Activity_ Course 7 Salifort Motors project lab

December 15, 2025

1 Capstone project: Providing data-driven suggestions for HR

1.1 Description and deliverables

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

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

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

2 PACE stages

2.1 Pace: Plan

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

In this stage, consider the following:

2.1.1 Understand the business scenario and problem

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

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

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

2.1.2 Familiarize yourself with the HR dataset

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

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

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

Reflect on these questions as you complete the plan stage.

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

[Double-click to enter your responses here.]

2.2 Step 1. Imports

- Import packages

- Load dataset

2.2.1 Import packages

[8]:
```
# Import packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
↪f1_score, confusion_matrix, ConfusionMatrixDisplay, classification_report,␣
↪roc_curve, roc_auc_score

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

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

import pickle

pd.set_option('display.max_columns', None)
```

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.

[9]:
```
# 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
```

```
df0.head()
```

[9]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	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

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

[10]:
```
# 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
```

2.3.2 Gather descriptive statistics about the data

[12]:
```
# Gather descriptive statistics about the data
df0.describe()
```

[12]:

	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.

[13]:
```
# Display all column names
df0.columns
```

[13]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
       'promotion_last_5years', 'Department', 'salary'],
      dtype='object')
```

[16]:
```
# Rename columns as needed
df0 = df0.rename(columns = {'Work_accident' : 'work_accident',
                            'Department':'department',
                            'average_montly_hours':'average_monthly_hours',
                            'time_spend_company':'tenure',
                             })


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

[16]:
```
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.

[17]:
```
# Check for missing values


df0.isna().sum()
```

[17]:
```
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.

[20]:
```
# Check for duplicates

df0.duplicated().sum()
```

[20]:
```
3008
```

[21]:
```
# Inspect some rows containing duplicates as needed

df0[df0.duplicated()].head()
```

[21]:
```
      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
```

[22]:
```
# 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()
```

[22]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
```

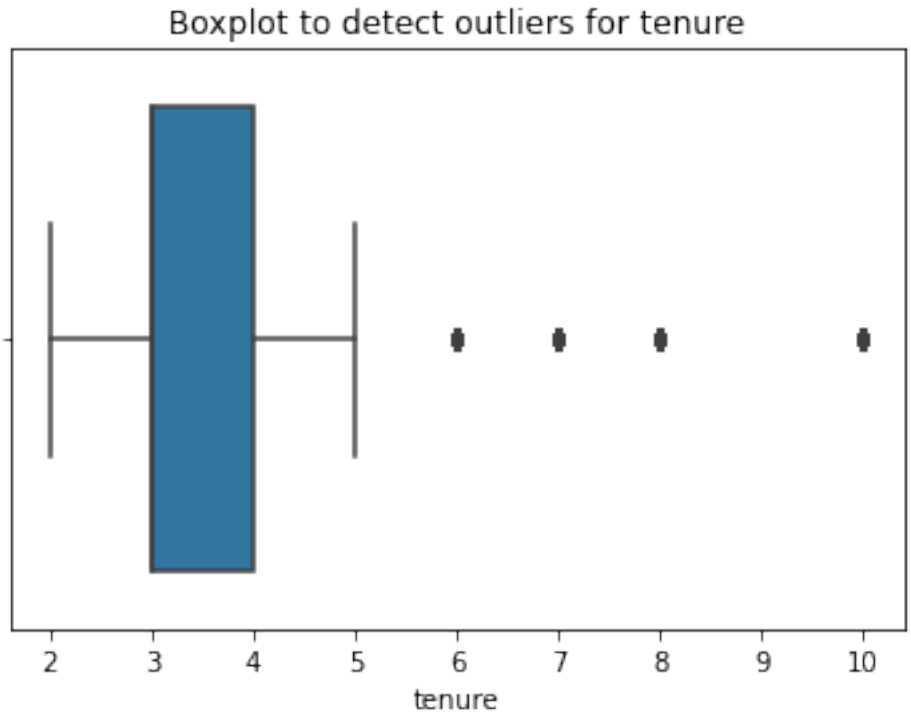
```
2                0.11             0.88              7                     272
3                0.72             0.87              5                     223
4                0.37             0.52              2                     159
```

```
   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.

