CAPSTONE PROJECT - SALIFORT MOTORS

February 9, 2024

1 Capstone project: Providing data-driven suggestions for HR

1.1 Description and deliverables

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

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

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

2 PACE stages

2.1 Pace: Plan

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

In this stage, consider the following:

2.1.1 Understand the business scenario and problem

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

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

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

2.1.2 Familiarize yourself with the HR dataset

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

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

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

Reflect on these questions as you complete the plan stage.

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

[Double-click to enter your responses here.]

2.2 Step 1. Imports

- Import packages

- Load dataset

2.2.1 Import packages

[53]:
```
# Import packages

# Import packages for data manipulation
import pandas as pd
import numpy as np

# Import packages for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Import packages for data preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.utils import resample

# Import packages for statistical analysis/hypothesis testing
from scipy import stats

# Import packages for data modeling
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score,precision_score, recall_score,␣
↪f1_score

# Import packages for machine learning modeling
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from xgboost import plot_importance
```

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.

[54]:
```
# 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()
```

[54]:
```
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7                   272
3                0.72             0.87               5                   223
4                0.37             0.52               2                   159

   time_spend_company  Work_accident  left  promotion_last_5years Department  \
0                   3              0     1                      0      sales
1                   6              0     1                      0      sales
2                   4              0     1                      0      sales
3                   5              0     1                      0      sales
4                   3              0     1                      0      sales

   salary
0     low
1  medium
2  medium
3     low
4     low
```

2.3 Step 2. Data Exploration (Initial EDA and data cleaning)

- Understand your variables
- Clean your dataset (missing data, redundant data, outliers)

2.3.1 Gather basic information about the data

[55]:
```
# 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

[56]:
```
# Gather descriptive statistics about the data
df0.describe()
```

[56]:
```
       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.

[57]:
```
# Display all column names
df0.columns
```

[57]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
       'promotion_last_5years', 'Department', 'salary'],
      dtype='object')
```

[58]:
```
# Rename columns as needed
df0 = df0.rename(columns={'Work_accident': 'work_accident',␣
↪'promotion_last_5years': 'promoted_5_years',
                          'average_monthly_hours': 'avg_monthly_hours',␣
↪'number_project': 'projects_worked',
                         'last_evaluation': 'last_eval_score', 'Department':␣
↪'department',
                         'time_spend_company': 'tenure', })


# Display all column names after the update
df0.head()
```

[58]:

	satisfaction_level	last_eval_score	projects_worked	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	

	tenure	work_accident	left	promoted_5_years	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

2.3.4 Check missing values

Check for any missing values in the data.

[59]:
```
# Check for missing values
df0.isna().sum()
```

[59]:
```
satisfaction_level      0
last_eval_score         0
projects_worked         0
average_montly_hours    0
tenure                  0
work_accident           0
left                    0
promoted_5_years        0
department              0
salary                  0
dtype: int64
```

2.3.5 Check duplicates

Check for any duplicate entries in the data.

[60]:
```python
# Check for duplicates
df0.duplicated(keep=False).sum()
```

[60]: 5346

[61]:
```python
# Inspect some rows containing duplicates as needed
# 1- Identify the rows with duplicated values

duplicates = df0[df0.duplicated(keep=False) | df0.duplicated(keep='first')]

# 2- Group the original and duplicated rows together
duplicates.sort_values(by=list(duplicates.columns))
```

[61]:

	satisfaction_level	last_eval_score	projects_worked	\
30	0.09	0.62	6	
12030	0.09	0.62	6	
14241	0.09	0.62	6	
71	0.09	0.77	5	
12071	0.09	0.77	5	
…	…	…	…	
13089	1.00	0.88	6	
11375	1.00	0.93	5	
13586	1.00	0.93	5	
10691	1.00	0.93	5	
12902	1.00	0.93	5	

	average_montly_hours	tenure	work_accident	left	promoted_5_years	\
30	294	4	0	1	0	
12030	294	4	0	1	0	
14241	294	4	0	1	0	

```
71                     275       4             0     1                 0
12071                  275       4             0     1                 0
…                    …     …            …    …              …
13089                  201       4             0     0                 0
11375                  167       3             0     0                 0
13586                  167       3             0     0                 0
10691                  231       2             0     0                 0
12902                  231       2             0     0                 0

        department  salary
30      accounting     low
12030   accounting     low
14241   accounting     low
71     product_mng  medium
12071  product_mng  medium
…             …      …
13089    technical     low
11375        sales  medium
13586        sales  medium
10691    marketing  medium
12902    marketing  medium

[5346 rows x 10 columns]
```

[62]:

```
# 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
print('num of dupes after drop', df1.duplicated().sum())
df1.head()
```

```
num of dupes after drop 0
```

[62]:

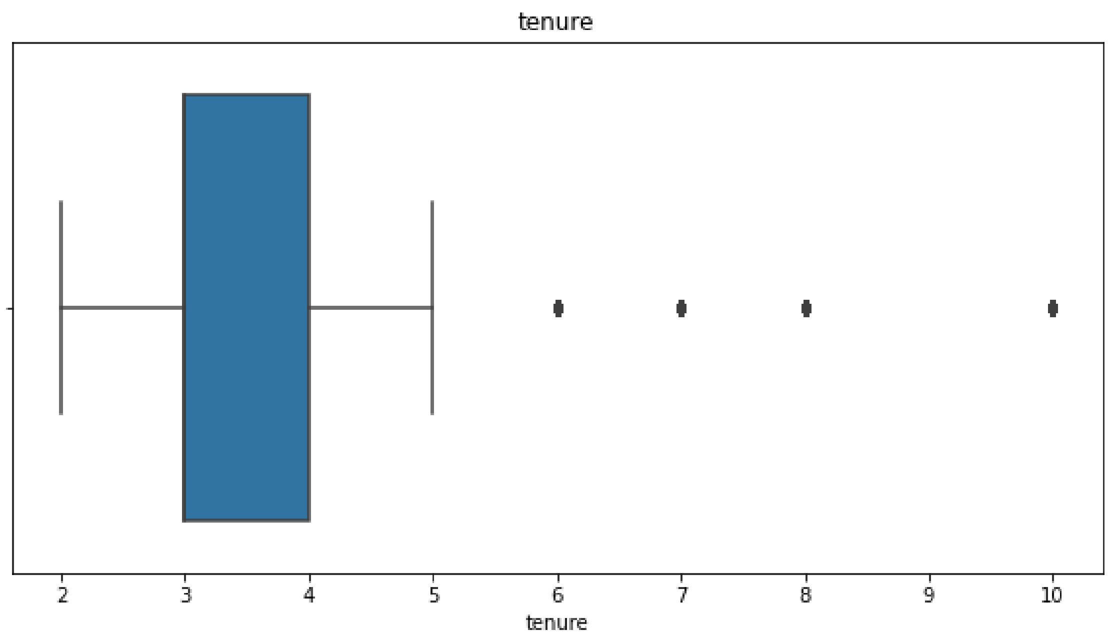
```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.38             0.53                2                   157
1                0.80             0.86                5                   262
2                0.11             0.88                7                   272
3                0.72             0.87                5                   223
4                0.37             0.52                2                   159

   tenure  work_accident  left  promoted_5_years 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.

[63]:
```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
plt.figure(figsize=(10,5))
plt.title('tenure')
sns.boxplot(x=df1['tenure'])
```

[63]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f4704312dd0>`



[64]:
```
# Determine the number of rows containing outliers
(df1['tenure'] > 5).sum()
```

[64]: 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.

[65]:
```
# Get numbers of people who left vs. stayed
print(df1['left'].value_counts())

# Get percentages of people who left vs. stayed
left_company = (df1['left'] == 1).sum()
stayed = (df1['left'] == 0).sum()
total = left_company + stayed

left_company_percent = (left_company / total) * 100
stayed_company_percent = (stayed / total) * 100

print('Left Company Percentage: ', round(left_company_percent, 3))
print('Stayed Company Percentage: ', round(stayed_company_percent, 3))
```

```
0    10000
1     1991
Name: left, dtype: int64
Left Company Percentage:  16.604
Stayed Company Percentage:  83.396
```

3.1.1 Data visualizations

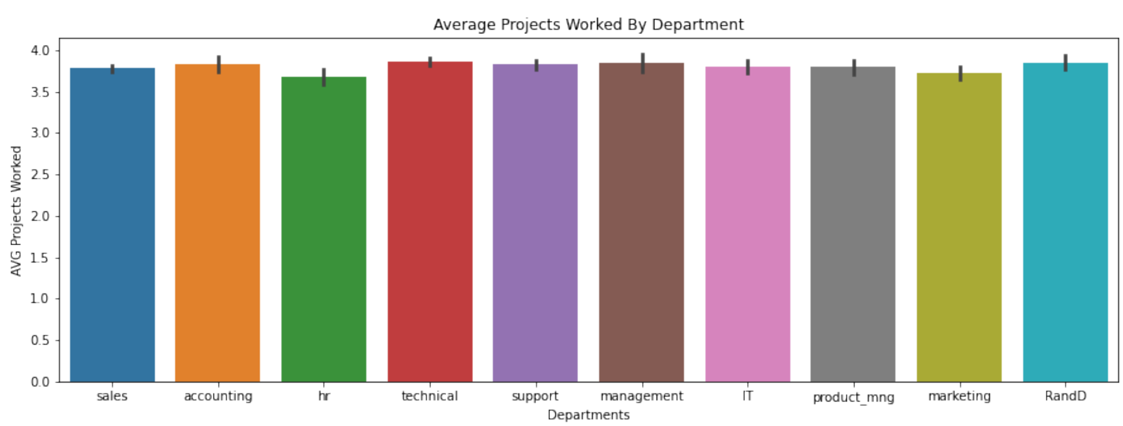
Now, examine variables that you're interested in, and create plots to visualize relationships between variables in the data.

[66]:
```
### PLOTS FOR EARLY EDA ANALYSIS ###
### ADVANCE TO NEXT CODE CELL FOR FURTHER EDA ON 'LEFT' EMPLOYEES ###

# PLOT 1: Barplot - Bar plot of the average number of projects worked by an␣
↪employee based on the deparment
plt.figure(figsize=(15,5))
sns.barplot(data=df1, x='department', y='projects_worked')
```

```
plt.title('Average Projects Worked By Department')
plt.xlabel('Departments')
plt.ylabel('AVG Projects Worked')

plt.show()

# Actual Figures
projects_by_dept = df1.groupby(['department']).
↪mean(numeric_only=True)['projects_worked']

print(projects_by_dept)

# PLOT 2: Histplot - Compare Projects Worked Based On Still With Company

plt.figure(figsize=(7,4))

sns.histplot(data=df1, x='projects_worked', hue='left', multiple='dodge',␣
↪shrink=3)
plt.title('Compare Projects Worked Based On Still With Company')
plt.xlabel('# Projects Worked')
plt.ylabel('Stayed/Left')
plt.show()

# PLOT 3: Barplot - # Projects worked vs Satisfaction level

sns.barplot(data=df1, x='projects_worked', y='satisfaction_level')
plt.title('Compare Projects Worked Based On Satisfaction Level')
plt.xlabel('# Projects Worked')
plt.ylabel('Satisfaction Level')
plt.show()

# PLOT 4: Histplot - Show the number of employees who left vs who stayed based␣
↪on department
plt.figure(figsize=(15,10))
sns.histplot(data=df1, x='department', hue='left', multiple='dodge', shrink=0.5)
plt.title('Number Of Employees Who Left Company Based On Deparment')
plt.xlabel('Deparment')
plt.ylabel('Left vs Stayed')
plt.show()

# Investigating Average Department By Satisfaction Levels
print('Investigating Average Department By Satisfaction Levels')

avg_satisfaction_level_by_dept = df1.groupby(['department']).
↪mean(numeric_only=True)['satisfaction_level']
print(avg_satisfaction_level_by_dept)
```

