Image credit: VantageCircle (https://blog.vantagecircle.com/employee-attrition/)

WELCOME!

Welcome to "***Employee Churn Analysis***" study. In this study, we will be able to build our own classification models for a variety of business settings.

Also we will learn what is Employee Churn?, How it is different from customer churn, Exploratory data analysis and visualization of employee churn dataset using ***matplotlib*** and ***seaborn***, *model building and evaluation using python* **scikit-learn*** package.

We will be able to implement classification techniques in Python. Using Scikit-Learn allowing us to successfully make predictions with the Random Forest, Gradient Descent Boosting , KNN and CatBoost algorithms.

At the end of the project, we will have the opportunity to deploy your model using *Streamlit*.

1 - DATA

In this project we have HR data of a company. A study is requested from us to predict which employee will churn by using this data.

The HR dataset has 14,999 samples with various information about the employees. In the given dataset, we have two types of employee one who stayed and another who left the company. This given dataset will be used to predict when employees are going to quit by understanding the main drivers of employee churn.

For a better understanding and more information, please refer to DataCamp (https://www.datacamp.com/community/tutorials/predicting-employee-churn-python) and Kaggle Website (https://www.kaggle.com/c/employee-churn-prediction/data)

1.1 Context

"Analyze employee churn. Find out why employees are leaving the company, and learn to predict who will leave the company.." DataCamp (https://www.datacamp.com/community/tutorials/predicting-employee-churn-python)

Employee turn-over (also known as "employee churn") is a costly problem for companies. The true cost of replacing an employee can often be quite large. A study by the Center for American Progress found that companies typically pay about one-fifth of an employee’s salary to replace that employee, and the cost can significantly increase if executives or highest-paid employees are to be replaced. In other words, the cost of replacing employees for most employers remains significant. This is due to the amount of time spent to interview and find a replacement, sign-on bonuses, and the loss of productivity for several months while the new employee gets accustomed to the new role.

In the past, most of the focus on the "rates" such as attrition rate and retention rates. HR Managers compute the previous rates try to predict the future rates using data warehousing tools. These rates present the aggregate impact of churn, but this is the half picture. Another approach can be the focus on individual records in addition to aggregate.

There are lots of case studies on customer churn are available. In customer churn, you can predict who and when a customer will stop buying. Employee churn is similar to customer churn. It mainly focuses on the employee rather than the customer. Here, you can predict who, and when an employee will terminate the service. Employee churn is expensive, and incremental improvements will give significant results. It will help us in designing better retention plans and improving employee satisfaction.

1.2 About The Features

We can describe 10 attributes (features) in detail as:

- ***satisfaction_level :*** It is employee satisfaction point, which ranges from 0-1.
- ***last_evaluation :*** It is evaluated performance by the employer, which also ranges from 0-1.
- ***number_projects :*** How many of projects assigned to an employee?
- ***average_monthly_hours:*** How many hours in averega an employee worked in a month?
- ***time_spent_company :*** time_spent_company means employee experience. The number of years spent by an employee in the company.
- ***work_accident :*** Whether an employee has had a work accident or not.
- ***promotion_last_5years:*** Whether an employee has had a promotion in the last 5 years or not.
- ***Departments :*** Employee's working department/division.
- ***Salary :*** Salary level of the employee such as low, medium and high.
- ***left :*** Whether the employee has left the company or not.

1.3 What The Problem Is

First of all, to observe the structure of the data, outliers, missing values and features that affect the target variable, we must use exploratory data analysis and data visualization techniques.

Then, we must perform data pre-processing operations such as ***Scaling*** and ***Label Encoding*** to increase the accuracy score of Gradient Descent Based or Distance-Based algorithms. we are asked to perform ***Cluster Analysis*** based on the information you obtain during exploratory data analysis and data visualization processes.

The purpose of clustering analysis is to cluster data with similar characteristics. We are asked to use the ***K-means*** algorithm to make cluster analysis. However, you must provide the K-means algorithm with information about the number of clusters it will make predictions. Also, the data we apply to the K-means algorithm must be scaled. In order to find the optimal number of clusters, we are asked to use the ***Elbow method***. Briefly, try to predict the set to which individuals are related by using K-means and evaluate the estimation results.

Once the data is ready to be applied to the model, we must ***split the data into train and test***. Then build a model to predict whether employees will churn or not. Train our models with our train set, test the success of our model with our test set.

Try to make our predictions by using the algorithms ***Gradient Boosting Classifier**, **K Neighbors Classifier**, **Random Forest Classifier**, *and* **CatBoost Classifier. We can use the related modules of the** *scikit-learn*** library. We can use scikit-learn ***Confusion Metrics*** module for accuracy calculation. We can use the ***Yellowbrick*** module for model selection and visualization.

In the final step, we will deploy your model using Streamlit tool.

1.4 Project Structure & Tasks

1. Exploratory Data Analysis

- Importing Modules
- Loading Dataset
- Data Insigts

2. Data Visualization

- Employees Left
- Determine Number of Projects
- Determine Time Spent in Company
- Subplots of Features

3. Data Pre-Processing

- Scaling
- Label Encoding

4. Cluster Analysis

- Find the optimal number of clusters (k) using the elbow method for for K-means.
- Determine the clusters by using K-Means then Evaluate predicted results.

5. Model Building

- Split Data as Train and Test set
- Built Gradient Boosting Classifier, Evaluate Model Performance and Predict Test Data
- Built K Neighbors Classifier and Evaluate Model Performance and Predict Test Data
- Built Random Forest Classifier and Evaluate Model Performance and Predict Test Data

6. Model Deployement

- Save and Export the Model as .pkl
- Save and Export Variables as .pkl

2 - LIBRARIES NEEDED IN THE STUDY

In [1]:
```
# 1-Import Libraies

import pandas_profiling
# import pyforest

import ipywidgets
from ipywidgets import interact

import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.ticker as mticker
import squarify as sq

# Importing plotly and cufflinks in offline mode
import plotly
import plotly.express as px
import cufflinks as cf
import plotly.graph_objs as go
import plotly.offline as py
from plotly.offline import iplot
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

# !pip install termcolor
import colorama
from colorama import Fore, Style  # makes strings colored
from termcolor import colored
from termcolor import cprint

from wordcloud import WordCloud

import scipy.stats as stats
from scipy.cluster.hierarchy import linkage, dendrogram
import statsmodels.api as sm
import statsmodels.formula.api as smf
import missingno as msno

import datetime as dt
from datetime import datetime


import optuna

from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.compose import make_column_transformer, ColumnTransformer
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif, f_regression, mutual_info_regression
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet, LogisticRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import make_scorer, precision_score, precision_recall_curve
from sklearn.metrics import  roc_auc_score, roc_curve, f1_score, accuracy_score, recall_score
from sklearn.metrics import silhouette_samples,silhouette_score
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold, KFold, cross_val_predict, train_test_split
from sklearn.model_selection import StratifiedKFold, GridSearchCV, cross_val_score, cross_validate

from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.preprocessing import MinMaxScaler, scale, StandardScaler, RobustScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, PolynomialFeatures, PowerTransformer
from sklearn.svm import SVR, SVC
from sklearn.tree import DecisionTreeClassifier, plot_tree

from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBRegressor, XGBClassifier, plot_importance

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
warnings.warn("this will not show")

# Figure&Display options
plt.rcParams["figure.figsize"] = (10,6)
pd.set_option('max_colwidth',200)
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 200)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

```
C:\Users\aryaa\AppData\Local\Temp\ipykernel_18572\3784034954.py:3: DeprecationWarning: `import pandas_profiling` is going to be deprecated by April 1st. Please use `import ydata_profiling` instead.
  import pandas_profiling
```

2.1 User Defined Functions

We have defined some useful user defined functions.

In [2]:
```
## Some Useful Functions

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

def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending = False)
    missing_percent = (df.isnull().sum() / df.isnull().count()).sort_values(ascending = False)
    missing_values = pd.concat([missing_number, missing_percent], axis = 1, keys = ['Missing_Number', 'Missing_Percent'])
    return missing_values[missing_values['Missing_Number'] > 0]

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

def first_looking(df):
    print(colored("Shape:", attrs=['bold']), df.shape,'\n',
          colored('*'*100, 'red', attrs = ['bold']),
          colored("\nInfo:\n", attrs = ['bold']), sep = '')
    print(df.info(), '\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Number of Uniques:\n", attrs = ['bold']), df.nunique(),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Missing Values:\n", attrs=['bold']), missing_values(df),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("All Columns:", attrs = ['bold']), list(df.columns),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')

    df.columns = df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')
    print(colored("Columns after rename:", attrs = ['bold']), list(df.columns),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Columns after rename:", attrs = ['bold']), list(df.columns),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Descriptive Statistics \n", attrs = ['bold']), df.describe().round(2),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '') # Gives a statstical breakdown of the data.
    print(colored("Descriptive Statistics (Categorical Columns) \n", attrs = ['bold']), df.describe(include = object).T,'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '') # Gives a statstical breakdown of the data.

def multicolinearity_control(df):
    feature = []
    collinear = []
    for col in df.corr().columns:
        for i in df.corr().index:
            if (abs(df.corr()[col][i]) > .9 and abs(df.corr()[col][i]) < 1):
                    feature.append(col)
                    collinear.append(i)
                    print(colored(f"Multicolinearity alert in between:{col} - {i}",
                                  "red", attrs = ['bold']), df.shape,'\n',
                                  colored('*'*100, 'red', attrs = ['bold']), sep = '')

def duplicate_values(df):
    print(colored("Duplicate check...", attrs = ['bold']), sep = '')
    print("There are", df.duplicated(subset = None, keep = 'first').sum(), "duplicated observations in the dataset.")
    duplicate_values = df.duplicated(subset = None, keep = 'first').sum()
    if duplicate_values > 0:
        df.drop_duplicates(keep = 'first', inplace = True)
        print(duplicate_values, colored(" Duplicates were dropped!"),'\n',
              colored('*'*100, 'red', attrs = ['bold']), sep = '')
#     else:
#         print(colored("There are no duplicates"),'\n',
#               colored('*'*100, 'red', attrs = ['bold']), sep = '')

# def drop_columns(df, drop_columns):
#     if drop_columns != []:
#         df.drop(drop_columns, axis = 1, inplace = True)
#         print(drop_columns, 'were dropped')
#     else:
#         print(colored('We will now check the missing values and if necessary, the related columns will be dropped!', attrs = ['bold']),'\n',
#               colored('*'*100, 'red', attrs = ['bold']), sep = '')

def drop_null(df, limit):
    print('Shape:', df.shape)
    for i in df.isnull().sum().index:
        if (df.isnull().sum()[i] / df.shape[0]*100) > limit:
            print(df.isnull().sum()[i], 'percent of', i ,'null and were dropped')
            df.drop(i, axis = 1, inplace = True)
            print('new shape:', df.shape)
    print('New shape after missing value control:', df.shape)

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

# To view summary information about the columns

def first_look(col):
    print("column name    : ", col)
    print("--------------------------------")
    print("Per_of_Nulls   : ", "%", round(df[col].isnull().sum() / df.shape[0]*100, 2))
    print("Num_of_Nulls   : ", df[col].isnull().sum())
    print("Num_of_Uniques : ", df[col].nunique())
    print("Duplicates     : ", df.duplicated(subset = None, keep = 'first').sum())
    print(df[col].value_counts(dropna = False))

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

def fill_most(df, group_col, col_name):
    '''Fills the missing values with the most existing value (mode) in the relevant column according to single-stage grouping'''
    for group in list(df[group_col].unique()):
        cond = df[group_col] == group
        mode = list(df[cond][col_name].mode())
        if mode != []:
            df.loc[cond, col_name] = df.loc[cond, col_name].fillna(df[cond][col_name].mode()[0])
        else:
            df.loc[cond, col_name] = df.loc[cond, col_name].fillna(df[col_name].mode()[0])
    print("Number of NaN : ",df[col_name].isnull().sum())
    print("------------------")
    print(df[col_name].value_counts(dropna = False))

###############################################################################
# bar grafiğindeki değerlerin gösterilmesi
# show values in bar graphic
def show_values_on_bars(axs):
    def _show_on_single_plot(ax):
        for p in ax.patches:
            _x = p.get_x() + p.get_width() / 2
            _y = p.get_y() + p.get_height()
            value = '{:.2f}'.format(p.get_height())
            ax.text(_x, _y, value, ha="center")
    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)
```

3 - ANALYSIS

3.1 Loading & Reading the Data

Let's first load the required HR dataset using pandas's "read_csv" function.

In [3]:
```
df0 = pd.read_csv('HR_Dataset.csv')
df = df0.copy()
df.head(3)
```

Out[3]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departments	salary
0	0.380	0.530	2	157	3	0	1	0	sales	low
1	0.800	0.860	5	262	6	0	1	0	sales	medium
2	0.110	0.880	7	272	4	0	1	0	sales	medium

In [4]:
```
df.tail(3)
```

Out[4]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departments	salary
14996	0.370	0.530	2	143	3	0	1	0	support	low
14997	0.110	0.960	6	280	4	0	1	0	support	low
14998	0.370	0.520	2	158	3	0	1	0	support	low

In [5]:
```
df.sample(3)
```

Out[5]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departments	salary
13182	0.800	0.800	4	263	4	0	0	0	support	medium
8435	0.570	0.370	3	108	4	0	0	0	technical	low
5228	0.270	0.450	3	239	4	0	0	0	technical	low

4 - DATA CLEANING & EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis is an initial process of analysis, in which you can summarize characteristics of data such as pattern, trends, outliers, and hypothesis testing using descriptive statistics and visualization.

4.1 - A General Look at the Data

In [6]:
```
first_looking(df)
duplicate_values(df)
print(colored("Shape:", attrs = ['bold']), df.shape,'\n', colored('*'*100, 'red', attrs = ['bold']))
```

```
Shape:(14999, 10)
****************************************************************************************************
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   Departments            14999 non-null  object 
 9   salary                 14999 non-null  object 
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
None
****************************************************************************************************
Number of Uniques:
satisfaction_level        92
last_evaluation           65
number_project             6
average_montly_hours     215
time_spend_company         8
Work_accident              2
left                       2
promotion_last_5years      2
Departments               10
salary                     3
dtype: int64
****************************************************************************************************
Missing Values:
Empty DataFrame
Columns: [Missing_Number, Missing_Percent]
Index: []
****************************************************************************************************
All Columns:['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', 'time_spend_company', 'Work_accident', 'left', 'pro
motion_last_5years', 'Departments ', 'salary']
****************************************************************************************************
Columns after rename:['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', 'time_spend_company', 'work_accident', 'le
ft', 'promotion_last_5years', 'departments_', 'salary']
****************************************************************************************************
Columns after rename:['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', 'time_spend_company', 'work_accident', 'le
ft', 'promotion_last_5years', 'departments_', 'salary']
****************************************************************************************************
Descriptive Statistics
       satisfaction_level  last_evaluation  number_project  \
count           14999.000        14999.000       14999.000   
mean                0.610            0.720           3.800   
std                 0.250            0.170           1.230   
min                 0.090            0.360           2.000   
25%                 0.440            0.560           3.000   
50%                 0.640            0.720           4.000   
75%                 0.820            0.870           5.000   
max                 1.000            1.000           7.000   

       average_montly_hours  time_spend_company  work_accident      left  \
count             14999.000           14999.000      14999.000 14999.000   
mean                201.050               3.500          0.140     0.240   
std                  49.940               1.460          0.350     0.430   
min                  96.000               2.000          0.000     0.000   
25%                 156.000               3.000          0.000     0.000   
50%                 200.000               3.000          0.000     0.000   
75%                 245.000               4.000          0.000     0.000   
max                 310.000              10.000          1.000     1.000   

       promotion_last_5years  
count              14999.000  
mean                   0.020  
std                    0.140  
min                    0.000  
25%                    0.000  
50%                    0.000  
75%                    0.000  
max                    1.000  
****************************************************************************************************
Descriptive Statistics (Categorical Columns)
              count unique    top  freq
departments_  14999     10  sales  4140
salary        14999      3    low  7316
****************************************************************************************************
Duplicate check...
There are 3008 duplicated observations in the dataset.
3008 Duplicates were dropped!
****************************************************************************************************
Shape: (11991, 10)
 ****************************************************************************************************
```

According to the basic examinations on the dataset;

- We have a classification problem.
- We are going to make classification on the target variable "left".
- And we will build a model to get the best classification on the "left" column.
- Because of that we are going to look at the balance of "left" column.
- The dataset has 10 columns and 11991 observations after dropping of duplicated observations.
- 8 columns contain numerical values and 2 columns contain categorical values.
- There seems to be no missing value.

In [7]:
```python
df.columns
```

Out[7]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'work_accident', 'left',
       'promotion_last_5years', 'departments_', 'salary'],
      dtype='object')
```

In [8]:
```python
df.rename({'departments_': 'department'}, axis=1, inplace=True)
df.head(1)
```

Out[8]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	left	promotion_last_5years	department	salary
0	0.380	0.530	2	157	3	0	1	0	sales	low

In [9]:
```python
df = df[['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'work_accident',
       'promotion_last_5years', 'department', 'salary', 'left']]
df.head(1)
```

Out[9]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	department	salary	left
0	0.380	0.530	2	157	3	0	0	sales	low	1

- I want to move the 'left' column, which is my target column, from where it is to the end. In this way, I will work more comfortably psychologically :))

4.2 - Examination of Features and Data Insights

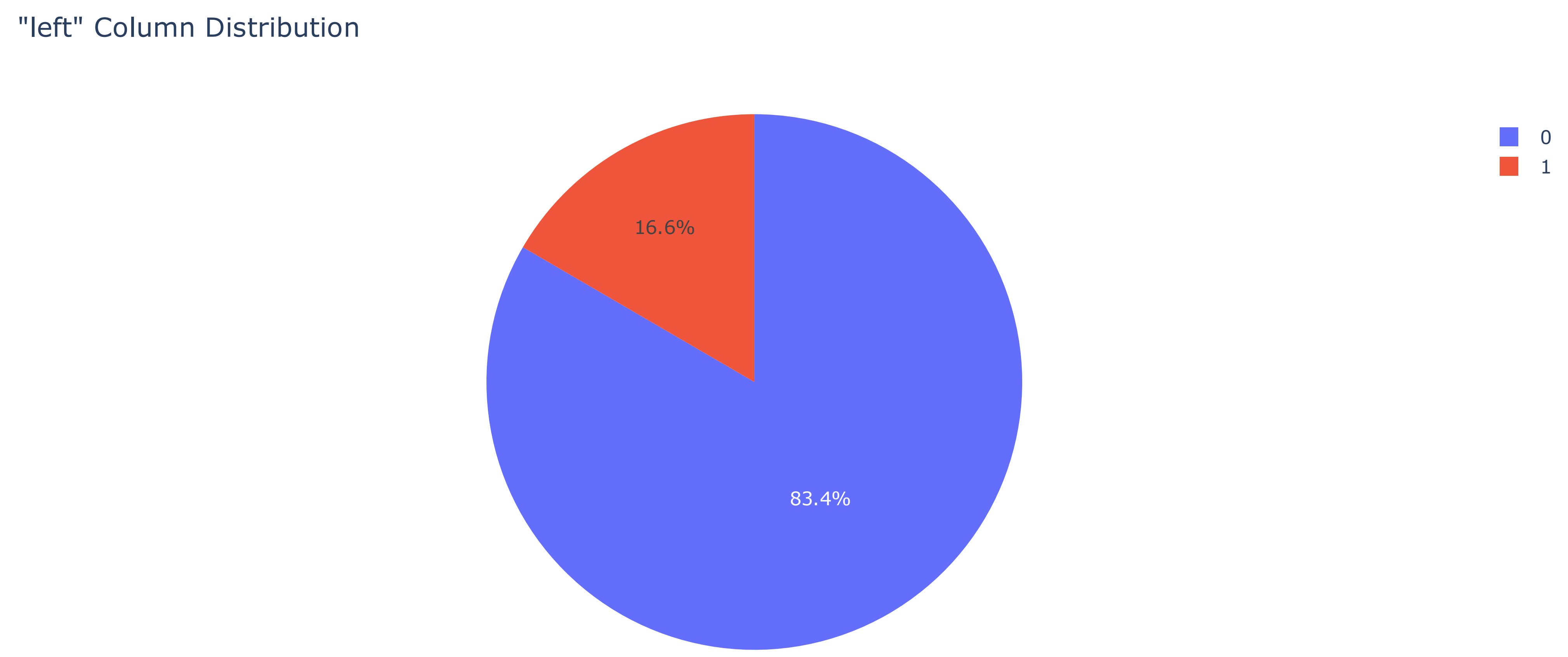
In the given dataset, we have two types of employee one who stayed and another who left the company. So, we can divide data into two groups and compare their characteristics. Here, we can find the average of both the groups using groupby() and mean() function.

'Left' Column-Target Column

In [10]:
```
cprint("Have a First Look to 'left' Column",'green')
first_look('left')
```

```
Have a First Look to 'left' Column
column name    :  left
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  2
Duplicates     :  0
0    10000
1     1991
Name: left, dtype: int64
```

In [11]:
```
import plotly
import plotly.express as px
fig = px.pie(df, values = df['left'].value_counts(),
             names = (df['left'].value_counts()).index,
             title = '"left" Column Distribution')
fig.show()
```



In [12]:
```
y = df['left']
print(f'Percentage of left-1: % {round(y.value_counts(normalize=True)[1]*100,2)} --> \
({y.value_counts()[1]} observations for left-1)\nPercentage of left-0: % {round(y.value_counts(normalize=True)[0]*100,2)} --> ({y.value_counts()[0]
```

```
Percentage of left-1: % 16.6 --> (1991 observations for left-1)
Percentage of left-0: % 83.4 --> (10000 observations for left-0)
```

- 'left' column has binary type values.
- We have an imbalanced data.
- Almost 17% of the employees didn't continue with the company and left.
- 1991 employees left.
- Almost 83% of the employees continue with the company and didn't leave.
- 10000 employees didn't leave.

In [13]:
```
df.groupby('left').mean()
```

Out[13]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years
left							
0	0.667	0.716	3.787	198.943	3.262	0.174	0.019
1	0.440	0.722	3.883	208.162	3.881	0.053	0.004

In [14]:
```
cprint('Dataset describe results according to the "left==1" condition','green', 'on_black')
df[df['left'] == 1].describe().T.style.background_gradient(subset = ['mean','min','50%', 'max'], cmap = 'RdPu')
```

```
Dataset describe results according to the "left==1" condition
```

Out[14]:

	count	mean	std	min	25%	50%	75%	max
satisfaction_level	1991.000000	0.440271	0.265207	0.090000	0.110000	0.410000	0.730000	0.920000
last_evaluation	1991.000000	0.721783	0.197436	0.450000	0.520000	0.790000	0.910000	1.000000
number_project	1991.000000	3.883476	1.817139	2.000000	2.000000	4.000000	6.000000	7.000000
average_montly_hours	1991.000000	208.162230	61.295145	126.000000	146.000000	226.000000	262.500000	310.000000
time_spend_company	1991.000000	3.881467	0.974041	2.000000	3.000000	4.000000	5.000000	6.000000
work_accident	1991.000000	0.052737	0.223565	0.000000	0.000000	0.000000	0.000000	1.000000
promotion_last_5years	1991.000000	0.004018	0.063277	0.000000	0.000000	0.000000	0.000000	1.000000
left	1991.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [15]:
```
cprint('Dataset describe results according to the "left==0" condition','green', 'on_black')
df[df['left'] == 0].describe().T.style.background_gradient(subset = ['mean','min','50%', 'max'], cmap = 'RdPu')
```

```
Dataset describe results according to the "left==0" condition
```

Out[15]:

	count	mean	std	min	25%	50%	75%	max
satisfaction_level	10000.000000	0.667365	0.217082	0.120000	0.540000	0.690000	0.840000	1.000000
last_evaluation	10000.000000	0.715667	0.161919	0.360000	0.580000	0.710000	0.850000	1.000000
number_project	10000.000000	3.786800	0.981755	2.000000	3.000000	4.000000	4.000000	6.000000
average_montly_hours	10000.000000	198.942700	45.665507	96.000000	162.000000	198.000000	238.000000	287.000000
time_spend_company	10000.000000	3.262000	1.367239	2.000000	2.000000	3.000000	4.000000	10.000000
work_accident	10000.000000	0.174500	0.379558	0.000000	0.000000	0.000000	0.000000	1.000000
promotion_last_5years	10000.000000	0.019500	0.138281	0.000000	0.000000	0.000000	0.000000	1.000000
left	10000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

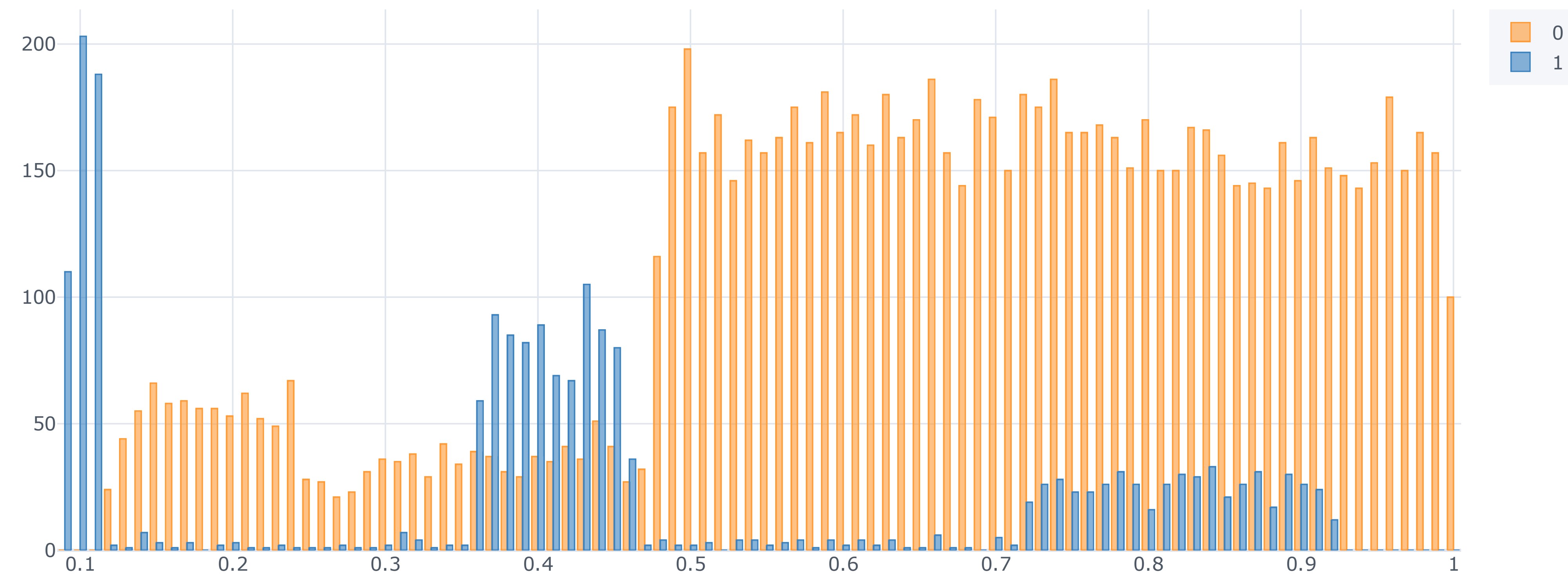
'satisfaction_level' Column

In [16]:
```
print("Have a First Look to 'left' Column")
first_look('satisfaction_level')
```

```
Have a First Look to 'left' Column
column name    :  satisfaction_level
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  92
Duplicates     :  0
0.740    214
0.100    203
0.730    201
0.500    200
0.720    199
0.840    199
0.830    196
0.770    194
0.780    194
0.660    192
0.890    191
0.110    188
0.750    188
0.760    188
0.910    187
0.800    186
0.590    185
0.630    184
0.820    180
0.570    179
0.960    179
0.690    178
0.850    177
0.490    177
0.790    177
0.810    176
0.610    176
0.870    176
0.700    176
0.900    172
0.520    172
0.650    171
0.860    170
0.600    167
0.560    166
0.540    166
0.980    165
0.640    164
0.920    163
0.580    162
0.620    162
0.880    160
0.510    160
0.550    159
0.670    158
0.990    157
0.950    153
0.710    152
0.970    150
0.530    150
0.930    148
0.680    145
0.940    143
0.430    141
0.440    138
0.370    130
0.400    126
0.450    121
0.480    120
0.380    116
0.390    111
0.090    110
0.420    108
0.410    104
1.000    100
0.360     98
0.150     69
0.240     68
0.460     63
0.210     63
0.140     62
0.170     62
0.160     59
0.190     58
0.180     56
0.200     56
0.220     53
0.230     51
0.130     45
0.340     44
0.310     42
0.320     42
0.300     38
0.350     36
0.470     34
0.290     32
0.330     30
0.250     29
0.260     28
0.120     26
0.280     24
0.270     23
Name: satisfaction_level, dtype: int64
```

In [17]:
```
pd.crosstab(df['satisfaction_level'], df['left']).iplot(kind='bar', title = 'satisfaction_level and left')
```

satisfaction_level and left



Export to plot.ly »

- Although it comes to mind that there should be a linear relationship between 'satisfaction_level' and 'left', it does not look like this on the graph.
- Those with a 'satisfaction_level' value of around 0.1 are very likely to 'left'.
- There is a significant increase in the number of those whose 'satisfaction_level' value is between 3.5 and 4.5 and 'left'. In fact, the number of left ones exceeds the notleft ones.
- When the 'satisfaction_level' value is between 7 and 9, there is an increase in the number of those left.

- Normally we expect low satisfaction level for the employees who has left, so the part near to 0 on the x-axis is make sense.
- Besides a group of employee who are not very decisive about their satisfaction level have also been left the company. This group may need extra motivation for employee loyalty. Because they are not so clear in their assesments about their future in the company.
- Also a group of employee whose satisfaction level is above the avarage have been left the company. This does not make sense so this must be investigated deeply.

There may be some other issues:

a. The method of gathering this information may be wrong. So the assessment of satisfaction level and the resignings may not be directly proportional.

b. The assessment may not be up to date. By the time the satisfaction level may be decreased so at the real time the satisfaction level of all resigning employees may be close to 0.

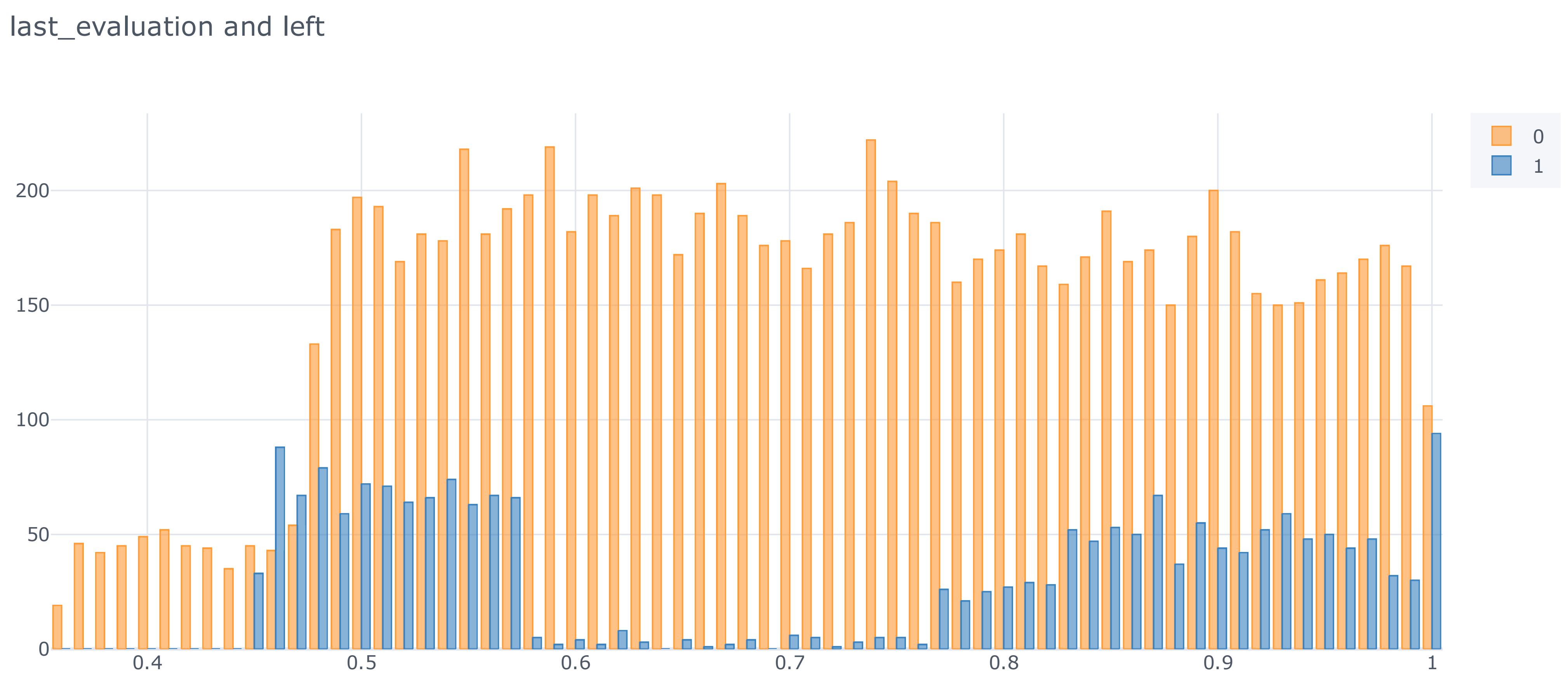
c. Some of the employees may have hidden their true feelings.

'last_evaluation' Column

In [18]:
```
cprint("Have a First Look to 'last_evaluation' Column",'green', 'on_black')
first_look('last_evaluation')
```

```
Have a First Look to 'last_evaluation' Column
column name    :  last_evaluation
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  65
Duplicates     :  0
0.550    281
0.500    269
0.510    264
0.570    258
0.540    252
0.560    248
0.530    247
0.850    244
0.900    244
0.490    242
0.870    241
0.890    235
0.520    233
0.740    227
0.910    224
0.590    221
0.860    219
0.840    218
0.970    218
0.770    212
0.480    212
0.830    211
0.950    211
0.810    210
0.750    209
0.930    209
0.960    208
0.980    208
0.920    207
0.670    205
0.630    204
0.580    203
0.800    201
1.000    200
0.610    200
0.940    199
0.640    198
0.620    197
0.990    197
0.790    195
0.820    195
0.680    193
0.760    192
0.660    191
0.730    189
0.880    187
0.600    186
0.700    184
0.720    182
0.780    181
0.650    176
0.690    176
0.710    171
0.460    131
0.470    121
0.450     78
0.410     52
0.400     49
0.370     46
0.390     45
0.420     45
0.430     44
0.380     42
0.440     35
0.360     19
Name: last_evaluation, dtype: int64
```

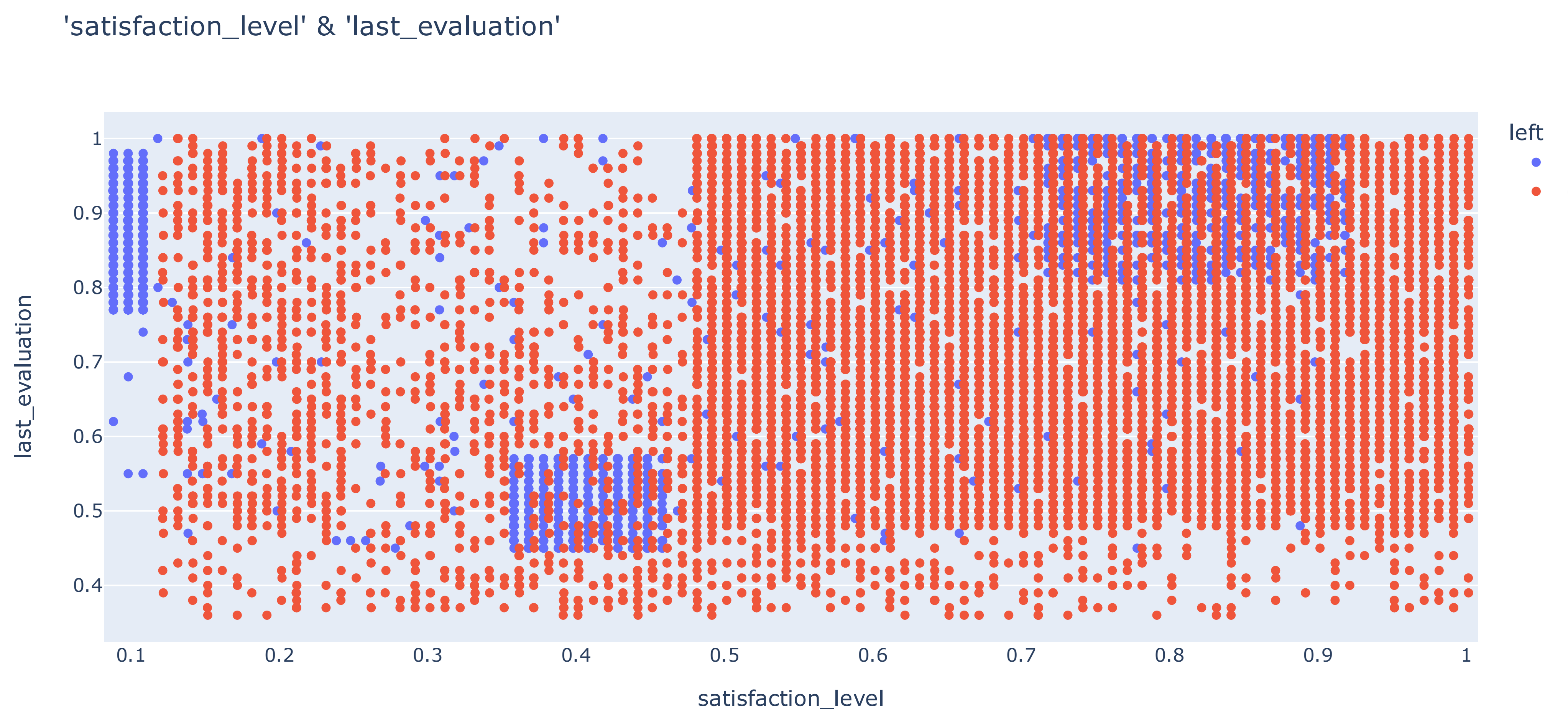
In [19]:
```
pd.crosstab(df['last_evaluation'], df['left']).iplot(kind='bar', title = 'last_evaluation and left')
```



Export to plot.ly »

- Most of the employees have been assessed above 0.4.
- There is a local increase between 0.45-0.6 and 0.8-1 in 'last_evaluation' values, as in 'satisfaction_level' values. There is an increase in the number of people who quit their jobs in these intervals
- Intensive work may cause the resign of high evaluated employees (second group). Because employer will be happy with performance of these staff, however it will be a burden for employee.

In [20]:
```
fig = px.strip(df, x = 'satisfaction_level', y = 'last_evaluation', color = 'left',
               title = "'satisfaction_level' & 'last_evaluation'")
fig.show()
```



It becomes meaningful when the satisfaction level of employees and the evaluation of the employer shown together.

As seen in the graph; the resigning employees are grouping in three diferent clusters.

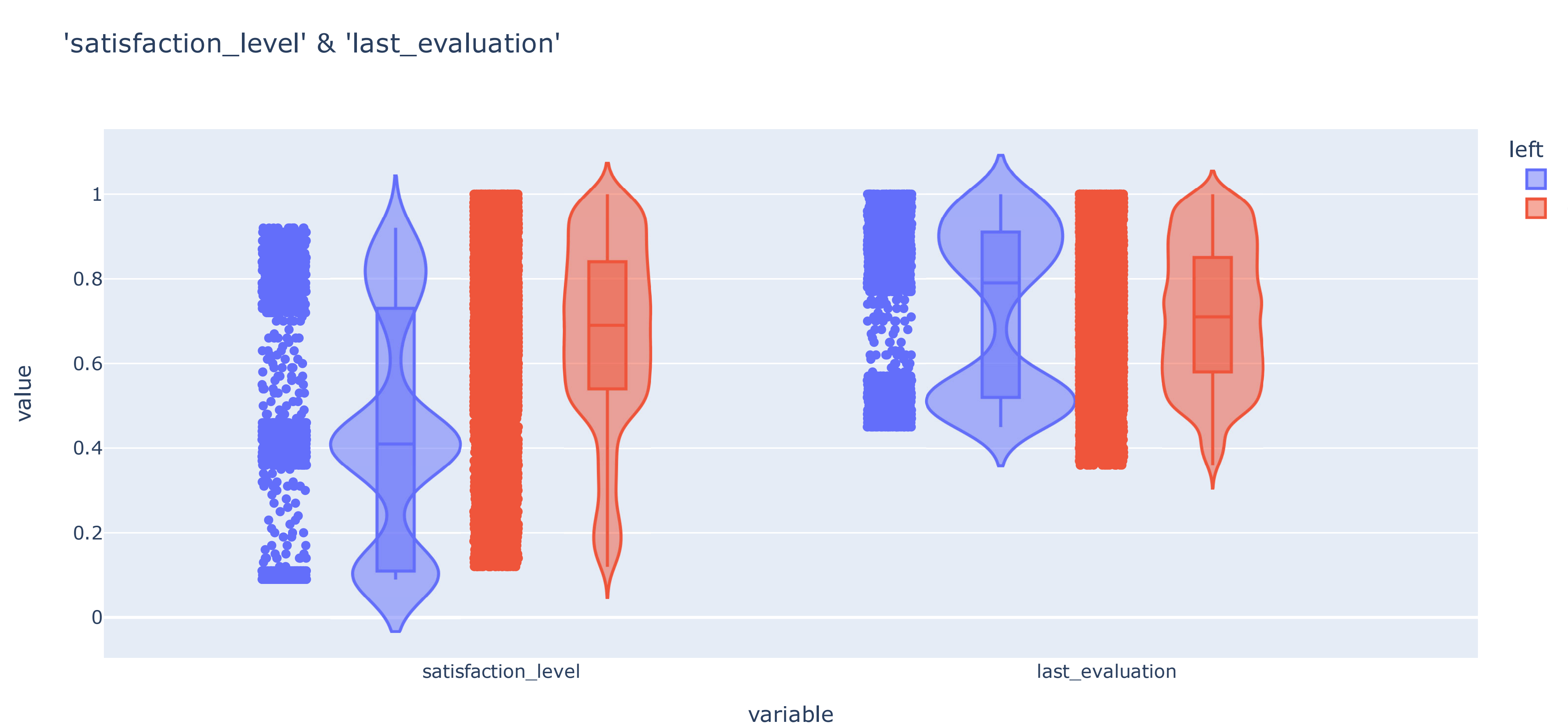
1. First group has a satisfaction level of 0.4 and last evaluation of 0.5. This group has not a clear idea about the company and the employer does not have a clear assessment about them. Other features affecting this group must to be investigated. What are the main questions of this group? Why they are confusing? What are the pros and cons of the company for these group? and so on...

2. The second group has a low satisfaction even if the employer evaluated them with high degrees. Then what can be the main problem of this group?

Intensive work with a low salary may affect this group. Or intensive work without promotion may cause to leave. On the next steps workload and motivation factors of this group have to be investigated.

3. The third group has a high satisfaction level and evaluation point as well. The density of this group is fewer than the others. The issues that triger the leave of this group need to be investigated.

In [21]:
```
fig = px.violin(df[['satisfaction_level', 'last_evaluation', 'left']], color = 'left', box = True, points='all',
                title = "'satisfaction_level' & 'last_evaluation'")
fig.show()
```

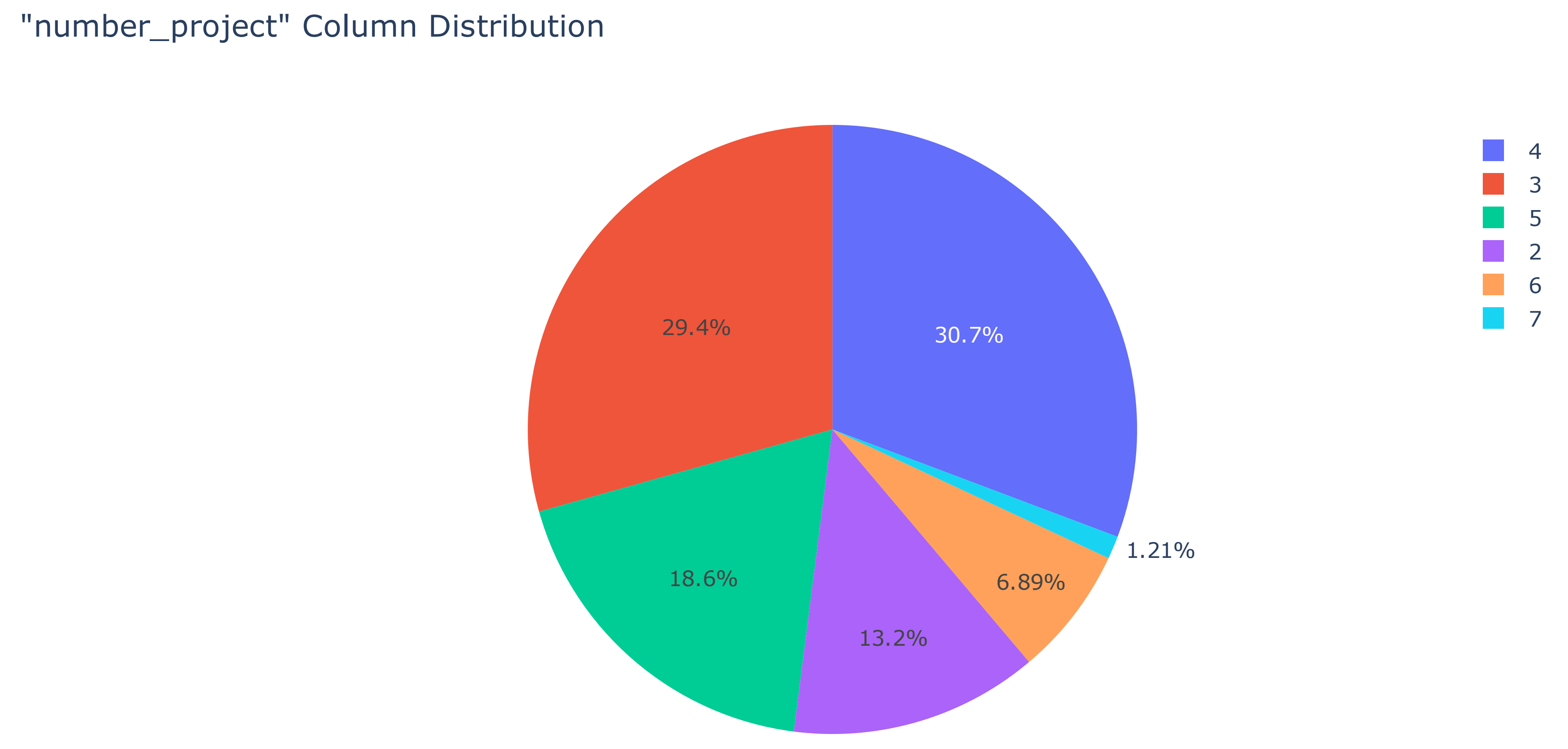


'number_project' Column

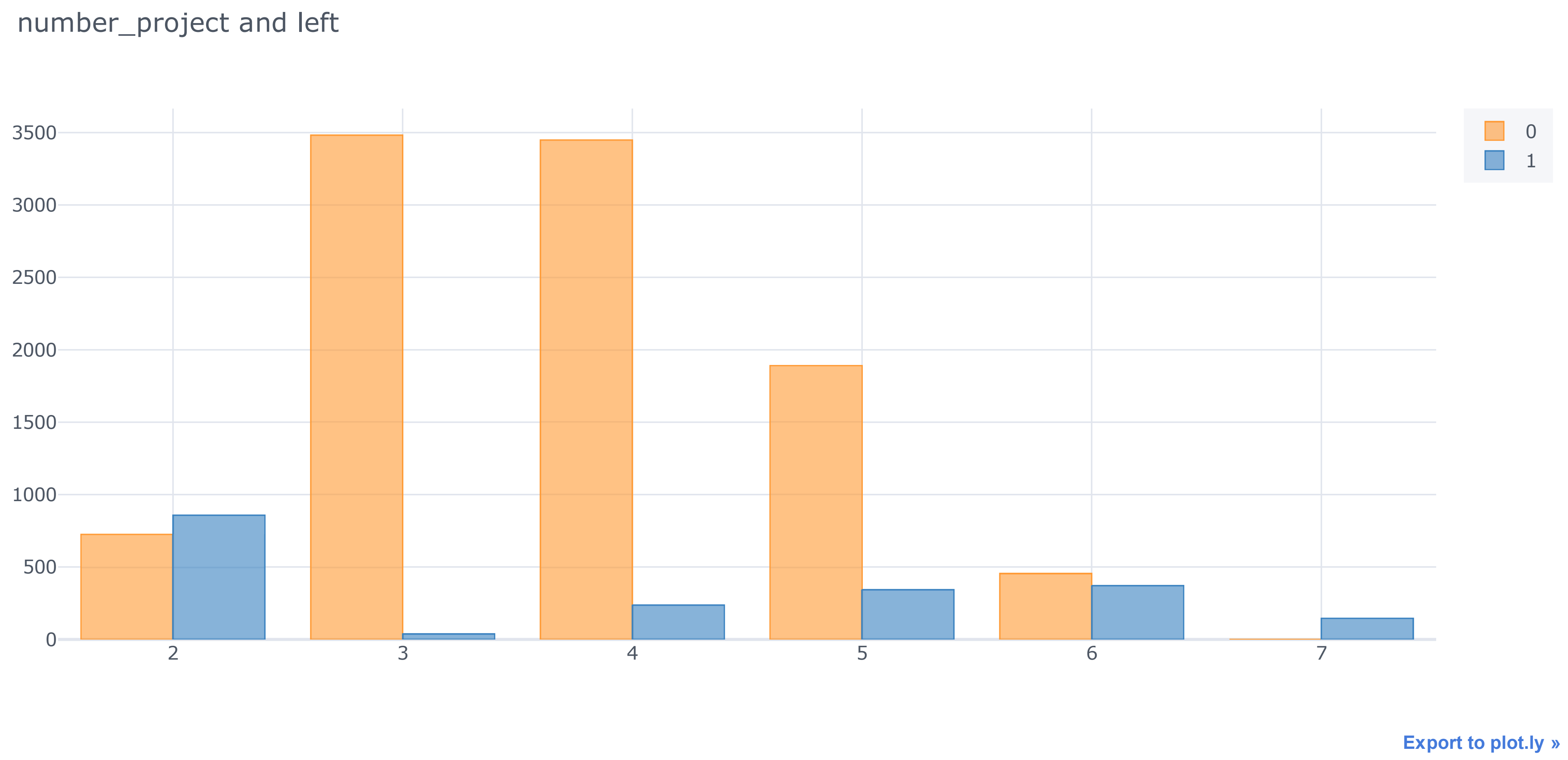
In [22]:
```
cprint("Have a First Look to 'number_project' Column",'green', 'on_black')
first_look('number_project')
```

```
Have a First Look to 'number_project' Column
column name    :  number_project
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  6
Duplicates     :  0
4    3685
3    3520
5    2233
2    1582
6     826
7     145
Name: number_project, dtype: int64
```

In [23]:
```
fig = px.pie(df, values = df['number_project'].value_counts(),
             names = (df['number_project'].value_counts()).index,
             title = '"number_project" Column Distribution')
fig.show()
```



In [24]:
```
pd.crosstab(df['number_project'], df['left']).iplot(kind='bar', title = 'number_project and left')
```



The number of leaving employees is higher among those who have only two projects during the period. This can be summed up as: "the employees with only two projects feel worthless or emptied". Because most of the employees work on three or four projects.

With the 6th project, the number of resignings is getting over the number of ongoings. There are no ongoing staff members who were assigned to 7 projects.

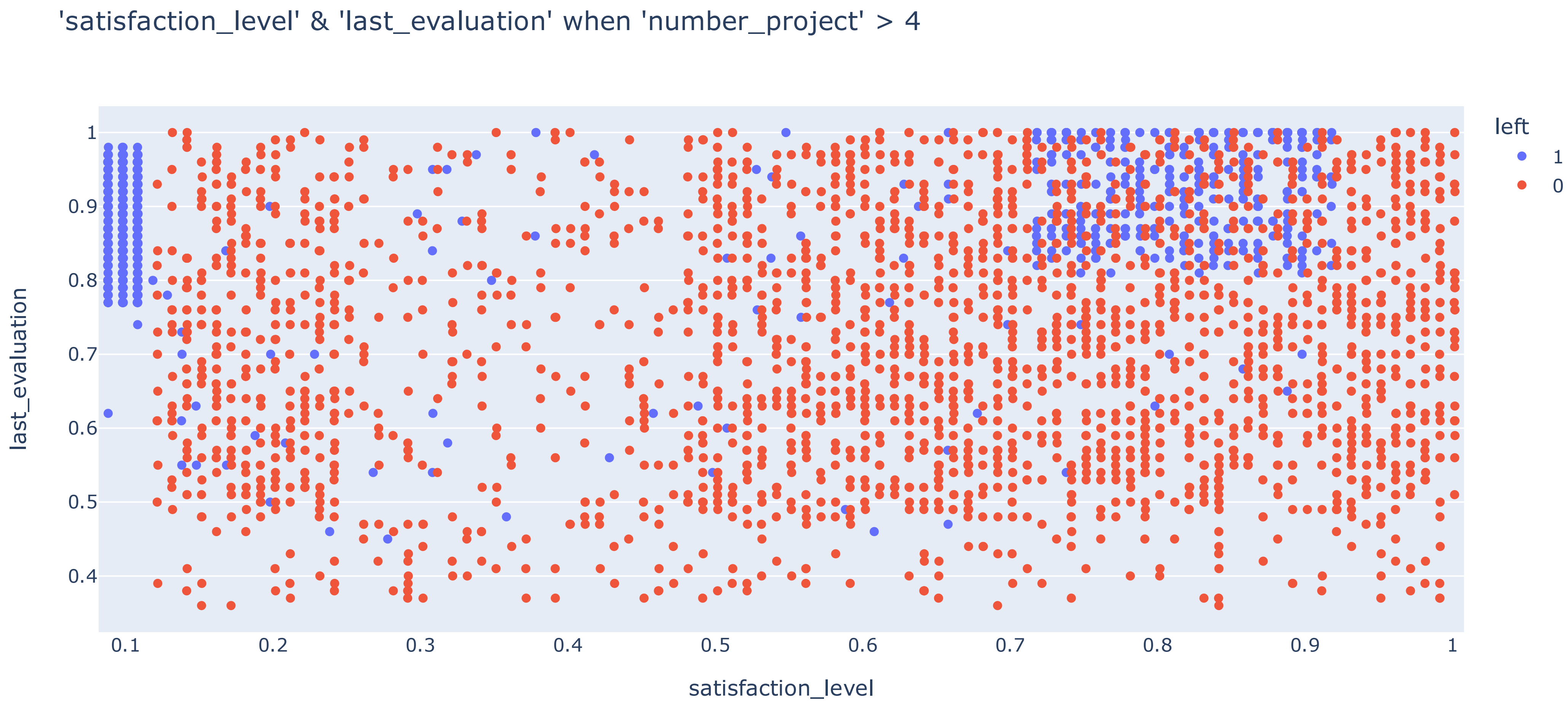
Working on more projects may cause intensive workload, regarding to this the satisfaction level may decrease with the insufficient motivators.

In [25]:
```
fig = px.strip(df[df['number_project'] == 2], x = 'satisfaction_level', y = 'last_evaluation', color = 'left',
               title = "'satisfaction_level' & 'last_evaluation' when 'number_project' == 2")
fig.show()
```

If we look at the satisfaction level, evaluation score and the number of projects together;

The group of undecideds who were evaluated as 0.5 are the group who worked on only two projects. As a result, our hypothesis about this group is becoming more clear. As the employer does not assign enough projects to this group, he/she cannot evaluate their performance and they feel worthless. Therefore, they are unsure about their future in the company. This may lead them to leave.

In [26]:
```
fig = px.strip(df[df['number_project'] > 4], x = 'satisfaction_level', y = 'last_evaluation', color = 'left',
               title = "'satisfaction_level' & 'last_evaluation' when 'number_project' > 4")
fig.show()
```



In [27]:
```
fig = px.strip(df[df['number_project'] == 7], x = 'satisfaction_level', y = 'last_evaluation', color = 'left',
               title = "'satisfaction_level' & 'last_evaluation' when 'number_project' == 7")
fig.show()
```

The leaving employees who worked on more than four projects are the group two and three of the last_evaluation section.

Especially most of the second group of last_evaluation section are worked on seven projects and left the company. So again our hypothesis about this group is now more definite.

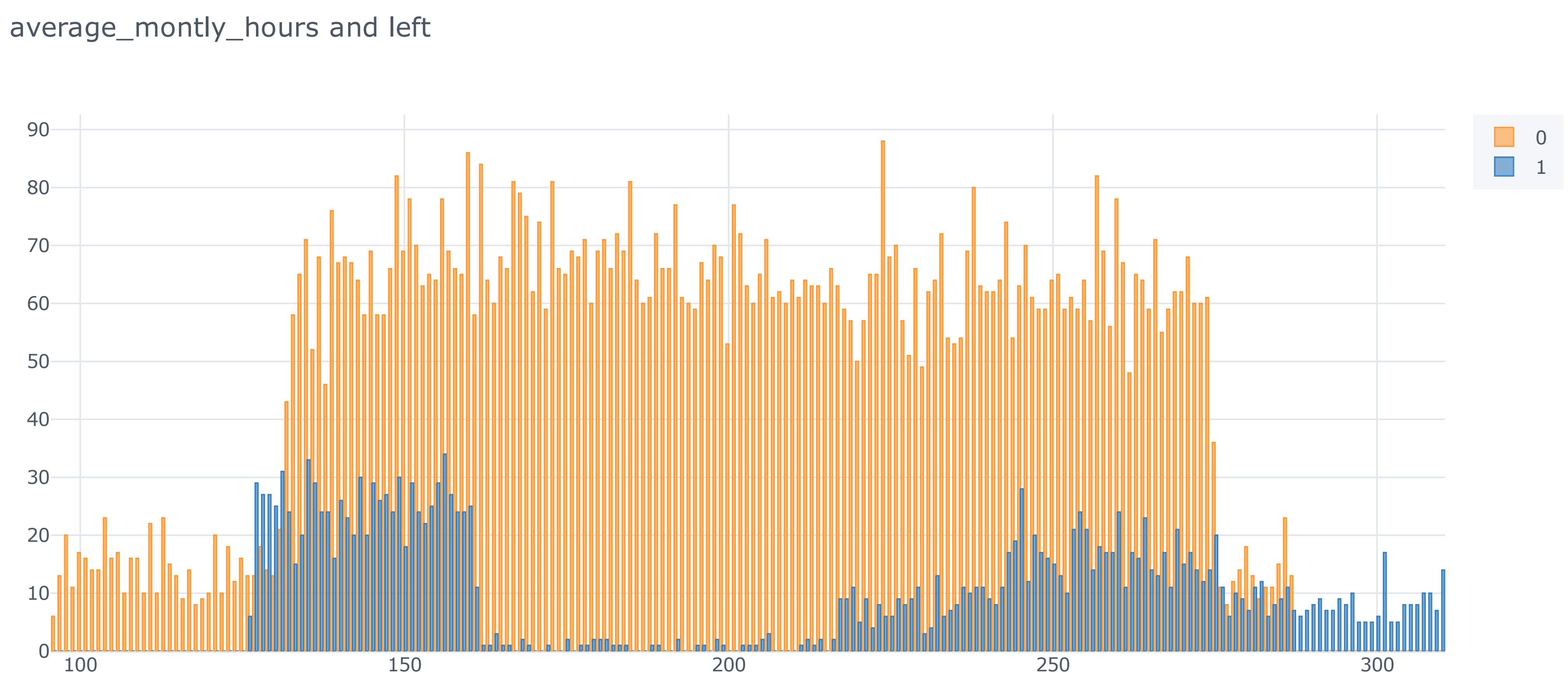
'average_montly_hours' Column

In [28]:
```
cprint("Have a First Look to 'average_montly_hours' Column",'green', 'on_black')
first_look('average_montly_hours')
```

```
Have a First Look to 'average_montly_hours' Column
column name    :  average_montly_hours
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  215
Duplicates     :  0
156    112
149    112
160    111
151    107
135    104
260    102
257    100
145     98
157     96
224     94
152     94
143     94
140     93
155     93
139     92
137     92
141     91
243     91
245     91
238     91
158     90
154     90
148     90
159     89
264     87
150     87
142     87
258     86
134     85
271     85
162     85
255     85
147     85
153     85
266     84
146     84
269     83
254     83
246     82
253     82
167     81
263     81
168     81
185     81
136     81
247     81
173     81
237     79
250     79
226     79
192     79
251     78
144     78
233     78
261     78
201     77
229     77
270     77
232     77
248     76
169     76
242     75
249     75
274     75
206     74
171     74
225     74
272     74
239     74
259     73
189     73
183     73
244     73
133     73
223     73
265     73
202     73
181     73
198     72
217     72
178     72
267     72
273     72
240     71
180     71
256     71
241     70
184     70
268     70
138     70
252     69
176     69
199     69
161     69
177     69
222     69
165     69
218     68
216     68
219     68
196     68
175     67
182     67
132     67
166     67
205     67
231     66
212     66
190     66
221     66
191     66
174     66
262     65
163     65
214     65
227     65
236     65
210     64
197     64
213     64
186     64
203     64
164     63
188     62
211     62
```

```
170    62
179    62
208    62
235    61
193    61
207    61
204    61
234    61
187    60
172    60
195    60
209    60
194    60
215    60
228    60
275    56
220    55
200    53
230    52
131    52
128    45
127    42
129    41
130    38
286    34
280    25
285    24
281    24
279    23
104    23
113    23
276    22
278    22
111    22
282    21
98     20
121    20
287    20
126    19
284    19
123    18
301    17
100    17
106    17
283    17
109    16
101    16
105    16
108    16
125    16
114    15
102    14
277    14
117    14
103    14
310    14
115    13
97     13
124    12
99     11
112    10
307    10
107    10
120    10
296    10
308    10
110    10
122    10
291     9
116     9
119     9
294     9
305     8
304     8
290     8
118     8
306     8
295     8
289     7
293     7
309     7
292     7
300     6
96      6
288     6
298     5
302     5
297     5
299     5
303     5
Name: average_montly_hours, dtype: int64
```

In [29]:
```
pd.crosstab(df['average_montly_hours'], df['left']).iplot(kind='bar', title = 'average_montly_hours and left')
```



Export to plot.ly »

- Looking at the 'average_montly_hours' values, there is a local increase in turnover in the 125-160 month working hours range and 210-290 monthly working hours.
- Those who work more than 290 hours per month are more likely to quit their jobs than those who do not.
- So the next question is "The average monthly working hours are related to projects number or not?"

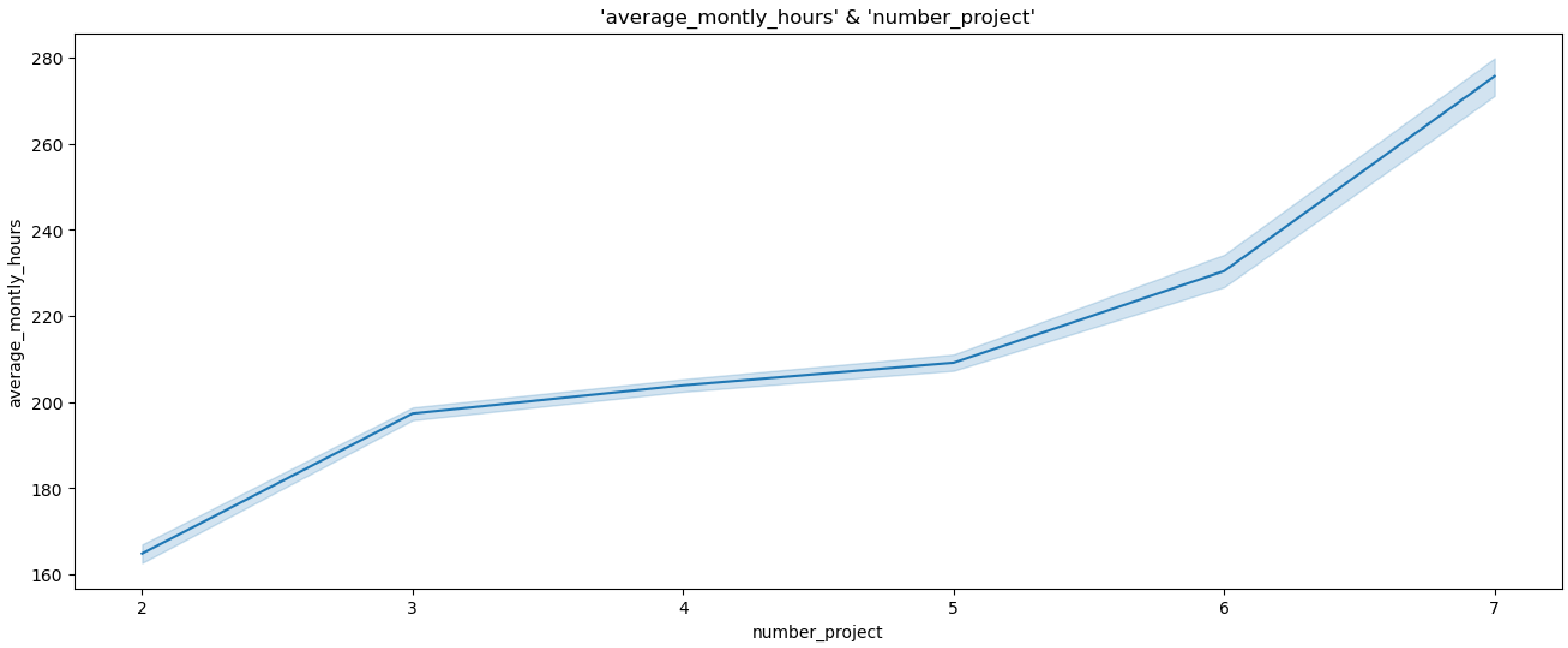
In [30]:
```
plt.figure(figsize = (16,6))
sns.lineplot(data = df, x = 'average_montly_hours', y = 'number_project', hue = 'left')
plt.title("'average_montly_hours' & 'number_project'");
```

At the graph above it is seen that the group working on two projects is working nearly 130-160 hours monthly. It can be assessed that they have only two simple projects that they don't need to work hard, so their loyalty is weak.

Most of the employees are working 135-275 hours monthly. In this group usually the employees who get two or more than five projects leaving the company.

When there is an increase on the number of projects and the average monthly working hours, there is also an increase on the number of resignings.

In [31]:
```
plt.figure(figsize = (16,6))
sns.lineplot(data = df, y = 'average_montly_hours', x = 'number_project')
plt.title("'average_montly_hours' & 'number_project'");
```



The increasing of the average monthly hours according to number of projects is seen on the graph.

The rate of increase is higher between two and three projects, and after five projects. So it is clearly define the number of resignings due to the working hours.

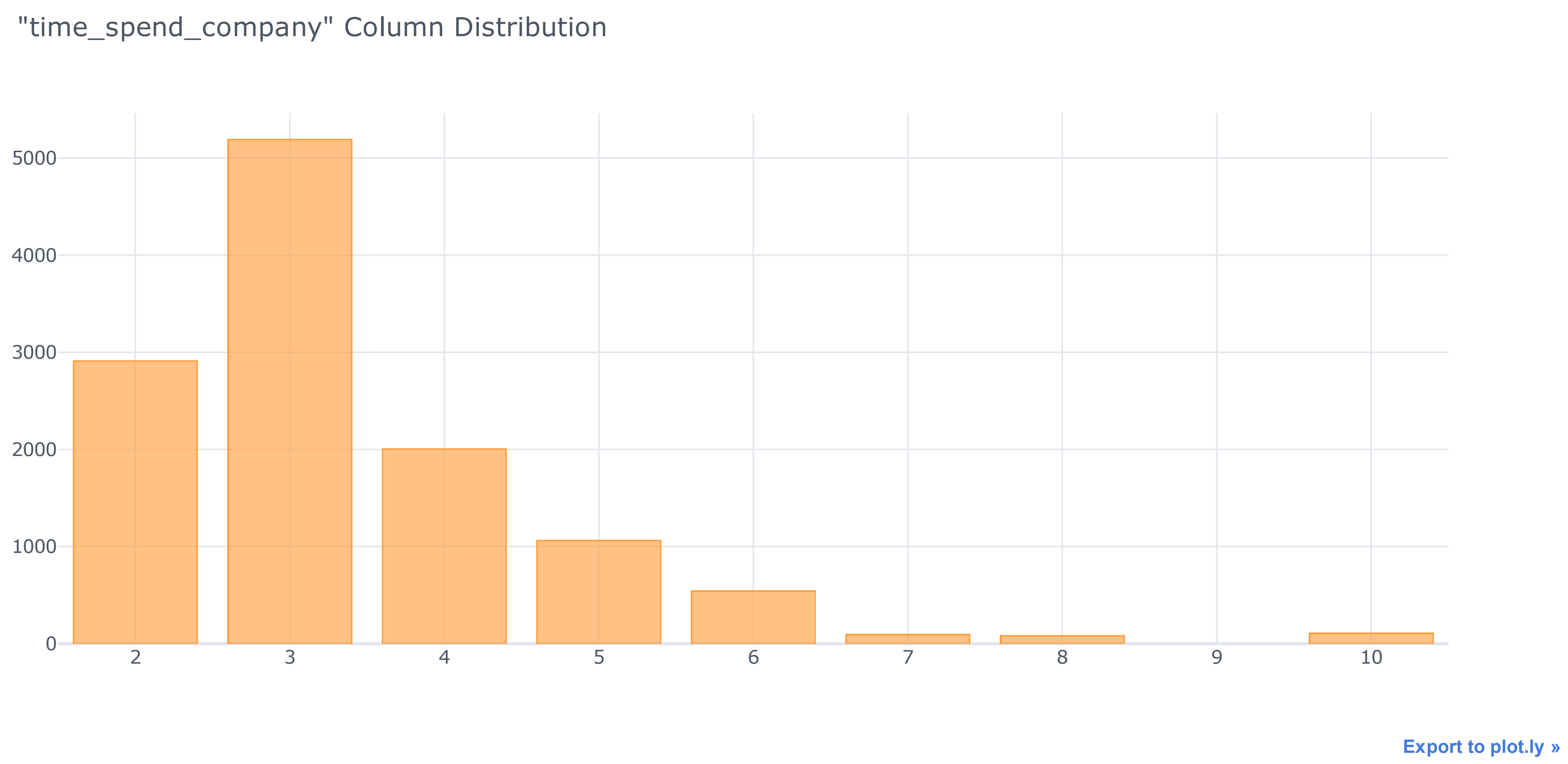
There need to be an adjustment about the project numbers, working hours and workload. The projects must be assigned to more employees. Also, better incentives must be offered to staff who are working hard.

'time_spend_company' Column

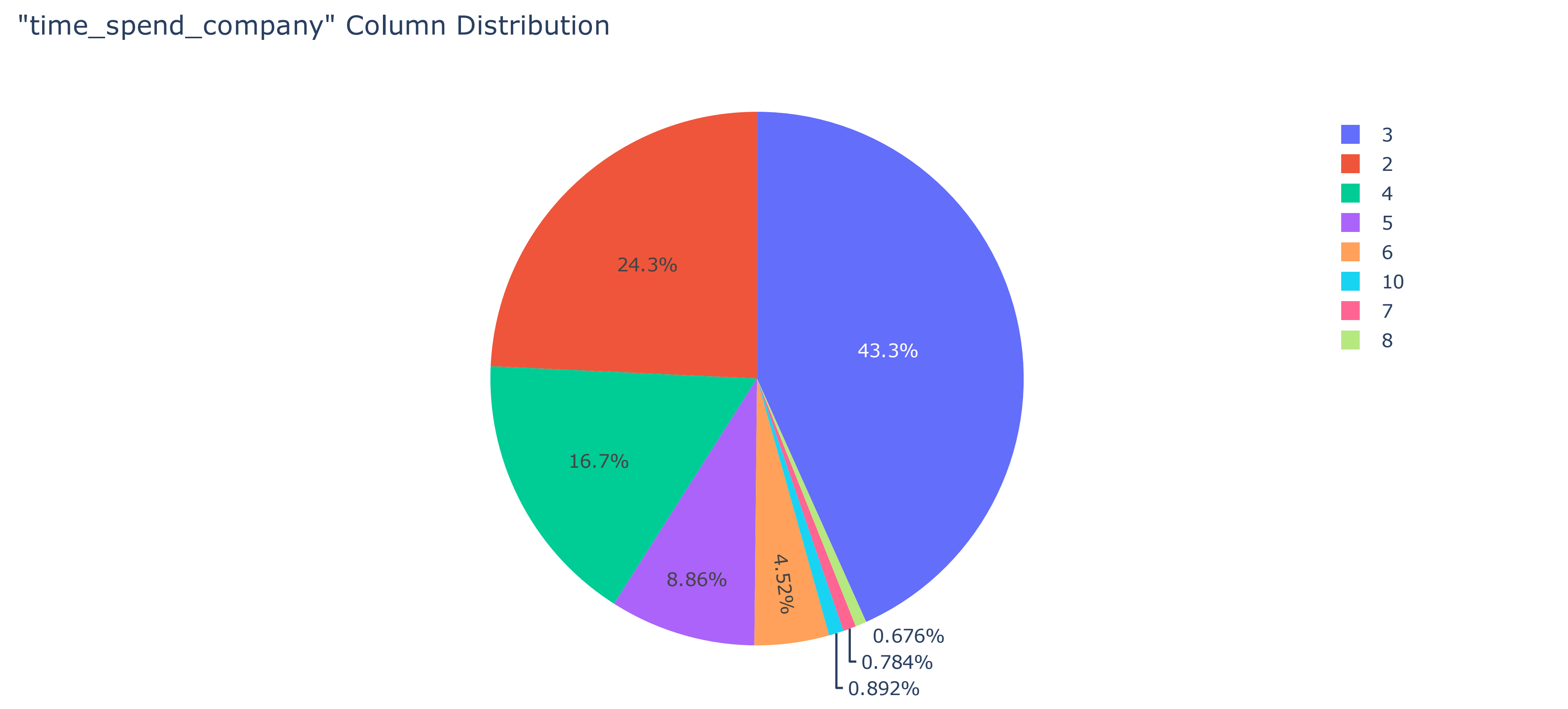
In [32]:
```
cprint("Have a First Look to 'time_spend_company' Column",'green', 'on_black')
first_look('number_project')
```

```
Have a First Look to 'time_spend_company' Column
column name    :  number_project
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  6
Duplicates     :  0
4    3685
3    3520
5    2233
2    1582
6     826
7     145
Name: number_project, dtype: int64
```

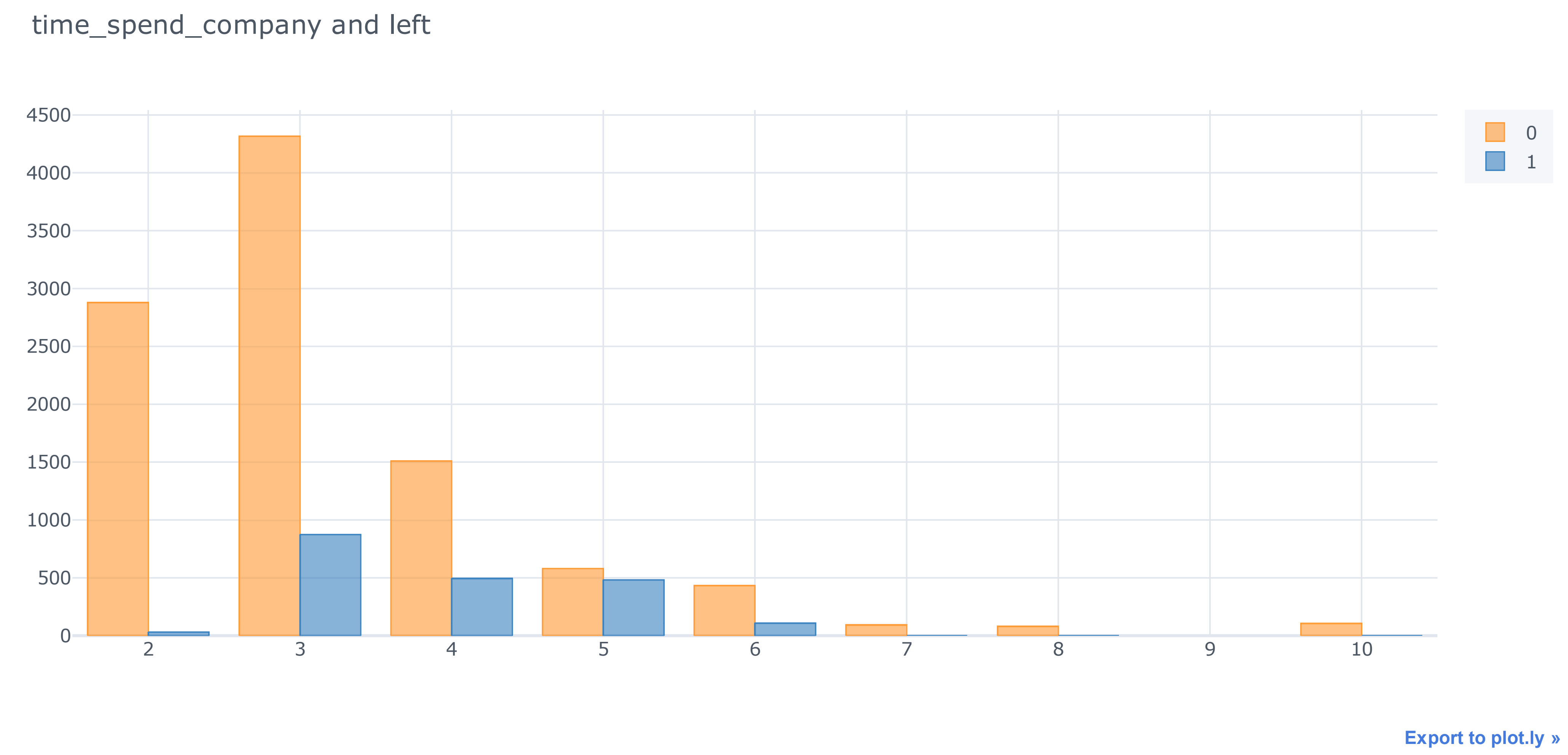
In [33]:
```
df['time_spend_company'].value_counts().iplot(kind="bar", title = '"time_spend_company" Column Distribution')
```



```
In [34]: fig = px.pie(df, values = df['time_spend_company'].value_counts(),
                      names = (df['time_spend_company'].value_counts()).index,
                      title = '"time_spend_company" Column Distribution')
         fig.show()
```

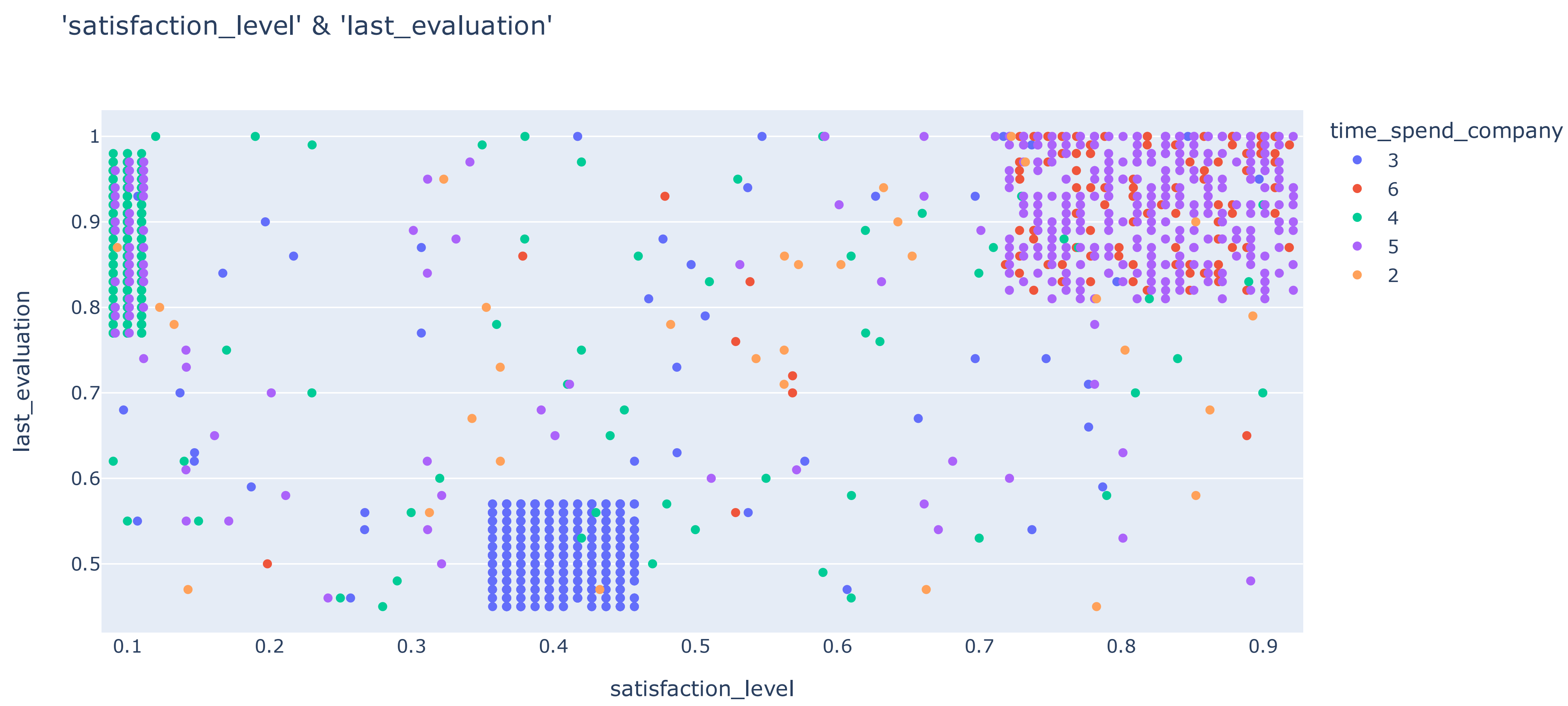



```
In [35]: pd.crosstab(df['time_spend_company'], df['left']).iplot(kind='bar', title = 'time_spend_company and left')
```



- Looking at the 'time_spent_company' values, there is an increase in turnover in the 3rd working year, but this increase gradually decreases until the 6th working year.

```
In [36]: fig = px.strip(df[df['left'] == 1], x = 'satisfaction_level', y = 'last_evaluation', color = 'time_spend_company',
                        title ="'satisfaction_level' & 'last_evaluation'")
         fig.show()
```



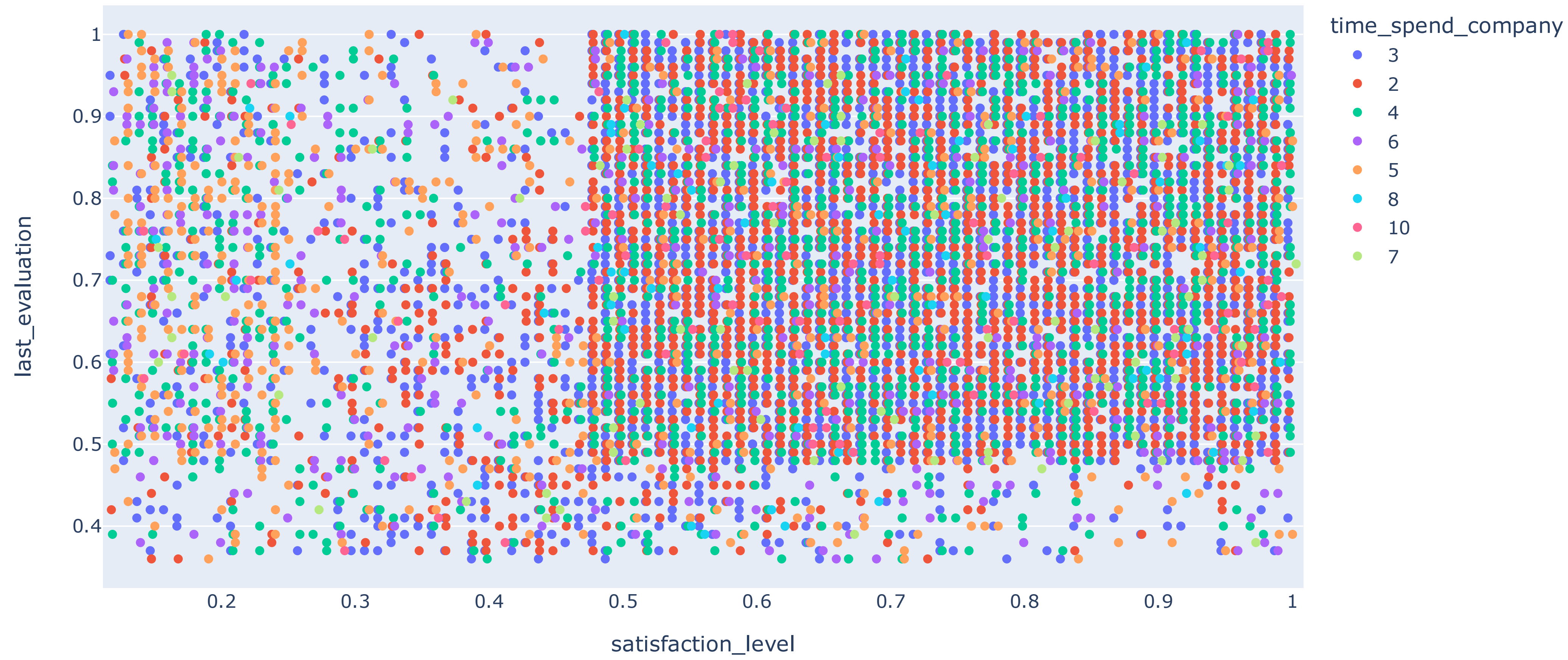
As can be seen on the graph, the employees are not able to make a clear assessment of the company during the first three years of their employment. This, coupled with the other factors, tends to lead to leaving the company after three years.

By the fourth year, their workload increases and their satisfaction decreases.

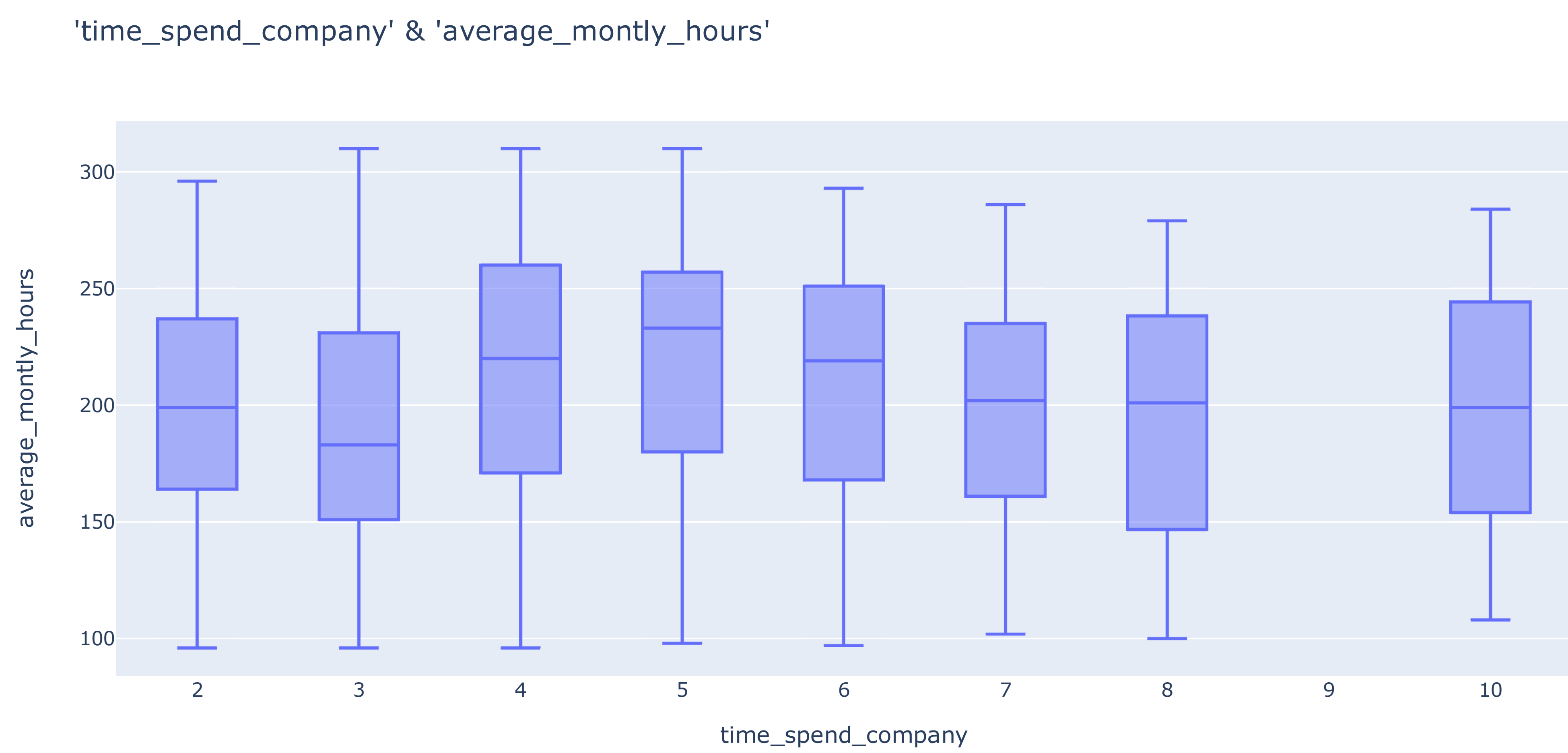
After the fifth year, they make an assessment, "they will leave or not".

If they decide to continue in the company, they never consider leaving after the sixth year.

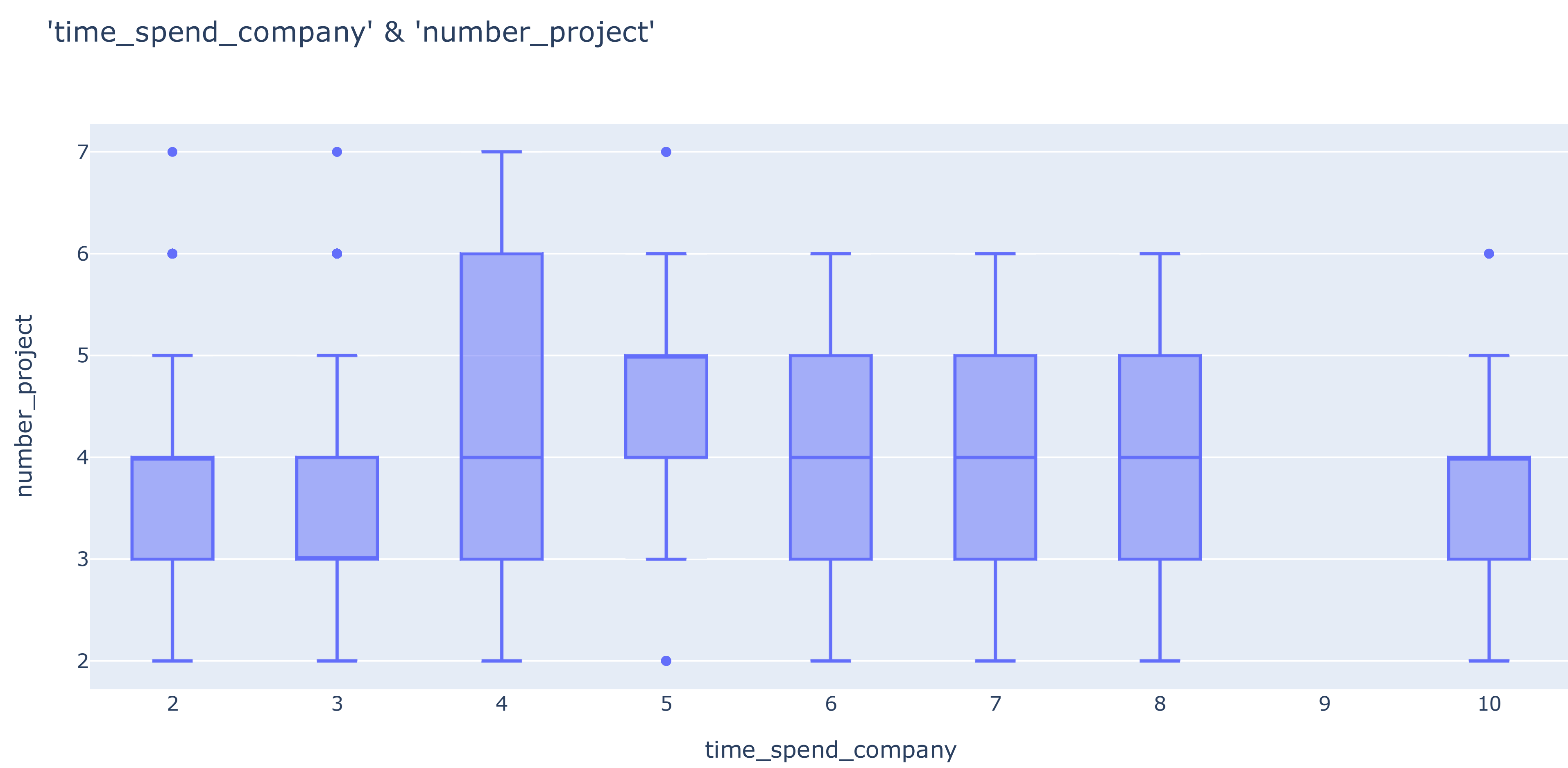
```
In [37]: fig = px.strip(df[df['left'] == 0], x = 'satisfaction_level', y = 'last_evaluation', color = 'time_spend_company')
         fig.show()
```



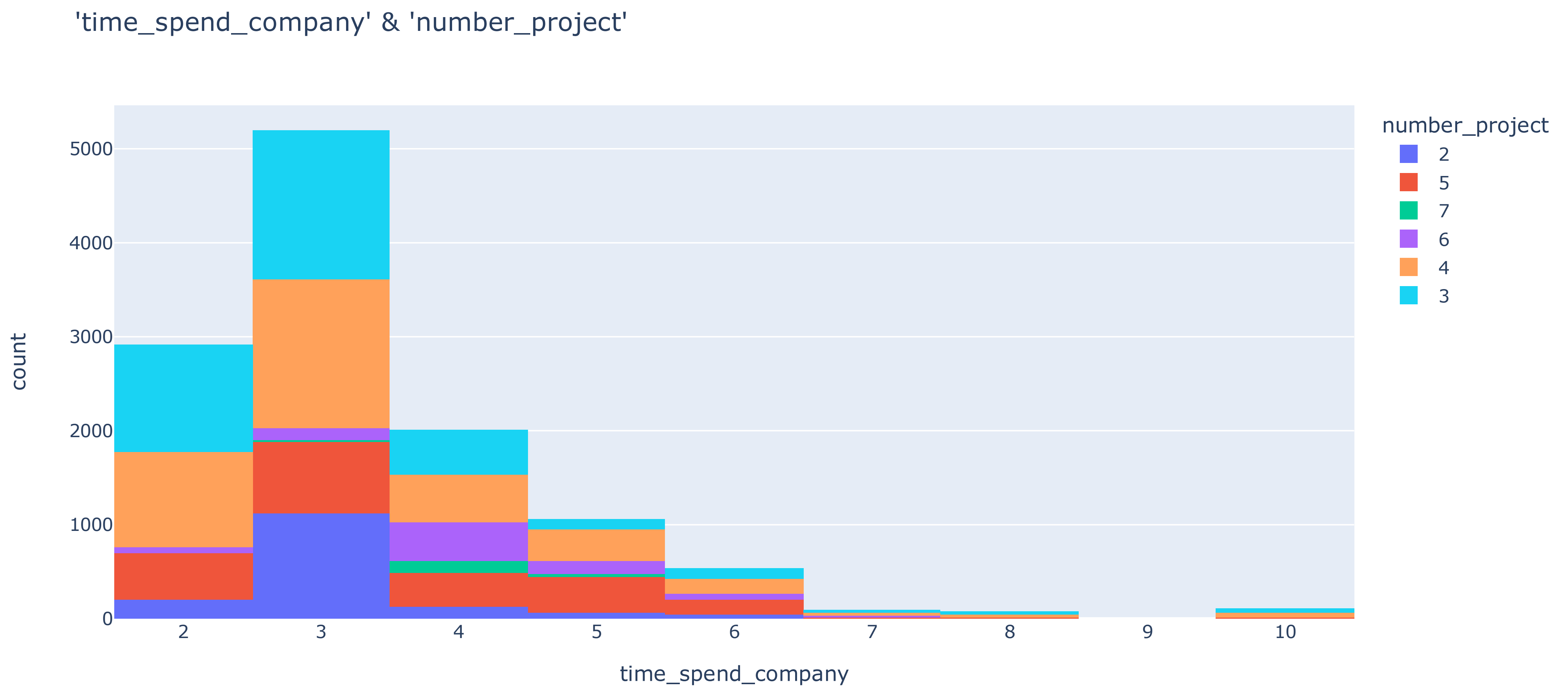
In [38]:
```
fig = px.box(df, x = 'time_spend_company', y = 'average_montly_hours', title = "'time_spend_company' & 'average_montly_hours'")
fig.show()
```



In [39]:
```
fig = px.box(df, x = 'time_spend_company', y = 'number_project', title = "'time_spend_company' & 'number_project'")
fig.show()
```



In [40]:
```
px.histogram(df, x = 'time_spend_company', color = 'number_project', title = "'time_spend_company' & 'number_project'")
```



Then how is the relation between workload and time spend in the company?

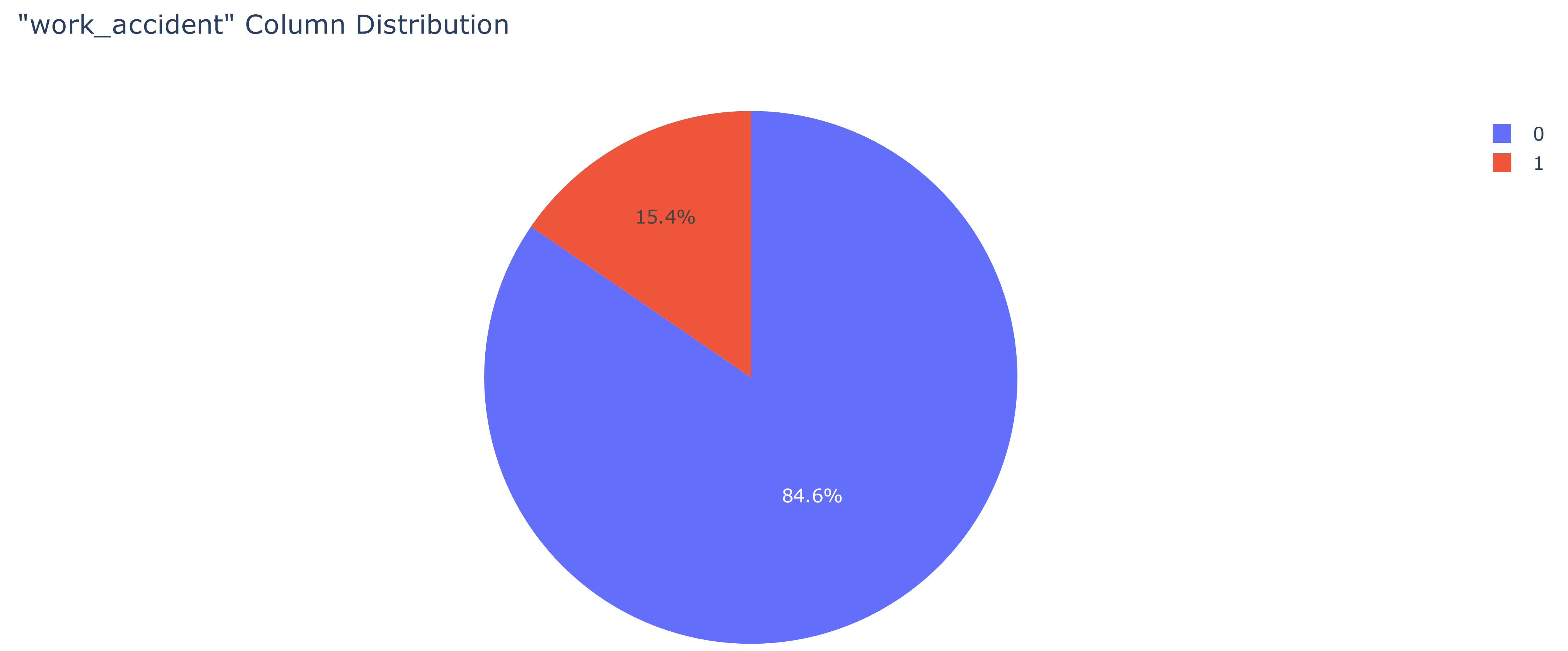
Third year staff has the most workload. After that year number of participated project is decreasing stepped. It makes sense. The experienced staff becoming team leader or manager position. That's why less of them can be assigned to projects.

'work_accident' Column

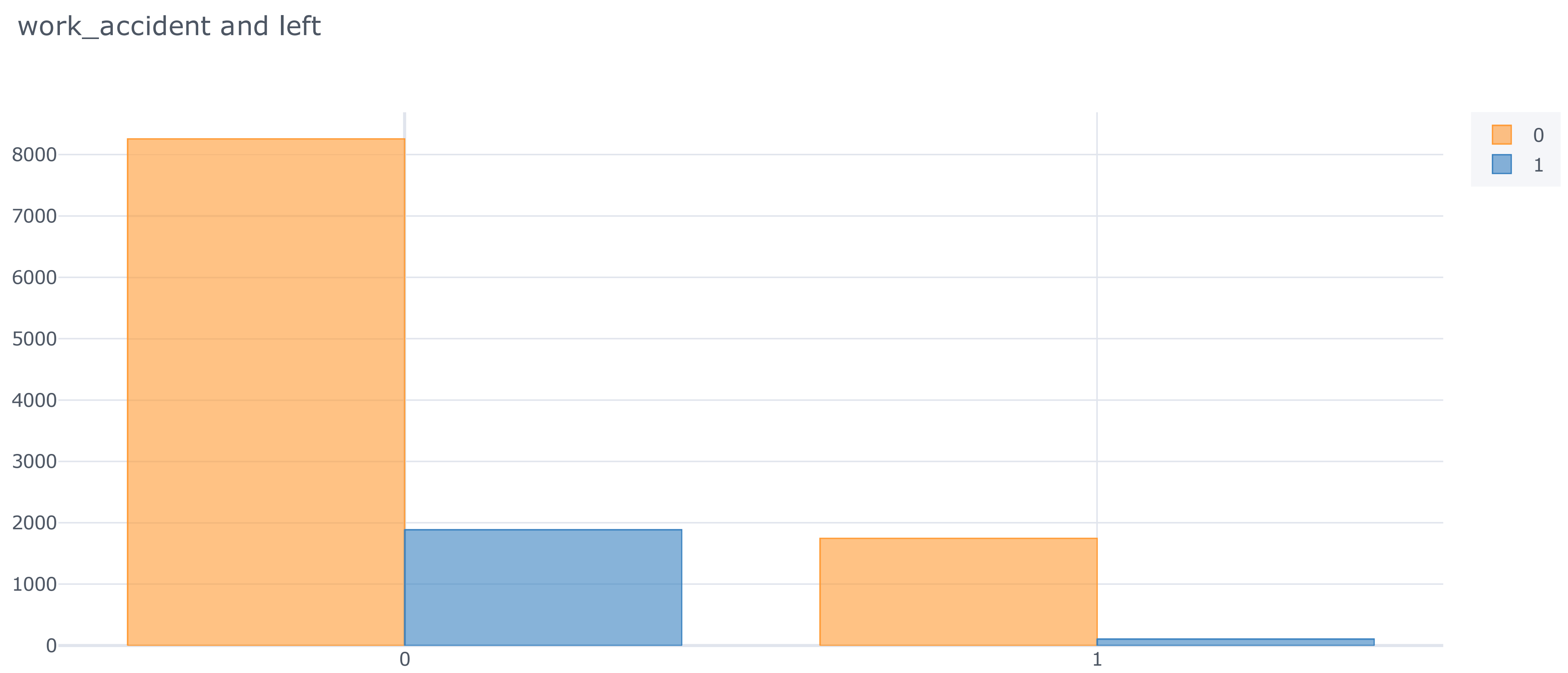
In [41]:
```
cprint("Have a First Look to 'work_accident' Column",'green', 'on_black')
first_look('work_accident')
```

```
Have a First Look to 'work_accident' Column
column name    :  work_accident
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  2
Duplicates     :  0
0    10141
1     1850
Name: work_accident, dtype: int64
```

In [42]:
```
fig = px.pie(df, values = df['work_accident'].value_counts(),
             names = (df['work_accident'].value_counts()).index,
             title = '"work_accident" Column Distribution')
fig.show()
```

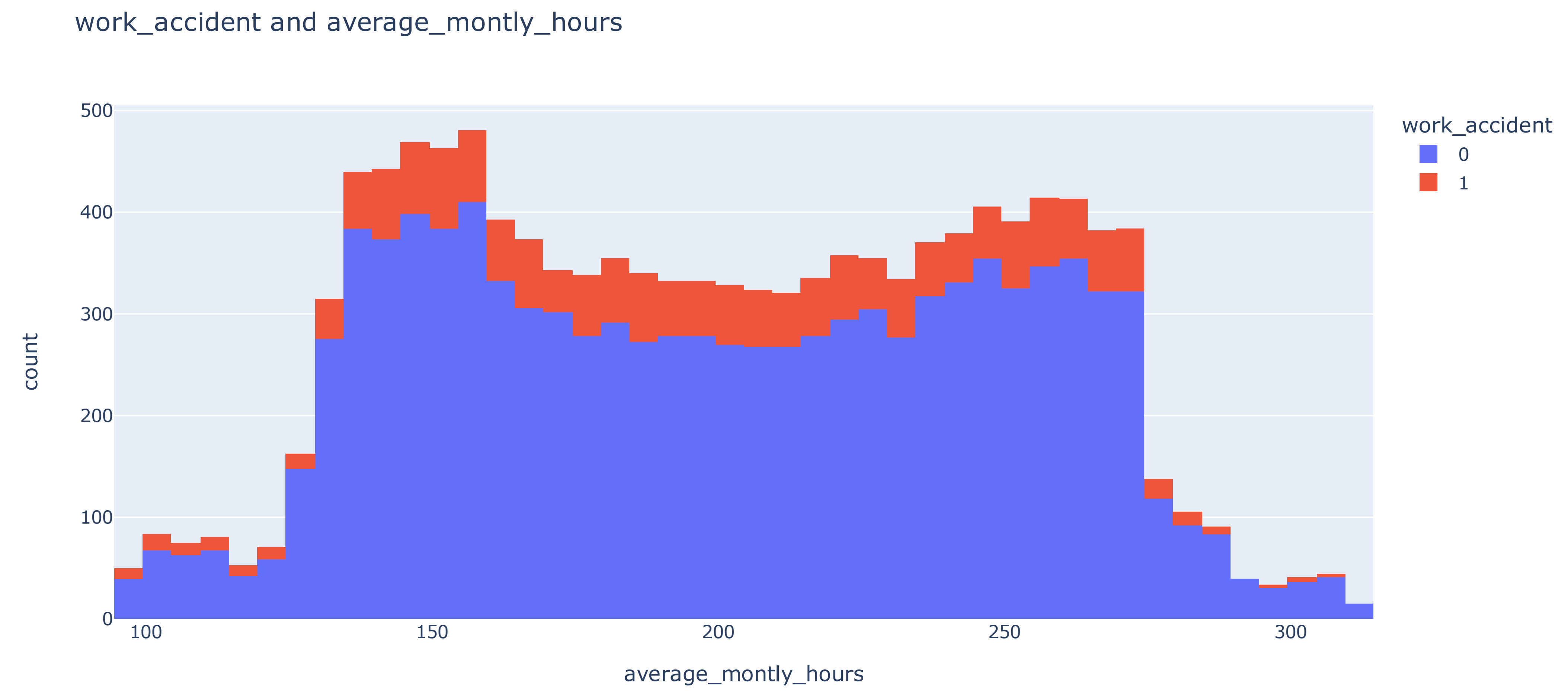


In [43]:
```
pd.crosstab(df['work_accident'], df['left']).iplot(kind='bar', title = 'work_accident and left')
```

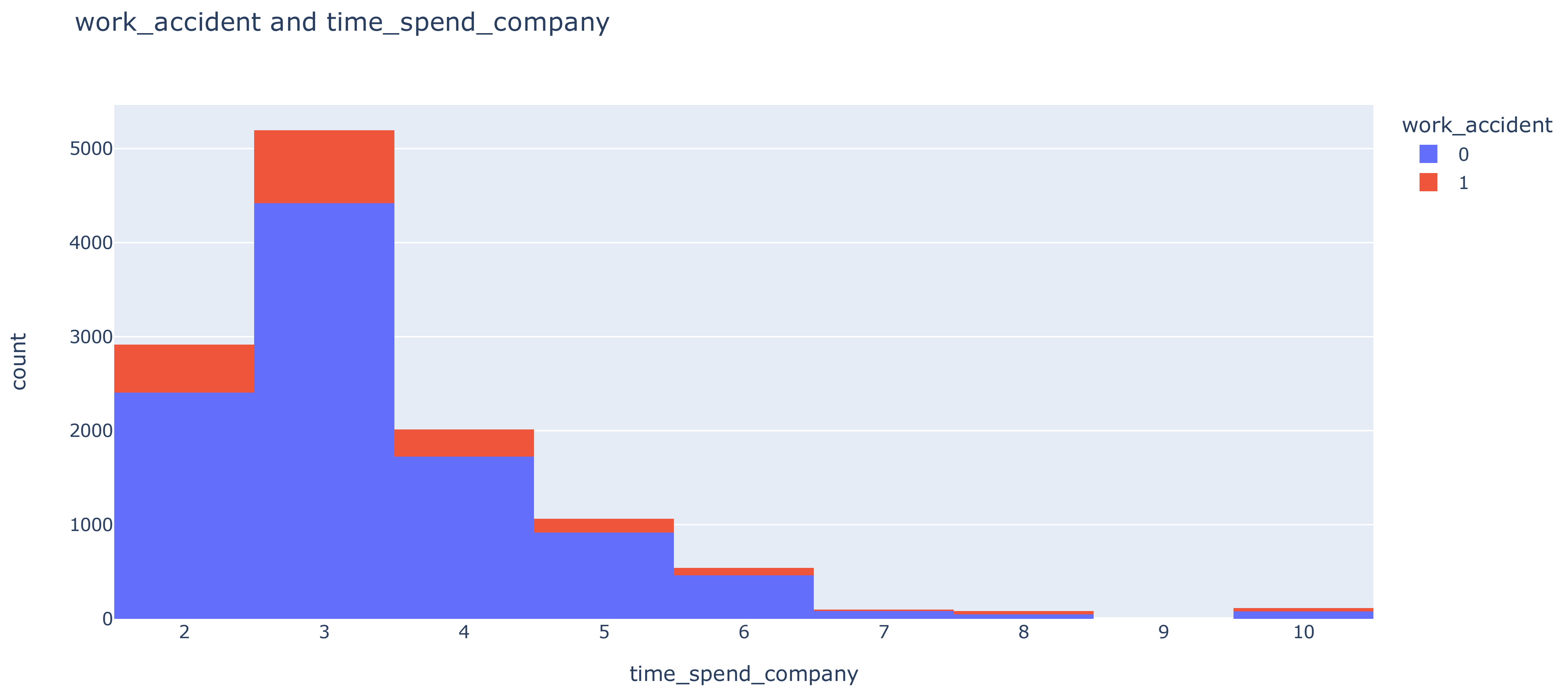


Export to plot.ly »

In [44]:
```
px.histogram(df, x = df['average_montly_hours'], color='work_accident', title = 'work_accident and average_montly_hours')
```



In [45]:
```
px.histogram(df, x = df['time_spend_company'], color='work_accident', title = 'work_accident and time_spend_company')
```



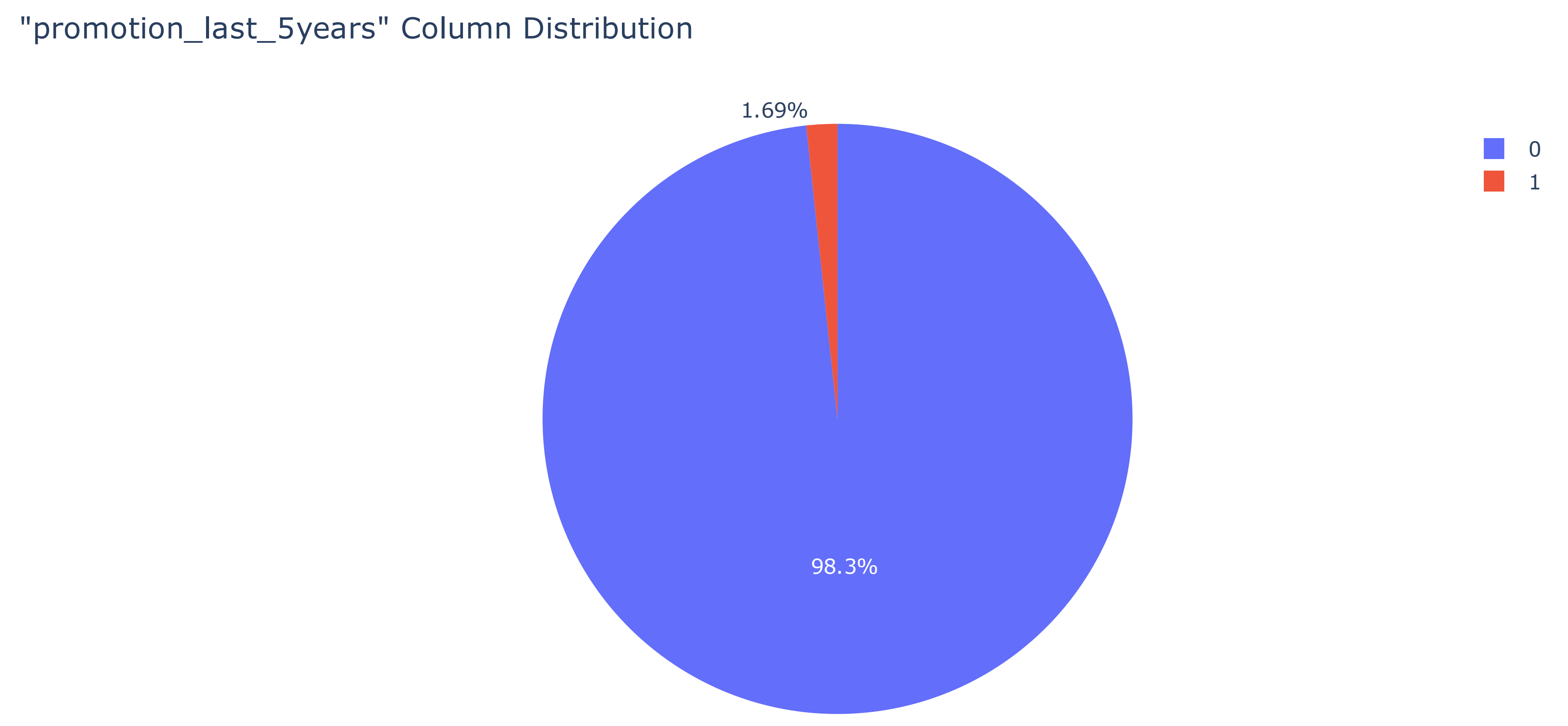
- 'work_accident' column has binary type values.
- Left ratios are similar between those who have had a work accident and those who have not.
- It does not appear to be a determining factor. In fact, it can be said that the left rate of those who have had a work accident is proportionally lower.

'promotion_last_5years' Column

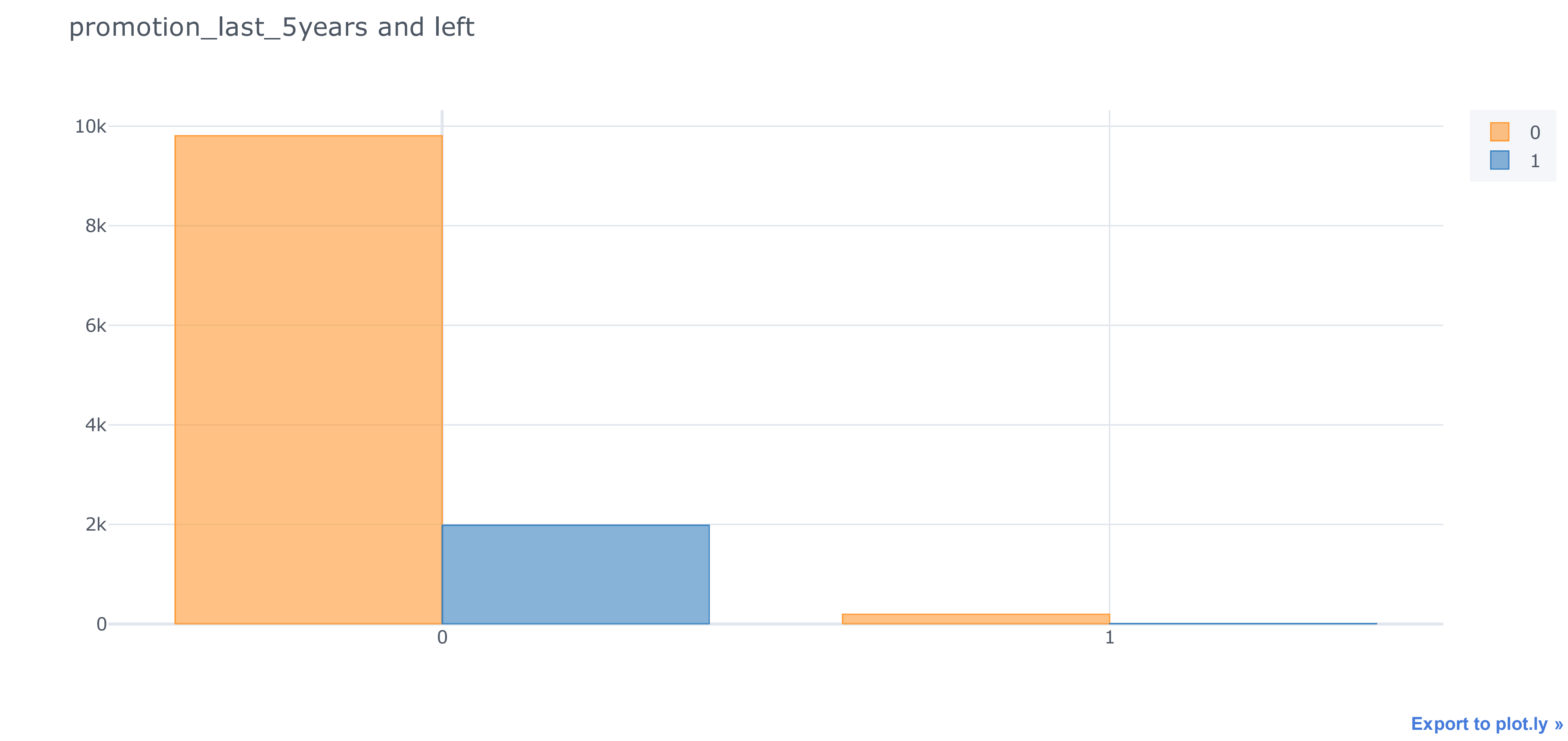
In [46]:
```
cprint("Have a First Look to 'promotion_last_5years' Column",'green', 'on_black')
first_look('promotion_last_5years')
```

```
Have a First Look to 'promotion_last_5years' Column
column name    :  promotion_last_5years
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  2
Duplicates     :  0
0    11788
1      203
Name: promotion_last_5years, dtype: int64
```

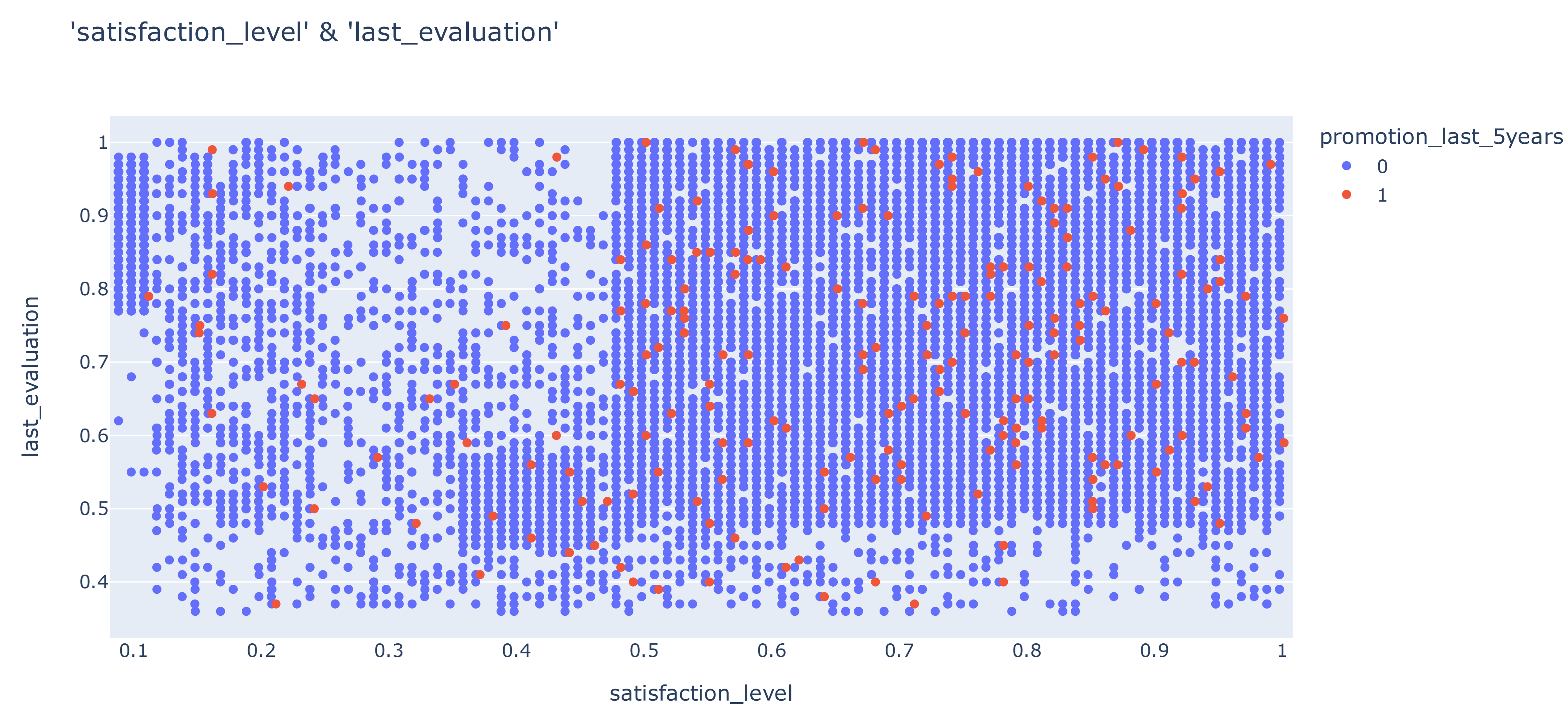
In [47]:
```
fig = px.pie(df, values = df['promotion_last_5years'].value_counts(),
             names = (df['promotion_last_5years'].value_counts()).index,
             title = '"promotion_last_5years" Column Distribution')
fig.show()
```



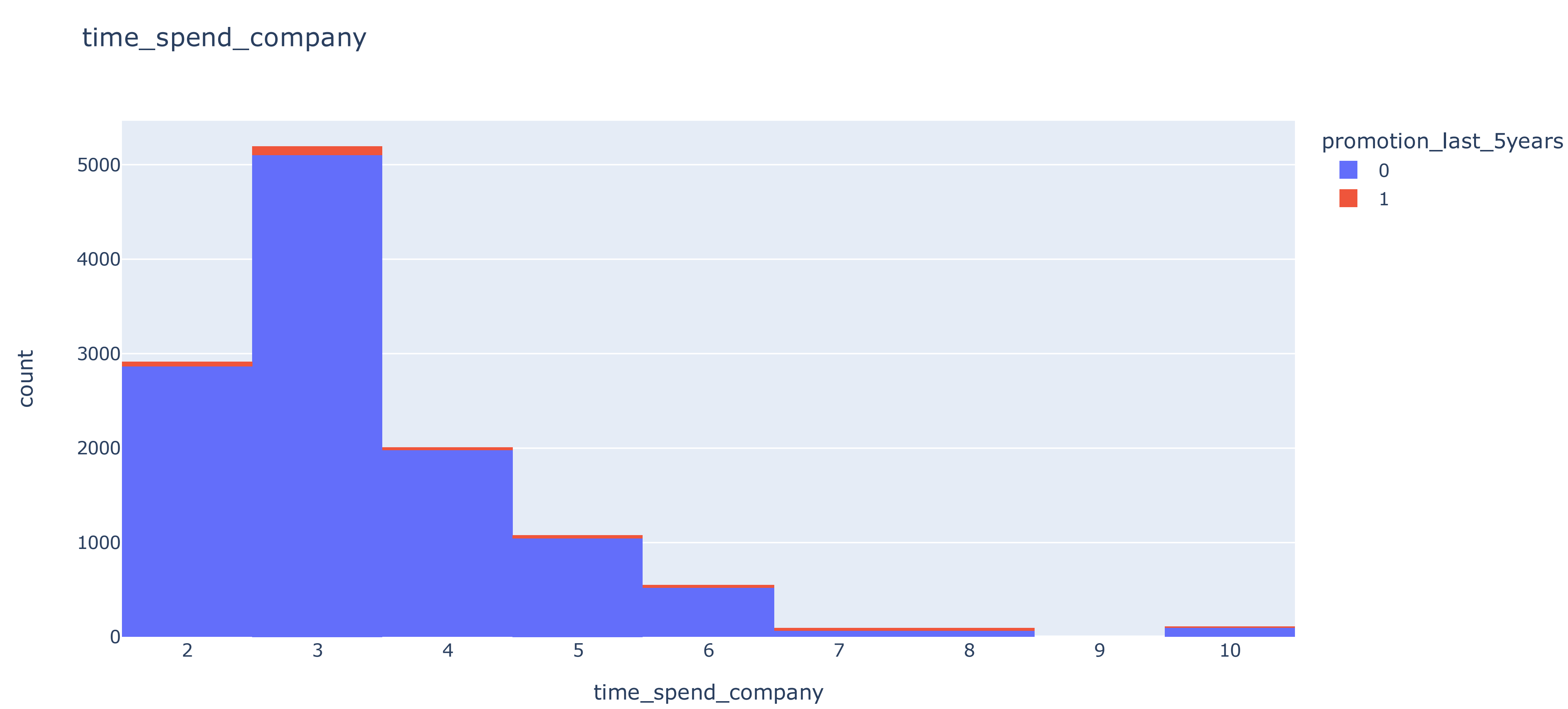
In [48]:
```
pd.crosstab(df['promotion_last_5years'], df['left']).iplot(kind='bar', title = 'promotion_last_5years and left')
```



In [49]:
```
fig = px.strip(df, x = 'satisfaction_level', y = 'last_evaluation', color = 'promotion_last_5years',
               title = "'satisfaction_level' & 'last_evaluation'")
fig.show()
```

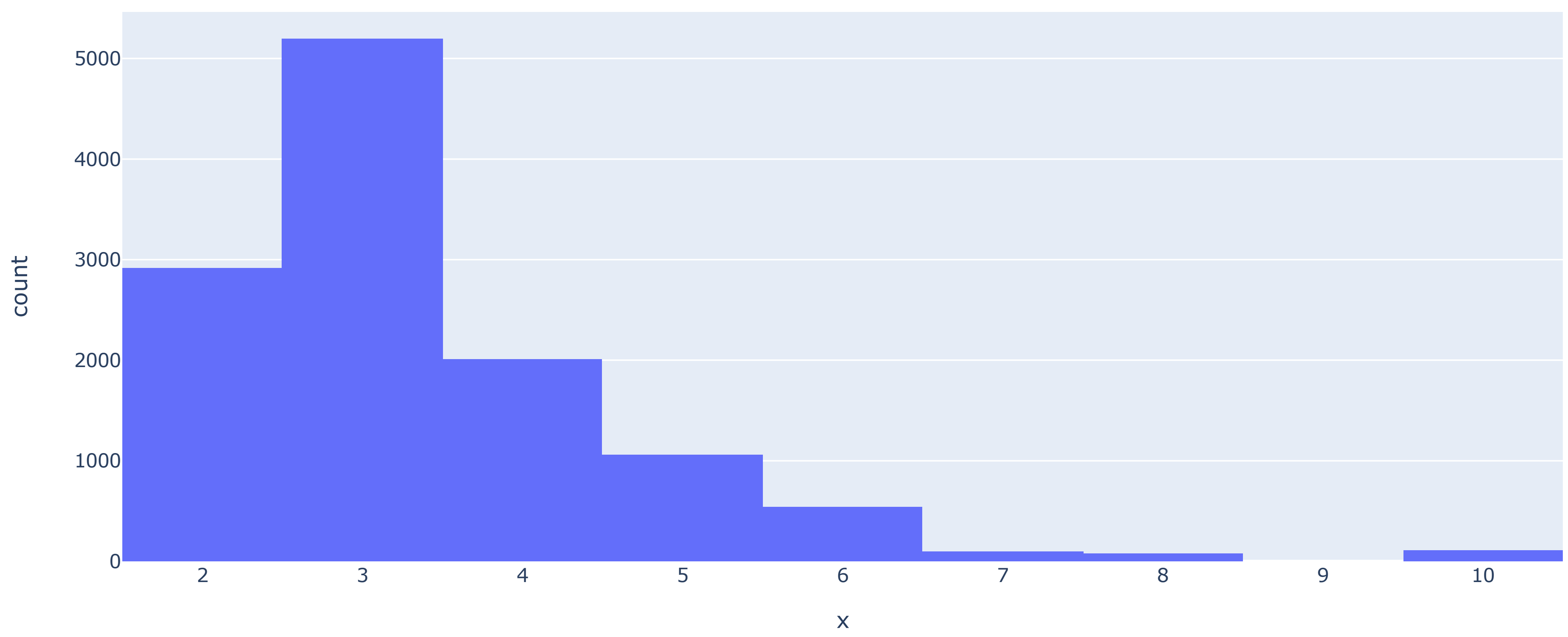


In [50]:
```
px.histogram(df, x = df['time_spend_company'], color='promotion_last_5years', title = 'time_spend_company')
```



In [51]:
```
px.histogram(df[df['promotion_last_5years'] == 1], x = df['time_spend_company'], title = 'promotion_last_5years')
```

promotion_last_5years



- 'promotion_last_5years' column has binary type values.
- Receiving a promotion in the last 5 working years is not determinative in terms of leaving or continuing to work.

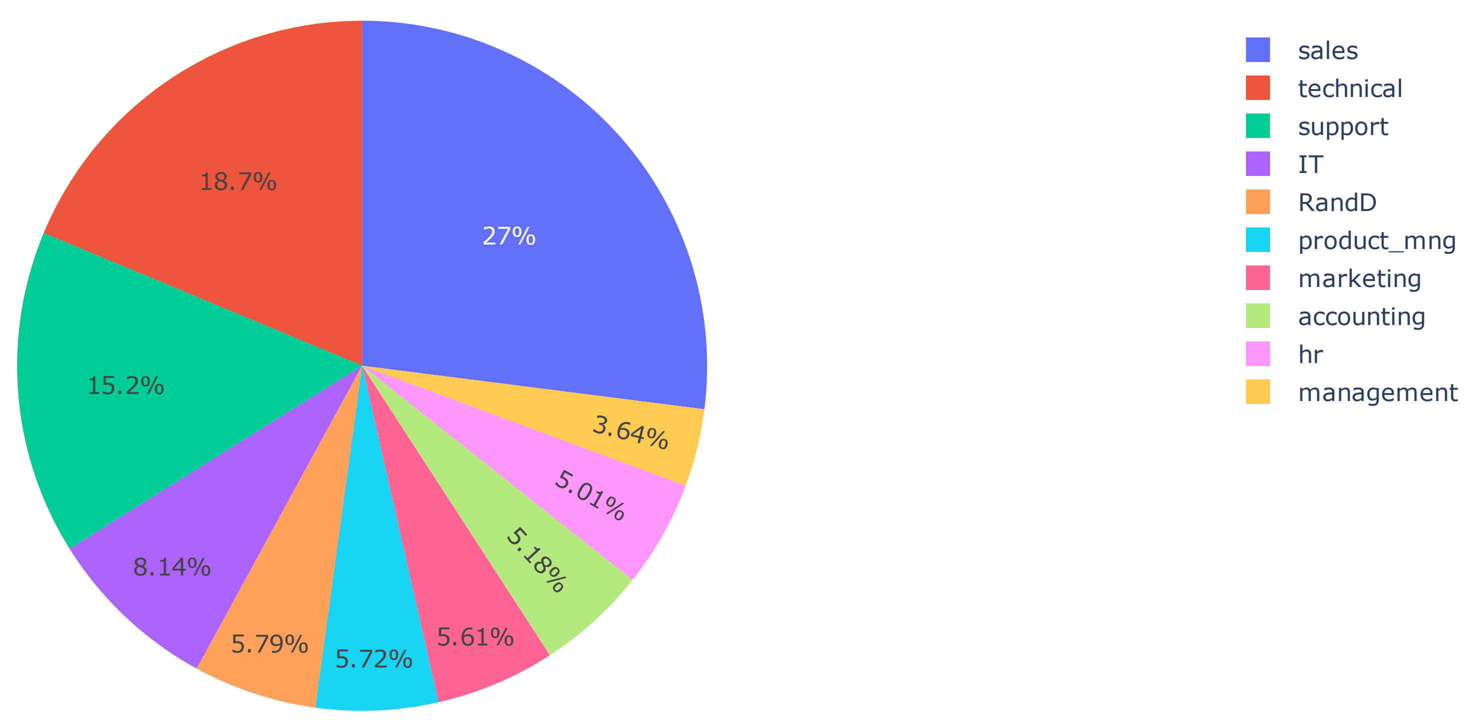
'department' Column

In [52]:
```
cprint("Have a First Look to 'department' Column",'green', 'on_black')
first_look('department')
```

```
Have a First Look to 'department' Column
column name    :  department
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  10
Duplicates     :  0
sales          3239
technical      2244
support        1821
IT              976
RandD           694
product_mng     686
marketing       673
accounting      621
hr              601
management      436
Name: department, dtype: int64
```

In [53]:
```
fig = px.pie(df, values = df['department'].value_counts(),
             names = (df['department'].value_counts()).index,
             title = '"department" Column Distribution')
fig.show()
```

"department" Column Distribution



In [54]:
```
cprint('left, not_left values and left percentage','green', 'on_red')
df_dep = pd.DataFrame(pd.crosstab(df['department'], df['left']))
df_dep.rename(columns = {0 : 'not_left', 1 : 'left'}, inplace = True)
df_dep = df_dep.assign(total = lambda x: (x['not_left'] + x['left']))
df_dep = df_dep.assign(left_percentage = lambda x: (x['left'] / x['total'] * 100))
df_dep
```

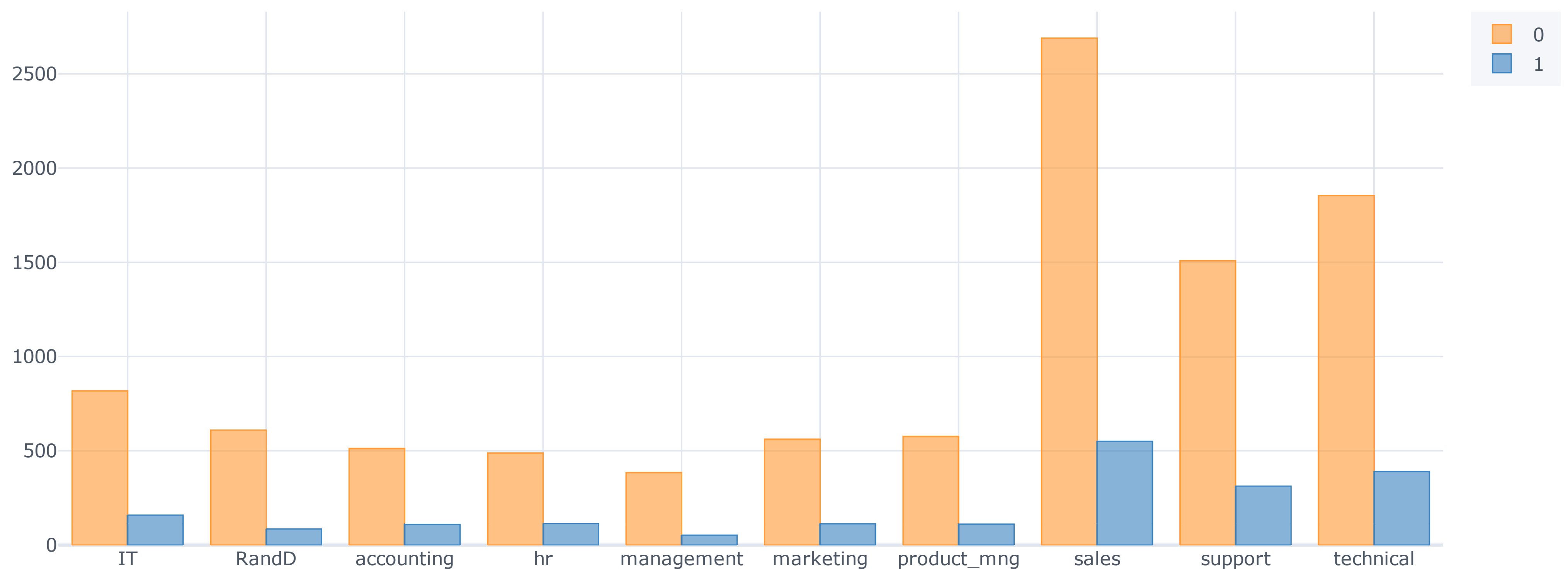
left, not_left values and left percentage

Out[54]:

left	not_left	left	total	left_percentage
department				
IT	818	158	976	16.189
RandD	609	85	694	12.248
accounting	512	109	621	17.552
hr	488	113	601	18.802
management	384	52	436	11.927
marketing	561	112	673	16.642
product_mng	576	110	686	16.035
sales	2689	550	3239	16.981
support	1509	312	1821	17.133
technical	1854	390	2244	17.380

In [55]:
```
pd.crosstab(df['department'], df['left']).iplot(kind='bar', title = 'department and left')
```

department and left



Export to plot.ly »

- It is not observed that the departments worked alone have an effect on the left decision.
- It is seen that the left percentages of the departments are similar.

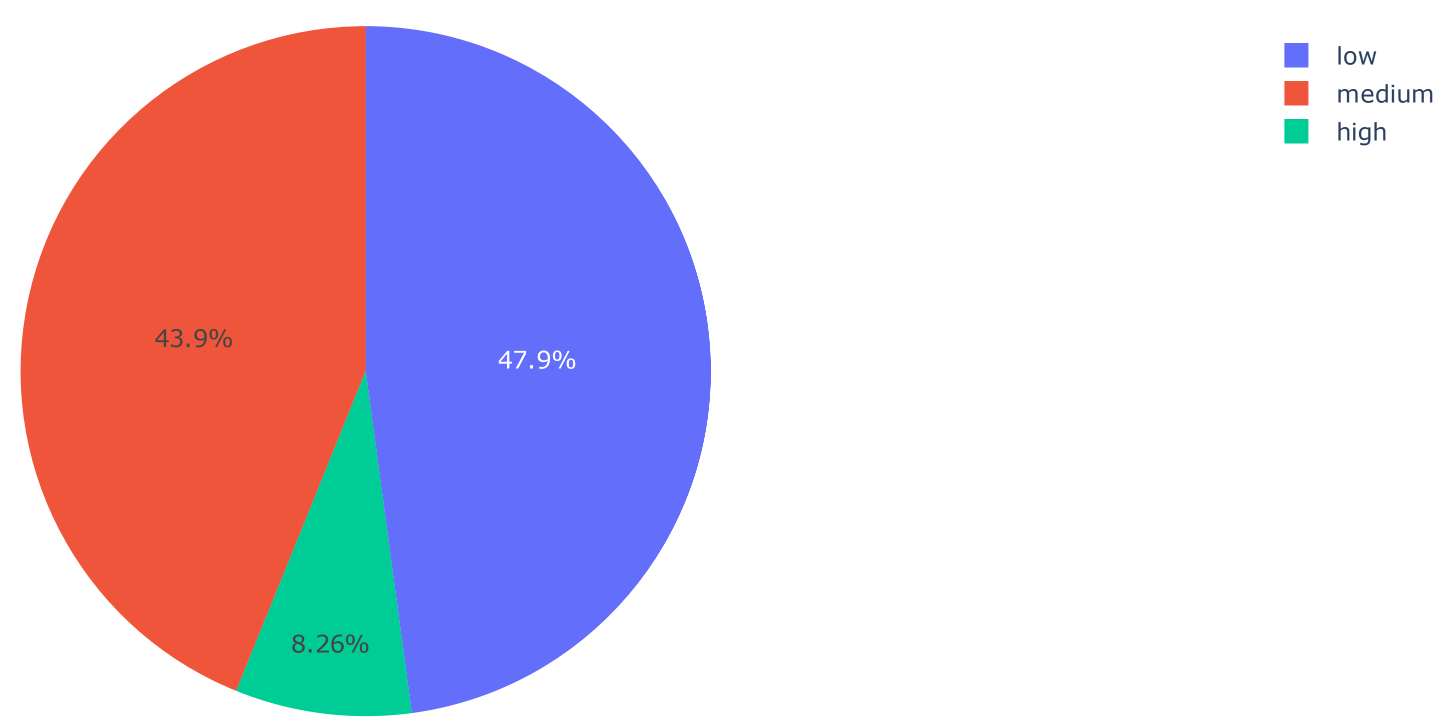
'salary' Column

In [56]:
```
cprint("Have a First Look to 'salary' Column",'green', 'on_black')
first_look('salary')
```

```
Have a First Look to 'salary' Column
column name    :  salary
--------------------------------
Per_of_Nulls   :  % 0.0
Num_of_Nulls   :  0
Num_of_Uniques :  3
Duplicates     :  0
low       5740
medium    5261
high       990
Name: salary, dtype: int64
```

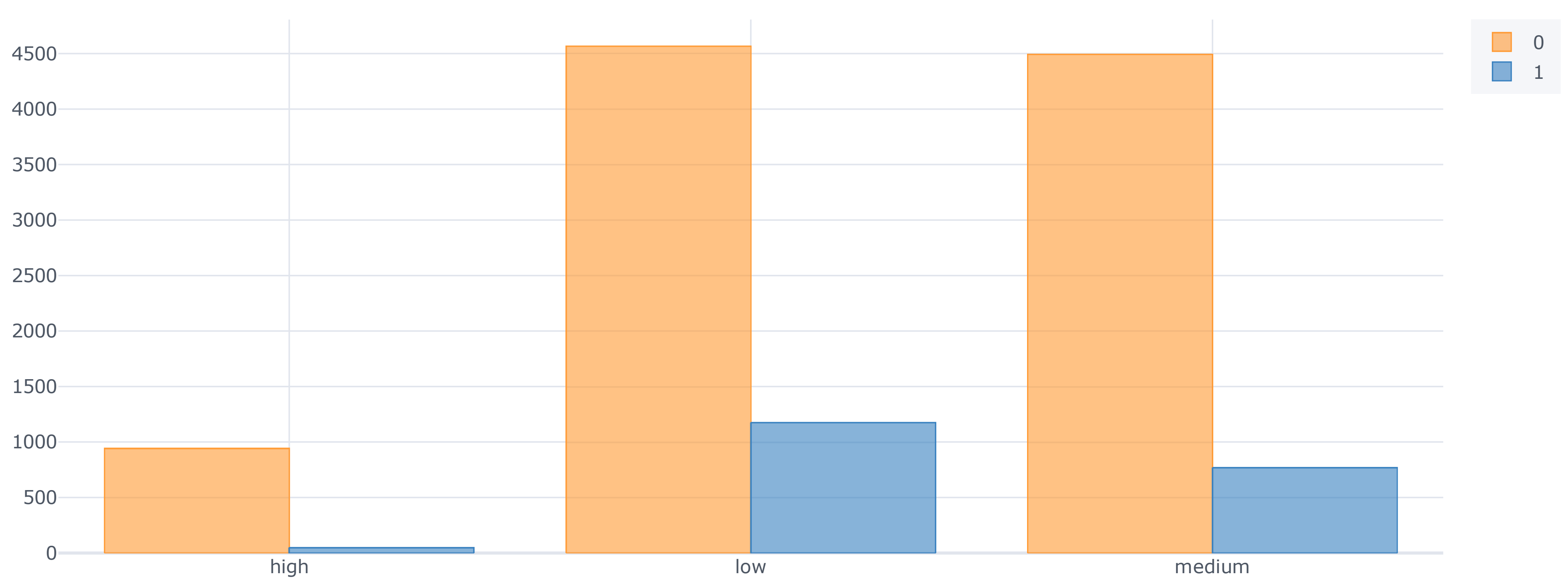
In [57]:
```
fig = px.pie(df, values = df['salary'].value_counts(),
             names = (df['salary'].value_counts()).index,
             title = '"salary" Column Distribution')
fig.show()
```

"salary" Column Distribution



In [58]:
```
pd.crosstab(df['salary'], df['left']).iplot(kind='bar', title = 'salary and left')
```

salary and left



Export to plot.ly »

- It is seen that the left percentages of the salary are similar.
- Even if it is small, there is an increase in the form of high-medium-low according to the salary status.

Let's go on with the examination of numerical and categorical columns.

In [59]:
```python
numerical= df.drop(['left'], axis = 1).select_dtypes('number').columns

categorical = df.select_dtypes('object').columns

print('---------------------')
print(f'Numerical Columns:  {df[numerical].columns}')
print(f'Categorical Columns: {df[categorical].columns}')
print('---------------------')
```

```
---------------------
Numerical Columns:  Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'work_accident',
       'promotion_last_5years'],
      dtype='object')
Categorical Columns: Index(['department', 'salary'], dtype='object')
---------------------
```

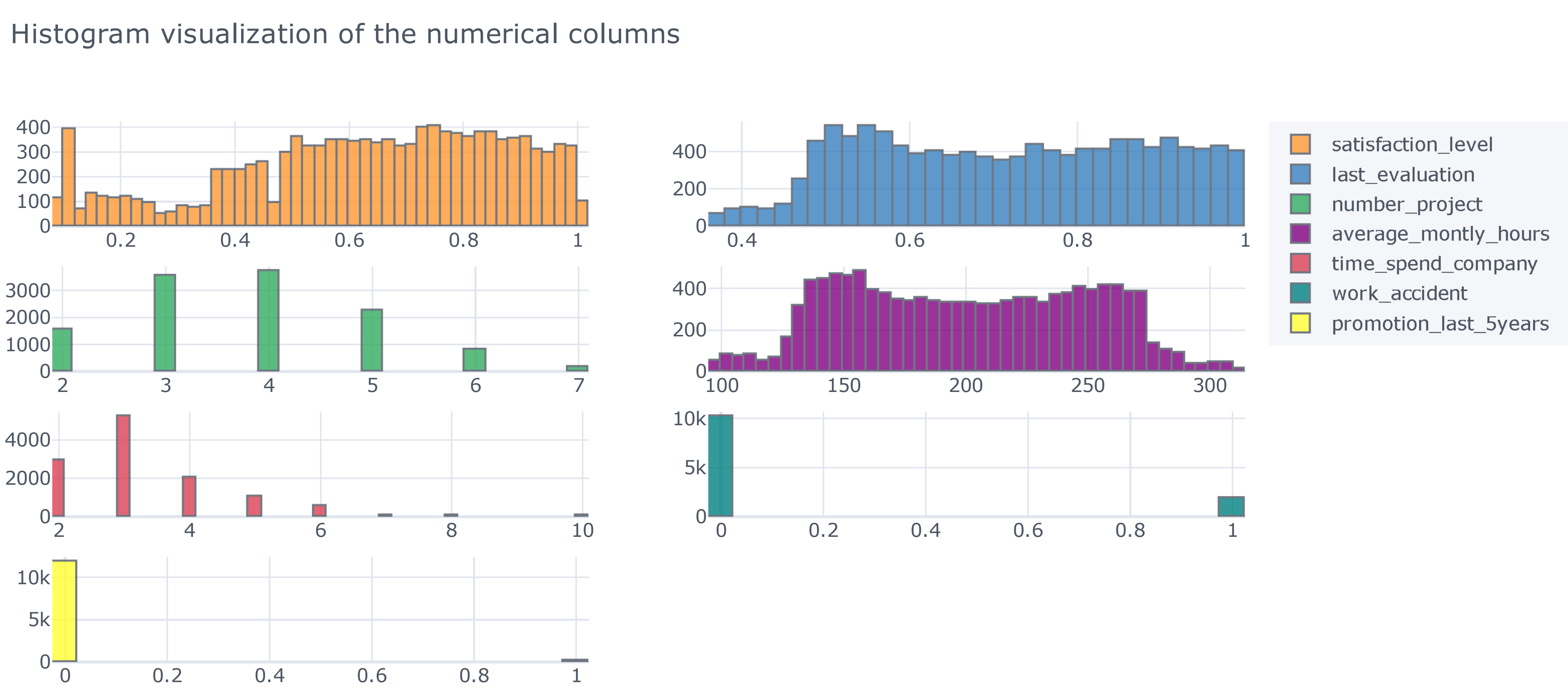
In [60]:
```python
cprint("The describe values of the numerical columns",'green', 'on_black')
df[numerical].describe().T.style.background_gradient(subset = ['mean','std','50%','count'], cmap = 'RdPu')
```

```
The describe values of the numerical columns
```

Out[60]:

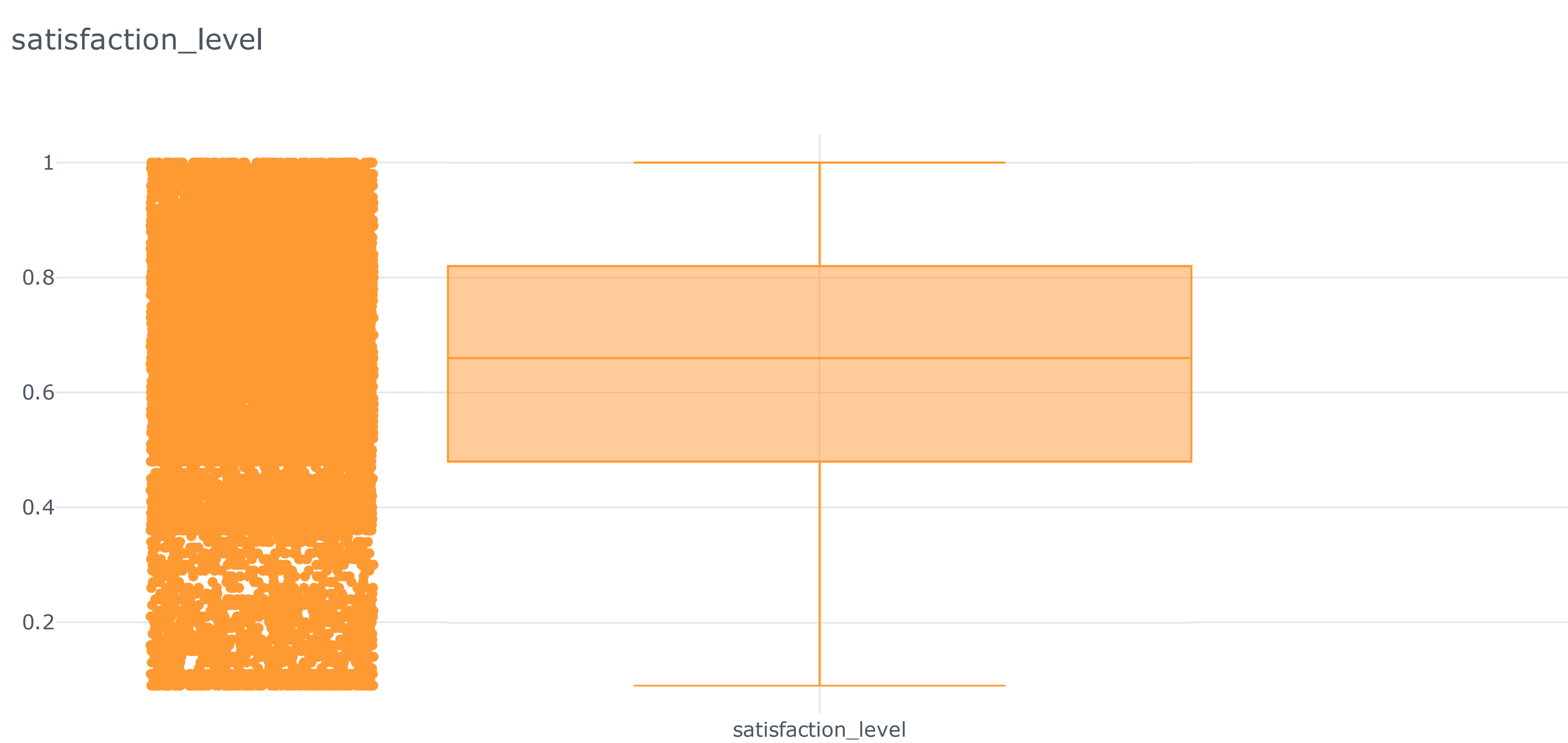
	count	mean	std	min	25%	50%	75%	max
satisfaction_level	11991.000000	0.629658	0.241070	0.090000	0.480000	0.660000	0.820000	1.000000
last_evaluation	11991.000000	0.716683	0.168343	0.360000	0.570000	0.720000	0.860000	1.000000
number_project	11991.000000	3.802852	1.163238	2.000000	3.000000	4.000000	5.000000	7.000000
average_montly_hours	11991.000000	200.473522	48.727813	96.000000	157.000000	200.000000	243.000000	310.000000
time_spend_company	11991.000000	3.364857	1.330240	2.000000	3.000000	3.000000	4.000000	10.000000
work_accident	11991.000000	0.154282	0.361234	0.000000	0.000000	0.000000	0.000000	1.000000
promotion_last_5years	11991.000000	0.016929	0.129012	0.000000	0.000000	0.000000	0.000000	1.000000

In [61]:
```python
df[numerical].iplot(kind = 'histogram', subplots = True, bins = 50, title = 'Histogram visualization of the numerical columns')
```



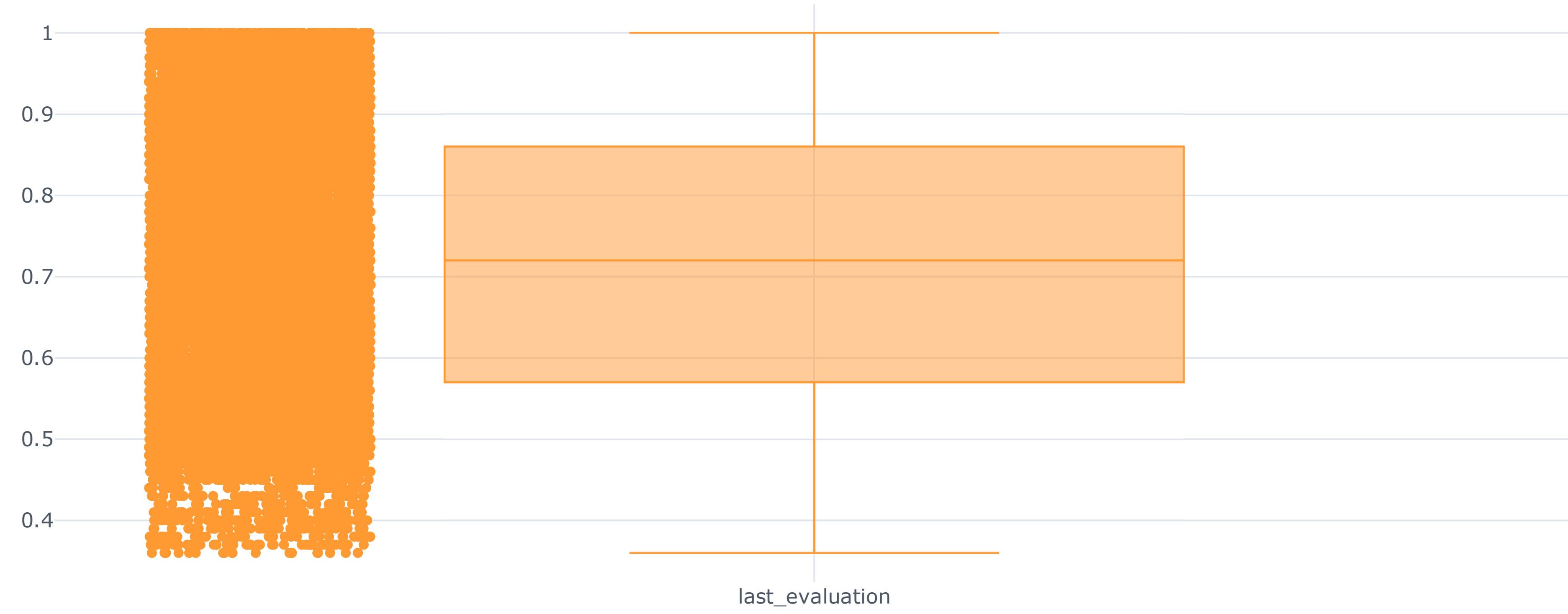
Export to plot.ly »

In [62]:
```python
for i in numerical:
    df[i].iplot(kind = 'box', title = i, boxpoints = 'all')
```



Export to plot.ly »

last_evaluation

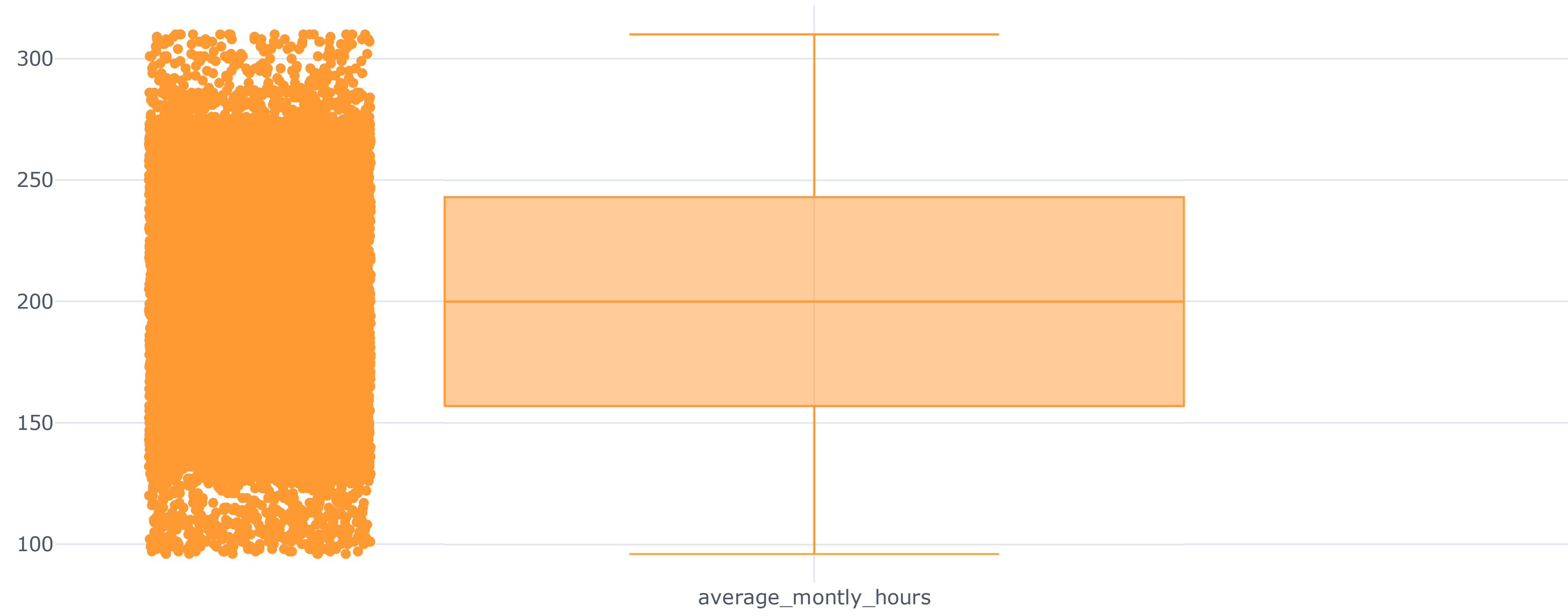


Export to plot.ly »

number_project

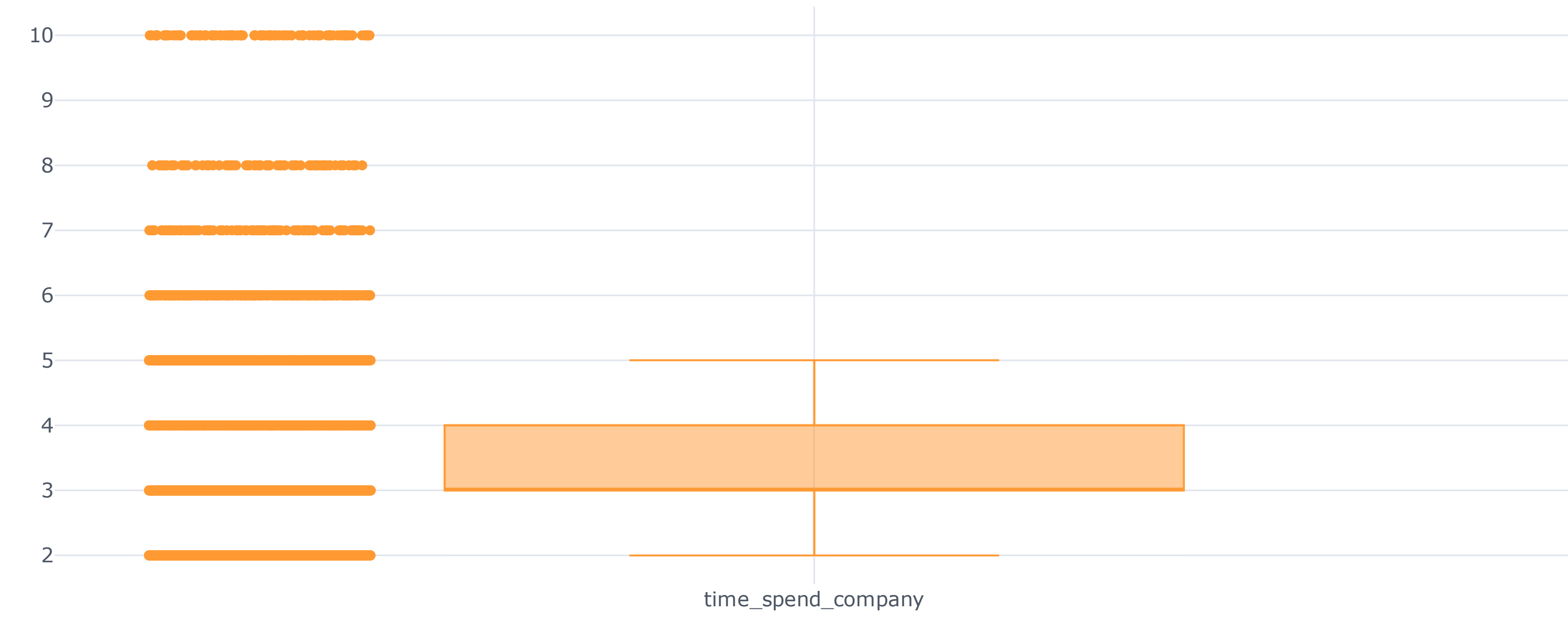
Export to plot.ly »

average_montly_hours



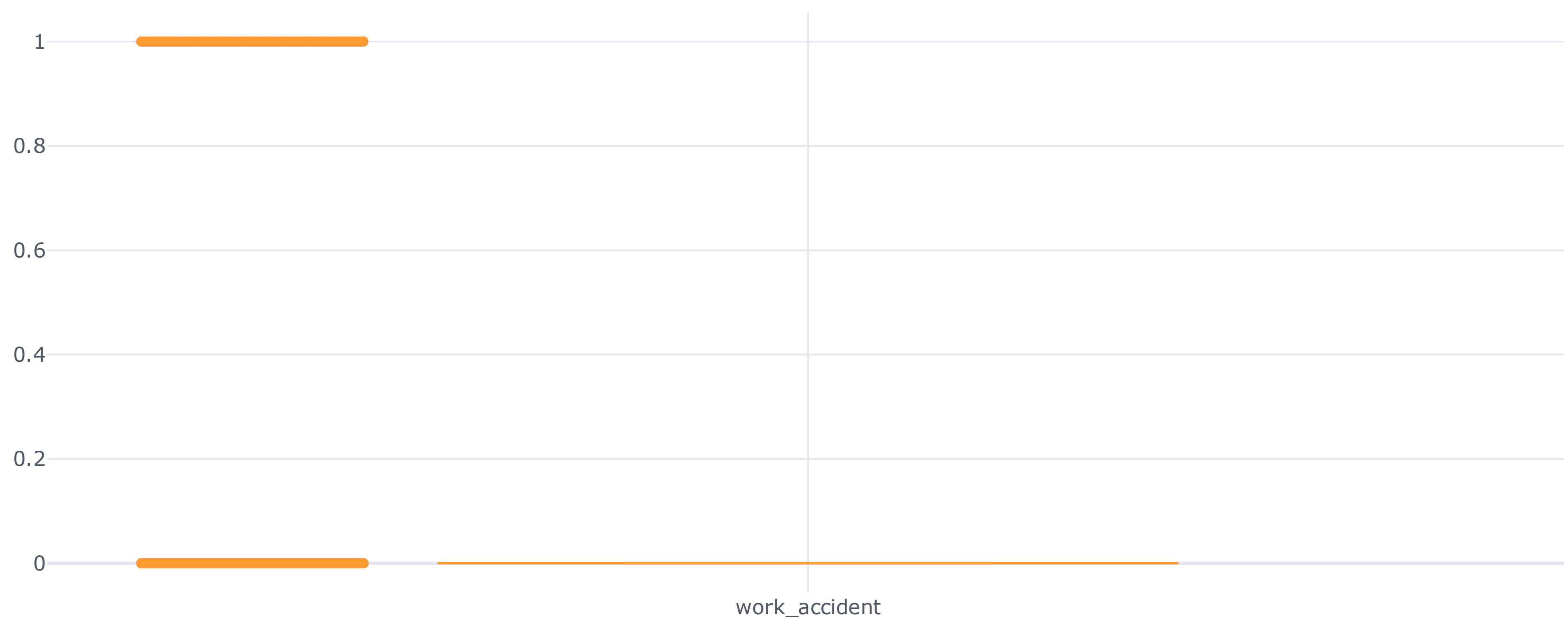
Export to plot.ly »

time_spend_company



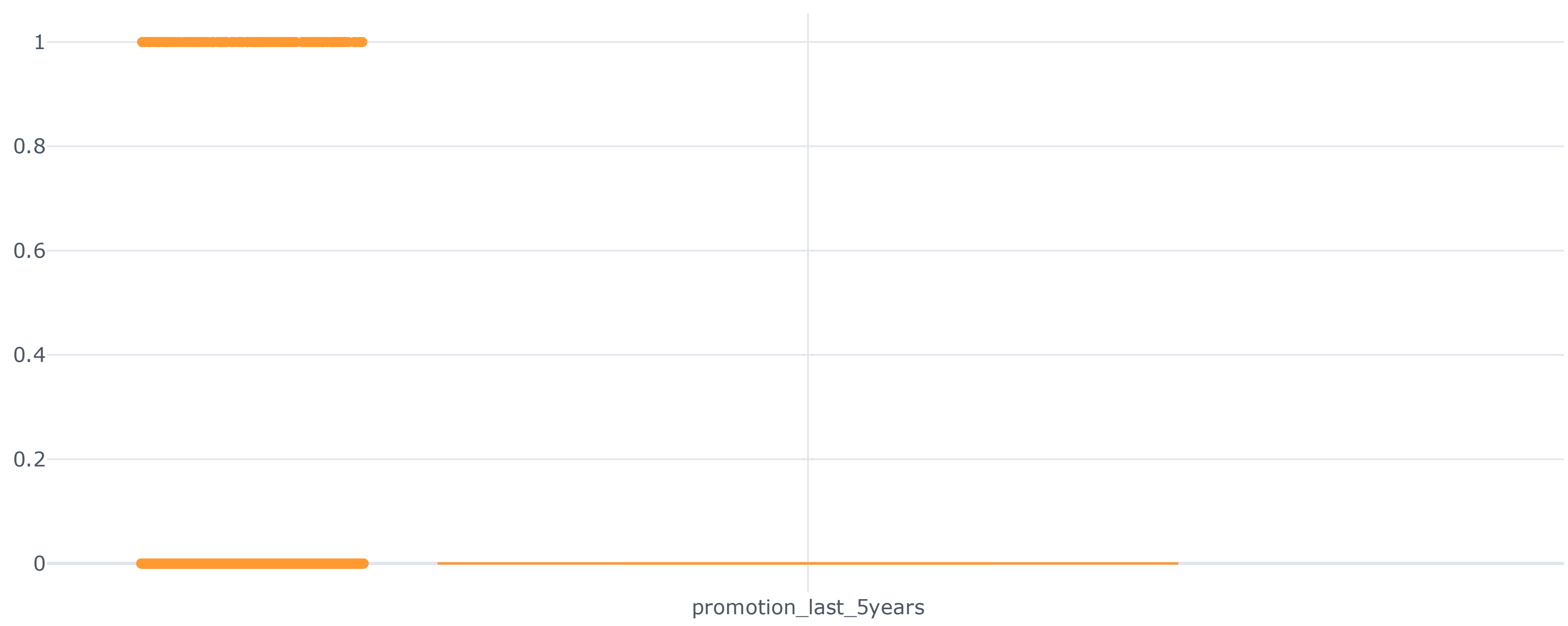
Export to plot.ly »

work_accident



Export to plot.ly »

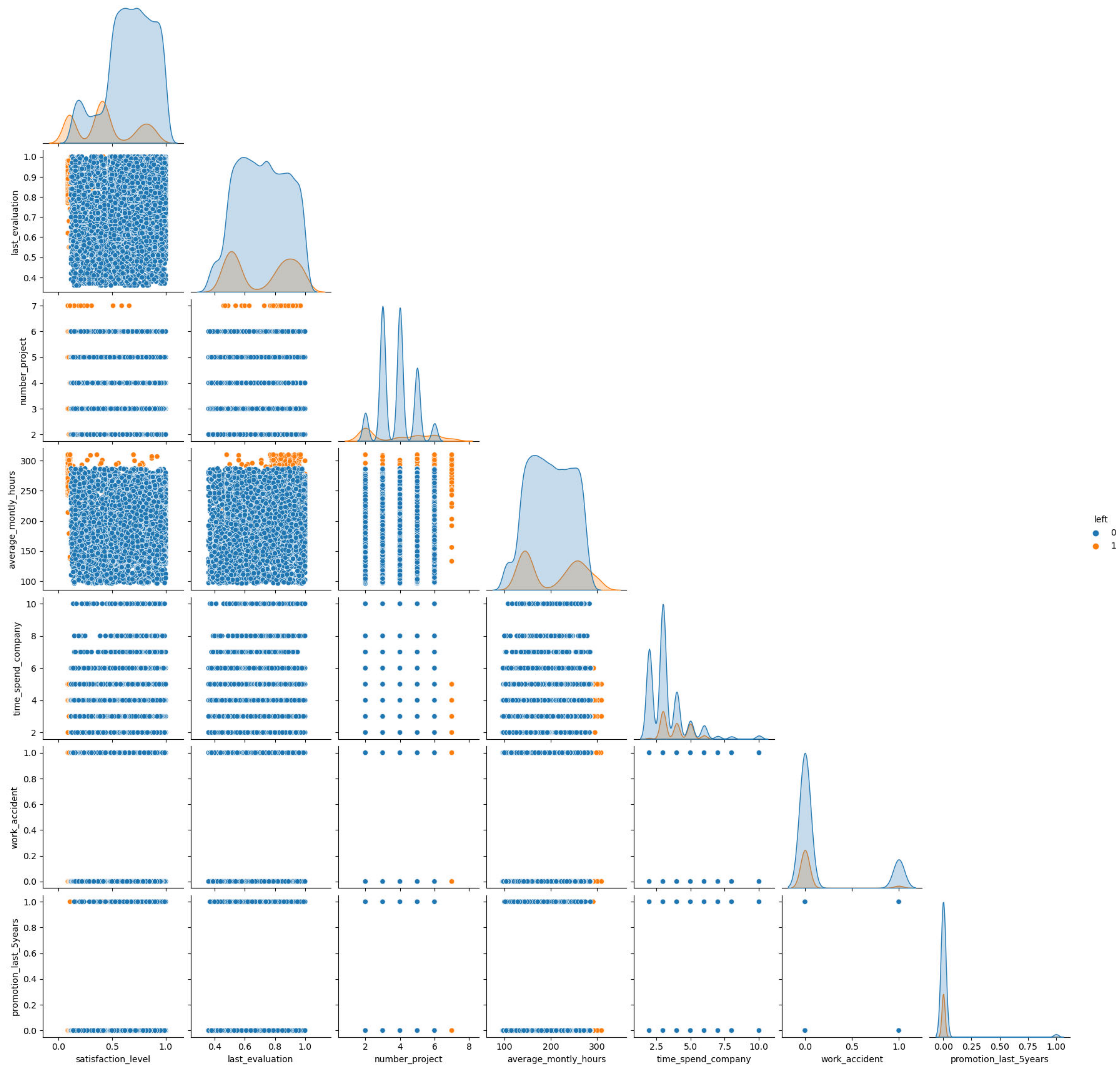
promotion_last_5years



Export to plot.ly »

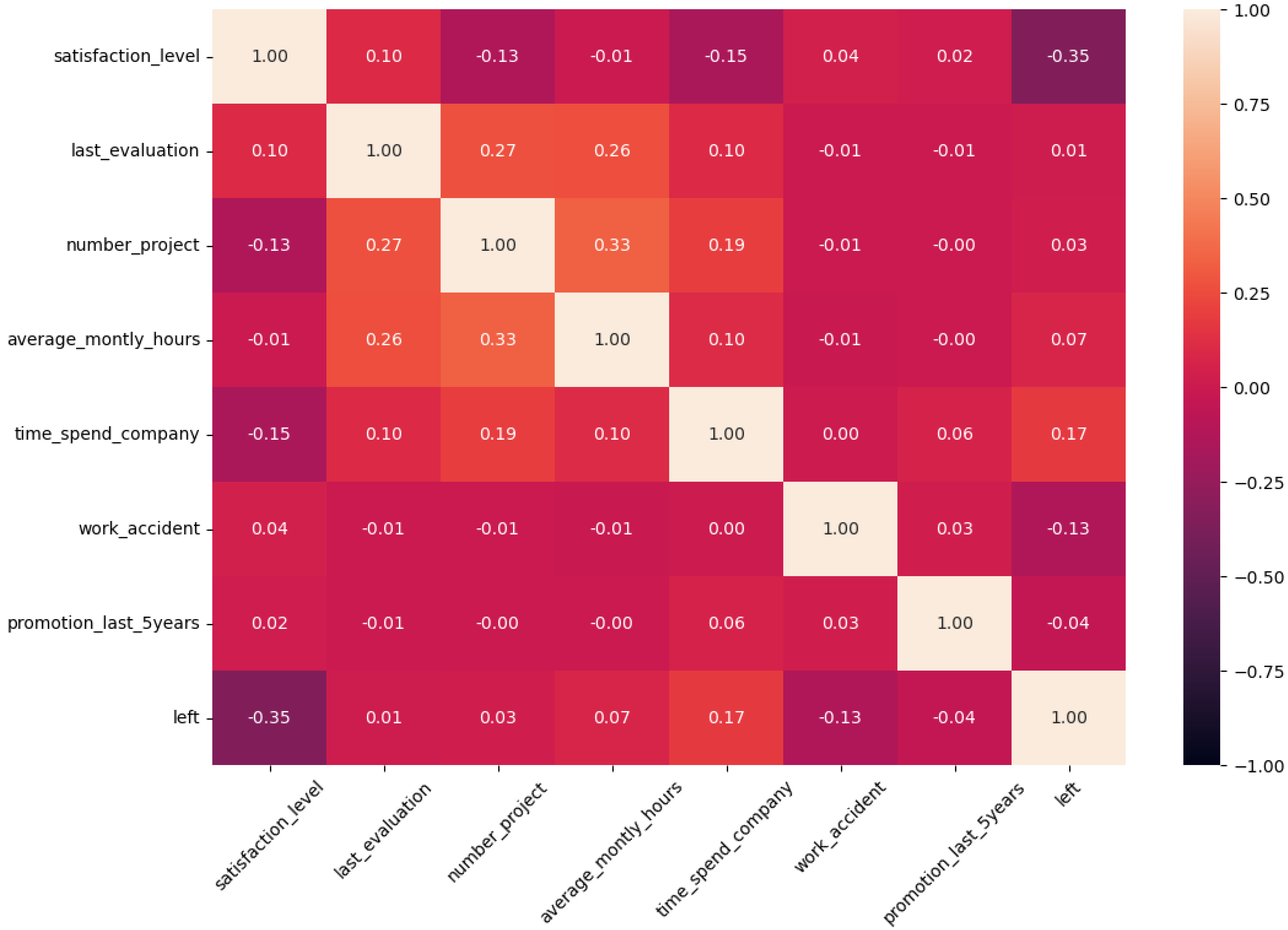
In [63]:
```
cprint("The pairplot visualization of the numerical columns",'green', 'on_red')
sns.pairplot(df, hue = "left", corner = True);
```

```
The pairplot visualization of the numerical columns
```



In [64]:
```python
cprint("Heatmap of the numerical columns",'green', 'on_red')

plt.figure(figsize = (12, 8))
sns.heatmap (df.corr(), annot = True, fmt = '.2f', vmin = -1, vmax = 1)
plt.xticks(rotation = 45);
```

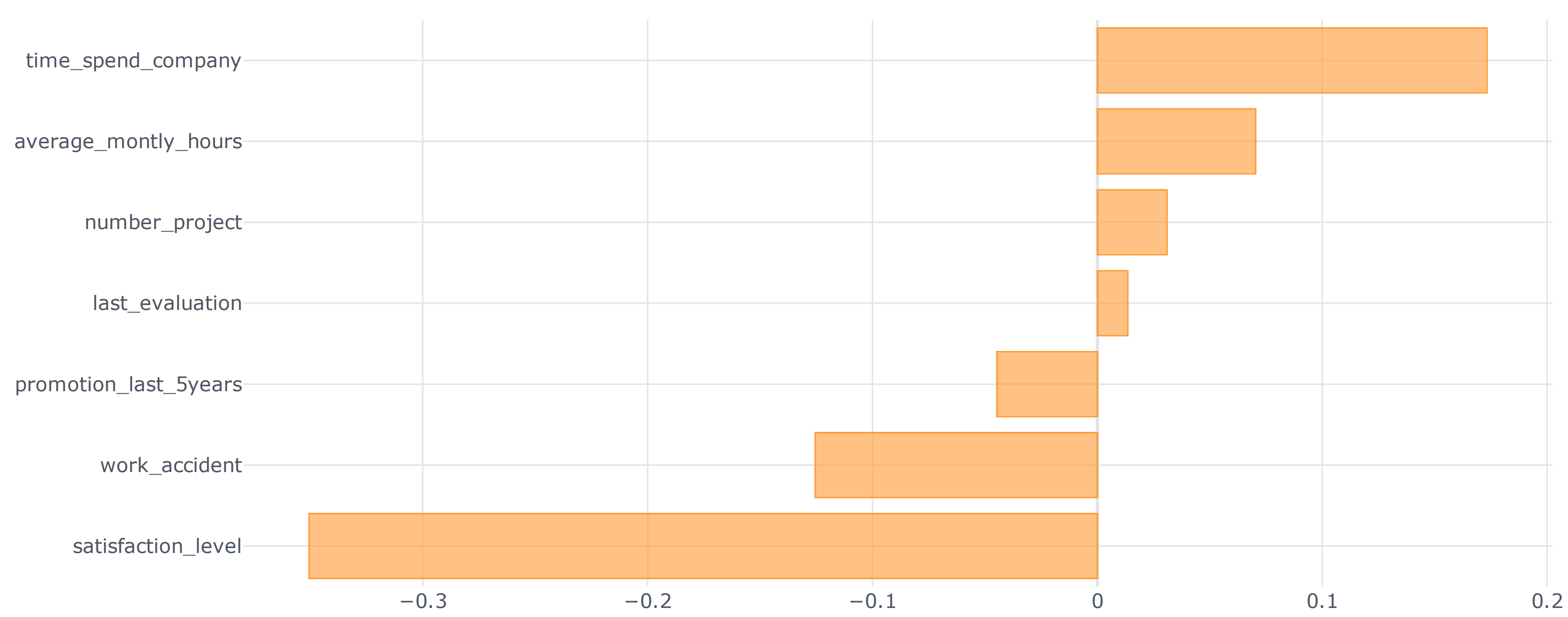
```
Heatmap of the numerical columns
```



In [65]:
```python
cprint("Multicollinearity among the features",'green', 'on_red')

df_temp = df.corr()

count = 'Done'
feature =[]
collinear= []
for col in df_temp.columns:
    for i in df_temp.index:
        if (df_temp[col][i] > .9 and df_temp[col][i] < 1) or (df_temp[col][i] < -.9 and df_temp[col][i] > -1) :
                feature.append(col)
                collinear.append(i)
                print(Fore.RED + f'\033[1mmulticolinearity alert in between\033[0m {col} - {i}')
        else:
            print(f'For {col} and {i}, there is NO multicollinearity problem')

print('\033[1mThe number of strong corelated features:\033[0m', count)
```

```
Multicollinearity among the features
For satisfaction_level and satisfaction_level, there is NO multicollinearity problem
For satisfaction_level and last_evaluation, there is NO multicollinearity problem
For satisfaction_level and number_project, there is NO multicollinearity problem
For satisfaction_level and average_montly_hours, there is NO multicollinearity problem
For satisfaction_level and time_spend_company, there is NO multicollinearity problem
For satisfaction_level and work_accident, there is NO multicollinearity problem
For satisfaction_level and promotion_last_5years, there is NO multicollinearity problem
For satisfaction_level and left, there is NO multicollinearity problem
For last_evaluation and satisfaction_level, there is NO multicollinearity problem
For last_evaluation and last_evaluation, there is NO multicollinearity problem
For last_evaluation and number_project, there is NO multicollinearity problem
For last_evaluation and average_montly_hours, there is NO multicollinearity problem
For last_evaluation and time_spend_company, there is NO multicollinearity problem
For last_evaluation and work_accident, there is NO multicollinearity problem
For last_evaluation and promotion_last_5years, there is NO multicollinearity problem
For last_evaluation and left, there is NO multicollinearity problem
For number_project and satisfaction_level, there is NO multicollinearity problem
For number_project and last_evaluation, there is NO multicollinearity problem
For number_project and number_project, there is NO multicollinearity problem
For number_project and average_montly_hours, there is NO multicollinearity problem
For number_project and time_spend_company, there is NO multicollinearity problem
For number_project and work_accident, there is NO multicollinearity problem
For number_project and promotion_last_5years, there is NO multicollinearity problem
For number_project and left, there is NO multicollinearity problem
For average_montly_hours and satisfaction_level, there is NO multicollinearity problem
For average_montly_hours and last_evaluation, there is NO multicollinearity problem
For average_montly_hours and number_project, there is NO multicollinearity problem
For average_montly_hours and average_montly_hours, there is NO multicollinearity problem
For average_montly_hours and time_spend_company, there is NO multicollinearity problem
For average_montly_hours and work_accident, there is NO multicollinearity problem
For average_montly_hours and promotion_last_5years, there is NO multicollinearity problem
For average_montly_hours and left, there is NO multicollinearity problem
For time_spend_company and satisfaction_level, there is NO multicollinearity problem
For time_spend_company and last_evaluation, there is NO multicollinearity problem
For time_spend_company and number_project, there is NO multicollinearity problem
For time_spend_company and average_montly_hours, there is NO multicollinearity problem
For time_spend_company and time_spend_company, there is NO multicollinearity problem
For time_spend_company and work_accident, there is NO multicollinearity problem
For time_spend_company and promotion_last_5years, there is NO multicollinearity problem
For time_spend_company and left, there is NO multicollinearity problem
For work_accident and satisfaction_level, there is NO multicollinearity problem
For work_accident and last_evaluation, there is NO multicollinearity problem
For work_accident and number_project, there is NO multicollinearity problem
For work_accident and average_montly_hours, there is NO multicollinearity problem
For work_accident and time_spend_company, there is NO multicollinearity problem
For work_accident and work_accident, there is NO multicollinearity problem
For work_accident and promotion_last_5years, there is NO multicollinearity problem
For work_accident and left, there is NO multicollinearity problem
For promotion_last_5years and satisfaction_level, there is NO multicollinearity problem
For promotion_last_5years and last_evaluation, there is NO multicollinearity problem
For promotion_last_5years and number_project, there is NO multicollinearity problem
For promotion_last_5years and average_montly_hours, there is NO multicollinearity problem
For promotion_last_5years and time_spend_company, there is NO multicollinearity problem
For promotion_last_5years and work_accident, there is NO multicollinearity problem
For promotion_last_5years and promotion_last_5years, there is NO multicollinearity problem
For promotion_last_5years and left, there is NO multicollinearity problem
For left and satisfaction_level, there is NO multicollinearity problem
For left and last_evaluation, there is NO multicollinearity problem
For left and number_project, there is NO multicollinearity problem
For left and average_montly_hours, there is NO multicollinearity problem
For left and time_spend_company, there is NO multicollinearity problem
For left and work_accident, there is NO multicollinearity problem
For left and promotion_last_5years, there is NO multicollinearity problem
For left and left, there is NO multicollinearity problem
```
The number of strong corelated features: Done

In [66]:
```
df.corr()['left'].sort_values().drop('left').iplot(kind = 'barh');
```



Export to plot.ly »

Based on the examinations made above,

- There is no multicollinearity problem among the features.
- We have weak level correlation between the numerical features and the target column.
- Also there is weak level correlation between the columns.
- Target variable demonstrates a slight negative correlation with the variables of "promotion_last_5years", "work_accident", "satisfaction_level",
- Target variable demonstrates slight positive correlation with the variables of 'time_spend_company', 'average_montly_hours', 'number_project' and 'last_evaluation".
- satisfaction_level has more influence on the decision to leave the work than the other columns.

5 - DATA VISUALIZATION

- Employees Left
- Determine Number of Projects
- Determine Time Spent in Company

- Subplots of Features

We can search for answers to the following questions using data visualization methods. Based on these responses, we can develop comments about the factors that cause churn.

- How does the promotion status affect employee churn?
- How does years of experience affect employee churn?
- How does workload affect employee churn?
- How does the salary level affect employee churn?

5.1 - Employees Left

Let's check how many employees were left?
Here, we can plot a bar graph using Matplotlib. The bar graph is suitable for showing discrete variable counts.

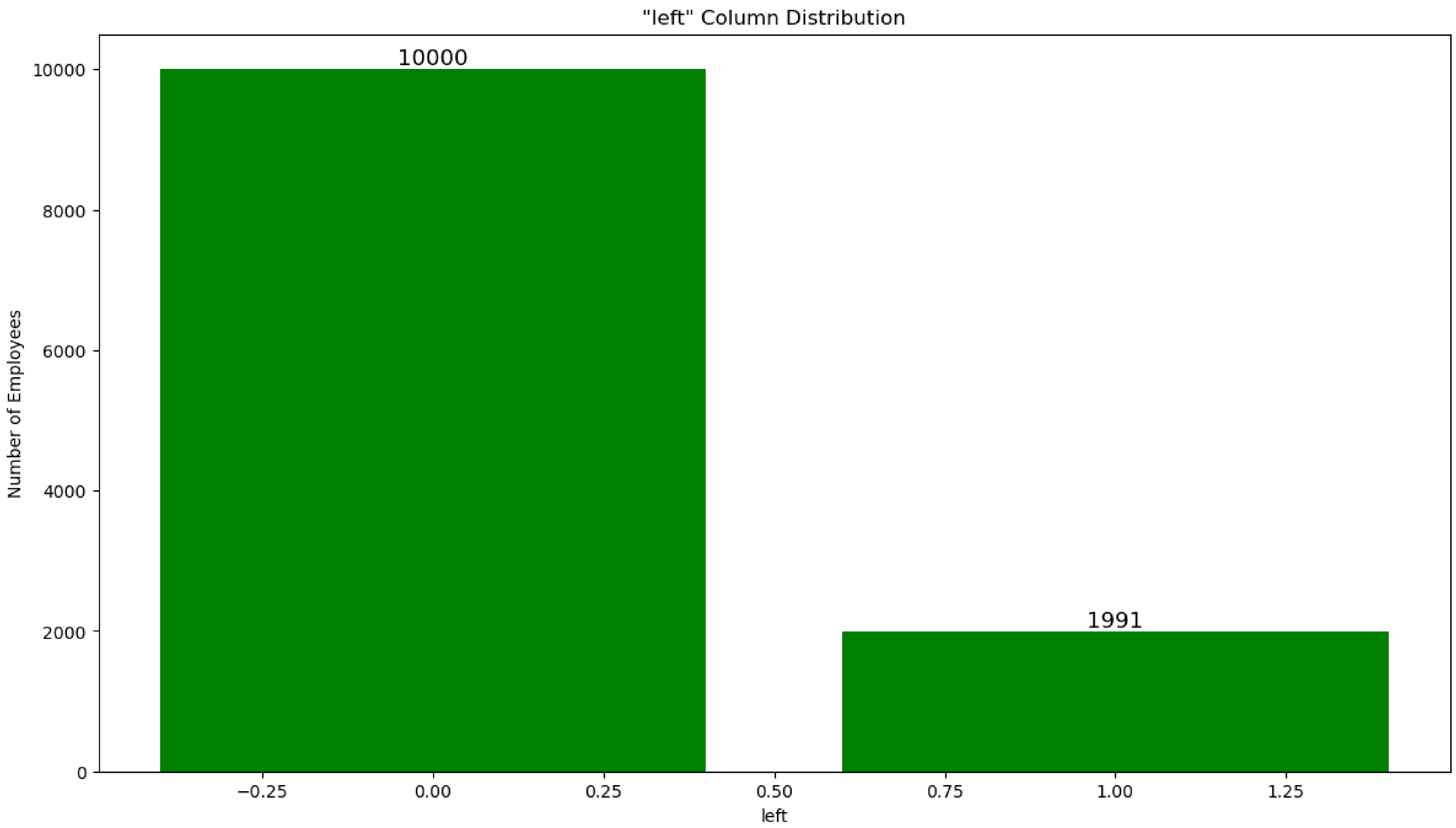
In [67]:
```
cprint('"left" Column Distribution','green', 'on_red')
df.left.value_counts()
```

```
"left" Column Distribution
```

Out[67]:
```
0    10000
1     1991
Name: left, dtype: int64
```

In [68]:
```
cprint('"left" Column Distribution','green', 'on_red')
fig = plt.figure(figsize = (11,6))
ax = fig.add_axes([0,0,1,1])
ax.bar(df.left.value_counts().index, df.left.value_counts().values, color = 'green')
plt.title('"left" Column Distribution')
plt.xlabel('left')
plt.ylabel('Number of Employees')
for index,value in enumerate(df.left.value_counts()):
    plt.text(index, value, f'{value}', ha = 'center', va = 'bottom', fontsize = 13)
plt.show()
```

```
"left" Column Distribution
```



5.2 - Number of Projects

Similarly, we can also plot a bar graph to count the number of employees deployed on how many projects?

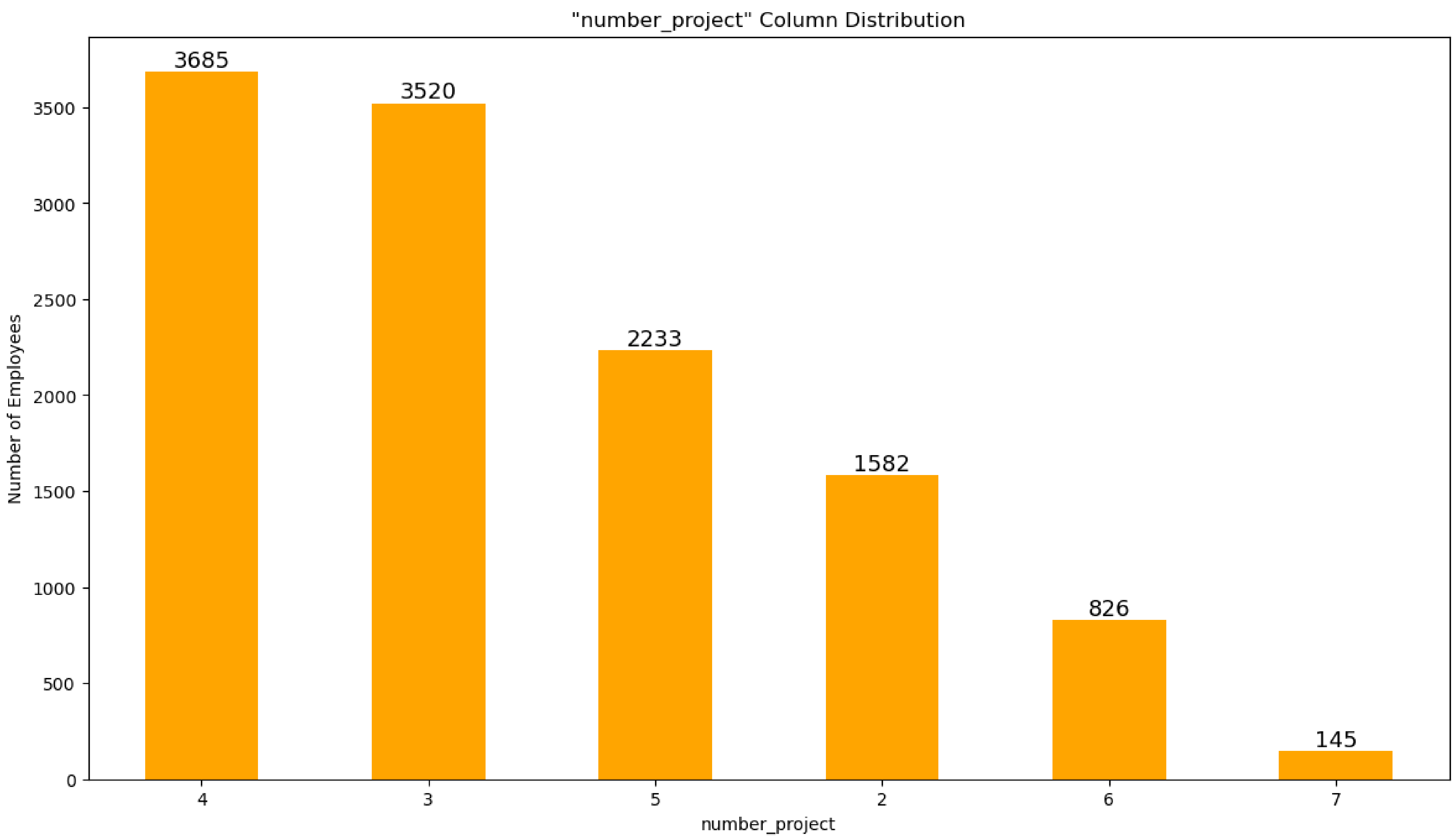
In [69]:
```
cprint('"number_project" Column Distribution','green', 'on_red')
df.number_project.value_counts()
```

```
"number_project" Column Distribution
```

Out[69]:
```
4    3685
3    3520
5    2233
2    1582
6     826
7     145
Name: number_project, dtype: int64
```

In [70]:
```
cprint('"number_project" Column Distribution','green', 'on_red')
fig = plt.figure(figsize = (11,6))
ax = fig.add_axes([0,0,1,1])
# x = df.number_project.value_counts().index
# y = df.number_project.value_counts().values
df.number_project.value_counts().plot(kind = "bar", color = "orange")
plt.title('"number_project" Column Distribution')
plt.xlabel('number_project')
plt.ylabel('Number of Employees')
plt.xticks(rotation = 0)
for index,value in enumerate(df.number_project.value_counts().sort_values(ascending=False)):
    plt.text(index, value, f'{value}', ha = 'center', va = 'bottom', fontsize = 13)
plt.show()
```

```
"number_project" Column Distribution
```



5.3 - Time Spent in the Company

Similarly, we can also plot a bar graph to count the number of employees have based on how much experience?

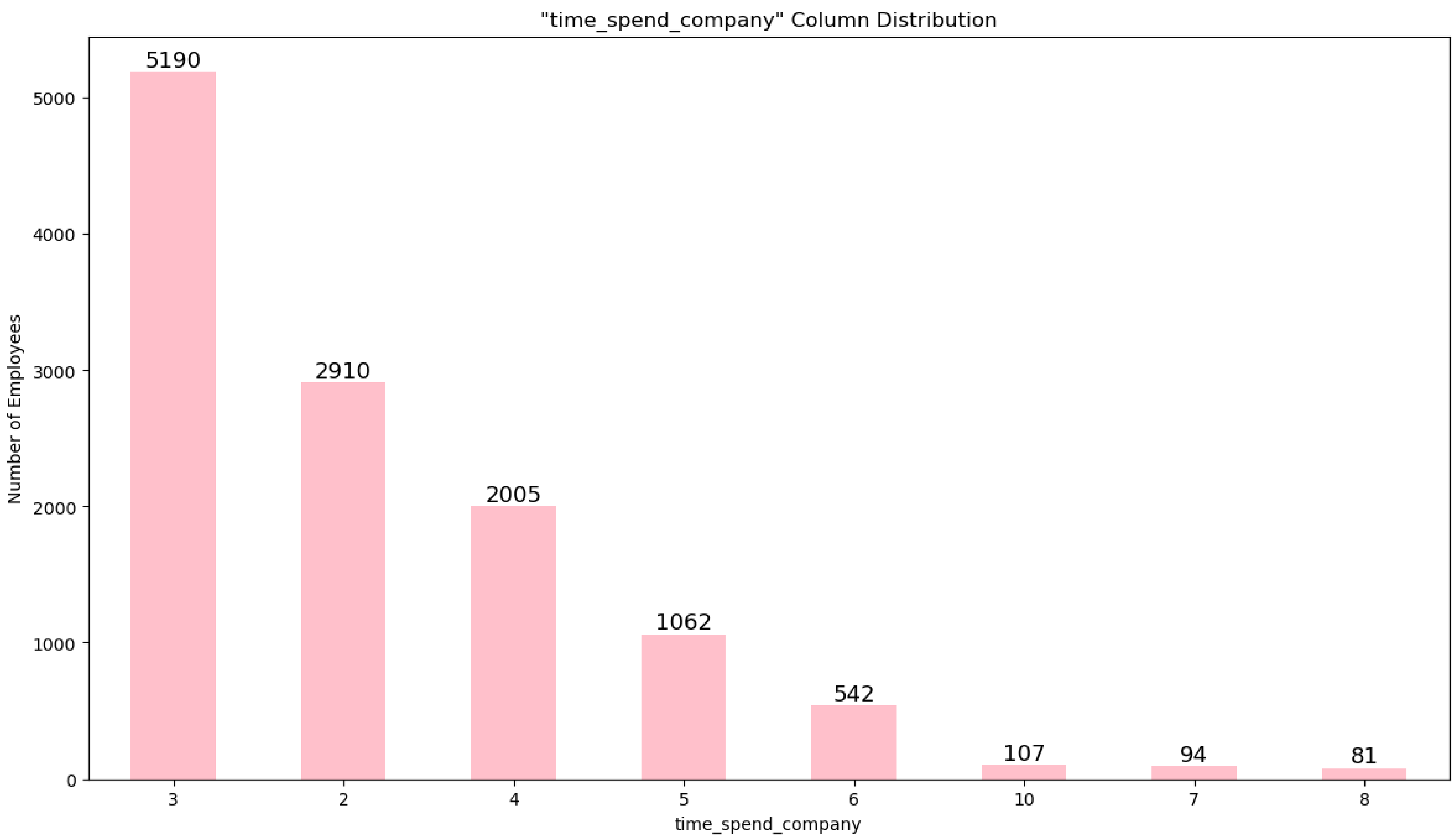
In [71]:
```
cprint('"time_spend_company" Column Distribution','green', 'on_red')
df.time_spend_company.value_counts()
```

```
"time_spend_company" Column Distribution
```

Out[71]:
```
3     5190
2     2910
4     2005
5     1062
6      542
10     107
7       94
8       81
Name: time_spend_company, dtype: int64
```

In [72]:
```
cprint('"time_spend_company" Column Distribution','green', 'on_red')
fig = plt.figure(figsize = (11,6))
ax = fig.add_axes([0,0,1,1])
df.time_spend_company.value_counts().plot(kind = "bar", color = "pink")
plt.title('"time_spend_company" Column Distribution')
plt.xlabel('time_spend_company')
plt.ylabel('Number of Employees')
plt.xticks(rotation = 0)
for index,value in enumerate(df.time_spend_company.value_counts().sort_values(ascending=False)):
    plt.text(index, value, f'{value}', ha = 'center', va = 'bottom', fontsize = 13)
plt.show()
```

```
"time_spend_company" Column Distribution
```

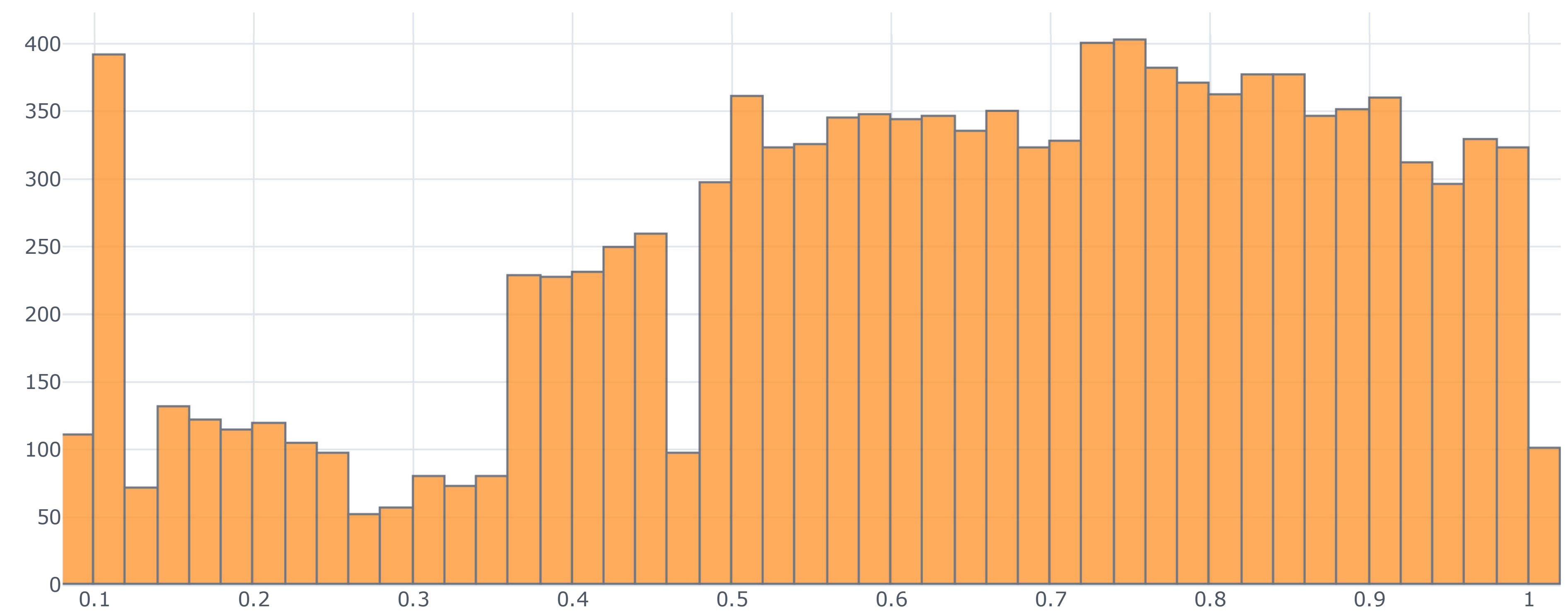


5.4 - Subplots of Features

We can use the methods of the plotly.

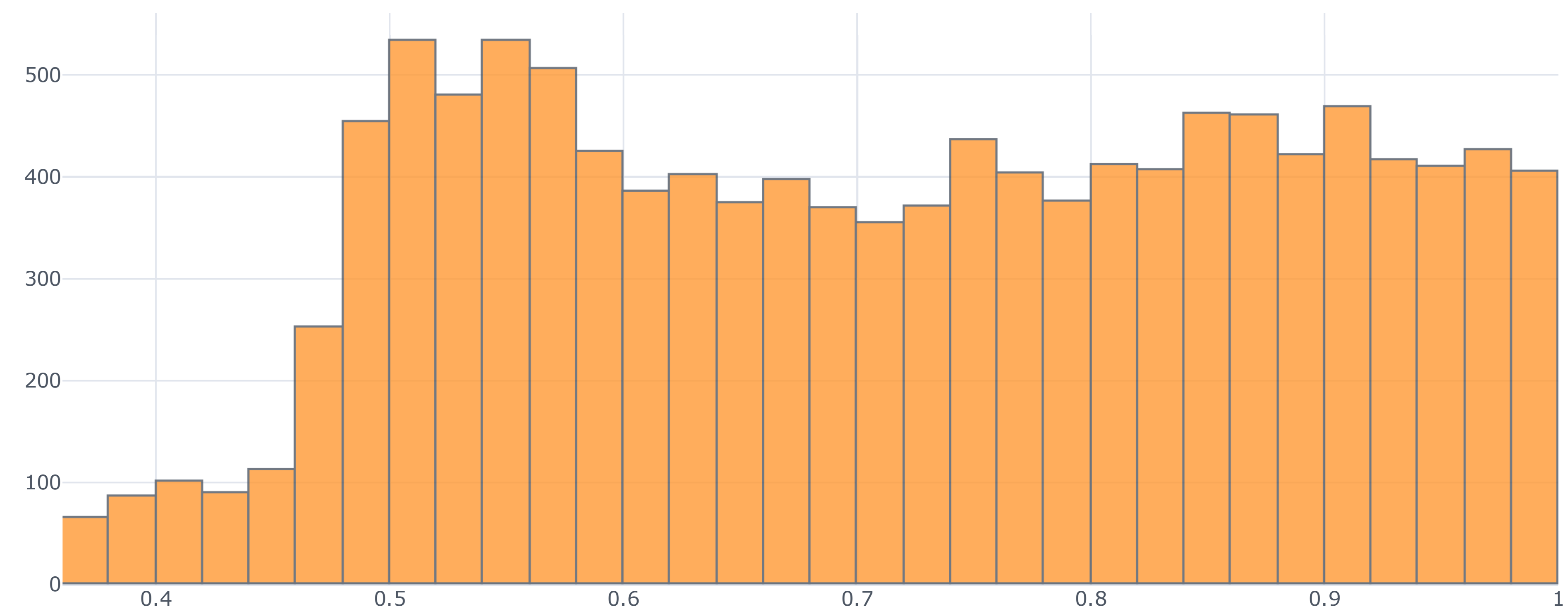
In [73]:
```
for i in df:
    df[i].iplot(kind = 'histogram', subplots = True, bins = 50, title = 'Subplots of Features')
```

Subplots of Features



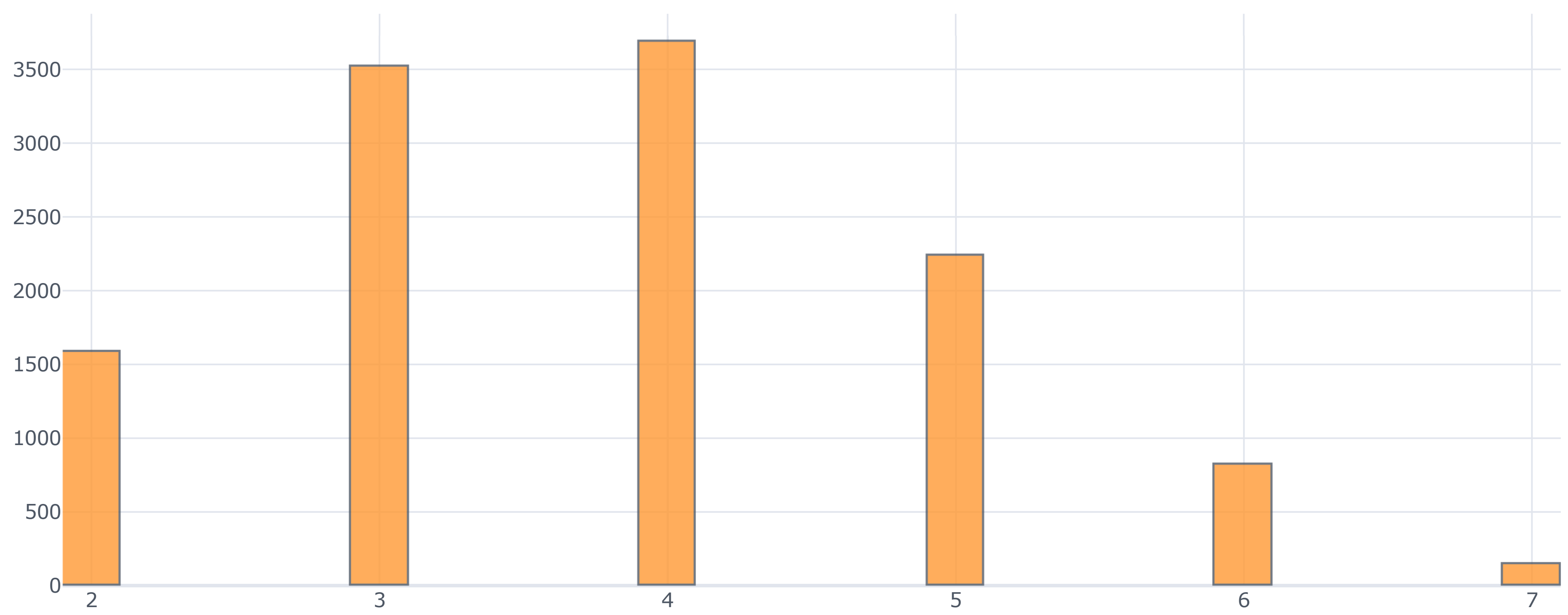
Export to plot.ly »

Subplots of Features



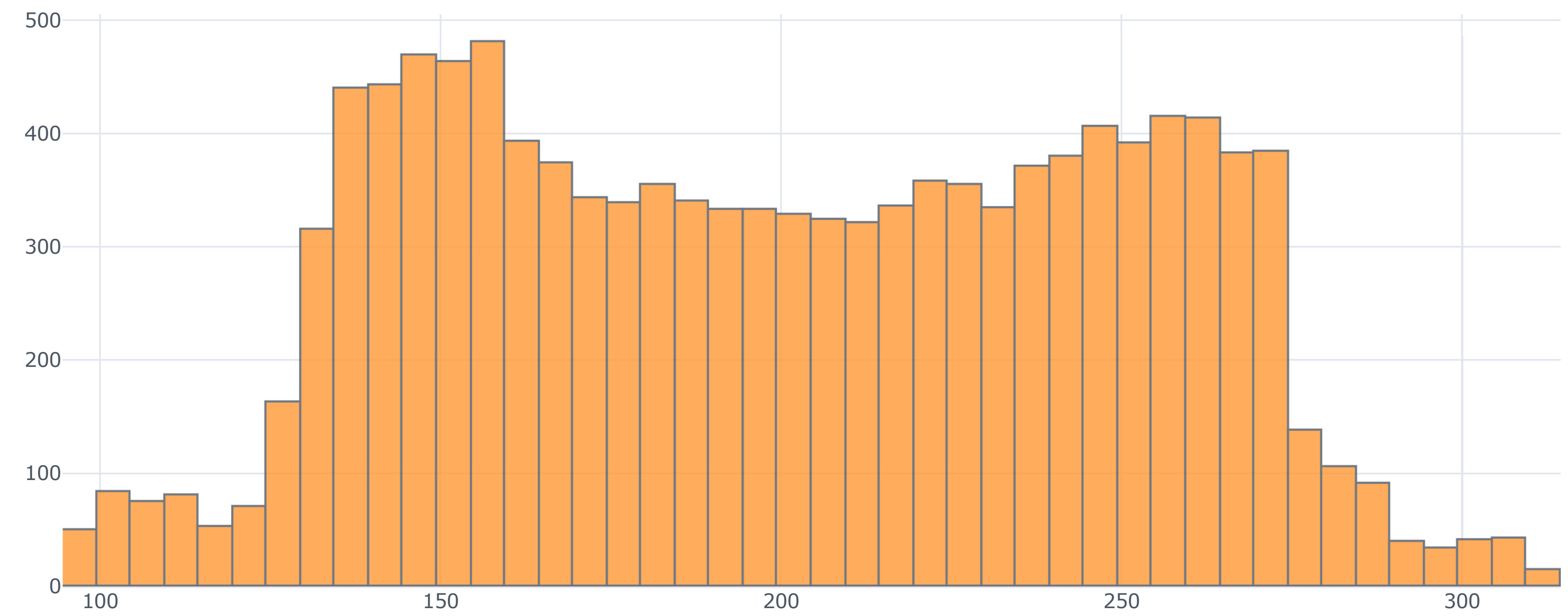
Export to plot.ly »

Subplots of Features



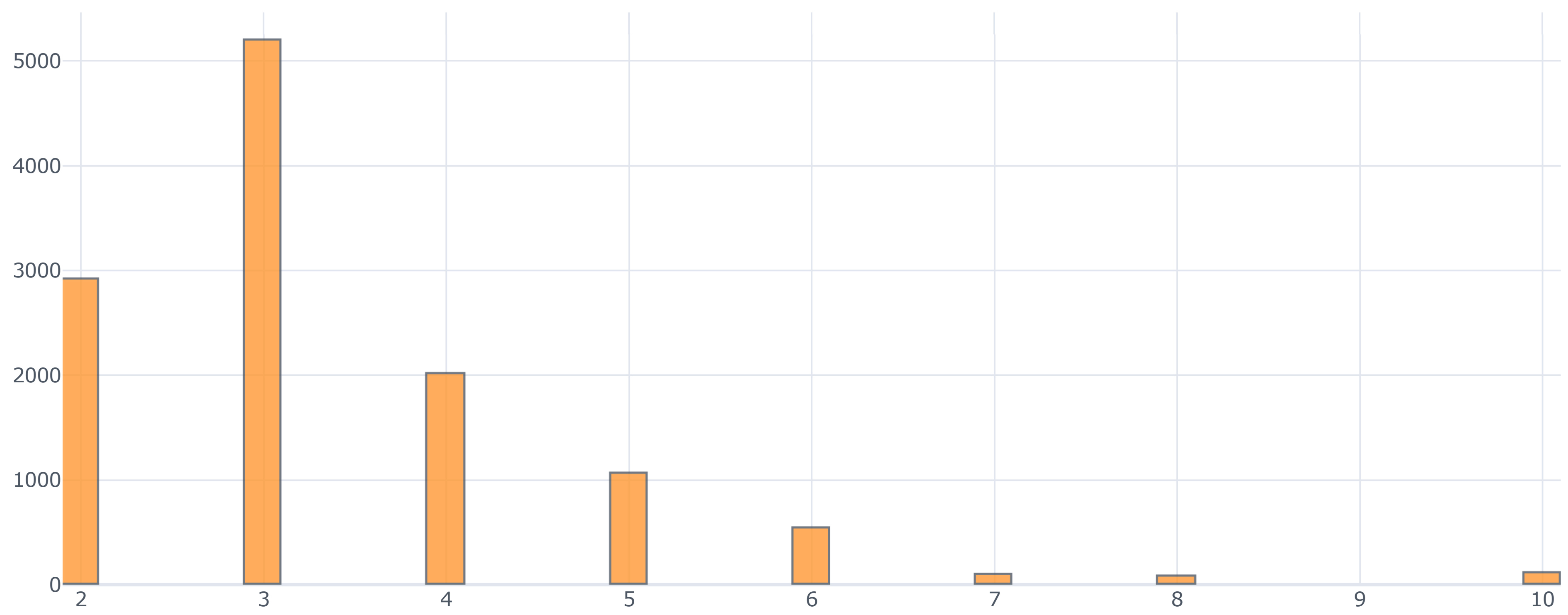
Export to plot.ly »

Subplots of Features



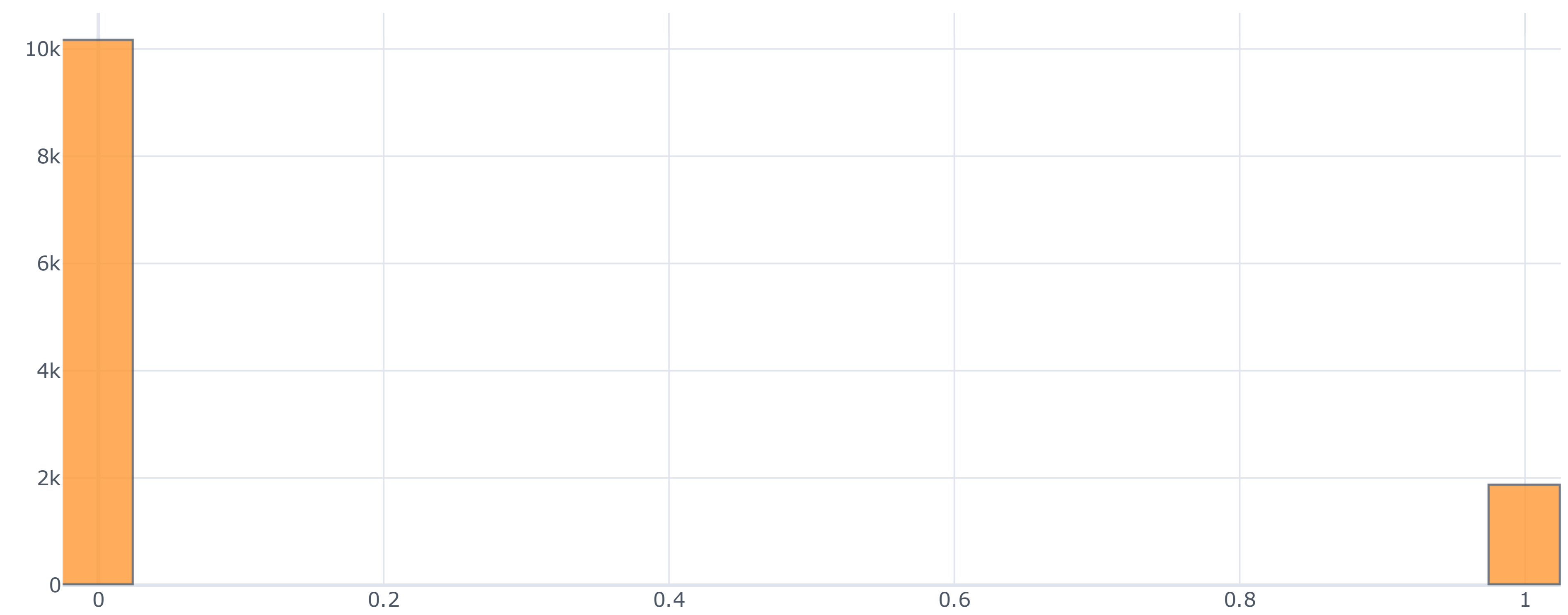
Export to plot.ly »

Subplots of Features



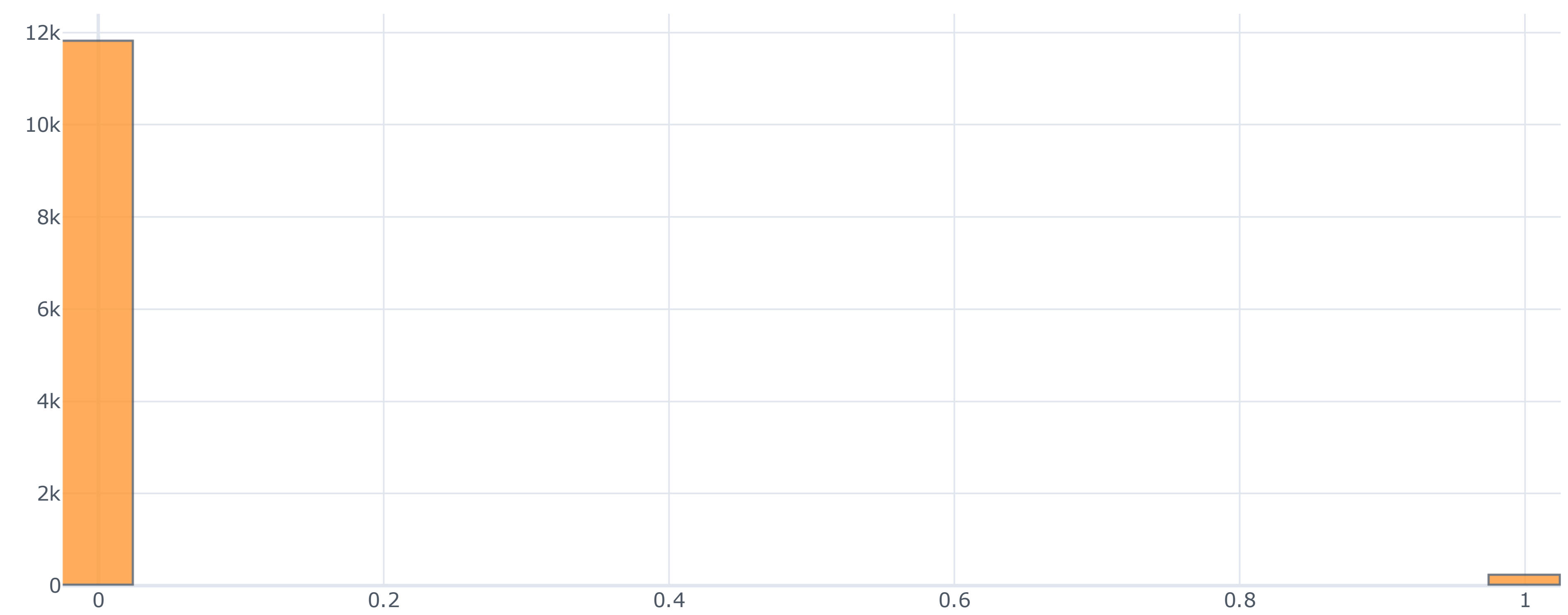
Export to plot.ly »

Subplots of Features



Export to plot.ly »

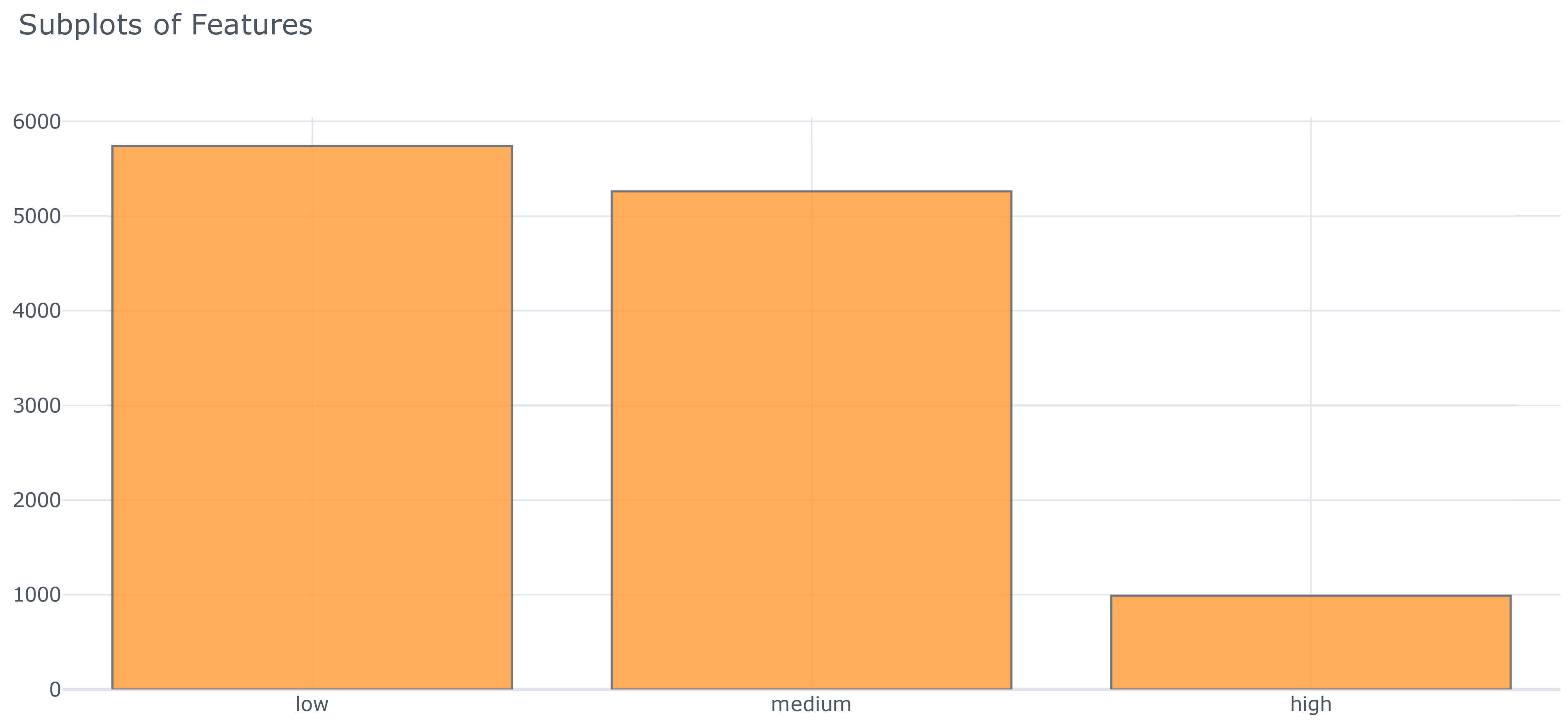
Subplots of Features



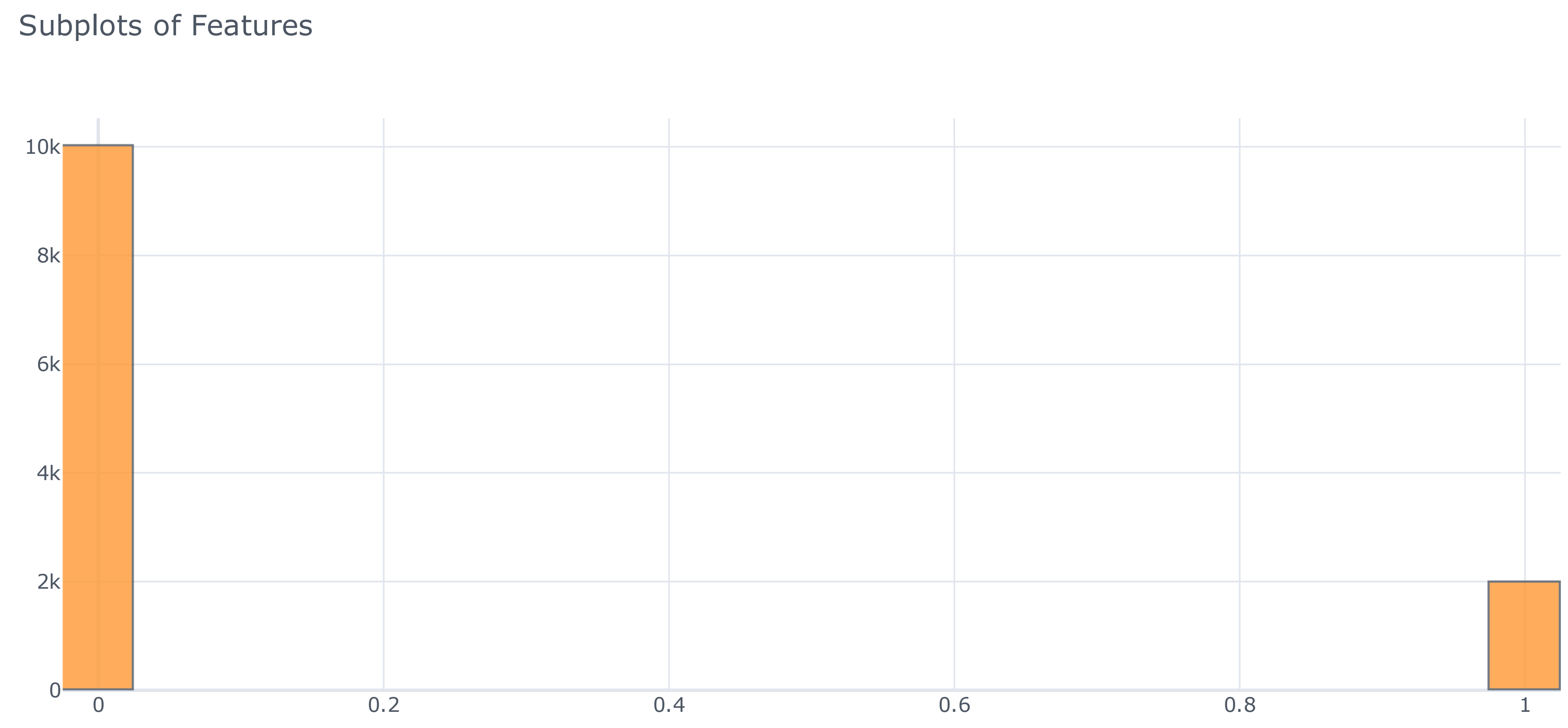
Export to plot.ly »

Subplots of Features

Export to plot.ly »



Export to plot.ly »



Export to plot.ly »

In [102…
```
## Still Data visualization had to be good
```

6 - DATA PRE-PROCESSING

- Label Encoding
- Scaling

6.1 - Label Encoding

Lots of machine learning algorithms require numerical input data, so you need to represent categorical columns in a numerical column. In order to encode this data, you could map each value to a number. e.g. Salary column's value can be represented as low:0, medium:1, and high:2. This process is known as label encoding, and sklearn conveniently will do this for you using LabelEncoder.

In [74]:
```
cprint('New df for Kmeans clustering','green', 'on_black')
df1 = df.drop('left', axis = 1)
df1.head(1)
```

```
New df for Kmeans clustering
```

Out[74]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	department	salary
0	0.380	0.530	2	157	3	0	0	sales	low

In [75]:
```
cprint('New df after getting dummied','green', 'on_black')
df1 = pd.get_dummies(df1, columns = ['department','salary'], drop_first = True)
df1.head(1)
```

```
New df after getting dummied
```

Out[75]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	department_RandD	department_accounting
0	0.380	0.530	2	157	3	0	0	0	(

6.2 - Scalling

Some machine learning algorithms are sensitive to feature scaling while others are virtually invariant to it. Machine learning algorithms like linear regression, logistic regression, neural network, etc. that use gradient descent as an optimization technique require data to be scaled. Also distance algorithms like KNN, K-means, and SVM are most affected by the range of features. This is because behind the scenes they are using distances between data points to determine their similarity.

Scaling Types:

- Normalization: Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.
- Standardization: Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Click here for more on scaling. (https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35)

In [76]:
```
df1.head()
```

Out[76]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	department_RandD	department_accounting
0	0.380	0.530	2	157	3	0	0	0	(
1	0.800	0.860	5	262	6	0	0	0	(
2	0.110	0.880	7	272	4	0	0	0	(
3	0.720	0.870	5	223	5	0	0	0	(
4	0.370	0.520	2	159	3	0	0	0	(

In [77]:
```
cprint('Scaling','green', 'on_black')
scaler = MinMaxScaler()
scaler.fit(df1)
#Store it separately for clustering
df1_scaled= scaler.transform(df1)
```

```
Scaling
```

In [78]:
```
df1_scaled
```

Out[78]:
```
array([[0.31868132, 0.265625  , 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.78021978, 0.78125   , 0.6       , ..., 0.        , 0.        ,
        1.        ],
       [0.02197802, 0.8125    , 1.        , ..., 0.        , 0.        ,
        1.        ],
       ...,
       [0.83516484, 0.28125   , 0.2       , ..., 0.        , 0.        ,
        0.        ],
       [0.26373626, 0.453125  , 0.2       , ..., 0.        , 0.        ,
        0.        ],
       [0.45054945, 0.578125  , 0.4       , ..., 0.        , 1.        ,
        0.        ]])
```

7 - CLUSTER ANALYSIS

- Find the optimal number of clusters (k) using the elbow method for for K-means.
- Determine the clusters by using K-Means then Evaluate predicted results.

- Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including pattern recognition, image analysis, information retrieval, bioinformatics, data compression, computer graphics and machine learning.

  Cluster Analysis (https://en.wikipedia.org/wiki/Cluster_analysis)

  Cluster Analysis2 (https://realpython.com/k-means-clustering-python/)

The Elbow Method

- "Elbow Method" can be used to find the optimum number of clusters in cluster analysis. The elbow method is used to determine the optimal number of clusters in k-means clustering. The elbow method plots the value of the cost function produced by different values of k. If k increases, average distortion will decrease, each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids. However, the improvements in average distortion will decline as k increases. The value of k at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters.

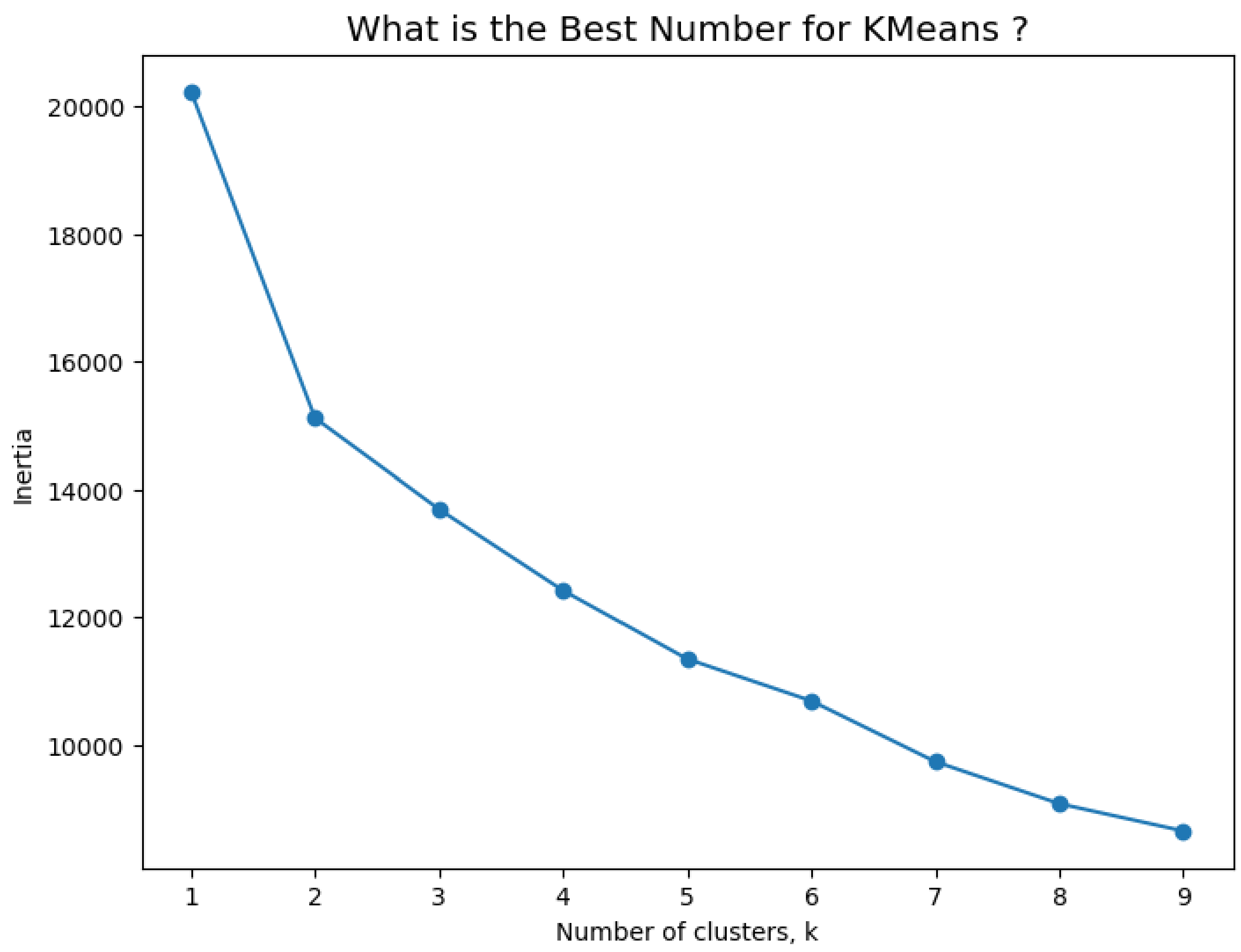
  [The Elbow Method](https://en.wikipedia.org/wiki/Elbow_method_(clustering (https://en.wikipedia.org/wiki/Elbow_method_(clustering))

  The Elbow Method2 (https://medium.com/@mudgalvivek2911/machine-learning-clustering-elbow-method-4e8c2b404a5d)

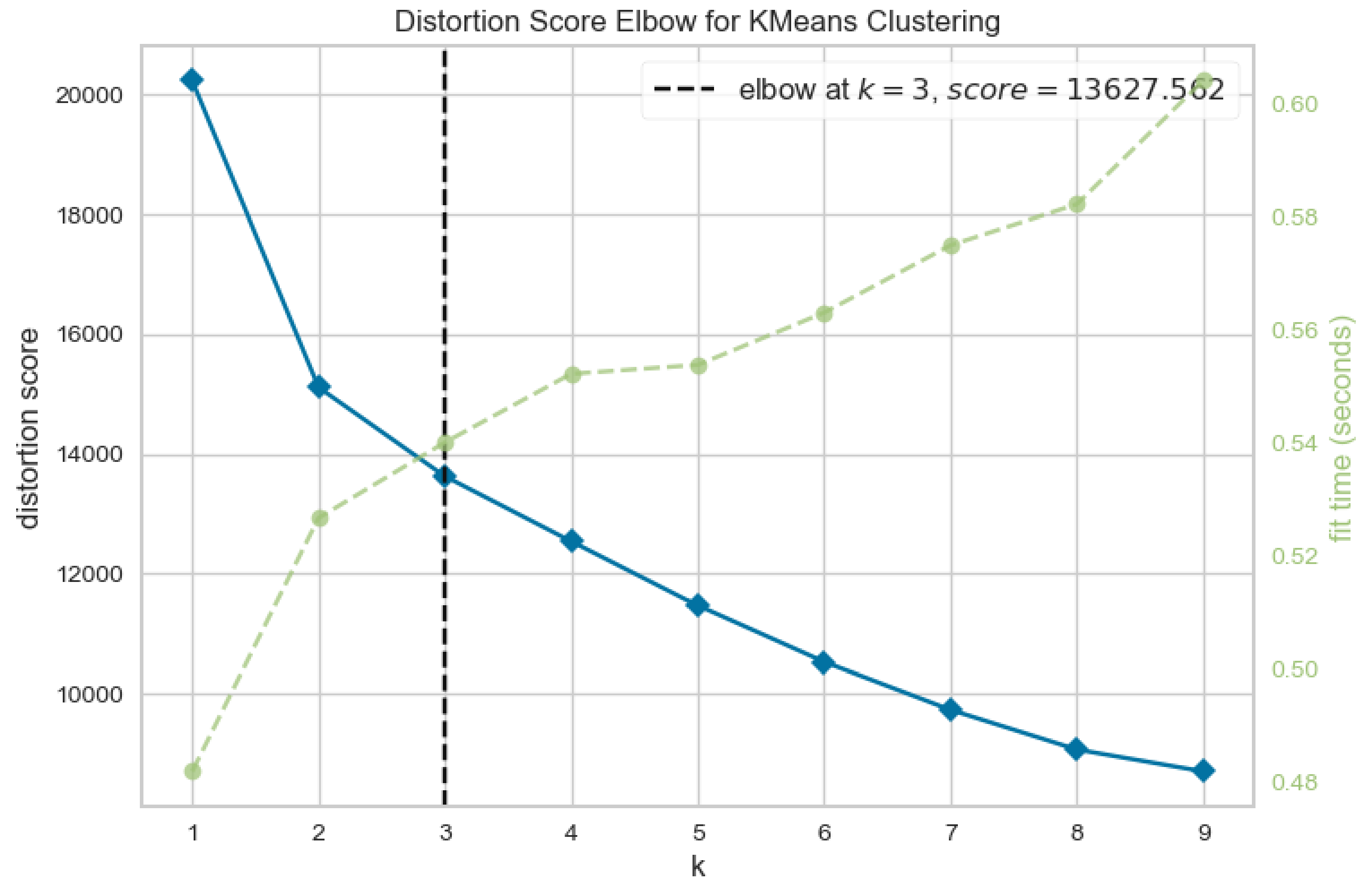
  KMeans (https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1)

Let's find out the groups of employees who left. You can observe that the most important factor for any employee to stay or leave is satisfaction and performance in the company. So let's bunch them in the group of people using cluster analysis.

In [79]:
```
#First : Get the Best KMeans
ks = range(1,10)
inertias=[]
for k in ks :
    # Create a KMeans clusters
    kc = KMeans(n_clusters=k,random_state=1)
    kc.fit(df1_scaled)
    inertias.append(kc.inertia_)

# Plot ks vs inertias
f, ax = plt.subplots(figsize=(8, 6))
plt.plot(ks, inertias, '-o')
plt.xlabel('Number of clusters, k')
plt.ylabel('Inertia')
plt.xticks(ks)
plt.style.use('ggplot')
plt.title('What is the Best Number for KMeans ?')
plt.show()
```



In [80]:
```
from yellowbrick.cluster import KElbowVisualizer
kmeans = KMeans()
visu = KElbowVisualizer(kmeans, k = (1,10))
visu.fit(df1_scaled)
visu.show();
```

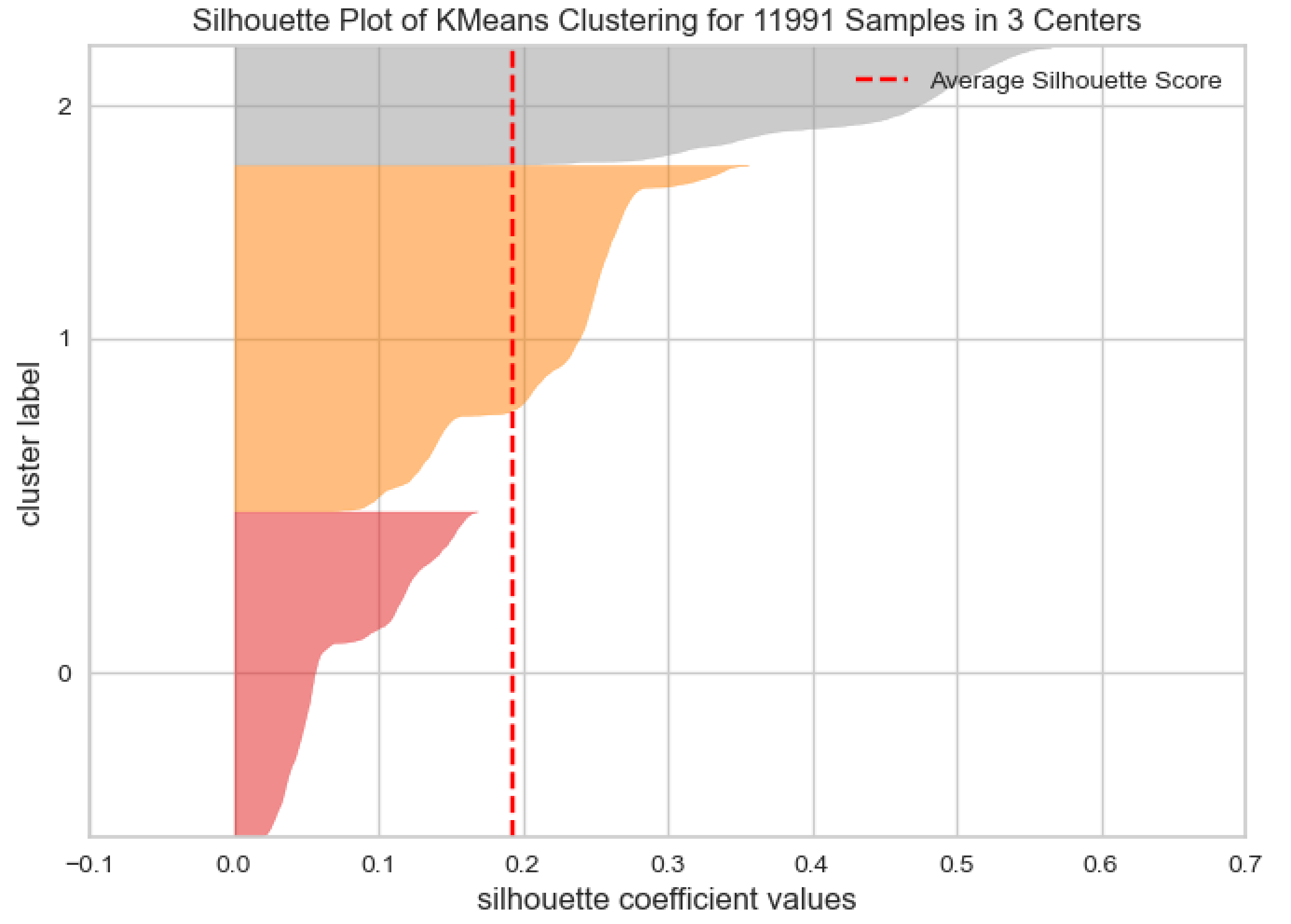


In [81]:
```
cprint("Silhouette Scores",'green', 'on_red')

ssd =[]

K = range(2,10)

for k in K:
    model = KMeans(n_clusters=k)
    model.fit(df1_scaled)
    ssd.append(model.inertia_)
    print(f'Silhouette Score for {k} clusters: {silhouette_score(df1_scaled, model.labels_)}')
```

```
Silhouette Scores
Silhouette Score for 2 clusters: 0.24293849820807237
Silhouette Score for 3 clusters: 0.19250986825854602
Silhouette Score for 4 clusters: 0.18041466053129487
Silhouette Score for 5 clusters: 0.21174582903978412
Silhouette Score for 6 clusters: 0.2405335474200126
Silhouette Score for 7 clusters: 0.2686037509709165
Silhouette Score for 8 clusters: 0.29285588884780916
Silhouette Score for 9 clusters: 0.3069720256203487
```

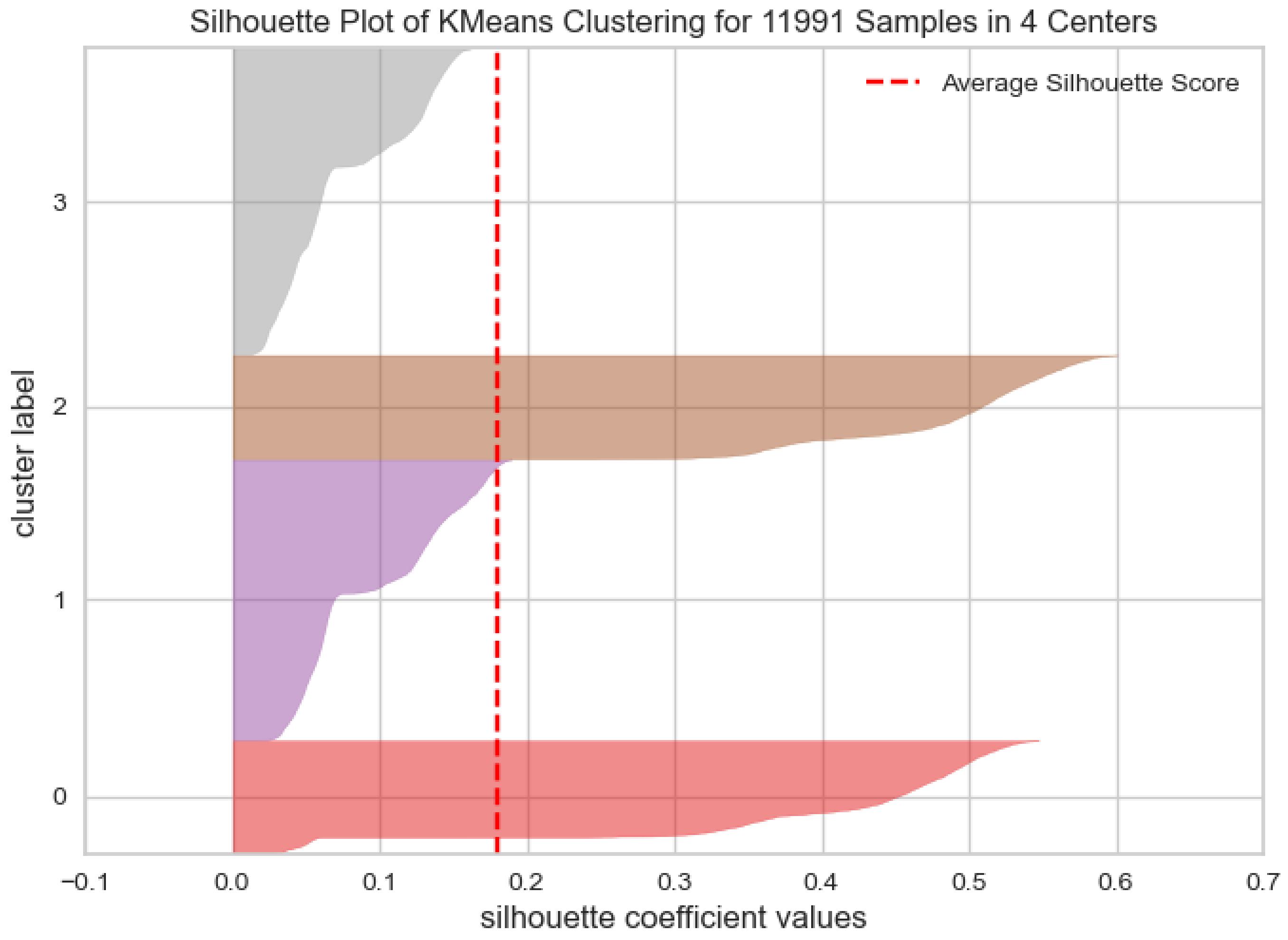
In [83]:
```
cprint("Silhouette Plot for K=3",'green', 'on_red')

from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer

model_3 = KMeans(n_clusters = 3, random_state = 101)
visualizer = SilhouetteVisualizer(model_3)
visualizer.fit(df1_scaled)
visualizer.poof();
```

```
Silhouette Plot for K=3
```



In [84]:
```
cprint("Silhouette Plot for K=4",'green', 'on_red')

from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer

model_4 = KMeans(n_clusters = 4, random_state = 101)
visualizer = SilhouetteVisualizer(model_4)
visualizer.fit(df1_scaled)
visualizer.poof();
```

```
Silhouette Plot for K=4
```



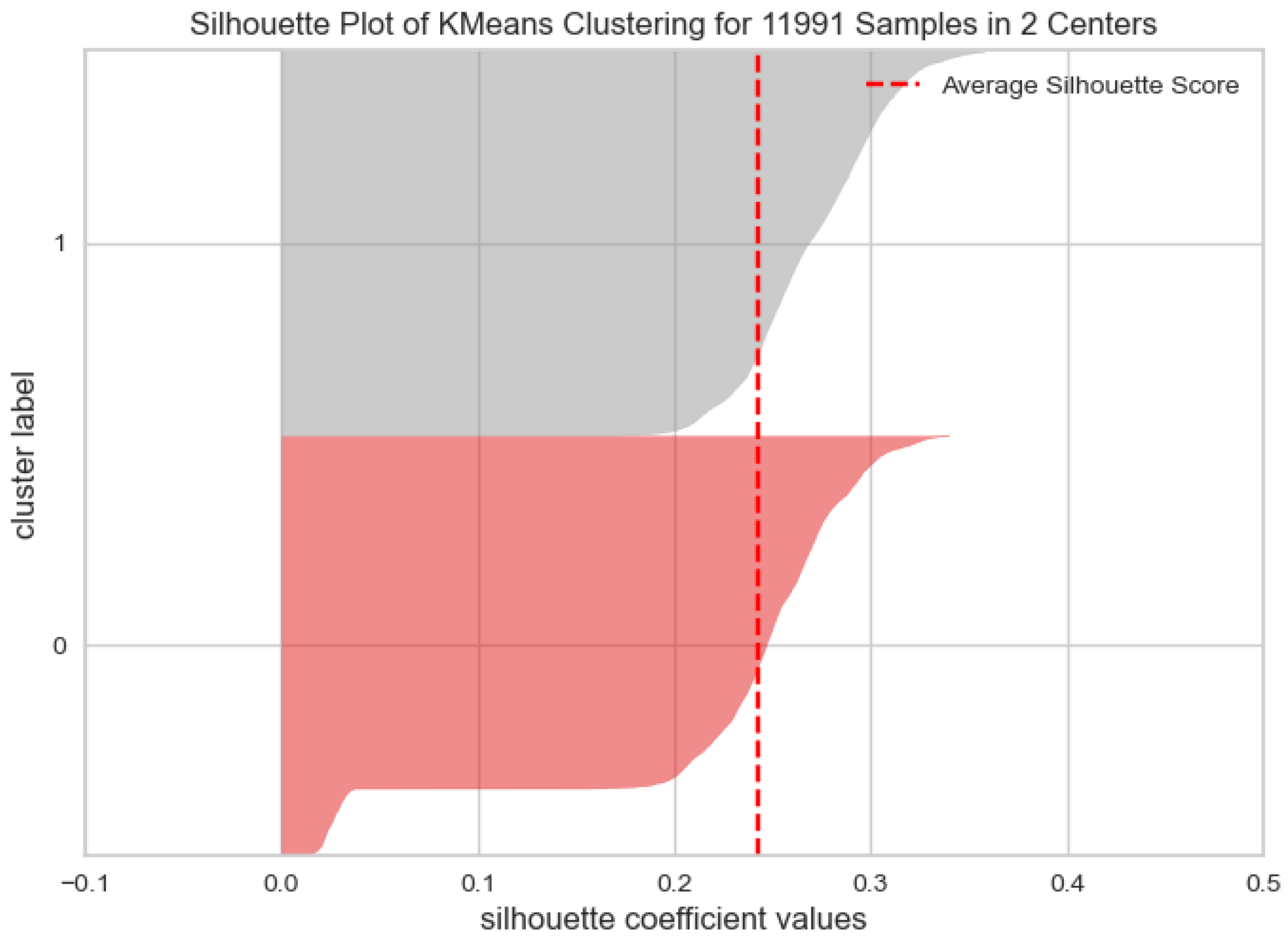
According to the silhouette score, clustering according to the K=2, K=3 and K=4 are seen above.

- For K=3, 0 labelled cluster is below the average silhouette score.
- For K=4, 0 and 3 labelled clusters are below the average silhouette score.
- For K=3 (According to Elbow) and for K=4 (According to the silhouette score) clustering is not suitable for our dataset.

Let's see How it is when K=2 (According to our target variable classes)

In [85]:
```
cprint("Silhouette Plot for K=2",'green', 'on_red')

from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer

model_2 = KMeans(n_clusters = 2, random_state = 101)
visualizer = SilhouetteVisualizer(model_2)
visualizer.fit(df1_scaled)
visualizer.poof();
```

```
Silhouette Plot for K=2
```



In [86]:
```
cprint("KMeans Clustering with K=2",'green', 'on_red')

k_means_model2 = KMeans(n_clusters = 2, random_state = 101)
k_means_model2.fit_predict(df1_scaled)
labels = k_means_model2.labels_
labels
```

```
KMeans Clustering with K=2
```

Out[86]:
```
array([1, 0, 0, ..., 0, 0, 1])
```

In [87]:
```
cprint("Predicted clusters on our dataframe",'green', 'on_red')

df['predicted_clusters'] = labels
df
```

```
Predicted clusters on our dataframe
```

Out[87]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	department	salary	left	predicted_
0	0.380	0.530	2	157	3	0	0	sales	low	1	
1	0.800	0.860	5	262	6	0	0	sales	medium	1	
2	0.110	0.880	7	272	4	0	0	sales	medium	1	
3	0.720	0.870	5	223	5	0	0	sales	low	1	
4	0.370	0.520	2	159	3	0	0	sales	low	1	
...	...	...	...	...	...	...	...	...	...	...	
11995	0.900	0.550	3	259	10	1	1	management	high	0	
11996	0.740	0.950	5	266	10	0	1	management	high	0	
11997	0.850	0.540	3	185	10	0	1	management	high	0	
11998	0.330	0.650	3	172	10	0	1	marketing	high	0	
11999	0.500	0.730	4	180	3	0	0	IT	low	0	

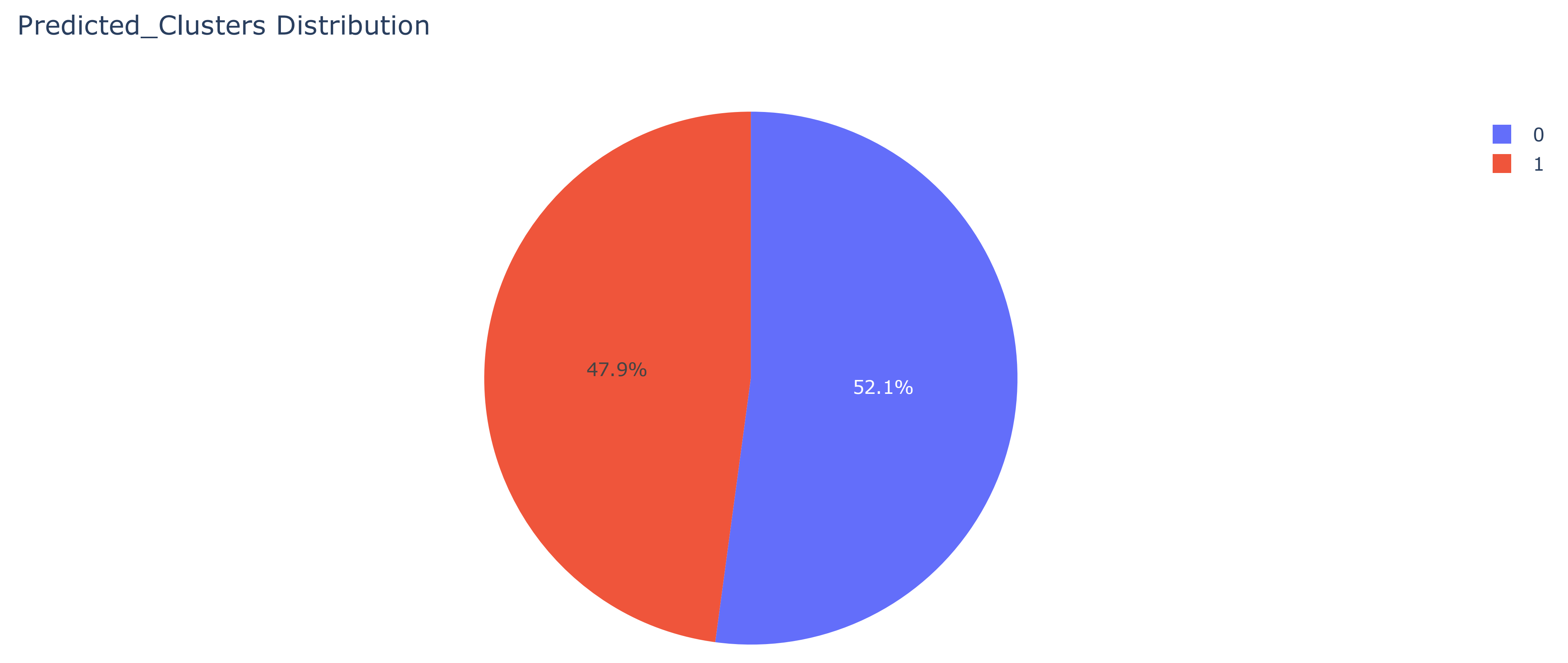
11991 rows × 11 columns

In [88]:
```
cprint('"predicted_clusters" value counts','green', 'on_red')
df['predicted_clusters'].value_counts()
```

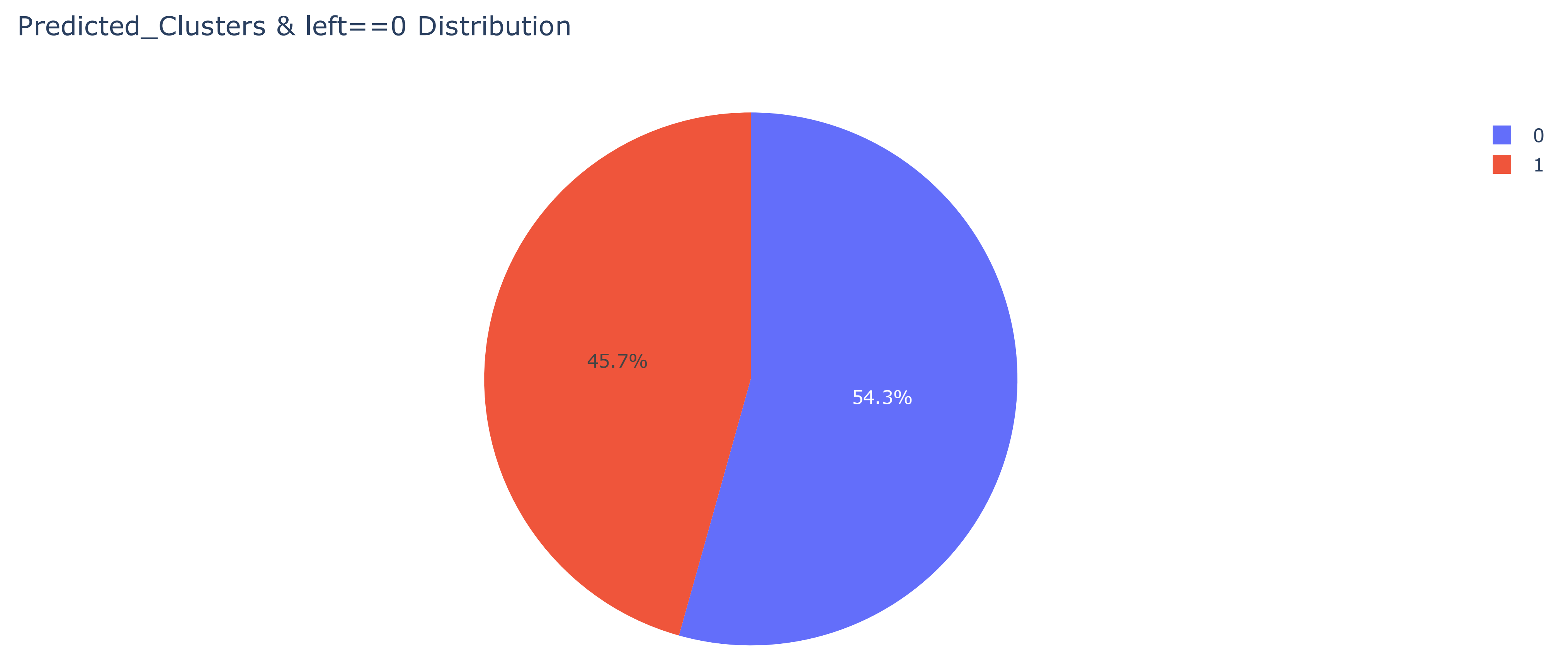
```
"predicted_clusters" value counts
```

Out[88]:
```
0    6251
1    5740
Name: predicted_clusters, dtype: int64
```

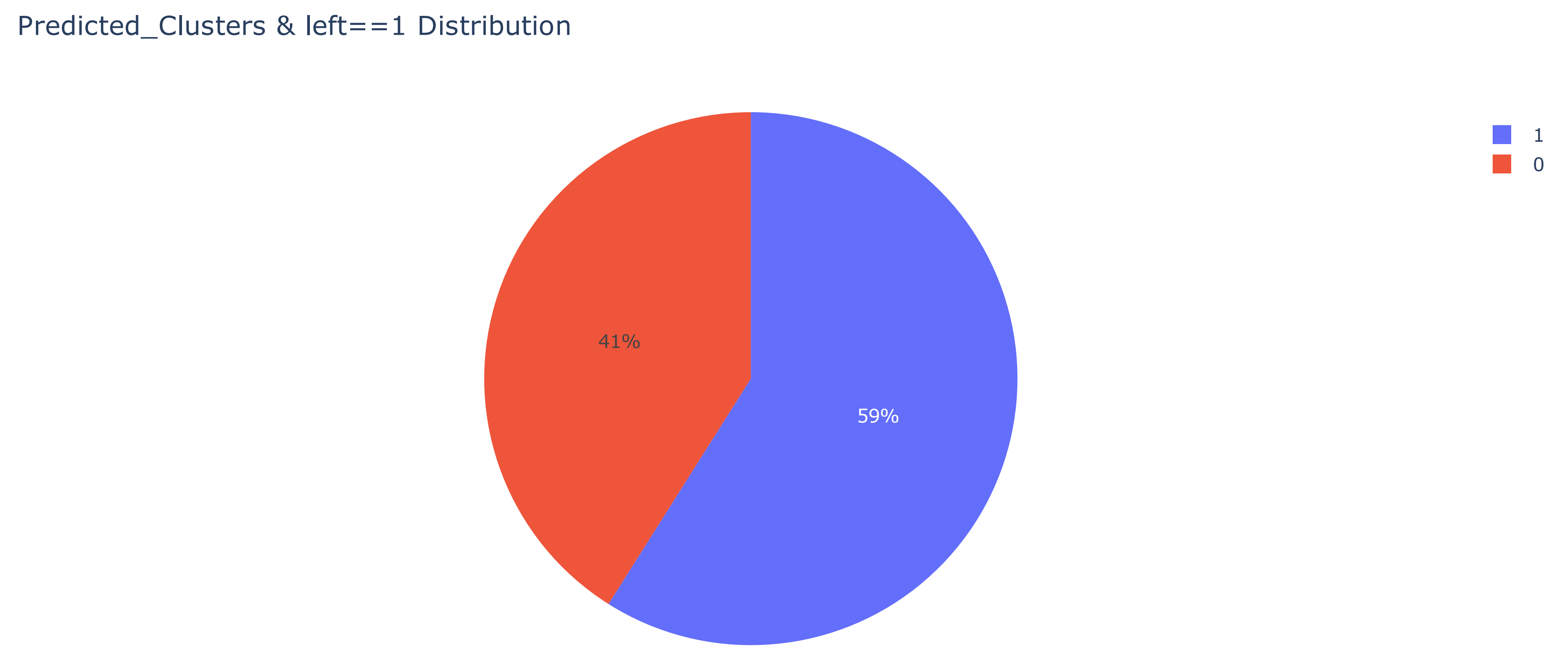
In [89]:
```
fig = px.pie(df, values = df['predicted_clusters'].value_counts(),
            names = (df['predicted_clusters'].value_counts()).index,
            title = 'Predicted_Clusters Distribution')
fig.show()
```



In [90]:
```
fig = px.pie(df, values = df[df['left']==0]['predicted_clusters'].value_counts(),
            names = df[df['left']==0]['predicted_clusters'].value_counts().index,
            title = 'Predicted_Clusters & left==0 Distribution')
fig.show()
```



In [91]:
```
fig = px.pie(df, values = df[df['left']==1]['predicted_clusters'].value_counts(),
            names = df[df['left']==1]['predicted_clusters'].value_counts().index,
            title = 'Predicted_Clusters & left==1 Distribution')
fig.show()
```



In [92]:
```
cprint('Mean values according to the left','green', 'on_red')
df.groupby('left').mean()
```

Mean values according to the left

Out[92]:

left	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	predicted_clusters
0	0.667	0.716	3.787	198.943	3.262	0.174	0.019	0.457
1	0.440	0.722	3.883	208.162	3.881	0.053	0.004	0.590

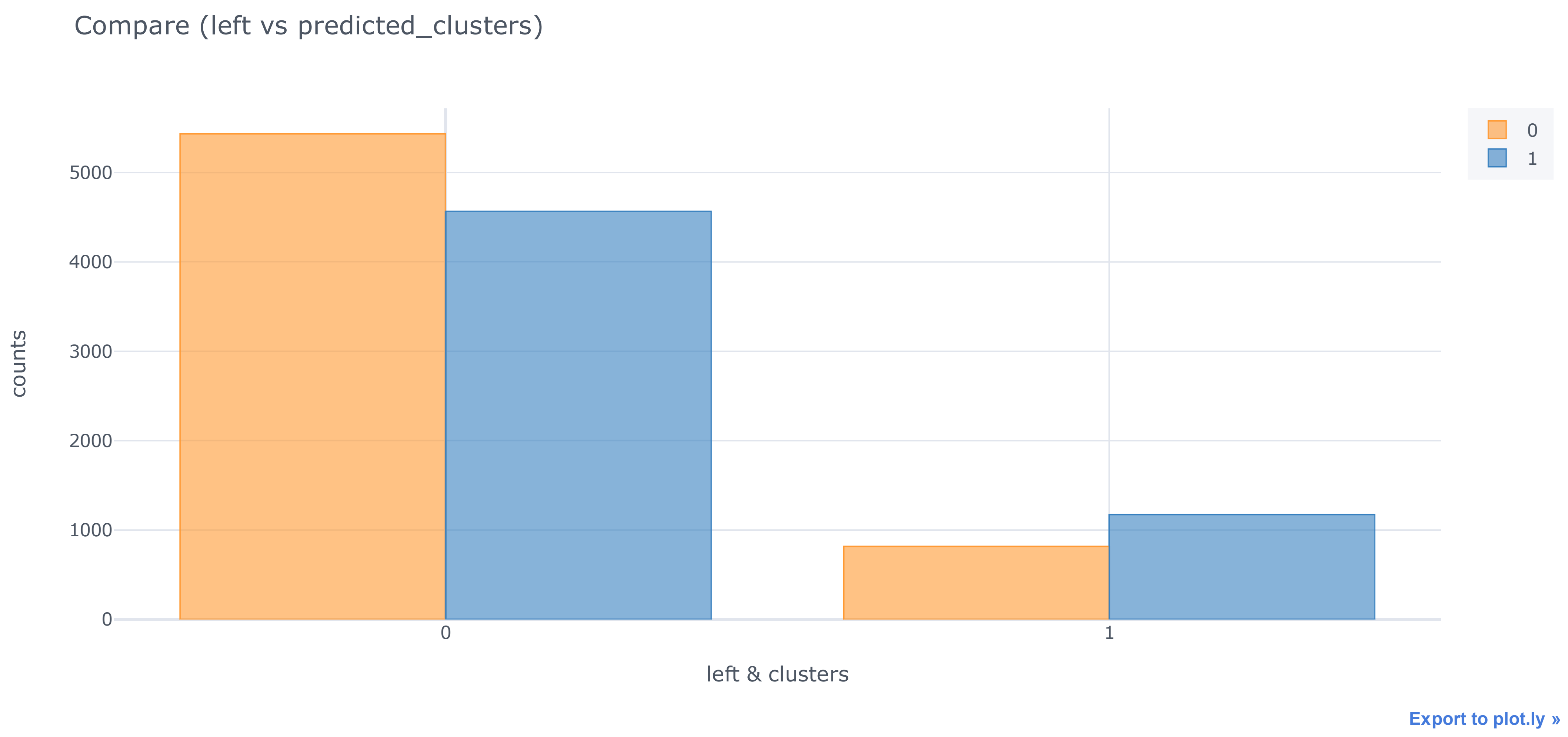
In [93]:
```
cprint('Mean values according to the left and predicted clusters','green', 'on_red')
df.groupby(['left', 'predicted_clusters']).mean()
```

Mean values according to the left and predicted clusters

Out[93]:

left	predicted_clusters	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years
0	0	0.666	0.714	3.788	199.173	3.334	0.168	0.030
	1	0.669	0.718	3.786	198.669	3.176	0.182	0.007
1	0	0.440	0.722	3.911	209.366	3.886	0.059	0.004
	1	0.441	0.722	3.865	207.325	3.878	0.049	0.004

In [94]:
```
pd.crosstab(df['left'], df['predicted_clusters']).iplot(kind="bar", title = 'Compare (left vs predicted_clusters)',
            xTitle = 'left & clusters', yTitle = 'counts')
```



In [95]:
```
cprint('Mean values according to the predicted clusters','green', 'on_red')
df.groupby('predicted_clusters').mean()
```

Mean values according to the predicted clusters

Out[95]:

predicted_clusters	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	left
0	0.636	0.715	3.804	200.505	3.406	0.154	0.026	0.131
1	0.623	0.719	3.802	200.439	3.320	0.155	0.007	0.205

In [96]:
```
cprint('Mean values according to the predicted clusters and left','green', 'on_red')
df.groupby(['predicted_clusters', 'left']).mean()
```

Mean values according to the predicted clusters and left

Out[96]:

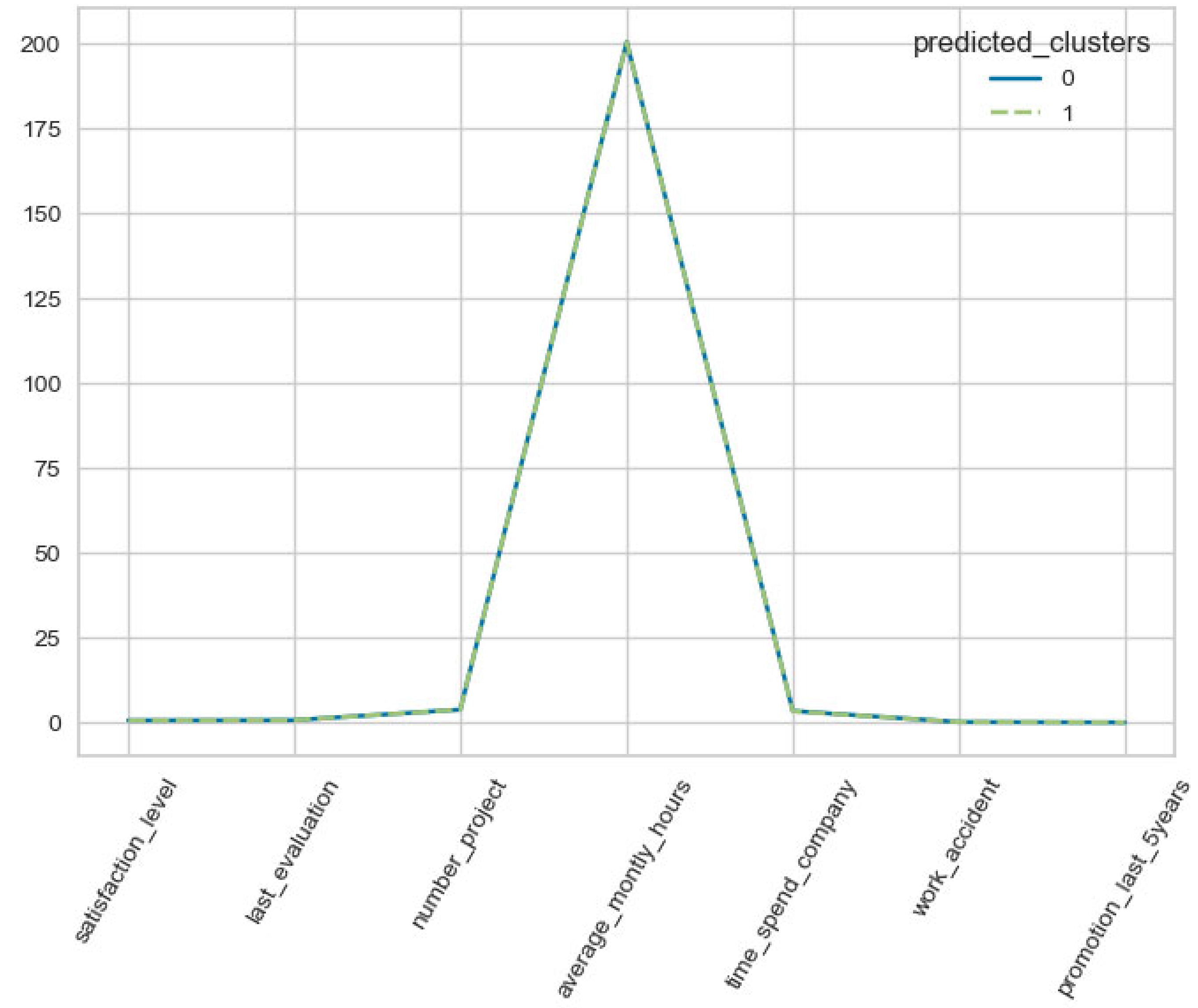
predicted_clusters	left	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years
0	0	0.666	0.714	3.788	199.173	3.334	0.168	0.030
	1	0.440	0.722	3.911	209.366	3.886	0.059	0.004
1	0	0.669	0.718	3.786	198.669	3.176	0.182	0.007
	1	0.441	0.722	3.865	207.325	3.878	0.049	0.004

In [97]:
```
pd.crosstab(df['predicted_clusters'],
            df['left']).iplot(kind="bar", title = 'Compare (predicted_clusters vs left)',
            xTitle = 'clusters & left', yTitle = 'counts')
```

In [98]:
```
cprint('Mean values of the features according to the predicted clusters','green', 'on_red')

sns.lineplot(data = df.iloc[:, [0, 1, 2, 3, 4, 5, 6, 7, 8, 10]].groupby("predicted_clusters").mean().T)
plt.xticks(rotation = 60)
```

Mean values of the features according to the predicted clusters

Out[98]:
```
([0, 1, 2, 3, 4, 5, 6],
 [Text(0, 0, 'satisfaction_level'),
  Text(1, 0, 'last_evaluation'),
  Text(2, 0, 'number_project'),
  Text(3, 0, 'average_montly_hours'),
  Text(4, 0, 'time_spend_company'),
  Text(5, 0, 'work_accident'),
  Text(6, 0, 'promotion_last_5years')])
```



As seen above, the columns in the data set do not separate from each other. All columns are intertwined with each other.

As seen above it is visually obvious that **clustering is not a good approach to our dataset**.

From now on we are going to use **classification models to make churn predictions**.

8 - MODEL BUILDING

- Split Data as Train and Test set
- Built Gradient Boosting Classifier, Evaluate Model Performance and Predict Test Data
- Built K Neighbors Classifier and Evaluate Model Performance and Predict Test Data

- Built Random Forest Classifier and Evaluate Model Performance and Predict Test Data

Evaluating Model Performance

- Confusion Matrix : You can use scikit-learn metrics module for accuracy calculation. A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

  Confusion Matrix (https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/)

- Yellowbrick: Yellowbrick is a suite of visualization and diagnostic tools that will enable quicker model selection. It's a Python package that combines scikit-learn and matplotlib. Some of the more popular visualization tools include model selection, feature visualization, classification and regression visualization

  Yellowbrick (https://www.analyticsvidhya.com/blog/2018/05/yellowbrick-a-set-of-visualization-tools-to-accelerate-your-model-selection-process/)

Here, Dataset is broken into two parts in ratio of 70:30. It means 70% data will used for model training and 30% for model testing.

In [99]:
```
cprint('New df for Classification','green', 'on_red')
df2 = df.drop('predicted_clusters', axis = 1)
df2.head(1)
```

```
New df for Classification
```

Out[99]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	department	salary	left
0	0.380	0.530	2	157	3	0	0	sales	low	1

In [100…
```
cprint('New df after getting dummied','green', 'on_red')
df2 = pd.get_dummies(df2, columns = ['department','salary'], drop_first = True)
df2.head(1)
```

```
New df after getting dummied
```

Out[100]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	work_accident	promotion_last_5years	left	department_RandD	department_acco
0	0.380	0.530	2	157	3	0	0	1	0	

8.1 - Spliting Data as Train & Test

In [101…
```
X = df2.drop('left', axis = 1)
y = df2['left']
```

In [102…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.3, random_state = 101)
```

In [103…
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [104…
```
def eval(model, X_train, X_test):
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)

    print(confusion_matrix(y_test, y_pred))
    print("Test_Set")
    print(classification_report(y_test,y_pred))
    print("Train_Set")
    print(classification_report(y_train,y_pred_train))
```

In [105…
```
def train_val(y_train, y_train_pred, y_test, y_pred):

    scores = {"train_set": {"Accuracy" : accuracy_score(y_train, y_train_pred),
                            "Precision" : precision_score(y_train, y_train_pred),
                            "Recall" : recall_score(y_train, y_train_pred),
                            "f1" : f1_score(y_train, y_train_pred)},

              "test_set": {"Accuracy" : accuracy_score(y_test, y_pred),
                           "Precision" : precision_score(y_test, y_pred),
                           "Recall" : recall_score(y_test, y_pred),
                           "f1" : f1_score(y_test, y_pred)}}

    return pd.DataFrame(scores)
```

8.2 - Gradient Boosting Classifier

8.2.1 Model Building

In [106…
```
GB_model = GradientBoostingClassifier(random_state = 101)
GB_model.fit(X_train, y_train)
y_pred = GB_model.predict(X_test)
y_train_pred = GB_model.predict(X_train)

GB_model_f1 = f1_score(y_test, y_pred)
GB_model_acc = accuracy_score(y_test, y_pred)
GB_model_recall = recall_score(y_test, y_pred)
GB_model_auc = roc_auc_score(y_test, y_pred)
```

8.2.2 Evaluating Model Performance

In [107…
```
print("GB_Model")
print ("------------------")
eval(GB_model, X_train, X_test)
```

```
GB_Model
------------------
[[2974   27]
 [  45  552]]
Test_Set
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3001
           1       0.95      0.92      0.94       597

    accuracy                           0.98      3598
   macro avg       0.97      0.96      0.96      3598
weighted avg       0.98      0.98      0.98      3598

Train_Set
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6999
           1       0.97      0.93      0.95      1394

    accuracy                           0.98      8393
   macro avg       0.98      0.96      0.97      8393
weighted avg       0.98      0.98      0.98      8393
```

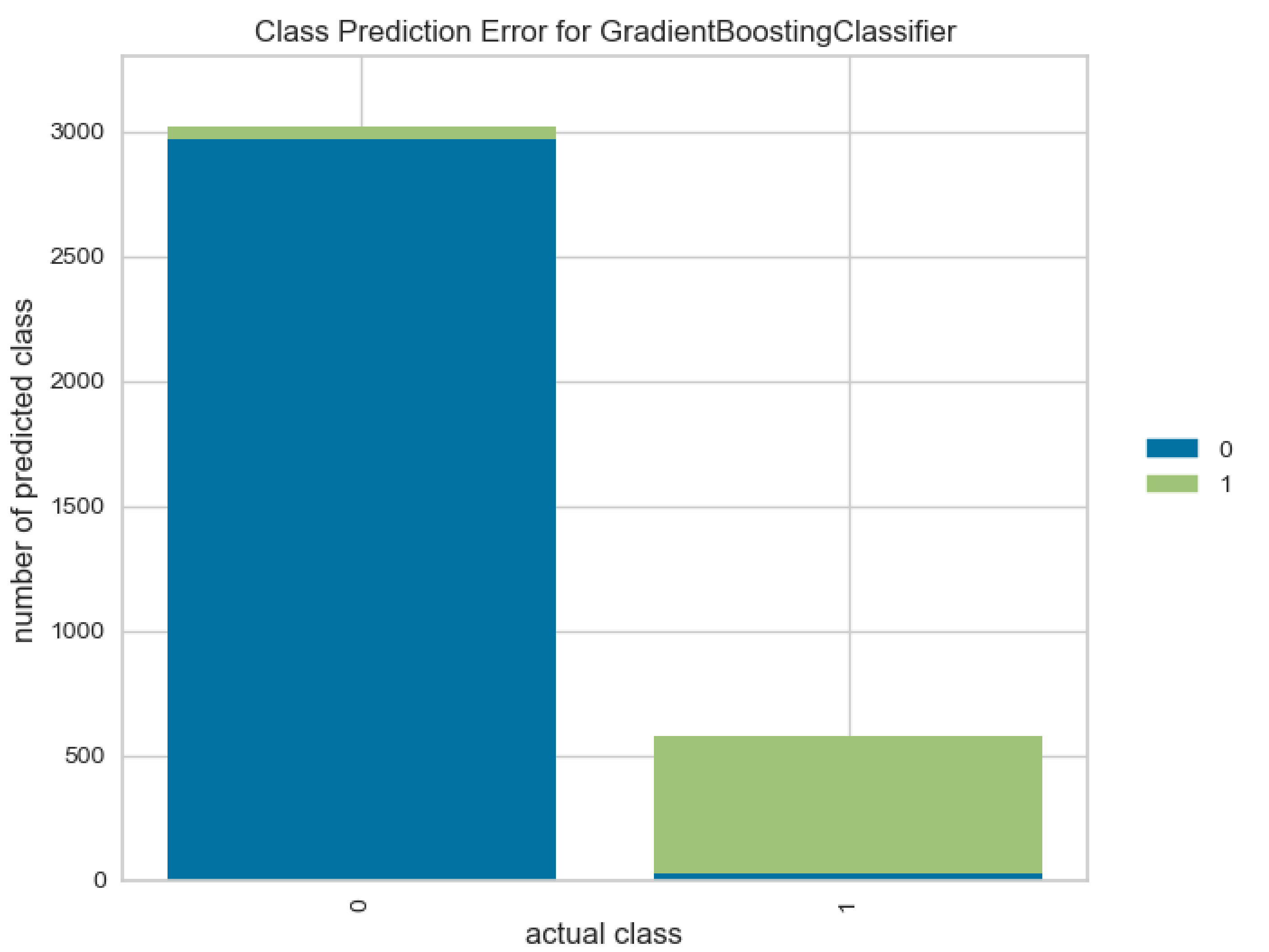
In [108…
```
cprint('GB_model Scores','green', 'on_red')
train_val(y_train, y_train_pred, y_test, y_pred)
```

GB_model Scores

Out[108]:

	train_set	test_set
Accuracy	0.983	0.980
Precision	0.969	0.953
Recall	0.928	0.925
f1	0.948	0.939

In [109…
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(GB_model)
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
# Draw visualization
visualizer.poof();
```



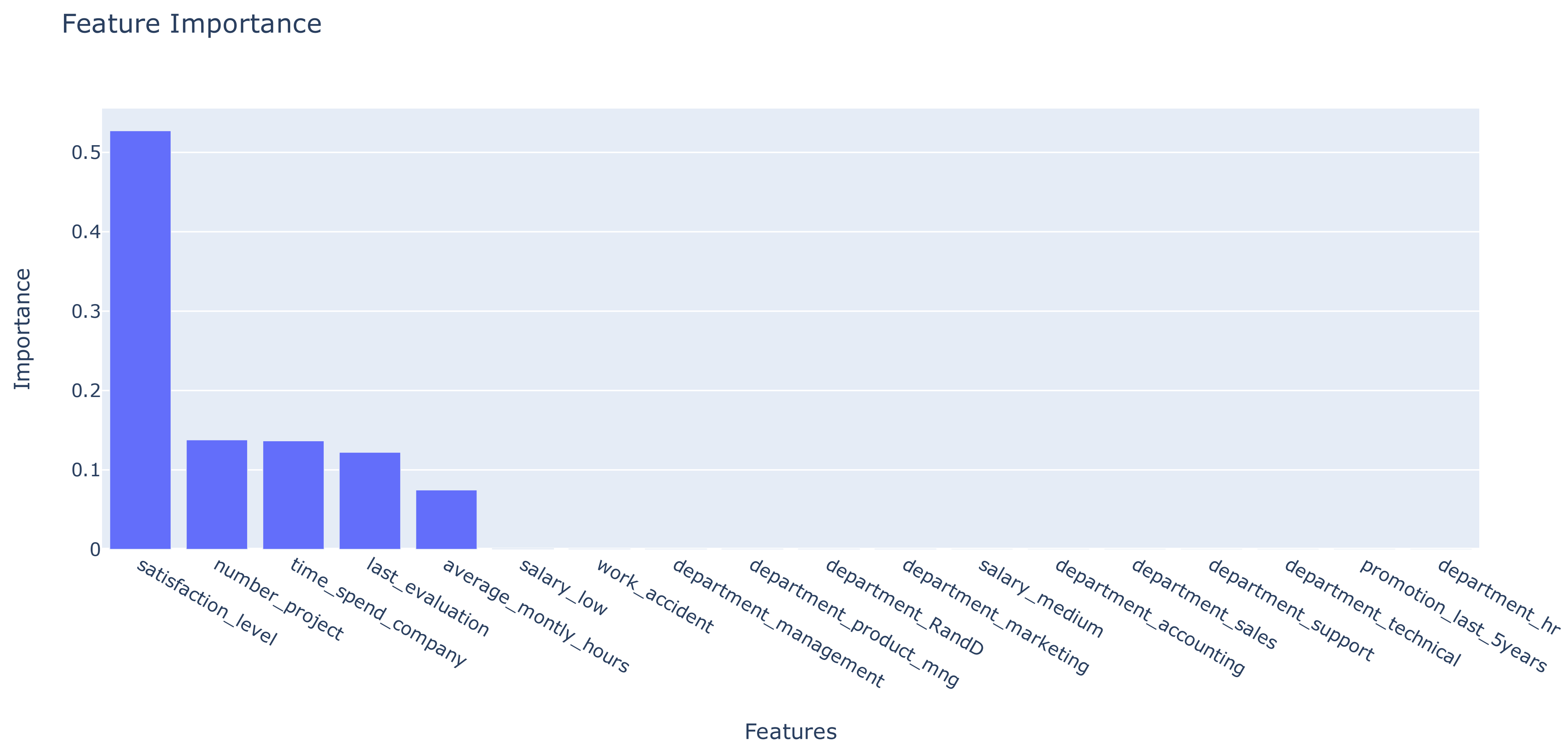
8.2.3 Feature Importance for Gradient Boosting Model

In [110…
```
GB_feature_imp = pd.DataFrame(index=X.columns, data = GB_model.feature_importances_, columns = ['Importance']).sort_values("Importance", ascending =
GB_feature_imp
```

Out[110]:

	Importance
satisfaction_level	0.527
number_project	0.138
time_spend_company	0.137
last_evaluation	0.122
average_montly_hours	0.075
salary_low	0.001
work_accident	0.000
department_management	0.000
department_product_mng	0.000
department_RandD	0.000
department_marketing	0.000
salary_medium	0.000
department_accounting	0.000
department_sales	0.000
department_support	0.000
department_technical	0.000
promotion_last_5years	0.000
department_hr	0.000

In [111…
```
fig = px.bar(GB_feature_imp.sort_values('Importance', ascending = False), x = GB_feature_imp.sort_values('Importance',
             ascending = False).index, y = 'Importance', title = "Feature Importance",
             labels = dict(x = "Features", y ="Feature_Importance"))
fig.show()
```



8.2.4 Gradient Boosting Classifier Cross Validation

In [112…
```
GB_cv = GradientBoostingClassifier(random_state = 101)
GB_cv_scores = cross_validate(GB_cv, X_train, y_train,
                              scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
GB_cv_scores = pd.DataFrame(GB_cv_scores, index = range(1, 11))
GB_cv_scores.mean()[2:]
```

Out[112]:
```
test_accuracy    0.981
test_precision   0.965
test_recall      0.922
test_f1          0.943
test_roc_auc     0.985
dtype: float64
```

8.2.5 Gradient Boosting Classifier GridSearchCV

In [113…
```
param_grid = {"n_estimators":[100, 200, 300],
              "subsample":[0.5, 1],
              "max_features" : [None, 2, 3, 4],
              "learning_rate": [0.001, 0.01, 0.1],
              'max_depth':[3, 4, 5, 6]}
```

In [114…
```
GB_grid = GradientBoostingClassifier(random_state = 101)
GB_grid_model = GridSearchCV(GB_grid, param_grid, scoring = "f1", verbose = 2, n_jobs = -1).fit(X_train, y_train)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
```

In [115…
```
GB_grid_model.best_estimator_
```

Out[115]:
```
▾                    GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.01, max_depth=6, n_estimators=200,
                           random_state=101, subsample=0.5)
```

In [116…
```
print(colored('\033[1mBest Parameters of GridSearchCV for Gradient Boosting Model:\033[0m', 'blue'), colored(GB_grid_model.best_params_, 'red'))
```

Best Parameters of GridSearchCV for Gradient Boosting Model: {'learning_rate': 0.01, 'max_depth': 6, 'max_features': None, 'n_estimators': 200, 'subsample': 0.5}

In [117…
```
GB_tuned = GradientBoostingClassifier(learning_rate = 0.01,
                                      max_depth = 6,
                                      n_estimators = 200,
                                      subsample = 0.5,
                                      random_state = 101).fit(X_train, y_train)
```

In [118…
```
y_pred = GB_tuned.predict(X_test)
y_train_pred = GB_tuned.predict(X_train)

GB_tuned_f1 = f1_score(y_test, y_pred)
GB_tuned_acc = accuracy_score(y_test, y_pred)
GB_tuned_recall = recall_score(y_test, y_pred)
GB_tuned_auc = roc_auc_score(y_test, y_pred)
```

In [119…

```
print("GB_tuned")
print ("------------------")
eval(GB_tuned, X_train, X_test)
```

```
GB_tuned
------------------
[[2994    7]
 [  48  549]]
Test_Set
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3001
           1       0.99      0.92      0.95       597

    accuracy                           0.98      3598
   macro avg       0.99      0.96      0.97      3598
weighted avg       0.98      0.98      0.98      3598

Train_Set
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      6999
           1       0.99      0.93      0.96      1394

    accuracy                           0.99      8393
   macro avg       0.99      0.96      0.97      8393
weighted avg       0.99      0.99      0.99      8393
```

In [120…

```
cprint('GB_tuned Scores','green', 'on_red')
train_val(y_train, y_train_pred, y_test, y_pred)
```

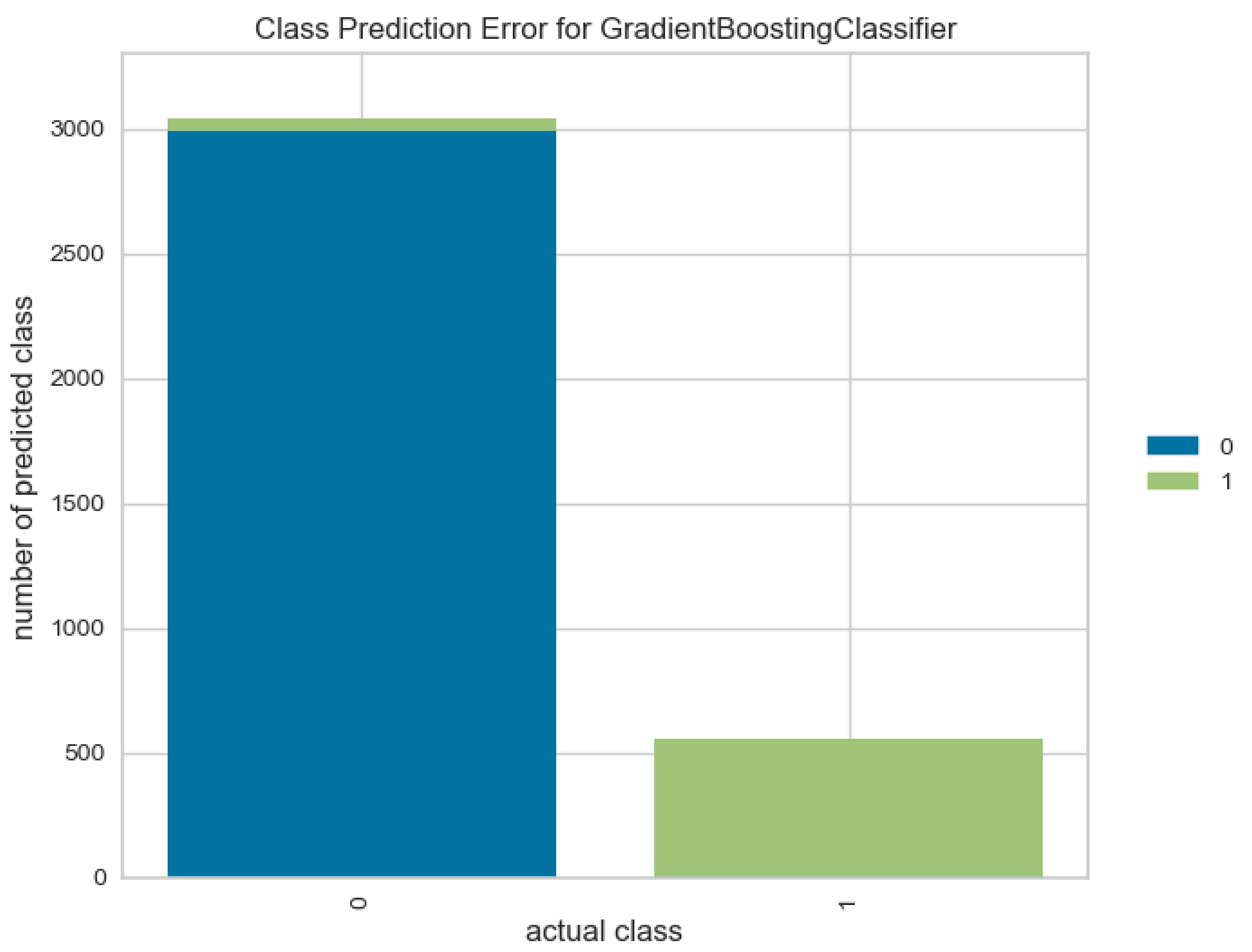
GB_tuned Scores

Out[120]:

	train_set	test_set
Accuracy	0.986	0.985
Precision	0.989	0.987
Recall	0.927	0.920
f1	0.957	0.952

In [121…

```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(GB_tuned)
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
# Draw visualization
visualizer.poof();
```



8.2.7 Prediction

In [122…

```
cprint('GB_tuned Predictions','green', 'on_red')
GB_Pred = {"Actual": y_test, "GB_Pred":y_pred}
GB_Pred = pd.DataFrame.from_dict(GB_Pred)
GB_Pred.head()
```

GB_tuned Predictions

Out[122]:

	Actual	GB_Pred
3118	0	0
10490	0	0
1106	1	1
3822	0	0
6873	0	0

In [123…

```
cprint('Predictions','green', 'on_red')
Model_Preds = GB_Pred
Model_Preds.head()
```

Predictions

Out[123]:

	Actual	GB_Pred
3118	0	0
10490	0	0
1106	1	1
3822	0	0
6873	0	0

8.3 - KNeighbors Classifier

8.3.1 Model Building

In [124…
```
KNN_model = KNeighborsClassifier(n_neighbors = 5)
KNN_model.fit(X_train, y_train)
y_pred = KNN_model.predict(X_test)
y_train_pred = KNN_model.predict(X_train)

KNN_model_f1 = f1_score(y_test, y_pred)
KNN_model_acc = accuracy_score(y_test, y_pred)
KNN_model_recall = recall_score(y_test, y_pred)
KNN_model_auc = roc_auc_score(y_test, y_pred)
```

8.3.2 Evaluating Model Performance

In [125…
```
print("KNN_Model")
print ("------------------")
eval(KNN_model, X_train, X_test)
```

```
KNN_Model
------------------
[[2883  118]
 [  83  514]]
Test_Set
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      3001
           1       0.81      0.86      0.84       597

    accuracy                           0.94      3598
   macro avg       0.89      0.91      0.90      3598
weighted avg       0.95      0.94      0.94      3598

Train_Set
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      6999
           1       0.88      0.88      0.88      1394

    accuracy                           0.96      8393
   macro avg       0.93      0.93      0.93      8393
weighted avg       0.96      0.96      0.96      8393
```

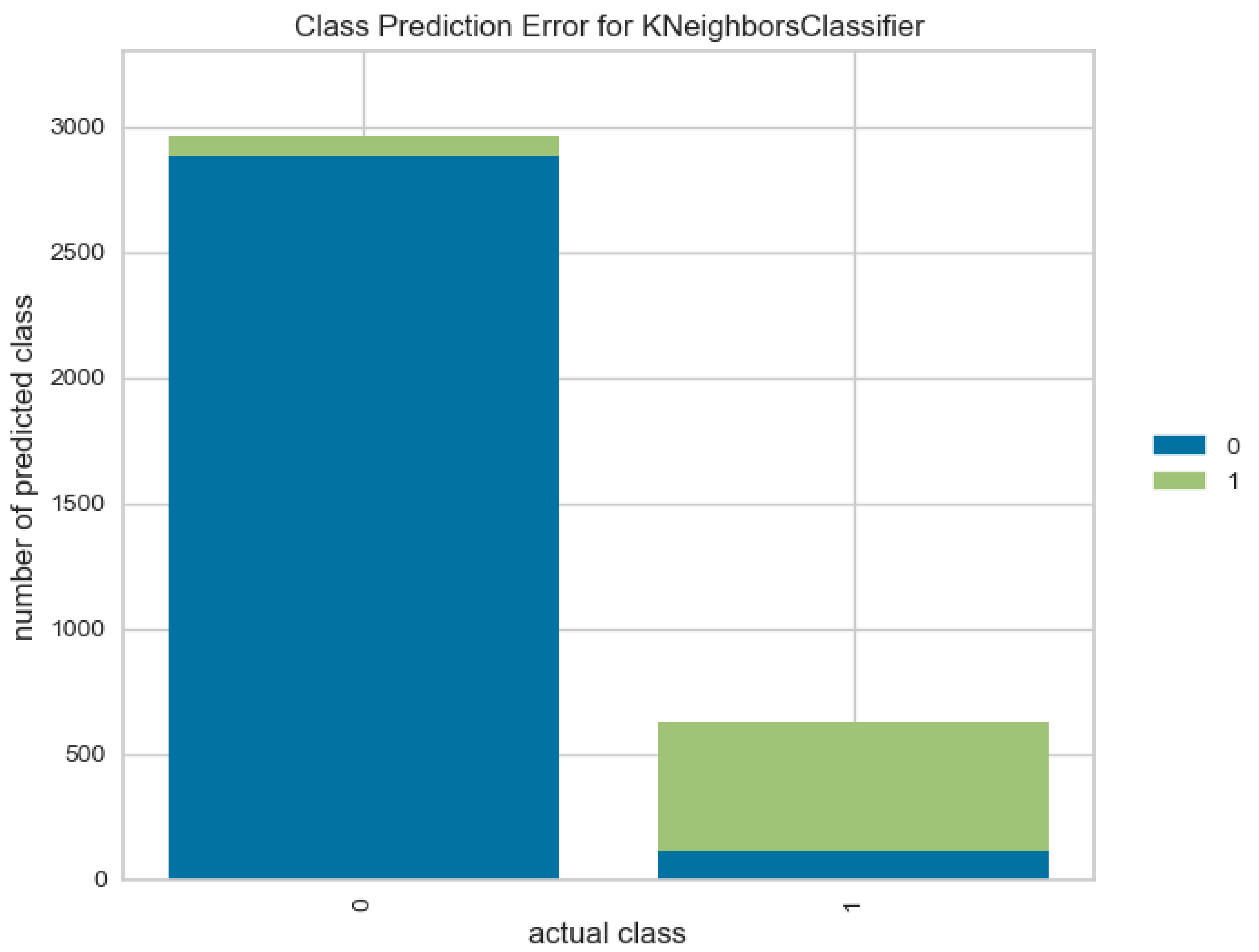
In [126…
```
cprint('KNN_model Scores','green', 'on_red')
train_val(y_train, y_train_pred, y_test, y_pred)
```

KNN_model Scores

Out[126]:

	train_set	test_set
Accuracy	0.960	0.944
Precision	0.879	0.813
Recall	0.878	0.861
f1	0.879	0.836

In [127…
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(KNN_model)
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
# Draw visualization
visualizer.poof();
```



8.3.3 KNeighbors Classifier Cross Validation

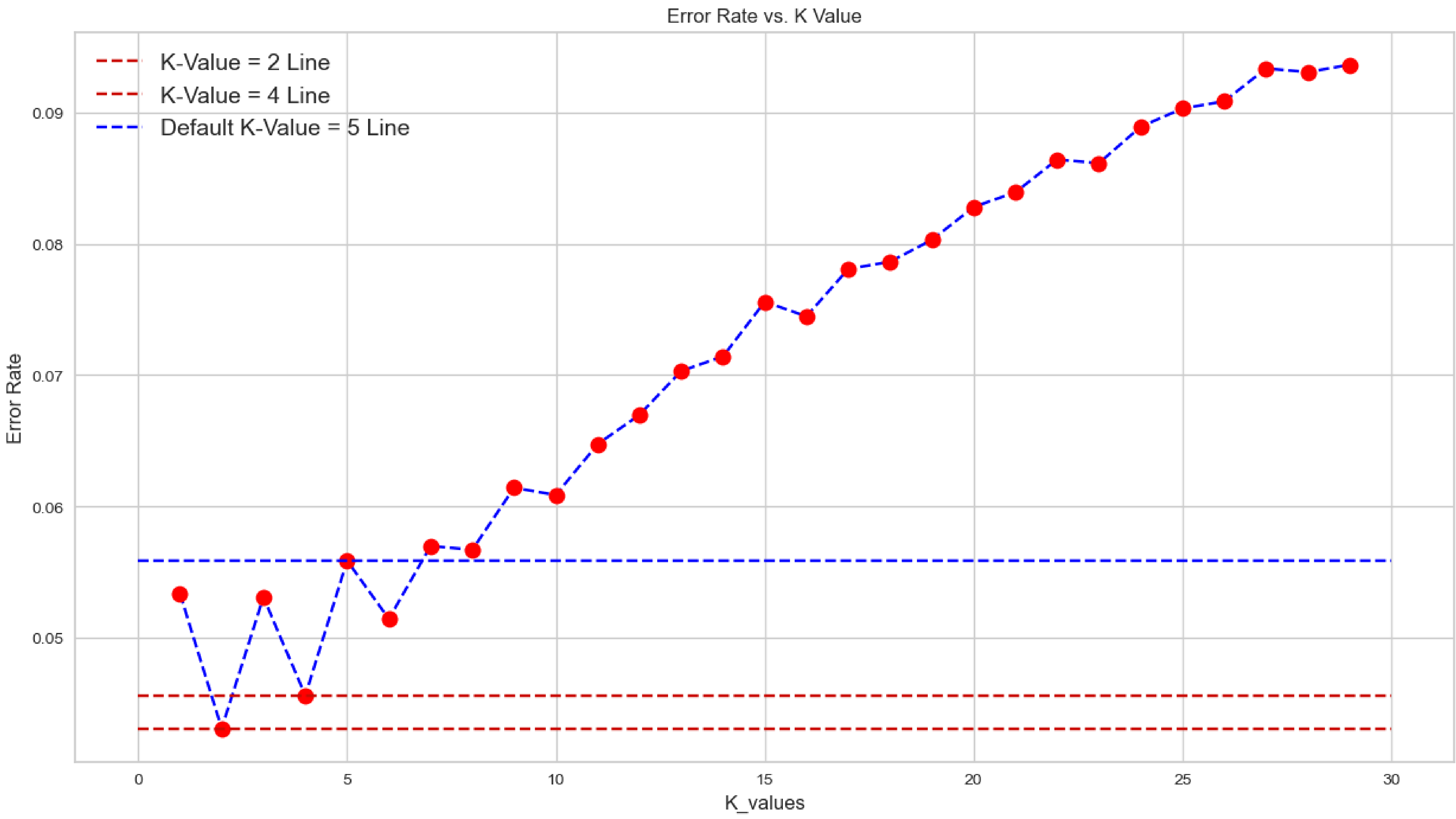
In [128…
```
KNN_cv = KNeighborsClassifier(n_neighbors = 5)
KNN_cv_scores = cross_validate(KNN_cv, X_train, y_train,
                              scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
KNN_cv_scores = pd.DataFrame(KNN_cv_scores, index = range(1, 11))
KNN_cv_scores.mean()[2:]
```

Out[128]:
```
test_accuracy     0.948
test_precision    0.835
test_recall       0.859
test_f1           0.846
test_roc_auc      0.943
dtype: float64
```

8.3.4 Elbow Method for Choosing Reasonable K Values

In [129…
```
test_error_rates = []

for k in range(1, 30):
    KNN = KNeighborsClassifier(n_neighbors = k)
    KNN.fit(X_train, y_train)

    y_pred = KNN.predict(X_test)

    test_error = 1 - accuracy_score(y_test, y_pred)
    test_error_rates.append(test_error)

print(test_error_rates)

plt.figure(figsize = (15, 8))
plt.plot(range(1, 30), test_error_rates, color = 'blue', linestyle = '--', marker = 'o',
         markerfacecolor = 'red', markersize = 10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')

plt.hlines(y = 0.04307948860478039, xmin = 0, xmax = 30, colors = 'r', linestyles = "--", label = "K-Value = 2 Line")
plt.hlines(y = 0.04558087826570312, xmin = 0, xmax = 30, colors = 'r', linestyles = "--", label = "K-Value = 4 Line")
plt.hlines(y = 0.055864369093941, xmin = 0, xmax = 30, colors = 'blue', linestyles = "--", label = "Default K-Value = 5 Line")
plt.legend(prop = {"size":14});
```

```
[0.053362979433018376, 0.04307948860478039, 0.053085047248471406, 0.04558087826570312, 0.0558643690939411, 0.05141745414118959, 0.05697609783212898, 0.05669816564758201, 0.06142301278488049, 0.060867148415786554, 0.06475819899944413, 0.06698165647581988, 0.07031684269038352, 0.0714285714285714, 0.07559755419677594, 0.07448582545858806, 0.07809894385769867, 0.07865480822679272, 0.08032240133407453, 0.08282379099499726, 0.0839355197331851, 0.08643690939410786, 0.0861589772095609, 0.08893829905503059, 0.09032795997776544, 0.09088382434685938, 0.0933852140077821, 0.09310728182323513, 0.09366314619232907]
```



Let's have look to recall values for different K's ranging from 1 to 10

In [130…
```
# FIRST A QUICK COMPARISON TO OUR DEFAULT K=5

knn5 = KNeighborsClassifier(n_neighbors = 5)

knn5.fit(X_train,y_train)
pred = knn5.predict(X_test)

print('WITH K=5')
print('---------------')
print(confusion_matrix(y_test, pred))
print('---------------')
print(classification_report(y_test, pred))
```

```
WITH K=5
---------------
[[2883  118]
 [  83  514]]
---------------
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      3001
           1       0.81      0.86      0.84       597

    accuracy                           0.94      3598
   macro avg       0.89      0.91      0.90      3598
weighted avg       0.95      0.94      0.94      3598
```

In [131…
```
# NOW K=2

knn2 = KNeighborsClassifier(n_neighbors = 2)

knn2.fit(X_train,y_train)
pred = knn2.predict(X_test)

print('WITH K=2')
print('---------------')
print(confusion_matrix(y_test, pred))
print('---------------')
print(classification_report(y_test, pred))
```

```
WITH K=2
---------------
[[2951   50]
 [ 105  492]]
---------------
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      3001
           1       0.91      0.82      0.86       597

    accuracy                           0.96      3598
   macro avg       0.94      0.90      0.92      3598
weighted avg       0.96      0.96      0.96      3598
```

In [132…
```
# NOW K=4

knn4 = KNeighborsClassifier(n_neighbors = 4)

knn4.fit(X_train,y_train)
pred = knn4.predict(X_test)

print('WITH K=4')
print('---------------')
print(confusion_matrix(y_test, pred))
print('---------------')
print(classification_report(y_test, pred))
```

```
WITH K=4
---------------
[[2929   72]
 [  92  505]]
---------------
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      3001
           1       0.88      0.85      0.86       597

    accuracy                           0.95      3598
   macro avg       0.92      0.91      0.92      3598
weighted avg       0.95      0.95      0.95      3598
```

In [133…
```
# NOW K=6

knn6 = KNeighborsClassifier(n_neighbors = 6)

knn6.fit(X_train,y_train)
pred = knn6.predict(X_test)

print('WITH K=6')
print('---------------')
print(confusion_matrix(y_test, pred))
print('---------------')
print(classification_report(y_test, pred))
```

```
WITH K=6
---------------
[[2905   96]
 [  89  508]]
---------------
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      3001
           1       0.84      0.85      0.85       597

    accuracy                           0.95      3598
   macro avg       0.91      0.91      0.91      3598
weighted avg       0.95      0.95      0.95      3598
```

In [134…
```
# NOW K=8

knn8 = KNeighborsClassifier(n_neighbors = 8)

knn8.fit(X_train,y_train)
pred = knn8.predict(X_test)

print('WITH K=8')
print('---------------')
print(confusion_matrix(y_test, pred))
print('---------------')
print(classification_report(y_test, pred))
```

```
WITH K=8
---------------
[[2903   98]
 [ 106  491]]
---------------
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      3001
           1       0.83      0.82      0.83       597

    accuracy                           0.94      3598
   macro avg       0.90      0.89      0.90      3598
weighted avg       0.94      0.94      0.94      3598
```

In [135…
```
# NOW K=10

knn10 = KNeighborsClassifier(n_neighbors = 10)

knn10.fit(X_train,y_train)
pred = knn10.predict(X_test)

print('WITH K=10')
print('---------------')
print(confusion_matrix(y_test, pred))
print('---------------')
print(classification_report(y_test, pred))
```

```
WITH K=10
---------------
[[2898  103]
 [ 116  481]]
---------------
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      3001
           1       0.82      0.81      0.81       597

    accuracy                           0.94      3598
   macro avg       0.89      0.89      0.89      3598
weighted avg       0.94      0.94      0.94      3598
```

As seen above we are getting the best results with default K (K=5)

8.3.5 KNeighbors Classifier GridsearchCV for Choosing Reasonable K Values

In [136…
```
k_values = range(1, 30)
param_grid = {"n_neighbors": k_values, "p": [1, 2], "weights": ['uniform', "distance"]}
```

In [137…
```
KNN_grid = KNeighborsClassifier()
KNN_grid_model = GridSearchCV(KNN_grid, param_grid, cv = 10, scoring = 'recall')
KNN_grid_model.fit(X_train, y_train)
```

Out[137]:
```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

In [138…
```
KNN_grid_model.best_estimator_
```
Out[138]:
```
▾                         KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3, p=1, weights='distance')
```

In [139…
```
print(colored('\033[1mBest Parameters of GridSearchCV for KNN Model:\033[0m', 'blue'), colored(KNN_grid_model.best_params_, 'red'))
```
Best Parameters of GridSearchCV for KNN Model: {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}

In [140…
```
# NOW WITH K=3

KNN_tuned3 = KNeighborsClassifier(n_neighbors = 3, p = 1, weights = 'distance')
KNN_tuned3.fit(X_train, y_train)
y_pred = KNN_tuned3.predict(X_test)
y_train_pred = KNN_tuned3.predict(X_train)

KNN_tuned3_f1 = f1_score(y_test, y_pred)
KNN_tuned3_acc = accuracy_score(y_test, y_pred)
KNN_tuned3_recall = recall_score(y_test, y_pred)
KNN_tuned3_auc = roc_auc_score(y_test, y_pred)

print("KNN_tuned (K=3)")
print ("------------------")
eval(KNN_tuned3, X_train, X_test)
train_val(y_train, y_train_pred, y_test, y_pred)
```
```
KNN_tuned (K=3)
------------------
[[2897  104]
 [  79  518]]
Test_Set
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      3001
           1       0.83      0.87      0.85       597

    accuracy                           0.95      3598
   macro avg       0.90      0.92      0.91      3598
weighted avg       0.95      0.95      0.95      3598

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6999
           1       1.00      1.00      1.00      1394

    accuracy                           1.00      8393
   macro avg       1.00      1.00      1.00      8393
weighted avg       1.00      1.00      1.00      8393
```
Out[140]:

	train_set	test_set
Accuracy	1.000	0.949
Precision	1.000	0.833
Recall	1.000	0.868
f1	1.000	0.850

In [141…
```
# NOW WITH K=1

KNN_tuned1 = KNeighborsClassifier(n_neighbors = 1, p = 1, weights = 'distance')
KNN_tuned1.fit(X_train, y_train)
y_pred = KNN_tuned1.predict(X_test)
y_train_pred = KNN_tuned1.predict(X_train)

KNN_tuned1_f1 = f1_score(y_test, y_pred)
KNN_tuned1_acc = accuracy_score(y_test, y_pred)
KNN_tuned1_recall = recall_score(y_test, y_pred)
KNN_tuned1_auc = roc_auc_score(y_test, y_pred)

print("KNN_tuned (K=1)")
print ("------------------")
eval(KNN_tuned1, X_train, X_test)
```
```
KNN_tuned (K=1)
------------------
[[2882  119]
 [  84  513]]
Test_Set
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      3001
           1       0.81      0.86      0.83       597

    accuracy                           0.94      3598
   macro avg       0.89      0.91      0.90      3598
weighted avg       0.95      0.94      0.94      3598

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6999
           1       1.00      1.00      1.00      1394

    accuracy                           1.00      8393
   macro avg       1.00      1.00      1.00      8393
weighted avg       1.00      1.00      1.00      8393
```

K=1 and K=3 have the same error rate. In order to reduce the complexity of the model, we can be continue with K=1 as the tuned_model. However, I will continue with tuned_model K=3 since recall values are better for K=3.

8.3.6 Prediction

In [142…
```
cprint('KNN_tuned Predictions','green', 'on_red')
KNN_Pred = {"Actual": y_test, "KNN_Pred":y_pred}
KNN_Pred = pd.DataFrame.from_dict(KNN_Pred)
KNN_Pred.head()
```
KNN_tuned Predictions

Out[142]:

	Actual	KNN_Pred
3118	0	0
10490	0	0
1106	1	1
3822	0	0
6873	0	0

In [143…
```
cprint('Predictions','green', 'on_red')
KNN_Pred.drop("Actual", axis = 1, inplace = True)
Model_Preds = pd.merge(Model_Preds, KNN_Pred, left_index = True, right_index = True)
Model_Preds.head()
```

Predictions

Out[143]:

	Actual	GB_Pred	KNN_Pred
3118	0	0	0
10490	0	0	0
1106	1	1	1
3822	0	0	0
6873	0	0	0

8.4 - Random Forest Classifier

8.4.1 Model Building

In [144…

```
RF_model = RandomForestClassifier(class_weight = "balanced", random_state = 101)
RF_model.fit(X_train, y_train)
y_pred = RF_model.predict(X_test)
y_train_pred = RF_model.predict(X_train)

RF_model_f1 = f1_score(y_test, y_pred)
RF_model_acc = accuracy_score(y_test, y_pred)
RF_model_recall = recall_score(y_test, y_pred)
RF_model_auc = roc_auc_score(y_test, y_pred)
```

8.4.2 Evaluating Model Performance

In [145…

```
print("RF_Model")
print ("------------------")
eval(RF_model, X_train, X_test)
```

```
RF_Model
------------------
[[2996    5]
 [  56  541]]
Test_Set
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3001
           1       0.99      0.91      0.95       597

    accuracy                           0.98      3598
   macro avg       0.99      0.95      0.97      3598
weighted avg       0.98      0.98      0.98      3598

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6999
           1       1.00      1.00      1.00      1394

    accuracy                           1.00      8393
   macro avg       1.00      1.00      1.00      8393
weighted avg       1.00      1.00      1.00      8393
```

In [146…

```
cprint('RF_model Scores','green', 'on_red')
train_val(y_train, y_train_pred, y_test, y_pred)
```

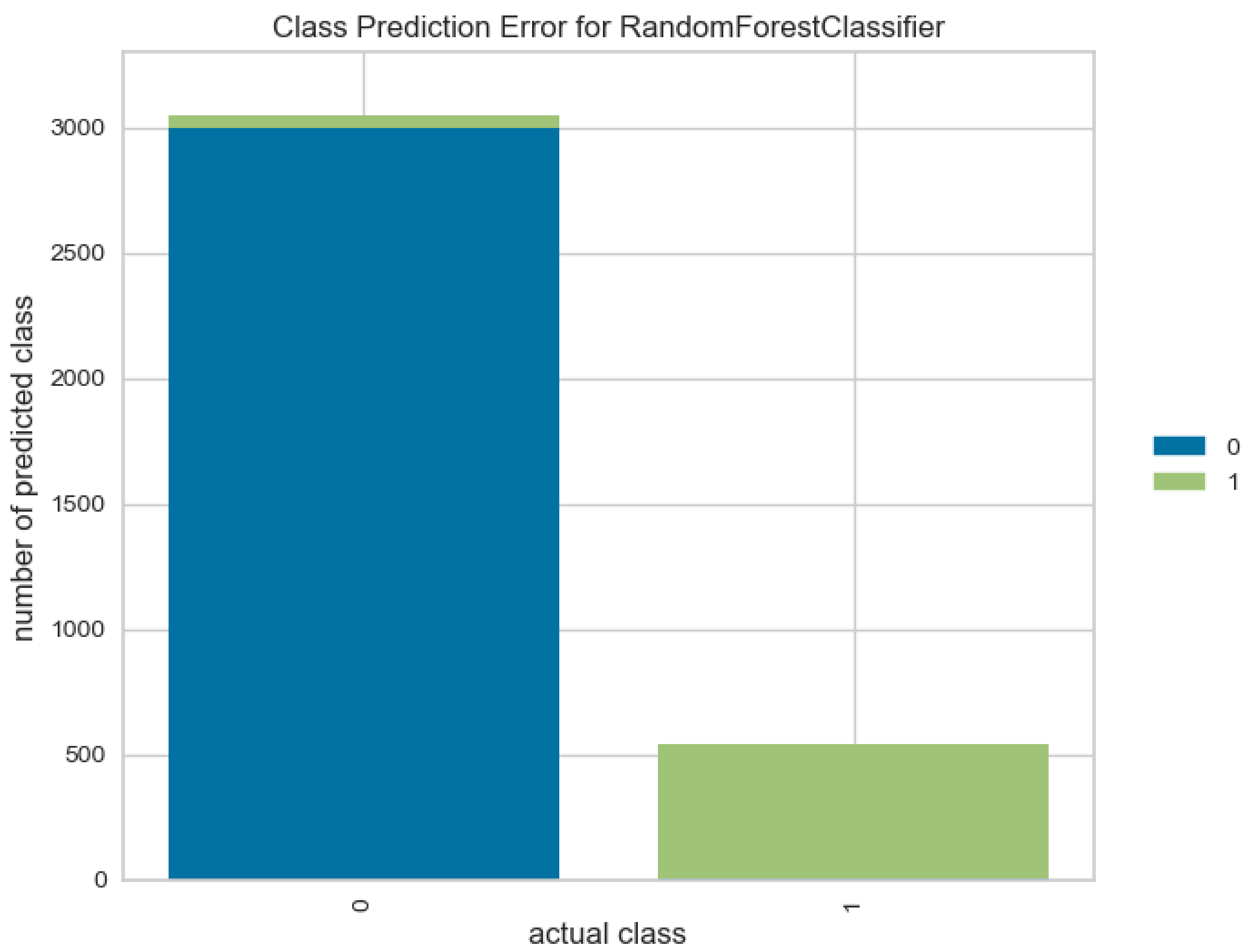
RF_model Scores

Out[146]:

	train_set	test_set
Accuracy	1.000	0.983
Precision	1.000	0.991
Recall	1.000	0.906
f1	1.000	0.947

In [147…

```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(RF_model)
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
# Draw visualization
visualizer.poof();
```



8.4.3 Feature Importance for Random Forest Model

In [148…

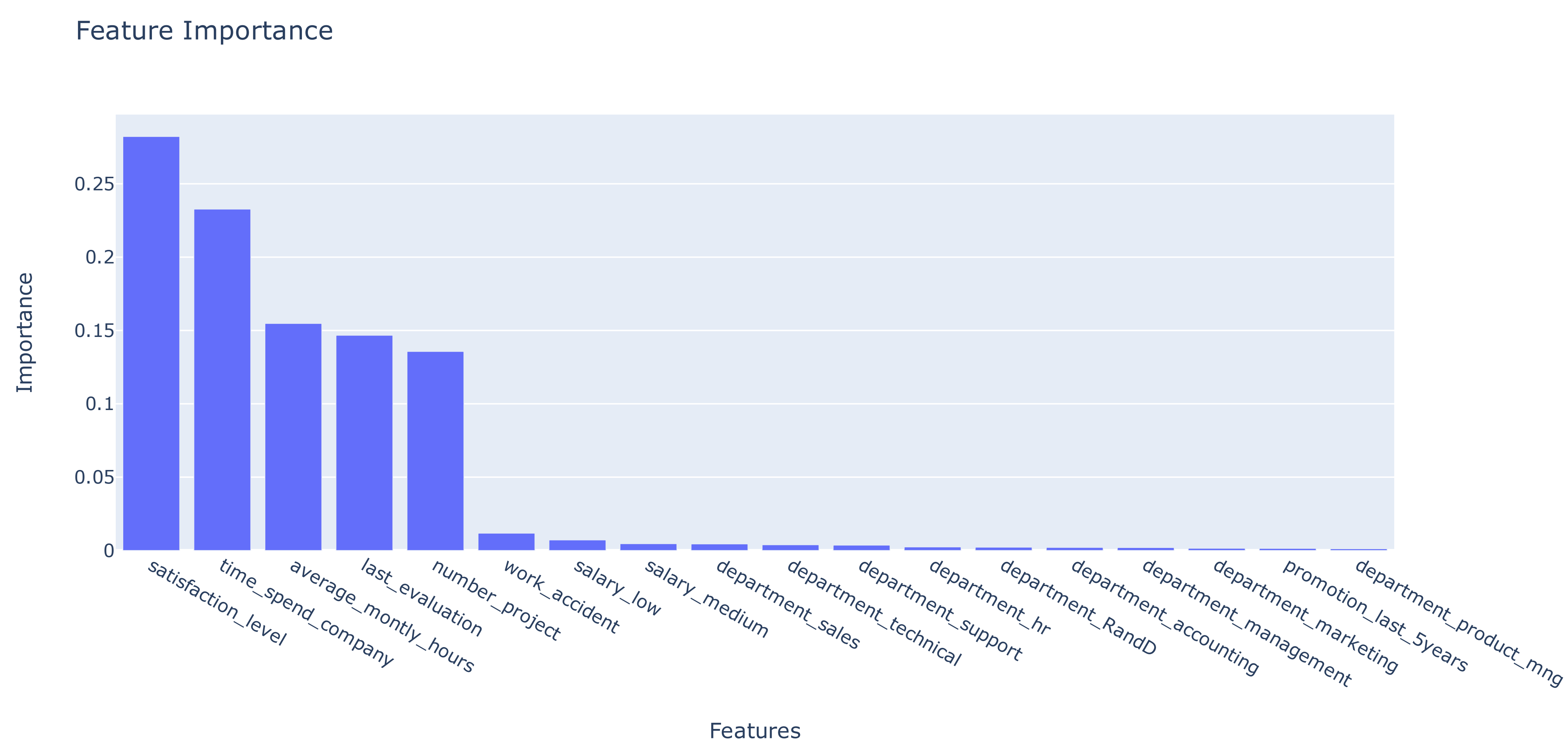
```
RF_feature_imp = pd.DataFrame(index=X.columns, data = RF_model.feature_importances_, columns = ['Importance']).sort_values("Importance", ascending 
RF_feature_imp
```

Out[148]:

	Importance
satisfaction_level	0.282
time_spend_company	0.233
average_montly_hours	0.155
last_evaluation	0.147
number_project	0.136
work_accident	0.012
salary_low	0.007
salary_medium	0.005
department_sales	0.004
department_technical	0.004
department_support	0.004
department_hr	0.002
department_RandD	0.002
department_accounting	0.002
department_management	0.002
department_marketing	0.001
promotion_last_5years	0.001
department_product_mng	0.001

In [149…

```
fig = px.bar(RF_feature_imp.sort_values('Importance', ascending = False), x = RF_feature_imp.sort_values('Importance',
             ascending = False).index, y = 'Importance', title = "Feature Importance",
             labels = dict(x = "Features", y ="Feature_Importance"))
fig.show()
```



8.4.4 Random Forest Classifier Cross Validation

In [150…

```
RF_cv = RandomForestClassifier(class_weight = "balanced", random_state = 101)
RF_cv_scores = cross_validate(RF_cv, X_train, y_train,
                              scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
RF_cv_scores = pd.DataFrame(RF_cv_scores, index = range(1, 11))
RF_cv_scores.mean()[2:]
```

Out[150]:

```
test_accuracy    0.982
test_precision   0.982
test_recall      0.906
test_f1          0.942
test_roc_auc     0.981
dtype: float64
```

8.4.5 Random Forest Classifier GridSearchCV

In [151…

```
param_grid = {'n_estimators' : [50, 100, 300],
              'max_features' : [2, 3, 4],
              'max_depth' : [3, 5, 7, 9],
              'min_samples_split' : [2, 5, 8]}
```

In [152…

```
RF_grid = RandomForestClassifier(class_weight = 'balanced', random_state = 101)
RF_grid_model = GridSearchCV(estimator = RF_grid,
                             param_grid = param_grid,
                             scoring = "recall",
                             n_jobs = -1, verbose = 2)
RF_grid_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

Out[152]:

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

In [153…

```
RF_grid_model.best_estimator_
```

Out[153]:

```
▾ RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=3, max_features=4,
                       n_estimators=300, random_state=101)
```

In [154…

```
print(colored('\033[1mBest Parameters of GridSearchCV for Random Forest Model:\033[0m', 'blue'), colored(RF_grid_model.best_params_, 'red'))
```

```
Best Parameters of GridSearchCV for Random Forest Model: {'max_depth': 3, 'max_features': 4, 'min_samples_split': 2, 'n_estimators': 300}
```

In [155…

```
RF_tuned = RandomForestClassifier(class_weight = 'balanced',
                                  max_depth = 3,
                                  max_features = 4,
                                  n_estimators = 300,
                                  min_samples_split = 2,
                                  random_state = 101).fit(X_train, y_train)
```

In [156…
```
y_pred = RF_tuned.predict(X_test)
y_train_pred = RF_tuned.predict(X_train)

RF_tuned_f1 = f1_score(y_test, y_pred)
RF_tuned_acc = accuracy_score(y_test, y_pred)
RF_tuned_recall = recall_score(y_test, y_pred)
RF_tuned_auc = roc_auc_score(y_test, y_pred)
```

In [157…
```
print("RF_tuned")
print ("------------------")
eval(RF_tuned, X_train, X_test)
```

```
RF_tuned
------------------
[[2804  197]
 [  47  550]]
Test_Set
              precision    recall  f1-score   support

           0       0.98      0.93      0.96      3001
           1       0.74      0.92      0.82       597

    accuracy                           0.93      3598
   macro avg       0.86      0.93      0.89      3598
weighted avg       0.94      0.93      0.94      3598

Train_Set
              precision    recall  f1-score   support

           0       0.99      0.93      0.96      6999
           1       0.73      0.94      0.82      1394

    accuracy                           0.93      8393
   macro avg       0.86      0.93      0.89      8393
weighted avg       0.94      0.93      0.94      8393
```

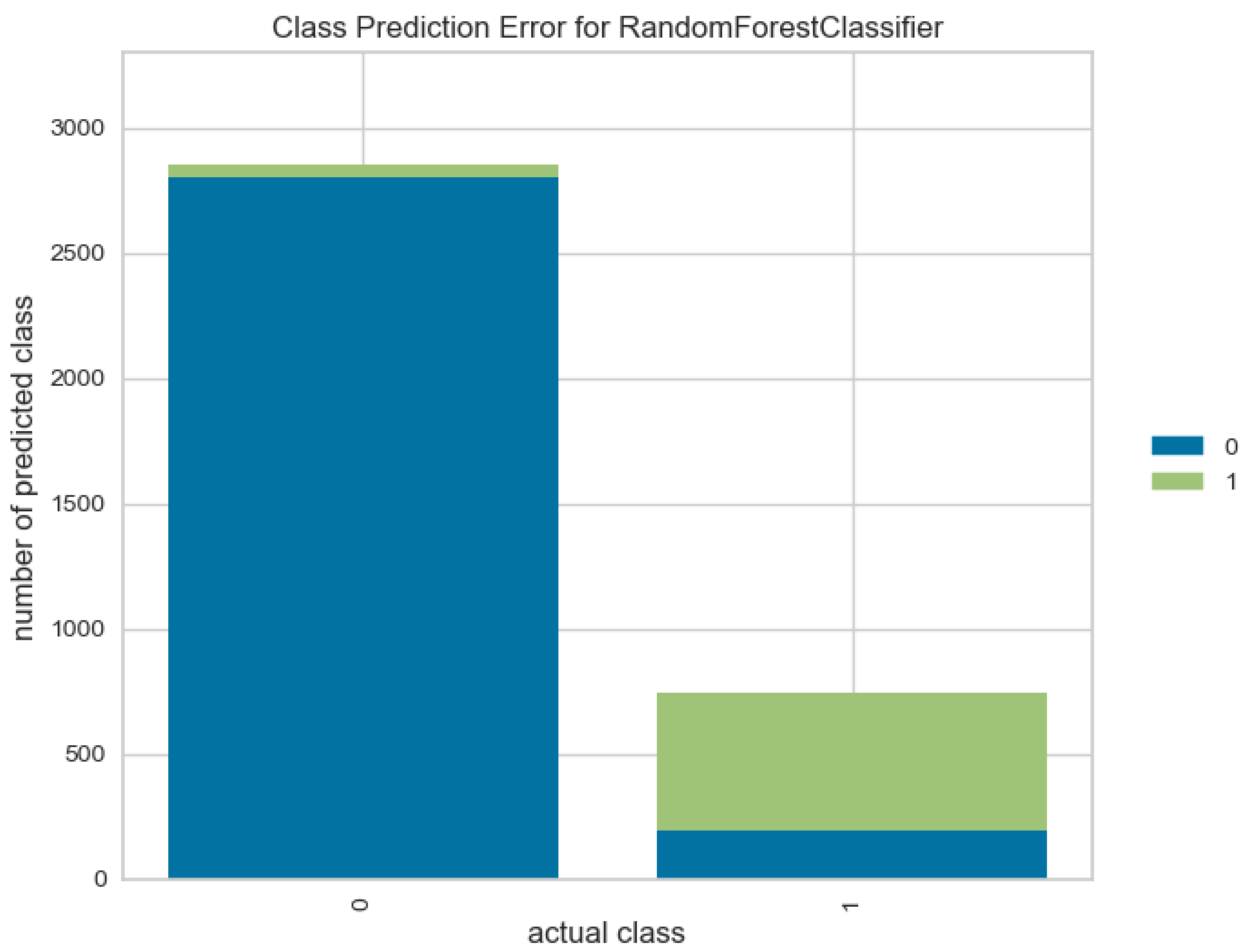
In [158…
```
cprint('RF_tuned Scores','green', 'on_red')
train_val(y_train, y_train_pred, y_test, y_pred)
```

RF_tuned Scores

Out[158]:

	train_set	test_set
Accuracy	0.932	0.932
Precision	0.731	0.736
Recall	0.937	0.921
f1	0.821	0.818

In [159…
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(RF_tuned)
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
# Draw visualization
visualizer.poof();
```



8.4.6 Prediction

In [160…
```
cprint('RF_tuned Predictions','green', 'on_red')
RF_Pred = {"Actual": y_test, "RF_Pred":y_pred}
RF_Pred = pd.DataFrame.from_dict(RF_Pred)
RF_Pred.head()
```

RF_tuned Predictions

Out[160]:

	Actual	RF_Pred
3118	0	0
10490	0	0
1106	1	1
3822	0	0
6873	0	0

In [161…
```
cprint('Predictions','green', 'on_red')
RF_Pred.drop("Actual", axis = 1, inplace = True)
Model_Preds = pd.merge(Model_Preds, RF_Pred, left_index = True, right_index = True)
Model_Preds.head()
```

Predictions

Out[161]:

	Actual	GB_Pred	KNN_Pred	RF_Pred
3118	0	0	0	0
10490	0	0	0	0
1106	1	1	1	1
3822	0	0	0	0
6873	0	0	0	0

8.5 - CatBoost Classifier

8.5.1 Model Building

In [162…
```
CB_model = CatBoostClassifier(verbose = False, scale_pos_weight = 4, random_state = 101)
CB_model.fit(X_train, y_train)
y_pred = CB_model.predict(X_test)
y_train_pred = CB_model.predict(X_train)

CB_model_f1 = f1_score(y_test, y_pred)
CB_model_acc = accuracy_score(y_test, y_pred)
CB_model_recall = recall_score(y_test, y_pred)
CB_model_auc = roc_auc_score(y_test, y_pred)
```

8.5.2 Evaluating Model Performance

In [163…
```
print("CB_Model")
print ("------------------")
eval(CB_model, X_train, X_test)
```

```
CB_Model
------------------
[[2977   24]
 [  47  550]]
Test_Set
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3001
           1       0.96      0.92      0.94       597

    accuracy                           0.98      3598
   macro avg       0.97      0.96      0.96      3598
weighted avg       0.98      0.98      0.98      3598

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6999
           1       0.98      0.99      0.98      1394

    accuracy                           0.99      8393
   macro avg       0.99      0.99      0.99      8393
weighted avg       0.99      0.99      0.99      8393
```

In [164…
```
cprint('CB_model Scores','green', 'on_red')
train_val(y_train, y_train_pred, y_test, y_pred)
```

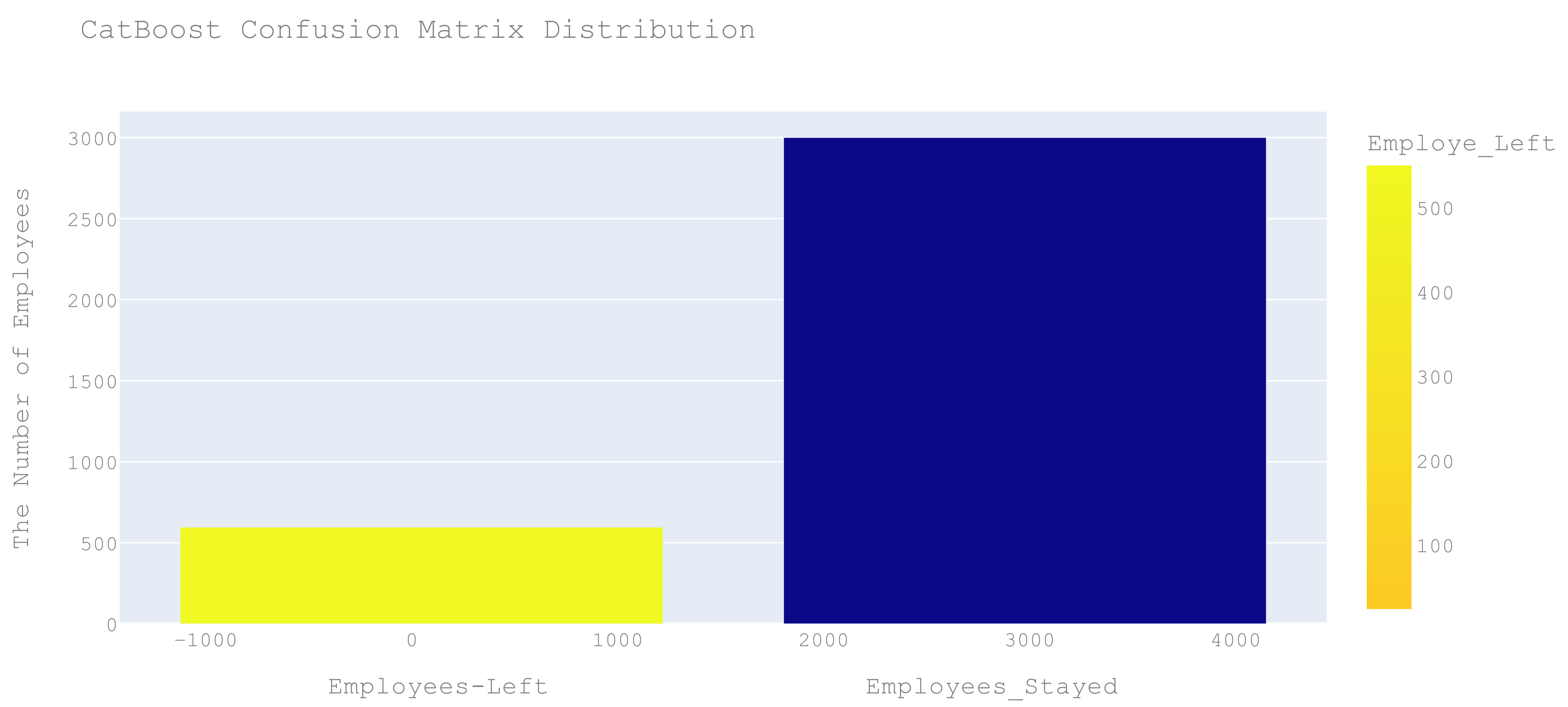
```
CB_model Scores
```

Out[164]:

	train_set	test_set
Accuracy	0.994	0.980
Precision	0.977	0.958
Recall	0.987	0.921
f1	0.982	0.939

In [165…
```
CB_cm = confusion_matrix(y_test, y_pred)
CB_cm_df = pd.DataFrame(CB_cm)
CB_cm_df = CB_cm_df.rename(columns={0:"Employee_Stayed", 1:"Employe_Left"}, index={0:"Employee_Stayed", 1:"Employe_Left"})
CB_cm_df["Total"] = CB_cm_df["Employee_Stayed"] + CB_cm_df["Employe_Left"]
```

In [166…
```
fig = px.bar(CB_cm_df, x="Employee_Stayed", y="Total", color="Employe_Left", title="CatBoost Confusion Matrix Distribution")

fig.update_layout(
    xaxis_title="Employees-Left                    Employees_Stayed",
    yaxis_title="The Number of Employees",
    font=dict(
        family="Courier New, monospace",
        size=14,
        color="#7f7f7f"
    )
)

fig.show()
```



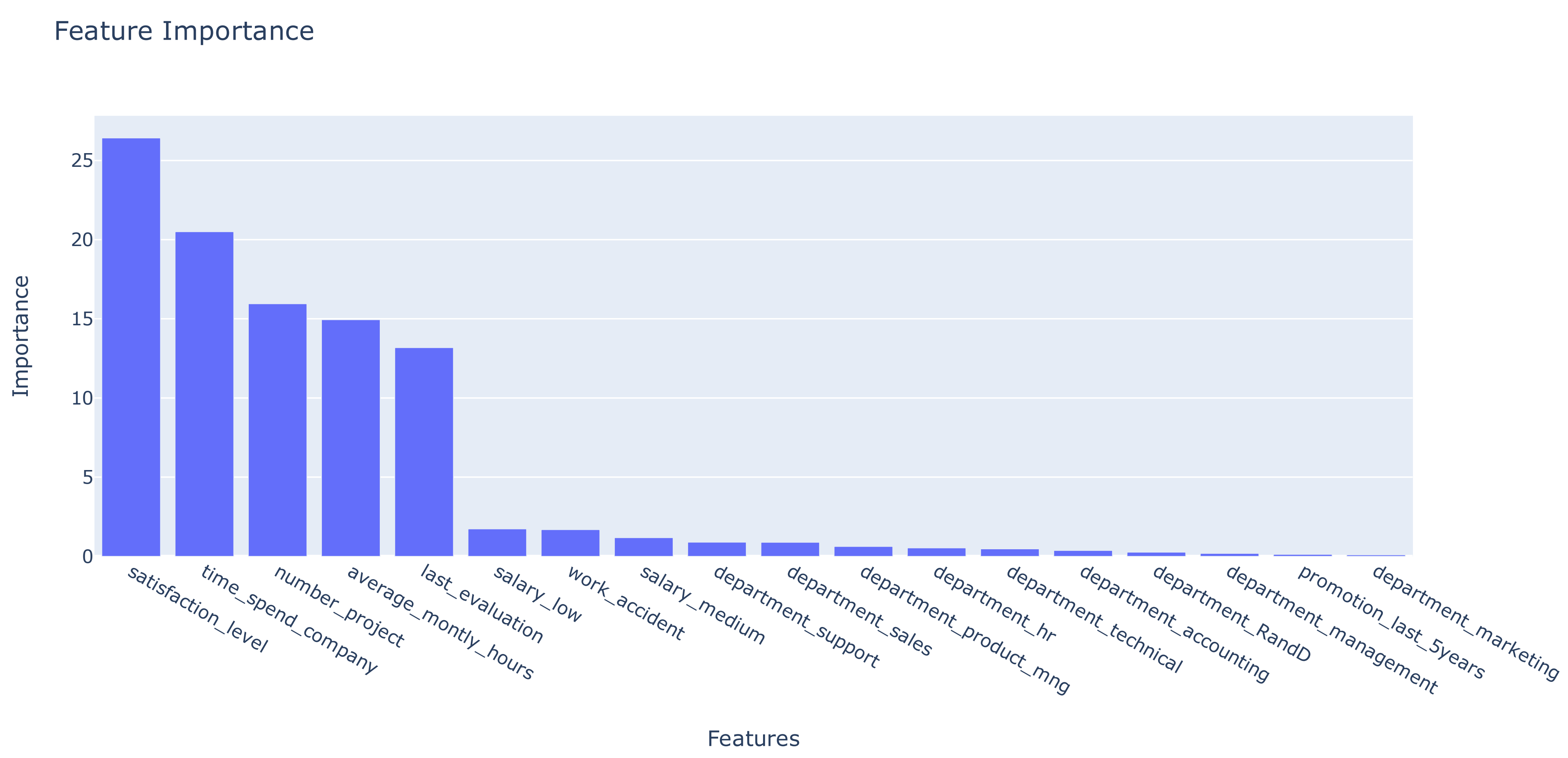
8.5.3 Feature Importance for CatBoost Model

In [167…
```
CB_feature_imp = pd.DataFrame(index = X.columns, data = CB_model.feature_importances_, columns = ['Importance']).sort_values("Importance", ascending
CB_feature_imp
```

Out[167]:

	Importance
satisfaction_level	26.423
time_spend_company	20.504
number_project	15.951
average_montly_hours	14.938
last_evaluation	13.180
salary_low	1.730
work_accident	1.684
salary_medium	1.175
department_support	0.892
department_sales	0.886
department_product_mng	0.618
department_hr	0.528
department_technical	0.466
department_accounting	0.370
department_RandD	0.260
department_management	0.181
promotion_last_5years	0.124
department_marketing	0.091

In [168…
```
fig = px.bar(CB_feature_imp.sort_values('Importance', ascending = False), x = CB_feature_imp.sort_values('Importance',
             ascending = False).index, y = 'Importance', title = "Feature Importance",
             labels = dict(x = "Features", y ="Feature_Importance"))
fig.show()
```



8.5.4 CatBoost Classifier Cross Validation

In [169…
```
CB_cv = CatBoostClassifier(verbose = False, scale_pos_weight = 4, random_state = 101)
CB_cv_scores = cross_validate(CB_cv, X_train, y_train,
                              scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
CB_cv_scores = pd.DataFrame(CB_cv_scores, index = range(1, 11))

CB_cv_scores.mean()[2:]
```

Out[169]:
```
test_accuracy    0.980
test_precision   0.952
test_recall      0.924
test_f1          0.938
test_roc_auc     0.983
dtype: float64
```

8.5.5 CatBoost Classifier GridSearchCV

In [170…
```
param_grid = {'learning_rate': [0.01, 0.03, 0.1, 0.5],
              'depth': [4, 6, 8, 10],
              'l2_leaf_reg': [1, 3, 5, 7, 9]}
```

In [171…
```
CB_grid = CatBoostClassifier(verbose = False, scale_pos_weight = 4, random_state = 101)
CB_grid_model = GridSearchCV(estimator = CB_grid,
                             param_grid = param_grid,
                             scoring = "recall",
                             n_jobs = -1, verbose = 2)
CB_grid_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
```

Out[171]:
```
▸ GridSearchCV
▸ estimator: CatBoostClassifier
  ▸ CatBoostClassifier
```

In [172…
```
CB_grid_model.best_params_
```

Out[172]: `{'depth': 4, 'l2_leaf_reg': 3, 'learning_rate': 0.01}`

In [173…
```
print(colored('\033[1mBest Parameters of GridSearchCV forCatBoost Model:\033[0m', 'blue'), colored(CB_grid_model.best_params_, 'red'))
```

Best Parameters of GridSearchCV forCatBoost Model: {'depth': 4, 'l2_leaf_reg': 3, 'learning_rate': 0.01}

In [174…
```
CB_tuned = CatBoostClassifier(verbose = False,
                              scale_pos_weight = 4,
                              depth = 4,
                              l2_leaf_reg = 3,
                              learning_rate = 0.01,
                              random_state = 101).fit(X_train, y_train)
```

In [175…
```
y_pred = CB_tuned.predict(X_test)
y_train_pred = CB_tuned.predict(X_train)

CB_tuned_f1 = f1_score(y_test, y_pred)
CB_tuned_acc = accuracy_score(y_test, y_pred)
CB_tuned_recall = recall_score(y_test, y_pred)
CB_tuned_auc = roc_auc_score(y_test, y_pred)
```

In [176…
```
print("CB_tuned")
print ("------------------")
eval(CB_tuned, X_train, X_test)
```

```
CB_tuned
------------------
[[2946   55]
 [  45  552]]
Test_Set
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3001
           1       0.91      0.92      0.92       597

    accuracy                           0.97      3598
   macro avg       0.95      0.95      0.95      3598
weighted avg       0.97      0.97      0.97      3598

Train_Set
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      6999
           1       0.92      0.94      0.93      1394

    accuracy                           0.98      8393
   macro avg       0.96      0.96      0.96      8393
weighted avg       0.98      0.98      0.98      8393
```

In [177…
```
cprint('CB_tuned Scores','green', 'on_red')
train_val(y_train, y_train_pred, y_test, y_pred)
```

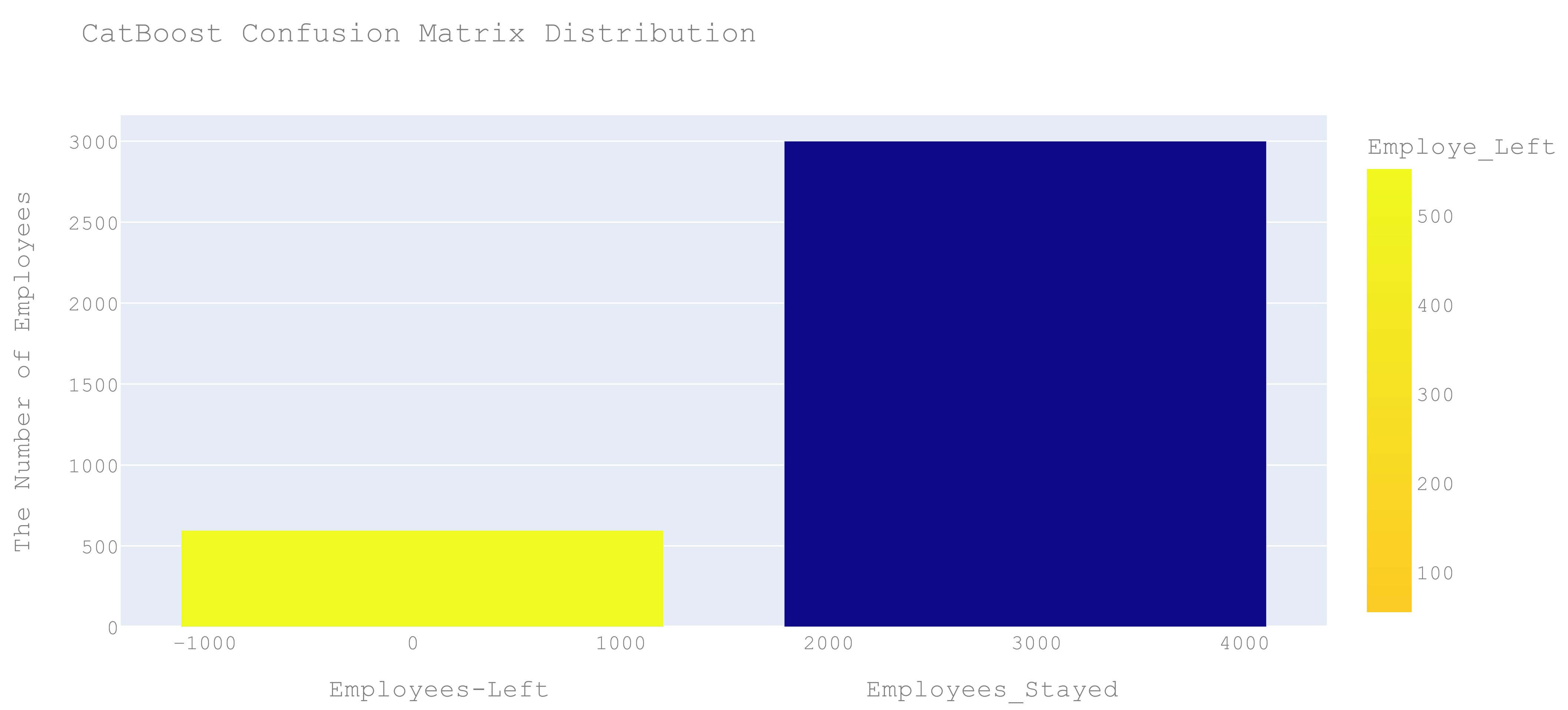
CB_tuned Scores

Out[177]:

	train_set	test_set
Accuracy	0.977	0.972
Precision	0.923	0.909
Recall	0.940	0.925
f1	0.931	0.917

In [178…
```
CB_cm = confusion_matrix(y_test, y_pred)
CB_cm_df = pd.DataFrame(CB_cm)
CB_cm_df = CB_cm_df.rename(columns={0:"Employee_Stayed", 1:"Employe_Left"}, index={0:"Employee_Stayed", 1:"Employe_Left"})
CB_cm_df["Total"] = CB_cm_df["Employee_Stayed"] + CB_cm_df["Employe_Left"]
```

In [179…
```
fig = px.bar(CB_cm_df, x="Employee_Stayed", y="Total", color="Employe_Left", title="CatBoost Confusion Matrix Distribution")

fig.update_layout(
    xaxis_title="Employees-Left                    Employees_Stayed",
    yaxis_title="The Number of Employees",
    font=dict(
        family="Courier New, monospace",
        size=14,
        color="#7f7f7f"
    )
)

fig.show()
```



8.5.6 Prediction

In [180…
```
cprint('CB_tuned Predictions','green', 'on_red')
CB_Pred = {"Actual": y_test, "CB_Pred":y_pred}
CB_Pred = pd.DataFrame.from_dict(CB_Pred)
CB_Pred.head()
```

CB_tuned Predictions

Out[180]:

	Actual	CB_Pred
3118	0	0
10490	0	0
1106	1	1
3822	0	0
6873	0	0

In [181…
```
cprint('Predictions','green', 'on_red')
CB_Pred.drop("Actual", axis = 1, inplace = True)
Model_Preds = pd.merge(Model_Preds, CB_Pred, left_index = True, right_index = True)
Model_Preds.head()
```

Predictions

Out[181]:

	Actual	GB_Pred	KNN_Pred	RF_Pred	CB_Pred
3118	0	0	0	0	0
10490	0	0	0	0	0
1106	1	1	1	1	1
3822	0	0	0	0	0
6873	0	0	0	0	0

In [182…

```
cprint('Random Predictions','green', 'on_red')
Model_Preds.sample(10)
```

```
Random Predictions
```

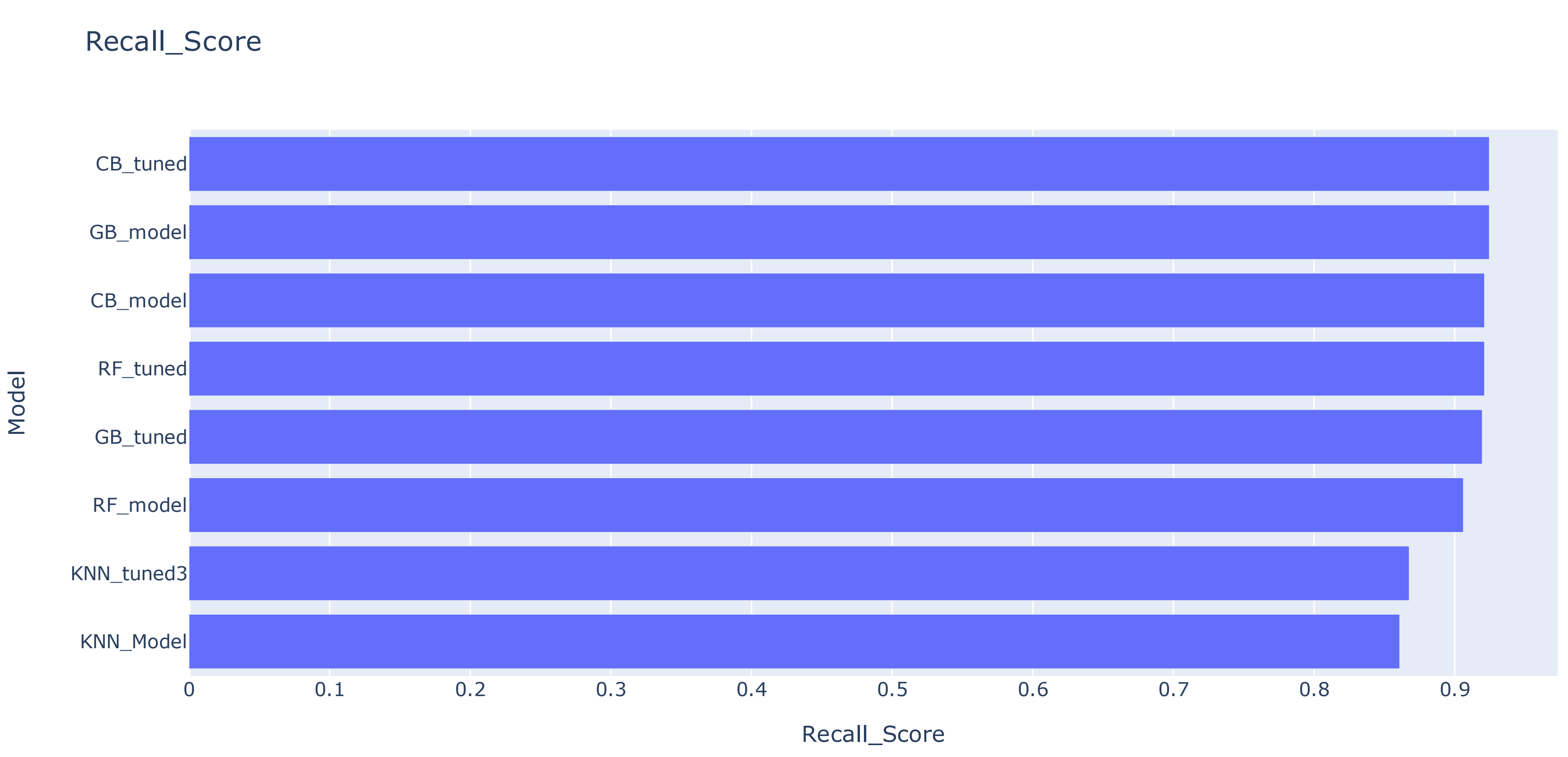
Out[182]:

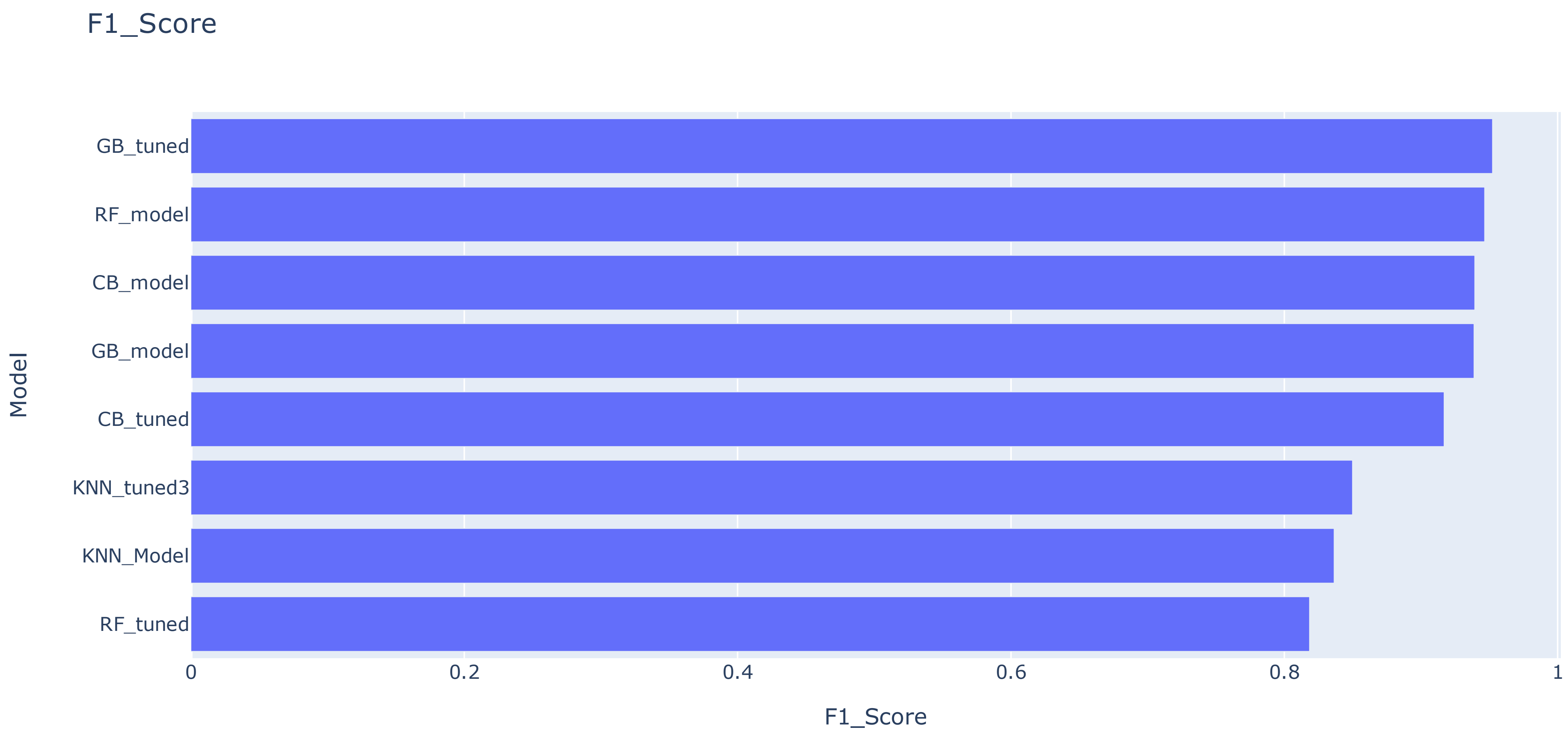
	Actual	GB_Pred	KNN_Pred	RF_Pred	CB_Pred
1253	1	1	1	1	1
239	1	1	1	1	1
10283	0	0	0	0	0
10044	0	0	0	0	0
11144	0	0	0	0	0
779	1	1	1	1	1
5321	0	0	0	0	0
2017	0	0	0	0	0
7664	0	0	1	0	0
10710	0	0	0	0	0

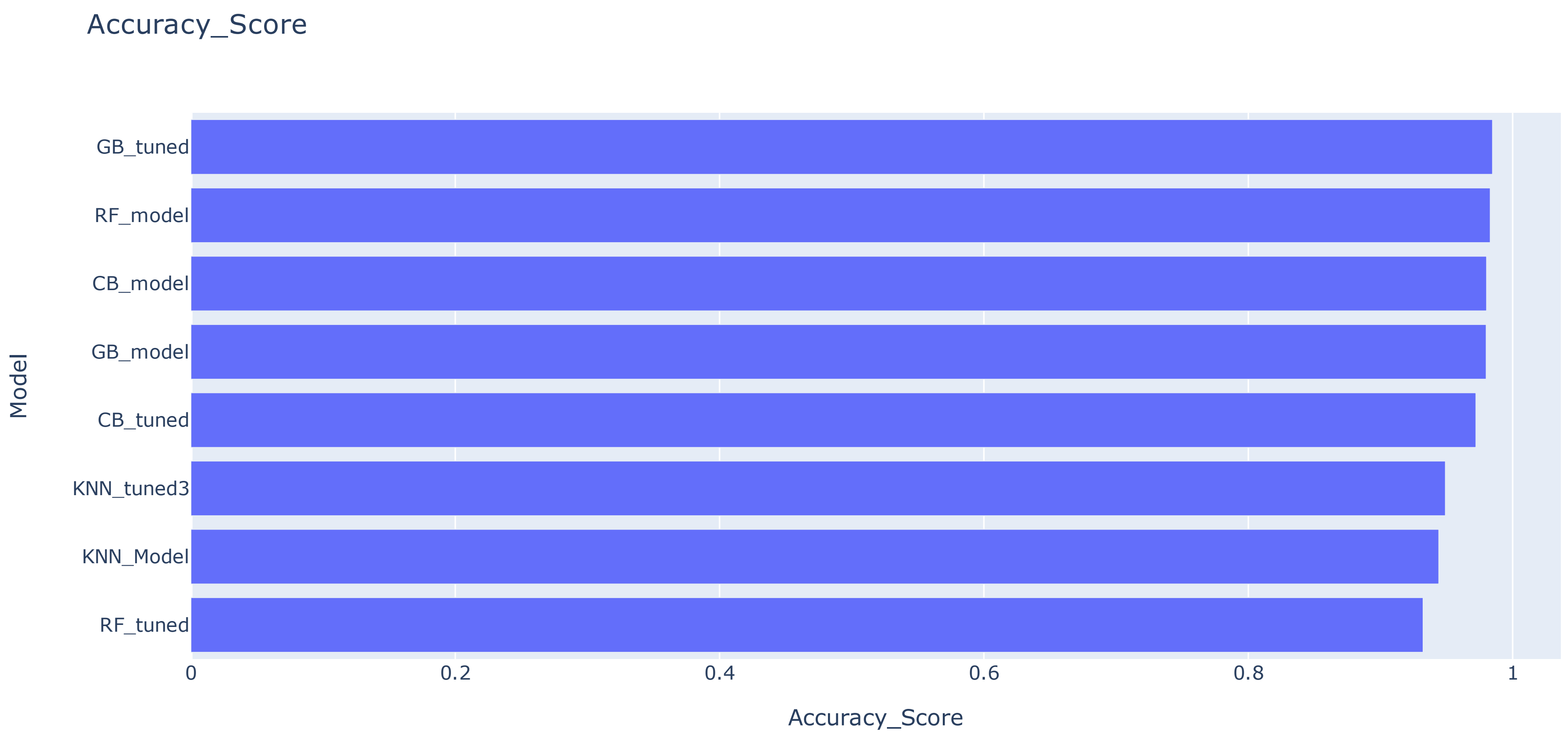
9 - THE COMPARISON OF MODELS

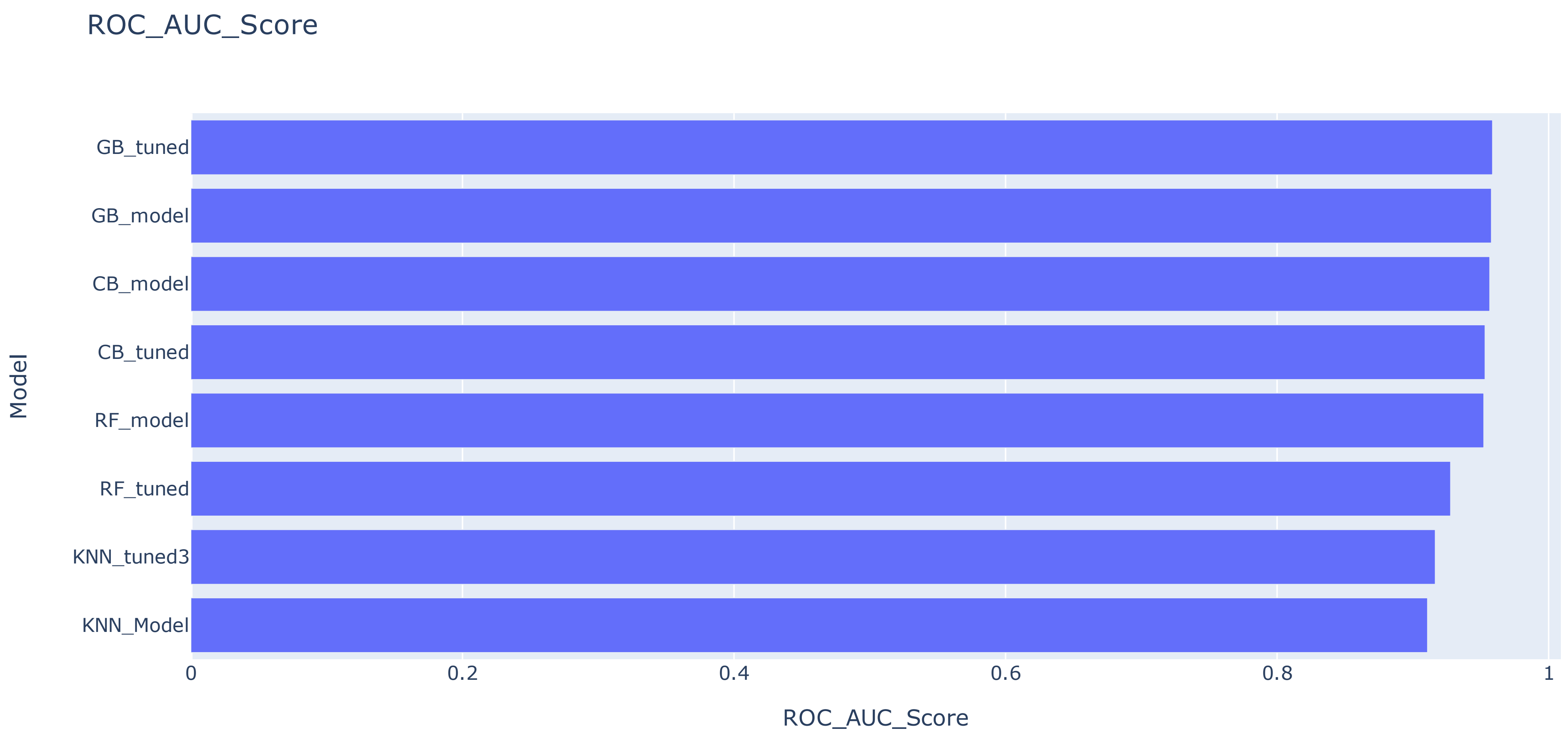
In [183…

```
compare = pd.DataFrame({"Model": ["GB_model", "GB_tuned", "KNN_Model", "KNN_tuned3", "RF_model", "RF_tuned", "CB_model",
                                  "CB_tuned"],

                        "F1_Score": [GB_model_f1, GB_tuned_f1, KNN_model_f1, KNN_tuned3_f1, RF_model_f1, RF_tuned_f1,
                                     CB_model_f1, CB_tuned_f1],

                        "Accuracy_Score": [GB_model_acc, GB_tuned_acc, KNN_model_acc, KNN_tuned3_acc, RF_model_acc,
                                           RF_tuned_acc, CB_model_acc, CB_tuned_acc],

                        "Recall_Score": [GB_model_recall, GB_tuned_recall, KNN_model_recall, KNN_tuned3_recall, RF_model_recall,
                                         RF_tuned_recall, CB_model_recall, CB_tuned_recall],

                        "ROC_AUC_Score": [GB_model_auc, GB_tuned_auc, KNN_model_auc, KNN_tuned3_auc, RF_model_auc,
                                          RF_tuned_auc, CB_model_auc, CB_tuned_auc]})

compare = compare.sort_values(by="Recall_Score", ascending=True)
fig = px.bar(compare, x = "Recall_Score", y = "Model", title = "Recall_Score")
fig.show()

compare = compare.sort_values(by="F1_Score", ascending=True)
fig = px.bar(compare, x = "F1_Score", y = "Model", title = "F1_Score")
fig.show()

compare = compare.sort_values(by="Accuracy_Score", ascending=True)
fig = px.bar(compare, x = "Accuracy_Score", y = "Model", title = "Accuracy_Score")
fig.show()

compare = compare.sort_values(by="ROC_AUC_Score", ascending=True)
fig = px.bar(compare, x = "ROC_AUC_Score", y = "Model", title = "ROC_AUC_Score")
fig.show()
```









In [184…

```
cprint('Scores','green', 'on_red')
compare.T
```

Scores

Out[184]:

	2	3	5	4	7	6	0	1
Model	KNN_Model	KNN_tuned3	RF_tuned	RF_model	CB_tuned	CB_model	GB_model	GB_tuned
F1_Score	0.836	0.850	0.818	0.947	0.917	0.939	0.939	0.952
Accuracy_Score	0.944	0.949	0.932	0.983	0.972	0.980	0.980	0.985
Recall_Score	0.861	0.868	0.921	0.906	0.925	0.921	0.925	0.920
ROC_AUC_Score	0.911	0.917	0.928	0.952	0.953	0.957	0.958	0.959

10 - MODEL DEPLOYMENT

- Save and Export the Model as .pkl
- Save and Export Variables as .pkl

You cooked the food in the kitchen and moved on to the serving stage. The question is how do you showcase your work to others? Model Deployement helps you showcase your work to the world and make better decisions with it. But, deploying a model can get a little tricky at times. Before deploying the model, many things such as data storage, preprocessing, model building and monitoring need to be studied. Streamlit is a popular open source framework used by data scientists for model distribution.

Deployment of machine learning models, means making your models available to your other business systems. By deploying models, other systems can send data to them and get their predictions, which are in turn populated back into the company systems. Through machine learning model deployment, can begin to take full advantage of the model you built.

Data science is concerned with how to build machine learning models, which algorithm is more predictive, how to design features, and what variables to use to make the models more accurate. However, how these models are actually used is often neglected. And yet this is the most important step in the machine learning pipline. Only when a model is fully integrated with the business systems, real values can be extract from its predictions.

After doing the following operations in this notebook, jump to new .py file and create your web app with Streamlit.

In [185…
```
gradient_boosting_classifier = pickle.dump(GB_tuned, open('gradient_boosting_model', 'wb'))
```

In [186…
```
kneighbors_classifier = pickle.dump(KNN_tuned3, open('kneighbors_model', 'wb'))
```

In [187…
```
random_forest_classifier = pickle.dump(RF_tuned, open('random_forest_model', 'wb'))
```

In [188…
```
catboost_classifier = pickle.dump(CB_tuned, open('catboost_model', 'wb'))
```

10.2 - Save and Export Variables as .pkl

In [189…
```
# col_lst = ['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', 'time_spend_company',
#            'work_accident', 'promotion_last_5years', 'department_RandD', 'department_accounting', 'department_hr',
#            'department_management', 'department_marketing', 'department_product_mng', 'department_sales',
#            'department_support', 'department_technical', 'salary_low', 'salary_medium']
# scaler = MinMaxScaler()
# scaler_fitted = scaler.fit(df2[col_lst])
# scaler_deploy = pickle.dump(scaler_fitted, open('scaler.sav', 'wb'))
```

11 - CONCLUSION

In this project we have HR data of a company. A study is requested from us to predict which employee will churn by using this data.

First of all, to observe the structure of the data, outliers, missing values and features that affect the target variable, we used exploratory data analysis and data visualization techniques.

Then, we performed data pre-processing operations such as ***Scaling*** and ***Label Encoding*** to increase the accuracy score of Gradient Descent Based or Distance-Based algorithms.**

We used the ***K-means*** algorithm to make cluster analysis. In order to find the optimal number of clusters, we used the ***Elbow method***. Briefly, tried to predict the set to which individuals were related by using K-means and evaluate the estimation results.

Then we built models to predict whether employees will churn or not. We trained our models with train set, tested the success of models with test set.

In this study, we made modelling with ***Gradient Boosting Classifier**, **K Neighbors Classifier**, **Random Forest Classifier*** and ***CatBoost Classifier***.

We used scikit-learn ***Confusion Metrics*** module for accuracy calculation and the ***Yellowbrick*** module for model selection and visualization.

In [ ]:

In [ ]: