Dataset Introduction

Employee attrition refers to the rate at which employees leave a company over a given period of time. High attrition rates can lead to **increased recruitment costs**, **loss of experienced talent**, and **disruption in workflow**. Understanding the factors that contribute to attrition can help organizations create better employee retention strategies.

In this project, we will analyze the **IBM HR Analytics Attrition Dataset**, which contains various employee-related features such as `job role`, `satisfaction level`, `age`, `salary`, and more. By using machine learning techniques, ***we aim to build a predictive model that can estimate the likelihood of an employee leaving the company***.

Dataset Link:
Kaggle - IBM HR Analytics Attrition Dataset

This project aims to:

- **Explore** and **understand** the patterns in the HR dataset.
- **Identify** the key factors that influence employee attrition.
- **Build** and **evaluate** a predictive machine learning model for attrition.

Note: *This project is for educational and exploratory purposes only. In a real-world application, demographic variables which could be found in these kinds of datasets **should be reviewed for fairness implications** before deployment.*

Importing Dependecies

We import all necessary libraries for data manipulation, visualization and analysis.

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

Importing the dataset

We import the dataset from Google Drive. We simply connect our colab file to Google Drive by clicking the Drive icon on the navigation bar at the left or by running the code:
`from google.colab import drive`
`drive.mount('/content/drive')`

```
dataset = pd.read_csv('/content/drive/MyDrive/Machine Learning Datasets/HR Employees Attrition/HR-Employee-Attrition.csv')
dataset.head(5)
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	.
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	

5 rows × 35 columns

Dataset Overview

The **IBM HR Analytics Attrition Dataset** contains records of employees along with various demographic, job-related, and performance-related attributes. The data was designed for HR analytics and attrition prediction studies.

- **Total Records:** 1,470 employees
- **Total Features:** 35 columns (including the target variable `Attrition`)

Key Components

- **Target Variable:**
  - `Attrition`: Indicates whether an employee has left the company (`Yes` or `No`).
- **Demographic Attributes:**
  - `Age`, `Gender`, `MaritalStatus`, `Education`, `EducationField`.
- **Job-Related Attributes:**
  - `JobRole`, `Department`, `JobLevel`, `YearsAtCompany`, `YearsInCurrentRole`, `YearsWithCurrManager`, `YearsSinceLastPromotion`.
- **Work Environment & Performance:**
  - `OverTime`, `WorkLifeBalance`, `JobSatisfaction`, `EnvironmentSatisfaction`, `PerformanceRating`.
- **Compensation & Benefits:**
  - `MonthlyIncome`, `PercentSalaryHike`, `StockOptionLevel`.
- **Other Factors:**
  - `DistanceFromHome`, `TrainingTimesLastYear`, `NumCompaniesWorked`.

Data Type Summary

- **Categorical Features:** Job role, department, marital status, overtime, etc.
- **Numerical Features:** Age, monthly income, years at company, etc.

This dataset is **clean** (no missing values) and ready for analysis, making it an excellent choice for classification tasks such as predicting employee attrition.

```
print(f'Dataset has {dataset.shape[0]:,} rows and {dataset.shape[1]:,} columns.')
dataset.head(5)
```

```
Dataset has 1,470 rows and 35 columns.
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	.
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	

5 rows × 35 columns

Data types and Missing Values

Inspecting column data types and identifying missing values for cleaning and preprocessing.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
 ---  ------                    --------------  -----
  0   Age                       1470 non-null   int64
  1   Attrition                 1470 non-null   object
  2   BusinessTravel            1470 non-null   object
  3   DailyRate                 1470 non-null   int64
  4   Department                1470 non-null   object
  5   DistanceFromHome          1470 non-null   int64
  6   Education                 1470 non-null   int64
  7   EducationField            1470 non-null   object
  8   EmployeeCount             1470 non-null   int64
  9   EmployeeNumber            1470 non-null   int64
  10  EnvironmentSatisfaction   1470 non-null   int64
  11  Gender                    1470 non-null   object
  12  HourlyRate                1470 non-null   int64
  13  JobInvolvement            1470 non-null   int64
  14  JobLevel                  1470 non-null   int64
  15  JobRole                   1470 non-null   object
  16  JobSatisfaction           1470 non-null   int64
  17  MaritalStatus             1470 non-null   object
  18  MonthlyIncome             1470 non-null   int64
  19  MonthlyRate               1470 non-null   int64
  20  NumCompaniesWorked        1470 non-null   int64
  21  Over18                    1470 non-null   object
  22  OverTime                  1470 non-null   object
  23  PercentSalaryHike         1470 non-null   int64
  24  PerformanceRating         1470 non-null   int64
  25  RelationshipSatisfaction  1470 non-null   int64
  26  StandardHours             1470 non-null   int64
  27  StockOptionLevel          1470 non-null   int64
  28  TotalWorkingYears         1470 non-null   int64
  29  TrainingTimesLastYear     1470 non-null   int64
  30  WorkLifeBalance           1470 non-null   int64
  31  YearsAtCompany            1470 non-null   int64
  32  YearsInCurrentRole        1470 non-null   int64
  33  YearsSinceLastPromotion   1470 non-null   int64
  34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

```
dataset.isnull().sum()
```

