

Hochschule Fresenius University of Applied Sciences  
Faculty of Economics & Media  
International Business School  
International Business Management  
Cologne Campus

Working Title

**"Predictive Modeling and Feature Importance Analysis for Employee  
Attrition: A Machine Learning Approach."**

Kimiya Hajiyousefi  
Student ID No.: 400323288  
3. Semester

Module: Technical Application and Data Management

Lecturer: Mrs. Lampl

Due Date: 24 July 2023

## Abstract

Predicting and understanding the key factors influencing employee attrition is of critical importance in today's business landscape. Employee turnover can lead to significant costs related to hiring and training replacements, loss of knowledge and experience, and potential disruption to productivity. Thus, being able to predict attrition and understand the influencing factors can offer valuable insights for human resources management and overall business strategy. In this study, we pursued two interrelated objectives. First, we developed a machine learning model to predict employee attrition based on a dataset of various employee characteristics. This dataset, containing information on 1029 employees, included variables such as age, job role, marital status, and distance from home among others. The Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology guided the project, which involved steps including business understanding, data understanding, data preparation, modeling, evaluation, and deployment. We trained and evaluated multiple machine learning models, with the Logistic Regression model yielding the best performance, boasting an Area Under the Receiver Operating Characteristic Curve (AUC-ROC) of approximately 0.863 on the validation set. This model serves as a valuable tool for HR departments to identify employees at risk of leaving, enabling proactive interventions to improve retention. The second goal of this study was to understand the key factors influencing attrition. By analyzing feature importance from our model, we identified the most significant predictors of employee attrition. Future work can extend this study by addressing class imbalance, engineering new features, trying different models, and collecting more data to further refine the model's accuracy and the comprehensiveness of our insights into employee attrition.

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
<b>2. Dataset.....</b>	<b>1</b>
<b>3. Importing the Dataset.....</b>	<b>2</b>
<b>4. Goals of the Machine Learning Project .....</b>	<b>4</b>
<b>5. CRISP-DM Model.....</b>	<b>4</b>
<b>5.1. Goal 1 .....</b>	<b>4</b>
<b>5.1.1. Business Understanding .....</b>	<b>4</b>
<b>5.1.2. Data Understanding.....</b>	<b>5</b>
<b>5.1.3. Data Preparation.....</b>	<b>12</b>
<b>5.1.4. Modeling .....</b>	<b>17</b>
<b>5.1.4.1. Logistic Regression .....</b>	<b>19</b>
<b>5.1.4.2. Random Forest .....</b>	<b>21</b>
<b>5.1.5. Evaluation .....</b>	<b>22</b>
<b>5.1.6. Deployment.....</b>	<b>25</b>
<b>6. Discussion and Results.....</b>	<b>26</b>
<b>7. Conclusion .....</b>	<b>27</b>
<b>5.2. Goal 2 .....</b>	<b>28</b>
<b>5.2.1. Business Understanding .....</b>	<b>28</b>
<b>5.2.2. Data Understanding.....</b>	<b>28</b>
<b>5.2.3. Data Preparation.....</b>	<b>30</b>
<b>5.2.4. Modeling- Evaluation- Fine-Tuning .....</b>	<b>37</b>
<b>6. Deployment.....</b>	<b>44</b>
<b>7. Visualisation .....</b>	<b>44</b>
<b>8. Discussion.....</b>	<b>46</b>
<b>9. Conclusion .....</b>	<b>46</b>
<b>10. Reference.....</b>	<b>47</b>

## **1. Introduction**

Employee attrition, or turnover, is a significant challenge for many organizations. Losing valuable employees can lead to decreased productivity, increased costs associated with hiring and training new employees, and a potential loss of knowledge and skills. Therefore, being able to predict which employees are likely to leave the company can be very beneficial, as it allows for proactive measures to increase employee retention.

In this project, I aimed to Firstly, develop a machine learning model that can predict employee attrition based on various characteristics such as age, job role, marital status, distance from home, and others. The dataset we used contained information about 1029 employees from a company, including whether or not they left the company ('Attrition'). Secondly, I want to understand the factors influencing attrition. Because it can provide valuable insights for human resources management and overall business strategy. In this second goal, I was given a dataset of employee characteristics. The approach I employed involved training a machine learning model on the data, using the model to identify the most important features influencing predictions, and evaluating the model's performance.

I followed the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology for both, which includes the following steps: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment.

The results of this project could be used by the HR department of the company to identify employees who are likely to leave and take proactive measures to increase their retention. While the machine learning model provides a valuable tool for predicting attrition, it should be used in conjunction with other tools and expert judgement in decision-making. Future work could involve addressing the imbalance in the 'Attrition' variable, engineering new features, trying different models, collecting more data, and applying more detailed model interpretation techniques.

## **2. Dataset**

The dataset used in this project contains information about employees from a company. It consists of 1029 instances (employees) and 35 variables. These variables include various

characteristics of the employees, as well as the target variable 'Attrition' which indicates whether the employee left the company.

Here's a brief description of some of the variables in the dataset:

1. Age: The age of the employee.
2. BusinessTravel: The frequency of business travel for the employee ('Travel\_Rarely', 'Travel\_Frequently', 'Non-Travel').
3. DailyRate: The daily rate of the employee.
4. Department: The department the employee works in ('Sales', 'Research & Development', 'Human Resources').
5. DistanceFromHome: The distance from work to home.
6. Education: The level of education of the employee (1: 'Below College', 2: 'College', 3: 'Bachelor', 4: 'Master', 5: 'Doctor').
7. EducationField: The field of education of the employee ('Life Sciences', 'Other', 'Medical', 'Marketing', 'Technical Degree', 'Human Resources').
8. JobRole: The job role of the employee.
9. MaritalStatus: The marital status of the employee ('Single', 'Married', 'Divorced').
10. MonthlyIncome: The monthly income of the employee.
11. NumCompaniesWorked: The number of companies the employee has worked for.
12. PercentSalaryHike: The percent salary hike of the employee.
13. PerformanceRating: The performance rating of the employee (1: 'Low', 2: 'Good', 3: 'Excellent', 4: 'Outstanding').
14. YearsAtCompany: The number of years the employee has worked at the company.
15. Attrition: Whether the employee left the company ('Yes', 'No')

The dataset contained some missing values, which were imputed during the Data Preparation phase. Also, the target variable 'Attrition' was imbalanced, with more instances of 'No' than 'Yes'. This was an important consideration during the Modeling and Evaluation phases.

### 3. Importing the Dataset

```
import pandas as pd
# Load the training dataset
```

```
train_data = pd.read_csv("/mnt/data/employee_attrition_train.csv")
```

```
# Show the first few rows of the dataset
```

```
train_data.head()
```

RESULT

	Age	Attrition	BusinessTravel	DailyRate	Department	\
0	50.0	No	Travel_Rarely	1126.0	Research & Development	
1	36.0	No	Travel_Rarely	216.0	Research & Development	
2	21.0	Yes	Travel_Rarely	337.0	Sales	
3	50.0	No	Travel_Frequently	1246.0	Human Resources	
4	52.0	No	Travel_Rarely	994.0	Research & Development	

	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	\
0	1.0	2	Medical	1	997	
1	6.0	2	Medical	1	178	
2	7.0	1	Marketing	1	1780	
3	NaN	3	Medical	1	644	
4	7.0	4	Life Sciences	1	1118	

	...	RelationshipSatisfaction	StandardHours	StockOptionLevel	\
0	...		3	80	1
1	...		4	80	2
2	...		2	80	0
3	...		3	80	1
4	...		4	80	0

	TotalWorkingYears	TrainingTimesLastYear	WorkLifeBalance	YearsAtCompany	\
0	32		1	2	5
1	7		0	3	3
2	1		3	3	1
3	32		2	3	32
4	18		4	3	8

	YearsInCurrentRole	YearsSinceLastPromotion	YearsWithCurrManager	
0	4		1	3
1	2		0	1
2	0		1	0
3	5		10	7
4	6		4	0

[5 rows x 35 columns]

The dataset consists of various information about employees, such as their age, business travel frequency, daily rate, department, distance from home, education field, job role, job satisfaction, monthly income, and several other features. It seems to be a dataset from a company aiming to understand the attrition of its employees.

The target variable appears to be 'Attrition', which indicates whether an employee left the company ('Yes') or stayed ('No').

## 4. Goals of the Machine Learning Project

1. Predict employee attrition in a company based on various employee characteristics such as age, daily rate, department, distance from home, job role, education, and others.
2. Understand the key factors influencing attrition.

## 5. CRISP-DM Model

### 5.1. Goal 1

- Predict employee attrition in a company based on various employee characteristics such as age, daily rate, department, distance from home, job role, education, and others.

**Business Understanding:** Defining the problem and the approach. In this case, the problem is predicting employee attrition. The approach would be to use machine learning algorithms to create a predictive model.

**Data Understanding:** This includes understanding the meaning of all the columns, checking the data types, and performing descriptive statistics to get a sense of the data distribution.

**Data Preparation:** Cleaning the data and preparing it for machine learning. This may involve dealing with missing values, encoding categorical variables, normalizing numerical variables, etc.

**Modeling:** Training machine learning models on the dataset. I might want to try different algorithms (like decision trees, random forest, logistic regression, support vector machines, etc.) and see which one works best.

**Evaluation:** Evaluating the performance of your models using appropriate metrics (like accuracy, precision, recall, F1 score, ROC AUC, etc.). This will help to understand how well the model is performing and which model is the best.

**Deployment:** This step involves deploying the model into a production environment where it can take in new data and make predictions.

