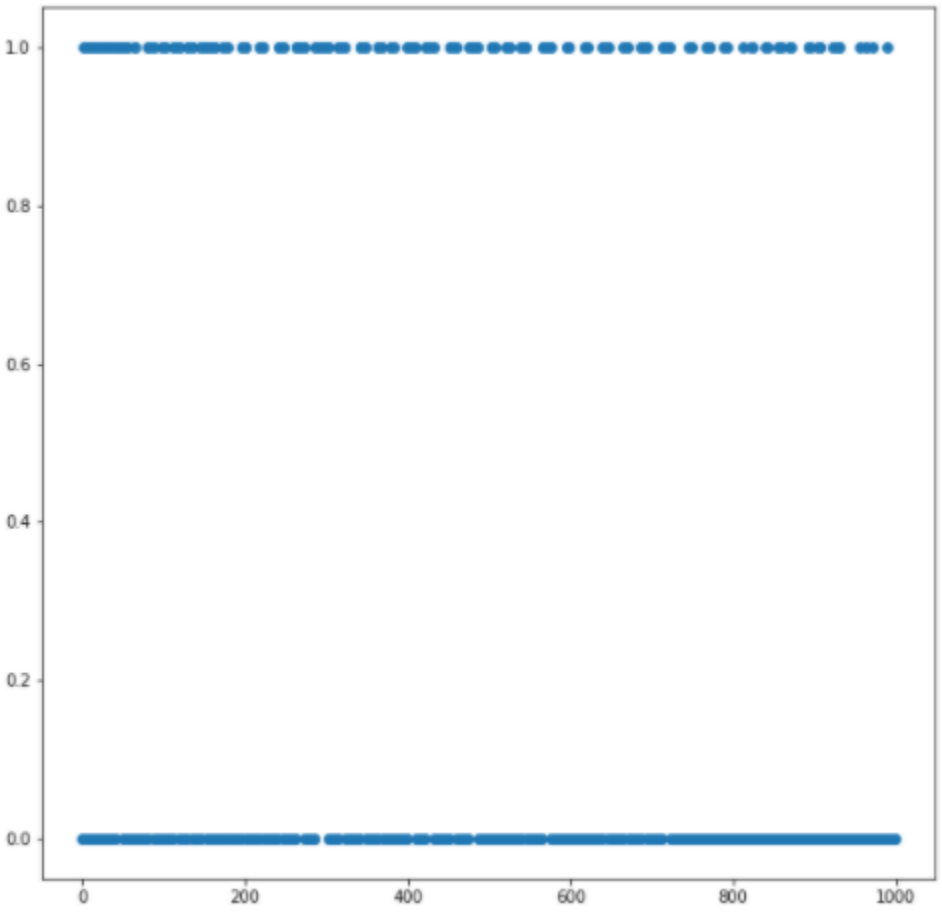
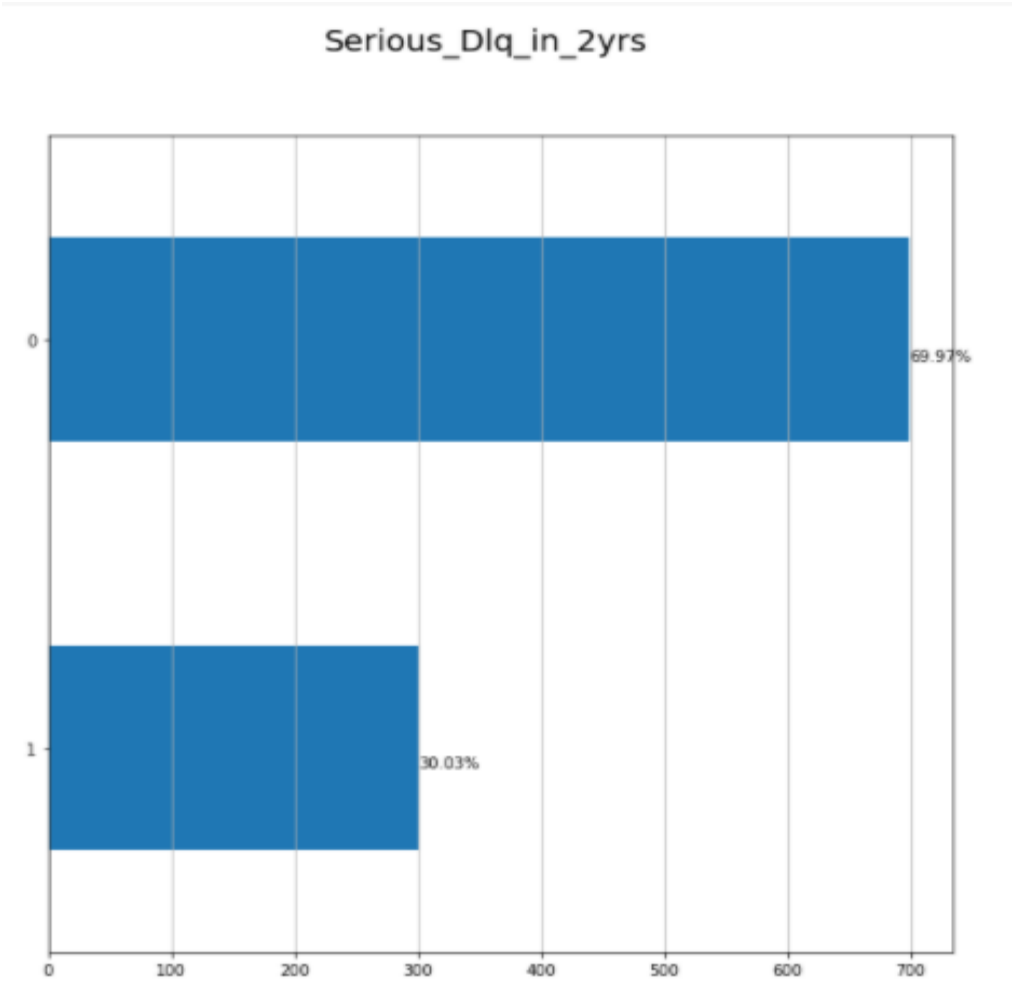
Question 1- The dataset consists of 999 samples and 7 features with 1 target variable.

