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.

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)

In [21]:
```
# For data manipulation
import numpy as np
import pandas as pd
```

```
# For data visualization
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option('display.max_columns', None)

# For data modeling
from xgboost import XGBClassifier
from xgboost import XGBRegressor
from xgboost import plot_importance

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

# For metrics and helpful functions
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,\
f1_score, confusion_matrix, ConfusionMatrixDisplay, classification_report
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.tree import plot_tree

# For saving models
import pickle
```

In [22]:
```
file_path = r'C:\Users\Pratik Sonawane\Downloads\HR_capstone_dataset.csv'
df = pd.read_csv(file_path)
df.head()
```

Out[22]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departmer
0	0.38	0.53	2	157	3	0	1	0	sale
1	0.80	0.86	5	262	6	0	1	0	sale
2	0.11	0.88	7	272	4	0	1	0	sale
3	0.72	0.87	5	223	5	0	1	0	sale
4	0.37	0.52	2	159	3	0	1	0	sale

EDA and data cleaning

In [23]:
```
df.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
```

In [24]:
```
df.describe()
```

Out[24]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5yea
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.00000
mean	0.612834	0.716102	3.803054	201.050337	3.498233	0.144610	0.238083	0.02126
std	0.248631	0.171169	1.232592	49.943099	1.460136	0.351719	0.425924	0.14428
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000	0.000000	0.00000
25%	0.440000	0.560000	3.000000	156.000000	3.000000	0.000000	0.000000	0.00000
50%	0.640000	0.720000	4.000000	200.000000	3.000000	0.000000	0.000000	0.00000
75%	0.820000	0.870000	5.000000	245.000000	4.000000	0.000000	0.000000	0.00000
max	1.000000	1.000000	7.000000	310.000000	10.000000	1.000000	1.000000	1.00000

Employee Satisfaction: The average satisfaction level is 0.61, indicating room for improvement.

Performance Evaluations: The average evaluation score is 0.72, suggesting generally positive performance.

Workload: Employees work on an average of 3.8 projects and 201 hours per month, suggesting a potentially demanding workload.

Tenure: The average tenure is 3.5 years, implying a moderate level of employee retention.

Employee Turnover: 23.8% of employees left the company, which is a significant turnover rate.

Work Accidents: 14.5% of employees have experienced a work accident, which is a potential concern for employee well-being and safety.

Promotions: Only 2.1% of employees have received a promotion in the last 5 years, suggesting limited opportunities for career advancement.

In [25]:
```
df = df.rename(columns={'Work_accident': 'work_accident',
                        'average_montly_hours': 'average_monthly_hours',
                        'time_spend_company': 'tenure',
                        'Department': 'department'})

df.columns
```

Out[25]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_monthly_hours', 'tenure', 'work_accident', 'left',
       'promotion_last_5years', 'department', 'salary'],
      dtype='object')
```

In [26]:
```
df.isna().sum()
```

Out[26]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_monthly_hours    0
tenure                   0
work_accident            0
left                     0
promotion_last_5years    0
department               0
salary                   0
dtype: int64
```

In [27]:
```
df.duplicated().sum()
```

Out[27]:
```
3008
```

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

In [28]:
```
df[df.duplicated()].head()
```

Out[28]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_accident	left	promotion_last_5years	department	salary
396	0.46	0.57	2	139	3	0	1	0	sales	low
866	0.41	0.46	2	128	3	0	1	0	accounting	low
1317	0.37	0.51	2	127	3	0	1	0	sales	medium
1368	0.41	0.52	2	132	3	0	1	0	RandD	low
1461	0.42	0.53	2	142	3	0	1	0	sales	low

While a formal likelihood analysis could be conducted using Bayes' theorem, the presence of multiple continuous variables across 10 columns renders the probability of legitimate identical responses across all metrics for distinct employees extremely low. Given this negligible probability, proceeding with the removal of duplicates is strongly recommended to safeguard data integrity and ensure the validity of subsequent analyses.

In [29]:
```
df1 = df.drop_duplicates(keep='first')
```

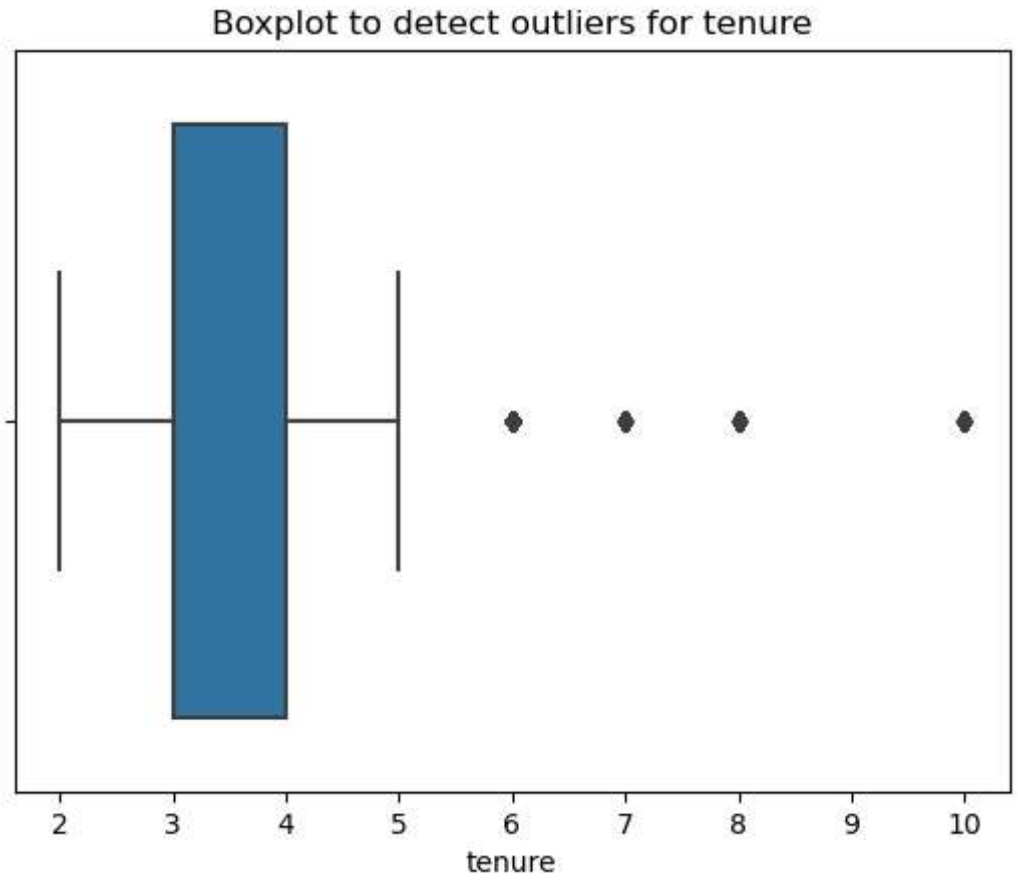
Check Outliers

In [30]:
```
df1.describe()
```

Out[30]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_accident	left	promotion_last_5years
count	11991.000000	11991.000000	11991.000000	11991.000000	11991.000000	11991.000000	11991.000000	11991.000000
mean	0.629658	0.716683	3.802852	200.473522	3.364857	0.154282	0.166041	0.016929
std	0.241070	0.168343	1.163238	48.727813	1.330240	0.361234	0.372133	0.129012
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000	0.000000	0.000000
25%	0.480000	0.570000	3.000000	157.000000	3.000000	0.000000	0.000000	0.000000
50%	0.660000	0.720000	4.000000	200.000000	3.000000	0.000000	0.000000	0.000000
75%	0.820000	0.860000	5.000000	243.000000	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

In [31]:
```
sns.boxplot(x=df1['tenure'])
plt.title('Boxplot to detect outliers for tenure')
plt.show()
```



The boxplot above shows that there are outliers in the tenure variable.

It would be helpful to investigate how many rows in the data contain outliers in the tenure column.

In [32]:
```
percentile25 = df1['tenure'].quantile(0.25)
percentile75 = df1['tenure'].quantile(0.75)

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

Out[32]:
```
1.0
```

In [33]:
```
# Define the upper limit and lower limit for non-outlier values in `tenure`
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr
print("Lower limit:", lower_limit)
print("Upper limit:", upper_limit)
```

