Employee promotion prediction using KNN

Problem statement: Using K-Nearest Neighbours (KNN) classification, we aim to predict whether an employee will be promoted based on their individual details. In this HR dataset, the company promotes 5% of its employees each year, and our task is to identify eligible candidates for promotion. KNN will help us achieve this by analyzing the characteristics of employees and classifying them as potential candidates for promotion or not.

Data Preparation in Machine Learning:

In our project, we employed a structured approach to data preparation, a critical step in ensuring the reliability and accuracy of our machine learning model. This process was encapsulated within a Python class, DataPreparation, designed to streamline the handling of our dataset. The class was initialized with paths to both training and testing data, reflecting our commitment to a robust evaluation of the model's performance.

Initialization and Data Loading

The DataPreparation class was initialized with two parameters: train_path and test_path, representing the file paths to the training and testing datasets, respectively. This design choice allowed for flexibility and ease of use, accommodating different datasets as needed. Upon initialization, the class loaded the data from these CSV files into two separate DataFrame objects, train_df and test_df, using the Pandas library. This step was crucial for subsequent data manipulation and analysis.

```
class DataPreparation:
    """
    Class for preparing the data.
    """
    def __init__(self, train_path, test_path):
        """
        Initialize DataPreparation with train and test file paths.

        Args:
            train_path (str): Path to the training data CSV file.
            test_path (str): Path to the testing data CSV file.
        """
        self.train_path = train_path
        self.test_path = test_path
        self.train_df = None
        self.test_df = None

    def load_data(self):
        """
        Load data from CSV files into train_df and test_df.
        """
        self.train_df = pd.read_csv(self.train_path)
        self.test_df = pd.read_csv(self.test_path)
        print("Data loaded successfully.")
```

Removing Duplicates

Our methodology included a stringent step to remove duplicate entries from both datasets. Duplicate data can lead to biased or inaccurate models, as it may overemphasize certain patterns or

features. By dropping these duplicates, we ensured that each data point was unique, thereby maintaining the integrity of our analysis.

```
def remove_duplicates(self):
    """
    Remove duplicate rows from train_df and test_df.
    """
    self.train_df.drop_duplicates(inplace=True)
    self.test_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
```

Handling Missing Values

A significant aspect of our data preparation involved handling missing values, a common challenge in real-world datasets. Our approach was twofold, treating categorical and numerical data differently:

For categorical columns, we replaced missing values with the mode (the most frequently occurring value) of the respective column in the training dataset. This method was chosen as it preserves the distribution of categorical data, which is often crucial in classification tasks.

For numerical columns, we used the median of the column from the training dataset as a replacement for missing values. The median, being less sensitive to outliers than the mean, was selected to maintain the central tendency of the data without being skewed by extreme values.

This nuanced approach to handling missing values was applied consistently across both the training and testing datasets, ensuring uniformity in data treatment.

```
def handle_missing_values(self):
    """
    Handle missing values in the data.
    """
     # Handle missing values in train_df
    for column in self.train_df.columns:
        if self.train_df[column].dtype == 'object':
            mode_val = self.train_df[column].mode()[0]
            self.train_df[column].fillna(mode_val, inplace=True)
        else:  # Numeric columns
            median_val = self.train_df[column].median()
            self.train_df[column].fillna(median_val, inplace=True)

    # Handle missing values in test_df
    for column in self.test_df.columns:
        if self.test_df[column].dtype == 'object':
            mode_val = self.train_df[column].mode()[0]  # Use mode from train_df
            self.test_df[column].fillna(mode_val, inplace=True)
        else:  # Numeric columns
            median_val = self.train_df[column].median()  # Use median from train_df
            self.test_df[column].fillna(median_val, inplace=True)

    print("Missing values handled.")
```

Error Handling

To ensure the robustness of our data preparation process, we incorporated error handling mechanisms. This was achieved through a try-except block, which captured and reported any exceptions or anomalies encountered during data loading and processing. This not only helped in debugging but also ensured that our data preparation pipeline was resilient to unexpected errors.

```
try:
    # Initialize and load data
    data_prep = DataPreparation(train_path="C:\\Users\\vishn\\Downloads\\train.csv.zip",
                                test_path="C:\\Users\\vishn\\Downloads\\test.csv.zip")
    data_prep.load_data()
    data_prep.remove_duplicates()
    data_prep.handle_missing_values()

except Exception as e:
    print(f"An error occurred: {e}")
```

Comprehensive Machine Learning Workflow for Predictive Analysis

Our study employed a structured machine learning workflow, encompassing data visualization, transformation, model training, evaluation, and application. This workflow was implemented through a series of Python classes, each focusing on a specific aspect of the process.

Data Visualization

The DataVisualization class was instrumental in exploring and understanding our dataset. Upon initialization with training and testing dataframes, the class executed a series of steps:

Data Summary and Structure Analysis: The class began by printing descriptive statistics, initial records, and structural information of both datasets, providing an overview of the data's nature and composition. **Train and test df data columns:**

```
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   employee_id           54808 non-null  int64
 1   department            54808 non-null  object
 2   region                54808 non-null  object
 3   education             54808 non-null  object
 4   gender                54808 non-null  object
 5   recruitment_channel   54808 non-null  object
 6   no_of_trainings       54808 non-null  int64
 7   age                   54808 non-null  int64
 8   previous_year_rating  54808 non-null  float64
 9   length_of_service     54808 non-null  int64
 10  KPIs_met >80%         54808 non-null  int64
 11  awards_won?           54808 non-null  int64
 12  avg_training_score    54808 non-null  int64
 13  is_promoted           54808 non-null  int64
dtypes: float64(1), int64(8), object(5)
memory usage: 5.9+ MB
```

```
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   employee_id           23490 non-null  int64
 1   department            23490 non-null  object
 2   region                23490 non-null  object
 3   education             23490 non-null  object
 4   gender                23490 non-null  object
 5   recruitment_channel   23490 non-null  object
 6   no_of_trainings       23490 non-null  int64
 7   age                   23490 non-null  int64
 8   previous_year_rating  23490 non-null  float64
 9   length_of_service     23490 non-null  int64
 10  KPIs_met >80%         23490 non-null  int64
 11  awards_won?           23490 non-null  int64
 12  avg_training_score    23490 non-null  int64
dtypes: float64(1), int64(7), object(5)
memory usage: 2.3+ MB
```

