Google Advanced Data Analytics Capstone Project

Analyzing Employee Turnover: Predictive Factors for Departure from the Company

Salifort Motors

Created By: Fadhilah Arfu

Date: Desember 2023

Our Company

Salifort Motors is a fictional French-based alternative energy vehicle manufacturer. Its global workforce of over 100,000 employees research, design, construct, validate, and distribute electric, solar, algae, and hydrogen-based vehicles. Salifort's end-to-end vertical integration model has made it a global leader at the intersection of alternative energy and automobiles.

list of contents

Prolem Statement

Problem

Currently, there is a **high rate of turnover** among Salifort employees. (Note: In this context, turnover data includes both employees who choose to quit their job and employees who are let go). Salifort's senior leadership team is concerned about how many employees are leaving the company. Salifort strives to create a corporate culture that supports employee success and professional development. Further, the high turnover rate is **costly** in the **financial sense**. Salifort makes a **big investment** in recruiting, training, and upskilling its employees.

Deliverable

predict whether an employee **will leave the company**, and discover the **reasons** behind their departure, they could better understand the problem and develop a solution.

Business Case

Set Goals

Design a model that **predicts** whether an **employee will leave** the company based on their department, number of projects, average monthly hours, and any other data points

Dataset

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

	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
5	0.41	0.50	2	153	3	0	1	0	sales	low
6	0.10	0.77	6	247	4	0	1	0	sales	low
7	0.92	0.85	5	259	5	0	1	0	sales	low
8	0.89	1.00	5	224	5	0	1	0	sales	low
9	0.42	0.53	2	142	3	0	1	0	sales	low

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)

Data Exploration

Basic information about the data

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

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

Descriptive statistics about the data

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

Out[4]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000
mean	0.612834	0.716102	3.803054	201.050337	3.498233	0.144610	0.238083	0.021268
std	0.248631	0.171169	1.232592	49.943099	1.460136	0.351719	0.425924	0.144281
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000	0.000000	0.000000
25%	0.440000	0.560000	3.000000	156.000000	3.000000	0.000000	0.000000	0.000000
50%	0.640000	0.720000	4.000000	200.000000	3.000000	0.000000	0.000000	0.000000
75%	0.820000	0.870000	5.000000	245.000000	4.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	7.000000	310.000000	10.000000	1.000000	1.000000	1.000000

Data Cleaning

1. Check Missing Values

```
In [7]:  # Check for missing values
         ### YOUR CODE HERE ###
         df0.isna().sum()
```

```
Out[7]: satisfaction_level        0
        last_evaluation           0
        number_of_project         0
        average_montly_hours      0
        year_in_company           0
        Work_accident             0
        left                      0
        promotion_last_5years     0
        Department                0
        salary                    0
        dtype: int64
```

the data does not have missing values

2. Check duplicates

```
In [8]:  # Check for duplicates
         ### YOUR CODE HERE ###
         df0.duplicated().sum()
```

```
Out[8]: 3008
```

There are 3.008 duplicate rows. That is 20% of the data

Drop Duplcates

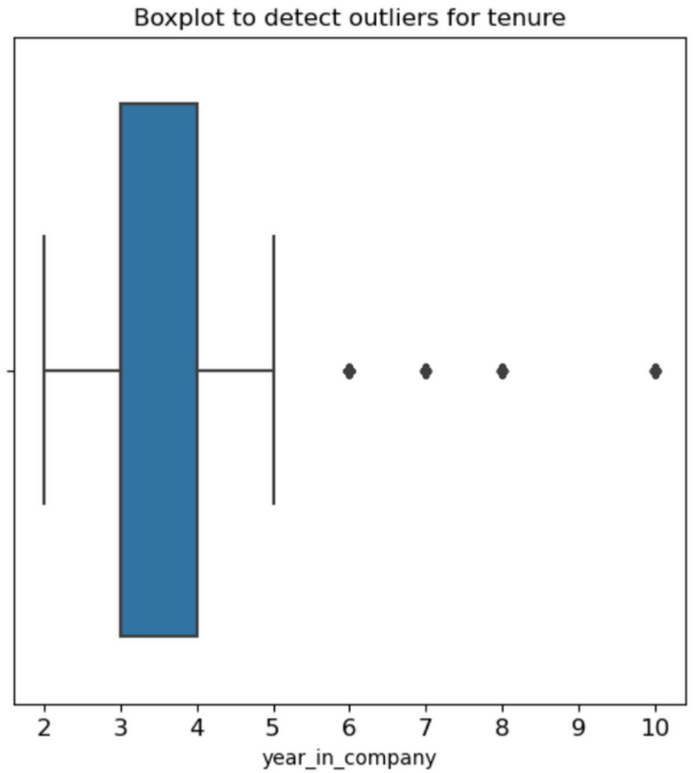
```
In [10]: # Drop duplicates and save resulting dataframe in a new variable as needed
         ### YOUR CODE HERE ###
         df1 = df0.drop_duplicates(keep='first')

         # Display first few rows of new dataframe as needed
         ### YOUR CODE HERE ###
         df1.head(10)
```

Out[10]:

	satisfaction_level	last_evaluation	number_of_project	average_montly_hours	year_in_company	Work_accident	left	promotion_last_5years	Department
0	0.38	0.53	2	157	3	0	1	0	sales
1	0.80	0.86	5	262	6	0	1	0	sales
2	0.11	0.88	7	272	4	0	1	0	sales
3	0.72	0.87	5	223	5	0	1	0	sales
4	0.37	0.52	2	159	3	0	1	0	sales
5	0.41	0.50	2	153	3	0	1	0	sales
6	0.10	0.77	6	247	4	0	1	0	sales
7	0.92	0.85	5	259	5	0	1	0	sales
8	0.89	1.00	5	224	5	0	1	0	sales
9	0.42	0.53	2	142	3	0	1	0	sales

3. Check Outlier

In [11]:
```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
### YOUR CODE HERE ###
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers for tenure', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=df1['year_in_company'])
plt.show()
```



