Activity_ Course 7 Salifort Motors project lab

April 16, 2024

1 Capstone project: Providing data-driven suggestions for HR

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.

2 PACE stages

2.1 Pace: Plan

Consider the questions in your PACE Strategy Document to reflect on the Plan stage.

In this stage, consider the following:

2.1.1 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.

2.1.2 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.]

2.2 Step 1. Imports

- Import packages

- Load dataset

2.2.1 Import packages

```
[1]: # Import packages
     ### YOUR CODE HERE ###
     # Import packages for data manipulation
     import pandas as pd
     import numpy as np
     # Import packages for data visualization
     import matplotlib.pyplot as plt
     import seaborn as sns
     # Import packages for data preprocessing
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
     from sklearn.utils import resample
     # Import packages for data modeling
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report
     from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

2.2.2 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 ###
     df0.head()
```

```
[2]:    satisfaction_level  last_evaluation  number_project  average_montly_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
```

```
   time_spend_company  Work_accident  left  promotion_last_5years Department  \
0                   3              0     1                      0      sales
1                   6              0     1                      0      sales
2                   4              0     1                      0      sales
3                   5              0     1                      0      sales
4                   3              0     1                      0      sales

   salary
0     low
1  medium
2  medium
3     low
4     low
```

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

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

2.3.1 Gather basic information about the data

[16]:
```
# Gather basic information about the data
### YOUR CODE HERE ###
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
```

2.3.2 Gather descriptive statistics about the data

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

```
[4]:        satisfaction_level  last_evaluation  number_project  \
     count        14999.000000     14999.000000    14999.000000
     unique                NaN              NaN             NaN
     top                   NaN              NaN             NaN
     freq                  NaN              NaN             NaN
     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
     unique                  NaN                 NaN            NaN           NaN
     top                     NaN                 NaN            NaN           NaN
     freq                    NaN                 NaN            NaN           NaN
     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 Department salary
     count           14999.000000      14999  14999
     unique                   NaN         10      3
     top                      NaN      sales    low
     freq                     NaN       4140   7316
     mean                0.021268        NaN    NaN
     std                 0.144281        NaN    NaN
     min                 0.000000        NaN    NaN
     25%                 0.000000        NaN    NaN
     50%                 0.000000        NaN    NaN
     75%                 0.000000        NaN    NaN
     max                 1.000000        NaN    NaN
```

2.3.3 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.

[4]:
```
# Display all column names
### YOUR CODE HERE ###
pd.set_option('display.max_columns', None)
columns_list = df0.columns.tolist()
print(columns_list)
```

```
['satisfaction_level', 'last_evaluation', 'number_project',
'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
'promotion_last_5years', 'Department', 'salary']
```

[3]:
```
# Rename columns as needed
### YOUR CODE HERE ###
df0 = df0.rename(columns = {'average_montly_hours': 'average_monthly_hours',␣
 ↪'Department': 'department', 'Work_accident': 'work_accident',␣
 ↪'time_spend_company': 'Tenure'})

# Display all column names after the update
### YOUR CODE HERE ###
columns_list = df0.columns.tolist()
print(columns_list)
```

```
['satisfaction_level', 'last_evaluation', 'number_project',
'average_monthly_hours', 'Tenure', 'work_accident', 'left',
'promotion_last_5years', 'department', 'salary']
```

2.3.4 Check missing values

Check for any missing values in the data.

[6]:
```
# Check for missing values
### YOUR CODE HERE ###
df0.isna().sum()
```

[6]:
```
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
```

2.3.5 Check duplicates

Check for any duplicate entries in the data.

[7]:
```
# Check for duplicates
### YOUR CODE HERE ###
df0.duplicated().sum()
```

[7]:
```
3008
```

[8]:
```
# Inspect some rows containing duplicates as needed
### YOUR CODE HERE ###
df0[df0.duplicated()]
```

[8]:
```
       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
…                     …                …              …
14994                0.40             0.57               2
14995                0.37             0.48               2
14996                0.37             0.53               2
14997                0.11             0.96               6
14998                0.37             0.52               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
…                         …     …             …    …
14994                    151       3              0     1
14995                    160       3              0     1
14996                    143       3              0     1
14997                    280       4              0     1
14998                    158       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
…                      …         …      …
14994                        0     support     low
14995                        0     support     low
14996                        0     support     low
14997                        0     support     low
14998                        0     support     low

[3008 rows x 10 columns]
```

[4]:
```
# Drop duplicates and save resulting dataframe in a new variable as needed
### YOUR CODE HERE ###
df = df0.drop_duplicates()

# Display first few rows of new dataframe as needed
### YOUR CODE HERE ###
df.duplicated().sum()
```

[4]:
```
0
```

[24]:
```
df.head()
```

[24]:
```
   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
```

[25]:
```
df.dtypes
```

[25]:
```
satisfaction_level       float64
last_evaluation          float64
number_project             int64
average_monthly_hours      int64
Tenure                     int64
work_accident              int64
```

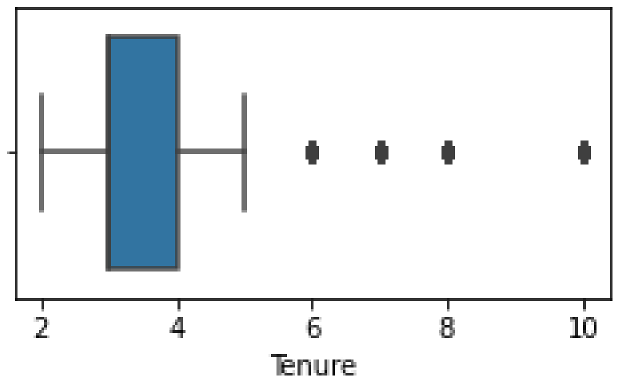
```
left                          int64
promotion_last_5years         int64
department                   object
salary                       object
dtype: object
```

2.3.6 Check outliers

Check for outliers in the data.

[27]:
```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
### YOUR CODE HERE ###
plt.figure(figsize=(4,2))
sns.boxplot(x=df['Tenure'])
plt.plot()
```

[27]:
```
[]
```



[5]:
```
# Determine the number of rows containing outliers
### YOUR CODE HERE ###
percentile25 = df['Tenure'].quantile(0.25)
percentile75 = df['Tenure'].quantile(0.75)
iqr = percentile75 - percentile25
upper_limit = percentile75 + 1.5 * iqr
df.loc[df['Tenure'] > upper_limit, 'Tenure'] = upper_limit
```

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.

3 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?
- What do you observe about the distributions in the data?
- What transformations did you make with your data? Why did you chose to make those decisions?
- What are some purposes of EDA before constructing a predictive 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?

[Double-click to enter your responses here.]

3.1 Step 2. Data Exploration (Continue EDA)

Begin by understanding how many employees left and what percentage of all employees this figure represents.

```
[19]: # Get numbers of people who left vs. stayed
      ### YOUR CODE HERE ###
      df['left'].value_counts(normalize=True)
      # Get percentages of people who left vs. stayed
      ### YOUR CODE HERE ###
```

```
[19]: 0    0.833959
      1    0.166041
      Name: left, dtype: float64
```

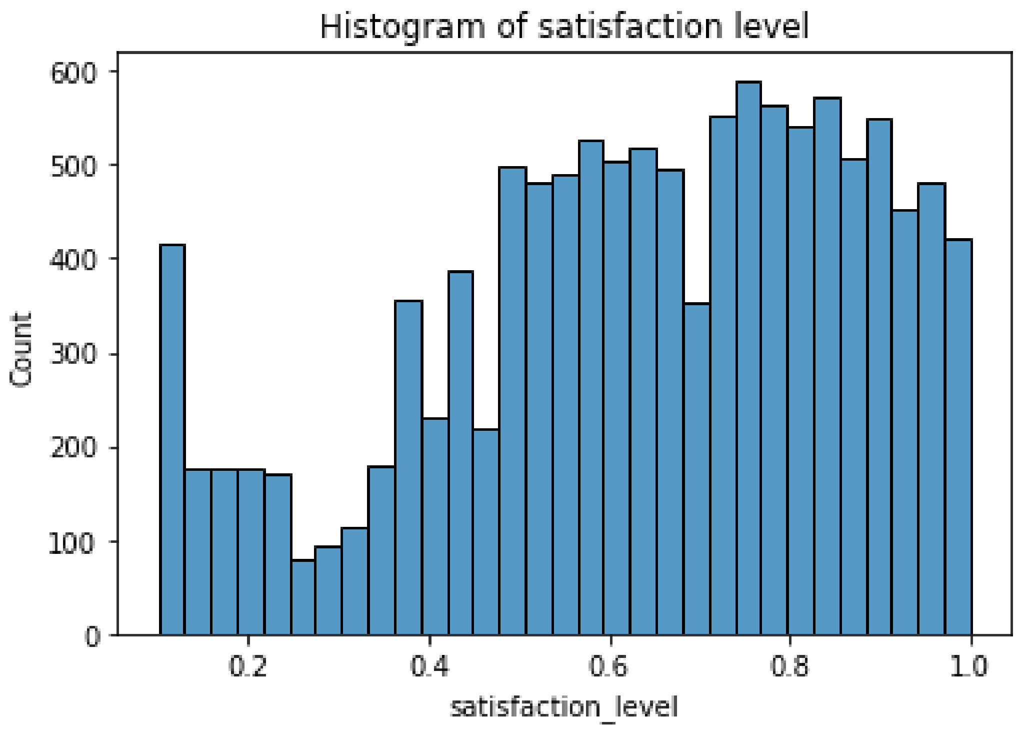
the dataset is imbalanced in terms of people that left and people that did not leave. people that left took up 17% of the whole dataset

3.1.1 Data visualizations

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

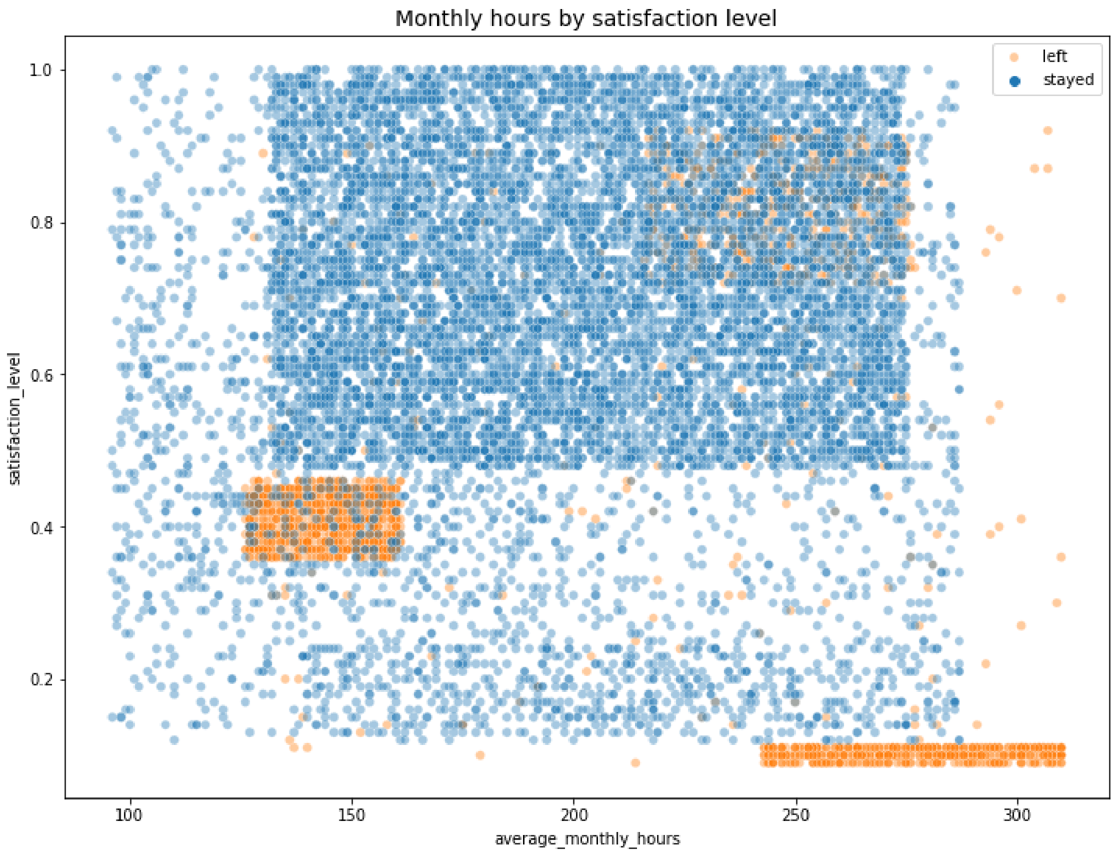
```
[104]: #histogram of satisfaction level
       plt.figure(figsize=(6,4))
       sns.histplot(df['satisfaction_level'], binrange = (0.1,1))
       plt.title('Histogram of satisfaction level')
```

```
[104]: Text(0.5, 1.0, 'Histogram of satisfaction level')
```



The satisfaction level distribution is left-skewed which has a longer tail on the left side, indicating more data points on the right side.