[25]:
```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers

sns.boxplot(x = df1['tenure'])
plt.title('Boxplot to detect outliers for tenure')
plt.show()
```



[29]:
```
# Determine the number of rows containing outliers

percentile25 = df1['tenure'].quantile(0.25)
percentile75 = df1['tenure'].quantile(0.75)

iqr = percentile75 - percentile25

upper_limit = percentile75 + 1.5*iqr
lower_limit = percentile25 - 1.5*iqr

print('Upper Limit is:',upper_limit)
print('Lower Limit is:',lower_limit)

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

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

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

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

3 pAce: Analyze Stage

- Perform EDA (analyze relationships between variables)

Reflect on these questions as you complete the analyze stage.

- What did you observe about the relationships between variables?
- What do you observe about the distributions in the data?
- What transformations did you make with your data? Why did you chose to make those decisions?
- What are some purposes of EDA before constructing a predictive model?
- What resources do you find yourself using as you complete this stage? (Make sure to include the links.)
- Do you have any ethical considerations in this stage?

[Double-click to enter your responses here.]

3.1 Step 2. Data Exploration (Continue EDA)

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

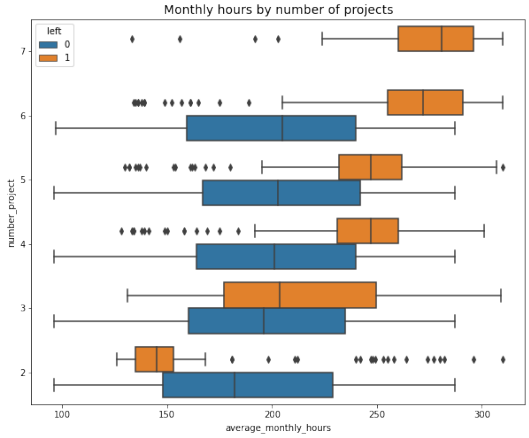
[32]:
```
# Get numbers of people who left vs. stayed

print(df1['left'].value_counts())
print()

# Get percentages of people who left vs. stayed

print(df1['left'].value_counts(normalize=True))
```

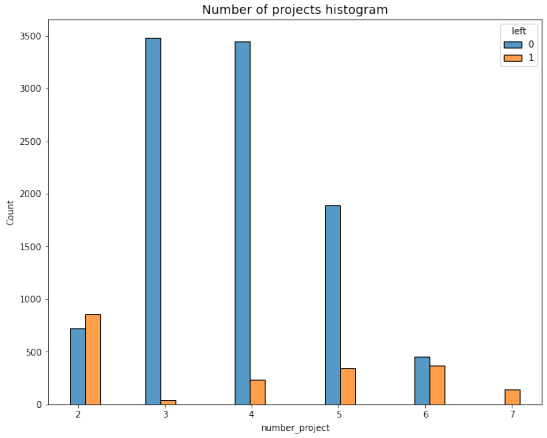
```
0    10000
1     1991
Name: left, dtype: int64

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.

[40]:
```
# Create a plot as needed

fig, ax = plt.subplots(1, 2, figsize = (22,8))

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')

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()
```

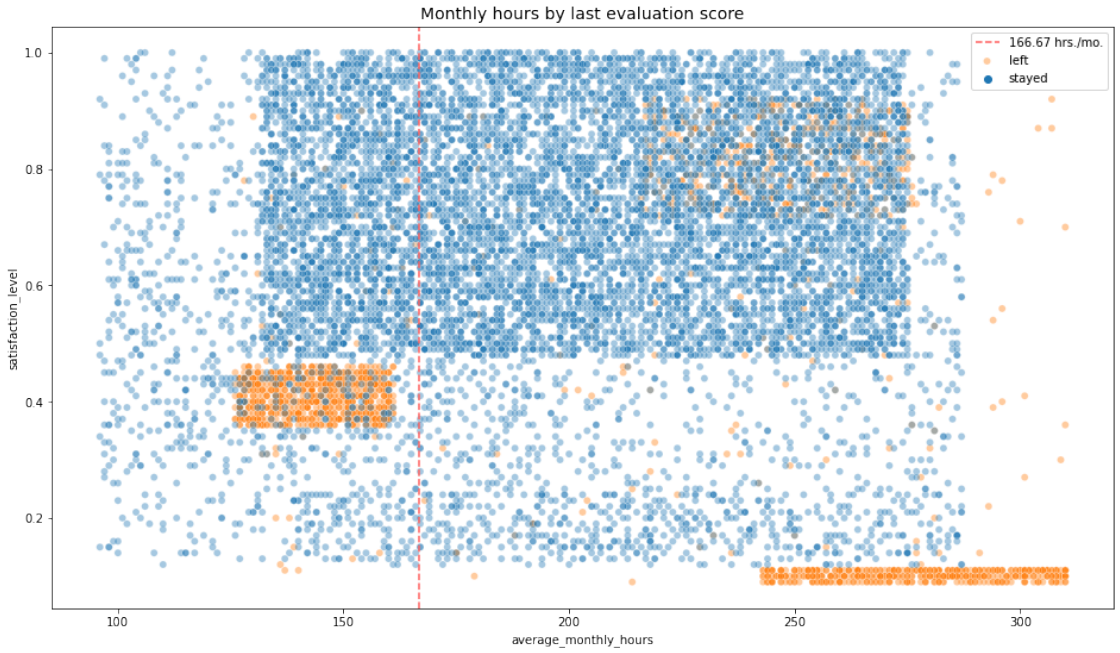




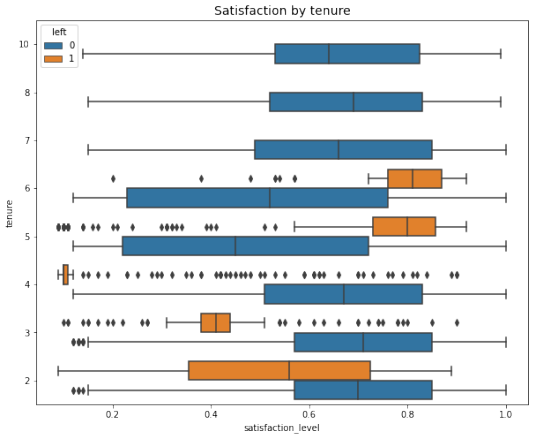
[43]:
```
# Create a plot as needed

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

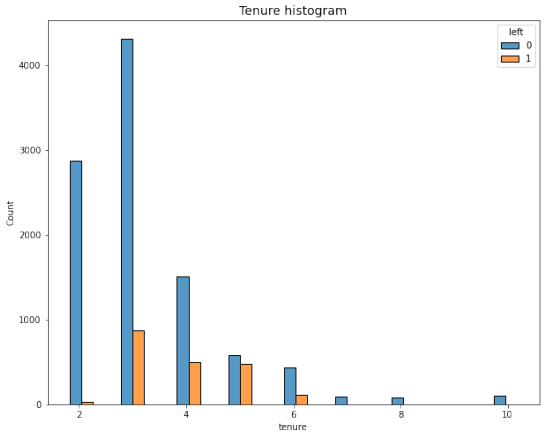
[43]:
```
1    145
Name: left, dtype: int64
```

[44]:
```
# Create a plot as needed

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');
```



[45]:
```
# Create a plot as needed

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();
```



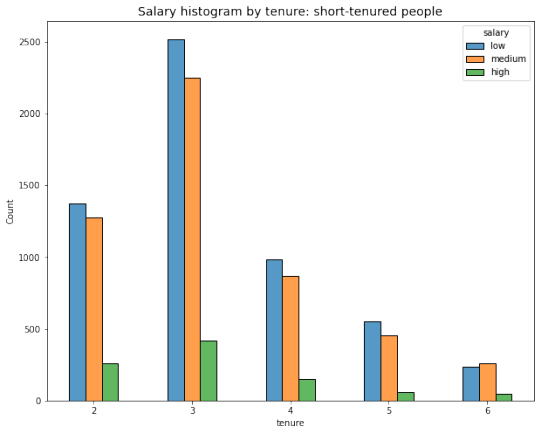


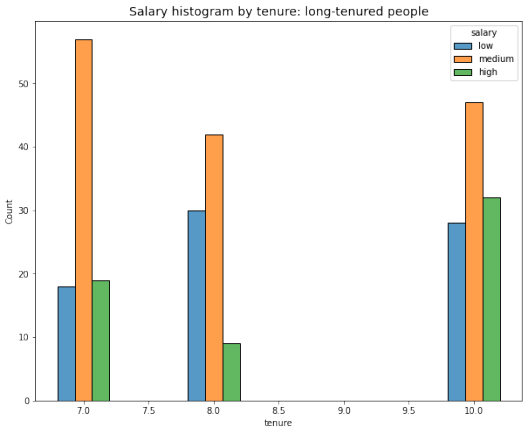
[46]:
```
# Create a plot as needed

df1.groupby(['left'])['satisfaction_level'].agg([np.mean,np.median])
```

[46]:

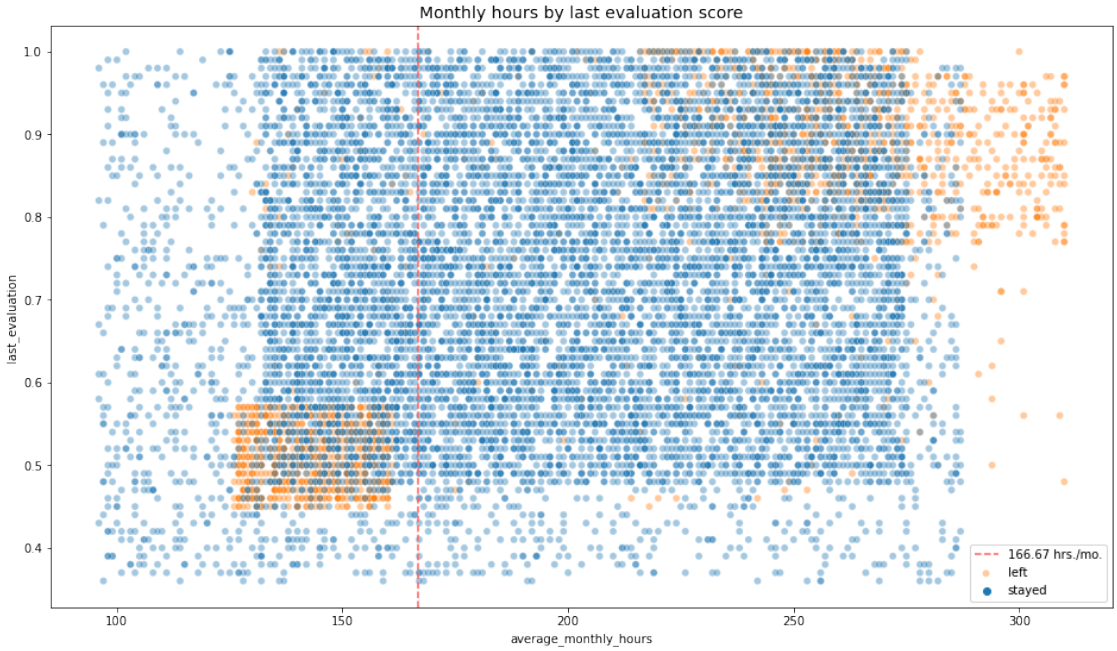
left	mean	median
0	0.667365	0.69
1	0.440271	0.41

[47]:
```
# Create a plot as needed

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');
```

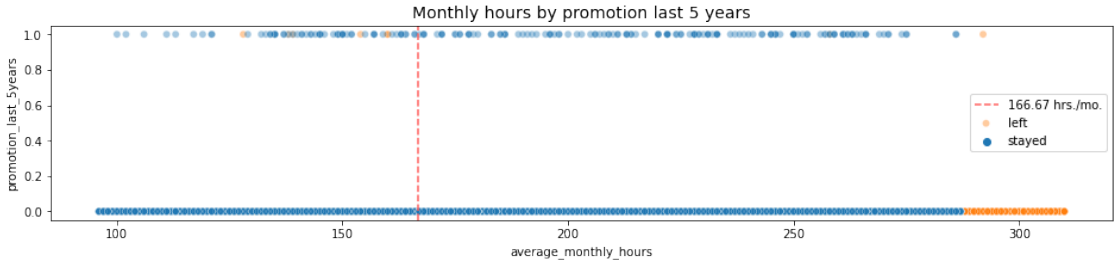




[49]:
```
# Create a plot as needed

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');
```



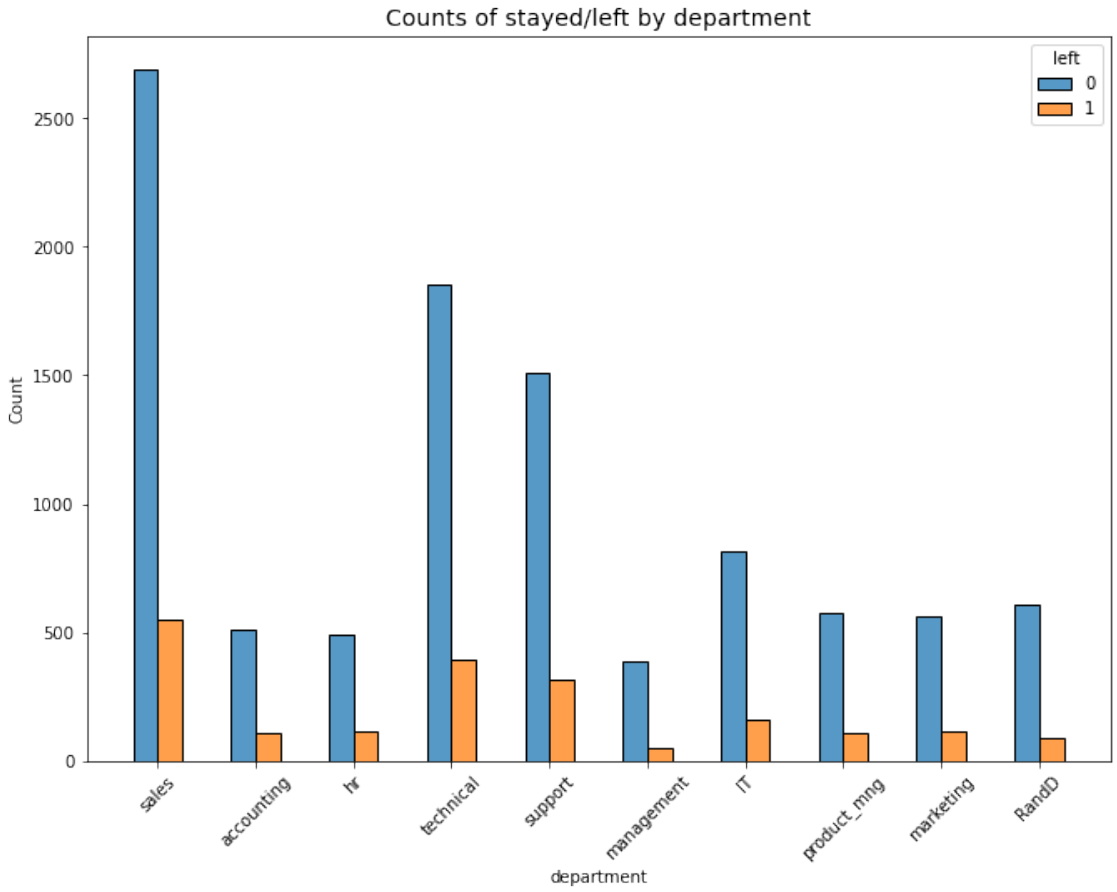
[50]:
```
# Create a plot as needed