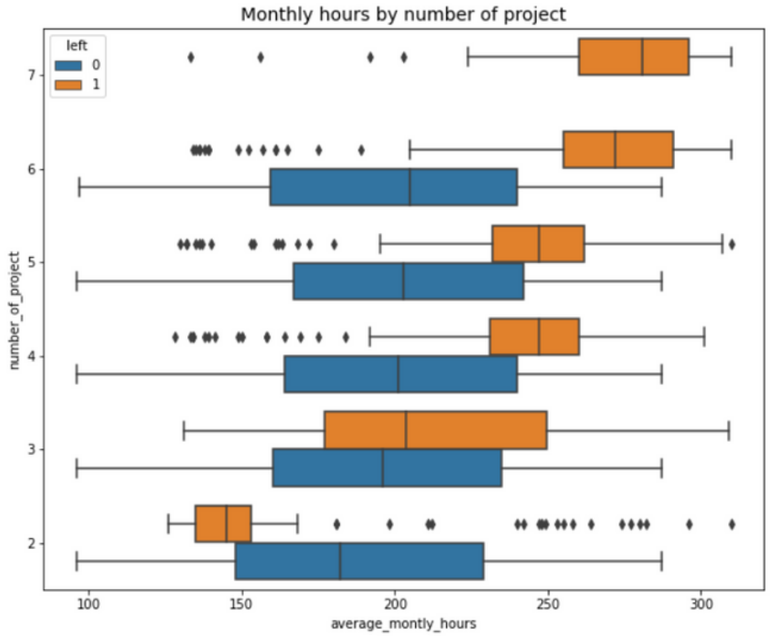
In [12]:
```
# Determine the number of rows containing outliers
### YOUR CODE HERE ###

# Compute 25 percentile value in "year_in_company"
percentile25 = df1['year_in_company'].quantile(0.25)

# Compute 75 percentile value in "year_in_company"
percentile75 =  df1['year_in_company'].quantile(0.75)

# Compute interquartile range in "year_in_company"
iqr = percentile75 - percentile25

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

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

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

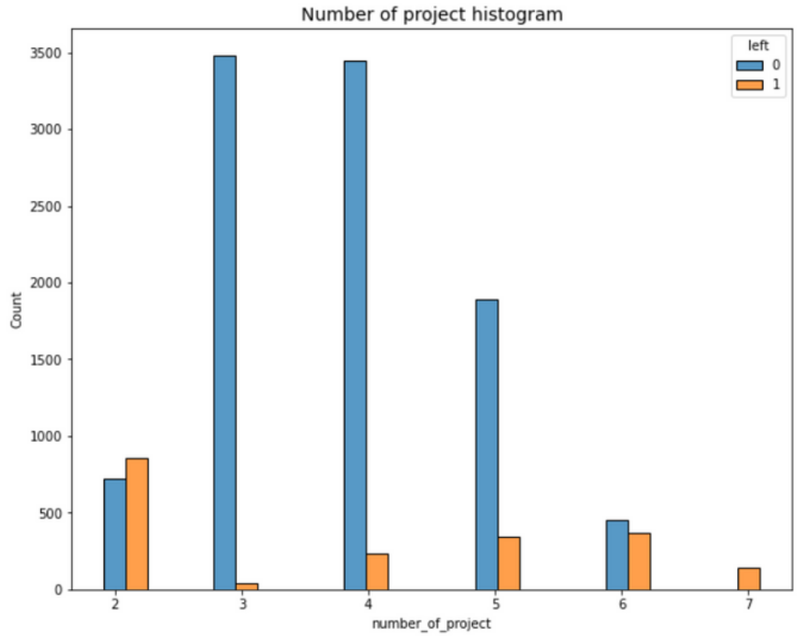
```
Lower limit: 1.5
Upper limit: 5.5
Number of rows in the data containing outliers in `tenure`: 824
```

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.

Exploratory Data Analysis (EDA)

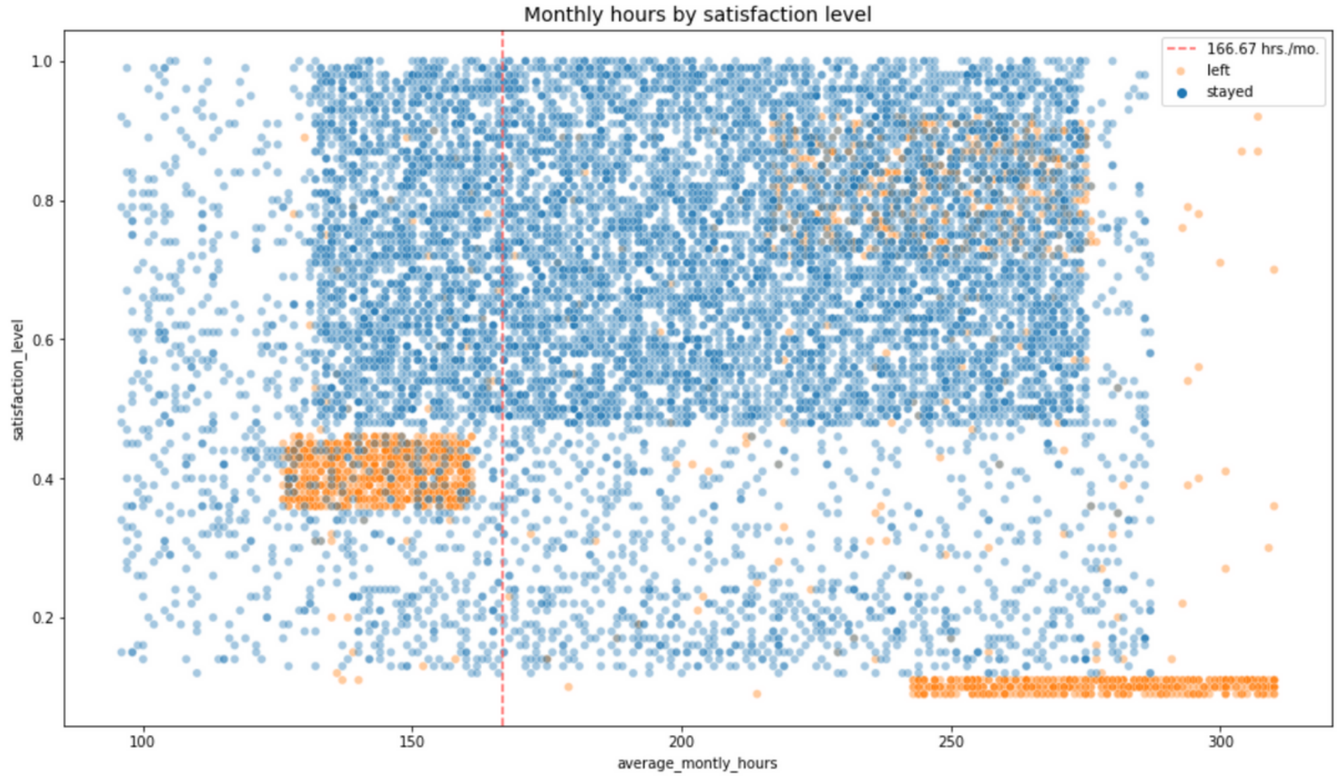
Distributions of Employees Who Stayed Versus Those Who Left Based On Average Monthly Hours and Number Project





1. We noticed two types of employees leaving the company: **(A)** those **working way less than others with the same projects,** maybe indicating **they were let go or about to leave**, and **(B) those working a lot more**, possibly quitting, especially if they were **big contributors** to their projects.
