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

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Attrition data.csv to Attrition data.csv

```
df = pd.read_csv('/content/Attrition data.csv')
```

Data Exploration and Cleaning

```
df.head()
```

	EmployeeID	Age	Attrition	BusinessTravel	Department	DistanceFromHome	Education	EducationField	EmployeeCount	Gender	...
0	1	51	No	Travel_Rarely	Sales	6	2	Life Sciences	1	Female	...
1	2	31	Yes	Travel_Frequently	Research & Development	10	1	Life Sciences	1	Female	...
2	3	32	No	Travel_Frequently	Research & Development	17	4	Other	1	Male	...
3	4	38	No	Non-Travel	Research & Development	2	5	Life Sciences	1	Male	...
4	5	32	No	Travel_Rarely	Research & Development	10	1	Medical	1	Male	...

5 rows × 29 columns

```
df.tail()
```

	EmployeeID	Age	Attrition	BusinessTravel	Department	DistanceFromHome	Education	EducationField	EmployeeCount	Gender	..
4405	4406	42	No	Travel_Rarely	Research & Development	5	4	Medical	1	Female	
4406	4407	29	No	Travel_Rarely	Research & Development	2	4	Medical	1	Male	
4407	4408	25	No	Travel_Rarely	Research & Development	25	2	Life Sciences	1	Male	
4408	4409	42	No	Travel_Rarely	Sales	18	2	Medical	1	Male	
4409	4410	40	No	Travel_Rarely	Research & Development	28	3	Medical	1	Male	

5 rows × 29 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4410 entries, 0 to 4409
Data columns (total 29 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   EmployeeID               4410 non-null   int64  
 1   Age                      4410 non-null   int64  
 2   Attrition                4410 non-null   object 
 3   BusinessTravel           4410 non-null   object 
 4   Department               4410 non-null   object 
 5   DistanceFromHome         4410 non-null   int64  
 6   Education                4410 non-null   int64  
 7   EducationField           4410 non-null   object 
 8   EmployeeCount            4410 non-null   int64  
 9   Gender                   4410 non-null   object 
 10  JobLevel                 4410 non-null   int64  
 11  JobRole                  4410 non-null   object 
 12  MaritalStatus            4410 non-null   object 
 13  MonthlyIncome            4410 non-null   int64  
 14  NumCompaniesWorked       4391 non-null   float64
 15  Over18                   4410 non-null   object 
 16  PercentSalaryHike        4410 non-null   int64  
 17  StandardHours            4410 non-null   int64  
```

```
 18  StockOptionLevel         4410 non-null   int64
 19  TotalWorkingYears        4401 non-null   float64
 20  TrainingTimesLastYear    4410 non-null   int64
 21  YearsAtCompany           4410 non-null   int64
 22  YearsSinceLastPromotion  4410 non-null   int64
 23  YearsWithCurrManager     4410 non-null   int64
 24  EnvironmentSatisfaction  4385 non-null   float64
 25  JobSatisfaction          4390 non-null   float64
 26  WorkLifeBalance          4372 non-null   float64
 27  JobInvolvement           4410 non-null   int64
 28  PerformanceRating        4410 non-null   int64
dtypes: float64(5), int64(16), object(8)
memory usage: 999.3+ KB
```

```
df.describe()
```

	EmployeeID	Age	DistanceFromHome	Education	EmployeeCount	JobLevel	MonthlyIncome	NumCompaniesWorked	Percent
count	4410.000000	4410.000000	4410.000000	4410.000000	4410.0	4410.000000	4410.000000	4391.000000	
mean	2205.500000	36.923810	9.192517	2.912925	1.0	2.063946	65029.312925	2.694830	
std	1273.201673	9.133301	8.105026	1.023933	0.0	1.106689	47068.888559	2.498887	
min	1.000000	18.000000	1.000000	1.000000	1.0	1.000000	10090.000000	0.000000	
25%	1103.250000	30.000000	2.000000	2.000000	1.0	1.000000	29110.000000	1.000000	
50%	2205.500000	36.000000	7.000000	3.000000	1.0	2.000000	49190.000000	2.000000	
75%	3307.750000	43.000000	14.000000	4.000000	1.0	3.000000	83800.000000	4.000000	
max	4410.000000	60.000000	29.000000	5.000000	1.0	5.000000	199990.000000	9.000000	

8 rows × 21 columns

```
df.columns
```

```
Index(['EmployeeID', 'Age', 'Attrition', 'BusinessTravel', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'Gender', 'JobLevel', 'JobRole', 'MaritalStatus', 'MonthlyIncome',
       'NumCompaniesWorked', 'Over18', 'PercentSalaryHike', 'StandardHours',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'YearsAtCompany', 'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'EnvironmentSatisfaction', 'JobSatisfaction', 'WorkLifeBalance',
       'JobInvolvement', 'PerformanceRating'],
      dtype='object')
```

```
df['Gender']
```

	Gender
0	Female
1	Female
2	Male
3	Male
4	Male
...	...
4405	Female
4406	Male
4407	Male
4408	Male
4409	Male

4410 rows × 1 columns

dtype: object

```
df['Education']
```

	Education
0	2
1	1
2	4
3	5
4	1
...	...
4405	4
4406	4
4407	2
4408	2
4409	3

4410 rows × 1 columns

dtype: int64

```
df['EducationField']
```

	EducationField
0	Life Sciences
1	Life Sciences
2	Other
3	Life Sciences
4	Medical
...	...
4405	Medical
4406	Medical
4407	Life Sciences
4408	Medical
4409	Medical

4410 rows × 1 columns

dtype: object

```
df['BusinessTravel']
```

	BusinessTravel
0	Travel_Rarely
1	Travel_Frequently
2	Travel_Frequently
3	Non-Travel
4	Travel_Rarely
...	...
4405	Travel_Rarely
4406	Travel_Rarely
4407	Travel_Rarely
4408	Travel_Rarely
4409	Travel_Rarely

4410 rows × 1 columns

dtype: object

```
df[['EmployeeCount', 'JobLevel']]
```

	EmployeeCount	JobLevel
0	1	1
1	1	1
2	1	4
3	1	3
4	1	1
...	...	...
4405	1	1
4406	1	1
4407	1	2
4408	1	1
4409	1	2

4410 rows × 2 columns

```
df[['EmployeeCount', 'Over18', 'StandardHours']]
```

	EmployeeCount	Over18	StandardHours
0	1	Y	8
1	1	Y	8
2	1	Y	8
3	1	Y	8
4	1	Y	8
...	...	...	...
4405	1	Y	8
4406	1	Y	8
4407	1	Y	8
4408	1	Y	8
4409	1	Y	8

4410 rows × 3 columns

```
df.drop(columns=['EmployeeCount', 'StandardHours','Over18'], inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4410 entries, 0 to 4409
Data columns (total 29 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   EmployeeID               4410 non-null   int64  
 1   Age                      4410 non-null   int64  
 2   Attrition                4410 non-null   object 
 3   BusinessTravel           4410 non-null   object 
 4   Department               4410 non-null   object 
 5   DistanceFromHome         4410 non-null   int64  
 6   Education                4410 non-null   int64  
 7   EducationField           4410 non-null   object 
 8   EmployeeCount            4410 non-null   int64  
 9   Gender                   4410 non-null   object 
 10  JobLevel                 4410 non-null   int64  
 11  JobRole                  4410 non-null   object 
 12  MaritalStatus            4410 non-null   object 
 13  MonthlyIncome            4410 non-null   int64  
 14  NumCompaniesWorked       4391 non-null   float64
 15  Over18                   4410 non-null   object 
 16  PercentSalaryHike        4410 non-null   int64  
 17  StandardHours            4410 non-null   int64  
 18  StockOptionLevel         4410 non-null   int64  
 19  TotalWorkingYears        4401 non-null   float64
 20  TrainingTimesLastYear    4410 non-null   int64  
 21  YearsAtCompany           4410 non-null   int64  
 22  YearsSinceLastPromotion  4410 non-null   int64  
 23  YearsWithCurrManager     4410 non-null   int64  
 24  EnvironmentSatisfaction  4385 non-null   float64
 25  JobSatisfaction          4390 non-null   float64
```

```
 26  WorkLifeBalance          4372 non-null   float64
 27  JobInvolvement           4410 non-null   int64
 28  PerformanceRating        4410 non-null   int64
dtypes: float64(5), int64(16), object(8)
memory usage: 999.3+ KB
```

```
missing_values = df.isnull().sum()
print("Missing Values\n", missing_values)
```

```
Missing Values
 EmployeeID                  0
Age                         0
Attrition                   0
BusinessTravel              0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
Gender                      0
JobLevel                    0
JobRole                     0
MaritalStatus               0
MonthlyIncome               0
NumCompaniesWorked         19
PercentSalaryHike           0
StockOptionLevel            0
TotalWorkingYears           9
TrainingTimesLastYear       0
YearsAtCompany              0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
EnvironmentSatisfaction    25
JobSatisfaction            20
WorkLifeBalance            38
JobInvolvement              0
PerformanceRating           0
dtype: int64
```

```
duplicate_values = df.duplicated().sum()
print('Number of duplicates', duplicate_values)
```

```
Number of duplicates 0
```

```
df[['EnvironmentSatisfaction', 'WorkLifeBalance', 'JobSatisfaction', 'NumCompaniesWorked','TotalWorkingYears']]
```

	EnvironmentSatisfaction	WorkLifeBalance	JobSatisfaction	NumCompaniesWorked	TotalWorkingYears
0	3.0	2.0	4.0	1.0	1.0
1	3.0	4.0	2.0	0.0	6.0
2	2.0	1.0	2.0	1.0	5.0
3	4.0	3.0	4.0	3.0	13.0
4	4.0	3.0	1.0	4.0	9.0
...	...	...	...	...	...
4405	4.0	3.0	1.0	3.0	10.0
4406	4.0	3.0	4.0	2.0	10.0
4407	1.0	3.0	3.0	0.0	5.0
4408	4.0	3.0	1.0	0.0	10.0
4409	1.0	NaN	3.0	0.0	NaN

4410 rows × 5 columns

```
df['NumCompaniesWorked'].fillna(df['NumCompaniesWorked'].median(), inplace=True)
df['TotalWorkingYears'].fillna(df['TotalWorkingYears'].median(), inplace=True)
```

```
<ipython-input-13-929152a4f54b>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting 

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['NumCompaniesWorked'].fillna(df['NumCompaniesWorked'].median(), inplace=True)
<ipython-input-13-929152a4f54b>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting 

