# CO 544- MACHINE LEARNING & DATA MINING PROJECT REPORT

**GROUP 04**
**E/15/065 (DE SILVA K.G.P.M.)**
**E/15/076 (DILEKA J.H.S.)**
**E/15/220 (MALITHTHA K.H.H.)**

**FACULTY OF ENGINEERING**
**UNIVERSITY OF PERADENIYA**

## CONTENTS

## 1.INTRODUCTION

Machine learning and Data mining can aid in solving problems which may include a large amount of heterogeneous data. In this project we explored the application of machine learning and data mining to solve a classification problem.

A classification problem is when the output variable is a category,In here there are two categories “success” and “failure”. A classification model attempts to draw some conclusions from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes.

Classification is an important method in today’s world, where big data is used to make all kinds of decisions in government, economics, medicine, and more. Researchers have access to huge amounts of data, and classification is one tool that helps them to make sense of the data and find patterns

In this project, we used a number of different supervised algorithms to precisely predict the success or failure of the input .Then we choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data.
Our goal with this implementation is to build a model that accurately predicts whether the class attribute is success or failure.

## 2.DATA

In order to maintain the confidentiality, the attribute names have been altered. The attributes can be continuous, nominal with small numbers of values, and nominal with larger numbers of values.

### 2.1 Feature Explanation

This dataset has 15 feature attributes and one class attribute.

Feature attributes with the values they can have:

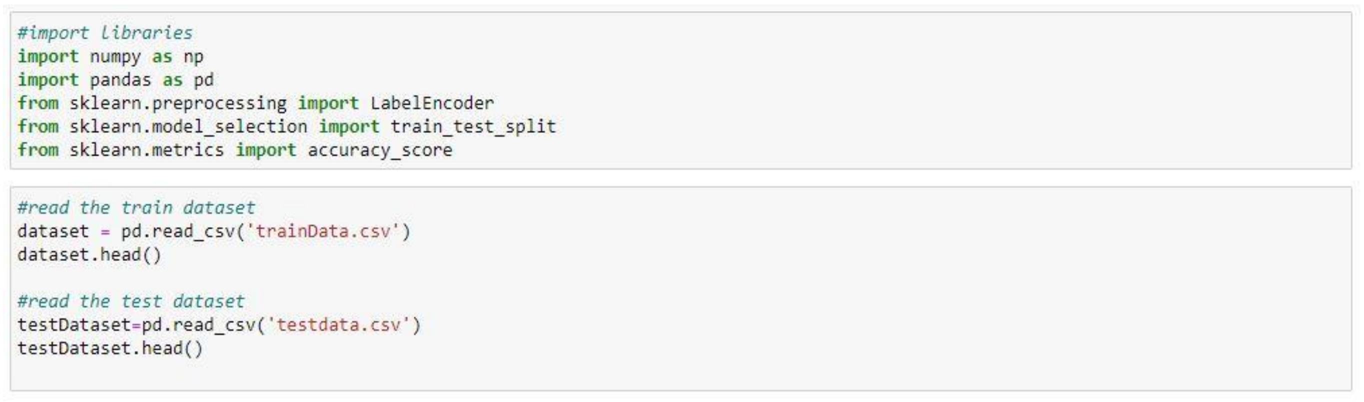
A1: b, a
A2: continuous
A3: u, y, l
A4: g, p, gg
A5: continuous
A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff
A7: continuous
A8: TRUE, FALSE
A9: v, h, bb, j, n, z, dd, ff, o
A10: continuous
A11: TRUE, FALSE
A12: continuous
A13: TRUE, FALSE
A14: continuous
A15: g, p, s

Target attribute (class attribute) : A16 {Success,Failure(class attribute)}

## 3.PROCEDURE

### 3.1 Import Libraries and Load data

We first loaded the Python libraries that we used, as well as the dataset. The last column is our target attribute ‘A16’, and the rest are the other 15 attributes from ‘A1’ to ‘A15’.

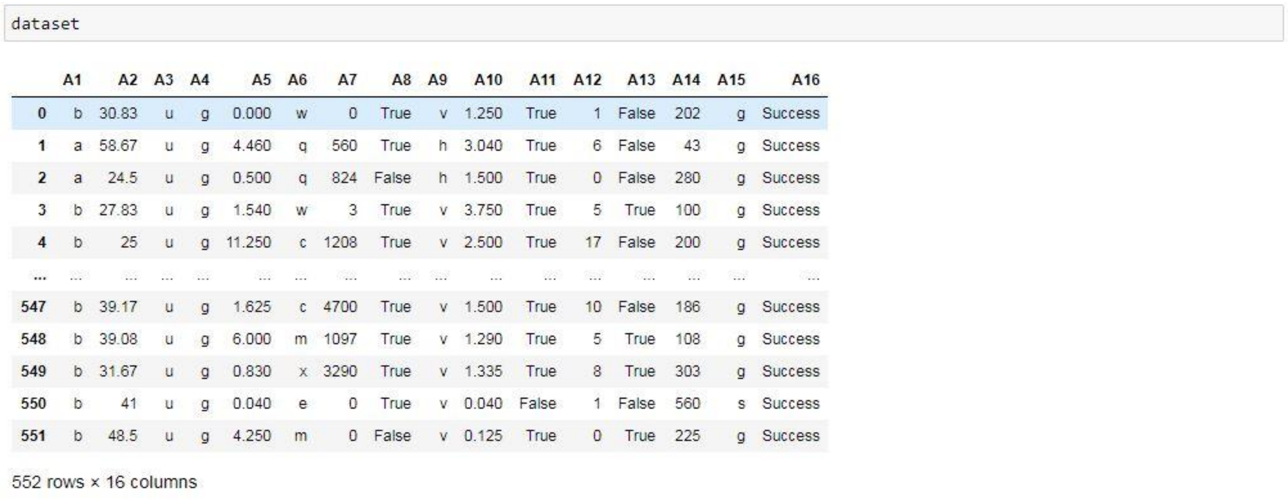


```
#import libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
#read the train dataset
dataset = pd.read_csv('trainData.csv')
dataset.head()

#read the test dataset
testDataset=pd.read_csv('testdata.csv')
testDataset.head()
```

Here by using pandas library, we read comma-separated values (CSVs) files into DataFrames.