2. People who left and had **either six or seven projects** had **longer work hours (255–295 hours/month**) compared to other groups.
3. It looks like **the best number of projects** for employees to stay happy is around **3–4.** Not many people leave from these groups.
4. If we assume a regular 40-hour workweek and two weeks of vacation per year, the average monthly working hours should be around 166.67. Except for the group with two projects, everyone, even those who stayed, worked more than this average. It seems like people here might be **working too much**

The Average Monthly Hours versus The Satisfaction Levels.

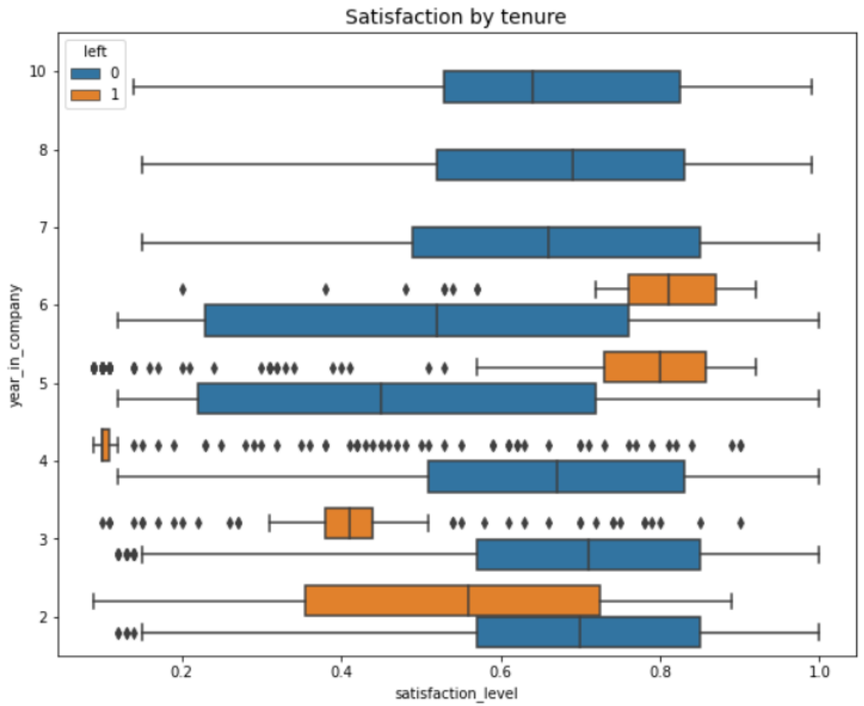


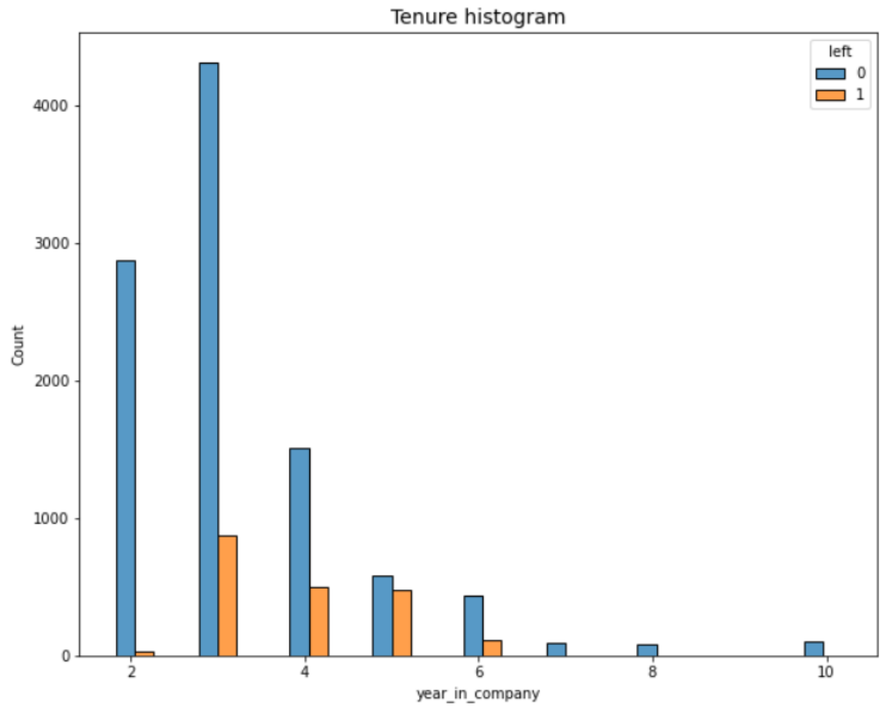
The scatterplot above shows that there was a sizeable group of employees who **worked ~240–315 hours per month.** 315 hours per month is **over 75 hours per week** for a whole year. It's likely this is related to their **satisfaction levels being close to zero.**

The plot also shows another group of people who left, those who had more normal working hours. Even so, their satisfaction was only around 0.4. It's **difficult to speculate about why they might have left**. It's possible they **felt pressured to work more**, considering so many of their peers worked more. And that pressure could have lowered their satisfaction levels.