1. Preprocessing :

a.) We make a scatter plot and a bin plot of the target variable to see its distribution.



Scatter plot



bin plot

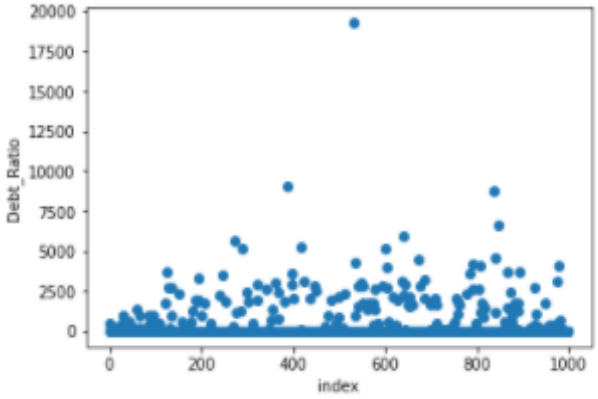
b.) For analysing the dataset , we need to remove NaN values . So we use the below written function to get total number of NaN values in the dataset

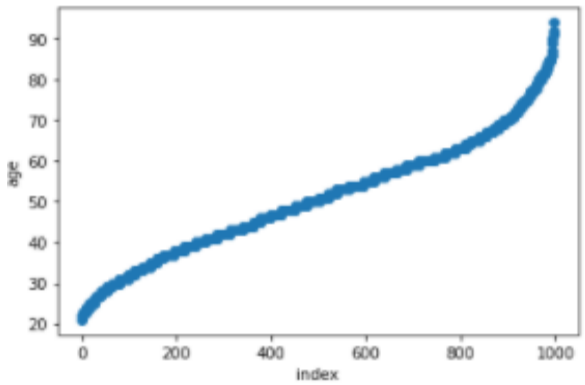
```
data.isna().sum().sum().
```

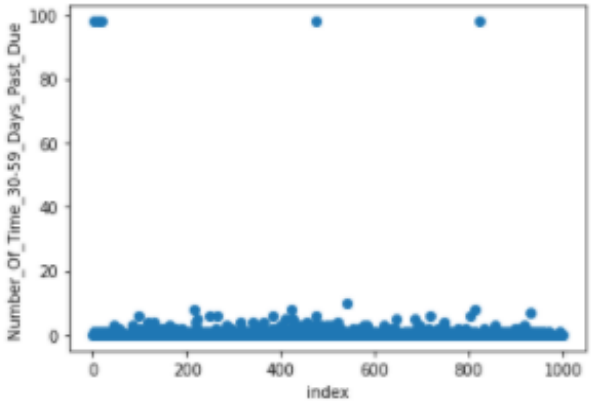
Now the NaN values are replaced with the mean value using the following code.

