GROUP NUMBER:- **11**

GURPREET SINGH:- **0775814**

MANAV SINGH:- **0778064**

To Import the neccessary libraries for data manipulation and visual representation.

In [387…

```
import pandas as pd                # used for manipulation and analysis
import numpy as np                 # to perform mathematical operation
import matplotlib.pyplot as plt    # used to create plots
import matplotlib as matplot
import seaborn as sns              # for data visualization and exploratory
%matplotlib inline
```

Read the analytics csv file

In [388…

```
df = pd.read_csv('/content/Project dataset.csv')
```

CHECKING THE SHAPE

In [389…

```
# The dataset contains 10 attributes and 14999 observations
df.shape
```

Out[389…

```
(14999, 10)
```

PRINTING THE TOP 10 ROWS FROM THE DATASET

In [390…

```
df.head()
```

Out[390…

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work
0	0.38	0.53	2	157	3	
1	0.80	0.86	5	262	6	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	

In [391…

```
## With this code, we can analyze the whole dataset.
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
```

DESCRIBING THE DATASET

In [392…

```
# Display the statistical overview of the employees
df.describe()
```

Out[392…

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	V
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	
mean	0.612834	0.716102	3.803054	201.050337	3.498233	
std	0.248631	0.171169	1.232592	49.943099	1.460136	
min	0.090000	0.360000	2.000000	96.000000	2.000000	
25%	0.440000	0.560000	3.000000	156.000000	3.000000	
50%	0.640000	0.720000	4.000000	200.000000	3.000000	
75%	0.820000	0.870000	5.000000	245.000000	4.000000	
max	1.000000	1.000000	7.000000	310.000000	10.000000	

In [393…

```
NumericAttribute = round(df.describe().T,0)
NumericAttribute
# The round() function returns a floting-point numbers rounded to the specified number
```

Out[393…

	count	mean	std	min	25%	50%	75%	max
satisfaction_level	14999.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0
last_evaluation	14999.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0

	count	mean	std	min	25%	50%	75%	max
number_project	14999.0	4.0	1.0	2.0	3.0	4.0	5.0	7.0
average_montly_hours	14999.0	201.0	50.0	96.0	156.0	200.0	245.0	310.0
time_spend_company	14999.0	3.0	1.0	2.0	3.0	3.0	4.0	10.0
Work_accident	14999.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
left	14999.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
promotion_last_5years	14999.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

In [394…
```
# To save file as csv format
NumericAttribute.T.to_csv('NumericAttribute.csv')
```

CHECKING THE ATTRIBUTES

In [395…
```
# To columns's of the dataset names
df.columns
```

Out[395…
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
       'promotion_last_5years', 'sales', 'salary'],
      dtype='object')
```

RENAMING THE ATTRIBUTES

In [396…
```
# Renaming certain columns for better readability
df = df.rename(columns={'satisfaction_level': 'satisfaction',
                        'last_evaluation': 'evaluation',
                        'number_project': 'projectCount',
                        'average_montly_hours': 'averageMonthlyHours',
                        'time_spend_company': 'yearsAtCompany',
                        'Work_accident': 'workAccident',
                        'promotion_last_5years': 'promotion',
                        'sales' : 'department',
                        'left' : 'turnover'
                        })
```

In [397…
```
df.columns
```

Out[397…
```
Index(['satisfaction', 'evaluation', 'projectCount', 'averageMonthlyHours',
       'yearsAtCompany', 'workAccident', 'turnover', 'promotion', 'department',
       'salary'],
      dtype='object')
```

In [398…
```
# Move the reponse variable "turnover" to the front of the table
front = df['turnover']
df.drop(labels=['turnover'], axis=1,inplace = True)
df.insert(0, 'turnover', front)
```

In [399…
```
df.head()
```

Out[399…

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccide
0	1	0.38	0.53	2	157	3	
1	1	0.80	0.86	5	262	6	
2	1	0.11	0.88	7	272	4	
3	1	0.72	0.87	5	223	5	
4	1	0.37	0.52	2	159	3	

COUNTING NULL VALUES IN THE DATASET

In [400…
```
df.isnull()
```

Out[400…

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workA
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
...	...	...	...	...	...	...	
14994	False	False	False	False	False	False	
14995	False	False	False	False	False	False	
14996	False	False	False	False	False	False	
14997	False	False	False	False	False	False	
14998	False	False	False	False	False	False	

14999 rows × 10 columns

In [401…
```
df.isnull().sum()
```

Out[401…
```
turnover              0
satisfaction          0
evaluation            0
projectCount          0
averageMonthlyHours   0
yearsAtCompany        0
workAccident          0
promotion             0
department            0
```