Visual Exploration: Utilizing Seaborn and Matplotlib, the class generated various plots to visually analyze the data. These included count plots for categorical variables like 'department', 'region', 'education', and 'gender', highlighting their distribution and relationship with the target variable 'is_promoted'. Boxplots and violin plots were also used to explore numerical variables and their association with promotion status.

```
        # Various plots (using Seaborn and Matplotlib)
        sns.countplot(data=self.train_df, x='is_promoted')
        plt.show()

        numerical_cols = self.train_df.select_dtypes(include=[np.number]).columns


        plt.figure(figsize=(12, 6))
        sns.countplot(data=self.train_df, x='department', hue='is_promoted')
        plt.xticks(rotation=45)
        plt.title('Promotion Status by Department')
        plt.show()

        # Bar plot for 'region' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.countplot(data=self.train_df, x='region', hue='is_promoted')
        plt.xticks(rotation=45)
        plt.title('Promotion Status by Region')
        plt.show()

        # Bar plot for 'education' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.countplot(data=self.train_df, x='education', hue='is_promoted')
        plt.xticks(rotation=45)
        plt.title('Promotion Status by Education')
        plt.show()

        # Bar plot for 'gender' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.countplot(data=self.train_df, x='gender', hue='is_promoted')
        plt.title('Promotion Status by Gender')
        plt.show()

        # Boxplot for 'recruitment_channel' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.boxplot(data=self.train_df, x='recruitment_channel', y='is_promoted')
        plt.title('Promotion Status by Recruitment Channel')
        plt.show()
```

```
        # Boxplot for 'recruitment_channel' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.boxplot(data=self.train_df, x='recruitment_channel', y='is_promoted')
        plt.title('Promotion Status by Recruitment Channel')
        plt.show()

        # Boxplot for 'no_of_trainings' and 'avg_training_score'
        plt.figure(figsize=(12, 6))
        sns.boxplot(data=self.train_df, x='no_of_trainings', y='avg_training_score'
        plt.title('Average Training Score by Number of Trainings')
        plt.show()
```

```
        # Boxplot for 'age' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.boxplot(data=self.train_df, x='is_promoted', y='age')
        plt.title('Age Distribution by Promotion Status')
        plt.show()

        # Violin plot for 'age' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.violinplot(data=self.train_df, x='is_promoted', y='age')
        plt.title('Age Distribution by Promotion Status - Violin Plot')
        plt.show()

        # Violin plot for 'previous_year_rating' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.violinplot(data=self.train_df, x='is_promoted', y='previous_year_rating
        plt.title('Previous Year Rating by Promotion Status')
        plt.show()
```

```
        # Violin plot for 'length_of_service' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.violinplot(data=self.train_df, x='is_promoted', y='length_of_service')
        plt.title('Length of Service by Promotion Status')
        plt.show()

        # Bar plot for 'KPIs_met..80.' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.countplot(data=self.train_df, x='KPIs_met >80%', hue='is_promoted')
        plt.title('Promotion Status by KPIs Met')
        plt.show()

        # Bar plot for 'awards_won.' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.countplot(data=self.train_df, x='awards_won?', hue='is_promoted')
        plt.title('Promotion Status by Awards Won')
        plt.show()

        # Boxplot for 'avg_training_score' and 'is_promoted'
        plt.figure(figsize=(12, 6))
        sns.boxplot(data=self.train_df, x='is_promoted', y='avg_training_score')
        plt.title('Average Training Score by Promotion Status')
        plt.show()

        # Boxplot for numerical variables
        plt.figure(figsize=(12, 8))
        sns.boxplot(data=self.train_df[numerical_cols])
        plt.title('Boxplot of Numerical Variables')
        plt.show()
```

```
try:
    # Visualize data
    data_viz = DataVisualization(data_prep.train_df,data_prep.test_df)

except Exception as e:
    print(f"An error occurred: {e}")
```

These plots are intended to provide insights into the dataset, especially in the context of predicting promotion status (is_promoted). Here's a description of each plot:

Countplot of 'is_promoted':

This is a basic countplot that shows the distribution of the is_promoted variable, indicating the number of employees who were promoted (1) and those who were not (0) among 50000.

Countplot of 'department' vs. 'is_promoted':

This plot displays the distribution of promotion status within different departments. It helps visualize whether promotion rates vary across departments.

Countplot of 'region' vs. 'is_promoted':

Similar to the previous plot, this one explores the distribution of promotion status across different regions.

Countplot of 'education' vs. 'is_promoted':
Examines the distribution of promotion status among employees with different education levels.

Countplot of 'gender' vs. 'is_promoted':
Illustrates the distribution of promotion status based on gender.

Boxplot of 'recruitment_channel' vs. 'is_promoted':
Uses boxplots to visualize the relationship between the recruitment channel and promotion status.

Boxplot of 'no_of_trainings' vs. 'avg_training_score':
Explores the relationship between the number of trainings and the average training score, helping to identify potential trends.

Boxplot of 'age' vs. 'is_promoted':
This plot compares the age distribution of employees who were promoted and those who were not.

Violin plot of 'age' vs. 'is_promoted':
Similar to the previous plot but uses a violin plot to visualize the age distribution more comprehensively.

Violin plot of 'previous_year_rating' vs. 'is_promoted':
Shows the distribution of previous year ratings for promoted and non-promoted employees.

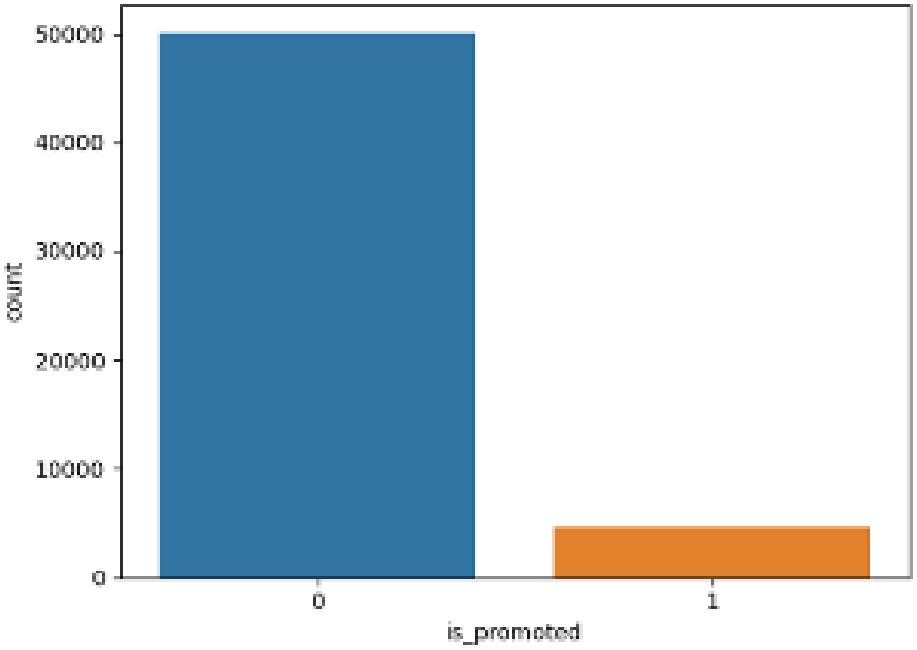
Violin plot of 'length_of_service' vs. 'is_promoted':
Examines how the length of service varies between promoted and non-promoted employees.

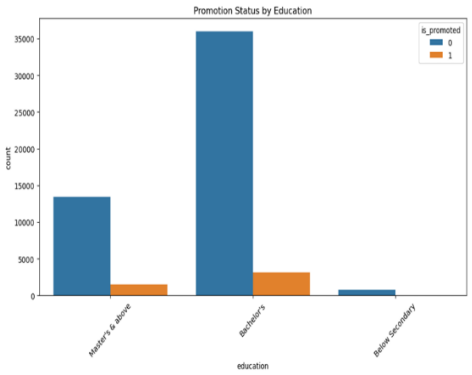
Countplot of 'KPIs_met >80%' vs. 'is_promoted':
Analyzes the distribution of promotion status based on whether employees met more than 80% of their Key Performance Indicators (KPIs).

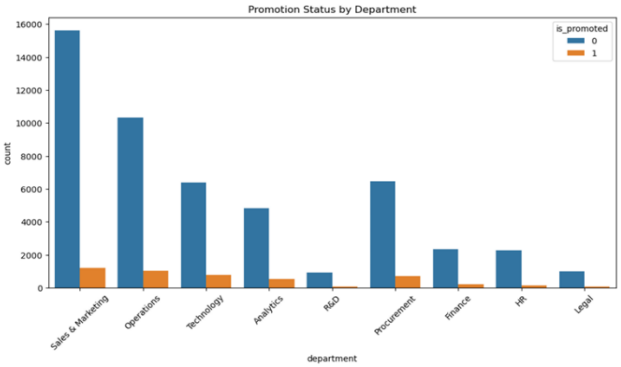
Countplot of 'awards_won?' vs. 'is_promoted':
Visualizes the relationship between winning awards and promotion status.

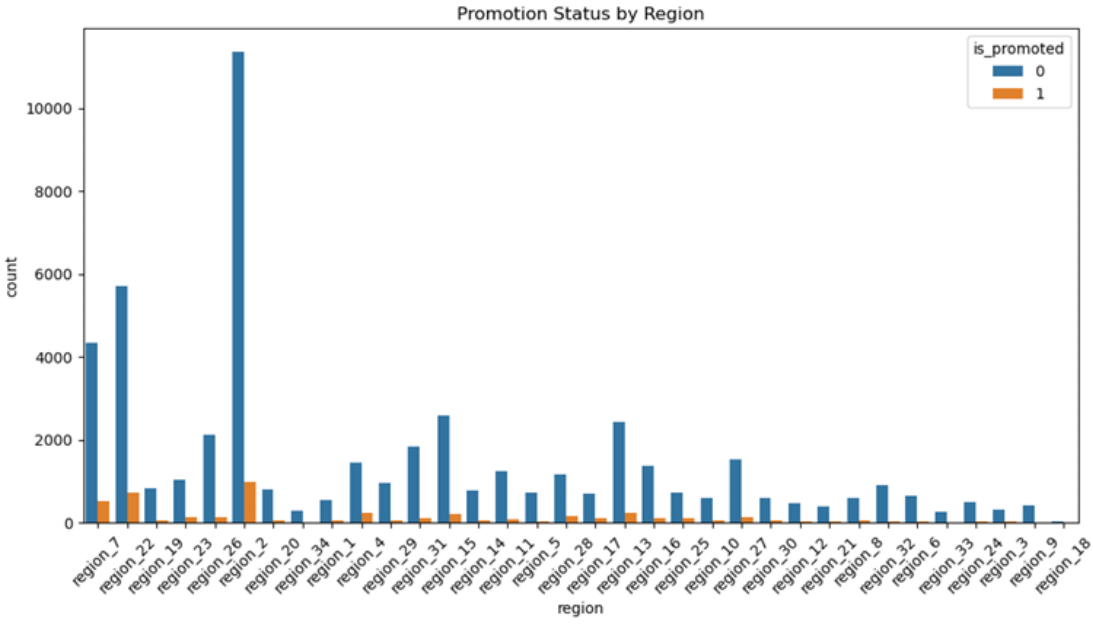
Boxplot of 'avg_training_score' vs. 'is_promoted':
Compares the distribution of average training scores for promoted and non-promoted employees.

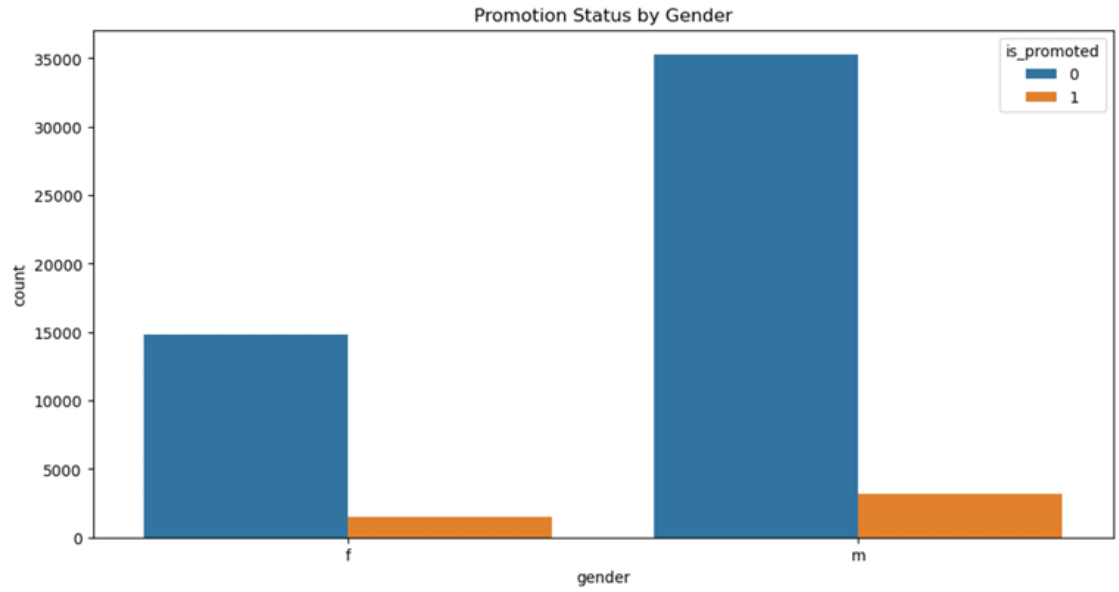
Boxplot of numerical variables:
Displays boxplots for all numerical variables in the dataset, providing an overview of their distributions and identifying potential outliers.

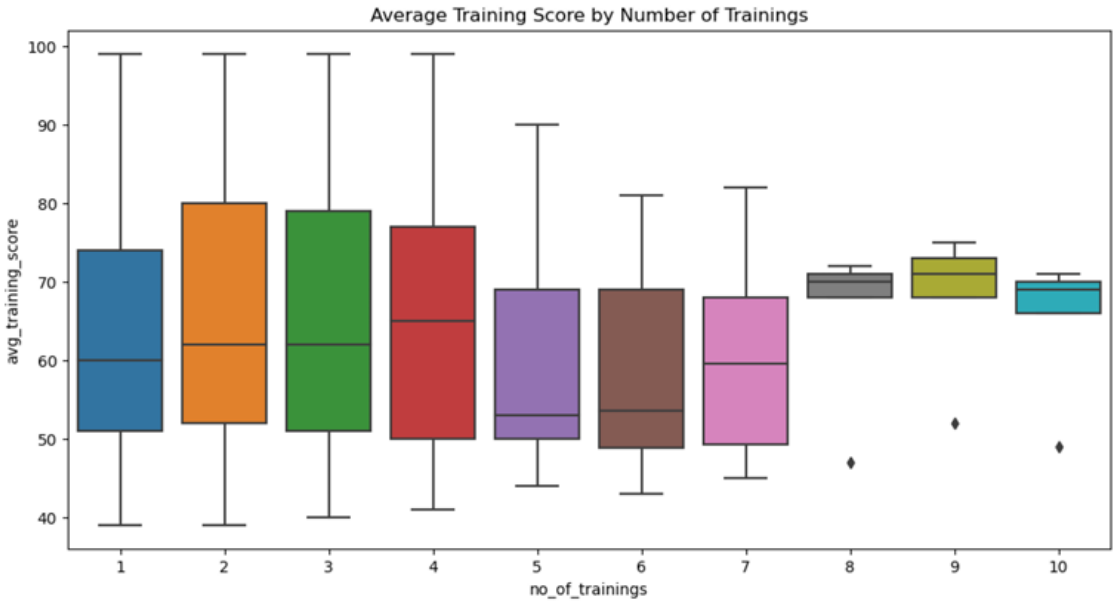

50000
40000
30000
20000
10000
0
count
0
1
is_promoted

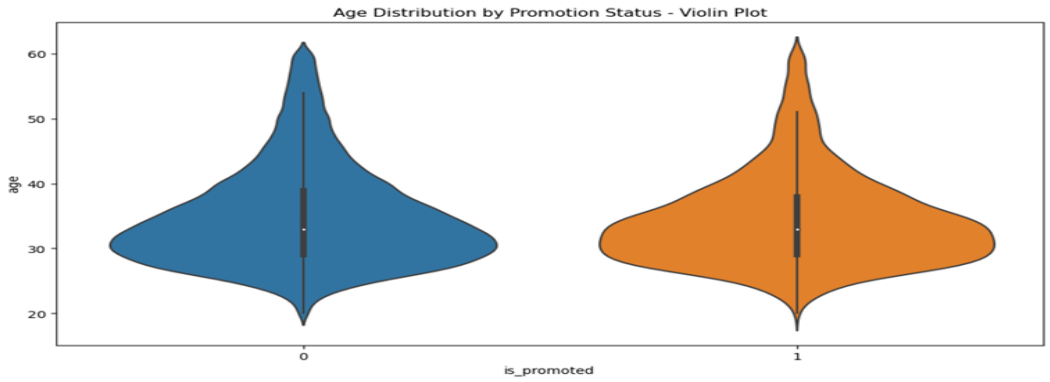

Promotion Status by Education
is_promoted
0
1
count
Master's & above
Bachelor's
Below Secondary
education

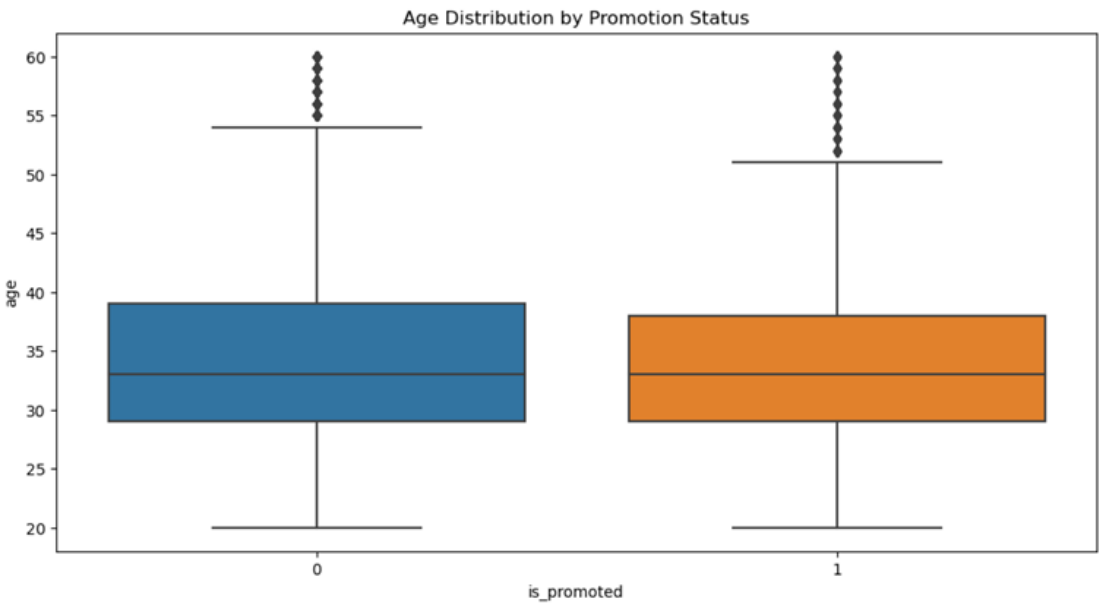

Promotion Status by Department
is_promoted
0
1
count
Sales & Marketing
Operations
Technology
Analytics
R&D
Procurement
Finance
HR
Legal
department

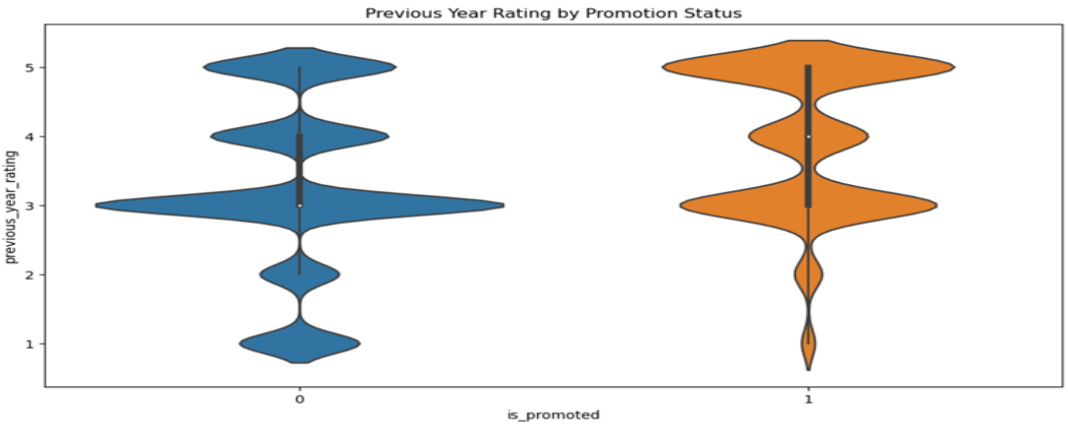

Promotion Status by Region
is_promoted
0
1
count
region_7
region_22
region_19
region_23
region_26
region_2
region_20
region_34
region_1
region_4
region_29
region_31
region_15
region_14
region_11
region_5
region_28
region_17
region_13
region_16
region_25
region_10
region_27
region_30
region_12
region_21
region_8
region_32
region_6
region_33
region_24
region_3
region_9
region_18
region

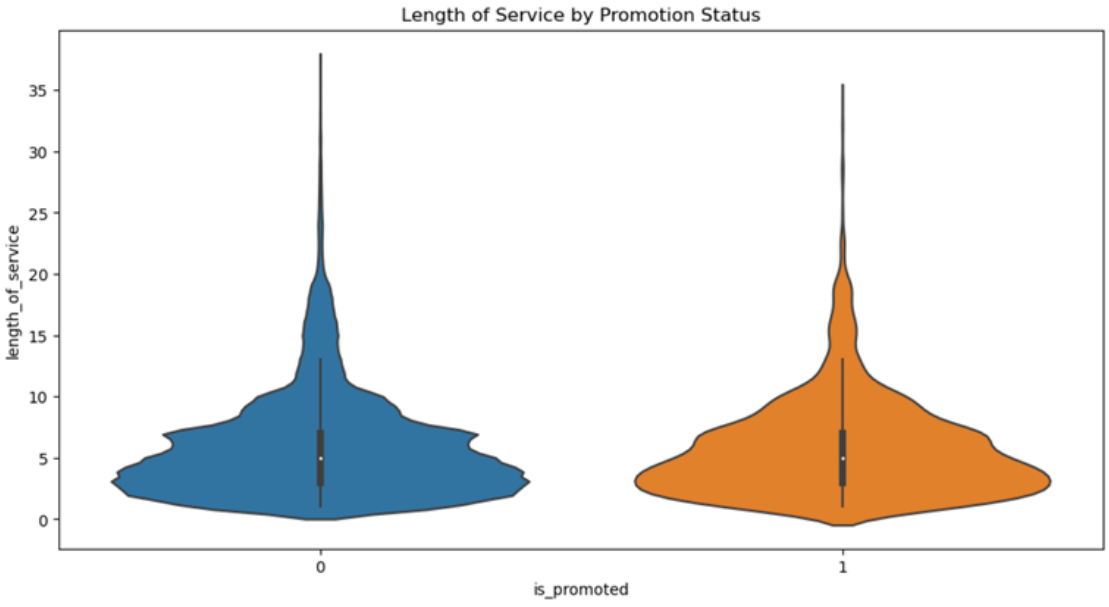

Promotion Status by Gender
is_promoted
0
1
count
35000
30000
25000
20000
15000
10000
5000
0
f
m
gender

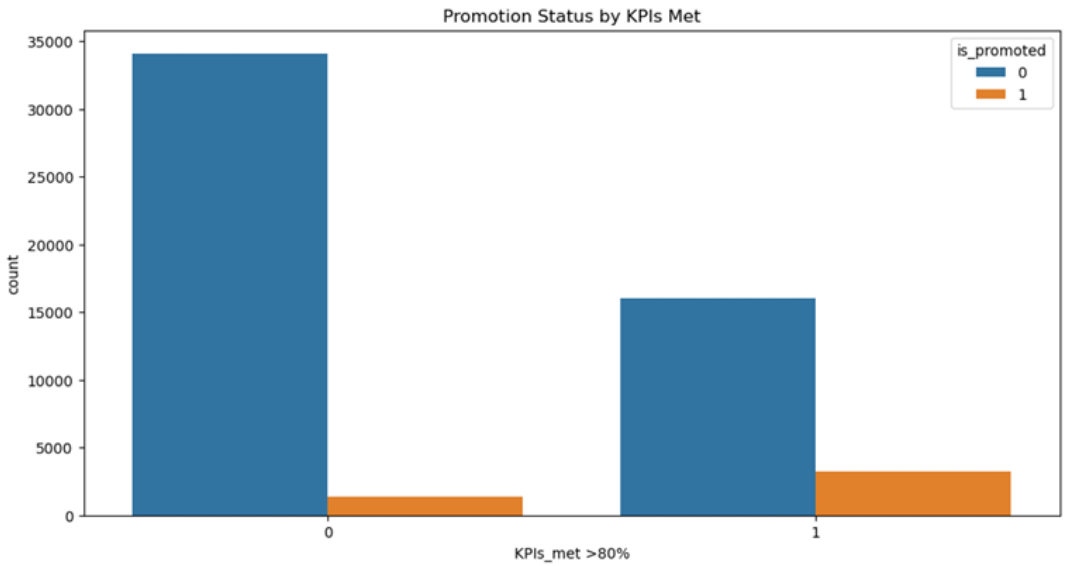

Average Training Score by Number of Trainings
avg_training_score
100
90
80
70
60
50
40
1
2
3
4
5
6
7
8
9
10
no_of_trainings

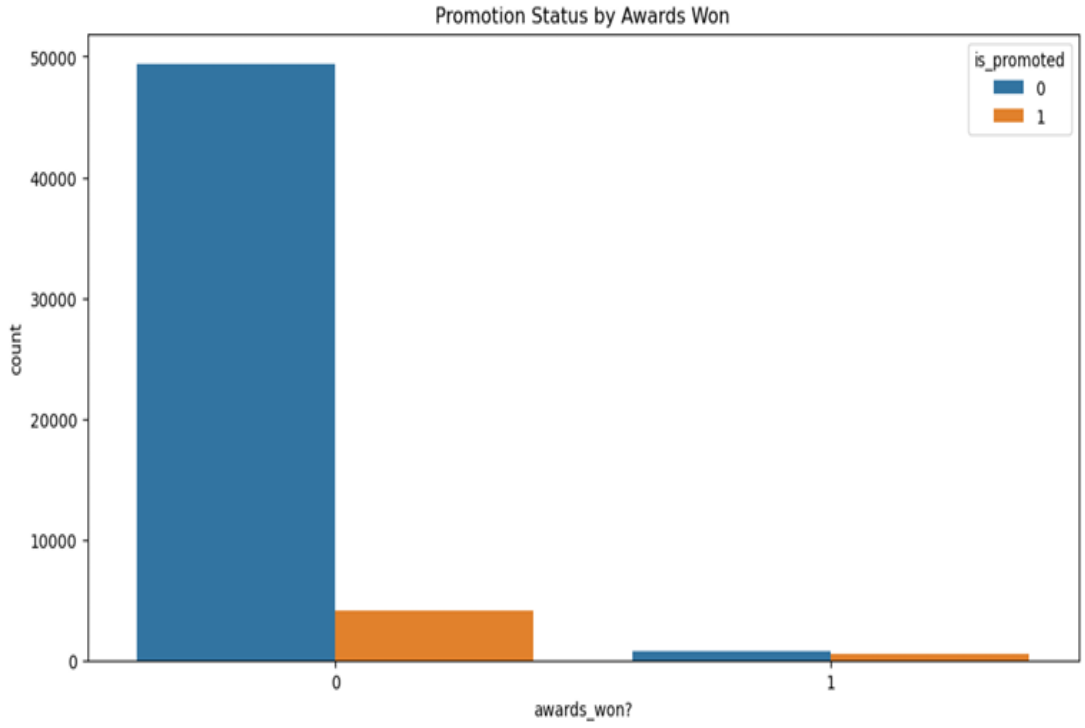

Age Distribution by Promotion Status - Violin Plot
age
60
50
40
30
20
0
1
is_promoted

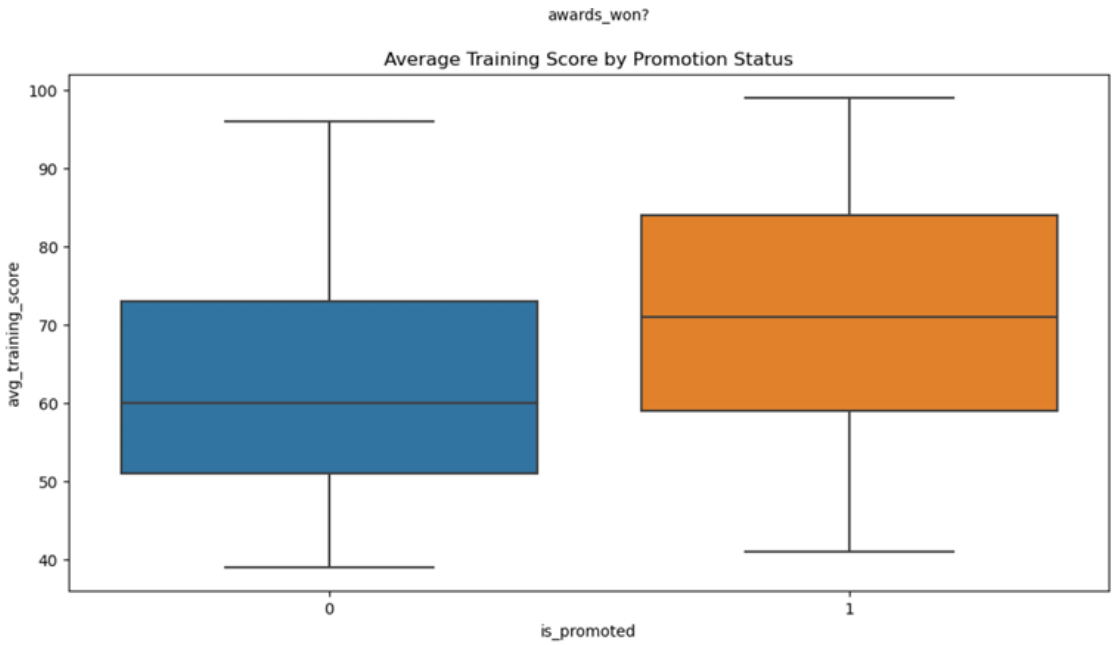

Age Distribution by Promotion Status
age
60
55
50
45
40
35
30
25
20
0
1
is_promoted


Previous Year Rating by Promotion Status
previous_year_rating
5
4
3
2
1
0
1
is_promoted


Length of Service by Promotion Status
length_of_service
35
30
25
20
15
10
5
0
0
1
is_promoted


Promotion Status by KPIs Met
is_promoted
0
1
35000
30000
25000
20000
15000
10000
5000
0
count
0
1
KPIs_met >80%


Promotion Status by Awards Won
is_promoted
0
1
50000
40000
30000
20000
10000
0
count
0
1
awards_won?



Insightful Observations: The visualizations facilitated the identification of patterns and trends, such as the impact of different departments, regions, and educational backgrounds on promotion outcomes.

Countplot of 'is_promoted' shows only **5% (2500)** of employees were promoted among **50000.**

Sales & marketing, operations, procurement, technology, and analytics are the top 5 departments to which the promoted employees belong.

- If the education of the employee is Bachelors or Masters&above, then the chances of promotion are high.
- Male and Female ratio in the company is 2:1 and their promotion ratio is also in the ratio 2:1
- If the employee is hired through Sourcing or other then the chance of promotion is hiring than that for referred people.
- Number of trainings has an inverse relation with promotion.
- Most of the employees between the age group of 25 to 45 have higher chances of promotion.
- If the training score of the employee is at least 70, then the chances of promotion are high.
- If the employee rating for the previous year is more than 3, then the chances of promotion are high.
- If the employee who's kpi's is more than 80% chances of promotion is more.

Data Transformation

The DataTransformation class focused on converting the data into a format suitable for machine learning models:

Categorical to Numeric Conversion: Categorical variables were transformed into numeric formats using LabelEncoder, ensuring they could be effectively utilized by the algorithms.

Conversion of Categorical Data to Numeric

The first step in the data transformation process involved converting categorical variables into numeric formats. This was achieved using the LabelEncoder from the sklearn.preprocessing module. The categorical columns identified for this transformation included 'department', 'region', 'education', 'gender', and 'recruitment channel'.

Label Encoding: Each categorical variable was encoded into numeric labels, ensuring that the machine learning algorithms could process them effectively. This step was crucial as many algorithms are designed to work with numerical input and may not handle categorical data appropriately.

```
def convert_to_numeric(self):
    """
    Convert categorical data to numeric using LabelEncoder.
    """
    label_encoder = LabelEncoder()
    categorical_columns = ['department', 'region', 'education', 'gender', 'recruitment_channel']
    for col in categorical_columns:
        self.train_df[col] = label_encoder.fit_transform(self.train_df[col])
        self.test_df[col] = label_encoder.transform(self.test_df[col])
    print("Categorical data converted to numeric.")
    print()
```

Normalization of Numerical Data

Following the encoding of categorical variables, the class focused on normalizing the numerical data columns.

Normalization Technique: The normalization involved scaling the numerical features to a range between 0 and 1. This scaling was performed on all numerical columns except 'employee_id' and 'is_promoted', as these were not features but identifiers and target variables, respectively.

```
def normalize_data(self):
    """
    Normalize numerical data columns.
    """
        # Normalize each numeric column except 'employee_id' and 'is_promoted'
    def normalize_column(col):
        if pd.api.types.is_numeric_dtype(col) and not pd.api.types.is_bool_dtype(col):
            return (col - col.min()) / (col.max() - col.min())
        return col

    cols_to_normalize = [col for col in self.train_df.columns if col not in ['employee_id', 'is_promoted']]
    self.train_df[cols_to_normalize] = self.train_df[cols_to_normalize].apply(normalize_column)
    self.test_df[cols_to_normalize] = self.test_df[cols_to_normalize].apply(normalize_column)

    print(self.train_df.head())
    print()
    print(self.test_df.head())
    print()
    print("Data normalized.")

    self.correlation_analysis()
```

Normalization Impact: By normalizing the data, we mitigated the risk of certain features dominating others due to differences in scale, thereby enhancing the model's ability to learn from the data more effectively.

```
Categorical data converted to numeric.

   employee_id  department    region  education  gender  recruitment_channel  \
0        65438       0.875  0.939394   0.666667     0.0                  1.0
1        65141       0.500  0.424242   0.000000     1.0                  0.0
2         7513       0.875  0.303030   0.000000     1.0                  1.0
3         2542       0.875  0.454545   0.000000     1.0                  0.0
4        48945       1.000  0.545455   0.000000     1.0                  0.0

   no_of_trainings    age  previous_year_rating  length_of_service  \
0         0.000000  0.375                   1.0           0.194444
1         0.000000  0.250                   1.0           0.083333
2         0.000000  0.350                   0.5           0.166667
3         0.111111  0.475                   0.0           0.250000
4         0.000000  0.625                   0.5           0.027778

   KPIs_met >80%  awards_won?  avg_training_score  is_promoted
0            1.0          0.0            0.166667            0
1            0.0          0.0            0.350000            0
2            0.0          0.0            0.183333            0
3            0.0          0.0            0.183333            0
4            0.0          0.0            0.566667            0
```

```
   employee_id  department    region  education  gender  recruitment_channel  \
0         8724       1.000  0.545455        0.0     1.0                  1.0
1        74430       0.250  0.848485        0.0     0.0                  0.0
2        72255       0.875  0.121212        0.0     1.0                  0.0
3        38562       0.625  0.333333        0.0     0.0                  0.0
4        64486       0.125  0.636364        0.0     1.0                  1.0

   no_of_trainings    age  previous_year_rating  length_of_service  \
0             0.00  0.100                   NaN           0.000000
1             0.00  0.275                  0.50           0.121212
2             0.00  0.275                  0.00           0.090909
3             0.25  0.275                  0.25           0.242424
4             0.00  0.250                  0.75           0.181818

   KPIs_met >80%  awards_won?  avg_training_score
0            1.0          0.0            0.633333
1            0.0          0.0            0.200000
2            0.0          0.0            0.133333
3            0.0          0.0            0.433333
4            0.0          0.0            0.366667
```

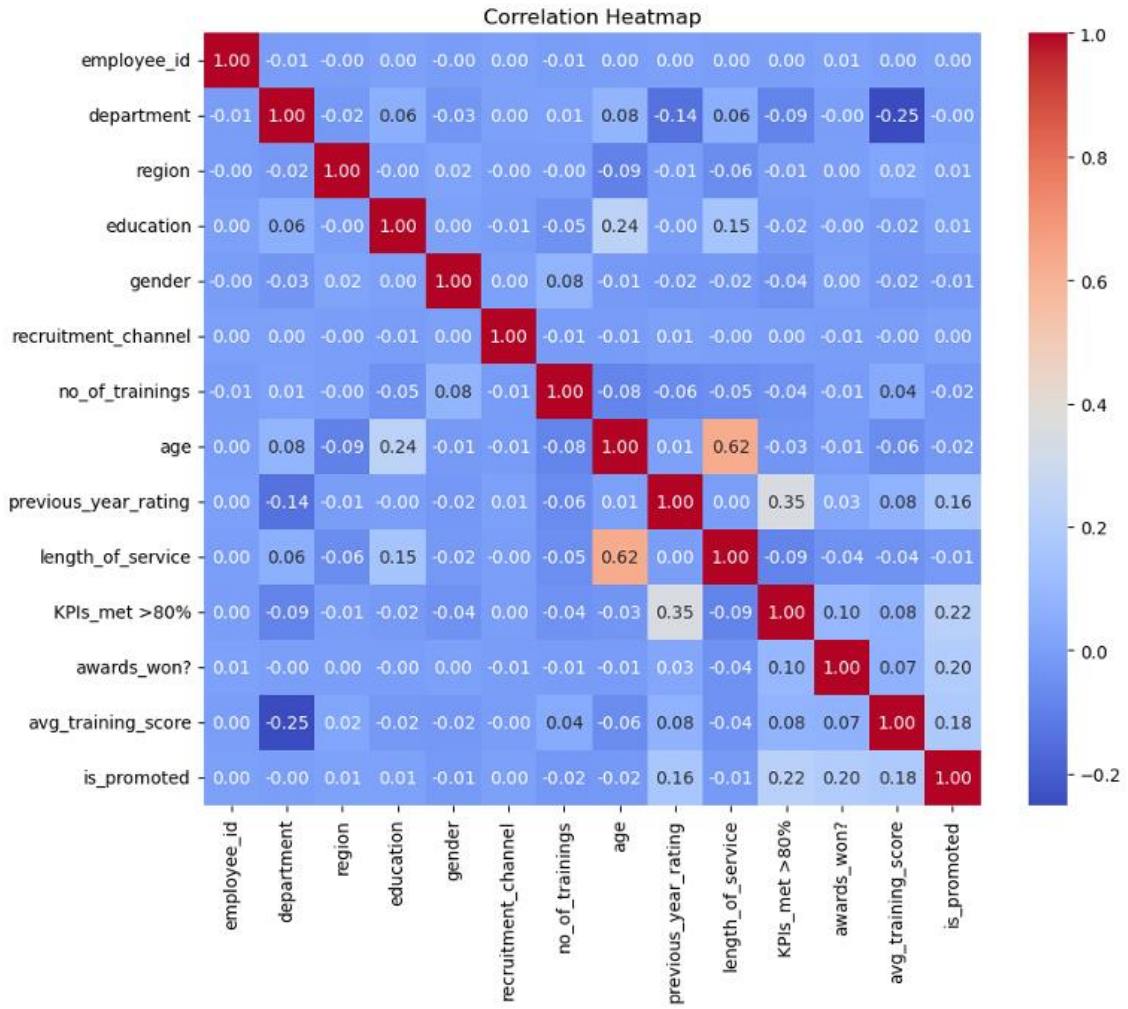
Above are train and test data frames after Normalization.

Correlation Analysis

The final step in the data transformation process was the correlation analysis.

```python
def correlation_analysis(self):
    """
    Perform correlation analysis and visualize the correlation heatmap.
    """
    # Select only numerical columns
    numerical_cols = self.train_df.select_dtypes(include=[np.number]).columns
    # Correlation heatmap
    correlation_matrix = self.train_df[numerical_cols].corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
    plt.title('Correlation Heatmap')
    plt.show()
```

Correlation Heatmap: A heatmap of the correlation matrix was generated for the numerical columns. This visualization provided a clear and concise representation of how different features were related to each other.



Correlation coefficients range from -1 to +1, where:

+1 indicates a perfect positive correlation (as one variable increases, the other variable increases).
-1 indicates a perfect negative correlation (as one variable increases, the other variable decreases).
0 indicates no correlation (the variables do not appear to affect each other).

Insights from Correlation Analysis:

previous_year_rating: This has a positive correlation of 0.16 with is_promoted, suggesting that employees with higher ratings in the previous year have a better chance of being promoted.
KPIs_met >80%: With a correlation of 0.22, this indicates a moderate positive relationship, suggesting employees who meet more than 80% of their KPIs have a higher chance of being promoted.
awards_won?: The correlation is 0.20, which is relatively significant, implying that employees who have won awards are more likely to be promoted.
avg_training_score: Shows a positive correlation of 0.18, indicating that higher training scores might be associated with higher promotion rates.

The correlation heatmap was instrumental in identifying features that were strongly correlated with each other. This helped in understanding the relationships within the data, which could influence feature selection and model interpretability.

Model Training and Evaluation

Implementing K-Nearest Neighbors (KNN)

Our study advanced into the model training phase with the implementation of the K-Nearest Neighbors (KNN) algorithm, encapsulated within the ModelTraining class. This class was designed to not only train the KNN model but also to rigorously evaluate its performance.

Data Preparation for Model Training

```
def split_data(self):
    """
    Split the data into training and validation sets.
    """
    # Handle missing values by removing rows with missing values
    self.train_df.dropna(inplace=True)

    X = self.train_df[['KPIs_met >80%', 'awards_won?', 'avg_training_score', 'previous_year_rating']]
    y = self.train_df['is_promoted']
    self.X_train, self.X_val, self.y_train, self.y_val = train_test_split(X, y, test_size=0.2, random_state=42)
    print("Data split into train and validation sets.")
```

Initially, the class handled missing values in the training dataset by removing rows with missing data, ensuring the quality and reliability of the data fed into the model. 'KPIs_met >80%', 'Awards_won?', 'Avg_training_score', 'Previous_year_rating' - These features were selected based on their observed correlations and perceived relevance to the target variable 'is_promoted', as revealed by the heatmap. With the features identified, we proceeded to split the dataset into training and validation sets. This was achieved using the train_test_split function from the sklearn.model_selection module. We allocated 80% of the data for training and 20% for validation, a common split ratio that balances the need for sufficient training data with the necessity of an unbiased evaluation set. The random_state parameter was set to 42 to ensure reproducibility of our results.

Training the KNN Classifier

```
def knn_classifier(self):
    """
    Train a KNN classifier.
    """
    self.knn = KNeighborsClassifier(n_neighbors=10)
    self.knn.fit(self.X_train, self.y_train)
    print("KNN model trained.")
```

The KNN classifier was trained using the selected features from our dataset. We instantiated the classifier with n_neighbors=5, indicating that the model should consider the five nearest neighbors in the feature space for making predictions. This choice was based on preliminary testing, where we observed that a K-value of 5 provided a balance between bias and variance, leading to more accurate and generalizable results.

Justification for k value:

K=5 offers a balanced approach, with good accuracy and a moderate trade-off between precision and recall.

K=3 increases sensitivity to actual promotions but at the cost of making more incorrect promotion predictions.

K=10 is the most cautious, with high accuracy and precision but at the risk of missing many actual promotions.

Model Fitting: The classifier was fitted to our training data, a process that involved mapping the feature space and preparing the model to make predictions based on the proximity of data points.

Model Evaluation Metrics

```
def evaluate_model(self):
    """
    Evaluate the KNN model and print evaluation metrics.
    """
    y_pred = self.knn.predict(self.X_val)
    accuracy = accuracy_score(self.y_val, y_pred)
    precision = precision_score(self.y_val, y_pred)
    recall = recall_score(self.y_val, y_pred)
    f1 = f1_score(self.y_val, y_pred)
    print(f"Accuracy: {accuracy*100}, Precision: {precision}, Recall: {recall}, F1 Score: {f1}")
```

Post-training, the model's performance was evaluated using a suite of metrics:

Accuracy: This metric provided an overall success rate of the model in correctly predicting promotions.

Precision and Recall: Precision measured the model's accuracy in predicting positive instances, while recall assessed its ability to identify all actual positive instances.

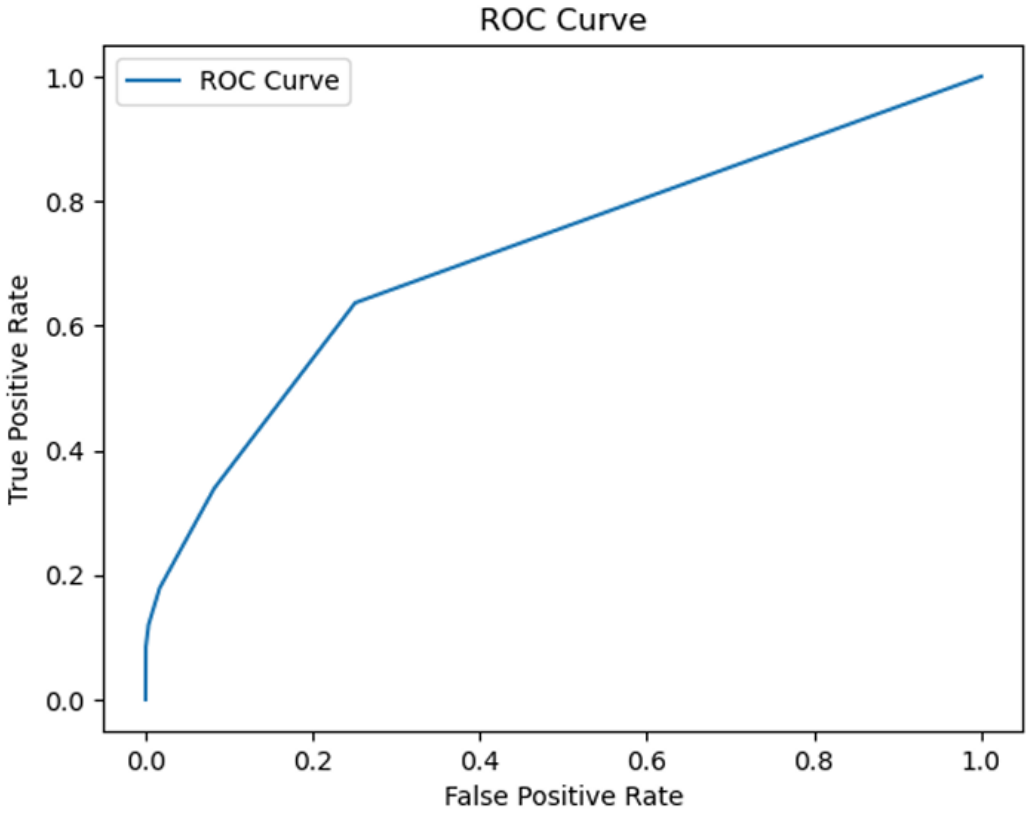
F1 Score: The F1 score, a harmonic mean of precision and recall, offered a single metric to gauge the model's balance between precision and recall.

```
Data split into train and validation sets.
KNN model trained.
Accuracy: 91.17095787708395, Precision: 0.5078864353312302, Recall: 0.17888888888888
888, F1 Score: 0.2645850451930978
```

ROC Curve Analysis

```
def plot_roc_curve(self):
    probabilities = self.knn.predict_proba(self.X_val)[:, 1]
    fpr, tpr, _ = roc_curve(self.y_val, probabilities)
    plt.figure()
    plt.plot(fpr, tpr, label='ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend()
    plt.show()
```

To further assess the model's diagnostic ability, we plotted a Receiver Operating Characteristic (ROC) curve. This curve illustrated the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) at various threshold settings, providing a comprehensive view of the model's performance.



Justification for Choosing KNN

KNN was selected for its non-parametric nature and its ability to make predictions based on the local structure of the data. This made it particularly suitable for our dataset, which exhibited patterns that could be effectively captured through KNN's instance-based learning approach. Additionally, KNN's simplicity and interpretability aligned well with our objective of creating a transparent and understandable model.

As per paper Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants by Shahadat Uddin: Classic KNN has Low time complexity and can classify at high speeds compared to other machine learning algorithms.

The ModelApplication class is designed to facilitate the application of a pre-trained KNN model to a test dataset. It consists of the following key components:

Initialization: The class is initialized with the trained KNN model and the test dataset. The KNN model is assumed to have been previously trained on a labeled training dataset.

Data Preprocessing: The preprocess_test_data method is responsible for preparing the test data for prediction. It handles missing values by removing rows with missing values and selects the same features as used during the training of the KNN model.

```python
class ModelApplication:
    def __init__(self, trained_model, test_df):
        """
        Initialize with the trained model and test dataframe.

        Args:
            trained_model (ModelTraining): The trained KNN model.
            test_df (pd.DataFrame): Test data without target labels.
        """
        self.trained_model = trained_model
        self.test_df = test_df
        self.predictions = None

    def preprocess_test_data(self):
        """
        Preprocess the test data (similar to how training data was preprocessed).--
        """
        # Handle missing values by removing rows with missing values
        self.test_df.dropna(inplace=True)

        # Selecting the same features as used in the training model
        self.test_features = self.test_df[['KPIs_met >80%', 'awards_won?', 'avg_training_score', 'previous_year_rating']]

    def make_predictions(self):
        """
        Make predictions on the test data using the trained model.
        """
        self.predictions = self.trained_model.knn.predict(self.test_features)
        print("Predictions made on the test data.")
        print(self.predictions)
        self.test_df['predictions'] = self.predictions
        print(self.test_df.describe())
        print(self.test_df.head(30))

# Usage
try:
    model_app = ModelApplication(model_train, data_transform.test_df)  # Replace 'your_initial_test_data' with your actual test
    model_app.preprocess_test_data()
    model_app.make_predictions()

    print(model_app)
except Exception as e:
    print(f"An error occurred: {e}")
```

The make_predictions method applies the trained KNN model to the preprocessed test data, generating predictions for each data point. The predictions are stored in the predictions attribute of the class instance.

```
       KPIs_met >80%  awards_won?  avg_training_score  predictions
0               1.0          0.0            0.633333            0
1               0.0          0.0            0.200000            0
2               0.0          0.0            0.133333            0
3               0.0          0.0            0.433333            0
4               0.0          0.0            0.366667            0
5               0.0          0.0            0.483333            0
```

```
6               1.0          0.0            0.300000            0
7               0.0          0.0            0.766667            0
8               0.0          0.0            0.600000            0
9               1.0          0.0            0.616667            0
10              1.0          0.0            0.183333            0
11              0.0          0.0            0.116667            0
12              0.0          0.0            0.216667            0
13              0.0          0.0            0.716667            0
14              1.0          0.0            0.316667            0
15              1.0          0.0            0.133333            1
16              0.0          0.0            0.283333            0
17              1.0          0.0            0.183333            0
18              1.0          0.0            0.416667            0
19              0.0          0.0            0.433333            0
20              0.0          0.0            0.133333            0
21              0.0          0.0            0.683333            0
22              0.0          0.0            0.733333            0
23              0.0          0.0            0.616667            0
24              1.0          0.0            0.316667            0
25              1.0          1.0            0.383333            0
26              1.0          0.0            0.183333            0
27              1.0          0.0            0.800000            1
28              0.0          0.0            0.366667            0
29              0.0          0.0            0.183333            0
<__main__.ModelApplication object at 0x00000213B4BF4AD0>
```

Results: Upon executing the make_predictions method, the class provides a summary of the predictions, including statistical descriptions of the predicted values and the first 30 rows of the test dataset with corresponding predictions. This information is crucial for evaluating the model's performance on the test data and the predictions indicate whether an employee is promote or not.

Conclusion:

In conclusion, this study demonstrates a comprehensive application of the K-Nearest Neighbors (KNN) algorithm for predicting employee promotions within an HR dataset. The project was underpinned by a meticulous data preparation process, which included handling missing values, removing duplicates, and normalizing data to ensure the reliability and accuracy of the machine learning model. Key features were carefully selected based on their correlation with the target variable 'is_promoted', ensuring a data-driven approach to model training.

The KNN algorithm, known for its simplicity and effectiveness in pattern recognition, was aptly chosen for its ability to classify based on the local structure of the data. The model was trained with a balanced approach, considering precision, recall, and overall accuracy. The evaluation of the model included accuracy metrics, ROC curve analysis, and the F1 score, providing a holistic view of its performance.

The ModelApplication class streamlined the application of the trained model to the test dataset, demonstrating the model's practical utility in a real-world scenario. The results of the model provided insightful predictions about potential candidates for promotion, thereby assisting HR decisions.

References:

[1]. Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants by Shahadat Uddin, Ibtisham Haque, Haohui Lu, Mohammad Ali Moni & Ergun Gide comparative analysis

[2]. A Brief Review of Nearest Neighbor Algorithm for Learning and Classification Kashvi Taunk; Sanjukta De; Srishti Verma; Aleena Swetapadma Published in 2019 International Conference on Intelligent Computing and Control Systems (ICCS).