```
# importing libraries for data handling and analysis
import pandas as pd
from pandas.plotting import scatter_matrix
from pandas import ExcelWriter
from pandas import ExcelFile
from openpyxl import load_workbook
import numpy as np
from scipy.stats import norm, skew
from scipy import stats
import statsmodels.api as sm

# importing libraries for data visualisations
import seaborn as sns
from matplotlib import pyplot
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import matplotlib
%matplotlib inline
color = sns.color_palette()
from IPython.display import display
pd.options.display.max_columns = None
# Standard plotly imports
import plotly
import chart_studio.plotly as py
import plotly.graph_objs as go
import plotly.figure_factory as ff

from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import cufflinks as cf
cf.set_config_file(offline=True)
import cufflinks
cufflinks.go_offline(connected=True)
init_notebook_mode(connected=True)

# sklearn modules for preprocessing
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
# from imblearn.over_sampling import SMOTE  # SMOTE
# sklearn modules for ML model selection
from sklearn.model_selection import train_test_split  # import 'train_test_split'
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# Libraries for data modelling
from sklearn import svm, tree, linear_model, neighbors
from sklearn import naive_bayes, ensemble, discriminant_analysis, gaussian_process
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier

# Common sklearn Model Helpers
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics
# from sklearn.datasets import make_classification

# sklearn modules for performance metrics
from sklearn.metrics import confusion_matrix, classification_report, precision_recall_curv
from sklearn.metrics import auc, roc_auc_score, roc_curve, recall_score, log_loss
from sklearn.metrics import f1_score, accuracy_score, roc_auc_score, make_scorer
from sklearn.metrics import average_precision_score

# importing misceallenous libraries
import os
import re
import sys
import timeit
import string
from datetime import datetime
from time import time
from dateutil.parser import parse


#Enabling Interactive plots on google colab
def enable_plotly_in_cell():
  import IPython
  from plotly.offline import init_notebook_mode
  display(IPython.core.display.HTML('''<script src="/static/components/requirejs/require.j
  init_notebook_mode(connected=False)
```

```
! pip install chart_studio
```

```
df = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition.csv')
print(df.shape)
```

Data Description and Visualization

```
df_hr = df.copy()
df_hr.columns
```

```
df_hr.head()
```

```
df_hr.info()
```

Data contains no missing values.

```
df_hr.describe()
```

```
df_hr.hist(figsize=(20,20))
plt.show()
```

.