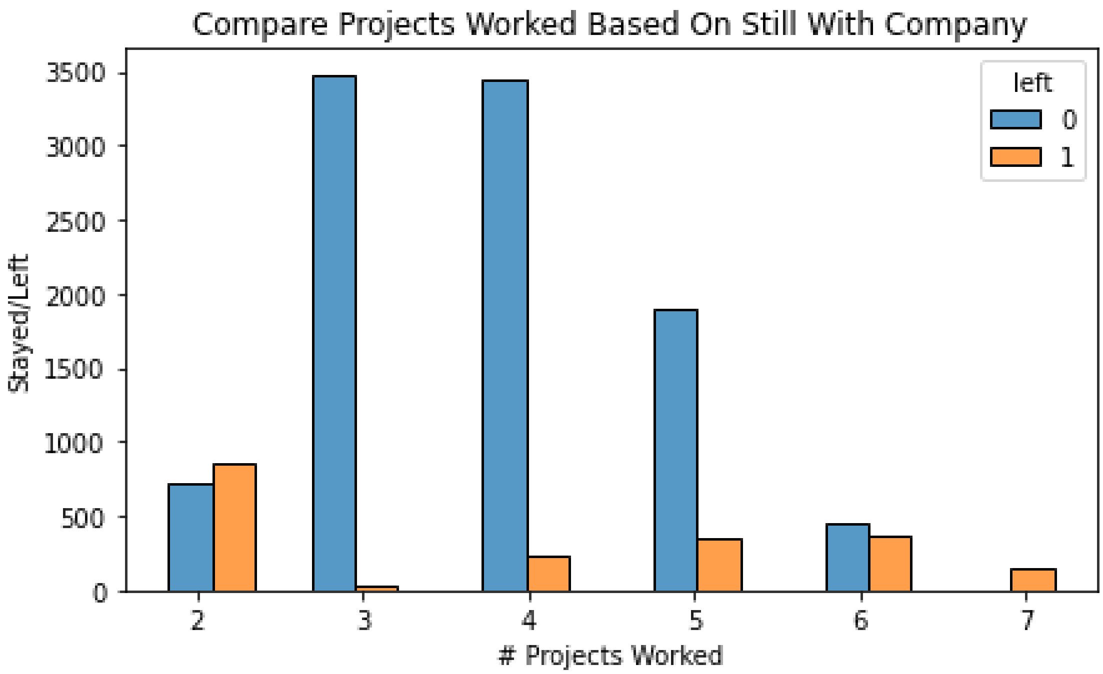
```
avg_last_eval_score_by_dept = df1.groupby(['department']).
 ↪mean(numeric_only=True)['last_eval_score']
print(avg_last_eval_score_by_dept)

average_montly_hours_by_dept = df1.groupby(['department']).
 ↪mean(numeric_only=True)['average_montly_hours']
print(average_montly_hours_by_dept)

# INVESTIGATE EMPLOYEES WHO LEFT COMPANY
print('INVESTIGATE EMPLOYEES WHO LEFT COMPANY')

promoted = df1[df1["promoted_5_years"] == 1]

not_promoted = df1[df1["promoted_5_years"] == 0]

promoted_left = promoted[promoted["left"] == 1]
promoted_stayed = promoted[promoted["left"] == 0]

not_promoted_left = not_promoted[not_promoted["left"] == 1]
not_promoted_stayed = not_promoted[not_promoted["left"] == 0]

'''
plt.figure(figsize=(20,10))
sns.histplot(data=promoted, x='tenure', hue='left', multiple='dodge', shrink=0.
 ↪5)
#plt.title('Number Of Employees Who Left Company Based On Deparment')
#plt.xlabel('Deparment')
#plt.ylabel('Left vs Stayed')
plt.show()

plt.figure(figsize=(10,5))
sns.histplot(data=not_promoted, x='tenure', hue='left', multiple='dodge',␣
 ↪shrink=7)
plt.show()
'''

sns.barplot(data=promoted_left, x='salary', y='last_eval_score', order=['low',␣
 ↪'medium', 'high'])
plt.title('promoted_left')
plt.show()

sns.barplot(data=promoted_stayed, x='salary', y='last_eval_score',␣
 ↪order=['low', 'medium', 'high'] )
plt.title('promoted_stayed')
plt.show()
```

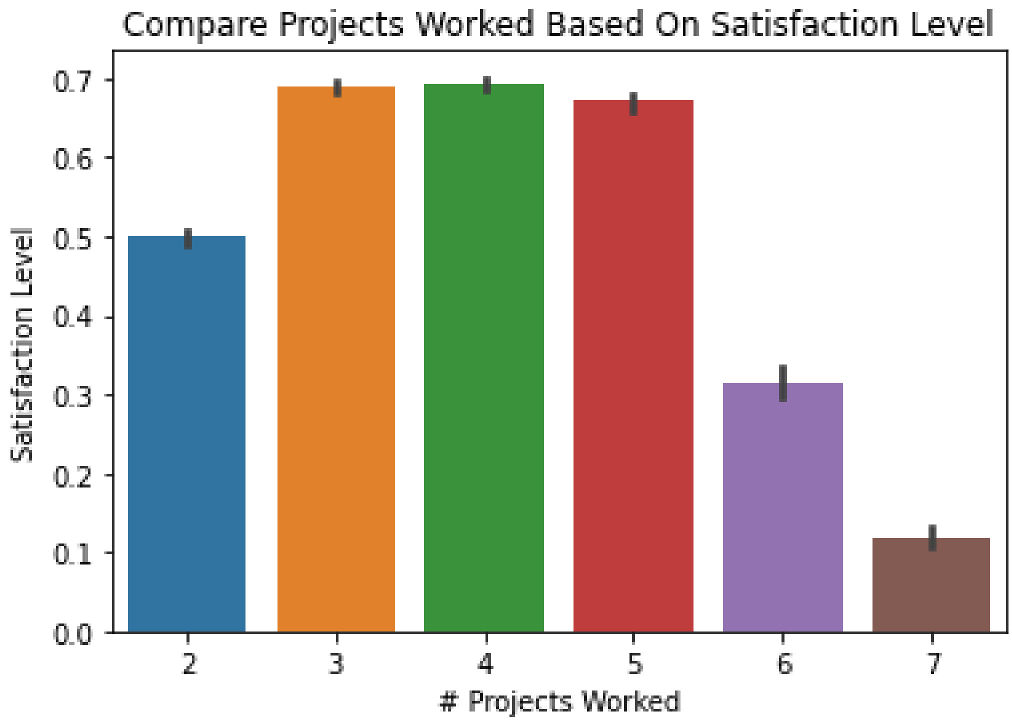
```python
# INVESTIGATE EMPLOYEES WHO LEFT COMPANY BASED ON SALARY
print('INVESTIGATE EMPLOYEES WHO LEFT COMPANY BASED ON SALARY')

left_company = df1[df1["left"] == 1]

left_company_low = left_company[left_company["salary"] == 'low']
left_company_med = left_company[left_company["salary"] == 'medium']
left_company_high = left_company[left_company["salary"] == 'high']

sns.barplot(data=left_company_low, x='tenure', y='satisfaction_level')
plt.title('left_company_low')
plt.show()

print(left_company_low['tenure'].value_counts().sort_values())

sns.barplot(data=left_company_med, x='tenure', y='satisfaction_level')
plt.title('left_company_med')
plt.show()

print(left_company_med['tenure'].value_counts().sort_values())

sns.barplot(data=left_company_high, x='tenure', y='satisfaction_level')
plt.title('left_company_high')
plt.show()

print(left_company_high['tenure'].value_counts().sort_values())
```

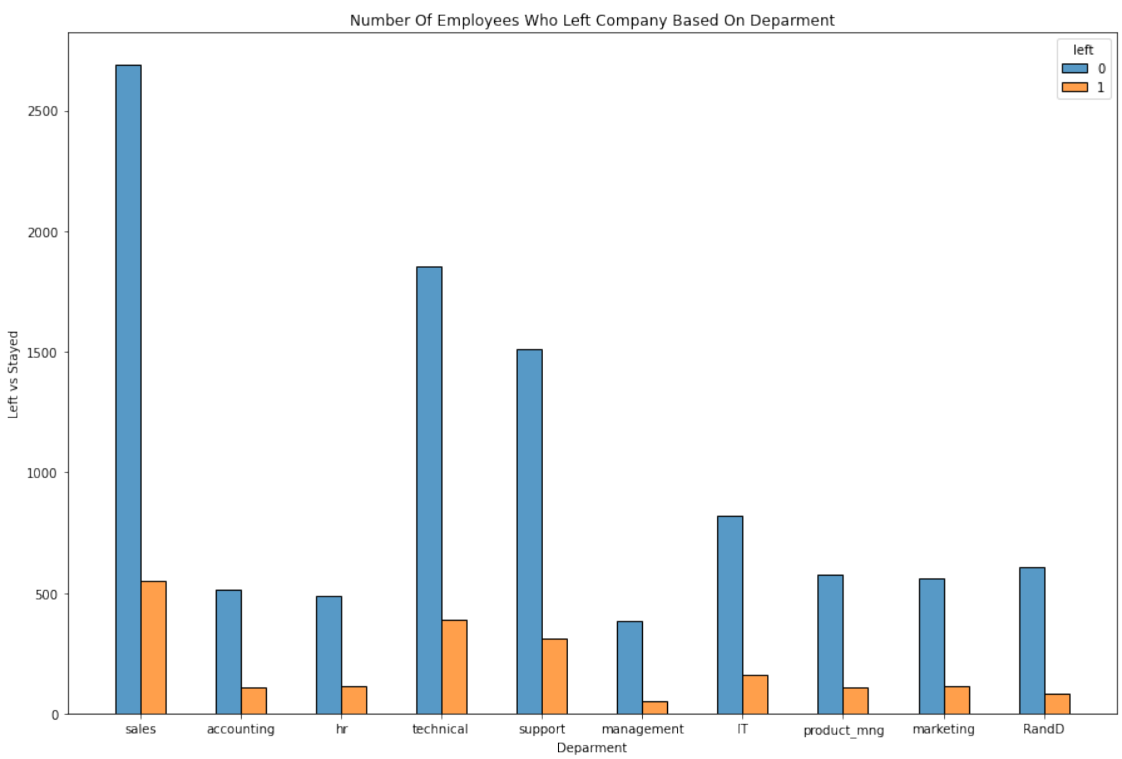



```
department
IT            3.797131
RandD         3.850144
accounting    3.834138
hr            3.675541
management    3.837156
```

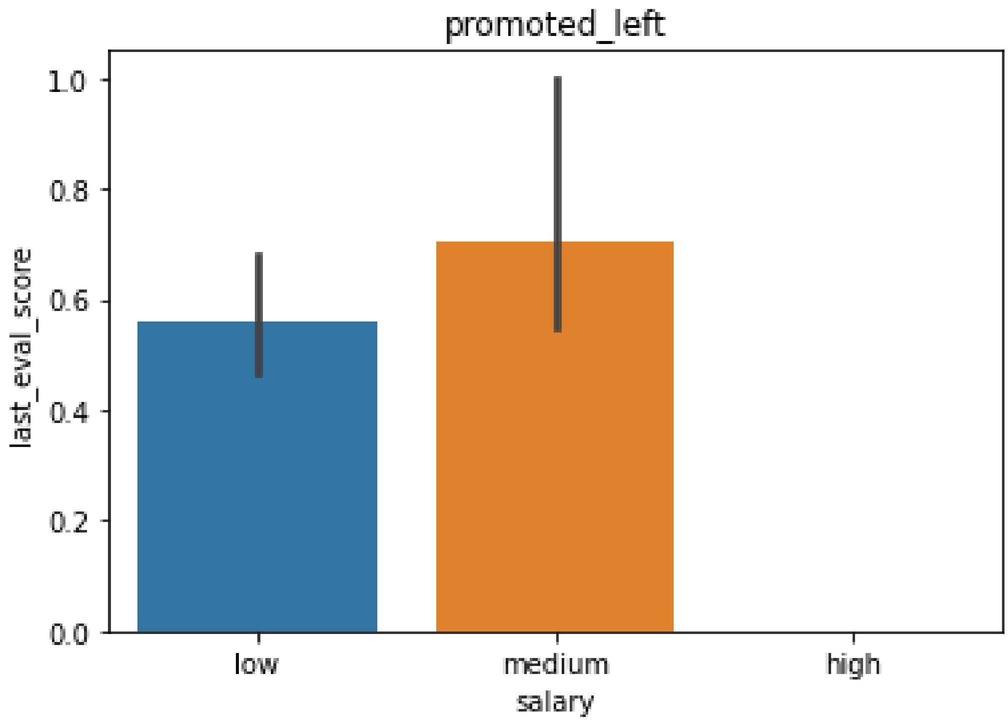
```
marketing      3.720654
product_mng    3.794461
sales          3.777092
support        3.820977
technical      3.859180
Name: projects_worked, dtype: float64
```

