Salifort Motors Employee Churn

November 13, 2024

[1]:
```
# 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
```

[2]:
```
df0 = pd.read_csv("HR_capstone_dataset.csv")

# Display 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
```

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

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

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

[4]:
```
       satisfaction_level  last_evaluation  number_project  \
count        14999.000000     14999.000000    14999.000000
```

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

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

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

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

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

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

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

[6]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_monthly_hours', 'tenure', 'work_accident', 'left',
```

```
       'promotion_last_5years', 'department', 'salary'],
      dtype='object')
```

[7]:
```
# Check for missing values
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
```

There are no missing values in the data.

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

[8]:
```
3008
```

3,008 rows contain duplicates. That is 20% of the data.

[9]:
```
# Inspect 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

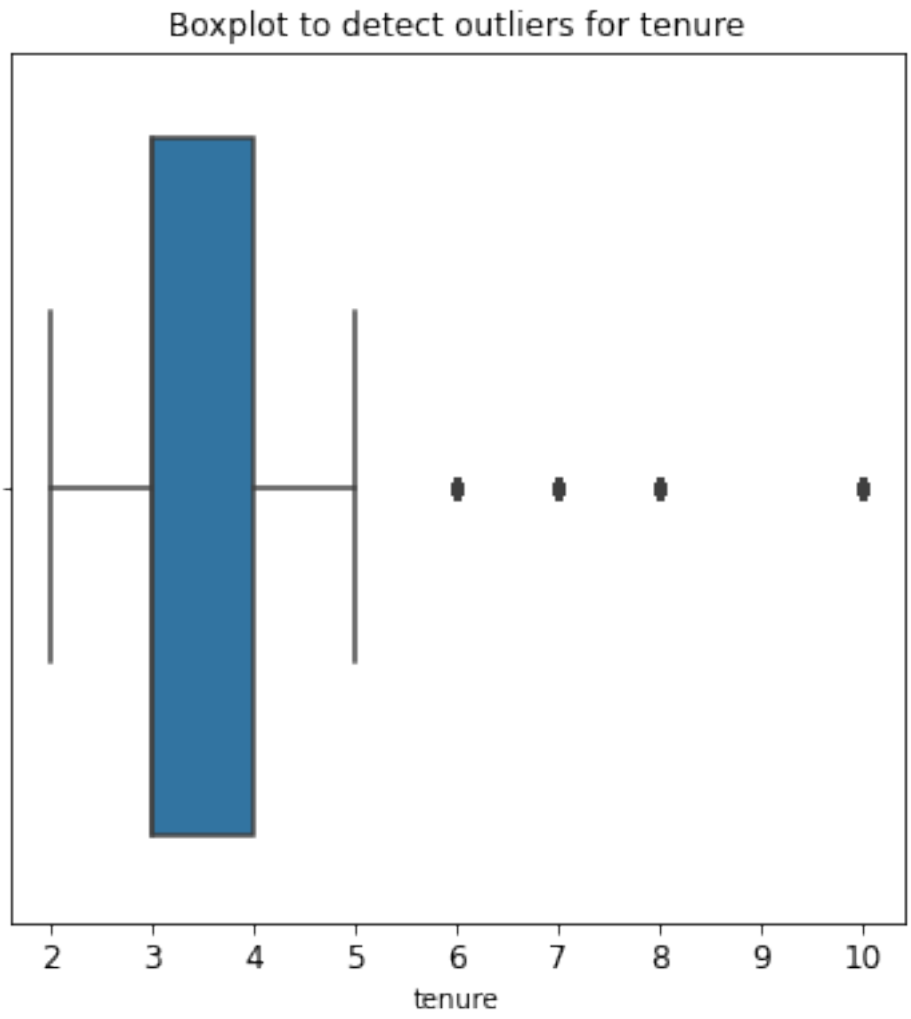
```
1368                     0       RandD     low
1461                     0       sales     low
```

[10]:
```
# Drop duplicates and save resulting dataframe in a new variable as needed

df1 = df0.drop_duplicates(keep='first')

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

[10]:

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

[11]:
```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
plt.figure(figsize=(6,6))
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]:
```
# Determine the number of rows containing outliers

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

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

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

# Define the upper limit and lower limit for non-outlier values in `tenure`
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr
print("Lower limit:", lower_limit)
```

```
print("Upper limit:", upper_limit)

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

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

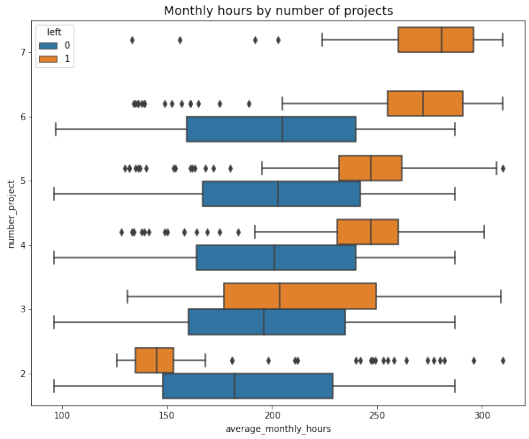
```
Lower limit: 1.5
Upper limit: 5.5
Number of rows in the data containing outliers in `tenure`: 824
```

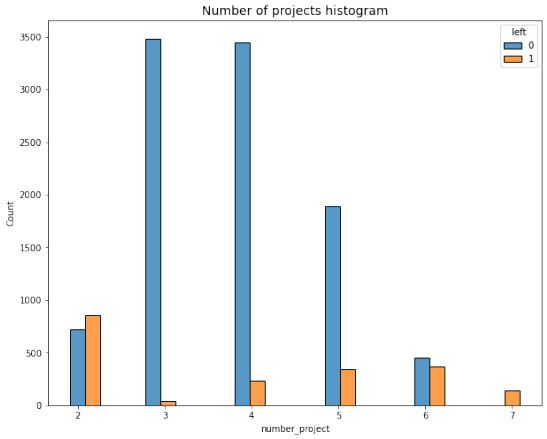
[13]:
```
# Get numbers of people who left vs. stayed

