Activity_ Course 7 Salifort Motors project lab

May 15, 2025

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 ###
# For data manipulation
import numpy as np
import pandas as pd

# For data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# For displaying all of the columns in dataframes
pd.set_option('display.max_columns', None)

# For data modeling
from xgboost import XGBClassifier
from xgboost import XGBRegressor
from xgboost import plot_importance

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# 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
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.tree import plot_tree

# For saving models
import pickle
```

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

[3]:
```
# 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()
```

[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
```

2.3.3 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.

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

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

2.3.4 Check missing values

Check for any missing values in the data.

[7]:
```python
# Check for missing values
### YOUR CODE HERE ###
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
```

2.3.5 Check duplicates

Check for any duplicate entries in the data.

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

[8]:
```
3008
```

[9]:
```python
# Inspect some rows containing duplicates as needed
### YOUR CODE HERE ###
df0[df0.duplicated()].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]:
```python
# Drop duplicates and save resulting dataframe in a new variable as needed
### YOUR CODE HERE ###
df1 = df0.drop_duplicates(keep='first')
```

```
# Display first few rows of new dataframe as needed
### YOUR CODE HERE ###
df1.head()
```

[10]:

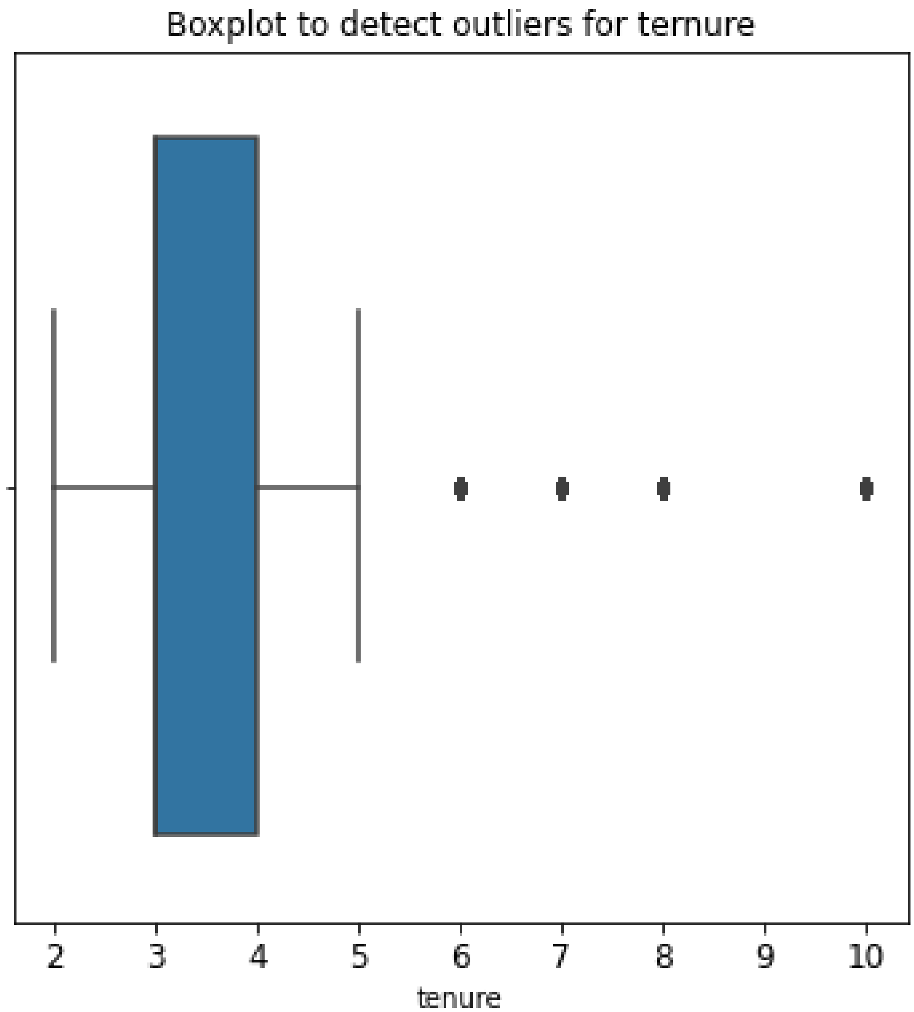
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
2                0.11             0.88               7                    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
```

2.3.6 Check outliers

Check for outliers in the data.

[11]:

```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
### YOUR CODE HERE ###
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers for ternure', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=df1['tenure'])
plt.show()
```



[12]:
```
# Determine the number of rows containing outliers
### YOUR CODE HERE ###
percentile25 = df1['tenure'].quantile(0.25)

percentile75 = df1['tenure'].quantile(0.75)

iqr = percentile75 - percentile25

upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr
print('Upper Limit:', upper_limit)
print('Lower Limit:', lower_limit)

outliers = df1[(df1['tenure'] > upper_limit) | (df1['tenure'] < lower_limit)]
```

```
print('Number of rows in the data containing outliers in tenure:',␣
↪len(outliers))
```

```
Upper Limit: 5.5
Lower Limit: 1.5
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.

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.

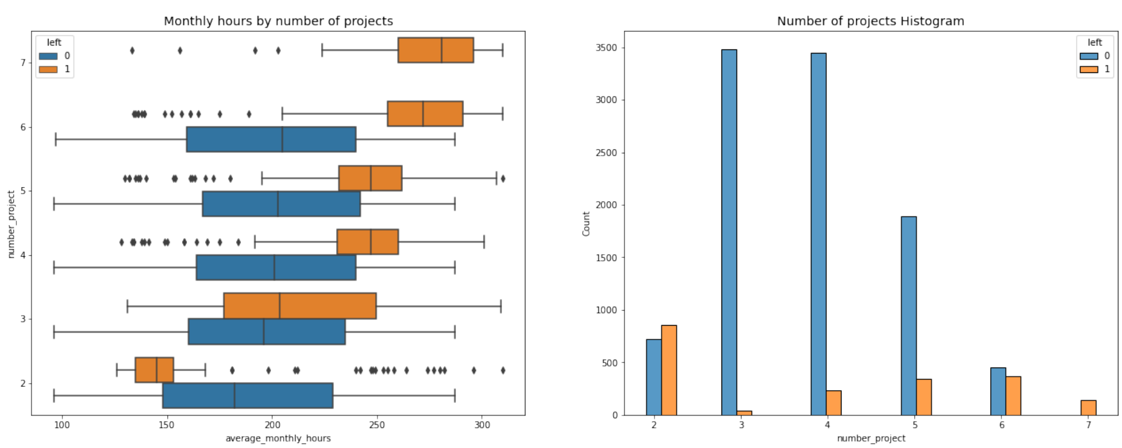
[13]:
```
# Get numbers of people who left vs. stayed
### YOUR CODE HERE ###
print(df1['left'].value_counts())
# Get percentages of people who left vs. stayed
### YOUR CODE HERE ###
print(df1['left'].value_counts(normalize=True))
```

```
0    10000
1     1991
Name: left, dtype: int64
0    0.833959
1    0.166041
Name: left, dtype: float64
```

3.1.1 Data visualizations

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

[14]:
```python
# Create a plot as needed
### YOUR CODE HERE ###