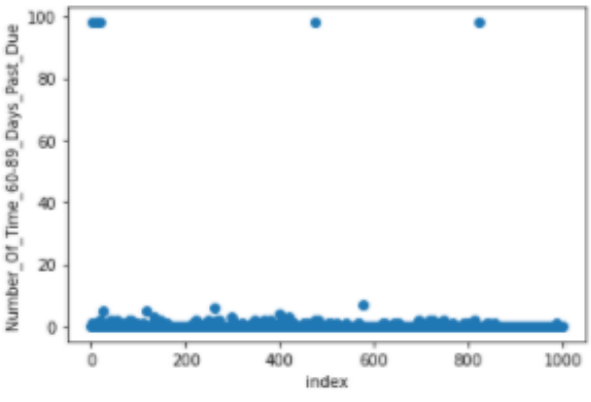
```
data.fillna(data.mean(), inplace=True)
```

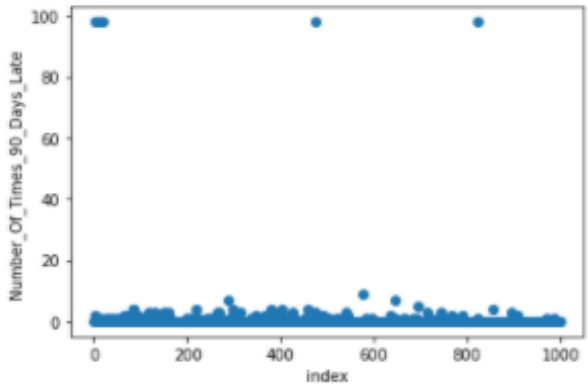
c.) We have used scatter plots to visualize the distribution of data for every feature.

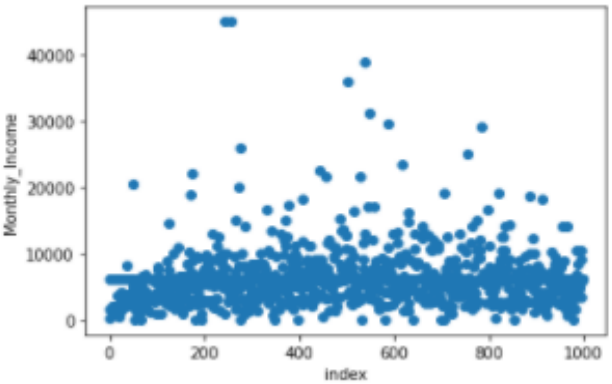

Debt_Ratio
index

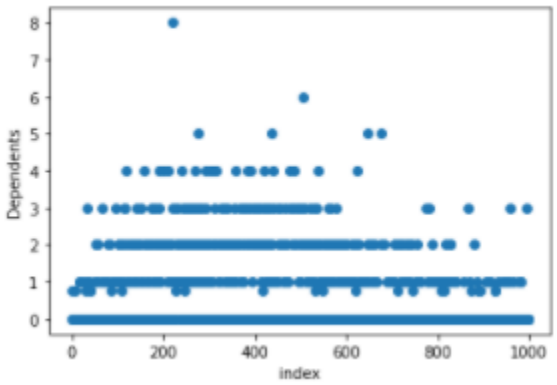

age
index


Number_Of_Time_30-59_Days_Past_Due
index


Number_Of_Time_60-89_Days_Past_Due
index


Number_Of_Times_90_Days_Late
index


Monthly_Income
index



2. Train the Random Forest Classifier :

First we separate the dataset into feature space and target space by using

```
X = data.iloc[:,1:].values #indep.
y = data.iloc[:,0:1].values #dep
```

We use the below written code to make a random forest model with 100 trees and the given max_features and max_depth .

```
rf = RandomForestClassifier(n_estimators=100, max_features=3,
max_depth=4 )
```

Note that we haven't trained the model . This is due to the reason that we would use grid search first to find the best parameters for the model and would train the model with best parameters.

3. Perform 5 fold cross-validation and perform the grid-search for the parameters to find the optimal value of the parameters:

We can cross validate using the `StratifiedKFold` function which will split the data into training and testing randomly 5 times and find scores for each split `.lst_accu_stratified` will contain the accuracy scores of each split

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=5)

lst_accu_stratified = []

for train_index, test_index in skf.split(X, y):
```

```
        # print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        rf.fit(X_train, y_train.ravel())
        lst_accu_stratified.append(rf.score(X_test, y_test))
    lst_accu_stratified
```

Now we use the GridSearchCV function to find the best parameters for our model

```
parameters = {'max_features': [1, 2, 4], 'max_depth': [2,3,4,5]}
grid_search = GridSearchCV(rf, parameters, n_jobs=-1, scoring='roc_auc', cv=skf)
grid_search.fit(X_train, y_train.ravel())
print(grid_search.best_params_)
```

Now that we have found the best classifier using grid_search.best_estimator_ function , we can train our classifier on training data.

```
clf = grid_search.best_estimator_
clf.fit(X_train,y_train.ravel())
```

4. Get the best score from the grid search. :

The following code is used to print best score of classifiers

```
print(grid_search.best_score_)
```

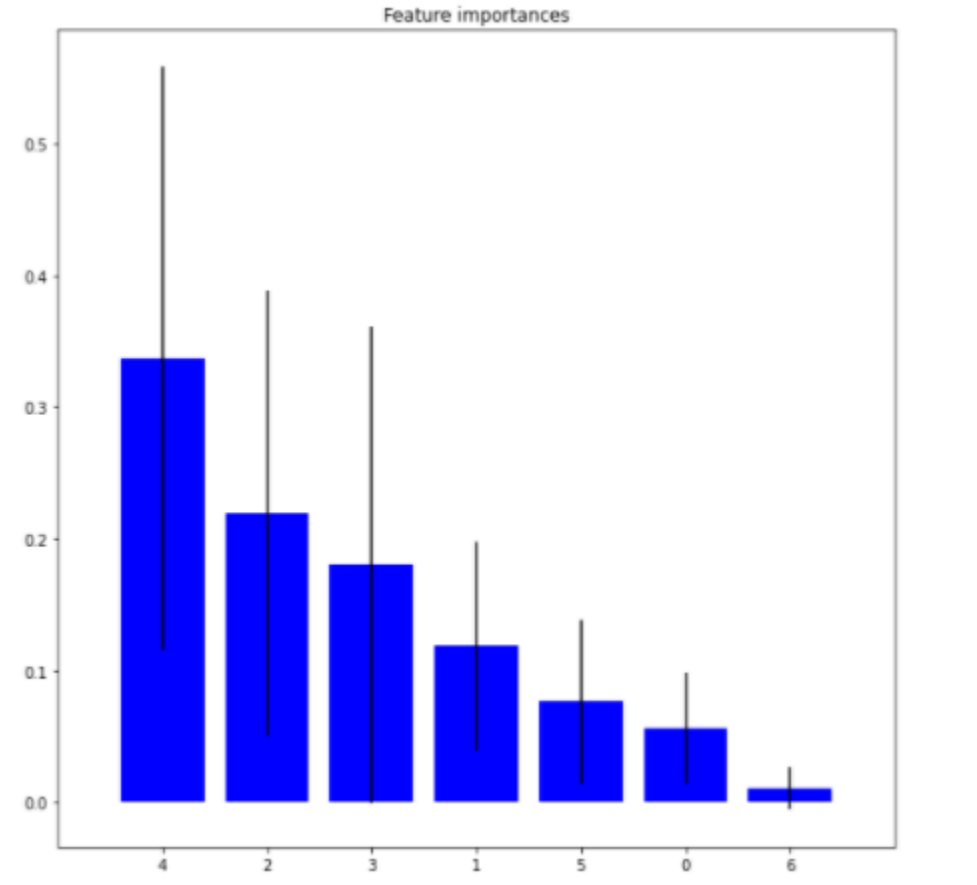
5. Find the feature which has the weakest impact in the Random Forest Model:

Using the clf.feature_importances_ , we get an array specifying feature importance for each feature . We then sort the feature importance array to get logical intuition of the

weakest feature i.e the first element after sorting in increasing order will be the weakest feature

```
pd.DataFrame({'feat': f[1:],

              'coef': clf.feature_importances_}).sort_values(by='coef',
ascending=True)
```

	feat	coef
6	Dependents	0.010952
0	Debt_Ratio	0.056553
5	Monthly_Income	0.076606
1	age	0.118678
3	Number_Of_Time_60-89_Days_Past_Due	0.180416
2	Number_Of_Time_30-59_Days_Past_Due	0.219724
4	Number_Of_Times_90_Days_Late	0.337072



Question 2-

6. Perform bagging-based classification using Decision Tree as the base classifier.

a.) We are supposed to make a bagging based classifier using decision trees as the base classifier. Note that the number of decision trees in this bagging based classifier can vary. For this lab, we have to compare bagging classifiers with 2,3,4 decision trees.