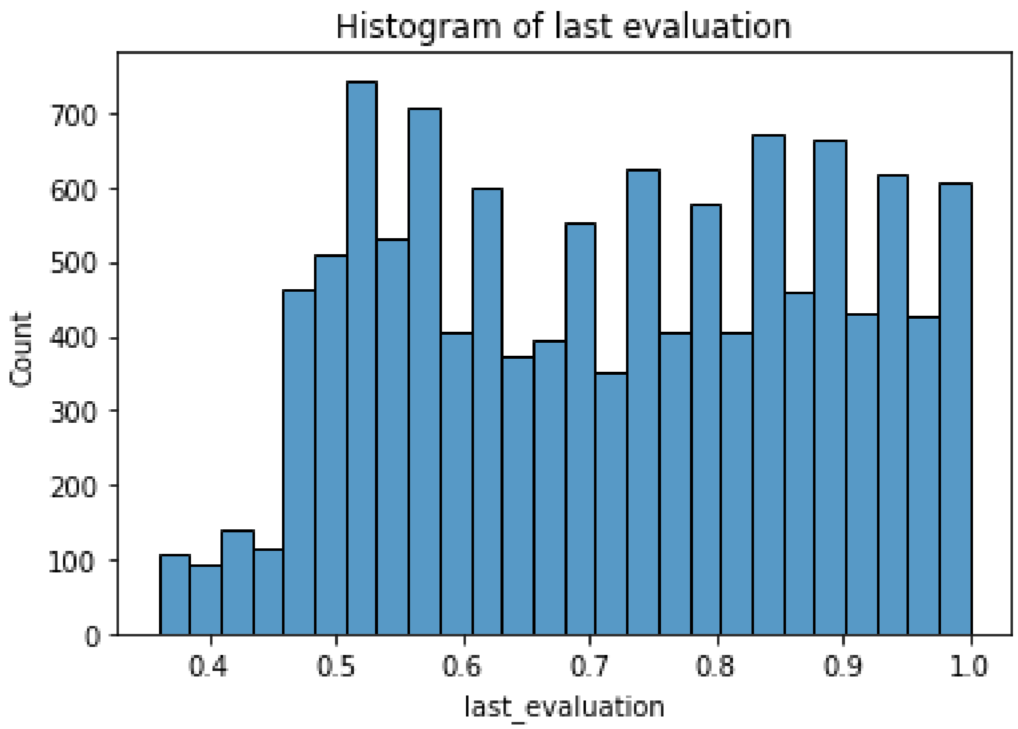
[124]:
```
# Create a plot as needed
### YOUR CODE HERE ###

# Create scatterplot of `average_monthly_hours` versus `satisfaction_level`,␣
↪comparing employees who stayed versus those who left
plt.figure(figsize=(12, 9))
sns.scatterplot(data=df, x='average_monthly_hours', y='satisfaction_level',␣
↪hue='left', alpha=0.4)
plt.legend(labels=['left', 'stayed'])
plt.title('Monthly hours by satisfaction level', fontsize='14');
```



- There are 3 groups of employees that left. The first group, employees that their satisfaction level were nearly 0 worked for long hours (average of 240 to 315 hours monthly). It is quite obvious why their satisfaction level was very low.
- The second group, employees whose satisfaction levels were about 0.4 and worked less hours (130 to 160 monthly) and their reasons for leaving are less clear.
- The third group, some employees with relatively high satisfaction levels and moderate working hours (210-280 monthly) also left.It is important to understand why they left.

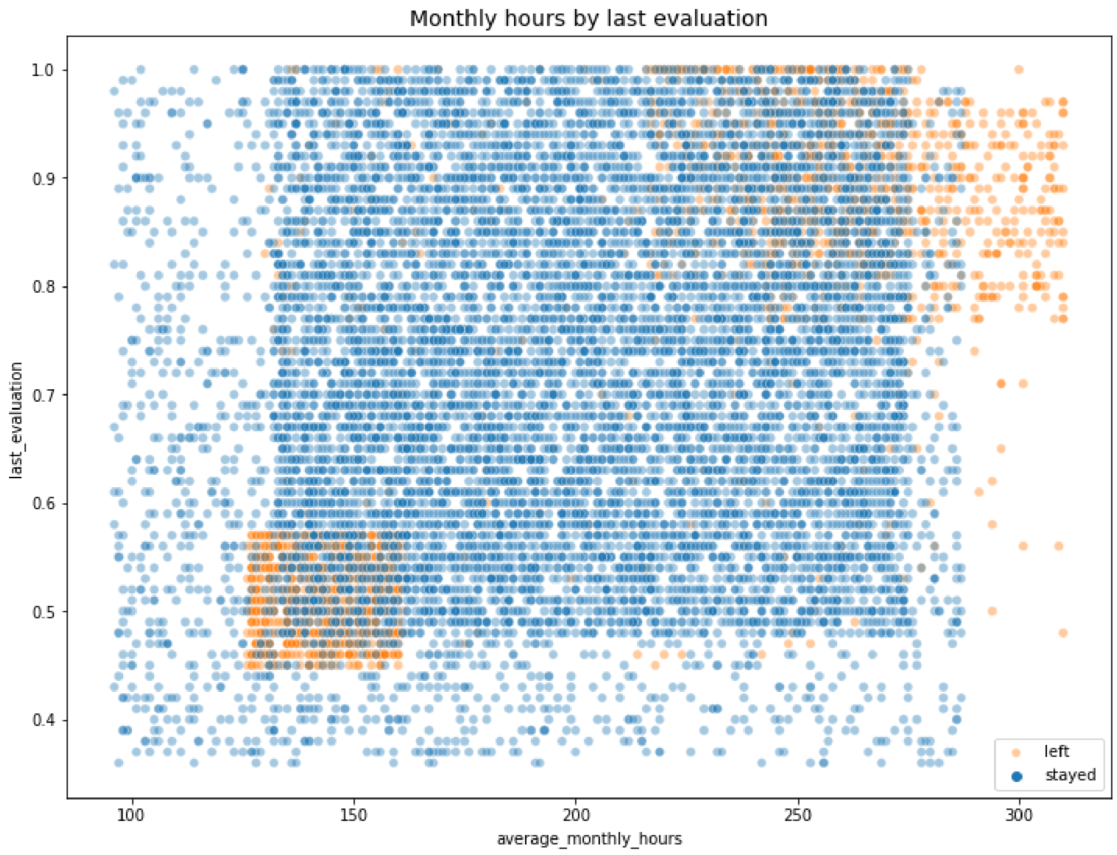
```
[6]: #last evaluation histogram
     plt.figure(figsize=(6,4))
     sns.histplot(df['last_evaluation'])
     plt.title('Histogram of last evaluation')
```

```
[6]: Text(0.5, 1.0, 'Histogram of last evaluation')
```



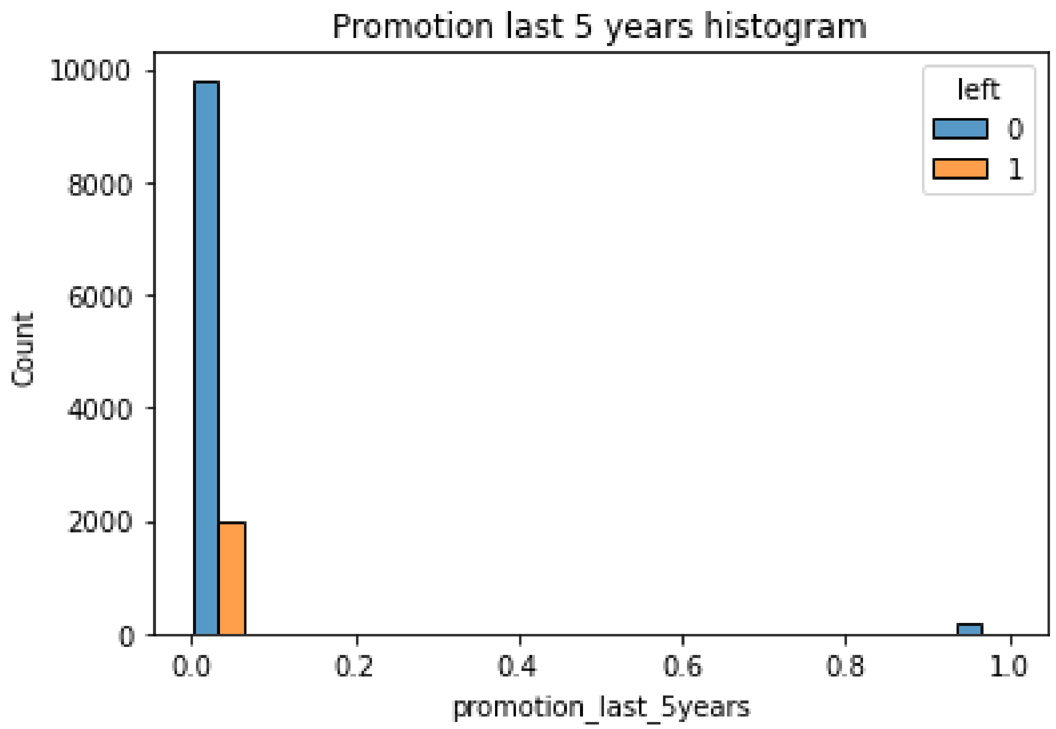
- The last evaluation distribution is uniform, indicating that all data points are evenly distributed.

[123]:
```
# Create a plot as needed
### YOUR CODE HERE ###

# Create scatterplot of `average_monthly_hours` versus `satisfaction_level`,␣
↪comparing employees who stayed versus those who left
plt.figure(figsize=(12, 9))
sns.scatterplot(data=df, x='average_monthly_hours', y='last_evaluation',␣
↪hue='left', alpha=0.4)
plt.legend(labels=['left', 'stayed'])
plt.title('Monthly hours by last evaluation', fontsize='14');
```



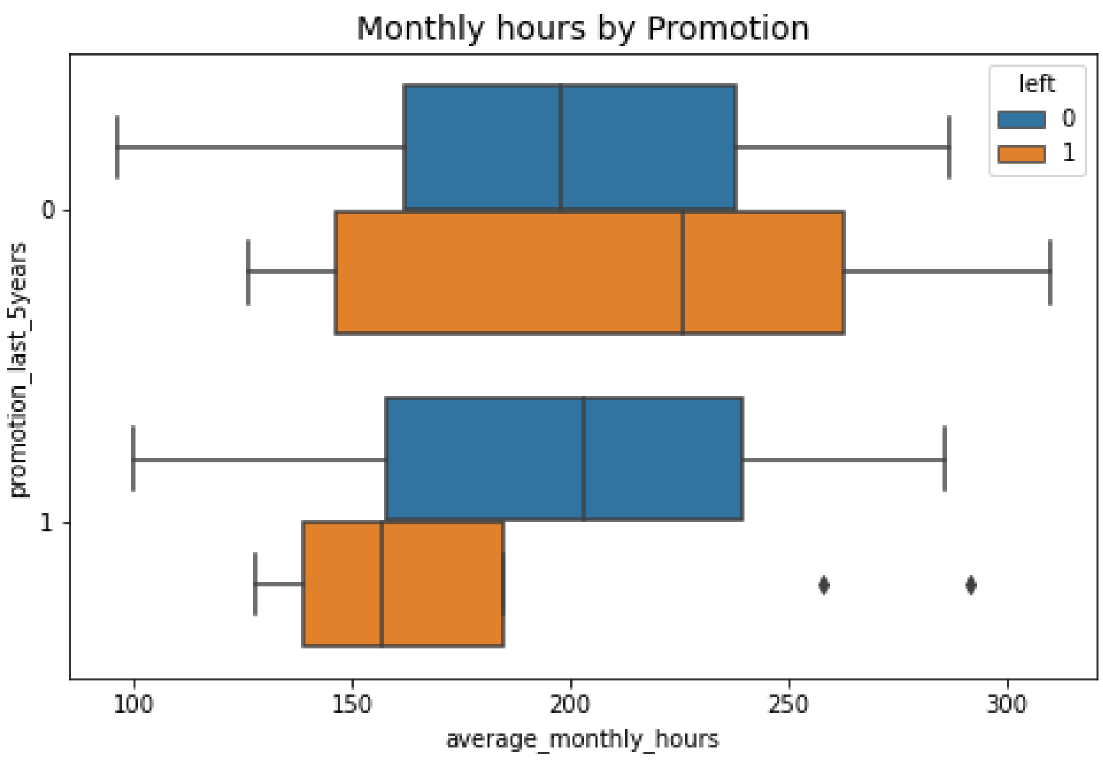
- There are 2 groups of employees that left by last evaluation, the first group worked around 130 to 160 hours but did not perform very well.
- The second group who performed excellently but were overworked.

[100]:
```
#histogram of promotion
plt.figure(figsize=(6,4))
sns.histplot(data=df,
             x='promotion_last_5years',
             hue='left',
             multiple='dodge',
             shrink=0.9)
plt.title('Promotion last 5 years histogram');
```



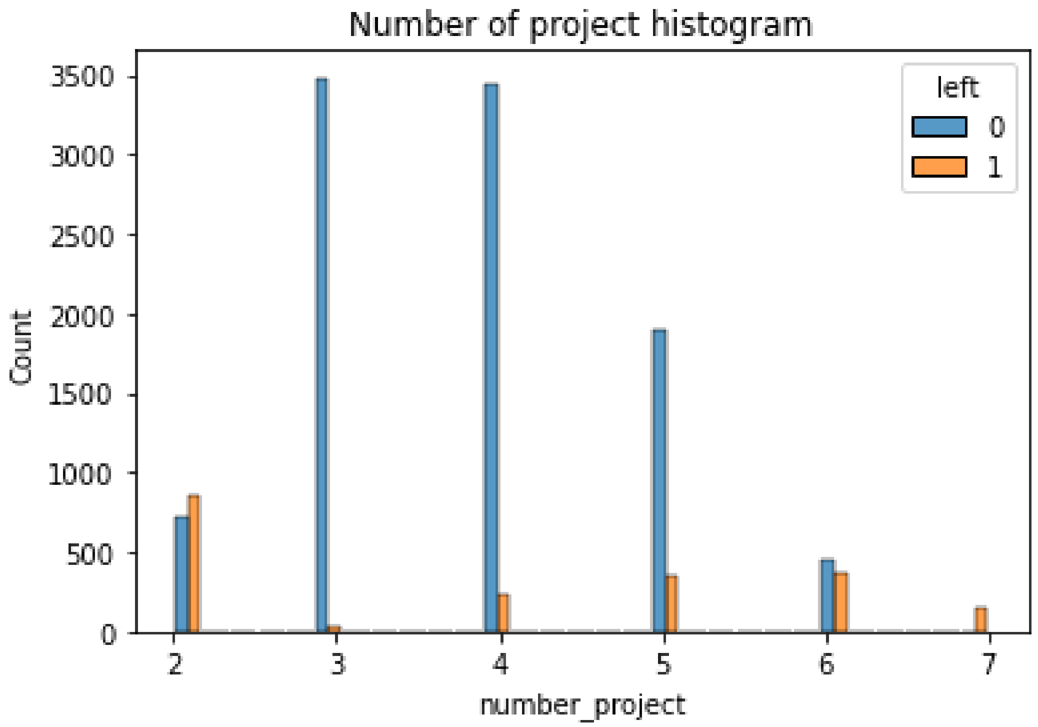
```
[ ]: - A small amount of employees were promoted
```

```
[134]: # Set figure and axes
plt.figure(figsize =(8,5))

# Create boxplot
sns.boxplot(data=df, x='average_monthly_hours', y='promotion_last_5years',␣
 ↪hue='left', orient="h")
tenure_stay = df[df['left']==0]['Tenure']
tenure_left = df[df['left']==1]['Tenure']
plt.title('Monthly hours by Promotion', fontsize='14')

plt.show();
```



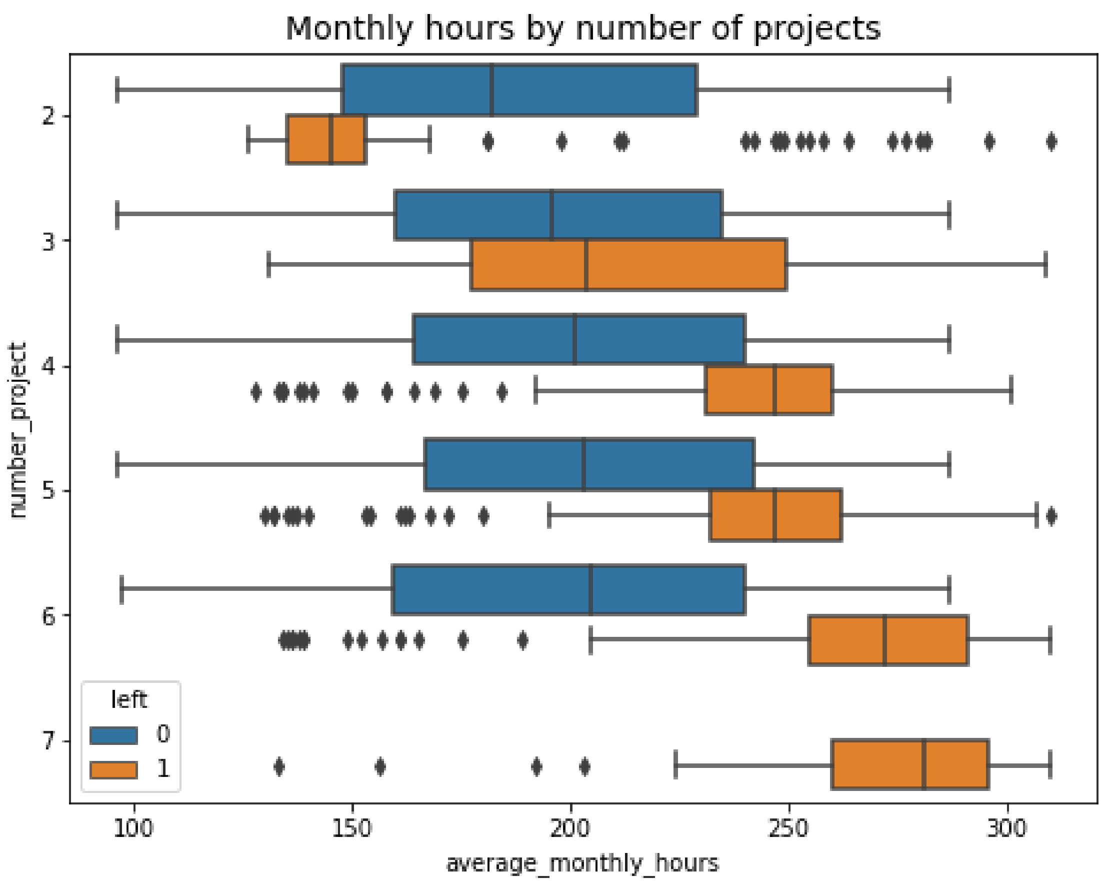
- Employees who worked the most(240+) hours were not promoted
- All employees who were promoted worked less(145-180) to average hours(155-240)

[96]:
```
#number of project histogram
plt.figure(figsize=(6,4))
sns.histplot(data=df,
             x='number_project',
             hue='left',
             multiple='dodge',
             shrink=0.9)
plt.title('Number of project histogram');
```



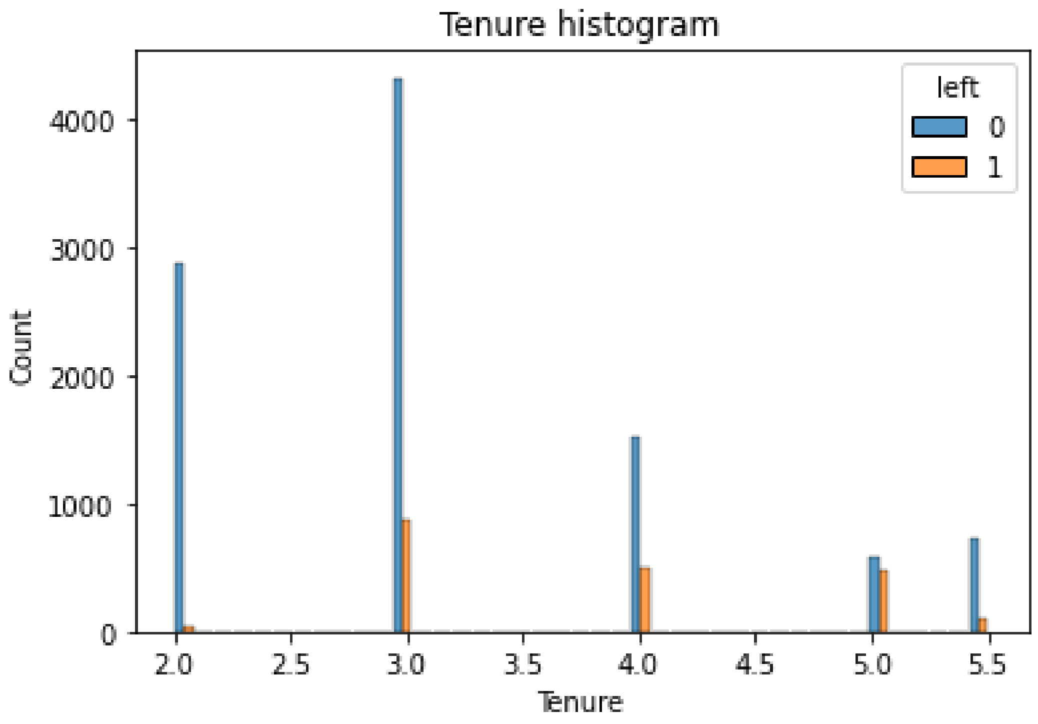
- Employees that stayed are more than people that left in regards to number of project
- The highest number of projects most employees work on are 3 and 4
- The highest number of people that left worked on 2 projects

[95]:
```
# Create a plot as needed
### YOUR CODE HERE ###
plt.figure(figsize = (8,6))
# Set figure and axes
# Create boxplot
sns.boxplot(data=df,  x='average_monthly_hours', y='number_project',␣
 ↪hue='left',orient="h")
plt.title('Monthly hours by number of projects', fontsize='14')
plt.show()
```



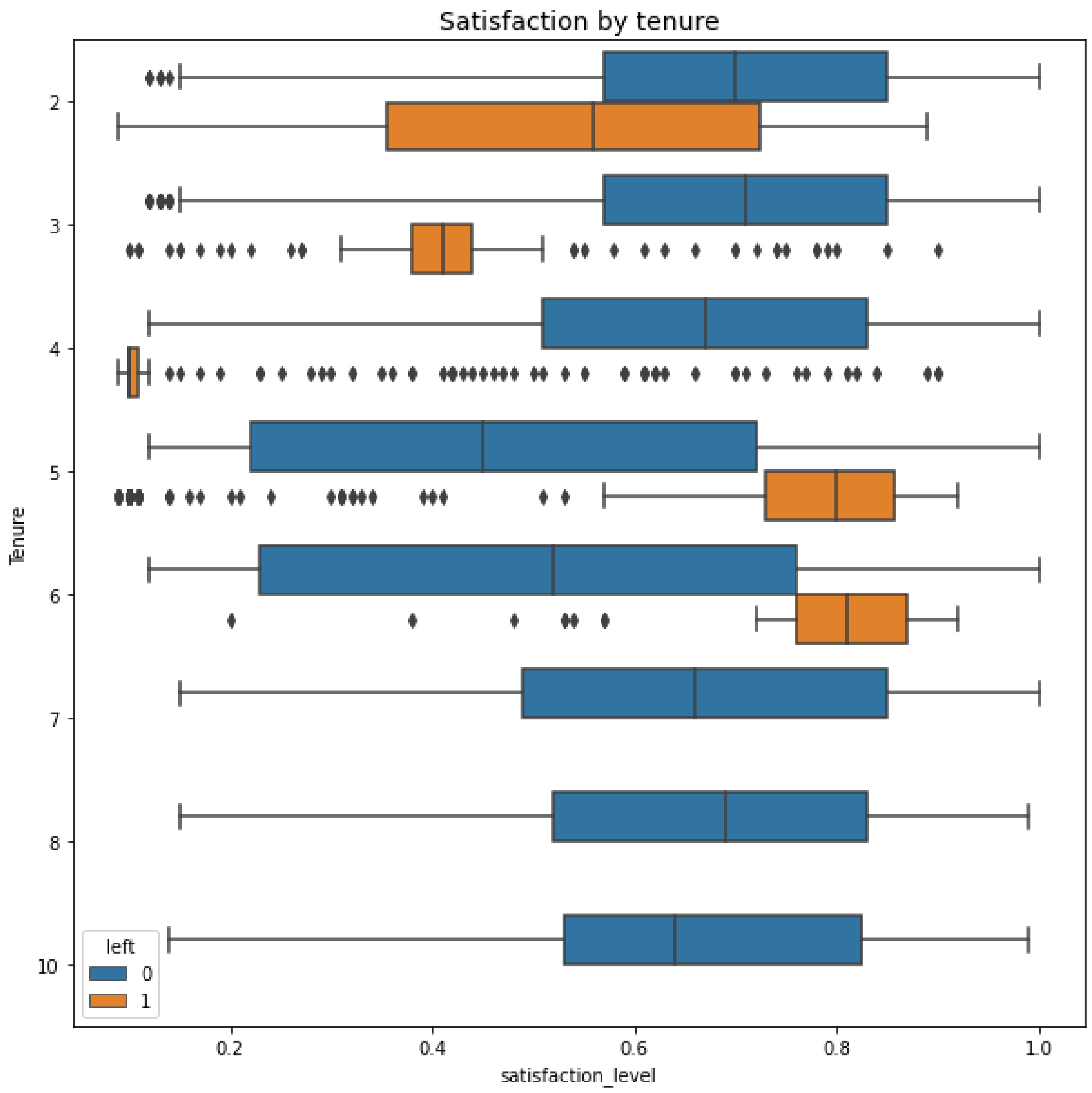
There are two groups of employees that left the company; - Group A: Those who worked on fewer projects and clocked considerably less than the average monthly hours (around 150 hours). - Group B: They were involved in a higher number of projects and worked significantly longer hours, exceeding 250 hours per month. - Those in Group A might have left due to factors like underutilization, lack of challenge, or they were fired. - Group B's heavy workload and long hours indicate a potential risk of burnout. They might have left due to excessive pressure, stress, or feeling overwhelmed.

[94]:
```
#number of project histogram
plt.figure(figsize=(6,4))
sns.histplot(data=df,
             x='Tenure',
             hue='left',
             multiple='dodge',
             shrink=0.9)
plt.title('Tenure histogram');
```



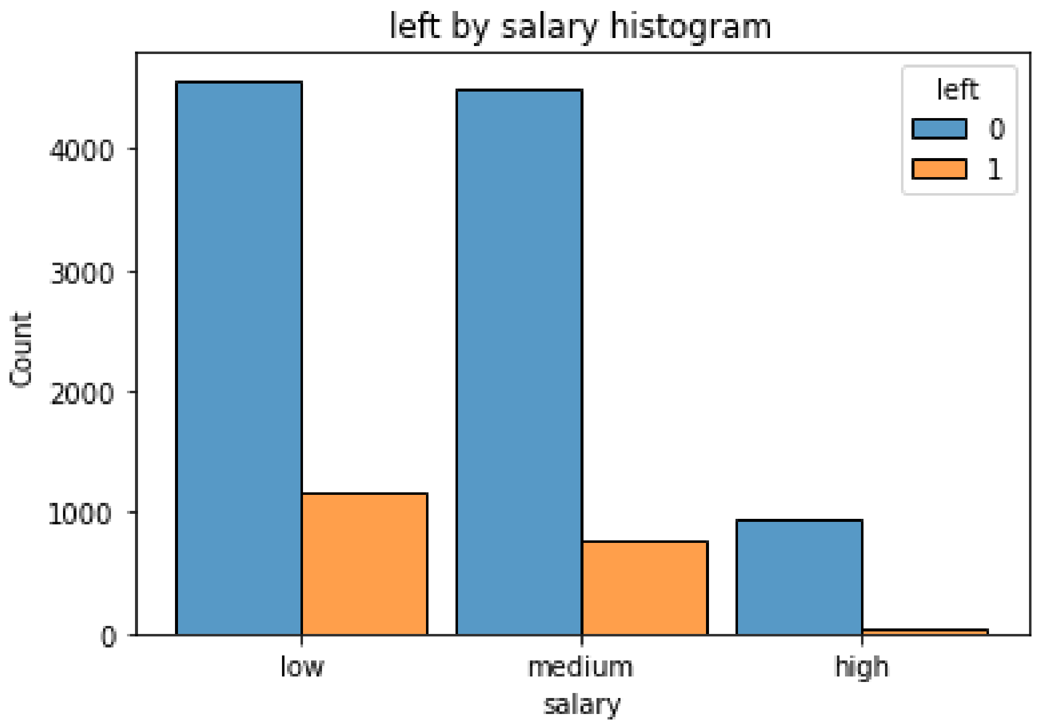
- The highest number of employees that stayed and left are in their third tenure
- There are few longer tenured employees, this could be high ranking employees with high salaries too.

