CAPSTONE PROJECT

DATA SCIENCE AND MACHINE LEARNING

GROUP # 1006

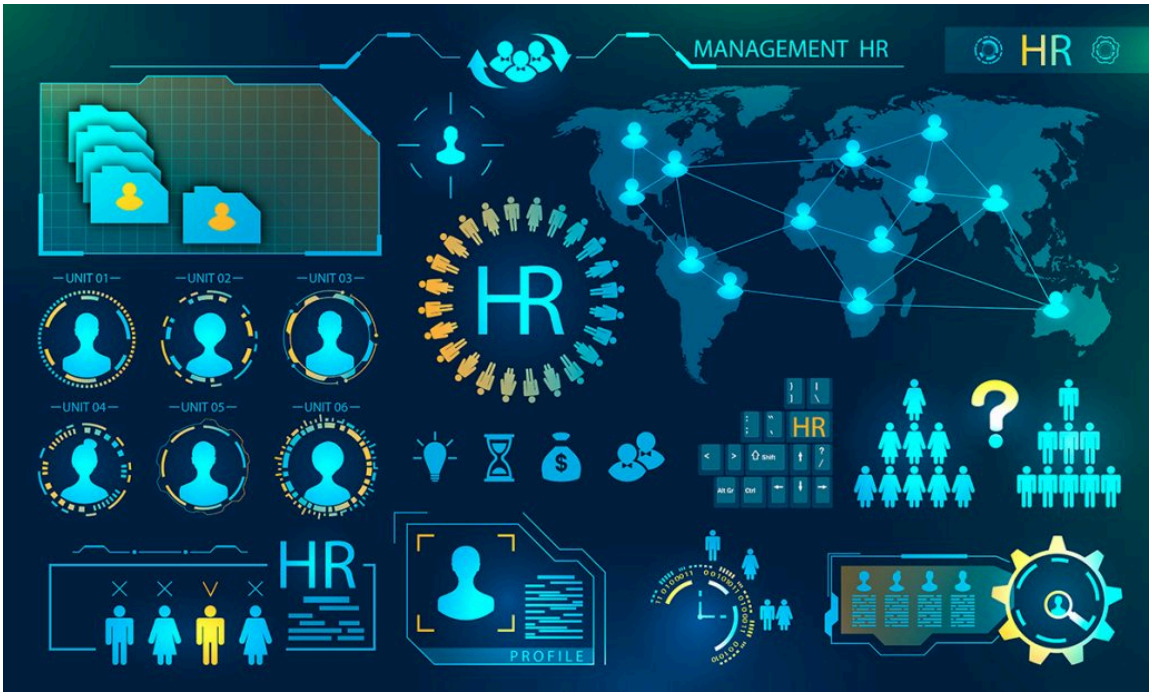


Table of Contents

Introduction

Employee Retention

Employee retention is the biggest challenge of many organisations. Employee retention is a phenomenon where employees choose to stay on with their current company and don't actively seek other job prospects. Employees leave the organisation for a variety of reasons. The HR department of a software company wants to ensure that its employees do not quit their jobs.

Current Practice

Once an employee leaves, he or she is taken an interview with the name "exit interview" and shares reasons for leaving. The HR Department then tries and learns insights from the interview and makes changes accordingly.

This suffers from the following problems:

- This approach is that it's too haphazard. The quality of insight gained from an interview depends heavily on the skill of the interviewer.
- The second problem is these insights can't be aggregated and interlaced across all employees who have left.
- The third is that it is too late by the time the proposed policy changes take effect.

Problem Statement

Our Client is HR department of a software company. They are looking for a solution to retain their employees. For that,

- They want to try a new initiative to retain employees.
- The idea is to use **data to predict whether an employee is likely to leave**.
- Once these employees are identified, HR can be more proactive in reaching out to them before it's too late.
- They only want to deal with the data that is related to **permanent employees.**
- HR Department has given datasets of past employees and their status (still employed or already left).
- As a data science consultant, our task is **to build a classification model to predict whether an employee will stay or leave**.
- Because there is no machine learning model for this problem in the company, there is no quantifiable win condition. **A best possible model** need to be built.

Data Description

The Business Intelligence Analysts of the Company provided you three datasets that contain information about past employees and their status (still employed or already left).

department_data

This dataset contains information about each department. The schema of the dataset is as follows:

- dept_id – Unique Department Code
- dept_name – Name of the Department
- dept_head – Name of the Head of the Department

employee_details_data

This dataset consists of Employee ID, their Age, Gender and Marital Status. The schema of this dataset is as follows:

- employee_id – Unique ID Number for each employee
- age – Age of the employee
- gender – Gender of the employee
- marital_status – Marital Status of the employee

employee_data

This dataset consists of each employee's Administrative Information, Workload Information, Mutual Evaluation Information and Status.

Target variable

- status – Current employment status (Employed / Left)

Administrative information

- department – Department to which the employees belong(ed) to
- salary – Salary level with respect to rest of their department
- tenure – Number of years at the company
- recently_promoted – Was the employee promoted in the last 3 years?
- employee_id – Unique ID Number for each employee

Workload information

- n_projects – Number of projects employee has worked on
- avg_monthly_hrs – Average number of hours worked per month

Mutual evaluation information

- satisfaction – Score for employee's satisfaction with the company (higher is better)
- last_evaluation – Score for most recent evaluation of employee (higher is better)
- filed_complaint – Has the employee filed a formal complaint in the last 3 years?

In [1]:
```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

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

Datasets Download

In [2]:
```
import mysql.connector

db = mysql.connector.connect(
   host="cpanel.insaid.co",
   user="student",
   passwd="student",
   database="Capstone2"
)
```

In [3]:
```
import pandas as pd
department_data=pd.read_sql_query("SELECT * FROM department_data",db)
employee_details_data=pd.read_sql_query("SELECT * FROM employee_details_data",db)
employee_data=pd.read_sql_query("SELECT * FROM employee_data",db)
```

Data Examination

Department_data

In [4]:
```
department_data.shape
```

Out[4]:
```
(11, 3)
```

In [5]:
```
department_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   dept_id    11 non-null     object
 1   dept_name  11 non-null     object
 2   dept_head  11 non-null     object
dtypes: object(3)
memory usage: 392.0+ bytes
```

In [6]:
```
department_data.head()
```

Out[6]:

	dept_id	dept_name	dept_head
0	D00-IT	IT	Henry Adey
1	D00-SS	Sales	Edward J Bayley
2	D00-TP	Temp	Micheal Zachrey
3	D00-ENG	Engineering	Sushant Raghunathan K
4	D00-SP	Support	Amelia Westray

In [7]:
```
department_data.describe()
```

Out[7]:

	dept_id	dept_name	dept_head
count	11	11	11
unique	11	11	11
top	D00-IT	IT	Henry Adey
freq	1	1	1

In [8]:
```
# checking for duplicate records
department_data.duplicated().value_counts()
```

Out[8]:
```
False    11
dtype: int64
```

Observations on department_data

department_data consists of information about each department.

Records	Features	Dataset Size
11	3	392.0 bytes

There are **11** unique departments in the department_data table.
There are **no NULL values** in the department_data table.

Employee_data

In [9]:
```
employee_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14150 entries, 0 to 14149
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   avg_monthly_hrs    14150 non-null  float64
 1   department         13443 non-null  object 
 2   filed_complaint    2046 non-null   float64
 3   last_evaluation    12663 non-null  float64
 4   n_projects         14150 non-null  int64  
 5   recently_promoted  297 non-null    float64
 6   salary             14150 non-null  object 
 7   satisfaction       14000 non-null  float64
 8   status             14150 non-null  object 
 9   tenure             14000 non-null  float64
 10  employee_id        14150 non-null  int64  
dtypes: float64(6), int64(2), object(3)
memory usage: 1.2+ MB
```

employee_data consists of information about each employee.

Records	Features	Dataset Size
14150	10	1.2+ MB

employee_data table has float64(6), int64(2), object(3). There are **NULL values** in the employee_data table.

In [10]:
```
employee_data.isna().sum()
```

Out[10]:
```
avg_monthly_hrs          0
department             707
filed_complaint      12104
last_evaluation       1487
n_projects               0
recently_promoted    13853
salary                   0
satisfaction           150
status                   0
tenure                 150
employee_id              0
dtype: int64
```

- NULL values details in employee_data table:
  - There are 707 Null values in department column.
  - There are 12104 Null values in filed_complaint column.
  - There are 1487 Null values in last_evaluation column.
  - There are 13853 Null values in recently_promoted column.
  - There are 150 Null values in satisfaction .
  - There are 150 Null values in tenure column.

In [11]:
```
employee_data.nunique()
```

Out[11]:
```
avg_monthly_hrs        249
department              12
filed_complaint          1
last_evaluation      12188
n_projects               7
recently_promoted        1
salary                   3
satisfaction         13497
status                   2
tenure                   8
employee_id          14117
dtype: int64
```

In [12]:
```
employee_data['department'].unique()
```

Out[12]:
```
array([None, 'D00-SS', 'D00-SP', 'D00-MT', 'D00-PD', 'D00-IT', 'D00-AD',
       'D00-MN', 'D00-ENG', 'D00-PR', 'D00-TP', 'D00-FN', '-IT'],
      dtype=object)
```

In [13]:
```
employee_data['department'].value_counts()
```

Out[13]:
```
D00-SS     3905
D00-ENG    2575
D00-SP     2113
D00-IT     1157
D00-PD      855
D00-MT      815
D00-FN      725
D00-MN      593
-IT         207
D00-AD      175
D00-PR      173
D00-TP      150
Name: department, dtype: int64
```

In [14]:
```
employee_data.department.mode()
```

Out[14]:
```
0    D00-SS
Name: department, dtype: object
```

In [15]:
```
employee_data['filed_complaint'].unique()
```

Out[15]:
```
array([nan,  1.])
```

In [16]:
```
employee_data['filed_complaint'].value_counts()
```

Out[16]:
```
1.0    2046
Name: filed_complaint, dtype: int64
```

In [17]:
```
employee_data['recently_promoted'].unique()
```

Out[17]:
```
array([nan,  1.])
```

In [18]:
```
employee_data['tenure'].unique()
```

Out[18]:
```
array([ 4.,  3.,  5.,  6., nan,  2., 10.,  7.,  8.])
```

In [19]:
```
employee_data['last_evaluation'].unique()
```

Out[19]:
```
array([0.866838, 0.555718, 0.474082, ..., 0.643553, 0.836603, 0.907277])
```

In [20]:
```
employee_data['satisfaction'].unique()
```

Out[20]:
```
array([0.134415, 0.511041, 0.405101, ..., 0.944942, 0.740136, 0.506658])
```

In [21]:
```
employee_data.head()
```

Out[21]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promoted	
0	246.0	None	NaN	0.866838	6	NaN	r
1	134.0	None	NaN	0.555718	2	NaN	
2	156.0	D00-SS	1.0	0.474082	2	NaN	r
3	256.0	D00-SP	NaN	0.961360	6	NaN	
4	146.0	D00-SS	NaN	0.507349	2	NaN	r

In [22]:
```
employee_data.describe()
```

Out[22]:

	avg_monthly_hrs	filed_complaint	last_evaluation	n_projects	recently_promoted	satisfa
count	14150.000000	2046.0	12663.000000	14150.000000	297.0	14000.0
mean	199.994346	1.0	0.718399	3.778304	1.0	0.6
std	50.833697	0.0	0.173108	1.250162	0.0	0.2
min	49.000000	1.0	0.316175	1.000000	1.0	0.0
25%	155.000000	1.0	0.563711	3.000000	1.0	0.4
50%	199.000000	1.0	0.724731	4.000000	1.0	0.6
75%	245.000000	1.0	0.871409	5.000000	1.0	0.8
max	310.000000	1.0	1.000000	7.000000	1.0	1.0

There are some employee_id with value as 0

In [23]:
```
employee_data.describe(include=object)
```

Out[23]:

	department	salary	status
count	13443	14150	14150
unique	12	3	2
top	D00-SS	low	Employed
freq	3905	6906	10784

In [24]:
```
employee_data[employee_data["employee_id"]==0]
```

Out[24]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promot
34	284.0	D00-SS	NaN	0.852702	6	Na
304	264.0	D00-ENG	NaN	0.774554	6	Na
1234	129.0	D00-SP	NaN	0.402660	2	Na
11304	167.0	None	NaN	0.880053	5	Na
12304	259.0	D00-ENG	NaN	0.505883	5	Na

- There are 5 records with employee_id value as 0. These records need to be dropped

In [25]:
```
# checking for duplicate records
employee_data.duplicated().value_counts()
```

Out[25]:
```
False    14121
True        29
dtype: int64
```

- There are 29 duplicate rows in employee_data table. These records need to be dropped.

In [26]:
```
# displaying the 29 duplicate records
employee_data[employee_data.duplicated()]
```

Out[26]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promot
14121	265.0	D00-MN	1.0	0.825672	3	Na
14122	282.0	D00-SS	NaN	0.875427	6	Na
14123	206.0	D00-SS	NaN	1.000000	6	Na
14124	161.0	D00-IT	NaN	0.715893	6	Na
14125	163.0	D00-SP	NaN	0.486176	3	Na
14126	208.0	D00-FN	NaN	0.699756	4	Na
14127	152.0	D00-MT	NaN	0.510016	3	Na
14128	307.0	D00-IT	NaN	0.905708	7	Na
14129	212.0	D00-PD	NaN	0.773018	4	Na
14130	148.0	D00-MN	NaN	0.577112	3	Na
14131	184.0	D00-MT	NaN	0.639866	5	Na
14132	132.0	D00-PD	NaN	1.000000	4	Na
14133	168.0	D00-SP	NaN	0.747792	2	Na
14134	161.0	D00-SS	NaN	0.483513	2	Na
14135	136.0	D00-SS	NaN	0.645563	4	Na
14136	261.0	D00-IT	1.0	0.465430	5	Na
14137	181.0	D00-SS	1.0	0.703796	3	Na
14138	142.0	D00-SP	1.0	0.400566	4	Na
14139	135.0	D00-SP	NaN	0.504764	2	Na
14140	133.0	D00-SS	NaN	0.986741	3	Na
14141	224.0	D00-FN	NaN	0.877647	3	Na
14142	233.0	D00-IT	NaN	0.940863	4	Na
14143	141.0	D00-FN	NaN	0.801055	3	Na
14144	238.0	D00-SS	NaN	0.958633	3	Na
14145	245.0	D00-SS	NaN	0.850785	6	Na
14146	192.0	D00-SS	NaN	0.951901	3	Na
14147	175.0	D00-MN	NaN	0.854538	4	Na
14148	268.0	D00-MT	NaN	0.900887	5	Na
14149	268.0	D00-IT	1.0	0.897098	2	Na

Observations on employee_data

employee_data consists of information about each employee.

Records	Features	Dataset Size
14150	10	1.2+ MB

employee_data table has float64(6), int64(2), object(3). There are **NULL values** in the employee_data table.

- NULL values details in employee_data table:
  - There are 707 Null values in department column.
  - There are 12104 Null values in filed_complaint column.
  - There are 1487 Null values in last_evaluation column.
  - There are 13853 Null values in recently_promoted column.
  - There are 150 Null values in satisfaction .
  - There are 150 Null values in tenure column.

Observations on Missing data and Erroneous data enty

- There are 11 departments. D00-SS, D00-ENG, D00-SP, D00-IT, D00-PD, D00-MT, D00-FN, D00-MN, -IT, D00-AD, D00-PR, D00-TP. D00-IT is wrongly typed as -IT for 207 rows. This need to corrected. 707 missing values of this categorical column can be imputed with mode value. ie D00-SS
- filed_complaint has two values 1.0 and nan. If 1.0 is taken for yes, remaining rows with nan can be given a value 0 to show that they did not file complaint
- recently_promoted has two values 1.0 and nan. If 1.0 is taken for yes, remaining rows with nan can be given a value 0 to show that they were not recently_promoted
- missing values in last_evaluation and satisfaction columns can be replaced with mean value of the column. It is obseverd that mean and median values of both colums are very close showing that there are no outliers.

Employee_details_data

In [27]:
```
employee_details_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14245 entries, 0 to 14244
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   employee_id     14245 non-null  int64 
 1   age             14245 non-null  int64 
 2   gender          14245 non-null  object
 3   marital_status  14245 non-null  object
dtypes: int64(2), object(2)
memory usage: 445.3+ KB
```

In [28]:
```
employee_details_data.describe()
```

Out[28]:

	employee_id	age
count	14245.000000	14245.000000
mean	112123.050544	32.889926
std	8500.457343	9.970834
min	100101.000000	22.000000
25%	105775.000000	24.000000
50%	111298.000000	29.000000
75%	116658.000000	41.000000
max	148988.000000	57.000000

In [29]:
```
employee_details_data.isna().sum()
```

Out[29]:
```
employee_id       0
age               0
gender            0
marital_status    0
dtype: int64
```

In [30]:
```
employee_details_data.sample(5)
```

Out[30]:

	employee_id	age	gender	marital_status
8225	108479	52	Male	Married
3833	126639	23	Female	Unmarried
10555	110548	22	Male	Unmarried
5843	144197	46	Male	Married
11639	109714	26	Male	Unmarried

In [31]:
```
employee_details_data.nunique()
```

Out[31]:
```
employee_id       14245
age                  36
gender                2
marital_status        2
dtype: int64
```

In [32]:
```
employee_details_data["gender"].unique()
```

Out[32]:
```
array(['Male', 'Female'], dtype=object)
```

In [33]:
```
employee_details_data["marital_status"].unique()
```

Out[33]:
```
array(['Married', 'Unmarried'], dtype=object)
```

In [34]:
```
employee_details_data["age"].unique()
```

Out[34]:
```
array([43, 24, 22, 36, 38, 51, 54, 49, 37, 27, 47, 28, 53, 39, 35, 42, 40,
       23, 45, 25, 30, 34, 26, 44, 52, 31, 32, 33, 29, 41, 46, 48, 57, 50,
       55, 56], dtype=int64)
```

In [35]:
```
# Checking for duplicate rows
employee_details_data.duplicated().value_counts()
```

Out[35]:
```
False    14245
dtype: int64
```

observations on employee_details_data table

There are 14245 reccords and 4 features.
There are no **NULL VALUES** in the table
gender column has two values: Male, Female
marital_status has two values: Married and Unmarried

Data Pre-processing and Cleaning

In [36]:
```
employee_data.shape
```

Out[36]:
```
(14150, 11)
```

In [37]:
```
# removing 29 duplicate rows from employee_data table
employee_data=employee_data.drop_duplicates()
```

In [38]:
```
employee_data.shape
```

Out[38]:
```
(14121, 11)
```

In [39]:
```
# In the employee_data table, department column has 207 records with '-IT', which r
employee_data.replace({'-IT'},{'D00-IT'},inplace =True)
```

In [40]:
```
employee_data['department'].value_counts()
```

Out[40]:
```
D00-SS     3896
D00-ENG    2575
D00-SP     2109
D00-IT     1359
D00-PD      853
D00-MT      812
D00-FN      722
D00-MN      590
D00-AD      175
D00-PR      173
D00-TP      150
Name: department, dtype: int64
```

In [41]:
```
employee_data.shape
```

Out[41]:
```
(14121, 11)
```

In [42]:
```
# Dropping the 5 records where employee_id is 0
employee_data=employee_data.drop(employee_data[employee_data['employee_id']==0].inc
```

In [43]:
```
employee_data.shape
```

Out[43]:
```
(14116, 11)
```

Handling Missing Values

In [44]:
```
employee_data['filed_complaint'].unique()
```

Out[44]:
```
array([nan,  1.])
```

In [45]:
```
# Replacing the NaN value with 0 in filed_complaint column
employee_data['filed_complaint']=employee_data['filed_complaint'].replace(np.NaN,0)
```

In [46]:
```
employee_data['filed_complaint'].unique()
```

Out[46]:
```
array([0., 1.])
```

In [47]:
```
employee_data["recently_promoted"].unique()
```

Out[47]:
```
array([nan,  1.])
```

In [48]:
```
# Replacing the NaN value with 0 in recently_promoted column
employee_data['recently_promoted']=employee_data['recently_promoted'].replace(np.Na
```

In [49]:
```
employee_data["recently_promoted"].unique()
```

Out[49]:
```
array([0., 1.])
```

In [50]:
```
employee_data.describe()
```

Out[50]:

	avg_monthly_hrs	filed_complaint	last_evaluation	n_projects	recently_promoted	satisfa
count	14116.000000	14116.000000	12629.000000	14116.000000	14116.000000	13966.0
mean	199.992632	0.144588	0.718322	3.777770	0.021040	0.6
std	50.826952	0.351697	0.173069	1.249693	0.143523	0.2
min	49.000000	0.000000	0.316175	1.000000	0.000000	0.0
25%	155.000000	0.000000	0.563680	3.000000	0.000000	0.4
50%	199.000000	0.000000	0.724428	4.000000	0.000000	0.6
75%	245.000000	0.000000	0.871345	5.000000	0.000000	0.8
max	310.000000	1.000000	1.000000	7.000000	1.000000	1.0

In [51]:
```
employee_data["last_evaluation"].isna().count()
```

Out[51]:
```
14116
```

In [52]:
```
employee_data1=employee_data
```

In [53]:
```
employee_data1.shape
```

Out[53]:
```
(14116, 11)
```

In [54]:
```
#Calculating the mean value of last_evaluation
meanLEV=employee_data1["last_evaluation"].mean()
```

In [55]:
```
meanLEV
```

Out[55]:
```
0.7183221624831715
```

- for last_evaluation column mean and median values are very close. It means the distribution is normal distribution. Therefore, mean value of the column can be taken to replace the null values

In [56]:
```
# Null value count in last_evalutaion column
employee_data["last_evaluation"].isna().sum()
```

Out[56]:
```
1487
```

In [57]:
```
# Replace the null values of last_evaluation with mean value of that column

employee_data["last_evaluation"]=employee_data["last_evaluation"].replace(np.NaN,me
```

In [58]:
```
# Null value count in last_evalutaion column after replacing with mean value
employee_data["last_evaluation"].isna().sum()
```

Out[58]:
```
0
```

In [59]:
```
employee_data["department"].isna().sum()
```

Out[59]:
```
706
```

In [60]:
```
# for Null values of department, we use a new department D00-UN
employee_data["department"]=employee_data["department"].replace(np.NaN,"D00-UN")
```

In [61]:
```
employee_data["department"].value_counts()
```

Out[61]:
```
D00-SS     3895
D00-ENG    2573
D00-SP     2108
D00-IT     1359
D00-PD      853
D00-MT      812
D00-FN      722
D00-UN      706
D00-MN      590
D00-AD      175
D00-PR      173
D00-TP      150
Name: department, dtype: int64
```

In [62]:
```
employee_data.isna().sum()
```

Out[62]:
```
avg_monthly_hrs        0
department             0
filed_complaint        0
last_evaluation        0
n_projects             0
recently_promoted      0
salary                 0
satisfaction         150
status                 0
tenure               150
employee_id            0
dtype: int64
```

In [63]:
```
employee_data.groupby("department")["satisfaction","tenure"].count()
```

Out[63]:

department	satisfaction	tenure
D00-AD	175	175
D00-ENG	2573	2573
D00-FN	722	722
D00-IT	1359	1359
D00-MN	590	590
D00-MT	812	812
D00-PD	853	853
D00-PR	173	173
D00-SP	2108	2108
D00-SS	3895	3895
D00-TP	0	0
D00-UN	706	706

In [64]:
```
employee_data[employee_data["tenure"].isna()]
```

Out[64]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promot
44	125.0	D00-TP	0.0	0.718322	3	0
245	124.0	D00-TP	0.0	0.718322	3	0
251	119.0	D00-TP	0.0	0.718322	2	0
414	126.0	D00-TP	0.0	0.718322	2	0
468	65.0	D00-TP	0.0	0.718322	1	0
...	...	...	...	...	...	
13741	122.0	D00-TP	0.0	0.718322	2	0
13774	49.0	D00-TP	0.0	0.718322	2	0
13841	97.0	D00-TP	1.0	0.718322	1	0
13905	132.0	D00-TP	0.0	0.718322	2	0
13960	116.0	D00-TP	0.0	0.718322	2	0

150 rows × 11 columns

- Total rows in D00-TP department are 150. All these rows have null values for satisfaction and tenure column.
- Therefore, We can assume that the employees of this temporary department will have a satisfaction of 0 and tenure of 1.

In [65]:
```
# replacing null values of tenure with 1
employee_data['tenure']=employee_data["tenure"].replace(np.NaN,1)
```

In [66]:
```
# replacing null values in satisfaction with 0
employee_data['satisfaction']=employee_data["satisfaction"].replace(np.NaN,0)
```

In [67]:
```
employee_data.isna().sum()
```

Out[67]:
```
avg_monthly_hrs      0
department           0
filed_complaint      0
last_evaluation      0
n_projects           0
recently_promoted    0
salary               0
satisfaction         0
status               0
tenure               0
employee_id          0
dtype: int64
```

- All the missing values are handled.

In [68]:
```
employee_data.shape
```

Out[68]:
```
(14116, 11)
```

In [69]:
```
employee_details_data.shape
```

Out[69]:
```
(14245, 4)
```

Merging of Dataframes

In [70]:
```
# Now we have to merge employee_data and employee_details_data tables. Inner join i
final_data=pd.merge(employee_data,employee_details_data, how="inner", on="employee_
```

In [71]:
```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14116 entries, 0 to 14115
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   avg_monthly_hrs    14116 non-null  float64
 1   department         14116 non-null  object 
 2   filed_complaint    14116 non-null  float64
 3   last_evaluation    14116 non-null  float64
 4   n_projects         14116 non-null  int64  
 5   recently_promoted  14116 non-null  float64
 6   salary             14116 non-null  object 
 7   satisfaction       14116 non-null  float64
 8   status             14116 non-null  object 
 9   tenure             14116 non-null  float64
 10  employee_id        14116 non-null  int64  
 11  age                14116 non-null  int64  
 12  gender             14116 non-null  object 
 13  marital_status     14116 non-null  object 
dtypes: float64(6), int64(3), object(5)
memory usage: 1.6+ MB
```

In [72]:
```
final_data.head()
```

Out[72]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promoted	
0	246.0	D00-UN	0.0	0.866838	6	0.0	r
1	134.0	D00-UN	0.0	0.555718	2	0.0	
2	156.0	D00-SS	1.0	0.474082	2	0.0	r
3	256.0	D00-SP	0.0	0.961360	6	0.0	
4	146.0	D00-SS	0.0	0.507349	2	0.0	r

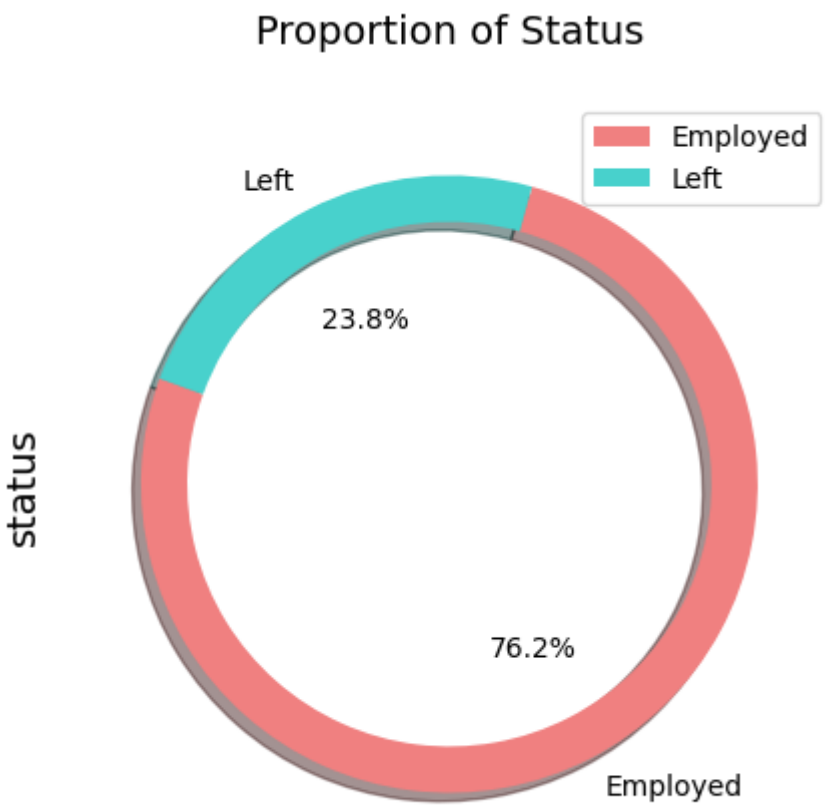
In [73]:
```
final_data.shape
```

Out[73]:
```
(14116, 14)
```

Exploratory Data Analysis

Pie chart to show the distribution of **status**

In [198…
```
def plot_brand_dist():
  labels = final_data['status'].value_counts().index
  values = final_data['status'].value_counts().values

  plt.figure(figsize = [5, 5])
  plt.pie(x = values, labels = labels, autopct = '%3.1f%%', wedgeprops = dict(width
                                        shadow = True, startangle = 160, colors = [
  plt.legend()
  plt.ylabel('status', size = 14)
  plt.title('Proportion of Status', size = 14, y = 1.05)
  plt.show()
plot_brand_dist()
```

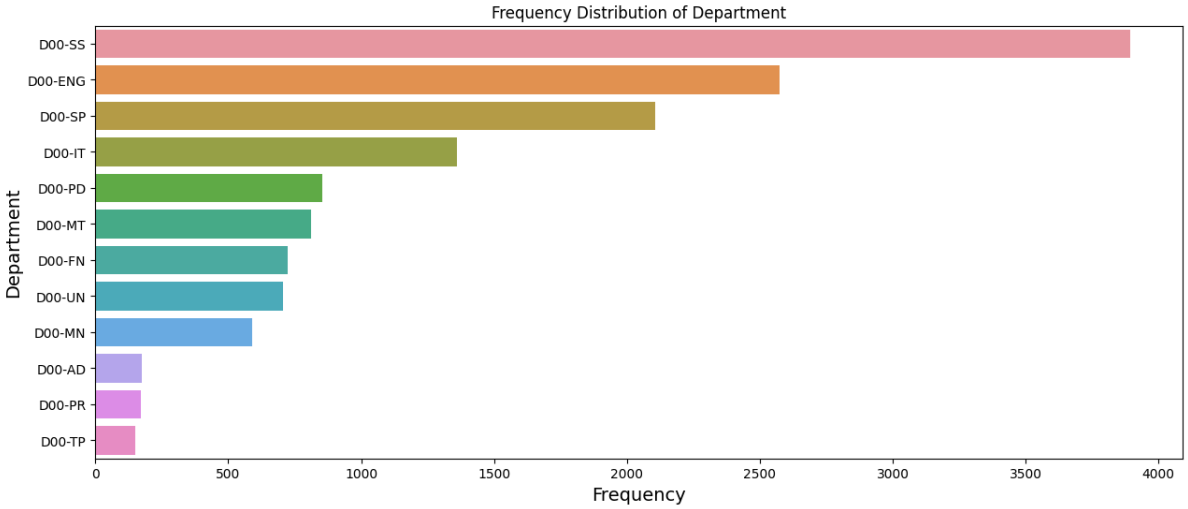


- The chart shows that **23.8%** of employees left the organisation

- Does **department** play an important role in **employee churn**?

Frequency distribution of department

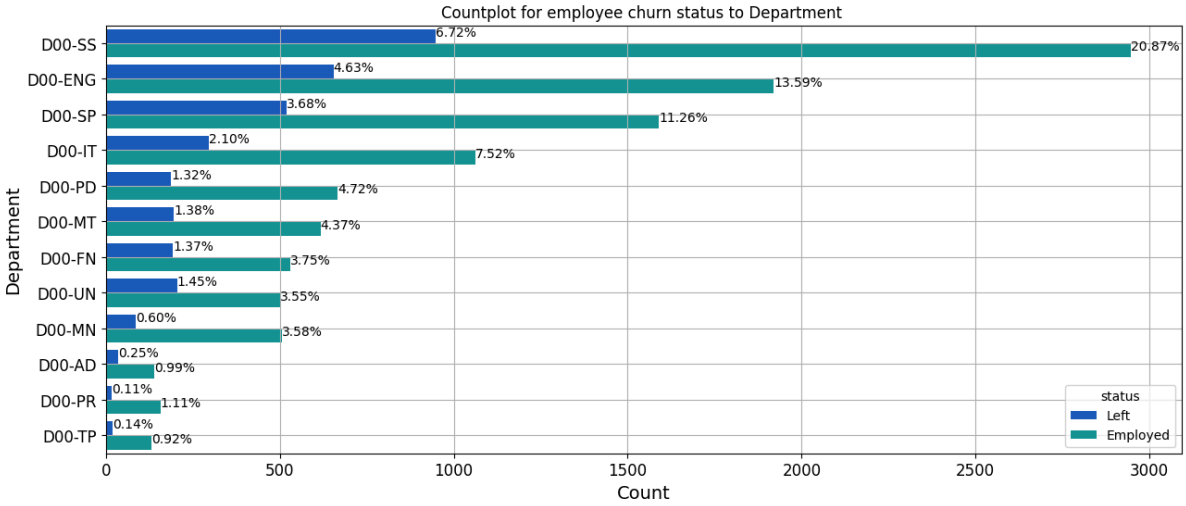
In [75]:
```
# Frequency distribution of department
def plot_department_freq():
  first_11 = final_data['department'].value_counts()
  plt.figure(figsize = [15, 6])
  sns.barplot(x = first_11.values, y = first_11.index)
  plt.xlabel('Frequency', size = 14)
  plt.ylabel('Department', size = 14)
  plt.title('Frequency Distribution of Department')
  plt.show()
plot_department_freq()
# same can be achieved by using countplot:
# sns.countplot(y='department',data=final_data,order=final_data['department'].value
```



- Frequency distribution plot shows that employees are highest in **D00-SS** and Lowest in **D00-TP**

department wise analysis

In [76]:
```
# instantiate a figure of size 15X7
fig=plt.figure(figsize=[15,6])

# Creating a count plot of department vs status
ax=sns.countplot(y='department',hue='status', data=final_data,palette="winter",orde


# Adding percentages to the bars

total=final_data.shape[0]

for p in ax.patches:
    percentage='{:.2f}%'.format(100*p.get_width()/total)
    x=p.get_x()+p.get_width()
    y=p.get_y()+p.get_height()/2
    ax.annotate(percentage,(x,y))

# Adding some cosmetics

plt.yticks(size=12)
plt.xticks(size=12)
plt.xlabel(xlabel="Count",size=14)
plt.ylabel(ylabel="Department",size=14)
plt.title(label="Countplot for employee churn status to Department ")
plt.grid(True)

# Displaying the figure
plt.show()
```

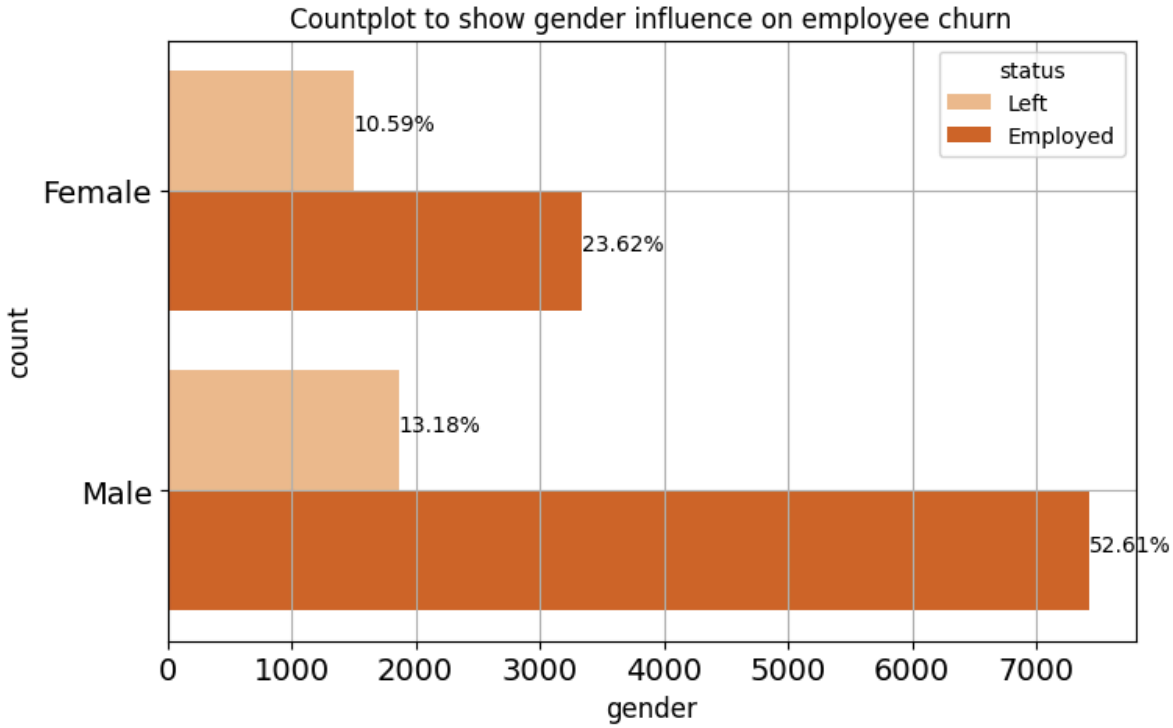


- The above chart indicates that the highest employee attrition belonged to **Sales** department followed by **Engineering, Support and IT departments**

- Does **gender** play a role in employee churn

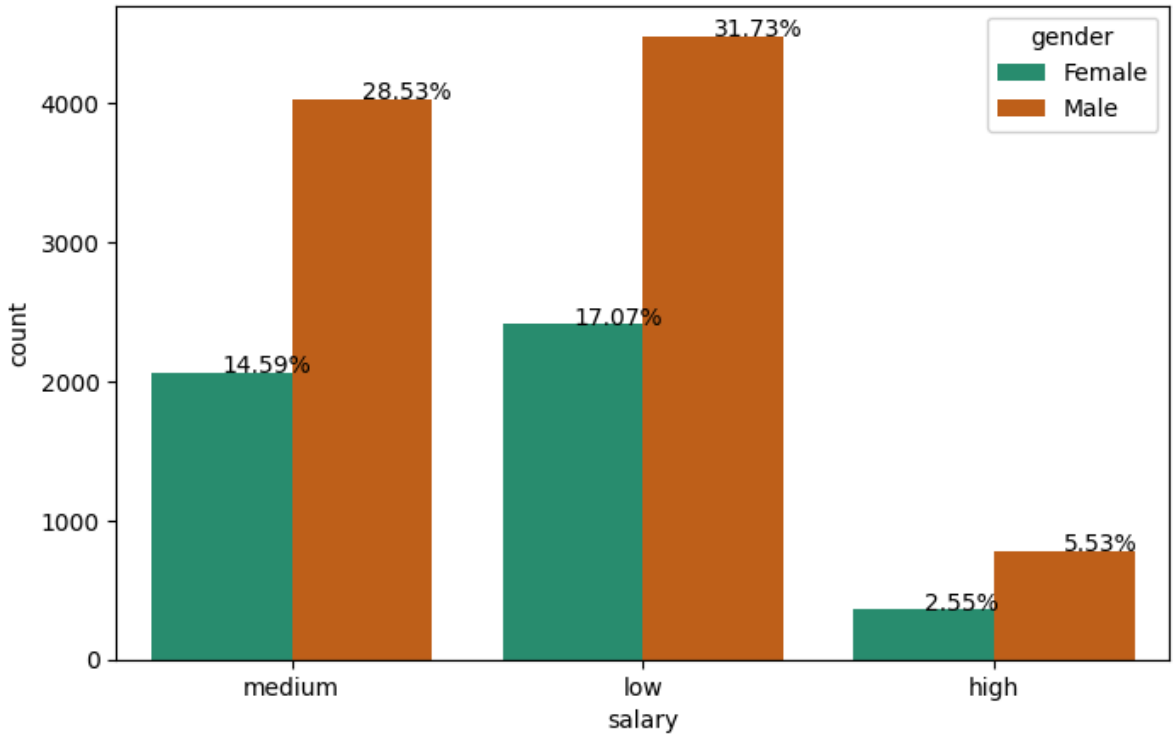
gender wise analysis

In [77]:
```
fig=plt.figure(figsize=[8,5])
ax=sns.countplot(y="gender",hue="status",data=final_data,palette="Oranges")


total=final_data.shape[0]

for p in ax.patches:
    percentage='{:.2f}%'.format(100*p.get_width()/total)
    x=p.get_x()+p.get_width()
    y=p.get_y()+p.get_height()/2
    ax.annotate(percentage,(x,y))

plt.yticks(size=14)
plt.xticks(size=14)
plt.ylabel(ylabel="count",size=12)
plt.xlabel(xlabel="gender",size=12)
plt.title(label="Countplot to show gender influence on employee churn")
plt.grid(True)

plt.show()
```



Gender based Salary analysis

In [78]:
```
fig = plt.figure(figsize=(8, 5))
ax=sns.countplot(x='salary', data=final_data, hue='gender', palette='Dark2')
total=final_data.shape[0]

for p in ax.patches:
    percentage='{:.2f}%'.format(100*p.get_height()/total)
    y=p.get_y()+p.get_height()
    x=p.get_x()+p.get_width()/2
    ax.annotate(percentage,(x,y))

plt.show()
```



- There are more number of employees with **Medium and Low** salaries

- **satisfaction** and **salary** are correlated ?
- Does both of them have contribute to the **employee churn rate

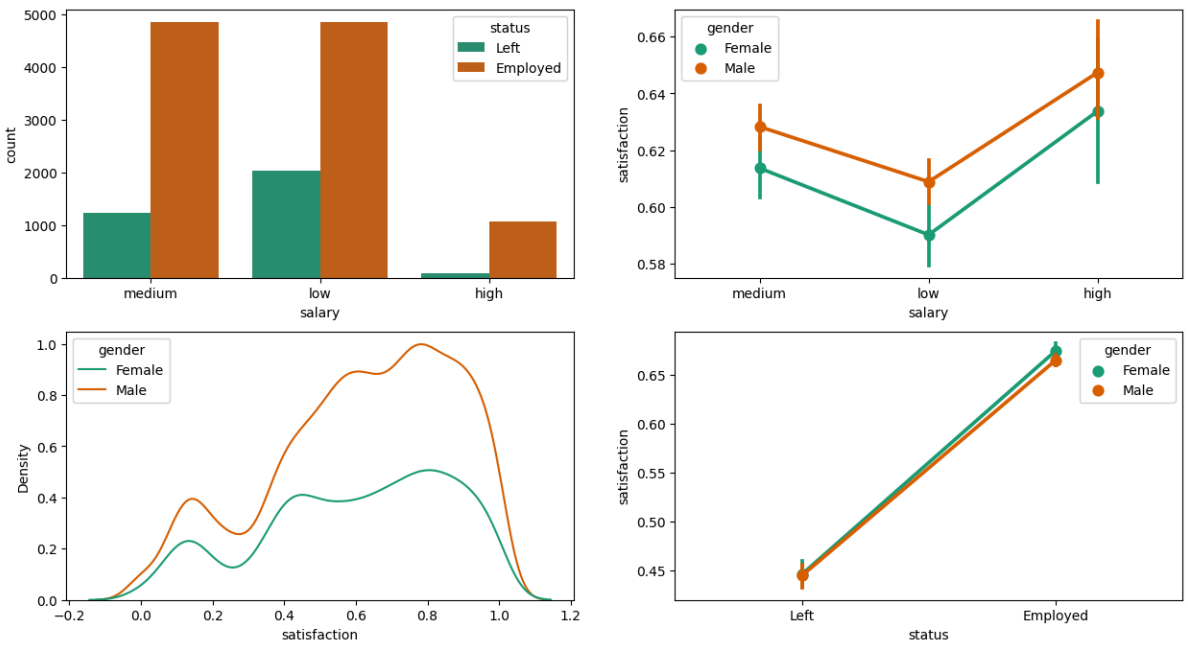
In [79]:
```
pd.crosstab(final_data['salary'],final_data['gender'],margins=True).sort_values('Al
```

Out[79]:

gender salary	Female	Male	All
high	360	781	1141
medium	2059	4027	6086
low	2410	4479	6889
All	4829	9287	14116

- Table show that in all the income groups, number of **males** are higher than **females**

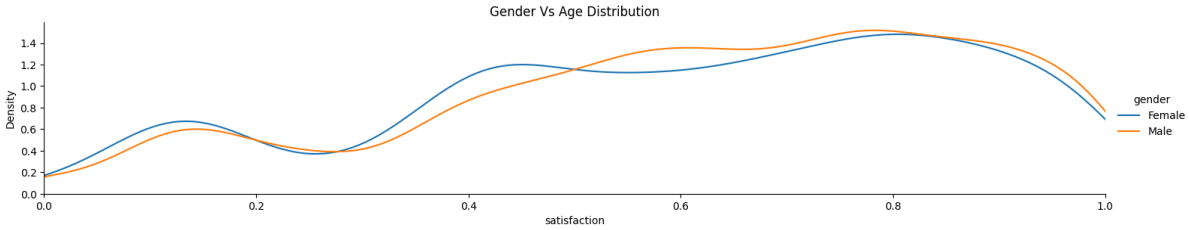
In [80]:
```
# Instantiate a figure of size of 15 x 7 inches with 4 subplots.
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15,8))

# Plotting a countplot of salary concerning employee churn rate in the first subplo

sns.countplot(x='salary', data=final_data, hue='status', palette='Dark2',ax=ax[0,0]
# Plotting a pointplot to show impact of salary on employee satisfaction second sub
sns.pointplot(x='salary',y='satisfaction', data=final_data, hue='gender', palette='
# Plotting a KDE plot to show density of employee satisfaction in third subplot
sns.kdeplot(x='satisfaction',hue='gender',data=final_data,palette='Dark2',ax=ax[1,0
# Plotting a pointplot to show impact of satisfaction on employee churn.
sns.pointplot(y='satisfaction', x='status', data=final_data, hue='gender', palette=


plt.show()
```



- Plot 1 shows that **low** and **medium** salary employees have higher churn rate then **high** salaries employees
- Plot 2 shows that there is correlation between **salary** and **satisfaction**. In both males and females **High** salary employees are more satisfied than **low and medium** salary.
- Plot 3 shows that more employees fall in the satisfaction levels between **0.4 tp 1.0**
- Plot 4 shows that as **satisfaction** increases, employee churn rate decreases

In [81]:
```
#sns.lineplot(y='satisfaction',x='salary', data=final_data, hue='gender', palette='
as_fig = sns.FacetGrid(final_data,hue='gender',aspect=5)
as_fig.map(sns.kdeplot,'satisfaction')
oldest = final_data['satisfaction'].max()
as_fig.set(xlim=(0,oldest))
as_fig.add_legend()
plt.title("Gender Vs Age Distribution")
```

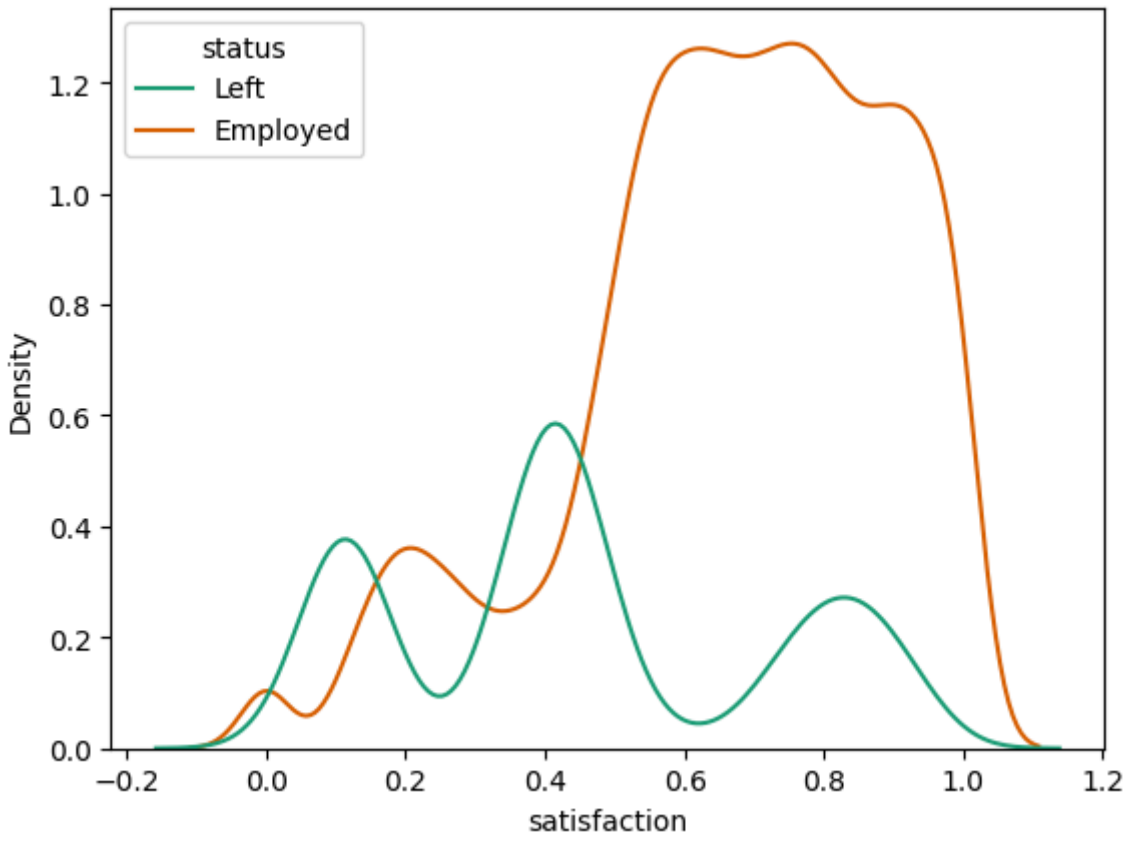
Out[81]:
```
Text(0.5, 1.0, 'Gender Vs Age Distribution')
```



Satisfaction based anlysis

In [82]:
```
sns.kdeplot(x='satisfaction',hue='status',data=final_data,palette='Dark2')
```

Out[82]:
```
<AxesSubplot: xlabel='satisfaction', ylabel='Density'>
```



- Does **tenure** play an important role employee churn

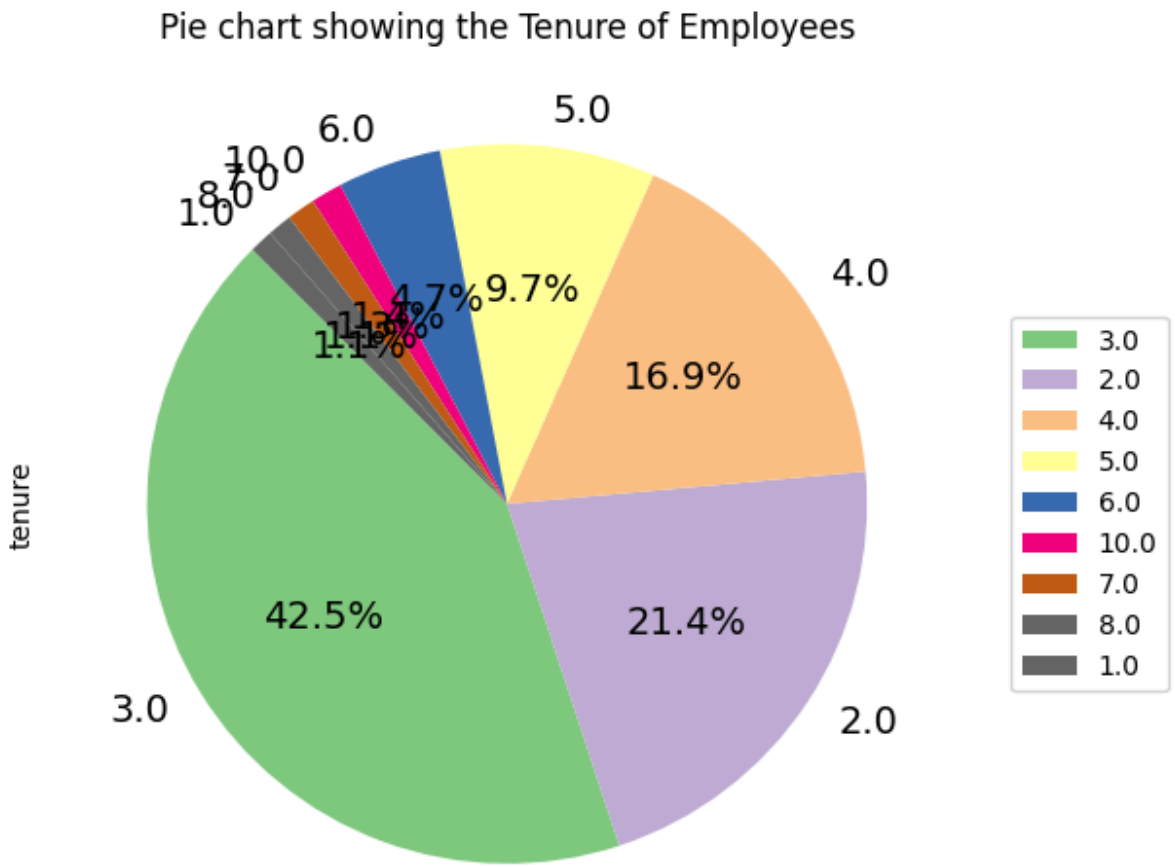
Tenure based analysis

In [83]:
```
final_data['tenure'].value_counts()
```

Out[83]:
```
3.0     6006
2.0     3019
4.0     2386
5.0     1363
6.0      659
10.0     198
7.0      180
8.0      155
1.0      150
Name: tenure, dtype: int64
```

In [84]:
```
final_data['tenure'].value_counts().plot(kind='pie',fontsize=14, autopct='%3.1f%%',
                                         cmap='Accent')
plt.legend(labels = final_data['tenure'].value_counts().index, bbox_to_anchor=(1.25
plt.title('Pie chart showing the Tenure of Employees')
```

Out[84]:
```
Text(0.5, 1.0, 'Pie chart showing the Tenure of Employees')
```



- Most of employees fall between **2 to 5** years of tenure

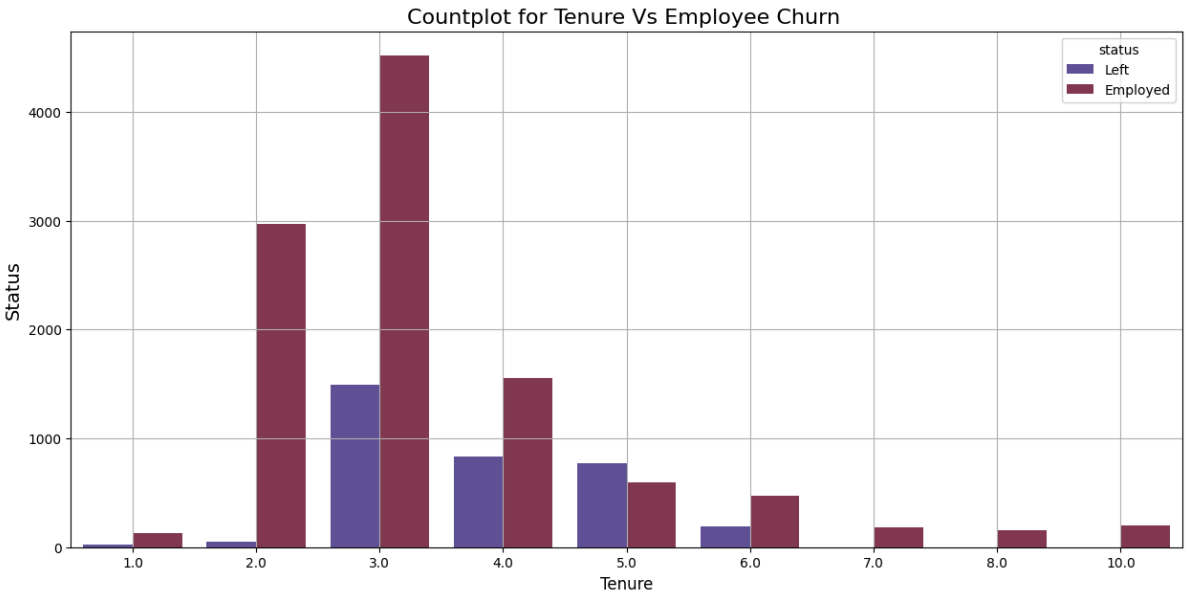
In [85]:
```
pd.crosstab(final_data['status'],final_data['tenure'],margins=True).sort_values('Al
```

Out[85]:

tenure	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	10.0	All
status										
Left	20	48	1492	834	771	190	0	0	0	3355
Employed	130	2971	4514	1552	592	469	180	155	198	10761
All	150	3019	6006	2386	1363	659	180	155	198	14116

In [86]:

```
# Instantiate a figure of size of 15 x 7 inches.
fig = plt.figure(figsize=(15, 7))

# Plotting a pointplot of FamilySize vs survived.
sns.countplot(x='tenure',hue='status', data=final_data, palette='twilight')

# Adding some cosmetics - ticks, labels, title, and grid to the pointplot.
plt.title(label='Countplot for Tenure Vs Employee Churn ', size=16)
plt.xticks(size=10)
plt.yticks(size=10)
plt.xlabel(xlabel='Tenure',size=12)
plt.ylabel(ylabel='Status', size=14)
plt.grid(b=True)

# Display the figure.
plt.show()
```



- People with tenure **3,4,5** have higher rate of churn.

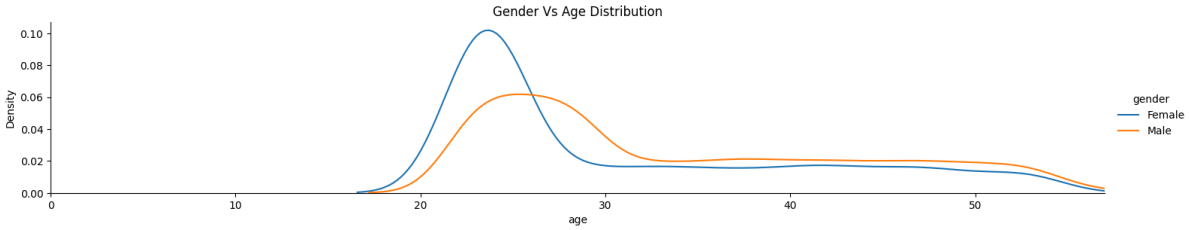
- Does **age** play a role in employee churn

age based analysis

In [87]:

```
as_fig = sns.FacetGrid(final_data,hue='gender',aspect=5)
as_fig.map(sns.kdeplot,'age')
oldest = final_data['age'].max()
as_fig.set(xlim=(0,oldest))
as_fig.add_legend()
plt.title("Gender Vs Age Distribution")
```

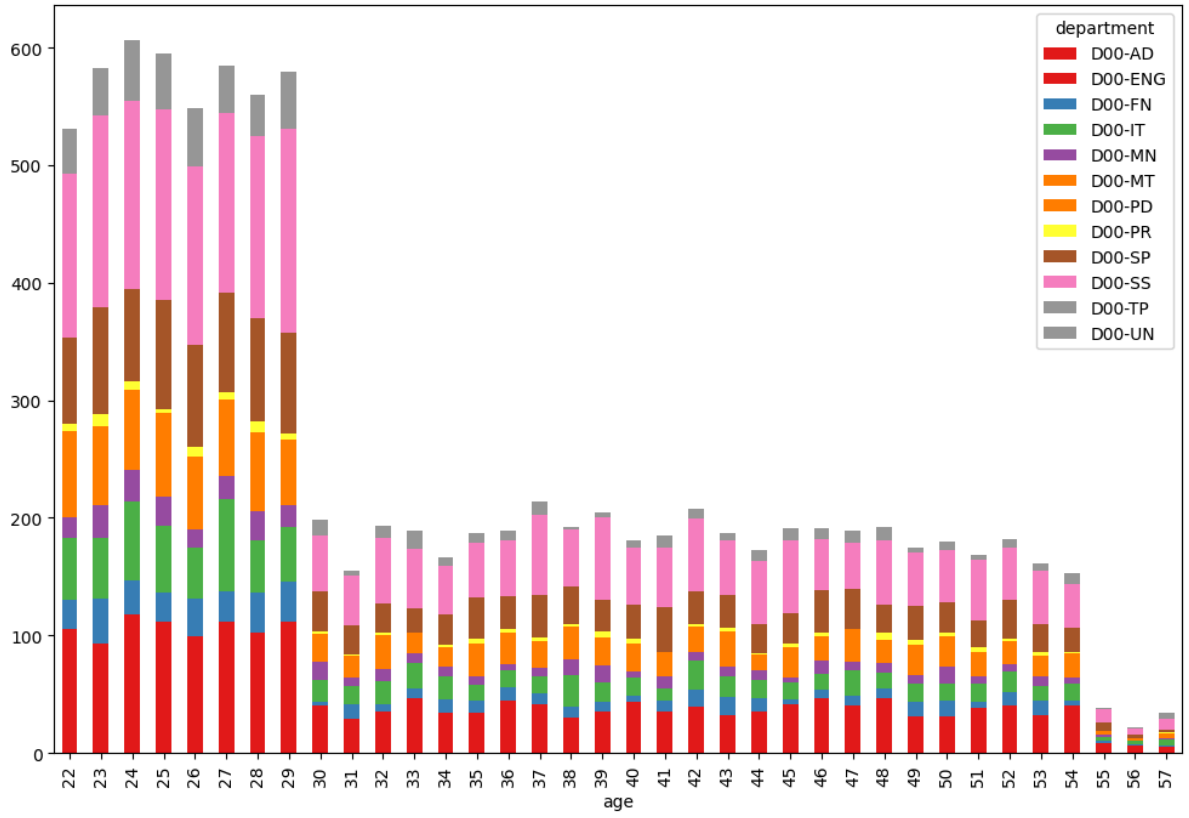
Out[87]:
```
Text(0.5, 1.0, 'Gender Vs Age Distribution')
```



- There are more number of females than males in the age group of 20 t0 25

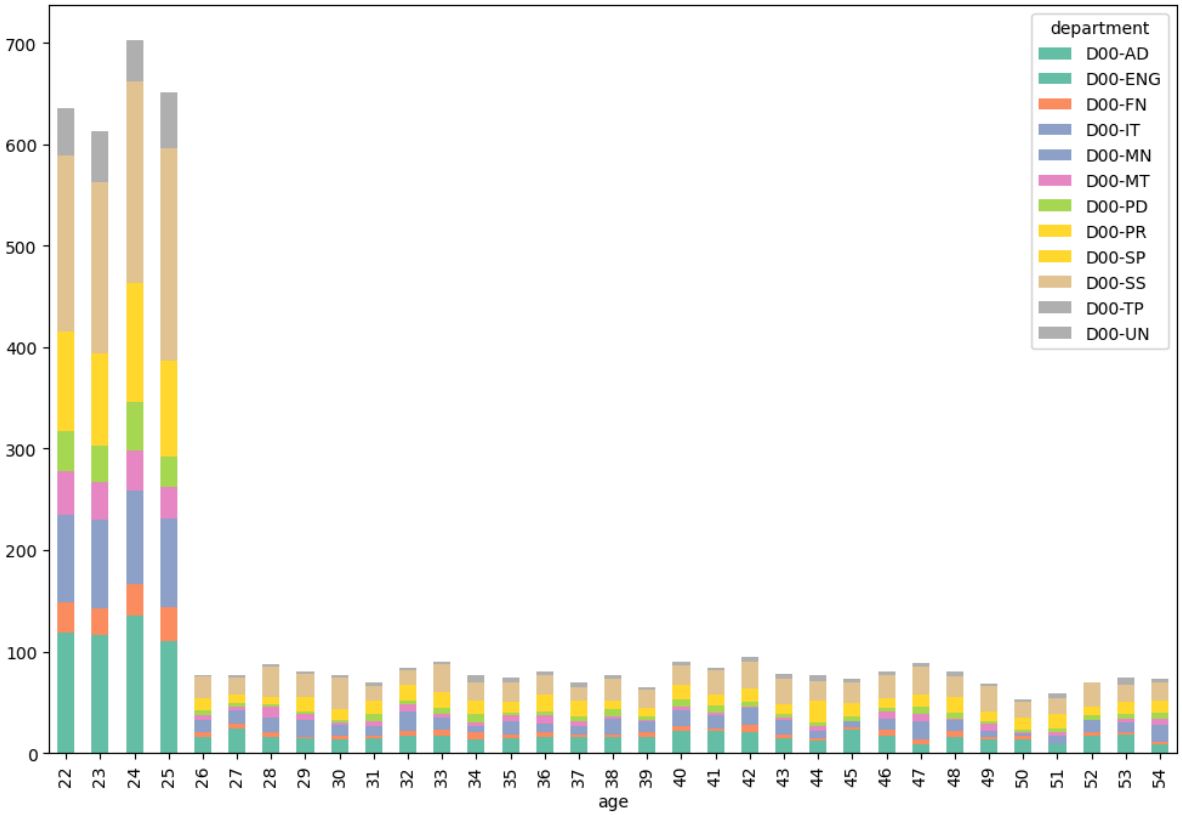
In [88]:
```
a= final_data.query("gender == 'Male'")
a.groupby('age')['department'].value_counts().unstack().plot(kind='bar',cmap='Set1'
                                                             figsize=(12,8),stack
```

Out[88]:
```
<AxesSubplot: xlabel='age'>
```



In [89]:
```
a= final_data.query("gender == 'Female'")
a.groupby('age')['department'].value_counts().unstack().plot(kind='bar',cmap='Set2'
                                                             figsize=(12,8),stack
```

Out[89]:
```
<AxesSubplot: xlabel='age'>
```



- Female employees are more in the age groups of **22 to 25**
- Male employees are more in the age groups of **22 to 29**

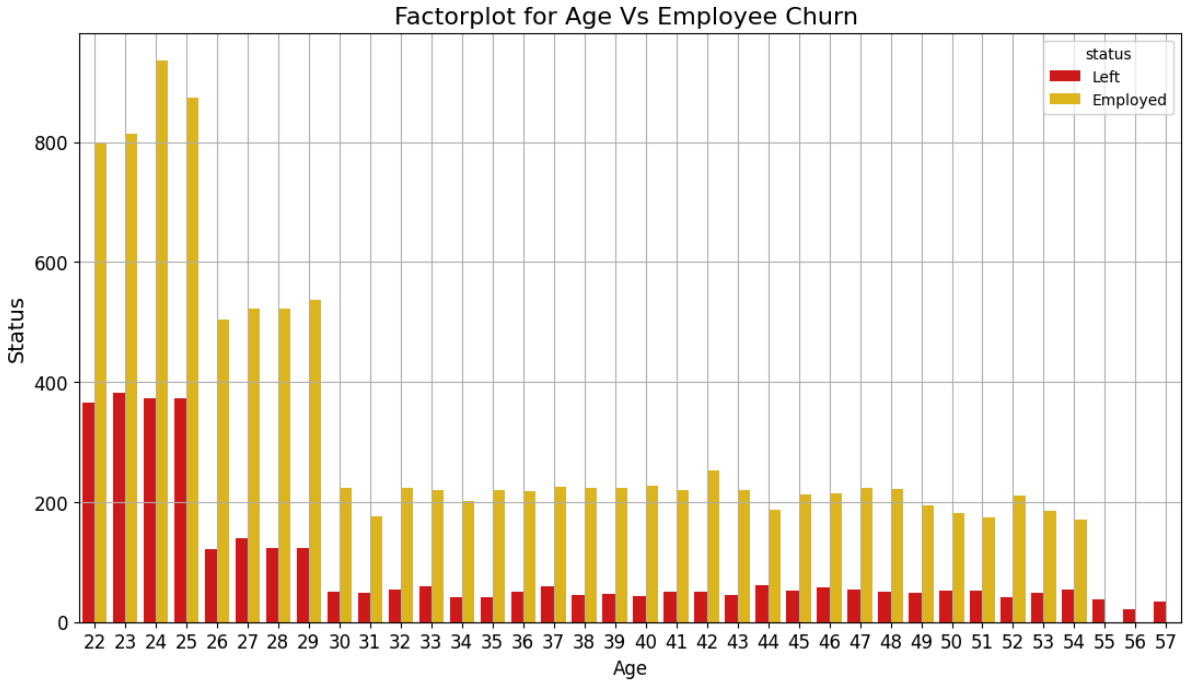
In [90]:
```
pd.crosstab(final_data['status'],final_data['age'],margins=True).sort_values('All')
```

Out[90]:

status	22	23	24	25	26	27	28	29	30	31	...	49	50	51	52	53	
Left	366	382	373	372	122	140	124	124	51	48	...	48	52	52	41	49	
Employed	800	814	935	874	504	522	523	536	224	177	...	195	181	175	211	186	1
All	1166	1196	1308	1246	626	662	647	660	275	225	...	243	233	227	252	235	2

3 rows × 37 columns

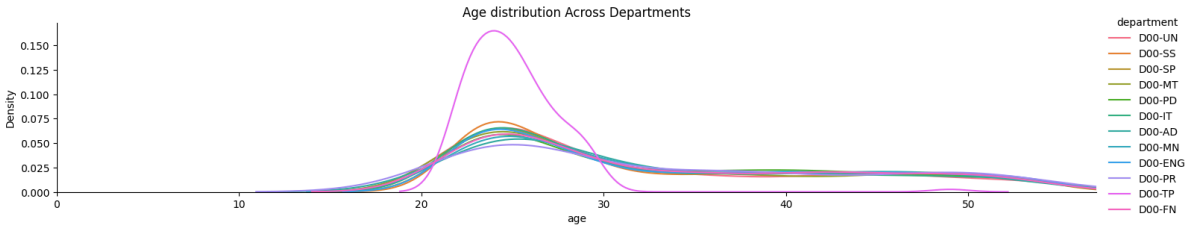
In [91]:
```
# Instantiate a figure of size of 15 x 7 inches.
fig = plt.figure(figsize=(13, 7))