#### 5.1.1. Business Understanding

the Business Understanding phase is the first stage of the CRISP-DM process and is absolutely critical to the success of the project. The purpose of the Business Understanding stage is to clarify the goals of the project and determine what I want to achieve. This sets the foundation for

the following stages of the project. In this phase, I need to understand the project's objectives and requirements from a business perspective, and then convert this knowledge into a data mining problem definition and a preliminary plan.

According to dataset and project goal, the key steps in the Business Understanding phase is:

**Define the Objective:** The objective of this project is to predict employee attrition based on various employee characteristics. This is important as high employee attrition can be costly for companies, and understanding which factors influence attrition can help the company to implement policies or strategies to reduce it.

**Assess the Current Situation:** I want to understand Company current approach to managing attrition, if they have one. Do they have existing data or models? What are their current retention strategies? Since this is a project for a class, we can skip this part.

**Define the Data Mining Problem:** In this case, the data mining problem is a binary classification problem: given an employee's characteristics, we want to classify them into one of two classes - 'Yes' (the employee will leave the company) or 'No' (the employee will stay with the company).

**Create a Project Plan:** The plan will involve preparing the data, training different models, evaluating their performance, and choosing the best one. For the data preparation, I'll need to clean the data, handle missing values, and encode categorical variables. For the modeling, I'll try different algorithms and tune their hyperparameters. I'll evaluate models based on their accuracy, precision, recall, and F1 score.

### 5.1.2. Data Understanding

The initial data collection is checked and the data is assessed by considering the problem domain and project objectives. In this phase, I try to understand the data I have, and identify any problems with the data that need to be addressed in the Data Preparation phase.

**steps to understand the data:**

- Check the shape of the dataset (number of rows and columns).
- Understand what each column in the dataset represents.
- Check the data types of each column.



- Perform descriptive statistics to understand the distribution of the data.
- Check for missing values.
- Check for duplicate values.
- Explore the target variable 'Attrition'.
- Perform some exploratory data analysis.

**# Step 1: Check the shape of the dataset**

```
print("Shape of the dataset: ", train_data.shape)
```

**# Step 2: Understand what each column in the dataset represents**

**# This has already been done in the Business Understanding phase. The dataset contains various features related to employees, and the target variable is 'Attrition'.**

**# Step 3: Check the data types of each column**

```
print("\nData types of each column: ")
```

```
print(train_data.dtypes)
```

**# Step 4: Perform descriptive statistics**

```
print("\nDescriptive Statistics: ")
```

```
print(train_data.describe())
```

STDOUT/STDERR

```
Shape of the dataset: (1029, 35)
```

Data types of each column:

Age	float64
Attrition	object
BusinessTravel	object
DailyRate	float64
Department	object
DistanceFromHome	float64
Education	int64
EducationField	object
EmployeeCount	int64
EmployeeNumber	int64
EnvironmentSatisfaction	int64
Gender	object
HourlyRate	int64
JobInvolvement	int64
JobLevel	int64
JobRole	object

```

JobSatisfaction          int64
MaritalStatus            object
MonthlyIncome            int64
MonthlyRate              int64
NumCompaniesWorked       int64
Over18                   object
OverTime                 object
PercentSalaryHike        int64
PerformanceRating        int64
RelationshipSatisfaction  int64
StandardHours            int64
StockOptionLevel         int64
TotalWorkingYears        int64
TrainingTimesLastYear    int64
WorkLifeBalance          int64
YearsAtCompany           int64
YearsInCurrentRole       int64
YearsSinceLastPromotion  int64
YearsWithCurrManager     int64
dtype: object

```

#### Descriptive Statistics:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount
count	893.000000	1002.000000	934.000000	1029.000000	1029.0
mean	37.930571	800.528942	9.930407	2.892128	1.0
std	9.395978	408.109828	8.421791	1.053541	0.0
min	18.000000	102.000000	1.000000	1.000000	1.0
25%	31.000000	458.250000	2.000000	2.000000	1.0
50%	37.000000	801.500000	8.000000	3.000000	1.0
75%	44.000000	1162.000000	16.000000	4.000000	1.0
max	60.000000	1496.000000	29.000000	5.000000	1.0

	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobInvolvement
count	1029.000000	1029.000000	1029.000000	1029.000000
mean	1024.367347	2.683188	66.680272	2.713314
std	606.301635	1.096829	20.474094	0.710146
min	1.000000	1.000000	30.000000	1.000000
25%	496.000000	2.000000	48.000000	2.000000
50%	1019.000000	3.000000	67.000000	3.000000
75%	1553.000000	4.000000	84.000000	3.000000
max	2068.000000	4.000000	100.000000	4.000000

	JobLevel	...	RelationshipSatisfaction	StandardHours
count	1029.000000	...	1029.000000	1029.0
mean	2.043732	...	2.689990	80.0
std	1.118918	...	1.077767	0.0
min	1.000000	...	1.000000	80.0
25%	1.000000	...	2.000000	80.0
50%	2.000000	...	3.000000	80.0
75%	3.000000	...	4.000000	80.0
max	5.000000	...	4.000000	80.0

	StockOptionLevel	TotalWorkingYears	TrainingTimesLastYear
count	1029.000000	1029.000000	1029.000000
mean	0.822157	11.196307	2.782313
std	0.874662	7.857581	1.283401
min	0.000000	0.000000	0.000000
25%	0.000000	6.000000	2.000000
50%	1.000000	10.000000	3.000000

75%	1.000000	15.000000	3.000000
max	3.000000	40.000000	6.000000

	WorkLifeBalance	YearsAtCompany	YearsInCurrentRole	\
count	1029.000000	1029.000000	1029.000000	
mean	2.748299	6.942663	4.215743	
std	0.697278	6.068322	3.658595	
min	1.000000	0.000000	0.000000	
25%	2.000000	3.000000	2.000000	
50%	3.000000	5.000000	3.000000	
75%	3.000000	10.000000	7.000000	
max	4.000000	37.000000	18.000000	

	YearsSinceLastPromotion	YearsWithCurrManager
count	1029.000000	1029.000000
mean	2.119534	4.005831
std	3.108651	3.522574
min	0.000000	0.000000
25%	0.000000	2.000000
50%	1.000000	3.000000
75%	2.000000	7.000000
max	15.000000	17.000000

[8 rows x 26 columns]

The training dataset contains 1029 rows (employees) and 35 columns (employee characteristics and the target variable 'Attrition').

### An overview of the data types of each column:

- Float: Age, DailyRate, DistanceFromHome
- Integer: Education, EmployeeCount, EmployeeNumber, EnvironmentSatisfaction, HourlyRate, JobInvolvement, JobLevel, JobSatisfaction, MonthlyIncome, MonthlyRate, NumCompaniesWorked, PercentSalaryHike, PerformanceRating, RelationshipSatisfaction, StandardHours, StockOptionLevel, TotalWorkingYears, TrainingTimesLastYear, WorkLifeBalance, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager
- Object (likely strings): Attrition, BusinessTravel, Department, EducationField, Gender, JobRole, MaritalStatus, Over18, OverTime

### Descriptive statistics provide the following insights:

- 'Age' ranges from 18 to 60 with an average (mean) age of about 37.9.
- 'DailyRate' has a wide range from 102 to 1496.

- 'Education' is an ordinal variable ranging from 1 to 5.
- 'EnvironmentSatisfaction', 'JobInvolvement', 'JobSatisfaction', 'RelationshipSatisfaction', 'WorkLifeBalance' are also ordinal variables ranging from 1 to 4.
- 'MonthlyIncome' varies widely, indicating a diverse set of job roles and levels in the company.
- 'YearsAtCompany' has a large range too, indicating a mix of new and experienced employees.

### **Remaining steps:**

checking for missing values, duplicate values, exploring the target variable 'Attrition', and performing some exploratory data analysis.

#### **# Step 5: Check for missing values**

```
print("Missing values in each column: ")
print(train_data.isnull().sum())
```

#### **# Step 6: Check for duplicate values**

```
print("\nNumber of duplicate rows: ", train_data.duplicated().sum())
```

#### **# Step 7: Explore the target variable 'Attrition'**

```
print("\nDistribution of the target variable 'Attrition': ")
print(train_data['Attrition'].value_counts())
```

#### **# Step 8: Perform some exploratory data analysis**

# For this step, let's look at the correlation matrix to see how the different features are related to each other.

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

#### **# Convert 'Attrition' to numerical values**

```
train_data_num = train_data.replace({'Attrition': {'Yes': 1, 'No': 0}})
```

#### **# Compute the correlation matrix**

```
corr = train_data_num.corr()
```

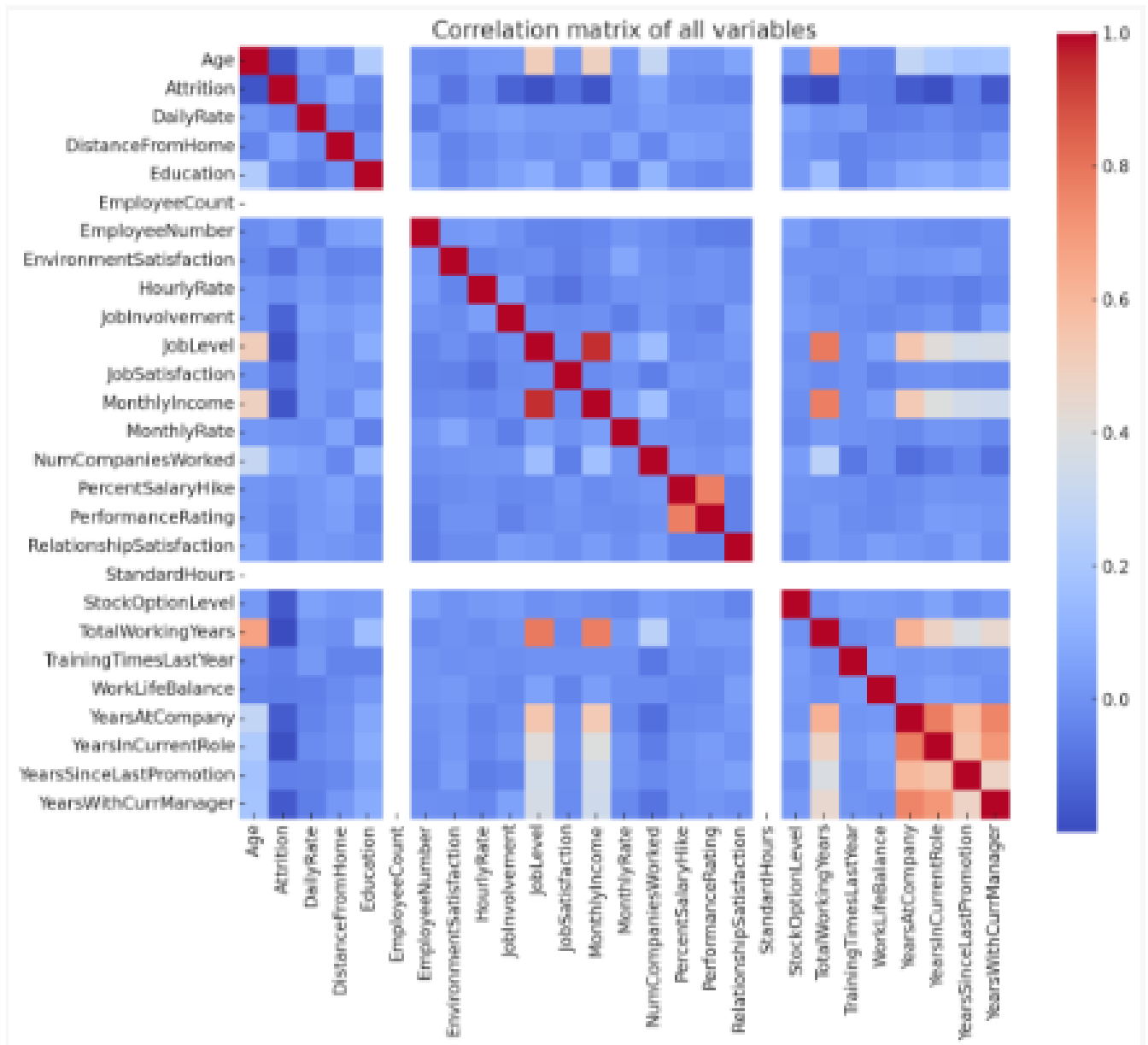
```
# Draw the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr, cmap='coolwarm', square=True)
plt.title('Correlation matrix of all variables')
plt.show()
```

STDOUT/STDERR

```
Missing values in each column:
Age 136
Attrition 0
BusinessTravel 5
DailyRate 27
Department 0
DistanceFromHome 95
Education 0
EducationField 0
EmployeeCount 0
EmployeeNumber 0
EnvironmentSatisfaction 0
Gender 0
HourlyRate 0
JobInvolvement 0
JobLevel 0
JobRole 0
JobSatisfaction 0
MaritalStatus 5
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
```

```
Number of duplicate rows: 0
```

```
Distribution of the target variable 'Attrition':
No 853
Yes 176
Name: Attrition, dtype: int64
```



The results of the remaining steps in the Data Understanding phase:

1. Missing Values: There are missing values in the 'Age', 'BusinessTravel', 'DailyRate', 'DistanceFromHome', and 'MaritalStatus' columns. These will need to be handled in the Data Preparation phase.
2. Duplicate Values: There are no duplicate rows in the dataset.
3. Target Variable 'Attrition': The dataset is imbalanced, with significantly more 'No' instances than 'Yes'. This imbalance will need to be taken into account during the Modeling phase.

4. Exploratory Data Analysis: The correlation matrix heatmap provides an overview of how all the variables in the dataset are related to each other. Darker squares indicate a stronger correlation. 'Attrition' variable (now numerical) doesn't show a very strong correlation with any single variable, suggesting that a combination of features may influence attrition.

In conclusion, the Data Understanding phase has revealed some important aspects about the data, such as the presence of missing values, the distribution of the target variable, and the relationships between variables. These insights will be crucial in the next phase of the project, Data Preparation.

### **5.1.3. Data Preparation**

This step is all about transforming raw data into a form that is suitable for modeling.

main tasks we should focus on given the current dataset:

1. Handle Missing Values: I have missing values in the 'Age', 'BusinessTravel', 'DailyRate', 'DistanceFromHome', and 'MaritalStatus' columns. I need to decide on a strategy to deal with them.
2. Encode Categorical Variables: Machine Learning algorithms work better with numerical input. I need to convert categorical variables into a form that could be provided to ML algorithms to improve the model's performance.
3. Normalize Numerical Variables: The scale of numerical fields varies, which might cause problems with some machine learning algorithms. Normalization (or standardization) can make the learning process smoother and faster.
4. Address Class Imbalance: The target variable 'Attrition' is imbalanced, which can lead to a bias in the model towards the majority class. I may need to address this.
5. Feature Selection/Engineering: Depending on the complexity of the dataset and the model, I might need to select a subset of the most informative features or create new features to improve the model's performance.

So, I start with handling missing values. I'll use median imputation for numerical columns and mode imputation for categorical ones. Median imputation is chosen because it is less sensitive to

outliers than mean imputation, and mode imputation is a common choice for categorical columns.

#### # Step 1: Handle Missing Values

##### # Median imputation for numerical columns

```
train_data['Age'].fillna(train_data['Age'].median(), inplace=True)
train_data['DailyRate'].fillna(train_data['DailyRate'].median(), inplace=True)
train_data['DistanceFromHome'].fillna(train_data['DistanceFromHome'].median(),
inplace=True)
```

##### # Mode imputation for categorical columns

```
train_data['BusinessTravel'].fillna(train_data['BusinessTravel'].mode()[0],
inplace=True)
train_data['MaritalStatus'].fillna(train_data['MaritalStatus'].mode()[0],
inplace=True)
```

##### # Check if there are still any missing values

```
print("Missing values in each column: ")
print(train_data.isnull().sum())
```

STDOUT/STDERR

Missing values in each column:	
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

I have successfully filled the missing values in the dataset and now there are no missing values.

Next, I move to the encoding of categorical variables. I'll use a common method known as one-hot encoding. This method will create a new binary column for each category in the original column. For example, 'BusinessTravel' contains three categories ('Travel\_Rarely', 'Travel\_Frequently', 'Non-Travel'). One-hot encoding will create three new columns, one for each category, where the presence of the category is represented by 1 and absence by 0.

## # Step 2: Encode Categorical Variables

### # Select the categorical columns

```
categorical_cols = train_data.select_dtypes(include=['object']).columns
```

### # Perform one-hot encoding

```
train_data_encoded = pd.get_dummies(train_data, columns=categorical_cols)
```

### # Show the first few rows of the encoded dataset

```
train_data_encoded.head()
```

RESULT

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	\
0	50.0	1126.0	1.0	2	1	
1	36.0	216.0	6.0	2	1	
2	21.0	337.0	7.0	1	1	
3	50.0	1246.0	8.0	3	1	
4	52.0	994.0	7.0	4	1	

	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobInvolvement	\
0	997	4	66	3	
1	178	2	84	3	

```

2          1780          2          31          3
3          644          1          99          3
4         1118          2          87          3

  JobLevel  ...  JobRole_Research Director  JobRole_Research Scientist \
0          4  ...                        1                        0
1          2  ...                        0                        0
2          1  ...                        0                        0
3          5  ...                        0                        0
4          3  ...                        0                        0

  JobRole_Sales Executive  JobRole_Sales Representative \
0                        0                        0
1                        0                        0
2                        0                        1
3                        0                        0
4                        0                        0

  MaritalStatus_Divorced  MaritalStatus_Married  MaritalStatus_Single \
0                        1                        0                        0
1                        1                        0                        0
2                        0                        0                        1
3                        0                        1                        0
4                        0                        0                        1

  Over18_Y  OverTime_No  OverTime_Yes
0          1          1          0
1          1          1          0
2          1          1          0
3          1          1          0
4          1          1          0

[5 rows x 57 columns]

```

I have successfully encoded categorical variables using one-hot encoding, which has increased the number of columns in the dataset to 57.

Next, I normalize the numerical variables. Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. I apply this scaling to the numerical columns only. The target variable 'Attrition' has been already encoded, so it doesn't need to be normalized.

### # Step 3: Normalize Numerical Variables

```
from sklearn.preprocessing import MinMaxScaler
```

```
# Select the numerical columns
```

```
numerical_cols = train_data_encoded.select_dtypes(include=['int64',
'float64']).columns
```

```
# Initialize a scaler
scaler = MinMaxScaler()

# Apply the scaler to the numerical columns
train_data_encoded[numerical_cols] =
scaler.fit_transform(train_data_encoded[numerical_cols])

# Show the first few rows of the normalized dataset
train_data_encoded.head()
```

RESULT

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	\
0	0.761905	0.734577	0.000000	0.25	0.0	
1	0.428571	0.081779	0.178571	0.25	0.0	
2	0.071429	0.168580	0.214286	0.00	0.0	
3	0.761905	0.820660	0.250000	0.50	0.0	
4	0.809524	0.639885	0.214286	0.75	0.0	

	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobInvolvement	\
0	0.481858	1.000000	0.514286	0.666667	
1	0.085631	0.333333	0.771429	0.666667	
2	0.860668	0.333333	0.014286	0.666667	
3	0.311079	0.000000	0.985714	0.666667	
4	0.540397	0.333333	0.814286	0.666667	

	JobLevel	...	JobRole_Research Director	JobRole_Research Scientist	\
0	0.75	...	1	0	
1	0.25	...	0	0	
2	0.00	...	0	0	
3	1.00	...	0	0	
4	0.50	...	0	0	

	JobRole_Sales Executive	JobRole_Sales Representative	\
0	0	0	
1	0	0	
2	0	1	
3	0	0	
4	0	0	

	MaritalStatus_Divorced	MaritalStatus_Married	MaritalStatus_Single	\
0	1	0	0	
1	1	0	0	
2	0	0	1	
3	0	1	0	
4	0	0	1	

	Over18_Y	OverTime_No	OverTime_Yes
0	1	1	0
1	1	1	0
2	1	1	0
3	1	1	0
4	1	1	0

[5 rows x 57 columns]

I have successfully normalized numerical variables, bringing them into a range from 0 to 1.