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col
```

```
  df['TotalWorkingYears'].fillna(df['TotalWorkingYears'].median(), inplace=True)
```

```
df['EnvironmentSatisfaction'].fillna(df['EnvironmentSatisfaction'].mode()[0], inplace=True)
df['JobSatisfaction'].fillna(df['JobSatisfaction'].mode()[0], inplace=True)
df['WorkLifeBalance'].fillna(df['WorkLifeBalance'].mode()[0], inplace=True)
```

```
<ipython-input-14-d940df63d19b>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['EnvironmentSatisfaction'].fillna(df['EnvironmentSatisfaction'].mode()[0], inplace=True)
<ipython-input-14-d940df63d19b>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['JobSatisfaction'].fillna(df['JobSatisfaction'].mode()[0], inplace=True)
<ipython-input-14-d940df63d19b>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['WorkLifeBalance'].fillna(df['WorkLifeBalance'].mode()[0], inplace=True)
```

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

	0
EmployeeID	0
Age	0
Attrition	0
BusinessTravel	0
Department	0
DistanceFromHome	0
Education	0
EducationField	0
Gender	0
JobLevel	0
JobRole	0
MaritalStatus	0
MonthlyIncome	0
NumCompaniesWorked	0
PercentSalaryHike	0
StockOptionLevel	0
TotalWorkingYears	0
TrainingTimesLastYear	0
YearsAtCompany	0
YearsSinceLastPromotion	0
YearsWithCurrManager	0
EnvironmentSatisfaction	0
JobSatisfaction	0
WorkLifeBalance	0
JobInvolvement	0
PerformanceRating	0

dtype: int64

```
df.to_csv('Cleaned_Attrition_Data.csv', index=False)
```

Descriptive Stats

```
total_employees = df['EmployeeID'].nunique()
print(f"Total Employees: {total_employees}")
```

```
Total Employees: 4410
```

```
demographics = df.groupby('Gender').size()
print(f"Employee Demographics: {demographics}")
```

```
Employee Demographics: Gender
Female    1764
Male      2646
dtype: int64
```

```
total_attrition = df['Attrition'].value_counts()['Yes']
print(f"Total Attrition: {total_attrition}")
```

```
Total Attrition: 711
```

```
total_department = df['Department'].value_counts()
print(f"Total Employees per Department: {total_department}")
```

```
Total Employees per Department: Department
Research & Development    2883
Sales                     1338
Human Resources            189
Name: count, dtype: int64
```

```
total_job_role = df['JobRole'].value_counts()
print(f"Total employees per job role: {total_job_role}")
```

```
Total employees per job role: JobRole
Sales Executive              978
Research Scientist           876
Laboratory Technician        777
Manufacturing Director       435
Healthcare Representative    393
Manager                      306
Sales Representative         249
Research Director            240
Human Resources              156
Name: count, dtype: int64
```

```
total_job_level = df['JobLevel'].value_counts()
print(f"Total employees per job level: {total_job_level}")
```

```
Total employees per job level: JobLevel
1    1629
2    1602
3     654
4     318
5     207
Name: count, dtype: int64
```

KPIs

Attrition Rate The percentage of employees leaving the company over a specific period.

Formula: (Number of employees who left/ Total number of employees)*100

```
attrition_rate = (df['Attrition'].value_counts()['Yes'] / len(df)) * 100
print(f"Attrition Rate: {attrition_rate:.2f} %")
```

```
Attrition Rate: 16.12 %
```

```
total_retention = total_employees - total_attrition
print(f"Retention: {total_retention}")
```

```
Retention: 3699
```

```
retention_rate = (total_retention / len(df))*100
print(f"Retention Rate: {retention_rate:.2f} %")
```

```
Retention Rate: 83.88 %
```

Department Wise Attrition and Retention

```
att_dept = df[df['Attrition'] == 'Yes'] ['Department'].value_counts()
ret_dept = df[df['Attrition'] == 'No'] ['Department'].value_counts()

value = pd.DataFrame({
    'Attrition per department': att_dept,
    'Retention per department': ret_dept
    })
```

```
value
```

Department	Attrition per department	Retention per department
Research & Development	453	2430
Sales	201	1137
Human Resources	57	132

```
att_rate_dept = df[df['Attrition'] == 'Yes']['Department'].value_counts(normalize=True) * 100

ret_rate_dept = df[df['Attrition'] == 'No']['Department'].value_counts(normalize=True) * 100

value = pd.DataFrame({
    'Attrition Rate (%)': att_rate_dept,
    'Retention Rate (%)': ret_rate_dept
})
```

```
value
```

Department	Attrition Rate (%)	Retention Rate (%)
Research & Development	63.713080	65.693431
Sales	28.270042	30.738037
Human Resources	8.016878	3.568532

Average Tenure of Employees

```
avg_tenure = df['YearsAtCompany'].mean()
print(f"Average Tenure: {avg_tenure:.2f} years")
```

```
Average Tenure: 7.01 years
```

```
avg_tenure_by_att =df.groupby('Attrition')['YearsAtCompany'].mean()
print(f"Average Tenure by Attriton: {avg_tenure_by_att}")
```

```
Average Tenure by Attriton: Attrition
No     7.369019
Yes    5.130802
Name: YearsAtCompany, dtype: float64
```

```
avg_tenure_by_att =df[df['Attrition'] == 'Yes']['YearsAtCompany'].mean()
print(f"Average Tenure by Attrition: {avg_tenure_by_att}")
```

```
Average Tenure by Attrition: 5.1308016877637135
```

Avg Experience of Employees

```
avg_working_years = df['TotalWorkingYears'].mean()
print('Average Experience of Employees:' , avg_working_years)
```

```
Average Experience of Employees: 11.27732426303855
```

Job Satisfaction Level

```
avg_job_satisfaction = df['JobSatisfaction'].mean()
print(f"Average Job Satisfaction: {avg_job_satisfaction:.2f}")
```

```
Average Job Satisfaction: 2.73
```

Work Life Balance Score

```
avg_work_life_balance = df['WorkLifeBalance'].mean()
print(f"Average Work-Life Balance: {avg_work_life_balance:.2f}")
```

```
Average Work-Life Balance: 2.76
```

Avg Work Environment Rating

```
avg_env_rating = df['EnvironmentSatisfaction'].mean()
print(f"Average Rating for Work Environment: {avg_env_rating:.2f}")
```

```
Average Rating for Work Environment: 2.73
```

Average Salary

```
avg_salary = df['MonthlyIncome'].mean()

print('Average Salary:' , avg_salary)
```

```
Average Salary: 65029.31292517007
```

Monthly Income Distribution for Attrition

```
income_by_att = df.groupby('Attrition')['MonthlyIncome'].mean()
print(f"Avg monthly income: {income_by_att}")
```

```
Avg monthly income: Attrition
No     65672.595296
Yes    61682.616034
Name: MonthlyIncome, dtype: float64
```

Average Salary by Job level

```
avg_salary_by_job_level = df.groupby('JobLevel')['MonthlyIncome'].mean()
print('Avg salary by Job Level:', avg_salary_by_job_level)
```

```
Avg salary by Job Level: JobLevel
1    62677.421731
2    65506.479401
3    63545.321101
4    77940.754717
5    64698.405797
Name: MonthlyIncome, dtype: float64
```

```
df.columns
```

```
Index(['EmployeeID', 'Age', 'Attrition', 'BusinessTravel', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'Gender', 'JobLevel',
       'JobRole', 'MaritalStatus', 'MonthlyIncome', 'NumCompaniesWorked',
       'PercentSalaryHike', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'YearsAtCompany', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'EnvironmentSatisfaction', 'JobSatisfaction',
       'WorkLifeBalance', 'JobInvolvement', 'PerformanceRating'],
      dtype='object')
```

Avg Salary by Job Role

```
avg_salary_by_job_role = df.groupby('JobRole')['MonthlyIncome'].mean()
print('Avg salary by Job role:', avg_salary_by_job_role)
```

```
Avg salary by Job role: JobRole
Healthcare Representative    60983.740458
Human Resources              58528.076923
Laboratory Technician        66314.054054
```

```
Manager                      63395.882353
Manufacturing Director       69183.724138
Research Director            65473.125000
Research Scientist           64975.684932
Sales Executive              65186.687117
Sales Representative         65370.963855
Name: MonthlyIncome, dtype: float64
```

```
df[['JobLevel', 'JobRole']]
```

	JobLevel	JobRole
0	1	Healthcare Representative
1	1	Research Scientist
2	4	Sales Executive
3	3	Human Resources
4	1	Sales Executive
...	...	...
4405	1	Research Scientist
4406	1	Laboratory Technician
4407	2	Sales Executive
4408	1	Laboratory Technician
4409	2	Laboratory Technician

4410 rows × 2 columns

Avg Salary by Department

```
avg_salary_by_Department = df.groupby('Department')['MonthlyIncome'].mean()
avg_salary_by_Department
```

	MonthlyIncome
Department	
Human Resources	57904.444444
Research & Development	67187.960458
Sales	61384.484305

dtype: float64

Avg Salary Hike

```
avg_salary_hike = df['PercentSalaryHike'].mean()
print('Average Salary Hike:' , avg_salary_hike, '%')
```

```
Average Salary Hike: 15.209523809523809 %
```

Avg Salary Hike by Attrition

```
avg_sal_hike_by_att = df.groupby('Attrition')['PercentSalaryHike'].mean()
avg_sal_hike_by_att
```

	PercentSalaryHike
Attrition	
No	15.157340
Yes	15.481013

dtype: float64

Average distance from home to office

```
avg_distance = df['DistanceFromHome'].mean()
print('Average distance from home to office:' , avg_distance)
```

```
Average distance from home to office: 9.19251700680272
```

Promotion and Attrition - The average number of years since the last promotion for employees who left vs. those who stayed.

```
promotion_att = df.groupby('Attrition')['YearsSinceLastPromotion'].mean()
print(f"Avg number of years since last promotion for employees who left: {promotion_att}")
```

```
Avg number of years since last promotion for employees who left: Attrition
No     2.234388
Yes    1.945148
Name: YearsSinceLastPromotion, dtype: float64
```

Training and Development Impact - The average number of training sessions employees attended and its relation to attrition.

```
training_by_att = df.groupby('Attrition')['TrainingTimesLastYear'].mean()
print(f" Avg number of training sessions attented by ex employees: {training_by_att}")
```

```
 Avg number of training sessions attented by ex employees: Attrition
No     2.827251
Yes    2.654008
Name: TrainingTimesLastYear, dtype: float64
```

Avg Performance Rating

```
avg_performance_rating = df['PerformanceRating'].mean()

print('Average Performance Rating of Employees:' , avg_performance_rating)
```

```
Average Performance Rating of Employees: 3.1537414965986397
```

Performance Rating and Attrition- The performance score comparison between employees who stayed and those who left.

```
performance_att = df.groupby('Attrition')['PerformanceRating'].mean()
print(performance_att)
```

```
Attrition
No     3.150041
Yes    3.172996
Name: PerformanceRating, dtype: float64
```

```
df['JobSatisfaction']
```

	JobSatisfaction
0	4.0
1	2.0
2	2.0
3	4.0
4	1.0
...	...
4405	1.0
4406	4.0
4407	3.0
4408	1.0
4409	3.0

4410 rows × 1 columns

dtype: float64

```
df['JobSatisfaction'].value_counts()
```

JobSatisfaction	count
4.0	1387
3.0	1323
1.0	860
2.0	840

dtype: int64

```
df['Education']
```

	Education
0	2
1	1
2	4
3	5
4	1
...	...
4405	4
4406	4
4407	2
4408	2
4409	3

4410 rows × 1 columns

dtype: int64

```
df['Education'].value_counts()
```

Education	count
3	1716
4	1194
2	846
1	510
5	144

dtype: int64

```
df['BusinessTravel'].value_counts()
```

BusinessTravel	count
Travel_Rarely	3129
Travel_Frequently	831
Non-Travel	450

dtype: int64

```
df['YearsAtCompany'].value_counts()
```

YearsAtCompany	count
5	588
1	513
3	384
2	381
10	360
4	330
7	270
9	246
8	240
6	228
0	132
11	96
20	81
13	72
15	60
14	54
22	45
12	42
21	42
18	39
16	36
19	33
17	27
24	18
33	15
26	12
25	12
31	9
32	9
29	6

```
df.columns
```

```
Index(['EmployeeID', 'Age', 'Attrition', 'BusinessTravel', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'Gender', 'JobLevel',
       'JobRole', 'MaritalStatus', 'MonthlyIncome', 'NumCompaniesWorked',
       'PercentSalaryHike', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'YearsAtCompany', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'EnvironmentSatisfaction', 'JobSatisfaction',
       'WorkLifeBalance', 'JobInvolvement', 'PerformanceRating'],
      dtype='object')
```

40	3

```
for col in ['Education', 'EducationField', 'EnvironmentSatisfaction', 'JobSatisfaction', 'WorkLifeBalance', 'PerformanceRating', 'Age',
    print(f"Unique values for {col}:\n", df[col].value_counts(), "\n")
```

```
Unique values for Education:
 Education
3    1716
4    1194
2     846
1     510
5     144
Name: count, dtype: int64

Unique values for EducationField:
 EducationField
Life Sciences       1818
Medical             1392
Marketing            477
Technical Degree     396
```

```
Other                 246
Human Resources        81
Name: count, dtype: int64

Unique values for EnvironmentSatisfaction:
 EnvironmentSatisfaction
3.0    1375
4.0    1334
2.0     856
1.0     845
Name: count, dtype: int64

Unique values for JobSatisfaction:
 JobSatisfaction
4.0    1387
3.0    1323
1.0     860
2.0     840
Name: count, dtype: int64

Unique values for WorkLifeBalance:
 WorkLifeBalance
3.0    2698
2.0    1019
4.0     454
1.0     239
Name: count, dtype: int64

Unique values for PerformanceRating:
 PerformanceRating
3    3732
4     678
Name: count, dtype: int64

Unique values for Age:
 Age
35    234
34    231
31    207
36    207
29    204
32    183
30    180
```

```
df.columns
```

```
Index(['EmployeeID', 'Age', 'Attrition', 'BusinessTravel', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'Gender', 'JobLevel',
       'JobRole', 'MaritalStatus', 'MonthlyIncome', 'NumCompaniesWorked',
       'PercentSalaryHike', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'YearsAtCompany', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'EnvironmentSatisfaction', 'JobSatisfaction',
       'WorkLifeBalance', 'JobInvolvement', 'PerformanceRating'],
      dtype='object')
```

```
df ['BusinessTravel']
```

	BusinessTravel
0	Travel_Rarely
1	Travel_Frequently
2	Travel_Frequently
3	Non-Travel
4	Travel_Rarely
...	...
4405	Travel_Rarely
4406	Travel_Rarely
4407	Travel_Rarely
4408	Travel_Rarely
4409	Travel_Rarely

4410 rows × 1 columns

dtype: object

```
df['Education']
```

	Education
0	2
1	1
2	4
3	5
4	1
...	...
4405	4
4406	4
4407	2
4408	2
4409	3

4410 rows × 1 columns

dtype: int64

```
df['EducationField']
```

	EducationField
0	Life Sciences
1	Life Sciences
2	Other
3	Life Sciences
4	Medical
...	...
4405	Medical
4406	Medical
4407	Life Sciences
4408	Medical
4409	Medical

4410 rows × 1 columns

dtype: object

```
df['StockOptionLevel']
```

	StockOptionLevel
0	0
1	1
2	3
3	3
4	2
...	...
4405	1
4406	0
4407	0
4408	1
4409	0

4410 rows × 1 columns

dtype: int64

```
df['StockOptionLevel'].value_counts()
```

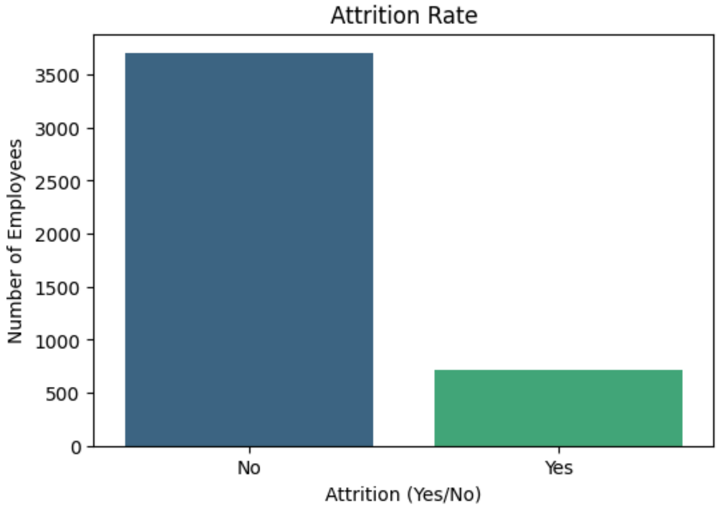
StockOptionLevel	count
0	1893
1	1788
2	474
3	255

dtype: int64

Exploratory Data Analysis

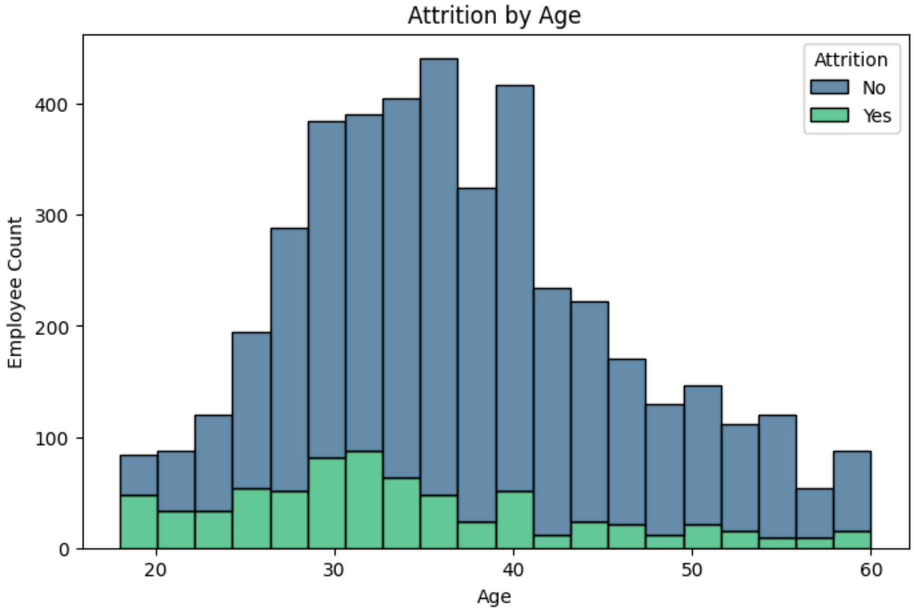
```
# Attrition Rate Plot
plt.figure(figsize=(6, 4))
attrition_counts = df['Attrition'].value_counts()
sns.barplot(x=attrition_counts.index, y=attrition_counts.values, palette='viridis')
plt.title("Attrition Rate")
plt.ylabel("Number of Employees")
plt.xlabel("Attrition (Yes/No)")
plt.show()
```

```
<ipython-input-63-85c2fac63d54>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.barplot(x=attrition_counts.index, y=attrition_counts.values, palette='viridis')
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