# Plotting a pointplot of FamilySize vs survived.
sns.countplot(x='age',hue='status', data=final_data, palette='hot')


# Adding some cosmetics - ticks, labels, title, and grid to the pointplot.
plt.title(label='Factorplot for Age Vs Employee Churn ', size=16)
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel(xlabel='Age',size=12)
plt.ylabel(ylabel='Status', size=14)
plt.grid(b=True)

# Display the figure.
plt.show()
```



- People between the age **23 to 25** have a higher attrition rate.

In [92]:
```
as_fig = sns.FacetGrid(final_data,hue='department',aspect=5)
as_fig.map(sns.kdeplot,'age')
oldest = final_data['age'].max()
as_fig.set(xlim=(0,oldest))
as_fig.add_legend()
plt.title('Age distribution Across Departments')
```

Out[92]:
```
Text(0.5, 1.0, 'Age distribution Across Departments')
```



There are more number of employees who belong to **D00-TP** in the age group of **20 to 30**

- Does **marital_status** play a role in employee churn?

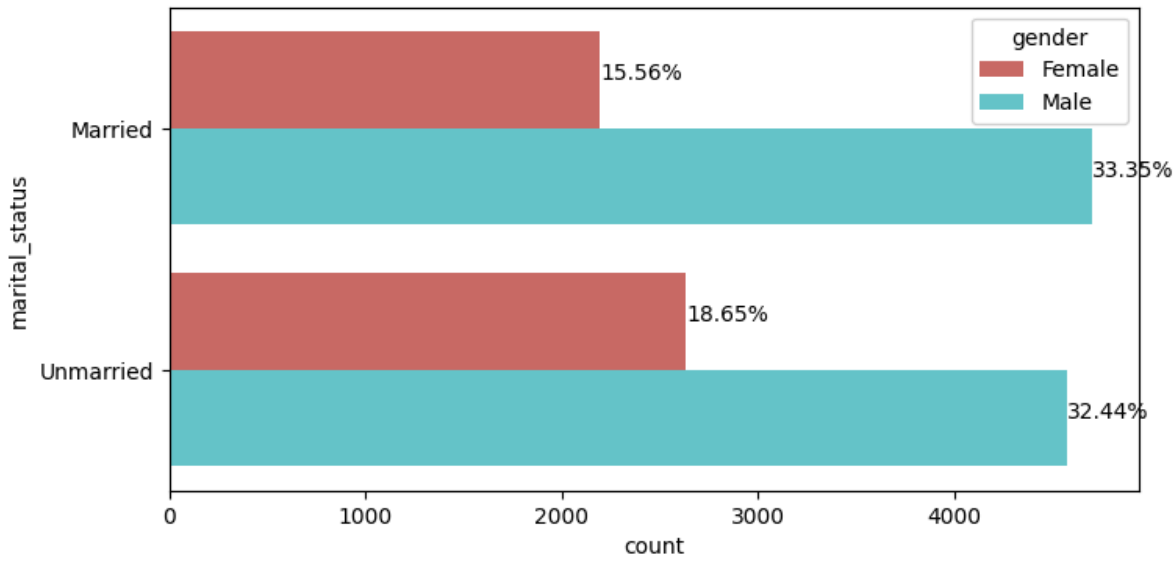
marital status impact on employee churn

In [93]:
```
pd.crosstab(final_data['marital_status'],final_data['gender'],margins=True).sort_va
```

Out[93]:

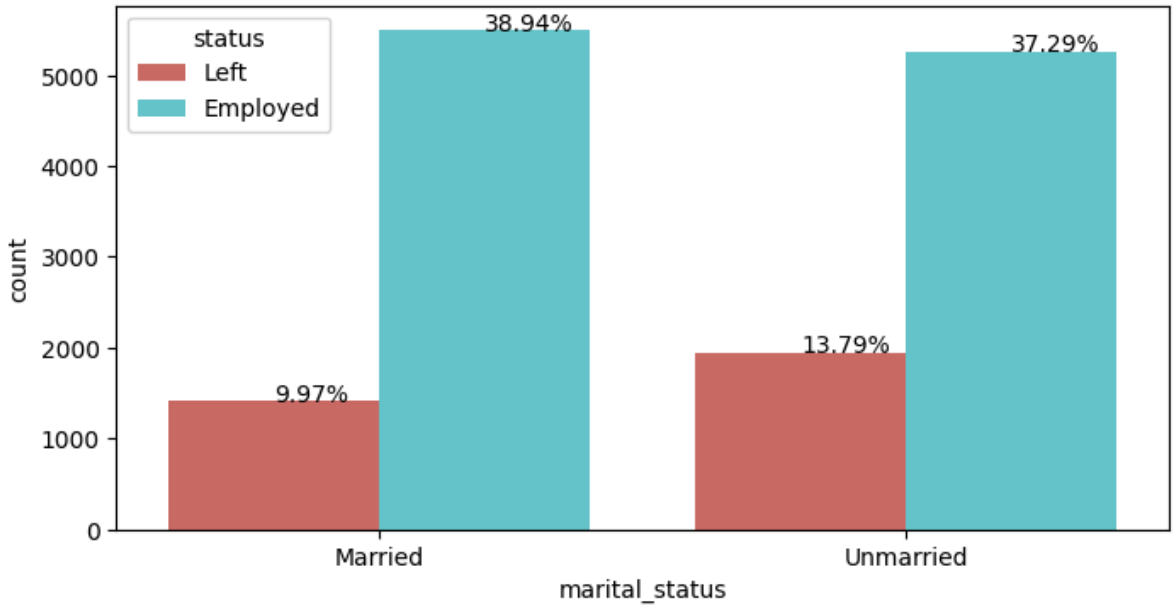
gender	Female	Male	All
marital_status			
Married	2197	4708	6905
Unmarried	2632	4579	7211
All	4829	9287	14116

In [94]:
```
plt.figure(figsize = [8, 4])
ax=sns.countplot(y='marital_status',data=final_data,hue='gender',palette='hls')

total=final_data.shape[0]

for p in ax.patches:
    percentage='{:.2f}%'.format(100*p.get_width()/total)
    x=p.get_x()+p.get_width()
    y=p.get_y()+p.get_height()/2
    ax.annotate(percentage,(x,y))

plt.show()
```



There are more number of Unmarried female than male.

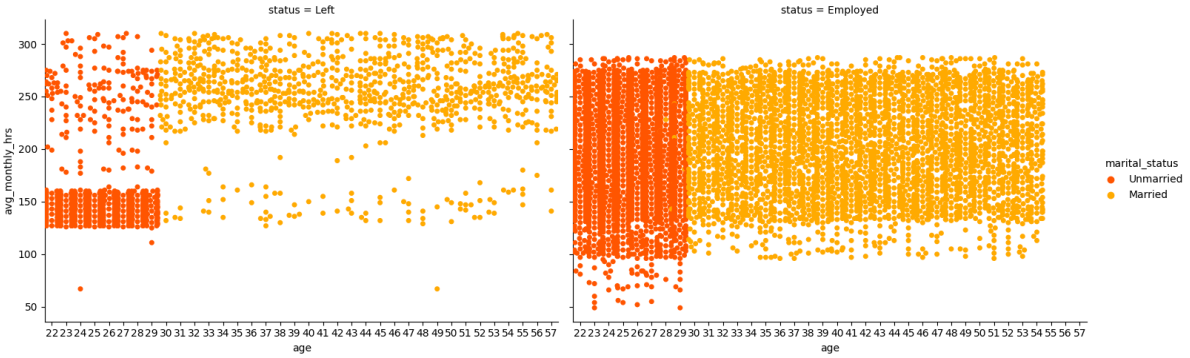
In [95]:
```
plt.figure(figsize = [8, 4])
ax=sns.countplot(x='marital_status',data=final_data,hue='status',palette='hls')

total=final_data.shape[0]

for p in ax.patches:
    percentage='{:.2f}%'.format(100*p.get_height()/total)
    y=p.get_y()+p.get_height()
    x=p.get_x()+p.get_width()/2
    ax.annotate(percentage,(x,y))

plt.show()
```



There is more attrition in **unmarried** employees

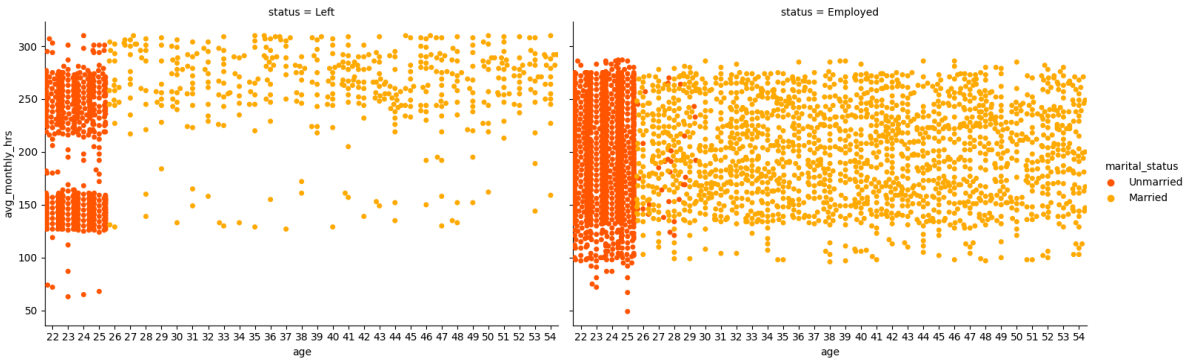
In [96]:
```
plt.figure(figsize = [15, 8])
sns.catplot(x="age", y='avg_monthly_hrs', hue="marital_status",
            col="status", aspect=1.5,
            kind="swarm", data=final_data[(final_data.gender == 'Male')],palette='a
```

Out[96]:
```
<seaborn.axisgrid.FacetGrid at 0x1ad366b6280>

<Figure size 1500x800 with 0 Axes>
```



In [97]:
```
plt.figure(figsize = [15, 8])
sns.catplot(x="age", y='avg_monthly_hrs', hue="marital_status",
            col="status", aspect=1.5,
            kind="swarm", data=final_data[(final_data.gender == 'Female')],palette=
```

Out[97]:
```
<seaborn.axisgrid.FacetGrid at 0x1ad380598b0>

<Figure size 1500x800 with 0 Axes>
```



- For unmarried male left employees, one cluster is seen at around 150hours of avg_monthly_hrs
- For unmarried female left employees, two clusters are seen with respect to avg_monthly_hrs, one higher (around 250hours) and another lower (around 150hours)

- Does **promotion** play a role in employee churn

promotion based analysis on employee churn

In [98]:
```
print(final_data.pivot_table('employee_id', index='recently_promoted', columns='ger
```

```
gender              Female  Male
recently_promoted
0.0                   4739  9080
1.0                     90   207
```

- Very few employees are promoted. Therefore the data is not sufficient to gain any insights on this feature

- Does **number of Projects** an employee is working on will impact churn rate?

Number of projects based analysis on employee churn

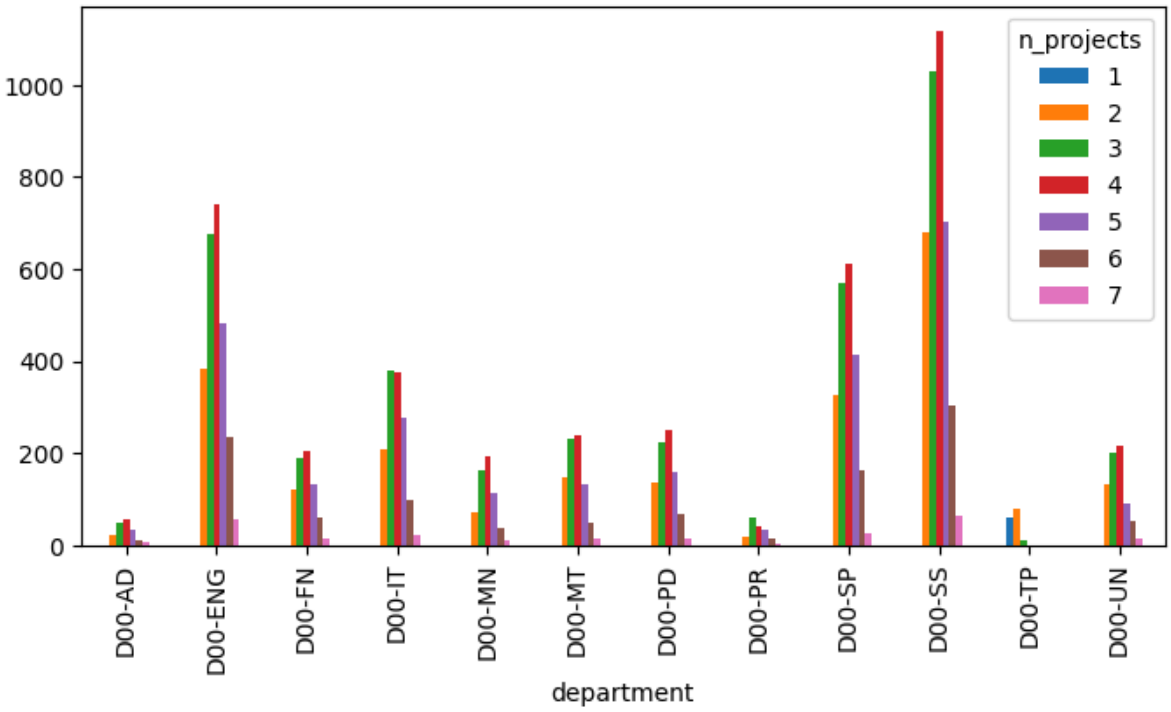
In [99]:
```
pd.crosstab(final_data['department'],final_data['n_projects'])
```

Out[99]:

n_projects	1	2	3	4	5	6	7
department							
D00-AD	0	20	50	56	32	12	5
D00-ENG	0	384	678	742	481	233	55
D00-FN	0	119	190	206	131	61	15
D00-IT	0	207	380	376	276	98	22
D00-MN	0	73	163	192	114	36	12
D00-MT	0	146	232	237	134	50	13
D00-PD	0	137	223	249	160	69	15
D00-PR	0	17	61	42	34	16	3
D00-SP	0	327	569	613	412	162	25
D00-SS	0	680	1031	1116	701	303	64
D00-TP	61	79	10	0	0	0	0
D00-UN	0	133	201	215	91	53	13

In [200…
```
pd.crosstab(final_data.department,final_data.n_projects).plot(kind='bar',figsize=(8
```

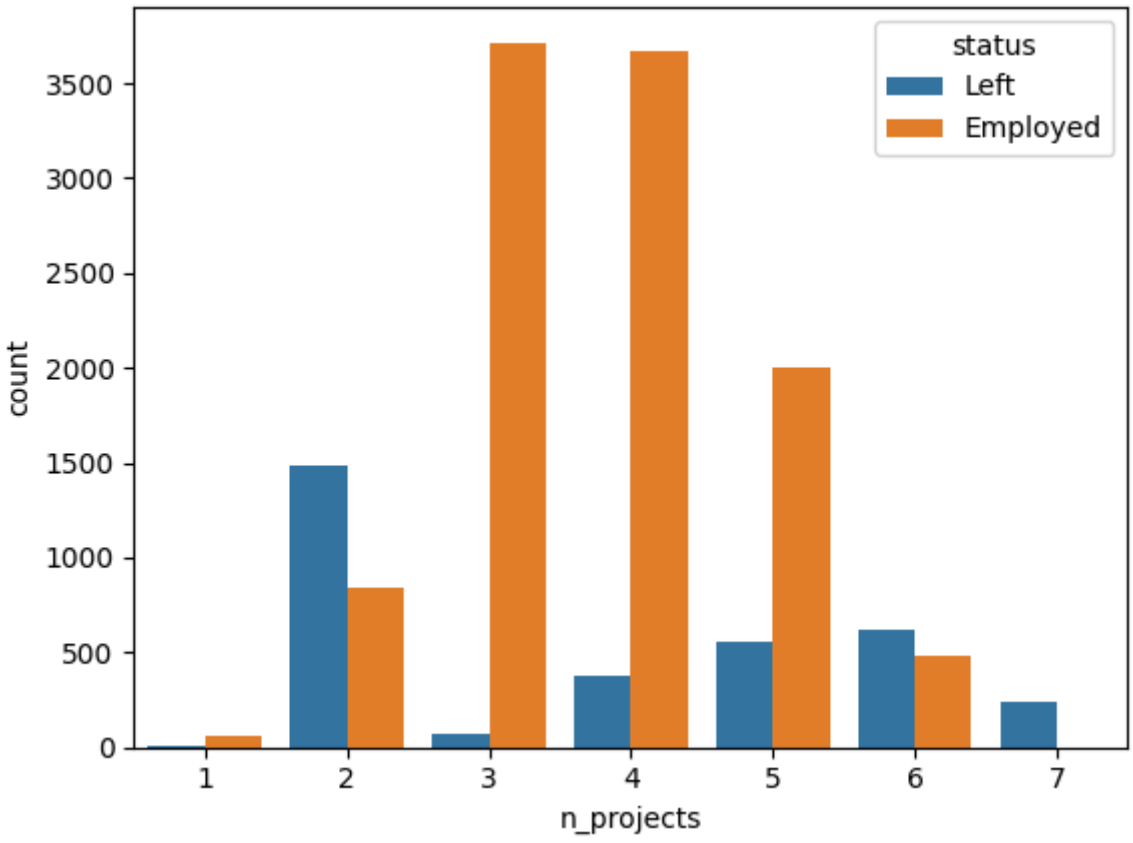
Out[200]:
```
<AxesSubplot: xlabel='department'>
```



- Sales, Engineering followed by Support deparment employees work on more number of projects
- Advertisement, Promotion and Temporary departments work on low number of projects

In [101…
```
sns.countplot(x="n_projects", data=final_data,hue='status')
```

Out[101]:
```
<AxesSubplot: xlabel='n_projects', ylabel='count'>
```



- people working on **2** projects have a higher churn rate

Does **satisfaction and last evalution** impact attrition rate

Satisfaction and last evaluation wise analysis

In [102…
```
emp_left=final_data[final_data['status']=='Left']

emp_left.head()

from sklearn.cluster import KMeans
# Filter data
left_emp =  emp_left[['satisfaction', 'last_evaluation']]
# Create groups using K-means clustering.
kmeans = KMeans(n_clusters = 3, random_state = 1).fit(left_emp)
```

In [103…
```
# Add new column "label" annd assign cluster labels.
left_emp['label'] = kmeans.labels_
# Create scatter plot
plt.scatter(left_emp['satisfaction'], left_emp['last_evaluation'], c=left_emp['labe
plt.xlabel('Satisfaction Level')
plt.ylabel('Last Evaluation Rating')
plt.title('Clusters of employees who left')
plt.show()
```

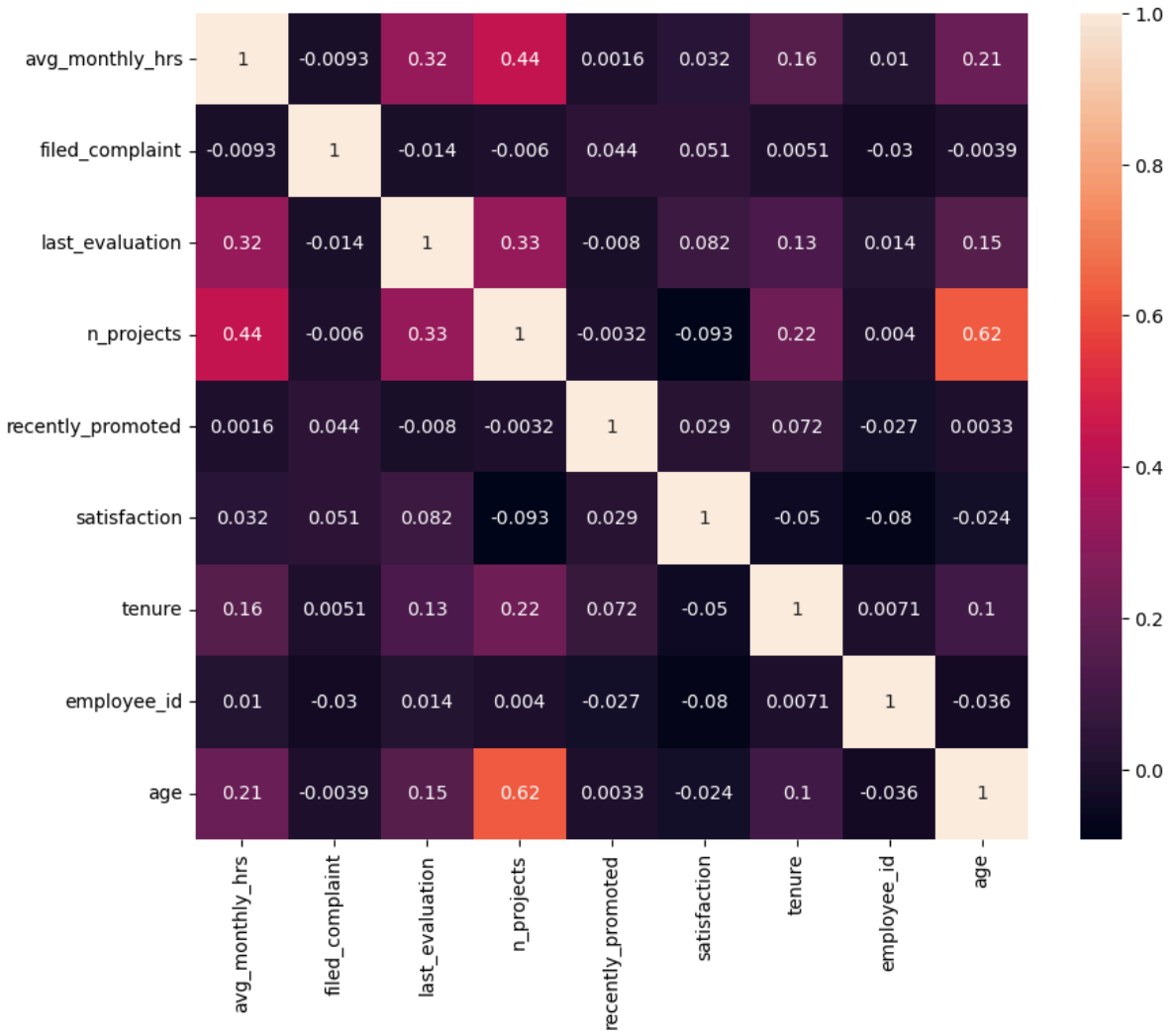
From the above clusters we can find that Employee who left the company can be grouped into 3 clusters of employees:

- High Evaluation rating, but low satisfaction
- Medium to low Evaluation rating and medium and low satisfaction
- High Evaluation rating and high satisfaction

Heat Map - Correlation between features

In [104…]
```
plt.figure(figsize=(10, 8))
sns.heatmap(final_data.corr(),annot=True)
```

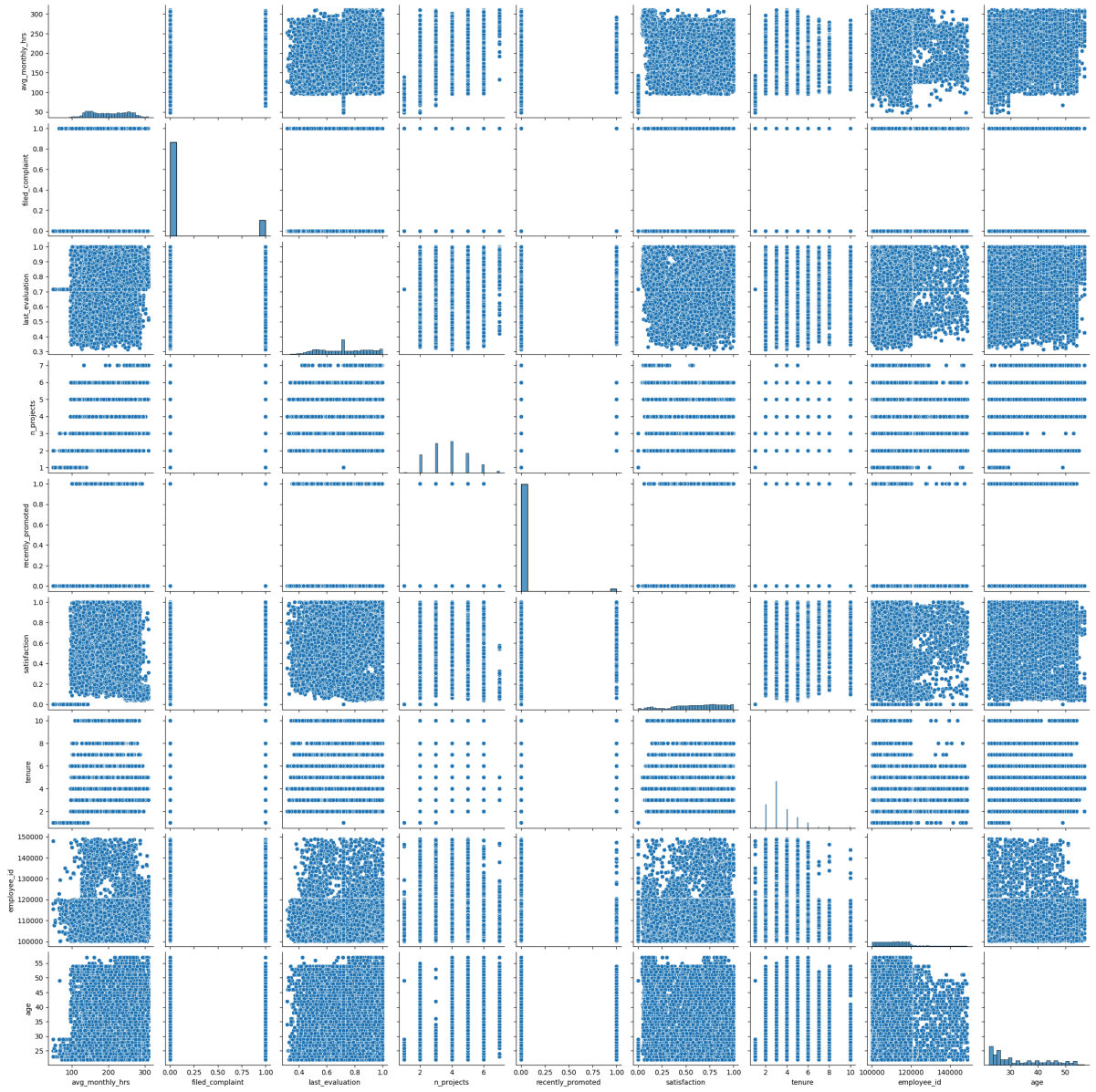
Out[104]:
```
<AxesSubplot: >
```



Pair Plot- Correlation between features

In [105…]
```
sns.pairplot(data=final_data)
```

Out[105]:
```
<seaborn.axisgrid.PairGrid at 0x1ad39ca7430>
```



Feature Engineering

In [106…

```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14116 entries, 0 to 14115
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   avg_monthly_hrs    14116 non-null  float64
 1   department         14116 non-null  object 
 2   filed_complaint    14116 non-null  float64
 3   last_evaluation    14116 non-null  float64
 4   n_projects         14116 non-null  int64  
 5   recently_promoted  14116 non-null  float64
 6   salary             14116 non-null  object 
 7   satisfaction       14116 non-null  float64
 8   status             14116 non-null  object 
 9   tenure             14116 non-null  float64
 10  employee_id        14116 non-null  int64  
 11  age                14116 non-null  int64  
 12  gender             14116 non-null  object 
 13  marital_status     14116 non-null  object 
dtypes: float64(6), int64(3), object(5)
memory usage: 1.6+ MB
```

- Remove employee id
- Convert department, salary, status, gender and marital status into numerical variables.

In [107…
```
final_data1=final_data.drop('employee_id',axis=1)
```

In [108…
```
final_data1.shape
```

Out[108]:
```
(14116, 13)
```

- Converting categorical **status** variable to numerical variable. i.e changing from object datatype to integer datatype.

In [109…
```
final_data1['status']= np.where(final_data1.status=='Left', 1, 0)
```

Encoding

Label encoding

- Apply Label encoding on **department** feature

In [110…
```
from sklearn.preprocessing import LabelEncoder
Label_Encoder = LabelEncoder()
Label_Encoder.fit(final_data1['department'])
final_data1['department'] = Label_Encoder.transform(final_data1['department'])
```

One hot encoding

- Apply one-hot encoding on **salary, gender and marital status**

In [111…
```
categorical_feature_oh=['salary','gender','marital_status']
final_data1=pd.get_dummies(final_data1,columns = categorical_feature_oh)
```

```
final_data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14116 entries, 0 to 14115
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   avg_monthly_hrs          14116 non-null  float64
 1   department               14116 non-null  int32  
 2   filed_complaint          14116 non-null  float64
 3   last_evaluation          14116 non-null  float64
 4   n_projects               14116 non-null  int64  
 5   recently_promoted        14116 non-null  float64
 6   satisfaction             14116 non-null  float64
 7   status                   14116 non-null  int32  
 8   tenure                   14116 non-null  float64
 9   age                      14116 non-null  int64  
 10  salary_high              14116 non-null  uint8  
 11  salary_low               14116 non-null  uint8  
 12  salary_medium            14116 non-null  uint8  
 13  gender_Female            14116 non-null  uint8  
 14  gender_Male              14116 non-null  uint8  
 15  marital_status_Married   14116 non-null  uint8  
 16  marital_status_Unmarried  14116 non-null  uint8  
dtypes: float64(6), int32(2), int64(2), uint8(7)
memory usage: 1.2 MB
```

Splitting dataset into Train and Test

In [112...
```
Y = final_data1['status']

X = final_data1.drop(['status'],axis=1)
```

In [113...
```
print(X.shape)
print(Y.shape)
```

```
(14116, 16)
(14116,)
```

In [114...
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test= train_test_split(X,Y,test_size=0.2,random_state=1,st
```

In [115...
```
print(X_train.shape)
print(Y_train.shape)
```

```
(11292, 16)
(11292,)
```

In [116...
```
print(X_test.shape)
print(Y_test.shape)
```

```
(2824, 16)
(2824,)
```

print(X_train.shape) print(Y_train.shape) print(X_test.shape) print(Y_test.shape)

-------- Scaling

In [117...
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

```
X_train_sc=scaler.fit_transform(X_train)
X_test_sc=scaler.fit_transform(X_test)
```

In [118…
```
X_s = scaler.fit_transform(X)
```

Modeling

1. Logistic Regression

In [119…
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(X_train_sc,Y_train)
Y_test_predsLR=lr.predict(X_test_sc)
```

In [120…
```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score

print("Accuracy Score: %.2f"% accuracy_score(Y_test_predsLR,Y_test))
print("Precision Score: %.2f "% precision_score(Y_test_predsLR,Y_test))
print("Recall Score: %.2f"% recall_score(Y_test_predsLR,Y_test))
print("F1 score: %.2f"% f1_score(Y_test_predsLR,Y_test))
```

```
Accuracy Score: 0.79
Precision Score: 0.35
Recall Score: 0.61
F1 score: 0.45
```

In [121…
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(Y_test_predsLR,Y_test))
print(confusion_matrix(Y_test_predsLR,Y_test))
print("Logistic Regression accuracy_score: {:.2f}% ". format(accuracy_score(Y_test_
```

```
              precision    recall  f1-score   support

           0       0.93      0.82      0.87      2438
           1       0.35      0.61      0.45       386

    accuracy                           0.79      2824
   macro avg       0.64      0.72      0.66      2824
weighted avg       0.85      0.79      0.81      2824

[[2004  434]
 [ 149  237]]
Logistic Regression accuracy_score: 79.36%
```

2. Decision Tree Classifier

In [122…
```
from sklearn.tree import DecisionTreeClassifier
```

In [123…
```
dt=DecisionTreeClassifier()
dt.fit(X_train_sc,Y_train)
Y_test_predsDT=dt.predict(X_test_sc)
```

In [124…
```
print(classification_report(Y_test_predsDT,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsDT,Y_test)))
print("Decision Tree Accuracy {:.2f}% " .format(accuracy_score(Y_test_predsDT,Y_tes
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      2142
           1       0.92      0.90      0.91       682

    accuracy                           0.96      2824
   macro avg       0.94      0.94      0.94      2824
weighted avg       0.96      0.96      0.96      2824

      0    1
0  2088   54
1    65  617
Decision Tree Accuracy 95.79%
```

3. Random Forest

In [125…

```
from sklearn.ensemble import RandomForestClassifier
rt=RandomForestClassifier()
rt.fit(X_train_sc,Y_train)
Y_test_predsRF=rt.predict(X_test_sc)
print(classification_report(Y_test_predsRF,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsRF,Y_test)))
print("RandomForest Accuracy: {:.2f}%" .format(accuracy_score(Y_test_predsRF,Y_test
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      2195
           1       0.91      0.97      0.94       629

    accuracy                           0.97      2824
   macro avg       0.95      0.97      0.96      2824
weighted avg       0.97      0.97      0.97      2824

      0    1
0  2134   61
1    19  610
RandomForest Accuracy: 97.17%
```

4. Naive Bayes

In [126…

```
from sklearn.naive_bayes import GaussianNB
nbc=GaussianNB()
nbc.fit(X_train_sc,Y_train)
Y_test_predsNB=nbc.predict(X_test_sc)
print(classification_report(Y_test_predsNB,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsNB,Y_test)))
print("Naive Bayes Accuracy score: {:.2f}% ".format(accuracy_score(Y_test_predsNB,Y
```

```
              precision    recall  f1-score   support

           0       0.69      0.92      0.79      1606
           1       0.82      0.45      0.58      1218

    accuracy                           0.72      2824
   macro avg       0.75      0.69      0.69      2824
weighted avg       0.74      0.72      0.70      2824

      0    1
0  1484  122
1   669  549
Naive Bayes Accuracy score: 71.99%
```

5. K-Nearest Neighbor

In [127…

```
from sklearn.neighbors import KNeighborsClassifier

accuracy_list=[]

def generate():
    for i in range(1,30):
        knn_model=KNeighborsClassifier(n_neighbors=i)
        knn_model.fit(X_train_sc,Y_train)
        Y_preds_knn=knn_model.predict(X_test_sc)
        accuracy_list.append(accuracy_score(Y_preds_knn,Y_test))

    print("Accuracy and k Value", max([(v,i+1) for i,v in enumerate(accuracy_list)


generate()
```

```
Accuracy and k Value (0.9507790368271954, 2)
```

In [128…

```
# From the above codde, k value is determined as 2
# Creating model with k=2 again to check accuracy, precision, recall and other metr
knn_model=KNeighborsClassifier(n_neighbors=2,p=2,metric='minkowski')
knn_model.fit(X_train_sc,Y_train)
Y_test_predsKNN=knn_model.predict(X_test_sc)

print(classification_report(Y_test_predsKNN,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsKNN,Y_test)))
print("Naive Bayes Accuracy score: {:.2f}% ".format(accuracy_score(Y_test_predsKNN,
```

```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      2198
           1       0.86      0.92      0.89       626

    accuracy                           0.95      2824
   macro avg       0.92      0.94      0.93      2824
weighted avg       0.95      0.95      0.95      2824

      0    1
0  2106   92
1    47  579
Naive Bayes Accuracy score: 95.08%
```

6. Support Vector Machines

In [129…

```
from sklearn.svm import SVC
svc=SVC(random_state=0)
svc.fit(X_train_sc,Y_train)
Y_test_predsSVM=svc.predict(X_test_sc)

print(classification_report(Y_test_predsSVM,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsSVM,Y_test)))
print("Naive Bayes Accuracy score: {:.2f}% ".format(accuracy_score(Y_test_predsSVM,
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      2153
           1       0.90      0.90      0.90       671

    accuracy                           0.95      2824
   macro avg       0.93      0.93      0.93      2824
weighted avg       0.95      0.95      0.95      2824

      0    1
0  2083   70
1    70  601
Naive Bayes Accuracy score: 95.04%
```

7. Bagging Classifier

In [130…

```
from sklearn.ensemble import BaggingClassifier
dtc=DecisionTreeClassifier()
# base estimator/(weaklearner) is Decision Tree Classifier, number of base estimato
# number of samples to draw from X is 60%
bgc=BaggingClassifier(base_estimator=dtc,n_estimators=200,max_samples=0.6,random_st
bgc.fit(X_train_sc,Y_train)
Y_test_predsBG=bgc.predict(X_test_sc)

print(classification_report(Y_test_predsBG,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsBG,Y_test)))
print("Naive Bayes Accuracy score: {:.2f}% ".format(accuracy_score(Y_test_predsBG,Y
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      2195
           1       0.92      0.98      0.95       629

    accuracy                           0.98      2824
   macro avg       0.96      0.98      0.97      2824
weighted avg       0.98      0.98      0.98      2824

      0    1
0  2139   56
1    14  615
Naive Bayes Accuracy score: 97.52%
```

8. AdaBoost Classifier

In [131…

```
from sklearn.ensemble import AdaBoostClassifier
# default base estimator/weak leaner for AdaBoost is Decision Tree classifier with
abc=AdaBoostClassifier(n_estimators=500)
abc.fit(X_train_sc,Y_train)
Y_test_predsABC=abc.predict(X_test_sc)

print(classification_report(Y_test_predsABC,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsABC,Y_test)))
print("AdaBoost Accuracy score: {:.2f}% ".format(accuracy_score(Y_test_predsABC,Y_t
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      2100
           1       0.88      0.81      0.84       724

    accuracy                           0.92      2824
   macro avg       0.91      0.89      0.90      2824
weighted avg       0.92      0.92      0.92      2824

      0    1
0  2018   82
1   135  589
AdaBoost Accuracy score: 92.32%
```

9. Gradient Boosting

In [132…

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_sc

model_gb = GradientBoostingClassifier(n_estimators=50, random_state=1, learning_rat
model_gb.fit(X_train_sc,Y_train)

Y_test_predsGB=model_gb.predict(X_test_sc)

print(classification_report(Y_test_predsGB,Y_test))
print(pd.DataFrame(confusion_matrix(Y_test_predsGB,Y_test)))
print("AdaBoost Accuracy score: {:.2f}% ".format(accuracy_score(Y_test_predsGB,Y_te
#results_gb = model_selection.cross_validate(model_gb, , Y, cv=kfold, scoring=scori
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      2167
           1       0.91      0.93      0.92       657

    accuracy                           0.96      2824
   macro avg       0.94      0.95      0.95      2824
weighted avg       0.96      0.96      0.96      2824

      0    1
0  2106   61
1    47  610
AdaBoost Accuracy score: 96.18%
```

10. XG Boosting

In [133…

```
# pip install xgboost
```

In [134…

```
# if xgboost module is not available run the command-  pip install xgboost
from xgboost import XGBClassifier


XGB=XGBClassifier(n_estimators=100, random_state=1, learning_rate = 0.15)
XGB.fit(X_train_sc,Y_train)
y_test_predsXGBC = XGB.predict(X_test_sc)

print(classification_report(y_test_predsXGBC,Y_test))
print(pd.DataFrame(confusion_matrix(y_test_predsXGBC,Y_test)))
print("AdaBoost Accuracy score: {:.2f}% ".format(accuracy_score(y_test_predsXGBC,Y_
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      2182
           1       0.92      0.96      0.94       642

    accuracy                           0.97      2824
   macro avg       0.95      0.97      0.96      2824
weighted avg       0.97      0.97      0.97      2824

      0    1
0  2127   55
1    26  616
AdaBoost Accuracy score: 97.13%
```

Model Comparision Before Hyper Parameter Tuning

In [135…

```
compare_model = pd.DataFrame({'Model': ['Logistic Regression',"Decision","RandomFor
 , "K-Nearest Neighbour", " Bagging","AdaBoost","GradientBoosting","XGBoosting"],
 'Accuracy_Score': [format(accuracy_score(Y_test_predsLR,Y_test)*100),format(accura
                    format(accuracy_score(Y_test_predsRF,Y_test)*100),format(accura
                    format(accuracy_score(Y_test_predsNB,Y_test)*100),format(accura
                    format(accuracy_score(Y_test_predsBG,Y_test)*100),format(accura
                    format(accuracy_score(Y_test_predsGB,Y_test)*100),format(accura
 , "K-Nearest Neighbour", " Bagging","AdaBoost","GradientBoosting","XGBoosting"]})

compare_model.sort_values(by='Accuracy_Score')
```

Out[135]:

	Model	Accuracy_Score	Model with Tunig
4	NaiveBayes	71.9900849858357	NaiveBayes
0	Logistic Regression	79.35552407932012	Logistic Regression
7	AdaBoost	92.31586402266288	AdaBoost
3	Support Vector Machine	95.04249291784703	Support Vector Machine
5	K-Nearest Neighbour	95.07790368271955	K-Nearest Neighbour
1	Decision	95.78611898016997	Decision
8	GradientBoosting	96.17563739376772	GradientBoosting
9	XGBoosting	97.13172804532579	XGBoosting
2	RandomForest	97.16713881019831	RandomForest
6	Bagging	97.52124645892351	Bagging

- **Bagging** got the highest accuracy of **97.45** followed by **RandomForest, XGBoosting** with accuracies **97.41**,**97.13**

Hyper parameter Tuning

Decision Tree Classifier with Grid search

In [136…

```
from sklearn.model_selection import GridSearchCV
DecTreeClassifier=DecisionTreeClassifier(random_state=1)
tree_para=[{'criterion':['gini','entropy'],'max_depth':range(2,60),'max_features':[
```

```
grid_search_dtc=GridSearchCV(DecTreeClassifier,tree_para,cv=10,refit='AUC')
grid_search_dtc.fit(X_train_sc,Y_train)
```

Out[136]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(random_state=1),
             param_grid=[{'criterion': ['gini', 'entropy'],
                          'max_depth': range(2, 60),
                          'max_features': ['sqrt', 'log2', None]}],
             refit='AUC')
```

In [137…
```
y_test_predsDTCH = grid_search_dtc.predict(X_test_sc)

print(classification_report(y_test_predsDTCH,Y_test))
print(pd.DataFrame(confusion_matrix(y_test_predsDTCH,Y_test)))
print("Decision Tree Classifier wtih Grid Search: {:.2f}% ".format(accuracy_score(y
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      2183
           1       0.90      0.94      0.92       641

    accuracy                           0.96      2824
   macro avg       0.94      0.96      0.95      2824
weighted avg       0.96      0.96      0.96      2824

      0    1
0  2117   66
1    36  605
Decision Tree Classifier wtih Grid Search: 96.39%
```

Bagging Classifier with Grid Search

In [138…
```
param_grid={'base_estimator__max_depth':[1,2,3,4,5],
            'max_samples':[0.05,0.1,0.2,0.5]}
grid_search_bc=GridSearchCV(BaggingClassifier(DecisionTreeClassifier(),
                                              n_estimators=200,max_features=0.5),pa
grid_search_bc.fit(X_train_sc,Y_train)
```

Out[138]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(),
                                         max_features=0.5, n_estimators=200),
             param_grid={'base_estimator__max_depth': [1, 2, 3, 4, 5],
                         'max_samples': [0.05, 0.1, 0.2, 0.5]})
```

In [139…
```
y_test_predsBCGS=grid_search_bc.predict(X_test_sc)
```

In [140…
```
print(classification_report(y_test_predsBCGS,Y_test))
print(pd.DataFrame(confusion_matrix(y_test_predsBCGS,Y_test)))
print("AdaBoost Accuracy score: {:.2f}% ".format(accuracy_score(y_test_predsBCGS,Y_
```

```
              precision    recall  f1-score   support

           0       0.99      0.96      0.98      2215
           1       0.88      0.97      0.92       609

    accuracy                           0.97      2824
   macro avg       0.94      0.97      0.95      2824
weighted avg       0.97      0.97      0.97      2824

      0    1
0  2135   80
1    18  591
AdaBoost Accuracy score: 96.53%
```

RandomizedsearchCV with RandomForest

In [141…
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
model_rfcv=RandomForestClassifier(n_estimators=100,max_samples=0.60,random_state=1)
# parameters for GridSearchCV
# specify parameters and distributions to sample from
param_dist={"max_depth": range(2,5),
              "min_samples_split": sp_randint(2, 11),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "n_estimators": [100, 400, 700, 1000, 1500],
              "criterion" : ["gini", "entropy"],
              'max_features': ['sqrt', 'log2', None]
             }
# run randomized search

n_iter_search=50
random_search=RandomizedSearchCV(model_rfcv, param_distributions = param_dist,
                                 n_iter = n_iter_search,
                                 n_jobs = -1)
```

In [142…
```
random_search.fit(X_train_sc,Y_train)
```

Out[142]:
```
RandomizedSearchCV(estimator=RandomForestClassifier(max_samples=0.6,
                                                    random_state=1),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'criterion': ['gini', 'entropy'],
                                        'max_depth': range(2, 5),
                                        'max_features': ['sqrt', 'log2', None],
                                        'min_samples_leaf': <scipy.stats._distn_in
frastructure.rv_discrete_frozen object at 0x000001AD4174E550>,
                                        'min_samples_split': <scipy.stats._distn_i
nfrastructure.rv_discrete_frozen object at 0x000001AD49A5B490>,
                                        'n_estimators': [100, 400, 700, 1000,
                                                         1500]})
```

In [143…
```
y_test_predsRSCV=random_search.predict(X_test_sc)
```

In [144…
```
print(classification_report(y_test_predsRSCV,Y_test))
print(pd.DataFrame(confusion_matrix(y_test_predsRSCV,Y_test)))
print("AdaBoost Accuracy score: {:.2f}% ".format(accuracy_score(y_test_predsRSCV,Y_
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      2150
           1       0.90      0.90      0.90       674

    accuracy                           0.95      2824
   macro avg       0.94      0.94      0.94      2824
weighted avg       0.95      0.95      0.95      2824

      0    1
0  2086   64
1    67  607
AdaBoost Accuracy score: 95.36%
```

Support Vector Classifer with GridSearch

In [145…
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
Cs = [0.001,0.01,0.1,1,10]
gammas=[0.001,0.01,0.1,1]

param_grid = {'C': Cs,'gamma':gammas}
grid_search = GridSearchCV(SVC(), param_grid, cv=10)
grid_search.fit(X_train_sc,Y_train)
grid_search.best_params_
```

Out[145]:
```
{'C': 10, 'gamma': 1}
```

In [146…
```
model=SVC(C=10,gamma=1)
model.fit(X_train_sc,Y_train)
y_pred_svc_cv = model.predict(X_test_sc)
```

In [147…
```
print(classification_report(y_pred_svc_cv,Y_test))
print(pd.DataFrame(confusion_matrix(y_pred_svc_cv,Y_test)))
print("Support Vector Classifier with Grid Search score: {:.2f}% ".format(accuracy_
```

```
              precision    recall  f1-score   support

           0       0.99      0.96      0.97      2208
           1       0.87      0.95      0.91       616

    accuracy                           0.96      2824
   macro avg       0.93      0.96      0.94      2824
weighted avg       0.96      0.96      0.96      2824

      0    1
0  2124   84
1    29  587
Support Vector Classifier with Grid Search score: 96.00%
```

Fit and Tune with Cross Validation

Logistic Regression with CV

In [148…
```
from sklearn import model_selection
kfold = model_selection.KFold(n_splits=10,shuffle=True,random_state=2)
modelLR = LogisticRegression()
results = model_selection.cross_val_score(modelLR, X_train_sc, Y_train, cv=kfold, s
print("10-fold cross validation average Accuracy score for Logistic Regression: %.3
```

```
10-fold cross validation average Accuracy score for Logistic Regression: 0.793
```

In [149…
```
results = model_selection.cross_val_score(modelLR, X_test_sc, Y_test, cv=kfold, scc
print("10-fold cross validation average roc_auc score for Logistic Regression: %.3f
```

```
10-fold cross validation average roc_auc score for Logistic Regression: 0.830
```

Support Vector Classifier with CV

In [150…
```
from sklearn.model_selection import cross_val_score

svc = SVC(kernel='rbf',random_state = 0)
scores = cross_val_score(svc,X_s, Y, cv=10, scoring='accuracy')
print(scores.mean())
```

```
0.9477193560109459
```

Bagging Classifier with CV

In [151…

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
logreg = LogisticRegression()

kfold = KFold(n_splits=10) # change this during train-test split method
#cart = DecisionTreeClassifier()

model_k1 = BaggingClassifier(base_estimator=logreg, n_estimators=100, random_state=

results_k1 = cross_val_score(model_k1, X_s, Y, cv=kfold)

print('test accuracy: ',results_k1.mean())
```

```
test accuracy:  0.7442855407632865
```

Decision Tree Classifier with CV

In [152…

```
cart = DecisionTreeClassifier()

kfold = KFold(n_splits=10)

model_k2 = BaggingClassifier(base_estimator=cart, n_estimators=100, random_state=1)

results_k2 = cross_val_score(model_k2, X_s, Y, cv=kfold)

print('test accuracy: ',results_k2.mean())
```

```
test accuracy:  0.9722332422507897
```

AdaBoost Classifier

In [153…

```
score = []
for depth in [1,2,10] :
    reg_ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=depth))
    scores_ada = cross_val_score(reg_ada, X_s, Y, cv=10)
    score.append(scores_ada.mean())
print(score)
```

```
[0.9292973761401212, 0.9633739758233066, 0.974992973058707]
```

XGB Classifier

In [154…

```
kfold = KFold(n_splits=10)
XGB_cv = XGBClassifier(n_estimators=100, random_state=1, learning_rate = 0.1)
results_xgb = cross_val_score(XGB_cv, X_s, Y, cv=kfold)

print('test accuracy: ',results_xgb.mean())
```

```
test accuracy:  0.9729413069709265
```

Evaluate and Finalise the Model

Model Comparision After Finetuning

Algorithms	Accuracy Score
Decision Tree Classifier with Grid Search	**96.3%**
Bagging Classifier with Grid Search	**96.5%**
RandomizedSearchCV with RandomForest	**95.3%**
Support Vector Machines with Grid Searc	**96%**
Logistic Regression with CV	**83%**
Support Vector Classifier with CV	**94.7%**
Bagging Classifier with CV	**74.4%**
Decision Tress Classifier with CV	**97.2%**
AdaBoost Classifier with CV	**92.9%**
XGB Classifier with CVLogistic Regression	**97.4%**

Finalising the Model

- XGB classifer with highest accuracy of 97.4 is chosen as the final model.

Applying model on Unseen Data

In [155…
```
data_unseen = pd.read_csv('https://raw.githubusercontent.com/sruthayanan/INSAID_Cap
data_unseen.head()
```

Out[155]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promoted	
0	134	D00-IT	NaN	0.528841	2	NaN	
1	221	D00-PD	NaN	0.784561	2	NaN	
2	156	D00-SS	NaN	0.545183	2	NaN	r
3	133	D00-PR	NaN	NaN	4	NaN	r
4	135	D00-SS	NaN	0.454637	2	NaN	

In [156…
```
data_unseen.shape
```

Out[156]:
```
(100, 10)
```

In [157…
```
data_unseen.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   avg_monthly_hrs    100 non-null    int64  
 1   department         98 non-null     object 
 2   filed_complaint    15 non-null     float64
 3   last_evaluation    83 non-null     float64
 4   n_projects         100 non-null    int64  
 5   recently_promoted  3 non-null      float64
 6   salary             100 non-null    object 
 7   satisfaction       97 non-null     float64
 8   tenure             97 non-null     float64
 9   employee_id        100 non-null    int64  
dtypes: float64(5), int64(3), object(2)
memory usage: 7.9+ KB
```

In [158…
```
employee_details_data.head()
```

Out[158]:

	employee_id	age	gender	marital_status
0	113558	43	Male	Married
1	112256	24	Female	Unmarried
2	112586	22	Female	Unmarried
3	108071	36	Male	Married
4	116915	38	Male	Married

In [159…
```
common1 = np.intersect1d(data_unseen.employee_id,employee_details_data.employee_id)
common1.shape
```

Out[159]:
```
(100,)
```

In [160…
```
data_pred = pd.merge(left=data_unseen,right=employee_details_data, how='inner',on='
data_pred.head()
```

Out[160]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promoted	
0	134	D00-IT	NaN	0.528841	2	NaN	
1	221	D00-PD	NaN	0.784561	2	NaN	
2	156	D00-SS	NaN	0.545183	2	NaN	r
3	133	D00-PR	NaN	NaN	4	NaN	r
4	135	D00-SS	NaN	0.454637	2	NaN	

In [161…
```
data_pred.shape
```

Out[161]:
```
(100, 13)
```

In [162…
```
data_pred.isnull().sum().sort_values(ascending = False)
```

Out[162]:
```
recently_promoted    97
filed_complaint      85
last_evaluation      17
satisfaction          3
tenure                3
department            2
avg_monthly_hrs       0
n_projects            0
salary                0
employee_id           0
age                   0
gender                0
marital_status        0
dtype: int64
```

Missing Value Treatment

In [163…
```
data_pred.recently_promoted.value_counts()
```

Out[163]:
```
1.0    3
Name: recently_promoted, dtype: int64
```

- Only **three** were promoted. Remaining all the rows have NAN values
- We will replace **NAN** values with zeros

In [164…
```
data_pred.recently_promoted.head()
```

Out[164]:
```
0   NaN
1   NaN
2   NaN
3   NaN
4   NaN
Name: recently_promoted, dtype: float64
```

In [165…
```
data_pred.filed_complaint.value_counts()
```

Out[165]:
```
1.0    15
Name: filed_complaint, dtype: int64
```

- We will replace **NAN** values with zeros

In [166…
```
data_pred[['recently_promoted','filed_complaint']] = data_pred[['recently_promoted'
```

In [167…
```
data_pred.tenure.value_counts()
```

Out[167]:
```
3.0     40
2.0     27
4.0     11
5.0     10
6.0      5
10.0     2
7.0      2
Name: tenure, dtype: int64
```

- we will replace **NAN** values with 1

In [168…
```
data_pred.department.value_counts()
```

Out[168]:
```
D00-SS     27
D00-ENG    17
D00-SP     13
D00-IT      9
D00-PD      7
D00-MT      7
D00-FN      6
D00-MN      5
D00-TP      3
-IT         2
D00-PR      1
D00-AD      1
Name: department, dtype: int64
```

In [169…
```
data_pred.replace({'-IT'},{'D00-IT'},inplace =True)
```

In [170…
```
data_pred['department'] = data_pred['department'].replace(np.NaN,'D00-UN')
```

In [171…
```
data_pred['tenure'] = data_pred['tenure'].replace(np.NaN,1)
```

In [172…
```
data_pred['satisfaction'] = data_pred['satisfaction'].replace(np.NaN,0)
```

In [173…
```
# Calculating the mean of last_evaluation for each department,
means = data_pred.groupby('department')['last_evaluation'].mean()
# Convert pandas series to dictionary
dd=means.to_dict()
```

In [174…
```
# replacing the null values of last_evaluation with mean of that department.
data_pred.last_evaluation = data_pred.last_evaluation.fillna(data_pred.department.m
```

In [175…
```
data_pred['last_evaluation'] = data_pred['last_evaluation'].fillna(0)
```

In [176…
```
null_data2 = pd.DataFrame(data_pred.isnull().sum(), columns = ['Frequency'])
null_data2.transpose()
```

Out[176]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_pro
Frequency	0	0	0	0	0	

In [177…
```
data_pred.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 0 to 99
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   avg_monthly_hrs    100 non-null    int64  
 1   department         100 non-null    object 
 2   filed_complaint    100 non-null    float64
 3   last_evaluation    100 non-null    float64
 4   n_projects         100 non-null    int64  
 5   recently_promoted  100 non-null    float64
 6   salary             100 non-null    object 
 7   satisfaction       100 non-null    float64
 8   tenure             100 non-null    float64
 9   employee_id        100 non-null    int64  
 10  age                100 non-null    int64  
 11  gender             100 non-null    object 
 12  marital_status     100 non-null    object 
dtypes: float64(5), int64(4), object(4)
memory usage: 10.9+ KB
```

In [178…
```
data_pred.nunique()
```

Out[178]:
```
avg_monthly_hrs       80
department            12
filed_complaint        2
last_evaluation       88
n_projects             7
recently_promoted      2
salary                 3
satisfaction          96
tenure                 8
employee_id          100
age                   30
gender                 2
marital_status         2
dtype: int64
```

Feature Engineering of Unseen Data

Encoding

In [179…
```
# Label Encoding
from sklearn.preprocessing import LabelEncoder
Label_Encoder = LabelEncoder()
Label_Encoder.fit(data_pred['department'])
data_pred['department'] = Label_Encoder.transform(data_pred['department'])
```

In [180…
```
#One-Hot Encoding
categorical_feature_oh=['salary','gender','marital_status']
data_pred=pd.get_dummies(data_pred,columns = categorical_feature_oh)

data_pred.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 0 to 99
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   avg_monthly_hrs          100 non-null    int64
 1   department               100 non-null    int32
 2   filed_complaint          100 non-null    float64
 3   last_evaluation          100 non-null    float64
 4   n_projects               100 non-null    int64
 5   recently_promoted        100 non-null    float64
 6   satisfaction             100 non-null    float64
 7   tenure                   100 non-null    float64
 8   employee_id              100 non-null    int64
 9   age                      100 non-null    int64
 10  salary_high              100 non-null    uint8
 11  salary_low               100 non-null    uint8
 12  salary_medium            100 non-null    uint8
 13  gender_Female            100 non-null    uint8
 14  gender_Male              100 non-null    uint8
 15  marital_status_Married   100 non-null    uint8
 16  marital_status_Unmarried 100 non-null    uint8
dtypes: float64(5), int32(1), int64(4), uint8(7)
memory usage: 8.9 KB
```

In [181…
```
data_pred.head()
```

Out[181]:

	avg_monthly_hrs	department	filed_complaint	last_evaluation	n_projects	recently_promoted	s
0	134	3	0.0	0.528841	2	0.0	
1	221	6	0.0	0.784561	2	0.0	
2	156	9	0.0	0.545183	2	0.0	
3	133	7	0.0	0.000000	4	0.0	
4	135	9	0.0	0.454637	2	0.0	

In [182…
```
X_valid = data_pred.drop(columns = 'employee_id')
```

In [183…
```
print(X_valid.shape)
```

```
(100, 16)
```

In [184…
```
X_valid_SC = scaler.transform(X_valid)
```

XGB Model on Unseen Data

In [185…
```
y_pred_unseenData = XGB.predict(X_valid_SC)
y_pred_unseenData_probability = XGB.predict_proba(X_valid_SC)
predictions = y_pred_unseenData_probability[:,1]
```

In [186…
```
print (y_pred_unseenData.shape, y_pred_unseenData_probability.shape, predictions.sh
```

```
(100,) (100, 2) (100,)
```

In [187…
```
df_y_preds = pd.DataFrame(y_pred_unseenData, columns=['status'])
df_y_pred_proba = pd.DataFrame(predictions, columns=['Probability to Leave'])
```

In [188…
```
df_y_preds.head()
```

Out[188]:

	status
0	1
1	0
2	1
3	0
4	0

In [189…
```
df_y_preds.status.value_counts()
```

Out[189]:
```
0    72
1    28
Name: status, dtype: int64
```

In [190…
```
df_y_pred_proba.head()
```

Out[190]:

	Probability to Leave
0	0.994174
1	0.054260
2	0.986490
3	0.000450
4	0.205264

In [191…
```
emp_id = data_pred[['employee_id']]
emp_id.shape
```

Out[191]:
```
(100, 1)
```

In [192…
```
df_submission = pd.concat([emp_id, df_y_pred_proba], axis=1)
```

In [193…
```
df_submission
```

Out[193]:

	employee_id	Probability to Leave
0	119045	0.994174
1	112223	0.054260
2	103924	0.986490
3	105922	0.000450
4	103297	0.205264
...	...	...
95	116666	0.009632
96	106422	0.004616
97	107889	0.000955
98	117622	0.007621
99	108058	0.091560

100 rows × 2 columns

In [194...
```
df_submission.to_csv('C:\\Users\\Dell\\Downloads\\Capstone2-HRAnalytics.csv', heade
```

In [195...
```
test = pd.read_csv('Capstone2-HRAnalytics.csv',header=None)
test.head()
```

Out[195]:

	0	1
0	119045	0.994174
1	112223	0.054260
2	103924	0.986490
3	105922	0.000450
4	103297	0.205264

In [196...
```
test.shape
```

Out[196]:
```
(100, 2)
```

Actionable Insights

- **Delay in Promotion** could be the reason for employees to leave the organisation.
- Satisfaction level of left employees is much below average in comparison to those who are still employed.
- Employees who left had higher number of average working hours per month
- Average age of employees who left is younger in comparison to those still employed
- Average of employees promoted among those left is less compared to those still employed
- Number of projects worked upon by both who left and still employed remains almost the same,
- Majority of unmarried females are in the age group 22-25, whereas males are in 22-29.

- The number of females leaving in this age group is also more compared to males. For females- ~250(Left) vs. ~400(Employed), males- ~120(Left) vs. 450(Employed)
- 14.46% of employees filed complaints which is quite a high number and this also shows that employees are not happy in the organisation.

"EDA of the dataset suggests that Employee who left the company can be grouped into 3 types of employees:

1) High Evaluation rating, but low satisfaction 2) High Evaluation rating and high satisfaction 3) Medium to low Evaluation rating and medium and low satisfaction

Actionable Insight:

- From above observations on 'employees promoted' and 'tenure' of employees, it is envisaged that the company should have a promotion policy in such a way that as soon as the employee is nearing 3 to 4 years of tenure, he/she should get a fair chance for promotion/incentives/hike in salary etc.

- The company should focus on retaining employess in the age group from 22-29

  By introducing career guidance for future growth of younger employees

  Implementing women friendly policies to retain Female workforce as their attrition rate is proportioantely more compare to male

- The grievance redressal mechanism needs to be made strong to promptly address the employees complaints/issues.
- Performance based incentives to be given to improve the employee satisfaction level.