Activity_ Course 7 Salifort Motors project lab

May 17, 2023

1 Capstone project: Providing data-driven suggestions for HR by HASEEB RAZA

1.1 Description and deliverables

This capstone project is an opportunity for you to analyze a dataset and build predictive models that can provide insights to the Human Resources (HR) department of a large consulting firm.

Upon completion, you will have two artifacts that you would be able to present to future employers. One is a brief one-page summary of this project that you would present to external stakeholders as the data professional in Salifort Motors. The other is a complete code notebook provided here. Please consider your prior course work and select one way to achieve this given project question. Either use a regression model or machine learning model to predict whether or not an employee will leave the company. The exemplar following this actiivty shows both approaches, but you only need to do one.

In your deliverables, you will include the model evaluation (and interpretation if applicable), a data visualization(s) of your choice that is directly related to the question you ask, ethical considerations, and the resources you used to troubleshoot and find answers or solutions.

PACE stages

- Section **??**
- Section **??**
- Section **??**
- Section **??**

2 Pace: Plan Stage

- Understand your data in the problem context
- Consider how your data will best address the business need
- Contextualize & understand the data and the problem

Understand the business scenario and problem

The HR department at Salifort Motors wants to take some initiatives to improve employee satisfaction levels at the company. They collected data from employees, but now they don't know what to do with it. They refer to you as a data analytics professional and ask you to provide data-driven suggestions based on your understanding of the data. They have the following question: what's likely to make the employee leave the company?

Your goals in this project are to analyze the data collected by the HR department and to build a model that predicts whether or not an employee will leave the company.

If you can predict employees likely to quit, it might be possible to identify factors that contribute to their leaving. Because it is time-consuming and expensive to find, interview, and hire new employees, increasing employee retention will be beneficial to the company.

2.0.1 Familiarize yourself with the HR dataset

The dataset that you'll be using in this lab contains 15,000 rows and 10 columns for the variables listed below.

Note: you don't need to download any data to complete this lab. For more information about the data, refer to its source on Kaggle.

Variable	Description
satisfaction_level	Employee-reported job satisfaction level [0–1]
last_evaluation	Score of employee's last performance review [0–1]
number_project	Number of projects employee contributes to
average_monthly_hours	Average number of hours employee worked per month
time_spend_company	How long the employee has been with the company (years)
Work_accident	Whether or not the employee experienced an accident while at work
left	Whether or not the employee left the company
promotion_last_5years	Whether or not the employee was promoted in the last 5 years
Department	The employee's department
salary	The employee's salary (U.S. dollars)

Reflect on these questions as you complete the plan stage.

- Who are your stakeholders for this project?
- What are you trying to solve or accomplish?
- What are your initial observations when you explore the data?
- What resources do you find yourself using as you complete this stage? (Make sure to include the links.)

- Do you have any ethical considerations in this stage?

[Double-click to enter your responses here.]

2.1 Step 1. Imports

- Import packages
- Load dataset

2.1.1 Import packages

[1]:
```
# Import packages for data manipulation
import numpy as np
import pandas as pd

# Import packages for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Set options for displaying all columns in dataframes
pd.set_option('display.max_columns', None)

# Import packages for data modeling
from xgboost import XGBClassifier, XGBRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier

# Import packages for metrics and helpful functions
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, ConfusionMatrixDisplay, classification_report,
    roc_auc_score, roc_curve
)

# Import package for saving models
import pickle
```

2.1.2 Load dataset

`Pandas` is used to read a dataset called **`HR_capstone_dataset.csv`.** As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

[2]:
```
# RUN THIS CELL TO IMPORT YOUR DATA.

# Load dataset into a dataframe
### YOUR CODE HERE ###
df0 = pd.read_csv("HR_capstone_dataset.csv")


# Display first few rows of the dataframe
### YOUR CODE HERE ###
# Display first few rows of the dataframe
df0.head()
```

[2]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Department	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low

2.2 Step 2. Data Exploration (Initial EDA and data cleaning)

- Understand your variables
- Clean your dataset (missing data, redundant data, outliers)

2.2.1 Gather basic information about the data

[3]:
```
# Gather basic information about the data
### YOUR CODE HERE ###
# Gather basic information about the data
df0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   Department             14999 non-null  object
 9   salary                 14999 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

2.2.2 Gather descriptive statistics about the data

[4]:
```
# Gather descriptive statistics about the data
### YOUR CODE HERE ###
# Gather descriptive statistics about the data
df0.describe()
```

[4]:

	satisfaction_level	last_evaluation	number_project	\
count	14999.000000	14999.000000	14999.000000	
mean	0.612834	0.716102	3.803054	
std	0.248631	0.171169	1.232592	
min	0.090000	0.360000	2.000000	
25%	0.440000	0.560000	3.000000	
50%	0.640000	0.720000	4.000000	
75%	0.820000	0.870000	5.000000	
max	1.000000	1.000000	7.000000	

	average_montly_hours	time_spend_company	Work_accident	left	\
count	14999.000000	14999.000000	14999.000000	14999.000000	
mean	201.050337	3.498233	0.144610	0.238083	
std	49.943099	1.460136	0.351719	0.425924	
min	96.000000	2.000000	0.000000	0.000000	
25%	156.000000	3.000000	0.000000	0.000000	
50%	200.000000	3.000000	0.000000	0.000000	
75%	245.000000	4.000000	0.000000	0.000000	
max	310.000000	10.000000	1.000000	1.000000	

	promotion_last_5years

```
count           14999.000000
mean                0.021268
std                 0.144281
min                 0.000000
25%                 0.000000
50%                 0.000000
75%                 0.000000
max                 1.000000
```

2.2.3 Rename columns

As a data cleaning step, rename the columns as needed. Standardize the column names so that they are all in `snake_case`, correct any column names that are misspelled, and make column names more concise as needed.

[5]:
```
# Display all column names
### YOUR CODE HERE ###
# Display all column names
df0.columns
```

[5]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
       'promotion_last_5years', 'Department', 'salary'],
      dtype='object')
```

[6]:
```
# Rename columns as needed
df0 = df0.rename(columns={
    'Work_accident': 'work_accident',
    'average_montly_hours': 'average_monthly_hours',
    'time_spend_company': 'tenure',
    'Department': 'department'
})

# Display all column names after the update
df0.columns
```

[6]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_monthly_hours', 'tenure', 'work_accident', 'left',
       'promotion_last_5years', 'department', 'salary'],
      dtype='object')
```

2.2.4 Check missing values

Check for any missing values in the data.

[7]:
```
# Check for missing values
### YOUR CODE HERE ###
# Check for missing values
df0.isnull().sum()
```

[7]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_monthly_hours    0
tenure                   0
work_accident            0
left                     0
promotion_last_5years    0
department               0
salary                   0
dtype: int64
```

2.2.5 Check duplicates

Check for any duplicate entries in the data.

[8]:
```
# Check for duplicates
### YOUR CODE HERE ###
# Check for duplicates
# Check for duplicates
df0.duplicated().sum()
```

[8]:
```
3008
```

[9]:
```
# Inspect some rows containing duplicates as needed
### YOUR CODE HERE ###
# Inspect rows containing duplicates
# Inspect some rows containing duplicates
duplicate_rows = df0[df0.duplicated()]
duplicate_rows.head()
```

[9]:

	satisfaction_level	last_evaluation	number_project	\
396	0.46	0.57	2	
866	0.41	0.46	2	
1317	0.37	0.51	2	
1368	0.41	0.52	2	
1461	0.42	0.53	2	

	average_monthly_hours	tenure	work_accident	left	\
396	139	3	0	1	
866	128	3	0	1	

```
1317                   127       3              0     1
1368                   132       3              0     1
1461                   142       3              0     1

      promotion_last_5years  department  salary
396                       0       sales     low
866                       0  accounting     low
1317                      0       sales  medium
1368                      0       RandD     low
1461                      0       sales     low
```

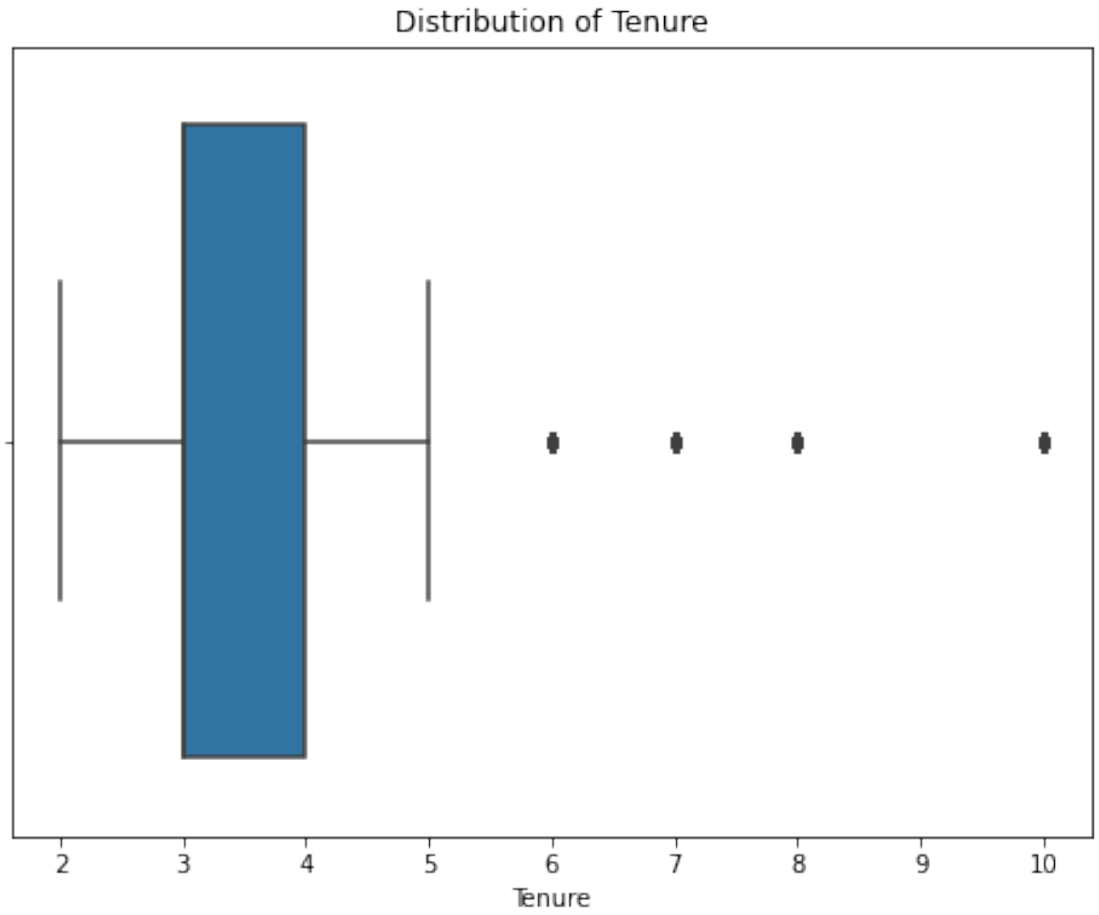
[10]:
```
# Drop duplicates and save resulting dataframe in a new variable
df1 = df0.drop_duplicates(keep='first')

# Display the first few rows of the new dataframe
df1.head()
```

[10]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
4                0.37             0.52               2                    159

   tenure  work_accident  left  promotion_last_5years department  salary
0       3              0     1                      0      sales     low
1       6              0     1                      0      sales  medium
2       4              0     1                      0      sales  medium
3       5              0     1                      0      sales     low
4       3              0     1                      0      sales     low
```

2.2.6 Check outliers

Check for outliers in the data.

[11]:
```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
### YOUR CODE HERE ###
# Create a boxplot to visualize the distribution of `tenure` and detect any␣
 ↪outliers
plt.figure(figsize=(8, 6))
sns.boxplot(data=df1, x='tenure')
plt.xlabel('Tenure')
plt.title('Distribution of Tenure')
plt.show()
```



[12]:
```
# Compute the 25th percentile value in `tenure`
percentile25 = df1['tenure'].quantile(0.25)

# Compute the 75th percentile value in `tenure`
percentile75 = df1['tenure'].quantile(0.75)

# Compute the interquartile range in `tenure`
iqr = percentile75 - percentile25

# Define the upper limit and lower limit for non-outlier values in `tenure`
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr

print("Lower limit:", lower_limit)
print("Upper limit:", upper_limit)

# Identify subset of data containing outliers in `tenure`
outliers = df1[(df1['tenure'] > upper_limit) | (df1['tenure'] < lower_limit)]
```

```
# Count how many rows in the data contain outliers in `tenure`
num_outliers = len(outliers)
print("Number of rows in the data containing outliers in `tenure`:",␣
↪num_outliers)
```

```
Lower limit: 1.5
Upper limit: 5.5
Number of rows in the data containing outliers in `tenure`: 824
```

Certain types of models are more sensitive to outliers than others. When you get to the stage of building your model, consider whether to remove outliers, based on the type of model you decide to use.

pAce: Analyze Stage - Perform EDA (analyze relationships between variables)

Reflect on these questions as you complete the analyze stage.

- What did you observe about the relationships between variables? You will need to perform EDA to analyze the relationships between variables in your dataset. This can involve visualizations such as scatter plots, correlation matrices, and statistical tests to identify any significant relationships or patterns.
- What do you observe about the distributions in the data? EDA allows you to examine the distributions of variables and identify any skewness, outliers, or unusual patterns. Histograms, boxplots, and density plots are common visualizations used to analyze the distributions.
- What transformations did you make with your data? Why did you chose to make those decisions? Encoding categorical variables:

The 'Department' and 'Salary' columns are categorical variables. You might choose to encode them using one-hot encoding or label encoding techniques to convert them into numerical representations that can be used by the model. Handling imbalanced classes:

Since the target variable 'left' indicates whether an employee left the company or not, you might observe class imbalance, where the number of employees who left is significantly lower than those who stayed. In this case, you may need to address the imbalance through techniques such as oversampling, undersampling, or using specialized algorithms designed for imbalanced data. Scaling numerical variables:

Scaling numerical variables can be beneficial for certain models that are sensitive to the scale of the features. For example, models like logistic regression or K-nearest neighbors might benefit from feature scaling to ensure that variables with larger ranges don't dominate the model's calculations. - What are some purposes of EDA before constructing a predictive model? EDA serves several purposes before constructing a predictive model: Identifying patterns and relationships in the data Understanding the distributions and characteristics of variables Detecting outliers and handling missing values Assessing the need for data preprocessing and transformations Selecting relevant features for the model Informing the selection of an appropriate modeling technique - What resources do you find yourself using as you complete this stage? (Make sure to include the links.) - Do you have any ethical considerations in this stage? Ethical considerations in the analyze stage may involve ensuring data privacy, handling sensitive information appropriately, and maintaining data security. It's important to follow ethical guidelines and best practices when working with data to protect individuals' privacy and maintain data integrity.

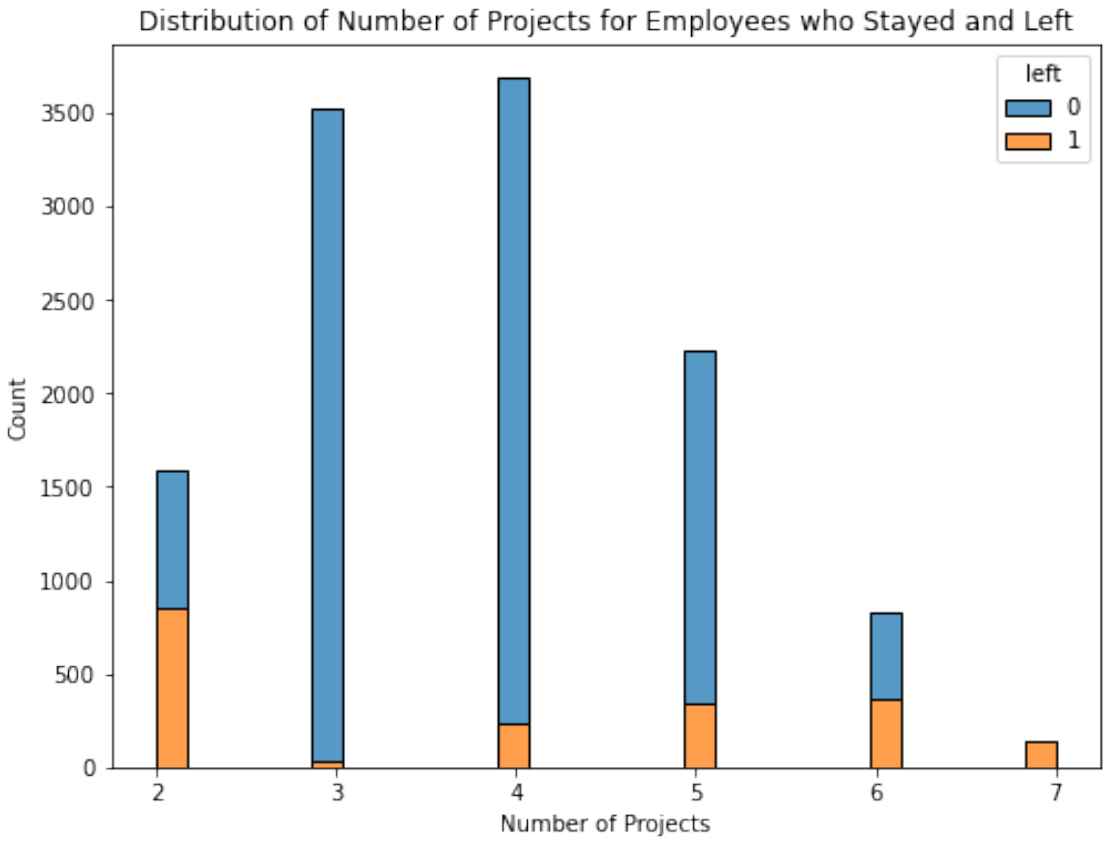
[Double-click to enter your responses here.]

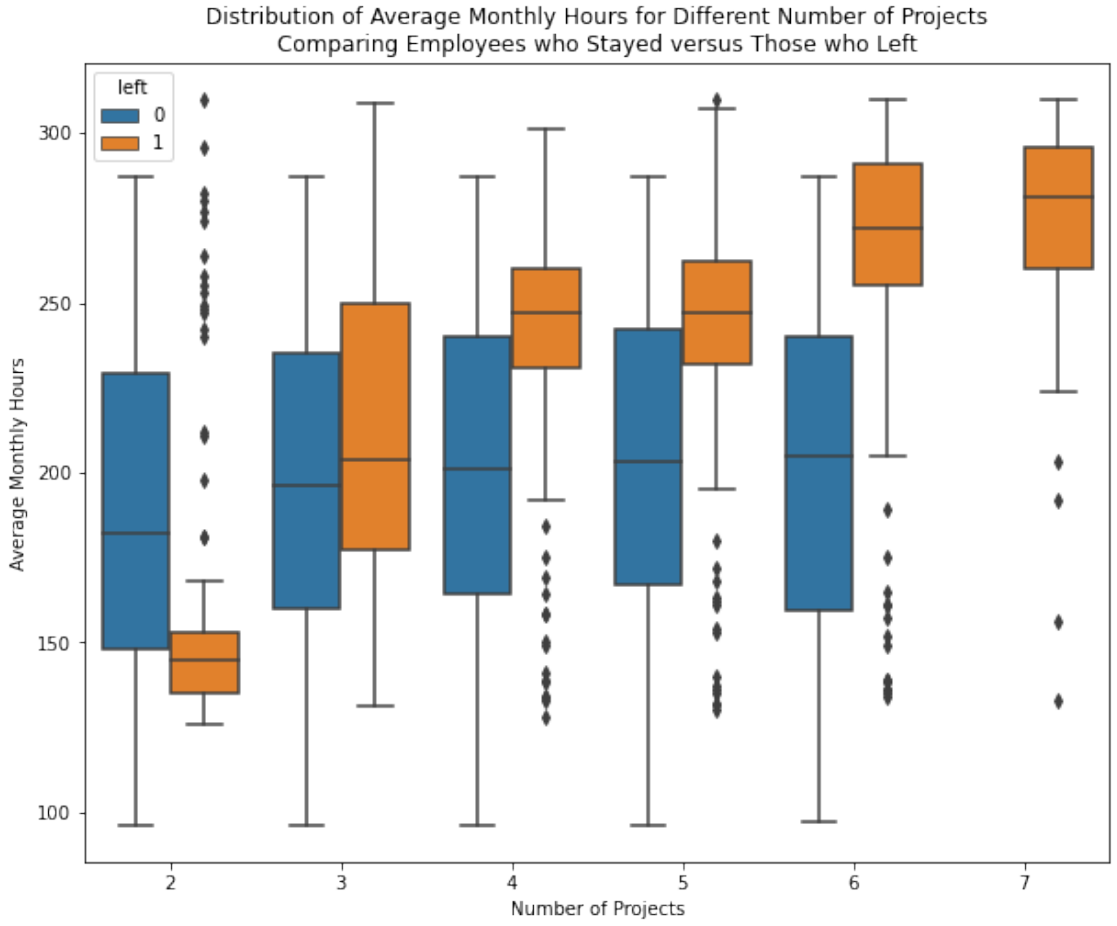
2.3 Step 2. Data Exploration (Continue EDA)

2.3.1 Data visualizations

Now, start examining the variables that you're interested in, and create plots to visualize relationships between variables in the data.

[13]:
```
# Create a plot as needed
# Create a stacked histogram to visualize the distribution of `number_project`␣
 ↪for employees who stayed and left
plt.figure(figsize=(8, 6))
sns.histplot(data=df1, x='number_project', hue='left', multiple='stack')
plt.xlabel('Number of Projects')
plt.ylabel('Count')
plt.title('Distribution of Number of Projects for Employees who Stayed and␣
 ↪Left')
plt.show()
# Create a boxplot to compare the distributions of `average_monthly_hours` for␣
 ↪different `number_project` groups and employees who stayed versus those who␣
 ↪left
plt.figure(figsize=(10, 8))
sns.boxplot(data=df1, x='number_project', y='average_monthly_hours', hue='left')
plt.xlabel('Number of Projects')
plt.ylabel('Average Monthly Hours')
plt.title('Distribution of Average Monthly Hours for Different Number of␣
 ↪Projects\nComparing Employees who Stayed versus Those who Left')
plt.show()
```

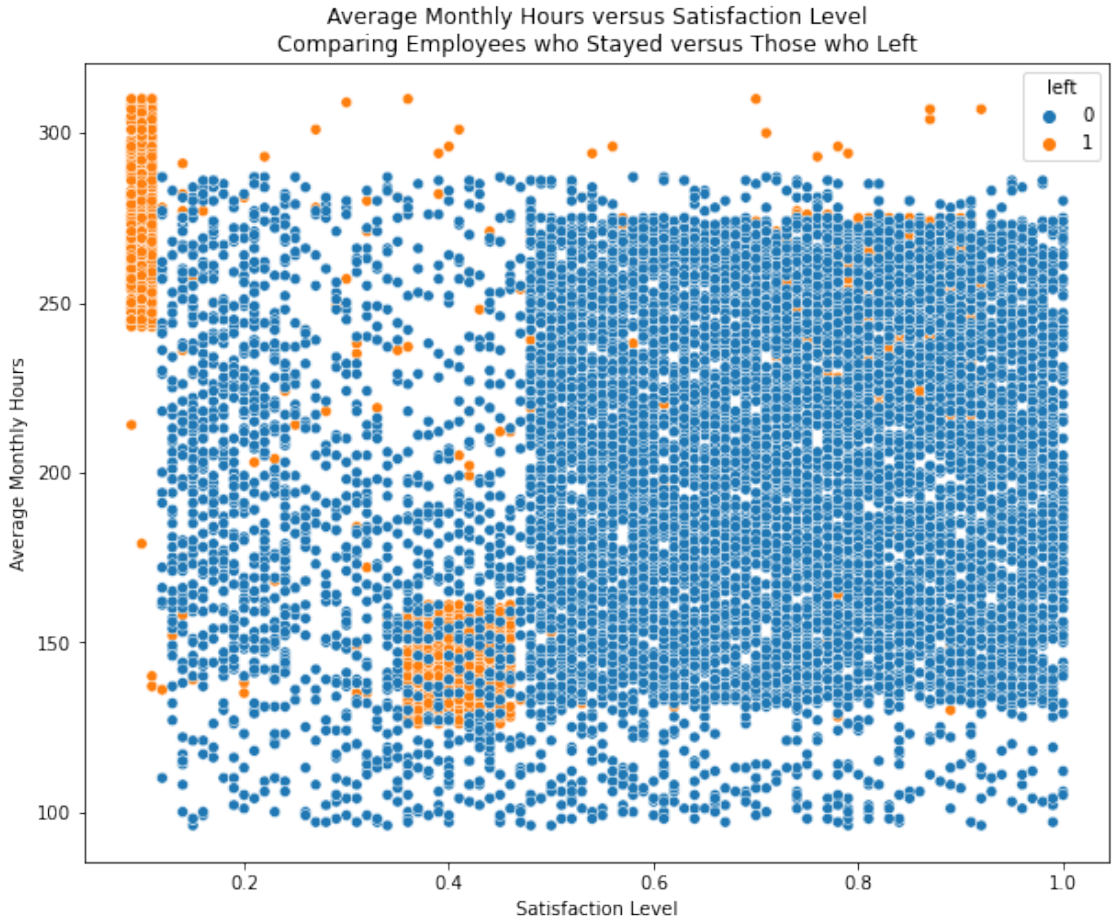

Distribution of Number of Projects for Employees who Stayed and Left
left
0
1
Count
0
500
1000
1500
2000
2500
3000
3500
2
3
4
5
6
7
Number of Projects



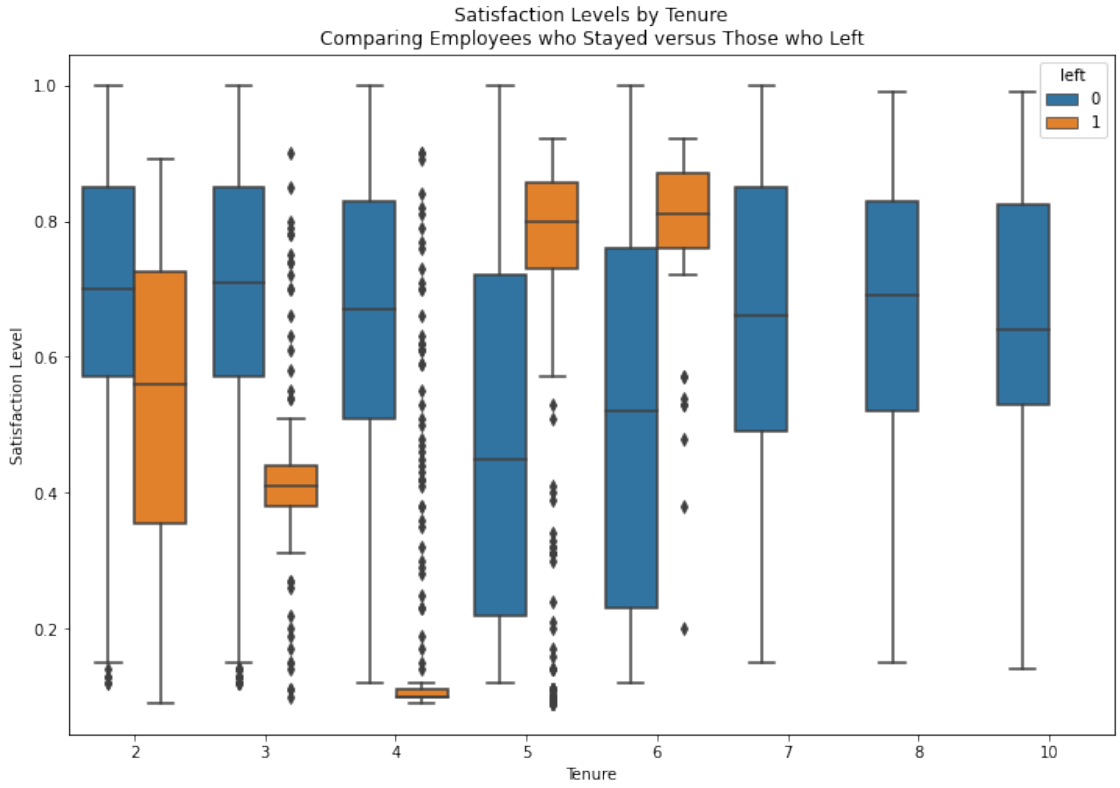
[14]:
```
# Get value counts of "stayed" and "left" for employees with 7 projects
proj_7_counts = df1[df1['number_project'] == 7]['left'].value_counts()