```
salary                0
dtype: int64
```

In [402…
```
df.isnull().sum().sum()
```

Out[402…
```
0
```

In [403…
```
#check duplicated
df.duplicated()
```

Out[403…
```
0        False
1        False
2        False
3        False
4        False
         ...
14994     True
14995     True
14996     True
14997     True
14998     True
Length: 14999, dtype: bool
```

In [404…
```
df.duplicated().sum()
```

Out[404…
```
3008
```

In [405…
```
#df.drop_duplicates()
```

In [406…
```
# To check the datatypes of all the variables
df.dtypes
```

Out[406…
```
turnover                int64
satisfaction          float64
evaluation            float64
projectCount            int64
averageMonthlyHours     int64
yearsAtCompany          int64
workAccident            int64
promotion               int64
department             object
salary                 object
dtype: object
```

No of employee conitnue/Turnover the organisation

In [407…
```
# Looks like about 76% of employees stayed and 24% of employees left.
turnover_rate = df.turnover.value_counts() / 14999
turnover_rate
```

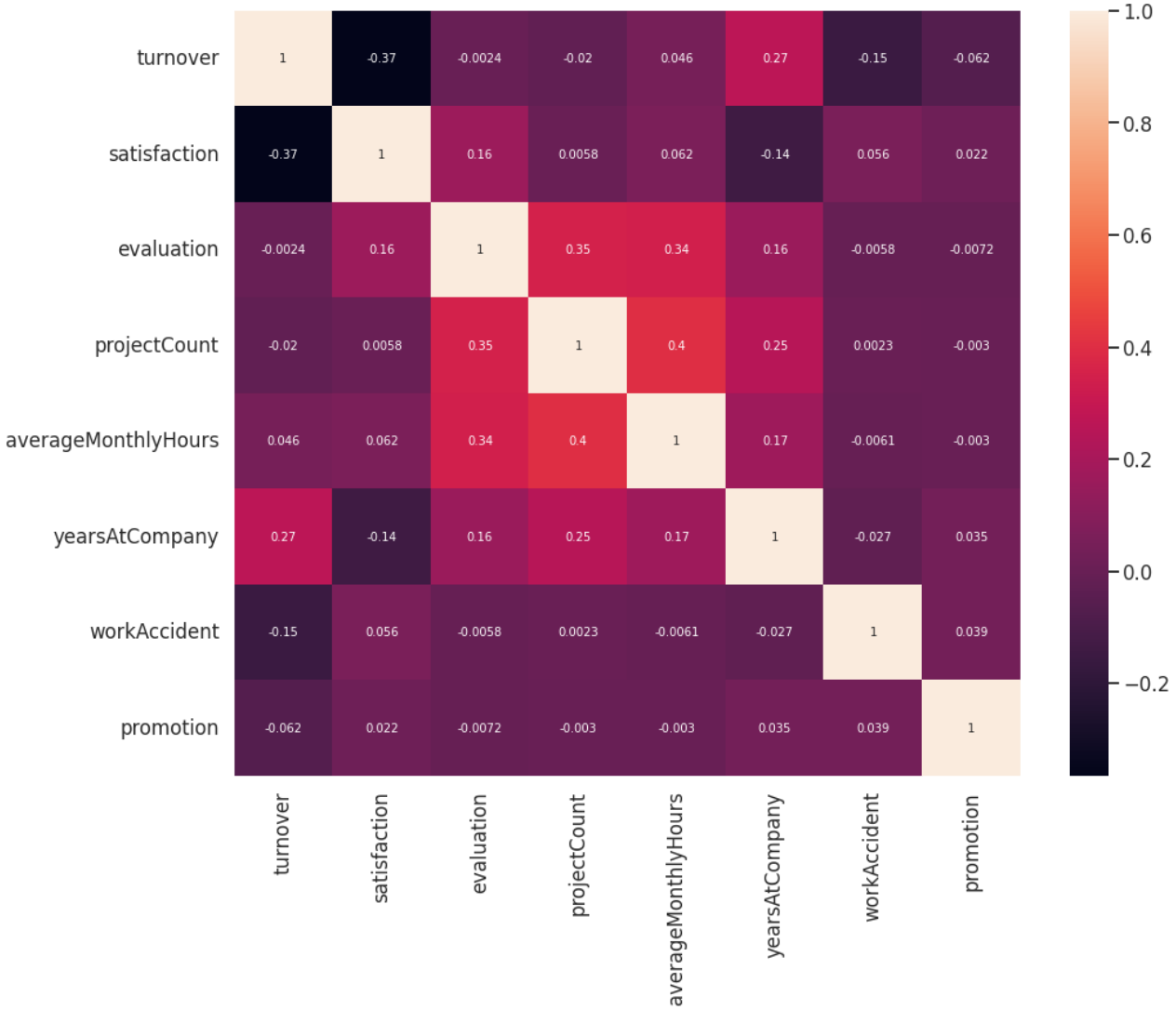
Out[407…
```
0    0.761917
1    0.238083
Name: turnover, dtype: float64
```

In [408…
```
# Overview of summary (Turnover V.S. Non-turnover)
turnover_Summary = df.groupby('turnover')
turnover_Summary.mean()
```

Out[408…

turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccident
0	0.666810	0.715473	3.786664	199.060203	3.380032	0.175009
1	0.440098	0.718113	3.855503	207.419210	3.876505	0.047326

In [409…
```
#Correlation Matrix
plt.figure(figsize = (15,12))
sns.heatmap(df.corr(method = 'spearman'), annot = True)
plt.savefig("Correlation.jpg")
```



From the heatmap, there is a positive(+) correlation between projectCount,

averageMonthlyHours, and evaluation. Which could mean that the employees who spent more hours and did more projects were evaluated highly.

For the negative(-) relationships, turnover and satisfaction are highly correlated. we assume that people tend to leave a company more when they are less satisfied.

Salary V.S. Turnover

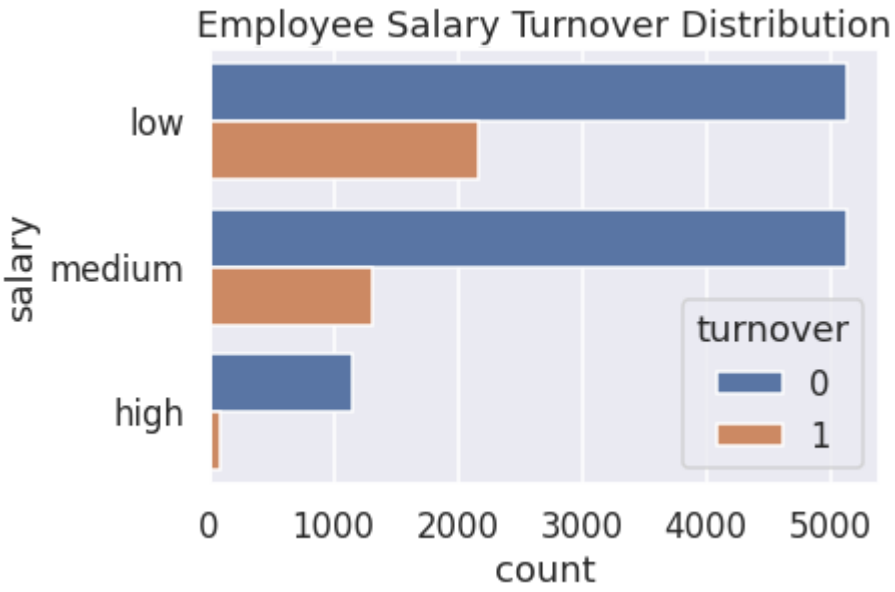
In [410…
```
# To check unique values
df.salary.unique()
```

Out[410…
```
array(['low', 'medium', 'high'], dtype=object)
```

In [411…
```
df.department.unique()
```

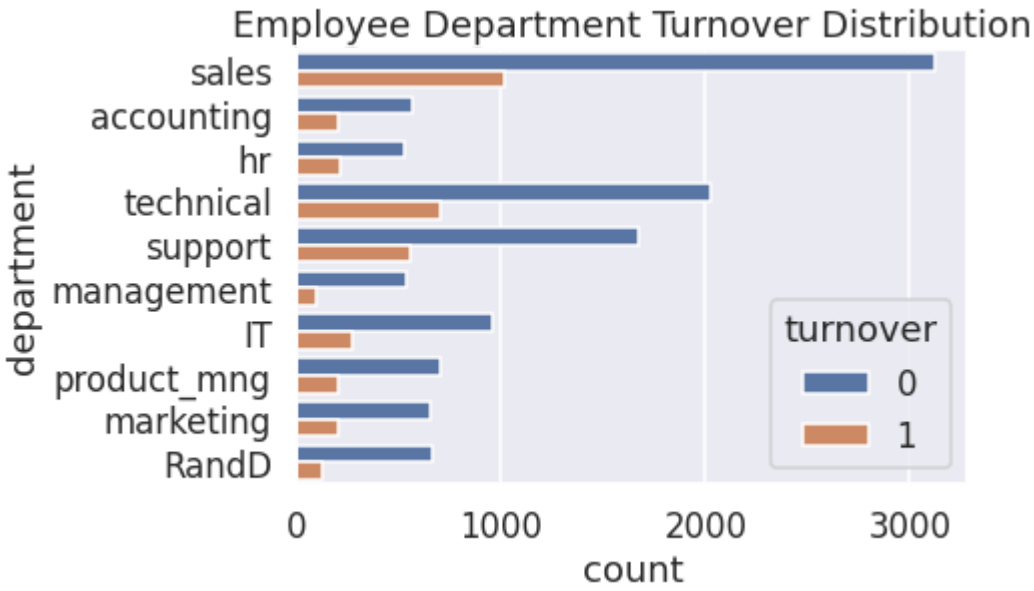
Out[411…
```
array(['sales', 'accounting', 'hr', 'technical', 'support', 'management',
       'IT', 'product_mng', 'marketing', 'RandD'], dtype=object)
```

In [412…
```
sns.countplot(y="salary", hue='turnover', data=df).set_title('Employee Salary Turnover
plt.savefig('Employee Salary Turnover Distribution.jpg');
```



Department V.S. Turnover

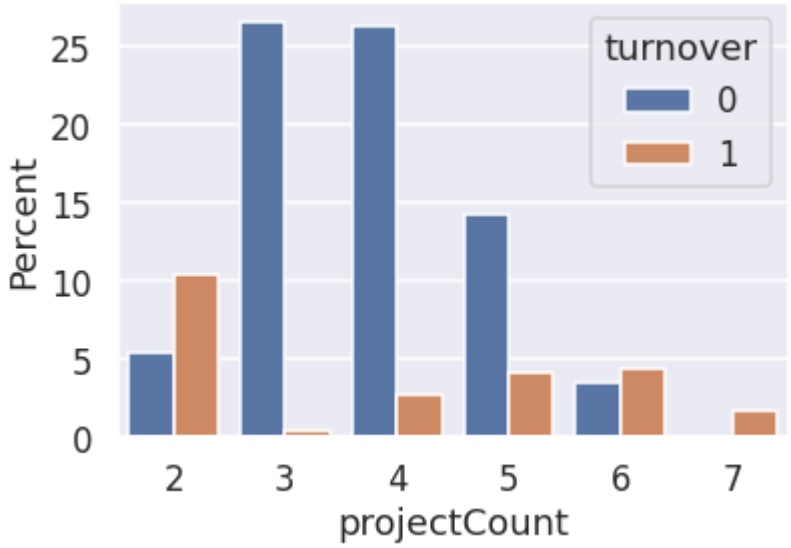
In [413…
```
sns.countplot(y="department", hue='turnover', data=df).set_title('Employee Department T
plt.savefig('Employee Department Turnover Distribution.jpg');
```



Turnover V.S. ProjectCount

In [414…

```
ax = sns.barplot(x="projectCount", y="projectCount", hue="turnover", data=df, estimator
ax.set(ylabel="Percent")
plt.savefig('Turnover vs PeojectCount.jpg');
```



Number of emplyees left the orginasation

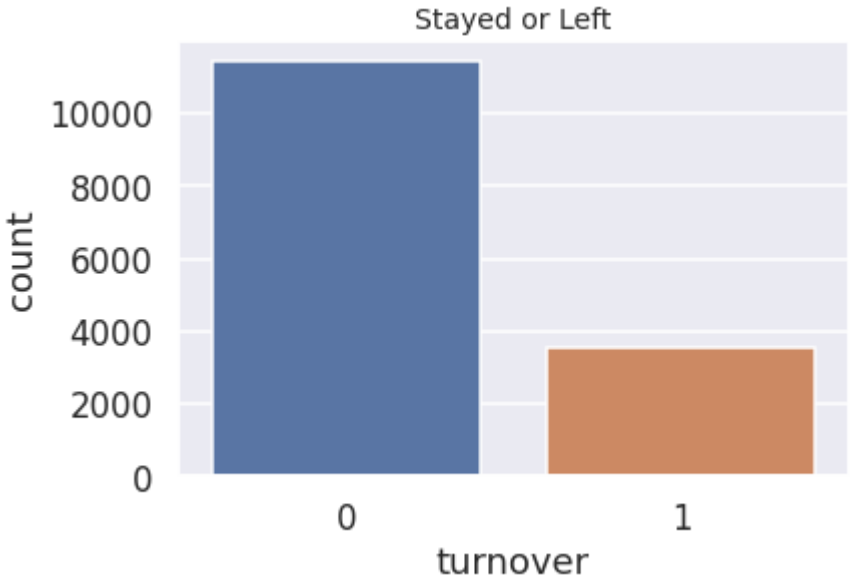
In [415…