Training dataset



```
dataset
```

| | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | b | 30.83 | u | g | 0.000 | w | 0 | True | v | 1.250 | True | 1 | False | 202 | g | Success |
| **1** | a | 58.67 | u | g | 4.460 | q | 560 | True | h | 3.040 | True | 6 | False | 43 | g | Success |
| **2** | a | 24.5 | u | g | 0.500 | q | 824 | False | h | 1.500 | True | 0 | False | 280 | g | Success |
| **3** | b | 27.83 | u | g | 1.540 | w | 3 | True | v | 3.750 | True | 5 | True | 100 | g | Success |
| **4** | b | 25 | u | g | 11.250 | c | 1208 | True | v | 2.500 | True | 17 | False | 200 | g | Success |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **547** | b | 39.17 | u | g | 1.625 | c | 4700 | True | v | 1.500 | True | 10 | False | 186 | g | Success |
| **548** | b | 39.08 | u | g | 6.000 | m | 1097 | True | v | 1.290 | True | 5 | True | 108 | g | Success |
| **549** | b | 31.67 | u | g | 0.830 | x | 3290 | True | v | 1.335 | True | 8 | True | 303 | g | Success |
| **550** | b | 41 | u | g | 0.040 | e | 0 | True | v | 0.040 | False | 1 | False | 560 | s | Success |
| **551** | b | 48.5 | u | g | 4.250 | m | 0 | False | v | 0.125 | True | 0 | True | 225 | g | Success |

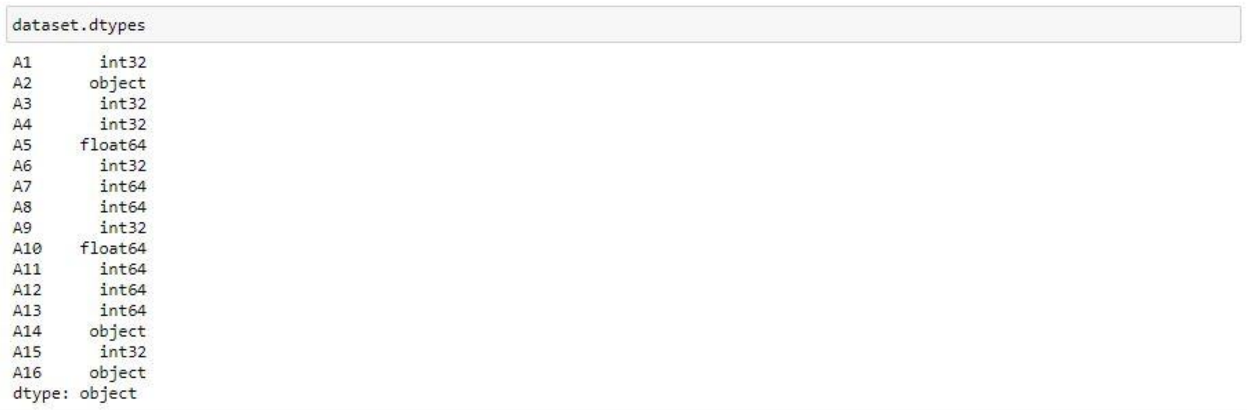
552 rows × 16 columns

## 3.2 Preprocessing the data.

Data must be preprocessed in order to be used in Machine Learning algorithms. This preprocessing phase includes the cleaning, formatting and restructuring of the data.

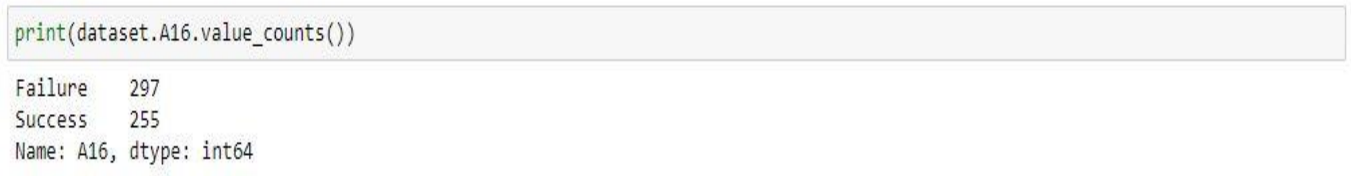
### 3.2.1 Exploratory Data Analysis (EDA)

An initial exploration of the dataset show us how many successes and failures and the types of the attributes in the training dataset.



```
dataset.dtypes
```

```
A1       int32
A2      object
A3       int32
A4       int32
A5     float64
A6       int32
A7       int64
A8       int64
A9       int32
A10    float64
A11      int64
A12      int64
A13      int64
A14     object
A15      int32
A16     object
dtype: object
```



```
print(dataset.A16.value_counts())
```

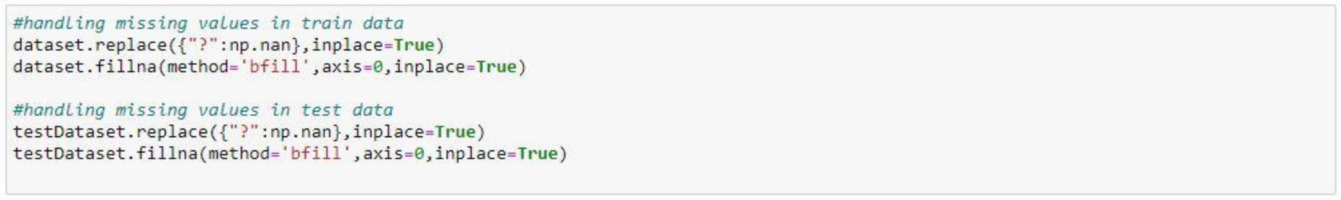
```
Failure    297
Success    255
Name: A16, dtype: int64
```

### 3.2.2 Handling the missing values

Some algorithms can factor in the missing values and learn the best imputation values for the missing data based on the training loss reduction. Some others have the option to just ignore them . However, other algorithms will panic and throw an error complaining about the missing values (ie. Scikit learn — LogisticRegression). In that case, we need to handle the missing data and clean it before feeding it to the algorithm.

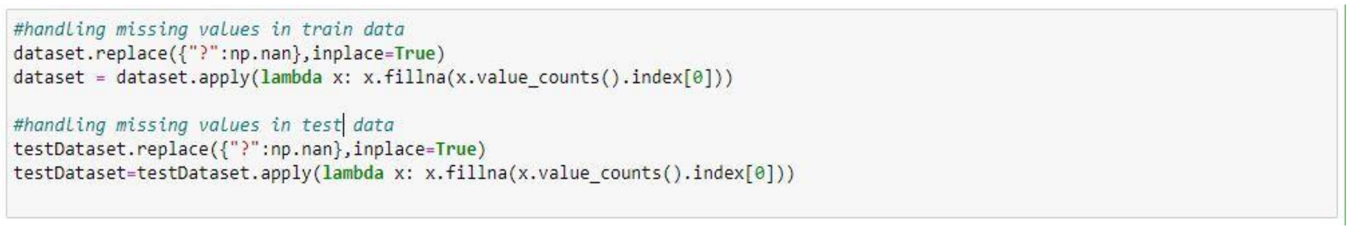
In our dataset, there are some missing values. Before training the data , we have to replace the missing values with some appropriate values. In this project we tried different ways to handle these missing values.

