Employee_turnover

July 18, 2023

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

[2]:
```
df = pd.read_excel("hr_comma_sep.xlsx")
```

[3]:
```
df.head(2)
```

[3]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   sales                  14999 non-null  object
 9   salary                 14999 non-null  object
```

```
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

[5]:
```
df.isna().sum()
```

[5]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
sales                    0
salary                   0
dtype: int64
```

[6]:
```
df["left"].unique()
```

[6]:
```
array([1, 0])
```

[7]:
```
df["promotion_last_5years"].unique()
```

[7]:
```
array([0, 1])
```

[8]:
```
df["number_project"].unique()
```

[8]:
```
array([2, 5, 7, 6, 4, 3])
```

[9]:
```
df.satisfaction_level.unique()
```

[9]:
```
array([0.38, 0.8 , 0.11, 0.72, 0.37, 0.41, 0.1 , 0.92, 0.89, 0.42, 0.45,
       0.84, 0.36, 0.78, 0.76, 0.09, 0.46, 0.4 , 0.82, 0.87, 0.57, 0.43,
       0.13, 0.44, 0.39, 0.85, 0.81, 0.9 , 0.74, 0.79, 0.17, 0.24, 0.91,
       0.71, 0.86, 0.14, 0.75, 0.7 , 0.31, 0.73, 0.83, 0.32, 0.54, 0.27,
       0.77, 0.88, 0.48, 0.19, 0.6 , 0.12, 0.61, 0.33, 0.56, 0.47, 0.28,
       0.55, 0.53, 0.59, 0.66, 0.25, 0.34, 0.58, 0.51, 0.35, 0.64, 0.5 ,
       0.23, 0.15, 0.49, 0.3 , 0.63, 0.21, 0.62, 0.29, 0.2 , 0.16, 0.65,
       0.68, 0.67, 0.22, 0.26, 0.99, 0.98, 1.  , 0.52, 0.93, 0.97, 0.69,
       0.94, 0.96, 0.18, 0.95])
```

[10]:
```
df.last_evaluation.unique()
```

[10]:
```
array([0.53, 0.86, 0.88, 0.87, 0.52, 0.5 , 0.77, 0.85, 1.  , 0.54, 0.81,
       0.92, 0.55, 0.56, 0.47, 0.99, 0.51, 0.89, 0.83, 0.95, 0.57, 0.49,
       0.46, 0.62, 0.94, 0.48, 0.8 , 0.74, 0.7 , 0.78, 0.91, 0.93, 0.98,
       0.97, 0.79, 0.59, 0.84, 0.45, 0.96, 0.68, 0.82, 0.9 , 0.71, 0.6 ,
```

```
       0.65, 0.58, 0.72, 0.67, 0.75, 0.73, 0.63, 0.61, 0.76, 0.66, 0.69,
       0.37, 0.64, 0.39, 0.41, 0.43, 0.44, 0.36, 0.38, 0.4 , 0.42])
```

[11]:
```
df.time_spend_company.unique()
```

[11]:
```
array([ 3,  6,  4,  5,  2,  8, 10,  7])
```

[12]:
```
df.Work_accident.unique()
```

[12]:
```
array([0, 1])
```

[13]:
```
df.sales.unique()
```

[13]:
```
array(['sales', 'accounting', 'hr', 'technical', 'support', 'management',
       'IT', 'product_mng', 'marketing', 'RandD'], dtype=object)
```

[14]:
```
df.salary.unique()
```

[14]:
```
array(['low', 'medium', 'high'], dtype=object)
```

[15]:
```
df.corr()
```

[15]:

	satisfaction_level	last_evaluation	number_project	\
satisfaction_level	1.000000	0.105021	-0.142970	
last_evaluation	0.105021	1.000000	0.349333	
number_project	-0.142970	0.349333	1.000000	
average_montly_hours	-0.020048	0.339742	0.417211	
time_spend_company	-0.100866	0.131591	0.196786	
Work_accident	0.058697	-0.007104	-0.004741	
left	-0.388375	0.006567	0.023787	
promotion_last_5years	0.025605	-0.008684	-0.006064	

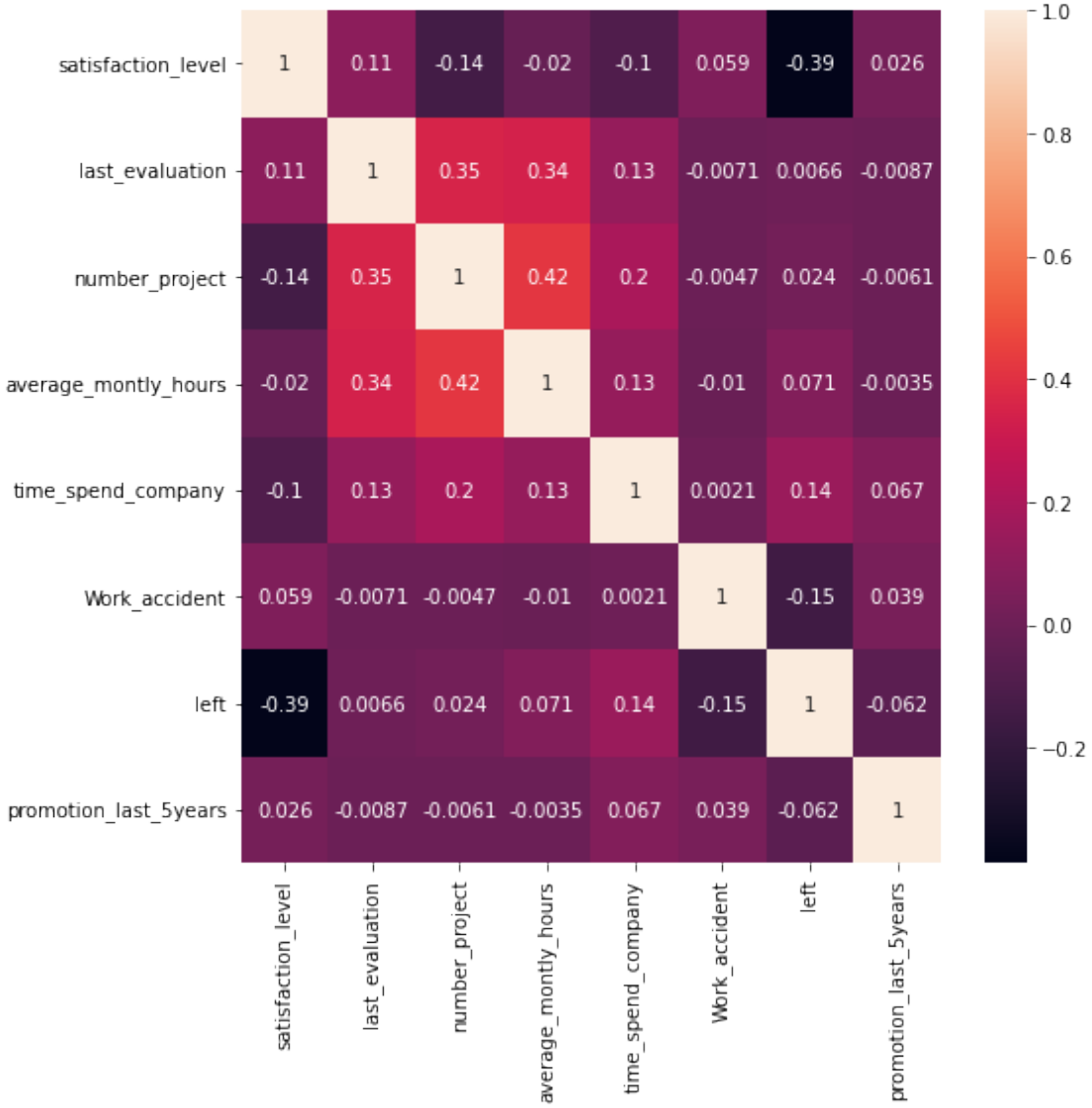
	average_montly_hours	time_spend_company	\
satisfaction_level	-0.020048	-0.100866	
last_evaluation	0.339742	0.131591	
number_project	0.417211	0.196786	
average_montly_hours	1.000000	0.127755	
time_spend_company	0.127755	1.000000	
Work_accident	-0.010143	0.002120	
left	0.071287	0.144822	
promotion_last_5years	-0.003544	0.067433	

	Work_accident	left	promotion_last_5years
satisfaction_level	0.058697	-0.388375	0.025605
last_evaluation	-0.007104	0.006567	-0.008684
number_project	-0.004741	0.023787	-0.006064
average_montly_hours	-0.010143	0.071287	-0.003544

```
time_spend_company          0.002120  0.144822              0.067433
Work_accident               1.000000 -0.154622              0.039245
left                       -0.154622  1.000000             -0.061788
promotion_last_5years       0.039245 -0.061788              1.000000
```

[16]:
```
plt.figure(figsize=(8,8))
sns.heatmap(df.corr(),annot=True)
```

[16]: <AxesSubplot:>



[17]:
```
df1= df.groupby(["sales"])["left"].value_counts().reset_index(name="count")
df1=pd.DataFrame(df1)
```

[18]:
```
df["sales"].value_counts()
```

[18]:
```
sales          4140
technical      2720
support        2229
IT             1227
product_mng     902
marketing       858
RandD           787
accounting      767
hr              739
management      630
Name: sales, dtype: int64
```

[19]:
```
dft=df["sales"].value_counts().reset_index(name="Total")
```

[20]:
```
dft=dft.rename(columns={"index":"sales"})
```

[21]:
```
dft
```

[21]:

	sales	Total
0	sales	4140
1	technical	2720
2	support	2229
3	IT	1227
4	product_mng	902
5	marketing	858
6	RandD	787
7	accounting	767
8	hr	739
9	management	630

[22]:
```
dfmer=df1.merge(dft,how="left")
```

[23]:
```
dfmer
```

[23]:

	sales	left	count	Total
0	IT	0	954	1227
1	IT	1	273	1227
2	RandD	0	666	787
3	RandD	1	121	787
4	accounting	0	563	767
5	accounting	1	204	767
6	hr	0	524	739
7	hr	1	215	739
8	management	0	539	630
9	management	1	91	630