[8]:
```
# Create a plot as needed
### YOUR CODE HERE ###

# Set figure and axes
plt.figure(figsize =(10,10))

# Create boxplot
sns.boxplot(data=df, x='satisfaction_level', y='Tenure', hue='left', orient="h")
plt.title('Satisfaction by tenure', fontsize='14')

plt.show();
```



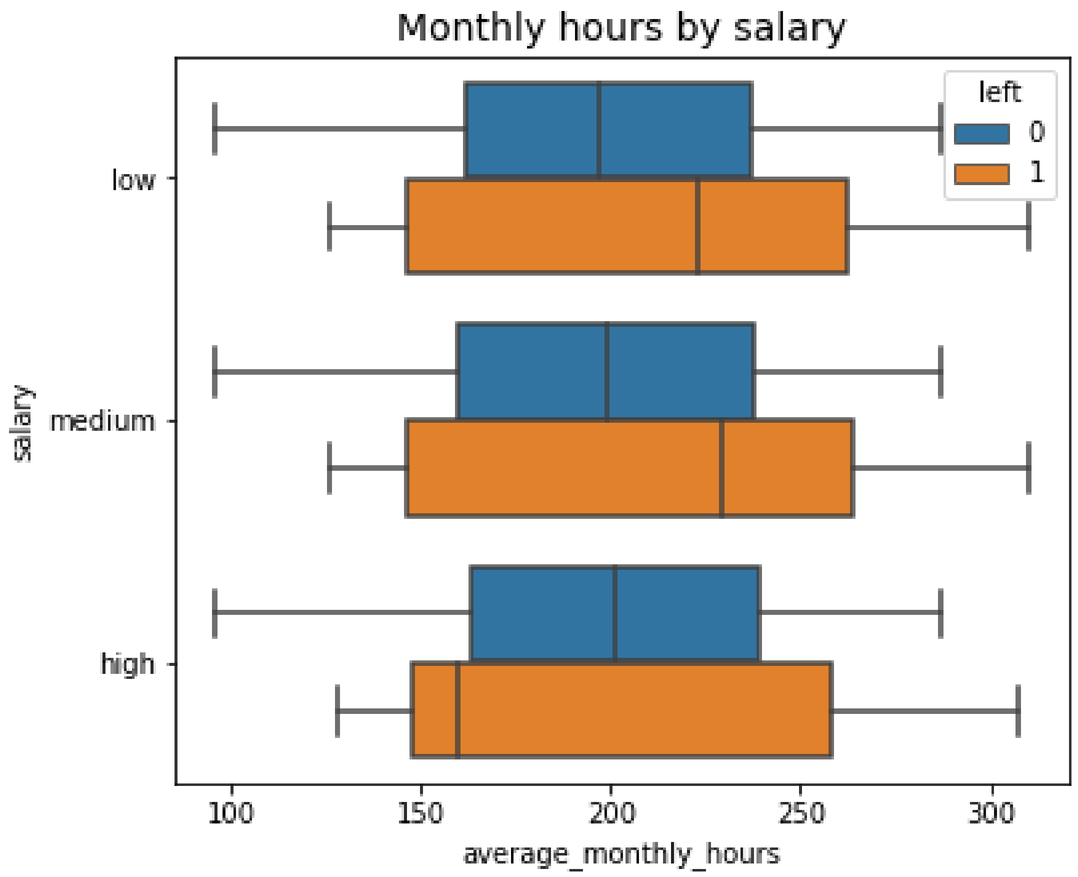
There are many observations in this plot; - Dissatisfied New Hires: the highest number of employees that left fall into the category of dissatisfied individuals with relatively shorter tenures. - 4 years tenured employees who left had seemingly low satisfaction levels. - Retention of Long-Term Employees: The plot shows that most employees with the longest tenures did not leave. Their satisfaction levels appear similar to those of newer employees who chose to stay.

[92]:
```
plt.figure(figsize=(6,4))
sns.histplot(data=df,
             x='salary',
             hue='left',
             multiple='dodge',
             shrink=0.9)
plt.title('left by salary histogram');
```



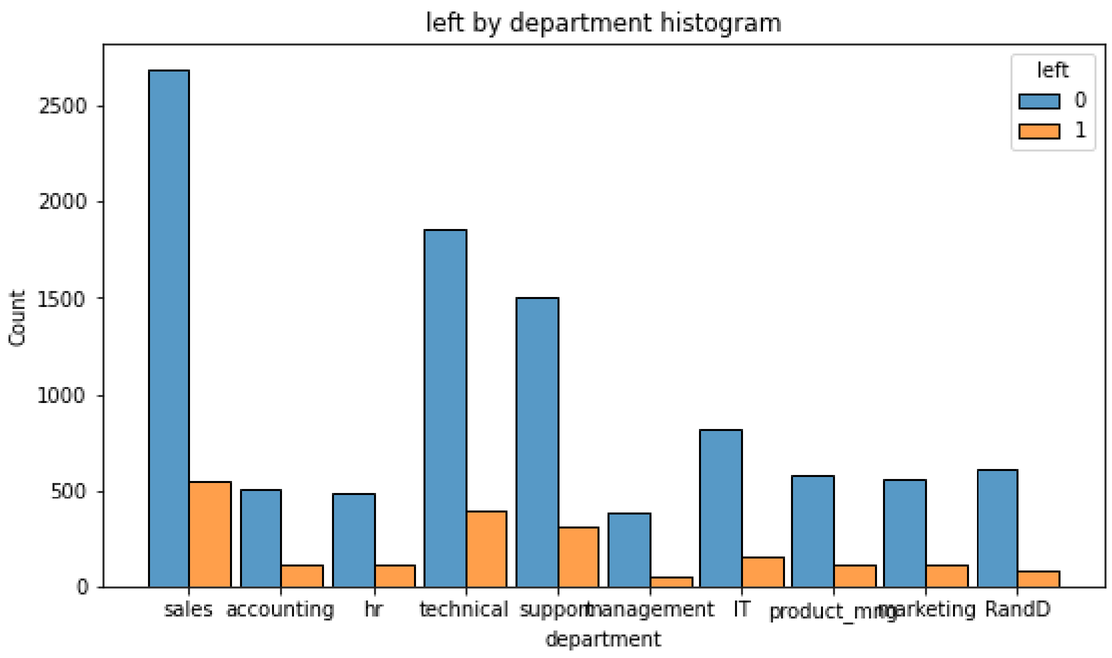
- Employees with low salaries left the most.
- There are few employees with high salaries.

[91]:
```
# Create a plot as needed
### YOUR CODE HERE ###
plt.figure(figsize = (6,5))
# Set figure and axes
# Create boxplot
sns.boxplot(data=df,  x='average_monthly_hours', y='salary',␣
 ↪hue='left',orient="h")
plt.title('Monthly hours by salary', fontsize='14')
plt.show()
```



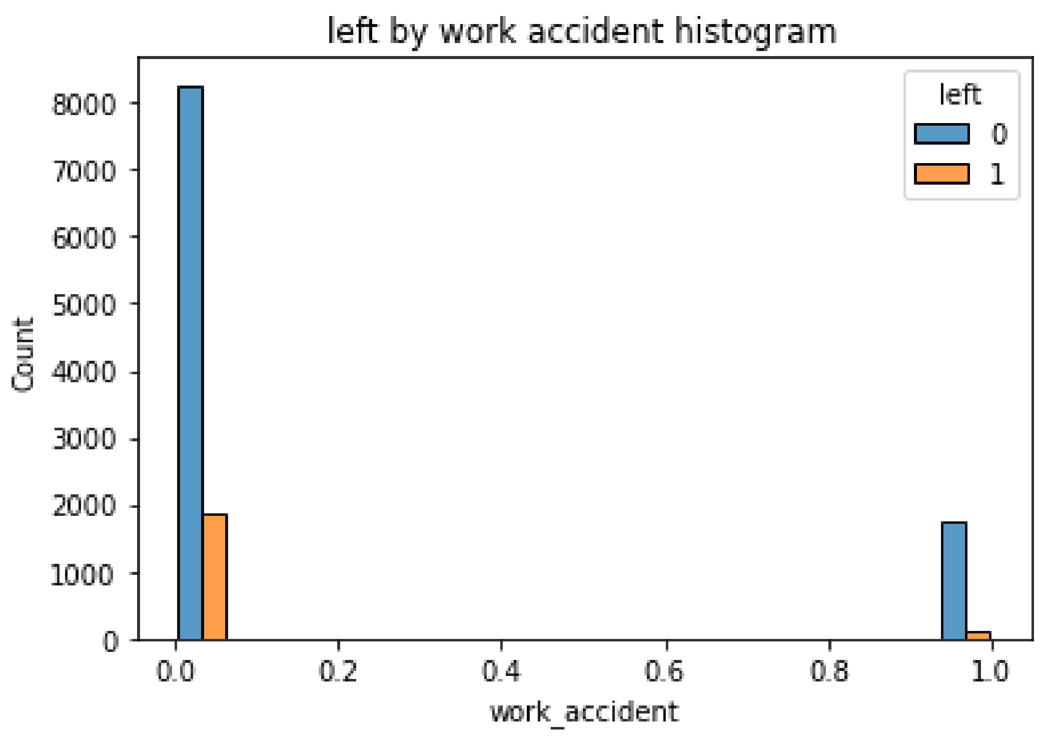
- The amount of hours employees work do not determine the salary category they fall into because employees with high, medium and low salaries fall in the same range of monthly hours.

[85]:
```
plt.figure(figsize=(9,5))
sns.histplot(data=df,
             x='department',
             hue='left',
             multiple='dodge',
             shrink=0.9)
plt.title('left by department histogram');
```



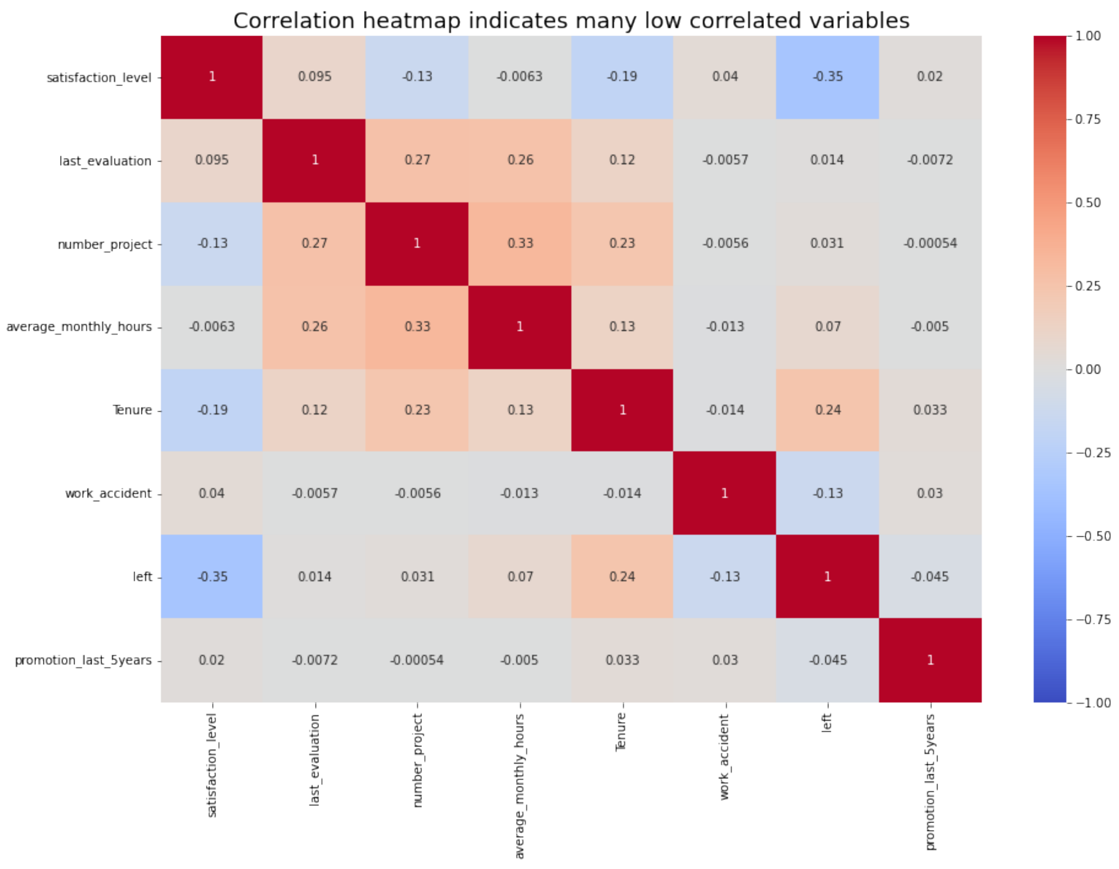
- Employees in sales department left the most followed by the technical department.

[88]:
```
plt.figure(figsize=(6,4))
sns.histplot(data=df,
             x='work_accident',
             hue='left',
             multiple='dodge',
             shrink=0.9)
plt.title('left by work accident histogram');
```



```
[ ]: - A small amount of people left by work accident
```

```
[87]: # Create a plot as needed
      ### YOUR CODE HERE ###
      # Plot correlation heatmap
      plt.figure(figsize=(15,10))
      sns.heatmap(df.corr(method='pearson'), vmin=-1, vmax=1, annot=True,␣
       ↪cmap='coolwarm')
      plt.title('Correlation heatmap indicates many low correlated variables',
                fontsize=18)
      plt.show();
```



	satisfaction_level	last_evaluation	number_project	average_monthly_hours	Tenure	work_accident	left	promotion_last_5years
satisfaction_level	1	0.095	-0.13	-0.0063	-0.19	0.04	-0.35	0.02
last_evaluation	0.095	1	0.27	0.26	0.12	-0.0057	0.014	-0.0072
number_project	-0.13	0.27	1	0.33	0.23	-0.0056	0.031	-0.00054
average_monthly_hours	-0.0063	0.26	0.33	1	0.13	-0.013	0.07	-0.005
Tenure	-0.19	0.12	0.23	0.13	1	-0.014	0.24	0.033
work_accident	0.04	-0.0057	-0.0056	-0.013	-0.014	1	-0.13	0.03
left	-0.35	0.014	0.031	0.07	0.24	-0.13	1	-0.045
promotion_last_5years	0.02	-0.0072	-0.00054	-0.005	0.033	0.03	-0.045	1

- The heatmap shows that none of the variables has high multicollinearity
- Last evaluation, number of project, average monthly hours are positively correlated with each other. They also have postive correlation with left.
- Satisfaction level and promotion are negatively correlated with left.

3.1.2 Insights

It appears that employees are leaving the company as a result of poor management. Leaving is tied to longer working hours, many projects, and generally lower satisfaction levels. It can be ungratifying to work long hours and not receive promotions or good evaluation scores. There's a sizeable group of employees at this company who are probably burned out. It also appears that if an employee has spent more than six years at the company, they tend not to leave.

4 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?

[Double-click to enter your responses here.]

4.1 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

4.1.1 Identify the type of prediction task.

categorical prediction task - to find out why employees are leaving and to predict whether they will leave or not

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

logistic regression and random forest model and xgboost model, i will cross validate too

4.1.3 Modeling

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

Ethical considerations

1. What are you being asked to do? Business need and modeling objective

Currently, there is a high rate of turnover among Salifort employees. (Note: In this context, turnover data includes both employees who choose to quit their job and employees who are let go). Salifort's senior leadership team is concerned about how many employees are leaving the company. Salifort strives to create a corporate culture that supports employee success and professional development. Further, the high turnover rate is costly in the financial sense. Salifort makes a big investment in recruiting, training, and upskilling its employees.

If Salifort could predict whether an employee will leave the company, and discover the reasons behind their departure, they could better understand the problem and develop a solution. A machine learning model that predicts whether an employee will leave the company based on their job title, department, number of projects, average monthly hours, and any other relevant data points. A good model will help the company increase retention and job satisfaction for current employees, and save money and time training new employees.

Modeling design and target variable

The data dictionary shows that there is a column called left. This is a binary value that indicates whether an employee left the company or not. This will be the target variable. In other words, for employee, the model should predict whether an employee left or not.

This is a classification task because the model is predicting a binary class.

Select an evaluation metric

To determine which evaluation metric might be best, consider how the model might be wrong. There are two possibilities for bad predictions:

False positives: When the model predicts an employee left when in fact the employee stayed False negatives: When the model predicts an employee stayed when in fact the employee left

2. What are the ethical implications of building the model? False positives are worse for the company, because Time and money spent trying to retain employees who were likely to stay anyway could be better invested in other areas and Employees may feel pressured or micromanaged if they are wrongly identified as flight risks. False negatives are worse for the company too, because it can result in higher turnover rates, impacting productivity, knowledge retention, and recruitment costs and if signs of dissatisfaction are missed, you lose the chance to address concerns and potentially prevent departures.

The stakes are relatively even. You want to help reduce turnover rates and time and money spent trying to retain customers F1 score is the metric that places equal weight on true postives and false positives, and so therefore on precision and recall.

3. How would you proceed? Modeling workflow and model selection process

Previous work with this data has revealed that there are ~10,000 employees in the sample. This is sufficient to conduct a rigorous model validation workflow, broken into the following steps:

- Split the data into train/validation/test sets (60/20/20)
- Fit models and tune hyperparameters on the training set
- Perform final model selection on the validation set
- Assess the champion model's performance on the test set

[31]:
```
### YOUR CODE HERE ###
X = df.copy()
# Drop unnecessary columns
X = X.drop(['left'], axis=1)
# Encode the `salary` column as an ordinal numeric category
X['salary'] = (
    X['salary'].astype('category')
    .cat.set_categories(['low', 'medium', 'high'])
    .cat.codes
)

# Dummy encode the `department` column
X = pd.get_dummies(X, drop_first=False)

# Display the new dataframe
X.head()
```

[31]:
```
   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  salary  department_IT  \
0     3.0              0                      0       0              0
1     5.5              0                      0       1              0
2     4.0              0                      0       1              0
3     5.0              0                      0       0              0
4     3.0              0                      0       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
0                 1                   0                     0
1                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
```

[32]:
```
# Isolate target variable
y = df['left']
y.head()
```

[32]:
```
0    1
1    1
2    1
3    1
4    1
Name: left, dtype: int64
```

[33]:
```
X.dtypes
```

[33]:
```
satisfaction_level        float64
last_evaluation           float64
number_project              int64
average_monthly_hours       int64
Tenure                    float64
work_accident               int64
promotion_last_5years       int64
salary                       int8
department_IT               uint8
department_RandD            uint8
department_accounting       uint8
department_hr               uint8
department_management       uint8
department_marketing        uint8
department_product_mng      uint8
department_sales            uint8
department_support          uint8
department_technical        uint8
dtype: object
```

[34]:
```
#split your data
# Split dataset into training and holdout datasets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25,␣
↪stratify=y, random_state=0)
```

[35]:
```
#get shape of the data
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[35]:
```
((8993, 18), (2998, 18), (8993,), (2998,))
```

[36]:
```
#build the model
model = LogisticRegression(penalty='none', max_iter=500)
model.fit(X_train, y_train)
```

[36]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=500,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                              warm_start=False)
```

[37]:
```
#call the model.coef to output all the coefficients
pd.Series(model.coef_[0], index=X.columns)
```

[37]:
```
satisfaction_level       -3.874511
last_evaluation           0.421774
number_project           -0.335704
average_monthly_hours     0.003772
Tenure                    0.608700
work_accident            -1.610913
promotion_last_5years    -1.242043
salary                   -0.634687
department_IT            -0.050649
department_RandD         -0.369944
department_accounting    -0.066348
department_hr            -0.104877
department_management    -0.292920
department_marketing      0.053521
department_product_mng   -0.082384
department_sales         -0.004677
department_support        0.043382
department_technical      0.032705
dtype: float64
```

5 Interpretation

Based on the logistic regression model, a one unit increase in satisfaction level is associated with a 3.8 decrease in the log odds of employees leaving. To interpret this in a more explanatory way, we will get the exponention of -3.87 and use 1 to minus it eg e-3.87 = 0.02, 1-0.02 = 50 = 0.5%
we can then say, for every one unit increase in satisfaction level, we wxpect that the odds the person will leave decreases by 0.5% 2. for every one unit increase in the evaluation score, we expect that the odds the person will leave increases by 52% 3. for every one unit increase in the average monthly hours, we expect that the odds the person will leave increases by 100.37% 4. for every one unit increase in tenure, we expect that the odds the person will leave increases by 182%

[38]:
```
model.intercept_
```

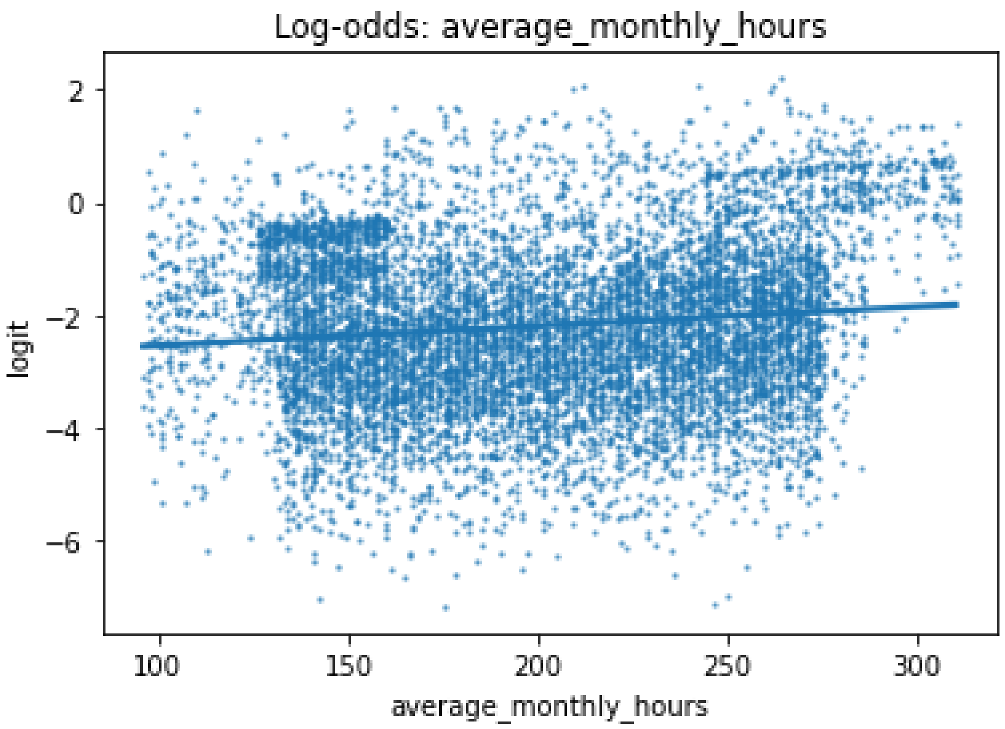
[38]:
```
array([-0.84218998])
```

[39]:
```
# Get the predicted probabilities of the training data
training_probabilities = model.predict_proba(X_train)
training_probabilities
```

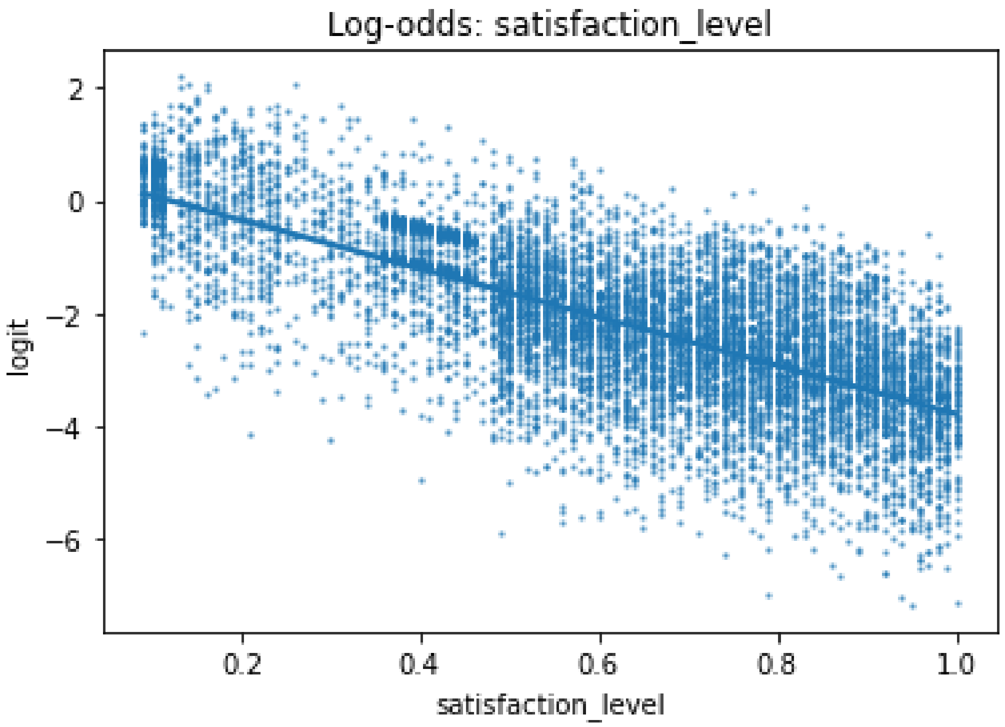
```
[39]: array([[0.97122564, 0.02877436],
             [0.83227563, 0.16772437],
             [0.22276673, 0.77723327],
             …,
             [0.92007255, 0.07992745],
             [0.96236292, 0.03763708],
             [0.8932481 , 0.1067519 ]])
```

```
[40]: # 1. Copy the `X_train` dataframe and assign to `logit_data`
      logit_data = X_train.copy()
      # 2. Create a new `logit` column in the `logit_data` df
      logit_data['logit'] = [np.log(prob[1] / prob[0]) for prob in␣
       ↪training_probabilities]
```

```
[41]: sns.regplot(x='average_monthly_hours', y='logit', data=logit_data,␣
       ↪scatter_kws={'s': 2,'alpha': 0.5})
      plt.title('Log-odds: average_monthly_hours');
```



```
[42]: sns.regplot(x='satisfaction_level', y='logit', data=logit_data,␣
       ↪scatter_kws={'s': 2,'alpha': 0.5})
      plt.title('Log-odds: satisfaction_level');
```

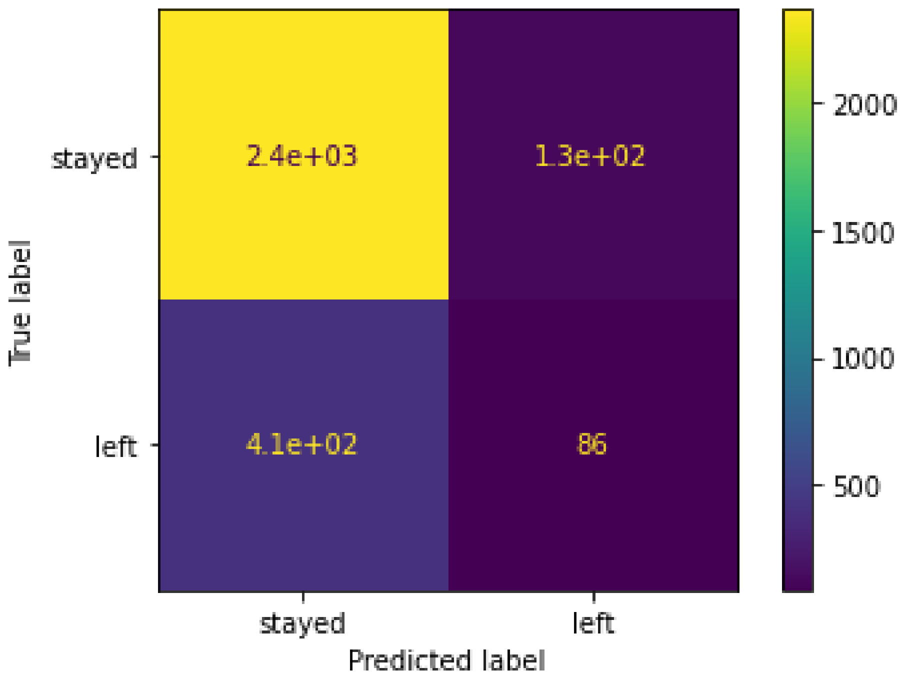


[43]:
```
#use your model to predict on the test data (unseen data to understand how the␣
↪data will perfrom on unseen data or data it hasnt experienced)
y_preds = model.predict(X_test)
```

[44]:
```
#Score the model (accuracy) on the test data
model.score(X_test, y_test)
```

[44]:
```
0.8178785857238159
```

[45]:
```
cm = confusion_matrix(y_test, y_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['stayed', 'left'],
)
disp.plot();
```



the true negatives are 720, that means the model predicted accurately that 720 did not leave the company . the true positives are 74, which means the model accurately predicted that 74 people left the company. the false positives are 240 which means 240 people did not leave the company but the model inaccurately predicted that they left. the false negatives are 800, which means 800 employees actually left the company but the model inaccurately predicted that they did not leave. the model is weak at predicting true positives

[46]:
```python
# Create a classification report
target_labels = ['stayed', 'left']
print(classification_report(y_test, y_preds, target_names=target_labels))
```

```
              precision    recall  f1-score   support

      stayed       0.85      0.95      0.90      2500
        left       0.39      0.17      0.24       498

    accuracy                           0.82      2998
   macro avg       0.62      0.56      0.57      2998
weighted avg       0.78      0.82      0.79      2998
```

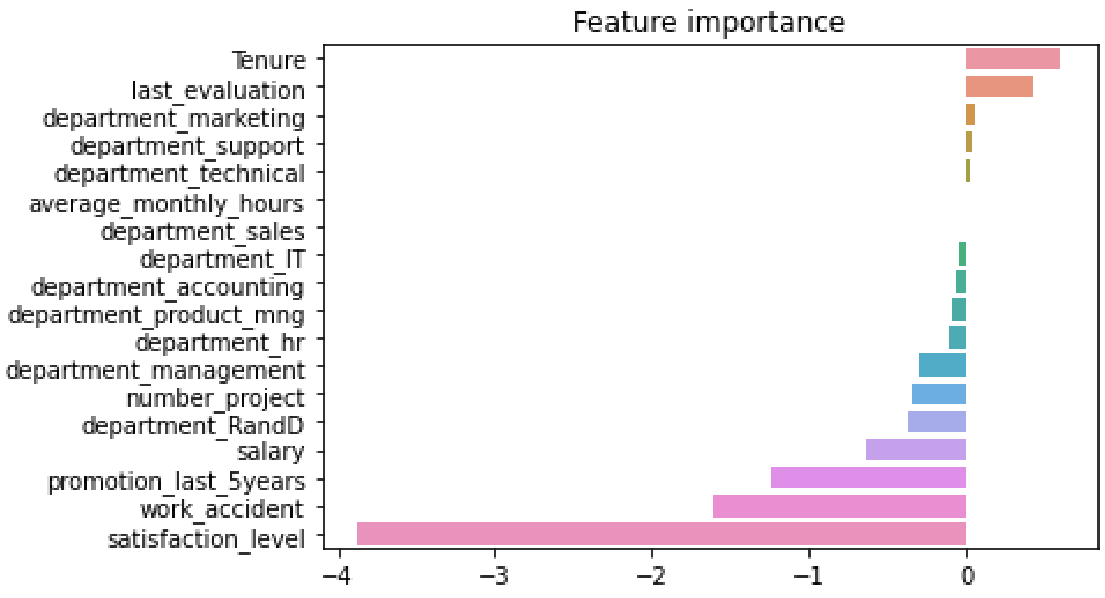
The model is 82% accurate. the precision score is 39% meaning that the model was poor at predicting false positives. the model was also bad at predicting false negatives as the recall score was 16%, the harmonic mean of the precision and recall score is 22%

[47]:
```
#Create a list of (column_name, coefficient) tuples
feature_importance = list(zip(X_train.columns, model.coef_[0]))
# Sort the list by coefficient value
feature_importance = sorted(feature_importance, key=lambda x: x[1],reverse=True)
feature_importance
```

[47]:
```
[('Tenure', 0.6086995595620502),
 ('last_evaluation', 0.42177378799300375),
 ('department_marketing', 0.05352078310530361),
 ('department_support', 0.04338220176376996),
 ('department_technical', 0.032705078644684756),
 ('average_monthly_hours', 0.0037722891620177563),
 ('department_sales', -0.004676692209398138),
 ('department_IT', -0.0506489430516253),
 ('department_accounting', -0.06634838090679528),
 ('department_product_mng', -0.08238355433508354),
 ('department_hr', -0.10487707065984019),
 ('department_management', -0.29291970951930224),
 ('number_project', -0.33570395966633804),
 ('department_RandD', -0.3699436933354437),
 ('salary', -0.634686830345437),
 ('promotion_last_5years', -1.242042704050925),
 ('work_accident', -1.6109125200781593),
 ('satisfaction_level', -3.874511441868459)]
```

In the logistic regression, tenure was the most important feature that was used in prediction followed by last evaluation and the departments

[48]:
```
# Plot the feature importances
import seaborn as sns
sns.barplot(x=[x[1] for x in feature_importance],
y=[x[0] for x in feature_importance],
orient='h')
plt.title('Feature importance');
```



6 TREE BASED MODEL

[64]:
```
### YOUR CODE HERE ###
# Important imports for modeling and evaluation
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import sklearn.metrics as metrics
from sklearn.metrics import roc_auc_score, roc_curve, auc
```

[50]:
```
#isolated x and y variables
y = df['left']
#copy the data
X = df.copy()
#isolate x varaibles
X = X.drop("left", axis = 1)
#encode salary feature as an ordinal catergorical variable
X['salary'] = (
    X['salary'].astype('category')
    .cat.set_categories(['low', 'medium', 'high'])
    .cat.codes
)

# Dummy encode the `department` column
X = pd.get_dummies(X, drop_first=False)
```

```
#split the data in train and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=0)
```

[51]:
```
#print the size of the train and test dataset
for x in [X_train, X_test]:
    print(len(x))
```

```
8993
2998
```

[52]:
```
#build the decision tree model
decision_tree = DecisionTreeClassifier(random_state=0)

decision_tree.fit(X_train, y_train)

dt_pred = decision_tree.predict(X_test)
```

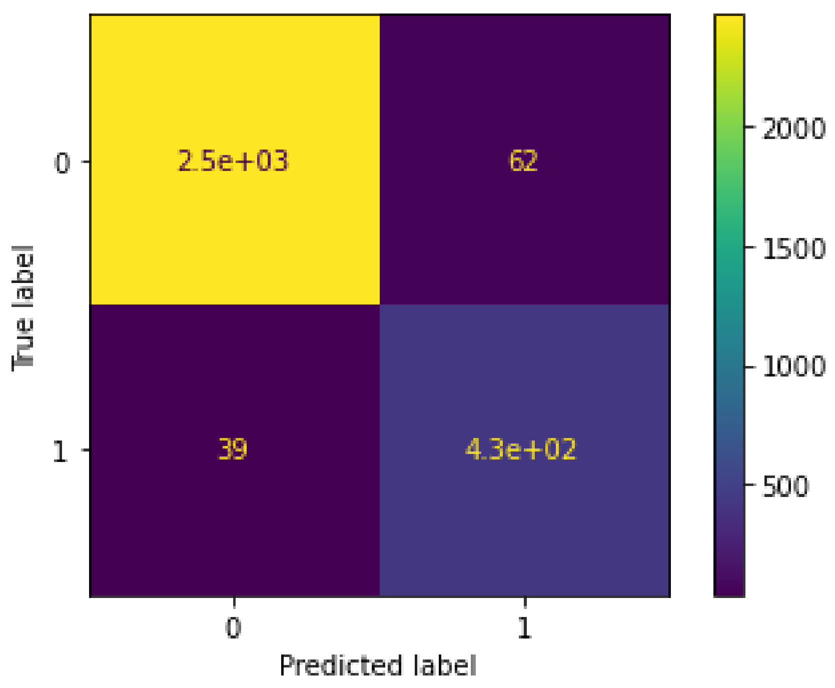
[66]:
```
#evaluate the model by attaining important metrics
print("Decision Tree")
print("Accuracy:", "%.6f" % metrics.accuracy_score(y_test, dt_pred))
print("Precision:", "%.6f" % metrics.precision_score(y_test, dt_pred))
print("Recall:", "%.6f" % metrics.recall_score(y_test, dt_pred))
print("F1 Score:", "%.6f" % metrics.f1_score(y_test, dt_pred))
print("auc:", "%.6f" % metrics.roc_auc_score(y_test, dt_pred))
```

```
Decision Tree
Accuracy: 0.966311
Precision: 0.874747
Recall: 0.917373
F1 Score: 0.895553
auc: 0.946414
```

The model is 96% accurate. the precision score is 87% meaning that the model was good at predicting false positives. the model was also good at predicting false negatives as the recall score was 91%, the harmonic mean of the precision and recall score is 89%. The overall performance of the model is satisfactory but we have to build other powerful models for more accuracy

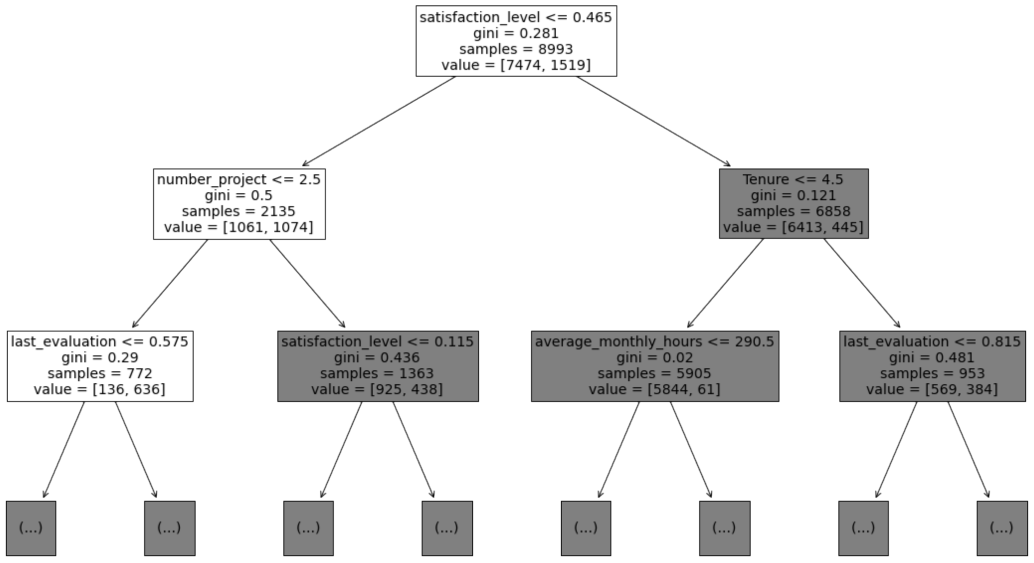
[54]:
```
#plot the confusion matrix
cm = metrics.confusion_matrix(y_test, dt_pred, labels = decision_tree.classes_)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = cm,display_labels =␣
↪decision_tree.classes_)
disp.plot()
```