First we replaced the missing values with **backward fill or ‘bfill’**. It filled the NaN values with the previous non-null value. But the problem was if a previous or next value was also a NaN value, then, the NaN remained even after back-filling.



```
#handling missing values in train data
dataset.replace({"?":np.nan},inplace=True)
dataset.fillna(method='bfill',axis=0,inplace=True)

#handling missing values in test data
testDataset.replace({"?":np.nan},inplace=True)
testDataset.fillna(method='bfill',axis=0,inplace=True)
```

Then, we used the **Most Frequent** statistical strategy to impute missing values by replacing missing data with the most frequent values within each column.



```
#handling missing values in train data
dataset.replace({"?":np.nan},inplace=True)
dataset = dataset.apply(lambda x: x.fillna(x.value_counts().index[0]))

#handling missing values in test data
testDataset.replace({"?":np.nan},inplace=True)
testDataset=testDataset.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

Finally we used the **Most frequent** strategy to replace the missing values as it works well with categorical features (strings and numerical representations).

### 3.2.3 Preprocessing Categorical Features (Converting nominal attributes to numerical attributes )

If we take a look at the features explanation, we can see that there are some features like ‘A1’ ,’A3’ and etc  that are not numerical, they are nominal. Machine learning algorithms expect to work with numerical values, so these nominal features should be transformed.

From here we described how our two final models(which we selected as final in kaggle) were trained. We handled the missing values for these two models using most frequent technique.

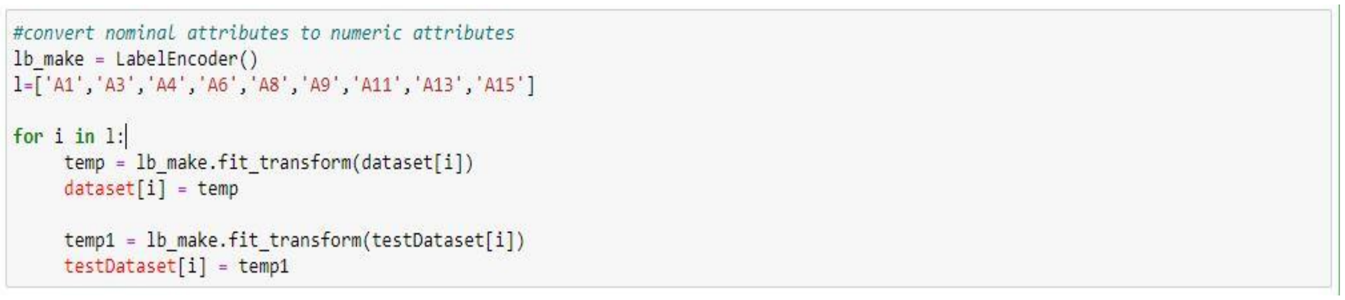
**Method 1**

Here our approach to encoding categorical values was to use a technique called label encoding. Label encoding is simply converting each value in a column to a number. For example, the A1 column contains 2 different values. We could choose to encode it like this:

a $\rightarrow$ 0

b $\rightarrow$ 1

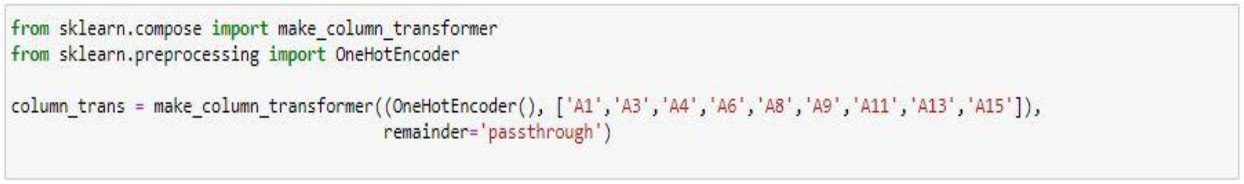
After label encoding numerical labels are always between 0 and number_of_categories -1.



```
#convert nominal attributes to numeric attributes
lb_make = LabelEncoder()
l=['A1','A3','A4','A6','A8','A9','A11','A13','A15']

for i in l:
     temp = lb_make.fit_transform(dataset[i])
     dataset[i] = temp

     temp1 = lb_make.fit_transform(testDataset[i])
     testDataset[i] = temp1
```

**Method 2**

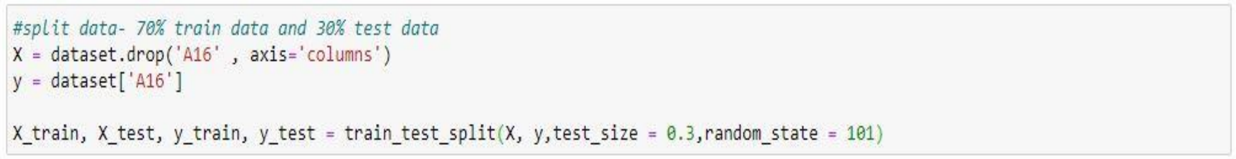
Here we ended up with categorical values using a method called OneHotEnconder. We used make_column_transfomer to specify which kind of transformation we are using. By specifying remainder='passthrough', all remaining columns that were not specified in OneHotEnconder transformers are automatically passed through. This subset of columns is concatenated with the output of the transformers.



```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder

column_trans = make_column_transformer((OneHotEncoder(), ['A1','A3','A4','A6','A8','A9','A11','A13','A15']),
                                       remainder='passthrough')
```

## 3.2.4 Shuffle and Split Data

When all categorical variables are transformed, we need to split our data into training and test sets. We used 70% of the data for training and 30% for testing. For Method 1 and Method 2 we used the same below code to split data.

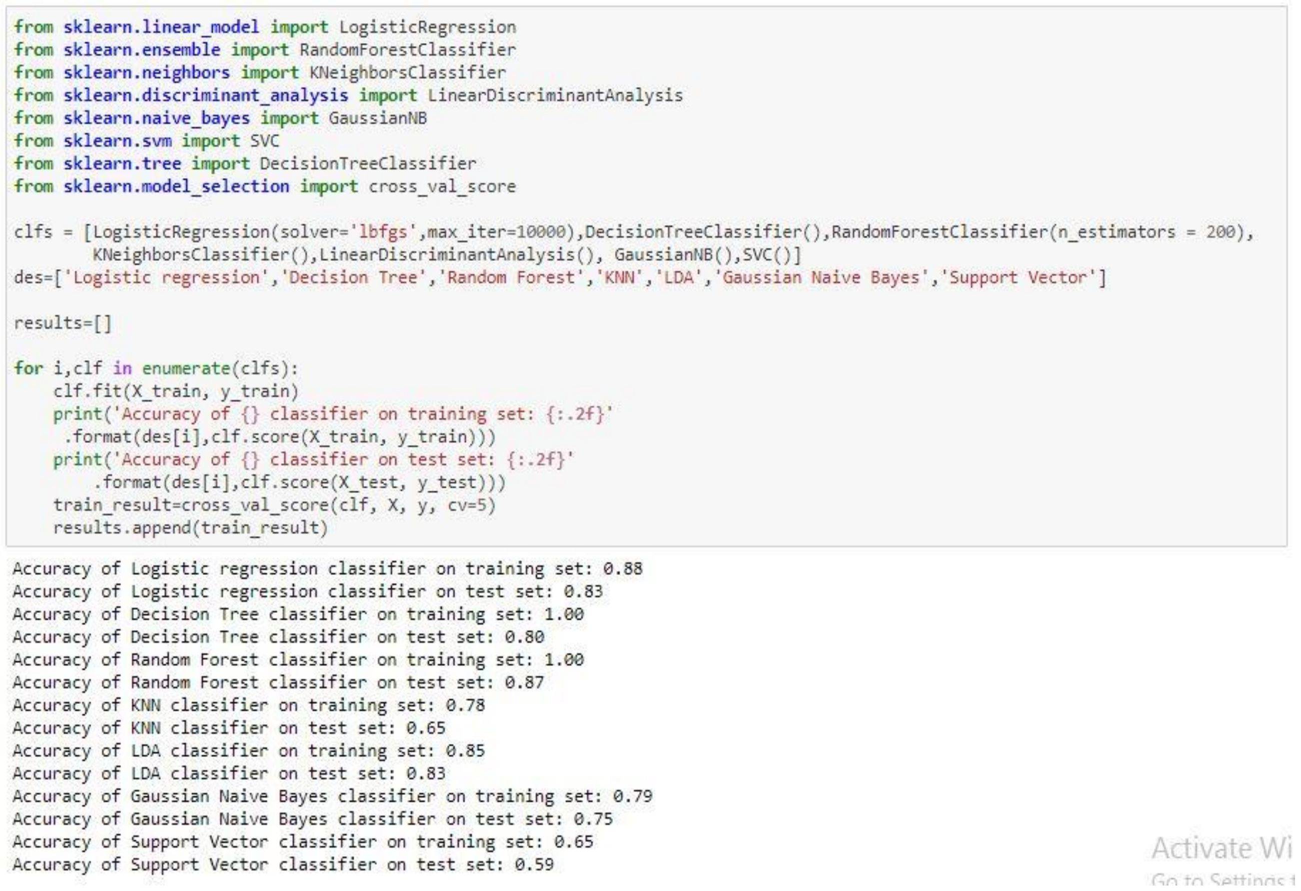