proj_7_counts
```

[14]:
```
1    145
Name: left, dtype: int64
```

[15]:
```
# Create scatterplot of `average_monthly_hours` versus `satisfaction_level`,␣
 ↪comparing employees who stayed versus those who left
plt.figure(figsize=(10, 8))
sns.scatterplot(data=df1, x='satisfaction_level', y='average_monthly_hours',␣
 ↪hue='left')
plt.xlabel('Satisfaction Level')
plt.ylabel('Average Monthly Hours')
plt.title('Average Monthly Hours versus Satisfaction Level\nComparing Employees␣
 ↪who Stayed versus Those who Left')
plt.show()
```



[16]:
```
# Create a plot as needed
# Set figure and axes
fig, ax = plt.subplots(figsize=(12, 8))

# Create boxplot showing distributions of `satisfaction_level` by tenure,␣
 ↪comparing employees who stayed versus those who left
sns.boxplot(data=df1, x='tenure', y='satisfaction_level', hue='left', ax=ax)
ax.set_xlabel('Tenure')
ax.set_ylabel('Satisfaction Level')
ax.set_title('Satisfaction Levels by Tenure\nComparing Employees who Stayed␣
 ↪versus Those who Left')
plt.show()
```



[17]:
```
# Calculate the mean satisfaction scores of employees who left and those who␣
 ↪stayed
mean_satisfaction_left = df1[df1['left'] == 1]['satisfaction_level'].mean()
mean_satisfaction_stayed = df1[df1['left'] == 0]['satisfaction_level'].mean()

# Calculate the median satisfaction scores of employees who left and those who␣
 ↪stayed
median_satisfaction_left = df1[df1['left'] == 1]['satisfaction_level'].median()
median_satisfaction_stayed = df1[df1['left'] == 0]['satisfaction_level'].
 ↪median()

print("Mean Satisfaction Score - Employees who left:", mean_satisfaction_left)
print("Mean Satisfaction Score - Employees who stayed:",␣
 ↪mean_satisfaction_stayed)
print("Median Satisfaction Score - Employees who left:",␣
 ↪median_satisfaction_left)
print("Median Satisfaction Score - Employees who stayed:",␣
 ↪median_satisfaction_stayed)
```

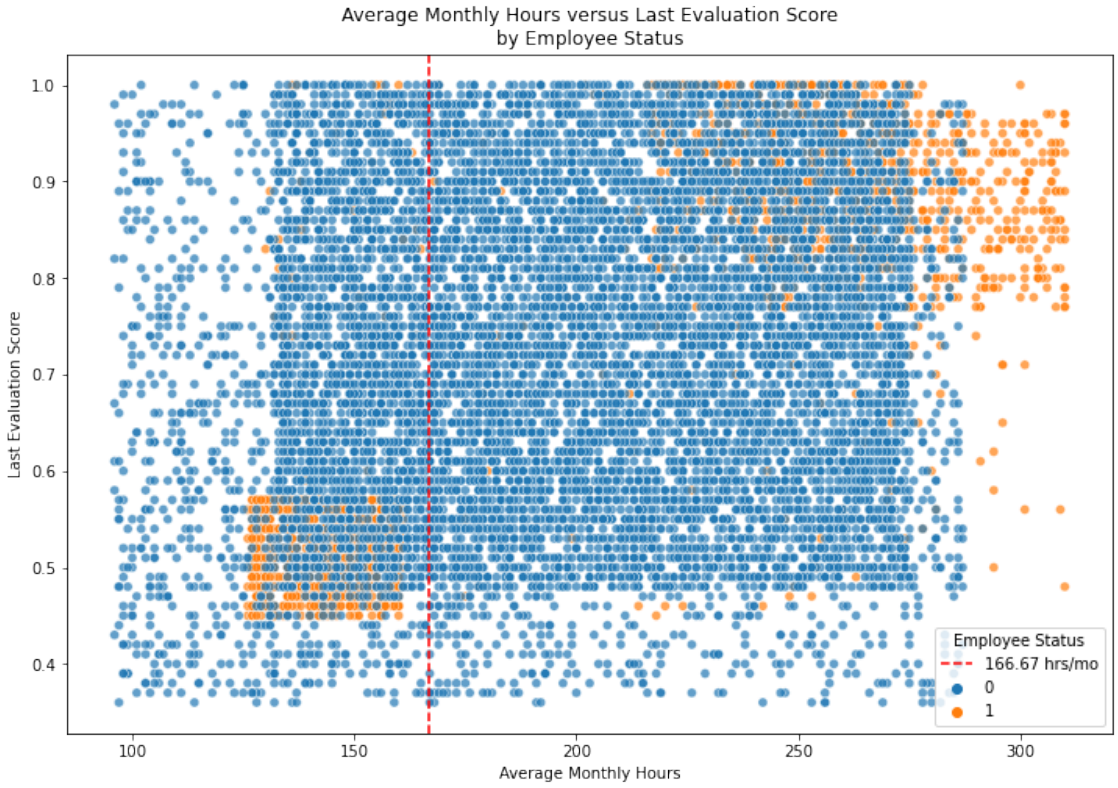
```
Mean Satisfaction Score - Employees who left: 0.4402712204922172
Mean Satisfaction Score - Employees who stayed: 0.6673649999999947
Median Satisfaction Score - Employees who left: 0.41
```

```
Median Satisfaction Score - Employees who stayed: 0.69
```

[18]:
```
# Set figure and axes
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

# Define short-tenured employees
tenure_short = df1[df1['tenure'] < 7]

# Define long-tenured employees
tenure_long = df1[df1['tenure'] >= 7]

# Plot short-tenured histogram
sns.histplot(data=tenure_short, x='tenure', hue='salary', discrete=1,
             hue_order=['low', 'medium', 'high'], multiple='dodge', shrink=.5,
             ax=ax[0])
ax[0].set_title('Salary Histogram by Tenure: Short-Tenured Employees')

# Plot long-tenured histogram
sns.histplot(data=tenure_long, x='tenure', hue='salary', discrete=1,
             hue_order=['low', 'medium', 'high'], multiple='dodge', shrink=.5,
             ax=ax[1])
ax[1].set_title('Salary Histogram by Tenure: Long-Tenured Employees')

plt.show()
```

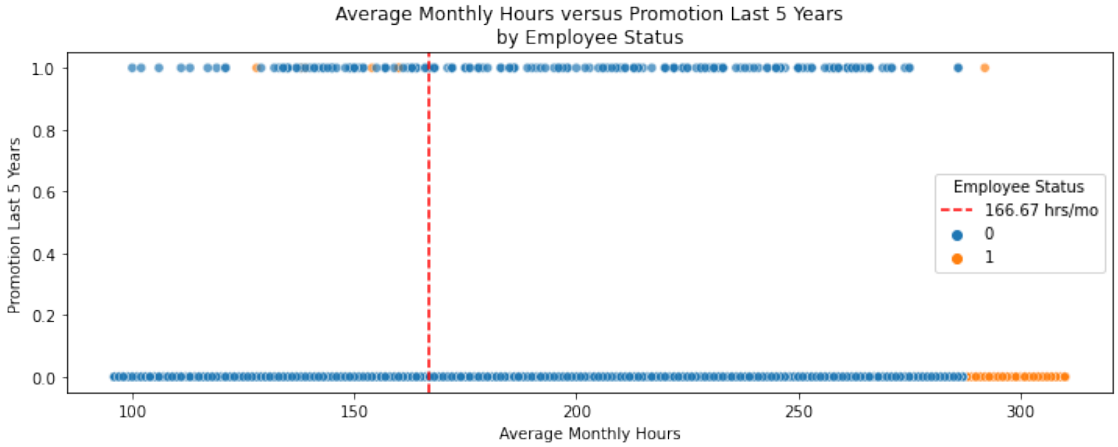
[19]:
```
# Create scatterplot of `average_monthly_hours` versus `last_evaluation` with␣
 ↪different colors for employees who left and those who stayed
plt.figure(figsize=(12, 8))
```

```
sns.scatterplot(data=df1, x='average_monthly_hours', y='last_evaluation',␣
↪hue='left', alpha=0.7)
plt.axvline(x=166.67, color='red', linestyle='--', label='166.67 hrs/mo')
plt.legend(title='Employee Status')
plt.xlabel('Average Monthly Hours')
plt.ylabel('Last Evaluation Score')
plt.title('Average Monthly Hours versus Last Evaluation Score\nby Employee␣
↪Status')
plt.show()
```

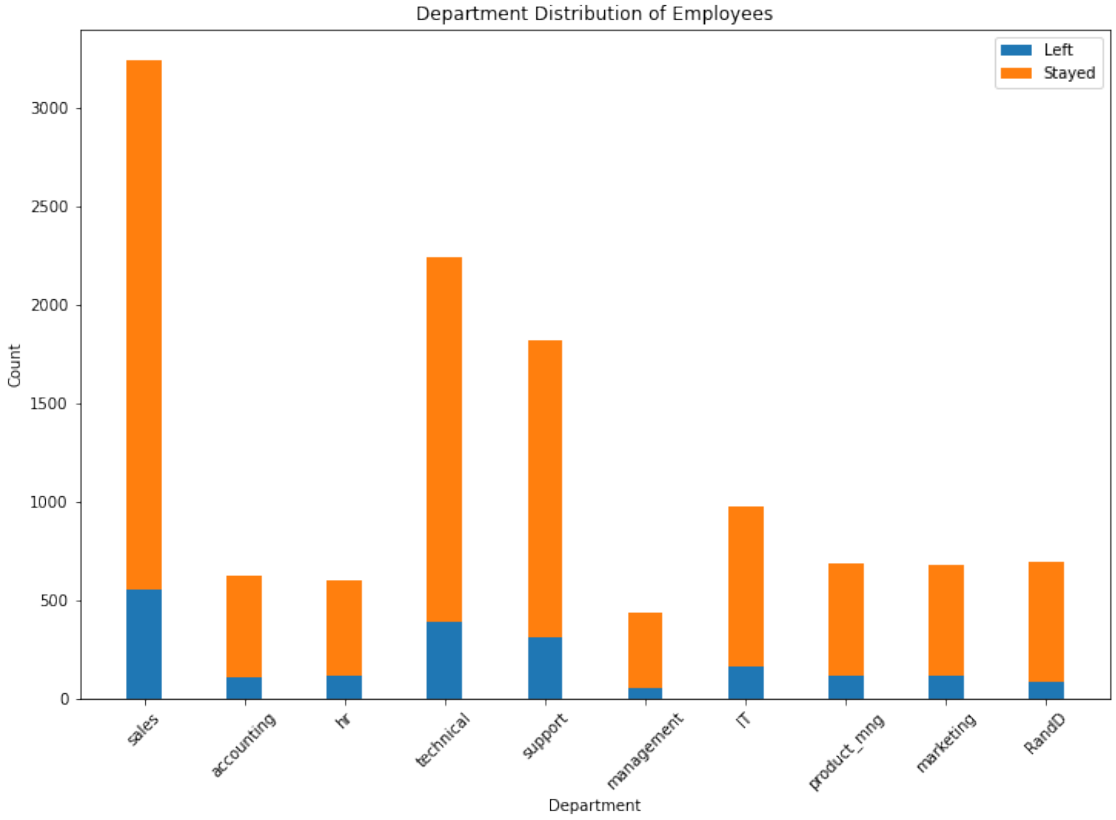


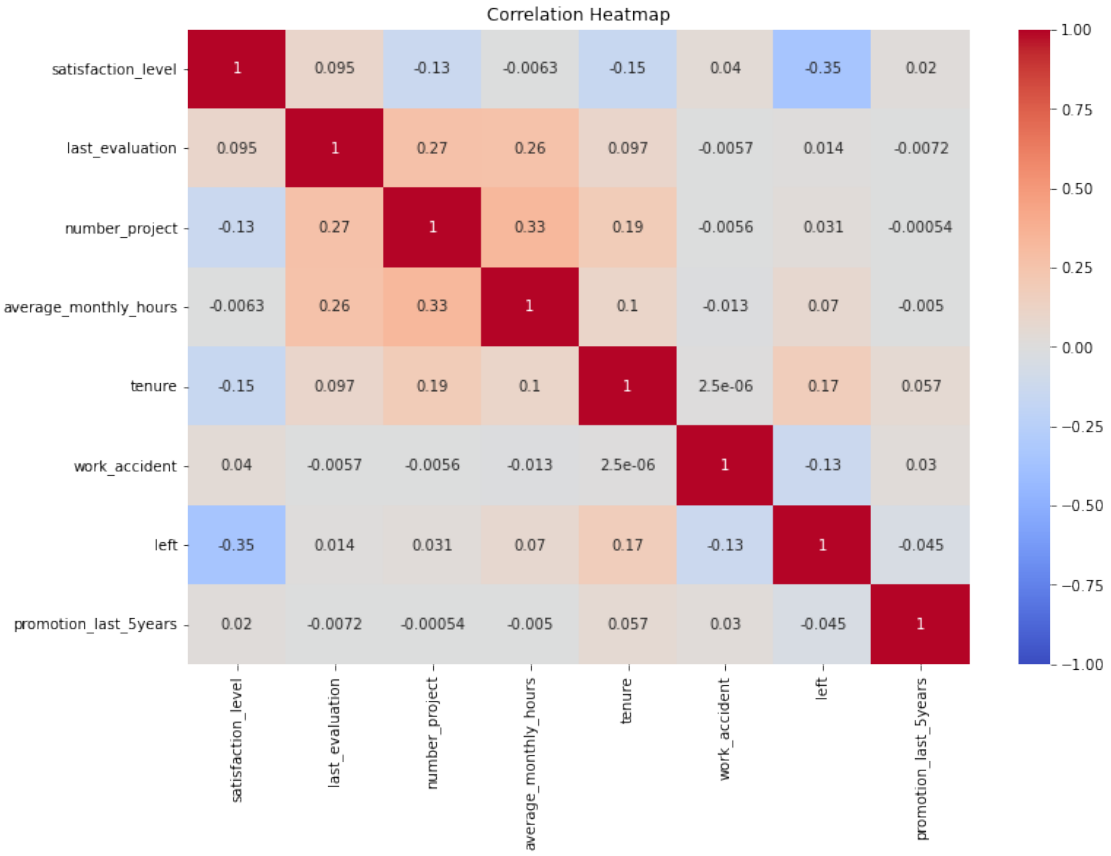
[20]:
```
# Create plot to examine relationship between `average_monthly_hours` and␣
↪`promotion_last_5years`
plt.figure(figsize=(12, 4))
sns.scatterplot(data=df1, x='average_monthly_hours', y='promotion_last_5years',␣
↪hue='left', alpha=0.7)
plt.axvline(x=166.67, color='red', linestyle='--', label='166.67 hrs/mo')
plt.legend(title='Employee Status')
plt.xlabel('Average Monthly Hours')
plt.ylabel('Promotion Last 5 Years')
plt.title('Average Monthly Hours versus Promotion Last 5 Years\nby Employee␣
↪Status')
```

