Capstone project: Salifort Motors - Employee Churn prediction

Google Advanced Data Analytics Professional Certificate Course

Author: Suranjan Sinha

List of contents

Introduction

Salifort Motors is a fictional, 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

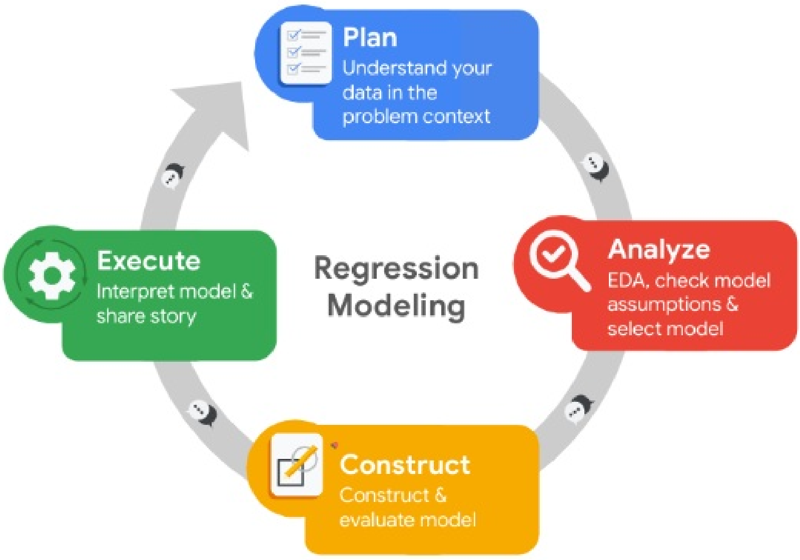
This capstone project is an opportunity to analyze a dataset and build predictive models that can provide insights to the Human Resources (HR) department of a large consulting firm.

Currently, there is a high rate of turnover among Salifort employees. 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.

As a first step, the leadership team asks Human Resources to survey a sample of employees to learn more about what might be driving turnover.

Next, the leadership team asked to analyze the survey data and come up with ideas for how to increase employee retention. To help with this, they suggest to design a model that predicts whether an employee will leave the company based on their job title, department, number of projects, average monthly hours, and any other relevant data points. A good model will help the company increase retention and job satisfaction for current employees, and save money and time training new employees.

PACE stages



Pace: Plan Stage

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 how to utilize this data. As data analytics professional I am being asked to provide data-driven suggestions based on understanding of the data. They have the following question: what's likely to make the employee leave the company?

Our 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 we can predict employees who are 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.

Familiarize with HR dataset

The dataset received contains **15,000 rows** and **10 columns** for the variables listed below. Each row is a different employee's self-reported information.

Column name	Type	Description
satisfaction_level	int64	The employee's self-reported satisfaction level [0-1]
last_evaluation	int64	Score of employee's last performance review [0–1]
number_project	int64	Number of projects employee contributes to
average_monthly_hours	int64	Average number of hours employee worked per month
time_spend_company	int64	How long the employee has been with the company (years)
work_accident	int64	Whether or not the employee experienced an accident while at work
left	int64	Whether or not the employee left the company
promotion_last_5years	int64	Whether or not the employee was promoted in the last 5 years
department	str	The employee's department
salary	str	The employee's salary (low, medium, or high)

Project Goal and Deliverables

Key Stakeholders:

- Salifort's Sr Management team
- Salifort's HR department team

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

Methodologies

- Exploratory Data Analysis
- Descriptive Statistics
- Logistic regression model
- Decision Tree Model
- Random Forest Model
- XGBoost Model

Deliverables:

- Jupyter notebook including:
  - codes
  - analysis workflow
  - model selection
  - model testing
  - Evaluation and results

Reflect on these questions as you complete the plan stage.

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

Reply:

1. Sr management of Salifort Motors and HR department head at Salifort Motors
2. Goal is to predict existing employees who are likely to quit
3.

Step 1. Imports

- Import packages
- Load dataset

In [1]:
```
# Import packages for data manipulation
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)
import pickle

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

# Import packages for data preprocessing
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Import packages for data modeling
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, accuracy_score, precision_score, \
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import roc_auc_score, roc_curve, classification_report, RocCurveDisplay
import sklearn.metrics as metrics

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```python
from xgboost import plot_importance

from scipy import stats
```

In [2]:
```python
df0 = pd.read_csv("C:/Personal/Google Advanced Data Analytics/Capstone Project/Raw Data/HR_capstone_dataset.csv
```

In [3]:
```python
df0.head()
```

Out[3]:

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

Pace: Analyze Stage

Step 2. Exploratory Data Analysis (EDA - Initial data cleaning)

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

In [4]:
```python
# 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
```

In [5]:
```python
# Gather descriptive statistics about the data
### YOUR CODE HERE ###
df0.describe()
```

Out[5]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	le
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.0000
mean	0.612834	0.716102	3.803054	201.050337	3.498233	0.144610	0.23808
std	0.248631	0.171169	1.232592	49.943099	1.460136	0.351719	0.42592
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000	0.00000
25%	0.440000	0.560000	3.000000	156.000000	3.000000	0.000000	0.00000
50%	0.640000	0.720000	4.000000	200.000000	3.000000	0.000000	0.00000
75%	0.820000	0.870000	5.000000	245.000000	4.000000	0.000000	0.00000
max	1.000000	1.000000	7.000000	310.000000	10.000000	1.000000	1.00000

In [6]:
```python
df0['Department'].describe()
```

Out[6]:
```
count      14999
unique        10
top        sales
freq        4140
Name: Department, dtype: object
```

In [7]:
```python
df0['Department'].value_counts()
```

Out[7]:
```
Department
sales           4140
technical       2720
support         2229
IT              1227
product_mng      902
marketing        858
RandD            787
accounting       767
hr               739
management       630
Name: count, dtype: int64
```

In [8]:
```
df0['salary'].describe()
```

Out[8]:
```
count      14999
unique         3
top          low
freq        7316
Name: salary, dtype: object
```

In [9]:
```
df0['salary'].value_counts()
```

Out[9]:
```
salary
low       7316
medium    6446
high      1237
Name: count, dtype: int64
```

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.

In [10]:
```
# Display all column names
df0.columns
```

Out[10]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
       'promotion_last_5years', 'Department', 'salary'],
      dtype='object')
```

In [11]:
```
# Rename columns as needed
df0.columns = df0.columns.str.lower()

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

Out[11]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'work_accident', 'left',
       'promotion_last_5years', 'department', 'salary'],
      dtype='object')
```

Check missing values

Check missing values in the data

In [12]:
```
# Check missing values
df0.isna().sum()
```

Out[12]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
work_accident            0
left                     0
promotion_last_5years    0
department               0
salary                   0
dtype: int64
```

Observation: No missing value were observed in the data

Check duplicates

Check for any duplicate entries in the data

In [13]:
```
# Check for duplicates
### YOUR CODE HERE ###
df0.duplicated().sum()
```

Out[13]: 3008

In [14]:
```
# Inspect some rows containing duplicates as needed
### YOUR CODE HERE ###
duplicates = df0[df0.duplicated()].sort_values(by=['satisfaction_level','last_evaluation', 'average_montly_hours
duplicates.head(6)
```

Out[14]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	left	promo
12030	0.09	0.62	6	294	4	0	1	
14241	0.09	0.62	6	294	4	0	1	
12071	0.09	0.77	5	275	4	0	1	
14282	0.09	0.77	5	275	4	0	1	
12652	0.09	0.77	6	290	4	0	1	
14863	0.09	0.77	6	290	4	0	1	

In [15]:
```
# check the number of duplicate entries
print(duplicates.shape)
print(round(3008/14999*100,2),'% of data which are duplicate')
```

```
(3008, 10)
20.05 % of data which are duplicate
```

In [16]:
```
# 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()
```

Out[16]:

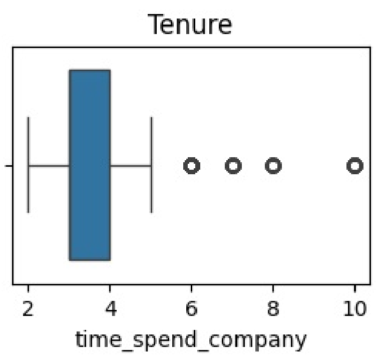
	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	left	promotion_
0	0.38	0.53	2	157	3	0	1	
1	0.80	0.86	5	262	6	0	1	
2	0.11	0.88	7	272	4	0	1	
3	0.72	0.87	5	223	5	0	1	
4	0.37	0.52	2	159	3	0	1	

Observation: There were 3008 duplicate entries identified in the dataset. Which constitutes about 20% of the total dataset. Keeping the last entry of the samilar duplicated rows, rest of the duplicates were deleted and was stored in a new df1 Data Frame. Keeping the original dataset intact.

Check outliers

In [17]:
```
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
plt.figure(figsize=(3,2))
sns.boxplot(x=df1['time_spend_company'])
plt.title('Tenure')
```

Out[17]: Text(0.5, 1.0, 'Tenure')



In [18]:
```
# Determine the number of rows containing outliers
### YOUR CODE HERE ###
percentile25 = df1['time_spend_company'].quantile(0.25)
percentile75 = df1['time_spend_company'].quantile(0.75)
iqr = percentile75 - percentile25
upper_limit = percentile75 + (1.5 * iqr)
df1[df1['time_spend_company'] > upper_limit].shape
```

Out[18]:
```
(824, 10)
```

Observation: Total 824 number of rows contains outliers as identified from column 'time_spend_company'. These values might not be outliers but actual years spent in the company.
Certain types of models are more sensitive to outliers than others. Based on the type of model we will decide to exclude the outliers or include them in modeling.

EDA - Analyse relationship between variables

In [19]:
```
# Creating a copy of 'satisfaction_level'
df1['employee_status'] = np.where(df1['left']==0, 'stayed','left')
```

```
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\146753406.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['employee_status'] = np.where(df1['left']==0, 'stayed','left')
```

In [20]:
```
df1.head()
```

Out[20]:

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

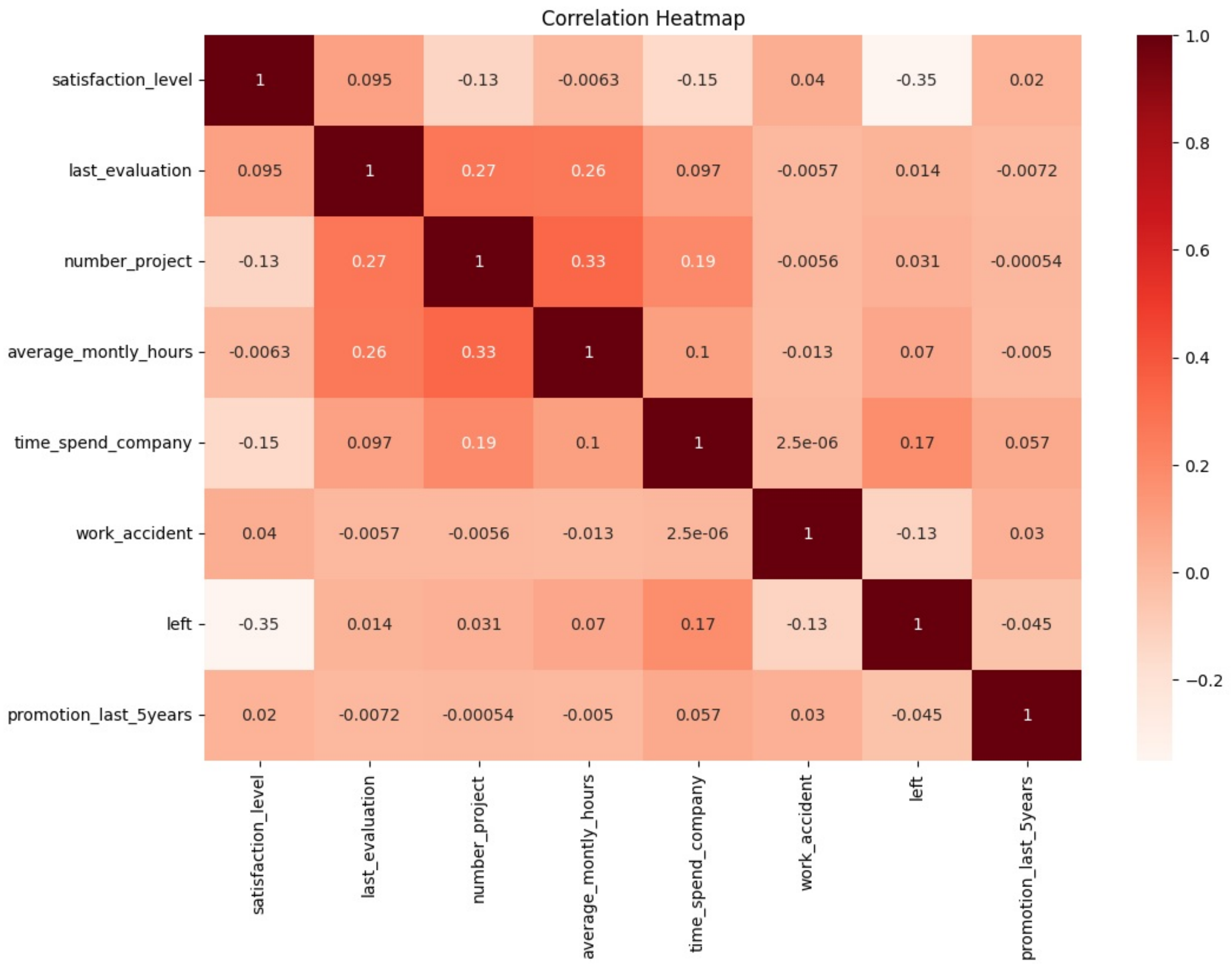
Identifying most correlated variables

In [21]:
```
df1.corr(method='pearson', numeric_only=True)
```

Out[21]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accid
satisfaction_level	1.000000	0.095186	-0.133246	-0.006252	-0.152915	0.039
last_evaluation	0.095186	1.000000	0.270256	0.264678	0.096829	-0.005
number_project	-0.133246	0.270256	1.000000	0.331516	0.188837	-0.005
average_montly_hours	-0.006252	0.264678	0.331516	1.000000	0.102875	-0.012
time_spend_company	-0.152915	0.096829	0.188837	0.102875	1.000000	0.000
work_accident	0.039940	-0.005695	-0.005612	-0.012860	0.000003	1.000
left	-0.350558	0.013520	0.030928	0.070409	0.173295	-0.125
promotion_last_5years	0.019789	-0.007206	-0.000544	-0.004964	0.056828	0.029

In [22]:
```
# Vizualize correlation heatmap of the data
plt.figure(figsize=(12,8))
sns.heatmap(df1.corr(method='pearson', numeric_only=True), annot=True, cmap='Reds')
plt.title('Correlation Heatmap')

plt.show()
```



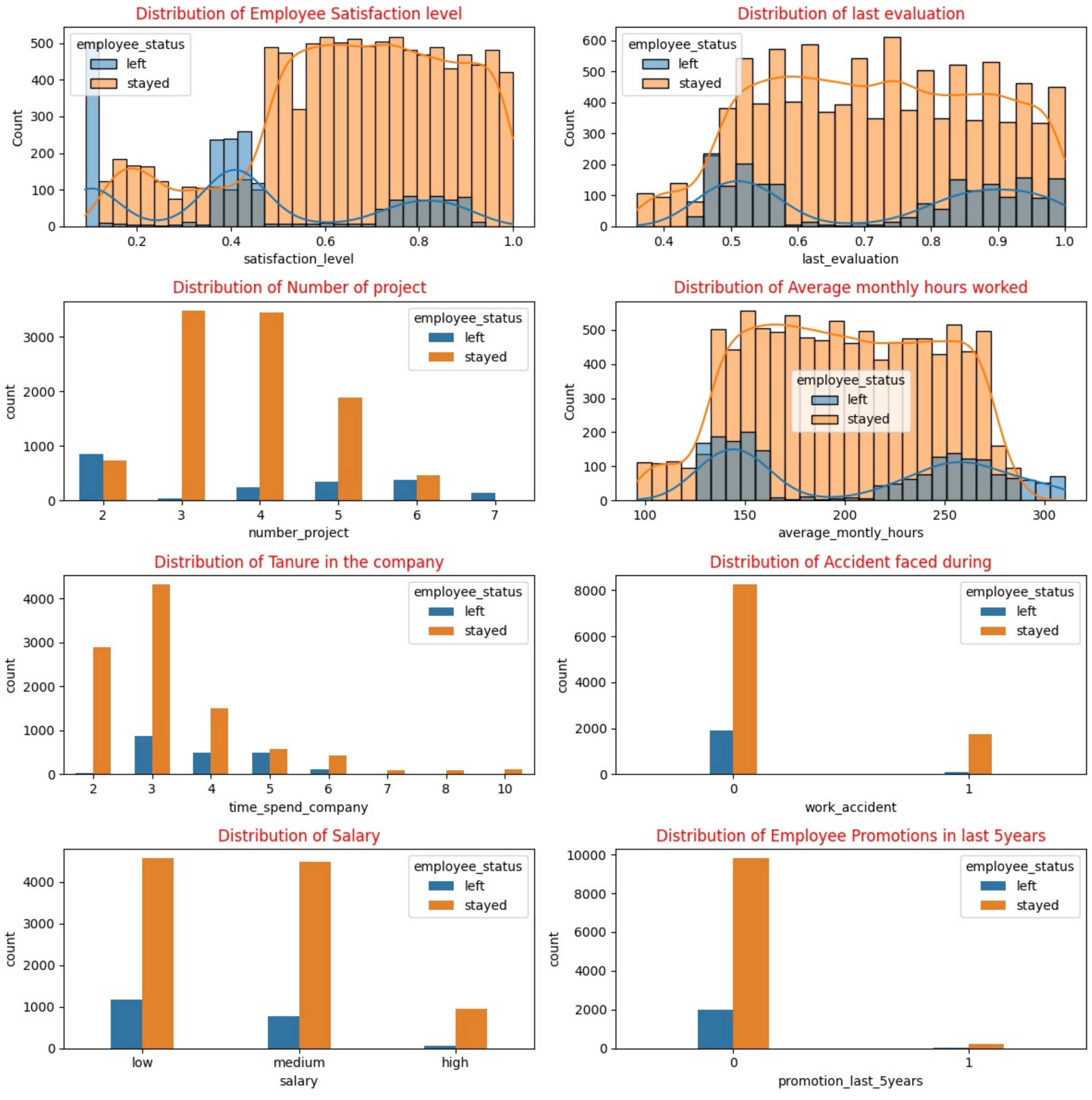
Observation: No variables are found to be highly correlated

EDA - Data Visualization

Now, examining variables and creating plots to visualize relationships between variables in the data.

In [23]:
```python
# Create a plot as needed
# Plot predictor variables and target variable distribution
fig, axes = plt.subplots(4,2, figsize=(12,12))

# Distribution of satisfaction_level
sns.histplot(data=df1, x=df1['satisfaction_level'], hue='employee_status', ax=axes[0,0], kde=True)
axes[0,0].set_title('Distribution of Employee Satisfaction level', color='r')

# Distribution of last_evaluation
sns.histplot(data=df1, x=df1['last_evaluation'], hue='employee_status', ax=axes[0,1], kde=True)
axes[0,1].set_title('Distribution of last evaluation', color='r')

# Distribution of number_project
sns.countplot(data=df1, x=df1['number_project'], width=0.6, hue='employee_status', ax=axes[1,0])
axes[1,0].set_title('Distribution of Number of project', color='r')

# Distribution of average_montly_hours
sns.histplot(data=df1, x=df1['average_montly_hours'], hue='employee_status', ax=axes[1,1], kde=True)
axes[1,1].set_title('Distribution of Average monthly hours worked', color='r')

# Distribution of time_spend_company
sns.countplot(data=df1, x=df1['time_spend_company'], width=0.6, hue='employee_status', ax=axes[2,0])
axes[2,0].set_title('Distribution of Tanure in the company', color='r')

# Distribution of work_accident
sns.countplot(data=df1, x=df1['work_accident'], width=0.2, hue='employee_status', ax=axes[2,1])
axes[2,1].set_title('Distribution of Accident faced during', color='r')

# Distribution of Salary
sns.countplot(data=df1, x=df1['salary'], width=0.4, hue='employee_status', ax=axes[3,0])
axes[3,0].set_title('Distribution of Salary', color='r')
```

```python
# Distribution of promotion_last_5years
sns.countplot(data=df1, x=df1['promotion_last_5years'], width=0.3, hue='employee_status', ax=axes[3,1])
axes[3,1].set_title('Distribution of Employee Promotions in last 5years', color='r')

# plot
plt.tight_layout()
```

