Activity_ Course 7 Salifort Motors project lab

July 27, 2024

1 Capstone project: Providing data-driven suggestions for HR

1.1 Description and deliverables

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

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

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

2 PACE stages

2.1 Pace: Plan

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

In this stage, consider the following:

2.1.1 Understand the business scenario and problem

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

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

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

2.1.2 Familiarize yourself with the HR dataset

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

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

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

Reflect on these questions as you complete the plan stage.

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

[Double-click to enter your responses here.]

2.2 Step 1. Imports

- Import packages

- Load dataset

2.2.1 Import packages

[3]:
```
# Import packages
### YOUR CODE HERE ###

#data manipulation
import pandas as pd
import numpy as np

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

#data modelling
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier
from xgboost import XGBRegressor
from xgboost import plot_importance

#metrics and other helpful functions
from sklearn.model_selection import train_test_split, GridSearchCV
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

#to save the model
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.

[4]:
```
# 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()
```

[4]:
```
   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

[5]:
```
# 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

[7]:
```python
# Gather descriptive statistics about the data
### YOUR CODE HERE ###
df0.describe()
```

[7]:
```
       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.

[8]:
```
# Display all column names
### YOUR CODE HERE ###
df0.columns
```

[8]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
       'promotion_last_5years', 'Department', 'salary'],
      dtype='object')
```

[5]:
```
# Rename columns as needed
### YOUR CODE HERE ###
df0= df0.rename(columns={'average_montly_hours': 'average_monthly_hours',
                        'Work_accident':'work_accident',
                        'Department':'department',
                        'time_spend_company':'tenure'})

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

[5]:
```
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.

[10]:
```
# Check for missing values
### YOUR CODE HERE ###
df0.isna().sum()
```

[10]:
```
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.

[11]:
```
# Check for duplicates
### YOUR CODE HERE ###
df0.duplicated().sum()
```

[11]:
```
3008
```

[12]:
```
# Inspect some rows containing duplicates as needed
### YOUR CODE HERE ###
df0[df0.duplicated()].head()
```

[12]:
```
      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
```

[6]:
```
# Drop duplicates and save resulting dataframe in a new variable as needed
### YOUR CODE HERE ###
df1= df0.drop_duplicates(keep='first')
```

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

[6]:

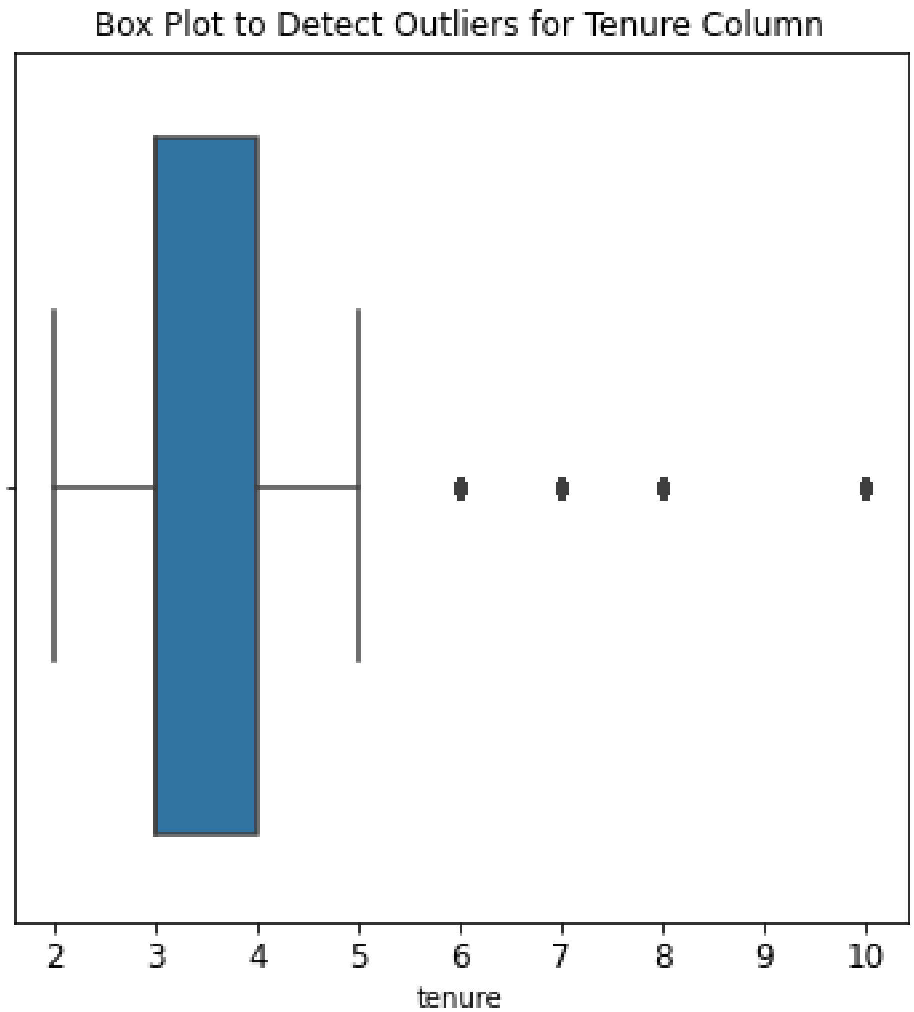
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
2                0.11             0.88               7     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

[15]:

```python
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
### YOUR CODE HERE ###
plt.figure(figsize=(6,6))
plt.title('Box Plot to Detect Outliers for Tenure Column', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(data=df1, x='tenure')
plt.show()
```



[7]:
```
# 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('Lower limit:', lower_limit)
print('Upper limit:', upper_limit)

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

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

```
Lower limit: 1.5
Upper limit: 5.5
Numer of rows in the data containing ouliers 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.