```
def get_models():

 param_grid = {

    'base_estimator__max_depth' : [1, 2, 3, 4, 5],

    'max_samples' : [0.05, 0.1, 0.2, 0.5]

}

 models= []

 clf = tree.DecisionTreeClassifier()

   model_1   =   GridSearchCV(BaggingClassifier(clf,   n_estimators=2,
bootstrap=True), param_grid, scoring = 'roc_auc')

   model_2   =   GridSearchCV(BaggingClassifier(clf,   n_estimators=3,
bootstrap=True), param_grid, scoring = 'roc_auc')

   model_3   =   GridSearchCV(BaggingClassifier(clf,   n_estimators=4,
bootstrap=True), param_grid, scoring = 'roc_auc')

 models.append( model_1)

 models.append( model_2)

 models.append( model_3)

 return models
```

The above code is used to make all 3 models i.e model_1, model_2, model_3 with 2,3,4 decision trees respectively. Here we use GridSearchCV on every bagging classifier to get the best model from each bagging classifier

b.) The function `get_models()` returns a list of models consisting of all 3 bagging classifiers.

```
def evaluate_model(model, X, y):

 scores = cross_val_score(model, X, y, cv=5)

 return scores
```

Apparently the above function evaluate_model will return the cross validation accuracy scores for the given model.

c.) `# Summarize the performance by getting mean and standard deviation of scores`

```
results = []

names = ["model_1", "model_2", "model_3"]

standard_deviations =[]

models = get_models()

for i in range (0,3):

 scores = evaluate_model(models[i], X, y.ravel())

 results.append(scores)

 msg = "%s: %f (%f)" % (names[i], scores.mean(), scores.std())

 standard_deviations.append(scores.std)

 print(msg)
```

We can summarise the performance of each bagging model using the above code. The “for” loop runs 3 times and each time the scores are calculated for the model using evaluate_model function and the scores are appended to results array The code gives following results :

Model: mean accuracy (standard deviation)

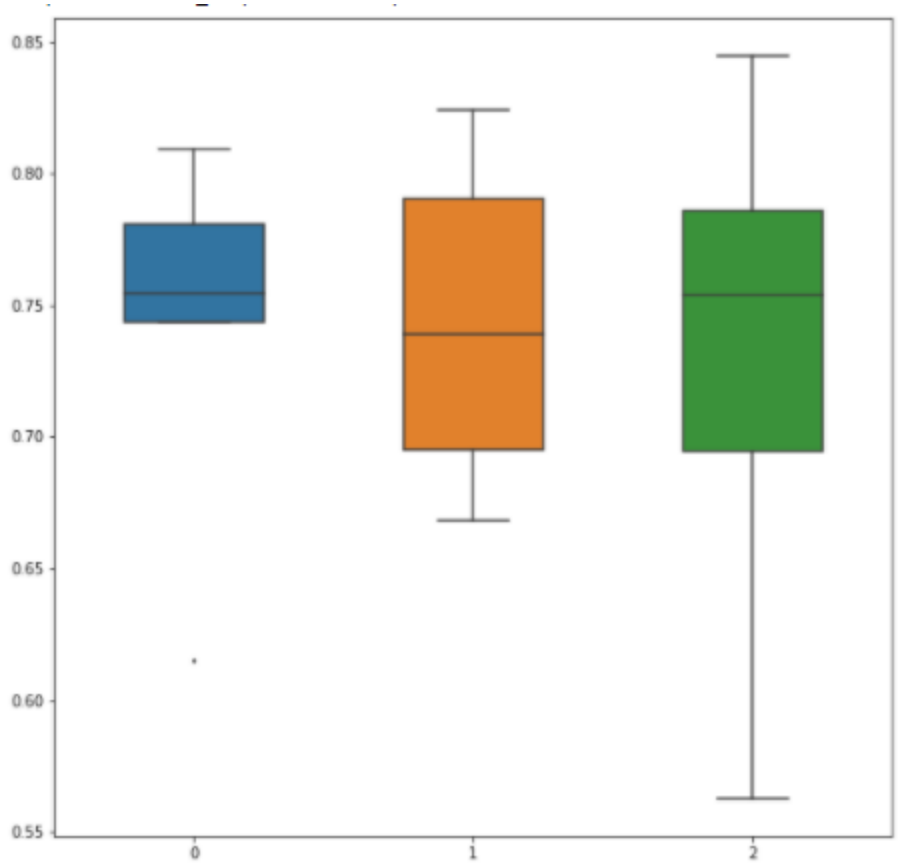
```
model_1: 0.740622 (0.066819)

model_2: 0.743493 (0.057856)

model_3: 0.728278 (0.096056)
```

d.) Plot the model performance for comparison using boxplot.

For each bagging classifier, we plot the box-model using the code