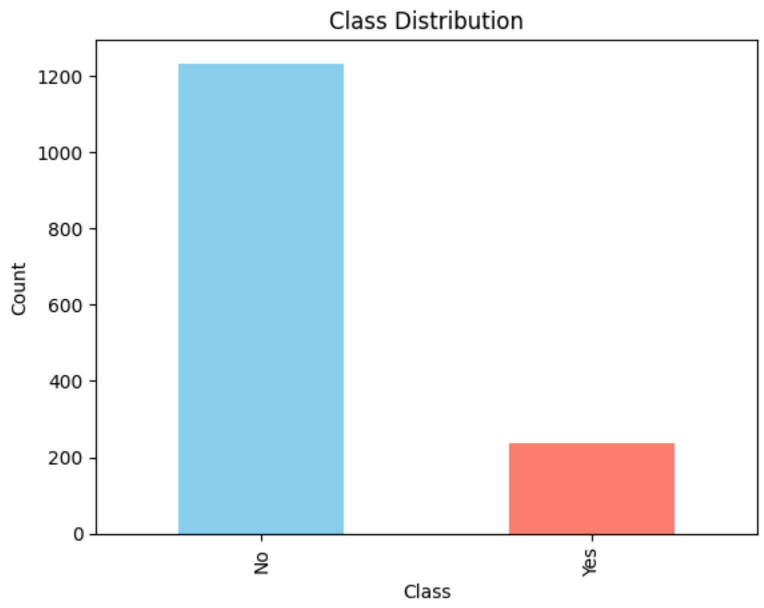
	0
Age	0
Attrition	0
BusinessTravel	0
DailyRate	0
Department	0
DistanceFromHome	0
Education	0
EducationField	0
EmployeeCount	0
EmployeeNumber	0
EnvironmentSatisfaction	0
Gender	0
HourlyRate	0
JobInvolvement	0
JobLevel	0
JobRole	0
JobSatisfaction	0
MaritalStatus	0
MonthlyIncome	0
MonthlyRate	0
NumCompaniesWorked	0
Over18	0
OverTime	0
PercentSalaryHike	0
PerformanceRating	0
RelationshipSatisfaction	0
StandardHours	0
StockOptionLevel	0
TotalWorkingYears	0
TrainingTimesLastYear	0
WorkLifeBalance	0
YearsAtCompany	0
YearsInCurrentRole	0
YearsSinceLastPromotion	0
YearsWithCurrManager	0

dtype: int64

Class Distribution

We check for class imbalance. If class count differ significantly a possible use case scenario for SMOTE might be needed later on to normalize the class distribution.

```
dataset['Attrition'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



Statistical Summary

We use the .describe() method to obtain a statistical summary of the numerical features. This includes measures such as mean, standard deviation, and quartiles, which help assess the central tendency and spread of data.

```
dataset.describe()
```

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobIn
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	1470.000000	1470.000000	14
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	2.721769	65.891156	
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	1.093082	20.329428	
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	1.000000	30.000000	
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	2.000000	48.000000	
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	3.000000	66.000000	
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	4.000000	83.750000	
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	4.000000	100.000000	

8 rows × 26 columns

Unique values for categorical features

Checking different categorical values in each columns. It helps in planning encoding strategies later on in the process.

```
categorical_cols = dataset.select_dtypes(include=['object']).columns
print(f'There are {len(categorical_cols)} categorical columns in the dataset.\n')
for col in categorical_cols:
  print(f'{col}: {dataset[col].unique()}\n')
```

```
There are 9 categorical columns in the dataset.

Attrition: ['Yes' 'No']

BusinessTravel: ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']

Department: ['Sales' 'Research & Development' 'Human Resources']

EducationField: ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical Degree'
 'Human Resources']

Gender: ['Female' 'Male']
```

```
JobRole: ['Sales Executive' 'Research Scientist' 'Laboratory Technician'
 'Manufacturing Director' 'Healthcare Representative' 'Manager'
 'Sales Representative' 'Research Director' 'Human Resources']

MaritalStatus: ['Single' 'Married' 'Divorced']

Over18: ['Y']

OverTime: ['Yes' 'No']
```

Exploratory Data Analysis

Univariate Analysis

Univariate analysis focuses on examining **one feature at a time** to understand its distribution, range, and general characteristics without considering other variables.

This step is important because it helps to:

- Identify data patterns and distribution shapes (normal, skewed, uniform, etc.).
- Spot unusual values or outliers that may require further attention.
- Understand the scale and variability of numerical features.
- Detect imbalanced categories in categorical variables, which could impact model training.

In this dataset, univariate analysis will be applied to:

- **Numerical Features** → Histograms, KDE plots, and boxplots.
- **Categorical Features** → Bar charts showing category frequencies.

These insights provide a foundation for later stages of the analysis, such as correlation checks and bivariate comparisons with the target variable.

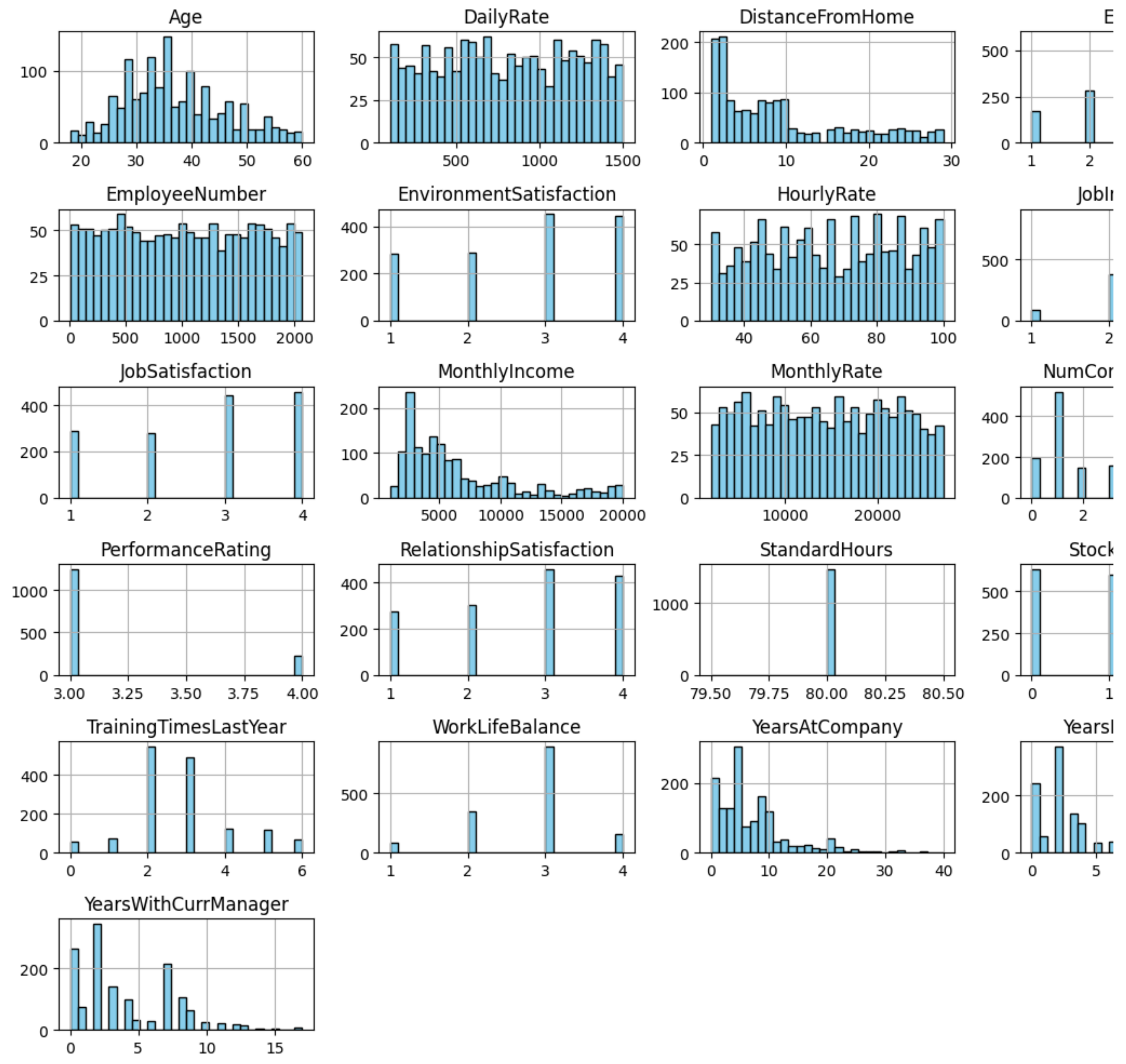
Numerical Feature Distribution

The dataset contains a significant number of numerical variables, such as `Age`, `MonthlyIncome`, `YearsAtCompany`, and `DistanceFromHome`.
Examining the distribution of these features is an essential first step in **Exploratory Data Analysis (EDA)** for several reasons:

- **Understanding Data Patterns:** Helps reveal the general shape (normal, skewed, uniform, etc.) of the data for each numerical feature.
- **Identifying Outliers:** Extreme values can affect model performance, especially for distance-based algorithms.
- **Spotting Data Entry Issues:** Unusual spikes or unrealistic values may indicate errors.
- **Guiding Preprocessing:** Knowing the distribution helps decide whether transformations (e.g., log scaling) or normalization are needed.

By visualizing these distributions, we can gain an initial sense of the data's range, spread, and potential problem areas before moving into more complex analysis.

```python
numerical_cols = dataset.select_dtypes(include=['int64','float64']).columns
dataset[numerical_cols].hist(figsize=(15, 10), bins=30, color='skyblue', edgecolor='black')
plt.tight_layout()
plt.show()
```

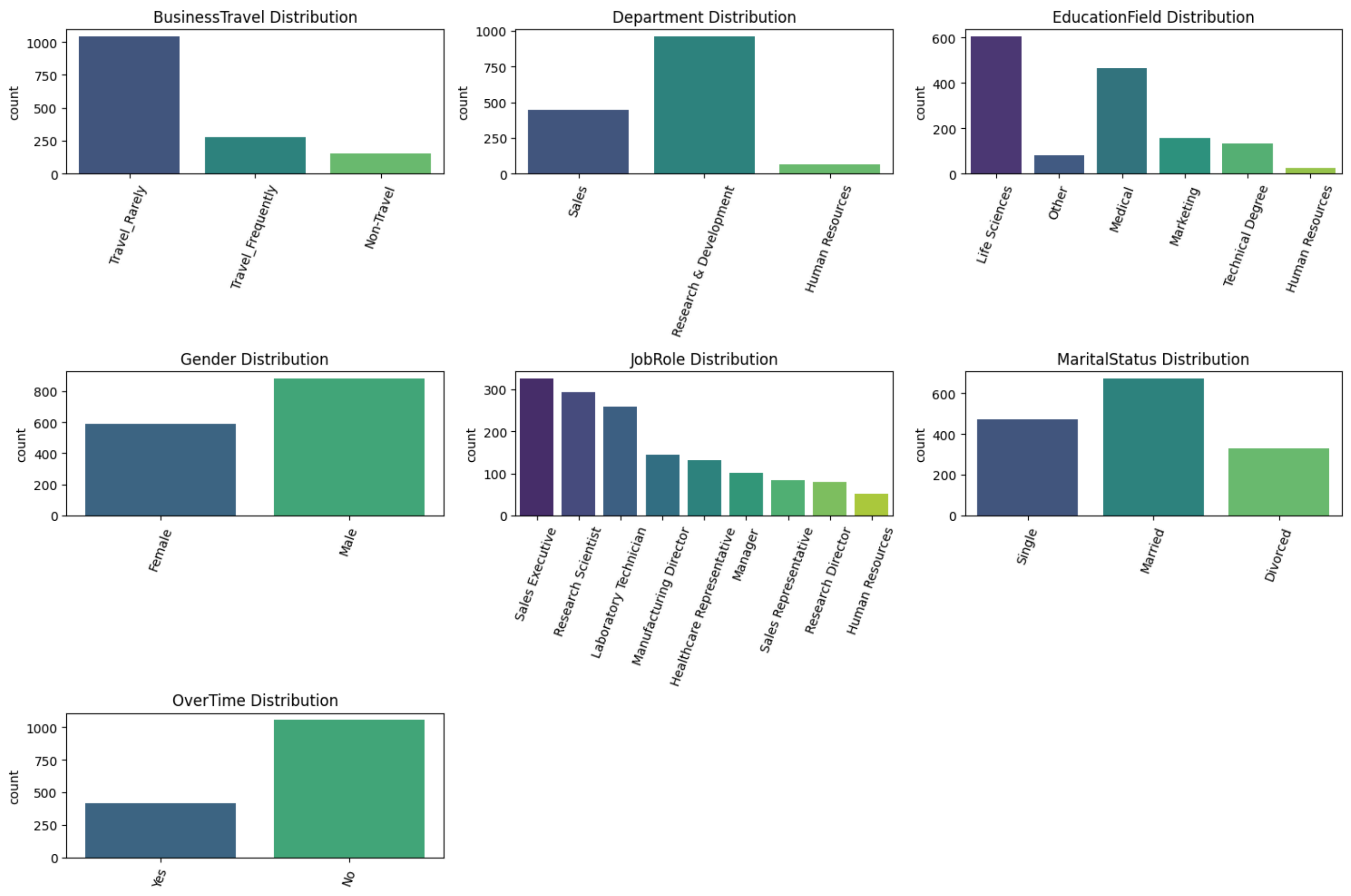


Categorical Feature Distribution

Categorical variables such as `JobRole`, `Department`, `MaritalStatus`, and `OverTime` provide insights into employee demographics and work arrangements.
Exploring these distributions is important because:

- It shows the frequency of each category in the dataset.
- It helps identify imbalanced categories that could bias the model.
- It provides context for later comparisons with attrition.

```
cat_cols = ['BusinessTravel', 'Department', 'EducationField',
            'Gender', 'JobRole', 'MaritalStatus', 'OverTime']

plt.figure(figsize=(15, 10))
for i, col in enumerate(cat_cols, 1):
    plt.subplot(3, 3, i)
    sns.countplot(data=dataset, x=col, palette='viridis', hue=col, legend=False)
    plt.title(f'{col} Distribution')
    plt.xlabel('')
    plt.xticks(rotation=70)
plt.tight_layout()
plt.show()
```

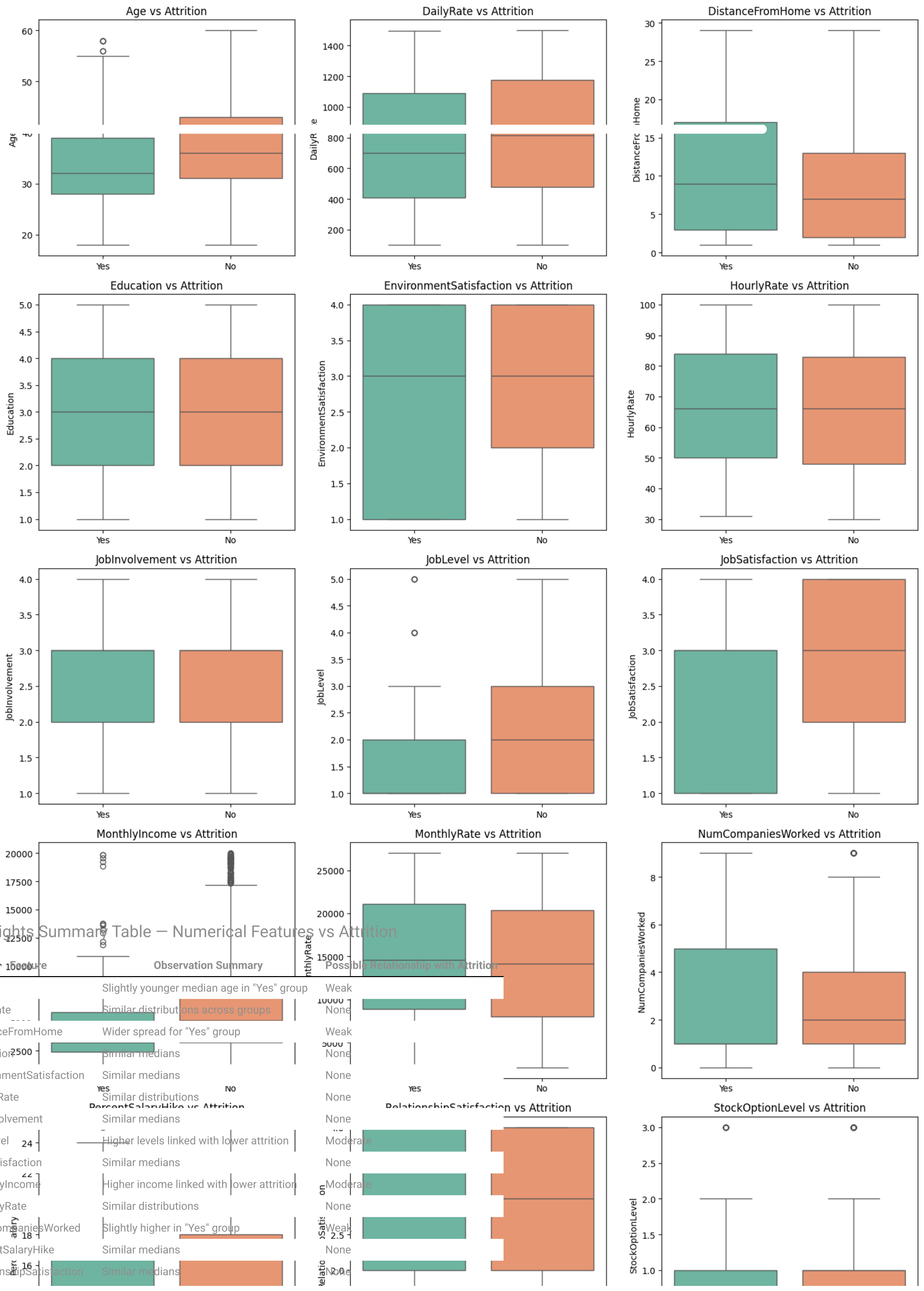


Bivariate Analysis

Bivariate analysis examines the relationship between two variables at a time.
Here, we will study how each numerical and categorical feature varies with `Attrition`. This step helps us:

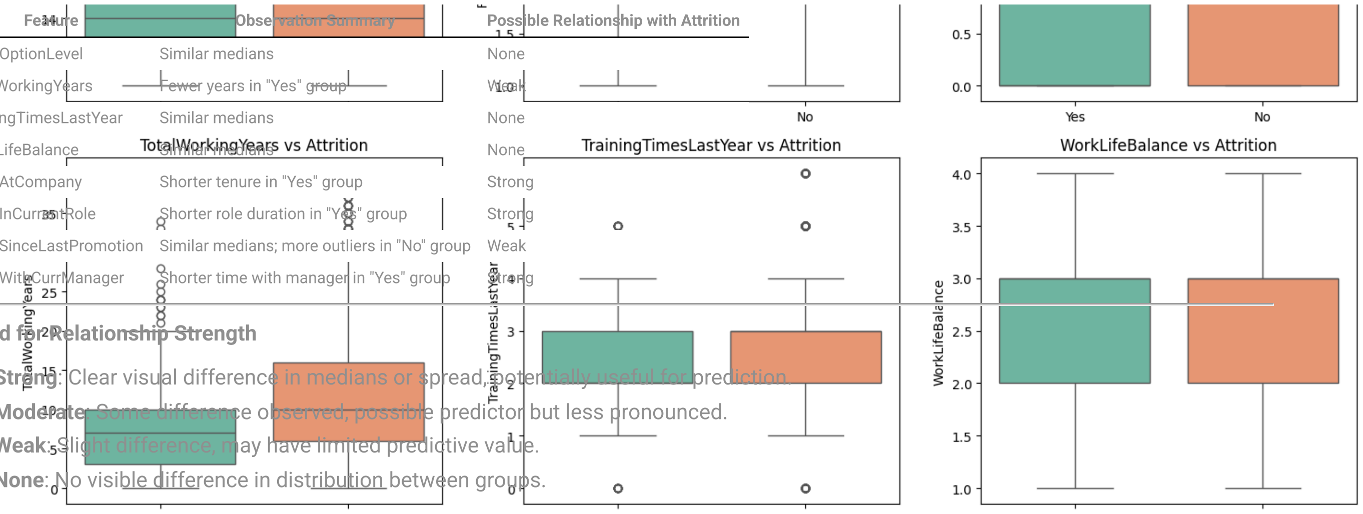
- Identify features that show different patterns between employees who left and those who stayed.
- Detect potential predictors of attrition early on.
- Understand how certain factors (e.g., overtime, income, years of service) relate to employee turnover.

```
num_cols = dataset.select_dtypes(include=['int64', 'float64']).columns.drop(['EmployeeNumber', 'EmployeeCount', 'PerformanceRating', 'Standa

plt.figure(figsize=(15, 35))
for i, col in enumerate(num_cols, 1):
    plt.subplot((len(num_cols) // 3) + 1, 3, i)
    sns.boxplot(data=dataset,x='Attrition', y=col, palette='Set2', hue='Attrition', legend=False)
    plt.title(f'{col} vs Attrition')
    plt.xlabel('')
plt.tight_layout()
plt.show()
```



Insights Summary Table — Numerical Features vs Attrition

Feature	Observation Summary	Possible Relationship with Attrition
Age	Slightly younger median age in "Yes" group	Weak
DailyRate	Similar distributions across groups	None
DistanceFromHome	Wider spread for "Yes" group	Weak
Education	Similar medians	None
EnvironmentSatisfaction	Similar medians	None
HourlyRate	Similar distributions	None
JobInvolvement	Similar medians	None
JobLevel	Higher levels linked with lower attrition	Moderate
JobSatisfaction	Similar medians	None
MonthlyIncome	Higher income linked with lower attrition	Moderate
MonthlyRate	Similar distributions	None
NumCompaniesWorked	Slightly higher in "Yes" group	Weak
PercentSalaryHike	Similar medians	None
RelationshipSatisfaction	Similar medians	None

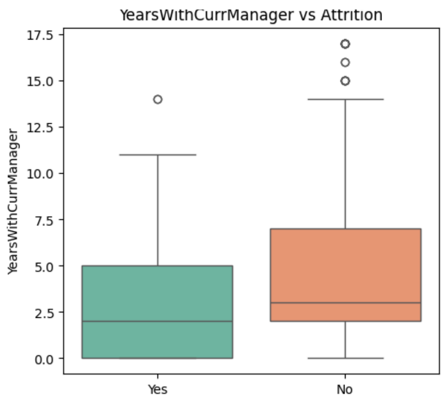


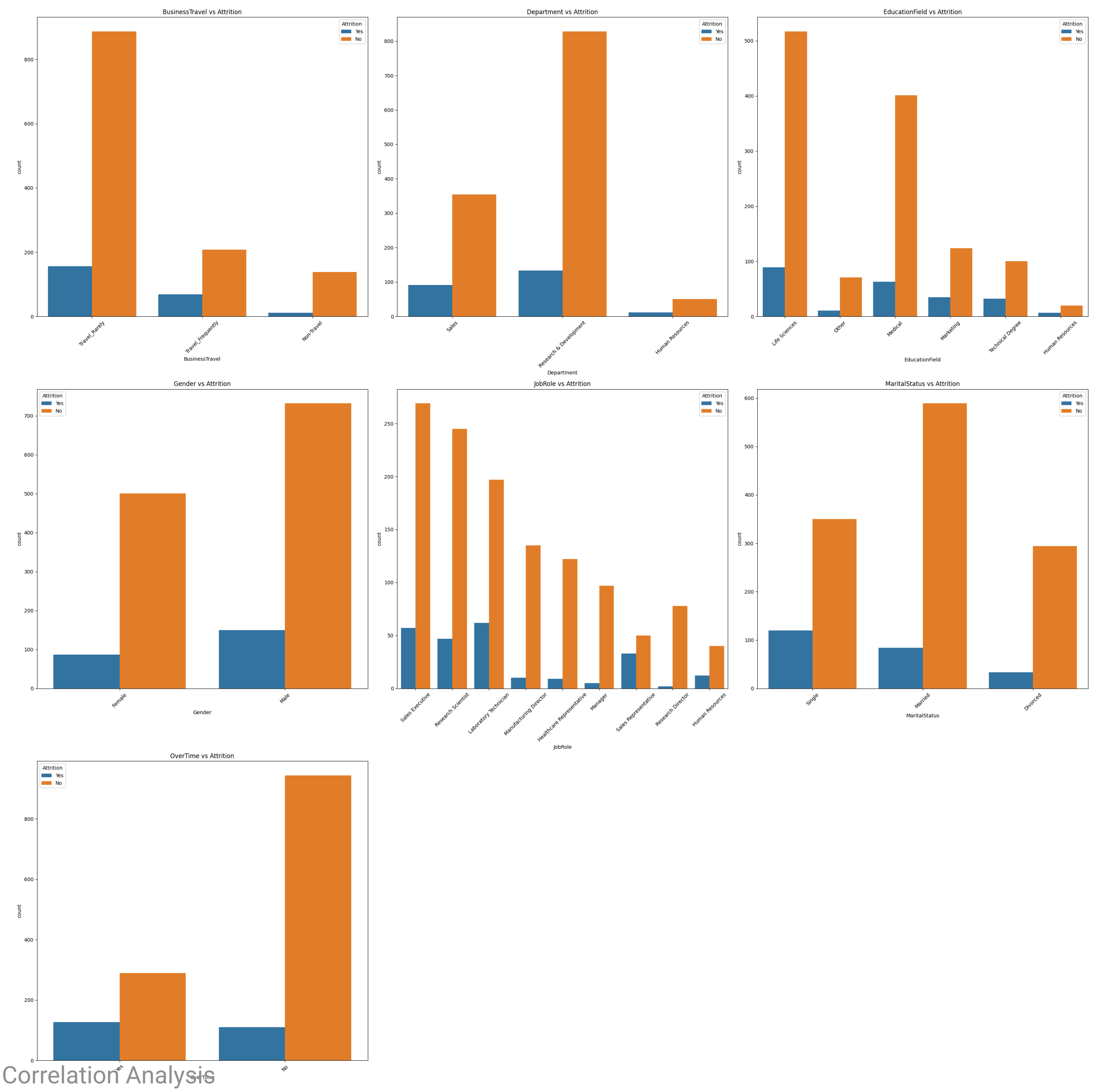
Feature	Observation Summary	Possible Relationship with Attrition
StockOptionLevel	Similar medians	None
TotalWorkingYears	Fewer years in "Yes" group	Weak
TrainingTimesLastYear	Similar medians	None
WorkLifeBalance	Similar medians	None
YearsAtCompany	Shorter tenure in "Yes" group	Strong
YearsInCurrentRole	Shorter role duration in "Yes" group	Strong
YearsSinceLastPromotion	Similar medians; more outliers in "No" group	Weak
YearsWithCurrManager	Shorter time with manager in "Yes" group	Strong

Legend for Relationship Strength

- **Strong**: Clear visual difference in medians or spread, potentially useful for prediction.
- **Moderate**: Some difference observed, possible predictor but less pronounced.
- **Weak**: Slight difference, may have limited predictive value.
- **None**: No visible difference in distribution between groups.

```
categorical_cols = ['BusinessTravel', 'Department', 'EducationField','Gender', 'JobRole', 'MaritalStatus', 'OverTime']

n_rows = math.ceil(len(categorical_cols) / 3)
fig, axes = plt.subplots(nrows=n_rows, ncols=3, figsize=(30, 10 * n_rows))
axes = axes.flatten()

for i, col in enumerate(categorical_cols):
    sns.countplot(data=dataset, x=col, hue='Attrition', ax=axes[i])
    axes[i].set_title(f"{col} vs Attrition")
    axes[i].tick_params(axis='x', rotation=45)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



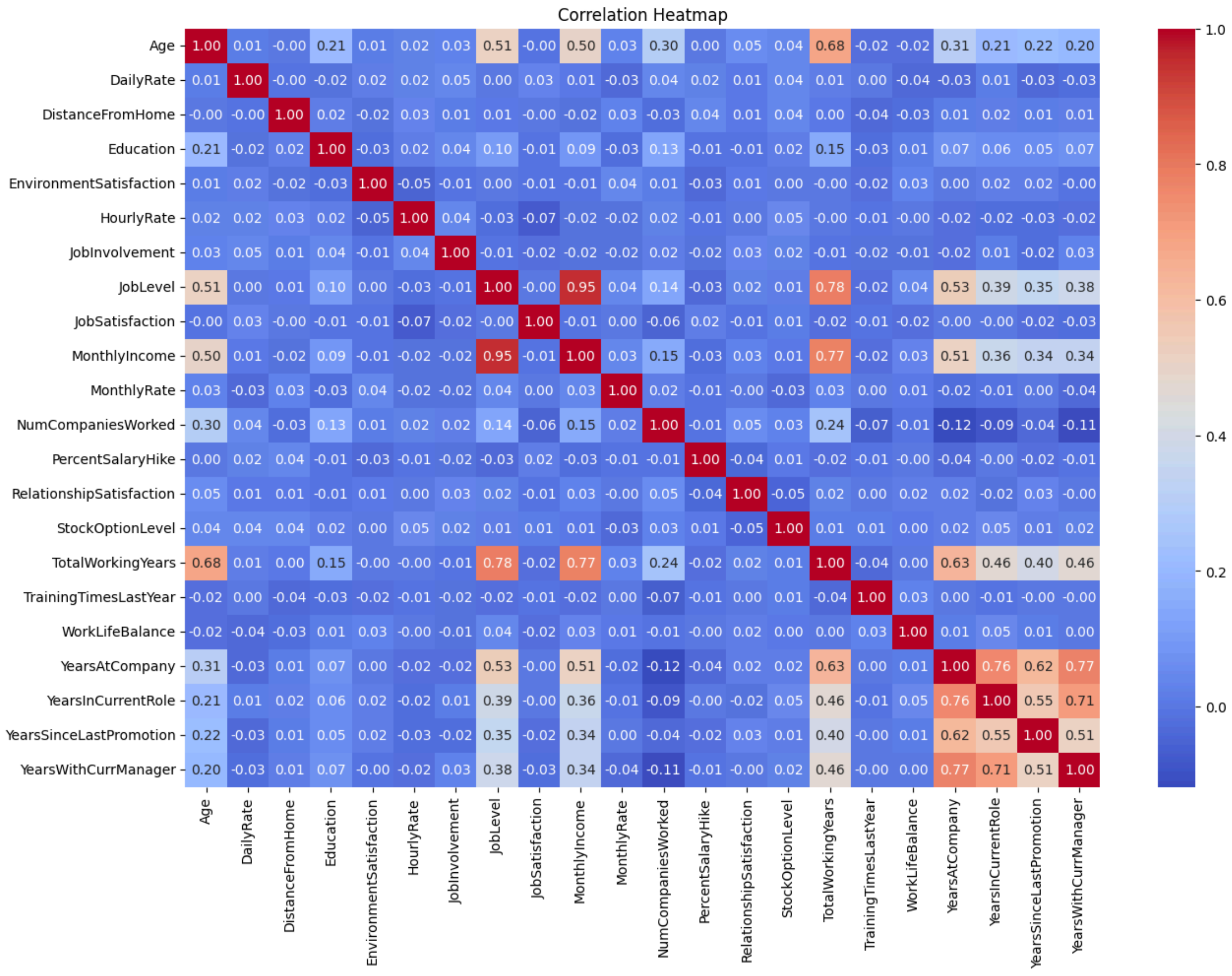


Correlation Analysis

We now check the relationship between numerical variables using correlation heatmap. This helps:

- Identify redundant features (high correlation, rule of thumb: `|correlation| > 0.85` consider dropping one feature).
- Spot variables with potential predictive power for attrition.

```
plt.figure(figsize=(15,10))
corr = dataset[num_cols].corr()
plt.title('Correlation Heatmap')
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```



Data Processing

This section focuses on preparing our dataset by handling **missing values**, **encoding categorical variables**, and **scaling numerical features**.

Handling Missing Data**

- Missing or null values can negatively impact model performance. Thus, we must check if out dataset contains any null values.
- Depending on the column type and context, missing values will be:
  - Filled using statistical measures (mean, median, mode).
  - Dropped if the percentage of missing data is too high or if the column is irrelevant.

```
dataset.isnull().sum()
```

	0
Age	0
Attrition	0
BusinessTravel	0
DailyRate	0
Department	0
DistanceFromHome	0
Education	0
EducationField	0
EmployeeCount	0
EmployeeNumber	0
EnvironmentSatisfaction	0
Gender	0
HourlyRate	0
JobInvolvement	0
JobLevel	0
JobRole	0
JobSatisfaction	0
MaritalStatus	0
MonthlyIncome	0
MonthlyRate	0
NumCompaniesWorked	0
Over18	0
OverTime	0
PercentSalaryHike	0
PerformanceRating	0
RelationshipSatisfaction	0
StandardHours	0
StockOptionLevel	0
TotalWorkingYears	0
TrainingTimesLastYear	0
WorkLifeBalance	0
YearsAtCompany	0
YearsInCurrentRole	0
YearsSinceLastPromotion	0
YearsWithCurrManager	0

dtype: int64

No **Null Values** thus, we are clear to proceed to the next step of data processing.

Encoding Categorical Variables

- Machine learning algorithms cannot directly process categorical data.
- We will use:
  - **Label Encoding** for binary categorical features.
  - **One-Hot Encoding** for multi-class categorical features to prevent introducing ordinal relationships where none exist.

```
categorical_cols = dataset.select_dtypes(include=['object']).columns.drop('Attrition', errors='ignore')

binary_features = [col for col in categorical_cols if dataset[col].nunique() == 2]
multi_class_features = [col for col in categorical_cols if dataset[col].nunique() > 2]

binary_features, multi_class_features
```

```
(['Gender', 'OverTime'],
 ['BusinessTravel',
  'Department',
  'EducationField',
  'JobRole',
  'MaritalStatus'])
```

Binary Categorical Features

Binary categorical features contain only two unique values.

- **Nominal:** Categories without any intrinsic order.
  Example: *Gender* (Male/Female)
- **Ordinal:** Categories with a clear ranking. (Not applicable in this dataset's binary columns.)

We will apply **Label Encoding** to binary nominal features.

Feature	Type	Encoding Method
Gender	Nominal	Label Encoding (0/1)
OverTime	Nominal	Label Encoding (0/1)

```
from sklearn.preprocessing import LabelEncoder

binary_nominal = ['Gender', 'OverTime']
dataset_binary_encoded = dataset.copy()

label_encoder = LabelEncoder()
for col in binary_nominal:
    print(f"{col} unique values before encoding: {dataset_binary_encoded[col].unique()}")
    dataset_binary_encoded[col] = label_encoder.fit_transform(dataset_binary_encoded[col])
    print(f"{col} unique values after encoding: {dataset_binary_encoded[col].unique()}\n")
```

```
Gender unique values before encoding: ['Female' 'Male']
Gender unique values after encoding: [0 1]

OverTime unique values before encoding: ['Yes' 'No']
OverTime unique values after encoding: [1 0]
```

Multi-Class Categorical Features

Multi-class features have more than two distinct categories.

- **Nominal:** No ranking between categories. Example: *Department*, *JobRole*
- **Ordinal:** Categories with a natural order. Example: *Education Level*

For **nominal** features, we use **One-Hot Encoding** to avoid implying order.
For **ordinal** features, we apply **Ordinal Encoding** using a custom mapping.

Feature	Type	Encoding Method
BusinessTravel	Nominal	One-Hot Encoding
Department	Nominal	One-Hot Encoding
EducationField	Nominal	One-Hot Encoding
JobRole	Nominal	One-Hot Encoding
MaritalStatus	Nominal	One-Hot Encoding

```
multi_nominal = ['BusinessTravel', 'Department', 'EducationField', 'JobRole', 'MaritalStatus']

dataset_multi_encoded = dataset.copy()
print("Applying One-Hot Encoding for multi-class nominal features...\n")
dataset_multi_encoded = pd.get_dummies(dataset_multi_encoded, columns=multi_nominal, drop_first=True)

dataset_multi_encoded.head()
```

```
Applying One-Hot Encoding for multi-class nominal features...
```

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	Gender	HourlyRate	..
0	41	Yes	1102	1	2	1	1	2	Female	94	
1	49	No	279	8	1	1	2	3	Male	61	
2	37	Yes	1373	2	2	1	4	4	Male	92	
3	33	No	1392	3	4	1	5	4	Female	56	
4	27	No	591	2	1	1	7	1	Male	40	

5 rows × 49 columns

Combining both encoded categorical columns.

```
dataset_multi_encoded = pd.get_dummies(dataset_binary_encoded, columns=multi_class_features, drop_first=True)
dataset_multi_encoded.head()
```

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	Gender	HourlyRate	..
0	41	Yes	1102	1	2	1	1	2	0	94	
1	49	No	279	8	1	1	2	3	1	61	
2	37	Yes	1373	2	2	1	4	4	1	92	
3	33	No	1392	3	4	1	5	4	0	56	
4	27	No	591	2	1	1	7	1	1	40	

5 rows × 49 columns

Feature Scaling

Feature scaling is the process of transforming the numerical features of a dataset so that they share a similar scale. This step is important because many machine learning algorithms — such as gradient descent–based models, k-nearest neighbors, and support vector machines — are sensitive to the range of values in the input features. Without scaling, features with larger ranges may dominate the learning process, leading to biased model performance. In this project, only continuous numerical variables will be scaled. Ordinal variables (such as ratings or ranked survey responses) are excluded from scaling to preserve their discrete and ordered meaning.

Identifying Continuous vs. Ordinal Numeric Features

The following code block examines all numeric columns in the dataset (after categorical encoding) and separates them into two groups: potential ordinal and potential continuous.

1. **Potential Ordinal**: Columns with a `small number of unique integer values (≤ 10)`. These are **likely categorical features stored as numbers**, such as `satisfaction ratings` or coded `education levels`.
2. **Potential Continuous**: Columns with a `larger range of unique values`, which are typically genuine continuous variables like `Age` or `MonthlyIncome`.

By performing this separation, we ensure that only continuous features are passed to the scaling process, while ordinal features retain their discrete value representation.

```
numeric_candidates = dataset_multi_encoded.select_dtypes(include=['int64', 'float64']).columns

potential_ordinal = []
potential_continuous = []

for col in numeric_candidates:
    unique_count = dataset_multi_encoded[col].nunique()
    if unique_count <= 10:
        potential_ordinal.append((col, unique_count))
    else:
        potential_continuous.append((col, unique_count))

print("Potential Ordinal Numeric Columns:")
for col, uniq in potential_ordinal:
    print(f" - {col} ({uniq} unique values)")

print("\nPotential Continuous Columns:")
for col, uniq in potential_continuous:
    print(f" - {col} ({uniq} unique values)")
```

```
Potential Ordinal Numeric Columns:
 - Education (5 unique values)
 - EmployeeCount (1 unique values)
 - EnvironmentSatisfaction (4 unique values)
 - Gender (2 unique values)
 - JobInvolvement (4 unique values)
 - JobLevel (5 unique values)
 - JobSatisfaction (4 unique values)
 - NumCompaniesWorked (10 unique values)
 - OverTime (2 unique values)
 - PerformanceRating (2 unique values)
 - RelationshipSatisfaction (4 unique values)
 - StandardHours (1 unique values)
 - StockOptionLevel (4 unique values)
 - TrainingTimesLastYear (7 unique values)
 - WorkLifeBalance (4 unique values)

Potential Continuous Columns:
 - Age (43 unique values)
 - DailyRate (886 unique values)
 - DistanceFromHome (29 unique values)
 - EmployeeNumber (1470 unique values)
 - HourlyRate (71 unique values)
 - MonthlyIncome (1349 unique values)
 - MonthlyRate (1427 unique values)
 - PercentSalaryHike (15 unique values)
 - TotalWorkingYears (40 unique values)
 - YearsAtCompany (37 unique values)
 - YearsInCurrentRole (19 unique values)
 - YearsSinceLastPromotion (16 unique values)
 - YearsWithCurrManager (18 unique values)
```

In this step, we applied ***Standard Scaling*** to ensure that all continuous numerical features have a mean of 0 and a standard deviation of 1. This is important because many machine learning algorithms, especially distance-based methods (e.g., KNN, logistic regression, SVM), can be sensitive to differences in feature magnitude. Without scaling, variables with larger numeric ranges can dominate the learning process and skew the model.

```
from sklearn.preprocessing import StandardScaler

continuous_cols = [col for col, _ in potential_continuous]

scaler = StandardScaler()
dataset_multi_encoded[continuous_cols] = scaler.fit_transform(dataset_multi_encoded[continuous_cols])

dataset_multi_encoded[continuous_cols].head()
```

	Age	DailyRate	DistanceFromHome	EmployeeNumber	HourlyRate	MonthlyIncome	MonthlyRate	PercentSalaryHike	TotalWorkingYears
0	0.446350	0.742527	-1.010909	-1.701283	1.383138	-0.108350	0.726020	-1.150554	-0.421642
1	1.322365	-1.297775	-0.147150	-1.699621	-0.240677	-0.291719	1.488876	2.129306	-0.164511
2	0.008343	1.414363	-0.887515	-1.696298	1.284725	-0.937654	-1.674841	-0.057267	-0.550208
3	-0.429664	1.461466	-0.764121	-1.694636	-0.486709	-0.763634	1.243211	-1.150554	-0.421642
4	-1.086676	-0.524295	-0.887515	-1.691313	-1.274014	-0.644858	0.325900	-0.877232	-0.678774

Data Splitting

Before building any machine learning model, it is essential to split the dataset into separate subsets for training and testing.

Steps Taken:

1. **Encoded the Target Variable**

- The Attrition column contained "Yes" and "No" values.
- Converted to binary representation:
    - `"Yes"` → 1 (employee left)
    - `"No"` → 0 (employee stayed)

2. **Separated Features and Target**

- Features (X): All columns except Attrition
- Target (y): The encoded Attrition column

3. **Train-Test Split**

- Train set: 80% of the data (used for model training)
- Test set: 20% of the data (used for final evaluation)
- Used `stratify=y` to maintain the same proportion of attrition cases in both sets, preventing skewed evaluation results.

```
from sklearn.model_selection import train_test_split

processed_dataset = dataset_multi_encoded
processed_dataset = processed_dataset.drop('Over18', axis=1)

processed_dataset['Attrition'] = processed_dataset['Attrition'].map({'Yes': 1, 'No': 0})

X = processed_dataset.drop('Attrition', axis=1)
y = processed_dataset['Attrition']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Train set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
print("Target distribution in train set:\n", y_train.value_counts(normalize=True))
```

```
Train set shape: (1176, 47)
Test set shape: (294, 47)
Target distribution in train set:
 Attrition
0    0.838435
1    0.161565
Name: proportion, dtype: float64
```

Handling Class Imbalance with SMOTE

The target variable **Attrition** is highly imbalanced, with the majority of employees labeled as "No" (did not leave) and only a small proportion labeled as "Yes" (left the company).
This imbalance can bias machine learning models toward predicting the majority class, leading to poor performance on the minority class, which is the primary class of interest.

To address this issue, we apply **Synthetic Minority Oversampling Technique (SMOTE)**.
SMOTE works by creating synthetic examples of the minority class (Attrition = 1) based on its existing feature space, instead of simply duplicating records. This results in a balanced training dataset while preserving the original feature distribution.

Important: SMOTE is applied **only to the training set**. The test set remains untouched to ensure the evaluation reflects the real-world distribution of attrition cases.

After applying SMOTE, the training dataset has a balanced distribution of Attrition = 0 and Attrition = 1, which helps the model learn patterns from both classes more effectively.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("Original training set shape:", y_train.value_counts())
print("Resampled training set shape:", y_train_resampled.value_counts())
```

```
Original training set shape: Attrition
```