Finally, there is a group who **worked ~210–280 hours per month**, and they had **satisfaction levels ranging ~0.7–0.9.**

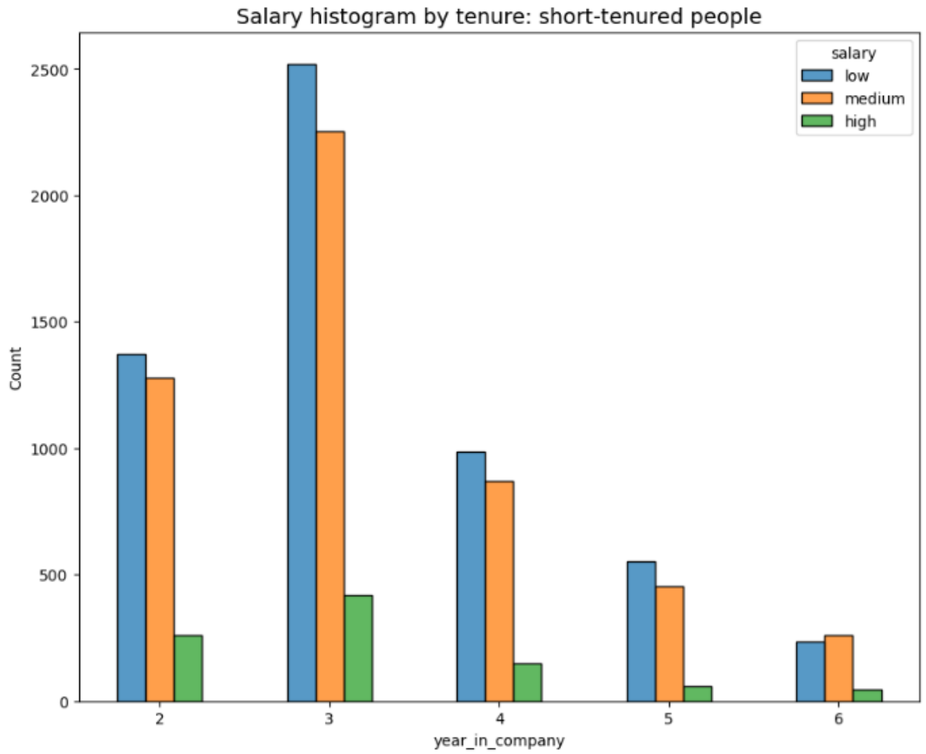
Distrubution Satisfaction Level by Year in Company (Tenure)

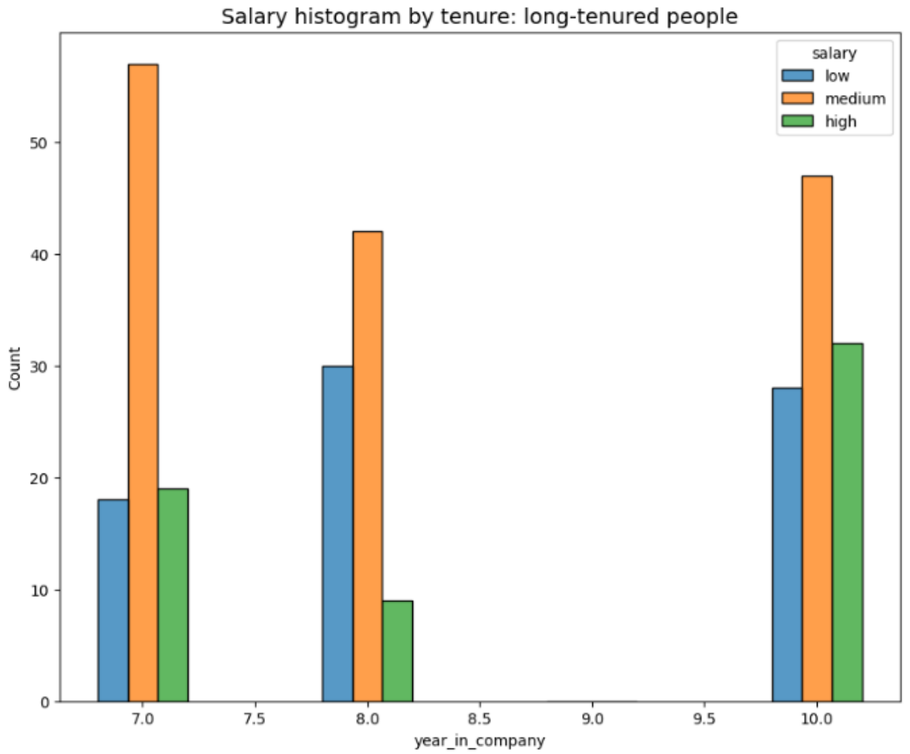




1. Employees who left fall into two general categories: **dissatisfied employees with shorter tenures** and **very satisfied employees with medium-length tenures**.
2. **Four-year employees** who left seem to have an **unusually low satisfaction level**. It's worth investigating changes to company policy that might have affected people specifically at the four-year mark, if possible.
3. **The longest-tenured employees didn't leave**. Their satisfaction levels aligned with those of newer employees who stayed.
4. The histogram shows that there are **relatively few longer-tenured employees**. It's possible that they're the **higher-ranking, higher-paid employees.**

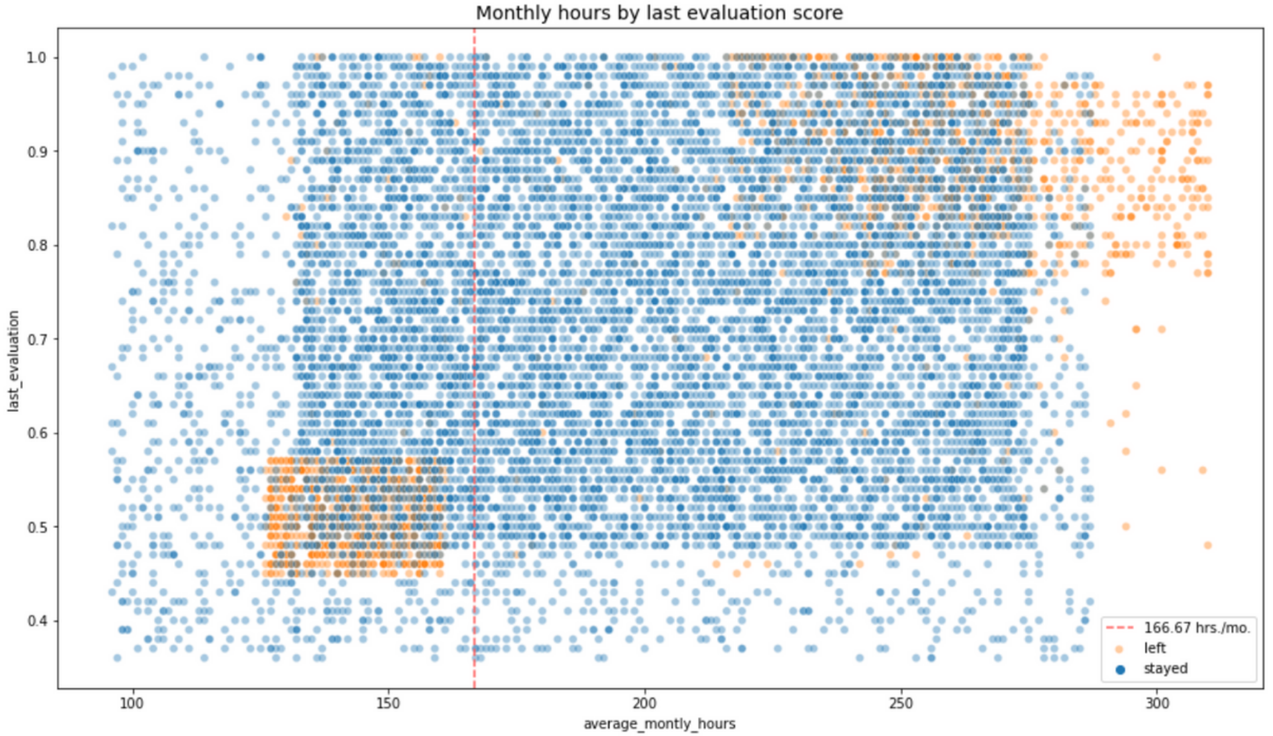
Distribution Salary Levels For Different Tenures.





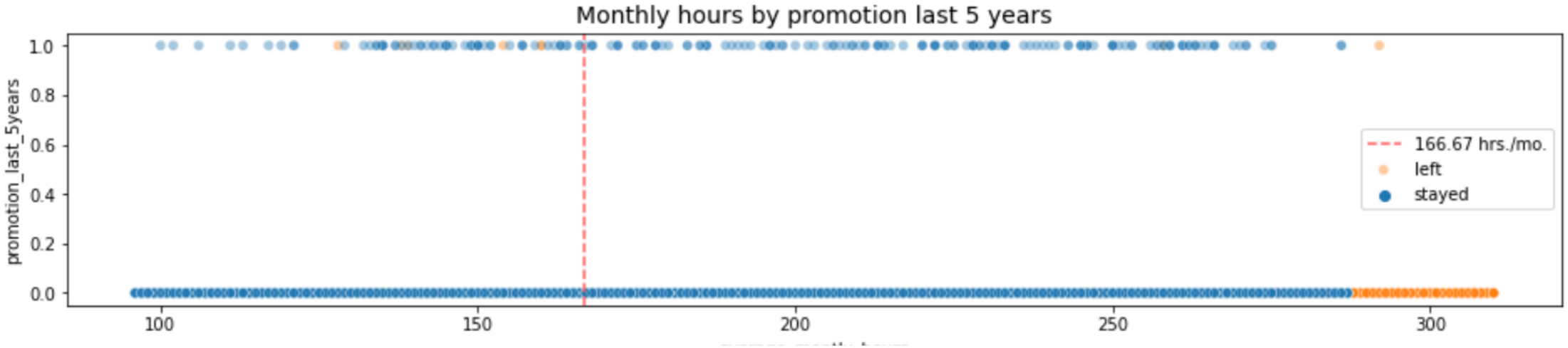
The plots above show that long-tenured employees were not disproportionately comprised of higher-paid employees.