```
sns.boxplot( data = results, width = 0.5, fliersize= 2)
```

Where results consists of accuracy scores for each bagging classifier



7. Compare the best performance of bagging with random forest by plotting using boxplot.

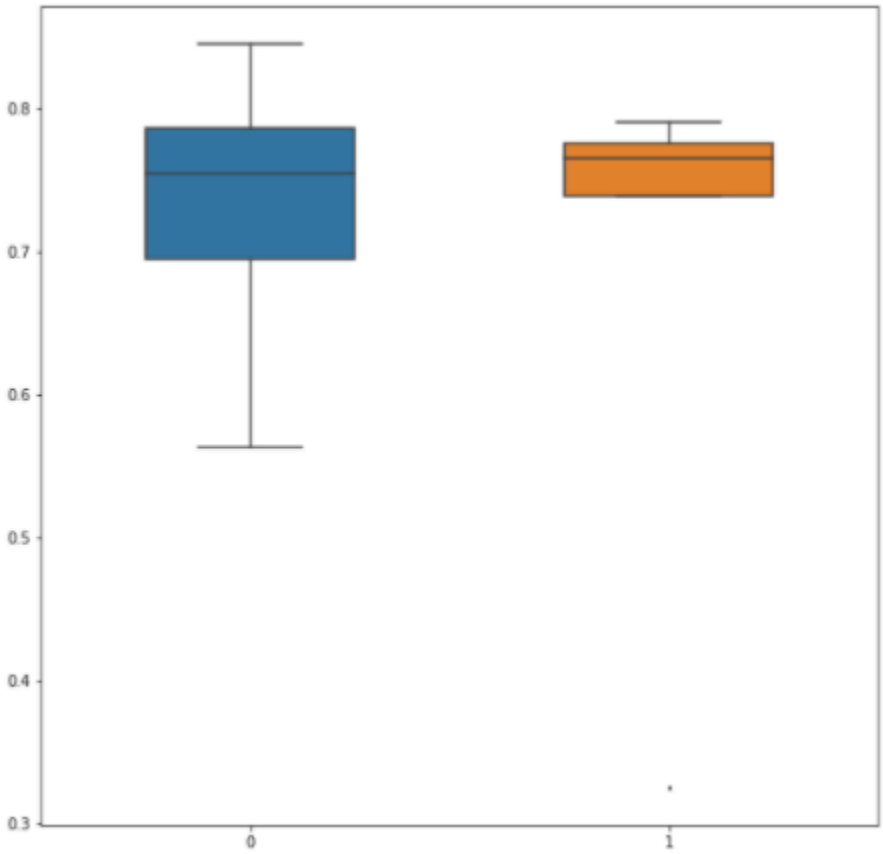
The idea is to use GridSearchCV function to find the best bagging classifier out of the 3 models i.e model_1, model_2 and model_3.

GridSearchCV will tell us the best n_estimators parameter out of 2,3,4 which will eventually tell us which model is the best.

```
parameters = {'n_estimators':[2,3,4]}

grid_search_bag= GridSearchCV(BaggingClassifier(), parameters, n_jobs=-1,
scoring='roc_auc', cv=skf)

grid_search_bag.fit(X_train, y_train.ravel())

num_estimators =grid_search_bag.best_params_['n_estimators']
```

```
num_est =[2,3,4]

index = num_est.index(num_estimators)

results_bag_rf= [results[index]]

results_bag_rf.append(lst_accu_stratified)

sns.boxplot( data = results_bag_rf, width = 0.5, fliersize= 2)
```

num_estimator will tell the best number of trees out of 2,3,4 .Index will give the index of the bagging classifier which is formed using num_estimator number of trees. results_bag_rf will consist of scores of both models i.e best bagging model and rf model.



Model at index 0 is bagging model and model at index 1 is rf model