#Set figures and axes
fig, ax = plt.subplots(1, 2, figsize = (22,8))
# Create boxplot showing `average_monthly_hours` distributions for␣
 ↪`number_project`, comparing employees who stayed versus those who left
sns.boxplot(data=df1, x='average_monthly_hours', y='number_project',␣
 ↪hue='left', orient='h', ax=ax[0])
ax[0].invert_yaxis()
ax[0].set_title('Monthly hours by number of projects', fontsize='14')
# Create histogram showing distribution of `number_project`, comparing␣
 ↪employees who stayed versus those who left
ternure_stay = df1[df1['left'] == 0]['number_project']
ternure_left = df1[df1['left'] == 1]['number_project']
sns.histplot(data=df1, x='number_project', hue='left', multiple='dodge',␣
 ↪shrink=2, ax=ax[1])
ax[1].set_title('Number of projects Histogram', fontsize='14')
#Display the plots
plt.show()
```



3.1.2 Insights

From the plots above, it shows that people that has more projects or things to work, has left the company. Thus, we remark a few things that stands out from the graphics:

1. It seems that there are two people that left the company. (A) There are employees with few

projects that left the company, then, an assumption from this scenario is that people that has entered to the company recently has been fired, or people that knew that they were going to be fired, the company reduced the work amount. (B) On the other hand, people that contributes more to the project, left due to the number of projects that they are working on it.

2. Everyone with seven projects left the company, and the interquartile ranges of this group and those who left with six projects was ~255–295 hours/month—much more than any other group.

3. The optimal number of projects for employees to work on seems to be 3–4.

4. If you assume a work week of 40 hours and two weeks of vacation per year, then the average number of working hours per month of employees working Monday–Friday `= 50 weeks * 40 hours per week / 12 months = 166.67 hours per month`. Since the average monthly hour is 200 hours, it means that the employees has been overworking.

As the next step, you could confirm that all employees with seven projects left.

[15]:
```
df1[df1['number_project']==7]['left'].value_counts()
```

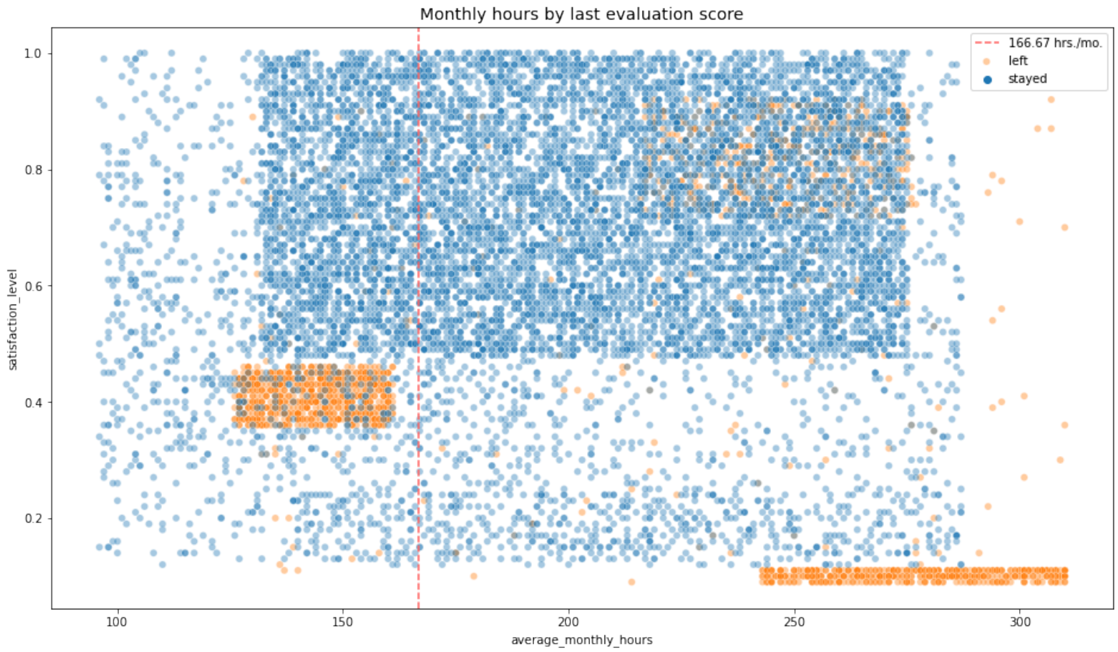
[15]:
```
1    145
Name: left, dtype: int64
```

This confirms that all the employes that works on 7 projects has left the company.

Next, you could examine the average monthly hours versus the satisfaction levels.

[16]:
```
# 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=(16,9))
sns.scatterplot(data=df1, x='average_monthly_hours', y='satisfaction_level',␣
↪hue='left', alpha=0.4)
plt.axvline(x=166.67, color='#ff6361', label='166.67 hrs./mo.',ls='--')
plt.legend(labels=['166.67 hrs./mo.', 'left', 'stayed'])
plt.title('Monthly hours by last evaluation score', fontsize=14)
```

[16]:
```
Text(0.5, 1.0, 'Monthly hours by last evaluation score')
```



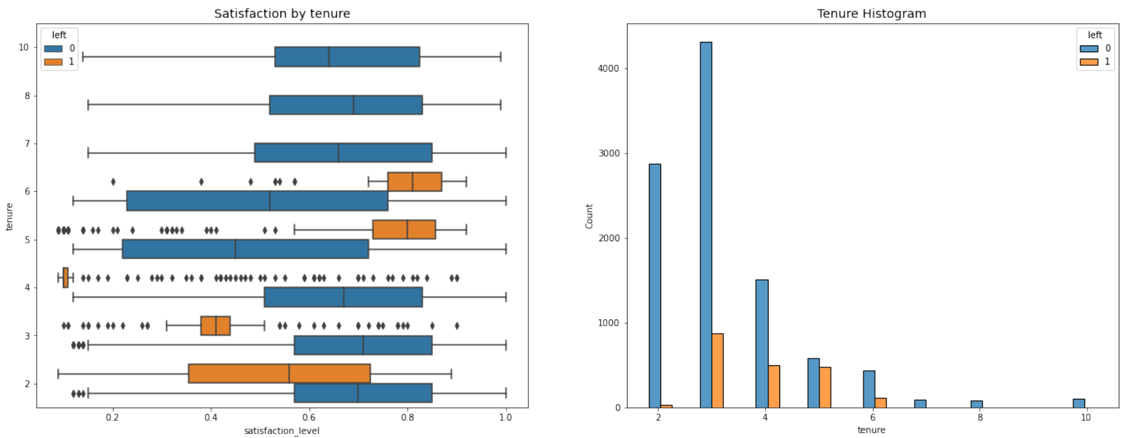
The scatterplot shows three sections of people who left the company:

1. There is a sector of employees who worked around ~240–310 hours per month and their satisfaction level was almost 0.
2. The second sector was employees that works below the ideal working hours by month. There could be several assumptions, one of them could be the pressure to work more, or other factors that could led to satisfaction levels of ~0.4.
3. Then, there is a sector that worked around ~200–260 hours that has a satisfaction levels is closest to 1.

Note the strange shape of the distributions here. This is indicative of data manipulation or synthetic data.

[17]:
```
# Create a plot as needed
### YOUR CODE HERE ###

# Set figure and axes
fig, ax = plt.subplots(1, 2, figsize = (22,8))
# Create boxplot showing distributions of `satisfaction_level` by tenure,␣
↪comparing employees who stayed versus those who left
sns.boxplot(data=df1, x='satisfaction_level', y='tenure', hue='left',␣
↪orient='h', ax=ax[0])
ax[0].invert_yaxis()
ax[0].set_title('Satisfaction by tenure', fontsize='14')
# Create histogram showing distribution of `tenure`, comparing employees who␣
↪stayed versus those who left
```

```
ternure_stay = df1[df1['left'] == 0]['tenure']
ternure_left = df1[df1['left'] == 1]['tenure']
sns.histplot(data=df1, x='tenure', hue='left', multiple='dodge', shrink=5,␣
 ↪ax=ax[1])
ax[1].set_title('Tenure Histogram', fontsize='14')
#Display the plots
plt.show()
```



There are many observations you could make from this plot. - There are two categories: Satisfied employes with longer ternure and dissastified employees with shorter time stayed in the company - There is a unusual that requires further details, and is the one which 4-years employeers that left and their satisfaction level are low. - The longest-tenured employees didn't leave. Their satisfaction levels aligned with those of newer employees who stayed. - The histogram shows that there are relatively few longer-tenured employees. It's possible that they're the higher-ranking, higher-paid employees.

[18]:
```
# Calculate mean and median satisfaction scores of employees who left and those␣
 ↪who stayed
df1.groupby(['left'])['satisfaction_level'].agg([np.mean, np.median])
```

[18]:

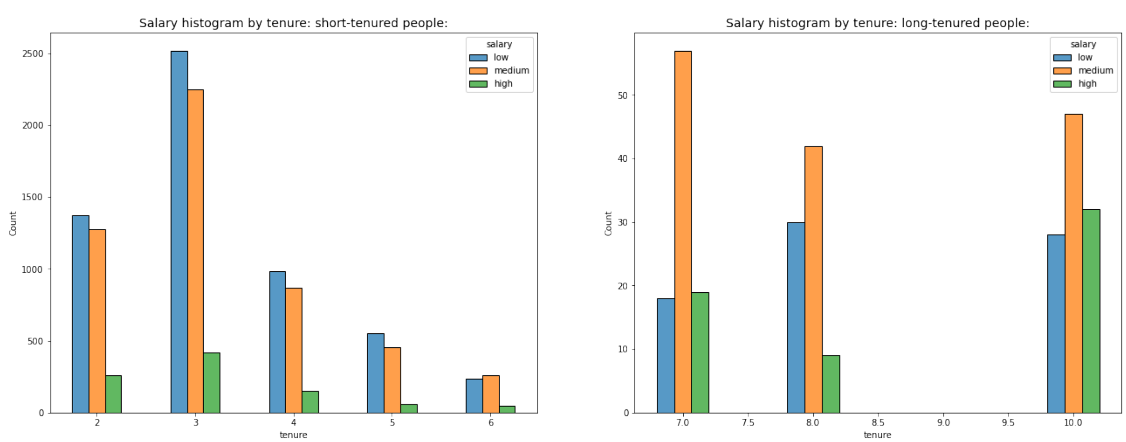
left	mean	median
0	0.667365	0.69
1	0.440271	0.41

The mean and median satisfaction scores of employees who left are lower than those of employees who stayed. Between the employees who stayed, the mean is lower than the media, idnicatint that satisfaction levels among those who stayed might be skewed to the left.

Next, you could examine salary levels for different tenures.

[19]:
```python
# Create a plot as needed
### YOUR CODE HERE ###

# Set figure and axes
fig, ax = plt.subplots(1, 2, figsize = (22,8))
# Define short-tenured employees
tenure_short = df1[df1['tenure'] < 7]
# Define long-tenured employees
tenure_long = df1[df1['tenure'] > 6]
# Plot short-tenured histogram
sns.histplot(data = tenure_short, x = 'tenure', hue = 'salary', discrete = 1,
             hue_order=['low','medium','high'], multiple='dodge', shrink=0.5, ax=ax[0])
ax[0].set_title('Salary histogram by tenure: short-tenured people:', fontsize=14)
# Plot long-tenured histogram
sns.histplot(data = tenure_long, x = 'tenure', hue = 'salary', discrete = 1,
             hue_order=['low','medium','high'], multiple='dodge', shrink=0.4, ax=ax[1])
ax[1].set_title('Salary histogram by tenure: long-tenured people:', fontsize=14)
```

[19]:
```
Text(0.5, 1.0, 'Salary histogram by tenure: long-tenured people:')
```



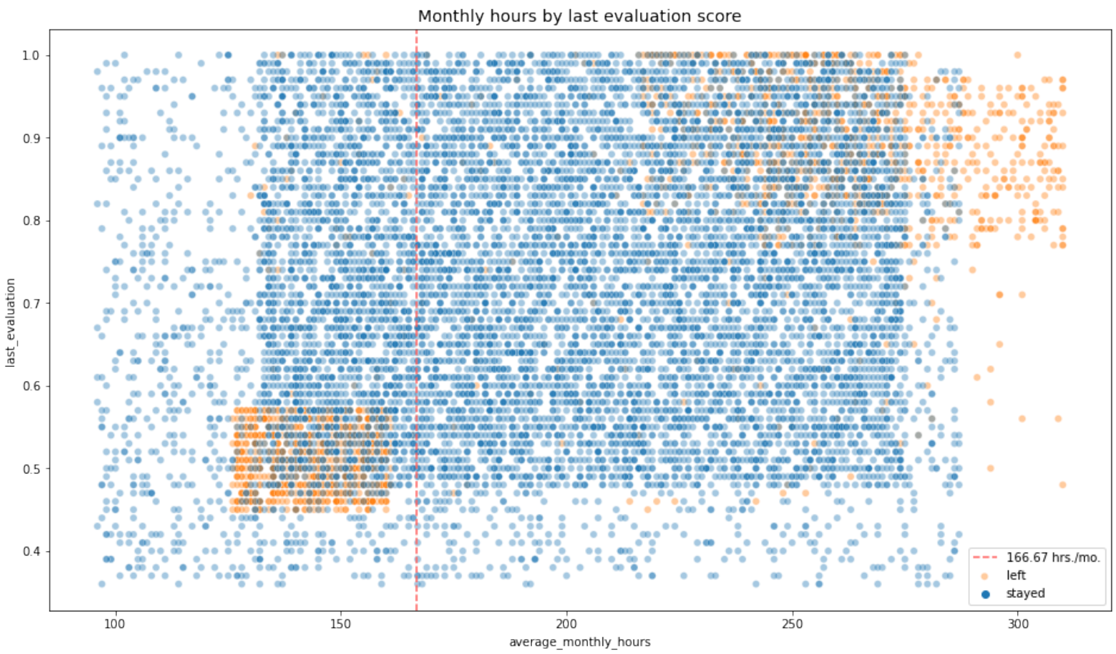
The plots above show that low salary is frequent for short-ternured employees, while mediumd salary was predominant for those that were long-ternured employees.

Next, you could explore whether there's a correlation between working long hours and receiving high evaluation scores. You could create a scatterplot of `average_monthly_hours` versus `last_evaluation`.

[20]:
```python
# Create a plot as needed
### YOUR CODE HERE ###
```

```python
# Create scatterplot of `average_monthly_hours` versus `last_evaluation`
plt.figure(figsize=(16,9))
sns.scatterplot(data=df1, x='average_monthly_hours', y='last_evaluation', hue='left', alpha=0.4)
plt.axvline(x=166.67, color='#ff6361', label='166.67 hrs./mo.',ls='--')
plt.legend(labels=['166.67 hrs./mo.', 'left', 'stayed'])
plt.title('Monthly hours by last evaluation score', fontsize='14')
```

[20]:
```
Text(0.5, 1.0, 'Monthly hours by last evaluation score')
```



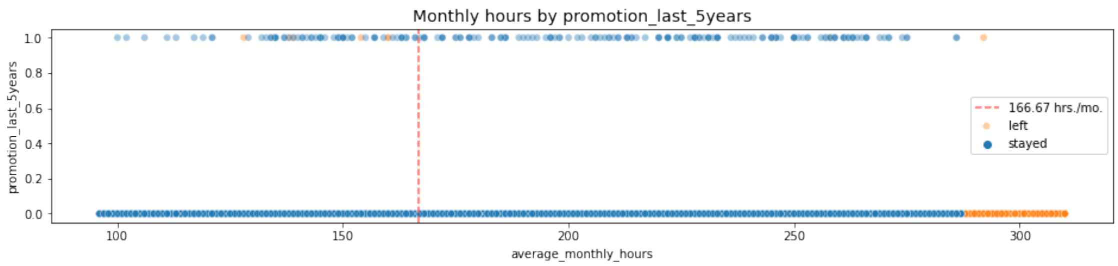
The following observations can be made from the scatterplot above: - We can observe two groups that left, ones that were working below the ideal average monthly hour and those that left the company when working more hours. - It looks disperse the satisfaction score, meaning that working more hour doesn't guarante higher scores, and viceversa. - In the ideal average monthly hour. it looks that most employes stayed. - There is a correlation between the hours worked and evaluation score (?)

Next, you could examine whether employees who worked very long hours were promoted in the last five years.

[21]:
```python
# Create a plot as needed
### YOUR CODE HERE ###

# Create plot to examine relationship between `average_monthly_hours` and `promotion_last_5years`
plt.figure(figsize=(16,3))
```

```
sns.scatterplot(data=df1, x='average_monthly_hours', y='promotion_last_5years',␣
↪hue='left', alpha=0.4)
plt.axvline(x=166.67, color='#ff6361',ls='--')
plt.legend(labels=['166.67 hrs./mo.', 'left', 'stayed'])
plt.title('Monthly hours by promotion_last_5years', fontsize='14')
```

[21]: Text(0.5, 1.0, 'Monthly hours by promotion_last_5years')



The plot above shows the following:

- When the employees were promoted, it seems just few from the left.
- Most of the employees that were working more hours, a fet of them were promoted.
- All the employees that work more, left the company.

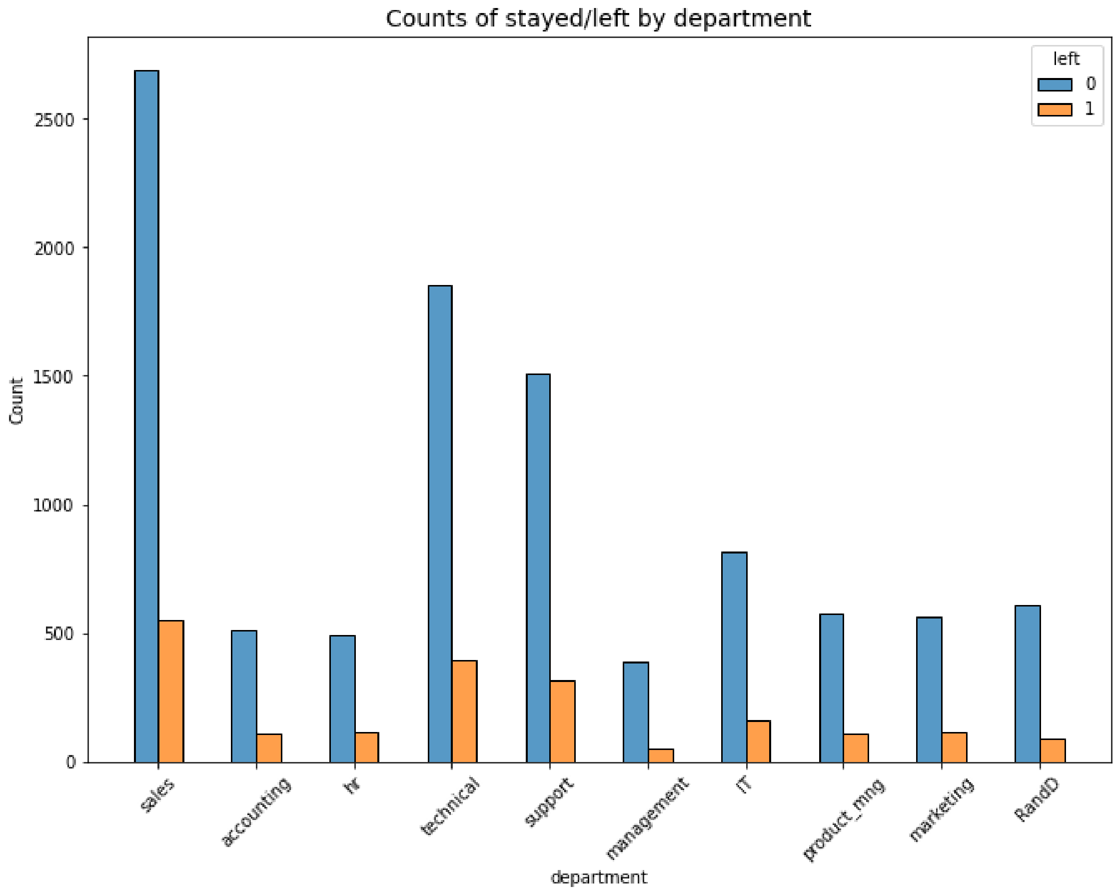
Next, you could inspect how the employees who left are distributed across departments.

[22]:
```
df1['department'].value_counts()
```

[22]:
```
sales          3239
technical      2244
support        1821
IT              976
RandD           694
product_mng     686
marketing       673
accounting      621
hr              601
management      436
Name: department, dtype: int64
```

[23]:
```
# Create a plot as needed
### YOUR CODE HERE ###

# Create stacked histogram to compare department distribution of employees who␣
↪left to that of employees who didn't
plt.figure(figsize=(11,8))
sns.histplot(data=df1, x='department', hue='left', discrete=1,
             hue_order=[0, 1], multiple='dodge', shrink=0.5)
```

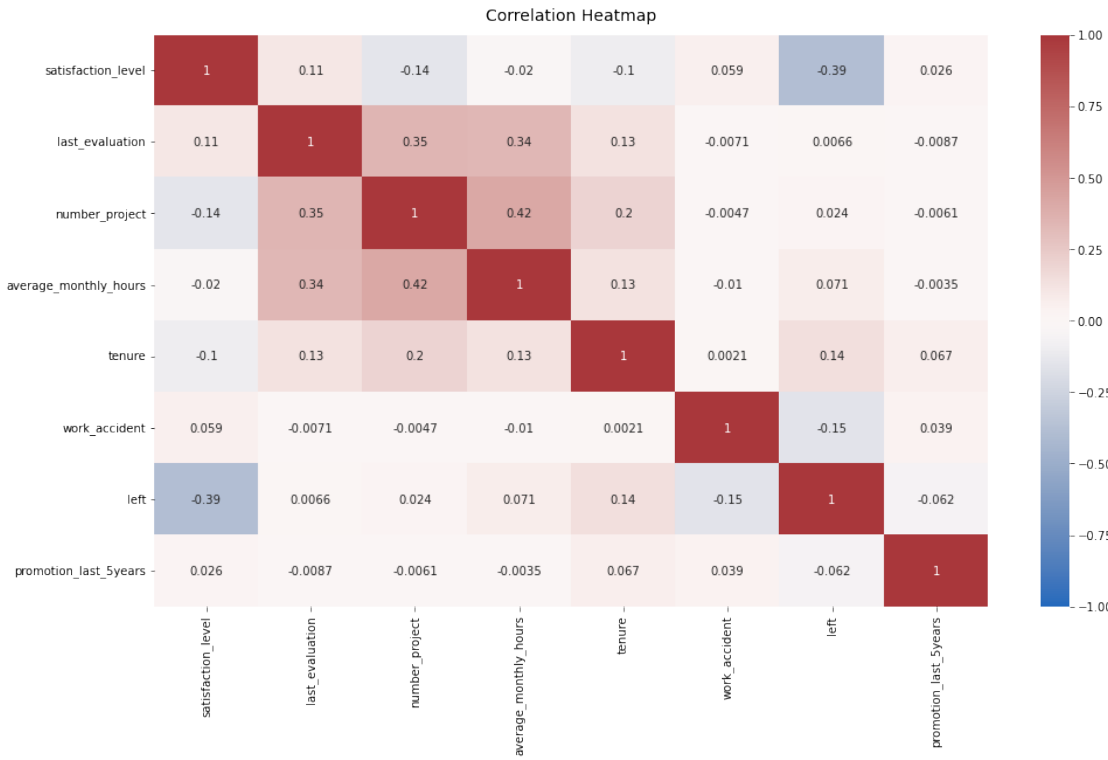
```
plt.xticks(rotation='45')
plt.title('Counts of stayed/left by department', fontsize=14);
```



The plot above doesn't show any relevant info because there is not big difference between the employees who left to those who stayed.

Lastly, you could check for strong correlations between variables in the data.

[24]:
```
# Create a plot as needed
### YOUR CODE HERE ###

# Plot a correlation heatmap
plt.figure(figsize=(16,9))
heatmap = sns.heatmap(df0.corr(), vmin=-1, vmax=1, annot=True, cmap=sns.
 ↪color_palette('vlag', as_cmap=True))
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':14}, pad=12);
```



There is atrong correlation bewteen the number of projects, working hours and score evalation. Also, it seems that scores and employees that left correlate each other.

3.1.3 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

For the next visualization, it might be interesting to visualize satisfaction levels by tenure.

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.

Our task is to identify or create a model to predict if a employee left or not. Then, it requires binary classification since `left` variable uses 0 (employees who stay) and 1 (employees that left)

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

Since the variable that we want to predict is categorical, we can use regression model or random forest. Although, we can create both models and compare them.

4.1.3 Modeling Approach: Logistic Regression Model

This approach covers Logistic Regression

Logistic regression Note that binomial logistic regression suits the task because it involves binary classification.

Before splitting the data, encode the non-numeric variables. There are two: `department` and `salary`.

`department` is a categorical variable, which means you can dummy it for modeling.

salary is categorical too, but it's ordinal. There's a hierarchy to the categories, so it's better not to dummy this column, but rather to convert the levels to numbers, 0–2.

[25]:
```
### YOUR CODE HERE ###
df_enc = df1.copy()
# Encode the `salary` column as an ordinal numeric category
df_enc['salary'] = (
    df_enc['salary'].astype('category')
    .cat.set_categories(['low','medium','high'])
    .cat.codes)
# Dummy encode the `department` column
df_enc = pd.get_dummies(df_enc, drop_first=True)
# Display the new dataframe
df_enc.head()
```

[25]:
```
   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  salary  \
0       3              0     1                      0       0
1       6              0     1                      0       1
2       4              0     1                      0       1
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
0                 1                   0                     0
1                 1                   0                     0
2                 1                   0                     0
```

3	1	0	0
4	1	0	0

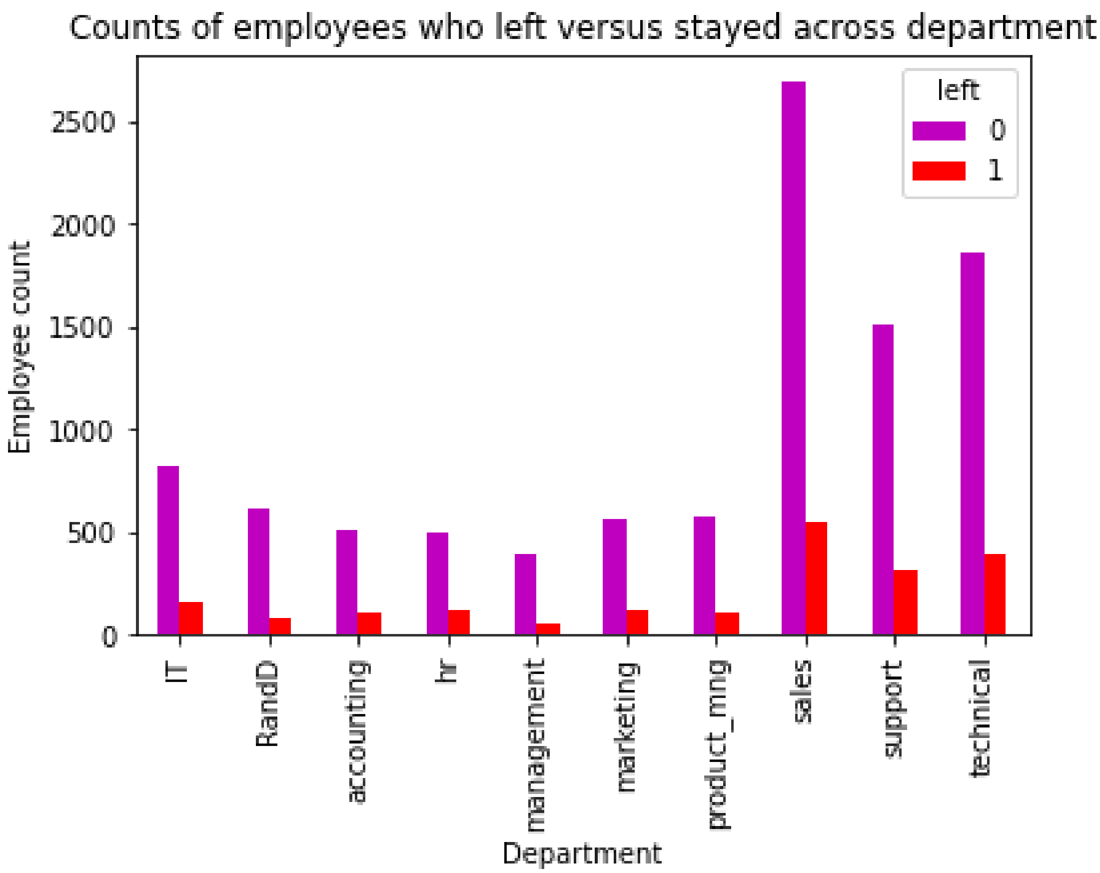
Create a heatmap to visualize how correlated variables are. Consider which variables you're interested in examining correlations between.

[26]:
```
# Create a heatmap to visualize how correlated variables are
plt.figure(figsize=(8,6))
sns.heatmap(df_enc[['satisfaction_level', 'last_evaluation',␣
↪'number_project','average_monthly_hours', 'tenure']]
                .corr(), annot=True, cmap='crest')
plt.title('Heatmap of the Datase')
plt.show()
```

Create a stacked bart plot to visualize number of employees across department, comparing those who left with those who didn't.

[27]:

```
# Create a stacked bart plot to visualize number of employees across␣
↪department, comparing those who left with those who didn't
# In the legend, 0 (purple color) represents employees who did not leave, 1␣
↪(red color) represents employees who left
pd.crosstab(df1['department'], df1['left']).plot(kind='bar',color='mr')
plt.title('Counts of employees who left versus stayed across department')
plt.ylabel('Employee count')
plt.xlabel('Department')
plt.show()
```



Since logistic regression is quite sensitive to outliers, it would be a good idea at this stage to remove the outliers in the `tenure` column that were identified earlier.

[28]:

```
# Select rows without outliers in `tenure` and save resulting dataframe in a␣
↪new variable
df_logreg = df_enc[(df_enc['tenure'] >= lower_limit) & (df_enc['tenure'] <=␣
↪upper_limit)]
# Display first few rows of new dataframe
df_logreg.head()
```

```
[28]:    satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                  0.38             0.53               2                    157
2                  0.11             0.88               7                    272
3                  0.72             0.87               5                    223
4                  0.37             0.52               2                    159
5                  0.41             0.50               2                    153

   tenure  work_accident  left  promotion_last_5years  salary  \
0       3              0     1                      0       0
2       4              0     1                      0       1
3       5              0     1                      0       0
4       3              0     1                      0       0
5       3              0     1                      0       0

   department_RandD  department_accounting  department_hr  \
0                 0                      0              0
2                 0                      0              0
3                 0                      0              0
4                 0                      0              0
5                 0                      0              0

   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
2                      0                     0                       0
3                      0                     0                       0
4                      0                     0                       0
5                      0                     0                       0

   department_sales  department_support  department_technical
0                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
5                 1                   0                     0
```

Isolate the outcome variable, which is the variable you want your model to predict.

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

```
[29]: 0    1
2    1
3    1
4    1
5    1
```

```
Name: left, dtype: int64
```

Select the features you want to use in your model. Consider which variables will help you predict the outcome variable, `left`.

[30]:
```python
# Select the features you want to use in your model
X = df_logreg.drop('left', axis=1)
# Display the first few rows of the selected features
X.head()
```

[30]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
4                0.37             0.52               2                    159
5                0.41             0.50               2                    153

   tenure  work_accident  promotion_last_5years  salary  department_RandD  \
0       3              0                      0       0                 0
2       4              0                      0       1                 0
3       5              0                      0       0                 0
4       3              0                      0       0                 0
5       3              0                      0       0                 0

   department_accounting  department_hr  department_management  \
0                      0              0                      0
2                      0              0                      0
3                      0              0                      0
4                      0              0                      0
5                      0              0                      0

   department_marketing  department_product_mng  department_sales  \
0                     0                       0                 1
2                     0                       0                 1
3                     0                       0                 1
4                     0                       0                 1
5                     0                       0                 1

   department_support  department_technical
0                   0                     0
2                   0                     0
3                   0                     0
4                   0                     0
5                   0                     0
```

Split the data into training set and testing set. Don't forget to stratify based on the values in `y`, since the classes are unbalanced.

[31]:
```python
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪stratify=y, random_state=42)
```

Construct a logistic regression model and fit it to the training dataset.

[32]:
```python
# Construct a logistic regression model and fit it to the training dataset
log_clf = LogisticRegression(random_state=42, max_iter=500).fit(X_train,␣
↪y_train)
```

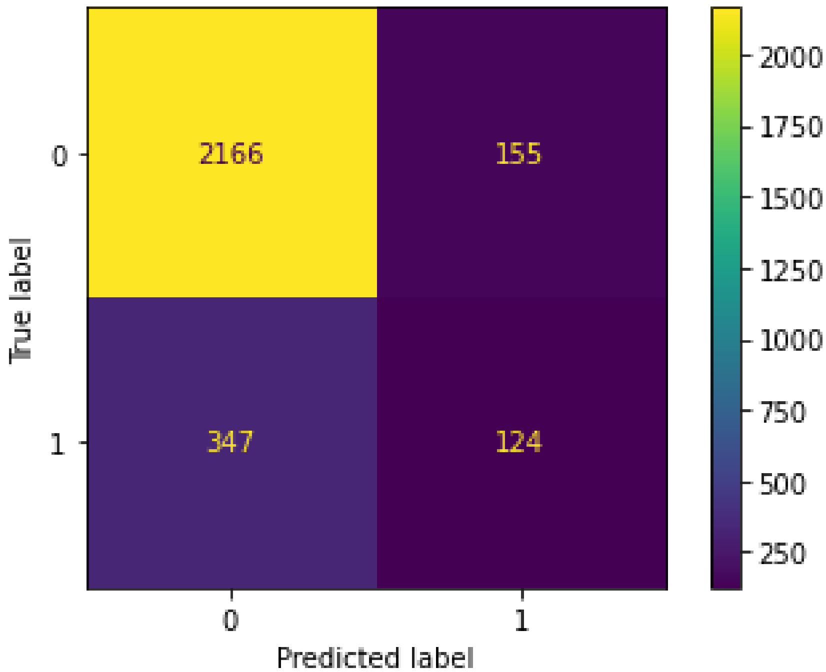
Test the logistic regression model: use the model to make predictions on the test set.

[33]:
```python
# Use the logistic regression model to get predictions on the test set
y_pred = log_clf.predict(X_test)
```

Create a confusion matrix to visualize the results of the logistic regression model.