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
```

[14]:
```
# Set figure and axes
fig, ax = plt.subplots(1, 2, figsize = (22,8))

# Create boxplot showing `average_monthly_hours` distributions for␣
 ↪`number_project`, comparing employees who stayed versus those who left
sns.boxplot(data=df1, x='average_monthly_hours', y='number_project',␣
 ↪hue='left', orient="h", ax=ax[0])
ax[0].invert_yaxis()
ax[0].set_title('Monthly hours by number of projects', fontsize='14')

# Create histogram showing distribution of `number_project`, comparing␣
 ↪employees who stayed versus those who left
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')
```

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

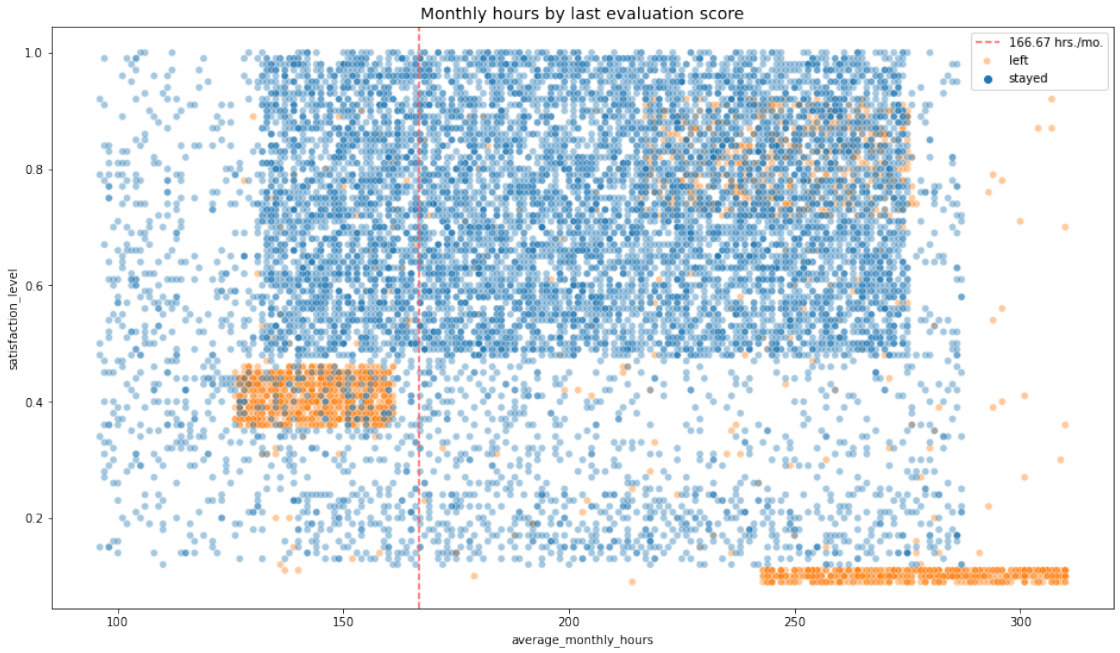




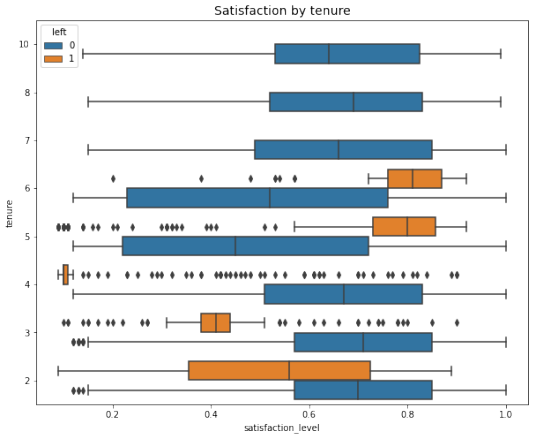
