Problem : Apply Logistic Regression Model to analyze the data to know why the employees are leaving a company and when they will stay back in the company

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#load the data into the dataframe
df = pd.read_csv("HR_comma_sep.csv")
df
```

Out[2]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	time_spend_compa
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	
...	...	...	...	...	
14994	0.40	0.57	2	151	
14995	0.37	0.48	2	160	
14996	0.37	0.53	2	143	
14997	0.11	0.96	6	280	
14998	0.37	0.52	2	158	

14999 rows × 10 columns

Let us now do a EDA or data exploration and find the number of rows and columns who left(if left=1) or who retained(if left=0)

In [3]:
```
left = df[df.left==1]
left.shape
```

Out[3]: (3571, 10)

In [4]:
```
retained = df[df.left==0]
retained.shape
```

Out[4]: (11428, 10)

Now we find averages separately for people who left and were retained on all columns -

In [5]:
```
averages = df.groupby('left').mean(numeric_only=True)
averages
```

Out[5]:

left	satisfaction_level	last_evaluation	number_project	average_monthly_hours	time_spend_company
0	0.666810	0.715473	3.786664	199.060203	3.380032
1	0.440098	0.718113	3.855503	207.419210	3.876505

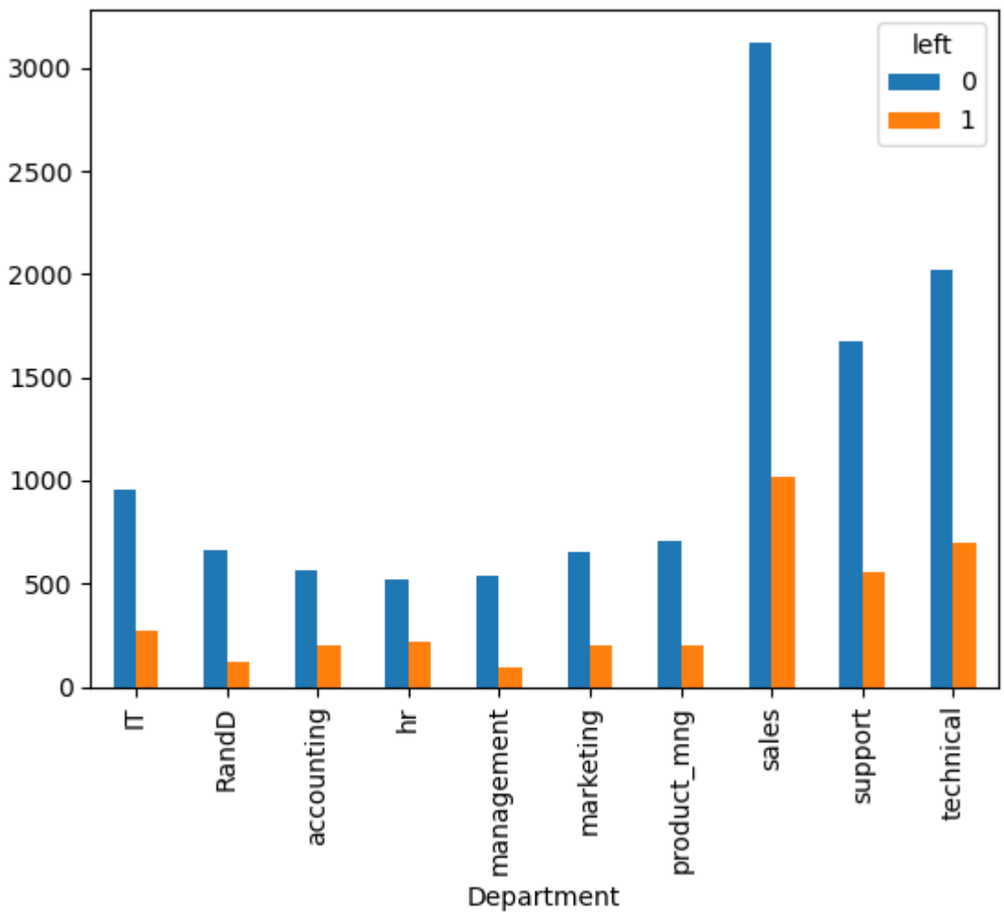
From the above table we see there is a **good difference** in the values of **satisfaction_level, average_monthly_hours, promotion_last_5_years**

Note : averages = df.groupby('left').mean() will lead to **TypeError(f"Could not convert string '{x}' to numeric")**, so we use a .mean(numeric_only=True) to filter out only the numeric_only values

Bar chart showing relation between **Department and left** -

In [6]:
```
pd.crosstab(df.Department, df.left).plot(kind='bar')
```

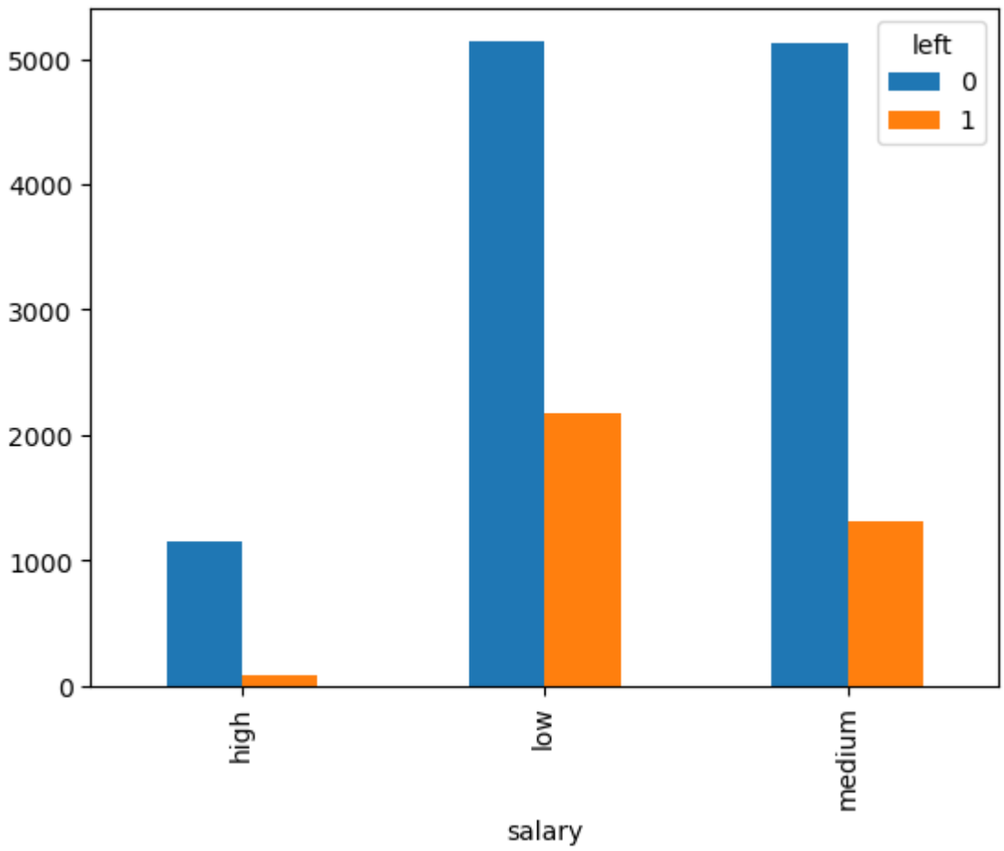
Out[6]:
```
<Axes: xlabel='Department'>
```



Bar chart to know impact of **employee salary** -

In [7]:
```
pd.crosstab(df.salary, df.left).plot(kind='bar')
```

Out[7]:
```
<Axes: xlabel='salary'>
```



This above, shows that people with high salary does not leave the company as low salaried people, so we also include salary among the list that affects the retention of an employee as it serves as a deciding factor -

In [8]:
```
subdf = df[['satisfaction_level', 'average_monthly_hours', 'promotion_last_5years',
subdf
```

Out[8]:

	satisfaction_level	average_monthly_hours	promotion_last_5years	salary
0	0.38	157	0	low
1	0.80	262	0	medium
2	0.11	272	0	medium
3	0.72	223	0	low
4	0.37	159	0	low
...	...	...	...	...
14994	0.40	151	0	low
14995	0.37	160	0	low
14996	0.37	143	0	low
14997	0.11	280	0	low
14998	0.37	158	0	low

14999 rows × 4 columns

Since salary is in text, we will convert it into numbers, we should split the salary variable into salary_high, salary_low, salary_medium. For this purpose we can use dummy variables -

To understand more on what dummy value variables are, refer to my previous notes that I have posted on LinkedIn under dummy_variables

In [9]:
```
salary_dummies = pd.get_dummies(subdf.salary, prefix='salary').astype(int)
salary_dummies
```

Out[9]:

	salary_high	salary_low	salary_medium
0	0	1	0
1	0	0	1
2	0	0	1
3	0	1	0
4	0	1	0
...	...	...	...
14994	0	1	0
14995	0	1	0
14996	0	1	0
14997	0	1	0
14998	0	1	0

14999 rows × 3 columns

Add the dummy_variables to the existing dataframe -

In [10]:
```
df_withdummies = pd.concat([subdf, salary_dummies], axis='columns')
df_withdummies
```

Out[10]:

	satisfaction_level	average_monthly_hours	promotion_last_5years	salary	salary_high	salary
0	0.38	157	0	low	0	
1	0.80	262	0	medium	0	
2	0.11	272	0	medium	0	
3	0.72	223	0	low	0	
4	0.37	159	0	low	0	
...	...	...	...	...	...	
14994	0.40	151	0	low	0	
14995	0.37	160	0	low	0	
14996	0.37	143	0	low	0	
14997	0.11	280	0	low	0	
14998	0.37	158	0	low	0	

14999 rows × 7 columns

To avoid dummy variable trap let us **delete salary_medium column** also let us **delete the salary column** as is is already there in the dummies -

In [11]:
```
df_withdummies.drop(['salary', 'salary_medium'], axis='columns', inplace=True)
df_withdummies
```

Out[11]:

	satisfaction_level	average_monthly_hours	promotion_last_5years	salary_high	salary_low
0	0.38	157	0	0	1
1	0.80	262	0	0	0
2	0.11	272	0	0	0
3	0.72	223	0	0	1
4	0.37	159	0	0	1
...	...	...	...	...	...
14994	0.40	151	0	0	1
14995	0.37	160	0	0	1
14996	0.37	143	0	0	1
14997	0.11	280	0	0	1
14998	0.37	158	0	0	1

14999 rows × 5 columns

Take independent features (x) and target feature (y) -

In [12]:
```
x = df_withdummies
x
```

Out[12]:

	satisfaction_level	average_monthly_hours	promotion_last_5years	salary_high	salary_low
0	0.38	157	0	0	1
1	0.80	262	0	0	0
2	0.11	272	0	0	0
3	0.72	223	0	0	1
4	0.37	159	0	0	1
...	...	...	...	...	...
14994	0.40	151	0	0	1
14995	0.37	160	0	0	1
14996	0.37	143	0	0	1
14997	0.11	280	0	0	1
14998	0.37	158	0	0	1

14999 rows × 5 columns

In [13]:
```
y = df.left
y
```

Out[13]:
```
0        1
1        1
2        1
3        1
4        1
       ..
14994    1
14995    1
14996    1
14997    1
14998    1
Name: left, Length: 14999, dtype: int64
```

Let us split the data into **80% train data** and remaining **20% test data** -

In [14]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y, train_size=0.8)
```

Apply **Logistic Regression** on the train data -

In [15]:
```
model = LogisticRegression()
model.fit(x_train, y_train)
```

Out[15]:
```
▾ LogisticRegression
LogisticRegression()
```

Accuracy of our model -

In [16]:
```
model.score(x_test, y_test)
```

Out[16]:
```
0.7713333333333333
```

Prediction 1 : Find if employees will stay back or leave in the company when satisfaction level is 0.11, average monthly hours is 286 and no promotion in last 5 years and medium salary

In [17]:
```
inputs = [[0.11,286,0,0,0]]
model.predict(inputs) #array[1] will leave
```

Out[17]:
```
array([1])
```

Prediction 2: Find if employees will leave or stay back in the company if satisfaction level is 0.48, average monthly hours is 228, no promotion in last 5 years and low salary

In [18]:
```
inputs = [[0.48, 228, 0, 0, 1]]
model.predict(inputs)
```

Out[18]:
```
array([0])
```