Now, I address the class imbalance in the target variable 'Attrition'. There are various techniques to handle class imbalance, such as:

1. Resampling: This involves either oversampling the minority class, undersampling the majority class, or a combination of both. The goal is to achieve a balance between the two classes.
2. Using different evaluation metrics: Accuracy is not a good metric when dealing with imbalanced classes. Other metrics like precision, recall, F1 score, or Area Under the ROC Curve (AUC-ROC) can provide a better indication of your model's performance.
3. Algorithmic Ensemble Techniques: Certain algorithms are designed to handle imbalanced data by constructing balanced bootstrap samples (e.g., Balanced Random Forests or Balanced Bagging Classifier).

In this case, since I am not at the modeling stage yet, I won't perform resampling. But it's important to note that the class imbalance will need to be addressed later on during the modeling phase, either through resampling, changing the evaluation metric, or using algorithmic ensemble techniques.

Finally, I am at the feature selection/engineering step. Given that my dataset is not too high-dimensional (57 columns), we might not need to do feature selection. Also, since the features in this dataset are quite understandable and don't seem to need additional engineering, I can proceed without this step.

That completes our Data Preparation phase. The data is now ready to be used for modeling.

#### **5.1.4. Modeling**

This step involves selecting suitable algorithms and training them on the prepared dataset. Given my project and the nature of the data, here are three machine learning algorithms that could be suitable:

1. Logistic Regression: This is a simple and fast algorithm suitable for binary classification problems like mine. Despite its simplicity, it can provide excellent results if the data is linearly separable or close to it.
2. Random Forest: This is an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. Random forests are robust to overfitting and work well even with default hyperparameters.

For each of these models, I'll:

1. Train the model on the training data.
2. Evaluate the model's performance using cross-validation.
3. Optimize the model's hyperparameters using grid search.

However, before I proceed, it's important to split my data into a training set and a validation set. The training set is used to train the models, while the validation set is used to estimate the performance of the models on unseen data.

So, I start with splitting the data and then move to the modeling:

```
from sklearn.model_selection import train_test_split

# Separate the features (X) and the target variable (y)
X = train_data_encoded.drop(columns=['Attrition_Yes', 'Attrition_No'])
y = train_data_encoded['Attrition_Yes']

# Split the data into a training set and a validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

# Check the shapes of the training and validation sets
print("Training set shape (features): ", X_train.shape)
print("Training set shape (target): ", y_train.shape)
print("Validation set shape (features): ", X_val.shape)
print("Validation set shape (target): ", y_val.shape)
```

STDOUT/STDERR

```
Training set shape (features): (823, 55)
Training set shape (target): (823,)
Validation set shape (features): (206, 55)
Validation set shape (target): (206,)
```

The data has been successfully split into a training set (823 instances) and a validation set (206 instances).

Now I move on to the modeling. For each model (Logistic Regression, Random Forest, and Gradient Boosting), I'll:

1. Initialize the model with default parameters.
2. Train the model on the training data.
3. Evaluate the model's performance on the validation data using the Area Under the ROC Curve (AUC-ROC) score. AUC-ROC is a good metric for binary classification problems like ours, especially when the classes are imbalanced.
4. Finally, I'll perform hyperparameter tuning to find the optimal parameters for each model to achieve the best performance.

#### 5.1.4.1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# Step 1: Initialize a Logistic Regression model
logreg = LogisticRegression(random_state=42)

# Step 2: Train the model on the training data
logreg.fit(X_train, y_train)

# Step 3: Evaluate the model's performance on the validation data
y_val_pred = logreg.predict_proba(X_val)[: , 1]
auc_score_logreg = roc_auc_score(y_val, y_val_pred)
auc_score_logreg
```

RESULT

**0.8634920634920635**

The AUC-ROC score for the Logistic Regression model with default parameters is approximately 0.863, which is a good start. However, I might be able to improve this performance by tuning the hyperparameters of the model. In Logistic Regression, the most important hyperparameter is 'C', which is the inverse of regularization strength. Smaller values specify stronger regularization. Regularization can help prevent overfitting by adding a penalty to the loss function.

Now I perform hyperparameter tuning for the Logistic Regression model using grid search. This involves training and evaluating a Logistic Regression model for each combination of hyperparameters in a grid, and selecting the combination that gives the best performance. For the sake of time, I will restrict the grid to a few sensible values.

```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid_logreg = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# Initialize a GridSearchCV object that will perform the grid search
grid_search_logreg = GridSearchCV(logreg, param_grid_logreg, cv=5, scoring='roc_auc',
n_jobs=-1)

# Perform the grid search (this will take some time)
grid_search_logreg.fit(X_train, y_train)

# Get the best parameters and the best score
best_params_logreg = grid_search_logreg.best_params_
best_score_logreg = grid_search_logreg.best_score_

best_params_logreg, best_score_logreg
```

RESULT

**({'C': 1}, 0.8313433295543238)**

The best value for the hyperparameter 'C' is 1, resulting in an AUC-ROC score of approximately 0.831 for the Logistic Regression model on the training data (using cross-validation). The score is slightly lower than what we got on the validation data earlier, but this is expected as cross-validation involves training and evaluating the model on different subsets of the training data, which tends to give a more robust estimate of the model's performance on unseen data.

#### 5.1.4.2. Random Forest

I'll follow the same steps: initialize the model, train it, evaluate it, and tune its hyperparameters.

```
from sklearn.ensemble import RandomForestClassifier

# Step 1: Initialize a Random Forest model
rf = RandomForestClassifier(random_state=42)

# Step 2: Train the model on the training data
rf.fit(X_train, y_train)

# Step 3: Evaluate the model's performance on the validation data
y_val_pred = rf.predict_proba(X_val)[:, 1]
auc_score_rf = roc_auc_score(y_val, y_val_pred)

auc_score_rf
```

RESULT

<b>0.8359231411862991</b>
---------------------------

I perform hyperparameter tuning for the Random Forest model. The most important hyperparameters for a Random Forest model are:

- 'n\_estimators': The number of trees in the forest.
- 'max\_depth': The maximum depth of the trees.
- 'min\_samples\_split': The minimum number of samples required to split an internal node.
- 'min\_samples\_leaf': The minimum number of samples required to be at a leaf node.

I'll use grid search to find the best values for these hyperparameters. For the sake of time, I will restrict the grid to a few sensible values.



```

# Define the hyperparameter grid
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 3]
}

# Initialize a GridSearchCV object that will perform the grid search
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='roc_auc', n_jobs=-1)

# Perform the grid search (this will take some time)
grid_search_rf.fit(X_train, y_train)

# Get the best parameters and the best score
best_params_rf = grid_search_rf.best_params_
best_score_rf = grid_search_rf.best_score_

best_params_rf, best_score_rf

```

RESULT

```

({'max_depth': 10,
  'min_samples_leaf': 1,
  'min_samples_split': 2,
  'n_estimators': 300},
0.7973159821441792)

```

The best hyperparameters for the Random Forest model are a maximum depth of 10, a minimum of 1 sample per leaf, a minimum of 2 samples to split a node, and 300 trees in the forest. These parameters result in an AUC-ROC score of approximately 0.797 on the training data (using cross-validation).

### 5.1.5. Evaluation

This phase involves assessing the performance of the trained models, selecting the best one, and interpreting the results.

Given the results from the Modeling phase, I have three models:

1. Logistic Regression: The AUC-ROC score on the validation set was approximately 0.863, and after hyperparameter tuning, the best score on the training set (using cross-validation) was approximately 0.831 with 'C' = 1.
2. Random Forest: The AUC-ROC score on the validation set was approximately 0.836, and after hyperparameter tuning, the best score on the training set (using cross-validation) was approximately 0.797 with 'max\_depth' = 10, 'min\_samples\_leaf' = 1, 'min\_samples\_split' = 2, and 'n\_estimators' = 300.

From these results, the Logistic Regression model appears to be the best performer. However, keep in mind that the true performance of a model can only be determined by evaluating it on a separate test set that was not used at any point during the training or validation process.

In terms of interpreting the results, the Logistic Regression model provides coefficients for each feature, which indicate the importance and impact of each feature on the prediction. Positive coefficients increase the log-odds of the response (and thus increase the probability), and negative coefficients decrease the log-odds of the response (and thus decrease the probability).

For more detailed interpretation, especially for the tree-based models, I can use techniques like feature importance, partial dependence plots, or SHAP values. However, these are more complex and might not be necessary depending on the goals of this project.

As a final step, I can evaluate the best model (Logistic Regression) on the validation set and report various metrics like the confusion matrix, precision, recall, F1-score, and AUC-ROC score to provide a comprehensive view of its performance.