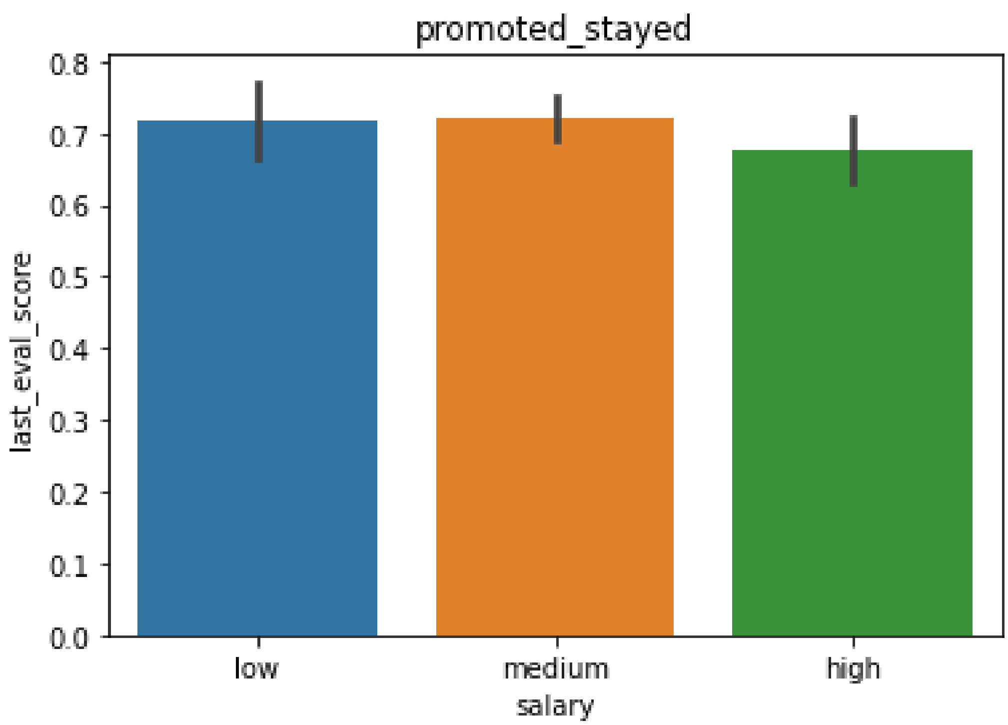



Compare Projects Worked Based On Satisfaction Level
Satisfaction Level
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2
3
4
5
6
7
Projects Worked

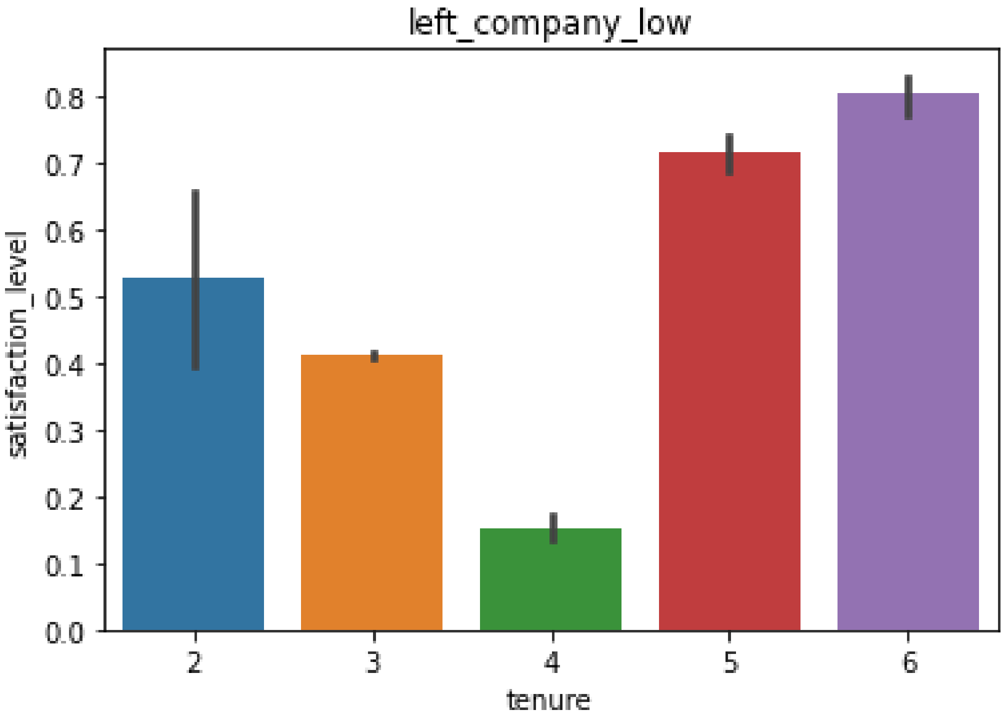

Number Of Employees Who Left Company Based On Deparment
left
0
1
Left vs Stayed
0
500
1000
1500
2000
2500
sales
accounting
hr
technical
support
management
IT
product_mng
marketing
RandD
Deparment

```
Investigating Average Department By Satisfaction Levels
department
IT             0.634016
RandD          0.627176
accounting     0.607939
hr             0.621947
management     0.631995
marketing      0.634770
product_mng    0.629825
sales          0.631349
support        0.634822
technical      0.627937
Name: satisfaction_level, dtype: float64
department
IT             0.715051
RandD          0.712983
accounting     0.721900
hr             0.715691
management     0.726307
marketing      0.718440
product_mng    0.713790
sales          0.710398
support        0.722998
technical      0.719791
Name: last_eval_score, dtype: float64
department
IT             200.638320
RandD          201.291066
accounting     200.877617
hr             199.371048
management     201.529817
marketing      199.487370
product_mng    198.893586
sales          200.242050
support        200.627128
technical      201.115419
Name: average_montly_hours, dtype: float64
INVESTIGATE EMPLOYEES WHO LEFT COMPANY
```

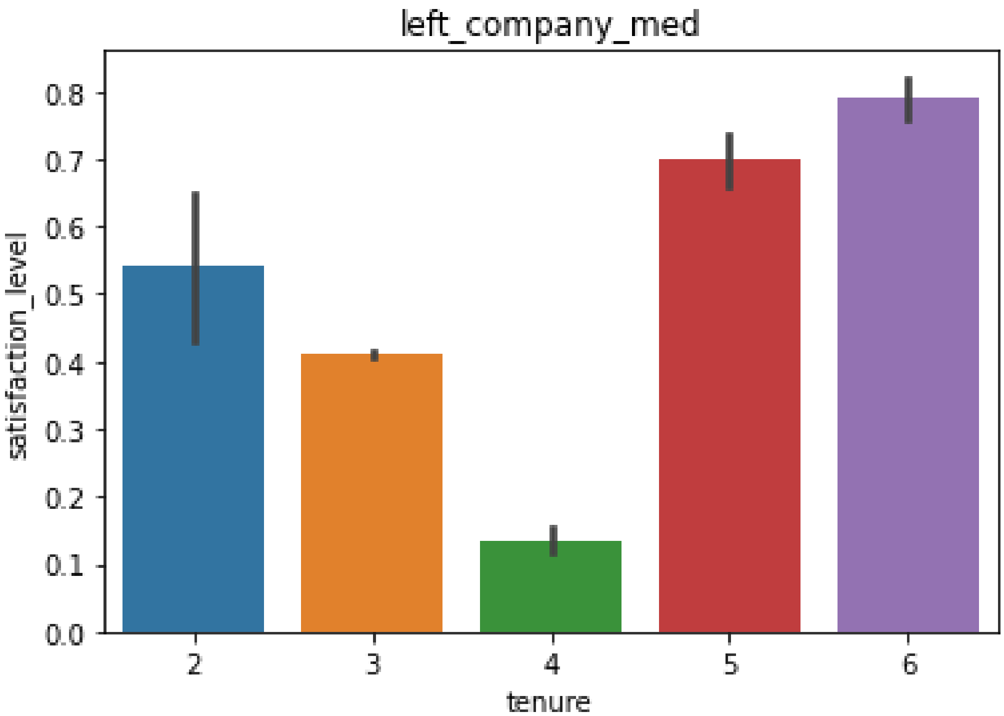

promoted_left
1.0
0.8
0.6
0.4
0.2
0.0
last_eval_score
low
medium
high
salary


promoted_stayed
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
last_eval_score
low
medium
high
salary

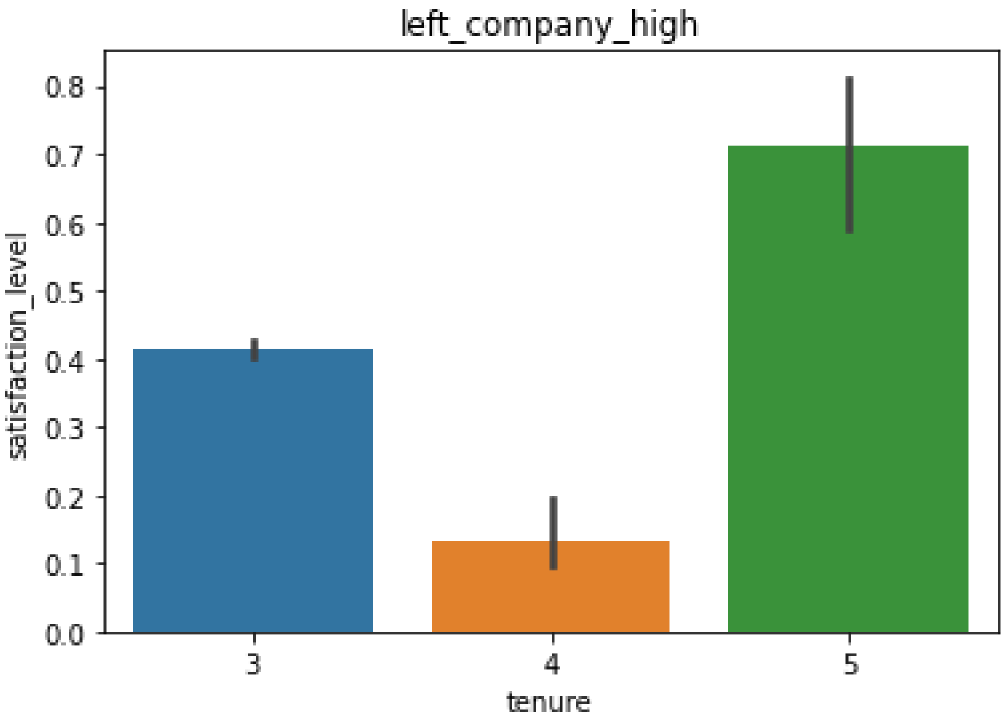
INVESTIGATE EMPLOYEES WHO LEFT COMPANY BASED ON SALARY




```
2     12
6     67
5    273
4    296
3    526
Name: tenure, dtype: int64
```




```
2     19
6     42
4    188
5    195
3    325
Name: tenure, dtype: int64
```




```
4    11
5    14
3    23
Name: tenure, dtype: int64
```

[67]:
```
# Column names and data for my refrence
df1.head(2)
```

[67]:
```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.38             0.53                2                   157
1                0.80             0.86                5                   262

   tenure  work_accident  left  promoted_5_years department  salary
0       3              0     1                 0      sales     low
1       6              0     1                 0      sales  medium
```

[150]:
```
## LEFT EMPLOYEES EDA ##
## FULL EDA NOTES AT BOTTOM OF THIS CELL ##

# Initial split and analysis of employees who left #

left_company = df1[df1["left"] == 1]
stayed_company = df1[df1["left"] == 0]
```

```python
print(left_company.shape)

print(left_company['department'].value_counts())

print(left_company['tenure'].value_counts())

print(left_company['promoted_5_years'].value_counts())

print(left_company['salary'].value_counts())

print(left_company['work_accident'].value_counts())

#############################################################################

# EDA of EMPLOYEES WHO LEFT who were PROMOTED

left_company_promoted = left_company[left_company["promoted_5_years"] == 1]

print(left_company_promoted['department'].value_counts())
print(left_company_promoted['tenure'].value_counts())

#############################################################################

# ANALYSIS OF MOST LEFT TENURE '3'

left_year_three = left_company[left_company["tenure"] == 3]

print(left_year_three['department'].value_counts())

###################################################################

# EDA OF LEFT WHO HAD WORK ACCIDENTS

accident_left = left_company[left_company["work_accident"] == 1]

print(accident_left['tenure'].value_counts())

#############################################################################

# EDA ON EVAL SCORE OF LEFT EMPLOYEES

eval_25_perc = left_company['last_eval_score'].quantile(0.25)
eval_75_perc = left_company['last_eval_score'].quantile(0.75)
eval_median = left_company['last_eval_score'].median()

print('EVAL 25th Percentile: ', eval_25_perc)
print('Eval Score Median', eval_median)
```

```
print('EVAL Percentile: ', eval_75_perc)

low_eval_thesh = left_company[left_company["last_eval_score"] <= eval_25_perc]
high_eval_thesh = left_company[left_company["last_eval_score"] >= eval_75_perc]

print(low_eval_thesh.shape)
print(low_eval_thesh['salary'].value_counts())
print(low_eval_thesh['tenure'].value_counts())
print(low_eval_thesh['projects_worked'].value_counts())
print()
print(high_eval_thesh.shape)
print(high_eval_thesh['salary'].value_counts())
print(high_eval_thesh['tenure'].value_counts())
print(high_eval_thesh['projects_worked'].value_counts())

print()
############################################################################

# EDA ON SATISFACTION LEVEL OF LEFT EMPLOYEES

satis_25_perc = left_company['satisfaction_level'].quantile(0.25)
satis_75_perc = left_company['satisfaction_level'].quantile(0.75)
satis_median = left_company['satisfaction_level'].median()


print('SATIS 25th Percentile: ', satis_25_perc)
print('SATIS Score Median', satis_median)
print('SATIS 75th Percentile: ', satis_75_perc)

low_satis_thesh = left_company[left_company["satisfaction_level"] <=␣
 ↪satis_25_perc]
high_satis_thesh = left_company[left_company["satisfaction_level"] >=␣
 ↪satis_75_perc]

print(low_satis_thesh.shape)
print(low_satis_thesh['salary'].value_counts())
print(low_satis_thesh['tenure'].value_counts())
print(low_satis_thesh['projects_worked'].value_counts())
print()
print(high_satis_thesh.shape)
print(high_satis_thesh['salary'].value_counts())
print(high_satis_thesh['tenure'].value_counts())
print(high_satis_thesh['projects_worked'].value_counts())


"""
MY NOTES
```

```
Checking first numeric breakdown there is a larger loss of employees in the big␣
↪3 fields this company hires for.
Those being: SALES, technical, support.
Followed by IT which is at 158, nearly half of support which is 3rd on most␣
↪left department


Based on the year most employees leave primarly in year 3 at 874 of 1991
Followed by year 4 at 495
Third is year 5 at 482
Years 4 & 5 are similar

Based on promotion, well there is hardly any for those who left. Out of 1991,␣
↪only 8 were promoted within 5 years.
The tenure of those promoted we're 6 in year 3, 1 in year 4 and 1 in year 5.

Being that year 3 was the largest year for leaving, I checked the departmemnt␣
↪breakdown for those.
This coorilated with my earlier note of the BIG 3 departments of SALES,␣
↪technical, support in that order.
This is again followed by IT at 71 with support at 3rd for most left at 128.


I then checked to see what the accidents at work breakdown was.
Out of 1991, there were 105 accidents that occured on those who left the␣
↪company.
Out of the 105 accidents, 46 of them occured in year 3, followed behind ay year␣
↪4 at 28 and 26 in year 5

I then broke down the salary of those who left the company.
Out of the 1991 who left, 1174 were in the LOW range for salary.
Out of the 1991 who left, 769 were in the MEDIUM range for salary.
Out of the 1991 who left, 48 were in the HIGH range for salary.

A pattern is begining to show for me:

Those of low salaries, who have not been promoted, and are in year 3-5 are the␣
↪most likley to leave.

Now I wanted to compare this in contrast to the employees eval score and␣
↪satisfaction levels.

To look at the eval scores and satisfaction levels I broke them to show the␣
↪25th and 75th quartiles and the median.
```

```
I then displayed the totals in those respective areas as well as broke them␣
↪down against tenure and salary.

EVAL SCORE:
EVAL 25th Percentile:  0.52
Eval Score Median 0.79
EVAL Percentile:  0.91


lower limit: 533 total
Salary: low  306
Tenure: 3    519
Project: 2    520

upper limit: 499 total
Salary: low  292
Tenure: 5    259
Project: 5    176

The latest eval scores ranged high for employees with the 25th percentile␣
↪hitting 0.52 out of 1.
Employees with lower eval scores tended to have low salaries and an␣
↪overwhelming majority left in year 3 at 519 out of 533.
Employees in the upper limit tended to have lower salaries and a majority left␣
↪in year 5.

Satisfaction level:
SATIS 25th Percentile:  0.11
SATIS Score Median 0.41
SATIS 75th Percentile:  0.73

lower limit: 501 total
Salary: low  292
Tenure: 4    443
Project: 6    339

upper limit: 504 total
Salary: low  292
Tenure: 5    377
Project: 5    285

The satisfaction levels lean more left towards unsatisfied with the lower␣
↪percentile being 0.11 out of 1.0.
The lower limit of SL's had a majority of low salaries and 443 of 501 left in␣
↪year 4.
```

```
The upper limit of SL's had a majority of low salaries but with a closer split␣
↪with medium. 292 low and 202 medium.
A majority of the upper limit left in year 5 at 377 out of 504.


#############################################################################################

FINAL STATEMENT AFTER THIS ANALYSIS:

I feel confident that there is a coorilation for employees leaving the company␣
↪and them:
1) Not being promoted within first 5 years, though this is quite low being as␣
↪most,
    if not all do not have
2) Having lower wages
3) Tenure is likley a big part on leaving employees. After, my initial guess, 3␣
↪years is when an employee
    is likley to leave.

However I need to build a regression model to analyze this information against
employees who have left the company.
I will random sample (upsample) those that left against those who stayed with␣
↪the company and draw
further conclusions about the data.

"""
```

```
(1991, 10)
sales          550
technical      390
support        312
IT             158
hr             113
marketing      112
product_mng    110
accounting     109
RandD           85
management      52
Name: department, dtype: int64
3    874
4    495
5    482
6    109
2     31
Name: tenure, dtype: int64
0    1983
1       8
```

```
Name: promoted_5_years, dtype: int64
sales         3
IT            2
management    1
technical     1
support       1
Name: department, dtype: int64
3    6
4    1
5    1
Name: tenure, dtype: int64
sales          253
technical      151
support        128
IT              71
hr              62
marketing       57
product_mng     51
accounting      46
RandD           31
management      24
Name: department, dtype: int64
0    1886
1     105
Name: work_accident, dtype: int64
3    46
4    28
5    26
6     4
2     1
Name: tenure, dtype: int64
low       1174
medium     769
high        48
Name: salary, dtype: int64
EVAL 25th Percentile:  0.52
Eval Score Median 0.79
EVAL Percentile:  0.91
(533, 10)
low       306
medium    213
high       14
Name: salary, dtype: int64
3    519
4      6
2      4
5      3
6      1
```

```
Name: tenure, dtype: int64
2    520
4      5
7      3
5      3
6      1
3      1
Name: projects_worked, dtype: int64

(499, 10)
low       292
medium    199
high        8
Name: salary, dtype: int64
5    259
4    163
6     63
3     10
2      4
Name: tenure, dtype: int64
5    176
4    134
6    122
7     52
3      8
2      7
Name: projects_worked, dtype: int64

SATIS 25th Percentile:  0.11
SATIS Score Median 0.41
SATIS 75th Percentile:  0.73
(501, 10)
low       292
medium    199
high       10
Name: salary, dtype: int64
4    443
5     54
3      3
2      1
Name: tenure, dtype: int64
6    339
7    135
5     22
2      2
3      2
4      1
Name: projects_worked, dtype: int64
```

```
(504, 10)
low       292
medium    202
high       10
Name: salary, dtype: int64
5    377
6    100
4     10
3      9
2      8
Name: tenure, dtype: int64
5    285
4    196
3     13
6      6
2      4
Name: projects_worked, dtype: int64
```

[150]:
```
"\nMY NOTES\n\nChecking first numeric breakdown there is a larger loss of
employees in the big 3 fields this company hires for.\nThose being: SALES,
technical, support. \nFollowed by IT which is at 158, nearly half of support
which is 3rd on most left department\n\n\nBased on the year most employees leave
primarily in year 3 at 874 of 1991\nFollowed by year 4 at 495\nThird is year 5 at
482 \nYears 4 & 5 are similar\n\nBased on promotion, well there is hardly any
for those who left. Out of 1991, only 8 were promoted within 5 years.\nThe
tenure of those promoted we're 6 in year 3, 1 in year 4 and 1 in year
5.\n\nBeing that year 3 was the largest year for leaving, I checked the
departmemnt breakdown for those. \nThis coorilated with my earlier note of the
BIG 3 departments of SALES, technical, support in that order.\nThis is again
followed by IT at 71 with support at 3rd for most left at 128.\n\n\nI then
checked to see what the accidents at work breakdown was. \nOut of 1991, there
were 105 accidents that occured on those who left the company. \nOut of the 105
accidents, 46 of them occured in year 3, followed behind ay year 4 at 28 and 26
in year 5\n\nI then broke down the salary of those who left the company. \nOut
of the 1991 who left, 1174 were in the LOW range for salary.\nOut of the 1991
who left, 769 were in the MEDIUM range for salary.\nOut of the 1991 who left, 48
were in the HIGH range for salary.\n\nA pattern is begining to show for
me:\n\nThose of low salaries, who have not been promoted, and are in year 3-5
are the most likley to leave. \n\nNow I wanted to compare this in contrast to
the employees eval score and satisfaction levels. \n\nTo look at the eval scores
and satisfaction levels I broke them to show the 25th and 75th quartiles and the
median.\nI then displayed the totals in those respective areas as well as broke
them down against tenure and salary.\n\nEVAL SCORE:\nEVAL 25th Percentile:
0.52\nEval Score Median 0.79\nEVAL Percentile:  0.91\n\n\nlower limit: 533
total\nSalary: low  306\nTenure: 3    519\nProject: 2    520\n\nupper limit: 499
total\nSalary: low  292\nTenure: 5    259\nProject: 5    176\n\nThe latest eval
```

```
scores ranged high for employees with the 25th percentile hitting 0.52 out of
1.\nEmployees with lower eval scores tended to have low salaries and an
overwhelming majority left in year 3 at 519 out of 533.\nEmployees in the upper
limit tended to have lower salaries and a majority left in year
5.\n\nSatisfaction level: \nSATIS 25th Percentile:  0.11\nSATIS Score Median
0.41\nSATIS 75th Percentile:  0.73\n\nlower limit: 501 total\nSalary: low
292\nTenure: 4    443\nProject: 6    339\n\nupper limit: 504 total\nSalary: low
292\nTenure: 5    377\nProject: 5    285\n\nThe satisfaction levels lean more
left towards unsatisfied with the lower percentile being 0.11 out of 1.0.\nThe
lower limit of SL's had a majority of low salaries and 443 of 501 left in year
4.\nThe upper limit of SL's had a majority of low salaries but with a closer
split with medium. 292 low and 202 medium.\nA majority of the upper limit left
in year 5 at 377 out of 504.\n\n\n#################################################
##################################################\n\nFINAL STATEMENT AFTER THIS
ANALYSIS:\n\nI feel confident that there is a coorilation for employees leaving
the company and them:\n1) Not being promoted within first 5 years\n2) Having
lower wages\n\nHowever I need to build a model to analyze this information
against employees who have left the company, and if possible\nautomate this. I
will random sample (upsample) those that left against those who stayed with the
company and draw further\nconclusions about the data. Especially that involving
the eval scores and the satisfaction levels. \n\n"
```

[69]:
```
# UPSERT RESAMPLING

df1.head(2)
```

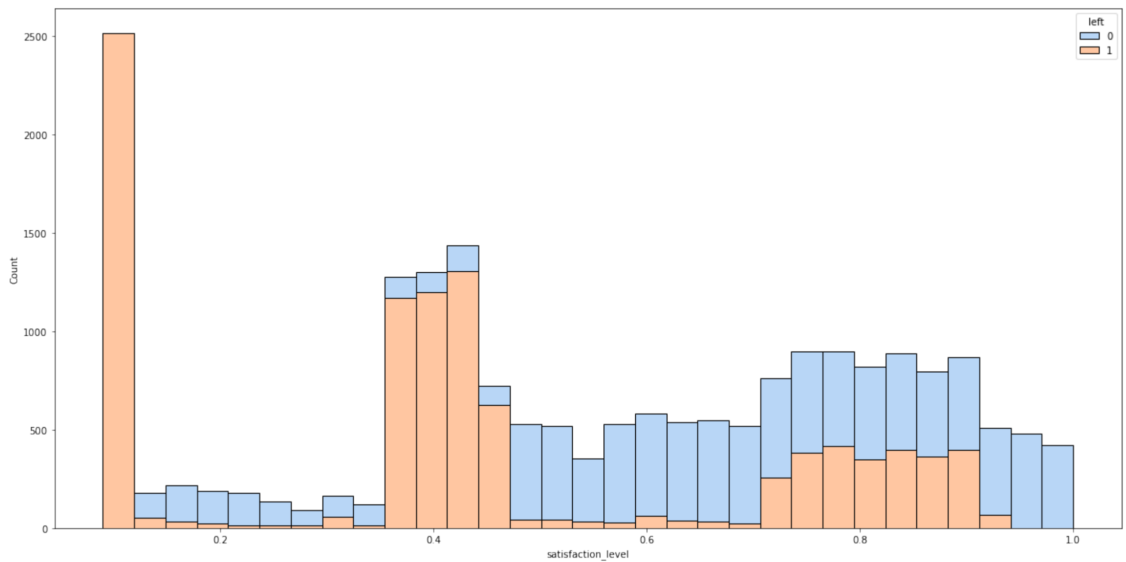
[69]:
```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.38             0.53                2                   157
1                0.80             0.86                5                   262

   tenure  work_accident  left  promoted_5_years department  salary
0       3              0     1                 0      sales     low
1       6              0     1                 0      sales  medium
```

[70]:
```
## START OF REGRESSION ANALYSIS
# Upsampling used to balence the number of left vs stayed

# Create a plot as needed
data_majority = df1[df1['left'] == 0]
data_minority = df1[df1['left'] == 1]

data_minority_upsampled = resample(data_minority, replace=True,␣
↪n_samples=len(data_majority), random_state = 42)

data_upsampled = pd.concat([data_majority, data_minority_upsampled]).
↪reset_index(drop=True)
```

```
data_upsampled['left'].value_counts()
```

[70]:
```
0    10000
1    10000
Name: left, dtype: int64
```

[71]:
```
# Create a plot as needed
plt.figure(figsize=(20,10))
sns.histplot(data=data_upsampled, stat="count", multiple="stack",
             x='satisfaction_level', kde=False, palette= 'pastel', hue='left',␣
 ↪element='bars', legend= True)
plt.show()
```



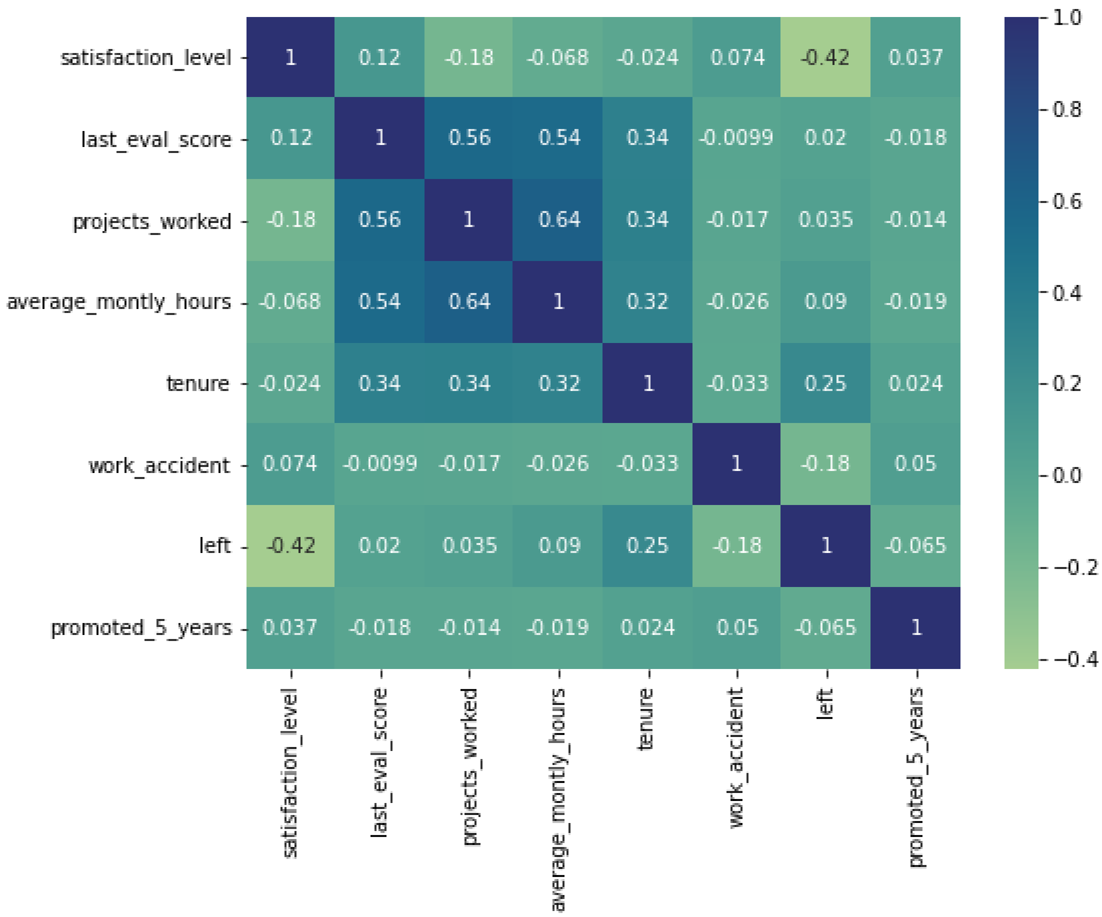
[72]:
```
# Create a plot as needed

data_upsampled.corr()
```

[72]:
```
                      satisfaction_level  last_eval_score  projects_worked  \
satisfaction_level              1.000000         0.122169        -0.180346
last_eval_score                 0.122169         1.000000         0.559230
projects_worked                -0.180346         0.559230         1.000000
average_montly_hours           -0.068442         0.542638         0.638351
tenure                         -0.024128         0.335608         0.343715
work_accident                   0.074384        -0.009893        -0.016896
left                           -0.423792         0.020233         0.035046
promoted_5_years                0.036549        -0.017511        -0.014039

                      average_montly_hours    tenure  work_accident      left  \
```

```
satisfaction_level            -0.068442 -0.024128       0.074384 -0.423792
last_eval_score                0.542638  0.335608      -0.009893  0.020233
projects_worked                0.638351  0.343715      -0.016896  0.035046
average_montly_hours           1.000000  0.318809      -0.026028  0.089985
tenure                         0.318809  1.000000      -0.033353  0.253539
work_accident                 -0.026028 -0.033353       1.000000 -0.181543
left                           0.089985  0.253539      -0.181543  1.000000
promoted_5_years              -0.018590  0.024364       0.050078 -0.065323

                      promoted_5_years
satisfaction_level            0.036549
last_eval_score              -0.017511
projects_worked              -0.014039
average_montly_hours         -0.018590
tenure                        0.024364
work_accident                 0.050078
left                         -0.065323
promoted_5_years              1.000000
```

[73]:

```python
# Create a plot as needed
plt.figure(figsize=(8,6))

sns.heatmap(data_upsampled[['satisfaction_level', 'last_eval_score',␣
↪'projects_worked', 'average_montly_hours',
                           'tenure', 'work_accident', 'left',␣
↪'promoted_5_years']].corr(), annot=True,
           cmap='crest')

plt.show()
```



[74]:
```
## Because of assumed model imbalence I am removing outliars based on Tenure as␣
↪seen from boxplot from above.

percentile25 = df1["tenure"].quantile(0.25)
percentile75 = df1["tenure"].quantile(0.75)
iqr = percentile75 - percentile25
upper_limit = percentile75 + 1.5 * iqr
print(upper_limit)
df2 = df1[df1['tenure'] <=upper_limit]
df2.head(2)
```

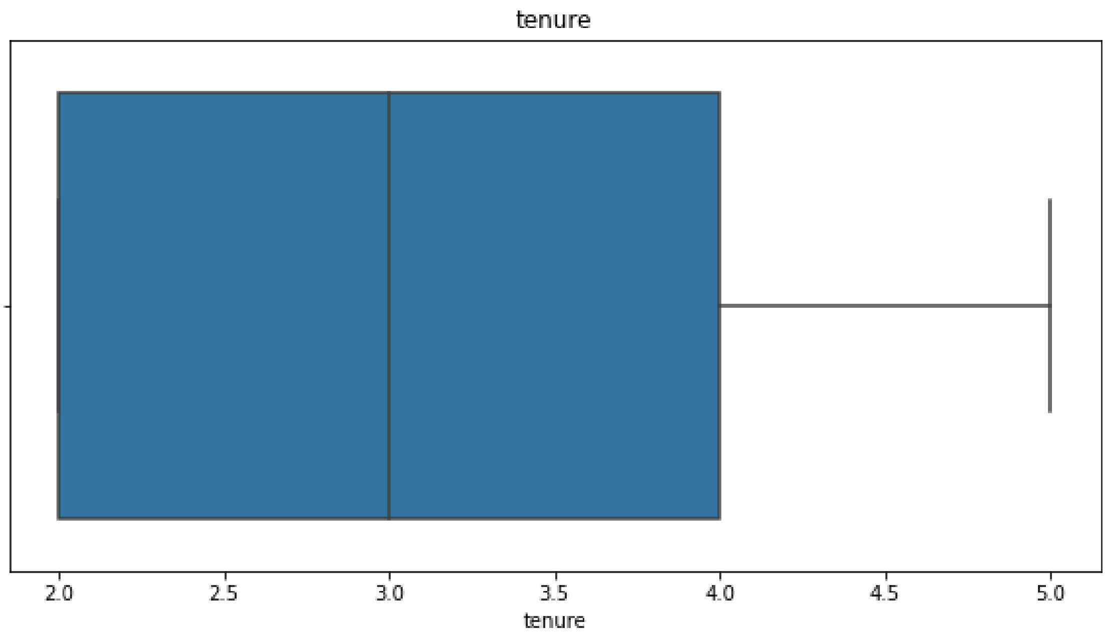
```
5.5
```

[74]:
```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.38             0.53                2                   157
2                0.11             0.88                7                   272

   tenure  work_accident  left  promoted_5_years department  salary
```

```
0       3              0     1                0      sales      low
2       4              0     1                0      sales   medium
```

[75]:
```
plt.figure(figsize=(10,5))
plt.title('tenure')
sns.boxplot(x=df2['tenure'])
```

[75]: <matplotlib.axes._subplots.AxesSubplot at 0x7f470434c5d0>

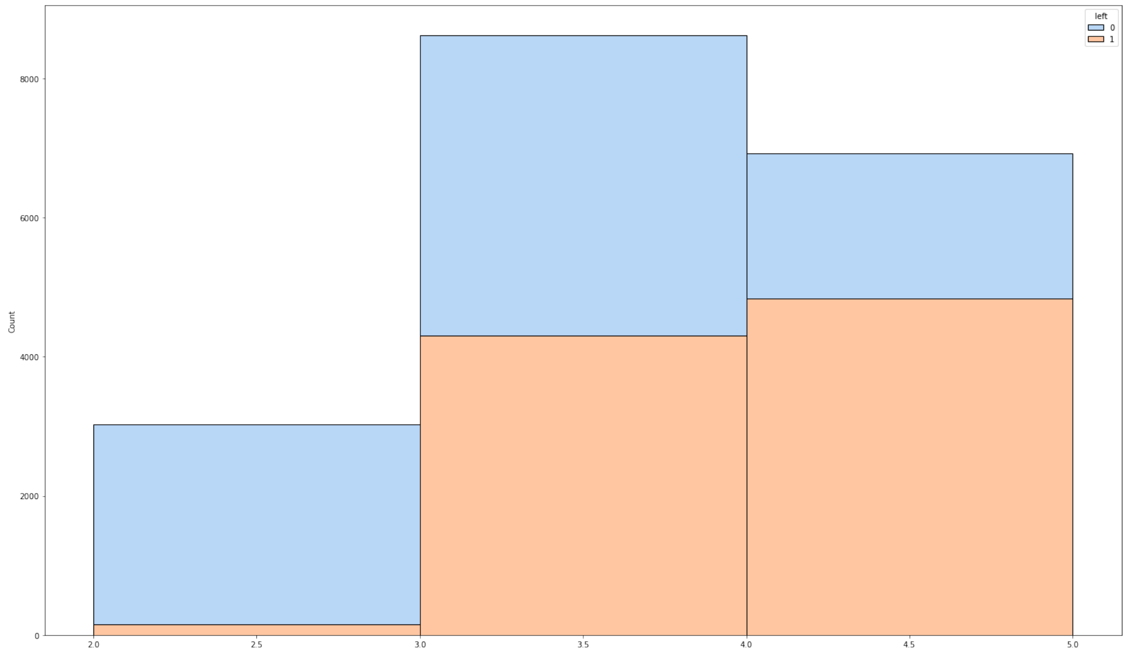


[76]:
```
data_majority = df2[df2['left'] == 0]
data_minority = df2[df2['left'] == 1]

data_minority_upsampled = resample(data_minority, replace=True,␣
 ↪n_samples=len(data_majority), random_state = 42)

data_upsampled = pd.concat([data_majority, data_minority_upsampled]).
 ↪reset_index(drop=True)

data_upsampled['left'].value_counts()
```

[76]:
```
0    9285
1    9285
Name: left, dtype: int64
```

[156]:
```
plt.figure(figsize=(25,15))
```

```
arr = np.array([2,3,4,5])

sns.histplot(data=data_upsampled, bins= arr, stat="count", multiple="stack",
            x='tenure', kde=False, palette= 'pastel', hue='left',␣
 ↪element='bars', legend= True)
plt.show()

"""
Tenure show's normal distrobution.
"""
```



[156]: "\nTenure show's normal distrobution.\n"

[78]:
```
data_upsampled.corr()
```

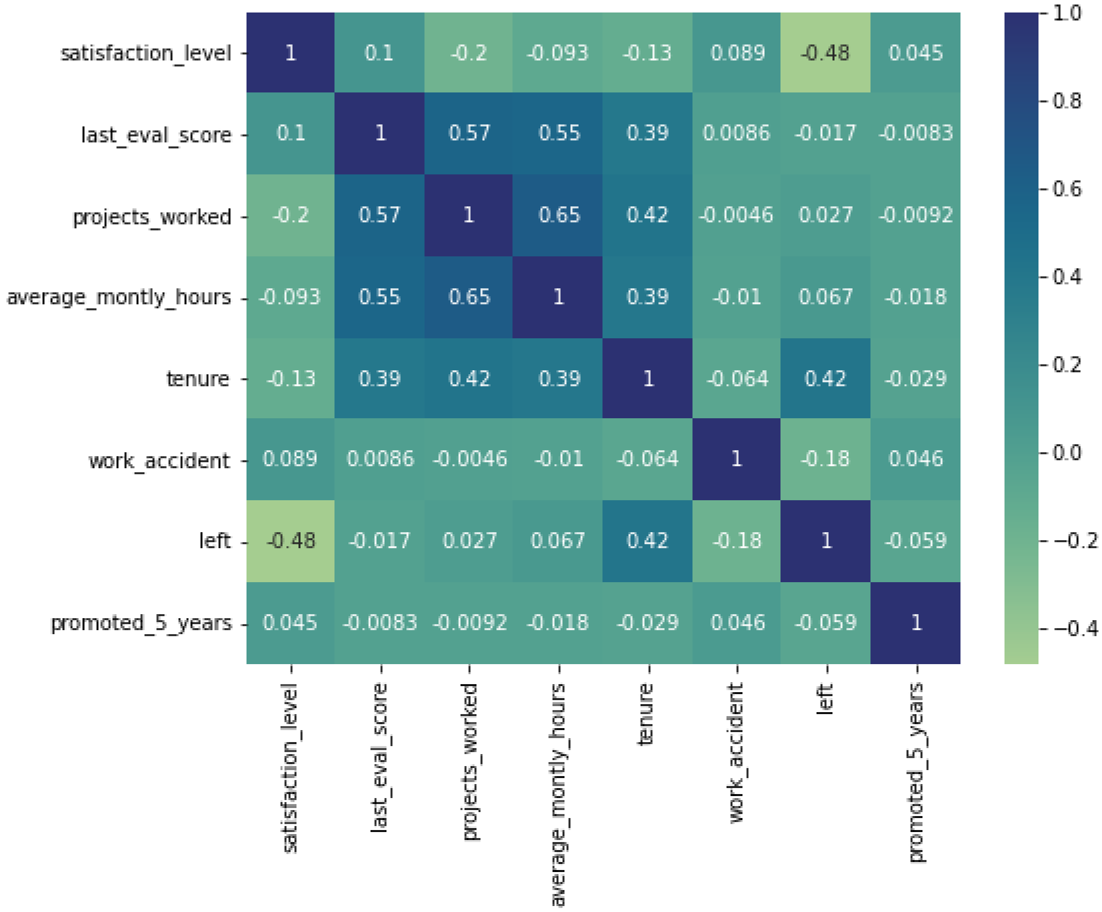
[78]:

	satisfaction_level	last_eval_score	projects_worked	\
satisfaction_level	1.000000	0.101346	-0.202478	
last_eval_score	0.101346	1.000000	0.568921	
projects_worked	-0.202478	0.568921	1.000000	
average_montly_hours	-0.092844	0.550764	0.651305	
tenure	-0.133904	0.387227	0.421993	
work_accident	0.089481	0.008561	-0.004558	
left	-0.481275	-0.016820	0.026651	
promoted_5_years	0.044674	-0.008346	-0.009188	

```
                       average_montly_hours    tenure  work_accident      left  \
satisfaction_level                -0.092844 -0.133904       0.089481 -0.481275
last_eval_score                    0.550764  0.387227       0.008561 -0.016820
projects_worked                    0.651305  0.421993      -0.004558  0.026651
average_montly_hours               1.000000  0.393953      -0.010384  0.066551
tenure                             0.393953  1.000000      -0.064163  0.418186
work_accident                     -0.010384 -0.064163       1.000000 -0.184880
left                               0.066551  0.418186      -0.184880  1.000000
promoted_5_years                  -0.018209 -0.028778       0.046295 -0.058640

                      promoted_5_years
satisfaction_level            0.044674
last_eval_score              -0.008346
projects_worked              -0.009188
average_montly_hours         -0.018209
tenure                       -0.028778
work_accident                 0.046295
left                         -0.058640
promoted_5_years              1.000000
```

[79]:

```python
plt.figure(figsize=(8,6))

sns.heatmap(data_upsampled[['satisfaction_level', 'last_eval_score',␣
 ↪'projects_worked', 'average_montly_hours',
                            'tenure', 'work_accident', 'left',␣
 ↪'promoted_5_years']].corr(), annot=True,
           cmap='crest')

plt.show()
```



[80]:
```
## Time for train test

# Select outcome variable
y = data_upsampled["left"]

# Select features

X = data_upsampled[['satisfaction_level', 'last_eval_score', 'projects_worked',␣
↪'average_montly_hours',
                              'tenure', 'work_accident', 'promoted_5_years',␣
↪'salary']]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25,␣
↪random_state=42)
```

[81]:
```
# Verify shapes of data
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[81]: ((13927, 8), (4643, 8), (13927,), (4643,))

[82]:
```
print(X_train['salary'].unique())
print(X_train['promoted_5_years'].unique())
```

```
['high' 'low' 'medium']
[0 1]
```

[83]:
```
# Select the training features that needs to be encoded
X_train_to_encode = X_train[['promoted_5_years', 'salary']]

# Display first few rows
X_train_to_encode.head()
```

[83]:

	promoted_5_years	salary
4397	0	high
12630	0	low
554	0	low
5465	0	low
345	0	low

[84]:
```
# Set up an encoder for one-hot encoding the categorical features
X_encoder = OneHotEncoder(drop='first', sparse=False)

# Fit and transform the training features using the encoder
X_train_encoded = X_encoder.fit_transform(X_train_to_encode)

# Display first few rows of encoded training features
X_train_encoded
```

[84]:
```
array([[0., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       …,
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.]])
```

[85]:
```
# Place encoded training features (which is currently an array) into a dataframe
X_train_encoded_df = pd.DataFrame(data=X_train_encoded, columns=X_encoder.
 ↪get_feature_names())

# RENAME COLUMNS TO REDUCE CONFUSION
X_train_encoded_df = X_train_encoded_df.rename(columns={'x0_1':␣
 ↪'promoted_is_1'})
```

```
#view dataframe
X_train_encoded_df.head()
```

[85]:

```
   promoted_is_1  x1_low  x1_medium
0            0.0     0.0        0.0
1            0.0     1.0        0.0
2            0.0     1.0        0.0
3            0.0     1.0        0.0
4            0.0     1.0        0.0
```

[86]:

```
#Drop the old left, salary, promoted, and department columns and concat the new␣
↪versions

X_train_final = pd.concat([X_train.drop(columns=['promoted_5_years','salary'])
                            .reset_index(drop=True), X_train_encoded_df], axis=1)



# Display first few rows
X_train_final.head()
```

[86]:

```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.64             0.64                3                   234
1                0.42             0.57                2                   159
2                0.53             0.88                3                   157
3                0.80             0.87                4                   209
4                0.65             0.60                5                   227

   tenure  work_accident  promoted_is_1  x1_low  x1_medium
0       3              1            0.0     0.0        0.0
1       3              0            0.0     1.0        0.0
2       3              0            0.0     1.0        0.0
3       3              0            0.0     1.0        0.0
4       3              0            0.0     1.0        0.0
```

[87]:

```
# Get unique values of outcome variable
y_train.unique()
```

[87]:

```
array([0, 1])
```

[88]:

```
# Set up an encoder for one-hot encoding the categorical outcome variable
y_encoder = OneHotEncoder(drop='first', sparse=False)
```

[89]:

```
# Encode the training outcome variable
y_train_final = y_encoder.fit_transform(y_train.values.reshape(-1, 1)).ravel()

y_train_final
```

[89]: array([0., 1., 0., …, 0., 0., 1.])

[90]:
```
# Construct a logistic regression model and fit it to the training set
log_clf = LogisticRegression(random_state=0, max_iter=800).fit(X_train_final,␣
↪y_train)
```

[91]:
```
# Select the testing features that needs to be encoded
X_test_to_encode = X_test[["promoted_5_years","salary"]]
X_test_to_encode.head()
```

[91]:

	promoted_5_years	salary
14810	0	medium
13698	0	medium
2026	0	medium
12004	0	high
18342	0	medium

[92]:
```
# Transform the testing features using the encoder
X_test_encoded = X_encoder.fit_transform(X_test_to_encode)

# Display first few rows of encoded testing features
X_test_encoded
```

[92]:
```
array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       …,
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]])
```

[93]:
```
# Place encoded testing features (which is currently an array) into a dataframe
X_test_encoded_df = pd.DataFrame(data=X_test_encoded, columns=X_encoder.
↪get_feature_names())

# RENAME COLUMNS TO REDUCE CONFUSION
X_test_encoded_df = X_test_encoded_df.rename(columns={'x0_1': 'promoted_is_1'})

# Display first few rows
X_test_encoded_df.head()
```

[93]:

	promoted_is_1	x1_low	x1_medium
0	0.0	0.0	1.0
1	0.0	0.0	1.0
2	0.0	0.0	1.0
3	0.0	0.0	0.0
4	0.0	0.0	1.0

[94]:
```
#Drop the old left, salary, promoted, and department columns and concat the new␣
↪versions
X_test_final = pd.concat([X_test.drop(columns=["promoted_5_years","salary"]).
↪reset_index(drop=True), X_test_encoded_df], axis=1)
# Display first few rows
X_test_final.head()
```

[94]:
```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.40             0.46                2                   156
1                0.44             0.52                2                   137
2                0.84             0.85                4                   185
3                0.38             0.50                2                   152
4                0.40             0.46                2                   127

   tenure  work_accident  promoted_is_1  x1_low  x1_medium
0       3              0            0.0     0.0        1.0
1       3              0            0.0     0.0        1.0
2       3              1            0.0     0.0        1.0
3       3              0            0.0     0.0        0.0
4       3              0            0.0     0.0        1.0
```

[95]:
```
# Use the logistic regression model to get predictions on the encoded testing␣
↪set

y_pred = log_clf.predict(X_test_final)

# Display the predictions on the encoded testing set
y_pred
```

[95]:
```
array([1, 1, 0, …, 1, 0, 1])
```

[96]:
```
# Display the true labels of the testing set
y_test
```

[96]:
```
14810    1
13698    1
2026     0
12004    1
18342    1
        ..
13110    1
5748     0
15789    1
5182     0
199      0
Name: left, Length: 4643, dtype: int64
```

[97]:
```
#Encode the testing outcome variable

y_test_final = y_encoder.transform(y_test.values.reshape(-1, 1)).ravel()

y_test_final
```

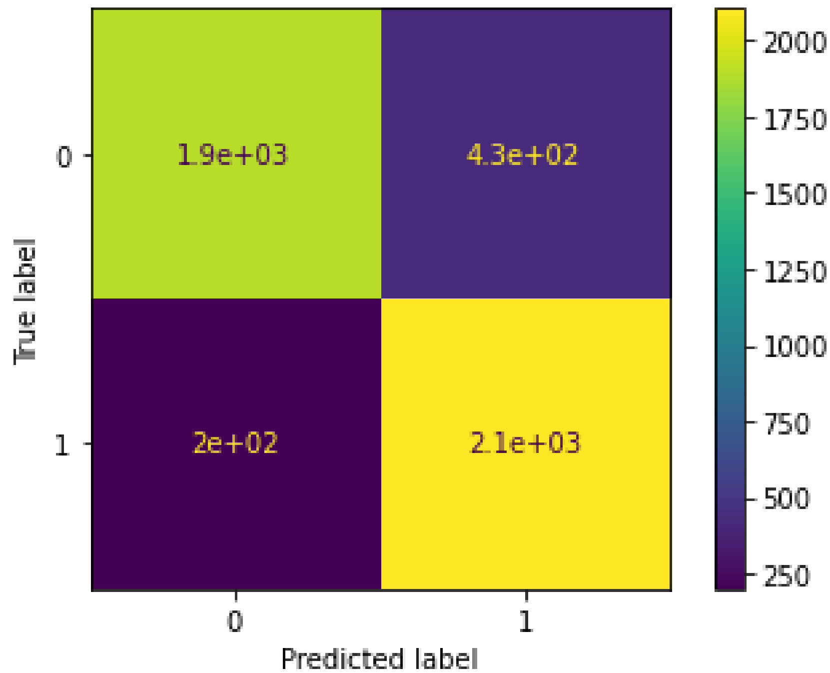
[97]: array([1., 1., 0., …, 1., 0., 0.])

[98]:
```
X_train_final.shape, y_train_final.shape, X_test_final.shape, y_test_final.shape
```

[98]: ((13927, 9), (13927,), (4643, 9), (4643,))

[99]:
```
# Compute values for confusion matrix

log_cm = confusion_matrix(y_test_final, 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()

# Display plot
plt.show()

"""
* The upper-left quadrant displays the number of true negatives: the number of␣
↪employees who
stayed that the model accurately classified as so.

The upper-right quadrant displays the number of false positives:  the number of␣
↪employees who
stayed that the  the model misclassified as left.

The lower-left quadrant displays the number of false negatives: the number of␣
↪employees who
left that the  the model misclassified as stayed.

* The lower-right quadrant displays the number of true positives: the number of␣
↪employees who
left that the  the model accurately classified as so.

A perfect model would yield all true negatives and true positives, and no false␣
↪negatives or false
positives
```

```
"""
```



[99]: '\n* The upper-left quadrant displays the number of true negatives: the number of employees who\nstayed that the model accurately classified as so.\n\nThe upper-right quadrant displays the number of false positives: the number of employees who\nstayed that the the model misclassified as left.\n\nThe lower-left quadrant displays the number of false negatives: the number of employees who\nleft that the the model misclassified as stayed.\n\n* The lower-right quadrant displays the number of true positives: the number of employees who\nleft that the the model accurately classified as so.\n\nA perfect model would yield all true negatives and true positives, and no false negatives or false\npositives\n\n'

[100]:
```
# Create classification report for logistic regression model
target_labels = ["Left", "Stayed"]
print(classification_report(y_test_final, y_pred, target_names=target_labels))

"""
The classification report above shows that the logistic regression model␣
↪achieved
a precision of 90% and a recall of 81%, and it achieved an accuracy of 86%.␣
↪Note that the precision
```

```
and recall scores are taken from the "left" row of the output because that is␣
 ↪the target class
that we are most interested in predicting. The "Stayed" class has its own␣
 ↪precision/recall metrics,
and the weighted average represents the combined metrics for both classes of␣
 ↪the target variable.

TERMS TO KNOW:

Accuracy: The proportion of data points that were correctly categorized
Recall: The proportion of actual positives that were identified correctly to␣
 ↪all actual positives
Precision: The proportion of positive predictions that were correct to all␣
 ↪positive predictions
F1-score: The harmonic mean of precision and recall

"""
```

```
              precision    recall  f1-score   support

        Left       0.90      0.81      0.86      2337
      Stayed       0.83      0.91      0.87      2306

    accuracy                           0.86      4643
   macro avg       0.87      0.86      0.86      4643
weighted avg       0.87      0.86      0.86      4643
```

[100]:
```
'\nThe classification report above shows that the logistic regression model
achieved\na precision of 90% and a recall of 81%, and it achieved an accuracy of
86%. Note that the precision\nand recall scores are taken from the "left" row of
the output because that is the target class\nthat we are most interested in
predicting. The "Stayed" class has its own precision/recall metrics,\nand the
weighted average represents the combined metrics for both classes of the target
variable.\n\nTERMS TO KNOW:\n\nAccuracy: The proportion of data points that were
correctly categorized\t\nRecall: The proportion of actual positives that were
identified correctly to all actual positives\nPrecision: The proportion of
positive predictions that were correct to all positive predictions\nF1-score:
The harmonic mean of precision and recall\n\n'
```

[ ]:
```

```

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.

[104]:
```
## MODELING TIME!!
```

```
# Select outcome variable
y = data_upsampled["left"]

# Select features

X = data_upsampled[['satisfaction_level', 'last_eval_score', 'projects_worked',␣
↪'average_montly_hours',
                           'tenure', 'work_accident', 'promoted_5_years',␣
↪'salary']]

# Split data into training and testing sets, 80/20.
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[106]:
```
# Split the training set into training and validation sets,
# 75/25, to result in a final ratio of 60/20/20 for train/validate/test sets.

# Split the training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=0.25,␣
↪random_state=0)

# Get shape of each training, validation, and testing set
X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.
↪shape
```

[106]:
```
((11142, 8), (3714, 8), (3714, 8), (11142,), (3714,), (3714,))
```

[110]:
```
## X_TRAIN ENCODING ##

# Select the training features that needs to be encoded
X_train_to_encode = X_train[['promoted_5_years', 'salary']]

# Set up an encoder for one-hot encoding the categorical features
X_encoder = OneHotEncoder(drop='first', sparse=False)

# Fit and transform the training features using the encoder
X_train_encoded = X_encoder.fit_transform(X_train_to_encode)


# Place encoded training features (which is currently an array) into a dataframe
X_train_encoded_df = pd.DataFrame(data=X_train_encoded, columns=X_encoder.
↪get_feature_names())

# RENAME COLUMNS TO REDUCE CONFUSION
X_train_encoded_df = X_train_encoded_df.rename(columns={'x0_1':␣
↪'promoted_is_1'})
```

```
#Drop the old left, salary, promoted, and department columns and concat the new␣
↪versions

X_train_final = pd.concat([X_train.drop(columns=['promoted_5_years','salary'])
                           .reset_index(drop=True), X_train_encoded_df], axis=1)


# Display first few rows
X_train_final.head()
```

[110]:

```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.10             0.79                6                   294
1                0.61             0.46                5                   220
2                0.60             0.80                4                   146
3                0.42             0.97                6                   259
4                0.51             0.48                5                   136

   tenure  work_accident  promoted_is_1  x1_low  x1_medium
0       4              0            0.0     0.0        1.0
1       4              0            0.0     1.0        0.0
2       2              0            0.0     0.0        1.0
3       4              0            0.0     0.0        0.0
4       4              0            0.0     1.0        0.0
```

[111]:

```
## Y_TRAIN ENCODING ##

# Set up an encoder for one-hot encoding the categorical outcome variable
y_encoder = OneHotEncoder(drop='first', sparse=False)

# Encode the training outcome variable
y_train_final = y_encoder.fit_transform(y_train.values.reshape(-1, 1)).ravel()

# Construct a logistic regression model and fit it to the training set
log_clf = LogisticRegression(random_state=0, max_iter=800).fit(X_train_final,␣
↪y_train)
X_test_to_encode.head()
```

[115]:

```
## X_TEST ENCODING ##

# Select the testing features that needs to be encoded
X_test_to_encode = X_test[["promoted_5_years","salary"]]
X_test_to_encode.head()

# Transform the testing features using the encoder
X_test_encoded = X_encoder.fit_transform(X_test_to_encode)
```

```
# Display first few rows of encoded testing features
X_test_encoded

# Place encoded testing features (which is currently an array) into a dataframe
X_test_encoded_df = pd.DataFrame(data=X_test_encoded, columns=X_encoder.
 ↪get_feature_names())

# RENAME COLUMNS TO REDUCE CONFUSION
X_test_encoded_df = X_test_encoded_df.rename(columns={'x0_1': 'promoted_is_1'})

# Display first few rows
X_test_encoded_df.head()

#Drop the old left, salary, promoted, and department columns and concat the new␣
 ↪versions
X_test_final = pd.concat([X_test.drop(columns=["promoted_5_years","salary"])
                          .reset_index(drop=True), X_test_encoded_df], axis=1)

# Display first few rows
X_test_final.head()
```

[115]:

```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.09             0.83                6                   295
1                0.40             0.47                2                   146
2                0.87             0.91                4                   228
3                0.71             0.74                3                   250
4                0.72             0.53                5                   240

   tenure  work_accident  promoted_is_1  x1_low  x1_medium
0       5              0            0.0     1.0        0.0
1       3              0            0.0     0.0        1.0
2       5              0            0.0     1.0        0.0
3       3              0            0.0     1.0        0.0
4       2              0            0.0     0.0        1.0
```

[117]:

```
## X_VAL ENCODING

# Select the training features that needs to be encoded
X_val_to_encode = X_val[['promoted_5_years', 'salary']]

# Set up an encoder for one-hot encoding the categorical features
X_val_encoder = OneHotEncoder(drop='first', sparse=False)

# Fit and transform the training features using the encoder
X_val_encoded = X_val_encoder.fit_transform(X_val_to_encode)
```

```
# Place encoded training features (which is currently an array) into a dataframe
X_val_encoded_df = pd.DataFrame(data=X_val_encoded, columns=X_encoder.
 ↪get_feature_names())

# RENAME COLUMNS TO REDUCE CONFUSION
X_val_encoded_df = X_val_encoded_df.rename(columns={'x0_1': 'promoted_is_1'})


#Drop the old left, salary, promoted, and department columns and concat the new␣
 ↪versions

X_val_final = pd.concat([X_val.drop(columns=['promoted_5_years','salary'])
                            .reset_index(drop=True), X_val_encoded_df], axis=1)


# Display first few rows
X_val_final.head()
```

[117]:

```
   satisfaction_level  last_eval_score  projects_worked  average_montly_hours  \
0                0.96             0.68                4                   137
1                0.64             0.90                6                   252
2                0.75             0.83                5                   262
3                0.51             0.69                3                   145
4                0.57             0.70                3                   172

   tenure  work_accident  promoted_is_1  x1_low  x1_medium
0       2              0            1.0     0.0        1.0
1       2              0            0.0     1.0        0.0
2       5              0            0.0     1.0        0.0
3       2              1            0.0     0.0        1.0
4       3              0            0.0     1.0        0.0
```

[118]:

```
## RANDOM FOREST MODEL ##

# Instantiate the random forest classifier
rf = RandomForestClassifier(random_state=0)

# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [5, 7, None],
             'max_features': [0.3, 0.6],
            #  'max_features': 'auto'
             'max_samples': [0.7],
             'min_samples_leaf': [1,2],
             'min_samples_split': [2,3],
             'n_estimators': [75,100,200],
             }
```

```python
# Define a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# Instantiate the GridSearchCV object
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='recall')
```

[119]:
```python
## NOTE TAKES SOME TIME TO RUN ~6-10min

rf_cv.fit(X_train_final, y_train)
```

[119]:
```
GridSearchCV(cv=5, 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,
                                              oob_score=False, random_state=0,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [5, 7, None], 'max_features': [0.3, 0.6],
                         'max_samples': [0.7], 'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 3],
                         'n_estimators': [75, 100, 200]},
             pre_dispatch='2*n_jobs', refit='recall', return_train_score=False,
             scoring={'accuracy', 'f1', 'precision', 'recall'}, verbose=0)
```

[122]:
```python
# Examine best recall score
rf_cv.best_score_

"""
This model performs exceptionally well, with an average recall score of 0.989␣
↪across
the five cross-validation folds. After checking the precision score to be sure␣
↪the
model is not classifying all samples as claims, it is clear that this model is␣
↪making
almost perfect classifications.
"""
```

[122]: 0.9899924996808375

[123]:
```python
# Examine best parameters
rf_cv.best_params_
```

[123]:
```
{'max_depth': None,
 'max_features': 0.3,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 75}
```

[124]:
```python
## Build an XGBoost model ##

# Instantiate the XGBoost classifier
xgb = XGBClassifier(objective='binary:logistic', random_state=0)

# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [4,8,12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [300, 500]
             }

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

# Instantiate the GridSearchCV object
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=5, refit='recall')
```

[125]:
```python
xgb_cv.fit(X_train_final, y_train)
```

[125]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=None, gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, max…
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     predictor=None, random_state=0,
                                     reg_alpha=None, …),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.01, 0.1], 'max_depth': [4, 8, 12],
```

```
                         'min_child_weight': [3, 5],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='recall', return_train_score=False,
             scoring={'accuracy', 'f1', 'precision', 'recall'}, verbose=0)
```

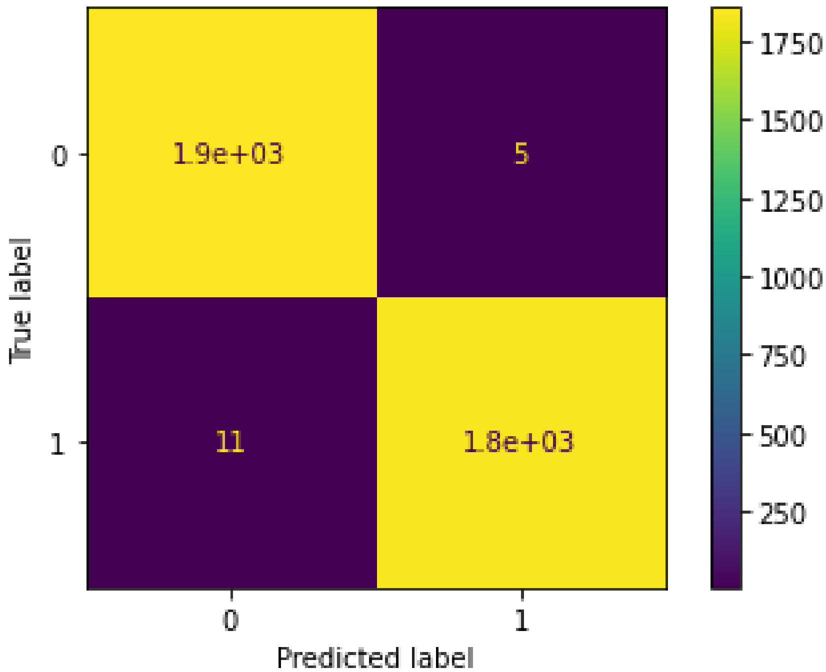
[126]:
```
xgb_cv.best_score_

"""
This model also performs exceptionally well. Although its recall score is very␣
↪slightly
higher than the random forest model's.
"""
```

[126]: 0.9933882292863526

[127]:
```
xgb_cv.best_params_
```

[127]:
```
{'learning_rate': 0.1,
 'max_depth': 12,
 'min_child_weight': 3,
 'n_estimators': 500}
```

[130]:
```
# Use the random forest "best estimator" model to get predictions on the␣
↪validation set
y_pred = rf_cv.best_estimator_.predict(X_val_final)

# Display the predictions on the validation set
y_pred

# Create a confusion matrix to visualize the results of the classification model

# Compute values for confusion matrix
log_cm = confusion_matrix(y_val, y_pred)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm,␣
↪display_labels=log_clf.classes_)

# Plot confusion matrix
log_disp.plot()

# Display plot
plt.show()
```

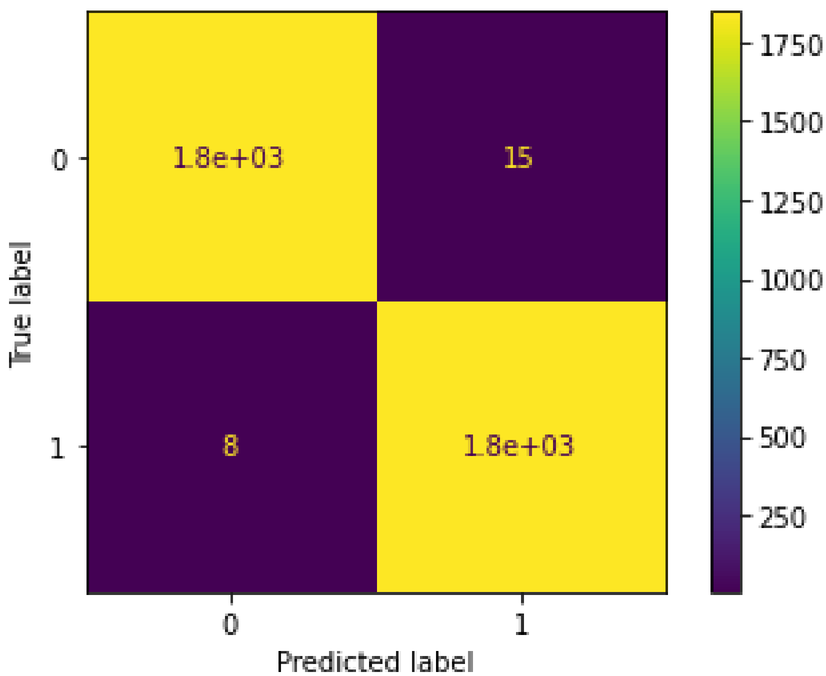


[133]:
```python
# Create a classification report
# Create classification report for random forest model
target_labels = ["Left", "Stayed"]
print(classification_report(y_val, y_pred, target_names=target_labels))

"""
The classification report above shows that the random forest model scores were␣
↪nearly perfect. The confusion matrix
indicates that there were 16 misclassifications- 5 false postives and 11 false␣
↪negatives.
"""
```

```
              precision    recall  f1-score   support

        Left       0.99      1.00      1.00      1863
      Stayed       1.00      0.99      1.00      1851

    accuracy                           1.00      3714
   macro avg       1.00      1.00      1.00      3714
weighted avg       1.00      1.00      1.00      3714
```

[144]:
```python
#Evaluate XGBoost model
y_pred = xgb_cv.best_estimator_.predict(X_val_final)
```

```
# Display the predictions on the validation set
y_pred

# Compute values for confusion matrix
log_cm = confusion_matrix(y_val, y_pred)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm,␣
↪display_labels=log_clf.classes_)

# Plot confusion matrix
log_disp.plot()

# Display plot
plt.show()
```



[145]:
```
# Create a classification report
# Create classification report for XGBOOST
target_labels = ["Left", "Stayed"]
print(classification_report(y_val, y_pred, target_names=target_labels))

"""
The results of the XGBoost model were also nearly perfect. Its errors tended to␣
↪be false positives. At 15
```

```
"""
```

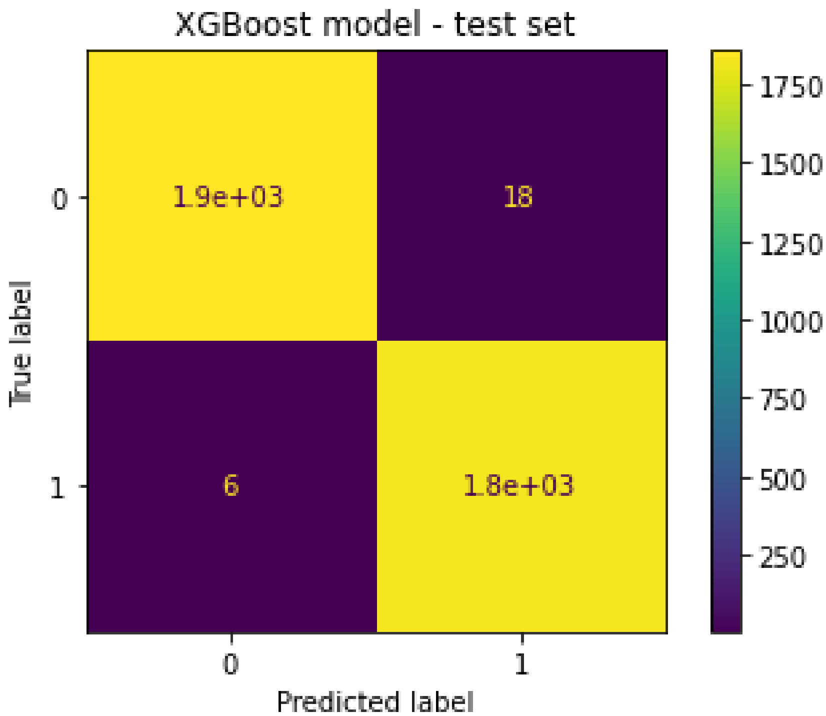
```
              precision    recall  f1-score   support

        Left       1.00      0.99      0.99      1863
      Stayed       0.99      1.00      0.99      1851

    accuracy                           0.99      3714
   macro avg       0.99      0.99      0.99      3714
weighted avg       0.99      0.99      0.99      3714
```

[145]:

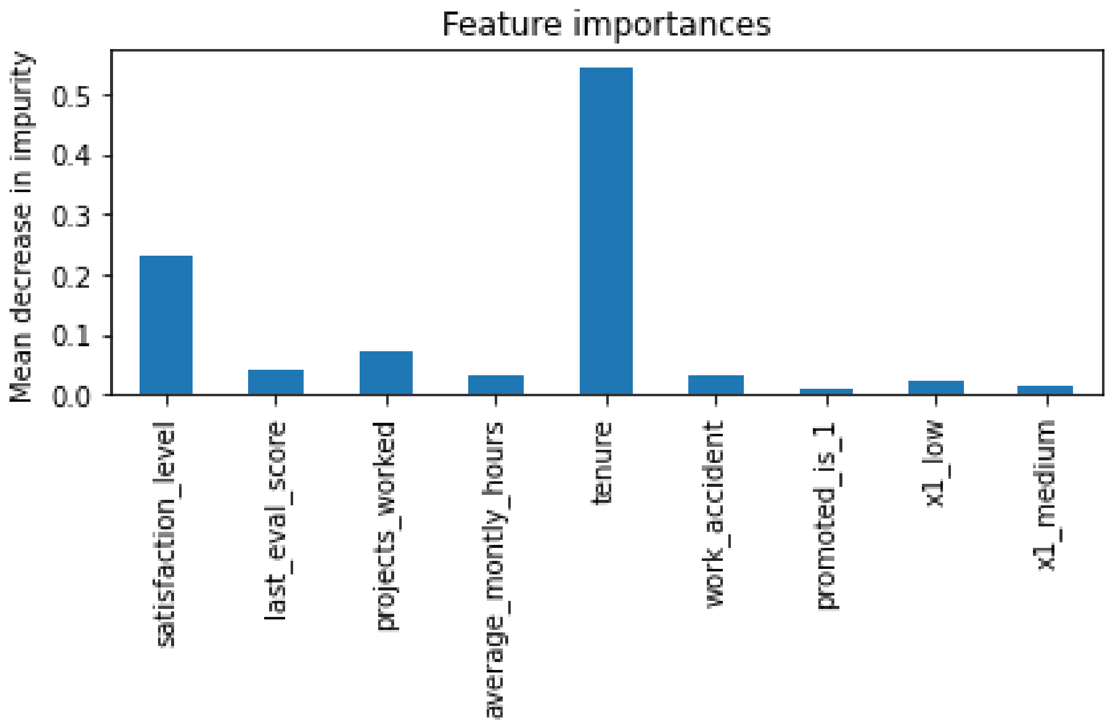
```
'\nThe results of the XGBoost model were also nearly perfect. Its errors tended
to be false positives. At 15 \n\n'
```

[148]:

```
# Use champion model to predict on test data

## CHAMPION MODEL WILL USE XGBoost model DUE TO IT HAVING THE MOST FASLE␣
↪POSITIVES
## SINCE PRIORITY WAS CAPTURING THOSE WHO WILL LEAVE WE'D RATHER PRIORITISE␣
↪HAVING MORE CATCHES
## ON THOSE LIKLEY TO LEAVE, EVEN IF THEY AREN'T


y_pred = xgb_cv.best_estimator_.predict(X_test_final)

# Compute values for confusion matrix
log_cm = confusion_matrix(y_test, y_pred)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm,␣
↪display_labels=log_clf.classes_)

# Plot confusion matrix
log_disp.plot()

# Display plot
plt.title('XGBoost model - test set');
plt.show()
```



[149]:
```
# Create a classification report
# Create classification report for random forest model
importances = xgb_cv.best_estimator_.feature_importances_
rf_importances = pd.Series(importances, index=X_test_final.columns)

fig, ax = plt.subplots()
rf_importances.plot.bar(ax=ax)
ax.set_title('Feature importances')
ax.set_ylabel('Mean decrease in impurity')
fig.tight_layout()

"""
The most predictive features all were related to satisfaction and tenure.
This is not unexpected, as analysis from prior EDA pointed to this conclusion.
"""
```



5 pacE: Execute Stage

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

Recall evaluation metrics

- **AUC** is the area under the ROC curve; it's also considered the probability that the model ranks a random positive example more highly than a random negative example.
- **Precision** measures the proportion of data points predicted as True that are actually True, in other words, the proportion of positive predictions that are true positives.
- **Recall** measures the proportion of data points that are predicted as True, out of all the data points that are actually True. In other words, it measures the proportion of positives that are correctly classified.
- **Accuracy** measures the proportion of data points that are correctly classified.
- **F1-score** is an aggregation of precision and recall.

Reflect on these questions as you complete the executing stage.

- What key insights emerged from your model(s)?
- What business recommendations do you propose based on the models built?
- What potential recommendations would you make to your manager/company?
- Do you think your model could be improved? Why or why not? How?

- Given what you know about the data and the models you were using, what other questions could you address for the team?
- What resources do you find yourself using as you complete this stage? (Make sure to include the links.)
- Do you have any ethical considerations in this stage?

Double-click to enter your responses here.

5.1 Step 4. Results and Evaluation

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

5.1.1 Summary of model results

[Double-click to enter your summary here.]

5.1.2 Conclusion, Recommendations, Next Steps

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

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