```
Lower limit: 1.5
Upper limit: 5.5
```

In [34]:
```
outliers = df1[(df1['tenure'] > upper_limit) | (df1['tenure'] < lower_limit)]
print("Number of rows in the data containing outliers in `tenure`:", len(outliers))
```

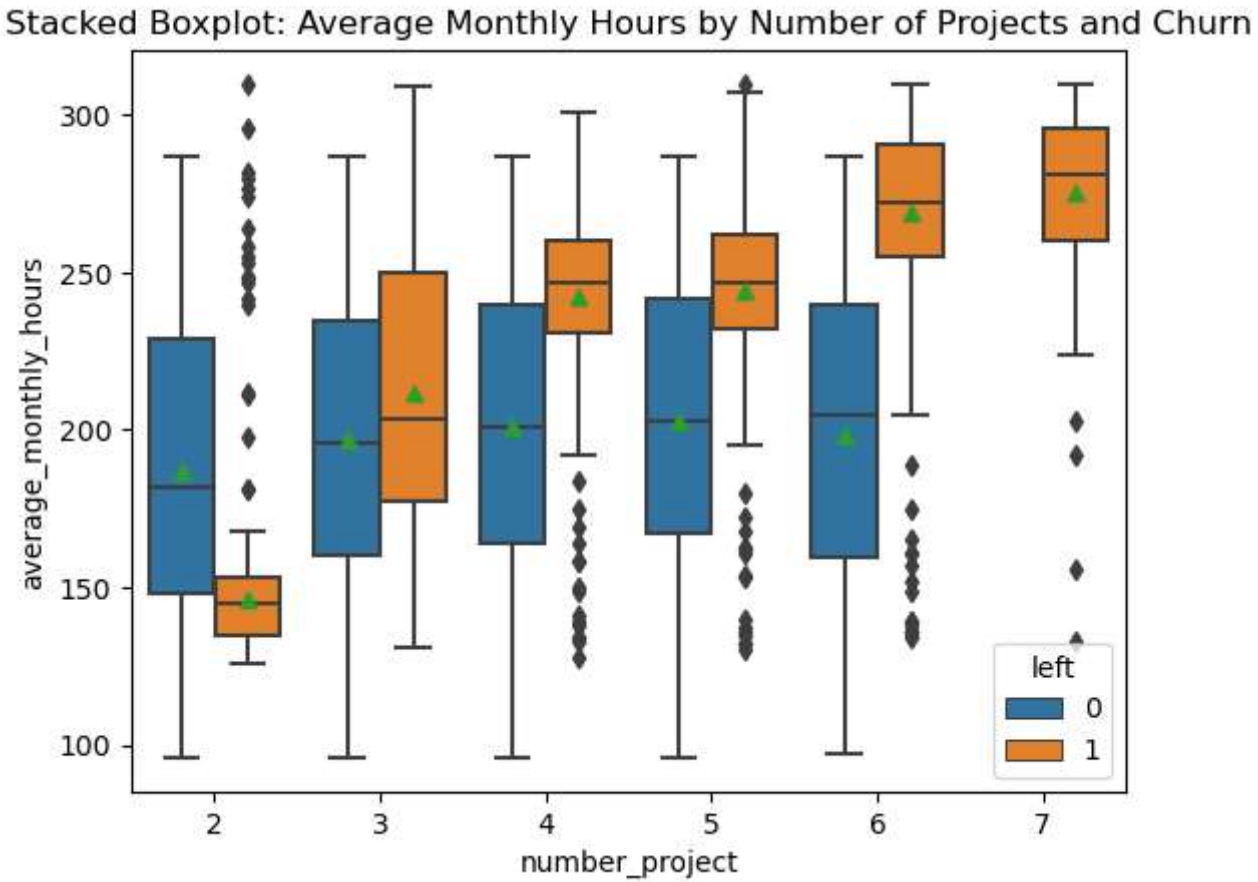
```
Number of rows in the data containing outliers in `tenure`: 824
```

Data Exploration (Continue EDA)

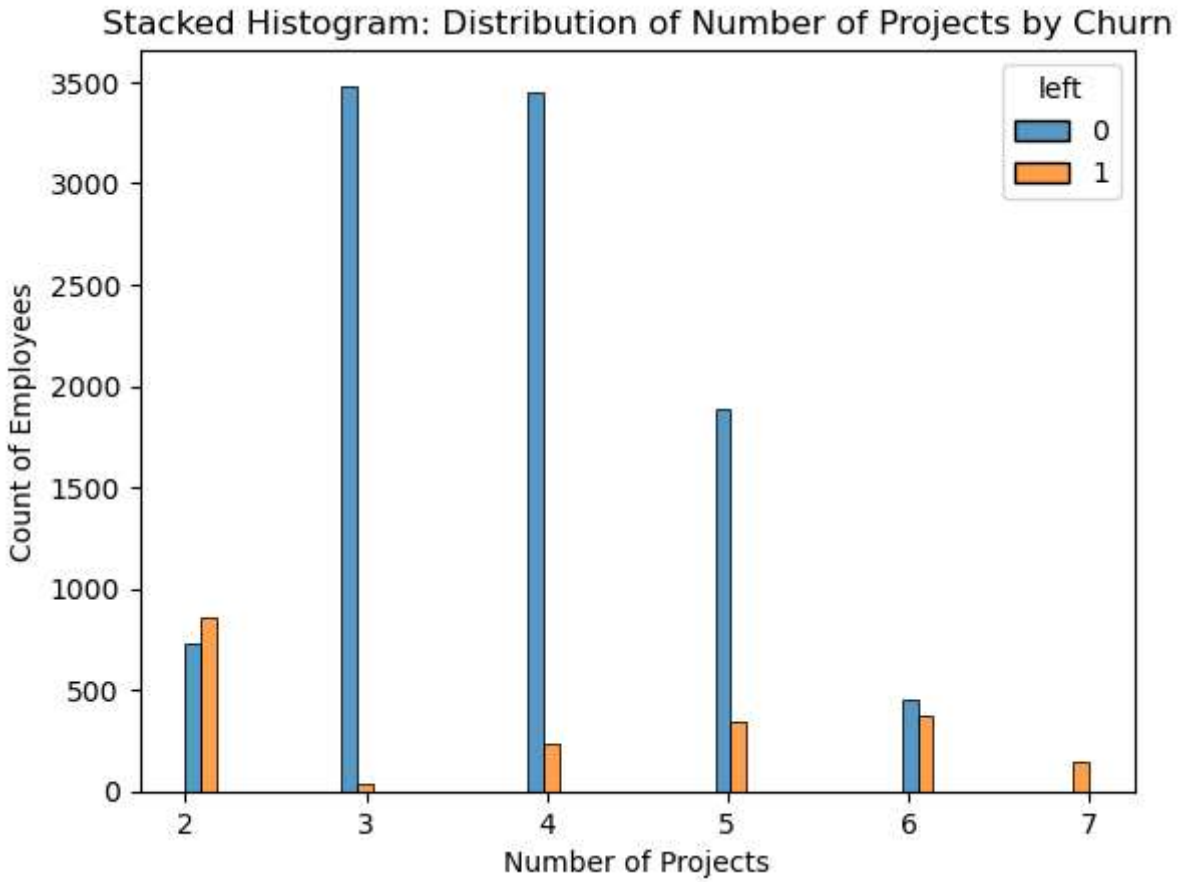
In [35]:
```
print(df1['left'].value_counts())
print()
print(df1['left'].value_counts(normalize=True))
```

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

left
0    0.833959
1    0.166041
Name: proportion, dtype: float64
```

In [36]:
```
sns.boxplot(
    x="number_project",
    y="average_monthly_hours",
    hue="left", # Specifies that the data should be grouped by the "left" variable,
    showmeans=True, # Show mean values per group
    data=df1,
)
plt.title("Stacked Boxplot: Average Monthly Hours by Number of Projects and Churn")
plt.show()
```



In [37]:
```
sns.histplot(
    x="number_project",
    hue="left",
    multiple="dodge", # Stack bars while avoiding overlap
    data=df1,
)
plt.title("Stacked Histogram: Distribution of Number of Projects by Churn")
plt.xlabel("Number of Projects")
plt.ylabel("Count of Employees")
plt.show()
```



We looked at employees' work hours based on how many projects they handled. Turns out, folks with more projects generally clocked more hours, no surprise there

`But there's a twist: Two groups of leavers stood out:`

- Group A: Worked way less than their peers with the same workload. Maybe they got fired, or maybe they were coasting on their way out.
- Group B: Burned the midnight oil, possibly carrying the team on their back. These are likely folks who quit, probably feeling overworked or unsatisfied.

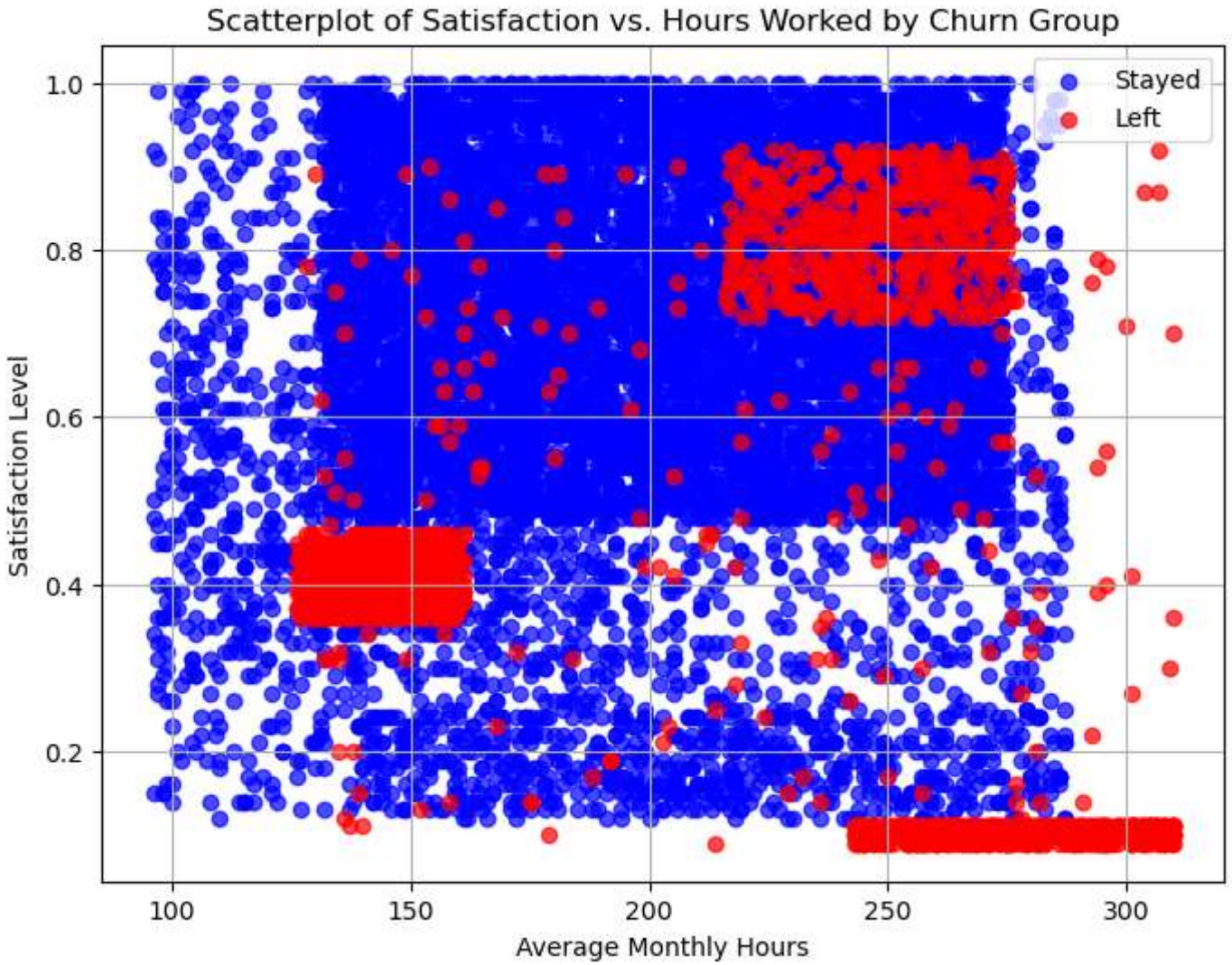
everyone with seven projects left the company. And for those juggling six projects, the workload was HUGE. Looks like the sweet spot might be 3-4 projects, where folks were more likely to stay put.

In [38]:
```
df1[df1['number_project']==7]['left'].value_counts()
```

Out[38]:
```
left
1    145
Name: count, dtype: int64
```

This confirms that all employees with 7 projects did leave.

In [39]:
```
# Create scatterplot of `average_monthly_hours` versus `satisfaction_level`, comparing employees who stayed versus those who left


# Separate data by churn group
stayed = df1[df1['left'] == 0]
left = df1[df1['left'] == 1]

# Create the scatterplot
plt.figure(figsize=(8, 6))

# Plot stayed employees as blue circles
plt.scatter(stayed['average_monthly_hours'], stayed['satisfaction_level'], c='blue', alpha=0.7, label='Stayed')

# Plot left employees as red circles
plt.scatter(left['average_monthly_hours'], left['satisfaction_level'], c='red', alpha=0.7, label='Left')

# Add labels and title
plt.xlabel('Average Monthly Hours')
plt.ylabel('Satisfaction Level')
plt.title('Scatterplot of Satisfaction vs. Hours Worked by Churn Group')

# Add legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



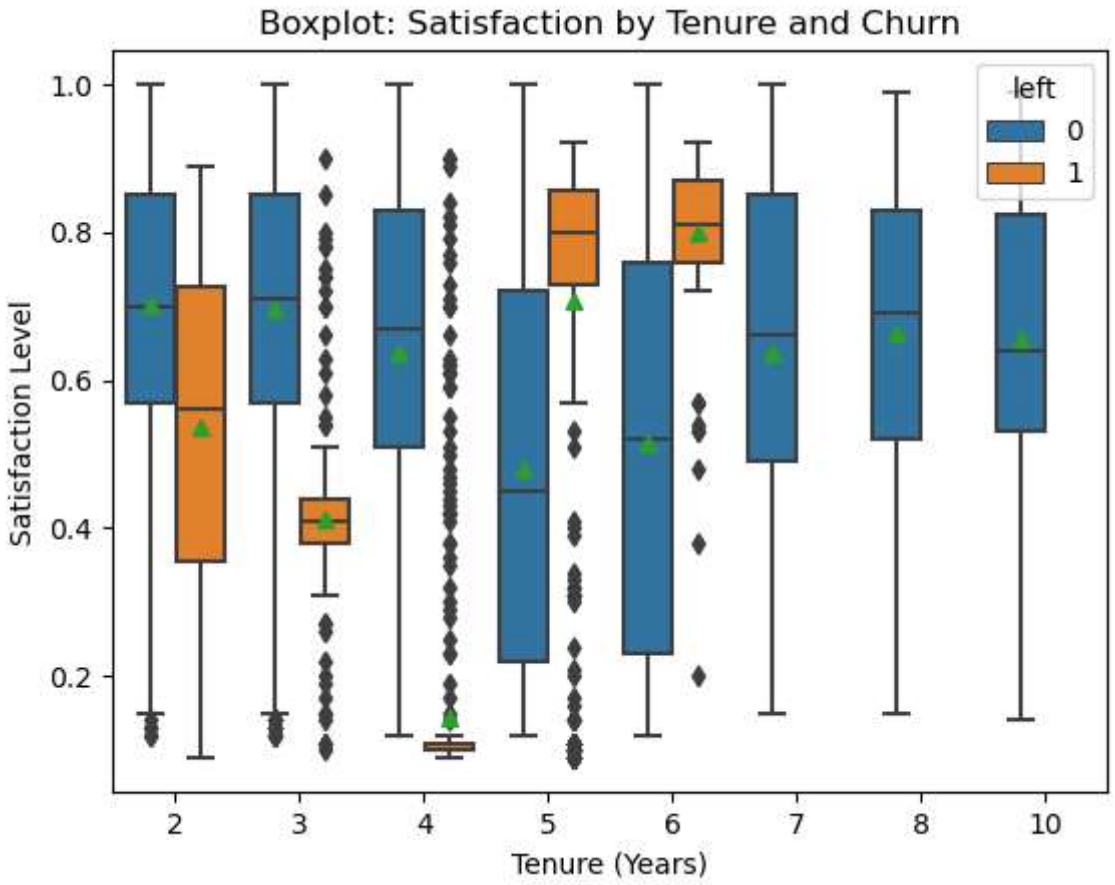
Long Hours, Low Satisfaction:

- Lots of folks (one big group) worked crazy hours, like 75 hours per week all year long! No wonder their happiness was almost non-existent.
- Others worked "normal" hours but were only about half happy. Maybe they felt pressure to work like their co-workers who did way too much, dragging down their mood.

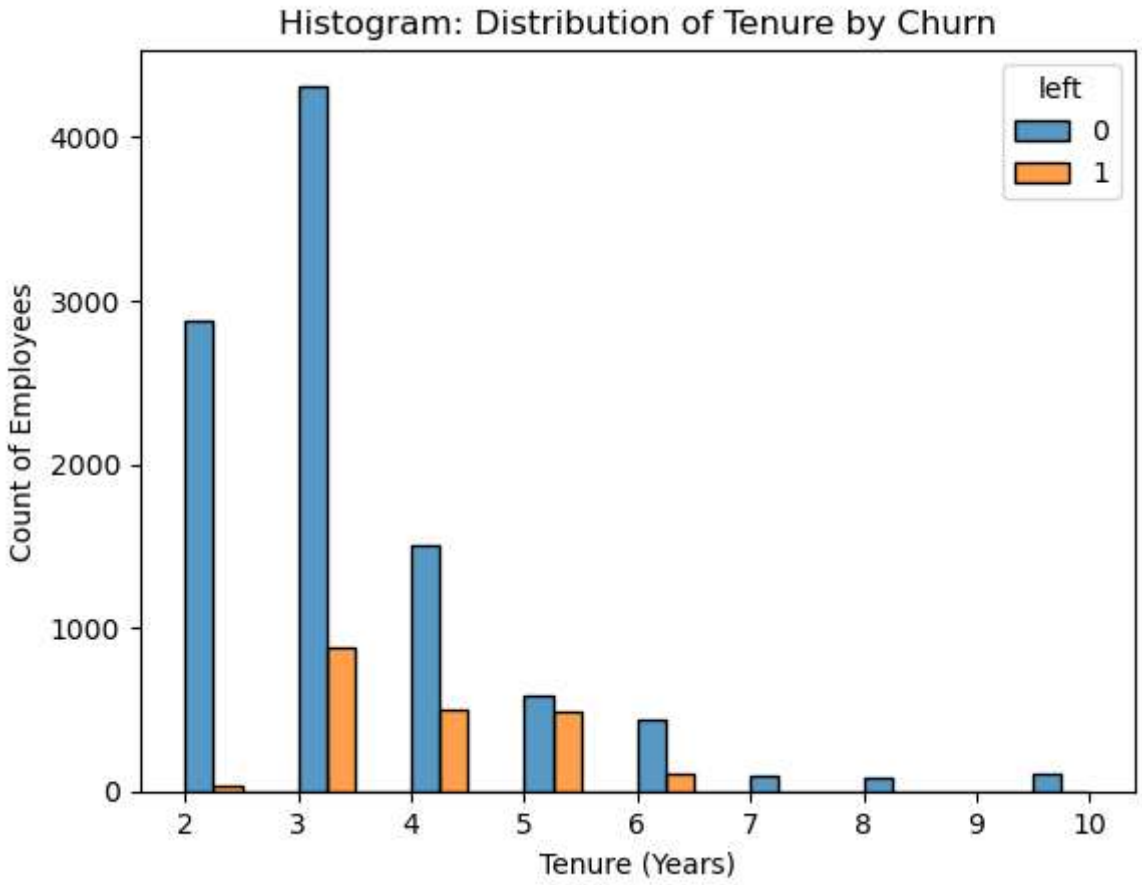
there is a group who worked ~210–280 hours per month, and they had satisfaction levels ranging ~0.7–0.9.(seemed happy)

Something strange is going on with the data - like someone messed with it or made it up. The way it's grouped is odd, which makes it hard to trust these findings.

In [40]:
```
# Create boxplot showing distributions of `satisfaction_level` by tenure, comparing employees who stayed versus those who left


sns.boxplot(
    x="tenure",
    y="satisfaction_level",
    hue="left",
    showmeans=True, # Show mean values per group
    data=df1,
)
plt.title("Boxplot: Satisfaction by Tenure and Churn")
plt.xlabel("Tenure (Years)")
plt.ylabel("Satisfaction Level")
plt.show()
```



In [41]:
```
sns.histplot(
    x="tenure",
    hue="left",
    multiple="dodge",# Stack bars while avoiding overlap
    bins =16, # adjust width of bar
    data=df1,
)
plt.title("Histogram: Distribution of Tenure by Churn")
plt.xlabel("Tenure (Years)")
plt.ylabel("Count of Employees")
plt.show()
```



Two main groups of folks quit:

- Unhappy with the job, but haven't been there long.
- Super happy, but spent some time at the company already.

There's something weird with the four-year folks who left. They were WAY less happy than everyone else. Maybe something changed around that time that pushed them out?

The histogram shows there aren't many employees who've been there a long time

In [42]:
```
df1.groupby('left')['satisfaction_level'].agg([np.mean,np.median])
```

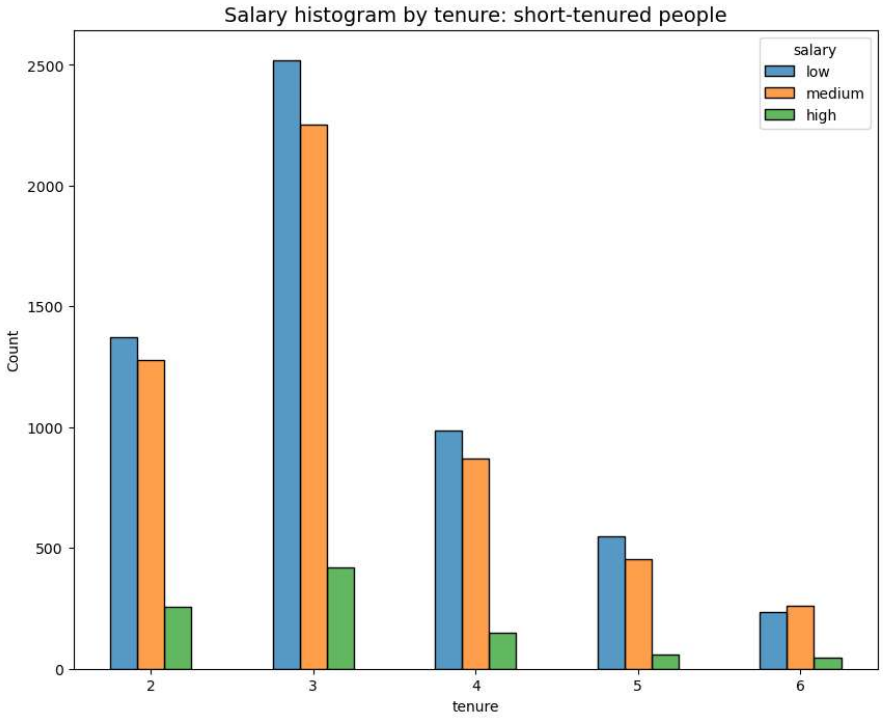
Out[42]:

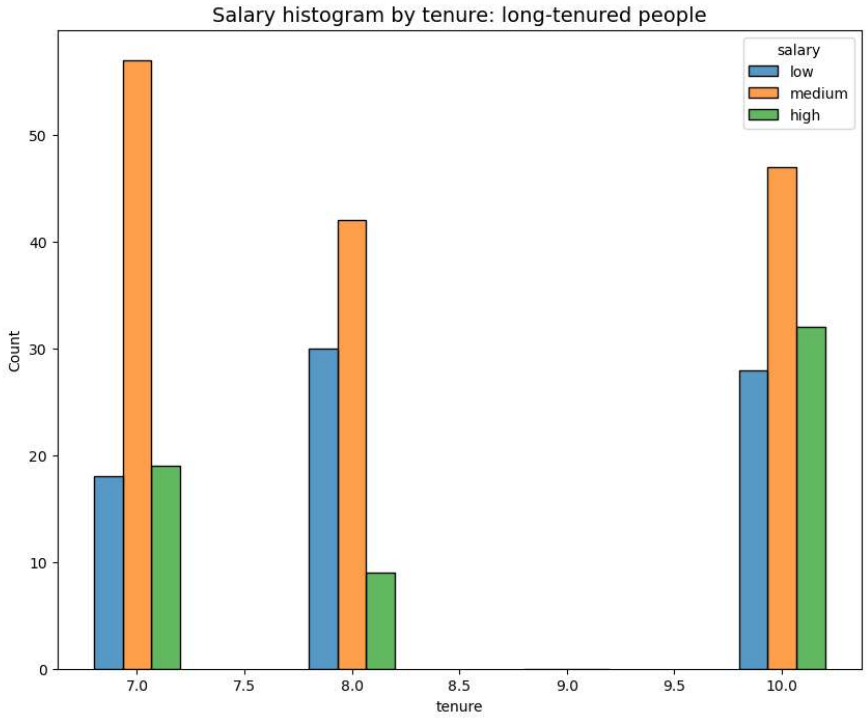
left	mean	median
0	0.667365	0.69
1	0.440271	0.41

Lower Satisfaction for Leavers: Both the mean and median scores for folks who left were lower than for those who stayed. This means, on average, they were less happy with their jobs.

In [43]:

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

tenure_short = df1[df1['tenure'] < 7]
tenure_long = df1[df1['tenure'] > 6]

sns.histplot(data=tenure_short,  # Use data for short-tenured employees
    x='tenure',  # Show tenure on the x-axis
    hue='salary',  # Color-code bars based on salary
    discrete=1,  # Treat tenure as categories (not continuous)
    hue_order=['low', 'medium', 'high'],  # Arrange salary categories in this order
    multiple='dodge',  # Stack bars side-by-side
    shrink=.5,  # Adjust bar width
    ax=ax[0])  # Place this plot on the first part of the canvas
ax[0].set_title('Salary histogram by tenure: short-tenured people', fontsize='14')  # Add a title


sns.histplot(data=tenure_long, x='tenure', hue='salary', discrete=1,
             hue_order=['low', 'medium', 'high'], multiple='dodge',
             shrink=.4, ax=ax[1])
ax[1].set_title('Salary histogram by tenure: long-tenured people', fontsize='14');
```





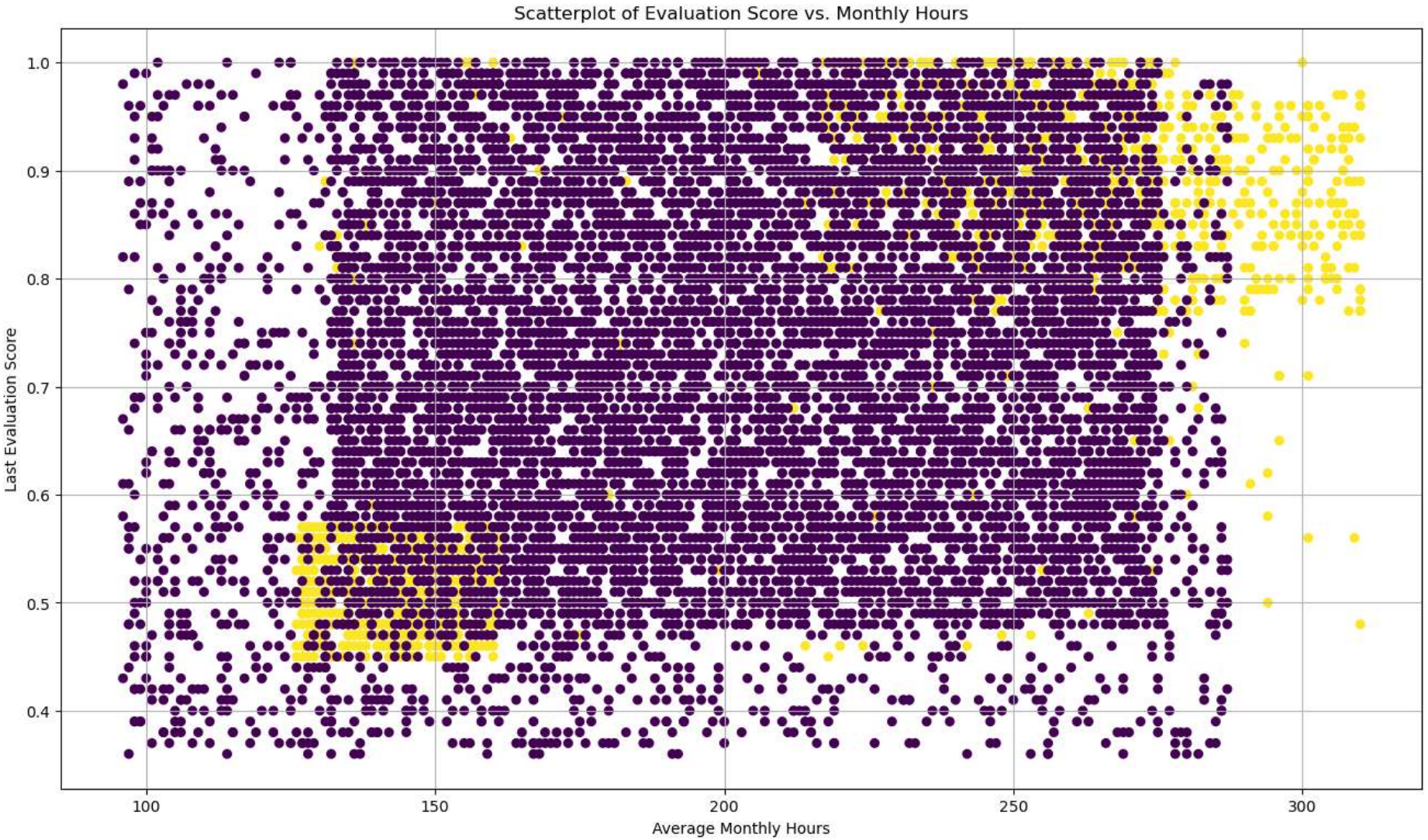
Long time at the company doesn't automatically mean high salary

If long-tenured employees were mostly earning the highest salaries, the graph would have looked like a mountain, with lots of folks up top on the high-salary side.

Of course, this doesn't mean there's no connection between time at the company and salary. Some people might get raises or promotions over time, moving up the pay scale. But the graphs show that it's not a simple cause-and-effect thing. Other factors, like job performance, skills, or even just being in the right place at the right time, likely play a role too.

In [44]:
```
# Create scatterplot of `average_monthly_hours` versus `last_evaluation`

plt.figure(figsize=(16, 9))

plt.scatter(x='average_monthly_hours', y='last_evaluation', data=df1, c='left', s=30)  # Corrected argument order

plt.xlabel('Average Monthly Hours')
plt.ylabel('Last Evaluation Score')
plt.title('Scatterplot of Evaluation Score vs. Monthly Hours')
```

```
plt.grid(True)
plt.show()
```



`Two types of employees who left:`

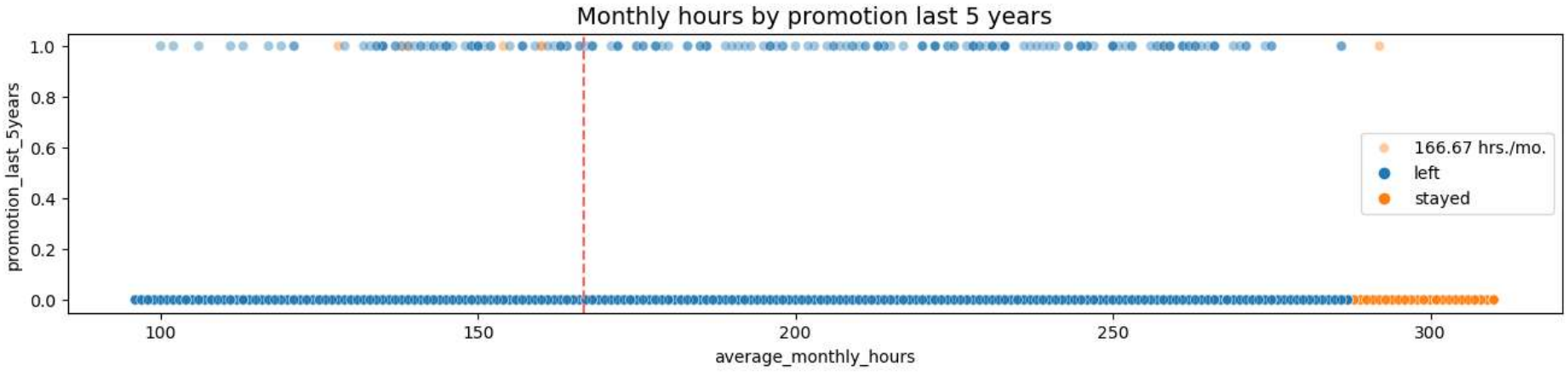
- Hard workers with good scores: Some employees put in tons of hours (more than average) and still got great evaluations. But they decided to leave anyway.

- Underworked with low scores: Other employees worked a bit less and had lower evaluation scores. Maybe they weren't happy with their performance or the job itself.

Most work overtime: The plot shows most employees in the company put in well over 167 hours per month (the average). That's quite a lot!

In [45]:
```
# Create plot to examine relationship between `average_monthly_hours` and `promotion_last_5years`
plt.figure(figsize=(16, 3))
sns.scatterplot(data=df1, x='average_monthly_hours', y='promotion_last_5years', hue='left', alpha=0.4)
plt.axvline(x=166.67, color='#ff6361', ls='--')
# color='#ff6361': Make the line red-orange.
# ls='--': Use a dashed line style.

plt.legend(labels=['166.67 hrs./mo.', 'left', 'stayed'])
plt.title('Monthly hours by promotion last 5 years', fontsize='14')
```

Out[45]:
```
Text(0.5, 1.0, 'Monthly hours by promotion last 5 years')
```



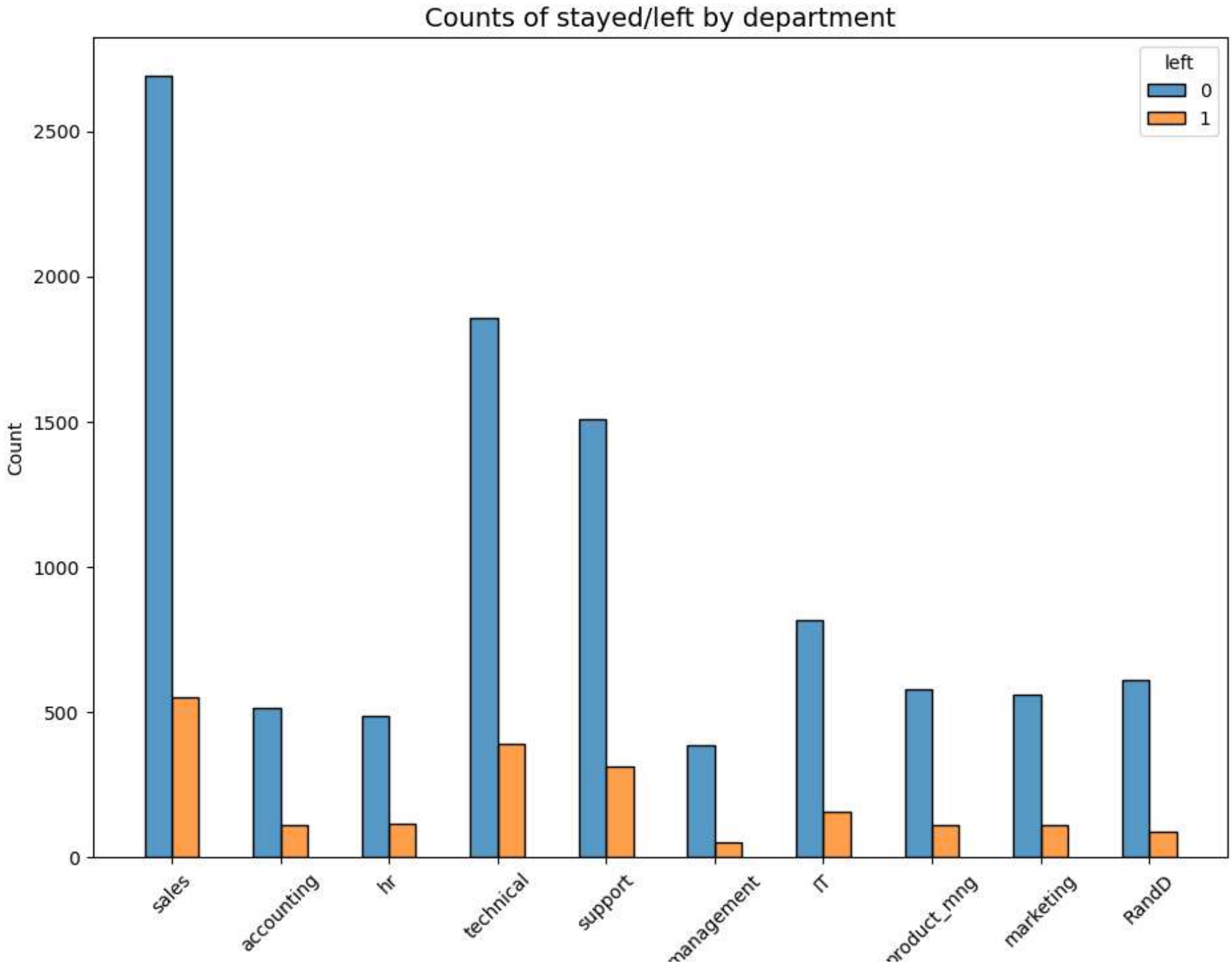
- very few employees who were promoted in the last five years left
- very few employees who worked the most hours were promoted
- all of the employees who left were working the longest hours

In [46]:
```
df1['department'].value_counts()
```

Out[46]:
```
department
sales          3239
technical      2244
support        1821
IT              976
RandD           694
product_mng     686
marketing       673
accounting      621
hr              601
management      436
Name: count, dtype: int64
```

In [47]:
```
plt.figure(figsize=(11, 8))
sns.histplot(
    data=df1,  x='department',  hue='left',discrete=1,  # Treat the x-axis as discrete categories (since departments are categori
    hue_order=[0, 1],  # Specify the order of colors in the legend (0 represents 'stayed', 1 represents 'left').
    multiple='dodge', shrink=.5  # Adjust the width of bars to avoid overlapping.
)
plt.xticks(rotation=45)  # Rotate department names for better readability.
plt.title('Counts of stayed/left by department', fontsize=14)  # Set the title of the plot.
```

Out[47]:
```
Text(0.5, 1.0, 'Counts of stayed/left by department')
```


Counts of stayed/left by department
left
0
1
Count
2500
2000
1500
1000
500
0
sales
accounting
hr
technical
support
management
IT
product_mng
marketing
RandD

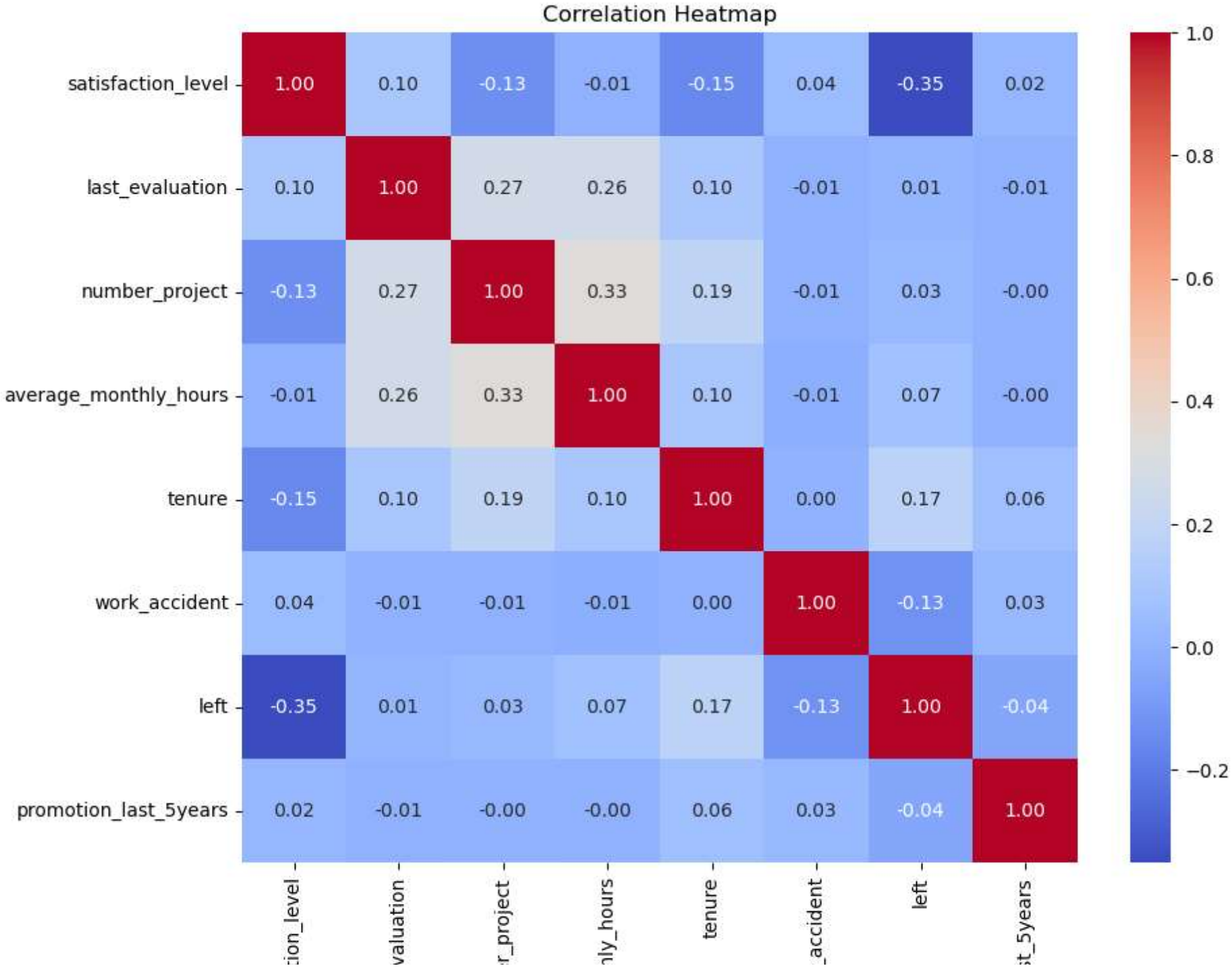
department

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

In [48]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 11991 entries, 0 to 11999
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   satisfaction_level     11991 non-null  float64
 1   last_evaluation        11991 non-null  float64
 2   number_project         11991 non-null  int64  
 3   average_monthly_hours  11991 non-null  int64  
 4   tenure                 11991 non-null  int64  
 5   work_accident          11991 non-null  int64  
 6   left                   11991 non-null  int64  
 7   promotion_last_5years  11991 non-null  int64  
 8   department             11991 non-null  object 
 9   salary                 11991 non-null  object 
dtypes: float64(2), int64(6), object(2)
memory usage: 1.0+ MB
```

In [49]:
```
# Correlation works mainly with numeric data

corr_matrix = df1.select_dtypes(include=['number']).corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
satisfaction_level
last_evaluation
number_project
average_monthly_hours
tenure
work_accident
left
promotion_last_5years
1.00 0.10 -0.13 -0.01 -0.15 0.04 -0.35 0.02
0.10 1.00 0.27 0.26 0.10 -0.01 0.01 -0.01
-0.13 0.27 1.00 0.33 0.19 -0.01 0.03 -0.00
-0.01 0.26 0.33 1.00 0.10 -0.01 0.07 -0.00
-0.15 0.10 0.19 0.10 1.00 0.00 0.17 0.06
0.04 -0.01 -0.01 -0.01 0.00 1.00 -0.13 0.03
-0.35 0.01 0.03 0.07 0.17 -0.13 1.00 -0.04
0.02 -0.01 -0.00 -0.00 0.06 0.03 -0.04 1.00
tion_level
valuation
r_project
nly_hours
tenure
_accident
left
t_5years
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Number of projects, monthly hours, and evaluation scores: These variables likely indicate higher workload and performance, which positively correlate with each other. This suggests that employees who take on more projects tend to work longer hours and might receive higher evaluations as a result.

A value of -0.35 indicates a moderate negative correlation, meaning that as satisfaction level increases, the likelihood of an employee leaving decreases, and vice versa.

Insights

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

Choosing Model

Logistic Regression Model

suits the task because it involves binary classification.

Before splitting the data, encode the non-numeric variables. There are two: `department` and `salary`.

`department` is a categorical variable, which means you can dummy it for modeling.

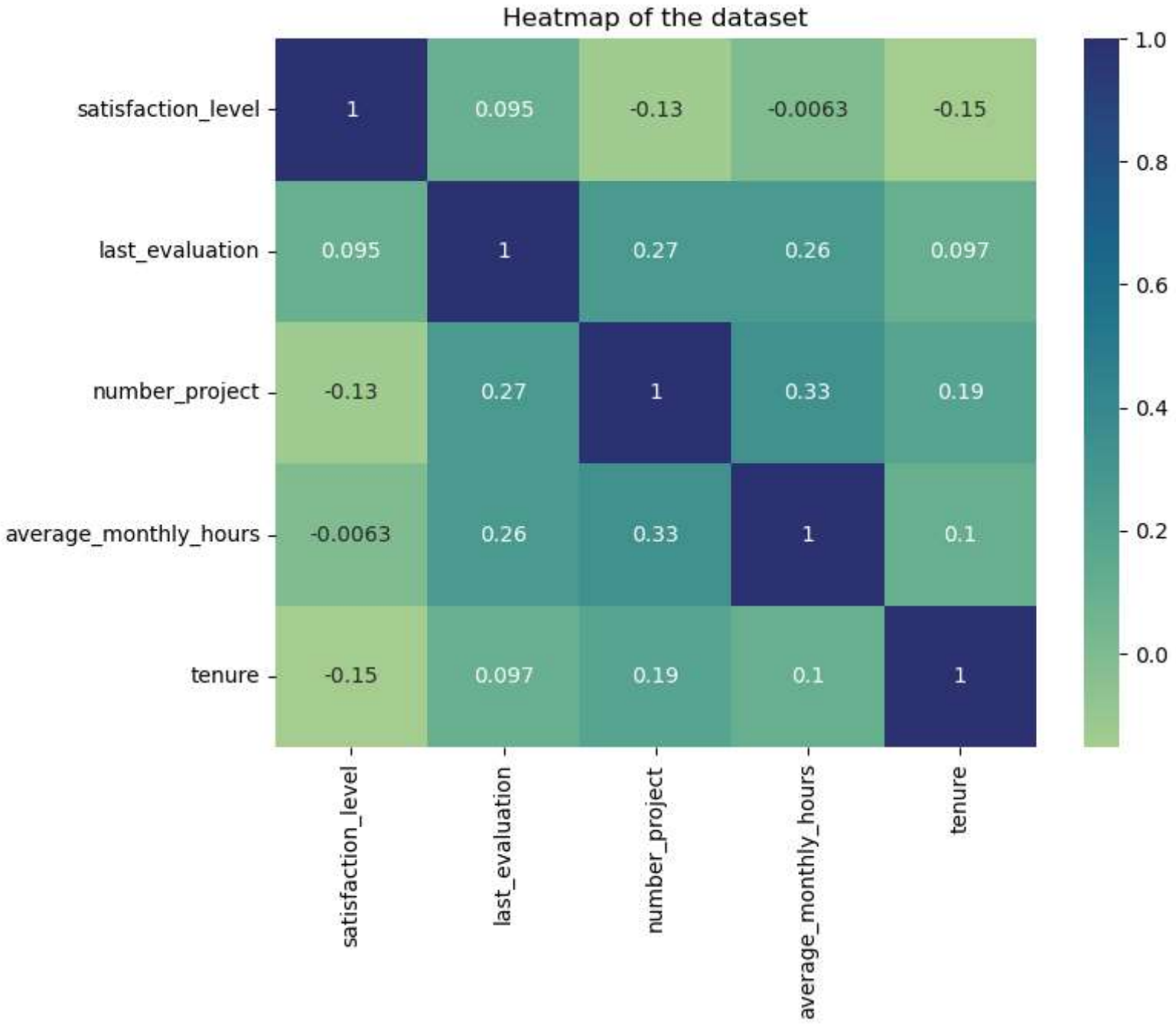
`salary` is categorical too, but it's ordinal. There's a hierarchy to the categories so it's better not to dummy this column, but rather to convert the levels to number

In [50]:
```
df2 = df1.copy()

df2['salary']= (
    df2['salary'].astype('category')
    .cat.set_categories(['low','medium','high'])
    .cat.codes)
# .cat.codes assigns numerical codes low 0 medium 1 high 2

df2 = pd.get_dummies(df2,drop_first=False)

df2.head()
```

Out[50]:

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

In [51]:
```
plt.figure(figsize=(8,6))
sns.heatmap(df2[['satisfaction_level', 'last_evaluation', 'number_project', 'average_monthly_hours', 'tenure']].corr(),
            annot = True ,cmap ='crest')
plt.title('Heatmap of the dataset')
plt.show()
```


Heatmap of the dataset
satisfaction_level
last_evaluation
number_project
average_monthly_hours
tenure
1
0.095
-0.13
-0.0063
-0.15
0.095
1
0.27
0.26
0.097
-0.13
0.27
1
0.33
0.19
-0.0063
0.26
0.33
1
0.1
-0.15
0.097
0.19
0.1
1
satisfaction_level
last_evaluation
number_project
average_monthly_hours
tenure
1.0
0.8
0.6
0.4
0.2
0.0

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.

In [52]:
```
df3 = df2[(df2['tenure']>= lower_limit) & (df2['tenure']<= upper_limit)]
df3.head()
```

Out[52]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_accident	left	promotion_last_5years	salary	department_IT
0	0.38	0.53	2	157	3	0	1	0	0	False
2	0.11	0.88	7	272	4	0	1	0	1	False
3	0.72	0.87	5	223	5	0	1	0	0	False
4	0.37	0.52	2	159	3	0	1	0	0	False
5	0.41	0.50	2	153	3	0	1	0	0	False

In [53]:
```
y = df3['left']

X = df3.drop('left',axis =1)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.25,stratify = y ,random_state = 42)
```

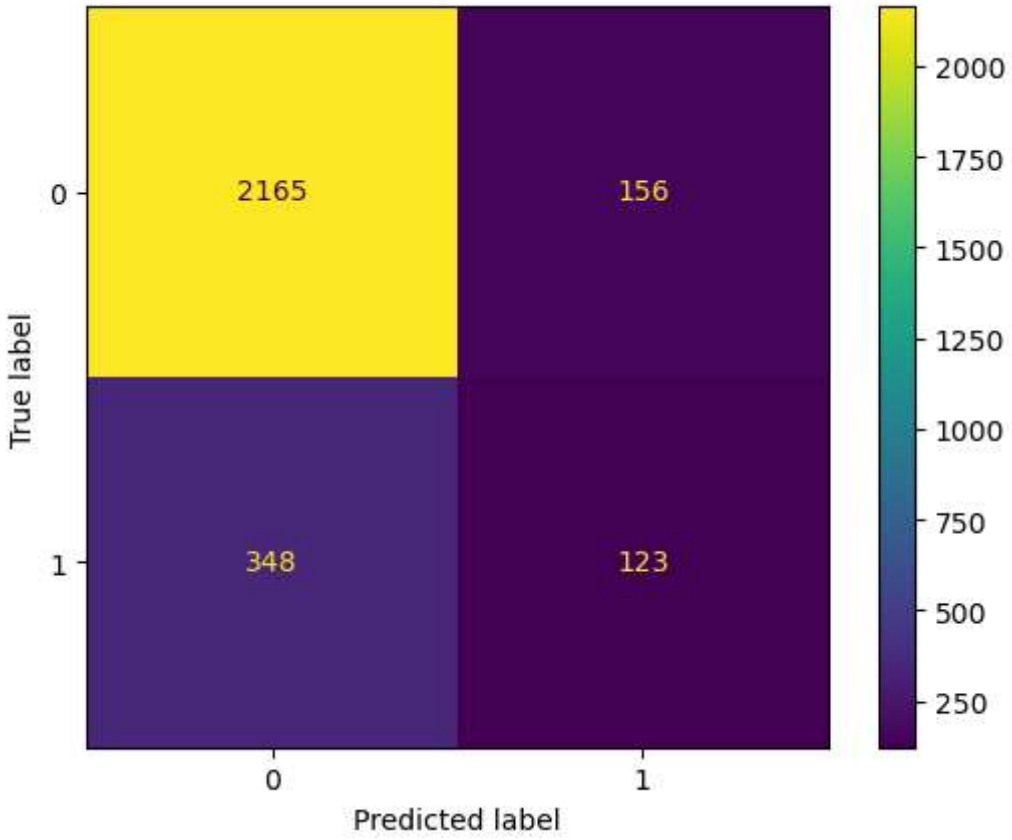
In [54]:
```
log_clf = LogisticRegression(random_state=42, max_iter=500).fit(X_train, y_train)
```

In [55]:
```
y_pred = log_clf.predict(X_test)
y_pred
```

Out[55]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [56]:
```
log_cm = confusion_matrix(y_test, y_pred, labels=log_clf.classes_)

log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm,
                                  display_labels=log_clf.classes_)
log_disp.plot()
```

Out[56]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f2c0bd78d0>
```



In [57]:
```
log_cm
```

Out[57]:
```
array([[2165,  156],
       [ 348,  123]], dtype=int64)
```

- `True Negatives (TN):` 2165 cases were correctly predicted to belong to the "Stayed" class.
- `False Positives (FP):` 156 cases were incorrectly predicted to belong to the "Left" class, but they actually belonged to the "Stayed" class.
- `False Negatives (FN):` 348 cases were incorrectly predicted to belong to the "Stayed" class, but they actually belonged to the "Left" class.
- `True Positives (TP):` 123 cases were correctly predicted to belong to the "Left" class.

In [58]:
```
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Recall:", recall)
print("Precision:", precision)
print("F1-score:", f1)
```

```
Accuracy: 0.8194842406876791
Recall: 0.2611464968152866
Precision: 0.44086021505376344
F1-score: 0.328
```

Accuracy: The model correctly predicts the outcome (whether an employee stays or leaves) in 81.9% of cases.

Recall: The model correctly identifies only 26.1% of employees who actually left.This suggests a significant number of false negatives (employees who left but were predicted to stay).

Precision: When the model predicts an employee will leave, it's correct 44.1% of the time. This indicates a moderate level of precision, with some false positives (employees predicted to leave who actually stayed).

In churn prediction, recall might be more critical than precision, as you'd want to prioritize identifying potential leavers even if it means some false positives.

In [59]:
```
# Create classification report for logistic regression model
target_names = ['Predicted would not leave','Predicted would leave']
print(classification_report(y_test,y_pred,target_names = target_names))
```

```
                          precision    recall  f1-score   support

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

                 accuracy                           0.82      2792
                macro avg       0.65      0.60      0.61      2792
             weighted avg       0.79      0.82      0.80      2792
```

The model performs better at predicting employees who stay than those who leave.

The relatively low recall for "Predicted would leave" suggests potential missed opportunities to identify employees at risk of churning.

This might be due to class imbalance (fewer employees leaving in the dataset), making it harder for the model to learn to identify those cases.

Tree-Based Model

In [60]:
```
y = df2['left']
X = df2.drop('left',axis = 1)

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

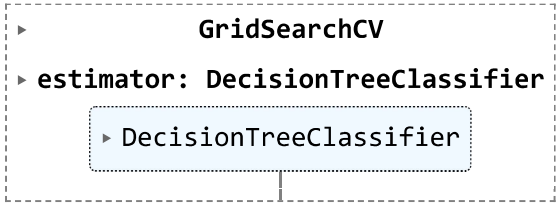
Decision Tree

In [61]:
```
tree = DecisionTreeClassifier(random_state=0)
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }
scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

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

In [62]:
```
%%time

tree1.fit(X_train,y_train)
```

```
CPU times: total: 10.1 s
Wall time: 13.1 s
```

Out[62]:



In [63]:
```
tree1.best_params_
```

Out[63]:
```
{'max_depth': 4, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

In [64]:
```
tree1.best_score_
```

Out[64]:
```
0.969819392792457
```

This is a strong AUC score, which shows that this model can predict employees who will leave very well. It suggests that the model can effectively distinguish between employees who stay and those who leave

In [65]:
```
def make_results(model_name:str,model_object,metric:str):
    metric_dict = {'auc': 'mean_test_roc_auc',
                   'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy'
                  }
    cv_results = pd.DataFrame(model_object.cv_results_)

    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].idxmax(), :]

    auc = best_estimator_results.mean_test_roc_auc
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy

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

    return table
```

In [66]:
```
tree1_cv_results = make_results('decision tree cv', tree1, 'auc')
tree1_cv_results
```

Out[66]:

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

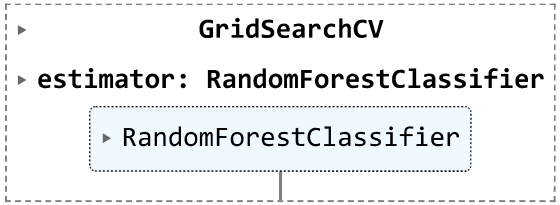
Random Forest

In [67]:
```
rf = RandomForestClassifier(random_state=0)

cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

# 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')
```

In [68]:
```
%%time
rf1.fit(X_train,y_train)
```

```
CPU times: total: 50min 41s
Wall time: 1h 2min 14s
```

Out[68]:



In [69]:
```
filename = "C:/Users/Pratik Sonawane/Videos/data anayst certificate/Machine Learning/rf1_model.pkl"
pickle.dump(rf1, open(filename, 'wb'))
```

In [70]:
```
loaded_model = pickle.load(open(filename, 'rb'))
```

In [71]:
```
loaded_model.best_score_
```

Out[71]:
```
0.9804250949807172
```

In [72]:
```
rf1_cv_results = make_results('random forestcv',rf1,'auc')
print(tree1_cv_results)
print()
print(rf1_cv_results)
```