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');
```



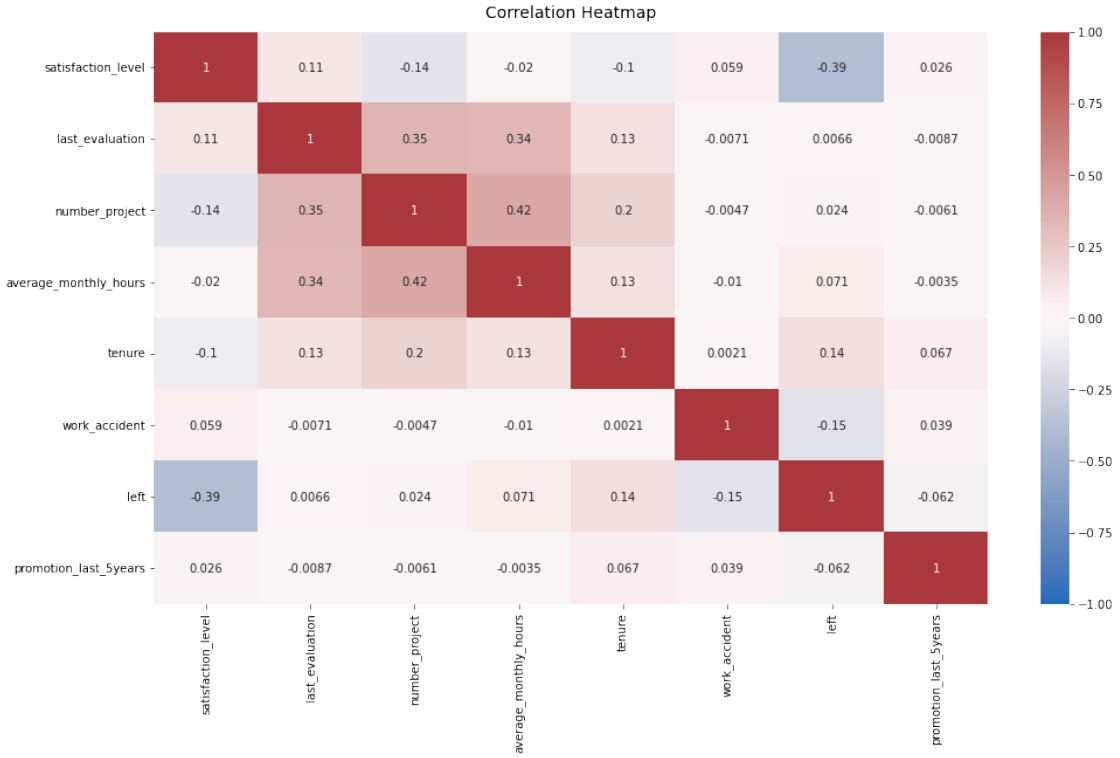
[51]:
```
df1["department"].value_counts()
```

[51]:
```
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
```

[52]:
```
# 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);
```



[53]:
```
# 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);
```



3.1.2 Insights

[What insights can you gather from the plots you created to visualize the data? Double-click to enter your responses here.]

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.

[Double-click to enter your responses here.]

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

[Double-click to enter your responses here.]

4.1.3 Modeling

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

[54]:
```
df_enc = df1.copy()
```

[61]:
```
y = df_enc['left']

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

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()
```

[61]:

	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

5 pacE: Execute Stage

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

[62]:
```
x = df_enc.drop('left', axis=1)

x.head()
```

[62]:
```
   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
```

[63]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.25,␣
↪stratify =y, random_state=0)
```

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

cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }

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

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

[65]:
```
%%time

tree1.fit(x_train, y_train)
```

```
CPU times: user 2.92 s, sys: 0 ns, total: 2.92 s
Wall time: 2.92 s
```

[65]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=0, splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'recall', 'f1', 'roc_auc', 'precision', 'accuracy'},
             verbose=0)
```

[66]:
```
tree1.best_params_
```

[66]:
```
{'max_depth': 4, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

[67]:
```
tree1.best_score_
```

[67]:
```
0.969819392792457
```

[69]:
```
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
```

[70]:
```
tree1_cv_results = make_results('decision tree cv', tree1, 'auc')
tree1_cv_results
```

[70]:

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

[71]:
```
rf = RandomForestClassifier(random_state=0)

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],
             }

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

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

[72]:
```
%%time

rf1.fit(x_train, y_train)
```

```
CPU times: user 9min 30s, sys: 951 ms, total: 9min 31s
Wall time: 9min 31s
```

[72]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,…
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'recall', 'f1', 'roc_auc', 'precision', 'accuracy'},
             verbose=0)
```

[73]:
```
rf1.best_params_
```

[73]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 500}
```

[74]:
```
rf1.best_score_
```

[74]:
```
0.9804250949807172
```

[75]:
```
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
```

[76]:
```
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
```

[77]:
```
rf1_test_scores = get_scores('random forest1 test', rf1, x_test, y_test)
rf1_test_scores
```

[77]:

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

[78]:
```
df2 = df_enc.drop('satisfaction_level', axis=1)
df2.head()
```

[78]:

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

```
3                  1                       0                       0
4                  1                       0                       0
```

[79]:
```
# 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
```

[85]:
```
df2['overworked'] = (df2['overworked'] > 175).astype(int)

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

[85]:
```
0    0
1    0
2    0
3    0
4    0
Name: overworked, dtype: int64
```

[86]:
```
df2 = df2.drop('average_monthly_hours', axis=1)
df2.head()
```

[86]:
```
   last_evaluation  number_project  tenure  work_accident  left  \
0             0.53               2       3              0     1
1             0.86               5       6              0     1
2             0.88               7       4              0     1
3             0.87               5       5              0     1
4             0.52               2       3              0     1

   promotion_last_5years  salary  department_IT  department_RandD  \
0                      0       0              0                 0
1                      0       1              0                 0
2                      0       1              0                 0
3                      0       0              0                 0
4                      0       0              0                 0

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

```
4                       0              0                       0

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

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

[87]:
```
y = df2['left']
x = df2.drop('left', axis=1)
```

[88]:
```
x_train, x_test, y_train,y_test = train_test_split(x,y,test_size=0.25,␣
↪stratify=y,random_state=0
                                                   )
```

[93]:
```
tree2 = DecisionTreeClassifier(random_state=0)

cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }

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

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

[94]:
```
%%time
tree2.fit(x_train,y_train)
```

```
CPU times: user 2.44 s, sys: 0 ns, total: 2.44 s
Wall time: 2.43 s
```

[94]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
```

```
                                       min_samples_leaf=1,
                                       min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       presort='deprecated',
                                       random_state=0, splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'recall', 'f1', 'roc_auc', 'precision', 'accuracy'},
             verbose=0)
```

[96]:
```
tree2.best_params_
```

[96]:
```
{'max_depth': 6, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

[97]:
```
tree2.best_score_
```

[97]:
```
0.9564598419138554
```

[98]:
```
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   0.844484  0.898862  0.870651  0.955632  0.95646
```

[99]:
```
rf = RandomForestClassifier(random_state=0)

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],
             }

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

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

[100]:
```
%%time
rf2.fit(x_train, y_train)
```

```
CPU times: user 6min 52s, sys: 900 ms, total: 6min 53s
Wall time: 6min 53s
```

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

[102]:
```
rf2.best_params_
```

[102]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 300}
```

[104]:
```
rf2.best_score_
```

[104]:
```
0.9626098263824174
```

[106]:
```
rf2_cv_results = make_results('random forest2 cv', rf2, 'auc')
print(tree2_cv_results)
print(rf2_cv_results)
```

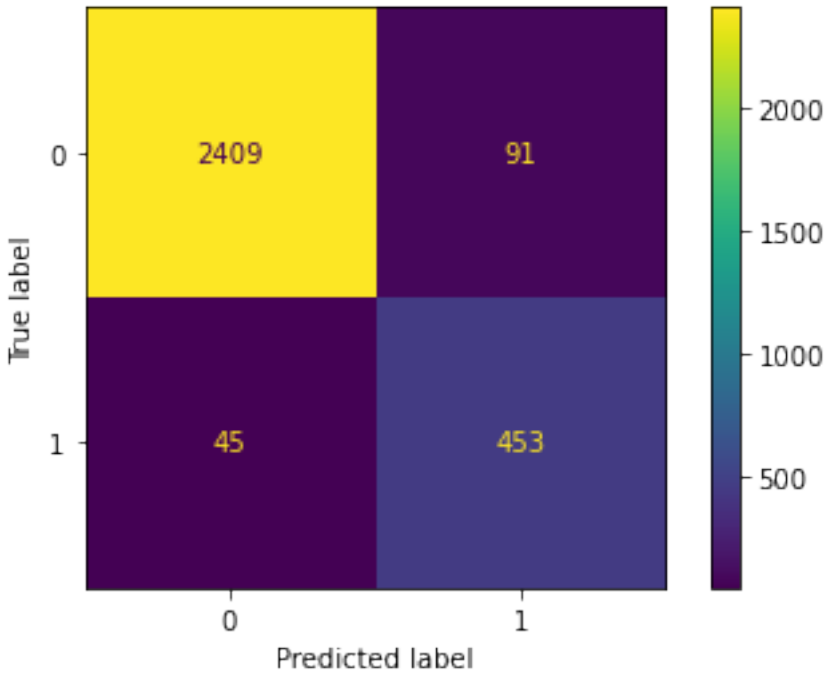
```
                model  precision    recall        F1  accuracy      auc
0  decision tree2 cv   0.844484  0.898862  0.870651  0.955632  0.95646
                model  precision   recall        F1  accuracy      auc
0  random forest2 cv   0.851347  0.86066  0.855515   0.95174  0.96261
```

[107]:
```
rf2_test_scores = get_scores('random forest2 test', rf2, x_test, y_test)
rf2_test_scores
```

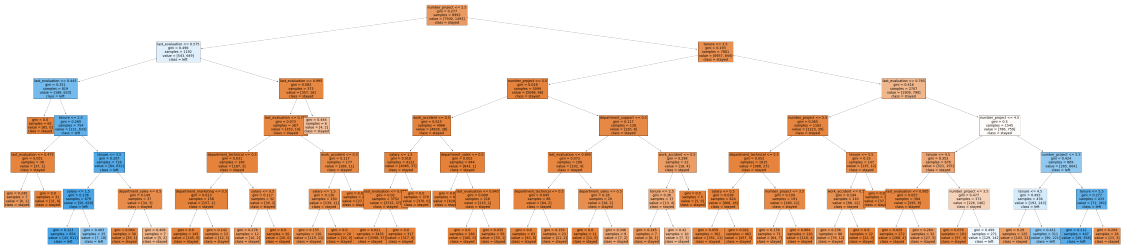
[107]:

	model	precision	recall	f1	accuracy	AUC
0	random forest2 test	0.832721	0.909639	0.869482	0.954636	0.936619

[108]:
```
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='');
```



[110]:
```
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()
```

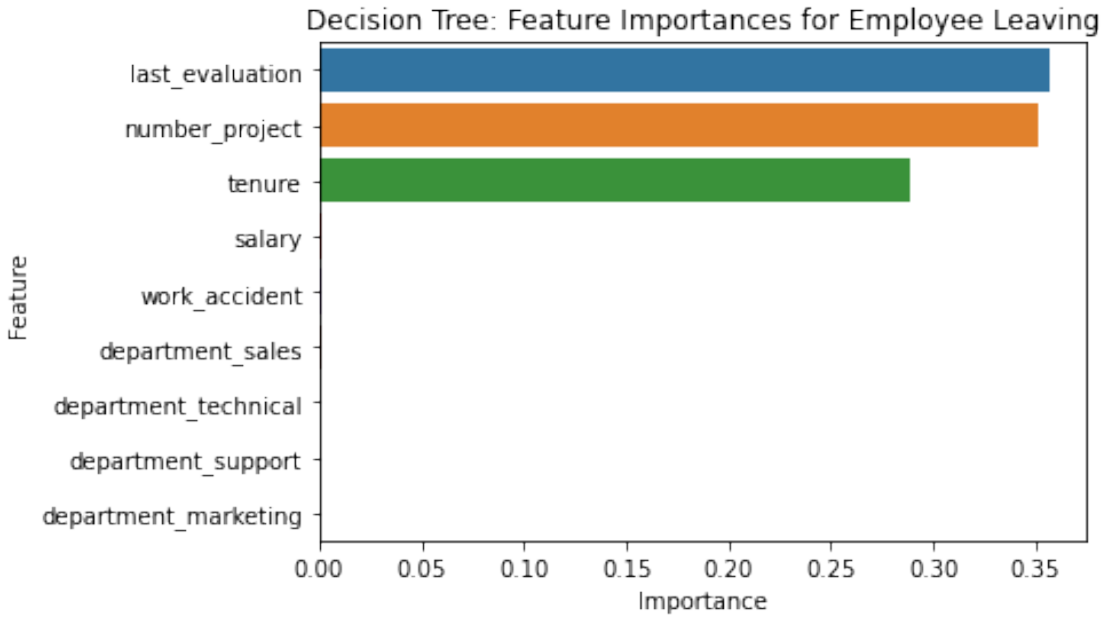


[111]:
```
tree2_importances = pd.DataFrame(tree2.best_estimator_.feature_importances_,
                                 columns=['gini_importance'],
                                 index=x.columns
                                )
tree2_importances = tree2_importances.sort_values(by='gini_importance',␣
↪ascending=False)

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

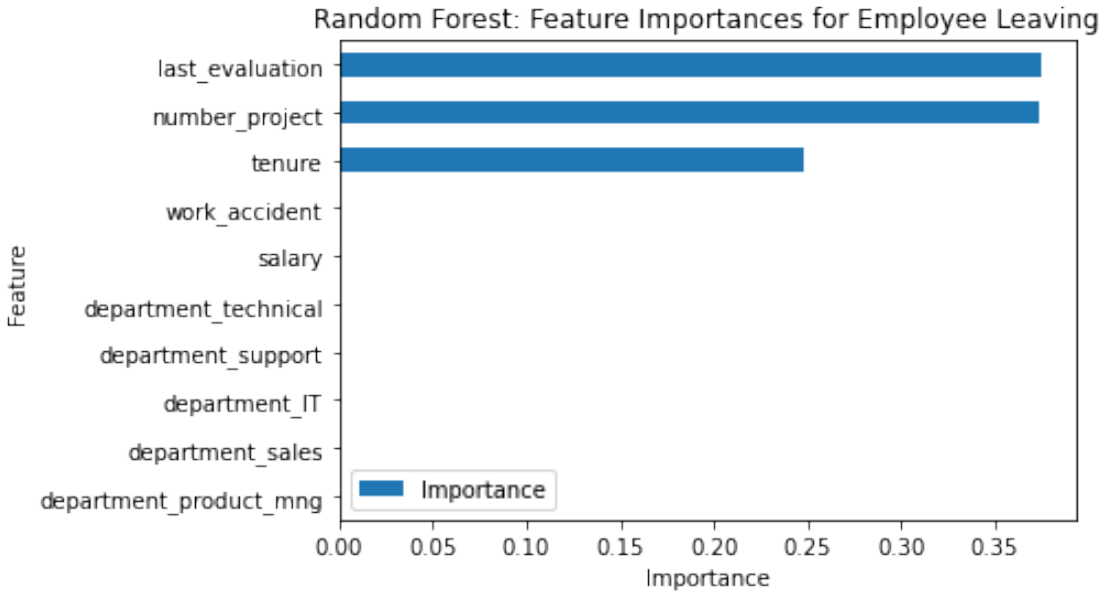
[111]:

	gini_importance
last_evaluation	0.356608
number_project	0.350883
tenure	0.288319
salary	0.001730
work_accident	0.000790
department_sales	0.000681
department_technical	0.000489
department_support	0.000423
department_marketing	0.000079

[115]:
```
sns.barplot(data=tree2_importances, x='gini_importance', y=tree2_importances.
↪index)
plt.title('Decision Tree: Feature Importances for Employee Leaving',␣
↪fontsize=12)
plt.ylabel('Feature')
plt.xlabel('Importance')
plt.show()
```



[116]:
```
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()
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

Recall evaluation metrics

- **AUC** is the area under the ROC curve; it's also considered the probability that the model ranks a random positive example more highly than a random negative example.
- **Precision** measures the proportion of data points predicted as True that are actually True, in other words, the proportion of positive predictions that are true positives.

- **Recall** measures the proportion of data points that are predicted as True, out of all the data points that are actually True. In other words, it measures the proportion of positives that are correctly classified.
- **Accuracy** measures the proportion of data points that are correctly classified.
- **F1-score** is an aggregation of precision and recall.

Reflect on these questions as you complete the executing stage.

- What key insights emerged from your model(s)?
- What business recommendations do you propose based on the models built?
- What potential recommendations would you make to your manager/company?
- Do you think your model could be improved? Why or why not? How?
- Given what you know about the data and the models you were using, what other questions could you address for the team?
- What resources do you find yourself using as you complete this stage? (Make sure to include the links.)
- Do you have any ethical considerations in this stage?

Double-click to enter your responses here.

5.1 Step 4. Results and Evaluation

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

5.1.1 Summary of model results

[Double-click to enter your summary here.]

5.1.2 Conclusion, Recommendations, Next Steps

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

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