There's a correlation between working long hours and receiving high evaluation scores?



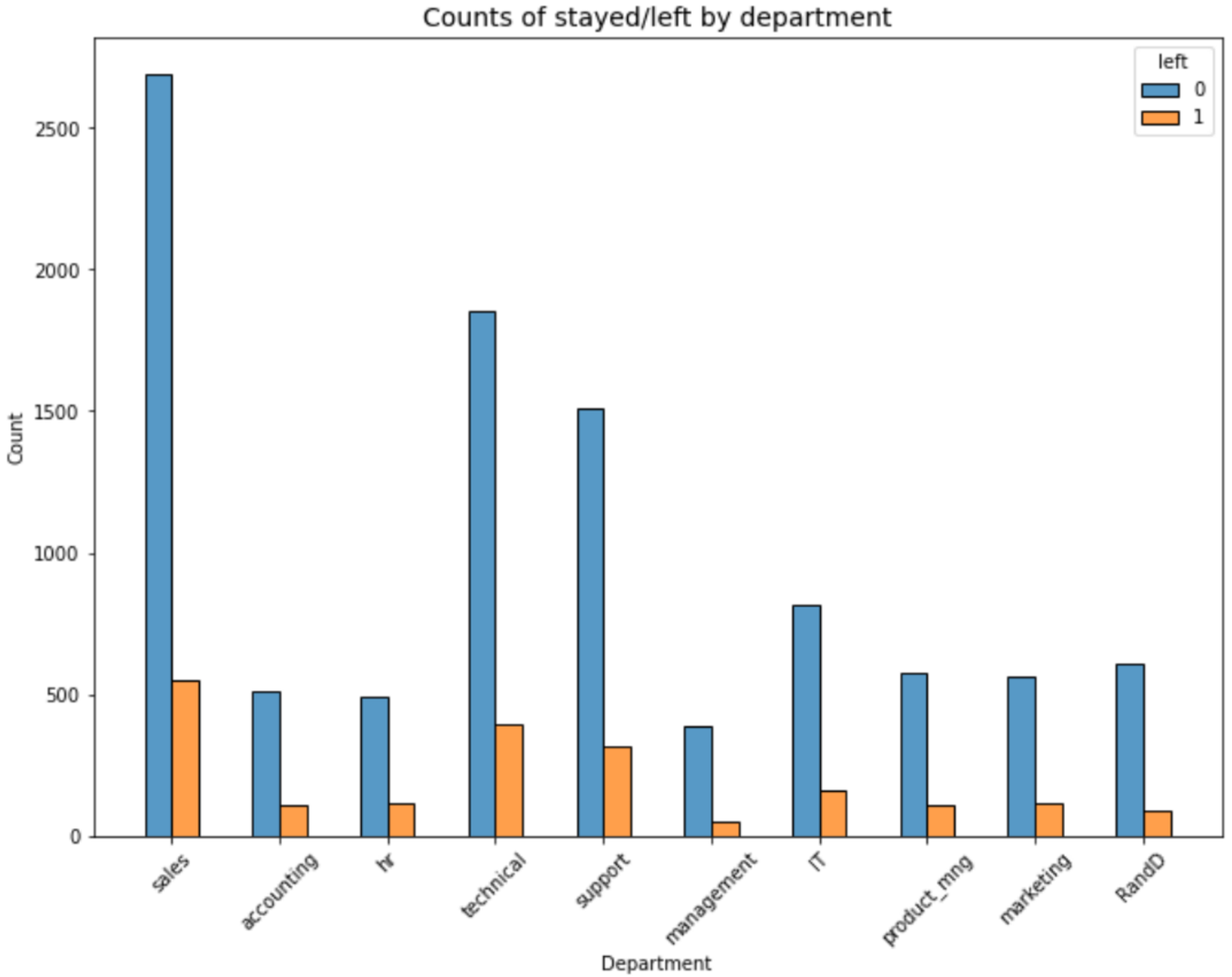
1. The scatterplot indicates two groups of employees who left: **overworked employees who performed very well** and **employees who worked slightly under the nominal monthly average of 166.67 hours with lower evaluation scores**.
2. There seems to be a **correlation between hours worked and evaluation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

whether employees who worked very long hours were promoted in the last five years?



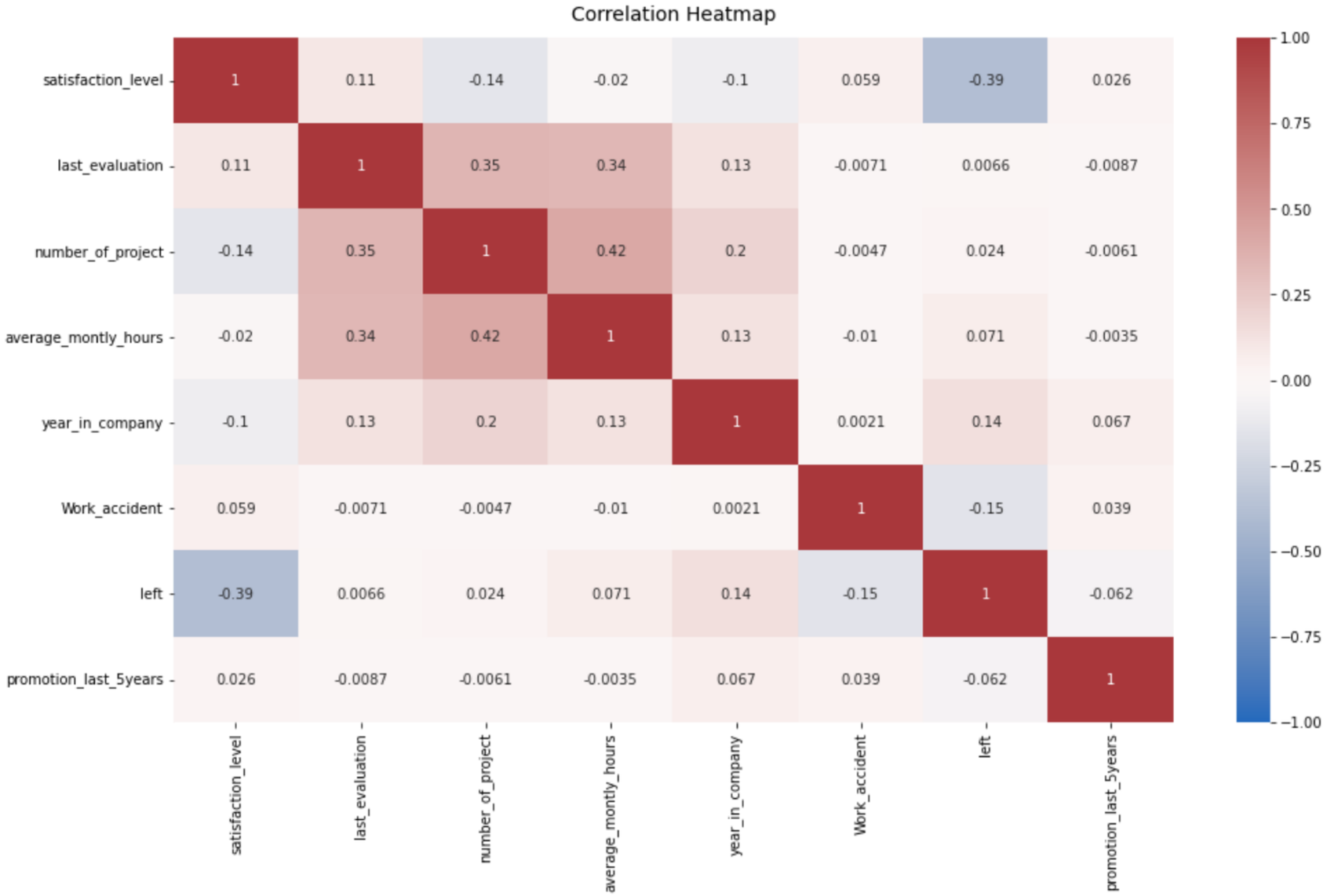
1. very few employees who were promoted in **the last five years left**
2. very few employees who worked **the most hours** were promoted
3. all of the employees **who left** were working **the longest hours**

How The Employees Who Left are Distributed Across Departments?



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

Correlations between variables in the data.



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

Build Machine Learning

Modeling Approach A: Logistic Regression Model

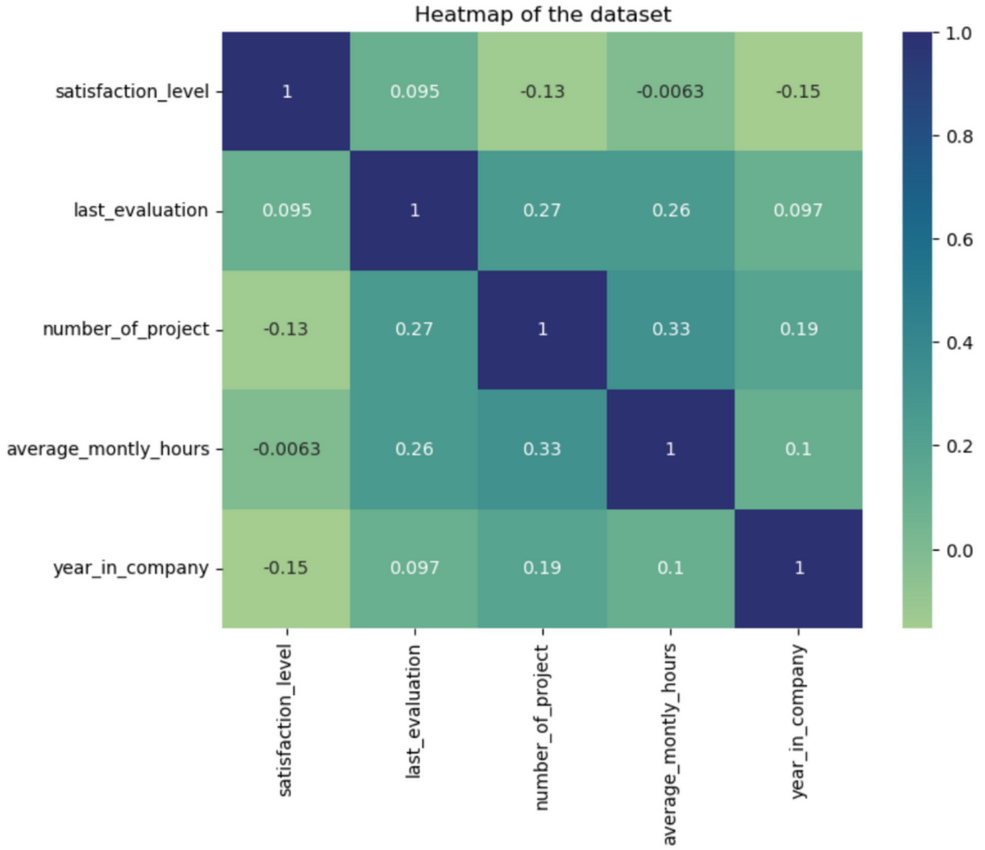
Logistic Regression Model

Prepare Data

1. Convert Categorical data to numerical data

```
### YOUR CODE HERE ###

# Copy dataframe
df_enc = df1.copy()

# Encode the "salary" columns as an ordinal numeric columns
df_enc['salary'] = (df_enc['salary'].astype('category').
                    cat.set_categories(['low', 'medium', 'high']).cat.codes)

# Dummy encode the 'Departmen' columns
df_enc = pd.get_dummies(df_enc, drop_first=True)
```

2. Check correlated variable to avoid multicollinearity



3. Handle Outlier

Since logistic regression is quite sensitive to outliers, it would be a good idea at this stage to remove the outliers in the `tenure` column that were identified earlier.

```
# Select rows without outlier in "year_in_company" and save resulting dataframe to new variable
df_logreg = df_enc[(df_enc['year_in_company']>= lower_limit) & (df_enc['year_in_company'] <= upper_limit)]

# Display a few rows
df_logreg.head(5)
```

	satisfaction_level	last_evaluation	number_of_project	average_montly_hours	year_in_company	Work_accident	left	promotion_last_5years	salary	Department_R
0	0.38	0.53	2	157	3	0	1	0	0	
2	0.11	0.88	7	272	4	0	1	0	1	
3	0.72	0.87	5	223	5	0	1	0	0	
4	0.37	0.52	2	159	3	0	1	0	0	
5	0.41	0.50	2	153	3	0	1	0	0	

Logistic Regression Model

Build The Model

1. Isolate Target and Predictive Variable

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

      # Display a few rows
      y.head(5)
```

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

      # Display a few rows
      X.head()
```

2. Split the data into training set and testing set

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

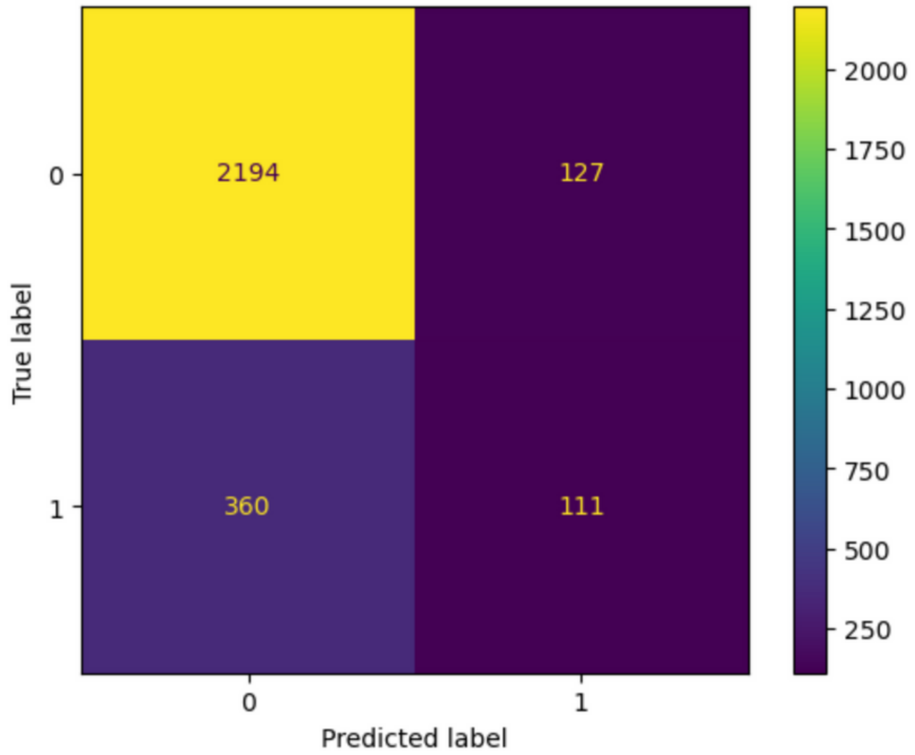
3. Construct a logistic regression model and fit it to the training dataset.

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

4. Test the logistic regression model: use the model to make predictions on the test set.


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

Result and Evaluation Logistic Regression



	precision	recall	f1-score	support
Predicted would not leave	0.86	0.95	0.90	2321
Predicted would leave	0.47	0.24	0.31	471
accuracy			0.83	2792
macro avg	0.66	0.59	0.61	2792
weighted avg	0.79	0.83	0.80	2792

The classification report above shows that the logistic regression model achieved a **precision of 79%**, **recall of 82%**, **f1-score of 80%** (all weighted averages), and **accuracy of 82%**. However, if it's most important to predict employees who leave, then the scores are **significantly lower.**

Modeling Approach B: Tree-based Model

Decision Tree And Random Forest

Tree Base Model
Round 1

Decision Tree Model

Build The Model

1 Isolate Target and Predictive Variable

[38]:
```
# Isolate the outcome variable
y = df_enc['left']

# Display the first few row of 'y'
y.head(5)
```

[39]:
```
# Select the features you want to use in your model
X = df_enc.drop('left', axis=1)

# Display a few rows
X.head()
```

2 Split the data into training set and testing set

[32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=0)
```

3 Construct a decision tree model and set up cross-validated grid-search to exhaustively search for the best model parameters.

[41]:
```
# Instantiate model
tree = DecisionTreeClassifier(random_state=0)

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

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

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

4 Fit the decision tree model to the training data.

[42]:
```
%%time
tree1.fit(X_train, y_train)
```

```
CPU times: total: 3.41 s
Wall time: 3.98 s
```

[42]:
```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             refit='roc_auc',
             scoring={'accuracy', 'precision', 'recall', 'f1', 'roc_auc'})
  estimator: DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
  DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

Decision Tree Model

Build The Model

5 Identify the optimal values for the decision tree parameters.

```
[43]: # Check best parameter
      tree1.best_params_
```

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

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

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

```
[44]: 0.969819392792457
```

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

Random Forest Model

Build The Model

1. Construct a random forest model and set up cross-validated grid-search to exhaustively search for the best model parameters.

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

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

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

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

2. Fit the decision tree model to the training data.

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

```
[48]: GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=0),
                  param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                              'max_samples': [0.7, 1.0],
                              'min_samples_leaf': [1, 2, 3],
                              'min_samples_split': [2, 3, 4],
                              'n_estimators': [300, 500]},
                  refit='roc_auc',
                  scoring={'accuracy', 'precision', 'recall', 'f1', 'roc_auc'})
      estimator: RandomForestClassifier
        RandomForestClassifier
```

3. Identify the best AUC score achieved by the random forest on the training data and optimal values for the parameters of the random forest model

```
[55]: rf1.best_score_

[55]: 0.9803963087745455
```

Better than Decision Tree Model

```
[56]: rf1.best_params_

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

Evaluation Model

Decision Tree and Random Forest

1 Write a function that will help you extract all the scores from the grid search.

```
[45]: 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 result 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(metrix) score
          best_estimator_result = cv_results.iloc[cv_results[metric_dict[metric]].idxmax(), :]

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

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

          return table
```

2 Use the function just defined to get all the scores from grid search.

```
In [46]: tree1_cv_result = make_results('decision tree cv', tree1, 'auc')
         tree1_cv_result
```

Out[46]:

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

```
In [57]: # Get all CV Score
         rf1_cv_result = make_results('random forest cv', rf1, 'auc')
         print(tree1_cv_result)
         print(rf1_cv_result)
```

```
              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   0.948704  0.915614  0.931836  0.977761  0.980396
```

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

Evaluation Model

Test Random Forest Model usisng test dataset

1 Define a function that gets all the scores from a model's prediction

In [58]:
```
def get_score(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
```

2 use the best performing model to predict the test data

In [59]:
```
rf1_test_score = get_score('random forest rf1', rf1, X_test, y_test)
rf1_test_score
```

Out[59]:

	model	precision	recall	f1	accuracy	AUC
0	random forest rf1	0.964211	0.919679	0.941418	0.980987	0.956439

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

Tree Base Model
Round 2

We might be skeptical of the high evaluation scores. There is a chance that there is some **data leakage occurring**. Data leakage is when we use data to train our model that should not be used during training, either because it appears in the test data or because it's not data that you'd expect to have when the model is actually deployed. Training a model with leaked data can give an **unrealistic score** that is **not replicated** in production.

In this case, it's likely that the company won't have **satisfaction levels** reported for all of its employees. It's also possible that **the average_monthly_hours** column is a source of **some data leakage**. If employees have already decided upon quitting, or have already been identified by management as people to be fired, they may be working fewer hours.

FEATURE ENGINERING

1 Dropping **satisfaction_level** and creating a new feature that roughly captures whether an employee is **overworked**.

In [60]:
```
# Drop 'satisfaction_level' and save resulting dataframe in new variable
df2 = df_enc.drop('satisfaction_level', axis=1)

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

In [61]:
```
# Create 'overworked' column. For now, it's identical to average monthly hours.
df2['overworked'] = df2['average_montly_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
```

We could define being **overworked** as **working more than 175 hours** per month on average.

- **df2['overworked'] > 175** creates a seriues of boolean consisiting of **True** for every value > 175 and **False** for every values ≤ 175
- **.astype(int)** convert all True to 1 and all False to 0

In [62]:
```
# Define `overworked` as working > 175 hrs/week
df2['overworked'] = (df2['overworked'] > 175).astype(int)

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

Out[62]:
```
0    0
1    1
2    1
3    1
4    0
Name: overworked, dtype: int32
```

2 Drop the average_monthly_hours columns

In [63]:
```
# Drop the 'average_monthly_hours' columns
df2 = df2.drop('average_montly_hours', axis=1)

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

Decision Tree Model

Build The Model

1. Isolate Target and Predictive Variable

```
In [64]:
# Isolate the outcome variable
y = df2['left']

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

2. Split the data into training set and testing set

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

3. Construct a Decision Tree model and set up cross-validated grid-search to exhuastively search for the best model parameters.

```
[66]:
# Instantiate model
tree2 = DecisionTreeClassifier(random_state=0)

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

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

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

4. Fit the decision tree model to the training data.

```
In [67]:
%%time
tree2.fit(X_train, y_train)
```

```
CPU times: total: 1.89 s
Wall time: 1.99 s
```

```
Out[67]:
GridSearchCV
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             refit='roc_auc',
             scoring={'accuracy', 'precision', 'recall', 'f1', 'roc_auc'})
```

Decision Tree Model

Build The Model

5 Identify the optimal values for the decision tree parameters.

```
In [68]: tree2.best_params_
Out[68]: {'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 6}
```

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

```
In [69]: tree2.best_score_
Out[69]: 0.9594361127439034
```

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

7 Check the score

```
In [70]: # Get all CV scores
tree2_cv_result = make_results('decision tree 2 cv', tree2, 'auc')
print(tree1_cv_result)
print(tree2_cv_result)
```

```
              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 tree 2 cv   0.856693  0.903553  0.878882  0.958523  0.959436
```

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

Random Forest

Build The Model

1. Construct a random forest model and set up **cross-validated** grid-search to exhaustively search for the **best model parameters.**

In [71]:
```
# Instantiate model
rf = RandomForestClassifier(random_state=0)

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

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

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

2. Fit the decision tree model to the training data.

In [72]:
```
%%time
rf2.fit(X_train, y_train)
```

```
CPU times: total: 10min 25s
Wall time: 14min 13s
```

Out[72]:

GridSearchCV

```
GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=0),
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             refit='roc_auc',
             scoring={'accuracy', 'precision', 'recall', 'f1', 'roc_auc'})
```

estimator: RandomForestClassifier

```
RandomForestClassifier(random_state=0)
```

RandomForestClassifier

Random Forest

Build The Model

3 Identify optimal values for the parameters of the random forest model

```
In [75]: rf2.best_params_
```

```
Out[75]: {'max_depth': 5,
          'max_features': 1.0,
          'max_samples': 1.0,
          'min_samples_leaf': 1,
          'min_samples_split': 4,
          'n_estimators': 300}
```

4 The best AUC score achieved by the random forest on the training data and

```
In [76]: rf2.best_score_
```

```
Out[76]: 0.9649187452509641
```

5 Use the function just defined to get all the scores from grid search.

```
In [77]: # Get all CV scores
rf2_cv_result = make_results('random fores2 cv', rf2, 'auc')
print(tree2_cv_result)
print(rf2_cv_result)
```