```
10    marketing     0    655    858
11    marketing     1    203    858
12  product_mng     0    704    902
13  product_mng     1    198    902
14        sales     0   3126   4140
15        sales     1   1014   4140
16      support     0   1674   2229
17      support     1    555   2229
18    technical     0   2023   2720
19    technical     1    697   2720
```

[24]:
```
dfmer["normal"]=dfmer["count"].div(dfmer["Total"].values)
dfmer["normal"]=dfmer["normal"]*100
```

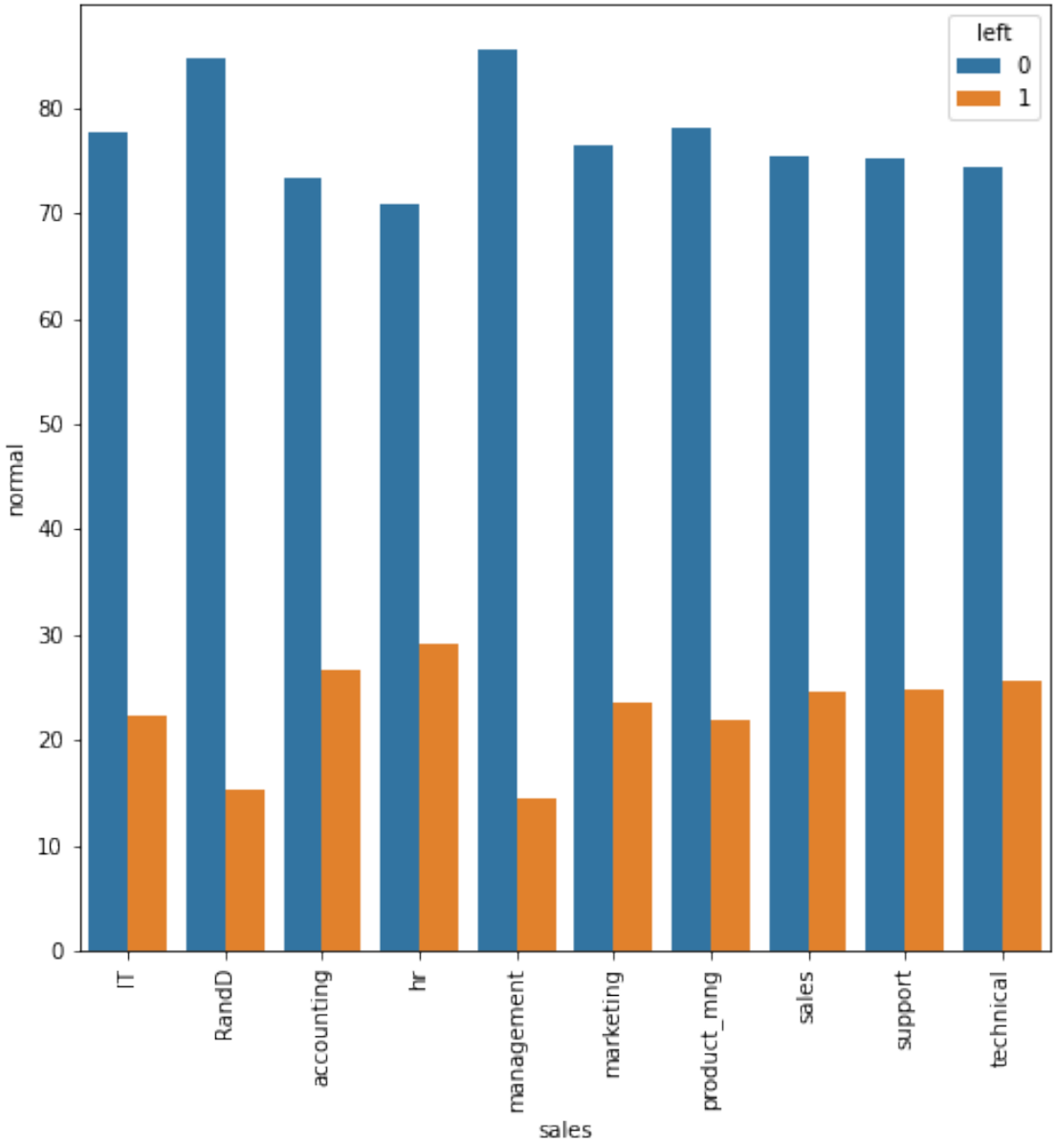
[25]:
```
dfmer
```

[25]:
```
          sales  left  count  Total     normal
0            IT     0    954   1227  77.750611
1            IT     1    273   1227  22.249389
2         RandD     0    666    787  84.625159
3         RandD     1    121    787  15.374841
4    accounting     0    563    767  73.402868
5    accounting     1    204    767  26.597132
6            hr     0    524    739  70.906631
7            hr     1    215    739  29.093369
8    management     0    539    630  85.555556
9    management     1     91    630  14.444444
10    marketing     0    655    858  76.340326
11    marketing     1    203    858  23.659674
12  product_mng     0    704    902  78.048780
13  product_mng     1    198    902  21.951220
14        sales     0   3126   4140  75.507246
15        sales     1   1014   4140  24.492754
16      support     0   1674   2229  75.100942
17      support     1    555   2229  24.899058
18    technical     0   2023   2720  74.375000
19    technical     1    697   2720  25.625000
```

[26]:
```
plt.figure(figsize=(8,8))
sns.barplot(x="sales",y='normal',hue="left",data=dfmer)
plt.xticks(rotation=90)
```

[26]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'IT'),
  Text(1, 0, 'RandD'),
  Text(2, 0, 'accounting'),
  Text(3, 0, 'hr'),
```

```
Text(4, 0, 'management'),
Text(5, 0, 'marketing'),
Text(6, 0, 'product_mng'),
Text(7, 0, 'sales'),
Text(8, 0, 'support'),
Text(9, 0, 'technical')])
```



People from the hr department are leaving the highest based on the normalized data.The Hr department has the highest percentage. Normal = (Count of people from leaving category in a department)/(Total number of people in that department)*100

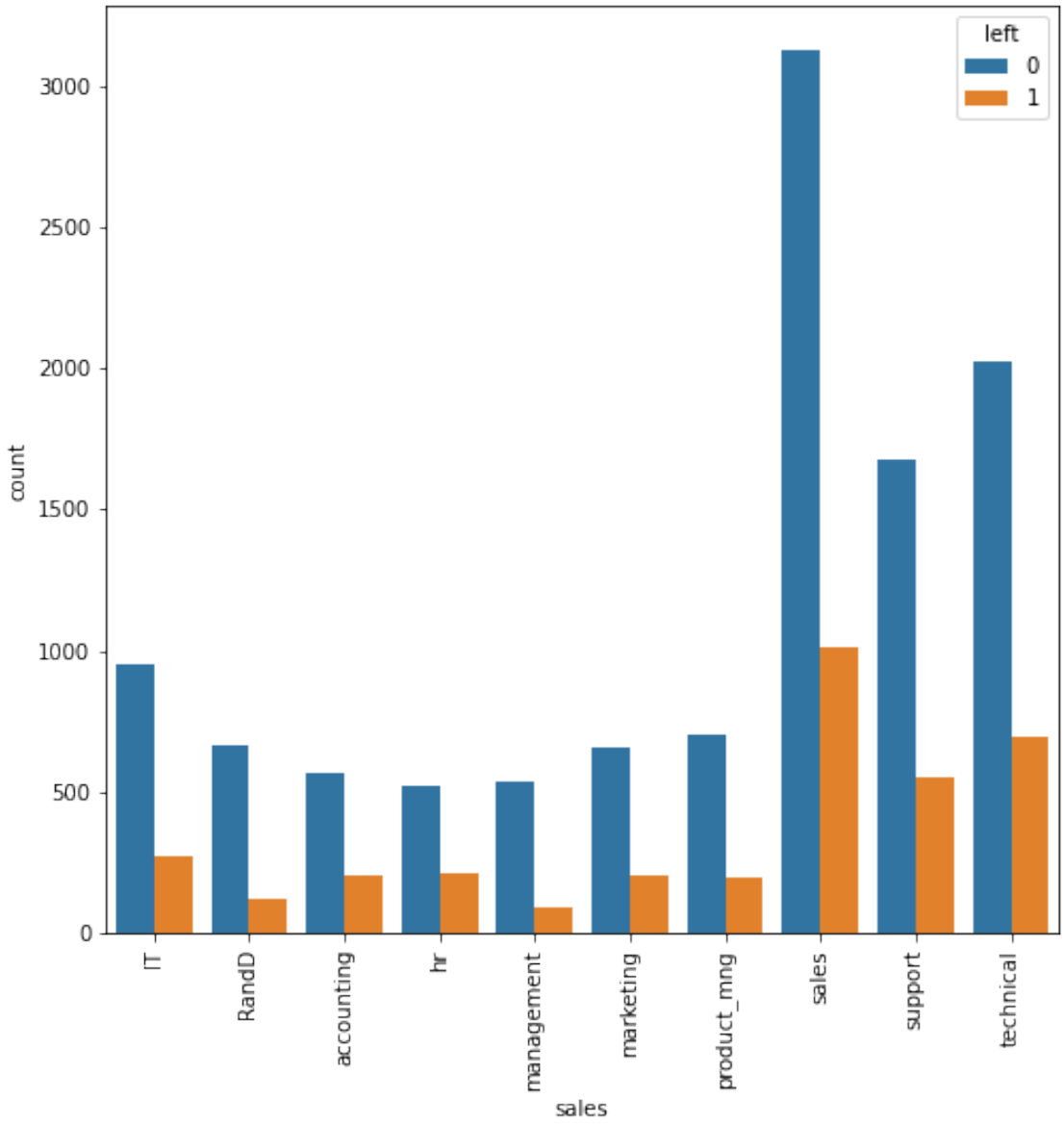
[27]:
```
df1.head()
```

[27]:

	sales	left	count
0	IT	0	954
1	IT	1	273
2	RandD	0	666
3	RandD	1	121
4	accounting	0	563