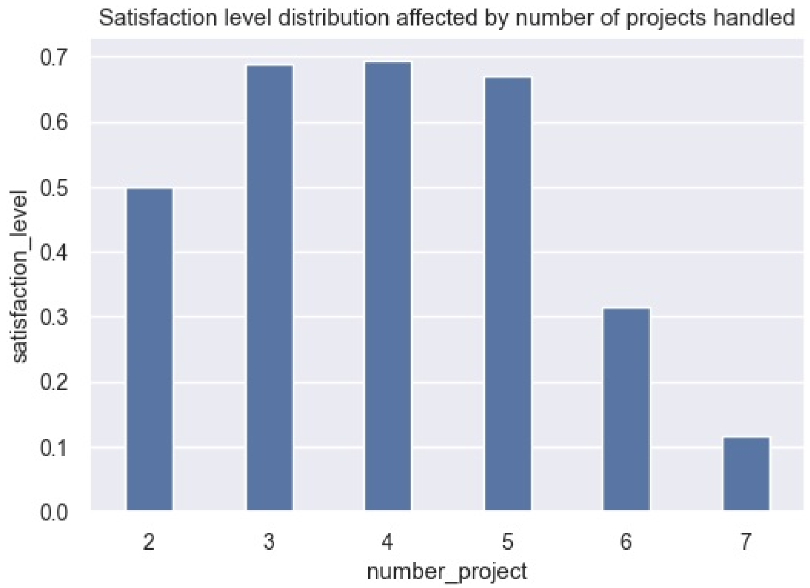


In [120…

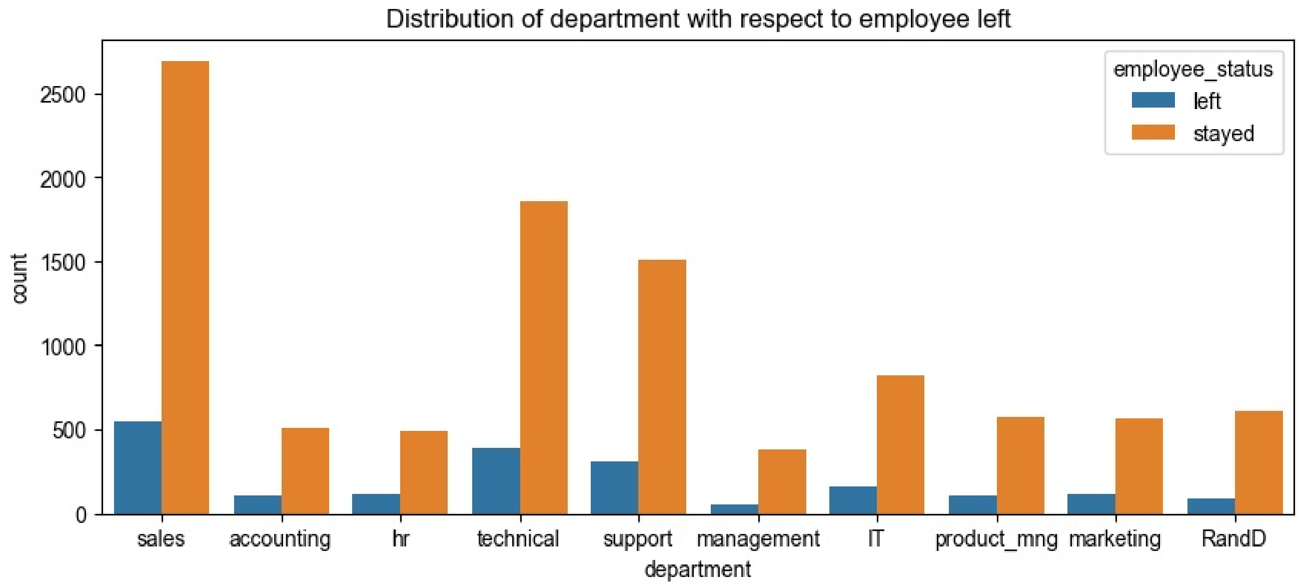
```python
# number of projects vs satisfaction level
plt.figure(figsize=(6,4))
b=df1.groupby('number_project')['satisfaction_level'].mean().reset_index()
sns.barplot(data=b, x='number_project', y='satisfaction_level', width=0.4)
plt.title('Satisfaction level distribution affected by number of projects handled')
```

Out[120…

```
Text(0.5, 1.0, 'Satisfaction level distribution affected by number of projects handled')
```



In [24]:
```python
# Distribution of Department vs. employee left
plt.figure(figsize=(10,4))
sns.countplot(data=df1, x=df1['department'], hue='employee_status')
plt.title('Distribution of department with respect to employee left')
sns.set(font_scale=0.9)
plt.show()
```



Observation: In the plot, Employees left vs average monthly hours worked, it shows that the histogram has two peaks also knows as bimodal. Bimodality shows that within left employees one group of employees who worked 7-8hrs per day while the other group worked more than 11hours a day.

In the plot, Employees left vs number of projects, it shows that there are employees who are involved in 6 to 7 projects. These employees % of churn ratio is very high.

In the plot, Employee left vs Time spent in the company, it shows that most of the employees are in the company for past 2-3 years. Thereafter from 4th year, employees retainability is less and tend to leave the company.

Looking at employee left vs individual Departmental team, its observed that maximum employees left from sales team followed by technical and support teams.

Checking salary distribution vs employees left, it is observed that most of the employees who left were from low salary group and very few left from high salary group.

EDA - Check Target class imbalance

In [25]:
```
# Get numbers of people who left vs. stayed
### YOUR CODE HERE ###
print('Number of people left vs. stayed:', '\n', df1['employee_status'].value_counts())

# Get percentages of people who left vs. stayed
### YOUR CODE HERE ###
print('Number of people left vs. stayed in %:', '\n', round(df1['employee_status'].value_counts(normalize=True)
```

```
Number of people left vs. stayed:
 employee_status
stayed    10000
left       1991
Name: count, dtype: int64
Number of people left vs. stayed in %:
 employee_status
stayed    83.4
left      16.6
Name: proportion, dtype: float64
```

Observation: The dataset has 83.4% employees retained and 16.6% employees left.
There is a imbalance in the target variable. Though this imbalance is within limit and can still be considered without oversampling the data.

Insights

- No missing value were observed in the data
- There were 3008 duplicate entries identified in the dataset. Which constitutes about 20% of the total dataset. Keeping the last entry of the samilar duplicated rows, rest of the duplicates were deleted and was stored in a new df1 Data Frame. Keeping the original dataset intact.
- Total 824 number of rows contains outliers as identified from column 'time_spend_company'. These values might not be outliers but actual years spent in the company. Certain types of models are more sensitive to outliers than others. Based on the type of model we will decide to exclude the outliers or include them in modeling.
- No variables are found to be highly correlated
- In the plot, Employees left vs average monthly hours worked, it shows that the histogram has two peaks also knows as bimodal. Bimodality shows that within left employees one group of employees who worked 7-8hrs per day while the other group worked more than 11hours a day.
- In the plot, Employees left vs number of projects, it shows that there are employees who are involved in 6 to 7 projects. These employees % of churn ratio is very high.
- In the plot, Employee left vs Time spent in the company, it shows that most of the employees are in the company for past 2-3 years. Thereafter from 4th year onwards, employees retainability is less and employees tend to leave the company.
- Looking at employee left vs individual Departmental team, its observed that maximum employees left from sales team followed by technical and support teams.
- After checking salary distribution vs employees left, it is observed that most of the employees who left were from low salary group and very few left from high salary group.
- The dataset has 83.4% employees retained and 16.6% employees left. There is a imbalance in the target variable. Though this imbalance is within limit and can still be considered without oversampling the data.

Pace: Construct Stage

As our predictor variable is categorical/binary we will first build 4 different classification models:

- Binomial Logistic Regression
- Decision Trees
- Random Forest
- XGBoost

We will then select the champion model with best evaluation score to predict on the test data

Evaluation metrics

Evaluation metrics:

1. Precision score
2. Recall score
3. F1 score
4. Accuracy score
5. Confusion matrix: A perfect model would yield all true negatives and true positives, and no false negatives or false positives.
   A. True Negative (TN): The upper-left quadrant displays the number of true negatives, the total count that classification model correctly predicted as False(0). In this case, the employees who didnt leave
   B. False Positive (FP): The upper-right quadrant displays the number of false positives, the total count that classification model incorrectly predicted as True(1). In this case the classification model predicted the employee as 'left' but in reality employee 'stayed'
   C. False Negative (FN): The lower-left quadrant displays the number of false negatives, the count that classification model incorrectly predicted as False(0). In this case the classification model incorrectly predicted an employee as 'stayed' but in reality that employee 'left'
   D. True Positive (TP): The lower-right quadrant displays the number of true positives, the count that classification model correctly predicted as True(1). In this case the classification model correctly predicted employees who left.

The False negatives may cause the company to spend more resources on an employee who decides to leave, as otherwise this may result in spending on hiring new employees and training them which is also time consuming. The False positives may cause the company to spend on the employee incentives and rewards with more benefit, thinking this employee might leave. False negatives will be worse for the company, however false positives will be unnecessary expense to Salifort Motors.

Feature Engineering- Feature Transformation

Extra working hours may lead to poor satisfaction and thereby leaving the company

- Normal working hours is 8hours per day.
- Average Working days in a year (considering 2days weekend) = 261 days
- Average working days in a month = 261 / 12 = 21.75 days
- Average working hours per month = 21.75 * 8 (hours per day) = 174 hrs

In [26]:
```
# creating a column to identify which employee worked more than normal working hours 174.
# keeping any working hours less than 174 as 0 and any working hours more than 174 as difference of actual hour:
df1['extra_working_hours'] = np.where(df1['average_montly_hours'] <= 174, 0, df1['average_montly_hours']-174)
df1.head()
```

```
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\419898338.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['extra_working_hours'] = np.where(df1['average_montly_hours'] <= 174, 0, df1['average_montly_hours']-174)
```

Out[26]:

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

Log of satisfaction_level and last_evaluation

- For feature scaling we will create a new column with log value of satisfaction_level and last_evaluation

In [27]:
```
df1['log_satisfaction_level'] = np.log(df1['satisfaction_level'])
df1['log_last_evaluation'] = np.log(df1['last_evaluation'])
df1.head()
```

```
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\3489145900.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['log_satisfaction_level'] = np.log(df1['satisfaction_level'])
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\3489145900.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['log_last_evaluation'] = np.log(df1['last_evaluation'])
```

Out[27]:

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

Employee Productivity: Calculate employee productivity to understand which employees devoted more efforts for the company

- formula: productivity = number_project * average_montly_hours

In [28]:

```
# Create a new column of employee productivity = number_project * average_montly_hours
df1['productivity'] = df1['number_project']*df1['average_montly_hours']
df1.head()
```

```
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\1458603458.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['productivity'] = df1['number_project']*df1['average_montly_hours']
```

Out[28]:

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

Employee ranking of Productivity per time spent in the company: Calculate employee productivity per time spent in the company to understand which employees devoted more efforts for the company during the whole tanure in the company

- formula: ranking of productivity_per_year = (number_project * average_montly_hours) / time_spend_company

In [29]:

```
# Create a new column ranking productivity per year
df1['productivity_per_year_rank'] = (df1['productivity']/df1['time_spend_company']).rank(pct=True)
df1.head()
```

```
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\1661980510.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['productivity_per_year_rank'] = (df1['productivity']/df1['time_spend_company']).rank(pct=True)
```

Out[29]:

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

Satisfaction level per effort estimation:

- We will compute the satisfaction level per effort given by each employee
- formula = satisfaction_level / (average_monthly_hours * time_spend_company)

In [30]:
```
# create a new column for satisfaction_per_effort
df1['satisfaction_per_effort'] = df1['satisfaction_level'] / (df1['average_montly_hours'] * df1['time_spend_com
df1.head()
```

```
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\1524468987.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['satisfaction_per_effort'] = df1['satisfaction_level'] / (df1['average_montly_hours'] * df1['time_spend_co
mpany'])
```

Out[30]:

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

Encoding categorical variables

In [31]:
```
# show few lines of transformed dataset
df1.head()
```

Out[31]:

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

Encode salary column

- salary column is in object type. We will encode it to integer so that it can be used as predictor variable in models

In [32]:
```
# Encode Salary column to integer
map_ref = {'low':1,
           'medium':2,
           'high':3}
df1['salary'] = df1['salary'].map(map_ref)
df1.head()
```

```
C:\Users\dsinh\AppData\Local\Temp\ipykernel_39640\1048059334.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df1['salary'] = df1['salary'].map(map_ref)
```

Out[32]:

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

Dummy encoding

- We will Encode 'department' column as dummies

In [33]:
```
df1 = pd.get_dummies(df1, columns=['department'])
df1.head()
```

Out[33]:

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

In [34]:
```
# drop employee_status column as satisfaction_level column is already present
df1 = df1.drop('employee_status', axis=1)
df1.head()
```

Out[34]:

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

Feature Selection

In [35]:
```
X = df1.copy()
```

In [36]:
```
# drop unnecessary columns
X = X.drop('left', axis=1)
```

Dropping Department and Sub-department columns

- As we are predicting the possiblity of employee who can leave the company, Department information can not be the driving factor for the affect of employee churn.

In [37]:
```
X = X.drop(['department_IT', 'department_RandD', 'department_accounting',
       'department_hr', 'department_management', 'department_marketing',
       'department_product_mng', 'department_sales', 'department_support',
       'department_technical'], axis=1)
```

In [38]:
```
X.columns
```

Out[38]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'work_accident',
       'promotion_last_5years', 'salary', 'extra_working_hours',
       'log_satisfaction_level', 'log_last_evaluation', 'productivity',
       'productivity_per_year_rank', 'satisfaction_per_effort'],
      dtype='object')
```

In [39]:
```
X.head()
```

Out[39]:

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

In [40]:
```
# Assign target variable
y=df1['left']
```

Create Train/Test sets

- Split data into training and testing sets, 75/25 ratio
- As target variable is imbalanced, accordingly we will use same ratio when creating train/test set, using parameter 'stratify'

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=0)
```

Build Model - Logistic Regression

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

In [42]:
```
# Fit logistic regression model to the data
log_clf = LogisticRegression(random_state=0, max_iter=6000).fit(X_train,y_train)

# Predict the outcome of the test data
y_pred = log_clf.predict(X_test)
```

In [43]:
```
# Analyse Logistic Regression results
# Print out the model's accuracy, precision, recall, and F1 score.
print("Logistic Regression results:")
print("Accuracy:", "%.6f" % metrics.accuracy_score(y_test, y_pred))
print("Precision:", "%.6f" % metrics.precision_score(y_test, y_pred))
print("Recall:", "%.6f" % metrics.recall_score(y_test, y_pred))
print("F1 Score:", "%.6f" % metrics.f1_score(y_test, y_pred))
```

```
Logistic Regression results:
Accuracy: 0.912608
Precision: 0.793532
Recall: 0.640562
F1 Score: 0.708889
```

Logistic Regression Results and Evaluation

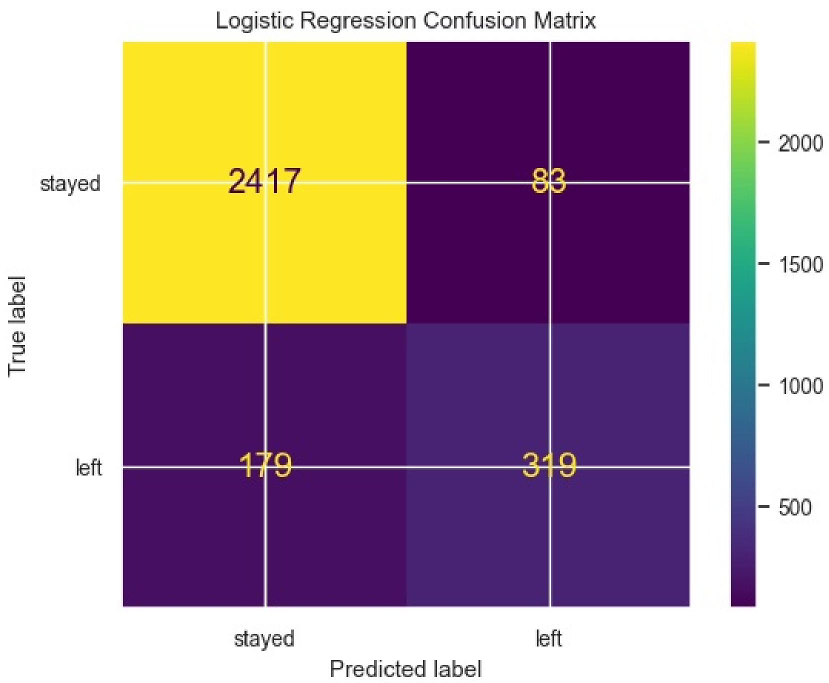
In [44]:
```
# Logistic Regression Test Results
model_name = 'Logistic Regression'
precision = "%.6f" % metrics.precision_score(y_test, y_pred)
recall = "%.6f" % metrics.recall_score(y_test, y_pred)
f1 = "%.6f" % metrics.f1_score(y_test, y_pred)
accuracy = "%.6f" % metrics.accuracy_score(y_test, y_pred)
result10 = pd.DataFrame({'Model':[model_name],
                       'Precision':[precision],
                        'Recall':[recall],
                        'F1':[f1],
                        'Accuracy':[accuracy]})
result10
```

Out[44]:

	Model	Precision	Recall	F1	Accuracy
0	Logistic Regression	0.793532	0.640562	0.708889	0.912608

In [45]:
```
# Confusion metrix
log_cm = metrics.confusion_matrix(y_test, y_pred, labels = log_clf.classes_)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = log_cm, display_labels = ['stayed', 'left']) # log_clf
plt.rcParams.update({'font.size': 16})
disp.plot()
disp.ax_.set_title("Logistic Regression Confusion Matrix")
```

Out[45]:
```
Text(0.5, 1.0, 'Logistic Regression Confusion Matrix')
```



Build Model - Decision Tree

In [46]:
```
# Fit Decision tree classifier model to the data
decision_tree = DecisionTreeClassifier(random_state=0)

decision_tree.fit(X_train, y_train)

dt_pred = decision_tree.predict(X_test)
```

In [47]:
```
# Analyse Decision tree results
# print out the decision tree models precision, recall, f1 and accuracy score
print("Decision Tree results:")
print("Accuracy:", "%.6f" % metrics.accuracy_score(y_test, dt_pred))
print("Precision:", "%.6f" % metrics.precision_score(y_test, dt_pred))
print("Recall:", "%.6f" % metrics.recall_score(y_test, dt_pred))
print("F1 Score:", "%.6f" % metrics.f1_score(y_test, dt_pred))
```

```
Decision Tree results:
Accuracy: 0.967645
Precision: 0.893910
Recall: 0.913655
F1 Score: 0.903674
```

Decision Tree Results and Evaluation

In [48]:
```
# Decision Tree Test Results
model_name = 'Decision Tree'
precision = "%.6f" % metrics.precision_score(y_test, dt_pred)
recall = "%.6f" % metrics.recall_score(y_test, dt_pred)
f1 = "%.6f" % metrics.f1_score(y_test, dt_pred)
accuracy = "%.6f" % metrics.accuracy_score(y_test, dt_pred)
result11 = pd.DataFrame({'Model':[model_name],
                       'Precision':[precision],
                        'Recall':[recall],
                        'F1':[f1],
                        'Accuracy':[accuracy]})
result11
```

Out[48]:

	Model	Precision	Recall	F1	Accuracy
0	Decision Tree	0.893910	0.913655	0.903674	0.967645

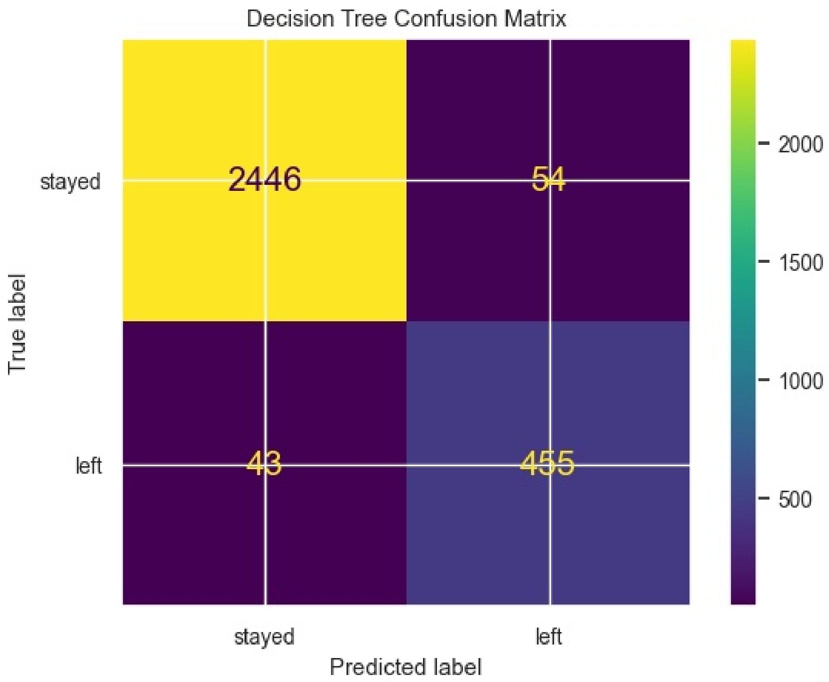
In [49]:
```
result = pd.concat([result10, result11]).sort_values(by=['F1'], ascending=False).reset_index(drop='first')
result
```

Out[49]:

	Model	Precision	Recall	F1	Accuracy
0	Decision Tree	0.893910	0.913655	0.903674	0.967645
1	Logistic Regression	0.793532	0.640562	0.708889	0.912608

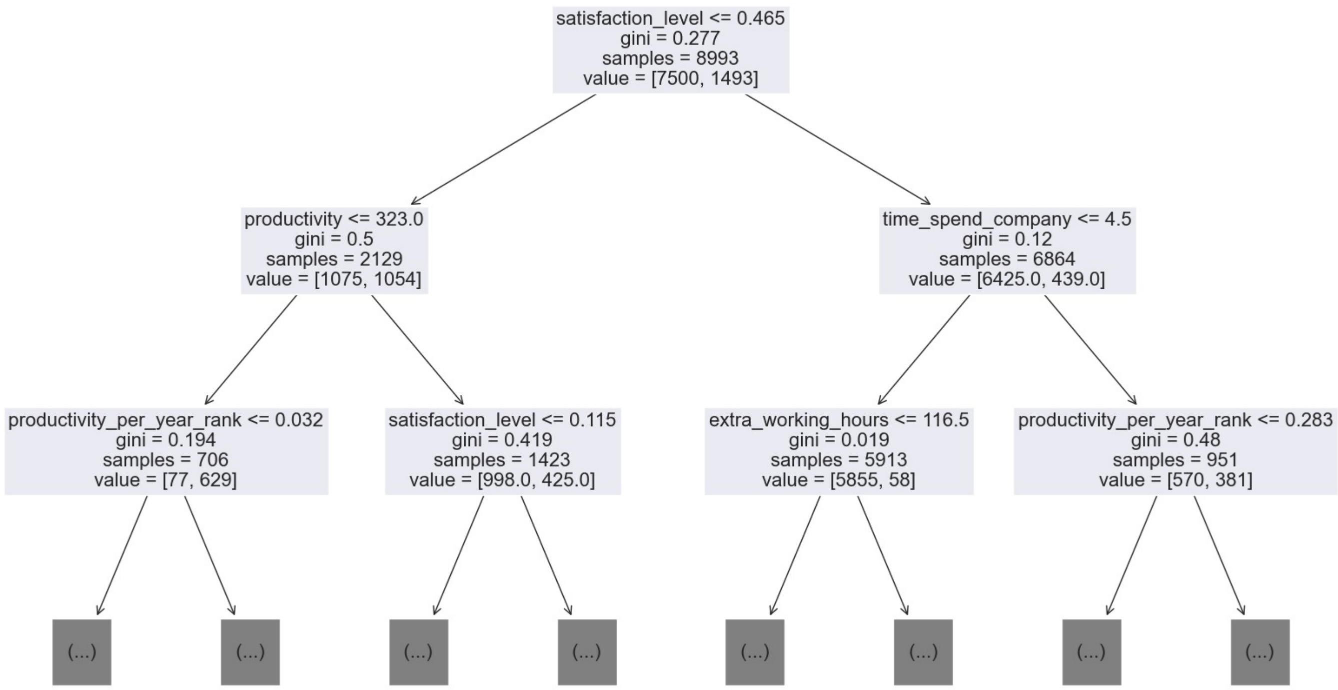
In [50]:
```
# Confusion Matrix
dt_cm = metrics.confusion_matrix(y_test, dt_pred, labels = decision_tree.classes_)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = dt_cm,display_labels = ['stayed', 'left']) # decision_
plt.rcParams.update({'font.size': 16})
disp.plot()
disp.ax_.set_title("Decision Tree Confusion Matrix")
```

Out[50]:
```
Text(0.5, 1.0, 'Decision Tree Confusion Matrix')
```



In [51]:
```
# Plot decision tree
plt.figure(figsize=(20,12))
plot_tree(decision_tree, max_depth=2, fontsize=16, feature_names=X.columns)
```

Out[51]:
```
[Text(0.5, 0.875, 'satisfaction_level <= 0.465\ngini = 0.277\nsamples = 8993\nvalue = [7500, 1493]'),
 Text(0.25, 0.625, 'productivity <= 323.0\ngini = 0.5\nsamples = 2129\nvalue = [1075, 1054]'),
 Text(0.125, 0.375, 'productivity_per_year_rank <= 0.032\ngini = 0.194\nsamples = 706\nvalue = [77, 629]'),
 Text(0.0625, 0.125, '\n  (...)  \n'),
 Text(0.1875, 0.125, '\n  (...)  \n'),
 Text(0.375, 0.375, 'satisfaction_level <= 0.115\ngini = 0.419\nsamples = 1423\nvalue = [998.0, 425.0]'),
 Text(0.3125, 0.125, '\n  (...)  \n'),
 Text(0.4375, 0.125, '\n  (...)  \n'),
 Text(0.75, 0.625, 'time_spend_company <= 4.5\ngini = 0.12\nsamples = 6864\nvalue = [6425.0, 439.0]'),
 Text(0.625, 0.375, 'extra_working_hours <= 116.5\ngini = 0.019\nsamples = 5913\nvalue = [5855, 58]'),
 Text(0.5625, 0.125, '\n  (...)  \n'),
 Text(0.6875, 0.125, '\n  (...)  \n'),
 Text(0.875, 0.375, 'productivity_per_year_rank <= 0.283\ngini = 0.48\nsamples = 951\nvalue = [570, 381]'),
 Text(0.8125, 0.125, '\n  (...)  \n'),
 Text(0.9375, 0.125, '\n  (...)  \n')]
```

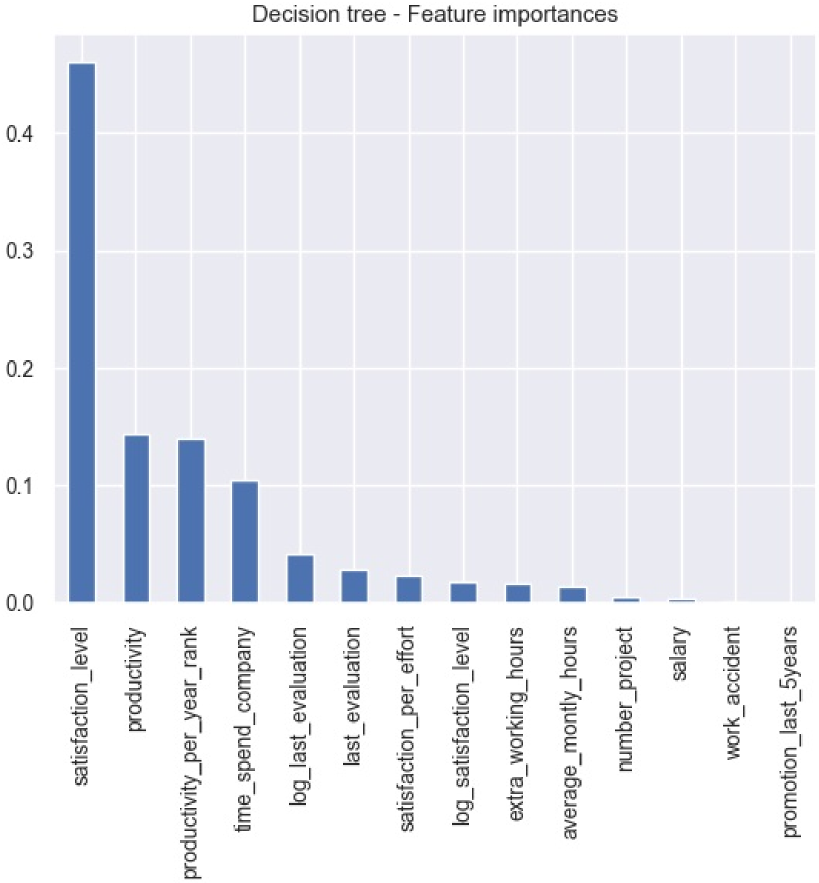


In [52]:

```
# Display feature importances
importances = decision_tree.feature_importances_
dt_importances = pd.Series(importances, index=X.columns).sort_values(ascending=False)

fig, ax = plt.subplots()
dt_importances.plot.bar(ax=ax)
plt.title('Decision tree - Feature importance')
```

Out[52]:

```
Text(0.5, 1.0, 'Decision tree - Feature importances')
```



Build Model - Random Forest (with hyperparameter tuning)

In [53]:
```
# Instantiate Random forest classifier
rf = RandomForestClassifier(random_state=0)

# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [5,7],
             'max_features':[0.3, 0.6],
             'max_samples': [0.7],
             'min_samples_leaf': [1,2],
             'min_samples_split': [2,3],
             'n_estimators': [50, 75, 100]}

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

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

In [54]:
```
%%time
# fit model
rf_cv = rf_cv.fit(X_train, y_train)
```

```
rf_cv
```

```
CPU times: total: 1min 8s
Wall time: 1min 41s
```

Out[54]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

Random Forest - Hyperparameter tuning

- 2nd iteration

In [55]:

```
# Check best recall score
rf_cv.best_score_
```

Out[55]: 0.9136001436555856

In [56]:

```
# Check best parameters
rf_cv.best_params_
```

Out[56]:

```
{'max_depth': 7,
 'max_features': 0.6,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 50}
```

In [57]:

```
# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [7,12],
             'max_features':[0.6, 0.9],
             'max_samples': [0.5, 0.7],
             'min_samples_leaf': [1,2],
             'min_samples_split': [2,3],
             'n_estimators': [25, 50, 75]}

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

In [58]:

```
%%time
# fit model again for 2nd iteraton
rf_cv = rf_cv.fit(X_train, y_train)
rf_cv
```

```
CPU times: total: 3min 3s
Wall time: 4min 10s
```

Out[58]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

In [59]:

```
# Check best recall score
rf_cv.best_score_
```

Out[59]: 0.9162802181769208

In [60]:

```
# Check best parameters
rf_cv.best_params_
```

Out[60]:

```
{'max_depth': 7,
 'max_features': 0.6,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 25}
```

Random Forest Results and evaluation

- Use the best parameters found via GridSearchCV to predict on the test data

In [61]:

```
# Use best parameters on GridSearchCV
rf_opt = RandomForestClassifier(n_estimators = 25, max_depth=7, max_features=0.6, max_samples=0.7,
                                min_samples_leaf=1, min_samples_split=2, random_state=0)
```

In [62]:

```
# Fit the optimal model
```

```python
rf_opt.fit(X_train, y_train)
```

Out[62]:

```
RandomForestClassifier
RandomForestClassifier(max_depth=7, max_features=0.6, max_samples=0.7,
                       n_estimators=25, random_state=0)
```

In [63]:

```python
# Predict on test set using the optimal model.
y_pred_rf = rf_opt.predict(X_test)
```

In [64]:

```python
# Get precision score.
pc_test = precision_score(y_test, y_pred_rf, pos_label = 1)
print("Precision score is {pc:.6f}".format(pc = pc_test))

# Get recall score
rc_test = recall_score(y_test, y_pred_rf, pos_label = 1)
print("Recall score is {rc:.6f}".format(rc = rc_test))

# Get accuracy score
ac_test = accuracy_score(y_test, y_pred_rf)
print("Accuracy score is {ac:.6f}".format(ac = ac_test))

# Get f1 score
f1_test = f1_score(y_test, y_pred_rf, pos_label = 1)
print("F1 score is {f1:.6f}".format(f1 = f1_test))
```

```
Precision score is 0.980603
Recall score is 0.913655
Accuracy score is 0.982655
F1 score is 0.945946
```

In [65]:

```python
# Random Forest Test Results
model_name = 'Random Forest'
precision = "%.6f" % metrics.precision_score(y_test, y_pred_rf, pos_label=1)
recall = "%.6f" % metrics.recall_score(y_test, y_pred_rf, pos_label=1)
f1 = "%.6f" % metrics.f1_score(y_test, y_pred_rf, pos_label=1)
accuracy = "%.6f" % metrics.accuracy_score(y_test, y_pred_rf)
result12 = pd.DataFrame({'Model':[model_name],
                        'Precision':[precision],
                         'Recall':[recall],
                         'F1':[f1],
                         'Accuracy':[accuracy]})
result12
```

Out[65]:

	Model	Precision	Recall	F1	Accuracy
0	Random Forest	0.980603	0.913655	0.945946	0.982655

In [66]:

```python
# Add Random Forest score result to other Model results
result = pd.concat([result, result12]).sort_values(by=['F1'], ascending=False).reset_index(drop='first')
result
```

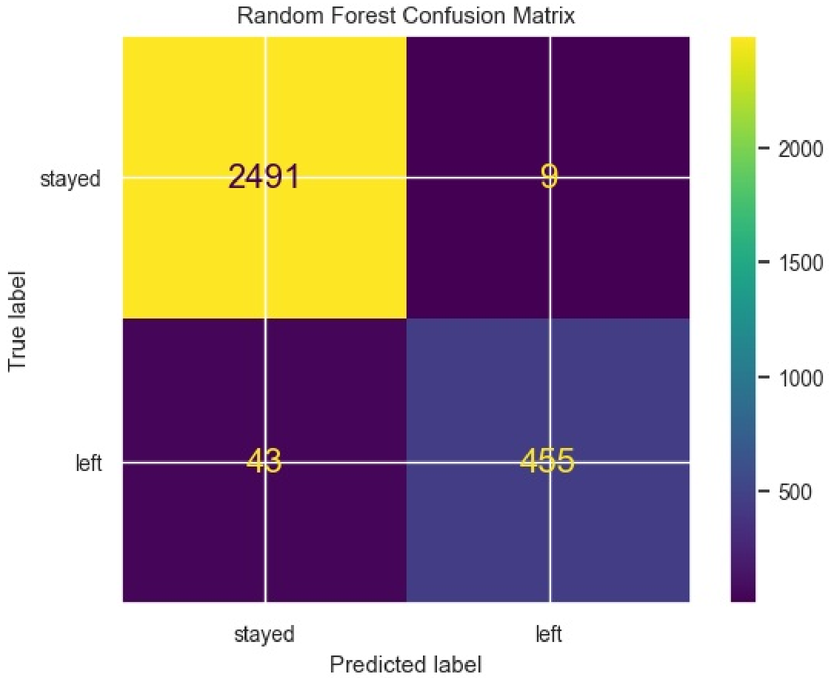
Out[66]:

	Model	Precision	Recall	F1	Accuracy
0	Random Forest	0.980603	0.913655	0.945946	0.982655
1	Decision Tree	0.893910	0.913655	0.903674	0.967645
2	Logistic Regression	0.793532	0.640562	0.708889	0.912608

In [67]:

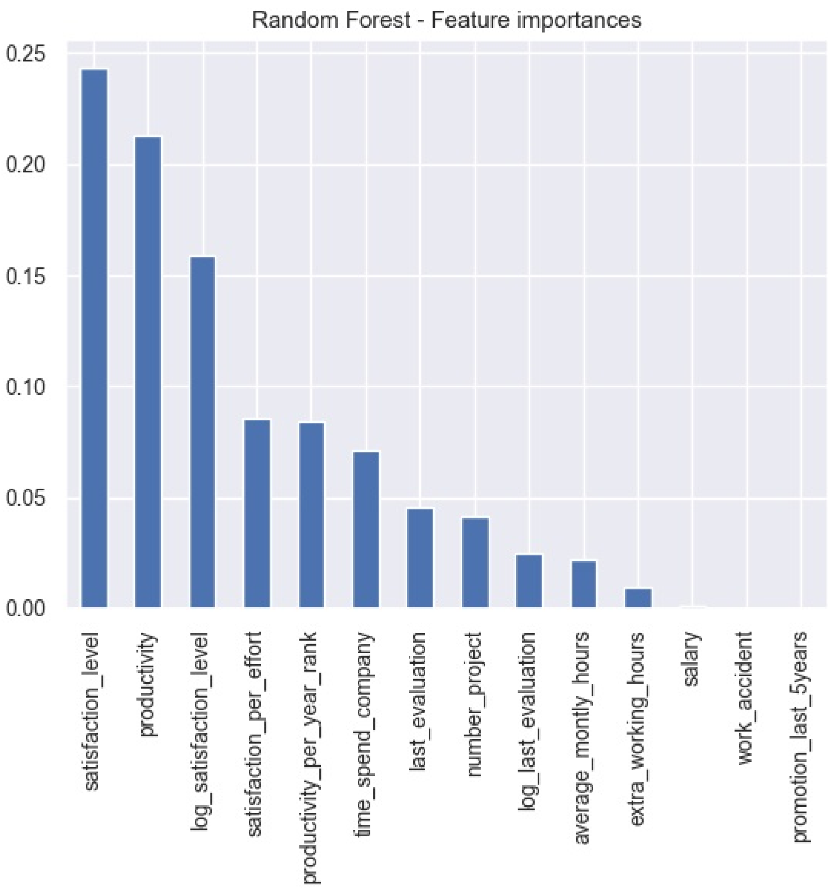
```python
# Confusion Matrix
rf_cm = metrics.confusion_matrix(y_test, y_pred_rf, labels = rf_opt.classes_)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = rf_cm, display_labels = ['stayed', 'left']) # rf_opt.c
plt.rcParams.update({'font.size': 16})
disp.plot()
disp.ax_.set_title("Random Forest Confusion Matrix")
```

Out[67]: Text(0.5, 1.0, 'Random Forest Confusion Matrix')



In [68]:
```
# Display feature importances
importances = rf_opt.feature_importances_
dt_importances = pd.Series(importances, index=X.columns).sort_values(ascending=False)

fig, ax = plt.subplots()
dt_importances.plot.bar(ax=ax)
plt.title('Random Forest - Feature importance')
```

Out[68]:
```
Text(0.5, 1.0, 'Random Forest - Feature importances')
```



Build Model - XGBoost (tune hyperparameter)

In [69]:
```
# 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': [0.1, 0.2, 0.3],
             'n_estimators': [100, 300]}

# Define a dictionary of scoring metrics to capture
```

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

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

In [70]:
```
%%time
# Now fit the model to the X_train and y_train data.
xgb_cv.fit(X_train, y_train)
```

```
CPU times: total: 1min 54s
Wall time: 32.7 s
```

Out[70]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

XGBoost - Hyperparameter tuning

- 2nd iteration

In [71]:
```
# Get best score
xgb_cv.best_score_
```

Out[71]: `0.9142623061210747`

In [72]:
```
# Get best params
xgb_cv.best_params_
```

Out[72]:
```
{'learning_rate': 0.1,
 'max_depth': 4,
 'min_child_weight': 3,
 'n_estimators': 300}
```

In [73]:
```
# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [3,4,5],
             'min_child_weight': [2, 3, 4],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [200, 300, 400]}

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

In [74]:
```
%%time
# Now fit the model to the X_train and y_train data for 2nd iteration
xgb_cv.fit(X_train, y_train)
```

```
CPU times: total: 3min 5s
Wall time: 53.3 s
```

Out[74]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

XGBoost - Hyperparameter tuning

- 3rd iteration

In [75]:
```
# Get best score
xgb_cv.best_score_
```

Out[75]: `0.9189468249870935`

In [76]:
```
# Get best params
xgb_cv.best_params_
```

Out[76]:
```
{'learning_rate': 0.1,
 'max_depth': 3,
 'min_child_weight': 3,
 'n_estimators': 200}
```

In [77]:
```
# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [1,2,3],
             'min_child_weight': [0.2, 0.25, 0.3],
             'learning_rate': [0.07, 0.08, 0.09],
             'n_estimators': [550, 600, 650]}
```

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

In [78]:
```
%%time
# Now fit the model to the X_train and y_train data for 3rd iteration
xgb_cv.fit(X_train, y_train)
```

```
CPU times: total: 5min 40s
Wall time: 1min 37s
```

Out[78]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```

In [79]:
```
# Get best score
xgb_cv.best_score_
```

Out[79]: 0.924975870350834

In [80]:
```
# Get best params
xgb_cv.best_params_
```

Out[80]:
```
{'learning_rate': 0.08,
 'max_depth': 2,
 'min_child_weight': 0.2,
 'n_estimators': 550}
```

XGBoost Results and evaluation

- Use the best estimators found via GridSearchCV to predict on the test data

In [81]:
```
# Use XGBoost model to predict on test data
xgb_preds = xgb_cv.best_estimator_.predict(X_test)
```

In [82]:
```
print('XGBoost Score:')

# Get precision score.
pc_test = precision_score(y_test, xgb_preds, pos_label = 1)
print("Precision score is {pc:.6f}".format(pc = pc_test))

# Get recall score
rc_test = recall_score(y_test, xgb_preds, pos_label = 1)
print("Recall score is {rc:.6f}".format(rc = rc_test))

# Get accuracy score
ac_test = accuracy_score(y_test, xgb_preds)
print("Accuracy score is {ac:.6f}".format(ac = ac_test))

