Employee attrition prediction model using Machine Learning

Gautam Varma Datla

Siddhardha Varma Vegesna

Dhushyanth Polkampalli

(gvd6@njit.edu , sv92@njit.edu , dp822@njit.edu)

Department of Computer Science

New Jersey Institute of technology (NJIT)

Newark,New Jersey - 07102

Abstract

Employee attrition refers to the voluntary or involuntary departure of employees from a company. It is one of the most pressing issues confronting most of the organizations around the world today because of the significant impact it could have on the organization’s productivity and money . As a result, firms must start devising extremely targeted techniques to predict and avert such a scenario, and it is such situations that machine learning comes in handy (to model the data and offer insights).This accompanied with the growing interest in machine learning in the field of business decision making necessitates that one should dwell deeper into this domain.So, in this study, we first train our data set using Logistic Regression , SupportVector Machine (SVM), Naïve Bayes, and random forest models, and then through a comprehensive evaluation process and hyper parameter tuning , we choose the best model based on the efficacy of these models in predicting employee attrition. We used the confusion matrix and ROC (receiver operating characteristic curve) for predicting the accuracy of these models.

Introduction

1.1 Problem Statement

While attrition is inevitable in any organization, limiting it and being prepared for situations that cannot be avoided would significantly enhance the operations of most organizations. We can do so by developing a certain "at risk" category of employees.With a plethora of data on employee demographics provided to us by the HR department, our main objective as researchers would be to analyze such data using classification techniques and provide the necessary insights to perform a segmentation of employees who are prone to attrition . Once the class of employees prone to attrition is figured out the company can take necessary steps to avert such situations .So this study mainly aims at predicting the factors (Input variables / Features) that have a huge impact on attrition and also build a robust classification model to accurately predict the employees prone to attrition.

1.2 Project summary and methodology

In this study, we first load the data and perform exploratory data analysis to understand the distributions of all the features and their relationship with our target variable (Attrition) using correlation plots, histograms, bar graphs,etc .Once we attain a thorough knowledge of our feature space, we begin pre-processing the data for modeling it. In this step along with feature selection and outlier removal (Using Box plots) we also used SMOTE technique (As our data set was imbalanced) and principal component analysis (On normalized data to improve performance). We then used Logistic regression , Support Vector Machine (SVM) , Decision Trees and Random Forests to build classification models for attrition prediction. For each of these models, Grid Search CV was used to tune hyperparameters and the f1.Score , Accuracy , R.O.C curve and Confusion matrix were used to evaluate the performance of these models.

1. Understand and identify the data (Numpy,Pandas)
2. Performing Exploratory Data Analysis on the data (univariate and bivariate analysis)
3. Visualization of data to understand the relationship (Matplotlib and Seaborn)
4. Data Pre-processing (Feature selection , Outlier removal , Principal component analysis (P.C.A) , SMOTE (For imbalanced data set) , Normalization)
5. Split the data set into two parts (Train data and Test data)
6. Choosing and training the chosen machine learning model on train data set (Scikit Learn)
7. Tuning the hyper parameters of each model to improve the prediction accuracy(Grid search)
8. Using the Test set on the best model to predict attrition
9. Evaluating the performance of all the models and choosing the best attrition prediction model.

1. Description of Data set

The dataset for this project was published by the Human Resources(HR) Department at IBM. It is a fictional dataset created by data scientists at IBM and has 1470 samples of employee data. Table 1 contains information about the name, type, and description of each feature in the data set.

2.1 Target Variable

The target feature of this dataset is the attrition column.It is a binary categorical variable taking values 0 and 1. Here category "0" for an employee means that there is no attrition and category "1" means that there is a chance of employee attrition.

2.2 Description of the classes in categorical features

•Environment Satisfaction(1:Low 2:Medium 3:High 4:Very High)

•Job Involvement (1:Low 2:Medium 3:High 4:Very High)

•Job satisfaction (1:Low 2:Medium 3:High 4:Very High)

•Relationship Status (1:Low 2:Medium 3:High 4:Very High)

•Performance Rating (1:Low 2:Good 3:Excellent 4:Outstanding)

•Work life Balance (1:Bad 2:Good 3:Better 4:Best)

•Education (1:Below College 2:College 3:Bachelor 4:Master 5:Doctor)

•Business Travel (1:No travel 2:Travel Frequently 3:Travel Rarely)

•Department (1:Human Resources(HR) 2: RD 3:Sales)

•Gender (1: Male 2: Female)

•Marital status (1:Divorced 2:Married 3:Single)

•Over 18 (1:Yes 2:No)

•Overtime (1:No 2:yes)

•Education field (1:HR 2:Life Sciences 3:Marketing 4:Medical Sciences 5:Others 6:Technical)

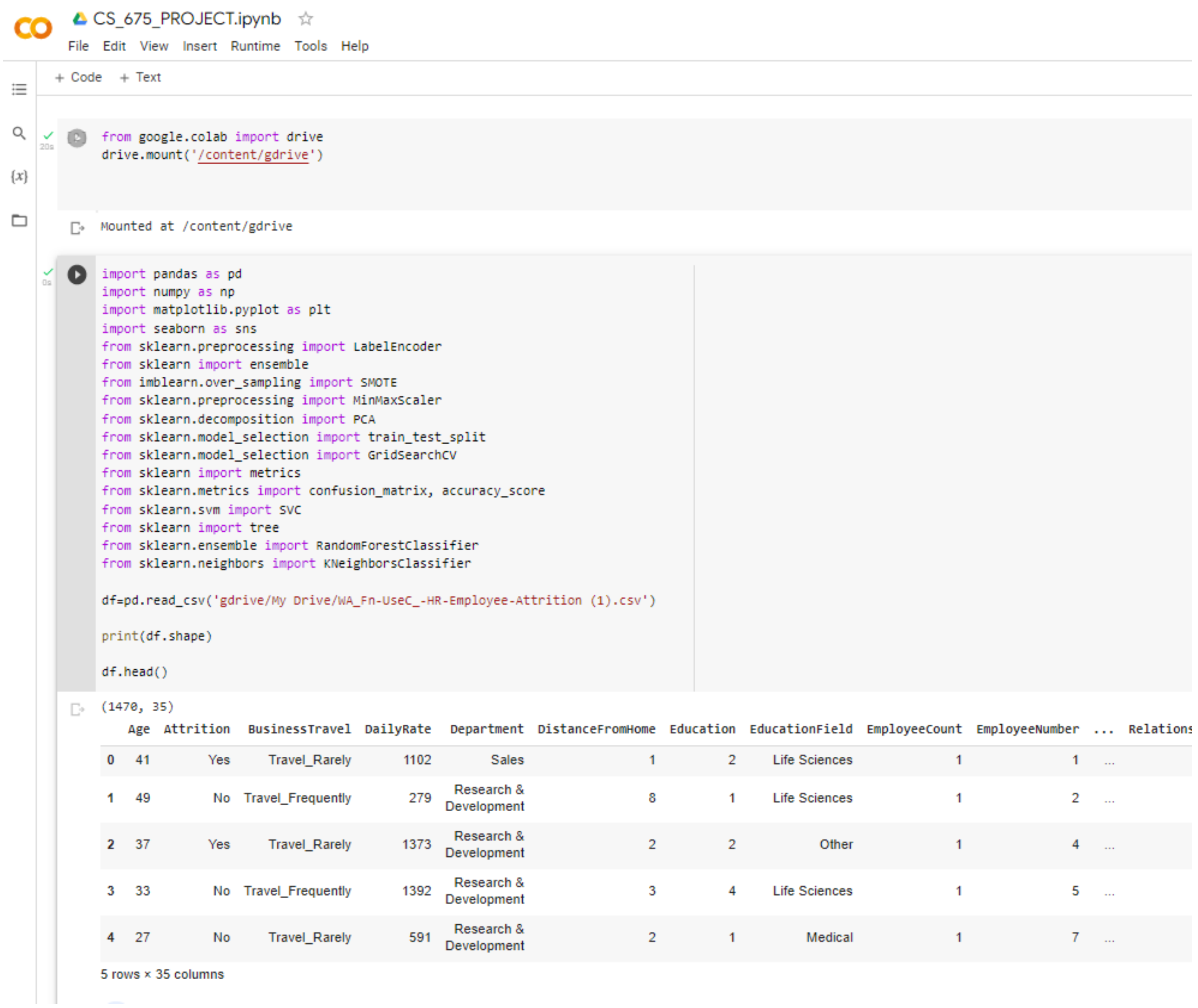
•Job Role (1:HC REP 2:HR 3:Lab Technician 4:Manager 5:Managing Director 6:Research Director 7:Research Scientist 8:Sales Executive 9:Sales Representative)

Table 1: Features of IBM dataset

Name	Description	Type of variable
Age	Age of the employee	Quantitative
Attrition	Will an employee leave the company	Categorical
Business Travel	How often does the employee travel	Categorical
Daily Rate	Daily rate of the employee	Quantitative
Department	Which department does the employee belong to	Categorical
Distance from Home	How far does the employee life from his workplace	Quantitative
Education	Education Level of the employee	Categorical
Education field	In which field was the employee educated in	Categorical
Employee count	Count of employee, it is a redundant column	Quantitative
Employee number	ID of employee	Quantitative
Environment Satisfaction	How satisfied is the employee with environment	Categorical
Gender	Gender of the employee	Categorical
Hourly Rate	Hourly salary of the employee	Quantitative
Job Involvement	Involvement with employee's job	Categorical
Job level	Level of employee's job in the firm	Categorical
Job Role	Job role of the employee	Categorical
Job Satisfaction	How satisfied is the employee	Categorical
Marital Status	Marital status of the employee	Categorical
Monthly Income	Monthly income of the employee	Quantitative
Monthly Rate	Monthly rate of the employee	Quantitative
Num companies Worked	Number of companies employee worked earlier	Quantitative
Over 18	If the employee is over 18	Categorical
Overtime	Overtime of the employee	Quantitative
Percent Salary Hike	Percentage of hike in employee's salary	Quantitative
Performance Rating	Performance rating of the employee	Categorical
Relations Satisfaction	Employee's Relations satisfaction	Categorical
Standard Hours	work of the employee	Quantitative
Stock options level	Stock option level of the employee	Categorical
Total working years	Number of years employee worked for	Quantitative
Training time last year	Train time for the employee	Quantitative
Work life balance	Work life of the employee	Categorical
Years at company	Number of year employee worked in this company	Quantitative
Years in current role	Employee years in current role	Quantitative
Years since last promotion	Years since last promotion for the employee	Quantitative
Years with current manager	Employee years with current manager	Quantitative