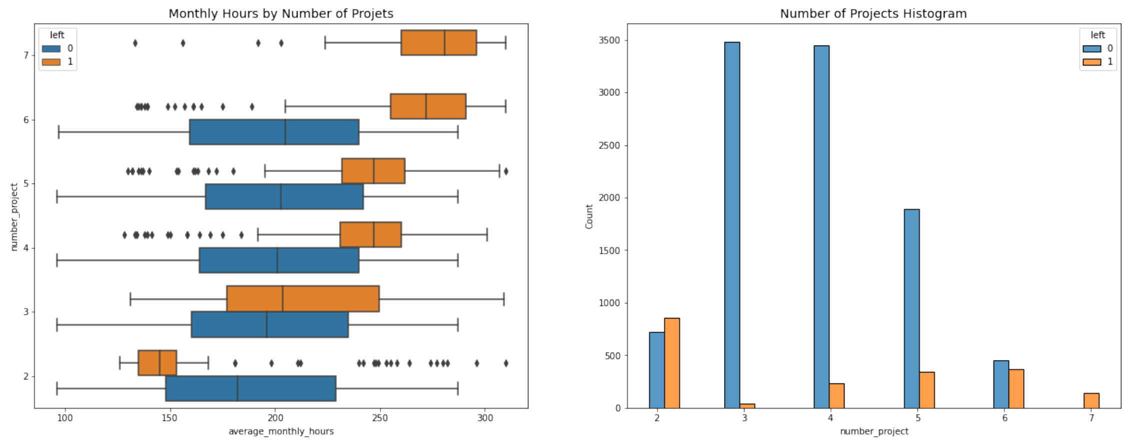
[8]:
```
# Get numbers of people who left vs. stayed
### YOUR CODE HERE ###
print(df1['left'].value_counts())
print()
# Get percentages of people who left vs. stayed
### YOUR CODE HERE ###
print(df1['left'].value_counts(normalize=True))
```

```
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.

[19]:
```
# Create a plot as needed
### YOUR CODE HERE ###
fig, ax = plt.subplots(1,2, figsize= (22,8))

#boxplot showing 'average_monthly_hours' distributions for 'number_project'␣
↪comparing employees who stayed versus those that 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 Projets', fontsize='14')

#boxplot showing 'average_monthly_hours' distributions for 'number_project'␣
↪comparing employees who stayed versus those that left
tenure_stay = df1[df1['left']==0]['number_project']
tenure_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')

plt.show()
```

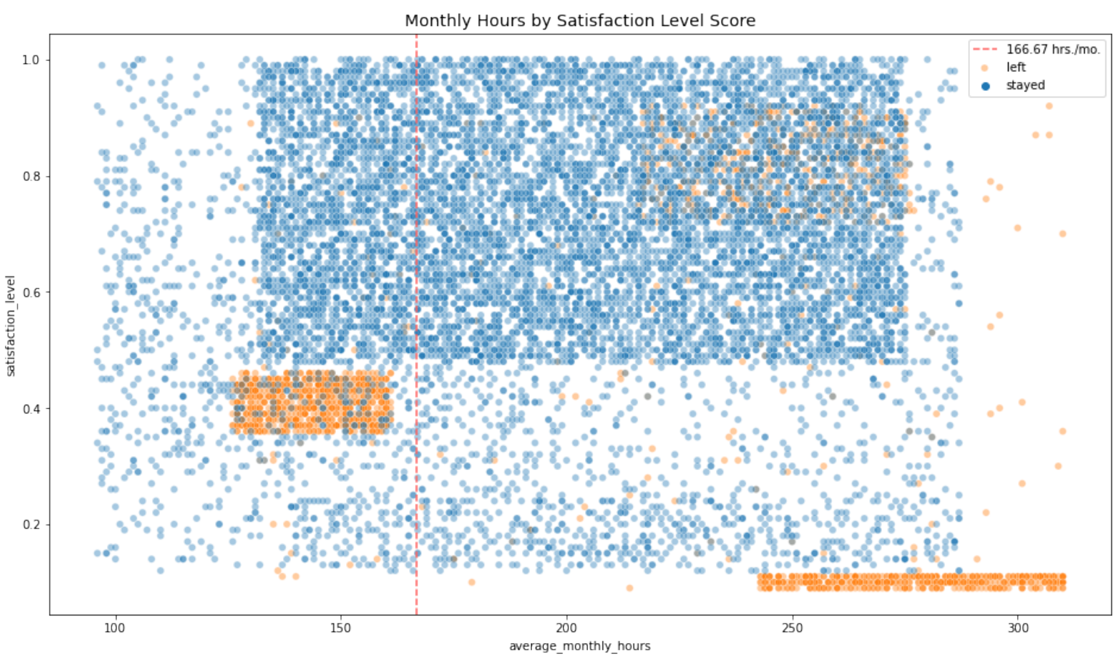


[20]:
```
# the value count for those that stayed/left for employees with 7 projects
df1[df1['number_project']==7]['left'].value_counts()
```

[20]:
```
1    145
Name: left, dtype: int64
```

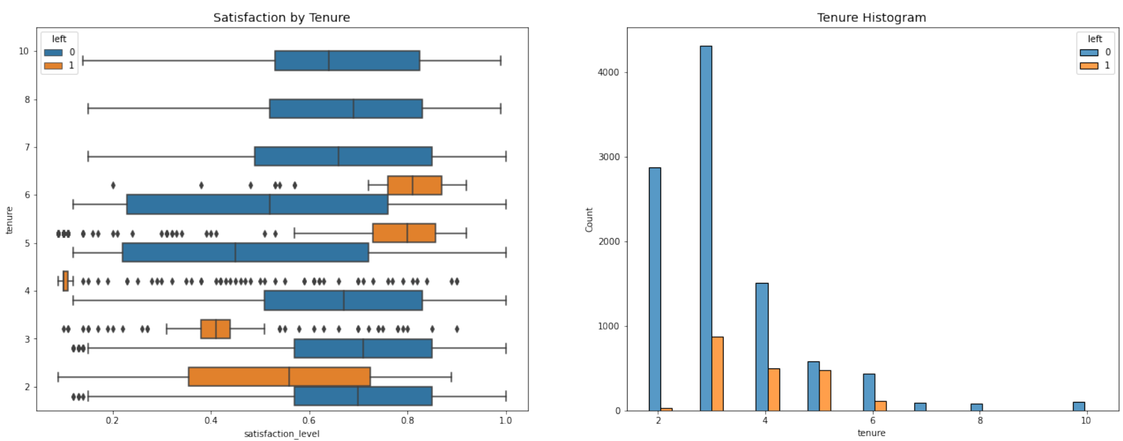
[24]:
```
# Create a plot as needed
### YOUR CODE HERE ###