```
                model  precision    recall        F1  accuracy       auc
0  decision tree 2 cv   0.856693  0.903553  0.878882  0.958523  0.959436
            model  precision    recall        F1  accuracy       auc
0  random fores2 cv   0.866431  0.880763  0.873246  0.957634  0.964919
```

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

Test Champions Model and Evaluation

Test Campions Model

Random Forest- Round 2

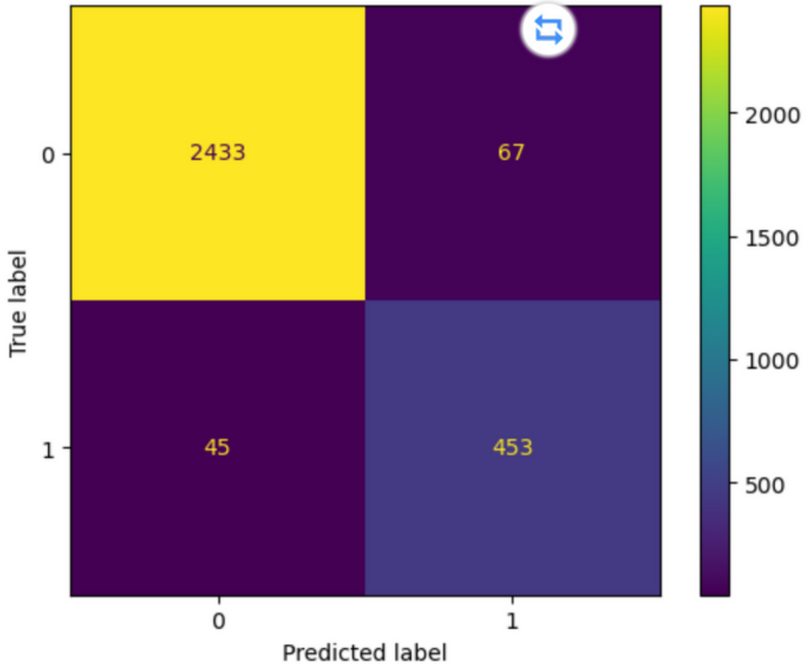
In [78]:
```
# Get predictions on test data
rf2_test_scores = get_score('random forest2 test', rf2, X_test, y_test)
rf2_test_scores
```

Out[78]:

	model	precision	recall	f1	accuracy	AUC
0	random forest2 test	0.871154	0.909639	0.88998	0.962642	0.941419

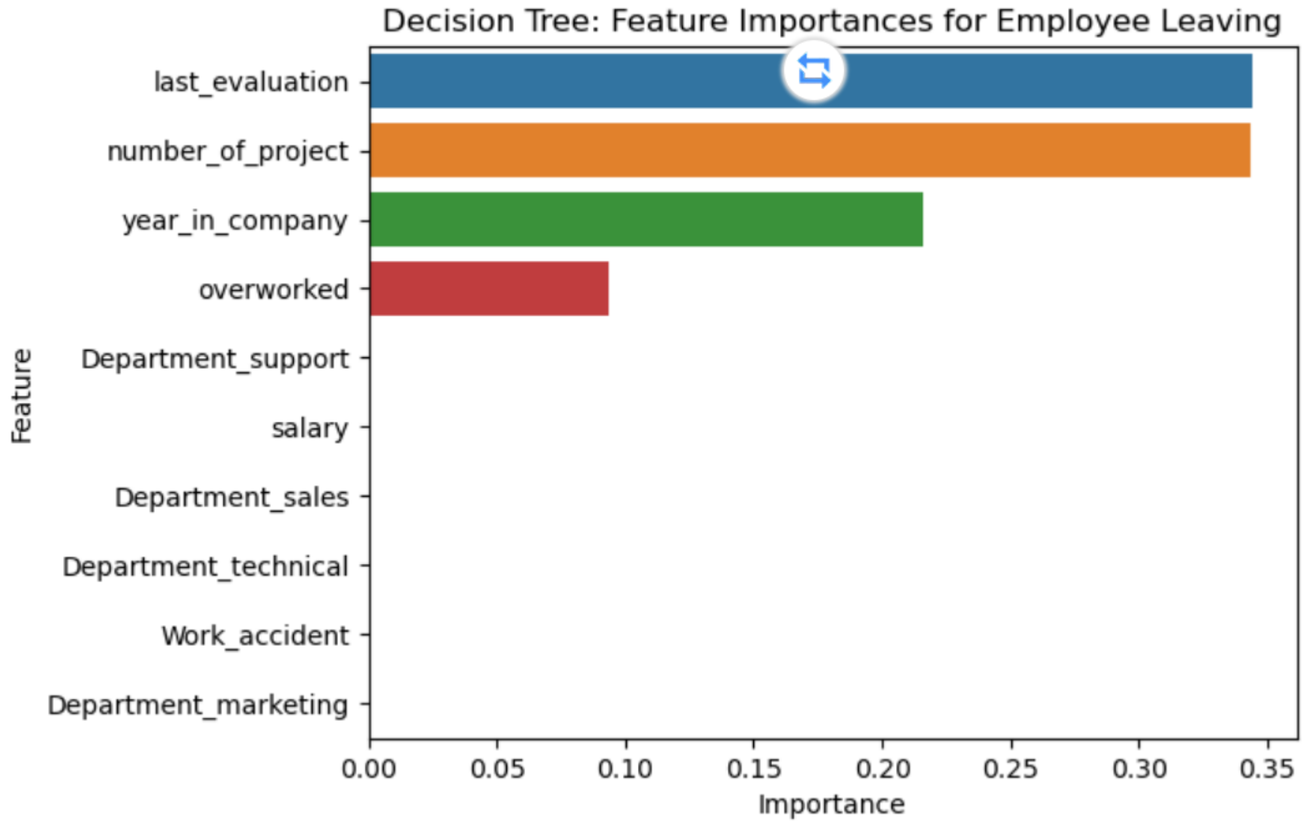
This seems to be a **stable**, well-performing final model.

confusion matrix to visualize how well it predicts on the test set



Feature Importance

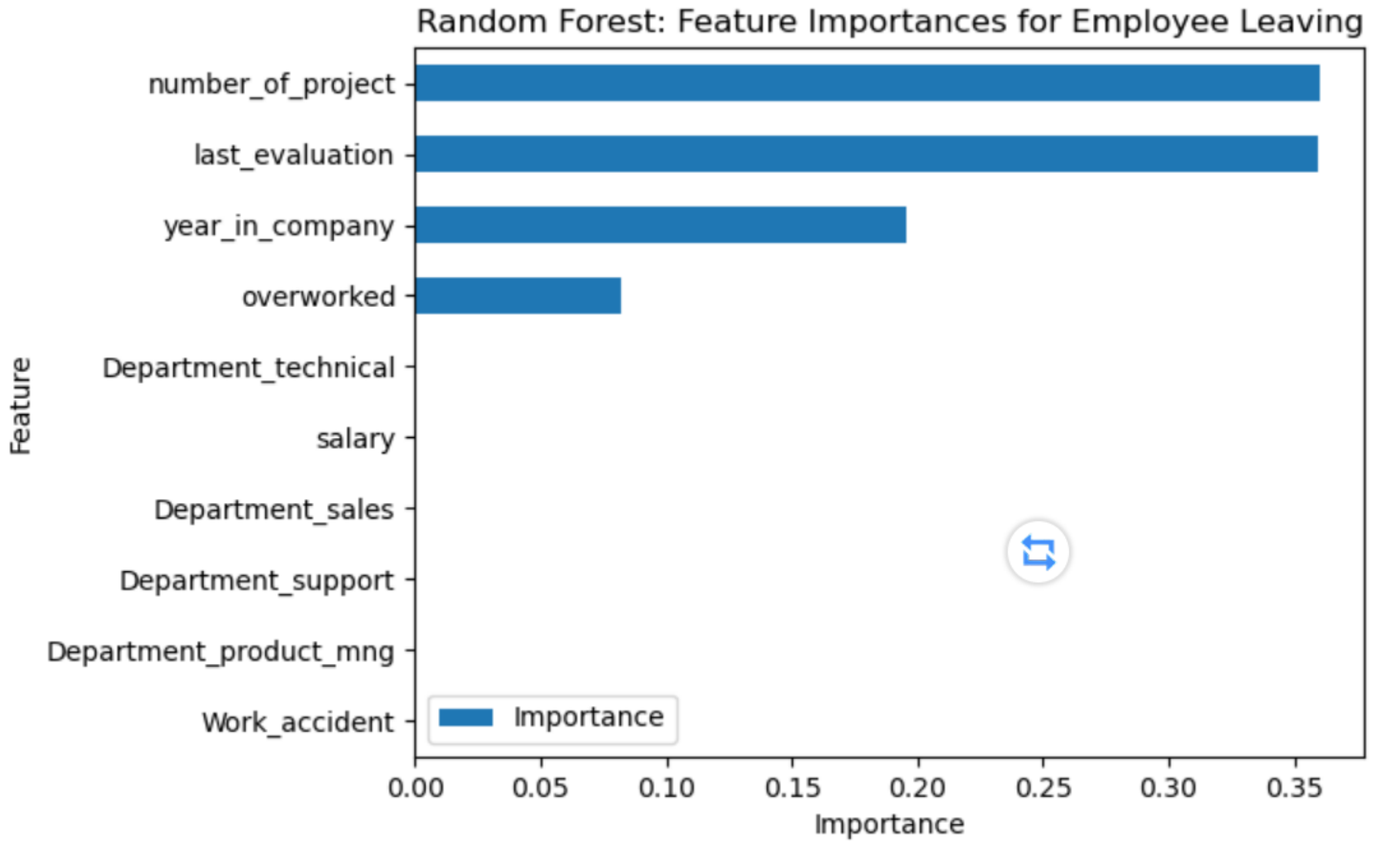
Decision Tree



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

Feature Importance

Random Forest



The plot shows that in this random forest model, **last_evaluation, number_project, year_in_company, and overworked** have the **highers importance**, in that order. These variable are most helpful in **predicting the outcome variable, left**, and they are the same as the ones used by the decision tree model.

Summary

Logistic Regression

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

Tree-based Machine Learning

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

Conclusion and Recommendations

The models and the feature importances extracted from the models confirm that employees at the company are overworked.

To **retain employees**, the following recommendations could be presented to **the stakeholders**:

- **Limit the number of projects** employees can engage in.
- **Evaluate the possibility of promoting employees** with a tenure of at least four years, or **investigate** the reasons behind the **dissatisfaction of those with four-year tenures.**
- **Provide incentives** for employees putting in longer hours, or **eliminate the requirement** for extended work hours altogether.
- Ensure that employees are familiar with **the company's overtime pay policies**. **Clearly communicate** expectations regarding workload and time off if they are not explicitly stated.
- Facilitate company-wide and team-specific discussions to comprehend and address the **overall work culture,** both **universally** and in **particular contexts**.
- Do not exclusively reserve **high evaluation scores** for employees **working 200+ hours** per month. Consider implementing a proportional scale to **reward employees** based on their **contributions and efforts**.

Thank You

For Your Attention