```
print(df['turnover'].value_counts()[1],"employees left the company")
```

```
3571 employees left the company
```

In [416…

```
# The plot show the amount of employees that stayed and left the company.
ax = sns.countplot(df.turnover)
plt.title('Stayed or Left', fontsize=14);
plt.savefig("Stayed or left the organation.jpg")
```

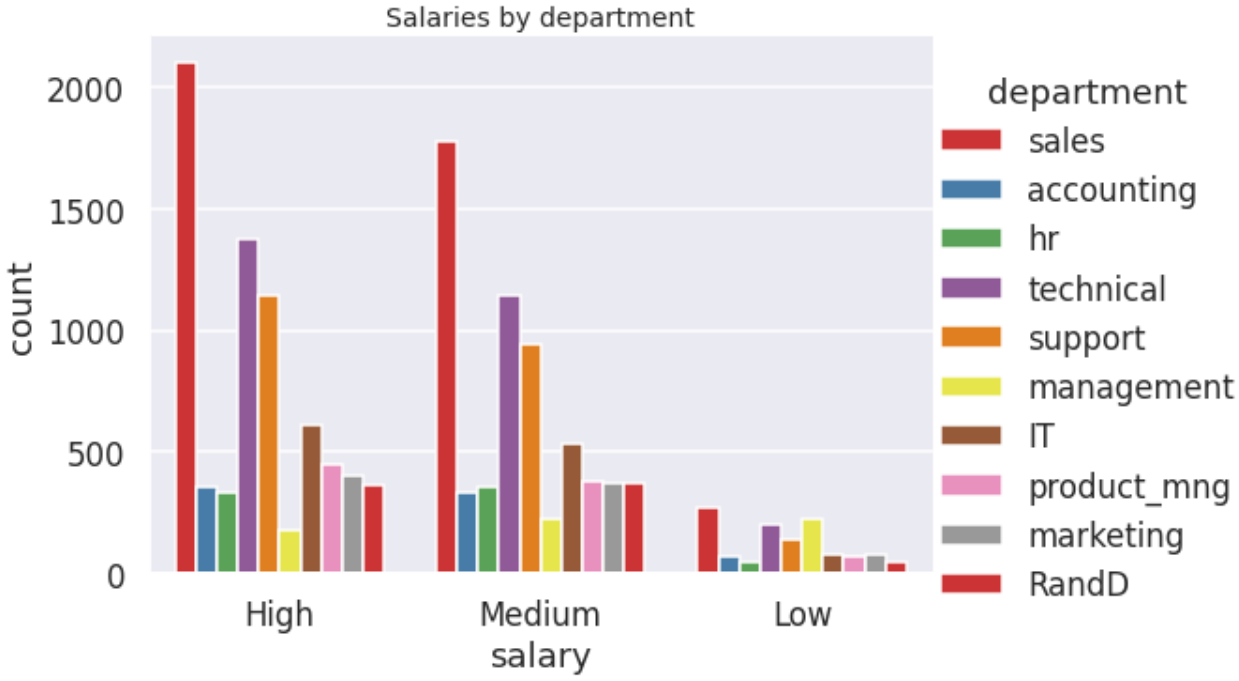


In [417…

```
j = sns.factorplot(x='salary', y='turnover', kind='bar', data=df)
plt.title('Employees that left by salary level', fontsize=14);
j.set_xticklabels(['High', 'Medium', 'Low'])
plt.savefig("Employee that left by salary level.jpg")
```

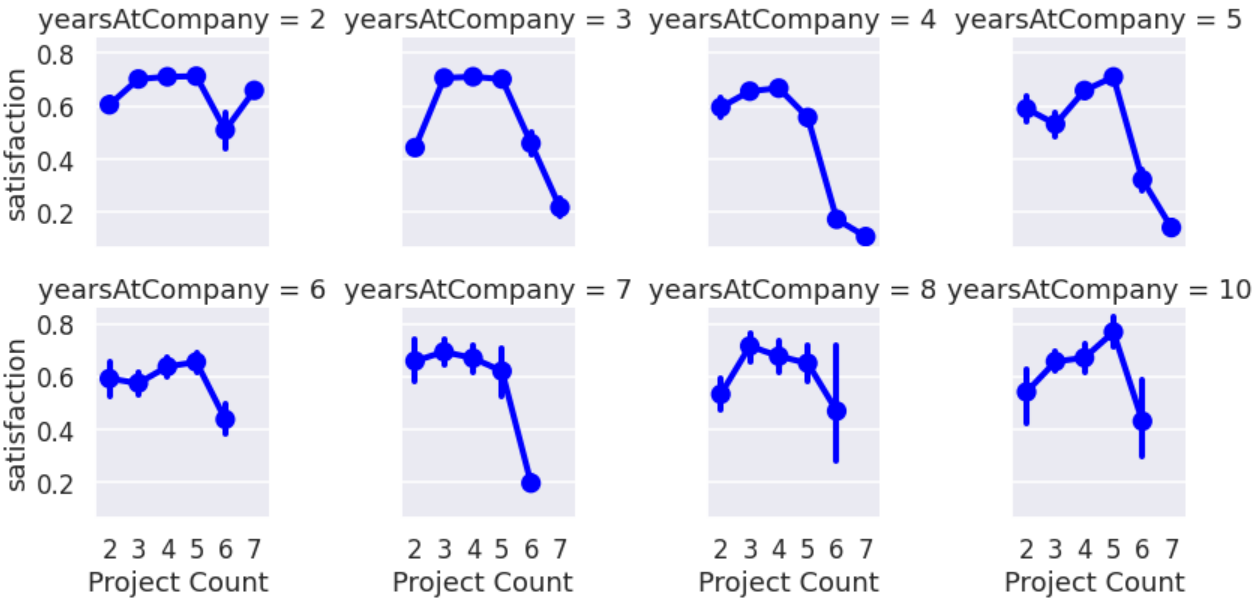
In [418…

```
h = sns.factorplot(x = 'salary', hue='department', kind ='count', size = 5,aspect=1.5,
plt.title("Salaries by department", fontsize=14)
h.set_xticklabels(['High', 'Medium', 'Low']);
plt.savefig("Salaries by department.jpg")
```



In [419…

```
sns.set()
sns.set_context("talk")
ax = sns.factorplot(x="projectCount", y="satisfaction", col="yearsAtCompany", col_wrap=
ax.set_xlabels('Project Count ');
plt.savefig("satisfaction according to the project done in the number of year by the em
```



In [322…

```
df["department"] = df["department"].astype("category")
df["salary"] = df["salary"].astype("category")
```

In [323…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
```

```
---  ------                --------------  -----
 0   turnover              14999 non-null  int64
 1   satisfaction          14999 non-null  float64
 2   evaluation            14999 non-null  float64
 3   projectCount          14999 non-null  int64
 4   averageMonthlyHours   14999 non-null  int64
 5   yearsAtCompany        14999 non-null  int64
 6   workAccident          14999 non-null  int64
 7   promotion             14999 non-null  int64
 8   department            14999 non-null  category
 9   salary                14999 non-null  category
dtypes: category(2), float64(2), int64(6)
memory usage: 967.4 KB
```

In [324…

```
#Check whether the provided array or dtype a category dtype

numeric = []
category = []
for col in df:
  if pd.api.types.is_numeric_dtype(df[col]):
    numeric.append(col)
  else:
      category.append(col)
print("category:",category)
```

```
category: ['department', 'salary']
```

In [325…

```
# Scalling
from sklearn.preprocessing import MinMaxScaler
integer = ['turnover', 'projectCount', 'averageMonthlyHours', 'yearsAtCompany', 'workAc

scaler = MinMaxScaler()
df[integer] = scaler.fit_transform(df[integer])
```

In [326…

```
df.head()
```

Out[326…

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccide
0	1.0	0.38	0.53	0.0	0.285047	0.125	0
1	1.0	0.80	0.86	0.6	0.775701	0.500	0
2	1.0	0.11	0.88	1.0	0.822430	0.250	0
3	1.0	0.72	0.87	0.6	0.593458	0.375	0
4	1.0	0.37	0.52	0.0	0.294393	0.125	0

In [327…

```
# Create dummy variables
df = pd.get_dummies(df) #converts integer data into dummy or indicator variables
```

In [328…

```
df.head()
```

Out[328…

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccide
0	1.0	0.38	0.53	0.0	0.285047	0.125	0
1	1.0	0.80	0.86	0.6	0.775701	0.500	0
2	1.0	0.11	0.88	1.0	0.822430	0.250	0
3	1.0	0.72	0.87	0.6	0.593458	0.375	0
4	1.0	0.37	0.52	0.0	0.294393	0.125	0

5 rows × 21 columns

In [329…

```
df.shape
```

Out[329…

```
(14999, 21)
```

In [330…

```
# import some classification models
from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

# import needed functions

from sklearn.model_selection import train_test_split

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

In [331…

```
#Separating the independent variables and dependent variables
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [332…

```
x.head()
```

Out[332…

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccide
0	1.0	0.38	0.53	0.0	0.285047	0.125	0
1	1.0	0.80	0.86	0.6	0.775701	0.500	0
2	1.0	0.11	0.88	1.0	0.822430	0.250	0
3	1.0	0.72	0.87	0.6	0.593458	0.375	0
4	1.0	0.37	0.52	0.0	0.294393	0.125	0

In [333…

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size = 0.30)
```

In [334…
```
x_test.head()
```

Out[334…

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workA
5214	0.0	0.57	0.53	0.6	0.560748	0.000	
3095	0.0	0.91	0.98	0.2	0.761682	0.250	
555	1.0	0.10	0.90	1.0	0.864486	0.250	
12197	1.0	0.10	0.91	0.8	0.892523	0.250	
14612	1.0	0.76	0.92	0.4	0.700935	0.375	

In [335…
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size= 0.30,random_state=3)
```

In [336…
```
# summerize class distribution
print("Before undersampling")
y_train.value_counts()
```

Out[336…
```
Before undersampling
0    5986
1    4513
Name: salary_medium, dtype: int64
```

In [337…
```
from imblearn.under_sampling import RandomUnderSampler
```

In [338…
```
from imblearn import under_sampling
under_sampling = RandomUnderSampler(sampling_strategy='majority', random_state=3)
```

In [339…
```
# Fit and apply the transform
x_train_under,y_train_under = under_sampling.fit_resample(x_train,y_train)
x_train_under.head(5)
```

Out[339…

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccide
0	1.0	0.10	0.91	0.8	0.892523	0.250	0
1	0.0	0.56	0.98	0.2	0.579439	0.125	0
2	1.0	0.11	0.84	0.8	0.724299	0.250	0
3	0.0	0.45	0.66	0.2	0.070093	0.250	0
4	0.0	0.25	0.75	0.6	0.457944	0.375	1

In [340…
```
print("after undersampling")
y_train_under.value_counts()
```

```
after undersampling
0    4513
1    4513
Name: salary_medium, dtype: int64
```

Out[340…

CROSS VALIDATION

In [340…

In [341…

```
#Importing train test split

from sklearn.model_selection import train_test_split
```

In [342…

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train,y_train)
predict_logistic = model.predict(x_test)
confusion_matrix_logistic = confusion_matrix(y_test,predict_logistic)
logistic_acc = accuracy_score(y_test,predict_logistic)*100
print("accuracy of logistic Regression:",logistic_acc)
```

```
accuracy of logistic Regression: 100.0
```

In [343…

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
pred_k=knn.predict(x_test)
knn_acc=accuracy_score(y_test,pred_k)*100
print("accuracy oh KNN:",knn_acc)
```

```
accuracy oh KNN: 99.91111111111111
```

In [344…

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb = model.fit(x_train,y_train)
pred_gnb=gnb.predict(x_test)
gnb_acc = accuracy_score(y_test,pred_gnb)*100
print("accuracy oh Gaussian Navie Baise:",gnb_acc)
```

```
accuracy oh Gaussian Navie Baise: 100.0
```

Comparing the accuracy

In [345…

```
labels = ["KNN", "Logistic Regression", "Naive Bayes"]
x = [logistic_acc, knn_acc, gnb_acc]
eval_frame = pd.DataFrame()
eval_frame['Model'] = labels
eval_frame['train_test_split'] = x
eval_frame
```

Out[345…

	Model	train_test_split

	Model	train_test_split
0	KNN	100.000000
1	Logistic Regression	99.911111
2	Naive Bayes	100.000000

Optimal value

In [346…
```
error_rate=[]
for i in range(1,20):
  knn=KNeighborsClassifier(n_neighbors=i)
  knn.fit(x_train,y_train)
  pred=knn.predict(x_test)
  error_rate.append(np.mean(pred!=y_test))
```

In [347…
```
#error rate = (-1(correct prediction/total prediction))*100
```

In [420…
```
plt.figure(figsize=(10,6))
plt.plot(range(1,20),error_rate,color='black',linestyle='dashed',marker='o',markerfacec
plt.title('Error rate vs K value')
plt.xlabel('K')
plt.ylabel("Error rate")
## disply the visuliztaion of the confusion matrix.
# plt.savefig("error rate corrosponding knn.png",bbox_inches='tight)
plt.savefig("Error Rate vs K value.jpg")
```

In [349…
```
churn_features = ["satisfaction","evaluation","projectCount","averageMonthlyHours","yea
```

In [350…
```
x = df[churn_features]
x.describe()
```

Out[350…

	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000
mean	0.612834	0.716102	0.360611	0.490889	0.187279
std	0.248631	0.171169	0.246518	0.233379	0.182517
min	0.090000	0.360000	0.000000	0.000000	0.000000
25%	0.440000	0.560000	0.200000	0.280374	0.125000
50%	0.640000	0.720000	0.400000	0.485981	0.125000
75%	0.820000	0.870000	0.600000	0.696262	0.250000
max	1.000000	1.000000	1.000000	1.000000	1.000000

In [351…
```
x.head()
```

Out[351…

	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany
0	0.38	0.53	0.0	0.285047	0.125
1	0.80	0.86	0.6	0.775701	0.500
2	0.11	0.88	1.0	0.822430	0.250
3	0.72	0.87	0.6	0.593458	0.375
4	0.37	0.52	0.0	0.294393	0.125

In [352…
```
from sklearn.model_selection import train_test_split
train_input,test_input,train_output,test_output = train_test_split(x,y,test_size=0.3,ra
```

In [353…
```
train_input
```

Out[353…

	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany
1354	0.10	0.92	1.0	0.827103	0.250
4026	0.91	0.93	0.4	0.411215	0.125
14452	0.37	0.46	0.0	0.280374	0.125
12170	0.81	0.99	0.4	0.761682	0.375
12342	0.90	1.00	0.6	0.635514	0.375
…	…	…	…	…	…
11798	0.61	0.42	0.2	0.228972	0.250
13896	0.41	0.38	0.4	0.214953	1.000

	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany
6637	0.88	0.89	0.2	0.350467	0.000
2575	0.89	0.83	0.6	0.799065	0.250
7336	0.52	0.75	0.4	0.794393	0.125

10499 rows × 5 columns

In [354…
```
x.shape[0]
```

Out[354…
```
14999
```

In [355…
```
train_input.shape
```

Out[355…
```
(10499, 5)
```

In [356…
```
test_input.shape
```

Out[356…
```
(4500, 5)
```

In [357…
```
test_output.shape
```

Out[357…
```
(4500,)
```

In [358…
```
from sklearn.linear_model import LogisticRegression
```

In [359…
```
model =LogisticRegression()
```

In [360…
```
model1 = model.fit(train_input,train_output)
```

In [361…
```
test_response_predict = model1.predict(test_input)
```

In [362…
```
test_response_predict
```

Out[362…
```
array([0, 0, 0, ..., 0, 0, 0], dtype=uint8)
```

In [363…
```
test_output
```

Out[363…
```
7231     0
12081    0
5364     0
14589    1
2353     0
```

```
        ..
10244    1
9121     0
3241     0
14414    0
7781     0
Name: salary_medium, Length: 4500, dtype: uint8
```

In [364…
```
df.columns
```

Out[364…
```
Index(['turnover', 'satisfaction', 'evaluation', 'projectCount',
       'averageMonthlyHours', 'yearsAtCompany', 'workAccident', 'promotion',
       'department_IT', 'department_RandD', 'department_accounting',
       'department_hr', 'department_management', 'department_marketing',
       'department_product_mng', 'department_sales', 'department_support',
       'department_technical', 'salary_high', 'salary_low', 'salary_medium'],
      dtype='object')
```

In [365…
```
churn_features = ['satisfaction', 'evaluation', 'projectCount',
       'averageMonthlyHours', 'yearsAtCompany']
x1 =  df[churn_features]
y1 = df["turnover"]
train_input,test_input,train_output,test_output = train_test_split(x,y,test_size=0.3,ra
model2 = model.fit(train_input,train_output)
```

In [366…
```
churn_features = ["satisfaction","evaluation","projectCount","averageMonthlyHours","yea
x =df[churn_features]
x.head()
```

Out[366…

	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany
0	0.38	0.53	0.0	0.285047	0.125
1	0.80	0.86	0.6	0.775701	0.500
2	0.11	0.88	1.0	0.822430	0.250
3	0.72	0.87	0.6	0.593458	0.375
4	0.37	0.52	0.0	0.294393	0.125

In [367…
```
from sklearn.preprocessing import MinMaxScaler
```

In [368…
```
from pandas.core.tools.datetimes import Scalar
Scalar = MinMaxScaler()
x = pd.DataFrame(Scalar.fit_transform(x),columns = x.columns)
x
```

Out[368…

	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany
0	0.318681	0.265625	0.0	0.285047	0.125
1	0.780220	0.781250	0.6	0.775701	0.500
2	0.021978	0.812500	1.0	0.822430	0.250

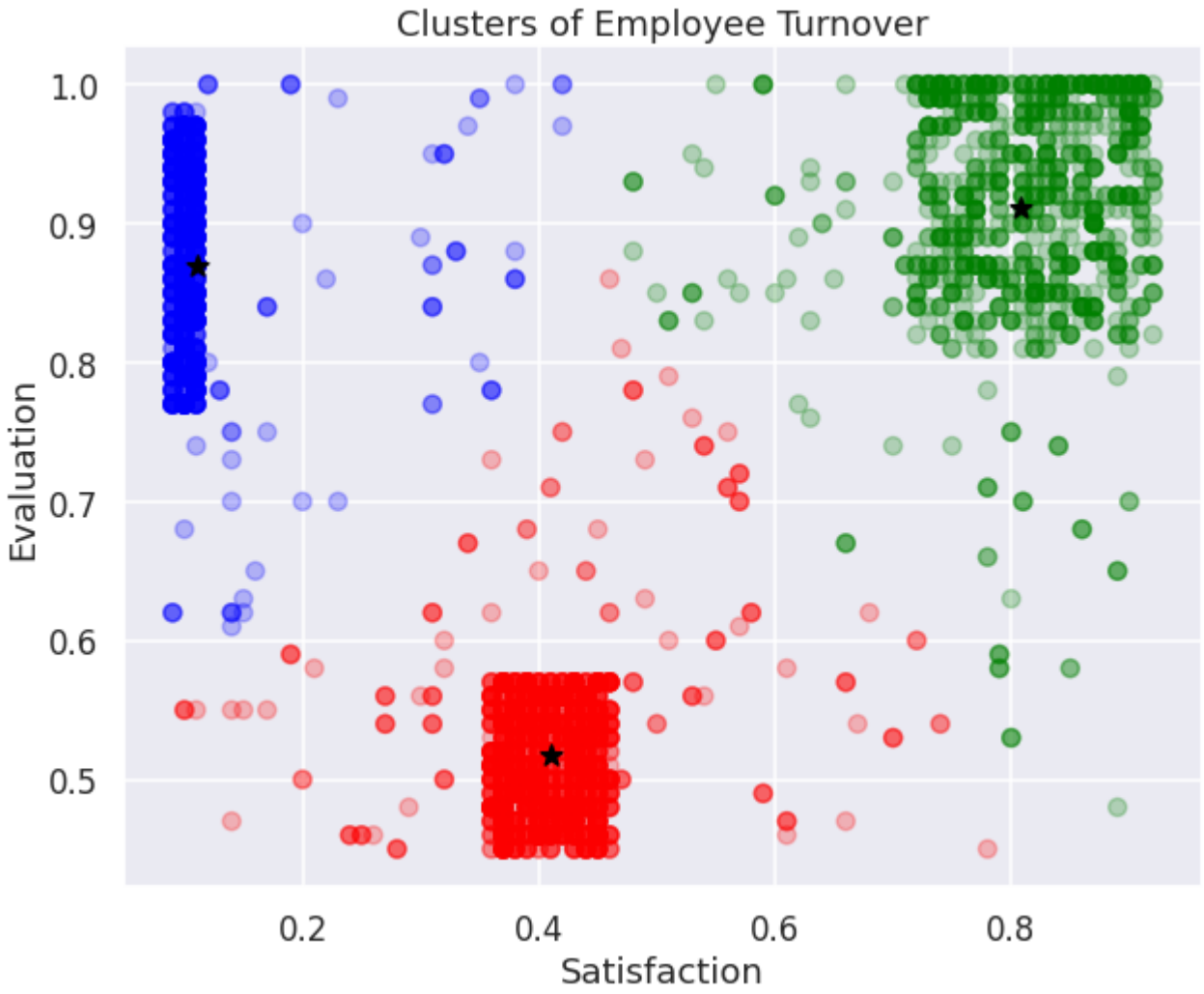
	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany
3	0.692308	0.796875	0.6	0.593458	0.375
4	0.307692	0.250000	0.0	0.294393	0.125
...	...	...	...	...	...
14994	0.340659	0.328125	0.0	0.257009	0.125
14995	0.307692	0.187500	0.0	0.299065	0.125
14996	0.307692	0.265625	0.0	0.219626	0.125
14997	0.021978	0.937500	0.8	0.859813	0.250
14998	0.307692	0.250000	0.0	0.289720	0.125

14999 rows × 5 columns

K-Means Clustering of Employee Turnover

In [421…

```
# Import KMeans Model
from sklearn.cluster import KMeans

# Graph and create 3 clusters of Employee Turnover
kmeans = KMeans(n_clusters=3,random_state=2)
kmeans.fit(df[df.turnover==1][["satisfaction","evaluation"]])

kmeans_colors = ['green' if c == 0 else 'blue' if c == 2 else 'red' for c in kmeans.lab

fig = plt.figure(figsize=(10, 8))
plt.scatter(x="satisfaction",y="evaluation", data=df[df.turnover==1],
            alpha=0.25,color = kmeans_colors)
plt.xlabel("Satisfaction")
plt.ylabel("Evaluation")
plt.scatter(x=kmeans.cluster_centers_[:,0],y=kmeans.cluster_centers_[:,1],color="black"
plt.title("Clusters of Employee Turnover")
plt.savefig("Cluster of Employee Turnover.jpg")
## (BLUE COLOR) THESE ARE THE PEOPLE WHO ARE HARDWORKER BUT SAD EMPLOYEE
## (RED COLOR) THESE ARE THE PEOPLE WHO ARE NOT HARDWORKER AND SAD EMPLOYEE
## (GREEN) THESE ARE THE PEOPLE WHO ARE HARDWORKER AND HAPPY EMPLOYEE
```



In [370…

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, precision_score, rec
from sklearn.preprocessing import RobustScaler
```

A Base Rate Model is a model that always selects the target variable's majority class. It's just used for reference to compare how better another model is against it. In this dataset, the majority class that will be predicted will be 0's, which are employees who did not leave the company. The Base Rate Model would simply predict every 0's and ignore all the 1's. 0's means who did not leave the company.

In [371…

```
# Create base rate model
def base_rate_model(X) :
    y = np.zeros(X.shape[0])
    return y
```

In [372…

```
# Create train and test splits
target_name = 'turnover'
X = df.drop('turnover', axis=1)
robust_scaler = RobustScaler()
X = robust_scaler.fit_transform(X)
```

```
y=df[target_name]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20, random_state=12
```

In [373…

```
# Check accuracy of base rate model
y_base_rate = base_rate_model(X_test)
from sklearn.metrics import accuracy_score
print ("Base rate accuracy is %2.2f" % accuracy_score(y_test, y_base_rate))
```

```
Base rate accuracy is 0.76
```

The %2.2f directive tells Python to format the number as at least two characters and to cut the precision to two characters after the decimal point. This is useful for printing floating-point numbers

In [374…

```
# Check accuracy of Logistic Model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(penalty='l2', C=1)

model.fit(X_train, y_train)
print ("Logistic accuracy is %2.2f" % accuracy_score(y_test, model.predict(X_test)))
```

```
Logistic accuracy is 0.79
```

In [375…

```
# Using 10 fold Cross-Validation to train our Logistic Regression Model
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
kfold = model_selection.KFold(n_splits=10)
modelCV = LogisticRegression(class_weight = "balanced")
scoring = 'roc_auc'
results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=
print("AUC: %.2f (%.2f)" % (results.mean(), results.std()))
```

```
AUC: 0.83 (0.01)
```

Class Imbalance

This dataset is an example of a class imbalance problem because of the skewed distribution of employees who did and did not leave. More skewed the class means that accuracy breaks down.

In this case, evaluating our model's algorithm based on accuracy is the wrong thing to measure. We would have to know the different errors that we care about and correct decisions. Accuracy alone does not measure an important concept that needs to be taken into consideration in this type of evaluation: False Positive and False Negative errors.

False Positives (Type I Error): we predict that the employee will leave, but do not

False Negatives (Type II Error): we predict that the employee will not leave, but does leave

Logistic Regression V.S. Random Forest V.S. Decision Tree

In [376…

```
# Compare the Logistic Regression Model V.S. Base Rate Model V.S. Random Forest Model
from sklearn.metrics import roc_auc_score
```

```
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import VotingClassifier
```

In [377…

```
# NOTE: By adding in "class_weight = balanced", the Logistic Auc increased by about 10%
logis = LogisticRegression(class_weight = "balanced")
logis.fit(X_train, y_train)
print ("Logistic Model")
logit_roc_auc = roc_auc_score(y_test, logis.predict(X_test))
print ("Logistic AUC = %2.2f" % logit_roc_auc)
print(classification_report(y_test, logis.predict(X_test)))
```

```
Logistic Model
Logistic AUC = 0.78
              precision    recall  f1-score   support

         0.0       0.93      0.75      0.83      2286
         1.0       0.51      0.81      0.62       714

    accuracy                           0.77      3000
   macro avg       0.72      0.78      0.73      3000
weighted avg       0.83      0.77      0.78      3000
```

AUC or AUROC is area under ROC curve. The value of AUC characterizes the model performance. Higher the AUC value, higher the performance of the model. The perfect classifier will have high value of true positive rate and low value of false positive rate.

A Receiver Operator Characteristic (ROC) curve is a graphical plot used to show the diagnostic ability of binary classifiers. It was first used in signal detection theory but is now used in many other areas such as medicine, radiology, natural hazards and machine learning.

In [378…

```
# Decision Tree Model
dtree = tree.DecisionTreeClassifier(
    #max_depth=3,
    class_weight="balanced",
    min_weight_fraction_leaf=0.01
    )
dtree = dtree.fit(X_train,y_train)
print ("Decision Tree Model")
dt_roc_auc = roc_auc_score(y_test, dtree.predict(X_test))
print ("Decision Tree AUC = %2.2f" % dt_roc_auc)
print(classification_report(y_test, dtree.predict(X_test)))
```

```
Decision Tree Model
Decision Tree AUC = 0.95
              precision    recall  f1-score   support
```

```
              0.0       0.97      0.98      0.98      2286
              1.0       0.92      0.92      0.92       714

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

In [379…

```
# Random Forest Model
rf = RandomForestClassifier(
    n_estimators=1000,
    max_depth=None,
    min_samples_split=10,
    class_weight="balanced"
    #min_weight_fraction_leaf=0.02
    )
rf.fit(X_train, y_train)
print ("Random Forest Model")
rf_roc_auc = roc_auc_score(y_test, rf.predict(X_test))
print ("Random Forest AUC = %2.2f" % rf_roc_auc)
print(classification_report(y_test, rf.predict(X_test)))
```

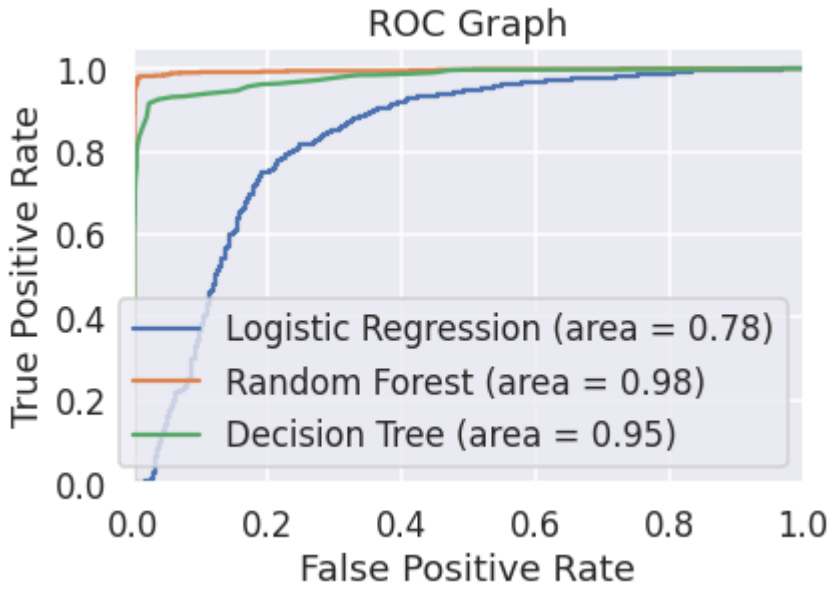
```
Random Forest Model
Random Forest AUC = 0.98
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      2286
         1.0       0.99      0.96      0.97       714

    accuracy                           0.99      3000
   macro avg       0.99      0.98      0.98      3000
weighted avg       0.99      0.99      0.99      3000
```

ROC Graph

In [422…

```
# Create ROC Graph
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, logis.predict_proba(X_test)[:,1])
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, rf.predict_proba(X_test)[:,1])
dt_fpr, dt_tpr, dt_thresholds = roc_curve(y_test, dtree.predict_proba(X_test)[:,1])

plt.figure()

# Plot Logistic Regression ROC
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)

# Plot Random Forest ROC
plt.plot(rf_fpr, rf_tpr, label='Random Forest (area = %0.2f)' % rf_roc_auc)

# Plot Decision Tree ROC
plt.plot(dt_fpr, dt_tpr, label='Decision Tree (area = %0.2f)' % dt_roc_auc)


plt.xlim([0.0,1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Graph')
```

```
plt.legend(loc="lower right")
plt.savefig("ROC GRAPH.jpg")
```



More the positive rate and less the neative rate in ROC graph means higher performance model for future test regarding this dataset.

CLASSIFICATION

In [381…

```
labels =["Logistic Model","Random Forest Model","Decision Tree Model"]
X = [logit_roc_auc,rf_roc_auc,dt_roc_auc]
eval_frame=pd.DataFrame()
eval_frame['Models']=labels
eval_frame['Train_test_split'] = X
eval_frame
```

Out[381…

	Models	Train_test_split
0	Logistic Model	0.780965
1	Random Forest Model	0.975623
2	Decision Tree Model	0.946172