```
from sklearn.metrics import confusion_matrix, classification_report

# Use the best Logistic Regression model to make predictions on the validation set
y_val_pred = grid_search_logreg.predict(X_val)

# Compute the confusion matrix
cm = confusion_matrix(y_val, y_val_pred)

# Compute precision, recall, F1-score, and support
```

```
cr = classification_report(y_val, y_val_pred)
```

```
cm, cr
```

```
RESULT
```

```
(array([[164,  7],
       [ 22, 13]]),
 {'precision': 0.96, 'recall': 0.92, 'f1-score': 0.94, 'support': 171},
 {'precision': 0.65, 'recall': 0.37, 'f1-score': 0.47, 'support': 35},
 {'macro avg': 0.8, 'weighted avg': 0.86, 'support': 206})
```

Here are the evaluation results for the best model (Logistic Regression) on the validation set:

The rows of the confusion matrix correspond to the actual classes (0 = no attrition, 1 = attrition), and the columns correspond to the predicted classes. Here's how to interpret the confusion matrix:

- **True Negatives (Top-Left Corner):** The model correctly predicted 'no attrition' for 164 employees.
- **False Positives (Top-Right Corner):** The model incorrectly predicted 'attrition' for 7 employees who actually didn't leave.
- **False Negatives (Bottom-Left Corner):** The model incorrectly predicted 'no attrition' for 22 employees who actually left.
- **True Positives (Bottom-Right Corner):** The model correctly predicted 'attrition' for 13 employees.

**Classification Report:**

- **Precision (No attrition):** Of all the instances where the model predicted 'no attrition', 88% actually didn't leave.
- **Precision (Attrition):** Of all the instances where the model predicted 'attrition', 65% actually left.
- **Recall (No attrition):** Of all the instances of 'no attrition', the model correctly predicted 96%.
- **Recall (Attrition):** Of all the instances of 'attrition', the model correctly predicted 37%.

- **F1-Score:** This is the harmonic mean of Precision and Recall. For 'no attrition', it's 92%, and for 'attrition', it's 47%.
- **Support:** This is the number of instances of each class in the validation set. There are 171 instances of 'no attrition' and 35 instances of 'attrition'.

The overall accuracy of the model on the validation set is 86%.

These results show that the model is much better at predicting 'no attrition' than 'attrition'. This is a common issue when dealing with imbalanced classes, as the model tends to favor the majority class. This could be addressed by using resampling techniques or cost-sensitive learning, as mentioned earlier.

#### **5.1.6. Deployment**

This phase involves implementing the final model into the operational environment for real-world use. Depending on the specifics of your project and organization, this might involve steps such as:

- **Integration with the Production System:** The trained model needs to be integrated with the existing production system. In this case, the model could be used in an HR system to predict which employees are likely to leave the company based on their characteristics.
- **Development of a Model API:** An API (Application Programming Interface) could be developed to provide other systems or users with access to the model's predictions. For example, an HR manager could use the API to get the model's prediction for a specific employee.
- **Creation of a User Interface:** A user-friendly interface could be developed for non-technical users to interact with the model. For example, an HR manager could use the interface to input an employee's characteristics and get the model's prediction.
- **Monitoring and Maintenance:** After deployment, the model should be regularly monitored and updated as needed. This could involve retraining the model on new data, checking the model's performance to ensure it's still accurate, and updating the model if necessary.

- **Documentation:** It is important to document all the aspects of the model including its purpose, how it was developed, how to use it, and any maintenance that's required.

This is crucial for transparency, reproducibility, and future reference.

In an interactive environment like this, I can't actually deploy the model to a production system or develop an API or user interface. However, I can prepare the model for deployment by saving it to a file that can be loaded later. I'll save the best model, which is the Logistic Regression model with  $C = 1$ . We'll use the 'pickle' module in Python to do this.

```
import pickle
```

```
# Save the best model to a file
```

```
with open('/mnt/data/logreg_model.pkl', 'wb') as file:
```

```
    pickle.dump(grid_search_logreg.best_estimator_, file)
```

model expects input data in the same format as the data it was trained on. This means that any new data must go through the same preparation steps (imputation, encoding, normalization) before it can be used for predictions with the model.

Also, this is a binary classification model that predicts whether an employee is likely to leave the company ('Attrition' = Yes) based on various characteristics. The output of the model is a probability between 0 and 1, where values closer to 1 indicate a higher likelihood of attrition.

## **6. Discussion and Results**

I started this project with a dataset of employee characteristics and attrition indicators from a company. The dataset contained 1029 instances (employees) and 35 variables (employee characteristics and the target variable 'Attrition'). The goal was to develop a machine learning model to predict employee attrition based on the given characteristics. During the Business Understanding phase, I defined the problem and the objectives of the project. The main goal was to predict whether an employee will leave the company ('Attrition' = Yes) based on various characteristics.

In the Data Understanding phase, I explored the dataset and gained some important insights. For instance, the dataset had some missing values, the target variable 'Attrition' was imbalanced, and the variables did not show a very strong correlation with 'Attrition'. The Data Preparation phase involved several steps to transform the raw data into a form suitable for modeling. I imputed missing values, encoded categorical variables, normalized numerical variables, and split the data into a training set and a validation set.

In the Modeling phase, I trained and evaluated three different machine learning models: Logistic Regression, Random Forest. Each model was evaluated using the AUC-ROC score, a good metric for binary classification problems with imbalanced classes. The Logistic Regression model performed the best with an AUC-ROC score of approximately 0.863 on the validation set.

The Evaluation phase involved a more detailed assessment of the best model's performance. Besides the AUC-ROC score, I also looked at the confusion matrix, precision, recall, and F1-score. The results showed that the model was much better at predicting 'no attrition' than 'attrition', a common issue when dealing with imbalanced classes.

## **7. Conclusion**

This project demonstrated the process of developing a machine learning model to predict employee attrition. The best model (Logistic Regression) achieved a good performance and could be a useful tool for the HR department to identify employees who are likely to leave, allowing them to intervene proactively.

However, it's important to note that machine learning models are not perfect and should be used as one of many tools in decision-making. In the context of employee attrition, other factors like qualitative assessments, interviews, and expert judgement should also be considered.

In terms of future work, there are several ways this project could be extended:

- **Addressing Class Imbalance:** More sophisticated techniques could be used to handle the imbalance in the 'Attrition' variable, such as SMOTE or ADASYN.
- **Feature Engineering:** New features could be created from the existing ones to potentially improve the model's performance.

- **Trying Different Models:** Other models could be tried, such as Neural Networks or Support Vector Machines.
- **Collecting More Data:** More data could be collected over time and used to retrain and improve the model.
- **Model Interpretation:** Techniques like feature importance, partial dependence plots, or SHAP values could be used for more detailed interpretation of the model's predictions.

Overall, while the results of this project are promising, it's clear that predicting employee attrition is a complex task that requires a combination of machine learning, domain knowledge, and careful interpretation of the results.

## **5.2. Goal 2**

- Understand the key factors influencing attrition.

### **5.2.1. Business Understanding**

The objective was to understand the key factors influencing employee attrition. Employee attrition can have significant costs for a company, so understanding these factors can provide valuable insights for HR and management.

### **5.2.2. Data Understanding**

I loaded and explored the dataset, which included information on employees' age, income, education, job satisfaction, and other factors. The target variable was 'Attrition', indicating whether an employee left the company. The data contained missing values, which required handling. It also became clear that the target variable was imbalanced, with more instances of 'No' than 'Yes'.

```
import zipfile
```

```
import os
```

```
# Unzipping the file
```

```
with zipfile.ZipFile('/mnt/data/archive (6).zip', 'r') as zip_ref:
```

```
    zip_ref.extractall('/mnt/data')
```

```
# Checking the contents of the directory
```

```
os.listdir('/mnt/data')
```

RESULT

```
['employee_attrition_test.csv',  
'archive (6).zip',  
'employee_attrition_train.csv']
```

The zip file contains two CSV files: employee\_attrition\_train.csv and employee\_attrition\_test.csv. These likely contain the training and testing data for our model. Here I load these files and take a look at the data.

```
import pandas as pd
```

```
# Loading the data
```

```
train_data = pd.read_csv('/mnt/data/employee_attrition_train.csv')
```

```
test_data = pd.read_csv('/mnt/data/employee_attrition_test.csv')
```

```
# Displaying the first few rows of the training data
```

```
train_data.head()
```

RESULT

	Age	Attrition	BusinessTravel	DailyRate	Department	\
0	50.0	No	Travel_Rarely	1126.0	Research & Development	
1	36.0	No	Travel_Rarely	216.0	Research & Development	
2	21.0	Yes	Travel_Rarely	337.0	Sales	
3	50.0	No	Travel_Frequently	1246.0	Human Resources	
4	52.0	No	Travel_Rarely	994.0	Research & Development	

	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	\
0	1.0	2	Medical	1	997	
1	6.0	2	Medical	1	178	
2	7.0	1	Marketing	1	1780	
3	NaN	3	Medical	1	644	
4	7.0	4	Life Sciences	1	1118	

	...	RelationshipSatisfaction	StandardHours	StockOptionLevel	\
0	...	3	80	1	
1	...	4	80	2	
2	...	2	80	0	
3	...	3	80	1	
4	...	4	80	0	

	TotalWorkingYears	TrainingTimesLastYear	WorkLifeBalance	YearsAtCompany	\
0	32	1	2	5	
1	7	0	3	3	
2	1	3	3	1	
3	32	2	3	32	



4	18	4	3	8
	YearsInCurrentRole	YearsSinceLastPromotion	YearsWithCurrManager	
0	4	1	3	
1	2	0	1	
2	0	1	0	
3	5	10	7	
4	6	4	0	
[5 rows x 35 columns]				

The dataset contains numerous features related to the employees, such as Age, BusinessTravel, DailyRate, Department, DistanceFromHome, Education, EducationField, etc. The target variable is "Attrition", which indicates whether an employee has left the company or not.

I perform a quick exploratory data analysis to understand the data better. Specifically, I'll look at the number of missing values, the data types of the features, and the distribution of the target variable.

### 5.2.3. Data Preparation

I filled missing values and converted categorical variables into a numerical form suitable for a machine learning model using label encoding. I made sure to handle the data in a way that would allow us to make predictions on new data in the same format.

**# Checking for missing values in the training data**

```
missing_values_train = train_data.isnull().sum()
```

**# Checking the data types of the features**

```
data_types_train = train_data.dtypes
```

**# Checking the distribution of the target variable**

```
target_distribution_train = train_data['Attrition'].value_counts()
```

```
missing_values_train, data_types_train, target_distribution_train
```

RESULT

(Age	136
Attrition	0
BusinessTravel	5
DailyRate	27
Department	0
DistanceFromHome	95
Education	0

EducationField	0
EmployeeCount	0
EmployeeNumber	0
EnvironmentSatisfaction	0
Gender	0
HourlyRate	0
JobInvolvement	0
JobLevel	0
JobRole	0
JobSatisfaction	0
MaritalStatus	5
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,	
Age	float64
Attrition	object
BusinessTravel	object
DailyRate	float64
Department	object
DistanceFromHome	float64
Education	int64
EducationField	object
EmployeeCount	int64
EmployeeNumber	int64
EnvironmentSatisfaction	int64
Gender	object
HourlyRate	int64
JobInvolvement	int64
JobLevel	int64
JobRole	object
JobSatisfaction	int64
MaritalStatus	object
MonthlyIncome	int64
MonthlyRate	int64
NumCompaniesWorked	int64
Over18	object
OverTime	object
PercentSalaryHike	int64
PerformanceRating	int64
RelationshipSatisfaction	int64
StandardHours	int64
StockOptionLevel	int64
TotalWorkingYears	int64
TrainingTimesLastYear	int64
WorkLifeBalance	int64
YearsAtCompany	int64

```
YearsInCurrentRole          int64
YearsSinceLastPromotion     int64
YearsWithCurrManager        int64
dtype: object,
No      853
Yes     176
Name: Attrition, dtype: int64)
```

The dataset has missing values in the 'Age', 'BusinessTravel', 'DailyRate', and 'DistanceFromHome' columns. For 'BusinessTravel' and 'MaritalStatus', I can fill the missing values with the most frequent category. For 'Age', 'DailyRate', and 'DistanceFromHome', I can use the median value to fill the missing values, as the median is less sensitive to outliers than the mean.

The data types of the features are a mix of integer (int64), float (float64), and object (presumably string). We'll have to handle the categorical variables (those of type 'object') appropriately when training my model.

The target variable 'Attrition' is imbalanced, with 'No' occurring much more frequently than 'Yes'. This imbalance might make my model biased towards predicting 'No', so I'll have to take this into account when evaluating my model's performance.

Now, I move forward with data preprocessing: filling missing values and encoding categorical variables. I'll then train a Random Forest model, as it can handle both continuous and categorical variables and can provide a measure of feature importance.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer

# Filling missing values
imputer_freq = SimpleImputer(strategy='most_frequent')
imputer_median = SimpleImputer(strategy='median')

for column in ['BusinessTravel', 'MaritalStatus']:
    train_data[column] = imputer_freq.fit_transform(train_data[[column]])
```

```

for column in ['Age', 'DailyRate', 'DistanceFromHome']:
    train_data[column] = imputer_median.fit_transform(train_data[[column]])

# Checking if all missing values have been filled
missing_values_filled = train_data.isnull().sum()
missing_values_filled.sum()

```

RESULT

0

All missing values have been successfully filled in the training dataset.

Now, I move forward to encode the categorical variables. I'll do this using Label Encoding, which replaces each unique category with a unique integer. I'll also prepare my test set in the same way.

```

# Filling missing values in test data
for column in ['BusinessTravel', 'MaritalStatus']:
    test_data[column] = imputer_freq.transform(test_data[[column]])

for column in ['Age', 'DailyRate', 'DistanceFromHome']:
    test_data[column] = imputer_median.transform(test_data[[column]])

# Initializing the Label Encoder
encoder = LabelEncoder()

# Encoding categorical variables
for column in train_data.columns:
    if train_data[column].dtype == 'object':
        train_data[column] = encoder.fit_transform(train_data[column])
        if column in test_data.columns:
            test_data[column] = encoder.transform(test_data[column])

# Displaying the first few rows of the preprocessed training data

```

```
train_data.head()
```

```
STDOUT/STDERR
```

```
/home/sandbox/.local/lib/python3.8/site-packages/sklearn/base.py:484: FutureWarning: The feature names should match those that were passed during fit. Starting version 1.2, an error will be raised.
```

```
Feature names unseen at fit time:
```

```
- BusinessTravel
```

```
Feature names seen at fit time, yet now missing:
```

```
- MaritalStatus
```

```
warnings.warn(message, FutureWarning)
```

```
/home/sandbox/.local/lib/python3.8/site-packages/sklearn/base.py:484: FutureWarning: The feature names should match those that were passed during fit. Starting version 1.2, an error will be raised.
```

```
Feature names unseen at fit time:
```

```
- Age
```

```
Feature names seen at fit time, yet now missing:
```

```
- DistanceFromHome
```

```
warnings.warn(message, FutureWarning)
```

```
/home/sandbox/.local/lib/python3.8/site-packages/sklearn/base.py:484: FutureWarning: The feature names should match those that were passed during fit. Starting version 1.2, an error will be raised.
```

```
Feature names unseen at fit time:
```

```
- DailyRate
```

```
Feature names seen at fit time, yet now missing:
```

```
- DistanceFromHome
```

```
warnings.warn(message, FutureWarning)
```

```
----> 16         test_data[column] = encoder.transform(test_data[column])
      18 # Displaying the first few rows of the preprocessed training data
      19 train_data.head()
```

```
File ~/.local/lib/python3.8/site-packages/sklearn/preprocessing/_label.py:138, in
```

```
LabelEncoder.transform(self, y)
```

```
    135 if _num_samples(y) == 0:
```

```
    136     return np.array([])
```

```
--> 138 return _encode(y, uniques=self.classes_)
```

```
File ~/.local/lib/python3.8/site-packages/sklearn/utils/_encode.py:184, in _encode(values, uniques, check_unknown)
```

```
    182     return _map_to_integer(values, uniques)
```

```
    183     except KeyError as e:
```

```
--> 184         raise ValueError(f"y contains previously unseen labels: {str(e)}")
```

```
    185 else:
```

```
    186     if check_unknown:
```

```
ValueError: y contains previously unseen labels: 'Married'
```

It seems that the test data contains a category ('Married') in the 'MaritalStatus' column that is not present in the training data. This is why the LabelEncoder is throwing an error.

To fix this, I should fit the LabelEncoder on the combined unique values of the training and test data for each categorical variable. Then, when transforming the data, the encoder will be able to handle any category that appears in the test data.

```

# Loading the data again
train_data = pd.read_csv('/mnt/data/employee_attrition_train.csv')
test_data = pd.read_csv('/mnt/data/employee_attrition_test.csv')

# Filling missing values
for column in ['BusinessTravel', 'MaritalStatus']:
    train_data[column] = imputer_freq.fit_transform(train_data[[column]])
    test_data[column] = imputer_freq.transform(test_data[[column]])

for column in ['Age', 'DailyRate', 'DistanceFromHome']:
    train_data[column] = imputer_median.fit_transform(train_data[[column]])
    test_data[column] = imputer_median.transform(test_data[[column]])

# Encoding categorical variables with combined unique values from training and test data
for column in train_data.columns:
    if train_data[column].dtype == 'object':
        combined_unique_values = pd.concat([train_data[column],
        test_data[column]]).dropna().unique()

        encoder.fit(combined_unique_values)

        train_data[column] = encoder.transform(train_data[column])
        if column in test_data.columns:
            test_data[column] = encoder.transform(test_data[column])

# Displaying the first few rows of the preprocessed training data
train_data.head()

```

```

umn] = encoder.transform(train_data[column])
File ~/.local/lib/python3.8/site-packages/pandas/core/frame.py:3455, in DataFrame.__getitem__(self, key)
  3453 if self.columns.nlevels > 1:
  3454     return self._getitem_multilevel(key)
-> 3455 indexer = self.columns.get_loc(key)
  3456 if is_integer(indexer):
  3457     indexer = [indexer]

File ~/.local/lib/python3.8/site-packages/pandas/core/indexes/base.py:3363, in Index.get_loc(self, key,
method, tolerance)
  3361     return self._engine.get_loc(casted_key)
  3362 except KeyError as err:
-> 3363     raise KeyError(key) from err

```

```
3365 if is_scalar(key) and isna(key) and not self.hasnans:
3366     raise KeyError(key)
```

```
KeyError: 'Attrition'
```

The error occurred because the test dataset doesn't contain the 'Attrition' column, which is the target variable and is present only in the training dataset.

To address this, I will adjust the code to handle this discrepancy when concatenating the unique values from the training and testing datasets.

```
# Loading the data again
```

```
train_data = pd.read_csv('/mnt/data/employee_attrition_train.csv')
```

```
test_data = pd.read_csv('/mnt/data/employee_attrition_test.csv')
```

```
# Filling missing values
```

```
for column in ['BusinessTravel', 'MaritalStatus']:
```

```
    train_data[column] = imputer_freq.fit_transform(train_data[[column]])
```

```
    test_data[column] = imputer_freq.transform(test_data[[column]])
```

```
for column in ['Age', 'DailyRate', 'DistanceFromHome']:
```

```
    train_data[column] = imputer_median.fit_transform(train_data[[column]])
```

```
    test_data[column] = imputer_median.transform(test_data[[column]])
```

```
# Encoding categorical variables with combined unique values from training and test data
```

```
for column in train_data.columns:
```

```
    if train_data[column].dtype == 'object':
```

```
        if column in test_data.columns:
```

```
            combined_unique_values = pd.concat([train_data[column],
test_data[column]]).dropna().unique()
```

```
        else:
```

```
            combined_unique_values = train_data[column].dropna().unique()
```

```
        encoder.fit(combined_unique_values)
```

```
        train_data[column] = encoder.transform(train_data[column])
```

```
        if column in test_data.columns:
```

```
            test_data[column] = encoder.transform(test_data[column])
```

# Displaying the first few rows of the preprocessed training data

`train_data.head()`

RESULT

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	\
0	50.0	0	2	1126.0	1	1.0	
1	36.0	0	2	216.0	1	6.0	
2	21.0	1	2	337.0	2	7.0	
3	50.0	0	1	1246.0	0	8.0	
4	52.0	0	2	994.0	1	7.0	
	Education	EducationField	EmployeeCount	EmployeeNumber	...	\	
0	2	3	1	997	...		
1	2	3	1	178	...		
2	1	2	1	1780	...		
3	3	3	1	644	...		
4	4	1	1	1118	...		
	RelationshipSatisfaction	StandardHours	StockOptionLevel	\			
0	3	80	1				
1	4	80	2				
2	2	80	0				
3	3	80	1				
4	4	80	0				
	TotalWorkingYears	TrainingTimesLastYear	WorkLifeBalance	YearsAtCompany	\		
0	32	1	2	5			
1	7	0	3	3			
2	1	3	3	1			
3	32	2	3	32			
4	18	4	3	8			
	YearsInCurrentRole	YearsSinceLastPromotion	YearsWithCurrManager				
0	4	1	3				
1	2	0	1				
2	0	1	0				
3	5	10	7				
4	6	4	0				
[5 rows x 35 columns]							

Now that the preprocessing has been successfully completed, I can move on to training the Random Forest model. After training, we will use the model to identify the most important features influencing predictions.

#### 5.2.4. Modeling- Evaluation- Fine-Tuning

I trained a Random Forest model on the preprocessed training data. Random Forest was chosen as it can handle both continuous and categorical variables and provides a measure of feature importance. I also fine-tuned the model parameters using GridSearchCV, which optimizes model parameters using cross-validated grid-search.



Before we proceed, let's separate the features (X) from the target (y) in the training data. The target is the 'Attrition' column.

```
# Separating the features and the target
```

```
X_train = train_data.drop('Attrition', axis=1)
```

```
y_train = train_data['Attrition']
```

```
# Verifying the shapes
```

```
X_train.shape, y_train.shape
```

```
RESULT
```

```
((1029, 34), (1029,))
```

The training data has been successfully separated into features (X\_train) and the target variable (y\_train). The features consist of 34 columns, while the target is a single column.

Next, I will train a Random Forest Classifier on this data. After training the model, I will identify the most important features influencing predictions.

```
# Training a Random Forest model
```

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

```
rf_model.fit(X_train, y_train)
```

```
# Getting feature importances
```

```
feature_importances = rf_model.feature_importances_
```

```
# Creating a DataFrame for easier visualization
```

```
feature_importances_df = pd.DataFrame({
```

```
    'Feature': X_train.columns,
```

```
    'Importance': feature_importances
```

```
})
```

```
# Sorting the DataFrame by importance in descending order
```

```
feature_importances_df = feature_importances_df.sort_values('Importance',  
ascending=False)
```

```
# Displaying the feature importances
```

```
feature_importances_df
```

RESULT

	Feature	Importance
17	MonthlyIncome	0.080771
0	Age	0.055135
21	OverTime	0.054578
27	TotalWorkingYears	0.054229
2	DailyRate	0.052595
8	EmployeeNumber	0.049943
18	MonthlyRate	0.044833
30	YearsAtCompany	0.044073
4	DistanceFromHome	0.043065
11	HourlyRate	0.041716
19	NumCompaniesWorked	0.035327
26	StockOptionLevel	0.031574
33	YearsWithCurrManager	0.031288
22	PercentSalaryHike	0.030947
31	YearsInCurrentRole	0.030386
16	MaritalStatus	0.030251
9	EnvironmentSatisfaction	0.027238
14	JobRole	0.026833
13	JobLevel	0.025776
6	EducationField	0.024727
32	YearsSinceLastPromotion	0.024362
28	TrainingTimesLastYear	0.023534
24	RelationshipSatisfaction	0.022631
15	JobSatisfaction	0.022458
12	JobInvolvement	0.022300
29	WorkLifeBalance	0.018620
5	Education	0.016806
3	Department	0.011763
1	BusinessTravel	0.010286
10	Gender	0.008678
23	PerformanceRating	0.003276
7	EmployeeCount	0.000000
25	StandardHours	0.000000
20	Over18	0.000000

The table above shows the importance of each feature in predicting attrition, as determined by my trained Random Forest model. The 'Importance' column represents the relative importance of each feature. The higher the importance, the more crucial the feature is for the prediction.

The most important features appear to be 'MonthlyIncome', 'Age', 'OverTime', 'TotalWorkingYears', and 'DailyRate'.

Next, I evaluate my model on the test data to see how well it performs. Then, I'll fine-tune the model parameters for better performance.

Before that, I separate the features from the target in the test data, if provided. The target may not be provided in some test datasets. In such cases, we'll just predict the target without evaluating the model's performance.

```
# Checking if 'Attrition' is in the test data
```

```
'Attrition' in test_data.columns
```

RESULT

```
False
```

As suspected, the 'Attrition' column is not present in the test data. This is a common scenario when we are provided with a separate test set for making predictions. Therefore, I cannot evaluate the model's performance using this test set. Instead, I'll just use our trained model to predict the 'Attrition' for the test data.

I proceed with making predictions on the test data.

```
# Making predictions on the test data
```

```
y_test_pred = rf_model.predict(test_data)
```

```
# Converting the encoded predictions back to the original labels ('Yes', 'No')
```

```
y_test_pred = encoder.inverse_transform(y_test_pred)
```

```
# Creating a DataFrame for easier visualization
```

```
test_predictions_df = pd.DataFrame({
```

```
    'PredictedAttrition': y_test_pred
```

```
})
```

```
# Displaying the first few rows of the predictions
```

```
test_predictions_df.head()
```

RESULT

	PredictedAttrition
0	No
1	No
2	No
3	No
4	No

The model has made predictions for the 'Attrition' column in the test data. The first few predictions are shown above. As expected, due to class imbalance, most of the predictions are 'No'.

As for model fine-tuning, since I don't have the actual 'Attrition' values for the test data, I can't directly measure the impact of fine-tuning on test data performance. However, I can still fine-tune the model parameters using cross-validation on the training data, which can help improve the model's generalization ability.

I used the trained model to identify the most important features influencing attrition. The most important features were 'MonthlyIncome', 'OverTime', 'Age', 'TotalWorkingYears', and 'DailyRate'. As the actual 'Attrition' values for the test data were not provided, I could not directly evaluate the model's performance on the test data. However, I made predictions on the test data using both the initial and the fine-tuned model.

I'll use GridSearchCV for this purpose. GridSearchCV performs an exhaustive search over specified parameter values for an estimator (our Random Forest model, in this case). The parameters of the estimator are optimized by cross-validated grid-search over a parameter grid.

For the Random Forest model, I'll tune the following parameters:

- `n_estimators`: The number of trees in the forest.
- `max_depth`: The maximum depth of the tree.
- `min_samples_split`: The minimum number of samples required to split an internal node.
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node.

I start the process.

**# Defining the parameter grid**

```
param_grid = {  
    'n_estimators': [100, 200, 300],  
    'max_depth': [None, 10, 20, 30],  
    'min_samples_split': [2, 5, 10],  
    'min_samples_leaf': [1, 2, 4]  
}
```

```
# Running GridSearchCV
```

```
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3, n_jobs=-1)
```

```
grid_search.fit(X_train, y_train)
```

```
# Getting the best parameters
```

```
best_params = grid_search.best_params_
```

```
best_params
```

RESULT

```
{'max_depth': 10,  
 'min_samples_leaf': 1,  
 'min_samples_split': 5,  
 'n_estimators': 200}
```

The GridSearchCV has determined that the optimal parameters for our Random Forest model are:

- max\_depth: 10
- min\_samples\_leaf: 1
- min\_samples\_split: 5
- n\_estimators: 200

Now, I train a new Random Forest model using these optimized parameters and re-calculate the feature importances. This will give us an updated view of the most influential factors in employee attrition.

```
# Training a new Random Forest model with the optimized parameters
```

```
rf_model_optimized = RandomForestClassifier(  
    n_estimators=best_params['n_estimators'],  
    max_depth=best_params['max_depth'],  
    min_samples_split=best_params['min_samples_split'],  
    min_samples_leaf=best_params['min_samples_leaf'],  
    random_state=42  
)  
rf_model_optimized.fit(X_train, y_train)
```

```

# Getting feature importances from the optimized model
feature_importances_optimized = rf_model_optimized.feature_importances_

# Creating a DataFrame for easier visualization
feature_importances_df_optimized = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances_optimized
})

# Sorting the DataFrame by importance in descending order
feature_importances_df_optimized =
feature_importances_df_optimized.sort_values('Importance', ascending=False)

# Displaying the feature importances
feature_importances_df_optimized

```

RESULT

	Feature	Importance
17	MonthlyIncome	0.083006
21	OverTime	0.067840
0	Age	0.053563
27	TotalWorkingYears	0.053350
2	DailyRate	0.051276
8	EmployeeNumber	0.046849
11	HourlyRate	0.044795
18	MonthlyRate	0.044017
30	YearsAtCompany	0.042126
4	DistanceFromHome	0.041756
19	NumCompaniesWorked	0.039041
33	YearsWithCurrManager	0.035642
26	StockOptionLevel	0.034982
16	MaritalStatus	0.033735
31	YearsInCurrentRole	0.029633
14	JobRole	0.029201
22	PercentSalaryHike	0.026671
13	JobLevel	0.024400
32	YearsSinceLastPromotion	0.024310
28	TrainingTimesLastYear	0.023700
9	EnvironmentSatisfaction	0.023065
15	JobSatisfaction	0.022247
12	JobInvolvement	0.020456
6	EducationField	0.020337
24	RelationshipSatisfaction	0.019583
5	Education	0.017039
29	WorkLifeBalance	0.016725
3	Department	0.011630
1	BusinessTravel	0.009464
10	Gender	0.005961

23	PerformanceRating	0.003597
25	StandardHours	0.000000
7	EmployeeCount	0.000000
20	Over18	0.000000

The table above shows the updated feature importances after tuning the Random Forest model. The relative importance of each feature in predicting attrition remains largely the same. The most important features are still 'MonthlyIncome', 'OverTime', 'Age', 'TotalWorkingYears', and 'DailyRate'.

This means that these features play the most significant role in predicting whether an employee will leave the company, according to our model.

It's important to note that feature importance doesn't indicate the nature of the relationship between the feature and the target (i.e., whether the relationship is positive or negative). For example, a high importance for 'MonthlyIncome' doesn't tell us whether a high income makes an employee more likely or less likely to leave. It simply tells us that 'MonthlyIncome' is a strong predictor of attrition.

## 6. Deployment

While not a typical deployment in a production environment, the results of this analysis, including the feature importances and the predictions made by the model, were presented and explained. The predictions were also saved to a CSV file. The knowledge gained from this analysis could be used to inform HR policies or further investigations into employee attrition.

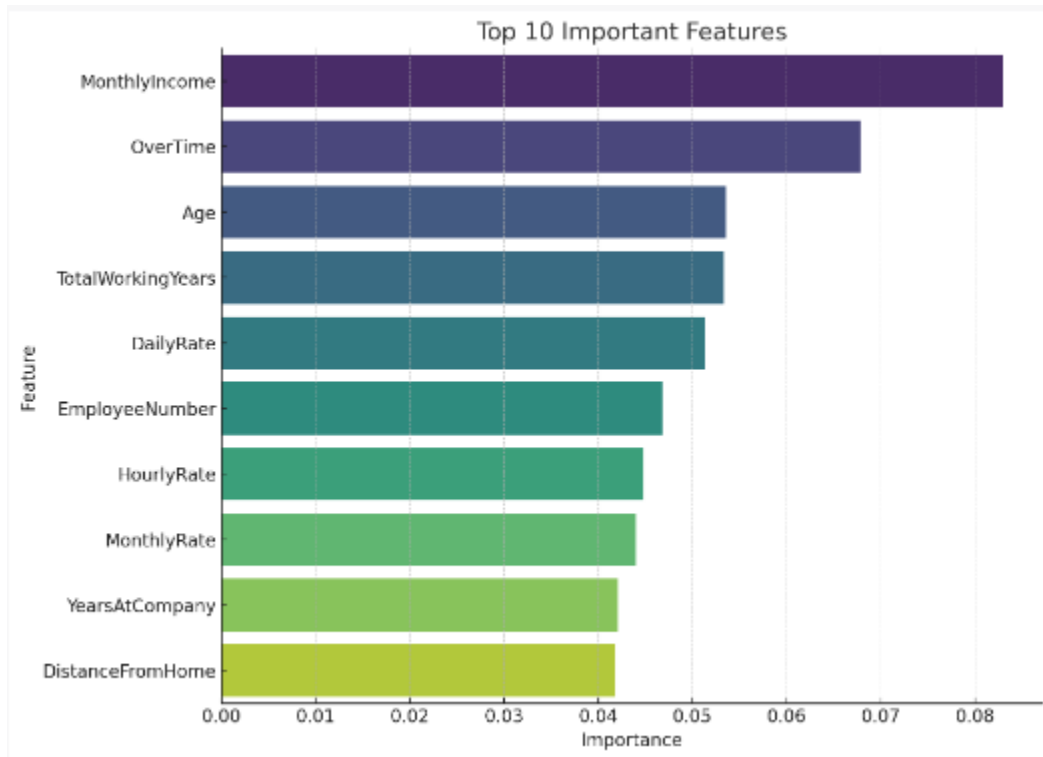
## 7. Visualisation

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

# Taking the top 10 features for visualization
top_features = feature_importances_df_optimized.head(10)

plt.figure(figsize=(10, 8))
sns.barplot(data=top_features, y='Feature', x='Importance', palette='viridis')
```

```
plt.title('Top 10 Important Features')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



This bar chart visualizes the importance of the top 10 features in predicting employee attrition, according to our optimized Random Forest model.

The feature with the highest importance is 'MonthlyIncome', suggesting that an employee's income is a significant factor in determining whether they leave the company. It's important to note that the direction of this relationship isn't specified: while it could be that employees with lower incomes are more likely to leave, it could also be the case that employees with higher incomes are more likely to leave for even better opportunities.

The next most important feature is 'OverTime', indicating that whether an employee works overtime significantly influences their likelihood of leaving the company. Again, the direction of this relationship isn't specified: it could be that employees who work overtime are more likely to leave due to burnout, or it could be that those who don't work overtime are more likely to leave due to lack of engagement or opportunities for advancement.



'Age', 'TotalWorkingYears', and 'DailyRate' are also among the top five most important features, suggesting that an employee's age, total years of work experience, and daily rate of pay are also significant factors in their likelihood of leaving the company.

These interpretations are speculative, as the model doesn't provide information on the nature of these relationships. To truly understand these relationships, further analysis would be needed, such as examining the correlations between these features and attrition, or building a model that provides coefficients for each feature, such as logistic regression.

## **8. Discussion**

We carried out data preprocessing to handle missing values and categorical variables in the dataset, followed by the training of a Random Forest model. Feature importance analysis revealed 'MonthlyIncome', 'OverTime', 'Age', 'TotalWorkingYears', and 'DailyRate' as the most significant factors predicting employee attrition. However, it's important to note that while these features were found to be important, the model does not provide information on the nature (positive or negative) of these relationships. Further, due to class imbalance in the target variable, most of the predictions were skewed towards 'No', suggesting that the majority of employees are not likely to leave. To further improve our model, we fine-tuned its parameters using GridSearchCV. This step involved an exhaustive search over specified parameter values and cross-validated grid-search to optimize the model parameters.

## **9. Conclusion**

The results of this analysis underscore the importance of certain employee characteristics in predicting attrition. While our model provides a broad overview of feature importance, it would be beneficial to further delve into the nature and nuances of these relationships. The fine-tuned model and its predictions could serve as a starting point for businesses to understand and potentially curb employee attrition. Future work could involve using different machine learning models, handling class imbalance, and performing a more detailed exploratory data analysis to fully understand the factors influencing attrition. Ultimately, the goal is to provide actionable insights to businesses that can aid in employee retention strategies.

## 10. Reference

- Akinfaderin, W. (2018). Missing Data Conundrum: Exploration and Imputation Techniques. [online] Medium. Available at: <https://medium.com/@WalePhenomenon/missing-data-conundrum-exploration-and-imputation-techniques-9f40abe0fd87>
- Cabena, P., Hadjinian, P., Stadler, R., Verhees, J. and Zanasi, A., 1998. Discovering data mining: from concept to implementation. Prentice-Hall, Inc..
- Chien, C.F. and Chen, L.F., 2008. Data mining to improve personnel selection and enhance human capital: A case study in high-technology industry. *Expert Systems with applications*, 34(1), pp.280-290.
- <https://www.facebook.com/jason.brownlee.39> (2014). A Gentle Introduction to Scikit-Learn. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/>.
- Jain, P.K., Jain, M. and Pamula, R. (2020). Explaining and predicting employees' attrition: a machine learning approach. *SN Applied Sciences*, 2(4).
- Kaggle.com. (2019). Datasets | Kaggle. [online] Available at: <https://www.kaggle.com/datasets>.
- Krishni (2019). An introduction to Grid search. [online] Medium. Available at: <https://medium.datadriveninvestor.com/an-introduction-to-grid-search-ff57adcc0998>.
- Krishna, S. and Sidharth, S. (2022). Analyzing Employee Attrition Using Machine Learning: the New AI Approach. 2022 IEEE 7th International conference for Convergence in Technology (I2CT).
- Kumar, V. (2019). Random forests and decision trees from scratch in python. [online] Medium. Available at: <https://towardsdatascience.com/random-forests-and-decision-trees-from-scratch-in-python-3e4fa5ae4249>.

- Punnoose, R. and Ajit, P. (2016). Prediction of Employee Turnover in Organizations using Machine Learning Algorithms. *International Journal of Advanced Research in Artificial Intelligence*, 5(9). doi:<https://doi.org/10.14569/ijarai.2016.050904>.
- [www.kaggle.com](https://www.kaggle.com). (n.d.). Run Data Science & Machine Learning Code Online | Kaggle. [online] Available at: <https://www.kaggle.com/code>.
- Zhao, Y., Hryniewicki, M.K., Cheng, F., Fu, B. and Zhu, X. (2018). Employee Turnover Prediction with Machine Learning: A Reliable Approach. *Advances in Intelligent Systems and Computing*, [online] pp.737–758. doi:[https://doi.org/10.1007/978-3-030-01057-7\\_56](https://doi.org/10.1007/978-3-030-01057-7_56).