Predicting Employee Attrition Using Machine Learning

Devika Kaladevi
Faculty of Engineering Environment and Computing
Coventry University
kaladevid@coventry.ac.uk

Aliya Anjum Azeez Ur Rahaman
Faculty of Engineering Environment and Computing
Coventry University
azeezuraa@coventry.ac.uk

Abstract*— The importance of using machine learning models for predicting employee attrition for the organization is explored in this paper. To analyze and understand the underlying patterns and factors influencing attrition, logistic regression, decision tree, random forest or support vector machines models have been used to address this. It can help organizations to take the necessary steps to retain talent personnel in a timely manner if they are able to predict employee turnover. With Python and its accompanying libraries, we have been able to create algorithms, analyze data, and evaluate model performance in a reliable and effective way.

***Keywords*— data preprocessing; employee attrition: exploratory data analysis; data imbalance; PCA; random forest; decision tree; SVM; logistic regression**

I. INTRODUCTION

Employee attrition, characterized by employees leaving the organization, poses a substantial challenge for businesses across diverse sectors. Elevated attrition rates can result in escalated recruitment expenses, the departure of key talent, and workflow disruptions, all of which can adversely affect the organization's financial performance and operational efficiency. In recent years, due to the fact that workers' quality and skills are an important factor in growth and a real competitive advantage for companies, attention has been focused on human resources. After the successful integration into sales and marketing departments, artificial intelligence is now extending its influence to guide decision-making processes related to employee management within HR departments. (fallucchi. et.al, 2020). Machine learning techniques offer powerful tools for analyzing complex datasets and making predictions based on patterns and trends. This study explores the application of logistic regression, decision trees, random forests, and support vector machines algorithms to analyze employee data and gain insights into their behavior. Through the development and evaluation of these predictive models, the aim is to identify the factors contributing to employee attrition and offer actionable insights to organizations for informed decision-making.

II. DATASET DESCRIPTION

The dataset used for the study is obtained from the Kaggle dataset Repository which contains 1471 instances and 35 attributes with no missing values from IBM Employee Attrition dataset. The dataset provided contains various attributes related to employees within an organization. These attributes encompass demographic information such as age, gender, marital status, as well as details about their employment, including job role, department, and distance from home. It captures aspects related to compensation and career progression, including hourly rate, monthly income, job level, years at the company, years in current role, years since last promotion, and years with the current manager. The dataset was meticulously chosen to ensure its pertinence in predicting both employee attrition or not. During the collection of data ethical consideration and data protection protocols were followed. This dataset holds an opportunity to evaluate the factors influencing the employee's attrition. Ultimately, the aim is to aid in the formulation of effective strategies to enhance attrition outcomes within the realm of business organization. The dataset contains various type of data such as numerical (continuous or discrete) categorical(nominal or ordinal) and binary variables. The variables such as daily rate, Education field etc fall into either numerical or discrete variables. So the data type in the dataset is discrete as well as categorical data. The comma separated values(CSV) formatted dataset is used to analyze using python programming languages using its libraries and tools. To understand the power of Python and its libraries, we conducted a thorough examination of the employee attrition dataset. Understanding the dataset's variables and structure is crucial for feature selection, data preprocessing, and model development processes. Feature selection plays a vital role in mitigating overfitting, enhancing accuracy, and expediting training by reducing data dimensionality and eliminating irrelevant information. The dataset utilized in this study is classified as a binary classification problem, with the target variable –attrition categorized into "yes" or "no" classes. Before analysis, several data preprocessing steps were executed.

III. Experimental Setup

A. Preprocessing

Initially, the attribute names were checked to determine whether they hold a meaningful name for interpretability. Then the dataset referred to as employee attrition was then checked for missing or null values. The first step was to check for missing/null values and found out that there are no such values in the dataset. The target named as attrition is a categorical variable then converted to numerical assigned '0' and '1' for' No' and 'Yes' respectively.

After the data preprocessing steps have been finalized, it becomes essential to evaluate the distribution and balance of classes within the target variable. This assessment is commonly carried out using Exploratory Data Analysis (EDA) techniques, which involve scrutinizing the dataset to identify patterns, trends, and relationships among the variables. The process of EDA encompasses various steps aimed at gaining a deeper understanding of the data structure and characteristics. The Steps are:

a) Counting the occuramce of target variables value Yes/ No:

Counting the occurrence is important because it provides information about the distribution of employees across the target. Occurrence creating code is mentioned un the appendix.