```
#split data- 70% train data and 30% test data
X = dataset.drop('A16' , axis='columns')
y = dataset['A16']

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.3,random_state = 101)
```

## 3.3 Model’s Performance Evaluation

We used some different algorithms, and determined the best at modeling and predicting our data.

We checked the accuracy of the algorithms using training data and test data  to select the best algorithm for the base classifier.

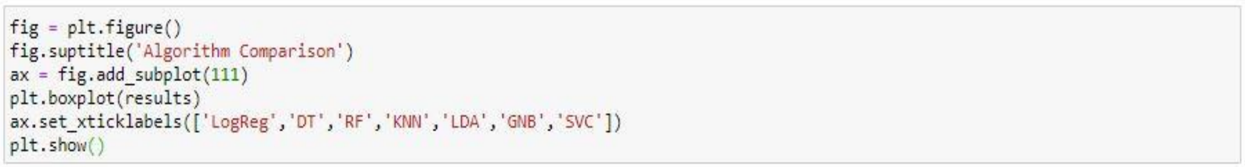


```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

clfs = [LogisticRegression(solver='lbfgs',max_iter=10000),DecisionTreeClassifier(),RandomForestClassifier(n_estimators = 200),
        KNeighborsClassifier(),LinearDiscriminantAnalysis(), GaussianNB(),SVC()]
des=['Logistic regression','Decision Tree','Random Forest','KNN','LDA','Gaussian Naive Bayes','Support Vector']

results=[]

for i,clf in enumerate(clfs):
    clf.fit(X_train, y_train)
    print('Accuracy of {} classifier on training set: {:.2f}'
     .format(des[i],clf.score(X_train, y_train)))
    print('Accuracy of {} classifier on test set: {:.2f}'
        .format(des[i],clf.score(X_test, y_test)))
    train_result=cross_val_score(clf, X, y, cv=5)
    results.append(train_result)
```

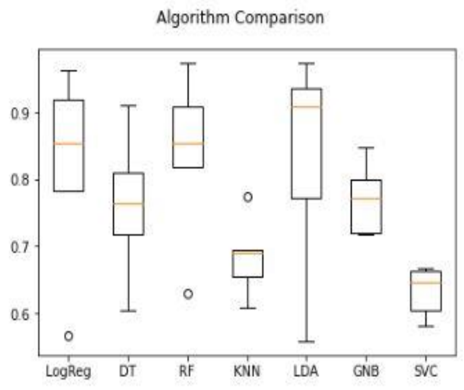
```
Accuracy of Logistic regression classifier on training set: 0.88
Accuracy of Logistic regression classifier on test set: 0.83
Accuracy of Decision Tree classifier on training set: 1.00
Accuracy of Decision Tree classifier on test set: 0.80
Accuracy of Random Forest classifier on training set: 1.00
Accuracy of Random Forest classifier on test set: 0.87
Accuracy of KNN classifier on training set: 0.78
Accuracy of KNN classifier on test set: 0.65
Accuracy of LDA classifier on training set: 0.85
Accuracy of LDA classifier on test set: 0.83
Accuracy of Gaussian Naive Bayes classifier on training set: 0.79
Accuracy of Gaussian Naive Bayes classifier on test set: 0.75
Accuracy of Support Vector classifier on training set: 0.65
Accuracy of Support Vector classifier on test set: 0.59
```

These are the algorithms we used in above code.

1. Logistic regression
2. Decision tree
3. Random forest
4. KNN(K-Nearest Neighbors)
5. LDA(Linear Discriminant Analysis)
6. Gaussian Naive Bayes
7. Support Vector Classifier



```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(['LogReg','DT','RF','KNN','LDA','GNB','SVC'])
plt.show()
```

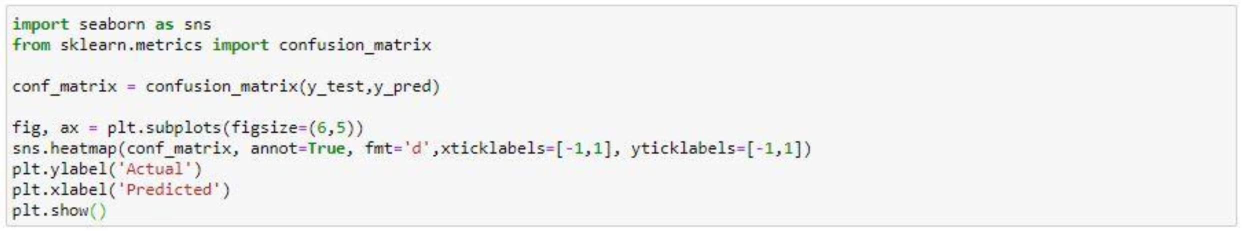


According to the above results we recognized that  Random Forest Classifier was best among other classifiers. Because it showed 100% accuracy with train data and 87% accuracy with test data.

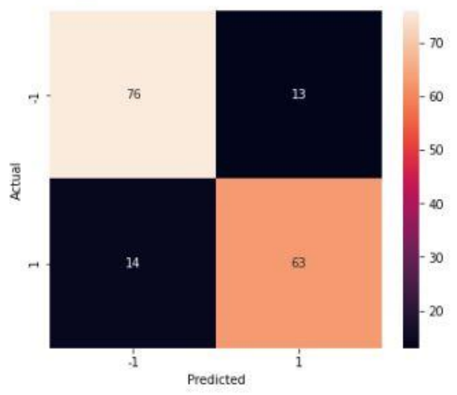
## 3.4 Initial Model

**Method 1:**
So as the initial model we chose the Random Forest classifier.
Random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Here we used 200 trees in the forest and the sub-sample size was always the same as the original input sample size.

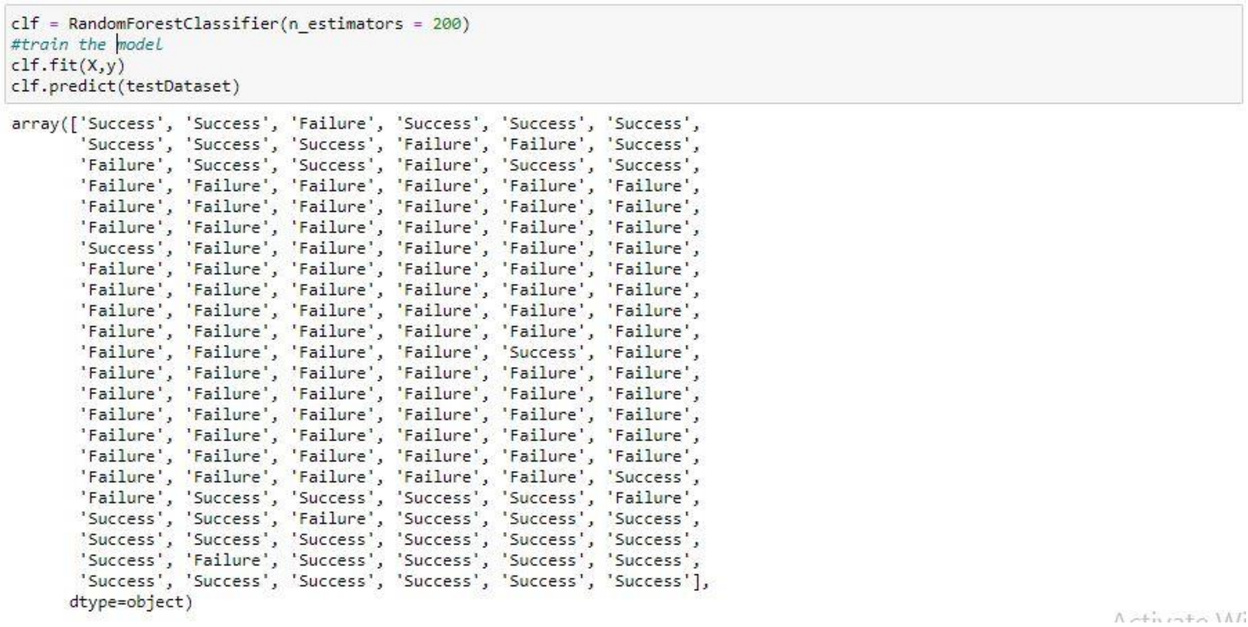
Confusion matrix for train data using Random Forest



```python
import seaborn as sns
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_test,y_pred)

fig, ax = plt.subplots(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d',xticklabels=[-1,1], yticklabels=[-1,1])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Predictions for test data



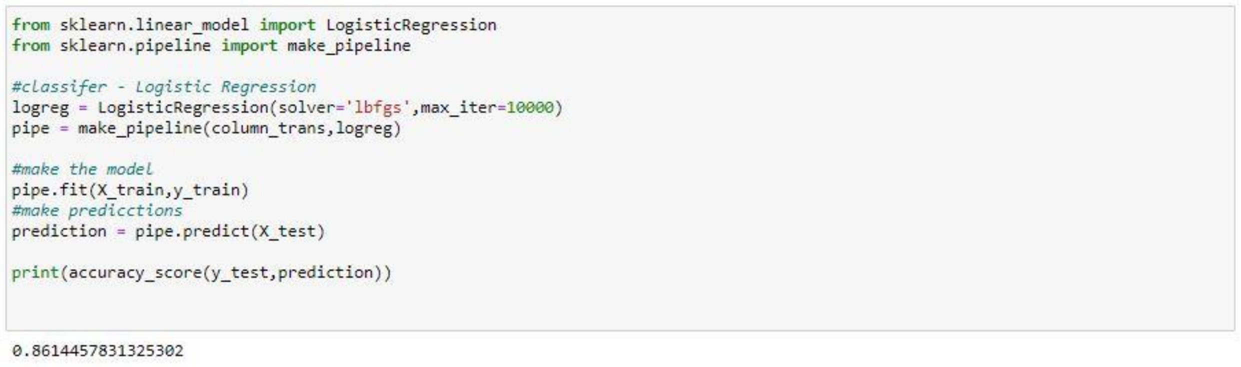
```python
clf = RandomForestClassifier(n_estimators = 200)
#train the model
clf.fit(X,y)
clf.predict(testDataset)
```

```
array(['Success', 'Success', 'Failure', 'Success', 'Success', 'Success',
       'Success', 'Success', 'Success', 'Failure', 'Failure', 'Success',
       'Failure', 'Success', 'Success', 'Failure', 'Success', 'Success',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Success', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Success', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Failure',
       'Failure', 'Failure', 'Failure', 'Failure', 'Failure', 'Success',
       'Failure', 'Success', 'Success', 'Success', 'Success', 'Failure',
       'Success', 'Success', 'Failure', 'Success', 'Success', 'Success',
       'Success', 'Success', 'Success', 'Success', 'Success', 'Success',
       'Success', 'Failure', 'Success', 'Success', 'Success', 'Success',
       'Success', 'Success', 'Success', 'Success', 'Success', 'Success'],
      dtype=object)
```

**Method 2 :**

Pipelines consist of several steps to train a model. Machine learning pipelines are iterative as every step is repeated to continuously improve the accuracy of the model and achieve a successful algorithm.

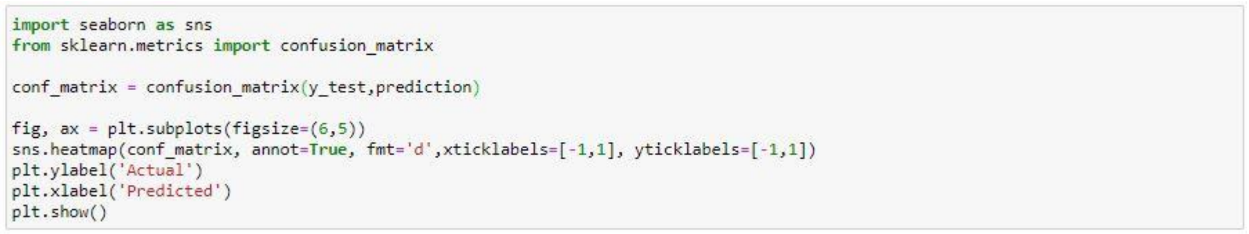
Here we used make_pipeline to create a pipeline of transforms with a final estimator. As our final estimator we used Logistic Regression with a maximum 10000 of iterations.



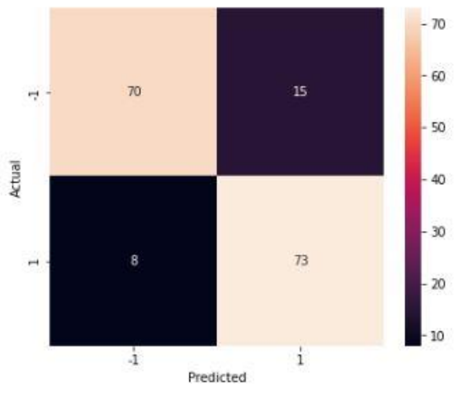
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

#classifer - Logistic Regression
logreg = LogisticRegression(solver='lbfgs',max_iter=10000)
pipe = make_pipeline(column_trans,logreg)

#make the model
pipe.fit(X_train,y_train)
#make predicctions
prediction = pipe.predict(X_test)

print(accuracy_score(y_test,prediction))
```

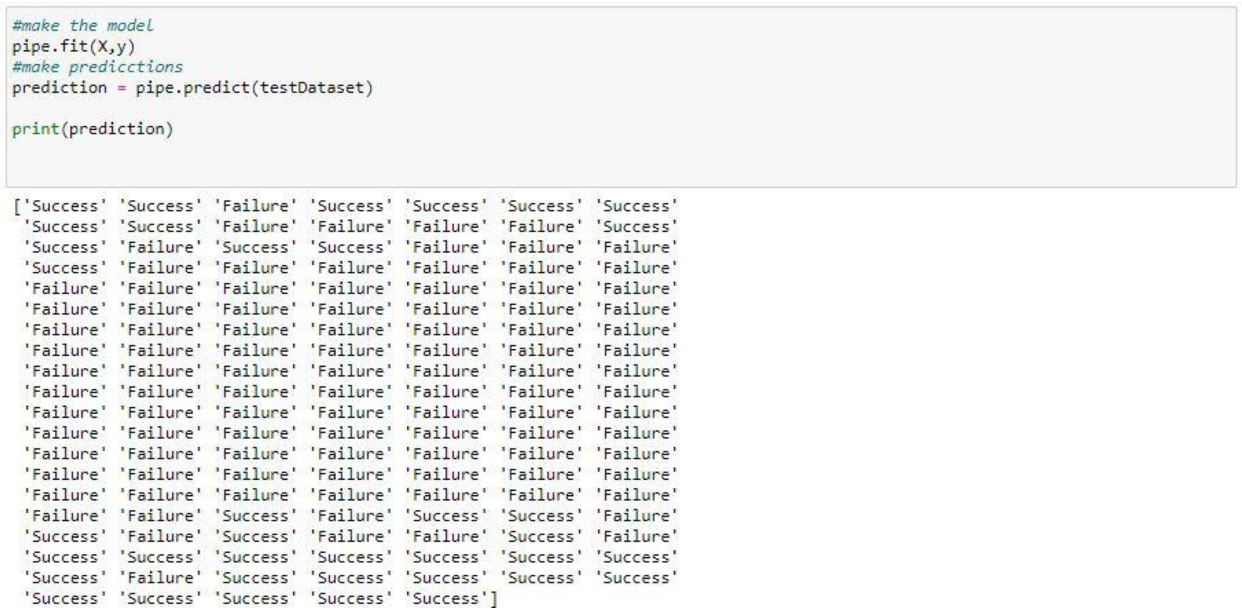
```
0.8614457831325302
```

Confusion Matrix for pipeline using Logistic Regression



```
import seaborn as sns
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_test,prediction)

fig, ax = plt.subplots(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d',xticklabels=[-1,1], yticklabels=[-1,1])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```





```python
#make the model
pipe.fit(X,y)
#make predicctions
prediction = pipe.predict(testDataset)

print(prediction)
```

```
['Success' 'Success' 'Failure' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Success'
 'Success' 'Failure' 'Success' 'Success' 'Failure' 'Failure' 'Failure'
 'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Success' 'Failure' 'Success' 'Success' 'Failure'
 'Success' 'Failure' 'Success' 'Failure' 'Failure' 'Success' 'Failure'
 'Success' 'Success' 'Success' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Failure' 'Success' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Success' 'Success' 'Success' 'Success']
```

By considering the confusion matrix of two methods, we can see that
Method 1 accuracy = 0.83734
Method 2 accuracy = 0.86144

By these accuracy results we can see the method 2 has more accuracy than method1.

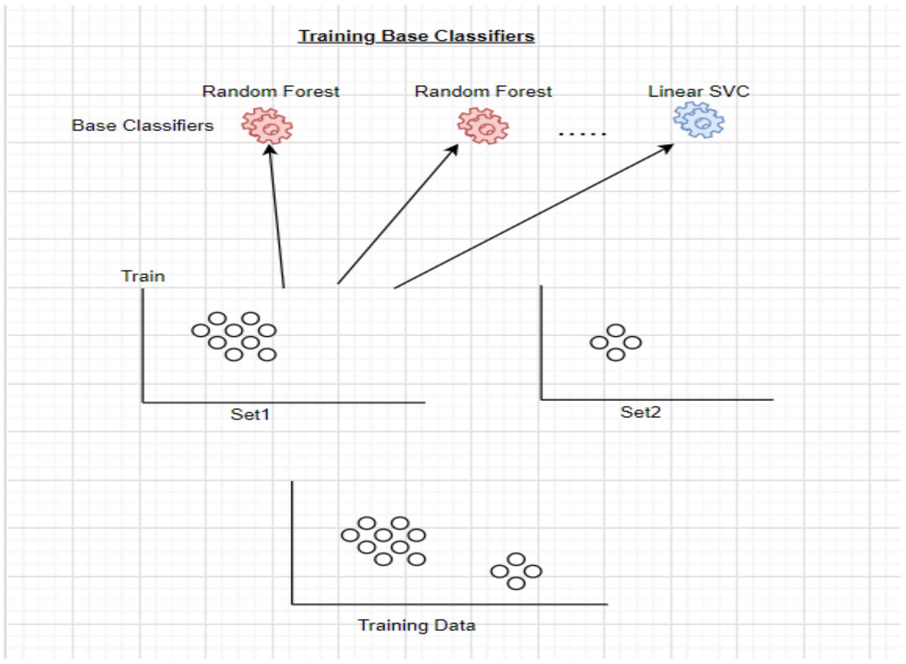
Let see , can we improve method 1 accuracy on test data.

