Loss of Employees in Company

Ever thought what happens when a company loose its employees ? For a service provider company , customer generally feels comfortable in commuting to the known person .The term that defines loss of employees in one word is called Attrition .

There are features like business travel , relationship satisfaction , years in current role , working hours , and many more , that conclude weather there will be attrition or no attrition . In the following dataset , the aim is to build a binary classification model , Lets import the necessary libraries ,

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
```

```
data=pd.read_csv(r"C:\Users\shashank agarwal\Downloads\ibm-hr-analytics-employee-attrition-performance (1).zip")
```

```
data.head(6)
```

The dataset looks like a bundle of features predicting the result , the size being 1470 rows and 35 columns ,having no null values , which is good!

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber
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	32	No	Travel_Frequently	1005	Research & Development	2	2	Life Sciences	1	8

Further analysis says that , the data is showing a good curve , but the target variable needs to be balanced so as to avoid the problem of overfitting of data , resulting in biased model . Tools like smote , nearmiss will act as a blessing here , split the data in 3 :1 ratio for training and testing , using train test split and using smote to balance the outcomes like this ….

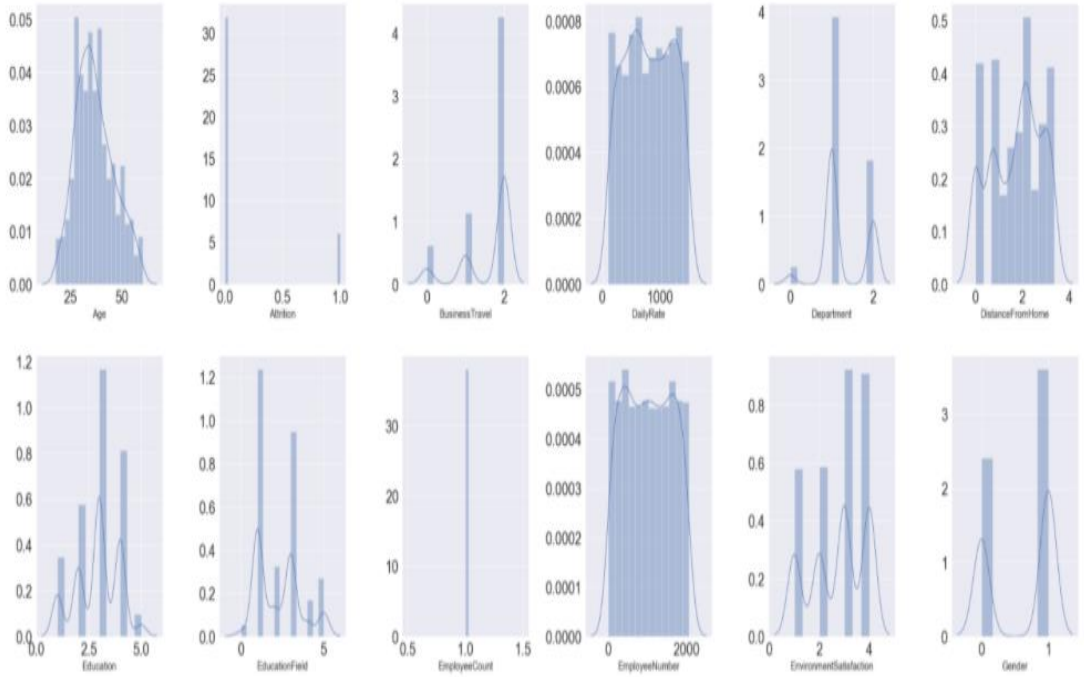
```
x=data.drop("Attrition",axis=1)
y=data.Attrition
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=41)
```

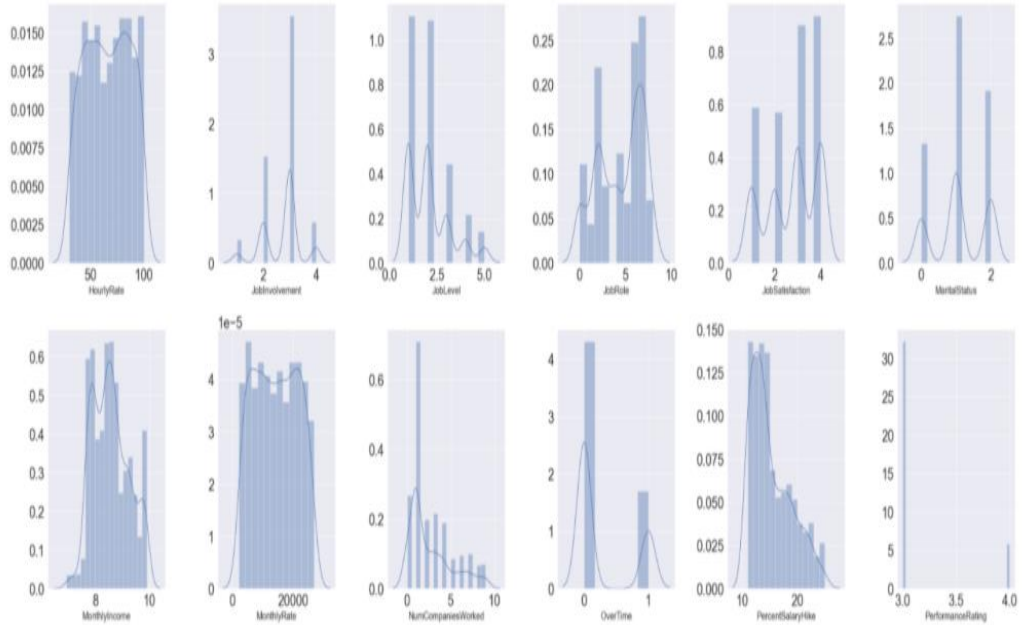
```
sm=SMOTE()
samp=SMOTE(0.75)
