Providing Suggestions for HR at Salifort Motors

September 13, 2024

1 Providing data-driven suggestions for HR

This project aims to help **Salifort Motors HR department** reduce employee turnover by identifying factors contributing to attrition. Using employee data, we will perform **exploratory data analysis** and build **machine learning models** to predict which employees are likely to leave. The insights will guide HR in making data-driven decisions to improve employee satisfaction and retention.

1.1 Pace: Plan

1.1.1 Understanding 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 me as a data analytics professional and ask me to provide data-driven suggestions based on my understanding of the data. They have the following question: what's likely to make the employee leave the company?

My 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 we 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.

1.1.2 Familiarizing ourselves with the HR dataset

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

Note: Link to data: 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)

Variable	Description
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)

1.2 1. Imports

- Import packages
- Load dataset

1.2.1 Import packages

[1]:
```python
# Import packages
# 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
```

```
/home/ismail/.local/lib/python3.8/site-packages/xgboost/core.py:265:
FutureWarning: Your system has an old version of glibc (< 2.28). We will stop
supporting Linux distros with glibc older than 2.28 after **May 31, 2025**.
Please upgrade to a recent Linux distro (with glibc 2.28+) to use future
versions of XGBoost.
Note: You have installed the 'manylinux2014' variant of XGBoost. Certain
features such as GPU algorithms or federated learning are not available. To use
these features, please upgrade to a recent Linux distro with glibc 2.28+, and
install the 'manylinux_2_28' variant.
  warnings.warn(
```

1.2.2 Load dataset

[2]:
```python
# Load dataset into a dataframe
df0 = pd.read_csv("HR_capstone_dataset.csv")
```

```
# The first few rows of the dataframe
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
```

1.3 2. Data Exploration (Initial EDA and data cleaning)

- Understanding the variables
- Cleaning the dataset (missing data, redundant data, outliers)

1.3.1 Gathering basic information about the data

[65]:
```
# Basic information about the data
df0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_monthly_hours  14999 non-null  int64
 4   tenure                 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
```

1.3.2 The number of rows and columns in the dataset.

[197]:
```
df0.shape
```

[197]:
```
(14999, 10)
```

1.3.3 Gathering descriptive statistics about the data

[4]:
```
# Descriptive statistics about the data
df0.describe()
```

[4]:
```
       satisfaction_level  last_evaluation  number_project  \
count        14999.000000     14999.000000    14999.000000
mean             0.612834         0.716102        3.803054
std              0.248631         0.171169        1.232592
min              0.090000         0.360000        2.000000
25%              0.440000         0.560000        3.000000
50%              0.640000         0.720000        4.000000
75%              0.820000         0.870000        5.000000
max              1.000000         1.000000        7.000000

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

       promotion_last_5years
count           14999.000000
mean                0.021268
std                 0.144281
min                 0.000000
25%                 0.000000
50%                 0.000000
75%                 0.000000
max                 1.000000
```

1.3.4 Renaming columns

Standardizing the column names so that they are all in `snake_case`.

[5]:
```
# Displaying all column names
df0.columns
```

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

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

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

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

1.3.5 Checking missing values

Checking for any missing values in the data.

[7]:
```
df0.isna().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
```

Note There does not seem to be any missing values.

1.3.6 Checking duplicates

Checking for any duplicate entries in the data.

[8]:
```python
# Checking for duplicates
df0.duplicated().sum()
```

[8]: 3008

[9]:
```python
# Inspecting some rows containing duplicates as needed
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
```

The above output shows the first five occurences of rows that are duplicated farther down in the dataframe. How likely is it that these are legitimate entries? In other words, how plausible is it that two employees self-reported the exact same response for every column?

We could perform a likelihood analysis by essentially applying Bayes' theorem and multiplying the probabilities of finding each value in each column, but this does not seem necessary. With several continuous variables across 10 columns, it seems very unlikely that these observations are legitimate. We can proceed by dropping them.

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


# The first few rows of new dataframe as needed
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
```

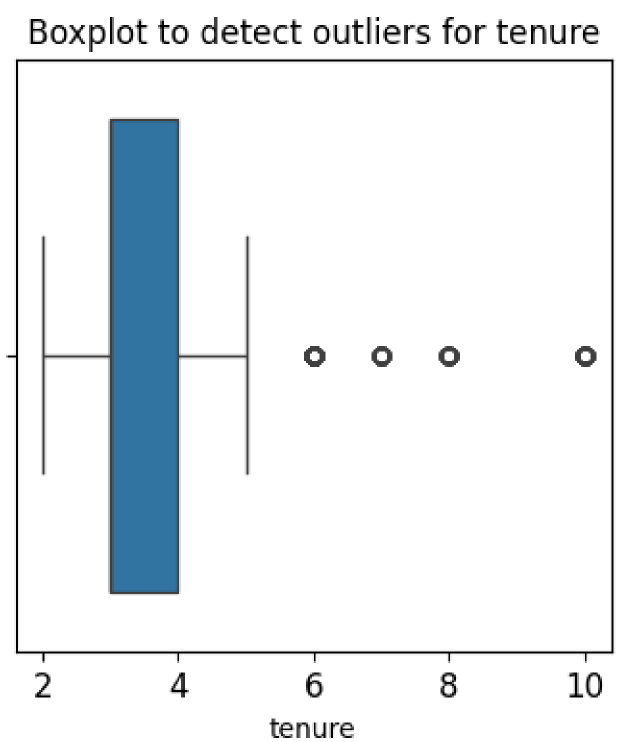
2	0.11	0.88	7	272
3	0.72	0.87	5	223
4	0.37	0.52	2	159

	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

1.3.7 Checking outliers

Checking for outliers in the data.

[11]:
```
# Creating a boxplot to visualize distribution of `tenure` and detect any␣
↪outliers
plt.figure(figsize=(4,4))
plt.title('Boxplot to detect outliers for tenure', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=df1['tenure'])
plt.show()
```



[12]:
```python
# Determining the number of rows containing outliers
# Compute the 25th percentile value in `tenure`
perc25 = df1['tenure'].quantile(0.25)
# Compute the 75th percentile value in `tenure`
perc75 = df1['tenure'].quantile(0.75)
# Compute the interquartile range in `tenure`
iqr = perc75 - perc25
upper_limit = perc25 + 1.5 * iqr
lower_limit = perc75 - 1.5 * iqr
print("Lower limit:", lower_limit)
print("Upper limit:", upper_limit)
outliers = df1[(df1['tenure'] > upper_limit) | (df1['tenure'] < lower_limit)]

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

```
Lower limit: 2.5
Upper limit: 4.5
Number of rows in the data containing outliers in `tenure`: 4796
```

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

2 PACE: Analyze Stage

- We will Perform EDA (analyze relationships between variables)

2.1 2. Data Exploration (Continue EDA)

We will begin by understanding how many employees left and what percentage of all employees this figure represents.