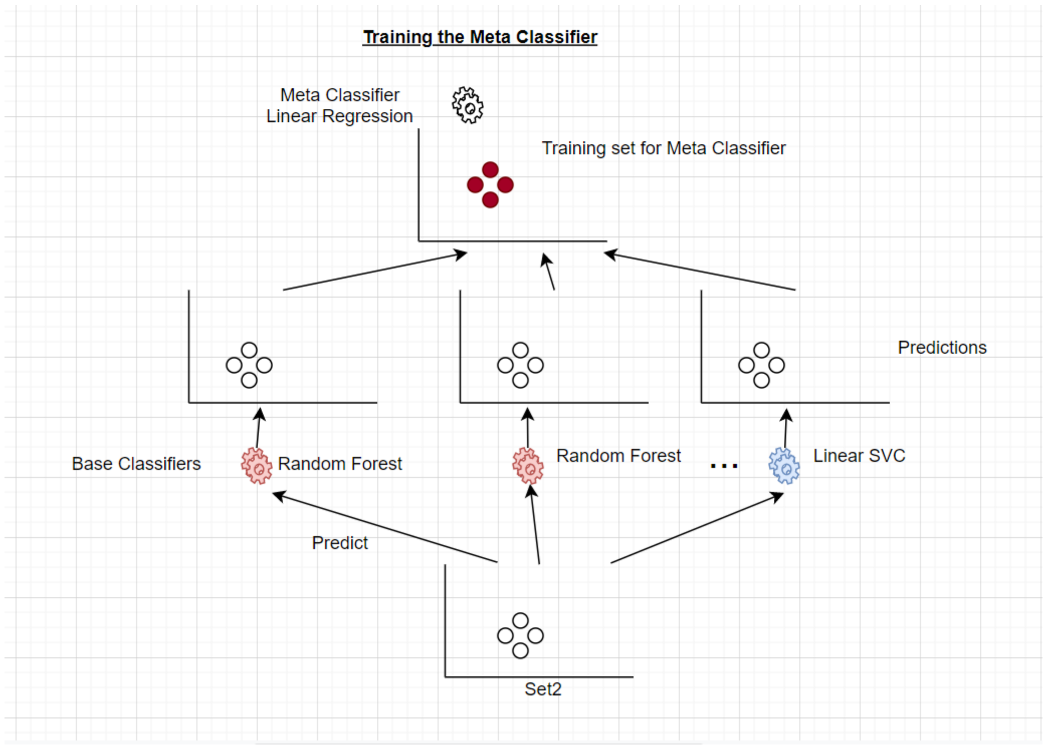
## 3.5 Improving the Results

**Method 1:**
But the above model when working with the test dataset, the accuracy was not  higher as we expected. So we had to improve our model to have good accuracy.

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features.The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous.So, using stacking we can improve the performance of our classifiers.

So here we used  to improve our model. In stacking as base estimators we used Random Forest Classifier with 2000 trees and  Support Vector Classification. And as the final estimator we used Logistic Regression. To train the final estimator we used  3-fold cross validation.


Training Base Classifiers
Random Forest
Random Forest
Linear SVC
Base Classifiers
.....
Train
Set1
Set2
Training Data

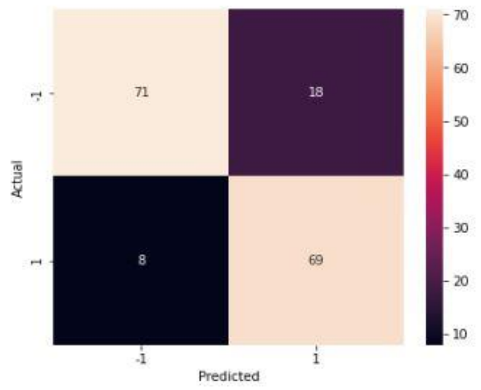

Training the Meta Classifier
Meta Classifier
Linear Regression
Training set for Meta Classifier
Predictions
Base Classifiers
Random Forest
Random Forest
...
Linear SVC
Predict
Set2

```
#convert nominal attributes to numeric attributes
lb_make = LabelEncoder()
l=['A1','A3','A4','A6','A8','A9','A11','A13','A15']
for i in l:
     temp = lb_make.fit_transform(dataset[i])
     dataset[i] = temp

     temp1 = lb_make.fit_transform(testDataset[i])
     testDataset[i] = temp1

#x and y
X = dataset.drop('A16' , axis='columns')
y = dataset['A16']

#base classifiers
estimators = [
    ('rf', RandomForestClassifier(n_estimators=2000, random_state=42)),
    ('svr', make_pipeline(StandardScaler(),
                          LinearSVC(random_state=42,max_iter=2000))) ]
#stacking
clf = StackingClassifier(
     estimators=estimators,final_estimator=LogisticRegression(),  cv=3, stack_method='auto', n_jobs=None,
    passthrough=False, verbose=0
 )

#train model
clf.fit(X_train,y_train)
#predict for test data
y_pred=clf.predict(X_test)
#calculate accuracy
accuracy_score(y_test,y_pred)
```

```
0.8433734939759037
```

Confusion matrix for train data using Stacking

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_test,y_pred)

fig, ax = plt.subplots(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d',xticklabels=[-1,1], yticklabels=[-1,1])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

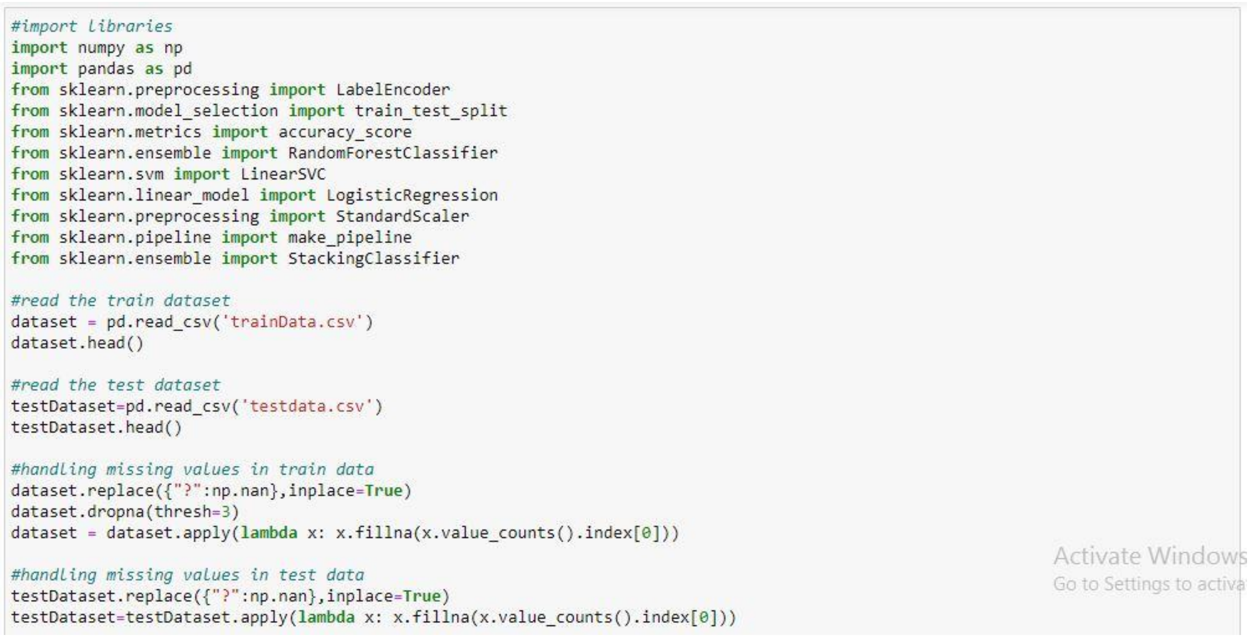


By considering this confusion matrix we can see the accuracy of method 1 was increased.