#a scatterplot of the 'average_monthly_hours' versus 'satisfaction_level',␣
↪comparing employees who stayed versus those that left
#using a 40 hour work week and 2 weeks for vacation in one year, the average␣
↪number of working hours for a full time employee would be 166.67
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 Satisfaction Level Score', fontsize='14')
```

[24]:
```
Text(0.5, 1.0, 'Monthly Hours by Satisfaction Level Score')
```



[26]:
```
# Create a plot as needed
### YOUR CODE HERE ###
fig, ax = plt.subplots(1,2, figsize=(22,8))

#boxplot showing distributions of 'satisfaction_levels' by 'tenure', comparing␣
↪employees who stayed versus those that 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')

#boxplot showing distributions of 'tenure', comparing employees who stayed␣
↪versus those that 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()
```



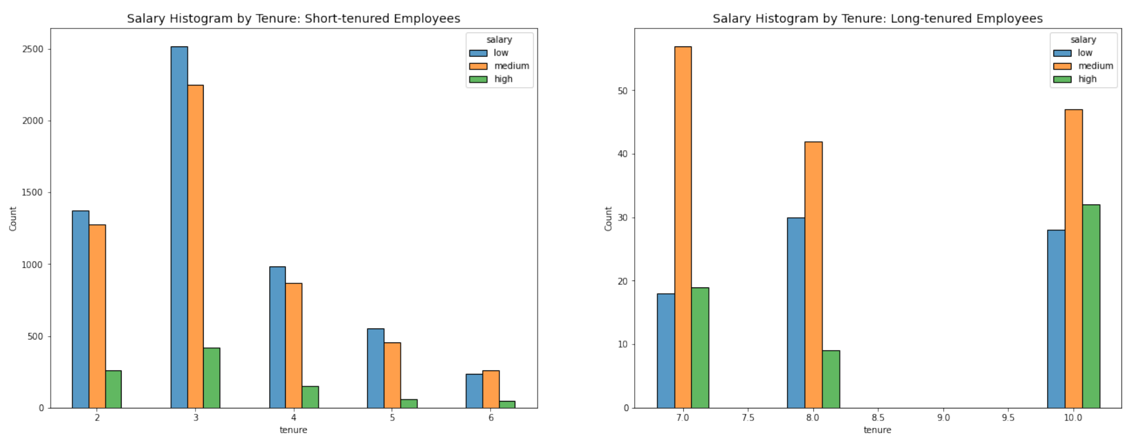
[27]:
```
#the mean and median satisfaction scores of employees who left and those who␣
↪didn't.
df1.groupby(['left'])['satisfaction_level'].agg([np.mean,np.median])
```

[27]:

left	mean	median
0	0.667365	0.69
1	0.440271	0.41

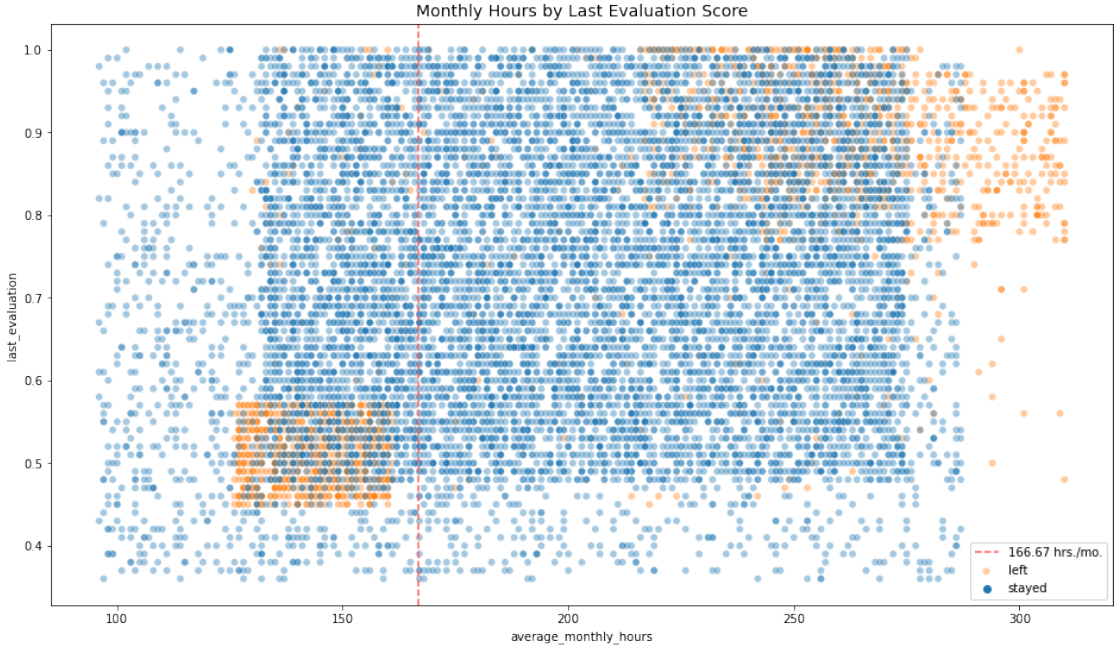
[31]:
```
# Create a plot as needed
### YOUR CODE HERE ###
#examining the salary levels for different tenures
fig, ax= plt.subplots(1,2, figsize= (22,8))

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

#long-term employees
tenure_long = df1[df1['tenure'] > 6]
```

```python
#plot the short-term tenure histogram
sns.histplot(data=tenure_short, x='tenure', hue='salary', discrete=1,
             hue_order=['low', 'medium', 'high'], multiple='dodge', shrink=.5, ax=ax[0])
ax[0].set_title('Salary Histogram by Tenure: Short-tenured Employees', fontsize='14')

#plot the long-term tenure 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 Employees', fontsize='14')

plt.show()
```



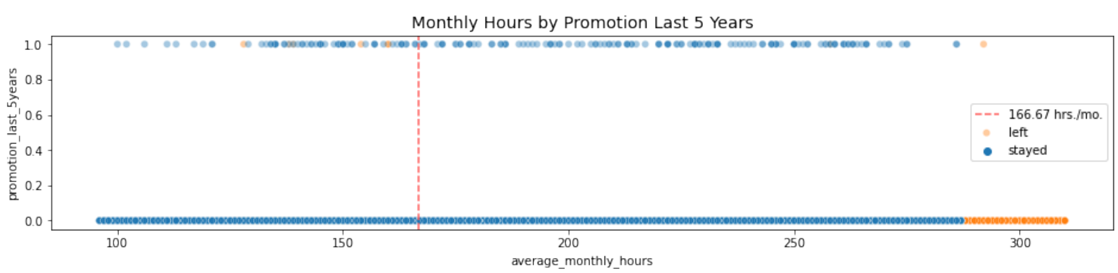
[32]:
```python
# Create a plot as needed
### YOUR CODE HERE ###
#a scatterplot of the 'average_monthly_hours' versus 'last_evaluation', comparing employees who stayed versus those that left
#using a 40 hour work week and 2 weeks for vacation in one year, the average number of working hours for a full time employee would be 166.67
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')
```

[32]:
```
Text(0.5, 1.0, 'Monthly Hours by Last Evaluation Score')
```



[33]:
```
# Create a plot as needed
### YOUR CODE HERE ###

#a scatterplot examining the 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 5 Years', fontsize='14')
```

[33]: Text(0.5, 1.0, 'Monthly Hours by Promotion Last 5 Years')

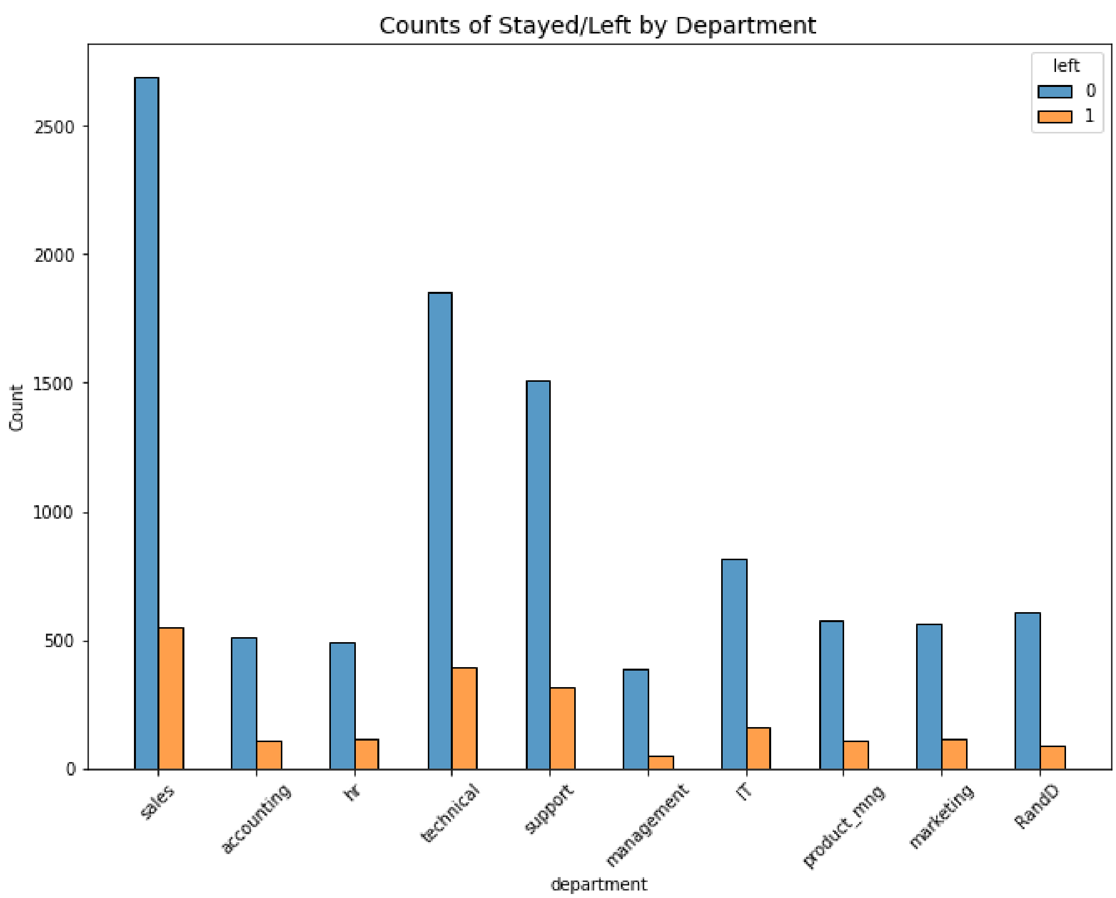


[36]:
```
df1['department'].value_counts()
```

[36]:
```
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
```

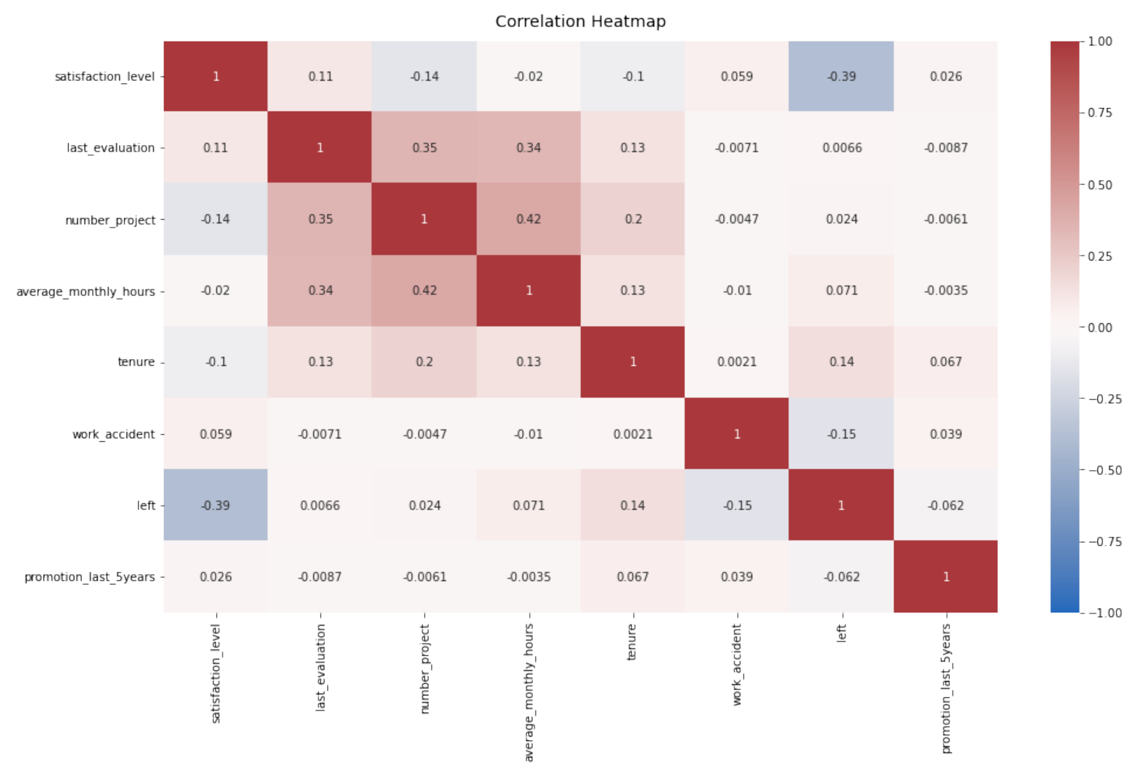
[37]:
```
#a histogram to compare department distribution of employees who left to those
↪that didn't
plt.figure(figsize=(11,8))
sns.histplot(data=df1, x='department', hue='left', discrete=1,
             hue_order=[0,1], multiple='dodge', shrink=.5)
plt.xticks(rotation='45')
plt.title('Counts of Stayed/Left by Department', fontsize='14')
```

[37]:
```
Text(0.5, 1.0, 'Counts of Stayed/Left by Department')
```



[38]:
```
#checking for strong correlations between variables in the data

#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)
```

[38]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```



3.1.2 Insights

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

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.

4 paCe: Construct Stage

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

Recall model assumptions

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

Reflect on these questions as you complete the constructing stage.

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

[Double-click to enter your responses here.]

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

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

4.1.1 Identify the type of prediction task.

The goal is to predict whether an employee leaves the company, which is a categorical outcome variable. So this task involves classification. More specifically, this involves binary classification, since the outcome variable left can be either 1 (indicating employee left) or 0 (indicating employee didn't leave).

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

A Logistic Regresssion Model or a Tree Based Machine Learning Model. This section of the project will utilize Logistic Regression.

Binomial logistic regression suits the task because it involves binary classification.

Before splitting the data, the non-numeric variables will be encoded. There are two: department and salary.

department is a categorical variable, which means it can be dummied 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.

4.1.3 Modeling

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

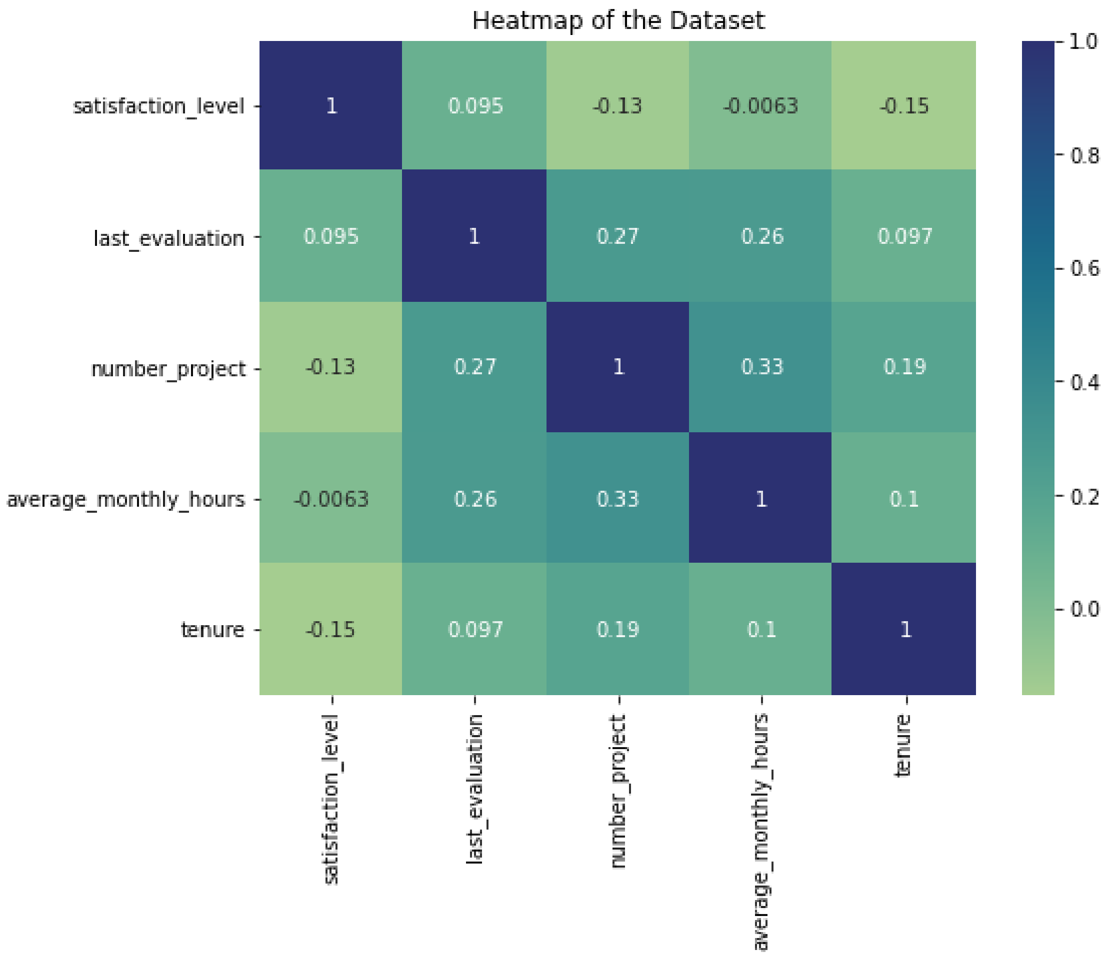
[9]:
```python
### YOUR CODE HERE ###
#a copy of the dataframe
df_enc = df1.copy()

#encode 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()
```

[9]:
```
   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              0
1       6              0     1                      0       1              0
2       4              0     1                      0       1              0
3       5              0     1                      0       0              0
4       3              0     1                      0       0              0

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

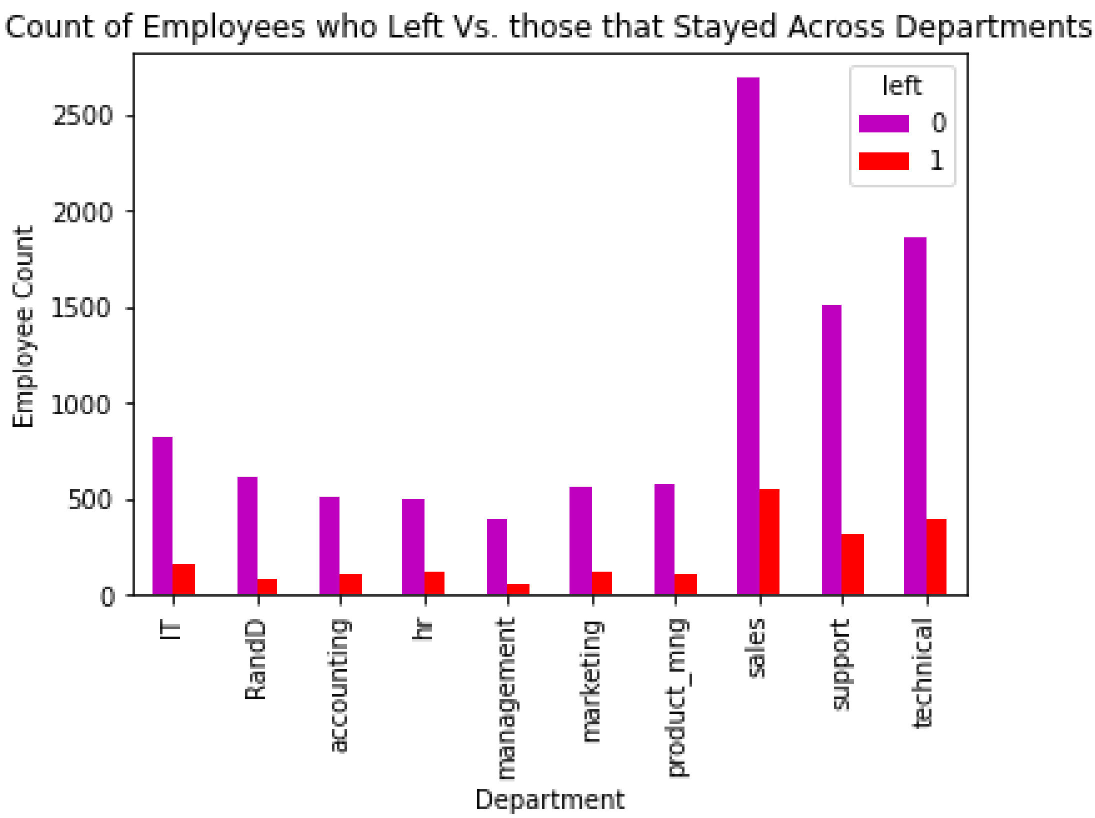
```
   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
1                      0                     0                       0
2                      0                     0                       0
3        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

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

[42]:

```
#a heatmap to visualize how correlated the 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 Dataset')
plt.show()
```



[43]:
```
# a stacked bart plot to visualize number of employees across department,␣
↪comparing those who left with those who didn't
pd.crosstab(df1['department'], df1['left']).plot(kind='bar', color='mr')
plt.title('Count of Employees who Left Vs. those that Stayed Across␣
↪Departments')
plt.ylabel('Employee Count')
plt.xlabel('Department')
plt.show()
```



[10]:

```python
# since logistic regression is sensitive to outliers, outliers have to be␣
↪removed from the tenure column
df_logreg = df_enc[(df_enc['tenure'] >= lower_limit) & (df_enc['tenure'] <=␣
↪upper_limit)]
df_logreg.head()
```

[10]:

	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	department_IT	\
0	3	0	1	0	0	0	
2	4	0	1	0	1	0	
3	5	0	1	0	0	0	
4	3	0	1	0	0	0	
5	3	0	1	0	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
```

[11]:
```
#isolate the outcome variable
y = df_logreg['left']
y.head()
```

[11]:
```
0    1
2    1
3    1
4    1
5    1
Name: left, dtype: int64
```

[12]:
```
#drop the outcome variable from X
X= df_logreg.drop('left', axis=1)
X.head()
```

[12]:
```
   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_IT  \
0       3              0                      0       0              0
2       4              0                      0       1              0
3       5              0                      0       0              0
4       3              0                      0       0              0
```

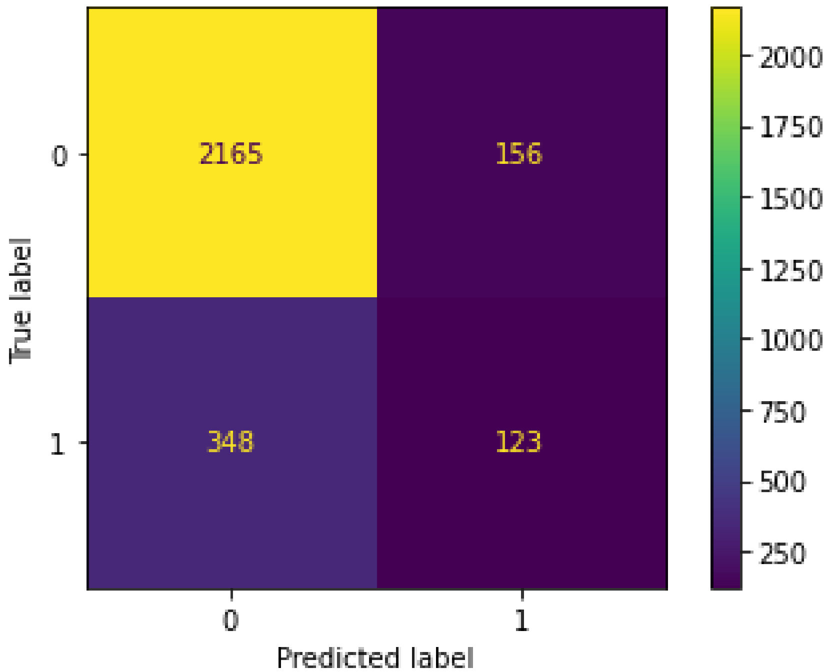
```
5       3              0                       0       0              0

   department_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
```

[14]:
```
#split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25,
 ↪stratify=y, random_state=42)
```

[15]:
```
#construct a logistic model and fit it ot the dataset
log_clf = LogisticRegression(random_state=42, max_iter=500).fit(X_train,y_train)
```

[16]:
```
# using the logistic regression model to get predictions on the test set
y_pred = log_clf.predict(X_test)
```

[19]:
```
# a confusion matrix to visualize the result of the model
cm = confusion_matrix(y_test, y_pred, labels=log_clf.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix= cm, display_labels=log_clf.
 ↪classes_)
disp.plot(values_format='')

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

[20]:
```
# checking the class balance in the left column
df_logreg['left'].value_counts(normalize=True)
```

[20]:
```
0    0.831468
1    0.168532
Name: left, dtype: float64
```

[21]:
```
# a classification report for the 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
```

4.2 Modeling Approach B: Tree-based Model

This approach will use the Decision Tree and Random Forest to build a model for Salifort motors that can predict whther or not an employee will leave the company.

[22]:
```
# Isolate the outcome variable
y = df_enc['left']
y.head()
```

[22]:
```
0    1
1    1
2    1
3    1
4    1
Name: left, dtype: int64
```

[24]:
```
X= df_enc.drop('left', axis=1)
X.head()
```

[24]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
4                0.37             0.52               2                    159

   tenure  work_accident  promotion_last_5years  salary  department_IT  \
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_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
```

[25]:
```
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.25,␣
↪stratify=y, random_state=0)
```

Decision Tree

[27]:
```
#Instantiate the model
tree = DecisionTreeClassifier(random_state=0)

#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]}

#A dictionary of scoring metrics to caprure
scoring ={'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

#Instantiate Gridsearch
tree1= GridSearchCV(tree, cv_params, scoring=scoring, refit='roc_auc')
```

[28]:
```
#fit the decision tree model to the training data
tree1.fit(X_train, y_train)
```

[28]:
```
GridSearchCV(cv=None, 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={'precision', 'accuracy', 'roc_auc', 'f1', 'recall'},
             verbose=0)
```

[29]:
```
#check best parameters
tree1.best_params_
```

[29]:
```
{'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 2}
```

[30]:
```
# check best AUC score
tree1.best_score_
```

[30]:
```
0.9719548921460799
```

The AUC score is strong meaning the model can predict employess who will leave well

[35]:
```
# his function will extract all the scores from the grid search
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 the results from the cv and put them in a dataframe
    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
```

[36]:
```
# Get all CV scores
tree1_cv_results = make_results('decision tree cv', tree1, 'auc')
tree1_cv_results
```

[36]:

	model	precision	recall	F1	accuracy	auc
0	decision tree cv	0.972227	0.914269	0.942307	0.98143	0.971955

Random Forest

[37]:
```
rf = RandomForestClassifier(random_state=0)

# 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],
             }

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

#Instatiate Gridsearch
```

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

[38]:
```
#fit the model to the training data
rf1.fit(X_train, y_train)
```

[38]:
```
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={'precision', 'accuracy', 'roc_auc', 'f1', 'recall'},
             verbose=0)
```

[39]:
```
#identify the best AUC score achieved by the random forest model
rf1.best_score_
```

[39]:
```
0.9804250949807172
```

[40]:
```
# check best params
rf1.best_params_
```

[40]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 500}
```

[41]:
```
#Get all CV scores for the decision tree and random forest models
rf1_cv_results = make_results('random forest cv', rf1, 'auc')
print(tree1_cv_results)
```

```python
print(rf1_cv_results)
```

```
              model  precision    recall        F1  accuracy       auc
0  decision tree cv   0.972227  0.914269  0.942307   0.98143  0.971955
              model  precision    recall        F1  accuracy       auc
0  random forest cv   0.950023  0.915614  0.932467  0.977983  0.980425
```

The evaluation score of the Decision Tree Model outperforms that of the Random Forest Model with the exception of the recall score. The recall score for the random forest model is 0.001 higher but this is not a significant amount. The scores above show that the Decision Tree Model outperforms the Random Tree Model.

[42]:
```python
#Evaluate the model on the test set

#define 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
```

[43]:
```
#Get predictions on the test data
tree1_test_scores= get_scores('decision tree1 test', tree1, X_test, y_test)
tree1_test_scores
```

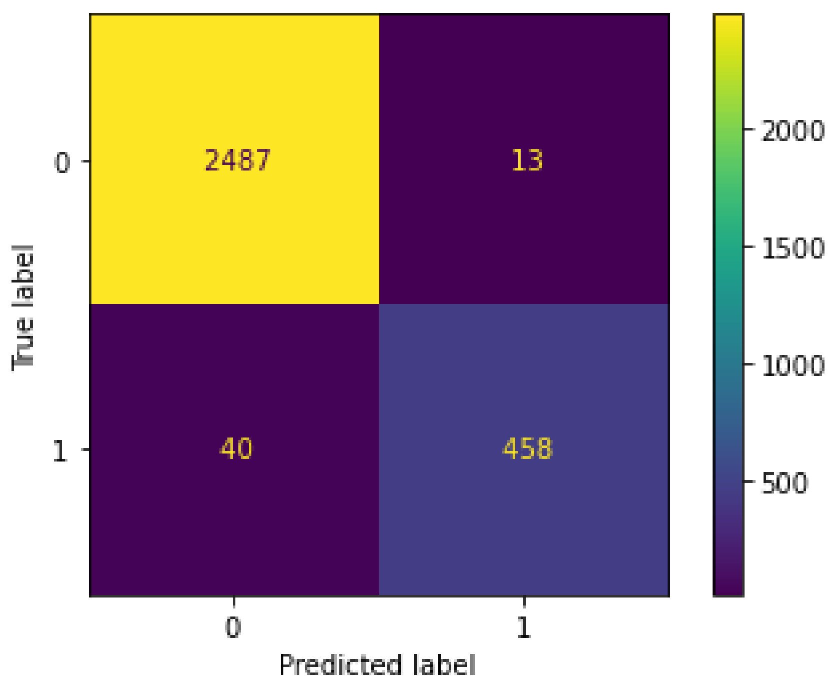
[43]:

	model	precision	recall	f1	accuracy	AUC
0	decision tree1 test	0.972399	0.919679	0.945304	0.982322	0.957239

The test scores are similar to the validation scores. This suggests that the model is strong and will perform well on unseen data.

[44]:
```
# plot a confusion matrix to see how well the model predicts on the test set
preds = tree1.best_estimator_.predict(X_test)
cm= confusion_matrix(y_test, preds, labels= tree1.classes_)

disp= ConfusionMatrixDisplay(confusion_matrix= cm, display_labels= tree1.
 ↪classes_)

disp.plot(values_format=' ')
```

[44]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x716cf5fcdf10>
```



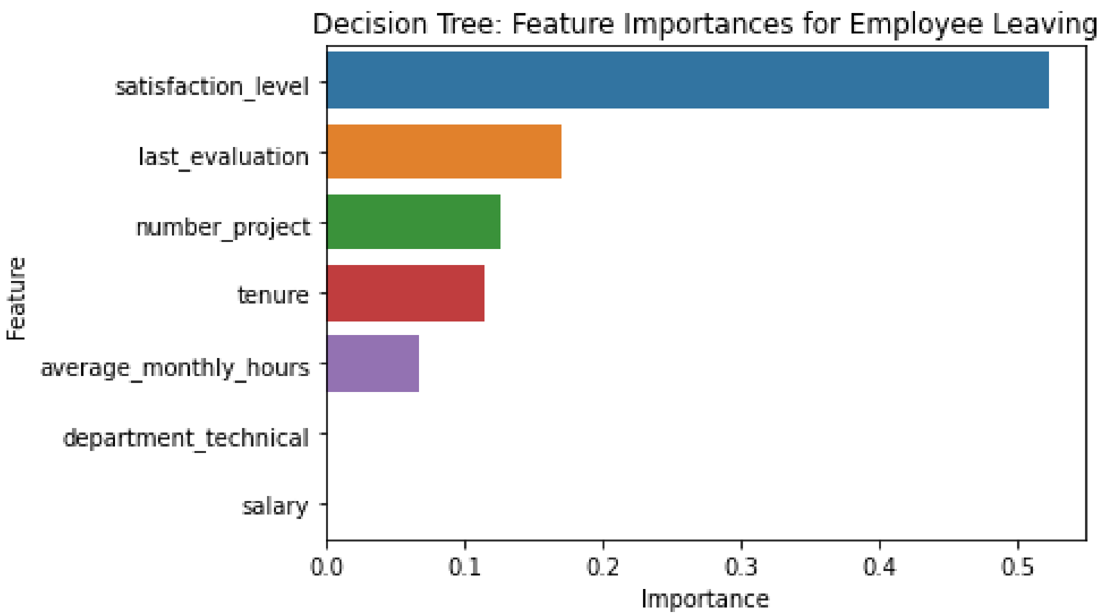
The model predicts more false negatives than false positives, but overall it is still a good model. It was able to correctly identify 2487 true negatives and 458 true positives.

[45]:
```
#Decision Tree Feature Importance

tree1_importances = pd.DataFrame(tree1.best_estimator_.feature_importances_,
                                 columns=['gini_importance'],
                                 index=X.columns
                                )
tree1_importances = tree1_importances.sort_values(by='gini_importance',␣
 ↪ascending=False)

# Only extract the features with importances > 0
tree1_importances = tree1_importances[tree1_importances['gini_importance'] != 0]
tree1_importances
```

[45]:

	gini_importance
satisfaction_level	0.522287
last_evaluation	0.169403
number_project	0.126216
tenure	0.114977
average_monthly_hours	0.066637
department_technical	0.000280
salary	0.000200

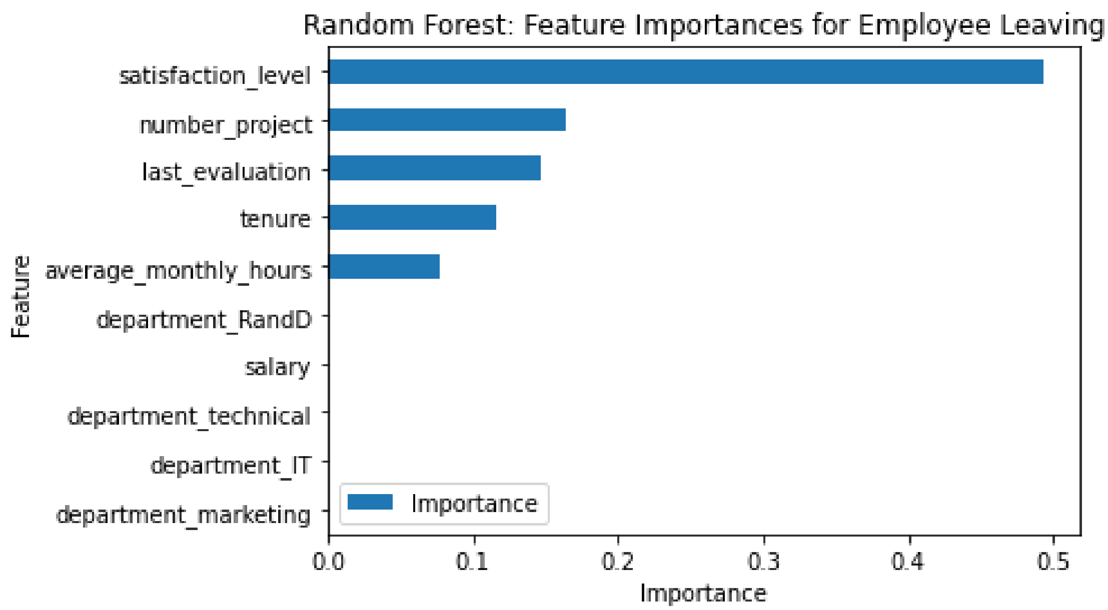
[46]:
```
# a barplot to visualize the decision tree feature importance
sns.barplot(data=tree1_importances, x="gini_importance", y=tree1_importances.
 ↪index, orient='h')
plt.title("Decision Tree: Feature Importances for Employee Leaving",␣
 ↪fontsize=12)
plt.ylabel("Feature")
plt.xlabel("Importance")
plt.show()
```



The barplot above shows that in this decision model, 'satisfation_level', 'last_evaluation', 'number_project', 'tenure' are the most helpful in predicting the outcome variable 'left'

[47]:
```
#Random Forest Feature Importance
# Get feature importances
feat_impt = rf1.best_estimator_.feature_importances_

# Get indices of top 10 features
ind = np.argpartition(rf1.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 the random forest model, 'satisfation_level', 'last_evaluation', 'number_project', 'tenure' are the most helpful in predicting the outcome variable 'left', just as in 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

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 The decision tree model achieved an AUC score of 95.7%, accuracy score of 98.3%, f1 score of 94.5%, a recall score of 91.9%, and a precision score of 97.2%. The decision tree model slightly outperformed the random forest model.

5.1.2 Conclusion, Recommendations, Next Steps

To retain employees, the following recommendations could considered:

- 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.

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.