```
import numpy as np # linear algebra
import pandas as pd # data processing,
CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import tr
ain_test_split
from sklearn.tree import DecisionTreeC
lassifier
from sklearn.metrics import precision_
score
from sklearn.tree import export_graphv
iz
from sklearn.metrics import recall_sco
re
import os
print(os.listdir("../input"))
sns.set()
```

```
['HR_comma_sep.csv']
```

In [2]:

```
df=pd.read_csv("../input/HR_comma_sep.
csv")
```

In [3]:

```
df.head(5)
```

Out[3]:

	satisfaction_level	last_evaluation	number_project	ave
0	0.38	0.53	2	15
1	0.80	0.86	5	26
2	0.11	0.88	7	27
3	0.72	0.87	5	22
4	0.37	0.52	2	15

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
satisfaction_level        14999 non-nul
l float64
last_evaluation           14999 non-nul
l float64
number_project            14999 non-nul
l int64
average_montly_hours      14999 non-nul
l int64
time_spend_company        14999 non-nul
l int64
Work_accident             14999 non-nul
l int64
left                      14999 non-nul
l int64
promotion_last_5years     14999 non-nul
```

```
department                    14999 non-nul
l object
salary                        14999 non-nul
l object
dtypes: float64(2), int64(6), object
(2)
memory usage: 1.1+ MB
```

Dataset contains 14999 rows and 10 columns, each row has the details of an employee.

2 variables are categorical, remaining columns are of int and float

Checking for any missing values

In [5]:

```
display(df.isnull().any())
```

```
salary                          False
dtype: bool
```

In [6]:

```
df.Department.unique()
```

Out[6]:

```
array(['sales', 'accounting', 'hr', 't
echnical', 'support', 'management',
       'IT', 'product_mng', 'marketin
g', 'RandD'], dtype=object)
```

In [7]:

```
df.salary.unique()
```

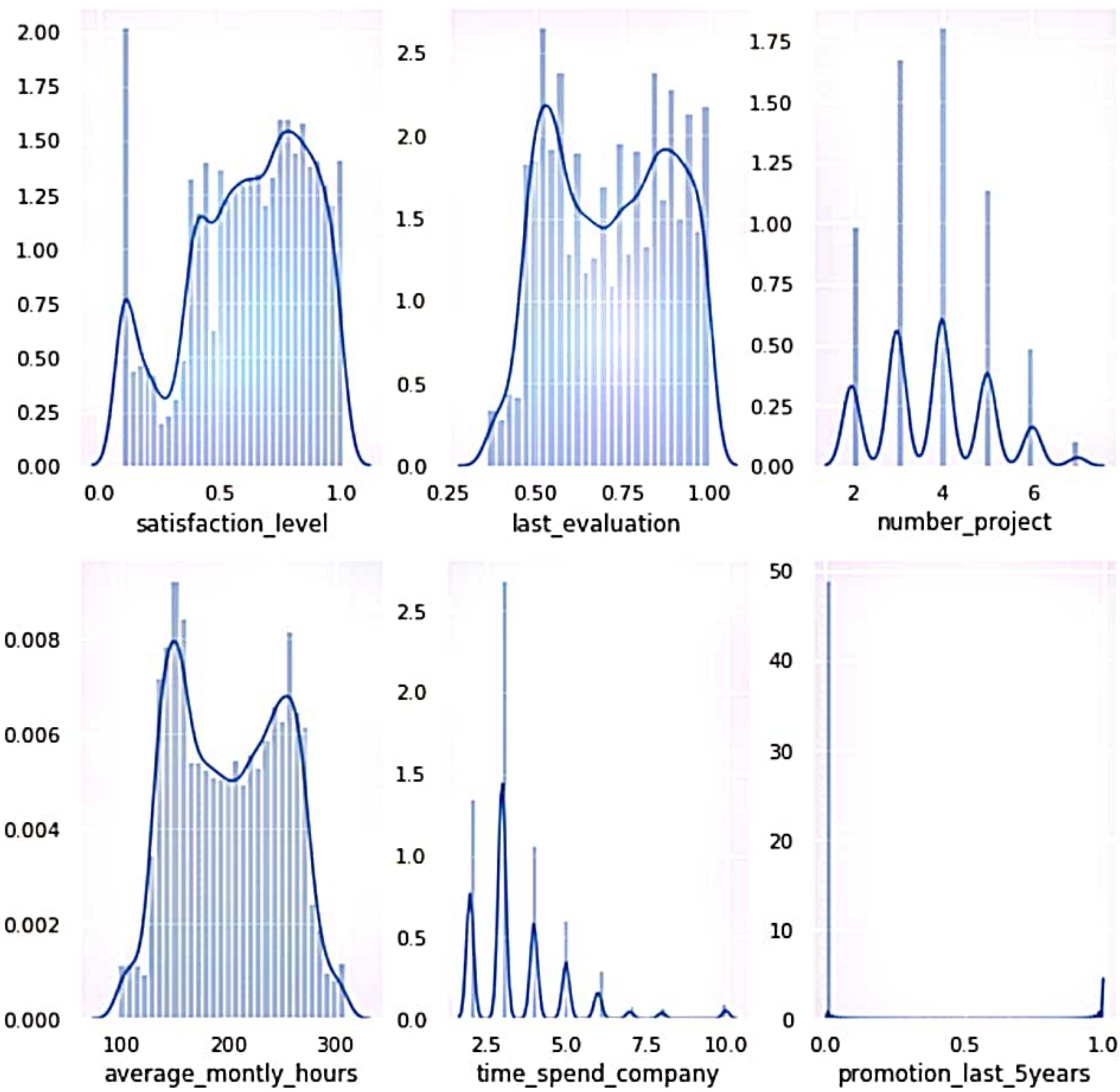
Out[7]:

```
array(['low', 'medium', 'high'], dtype
=object)
```

In [8]:

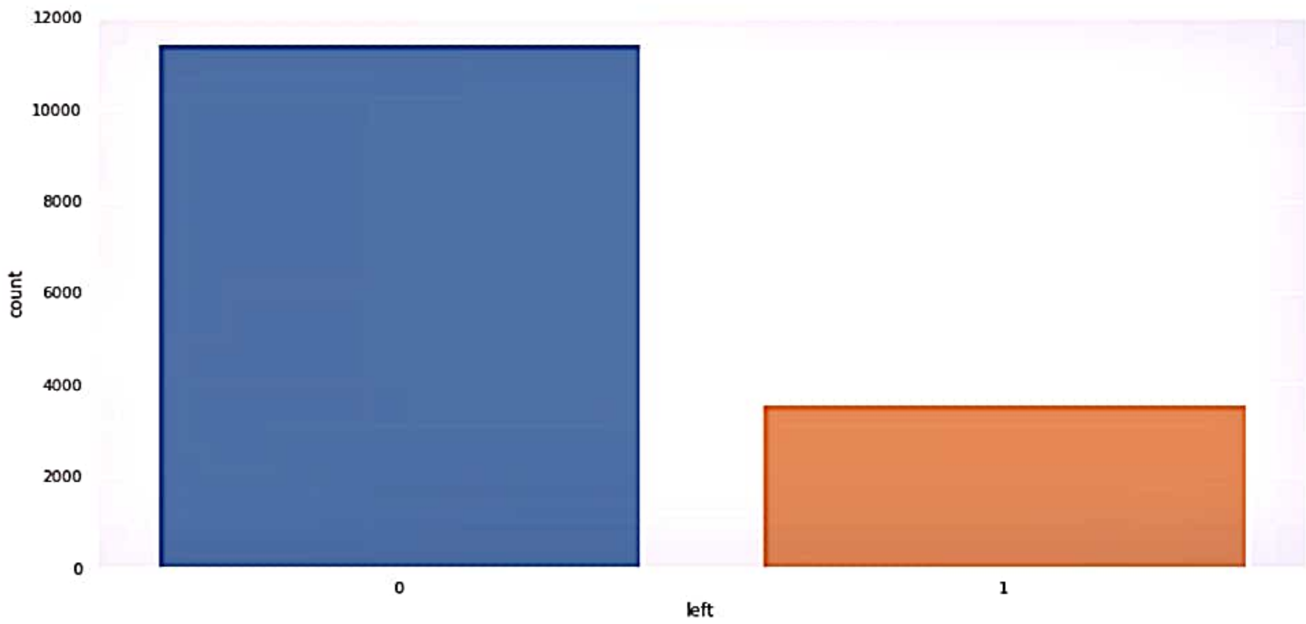
```
fig,ax = plt.subplots(2,3, figsize=(1
0,10))               # 'ax' has referen
ces to all the four axes
sns.distplot(df['satisfaction_level'],
ax = ax[0,0])
sns.distplot(df['last_evaluation'], ax
= ax[0,1])
sns.distplot(df['number_project'], ax
= ax[0,2])
sns.distplot(df['average_montly_hour
s'], ax = ax[1,0])
sns.distplot(df['time_spend_company'],
ax = ax[1,1])
sns.distplot(df['promotion_last_5year
s'], ax = ax[1,2])

plt.show()
```



satisfaction_level
last_evaluation
number_project
average_montly_hours
time_spend_company
promotion_last_5years

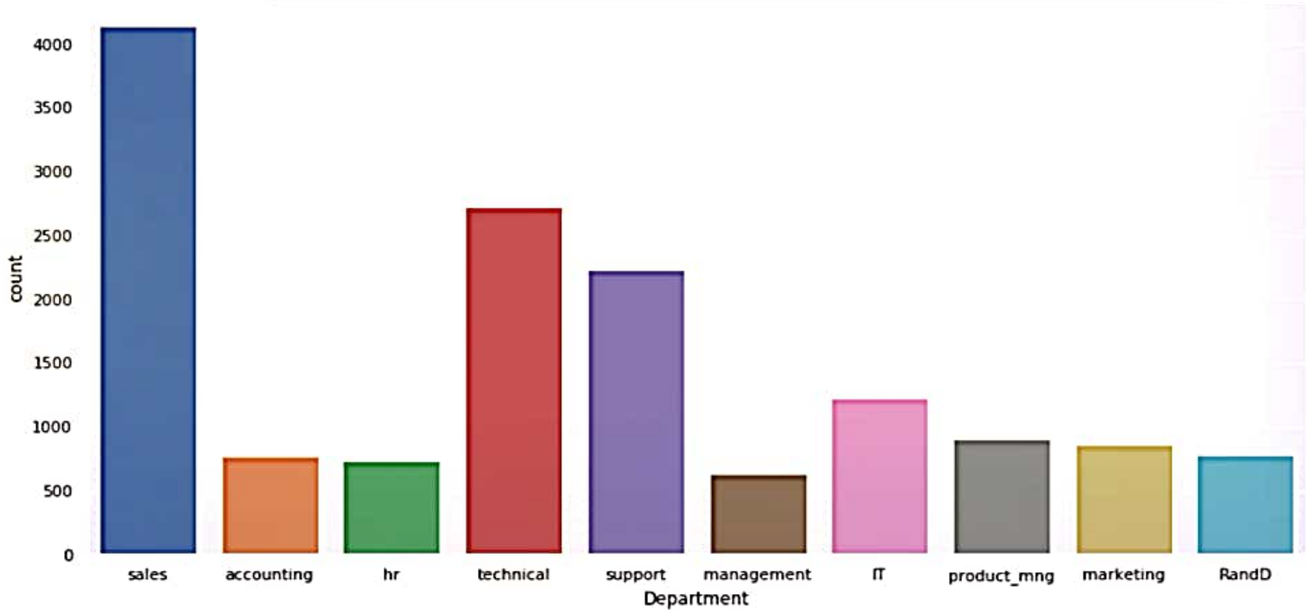
```
fig = plt.figure(figsize=(15,7))
sns.countplot(x='left',data=df)
plt.show()
```



Employees in each Department

In [10]:

```
fig = plt.figure(figsize=(15,7))
sns.countplot(x='Department',data=df)
plt.show()
```

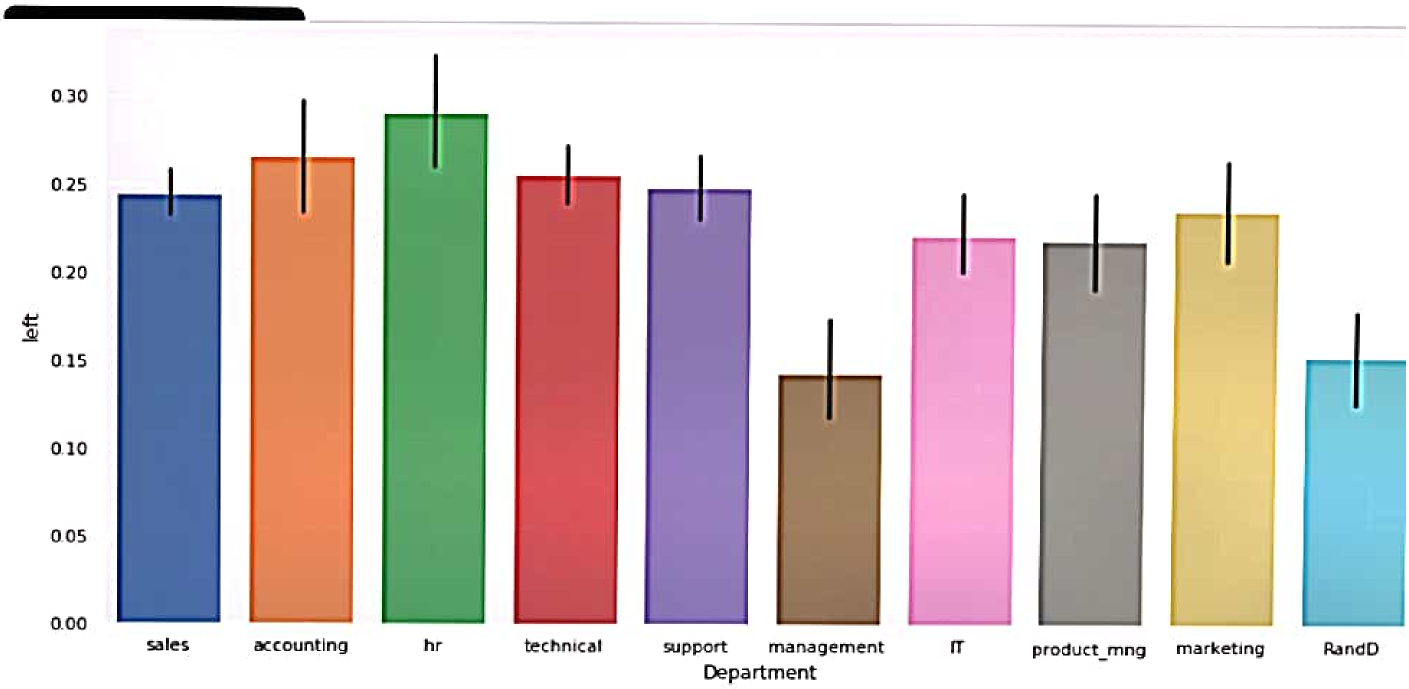


Sales Department has got more employees, next comes technical and Support departments.

Which Department employess left the company most

In [11]:

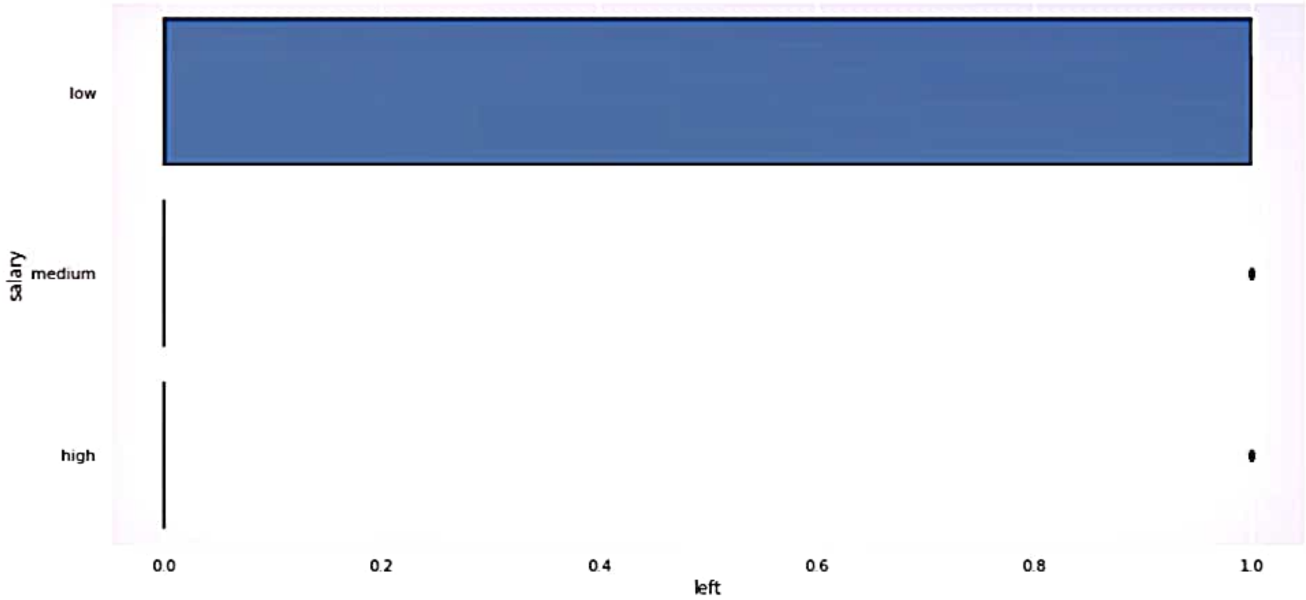
```
fig = plt.figure(figsize=(15,7))
sns.barplot(x='Department',y='left',da
ta=df)
```



hr Department employees has left the company most, next was accounting, technical, sales and support so on.

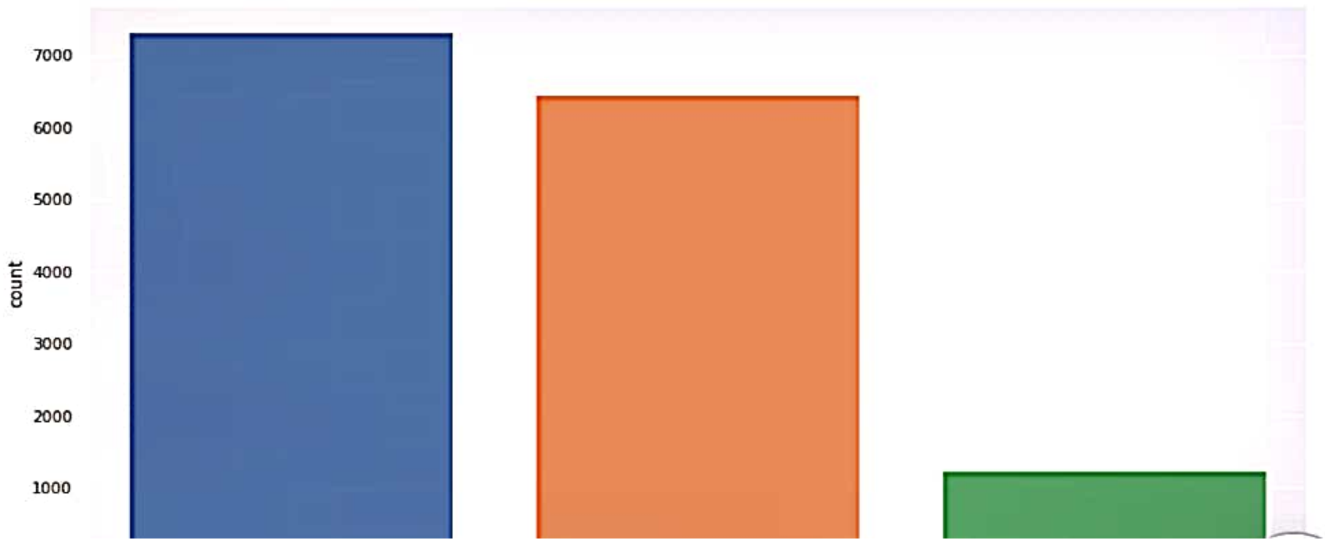
In [12]:

```
fig = plt.figure(figsize=(15,7))
sns.boxplot(x='left',y='salary',data=df)
plt.show()
```



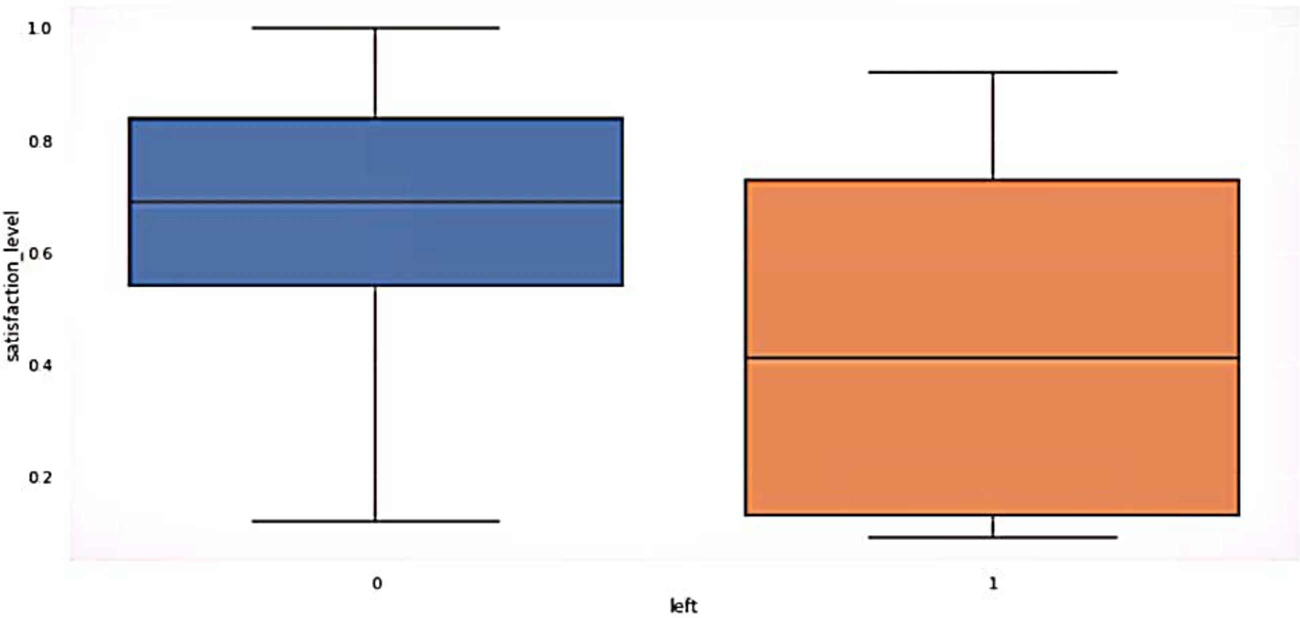
In [13]:

```
fig = plt.figure(figsize=(15,7))
sns.countplot(x='salary',data=df)
plt.show()
```



In [14]:

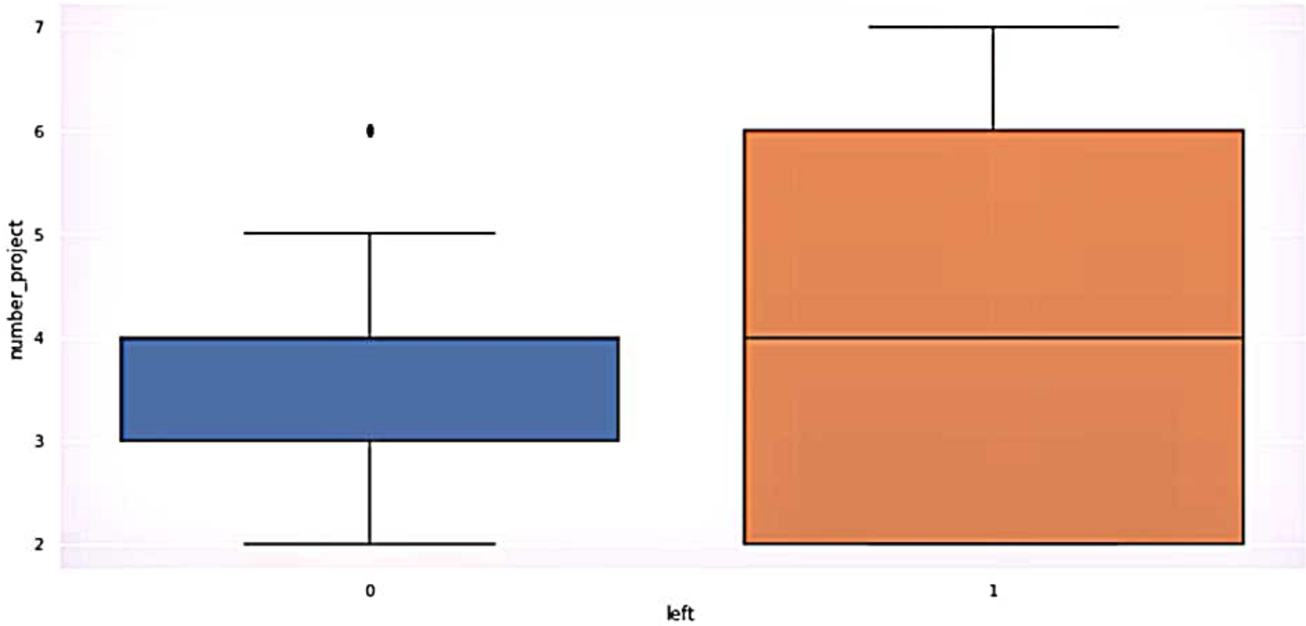
```
fig = plt.figure(figsize=(15,7))
sns.boxplot(x="left", y= "satisfaction_level", data=df)
plt.show()
```



In [15]:

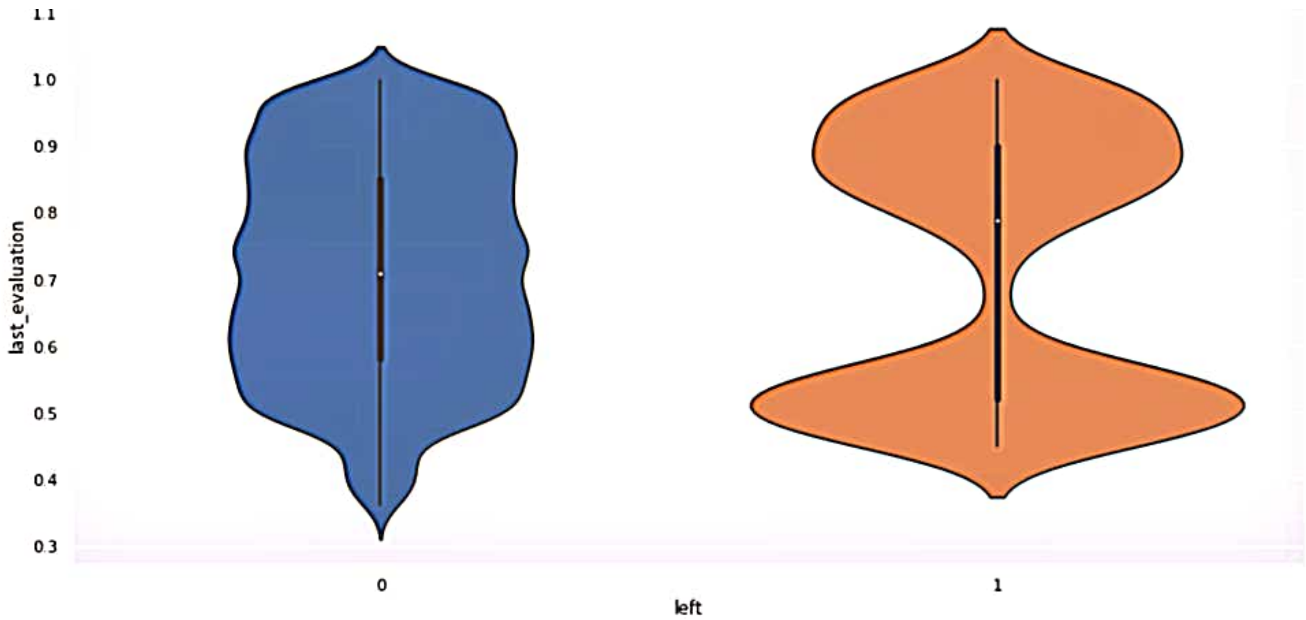
```
fig = plt.figure(figsize=(15,7))
sns.boxplot(x="left", y= "number_project", data=df)
```

```
fig = plt.figure(figsize=(15,7))
sns.boxplot(x="left", y= "number_proje
ct", data=df)
plt.show()
```



In [16]:

```
fig = plt.figure(figsize=(15,7))
sns.violinplot(x="left", y= "last_eval
uation", data=df)
plt.show()
```



Data Preprocessing

Convert the salary column to categorical

In [17]:

```
df.salary=df.salary.astype('category')
df.salary=df.salary.cat.reorder_catego
ries(['low', 'medium', 'high'])
df.salary = df.salary.cat.codes
```

In [18]:

```
# Get dummies and save them inside a ne
w DataFrame
departments = pd.get_dummies(df.Depart
ment)
# Take a quick look to the first 5 rows
of the new DataFrame called departments
print(departments.head(5))
```

```
   IT  RandD  accounting  hr  manageme
nt  marketing  product_mng  sales  \
0   0      0           0   0
0          0            0      1
1   0      0           0   0
0          0            0      1
2   0      0           0   0
0          0            0      1
3   0      0           0   0
0          0            0      1
4   0      0           0   0
0          0            0      1
```

	support	technical
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

In [19]:

```
departments = departments.drop("accoun
ting", axis=1)
df = df.drop("Department", axis=1)
df = df.join(departments)
df.head(5)
```

Out[19]:

	satisfaction_level	last_evaluation	number_project	ave
0	0.38	0.53	2	15
1	0.80	0.86	5	26
2	0.11	0.88	7	27
3	0.72	0.87	5	22

In [20]:

```
n_employees = len(df)

# Print the number of employees who left/stayed
print(df.left.value_counts())

# Print the percentage of employees who left/stayed
print(df.left.value_counts()/n_employees*100)
```

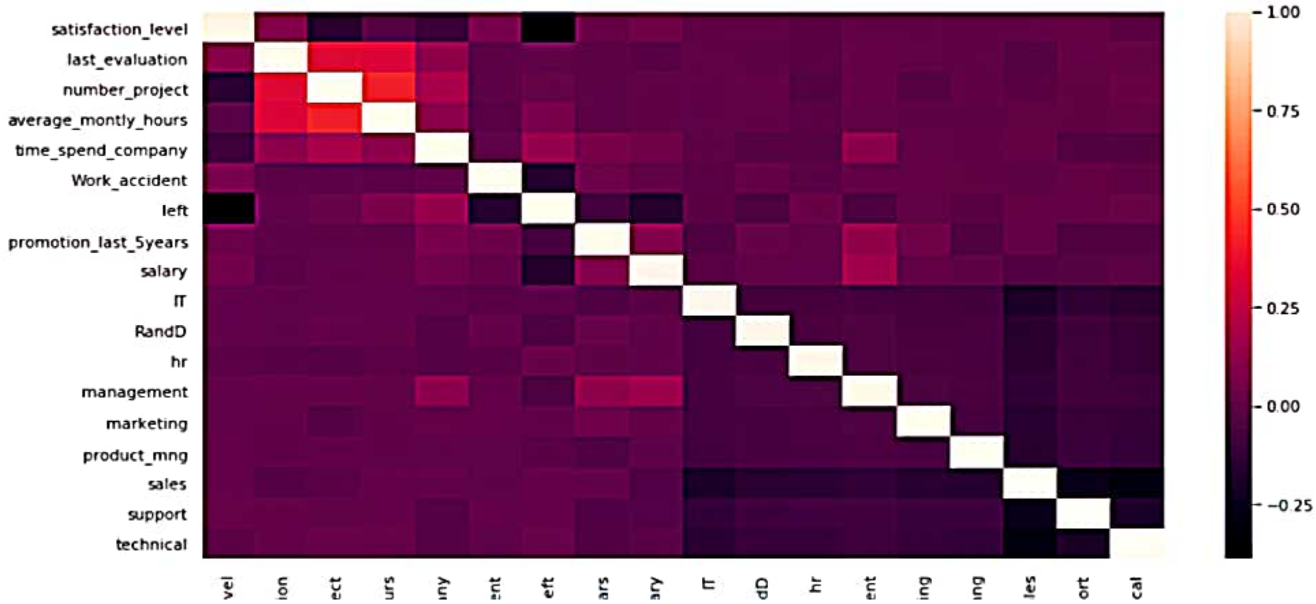
```
0    11428
1     3571
Name: left, dtype: int64
0    76.191746
1    23.808254
```

11,428 employees stayed, which accounts for about 76% of the total employee count. Similarly, 3,571 employees left, which accounts for about 24% of them

Correlation Matrix

In [21]:

```
fig = plt.figure(figsize=(15,7))
cor_mat=df.corr()
sns.heatmap(cor_mat)
plt.show()
```



In [22]:

```
target=df.left
features=df.drop('left',axis=1)
```

Splitting the dataset

will split both target and features into train and test sets with 75%/25% ratio, respectively

In [23]:

```
target_train, target_test, features_tr
ain, features_test = train_test_split
(target,features,test_size=0.25,random
```

In [24]:

```
model = DecisionTreeClassifier(random_
state=42)
model.fit(features_train, target_trai
n)
model.score(features_train,target_trai
n)*100
```

Out[24]:

```
100.0
```

In [25]:

```
#model.fit(features_test,target_test)
model.score(features_test,target_test)
*100
```

In [26]:

```
from sklearn import tree
from IPython.display import Image as P
Image
from subprocess import check_call
from PIL import Image, ImageDraw, Imag
eFont
import re
export_graphviz(model,"tree.dot")

check_call(['dot','-Tpng','tree.do
t','-o','tree.png'])

# Annotating chart with PIL
img = Image.open("tree.png")
draw = ImageDraw.Draw(img)
img.save('sample-out.png')
PImage("sample-out.png", height=2000,
width=1900)
```

As we saw above the accuracy is 100% on training and test set, model is overfitting, So fisrt check the option purne the tree, by setting the maximum depth

In [27]:

```
model_depth_5 = DecisionTreeClassifier
(max_depth=5, random_state=42)

# Fit the model
model_depth_5.fit(features_train,targe
t_train)

# Print the accuracy of the prediction
for the training set
print(model_depth_5.score(features_tra
in,target_train)*100)

# Print the accuracy of the prediction
for the test set
print(model_depth_5.score(features_tes
```

```
97.71535247577563
97.06666666666666
```

Second option to overfitting is limiting the sample size in a leaf(node)

In [28]:

```
model_sample_100 = DecisionTreeClassifier(min_samples_leaf=100, random_state=42)

# Fit the model
model_sample_100.fit(features_train,target_train)

# Print the accuracy of the prediction (in percentage points) for the training set
print(model_sample_100.score(features_train,target_train)*100)
```

```
# Print the accuracy of the prediction (in percentage points) for the test set
print(model_sample_100.score(features_test,target_test)*100)
```

```
96.57747355320473
96.13333333333334
```

Evaluating the model

In [29]:

```
#Predict whether employees will churn using the test set
prediction = model.predict(features_test)

# Calculate precision score by comparing target_test with the prediction
```

Out[29]:

```
0.9240641711229947
```

In [30]:

```
# Calculate recall score by comparing t
arget_test with the prediction
recall_score(target_test, prediction)
```

Out[30]:

```
0.9632107023411371
```

License