Attrition occurance

Yes	237
No	1233

b) Class Distribution:

To understand the distribution of employees across various categories, data visualization distribution gains an insight to the dataset to analyse upon the target.

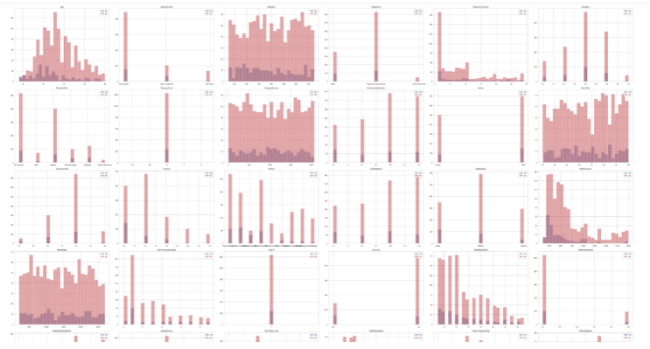


Fig 1: Class Distribution

c) Class Imbalance:

To resolve class imbalance, a number of techniques can be used, e.g. resampling methods like oversampled minority groups, under sampling the majority group or using algorithmic approaches that include cost-sensitive learning and ensembles method.

The dataset exhibits an imbalance in the target variable, where the distribution of classes for the "Attrition" feature is skewed. Upon visualizing the class distribution, it is evident that the "Yes" class, indicating attrition, is represented by 237 instances, while the "No" class, denoting no attrition, comprises 1233 instances, highlighting the class imbalance. See the appendix for the code.

Figure 2: Class Imbalance

Random Oversampling:

Given the imbalance in the dataset, addressing the minority class is essential to ensure model effectiveness. To tackle this, a combination of under sampling the majority class and employing the random oversampling technique is implemented. Random oversampling involves replicating instances from the minority class randomly to balance class distribution. This method helps alleviate the class imbalance issue by increasing the representation of the minority class, thereby allowing the model to learn from a more balanced dataset and potentially improve its performance in predicting both classes accurately.

d) Distribution of employees based on the attrition status of various instances:

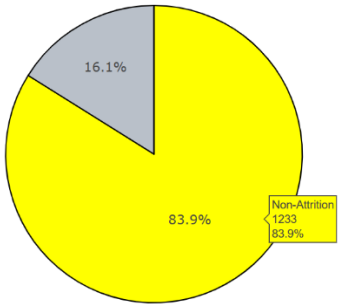


Fig 3: Distribution based on attrition status

The plot is divided into two segments: "Attrition" and "Non-Attrition." Each segment represents the proportion of employees who have experienced attrition and those who have not, respectively. The size of each segment corresponds to the number of employees falling into each category. This visualization allows for a clear understanding of the relative distribution of attrition within the dataset, providing valuable insights into the attrition patterns among employees.

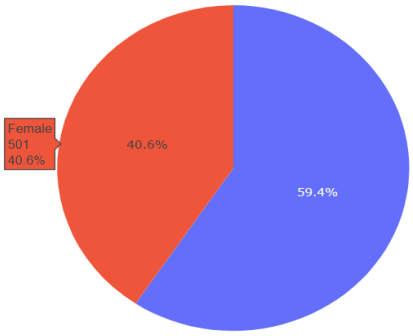


Fig 4: Distribution of attrition based on gender

The plot Fig 4 illustrates the distribution of attrition and non attrition based on the gender male and female.

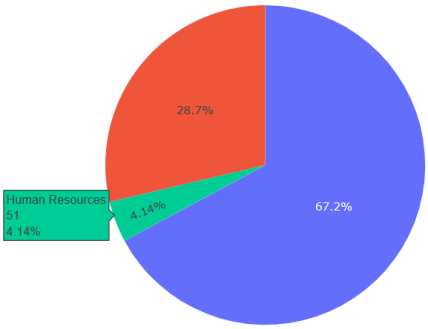


Fig 5: Distribution of attrition upon Department

The fig 5: explains the distribution of attrition of employees in various department and the specific code for this is added in appendix.

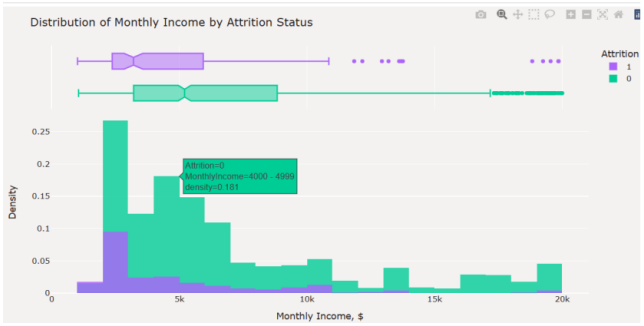


Fig 6: Distribution of monthly income by Attrition status

From fig 7: the output generated explains is a violin plot that visualizes the distribution among employees in the dataset of job satisfaction levels across different levels of education.

Each violin plot represents a combination of education level and job satisfaction. The width of the violin indicates the density of data points at various job satisfaction levels, while the shape of the violin reflects the distribution of values.

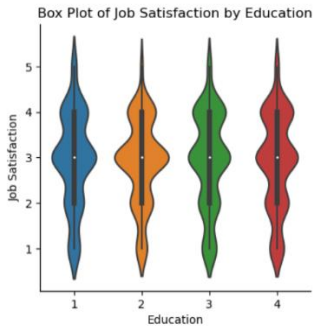


Fig 7: Box plot of job satisfaction by education

e) Dimensionality reduction and corelation:

The Figure 8 is obtained after calculates the correlation coefficients between the target variable "Attrition" and all other variables in the dataset. It then creates a correlation heatmap using seaborn and matplotlib libraries to visualize these correlations.

Variables that are strongly correlated with the target variable "Attrition." Positive correlations suggest that an increase in one variable is associated with an increase in attrition, while negative correlations suggest the opposite.

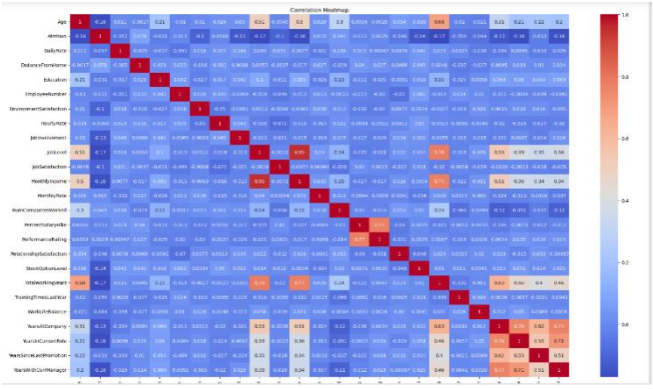


Fig 8: Corelation coefficient between target variable

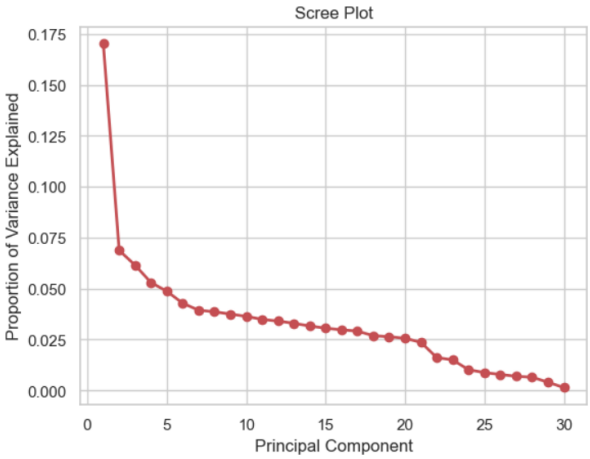


Fig 9: Explained variance Ratio by PCA

The plot obtained is Principal Component Analysis (PCA) on the training data after standardizing it using Standard Scaler. By analyzing the scree plot, we can identify the "elbow point" or the point where the explained variance

begins to level off, indicating the optimal number of principal components to retain while preserving most of the variance in the data. This information is crucial for determining the appropriate dimensionality reduction in subsequent analyses, such as building predictive models.

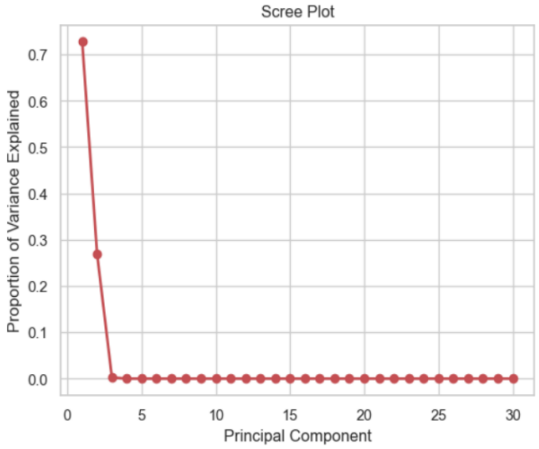


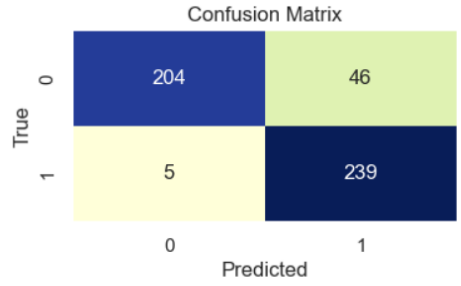
Fig 10: Scree plot for PCA

B) Model Building

To forecast employee attrition, a range of modeling techniques were utilized, such as Logistic Regression, SVM, Decision Tree, and Random Forest. To tackle the class imbalance present in the dataset, we implemented random oversampling as a balancing technique. These methodologies were employed with the aim of enhancing the performance and accuracy of the prediction models.

a) Decision tree Classifier:

The Decision Tree classifier achieves a balanced performance with high precision, recall, and F1-score for both classes, accurately predicting instances of employee attrition with an overall accuracy of 90%.

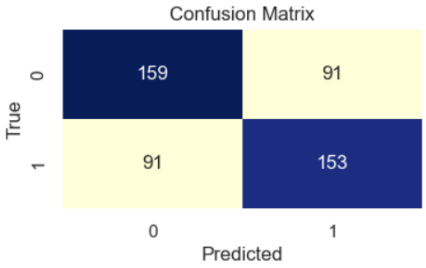


	precision	recall	f1-score	support
0	0.98	0.82	0.89	250
1	0.84	0.98	0.90	244
accuracy			0.90	494
macro avg	0.91	0.90	0.90	494
weighted avg	0.91	0.90	0.90	494

Fig 11: Confusion matrix and classification report for Decision tree

b) Logistic Regression

The Logistic Regression model achieves moderate performance, with a precision, recall, and F1-score of approximately 0.63 for both classes. Despite its balanced performance, the overall accuracy of the model is 63%.



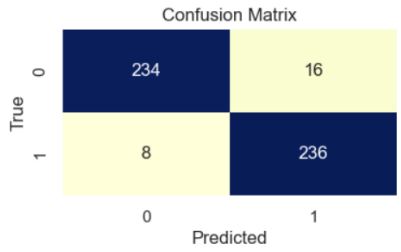
lassification Report:

	precision	recall	f1-score	support
0	0.64	0.64	0.64	250
1	0.63	0.63	0.63	244
accuracy			0.63	494
macro avg	0.63	0.63	0.63	494

Fig 12: Logistic Regression

c) Random Forest Classifier

The Random Forest classifier exhibits high performance, achieving precision, recall, and F1-scores of approximately 0.97 for class 0 and 0.94 for class 1, resulting in an overall accuracy of 95%.



Classification Report:

	precision	recall	f1-score	support
0	0.97	0.94	0.95	250
1	0.94	0.97	0.95	244
accuracy			0.95	494
macro avg	0.95	0.95	0.95	494
weighted avg	0.95	0.95	0.95	494

Fig 13: Random Forest Classifier

d) Support Vector Machine(SVM):

The output provided is from a Support Vector Machine (SVM) classifier. It includes metrics such as precision, recall, F1-score, and support for each class (0 and 1), as well as overall accuracy. In this case, the SVM model achieved an accuracy of 73%, with a precision of 83% for class 0 and 67% for class 1.

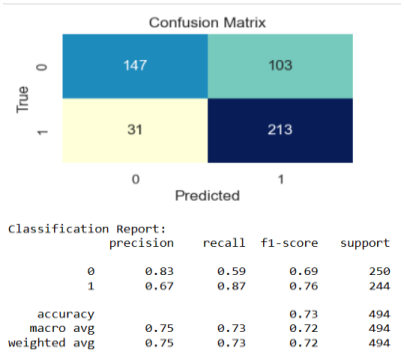


Fig 14: Support vector machine

IV. Discussion and Conclusions.

The results of the applied 4 classification metrics for the data are summarized in the table below. The precision, recall, and f1 score for both the classes are listed for easy comparison.

Model	Acc	Prec		Rec		F1	
		0	1	0	1	0	1
DT	**0.90**	**0.98**	**0.84**	**0.82**	**0.98**	**0.89**	**0.90**
LR	0.63	0.64	0.63	0.64	0.63	0.64	0.63
RF	**0.95**	**0.97**	**0.94**	**0.94**	**0.97**	**0.95**	**0.95**
SVM	0.73	0.83	0.67	0.59	0.87	0.69	0.76

Table 1: Classification scores for different models

When analyzing the classification scores for Decision Tree algorithm, it produced an accuracy of 90%, with precision scores of 0.98 for class 0 and0.83 for class 1. However, compared to the recall of 0.98 for class 1, its recall for class 0 was comparatively low at 0.81, suggesting that the model is more effective at identifying occurrences of class 1. Overall, class 0's F1-score was 0.88, while class 1's was 0.90, indicating a reasonably balanced performance.

The logistic regression approach showed an accuracy of 64% for both classes, with comparable precision and recall scores. However, the moderate F1-score of 0.64 for both classes suggests that this approach may not be the optimal fit for this dataset.

Based on the evaluation of high precision, recall, and F1-score for both classes, the Random Forest algorithm demonstrated superior performance compared to other methods with an accuracy rate of 95%, thereby exhibiting a strong and consistent performance overall. This suggests that Random Forest is a highly promising algorithm for this dataset, providing reliable predictions for both classes. As expected, the tree-based models fared well since they are known to perform well with nonlinear data. They can create more complex decision boundaries that fit well with non-linear data.

The Support Vector Machine (SVM) algorithm produced precision ratings of 0.83 for class 0 and 0.67 for class 1, resulting in an overall accuracy of 73%. The recall for class 1 was notably higher at 0.87 compared to 0.59 for class 0. Furthermore, F1 scores of 0.69 and 0.76 were obtained for class 0 and class 1, respectively, indicating a reasonably balanced performance. These results suggest that the SVM algorithm could be a suitable option for classification tasks with similar characteristics.

After evaluating four different methods for classification, the Random Forest approach was found to be the most effective, achieving a 95% accuracy rate. Additionally, it demonstrated strong precision, recall, and F1-score for both classes, indicating that it is well-suited for the dataset's classification objective and can be confidently used in real-world scenarios.

On the other hand, Decision Tree also delivered impressive results with 89% accuracy. Although SVM and logistic regression produced moderate results, they may require additional optimization or the consideration of other factors to perform better on this dataset.

V. FUTURE OPTIMIZATIONS.

It would be advantageous to explore additional techniques, such as feature engineering, hyperparameter tuning, or ensemble methods, to further enhance the performance of the models. This is particularly important for those models with lower accuracy or imbalance in class predictions. In addition, investigating the reasons behind the observed differences in performance among the algorithms could provide deeper insights into the characteristics of the dataset and the suitability of various machine-learning approaches. Such an investigation would help identify the strengths and weaknesses of the models and provide a foundation for further development and refinement.

VI. CONTRIBUTION

1. Aliya: Preprocessing, EDA, Decision Tree, and Logistic Regression, Report Writing
2. Devika: Random Forest, SVM, Random Oversampling with PCA, Report Writing

References

[1] Alao, D., & Adeyemo, A. B. (2014, January 6). Analyzing employee attrition using decision tree algorithms. *Computing, Information Systems, Development Informatics and Allied Research Journal*. Advance online publication. https://doi.org/ojs.localhost:article/10148

[2] Alduayj, S. S., & Rajpoot, K. (2018, November 1). *Predicting Employee Attrition using Machine Learning*. IEEE Xplore. https://doi.org/10.1109/INNOVATIONS.2018.8605976

[3] Fallucchi, F., Coladangelo, M., Giuliano, R., & William De Luca, E. (2020). Predicting Employee Attrition Using Machine Learning Techniques. *Computers*, *9*(4), 86. https://doi.org/10.3390/computers9040086

[4] Lobo, V., & Correia, A. (2022). *Applications of Machine Learning and Deep Learning for Privacy and Cybersecurity*. IGI Global.

[5] Nguyen, H., Vu, T., Vo, T. P., & Thai, H.-T. (2021). Efficient machine learning models for prediction of concrete strengths. *Construction and Building Materials*, *266*, 120950. https://doi.org/10.1016/j.conbuildmat.2020.120950

APPENDIX I

GitHub: https://github.com/Devikasarath77/7072CEM-Machine-Learning-Group-Devika-Aliya

Dataset: https://www.kaggle.com/code/faressayah/ibm-hr-analytics-employee-attrition performance/notebook

Employee Attrition

```
import pandas as pd
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, classification_report, confusion_matrix, Cor
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
# Read the dataset into a pandas DataFrame
hr_Attrition = pd.read_csv('HR-Employee-Attrition.csv')
```

```
hr_Attrition.shape
```

```
(1470, 35)
```

```
hr_Attrition.shape
```

```
(1470, 35)
```

Exploratory Data Analysis

In [13]:
```
# Handle missing values
hr_Attrition = hr_Attrition.dropna()
```

In [14]:
```
# To convert the target (categorical variable ) into numerical
hr_Attrition['Attrition'] = hr_Attrition['Attrition'].map({'Yes': 1, 'No': 0})
```

In [15]:
```
hr_Attrition.head(5)
```

```
In [18]: hr_Attrition[hr_Attrition['Attrition']==1].shape[0] # calculate the total no of emp
Out[18]: 237

In [19]: hr_Attrition[hr_Attrition['Attrition']==0].shape[0]
Out[19]: 1233

In [20]: grp_Attrition = hr_Attrition[hr_Attrition['Attrition']==1].mean()     # Divide work
         print(grp_Attrition)
```

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

# Assuming 'hr_Attrition' is your DataFrame containing the data
# Replace 'hr_Attrition' with the actual name of your DataFrame

# Set the style
sns.set(style="whitegrid")

# Create the plot
plt.figure(figsize=(10, 6))
sns.histplot(data=hr_Attrition, x='MonthlyIncome', hue='Attrition', element='step', kde=True, palette=['#B9C0C9', '#CDBBA7'])
plt.title('Distribution of Monthly Income by Attrition Status')
plt.xlabel('Monthly Income ($)')
plt.ylabel('Density')
plt.legend(title='Attrition', loc='upper right')
plt.show()
```

```
plot_df=hr_Attrition.sort_values(by="Attrition")
fig=px.histogram(plot_df, x='MonthlyIncome', color='Attrition',
                 opacity=0.8, histnorm='density', barmode='overlay', marginal='box',
                 color_discrete_map={'Yes': '#B9C0C9','No': '#CDBBA7'})
fig.update_layout(title_text='Distribution of Monthly Income by Attrition Status',
                  xaxis_title='Monthly Income, $', yaxis_title='Density',font_color='#28221D',
                  paper_bgcolor='#F4F2F0', plot_bgcolor='#F4F2F0', legend_traceorder='reversed')
fig.show()
```

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

# Assuming 'hr_Attrition' is your DataFrame
# Replace 'hr_Attrition' with the actual name of your DataFrame
sns.factorplot(x='JobSatisfaction', y='MonthlyIncome', data=hr_Attrition, kind='box', size=6)
plt.title('Box Plot of Job Satisfaction by MonthlyIncome')
plt.xlabel('Montly Income')
plt.ylabel('Job Satisfaction')
plt.show()
```

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

# Assuming 'hr_Attrition' is your DataFrame
# Replace 'hr_Attrition' with the actual name of your DataFrame
#Left= hr_Attrition[hr_Attrition['Attrition']==1]
sns.factorplot(x='Attrition', y='MonthlyIncome', data=hr_Attrition, kind='box', size=6)
plt.title('Box Plot of Attrition by MonthlyIncome')
plt.xlabel('Attrition')
plt.ylabel('Monthly Income')
plt.show()
```

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

# Assuming 'df' is your DataFrame containing the data
# Replace 'df' with the actual name of your DataFrame
df= pd.read_csv('HR-Employee-Attrition.csv')
# Plot distributions
k = 1
plt.figure(figsize=(60, 40))
for col in df:
    if col == "Attrition":
        continue
    yes = df[df['Attrition'] == 'Yes'][col]
    no = df[df['Attrition'] == 'No'][col]
    plt.subplot(6, 6, k)
    plt.hist(yes, bins=25, alpha=0.5, label='Yes', color='b')
    plt.hist(no, bins=25, alpha=0.5, label='No', color='r')
    plt.legend(loc='upper right')
    plt.title(col)
    k += 1

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

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

# Calculate the correlation coefficients
correlation_matrix = hr_Attrition.corr()
target_correlation = correlation_matrix['Attrition'].drop('Attrition')

# Create a heatmap
plt.figure(figsize=(24, 14))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

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

plt.figure(figsize=(2, 2))
sns.countplot(data = hr_Attrition, x="Attrition").set_title('Target')
plt.xticks(ticks=[1,0], labels=['Yes','No'])
plt.show()
```

```
sns.set(style="white", palette="muted", color_codes=True)

# Set up the matplotlib figure
f, axes = plt.subplots(3, 3, figsize=(8,8))
sns.despine(left=True)

#people that left
leavers = hr_Attrition.loc[hr_Attrition['Attrition'] == 1]

# Plot a simple histogram with binsize determined automatically
#department_mapping = {'Sales': 1, 'Human Resources': 2, 'Research & Development': 3}

#hr_Attrition['Department'] = hr_Attrition['Department'].map(department_mapping)

sns.distplot(leavers['EnvironmentSatisfaction'], kde=False, color="b", ax=axes[0,0])
sns.distplot(leavers['EmployeeNumber'], bins=3, kde=False, color="b", ax=axes[0, 1])
sns.distplot(leavers['JobSatisfaction'], kde=False, color="b", ax=axes[0, 2])

sns.distplot(leavers['JobInvolvement'], kde=False, color="b", ax=axes[1,0])
sns.distplot(leavers['NumCompaniesWorked'], kde=False, color="b", ax=axes[1, 1])
sns.distplot(leavers['RelationshipSatisfaction'], kde=False, bins=5, color="b", ax=axes[1, 2])
sns.distplot(leavers['TotalWorkingYears'],bins=10, kde=False, color="b", ax=axes[2,0])
sns.distplot(leavers['YearsAtCompany'], bins=10,kde=False, color="b", ax=axes[2, 1])


plt.tight_layout()
```

```
import matplotlib.pyplot as plt
import plotly.graph_objects as go

def plot_pie(df, column):
    counts = df[column].value_counts()
    labels = counts.index
    values = counts.values

    fig = go.Figure(data=[go.Pie(labels=labels, values=values)])
    fig.update_layout(title=f'Distribution of  {column}')
    fig.show()

def barplot(df, column, horizontal=True):
    counts = df[column].value_counts()
    plt.figure(figsize=(8, 6))
    if horizontal:
        counts.plot(kind='barh')
        plt.xlabel('Count')
        plt.ylabel(column)
    else:
        counts.plot(kind='bar')
        plt.xlabel(column)
        plt.ylabel('Count')
    plt.title(f'Bar Plot of {column}')
    plt.show()

#print('\n\nNonAttrition')
plot_pie(grp_nonAttrition, 'Gender')

#print('\n BAsed on attrition')
plot_pie(grp_Attrition, 'Gender')

barplot(grp_nonAttrition, 'Gender', horizontal=False)
#barplot(grp_Attrition, 'Gender', horizontal=False)
```

```
sns.set(style="white", palette="muted", color_codes=True)

# Set up the matplotlib figure
f, axes = plt.subplots(3, 3, figsize=(8,8))
sns.despine(left=True)

#people that left
leavers = hr_Attrition.loc[hr_Attrition['Attrition'] == 1]

# Plot a simple histogram with binsize determined automatically
#department_mapping = {'Sales': 1, 'Human Resources': 2, 'Research & Development': 3}

#hr_Attrition['Department'] = hr_Attrition['Department'].map(department_mapping)

sns.distplot(leavers['EnvironmentSatisfaction'], kde=False, color="b", ax=axes[0,0])
sns.distplot(leavers['EmployeeNumber'], bins=3, kde=False, color="b", ax=axes[0, 1])
sns.distplot(leavers['JobSatisfaction'], kde=False, color="b", ax=axes[0, 2])

sns.distplot(leavers['JobInvolvement'], kde=False, color="b", ax=axes[1,0])
sns.distplot(leavers['NumCompaniesWorked'], kde=False, color="b", ax=axes[1, 1])
sns.distplot(leavers['RelationshipSatisfaction'], kde=False, bins=5, color="b", ax=axes[1, 2])
sns.distplot(leavers['TotalWorkingYears'],bins=10, kde=False, color="b", ax=axes[2,0])
sns.distplot(leavers['YearsAtCompany'], bins=10,kde=False, color="b", ax=axes[2, 1])


plt.tight_layout()
```

PCA

```
from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import StandardScaler
pca = PCA(random_state=0)
principalComponents = pca.fit_transform(StandardScaler().fit_transform(X_train))
print(pca.explained_variance_ratio_)
print('Two PCs explain', sum(pca.explained_variance_ratio_)*100, '% of variance cumulatively')
# scree plot
PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.show()
```

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

# Assuming 'hr_Attrition' is your DataFrame
# Replace 'hr_Attrition' with the actual name of your DataFrame
sns.factorplot(x='JobSatisfaction', y='Gender', data=hr_Attrition, kind='violin', size=4)
plt.title('Box Plot of Job Satisfaction by Gender')
plt.xlabel('Gender')
plt.ylabel('Job Satisfaction')
plt.show()
```

RandomOver Sampling

```
from imblearn.over_sampling import RandomOverSampler
Y=df['Attrition']
X=df.drop(['EmployeeCount','Attrition','EmployeeNumber','Over18','StandardHours'],axis


ros = RandomOverSampler(random_state=42)
X_, Y = ros.fit_resample(X,Y)
X = pd.DataFrame(X_,columns=X.columns)
```

Train_Test_Data

```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=4)
```

Decision Tree

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(1972, 30) (1972,)
(494, 30) (494,)
```

```
from sklearn.tree import DecisionTreeClassifier

hr_Attrition= DecisionTreeClassifier()
hr_Attrition.fit(X_train, y_train)
hr_Attrition.score(X_test, y_test)
hr_Attrition_predictions = hr_Attrition.predict(X_test)
```

```
from sklearn.tree import DecisionTreeClassifier

# Create a DecisionTreeClassifier
hr_Attrition = DecisionTreeClassifier()

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

# Calculate and print the accuracy on the test data
accuracy = hr_Attrition.score(X_test, y_test)
print("Accuracy on the test data:", accuracy)

# Make predictions on the test data
df_predictions = df.predict(X_test)
```

LogisticRegression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Create a Logistic Regression classifier
logistic_regression = LogisticRegression()

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

# Make predictions on the test data
logistic_regression_predictions = logistic_regression.predict(X_test)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, logistic_regression_predictions)

# Display the confusion matrix as a heatmap
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(4, 2))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlGnBu', cbar=False,
            xticklabels=logistic_regression.classes_, yticklabels=logistic_regression.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

RandomForest CLassifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Create a Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)

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

# Make predictions on the test data
random_forest_predictions = random_forest.predict(X_test)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, random_forest_predictions)

# Display the confusion matrix as a heatmap
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(4, 2))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlGnBu', cbar=False,
            xticklabels=random_forest.classes_, yticklabels=random_forest.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

SVM

```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=4

# Create an SVM classifier (you can choose the kernel and other hyperparameters)
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)

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

# Make predictions on the test data
svm_predictions = svm_classifier.predict(X_test)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, svm_predictions)
```

```
# Display the confusion matrix as a heatmap
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(4, 2))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlGnBu', cbar=False,
            xticklabels=svm_classifier.classes_, yticklabels=svm_classifier.classes_)
plt.xlabel('Predicted')
```