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

             model  precision    recall        F1  accuracy       auc
0  random forestcv   0.950023  0.915614  0.932467  0.977983  0.980425
```

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.

In [73]:
```
def get_scores(model_name:str, model, X_test_data, y_test_data):

    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
```

In [74]:
```
rf1_test_scores = get_scores('random forest1 test',rf1,X_test,y_test)
rf1_test_scores
```

Out[74]:

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

The test scores are very similar to the validation scores, which is good. This appears to be a strong model.

`rf1_cv`

- Data Used: Cross-validation results, where the model is trained and evaluated multiple times on different folds of the training data.
- Purpose: To estimate model performance during model tuning and selection, ensuring it generalizes well to unseen data.
- Metrics: Provides average performance scores across cross-validation folds, giving a more robust assessment of generalization ability.

`rf1_test`

- Data Used: Results on the independent testing set, which has not been used for training or hyperparameter tuning.
- Purpose: To assess the final model's performance on truly unseen data, reflecting its real-world effectiveness.
- Metrics: Provides scores on the testing set, indicating how well the model generalizes to new, previously unseen data.

`rf_cv is like taking several practice test to estimate result in final exam`

`rf_test is like taking one big final test`

If the practice tests and final exam scores are similar, that's great! The model can probably handle real-world situations just like it handled the practice ones.

Potential data leakage impacting model performance

The high evaluation scores achieved by the initial models warrant further investigation due to the potential for data leakage. Leakage occurs when data used in training is not representative of the data the model will encounter in production, leading to overly optimistic performance evaluations.

`Two potential sources of leakage have been identified:`

Missing satisfaction levels: The model may be leveraging information not available in production, such as satisfaction levels potentially missing for some employees. This creates an unrealistic advantage during evaluation.

Average monthly hours as a proxy: The average_monthly_hours feature could be acting as a surrogate for unobserved variables like impending employee departure. While this allows the model to predict churn within the training data, it's unlikely to generalize effectively to production where such information is unavailable.

To address these concerns, the next iteration of model development will incorporate feature engineering and potentially remove problematic features

In [75]:
```
df3 = df2.drop('satisfaction_level',axis = 1)
```

In [76]:
```
df3['overworked']= df2['average_monthly_hours']

print('Max hours:', df3['overworked'].max())
print('Min hours:', df3['overworked'].min())
```

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

166.67 is approximately the average number of monthly hours for someone who works 50 weeks per year, 5 days per week, 8 hours per day.

`.astype(int)` converts all `True` to `1` and all `False`

In [78]:
```
# Define `overworked` as working > 175 hrs/week
df3['overworked']= (df3['overworked']> 175).astype(int)

df3 = df3.drop('average_monthly_hours',axis = 1)
```

Decision tree 2

In [79]:
```
y = df3['left']

X = df3.drop('left',axis = 1)

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

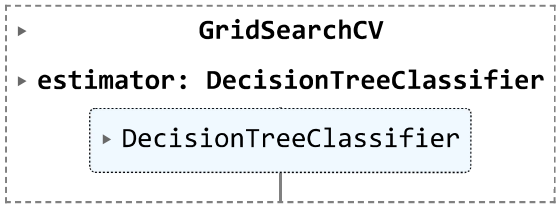
In [84]:
```
tree = DecisionTreeClassifier(random_state = 0)

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

```
            }

scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

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

In [85]:
```
%%time
tree2.fit(X_train,y_train)
```

```
CPU times: total: 3.7 s
Wall time: 5.17 s
```

Out[85]:



In [86]:
```
tree2.best_params_
```

Out[86]:
```
{'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 6}
```

In [87]:
```
tree2.best_score_
```

Out[87]:
```
0.9586752505340426
```

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

In [88]:
```
tree2_cv_results = make_results('decision tree2 cv',tree2 ,'auc')
```

In [89]:
```
print(tree1_cv_results)
print()
print(tree2_cv_results)
```

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

               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.958675
```

Decision Tree 2 exhibits slightly lower precision but higher recall, indicating a greater tendency to correctly identify employees who will actually leave.

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.

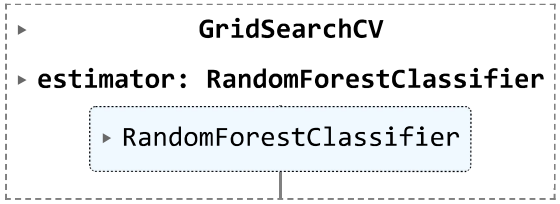
Random Forest 2

In [90]:
```
rf = RandomForestClassifier(random_state=0)
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

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

In [91]:
```
%%time
rf2.fit(X_train, y_train)
```

```
CPU times: total: 17min 33s
Wall time: 24min 28s
```

Out[91]:



In [93]:
```
filename = "C:/Users/Pratik Sonawane/Videos/data anayst certificate/Machine Learning/rf1_model.pkl"
pickle.dump(rf1, open(filename, 'wb'))
```

In [95]:
```
loaded_model2 = pickle.load(open(filename,'rb'))
```

In [96]:
```
rf2.best_params_
```

Out[96]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 300}
```

In [97]:
```
rf2.best_score_
```

Out[97]:
```
0.9648100662833985
```

In [100…
```
rf2_cv_results = make_results('random forest2 cv', rf2, 'auc')
print(tree2_cv_results)
print()
print(rf2_cv_results)
```

```
                model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.958675

                model  precision    recall        F1  accuracy      auc
0  random forest2 cv   0.866758  0.878754  0.872407  0.957411  0.96481
```

In [101…
```
rf2_test_scores = get_scores('random_forest2 test', rf2,X_test,y_test)
rf2_test_scores
```

Out[101]:

	model	precision	recall	f1	accuracy	AUC
0	random_forest2 test	0.870406	0.903614	0.8867	0.961641	0.938407

In [103…
```
preds = rf2.best_estimator_.predict(X_test)
preds
```

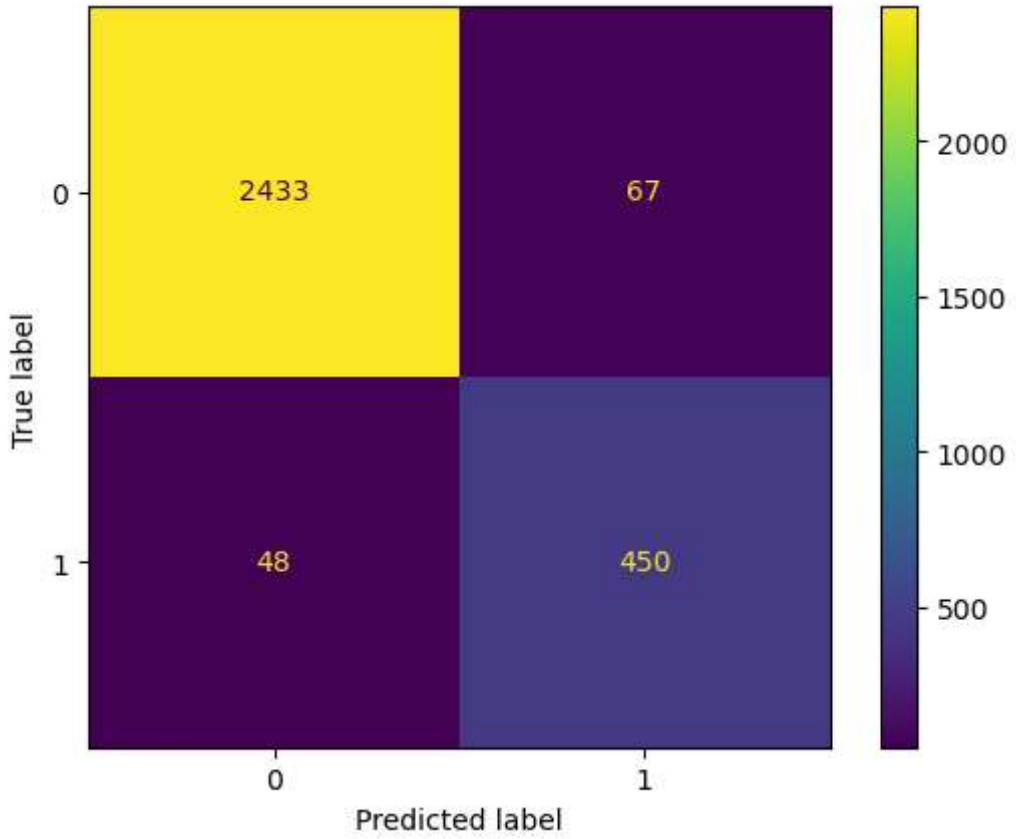
Out[103]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [105…
```
cm = confusion_matrix(y_test, preds, labels=rf2.classes_)
cm
```

Out[105]:
```
array([[2433,   67],
       [  48,  450]], dtype=int64)
```

In [106…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=rf2.classes_)
disp.plot()
```

Out[106]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f2c2536dd0>

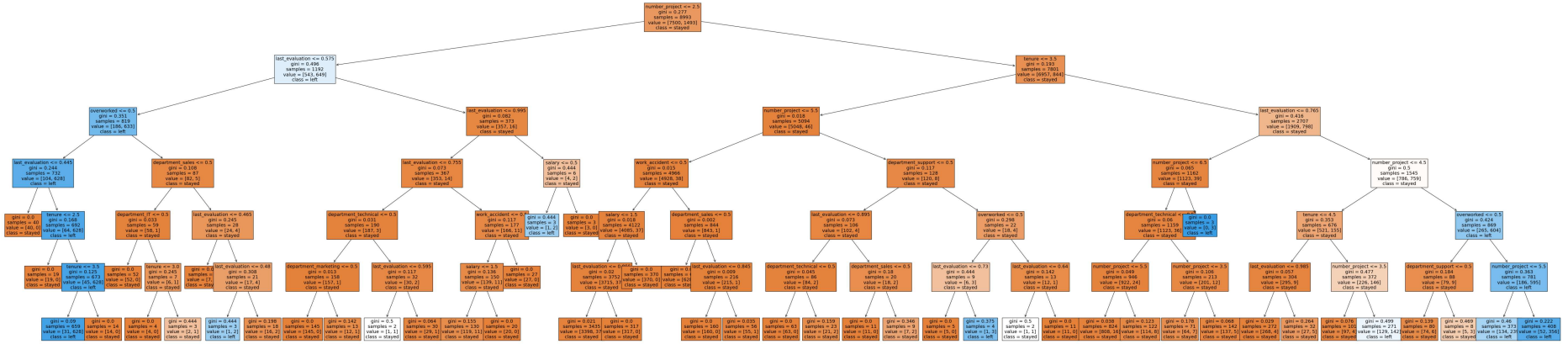


- True Negatives (2433): Correctly predicted employees who did not leave.
- False Positives (67): Incorrectly predicted employees would leave when they actually stayed.
- False Negatives (48): Incorrectly predicted employees would stay when they actually left.
- True Positives (450): Correctly predicted employees who left.

Decision Tree Split

In [112…

```
plt.figure(figsize=(85,20))


plot_tree(tree2.best_estimator_,max_depth = 6 , fontsize = 14,feature_names = list(X.columns),
          class_names = ['stayed','left'],filled=True)
plt.show()
```



Decision Tree Feature Imp

In [114…

```
tree2_imp = pd.DataFrame(tree2.best_estimator_.feature_importances_,
                         columns = ['gini_importances'],
                         index = X.columns)

tree2_imp = tree2_imp.sort_values(by='gini_importances', ascending=False)
tree2_imp
```

Out[114]:

	gini_importances
last_evaluation	0.343958
number_project	0.343385
tenure	0.215681
overworked	0.093498
department_support	0.001142
salary	0.000910
department_sales	0.000607
department_technical	0.000418
work_accident	0.000183
department_IT	0.000139
department_marketing	0.000078
promotion_last_5years	0.000000
department_RandD	0.000000
department_hr	0.000000
department_management	0.000000
department_product_mng	0.000000
department_accounting	0.000000

In [115…

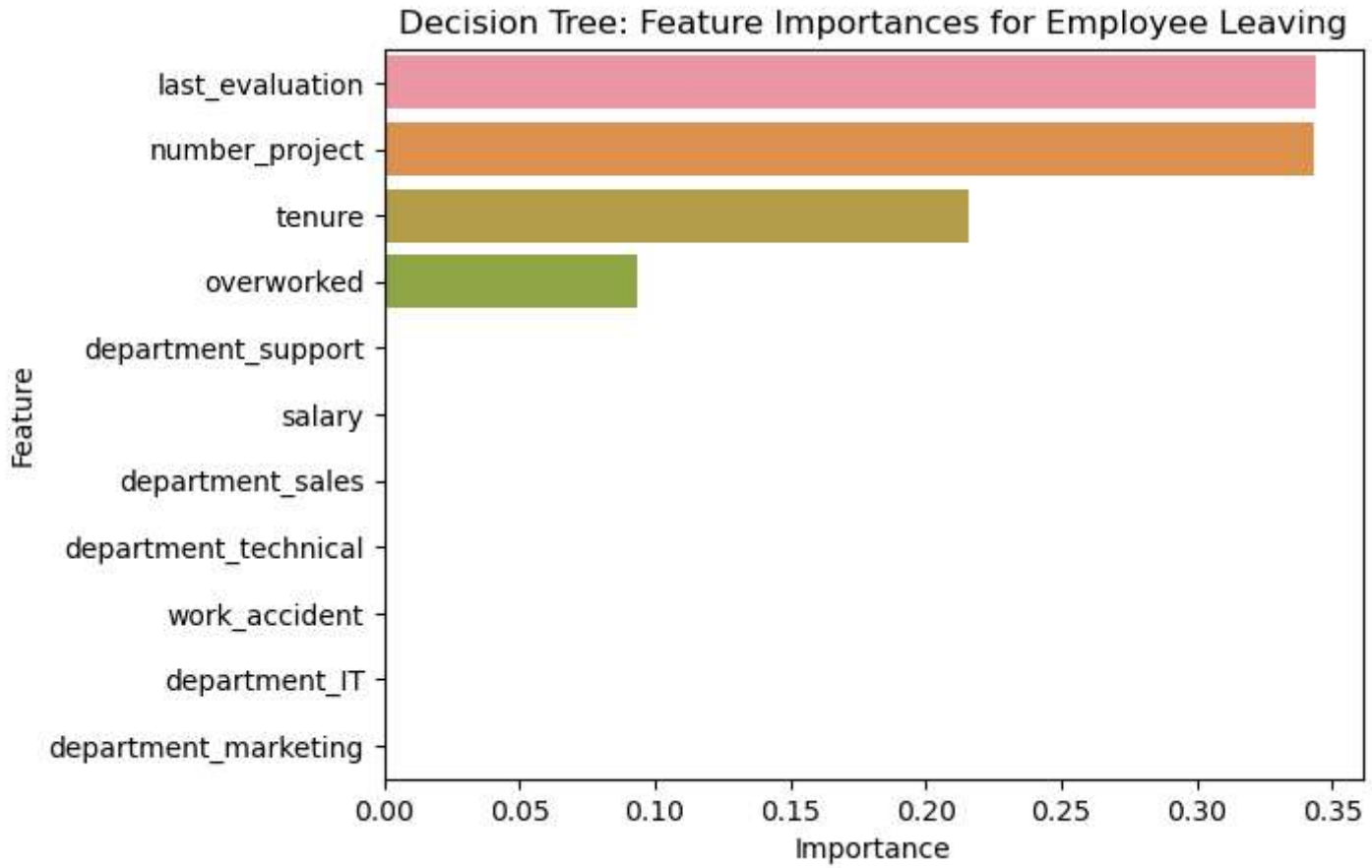
```
tree2_imp = tree2_imp[tree2_imp['gini_importances']>0]
tree2_imp
```

Out[115]:

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

In [119…

```
sns.barplot(data=tree2_imp, x="gini_importances", y=tree2_imp.index, orient='h')
plt.title("Decision Tree: Feature Importances for Employee Leaving", fontsize=12)
plt.ylabel("Feature")
plt.xlabel("Importance")
plt.show()
```



The top three features, **last_evaluation, number_project, and tenure,** are all related to employee performance and experience. Last_evaluation is the most important feature, suggesting that employee performance is a major driver of churn. Number_project is also a significant factor, indicating that employees with more projects are more likely to leave.**Tenure has a moderate positive impact on churn prediction.** Longer-tenured employees are generally less likely to leave than those with shorter tenures. However, it's not the most significant factor compared to performance and workload.

The remaining features are less important, but they may still play a role in churn. **Overworked is the second-most important feature** after last_evaluation, suggesting that employees who feel overworked are more likely to leave. Salary and department_support are also relatively important, suggesting that employee compensation and support from their department may influence churn.

Random Forest feature imp

In [127…

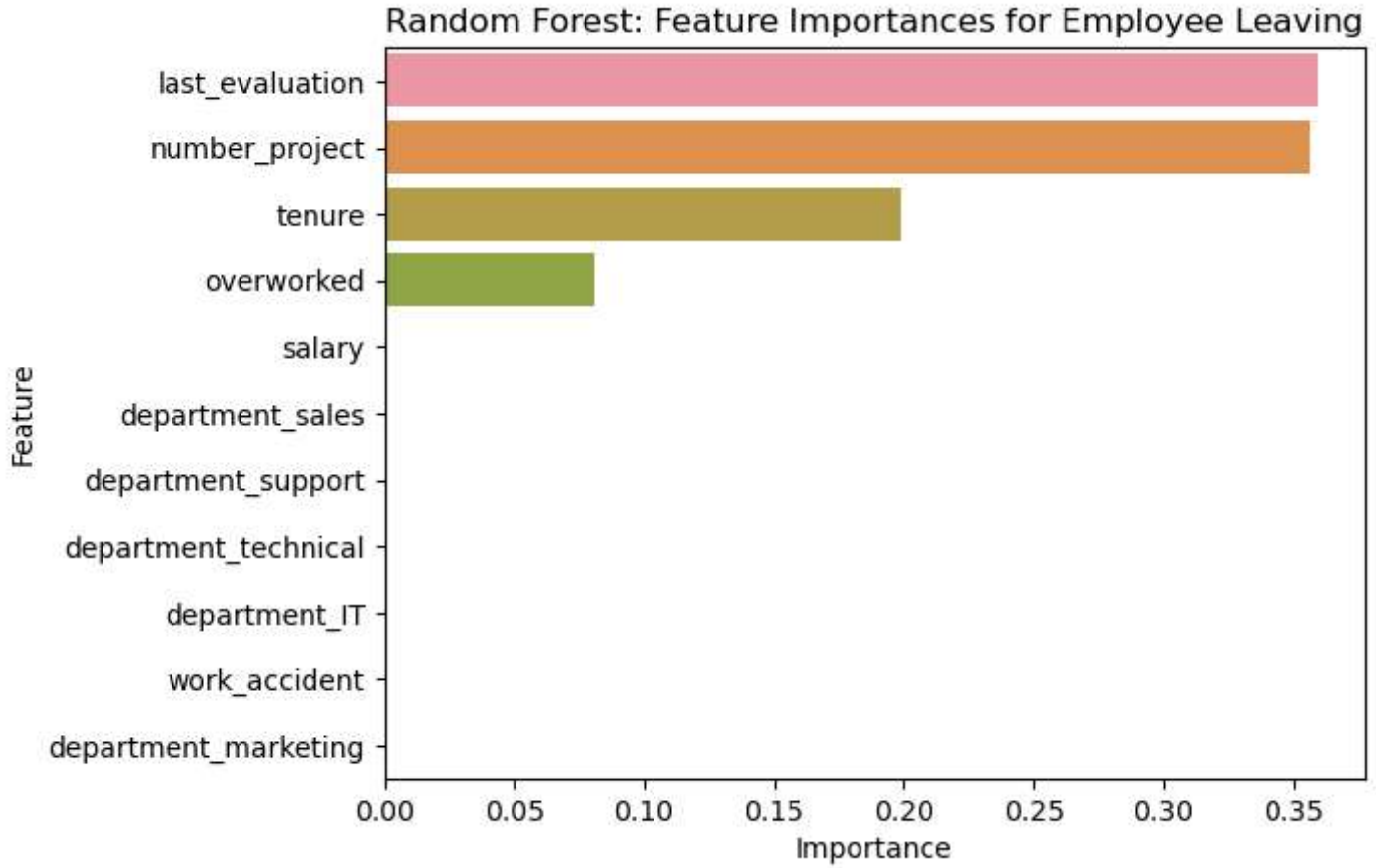
```
rf2_imp = pd.DataFrame(rf2.best_estimator_.feature_importances_,
                       columns=['importance'],
                       index=X.columns)

rf2_imp = rf2_imp.sort_values(by='importance', ascending=False)

rf2_imp
```

Out[127]:

	importance
last_evaluation	0.359494
number_project	0.356801
tenure	0.199109
overworked	0.080984
salary	0.000644
department_sales	0.000612
department_support	0.000578
department_technical	0.000414
department_IT	0.000291
work_accident	0.000276
department_accounting	0.000225
department_product_mng	0.000165
department_RandD	0.000125
department_marketing	0.000098
department_hr	0.000096
department_management	0.000049
promotion_last_5years	0.000040

In [130…

```
# Filter for consistent features (if needed)
rf2_imp = rf2_imp[rf2_imp.index.isin(tree2_imp.index)]

# Create the barplot
sns.barplot(data=rf2_imp, x="importance", y=rf2_imp.index, orient='h')
plt.title("Random Forest: Feature Importances for Employee Leaving", fontsize=12)
plt.ylabel("Feature")
plt.xlabel("Importance")
plt.show()
```



Summary:

- Model Performance:

Random forest outperformed the Logistic Regression by comfortable margin and decision tree by slight margin

- Feature Importances:

Key factors influencing churn include last_evaluation, number_project, tenure, workload, and department support.

Conclusion:

The models effectively predict employee churn, offering insights into factors driving turnover.

Overwork is a significant contributor to employee churn, as evidenced by feature importances.

Recommendations:

- **Prioritize interventions:**

Address overwork concerns by capping projects, rewarding overtime, clarifying expectations, and promoting work-life balance.

- **Investigate four-year tenure:**

Understand dissatisfaction among employees with this tenure and consider promotion opportunities or targeted support.

- **Foster open communication:**

Hold company-wide and team-level discussions to address work culture issues and create a more supportive environment.

- **Reevaluate performance evaluation:**

Ensure scores accurately reflect contributions and effort, not just long hours.

In [ ]: