THE MACHINE LEARNING JOURNEY

CHAPTER-05

LOGISTIC REGRESSION

✓ Part-1 Part-2

All Linear Regression model like Simple, Multiple or Polynomial, are useful when we want to predict certain values from a continuos range of values.

In some cases, we want to predict values which are called 'categorical', which means the values representing limited no. of possibilities. eg -

1. An email is spam/not-spam.
2. Marital Status (single/married/ widowed/divorced)

When we want to predict categorical data, we can use Logistic Regression Model.

@Shreyan

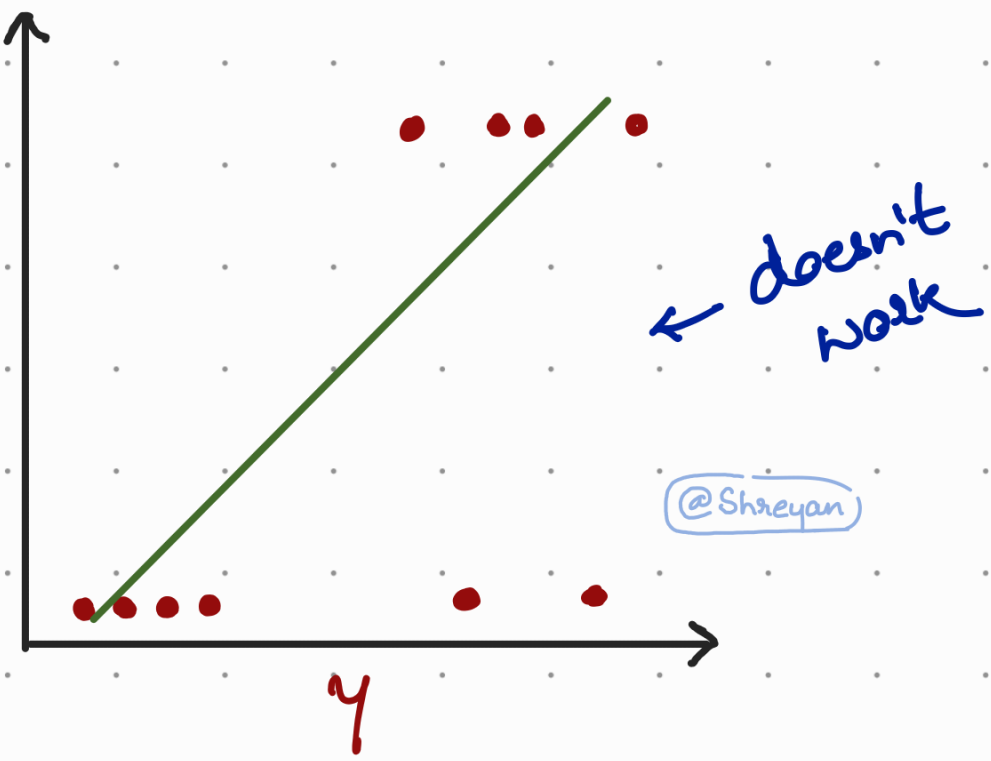
When the categorical data shows -

1. 2 possibilities - it is called 'Binary Classification'.
2. More than 2 possibilities - it is called 'Multiclass Classification.'

Binary Classification using LR.

In binary classification, we have to predict one of the 2 classes or groups of data.

We will understand this concept with 2 problems, but before that let us understand something else -



Suppose we have a plot like this, can we / do we find a linear relationship here? NOPE.

To connect such type of points, we can use an 'S'-shaped curve. This curve line fits most of the data points and becomes best fit.