[15]:
```
# Get value counts of stayed/left for employees with 7 projects
df1[df1['number_project']==7]['left'].value_counts()
```

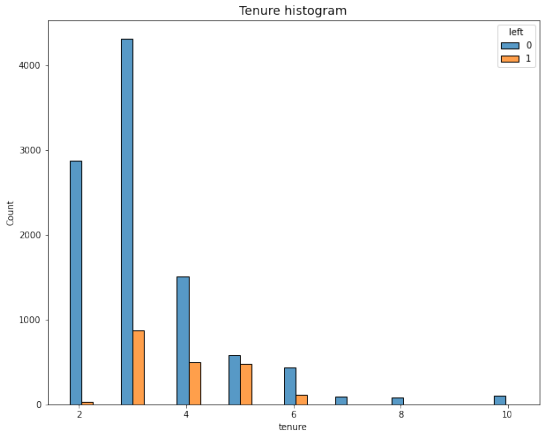
[15]:
```
1    145
Name: left, dtype: int64
```

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



[17]:
```
# 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();
```



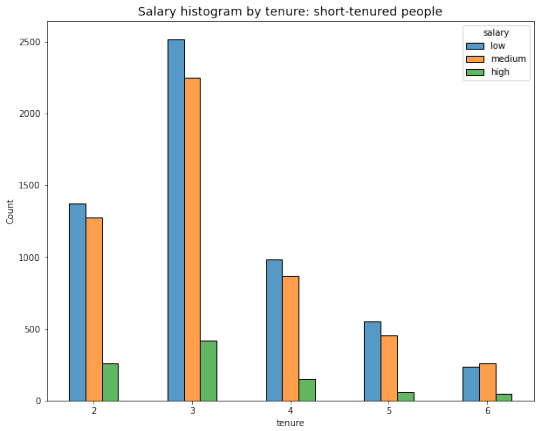


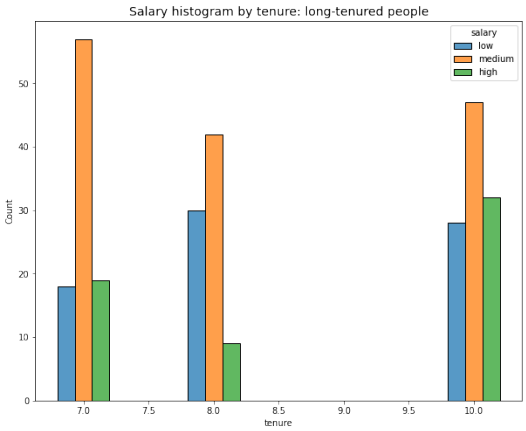
[18]:
```
# Calculate mean and median satisfaction scores of employees who left and those who stayed
df1.groupby(['left'])['satisfaction_level'].agg([np.mean,np.median])
```

[18]:

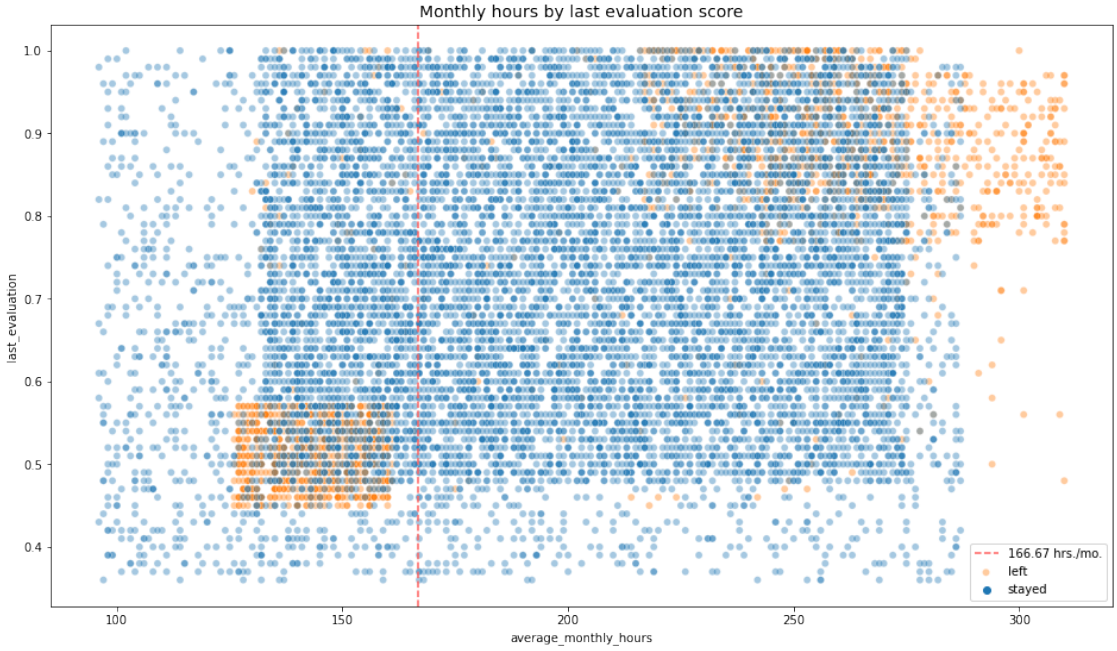
left	mean	median
0	0.667365	0.69
1	0.440271	0.41

[19]:
```
# 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');
```



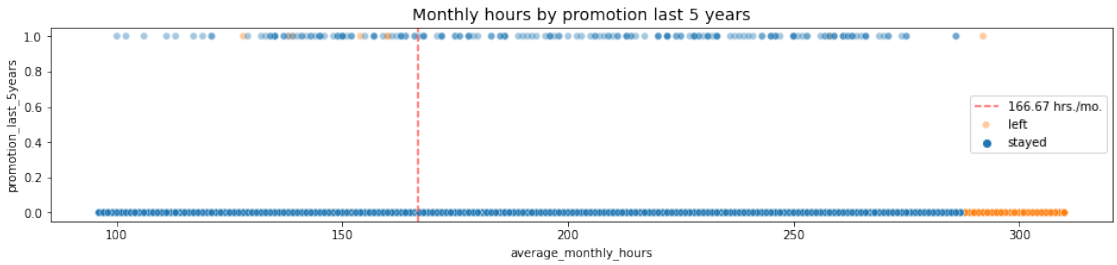


[20]:
```
# Create scatterplot of `average_monthly_hours` versus `last_evaluation`
plt.figure(figsize=(16, 9))
sns.scatterplot(data=df1, x='average_monthly_hours', y='last_evaluation',␣
↪hue='left', alpha=0.4)
plt.axvline(x=166.67, color='#ff6361', label='166.67 hrs./mo.', ls='--')
plt.legend(labels=['166.67 hrs./mo.', 'left', 'stayed'])
plt.title('Monthly hours by last evaluation score', fontsize='14');
```



[21]:
```
# Create plot to examine relationship between `average_monthly_hours` and␣
↪`promotion_last_5years`
```

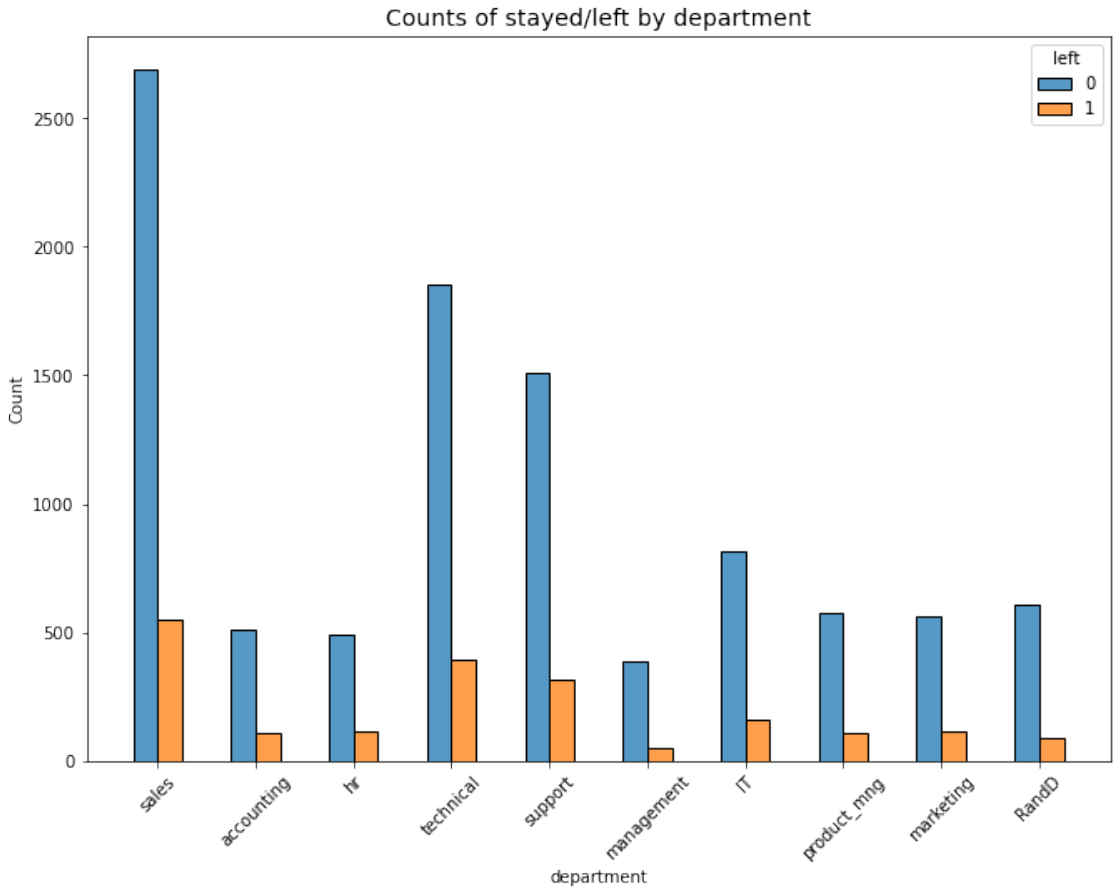
```
plt.figure(figsize=(16, 3))
sns.scatterplot(data=df1, x='average_monthly_hours', y='promotion_last_5years',␣
 ↪hue='left', alpha=0.4)
plt.axvline(x=166.67, color='#ff6361', ls='--')
plt.legend(labels=['166.67 hrs./mo.', 'left', 'stayed'])
plt.title('Monthly hours by promotion last 5 years', fontsize='14');
```



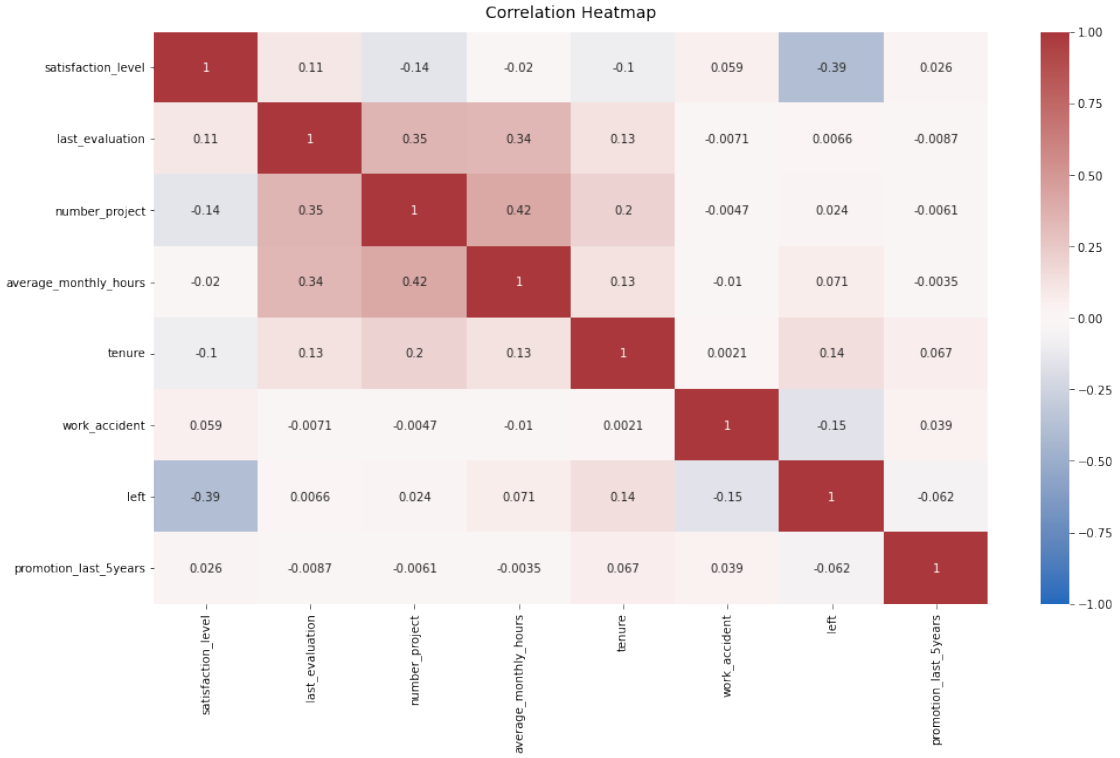
[22]:
```
# Display counts for each department
df1["department"].value_counts()
```

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

[23]:
```
# Create 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);
```



[24]:
```
# Plot a correlation heatmap
plt.figure(figsize=(16, 9))
heatmap = sns.heatmap(df0.corr(), vmin=-1, vmax=1, annot=True, cmap=sns.
 ↪color_palette("vlag", as_cmap=True))
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':14}, pad=12);
```



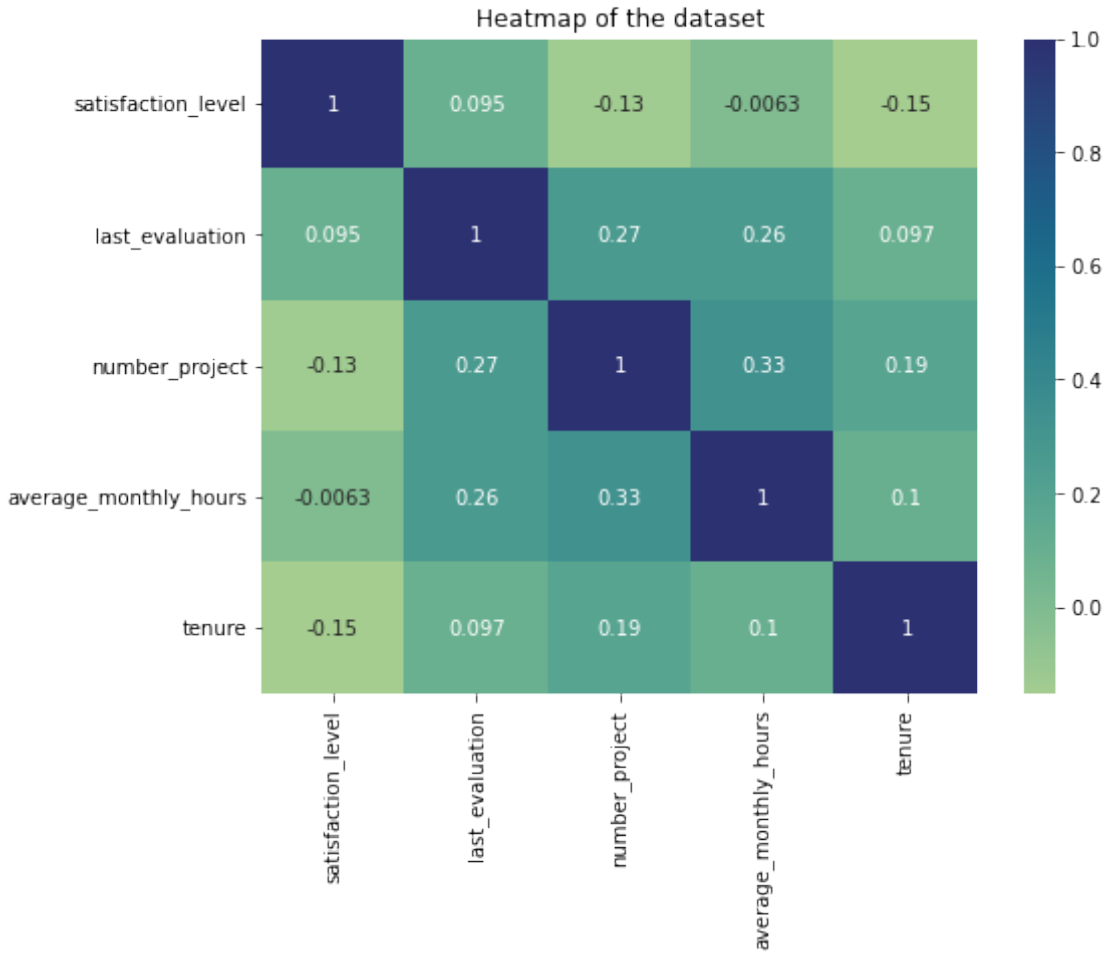
[25]:
```
# Copy the dataframe
df_enc = df1.copy()

# 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)

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

[25]:

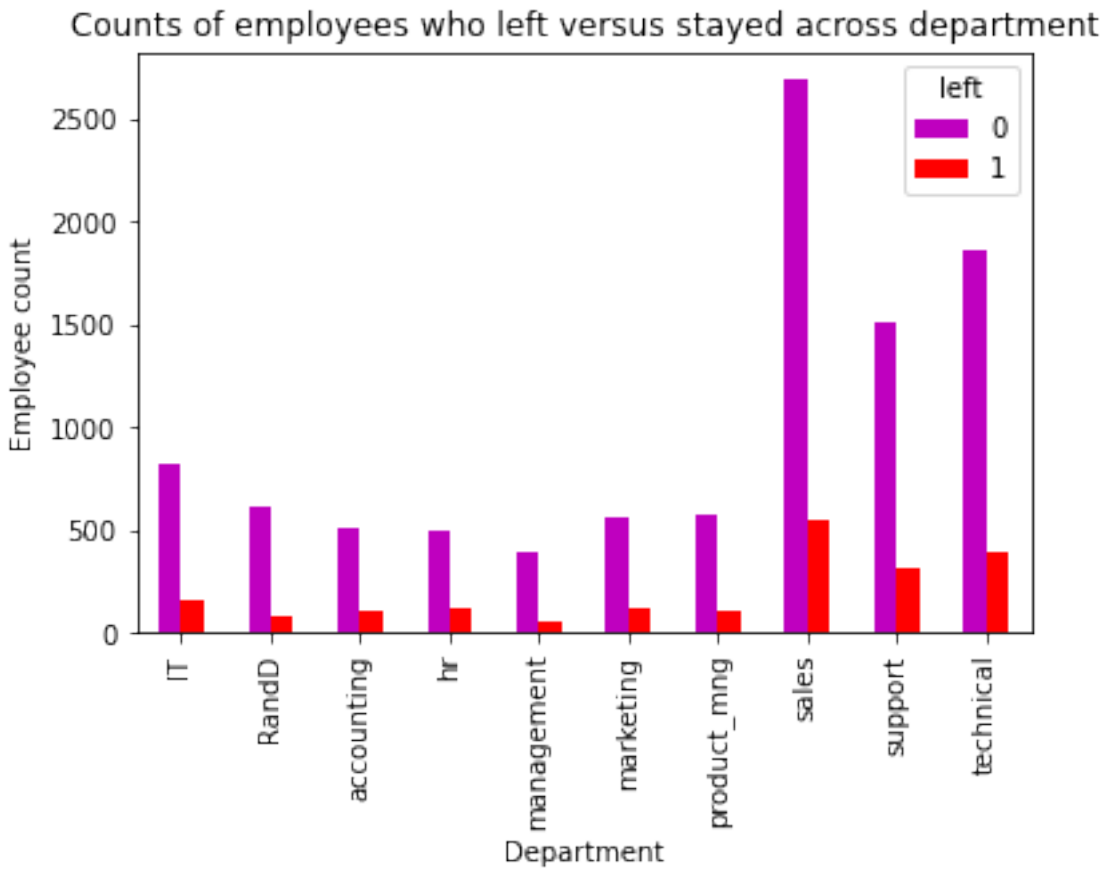
	satisfaction_level	last_evaluation	number_project	average_monthly_hours	\
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

```
   tenure  work_accident  left  promotion_last_5years  salary  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
```

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



[27]:
```
# Create a stacked bart plot to visualize number of employees across␣
 ↪department, comparing those who left with those who didn't
# In the legend, 0 (purple color) represents employees who did not leave, 1␣
 ↪(red color) represents employees who left
pd.crosstab(df1['department'], df1['left']).plot(kind ='bar',color='mr')
plt.title('Counts of employees who left versus stayed across department')
plt.ylabel('Employee count')
plt.xlabel('Department')
plt.show()
```



[28]:
```
# Select rows without outliers in `tenure` and save resulting dataframe in a␣
 ↪new variable
df_logreg = df_enc[(df_enc['tenure'] >= lower_limit) & (df_enc['tenure'] <=␣
 ↪upper_limit)]

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

[28]:

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

	tenure	work_accident	left	promotion_last_5years	salary	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
```

[29]:
```
# Isolate the outcome variable
y = df_logreg['left']

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

[29]:
```
0    1
2    1
3    1
4    1
5    1
Name: left, dtype: int64
```

[30]:
```
# Select the features you want to use in your model
X = df_logreg.drop('left', axis=1)

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

[30]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
```

```
4                0.37             0.52               2                    159
5                0.41             0.50               2                    153

   tenure  work_accident  promotion_last_5years  salary  department_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
```

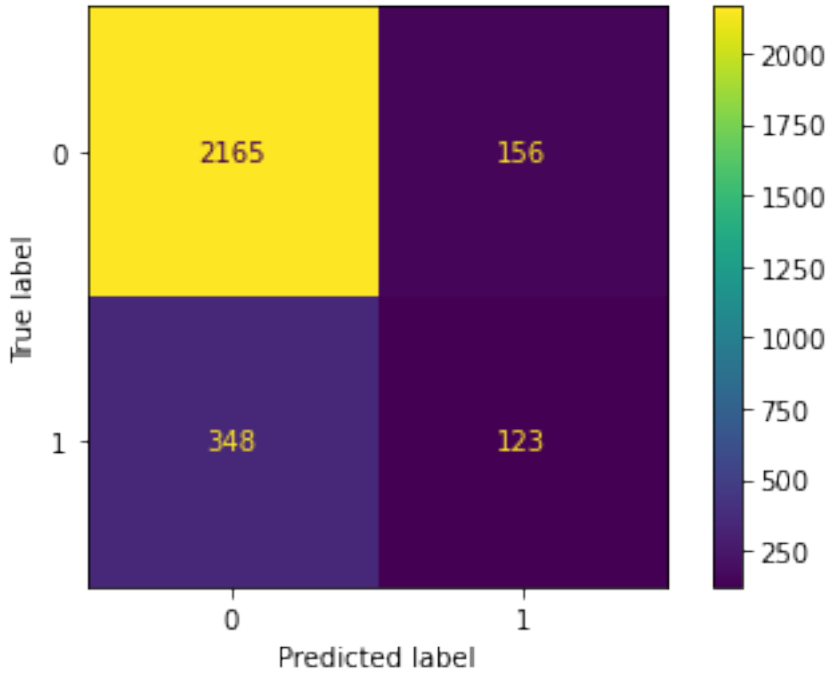
[31]:
```
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪stratify=y, random_state=42)
```

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

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

[34]:
```
# 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()
```



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

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

[36]:
```
# Create classification report for logistic regression model
target_names = ['Predicted would not leave', 'Predicted would leave']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                          precision    recall  f1-score   support

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

                 accuracy                           0.82      2792
                macro avg       0.65      0.60      0.61      2792
```

```
weighted avg       0.79      0.82      0.80      2792
```

[37]:
```
# Isolate the outcome variable
y = df_enc['left']

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

[37]:
```
0    1
1    1
2    1
3    1
4    1
Name: left, dtype: int64
```

[38]:
```
# Select the features
X = df_enc.drop('left', axis=1)

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

[38]:

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

[39]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪stratify=y, random_state=0)
```

[40]:
```
# Instantiate model
tree = DecisionTreeClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }

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

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

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

```
CPU times: user 2.7 s, sys: 91 ms, total: 2.79 s
Wall time: 2.8 s
```

[41]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=0, splitter='best'),
```

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

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

[42]:
```
{'max_depth': 4, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

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

[43]:
```
0.969819392792457
```

[44]:
```
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
        model_name (string): what you want the model to be called in the output␣
↪table
        model_object: a fit GridSearchCV object
        metric (string): precision, recall, f1, accuracy, or auc

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

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

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

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

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

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

    return table
```

[45]:
```
# Get all CV scores
tree1_cv_results = make_results('decision tree cv', tree1, 'auc')
tree1_cv_results
```

[45]:

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

[46]:
```
# Instantiate model
rf = RandomForestClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

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

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

[47]:
```
%%time
rf1.fit(X_train, y_train) # --> Wall time: ~10min
```

```
CPU times: user 9min 7s, sys: 3.01 s, total: 9min 10s
Wall time: 9min 10s
```

[47]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,…
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'f1', 'precision', 'accuracy', 'roc_auc', 'recall'},
             verbose=0)
```

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

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

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

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

[50]:
```
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
```

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

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

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

[53]:
```
0.9804250949807172
```

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

[54]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 500}
```

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

```
              model  precision    recall        F1  accuracy       auc
0  decision tree cv   0.914552  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
```

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

    In:
```

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

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

    preds = model.best_estimator_.predict(X_test_data)

    auc = roc_auc_score(y_test_data, preds)
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)

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

    return table
```

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

[57]:

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

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

# Display first few rows of new dataframe
df2.head()
```

[58]:

	last_evaluation	number_project	average_monthly_hours	tenure	\
0	0.53	2	157	3	
1	0.86	5	262	6	
2	0.88	7	272	4	
3	0.87	5	223	5	
4	0.52	2	159	3	

```
   work_accident  left  promotion_last_5years  salary  department_IT  \
0              0     1                      0       0              0
1              0     1                      0       1              0
2              0     1                      0       1              0
3              0     1                      0       0              0
4              0     1                      0       0              0

   department_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
```

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

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

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

[60]:
```
# Define `overworked` as working > 175 hrs/week
df2['overworked'] = (df2['overworked'] > 175).astype(int)

# Display first few rows of new column
df2['overworked'].head()
```

[60]:
```
0    0
1    1
```

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

[61]:
```
# Drop the `average_monthly_hours` column
df2 = df2.drop('average_monthly_hours', axis=1)

# Display first few rows of resulting dataframe
df2.head()
```

[61]:

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

[62]:
```
# Isolate the outcome variable
y = df2['left']

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

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

[64]:
```
# Instantiate model
tree = DecisionTreeClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }

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

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

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

```
CPU times: user 2.49 s, sys: 1.36 ms, total: 2.49 s
Wall time: 2.49 s
```

[65]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=0, splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
```

```
                   pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
                   scoring={'f1', 'precision', 'accuracy', 'roc_auc', 'recall'},
                   verbose=0)
```

[66]:
```
# Check best params
tree2.best_params_
```

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

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

[67]:
```
0.9586752505340426
```

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

```
              model  precision    recall        F1  accuracy       auc
0  decision tree cv   0.914552  0.916949  0.915707  0.971978  0.969819
               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.958675
```

[69]:
```
# Instantiate model
rf = RandomForestClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

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

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

[70]:
```
%%time
rf2.fit(X_train, y_train) # --> Wall time: 7min 5s
```

```
CPU times: user 7min 10s, sys: 1.17 s, total: 7min 11s
Wall time: 7min 11s
```

[70]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,…
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'f1', 'precision', 'accuracy', 'roc_auc', 'recall'},
             verbose=0)
```

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

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

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

[73]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 300}
```

[74]:
```
# Check best AUC score on CV
rf2.best_score_
```

[74]:
```
0.9648100662833985
```

[75]:
```
# Get all CV scores
rf2_cv_results = make_results('random forest2 cv', rf2, 'auc')
```

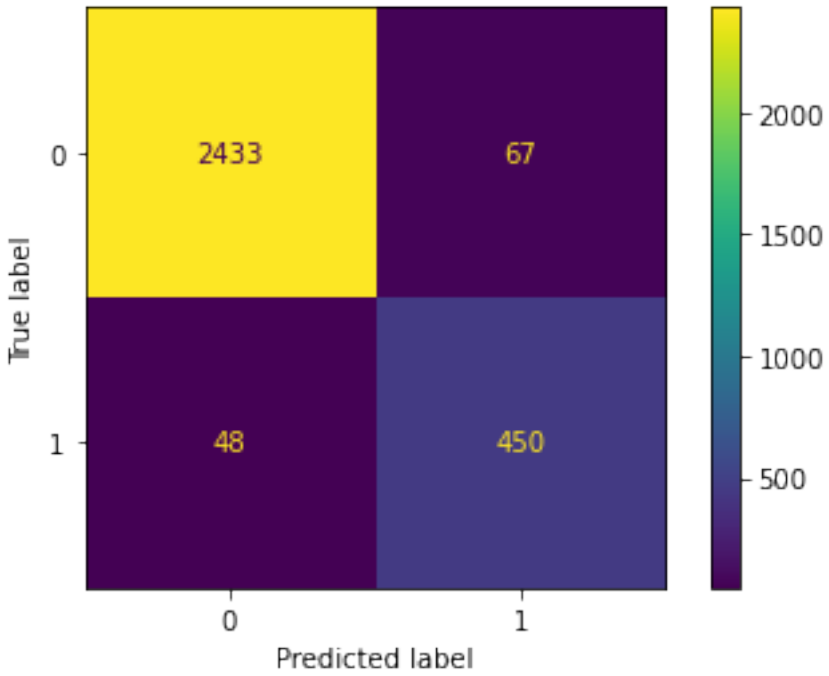
```
print(tree2_cv_results)
print(rf2_cv_results)
```

```
                model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.958675
                model  precision    recall        F1  accuracy      auc
0  random forest2 cv   0.866758  0.878754  0.872407  0.957411  0.96481
```

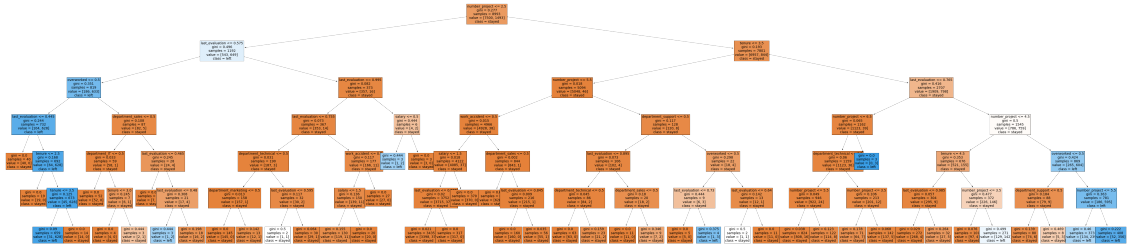
[76]:
```
# Get predictions on test data
rf2_test_scores = get_scores('random forest2 test', rf2, X_test, y_test)
rf2_test_scores
```

[76]:
```
                 model  precision    recall      f1  accuracy       AUC
0  random forest2 test   0.870406  0.903614  0.8867  0.961641  0.938407
```

[77]:
```
# Generate array of values for confusion matrix
preds = rf2.best_estimator_.predict(X_test)
cm = confusion_matrix(y_test, preds, labels=rf2.classes_)

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



[78]:
```
# Plot the tree
plt.figure(figsize=(85,20))
plot_tree(tree2.best_estimator_, max_depth=6, fontsize=14, feature_names=X.
 ↪columns,
          class_names={0:'stayed', 1:'left'}, filled=True);
plt.show()
```



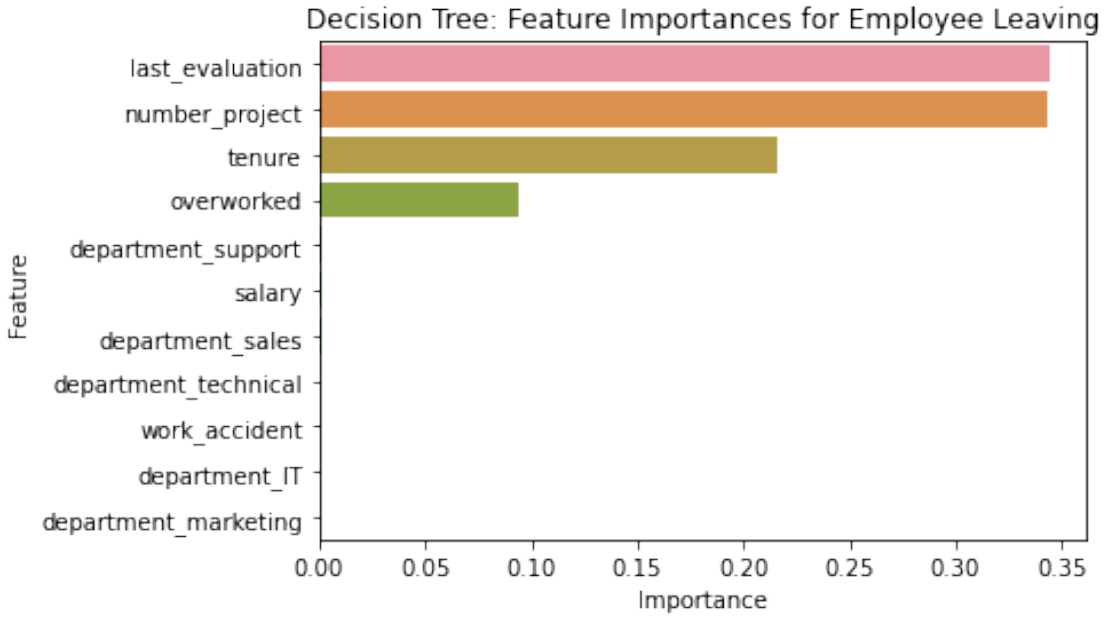
[79]:
```
#tree2_importances = pd.DataFrame(tree2.best_estimator_.feature_importances_,␣
 ↪columns=X.columns)
tree2_importances = pd.DataFrame(tree2.best_estimator_.feature_importances_,
                                 columns=['gini_importance'],
                                 index=X.columns
                                )
tree2_importances = tree2_importances.sort_values(by='gini_importance',␣
 ↪ascending=False)

# Only extract the features with importances > 0
tree2_importances = tree2_importances[tree2_importances['gini_importance'] != 0]
tree2_importances
```

[79]:

	gini_importance
last_evaluation	0.343958
number_project	0.343385
tenure	0.215681
overworked	0.093498
department_support	0.001142
salary	0.000910
department_sales	0.000607
department_technical	0.000418
work_accident	0.000183
department_IT	0.000139
department_marketing	0.000078

[80]:
```
sns.barplot(data=tree2_importances, x="gini_importance", y=tree2_importances.
 ↪index, orient='h')
```

```
plt.title("Decision Tree: Feature Importances for Employee Leaving",␣
 ↪fontsize=12)
plt.ylabel("Feature")
plt.xlabel("Importance")
plt.show()
```



[81]:
```
# Get feature importances
feat_impt = rf2.best_estimator_.feature_importances_

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

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

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

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

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

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

plt.show()
```