# Get f1 score
f1_test = f1_score(y_test, xgb_preds, pos_label = 1)
print("F1 score is {f1:.6f}".format(f1 = f1_test))
```

```
XGBoost Score:
Precision score is 0.976596
Recall score is 0.921687
Accuracy score is 0.983322
F1 score is 0.948347
```

In [83]:
```
# Random Forest Test Results
model_name = 'XGBoost'
precision = "%.6f" % metrics.precision_score(y_test, xgb_preds, pos_label=1)
recall = "%.6f" % metrics.recall_score(y_test, xgb_preds, pos_label=1)
f1 = "%.6f" % metrics.f1_score(y_test, xgb_preds, pos_label=1)
accuracy = "%.6f" % metrics.accuracy_score(y_test, xgb_preds)
result14 = pd.DataFrame({'Model':[model_name],
                       'Precision':[precision],
                        'Recall':[recall],
                        'F1':[f1],
                        'Accuracy':[accuracy]})
result14
```

Out[83]:

	Model	Precision	Recall	F1	Accuracy
0	XGBoost	0.976596	0.921687	0.948347	0.983322

In [84]:
```
# Add XGBoost score result to olther Model results
result = pd.concat([result, result14]).sort_values(by=['F1'], ascending=False).reset_index(drop='first')
result
```

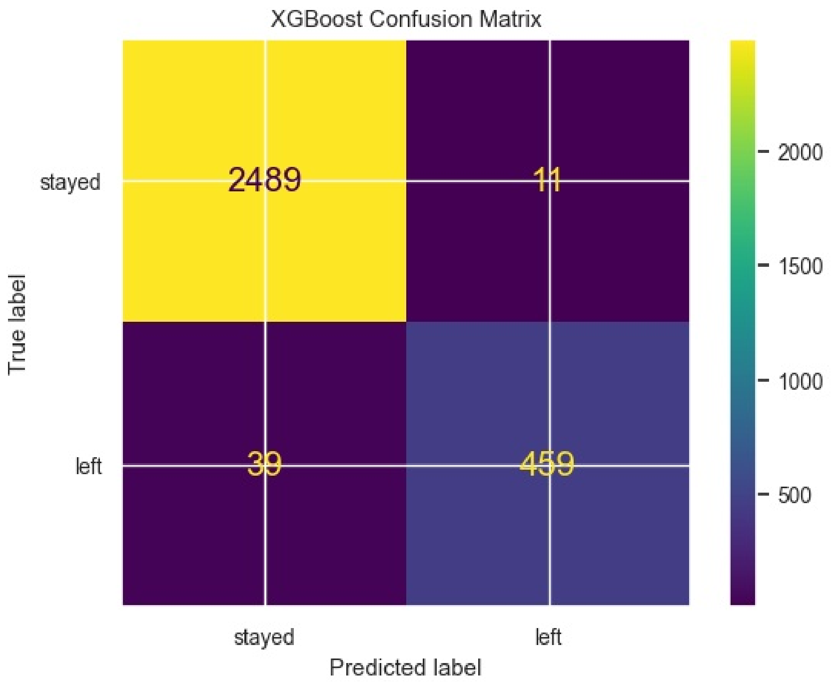
Out[84]:

	Model	Precision	Recall	F1	Accuracy
0	XGBoost	0.976596	0.921687	0.948347	0.983322
1	Random Forest	0.980603	0.913655	0.945946	0.982655
2	Decision Tree	0.893910	0.913655	0.903674	0.967645
3	Logistic Regression	0.793532	0.640562	0.708889	0.912608

In [85]:

```
# Confusion Matrix
xgb_cm = metrics.confusion_matrix(y_test, xgb_preds, labels = xgb_cv.classes_)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = xgb_cm, display_labels = ['stayed', 'left']) # rf_opt.
plt.rcParams.update({'font.size': 16})
disp.plot()
disp.ax_.set_title("XGBoost Confusion Matrix")
```

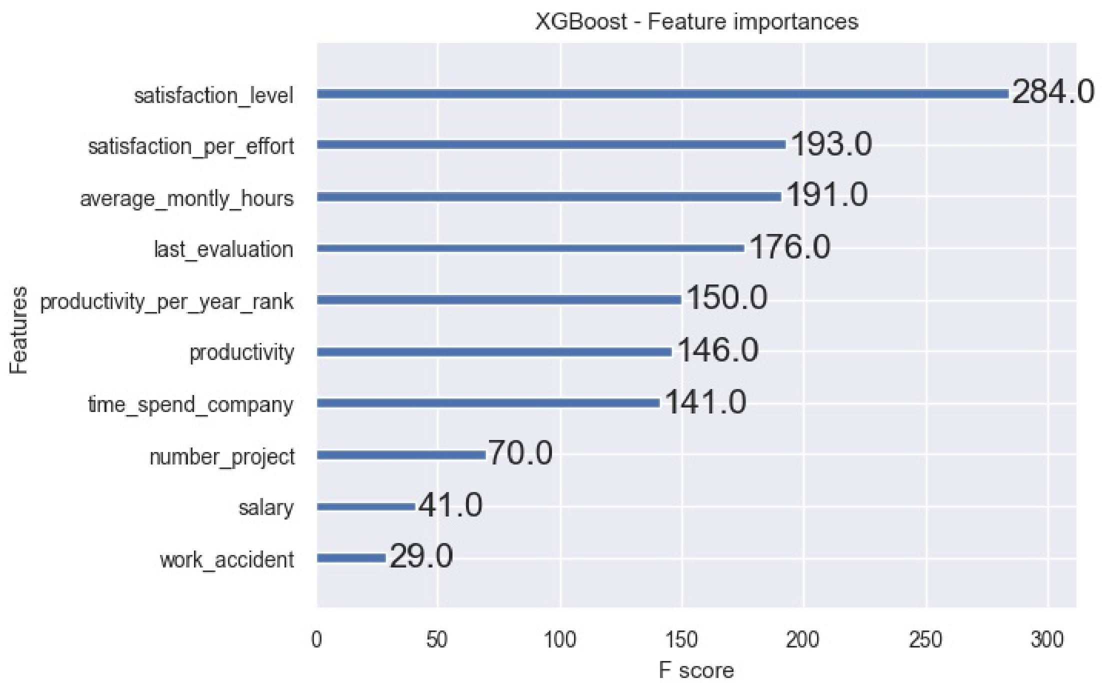
Out[85]: Text(0.5, 1.0, 'XGBoost Confusion Matrix')



In [86]:

```
# Display feature importances
plot_importance(xgb_cv.best_estimator_)
plt.title('XGBoost - Feature importance')
```

Out[86]: Text(0.5, 1.0, 'XGBoost - Feature importances')



Pace: Execute Stage

Project Steps followed

Steps followed in this project:
Project goal was set based on the requirement from Salifort HR team

STEP1. Performed detailed Exploratory Data Analysis (EDA) on HR_capstone_dataset.csv as provided by Salifort HR team. Including preprocessing, data cleaning, data readiness and normalize the data for model. Its to be noted that we couldn't carry out authentication and validation of dataset source as it was out of scope of this project.

STEP2. Additionally, we analysed the relationship between variables to understand the correlation.

STEP3. Identified the predictor variables and target variable and their relationship analysis.

STEP4. We also carried out Feature transformation and encoded the categorical variables to numerical.

STEP5. We carried out Supervised learning model on labeled data. Our goal is to learn the relationship from the input data and make predictions based on the learnings, on new data.

STEP6. We performed Logistic regression model, decision trees, Random forest and XGBoost model to compare and identify the best performing model that provides the best results.

STEP7. Various hyperparameters were considered specially for Random forest and XGBoost model preparation for tuning the model.

STEP8. To reach to a conclusion on the best model performance, Evaluation metric like precision, recall, f1, accuracy were analysed for each model and compared across all the models considered in this project.

STEP9. Confusion matrix was checked for all the models based on their best_score and conclusions were drawn for True Negative, True Positive, False Positive, False Negative.

STEP10. Feature importance graph was analysed to identify which features/variables are most contributors for employee to leave the company.

STEP11. The best performing model was finalised based on the project goal

---

Reference to Evaluate and Interpret Model performance

Evaluation metrics

- **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.

Confusion matrix

- **True Negative (TN)**: The upper-left quadrant displays the number of true negatives, the total count that classification model correctly predicted as False(0). In this case, the employees who didnt leave
- **False Positive (FP)**: The upper-right quadrant displays the number of false positives, the total count that classification model incorrectly predicted as True(1). In this case the classification model predicted the employee as 'left' but in reality employee 'stayed'
- **False Negative (FN)**: The lower-left quadrant displays the number of false negatives, the count that classification model incorrectly predicted as False(0). In this case the classification model incorrectly predicted an employee as 'stayed' but in reality that employee 'left'

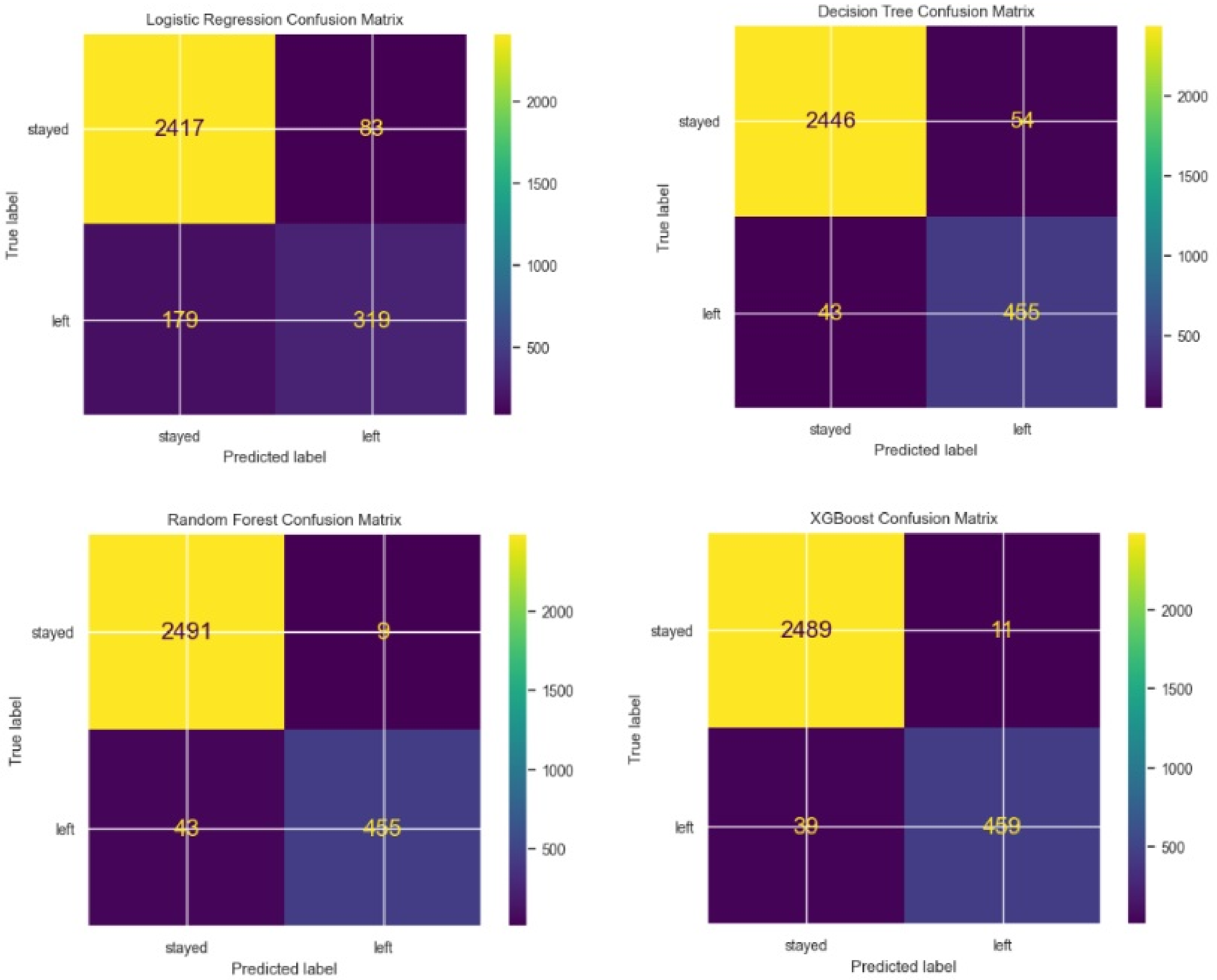
- **True Positive (TP)**: The lower-right quadrant displays the number of true positives, the count that classification model correctly predicted as True(1). In this case the classification model correctly predicted employees who left.

Feature importance graph

- This is a step to build a machine learning model.
- It involves calculating the score for all the input features (predictor variables) in a model to ectablish the importance of each feature in the decision-making process.
- The higher the score for a particular feature, the larger effect it has on the model prediction.
- The calculation is based on Gini gain. The amount of Gini impurity that was eliminated at each branch of decision tree.

Summary- Confusion Matrix

Comparison of Confusion matrix of all the models



Conclusion of Confusion Matrix

- Our focus is to reduce False Negative (Lower-left quadrant) as these are the employees who are predicted as they will stay but in reality they will leave.
- We find **XGBoost** model performing the best in predicting False Negative much better than other models.

Summary- Evaluation Metrics

In [90]:
```
# Comparison of Evaluation Matrics of all models
result
```

Out[90]:

	Model	Precision	Recall	F1	Accuracy
0	XGBoost	0.976596	0.921687	0.948347	0.983322
1	Random Forest	0.980603	0.913655	0.945946	0.982655
2	Decision Tree	0.893910	0.913655	0.903674	0.967645
3	Logistic Regression	0.793532	0.640562	0.708889	0.912608

Conclusion of Evaluation Metrics

- Our focus is more on 'Recall' and 'f1' metrics.
- **XGBoost** Model depicts best performing values for Recall and f1 metrics.
- This XGBoost Model with tuned hyperparameters provides
  - **Precision score** of **97.66%**
  - **Recall score** of **92.17%**
  - **F1 score** of **94.83%**
  - **Accuracy score** of **98.33%**

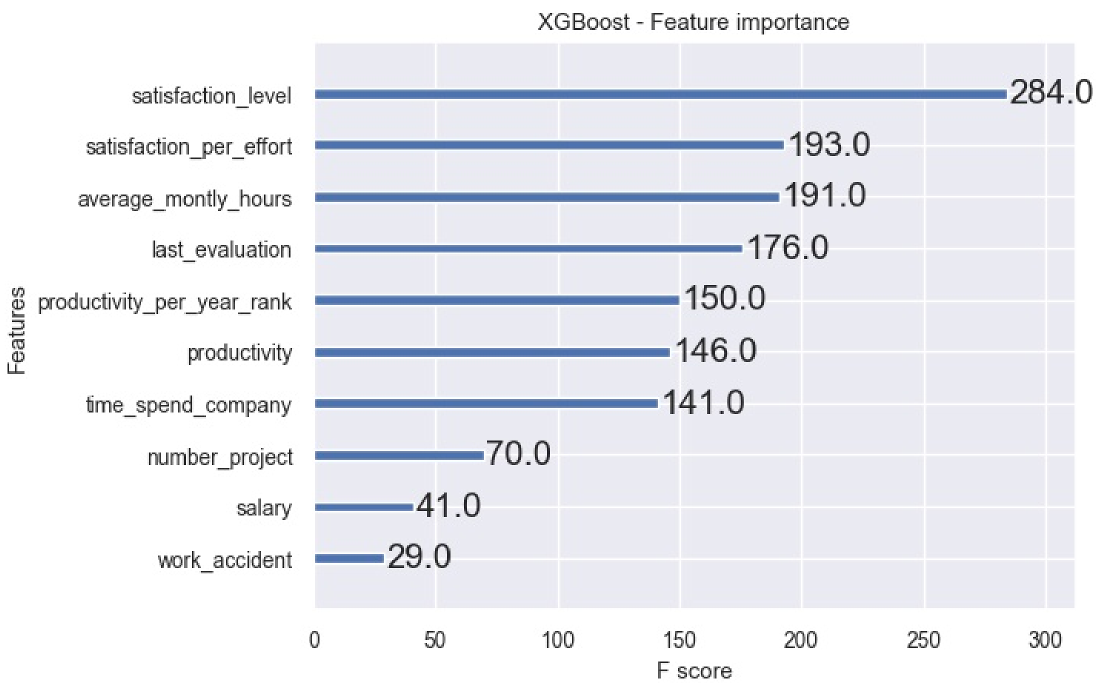
Summary- Feature importance

In [106…

```
# Display feature importances
plot_importance(xgb_cv.best_estimator_)
plt.title('XGBoost - Feature importance')
```

Out[106…

```
Text(0.5, 1.0, 'XGBoost - Feature importance')
```



Top Contributing Features from XGBoost model

- **satisfaction_level** : This is the most important factor and is the highest contributor for employees leaving the company. We have observed that satisfaction level of employees reduces drastically for the employees who left.
- **satisfaction_per_effort** : This is a feature transformed variable. This variable is calculated as satisfaction level of employee divided by their average monthly hours and their Tanure in the company. This variable was created to understand at what cost of employee's personal time was occupied against the satisfaction score mentioned. This feature emphasize that satisfaction score is inversely proportonate to the extra working hours and for a prolonged period, thereby causing employee to leave.
- **average_monthly_hours** : This feature shows that higher the average monthly hours of an employee, more likely the employee will leave the company. Its also observed that 64% employees work more than 174hours per month (8hrs per day).
- **last_evaluation** : This feature shows that poor promotions are resulting to employees to leave. Total 1.7% employees were promoted as observed from the sample dataset provided.
- **productivity_per_year_rank** : This is a feature transformed variable. This feature is calculated as product of number of project multiplied by average monthly hours and divided by the total Tanure in the company. This feature was created to understand which employees devoted more efforts for the company during the whole tanure. This feature shows that when employees working hours increases due to greater number of projects and this continues for a prolong time, then employees tend to leave the company.

Next step to improve model performance

- Model performance can be improve with larger sample dataset. 20% of the sample data provided had duplicate entries. Present dataset after cleaning contained only 11,991 unique employee details. Which even becomes more smaller after spliting the dataset to train/test.
- In the present project, 75:25 ratio was taken for train/test set. We can try to check if model performance improves with train/test ratio of 80:20
- Target variable data was imbalanced with 83.4% : 16.6%. Model performance can be checked with oversampling the target variable "1" which is "left" and validate if there is performance improvement.
- 'left' column has employees who left the company and are denoted as '1', but this also consists of employees who have been retrenched/sacked by Salifort Motors. There should be separate column or separate identification for employees left by themselves and employees who were retrenched/sacked by Salifort Motors.
- Sample dataset provided does not show monthly bifurcation or has no indication on datetime. This will help to understand how many projects is an employee engaged with at one particular month with respect to the duration when he is not engaged to any project.

Additional conclusion from Data and Model

- Total **1.7% employees** were promoted as observed from the sample dataset provided.
- **64% employees** work more than 8hrs per day, 5 days a week.
- In the plot, Employee left vs Time spent in the company, it shows that most of the employees are in the company for past 2-3 years. Thereafter **from 4th year onwards**, employees retainability is less and employees tend to leave the company
- From the plot: satisfaction level vs number of projects, Satisfaction level drastically drops when employees are engaged in **more than 5 projects**
- **number_projects= 3** is the most optimal number of projects for employees as left % in this category is least.
- **salary** from Feature importance shows that it is not the most contributor for employee leaving company. it is not even in the top 7 list of contributors for employee leaving the company.
- **High average monthly hour** is one of the top 3 contributors for employee leaving the company

---

Business recommendations

1. **High Average monthly hours** is significantly contributing to employees leaving company. There are 64% employees working more than 8hrs per day, 5 days a week.

   Recommendation:

   - Salifort Motors HR department to further analyse what is the cause behind employees doing high overtime.
   - As temporary measure, external consultants on contract basis should be put on duty to reduce employee extra working hours

2. **High number of project, for prolonged time** is significantly contributing to employees leaving company. Satisfaction level drastically drops when employees are engaged in more than 5 projects. number_projects= 3 is the most optimal number of projects for employees as left % in this category is least.

   Recommendation:

   - Salifort Motors HR department to check why employees are allocated huge number of projects (>4), if this is purely due to small number of skilled manpower available or there is shortage of total number of workforce needed.
   - For immediate solution, Salifort Motors HR team should identify the departments and the role of the individual employee who are engaged with higher number of projects and allocate contractual staff under their supervision to reduce the burden on individual

employee.

Ethical Considerations

- This model should not be used as a tool to promote employees or to provide incentives to the employees based on the results that show employees who have high probability to leave.
- Providing incentive to employees who have higher chances of leaving the company or depriving worthy and capable employees from giving incentives to those who are predicted to stay and will not leave the company: may bring short term benefits. Though in long term this can bring catastrophic effect
- This model should be used as a guideline to identify bottleneck where the work flow is going wrong and accordingly take preventive measure before actual wreckage

In [ ]:

In [ ]:

In [ ]:

In [ ]: