# Predicting Employee Attrition Using Machine Learning Algorithms

## Table of Contents

## 1.0 Executive Summary

Employee attrition has always been a challenge to companies. The cost of losing an employee is expensive. In general, the replacement cost for an entry-level employee is between 30% and 50% of their annual salary; for a middle-level employee, it is 1.5 times and above their annual salary; and for a high-level employee, it is about 4 times their annual salary. Over the past few years, companies have been investing in a range of people analytics solutions. One of the main reasons was to address the employee retention challenges and to improve organizational decision-making to gain a competitive advantage.

This project aims to analyze an HR dataset provided by IBM analytics to identify the key factors that may cause an employee to leave a company and to predict whether an employee will leave the company. In order to achieve the goal, a range of machine learning algorithms has been used for the analysis. Random Forest is selected as the best model due to its highest accuracy in predicting employee attrition. The results show that average working hours, monthly income, and age are the 3 key factors of employee attrition. Based on the analysis and prediction, the company can take appropriate actions to address the at-risk employees proactively to reduce the potential costs.

Recommendations are provided at the end based on the results of the analysis, such as optimizing the average work hours per day by identifying work priorities and designing an effective work schedule, rewarding the employees based on their achievements to make them feel appreciated, and implementing innovative strategies to invest in the future of young talents. In conclusion, using machine learning algorithms in people analytics can help companies uncover new statistical patterns on their employees effectively, thus improving the overall performance.

# 2.0 Introduction

## 2.1 Background

HR tech is one of the fastest-growing sectors and has been accelerated by the COVID-19 pandemic. According to Statista (2021), the HR tech companies outperformed the traditional HR companies in the stock market in 2020. This industry is expected to grow quickly over the next few years around the world. In 2019, HR tech's overall revenue was estimated at 47.4 billion and was expected to reach 90 billion U.S. dollars by 2025.

There are five segments of HR tech, which include:

- Recruitment
- Personnel management and payroll
- Engagement and connectivity
- Recognizing and rewarding
- Learning and development

Among all the five segments, recruitment and personnel management and payroll are the most established segments that generate more than 80% of the HR tech market. Meanwhile, research shows that the North America region generates the most revenue in the HR tech market, which is most likely due to the association with Silicon Valley; Asia-Pacific is expected to be the fastest growing region, for example, the business environment that favors domestic companies in China accelerates the growth of HR tech (Statista, 2021).

Meanwhile, people analytics has become the lifeblood of HR Tech. Many companies have started to invest in people analytics technologies and solutions to gain better insights into employee attrition and workforce planning over the past few years (Lesser & DeBellis, 2022). Companies can make good decisions and gain a competitive advantage by understanding all workforce elements through applying statistics and storytelling techniques. According to Bennett (2015), Google applies analytics to gain insights into the source of

employment and the impact of each interview; Facebook, Pfizer, and AOL apply analytics to find out the factors that correlate with talent retention; and BP applies analytics to assess its training.

## 2.2 Problem Statement

Employee attrition has always been a challenge to companies. Many believe that the attrition of employees can lead to a high cost to the company, which ultimately affects its competitive advantage in the market. In addition to cost, employee attrition can also lead to indirect impacts on a company (Mainkar, 2018), such as:

**Team Dynamics**

As most employees work in teams, losing a team member may disrupt the smooth functioning of a team and reduce overall productivity.

**Employee Morale**

Employees who leave the company negatively impact their co-workers' morale and decrease overall engagement levels.

**Extended Productivity**

When an experienced employee is replaced by others, it is hard for the company to maintain the same level of productivity as before due to the unfamiliarity of the role for the new employee.

**Culture**

During the period when new employees adapt to the company's culture, there might be collaboration issues among the co-workers.

**Acquired Knowledge**

Losing an employee means losing acquired knowledge. For example, an employee who has spent a long time on a project is not easy to be replaced by others with the same years of experience, thus causing a delay in the project.

People are the greatest assets for any company. Finding talented and experienced employees is complex, but replacing these employees is even more complex. Thus, companies recognize the need to identify the key factors that may cause an employee to leave and predict whether an employee will leave so that actions can be taken to reduce the employee attrition rate in a proactive way instead of a reactive way. By optimizing people through the implementation of effective strategies for employee retention, companies can ultimately increase their revenue and reputation in the market.

### 2.3 Objectives and Measurement

The objectives of this project are to identify the key factors that may cause employee attrition and to predict whether an employee will leave the company in the bio/pharma industry. In response to the analysis, recommendations are provided to the company in order to reduce the employee attrition rate, thus cutting down Human Resource costs such as departure, recruitment, and training, avoiding loss of employee productivity associated with the time of hiring and training of new employees, and preventing the company's proprietary information from being shared with its competitors. To achieve the objectives, the analysis is conducted based on four analytics levels as below:

**Descriptive (what's happened)**

Collect employee attrition-related data to find out whether there is any trend for why the employees are leaving, what job role the employees are working in, how many years the employees have been working with the company, and more.

**Diagnostic (why did it happen)**

Based on the information obtained from the previous level, some conclusions about the situation might be made. Through further analytics, the data can be explored in different dimensions to find out new correlations and factors that lead to employee attrition with a certain level of confidence.

**Predictive (what will happen)**

Based on the past data, use different models to find out what the future situation will look like if all factors contributing to employee attrition remain unchanged or if only one of these factors is changed.

**Prescriptive (what should be done)**

Provide recommendations based on the results from the analysis to improve the current situation. The management can make informed decisions at this level to reduce employee attrition.

The employee attrition rate is the primary measurement for the success of this project. After adopting the recommendations provided from the analysis, the overall employee attrition rate of the company is expected to reduce. For example, according to a study conducted by AON (2015), the turnover rate for bio/pharma companies is 9.4%. As the company's attrition rate in 2015 was about 16%, the targeted employee attrition rate can be set to less than 10%. Other measurements or key metrics can be:

- Number of employees
- Retention of high-performance employees
- Job satisfaction/employee happiness
- Job involvement
- Training time
- New hire satisfaction rate
- Cost of turnover
- Attrition rate by position or manager

## 2.4 Assumptions and Limitations

In order to prevent compromising the integrity of the project, it is necessary to identify the assumptions and limitations. Assumptions can make the models less complicated

than the real world, and limitations can affect the model results and capabilities.

**Assumptions**

- Resources in terms of tools and expertise are sufficient for the project
- The project cost is within the planned budget
- The model developed will be validated and monitored strictly based on the risk management framework
- The model performance will be re-evaluated over time, and action will be recommended based on the risk tiering guideline
- The analysis is only applicable to the bio/pharma industry
- The analysis only focuses on the initially identified objectives
- All employee attritions are regrettable that bring negative impact to the company
- All values in the dataset are accurate and reliable
- No additional features will be added during the analysis

**Limitations**

- The dataset only represents the status quo of 2015
- The dataset only represents India due to the MonthlyIncome in Indian Rupee
- Limited time to optimize the models for better results
- Limited sample size for the analysis
- Limited computer speed for more complex models

# 3.0 Data Source

## 3.1 Dataset Introduction

The dataset in this project is an HR survey dataset provided by IBM Analytics and downloaded from Kaggle.com (https://www.kaggle.com/datasets/vjchoudhary7/hr-analytics-case-study?select=general_data.csv). To help the company focus on understanding the main factors that trigger employees to leave the company, a set of classification models are developed to identify the controllable factors which can lower employee attrition. This dataset has 31 variables, 12 numerical and 19 categorical, and 4410 sample observations, of which about 16% of the employees are labeled as attrition. In addition, the dataset includes different aspects of the workplace, such as personal information about the employee (gender, age, social status, marital status, education, and more), compensation and benefits offered by the company, satisfaction with the workplace, employee's work attitude, working relationship between manager and subordinates, career development opportunities, skillset training times, and more.

## 3.2 Data Dictionary

This dataset has 31 variables, including 12 numerical features, 18 categorical features, and 1 binary target variable:

| Variables | Data Type | Variable Description | Categories |
|---|---|---|---|
| Attrition | Target Variable | Whether the employee left the company or not | |
| Age | Numerical | Age of the employee | |
| BusinessTravel | Categorical | How frequently do the employees travel for business purposes | 1 Non-Travel |
| | | | 2 Travel_Frequently |
| | | | 3 Travel_Rarely |
| Department | Categorical | Department that employee worked for in the company | 1 Human Resources |
| | | | 2 Research & Development |
| | | | 3 Sales |
| DistanceFrom | Numerical | Distance from office to home in | |

| Variables | Data Type | Variable Description | Categories |
|---|---|---|---|
| Home | | kilometers | |
| Education | Categorical | Education level of employee | 1 Below College |
| | | | 2 College |
| | | | 3 Bachelor |
| | | | 4 Master |
| | | | 5 Doctor |
| Education Field | Categorical | Field of education | 1 Human Resources |
| | | | 2 Life Sciences |
| | | | 3 Marketing |
| | | | 4 Medical |
| | | | 5 Other |
| | | | 6 Technical Degree |
| Employee Count | Categorical | Count of employee | |
| Employee Number | Categorical | Employee ID number | |
| Environment Satisfaction | Categorical | Work environment satisfaction level (scale of 1 to 4) | 1 Low |
| | | | 2 Medium |
| | | | 3 High |
| | | | 4 Very High |
| Gender | Categorical | Gender of employee | 0 Female |
| | | | 1 Male |
| Job Involvement | Categorical | Job involvement level (scale of 1 to 4) | 1 Low |
| | | | 2 Medium |
| | | | 3 High |
| | | | 4 Very High |
| JobLevel | Categorical | Job level at the company (scale of 1 to 5) | |
| JobRole | Categorical | Name of job role in the company | 1 Healthcare Representative |
| | | | 2 Human Resources |
| | | | 3 Laboratory Technician |
| | | | 4 Manager |
| | | | 5 Manufacturing Director |
| | | | 6 Research Director |
| | | | 7 Research Scientist |
| | | | 8 Sales Executive |
| | | | 9 Sales Representative |

| Variables | Data Type | Variable Description | Categories |
|---|---|---|---|
| JobSatisfaction | Categorical | Job satisfaction level (scale of 1 to 4) | 1 Low |
| | | | 2 Medium |
| | | | 3 High |
| | | | 4 Very High |
| MaritalStatus | Categorical | Marital status of the employee | 1 Divorced |
| | | | 2 Married |
| | | | 3 Single |
| Monthly Income | Numerical | Monthly income in rupees per month | |
| Num Companies Worked | Numerical | Total number of companies the employee has worked for | |
| Over18 | Categorical | Whether the employee is above 18 years of age or not | |
| PercentSalary Hike | Numerical | Percent salary hike for last year | |
| Performance Rating | Categorical | Performance rating of the employee for last year (scale of 1 to 4) | 1 Low |
| | | | 2 Good |
| | | | 3 Excellent |
| | | | 4 Outstanding |
| StandardHours | Categorical | Standard hours of work for the employee | |
| StockOption Level | Categorical | Stock options level of the employee | |
| in_time | Numerical | Employee clock-in time | |
| out_time | Numerical | Employee clock-out time | |
| TotalWorking Years | Numerical | Total number of years the employee has worked so far | |
| TrainingTimes LastYear | Numerical | Number of times training received last year | |
| WorkLife Balance | Categorical | Work-life balance level (scale of 1 to 4) | 1 Bad |
| | | | 2 Good |
| | | | 3 Better |
| | | | 4 Best |
| YearsAt Company | Numerical | Total number of years spent at the company by the employee | |
| YearsSince LastPromotion | Numerical | Number of years since last promotion | |
| YearsWith CurrManager | Numerical | Number of years with the current manager | |

### 3.3 Initial Data Preparation

Generally, companies have data well managed in the on-premises databases, and different types of data are stored in different schemas or tables. The information contained in this dataset is stored in five separate CSV files:

- **general_data.** This file contains all basic employee information, including personal and work-related data.
- **employee_survey_data.** This file contains the survey feedback from the employees regarding workplace satisfaction.
- **manager_survey_data.** This file contains the survey feedback from the managers regarding employee work performance.
- **in_time.** This file contains the employee clock-in time from January 1 to December 31 in 2015, excluding all the weekends.
- **out_time.** This file contains the employee clock-out time from January 1 to December 31 in 2015, excluding all the weekends.

In order to build models and analyze data effectively, the above five tables should be merged by using EmployeeID as the primary key into one well-formatted DataFrame using Python before performing exploratory data analysis.

**Step 1:** Check the data structure of the in_time table. Two issues are found in this table: the first column is missing a valid field name, and the clock-in data are organized in a wide data structure.

```
[ ] intime.head()
```

| | Unnamed: 0 | 2015-01-01 | 2015-01-02 | 2015-01-05 | 2015-01-06 | 2015-01-07 | 2015-01-08 | 2015-01-09 | 2015-01-12 | 2015-01-13 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 2015-01-02 09:43:45 | 2015-01-05 10:08:48 | 2015-01-06 09:54:26 | 2015-01-07 09:34:31 | 2015-01-08 09:51:09 | 2015-01-09 10:09:25 | 2015-01-12 09:42:53 | 2015-01-13 10:13:06 |
| 1 | 2 | NaN | 2015-01-02 10:15:44 | 2015-01-05 10:21:05 | NaN | 2015-01-07 09:45:17 | 2015-01-08 10:09:04 | 2015-01-09 09:43:26 | 2015-01-12 10:00:07 | 2015-01-13 10:43:29 |
| 2 | 3 | NaN | 2015-01-02 10:17:41 | 2015-01-05 09:50:50 | 2015-01-06 10:14:13 | 2015-01-07 09:47:27 | 2015-01-08 10:03:40 | 2015-01-09 10:05:49 | 2015-01-12 10:03:47 | 2015-01-13 10:21:26 |
| 3 | 4 | NaN | 2015-01-02 10:05:06 | 2015-01-05 09:56:32 | 2015-01-06 10:11:07 | 2015-01-07 09:37:30 | 2015-01-08 10:02:08 | 2015-01-09 10:08:12 | 2015-01-12 10:13:42 | 2015-01-13 09:53:22 |
| 4 | 5 | NaN | 2015-01-02 10:28:17 | 2015-01-05 09:49:58 | 2015-01-06 09:45:28 | 2015-01-07 09:49:37 | 2015-01-08 10:19:44 | 2015-01-09 10:00:50 | 2015-01-12 10:29:27 | 2015-01-13 09:59:32 |

5 rows × 262 columns

**Step 2:** Add a column name to the first column as Employee ID.

```
[ ]  # Add a column name to the first column (EmployeeID)

     intime = intime.rename({'Unnamed: 0':'EmployeeID'}, axis=1)
     intime.head()
```

| | EmployeeID | 2015-01-01 | 2015-01-02 | 2015-01-05 | 2015-01-06 | 2015-01-07 |
|---|---|---|---|---|---|---|
| **0** | 1 | NaN | 2015-01-02 09:43:45 | 2015-01-05 10:08:48 | 2015-01-06 09:54:26 | 2015-01-07 09:34:31 |
| **1** | 2 | NaN | 2015-01-02 10:15:44 | 2015-01-05 10:21:05 | NaN | 2015-01-07 09:45:17 |
| **2** | 3 | NaN | 2015-01-02 10:17:41 | 2015-01-05 09:50:50 | 2015-01-06 10:14:13 | 2015-01-07 09:47:27 |

**Step 3:** Transform the data structure from wide to long. After changing the data structure, the .info() function is used to get a summary of the transposed in_time table. The summary shows that there are missing values in the inTime column. As these missing values represent the absence of work, no further treatment needs to be conducted.

```
[ ]  # Transpsoe the date features from horizontal to vertical

     intime = intime.melt(id_vars=['EmployeeID'],
                          var_name='Date',
                          value_name='inTime')
     intime.tail()
```

| | EmployeeID | Date | inTime |
|---|---|---|---|
| **1151005** | 4406 | 2015-12-31 | 2015-12-31 10:00:12 |
| **1151006** | 4407 | 2015-12-31 | 2015-12-31 10:09:48 |
| **1151007** | 4408 | 2015-12-31 | 2015-12-31 10:03:30 |
| **1151008** | 4409 | 2015-12-31 | 2015-12-31 09:56:47 |
| **1151009** | 4410 | 2015-12-31 | 2015-12-31 10:09:28 |

```
[ ]  intime.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 1151010 entries, 0 to 1151009
     Data columns (total 3 columns):
      #   Column      Non-Null Count    Dtype
     ---  ------      --------------    -----
      0   EmployeeID  1151010 non-null  int64
      1   Date        1151010 non-null  object
      2   inTime      1041930 non-null  object
     dtypes: int64(1), object(2)
     memory usage: 26.3+ MB
```

**Step 4:** Repeat step 1 to 3 for the out_time table.

```
[ ] # Repeat the same steps for outtime dataset

    outtime = outtime.rename({'Unnamed: 0':'EmployeeID'},  axis=1)
    outtime.head()
```

| | EmployeeID | 2015-01-01 | 2015-01-02 | 2015-01-05 | 2015-01-06 | 2015-01-07 | 2015-01-08 |
|---|---|---|---|---|---|---|---|
| **0** | 1 | NaN | 2015-01-02 16:56:15 | 2015-01-05 17:20:11 | 2015-01-06 17:19:05 | 2015-01-07 16:34:55 | 2015-01-08 17:08:32 |
| **1** | 2 | NaN | 2015-01-02 18:22:17 | 2015-01-05 17:48:22 | NaN | 2015-01-07 17:09:06 | 2015-01-08 17:34:04 |
| **2** | 3 | NaN | 2015-01-02 16:59:14 | 2015-01-05 17:06:46 | 2015-01-06 16:38:32 | 2015-01-07 16:33:21 | 2015-01-08 17:24:22 |
| **3** | 4 | NaN | 2015-01-02 17:25:24 | 2015-01-05 17:14:03 | 2015-01-06 17:07:42 | 2015-01-07 16:32:40 | 2015-01-08 16:53:11 |
| **4** | 5 | NaN | 2015-01-02 18:31:37 | 2015-01-05 17:49:15 | 2015-01-06 17:26:25 | 2015-01-07 17:37:59 | 2015-01-08 17:59:28 |

5 rows × 262 columns

```
[ ] outtime = outtime.melt(id_vars=['EmployeeID'],
                           var_name='Date',
                           value_name='outTime')
    outtime.tail()
```

| | EmployeeID | Date | outTime |
|---|---|---|---|
| **1151005** | 4406 | 2015-12-31 | 2015-12-31 18:30:41 |
| **1151006** | 4407 | 2015-12-31 | 2015-12-31 16:18:39 |
| **1151007** | 4408 | 2015-12-31 | 2015-12-31 18:08:55 |
| **1151008** | 4409 | 2015-12-31 | 2015-12-31 19:33:45 |
| **1151009** | 4410 | 2015-12-31 | 2015-12-31 16:39:18 |

```
[ ] outtime.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1151010 entries, 0 to 1151009
Data columns (total 3 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   EmployeeID  1151010 non-null  int64 
 1   Date        1151010 non-null  object
 2   outTime     1041930 non-null  object
dtypes: int64(1), object(2)
memory usage: 26.3+ MB
```

**Step 5:** In the transposed in_time and out_time tables, since the EmployeeID columns contain duplicate values, a new unique identifier is created by combining EmployeeID and Date features to perform as the primary key to join these two tables. And, the data type of both inTime and outTime are changed to DateTime.

```
[ ] # Merge intime and outtime datasets using EmployeeID and Date features as primary key

    daytime = pd.merge(intime, outtime, how='left', left_on=['EmployeeID','Date'], right_on=['EmployeeID','Date'])
    daytime.tail()
```

| | EmployeeID | Date | inTime | outTime |
|---|---|---|---|---|
| **1151005** | 4406 | 2015-12-31 | 2015-12-31 10:00:12 | 2015-12-31 18:30:41 |
| **1151006** | 4407 | 2015-12-31 | 2015-12-31 10:09:48 | 2015-12-31 16:18:39 |
| **1151007** | 4408 | 2015-12-31 | 2015-12-31 10:03:30 | 2015-12-31 18:08:55 |
| **1151008** | 4409 | 2015-12-31 | 2015-12-31 09:56:47 | 2015-12-31 19:33:45 |
| **1151009** | 4410 | 2015-12-31 | 2015-12-31 10:09:28 | 2015-12-31 16:39:18 |

```
[ ] daytime.info()

    <class 'pandas.core.frame.DataFrame'>
    Int64Index: 1151010 entries, 0 to 1151009
    Data columns (total 4 columns):
     #   Column      Non-Null Count    Dtype
    ---  ------      --------------    -----
     0   EmployeeID  1151010 non-null  int64
     1   Date        1151010 non-null  object
     2   inTime      1041930 non-null  object
     3   outTime     1041930 non-null  object
    dtypes: int64(1), object(3)
    memory usage: 43.9+ MB
```

```
[ ] # Change Date, inTime and outTime features to datetime format

    daytime['Date'] = pd.to_datetime(daytime.Date, format='%Y-%m-%d')
    daytime['inTime'] = pd.to_datetime(daytime.inTime)
    daytime['outTime'] = pd.to_datetime(daytime.outTime)
```

**Step 6:** Calculate working hours for each employee per day by subtracting inTime from outTime.

```
[ ] # Calculate working hours for each employee per day

    daytime['WorkHrs'] = ((daytime.outTime - daytime.inTime).dt.seconds)/3600
    daytime.tail()
```

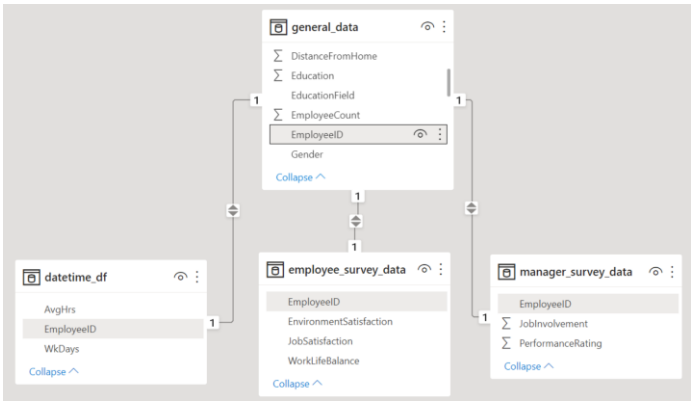
| | EmployeeID | Date | inTime | outTime | WorkHrs |
|---|---|---|---|---|---|
| **1151005** | 4406 | 2015-12-31 | 2015-12-31 10:00:12 | 2015-12-31 18:30:41 | 8.508056 |
| **1151006** | 4407 | 2015-12-31 | 2015-12-31 10:09:48 | 2015-12-31 16:18:39 | 6.147500 |
| **1151007** | 4408 | 2015-12-31 | 2015-12-31 10:03:30 | 2015-12-31 18:08:55 | 8.090278 |

**Step 7:** Calculate each employee's average working hours per day and total working days throughout the year.

```
[ ] # Calculate average working hours and total working days for each employee in the year

    daytime_df = daytime.groupby(['EmployeeID']).agg({'WorkHrs':['mean','count']})
    daytime_df.columns = ['AvgHrs','WkDays']
    daytime_df.head()
```

| | AvgHrs | WkDays |
|---|---|---|
| **EmployeeID** | | |
| **1** | 7.373651 | 232 |
| **2** | 7.718969 | 236 |
| **3** | 7.013240 | 242 |

**Step 8:** Join average working hours, working days, employee survey, and manager survey into the general_data by using EmployeeID as the primary key. And, for the convenience of stakeholders, the currency in the dataset is converted from Indian Rupee to the Canadian dollar at the real-time exchange rate.




```
[ ] # Merge average working hours, working days, employee and manager survey data into employee general dataset

    df = pd.merge(general, daytime_df, how = 'left', left_on=['EmployeeID'], right_on=['EmployeeID'])
    df = pd.merge(df, emp_survey, how = 'left', left_on=['EmployeeID'], right_on=['EmployeeID'])
    df = pd.merge(df, mgr_survey, how = 'left', left_on=['EmployeeID'], right_on=['EmployeeID'])
```

```
[ ] # Get real-time exchange rate for MonthlyIncome (rupees) to Canadian dollar

    c = CurrencyRates()
    rate = c.get_rate('INR', 'CAD')
    # rate = 0.016
```

```
[ ] # Create a new column of MonthlyIncome in Canadian dollar

    df['MonthlyIncomeCAD'] = df.MonthlyIncome * rate
    df.head()
```

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

**Step 9:** The string values in some categorical features are converted to different integers in order to build the models efficiently. The data type of these features remains categorical.

```
[ ] # Change target variable to binary

    df['Attrition'].replace(['No','Yes'], [0,1], inplace=True)

    # Replace text values in categorical features with codes (e.g. 1, 2, 3, etc.)

    df['BusinessTravel'].replace(['Non-Travel','Travel_Frequently','Travel_Rarely'], [1,2,3], inplace=True)
    df['Department'].replace(['Human Resources','Research & Development','Sales'], [1,2,3], inplace=True)
    df['EducationField'].replace(['Human Resources','Life Sciences','Marketing','Medical','Other','Technical Degree'], [1,2,3,4,5,6], inplace=True)
    df['Gender'].replace(['Female','Male'], [0,1], inplace=True)
    df['JobRole'].replace(['Healthcare Representative','Human Resources','Laboratory Technician','Manager','Manufacturing Director','Research Director','Research Scient
    df['MaritalStatus'].replace(['Divorced','Married','Single'], [1,2,3], inplace=True)

    df.head()
```

| | Age | Attrition | BusinessTravel | Department | DistanceFromHome | Education | EducationField | EmployeeCount | EmployeeID | Gender | ... | YearsSinceLastPromotion | YearsWith |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 51 | 0 | 3 | 3 | 6 | 2 | 2 | 1 | 1 | 0 | ... | 0 | |
| 1 | 31 | 1 | 2 | 2 | 10 | 1 | 2 | 1 | 2 | 0 | ... | 1 | |
| 2 | 32 | 0 | 2 | 2 | 17 | 4 | 5 | 1 | 3 | 1 | ... | 0 | |

# 4.0 Data Exploration

## 4.1 Introduction

Data exploration also referred to as exploratory data analysis, is a process of analyzing datasets in an unstructured way to discover the main characteristics, initial patterns, potential problems, and areas of interest (Sisense, n.d.). The goal of data exploration is not to reveal every single piece of information in the datasets but to show an overall picture of important trends and relationships among variables without any hypothesis testing task or formal modeling beforehand. Data exploration has become an essential process in every data analytics project. As a rule of thumb, 80% of the time should be spent on data exploration and preparation, while only 20% of the time should be spent on modeling (Prinsloo, 2022). Some valuable statistical graphics in data exploration include histograms, bar charts, scatter plots, box plots, pair plots, heat maps, and more.

## 4.2 Data Exploration

**Data Structure**

The dataset consists of 4410 rows (sample observations) and 32 columns (variables).

```
[ ]  # Copy the merged dataset to a new dataframe
     # Check number of rows and columns

     emp_attr = df.copy()
     emp_attr.shape
```

```
(4410, 32)
```

**Duplicates**

There is no duplicate sample observation in the dataset.

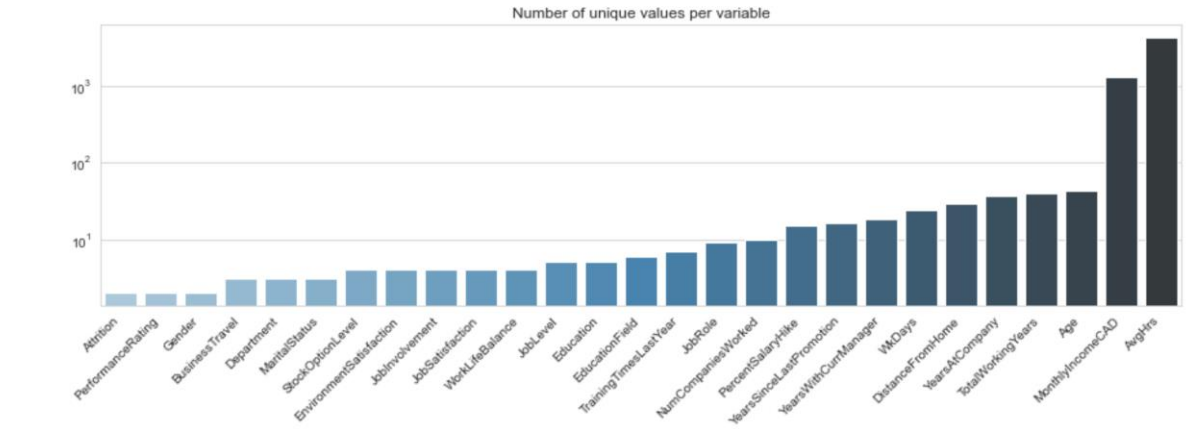
```
[ ]  # Check duplicates

     emp_attr.duplicated().any()
```

```
False
```

## Data Uniqueness

Knowing the number of unique values in each variable gives insights into the number of categorical and numerical variables in the dataset. Based on the business context, variables with 10 or below unique values can be considered categorical, except for the TrainingTimesLastYear and NumCompaniesWorked. The categorical variables are BusinessTravel, Department, Education, EducationField, Gender, JobLevel, JobRole, MaritalStatus, StockOptionLevel, EnvironmentSatisfaction, JobSatisfaction, WorkLifeBalance, JobInvolvement, and PerformanceRating.

```
[ ]  # Visualize the number of unique values

     unique_values = emp_attr.nunique().sort_values()
     plt.figure(figsize=(15,4))
     sns.set_style('whitegrid')

     g = sns.barplot(x=unique_values.index, y=unique_values,palette='Blues_d')
     g.set_yscale('log')
     g.set_xticklabels(g.get_xticklabels(), rotation=45, horizontalalignment='right')
     g.set_title('Number of unique values per variable')
     plt.show()
```

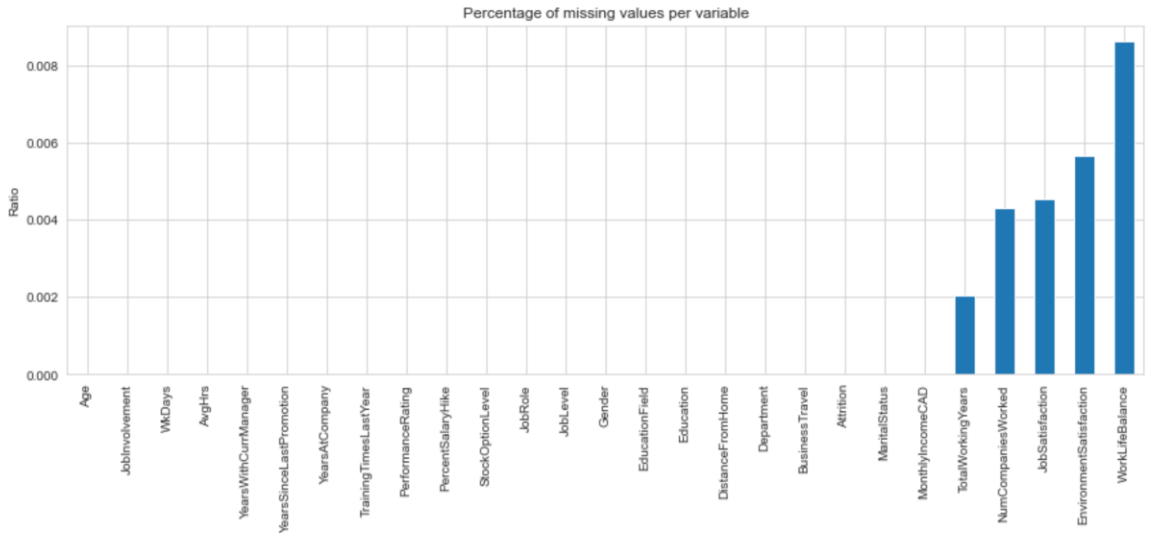



## Missing Values

There are some missing values in TotalWorkingYears, NumCompaniesWorked, JobSatisfaction, EnvironmentSatisfaction, and WorkLifeBalance. However, the percentage of missing values in each feature is less than 1%. Refer to section 5.3 for the imputation of missing values.

```
[ ]  # Calculate the percentage of missing values

     emp_attr.isna().mean().sort_values().plot(
         kind='bar', figsize=(15,5),
         title='Percentage of missing values per variable',
         ylabel='Ratio')
```

```
<AxesSubplot:title={'center':'Percentage of missing values per variable'}, ylabel='Ratio'>
```

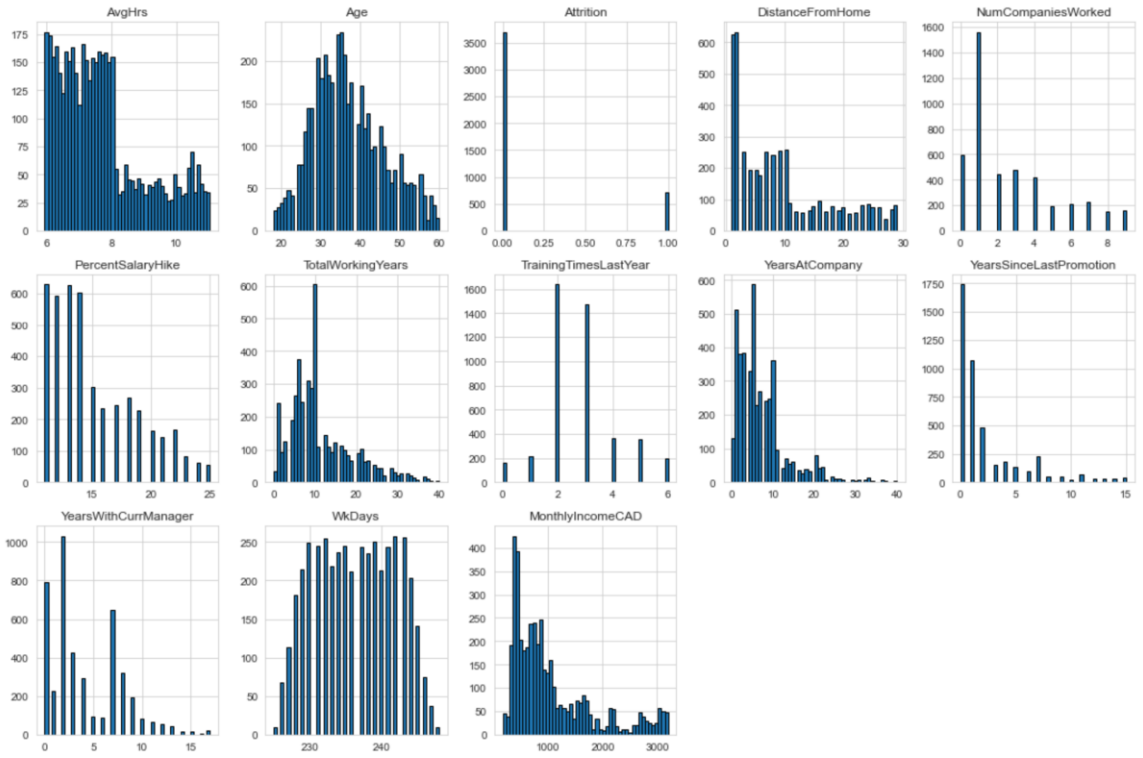



### *4.2.1 Numerical Features*

**Distributions**

The histograms below show that all numerical variables except WkDays, Age, and Attrition have moderately to highly skewed distribution (>0.5), which will be treated by transformation technique in section 5.3 as some models like Logistic Regression and K-Nearest Neighbors are sensitive to extreme values.

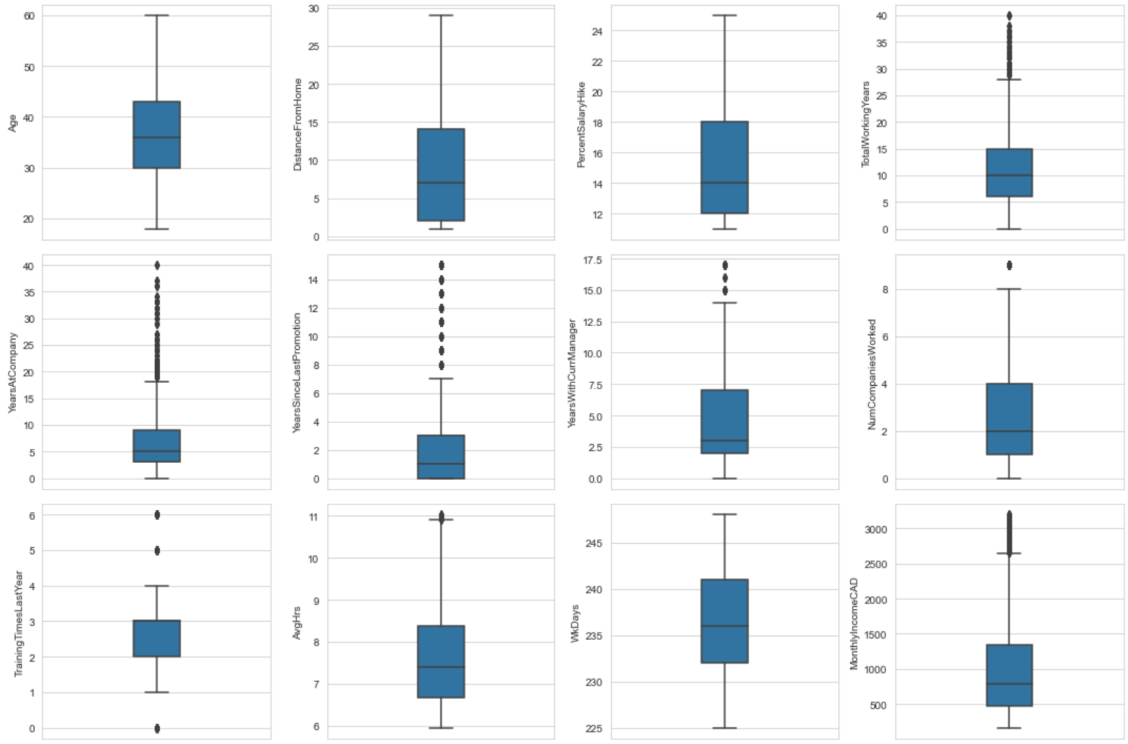
```
[ ]  # Visualize the distribution of each numerical feature

     emp_attr.hist(bins=50, figsize=(15,10), layout=(-1,5), edgecolor='black')
     plt.tight_layout()
```




## Outliers

The boxplots below show that TotalWorkingYears, YearsAtCompany, YearsSinceLastPromotion, YearsWithCurrentManager, NumCompaniesWorked, TrainingTimesLastYear, AvgHrs, MonthlyIncomeCAD have outliers. However, considering the nature of employee characteristics and the business context, these outliers are reasonable, and therefore treatment is unnecessary.

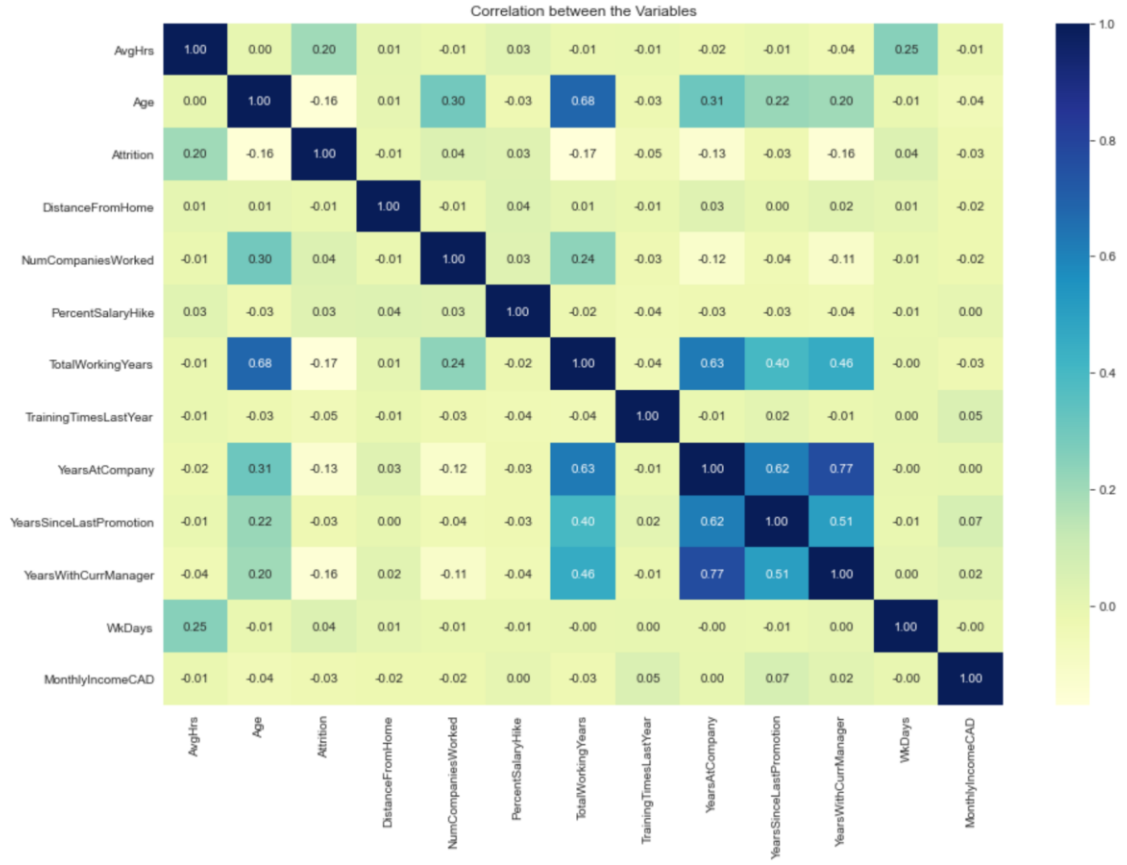
```
[ ]  # Visualize the outliers of each numerical feature

     variables = ['Age','DistanceFromHome','PercentSalaryHike',
                  'TotalWorkingYears','YearsAtCompany','YearsSinceLastPromotion',
                  'YearsWithCurrManager','NumCompaniesWorked','TrainingTimesLastYear',
                  'AvgHrs','WkDays','MonthlyIncomeCAD']
     fig, axes = plt.subplots(3, 4, figsize=(15,10))
     for i, var in enumerate(variables):
       ax = fig.axes[i]
       sns.boxplot(y=var, width=0.2, data=emp_attr, ax=ax, showfliers=True)
     plt.tight_layout()
     plt.show()
```




## Correlations

The heatmap below shows the relationship between numerical variables, both positive and negative correlations. For example, the strong positive correlation between YearsAtCompany and YearWithCurrManager (0.77) predicts that if the employee has spent more years at the company, he/she will have longer years with the current manager.

```
[ ]  # Visualize the correlation between numerical variables with heatmap

     plt.figure(figsize=(15,10))
     with sns.axes_style('darkgrid'):
       sns.heatmap(emp_attr.corr(), cmap='YlGnBu', annot=True, fmt='.2f')
     plt.title('Correlation between the Variables')
     plt.show()
```

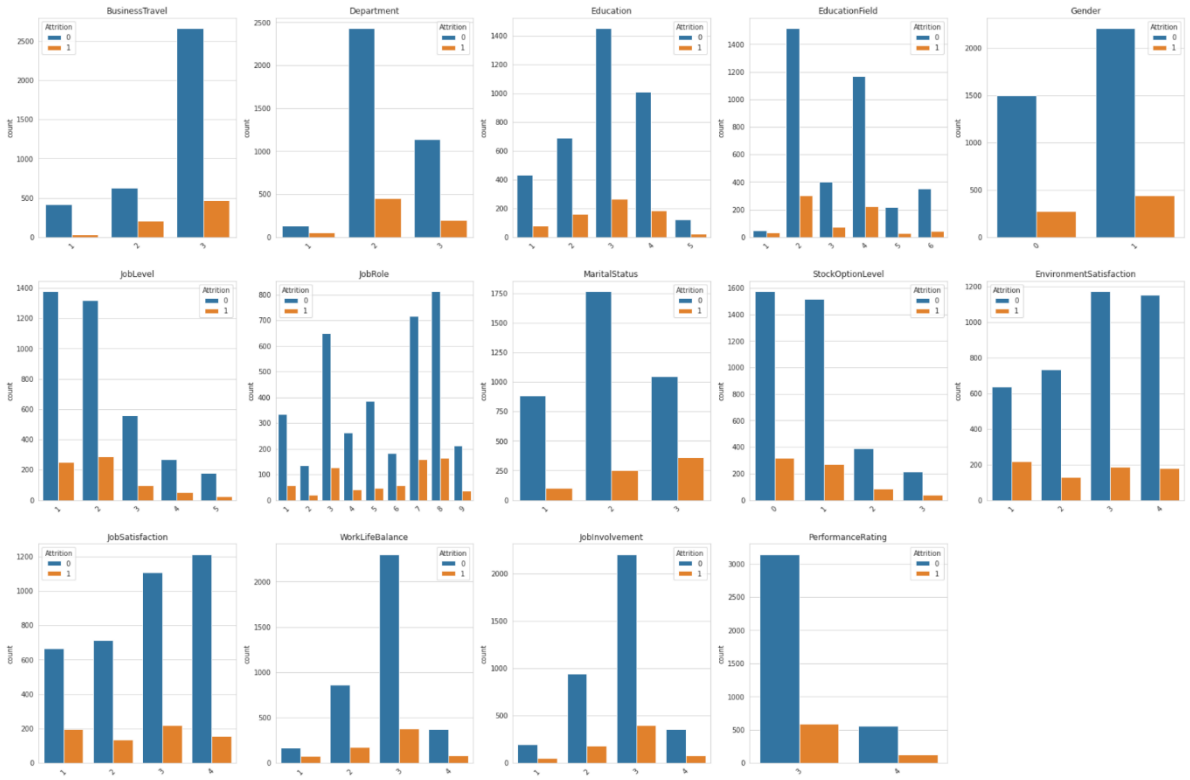



### *4.2.2 Categorical Features*

**Distributions**

The histograms below show that employees with a bachelor's degree, low job level, low stock option, and low-performance rating tend to leave the company. In addition, male employees or single employees are more likely to have a higher attrition rate than their counterparts.

```
[ ]  # Visualize categorical features

     categorical_features = emp_attr.describe(exclude='number').columns.tolist()
     fig, ax = plt.subplots(3, 5, figsize=(30,20))
     for i, var in enumerate(categorical_features):
       ax = fig.axes[i]
       sns.countplot(x=var, hue='Attrition', ax=ax, data=emp_attr).set(title=var, xlabel=None)
       ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
     fig.show()
```




## Correlations

A p-value less than 0.05 indicates a statistical significance. The chi-sqaure tests below show that BusinessTravel, Department, EducationField, JobRole, MaritalStatus, EnvironmentSatisfaction, JobSatisfaction, WorkLifeBalance, JobInvolvement are statistically significant to attrition.

```
[ ]  # Check correlation between categorical features and Attrition

     for i, var in enumerate(categorical_features):
       crosstab_result=pd.crosstab(index=emp_attr[var], columns=emp_attr['Attrition'])
       print(crosstab_result)

       # Perform Chi-squared test
       ChiSqResult = chi2_contingency(crosstab_result)

       print('The P-Value of the Chi-Squared Test is: {:.30f}'.format(ChiSqResult[1]),
             '\n','**********************************************************************','\n')
```

```
Attrition          0    1
BusinessTravel
1                414   36
2                624  207
3               2661  468
The P-Value of the Chi-Squared Test is: 0.000000000000000176427697298319
 **********************************************************************

Attrition      0    1
Department
1            132   57
2           2430  453
3           1137  201
The P-Value of the Chi-Squared Test is: 0.000000482088821817040684207208
 **********************************************************************

Attrition     0    1
Education
1           432   78
2           687  159
3          1449  267
4          1008  186
5           123   21
The P-Value of the Chi-Squared Test is: 0.22759826301782393187522757216 4
 **********************************************************************

Attrition          0    1
EducationField
1                 48   33
2               1515  303
3                402   75
4               1167  225
5                216   30
6                351   45
The P-Value of the Chi-Squared Test is: 0.000000008288917469574178642361
 **********************************************************************

Attrition     0    1
Gender
0          1494  270
1          2205  441
The P-Value of the Chi-Squared Test is: 0.24529482862926826891580844858 3
 **********************************************************************

Attrition     0    1
JobLevel
1          1377  252
2          1317  285
3           558   96
4           267   51
5           180   27
The P-Value of the Chi-Squared Test is: 0.17992768013371859825433318746 9

Attrition    0    1
JobRole
1          336   57
2          135   21
3          651  126
4          264   42
5          387   48
6          183   57
7          717  159
8          813  165
9          213   36
The P-Value of the Chi-Squared Test is: 0.00148554474481526685436527657 2
 **********************************************************************
```

```
Attrition         0    1
MaritalStatus
1               882   99
2              1767  252
3              1050  360
The P-Value of the Chi-Squared Test is: 0.000000000000000000000000000001
 **********************************************************************

Attrition            0    1
StockOptionLevel
0                 1575  318
1                 1518  270
2                  390   84
3                  216   39
The P-Value of the Chi-Squared Test is: 0.38454683657380506200951231221 6
 **********************************************************************

Attrition                    0    1
EnvironmentSatisfaction
1                          636  216
2                          735  129
3                         1173  187
4                         1155  179
The P-Value of the Chi-Squared Test is: 0.000000000000014987466319166133
 **********************************************************************

Attrition           0    1
JobSatisfaction
1                 664  197
2                 715  138
3                1108  220
4                1212  156
The P-Value of the Chi-Squared Test is: 0.000000000033049966243221400118
 **********************************************************************

Attrition           0    1
WorkLifeBalance
1                 166   75
2                 860  175
3                2300  380
4                 373   81
The P-Value of the Chi-Squared Test is: 0.000000000128185908456412556672
 **********************************************************************

Attrition          0    1
JobInvolvement
1                195   54
2                945  180
3               2205  399
4                354   78
The P-Value of the Chi-Squared Test is: 0.043222528373721660899242635878
 **********************************************************************

Attrition            0    1
PerformanceRating
3                 3144  588
4                  555  123
The P-Value of the Chi-Squared Test is: 0.13429324234493711376359215137 2
 **********************************************************************
```

### 4.3 Data Cleansing

All column names and values are in a proper format, and there is no duplicate sample observation in the dataset. Therefore, no data cleansing is required at this stage. Refer to section 5.3 for feature engineering.

### 4.4 Summary

The data exploration stage has explored the data structure, checked for duplicates and data uniqueness, as well as identified missing values. Statistical graphics are used to identify the distribution, outliers, and correlation. The correlation between each categorical feature and Attrition is uncovered through the chi-square test. After exploring the data in multiple dimensions, the next stage will be data preparation and feature engineering. Below is a summary of the data exploration section:

- The dataset consists of 4410 sample observations, with 31 variables, including 12 numerical features, 18 categorical features, and 1 binary target variable.
- The dataset is imbalanced, with 16% of observations labeled as Attrition.
- There are no duplicates in the dataset.
- There are five features with missing values, but the percentage of missing values in each of these features is less than 1%.
- There are eight features with outliers. However, these outliers are reasonable considering the nature of employee characteristics and the business context.
- There is no feature with a moderate to strong correlation with Attrition.
- Employees who have less monthly income show a higher attrition rate.
- Employees with a bachelor's degree, low job level, low stock option, and low-performance ratings tend to leave the company.
- Male employees or single employees are more likely to leave the company.
- Research Scientist and Sales Executive have a higher proposition of leavers.

# 5.0 Data Preparation and Feature Engineering

## 5.1 Introduction

Data preparation, along with feature engineering, is a process of manipulating and transforming the raw data into a ready-made dataset to train the machine learning models, better discover analytics insights, and make further business predictions. Having a well-formatted and high-quality dataset is much more important than building a complex model (Boyer, n.d.).

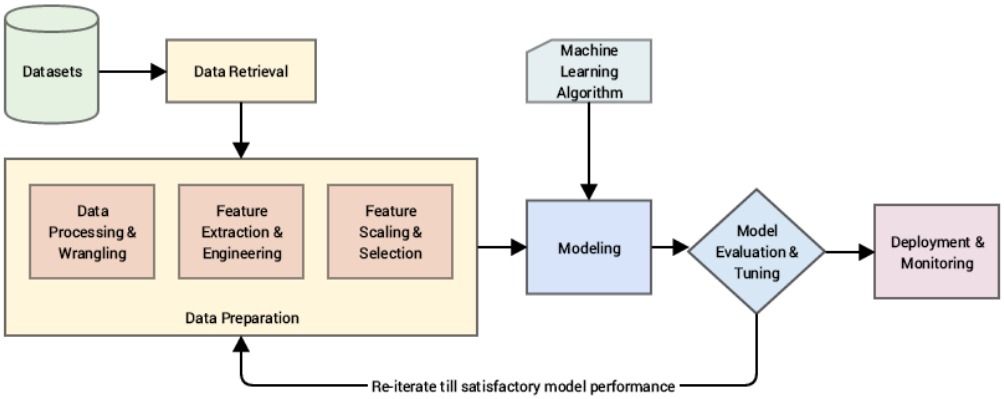



*Figure 1 Process of modeling*

Source: Heavy.AI
Retrieved from: https://www.heavy.ai/technical-glossary/feature-engineering

Generally, most machine learning models have specific requirements on the features or the data format. Thus, based on the different requirements, data preparation and feature engineering steps must be conducted on the dataset before building the model, as Figure 1 shows above. For example, the Logistic Regression model cannot reasonably handle missing values, and if the data contains missing values, the model's performance will be severely affected. In addition, Logistic Regression also requires feature scaling as all the Gradient Descent based algorithms are sensitive to the relative scale of the features. However, the Decision Tree can automatically handle missing values and is not sensitive to the variance in the data.

In section 4.2, data exploration reveals the characteristics of each feature and uncovers some issues of the dataset. Then, based on those findings and the requirements of

the models, eight tasks in the following section are performed to ensure the best data format.

### 5.2 Data Preparation

**Feature Creation**

New derived features are created when they are helpful to the model in providing more accurate predictions through addition, subtraction, multiplication, and division of the existing features. In section 3.3, three new features, AvgHrs, WkDays, and MonthlyIncome_CAD, are created to prepare more relevant data for the models.

```
[ ]  # Calculate average working hours and total working days for each employee in the year

     daytime_df = daytime.groupby(['EmployeeID']).agg({'WorkHrs':['mean','count']})
     daytime_df.columns = ['AvgHrs','WkDays']
     daytime_df.head()
```

```
[ ]  # Get real-time exchange rate for MonthlyIncome (rupees) to Canadian dollar

     c = CurrencyRates()
     rate = c.get_rate('INR', 'CAD')
     # rate = 0.016
```

```
[ ]  # Create a new column of MonthlyIncome in Canadian dollar

     df['MonthlyIncomeCAD'] = df.MonthlyIncome * rate
     df.head()
```

**Feature Selection**

During the feature selection process, all features in the dataset are analyzed and verified to determine whether they are relevant to the target variable. Redundant and irrelevant features are removed from the dataset to avoid any negative impact on the model performance. In this project, six features are removed from the dataset due to:

- EmployeeCount, Over18, and StandardHours are constant as they only contain one unique value
- EmployeeID is irrelevant to the target variable
- MonthlyIncome is replaced by the new feature MonthlyIncome_CAD

- YearsWithCurrManager and YearsAtCompany are highly correlated according to the findings in section 4.2.1. Since YearsWithCurrManager has a higher correlation coefficient with the target variable, it should be removed to avoid multicollinearity

```
[ ]  # Drop unnecessary features
     # EmployeeCount, Over18 and StandardHours should be dropped as they only contain one unique value
     # EmployeeID and MonthlyIncome variables are not useful for the analysis

     emp_attr.drop(['EmployeeCount','Over18','StandardHours','EmployeeID','MonthlyIncome'], axis='columns', inplace=True)
```

```
[ ]  # Remove YearsWithCurrManager feature

     emp_attr = emp_attr.drop('YearsWithCurrManager', axis=1)
```

After feature creation and selection steps, the dataset consists of 26 variables, 11 numerical features, 14 categorical features, and 1 binary target variable.

```
[ ]  emp_attr.info()

     <class 'pandas.core.frame.DataFrame'>
     Int64Index: 4410 entries, 0 to 4409
     Data columns (total 26 columns):
      #   Column                    Non-Null Count  Dtype
     ---  ------                    --------------  -----
      0   AvgHrs                    4410 non-null   float64
      1   Age                       4410 non-null   int32
      2   Attrition                 4410 non-null   int32
      3   BusinessTravel            4410 non-null   category
      4   Department                4410 non-null   category
      5   DistanceFromHome          4410 non-null   int32
      6   Education                 4410 non-null   category
      7   EducationField            4410 non-null   category
      8   Gender                    4410 non-null   category
      9   JobLevel                  4410 non-null   category
      10  JobRole                   4410 non-null   category
      11  MaritalStatus             4410 non-null   category
      12  NumCompaniesWorked        4410 non-null   int32
      13  PercentSalaryHike         4410 non-null   int32
      14  StockOptionLevel          4410 non-null   category
      15  TotalWorkingYears         4410 non-null   int32
      16  TrainingTimesLastYear     4410 non-null   int32
      17  YearsAtCompany            4410 non-null   int32
      18  YearsSinceLastPromotion   4410 non-null   int32
      19  WkDays                    4410 non-null   int32
      20  EnvironmentSatisfaction   4410 non-null   category
      21  JobSatisfaction           4410 non-null   category
      22  WorkLifeBalance           4410 non-null   category
      23  JobInvolvement            4410 non-null   category
      24  PerformanceRating         4410 non-null   category
      25  MonthlyIncomeCAD          4410 non-null   int32
     dtypes: category(14), float64(1), int32(11)
     memory usage: 450.3 KB
```

## Resampling

In classification predictive modeling, imbalanced data may pose challenges to the model since most machine learning algorithms assume by default that classes in the dataset contain the same number of sample observations. As a result, the predicted results may be biased towards the majority class (Brownlee, 2019). According to Google (2022), when the minority class is less than 20% of the overall sample size, this dataset can be considered as moderate to

extreme imbalance. One widely adopted method in handling imbalanced data is resampling, undersampling the majority class and oversampling the minority. In order to prevent bias in modeling toward predictions and to be more accurate in predicting both the majority and the minority classes, this project uses SMOTE (Synthetic Minority Oversampling Technique) and RandomUnderSampler algorithms to balance the data.

SMOTE can randomly select records from the minority class in the dataset and use the K-Nearest Neighbors algorithm to create synthetic data between the selected records and their neighbors. RandomUnderSampler algorithm, on the other hand, can randomly select a subset of the data in the majority class (Analytics Vidhya, 2022).

```
[ ]  # Function to oversample the minority class (Attrition == 1) with SMOTE
     # And undersample the majority class (Attrition == 0) with RandomUnderSampler class

     def sampling(feature, target, isCat):

       # Visualize the Attrition in training set
       colors = ['#66b3ff','#ff9999']
       explode = (0.05, 0.05)
       plt.figure(figsize=(5, 5))
       plt.pie(pd.Series(target).value_counts(), colors=colors, labels=['No','Yes'],
               autopct='%1.1f%%', startangle=90, pctdistance=0.85, explode=explode)
       plt.legend()
       plt.title('Targets in Training Set')
       plt.show()

       print(feature.shape)

       # Oversample and Undersample
       if isCat == True:
         oversampler = SMOTENC(categorical_features=[feature.columns.get_loc(c) for c in cat_list],
                               random_state=1)
       else:
         oversampler = SMOTE(random_state=1)
       undersampler = RandomUnderSampler(random_state=1)
       steps = [('o',oversampler), ('u',undersampler)]
       pipeline = Pipeline(steps=steps)
       sampled_X, sampled_y = pipeline.fit_resample(feature, target)

       # Visualize the Attrition in training set after sampling
       colors = ['#66b3ff','#ff9999']
       explode = (0.05, 0.05)
       plt.figure(figsize=(5, 5))
       plt.pie(pd.Series(sampled_y).value_counts(), colors=colors, labels=['No','Yes'],
             autopct='%1.1f%%', startangle=90, pctdistance=0.85, explode=explode)
       plt.legend()
       plt.title('Sampled Targets in Training Set')
       plt.show()

       print(sampled_X.shape)

       return sampled_X, sampled_y
```

### 5.3 Feature Engineering

**Imputation**

Generally, most machine learning algorithms require a complete dataset to make a prediction.

Therefore, missing values need to be reasonably handled before the data run through models. The most straightforward method is deleting the records containing missing values. However, this solution may lead to information loss and misleading results. In this project, all missing values are imputed and replaced with a reasonable value by using the K-Nearest Neighbors imputation technique, which assumes that the missing value can be approximated by its closest k neighbors (Obadia, 2017).

```
[ ]  # Handle missing values using KNN imputation technique

     imputer = KNNImputer(copy=False)
     imputed = imputer.fit_transform(emp_attr)
     emp_attr = pd.DataFrame(imputed, columns=emp_attr.columns)
```

```
[ ]  # Check missing values after the imputation

     emp_attr.isnull().sum()
```

```
Age                        0
Attrition                  0
BusinessTravel             0
Department                 0
DistanceFromHome           0
Education                  0
EducationField             0
Gender                     0
JobLevel                   0
JobRole                    0
MaritalStatus              0
NumCompaniesWorked         0
PercentSalaryHike          0
StockOptionLevel           0
TotalWorkingYears          0
TrainingTimesLastYear      0
YearsAtCompany             0
YearsSinceLastPromotion    0
YearsWithCurrManager       0
AvgHrs                     0
WkDays                     0
EnvironmentSatisfaction    0
JobSatisfaction            0
WorkLifeBalance            0
JobInvolvement             0
PerformanceRating          0
MonthlyIncomeCAD           0
dtype: int64
```

## Transformation

Skewness indicates that the data distribution is more towards one tail than the other instead of being symmetrical. Transforming the skewed features before training the models that are sensitive to the skewness can result in a significant improvement in model performance. In general, many machine learning algorithms will function better if the data are normally distributed. Box-Cox and Yeo-Johnson are the two transformation techniques that can make

distribution more Gaussian-like (Brownlee, 2020). Due to the inability of Box-Cox to handle zero and negative values, the Yeo-Johnson transformation technique is applied in this project to reduce skewness as it has no restriction on the data range.

```
[ ]  # Check skewness

     emp_attr.skew(axis=0, numeric_only=None).sort_values(ascending=False)
```

```
YearsSinceLastPromotion    1.982939
Attrition                  1.843111
YearsAtCompany             1.763328
MonthlyIncomeCAD           1.368885
TotalWorkingYears          1.116413
NumCompaniesWorked         1.025210
DistanceFromHome           0.957466
AvgHrs                     0.863133
PercentSalaryHike          0.820569
TrainingTimesLastYear      0.552748
Age                        0.413005
WkDays                    -0.001529
dtype: float64
```

```
[ ]  # Apply Yeo-Johnson transformation to reduce skewness

     sk_list = ['YearsSinceLastPromotion','MonthlyIncomeCAD','DistanceFromHome',
                'YearsAtCompany','TotalWorkingYears','NumCompaniesWorked',
                'AvgHrs','PercentSalaryHike','TrainingTimesLastYear']

     pt = PowerTransformer(method='yeo-johnson')
     pt.fit(emp_attr[sk_list])
     sk_trans = pt.transform(emp_attr[sk_list])
     columns = [i + '_Trans' for i in sk_list]
     sk_trans = pd.DataFrame(sk_trans, columns=columns)
     sk_trans
```

```
[ ]  # Check the skewness of skewed features after transformation

     sk_trans.skew(axis=0, numeric_only=None).sort_values(ascending=False)
```

```
YearsSinceLastPromotion_Trans    0.211985
PercentSalaryHike_Trans          0.116171
AvgHrs_Trans                     0.108956
TrainingTimesLastYear_Trans      0.058018
MonthlyIncomeCAD_Trans           0.032903
NumCompaniesWorked_Trans         0.014684
DistanceFromHome_Trans          -0.007463
YearsAtCompany_Trans            -0.008677
TotalWorkingYears_Trans         -0.010675
dtype: float64
```

## Discretization

Binning numerical features with non-standard distribution into discrete categories may potentially reduce variance in the data and better identify the pattern of the features (Engel, 2022). In this project, as additional work, TrainingTimesLastYear is binned into two groups

and treated as a categorical feature instead of a numerical feature to find out whether the model's performance is improved. TrainingTimesLastYear is grouped as below:

| TrainingTimesLastYear | Category |
|---|---|
| 0-2 | 1 |
| 3-6 | 2 |

Both datasets (with or without the binned feature) are passed to a simple Decision Tree model to test whether the feature after binning has higher importance and whether it may impact the model's performance. The overall accuracy of the model with the binned feature is slightly lower than the original one, but there are no significant differences. The importance of the binned feature also has a slight decrease. Therefore, the original feature is selected over the binned feature in this case.

```
[ ]  # Copy the original dataset to a new dataframe for testing
     # Bin TrainingTimesLastYear feature into two categories

     emp_attr_bin = emp_attr.copy()
     emp_attr_bin['TrainingTimesLastYear_Bin'] = np.where(emp_attr_bin['TrainingTimesLastYear']<=2, '1','2')
     emp_attr_bin = emp_attr_bin.drop(columns = ['TrainingTimesLastYear'])
     emp_attr_bin.head()
```

```
[ ]  # Check the importance of TrainingTimesLastYear_Bin (categorical) and
     # TrainingTimesLastYear (numerical) features with Decision Tree

     X = emp_attr_bin.drop(columns=['Attrition'])
     y = emp_attr_bin.Attrition

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)

     clf = DecisionTreeClassifier(random_state=11)
     clf.fit(X_train,y_train)
```

```
DecisionTreeClassifier(random_state=11)
```

```
[ ]  summary('bin',clf, y_train, X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0  498
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9645)

       Prediction
Actual    0    1
     0 1085   25
     1   22  191

Precision Score:  0.88425926
Recall Score:  0.89671362
F1 Score:  0.89044289
```

```
[ ] X = emp_attr.drop(columns=['Attrition'])
    y = emp_attr.Attrition

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)

    clf = DecisionTreeClassifier(random_state=11)
    clf.fit(X_train,y_train)

    DecisionTreeClassifier(random_state=11)
```

```
[ ] summary('nobin',clf, y_train, X_train, y_test, X_test)

    Train classification summary:
    Confusion Matrix (Accuracy 1.0000)

           Prediction
    Actual    0    1
         0 2589    0
         1    0  498
    ------------------------------------------------------------------------

    Test classification summary:
    Confusion Matrix (Accuracy 0.9683)

           Prediction
    Actual    0    1
         0 1093   17
         1   25  188

    Precision Score:  0.91707317
    Recall Score:  0.88262911
    F1 Score:  0.89952153
```

```
[ ] # Check the importance of each feature

    importance = clf.feature_importances_
    df = pd.DataFrame({'feature':X.columns,'importance':importance})
    print(df.sort_values('importance', ascending=False))
```

| | feature | importance |
|---|---|---|
| 0 | AvgHrs | 0.142058 |
| 14 | TotalWorkingYears | 0.086011 |
| 24 | MonthlyIncomeCAD | 0.084806 |
| 1 | Age | 0.082262 |
| 19 | EnvironmentSatisfaction | 0.057868 |
| 10 | MaritalStatus | 0.049607 |
| 12 | PercentSalaryHike | 0.043884 |
| 16 | YearsAtCompany | 0.042512 |
| 21 | WorkLifeBalance | 0.041779 |
| 6 | EducationField | 0.037822 |
| 11 | NumCompaniesWorked | 0.037386 |
| 9 | JobRole | 0.035114 |
| 4 | DistanceFromHome | 0.032867 |
| 5 | Education | 0.031050 |
| 13 | StockOptionLevel | 0.029711 |
| 15 | TrainingTimesLastYear | 0.026832 |
| 8 | JobLevel | 0.025028 |
| 20 | JobSatisfaction | 0.023889 |
| 3 | Department | 0.020587 |
| 17 | YearsSinceLastPromotion | 0.017856 |
| 7 | Gender | 0.017502 |
| 22 | JobInvolvement | 0.014285 |
| 25 | TrainingTimesLastYear_Bin | 0.013415 |
| 2 | BusinessTravel | 0.003354 |
| 18 | WkDays | 0.002515 |
| 23 | PerformanceRating | 0.000000 |

## One-Hot Encoding

One-hot encoding is a process of transforming categorical features into one-hot vectors so that they can be machine-readable. The one-hot encoding technique should be performed if the values in the categorical features are nominal. It will create new dummy variables based on each category in the categorical features and map each category with binary numbers (0 and 1) (Verma, 2021).

```
[ ]  # Function to encode categorical features

     def getDummies(dataframe, dropfirst):

       encodevar = pd.get_dummies(dataframe, drop_first=dropfirst)
       return encodevar
```

**Feature Scaling**

If there is a huge difference in the scale of the data between different features, some machine learning algorithms may have a bias towards the features with a higher data range, leading to biased results. Feature scaling is a technique to convert data into the same range so that different features become more comparable (Roy, 2020). It is significant for any machine learning algorithms that are sensitive to the size and range of the features as they calculate the distances between the data, such as K-Nearest Neighbors and K-Means. In this project, features are scaled into the range of [0,1] using MinMaxScaler.

```
[ ]  # Function to standardize features by using MinMaxScaler
     # MinMaxScaler: values are scaled in the range between 0 and 1

     def scaling(feature):

       scaler = MinMaxScaler()
       scalevar = pd.DataFrame(scaler.fit_transform(feature), columns=feature.columns)
       return scalevar
```

# 6.0 Model Exploration

## 6.1 Introduction

At this stage, the dataset is partitioned into two subsets: training set and test set. 70% of the entire dataset is used as the training set to develop the models, and the remaining 30% is used as the test set to evaluate the performance of the models based on the statistics measurements. The purpose of modeling is to address the two objectives below:

1) Identify the key factors that may cause employees to leave the company
2) Predict whether an employee will leave the company

### *6.1.1 Classification Modeling*

Since Attrition is a binary outcome, classification modeling will be applied to achieve the above objectives. Below are the models used for the analysis:

- **Decision Tree.** It allows the prediction of the target variable class by comparing, ordering, and classifying the features based on the decision rules. It is easy to understand and interpret and be able to list out all the possible outcomes.
- **Random Forest.** It is an expansion of the Decision Tree algorithm. It randomly creates multiple trees from subsets of features to calculate the outputs and then combines these outputs to generate the final result that will give a better predictive performance.
- **Naïve Bayes.** Instead of classifying the target variable, the Naïve Bayes algorithm calculates the likelihood of classifying the outcome based on Bayes' Theorem. It requires conditional independence between variables.
- **Logistic Regression.** Same as Naïve Bayes, it also calculates the probability of classifying the outcome. Logistic Regression describes the relationship between the target variable and other independent features and does not require independence between variables.

- **K-Nearest Neighbors.** It classifies an input based on the majority vote or most frequent label of a specific number (k) of data points closest to the input.

### *6.1.2 Modeling Techniques*

**GridSearchCV for Hyperparameter Optimization**

GridSearchCV helps the model loop through all possible combinations of pre-defined hyperparameters automatically, then evaluates each combination using the cross-validation method (Appendix A), and locates the best estimator by scoring the chosen statistics measurement (accuracy or loss). However, as the process of GridSearchCV requires very high computing power, it can be time-consuming and financially costly (Great Learning, 2020).

**RandomizedSearchCV for Hyperparameter Optimization**

As opposed to GridSearchCV, rather than trying every single set of hyperparameters, RandomizedSearchCV randomly samples all possible combinations while restricting the total number of search iterations through the parameter "n_iter". The search iteration is set according to the time or resources available. The best combination that gives the best score is selected as output at the end. The processing time of RandomizedSearchCV is generally shorter, yet it could deliver similar results or even outperform GridSearchCV (Worcester, 2019).

**AdaBoostClassifier for Model Accuracy Improvement**

AdaBoost is one of the ensemble learning methods (also called meta-learning), aiming to increase the robustness and boost the performance of classifiers. It fits a classifier on the dataset first, then uses an iterative approach to build more classifiers and learn the mistakes from the weak ones, eventually generating a strong model (Kurama, 2020). It is especially useful for weak learners who perform only better than random guesses.

**BaggingClassifier for Model Accuracy Improvement**

Like AdaBoost, Bagging is an ensemble machine learning technique that can improve the performance of the models. BaggingClassifier is known as an ensemble meta-estimator, which combines the predictions from different classifiers/estimators by averaging (regression) or by voting (classification), and ultimately helps reduce variance (Kumar A. , 2021). It is very commonly used in classification modeling like Decision Tree and Random Forest.

**VotingClassifier for Model Accuracy Improvement**

Instead of building various models and finding the one with the best performance, VotingClassifier can train multiple base estimators, then integrate and balance out all the findings, and make predictions based on the majority of the votes for each estimator output (Kumar S. , 2021). There are two types of voting:

- **Hard Voting.** It predicts based on class labels and the weights of each estimator.
- **Soft Voting.** It predicts based on the probabilities and the weights of each estimator.

### 6.2 Decision Tree

Decision Tree is one of the popular supervised machine learning algorithms that can perform either classification (categorical outcome) or regression (numerical outcome) analysis (Chauhan N. , 2022). To predict a target variable, the process starts from the root of the tree, followed by the values comparison between the sample observation's attribute and the root attribute. Below are some important terminologies of the Decision Tree:

- **Root Node.** The entire population or sample, which will be further split into many homogeneous sets.
- **Decision Node.** The sub-node, which will be split into further sub-nodes.
- **Terminal/Leaf Node.** The final node, which will not be split further.

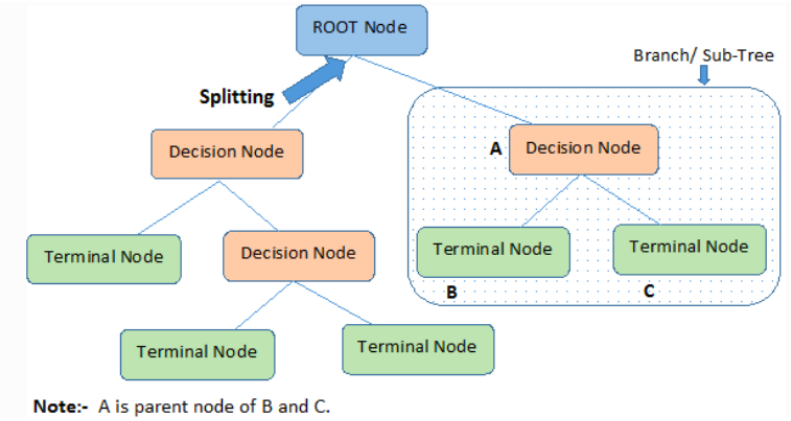



*Figure 2 Decision Tree important terminologies*

Source: KD Nuggets
Retrieved from: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

The Decision Tree applies recursive binary splitting, whereby every node is split based on a decision from one feature that can be expressed in a binary way. For example, $X_i < c$, where $X_i$ represents a feature and $c$ represents some constant, applies as a logical true or false test to each sample observation in a node (Dunkley-Hickin, 2021). The splitting process ends when the tree meets specific stopping rules.

For classification trees, the mainly applied measures to split a node are Gini index and Entropy. Gini index creates split points with the formula,

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

that is 1 minus the sum of the squared probabilities of each class (Chauhan N. , 2022). The value of Gini index ranges from 0 being the minimum to 0.5 being the maximum. Entropy is defined by the formula,

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

its value lies between 0 and 1. If the dataset is homogeneous, the Entropy is the lowest, while when the dataset is fully non-homogeneous, the Entropy is the highest (Pranto, 2020). In terms of computation, Gini index is faster while Entropy is more complicated as it uses logarithms. For the regression tree, the most applied measure is the sum of the squared

deviations from the average of the terminal/leaf node (Shmueli, Bruce, Gedeck, & Patel, 2020).

Decision Tree is easy to understand and interpret through the graphical representation. It only requires simple data preparation, handles both numerical and categorical features, uses statistical tests for validation, as well as supports multi-output problems (Scikit-Learn, 2022).

### *6.2.1 Decision Tree without Sampling*

```
[ ]  # Split the data into Train (70%) and Test (30%)

     X = emp_attr_dt.drop(columns=['Attrition'])
     y = emp_attr_dt['Attrition']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)
```

```
[ ]  # Run Decision Tree modeling

     dt_wt_cv = DecisionTreeClassifier(random_state=1)
     dt_wt_cv.fit(X_train, y_train)
```

```
[ ]  # Check the importance of each feature

     importance = dt_wt_cv.feature_importances_
     df = pd.DataFrame({'feature':X_train.columns,'importance':importance})
     print(df.sort_values('importance', ascending=False))
```

| | feature | importance |
|---|---|---|
| 0 | AvgHrs | 0.163589 |
| 14 | TotalWorkingYears | 0.115153 |
| 24 | MonthlyIncomeCAD | 0.093563 |
| 1 | Age | 0.060505 |
| 17 | YearsSinceLastPromotion | 0.055793 |
| 4 | DistanceFromHome | 0.053385 |
| 20 | JobSatisfaction | 0.047453 |
| 10 | MaritalStatus | 0.042310 |
| 16 | YearsAtCompany | 0.041543 |
| 6 | EducationField | 0.041273 |
| 11 | NumCompaniesWorked | 0.035667 |
| 3 | Department | 0.031665 |
| 21 | WorkLifeBalance | 0.027870 |
| 19 | EnvironmentSatisfaction | 0.027206 |
| 15 | TrainingTimesLastYear | 0.026631 |
| 9 | JobRole | 0.024512 |
| 7 | Gender | 0.023745 |
| 12 | PercentSalaryHike | 0.018279 |
| 13 | StockOptionLevel | 0.017959 |
| 5 | Education | 0.016986 |
| 22 | JobInvolvement | 0.011850 |
| 2 | BusinessTravel | 0.008549 |
| 8 | JobLevel | 0.007654 |
| 18 | WkDays | 0.006859 |
| 23 | PerformanceRating | 0.000000 |

**Findings:**

From the above analysis on features importance, AvgHrs is the most significant feature in predicting employee attrition, followed by TotalWorkingYears and MonthlyIncomeCAD.

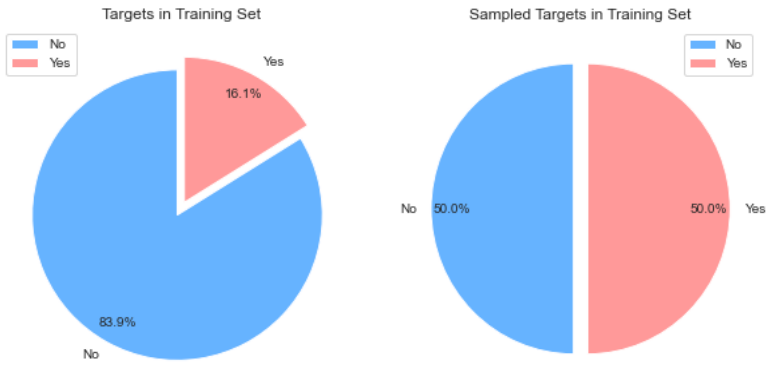
```
[ ]  # Display classification summary and accuracy

     summary('dt_wt_cv', dt_wt_cv, y_train, X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0  498
---------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9615)

       Prediction
Actual    0    1
     0 1087   23
     1   28  185

Precision Score:  0.88942308
Recall Score:  0.8685446
F1 Score:  0.87885986
```

### *6.2.2 Decision Tree with Sampling*

To prevent bias in the modeling towards predictions and to be more accurate in predicting both the majority and the minority classes, the data imbalanced issue in the dataset needs to be handled. Refer to section 5.2 for further details of the resampling algorithms used in this project.

```
[ ]  # Oversample the minority class and undersample the majority class

     sampled_X_train, sampled_y_train = sampling(X_train, y_train, True)
```




```
(3087, 25)
```

```
(5178, 25)
```

```
[ ]  # Run Decision Tree modeling

     dt_s_wt_cv = DecisionTreeClassifier(random_state=1)
     dt_s_wt_cv.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Check the importance of each feature

     importance = dt_s_wt_cv.feature_importances_
     df = pd.DataFrame({'feature':sampled_X_train.columns,'importance':importance})
     print(df.sort_values('importance', ascending=False))
```

```
                    feature  importance
0                     AvgHrs    0.138322
24          MonthlyIncomeCAD    0.117096
14         TotalWorkingYears    0.101649
16            YearsAtCompany    0.070996
1                        Age    0.060478
19  EnvironmentSatisfaction    0.060056
10             MaritalStatus    0.056104
4           DistanceFromHome    0.044725
12         PercentSalaryHike    0.039741
15    TrainingTimesLastYear    0.036815
7                     Gender    0.030511
9                    JobRole    0.025692
2             BusinessTravel    0.024548
22            JobInvolvement    0.024512
18                    WkDays    0.022296
11       NumCompaniesWorked    0.019200
20          JobSatisfaction    0.019116
17  YearsSinceLastPromotion    0.018288
13         StockOptionLevel    0.016981
5                 Education    0.016673
21          WorkLifeBalance    0.014659
23        PerformanceRating    0.012820
6            EducationField    0.012695
8                  JobLevel    0.008855
3                Department    0.007172
```

**Findings:**

From the above analysis on features importance, AvgHrs is the most significant feature in predicting employee attrition, followed by MonthlyIncomeCAD and TotalWorkingYears.

```
[ ]  # Display a classification summary and accuracy

     summary('dt_s_wt_cv', dt_s_wt_cv, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0 2589
--------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9456)

       Prediction
Actual    0    1
     0 1067   43
     1   29  184

Precision Score:  0.81057269
Recall Score:  0.86384977
F1 Score:  0.83636364
```

### *6.2.3 Decision Tree with RandomizedSearchCV*

Refer to section 6.1.2 for further details on RandomizedSearchCV for hyperparameter optimization.

```
[ ]  # Set hyperparameters

     params_dt = {'max_depth':np.arange(1,20),
                  'min_samples_leaf':np.arange(1,10),
                  'min_samples_split':np.arange(2,10)}
```

```
[ ]  # Use RandomizedSearchCV to select the best hyperparameters for optimal tree
     # Run Decision Tree modeling

     rs = RandomizedSearchCV(DecisionTreeClassifier(random_state=1),
                             param_distributions=params_dt,
                             n_iter=200, scoring="accuracy", random_state=1,
                             n_jobs=-1, cv=10, return_train_score=True)
     rs.fit(sampled_X_train,sampled_y_train)
```

```
[ ]  # Display the optimal parameters

     dt_s_w_rs = rs.best_estimator_
     dt_s_w_rs
```

```
DecisionTreeClassifier(max_depth=18, random_state=1)
```

```
[ ]  # Display classification summary and accuracy

     summary('dt_s_w_rs', dt_s_w_rs, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.9992)

       Prediction
Actual    0    1
     0 2586    3
     1    1 2588
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9327)

       Prediction
Actual    0    1
     0 1053   57
     1   32  181

Precision Score:  0.7605042
Recall Score:  0.84976526
F1 Score:  0.80266075
```

### *6.2.4 Decision Tree with AdaBoostClassifier*

Refer to section 6.1.2 for further details on AdaBoostClassifier for model accuracy improvement.

```
[ ]  # Use AdaBoostClassifier to boost the performance of Decision Tree modeling
     # Run AdaBoostClassifier

     dt_s_boost = AdaBoostClassifier(DecisionTreeClassifier(random_state=1),
                                     n_estimators=100, random_state=1)
     dt_s_boost.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display classification summary and accuracy

     summary('dt_s_boost', dt_s_boost, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0 2589
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9395)

       Prediction
Actual    0    1
     0 1065   45
     1   35  178

Precision Score:  0.79820628
Recall Score:  0.83568075
F1 Score:  0.81651376
```

### *6.2.5 Decision Tree with BaggingClassifier*

Refer to section 6.1.2 for further details on BaggingClassifier for model accuracy improvement.

```
[ ]  # Use BaggingClassifier to boost the performance of Decision Tree modeling
     # Run BaggingClassifier

     dt_s_bag = BaggingClassifier(DecisionTreeClassifier(random_state=1),
                                  n_estimators=100, random_state=1)
     dt_s_bag.fit(sampled_X_train,sampled_y_train)
```

```
[ ]  # Display classification summary and accuracy

     summary('dt_s_bag', dt_s_bag, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0 2589
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9743)

       Prediction
Actual    0    1
     0 1096   14
     1   20  193

Precision Score:  0.93236715
Recall Score:  0.90610329
F1 Score:  0.91904762
```

### *6.2.6 Decision Tree with GridSearchCV*

Refer to section 6.1.2 for further details on GridSearchCV for hyperparameter optimization.

```
[ ]  # Use GridSearchCV to select the best hyperparameters for optimal tree
     # Run Decision Tree modeling

     gs = GridSearchCV(DecisionTreeClassifier(random_state=1), params_dt, n_jobs=-1, cv=10)
     gs.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display the optimal parameters

     dt_s_w_cv = gs.best_estimator_
     dt_s_w_cv
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=18, random_state=1)
```

```
[ ]  # Display classification summary and accuracy

     summary('dt_s_w_cv', dt_s_w_cv, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.9992)

       Prediction
Actual    0    1
     0 2586    3
     1    1 2588
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9327)

       Prediction
Actual    0    1
     0 1053   57
     1   32  181

Precision Score:  0.7605042
Recall Score:  0.84976526
F1 Score:  0.80266075
```

### *6.2.7 Decision Tree with RandomizedSearchCV and AdaBoostClassifier*

```
[ ]  # Use AdaBoostClassifier to boost the performance of Decision Tree modeling

     boost = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(random_state=1),
                                n_estimators=100, random_state=1)
```

```
[ ]  # Set hyperparameters

     params_dt_en = {'base_estimator__max_depth':np.arange(1,20),
                     'base_estimator__min_samples_leaf':np.arange(1,10),
                     'base_estimator__min_samples_split':np.arange(2,10)}
```

```
[ ]  # Use RandomizedSearchCV to select the best hyperparameters for optimal tree
     # Run Decision Tree modeling

     rs = RandomizedSearchCV(boost, params_dt_en,
                             n_iter=200, scoring="accuracy", random_state=1,
                             n_jobs=-1, cv=10, return_train_score=True)
     rs.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display the optimal parameters

     dt_s_w_rs_boost = rs.best_estimator_
     dt_s_w_rs_boost
```

```
AdaBoostClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=14, min_samples_split=3, random_state=1)
```

```
[ ]  # Display classification summary and accuracy

     summary('dt_s_w_rs_boost', dt_s_w_rs_boost, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0 2589
--------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9879)

       Prediction
Actual    0    1
     0 1109    1
     1   15  198

Precision Score:  0.99497487
Recall Score:  0.92957746
F1 Score:  0.96116505
```

### *6.2.8 Decision Tree with RandomizedSearchCV and BaggingClassifier*

```
[ ]  # Use BaggingClassifier to boost the performance of Decision Tree modeling

     bag = BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=1),
                             n_estimators=100, random_state=1)
```

```
[ ]  # Use RandomizedSearchCV to select the best hyperparameters for optimal tree
     # Run Decision Tree modeling

     rs = RandomizedSearchCV(bag, params_dt_en,
                             n_iter=200, scoring="accuracy", random_state=1,
                             n_jobs=-1, cv=10, return_train_score=True)
     rs.fit(sampled_X_train,sampled_y_train)
```

```
[ ]  # Display the optimal parameters

     dt_s_w_rs_bag = rs.best_estimator_
     dt_s_w_rs_bag
```

```
BaggingClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=18, random_state=1)
```

```
[ ]  # Display classification summary and accuracy

     summary('dt_s_w_rs_bag', dt_s_w_rs_bag, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0 2589
------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9743)

       Prediction
Actual    0    1
     0 1097   13
     1   21  192

Precision Score:  0.93658537
Recall Score:  0.90140845
F1 Score:  0.91866029
```

### 6.3 Random Forest

Random Forest is one of the supervised machine learning algorithms in which the forest represents an ensemble of Decision Trees and is normally trained with the "bagging" technique. In other words, multiple Decision Trees are built and merged to achieve a prediction with higher accuracy. Like Decision Tree, Random Forest can be used for classification and regression analysis (Donges, 2021). "Bagging" and feature randomness are being used to ensure low correlation among all the trees. Random Forest selects only a subset of features rather than every possible feature split. For classification analysis, the determination of the prediction is based on the majority vote, while for regression analysis, the average of all trees is used (IBM Cloud Education, 2020). Figure 3 below shows a simple Random Forest Classifier.

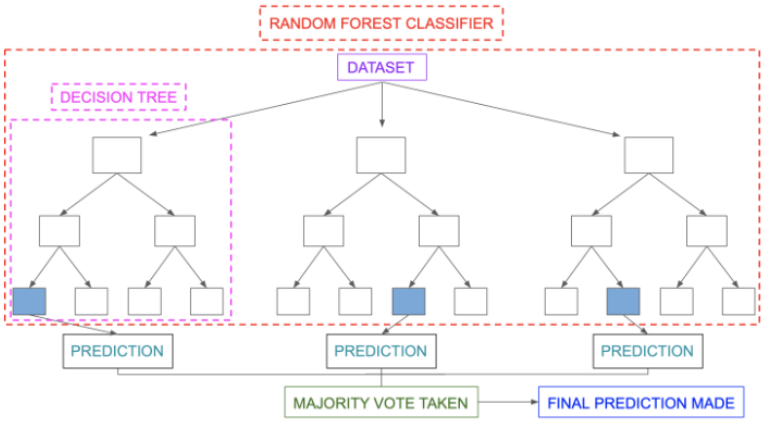



*Figure 3 Random Forest Classifier*

Source: Section
Retrieved from: https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/

Random Forest mitigates the overfitting issues by averaging the results from different trees, reduces the overall errors and variance, handles both classification and regression tasks with high accuracy, and determines feature importance more easily (IBM Cloud Education, 2020).

### *6.3.1 Random Forest without Sampling*

```
[ ]  # Run Random Forest modeling

     rf_wt_cv = RandomForestClassifier(random_state=1, n_estimators=500)
     rf_wt_cv.fit(X_train, y_train)
```

```
[ ]  # Check the importance of each feature

     feature_imp = pd.Series(rf_wt_cv.feature_importances_,
                             index= X_train.columns).sort_values(ascending=False)
     feature_imp
```

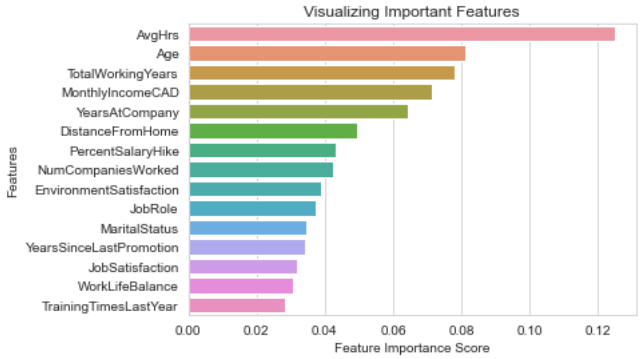
```
AvgHrs                    0.124762
Age                       0.081285
TotalWorkingYears         0.077865
MonthlyIncomeCAD          0.071483
YearsAtCompany            0.064092
DistanceFromHome          0.049394
PercentSalaryHike         0.043219
NumCompaniesWorked        0.042526
EnvironmentSatisfaction   0.038894
JobRole                   0.037283
MaritalStatus             0.034589
YearsSinceLastPromotion   0.034012
JobSatisfaction           0.031748
WorkLifeBalance           0.030631
TrainingTimesLastYear     0.028125
EducationField            0.027618
Education                 0.026929
WkDays                    0.025918
JobLevel                  0.025327
JobInvolvement            0.023510
StockOptionLevel          0.023298
BusinessTravel            0.018608
Department                0.018551
Gender                    0.014124
PerformanceRating         0.006207
dtype: float64
```

**Findings:**

From the above analysis on features importance, AvgHrs is the most significant feature in predicting employee attrition, followed by Age and TotalWorkingYears.

```
[ ]  # Visualize the importance of top 15 features

     feature_imp_top = feature_imp.head(15)

     sns.barplot(x=feature_imp_top, y=feature_imp_top.index)
     plt.xlabel('Feature Importance Score')
     plt.ylabel('Features')
     plt.title("Visualizing Important Features")
```

```
Text(0.5, 1.0, 'Visualizing Important Features')
```




```
[ ]  # Display classification summary and accuracy

     summary('rf_wt_cv', rf_wt_cv, y_train, X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0  498
---------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9841)

       Prediction
Actual    0    1
     0 1110    0
     1   21  192

Precision Score:  1.0
Recall Score:  0.90140845
F1 Score:  0.94814815
```

### *6.3.2 Random Forest with Sampling*

```
[ ]  # Run Random Forest modeling

     rf_s_wt_cv = RandomForestClassifier(random_state=1, n_estimators=500)
     rf_s_wt_cv.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Check the importance of each feature

     feature_imp = pd.Series(rf_s_wt_cv.feature_importances_,
                             index=sampled_X_train.columns).sort_values(ascending=False)
     feature_imp
```

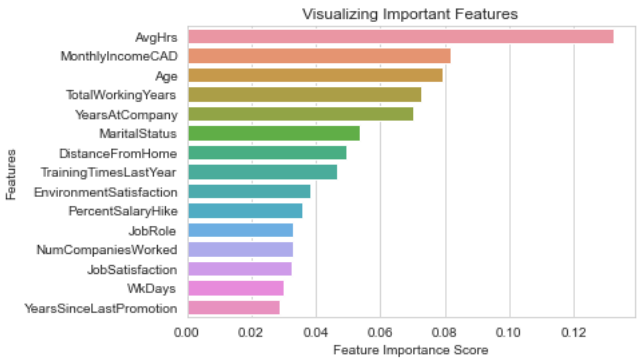
```
AvgHrs                     0.132259
MonthlyIncomeCAD           0.081786
Age                        0.079374
TotalWorkingYears          0.072839
YearsAtCompany             0.070197
MaritalStatus              0.053771
DistanceFromHome           0.049609
TrainingTimesLastYear      0.046643
EnvironmentSatisfaction    0.038154
PercentSalaryHike          0.036001
JobRole                    0.032821
NumCompaniesWorked         0.032772
JobSatisfaction            0.032661
WkDays                     0.029969
YearsSinceLastPromotion    0.028748
EducationField             0.025067
BusinessTravel             0.021396
Education                  0.021337
JobLevel                   0.021127
WorkLifeBalance            0.020202
JobInvolvement             0.019808
StockOptionLevel           0.019323
Department                 0.013555
Gender                     0.012441
PerformanceRating          0.008140
dtype: float64
```

**Findings:**

From the above analysis on features importance, AvgHrs is the most significant feature in predicting employee attrition, followed by MonthlyIncomeCAD and Age.

```
[ ]  # Visualize the importance of top 15 features

     feature_imp_top = feature_imp.head(15)

     sns.barplot(x=feature_imp_top, y=feature_imp_top.index)
     plt.xlabel('Feature Importance Score')
     plt.ylabel('Features')
     plt.title("Visualizing Important Features")
```

```
Text(0.5, 1.0, 'Visualizing Important Features')
```



```
[ ]  # Display classification summary and accuracy

     summary('rf_s_wt_cv', rf_s_wt_cv, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0 2589
--------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9849)

       Prediction
Actual    0    1
     0 1107    3
     1   17  196

Precision Score:  0.98492462
Recall Score:  0.92018779
F1 Score:  0.95145631
```

### *6.3.3 Random Forest with RandomizedSearchCV*

```
[ ]  # Set hyperparameters

     params_rf = {'n_estimators': [200,250],
                  'max_depth':np.arange(1,20),
                  'min_samples_leaf':np.arange(1,20),
                  'min_samples_split':np.arange(2,20)}
```

```
[ ]  # Use RandomizedSearchCV to select the best hyperparameters for optimal forest
     # Run Random Forest modeling

     rs = RandomizedSearchCV(RandomForestClassifier(random_state=1),
                             param_distributions=params_rf,
                             n_iter=200, scoring="accuracy", random_state=1,
                             n_jobs=-1, cv=10, return_train_score=True)
     rs.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display the optimal parameters

     rf_s_w_rs = rs.best_estimator_
     rf_s_w_rs
```

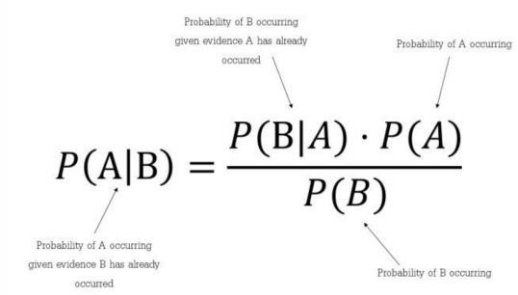
```
RandomForestClassifier(max_depth=14, min_samples_split=3, n_estimators=200,
                       random_state=1)
```

```
[ ]  # Display classification summary and accuracy

     summary('rf_s_w_rs', rf_s_w_rs, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.9996)

       Prediction
Actual    0    1
     0 2588    1
     1    1 2588
--------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9796)

       Prediction
Actual    0    1
     0 1101    9
     1   18  195

Precision Score:  0.95588235
Recall Score:  0.91549296
F1 Score:  0.9352518
```

### 6.4 Naïve Bayes

Naïve Bayes is one of the most efficient classification algorithms based on Bayes' Theorem, assuming that all features are independent and contribute equally to the outcomes. Bayes' Theorem is an equation to calculate the conditional probability of one event (A) occurring as another event (B) has already occurred (Chauhan N. S., 2022). In other words, Bayes' Theorem is a mathematical method to calculate the unknown probability from the known probability.




As mentioned above, the fundamental assumption made by Naïve Bayes is that all features are independent, which means there should be no relationship between any pair of features, even if this assumption is almost non-existent in the real situation. According to Chauhan (2022), The Naïve Bayes algorithm first calculates the posterior probability for each category by features using the above equation, then selects the category with the highest posterior probabilities as the outcome of the underlying feature. Next, use the equation below to calculate the posterior probability of a label given the observed features.

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

Finally, the Naïve Bayes algorithm decides the final outcome by comparing the ratio of the probabilities for each label (Yes or No). That is, if $P(L_1 \mid \text{features})$ is greater than $P(L_2 \mid \text{features})$, the final outcome would be L1.

**<u>Multinominal Naïve Bayes</u>**

Multinominal Naïve Bayes assumes that features follow a multinomial distribution. It is mainly used for classification with discrete features and usually requires integer feature

counts (Scikit-Learn, 2022).

**Gaussian Naïve Bayes**

Gaussian Naïve Bayes assumes that features follow a normal distribution. Therefore, the equation to calculate the conditional probability is based on the mean and standard deviation of the feature (Chauhan N. S., 2022).

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Naïve Bayes is very easy to perform and fast to make predictions. It also provides straightforward probabilities and is able to interpret the result easily (Ray, 2017).

### *6.4.1 Multinomial Naïve Bayes (only categorical features)*

```
[ ]  # Encode categorical features

     emp_attr_nb_encode = getDummies(emp_attr_nb, False)
```

```
[ ]  # Split the data into Train (70%) and Test (30%)

     X = emp_attr_nb_encode.drop(columns=['Attrition'])
     y = emp_attr_nb_encode['Attrition']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)
```

```
[ ]  # Oversample the minority class and undersample the majority class

     sampled_X_train, sampled_y_train = sampling(X_train, y_train, False)
```

```
[ ]  # Extract only the encoded categorical features

     encode_list = emp_attr_nb_encode.loc[:,emp_attr_nb_encode.dtypes==np.uint8].columns
     sampled_X_train = sampled_X_train[encode_list]
     X_test = X_test[encode_list]
```

```
[ ]  # Run Naive Bayes (Multinomial) modeling

     mnb_s = MultinomialNB(alpha=0.01)
     mnb_s.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display classification summary and accuracy

     summary('mnb_s', mnb_s, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.7607)

       Prediction
Actual    0    1
     0 1997  592
     1  647 1942
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.7181)

       Prediction
Actual   0   1
     0 846 264
     1 109 104

Precision Score:  0.2826087
Recall Score:  0.48826291
F1 Score:  0.35800344
```

```
[ ]  # Calculate probability for Attrition

     pred_prob_test = pd.DataFrame(mnb_s.predict_proba(X_test))
     pred_prob_test.rename(columns={0:'Prob_No',1:'Prob_Yes'}, inplace=True)

     y_test_pred = pd.DataFrame(mnb_s.predict(X_test))
     y_test_pred.rename(columns={0:'Predict'}, inplace=True)
```

```
[ ]  # Concatenate the original data with the probability for Attrition

     index = X_test.index
     indices = index.to_list()

     attr_prob_test = pd.concat([emp_attr.iloc[indices].reset_index(), pred_prob_test, y_test_pred], axis=1)
     attr_prob_test.head()
```

### *6.4.2 Gaussian Naïve Bayes (only numerical features)*

```
[ ]  # Split the data into Train (70%) and Test (30%)

     X = emp_attr_nb_encode.drop(columns=['Attrition']).drop(columns=encode_list)
     y = emp_attr_nb_encode['Attrition']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)
```

```
[ ]  # Oversample the minority class and undersample the majority class

     sampled_X_train, sampled_y_train = sampling(X_train, y_train, False)
```

```
[ ]  # Run Naive Bayes (Gaussian) modeling

     gnb_s_wt_cv = GaussianNB()
     gnb_s_wt_cv.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display classification summary and accuracy

     summary('gnb_s_wt_cv', gnb_s_wt_cv, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.6871)

       Prediction
Actual    0    1
     0 1820  769
     1  851 1738
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.6871)

       Prediction
Actual   0   1
     0 778 332
     1  82 131

Precision Score:  0.28293737
Recall Score:  0.61502347
F1 Score:  0.38757396
```

```
[ ]  # Calculate probability for Attrition

     pred_prob_test = pd.DataFrame(gnb_s_wt_cv.predict_proba(X_test))
     pred_prob_test.rename(columns={0:'Prob_No',1:'Prob_Yes'}, inplace=True)

     y_test_pred = pd.DataFrame(gnb_s_wt_cv.predict(X_test))
     y_test_pred.rename(columns={0:'Predict'}, inplace=True)
```

```
[ ]  # Concatenate the original data with the probability for Attrition

     index = X_test.index
     indices = index.to_list()

     attr_prob_test = pd.concat([emp_attr.iloc[indices].reset_index(), pred_prob_test, y_test_pred], axis=1)
     attr_prob_test.head()
```

### *6.4.3 Gaussian Naïve Bayes with RandomizedSearchCV*

```
[ ]  # Set hyperparameters

     param_grid_nb = {'var_smoothing':np.logspace(0, -9, num=100)}
```

```
[ ]  # Use RandomizedSearchCV to select the best hyperparameters
     # Run Naive Bayes (Gaussian) modeling

     nbModel_grid = RandomizedSearchCV(GaussianNB(), param_distributions=param_grid_nb,
                                       n_iter=100, scoring="accuracy", random_state=1,
                                       n_jobs=-1, cv=10, return_train_score=True)
     nbModel_grid.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display the optimal parameters

     gnb_s_w_rs = nbModel_grid.best_estimator_
     gnb_s_w_rs
```

```
GaussianNB
GaussianNB(var_smoothing=1.873817422860383e-05)
```

```
[ ]  # Display classification summary and accuracy

     summary('gnb_s_w_rs', gnb_s_w_rs, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.6871)

       Prediction
Actual    0    1
     0 1819  770
     1  850 1739
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.6871)

       Prediction
Actual   0   1
     0 778 332
     1  82 131

Precision Score:  0.28293737
Recall Score:  0.61502347
F1 Score:  0.38757396
```

## 6.5 Logistic Regression

Logistic Regression is a supervised machine learning algorithm for classification and predictive analytics. It estimates the probability of an event occurring based on the given features. Instead of fitting a line to the data, Logistic Regression uses a sigmoid function to fit an S-shaped curve to the features in a range between 0 and 1 (Swaminathan, 2018). By using the sigmoid function, the probability of each sample observation can be calculated and used for classification. That is, if the likelihood is greater than 50%, it will be classified as 1.

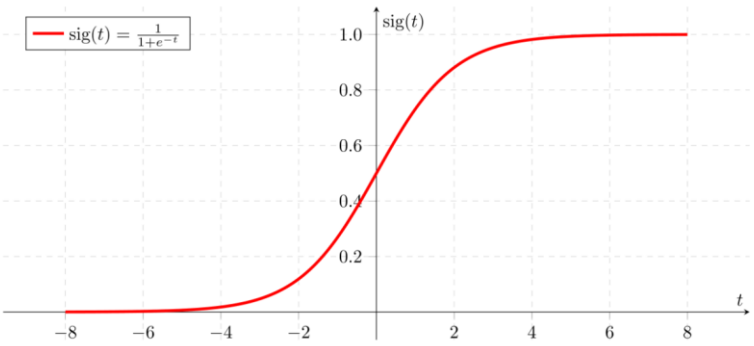



*Figure 4 Sigmoid Activation Function*

Source: Towards Data Science
Retrieved from: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Logistic Regression is easy to perform and interpretable compared to other advanced machine learning algorithms. Furthermore, it has no requirement for the distribution of classes. The predicted outcome can be easily explained by the statistics measurements such as coefficients and odds ratio (GeeksforGeeks, 2020).

### *6.5.1 Simple Logistic Regression*

```
[ ]  # Encode categorical features

     emp_attr_lr_encode = getDummies(emp_attr_lr, True)
```

```
[ ]  # Drop the encoded categorical features and target variable

     encode_list = emp_attr_lr_encode.loc[:,emp_attr_lr_encode.dtypes==np.uint8].columns
     X_num = emp_attr_lr_encode.drop(columns=['Attrition']).drop(columns=encode_list)
```

```
[ ]  # Standardize features by using MinMaxScaler
     # MinMaxScaler: values are scaled in the range between 0 and 1

     X_num = scaling(X_num)
```

```
[ ]  # Concatenate scaled numerical features with categorical features

     emp_attr_lr_encode_scaled = pd.concat([X_num, emp_attr_lr_encode.drop(columns=X_num.columns)], axis=1)
     emp_attr_lr_encode_scaled.head()
```

```
[ ]  # Split the data into Train (70%) and Test (30%)

     X = emp_attr_lr_encode_scaled.drop(columns=['Attrition'])
     y = emp_attr_lr_encode_scaled['Attrition']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)
```

```
[ ]  # Oversample the minority class and undersample the majority class

     sampled_X_train, sampled_y_train = sampling(X_train, y_train, False)
```

```
[ ]  # Run Logistic Regression modeling

     lr_s_wt_cv = LogisticRegression(solver='liblinear', random_state=1, C=1e42)
     lr_s_wt_cv.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display the intercept

     lr_s_wt_cv.intercept_
```

```
array([10.09865804])
```

```
[ ]  # Display the coefficients and odds ratio

     coeffdf = pd.DataFrame({'coef': lr_s_wt_cv.coef_[0]}, index=X.columns)
     coeffdf['odds'] = np.e**coeffdf['coef']
     coeffdf.sort_values('odds', ascending=False)
```

| | coef | odds |
|---|---|---|
| AvgHrs_Trans | 2.022120 | 7.554323 |
| YearsSinceLastPromotion_Trans | 1.157993 | 3.183539 |
| NumCompaniesWorked_Trans | 1.004108 | 2.729472 |
| BusinessTravel_2 | 0.845741 | 2.329703 |
| MaritalStatus_3 | 0.658652 | 1.932186 |
| PercentSalaryHike_Trans | 0.284377 | 1.328934 |
| BusinessTravel_3 | 0.211003 | 1.234917 |
| JobLevel_4 | 0.065940 | 1.068163 |
| WkDays | 0.028174 | 1.028575 |
| Gender_1 | -0.054471 | 0.946986 |
| MaritalStatus_2 | -0.119410 | 0.887444 |
| JobLevel_2 | -0.165710 | 0.847292 |
| PerformanceRating_4 | -0.208802 | 0.811556 |
| JobRole_6 | -0.271154 | 0.762499 |
| StockOptionLevel_1 | -0.385658 | 0.680003 |
| StockOptionLevel_3 | -0.391457 | 0.676071 |
| JobLevel_3 | -0.441032 | 0.643372 |
| DistanceFromHome_Trans | -0.458468 | 0.632251 |
| StockOptionLevel_2 | -0.561526 | 0.570338 |
| Education_2 | -0.622045 | 0.536845 |
| Education_3 | -0.686360 | 0.503405 |
| Education_4 | -0.695859 | 0.498646 |
| JobInvolvement_4 | -0.734487 | 0.479751 |
| JobInvolvement_3 | -0.736012 | 0.479021 |
| MonthlyIncomeCAD_Trans | -0.760493 | 0.467436 |
| Age | -0.761323 | 0.467048 |
| JobSatisfaction_3 | -0.793108 | 0.452436 |
| JobRole_8 | -0.844398 | 0.429816 |
| JobRole_2 | -0.890649 | 0.410389 |
| JobInvolvement_2 | -0.917737 | 0.399422 |
| JobLevel_5 | -1.018168 | 0.361256 |
| JobRole_3 | -1.111984 | 0.328906 |
| JobRole_7 | -1.126648 | 0.324118 |
| JobSatisfaction_2 | -1.208492 | 0.298647 |
| YearsAtCompany_Trans | -1.267320 | 0.281585 |
| EnvironmentSatisfaction_3 | -1.287742 | 0.275893 |
| Department_2 | -1.374941 | 0.252854 |
| JobSatisfaction_4 | -1.419074 | 0.241938 |
| WorkLifeBalance_4 | -1.433240 | 0.238535 |
| Department_3 | -1.446043 | 0.235500 |
| TrainingTimesLastYear_Trans | -1.495747 | 0.224081 |
| EnvironmentSatisfaction_2 | -1.525060 | 0.217608 |
| EnvironmentSatisfaction_4 | -1.558819 | 0.210384 |
| JobRole_4 | -1.621677 | 0.197567 |
| EducationField_4 | -1.630054 | 0.195919 |
| EducationField_6 | -1.701174 | 0.182469 |
| WorkLifeBalance_2 | -1.710164 | 0.180836 |
| EducationField_2 | -1.740518 | 0.175429 |
| JobRole_9 | -1.744274 | 0.174772 |
| WorkLifeBalance_3 | -1.752780 | 0.173291 |
| EducationField_3 | -1.828609 | 0.160637 |
| Education_5 | -1.840334 | 0.158764 |
| JobRole_5 | -2.207410 | 0.109985 |
| EducationField_5 | -2.212244 | 0.109455 |
| TotalWorkingYears_Trans | -3.705560 | 0.024586 |

**Findings:**

The above correlation coefficient and odds ratio analysis shows that for every unit change in AvgHrs, the employee is 8 times more likely to leave the company. And, with every unit change in YearsSinceLastPromotion, the employee is 3 times more likely to leave the company. Whereas every unit change in EducatoinField_5 (Other) and TotalWorkingYears, the employee is 89% and 98% less likely to leave the company, respectively.

```
[ ] # Display classification summary and accuracy

    summary('lr_s_wt_cv', lr_s_wt_cv, sampled_y_train, sampled_X_train, y_test, X_test)
```

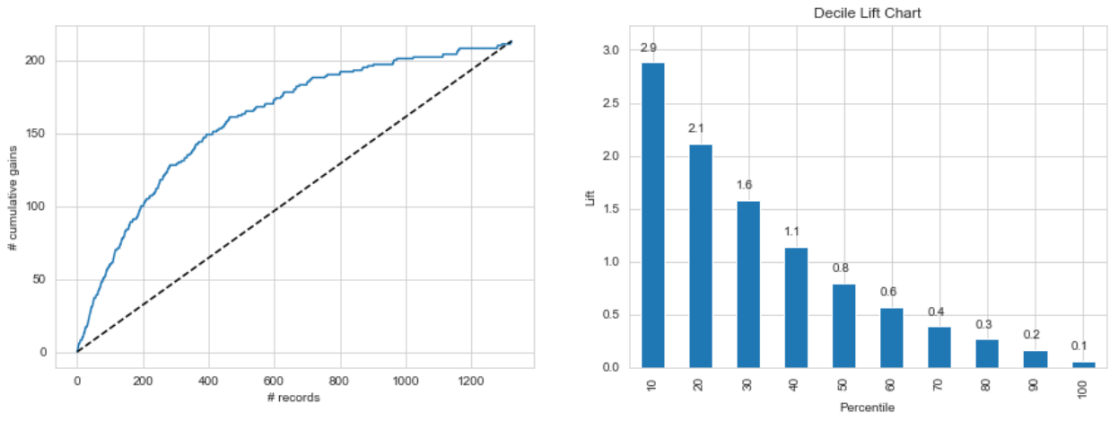
```
Train classification summary:
Confusion Matrix (Accuracy 0.8241)

       Prediction
Actual    0    1
     0 2184  405
     1  506 2083
----------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.8088)

       Prediction
Actual   0   1
     0 942 168
     1  85 128

Precision Score:  0.43243243
Recall Score:  0.60093897
F1 Score:  0.50294695
```

```
[ ] # Predict the probability of Attrition for each record

    logit_reg_prob = lr_s_wt_cv.predict_proba(X_test)
    logit_reg_pred = lr_s_wt_cv.predict(X_test)
    logit_result = pd.DataFrame({'actual':y_test,
                                 'p_0':[p[0] for p in logit_reg_prob],
                                 'p_1':[p[1] for p in logit_reg_prob],
                                 'predicted':logit_reg_pred}).sort_values(by='p_1', ascending=False)
    logit_result
```

| | actual | p_0 | p_1 | predicted |
|---|---|---|---|---|
| **2577** | 1 | 0.006854 | 0.993146 | 1 |
| **913** | 1 | 0.010231 | 0.989769 | 1 |
| **2383** | 1 | 0.010247 | 0.989753 | 1 |
| **1312** | 1 | 0.013169 | 0.986831 | 1 |
| **2782** | 1 | 0.014223 | 0.985777 | 1 |
| **...** | ... | ... | ... | ... |

```
[ ] # Visualize Cumulative Gains and Lift

    fig, axes = plt.subplots(1, 2, figsize=[15,5])
    gainsChart(logit_result['actual'], ax=axes[0])
    liftChart(logit_result['p_1'], ax=axes[1])
```

```
<AxesSubplot:title={'center':'Decile Lift Chart'}, xlabel='Percentile', ylabel='Lift'>
```



**Findings:**

The above Cumulative Gain and Lift Charts show that the top 10% of employee records have the highest predicted Attrition. For example, we would gain 2.9 times the number of Attrition compared to choosing 10% of the employee records at random. This lift number is computed from the cumulative gains chart by comparing the number of Attrition for 132 random employees (the value of the baseline curve at x = 132), which is around 20, with the actual Attrition of the 132 employees that have the highest predicted Attrition (the value of the cumulative gains curve at x = 132), which is around 60. The ratio between these numbers is about 3.

### *6.5.2 Logistic Regression with RandimizedSearchCV*

```
[ ] # Set hyperparameters

    params_lr = {'C':np.logspace(-3,5), 'penalty':['l1','l2'], 'solver':['liblinear','saga']}
```

```
[ ] # Use RandomizedSearchCV to select the best hyperparameters
    # Run Logistic Regression modeling

    rs_lr = RandomizedSearchCV(LogisticRegression(random_state=1),
                               param_distributions=params_lr,
                               n_iter=50, scoring="accuracy", random_state=1,
                               n_jobs=-1, cv=10, return_train_score=True)
    rs_lr.fit(sampled_X_train, sampled_y_train)
```

```
[ ] # Display the optimal parameters

    lr_s_w_rs = rs_lr.best_estimator_
    lr_s_w_rs
```

```
LogisticRegression
LogisticRegression(C=0.19306977288832497, random_state=1, solver='liblinear')
```

```
[ ] # Display the intercept

    lr_s_w_rs.intercept_
```

```
array([4.87848082])
```

```
[ ] # Display the coefficients and odds ratio

    coeffdf = pd.DataFrame({'coef': lr_s_w_rs.coef_[0]}, index=X.columns)
    coeffdf['odds'] = np.e**coeffdf['coef']
    coeffdf.sort_values('odds', ascending=False)
```

| | coef | odds |
|---|---|---|
| AvgHrs_Trans | 1.862989 | 6.442966 |
| BusinessTravel_2 | 0.912677 | 2.490982 |
| NumCompaniesWorked_Trans | 0.892794 | 2.441943 |
| YearsSinceLastPromotion_Trans | 0.794004 | 2.212238 |
| MaritalStatus_3 | 0.738478 | 2.092748 |
| PercentSalaryHike_Trans | 0.570529 | 1.769203 |
| BusinessTravel_3 | 0.387676 | 1.473553 |
| WkDays | 0.302552 | 1.353308 |
| MaritalStatus_2 | 0.076583 | 1.079591 |
| Gender_1 | 0.053406 | 1.054858 |
| JobLevel_4 | -0.047491 | 0.953619 |
| MonthlyIncomeCAD_Trans | -0.115464 | 0.890953 |
| JobLevel_2 | -0.143789 | 0.866070 |
| DistanceFromHome_Trans | -0.178522 | 0.836506 |
| JobRole_6 | -0.217014 | 0.804918 |
| Education_2 | -0.318530 | 0.727217 |
| StockOptionLevel_1 | -0.334080 | 0.715997 |
| JobLevel_3 | -0.362468 | 0.695957 |
| PerformanceRating_4 | -0.366856 | 0.692910 |
| JobInvolvement_4 | -0.421181 | 0.656271 |
| JobInvolvement_3 | -0.422801 | 0.655209 |
| Education_3 | -0.431425 | 0.649583 |
| Education_4 | -0.448255 | 0.638742 |
| StockOptionLevel_3 | -0.454193 | 0.634960 |
| StockOptionLevel_2 | -0.482931 | 0.616973 |
| JobRole_2 | -0.537868 | 0.583992 |
| JobRole_8 | -0.539083 | 0.583283 |
| JobSatisfaction_3 | -0.577777 | 0.561144 |
| EducationField_4 | -0.623079 | 0.536291 |
| JobInvolvement_2 | -0.640755 | 0.526895 |
| TrainingTimesLastYear_Trans | -0.655118 | 0.519381 |
| JobLevel_5 | -0.657990 | 0.517891 |
| JobRole_3 | -0.676902 | 0.508189 |
| JobRole_7 | -0.701161 | 0.496009 |
| EducationField_2 | -0.749791 | 0.472465 |
| EducationField_3 | -0.751272 | 0.471766 |
| EducationField_6 | -0.752658 | 0.471113 |
| WorkLifeBalance_4 | -0.837337 | 0.432862 |
| Age | -0.844466 | 0.429787 |
| JobSatisfaction_2 | -0.907588 | 0.403496 |
| YearsAtCompany_Trans | -0.962777 | 0.381831 |
| Education_5 | -0.969441 | 0.379295 |
| Department_2 | -0.971920 | 0.378356 |
| EducationField_5 | -1.028337 | 0.357601 |
| Department_3 | -1.034586 | 0.355373 |
| EnvironmentSatisfaction_3 | -1.044464 | 0.351880 |
| JobRole_4 | -1.089598 | 0.336352 |
| WorkLifeBalance_2 | -1.119139 | 0.326561 |
| JobSatisfaction_4 | -1.138279 | 0.320370 |
| WorkLifeBalance_3 | -1.155292 | 0.314966 |
| EnvironmentSatisfaction_4 | -1.208080 | 0.298770 |
| JobRole_9 | -1.211716 | 0.297686 |
| EnvironmentSatisfaction_2 | -1.231657 | 0.291809 |
| JobRole_5 | -1.485278 | 0.226439 |
| TotalWorkingYears_Trans | -2.116479 | 0.120455 |

**Findings:**

The above correlation coefficient and odds ratio analysis shows that for every unit change in AvgHrs, the employee is 6 times more likely to leave the company. And, for every unit change in BusinessTravel_2 (Travel Frequently), the employee is 2 times more likely to leave the company. Whereas every unit change in JobRole_5 (Manufacturing Director) and TotalWorkingYears, the employee is 77% and 88% less likely to leave the company, respectively.

```
[ ]  # Display classification summary and accuracy

     summary('lr_s_w_rs', lr_s_w_rs, sampled_y_train, sampled_X_train, y_test, X_test)
```

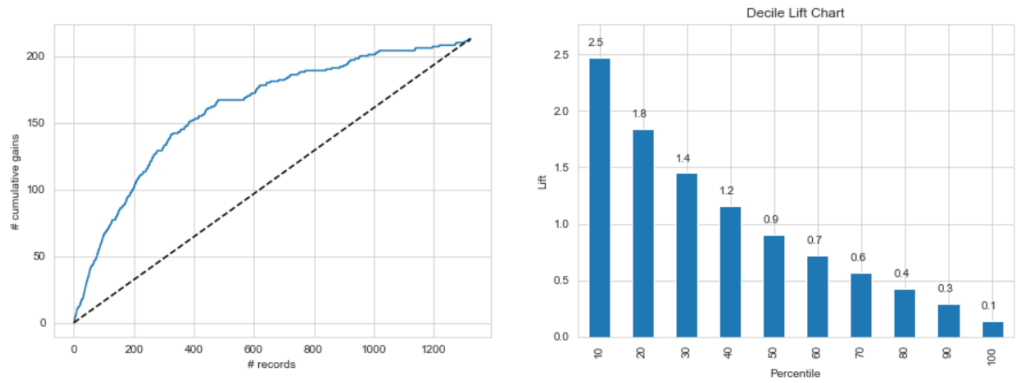
```
Train classification summary:
Confusion Matrix (Accuracy 0.8239)

       Prediction
Actual    0    1
     0 2202  387
     1  525 2064
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.8209)

       Prediction
Actual   0   1
     0 957 153
     1  84 129

Precision Score:  0.45744681
Recall Score:  0.6056338
F1 Score:  0.52121212
```

```
[ ]  # Predict the probability of Attrition for each record

     logit_reg_prob = lr_s_w_rs.predict_proba(X_test)
     logit_reg_pred = lr_s_w_rs.predict(X_test)
     logit_result = pd.DataFrame({'actual':y_test,
                                  'p_0':[p[0] for p in logit_reg_prob],
                                  'p_1':[p[1] for p in logit_reg_prob],
                                  'predicted':logit_reg_pred}).sort_values(by='p_1', ascending=False)
     logit_result
```

| | actual | p_0 | p_1 | predicted |
|---|---|---|---|---|
| **2383** | 1 | 0.028702 | 0.971298 | 1 |
| **913** | 1 | 0.028998 | 0.971002 | 1 |
| **1312** | 1 | 0.049481 | 0.950519 | 1 |
| **2577** | 1 | 0.052788 | 0.947212 | 1 |
| **2782** | 1 | 0.054070 | 0.945930 | 1 |

```
[ ]  # Visualize Cumulative Gains and Lift

     fig, axes = plt.subplots(1, 2, figsize=[15,5])
     gainsChart(logit_result['actual'], ax=axes[0])
     liftChart(logit_result['p_1'], ax=axes[1])
```

```
<AxesSubplot:title={'center':'Decile Lift Chart'}, xlabel='Percentile', ylabel='Lift'>
```



**Findings:**

The above Cumulative Gain and Lift Charts show that the top 10% of employee records have the highest predicted Attrition. For example, we would gain 2.5 times the number of Attrition compared to choosing 10% of the employee records at random. This lift number is computed from the cumulative gains chart by comparing the number of Attrition for 132 random employees (the value of the baseline curve at x = 132), which is around 25, with the actual Attrition of the 132 employees that have the highest predicted Attrition (the value of the cumulative gains curve at x = 132), which is around 60. The ratio between these numbers is about 2.5.

### *6.5.3 Logistic Regression with AdaBoostClassifier*

```
[ ]  # Use AdaBoostClassifier to boost the performance of Logistic Regression modeling
     # Run AdaBoostClassifier

     lr_s_boost = AdaBoostClassifier(LogisticRegression(solver='liblinear', random_state=1, C=1e42),
                                     n_estimators=100, random_state=1)
     lr_s_boost.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display classification summary and accuracy

     summary('lr_s_boost', lr_s_boost, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.8190)

       Prediction
Actual    0    1
     0 2160  429
     1  508 2081
------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.7906)

       Prediction
Actual   0   1
     0 924 186
     1  91 122

Precision Score:  0.3961039
Recall Score:  0.57276995
F1 Score:  0.46833013
```

### *6.5.4 Logistic Regression with BaggingClassifier*

```
[ ]  # Use BaggingClassifier to boost the performance of Logistic Regression modeling
     # Run BaggingClassifier

     lr_s_bag = BaggingClassifier(LogisticRegression(solver='liblinear', random_state=1, C=1e42),
                                  n_estimators=100, random_state=1)
     lr_s_bag.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display classification summary and accuracy

     summary('lr_s_bag', lr_s_bag, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.8237)

       Prediction
Actual    0    1
     0 2183  406
     1  507 2082
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.8103)

       Prediction
Actual   0   1
     0 944 166
     1  85 128

Precision Score:  0.43537415
Recall Score:  0.60093897
F1 Score:  0.50493097
```

### *6.5.5 Logistic Regression without Scaling*

```
[ ]  # Encode categorical features

     emp_attr_lr_encode = getDummies(emp_attr_lr, True)
```

```
[ ]  # Split the data into Train (70%) and Test (30%)

     X = emp_attr_lr_encode.drop(columns=['Attrition'])
     y = emp_attr_lr_encode['Attrition']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)
```

```
[ ]  # Oversample the minority class and undersample the majority class

     sampled_X_train, sampled_y_train = sampling(X_train, y_train, False)
```

```
[ ]  # Run Logistic Regression modeling

     lr_s_wt_scale = LogisticRegression(solver='liblinear', random_state=1, C=1e42)
     lr_s_wt_scale.fit(sampled_X_train, sampled_y_train)
```

```
[ ]  # Display the intercept

     lr_s_wt_scale.intercept_
```

```
array([8.53889228])
```

```
[ ]  # Display the coefficients and odds ratio

     coeffdf = pd.DataFrame({'coef': lr_s_wt_scale.coef_[0]}, index=X.columns)
     coeffdf['odds'] = np.e**coeffdf['coef']
     coeffdf.sort_values('odds',ascending=False)
```

| | coef | odds |
|---|---|---|
| BusinessTravel_2 | 0.628123 | 1.874089 |
| MaritalStatus_3 | 0.626268 | 1.870617 |
| AvgHrs_Trans | 0.561230 | 1.752827 |
| YearsSinceLastPromotion_Trans | 0.363672 | 1.438602 |
| JobLevel_4 | 0.327138 | 1.386993 |
| NumCompaniesWorked_Trans | 0.293075 | 1.340543 |
| PercentSalaryHike_Trans | 0.081927 | 1.085376 |
| JobRole_6 | 0.001516 | 1.001517 |
| WkDays | -0.003158 | 0.996847 |
| Age | -0.013124 | 0.986962 |
| BusinessTravel_3 | -0.047861 | 0.953266 |
| Gender_1 | -0.070596 | 0.931839 |
| JobLevel_2 | -0.084677 | 0.918809 |
| DistanceFromHome_Trans | -0.130463 | 0.877689 |
| YearsAtCompany_Trans | -0.156838 | 0.854842 |
| MaritalStatus_2 | -0.166393 | 0.846713 |
| PerformanceRating_4 | -0.168426 | 0.844994 |
| MonthlyIncomeCAD_Trans | -0.218404 | 0.803800 |
| StockOptionLevel_2 | -0.253872 | 0.775791 |
| TrainingTimesLastYear_Trans | -0.257600 | 0.772905 |
| StockOptionLevel_3 | -0.275728 | 0.759020 |
| StockOptionLevel_1 | -0.314328 | 0.730280 |
| JobLevel_3 | -0.432854 | 0.648655 |
| JobRole_2 | -0.459659 | 0.631499 |
| Education_2 | -0.480955 | 0.618193 |
| JobInvolvement_4 | -0.481106 | 0.618099 |
| JobRole_8 | -0.634771 | 0.530057 |
| Education_4 | -0.650963 | 0.521543 |
| Education_3 | -0.707809 | 0.492723 |
| JobInvolvement_3 | -0.771741 | 0.462208 |
| TotalWorkingYears_Trans | -0.779368 | 0.458696 |
| Department_2 | -0.791617 | 0.453112 |
| JobSatisfaction_3 | -0.801381 | 0.448709 |
| JobRole_3 | -0.801870 | 0.448490 |
| JobLevel_5 | -0.826666 | 0.437505 |
| JobInvolvement_2 | -0.847296 | 0.428572 |
| Department_3 | -0.874385 | 0.417118 |
| JobRole_7 | -0.929834 | 0.394619 |
| JobSatisfaction_2 | -1.152845 | 0.315737 |
| JobRole_4 | -1.168999 | 0.310678 |
| EnvironmentSatisfaction_3 | -1.315373 | 0.268374 |
| JobRole_9 | -1.358382 | 0.257076 |
| JobSatisfaction_4 | -1.387070 | 0.249806 |
| EnvironmentSatisfaction_2 | -1.575587 | 0.206886 |
| EnvironmentSatisfaction_4 | -1.639941 | 0.193991 |
| WorkLifeBalance_2 | -1.705298 | 0.181718 |
| WorkLifeBalance_4 | -1.707966 | 0.181234 |
| WorkLifeBalance_3 | -1.820658 | 0.161919 |
| Education_5 | -1.840042 | 0.158811 |
| JobRole_5 | -2.083583 | 0.124483 |
| EducationField_6 | -2.115115 | 0.120619 |
| EducationField_2 | -2.177765 | 0.113294 |
| EducationField_4 | -2.179210 | 0.113131 |
| EducationField_3 | -2.305072 | 0.099752 |
| EducationField_5 | -2.765678 | 0.062933 |

**Findings:**

The above correlation coefficient and odds ratio analysis shows that for every unit change in both BusinessTravel_2 (Travel Frequently) and MaritalStatus_3 (Single), the employee is 87% more likely to leave the company. Whereas every unit changes in Education_3 (Bachelor) and Education_5 (Doctor), the employee is 90% and 94% less likely to leave the company, respectively.

```
[ ]  # Display classification summary and accuracy

     summary('lr_s_wt_scale', lr_s_wt_scale, sampled_y_train, sampled_X_train, y_test, X_test)
```

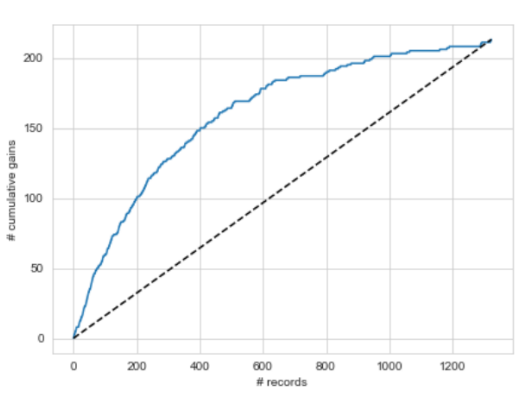
```
Train classification summary:
Confusion Matrix (Accuracy 0.8681)

       Prediction
Actual    0    1
     0 2352  237
     1  446 2143
------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.8413)

       Prediction
Actual    0    1
     0 1018   92
     1  118   95

Precision Score:  0.50802139
Recall Score:  0.44600939
F1 Score:  0.475
```

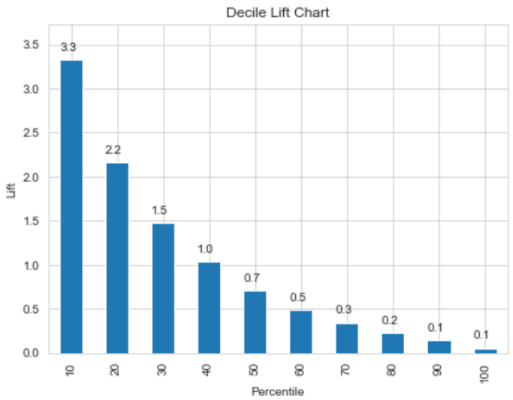
```
[ ]  # Predict the probability of Attrition for each record

     logit_reg_prob = lr_s_wt_scale.predict_proba(X_test)
     logit_reg_pred = lr_s_wt_scale.predict(X_test)
     logit_result = pd.DataFrame({'actual':y_test,
                                  'p_0':[p[0] for p in logit_reg_prob],
                                  'p_1':[p[1] for p in logit_reg_prob],
                                  'predicted':logit_reg_pred}).sort_values(by='p_1', ascending=False)
     logit_result
```

| | actual | p_0 | p_1 | predicted |
|---|---|---|---|---|
| **2577** | 1 | 0.013417 | 0.986583 | 1 |
| **913** | 1 | 0.013782 | 0.986218 | 1 |
| **2383** | 1 | 0.013864 | 0.986136 | 1 |
| **3475** | 0 | 0.029962 | 0.970038 | 1 |
| **1868** | 1 | 0.030354 | 0.969646 | 1 |

```
[ ]  # Visualize Cumulative Gains and Lift

     fig, axes = plt.subplots(1, 2, figsize=[15,5])
     gainsChart(logit_result['actual'], ax=axes[0])
     liftChart(logit_result['p_1'], ax=axes[1])
```

```
<AxesSubplot:title={'center':'Decile Lift Chart'}, xlabel='Percentile', ylabel='Lift'>
```



**Findings:**

The above Cumulative Gain and Lift Charts show that the top 10% of employee records have the highest predicted Attrition. For example, we would gain 3.3 times the number of Attrition compared to choosing 10% of the employee records at random. This lift number is computed from the cumulative gains chart by comparing the number of Attrition for 132 random employees (the value of the baseline curve at x = 132), which is around 23, with the actual Attrition of the 132 employees that have the highest predicted Attrition (the value of the cumulative gains curve at x = 132), which is around 75. The ratio between these numbers is about 3.3.

### 6.6 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) is one of the supervised machine learning algorithms that assumes similar things are close to each other. New data are put into the category, which is most similar to the available categories. It performs action on the dataset during the classification rather than immediately learning from the training set. As a non-parametric algorithm, KNN does not apply any assumption to the data (Datasciencelovers, 2020). After selecting the number of k neighbors, it uses Euclidean distance to identify the closest neighbors. When k is too small, there will be noise that affects the results; when k is too large, the computation will become more complex. K-fold cross-validation can be used to obtain the optimal k (Mirbozorgi, 2020). Figure 4 below shows the effect of k value and the selection region.

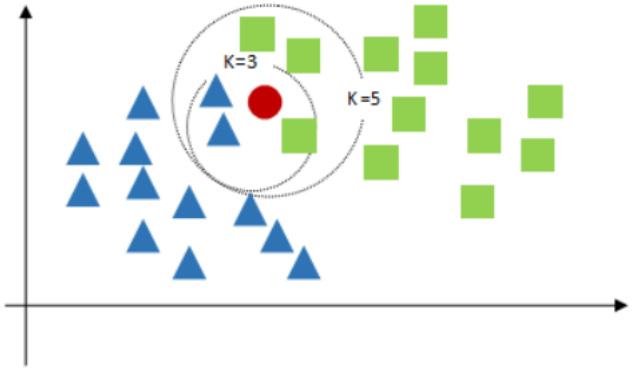



*Figure 5 Effect of K Value and Selection Region*

Source: Medium
Retrieved from: https://medium.com/analytics-vidhya/theory-of-k-nearest-neighbors-knn-234654826c50

KNN is easy to implement and can be used for both classification and regression analysis. It is not necessary to build models and make additional assumptions. The hyperparameter tuning process is easy because KNN only has a single hyperparameter, k value (MLNerds, 2019).

### *6.6.1 Simple K-Nearest Neighbors*

```
[ ]  # Encode categorical features

     emp_attr_knn_encode = getDummies(emp_attr_knn, True)
```

```
[ ]  # Drop the encoded categorical features and target variable

     encode_list = emp_attr_knn_encode.loc[:,emp_attr_knn_encode.dtypes==np.uint8].columns
     X_num = emp_attr_knn_encode.drop(columns=['Attrition']).drop(columns=encode_list)
```

```
[ ]  # Standardize features by using MinMaxScaler
     # MinMaxScaler: values are scaled in the range between 0 and 1

     X_num = scaling(X_num)
```

```
[ ]  # Concatenate scaled numerical features with categorical features

     emp_attr_knn_encode_scaled = pd.concat([X_num, emp_attr_knn_encode.drop(columns = X_num.columns)], axis=1)
     emp_attr_knn_encode_scaled.head()
```

| | Age | WkDays | YearsSinceLastPromotion_Trans | MonthlyIncomeCAD_Trans | DistanceFromHome_Trans | YearsAtCompany_Trans | TotalWorkingYears_Trans |
|---|---|---|---|---|---|---|---|
| **0** | 0.785714 | 0.304348 | 0.000000 | 0.892796 | 0.455371 | 0.158369 | 0.114002 |
| **1** | 0.309524 | 0.478261 | 0.407877 | 0.552095 | 0.622701 | 0.434173 | 0.389589 |
| **2** | 0.333333 | 0.739130 | 0.000000 | 0.991692 | 0.806883 | 0.434173 | 0.349938 |
| **3** | 0.476190 | 0.434783 | 0.872522 | 0.766355 | 0.146043 | 0.544257 | 0.591937 |
| **4** | 0.333333 | 0.869565 | 0.000000 | 0.345220 | 0.622701 | 0.475475 | 0.488546 |

5 rows × 56 columns

```
[ ] # Split the data into Train (70%) and Test (30%)

    X = emp_attr_knn_encode_scaled.drop(columns=['Attrition'])
    y = emp_attr_knn_encode_scaled['Attrition']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=1)
```

```
[ ] # Oversample the minority class and undersample the majority class

    sampled_X_train, sampled_y_train = sampling(X_train, y_train, False)
```

```
[ ] # Run K-Nearest Neighbors modeling

    knn_s_wt_cv = KNeighborsClassifier()
    knn_s_wt_cv.fit(sampled_X_train, sampled_y_train)
```

```
[ ] # Display classification summary and accuracy

    summary('knn_s_wt_cv', knn_s_wt_cv, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.9424)

       Prediction
Actual    0    1
     0 2293  296
     1    2 2587
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.7748)

       Prediction
Actual   0   1
     0 822 288
     1  10 203

Precision Score:  0.41344196
Recall Score:  0.95305164
F1 Score:  0.57670455
```

## *6.6.2 K-Nearest Neighbors with GridSearchCV*

```
[ ] # Use GridSearchCV to select the best hyperparameters
    # Run K-Nearest Neighbors modeling

    knn = KNeighborsClassifier()
    param_knn = {'n_neighbors':np.arange(1, 20)}
    gs_knn = GridSearchCV(knn, param_knn, cv=10, scoring='accuracy',
                          return_train_score=False, verbose=1)
    gs_knn.fit(sampled_X_train, sampled_y_train)
```

```
[ ] # Display the optimal parameters

    knn_s_w_cv = gs_knn.best_estimator_
    knn_s_w_cv
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=2)
```

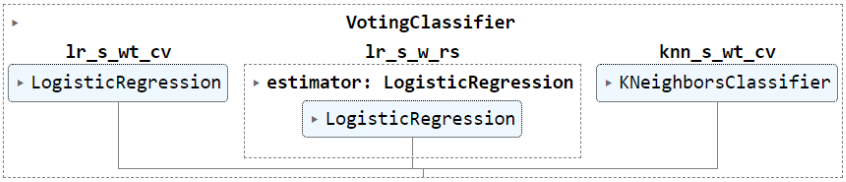
```
[ ] # Display classification summary and accuracy

    summary('knn_s_w_cv', knn_s_w_cv, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2589    0
     1    0 2589
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.9690)

       Prediction
Actual    0    1
     0 1078   32
     1    9  204

Precision Score:  0.86440678
Recall Score:  0.95774648
F1 Score:  0.90868597
```

## 6.7 Voting Classifier

Refer to section 6.1.2 for further details on VotingClassifier.

```
[ ]  # Run VotingClassifier on 3 sub-models from Logistic Regression and K-Nearest Neighbors

     vc3 = VotingClassifier(estimators=[('lr_s_wt_cv', lr_s_wt_cv), ('lr_s_w_rs', rs_lr),
                                        ('knn_s_wt_cv', knn_s_wt_cv)],
                            voting='soft', weights=[2,2,1])
     vc3.fit(sampled_X_train, sampled_y_train)
```




```
[ ]  # Display classification summary and accuracy

     summary('vc3', vc3, sampled_y_train, sampled_X_train, y_test, X_test)
```

```
Train classification summary:
Confusion Matrix (Accuracy 0.8874)

       Prediction
Actual    0    1
     0 2319  270
     1  313 2276
------------------------------------------------------------------------------

Test classification summary:
Confusion Matrix (Accuracy 0.8541)

       Prediction
Actual   0   1
     0 978 132
     1  61 152

Precision Score:  0.53521127
Recall Score:  0.71361502
F1 Score:  0.61167002
```

## 6.8 Model Comparison

The Area Under the Curve (AUC) of the Receiver-Operator-Characteristic (ROC) Curve is a graph used to evaluate binary classification models' performance at all possible classification thresholds. This curve is mathematically represented by four values or two statistics measurements (Lekhtman, 2019):

- **True Positive (TP).** The number of the data points (class = 1) above the threshold.
- **False Positive (FP).** The number of the data points (class = 0) above the threshold.
- **True Negative (TN).** The number of the data points (class = 0) below the threshold.
- **False Negative (FN).** The number of the data points (class = 1) below the threshold.

| | | Real | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted | Positive | True Positive (tp) | False Positive (fp) |
| | Negative | False Negative (fn) | True Negative (tn) |

*Figure 6 Confusion Matrix*

Source: Towards Data Science
Retrieved from: https://towardsdatascience.com/should-i-look-at-precision-recall-or-specificity-sensitivity-3946158aace1

- **True Positive Rate (TPR or Sensitivity or Recall).** TP/(TP+FN), identifying the number of predicted positives within all the positive observations.
- **False Positive Rate (FPR or Specificity).** FP/(FP+TN), identifying the number of predicted negatives within all the negative observations.

The ROC curve is plotted based on the TPR against FPR at all the possible thresholds, and AUC is basically the area under the ROC curve.

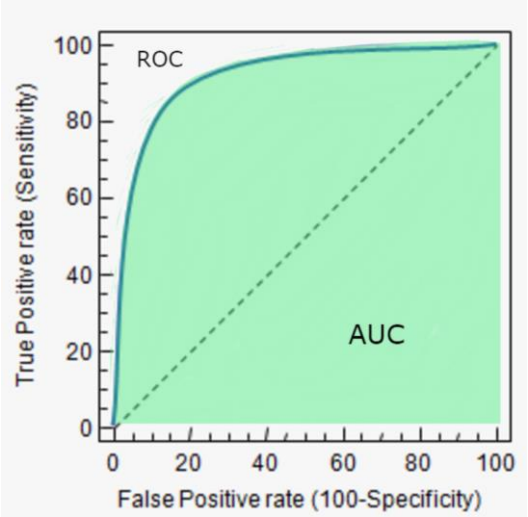


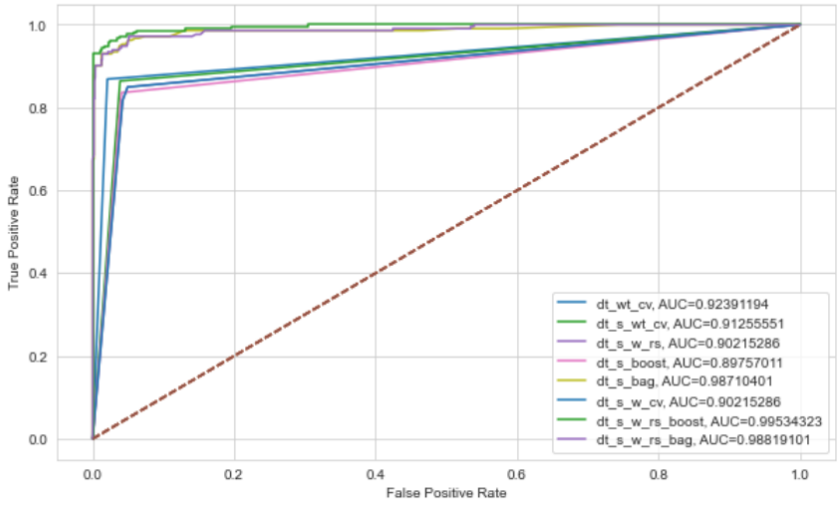

*Figure 7 AUC-ROC curve*

Source: Medium
Retrieved from: https://medium.com/geekculture/classification-model-performance-evaluation-using-auc-roc-and-cap-curves-66a1b3fc0480

The AUC-ROC curve indicates the capability of the model in distinguishing and predicting between classes. The higher the AUC, the better the model performs.

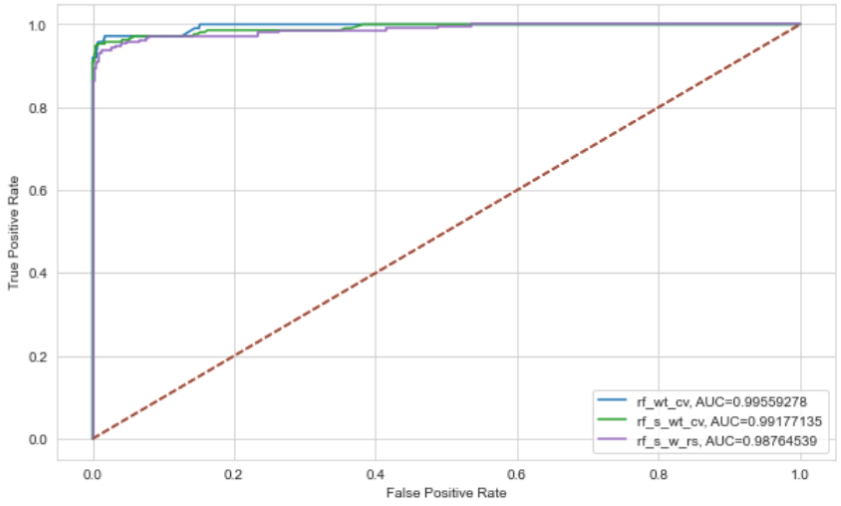
```
[ ]  # Visualize ROC Curve and AUC of all sub-models in Decision Tree

     dt_list_fpr = [fpr_dt_wt_cv,fpr_dt_s_wt_cv,fpr_dt_s_w_rs,fpr_dt_s_boost,fpr_dt_s_bag,
                    fpr_dt_s_w_cv,fpr_dt_s_w_rs_boost,fpr_dt_s_w_rs_bag]
     dt_list_tpr = [tpr_dt_wt_cv,tpr_dt_s_wt_cv,tpr_dt_s_w_rs,tpr_dt_s_boost,tpr_dt_s_bag,
                    tpr_dt_s_w_cv,tpr_dt_s_w_rs_boost,tpr_dt_s_w_rs_bag]
     dt_list_auc = [auc_dt_wt_cv,auc_dt_s_wt_cv,auc_dt_s_w_rs, auc_dt_s_boost, auc_dt_s_bag,
                    auc_dt_s_w_cv,auc_dt_s_w_rs_boost,auc_dt_s_w_rs_bag]
     dt_list = ['dt_wt_cv','dt_s_wt_cv','dt_s_w_rs','dt_s_boost','dt_s_bag',
                'dt_s_w_cv','dt_s_w_rs_boost','dt_s_w_rs_bag']

     highestAUC_score = 0
     plt.subplots(1, figsize=(10,6))

     for i, j, k, l in zip(dt_list_fpr, dt_list_tpr, dt_list_auc, dt_list):
       if k > highestAUC_score:
         highestAUC_score = k
         highestAUC_model = l
       plt.plot(i, j, label=str(l)+', AUC='+str(k))
       plt.plot([0,1], ls='--')
       plt.ylabel('True Positive Rate')
       plt.xlabel('False Positive Rate')
       plt.legend(loc='lower right')

     print('Decision Tree model with the best AUC is: '+str(highestAUC_model)+'['+str(highestAUC_score)+']\n')
```

```
Decision Tree model with the best AUC is: dt_s_w_rs_boost[0.99534323]
```



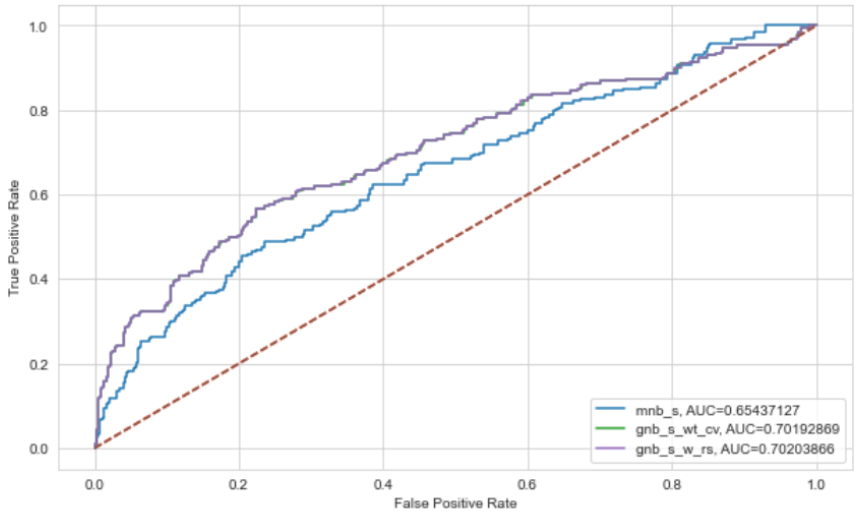
```
[ ]  # Visualize the ROC Curve and AUC of all sub-models in Random Forest

     rf_list_fpr = [fpr_rf_wt_cv,fpr_rf_s_wt_cv,fpr_rf_s_w_rs]
     rf_list_tpr = [tpr_rf_wt_cv,tpr_rf_s_wt_cv,tpr_rf_s_w_rs]
     rf_list_auc = [auc_rf_wt_cv,auc_rf_s_wt_cv,auc_rf_s_w_rs]
     rf_list = ['rf_wt_cv','rf_s_wt_cv','rf_s_w_rs']

     highestAUC_score = 0
     plt.subplots(1, figsize=(10,6))

     for i, j, k, l in zip(rf_list_fpr, rf_list_tpr, rf_list_auc, rf_list):
       if k > highestAUC_score:
         highestAUC_score = k
         highestAUC_model = l
       plt.plot(i, j, label=str(l)+', AUC='+str(k))
       plt.plot([0,1], ls='--')
       plt.ylabel('True Positive Rate')
       plt.xlabel('False Positive Rate')
       plt.legend(loc='lower right')

     print('Random Forest model with the best AUC is: '+str(highestAUC_model)+'['+str(highestAUC_score)+']\n')
```

```
Random Forest model with the best AUC is: rf_wt_cv[0.99559278]
```



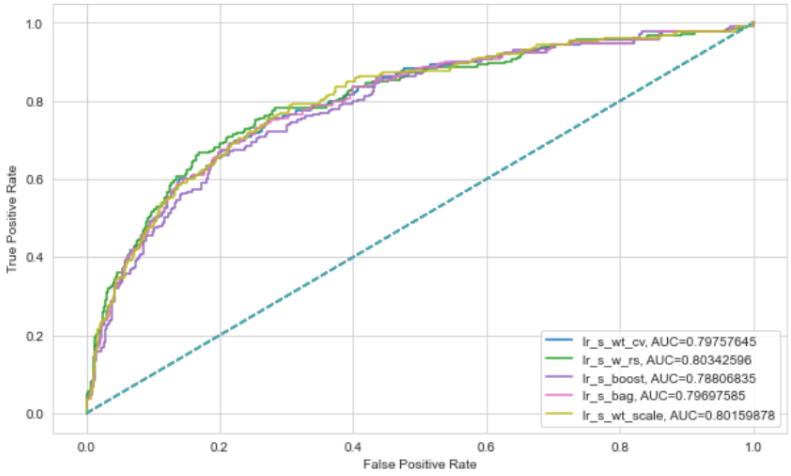
```
[ ]  # Visualize the ROC Curve and AUC of all sub-models in Naive Bayes

     nb_list_fpr = [fpr_mnb_s,fpr_gnb_s_wt_cv,fpr_gnb_s_w_rs]
     nb_list_tpr = [tpr_mnb_s,tpr_gnb_s_wt_cv,tpr_gnb_s_w_rs]
     nb_list_auc = [auc_mnb_s,auc_gnb_s_wt_cv,auc_gnb_s_w_rs]
     nb_list = ['mnb_s','gnb_s_wt_cv','gnb_s_w_rs']

     highestAUC_score = 0
     plt.subplots(1, figsize=(10,6))

     for i, j, k, l in zip(nb_list_fpr, nb_list_tpr, nb_list_auc, nb_list):
       if k > highestAUC_score:
         highestAUC_score = k
         highestAUC_model = l
       plt.plot(i, j, label=str(l)+', AUC='+str(k))
       plt.plot([0,1], ls='--')
       plt.ylabel('True Positive Rate')
       plt.xlabel('False Positive Rate')
       plt.legend(loc='lower right')

     print('Naive Bayes model with the best AUC is: '+str(highestAUC_model)+'['+str(highestAUC_score)+']\n')
```

```
Naive Bayes model with the best AUC is: gnb_s_w_rs[0.70203866]
```



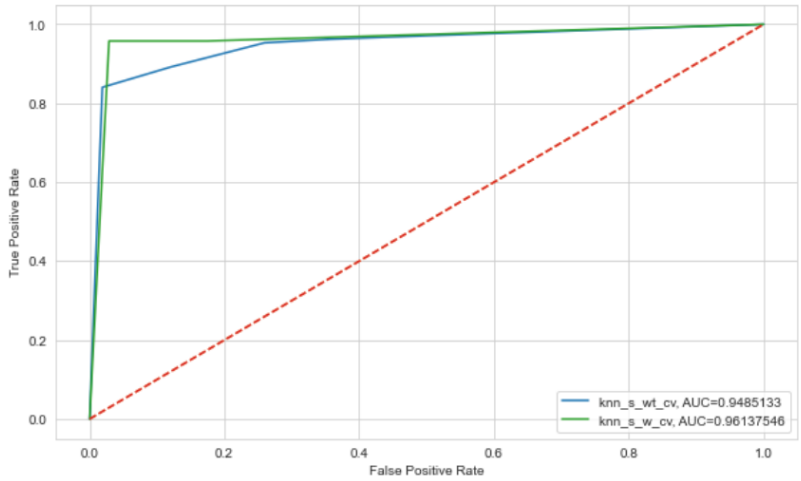
```
[ ]  # Visualize the ROC Curve and AUC of all sub-models in Logistic Regression

     lr_list_fpr = [fpr_lr_s_wt_cv,fpr_lr_s_w_rs,fpr_lr_s_boost,fpr_lr_s_bag,
                    fpr_lr_s_wt_scale]
     lr_list_tpr = [tpr_lr_s_wt_cv,tpr_lr_s_w_rs,tpr_lr_s_boost,tpr_lr_s_bag,
                    tpr_lr_s_wt_scale]
     lr_list_auc = [auc_lr_s_wt_cv,auc_lr_s_w_rs,auc_lr_s_boost,auc_lr_s_bag,
                    auc_lr_s_wt_scale]
     lr_list = ['lr_s_wt_cv','lr_s_w_rs','lr_s_boost','lr_s_bag','lr_s_wt_scale']

     highestAUC_score = 0
     plt.subplots(1, figsize=(10,6))

     for i, j, k, l in zip(lr_list_fpr, lr_list_tpr, lr_list_auc, lr_list):
       if k > highestAUC_score:
         highestAUC_score = k
         highestAUC_model = l
       plt.plot(i, j, label=str(l)+', AUC='+str(k))
       plt.plot([0,1], ls='--')
       plt.ylabel('True Positive Rate')
       plt.xlabel('False Positive Rate')
       plt.legend(loc='lower right')

     print('Logistic Regression model with the best AUC is: '+str(highestAUC_model)+'['+str(highestAUC_score)+']\n')
```

```
Logistic Regression model with the best AUC is: lr_s_w_rs[0.80342596]
```



```
[ ]  # Visualize the ROC Curve and AUC of all sub-models in K-Nearest Neighbors

     knn_list_fpr = [fpr_knn_s_wt_cv,fpr_knn_s_w_cv]
     knn_list_tpr = [tpr_knn_s_wt_cv,tpr_knn_s_w_cv]
     knn_list_auc = [auc_knn_s_wt_cv,auc_knn_s_w_cv]
     knn_list = ['knn_s_wt_cv','knn_s_w_cv']

     highestAUC_score = 0
     plt.subplots(1, figsize=(10,6))

     for i, j, k, l in zip(knn_list_fpr, knn_list_tpr, knn_list_auc, knn_list):
       if k > highestAUC_score:
         highestAUC_score = k
         highestAUC_model = l
       plt.plot(i, j, label=str(l)+', AUC='+str(k))
       plt.plot([0,1], ls='--')
       plt.ylabel('True Positive Rate')
       plt.xlabel('False Positive Rate')
       plt.legend(loc='lower right')

     print('K-Nearest Neighbors model with the best AUC is: '+str(highestAUC_model)+'['+str(highestAUC_score)+']\n')
```

```
K-Nearest Neighbors model with the best AUC is: knn_s_w_cv[0.96137546]
```



| Model Name | Model Description | AUC |
|---|---|---|
| **Decision Tree** | | |
| dt_wt_cv | Without sampling | 0.9239 |
| dt_s_wt_cv | With sampling | 0.9126 |
| dt_s_w_rs | RandomizedSearchCV | 0.9022 |
| dt_s_boost | AdaBoostClassifier | 0.8976 |
| dt_s_bag | BaggingClassifier | 0.9871 |
| dt_s_w_cv | GridSearchCV | 0.9022 |
| dt_s_w_rs_boost | RandomizedSearchCV with AdaBoostClassifier | 0.9953 |
| dt_s_w_rs_bag | RandomizedSearchCV with BaggingClassifier | 0.9882 |
| **Random Forest** | | |
| rf_wt_cv | Without sampling | 0.9956 |
| rf_s_wt_cv | With sampling | 0.9918 |
| rf_s_w_rs | RandomizedSearchCV | 0.9876 |
| **Naïve Bayes** | | |
| mnb_s | With sampling - Mutinomial Naïve Bayes (categorical) | 0.6544 |
| gnb_s_wt_cv | With sampling - Gaussian Naïve Bayes (numerical) | 0.7019 |
| gnb_s_w_rs | RandomizedSearchCV | 0.7020 |
| **Logistic Regression** | | |
| lr_s_wt_cv | With sampling | 0.7976 |
| lr_s_w_rs | RandomizedSearchCV | 0.8034 |
| lr_s_boost | AdaBoostClassifier | 0.7881 |
| lr_s_bag | BaggingClassifier | 0.7970 |
| lr_s_wt_scale | Without scaling | 0.8016 |
| **K-Nearest Neighbors** | | |
| knn_s_wt_cv | With sampling | 0.9485 |
| knn_s_w_cv | GridSearchCV | 0.9614 |
| **Voting** | | |
| vc3 | Ensemble of lr_s_wt_cv, lr_s_w_rs, and knn_s_wt_cv | 0.8879 |

Based on AUC, the three best models for predicting Attrition are Random Forest, Random Forest with Sampling, and Decision Tree using RandomizedSearchCV with AdaBoostClassifier.

# 7.0 Model Recommendation

## 7.1 Model Selection

### Accuracy

The accuracy of the model refers to the ratio of the number of correct classifications predicted by the model to the total number of predictions. It is easy to understand and provides an efficient overview of the model's performance based on a given dataset—the higher the accuracy, the better the classifier. Accuracy is indeed an intuitive evaluation statistics measurement, but sometimes a high accuracy does not mean an algorithm is good, especially when the dataset contains imbalanced classes. For example, the dataset has 1000 sample observations, 999 records are "No", and only 1 record is "Yes". Even if the model has an accuracy of 99%, when the "Yes" happens, and the algorithm fails to predict correctly, the loss or the impact could still be huge.

The dataset in this project contains 4410 sample observations, of which about 16% of the employees are labeled as attrition ("Yes"). As Human Resources will pay more attention to the employees who are likely to leave due to the cost incurred from attrition, Precision, Recall, and F1 Score will be used to measure the model's performance.

### Precision

Precision, in this case, is the proportion of real attrition over all the sample observations that have been predicted as Yes.

### Recall

Recall is the proportion of correctly predicted attritions over all the attrition.

### F1 Score

In other words, Precision can be described as a measurement of the extent of the error caused by False Positive, whereas Recall measures the extent of the error caused by False Negative (LT, 2021). F1 Score measures the model's predictive performance by combining Precision

and Recall to maximize both measurements.

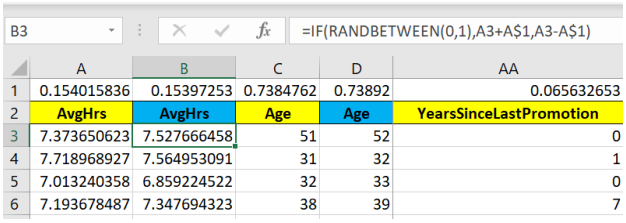
Based on the F1 Score, the best model for this project is the Decision Tree using RandomizedSearchCV with AdaBoostClassifier. However, the training time of this model is too long, which leads to a high cost to the company. Therefore, the next best model, the Random Forest with Sampling is selected. This model has a shorter training time but performs as well as the prior model.

| Model Name | Model Description | Accuracy (Train) | Accuracy (Test) | Precision | Recall | F1-Score |
|---|---|---|---|---|---|---|
| **Decision Tree** | | | | | | |
| dt_wt_cv | Without sampling | 1.0000 | 0.9615 | 0.8894 | 0.8685 | 0.8789 |
| dt_s_wt_cv | With sampling | 1.0000 | 0.9456 | 0.8106 | 0.8638 | 0.8364 |
| dt_s_w_rs | RandomizedSearchCV | 0.9992 | 0.9327 | 0.7605 | 0.8498 | 0.8027 |
| dt_s_boost | AdaBoostClassifier | 1.0000 | 0.9395 | 0.7982 | 0.8357 | 0.8165 |
| dt_s_bag | BaggingClassifier | 1.0000 | 0.9743 | 0.9324 | 0.9061 | 0.9190 |
| dt_s_w_cv | GridSearchCV | 0.9992 | 0.9327 | 0.7605 | 0.8498 | 0.8027 |
| dt_s_w_rs_boost | RandomizedSearchCV with AdaBoostClassifier | 1.0000 | 0.9879 | 0.9950 | 0.9296 | 0.9612 |
| dt_s_w_rs_bag | RandomizedSearchCV with BaggingClassifier | 1.0000 | 0.9743 | 0.9366 | 0.9014 | 0.9187 |
| **Random Forest** | | | | | | |
| rf_wt_cv | Without sampling | 1.0000 | 0.9841 | 1.0000 | 0.9014 | 0.9481 |
| rf_s_wt_cv | With sampling | 1.0000 | 0.9849 | 0.9849 | 0.9202 | 0.9515 |
| rf_s_w_rs | RandomizedSearchCV | 0.9996 | 0.9796 | 0.9559 | 0.9155 | 0.9353 |
| **Naïve Bayes** | | | | | | |
| mnb_s | With sampling - Mutinomial Naïve Bayes (categorical) | 0.7607 | 0.7181 | 0.2826 | 0.4883 | 0.3580 |
| gnb_s_wt_cv | With sampling - Gaussian Naïve Bayes (numerical) | 0.6871 | 0.6871 | 0.2829 | 0.6150 | 0.3876 |
| gnb_s_w_rs | RandomizedSearchCV | 0.6871 | 0.6871 | 0.2829 | 0.6150 | 0.3876 |
| **Logistic Regression** | | | | | | |
| lr_s_wt_cv | With sampling | 0.8241 | 0.8088 | 0.4324 | 0.6009 | 0.5029 |
| lr_s_w_rs | RandomizedSearchCV | 0.8239 | 0.8209 | 0.4574 | 0.6056 | 0.5212 |
| lr_s_boost | AdaBoostClassifier | 0.8190 | 0.7906 | 0.3961 | 0.5728 | 0.4683 |
| lr_s_bag | BaggingClassifier | 0.8237 | 0.8103 | 0.4354 | 0.6009 | 0.5049 |
| lr_s_wt_scale | Without scaling | 0.8681 | 0.8413 | 0.5080 | 0.4460 | 0.4750 |
| **K-Nearest Neighbors** | | | | | | |
| knn_s_wt_cv | With sampling | 0.9424 | 0.7748 | 0.4134 | 0.9531 | 0.5767 |
| knn_s_w_cv | GridSearchCV | 1.0000 | 0.9690 | 0.8644 | 0.9577 | 0.9087 |
| **Voting** | | | | | | |
| vc3 | Ensemble of lr_s_wt_cv, lr_s_w_rs, and knn_s_wt_cv | 0.8874 | 0.8541 | 0.5352 | 0.7136 | 0.6117 |

It is worth mentioning that almost all the tree-based algorithms have excellent performance, as the table shows above. Most of these models achieve 100% accuracy on the training dataset and also a relatively high accuracy on the test dataset. This result may indicate a risk of overfitting. Therefore, a synthetic test dataset is created and then run through the model to see if the accuracy is significantly decreased to verify whether these models are at risk of overfitting. If there is a significant decrease in the accuracy of this synthetic dataset compared to the test dataset, then the model is overfitting. However, if there is no significant difference between the statistics, then the model does not have the risk of overfitting, which also mean that the model is capable of producing high-quality predictive

outcomes.

Excel is used to create the synthetic dataset. Each record in the four variables, AvgHrs, Age, WkDays, and MonthlyIncomeCAD, is randomly added or subtracted by 1% of the mean of that variable. Using the random function can ensure that the mean of the synthesized variables remains basically the same as the original ones without changing the variables' characteristics.




| | A | B | C | D | AA |
|---|---|---|---|---|---|
| 1 | 0.154015836 | 0.15397253 | 0.7384762 | 0.73892 | 0.065632653 |
| 2 | AvgHrs | AvgHrs | Age | Age | YearsSinceLastPromotion |
| 3 | 7.373650623 | 7.527666458 | 51 | 52 | 0 |
| 4 | 7.718968927 | 7.564953091 | 31 | 32 | 1 |
| 5 | 7.013240358 | 6.859224522 | 32 | 33 | 0 |
| 6 | 7.193678487 | 7.347694323 | 38 | 39 | 7 |

The accuracy of the synthetic data after running through the model is slightly higher than the test dataset, so the model is considered not at the risk of overfitting.

| Model Name | Model Description | Accuracy (Train) | Accuracy (Test) | Accuracy (Syn) |
|---|---|---|---|---|
| **Random Forest** | | | | |
| rf_wt_cv | Without sampling | 1.0000 | 0.9841 | 0.9900 |
| rf_s_wt_cv | With sampling | 1.0000 | 0.9849 | 0.9900 |
| rf_s_w_rs | RandomizedSearchCV | 0.9996 | 0.9796 | 0.9900 |

## 7.2 Model Theory

Random Forest is a supervised machine learning algorithm. It builds a number of Decision Trees based on the parameter "n_estimators" defined in the RandomForestClassifier and takes the majority vote from all the trees for classification. Random Forest is trained with the "bagging" technique, which is one of the ensemble techniques. Refer to section 6.3 for further details on Random Forest.

## 7.3 Model Assumptions and Limitations

**Assumptions**

- Can handle skewed, multi-modal, and categorical data
- Can handle a large number of variables
- No missing values in the data

- No drifting in the data
- No changes in data structure
- Low correlations among the predictions from all trees

**Limitations**

- Have a risk of overfitting, especially when the dataset is noisy
- Bias towards the categorical variables with more levels than those with fewer levels
- Feature importance may not be reliable when handling categorical variables with different levels
- Can not interpret the positive/negative relationship between the features and the target variable

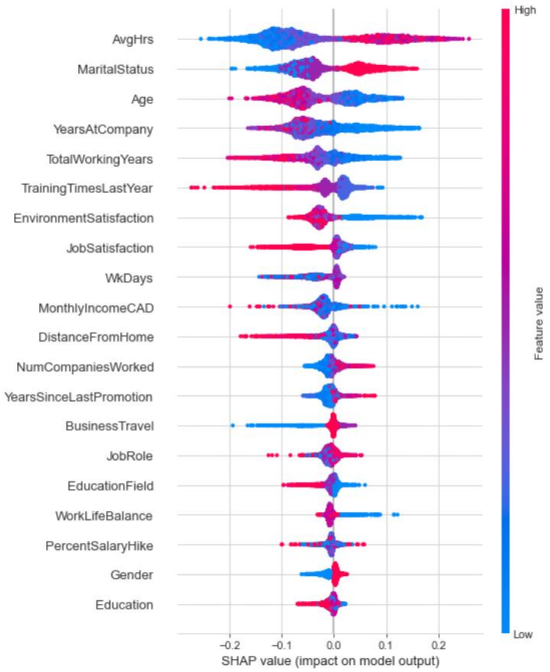
### 7.4 Model Sensitivity to Key Drivers

Random Forest can generate feature importance scores, which measure the contribution of each feature to the target variable. The score for a specific feature is calculated by summing up the Gini index decrease for that feature over all the trees. Below are the top ten important features ranked in descending order:

| Features | Importance |
|---|---|
| AvgHrs | 0.132259 |
| MonthlyIncomeCAD | 0.081786 |
| Age | 0.079374 |
| TotalWorkingYears | 0.072839 |
| YearsAtCompany | 0.070197 |
| MaritalStatus | 0.053771 |
| DistanceFromHome | 0.049609 |
| TrainingTimesLastYear | 0.046643 |
| EnvironmentSatisfaction | 0.038154 |
| PercentSalaryHike | 0.036001 |

Based on the above feature importance table, AvgHrs has the highest importance score, followed by MonthlyIncomeCAD, Age, TotalWorkingYears, and YearsAtCompany. The importance scores for the other features are considerably lower.

Random Forest is not able to generate a tree-like diagram like Decision Tree. Therefore, the findings are not as easily interpreted as the tree. In order to be able to explain the features and their impact on the target variable, SHAP value (SHapley Additive exPlanations) is applied. This method can increase the interpretability of any machine learning model. The beewarm plot below is created based on the SHAP values. It shows both positive and negative relationships between the features and the target variable. Meanwhile, colors are used to differentiate the feature value, red being high and blue being low.




| Features | Importance | Impact To Attrition |
|---|---|---|
| **AvgHrs** | 0.132259 | Positive |
| **MonthlyIncomeCAD** | 0.081786 | Negative - low |
| **Age** | 0.079374 | Negative |
| **TotalWorkingYears** | 0.072839 | Negative |
| **YearsAtCompany** | 0.070197 | Negative |
| **MaritalStatus_Single** | 0.053771 | Positive |
| **DistanceFromHome** | 0.049609 | Negative |
| **TrainingTimesLastYear** | 0.046643 | Negative |
| **EnvironmentSatisfaction** | 0.038154 | Negative |
| **PercentSalaryHike** | 0.036001 | Negative - low |

# 8.0 Validation and Governance

## 8.1 Introduction

Models can produce different results in production over time due to the distribution change in the features or degradation of models' performance. The company needs an effective model validation and governance to ensure a good return on the machine learning investment. The model validation and governance include processes such as monitoring the model's performance and results, setting access controls for models, implementing the right documentation, and more. Through all these processes, the company can better understand and control the features that will be used in the model. Not only does model validation and governance allow the company to recognize and reduce the potential risks of machine learning, but they also increase the models' performance in production. Drifting in models can be identified quickly, thus preventing the company from making biased decisions due to the models' inaccuracy (DataRobot, 2020).

## 8.2 Variable Level Monitoring

Variable Level Monitoring is one of the model monitoring activities in which the features' characteristics and quality are observed over time. For numerical features, descriptive statistics can be used to check if there is any difference in mean, median, and standard deviation between features involved in the model training and features in the future.

| Descriptive Statistics | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|
| **AvgHrs** | 7.7 | 1.3 | 6.0 | 6.7 | 7.4 | 8.4 | 11.0 |
| **Age** | 36.9 | 9.1 | 18.0 | 30.0 | 36.0 | 43.0 | 60.0 |
| **DistanceFromHome** | 9.2 | 8.1 | 1.0 | 2.0 | 7.0 | 14.0 | 29.0 |
| **NumCompaniesWorked** | 2.7 | 2.5 | 0.0 | 1.0 | 2.0 | 4.0 | 9.0 |
| **PercentSalaryHike** | 15.2 | 3.7 | 11.0 | 12.0 | 14.0 | 18.0 | 25.0 |
| **TotalWorkingYears** | 11.3 | 7.8 | 0.0 | 6.0 | 10.0 | 15.0 | 40.0 |
| **TrainingTimesLastYear** | 2.8 | 1.3 | 0.0 | 2.0 | 3.0 | 3.0 | 6.0 |
| **YearsAtCompany** | 7.0 | 6.1 | 0.0 | 3.0 | 5.0 | 9.0 | 40.0 |
| **YearsSinceLastPromotion** | 2.2 | 3.2 | 0.0 | 0.0 | 1.0 | 3.0 | 15.0 |
| **YearsWithCurrManager** | 4.1 | 3.6 | 0.0 | 2.0 | 3.0 | 7.0 | 17.0 |
| **WkDays** | 236.3 | 5.5 | 225.0 | 232.0 | 236.0 | 241.0 | 248.0 |
| **MonthlyIncomeCAD** | 1042.9 | 755.2 | 161.0 | 467.0 | 788.5 | 1344.0 | 3208.0 |

Furthermore, the acceptable range for each feature is represented by the range between min and max. If the future values are out of the acceptable range and being identified as outliers, Caps and Floors will be applied so that all values are kept within the min and max. Same as the current approach, missing values in the future will be handled by using KNN imputation technique, which takes the observation values for neighboring data points. For categorical features, bar charts can be created to identify the change in the distribution of categories over time. For imputing missing values in categorical features, the categorical values will be first encoded into numerical values before going through KNN imputation.

### *8.2.1 Variable Drift Monitoring*

The models developed in this project provide the company an in-depth insight into which key factors that may cause the employees to leave and who is most likely to leave. The results from the analysis help the company to take appropriate actions that will address the at-risk employees in a proactive way. However, the features used for model training may drift or lose statistical significance a few months or years later due to various reasons.

Not only the accuracy of the analysis, but data drift can refer to the features' relevance to the analysis as well. For example, DistanceFromHome may not be a relevant feature for employee attrition analysis after the COVID-19 pandemic, as most people have been working in hybrid mode since then. If the employees spend more time working from home, rating for EnvironmentSatisfaction may not be suitable to be included in the employee survey anymore. Therefore, the statistical significance of both DistanceFromHome and EnvironmentSatisfaction has the potential to change over time.

To ensure the profitability of the company, all features are expected to remain predictive and work well in the models all the time. Data drift must be monitored closely as it will eventually lead to model drift.

### *8.2.2 Tolerance for Drift of Each Variable*

Drift tolerance for the more important features is set at 7% of the mean of the features used in model training, while drift tolerance for the less important features is higher, which is 10%. When the drift of a feature is more than the threshold, instead of rebuilding the models directly, model health and stability will be first assessed by looking into the AUC-ROC and F1-score. Then, appropriate actions will be taken based on the risk tiering.

## 8.3 Model Health & Stability

As more companies adopt data-driven business models, the concept of machine learning and artificial intelligence is rapidly gaining popularity across a wide range of industries. Companies are using models to make business decisions with the power of data analysis. Therefore, it is essential to create a robust review and risk management mechanism for the underlying models to ensure the quality, accuracy, stability, and validity of business decisions built on these models (Farnworth, 2020). In addition, it is not difficult to imagine that the models built by companies are based on the internal and external environment and conditions at that time. However, once the model is deployed in a production environment, if the inputs or any assumptions change, the effectiveness of the model will decrease, and bad business decisions can be made due to inaccurate model predictions, which can be financially costly to the business. Furthermore, as technology, corporate compliance, and business requirements and needs become more complex, model validation and governance have become increasingly important over the years (Deloitte, 2017).

According to Deloitte (2017), a proper model risk management framework can be built on four essential perspectives:

1) **Model Oversight.** The company should have strict approval and review procedures for conducting modeling activities and ensure that different phases of reviews are not independent of each other, such as algorithms or coding sections should be integrated

with the assessment of the model. More importantly, the framework must have the support of the management and stakeholders. During the process of monitoring and validation, the following aspects should be included:

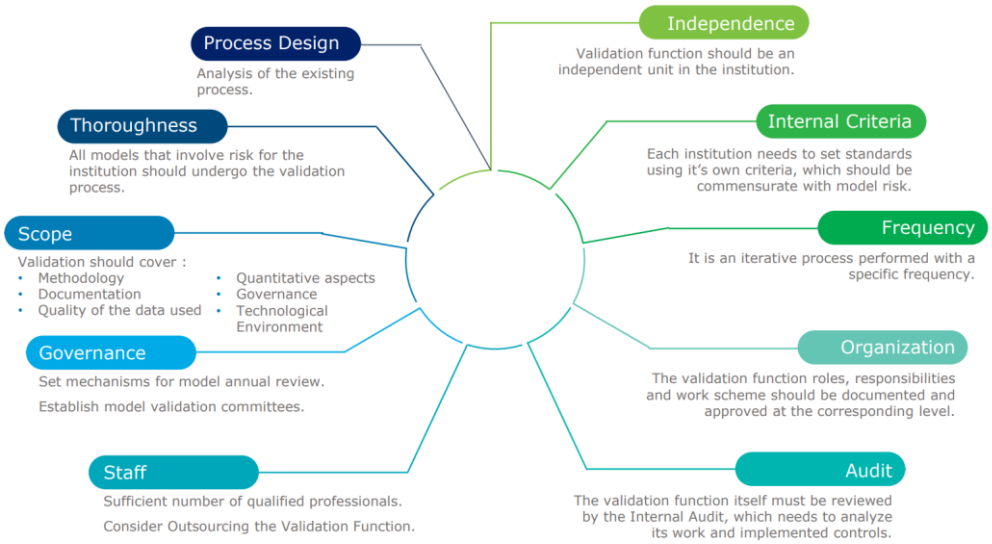



*Figure 8 Model Control Framework*

Source: Deloitte
Retrieved from: https://www2.deloitte.com/content/dam/Deloitte/fr/Documents/risk/deloitte_model-risk-management_plaquette.pdf

2) **Model Standards.** Companies can refer to industry standards for model development, implementation, and application. Model development includes:
    - The correct use of model design and code
    - The completeness, accuracy, and relevance of data
    - The validation of model assumptions and limitations
    - The maintenance of model documentation
3) **Model Validation.** To ensure that the model can produce accurate business predictions, companies should perform qualitative and quantitative assessments of the model's performance before and after model implementation.
4) **Risk Culture.** The company's senior management should support and foster a risk culture supported by a model risk management framework.

### *8.2.1 Initial Model Fit Statistics*

The model risk management framework above mentions that companies should periodically perform qualitative and quantitative assessments of the model's performance to ensure the effectiveness of the model predictions and the business decisions. In order to better evaluate the health of algorithms and monitor the stability of the model over time, the Area Under the Curve (AUC) of the Receiver-Operator-Characteristic (ROC) curve and F1 Score are selected as the model diagnostics and performance indicators and benchmarks.

**AUC-ROC**

AUC-ROC is a useful technique in the assessment of the performance of the models. Refer to section 6.8 for further details on AUC-ROC.

The selected model in this project, the Random Forest with Sampling, has an AUC of 0.9918.

**F1 Score**

F1 Score measures the model's predictive performance by combining Precision and Recall. Refer to section 7.1 for further details on the F1 Score.

The selected model in this project has an F1 Score of 0.9515.

### *8.2.2 Risk Tiering*

The ongoing model monitoring and validation are critical to identifying whether the model will be able to continue to perform and deliver accurate business predictions as expected. The company needs to re-evaluate the model and take actions depending on the risk tiering criteria when the following situations occur (Deloitte, 2017):

- Changes in the market
- Changes in the products or portfolio
- Change in the model risk standards
- Changes in regulatory and audit standards
- Model performance deterioration

Risk Tiering aims to classify potential harm and suggest the corresponding action. Some companies use a decision tree approach, as shown below, to determine what action to take.

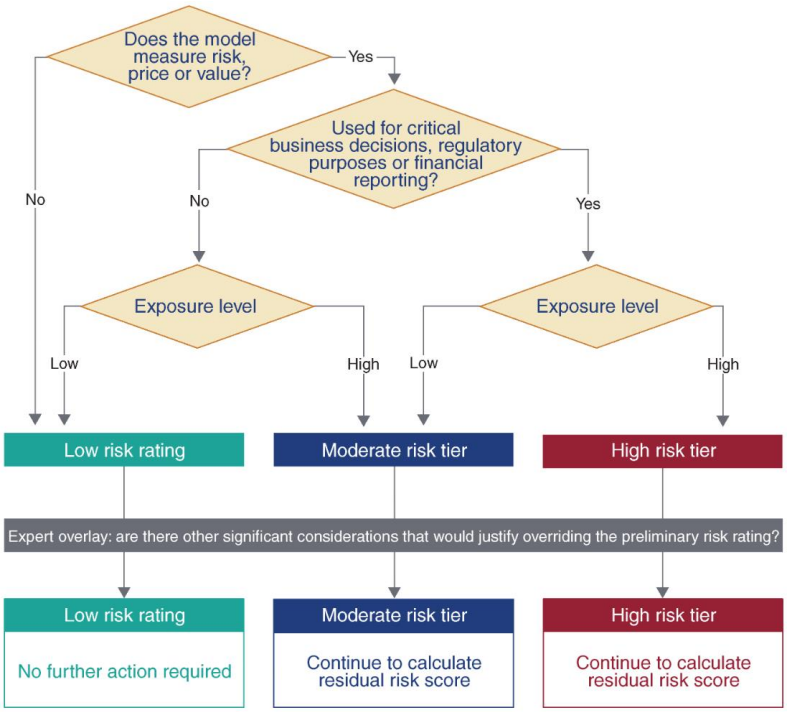



*Figure 9 Model Risk Tree*

Source: Risk.net
Retrieved from: https://www.risk.net/journal-of-risk-model-validation/6710566/model-risk-tiering-an-exploration-of-industry-practices-and-principles

In this project, AUC-ROC and F1 Score are selected as the model performance benchmarks. If the measurements have a 1% to 10% drift from the current values, to ensure the model is able to continue to implement, actions will be taken based on the risk tiering.

| | **Current Value** | **Acceptable Minimum** |
|---|---|---|
| **AUC-ROC** | 0.9918 | 0.9 |
| **F1 Score** | 0.9515 | 0.85 |

- **Low risk tier.** No intervention will be needed if the drift is less than 2%.
- **Moderate risk tier.** If the drift is between 2% and 5%, optimization techniques, such as hyperparameter tuning, ensemble learning methods, principal component analysis, and more, can be used to tune the model.

- **High risk tier.** If the drift is between 6% and 10%, the model will be refitted with the new dataset as it may indicate changes in the data over time.
- **Unacceptable risk tier.** When the drift is over 10%, the model will be rebuilt as it may indicate a structural change in the data, and the model will no longer be able to make predictions.

# 9.0 Conclusion and Recommendations

## 9.1 Impacts on Business Problem

As a HR tech company, we are committed to providing valuable analytics insights and strategies to help companies optimize their human resources management and operations, thus bringing benefits to the company. Employee attrition is a costly issue for all companies across industries. The cost of replacing employees is much higher than the cost of retaining them. This project has focused on building machine learning algorithms to find the best model to effectively predict employee attrition and analyze the key factors causing the attrition. Therefore, management in the company will be able to act proactively and reduce the potential cost. A strategic retention plan can be drawn based on the following aspects:

- **Average Working Hours.** The longer the average working hours, the more likely people tend to leave the company. Therefore, the company, especially business units, should identify work priorities, design the scope of projects reasonably and validate the effectiveness of each task while ensuring the schedule and quality of work.
- **Age, Total Working Years, and Year at Company.** The younger the employee and the shorter the length of service, the more likely they are to leave the company. Young talents can bring fresh blood to the company and provide more valuable innovative ideas. Therefore, companies should adopt new strategies to retain young talents, such as focusing on investing in their future instead of monetary incentives.
- **Training Times.** Employees who have fewer training times are more likely to leave. Training is an effective way to increase employee productivity and loyalty to companies. It can not only improve positive communication between staff and managers but also allows employees to feel that they are being valued, helping the company build a committed and productive workforce.
- **Environment Satisfaction.** The lower the environment satisfaction, the more likely

employees are to leave. The work environment mainly refers to the elements that can impact daily work activities. It includes the physical work environment, such as job security, office equipment, and even the location of the office building, and the non-physical work environment, including supervisor relationships, encouragement and career development, corporate culture, and more. Companies can invest in the physical environment, for example, by using Smart Office systems to support employees' daily tasks. It can not only bring a sense of hi-tech to the workplace but also reflect the care for employees and corporate social responsibility. In addition, management should ensure a positive, friendly, and up-to-date corporate culture to attract and retain talents.

- **Percent Salary Hike and Monthly Income.** The lower the pay, the more likely the employees will leave the company. Compensation management is one of the most fundamental strategies that companies can adopt to reduce employee attrition. To ensure compensation is in line with or above the market, the company can measure its salary standards by looking at its competitors' compensation levels. The company should also publicly rewards employees based on their skills, abilities, and achievements.

## 9.2 Future Work

On the one hand, the source of the current dataset is limited. In the future, sample observations can be collected from more sources using a wide range of data collection methods. In addition to increasing the sample observations, more features can be added to the analysis, such as Voluntary/Non-voluntary Attrition and Attrition Cost by Position, bringing more analytics insights to the project. Consequently, the employee attrition analysis can become more meaningful and comprehensive.

On the other hand, due to time constraints, this project only applied a limited number

of machine learning algorithms to make predictions. In the future, more techniques can be applied to find models with better quality. For example, Neural Network algorithm is a powerful tool in predictive analytics as it can learn from complex situations and provide higher accuracy and more remarkable results. Although Neural Network is commonly considered a black box and lacks interpretability, traditional machine learning algorithms, such as Decision Trees or Logistic Regression, can be built based on the results of a Neural Network. By doing so, the outcomes can be explained clearly to the stakeholders.

## References

Analytics Vidhya. (2022, July 19). *10 Techniques to deal with Imbalanced Classes in Machine Learning*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/

AON. (2015, June). *In a Competitive Market, Biotech Firms Proving Effective in Retaining Employees.* Retrieved from AON: https://humancapital.aon.com/insights/articles/2015/biotech-firms-proving-effective-in-retaining-employees

Bennett, C. (2015, February 28). *HR and people analytics*. Retrieved from Deloitte: https://www2.deloitte.com/us/en/insights/focus/human-capital-trends/2015/people-and-hr-analytics-human-capital-trends-2015.html

Boyer, J. (n.d.). *Prepare data for machine learning*. Retrieved from IBM: https://www.ibm.com/garage/method/practices/reason/prepare-data-for-machine-learning/

Brownlee, J. (2019, December 23). *A Gentle Introduction to Imbalanced Classification*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/what-is-imbalanced-classification/

Brownlee, J. (2020, May 18). *How to Use Power Transforms for Machine Learning*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/power-transforms-with-scikit-learn/

Chauhan, N. (2022, February 9). *Decision Tree Algorithm, Explained*. Retrieved from KD Nuggets: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

Chauhan, N. S. (2022, April 8). *Naïve Bayes Algorithm: Everything You Need to Know*.

Retrieved from KD nuggets: https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html

DataRobot. (2020, September 11). *What is Model Governance?* Retrieved from DataRobot: https://www.datarobot.com/blog/what-is-model-governance/

Datasciencelovers. (2020, March 28). *K-Nearest Neighbors (KNN) – Theory*. Retrieved from Datasciencelovers: http://www.datasciencelovers.com/machine-learning/k-nearest-neighbors-knn-theory/

Deloitte. (2017). *Model risk management: A practical approach*. Retrieved from Deloitte: https://www2.deloitte.com/us/en/pages/financial-services/articles/model-risk-management.html

Donges, N. (2021, July 22). *Random Forest Algorithm: A Complete Guide*. Retrieved from Built In: https://builtin.com/data-science/random-forest-algorithm

Dunkley-Hickin, K. (2021, October 28). *An introduction to decision tree theory*. Retrieved from Precision Analytics: https://www.precision-analytics.ca/articles/an-introduction-to-decision-tree-theory/

Engel, A. (2022, March 18). *From Numerical to Categorical*. Retrieved from Towards Data Science: https://towardsdatascience.com/from-numerical-to-categorical-3252cf805ea2

Farnworth, R. (2020, November 15). *What is Model Governance?* Retrieved from Towards Data Science: https://towardsdatascience.com/model-governance-cc907d9b1111

GeeksforGeeks. (2020, September 2). *Advantages and Disadvantages of Logistic Regression*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/

Google. (2022, July 18). *Machine Learning*. Retrieved from Google Develpers: https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data

Great Learning. (2020, September 29). *Hyperparameter Tuning with GridSearchCV*. Retrieved from Great Learning: https://www.mygreatlearning.com/blog/gridsearchcv/#what-is-gridsearchcv

Heavy.AI. (n.d.). *Heavy.AI*. Retrieved from Feature Engineering: https://www.heavy.ai/technical-glossary/feature-engineering

IBM Cloud Education. (2020, December 7). *Random Forest*. Retrieved from IBM: https://www.ibm.com/cloud/learn/random-forest

Kumar, A. (2021, October 1). *Bagging Classifier Python Code Example*. Retrieved from Data Analytics: https://vitalflux.com/bagging-classifier-python-code-example/#Introduction_to_Bagging_Bagging_Classifier_Regressor

Kumar, S. (2021, October 26). *Use Voting Classifier to improve the performance of your ML model*. Retrieved from Towards Data Science: https://towardsdatascience.com/use-voting-classifier-to-improve-the-performance-of-your-ml-model-805345f9de0e

Kurama, V. (2020). *A Guide to AdaBoost: Boosting To Save The Day*. Retrieved from Paperspace Blog: https://blog.paperspace.com/adaboost-optimizer/

Lekhtman, A. (2019, August 5). *Data Science in Medicine — Precision & Recall or Specificity & Sensitivity?* Retrieved from Towards Data Science: https://towardsdatascience.com/should-i-look-at-precision-recall-or-specificity-sensitivity-3946158aace1

Lesser, E., & DeBellis, P. (2022, February 24). *What Holds Your People Analytics Strategy Back?* Retrieved from Deloitte: https://www2.deloitte.com/us/en/blog/human-capital-blog/2022/people-analytics-strategy.html

LT, Z. (2021, November 23). *Essential Things You Need to Know About F1-Score*. Retrieved from Towards Data Science: https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score-

dbd973bf1a3#:~:text=By%20definition%2C%20F1%2Dscore%20is,for%20both%20FPs%20and%20FNs.

Mainkar, G. (2018, July 5). *Understanding the impact of attrition on an organization*. Retrieved from Harbinger: https://harbinger-systems.com/blog/2018/07/understanding-the-impact-of-attrition-on-an-organization/

Mirbozorgi, B. (2020, January 7). *Theory of K-Nearest Neighbors (KNN)*. Retrieved from Medium: https://medium.com/analytics-vidhya/theory-of-k-nearest-neighbors-knn-234654826c50

MLNerds. (2019, February 14). *How does KNN algorithm work ? What are the advantages and disadvantages of KNN ?* Retrieved from Machine Learning Interviews: https://machinelearninginterview.com/topics/machine-learning/how-does-knn-algorithm-work-what-are-the-advantages-and-disadvantages-of-knn/

Obadia, Y. (2017, January 31). *The use of KNN for missing values*. Retrieved from Towards Data Science: https://towardsdatascience.com/the-use-of-knn-for-missing-values-cf33d935c637

Pranto, B. (2020, January 2). *Entropy Calculation, Information Gain & Decision Tree Learning*. Retrieved from Medium: https://medium.com/analytics-vidhya/entropy-calculation-information-gain-decision-tree-learning-771325d16f

Prinsloo, K. (2022, June 9). *Advanced Exploratory Data Analysis (EDA) in Python*. Retrieved from Medium: https://kevinprinsloo.medium.com/advanced-eda-e6fea0193dbd

Ray, S. (2017, September 11). *6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

Roy, B. (2020, April 6). *All about Feature Scaling*. Retrieved from Towards Data Science: https://medium.com/towards-data-science/all-about-feature-scaling-bcc0ad75cb35

Scikit-Learn. (2022). *Decision Trees*. Retrieved from Scikit-Learn: https://scikit-learn.org/stable/modules/tree.html

Scikit-Learn. (2022). *sklearn.naive_bayes.MultinomialNB*. Retrieved from Scikit-Learn: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

Shmueli, G., Bruce, P., Gedeck, P., & Patel, N. (2020). *Data Mining for Business Analytics.* John Wiley & Sons, Inc.

Sisense. (n.d.). *Data Exploration*. Retrieved from Sisense: https://www.sisense.com/glossary/data-exploration/

Statista. (2021). *HR Tech - A Statista Dossierplus on the Global Market for Digitalized Human Resources Solutions*. Retrieved from Statista: https://www.statista.com/study/85007/hr-tech/

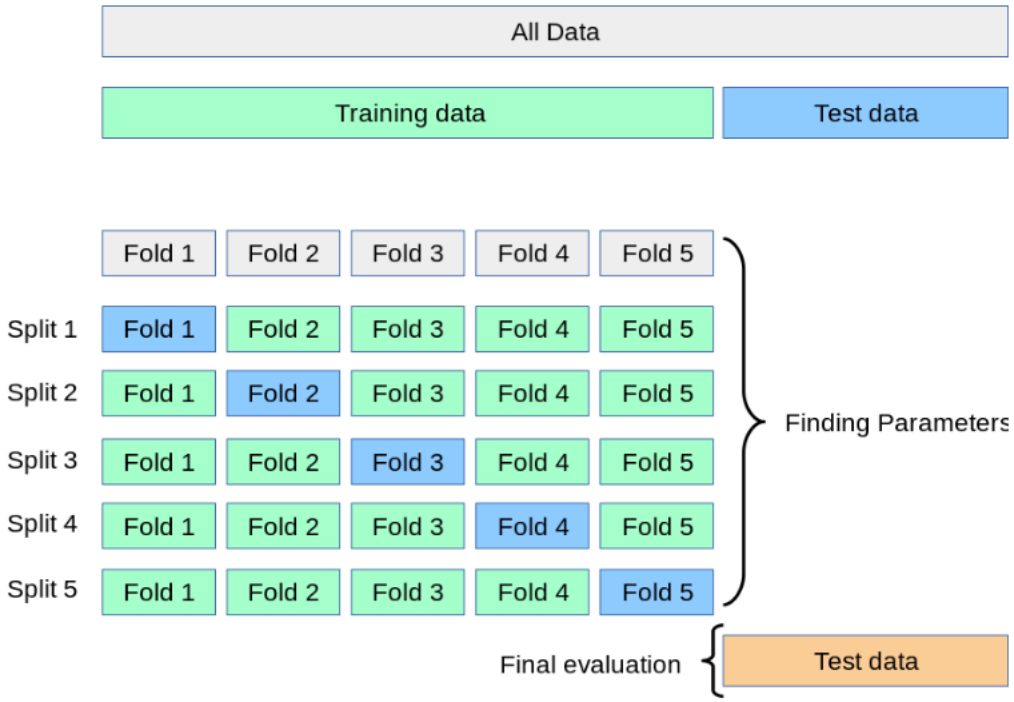
Swaminathan, S. (2018, March 15). *Logistic Regression — Detailed Overview*. Retrieved from Towards Data Science: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Verma, Y. (2021, September 6). *A Complete Guide to Categorical Data Encoding*. Retrieved from Analyticsindiamag: https://analyticsindiamag.com/a-complete-guide-to-categorical-data-encoding/

Worcester, P. (2019, June 5). *A Comparison of Grid Search and Randomized Search Using Scikit Learn*. Retrieved from Medium: https://medium.com/@peterworcester_29377/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85

# Appendix

Appendix A:




Source: Scikit Learn
Retrieved from: https://scikit-learn.org/stable/modules/cross_validation.html