Objective : Predicting Employee Attrition Using Machine Learning

Data Description

The dataset consists of 14999 obseravtions and 10 variables. Each row in dataset represents an employee; each column contains employee attributes:

- satisfaction_level (0–1)
- last_evaluation (Time since last evaluation in years)
- number_projects (Number of projects completed while at work)
- average_monthly_hours (Average monthly hours at workplace)
- time_spend_company (Time spent at the company in years)
- Work_accident (Whether the employee had a workplace accident)
- left (Whether the employee left the workplace or not (1 or 0))
- promotion_last_5years (Whether the employee was promoted in the last five years)
- sales (Department in which they work for)
- salary (Relative level of salary)

Approach

We perform turnover analysis project by using Python's Scikit-Learn library. We use Logistic Regression, Random Forest, and Support Vector Machine as classifier for employee attrition and measure the accuracy of models that are built.

Step 1 : Data Import and Preprocessing

```
#Import Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import re
import sys,traceback
```

```
'''Function to load the dataset'''
def data_init(data_filepath):
    try:
        hr = pd.read_csv(data_filepath,low_memory= False)

        col_list = list(hr)
```

```
        print("Loaded successfully.")

        return hr
    except:
        print("File Could not be loaded")
        print("Check your file or filepathname")
        return False



'''User interacive way to access the dataset'''
c = 1
while (c!=0):
    data_filepath = str(input("Enter data filepath:"))
    if os.path.isfile(data_filepath) :
        hr_data = data_init(data_filepath)
    else:
        '''Add double slash in filepath and try again!'''
        data_filepath = re.escape(data_filepath)
        hr_data = data_init(data_filepath)
    if type(hr_data) != str: c = 0
    else: print ("Check if file exists in the filepath and Let's try again ! \n")
```

```
Enter data filepath:/content/HR_comma_sep.csv
Loaded successfully.
```

```
#Import Data
hr = hr_data
col_names = hr.columns.tolist()
print("Column names:")
print(col_names)

print("\nSample data:")
hr.head()
```

```
Column names:
['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', '

Sample data:
```

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

```
#Display data type for each column
hr.dtypes
```

```
satisfaction_level       float64
last_evaluation          float64
number_project             int64
average_montly_hours       int64
time_spend_company         int64
Work_accident              int64
left                       int64
promotion_last_5years      int64
Department                object
salary                    object
dtype: object
```

```
#Check for Missing Values
hr.isnull().any()
```

```
satisfaction_level       False
last_evaluation          False
number_project           False
average_montly_hours     False
time_spend_company       False
Work_accident            False
left                     False
promotion_last_5years    False
Department               False
salary                   False
dtype: bool
```

The “left” column is the outcome variable recording 1 and 0. 1 for employees who left the company and 0 for those who didn’t.

```
#Dimensions of our dataset
hr.shape
```

```
(14999, 10)
```

```
#Summary for each variable
hr.describe()
```

satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spen
14999.000000	14999.000000	14999.000000	14999.000000	14!
0.612834	0.716102	3.803054	201.050337	
0.248631	0.171169	1.232592	49.943099	
0.090000	0.360000	2.000000	96.000000	
0.440000	0.560000	3.000000	156.000000	

The department column of the dataset has many categories and we need to reduce the categories for a better modeling. The department column has the following categories:

1.000000	1.000000	7.000000	310.000000

```
#To get the unique values for department
hr['Department'].unique()
```

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

Let us combine “technical”, “support” and “IT” these three together and call them “technical”.

```
#Combine "technical","support" and "IT" into one department
hr['Department']=np.where(hr['Department'] =='support', 'technical', hr['Department'])
hr['Department']=np.where(hr['Department'] =='IT', 'technical', hr['Department'])
```

After the change, this is how the department categories look:

```
#Print the updated values of departments
print(hr['Department'].unique())
```

```
['sales' 'accounting' 'hr' 'technical' 'management' 'product_mng'
 'marketing' 'RandD']
```

Data Exploration

Let us find out the number of employees who left the company and those who didn’t:

```
hr['left'].value_counts()
```

```
0    11428
1     3571
Name: left, dtype: int64
```

```
print((11428/(11428+3571))*100)
```

```
76.19174611640777
```

We observe that 11428 employees left the company, which is 76.19 per cent of the total employees in the organisation.

```
hr.groupby('left').mean()
```

left	satisfaction_level	last_evaluation	number_project	average_montly_hours	tim
0	0.666810	0.715473	3.786664	199.060203	
1	0.440098	0.718113	3.855503	207.419210	

Several observations:

- The average satisfaction level of employees who stayed with the company is higher than that of the employees who left.
- The average monthly work hours of employees who left the company is more than that of the employees who stayed.
- The employees who had workplace accidents are less likely to leave than that of the employee who did not have workplace accidents.
- The employees who were promoted in the last five years are less likely to leave than those who did not get a promotion in the last five years.

We can calculate categorical means for categorical variables such as department and salary to get a more detailed sense of our data like so:

```
hr.groupby('Department').mean()
```

	satisfaction_level	last_evaluation	number_project	average_montly_ho
Department				

```
hr.groupby('salary').mean()
```

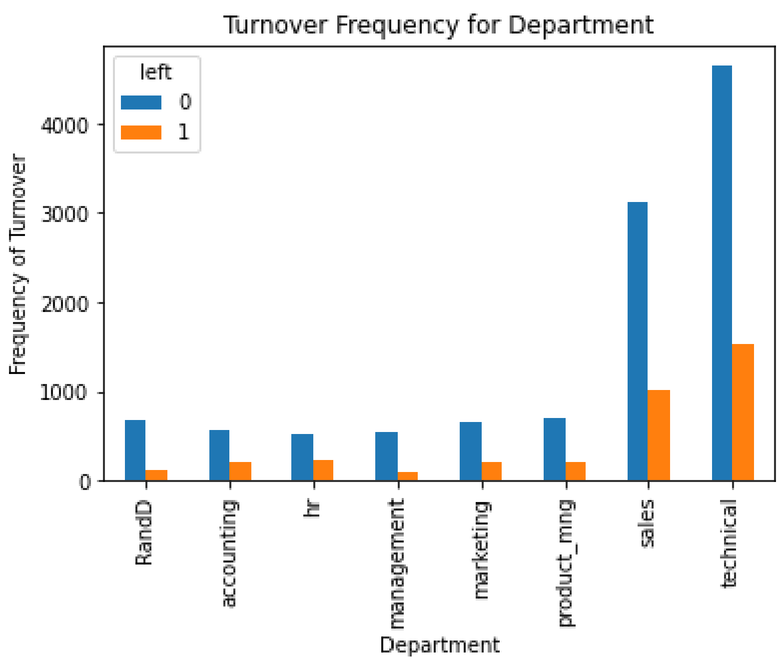
	satisfaction_level	last_evaluation	number_project	average_montly_hours
salary				
high	0.637470	0.704325	3.767179	199.867421
low	0.600753	0.717017	3.799891	200.996583
medium	0.621817	0.717322	3.813528	201.338349

Data Visualization

Let us visualize our data to get a much clearer picture of the data and the significant features.

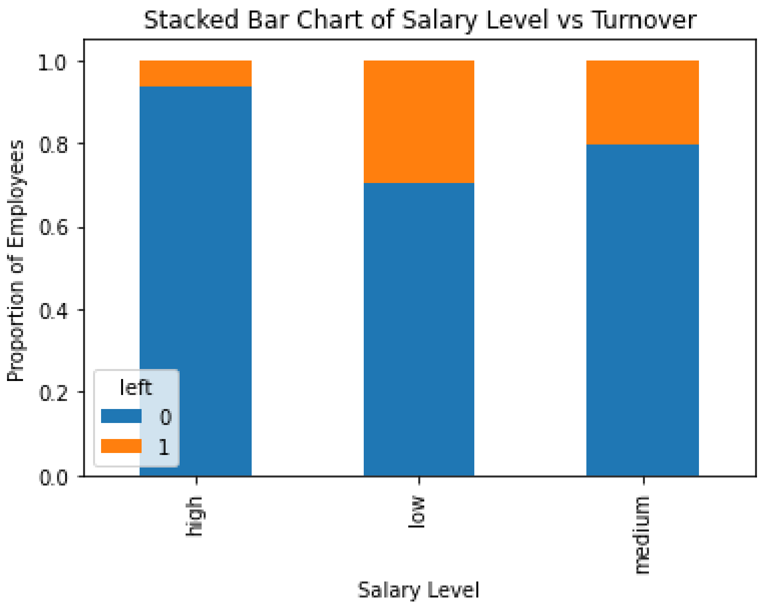
```
%matplotlib inline

#Bar chart for department employee work for and the frequency of turnover
pd.crosstab(hr.Department,hr.left).plot(kind='bar')
plt.title('Turnover Frequency for Department')
plt.xlabel('Department')
plt.ylabel('Frequency of Turnover')
plt.savefig('department_bar_chart')
```



It is evident that the frequency of employee turnover depends a great deal on the department they work for. Thus, department can be a good predictor of the outcome variable.

```
#Bar chart for employee salary level and the frequency of turnover
table=pd.crosstab(hr.salary, hr.left)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
plt.title('Stacked Bar Chart of Salary Level vs Turnover')
plt.xlabel('Salary Level')
plt.ylabel('Proportion of Employees')
plt.savefig('salary_bar_chart')
```

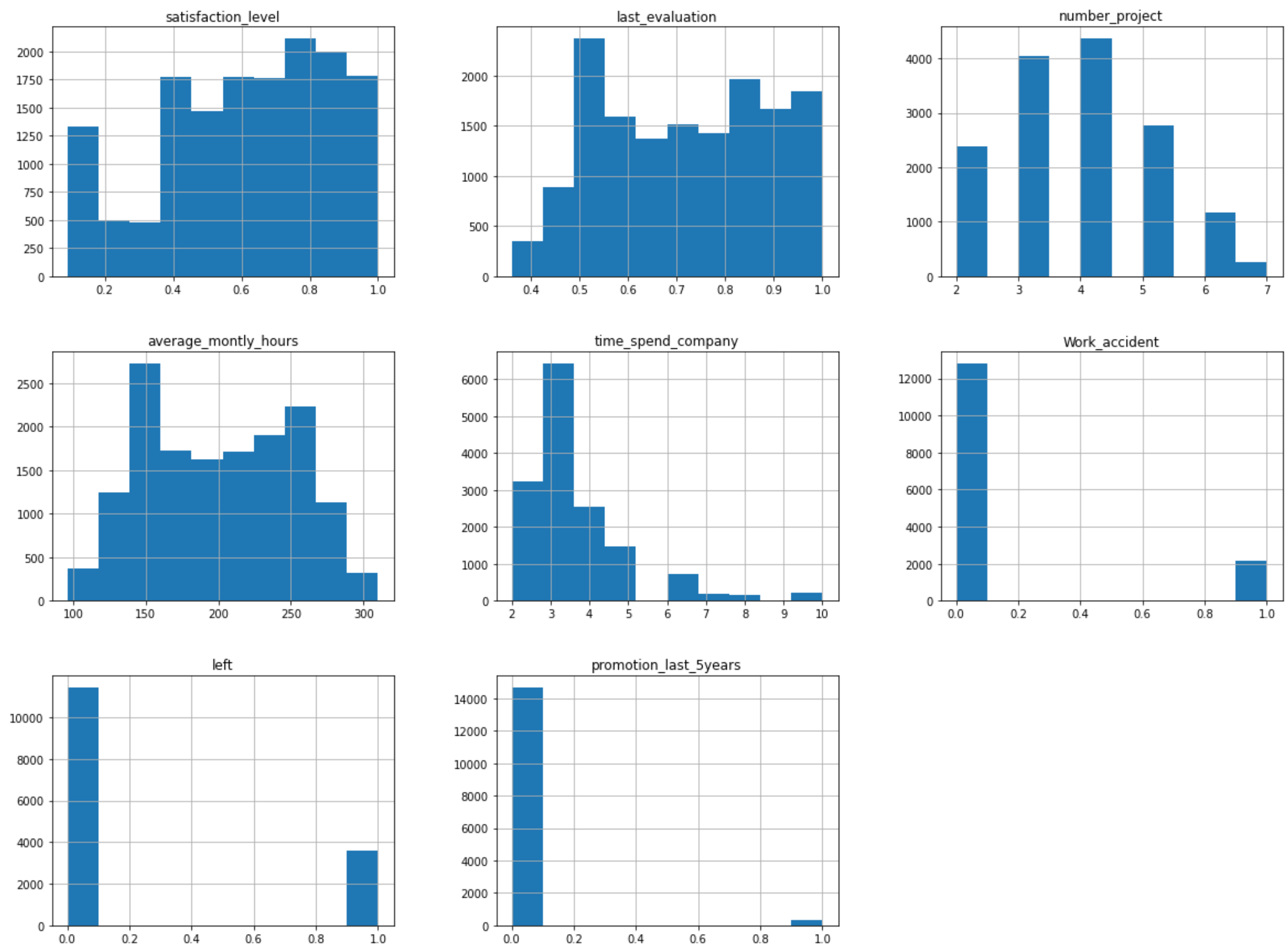


The proportion of the employee turnover depends a great deal on their salary level; hence, salary level can be a good predictor in predicting the outcome.

```
#Proportion of employees left by department
pd.crosstab(hr.Department, hr.left)
```

```
#Histogram of numeric variables
num_bins = 10

hr.hist(bins=num_bins, figsize=(20,15))
plt.savefig("hr_histogram_plots")
plt.show()
```



Create Dummy Variable for Categorical Variable

There are two categorical variables in the dataset and they need to be converted to dummy variables before they can be used for modelling.

```
hr.head()
```

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

```
cat_vars=['Department','salary']
for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(hr[var], prefix=var)
    hr1=hr.join(cat_list)
    hr=hr1
```

We drop the actual categorical variables once the dummy variables have been created.

```
hr.drop(hr.columns[[8, 9]], axis=1, inplace=True)
```

Column names after creating dummy variables for categorical variables:

```
hr.columns.values
```

```
array(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident',
       'left', 'promotion_last_5years', 'Department_RandD',
       'Department_accounting', 'Department_hr', 'Department_management',
       'Department_marketing', 'Department_product_mng',
       'Department_sales', 'Department_technical', 'salary_high',
       'salary_low', 'salary_medium'], dtype=object)
```

```
hr_vars=hr.columns.values.tolist()
y=['left']
X=[i for i in hr_vars if i not in y]
```

The response variable is “left”, and all the other variables are predictors.

```
X
```

```
['satisfaction_level',
 'last_evaluation',
 'number_project',
 'average_montly_hours',
 'time_spend_company',
 'Work_accident',
 'promotion_last_5years',
 'Department_RandD',
 'Department_accounting',
 'Department_hr',
 'Department_management',
 'Department_marketing',
 'Department_product_mng',
 'Department_sales',
 'Department_technical',
 'salary_high',
 'salary_low',
 'salary_medium']
```

Feature Selection

The Recursive Feature Elimination (RFE) works by recursively removing variables and building a model on those variables that remain. It uses the model accuracy to identify which variables (and combination of variables) contribute the most to predicting the target attribute.

Let’s use feature selection to help us decide which variables are significant that can predict employee turnover with great accuracy. There are total 18 columns in X.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

#Recursive Feature Elimination (RFE)
model = LogisticRegression()

rfe = RFE(model)
rfe = rfe.fit(hr[X], hr[y])
print(rfe.support_)
print(rfe.ranking_)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversi
  y = column_or_1d(y, warn=True)
[ True False False False  True  True  True  True False  True  True False
 False False False  True  True False]
[ 1  2  4 10  1  1  1  1  6  1  1  7  9  8  5  1  1  3]
```

We can see that RFE chose the variables for us, which are marked True in the support_ array and marked with a choice “1” in the ranking_array.

```
cols=['satisfaction_level', 'last_evaluation', 'time_spend_company', 'Work_accident', 'pro
      'Department_RandD', 'Department_hr', 'Department_management', 'salary_high', 'salary
X=hr[cols]
y=hr['left']
```

Logistic Regression Model

```
#Split data into training and test samples
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
#Logistic Regression Classifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
from sklearn.metrics import accuracy_score
print('Logistic regression accuracy: {:.3f}'.format(accuracy_score(y_test, logreg.predict(
```

```
Logistic regression accuracy: 0.771
```

Random Forest

```
#Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
print('Random Forest Accuracy: {:.3f}'.format(accuracy_score(y_test, rf.predict(X_test))))
```

```
Random Forest Accuracy: 0.978
```

Support Vector Machine

```
#SVM Classifier
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train, y_train)
```

```
SVC()
```

```
print('Support vector machine accuracy: {:.3f}'.format(accuracy_score(y_test, svc.predict(
```

```
Support vector machine accuracy: 0.907
```

Out of the three models, Random Forest has the best performance. We will perform 10-fold cross validation to confirm our results.

10 Fold Cross Validation

Cross validation attempts to avoid overfitting while still producing a prediction for each observation dataset. We are using 10-fold Cross-Validation to train our Random Forest and SVM model.

```
#For Random Forest
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
kfold = model_selection.KFold(n_splits=10)
modelCV = RandomForestClassifier()
scoring = 'accuracy'
results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=sco
print("10-fold cross validation average accuracy for Random Forest Classifier: %.3f" % (re
```

```
10-fold cross validation average accuracy for Random Forest Classifier: 0.982
```

```
#For SVM
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
kfold = model_selection.KFold(n_splits=10)
modelCV = SVC()
scoring = 'accuracy'
results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=scc
print("10-fold cross validation average accuracy for SVM Classifier: %.3f" % (results.mear
```

```
10-fold cross validation average accuracy for SVM Classifier: 0.906
```

From the CV results we observe that the average accuracy remains very close to the Random Forest & SVM model accuracy; hence, we can conclude that the models generalize well.

Precision and Recall

We construct confusion matrix to visualize predictions made by a classifier and evaluate the accuracy of a classification.

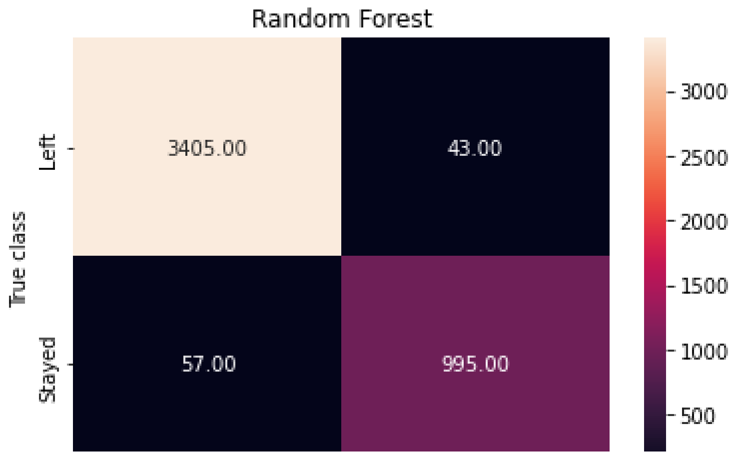
```
#Precison Recall Scores for Random Forest
from sklearn.metrics import classification_report
print(classification_report(y_test, rf.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      3462
           1       0.95      0.96      0.95      1038

    accuracy                           0.98      4500
   macro avg       0.97      0.97      0.97      4500
weighted avg       0.98      0.98      0.98      4500
```

```
#Confusion Matrix for Random Forest
y_pred = rf.predict(X_test)
from sklearn.metrics import confusion_matrix
import seaborn as sns
classes=[0,1]
forest_cm = metrics.confusion_matrix(y_pred, y_test,labels=classes)
sns.heatmap(forest_cm, annot=True, fmt='.2f',xticklabels = ["Left", "Stayed"] , yticklabel
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Random Forest')
plt.savefig('random_forest')
```



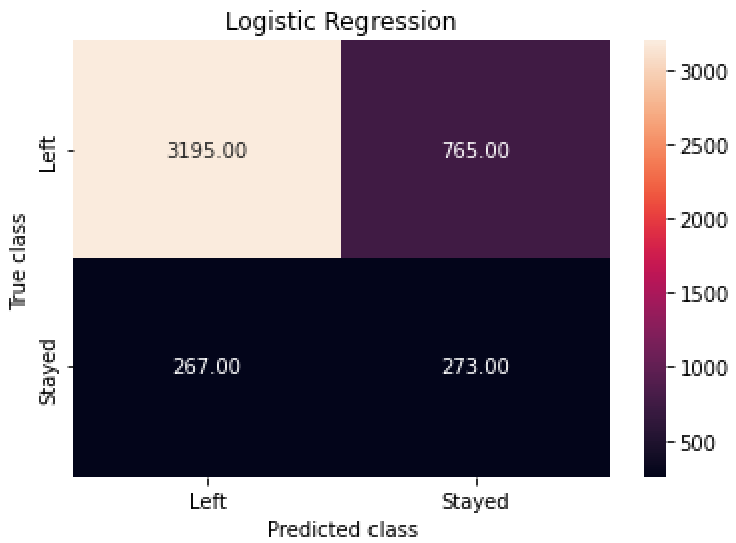

```
#PRScores for Logistic Regression
print(classification_report(y_test, logreg.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.92      0.86      3462
           1       0.51      0.26      0.35      1038

    accuracy                           0.77      4500
   macro avg       0.66      0.59      0.60      4500
weighted avg       0.74      0.77      0.74      4500
```

```
#Confusion Matrix for Logistic Regression
logreg_y_pred = logreg.predict(X_test)
logreg_cm = metrics.confusion_matrix(logreg_y_pred, y_test,labels=classes)
sns.heatmap(logreg_cm, annot=True, fmt='.2f',xticklabels = ["Left", "Stayed"] , yticklabel
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Logistic Regression')
plt.savefig('logistic_regression')
```



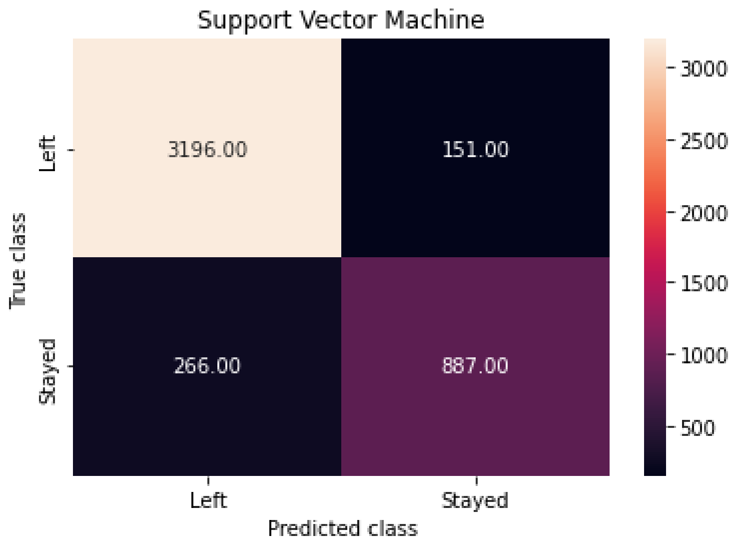

```
#PR scores for SVM
print(classification_report(y_test, svc.predict(X_test)))
```

```
              precision    recall  f1-score   support
```

```
             0       0.95      0.92      0.94      3462
             1       0.77      0.85      0.81      1038

      accuracy                           0.91      4500
     macro avg       0.86      0.89      0.87      4500
  weighted avg       0.91      0.91      0.91      4500
```

```
#Confusion Matrix for SVM
svc_y_pred = svc.predict(X_test)
svc_cm = metrics.confusion_matrix(svc_y_pred, y_test,labels=classes)
sns.heatmap(svc_cm, annot=True, fmt='.2f',xticklabels = ["Left", "Stayed"] , yticklabels =
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Support Vector Machine')
plt.savefig('support_vector_machine')
```



ROC Curve

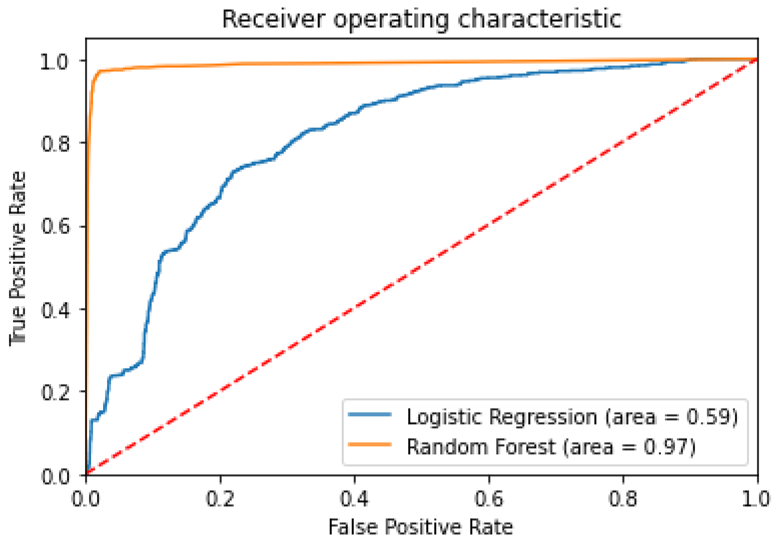
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

#ROC for logistic regression
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])

#ROC for Random Forrest
rf_roc_auc = roc_auc_score(y_test, rf.predict(X_test))
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, rf.predict_proba(X_test)[:,1])

#ROC Curve for Random Forest & Logistic Regression
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(rf_fpr, rf_tpr, label='Random Forest (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('ROC')
plt.show()
```



The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (toward the top-left corner)

Variable Imporatnce for Random Forest Classifier

According to our Random Forest model, the the most important features which influence whether to leave the company, in ascending order are as follows:

```
feature_labels = np.array(['satisfaction_level', 'last_evaluation', 'time_spend_company',
      'Department_RandD', 'Department_hr', 'Department_management', 'salary_high', 'salary
importance = rf.feature_importances_
feature_indexes_by_importance = importance.argsort()
analysis_result=''
for index in feature_indexes_by_importance:
    print('{}-{:.2f}%'.format(feature_labels[index], (importance[index] *100.0)))
    analysis_result += ('{}-{:.2f}%'.format(feature_labels[index], (importance[index] *10(
```

```
Department_management-0.26%
Department_hr-0.26%
promotion_last_5years-0.27%
Department_RandD-0.31%
salary_high-0.70%
salary_low-1.26%
Work_accident-1.59%
```

```
last_evaluation-19.43%
time_spend_company-26.16%
satisfaction_level-49.77%
```

```
file = open("variable_importance.txt","w+")
file.write(analysis_result)
file.close()
```

Results & Conclusion

Random Forest is the best classfier for predicting employee attrition for our dataset. Some of the most important factors on which employee attrition depends are

- Satisfaction Level
- Tenure with organisation
- Time since last evaluation
- Work Accident
- Salary
- Department
- Career Advancement (If Promoted in last five years or not)