Feature distribution by target attribute

```
#Age

(mu, sigma) = norm.fit(df_hr.loc[df_hr['Attrition'] == 'Yes', 'Age'])
print('Ex-exmployees: average age = {:.1f} years old and standard deviation = {:.1f}'.form

(mu, sigma) = norm.fit(df_hr.loc[df_hr['Attrition'] == 'No', 'Age'])
print('Current exmployees: average age = {:.1f} years old and standard deviation = {:.1f}'
```

```
#Creating a Kernal Density Estimation plot

x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'Age']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'Age']

hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels, curve_type='kde', show_hist=False, show_
fig['layout'].update(title='Age Distribution in Percent by Attrition Status')
fig['layout'].update(xaxis=dict(range=[15, 60], dtick=5))


fig.show(renderer = "colab")
#py.iplot(fig, filename='Distplot with Multiple Datasets')
```

```
#Educational Fields
df_hr['EducationField'].value_counts()
```

```
#Normalized percentage of Leavers for each Field.

df_BusinessTravel = pd.DataFrame(columns=["Business Travel", "% of Leavers"])
i=0
for field in list(df_hr['BusinessTravel'].unique()):
    ratio = df_hr[(df_hr['BusinessTravel']==field)&(df_hr['Attrition']=="Yes")].shape[0] /
    df_BusinessTravel.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_BT = df_BusinessTravel.groupby(by="Business Travel").sum()
df_BT.iplot(kind='bar',title='Leavers by Business Travel (%)')
```

```
#Gender
df_hr['Gender'].value_counts()
```

```
print("Normalised gender distribution of ex-employees in the dataset: Male = {:.1f}%; Fema
        .format((df_hr[(df_hr['Attrition'] == 'Yes') & (
        df_hr['Gender'] == 'Male')].shape[0] / df_hr[df_hr['Gender'] == 'Male'].shape[0])*
        (df_hr[(df_hr['Attrition'] == 'Yes') & (df_hr['Gender'] == 'Female')].shape[0] / d
```

```
df_Gender = pd.DataFrame(columns=["Gender", "% of Leavers"])
i=0
for field in list(df_hr['Gender'].unique()):
    ratio = df_hr[(df_hr['Gender']==field)&(df_hr['Attrition']=="Yes")].shape[0] / df_hr[d
    df_Gender.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_G = df_Gender.groupby(by="Gender").sum()
df_G.iplot(kind='bar',title='Leavers by Gender (%)')
```

```
#Marital Status
df_hr['MaritalStatus'].value_counts()
```

```
df_Marital = pd.DataFrame(columns=["Marital Status", "% of Leavers"])
i=0
for field in list(df_hr['MaritalStatus'].unique()):
    ratio = df_hr[(df_hr['MaritalStatus']==field)&(df_hr['Attrition']=="Yes")].shape[0] /
    df_Marital.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))
```

```
enable_plotly_in_cell()
df_MF = df_Marital.groupby(by="Marital Status").sum()
df_MF.iplot(kind='bar',title='Leavers by Marital Status (%)')
```

```
#Distance from Home

print("Distance from home for employees to get to work is from {} to {} miles."
        .format(df_hr['DistanceFromHome'].min(),
                df_hr['DistanceFromHome'].max()))
```

```
print('Average distance from home for currently active employees: {:.2f} miles and ex-empl
    df_hr[df_hr['Attrition'] == 'No']['DistanceFromHome'].mean(), df_hr[df_hr['Attrition']
```

```
x1 = df_hr.loc[df_hr['Attrition']=='No','DistanceFromHome']
x2 = df_hr.loc[df_hr['Attrition']=='Yes','DistanceFromHome']

hist_data = [x1, x2]
group_labels = ['Active- Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)
fig['layout'].update( title='Distance From Home Distribution in Percent by Attrition Statu
fig['layout'].update(xaxis=dict(range=[0, 30], dtick=2))

fig.show(renderer = "colab")
```

```
#Department
df_hr['Department'].value_counts()
```

```
df_Department = pd.DataFrame(columns=["Department", "% of Leavers"])
i=0
for field in list(df_hr['Department'].unique()):
    ratio = df_hr[(df_hr['Department']==field)&(df_hr['Attrition']=="Yes")].shape[0] / df_
    df_Department.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_DF = df_Department.groupby(by="Department").sum()
df_DF.iplot(kind='bar',title='Leavers by Department (%)')
```

```
#Role and Work conditions (Travel commitment varies)
df_hr['BusinessTravel'].value_counts()
```

```
df_BusinessTravel = pd.DataFrame(columns=["Business Travel", "% of Leavers"])
i=0
for field in list(df_hr['BusinessTravel'].unique()):
    ratio = df_hr[(df_hr['BusinessTravel']==field)&(df_hr['Attrition']=="Yes")].shape[0] /
    df_BusinessTravel.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_BT = df_BusinessTravel.groupby(by="Business Travel").sum()
df_BT.iplot(kind='bar',title='Leavers by Business Travel (%)')
```

```
df_hr['JobRole'].value_counts()
```

```
df_JobRole = pd.DataFrame(columns=["Job Role", "% of Leavers"])
i=0
for field in list(df_hr['JobRole'].unique()):
    ratio = df_hr[(df_hr['JobRole']==field)&(df_hr['Attrition']=="Yes")].shape[0] / df_hr[
    df_JobRole.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_JR = df_JobRole.groupby(by="Job Role").sum()
df_JR.iplot(kind='bar',title='Leavers by Job Role (%)')
```

```
df_hr['JobLevel'].value_counts()
```

```
df_JobLevel = pd.DataFrame(columns=["Job Level", "% of Leavers"])
i=0
for field in list(df_hr['JobLevel'].unique()):
    ratio = df_hr[(df_hr['JobLevel']==field)&(df_hr['Attrition']=="Yes")].shape[0] / df_hr
    df_JobLevel.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_JL = df_JobLevel.groupby(by="Job Level").sum()
df_JL.iplot(kind='bar',title='Leavers by Job Level (%)')
```

```
df_hr['JobInvolvement'].value_counts()
# Ranges from 1 = Low to 4 = Very High
```

```
df_JobInvolvement = pd.DataFrame(columns=["Job Involvement", "% of Leavers"])
i=0
for field in list(df_hr['JobInvolvement'].unique()):
    ratio = df_hr[(df_hr['JobInvolvement']==field)&(df_hr['Attrition']=="Yes")].shape[0] /
    df_JobInvolvement.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_JI = df_JobInvolvement.groupby(by="Job Involvement").sum()
df_JI.iplot(kind='bar',title='Leavers by Job Involvement (%)')
```

```
print("Number of training times last year varies from {} to {} years.".format(
    df_hr['TrainingTimesLastYear'].min(), df_hr['TrainingTimesLastYear'].max()))
```

```
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'TrainingTimesLastYear']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'TrainingTimesLastYear']


hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']
fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)
fig['layout'].update(
    title='Training Times Last Year metric in Percent by Attrition Status')
fig['layout'].update(xaxis=dict(range=[0, 6], dtick=1))

fig.show(renderer='colab')
```

```
df_hr['NumCompaniesWorked'].value_counts()
```

```
df_hr['NumCompaniesWorked'].value_counts()
```

```
df_NumCompaniesWorked = pd.DataFrame(columns=["Num Companies Worked", "% of Leavers"])
i=0
for field in list(df_hr['NumCompaniesWorked'].unique()):
    ratio = df_hr[(df_hr['NumCompaniesWorked']==field)&(df_hr['Attrition']=="Yes")].shape[
    df_NumCompaniesWorked.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_NC = df_NumCompaniesWorked.groupby(by="Num Companies Worked").sum()
df_NC.iplot(kind='bar',title='Leavers by Num Companies Worked (%)')
```

```
#Years at Company
df_hr['YearsAtCompany'].value_counts()
```

```
print('Average Number of Years at the company for currently active employees: {:.2f} miles
        .format( df_hr[df_hr['Attrition'] == 'No']['YearsAtCompany'].mean(), df_hr[df_hr['
```

```
print("Number of Years at the company varies from {} to {} years.".format(
    df_hr['YearsAtCompany'].min(), df_hr['YearsAtCompany'].max()))
```

```
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'YearsAtCompany']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'YearsAtCompany']

hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)
fig['layout'].update(title='Years At Company in Percent by Attrition Status')
fig['layout'].update(xaxis=dict(range=[0, 40], dtick=5))

fig.show(renderer='colab')
```

```
print("Number of Years in the current role varies from {} to {} years.".format(
    df_hr['YearsInCurrentRole'].min(), df_hr['YearsInCurrentRole'].max()))
```

```
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'YearsInCurrentRole']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'YearsInCurrentRole']

hist_data = [x1, x2]
```

```
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)

fig['layout'].update(title='Years In Current Role in Percent by Attrition Status')
fig['layout'].update(xaxis=dict(range=[0, 18], dtick=1))

fig.show(renderer='colab')
```

```
print("Number of Years since last promotion varies from {} to {} years.".format(
    df_hr['YearsSinceLastPromotion'].min(), df_hr['YearsSinceLastPromotion'].max()))
```

```
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'YearsSinceLastPromotion']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'YearsSinceLastPromotion']

hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)
```

```
fig['layout'].update(title='Years Since Last Promotion in Percent by Attrition Status')
fig['layout'].update(xaxis=dict(range=[0, 15], dtick=1))

fig.show(renderer='colab')
```

```
print("Total working years varies from {} to {} years.".format(
    df_hr['TotalWorkingYears'].min(), df_hr['TotalWorkingYears'].max()))
```

```
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'TotalWorkingYears']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'TotalWorkingYears']

hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)

fig['layout'].update(title='Total Working Years in Percent by Attrition Status')
fig['layout'].update(xaxis=dict(range=[0, 40], dtick=5))

fig.show(renderer='colab')
```

```
#Years with current Manager

print('Average Number of Years with current manager for currently active employees: {:.2f}
    df_hr[df_hr['Attrition'] == 'No']['YearsWithCurrManager'].mean(), df_hr[df_hr['Attriti
```

```
print("Number of Years with current manager varies from {} to {} years.".format(
    df_hr['YearsWithCurrManager'].min(), df_hr['YearsWithCurrManager'].max()))
```

```
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'YearsWithCurrManager']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'YearsWithCurrManager']

hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)

fig['layout'].update(
    title='Years With Current Manager in Percent by Attrition Status')
fig['layout'].update(xaxis=dict(range=[0, 17], dtick=1))
```

```
fig['layout'].update(xaxis=dict(range=[0, 1], dtick=1))

fig.show(renderer='colab')
```

```
#Work-Life Balance
df_hr['WorkLifeBalance'].value_counts()
```

```
df_WorkLifeBalance = pd.DataFrame(columns=["WorkLifeBalance", "% of Leavers"])
i=0
for field in list(df_hr['WorkLifeBalance'].unique()):
    ratio = df_hr[(df_hr['WorkLifeBalance']==field)&(df_hr['Attrition']=="Yes")].shape[0]
    df_WorkLifeBalance.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_WLB = df_WorkLifeBalance.groupby(by="WorkLifeBalance").sum()
df_WLB.iplot(kind='bar',title='Leavers by WorkLifeBalance (%)')
```

```
df_hr['StandardHours'].value_counts()
```

```
df_hr['OverTime'].value_counts()
```

```
df_OverTime = pd.DataFrame(columns=["OverTime", "% of Leavers"])
i=0
for field in list(df_hr['OverTime'].unique()):
    ratio = df_hr[(df_hr['OverTime']==field)&(df_hr['Attrition']=="Yes")].shape[0] / df_hr
    df_OverTime.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_OT = df_OverTime.groupby(by="OverTime").sum()
```

```
df_OT.iplot(kind='bar',title='Leavers by OverTime (%)')
```

```
#Employee wage information

print("Employee Hourly Rate varies from ${} to ${}.".format(
    df_hr['HourlyRate'].min(), df_hr['HourlyRate'].max()))
```

```
print("Employee Daily Rate varies from ${} to ${}.".format(
    df_hr['DailyRate'].min(), df_hr['DailyRate'].max()))
```

```
print("Employee Monthly Rate varies from ${} to ${}.".format(
    df_hr['MonthlyRate'].min(), df_hr['MonthlyRate'].max()))
```

```
print("Employee Monthly Income varies from ${} to ${}.".format(
    df_hr['MonthlyIncome'].min(), df_hr['MonthlyIncome'].max()))
```

```python
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'MonthlyIncome']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'MonthlyIncome']

hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)

fig['layout'].update(title='Monthly Income by Attrition Status')
fig['layout'].update(xaxis=dict(range=[0, 20000], dtick=2000))

fig.show(renderer='colab')
```

```python
print("Percentage Salary Hikes varies from {}% to {}%.".format(
    df_hr['PercentSalaryHike'].min(), df_hr['PercentSalaryHike'].max()))
```

```python
x1 = df_hr.loc[df_hr['Attrition'] == 'No', 'PercentSalaryHike']
x2 = df_hr.loc[df_hr['Attrition'] == 'Yes', 'PercentSalaryHike']

hist_data = [x1, x2]
```

```
hist_data = [x1, x2]
group_labels = ['Active Employees', 'Ex-Employees']

fig = ff.create_distplot(hist_data, group_labels,
                         curve_type='kde', show_hist=False, show_rug=False)

fig['layout'].update(title='Percent Salary Hike by Attrition Status')
fig['layout'].update(xaxis=dict(range=[10, 26], dtick=1))

fig.show(renderer='colab')
```

```
print("Stock Option Levels varies from {} to {}.".format(
    df_hr['StockOptionLevel'].min(), df_hr['StockOptionLevel'].max()))
```

```
print("Normalised percentage of leavers by Stock Option Level: 1: {:.2f}%, 2: {:.2f}%, 3:
    df_hr[(df_hr['Attrition'] == 'Yes') & (df_hr['StockOptionLevel'] == 1)
          ].shape[0] / df_hr[df_hr['StockOptionLevel'] == 1].shape[0]*100,
    df_hr[(df_hr['Attrition'] == 'Yes') & (df_hr['StockOptionLevel'] == 2)
          ].shape[0] / df_hr[df_hr['StockOptionLevel'] == 1].shape[0]*100,
    df_hr[(df_hr['Attrition'] == 'Yes') & (df_hr['StockOptionLevel'] == 3)].shape[0] / df_
```

```
df_StockOptionLevel = pd.DataFrame(columns=["StockOptionLevel", "% of Leavers"])
```

```
i=0
for field in list(df_hr['StockOptionLevel'].unique()):
    ratio = df_hr[(df_hr['StockOptionLevel']==field)&(df_hr['Attrition']=="Yes")].shape[0]
    df_StockOptionLevel.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_SOL = df_StockOptionLevel.groupby(by="StockOptionLevel").sum()
df_SOL.iplot(kind='bar',title='Leavers by Stock Option Level (%)')
```

```
#Employee Satisfaction and Performance

df_hr['EnvironmentSatisfaction'].value_counts()
```

```
df_EnvironmentSatisfaction = pd.DataFrame(columns=["EnvironmentSatisfaction", "% of Leaver
```

```
df_EnvironmentSatisfaction = pd.DataFrame(columns=["EnvironmentSatisfaction", "% of Leaver
i=0
for field in list(df_hr['EnvironmentSatisfaction'].unique()):
    ratio = df_hr[(df_hr['EnvironmentSatisfaction']==field)&(df_hr['Attrition']=="Yes")].s
    df_EnvironmentSatisfaction.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_Env = df_EnvironmentSatisfaction.groupby(by="EnvironmentSatisfaction").sum()
df_Env.iplot(kind='bar',title='Leavers by Environment Satisfaction (%)')
```

```
df_hr['JobSatisfaction'].value_counts()
```

```
df_JobSatisfaction = pd.DataFrame(columns=["JobSatisfaction", "% of Leavers"])
i=0
for field in list(df_hr['JobSatisfaction'].unique()):
```

```
    ratio = df_hr[(df_hr['JobSatisfaction']==field)&(df_hr['Attrition']=="Yes")].shape[0]
    df_JobSatisfaction.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_JS = df_JobSatisfaction.groupby(by="JobSatisfaction").sum()
df_JS.iplot(kind='bar',title='Leavers by Job Satisfaction (%)')
```

```
df_hr['RelationshipSatisfaction'].value_counts()
```

```
df_RelationshipSatisfaction = pd.DataFrame(columns=["RelationshipSatisfaction", "% of Leav
i=0
for field in list(df_hr['RelationshipSatisfaction'].unique()):
    ratio = df_hr[(df_hr['RelationshipSatisfaction']==field)&(df_hr['Attrition']=="Yes")].
    df_RelationshipSatisfaction.loc[i] = (field, ratio*100)
```

```
    df_RelationshipSatisfaction.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_RS = df_RelationshipSatisfaction.groupby(by="RelationshipSatisfaction").sum()
df_RS.iplot(kind='bar',title='Leavers by Relationship Satisfaction (%)')
```

```
df_hr['PerformanceRating'].value_counts()
```

```
print("Normalised percentage of leavers by Stock Option Level: 3: {:.2f}%, 4: {:.2f}%".for
    df_hr[(df_hr['Attrition'] == 'Yes') & (df_hr['PerformanceRating'] == 3)
          ].shape[0] / df_hr[df_hr['StockOptionLevel'] == 1].shape[0]*100,
    df_hr[(df_hr['Attrition'] == 'Yes') & (df_hr['PerformanceRating'] == 4)].shape[0] / df_
```

```
df_PerformanceRating = pd.DataFrame(columns=["PerformanceRating", "% of Leavers"])
```

```
df_PerformanceRating = pd.DataFrame(columns=['PerformanceRating', '% of Leavers'])
i=0
for field in list(df_hr['PerformanceRating'].unique()):
    ratio = df_hr[(df_hr['PerformanceRating']==field)&(df_hr['Attrition']=="Yes")].shape[0
    df_PerformanceRating.loc[i] = (field, ratio*100)
    i += 1
    print("In {}, the ratio of leavers is {:.2f}%".format(field, ratio*100))

enable_plotly_in_cell()
df_PR = df_PerformanceRating.groupby(by="PerformanceRating").sum()
df_PR.iplot(kind='bar',title='Leavers by Performance Rating (%)')
```

```
#Attrition (Target Variable)
df_hr['Attrition'].value_counts()
```

```
print("Percentage of Current Employees is {:.1f}% and of Ex-employees is: {:.1f}%".format(
    df_hr[df_hr['Attrition'] == 'No'].shape[0] / df_hr.shape[0]*100,
    df_hr[df_hr['Attrition'] == 'Yes'].shape[0] / df_hr.shape[0]*100))
```

```
enable_plotly_in_cell()
df_hr['Attrition'].iplot(kind='hist', xTitle='Attrition',
                         yTitle='count', title='Attrition Distribution')
```

This is an imbalanced class program.

Computing Correlation

```
# Taking only significant correlations
df_HR_trans = df_hr.copy()
df_HR_trans['Target'] = df_HR_trans['Attrition'].apply(
    lambda x: 0 if x == 'No' else 1)
df_HR_trans = df_HR_trans.drop(
    ['Attrition', 'EmployeeCount', 'EmployeeNumber', 'StandardHours', 'Over18'], axis=1)
correlations = df_HR_trans.corr()['Target'].sort_values()
print('Most Positive Correlations: \n', correlations.tail(5))
print('\nMost Negative Correlations: \n', correlations.head(5))
```

```
# Calculate correlations
corr = df_HR_trans.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
# Heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(corr,
            vmax=.5,
            mask=mask,
            linewidths=.2, cmap="YlGnBu")
```

Results of Exploratory Data Analysis:

1) The data set doesn't have any missing/erraneous value

2) Strongest positive correlations with the target features are : DistanceFromHome, Monthly Rate, Rating

3) Strongest negative correlation with the target features are : Total Working Years, Job Level, Year

4) Dataset is observed to be imbalanced

5) Redundant features include : EmployeeCount, EmployeeNumber, StandardHours, and Over18.

Some observations about the people leaving :

1) Single employees comprise of the largest proportion.

2) People who will further away.

3) People who travel frequently

4) People who often work overtime

```
#Data Pre-processing (Encoding categorical labels with numerical values ,
#                      One-Hot encoding to avoid intoducing feature importance)

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

le = LabelEncoder()
print(df_hr.shape)
df_hr.head()
```

```
#label encoding for features with less than 3 unique values
le_count = 0
for col in df_hr.columns[1:]:
    if df_hr[col].dtype == 'object':
        if len(list(df_hr[col].unique())) <= 2:
            le.fit(df_hr[col])
            df_hr[col] = le.transform(df_hr[col])
            le_count += 1
print('{} columns were label encoded.'.format(le_count))
```

```
df_hr = pd.get_dummies(df_hr, drop_first=True) #Rest catergorical variables converted to d
```

```
print(df_hr.shape)
df_hr.head()
```

```
#Feature Scaling (Range 0 to 5)
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 5))
HR_col = list(df_hr.columns)
HR_col.remove('Attrition')
for col in HR_col:
    df_hr[col] = df_hr[col].astype(float)
    df_hr[[col]] = scaler.fit_transform(df_hr[[col]])
df_hr['Attrition'] = pd.to_numeric(df_hr['Attrition'], downcast='float')
df_hr.head()
```

```
print('Size of Full Encoded Dataset: {}'. format(df_hr.shape))
```

```
#Splitting data into train and test sets

target = df_hr['Attrition'].copy()

df_hr.drop(['Attrition', 'EmployeeCount', 'EmployeeNumber',
            'StandardHours', 'Over18'], axis=1, inplace=True)
print('Size of Full dataset is: {}'.format(df_hr.shape))
```

```
X_train, X_test, y_train, y_test = train_test_split(df_hr,
                                                    target,
                                                    test_size=0.25,
                                                    random_state=7,
                                                    stratify=target)
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

Building Machine learning Algortihms

```
#Algorithms implemented : Logistic Regression, Random Forest, SVM, KNN, Decision Tree Clas

models = []
models.append(('Logistic Regression', LogisticRegression(solver='liblinear', random_state=
                                                         class_weight='balanced')))
models.append(('Random Forest', RandomForestClassifier(
    n_estimators=100, random_state=7)))
models.append(('SVM', SVC(gamma='auto', random_state=7)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('Decision Tree Classifier',
               DecisionTreeClassifier(random_state=7)))
models.append(('Gaussian NB', GaussianNB()))


#Evaulating each model and providing the accuracy and standard deviation scores

acc_results = []
auc_results = []
```

```
auc_results = []
names = []

col = ['Algorithm', 'ROC AUC Mean', 'ROC AUC STD',
       'Accuracy Mean', 'Accuracy STD']
df_results = pd.DataFrame(columns=col)
i = 0
# evaluating each model using cross-validation
for name, model in models:
    kfold = model_selection.KFold(
        n_splits=10)  # 10-fold cross-validation

    cv_acc_results = model_selection.cross_val_score(  # accuracy scoring
        model, X_train, y_train, cv=kfold, scoring='accuracy')

    cv_auc_results = model_selection.cross_val_score(  # roc_auc scoring
        model, X_train, y_train, cv=kfold, scoring='roc_auc')

    acc_results.append(cv_acc_results)
    auc_results.append(cv_auc_results)
    names.append(name)
    df_results.loc[i] = [name,
                         round(cv_auc_results.mean()*100, 2),
                         round(cv_auc_results.std()*100, 2),
                         round(cv_acc_results.mean()*100, 2),
                         round(cv_acc_results.std()*100, 2)
                         ]
    i += 1
df_results.sort_values(by=['ROC AUC Mean'], ascending=False)
```

```
fig = plt.figure(figsize=(15, 7))
fig.suptitle('Algorithm Accuracy Comparison')
ax = fig.add_subplot(111)
plt.boxplot(acc_results)
ax.set_xticklabels(names)
plt.show()
```

```
fig = plt.figure(figsize=(15, 7))
fig.suptitle('Algorithm ROC AUC Comparison')
ax = fig.add_subplot(111)
plt.boxplot(auc_results)
ax.set_xticklabels(names)
plt.show()
```

```
#Logistic Regression
kfold = model_selection.KFold(n_splits=10)
modelCV = LogisticRegression(solver='liblinear',
                             class_weight="balanced",
                             random_state=7)
scoring = 'roc_auc'
results = model_selection.cross_val_score(
    modelCV, X_train, y_train, cv=kfold, scoring=scoring)
print("AUC score (STD): %.2f (%.2f)" % (results.mean(), results.std()))
```

```
#Fine tuning the hyper-parameters
param_grid = {'C': np.arange(1e-03, 2, 0.01)}
log_gs = GridSearchCV(LogisticRegression(solver='liblinear',
                                         class_weight="balanced",
                                         random_state=7),
                      iid=True,
                      return_train_score=True,
                      param_grid=param_grid,
                      scoring='roc_auc',
                      cv=10)

log_grid = log_gs.fit(X_train, y_train)
log_opt = log_grid.best_estimator_
results = log_gs.cv_results_

print('='*20)
print("best params: " + str(log_gs.best_estimator_))
print("best params: " + str(log_gs.best_params_))
print('best score:', log_gs.best_score_)
print('='*20)
```

```
#Evaluating the model

cnf_matrix = metrics.confusion_matrix(y_test, log_opt.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
```

```
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
print('Accuracy of Logistic Regression Classifier on test set: {:.2f}'.format(log_opt.scor
```

```
# Classification report for the optimised Logistic Regression
log_opt.fit(X_train, y_train)
print(classification_report(y_test, log_opt.predict(X_test)))
```

```
log_opt.fit(X_train, y_train)
probs = log_opt.predict_proba(X_test)
probs = probs[:, 1]
logit_roc_auc = roc_auc_score(y_test, probs)
print('AUC score: %.3f' % logit_roc_auc)
```

```
#Random Forest Classifier
```

```
rf_classifier = RandomForestClassifier(class_weight = "balanced"
```

```
rf_classifier = RandomForestClassifier(class_weight = 'balanced',
                                       random_state=7)
param_grid = {'n_estimators': [50, 75, 100, 125, 150, 175],
            'min_samples_leaf': [1, 2, 3, 4],
            'max_depth': [5, 10, 15, 20, 25]}

grid_obj = GridSearchCV(rf_classifier,
                        iid=True,
                        return_train_score=True,
                        param_grid=param_grid,
                        scoring='roc_auc',
                        cv=10)

grid_fit = grid_obj.fit(X_train, y_train)
rf_opt = grid_fit.best_estimator_

print('='*20)
print("best params: " + str(grid_obj.best_estimator_))
print("best params: " + str(grid_obj.best_params_))
print('best score:', grid_obj.best_score_)
print('='*20)
```

```
importances = rf_opt.feature_importances_
indices = np.argsort(importances)[::-1] # Sort feature importances in descending order
names = [X_train.columns[i] for i in indices] # Rearrange feature names so they match the
plt.figure(figsize=(15, 7))
plt.title("Feature Importance")
plt.bar(range(X_train.shape[1]), importances[indices])
plt.xticks(range(X_train.shape[1]), names, rotation=90)
plt.show()
```

```
#Random Forest helped us identify the Top 10 most important indicators

importances = rf_opt.feature_importances_
df_param_coeff = pd.DataFrame(columns=['Feature', 'Coefficient'])
for i in range(44):
    feat = X_train.columns[i]
    coeff = importances[i]
    df_param_coeff.loc[i] = (feat, coeff)
df_param_coeff.sort_values(by='Coefficient', ascending=False, inplace=True)
df_param_coeff = df_param_coeff.reset_index(drop=True)
df_param_coeff.head(10)
```

```
#Evaluating the Algorithm

cnf_matrix = metrics.confusion_matrix(y_test, rf_opt.predict(X_test))
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
print('Accuracy of RandomForest Regression Classifier on test set: {:.2f}'.format(rf_opt.s
```

```
# Classification report for the optimised RF Regression

rf_opt.fit(X_train, y_train)
print(classification_report(y_test, rf_opt.predict(X_test)))
```

```
rf_opt.fit(X_train, y_train)
probs = rf_opt.predict_proba(X_test)
probs = probs[:, 1]
rf_opt_roc_auc = roc_auc_score(y_test, probs)
print('AUC score: %.3f' % rf_opt_roc_auc)
```

```
# Create ROC Graph
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, log_opt.predict_proba(X_test)[:,1])
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, rf_opt.predict_proba(X_test)[:,1])
plt.figure(figsize=(14, 6))

# 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_opt_roc_auc)
# Plot Base Rate ROC
plt.plot([0,1], [0,1],label='Base Rate' 'k--')

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.show()
```

The fine-tuned Logistic Regression model showed a higher AUC score compared to the Random F