[28]:
```
plt.figure(figsize=(8,8))
sns.barplot(x="sales",y='count',hue="left",data=df1)
plt.xticks(rotation=90)
```

[28]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'IT'),
  Text(1, 0, 'RandD'),
  Text(2, 0, 'accounting'),
  Text(3, 0, 'hr'),
  Text(4, 0, 'management'),
  Text(5, 0, 'marketing'),
  Text(6, 0, 'product_mng'),
  Text(7, 0, 'sales'),
  Text(8, 0, 'support'),
  Text(9, 0, 'technical')])
```



The people from the sales department are leaing the highest if we look at only the count of leaving people.

[29]:
```
df2= df.groupby(["salary"])["left"].value_counts().reset_index(name="count")
df2=pd.DataFrame(df2)
```

[30]:
```
df2.head()
```

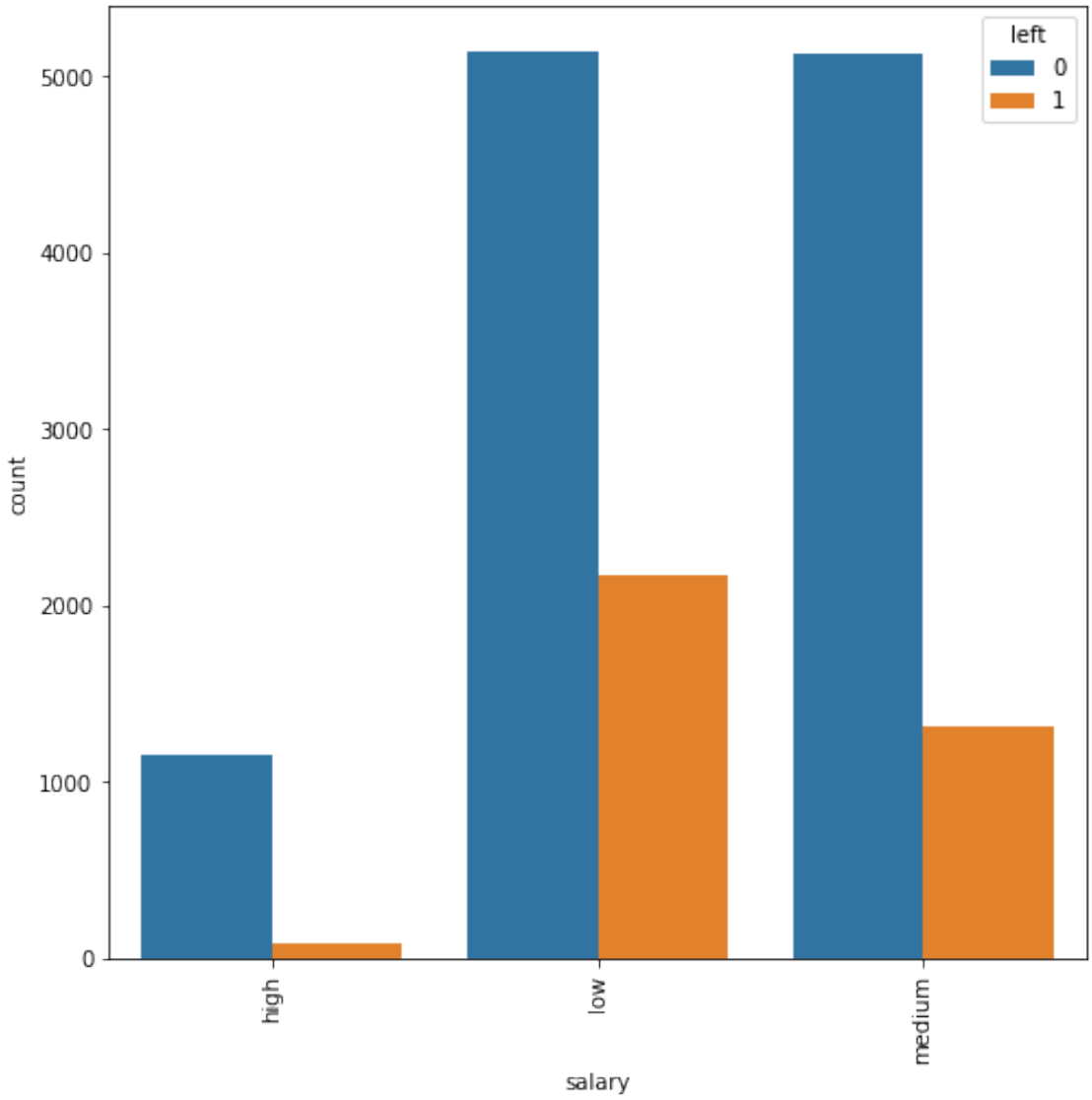
[30]:

	salary	left	count
0	high	0	1155

```
1    high     1     82
2     low     0   5144
3     low     1   2172
4  medium     0   5129
```

[31]:
```
plt.figure(figsize=(8,8))
sns.barplot(x="salary",y='count',hue="left",data=df2)
plt.xticks(rotation=90)
```

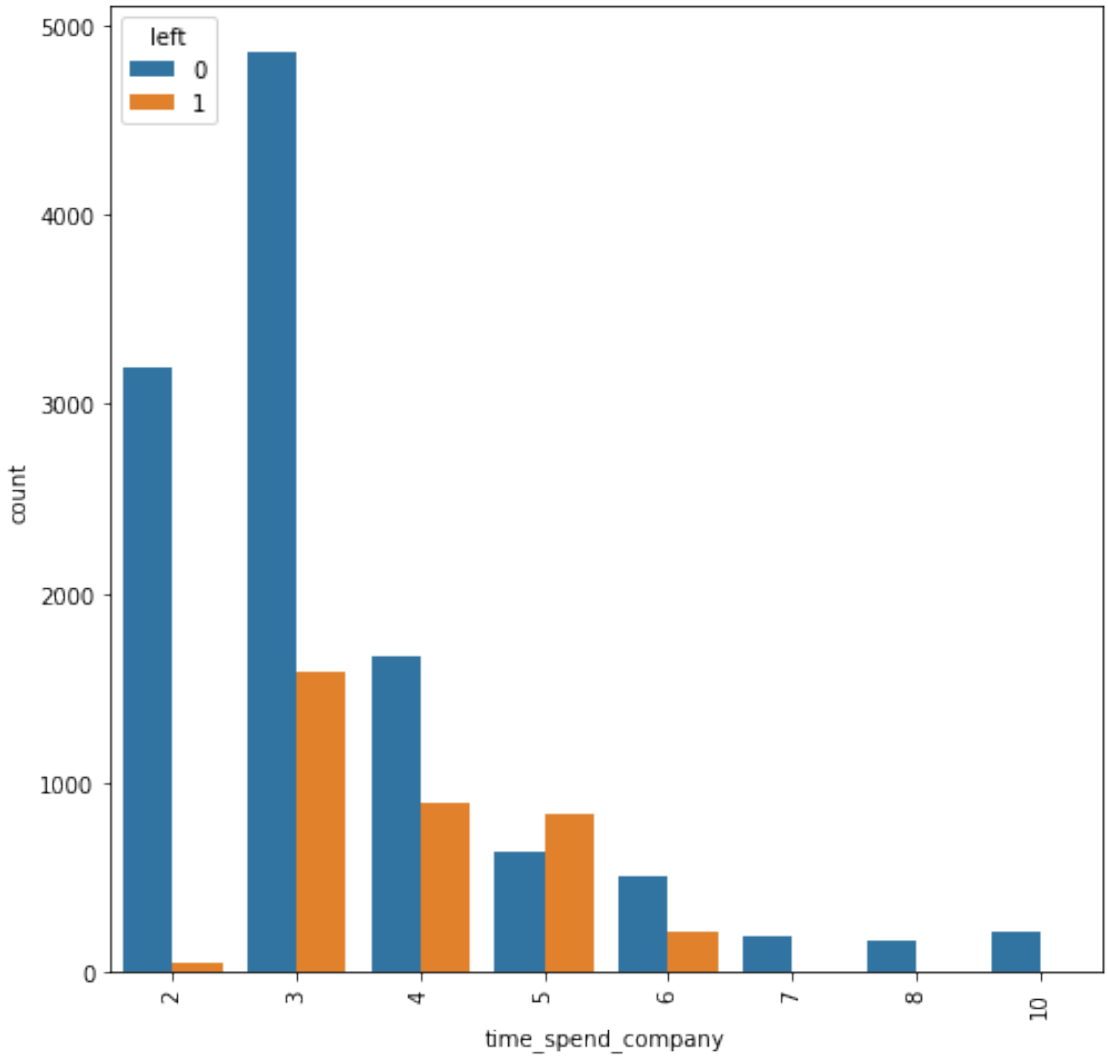
[31]:
```
(array([0, 1, 2]),
 [Text(0, 0, 'high'), Text(1, 0, 'low'), Text(2, 0, 'medium')])
```



[32]:
```
df3= df.groupby(["time_spend_company"])["left"].value_counts().
 ↪reset_index(name="count")
df3=pd.DataFrame(df3)
```

[33]:
```
#time_spend_company
plt.figure(figsize=(8,8))
sns.barplot(x="time_spend_company",y='count',hue="left",data=df3)
plt.xticks(rotation=90)
```

[33]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 [Text(0, 0, '2'),
  Text(1, 0, '3'),
  Text(2, 0, '4'),
  Text(3, 0, '5'),
  Text(4, 0, '6'),
  Text(5, 0, '7'),
  Text(6, 0, '8'),
  Text(7, 0, '10')])
```

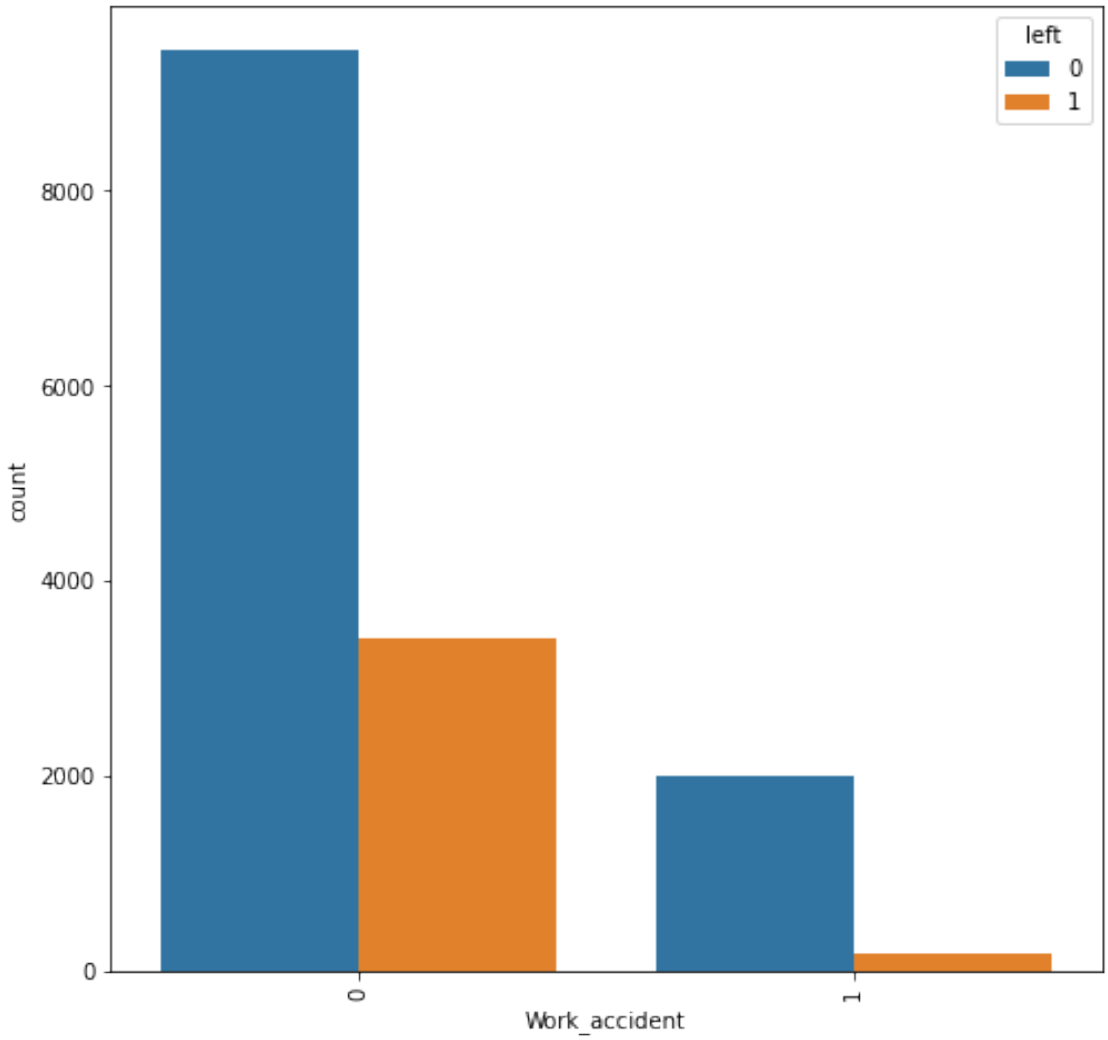


People with experience of 3 to 5 years are leaving the comapny more.

[34]:
```
plt.figure(figsize=(8,8))
sns.countplot("Work_accident",hue="left",data=df)
plt.xticks(rotation=90)
```

```
/usr/local/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

[34]: `(array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])`

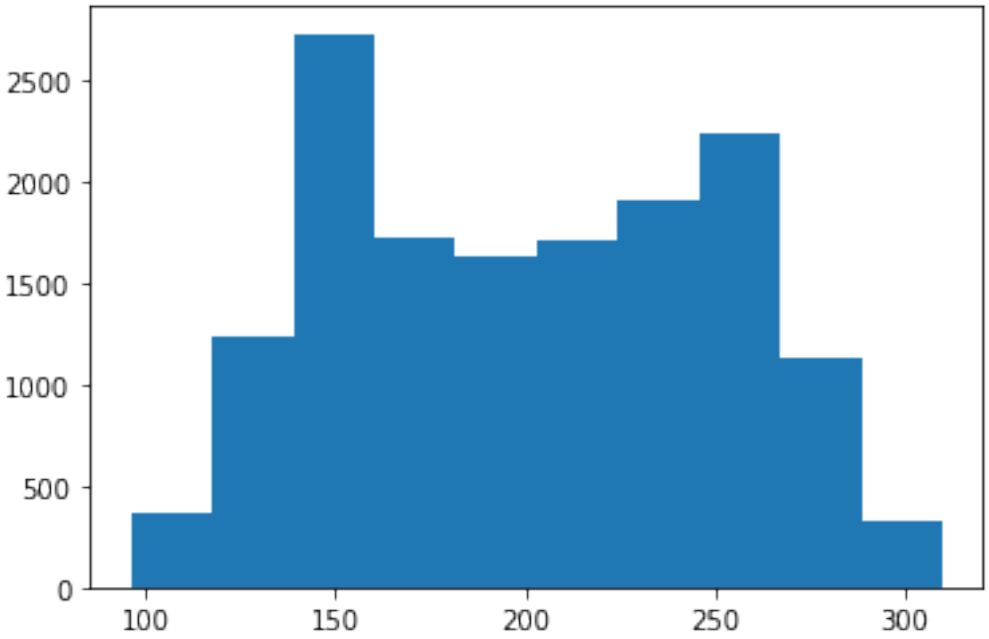


[35]:
```
df.columns
```

[35]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
       'promotion_last_5years', 'sales', 'salary'],
      dtype='object')
```

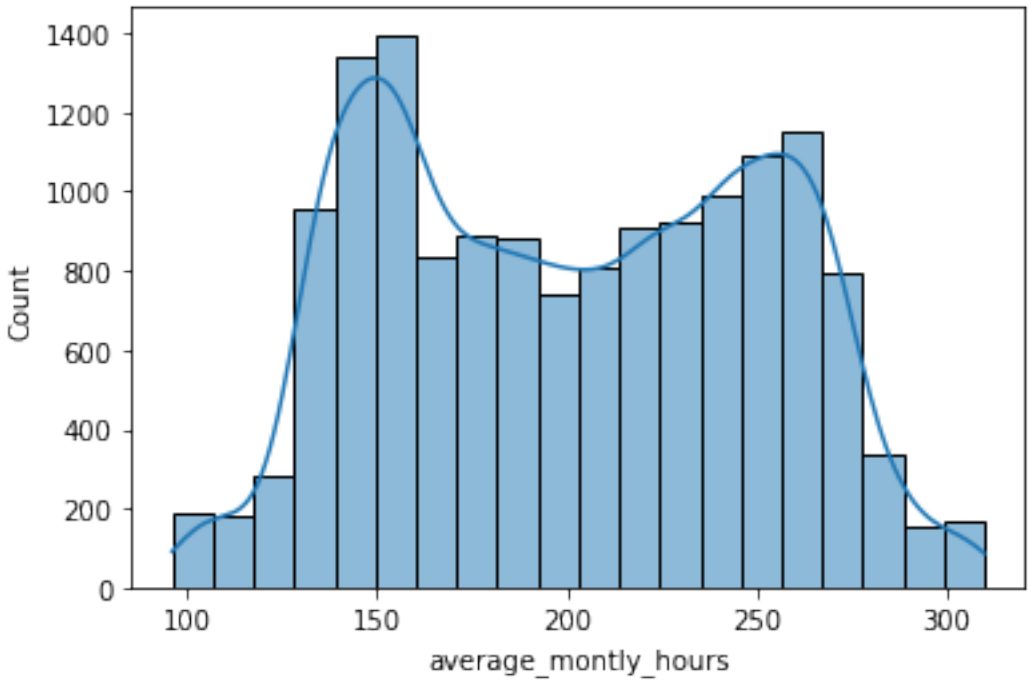
[36]:
```
plt.hist(df["average_montly_hours"])
```

[36]:
```
(array([ 367., 1240., 2733., 1722., 1628., 1712., 1906., 2240., 1127.,
         324.]),
 array([ 96. , 117.4, 138.8, 160.2, 181.6, 203. , 224.4, 245.8, 267.2,
        288.6, 310. ]),
 <BarContainer object of 10 artists>)
```



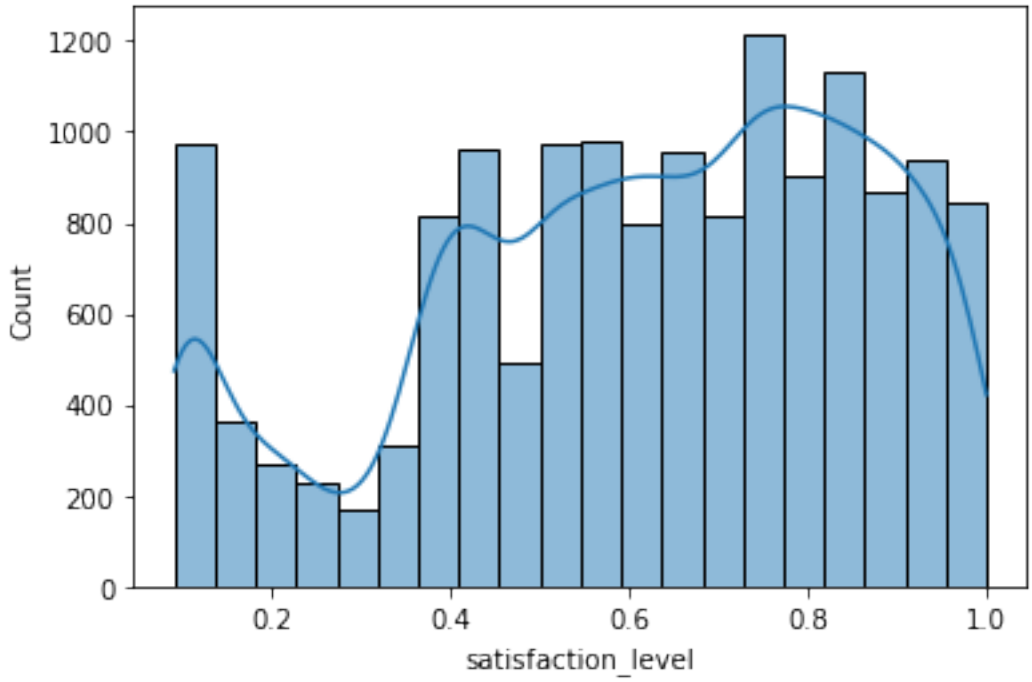
[37]:
```
sns.histplot(data = df,x="average_montly_hours", kde = True,bins=20)
```

[37]:
```
<AxesSubplot:xlabel='average_montly_hours', ylabel='Count'>
```



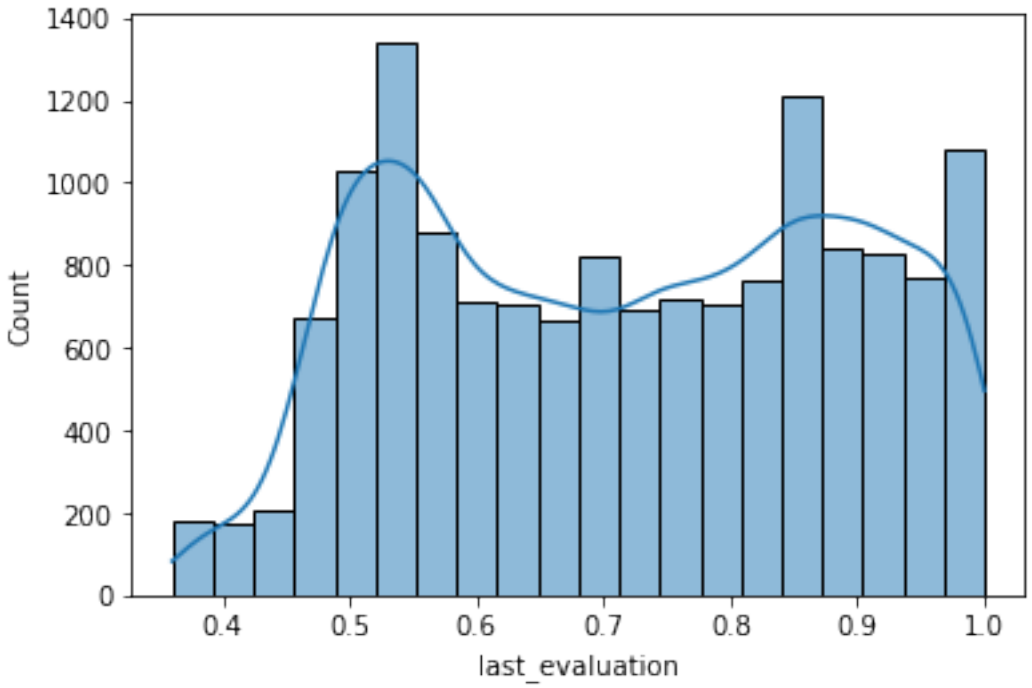
[38]:
```
sns.histplot(data = df,x="satisfaction_level", kde = True,bins=20)
```

[38]: <AxesSubplot:xlabel='satisfaction_level', ylabel='Count'>



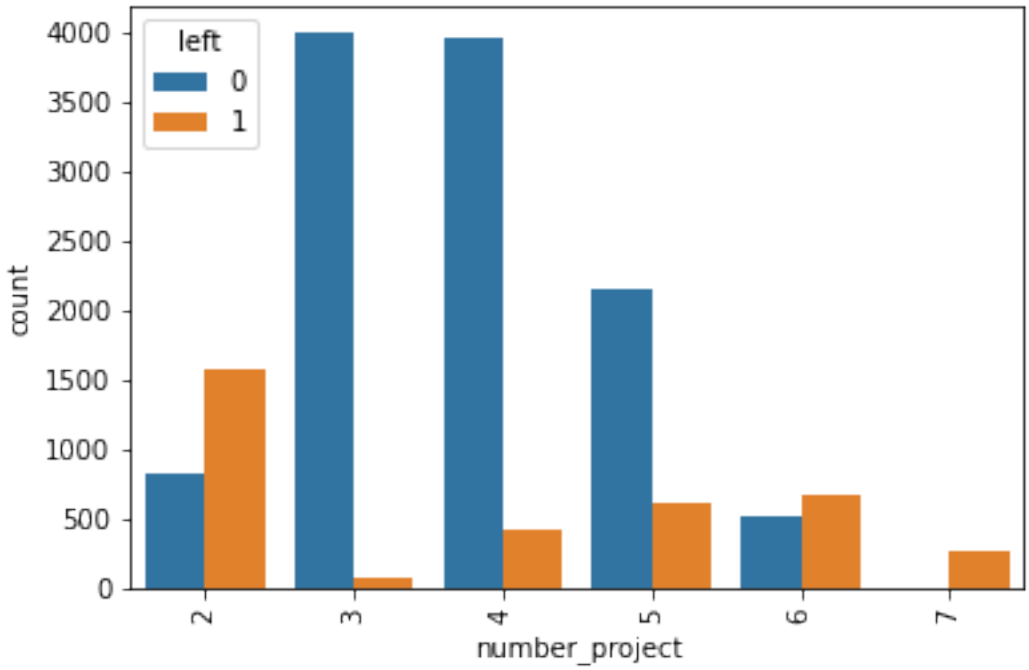
[39]:
```
sns.histplot(data = df,x="last_evaluation", kde = True,bins=20)
```

[39]: <AxesSubplot:xlabel='last_evaluation', ylabel='Count'>



[40]:
```
sns.countplot(x="number_project",hue="left",data=df)
plt.xticks(rotation=90)
```

[40]:
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, '2'),
  Text(1, 0, '3'),
  Text(2, 0, '4'),
  Text(3, 0, '5'),
  Text(4, 0, '6'),
  Text(5, 0, '7')])
```



People who have worked on 3 or 4 projects have left the organisation more.

[41]:
```
dfclus = df[["satisfaction_level","last_evaluation","left"]]
```

[42]:
```
dfclus
```

[42]:
```
       satisfaction_level  last_evaluation  left
0                    0.38             0.53     1
1                    0.80             0.86     1
2                    0.11             0.88     1
3                    0.72             0.87     1
4                    0.37             0.52     1
…                     …                …     …
14994                0.40             0.57     1
14995                0.37             0.48     1
14996                0.37             0.53     1
14997                0.11             0.96     1
14998                0.37             0.52     1

[14999 rows x 3 columns]
```

[43]:
```
from sklearn.cluster import KMeans
```

[44]:
```
km=dfclus.iloc[:,:].values
kmeans = KMeans(n_clusters=3, random_state=0)
```

```
label = kmeans.fit_predict(dfclus)
labelarr = kmeans.fit_predict(km)
```

[45]:
```
label
```

[45]:
```
array([1, 1, 1, …, 1, 1, 1], dtype=int32)
```

[46]:
```
dfclus[label==0].describe()
```

[46]:

	satisfaction_level	last_evaluation	left
count	6720.000000	6720.000000	6720.0
mean	0.813112	0.739728	0.0
std	0.108167	0.154900	0.0
min	0.590000	0.360000	0.0
25%	0.720000	0.610000	0.0
50%	0.810000	0.740000	0.0
75%	0.910000	0.870000	0.0
max	1.000000	1.000000	0.0

[47]:
```
dfclus[label==1].describe()
```

[47]:

	satisfaction_level	last_evaluation	left
count	3571.000000	3571.000000	3571.0
mean	0.440098	0.718113	1.0
std	0.263933	0.197673	0.0
min	0.090000	0.450000	1.0
25%	0.130000	0.520000	1.0
50%	0.410000	0.790000	1.0
75%	0.730000	0.900000	1.0
max	0.920000	1.000000	1.0

[48]:
```
dfclus[label==2].describe()
```

[48]:

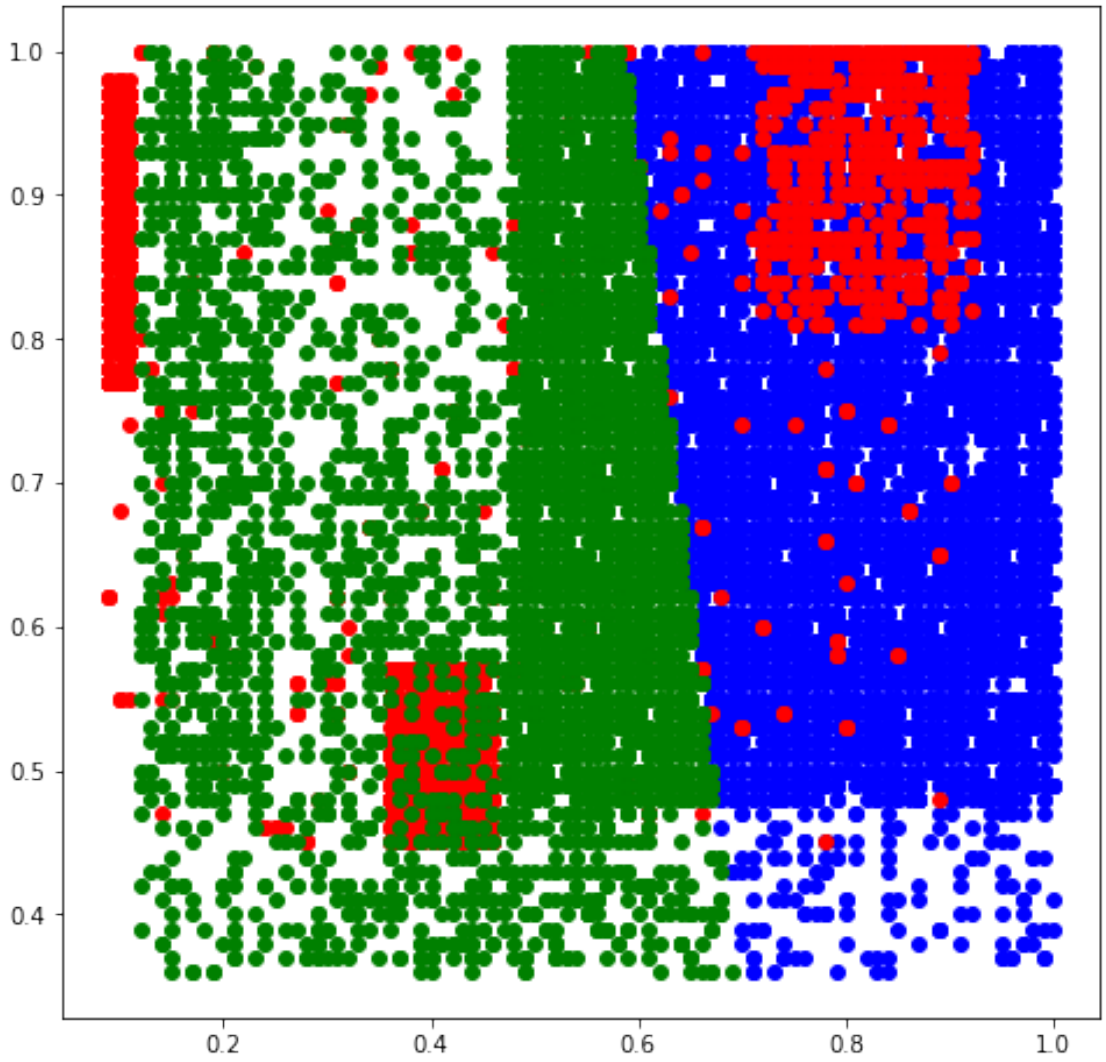
	satisfaction_level	last_evaluation	left
count	4708.000000	4708.000000	4708.0
mean	0.457984	0.680854	0.0
std	0.153456	0.165609	0.0
min	0.120000	0.360000	0.0
25%	0.350000	0.550000	0.0
50%	0.510000	0.670000	0.0
75%	0.570000	0.810000	0.0
max	0.690000	1.000000	0.0

[49]:
```
km[label==0,1]
```

[49]:
```
array([0.67, 0.82, 0.91, …, 0.55, 0.95, 0.54])
```

[50]:
```
plt.figure(figsize=(8,8))
plt.scatter(km[label==0,0],km[label==0,1],color="blue")
plt.scatter(km[label==1,0],km[label==1,1],color="red")
plt.scatter(km[label==2,0],km[label==2,1],color="green")
```

[50]: <matplotlib.collections.PathCollection at 0x7f9d05e80d50>



The Blue cluster denotes people with best satisfaction levels and scored high in the last evaluation.

The Red cluster denotes people with medium satisfaction levels and scored average to high in the last evaluation

The green cluster denotes people with lower satisfaction levels and scored fairly than the above mentioned clusters.

[51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   sales                  14999 non-null  object
 9   salary                 14999 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

[54]:
```
df_numerical=df.select_dtypes(include=['int64','float64'])
df_categorical=df.select_dtypes(include=['object'])
```

Converting the categorical data into numerical using one hot encoding

[57]:
```
#df = pd.get_dummies(data=df,columns=['sales','salary'])
df_converted = pd.get_dummies(data=df_categorical)
```

[58]:
```
df_converted.head()
```

[58]:

	sales_IT	sales_RandD	sales_accounting	sales_hr	sales_management	\
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	

	sales_marketing	sales_product_mng	sales_sales	sales_support	\
0	0	0	1	0	
1	0	0	1	0	
2	0	0	1	0	
3	0	0	1	0	
4	0	0	1	0	

	sales_technical	salary_high	salary_low	salary_medium
0	0	0	1	0
1	0	0	0	1

```
2                0            0            0              1
3                0            0            1              0
4                0            0            1              0
```

[59]:
```
dfn = pd.concat([df_numerical, df_converted], axis=1, join="inner")
```

[60]:
```
dfn.shape
```

[60]:
```
(14999, 21)
```

[61]:
```
dfn.head()
```

[61]:
```
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
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

   time_spend_company  Work_accident  left  promotion_last_5years  sales_IT  \
0                   3              0     1                      0         0
1                   6              0     1                      0         0
2                   4              0     1                      0         0
3                   5              0     1                      0         0
4                   3              0     1                      0         0

   sales_RandD  …  sales_hr  sales_management  sales_marketing  \
0            0  …         0                 0                0
1            0  …         0                 0                0
2            0  …         0                 0                0
3            0  …         0                 0                0
4            0  …         0                 0                0

   sales_product_mng  sales_sales  sales_support  sales_technical  \
0                  0            1              0                0
1                  0            1              0                0
2                  0            1              0                0
3                  0            1              0                0
4                  0            1              0                0

   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
```

```
[5 rows x 21 columns]
```

Splitting the dataset into training and testing in the ratio of 80:20 with random state = 123.

[62]:
```
x =dfn.drop("left",axis=1)
y = dfn["left"]
```

[63]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.2,random_state=123)
```

[64]:
```
xtrain.shape,ytrain.shape,xtest.shape,ytest.shape
```

[64]:
```
((11999, 20), (11999,), (3000, 20), (3000,))
```

[65]:
```
ytrain.value_counts()
```

[65]:
```
0    9137
1    2862
Name: left, dtype: int64
```

Data is highly imbalanced for the training dataset as the record of people who left is very low in comparision to the record of people who didn't leave.

Using SMOTE to handle the imbalance for the left category

[66]:
```
from imblearn.over_sampling import SMOTE
```

[67]:
```
sm = SMOTE(random_state = 2)
xtrainres, ytrainres = sm.fit_resample(xtrain, ytrain)
```

[68]:
```
ytrainres.value_counts()
```

[68]:
```
1    9137
0    9137
Name: left, dtype: int64
```

[69]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
import sklearn.metrics as metrics
```

[70]:
```
logreg = LogisticRegression(solver='lbfgs', max_iter=10000)
```

[71]:
```
print(cross_val_score(logreg, xtrainres, ytrainres, cv=5).mean())
```

```
0.8062837195824601
```

[72]:
```
logreg.fit(xtrainres,ytrainres)
ypred = logreg.predict(xtest)
```

[73]:
```
from sklearn.metrics import classification_report
```

Logistic regression report

[74]:
```
metrics.confusion_matrix(ytest,ypred)
```

[74]:
```
array([[1830,  461],
       [ 228,  481]])
```

[75]:
```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.80      0.84      2291
           1       0.51      0.68      0.58       709

    accuracy                           0.77      3000
   macro avg       0.70      0.74      0.71      3000
weighted avg       0.80      0.77      0.78      3000
```

[76]:
```
roc_auc_score(ytest,ypred)
```

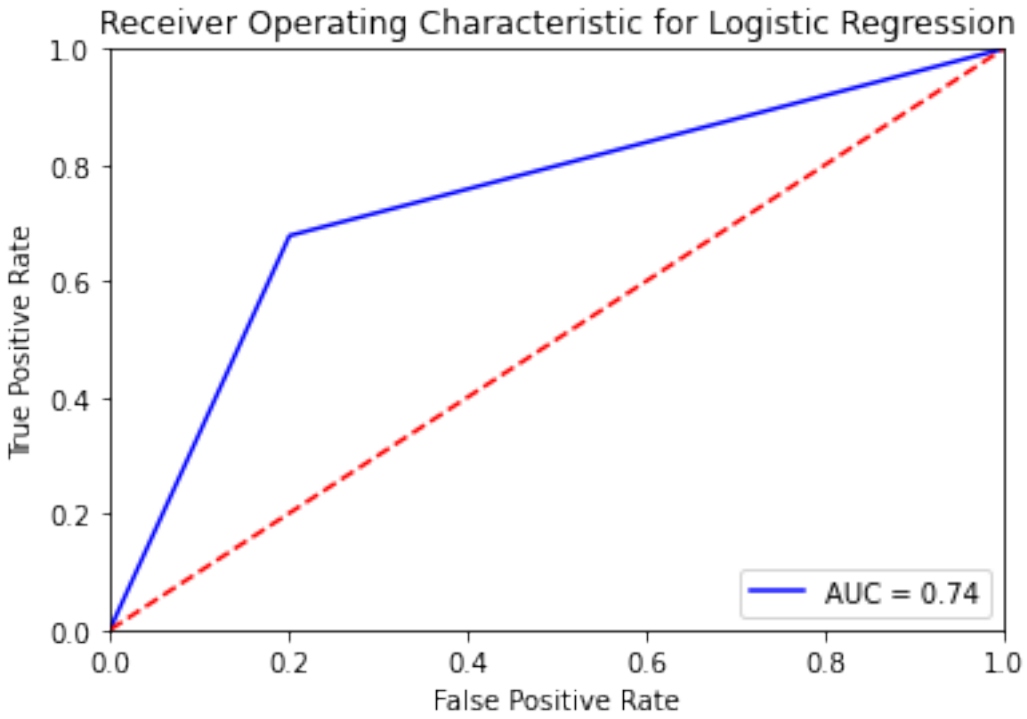
[76]:
```
0.7385990682864635
```

[77]:
```
fpr, tpr, threshold = metrics.roc_curve(ytest, ypred)
print(fpr)
print(tpr)
print(threshold)
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)

# method I: plt
plt.title('Receiver Operating Characteristic for Logistic Regression')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[0.         0.20122217 1.        ]
[0.         0.67842031 1.        ]
```

```
[2 1 0]
0.7385990682864635
```



Random Forest Classifier

[78]:
```
randm=RandomForestClassifier(max_depth=5)
```

[79]:
```
print(cross_val_score(randm, xtrainres, ytrainres, cv=5).mean())
```

```
0.9493269898175789
```

[81]:
```
randm.fit(xtrainres,ytrainres)
ypred1=randm.predict(xtest)
```

Random Forest Classification report

[83]:
```
metrics.confusion_matrix(ytest,ypred1)
```

[83]:
```
array([[2223,   68],
       [  54,  655]])
```

[84]:
```
print(classification_report(ytest,ypred1))
```

```
              precision    recall  f1-score   support
```

```
              0       0.98      0.97      0.97      2291
              1       0.91      0.92      0.91       709

       accuracy                           0.96      3000
      macro avg       0.94      0.95      0.94      3000
   weighted avg       0.96      0.96      0.96      3000
```

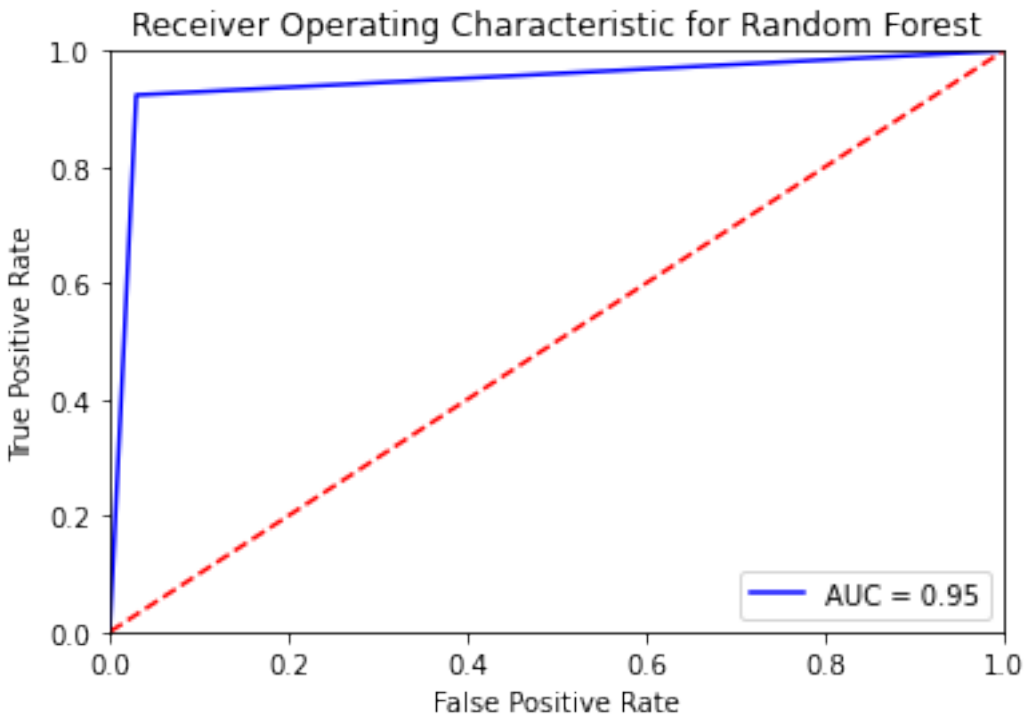
[85]:
```
roc_auc_score(ytest,ypred1)
```

[85]:
```
0.9470775137149785
```

[86]:
```
fpr, tpr, threshold = metrics.roc_curve(ytest, ypred1)
print(fpr)
print(tpr)
print(threshold)
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)

# method I: plt
plt.title('Receiver Operating Characteristic for Random Forest')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[0.         0.02968136 1.        ]
[0.         0.92383639 1.        ]
[2 1 0]
0.9470775137149785
```



Gradient Boosting Classifier

[88]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

[89]:
```
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=1.
↪0,max_depth=1, random_state=0)
```

[90]:
```
print(cross_val_score(gb, xtrainres, ytrainres, cv=5).mean())
```

```
0.9472476764028253
```

[91]:
```
gb.fit(xtrainres,ytrainres)
```

[91]:
```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=0)
```

[92]:
```
ypred2 = gb.predict(xtest)
```

Gradient boosting Classification Report

[93]:
```
metrics.confusion_matrix(ytest,ypred2)
```

[93]:
```
array([[2172,  119],
       [  46,  663]])
```

[94]:
```
print(classification_report(ytest,ypred2))
```

```
              precision    recall  f1-score   support

           0       0.98      0.95      0.96      2291
           1       0.85      0.94      0.89       709

    accuracy                           0.94      3000
   macro avg       0.91      0.94      0.93      3000
weighted avg       0.95      0.94      0.95      3000
```

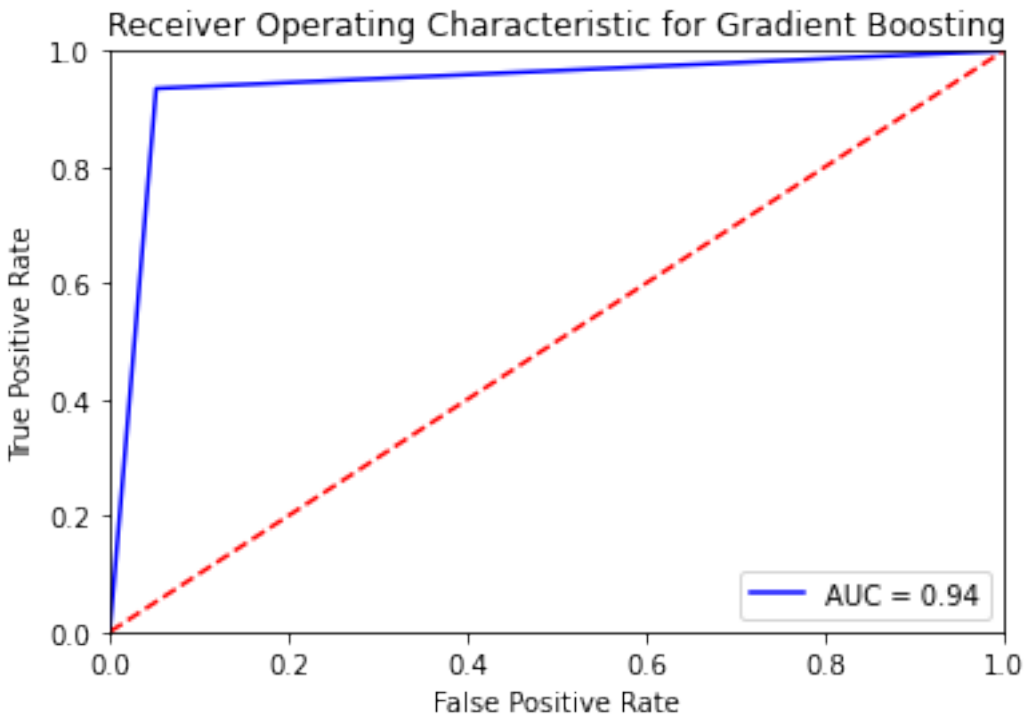
[95]:
```
roc_auc_score(ytest,ypred2)
```

[95]:
```
0.9415887519631305
```

[96]:
```
fpr, tpr, threshold = metrics.roc_curve(ytest, ypred2)
print(fpr)
print(tpr)
print(threshold)
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)

# method I: plt
plt.title('Receiver Operating Characteristic for Gradient Boosting')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[0.         0.05194238 1.        ]
[0.         0.93511989 1.        ]
[2 1 0]
0.9415887519631305
```



Based on the confusion matrix, the false negatives should be low because if an employee who might leave the organisation is misclassified as someone who won't leave then proper strategies to retain that person will not be implemented on him or her. Hence Recall is better metric to be used

[97]:
```
col = xtrainres.columns
```

[98]:
```
col
```

[98]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident',
       'promotion_last_5years', 'sales_IT', 'sales_RandD', 'sales_accounting',
       'sales_hr', 'sales_management', 'sales_marketing', 'sales_product_mng',
       'sales_sales', 'sales_support', 'sales_technical', 'salary_high',
       'salary_low', 'salary_medium'],
      dtype='object')
```

Since Random Forest shows the highest accuracy with good f1 score, we will conclude that to be our best performing model.

[99]:
```
feature_labels = np.array(col)
```

[100]:
```
importance = randm.feature_importances_
feature_indexes_by_importance = importance.argsort()
for index in feature_indexes_by_importance:
```

```
    print('{}-{:.2f}%'.format(feature_labels[index], (importance[index] *100.
↪0)))
```

```
sales_hr-0.01%
sales_marketing-0.01%
sales_accounting-0.02%
sales_support-0.02%
sales_technical-0.03%
sales_sales-0.06%
sales_IT-0.07%
sales_product_mng-0.09%
promotion_last_5years-0.15%
sales_management-0.18%
salary_medium-0.26%
sales_RandD-0.34%
salary_low-0.63%
salary_high-1.38%
Work_accident-2.83%
last_evaluation-11.45%
average_montly_hours-12.14%
number_project-17.96%
time_spend_company-22.01%
satisfaction_level-30.35%
```

[101]:
```
predict_probability = randm.predict_proba(xtest)
```

[102]:
```
predict_probability[:,1]
```

[102]:
```
array([0.04964795, 0.10749424, 0.11349761, …, 0.69553956, 0.06472184,
       0.13966875])
```

[103]:
```
zone=[]
prob=[]

for i in predict_probability[:,1]:
  prob.append(i)
  if (i<=0.2):
    zone.append("Safe Zone")
  elif (i>0.2 and i<=0.6):
    zone.append("Low Risk Zone")
  elif (i>0.6 and i<=0.9):
    zone.append("Medium Risk Zone ")
  else:
    zone.append("High Risk Zone ")
```

[104]:
```
categories = ["Safe Zone","Low Risk Zone","Medium Risk Zone ","High Risk Zone "]
color = ["Green","Yellow","Orange","Red"]
```

```
```

[105]:
```
colordict = dict(zip(categories, color))
```

[106]:
```
clr = pd.DataFrame({"zone":zone,"probability":prob})
```

[107]:
```
clr["zone"].unique()
```

[107]:
```
array(['Safe Zone', 'High Risk Zone ', 'Medium Risk Zone ',
       'Low Risk Zone'], dtype=object)
```

[108]:
```
clr["Color"] = clr["zone"].apply(lambda x: colordict[x])
```

[109]:
```
clr.head(10)
```

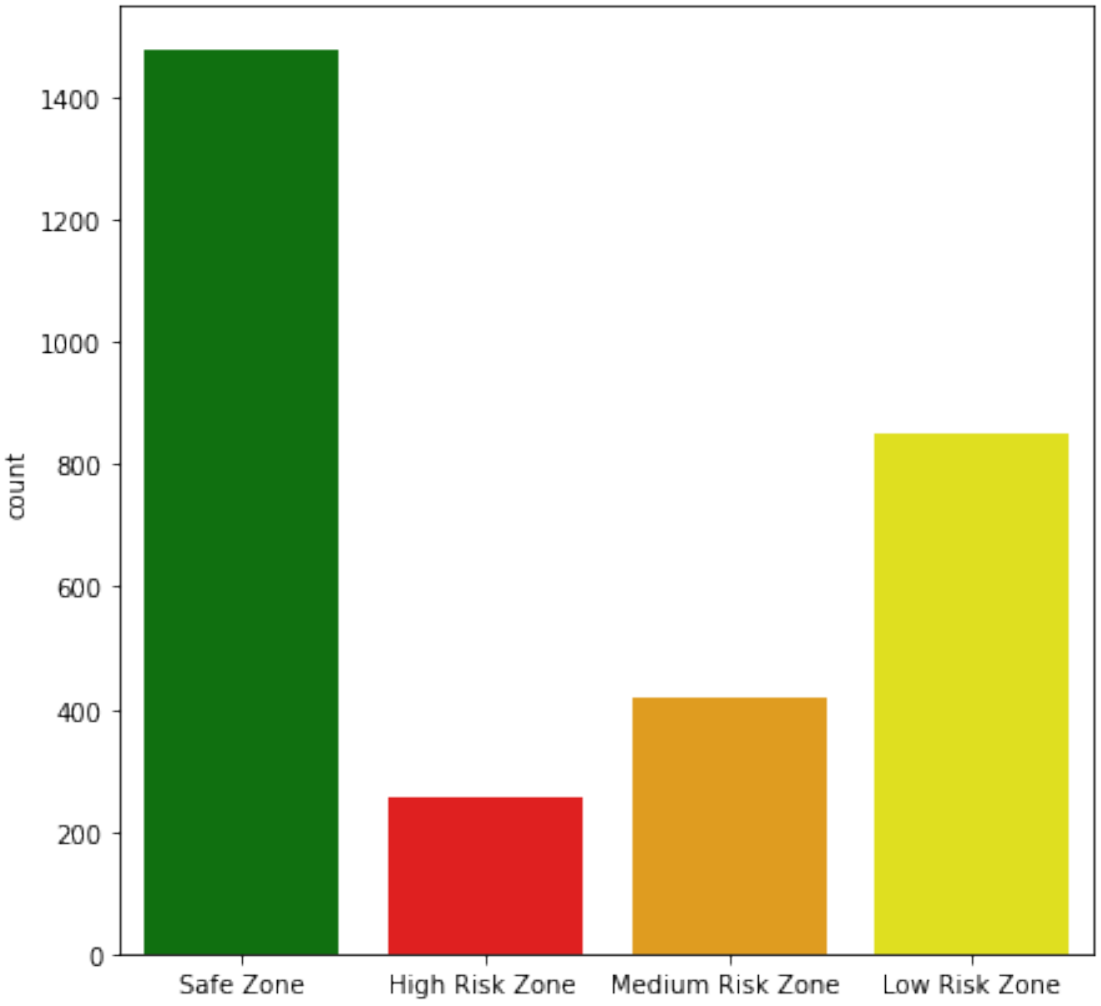
[109]:

	zone	probability	Color
0	Safe Zone	0.049648	Green
1	Safe Zone	0.107494	Green
2	Safe Zone	0.113498	Green
3	Safe Zone	0.106114	Green
4	Safe Zone	0.099141	Green
5	Safe Zone	0.085726	Green
6	High Risk Zone	0.942459	Red
7	Medium Risk Zone	0.746181	Orange
8	Safe Zone	0.146142	Green
9	Safe Zone	0.070108	Green

[111]:
```
color= clr["Color"].tolist()
c = ["Green","Red","Orange","Yellow"]
plt.figure(figsize=(7,7))
sns.countplot(zone,palette=c)
```

```
/usr/local/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

[111]:
```
<AxesSubplot:ylabel='count'>
```



Career Development Opportunities: Provide opportunities for professional growth and development. Offer training programs, mentoring, and coaching to help employees acquire new skills and advance in their careers within the organization. Competitive Compensation: Offer competitive salaries and benefits packages to attract and retain talented employees. Regularly review and adjust compensation based on market trends to ensure employees feel fairly compensated for their work. Recognition and Rewards: Recognize and appreciate employees' contributions and achievements. Implement a formal recognition program that acknowledges outstanding performance and provides rewards such as bonuses, incentives, or public recognition.

[ ]: