Project Report

AIM-511: Machine Learning

IMT2022534, IMT2022098, IMT2022030

International Institute of Information Technology, Bangalore

TU Delft

Contents

1

Introduction

1.1. Project Objective

This project is a part of the course project assigned to us for the course AIM-511 Machine learning. This is the part 1 of the project.

In this project, we try to predict the exit status of different employees given their personal and professional factors. These features are available in a CSV file so we can make our model accordingly.

This dataset has a rich amounts of features and demands various preprocessing techniques to make it model friendly i.e. compatible with the models we are going to apply on it.

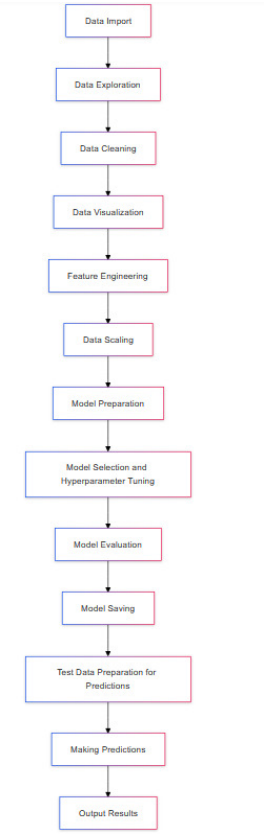

Figure 1.1: Flow chart of the project implementation

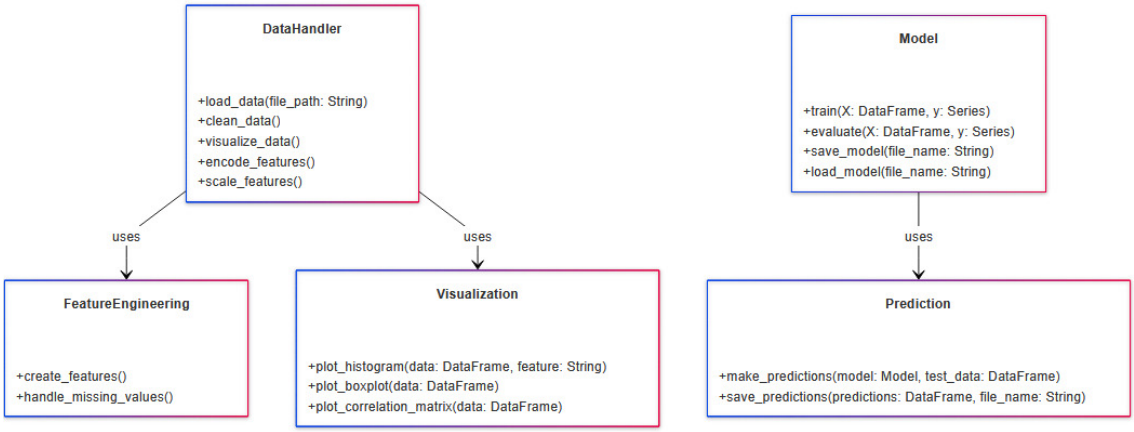

Figure 1.2: basic architecture of project code

figure 1.1 shows an elaborate flow chart as to how our code is structured and in what are we doing things.

2

Data Exploration and Analysis

2.1. Data Overview

We have two types of data files given to us, training and testing data. The initial shape of the training data contains 59611 rows and 24 columns. There are two types of features in this dataset, namely numerical and categorical features. Numerical features are data points represented by numbers that can be measured on a scale, like age, height, or income, while Categorical Features are data points that belong to distinct categories or groups, like gender, hair color, or shirt size.

2.2. Initial Observations

Immediately analysing the CSV of the data, we got to know that a good amount of values are missing and for some entries, multiple features are not available. The features whose values were missing the most was work_life_balance with around 10000 values missing. We will see in the Data Preprocessing section as to how we deal with missing values based upon the feature type they are of.

Using the code block

```
test_description = test.describe()
train_description = train.describe()
print(test_description)
# Get basic statistics
print(train_description)
```

We get basic statistical analysis of different features in the dataset

	response_id	age	years_at_company	monthly_income	\
count	59611.000000	59611.000000	59611.000000	57811.000000	
mean	37224.692171	38.562497	15.752630	7311.070350	
std	21519.598309	12.082500	11.245724	2197.444033	
min	1.000000	6.000000	1.000000	1316.000000	
25%	18576.000000	28.000000	7.000000	5661.000000	
50%	37207.000000	39.000000	13.000000	7358.000000	
75%	55874.500000	49.000000	23.000000	8882.000000	
max	74498.000000	59.000000	51.000000	56050.000000	

	promotions_count	distance_from_home	dependents_count	company_tenure
count	59611.000000	59611.000000	54831.000000	55427.000000
mean	0.832514	50.024912	1.648465	55.772909
std	0.994987	28.519542	1.555767	25.395430
min	0.000000	1.000000	0.000000	2.000000
25%	0.000000	25.000000	0.000000	36.000000
50%	1.000000	50.000000	1.000000	56.000000
75%	2.000000	75.000000	3.000000	76.000000
max	4.000000	297.000000	6.000000	128.000000

Figure 2.1: Statistical data of training set

We can conclude from the output that features have very different statistical parameters with some features having measures of central tendency in the magnitude of 10 to the power 4 while some being as low as 0.8, this can lead to bad model training. We will see in Data Preprocessing how we avoid this issue by changing the statistical measures of features.

2.3. Data Visualization

We used seaborn and matplotlib to plot graphs. These help us visualise distributions, correlation and outliers.

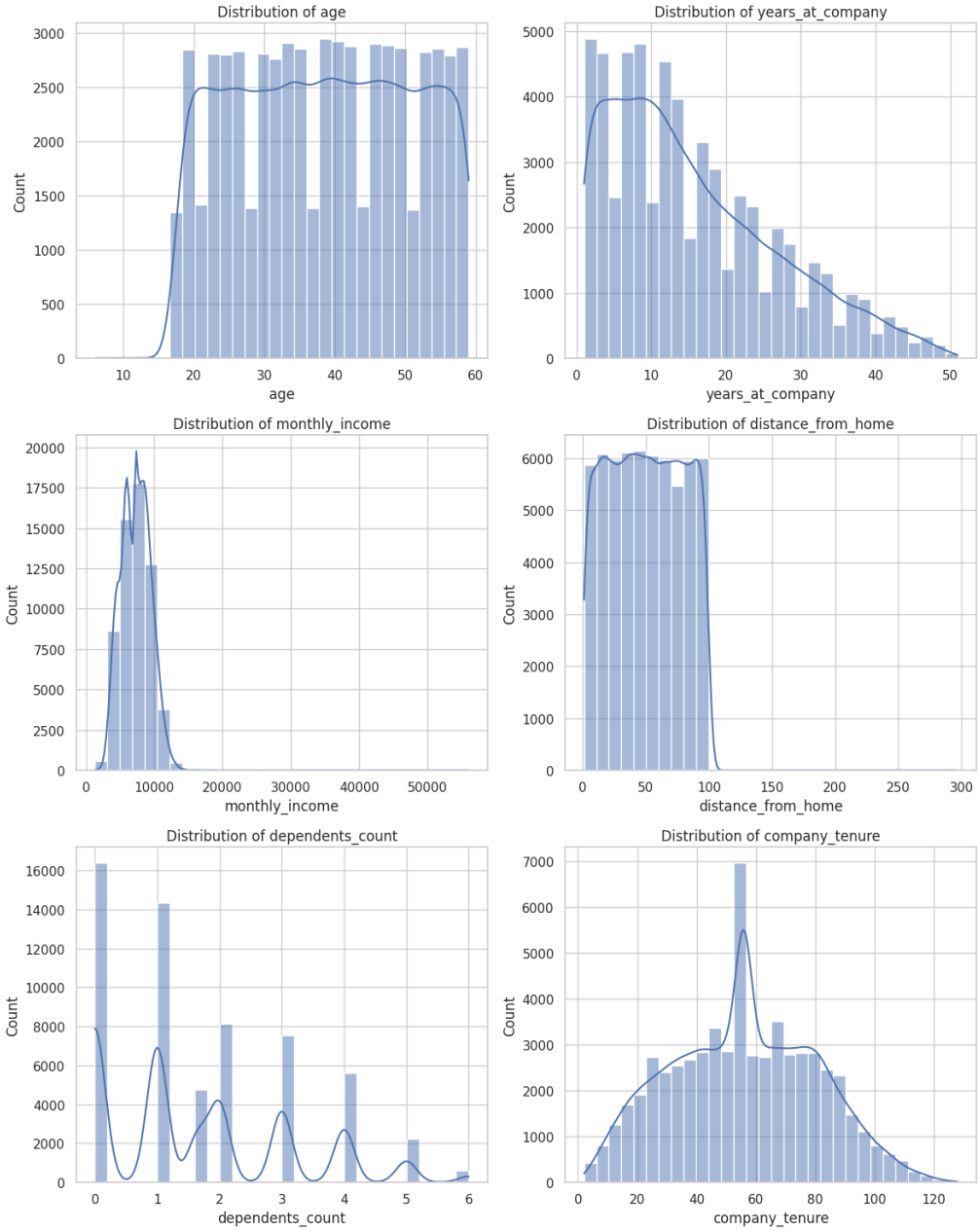

Figure 2.2: Distributions of various features

The figure above shows the distributions of various features and approximate them with a plot line. The age distribution appears relatively uniform between ages 20 and 60, with a noticeable drop in counts at the extremes. There is a right-skewed distribution, with most employees having worked less than 10 years at the company. Only a few employees have stayed for over 20 years, indicating lower long-term retention.

Monthly income shows a left-skewed distribution, where the majority of employees earn between 5,000 and 15,000. There are fewer employees in higher income brackets, and income distribution is relatively tight around a lower range.

The distribution of the distance from home shows a high concentration around shorter distances, indicating that most employees live relatively close to the workplace. The count of dependents is highly varied, with peaks at 0, 1, and 2 dependents.

Company tenure has a bell-shaped distribution, with a peak around 40 to 60 months, suggesting most employees stay with the company for about 3 to 5 years.

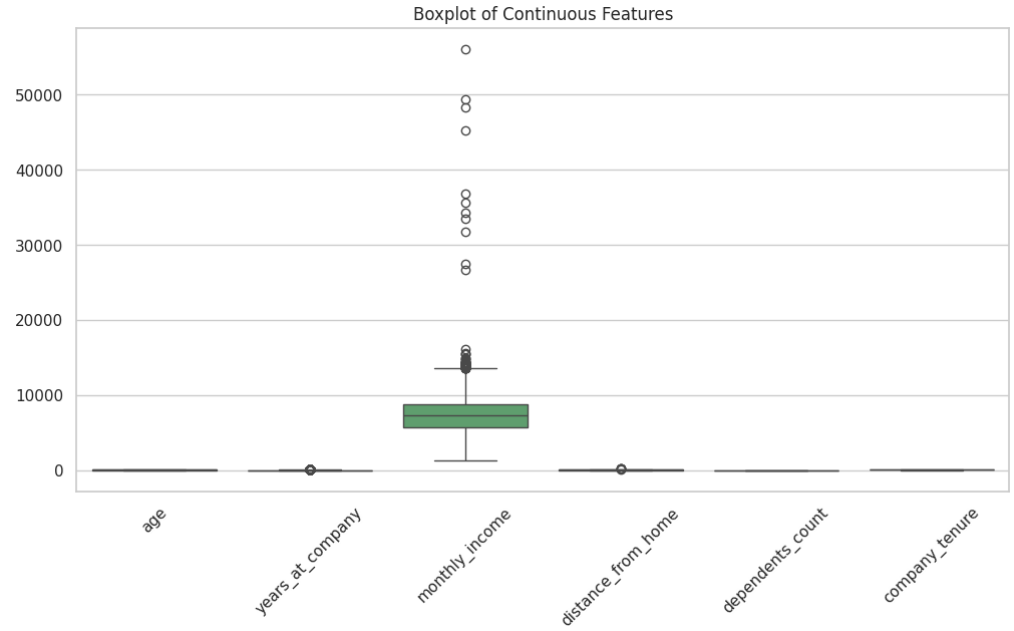

Figure 2.3: Box plots of continuous features

This is the box plot of the continuous features in the data. No other features except monthly income shows some interest. Monthly income has a wider interquartile range (IQR), and there are many high-income outliers above the upper whisker. The large number of outliers indicates substantial income disparity within the company.

Figure 2.4 shows count plots for categorical features, most of these features have only 3-4 categories. Some insights that we can draw from these are:-

1. Gender: There is a slightly higher number of male employees than female employees.
2. The most common job roles are in Technology and Healthcare, followed by Education. Finance and Media have fewer employees.
3. Work-Life Balance: Most employees report a ”Good” work-life balance, with fewer rating it as ”Excellent” or ”Poor.”
4. Job Satisfaction: A large proportion of employees rate their job satisfaction as ”High,” followed by ”Medium.” Very few rate it as ”Low.”
5. Performance Rating: Most employees have an ”Average” performance rating, while fewer fall into ”High,” ”Low,” or ”Below Average” categories.
6. Overtime: The majority of employees do not work overtime, with a smaller proportion working overtime.
7. Employee Recognition: ”Medium” and ”Low” levels of employee recognition are the most common, with fewer employees reporting ”Very High” or ”High” recognition.

Many such insights can be drawn from these count plots, not to mention all of them.

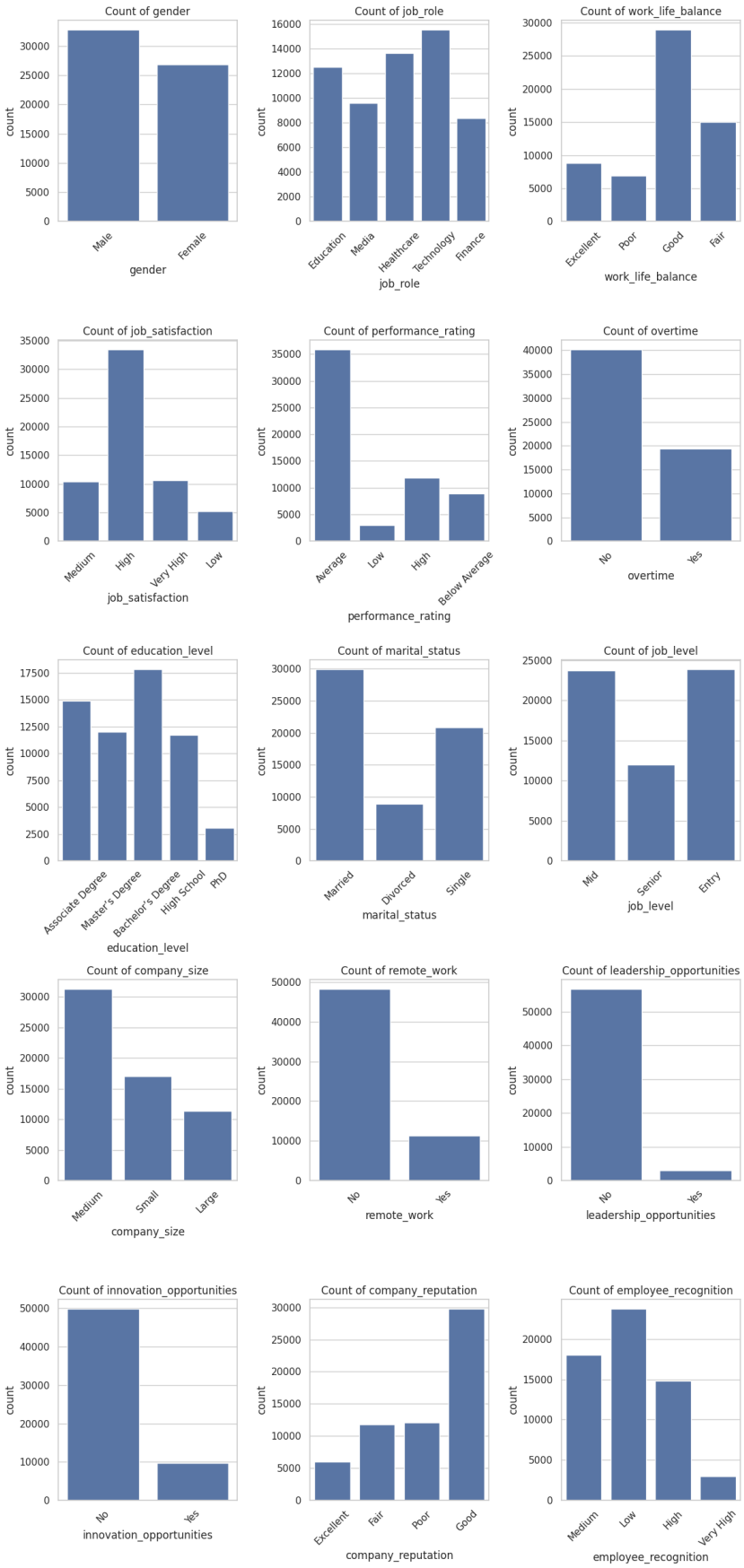

Figure 2.4: Count Plots for Categorical Features

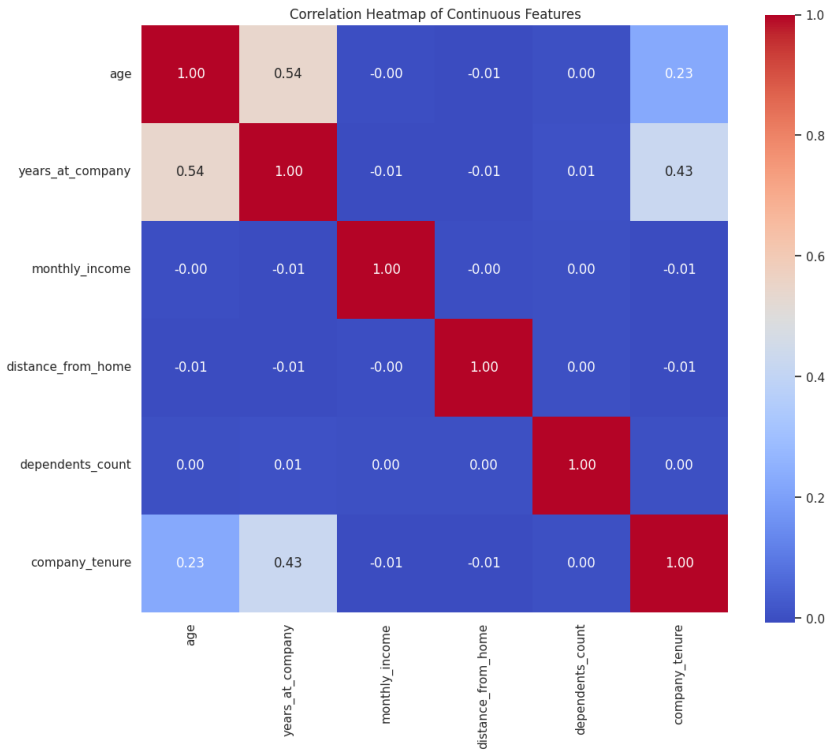

Figure 2.5: correlation matrix

The correlation matrix in figure 2.5 shows some interesting and logical trends in the dataset.

1. Age and years at company show a high positive correlation(0.54) suggesting that older employees tend to have been at the company longer, which is logically expected. Similar is the case with company tenure and years at company.
2. Monthly income shows little to no correlation with any other continuous features. Salary isn't significantly tied to age or experience at the company, which is counter intuitive. Income might be determined by other factors like categorical variables or features not shown in the dataset.

Most correlations in the dataset are quite weak (close to 0), suggesting these features are largely independent of each other, this is good for training a model as features don't appear to be redundant without providing any new information.

Figure 2.6 provides pair plots of continuous features, colored by the target variable, we can observe some density trends in the data. This can be useful for determining if we should apply certain models to the dataset. A non exhaustive list of observations can be:

1. There appears to be a slight pattern where employees who left (blue) are more concentrated in certain age ranges.
2. Years at Company: strong right-skewed distribution (most employees have fewer years, with a long tail). Shows moderate correlation with exit status - longer-tenured employees appear more likely to stay, which is logical.
3. Monthly income: there seems to be a slight pattern where higher-paid employees are less likely to leave

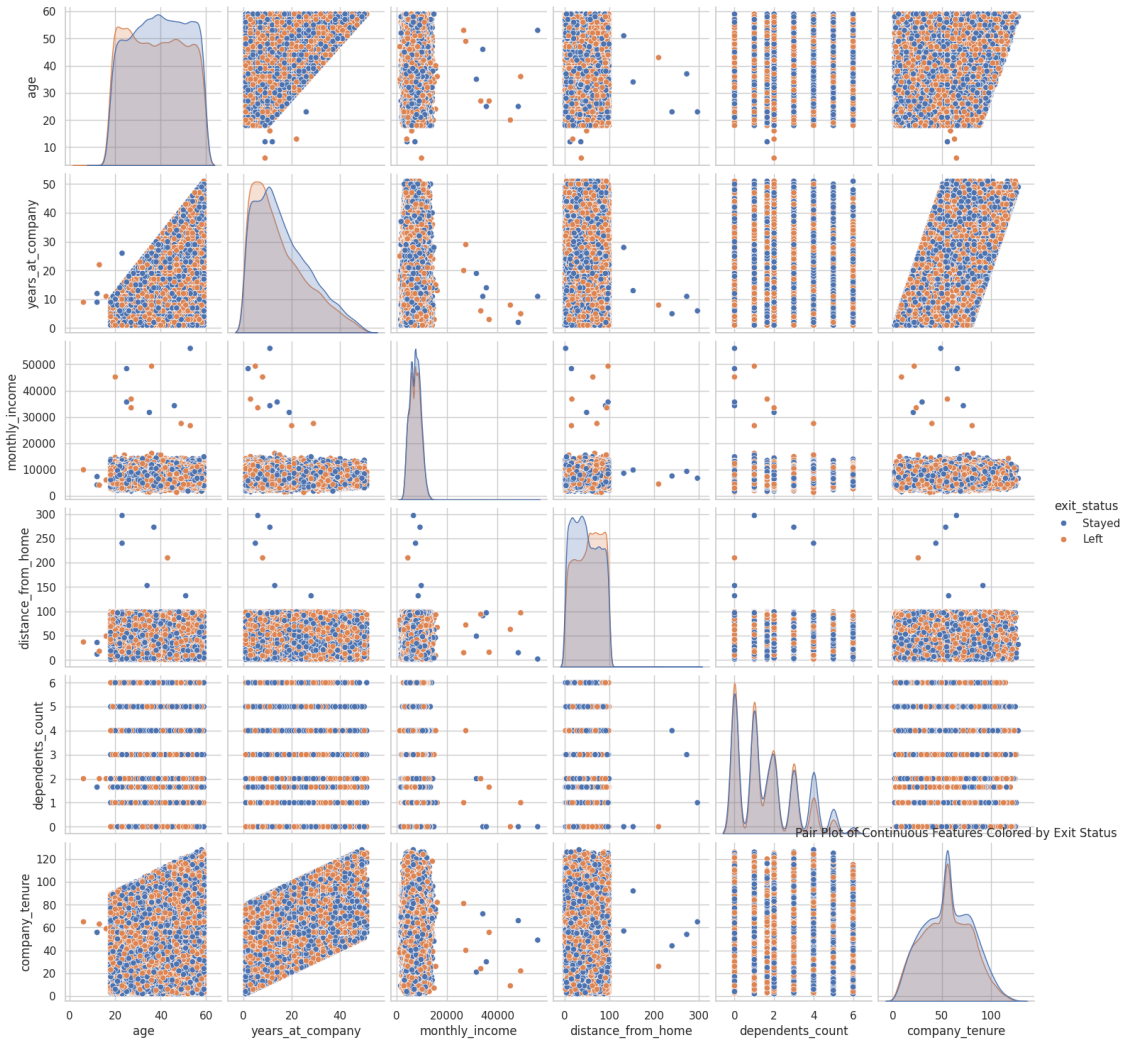

Figure 2.6: Pair Plot of Continuous Features Colored by Exit Status

As observed before, we couldn't find any correlation between continuous variables and monthly income, leading us to the conclusion the factors should be either one of the categorical ones or unknown.

To explore one possibility, we make bar plots correlating different categories in categorical features with monthly income to find any trends.

Figure 2.7 on the next page shows that our arguments are unfounded and average monthly income shows no disparity among the different categories in categorical variables. Average monthly income staying constant with increasing education level is counter-intuitive, but in some cases credentials don't matter that much so should be alright.

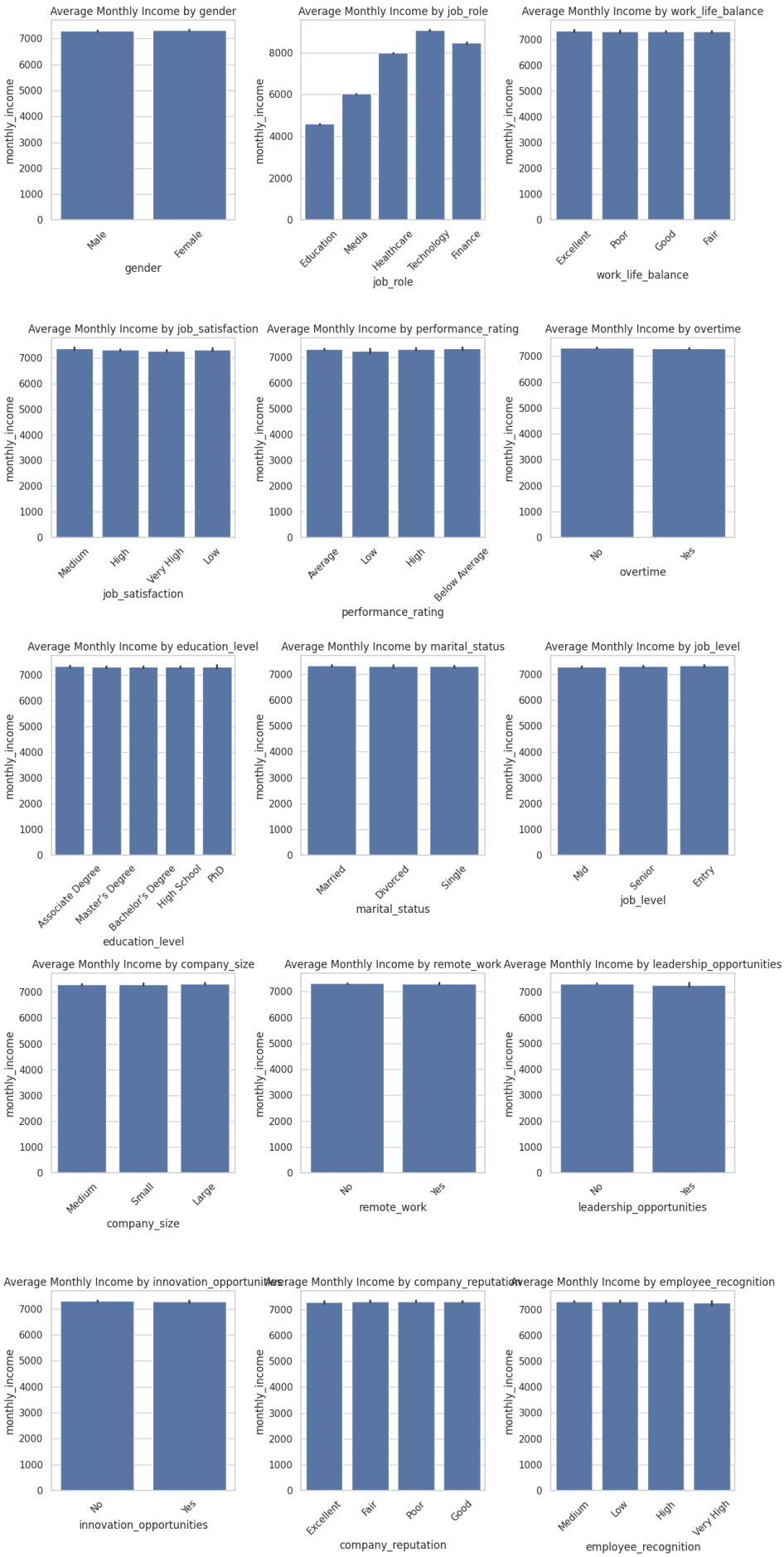

Figure 2.7: Bar Plots for Categorical Features with Average Monthly Income

3

Data Preprocessing

3.1. Duplicates and Missing values

for initial preprocessing, we decided to deal with duplicate values by dropping them, this step has been done because

1. Reduce Redundancy: Duplicate rows do not add any new information to the model. By removing them, we avoid unnecessary redundancy in the data, making the dataset cleaner and easier to analyze.
2. Prevent Bias: Duplicate entries can bias the model if certain data points are repeated, causing the model to ”learn” from some samples more than others.

For missing values, we have classified the missing values into two types, numerical and categorical. Missing values are often replaced with statistical measures according the type and nature of data. Numerical values are replaced with median. Mean is not used for numerical values because:-

1. Robustness: The median is less sensitive to outliers compared to the mean. Using the median as a filler for missing values reduces the risk of skewing the data if there are extreme values in the dataset.
2. Better Representation: In datasets with skewed distributions, the median provides a more accurate central value for missing data, preserving the distribution’s shape.

Categorical values are replaced with mode, possible reasons include likelihood based data labeling and minimizing distortion.

3.2. Encoding and Standardization

This code block is applying label encoding to binary features.

```
# Apply Label Encoding for binary features
for feature in binary_features:
    le = LabelEncoder()
    df[feature] = le.fit_transform(df[feature])
    label_encoders[feature] = le  # Store the encoder
    df[feature] = df[feature].astype(bool)
```

for multi-category features, we are using one hot encoding.

```
df = pd.get_dummies(df, columns=multi_category_features, drop_first=True)
```

Most machine learning models require numerical input, so categorical values (like strings) must be converted into numbers. Label encoding does this by assigning unique numerical values to each category, making the data easier for algorithms to process and interpret.

Additionally, ensuring consistent encoding across training and test data helps maintain model accuracy, as models trained with a specific encoding cannot interpret different or misaligned encodings. transform and fit_transform in sklearn have been used to maintain consistency in the encoding between the training and testing data.

After that, we performed a test-train split with coefficient as 0.2, this helps in the validation scheme of the model.

This code block standardizes the numerical features in the dataset, a preprocessing step that is crucial for many machine learning models.

```
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

We are using the StandardScaler library in SKLearn for standardization, it is beneficial for many reasons:

1. Model Performance: Many machine learning algorithms perform better when features have similar scales, as they are sensitive to the magnitude of the data.
2. Improves Convergence: Standardized data often speeds up the training process, especially for models that use gradient-based optimization

Standardization generally improves model stability and accuracy by ensuring consistent and comparable feature scales across the dataset.
We further tested for highly skewed features, so we have to deal with them

```
# Calculate skewness for numerical features
skewness = df1[numerical_features].skew().sort_values(ascending=False)
print("Skewness of numerical features:\n", skewness)

# Identify features with absolute skewness greater than 1
high_skew = skewness[abs(skewness) > 1].index.tolist()
print("Highly skewed features:", high_skew)
```

The output showed no features are highly skewed with absolute skewness greater than one.

3.3. Feature engineering

We created some extra features depending upon trends and observations found during data analysis and visualization.

```
df1['remote_work_distance'] = np.where(df1['remote_work'] == 'Yes', df1['distance_from_home'
    ], 0)

# 2. Aggregation Features
# Promotion Rate
df1['promotion_rate'] = np.where(df1['years_at_company'] != 0, df1['promotions_count'] / df1[
    'years_at_company'], 0)

# Income per Dependent
df1['income_per_dependent'] = df1['monthly_income'] / (df1['dependents_count'] + 1)
```

Engineered features can offer many advantages like:

1. They help in clustering or segmentation, as employees with similar promotion rate or income per dependent may share certain needs or behaviors.
2. They improve the model's predictive power by incorporating meaningful, derived information from the original features.

Then we dropped the column of response id as it is an identifier and provides no valuable information or trend towards the target variable.

4

Model Selection

We selected 6 models for training and comparing their accuracies on the training data.

```
models = {
    'Decision Tree': (DecisionTreeClassifier(), {
        'max_depth': [3, 5, 10, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }),
    'Random Forest': (RandomForestClassifier(random_state=42), {
        'n_estimators': [50, 100, 200],
        'max_depth': [3, 5, 10, None],
        'min_samples_split': [2, 5, 10]
    }),
    'XGBoost': (XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state
        =6969), {
        'n_estimators': [1800],
        'max_depth': [1],
        'learning_rate': [0.4],
        'subsample': [0.60]
    }),
    'Naive Bayes': (GaussianNB(), {
        'var_smoothing': [1e-9, 1e-8, 1e-7]
    }),
    'KNN': (KNeighborsClassifier(), {
        'n_neighbors': [3, 5, 7, 10],
        'weights': ['uniform', 'distance']
    }),
    'AdaBoost': (AdaBoostClassifier(random_state=42), {
        'n_estimators': [50, 100],
        'learning_rate': [0.2, 0.3]
    })

}
```

The models we are using are:

- Decision Tree
- Random Forest
- XGBoost
- Naive Bayes
- K Nearest Neighbours
- Adaboost

It is important to note that Logistic Regression was not utilized in this phase of the project. It is planned for inclusion in Part 2 of the project.

We have made our model selection based upon some parameters.

- Decision Tree: Trees capture non-linear relationships, which is beneficial if there are complex interactions among features.
- Random Forest: Random Forests are an ensemble of multiple decision trees trained on different subsets of data, reducing overfitting and increasing the model's robustness compared to a single decision tree.
- XGBoost: XGBoost is a powerful gradient-boosting algorithm known for its accuracy and speed, often performing better than many other models, especially on structured/tabular data.XGBoost includes regularization terms (L1 and L2), which help prevent overfitting. It also has a range of hyperparameters that allow fine-tuning for better performance.
- Naive Bayes: Naive Bayes is a simple yet efficient probabilistic model, making it fast and suitable for high-dimensional datasets. It's based on Bayes' theorem and assumes independence between features. Naive Bayes can perform well on small datasets or when the features are independent, which might be useful if there is limited data.
- K-Nearest Neighbours: KNN is a non-parametric, instance-based learning algorithm, meaning it doesn't assume a particular form for the underlying data distribution. Since KNN makes predictions based on the majority class among nearest neighbors, it can naturally capture non-linear decision boundaries.
- AdaBoost: AdaBoost is an ensemble method that combines multiple weak classifiers, typically decision trees, to create a strong classifier. It focuses on improving predictions by re-weighting misclassified instances, which helps in boosting accuracy. AdaBoost is useful when the base learners (e.g., decision trees) underfit, as it sequentially reduces bias by focusing on errors.

5

Model Evaluation

in the code snippet that was attached in Model Selection section, we defined a list of parameters for all the models we are using, now we use something called as Grid Search for determining the best combination of parameters for each model out of the ones given in the list.

```
for model_name, (model, params) in models.items():
    print(f"Running␣GridSearchCV␣for␣{model_name}...")
    grid_search = GridSearchCV(model, params, cv=5, scoring='f1_weighted', n_jobs=-1)  # Use
        weighted F1 score for scoring
    grid_search.fit(X_train, y_train)

    # Get the best model and its parameters
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
    best_score = grid_search.best_score_  # F1 score from cross-validation
```

Comparing the best parameters of all the models and their scores on the training data, we select one best model to make the predictions on testing data. In our case, that is the XGBoost model.

```
# Store the results
    best_results[model_name] = {
        'Best␣Parameters': best_params,
        'CV␣F1␣Score': best_score,
        'Test␣Accuracy': test_accuracy,
        'Test␣F1␣Score': test_f1_score
    }

    # Track the model with the highest F1 score for saving
    if test_f1_score > highest_f1_score:
        highest_f1_score = test_f1_score
        best_overall_model = best_model
```

Figure 5.1 shows a line plot comparing Cross Validation scores of different models. This shows that XGBoost and AdaBoost models are giving very similar accuracies with XGBoost taking the lead by a small margin. K-Nearest Neighbours performing the worst out of all of them.

XGBoost performed the best, and some models performed very poorly, based upon literature and reasoning, here are some possible reasons as to why the other models could hav failed and not achieved the same performance as XGBoost:-

- A single decision tree does not benefit from the ensemble approach, which combines multiple models to improve performance.
- While Random Forest is an ensemble method that reduces overfitting compared to a single decision tree, it may still struggle with datasets that have a lot of noise or irrelevant features.
- Naive Bayes assumes that features are independent given the class label. This is not the case for some of our features as per the correlation matrix.

- As the number of features increases, the distance between points becomes less meaningful, which can degrade the performance of KNN.
- AdaBoost can be sensitive to noisy data and outliers, which can negatively impact its performance.

Images containing best parameters and individual scores of all the models have been attached from Figure 5.2-5.7. Now we will proceed to the conclusion section where we will summarise the content of our project.

Figure 5.1: line plot comparing scores of models

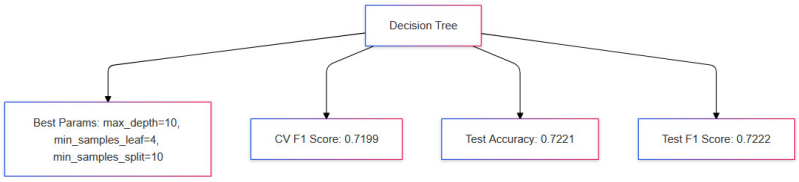

Figure 5.2: Decision Tree

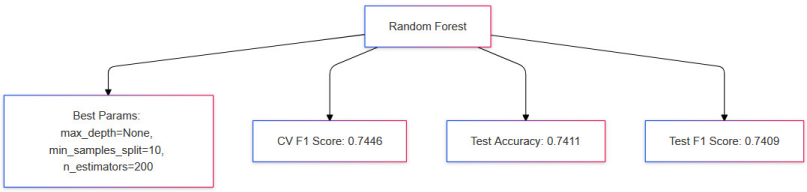

Figure 5.3: Random Forest

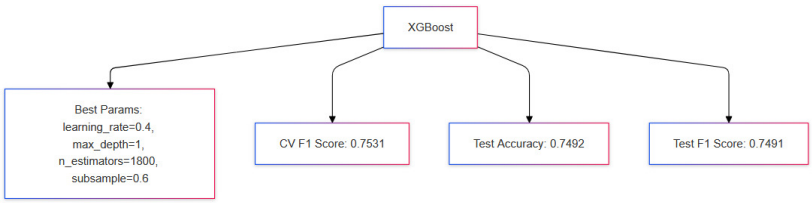

Figure 5.4: XGBoost

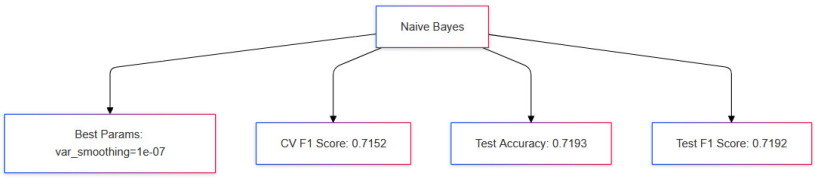

Figure 5.5: Naive Bayes

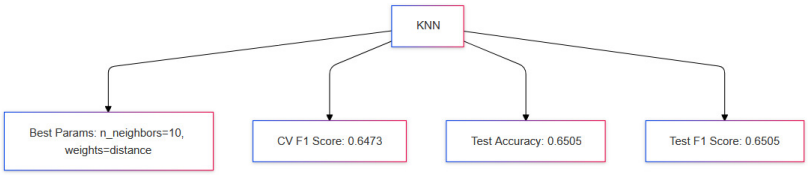

Figure 5.6: K-Nearest Neighbours

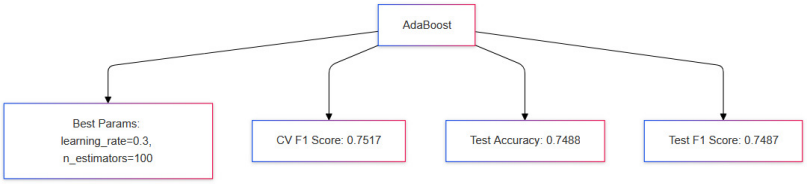

Figure 5.7: AdaBoost

6

Conclusion

We take the best model, perform predictions on the testing data. We also have to perform all the preprocessing steps on the testing data that we did on the training data(code snippet shown below) also note that we have used transform here and not fit transform, this maintains common encoding between training and testing data. We submitted our predictions on Kaggle and ended up with an accuracy score of 0.75757

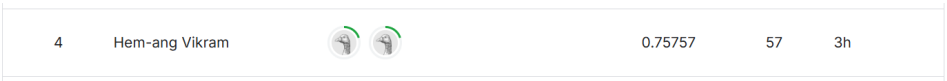

Figure 6.1: Kaggle accuracy

Out of all the models we tested, we found out XGBoost model performed the best, XGBoost models are performing amazing on various datasets on Kaggle due to a lot of factors like Gradient Boosting Framework, Regularization and robustness to overfitting.

```
test=test.drop(columns=['response_id'])

for col in test.select_dtypes(include=['float', 'int']).columns:
    test[col].fillna(test[col].mean(), inplace=True)
for col in test.select_dtypes(include=['object']).columns:
    test[col].fillna(test[col].mode()[0], inplace=True)

test = pd.get_dummies(test, columns=multi_category_features, drop_first=True)

test['remote_work_distance'] = np.where(test['remote_work'] == 'Yes', test['
    distance_from_home'], 0)

# 2. Aggregation Features
# Promotion Rate
test['promotion_rate'] = np.where(test['years_at_company'] != 0, test['promotions_count'] /
    test['years_at_company'], 0)

# Income per Dependent
test['income_per_dependent'] = test['monthly_income'] / (test['dependents_count'] + 1)

# Display the first few rows of the updated DataFrame to check results
# Check the resulting test dataframe

test[numerical_features] = scaler.transform(test[numerical_features])
```