2. Experiments and methodology

The entire code for this project was written using python on google colab. Using the .csv published by IBM we first loaded it onto our google drive then accessed it using pandas.read_csv() . Other important libraries used in this project include numpy , pandas for making changes to data ,imblearn for the oversampling minority class (SMOTE) , matplotlib and seaborn for data visualization , scikit learn for normalizing the data (Min-Max Scalar) , hyper parameter tuning (GridSearchCV and RandomSearchCV) , model building (KNeighborsClassifier , SVC , tree , Random Forest classifier) and model evaluation (Metrics , Confusion Matrix , accuracy_score)



```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn import ensemble
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.svm import SVC
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

df=pd.read_csv('gdrive/My Drive/WA_Fn-UseC_-HR-Employee-Attrition (1).csv')

print(df.shape)

df.head()
```

```
(1470, 35)
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	Relations
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	...	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	...	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	...	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	...	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	...	

5 rows × 35 columns

Fig. Project setup

EDA (Exploratory data analysis)

Once we have loaded our dataset and figured out the variable types, we next begin our analysis on each feature. Here we try to explore the distribution of each feature and relationship between each feature and our target variable (Attrition) . For categorical variables we used matpltlib's bar graph and seaborn's countplot to visualize and analyze the distribution.

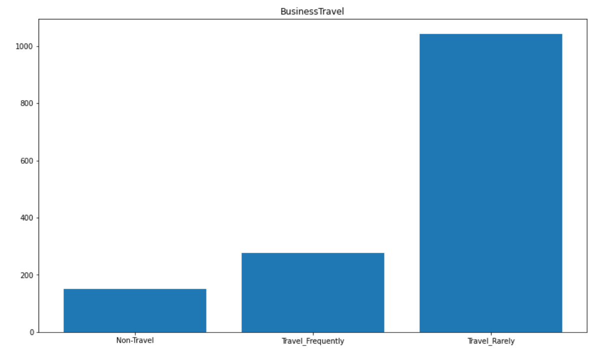


Fig1a. Business travel distribution

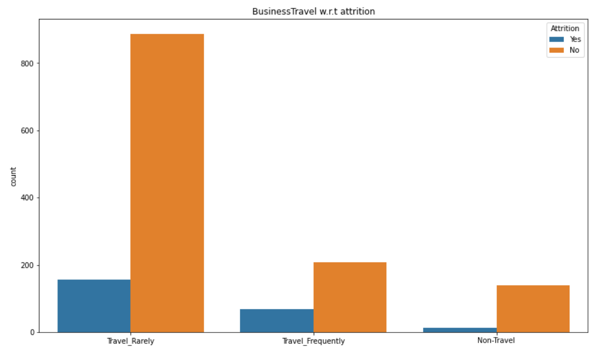


Fig1b. Business travel distribution w.r.t attrition

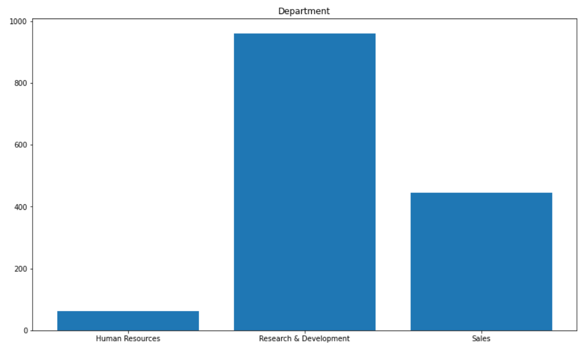


Fig2a. Department distribution

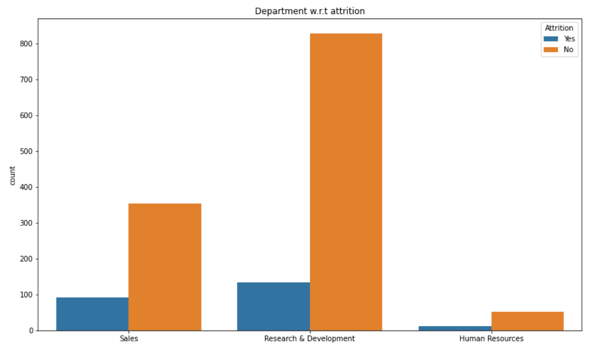


Fig2b. Department distribution w.r.t attrition

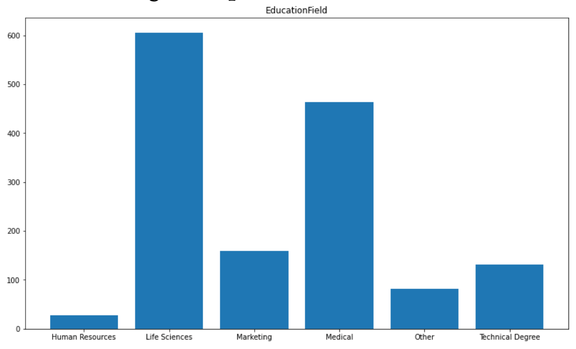


Fig3a. Education field

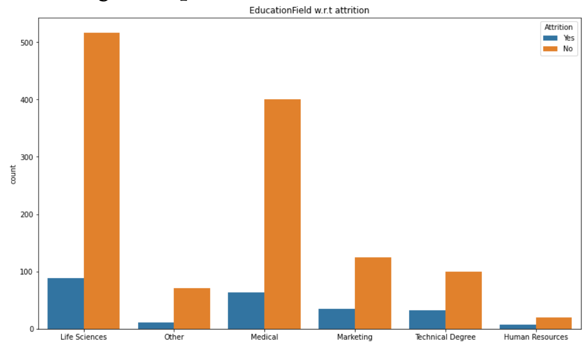


Fig3b. Education Field w.r.t attrition

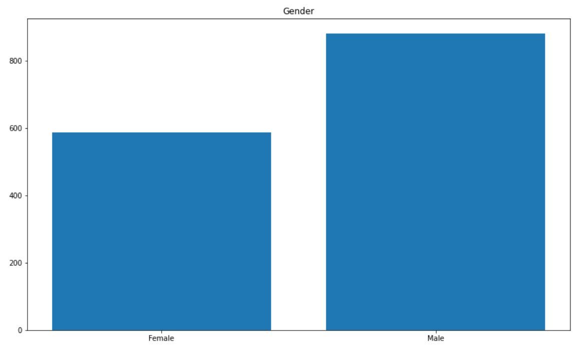


Fig4a. Gender

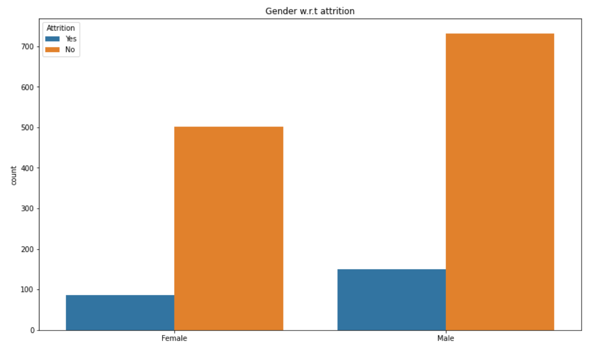


Fig 4b. Gender w.r.t attrition

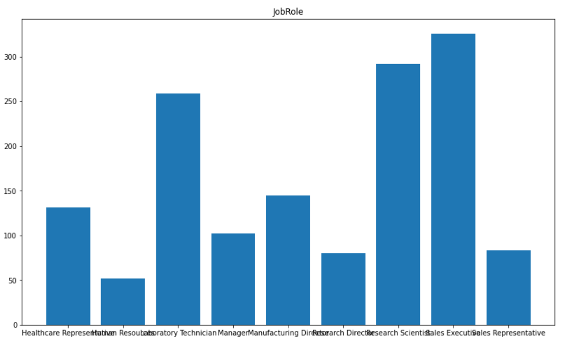


Fig5a. Job role

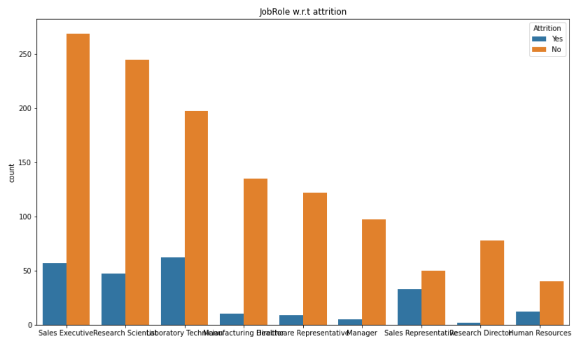


Fig5b. Job role w.r.t attrition

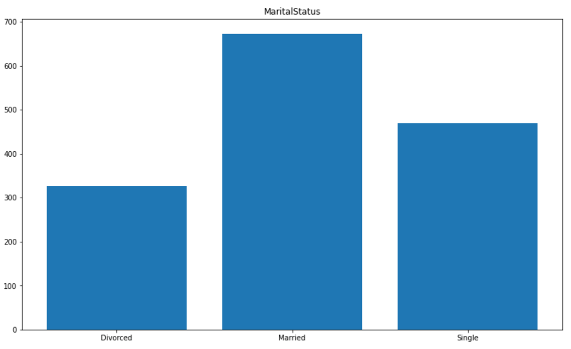


Fig6a. Marital Status

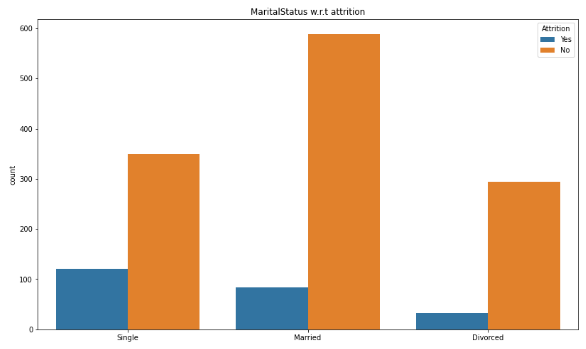


Fig6b. Marital status w.r.t attrition

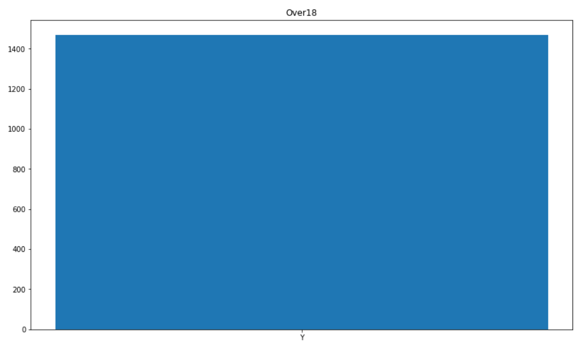


Fig7a. Over 18 (Age)

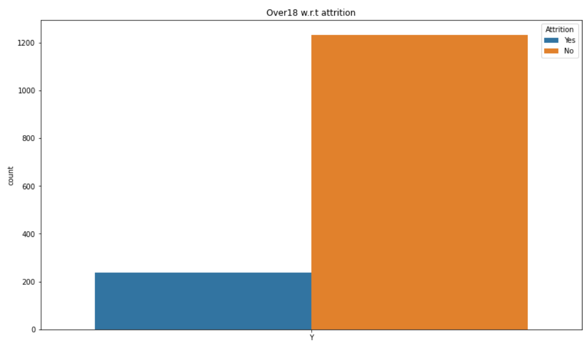


Fig7b. Over 18 w.r.t attrition

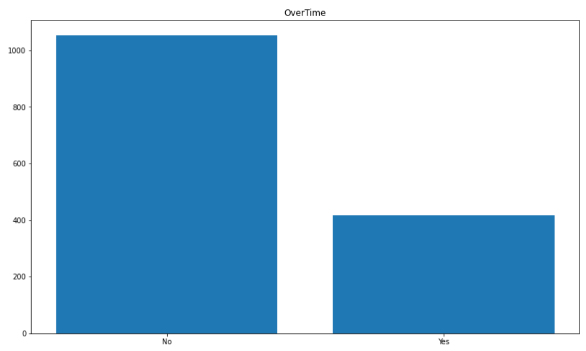


Fig8a. Overtime

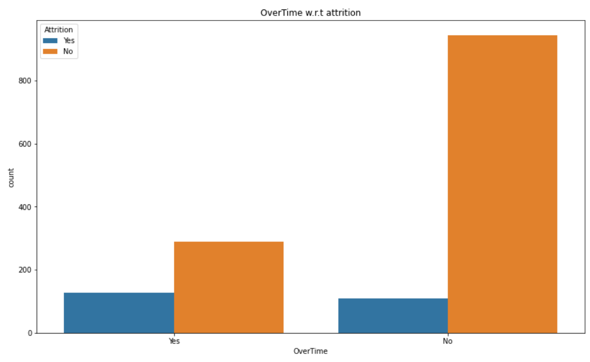


Fig8b. Overtime w.r.t attrition

Feature Name	Most frequent category	Category causing most attrition
Business Travel	Travel Rarely	Travel Rarely
Job role	Sales Executive	Laboratory Technician
Gender	Male	Male
Education Field	Life sciences	Life sciences
Department	Research and Development	Research and Development
Marital Status	Married	Single
Overtime	No	Both categories equally cause attrition

The above mentioned table briefly summarizes our categorical feature analysis, which includes the most frequently occurring category of each categorical feature as well as the category in each feature that causes the most attrition.

Also, from Fig 7 We can clearly see that the feature has only one unique value (Yes) So, this particular feature can be dropped from the data set. The main intuition behind this is that since all the employees are above 18 years of age, we will not be capturing any new useful information using this feature. So the corresponding column will be dropped later on while pre-processing our data for building ML models.

Now , For numerical features and categorical features encoded as numbers we used the histogram plots to capture the distribution of each feature. To visualize the distribution we used seaborn's distplot() and histplot().

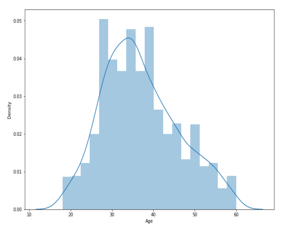


Figure 9a. Age distribution

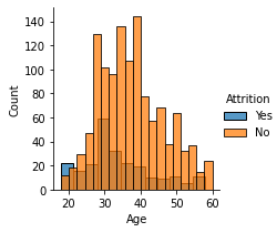


Fig9b. Age w.r.t attrition

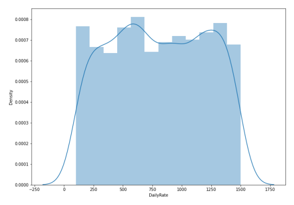


Fig10a.Daily Rate

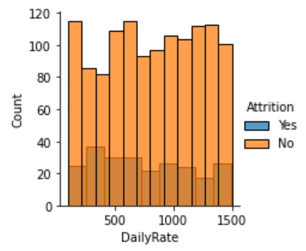


Fig 10b. Daily rate w.r.t attrition

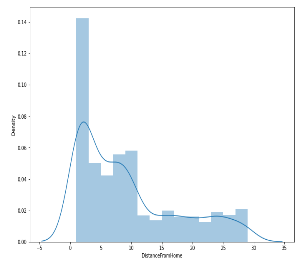


Fig11. Distance from home

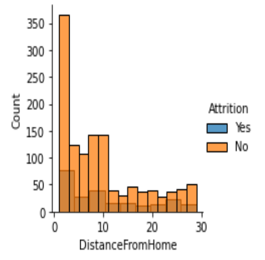


Fig11b. 11a. w.r.t attrition

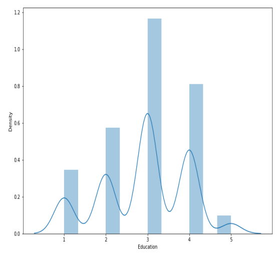


Fig12a. Education

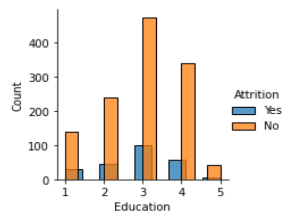


Fig12b.Education w.r.t attrition

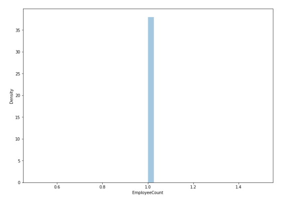


Fig13a. Employee count

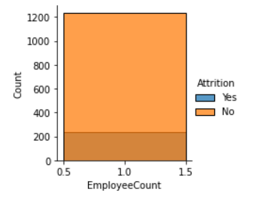


Fig13b. 13aw.r.t attrition

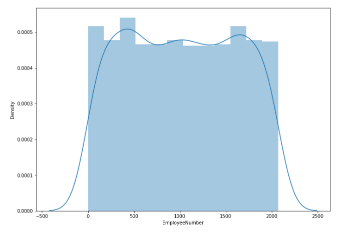


Fig14a. Employee number

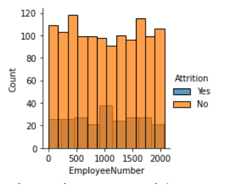


Fig14b. 14a w.r.t. attrition

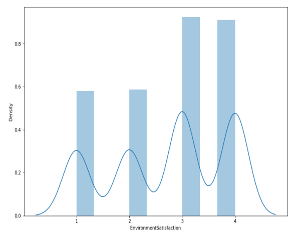


Fig15a. Env satisfaction

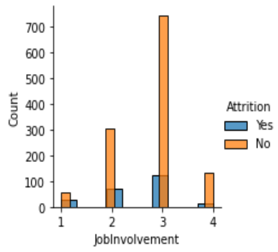


Fig15b. Job involvement w.r.t attrition

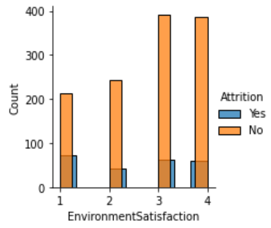


Fig16a. Hourly rate

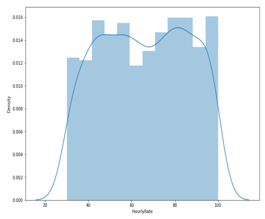


Fig16b.Hourly rate w.r.t. attrition

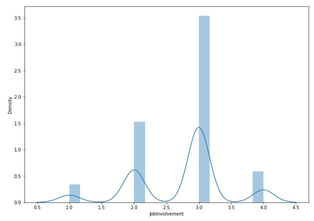


Fig 17a. Job inv

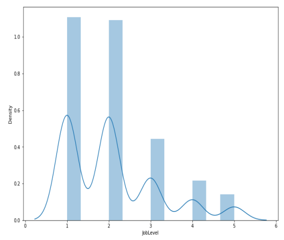


Fig 18a. Job level

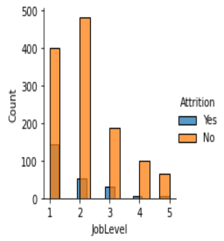


Fig 18b. Job level w.r.t attrition

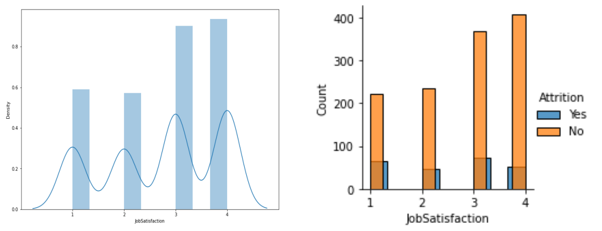


Fig 19a. Job satisfaction and job satisfaction wrt attrition

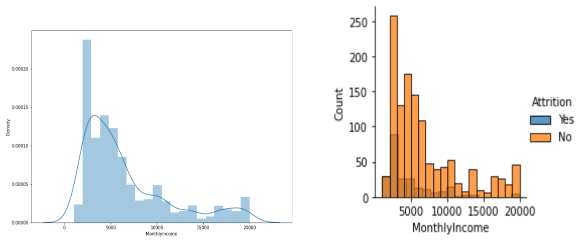


Fig19b. Monthly income dist and income w.r.t attrition

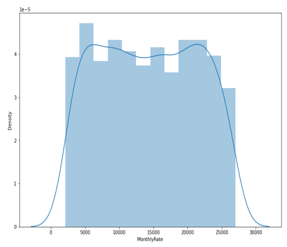


Fig20a. Monthly Rate

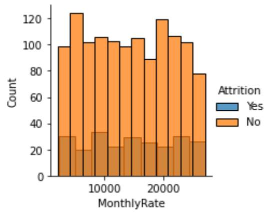


Fig 20b. Monthly rate w.r.t attrition

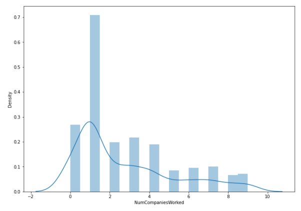


Fig 21a. Num companies worked

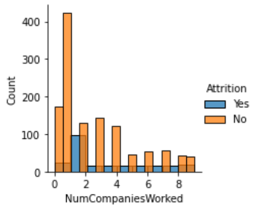


Fig 21b. 21a. w.r.t attrition.

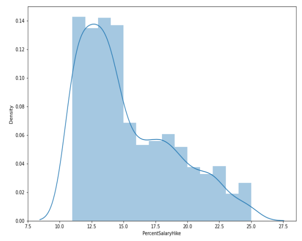


Fig22a. Percentage Salary Hike

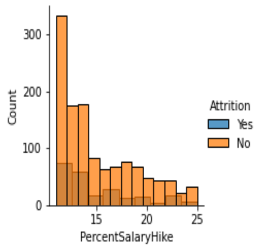


Fig 22b. Percentage salary hike w.r. Attrition

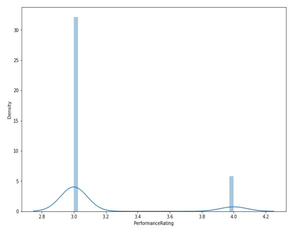


Fig 23a. Performance rating

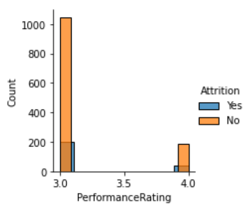
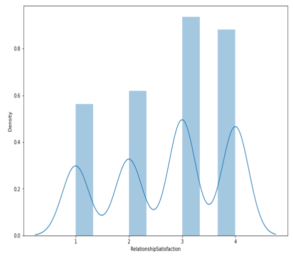
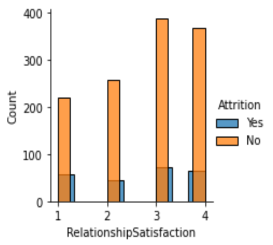
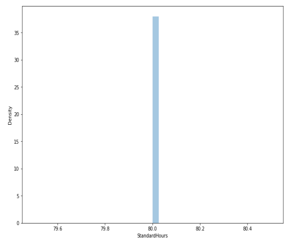


Fig23b. Performance rating w.r.t attrition







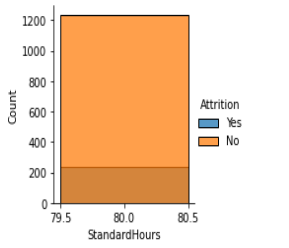
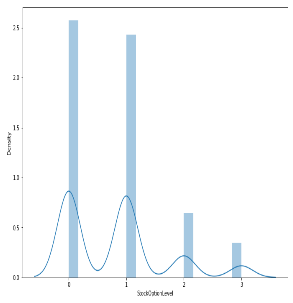
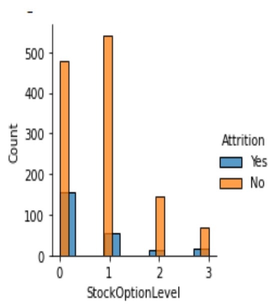
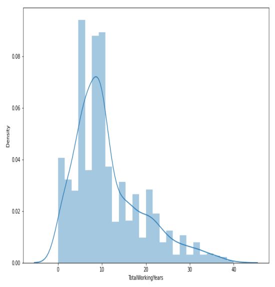


Fig 24a. Relationship satisfaction Fig24b. Relationship satisfaction w.r.t. Attrition Fig 25a. Standard hours Fig 25b. Standard hours wrt attrition







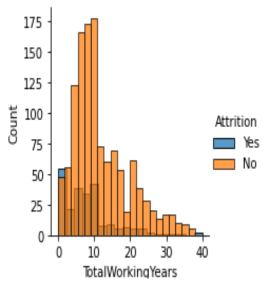
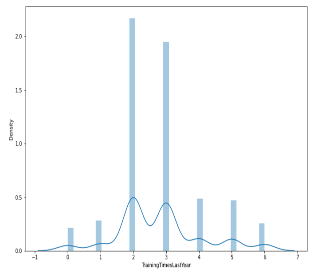
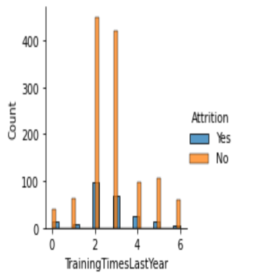
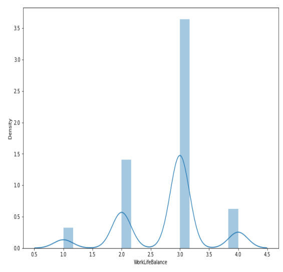


Fig26. Stock option level Fig26b. 26a. w.r.t attrition Fig27a. Total working years Fig27b. 27a. w.r.t attrition







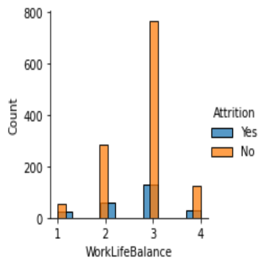
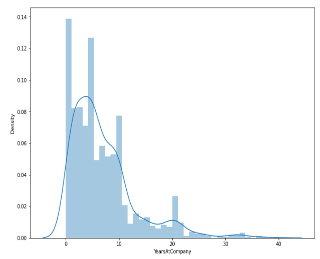
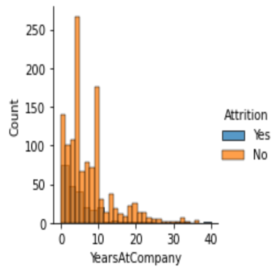
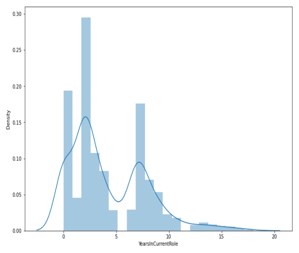


Fig28a. Training time Fig 28b. Training time w.r.t attrition Fig29a. Work life Balance Fig 29b. 29a. w.r.t. Attrition







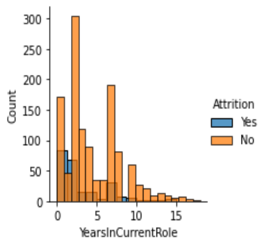


Fig 30a. Years at company Fig 30b. 30a. w.r.t attrition Fig31a. Years in current role Fig 31b. 31a. w.r.t attrition

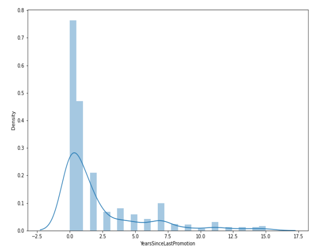


Fig32a. Years since last promotion

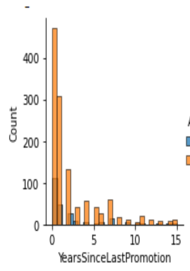


Fig32b. 32a w.r.t attrition

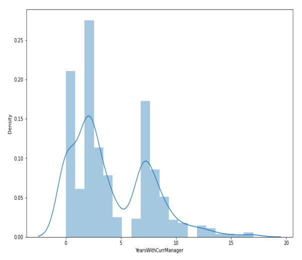


Fig 33a. Years with current manager

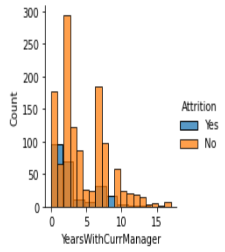


Fig33b. w.r.t attrition

Correlation Plots :

Having garnered a comprehensive understanding of all the variables (Features) and their corresponding distributions , we can now analyze the relationship between features and the relation that each feature has w.r.t attrition using the correlation plots as shown below.
When we have dependent variables, changes to one variable can cause a swing of our output during model fitting , which is highly undesirable. So identifying such correlations and dealing with them during data preprocessing plays a pivotal role in improving the accuracy of our classification model

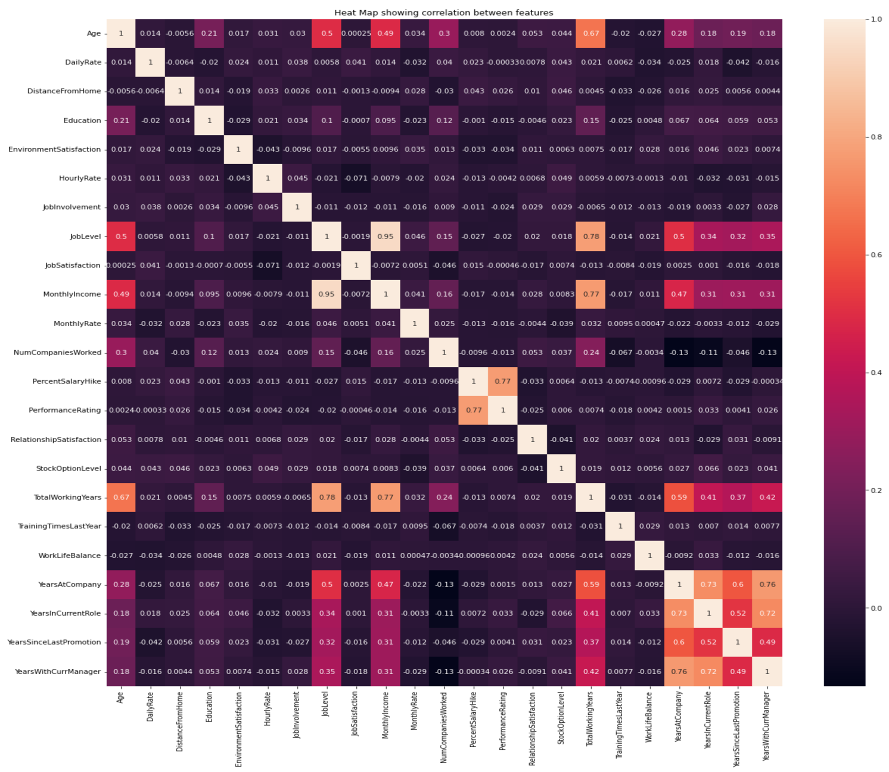


Fig 34. Correlation heat map

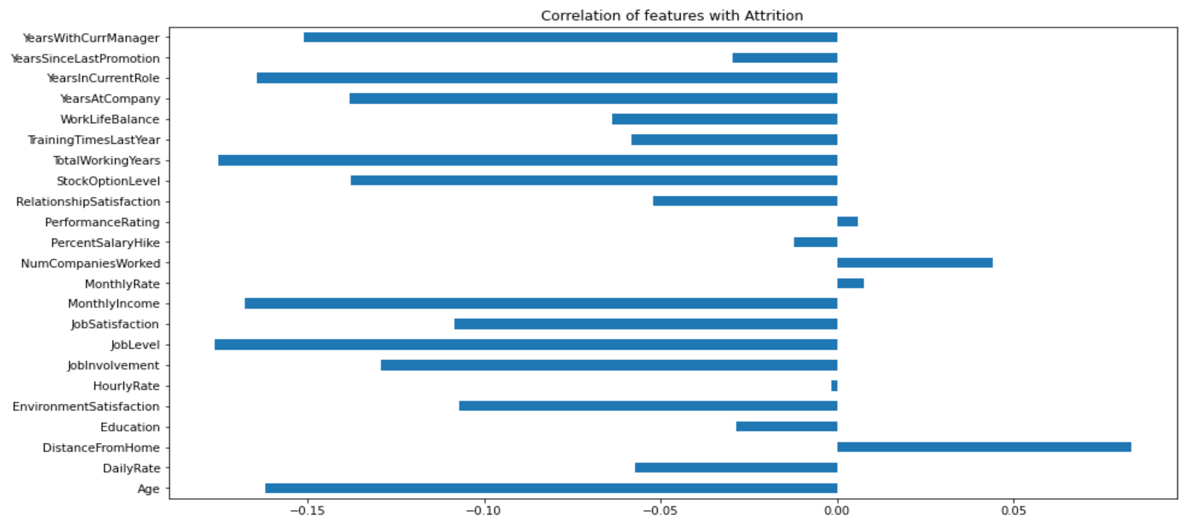


Fig 35. Correlation of features w.r.t attrition

Outlier Identification :

Outliers we have in our data could either be natural outliers , data entry errors or measurement errors. All these outliers increase the variance in our data and reduce the power of statistical tests , moreover most of the ML algorithms are sensitive to outliers , so it is desirable that we remove such outliers before we start building our ML models. Detection and removal of outliers can be done by visualizing the box plot distribution of each feature.

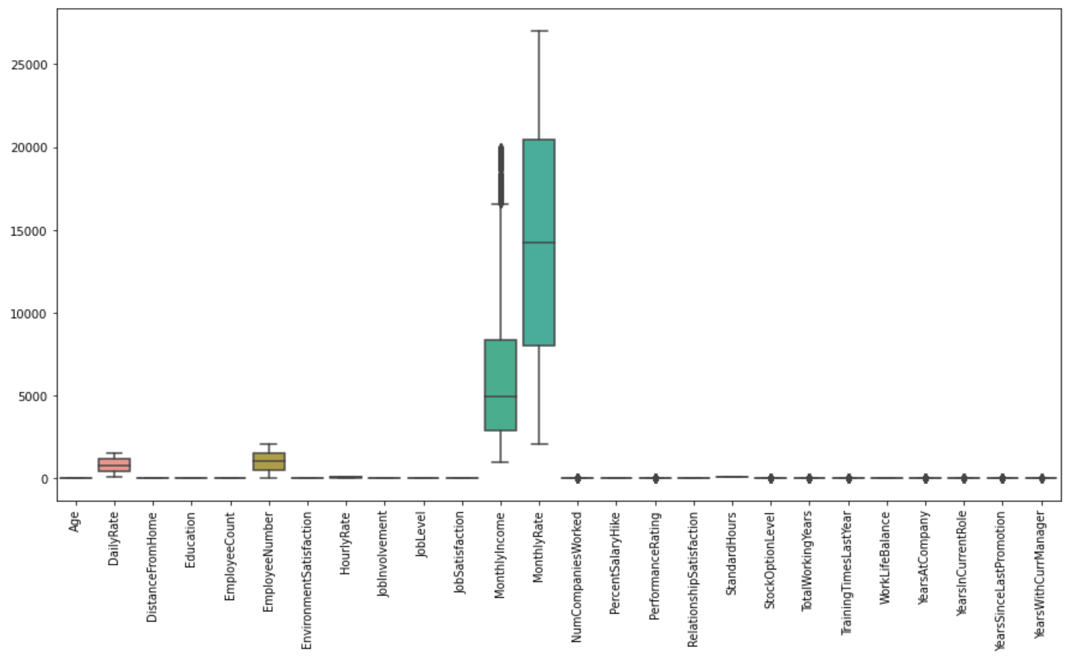


Fig 36.Box plots of features

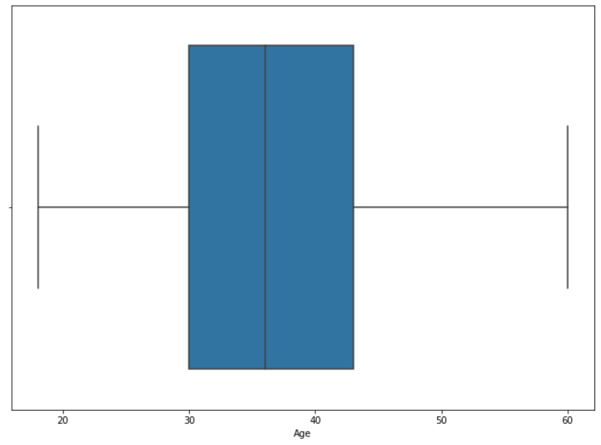


Fig37 . Age box plot

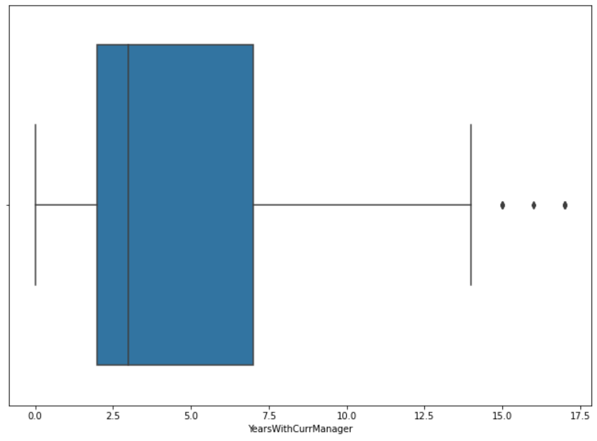


Fig 38. Years with current manager box plot

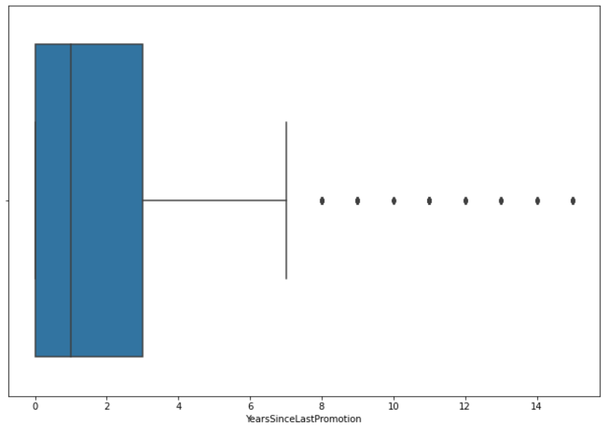


Fig39. Years since last promotion

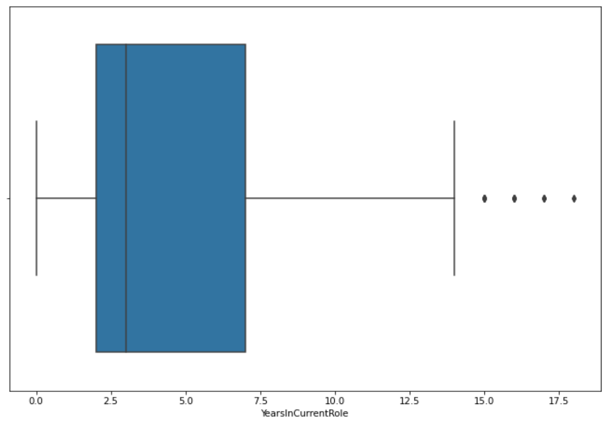


Fig40. Years in current role

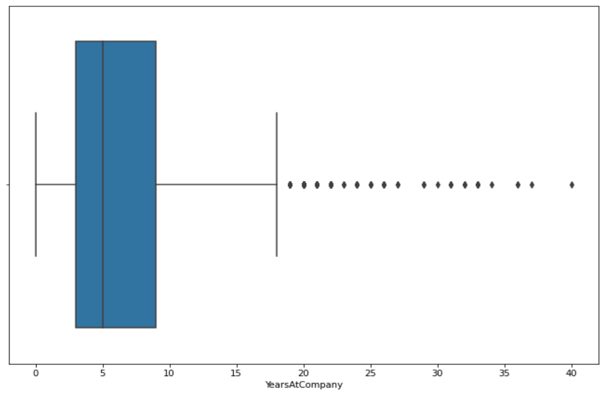


Fig41. Years at company box plot

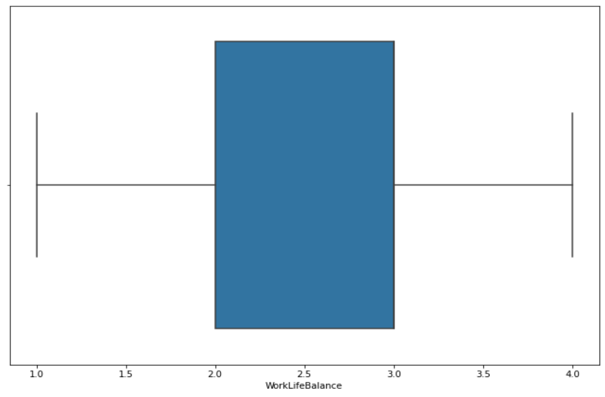


Fig42. Work life balance box plot

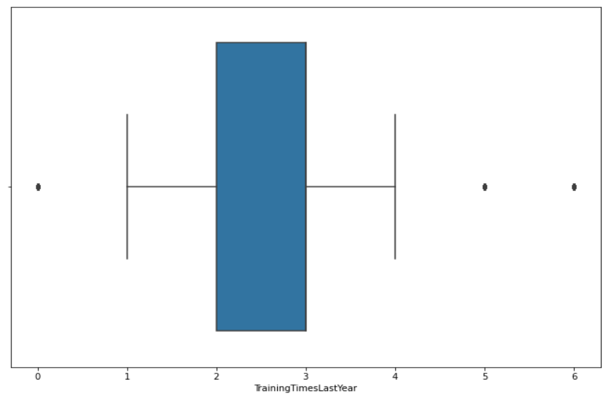


Fig43. Training times last year

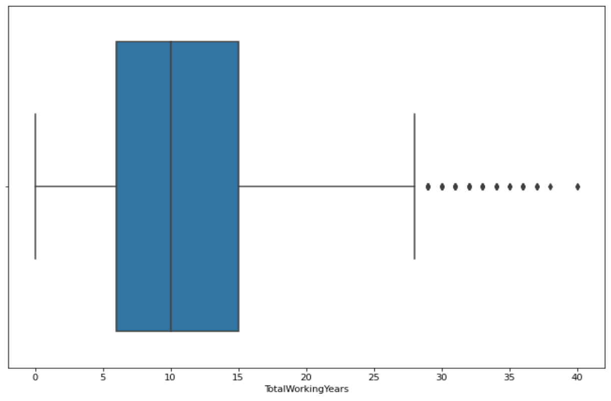


Fig44. Total working years

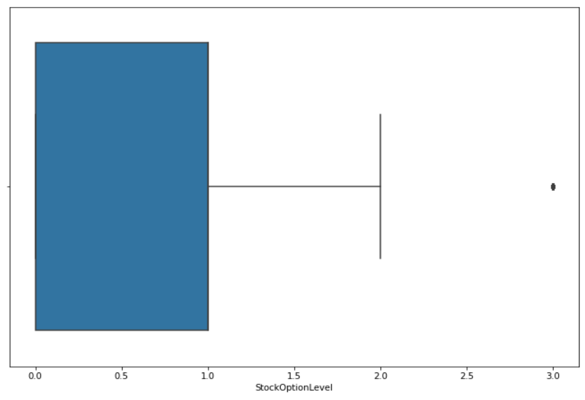


Fig45. Stock Option Level Box plot

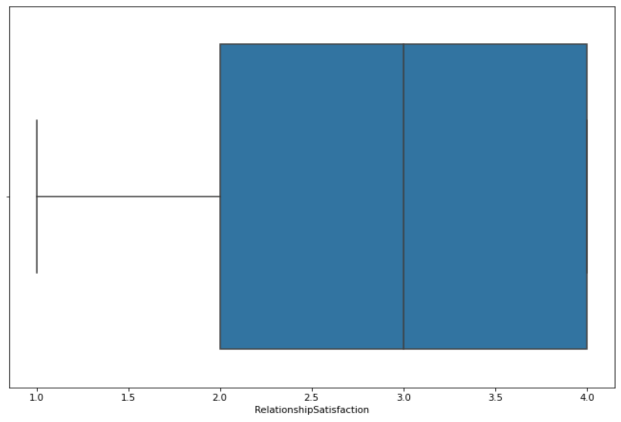


Fig46. Relationship satisfaction box plot

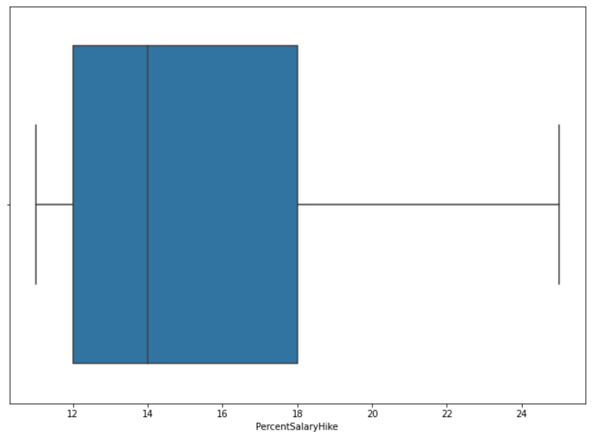


Fig47. Percentage Salary Hike box plot

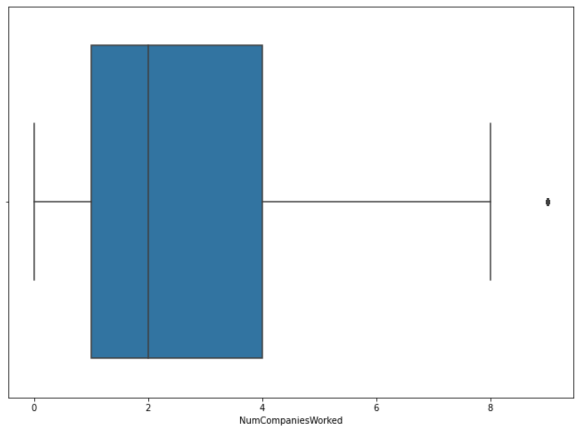


Fig48. Number of companies worked box plot

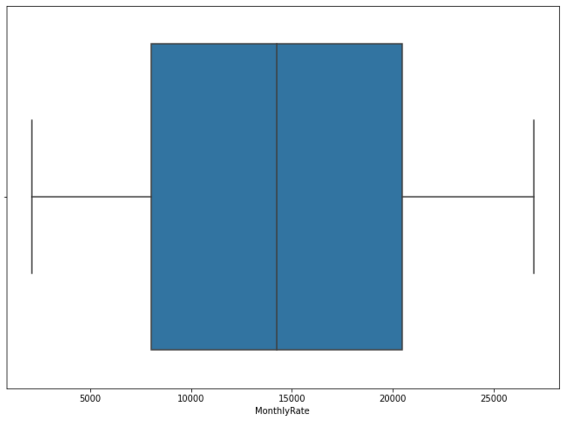


Fig 49. Monthly Rate

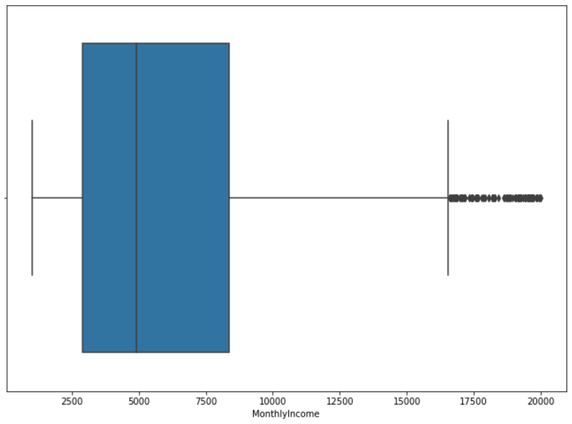


Fig50. Monthly Income box plot

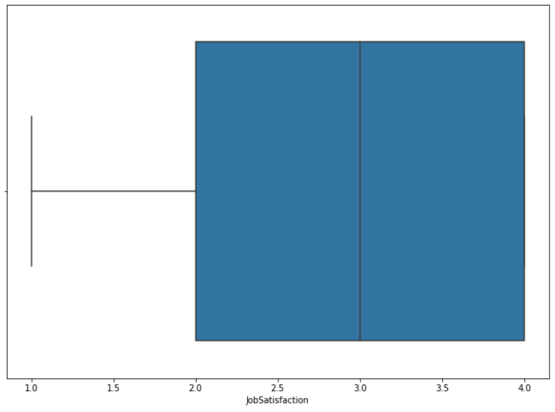


Fig 51. Job satisfaction Box plot

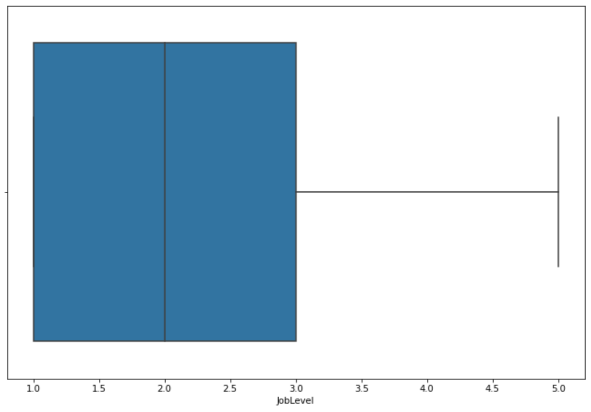


Fig52. Job level box plot

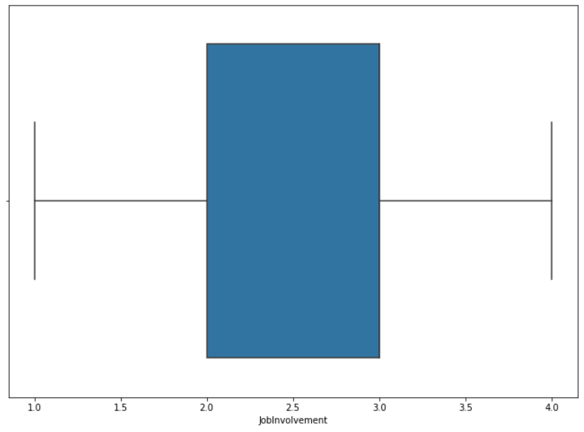


Fig 53. Job involvement box plot

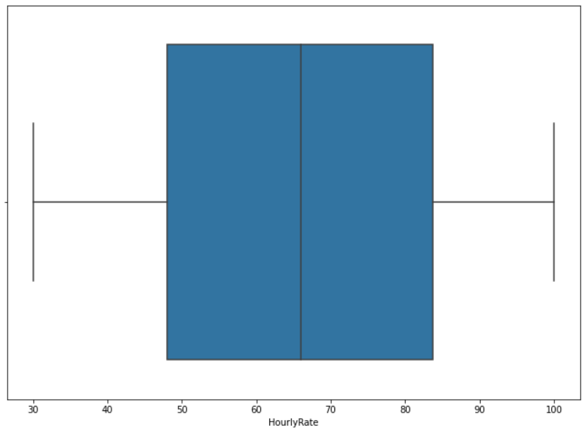


Fig54. Hourly rate box plot

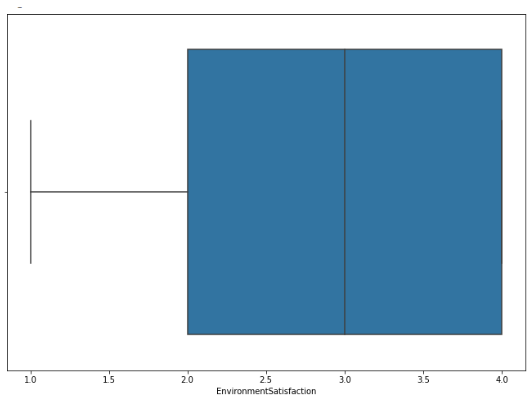


Fig 55. Environment satisfaction

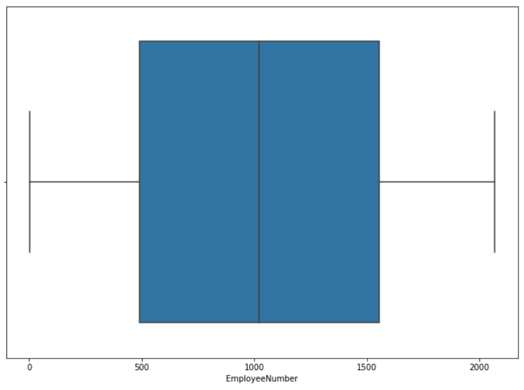


Fig56. Employe number box plot

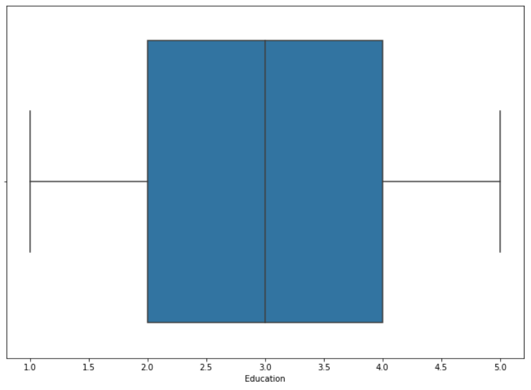


Fig 57. Education box plot

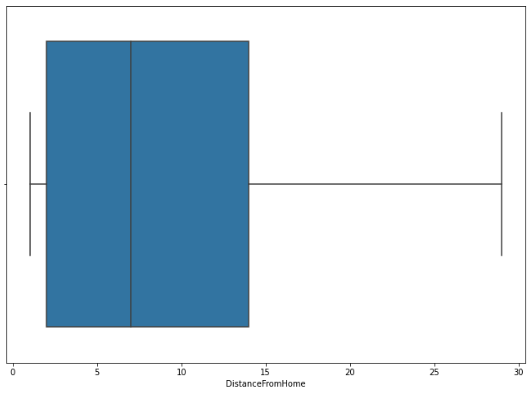


Fig58. Distance from home boxplot

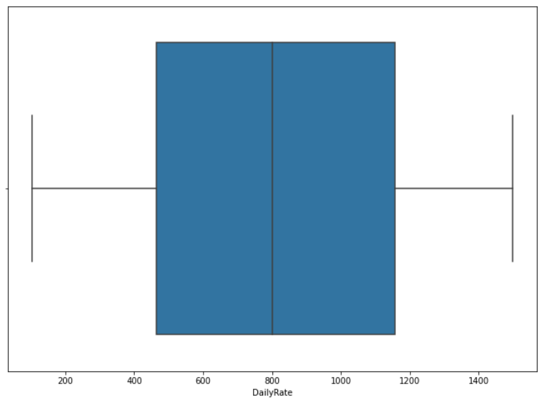
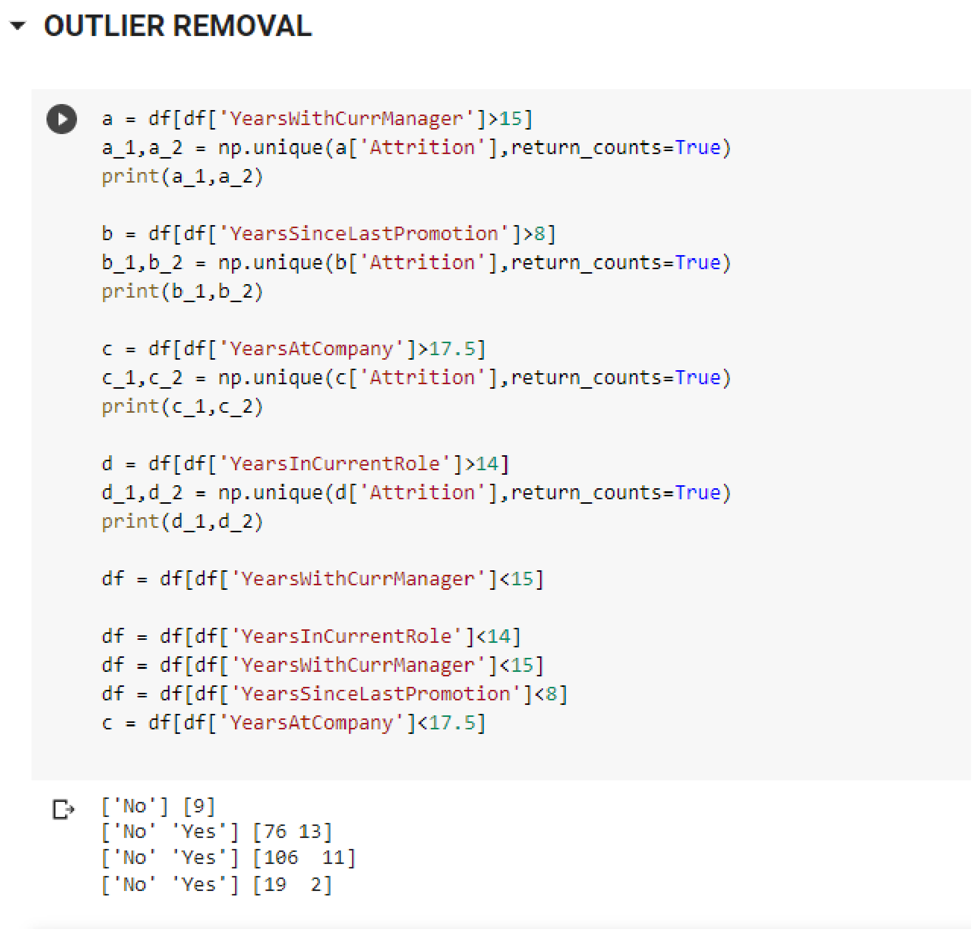


Fig 60. Daily rate boxplot

DATA PREPROCESSING:

i) Outlier removal:

Of all the features having outliers in the above box plots we remove outliers only for, Years with current manager , years since last promotion , years at company and years in current role
The reason is that if we remove outliers for other features having outliers such as Total working years (For example an employee could worked for a period significantly more than other employees) and monthly income (For example an employee might have very high salary) we might tend to lose out data points which are differ significantly from other observations, but carry actual information. Now for each of the above mentioned features based on the fact that the majority of the outliers in each feature contribute to no attribution (Shown in Fig.) , we find the threshold below which we would like all our data points to be in and then finally filter out the data to remove all the required outliers


OUTLIER REMOVAL

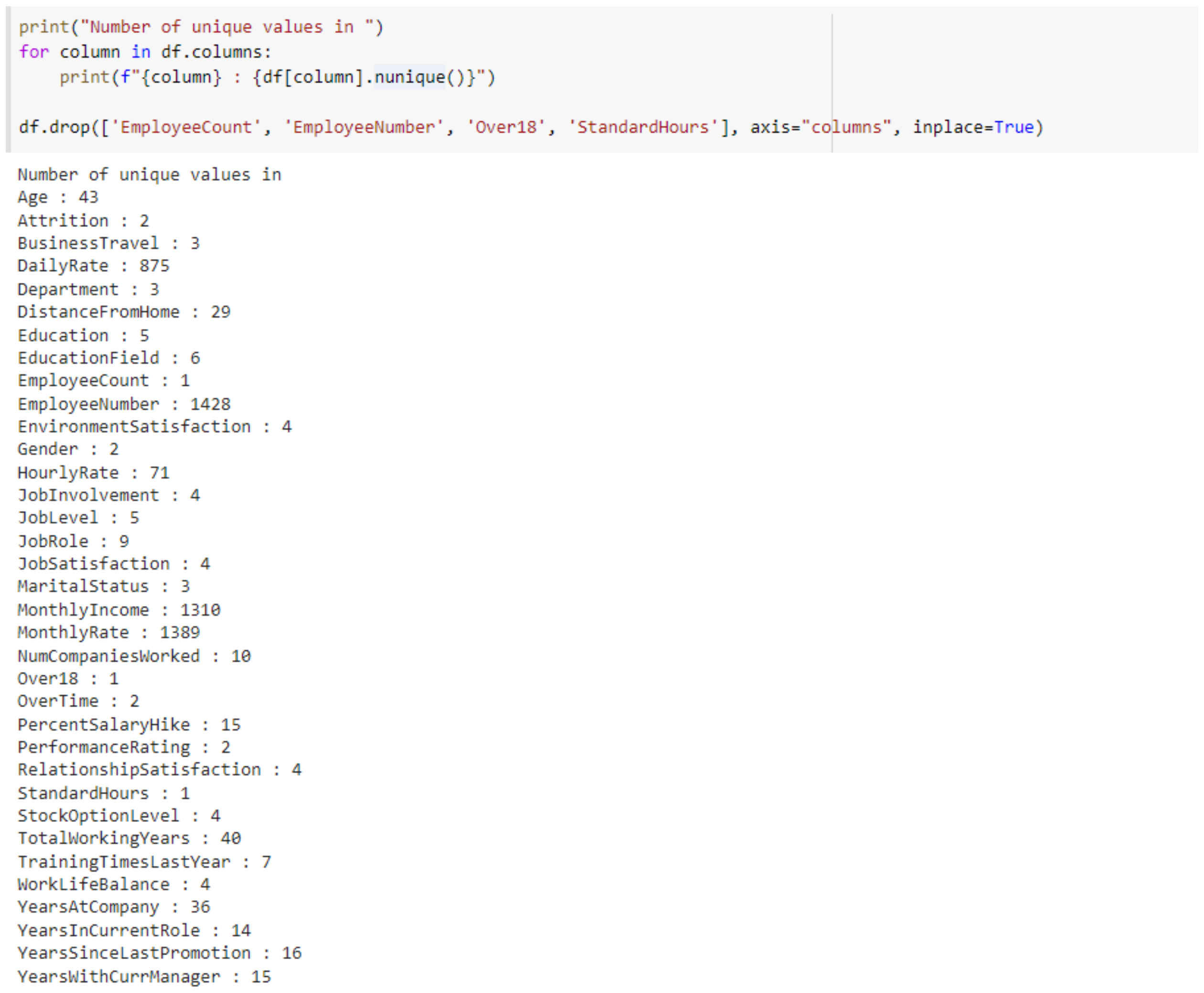
```
a = df[df['YearsWithCurrManager']>15]
a_1,a_2 = np.unique(a['Attrition'],return_counts=True)
print(a_1,a_2)

b = df[df['YearsSinceLastPromotion']>8]
b_1,b_2 = np.unique(b['Attrition'],return_counts=True)
print(b_1,b_2)

c = df[df['YearsAtCompany']>17.5]
c_1,c_2 = np.unique(c['Attrition'],return_counts=True)
print(c_1,c_2)

d = df[df['YearsInCurrentRole']>14]
d_1,d_2 = np.unique(d['Attrition'],return_counts=True)
print(d_1,d_2)

df = df[df['YearsWithCurrManager']<15]

df = df[df['YearsInCurrentRole']<14]
df = df[df['YearsWithCurrManager']<15]
df = df[df['YearsSinceLastPromotion']<8]
c = df[df['YearsAtCompany']<17.5]
```

```
['No'] [9]
['No' 'Yes'] [76 13]
['No' 'Yes'] [106  11]
['No' 'Yes'] [19  2]
```

Fig 61. Outlier removal code snippet

ii) Feature selection:

While performing EDA on the data set we found out that there were a few columns having only a single value such as , Employee count , Over 18 , Standard Hours have a single value , So we drop such columns.



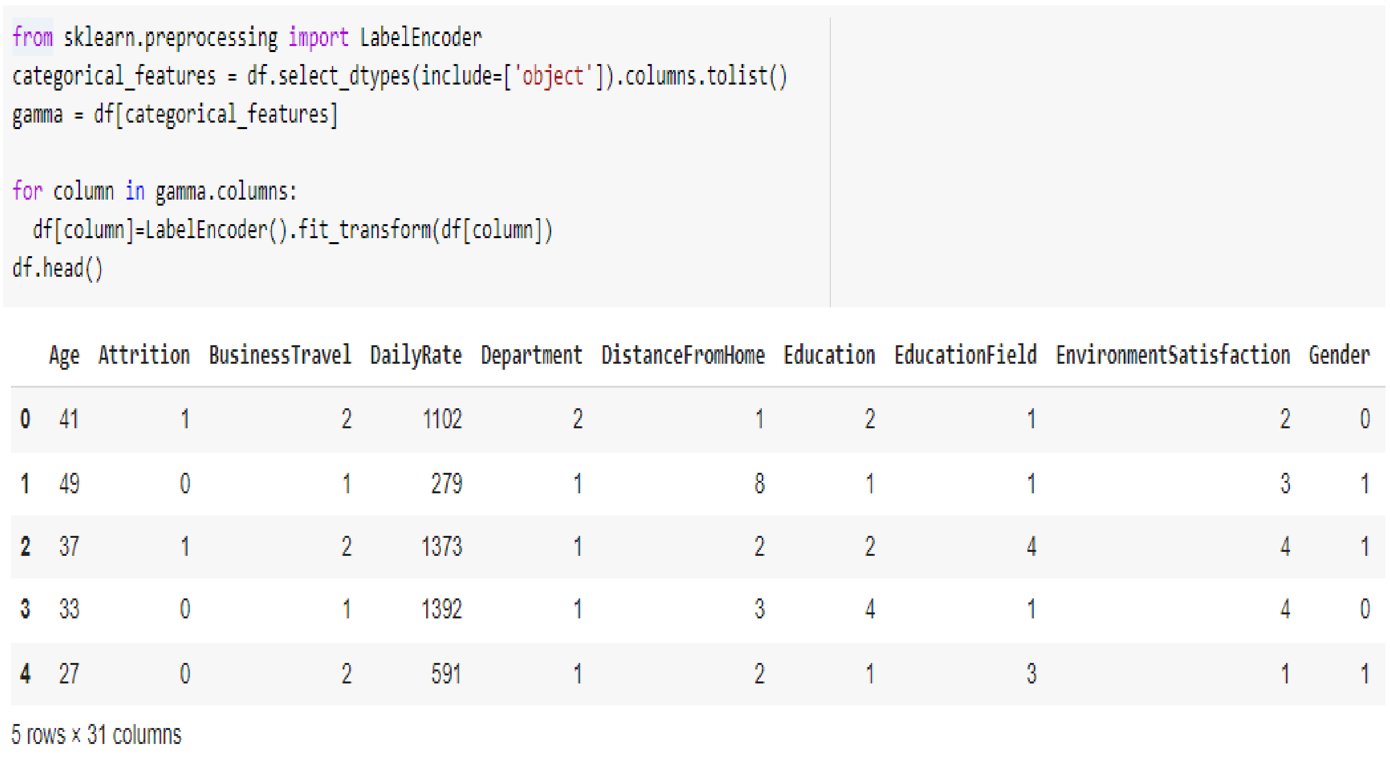
```
print("Number of unique values in ")
for column in df.columns:
    print(f"{column} : {df[column].nunique()}")

df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'], axis="columns", inplace=True)
```

```
Number of unique values in
Age : 43
Attrition : 2
BusinessTravel : 3
DailyRate : 875
Department : 3
DistanceFromHome : 29
Education : 5
EducationField : 6
EmployeeCount : 1
EmployeeNumber : 1428
EnvironmentSatisfaction : 4
Gender : 2
HourlyRate : 71
JobInvolvement : 4
JobLevel : 5
JobRole : 9
JobSatisfaction : 4
MaritalStatus : 3
MonthlyIncome : 1310
MonthlyRate : 1389
NumCompaniesWorked : 10
Over18 : 1
OverTime : 2
PercentSalaryHike : 15
PerformanceRating : 2
RelationshipSatisfaction : 4
StandardHours : 1
StockOptionLevel : 4
TotalWorkingYears : 40
TrainingTimesLastYear : 7
WorkLifeBalance : 4
YearsAtCompany : 36
YearsInCurrentRole : 14
YearsSinceLastPromotion : 16
YearsWithCurrManager : 15
```

Fig62. Code snippet showing number of unique values in each feature

iii) Encoding categorical features:

Now before we fit and evaluate our machine learning models we require all our input variables to the model to be numeric. So we encode all the categorical features to numeric values. For this we used the scikit-learn's LabelEncoder() , which encodes the corresponding values of each feature into values between (0,n_feature_classes-1).



```
from sklearn.preprocessing import LabelEncoder
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
gamma = df[categorical_features]

for column in gamma.columns:
  df[column]=LabelEncoder().fit_transform(df[column])
df.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction	Gender
0	41	1	2	1102	2	1	2	1	2	0
1	49	0	1	279	1	8	1	1	3	1
2	37	1	2	1373	1	2	2	4	4	1
3	33	0	1	1392	1	3	4	1	4	0
4	27	0	2	591	1	2	1	3	1	1

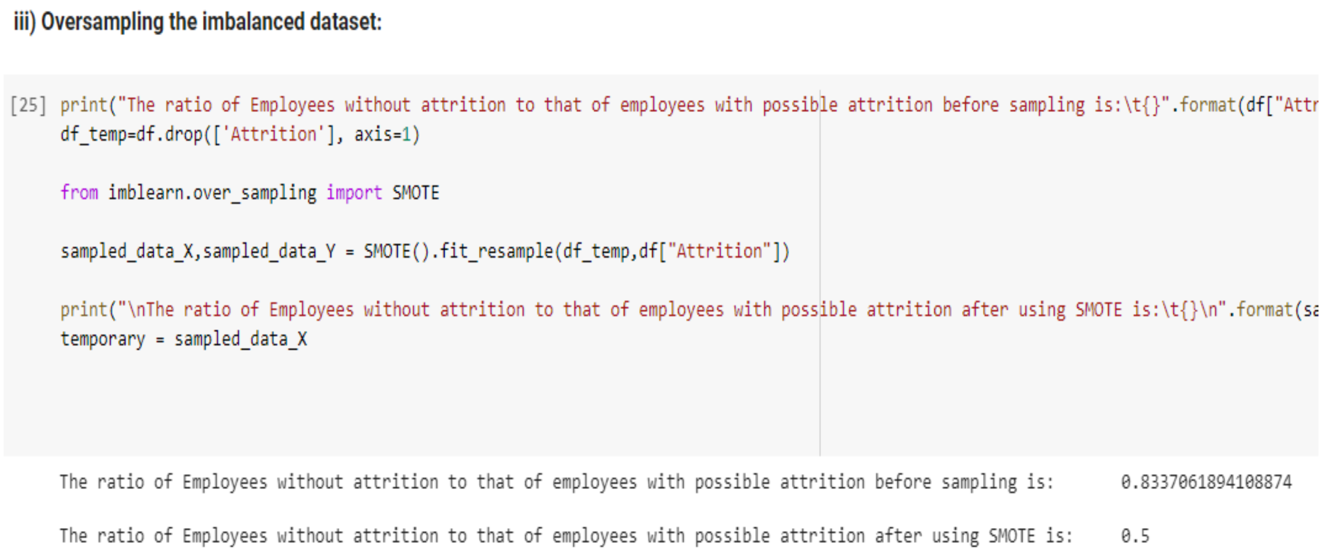
5 rows × 31 columns

Fig63. Data set after encoding the class labels of each categorical variable

iv) SMOTE :

As we can see below the ratio of employees without attrition to that of employees with attrition is 0.84. So we have an uneven distribution of observations with the minority class being employees prone to attrition and the majority class being employees who are not prone to attrition. Such imbalanced data sets tend to pose a challenge for predictions as most of the ML classification models have been designed for balanced data sets. So building our models on such a dataset would result in poor predictive performance especially for minority cases (Employees prone to attrition in our case) which is what we are interested in.

The solution to this problem would be to upsample the minority class so that the classifier would give equal importance to both the classes. Simply adding duplicate entries will not be of use to us as our model will not be getting any new information . So we need to generate new instances in our feature space. For this we used the SMOTE technique i.e. Synthetic minority oversampling technique in which we select a random nearest neighbor using K-NN for minority class instances and a new synthetic instance is generated at random in feature space. Using this we make our dataset balanced by oversampling the cases in which employees are prone to attrition. After upsampling we have 2236 sample points (initially the number of samples were 1341).



iii) Oversampling the imbalanced dataset:

```
[25] print("The ratio of Employees without attrition to that of employees with possible attrition before sampling is:\t{}".format(df["Attr
     df_temp=df.drop(['Attrition'], axis=1)

     from imblearn.over_sampling import SMOTE

     sampled_data_X,sampled_data_Y = SMOTE().fit_resample(df_temp,df["Attrition"])

     print("\nThe ratio of Employees without attrition to that of employees with possible attrition after using SMOTE is:\t{}\n".format(sa
     temporary = sampled_data_X
```

```
The ratio of Employees without attrition to that of employees with possible attrition before sampling is:       0.8337061894108874

The ratio of Employees without attrition to that of employees with possible attrition after using SMOTE is:     0.5
```

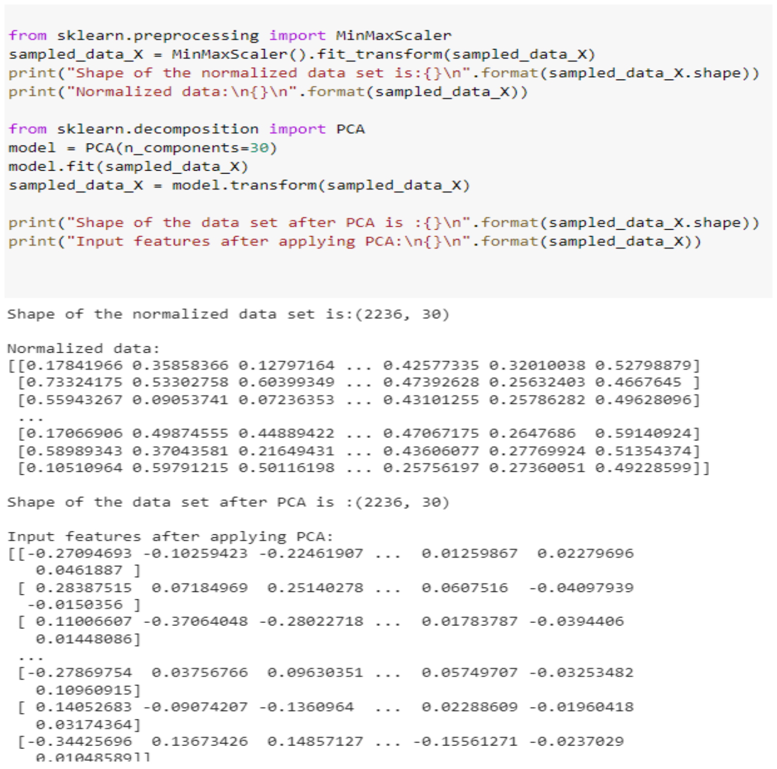
Fig 64. Result after oversampling the imbalanced data set using SMOTE

v) Normalizing features and PCA:

As we can see in the correlation plot , several independent variables(features) in the feature space are correlated to each other . Now because of this multicollinearity a small change in one of the features could lead to a significant amount of swing in the classification model's output. To avoid such a scenario we use the principal component analysis (PCA) .

However, because distinct features in our feature space have different units of measurement, the standard deviation of all features will vary substantially in scale.Now, PCA produces axes based on the variance (S.D) of our data, so it is necessary that we first normalize our data so that all the variables have the same S.D which would result in all variables having the same weight for PCA.

Here we used the MinMaxScaler from sklearn library to normalize our data first . Here the MinMaxScalar maps the values of all the features into range between [0,1] using mean and variance. Here for PCA we used n_components == min(number_samples, number_features)



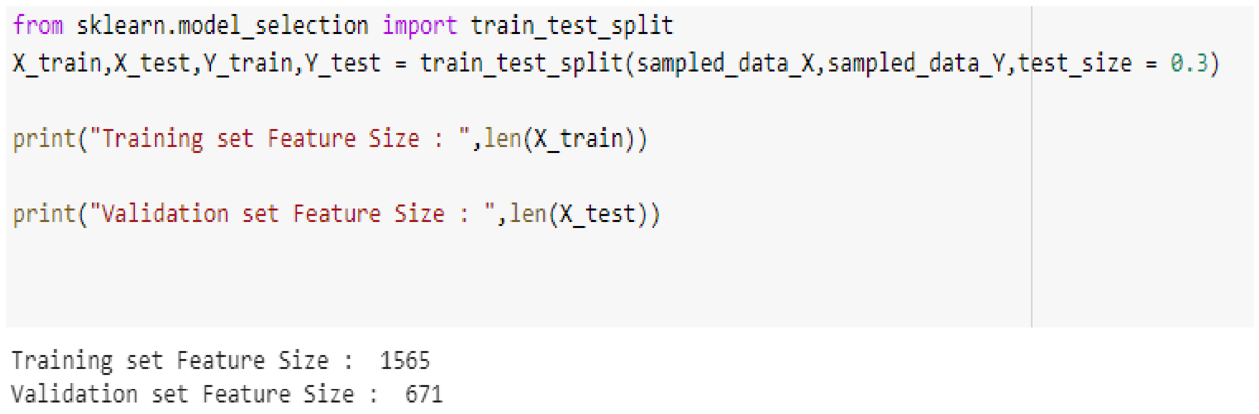
```
from sklearn.preprocessing import MinMaxScaler
sampled_data_X = MinMaxScaler().fit_transform(sampled_data_X)
print("Shape of the normalized data set is:{}\n".format(sampled_data_X.shape))
print("Normalized data:\n{}\n".format(sampled_data_X))

from sklearn.decomposition import PCA
model = PCA(n_components=30)
model.fit(sampled_data_X)
sampled_data_X = model.transform(sampled_data_X)

print("Shape of the data set after PCA is :{}\n".format(sampled_data_X.shape))
print("Input features after applying PCA:\n{}\n".format(sampled_data_X))
```

```
Shape of the normalized data set is:(2236, 30)

Normalized data:
[[0.17841966 0.35858366 0.12797164 ... 0.42577335 0.32010038 0.52798879]
 [0.73324175 0.53302758 0.60399349 ... 0.47392628 0.25632403 0.4667645 ]
 [0.55943267 0.09053741 0.07236353 ... 0.43101255 0.25786282 0.49628096]
 ...
 [0.17066906 0.49874555 0.44889422 ... 0.47067175 0.2647686  0.59140924]
 [0.58989343 0.37043581 0.21649431 ... 0.43606077 0.27769924 0.51354374]
 [0.10510964 0.59791215 0.50116198 ... 0.25756197 0.27360051 0.49228599]]

Shape of the data set after PCA is :(2236, 30)

Input features after applying PCA:
[[-0.27094693 -0.10259423 -0.22461907 ...  0.01259867  0.02279696
   0.0461887 ]
 [ 0.28387515  0.07184969  0.25140278 ...  0.0607516  -0.04097939
  -0.0150356 ]
 [ 0.11006607 -0.37064048 -0.28022718 ...  0.01783787 -0.0394406
   0.01448086]
 ...
 [-0.27869754  0.03756766  0.09630351 ...  0.05749707 -0.03253482
   0.10960915]
 [ 0.14052683 -0.09074207 -0.1360964  ...  0.02288609 -0.01960418
   0.03174364]
 [-0.34425696  0.13673426  0.14857127 ... -0.15561271 -0.0237029
   0.01048589]]
```

Fig 65. Normalizing the data and PCA code snippet

vi) Test-Train split:

Once data set has been completely cleaned and filtered , we split our data into test and train sets for model building and validation



```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(sampled_data_X,sampled_data_Y,test_size = 0.3)

print("Training set Feature Size : ",len(X_train))

print("Validation set Feature Size : ",len(X_test))
```

```
Training set Feature Size :  1565
Validation set Feature Size :  671
```

Fig 66. Test-train split

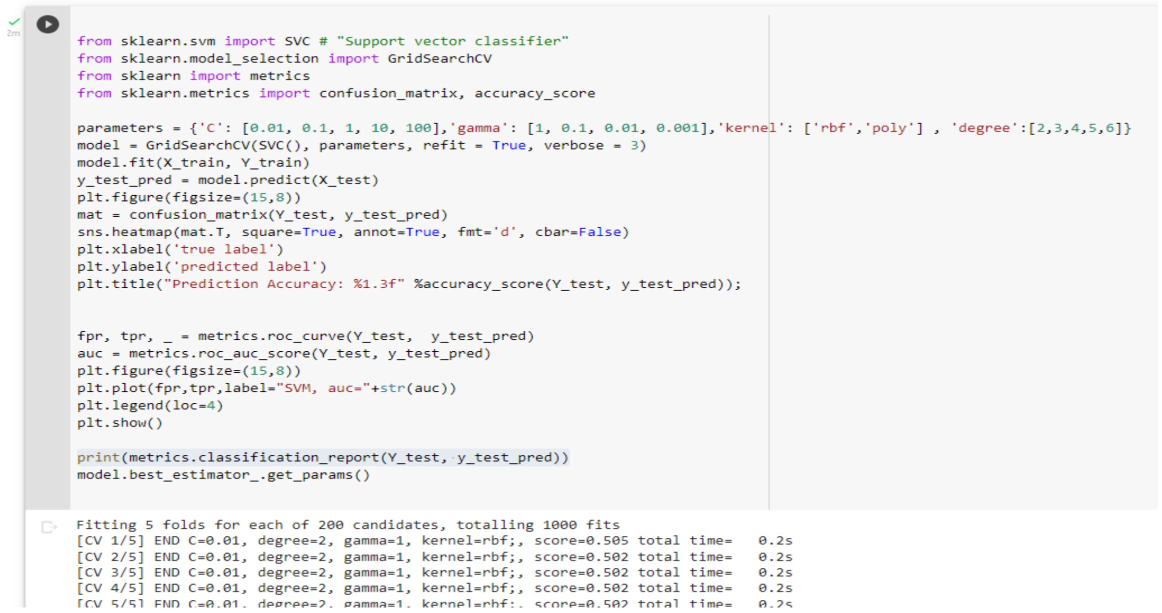
Model building

a) Support vector machine:

In the Support vector machine algorithm we generate a hyperplane in the feature space which separates data into classes. It tries to maximize the margin to the closest point. One of the main reasons to choose SVM for our dataset is because of its effectiveness in higher dimension feature spaces. Hyper parameter tuning for the model was done using Grid Search CV (Library function which is a part of scikit learn's model selection package) .

In this project we trained our Soft margin Support vector machine classification model with different penalty terms (C) and Gamma values for rbf and polynomial kernels.
As seen in Fig68. The best SVM classification model for our dataset uses the Radial Basis Function (RBF) kernel with Gamma value 1 and C value 10 , giving us a prediction accuracy of 91.8% and F1-scores of 0.92 for both the classes.

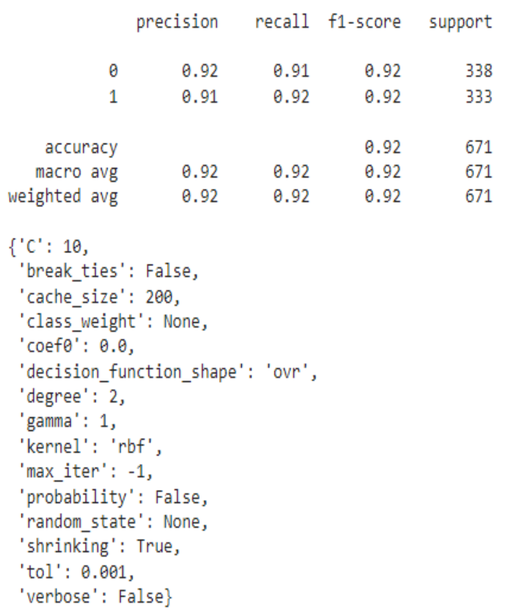
ii) Support Vector Machine (SVM)



```
from sklearn.svm import SVC # "Support vector classifier"
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import confusion_matrix, accuracy_score

parameters = {'C': [0.01, 0.1, 1, 10, 100],'gamma': [1, 0.1, 0.01, 0.001],'kernel': ['rbf','poly'] , 'degree':[2,3,4,5,6]}
model = GridSearchCV(SVC(), parameters, refit = True, verbose = 3)
model.fit(X_train, Y_train)
y_test_pred = model.predict(X_test)
plt.figure(figsize=(15,8))
mat = confusion_matrix(Y_test, y_test_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title("Prediction Accuracy: %1.3f" %accuracy_score(Y_test, y_test_pred));


fpr, tpr, _ = metrics.roc_curve(Y_test,  y_test_pred)
auc = metrics.roc_auc_score(Y_test, y_test_pred)
plt.figure(figsize=(15,8))
plt.plot(fpr,tpr,label="SVM, auc="+str(auc))
plt.legend(loc=4)
plt.show()

print(metrics.classification_report(Y_test, y_test_pred))
model.best_estimator_.get_params()
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
[CV 1/5] END C=0.01, degree=2, gamma=1, kernel=rbf;, score=0.505 total time=   0.2s
[CV 2/5] END C=0.01, degree=2, gamma=1, kernel=rbf;, score=0.502 total time=   0.2s
[CV 3/5] END C=0.01, degree=2, gamma=1, kernel=rbf;, score=0.502 total time=   0.2s
[CV 4/5] END C=0.01, degree=2, gamma=1, kernel=rbf;, score=0.502 total time=   0.2s
[CV 5/5] END C=0.01, degree=2, gamma=1, kernel=rbf;, score=0.502 total time=   0.2s
```

Fig 67. SVM code snippet



```
              precision    recall  f1-score   support

           0       0.92      0.91      0.92       338
           1       0.91      0.92      0.92       333

    accuracy                           0.92       671
   macro avg       0.92      0.92      0.92       671
weighted avg       0.92      0.92      0.92       671

{'C': 10,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 2,
 'gamma': 1,
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

Fig 68. SVM model evaluation metrics

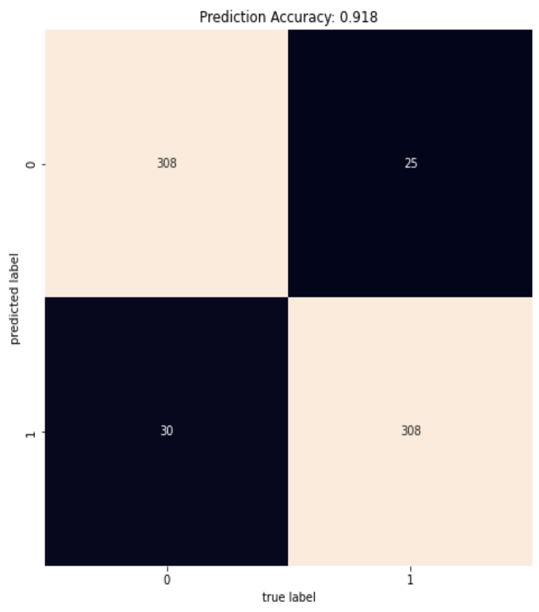


Fig 69. Confusion Matrix

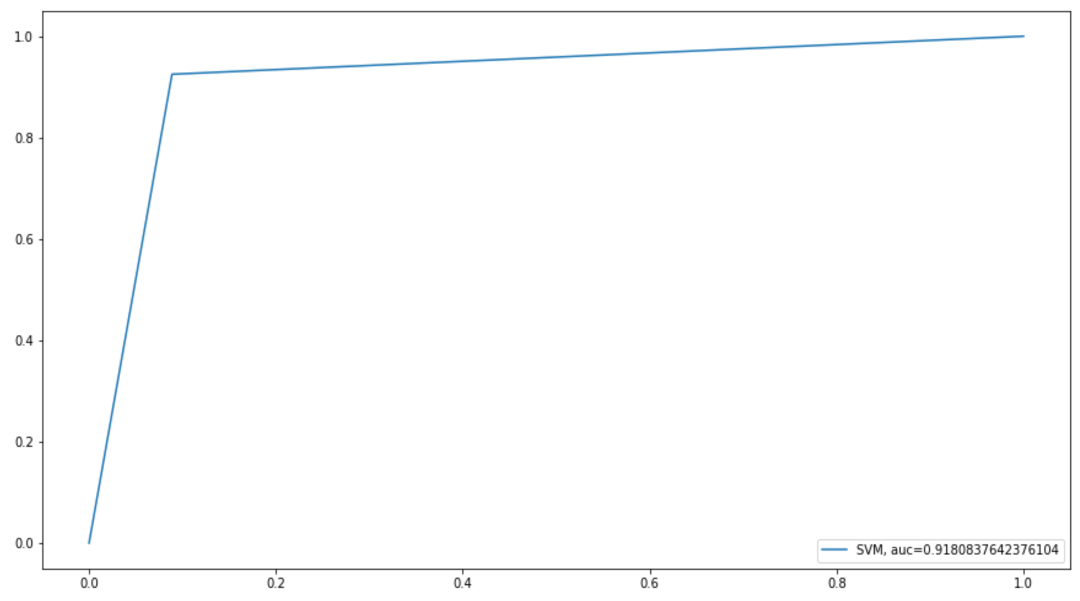


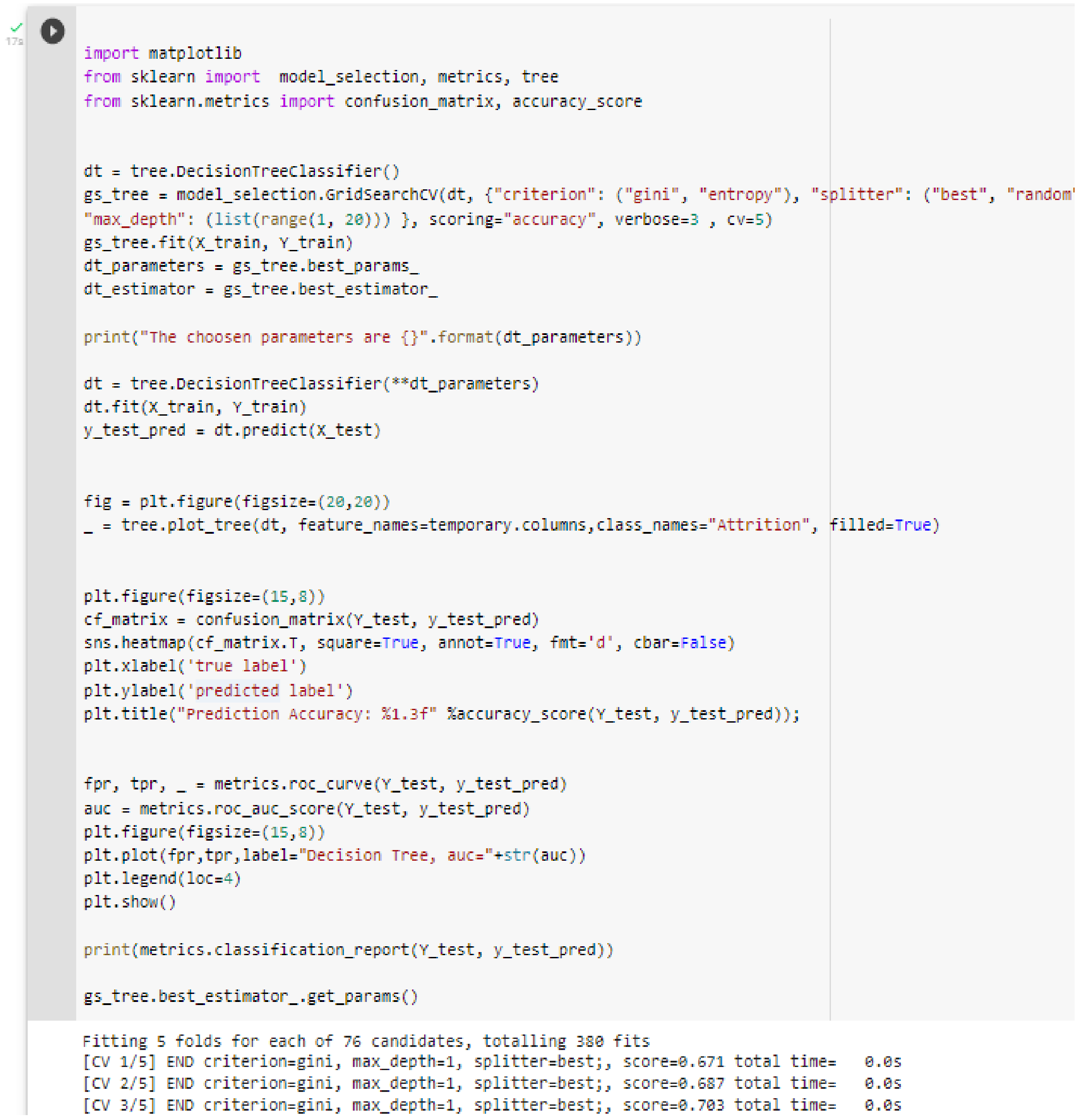
Fig 70. SVM ROC curve

b) Decision Tree :

Decision Tree classifiers are built as hierarchical data structures using divide and conquer strategy. In the decision tree we can predict the class of a sample by simply traversing the tree from root node to leaf node (leaf nodes give us the class labels) .Now given our data set we can construct many decision trees to represent the data , So our main goal would be to choose the best decision tree so that we can learn a good representation of our data.

In this project we used sklearn's tree library to build our classification model. Hyper parameter tuning for this model was done using Grid Search CV . For a Cross Validation value of 5 , we trained the data with splitting criterion as entropy and gini index on all possible splits for each chosen feature. The best decision tree classifier which best fits our data has a highest depth of 19 and uses entropy as the criterion for splitting.

We used sklearn.plot_tree to plot the final decision tree as shown in Fig 72 and Fig73. , as we traverse down the decision tree , I.G decreases with the root node (Age) having the highest Information gain . The final classification model gives us an accuracy of 79.1% and has an F1-Score of 0.79 for both positive and negative classes.



```
import matplotlib
from sklearn import  model_selection, metrics, tree
from sklearn.metrics import confusion_matrix, accuracy_score


dt = tree.DecisionTreeClassifier()
gs_tree = model_selection.GridSearchCV(dt, {"criterion": ("gini", "entropy"), "splitter": ("best", "random'
"max_depth": (list(range(1, 20))) }, scoring="accuracy", verbose=3 , cv=5)
gs_tree.fit(X_train, Y_train)
dt_parameters = gs_tree.best_params_
dt_estimator = gs_tree.best_estimator_

print("The choosen parameters are {}".format(dt_parameters))

dt = tree.DecisionTreeClassifier(**dt_parameters)
dt.fit(X_train, Y_train)
y_test_pred = dt.predict(X_test)


fig = plt.figure(figsize=(20,20))
_ = tree.plot_tree(dt, feature_names=temporary.columns,class_names="Attrition", filled=True)


plt.figure(figsize=(15,8))
cf_matrix = confusion_matrix(Y_test, y_test_pred)
sns.heatmap(cf_matrix.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title("Prediction Accuracy: %1.3f" %accuracy_score(Y_test, y_test_pred));


fpr, tpr, _ = metrics.roc_curve(Y_test, y_test_pred)
auc = metrics.roc_auc_score(Y_test, y_test_pred)
plt.figure(figsize=(15,8))
plt.plot(fpr,tpr,label="Decision Tree, auc="+str(auc))
plt.legend(loc=4)
plt.show()

print(metrics.classification_report(Y_test, y_test_pred))

gs_tree.best_estimator_.get_params()
```

```
Fitting 5 folds for each of 76 candidates, totalling 380 fits
[CV 1/5] END criterion=gini, max_depth=1, splitter=best;, score=0.671 total time=   0.0s
[CV 2/5] END criterion=gini, max_depth=1, splitter=best;, score=0.687 total time=   0.0s
[CV 3/5] END criterion=gini, max_depth=1, splitter=best;, score=0.703 total time=   0.0s
```

Fig71. Decision Tree code snippet

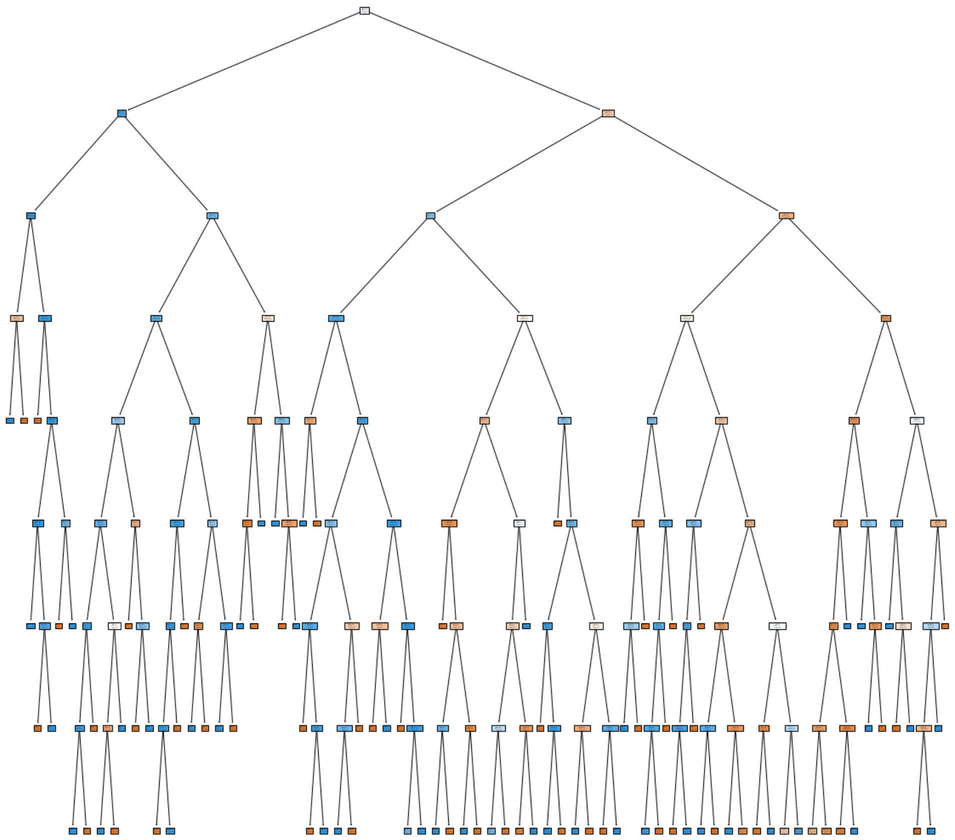


Fig 72. Visualization of Decision Tree

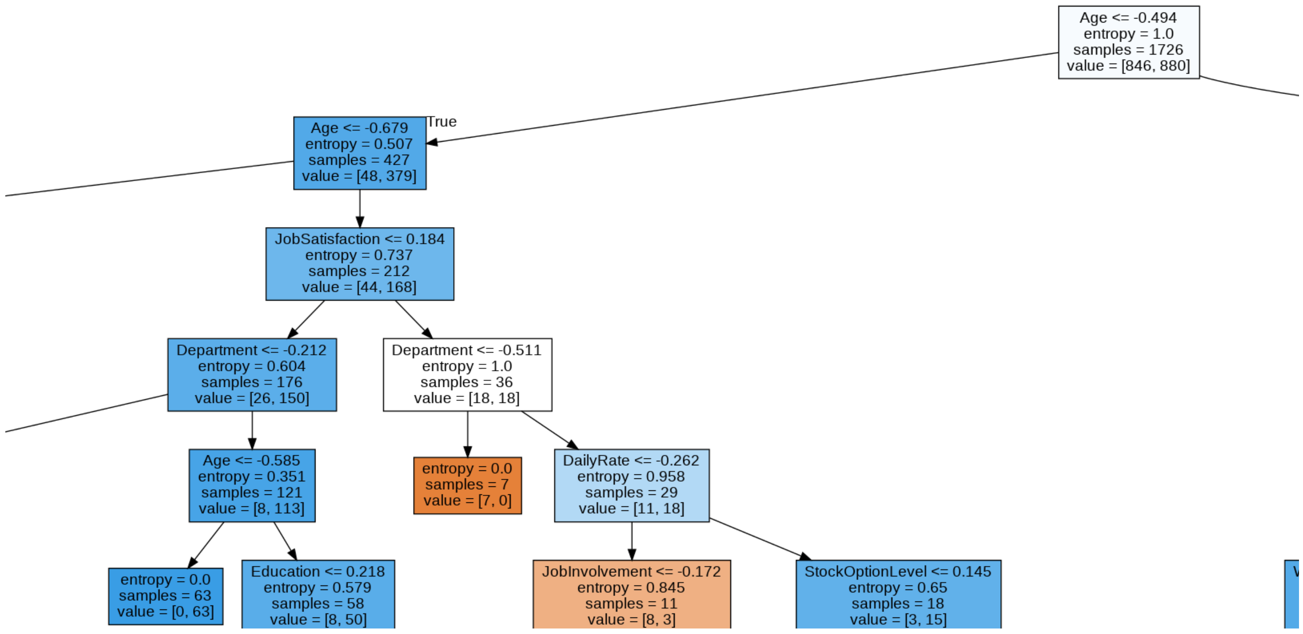


Fig 73. Visualizing the criterion (entropy) used for splitting our decision tree

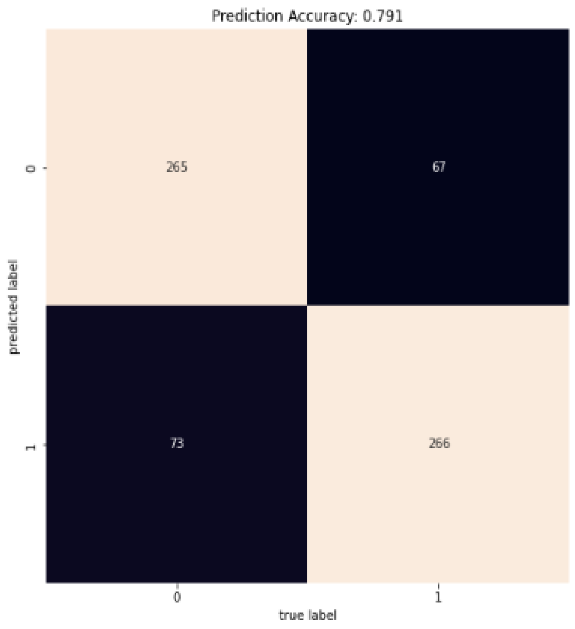
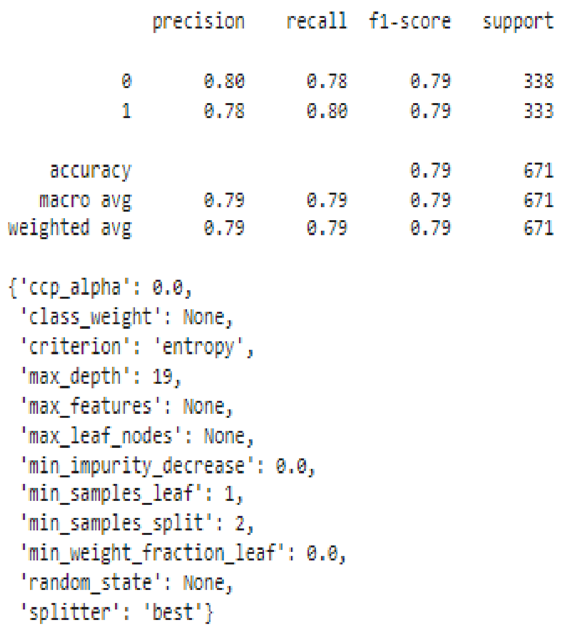


Fig 74. Decision Tree confusion matrix



```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       338
           1       0.78      0.80      0.79       333

    accuracy                           0.79       671
   macro avg       0.79      0.79      0.79       671
weighted avg       0.79      0.79      0.79       671

{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 19,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

Fig75. Decision Tree evaluation metrics

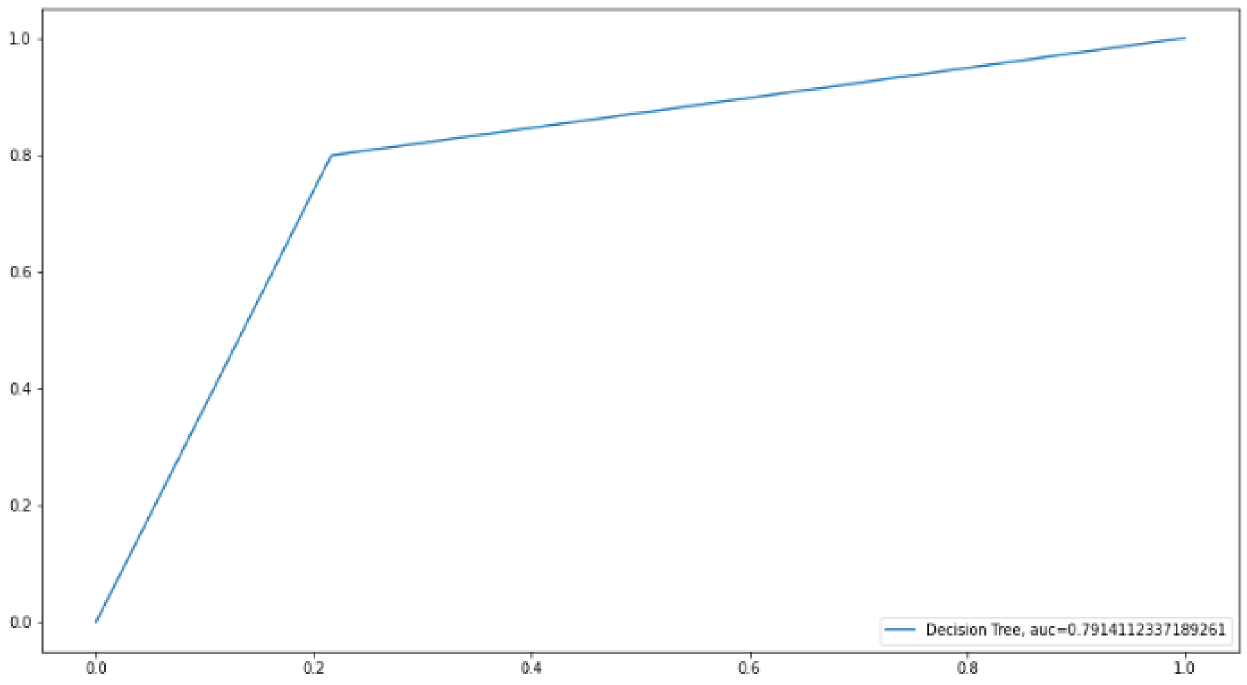
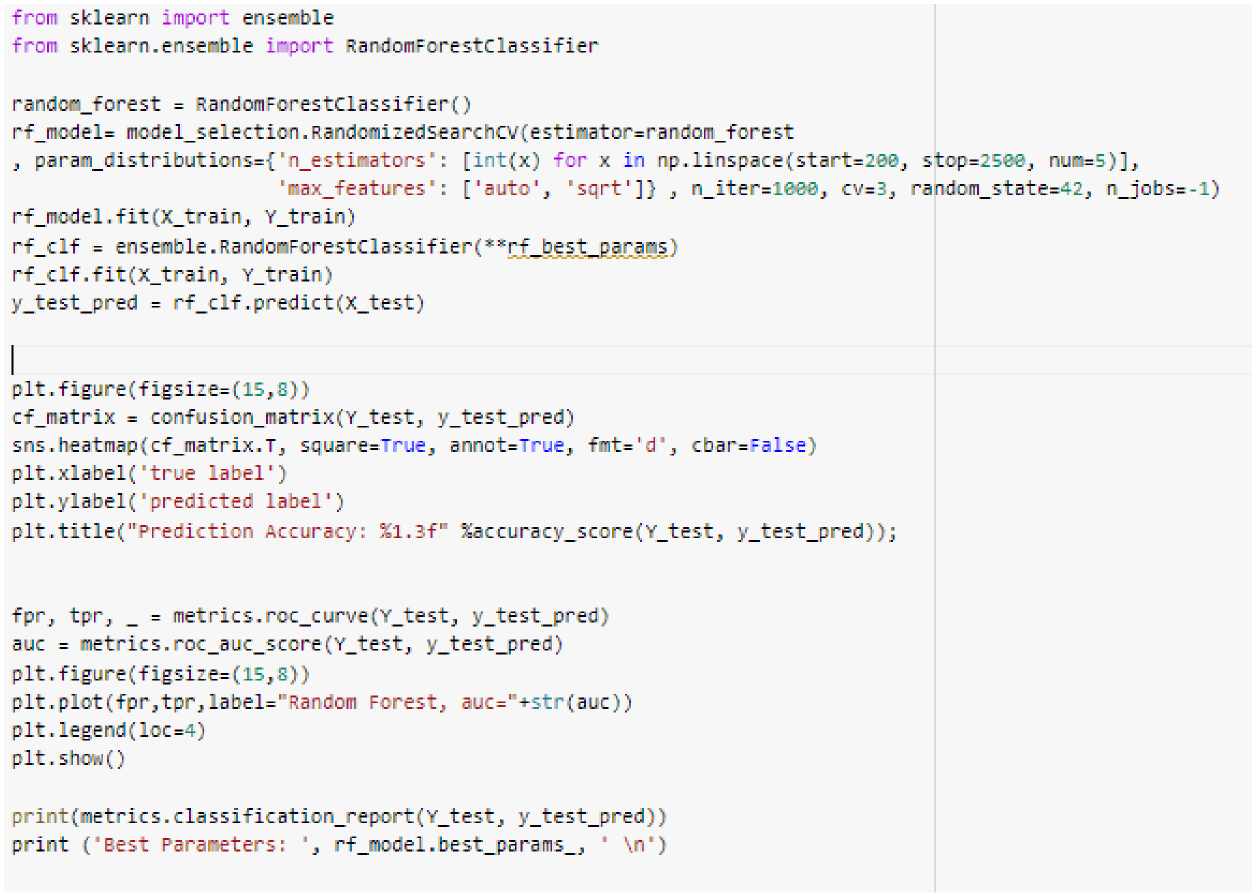


Fig 76. Decision Tree ROC curve

c) Random forest:

Random Forest is an ensemble classification technique majorly used for classification and regression tasks. The "Forest" here refers to the ensemble of trees generated via bootstrap aggregation (bagging) and feature randomness . Hyper parameter tuning for random forest was done using Random Search CV (Here we change the number of trees in random forest and maximum number of features allowed in each tree) using a cross validation value of 3.

The random forest classification model that best fits our data uses 1350 estimators and has an accuracy of 88.37% and an F1-Score of 0.88 for positive class and 0.89 for the negative class.



```
from sklearn import ensemble
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier()
rf_model= model_selection.RandomizedSearchCV(estimator=random_forest
, param_distributions={'n_estimators': [int(x) for x in np.linspace(start=200, stop=2500, num=5)],
                       'max_features': ['auto', 'sqrt']} , n_iter=1000, cv=3, random_state=42, n_jobs=-1)
rf_model.fit(X_train, Y_train)
rf_clf = ensemble.RandomForestClassifier(**rf_best_params)
rf_clf.fit(X_train, Y_train)
y_test_pred = rf_clf.predict(X_test)


plt.figure(figsize=(15,8))
cf_matrix = confusion_matrix(Y_test, y_test_pred)
sns.heatmap(cf_matrix.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title("Prediction Accuracy: %1.3f" %accuracy_score(Y_test, y_test_pred));


fpr, tpr, _ = metrics.roc_curve(Y_test, y_test_pred)
auc = metrics.roc_auc_score(Y_test, y_test_pred)
plt.figure(figsize=(15,8))
plt.plot(fpr,tpr,label="Random Forest, auc="+str(auc))
plt.legend(loc=4)
plt.show()

print(metrics.classification_report(Y_test, y_test_pred))
print ('Best Parameters: ', rf_model.best_params_, ' \n')
```

Fig 77. Random Forest code snippet

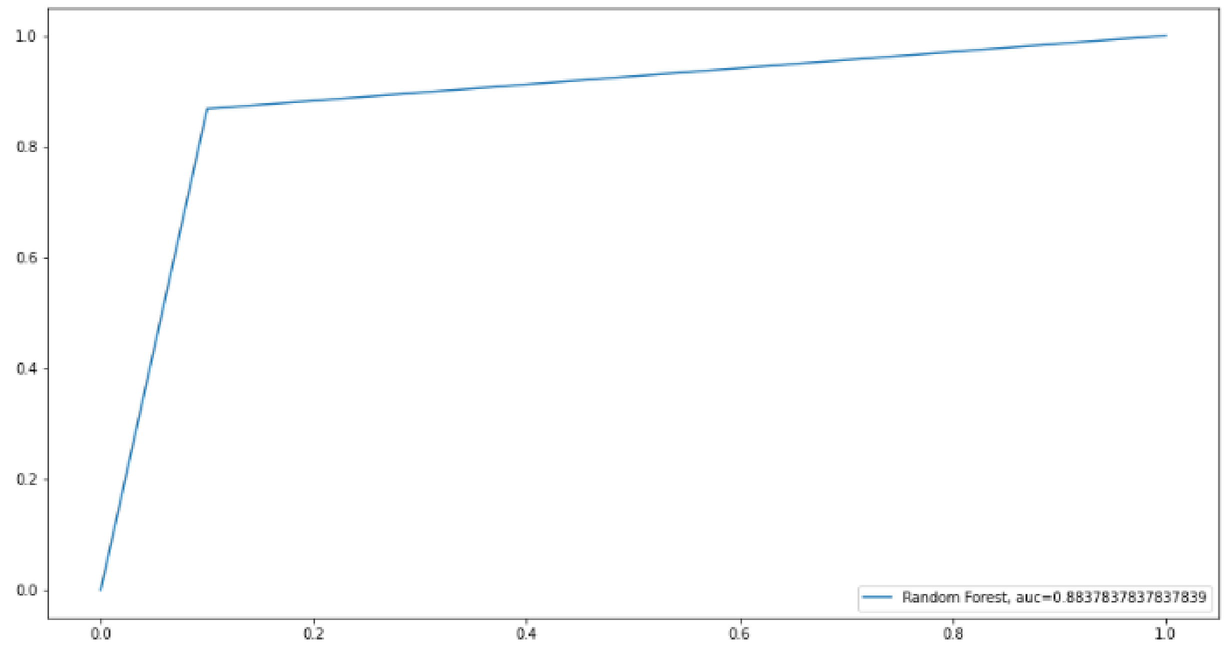
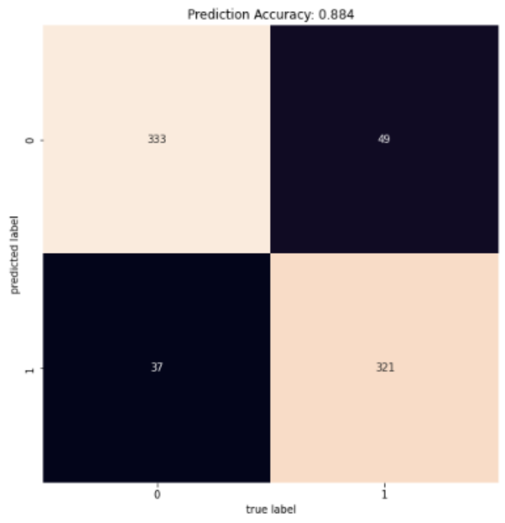
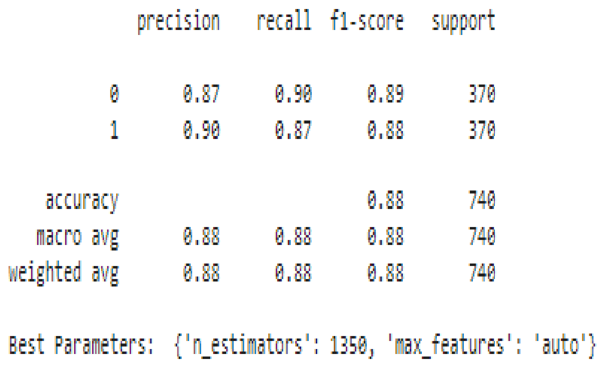


Fig 78. Random Forest ROC curve





```
              precision    recall  f1-score   support

           0       0.87      0.90      0.89       370
           1       0.90      0.87      0.88       370

    accuracy                           0.88       740
   macro avg       0.88      0.88      0.88       740
weighted avg       0.88      0.88      0.88       740

Best Parameters:  {'n_estimators': 1350, 'max_features': 'auto'}
```

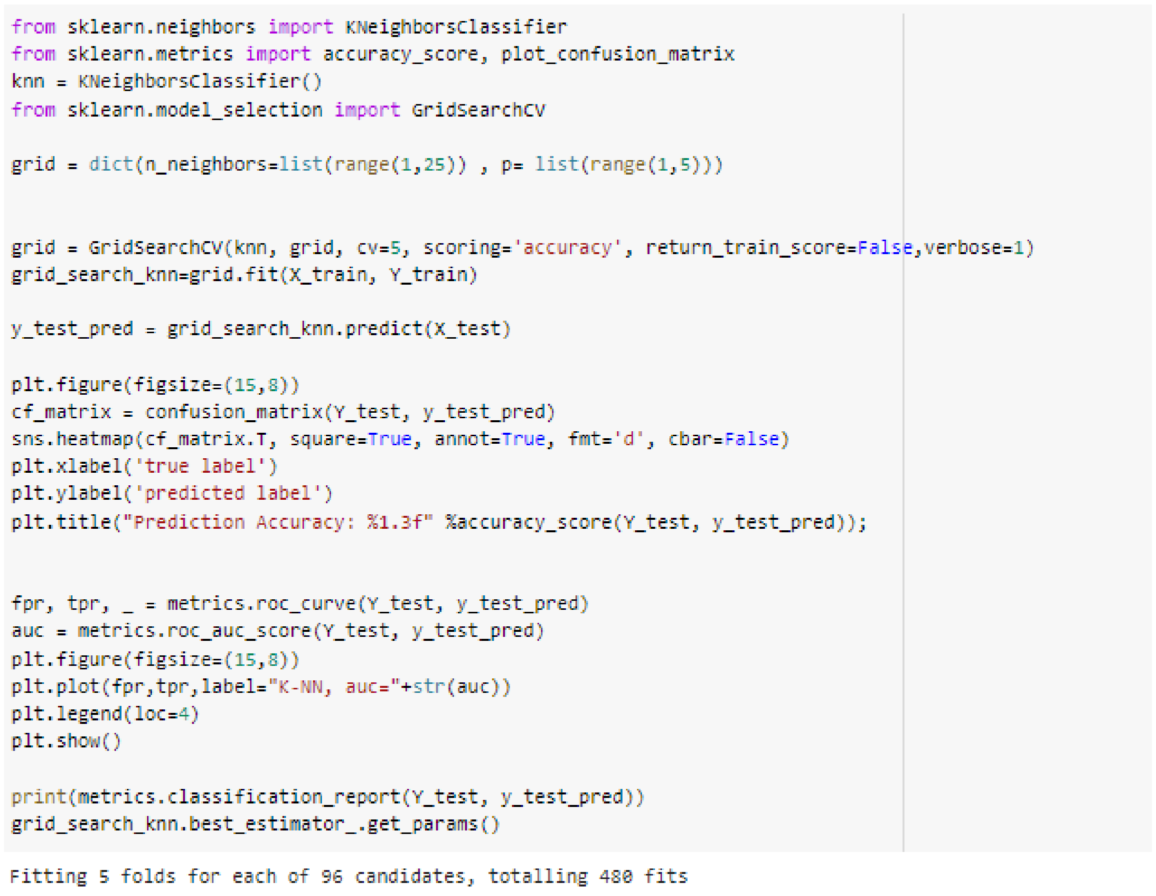
Fig 79. Random forest Confusion Matrix

Fig 80. RF metrics and optimal parameters

e) K-NN :

In this algorithm using the labels of k- nearest neighbors , the training data partitions the feature space. While using K-NN for classification we predict the class of a sample as the most frequently occurring label among its neighbors. Distance to these "k" closest neighbors can be measured using different metrics.

For the hyper parameter training of our model we used Minowski distance as the measurement metric , with the values of p ranging from 1 to 4 i.e we used L1,L2,L3,4 norms for measuring the distance to closest neighbors. We tested all the above norms for values of k ranging from 1 to 24 and cross validation value of 5 . The best classification model has k=2 and uses Manhattan distance as the distance measurement metric , giving us an accuracy of 88.8% with F1 scores of 0.88 , 0.90 for positive and negative classes respectively.



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, plot_confusion_matrix
knn = KNeighborsClassifier()
from sklearn.model_selection import GridSearchCV

grid = dict(n_neighbors=list(range(1,25)) , p= list(range(1,5)))


grid = GridSearchCV(knn, grid, cv=5, scoring='accuracy', return_train_score=False,verbose=1)
grid_search_knn=grid.fit(X_train, Y_train)

y_test_pred = grid_search_knn.predict(X_test)

plt.figure(figsize=(15,8))
cf_matrix = confusion_matrix(Y_test, y_test_pred)
sns.heatmap(cf_matrix.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title("Prediction Accuracy: %1.3f" %accuracy_score(Y_test, y_test_pred));


fpr, tpr, _ = metrics.roc_curve(Y_test, y_test_pred)
auc = metrics.roc_auc_score(Y_test, y_test_pred)
plt.figure(figsize=(15,8))
plt.plot(fpr,tpr,label="K-NN, auc="+str(auc))
plt.legend(loc=4)
plt.show()

print(metrics.classification_report(Y_test, y_test_pred))
grid_search_knn.best_estimator_.get_params()
```

```
Fitting 5 folds for each of 96 candidates, totalling 480 fits
```

Fig 80. KNN code snippet

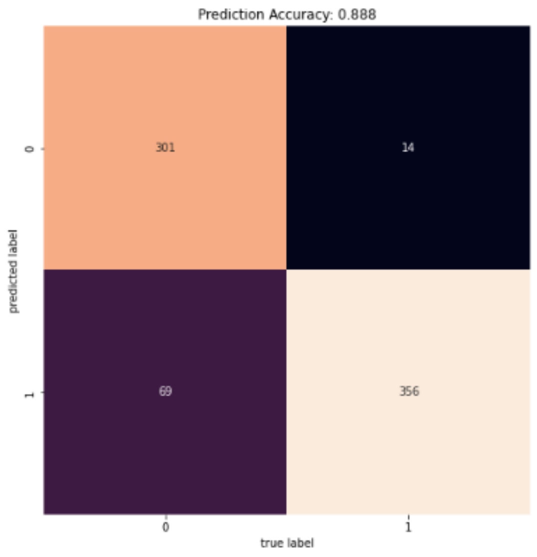


Fig 81. K-NN Confusion matrix

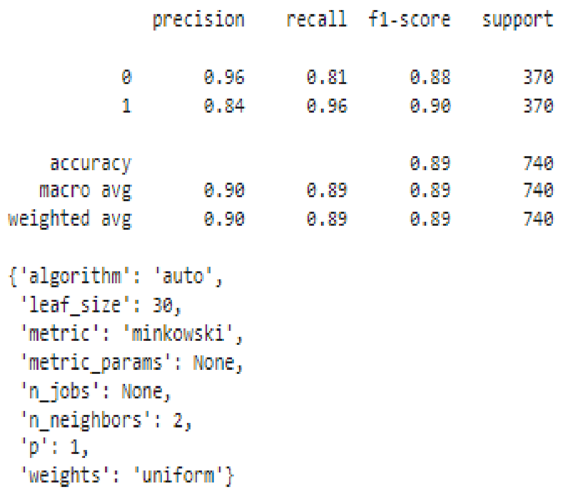

```
              precision    recall  f1-score   support

           0       0.96      0.81      0.88       370
           1       0.84      0.96      0.90       370

    accuracy                           0.89       740
   macro avg       0.90      0.89      0.89       740
weighted avg       0.90      0.89      0.89       740

{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 2,
 'p': 1,
 'weights': 'uniform'}
```

Fig 82. K-NN Evaluation Metrics

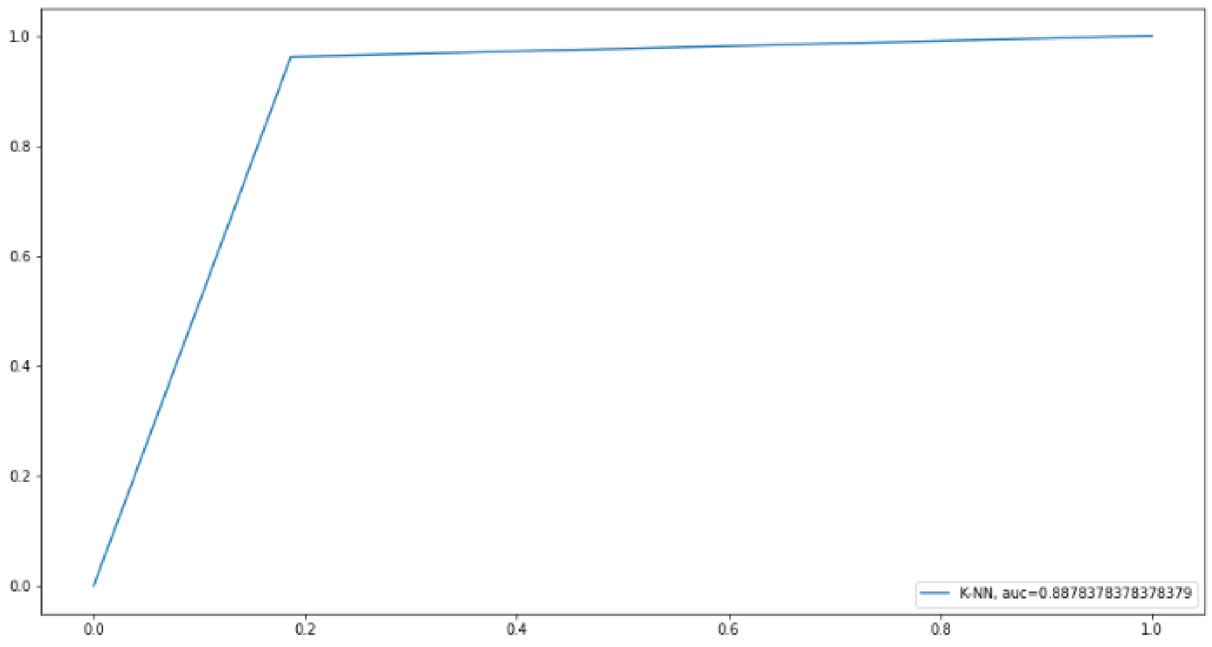


Fig 83. K-NN ROC curve

Conclusion and future work:

To measure the performance of machine learning models we used accuracy , Area under ROC curve and F1-score as our metrics. We used F1-score as a metric for measuring the classification performance because of its ability to not only attain a balance between precision and recall , but also because our data set is highly imbalanced.

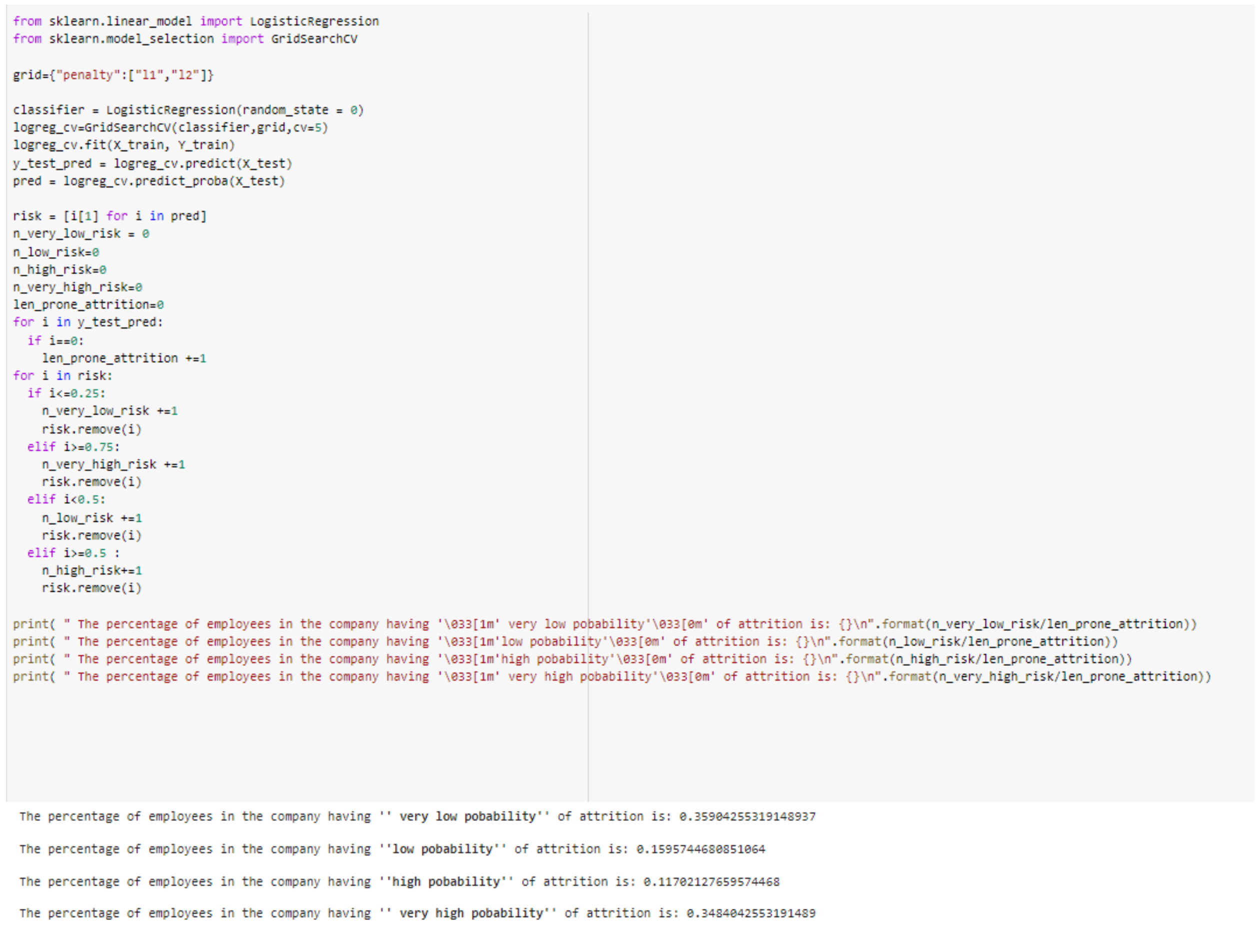
Now summarizing all the data we have on hand we find out Support Vector Machine (SVM) algorithm (gamma=1 , c=10 and kernel as rbf kernel) offers us the best attrition prediction model with F-1 Score = 0.92 , AUC = 0.918 and accuracy = 92%

		Precision	Recall	F1-Score	Support	Accuracy
SVM	**0**	0.92	0.91	0.92	338	92%
	1	0.91	0.92	0.92	333	
K-NN	**0**	0.96	0.81	0.88	370	88.8%
	1	0.84	0.96	0.90	370	
Decision Tree	**0**	0.8	0.78	0.78	338	79.1%
	1	0.78	0.8	0.80	333	
Random Forest	**0**	0.87	0.90	0.89	370	88.4%
	1	0.90	0.87	0.88	370	

In addition to predicting employee attrition (Which employee might leave the company next quarter) , we can also use the predicted class probabilities of each employee to actually find out the risk category (Amount of risk involved of an employee to leave the company) .

As an example, We assigned every employee a "risk category" based on the probability of predicted labels.,

i) Very Low Risk : Probability of predicted label<0.25
ii) Low Risk : Probability of predicted label>=0.25&<0.5
iii) High Risk : Probability of predicted label >=0.5 & < 0.75
iv) Very High Risk : Probability of predicted label >=0.75



```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

grid={"penalty":["l1","l2"]}

classifier = LogisticRegression(random_state = 0)
logreg_cv=GridSearchCV(classifier,grid,cv=5)
logreg_cv.fit(X_train, Y_train)
y_test_pred = logreg_cv.predict(X_test)
pred = logreg_cv.predict_proba(X_test)

risk = [i[1] for i in pred]
n_very_low_risk = 0
n_low_risk=0
n_high_risk=0
n_very_high_risk=0
len_prone_attrition=0
for i in y_test_pred:
  if i==0:
    len_prone_attrition +=1
for i in risk:
  if i<=0.25:
    n_very_low_risk +=1
    risk.remove(i)
  elif i>=0.75:
    n_very_high_risk +=1
    risk.remove(i)
  elif i<0.5:
    n_low_risk +=1
    risk.remove(i)
  elif i>=0.5 :
    n_high_risk+=1
    risk.remove(i)

print( " The percentage of employees in the company having '\033[1m' very low pobability'\033[0m' of attrition is: {}\n".format(n_very_low_risk/len_prone_attrition))
print( " The percentage of employees in the company having '\033[1m'low pobability'\033[0m' of attrition is: {}\n".format(n_low_risk/len_prone_attrition))
print( " The percentage of employees in the company having '\033[1m'high pobability'\033[0m' of attrition is: {}\n".format(n_high_risk/len_prone_attrition))
print( " The percentage of employees in the company having '\033[1m' very high pobability'\033[0m' of attrition is: {}\n".format(n_very_high_risk/len_prone_attrition))
```

```
 The percentage of employees in the company having '' very low pobability'' of attrition is: 0.35904255319148937

 The percentage of employees in the company having ''low pobability'' of attrition is: 0.1595744680851064

 The percentage of employees in the company having ''high pobability'' of attrition is: 0.11702127659574468

 The percentage of employees in the company having '' very high pobability'' of attrition is: 0.3484042553191489
```

Fig 84. Attrition risk prediction for employee

As shown in the figure above , we tried this method by using logistic regression (Accuracy 82%) and found out that 34.84% of all the employees have a very high probability to leave the company. Now any retention plan would have to consider all the strongest indicators causing attrition (which can be drawn based on feature importance or strength of correlation of a feature w.r.t attrition). Now for the given data set , the strongest indicators for attrition are , Age , Job level , Distance From Home , Monthly Income , Years with current manager , Total working years and years in current role. Now keeping all these factors in mind , the HR management team can act accordingly and draw a retention plan for each risk category to mitigate any hindrances caused by attrition. Though we have been able to predict and build a strategic retention plan for employees who are prone to attrition, the predictions based on our data can be further improved . Few such scenarios where we believe that our project has the most potential for future growth is in making our data set more balanced by increasing the number of samples of employees prone to attrition (Design of a better balanced data set) , implementation of better risk buckets and strategic retention plans . Moreover ,implementing our classification models using deep neural networks is also one of the most possible areas in this project having a great growth potential in future.

References:

[1]N. Bhartiya, S. Jannu, P. Shukla and R. Chapaneri, "Employee Attrition Prediction Using Classification Models," 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), 2019, pp. 1-6, doi: 10.1109/I2CT45611.2019.9033784.

[2] P. Sadana and D. Munnuru, "Machine Learning Model to Predict WorkForce Attrition," 2021 6th International Conference for Convergence in Technology (I2CT), 2021, pp. 1-6, doi: 10.1109/I2CT51068.2021.9418140.

[3] S. S. Alduayj and K. Rajpoot, "Predicting Employee Attrition using Machine Learning," 2018 International Conference on Innovations in Information Technology (IIT), 2018, pp. 93-98, doi: 10.1109/INNOVATIONS.2018.8605976.

[4]Employee Turnover Prediction with Machine Learning: A Reliable Approach Yue Zhao1() , Maciej K. Hryniewicki2 , Francesca Cheng2, Boyang Fu3, and Xiaoyu Zhu4

Contributions:

1. Gautam Varma Datla : Exploratory data analysis, Data Preprocessing , Model building, Model evaluation
2. Siddardha Varma Vegesna : Research on data set , Exploratory data analysis , Model Building , Conclusion and Future work
3. Dhushyanth Polkampalli: Research on data set , Model evaluation , Conclusion and future work