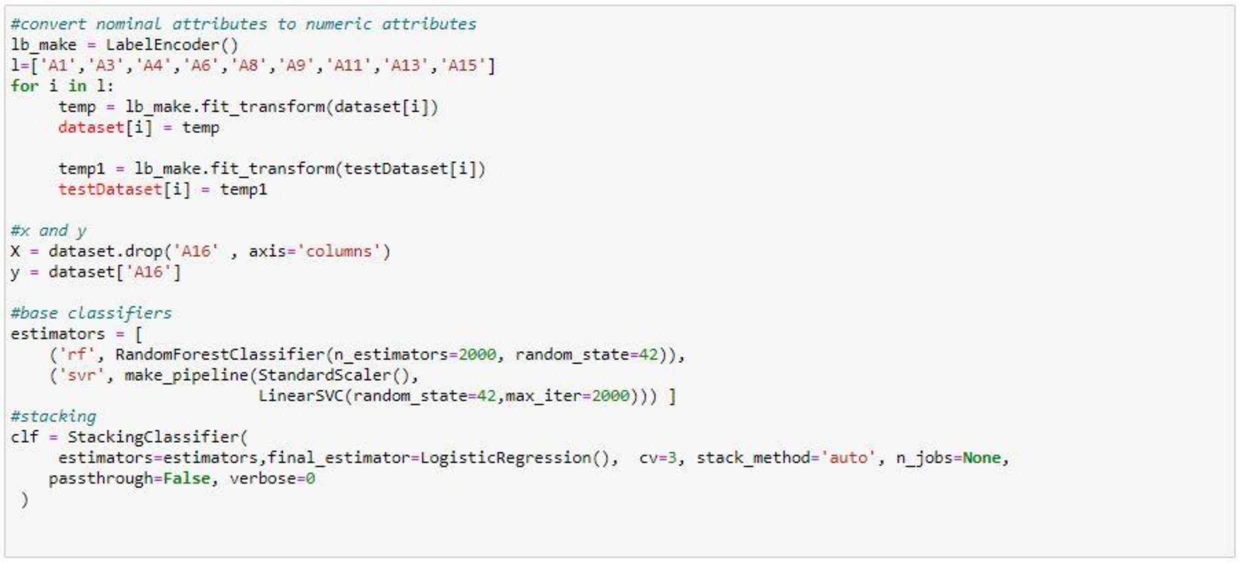
## 3.6 Final Model

We tried in various methods to get the predictions and we finalized the Method 1 and Method 2 as our final models on kaggle . As we got the highest marks for method 1, we selected that model as our final model.

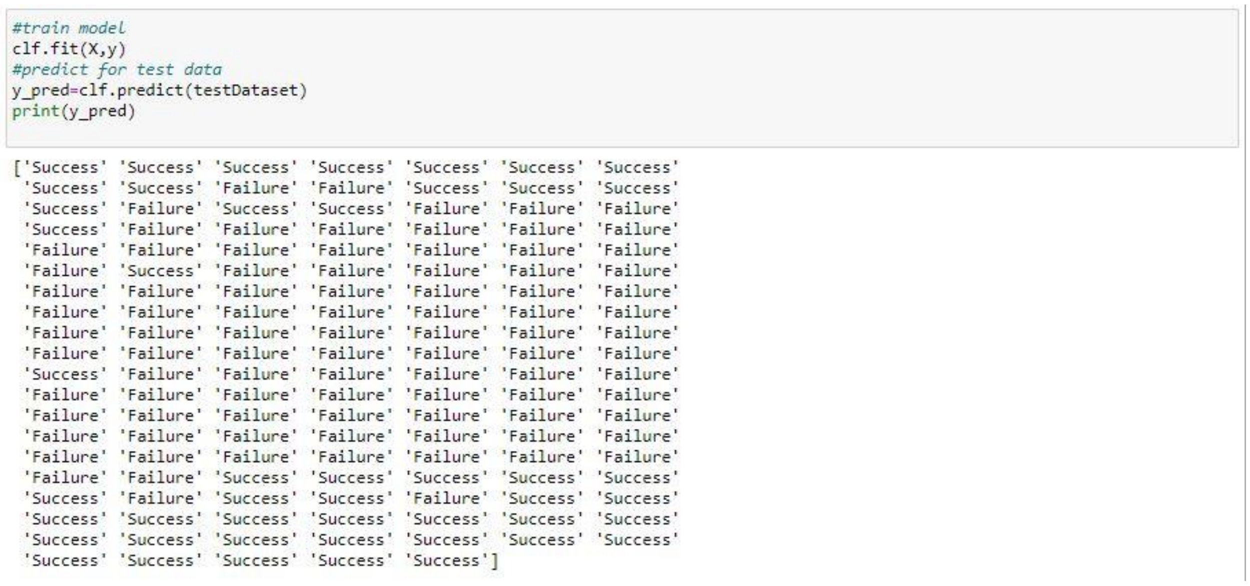
This is our final model (using Method 1) on test data to get the predictions.



```
#import libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier

#read the train dataset
dataset = pd.read_csv('trainData.csv')
dataset.head()

#read the test dataset
testDataset=pd.read_csv('testdata.csv')
testDataset.head()

#handling missing values in train data
dataset.replace({"?":np.nan},inplace=True)
dataset.dropna(thresh=3)
dataset = dataset.apply(lambda x: x.fillna(x.value_counts().index[0]))

#handling missing values in test data
testDataset.replace({"?":np.nan},inplace=True)
testDataset=testDataset.apply(lambda x: x.fillna(x.value_counts().index[0]))
```



```
#convert nominal attributes to numeric attributes
lb_make = LabelEncoder()
l=['A1','A3','A4','A6','A8','A9','A11','A13','A15']
for i in l:
     temp = lb_make.fit_transform(dataset[i])
     dataset[i] = temp

     temp1 = lb_make.fit_transform(testDataset[i])
     testDataset[i] = temp1

#x and y
X = dataset.drop('A16' , axis='columns')
y = dataset['A16']

#base classifiers
estimators = [
    ('rf', RandomForestClassifier(n_estimators=2000, random_state=42)),
    ('svr', make_pipeline(StandardScaler(),
                          LinearSVC(random_state=42,max_iter=2000))) ]
#stacking
clf = StackingClassifier(
     estimators=estimators,final_estimator=LogisticRegression(),  cv=3, stack_method='auto', n_jobs=None,
    passthrough=False, verbose=0
 )
```



```
#train model
clf.fit(X,y)
#predict for test data
y_pred=clf.predict(testDataset)
print(y_pred)
```

```
['Success' 'Success' 'Success' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Success' 'Failure' 'Failure' 'Success' 'Success' 'Success'
 'Success' 'Failure' 'Success' 'Success' 'Failure' 'Failure' 'Failure'
 'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Success' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Failure' 'Success' 'Success' 'Failure' 'Success' 'Success'
 'Success' 'Success' 'Success' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Success' 'Success' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Success' 'Success' 'Success' 'Success']
```

## 4. CONCLUSION

We tried in various methods to train this model by replacing the missing values using many techniques, converting the nominal values to numerical values using various encoders and using various algorithms to train the model.

In conclusion, by considering the accuracy of the algorithms and number of  correct predictions of the models (marks we get for predictions on kaggle) we can see the ensemble learning improves the performance of the model.