x_trains,y_trains=samp.fit_resample(x_train,y_train)
print("Before transformation",format(Counter(y_train)))
print("After Transformation",format(Counter(y_trains)))
```

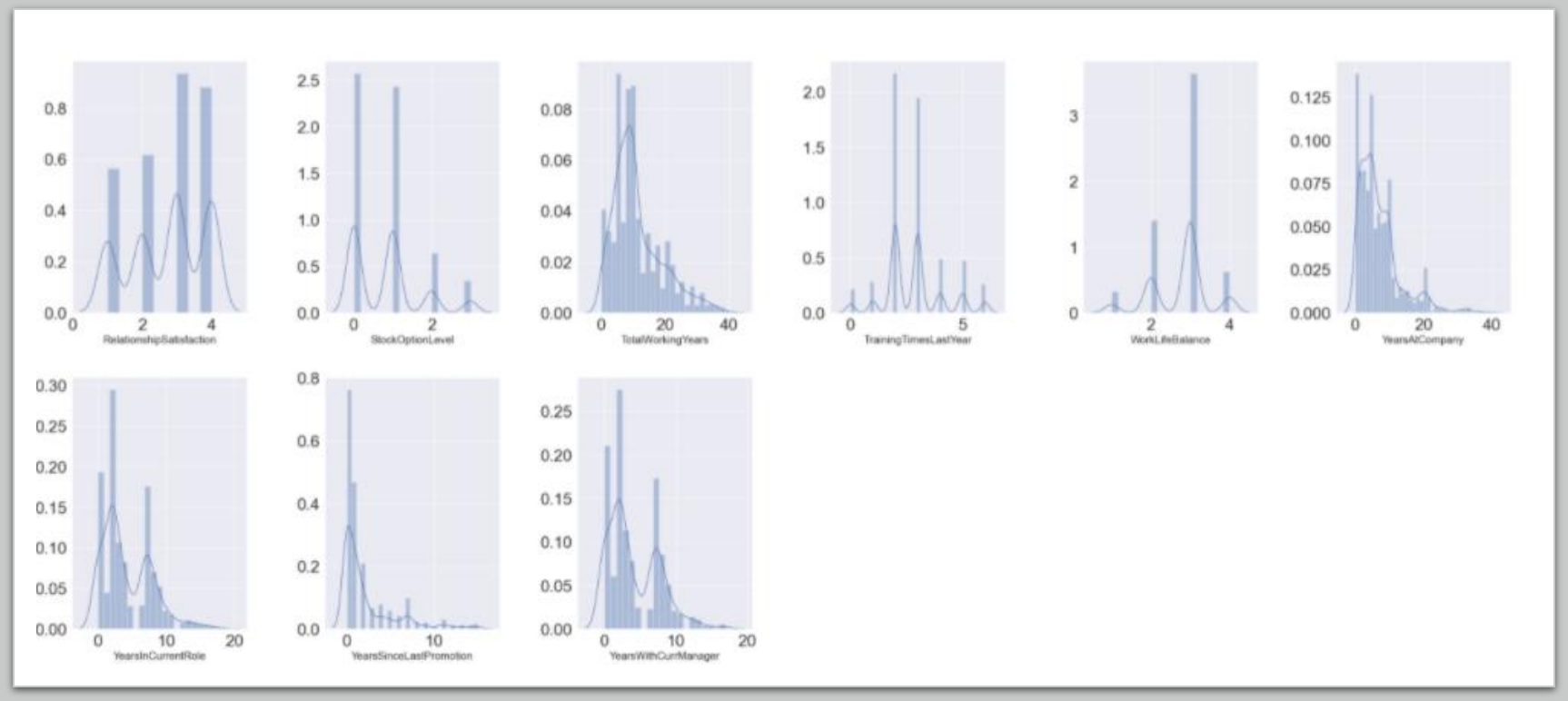
```
Before transformation Counter({0: 915, 1: 187})
After Transformation Counter({0: 915, 1: 686})
```

Having balanced the data , Data visualisation comes into picture , the data is checked upon swekness , and standard deviation , the following image gives complete details of the type of inputs in the features of the model ,

- The data looks like ,







Categorical columns as well regular data are good to go , but there is some skewness in a few columns which can be treated by outlier management techniques , like box plot .

- Now the problem is , after removing the outliers , the dataset is not at proper indexing , before any further analysis the indexing has to be done , so the method dataset_name.reset_index() will simply categorise the index , and the data set will be in proper shape , now the dataset is free from nulls , outliers , inbalanced output , hence we move towards the next phase that is , moving further , one need rename the columns so as to avoid error , form this dataset the unwanted columns need to be droped by the tool dataset_name.drop() , axis=0 will drop the row , and axis=1 , drops the columns , moving further we need to check the correlation between the columns , .corr() is one such method that is used to check the correlation , using a heatmap.

One can clearly see that there is correlation between different features , one of the column can be droped , which are highly correlated , so in this case , we analyse the heat map clearly and check the correlation between features whch are ploted here ,

```
plt.figure(figsize=(65,70))
sns.set(font_scale=4)
sns.heatmap(corr,annot=True,annot_kws={'size':24})
plt.show()
```

```
plt.scatter(data["JobLevel"],data["TotalWorkingYears"])
```

There are many correlations and we come up with conclusion that AGE , and Years with company can be droped without much loss of accuracy .
after dropping these two columns we are left with 31 features , hence concluding the data cleaning has been done , and we can play around with the data successfully , this is how the final data looks like .....

Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction
1	2	1102	2	0.000000	2	1	1	1	2
0	1	279	1	2.079442	1	1	1	2	3
1	2	1373	1	0.693147	2	4	1	4	4
0	1	1392	1	1.098612	4	1	1	5	4
0	1	1005	1	0.693147	2	1	1	8	4
0	2	1324	1	1.098612	3	3	1	10	3
0	2	1358	1	3.178054	1	1	1	11	4
0	1	216	1	3.135494	3	1	1	12	4
0	2	1299	1	3.295837	3	3	1	13	3
0	2	809	1	2.772589	3	3	1	14	1
0	2	153	1	2.708050	2	1	1	15	4
0	2	670	1	3.258097	1	1	1	16	1

now select the best features , from the given dataset , for this we can use Variance inflation factor or selectK best , feature selection techniques , it will give a score to every feature , and the output is an ascending series , of features which are best to predict the results , out of all 30 features , the select K best is applied to the dataset feature columns , and the results are stored in the new dataset that is used for further processing of model . The selectkbest technique gives following results ..

```
from sklearn.feature_selection import SelectKBest,f_classif
x=data.drop("Attrition",axis=1)
y=data.Attrition
```

```
best_features=SelectKBest(score_func=f_classif,k=25)
fit=best_features.fit(x,y)
df_scores=pd.DataFrame(fit.scores_)
df_columns=pd.DataFrame(x.columns)
feature_scores=pd.concat([df_columns,df_scores],axis=1)
feature_scores.columns=["Column_names","Scores_obtained"]
print(feature_scores.nlargest(25,"Scores_obtained"))
```

	Column_names	Scores_obtained
19	OverTime	81.939272
16	MonthlyIncome	59.115231
24	TotalWorkingYears	44.295072
12	JobLevel	42.638915
15	MaritalStatus	40.242881
27	YearsInCurrentRole	37.712574
23	StockOptionLevel	34.212702
29	YearsWithCurrManager	34.176343
11	JobInvolvement	23.683543
14	JobSatisfaction	15.812414
8	EnvironmentSatisfaction	12.983215
3	DistanceFromHome	10.126892
26	WorkLifeBalance	8.163499
2	Department	6.792431
13	JobRole	6.013881
1	DailyRate	5.239027
25	TrainingTimesLastYear	5.195271
22	RelationshipSatisfaction	3.580866
28	YearsSinceLastPromotion	1.643559
18	NumCompaniesWorked	1.311180
5	EducationField	1.199236
4	Education	0.908933
9	Gender	0.703151
20	PercentSalaryHike	0.515045
17	MonthlyRate	0.380775

Having done the the contents , in pipeline , the final data of features is kept in a new dataframe , and the new dataframe , is used to train the model on diff diff algorithms , the data cleaning , data visualising , data balancing and normalisation and hence the data selection is done at this point and we can use this data to fit in first algorithm that is KNN , KNEighbour Classifier , it is a supervised machine learning algorithm that uses the method of eucledian distance , to calculate the distance between test and train data and predict the results . It simply takes the data , and uses the law of similarity to get the classified result , it also has a drawback that I does not test , until he command is given , that is it is a lazy learner and a time consuming algorithm as well.

- This algorithm follows that similar thing exists in close proximity . Also this a versatile algorithm that is used for regression and classification both , also it is used to impute the missing values . Now the data is provided and is fitted intoo the knn algorithm , as shown below , after that the results are predicted and the clssification report of the same is printed , to check how accurate the results are ,

Here you can see , I have used classification report and has got 80 percent accuracy , , there is a confusion matrix , which is a special kind of matirx with actual values on x axis and predicted on y axis , it clearly depicts , 7 false positive values and 63 true negative value .

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn=KNeighborsClassifier()
from time import time
start=time()
knn.fit(x_train,y_train)
print("Training Time:",time()-start)
```

```
Training time: 0.016002655029296875
```

```
start=time()
y_pred=knn.predict(x_test)
print("Testing Time:",time()-start)
```

```
Testing Time: 0.16061758995056152
```

```
print(classification_report(y_test,y_pred))
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.98      0.89       290
           1       0.22      0.03      0.05        65

    accuracy                           0.80       355
   macro avg       0.52      0.50      0.47       355
weighted avg       0.71      0.80      0.74       355
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[283,   7],
       [ 63,   2]], dtype=int64)
```

Now comes a new variable that is cross val score , it is cross validation technique that is used to predict the results , by keeping a part of data for training athe other for testing , and continue the same process , so as to achieve cross validation scores , at last mean of all the value is printed , and I vert helpful as it ,in one line tells how much accutare the predicted results ar e, any value in plus minus 5% of accuracy means the data is over or underfitting the dat arespectively . For every algorith the cross val is reported so as to clearify the authencity of the algorithm so used . The cross val score comes out to be 82% , which is good enough to rely , and we can go for tuning of parametres , also called as Hyperparameter tuning .

```
cross_val_score(knn,x,y,cv=6).mean()
```

```
0.8251120408114615
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.99      0.89       290
           1       0.00      0.00      0.00        65

    accuracy                           0.81       355
   macro avg       0.41      0.49      0.45       355
weighted avg       0.67      0.81      0.73       355
```

After tuning the parameters , the accuracy score increased by 1 persent and hence the final result is reported , the tuning of the parameters is done by Grid Search cv or Random Search CV , where CV stands for cross validation ,this is a technique by which we can increase the accuracy , and also prove that the model is best fitted , in grid search CV the parameter range is provided all at random and the algorithm associated to it , is used , the best parameters are defiend , and using grid_search.best_params_ the best parameters are given by the compuuter , we simply fit those results into the algorithm , that is we ask the algorithm to train , based on the parameters provided . Grid search usesm large number of combinations(specified by us) and finds the combination of each and every combination provided by us , at last , it will give the best parameter which will give the best fitted results and hence we can inhance the accuracy of the model for hat particular algorithm , moving further their is Randomisez search CV , the only differentiatinthin line difference is , we don’t provide the parameters ,it picks up the combination on its own , and give the best fittind results , that is the combination that gives the best results , being haighly accurate , but ofcourse there is some limitation associated to every algorithm

The limitation is that , not every algorithm is best suited for a particular type of data different type of data , is fitted by different algorithm , and hence try other algorithms as well , so that we get the best results out of it . The next algorithm I used is Decision tree , it is also a supervised machine learning algorithm , that uses the method of making a decision by different different parameters , like the roots of a tree , in a particular directional flow .The decision tree uses nodes , it split a node into particular sub nodes , the tree starts with a root , called as nodes , and the sub roots , called as leaf nodes . A very limited portion of a tree is called a sub tree it is basically a reference , to know how things are actually working behind the picture .Decision tree , also undergoes Pruning , it is cutting down of big tree nodes to prevent the condition of overfitted results and run time also decrease . Lets fit the final data in this algorithm and check the classification report .

Having seen the classification report we can observe no overfitting of data is taking place,

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt=DecisionTreeClassifier()
```

```
dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier()
```

```
start=time()
dt.fit(x_train,y_train)
print("Training Time:",time()-start)
```

```
Training Time: 0.12801790237426758
```

```
start=time()
y_pred=dt.predict(x_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       290
           1       0.38      0.28      0.32        65

    accuracy                           0.79       355
   macro avg       0.62      0.59      0.60       355
weighted avg       0.76      0.79      0.77       355
```

```
cross_val_score(dt,x,y,cv=10).mean()
```

```
0.7700978923184497
```

```
# Data is not overfitting
```

For pruning in decision tree , the criterion used is Gini impurity , Entropy , within entropy there is selection criterion called Information gain . The formulae for entropy is summation –p*log2(p) .
Talking about gini entropy we , take the least value for reference .
There is something called maximum depth , which tells how much levels of a flowchart dree , must possess , we can limit the split further , by predefining the number (integer) ,there are terms called min samples leaf , min samples split which can always be predefined . Min samples leaf is minimum number of samples required to be a leaf node .
the model pruning is as shown,

```
dt=DecisionTreeClassifier(criterion='entropy',
 max_depth= 6,
 min_samples_leaf= 7,
 min_samples_split= 3)
```

```
start=time()
dt.fit(x_train,y_train)
print("Training Time:",time()-start)
```

```
Training Time: 0.03200387954711914
```

```
start=time()
y_pred=dt.predict(x_test)
print("Testing Time:",time()-start)
```

```
Testing Time: 0.00800180435180664
```

```
print(classification_report(y_pred,y_test))
```

```
print(classification_report(y_pred,y_test))
```

	precision	recall	f1-score	support
0	0.96	0.87	0.91	318
1	0.37	0.65	0.47	37
accuracy			0.85	355
macro avg	0.66	0.76	0.69	355
weighted avg	0.89	0.85	0.87	355

```
cfm(y_test,y_pred)
```

```
array([[277,  13],
       [ 41,  24]], dtype=int64)
```

Lets see AdaboostClassifier now , this again a supervised machine learning model , it is basically an ensemble method , used with decision tree having one single split , this theorem fits the estimator in the given dataset ,and the classification results are recorded , the wrong classified results are adjusted , a final decision is made by finding the weighted average , subsequently the average is calculated and classification s being done. The predicted model is as shown ,

```
from sklearn.ensemble import AdaBoostClassifier
```

```
ad=AdaBoostClassifier()
```

```
ad.fit(x_train,y_train)
```

```
AdaBoostClassifier()
```

```
y_pred=ad.predict(x_test)
```

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.88	0.95	0.92	290
1	0.67	0.45	0.54	65
accuracy			0.86	355
macro avg	0.78	0.70	0.73	355
weighted avg	0.85	0.86	0.85	355

```
cross_val_score(ad,x,y,cv=10).mean()
```

```
0.8709719308760364
```

Ada boost uses boosting technique, same process is followed here as well , a model is trained on a data , and then second model is trained which tries to correct the errors in the first model .

```
from sklearn.ensemble import BaggingClassifier
bag_cls=BaggingClassifier(dt,n_estimators=6,max_samples=0.6,bootstrap=True,random_state=4,oob_score=True)
```

```
ad.score(x_test,y_test)
```

```
0.8591549295774648
```

```
bag_cls.fit(x_train,y_train)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy',
                                                        max_depth=6,
                                                        min_samples_leaf=7,
                                                        min_samples_split=3),
                  max_samples=0.6, n_estimators=6, oob_score=True,
                  random_state=4)
```

```
bag_cls.score(x_test,y_test)
```

```
0.8450704225352113
```

This bagging classifier is a meta estimator that fits base classifier on random on random subset , it undergoes voting or averaging to do the final predictions .

Last but not the least comes , the gradient boosting classifier , this algorithm is very apt , and is very famous , as it simply focuses on reducing the error , or reducing the bias . We are free to mention the number of estimators , but the base estimator is Decision stump . The gradient boosted model is as here,

```
from sklearn.ensemble import GradientBoostingClassifier
gbdt=GradientBoostingClassifier()
```

```
gbdt.fit(x_train,y_train)
```

```
GradientBoostingClassifier()
```

```
y_pred=gbdt.predict(x_test)
```

```
cfm(y_test,y_pred)
```

```
array([[284,   6],
       [ 47,  18]], dtype=int64)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.98      0.91       290
           1       0.75      0.28      0.40        65

    accuracy                           0.85       355
   macro avg       0.80      0.63      0.66       355
weighted avg       0.84      0.85      0.82       355
```

Having done tuning in all the four algorithms , we can say that attrition prediction is done most apt by the gradient boosting decision tree , also the randomised search cv for all the algorithm is resulting the best classification report with 86% accuracy , now the conclusion is attrition in company is a boon , and should be as less as possible , based on all the features we can use this model to predict the attrition upto an accuracy rate of 86% , all the machine learning algorithms used different methods , and the techniques are different too , but the final result is the best fit for the attrition prediction and is not overfitting the data, that is the model is not biased , the final model is saved using pickle which can be picked directly and used as and when required . We can save the model as .....

```
# Hence the accuracy is 86% by GradientBoosting.
```

```
import pickle
```

```
final_model="attrition.pickle"
pickle.dump(gbdt,open(final_model,'wb'))
```

Apart from this there are many other methods , and techniques applied in the model to obtain this result , to stick to the point the model for attrition is ready to deploy and any accuracy change can be played over , by simply changing the parameters , for the tuning to increase the accuracy and low attrition in a company is highly desired , so using the graphics of data , which feature is resulting to high attrition can be interpreted , and hence company can treat that aspect , means work on it , which would ultimately yield good results , so model can be used for interpretaion purpose also , I end my discussion here , if you want more reference , and complete readme , feel free to contact me on tarushiagarwal23@gmail.com