[54]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fc35dbe3890>
```



the true negatives of the decision model are 2,500, that means the model predicted accurately that 2,500 did not leave the company . the true positives are 430, which means the model accurately predicted that 430 people left the company. the false positives are 62 which means 62 people did not leave the company but the model inaccurately predicted that they left. the false negatives are 39, which means 39 employees actually left the company but the model inaccurately predicted that they did not leave.

[55]:
```
plt.figure(figsize=(20,12))
plot_tree(decision_tree, max_depth=2, fontsize=14, feature_names=X.columns);
```



The first line of information in each node is the feature and split point that the model identified as being most predictive. In other words, this is the question that is being asked at that split. For our root node, the question was: Is the customer satisfactory level less than or equal to 0.46?

At each node, if the answer to the question it asks is "yes," the sample would move to the child node on the left. If the answer is "no," the sample would go to the child node on the right.

gini refers to the node's Gini impurity. This is a way of measuring how "pure" a node is. The value can range from 0 to 0.5. A Gini score of 0 means there is no impurity—the node is a leaf, and all of its samples are of a single class. A score of 0.5 means the classes are all equally represented in that node.

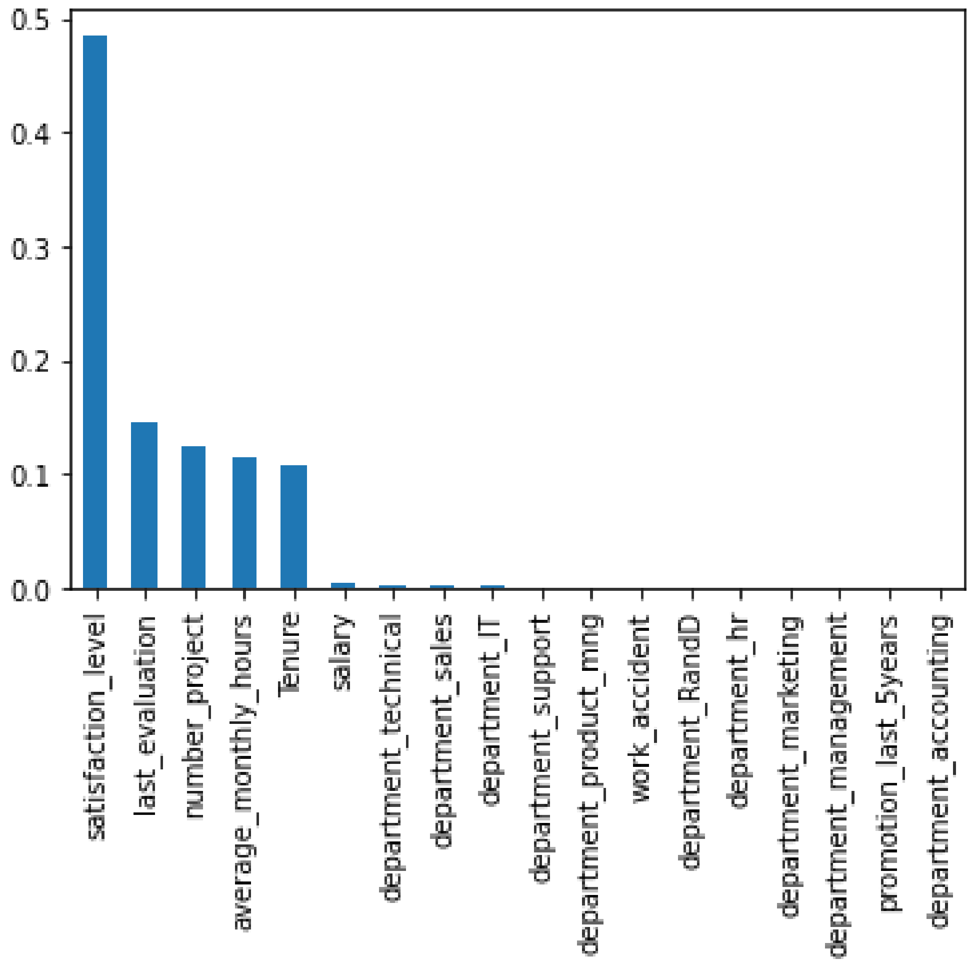
samples is simply how many samples are in that node, and value indicates how many of each class are in the node. Returning to the root node, we have value = [7474, 1519]. Notice that these numbers sum to 8,993, which is the number of samples in the node. This tells us that 7,474 employees in this node did not leave (y=0) and 1,519 employees left (y=1).

This plot tells us that, if we could only do a single split on a single variable, the one that would most help us predict whether an employee will leave is their satisfactory level.

If we look at the nodes at depth one, we notice that the number of projects and tenure also are both strong predictors (relative to the features we have) of whether or not employees will leave.

[56]:
```
importances = decision_tree.feature_importances_

forest_importances = pd.Series(importances, index=X.columns).
 ↪sort_values(ascending=False)
```

```
fig, ax = plt.subplots()
forest_importances.plot.bar(ax=ax);
```



[57]:
```
#tune the parameters because decision trees are prone to overfitting

tree_para = {'max_depth':
↪[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,30,40,50],
             'min_samples_leaf': [2,3,4,5,6,7,8,9, 10, 15, 20, 50]}

scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}
```

[58]:
```
#check combination of values
tuned_decision_tree = DecisionTreeClassifier(random_state=0)

clf = GridSearchCV(tuned_decision_tree,
                   tree_para,
                   scoring = scoring,
                   cv=5,
                   refit="f1")

clf.fit(X_train, y_train)
```

```
[58]: 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=0, splitter='best'),
                   iid='deprecated', n_jobs=None,
                   param_grid={'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                             13, 14, 15, 16, 17, 18, 19, 20, 30, 40,
                                             50],
                               'min_samples_leaf': [2, 3, 4, 5, 6, 7, 8, 9, 10, 15,
                                                    20, 50]},
                   pre_dispatch='2*n_jobs', refit='f1', return_train_score=False,
                   scoring={'f1', 'precision', 'accuracy', 'recall', 'roc_auc'},
                   verbose=0)
```

```
[106]: clf.best_estimator_
```

```
[106]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                              max_depth=7, max_features=None, max_leaf_nodes=None,
                              min_impurity_decrease=0.0, min_impurity_split=None,
                              min_samples_leaf=2, min_samples_split=2,
                              min_weight_fraction_leaf=0.0, presort='deprecated',
                              random_state=0, splitter='best')
```

```
[59]: print("Best Avg. Validation Score: ", "%.4f" % clf.best_score_)
```

```
Best Avg. Validation Score:  0.9388
```

```
[73]: #determine the best scores combination
      ### YOUR CODE HERE ###

      results = pd.DataFrame(columns=['Model', 'F1', 'Recall', 'Precision',␣
       ↪'Accuracy','auc'])
```

```
def make_results(model_name, model_object):
    """
    Accepts as arguments a model name (your choice - string) and
    a fit GridSearchCV model object.

    Returns a pandas df with the F1, recall, precision, and accuracy scores
    for the model with the best mean F1 score across all validation folds.
    """

    # Get all the results from the CV and put them in a df.
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(mean f1 score).
    best_estimator_results = cv_results.iloc[cv_results['mean_test_f1'].
 ↪idxmax(), :]

    # Extract accuracy, precision, recall, and f1 score from that row.
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy
    auc = best_estimator_results.mean_test_roc_auc


    # Create table of results
    table = pd.DataFrame({'Model': [model_name],
                          'F1': [f1],
                          'Recall': [recall],
                          'Precision': [precision],
                          'Accuracy': [accuracy],
                          'auc': [auc]
                         }
                        )

    return table

result_table = make_results("Tuned Decision Tree", clf)

result_table
```

[73]:
```
                  Model        F1    Recall  Precision  Accuracy       auc
0  Tuned Decision Tree  0.938829  0.911141   0.968643  0.979985  0.961587
```

The model after tuning the hyperparameters is 97% accurate. the precision score is 96% meaning that the model was good at predicting false positives. the model was also good at predicting false negatives as the recall score was 91%, the harmonic mean of the precision and recall score is 93%. The auc score is 96.1 meaning that the model's predictions are 96.1% correct. The overall performance of the model is satisfactory but we have to build other powerful models for more accuracy

7 Random Forest Model and XGBoost

[74]:
```
#import all necessary packages
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import accuracy_score, precision_score, recall_score,\
f1_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay,␣
 ↪PrecisionRecallDisplay
```

[75]:
```
#isolated x and y variables
y = df['left']
#copy the data
X = df.copy()
# Drop unnecessary columns
X = X.drop(['left'], axis=1)
# Encode the `salary` column as an ordinal numeric category
X['salary'] = (
    X['salary'].astype('category')
    .cat.set_categories(['low', 'medium', 'high'])
    .cat.codes
)

# Dummy encode the `department` column
X = pd.get_dummies(X, drop_first=False)

# Display the new dataframe
X.head()
```

[75]:
```
   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  salary  department_IT  \
0     3.0              0                      0       0              0
1     5.5              0                      0       1              0
2     4.0              0                      0       1              0
3     5.0              0                      0       0              0
4     3.0              0                      0       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
0                 1                   0                     0
1                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
```

[76]:
```
#split the data
# 3. Split into train and test sets
X_tr, X_test, y_tr, y_test = train_test_split(X, y, stratify=y,
                                              test_size=0.2, random_state=42)

# 4. Split into train and validate sets
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, stratify=y_tr,
                                                  test_size=0.25,␣
 ↪random_state=42)
```

[77]:
```
#print the size of the train, validation and test dataset
for x in [X_train, X_val, X_test]:
    print(len(x))
```

```
7194
2398
2399
```

[78]:
```
# 1. Instantiate the random forest classifier
rf = RandomForestClassifier(random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [None],
             'max_features': [1.0],
             'max_samples': [0.7],
             'min_samples_leaf': [2],
             'min_samples_split': [2],
             'n_estimators': [300],
             }

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

# 4. Instantiate the GridSearchCV object
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='f1')
```

[81]:
```
#fit the model
rf_cv.fit(X_train, y_train)
```

[81]:
```
GridSearchCV(cv=4, 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,
                                              oob_score=False, random_state=42,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [None], 'max_features': [1.0],
                         'max_samples': [0.7], 'min_samples_leaf': [2],
                         'min_samples_split': [2], 'n_estimators': [300]},
             pre_dispatch='2*n_jobs', refit='f1', return_train_score=False,
             scoring={'f1', 'precision', 'accuracy', 'recall', 'roc_auc'},
             verbose=0)
```

[82]:
```
rf_cv.best_score_
```

[82]:
```
0.9412137989879787
```

[83]:
```
rf_cv.best_params_
```

[83]:
```
{'max_depth': None,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 300}
```

[86]:
```
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
        model_name (string): what you want the model to be called in the output␣
↪table
        model_object: a fit GridSearchCV object
        metric (string): precision, recall, f1, or accuracy

    Returns a pandas df with the F1, recall, precision, and accuracy scores
    for the model with the best mean 'metric' score across all validation folds.
    '''

    # Create dictionary that maps input metric to actual metric name in␣
↪GridSearchCV
    metric_dict = {'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy',
                   'auc': 'mean_test_roc_auc'
                   }

    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].
↪idxmax(), :]

    # Extract accuracy, precision, recall, and f1 score from that row
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy
    auc = best_estimator_results.mean_test_roc_auc

    # Create table of results
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          'auc': [auc]
                          },
                         )
    return table
```

[87]:
```
results = make_results('RF cv', rf_cv, 'f1')
results
```

[87]:
```
   model  precision    recall        F1  accuracy       auc
0  RF cv   0.978531  0.907106  0.941214  0.981234  0.976757
```

The random forest model is 98% accurate. the precision score is 97% meaning that the model is good at predicting false positives. the model is also good at predicting false negatives as the recall score was 90%, the harmonic mean of the precision and recall score is 94%. The auc score is 97.6 meaning that the model's predictions are 97.6% correct.The overall performance of the model is satisfactory.

[88]:
```
import pickle
path = '/home/jovyan/work/'
```

[89]:
```
# Pickle the model
with open(path + 'rf_cv_model.pickle', 'wb') as to_write:
    pickle.dump(rf_cv, to_write)
```

[90]:
```
# Open pickled model
with open(path+'rf_cv_model.pickle', 'rb') as to_read:
    rf_cv = pickle.load(to_read)
```

[ ]:
```

```

8 XGBoost model

[91]:
```
# 1. Instantiate the XGBoost classifier
xgb = XGBClassifier(objective='binary:logistic', random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [6, 12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [300]
             }

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

# 4. Instantiate the GridSearchCV object
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='f1')
```

[92]:
```
%%time
xgb_cv.fit(X_train, y_train)
```

```
CPU times: user 1min 52s, sys: 0 ns, total: 1min 52s
Wall time: 57 s
```

[92]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=None, gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, max…
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     predictor=None, random_state=42,
                                     reg_alpha=None, …),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.01, 0.1], 'max_depth': [6, 12],
                         'min_child_weight': [3, 5], 'n_estimators': [300]},
             pre_dispatch='2*n_jobs', refit='f1', return_train_score=False,
             scoring={'f1', 'precision', 'accuracy', 'recall', 'roc_auc'},
             verbose=0)
```

[93]:
```
path = '/home/jovyan/work/'
```

[ ]:
```
# Pickle the model
with open(path + 'xgb_cv_model.pickle', 'wb') as to_write:
    pickle.dump(xgb_cv, to_write)
```

[95]:
```
# Open pickled model
with open(path+'xgb_cv_model.pickle', 'rb') as to_read:
    xgb_cv = pickle.load(to_read)
```

[96]:
```
xgb_cv.best_score_
```

[96]:
```
0.9384184790059694
```

[97]:
```
xgb_cv.best_params_
```

[97]:
```
{'learning_rate': 0.1,
 'max_depth': 12,
 'min_child_weight': 3,
 'n_estimators': 300}
```

[98]:
```
# Call 'make_results()' on the GridSearch object
xgb_cv_results = make_results('XGB cv', xgb_cv, 'f1')
results = pd.concat([results, xgb_cv_results], axis=0)
results
```

[98]:
```
    model  precision    recall        F1  accuracy       auc
0   RF cv   0.978531  0.907106  0.941214  0.981234  0.976757
0  XGB cv   0.971799  0.907937  0.938418  0.980261  0.979330
```

The xgboost model is 98% accurate. the precision score is 97.17% meaning that the model was good at predicting false positives. the model was also good at predicting false negatives as the recall score was 90.79%, the harmonic mean of the precision and recall score is 93%. The auc score is 97.9 meaning that the model's predictions are 97.9% correct.The overall performance of the model is satisfactory but the rf f1 score is better

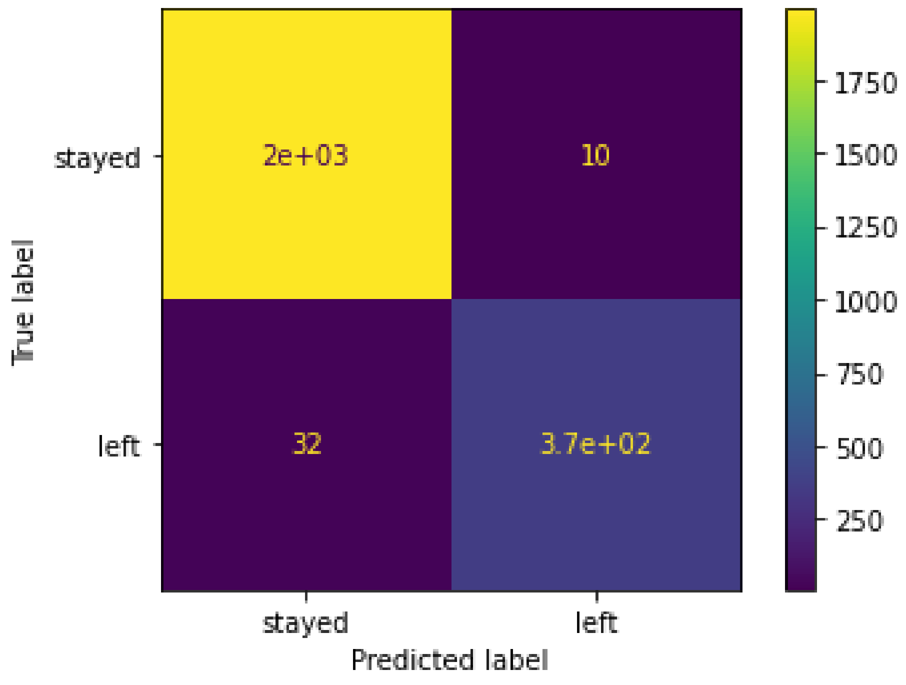
[102]:
```
# Use random forest model to predict on validation data
rf_val_preds = rf_cv.best_estimator_.predict(X_val)
```

