LIBRARIES

This section imports all the necessary Python libraries required throughout the project:

- **pandas**: For data manipulation and analysis (e.g., reading datasets, working with dataframes).
- **numpy**: For numerical operations and array handling.
- **matplotlib.pyplot** and **seaborn**: For data visualization, enabling plots like bar charts, histograms, and heatmaps.
- **sklearn** modules: For building machine learning models, preprocessing data, splitting datasets, and evaluating performance.

By importing all required libraries at the start, we ensure the rest of the code runs efficiently without interruption.

In [52]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import pickle
```

DATA SET

In this section, we load the dataset that will be used for analysis and modeling.

- The dataset is typically read into a **pandas DataFrame** using functions like `pd.read_csv()`.
- After loading, we may display the first few rows using `head()` to get a quick look at the structure and contents.
- This step helps verify that the data has been imported correctly and gives a preliminary view of the features (columns) and data types.

Understanding the dataset at this stage is crucial for effective preprocessing and modeling later.

In [53]:
```
df
```

Out[53]:

	Age	MonthlyIncome	DistanceFromHome	YearsAtCompany	NumCompaniesWorked	P
0	50	8056	3	31	9	
1	36	17948	19	22	4	
2	29	11110	16	32	6	
3	42	16773	16	2	3	
4	40	3502	3	17	0	
...	...	...	...	...	...	
95	58	8423	28	29	6	
96	56	17589	1	4	3	
97	43	10158	20	11	2	
98	48	13248	29	15	9	
99	56	10400	21	25	4	

100 rows × 10 columns

STEP 1: DATA COLLECTION AND UNDERSTANDING

In this step, we perform an initial exploration of the dataset to understand its structure and contents.

Key activities typically include:

- **Viewing dataset shape** (rows and columns) to understand its size.
- **Inspecting column names and data types** to identify features and their types (e.g., numerical, categorical).
- **Checking for missing values or duplicates**, which may require cleaning.
- **Using descriptive statistics** (`describe()` , `info()`) to summarize distributions, ranges, and potential outliers.
- **Visualizing relationships** between variables using plots like histograms, boxplots, or correlation heatmaps.

This foundational understanding helps inform decisions for preprocessing and feature engineering.

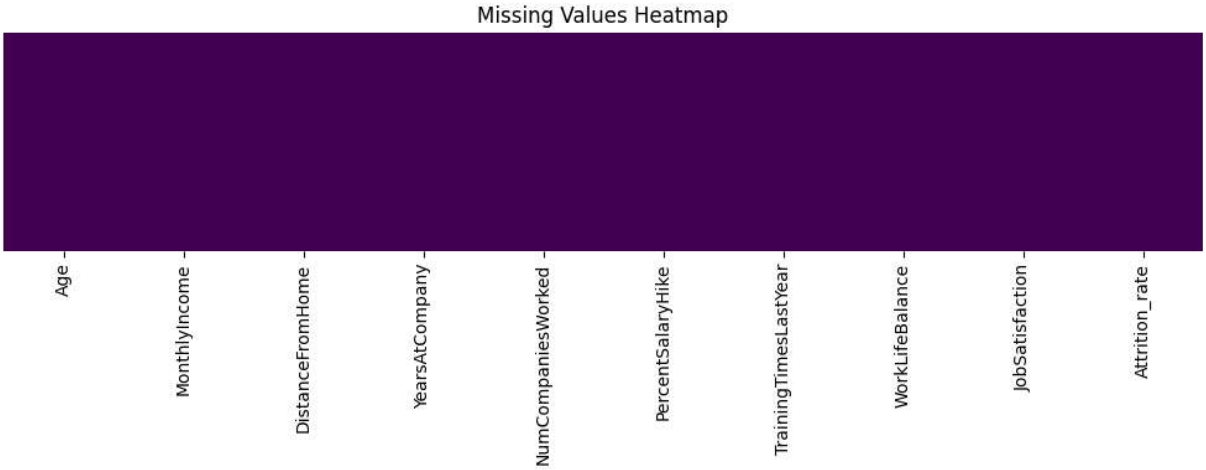
In [54]:
```
class DataIngestor:
    def __init__(self, path):
        self.path = path
```

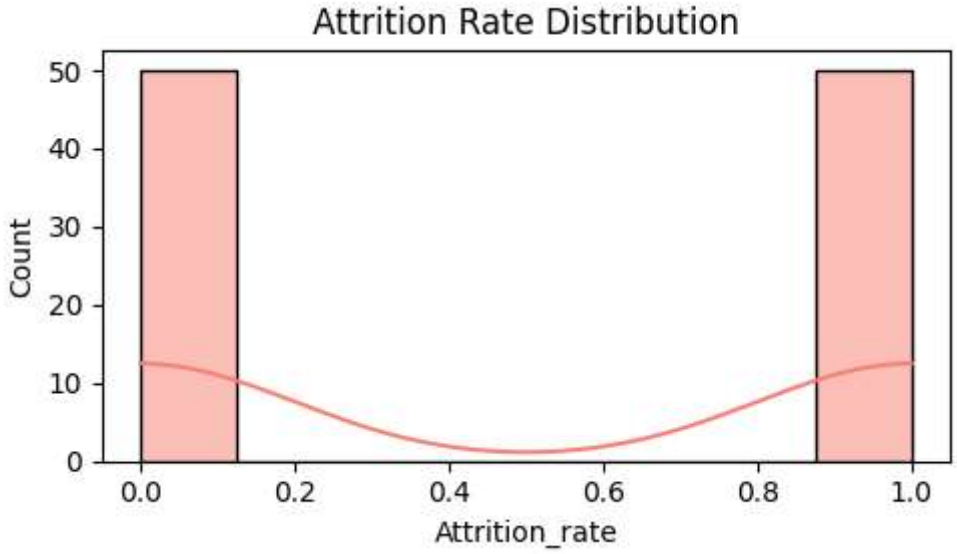
```
    def load_data(self):
        print("\nLoading data...")
        try:
            df = pd.read_csv(self.path)

            print("\n Data Preview:\n", df.head())
            print("\nData Info:")
            print(df.info())
            print("\nData Description:\n", df.describe())


            plt.figure(figsize=(10, 4))
            sns.heatmap(df.isnull(), cbar=False, cmap='viridis', yticklabels=False)
            plt.title("Missing Values Heatmap")
            plt.tight_layout()
            plt.show()


            cat_cols = df.select_dtypes(include='object').columns
            for col in cat_cols:
                plt.figure(figsize=(6, 3))
                sns.countplot(data=df, x=col, palette='Set2')
                plt.title(f"Distribution of {col}")
                plt.xticks(rotation=30)
                plt.tight_layout()
                plt.show()


            if 'Attrition_rate' in df.columns:
                plt.figure(figsize=(5, 3))
                sns.histplot(df['Attrition_rate'], kde=True, color='salmon')
                plt.title("Attrition Rate Distribution")
                plt.tight_layout()
                plt.show()
            return df

        except FileNotFoundError:
            print(f"File not found at path: {self.path}")
            return None


ingestor = DataIngestor("small_employee_attrition.csv")
df = ingestor.load_data()
```

```
Loading data...

 Data Preview:
    Age  MonthlyIncome  DistanceFromHome  YearsAtCompany  NumCompaniesWorked  \
0   50           8056                 3              31                   9
1   36          17948                19              22                   4
2   29          11110                16              32                   6
3   42          16773                16               2                   3
4   40           3502                 3              17                   0

   PercentSalaryHike  TrainingTimesLastYear  WorkLifeBalance  JobSatisfaction  \
0                 14                      0                3                2
1                 12                      0                1                2
2                 21                      5                2                1
3                 18                      4                3                2
4                 13                      5                1                1

   Attrition_rate
0               0
1               0
2               1
3               0
4               0

Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    100 non-null    int64
 1   MonthlyIncome          100 non-null    int64
 2   DistanceFromHome       100 non-null    int64
 3   YearsAtCompany         100 non-null    int64
 4   NumCompaniesWorked     100 non-null    int64
 5   PercentSalaryHike      100 non-null    int64
 6   TrainingTimesLastYear  100 non-null    int64
 7   WorkLifeBalance        100 non-null    int64
 8   JobSatisfaction        100 non-null    int64
 9   Attrition_rate         100 non-null    int64
dtypes: int64(10)
memory usage: 7.9 KB
None

Data Description:
               Age  MonthlyIncome  DistanceFromHome  YearsAtCompany  \
count  100.000000     100.000000        100.000000      100.000000
mean    40.060000   11441.800000         14.900000       20.090000
std     10.688255    4906.197767          9.083785       11.541839
min     22.000000    3197.000000          1.000000        0.000000
25%     30.000000    7649.250000          6.000000       11.000000
50%     41.500000   11137.500000         16.000000       22.000000
75%     48.000000   15685.750000         22.000000       29.000000
max     59.000000   19646.000000         29.000000       39.000000

       NumCompaniesWorked  PercentSalaryHike  TrainingTimesLastYear  \
```

count	100.000000	100.000000	100.00000
mean	4.340000	17.520000	2.46000
std	3.108217	4.186269	1.76051
min	0.000000	10.000000	0.00000
25%	2.000000	14.000000	1.00000
50%	4.000000	18.000000	3.00000
75%	7.000000	21.000000	4.00000
max	9.000000	24.000000	5.00000

	WorkLifeBalance	JobSatisfaction	Attrition_rate
count	100.000000	100.000000	100.000000
mean	2.330000	2.460000	0.500000
std	1.064154	1.077033	0.502519
min	1.000000	1.000000	0.000000
25%	1.000000	2.000000	0.000000
50%	2.000000	2.000000	0.500000
75%	3.000000	3.000000	1.000000
max	4.000000	4.000000	1.000000





STEP 2: DATA PREPROCESSING

This step involves preparing the raw dataset for modeling by cleaning and transforming the data.

Common preprocessing tasks include:

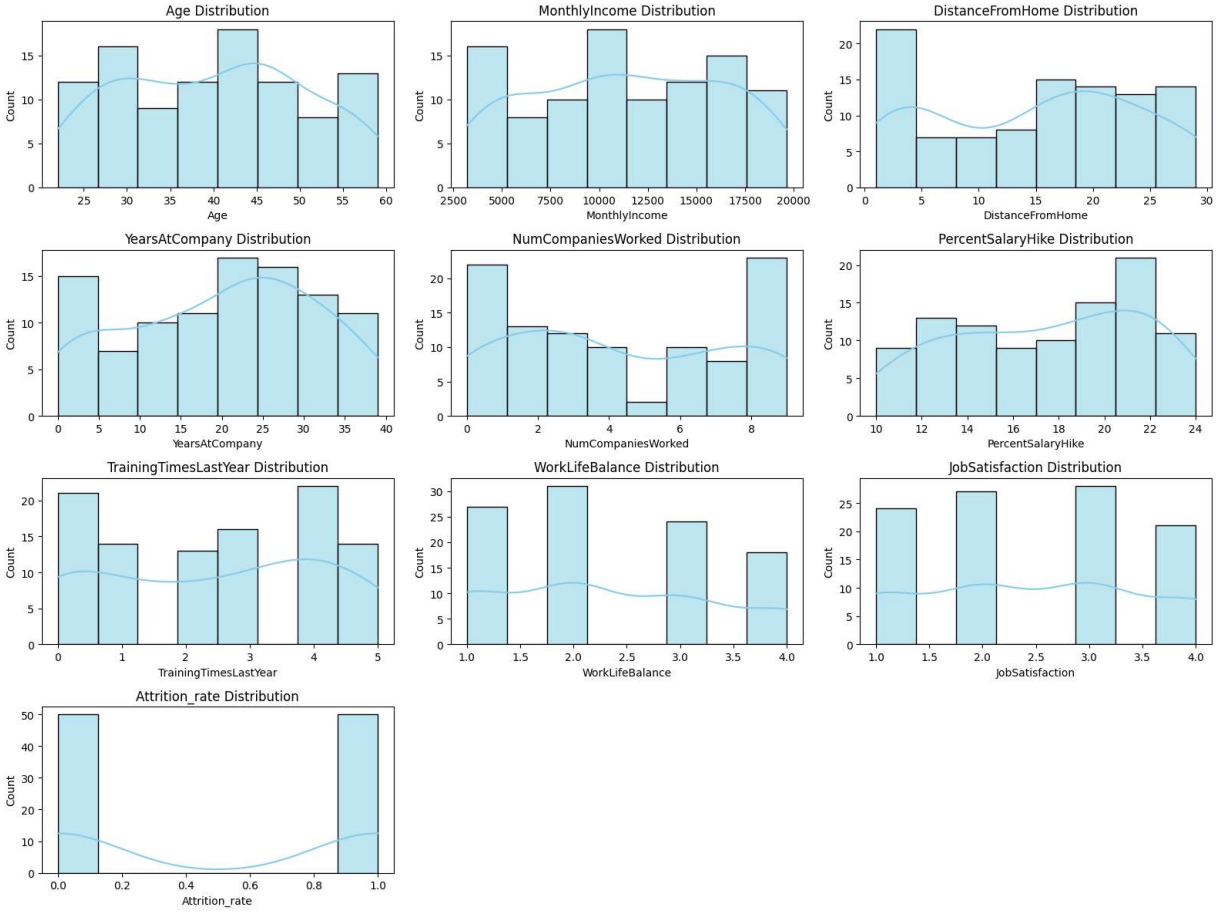
- **Handling missing values**: Filling them with mean/median/mode, or dropping rows/columns.
- **Encoding categorical variables**: Converting text labels into numeric format using techniques like label encoding or one-hot encoding.
- **Normalizing or scaling** features: Ensuring numerical features are on the same scale (important for many ML algorithms).
- **Removing duplicates or irrelevant features** that don't contribute to the prediction task.
- **Feature engineering**: Creating new relevant features from existing data to improve model performance.

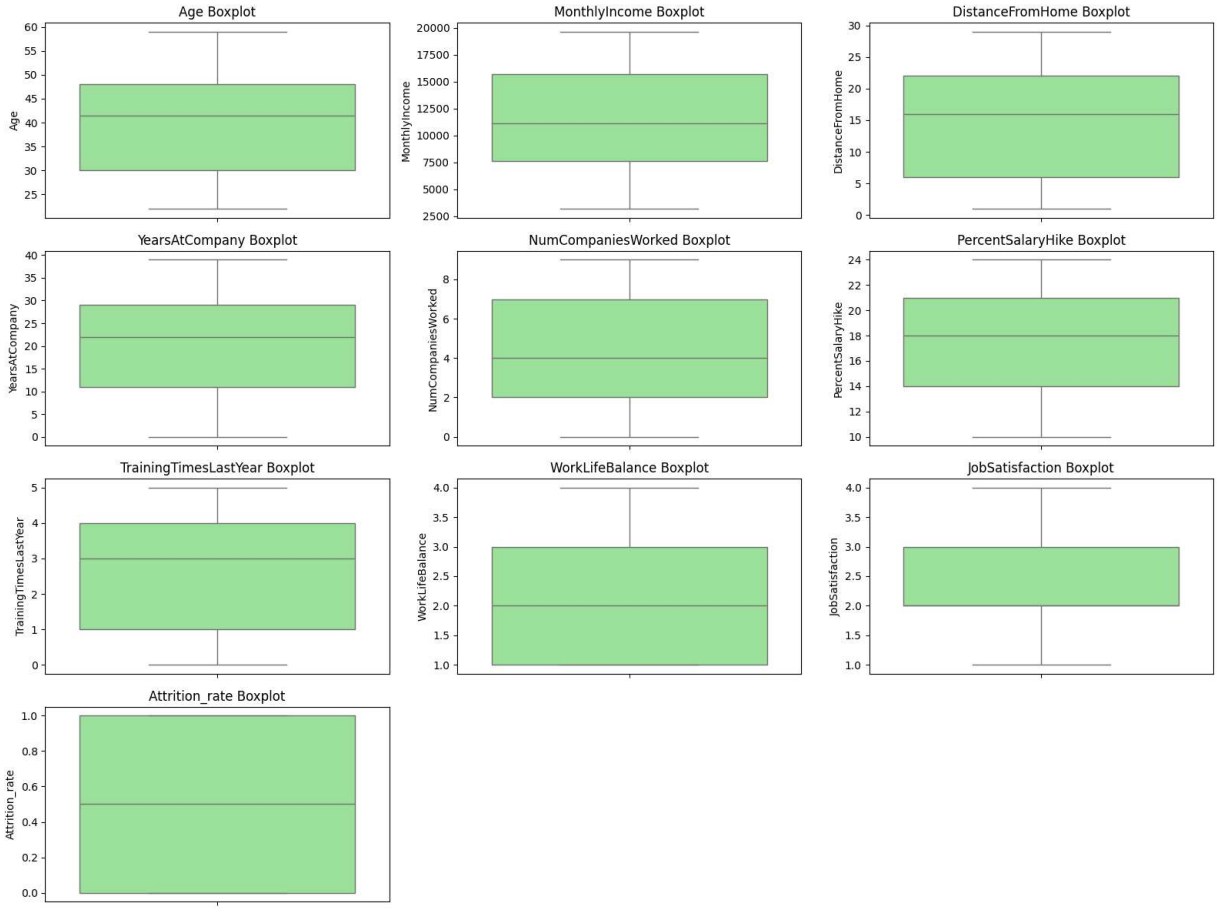
Effective preprocessing improves model accuracy, stability, and generalization to new data.

In [56]:
```
class DataPreprocessor:
    def __init__(self, dataframe: pd.DataFrame):
        self.df = dataframe.copy()

    def preprocess(self):
        print("\nPreprocessing data...")
        before_rows = self.df.shape[0]
        self.df.dropna(inplace=True)
        after_rows = self.df.shape[0]
        print(f"Dropped {before_rows - after_rows} rows with missing values.")
        return self.df

    def univariate_analysis(self):
        print("\nUnivariate Analysis:")

        numeric_cols = self.df.select_dtypes(include=np.number).columns.tolist()
        cat_cols = self.df.select_dtypes(include='object').columns.tolist()


        plt.figure(figsize=(16, 12))
        for i, col in enumerate(numeric_cols):
            plt.subplot(4, 3, i + 1)
            sns.histplot(self.df[col], kde=True, color="skyblue")
            plt.title(f"{col} Distribution")
        plt.tight_layout()
        plt.show()


        plt.figure(figsize=(16, 12))
        for i, col in enumerate(numeric_cols):
            plt.subplot(4, 3, i + 1)
            sns.boxplot(y=self.df[col], color="lightgreen")
            plt.title(f"{col} Boxplot")
        plt.tight_layout()
```

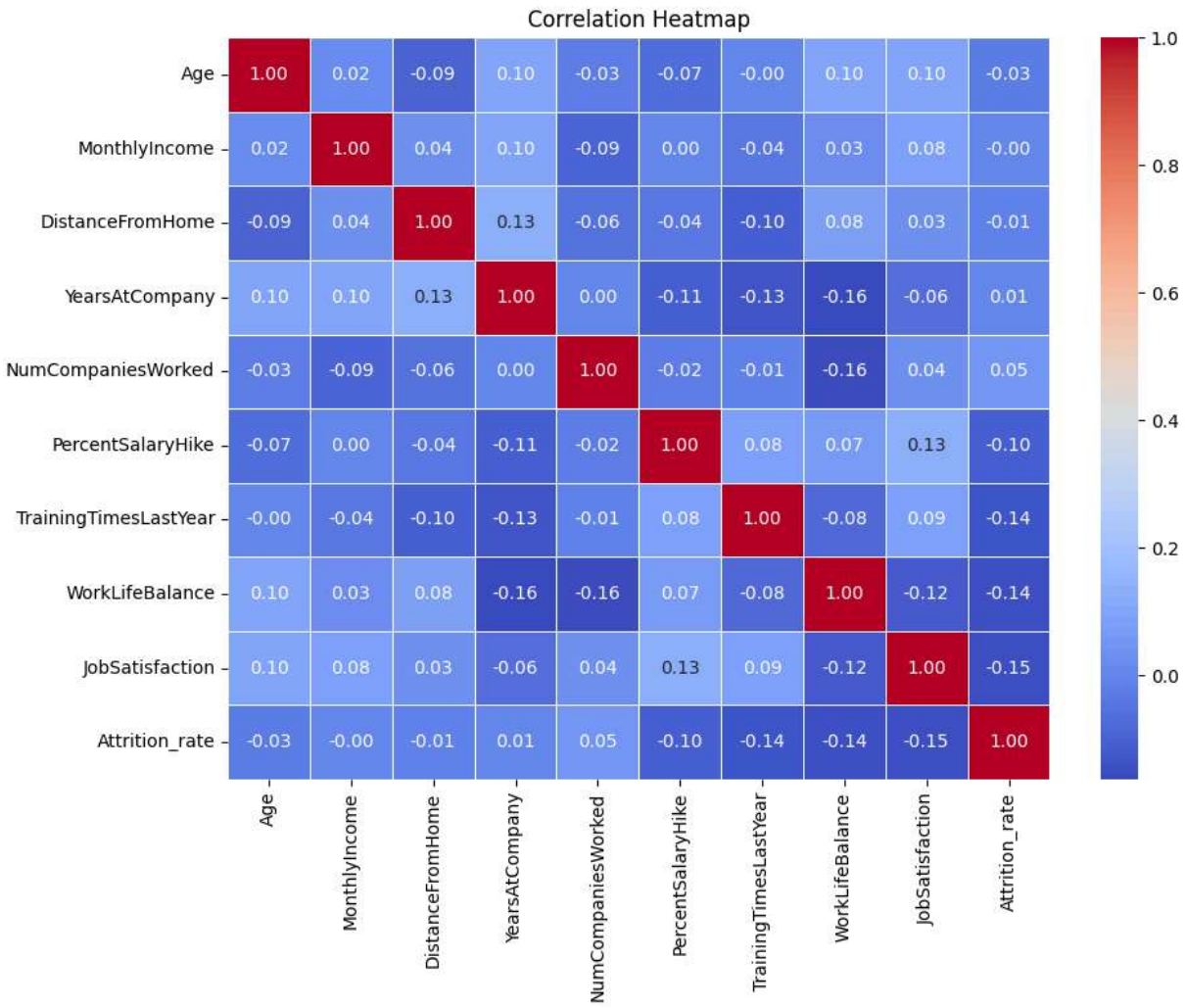
```
        plt.show()


        for col in cat_cols:
            plt.figure(figsize=(6, 4))
            sns.countplot(x=self.df[col], palette="pastel")
            plt.title(f"{col} Count")
            plt.xticks(rotation=45)
            plt.tight_layout()
            plt.show()

    def bivariate_analysis(self):
        print("\nBivariate Analysis:")

        numeric_cols = self.df.select_dtypes(include=np.number).columns.tolist()


        plt.figure(figsize=(10, 8))
        sns.heatmap(self.df[numeric_cols].corr(), annot=True, cmap="coolwarm", fmt=
        plt.title("Correlation Heatmap")
        plt.tight_layout()
        plt.show()


        if len(numeric_cols) >= 2:
            top_corr = self.df[numeric_cols].corr().abs().unstack().sort_values(asc
            top_pairs = [(a, b) for a, b in top_corr.index if a != b][:4]
            top_features = list(set([item for pair in top_pairs for item in pair]))

            print(f"\nPairplot of top correlated features: {top_features}")
            sns.pairplot(self.df[top_features])
            plt.show()


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

processor = DataPreprocessor(df)

clean_df = processor.preprocess()

processor.univariate_analysis()
processor.bivariate_analysis()
```

```
Preprocessing data...
Dropped 0 rows with missing values.

Univariate Analysis:
```

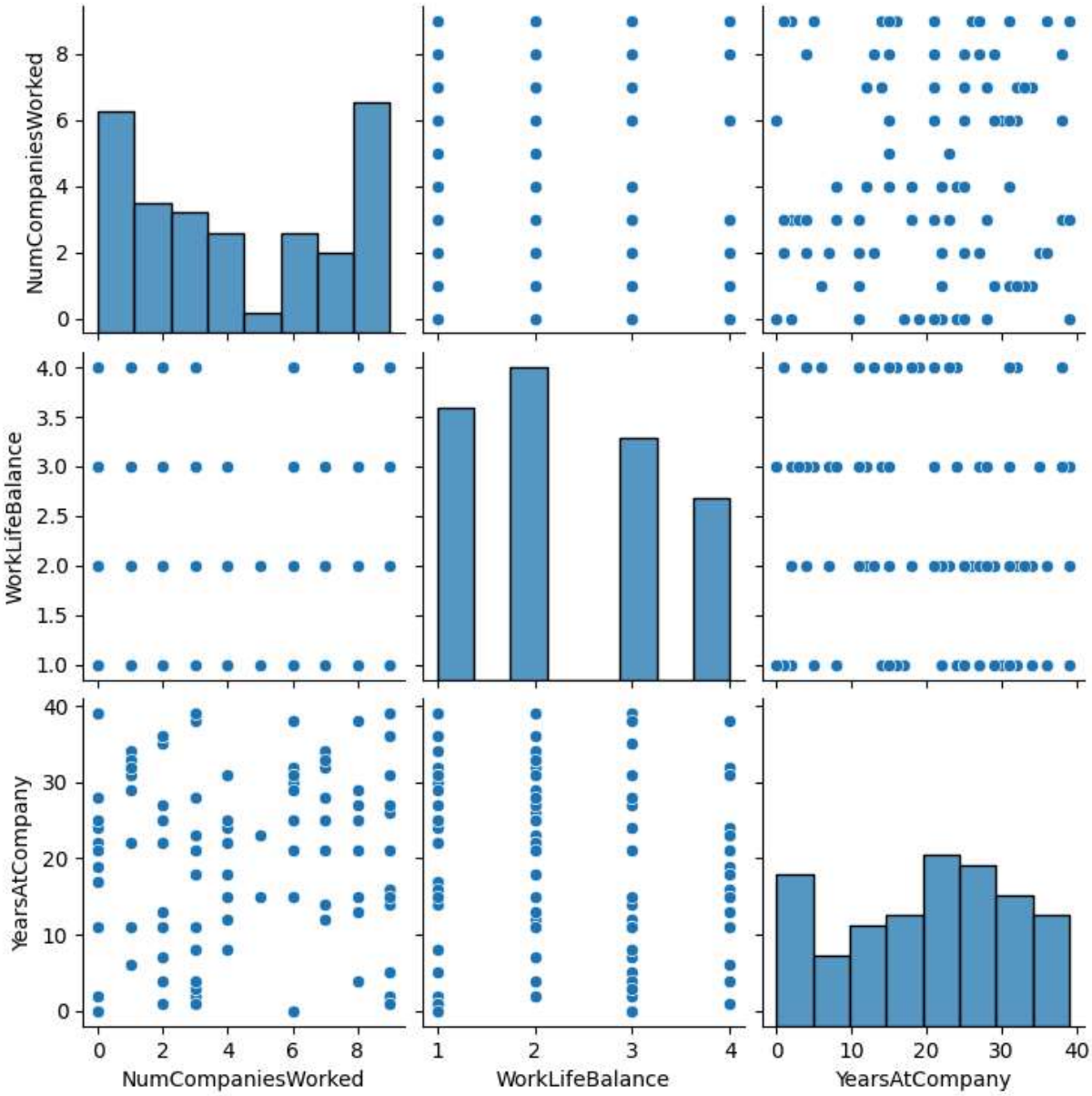




Bivariate Analysis:




```
Pairplot of top correlated features: ['NumCompaniesWorked', 'WorkLifeBalance', 'Year
sAtCompany']
```



STEP 3: DATA SPLITTING

In this step, we divide the dataset into separate subsets for training and testing the model.

Typically:

- **Training set**: Used to train the machine learning model.
- **Testing set**: Used to evaluate the model's performance on unseen data.

We use functions like `train_test_split()` from `sklearn.model_selection` to perform the split. This helps ensure:

- The model can generalize to new data.
- We avoid overfitting, where the model performs well on training data but poorly on real-world or testing data.

A common split ratio is **80% for training** and **20% for testing**, but this may vary depending on the dataset size and modeling goals.

In [57]:
```
class DataSplitter:
    def __init__(self, df, target_column):
        self.df = df
        self.target_column = target_column

    def split(self):
        print("\nSplitting data into train and test sets...")
        if self.target_column not in self.df.columns:
            raise ValueError(f"Target column '{self.target_column}' not found in Da

        X = self.df.drop(self.target_column, axis=1)
        y = self.df[self.target_column]

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

        print(f"Split complete: {X_train.shape[0]} train rows, {X_test.shape[0]} te
        return X_train, X_test, y_train, y_test


splitter = DataSplitter(df, target_column="Attrition_rate")
X_train, X_test, y_train, y_test = splitter.split()
```

```
Splitting data into train and test sets...
Split complete: 80 train rows, 20 test rows.
```

STEP 4: MODEL TRAINING

In this step, we build and train a machine learning model using the training dataset.

Key steps include:

- **Selecting an appropriate algorithm** based on the problem type (e.g., classification or regression). Examples: Decision Tree, Random Forest, Logistic Regression, etc.
- **Fitting the model** to the training data using `.fit(X_train, y_train)`, where the model learns patterns and relationships in the data.
- **Monitoring for overfitting** by ensuring the model doesn't memorize the training data and can generalize well.

After training, the model is ready for evaluation on the test set to assess its performance on unseen data.

In [58]:
```
class ModelTrainer:
    def __init__(self):
        self.model = SVC(kernel='linear')

    def train(self, X_train, y_train):
```

```
        print("\nTraining model (SVM)...")
        self.model.fit(X_train, y_train)
        print("Training complete.")
        return self.model

    def evaluate(self, X_test, y_test):
        print("\nModel Evaluation:")
        predictions = self.model.predict(X_test)

        print("\nConfusion Matrix:")
        print(confusion_matrix(y_test, predictions))

        print("\nClassification Report:")
        print(classification_report(y_test, predictions))


trainer = ModelTrainer()
model = trainer.train(X_train, y_train)
trainer.evaluate(X_test, y_test)
```

```
Training model (SVM)...
Training complete.

Model Evaluation:

Confusion Matrix:
[[5 5]
 [5 5]]

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.50      0.50        10
           1       0.50      0.50      0.50        10

    accuracy                           0.50        20
   macro avg       0.50      0.50      0.50        20
weighted avg       0.50      0.50      0.50        20
```

STEP 5: SAVING MODEL

In this step, we save the trained machine learning model to a file so it can be reused later without retraining.

Common tools for model saving:

- **`joblib`** or **`pickle`**: Python libraries used to serialize (save) and deserialize (load) machine learning models.

Typical saving process:

- Save the model using `joblib.dump(model, 'model_filename.pkl')`.

- Load the model later using `joblib.load('model_filename.pkl')`.

This is useful when deploying the model into a real-world application or sharing it with others without needing to retrain.

In [59]:
```
class ModelSaver:
    def __init__(self, model, filename='small_employee_attrition.pkl'):
        self.model = model
        self.filename = filename

    def save(self):
        print("\nSaving model with Pickle...")
        try:
            with open(self.filename, 'wb') as f:
                pickle.dump(self.model, f)
            print(f"Model saved successfully as '{self.filename}'")
        except Exception as e:
            print(f"Error saving model: {e}")


saver = ModelSaver(model, filename='small_employee_attrition.pkl')
saver.save()
```

```
Saving model with Pickle...
Model saved successfully as 'small_employee_attrition.pkl'
```

EXECUTION

This section runs the full workflow — from data preprocessing to model training and saving — in sequence.

The purpose is to:

- Ensure all steps in the pipeline execute correctly.
- Validate that the model is trained, evaluated, and saved without errors.
- Simulate a real-use scenario where the entire process is automated or triggered in a single run (e.g., in a script or application backend).

It helps verify that the complete solution works as expected end-to-end.

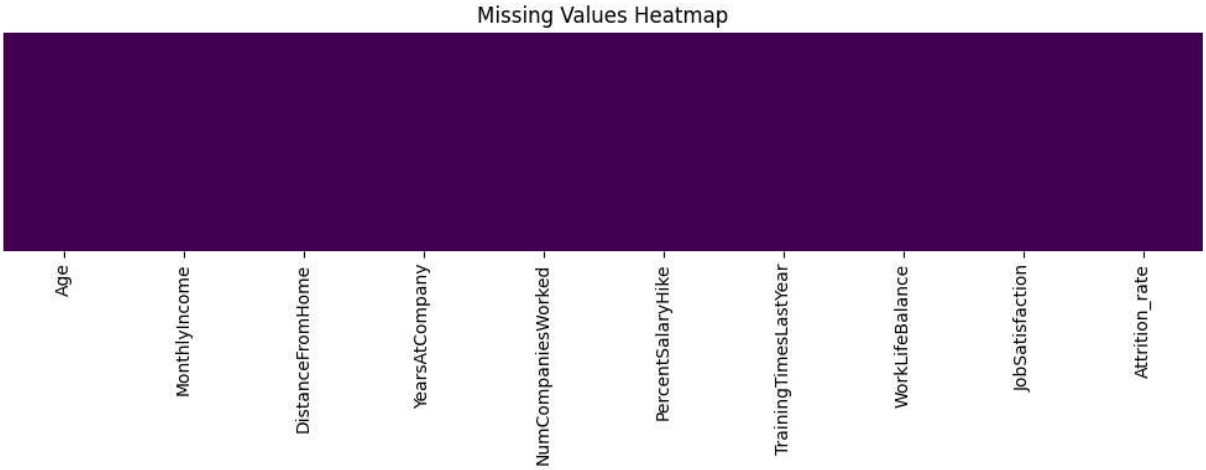
In [60]:
```
if __name__ == "__main__":
    dataset_path = "small_employee_attrition.csv"


    loader = DataIngestor(dataset_path)
    df = loader.load_data()


    processor = DataPreprocessor(df)
    df_clean = processor.preprocess()
    processor.univariate_analysis()
```

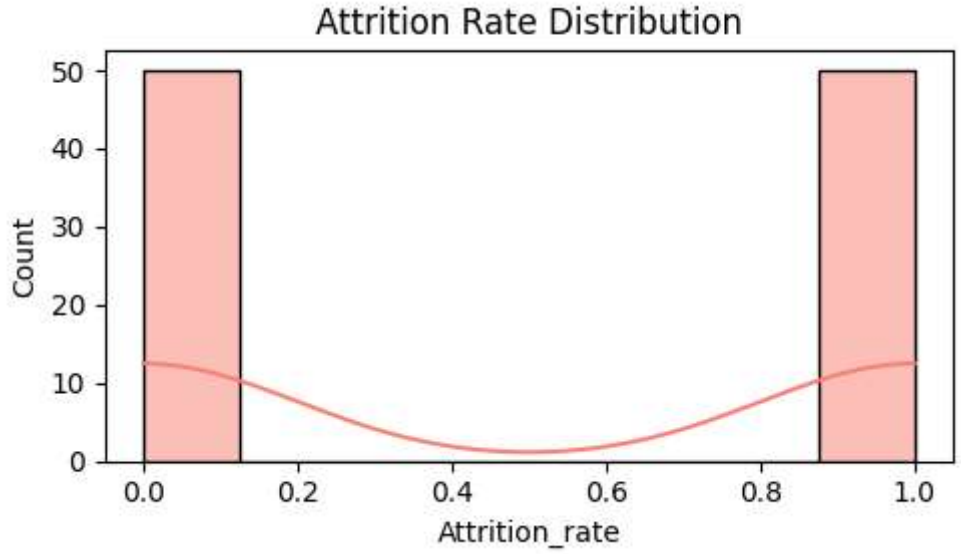
```
    processor.bivariate_analysis()


    splitter = DataSplitter(df_clean, target_column='Attrition_rate')
    X_train, X_test, y_train, y_test = splitter.split()


    trainer = ModelTrainer()
    model = trainer.train(X_train, y_train)


    saver = ModelSaver(model)
    saver.save()
```

```
Loading data...

 Data Preview:
    Age  MonthlyIncome  DistanceFromHome  YearsAtCompany  NumCompaniesWorked  \
0   50           8056                 3              31                   9
1   36          17948                19              22                   4
2   29          11110                16              32                   6
3   42          16773                16               2                   3
4   40           3502                 3              17                   0

   PercentSalaryHike  TrainingTimesLastYear  WorkLifeBalance  JobSatisfaction  \
0                 14                      0                3                2
1                 12                      0                1                2
2                 21                      5                2                1
3                 18                      4                3                2
4                 13                      5                1                1

   Attrition_rate
0               0
1               0
2               1
3               0
4               0

Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    100 non-null    int64
 1   MonthlyIncome          100 non-null    int64
 2   DistanceFromHome       100 non-null    int64
 3   YearsAtCompany         100 non-null    int64
 4   NumCompaniesWorked     100 non-null    int64
 5   PercentSalaryHike      100 non-null    int64
 6   TrainingTimesLastYear  100 non-null    int64
 7   WorkLifeBalance        100 non-null    int64
 8   JobSatisfaction        100 non-null    int64
 9   Attrition_rate         100 non-null    int64
dtypes: int64(10)
memory usage: 7.9 KB
None

Data Description:
               Age  MonthlyIncome  DistanceFromHome  YearsAtCompany  \
count  100.000000     100.000000        100.000000      100.000000
mean    40.060000   11441.800000         14.900000       20.090000
std     10.688255    4906.197767          9.083785       11.541839
min     22.000000    3197.000000          1.000000        0.000000
25%     30.000000    7649.250000          6.000000       11.000000
50%     41.500000   11137.500000         16.000000       22.000000
75%     48.000000   15685.750000         22.000000       29.000000
max     59.000000   19646.000000         29.000000       39.000000

       NumCompaniesWorked  PercentSalaryHike  TrainingTimesLastYear  \
```

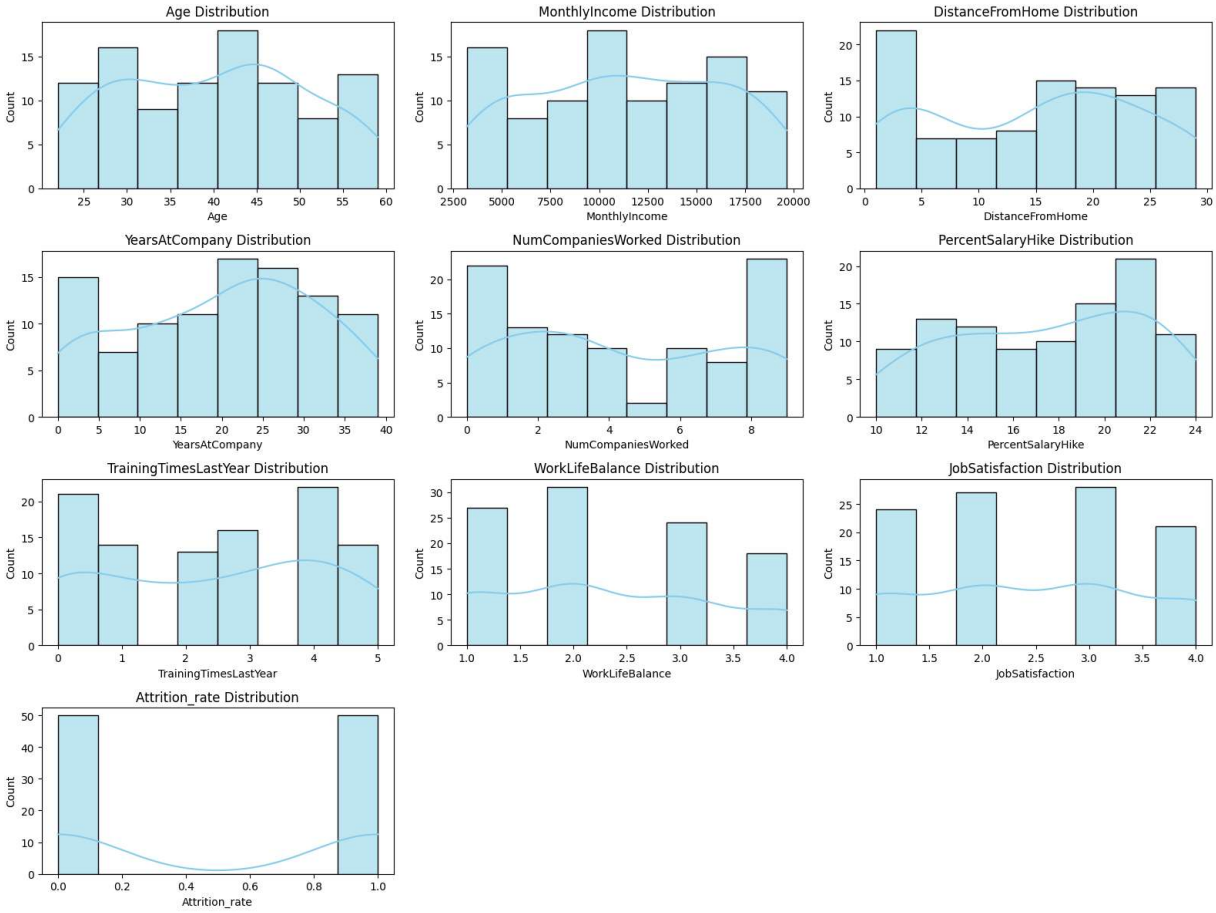
count	100.000000	100.000000	100.00000
mean	4.340000	17.520000	2.46000
std	3.108217	4.186269	1.76051
min	0.000000	10.000000	0.00000
25%	2.000000	14.000000	1.00000
50%	4.000000	18.000000	3.00000
75%	7.000000	21.000000	4.00000
max	9.000000	24.000000	5.00000

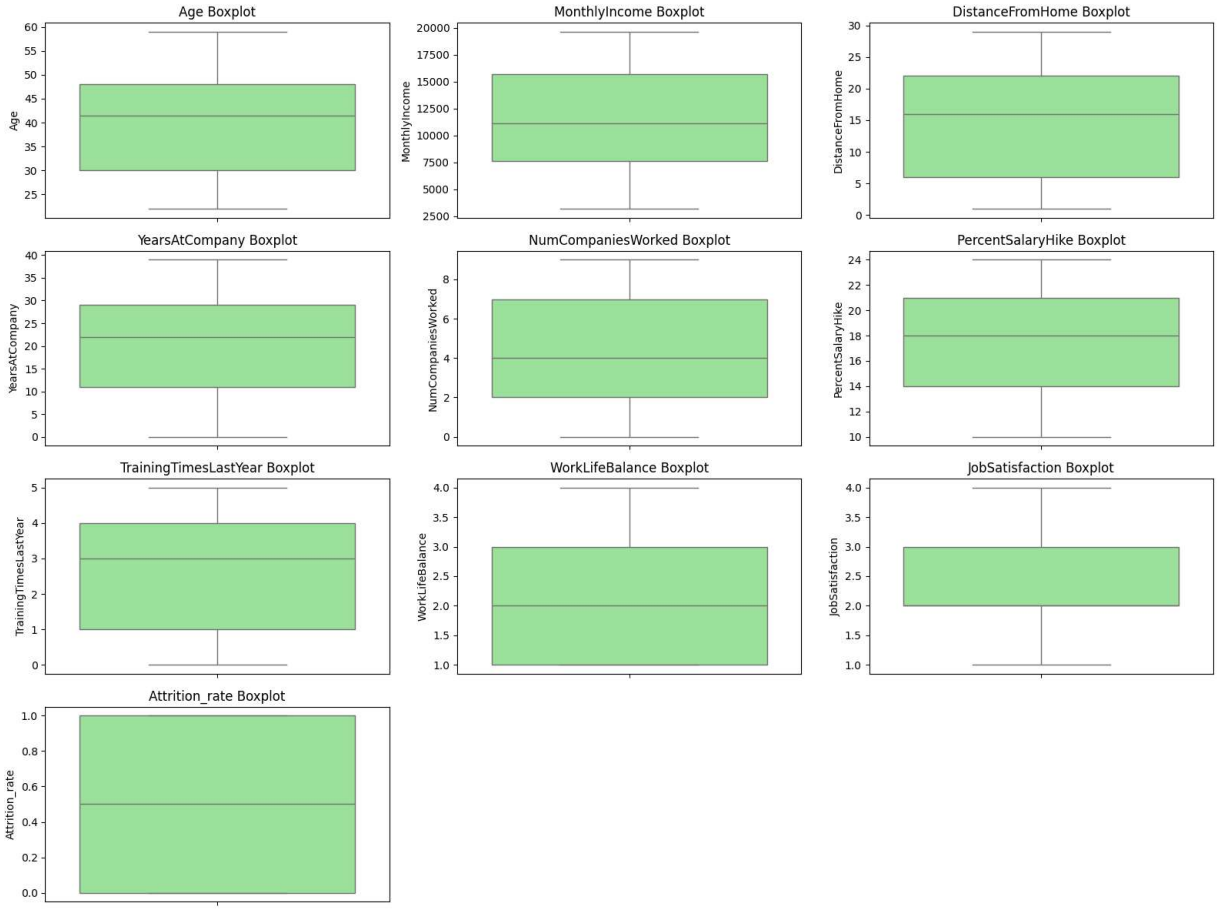
	WorkLifeBalance	JobSatisfaction	Attrition_rate
count	100.000000	100.000000	100.000000
mean	2.330000	2.460000	0.500000
std	1.064154	1.077033	0.502519
min	1.000000	1.000000	0.000000
25%	1.000000	2.000000	0.000000
50%	2.000000	2.000000	0.500000
75%	3.000000	3.000000	1.000000
max	4.000000	4.000000	1.000000



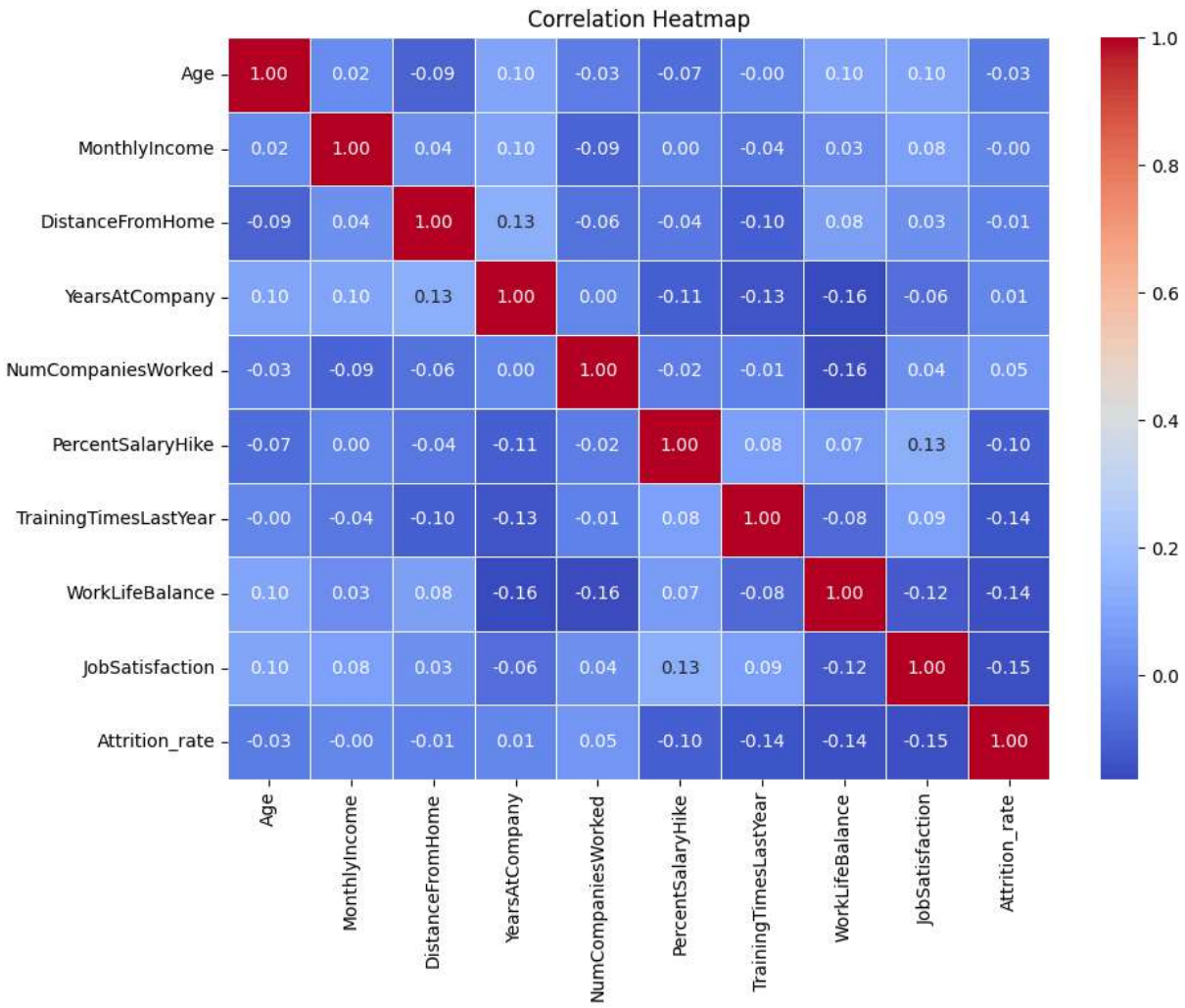



```
Preprocessing data...
Dropped 0 rows with missing values.

Univariate Analysis:
```

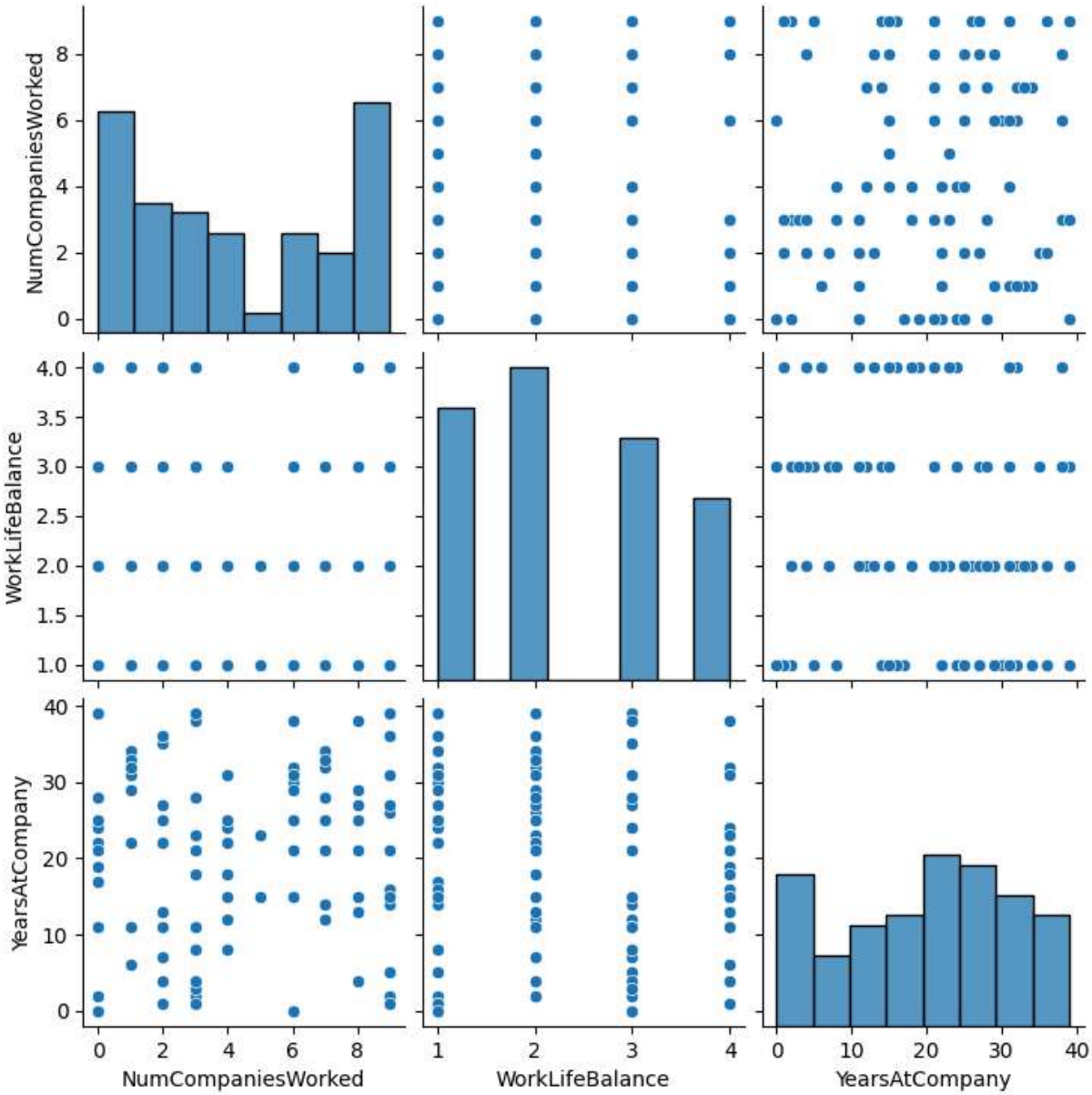




Bivariate Analysis:




```
Pairplot of top correlated features: ['NumCompaniesWorked', 'WorkLifeBalance', 'Year
sAtCompany']
```




```
Splitting data into train and test sets...
Split complete: 80 train rows, 20 test rows.

Training model (SVM)...
Training complete.

Saving model with Pickle...
Model saved successfully as 'small_employee_attrition.pkl'
```