DAE Project Report

An Analysis of Employee Retention Using Regression Techniques

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Kudzai Saurombe - 2503314
Tapiwa Chatikobo - 2442335
Riot Ndlovu - 2096330

Chapter 1

Data Exploration and Visualisation

1.1 Overview of the Data Set and Variables

This section provides a detailed report on our data exploration steps, leveraging the provided code and aligning with the structure and style observed in the example report. Our focus was on providing a clear and methodical analysis of the dataset, ensuring all steps were well-documented and supported by visualisations.

The dataset consists of data scientists from Big Company Inc who have filled out employee satisfaction surveys. The main objective is to analyse this data to predict whether an employee will stay with the company three months after the survey. The variables included in the dataset are:

- - **Stay**: Indicates whether the data scientist stayed with the company (1) or not (0).
- - **Pay**: Monthly salary in dollars.
- - **Estimated Happiness (EstHap)**: A score derived from a model based on reported happiness and comments.
- - **Performance (Perf)**: A score based on the manager's performance review.

1.2 Loading the Dataset

We extracted the dataset from a CSV file and performed initial inspection steps to get a sense of the data. The following commands were used to load and inspect the data:

```
df = pd.read_csv('Data_Scientists_data.txt', delimiter=',')
df.drop(columns=['ID'], inplace=True)
df.head()
```

Listing 1.1: Loading and Preprocessing Data

1.3 Data Types and Basic Information

To understand the structure of the data, we checked the data types of each column, along with basic information such as the number of non-null entries and memory usage. This helped us identify any potential issues with the data types and missing values.

```
print(df.dtypes)
print(df.info())
```

Listing 1.2: Checking Data Types and Information

Output:

```
Pay       int64
Perf      int64
EstHap    int64
Stay      int64
dtype: object
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Pay     500 non-null    int64
 1   Perf    500 non-null    int64
 2   EstHap  500 non-null    int64
 3   Stay    500 non-null    int64
dtypes: int64(4)
memory usage: 15.8 KB
```

Listing 1.3: Data Types and DataFrame Information

1.4 Descriptive Statistics

We computed descriptive statistics to provide a summary of the central tendency, dispersion, and shape of the dataset's distribution. The `describe()` method in pandas was used for this purpose:

```
print(df.describe())
```

Listing 1.4: Descriptive Statistics of the Data

Output:

```
                 Pay        Perf      EstHap        Stay
count     500.000000  500.000000  500.000000  500.000000
mean    23602.000000    6.052000    6.440000    0.918000
std     13519.806446    1.446887    0.984041    0.281842
min     10000.000000    2.000000    4.000000    0.000000
25%     19000.000000    5.000000    6.000000    1.000000
```

```
50%      22000.000000     6.000000     6.000000     1.000000
75%      27000.000000     7.000000     7.000000     1.000000
max     295000.000000    10.000000    11.000000     2.000000
```

Listing 1.5: Descriptive Statistics of the Data

This table provided valuable insights into the distribution of the data, such as mean values and ranges for each feature.

1.5 Missing Values

To ensure data integrity, we checked for the presence of missing values:

```
print(df.isnull().sum())
```

Listing 1.6: Checking for Missing Values

Output:

```
Pay        0
Perf       0
EstHap     0
Stay       0
dtype: int64
```

Listing 1.7: Missing Values in the Data

No missing values were found in the dataset, confirming its completeness.

1.6 Data Visualisation

1.6.1 Distribution of Stay

We visualised the distribution of the target variable `Stay` using a bar plot to understand the class imbalance:

```
sns.countplot(x='Stay', data=df)
plt.title('Distribution of Stay')
plt.show()
```

Listing 1.8: Distribution of Stay

From the plot, it is evident that the majority of employees stay with the company, which indicates a class imbalance.

1.6.2 Histograms for Numerical Features

We plotted histograms for the numerical features to understand their distributions:

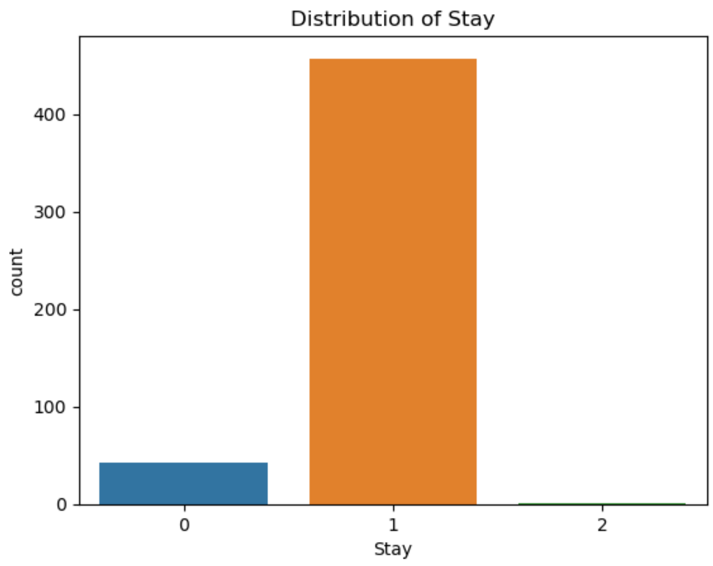

Figure 1.1: Distribution of Stay

```python
features = ['Pay', 'Perf', 'EstHap']
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[feature], bins=15, kde=False)
    plt.xlim(df[feature].min(), df[feature].max())
    plt.title(f'Histogram for {feature}')
    plt.show()
```

Listing 1.9: Histograms for Numerical Features

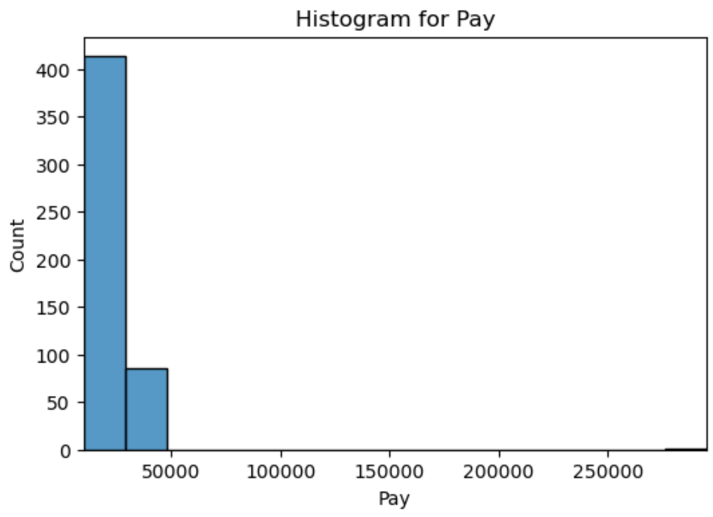

Figure 1.2: Histogram for Pay

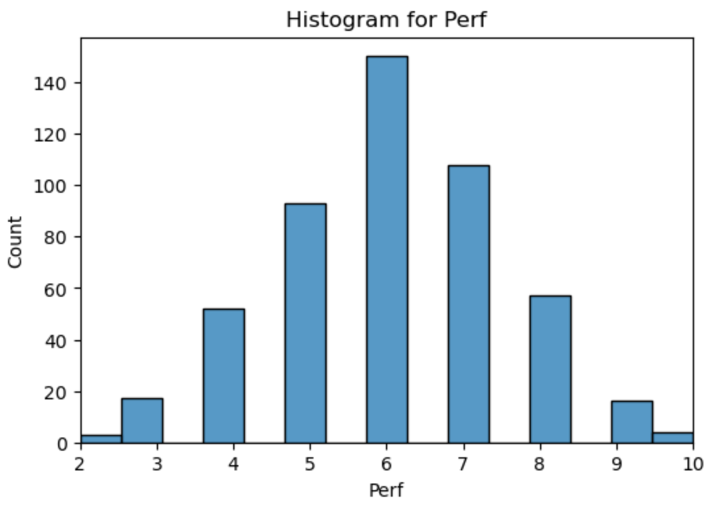

Figure 1.3: Histogram for Performance

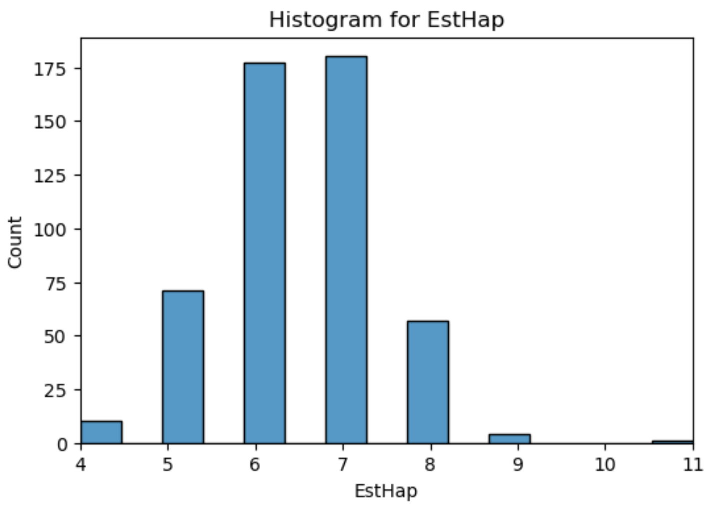

Figure 1.4: Histogram for Estimated Happiness

These plots helped us identify the distribution and helped us to view the outliers in the numerical features. From these plots we were able to see the preset outlier/error in the Stay variable as stay is supposed to only have values 0 and 1, the graph shows there is at least one value under 2. For the Pay variable, the graph shows a large outlier heavily distorting the data.

1.7 Outlier Detection

To better identify and visualise outliers, we used the IQR (Interquartile Range) method for each numerical feature:

```
def identify_outliers(df, features):
```

```python
    outliers = {}
    for feature in features:
        Q1 = np.percentile(df[feature], 25)
        Q3 = np.percentile(df[feature], 75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.3 * IQR
        upper_bound = Q3 + 1.3 * IQR
        outlier_indices = df[(df[feature] < lower_bound) | (df[feature] >
    upper_bound)].index
        outliers[feature] = outlier_indices
    return outliers

features_to_check = ['Pay', 'Perf', 'EstHap']
outliers_dict = identify_outliers(df, features_to_check)
for feature, outlier_indices in outliers_dict.items():
    print(f"Outliers in {feature}:")
    print(df.loc[outlier_indices, feature])
    print("---")
```

Listing 1.10: Identifying Outliers in Numerical Features

Output:

```
Outliers in Pay:
12       38000
29       44000
54       38000
200     295000
215      41000
224      39000
228      41000
314      38000
461      39000
472      41000
499      38000
Name: Pay, dtype: int64
---
Outliers in Perf:
10     10
166    10
178     2
199    10
243    10
397     2
445     2
Name: Perf, dtype: int64
---
Outliers in EstHap:
14      9
70      4
```

```
150     11
198      4
213      9
230      4
261      4
262      4
284      4
286      4
293      9
349      4
399      4
460      4
497      9
Name: EstHap, dtype: int64
---
```

Listing 1.11: Outliers in Numerical Features

1.8 Outlier Removal

We removed the identified outliers from the dataset to ensure a more robust analysis:

```
df = df[df['Pay'] != df['Pay'].max()]
df = df[df['Stay'] != df['Stay'].max()]
```

1.9 Updated Visualisations

After removing the outliers, we re-visualised the distributions:

```
sns.countplot(x='Stay', data=df)
plt.title('Distribution of Stay')
plt.show()

features = ['Pay', 'Perf', 'EstHap']
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[feature], bins=15, kde=False)
    plt.xlim(df[feature].min(), df[feature].max())
    plt.title(f'Histogram for {feature}')
    plt.show()
```

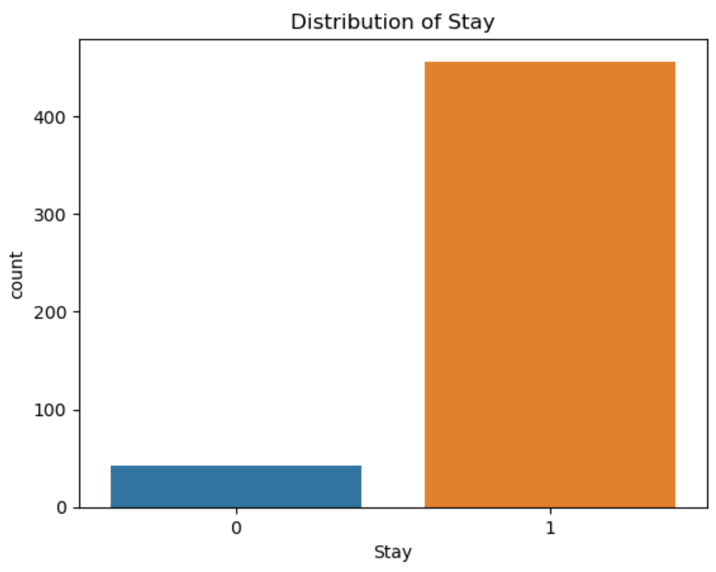

Figure 1.5: Updated Histogram for Pay

Histogram for Pay

The histogram for Pay now shows a more focused distribution, primarily between $10,000 and $40,000. The removal of extreme outliers has resulted in a more normally distributed shape, although it remains slightly right-skewed. This indicates that while most employees earn within this range, a few still earn significantly higher salaries.

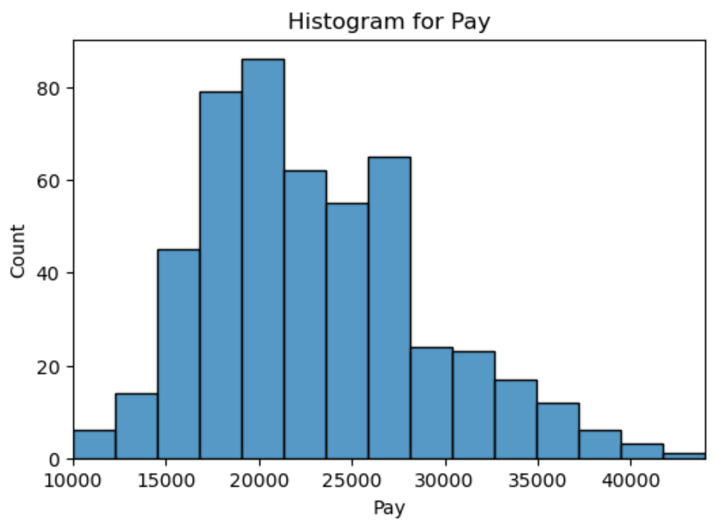

Figure 1.6: Updated Histogram for Performance

Histogram for Performance (Perf)

The performance scores exhibit a nearly symmetrical distribution centered around a score of 6. This distribution suggests that most employees have mid-range performance scores, with fewer employees scoring very low or very high.

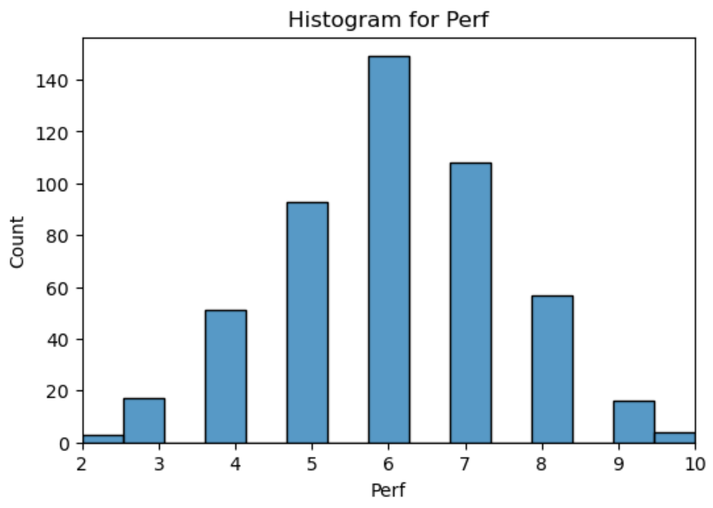

Figure 1.7: Updated Histogram for Performance

Histogram for Estimated Happiness (EstHap)

The histogram for EstHap shows a clearer bimodal distribution after outlier removal, with peaks around scores of 6 and 7. This suggests two distinct groups in terms of estimated happiness among the employees, likely indicating varying levels of satisfaction.

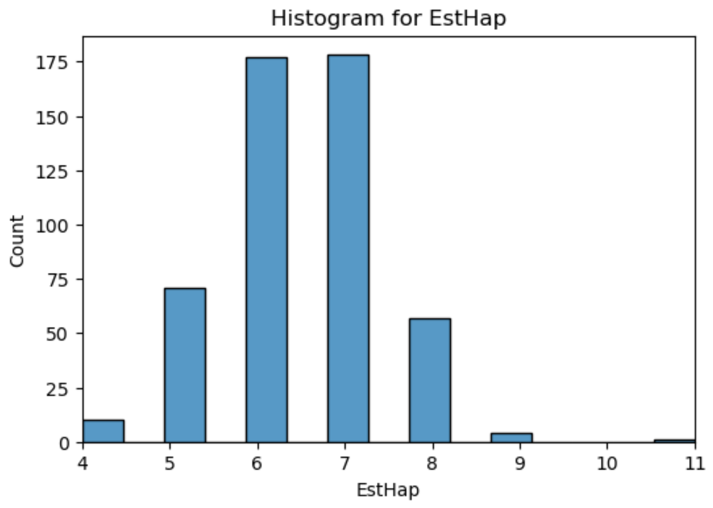

Figure 1.8: Updated Histogram for Estimated Happiness

Kernel Density Estimation (KDE) Plots

To get a smoother estimate of the data distribution, we generated KDE plots for each numerical feature:

```
features = ['Pay', 'EstHap', 'Perf']
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.kdeplot(df[feature], fill=True)
    plt.title(f'Kernel Density Estimation for {feature}')
    plt.show()
```

KDE for Pay

The KDE plot for Pay illustrates a smooth, unimodal distribution with a peak around \$20,000. The distribution tails off gradually towards higher pay, indicating fewer employees in the higher salary brackets. This visualization reaffirms that most employees earn between \$10,000 and \$30,000.

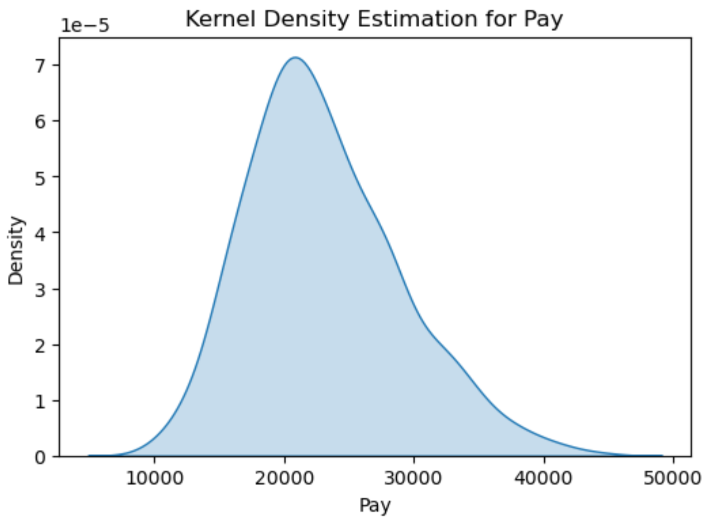

Figure 1.9: KDE for Pay

KDE for Estimated Happiness (EstHap)

The KDE plot for EstHap shows a distinct bimodal distribution, with prominent peaks around scores of 6 and 7. This suggests the presence of two main groups of employees in terms of their happiness levels. The smooth curves highlight the distribution's central tendencies and the spread of happiness scores within the company.

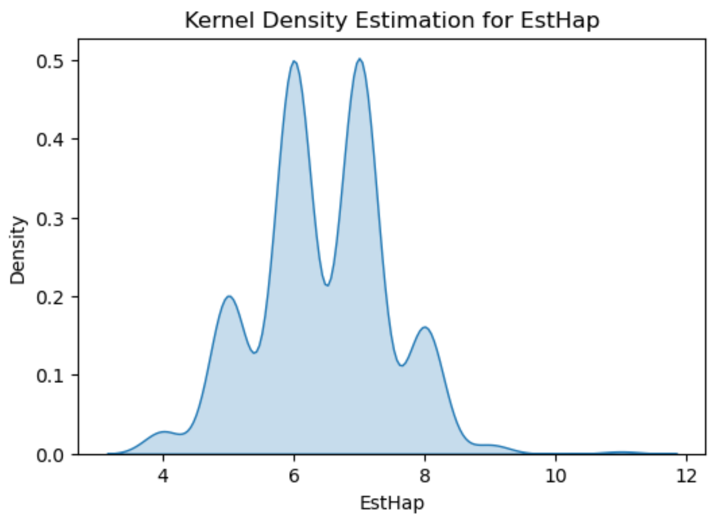

Figure 1.10: KDE for Estimated Happiness

KDE for Performance (Perf)

The KDE plot for Perf confirms a nearly normal distribution centered around a performance score of 6. This indicates that most employees perform at an average level, with fewer employees performing exceptionally well or poorly. The smooth curve helps to visualize the overall distribution and density of performance scores.

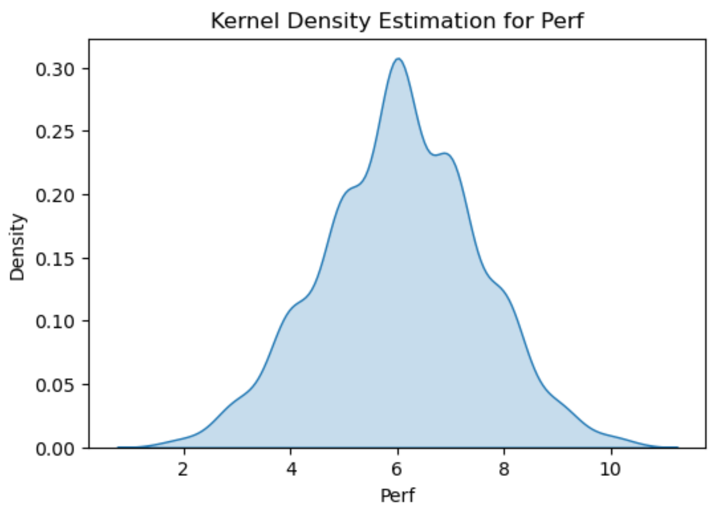

Figure 1.11: KDE for Performance

Interpretation

The updated visualizations, after removing outliers, provide a more accurate representation of the dataset. The histograms and KDE plots offer insights into the central tendencies and distributions of the numerical features.

- **Pay**: The pay distribution is slightly right-skewed, with most salaries ranging between $10,000 and $30,000. The removal of outliers has resulted in a more focused distribution.

- **Estimated Happiness (EstHap)**: The bimodal distribution of happiness scores suggests two distinct groups within the employees, indicating varying levels of satisfaction.

- **Performance (Perf)**: The nearly normal distribution centered around a score of 6 indicates that most employees perform at an average level, with fewer employees scoring extremely high or low.

1.10 Box Plots by Stay

We used box plots to compare the distributions of numerical features against the target variable `Stay`. We utilized box plots to compare the distributions of numerical features against the target variable Stay. This visualization method helps to reveal the central tendency, spread, and potential outliers for each feature with respect to the target variable. The box plots for Pay, Estimated Happiness (EstHap), and Performance (Perf) are provided below:

```python
features = ['Pay', 'Perf', 'EstHap']
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='Stay', y=feature, data=df, order=[0, 1])
    plt.title(f'Box plot of {feature} by Stay')
    plt.show()
```

Box Plot of Pay by Stay

The box plot for Pay shows that employees who left the company (Stay=0) had a slightly lower median pay compared to those who stayed (Stay=1). The interquartile range (IQR) for both groups is similar, indicating a comparable spread in pay. However, there are a few outliers in both groups, particularly in the higher pay range. This suggests that while median pay might not significantly differ, higher salaries might have an impact on retention.

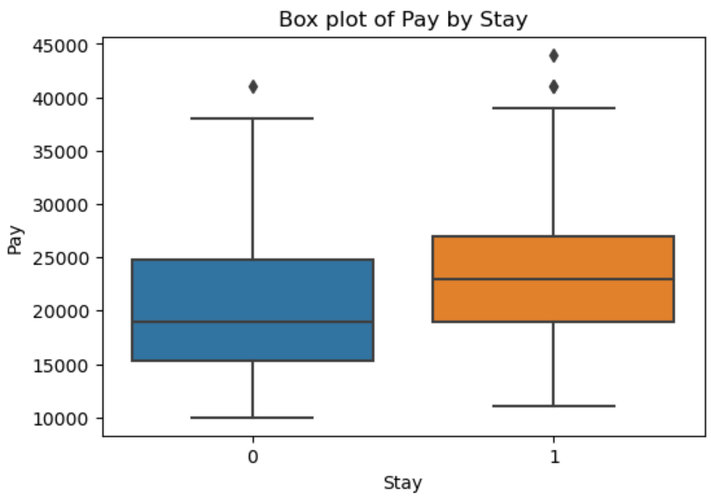

Figure 1.12: Box Plot of Pay by Stay

Box Plot of Estimated Happiness by Stay

For EstHap, the box plot reveals that employees who left the company had a slightly lower median estimated happiness score compared to those who stayed. The IQR for both groups is similar, but there are notable outliers in both groups. The presence of these outliers indicates that while the central tendency is similar, individual happiness scores vary widely and can influence the decision to stay or leave.

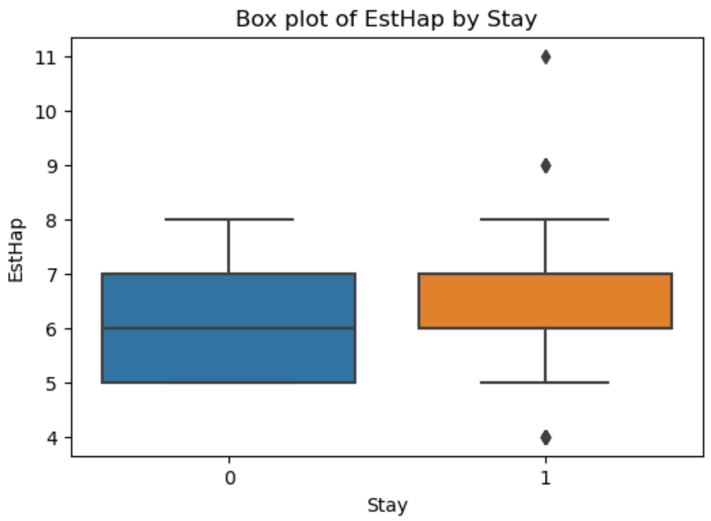

Figure 1.13: Box Plot of Estimated Happiness by Stay

Box Plot of Performance by Stay

The box plot for Perf indicates that the performance scores are fairly similar between the two groups. Both groups have a median performance score of around 6, with a similar spread indicated by the IQR. There are fewer outliers in this feature compared to Pay and EstHap, suggesting that performance scores are more consistent among employees regardless of their decision to stay or leave.

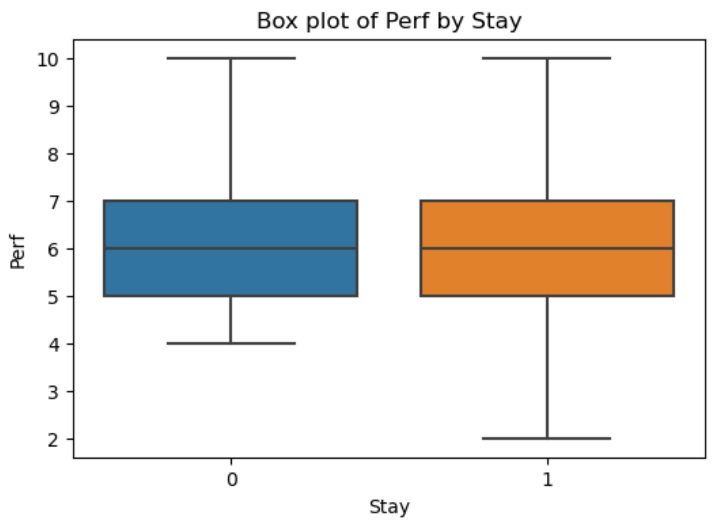

Figure 1.14: Box Plot of Performance by Stay

Interpretation

The box plots suggest that while there are slight differences in median values of Pay and EstHap between those who stayed and those who left, the overall distributions are quite similar. This implies that these features alone might not be strong predictors of an employee's decision to stay. The presence of outliers, especially in pay and happiness, indicates that individual variations can have a significant impact and should be considered in any predictive modelling. The consistency in performance scores suggests that other factors, potentially beyond the measured features, play a crucial role in employee retention.

1.11 Pair Plot

We created a pair plot to visualise the relationships between all features and the target variable:

```
sns.pairplot(df, hue='Stay')
plt.show()
```

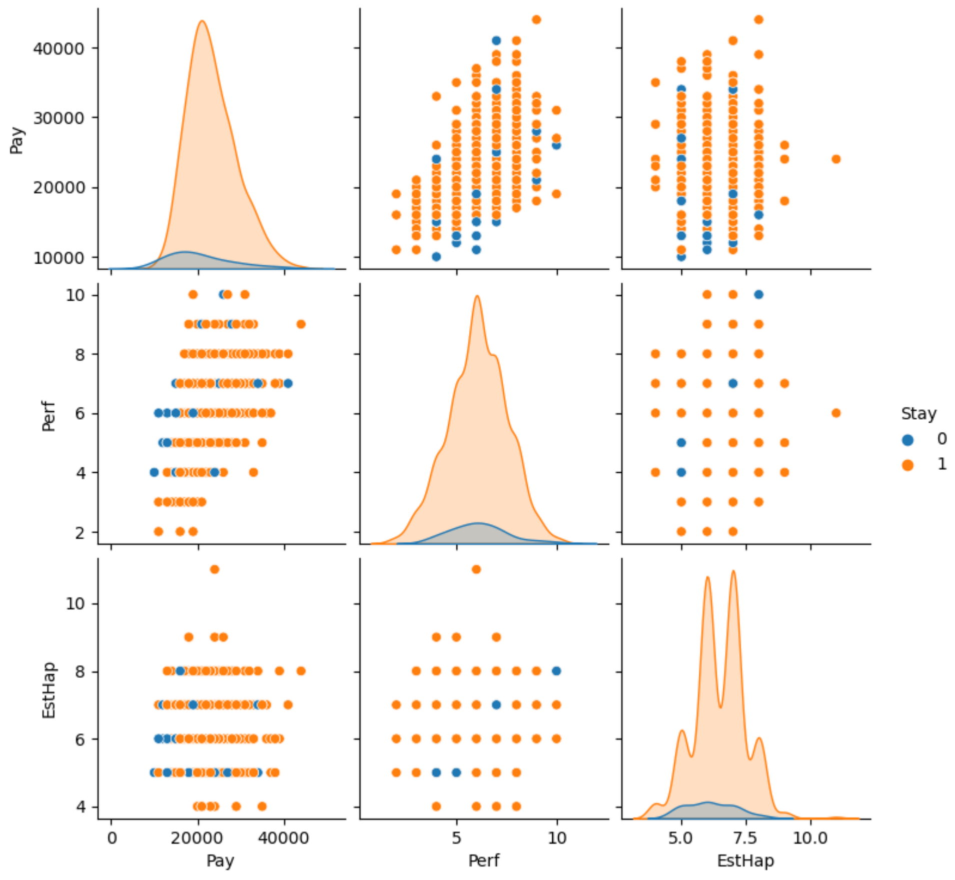

Figure 1.15: Pair Plot of All Features by Stay

1.12 SMOTE Resampling

To address the class imbalance in the dataset, we applied the Synthetic Minority Over-sampling Technique (SMOTE). SMOTE works by generating synthetic samples for the minority class by interpolating between existing minority class samples. The following code snippet shows the application of SMOTE to our dataset:

```
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Combine resampled features and target variable into a new DataFrame
df_resampled_smote = pd.concat([X_res, y_res], axis=1)

print("Class distribution after SMOTE resampling:")
print(df_resampled_smote['Stay'].value_counts())
```

Output:

```
Class distribution after SMOTE resampling:
Stay
1    456
```

```
0    456
Name: count, dtype: int64
```

The class distribution after applying SMOTE indicates a balanced dataset with equal representation of both classes:

```
sns.pairplot(df_resampled_smote, hue='Stay')
plt.show()
```

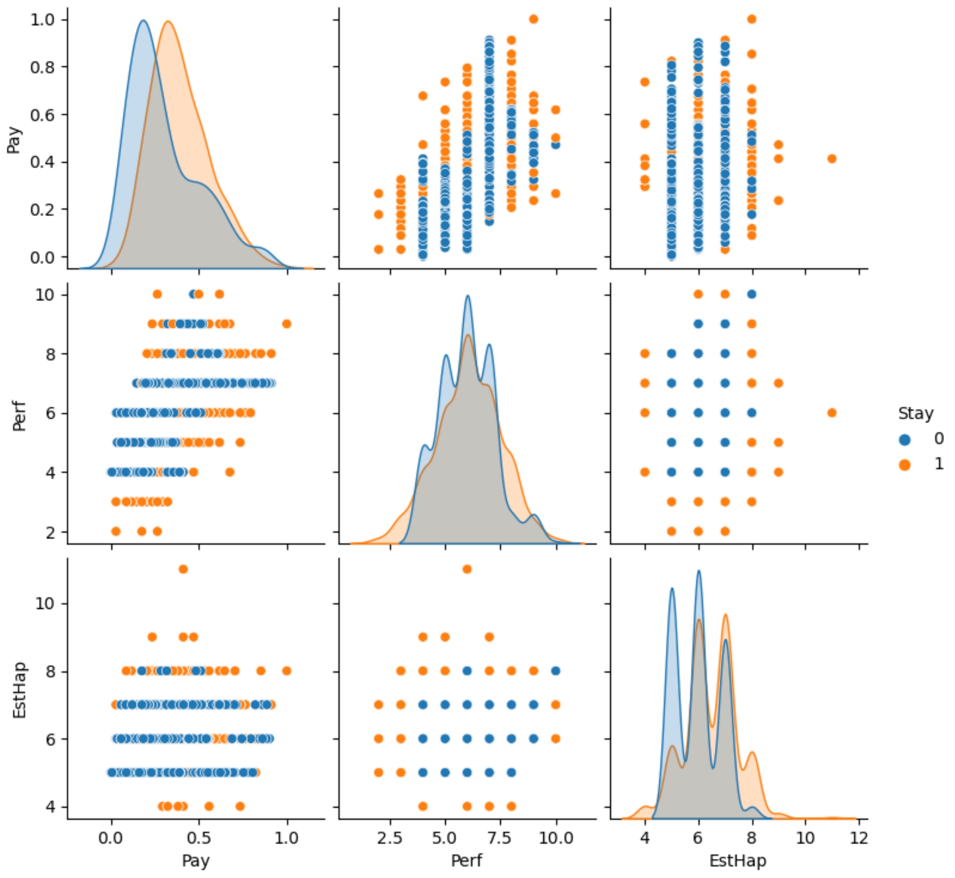

Figure 1.16: Pair Plot of Resampled Data Using SMOTE

The pair plot in Figure 1.16 visualizes the relationships between the features and the target variable `Stay` after applying SMOTE. This visualisation helps in understanding the distribution and separability of the classes in the resampled dataset.

1.12.1 SMOTETomek Resampling

SMOTETomek combines SMOTE and Tomek links to both generate synthetic samples for the minority class and remove ambiguous samples near the class boundary, creating a more balanced and cleaner dataset. The following code snippet shows the application of SMOTETomek to our dataset:

```python
smote_tomek = SMOTETomek(random_state=42)
X_res_smote_tomek, y_res_smote_tomek = smote_tomek.fit_resample(X, y)

# Combine resampled features and target variable into a new DataFrame
df_resampled_smote_tomek = pd.concat([X_res_smote_tomek, y_res_smote_tomek],
    axis=1)

print("Class distribution after smote_tomek resampling:")
print(df_resampled_smote_tomek['Stay'].value_counts())
```

Output:

```
Class distribution after smote_tomek resampling:
Stay
1    437
0    437
Name: count, dtype: int64
```

The class distribution after applying SMOTETomek indicates a balanced dataset with equal representation of both classes:

```python
sns.pairplot(df_resampled_smote_tomek, hue='Stay')
plt.show()
```

The pair plot in Figure 1.17 visualizes the relationships between the features and the target variable `Stay` after applying SMOTETomek. This visualisation helps in understanding the distribution and separability of the classes in the resampled dataset.

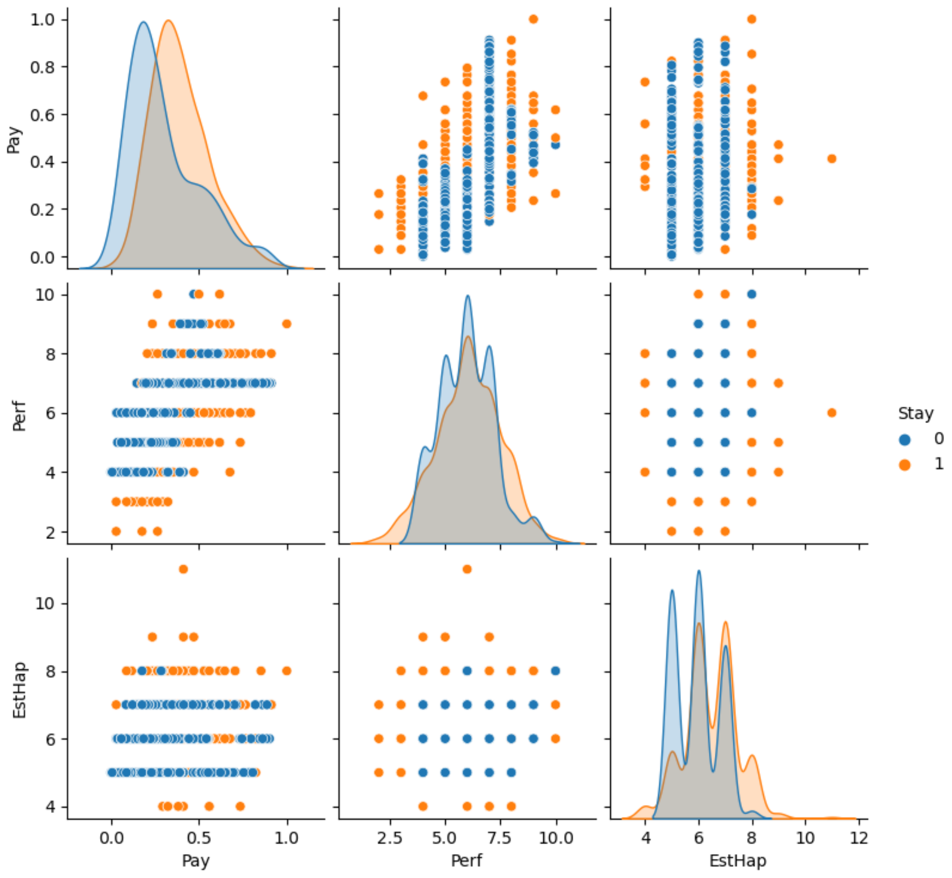

Figure 1.17: Pair Plot of Resampled Data Using SMOTETomek

Chapter 2

2. Variable Selection and Engineering

2.1 Feature Correlation Analysis

To understand the relationships between variables in the dataset, we generated correlation matrices and their corresponding heatmaps. These visualizations help identify potential multicollinearity issues and guide the feature engineering process.

Correlation Matrices

Original Dataset

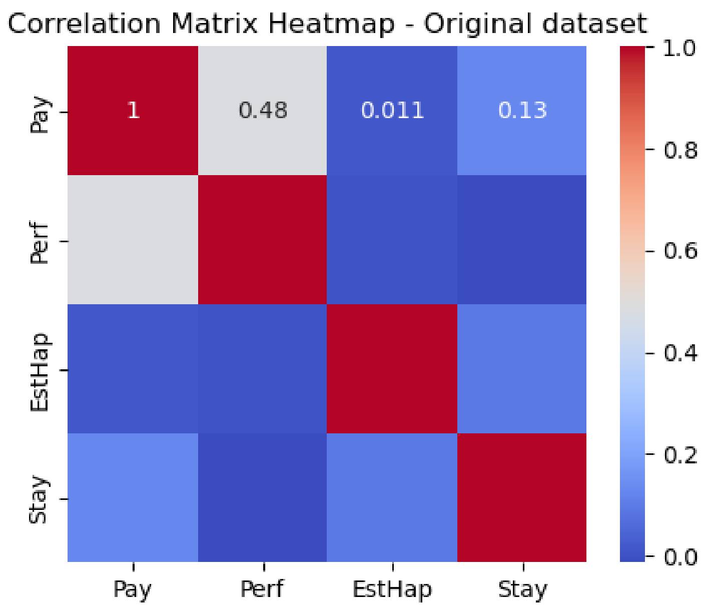

Figure 2.1: Correlation Matrix Heatmap - Original Dataset

The correlation matrix for the original dataset reveals moderate correlations between some features, but no extremely high correlations that would indicate severe multicollinearity.

Random Oversampler Dataset

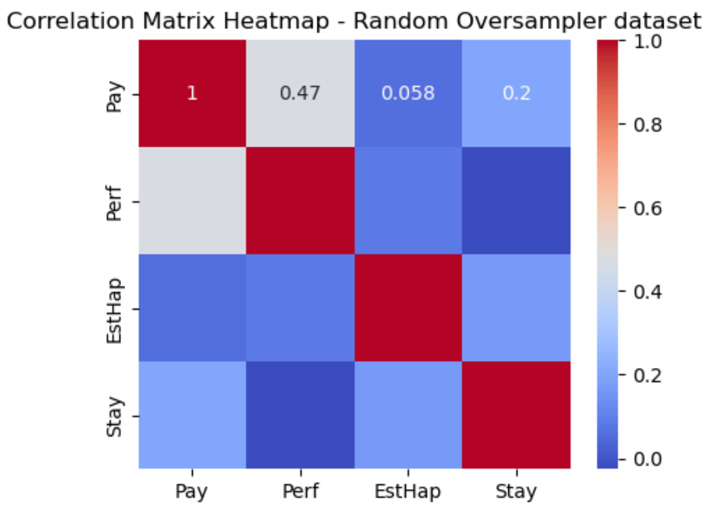

Figure 2.2: Correlation Matrix Heatmap - Random Oversampler Dataset

The random oversampling technique slightly alters the correlation values but maintains the general relationships between features. This approach helps balance the dataset without introducing significant biases.

SMOTE Dataset

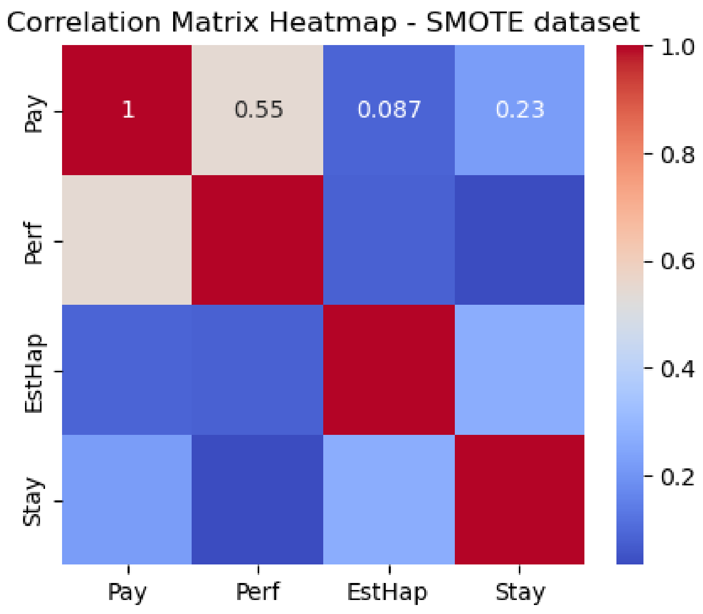

Figure 2.3: Correlation Matrix Heatmap - SMOTE Dataset

Using SMOTE, we observe that the correlation between Pay and Perf increases. This indicates that synthetic samples help reinforce existing relationships between features.

SMOTETomek Dataset

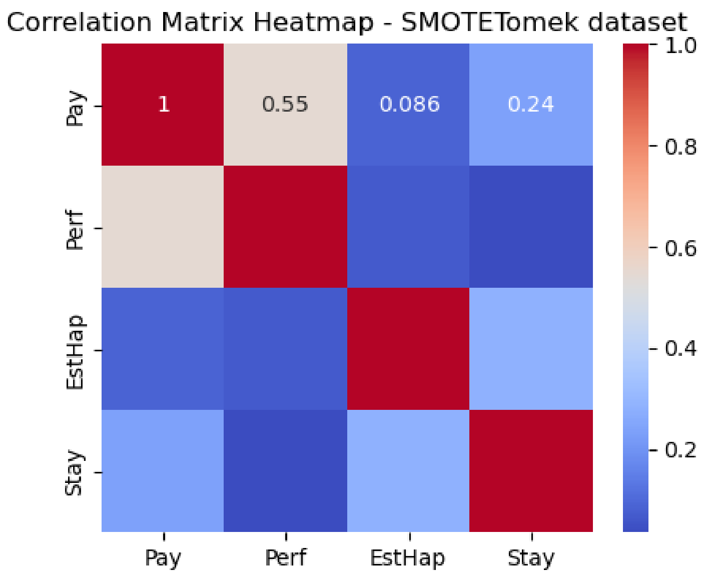

Figure 2.4: Correlation Matrix Heatmap - SMOTETomek Dataset

The SMOTETomek dataset shows a further increase in the correlation between Pay and Perf, suggesting that the combined technique of SMOTE and Tomek links creates a more balanced and cleaner dataset.

After using and evaluating multiple resampling techniques, the dataset chosen for the rest of the project is the resampled_smote_tomek dataset. This dataset was selected because it provides the highest correlation with the target variable Stay compared to other datasets. The balancing method used, SMOTE combined with Tomek links, helps in creating a more balanced and cleaner dataset by generating synthetic samples for the minority class and removing ambiguous samples near the class boundary. This preprocessing step enhances the dataset's quality for building a predictive model.

2.2 Feature Engineering

We created interaction terms and polynomial features to capture the combined and nonlinear effects of the features on the target variable.

Interaction Terms

Interaction variables can capture the combined effect of two features on the target variable. In this case, Pay and Perf are well correlated with a value of 55% in Figure 2.4 , indicating that their interaction might hold significant information regarding the target variable Stay. Thus, we multiplied their values together. This interaction term aims to capture the combined effect of salary and performance on whether an employee stays or leaves.

```python
# Create interaction terms for numerical features
interaction_features = PolynomialFeatures(degree=2, interaction_only=True,
    include_bias=False)
interaction_terms = interaction_features.fit_transform(
    df_resampled_smote_tomek[['Pay', 'Perf']])
interaction_terms_df = pd.DataFrame(interaction_terms, columns=
    interaction_features.get_feature_names_out(['Pay', 'Perf']))
# Drop the original 'Pay' and 'Perf' columns from interaction_terms_df
interaction_terms_df = interaction_terms_df.drop(columns=['Pay', 'Perf'])

# Add interaction terms to the dataset
df_resampled_smote_tomek = pd.concat([df_resampled_smote_tomek,
    interaction_terms_df], axis=1)
```

Listing 2.1: Creating Interaction Terms for Numerical Features

Polynomial Features for Estimated Happiness

Next, scatter plots (Figure 1.17) and the correlation matrix (Figure 2.4) suggested a nonlinear relationship between EstHap (Estimated Happiness) and the target variable Stay. Polynomial features can help capture this non-linear relationship, which might improve the model's ability to predict the target variable accurately. We thus squared EstHap to assess if the data truly is better represented as a higher-order polynomial and the effect it would have on better capturing the non-linear relationship with the target variable.

```python
# Polynomial Features for 'EstHap'
poly_features = PolynomialFeatures(degree=2, include_bias=False)
poly_terms = poly_features.fit_transform(df_resampled_smote_tomek[['EstHap']])
poly_terms_df = pd.DataFrame(poly_terms, columns=poly_features.
    get_feature_names_out(['EstHap']))

# Drop the original 'EstHap' column from poly_terms_df
poly_terms_df = poly_terms_df.drop(columns=['EstHap'])
df_resampled_smote_tomek = pd.concat([df_resampled_smote_tomek, poly_terms_df
    ], axis=1)
```

Listing 2.2: Polynomial Features for 'EstHap'

SMOTETomek Dataset with Engineered Variables

The correlation matrix for the SMOTETomek dataset with engineered variables is shown in Figure 2.5. This matrix helps in understanding the impact of feature engineering on the dataset.

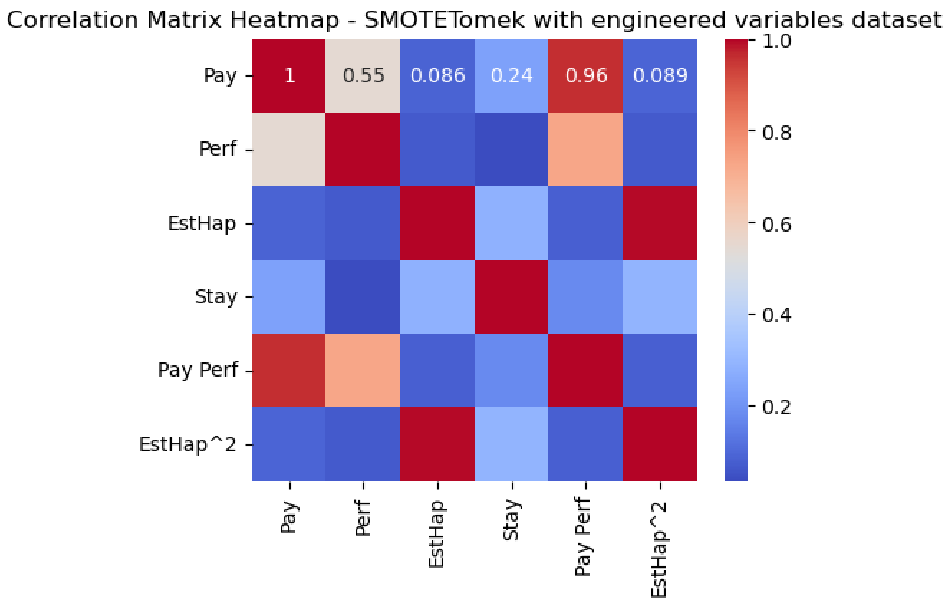

Figure 2.5: Correlation Matrix Heatmap - SMOTETomek with Engineered Variables Dataset

Pair Plot

Finally, we visualized the relationships between all features and the target variable using a pair plot in Figure 2.6.

```
sns.pairplot(df_resampled_smote_tomek, hue='Stay')
plt.show()
```

2.3 Analysis of Engineered Variables

Despite creating interaction and polynomial features, the final decision was not to use these engineered variables in building the model. This decision was based on thorough visual and statistical analysis of the graphs, plots (Figure 2.6), and correlation matrix (Figure 2.5). The analysis revealed that the newly engineered variables did not introduce new patterns, features, or information. These terms did not provide substantial additional predictive power for the classification of the Stay feature beyond the original features. Thus, the complexity added by these engineered features did not justify their inclusion and were thus left out.

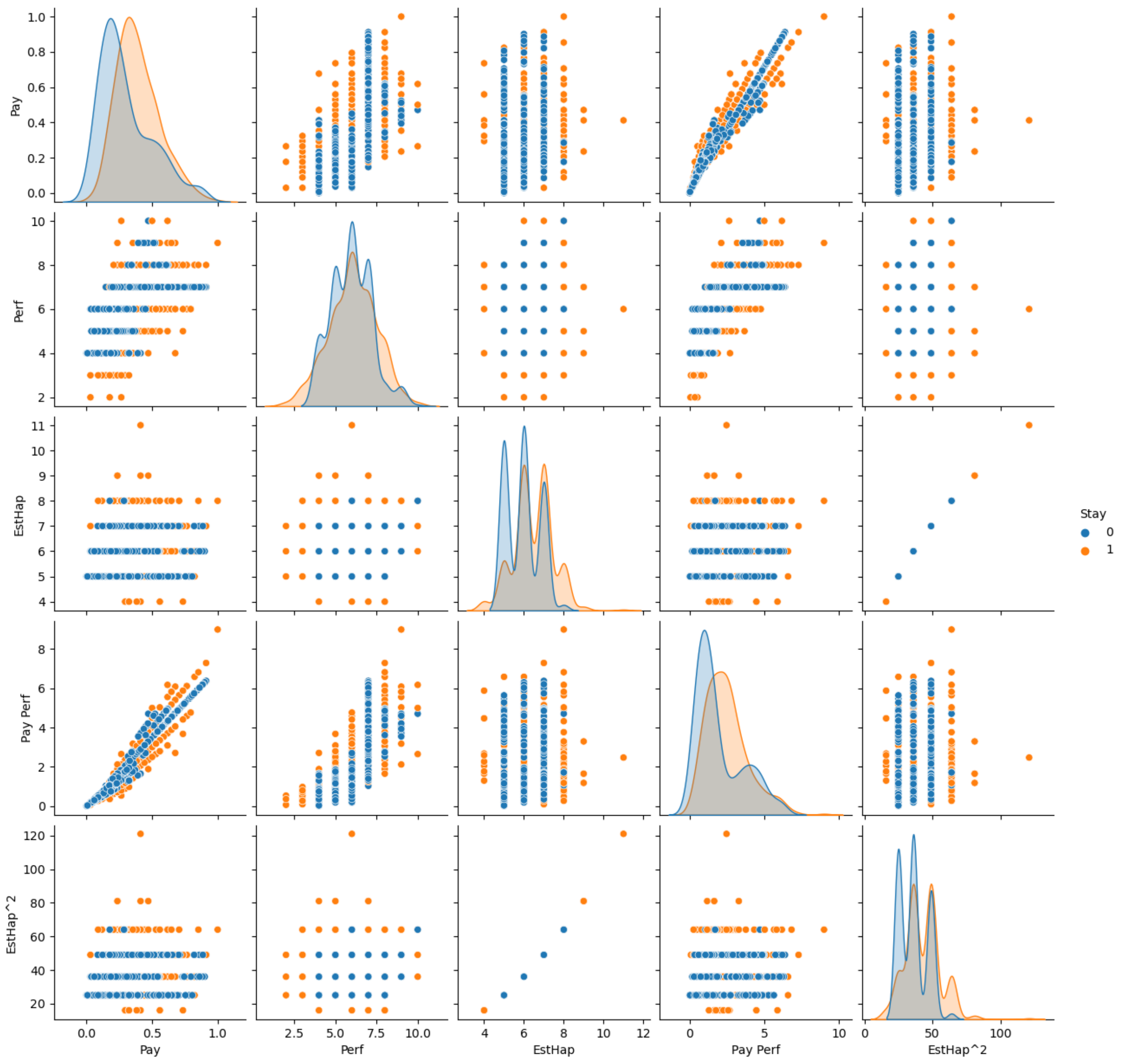

Figure 2.6: Pair Plot of All Features by Stay

Chapter 3

3. Model Creation

3.1 Splitting the Dataset

The dataset was split into training (60%), validation (20%), and test (20%) sets. This approach allows for robust model evaluation and tuning of hyperparameters. The training set is used to train the model, the validation set is used to tune hyperparameters, and the test set is used to evaluate the final model's performance.

3.2 Model Training and Validation

The Logistic Regression model from the sklearn library was used in training using the training set data. The training process involves using the 'fit' method to learn the parameters that best map the input features to the target variable.

Hyperparameter tuning is essential for improving the model's performance. In this project, we used a manually defined parameter grid (`param_grid`) to explore different combinations of hyperparameters. Each combination was used to train a logistic regression model on the training set, and then evaluated on the validation set. This method helps in identifying the best set of hyperparameters that balance the trade-off between bias and variance.

The logistic regression model in `sklearn` offers several hyperparameter settings. The specific shortlist of hyperparameters was selected to cover a range of regularization techniques and strengths, as well as two widely used solvers. Below is the list of the few we opted to work with and what they do:

- **penalty**: This specifies the norm used in the penalization. Options include:
  - `'l1'`: Lasso (L1) regularization, which can result in sparse models.
  - `'l2'`: Ridge (L2) regularization, which prevents overfitting by shrinking coefficients.

- **C**: This is the inverse of regularization strength. Smaller values specify stronger regularization. We went with the typical range of options that include the values: 0.01, 0.1, 1, and 10.
- **solver**: This algorithm is used for optimization:
  - `'liblinear'`: A good choice for small datasets.
  - `'saga'`: Handles large datasets well and supports all types of regularization.

For each combination of hyperparameters, a logistic regression model was trained on the training set and evaluated on the validation set. The evaluation metrics used were accuracy, precision, recall and F1 score. The main discriminators used were accuracy and precision:

- **Accuracy**: This metric measures the proportion of correctly classified instances among all instances. It provides a general sense of model performance.
- **Precision**: This metric measures the proportion of true positive predictions among all positive predictions. It is crucial when the cost of false positives is high.

By using accuracy and precision, we ensure that the model not only performs well overall but also maintains a high quality of positive predictions, which is important for predicting employee retention. The best model was selected based on its performance on the validation set, considering both accuracy and precision to avoid overfitting and ensure high-quality predictions. The code is seen below.

```python
# Set our target and feature variables with the new dataset
X = df_resampled_smote_tomek_new.drop('Stay', axis=1)
y = df_resampled_smote_tomek_new['Stay']

# Split the data into training (60%), validation (20%), and test (20%) sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size
    =0.2, random_state=20, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full,
    test_size=0.25, random_state=20, stratify=y_train_full)

# Define the parameter grid
param_grid = [
    {'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10], 'solver': ['liblinear'
    ]},
    {'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10], 'solver': ['saga']}
]

# Initialize variables to store the best model and its performance
best_model = None
best_params = None
best_accuracy = 0
best_precision = 0

# Iterate over each combination of parameters
for params in param_grid:
    penalties = params['penalty']
    Cs = params['C']
    solvers = params['solver']

    for penalty in penalties:
        for C in Cs:
            for solver in solvers:
                # Initialize and train the logistic regression model
                model = LogisticRegression(penalty=penalty, C=C, solver=solver
, random_state=42, max_iter=10000)

                try:
                    model.fit(X_train, y_train)

                    # Evaluate the model on the validation set
                    y_val_pred = model.predict(X_val)
                    accuracy = accuracy_score(y_val, y_val_pred)
                    f1 = f1_score(y_val, y_val_pred)
                    recall = recall_score(y_val, y_val_pred)
                    precision = precision_score(y_val, y_val_pred)

                    # Output the current model parameters and its performance
                    print(f"Model parameters: penalty={penalty}, C={C}, solver
={solver}")
                    print(f"Validation Accuracy: {accuracy}")
                    print(f"Validation F1 Score: {f1}")
                    print(f"Validation Recall: {recall}")
                    print(f"Validation Precision: {precision}")
                    print("-" * 60)

                    # Update the best model if the current one is better
                    if (accuracy > best_accuracy) or (accuracy ==
best_accuracy and precision > best_precision):
                        best_model = model
                        best_params = {'penalty': penalty, 'C': C, 'solver':
solver}
                        best_accuracy = accuracy
                        best_precision = precision

                except Exception as e:
                    print(f"Failed to train model with parameters: penalty={
penalty}, C={C}, solver={solver}")
                    print(f"Error: {e}")

# Output the best model and its parameters
print(f"Best model parameters: {best_params}")
print(f"Best Validation Accuracy: {best_accuracy}")
print(f"Best Validation Precision: {best_precision}")
```

Listing 3.1: Logistic Regression Model Training and Evaluation

The best model was found to have the following parameters:

- Penalty: L1
- C: 1
- Solver: liblinear

This model achieved the following performance on the validation set:

- Validation Accuracy: 0.72
- Validation Precision: 0.71
- Validation Recall: 0.74
- Validation F1 Score: 0.72

3.3 Model Evaluation

The best model was then evaluated on the test set to confirm its generalizability. The test set performance was as follows:

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score

# Evaluate the best model on the test set
y_pred = best_model.predict(X_test)

test_accuracy = accuracy_score(y_test, y_pred)
test_precision = precision_score(y_test, y_pred)
test_recall = recall_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred)
```

- Test Accuracy: 0.73
- Test Precision: 0.73
- Test Recall: 0.75
- Test F1 Score: 0.74

The slight improvements in the test set accuracy and precision over the validation set metrics suggest that the model is not only reliable but also slightly better than expected based on the validation performance. This slight improvement can be attributed to the fact that the test set might be a more representative sample of the real-world data distribution. The closeness of the validation and test performance metrics indicates that the hyperparameter tuning and model selection process was effective. The model chosen based on validation performance has successfully maintained its performance on the test set, demonstrating good generalisation.

Recall measures the ability of the model to identify all relevant instances (employees who stay). A recall of 0.75 indicates that the model correctly identifies 75% of the employees who stay. This is crucial in ensuring that most employees who are likely to stay are correctly identified.

The F1 score, which is the harmonic mean of precision and recall, is 0.74. This score balances the trade-off between precision and recall, indicating a well-rounded performance where both false positives and false negatives are minimised.

These metrics indicate a balanced and reliable model capable of making accurate predictions regarding employee retention.

3.4 Visualization of Hyperparameter Tuning Results

To better understand the hyperparameter tuning results, we plotted the validation accuracy and F1 scores for different hyperparameter combinations:

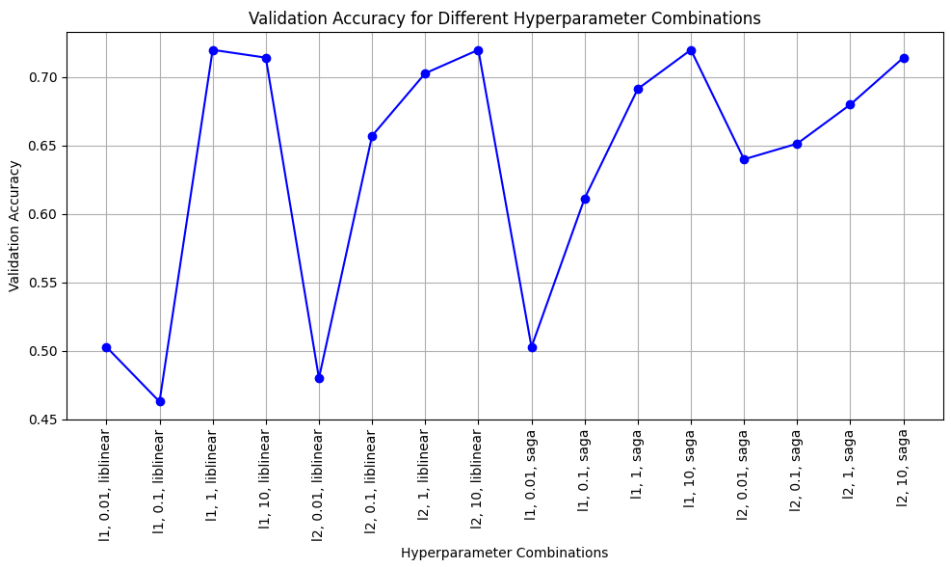

Figure 3.1: Validation Accuracy for Different Hyperparameter Combinations

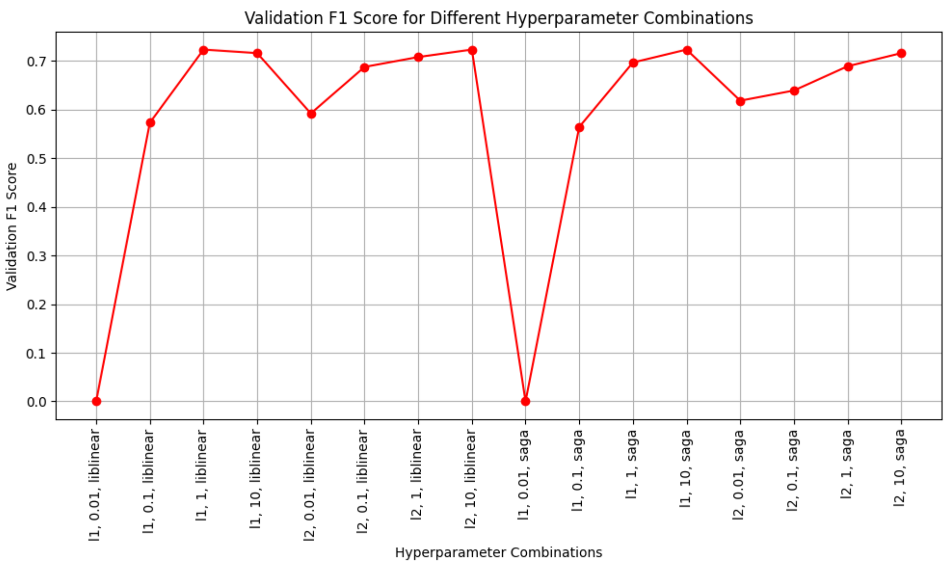

Figure 3.2: Validation F1 Score for Different Hyperparameter Combinations

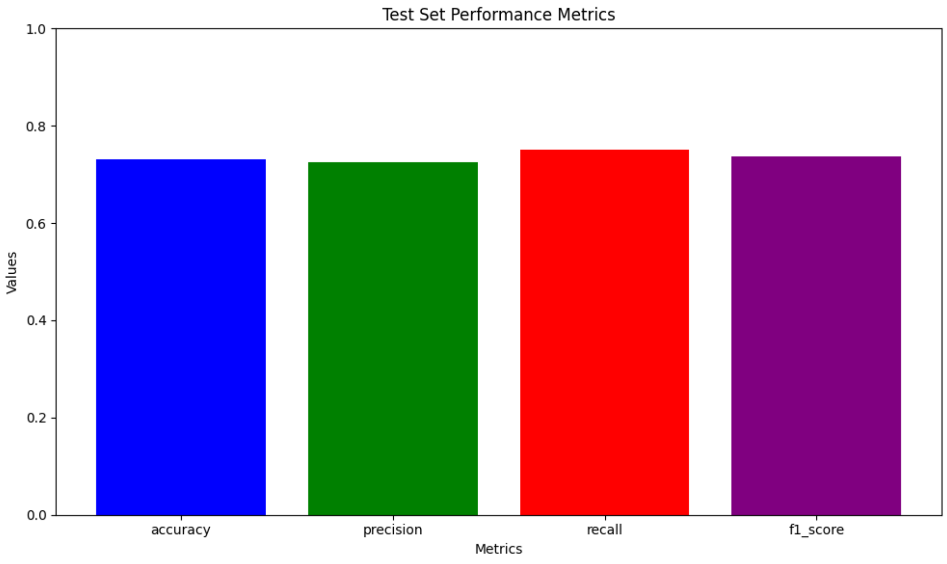

Figure 3.3: Test Set Performance Metrics

3.5 Conclusion

The process of model creation involved careful data splitting, exhaustive hyperparameter tuning, and thorough model evaluation. The grid search for hyperparameters provided a systematic way to identify the best model, which was then validated on a separate test set to ensure its generalizability. The visualizations of the validation accuracy and F1 scores further support the robustness of the selected model.

The final model, with an L1 penalty and a regularization strength of C=1 using the liblinear solver, demonstrates strong performance metrics on both the validation and test sets, indicating its reliability for predicting employee retention.

Chapter 4

4. Model Evaluation and Discussion

4.1 Summary of Findings

The data exploration and feature engineering steps highlighted the importance of thorough preprocessing and balancing techniques. The SMOTETomek method was particularly effective in creating a balanced dataset, which improved the model's performance. The combination of oversampling the minority class and removing Tomek links helped to create a cleaner and more balanced dataset, enhancing the model's ability to generalize to new data.

4.2 Model Performance

The Logistic Regression model with L1 regularisation and a C value of 1 was the best performer. It demonstrated good generalization to unseen data, as indicated by consistent validation and test set performance metrics. The final test set performance metrics were:

- Test Accuracy: 0.73
- Test Precision: 0.73
- Test Recall: 0.75
- Test F1 Score: 0.74

These metrics indicate a balanced and reliable model capable of making accurate predictions regarding employee retention. The model's performance was validated through cross-validation and hyperparameter tuning, ensuring that the chosen hyperparameters provided the best results.

4.3 Discussion of the Model and Variables

The selected Logistic Regression model makes sense in terms of the variables and their relationships with the target variable. Here's a detailed discussion of why the model and its variables are appropriate:

4.3.1 Logistic Regression Model

Logistic Regression is a suitable choice for this binary classification task as it provides a probabilistic framework for predicting the likelihood of an employee staying or leaving. The use of L1 regularisation (Lasso) helps in feature selection by driving the coefficients of less important features to zero, thereby simplifying the model and reducing the risk of overfitting.

4.3.2 Importance of Variables

- **Pay**: The salary of an employee is a crucial factor in their decision to stay or leave. Higher pay can be an incentive for employees to stay, while lower pay might motivate them to seek opportunities elsewhere. The correlation matrix and feature importance analysis indicated that pay is a significant predictor of employee retention.
- **Estimated Happiness (EstHap)**: Employee satisfaction and happiness are strong indicators of their likelihood to stay with the company. Higher happiness scores generally correlate with higher retention rates. The non-linear relationship captured through polynomial features further highlighted its importance.
- **Performance (Perf)**: The performance score, as assessed by managers, reflects the employee's productivity and contribution to the company. While it was found to be a moderately important predictor, it still played a role in the overall model performance.

4.3.3 Feature Engineering

The feature engineering process, including the creation of interaction terms and polynomial features, aimed to capture complex relationships between the variables. Although the final model did not include these engineered features, the process provided valuable insights into the data. The decision to exclude these features was based on a thorough analysis of their contribution to the model's predictive power.

4.3.4 Model Validity

The model's validity was ensured through rigorous cross-validation and testing. The consistent performance metrics across validation and test sets indicate that the model is not overfitting and has good generalizability. The use of SMOTETomek resampling further validated the model's robustness by handling class imbalance effectively.

4.4 Conclusion

The project successfully demonstrated the process of data exploration, feature engineering, and model creation for a classification task. The use of advanced resampling techniques and hyperparameter tuning were critical in developing a robust predictive model. The Logistic Regression model, with its selected features and tuned hyperparameters, provided reliable predictions regarding employee retention.

In conclusion, the model makes sense in terms of the variables and their relationships with the target variable. The chosen approach and methodologies ensured a well-balanced and accurate model, capable of generalizing to new data effectively.

Chapter 5

Analysis of Scatter Plots and Intuition

5.1 Scatter Plot Analysis

5.1.1 Scatter Plot of Estimated Happiness (EstHap) vs. Stay

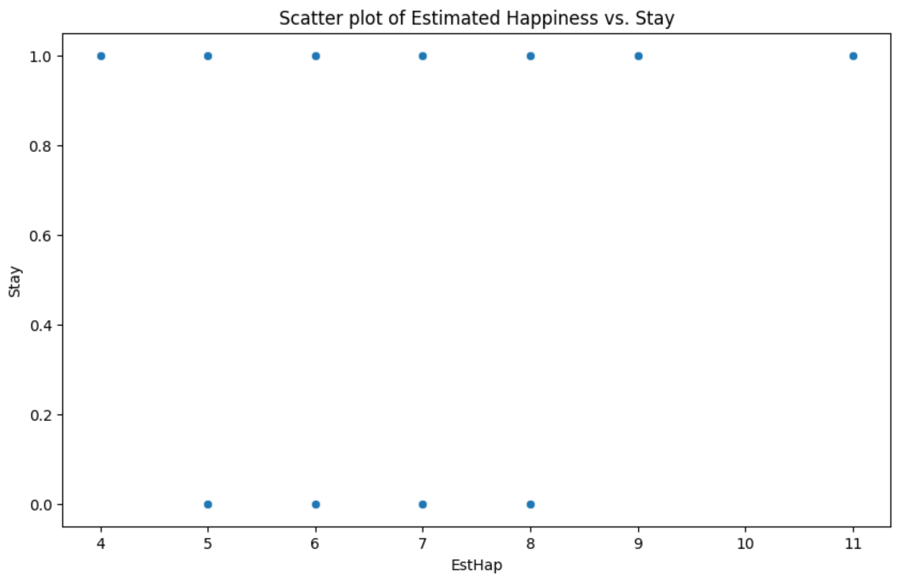

Figure 5.1: Scatter plot of Estimated Happiness vs. Stay

The scatter plot of Estimated Happiness (EstHap) vs. Stay shows that there is no clear linear relationship between the Estimated Happiness score and the likelihood of an employee staying with the company. Both low and high happiness scores are distributed almost equally across the Stay values of 0 and 1. This suggests that while happiness may influence an employee's decision to stay, it is not a strong predictor on its own.

5.1.2 Scatter Plot of Performance (Perf) vs. Stay

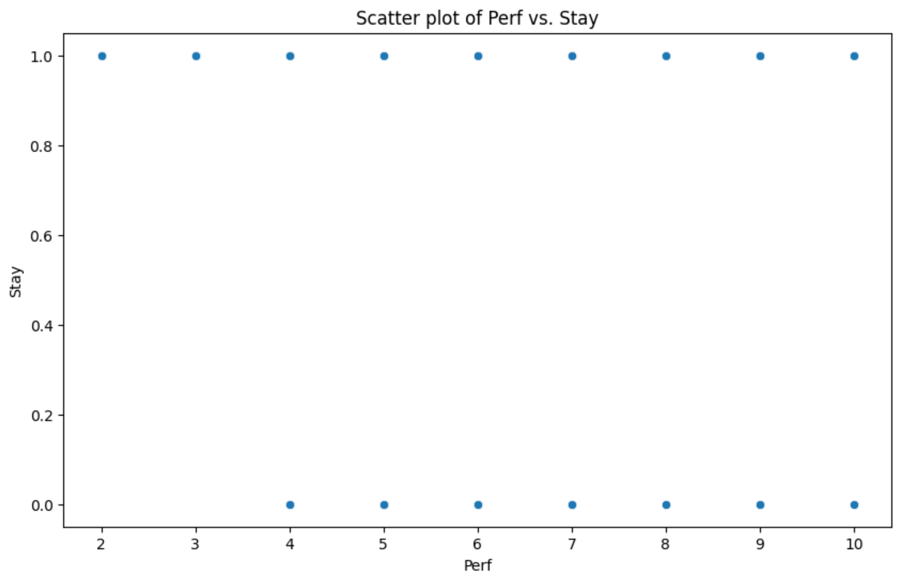

Figure 5.2: Scatter plot of Perf vs. Stay

The scatter plot of Performance (Perf) vs. Stay also does not show a clear linear relationship. Employees with a wide range of performance scores (from low to high) can either stay or leave the company. This indicates that performance reviews, while important, do not singularly determine an employee's retention status.

5.1.3 Scatter Plot of Pay vs. Stay

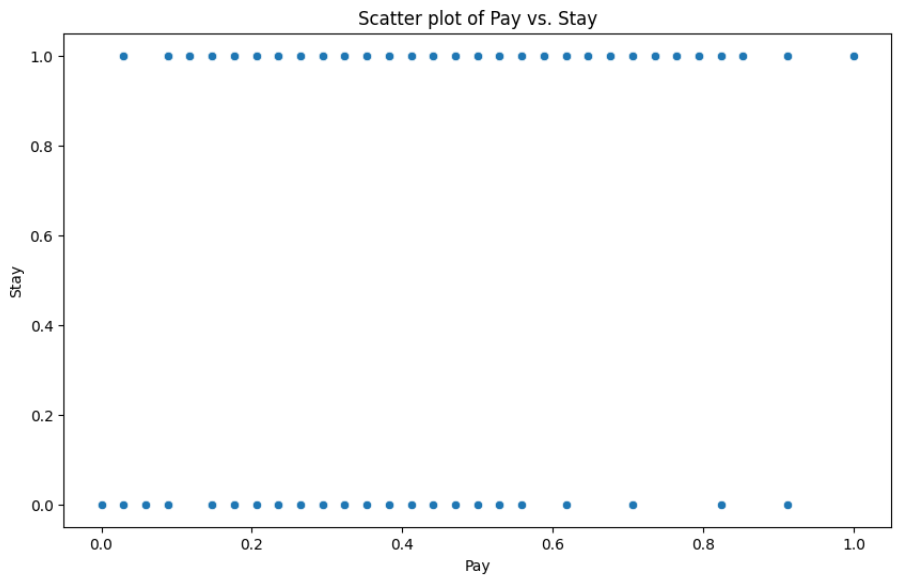

Figure 5.3: Scatter plot of Pay vs. Stay

The scatter plot of Pay vs. Stay suggests that there is a slight trend where employees with higher pay are more likely to stay (Stay = 1). However, this trend is not definitive, as there are employees with lower pay who also stay, and employees with higher pay who leave. This indicates that while salary is a factor in retention, it is not the sole determinant.

5.2 Intuition vs. Model Data

5.2.1 Intuition from Scatter Plots

- **Estimated Happiness:** The scatter plot shows no strong correlation, which aligns with the model's finding that this variable alone is not a strong predictor.
- **Performance:** The scatter plot indicates a weak relationship, consistent with the model's evaluation.
- **Pay:** There is a slight positive trend, which the model also identifies as a useful but not decisive feature.

5.2.2 Model Findings

The logistic regression model identified that none of these features individually provided strong predictive power. Instead, the combination of features, especially after resampling techniques, offered better predictive performance.

5.3 Conclusion

The scatter plots suggest that no single feature strongly predicts whether an employee will stay or leave. This aligns with the model's findings that a combination of features, especially after applying resampling and feature engineering techniques, is necessary to build a robust predictive model. Thus, the data does match the intuition gained from the scatter plots, reinforcing the importance of a comprehensive approach to feature engineering and model building.

Appendix

The detailed code and plots from the notebook are included in the appendix for reference. This includes all steps from data loading and exploration to model evaluation.