We can use Sigmoid or Logit function given by the formula -

$$\text{Sigmoid}(t) = \frac{1}{1+e^{-t}}$$

e → Eulers number ≈ 2.71828

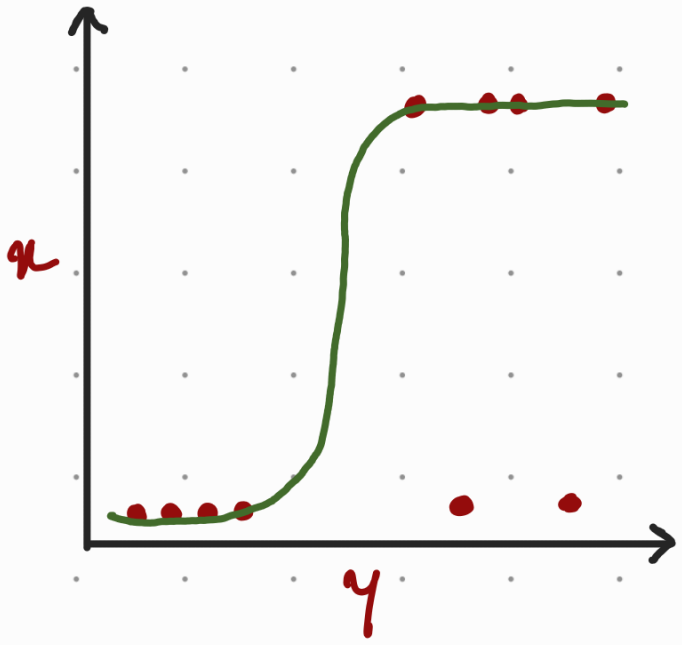
So –

$$t = \frac{1}{1+(\cdots)} = \frac{1}{(\text{value} > 1)}$$

$$\therefore \boxed{t \leq 1}$$

So, the purpose of Sigmoid function is to convert the input into a range from 0 to 1. It shows a smooth curve when plotted.

@Shreyan



In Logistic Regression, we are converting the straight line of Linear Regression into S-shaped line.

Lin. Reg. formula → $\boxed{y = mx + b}$

∴ Logistic Reg. formula → $\boxed{y = \frac{1}{1+e^{-(mx+b)}}}$

Problem: We are given data like age and bought_insurance. Apply Logistic Regression Model and predict whether a person takes insurance or not based on his age.

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LogisticRegression
```

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

Out[2]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1
10	18	0
11	28	0
12	27	0
13	29	0
14	49	1

In [3]:
```
#retreive the data
x = df.iloc[:, :-1].values #retreive only 0th column
x
```

Out[3]:
```
array([[22],
       [25],
       [47],
       [52],
       [46],
       [56],
       [55],
       [60],
       [62],
       [61],
       [18],
       [28],
       [27],
       [29],
       [49]])
```

Note: We need to get a 2D column afterwards, retreiving it as [:, :0] will keep it 1D and it wont work. So we use :-1 to get a column data to multiply it with y and get 2D

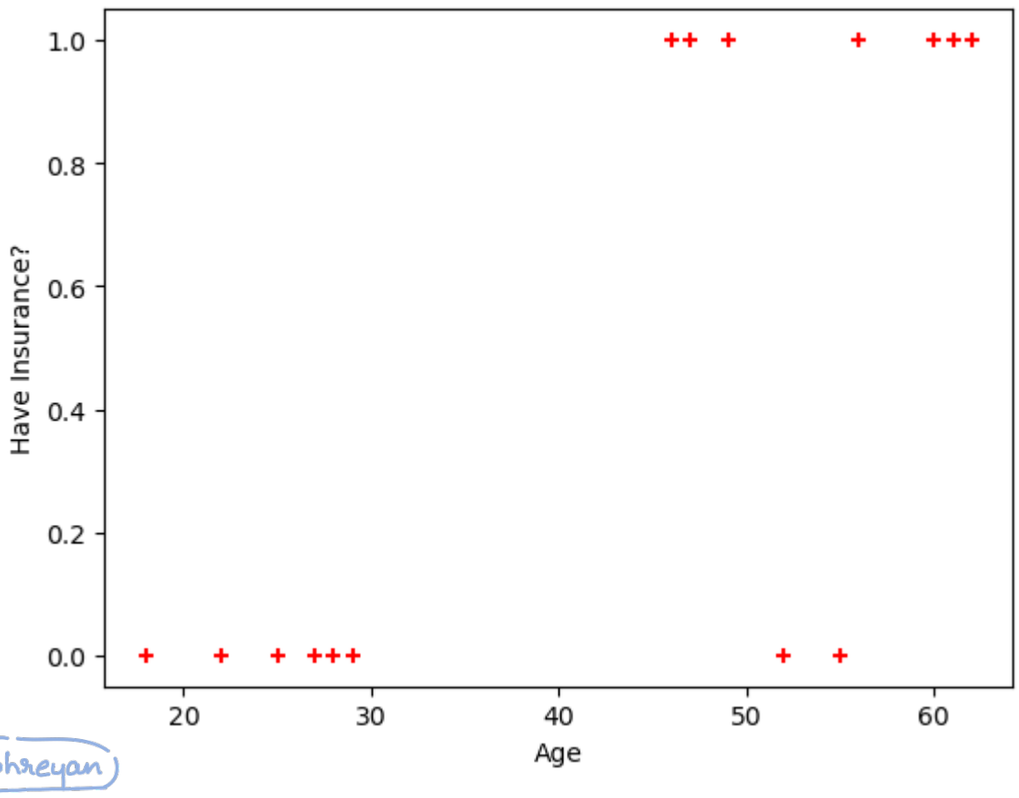
In [4]:
```
y = df.iloc[:, 1].values #retreive 1st column
y
```

Out[4]:
```
array([0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1])
```

Display the scatter plot to know how the datapoints are aligned -

In [5]:
```
plt.xlabel('Age')
plt.ylabel('Have Insurance?')
plt.scatter(x,y, marker='+', color='red')
```

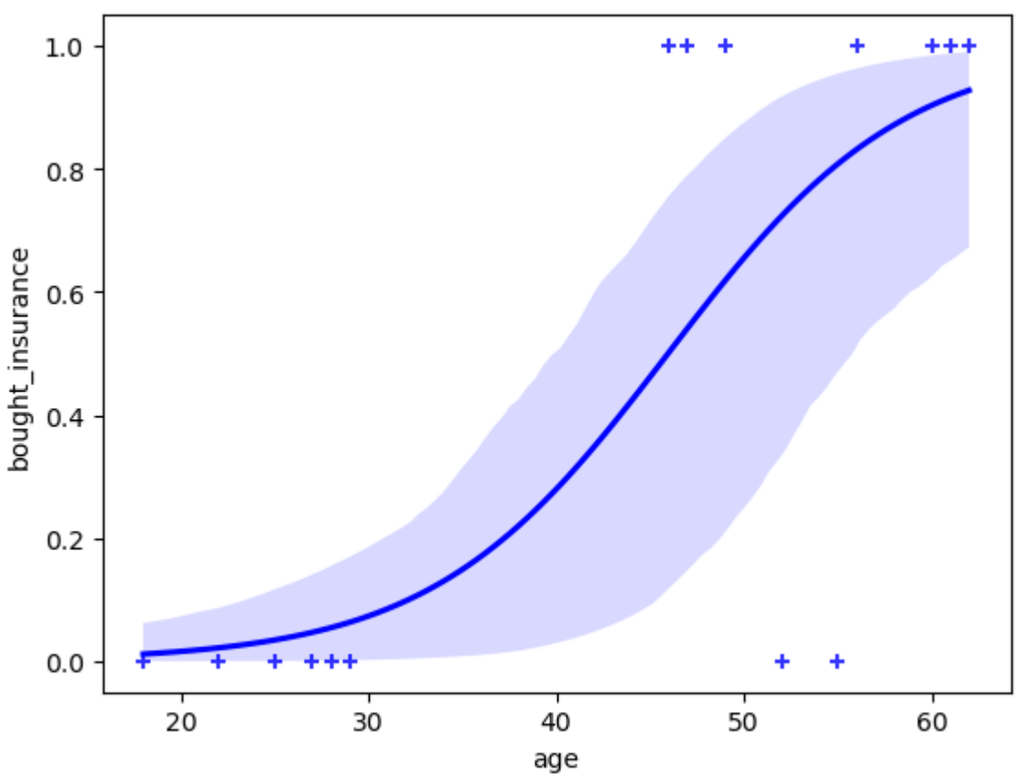
Out[5]:
```
<matplotlib.collections.PathCollection at 0x7f653f0a12d0>
```



For the above data we cannot use **Linear regression** as we can see from the graph itself that there is a linear relationship does not exist here, so we use a **Logistic Regression Model** for this -

In [6]:
```
plt.xlabel('Age')
plt.ylabel('Probability of Buying Insurance')
sns.regplot(data=df, x='age', y='bought_insurance', logistic=True, marker='+', colo
```

Out[6]:
```
<Axes: xlabel='age', ylabel='bought_insurance'>
```



You might get a RuntimeWarning: overflow encountered in [exp t = np.exp(-z)], but we can ignore it using the filter warnings that we imported from the warnings library

In [7]:
```
#create logistic regression model
model = LogisticRegression()
```

In [8]:
```
#train the model
model.fit(x,y)
```

Out[8]:
```
▾ LogisticRegression

LogisticRegression()
```

In [9]:
```
#find the accuracy of the model
model.score(x,y)
```

Out[9]: 0.8666666666666667 → 86.66%

Prediction 1: Let us now predict if a 66 years aged person will buy insurance or not -

In [10]:
```
model.predict([[56]]) #array([[1]]) means Yes
```

Out[10]: array([1])

Prediction 2: Let us now predict if a 23 years old person will buy insurance or not -

In [11]:
```
#predict if 23 years aged person will buy insurance or not
model.predict([[23]]) #array([[0]]) means No
```

Out[11]: array([0])

A 66 yrs old person is thus more likely to buy insurance than a 23 yrs old person.

Now, we will solve another problem using Logistic Regression Model. Let us take an employee retention data set to check under what factors will an employee stay or leave a company.

@Shreyan

↓

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	

size

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)

@Shreyan

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

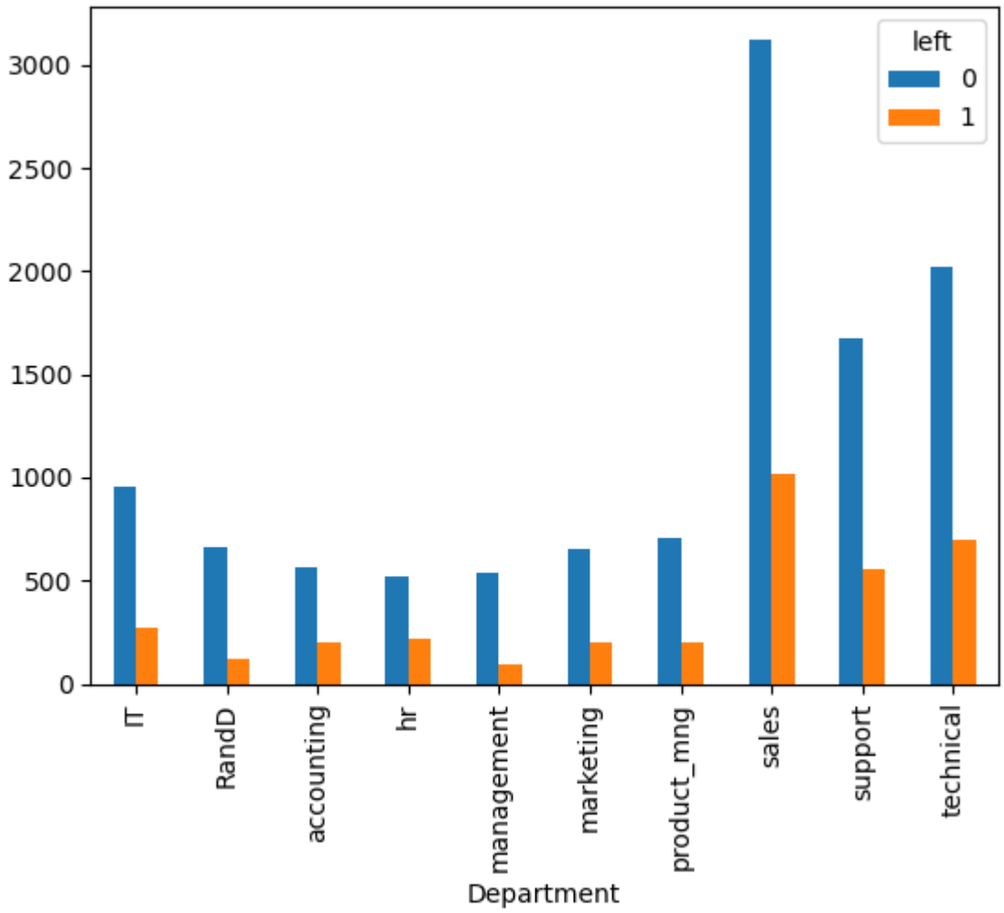
From the above table we see there is a **good difference** in the values of **satisfaction_level**, **average_monthly_hours**, **promotion_last_5_years**

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'>
```

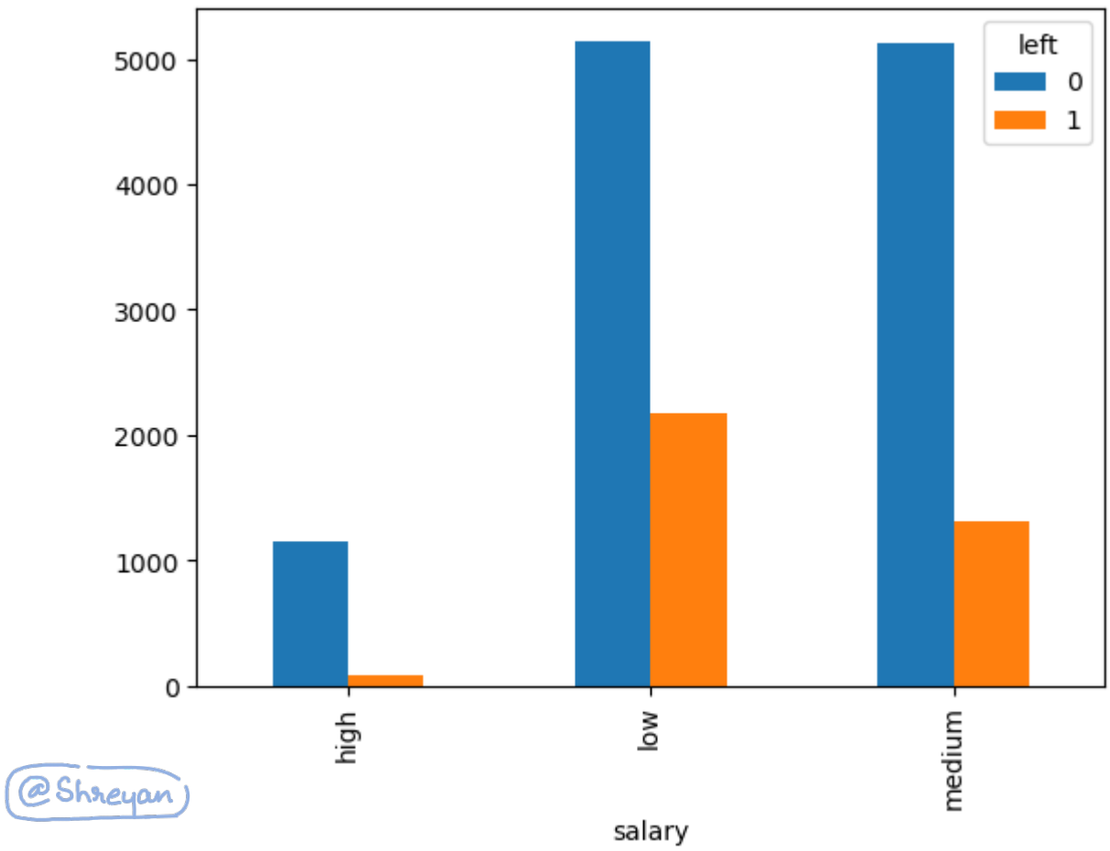


Bar chart to know impact of **employee salary** -

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

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

@Shreyan



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
```
→ 77.13%

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])
```
→ leave.

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])
```
→ retained.

@Shreyan

SHREYAN BASU RAY

OPENSOURCE MAINTAINER @ SAGE.AI

SUPPORT ME → github.com/sponsors/Shreyan1