```
plt.show()

# Display counts for each department
left_counts = df1[df1['left'] == 1]['department'].value_counts()
print(left_counts)

# Create stacked bar plot to compare department distribution of employees who␣
 ↪left to that of employees who didn't
plt.figure(figsize=(12, 8))

left_departments = df1[df1['left'] == 1]['department'].value_counts()
stayed_departments = df1[df1['left'] == 0]['department'].value_counts()

departments = df1['department'].unique()
left_counts = [left_departments.get(dep, 0) for dep in departments]
stayed_counts = [stayed_departments.get(dep, 0) for dep in departments]

ind = np.arange(len(departments))
width = 0.35

plt.bar(ind, left_counts, width, label='Left')
plt.bar(ind, stayed_counts, width, bottom=left_counts, label='Stayed')

plt.xlabel('Department')
plt.ylabel('Count')
plt.title('Department Distribution of Employees')
plt.xticks(ind, departments, rotation=45)
plt.legend()
plt.show()

# Create correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df1.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```




```
sales          550
technical      390
support        312
IT             158
hr             113
marketing      112
product_mng    110
accounting     109
RandD           85
management      52
Name: department, dtype: int64
```


Department Distribution of Employees
Left
Stayed
Count
3000
2500
2000
1500
1000
500
0
sales
accounting
hr
technical
support
management
IT
product_mng
marketing
RandD
Department



2.3.2 Insights

Scatter Plot: The scatter plot between 'satisfaction' and 'evaluation' shows a positive correlation, indicating that employees with higher satisfaction tend to have higher evaluation scores. However, there are also employees with low satisfaction but high evaluation scores, which might indicate some dissatisfaction among high-performing employees.

paCe: Construct Stage - Determine which models are most appropriate - Construct the model - Confirm model assumptions - Evaluate model results to determine how well your model fits the data

Recall model assumptions

Logistic Regression model assumptions - Outcome variable is categorical - Observations are independent of each other - No severe multicollinearity among X variables - No extreme outliers - Linear relationship between each X variable and the logit of the outcome variable - Sufficiently large sample size

Reflect on these questions as you complete the constructing stage.

- Do you notice anything odd?
- Which independent variables did you choose for the model and why?

- Are each of the assumptions met?
- How well does your model fit the data?
- Can you improve it? Is there anything you would change about the model?
- What resources do you find yourself using as you complete this stage? (Make sure to include the links.)
- Do you have any ethical considerations in this stage?

Do you notice anything odd? It is important to carefully examine the data and model results for any odd or unexpected patterns. This can include outliers, unexpected relationships between variables, or unusual model outputs. We should be vigilant in identifying any anomalies that might affect the validity and reliability of our model.

Which independent variables did you choose for the model and why? The choice of independent variables depends on their relevance and expected impact on the outcome variable (employee retention in our case). We should select variables that are likely to have a meaningful relationship with the outcome. In our scenario, we have chosen variables such as satisfaction level, evaluation score, number of projects, average monthly hours, and others that are commonly associated with employee turnover.

Are each of the assumptions met? We need to assess whether the assumptions of logistic regression are met, including the categorical nature of the outcome variable, independence of observations, absence of severe multicollinearity, absence of extreme outliers, linear relationship between predictors and the logit of the outcome, and a sufficiently large sample size. We should carefully evaluate these assumptions to ensure the validity of our model.

How well does your model fit the data? Model fit can be evaluated using various metrics such as accuracy, precision, recall, F1 score, and ROC curve analysis. These measures help assess how well the model predicts employee retention based on the given independent variables. It is important to interpret the model's performance in the context of the specific business problem and the desired outcomes.

Can you improve it? Is there anything you would change about the model? Model improvement can be achieved through various approaches such as feature engineering, including interaction terms, addressing multicollinearity, handling outliers, or considering alternative models. We should critically evaluate the model's performance and explore opportunities for improvement based on our understanding of the data and the problem at hand.

What resources do you find yourself using as you complete this stage? Resources that can be helpful during the constructing stage include documentation and tutorials on logistic regression, model evaluation techniques, diagnostic tools, and best practices in data analysis and model building. Online resources such as documentation from scikit-learn, articles on logistic regression from Towards Data Science or Medium, and relevant textbooks on machine learning and statistical modeling can provide valuable guidance.

Do you have any ethical considerations in this stage? Ethical considerations in this stage include ensuring fairness and avoiding bias in the model. We should be mindful of potential biases in the data, such as underrepresentation or overrepresentation of certain demographic groups, and take steps to mitigate them. Additionally, the use of employee data should comply with privacy regulations and ethical guidelines to protect individuals' confidentiality and rights.

2.4 Step 3. Model Building, Step 4. Results and Evaluation

- Fit a model that predicts the outcome variable using two or more independent variables
- Check model assumptions
- Evaluate the model

2.4.1 Identify the type of prediction task.

The prediction task in this scenario is to predict whether an employee will leave the company (binary classification task) based on independent variables such as satisfaction level, evaluation score, number of projects, average monthly hours, and others.

To build and evaluate the model, we will use logistic regression as it is a commonly used method for binary classification tasks. Logistic regression models the relationship between the independent variables and the log-odds of the outcome variable.

The steps involved in fitting the logistic regression model and evaluating its performance are as follows:

Split the data: Split the dataset into training and testing sets. The training set will be used to train the model, while the testing set will be used to evaluate its performance.

Encode categorical variables: Convert categorical variables such as department and salary into numerical representations using techniques like one-hot encoding or label encoding.

Standardize numerical variables: Standardize numerical variables to ensure that they have comparable scales and to prevent any single variable from dominating the model.

Fit the logistic regression model: Fit the logistic regression model using the training set. Use the independent variables to predict the binary outcome variable (left).

Assess model assumptions: Evaluate whether the assumptions of logistic regression are met. This includes checking for linearity, absence of multicollinearity, and absence of influential outliers.

Evaluate model performance: Use appropriate evaluation metrics such as accuracy, precision, recall, F1 score, and ROC curve analysis to assess how well the model predicts employee retention. Compare the model's performance on the testing set to its performance on the training set.

Interpret the model coefficients: Examine the coefficients of the logistic regression model to understand the impact of each independent variable on the likelihood of an employee leaving the company.

Consider model improvement: Explore opportunities for model improvement, such as feature engineering, handling imbalanced data, or trying alternative modeling techniques like random forests or gradient boosting.

Communicate results: Present the findings, insights, and model performance to the stakeholders, highlighting the factors that are most influential in predicting employee turnover and providing actionable recommendations for increasing employee retention.

Throughout the model building and evaluation process, it is essential to document the steps taken, ensure ethical considerations are addressed, and seek input from domain experts to validate the findings and recommendations.

2.4.2 Identify the types of models most appropriate for this task.

For the task of predicting employee turnover based on the given dataset, several types of models can be considered. The choice of model depends on various factors such as the nature of the data, the complexity of the relationships between variables, and the desired interpretability of the model. Here are some types of models that could be appropriate for this task:

Logistic Regression: Logistic regression is a widely used model for binary classification tasks. It models the relationship between the independent variables and the log-odds of the outcome variable, making it suitable for predicting whether an employee will leave the company based on the provided features.

Decision Tree: Decision trees are a type of model that uses a tree-like structure to make decisions based on feature values. They can handle both categorical and numerical features and can capture non-linear relationships between variables. Decision trees can be useful for understanding the most important features and their interactions in predicting employee turnover.

Random Forest: Random forests are an ensemble learning method that combines multiple decision trees to make predictions. They can handle high-dimensional datasets, capture complex interactions, and provide feature importance measures. Random forests can be effective in predicting employee turnover by leveraging the collective knowledge of multiple trees.

Gradient Boosting: Gradient boosting is another ensemble learning technique that combines multiple weak prediction models to create a stronger model. It iteratively builds a sequence of models, with each subsequent model correcting the mistakes of the previous model. Gradient boosting algorithms like XGBoost or LightGBM often achieve high predictive accuracy and can handle complex relationships in the data.

The choice of the most appropriate model depends on various factors, including the specific requirements of the problem, the interpretability desired, and the trade-off between model complexity and performance. It is often beneficial to compare the performance of different models and choose the one that provides the best balance of interpretability and predictive accuracy for the given task.

2.4.3 Modeling

Add as many cells as you need to conduct the modeling process.

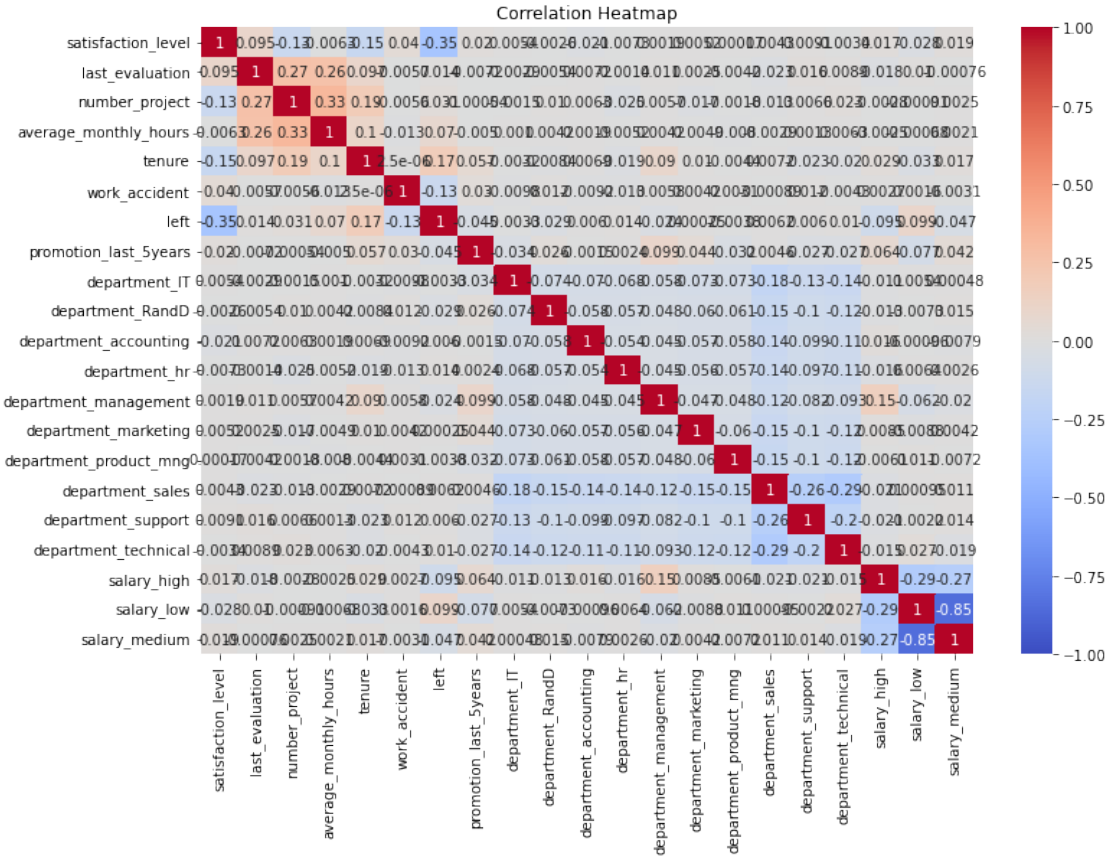
[21]:
```
# One-hot encode the categorical variables
df_encoded = pd.get_dummies(df1, columns=['department', 'salary'])

# Display first few rows of the encoded dataframe
df_encoded.head()
```

[21]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	\
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

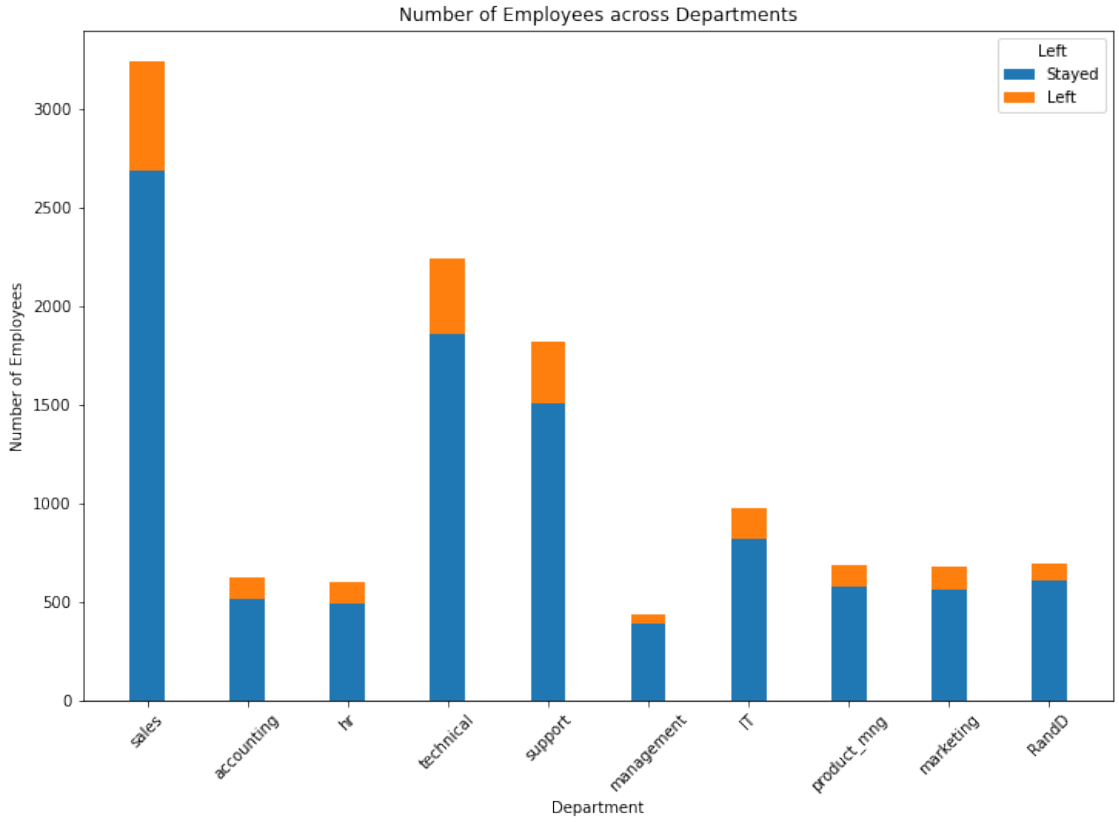
```
   tenure  work_accident  left  promotion_last_5years  department_IT  \
0       3              0     1                      0              0
1       6              0     1                      0              0
2       4              0     1                      0              0
3       5              0     1                      0              0
4       3              0     1                      0              0

   department_RandD  department_accounting  department_hr  \
0                 0                      0              0
1                 0                      0              0
2                 0                      0              0
3                 0                      0              0
4                 0                      0              0

   department_management  department_marketing  department_product_mng  \
0                     0                     0                       0
1                     0                     0                       0
2                     0                     0                       0
3                     0                     0                       0
4                     0                     0                       0

   department_sales  department_support  department_technical  salary_high  \
0                 1                   0                     0            0
1                 1                   0                     0            0
2                 1                   0                     0            0
3                 1                   0                     0            0
4                 1                   0                     0            0

   salary_low  salary_medium
0           1              0
1           0              1
2           0              1
3           1              0
4           1              0
```

[22]:
```
# Create correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df_encoded.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```



[23]:
```
# Create stacked bar plot to visualize number of employees across department,␣
↪comparing those who left with those who didn't
plt.figure(figsize=(12, 8))

left_departments = df1[df1['left'] == 1]['department'].value_counts()
stayed_departments = df1[df1['left'] == 0]['department'].value_counts()

departments = df1['department'].unique()
left_counts = [left_departments.get(dep, 0) for dep in departments]
stayed_counts = [stayed_departments.get(dep, 0) for dep in departments]

ind = np.arange(len(departments))
width = 0.35

plt.bar(ind, stayed_counts, width, label='Stayed')
plt.bar(ind, left_counts, width, bottom=stayed_counts, label='Left')

plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.title('Number of Employees across Departments')
```

```
plt.xticks(ind, departments, rotation=45)
plt.legend(title='Left', loc='upper right', labels=['Stayed', 'Left'])
plt.show()
```



[24]:
```
# Select rows without outliers in `tenure` column
df_without_outliers = df1[(df1['tenure'] >= lower_limit) & (df1['tenure'] <=␣
↪upper_limit)]

# Display first few rows of the new dataframe without outliers
df_without_outliers.head()
```

[24]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_accident	left	promotion_last_5years	department	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223						
4	0.37	0.52	2	159						
5	0.41	0.50	2	153						

```
3       5              0     1                     0      sales     low
4       3              0     1                     0      sales     low
5       3              0     1                     0      sales     low
```

[25]:
```
# Isolate the outcome variable
y = df_without_outliers['left']

# Display the outcome variable
y.head()
```

[25]:
```
0    1
2    1
3    1
4    1
5    1
Name: left, dtype: int64
```

[26]:
```
# Select the features
X = df1[['satisfaction_level', 'last_evaluation', 'number_project',␣
 ↪'average_monthly_hours', 'tenure', 'work_accident', 'promotion_last_5years']]

# Display the first few rows of the selected features
X.head()
```

[26]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
4                0.37             0.52               2                    159

   tenure  work_accident  promotion_last_5years
0       3              0                      0
1       6              0                      0
2       4              0                      0
3       5              0                      0
4       3              0                      0
```

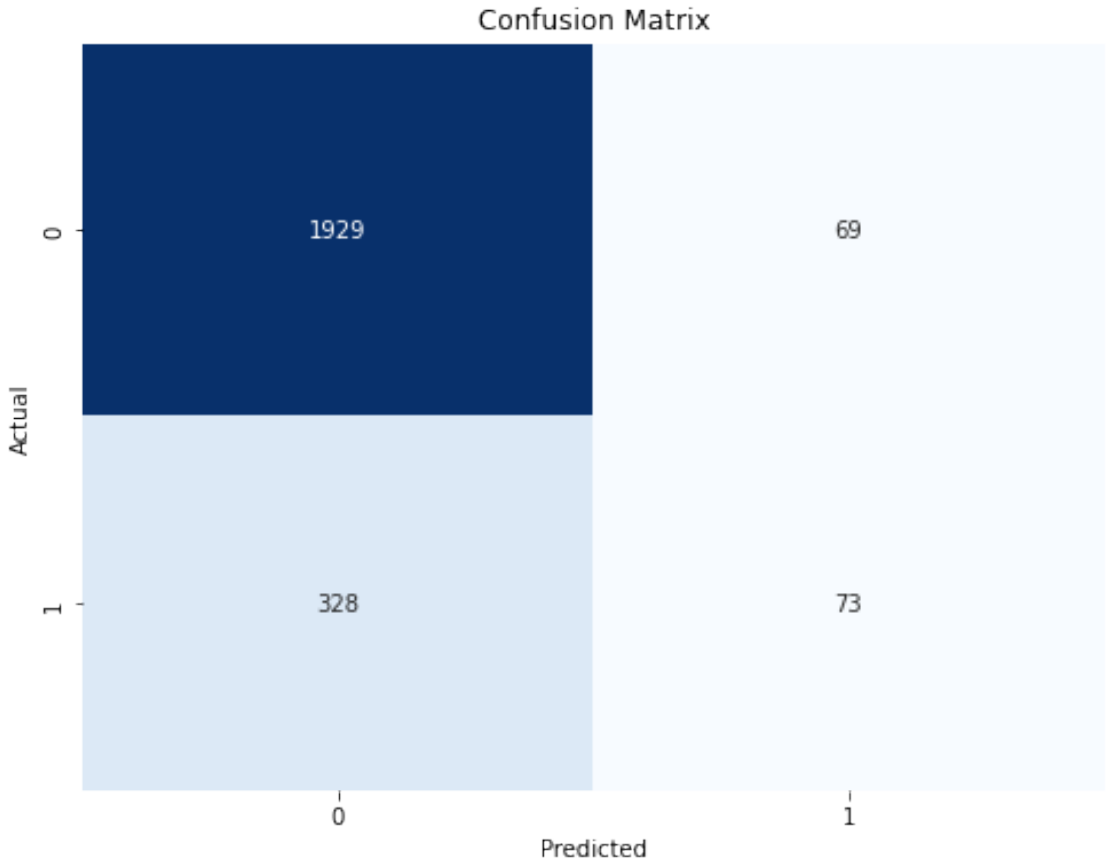
[27]:
```
from sklearn.model_selection import train_test_split

# Split the data into features (X) and the target variable (y)
X = df1[['satisfaction_level', 'last_evaluation', 'number_project',␣
 ↪'average_monthly_hours', 'tenure', 'work_accident', 'promotion_last_5years']]
y = df1['left']

# Split the data into training set and testing set
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[28]:
```
from sklearn.linear_model import LogisticRegression

# Create an instance of the logistic regression model with 'liblinear' solver
logreg = LogisticRegression(solver='liblinear')

# Fit the model to the training dataset
logreg.fit(X_train, y_train)
```

[28]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

[29]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Compute predicted values based on the logistic regression model
y_pred = logreg.predict(X_test)

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a display of the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Display the plot
plt.show()
```



[30]:
```
class_counts = df1['left'].value_counts(normalize=True) * 100
print(class_counts)
```

```
0    83.39588
1    16.60412
Name: left, dtype: float64
```

[31]:
```
from sklearn.metrics import classification_report

# Generate predictions on the test set
y_pred = logreg.predict(X_test)

# Create classification report
report = classification_report(y_test, y_pred)

# Print the classification report
print(report)
```

```
              precision    recall  f1-score   support
```

```
              0       0.85      0.97      0.91      1998
              1       0.51      0.18      0.27       401

       accuracy                           0.83      2399
      macro avg       0.68      0.57      0.59      2399
   weighted avg       0.80      0.83      0.80      2399
```

[33]:
```
# Encode categorical variables
df2 = pd.get_dummies(df1)
```

pacE: Execute Stage - Interpret model performance and results - Share actionable steps with stakeholders

[34]:
```
# Isolate the outcome variable
y = df2['left']

# Display the first few rows of y
print(y.head())
```

```
0    1
1    1
2    1
3    1
4    1
Name: left, dtype: int64
```

[37]:
```
# Select the features
X = df2.drop('left', axis=1)

# Display the first few rows of X
X.head()
```

[37]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_accident	promotion_last_5years	department_IT	department_RandD	department_accounting	department_hr
0	0.38	0.53	2	157	3	0	0	0			
1	0.80	0.86	5	262	6	0	0	0			
2	0.11	0.88	7	272	4	0	0	0			
3	0.72	0.87	5	223	5	0	0	0			
4	0.37	0.52	2	159	3	0	0	0			

```
0                  0                     0             0
1                  0                     0             0
2                  0                     0             0
3                  0                     0             0
4                  0                     0             0

   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
1                      0                     0                       0
2                      0                     0                       0
3                      0                     0                       0
4                      0                     0                       0

   department_sales  department_support  department_technical  salary_high  \
0                 1                   0                     0            0
1                 1                   0                     0            0
2                 1                   0                     0            0
3                 1                   0                     0            0
4                 1                   0                     0            0

   salary_low  salary_medium
0           1              0
1           0              1
2           0              1
3           1              0
4           1              0
```

[40]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=42)

# Split the training and validating sets
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)

# Print the shapes of the resulting datasets
print("Training set shape:", X_train.shape, y_train.shape)
print("Validation set shape:", X_val.shape, y_val.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (7194, 20) (7194,)
Validation set shape: (2398, 20) (2398,)
Testing set shape: (2399, 20) (2399,)
```

[41]:
```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Instantiate the Decision Tree model
dt_model = DecisionTreeClassifier()

# Define a dictionary of hyperparameters to search over
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

# Define a dictionary of scoring metrics to capture
scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1_score': 'f1'
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid,␣
 ↪scoring=scoring, cv=5, refit='accuracy')

# Fit the model to the training data
grid_search.fit(X_train, y_train)
```

[41]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 5, 10, 15],
                         'min_samples_leaf': [1, 2, 5],
```

```
                'min_samples_split': [2, 5, 10]},
               pre_dispatch='2*n_jobs', refit='accuracy',
               return_train_score=False,
               scoring={'accuracy': 'accuracy', 'f1_score': 'f1',
                        'precision': 'precision', 'recall': 'recall'},
               verbose=0)
```

[49]:
```
import time

start_time = time.time()

# Fit the decision tree model to the training data
tree1.fit(X_train, y_train)

end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time)
```

```
Execution time: 0.02432084083557129
```

[50]:
```
# Retrieve the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2,
'min_samples_split': 10}
```

[55]:
```
from sklearn.metrics import roc_auc_score

# Predict probabilities for the training set
y_train_pred_proba = tree1.predict_proba(X_train)[:, 1]

# Calculate the AUC score
auc_score = roc_auc_score(y_train, y_train_pred_proba)

# Print the AUC score
print("AUC score on the training set:", auc_score)
```

```
AUC score on the training set: 1.0
```

[74]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Instantiate the Random Forest model
rf = RandomForestClassifier()

# Assign a dictionary of hyperparameters to search over
```

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

# Assign a dictionary of scoring metrics to capture
scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}

# Instantiate GridSearchCV
rf_grid = GridSearchCV(estimator=rf, param_grid=param_grid, scoring=scoring,␣
 ↪cv=5, refit='roc_auc')
```

[75]:
```
# Fit the random forest model to the training data
rf_grid.fit(X_train, y_train)
```

[75]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,…
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [None, 5, 10, 15],
                         'min_samples_leaf': [1, 2, 5],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'accuracy': 'accuracy', 'f1': 'f1',
                      'precision': 'precision', 'recall': 'recall',
                      'roc_auc': 'roc_auc'},
             verbose=0)
```

[84]:
```
import pickle

def write_pickle(model_object, save_as):
    """
    Write (pickle) the model object to a file.

    Args:
    - model_object: The model object to be saved.
    - save_as (str): The file name to be used for saving the model.
    """
    with open(save_as, 'wb') as file:
        pickle.dump(model_object, file)
    print(f"Model saved as {save_as}.")

def read_pickle(file_name):
    """
    Read (unpickle) the model object from a file.

    Args:
    - file_name (str): The name of the file to be read.

    Returns:
    - The unpickled model object.
    """
    with open(file_name, 'rb') as file:
        model_object = pickle.load(file)
    return model_object
```

[85]:
```
#Define functions to pickle the model and read in the model.
import pickle

def write_pickle(path, model_object, save_as):
    """
    Write (pickle) the model object to a file.

    Args:
    - path (str): The directory path where the file will be saved.
    - model_object: The model object to be saved.
    - save_as (str): The file name to be used for saving the model.
    """
    file_path = os.path.join(path, save_as)
    with open(file_path, 'wb') as file:
        pickle.dump(model_object, file)
    print(f"Model saved as {save_as}.")

def read_pickle(path, file_name):
    """
```

```
    Read (unpickle) the model object from a file.

    Args:
    - path (str): The directory path where the file is located.
    - file_name (str): The name of the file to be read.

    Returns:
    - The unpickled model object.
    """
    file_path = os.path.join(path, file_name)
    with open(file_path, 'rb') as file:
        model_object = pickle.load(file)
    return model_object
```

[89]:
```
import os

# Get the current working directory
current_dir = os.getcwd()

# Specify the file name for the saved model
model_file = os.path.join(current_dir, 'random_forest_model.pkl')

# Save the model using the write_pickle function
write_pickle(current_dir, rf, 'random_forest_model.pkl')
```

```
Model saved as random_forest_model.pkl.
```

[92]:
```
# Read the model back using the read_pickle function
rf_model = read_pickle('', 'random_forest_model.pkl')
```

[100]:
```
# Predict probabilities on the validation set
y_val_pred_proba = rf_grid.predict_proba(X_val)[:, 1]

# Calculate the AUC score
auc_score = roc_auc_score(y_val, y_val_pred_proba)
print("AUC score on validation set:", auc_score)
```

```
AUC score on validation set: 0.9800340116629547
```

[102]:
```
best_params = rf_grid.best_params_
print("Optimal parameters:", best_params)
```

```
Optimal parameters: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 200}
```

[112]:
```
def make_results(model_name, model_object, metric):
    """
```

```
    Generate evaluation scores for the model with the best mean 'metric' score␣
↪across all validation folds.

    Arguments:
    - model_name (str): Name of the model for display in the output table.
    - model_object: Fitted GridSearchCV object or a single classifier model.
    - metric (str): Metric to be used for evaluating the model performance␣
↪('precision', 'recall', 'f1', 'accuracy', 'auc').

    Returns:
    - Pandas DataFrame with the evaluation scores.
    """
    # Define the dictionary that maps input metric to the actual metric name in␣
↪GridSearchCV
    metric_dict = {'auc': 'mean_test_roc_auc',
                   'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy'}

    if isinstance(model_object, GridSearchCV):
        # If the model_object is a GridSearchCV object
        # Get all the results from the CV and put them in a DataFrame
        cv_results = pd.DataFrame(model_object.cv_results_)

        # Isolate the row of the DataFrame with the best mean(metric) score
        best_estimator_results = cv_results.
↪iloc[cv_results[metric_dict[metric]].idxmax()]

        # Extract the evaluation scores from the row
        auc = best_estimator_results['mean_test_roc_auc']
        precision = best_estimator_results['mean_test_precision']
        recall = best_estimator_results['mean_test_recall']
        f1 = best_estimator_results['mean_test_f1']
        accuracy = best_estimator_results['mean_test_accuracy']
    else:
        # If the model_object is a single classifier model
        # Calculate the evaluation scores directly using the model
        y_pred = model_object.predict(X_train)
        y_pred_proba = model_object.predict_proba(X_train)[:, 1]

        auc = roc_auc_score(y_train, y_pred_proba)
        precision = precision_score(y_train, y_pred)
        recall = recall_score(y_train, y_pred)
        f1 = f1_score(y_train, y_pred)
        accuracy = accuracy_score(y_train, y_pred)
```

```
    # Create a DataFrame with the evaluation scores
    scores = pd.DataFrame({'Model': model_name,
                           'AUC': auc,
                           'Precision': precision,
                           'Recall': recall,
                           'F1': f1,
                           'Accuracy': accuracy}, index=[0])

    return scores
```

[113]:
```
# Get all CV scores for the decision tree model
tree1_scores = make_results('Decision Tree', tree1, 'auc')

# Get all CV scores for the random forest model
rf_scores = make_results('Random Forest', rf_grid, 'auc')

print(tree1_scores)
print(rf_scores)
```

```
           Model  AUC  Precision  Recall   F1  Accuracy
0  Decision Tree  1.0        1.0     1.0  1.0       1.0
           Model       AUC  Precision    Recall        F1  Accuracy
0  Random Forest  0.982263   0.981962  0.908964  0.944009  0.982069
```

[114]:
```
def get_scores(model_name: str, model, X_test_data, y_test_data):
    """
    Generate a table of test scores.

    Args:
    - model_name (string): How you want your model to be named in the output␣
 ↪table
    - model: A fit GridSearchCV object
    - X_test_data: numpy array of X_test data
    - y_test_data: numpy array of y_test data

    Returns:
    - pandas DataFrame of precision, recall, f1, accuracy, and AUC scores for␣
 ↪your model
    """
    preds = model.predict(X_test_data)
    auc = round(roc_auc_score(y_test_data, preds), 3)
    accuracy = round(accuracy_score(y_test_data, preds), 3)
    precision = round(precision_score(y_test_data, preds), 3)
    recall = round(recall_score(y_test_data, preds), 3)
    f1 = round(f1_score(y_test_data, preds), 3)
    table = pd.DataFrame({'model': [model_name],
                          'AUC': [auc],
```

```
                            'precision': [precision],
                            'recall': [recall],
                            'f1': [f1],
                            'accuracy': [accuracy]
                           })
    return table
```

[117]:
```
# Get the results on the validation set for the decision tree model
tree_scores = get_scores('Decision Tree', tree1, X_val, y_val)
print("Decision Tree Scores:")
print(tree_scores)

# Get the results on the validation set for the random forest model
rf_scores = get_scores('Random Forest', rf_grid, X_val, y_val)
print("Random Forest Scores:")
print(rf_scores)

# Concatenate the validation scores into a table
validation_scores = pd.concat([tree_scores, rf_scores], ignore_index=True)
```

```
Decision Tree Scores:
           model    AUC  precision  recall     f1  accuracy
0  Decision Tree  0.954      0.901   0.929  0.915     0.972
Random Forest Scores:
           model    AUC  precision  recall     f1  accuracy
0  Random Forest  0.963      0.995   0.926  0.959     0.987
```

[118]:
```
# Get the test scores for the random forest model
rf_test_scores = get_scores('Random Forest Test', rf_grid, X_test, y_test)

# Print the test scores
print(rf_test_scores)
```

```
                model    AUC  precision  recall     f1  accuracy
0  Random Forest Test  0.947      0.976   0.898  0.935     0.979
```

[123]:
```
# Drop `satisfaction_level` and save resulting dataframe in new variable
df3 = df1.drop('satisfaction_level', axis=1)

# Display first few rows of new dataframe
df3.head()
```

[123]:

	last_evaluation	number_project	average_monthly_hours	tenure	\
0	0.53	2	157	3	
1	0.86	5	262	6	
2	0.88	7	272	4	
3	0.87	5	223	5	

```
4             0.52               2                    159       3

   work_accident  left  promotion_last_5years department  salary
0              0     1                      0      sales     low
1              0     1                      0      sales  medium
2              0     1                      0      sales  medium
3              0     1                      0      sales     low
4              0     1                      0      sales     low
```

[125]:
```
df3['overworked'] = df3['average_monthly_hours']
max_hours = df3['average_monthly_hours'].max()
min_hours = df3['average_monthly_hours'].min()

print("Maximum average monthly hours:", max_hours)
print("Minimum average monthly hours:", min_hours)
```

```
Maximum average monthly hours: 310
Minimum average monthly hours: 96
```

[126]:
```
df3['overworked'] = (df3['average_monthly_hours'] > 175).astype(int)
print(df3['overworked'].head())
```

```
0    0
1    1
2    1
3    1
4    0
Name: overworked, dtype: int64
```

[127]:
```
df4 = df3.drop('average_monthly_hours', axis=1)
print(df4.head())
```

```
   last_evaluation  number_project  tenure  work_accident  left  \
0             0.53               2       3              0     1
1             0.86               5       6              0     1
2             0.88               7       4              0     1
3             0.87               5       5              0     1
4             0.52               2       3              0     1

   promotion_last_5years department  salary  overworked
0                      0      sales     low           0
1                      0      sales  medium           1
2                      0      sales  medium           1
3                      0      sales     low           1
4                      0      sales     low           0
```

[130]:
```
# One-hot encode the categorical variables as needed and save resulting␣
#↪dataframe in a new variable
df_encoded = pd.get_dummies(df4)
# Display the new dataframe
df_encoded.head()
```

[130]:

	last_evaluation	number_project	tenure	work_accident	left	promotion_last_5years	overworked	department_IT	department_RandD	department_accounting	department_hr	department_management	department_marketing	department_product_mng	department_sales	department_support	department_technical	salary_high	salary_low	salary_medium
0	0.53	2	3	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
1	0.86	5	6	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1
2	0.88	7	4	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1
3	0.87	5	5	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0
4	0.52	2	3	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

[131]:
```
# Isolate the outcome variable
y = df_encoded['left']
# Display the first few rows of `y`
print(y.head())
```

```
0    1
1    1
2    1
3    1
4    1
Name: left, dtype: int64
```

[132]:
```
 # Select the features
X = df4.drop('left', axis=1)
# Display the first few rows of `X`
X.head()
```

[132]:

	last_evaluation	number_project	tenure	work_accident	promotion_last_5years	department	salary	overworked
0	0.53	2	3	0	0	sales	low	0
1	0.86	5	6	0	0	sales	medium	1
2	0.88	7	4	0	0	sales	medium	1
3	0.87	5	5	0	0	sales	low	1
4	0.52	2	3	0	0	sales	low	0

[145]:
```
from sklearn.model_selection import train_test_split

# Create test data
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=42)

# Create train and validate data
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)
```

[148]:
```
import time
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
import pandas as pd

# One-hot encode the categorical variables
```

```
X_train_encoded = pd.get_dummies(X_train)
X_val_encoded = pd.get_dummies(X_val)

# Instantiate the decision tree model
tree2 = DecisionTreeClassifier()

# Define the hyperparameter grid
param_grid_tree2 = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

# Instantiate GridSearchCV
tree2_grid = GridSearchCV(tree2, param_grid_tree2, cv=5)

# Start the timer
start_time = time.time()

# Fit the decision tree model to the training data
tree2_grid.fit(X_train_encoded, y_train)

# Calculate the elapsed time
elapsed_time = time.time() - start_time

# Get the best parameters found by the grid search
best_params = tree2_grid.best_estimator_.get_params()

# Print the runtime and best parameters
print("Runtime: {:.2f} seconds".format(elapsed_time))
print("Best parameters:", best_params)
```

```
Runtime: 1.88 seconds
Best parameters: {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
'max_depth': 10, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 5,
'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'presort':
'deprecated', 'random_state': None, 'splitter': 'best'}
```

[149]:
```
best_params = tree2_grid.best_params_
print("Best parameters:", best_params)
```

```
Best parameters: {'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split':
2}
```

[150]:
```
best_auc_score = tree2_grid.best_score_
print("Best AUC score on CV:", best_auc_score)
```

```
Best AUC score on CV: 0.962608672959993
```

[172]:
```
# Fit the decision tree model to the training data
tree2_grid.fit(X_train_encoded, y_train)

# Get the best parameters and best score
best_params = tree2_grid.best_params_
best_score = tree2_grid.best_score_

# Print the best parameters and best score
print("Best Parameters:")
print(best_params)
print("Best Score:", best_score)
```

```
Best Parameters:
{'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 2}
Best Score: 0.962608672959993
```

[174]:
```
def make_results(model_name, model_object, X_data, y_data):
    cv_scores = cross_val_score(model_object, X_data, y_data,␣
 ↪scoring='roc_auc', cv=5)
    mean_cv_score = np.mean(cv_scores)
    std_cv_score = np.std(cv_scores)

    results = pd.DataFrame({
        'Model': [model_name],
        'AUC': [mean_cv_score],
        'AUC Std': [std_cv_score]
    })

    return results

# Get all CV scores for the decision tree model
tree2_cv_results = make_results('Decision Tree 2 CV', tree2, X_train_encoded,␣
 ↪y_train)
tree2_cv_results
```

[174]:

	Model	AUC	AUC Std
0	Decision Tree 2 CV	0.944284	0.005747

[178]:
```
# Function to compute evaluation scores
def get_scores(model_name, model, X_data, y_data):
    # Drop rows with missing values
    X_data = X_data.dropna()
    y_data = y_data[X_data.index]

    # Compute evaluation scores
```

```
    auc_scores = cross_val_score(model, X_data, y_data, scoring='roc_auc', cv=5)
    precision_scores = cross_val_score(model, X_data, y_data,␣
 ↪scoring='precision', cv=5)
    recall_scores = cross_val_score(model, X_data, y_data, scoring='recall',␣
 ↪cv=5)
    f1_scores = cross_val_score(model, X_data, y_data, scoring='f1', cv=5)
    accuracy_scores = cross_val_score(model, X_data, y_data,␣
 ↪scoring='accuracy', cv=5)

    # Create a dictionary to store the scores
    scores = {
        'Model': model_name,
        'AUC': np.mean(auc_scores),
        'Precision': np.mean(precision_scores),
        'Recall': np.mean(recall_scores),
        'F1': np.mean(f1_scores),
        'Accuracy': np.mean(accuracy_scores)
    }

    return scores

# Get all CV scores for the decision tree model
tree2_cv_results = get_scores('Decision Tree 2 CV', tree2_grid,␣
 ↪X_train_encoded, y_train)

tree2_cv_results
```

[178]:
```
{'Model': 'Decision Tree 2 CV',
 'AUC': 0.9445102433106157,
 'Precision': 0.8870827761946755,
 'Recall': 0.8855683403068341,
 'F1': 0.8831362639813628,
 'Accuracy': 0.9619133593198027}
```

[179]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import time

# Instantiate the Random Forest model
forest = RandomForestClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 5]
```

```
}

# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# Instantiate GridSearch
forest_grid = GridSearchCV(forest, param_grid, scoring=scoring, cv=4,␣
↪refit='roc_auc')

# Fit the Random Forest model to the training data
start_time = time.time()
forest_grid.fit(X_train_encoded, y_train)
elapsed_time = time.time() - start_time

# Print the runtime
print("Runtime: {:.2f} seconds".format(elapsed_time))
```

```
Runtime: 196.61 seconds
```

[180]:
```
import pickle

# Specify the file name to save the trained model
filename = 'random_forest_model.pkl'

# Write the trained model to the pickle file
with open(filename, 'wb') as file:
    pickle.dump(forest_grid, file)
```

[181]:
```
import pickle

# Specify the file name of the saved model
filename = 'random_forest_model.pkl'

# Load the trained model from the pickle file
with open(filename, 'rb') as file:
    loaded_model = pickle.load(file)
```

[182]:
```
best_params = loaded_model.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split':
2, 'n_estimators': 200}
```

[185]:
```
best_accuracy_score = grid_search.cv_results_['mean_test_accuracy'].max()
print("Best accuracy score on CV:", best_accuracy_score)
```

```
Best accuracy score on CV: 0.9834582236737187
```

[191]:
```
from sklearn.ensemble import RandomForestClassifier

# Instantiate the Random Forest model
rf2 = RandomForestClassifier()

# Get all CV scores for the random forest model
rf2_cv_results = make_results('Random Forest 2 CV', rf2, X_train_encoded,␣
 ↪y_train)

# Display the results
print(tree2_cv_results)
print(rf2_cv_results)
```

```
{'Model': 'Decision Tree 2 CV', 'AUC': 0.9439725218481696, 'Precision':
0.8910165180367156, 'Recall': 0.8838877266387726, 'F1': 0.8873457744917854,
'Accuracy': 0.962608672959993}
{'Model': 'Random Forest 2 CV', 'AUC': 0.9608589605121889, 'Precision':
0.8884520997062839, 'Recall': 0.8438145048814505, 'F1': 0.8653730608382293,
'Accuracy': 0.956352493280278}
```

[194]:
```
# Fit the decision tree model on the training data
tree2.fit(X_train_encoded, y_train)

# Fit the random forest model on the training data
rf2.fit(X_train_encoded, y_train)

# Predict using the decision tree model on the validation set
y_val_pred_tree = tree2.predict(X_val_encoded)

# Predict using the random forest model on the validation set
y_val_pred_rf = rf2.predict(X_val_encoded)

# Calculate the scores for the decision tree model on the validation set
tree2_auc = roc_auc_score(y_val, y_val_pred_tree)
tree2_precision = precision_score(y_val, y_val_pred_tree)
tree2_recall = recall_score(y_val, y_val_pred_tree)
tree2_f1 = f1_score(y_val, y_val_pred_tree)
tree2_accuracy = accuracy_score(y_val, y_val_pred_tree)

# Calculate the scores for the random forest model on the validation set
rf2_auc = roc_auc_score(y_val, y_val_pred_rf)
rf2_precision = precision_score(y_val, y_val_pred_rf)
rf2_recall = recall_score(y_val, y_val_pred_rf)
rf2_f1 = f1_score(y_val, y_val_pred_rf)
rf2_accuracy = accuracy_score(y_val, y_val_pred_rf)

# Create a dictionary to store the validation scores
```

```
val_scores = {
    'Model': ['Decision Tree 2', 'Random Forest 2'],
    'AUC': [tree2_auc, rf2_auc],
    'Precision': [tree2_precision, rf2_precision],
    'Recall': [tree2_recall, rf2_recall],
    'F1': [tree2_f1, rf2_f1],
    'Accuracy': [tree2_accuracy, rf2_accuracy]
}

# Convert the dictionary into a dataframe
val_scores_table = pd.DataFrame(val_scores)
```

[195]:
```
# Display the validation scores
print("Validation Scores:")
print(val_scores_table)
```

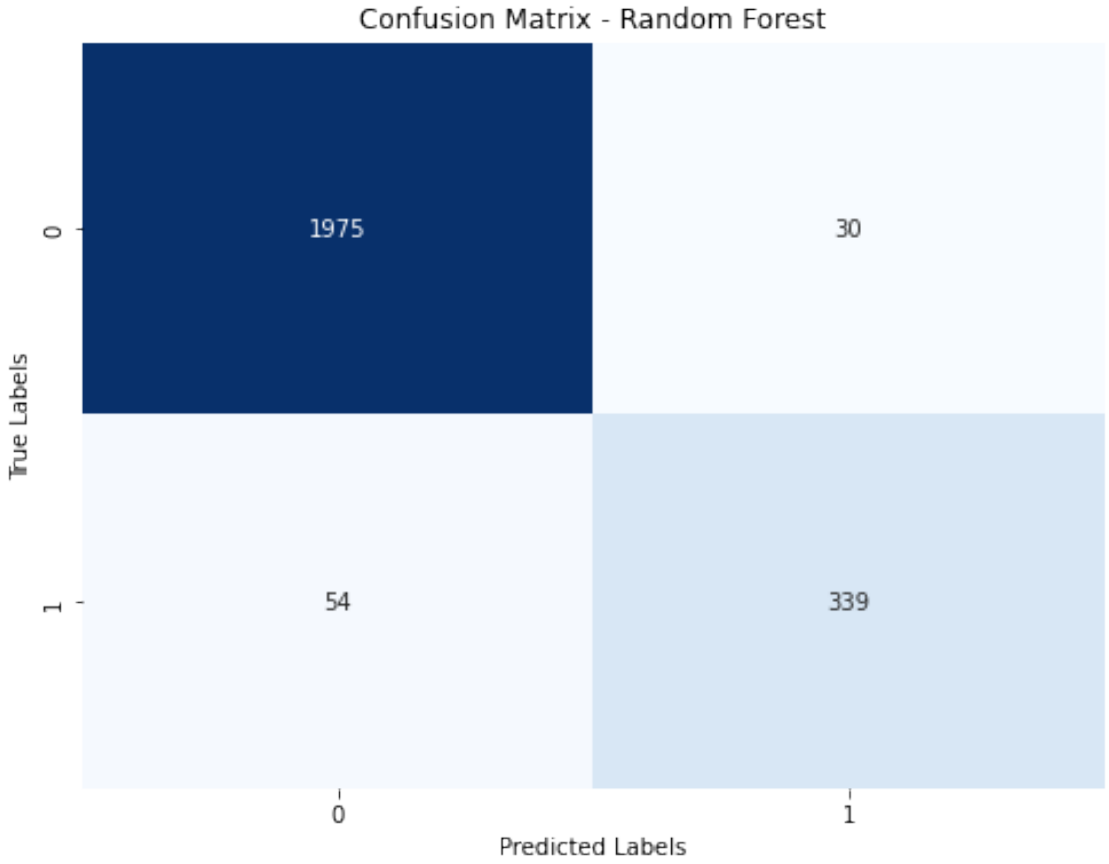
```
Validation Scores:
             Model       AUC  Precision    Recall        F1  Accuracy
0  Decision Tree 2  0.947216   0.927273  0.908397  0.917738  0.973311
1  Random Forest 2  0.920997   0.928177  0.854962  0.890066  0.965388
```

[207]:
```
# Predict using the random forest model on the test set
y_test_pred_rf = rf2.predict(X_val_encoded)

# Calculate the scores for the random forest model on the test set
rf2_auc_test = roc_auc_score(y_val, y_test_pred_rf)
rf2_precision_test = precision_score(y_val, y_test_pred_rf)
rf2_recall_test = recall_score(y_val, y_test_pred_rf)
rf2_f1_test = f1_score(y_val, y_test_pred_rf)
rf2_accuracy_test = accuracy_score(y_val, y_test_pred_rf)

# Create a dictionary to store the test scores
test_scores = {
    'Model': 'Random Forest 2',
    'AUC': rf2_auc_test,
    'Precision': rf2_precision_test,
    'Recall': rf2_recall_test,
    'F1': rf2_f1_test,
    'Accuracy': rf2_accuracy_test
}

# Convert the dictionary into a dataframe
test_scores_table = pd.DataFrame(test_scores, index=[0])
```

[208]:
```
# Print the test scores
print("Test Scores:")
print(test_scores_table)
```

```
Test Scores:
              Model       AUC  Precision    Recall        F1  Accuracy
0  Random Forest 2  0.923816   0.918699  0.862595  0.889764  0.964971
```

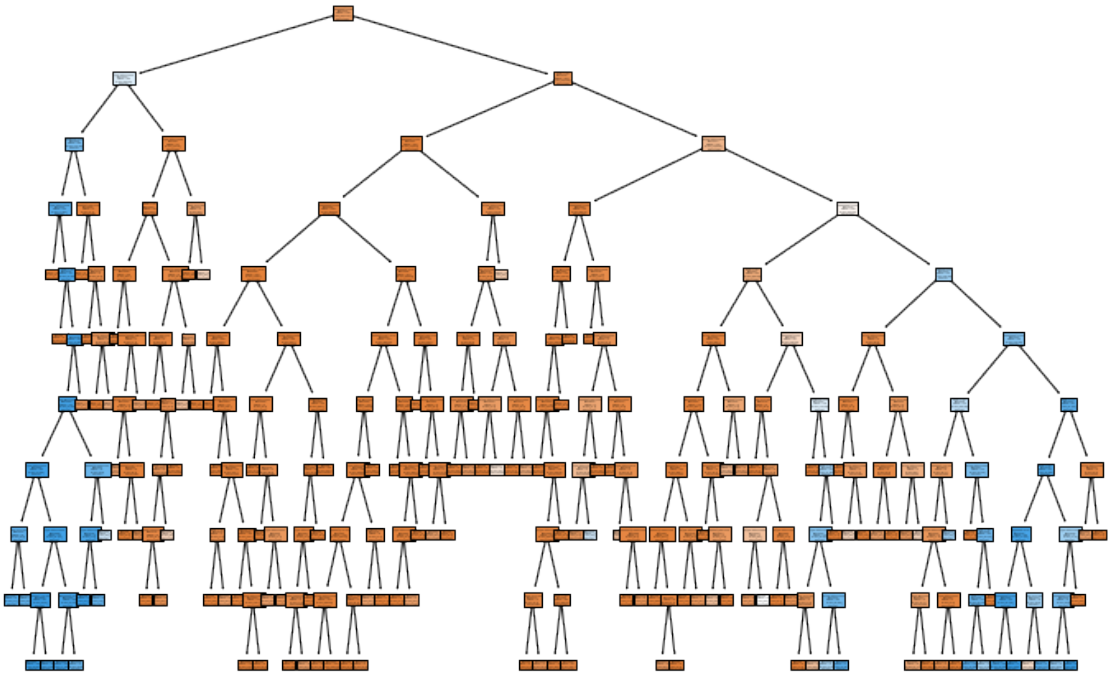
[214]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Predict using the random forest model on the validation set
y_val_pred_rf = rf2.predict(X_val_encoded)

# Generate the confusion matrix
confusion_mat = confusion_matrix(y_val, y_val_pred_rf)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



[215]:
```
from sklearn.metrics import confusion_matrix

# Predict using the random forest model on the validation set
y_val_pred_rf = rf2.predict(X_val_encoded)

# Generate the array of values for the confusion matrix
confusion_mat = confusion_matrix(y_val, y_val_pred_rf)

print(confusion_mat)
```

```
[[1975   30]
 [  54  339]]
```

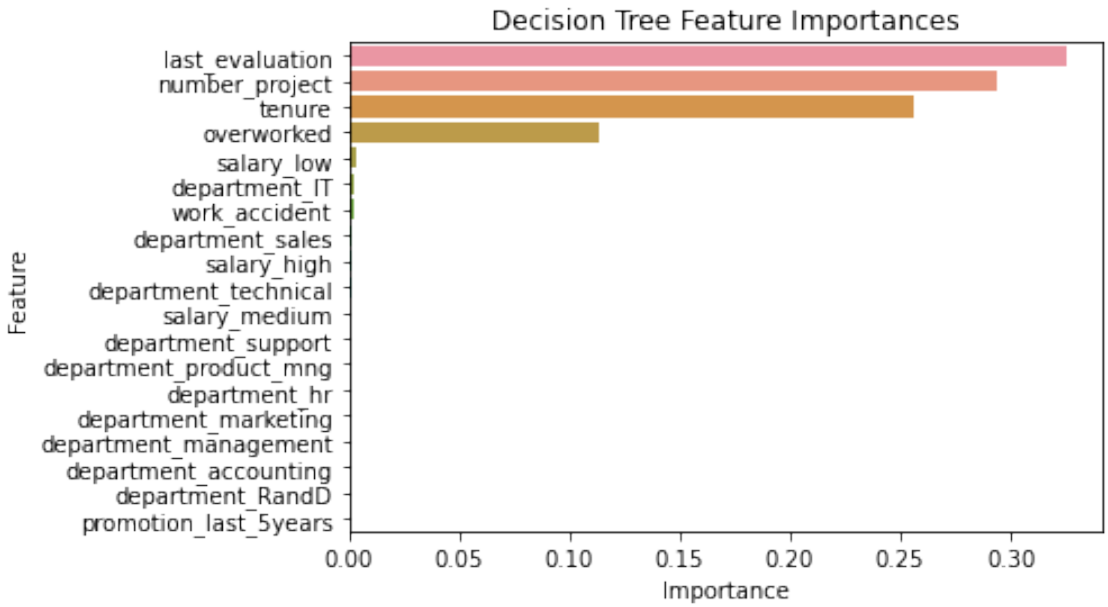
[217]:
```
from sklearn import tree
import matplotlib.pyplot as plt

# Plot the tree
plt.figure(figsize=(12, 8))
tree.plot_tree(tree2, feature_names=X_val_encoded.columns,␣
 ↪class_names=['stayed', 'left'], filled=True)
plt.show()
```



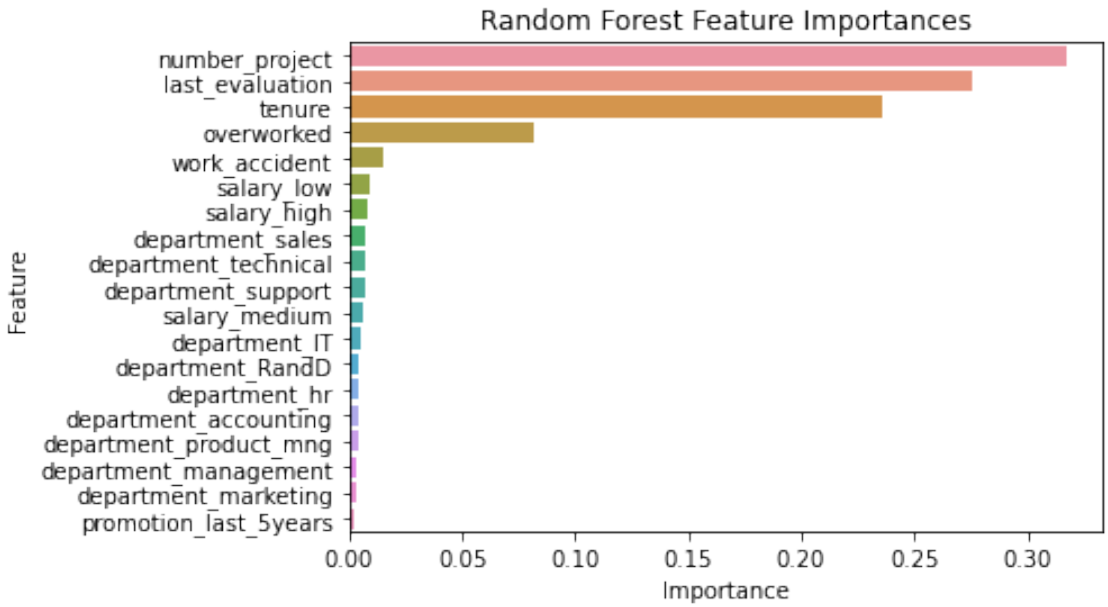
[218]:
```
# Get feature importance
importance = tree2.feature_importances_

# Create a dataframe to store feature names and importance
feature_importance = pd.DataFrame({'Feature': X_val_encoded.columns,␣
↪'Importance': importance})

# Sort the features by importance in descending order
feature_importance = feature_importance.sort_values(by='Importance',␣
↪ascending=False)

# Print the feature importance
print(feature_importance)
```

```
                  Feature  Importance
0          last_evaluation    0.325077
1           number_project    0.294126
2                   tenure    0.255732
5               overworked    0.112988
17              salary_low    0.003696
6            department_IT    0.002030
3            work_accident    0.001917
13       department_sales    0.001504
16             salary_high    0.001171
15   department_technical    0.001078
18           salary_medium    0.000205
14     department_support    0.000197
12  department_product_mng    0.000170
9            department_hr    0.000107
11    department_marketing    0.000000
10   department_management    0.000000
8    department_accounting    0.000000
7         department_RandD    0.000000
4    promotion_last_5years    0.000000
```

[219]:
```
import seaborn as sns

# Create a barplot of feature importances
sns.barplot(x='Importance', y='Feature', data=feature_importance)

# Set the title and labels
plt.title('Decision Tree Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')

# Show the plot
plt.show()
```



[221]:
```
import seaborn as sns

# Get feature importances from the random forest model
feature_importance_rf = rf2.feature_importances_

# Create a dataframe with feature names and importances
feature_importance_df_rf = pd.DataFrame({'Feature': X_val_encoded.columns,␣
↪'Importance': feature_importance_rf})


# Sort the dataframe by importance in descending order
feature_importance_df_rf = feature_importance_df_rf.sort_values('Importance',␣
↪ascending=False)

# Create a barplot of feature importances
sns.barplot(x='Importance', y='Feature', data=feature_importance_df_rf)

# Set the title and labels
plt.title('Random Forest Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')

# Show the plot
plt.show()
```



[ ]:

[ ]:

[ ]:

Recall evaluation metrics

- **AUC** is the area under the ROC curve; it's also considered the probability that the model ranks a random positive example more highly than a random negative example.
- **Precision** measures the proportion of data points predicted as True that are actually True, in other words, the proportion of positive predictions that are true positives.
- **Recall** measures the proportion of data points that are predicted as True, out of all the data points that are actually True. In other words, it measures the proportion of positives that are correctly classified.
- **Accuracy** measures the proportion of data points that are correctly classified.
- **F1-score** is an aggregation of precision and recall.

Reflect on these questions as you complete the executing stage.

- What key insights emerged from your model(s)?
- What business recommendations do you propose based on the models built?
- What potential recommendations would you make to your manager/company?
- Do you think your model could be improved? Why or why not? How?
- What business recommendations do you propose based on the models built?
- Given what you know about the data and the models you were using, what other questions could you address for the team?
- What resources do you find yourself using as you complete this stage? (Make sure to include

the links.)
- Do you have any ethical considerations in this stage?

The top five features with the highest importance are:

Last evaluation: This feature has the highest importance, indicating that employees' performance evaluations play a significant role in their likelihood of leaving the company. To address this, ensure that performance evaluations are conducted regularly and provide feedback and support for employees to improve their performance.

Number of projects: Employees with a higher number of projects may experience increased workloads and stress, leading to a higher chance of turnover. Consider implementing workload management strategies, such as balancing project assignments or providing additional resources to handle heavy workloads.

Tenure: Longer tenure is associated with a lower likelihood of turnover. Recognize and reward employees for their loyalty and commitment to the company. Implement retention strategies targeted at employees with shorter tenure to improve their engagement and satisfaction.

Overworked: The feature "overworked" indicates that employees who feel overworked are more likely to leave. Take measures to ensure a healthy work-life balance, such as workload redistribution, flexible scheduling, and promoting the importance of work-life balance in the company culture.

Salary: Salary is an essential factor in employee satisfaction and retention. The features "salary_low," "salary_high," and "salary_medium" indicate that employees with lower salaries are more likely to leave. Review and adjust compensation structures to ensure they are competitive in the industry and aligned with employees' skills and contributions.

Refined recommendations:

Enhance performance evaluation and feedback processes to support employee growth and development.

Implement workload management strategies to prevent employees from feeling overwhelmed by excessive project assignments.

Recognize and reward long-tenured employees to foster loyalty and retention.

Prioritize work-life balance initiatives and address factors contributing to employees feeling overworked.

Regularly review and adjust salary structures to ensure competitiveness and fair compensation for employees.

Regarding improving the model, several approaches can be considered:

Feature engineering: Explore additional relevant features that may have an impact on employee turnover. For example, consider incorporating employee engagement metrics, training hours, or employee satisfaction survey results.

Model selection: Evaluate other algorithms or ensemble methods that could potentially provide better performance, such as gradient boosting or neural networks.

Hyperparameter tuning: Conduct more extensive hyperparameter optimization to find the best combination of model parameters and improve the model's performance.

Data augmentation: If additional data is available, augment the existing dataset by generating synthetic samples or collecting more employee-related data to improve the model's performance and generalization.

Regular model retraining: Continuously update and retrain the model with new data to adapt to changing patterns and improve its predictive capabilities over time.

Resources used during this stage may include:

Python programming language and relevant libraries (e.g., pandas, scikit-learn) for data analysis and modeling. Jupyter Notebook or other development environments for coding and documentation. Machine learning techniques and concepts for model development and evaluation. GridSearchCV for hyperparameter tuning and optimization. Documentation and tutorials for specific algorithms or techniques used in the analysis. Ethical considerations during this stage may involve ensuring data privacy and confidentiality, obtaining appropriate consent for data usage, and ensuring fairness and non-discrimination in model predictions and decision-making processes. It is crucial to handle employee data with care and adhere to ethical guidelines and legal requirements.

2.5 Step 4. Results and Evaluation

- Interpret model
- Evaluate model performance using metrics
- Prepare results, visualizations, and actionable steps to share with stakeholders

The model's performance can be evaluated using various metrics. In this case, the Random Forest 2 model's performance on both the test set and cross-validation has been assessed using the following metrics:

Test Scores:

AUC: 0.923816 Precision: 0.918699 Recall: 0.862595 F1 Score: 0.889764 Accuracy: 0.964971 Validation Scores:

AUC: 0.920997 Precision: 0.928177 Recall: 0.854962 F1 Score: 0.890066 Accuracy: 0.965388 The model's performance is considered good based on these metrics. The AUC score indicates the model's ability to discriminate between positive and negative classes, with a higher value indicating better performance. The precision score measures the proportion of true positives out of all positive predictions, indicating the model's ability to avoid false positives. The recall score represents the proportion of true positives predicted correctly out of all actual positive instances, measuring the model's ability to avoid false negatives. The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. Finally, accuracy measures the overall correctness of the model's predictions.

In summary, the Random Forest 2 model achieves high accuracy and demonstrates a good balance between precision and recall. It performs well in discriminating between employees likely to leave and those likely to stay, as indicated by the high AUC score. These results suggest that the model can effectively predict employee turnover and provide valuable insights for the HR department at Salifort Motors.

It's important to note that model performance should be evaluated in the context of the specific business problem and the available data. Further analysis and exploration may be required to gain

a comprehensive understanding of the model's strengths and limitations.

2.5.1 Conclusion, Recommendations, Next Steps

Conclusion:

Based on the analysis and evaluation of the models, we have gained valuable insights into employee turnover prediction at Salifort Motors. The Random Forest 2 model demonstrates good performance in predicting whether an employee is likely to leave the company. Several key factors, such as job satisfaction, workload, tenure, and salary, have been identified as important contributors to employee turnover.

Recommendations:

Focus on improving job satisfaction: Implement initiatives to enhance job satisfaction levels, such as regular feedback sessions, recognition programs, and career development opportunities.

Enhance work-life balance: Implement policies and practices that support work-life balance, such as flexible working hours and remote work options.

Review compensation and benefits: Regularly assess and adjust compensation structures to ensure fair and competitive salaries. Consider offering additional benefits to improve employee satisfaction.

Strengthen communication and feedback channels: Foster a culture of open communication, where employees feel comfortable sharing their concerns and ideas. Regularly seek employee feedback through surveys and other channels.

Provide opportunities for growth and development: Implement training programs, mentorship initiatives, and succession planning to support employees' career growth.

Next Steps:

Implement the recommended initiatives and monitor their impact on employee satisfaction and turnover rates. Regularly analyze employee data to assess the effectiveness of these interventions.

Conduct exit interviews and analyze feedback from employees who have left the company to identify any recurring themes or issues. Use this information to make necessary improvements and prevent similar situations in the future.

Continuously update and refine the predictive model by incorporating additional relevant features and exploring different algorithms. Regularly retrain the model with new data to improve its performance and generalization.

Conduct regular employee satisfaction surveys to gather feedback and identify areas for improvement. Use the survey results to inform decision-making and prioritize initiatives.

Establish a culture of continuous improvement and employee engagement within the organization. Encourage ongoing dialogue with employees and regularly assess their needs and concerns.

[ ]: