EMPLOYEE CHURN PREDICTION

A project report submitted to Department of Computer Science and Engineering for partial fulfilments of the requirements for the award of the degree

of

BACHELOR OF

TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

B.Teja Sri Venkat, 121910317023

Jana Venkata Chaitanya, 121910317060

M.Subhani, 121910317038

S.Sai Teja, 121910317048

Under the esteemed guidance of

Smt N. Madhuri

DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING

GITAM

(Deemed to be University)

VISAKHAPATNAM

OCTOBER 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

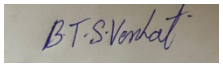
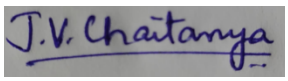
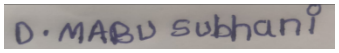
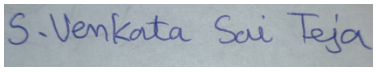
GITAM

(Deemed to be University)

DECLARATION

We, hereby declare that the project report entitled "**Employee Churn Prediction**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:17-10-2022

Registration No(s).	Name(s)	Signature(s)
121910317023	**B.Teja Sri Venkat**	
121910317060	**Jana Venkata Chaitanya**	
121910317038	**D.Subhani**	
121910317048	**S.Sai Teja**	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled "**Employee Churn Prediction**" is a bonafide record of work carried out by **B.Teja Sri Venkat(121910317023), D.Subhani(121910317038), J.V.Chaitanya(121910317060), S.Sai Teja(121910317048)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	**Head of the Department**
Smt N. Madhuri	**Dr. R. Sireesha**
Assistant Professor	**Professor**

TABLE OF CONTENTS

TABLE OF DIAGRAMS

ABSTRACT

An employee is a valuable resource who works for a company. If one of them abruptly departs the company, it may have a significant impact and expense to the specific company. And, in addition to time and money, the newly hired employee requires some time to make the particular firm cost-effective. This model will assist in predicting the rate at which people quit occupations based on available analytic data and will apply various machine learning algorithms to reduce prediction error. The prediction of a personalized or individual employee varies depending on the setting in which they work. While it has become clear that employee churn forecasts differ depending on geography, lifestyle, and environment, the associated knowledge and understanding remains fragmented.

INTRODUCTION

Employee churn refers to the departure or retirement of a knowledgeable individual or asset from a business. Instead, we can state that churn occurs when an employee departs an organization. This churn could be forced or voluntary on the part of the employee. Employee churn refers to any employee who wishes to leave a company or whose contract duration has expired, resulting in an unfavorable reduction in the organization. When former employees are replaced by newly hired employees, this is referred to as employee turnover. Both turnover and attrition are linked. This raises the entire cost of acquiring new personnel, managing the recruitment process, and training newly hired employees. All minor issues that arise following an employee's departure from the organization contribute to the integration of subsequent issues that may have an indirect impact on the company. Employee turnover is unavoidable in any sector. Avoiding such a massive disaster is a difficult task for everyone. Companies can avoid this problem by knowing the factor generating abrupt attrition. They can also inspire employees to stay with the organization. The employment and termination conditions determine the frequency with which employees leave an organization. Each employee may have the same or a different reason for leaving a company. To reduce turnover, the human resource team and management must carefully select the parameters.

Employee turnover occurs for a variety of reasons. If the company is losing money, some employees may be let go, or they may leave for a better future and job satisfaction. This prompted the search for a new job replacement. In order for the firm to work effectively and profitably, we must implement a good churn prediction system that will not only reduce the possibility of major financial losses but will also support overall growth of each and every employee.

LITERATURE REVIEW

The Techniques used in [1] are Area Under Curve, Standard mechanism and Social network analysis and the model takes full account of tiny features in the dataset while training, thus detecting smaller objects more efficiently. The AUD & SNA is a lightweight model, multiple.

The techniques used in [2] are Stratified K-fold cross validation technique, the model consists of Different algorithms were investigated to get good precision and recall. Through a series of experiments, It is concluded that Random Forest model Performs better than other algorithms.

The techniques used in [3] are Artificial neural networks, logistic regression, classification and regression trees (CART), classification trees (C5.0), and discriminant analysis) .This technique is capable of handling multiple objects using ANN, regression and classification techniques. It is capable of dealing large data using the simplest algorithms present.

The techniques used in [4] are Naïve Bayes, Support Vector Machines, Logistic Regression, Decision Trees and Random Forests. Demonstration of SVM technique can be used to build reliable and accurate predictive models for employee churn

The techniques used in [5] are Multiple analysis of variance (MANOVA), independent t-test, factor analysis, multiple regression, and binomial logistic regression analysis. The hypothesis was not completely supported when all of the factors required to continue with the prediction were considered. It is limited to a single company and cannot be used for all convenience.

PROBLEM IDENTIFICATION

Employee Churn is an expensive issue for businesses. The true cost of replacing an employee is frequently fairly high. According to a research conducted by the Center for American Progress, corporations normally pay around one-fifth of an employee's compensation to replace that individual, and the cost can skyrocket if CEOs or the highest-paid staff have to be replaced. In other words, the cost of replacing staff remains high for the majority of firms. This is due to the time spent interviewing and finding a successor, sign-on bonuses, and the loss of production for several months as the new employee adjusts to the new role. Previously, the majority of the focus was on "rates" such as attrition and retention rates. Using data warehousing techniques, HR managers compute prior rates and attempt to anticipate future rates. These rates represent the collective impact of churn, but they only provide half of the picture. Another strategy is to concentrate on individual records rather than aggregates. There are numerous case studies available on client churn. Employee turnover is analogous to consumer turnover. It is primarily concerned with the employee rather than the customer. You can predict who and when an employee will leave the company. Employee attrition is costly, and minor improvements will yield huge results.

OBJECTIVES

The main Objective of the Project is to solve the issue of employee attrition and retention.So the HR department can recruit the employee as they desire . The algorithm works of the previous and past source data of the employee , the data collected is in the form of a dataset and undergoes various process as discussed below and generates the attrition rate

1. DATA COLLECTION: A prediction system's first phase entails gathering data and dividing it into training and testing datasets. In this project, 30% of the dataset was used for testing and 70% was used for training. Employee datasets include all the information needed to predict employee attrition, including employee id, satisfaction level, most recent evaluation, number of projects, average monthly hours worked, time spent at the company (years), departments, and salary level.

2. ATTRIBUTE ANALYSIS: Radar charts, correlation matrices, and pair plots are used as data visualization techniques to examine the dataset's properties. The characteristics that significantly influence employee attrition are examined. During the procedure, irrelevant attributes like employee ID are removed.

3. DATA PREPROCESSING: Data preprocessing is crucial to machine learning models since it converts the data to the necessary format. This keeps the results from being misinterpreted. The issues of redundant data, unbalanced data, missing values, and categorical values are dealt with during data pre-processing. Dummy variables are dealt with using label encoding.

4. EMPLOYEE CHURN: Different machine learning classification models are used for classification, including Decision Tree, KNN, SVC, and Light GBM. Comparative analysis is used to compare the algorithms. In the suggested approach, the algorithm that has the highest accuracy rating is used to forecast Employee churn.

SYSTEM METHODOLOGY

The project's suggested system architecture provides a summary of how the system functions. The project begins with data gathering, followed by attribute analysis and data pre-processing. The data is preprocessed into the necessary format so that machine learning classification techniques can be used. The data must be divided into training and testing groups in the following phase. On the training dataset, machine learning techniques are used, and on the testing dataset, performance metric analysis is performed later. The following modules are included in this project:
1. Dataset collection 2. Attribute analysis 3. Data Preprocessing 4. Employee churn

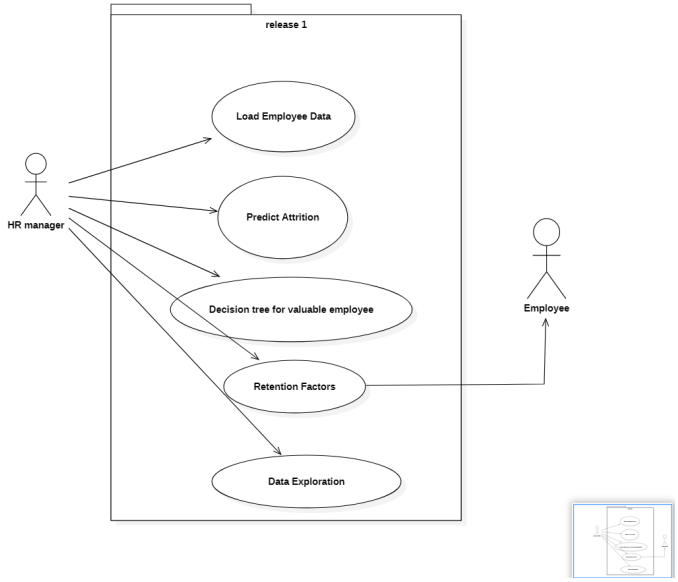


Figure 1: Use Case Diagram

As the diagram shows both the HR manager and employee have their roles to play .And their Operations are performed in a sequential manner .First the HR loads the data and basing on the data the attrition prediction will be done and basing on the attrition rate retention factors of the employee are observed.

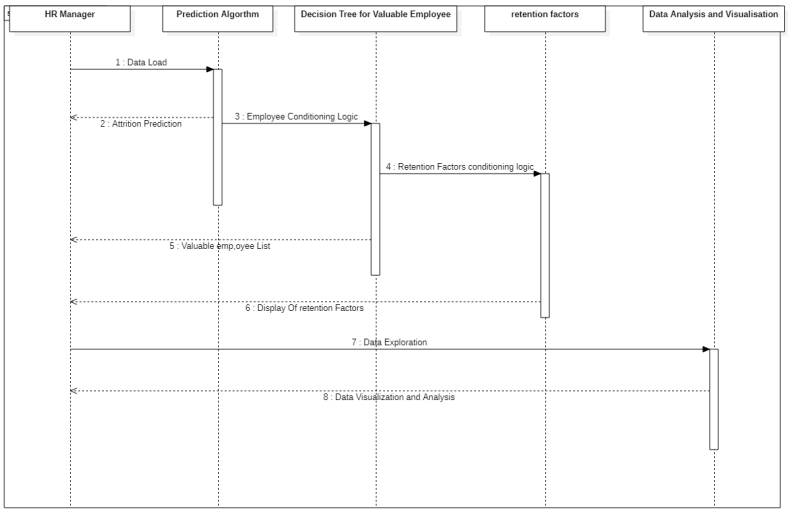


Figure 2: Sequence Diagram

This Diagram depicts the overall operations occurring in a correct sequential manner performed from time to time by various users

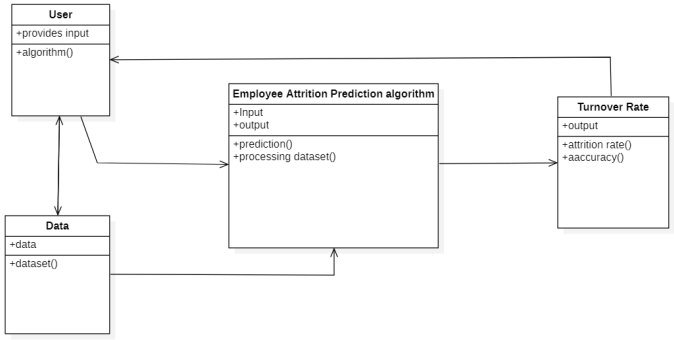


Figure 3: Class Diagram

The attributes and operations performed by various Class entities are given .And this also shows how various entities are correlated to each other

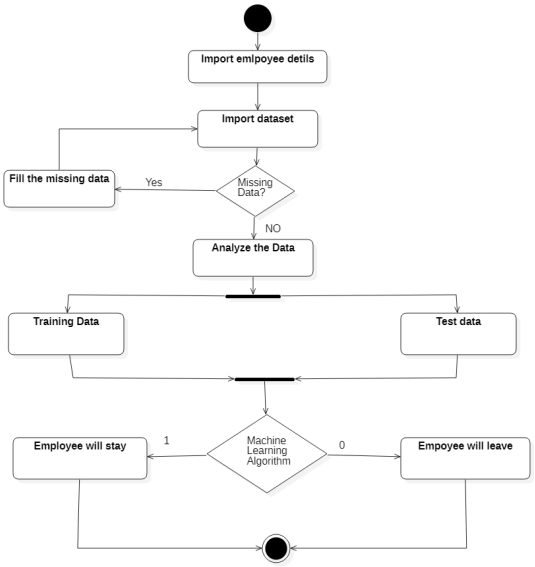


Figure 4: State Chart Diagram

In Activity and State Chart diagrams ,it shows various transactions between objects and other related data in a step by step order

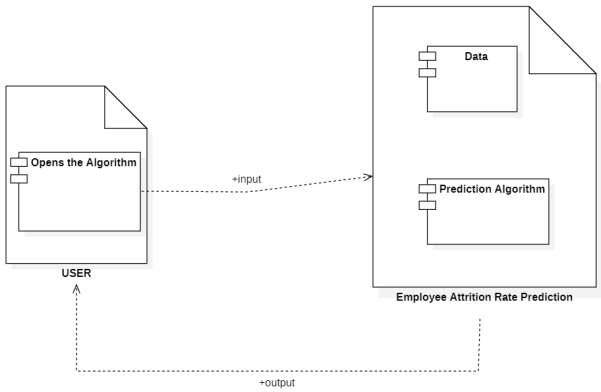


Figure 5: Component Diagram

Shows all the Components and the internal processes occurring in the algorithm .This practically displays all the internal operations from Input to the Output

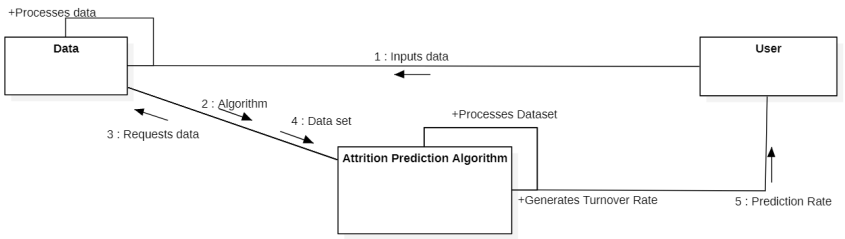


Figure 6: Communication Diagram

Provides all the information of how the internal communication is going between all the working components in the algorithm

OVERVIEW OF TECHNOLOGIES

GRADIENT BOOSTING ALGORITHM:

One of the most potent algorithms used in machine learning is the gradient boosting technique. As is well known, bias and variance mistakes can be used to broadly categorize errors in machine learning systems. As one of the boosting strategies, gradient boosting is used to reduce the model's bias error.

We are unable to discuss the base estimator in the gradient boosting process, unlike the Adaboosting algorithm. The Gradient Boost algorithm's default estimator, Decision Stump, is fixed. The gradient boosting algorithm's n estimator can be adjusted, just like AdaBoost. The default value of n estimator for this algorithm, however, is 100 if we do not specify a number for it.

Both continuous and categorical target variables can be predicted using the gradient boosting approach (as a Regressor) (as a Classifier). The cost function is Mean Square Error (MSE) when it is used as a regressor, while it is Log loss when it is used as a classifier.

GRADIENT DESCENT CLASSIFIER:

In order to train machine learning models, gradient descent is one of the most often utilized optimization techniques. It does this by reducing the discrepancy between the actual and expected outputs. Furthermore, neural networks are trained using gradient descent.

The term "optimization algorithm" in mathematics describes the process of minimizing or maximizing an objective function f(x) parameterized by x. Similar to this, the goal of optimization in machine learning is to reduce the cost function, which is parameterized by the model's parameters. Utilizing parameter updates in iterations, the primary goal of gradient descent is to minimize the convex function. These machine learning models can be utilized as effective tools for artificial intelligence and a variety of computer science applications once they have been tuned.

KNN ALGORITHM:

With k-NN, all computation is postponed until after the function has been evaluated and the function is only locally approximated. Since this technique relies on distance for classification, normalizing the training data can significantly increase accuracy if the features reflect several physical units or have distinct sizes.

Assigning weights to neighbor contributions can be a helpful strategy for both classification and regression, making the closer neighbor contribute more to the average than the farther neighbor. As an illustration, a typical weighting method assigns each neighbor a weight of 1/d, where d is the distance between the neighbors.

When using k-NN classification or regression, the neighbors are chosen from a set of objects for which the class or object property value is known. Although there is no need for an explicit training step, this can be considered of as the algorithm's training set.

The k-NN algorithm has the feature of being sensitive to the local structure of the data. The existence of noisy or irrelevant features, or if the feature scales are inconsistent with their relevance, can significantly reduce the accuracy of the k-NN method. A lot of study has gone into choosing or scaling characteristics to enhance classification. The application of evolutionary algorithms to enhance feature scaling is a particularly well-liked method. Scaling features based on the mutual information of training data and training classes is a popular alternative.

RANDOM FOREST CLASSIFIER:

A large number of decision trees are built during the training phase of the random forests or random decision forests ensemble learning approach, which is used for classification, regression, and other tasks. The class that the majority of the trees chose is the output of the random forest for classification problems. The mean or average prediction of each individual tree is returned for regression tasks. The tendency of decision trees to overfit their training set is corrected by random decision forests. Although they frequently outperform decision trees, gradient boosted trees are more accurate than random forests. [Reference needed] However, their effectiveness may be impacted by data peculiarities.
Random forest predictors automatically produce a dissimilarity metric between the observations as part of their creation. The objective is to build a random forest predictor that separates the "observed" data from adequately generated synthetic data. This is also known as a random forest dissimilarity measure between unlabeled data. The synthetic data are taken from a reference distribution, whereas the observed data are the original unlabeled data.

CATBOOST CLASSIFIER:

CatBoost is a technique for decision trees that uses gradient boosting. It is created by Yandex researchers and engineers and is used for a variety of jobs at Yandex and in other businesses, such as CERN, Cloudflare, and Careem taxi, including search, recommendation systems, personal assistants, self-driving cars, weather prediction, and many more. Everyone is welcome to use it because it is open-source.

There are numerous methods for handling categorical features in boosted trees, which are frequently present in datasets. CatBoost automatically handles categorical features in contrast to other gradient boosting techniques (which need numeric input). One-hot encoding is one of the most popular methods for dealing with categorical data, however it is impractical for many characteristics. Target statistics are used to categorize features in order to address this (estimate target value for each category). There are several techniques to calculate the target statistics: greedy, hold out, leave one out, and ordered. Target statistics are ordered in CatBoost.

IMPLEMENTATION

1. Coding

```
!pip install pyforest
!pip install --user pycaret
!pip install pandas_profiling
!pip install squarify
!pip install termcolor
!pip install catboost
!pip install pyclustertend
```

```
import pandas_profiling
import pyforest

import ipywidgets
from ipywidgets import interact

import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.ticker as mticker
import squarify as sq
```

[1] Python

```
import plotly.express as px
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)
```

[2] Python

...

```
pip install yellowbrick
```

[3] Python

```
pip install --upgrade scikit-learn
```

[4] Python

```
import colorama
from colorama import Fore, Style
from termcolor import colored
from termcolor import cprint



import scipy.stats as stats
from scipy.cluster.hierarchy import linkage, dendrogram
import statsmodels.api as sm
import statsmodels.formula.api as smf
import missingno as msno

import datetime as dt
from datetime import datetime

from pyclustertend import hopkins
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.compose import make_column_transformer
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor, AdaBoostClassifier
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif, f_regression, mutual_info_regression
```

```python
from scipy.cluster.hierarchy import linkage, dendrogram
import statsmodels.api as sm
import statsmodels.formula.api as smf
import missingno as msno

import datetime as dt
from datetime import datetime

from pyclustertend import hopkins
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.compose import make_column_transformer
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor, AdaBoostClassifier
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif, f_regression, mutual_info_regression
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet


from sklearn.metrics import silhouette_samples,silhouette_score
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold, KFold, cross_val_predict, train_test_split
from sklearn.model_selection import StratifiedKFold, GridSearchCV, cross_val_score, cross_validate
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.preprocessing import scale, StandardScaler, MinMaxScaler, RobustScaler
from sklearn.preprocessing import PolynomialFeatures, OneHotEncoder, PowerTransformer, LabelEncoder
from sklearn.svm import SVR, SVC
from sklearn.tree import plot_tree, DecisionTreeClassifier
from yellowbrick.classifier import ClassificationReport

from xgboost import XGBRegressor, XGBClassifier, plot_importance
```

```python
def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending = False)
    missing_percent = (df.isnull().sum() / df.isnull().count()).sort_values(ascending = False)
    missing_values = pd.concat([missing_number, missing_percent], axis = 1, keys = ['Missing_Number', 'Missing_Percent'])
    return missing_values[missing_values['Missing_Number'] > 0]



def first_looking(df):
    print(colored("Shape:", attrs=['bold']), df.shape,'\n',
          colored('*'*100, 'red', attrs = ['bold']),
          colored("\nInfo:\n", attrs = ['bold']), sep = '')
    print(df.info(), '\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Number of Uniques:\n", attrs = ['bold']), df.nunique(),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Missing Values:\n", attrs=['bold']), missing_values(df),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("All Columns:", attrs = ['bold']), list(df.columns),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')

    df.columns = df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')
    print(colored("Columns after rename:", attrs = ['bold']), list(df.columns),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Columns after rename:", attrs = ['bold']), list(df.columns),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '')
    print(colored("Descriptive Statistics \n", attrs = ['bold']), df.describe().round(2),'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '') # Gives a statstical breakdown of the data.
    print(colored("Descriptive Statistics (Categorical Columns) \n", attrs = ['bold']), df.describe(include = object).T,'\n',
          colored('*'*100, 'red', attrs = ['bold']), sep = '') # Gives a statstical breakdown of the data.

def multicolinearity_control(df):
    feature = []
    collinear = []
    for col in df.corr().columns:
        for i in df.corr().index:
```

```python
def multicolinearity_control(df):
    feature = []
    collinear = []
    for col in df.corr().columns:
        for i in df.corr().index:
            if (abs(df.corr()[col][i]) > .9 and abs(df.corr()[col][i]) < 1):
                    feature.append(col)
                    collinear.append(i)
                    print(colored(f"Multicolinearity alert in between:{col} - {i}",
                                  "red", attrs = ["bold"]), df.shape,'\n',
                                  colored('*'*100, 'red', attrs = ['bold']), sep = '')

def duplicate_values(df):
    print(colored("Duplicate check...", attrs = ['bold']), sep = '')
    print("There are", df.duplicated(subset = None, keep = 'first').sum(), "duplicated observations in the dataset.")
    duplicate_values = df.duplicated(subset = None, keep = 'first').sum()
    if duplicate_values > 0:
        df.drop_duplicates(keep = 'first', inplace = True)
        print(duplicate_values, colored(" Duplicates were dropped!"),'\n',
              colored('*'*100, 'red', attrs = ['bold']), sep = '')

def drop_null(df, limit):
    print('Shape:', df.shape)
    for i in df.isnull().sum().index:
        if (df.isnull().sum()[i] / df.shape[0]*100) > limit:
            print(df.isnull().sum()[i], 'percent of', i ,'null and were dropped')
            df.drop(i, axis = 1, inplace = True)
            print('new shape:', df.shape)
    print('New shape after missing value control:', df.shape)


def first_look(col):
    print("column name    : ", col)
    print("----------------------------")
```

```python
def first_look(col):
    print("column name    : ", col)
    print("----------------------------------")
    print("Per_of_Nulls   : ", "%", round(df[col].isnull().sum() / df.shape[0]*100, 2))
    print("Num_of_Nulls   : ", df[col].isnull().sum())
    print("Num_of_Uniques : ", df[col].nunique())
    print("Duplicates     : ", df.duplicated(subset = None, keep = 'first').sum())
    print(df[col].value_counts(dropna = False))



def fill_most(df, group_col, col_name):
    '''Fills the missing values with the most existing value (mode) in the relevant column according to single-stage grouping'''
    for group in list(df[group_col].unique()):
        cond = df[group_col] == group
        mode = list(df[cond][col_name].mode())
        if mode != []:
            df.loc[cond, col_name] = df.loc[cond, col_name].fillna(df[cond][col_name].mode()[0])
        else:
            df.loc[cond, col_name] = df.loc[cond, col_name].fillna(df[col_name].mode()[0])
    print("Number of NaN : ",df[col_name].isnull().sum())
    print("------------------")
    print(df[col_name].value_counts(dropna = False))


def show_values_on_bars(axs):
    def _show_on_single_plot(ax):
        for p in ax.patches:
            _x = p.get_x() + p.get_width() / 2
            _y = p.get_y() + p.get_height()
            value = '{:.2f}'.format(p.get_height())
            ax.text(_x, _y, value, ha="center")
    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)
```

```python
df0 = pd.read_csv('HR_Dataset.csv')
df = df0.copy()
df.head(3)
```

Python

```python
df.profile_report()
```

Python

```python
first_looking(df)
duplicate_values(df)
print(colored("Shape:", attrs = ['bold']), df.shape,'\n', colored('*'*100, 'red', attrs = ['bold']))
```

Python

```python
df.columns
```

Python

```python
df.rename({'departments_': 'department'}, axis=1, inplace=True)
df.head(1)
```

Python

```python
df = df[['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'work_accident',
       'promotion_last_5years', 'department', 'salary', 'left']]
df.head(1)
```

Python

```python
cprint("Have a First Look to 'left' Column",'green')
first_look('left')
```

Python

```python
fig = px.pie(df, values = df['left'].value_counts(),
             names = (df['left'].value_counts()).index,
             title = '"left" Column Distribution')
fig.show()
```

Python

```python
y = df['left']
print(f'Percentage of left-1: % {round(y.value_counts(normalize=True)[1]*100,2)} --> \
({y.value_counts()[1]} observations for left-1)\nPercentage of left-0: % {round(y.value_counts(normalize=True)[0]*100,2)} --> ({y.value_counts()[0]} observations for left-0)')
```

Python

```python
df.groupby('left').mean()
```

```python
cprint('Dataset describe results according to the "left==1" condition','green')
df[df['left'] == 1].describe().T.style.background_gradient(subset = ['mean','min','50%', 'max'], cmap = 'RdPu')
```

Python

```python
cprint('Dataset describe results according to the "left==0" condition','green')
df[df['left'] == 0].describe().T.style.background_gradient(subset = ['mean','min','50%', 'max'], cmap = 'RdPu')
```

Python

```python
cprint("Have a First Look to 'left' Column",'green')
first_look('satisfaction_level')
```

Python

```python
df['satisfaction_level'].value_counts().iplot(kind="bar", title = '"satisfaction_level" Column Distribution')
```

Python

```python
px.histogram(df, x = df['satisfaction_level'], color='left', marginal = "box", hover_data = df.columns,
             title = 'satisfaction_level and left')
```

Python

```python
pd.crosstab(df['satisfaction_level'], df['left']).iplot(kind='bar', title = 'satisfaction_level and left')
```

Python

```python
sns.scatterplot(data=df, x="satisfaction_level", y="left", hue="left");
```

```python
df.corr(method='pearson')[7:]
```

```python
cprint("Have a First Look to 'last_evaluation' Column",'green')
first_look('last_evaluation')
```

```python
df['last_evaluation'].value_counts().iplot(kind="bar", title = '"last_evaluation" Column Distribution')
```

```python
px.histogram(df, x = df['last_evaluation'], color='left', marginal = "box", hover_data = df.columns,
             title = 'last_evaluation and left')
```

```python
pd.crosstab(df['last_evaluation'], df['left']).iplot(kind='bar', title = 'last_evaluation and left')
```

```python
cprint("Have a First Look to 'number_project' Column",'green')
first_look('number_project')
```

```python
df['number_project'].value_counts().iplot(kind="bar", title = '"number_project" Column Distribution')
```

```python
fig = px.pie(df, values = df['number_project'].value_counts(),
             names = (df['number_project'].value_counts()).index,
             title = '"number_project" Column Distribution')
fig.show()
```

```python
px.histogram(df, x = df['number_project'], color='left', marginal = "box", hover_data = df.columns,
             title = 'number_project and left')
```

```python
pd.crosstab(df['number_project'], df['left']).iplot(kind='bar', title = 'number_project and left')
```

```python
cprint("Have a First Look to 'average_montly_hours' Column",'green')
first_look('average_montly_hours')
```

```python
plt.figure(figsize=(15, 8))
fig = sns.histplot(
    df,
    x="average_montly_hours", hue="left",
    multiple="stack",
    palette="light:m_r",
    edgecolor=".3",
    linewidth=.5
)
fig2 = fig.twinx()

fig.yaxis.set_label_position('left')
fig2.yaxis.set_label_position('right')

fig2.set_ylabel('frequency [%]')

for p in fig.patches:
    x=p.get_bbox().get_points()[:,0]
    y=p.get_bbox().get_points()[1,1]
    fig.annotate('{:.1f}%'.format(100.*y/len(df)), (x.mean(), y),
                 ha='center', va='bottom') # set the alignment of the text
fig2.grid(None)
fig2.set_ylim(0,100)
```

```python
numerical= df.drop(['left'], axis=1).select_dtypes('number').columns

categorical = df.select_dtypes('object').columns

print(colored("Numerical Columns:", attrs=['bold']), list(df[numerical].columns),'\n',
              colored('-'*124, 'red', attrs=['bold']), sep='')
print(colored("Categorical Columns:", attrs=['bold']), list(df[categorical].columns),'\n',
              colored('-'*124, 'red', attrs=['bold']), sep='')
```

```python
df[categorical].head().T
```

```python
df[categorical].describe().T.style.background_gradient(subset=['unique','freq','count'], cmap='RdPu')
```

```python
df[categorical].nunique()
```

```python
for i, col in enumerate(df[categorical].columns):
    fig = px.histogram(df[col], color=df["left"], width=800, height=800, title=col, pattern_shape=df["salary"], pattern_shape_sequence=["x", "-", "+"])
    fig.show()
```

```python
for i, col in enumerate(df[categorical].columns):
    xtab = pd.crosstab(df[col], df["left"], normalize=True)
    print(colored('-'*40, 'red', attrs=['bold']), sep='')
    print(xtab*100)
```

```python
numerical= df.drop(['left'], axis = 1).select_dtypes('number').columns

categorical = df.select_dtypes('object').columns

print('--------------------')
print(f'Numerical Columns:  {df[numerical].columns}')
print(f'Categorical Columns: {df[categorical].columns}')
print('--------------------')
```

[42] Python

```python
df[numerical].describe().T.style.background_gradient(subset = ['mean','std','50%','count'], cmap = 'RdPu')
```

[43] Python

```python
df[numerical].iplot(kind = 'histogram', subplots = True, bins = 50)
```

[44] Python

```python
for i in numerical:
    df[i].iplot(kind = 'box', title = i, boxpoints = 'all')
```

[45] Python

```python
sns.pairplot(df, hue = "left", corner = True);
```

[46] Python

```python
df.corr()['left'].sort_values().drop('left').iplot(kind = 'barh', colors='purple');
```

[48] Python

```python
df_temp = df.corr()

count = 'Done'
feature =[]
collinear= []
for col in df_temp.columns:
    for i in df_temp.index:
        if (df_temp[col][i] > .9 and df_temp[col][i] < 1) or (df_temp[col][i] < -.9 and df_temp[col][i] > -1) :
                feature.append(col)
                collinear.append(i)
                print(Fore.RED + f'\033[1mmulticolinearity alert in between\033[0m {col} - {i}')
        else:
            print(f'For {col} and {i}, there is NO multicollinearity problem')

print('\033[1mThe number of strong corelated features:\033[0m', count)
```

[49] Python

```python
plt.figure(figsize = (7,5))
sns.countplot(data = df, x = 'left');
for index,value in enumerate(df.left.value_counts()):
    plt.text(index, value, f'{value}', ha = 'center', va = 'bottom', fontsize = 13);
```

[53] Python

```python
cprint('"left" Column Distribution','green')
fig = plt.figure(figsize = (11, 6))
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(df.left.value_counts().index, df.left.value_counts().values, color = 'green')
plt.title('"left" Column Distribution')
plt.xlabel('left')
plt.ylabel('Number of Employees')
for index,value in enumerate(df.left.value_counts()):
    plt.text(index, value, f'{value}', ha = 'center', va = 'bottom', fontsize = 13)
plt.show()
```

[54] Python

```python
cprint('"number_project" Column Distribution','green')
df.number_project.value_counts()
```

[55] Python

```python
df['number_project'].value_counts().iplot(kind="bar", title = '"number_project" Column Distribution')
```

[56] Python

```python
plt.xticks(rotation = 0)
for index,value in enumerate(df.number_project.value_counts().sort_values(ascending=False)):
    plt.text(index, value, f'{value}', ha = 'center', va = 'bottom', fontsize = 13)
plt.show()
```

[57] Python

```python
cprint('"time_spend_company" Column Distribution','green')
df.time_spend_company.value_counts()
```

[58] Python

```python
df['time_spend_company'].value_counts().iplot(kind="bar", title = '"time_spend_company" Column Distribution')
```

[59] Python

```python
plt.ylabel('Number of Employees')
plt.xticks(rotation = 0)
for index,value in enumerate(df.time_spend_company.value_counts().sort_values(ascending=False)):
    plt.text(index, value, f'{value}', ha = 'center', va = 'bottom', fontsize = 13)
plt.show()
```

[60] Python

```python
for i in df:
    df[i].iplot(kind = 'histogram', subplots = True, bins = 50)
```

[61] Python

```python
df1 = df.drop('left', axis = 1)
df1.head(1)
```

[62] Python

```python
df1 = pd.get_dummies(df1, columns = ['department','salary'], drop_first = True)
df1.head(1)
```

[63] Python

```python
df1.head()
```

[64] Python

```python
scaler = MinMaxScaler()
scaler.fit(df1)

df1_scaled= scaler.transform(df1)
```

Python

```python
df1_scaled
```

Python

```python
df1_scaled
```

Python

```python
from pyclustertend import hopkins

hopkins(df1_scaled, df1.shape[0])
```

Python

```python
ks = range(1,10)
inertias=[]
for k in ks :

    kc = KMeans(n_clusters=k,random_state=1)
    kc.fit(df1_scaled)
    inertias.append(kc.inertia_)

f, ax = plt.subplots(figsize=(8, 6))
plt.plot(ks, inertias, '-o')
plt.xlabel('Number of clusters, k')
plt.ylabel('Inertia')
plt.xticks(ks)
plt.style.use('ggplot')
plt.title('What is the Best Number for KMeans ?')
plt.show()
```

```python
from yellowbrick.cluster import KElbowVisualizer

kmeans = KMeans()
visu = KElbowVisualizer(kmeans, k = (1,10))
visu.fit(df1_scaled)
visu.show();
```

Python

```python
from sklearn.metrics import silhouette_samples,silhouette_score

ssd =[]

K = range(2, 10)

for k in K:
    model = KMeans(n_clusters=k)
    model.fit(df1_scaled)
    ssd.append(model.inertia_)
    print(f'Silhouette Score for {k} clusters: {silhouette_score(df1_scaled, model.labels_)}')
```

Python

```python
from sklearn.cluster import KMeans

from yellowbrick.cluster import SilhouetteVisualizer

model_4 = KMeans(n_clusters=4, random_state=101)
visualizer = SilhouetteVisualizer(model_4)
visualizer.fit(df1_scaled)
visualizer.poof();
```

Python

```python
from sklearn.cluster import KMeans

from yellowbrick.cluster import SilhouetteVisualizer

model_3 = KMeans(n_clusters=3, random_state=101)
visualizer = SilhouetteVisualizer(model_3)
visualizer.fit(df1_scaled)
visualizer.poof();
```

Python

```python
k_means_model = KMeans(n_clusters = 3, random_state = 101)
k_means_model.fit_predict(df1_scaled)
labels = k_means_model.labels_
labels
```

Python

```python
df['predicted_clusters'] = labels
df
```

Python

```python
df['predicted_clusters'].value_counts()
```

Python

```python
fig = px.pie(df, values = df['predicted_clusters'].value_counts(),
             names = (df['predicted_clusters'].value_counts()).index,
             title = 'Predicted_Clusters Distribution')
fig.show()
```

Python

```python
fig = px.pie(df, values = df[df['left']==0]['predicted_clusters'].value_counts(),
             names = df[df['left']==0]['predicted_clusters'].value_counts().index,
             title = 'Predicted_Clusters_Almost_Lost Distribution')
fig.show()
```

Python

```python
fig = px.pie(df, values = df[df['left']==1]['predicted_clusters'].value_counts(),
             names = df[df['left']==1]['predicted_clusters'].value_counts().index,
             title = 'Predicted_Clusters_Almost_Lost Distribution')
fig.show()
```

Python

```python
pd.crosstab(df['left'],
            df['predicted_clusters']).iplot(kind="bar", title = 'Compare (left vs predicted-clusters)',
            xTitle = 'left & clusters', yTitle = 'counts')
```

Python

```python
df.groupby('left').mean()
```

Python

```
[81] df.groupby(['left', 'predicted_clusters']).mean()
```
Python

```
[82] pd.crosstab(df['predicted_clusters'],
                 df['left']).iplot(kind="bar", title = 'Compare (predicted-clusters vs left)',
                 xTitle = 'clusters & left', yTitle = 'counts')
```
Python

```
[83] df.left.value_counts()
```
Python

```
[84] df.groupby('predicted_clusters').mean()
```
Python

```
[85] df.groupby(['predicted_clusters', 'left']).mean()
```
Python

```
[86] df2 = df.drop('predicted_clusters', axis = 1)
     df2.head(1)
```
Python

```
[87] df2 = pd.get_dummies(df2, columns = ['department','salary'], drop_first = True)
     df2.head(1)
```
Python

```
[88] X = df2.drop('left', axis = 1)
     y = df2['left']
```
Python

```
[89] X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.3, random_state = 101)
```
Python

```
[90] scaler = MinMaxScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```
Python

```
[163] def eval(model, X_train, X_test):
          y_pred = model.predict(X_test)
          y_pred_train = model.predict(X_train)

          print(confusion_matrix(y_test, y_pred))
          print("Test_Set")
          print(classification_report(y_test,y_pred))
          print("Train_Set")
          print(classification_report(y_train,y_pred_train))
          cm = confusion_matrix(y_test,y_pred)
          disp = ConfusionMatrixDisplay(confusion_matrix=cm)
          disp.plot(model, X_test, y_test, cmap="plasma")
```
Python

```
[164] def train_val(y_train, y_train_pred, y_test, y_pred):

          scores = {"train_set": {"Accuracy" : accuracy_score(y_train, y_train_pred),
                                  "Precision" : precision_score(y_train, y_train_pred),
                                  "Recall" : recall_score(y_train, y_train_pred),
                                  "f1" : f1_score(y_train, y_train_pred)},

                    "test_set": {"Accuracy" : accuracy_score(y_test, y_pred),
                                 "Precision" : precision_score(y_test, y_pred),
                                 "Recall" : recall_score(y_test, y_pred),
                                 "f1" : f1_score(y_test, y_pred)}}

          return pd.DataFrame(scores)
```
Python

```
[165] from sklearn.metrics import RocCurveDisplay, roc_auc_score, roc_curve, f1_score, accuracy_score, recall_score
```
Python

```
[166] GB_model = GradientBoostingClassifier(random_state = 101)
      GB_model.fit(X_train, y_train)
      y_pred = GB_model.predict(X_test)
      y_train_pred = GB_model.predict(X_train)

      GB_model_f1 = f1_score(y_test, y_pred)
      GB_model_acc = accuracy_score(y_test, y_pred)
      GB_model_recall = recall_score(y_test, y_pred)
      GB_model_auc = roc_auc_score(y_test, y_pred)
```
Python

...

```
[167] from sklearn.metrics import classification_report
      from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```
Python

```
[168] print("GB_Model")
      print ("--------------------")
      eval(GB_model, X_train, X_test)
```
Python

```
[169] cf_matrix = confusion_matrix(y_test, y_pred)

      group_names = ["True Negatives (TN)", "False Positives (FP)\n(Type I Error)", "False Negatives (FN)\n(Type II Error)", "True Positives (TP)"]

      group_counts = ["{0:0.0f}".format(value) for value in
                      cf_matrix.flatten()]

      group_percentages = ["{0:.2%}".format(value) for value in
                           cf_matrix.flatten()/np.sum(cf_matrix)]

      labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
                zip(group_names, group_counts, group_percentages)]

      labels = np.asarray(labels).reshape(2, 2)

      ax = sns.heatmap(cf_matrix, annot=labels, fmt="", cmap="Blues")
      ax.set(xlabel="Predicted Class", ylabel = "Actual Classs");
```
Python

```
plt.figure(figsize=(12, 8))
visualizer = ClassificationReport(GB_model, support=True)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```
[170] Python

```
from yellowbrick.classifier import ClassPredictionError

visualizer = ClassPredictionError(GB_model)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.poof();
```
[171] Python

```
GB_feature_imp = pd.DataFrame(index=X.columns, data = GB_model.feature_importances_, columns = ['Importance']).sort_values("Importance", ascending = False)
GB_feature_imp
```
[173] Python

```
plt.figure(figsize = (12,6))
sns.barplot(data = GB_feature_imp.sort_values('Importance', ascending = False), x = GB_feature_imp.sort_values('Importance', ascending = False).index, y = 'Importance')
plt.xticks(rotation = 75);
```
[174] Python

```
GB_cv = GradientBoostingClassifier(random_state = 101)

GB_cv_scores = cross_validate(GB_cv, X_train, y_train,
                              scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
GB_cv_scores = pd.DataFrame(GB_cv_scores, index = range(1, 11))

GB_cv_scores.mean()[2:]
```
[175] Python

```
param_grid = {"n_estimators":[100, 200, 300],
              "subsample":[0.5, 1],
              "max_features" : [None, 2, 3, 4],
              "learning_rate": [0.001, 0.01, 0.1],
              'max_depth':[3, 4, 5, 6]}
```
[176] Python

```
GB_grid = GradientBoostingClassifier(random_state = 101)
GB_grid_model = GridSearchCV(GB_grid, param_grid, scoring = "f1", verbose = 0, n_jobs = -1).fit(X_train, y_train)
```
[181] Python

...

```
GB_grid_model.best_estimator_
```
[182] Python

```
GB_tuned = GradientBoostingClassifier(learning_rate = 0.01,
                                      max_depth = 6,
                                      n_estimators = 200,
                                      subsample = 0.5,
                                      random_state = 101).fit(X_train, y_train)
```
[184] Python

...

```
y_pred = GB_tuned.predict(X_test)
y_train_pred = GB_tuned.predict(X_train)

GB_tuned_f1 = f1_score(y_test, y_pred)
GB_tuned_acc = accuracy_score(y_test, y_pred)
GB_tuned_recall = recall_score(y_test, y_pred)
GB_tuned_auc = roc_auc_score(y_test, y_pred)
```

```
GB_tuned = GradientBoostingClassifier(learning_rate = 0.01,
                                      max_depth = 6,
                                      n_estimators = 200,
                                      subsample = 0.5,
                                      random_state = 101).fit(X_train, y_train)
```
[184] Python

...

```
y_pred = GB_tuned.predict(X_test)
y_train_pred = GB_tuned.predict(X_train)

GB_tuned_f1 = f1_score(y_test, y_pred)
GB_tuned_acc = accuracy_score(y_test, y_pred)
GB_tuned_recall = recall_score(y_test, y_pred)
GB_tuned_auc = roc_auc_score(y_test, y_pred)
```
[185] Python

```
print("GB_tuned")
print ("------------------")
eval(GB_tuned, X_train, X_test)
```
[186] Python

```
cf_matrix = confusion_matrix(y_test, y_pred)

group_names = ["True Negatives (TN)", "False Positives (FP)\n(Type I Error)", "False Negatives (FN)\n(Type II Error)", "True Positives (TP)"]

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names, group_counts, group_percentages)]

labels = np.asarray(labels).reshape(2, 2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt="", cmap="Blues")
ax.set(xlabel="Predicted Class", ylabel = "Actual Class");
```
[187] Python

```
plt.figure(figsize=(12, 8))
visualizer = ClassificationReport(GB_tuned, support=True)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay
```
[197] Python

```
from yellowbrick.classifier import ClassPredictionError

visualizer = ClassPredictionError(GB_tuned)


visualizer.fit(X_train, y_train)


visualizer.score(X_test, y_test)


visualizer.poof();
```
[198] Python

```
RocCurveDisplay.from_estimator(GB_model, X_test, y_test);
```
[199] Python

```python
GB_Pred = {"Actual": y_test, "GB_Pred":y_pred}
GB_Pred = pd.DataFrame.from_dict(GB_Pred)
GB_Pred.head(20)
```

[285] Python


```python
GB_Pred = {"Actual": y_test, "GB_Pred":y_pred}
GB_Pred = pd.DataFrame.from_dict(GB_Pred)
GB_Pred.head(20)
```

[285] Python


```python
Model_Preds = GB_Pred
Model_Preds.head(1)
```

[286] Python

```python
KNN_model = KNeighborsClassifier(n_neighbors = 5)
KNN_model.fit(X_train, y_train)
y_pred = KNN_model.predict(X_test)
y_train_pred = KNN_model.predict(X_train)

KNN_model_f1 = f1_score(y_test, y_pred)
KNN_model_acc = accuracy_score(y_test, y_pred)
KNN_model_recall = recall_score(y_test, y_pred)
```


```python
Model_Preds = GB_Pred
Model_Preds.head(1)
```

[286] Python


```python
KNN_model = KNeighborsClassifier(n_neighbors = 5)
KNN_model.fit(X_train, y_train)
y_pred = KNN_model.predict(X_test)
y_train_pred = KNN_model.predict(X_train)

KNN_model_f1 = f1_score(y_test, y_pred)
KNN_model_acc = accuracy_score(y_test, y_pred)
KNN_model_recall = recall_score(y_test, y_pred)
KNN_model_auc = roc_auc_score(y_test, y_pred)
```

[287] Python

```python
print("KNN_Model")
print ("----------------")
eval(KNN_model, X_train, X_test)
```

[288] Python

```python
cf_matrix = confusion_matrix(y_test, y_pred)

group_names = ["True Negatives (TN)", "False Positives (FP)\n(Type I Error)", "False Negatives (FN)\n(Type II Error)", "True Positives (TP)"]

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names, group_counts, group_percentages)]

labels = np.asarray(labels).reshape(2, 2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt="", cmap="Blues")
ax.set(xlabel="Predicted Class", ylabel = "Actual Claass");
```

[289] Python

```python
plt.figure(figsize=(12, 8))
visualizer = ClassificationReport(KNN_model, support=True)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
```

```python
from yellowbrick.classifier import ClassPredictionError

visualizer = ClassPredictionError(KNN_model)


visualizer.fit(X_train, y_train)
```

```python
visualizer.poof();
```

[212] Python

```python
KNN_cv = KNeighborsClassifier(n_neighbors = 5)

KNN_cv_scores = cross_validate(KNN_cv, X_train, y_train,
                               scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
KNN_cv_scores = pd.DataFrame(KNN_cv_scores, index = range(1, 11))

KNN_cv_scores.mean()[2:]
```

[213] Python

```python
plt.figure(figsize = (15, 8))
plt.plot(range(1, 30), test_error_rates, color = 'blue', linestyle = '--', marker = 'o',
         markerfacecolor = 'red', markersize = 10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')

plt.hlines(y = 0.04307948860478039, xmin = 0, xmax = 30, colors = 'r', linestyles = "--", label = "K-Value = 2 Line")
plt.hlines(y = 0.04558087826570312, xmin = 0, xmax = 30, colors = 'r', linestyles = "--", label = "K-Value = 4 Line")
plt.hlines(y = 0.055864369093941, xmin = 0, xmax = 30, colors = 'blue', linestyles = "--", label = "Default K-Value = 5 Line")
plt.legend(prop = {"size":14});
```

[] Python

```python
knn5 = KNeighborsClassifier(n_neighbors = 5)

knn5.fit(X_train,y_train)
pred = knn5.predict(X_test)

print('WITH K=5')
print('----------------')
print(confusion_matrix(y_test, pred))
print('----------------')
print(classification_report(y_test, pred))
```

[215] Python

```python
knn2 = KNeighborsClassifier(n_neighbors = 2)

knn2.fit(X_train,y_train)
pred = knn2.predict(X_test)

print('WITH K=2')
print('----------------')
print(confusion_matrix(y_test, pred))
print('----------------')
print(classification_report(y_test, pred))
```

[216] Python

```python
k_values = range(1, 30)
param_grid = {"n_neighbors": k_values, "p": [1, 2], "weights": ['uniform', "distance"]}
```

[221] Python

```python
KNN_grid = KNeighborsClassifier()
KNN_grid_model = GridSearchCV(KNN_grid, param_grid, cv = 10, scoring = 'recall')
KNN_grid_model.fit(X_train, y_train)
```

[222] Python

```python
KNN_grid_model.best_estimator_
```

[223] Python

```
print(colored('\033[1mBest Parameters of GridSearchCV for KNN Model:\033[0m', 'blue'), colored(KNN_grid_model.best_params_, 'cyan'))
print("-----------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of GridSearchCV for KNN Model:\033[0m', 'blue'), colored(KNN_grid_model.best_estimator_, 'cyan'))
```

[224] Python

```
KNN_tuned3 = KNeighborsClassifier(n_neighbors = 3, p = 1)
KNN_tuned3.fit(X_train, y_train)
y_pred = KNN_tuned3.predict(X_test)
y_train_pred = KNN_tuned3.predict(X_train)

KNN_tuned3_f1 = f1_score(y_test, y_pred)
KNN_tuned3_acc = accuracy_score(y_test, y_pred)
KNN_tuned3_recall = recall_score(y_test, y_pred)
KNN_tuned3_auc = roc_auc_score(y_test, y_pred)

print("KNN_tuned (K-3)")
print ("------------------")
eval(KNN_tuned3, X_train, X_test)
train_val(y_train, y_train_pred, y_test, y_pred)
```

[225] Python

```
KNN_tuned1 = KNeighborsClassifier(n_neighbors = 1, p = 1)
KNN_tuned1.fit(X_train, y_train)
y_pred = KNN_tuned1.predict(X_test)
y_train_pred = KNN_tuned1.predict(X_train)

KNN_tuned1_f1 = f1_score(y_test, y_pred)
KNN_tuned1_acc = accuracy_score(y_test, y_pred)
KNN_tuned1_recall = recall_score(y_test, y_pred)
KNN_tuned1_auc = roc_auc_score(y_test, y_pred)

print("KNN_tuned (K-1)")
print ("------------------")
eval(KNN_tuned1, X_train, X_test)
```

[226] Python

```
RocCurveDisplay.from_estimator(KNN_model, X_test, y_test);
```

[228] Python

...

```
KNN_Pred = {"Actual": y_test, "KNN_Pred":y_pred}
KNN_Pred = pd.DataFrame.from_dict(KNN_Pred)
KNN_Pred.head(20)
```

[230] Python

```
KNN_Pred.drop("Actual", axis = 1, inplace = True)
Model_Preds = pd.merge(Model_Preds, KNN_Pred, left_index = True, right_index = True)
Model_Preds.head(20)
```

[231] Python

```
RF_model = RandomForestClassifier(class_weight = "balanced", random_state = 101)
RF_model.fit(X_train, y_train)
y_pred = RF_model.predict(X_test)
y_train_pred = RF_model.predict(X_train)

RF_model_f1 = f1_score(y_test, y_pred)
RF_model_acc = accuracy_score(y_test, y_pred)
RF_model_recall = recall_score(y_test, y_pred)
RF_model_auc = roc_auc_score(y_test, y_pred)
```

[232] Python

```
print("RF_Model")
print ("------------------")
eval(RF_model, X_train, X_test)
```

[233] Python

```
cf_matrix = confusion_matrix(y_test, y_pred)

group_names = ["True Negatives (TN)", "False Positives (FP)\n(Type I Error)", "False Negatives (FN)\n(Type II Error)", "True Positives (TP)"]

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names, group_counts, group_percentages)]

labels = np.asarray(labels).reshape(2, 2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt="", cmap="Blues")
ax.set(xlabel="Predicted Class", ylabel = "Actual Classs");
```

[234] Python

```
plt.figure(figsize=(12, 8))
visualizer = ClassificationReport(RF_model, support=True)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```

```
from yellowbrick.classifier import ClassPredictionError

visualizer = ClassPredictionError(RF_model)


visualizer.fit(X_train, y_train)
```

```
visualizer.score(X_test, y_test)


visualizer.poof();
```

[236] Python

```
RF_feature_imp = pd.DataFrame(index=X.columns, data = RF_model.feature_importances_, columns = ['Importance']).sort_values("Importance", ascending = False)
RF_feature_imp
```

[237] Python

```
plt.figure(figsize = (12,6))
sns.barplot(data = RF_feature_imp.sort_values('Importance', ascending = False), x = RF_feature_imp.sort_values('Importance', ascending = False).index, y = 'Importance')
plt.xticks(rotation = 75);
```

[238] Python

```
RF_cv = RandomForestClassifier(class_weight = "balanced", random_state = 101)

RF_cv_scores = cross_validate(RF_cv, X_train, y_train,
                              scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
RF_cv_scores = pd.DataFrame(RF_cv_scores, index = range(1, 11))

RF_cv_scores.mean()[2:]
```

[239] Python

```
RF_cv = RandomForestClassifier(class_weight = "balanced", random_state = 101)

RF_cv_scores = cross_validate(RF_cv, X_train, y_train,
                              scoring = ['accuracy', 'precision','recall', 'f1', 'roc_auc'], cv = 10)
RF_cv_scores = pd.DataFrame(RF_cv_scores, index = range(1, 11))

RF_cv_scores.mean()[2:]
```

[240] Python

```
param_grid = {'n_estimators' : [50, 100, 300],
              'max_features' : [2, 3, 4],
              'max_depth' : [3, 5, 7, 9],
              'min_samples_split' : [2, 5, 8]}
```
[240] Python

```
RF_grid = RandomForestClassifier(class_weight = 'balanced', random_state = 101)
RF_grid_model = GridSearchCV(estimator = RF_grid,
                             param_grid = param_grid,
                             scoring = "recall",
                             n_jobs = -1, verbose = 0)
RF_grid_model.fit(X_train, y_train)
```
[241] Python

```
RF_grid_model.best_estimator_
```
[242] Python

```
RF_tuned = RandomForestClassifier(class_weight = 'balanced',
                                  max_depth = 3,
                                  max_features = 4,
                                  n_estimators = 300,
                                  random_state = 101).fit(X_train, y_train)
```
[244] Python

```
y_pred = RF_tuned.predict(X_test)
y_train_pred = RF_tuned.predict(X_train)

RF_tuned_f1 = f1_score(y_test, y_pred)
RF_tuned_acc = accuracy_score(y_test, y_pred)
RF_tuned_recall = recall_score(y_test, y_pred)
RF_tuned_auc = roc_auc_score(y_test, y_pred)
```
[245] Python

```
print("RF_tuned")
print ("-----------------")
eval(RF_tuned, X_train, X_test)
```
[246] Python

```
cf_matrix = confusion_matrix(y_test, y_pred)

group_names = ["True Negatives (TN)", "False Positives (FP)\n(Type I Error)", "False Negatives (FN)\n(Type II Error)", "True Positives (TP)"]

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names, group_counts, group_percentages)]

labels = np.asarray(labels).reshape(2, 2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt="", cmap="Blues")
ax.set(xlabel="Predicted Class", ylabel = "Actual Class");
```
[247] Python

```
plt.figure(figsize=(12, 8))
visualizer = ClassificationReport(RF_tuned, support=True)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```

```
from yellowbrick.classifier import ClassPredictionError

visualizer = ClassPredictionError(RF_tuned)


visualizer.fit(X_train, y_train)
```

```
from yellowbrick.classifier import ClassPredictionError

visualizer = ClassPredictionError(RF_tuned)


visualizer.fit(X_train, y_train)


visualizer.score(X_test, y_test)


visualizer.poof();
```
[249] Python

```
RocCurveDisplay.from_estimator(RF_model, X_test, y_test);
```
[250] Python

```
RF_Pred = {"Actual": y_test, "RF_Pred":y_pred}
RF_Pred = pd.DataFrame.from_dict(RF_Pred)
RF_Pred.head(20)
```
[251] Python

```
Model_Preds.sample(10)
```
[253] Python

```
compare = pd.DataFrame({"Model": ["GB_model", "GB_tuned", "KNN_Model", "KNN_tuned3", "RF_model", "RF_tuned"],

                        "F1_Score": [GB_model_f1, GB_tuned_f1, KNN_model_f1, KNN_tuned3_f1, RF_model_f1, RF_tuned_f1],

                        "Accuracy_Score": [GB_model_acc, GB_tuned_acc, KNN_model_acc, KNN_tuned3_acc, RF_model_acc,
                                           RF_tuned_acc],

                        "Recall_Score": [GB_model_recall, GB_tuned_recall, KNN_model_recall, KNN_tuned3_recall, RF_model_recall,
                                         RF_tuned_recall],

                        "ROC_AUC_Score": [GB_model_auc, GB_tuned_auc, KNN_model_auc, KNN_tuned3_auc, RF_model_auc,
                                          RF_tuned_auc]})

compare = compare.sort_values(by="Recall_Score", ascending=True)
fig = px.bar(compare, x = "Recall_Score", y = "Model", title = "Recall_Score")
fig.show()

compare = compare.sort_values(by="F1_Score", ascending=True)
fig = px.bar(compare, x = "F1_Score", y = "Model", title = "F1_Score")
fig.show()

compare = compare.sort_values(by="Accuracy_Score", ascending=True)
fig = px.bar(compare, x = "Accuracy_Score", y = "Model", title = "Accuracy_Score")
fig.show()

compare = compare.sort_values(by="ROC_AUC_Score", ascending=True)
fig = px.bar(compare, x = "ROC_AUC_Score", y = "Model", title = "ROC_AUC_Scor")
fig.show()
```

2. Testing

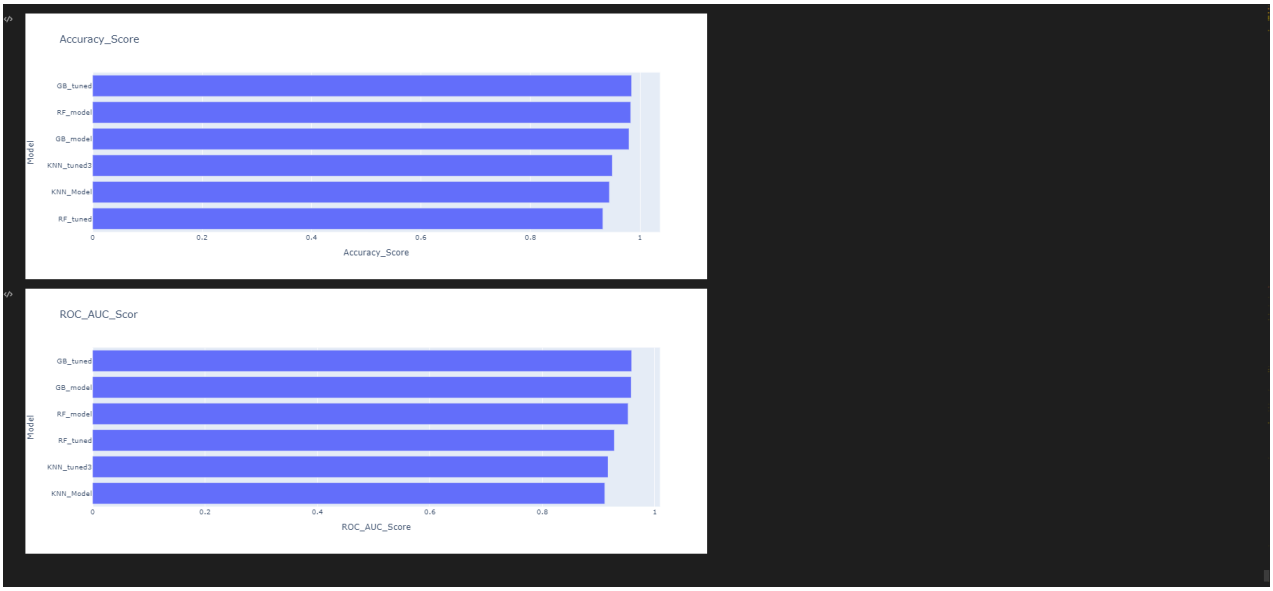
The goal of system testing, which consists of a variety of tests, is to fully indoctrinate the computer-based system. Even though each test has a specific goal, they are all created to check that every system component has been properly integrated and is carrying out its designated functions.To make sure the product performs exactly as intended, testing procedures are frequently used. The following goals are to be attained during the trial phase:

- To confirm the project's level of excellence.
- To locate and correct any lingering errors from previous steps.
- To guarantee the system's dependability in functioning
- To verify that the program is a viable fix for the initial problem.

RESULTS


Recall_Score
GB_model
RF_tuned
GB_tuned
RF_model
KNN_tuned3
KNN_Model
Model
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Recall_Score

F1_Score
GB_tuned
RF_model
GB_model
KNN_tuned3
KNN_Model
RF_tuned
Model
0
0.2
0.4
0.6
0.8
1
F1_Score


Accuracy_Score
GB_tuned
RF_model
GB_model
KNN_tuned3
KNN_Model
RF_tuned
Model
0
0.2
0.4
0.6
0.8
1
Accuracy_Score

ROC_AUC_Scor
GB_tuned
GB_model
RF_model
RF_tuned
KNN_tuned3
KNN_Model
Model
0
0.2
0.4
0.6
0.8
1
ROC_AUC_Score

CONCLUSION

An employee who works for a firm is a significant resource. If one of them leaves the firm unexpectedly, it could have a big effect and cost the particular company a lot of money. Additionally, the newly employed employee needs some time to make the particular company cost-effective in addition to the time and money needed. Based on analytical data that is currently accessible, this model will help estimate the pace at which people leave their jobs and will use a variety of machine learning algorithms to reduce prediction error.So basing on the accuracy of the prediction rate the HR of an organization can decide the peoples intake and provide good valuable employees to the company or organization.

REFERENCES

1. Base Paper
https://www.igi-global.com/article/intelligent-employee-retention-system-for-attrition-rate-analysis-and-churn-prediction/273145
2.Churn Prediction
https://github.com/Tarun-D/Bank-Customer-Churn-Predection

3.https://github.com/izkabrhm/Human-Resources-Employee-Churn-Prediction/blob/main/Employee_Churn_Prediction_Analysis.ipynb

4.https://github.com/zunicd/Bank-Churn-Prediction/blob/master/Bank%20Customer%20Churn%20Prediction%20-%20Summary.ipynb

5.https://github.com/wenkangwei/Bank-Churn-Prediction/blob/main/Bank_Customer_Churn_Prediction.ipynb