[32]:
```python
# Get numbers of people who left vs. stayed

print(df1['left'].value_counts())
# Get percentages of people who left vs. stayed
df1['left'].value_counts(normalize=True)
```

```
left
0    10000
1     1991
Name: count, dtype: int64
```

[32]:
```
left
0    0.833959
1    0.166041
```

```
Name: proportion, dtype: float64
```

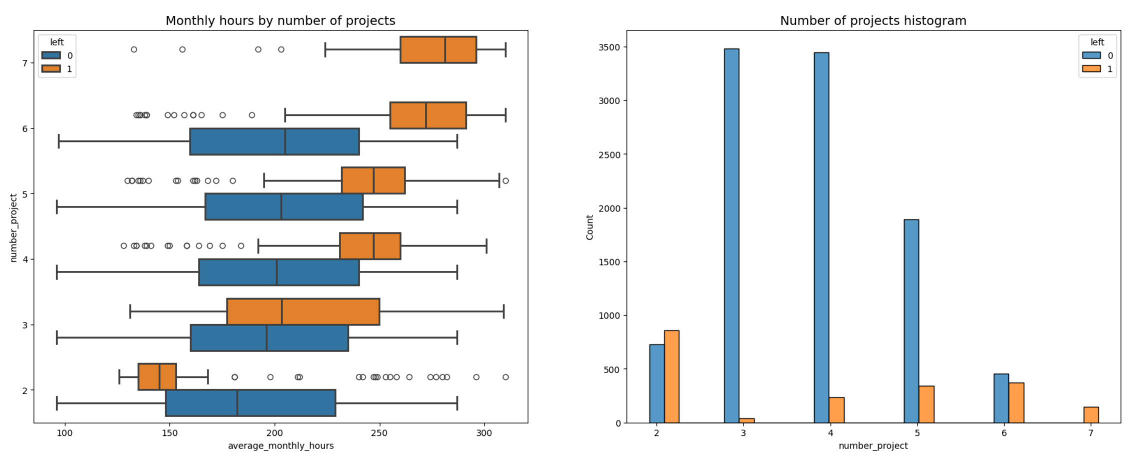
2.1.1 Data visualizations

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

First we start by creating a stacked boxplot showing `average_monthly_hours` distributions for `number_project`, comparing the distributions of employees who stayed versus those who left.

We will also plot a stacked histogram to visualize the distribution of `number_project` for those who stayed and those who left.

[62]:
```python
fig, ax = plt.subplots(1, 2, figsize = (22,8))

# 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], linewidth=2)
ax[0].invert_yaxis()
ax[0].set_title('Monthly hours by number of projects', fontsize='14')

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()
```



It might be natural that people who work on more projects would also work longer hours. This appears to be the case here, with the mean hours of each group (stayed and left) increasing with number of projects worked. However, a few things stand out from this plot.

1. There are two groups of employees who left the company: (A) those who worked considerably less than their peers with the same number of projects, and (B) those who worked much more. Of those in group A, it's possible that they were fired. It's also possible that this group includes employees who had already given their notice and were assigned fewer hours because they were already on their way out the door. For those in group B, it's reasonable to infer that they probably quit. The folks in group B likely contributed a lot to the projects they worked in; they might have been the largest contributors to their projects.
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. The ratio of left/stayed is very small for these cohorts.
4. If we 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`. This means that, aside from the employees who worked on two projects, every group—even those who didn't leave the company—worked considerably more hours than this. It seems that employees here are overworked.

As the next step, we will confirm that all employees with seven projects left.

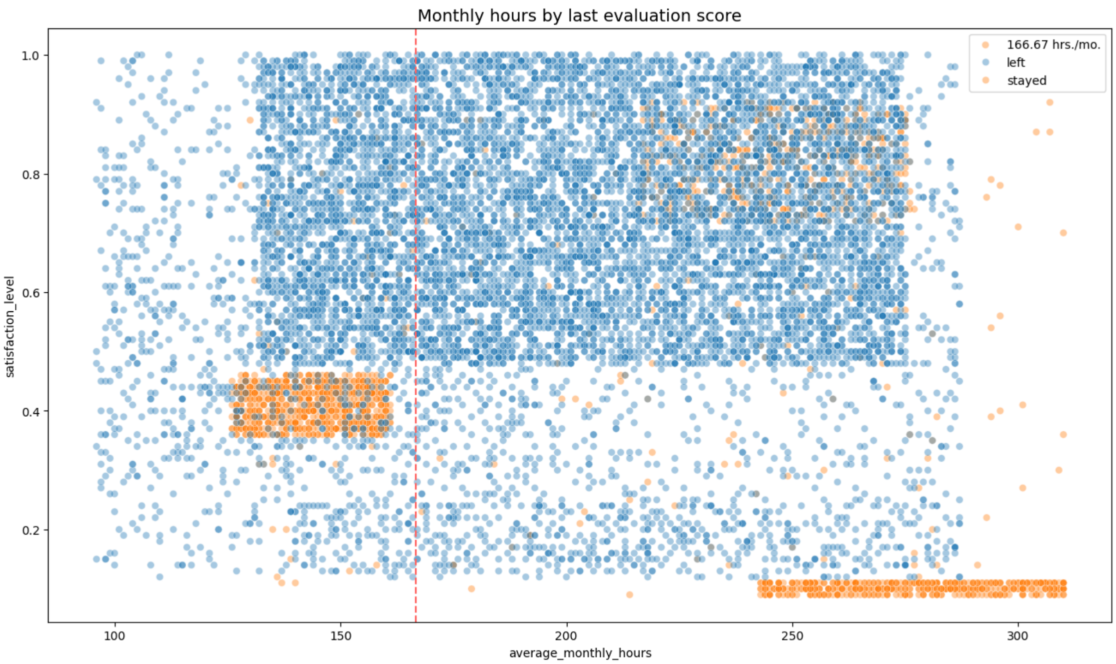
[64]:
```python
# Get value counts of stayed/left for employees with 7 projects
df1[df1['number_project']==7]['left'].value_counts()
```

[64]:
```
left
1    145
Name: count, dtype: int64
```

This confirms that all employees with 7 projects did leave.

Next, we will examine the average monthly hours versus the satisfaction levels.

[90]:
```python
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');
```



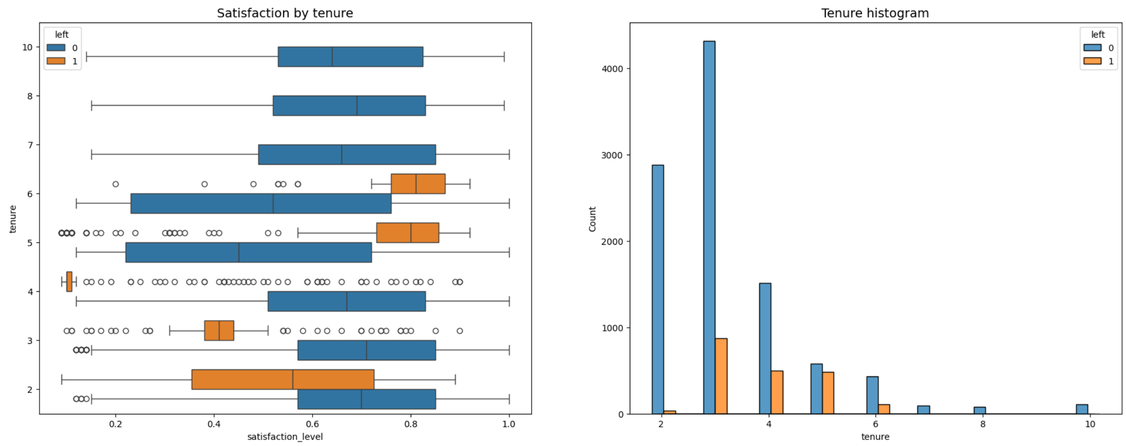
The following observations can be made from the scatterplot above:

- The scatterplot indicates two groups of employees who left: overworked employees who performe
- There seems to be a correlation between hours worked and evaluation score.
- There isn't a high percentage of employees in the upper left quadrant of this plot; but worki
- Most of the employees in this company work well over 167 hours per month.

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

[114]:
```
# 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
tenure_stay = df1[df1['left']==0]['tenure']
tenure_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')
```

```
plt.show();
```



There are many observations we could make from this plot. - Employees who left fall into two general categories: dissatisfied employees with shorter tenures and very satisfied employees with medium-length tenures. - Four-year employees who left seem to have an unusually low satisfaction level. It's worth investigating changes to company policy that might have affected people specifically at the four-year mark, if possible. - 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.

As the next step in analyzing the data, we will calculate the mean and median satisfaction scores of employees who left and those who didn't.