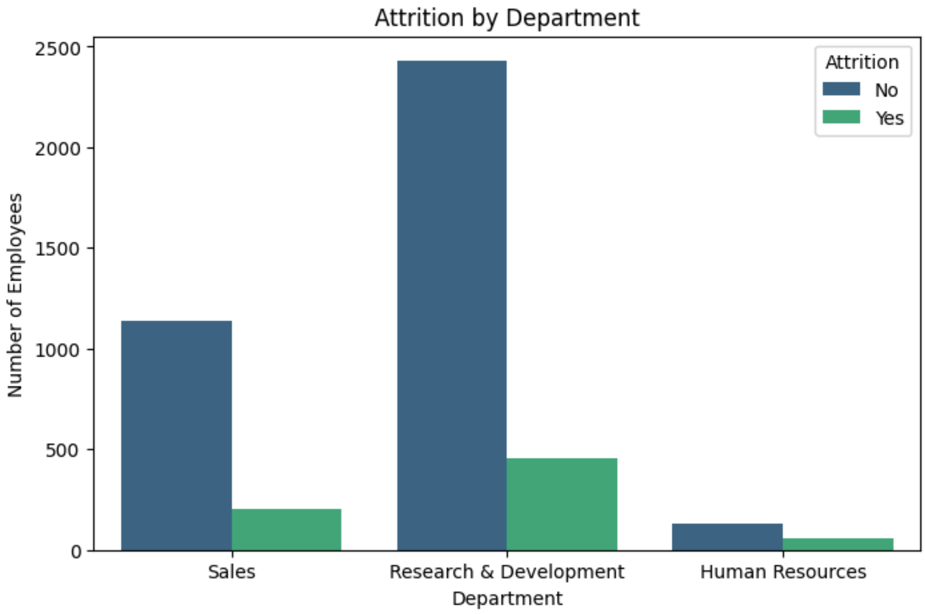

```
# Attrition by Age
plt.figure(figsize=(8, 5))
sns.histplot(data=df, x='Age', hue='Attrition', multiple='stack', palette='viridis', bins=20)
plt.title("Attrition by Age")
plt.xlabel("Age")
plt.ylabel("Employee Count")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



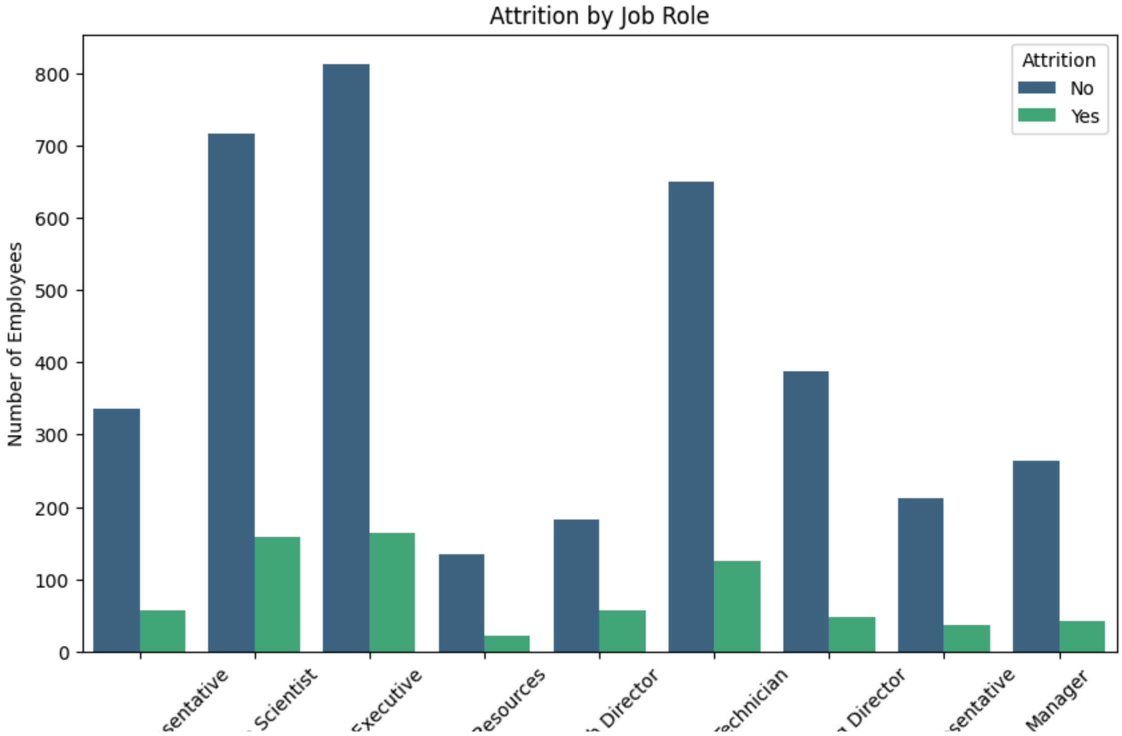

```
# Attrition by Department
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='Department', hue='Attrition', palette='viridis')
plt.title("Attrition by Department")
plt.ylabel("Number of Employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



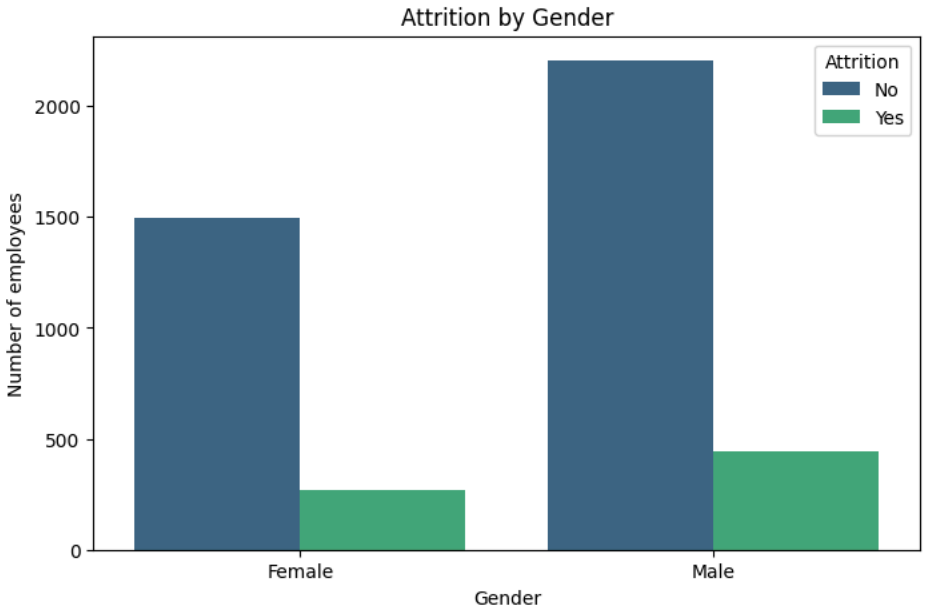

```
# Attrition by Job Role
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='JobRole', hue='Attrition', palette='viridis')
plt.xticks(rotation=45)
plt.title("Attrition by Job Role")
plt.ylabel("Number of Employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



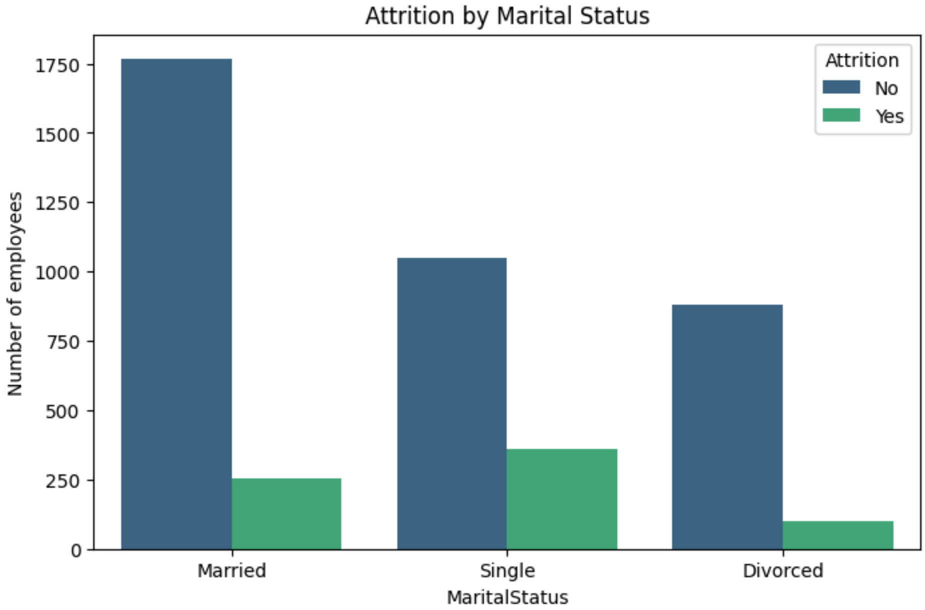

```
# Attrition by Gender
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='Gender', hue='Attrition', palette='viridis')
plt.title("Attrition by Gender")
plt.ylabel("Number of employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



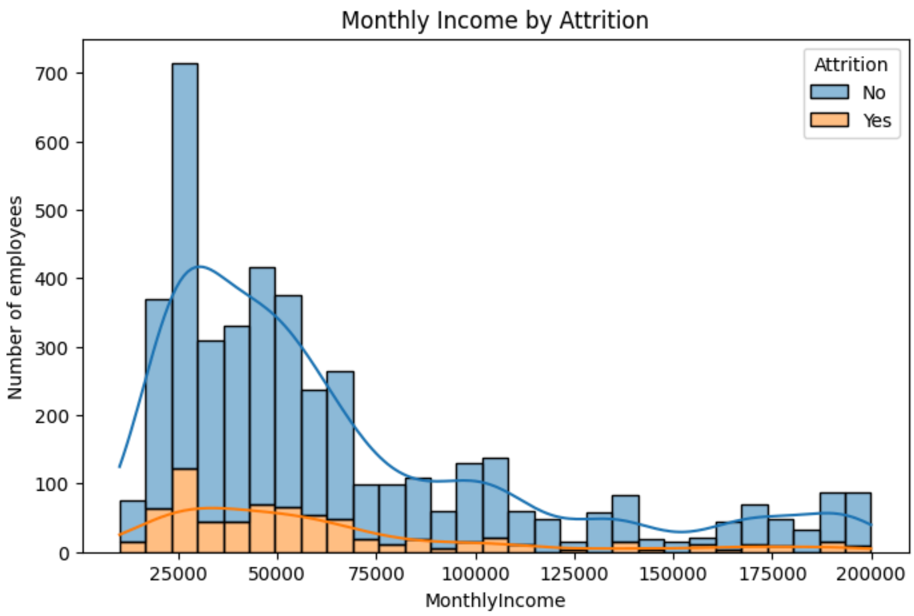

```
# Attrition by Marital Status
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='MaritalStatus', hue='Attrition', palette='viridis')
plt.title("Attrition by Marital Status")
plt.ylabel("Number of employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



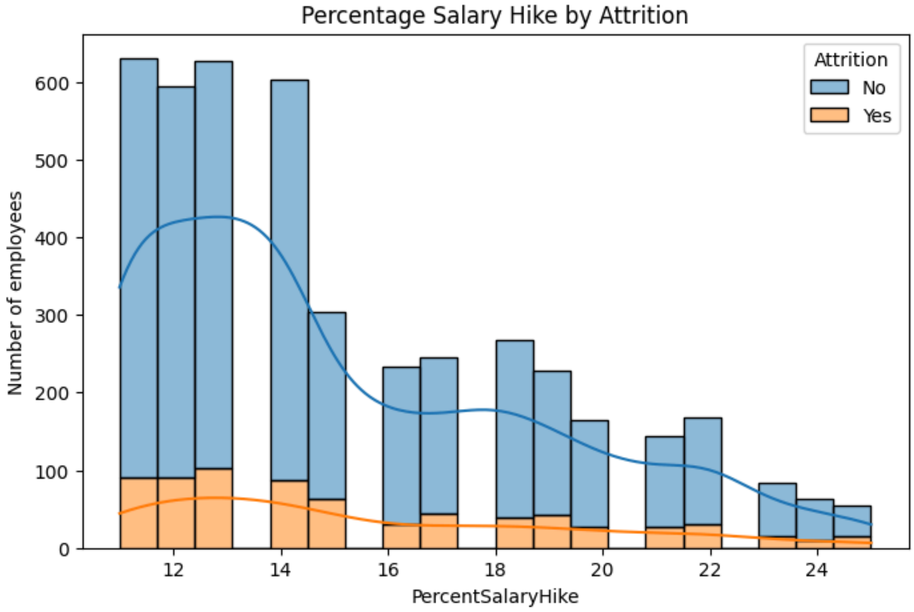

```
# Histogram for Monthly Income
plt.figure(figsize=(8, 5))
sns.histplot(data= df, x='MonthlyIncome', hue='Attrition', multiple='stack', kde=True)
plt.title('Monthly Income by Attrition')
plt.ylabel("Number of employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



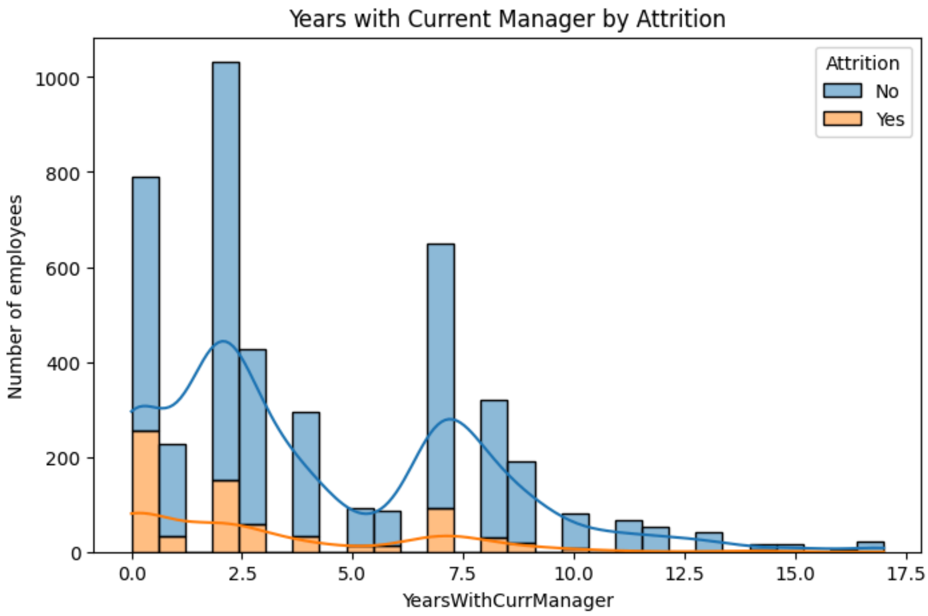

```
# Histogram for Percentage Salary Hike
plt.figure(figsize=(8, 5))
sns.histplot(data= df, x='PercentSalaryHike', hue='Attrition', multiple='stack', kde=True)
plt.title('Percentage Salary Hike by Attrition')
plt.ylabel("Number of employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



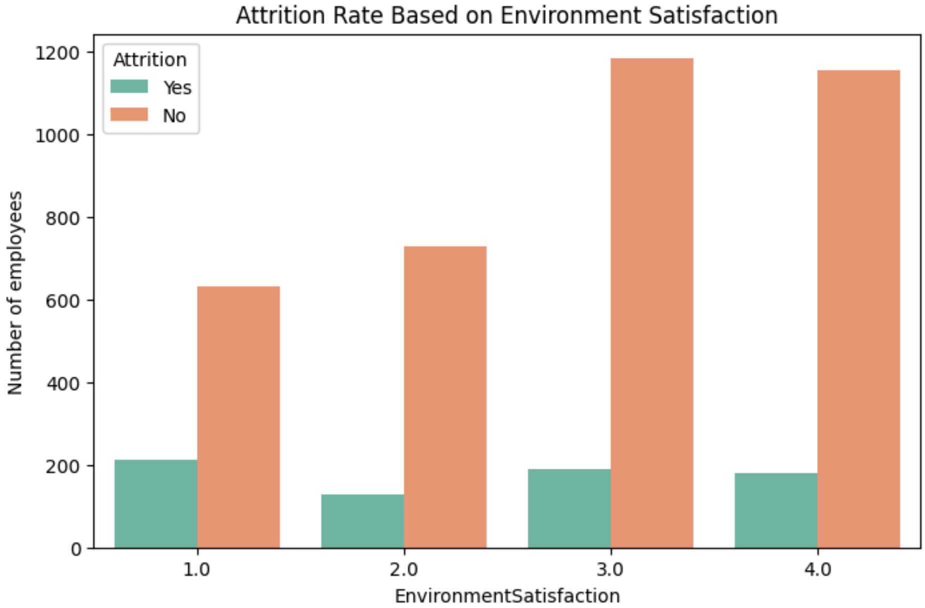

```
# Histogram for Years with Current Manager
plt.figure(figsize=(8, 5))
sns.histplot(data= df, x='YearsWithCurrManager', hue='Attrition', multiple='stack', kde=True)
plt.title('Years with Current Manager by Attrition')
plt.ylabel("Number of employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



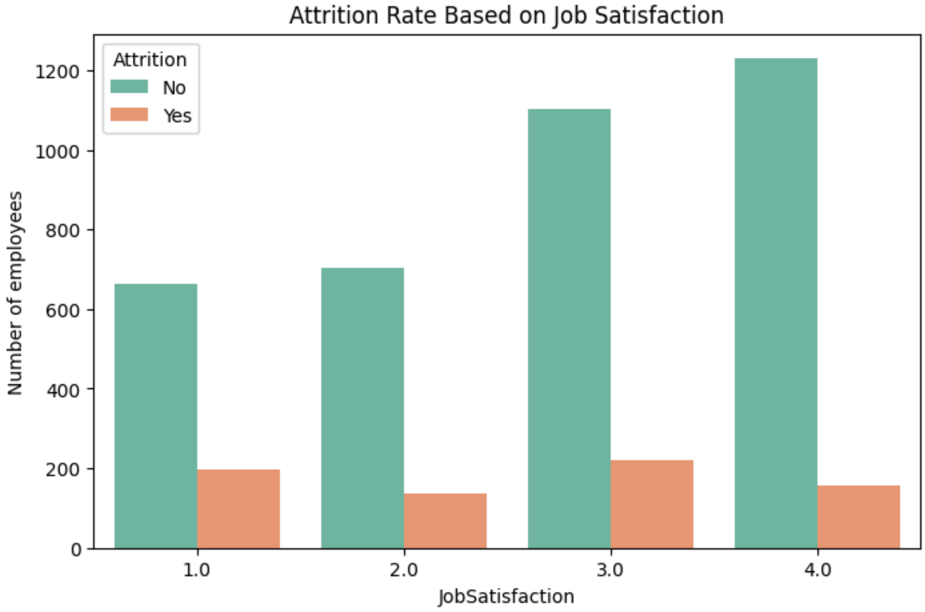
```
# Environment Satisfaction
plt.figure(figsize=(8, 5))
sns.countplot(data= df, x='EnvironmentSatisfaction', hue='Attrition', palette='Set2')
plt.title('Attrition Rate Based on Environment Satisfaction')
plt.ylabel("Number of employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```




```
# Job Satisfaction
plt.figure(figsize=(8, 5))
sns.countplot(data= df, x='JobSatisfaction', hue='Attrition', palette='Set2')
plt.title('Attrition Rate Based on Job Satisfaction')
plt.ylabel("Number of employees")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```

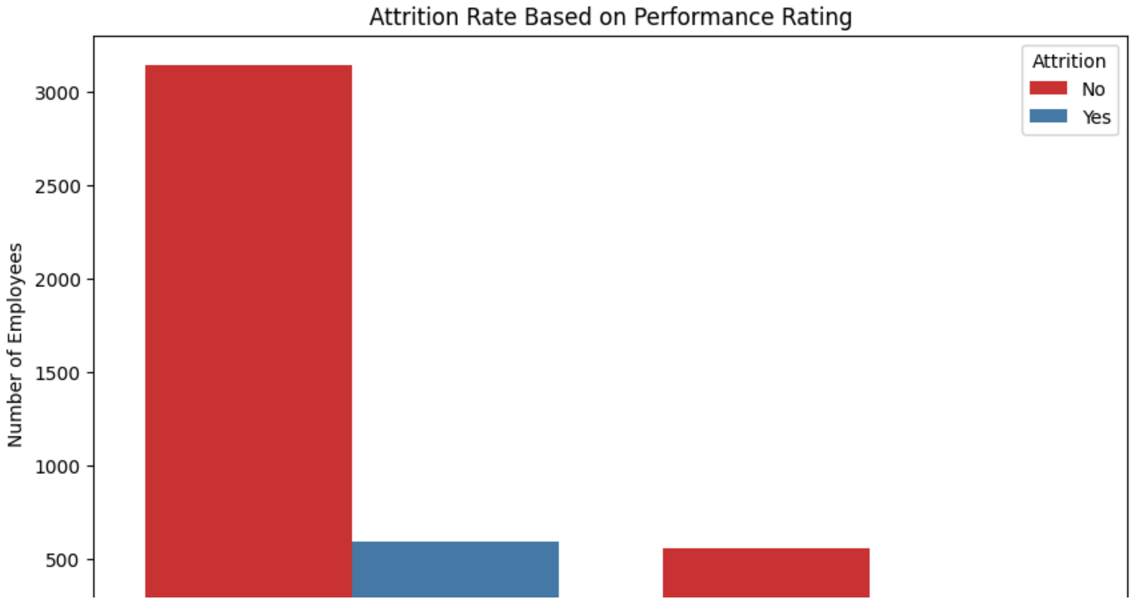



```
# Work-Life Balance
plt.figure(figsize=(8, 5))
sns.countplot(data= df, x='WorkLifeBalance', hue='Attrition', palette='Set2')
plt.title('Attrition Rate Based on Work-Life Balance')
plt.ylabel('Number of employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```

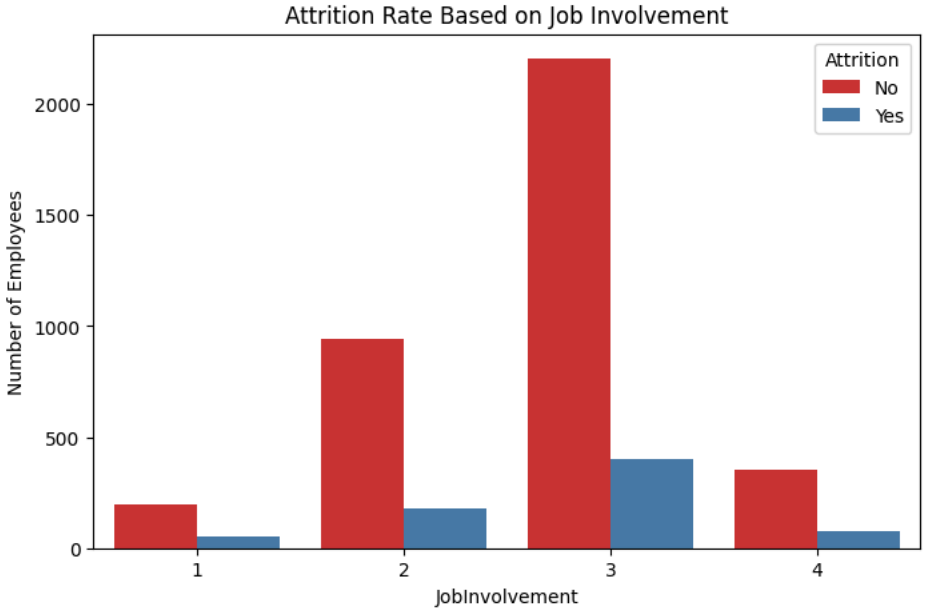

