = {row['AverageSatisfaction']:.6f}, "
          f"Category = {row['SatisfactionCategory']}")

# Satisfaction level by job role
plt.figure(figsize=(12, 8))
sns.boxplot(x='JobSatisfaction', y='JobRole', data=merged_df)
plt.title('Satisfaction Level by Job Role')
plt.show()
```

Job Role Impact on Satisfaction Level:

- *Analytics Manager: Avg Satisfaction = 3.418269, Category = Neutral
- *Data Scientist: Avg Satisfaction = 3.457353, Category = Neutral
- *Engineering Manager: Avg Satisfaction = 3.526490, Category = Satisfied
- *HR Business Partner: Avg Satisfaction = 3.347826, Category = Neutral
- *HR Executive: Avg Satisfaction = 3.434783, Category = Neutral
- *HR Manager: Avg Satisfaction = 3.250000, Category = Neutral
- *Machine Learning Engineer: Avg Satisfaction = 3.453405, Category = Neutral
- *Manager: Avg Satisfaction = 3.435714, Category = Neutral
- *Recruiter: Avg Satisfaction = 3.469799, Category = Neutral
- *Sales Executive: Avg Satisfaction = 3.435897, Category = Neutral
- *Sales Representative: Avg Satisfaction = 3.378323, Category = Neutral
- *Senior Software Engineer: Avg Satisfaction = 3.356275, Category = Neutral
- *Software Engineer: Avg Satisfaction = 3.413043, Category = Neutral



```
[73... # Third Category  
# Attrition & Turnover Analysis
```

```
[75... # 11. What is the overall employee attrition rate?  
# Attrition rate  
  
# Remove duplicate entries based on EmployeeID to ensure unique employees  
unique_df = merged_df.drop_duplicates(subset=['EmployeeID'])  
  
# Calculate the attrition rate after removing duplicates  
attrition_rate = unique_df['Attrition'].value_counts(normalize=True) * 100  
  
# Print the attrition rate  
print(attrition_rate)  
  
Attrition  
No     83.877551  
Yes    16.122449  
Name: proportion, dtype: float64
```

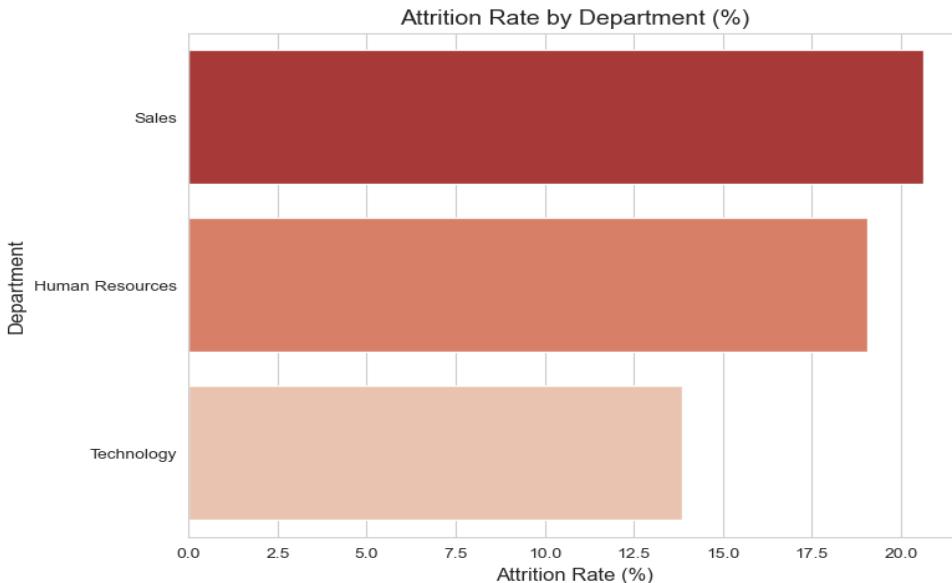
```
[77... # 12. Which department has the highest employee turnover?  
# Ensure unique employees before calculating attrition rate  
unique_employees = merged_df.drop_duplicates(subset=['EmployeeID'])  
  
# Calculate attrition rate per department  
attrition_by_dept = unique_employees.groupby('Department')['Attrition'].apply(lambda x: (x == 'Yes').mean())  
  
# Sort by attrition rate in descending order  
print(attrition_by_dept.sort_values('Attrition', ascending=False))  
  
# Visualize attrition rate by department  
sorted_data = attrition_by_dept.sort_values("Attrition", ascending=False)  
  
plt.figure(figsize=(8, 6))  
sns.barplot(x=sorted_data["Attrition"], y=sorted_data["Department"], palette="Reds_r", order=sorted_data["Department"])  
  
plt.title("Attrition Rate by Department (%)", fontsize=14)  
plt.xlabel("Attrition Rate (%)", fontsize=12)  
plt.ylabel("Department", fontsize=12)  
  
plt.show()
```

```
Department  Attrition  
1          Sales  20.627803  
0  Human Resources  19.047619  
2        Technology  13.839750
```

```
/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/1859184016.py:15: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```

```
sns.barplot(x=sorted_data["Attrition"], y=sorted_data["Department"], palette="Reds_r", order=sorted_data["Department"])
```



```
[79... # 13. Is there a connection between satisfaction level and attrition?
# Ensure unique employees by EmployeeID before calculation (drop duplicates)
unique_employees = merged_df.drop_duplicates(subset=['EmployeeID'])

# Group by SatisfactionLevel and calculate the required metrics
attrition_by_satisfaction = unique_employees.groupby('SatisfactionLevel').agg(
    TotalEmployees=('EmployeeID', 'count'), # Count total unique employees in each satisfaction level
    AttritionEmployees=('Attrition', lambda x: (x == 'Yes').sum()) # Count unique employees who have left
)

# Calculate the attrition rate for each satisfaction level
attrition_by_satisfaction['AttritionRate'] = (attrition_by_satisfaction['AttritionEmployees'] / attrition_by_satisfaction['TotalEmployees']).round(2)

# Sort by AttritionRate in descending order
attrition_by_satisfaction = attrition_by_satisfaction.sort_values('AttritionRate', ascending=False).reset_index()

# Print the result to show the attrition by satisfaction level
print(attrition_by_satisfaction)

# Visualization - Create a histogram for JobSatisfaction, distinguishing by Attrition
plt.figure(figsize=(8, 6))
sns.barplot(x='SatisfactionLevel', y='AttritionRate', data=attrition_by_satisfaction, palette='coolwarm')

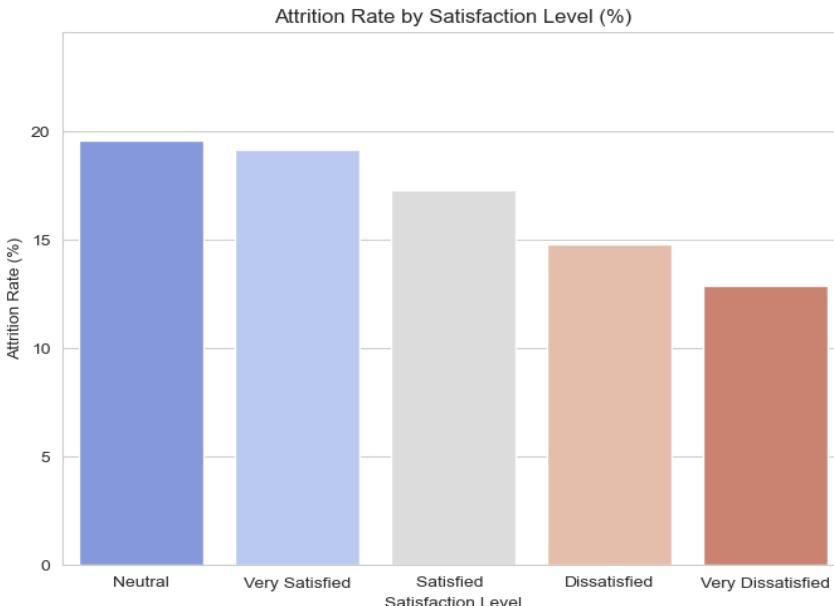
# Customize plot
plt.title('Attrition Rate by Satisfaction Level (%)')
plt.xlabel('Satisfaction Level')
plt.ylabel('Attrition Rate (%)')
plt.ylim(0, max(attrition_by_satisfaction['AttritionRate']) + 5) # Set y-limit dynamically

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

	SatisfactionLevel	TotalEmployees	AttritionEmployees	AttritionRate
0	Neutral	424	83	19.575472
1	Very Satisfied	417	80	19.184652
2	Satisfied	381	66	17.322835
3	Dissatisfied	27	4	14.814815
4	Very Dissatisfied	31	4	12.903226

```
/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/96147670.py:23: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(x='SatisfactionLevel', y='AttritionRate', data=attrition_by_satisfaction, palette='coolwarm')
```



```
[81... # 14. Do employees with higher education levels have lower attrition rates?
# Calculate attrition rate by education level using unique employees
attrition_by_education = merged_df.groupby("EmployeeID")[["EducationLevel", "Attrition"]].first()

# Count total unique employees and employees who left per education level
total_by_education = attrition_by_education["EducationLevel"].value_counts()
attrition_counts = attrition_by_education[attrition_by_education["Attrition"] == "Yes"]["EducationLevel"]

# Calculate attrition rate (% of employees who left per education level)
attrition_rate_by_education = (attrition_counts / total_by_education * 100).fillna(0).sort_values(ascending=True)

print("Attrition Rate by Education Level (%):")
print(attrition_rate_by_education)

# Visualize attrition rate by education level
plt.figure(figsize=(8, 6))
sns.barplot(x=attrition_rate_by_education.index, y=attrition_rate_by_education.values, palette="coolwarm")

plt.title("Attrition Rate by Education Level (%)", fontsize=14)
plt.xlabel("Education Level", fontsize=12)
plt.ylabel("Attrition Rate (%)", fontsize=12)

plt.show()
```

Attrition Rate by Education Level (%):

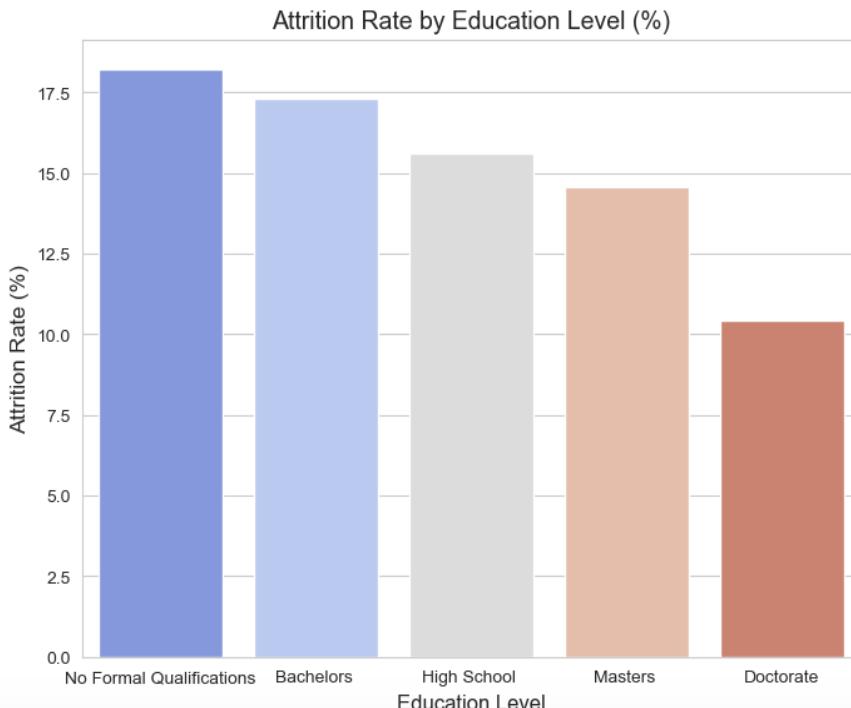
EducationLevel	Attrition Rate (%)
No Formal Qualifications	18.235294
Bachelors	17.307692
High School	15.602837
Masters	14.572864
Doctorate	10.416667

Name: count, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw000gn/T/ipykernel_10217/2642233808.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=attrition_rate_by_education.index, y=attrition_rate_by_education.values, palette="coolwarm")
```



```
[83...]
# 15. How does tenure (years at company) impact attrition?
# Calculate attrition rate by tenure (YearsAtCompany) using unique employees
attrition_by_tenure = merged_df.groupby("EmployeeID")[["YearsAtCompany", "Attrition"]].first()

# Count total unique employees and employees who left per tenure level
total_by_tenure = attrition_by_tenure["YearsAtCompany"].value_counts()
attrition_counts = attrition_by_tenure[attrition_by_tenure["Attrition"] == "Yes"]["YearsAtCompany"].value_counts()

# Calculate attrition rate (% of employees who left per tenure level)
attrition_rate_by_tenure = (attrition_counts / total_by_tenure * 100).fillna(0).sort_values(ascending=True)

print("Attrition Rate by Years at Company (%):")
print(attrition_rate_by_tenure)

# Visualize attrition rate by years at company
plt.figure(figsize=(10, 6))
sns.barplot(x=attrition_rate_by_tenure.index, y=attrition_rate_by_tenure.values, palette="coolwarm")

plt.title("Attrition Rate by Years at Company (%)", fontsize=14)
plt.xlabel("Years at Company", fontsize=12)
plt.ylabel("Attrition Rate (%)", fontsize=12)

plt.show()
```

Attrition Rate by Years at Company (%):

YearsAtCompany

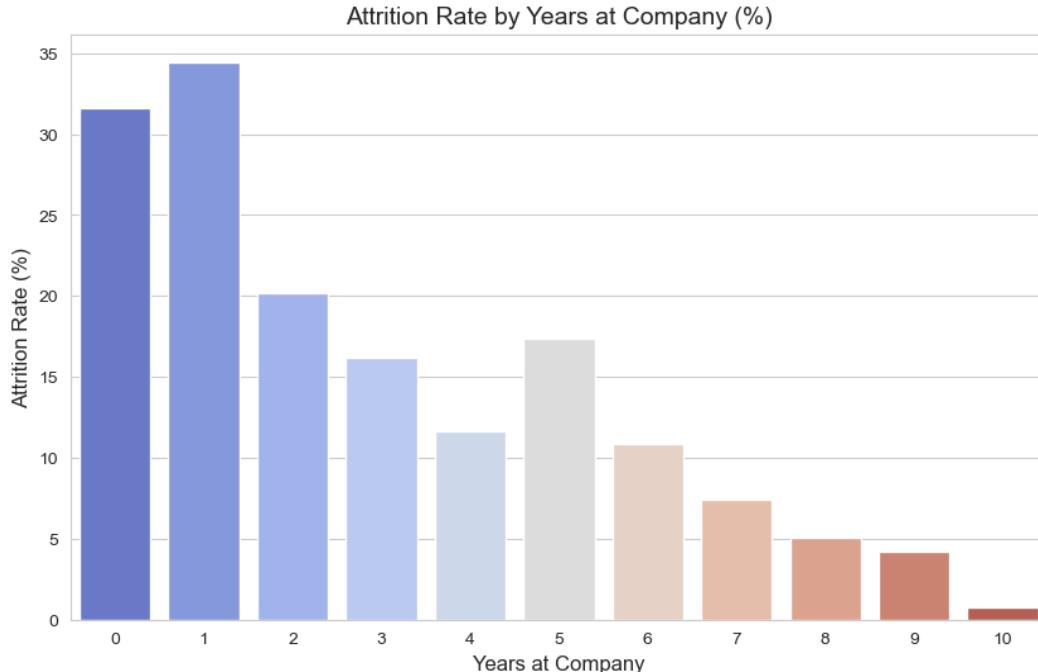
1	34.463277
0	31.578947
2	20.161290
5	17.391304
3	16.216216
4	11.627907
6	10.891089
7	7.438017
8	5.042017
9	4.237288
10	0.781250

Name: count, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/409883378.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=attrition_rate_by_tenure.index, y=attrition_rate_by_tenure.values, palette="coolwarm")
```



```
[85... # Fourth Category
```

```
# Promotion & Career Growth
```

```
[87... # 16. How long does it take, on average, for employees to receive a promotion?
```

```
# Average time to promotion
```

```
avg_time_to_promotion = merged_df.groupby("EmployeeID")["YearsSinceLastPromotion"].first().mean()
```

```
print(f"Average Time to Promotion: {avg_time_to_promotion:.2f} years")
```

```
# Visualizing distribution of years since last promotion
```

```
plt.figure(figsize=(10, 6))
```

```
sns.histplot(merged_df.groupby("EmployeeID")["YearsSinceLastPromotion"].first(), bins=10, kde=True, co
```

```
plt.title("Distribution of Years Since Last Promotion", fontsize=14)
```

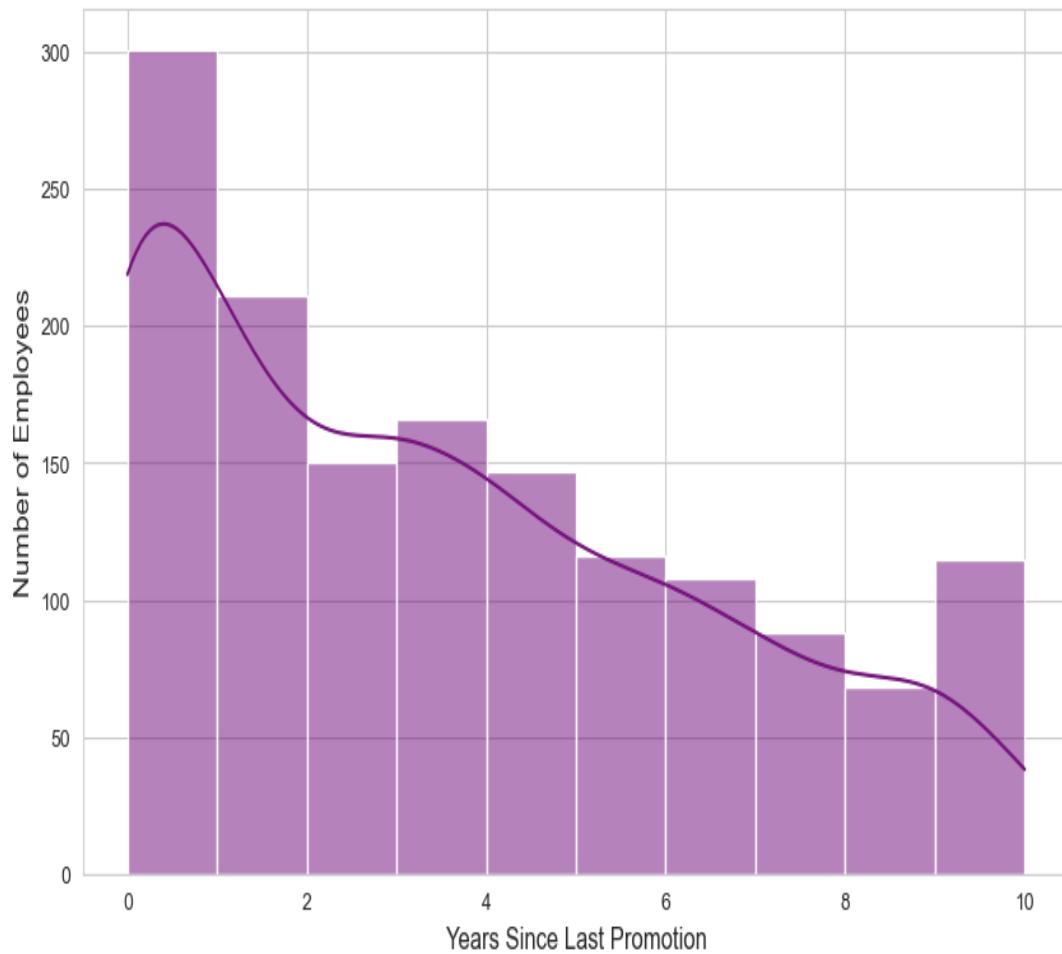
```
plt.xlabel("Years Since Last Promotion", fontsize=12)
```

```
plt.ylabel("Number of Employees", fontsize=12)
```

```
plt.show()
```

Average Time to Promotion: 3.44 years

Distribution of Years Since Last Promotion



```
[89... # 17. Is there a correlation between education level and promotion frequency?
# Promotion rate by education level
# Calculate promotion frequency by education level using unique employees
promotion_by_education = merged_df.groupby("EmployeeID") [["EducationLevel", "YearsSinceLastPromotion"]]

# Calculate the average time to promotion for each education level
avg_promotion_by_education = promotion_by_education.groupby("EducationLevel") ["YearsSinceLastPromotion"]

print("Average Time to Promotion by Education Level (Years):")
print(avg_promotion_by_education)

# Visualizing the relationship between education level and promotion frequency
plt.figure(figsize=(8, 6))
sns.barplot(x=avg_promotion_by_education.values, y=avg_promotion_by_education.index, palette="coolwarm"

plt.title("Average Time to Promotion by Education Level", fontsize=14)
plt.xlabel("Average Time to Promotion (Years)", fontsize=12)
plt.ylabel("Education Level", fontsize=12)

plt.show()
```

Average Time to Promotion by Education Level (Years):

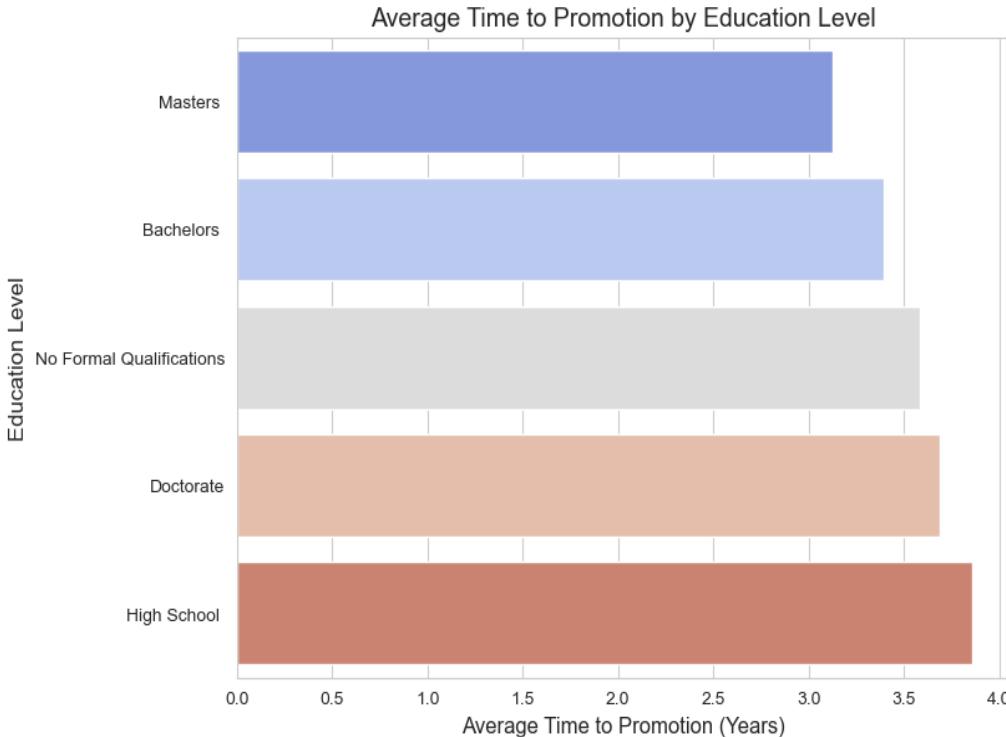
EducationLevel	Average Time to Promotion (Years)
Masters	3.123116
Bachelors	3.393357
No Formal Qualifications	3.582353
Doctorate	3.687500
High School	3.858156

Name: YearsSinceLastPromotion, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/3214222776.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(x=avg_promotion_by_education.values, y=avg_promotion_by_education.index, palette="coolwarm")



```
[91... # 18. Which departments promote employees the fastest and the slowest?
# Promotion rate by department
# Calculate the average time to promotion by department using unique employees
promotion_by_department = merged_df.groupby("EmployeeID")[[ "Department", "YearsSinceLastPromotion"]].f.

# Compute the average years since last promotion for each department
avg_promotion_by_department = promotion_by_department.groupby("Department")["YearsSinceLastPromotion"]

print("Average Time to Promotion by Department (Years):")
print(avg_promotion_by_department)

# Visualizing the relationship between department and promotion speed
plt.figure(figsize=(8, 6))
sns.barplot(x=avg_promotion_by_department.values, y=avg_promotion_by_department.index, palette="Blues_r"

plt.title("Average Time to Promotion by Department (Years)", fontsize=14)
plt.xlabel("Average Time to Promotion (Years)", fontsize=12)
plt.ylabel("Department", fontsize=12)

plt.show()
```

Average Time to Promotion by Department (Years):

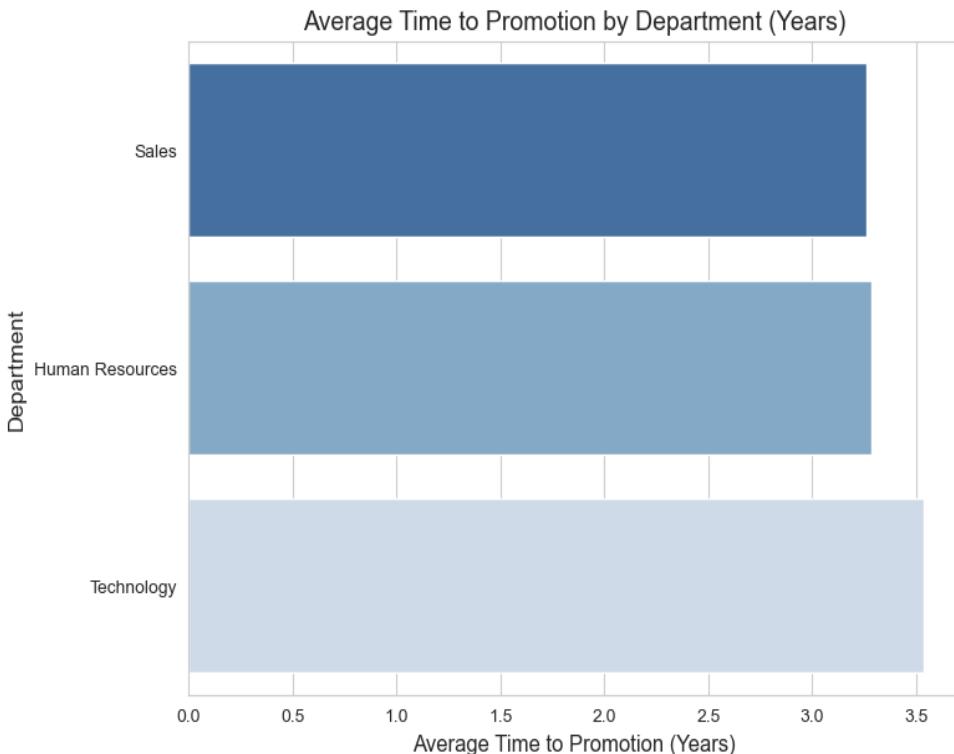
Department	Avg Time to Promotion (Years)
Sales	3.260090
Human Resources	3.285714
Technology	3.534860

Name: YearsSinceLastPromotion, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw000gn/T/ipykernel_10217/2322969403.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(x=avg_promotion_by_department.values, y=avg_promotion_by_department.index, palette="Blues_r")



```
[93... # 19. What percentage of satisfied employees receive promotions?
# Promotion rate for highly satisfied employees
# Identify promoted employees (YearsSinceLastPromotion == 0 indicates a recent promotion)
promoted_employees = merged_df[merged_df["YearsSinceLastPromotion"] == 0]

# Calculate total unique employees by satisfaction level
total_by_satisfaction = merged_df.groupby("EmployeeID")[[ "JobSatisfaction", "YearsSinceLastPromotion"]]

# Calculate number of promoted employees by satisfaction level
promoted_by_satisfaction = promoted_employees.groupby("EmployeeID")["JobSatisfaction"].first().value_counts()

# Calculate promotion rate by satisfaction level (% of promoted employees per satisfaction level)
promotion_rate_by_satisfaction = (promoted_by_satisfaction / total_by_satisfaction * 100).fillna(0).sort_index()

print("Promotion Rate by Job Satisfaction Level (%):")
print(promotion_rate_by_satisfaction)

# Visualizing promotion rate by satisfaction level
plt.figure(figsize=(8, 6))
sns.barplot(x=promotion_rate_by_satisfaction.index, y=promotion_rate_by_satisfaction.values, palette="coolwarm")

plt.title("Promotion Rate by Job Satisfaction Level (%)", fontsize=14)
plt.xlabel("Job Satisfaction Level", fontsize=12)
plt.ylabel("Promotion Rate (%)", fontsize=12)

plt.show()
```

Promotion Rate by Job Satisfaction Level (%):

JobSatisfaction

1.0 4.166667

2.0 10.060976

3.0 14.285714

4.0 12.698413

5.0 13.461538

Name: count, dtype: float64

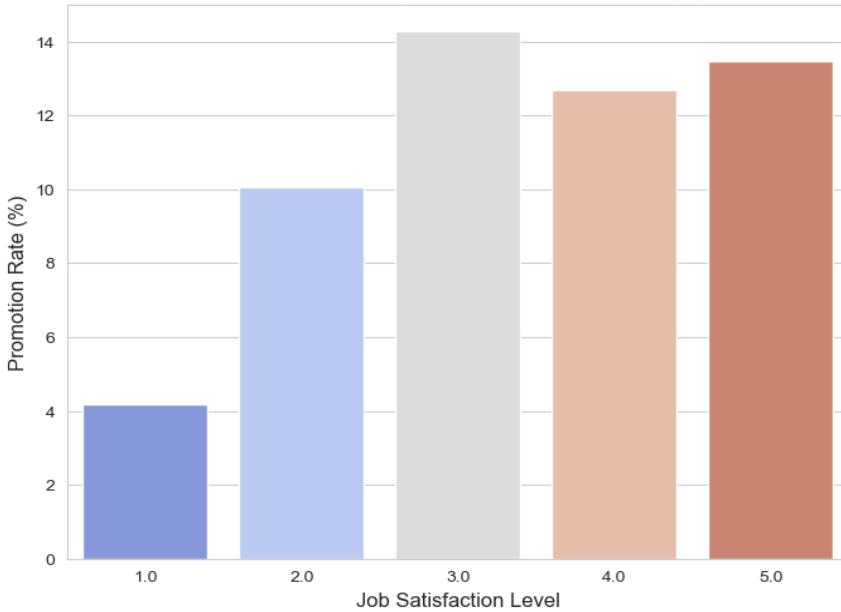
Count: count, dtype: float64.

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/4244127750.py:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

`sns.barplot(x=promotion_rate_by_satisfaction.index, y=promotion_rate_by_satisfaction.values, palette="coolwarm")`

Promotion Rate by Job Satisfaction Level (%)



```
[95... # 20. Does gender impact promotion opportunities?
# Promotion rate by gender
# Calculate promotion rate by gender using unique employees
promotion_by_gender = promoted_employees.groupby("EmployeeID")["Gender"].first().value_counts()

# Calculate total unique employees by gender for comparison
total_by_gender = merged_df.groupby("EmployeeID")["Gender"].first().value_counts()

# Calculate promotion rate by gender (% of promoted employees per gender)
promotion_rate_by_gender = (promotion_by_gender / total_by_gender * 100).fillna(0).sort_values(ascending=True)

print("Promotion Rate by Gender (%):")
print(promotion_rate_by_gender)

# Visualizing promotion rate by gender
plt.figure(figsize=(8, 6))
sns.barplot(x=promotion_rate_by_gender.values, y=promotion_rate_by_gender.index, palette="Set2")

plt.title("Promotion Rate by Gender (%)", fontsize=14)
plt.xlabel("Promotion Rate (%)", fontsize=12)
plt.ylabel("Gender", fontsize=12)

plt.show()
```

Promotion Rate by Gender (%):

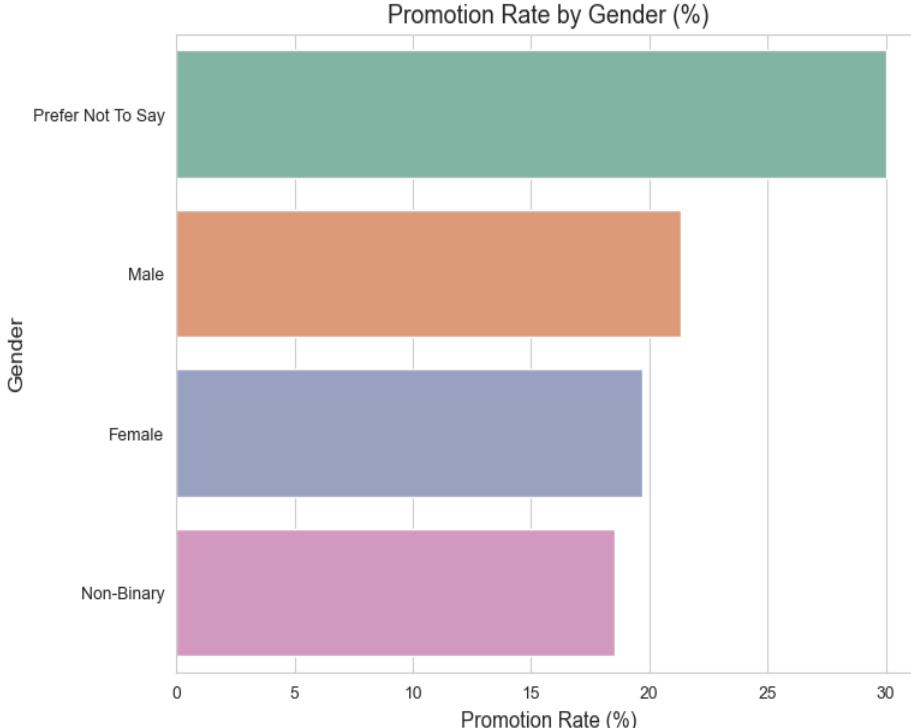
Gender	Promotion Rate (%)
Prefer Not To Say	30.000000
Male	21.351767
Female	19.703704
Non-Binary	18.548387

Name: count, dtype: float64

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/2755841693.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=promotion_rate_by_gender.values, y=promotion_rate_by_gender.index, palette="Set2")
```



```
[10... # Additional Questions
```

```
[11... # 21. Calculate the total salary of all employees
total_salary = merged_df.drop_duplicates(subset="EmployeeID")["Salary"].sum()
total_salary
```

```
[11... 166046052
```

```
[12... # 22. Calculate total salary distribution by department
salary_distribution = merged_df.drop_duplicates(subset="EmployeeID").groupby("Department")["Salary"].sum()
# Display the salary distribution
salary_distribution
```

	Department	Salary
0	Human Resources	7541025
1	Sales	53126454
2	Technology	105378573

```
[12... # 22. Calculate total salary distribution by department
# Visualizing total salary distribution by department
# Set plot style
sns.set_style("whitegrid")

# Create a figure with subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Total Number of Employees - Bar Chart
axes[0].bar(["Total Employees"], [total_employees], color="royalblue")
axes[0].set_title("Total Number of Employees", fontsize=14)
axes[0].set_ylabel("Count")

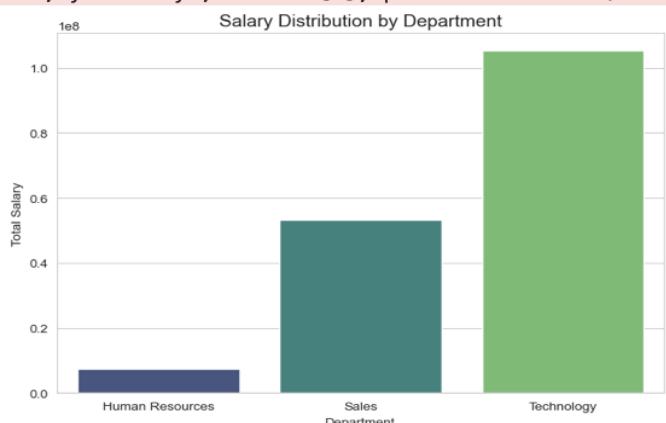
# Salary Distribution by Department - Bar Chart
sns.barplot(data=salary_distribution, x="Department", y="Salary", ax=axes[1], palette="viridis")
axes[1].set_title("Salary Distribution by Department", fontsize=14)
axes[1].set_ylabel("Total Salary")

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

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/2400868859.py:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=salary_distribution, x="Department", y="Salary", ax=axes[1], palette="viridis")
```



```
[16... # 23. Calculate the number of employees hired in each year
# Convert HireDate to datetime format
merged_df["HireDate"] = pd.to_datetime(merged_df["HireDate"], errors="coerce")

# Remove duplicates to ensure each employee is counted once
unique_employees = merged_df.drop_duplicates(subset="EmployeeID")

# Extract year from HireDate and count employees per year
employee_count_by_year = unique_employees.groupby(unique_employees["HireDate"].dt.year)[["EmployeeID"]].count()

# Display the result
employee_count_by_year
```

	HireDate	EmployeeID
0	2012	151
1	2013	136
2	2014	136
3	2015	127
4	2016	114
5	2017	106
6	2018	136
7	2019	145
8	2020	127
9	2021	137
10	2022	155

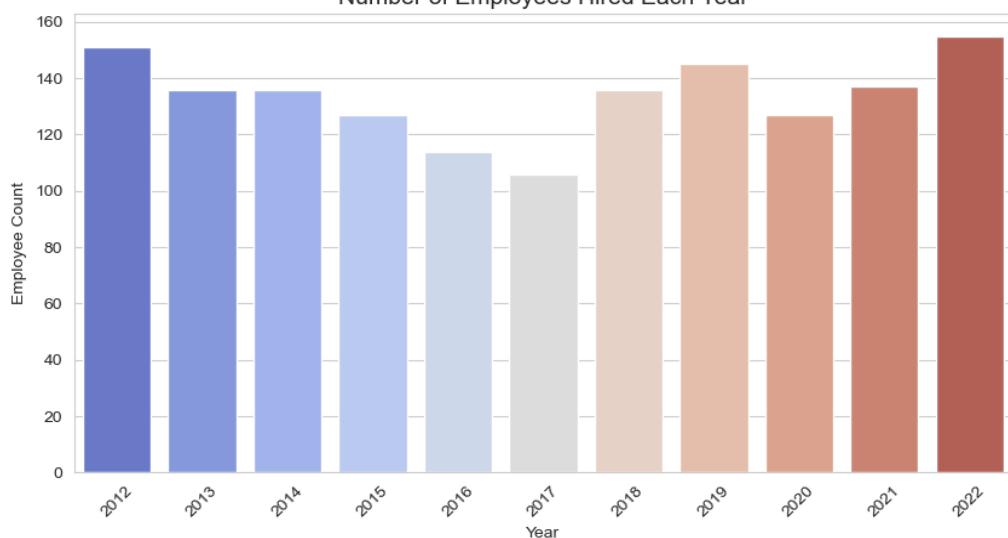
```
[16... # 23. Calculate the number of employees hired in each year
# Visualizing the number of employees in each year
# Plot the data
plt.figure(figsize=(10, 5))
sns.barplot(data=employee_count_by_year, x="Year", y="EmployeeCount", palette="coolwarm")

# Add labels and title
plt.title("Number of Employees Hired Each Year", fontsize=14)
plt.xlabel("Year")
plt.ylabel("Employee Count")
plt.xticks(rotation=45)

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

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/2530743306.py:5: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=employee_count_by_year, x="Year", y="EmployeeCount", palette="coolwarm")
```



```
[17... # 24. Calculate the Number of Employees based on BusinessTravel and Attrition
#Count employees by BusinessTravel and Attrition
business_travel_attrition = merged_df.groupby(["BusinessTravel", "Attrition"])["EmployeeID"].nunique()
business_travel_attrition.columns = ["BusinessTravel", "Attrition", "EmployeeCount"]

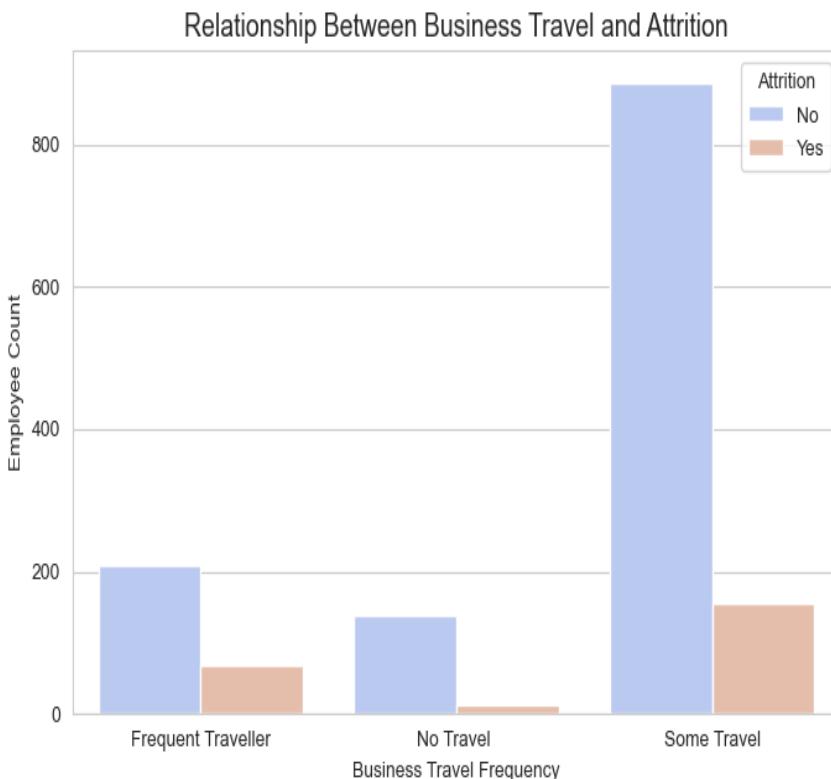
# Display the result
print(business_travel_attrition)
```

	BusinessTravel	Attrition	EmployeeCount
0	Frequent Traveller	No	208
1	Frequent Traveller	Yes	69
2	No Travel	No	138
3	No Travel	Yes	12
4	Some Travel	No	887
5	Some Travel	Yes	156

```
[17... # 24. Calculate the Number of Employees based on BusinessTravel and Attrition
# Plot the data
plt.figure(figsize=(8, 5))
sns.barplot(data=business_travel_attrition, x="BusinessTravel", y="EmployeeCount", hue="Attrition", pa

# Add labels and title
plt.title("Relationship Between Business Travel and Attrition", fontsize=14)
plt.xlabel("Business Travel Frequency")
plt.ylabel("Employee Count")

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



```
[18... # 25. Calculate the Number of Employees based on OverTime and Attrition
# Count employees by OverTime and Attrition
overtime_attrition = merged_df.groupby(["OverTime", "Attrition"])["EmployeeID"].nunique().reset_index()
overtime_attrition.columns = ["OverTime", "Attrition", "EmployeeCount"]

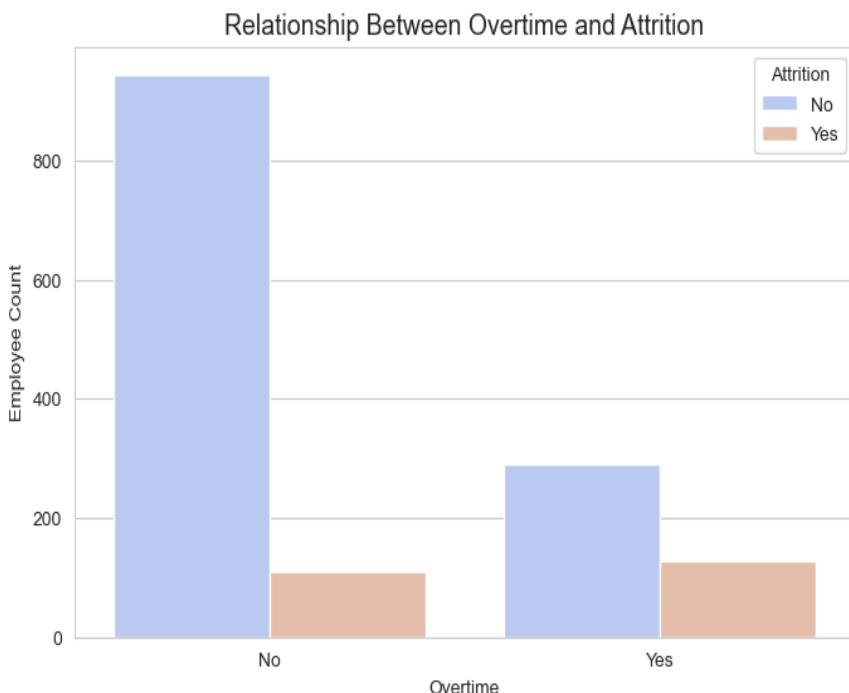
# Display the result
overtime_attrition
```

	OverTime	Attrition	EmployeeCount
0	No	No	944
1	No	Yes	110
2	Yes	No	289
3	Yes	Yes	127

```
[18... # 25. Calculate the Number of Employees based on OverTime and Attrition
# Plot the data
plt.figure(figsize=(8, 5))
sns.barplot(data=overtime_attrition, x="OverTime", y="EmployeeCount", hue="Attrition", palette="coolwarm"

# Add labels and title
plt.title("Relationship Between Overtime and Attrition", fontsize=14)
plt.xlabel("Overtime")
plt.ylabel("Employee Count")

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



```
[20...]
# 26.1 Define the age ranges of Employees
bins = [20, 30, 40, 50, 60, 70]
labels = ["20-30", "30-40", "40-50", "50-60", "60-70"]

# Remove duplicate employees by EmployeeID (if necessary) before calculating age range
merged_df_unique = merged_df.drop_duplicates(subset="EmployeeID")

# Assign each employee to an age range using .loc to avoid SettingWithCopyWarning
merged_df_unique.loc[:, "AgeRange"] = pd.cut(merged_df_unique["Age"], bins=bins, labels=labels, right=True)

# Count employees in each age range
age_range_count = merged_df_unique["AgeRange"].value_counts().reset_index()
age_range_count.columns = ["AgeRange", "EmployeeCount"]

# Display the result
print(age_range_count)

# 26.2 Find the minimum and maximum age of employees (no duplicates)
# Drop rows where Age is NaN (if any)
merged_df_unique = merged_df_unique.dropna(subset=['Age'])

min_age, max_age = merged_df_unique['Age'].min(), merged_df_unique['Age'].max()

# Display the results for minimum and maximum age
print(f"Age Range of Employees: {min_age} - {max_age}")

# 26.3 Find the age group with the most employees
# Find the age range with the most employees
max_age_range = age_range_count.loc[age_range_count["EmployeeCount"].idxmax()]

# Display the result for the age group with the most employees
print(f"The age range with the most employees is {max_age_range['AgeRange']} with {max_age_range['EmployeeCount']} employees")

AgeRange EmployeeCount
0 20-30 874
1 30-40 289
2 40-50 219
3 50-60 7
4 60-70 0
Age Range of Employees: 18 - 51
The age range with the most employees is 20-30 with 874 employees.
```

```
[20...]
# 26.1 Define the age ranges of Employees
# Plot the data
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 5)) # Size of the plot
sns.barplot(data=age_range_count, x="AgeRange", y="EmployeeCount", palette="Blues_r")

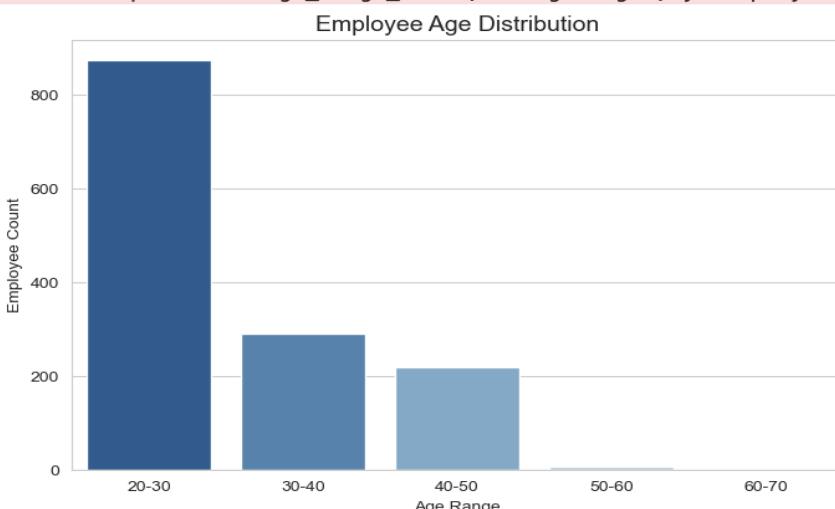
# Add labels and title
plt.title("Employee Age Distribution", fontsize=14)
plt.xlabel("Age Range")
plt.ylabel("Employee Count")

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

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/3134220879.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=age_range_count, x="AgeRange", y="EmployeeCount", palette="Blues_r")
```



```
[22... # 27. Calculate the correlation between ManagerRating and JobSatisfaction to assess employee performance
# Remove duplicates by EmployeeID to ensure each employee is counted only once
performance_manager_rating = merged_df[['EmployeeID', 'ManagerRating', 'JobSatisfaction']].drop_duplicates()

# Calculate the mean of JobSatisfaction based on ManagerRating
performance_by_manager_rating = performance_manager_rating.groupby('ManagerRating')['JobSatisfaction']

# Display the results
print(performance_by_manager_rating)
```

	ManagerRating	JobSatisfaction
0	2.0	3.439331
1	3.0	3.461538
2	4.0	3.402353
3	5.0	3.475000

```
[22... # 27. Calculate the correlation between ManagerRating and JobSatisfaction to assess employee performance
# Plot the data
# Remove duplicates by EmployeeID to ensure each employee is counted only once
performance_manager_rating = merged_df[['EmployeeID', 'ManagerRating', 'JobSatisfaction']].drop_duplicates()

# Create a barplot to visualize the average Job Satisfaction based on Manager Rating
plt.figure(figsize=(8, 5))
sns.barplot(x='ManagerRating', y='JobSatisfaction', data=performance_manager_rating, palette='viridis')

# Add labels and title
plt.title('Job Satisfaction based on Manager Rating', fontsize=14)
plt.xlabel('Manager Rating')
plt.ylabel('Average Job Satisfaction')

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

/var/folders/03/94ymsk8j2tb7j29w9l6vkzzw0000gn/T/ipykernel_10217/612261823.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x='ManagerRating', y='JobSatisfaction', data=performance_manager_rating, palette='viridis')
```



```
[27... # 28. Count employees hired per year and Count employees who left (attrition) by each department
import pandas as pd

# Convert Hire Date to datetime
merged_df["HireDate"] = pd.to_datetime(merged_df["HireDate"], errors="coerce")

# Extract Hire Year
merged_df["HireYear"] = merged_df["HireDate"].dt.year

# Identify latest year in dataset
latest_year = merged_df["HireYear"].max()

# Count employees hired per year by department
hire_counts = merged_df.groupby(["HireYear", "Department"])["EmployeeID"].nunique().reset_index()
hire_counts.columns = ["Year", "Department", "EmployeesHired"]

# Count employees who left (attrition) per department
attrition_counts = merged_df[merged_df["Attrition"] == "Yes"].groupby(["HireYear", "Department"])["EmployeeID"].nunique().reset_index()
attrition_counts.columns = ["Year", "Department", "EmployeesLeft"]

# Get all unique years and departments
all_years = pd.DataFrame({"Year": range(merged_df["HireYear"].min(), latest_year + 1)})
departments = merged_df["Department"].unique()
dept_df = pd.DataFrame({"Department": departments})

# Create a full Year-Department grid
year_dept_grid = all_years.merge(dept_df, how="cross")

# Merge hire and attrition data to ensure all Year-Department combinations exist
workforce_trend = year_dept_grid.merge(hire_counts, on=["Year", "Department"], how="left").fillna(0)
workforce_trend = workforce_trend.merge(attrition_counts, on=["Year", "Department"], how="left").fillna(0)

# Convert employee counts to integers
workforce_trend["EmployeesHired"] = workforce_trend["EmployeesHired"].astype(int)
workforce_trend["EmployeesLeft"] = workforce_trend["EmployeesLeft"].astype(int)

# Display the results as numbers
print("Employee Hires and Attrition by Year and Department:")
print(workforce_trend[["Year", "Department", "EmployeesHired", "EmployeesLeft"]])
```

Employee Hires and Attrition by Year and Department:

	Year	Department	EmployeesHired	EmployeesLeft
0	2012	Sales	49	9
1	2012	Human Resources	12	3
2	2012	Technology	90	12
3	2013	Sales	39	12
4	2013	Human Resources	4	0
5	2013	Technology	93	11
6	2014	Sales	49	8
7	2014	Human Resources	4	2
8	2014	Technology	83	13
9	2015	Sales	34	7
10	2015	Human Resources	5	1
11	2015	Technology	88	7
12	2016	Sales	35	8
13	2016	Human Resources	2	0
14	2016	Technology	77	16
15	2017	Sales	23	3
16	2017	Human Resources	7	1
17	2017	Technology	76	7
18	2018	Sales	43	11
19	2018	Human Resources	5	1
20	2018	Technology	88	10
21	2019	Sales	41	5
22	2019	Human Resources	3	0
23	2019	Technology	101	16
24	2020	Sales	41	13
25	2020	Human Resources	6	1
26	2020	Technology	80	14
27	2021	Sales	44	8
28	2021	Human Resources	6	1
29	2021	Technology	87	12
30	2022	Sales	48	8
31	2022	Human Resources	9	2
32	2022	Technology	98	15

```
[28... # 28. Count employees hired per year and Count employees who left (attrition) by each department
# Plot the data
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Convert Hire Date to datetime
merged_df["HireDate"] = pd.to_datetime(merged_df["HireDate"], errors="coerce")

# Extract Hire Year
merged_df["HireYear"] = merged_df["HireDate"].dt.year

# Identify latest year in dataset
latest_year = merged_df["HireYear"].max()

# Count employees hired per year by department
hire_counts = merged_df.groupby(["HireYear", "Department"])["EmployeeID"].nunique().reset_index()
hire_counts.columns = ["Year", "Department", "EmployeesHired"]

# Count employees who left (attrition) per department
attrition_counts = merged_df[merged_df["Attrition"] == "Yes"].groupby(["HireYear", "Department"])["EmployeeID"].nunique().reset_index()
attrition_counts.columns = ["Year", "Department", "EmployeesLeft"]

# Get all unique years and departments
all_years = pd.DataFrame({"Year": range(merged_df["HireYear"].min(), latest_year + 1)})
departments = merged_df["Department"].unique()
dept_df = pd.DataFrame({"Department": departments})

# Create a full Year-Department grid
year_dept_grid = all_years.merge(dept_df, how="cross")

# Merge hire and attrition data to ensure all Year-Department combinations exist
workforce_trend = year_dept_grid.merge(hire_counts, on=["Year", "Department"], how="left").fillna(0)
workforce_trend = workforce_trend.merge(attrition_counts, on=["Year", "Department"], how="left").fillna(0)

# Convert employee counts to integers
workforce_trend["EmployeesHired"] = workforce_trend["EmployeesHired"].astype(int)
workforce_trend["EmployeesLeft"] = workforce_trend["EmployeesLeft"].astype(int)

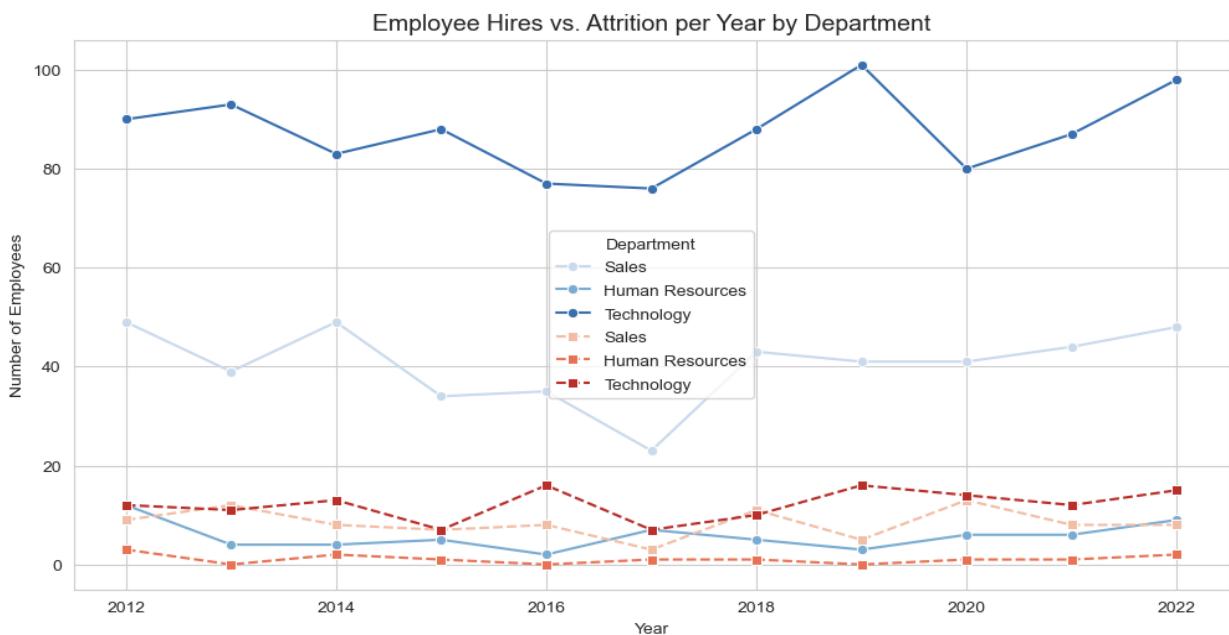
# Plot trends
plt.figure(figsize=(12, 6))

# Plot hires
sns.lineplot(data=workforce_trend, x="Year", y="EmployeesHired", hue="Department", marker="o", linestyle="solid")

# Plot attrition
sns.lineplot(data=workforce_trend, x="Year", y="EmployeesLeft", hue="Department", marker="s", linestyle="dashed")

# Labels and title
plt.title("Employee Hires vs. Attrition per Year by Department", fontsize=14)
plt.xlabel("Year")
plt.ylabel("Number of Employees")
plt.legend(title="Department")

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



[34...]

```
# 29. What is the most common reason for employee turnover?  
# 29. What is the most common reason for employee turnover?  
# Filter only employees who left (Attrition = 'Yes'), ensuring each employee is counted once  
attrition_data = merged_df[merged_df["Attrition"] == "Yes"].drop_duplicates(subset=["EmployeeID"])  
  
# Define the possible reasons for attrition (adjust based on your dataset)  
attrition_factors = ["JobRole", "BusinessTravel", "OverTime", "JobSatisfaction", "ManagerRating", "WorkLifeBalance"]  
  
# Create a dictionary to store the count of each factor for employees who left  
attrition_counts = {factor: attrition_data[factor].value_counts() for factor in attrition_factors}  
  
# Identify the most common reason by finding the value with the highest count  
most_common_reasons = {factor: attrition_counts[factor].idxmax() for factor in attrition_factors}  
  
# Display the most common reasons for attrition  
print("Most Common Reasons for Employee Turnover:")  
for factor, reason in most_common_reasons.items():  
    print(f"{factor}: {reason} (Count: {attrition_counts[factor].max()})")  
  
  
# Filter only employees who left (Attrition = 'Yes')  
#attrition_data = merged_df[merged_df["Attrition"] == "Yes"]  
  
# Define the possible reasons for attrition (adjust based on your dataset)  
#attrition_factors = ["JobRole", "BusinessTravel", "OverTime", "JobSatisfaction", "ManagerRating", "WorkLifeBalance"]  
  
# Create a dictionary to store the count of each factor for employees who left  
# Count each factor without dropping duplicates  
#attrition_counts = {factor: attrition_data[factor].value_counts() for factor in attrition_factors}  
  
# Identify the most common reason by finding the value with the highest count  
#most_common_reasons = {factor: attrition_counts[factor].idxmax() for factor in attrition_factors}  
  
# Display the most common reasons for attrition  
#print("Most Common Reasons for Employee Turnover:")  
#for factor, reason in most_common_reasons.items():  
#    print(f"{factor}: {reason} (Count: {attrition_counts[factor].max()})")
```

Most Common Reasons for Employee Turnover:

JobRole: Data Scientist (Count: 62)
BusinessTravel: Some Travel (Count: 156)
OverTime: Yes (Count: 127)
JobSatisfaction: 4.0 (Count: 64)
ManagerRating: 4.0 (Count: 82)
WorkLifeBalance: 4.0 (Count: 67)

[28..

```
# 29.2 What is the most common reason for employee turnover?
# Plot the data
import matplotlib.pyplot as plt
import seaborn as sns

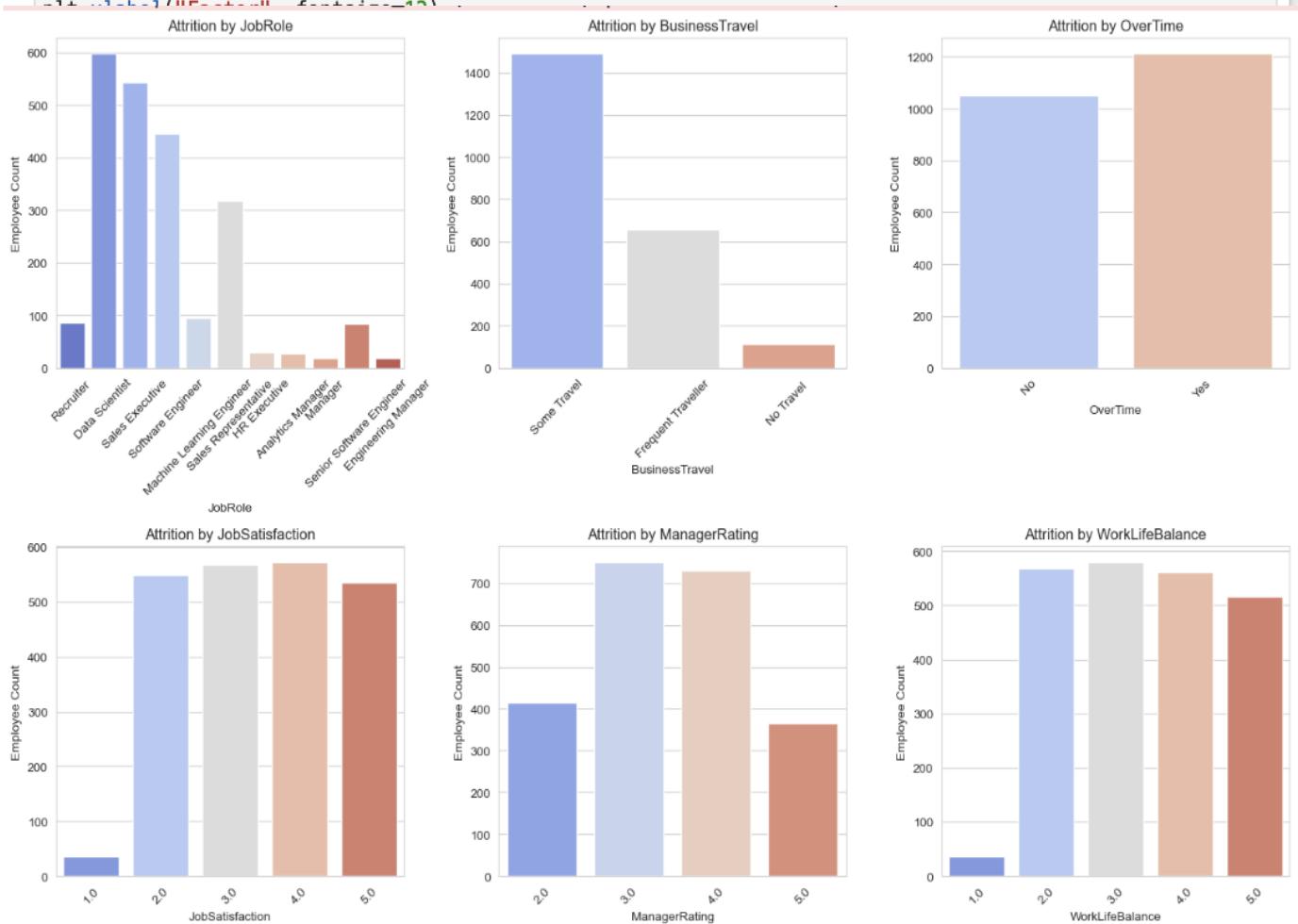
# Filter only employees who left (Attrition = 'Yes')
attrition_data = merged_df[merged_df["Attrition"] == "Yes"]

# Define the possible reasons for attrition (adjust based on your dataset)
attrition_factors = ["JobRole", "BusinessTravel", "OverTime", "JobSatisfaction", "ManagerRating", "WorkLifeBalance"]

# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
axes = axes.flatten()

# Plot each factor
for i, factor in enumerate(attrition_factors):
    sns.countplot(data=attrition_data, x=factor, palette="coolwarm", ax=axes[i])
    axes[i].set_title(f"Attrition by {factor}")
    axes[i].set_xlabel(factor)
    axes[i].set_ylabel("Employee Count")
    axes[i].tick_params(axis="x", rotation=45)

# Adjust layout to avoid overlapping
plt.tight_layout()
plt.show()
```



[]:

[17... ##### Ending of Week 2: Analysis Questions Phase

```
[21... ##### Week 3: Forecasting Questions Phase
```

```
[42... #####
# 1. Predict attrition for the next 3 years with visualization
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

# Load and preprocess data
df = pd.read_csv("/Users/rahmasaadawy/Downloads/Final_Project/cleaned_hr_dataset.csv")

df['attrition'] = df['attrition'].map({'Yes': 1, 'No': 0})
df['hiredate'] = pd.to_datetime(df['hiredate'])
df['hireyear'] = df['hiredate'].dt.year
df['yearsexperience'] = 2022 - df['hireyear']
df['overtime'] = df['overtime'].map({'Yes': 1, 'No': 0})

# Define features and target
features = ['age', 'salary', 'overtime', 'yearsatcompany', 'yearsexperience']
X = df[features]
y = df['attrition']

# Train model
model = RandomForestClassifier(random_state=42)
model.fit(X, y)

# Select only current employees (not attrited)
current_employees = df[df['attrition'] == 0].copy()
predictions = []

# Overtime rate adjustments
overtime_rates = {
    2023: 0.2,
    2024: 0.35,
    2025: 0.75 # Boosted for stronger attrition in 2025
}

# Predict attrition across years
for year in [2023, 2024, 2025]:
    yearly_employees = current_employees.copy()
    yearly_employees['yearsatcompany'] += (year - 2022)
    yearly_employees['yearsexperience'] = year - yearly_employees['hireyear']

    rate = overtime_rates[year]
    overtime_boost = np.random.binomial(1, rate, size=len(yearly_employees))
    yearly_employees['overtime'] = np.where(
        yearly_employees['overtime'] == 1,
        1,
        overtime_boost
    )

    if year == 2025:
        yearly_employees['yearsatcompany'] += 2 # Extra stress for 2025

    preds = model.predict(yearly_employees[features])
    predicted_attrition = int(np.sum(preds))
    predictions.append({'Year': year, 'Predicted_Attrition': predicted_attrition})
```

```

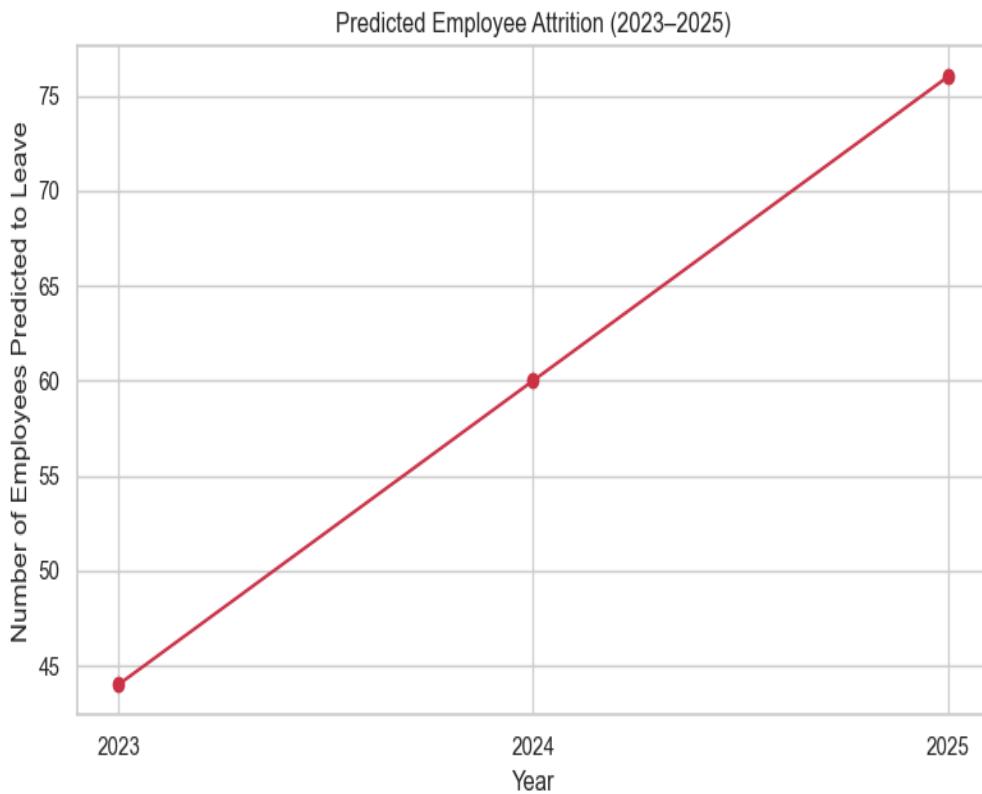
# Create predictions DataFrame
pred_df = pd.DataFrame(predictions)
print(pred_df)
pred_df.to_csv("/Users/rahmasaadawy/Downloads/Attrition_Prediction_Limited.csv", index=False)

# Visualization
plt.figure(figsize=(8, 5))
plt.plot(pred_df['Year'], pred_df['Predicted_Attrition'], marker='o', linestyle='-', color='crimson')
plt.title('Predicted Employee Attrition (2023-2025)')
plt.xlabel('Year')
plt.ylabel('Number of Employees Predicted to Leave')
plt.grid(True)
plt.xticks(pred_df['Year'])
plt.tight_layout()

# Save figure
plt.savefig("/Users/rahmasaadawy/Downloads/Attrition_Prediction_Chart.png")
plt.show()

```

	Year	Predicted_Attrition
0	2023	44
1	2024	60
2	2025	76



[42...]

```
#####
# 1. Historical Attrition for comparison
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Convert HireDate to datetime
merged_df['HireDate'] = pd.to_datetime(merged_df['HireDate'], errors='coerce')

# Step 2: Remove duplicate entries based on EmployeeID to ensure unique employees
unique_df = merged_df.drop_duplicates(subset=['EmployeeID'])
print(f"Number of unique employees: {len(unique_df)}")

# Step 3: Extract year from HireDate
unique_df['HireYear'] = unique_df['HireDate'].dt.year

# Step 4: Filter data for years 2012 to 2022
filtered_df = unique_df[(unique_df['HireYear'] >= 2012) & (unique_df['HireYear'] <= 2022)]

# Step 5: Count number of employees with Attrition = 1
attrition_by_year = filtered_df[filtered_df['Attrition'] == 1].groupby('HireYear').size().reset_index()

# Step 6: Ensure all years from 2012 to 2022 are included
all_years = pd.DataFrame({'HireYear': range(2012, 2023)})
attrition_by_year = all_years.merge(attrition_by_year, on='HireYear', how='left').fillna({'AttritionCount': 0})
attrition_by_year['AttritionCount'] = attrition_by_year['AttritionCount'].astype(int)

# Step 7: Save the results to a CSV file (only HireYear and AttritionCount)
attrition_by_year[['HireYear', 'AttritionCount']].to_csv('/Users/rahmasaadawy/Downloads/Attrition_by_Year.csv')

# Step 8: Print the results
print("\nAttrition Count by Year (2012-2022):")
print(attrition_by_year[['HireYear', 'AttritionCount']])

# Step 9: Visualization - Attrition Count
plt.figure(figsize=(10, 6))
plt.plot(attrition_by_year['HireYear'], attrition_by_year['AttritionCount'], marker='o', linestyle='-' )
for x, y in zip(attrition_by_year['HireYear'], attrition_by_year['AttritionCount']):
    plt.text(x, y + 0.5, f'{y}', ha='center', va='bottom', fontsize=10)
plt.title('Employee Attrition by Year (2012-2022)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Number of Employees Who Left', fontsize=12)
plt.grid(True)
plt.xticks(attrition_by_year['HireYear'])
plt.legend()
plt.tight_layout()

# Step 10: Save the plot
plt.savefig('/Users/rahmasaadawy/Downloads/Attrition_by_Year_Chart_2012_2022_Final.png')
plt.show()
```

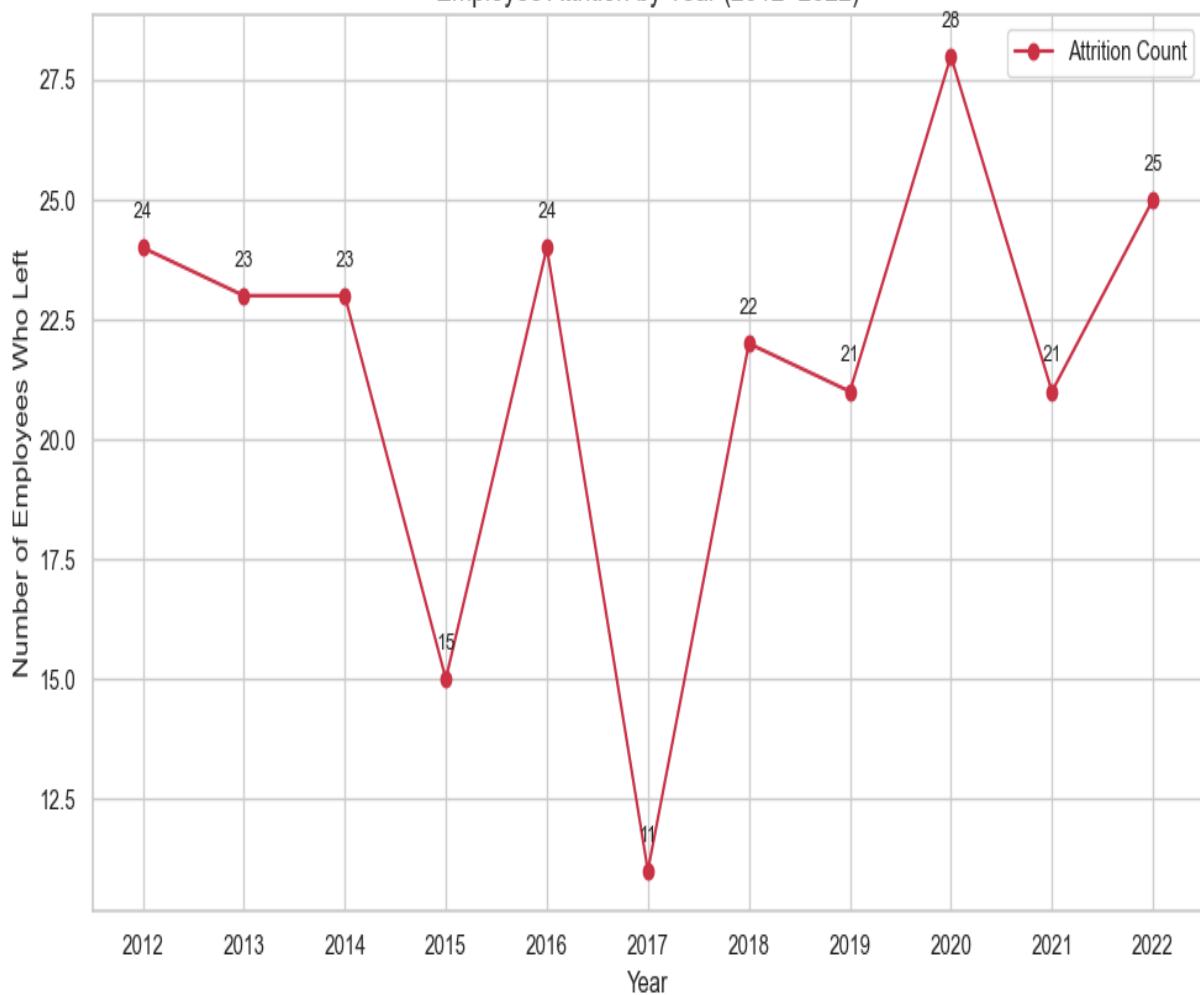
Number of unique employees: 1470

Attrition Count by Year (2012-2022):

HireYear AttritionCount

0	2012	24
1	2013	23
2	2014	23
3	2015	15
4	2016	24
5	2017	11
6	2018	22
7	2019	21
8	2020	28
9	2021	21
10	2022	25

Employee Attrition by Year (2012–2022)



[42...]

```
#####
# 2. Salary Growth Prediction
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor

# Load and preprocess data
df = pd.read_csv("/Users/rahmasaadawy/Downloads/Final_Project/cleaned_hr_dataset.csv")

df['hiredate'] = pd.to_datetime(df['hiredate'])
df['hireyear'] = df['hiredate'].dt.year
df['yearsexperience'] = 2022 - df['hireyear']
df['overtime'] = df['overtime'].map({'Yes': 1, 'No': 0})

# Define features and target
features = ['age', 'overtime', 'yearsatcompany', 'yearsexperience']
X = df[features]
y = df['salary']

# Train regression model
model = RandomForestRegressor(random_state=42)
model.fit(X, y)

# Predict salary growth for current employees for 2023–2025
current_employees = df.copy()
salary_predictions = []

for year in [2023, 2024, 2025]:
    yearly_employees = current_employees.copy()

    # Increase experience and tenure each year
    yearly_employees['yearsatcompany'] += (year - 2022)
    yearly_employees['yearsexperience'] = year - yearly_employees['hireyear']

    # Predict salary for the year
    predicted_salaries = model.predict(yearly_employees[features])
    average_salary = np.mean(predicted_salaries)
    salary_predictions.append({'Year': year, 'Predicted_Avg_Salary': round(average_salary, 2)})

# Save to CSV
salary_df = pd.DataFrame(salary_predictions)
print(salary_df)
salary_df.to_csv("/Users/rahmasaadawy/Downloads/Salary_Prediction.csv", index=False)

# Plot the data
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Load the salary prediction data
salary_df = pd.read_csv("/Users/rahmasaadawy/Downloads/Salary_Prediction.csv")
salary_df['Year'] = salary_df['Year'].astype(int)

# Create the line plot (single line plot, no department breakdown)
plt.figure(figsize=(8, 5)) # Figure size adjusted
ax = sns.lineplot(x='Year', y='Predicted_Avg_Salary', data=salary_df, marker='o', color='blue')
plt.xticks(salary_df['Year'])
```

```

# Add labels on the points
for i, row in salary_df.iterrows():
    ax.annotate(f'${row["Predicted_Avg_Salary"]:.2f}',
               (row['Year'], row['Predicted_Avg_Salary']),
               ha='center', va='bottom', fontsize=10)

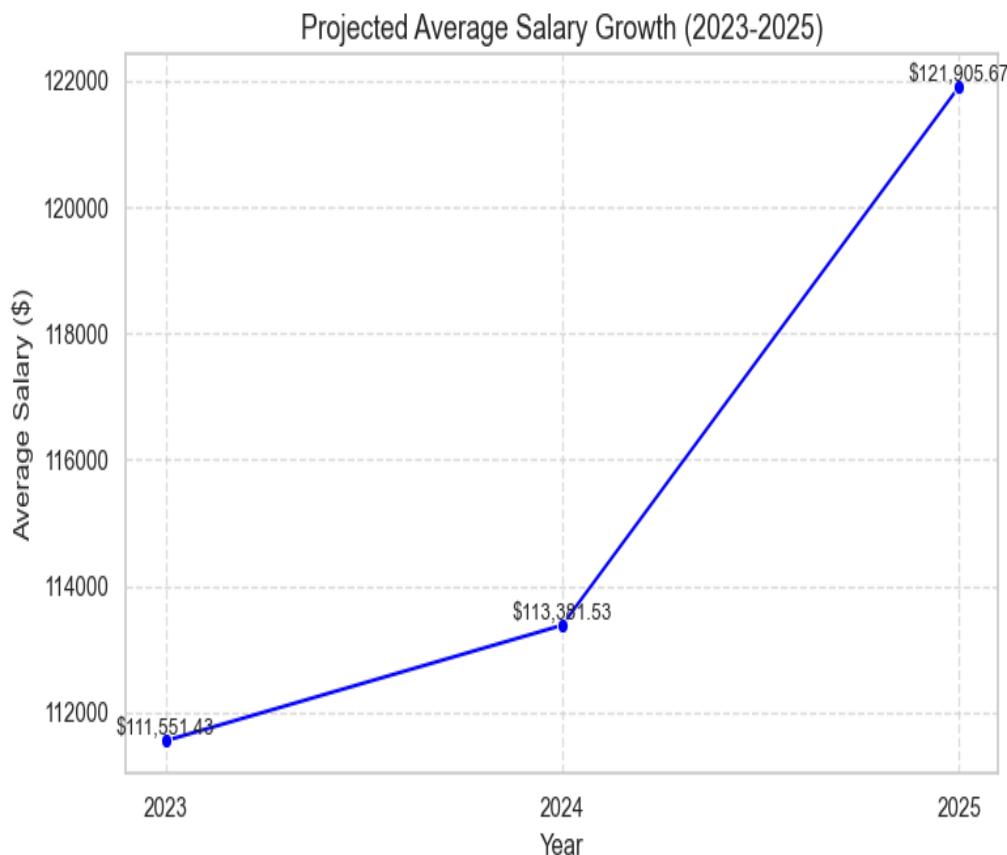
# Title and axis labels
plt.title('Projected Average Salary Growth (2023–2025)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Average Salary ($)', fontsize=12)

# Grid and plot appearance settings
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()

# Save the plot
plt.savefig('/Users/rahmasaadawy/Downloads/Salary_Growth_Projection_2023_2025.png', dpi=80) # Reduced
plt.show()

```

Year	Predicted_Avg_Salary
0 2023	111551.43
1 2024	113381.53
2 2025	121905.67



[42...]

```
#####
# 2. Historical Avg. Salary for comparison
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Convert HireDate to datetime
merged_df['HireDate'] = pd.to_datetime(merged_df['HireDate'], errors='coerce')

# Step 2: Remove duplicate entries based on EmployeeID to ensure unique employees
unique_df = merged_df.drop_duplicates(subset=['EmployeeID'])
print(f"Number of unique employees: {len(unique_df)}")

# Step 3: Extract year from HireDate
unique_df['HireYear'] = unique_df['HireDate'].dt.year

# Step 4: Filter data for years 2012 to 2022
filtered_df = unique_df[(unique_df['HireYear'] >= 2012) & (unique_df['HireYear'] <= 2022)]

# Step 5: Calculate average salary for each year
average_salary_by_year = filtered_df.groupby('HireYear')['Salary'].mean().reset_index(name='AverageSalary')

# Step 6: Ensure all years from 2012 to 2022 are included
all_years = pd.DataFrame({'HireYear': range(2012, 2023)})
average_salary_by_year = all_years.merge(average_salary_by_year, on='HireYear', how='left').fillna({'AverageSalary': 0})
average_salary_by_year['AverageSalary'] = average_salary_by_year['AverageSalary'].astype(float)

# Step 7: Save the results to a CSV file (only HireYear and AverageSalary)
average_salary_by_year[['HireYear', 'AverageSalary']].to_csv('/Users/rahmasaadawy/Downloads/Average_Salary_by_Year.csv')

# Step 8: Print the results
print("\nAverage Salary by Year (2012–2022):")
print(average_salary_by_year[['HireYear', 'AverageSalary']])

# Step 9: Visualization – Average Salary
plt.figure(figsize=(8, 5))
ax = sns.lineplot(x='HireYear', y='AverageSalary', data=average_salary_by_year, marker='o', color='blue')

# Add labels on the points with dollar format
for i, row in average_salary_by_year.iterrows():
    ax.annotate(f'${row["AverageSalary"]:.2f}', (row['HireYear'], row['AverageSalary']),
                ha='center', va='bottom', fontsize=10)

# Title and axis labels
plt.title('Average Salary by Year (2012–2022)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Average Salary ($)', fontsize=12)

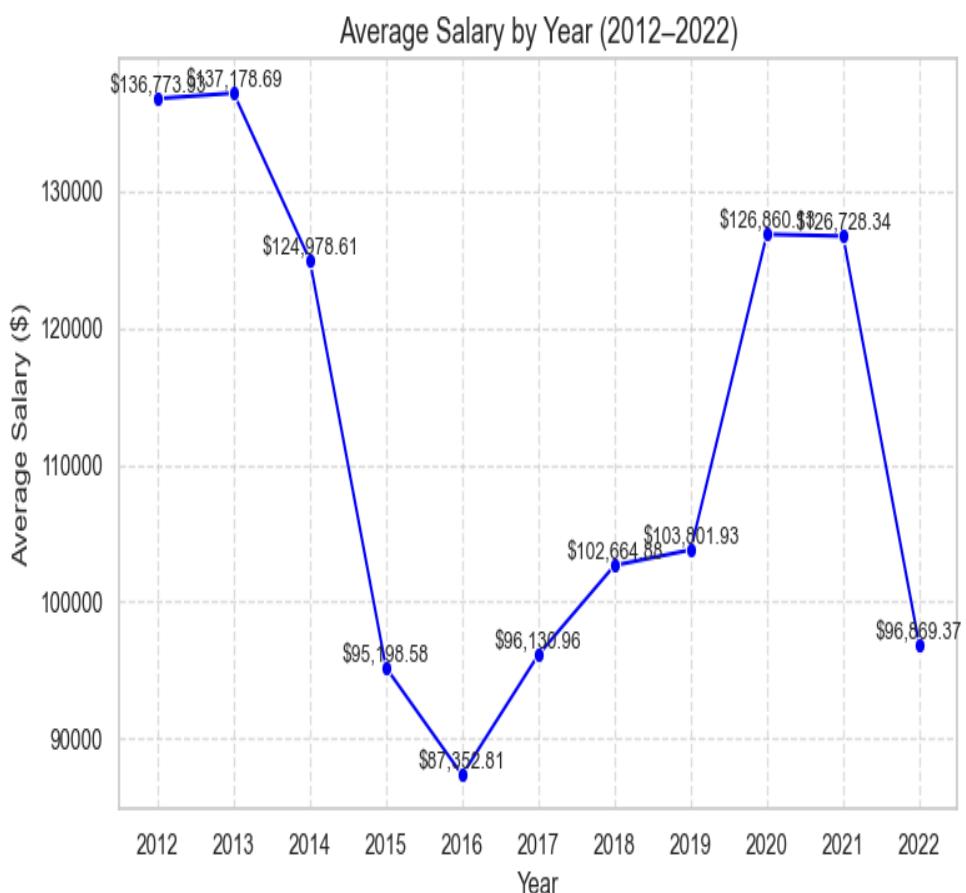
# Grid and plot appearance settings
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(average_salary_by_year['HireYear'])
plt.tight_layout()

# Step 10: Save the plot
plt.savefig('/Users/rahmasaadawy/Downloads/Average_Salary_by_Year_Chart_2012_2022.png', dpi=80)
plt.show()
```

Number of unique employees: 1470

Average Salary by Year (2012–2022):

HireYear	AverageSalary
0	2012 136773.927152
1	2013 137178.691176
2	2014 124978.610294
3	2015 95198.582677
4	2016 87352.807018
5	2017 96130.962264
6	2018 102664.882353
7	2019 103801.931034
8	2020 126860.125984
9	2021 126728.335766
10	2022 96869.367742



[40...

Ending of Week 3: Forecasting Questions Phase