[34]:
```python
# Compute values for confusion matrix
log_cm = confusion_matrix(y_test, y_pred, labels=log_clf.classes_)
# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm,
                                  display_labels=log_clf.classes_)
# Plot confusion matrix
log_disp.plot(values_format='')
# Display plot
plt
```

[34]:
```
<module 'matplotlib.pyplot' from '/opt/conda/lib/python3.7/site-
packages/matplotlib/pyplot.py'>
```



The plot above shows:

- Upper-left (True Positives): The number of people who left and the model accurately predicted as leaving.
- Upper-right (False Positive): The number of people who didn't leave and the model predicted as leaving.
- The bottom-left(False Negative): The number of people who left and the model predicted as staying.
- The bottom-right(True Negatives): The number of people who stayed and the model accurately predicted as staying.

Create a classification report that includes precision, recall, f1-score, and accuracy metrics to evaluate the performance of the logistic regression model.

Check the class balance in the data. In other words, check the value counts in the `left` column. Since this is a binary classification task, the class balance informs the way you interpret accuracy metrics.

[35]:
```
df_logreg['left'].value_counts(normalize=True)
```

[35]:
```
0    0.831468
1    0.168532
Name: left, dtype: float64
```

There is an 83.14-16.85 split, meaning that the data is not perfectly balanced.

[36]:

```
# Create classification report for logistic regression model
target_names = ['Predicted would not leave', 'Predicted would leave']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                           precision    recall  f1-score   support

Predicted would not leave       0.86      0.93      0.90      2321
    Predicted would leave       0.44      0.26      0.33       471

                 accuracy                           0.82      2792
                macro avg       0.65      0.60      0.61      2792
             weighted avg       0.79      0.82      0.80      2792
```

The classification report above shows that the logistic regression model achieved a precision of 79%, recall of 82%, f1-score of 80% (all weighted averages), and accuracy of 82%. However, if it's most important to predict employees who leave, then the scores are significantly lower.

4.1.4 Modeling Approach B: Tree-based Model

This approach covers implementation of Decision Tree and Random Forest.

Isolate the outcome variable.

[37]:

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

[37]:

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

Select the features

[38]:

```
# Select the features you want to use in your model
X = df_enc.drop('left', axis=1)
# Display the first few rows of the selected features
X.head()
```

[38]:

```
   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_RandD  \
0       3              0                      0       0                 0
1       6              0                      0       1                 0
2       4              0                      0       1                 0
3       5              0                      0       0                 0
4       3              0                      0       0                 0

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

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

   department_support  department_technical
0                   0                     0
1                   0                     0
2                   0                     0
3                   0                     0
4                   0                     0
```

Split the data into training set and testing set.

[39]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪stratify=y, random_state=42)
```

Decision tree - Round 1 Construct a decision tree model and set up cross-validated grid-search to exhuastively search for the best model parameters.

[40]:
```
# Instantiate model
tree = DecisionTreeClassifier(random_state=0)
# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }
# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}
```

```
# Instantiate GridSearch
tree1 = GridSearchCV(tree, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

Fit the decision tree model to the training data.

[41]:
```
%%time
tree1.fit(X_train, y_train)
```

```
CPU times: user 3.08 s, sys: 0 ns, total: 3.08 s
Wall time: 3.08 s
```

[41]:
```
GridSearchCV(cv=4, 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': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'recall', 'f1', 'precision', 'roc_auc', 'accuracy'},
             verbose=0)
```

Identify the optimal values for the decision tree parameters.

[42]:
```
# Check best parameters
tree1.best_params_
```

[42]:
```
{'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 6}
```

Identify the best AUC score achieved by the decision tree model on the training set.

[43]:
```
# Check best AUC score on CV
tree1.best_score_
```

[43]:
```
0.9757623078283226
```

It's a good AUC score, meaning that the model can accurately predict the employes who will leave or not.

Next, you can write a function that will help you extract all the scores from the grid search.

```
[44]: 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, accuracy, or auc

    Returns a pandas df with the F1, recall, precision, accuracy, and auc 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 = {'auc': 'mean_test_roc_auc',
                   'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy'
                  }

    # 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
    auc = best_estimator_results.mean_test_roc_auc
    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

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

    return table
```

Use the function just defined to get all the scores from grid search.

[45]:
```
tree1_cv_results = make_results('decision tree cv', tree1, 'auc')
tree1_cv_results
```

[45]:

	model	precision	recall	F1	accuracy	auc
0	decision tree cv	0.96758	0.918956	0.942626	0.98143	0.975762

The scores obtained from the decission seems strong, since they are above 90%.

Although, this model is sensible to overfitting, therefore, we're going to create a random forest model because it solves this problem.

Random forest - Round 1 Construct a random forest model and set up cross-validated grid-search to exhuastively search for the best model parameters.

[46]:
```
# Instantiate model
rf = RandomForestClassifier(random_state=0)
# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }
# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}
# Instantiate GridSearch
rf1 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

Fit the random forest model to the training data.

[47]:
```
%%time
rf1.fit(X_train, y_train)
```

```
CPU times: user 9min 49s, sys: 0 ns, total: 9min 49s
Wall time: 9min 49s
```

[47]:
```
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,…
                                        verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'recall', 'f1', 'precision', 'roc_auc', 'accuracy'},
             verbose=0)
```

Specify path to where you want to save your model.

[48]:
```python
# Define a path to the folder where you want to save the model
path = '/home/jovyan/work/'
```

Define functions to pickle the model and read in the model.

[49]:
```python
def write_pickle(path, model_object, save_as:str):
    '''
    In:
        path:         path of folder where you want to save the pickle
        model_object: a model you want to pickle
        save_as:      filename for how you want to save the model

    Out: A call to pickle the model in the folder indicated
    '''

    with open(path + save_as + '.pickle', 'wb') as to_write:
        pickle.dump(model_object, to_write)
```

[50]:
```python
def read_pickle(path, saved_model_name:str):
    '''
    In:
        path:             path to folder where you want to read from
        saved_model_name: filename of pickled model you want to read in

    Out:
        model: the pickled model
    '''
    with open(path + saved_model_name + '.pickle', 'rb') as to_read:
        model = pickle.load(to_read)

    return model
```

Use the functions defined above to save the model in a pickle file and then read it in.

[51]:
```
# Write pickle
write_pickle(path, rf1, 'hr_rf1')
```

[52]:
```
# Read pickle
rf1 = read_pickle(path, 'hr_rf1')
```

Identify the best AUC score achieved by the random forest model on the training set.

[53]:
```
# Check best AUC score on CV
rf1.best_score_
```

[53]:
```
0.9819470349290095
```

Identify the optimal values for the parameters of the random forest model.

[54]:
```
# Check best params
rf1.best_params_
```

[54]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 300}
```

Collect the evaluation scores on the training set for the decision tree and random forest models.

[55]:
```
# Get all CV scores
rf1_cv_results = make_results('random forest cv', rf1, 'auc')
print(tree1_cv_results)
print(rf1_cv_results)
```

```
               model  precision    recall        F1  accuracy       auc
0  decision tree cv    0.96758  0.918956  0.942626   0.98143  0.975762
               model  precision    recall        F1  accuracy       auc
0  random forest cv   0.945895  0.912254  0.928698   0.97676  0.981947
```

It seems that the decision tree is slightly better than the random forest mode, except for the auc score.

Define a function that gets all the scores from a model's predictions.

[56]:
```
def get_scores(model_name:str, model, X_test_data, y_test_data):
    '''
    Generate a table of test scores.

    In:
```

```
        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

    Out: pandas df of precision, recall, f1, accuracy, and AUC scores for your␣
↪model
    '''

    preds = model.best_estimator_.predict(X_test_data)

    auc = roc_auc_score(y_test_data, preds)
    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)

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

    return table
```

Now use the best performing model to predict on the test set.

[57]:
```
# Get predictions on test data
rf1_test_scores = get_scores('random forest1 test', rf1, X_test, y_test)
rf1_test_scores
```

[57]:

	model	precision	recall	f1	accuracy	AUC
0	random forest1 test	0.960499	0.927711	0.94382	0.981654	0.960055

The test scores are slightly higher than the validation scores. Then, it appears that this is a good model, meaning that our model on this data is representative of how it will perfomr on new, unseen data.

Feature Engineering Due to high score values obtained from our models, there is a chance that data leakage is occuring. Data leakage is data to train your model that shouldn't be used.

In this case, the company won't have satisfaction levels reported for all of its employees. Another column for data leakage could be `average_monthly_hours`.

The first round of decision tree and random forest models included all variables as features. This next round will incorporate feature engineering to build improved models.

You could proceed by dropping `satisfaction_level` and creating a new feature that roughly captures whether an employee is overworked. You could call this new feature `overworked`. It will be a binary variable.

[58]:
```
# Drop `satisfaction_level` and save resulting dataframe in new variable
df2 = df_enc.drop('satisfaction_level', axis=1)
# Display first few rows of new dataframe
df2.head()
```

[58]:
```
   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  salary  department_RandD  \
0              0     1                      0       0                 0
1              0     1                      0       1                 0
2              0     1                      0       1                 0
3              0     1                      0       0                 0
4              0     1                      0       0                 0

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

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

   department_support  department_technical
0                   0                     0
1                   0                     0
2                   0                     0
3                   0                     0
4                   0                     0
```

[59]:
```
# Create `overworked` column. For now, it's identical to average monthly hours.
df2['overworked'] = df2['average_monthly_hours']
```

```
# Inspect max and min average monthly hours values
print('Max hours', df2['overworked'].max())
print('Min hours', df2['overworked'].min())
```

```
Max hours 310
Min hours 96
```

166.67 is approximately the average number of monthly hours for someone who works 50 weeks per year, 5 days per week, 8 hours per day.

We defined overworked as working more than 175 hours per month on average.

To make the `overworked` column binary, you could reassign the column using a boolean mask. - `df3['overworked'] > 175` creates a series of booleans, consisting of `True` for every value > 175 and `False` for every values 175 - `.astype(int)` converts all `True` to `1` and all `False` to `0`

[60]:
```
# Define `overworked` as working > 175 hrs/week
df2['overworked'] = (df2['overworked'] > 175).astype(int)
# Display first few rows of new column
df2['overworked'].head()
```

[60]:
```
0    0
1    1
2    1
3    1
4    0
Name: overworked, dtype: int64
```

Drop the `average_monthly_hours` column.

[61]:
```
# Drop the `average_monthly_hours` column
df2 = df2.drop('average_monthly_hours', axis=1)
# Display first few rows of resulting dataframe
df2.head()
```

[61]:
```
   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  salary  department_RandD  department_accounting  \
0                      0       0                 0                      0
1                      0       1                 0                      0
2                      0       1                 0                      0
3                      0       0                 0                      0
4                      0       0                 0                      0
```

```
   department_hr  department_management  department_marketing  \
0              0                      0                     0
1              0                      0                     0
2              0                      0                     0
3              0                      0                     0
4              0                      0                     0

   department_product_mng  department_sales  department_support  \
0                       0                 1                   0
1                       0                 1                   0
2                       0                 1                   0
3                       0                 1                   0
4                       0                 1                   0

   department_technical  overworked
0                     0           0
1                     0           1
2                     0           1
3                     0           1
4                     0           0
```

Again, isolate the features and target variables

[62]:
```
# Isolate the outcome variable
y = df2['left']
# Select the features
X = df2.drop('left', axis=1)
```

Split the data into training and testing sets.

[63]:
```
# Create test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪stratify=y, random_state=0)
```

Decision tree - Round 2

[64]:
```
# Instantiate model
tree = DecisionTreeClassifier(random_state=0)
# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }
# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}
# Instantiate GridSearch
tree2 = GridSearchCV(tree, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

[65]:
```
%%time
tree2.fit(X_train, y_train)
```

```
CPU times: user 2.56 s, sys: 0 ns, total: 2.56 s
Wall time: 2.56 s
```

[65]:
```
GridSearchCV(cv=4, 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': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'recall', 'f1', 'precision', 'roc_auc', 'accuracy'},
             verbose=0)
```

[66]:
```
#Check Best Params
tree2.best_params_
```

[66]:
```
{'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 6}
```

[67]:
```
# Check best AUC score on CV
tree2.best_score_
```

[67]:
```
0.9594361127439034
```

The model performs well without the average monthly hours and satisfaction levels.

Next, check the other scores.

[68]:
```
# Get all CV scores
tree2_cv_results = make_results('decision tree2 cv', tree2, 'auc')
print(tree1_cv_results)
print(tree2_cv_results)
```

```
              model  precision    recall        F1  accuracy       auc
0  decision tree cv    0.96758  0.918956  0.942626   0.98143  0.975762
               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.959436
```

The scores fell which is expected since we removed some features. Although, these are good scores.

Random forest - Round 2

[69]:
```
# Instantiate model
rf = RandomForestClassifier(random_state=0)
# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }
# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}
# Instantiate GridSearch
rf2 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

[70]:
```
%%time
rf2.fit(X_train, y_train)
```

```
CPU times: user 7min 28s, sys: 0 ns, total: 7min 28s
Wall time: 7min 28s
```

[70]:
```
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,…
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'recall', 'f1', 'precision', 'roc_auc', 'accuracy'},
             verbose=0)
```

[71]:
```
# Write pickle
write_pickle(path, rf2, 'hr_rf2')
```

[72]:
```
# Read in pickle
rf2 = read_pickle(path, 'hr_rf2')
```

[73]:
```
# Check best params
rf2.best_params_
```

[73]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 3,
 'min_samples_split': 2,
 'n_estimators': 500}
```

[74]:
```
# Get all CV scores
rf2_cv_results = make_results('random forest2 cv', rf2, 'auc')
print(tree2_cv_results)
print(rf2_cv_results)
```

```
               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.959436
               model  precision    recall        F1  accuracy     auc
0  random forest2 cv   0.867692  0.876747  0.871905    0.9573  0.9649
```

Again, the scores fell but this time, there are some scores that are higher in the random forest than the decision tree, which it didn't appear in the first models.

Score the champion model on the test set now.

[75]:
```
# Get predictions on test data
rf2_test_scores = get_scores('random forest2 test', rf2, X_test, y_test)
rf2_test_scores
```

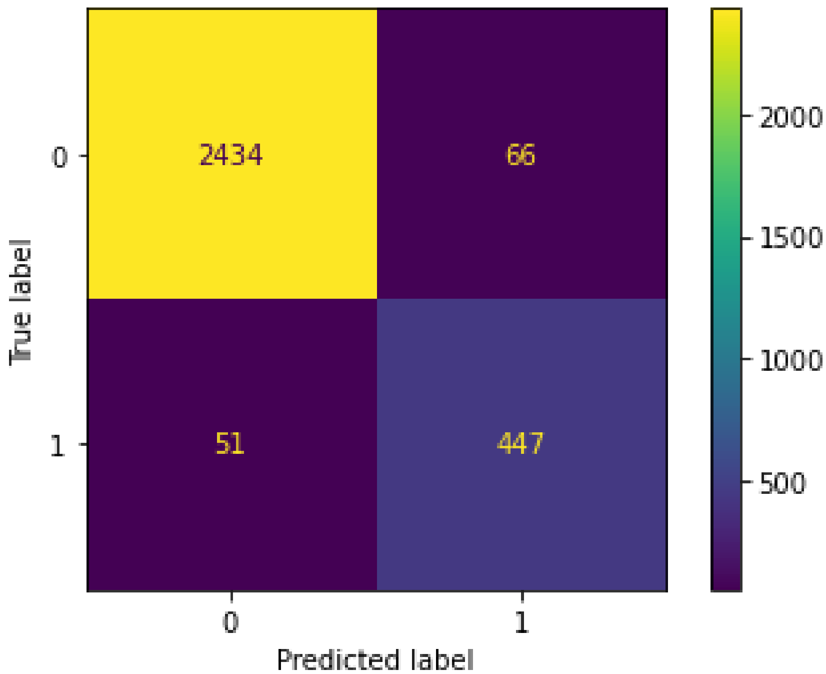
[75]:

	model	precision	recall	f1	accuracy	AUC
0	random forest2 test	0.871345	0.89759	0.884273	0.960974	0.935595

Plot a confusion matrix to visualize how well it predicts on the test set.

[76]:
```
# Generate array of values for confusion matrix
preds = rf2.best_estimator_.predict(X_test)
cm = confusion_matrix(y_test, preds, labels=rf2.classes_)
# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                             display_labels=rf2.classes_)
disp.plot(values_format='')
```

[76]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x76c2f13134d0>

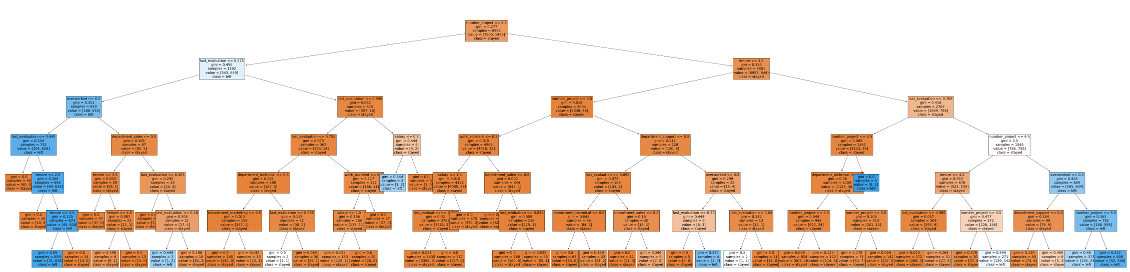


This model predicts more false positives than false negatives, meaning that some employeers may be identified as at risk or getting fired.

We inspect the splits of the decision tree and the important features of random forest.

Decision tree splits

[77]:
```
# Plot the tree
plt.figure(figsize=(82,20))
plot_tree(tree2.best_estimator_, max_depth=6, fontsize=14, feature_names=X.
 ↪columns,
          class_names={0:'stayed', 1:'left'}, filled=True);
plt.show()
```



Decision tree feature importance You can also get feature importance from decision trees (see the DecisionTreeClassifier scikit-learn documentation for details).

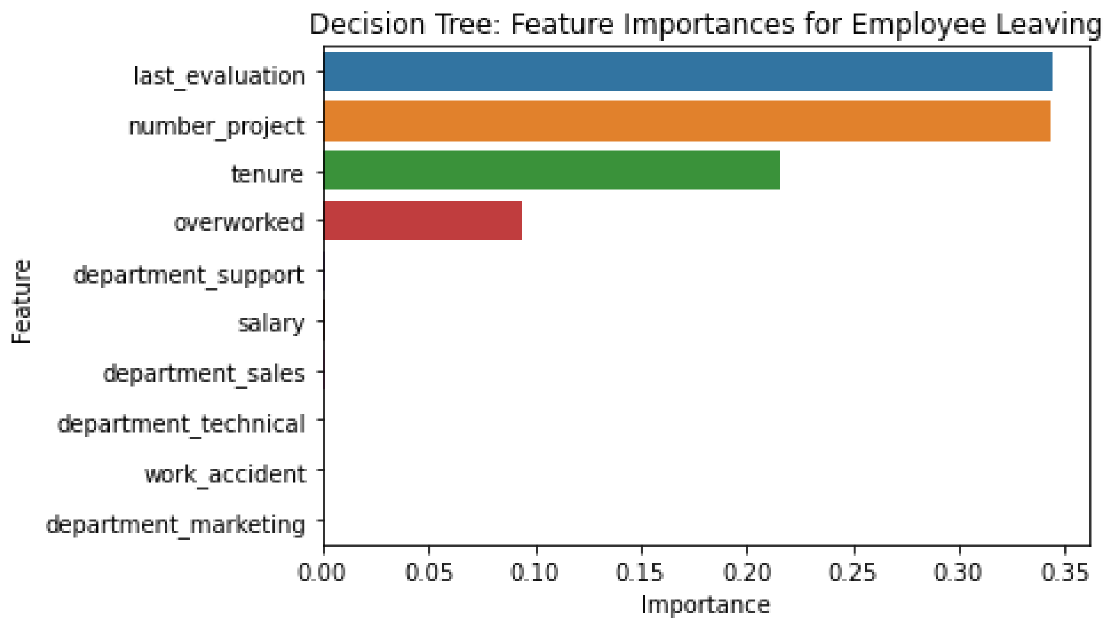
[78]:
```
#tree2_importances = pd.DataFrame(tree2.best_estimator_.feature_importances_,␣
↪columns=X.columns)
tree2_importances = pd.DataFrame(tree2.best_estimator_.feature_importances_,
                                 columns=['gini_importance'],
                                index=X.columns
                                )
tree2_importances = tree2_importances.sort_values(by='gini_importance',␣
↪ascending=False)
# Only extract the features with importances > 0
tree2_importances = tree2_importances[tree2_importances['gini_importance'] != 0]
tree2_importances
```

[78]:

	gini_importance
last_evaluation	0.344043
number_project	0.343470
tenure	0.215627
overworked	0.093521
department_support	0.001142
salary	0.000911
department_sales	0.000607
department_technical	0.000418
work_accident	0.000183
department_marketing	0.000078

You can then create a barplot to visualize the decision tree feature importances.

[79]:
```
sns.barplot(data=tree2_importances, x='gini_importance', y=tree2_importances.
↪index, orient='h')
plt.title("Decision Tree: Feature Importances for Employee Leaving",␣
↪fontsize=12)
plt.ylabel("Feature")
plt.xlabel("Importance")
plt.show()
```



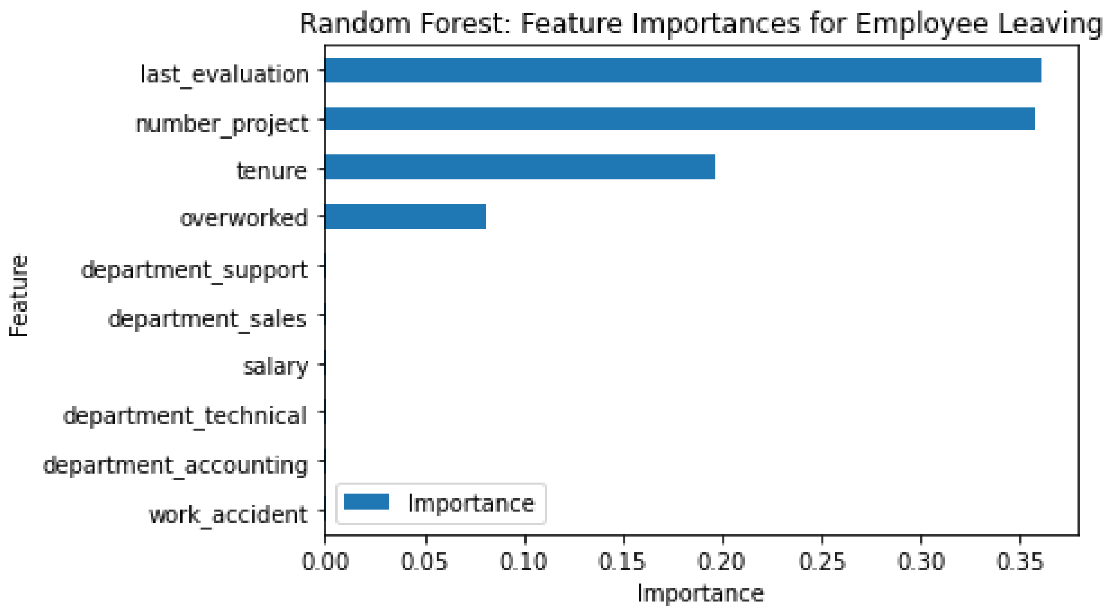
The barplot above shows that the features with the highest importances are `last_evaluation`, `number_project`, `ternure`, `overworked`

4.1.5 Random forest feature importance

Now, plot the feature importances for the random forest model.

[80]:
```
# Get feature importances
feat_impt = rf2.best_estimator_.feature_importances_

# Get indices of top 10 features
ind = np.argpartition(rf2.best_estimator_.feature_importances_, -10)[-10:]

# Get column labels of top 10 features
feat = X.columns[ind]

# Filter `feat_impt` to consist of top 10 feature importances
feat_impt = feat_impt[ind]

y_df = pd.DataFrame({"Feature":feat,"Importance":feat_impt})
y_sort_df = y_df.sort_values("Importance")
fig = plt.figure()
ax1 = fig.add_subplot(111)

y_sort_df.plot(kind='barh',ax=ax1,x="Feature",y="Importance")
```

```
ax1.set_title("Random Forest: Feature Importances for Employee Leaving",␣
↪fontsize=12)
ax1.set_ylabel("Feature")
ax1.set_xlabel("Importance")

plt.show()
```



The plot above shows that in this random forest model, `last_evaluation`, `number_project`, `tenure`, and `overworked` have the highest importance, in that order. These variables are most helpful in predicting the outcome variable, `left`, and they are the same as the ones used by the decision tree model.

5 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?

Double-click to enter your responses here.

5.1 Step 4. Results and Evaluation

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

5.1.1 Summary of model results

[Double-click to enter your summary here.]

5.1.2 Conclusion, Recommendations, Next Steps

[Double-click to enter your conclusion, recommendations, and next steps here.]

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.