[100]:
```
def get_test_scores(model_name:str, preds, y_test_data):
    '''
    Generate a table of test scores.

    In:
        model_name (string): Your choice: how the model will be named in the␣
 ↪output table
        preds: numpy array of test predictions
        y_test_data: numpy array of y_test data

    Out:
        table: a pandas df of precision, recall, f1, and accuracy scores for␣
 ↪your model
    '''
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)
    auc = roc_auc_score(y_test_data, preds)

    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          'auc': [auc]
                          })
    return table
```

[103]:
```
# Get validation scores for RF model
rf_val_scores = get_test_scores('RF val', rf_val_preds, y_val)

# Append to the results table
results = pd.concat([results, rf_val_scores], axis=0)
results
```

[103]:
```
    model  precision    recall        F1  accuracy       auc
0   RF cv   0.978531  0.907106  0.941214  0.981234  0.976757
0  XGB cv   0.971799  0.907937  0.938418  0.980261  0.979330
0  RF val   0.973404  0.919598  0.945736  0.982485  0.957299
```

[104]:
```
cm = confusion_matrix(y_val, rf_val_preds, labels=rf_cv.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['stayed', 'left'])
disp.plot();
```



[ ]:

[105]:
```python
# Use XGBoost model to predict on validation data
xgb_val_preds = xgb_cv.best_estimator_.predict(X_val)

# Get validation scores for XGBoost model
xgb_val_scores = get_test_scores('XGB val', xgb_val_preds, y_val)

# Append to the results table
results = pd.concat([results, xgb_val_scores], axis=0)
results
```

[105]:
```
     model  precision    recall        F1  accuracy       auc
0    RF cv   0.978531  0.907106  0.941214  0.981234  0.976757
0   XGB cv   0.971799  0.907937  0.938418  0.980261  0.979330
0   RF val   0.973404  0.919598  0.945736  0.982485  0.957299
0  XGB val   0.968000  0.912060  0.939198  0.980400  0.953030
```

[106]:
```python
#Use XGBoost model to predict on test data
xgb_test_preds = xgb_cv.best_estimator_.predict(X_test)

# Get test scores for XGBoost model
xgb_test_scores = get_test_scores('XGB test', xgb_test_preds, y_test)

# Append to the results table
results = pd.concat([results, xgb_test_scores], axis=0)
results
```

[106]:
```
      model  precision    recall        F1  accuracy       auc
0     RF cv   0.978531  0.907106  0.941214  0.981234  0.976757
0    XGB cv   0.971799  0.907937  0.938418  0.980261  0.979330
0    RF val   0.973404  0.919598  0.945736  0.982485  0.957299
0   XGB val   0.968000  0.912060  0.939198  0.980400  0.953030
0  XGB test   0.955844  0.924623  0.939974  0.980409  0.958064
```

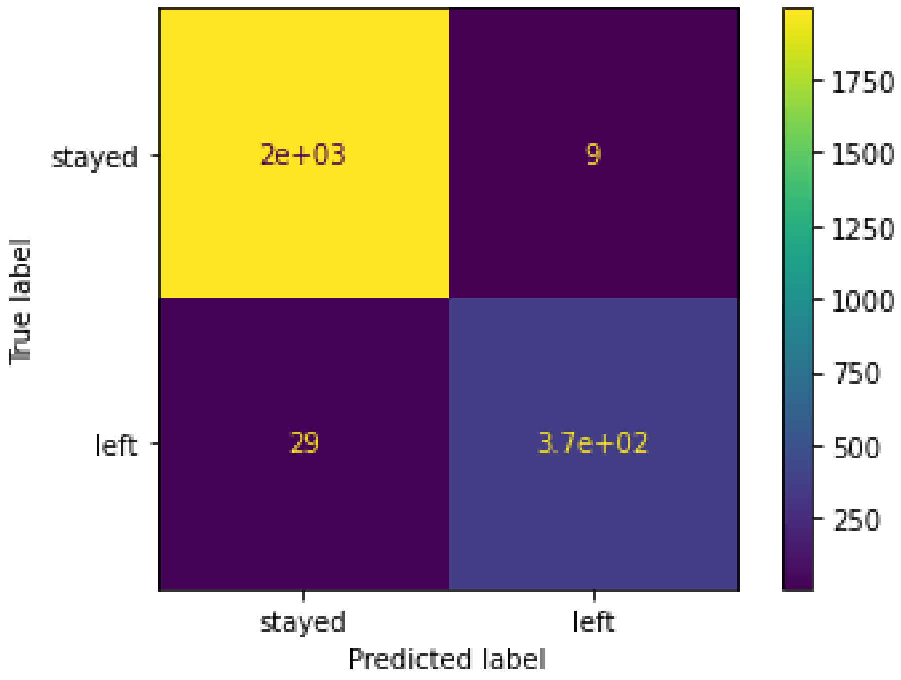
[107]:
```python
#Use XGBoost model to predict on test data
rf_test_preds = rf_cv.best_estimator_.predict(X_test)

# Get test scores for XGBoost model
rf_test_scores = get_test_scores('rf test', rf_test_preds, y_test)

# Append to the results table
results = pd.concat([results, rf_test_scores], axis=0)
results
```

[107]:
```
      model  precision    recall        F1  accuracy       auc
0     RF cv   0.978531  0.907106  0.941214  0.981234  0.976757
0    XGB cv   0.971799  0.907937  0.938418  0.980261  0.979330
0    RF val   0.973404  0.919598  0.945736  0.982485  0.957299
```

```
0   XGB val   0.968000  0.912060  0.939198  0.980400  0.953030
0  XGB test   0.955844  0.924623  0.939974  0.980409  0.958064
0   rf test   0.976190  0.927136  0.951031  0.984160  0.961319
```

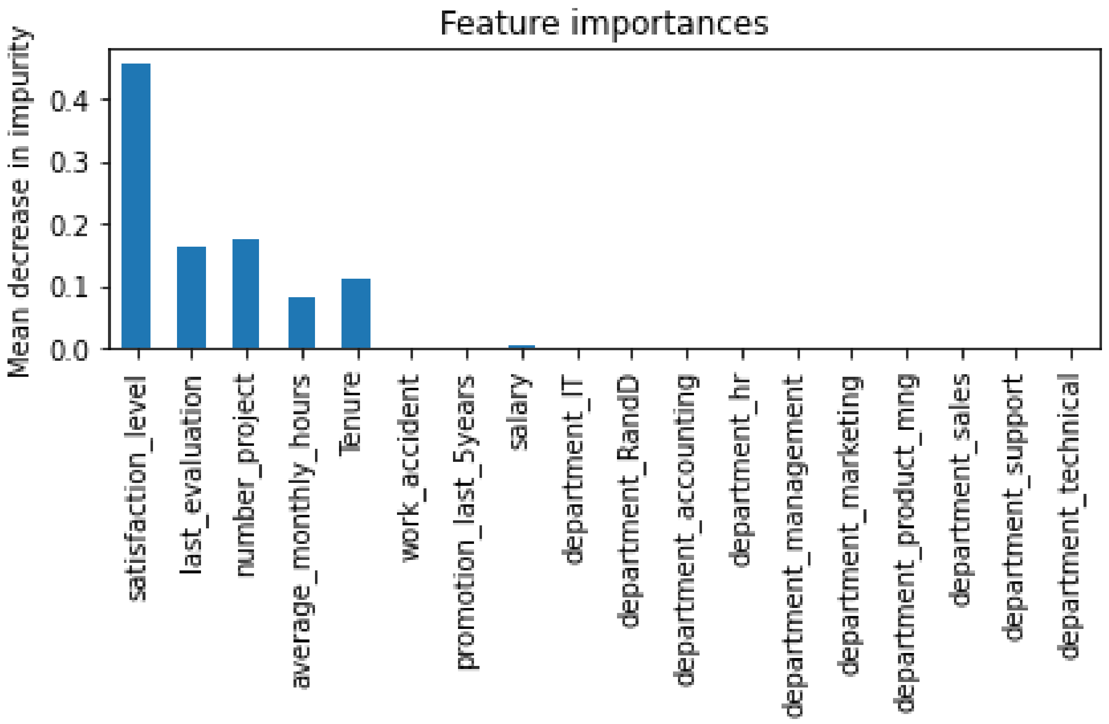
After using both models to predict on the test data, the random forest model emerged as the champion model because it has a stronger f1 score and recall, precision and accuracy scores. The random forest model is 98.41% accurate. the precision score is 97.61% meaning that the model was good at predicting false positives. the model was also good at predicting false negatives as the recall score was 92.71%, the harmonic mean of the precision and recall score is 95.1%. The auc score is 96.1 meaning that the model's predictions are 96.1% correct.

[108]:
```python
cm = confusion_matrix(y_test, rf_test_preds, labels=rf_cv.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['stayed', 'left'])
disp.plot();
```



the true negatives of the decision model are 2000, that means the model predicted accurately that 2000 did not leave the company . the true positives are 370, which means the model accurately predicted that 370 people left the company. the false positives are 9 which means 9 people did not leave the company but the model inaccurately predicted that they left. the false negatives are 29, which means 29 employees actually left the company but the model inaccurately predicted that they did not leave.

[109]:
```
importances = rf_cv.best_estimator_.feature_importances_
rf_importances = pd.Series(importances, index=X_test.columns)

fig, ax = plt.subplots()
rf_importances.plot.bar(ax=ax)
ax.set_title('Feature importances')
ax.set_ylabel('Mean decrease in impurity')
fig.tight_layout()
```



In the random forest model, satisfaction level was the most important feature that was used in prediction followed by last evaluation, number of projects, average monthly hours and tenure

[ ]:
```
```

[ ]:
```
```

[ ]:
```
```

9 pacE: Execute Stage

- Interpret model performance and results
- Share actionable steps with stakeholders

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?
- 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?

10 INSIGHTS

In the random forest model, satisfaction level was the most important feature that was used in prediction followed by last evaluation, number of projects, average monthly hours and tenure

10.1 Step 4. Results and Evaluation

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

10.1.1 Summary of model results

- Logistic Regression The logistic regression model achieved precision of 39%, recall of 16%, f1-score of 22% (all weighted averages), and accuracy of 82%, on the test set.

- Tree-based Machine Learning After conducting feature engineering, the decision tree model achieved AUC of 96.1%, precision of 96%, recall of 91%, f1-score of 93%, and accuracy of 97%, on the test set.

- Random Forest Machine Learning The model is 98.41% accurate. the precision score is 97.61% meaning that the model was good at predicting false positives. the model was also good at predicting false negatives as the recall score was 92.71%, the harmonic mean of the precision and recall score is 95.1%. The auc score is 96.1 meaning that the model's predictions are 96.1% correct. The random forest model emerged as the champion model.

- Xgboost Machine Learning The xgboost model is 98% accurate. the precision score is 97.17% meaning that the model was good at predicting false positives. the model was also good at predicting false negatives as the recall score was 90.79%, the harmonic mean of the precision and recall score is 93%. The overall performance of the model is satisfactory but the rf f1 score is better

10.1.2 Recommendations

The models and the feature importances extracted from the models confirm that employees at the company are overworked. To retain employees, the following recommendations could be presented to the stakeholders: - Investigate reasons behind such long hours for dissatisfied employees. Are they struggling to meet unrealistic deadlines? Are they under-resourced? Implement strategies to reduce workload or improve efficiency for these employees.

- Conduct exit interviews or surveys to understand why employees with moderate satisfaction and less monthly hours left. Explore factors like lack of growth opportunities, recognition issues, or company culture.
- Conduct stay interviews with satisfied employees who are at risk of leaving (e.g., those with high satisfaction but moderate hours). This might reveal underlying concerns or areas for improvement before they leave.
- For low performers, address performance issues through targeted training or mentorship. For high performers working long hours, investigate workload distribution and consider strategies like project delegation or hiring additional staff to prevent burnout.
- Focus on identifying and promoting high-performing employees with the potential to boost retention.
- Analyze project allocation to ensure employees are challenged but not overloaded. Implement strategies like job rotation or skill development opportunities to prevent boredom or stagnation for those with fewer projects.
- Develop targeted strategies for different tenure groups. Implement strong onboarding programs for new hires to improve engagement and satisfaction. For employees in the third or fourth year, explore opportunities for growth or skill development to prevent stagnation. Foster a positive work environment that retains long-tenured employees.
- Conduct a salary review to ensure competitive compensation across all positions. Consider merit-based raises or bonuses to recognize high performance and incentivize retention.
- Investigate reasons for high turnover in specific departments. Conduct surveys or focus groups to understand departmental challenges and develop strategies to address them.
- While accidents don't seem to be a major factor, prioritize safety protocols and employee well-being to prevent future incidents.

10.1.3 NEXT STEPS

Deeper Dives:

- Analyze Specific Departments: Focus on departments with high turnover (Sales and Technical) to understand their unique challenges. Conduct targeted surveys or focus groups within these departments to gather more specific information.
- Performance Reviews: Investigate the relationship between performance reviews and satisfaction levels. Are there biases in the evaluation process? Do low performers receive adequate feedback and support for improvement?
- Compensation Analysis: Conduct a more detailed analysis of salaries, bonuses, and benefits. Are there any pay gaps based on gender, race, or experience? Do the benefits packages address employee needs and preferences?

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.