[115]:
```
# Mean and median satisfaction scores of employees who left and those who stayed
df1.groupby(['left'])['satisfaction_level'].agg([np.mean,np.median])
```

[115]:

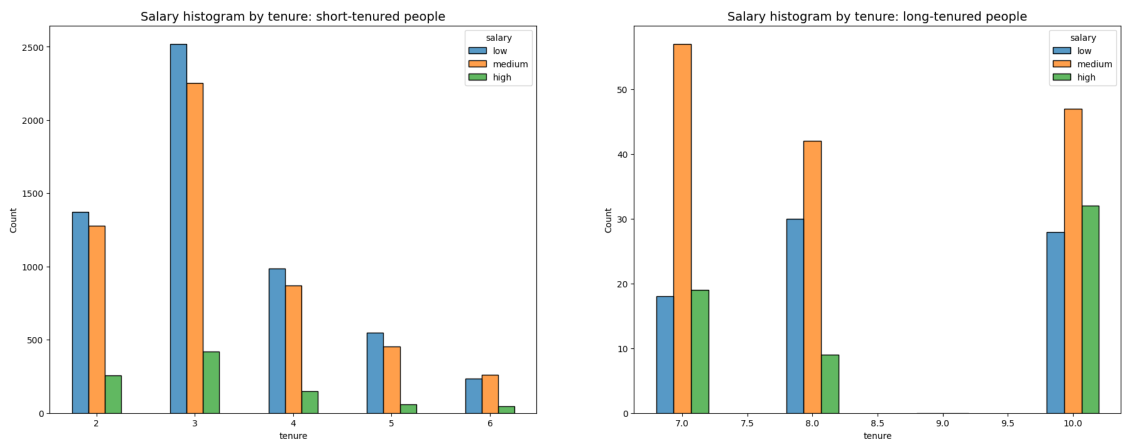
left	mean	median
0	0.667365	0.69
1	0.440271	0.41

As expected, the mean and median satisfaction scores of employees who left are lower than those of employees who stayed. Interestingly, among employees who stayed, the mean satisfaction score appears to be slightly below the median score. This indicates that satisfaction levels among those who stayed might be skewed to the left.

Next, we will examine salary levels for different tenures.

[116]:
```
# 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=.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=.4, ax=ax[1])
ax[1].set_title('Salary histogram by tenure: long-tenured people', fontsize='14');
```

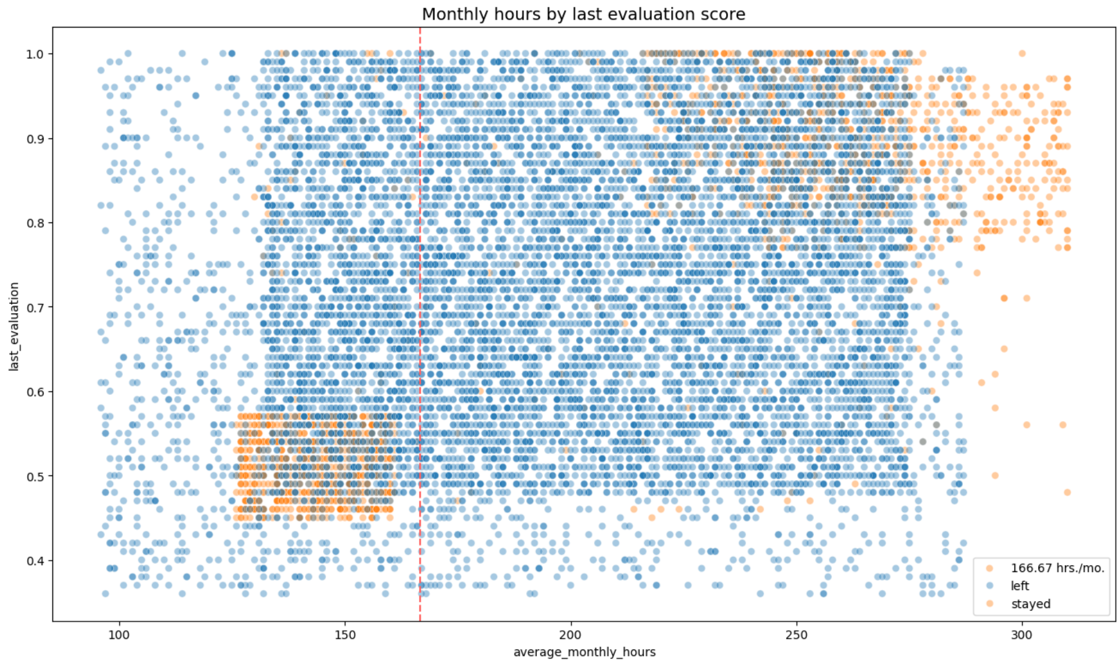


The plots above show that long-tenured employees were not disproportionately comprised of higher-paid employees.

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

[117]:
```
# 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='--')
```

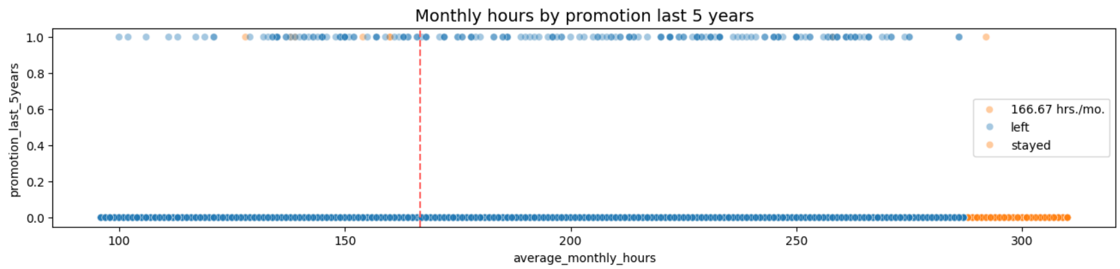
```
plt.legend(labels=['166.67 hrs./mo.', 'left', 'stayed'])
plt.title('Monthly hours by last evaluation score', fontsize='14');
```



The following observations can be made from the scatterplot above: - The scatterplot indicates two groups of employees who left: overworked employees who performed very well and employees who worked slightly under the nominal monthly average of 166.67 hours with lower evaluation scores. - There seems to be a correlation between hours worked and evaluation score. - There isn't a high percentage of employees in the upper left quadrant of this plot; but working long hours doesn't guarantee a good evaluation score. - Most of the employees in this company work well over 167 hours per month.

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

[95]:
```
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 5 years', fontsize='14');
```



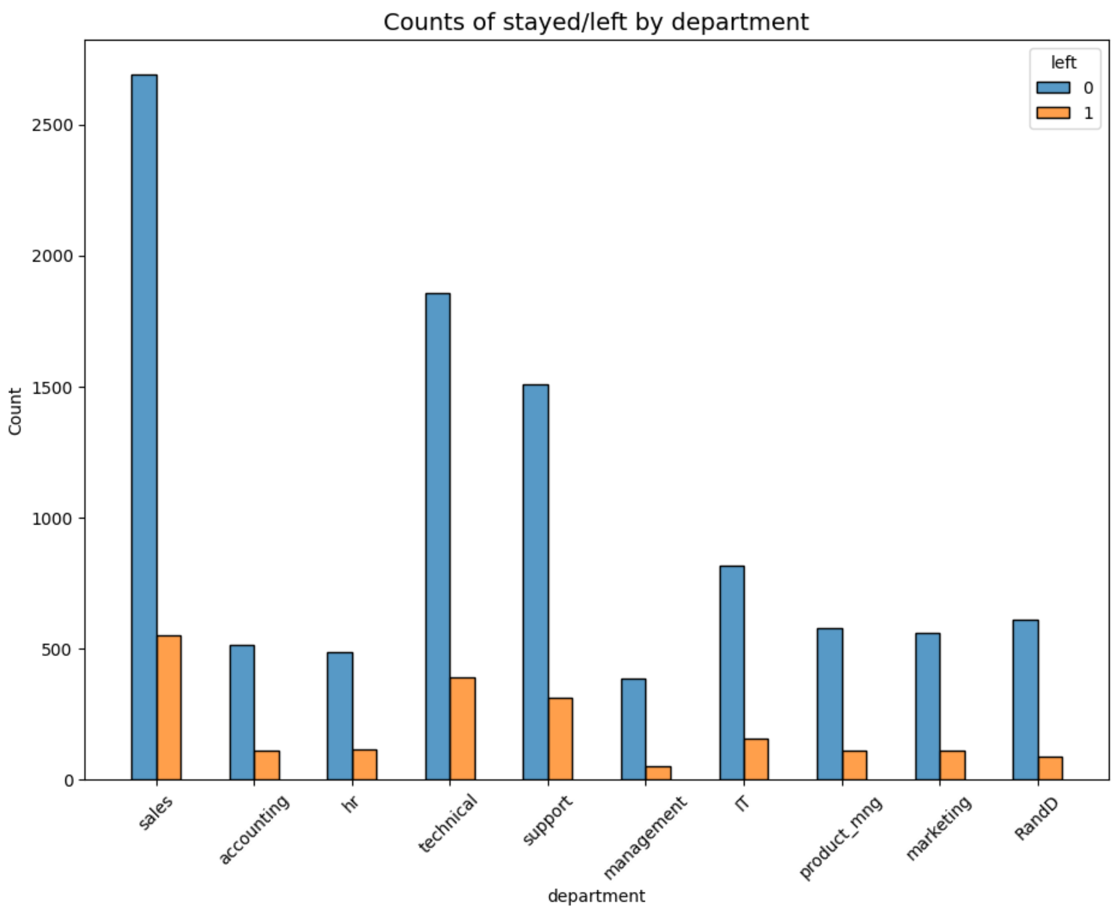
The plot above shows the following: - very few employees who were promoted in the last five years left - very few employees who worked the most hours were promoted - all of the employees who left were working the longest hours

Next, we will inspect how the employees who left are distributed across departments.

[96]:
```
df1["department"].value_counts()
```

[96]:
```
department
sales          3239
technical      2244
support        1821
IT              976
RandD           694
product_mng     686
marketing       673
accounting      621
hr              601
management      436
Name: count, dtype: int64
```

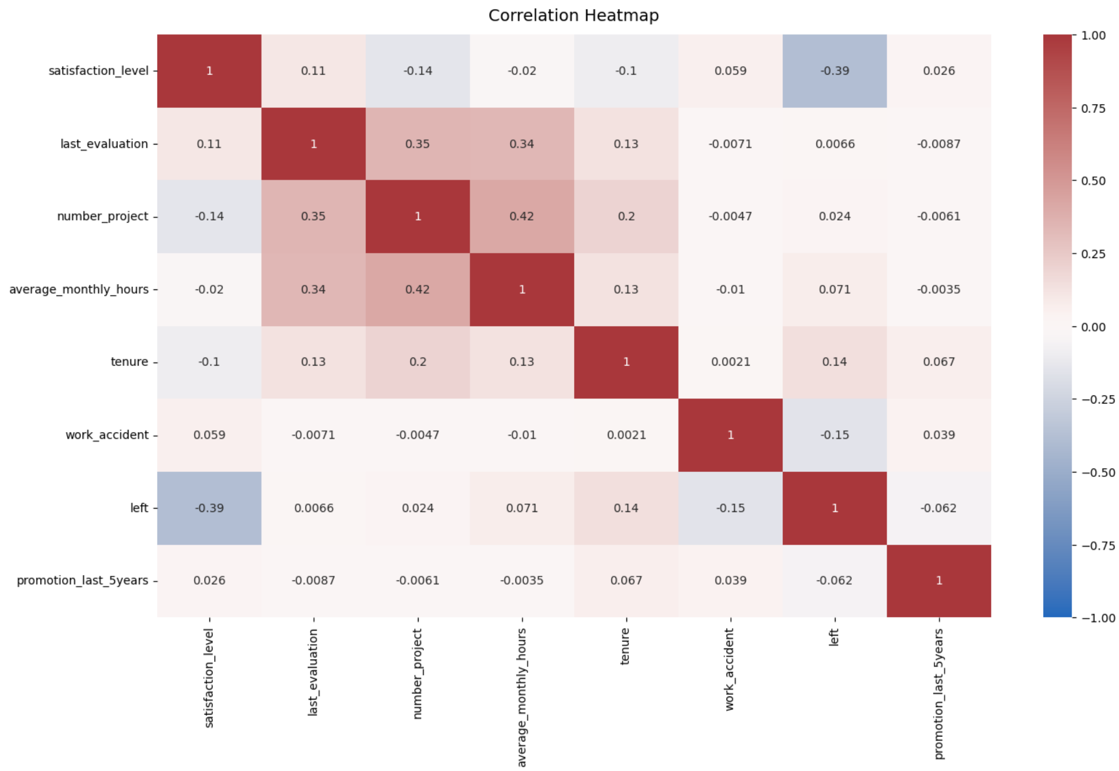
[110]:
```
# Stacked histogram to compare department distribution of employees who left to
↪that of employees who didn't
plt.figure(figsize=(11,8))
sns.histplot(data=df1, x='department', hue='left', discrete=1,
             hue_order=[0, 1], multiple='dodge', shrink=.5)
plt.xticks(rotation=45)
plt.title('Counts of stayed/left by department', fontsize=14);
```



There doesn't seem to be any department that differs significantly in its proportion of employees who left to those who stayed.

Lastly, we will check for strong correlations between variables in the data.

[107]:
```
plt.figure(figsize=(16, 9))
heatmap = sns.heatmap(df0.corr(numeric_only=True),
                      vmin=-1,
                      vmax=1,
                      annot=True,
                      cmap=sns.color_palette("vlag", as_cmap=True))
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':14}, pad=12);
```



The correlation heatmap confirms that the number of projects, monthly hours, and evaluation scores all have some positive correlation with each other, and whether an employee leaves is negatively correlated with their satisfaction level.

2.1.2 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.

3 paCe: Construct Stage

3.0.1 Modeling

This approach covers implementation of Logistic Regression.

3.0.2 Modeling Approach A: Logistic regression

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

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

`department` is a categorical variable, which means we 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.

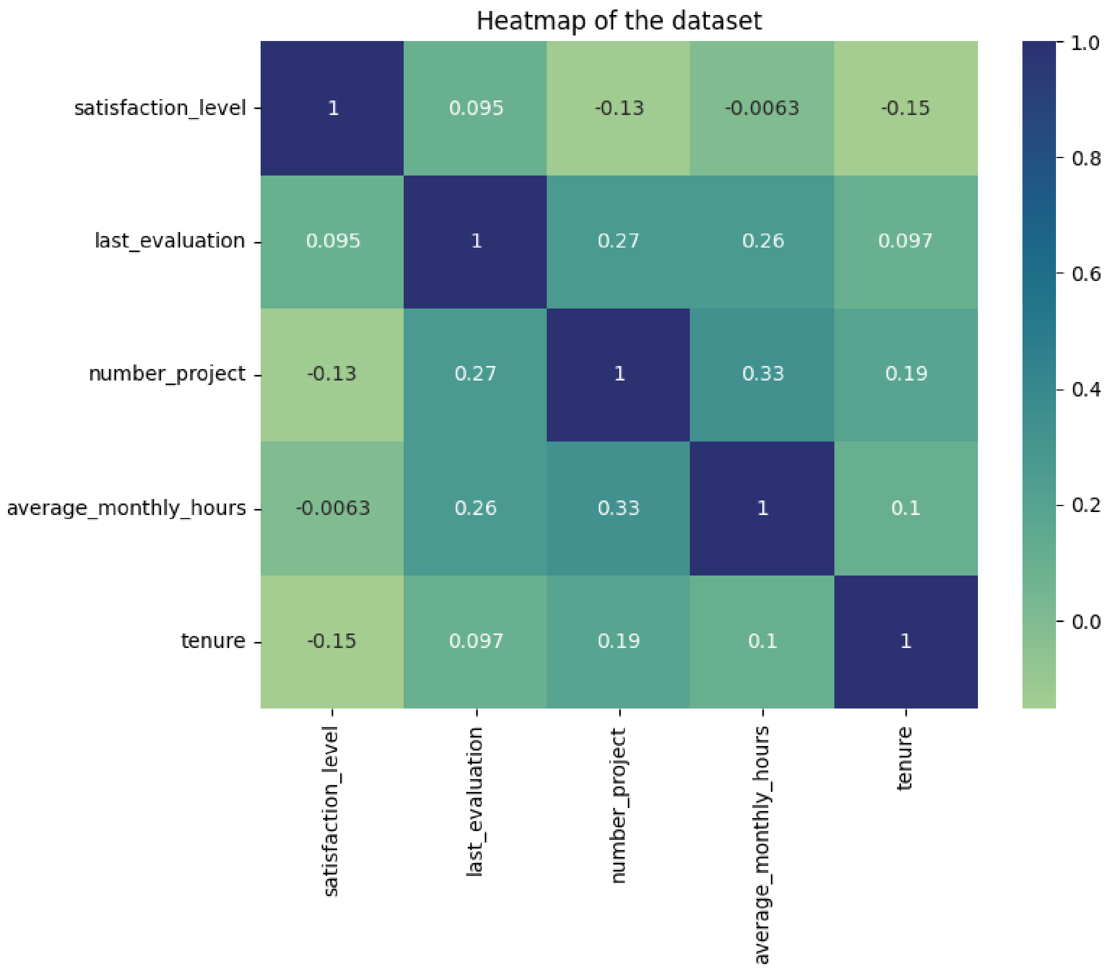
[119]:
```python
df_enc = df1.copy()

# 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=False)

df_enc.head()
```

[119]:
```
   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  department_IT  \
0       3              0     1                      0       0          False
1       6              0     1                      0       1          False
2       4              0     1                      0       1          False
3       5              0     1                      0       0          False
4       3              0     1                      0       0          False

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

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

   department_sales  department_support  department_technical
0              True               False                 False
```

```
1           True            False               False
2           True            False               False
3           True            False               False
4           True            False               False
```

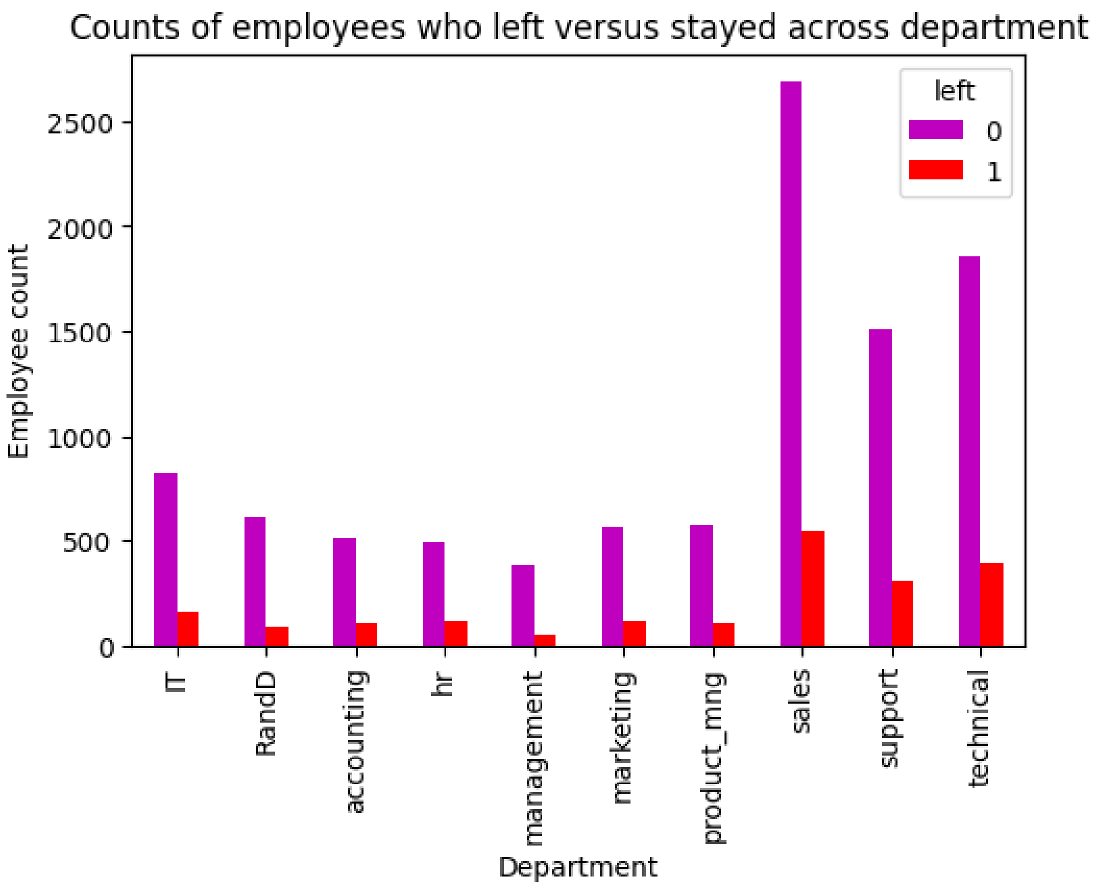
Heatmap to visualize how correlated variables are.

```
[120]: 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 dataset')
plt.show()
```



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

[202]:
```
pd.crosstab(df1['department'], df1['left']).plot(kind ='bar',color='mr',␣
↪figsize=(6,4))
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.

[203]:
```
# 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)]
df_logreg.head()
```

[203]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
2                0.11             0.88               7                    272
```

```
4              0.37              0.52               2                     159
5              0.41              0.50               2                     153
6              0.10              0.77               6                     247

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

   department_RandD  department_accounting  department_hr  \
0             False                  False          False
2             False                  False          False
4             False                  False          False
5             False                  False          False
6             False                  False          False

   department_management  department_marketing  department_product_mng  \
0                  False                 False                   False
2                  False                 False                   False
4                  False                 False                   False
5                  False                 False                   False
6                  False                 False                   False

   department_sales  department_support  department_technical
0              True               False                 False
2              True               False                 False
4              True               False                 False
5              True               False                 False
6              True               False                 False
```

Isolating the outcome variable.

[204]:
```
y = df_logreg['left']
y.head()
```

[204]:
```
0    1
2    1
4    1
5    1
6    1
Name: left, dtype: int64
```

Selecting the features we want to use in the model.

[206]:
```
X = df_logreg.drop('left', axis=1)
X.head()
```

[206]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
2                0.11             0.88               7                    272
4                0.37             0.52               2                    159
5                0.41             0.50               2                    153
6                0.10             0.77               6                    247

   tenure  work_accident  promotion_last_5years  salary  department_IT  \
0       3              0                      0       0          False
2       4              0                      0       1          False
4       3              0                      0       0          False
5       3              0                      0       0          False
6       4              0                      0       0          False

   department_RandD  department_accounting  department_hr  \
0             False                  False          False
2             False                  False          False
4             False                  False          False
5             False                  False          False
6             False                  False          False

   department_management  department_marketing  department_product_mng  \
0                  False                 False                   False
2                  False                 False                   False
4                  False                 False                   False
5                  False                 False                   False
6                  False                 False                   False

   department_sales  department_support  department_technical
0              True               False                 False
2              True               False                 False
4              True               False                 False
5              True               False                 False
6              True               False                 False
```

Splitting the data into training set and testing set.

[207]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
 ↪stratify=y, random_state=42)
```

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

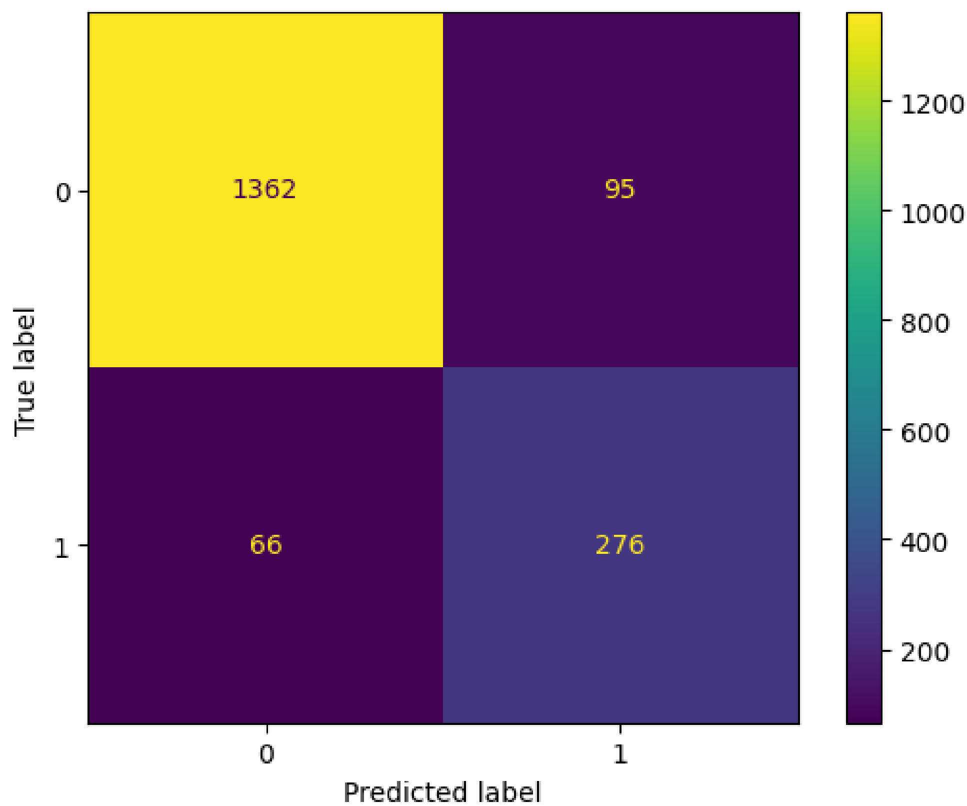
[208]:
```
log_clf = LogisticRegression(random_state=42, max_iter=500).fit(X_train,
 ↪y_train)
```

Testing the logistic regression model: we will use the model to make predictions on the test set.

[209]:
```
y_pred = log_clf.predict(X_test)
```

Confusion matrix to visualize the results of the logistic regression model.

[214]:
```
# 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.show()
```



The upper-left quadrant displays the number of true negatives. The upper-right quadrant displays the number of false positives. The bottom-left quadrant displays the number of false negatives. The bottom-right quadrant displays the number of true positives.

True negatives: The number of people who did not leave that the model accurately predicted did not leave.

False positives: The number of people who did not leave the model inaccurately predicted as leaving.

False negatives: The number of people who left that the model inaccurately predicted did not leave

True positives: The number of people who left the model accurately predicted as leaving

A perfect model would yield all true negatives and true positives, and no false negatives or false positives.

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

First, we will Check the class balance in the data.

[215]:
```
df_logreg['left'].value_counts(normalize=True)
```

[215]:
```
left
0    0.809729
1    0.190271
Name: proportion, dtype: float64
```

There is an approximately 83%-17% split. So the data is not perfectly balanced, but it is not too imbalanced. If it was more severely imbalanced, we might want to resample the data to make it more balanced. In this case, we can use this data without modifying the class balance and continue evaluating the model.

[216]:
```
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       0.95      0.93      0.94      1457
    Predicted would leave       0.74      0.81      0.77       342

                 accuracy                           0.91      1799
                macro avg       0.85      0.87      0.86      1799
             weighted avg       0.91      0.91      0.91      1799
```

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.

3.0.3 Modeling Approach B: Tree-based Model

This approach covers implementation of Decision Tree and Random Forest.

Isolating the outcome variable

[122]:
```
y = df_enc['left']
y.head()
```

```
[122]: 0    1
       1    1
       2    1
       3    1
       4    1
       Name: left, dtype: int64
```

Selecting the features

```python
[124]: X = df_enc.drop('left', axis=1)
       X.head()
```

```
[124]:    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          False
       1       6              0                      0       1          False
       2       4              0                      0       1          False
       3       5              0                      0       0          False
       4       3              0                      0       0          False

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

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

          department_sales  department_support  department_technical
       0              True               False                 False
       1              True               False                 False
       2              True               False                 False
       3              True               False                 False
       4              True               False                 False
```

Split the data into training, validating, and testing sets.

```
[125]: 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 1 Constructing a decision tree model and setting up cross-validated grid-search to exhuastively search for the best model parameters.

```
[138]: 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': 'accuracy', 'precision':'precision', 'recall':'recall',␣
        ↪'f1':'f1', 'roc_auc':'roc_auc'}

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

Fitting the decision tree model to the training data.

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

```
CPU times: user 3.34 s, sys: 23.7 ms, total: 3.36 s
Wall time: 3.38 s
```

```
[139]: GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=0),
                    param_grid={'max_depth': [4, 6, 8, None],
                                'min_samples_leaf': [2, 5, 1],
                                'min_samples_split': [2, 4, 6]},
                    refit='roc_auc',
                    scoring={'accuracy': 'accuracy', 'f1': 'f1',
                             'precision': 'precision', 'recall': 'recall',
                             'roc_auc': 'roc_auc'})
```

Identifying the optimal values for the decision tree parameters.

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

```
[141]: {'max_depth': 4, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

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

[143]:
```python
# Checking best AUC score on CV
tree1.best_score_
```

[143]: 0.969819392792457

This is a strong AUC score, which shows that this model can predict employees who will leave very well.

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

[144]:
```python
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
```

[145]:
```
# Get all CV scores
tree1_cv_results = make_results('decision tree cv', tree1, 'auc')
tree1_cv_results
```

[145]:

	model	precision	recall	F1	accuracy	auc
0	decision tree cv	0.914552	0.916949	0.915707	0.971978	0.969819

All of these scores from the decision tree model are strong indicators of good model performance.

Recall that decision trees can be vulnerable to overfitting, and random forests avoid overfitting by incorporating multiple trees to make predictions. We could construct a random forest model next.

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

[128]:
```
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': 'accuracy', 'precision':'precision', 'recall':'recall',␣
 ↪'f1':'f1', 'roc_auc':'roc_auc'}

# Instantiate GridSearch
rf1 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

Fitting the random forest model to the training data.

[129]:
```
%%time
rf1.fit(X_train, y_train)
```

```
CPU times: user 18min 1s, sys: 243 ms, total: 18min 1s
Wall time: 18min 4s
```

[129]:
```
GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=0),
             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]},
             refit='roc_auc',
             scoring={'accuracy': 'accuracy', 'f1': 'f1',
                      'precision': 'precision', 'recall': 'recall',
                      'roc_auc': 'roc_auc'})
```

saving themodel.

[131]:
```python
path = './'
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)

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

# Write pickle
write_pickle(path, rf1, 'hr_rf1')
# Read pickle
rf1 = read_pickle(path, 'hr_rf1')
```

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

[132]:
```
# Checking best AUC score on CV
rf1.best_score_
```

[132]: 0.9804250949807172

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

[133]:
```
# Checking best params
rf1.best_params_
```

[133]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 500}
```

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

[146]:
```
# 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   0.914552  0.916949  0.915707  0.971978  0.969819
               model  precision    recall        F1  accuracy       auc
0  random forest cv   0.950023  0.915614  0.932467  0.977983  0.980425
```

The evaluation scores of the random forest model are better than those of the decision tree model, with the exception of recall (the recall score of the random forest model is approximately 0.001 lower, which is a negligible amount). This indicates that the random forest model mostly outperforms the decision tree model.

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

[ ]:
```
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 we use the best performing model to predict on the test set.

[149]:
```
# Get predictions on test data
rf1_test_scores = get_scores('random forest1 test', rf1, X_test, y_test)
rf1_test_scores
```

[149]:

	model	precision	recall	f1	accuracy	AUC
0	random forest1 test	0.964211	0.919679	0.941418	0.980987	0.956439

The test scores are very similar to the validation scores, which is good. This appears to be a strong model. Since this test set was only used for this model, we can be more confident that your model's performance on this data is representative of how it will perform on new, unseeen data.

Feature Engineering 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.

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

[150]:
```
# Drop `satisfaction_level` and save resulting dataframe in new variable
df2 = df_enc.drop('satisfaction_level', axis=1)

df2.head()
```

[150]:
```
   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_IT  \
0              0     1                      0       0          False
1              0     1                      0       1          False
2              0     1                      0       1          False
3              0     1                      0       0          False
4              0     1                      0       0          False

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

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

   department_sales  department_support  department_technical
0              True               False                 False
1              True               False                 False
2              True               False                 False
3              True               False                 False
4              True               False                 False
```

[151]:

```
# 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 could define being overworked as working more than 175 hours per month on average.

To make the `overworked` column binary, we| 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`

[152]:
```
# Define `overworked` as working > 175 hrs/week
df2['overworked'] = (df2['overworked'] > 175).astype(int)

df2['overworked'].head()
```

[152]:
```
0    0
1    1
2    1
3    1
4    0
Name: overworked, dtype: int64
```

Drop the average_monthly_hours column.

[153]:
```
# Drop the `average_monthly_hours` column
df2 = df2.drop('average_monthly_hours', axis=1)

df2.head()
```

[153]:
```
   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_IT  department_RandD  \
0                      0       0          False             False
1                      0       1          False             False
2                      0       1          False             False
3                      0       0          False             False
4                      0       0          False             False

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

   department_marketing  department_product_mng  department_sales  \
0                 False                   False              True
1                 False                   False              True
2                 False                   False              True
3                 False                   False              True
4                 False                   False              True
```

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

Again, isolating the features and target variables, Splitting the data into training and testing sets.

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

# Select the features
X = df2.drop('left', axis=1)

# 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

```
[157]: tree = RandomForestClassifier(random_state=0)
# Assigning 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]
             }
# Assigning a dictionary of scoring metrics to capture
scoring = {'accuracy': 'accuracy', 'precision':'precision', 'recall':'recall',␣
 ↪'f1':'f1', 'roc_auc':'roc_auc'}

# Instantiating GridSearch
tree2 = GridSearchCV(tree, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

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

```
CPU times: user 40.2 s, sys: 239 ms, total: 40.4 s
Wall time: 41.6 s
```

```
[158]: GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=0),
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             refit='roc_auc',
             scoring={'accuracy': 'accuracy', 'f1': 'f1',
                      'precision': 'precision', 'recall': 'recall',
```

```
'roc_auc': 'roc_auc'})
```

[159]:
```
#checking best params
tree2.best_params_
```

[159]:
```
{'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

[160]:
```
# Check best AUC score on CV
tree2.best_score_
```

[160]:
```
0.9691362298270515
```

This model performs very well, even without satisfaction levels and detailed hours worked data.

Next, we will check the other scores.

[161]:
```
# 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   0.914552  0.916949  0.915707  0.971978  0.969819
               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.913587  0.864036  0.888083  0.963861  0.969136
```

Some of the other scores fell. That's to be expected given fewer features were taken into account in this round of the model. Still, the scores are very good.

Random forest - Round 2

[162]:
```
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': 'accuracy', 'precision':'precision', 'recall':'recall',␣
↪'f1':'f1', 'roc_auc':'roc_auc'}

# Instantiate GridSearch
rf2 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

[163]:
```
%%time
rf2.fit(X_train, y_train)
```

```
CPU times: user 12min 36s, sys: 621 ms, total: 12min 36s
Wall time: 12min 38s
```

[163]:
```
GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=0),
             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]},
             refit='roc_auc',
             scoring={'accuracy': 'accuracy', 'f1': 'f1',
                      'precision': 'precision', 'recall': 'recall',
                      'roc_auc': 'roc_auc'})
```

[164]:
```
# Write pickle
write_pickle(path, rf2, 'hr_rf2')
# Read in pickle
rf2 = read_pickle(path, 'hr_rf2')
```

[165]:
```
# Checking best params
rf2.best_params_
```

[165]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 300}
```

[166]:
```
# Checking best AUC score on CV
rf2.best_score_
```

[166]:
```
0.9648100662833985
```

[167]:
```
# 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   0.913587  0.864036  0.888083  0.963861  0.969136
               model  precision    recall        F1  accuracy      auc
0  random forest2 cv   0.866758  0.878754  0.872407  0.957411  0.96481
```

Again, the scores dropped slightly, but the random forest performs better than the decision tree if using AUC as the deciding metric.

Scoring the champion model on the test set now.

[168]:
```
# Get predictions on test data
rf2_test_scores = get_scores('random forest2 test', rf2, X_test, y_test)
rf2_test_scores
```

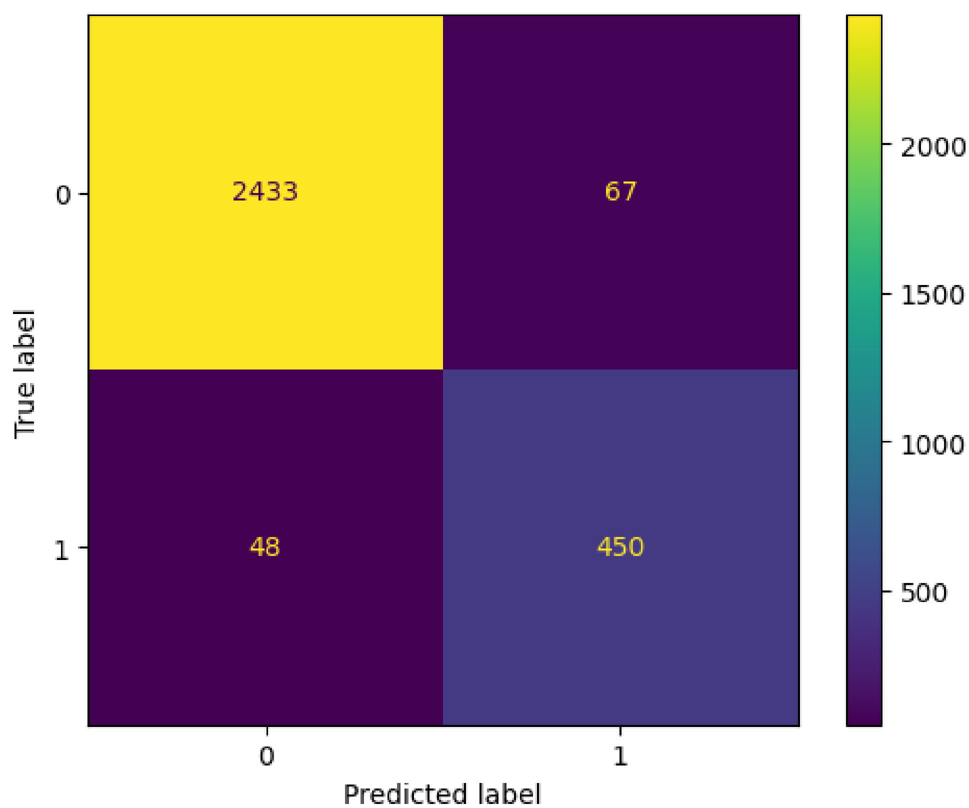
[168]:

	model	precision	recall	f1	accuracy	AUC
0	random forest2 test	0.870406	0.903614	0.8867	0.961641	0.938407

This seems to be a stable, well-performing final model.

Confusion matrix for visualizing how well it predicts on the test set.

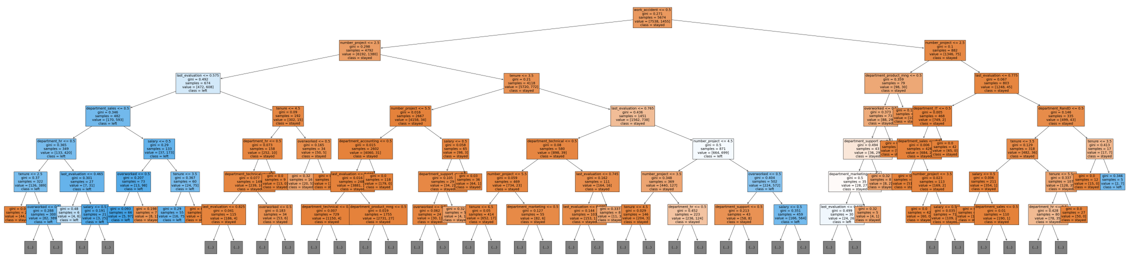
[169]:
```
# Generating array of values for confusion matrix
preds = rf2.best_estimator_.predict(X_test)
cm = confusion_matrix(y_test, preds, labels=rf2.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=rf2.classes_)
disp.plot(values_format='');
```



The model predicts more false positives than false negatives, which means that some employees may be identified as at risk of quitting or getting fired, when that's actually not the case. But this is still a strong model.

Decision tree splits

[175]:
```
# Plot the tree
plt.figure(figsize=(85,20))
plot_tree(tree2.best_estimator_.estimators_[1], max_depth=6, fontsize=14,␣
↪feature_names=X.columns,
          class_names={0:'stayed', 1:'left'}, filled=True);
plt.show()
```



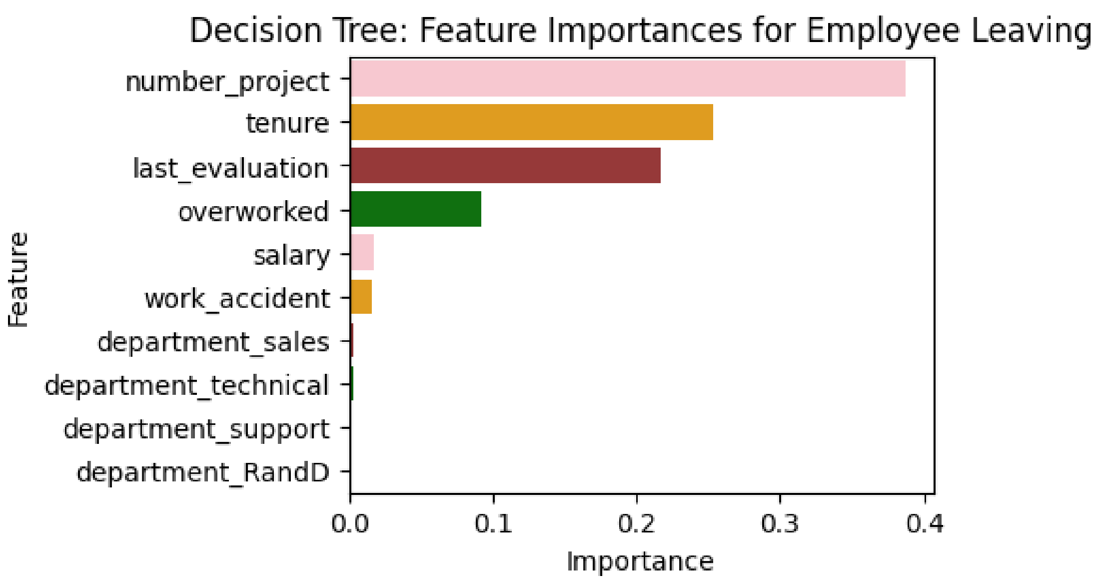
Decision tree feature importance

[185]:
```
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].head(10)
tree2_importances
```

[185]:

	gini_importance
number_project	0.387279
tenure	0.253402
last_evaluation	0.217147
overworked	0.092019
salary	0.016931
work_accident	0.015490
department_sales	0.002770
department_technical	0.002621
department_support	0.002204
department_RandD	0.001730

[191]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
plt.figure(figsize=(4,3))
sns.barplot(data=tree2_importances, x="gini_importance", y=tree2_importances.
 ↪index, orient='h', hue=tree2_importances.index,palette=["pink", "orange",␣
 ↪"brown", "green"])
plt.title("Decision Tree: Feature Importances for Employee Leaving",␣
 ↪fontsize=12)
plt.ylabel("Feature")
plt.xlabel("Importance")

plt.show()
```

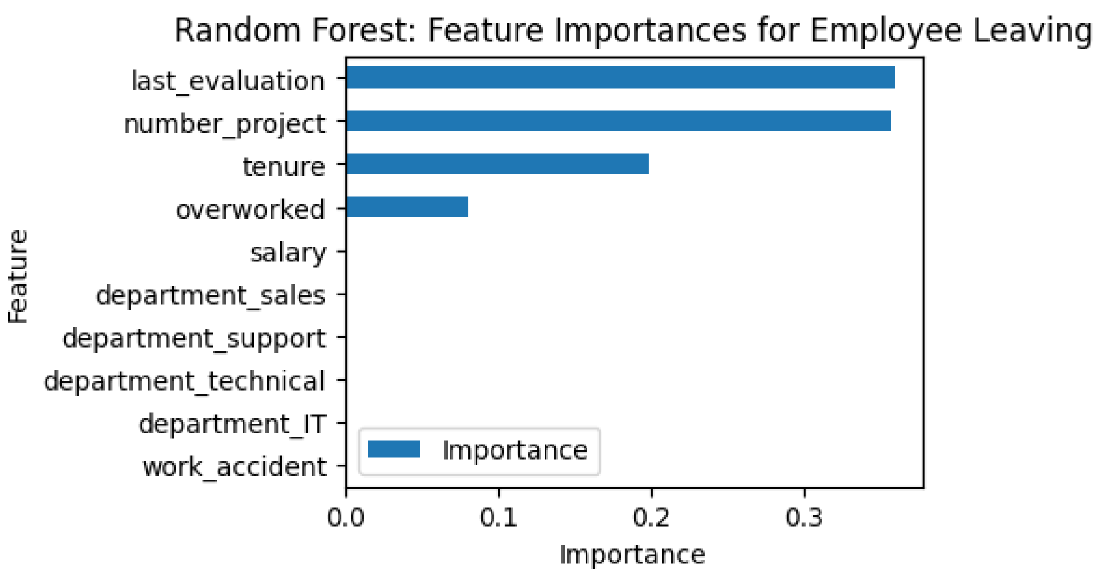


The barplot above shows that in this decision tree model, number_project, tenure, last_evaluation, and overworked have the highest importance, in that order. These variables are most helpful in predicting the outcome variable, left.

Random forest feature importance

[196]:
```
# 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", figsize=(4,3))

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.

4 pacE: Execute Stage

4.0.1 Summary of model results

Logistic Regression

The logistic regression model achieved precision of 80%, recall of 83%, f1-score of 80% (all weighted averages), and accuracy of 83%, on the test set.

Tree-based Machine Learning

After conducting feature engineering, the decision tree model achieved AUC of 93.8%, precision of 87.0%, recall of 90.4%, f1-score of 88.7%, and accuracy of 96.2%, on the test set. The random forest modestly outperformed the decision tree model.

4.0.2 Conclusion, Recommendations, Next Steps

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:

- Cap the number of projects that employees can work on.
- Consider promoting employees who have been with the company for atleast four years, or conduct further investigation about why four-year tenured employees are so dissatisfied.
- Either reward employees for working longer hours, or don't require them to do so.
- If employees aren't familiar with the company's overtime pay policies, inform them about this. If the expectations around workload and time off aren't explicit, make them clear.
- Hold company-wide and within-team discussions to understand and address the company work culture, across the board and in specific contexts.
- High evaluation scores should not be reserved for employees who work 200+ hours per month. Consider a proportionate scale for rewarding employees who contribute more/put in more effort.