```
# Performance Rating
plt.figure(figsize=(10, 6))
sns.countplot(x='PerformanceRating', hue='Attrition', data=df, palette='Set1')
plt.title('Attrition Rate Based on Performance Rating')
plt.ylabel('Number of Employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



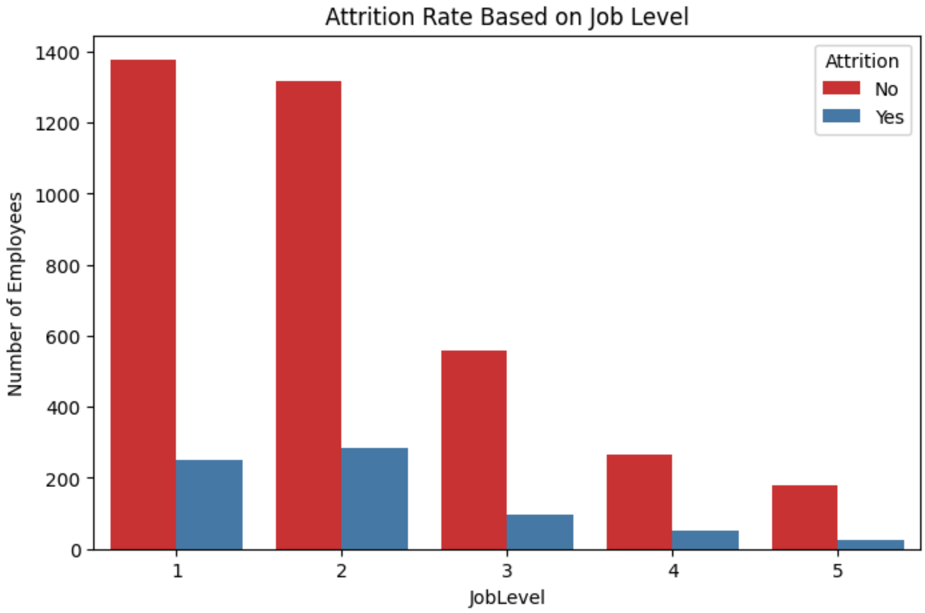

```
# Job Involvement
plt.figure(figsize=(8, 5))
sns.countplot(data= df, x='JobInvolvement', hue='Attrition', palette='Set1')
plt.title('Attrition Rate Based on Job Involvement')
plt.ylabel('Number of Employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



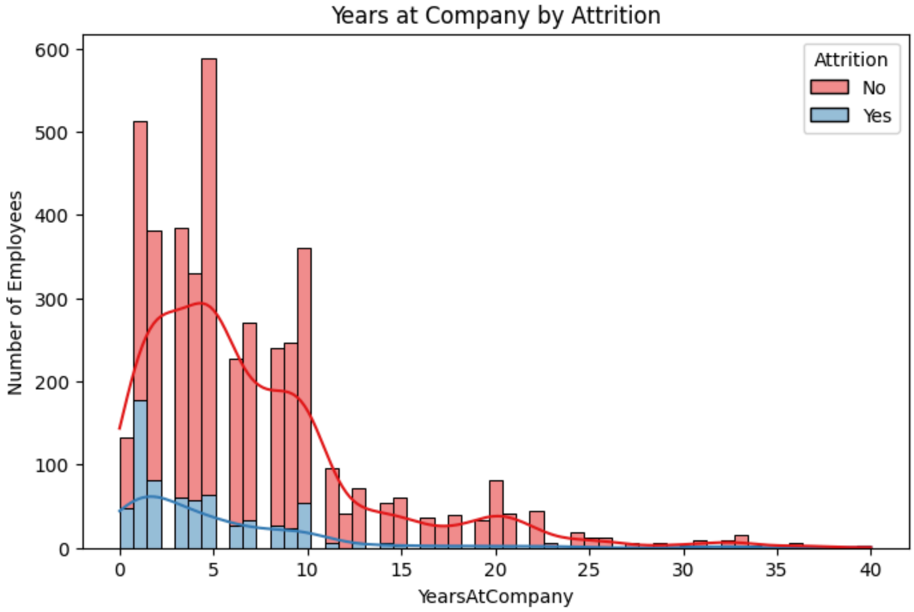

```
# Job Level
plt.figure(figsize=(8, 5))
sns.countplot(data= df, x='JobLevel', hue='Attrition', palette='Set1')
plt.title('Attrition Rate Based on Job Level')
plt.ylabel('Number of Employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```




```
# Histogram for Years at Company
plt.figure(figsize=(8, 5))
sns.histplot(df, x='YearsAtCompany', hue='Attrition', multiple='stack', palette='Set1', kde=True)
plt.title('Years at Company by Attrition')
plt.ylabel('Number of Employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```

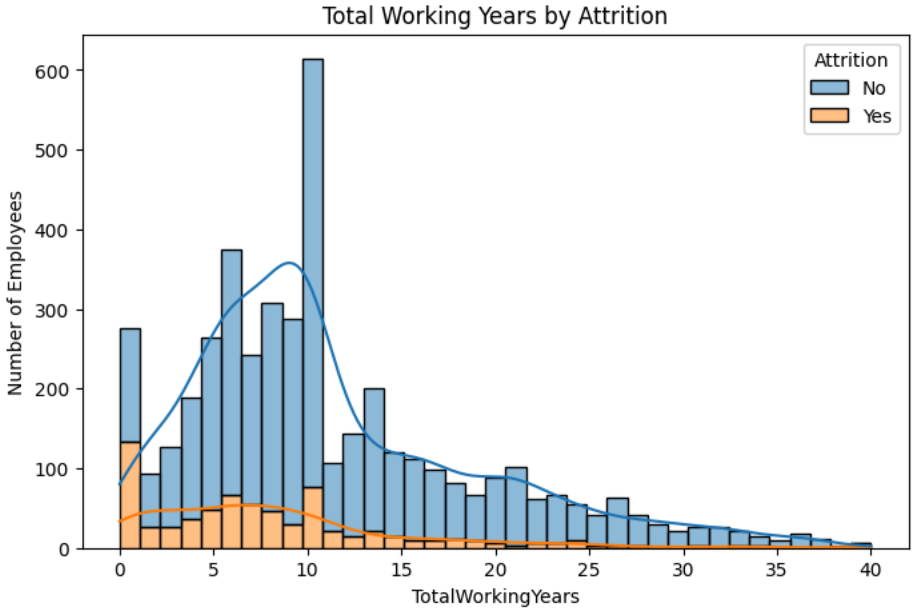



```
# Histogram for Number of Companies Worked
plt.figure(figsize=(8, 5))
sns.histplot(data= df, x='NumCompaniesWorked', hue='Attrition', multiple='stack', palette='Set1', kde=True)
plt.title('Number of Companies Worked by Attrition')
plt.ylabel('Number of Employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```

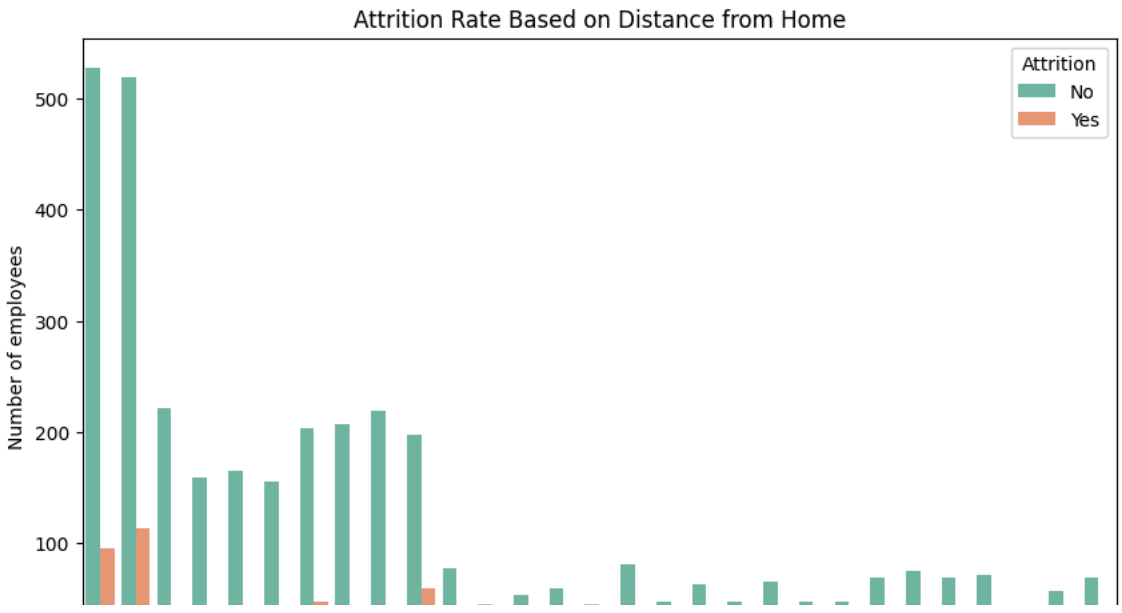
```
# Histogram for Total Working Years
plt.figure(figsize=(8, 5))
sns.histplot(df, x='TotalWorkingYears', hue='Attrition', multiple='stack', kde=True)
plt.title('Total Working Years by Attrition')
plt.ylabel('Number of Employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```




```
# Distance from Home to Work
plt.figure(figsize=(10, 6))
sns.countplot(x='DistanceFromHome', hue='Attrition', data=df, palette='Set2')
plt.title('Attrition Rate Based on Distance from Home')
plt.ylabel('Number of employees')
plt.show()
```

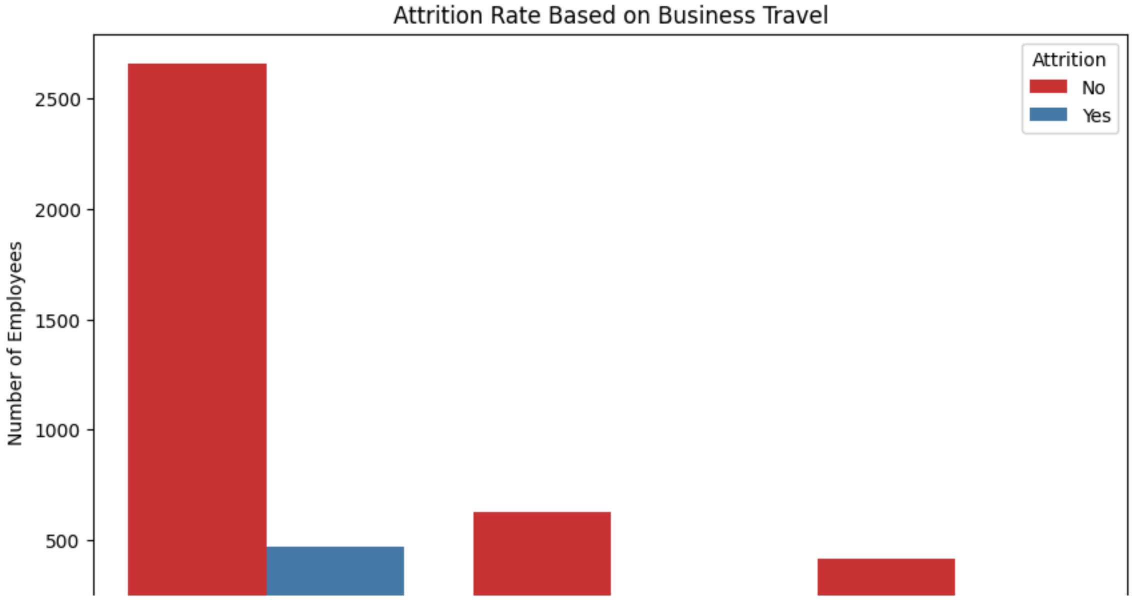
```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```




```
# Business Travel
plt.figure(figsize=(10, 6))
sns.countplot(x='BusinessTravel', hue='Attrition', data=df, palette='Set1')
plt.title('Attrition Rate Based on Business Travel')
```

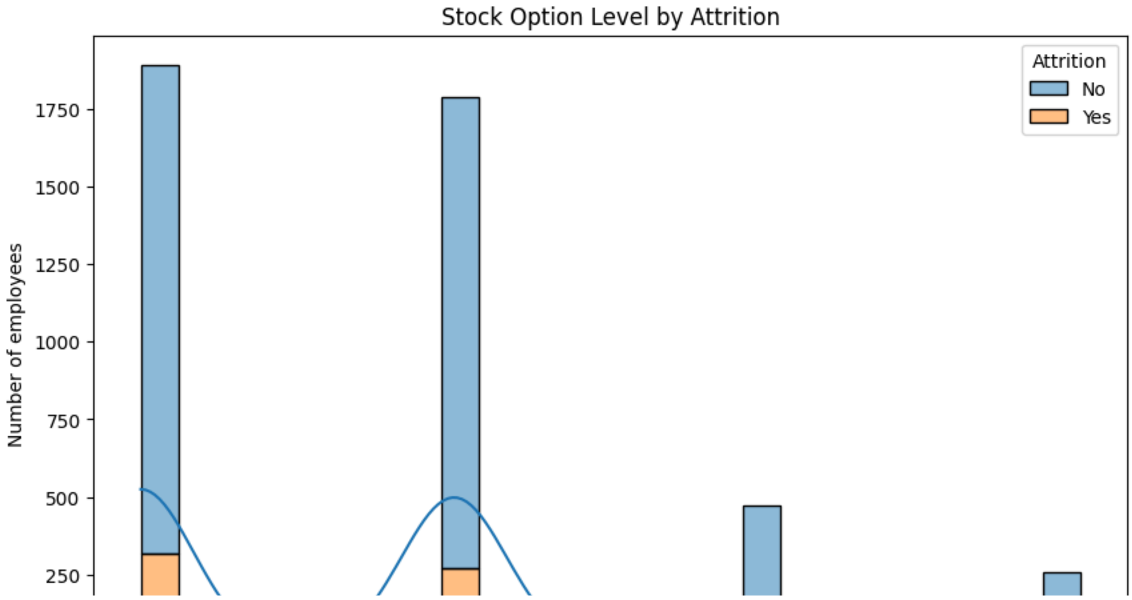
```
plt.ylabel('Number of Employees')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```




```
# Stock Option Level
plt.figure(figsize=(10, 6))
sns.histplot(data= df, x='StockOptionLevel', hue='Attrition', multiple='stack', kde=True)
plt.title('Stock Option Level by Attrition')
plt.ylabel('Number of employees')
plt.show()
```

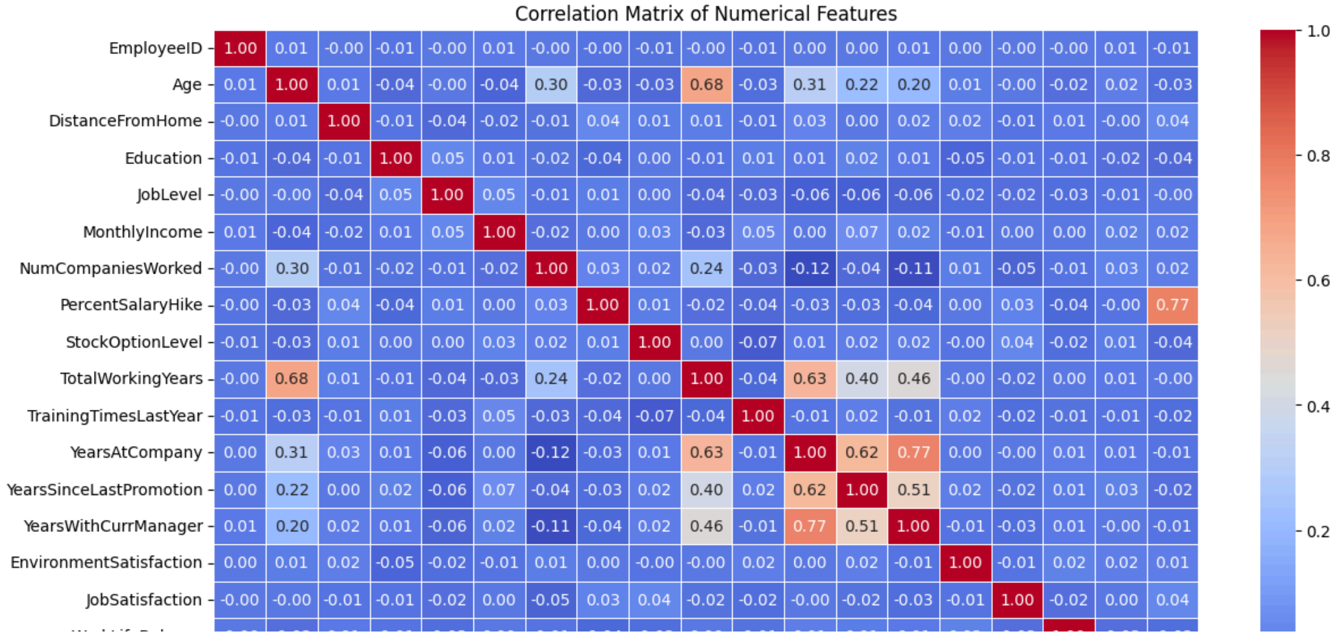
```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need
  data_subset = grouped_data.get_group(pd_key)
```



Correlation Analysis

StockOptionLevel

```
plt.figure(figsize=(14, 8))
correlation_matrix = df.corr(numeric_only=True)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



Key Observations

Strong Positive Correlations:

1. **Job Level and Monthly Income:** The correlation is high, indicating that as employees move to higher job levels, their monthly income increases significantly.
2. **Total Working Years and Monthly Income:** This suggests that employees with more experience tend to earn higher salaries.

Moderate Positive Correlations:

1. **Job Level and Total Working Years:** Employees with more working years tend to achieve higher job levels, which is expected as experience correlates with career progression.
2. **Years At Company and Years with Current Manager:** Employees who have been at the company longer often work with the same manager for extended periods.

Weak Correlations with Attrition:

1. **Years At Company and Attrition:** There is a slightly negative correlation, implying that employees who stay longer at the company are less likely to leave.
2. **Age and Attrition:** Older employees are slightly less likely to leave the company, but the correlation is weak.

Near Zero Correlations:

1. **Performance Rating and most other variables:** Performance rating does not show strong correlation with most features, indicating it's quite independent from these attributes.

These insights suggest factors like salary, experience, and tenure influence career progression and retention, while performance rating does not correlate as strongly with other factors in this dataset.

Feature Analysis and Predictive Analysis

Feature analysis refers to the process of understanding and selecting the most relevant attributes (features) in a dataset that contribute to the target outcome. In machine learning, features are the input variables used to predict the target variable.

Predictive analysis involves using statistical or machine learning techniques to build models that can predict an outcome based on input features. For example, predicting employee attrition using factors like age, job role, and satisfaction levels.

Steps for Feature and Predictive Analysis:

Feature Selection/Engineering:

1: Convert categorical features (e.g., JobRole, Gender, MaritalStatus) into numerical format using one-hot encoding or label encoding. Remove any irrelevant or redundant features (e.g., EmployeeID as it's just an identifier). Ensure the dataset is split into input features (X) and the target variable (Attrition).

2: Data Normalization/Standardization: For features like MonthlyIncome, Age, and other numerical columns, we may need to normalize the data to bring all features to the same scale, especially for algorithms like Logistic Regression.

Train-Test Split: Split the dataset into training and test sets (e.g., 80% training, 20% testing).

Predictive Modeling:

Train a few different models:

Logistic Regression: Basic classification.

Random Forest Classifier: For feature importance and handling complex interactions.

Support Vector Machine (SVM) or Gradient Boosting: For better performance on non-linear relationships. Evaluate models using metrics such as accuracy, precision, recall, F1 score, and ROC-AUC.

Feature Importance Analysis: Use techniques such as Random Forest or Logistic Regression Coefficients to analyze which features are the most important in predicting employee attrition.

Data Preparation

```
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer


df = pd.read_csv('Cleaned_Attrition_Data.csv')


# Drop unnecessary columns
data_cleaned = df.drop(columns=['EmployeeID'])


# Encode categorical variables using LabelEncoder
le = LabelEncoder()
categorical_cols = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus']
for col in categorical_cols:
    data_cleaned[col] = le.fit_transform(data_cleaned[col])


print(data_cleaned.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4410 entries, 0 to 4409
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Age                      4410 non-null   int64  
 1   Attrition                4410 non-null   object 
 2   BusinessTravel           4410 non-null   object 
 3   Department               4410 non-null   object 
 4   DistanceFromHome         4410 non-null   int64  
 5   Education                4410 non-null   int64  
 6   EducationField           4410 non-null   object 
 7   Gender                   4410 non-null   object 
 8   JobLevel                 4410 non-null   int64  
 9   JobRole                  4410 non-null   object 
 10  MaritalStatus            4410 non-null   object 
 11  MonthlyIncome            4410 non-null   int64  
 12  NumCompaniesWorked       4410 non-null   float64
 13  PercentSalaryHike        4410 non-null   int64  
 14  StockOptionLevel         4410 non-null   int64  
 15  TotalWorkingYears        4410 non-null   float64
 16  TrainingTimesLastYear    4410 non-null   int64  
 17  YearsAtCompany           4410 non-null   int64  
 18  YearsSinceLastPromotion  4410 non-null   int64  
 19  YearsWithCurrManager     4410 non-null   int64  
 20  EnvironmentSatisfaction  4410 non-null   float64
 21  JobSatisfaction          4410 non-null   float64
 22  WorkLifeBalance          4410 non-null   float64
 23  JobInvolvement           4410 non-null   int64  
 24  PerformanceRating        4410 non-null   int64  
dtypes: float64(5), int64(13), object(7)
memory usage: 861.5+ KB
None
```

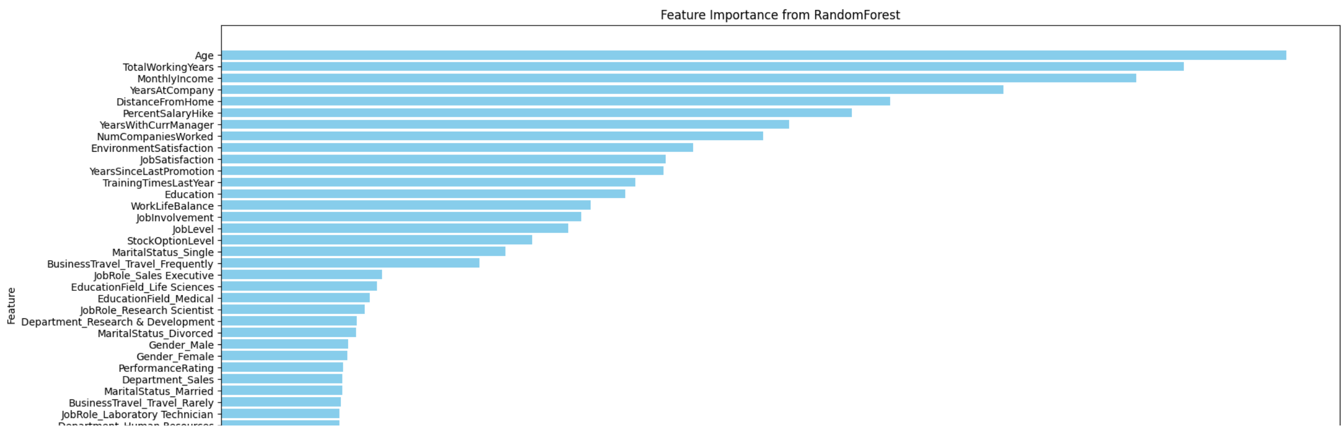
Feature Analysis

1: Define the features (X) and the target variable (y): The target is the Attrition column.

2: Train a RandomForestClassifier: To determine the importance of each feature.

3: Visualize the feature importances: using a bar plot.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# Define features (X) and target (y)
X = data_cleaned.drop(columns=['Attrition'])
y = data_cleaned['Attrition']

# One-Hot Encoding for categorical variables
categorical_cols = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus']

# Use ColumnTransformer to apply OneHotEncoder to categorical columns
preprocessor = ColumnTransformer(transformers=[
    ('cat', OneHotEncoder(), categorical_cols)
], remainder='passthrough')

# Transform the data
X_encoded = preprocessor.fit_transform(X)

# Initialize and train a Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_encoded, y)

# Get feature importances from the trained model
feature_importances = rf_model.feature_importances_

# Retrieve the names of the one-hot encoded columns
onehot_feature_names = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_cols)
all_feature_names = list(onehot_feature_names) + list(X.drop(columns=categorical_cols).columns)

# Create a DataFrame for visualization
features_df = pd.DataFrame({
    'Feature': all_feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(20, 10))
plt.barh(features_df['Feature'], features_df['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance from RandomForest')
plt.gca().invert_yaxis()  # Invert y-axis to display highest importance at the top
plt.show()
```



Predictive Analysis

1: Split the data into training and test sets.

2: Train a RandomForestClassifier on the training data.

3: Evaluate the model using: Confusion Matrix, Classification Report, ROC Curve

```
from sklearn.model_selection import train_test_split

# Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42, stratify=y)
```

```
# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Train the model on the training data
rf_model.fit(X_train, y_train)
```

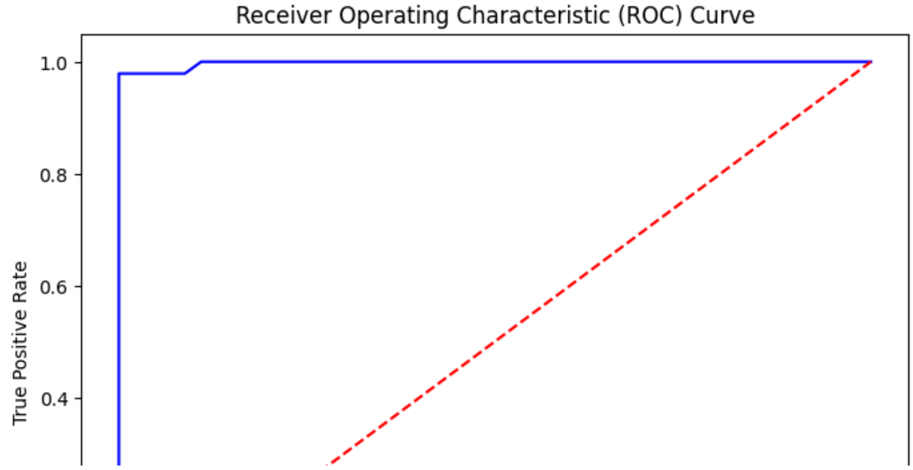
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
from sklearn.metrics import confusion_matrix, classification_report

# Predict on the test data
y_pred = rf_model.predict(X_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)

# Classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)
```

```
Confusion Matrix:
 [[740   0]
 [  4 138]]
Classification Report:
               precision    recall  f1-score   support

          No       0.99      1.00      1.00       740
         Yes       1.00      0.97      0.99       142

    accuracy                           1.00       882
   macro avg       1.00      0.99      0.99       882
weighted avg       1.00      1.00      1.00       882
```

```
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import LabelEncoder

# Encode the target variable y ('Yes' -> 1, 'No' -> 0)
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Now split the data using the encoded target
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded)

# Proceed with training the model and predicting probabilities
rf_model.fit(X_train, y_train)

# Get the probabilities for the positive class (attrition = 1)
y_pred_proba = rf_model.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_pred_proba)
print("AUC Score:", auc_score)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {auc_score:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
AUC Score: 0.9979158736200989
```




```
# Display the feature importances
print(features_df)
```

```
                             Feature  Importance
26                               Age    0.082524
34                 TotalWorkingYears    0.074584
30                     MonthlyIncome    0.070856
36                    YearsAtCompany    0.060606
27                  DistanceFromHome    0.051826
32                 PercentSalaryHike    0.048868
38              YearsWithCurrManager    0.044010
31                NumCompaniesWorked    0.041996
39           EnvironmentSatisfaction    0.036543
40                   JobSatisfaction    0.034446
37           YearsSinceLastPromotion    0.034240
35             TrainingTimesLastYear    0.032073
28                         Education    0.031274
41                   WorkLifeBalance    0.028600
42                    JobInvolvement    0.027903
29                          JobLevel    0.026856
33                  StockOptionLevel    0.024106
25              MaritalStatus_Single    0.022002
1     BusinessTravel_Travel_Frequently  0.020001
21             JobRole_Sales Executive  0.012469
7          EducationField_Life Sciences 0.012067
9              EducationField_Medical   0.011522
20          JobRole_Research Scientist  0.011127
4    Department_Research & Development  0.010508
23             MaritalStatus_Divorced   0.010462
13                        Gender_Male   0.009830
12                      Gender_Female   0.009753
43                  PerformanceRating   0.009430
5                    Department_Sales   0.009407
24              MaritalStatus_Married   0.009385
2        BusinessTravel_Travel_Rarely   0.009260
16      JobRole_Laboratory Technician   0.009181
3          Department_Human Resources   0.009133
22       JobRole_Sales Representative   0.007650
19          JobRole_Research Director   0.007445
6      EducationField_Human Resources   0.007419
14  JobRole_Healthcare Representative   0.006157
18      JobRole_Manufacturing Director  0.005773
8           EducationField_Marketing    0.005660
0           BusinessTravel_Non-Travel   0.005485
11    EducationField_Technical Degree   0.004903
17                    JobRole_Manager   0.004523
10              EducationField_Other    0.004373
15            JobRole_Human Resources   0.003764
```

Simulate Policy Change

```
import numpy as np

# Copy the test data for simulation
```