



**NGEE ANN**  
POLYTECHNIC

**School of InfoComm Technology**

---

## **Machine Learning**

Diploma in Data Science (DS)

Diploma in Information Technology (IT)

October 2022 Semester

### **INDIVIDUAL ASSIGNMENT 2**

(40% of Machine Learning Module)

## Table of Contents

Introduction.....	4
HR Analysis.....	6
Logistic Regression.....	7
Loading of data .....	7
Building a logistic regression model.....	7
Evaluate and improve the model.....	9
Decision Tree Classifier .....	10
Loading of data .....	10
Building a Decision Tree model .....	10
Evaluate and improve the model.....	12
Artificial Neural Network.....	14
Loading of data .....	14
Building a ANN model .....	14
Evaluate and improve the model.....	16
Random Forest .....	18
Loading of data .....	18
Building a Random Forest model.....	18
Evaluate and improve the model.....	18
Support Vector Machine.....	21
Loading of data .....	21
Building a SVM model.....	21
Evaluate and improve the model.....	22
XGBoost.....	23
Loading of data .....	23
Building a XGB model .....	23
Evaluate and improve the model.....	24
Voting .....	25
Loading of data .....	25
Building a Voting model .....	26
Bagging .....	26
Loading of data .....	26
Building a Bagging model .....	26
Evaluate and improve the model.....	27
Ada Boost.....	29
Loading of data .....	29
Building a ADB model .....	29
Evaluate and improve the model.....	30
Summary .....	32
AirBnb.....	33
Isolation Forest.....	34
Build the model.....	34

Split the data .....	34
Linear Regression .....	34
Loading of data .....	34
Normal Dataframe .....	34
Building a LR model .....	34
Evaluate and improve the model.....	35
Linear Regression –Inliers and Outliers Dataframe .....	36
Building a LR model .....	36
Evaluate and improve the model.....	37
Linear Regression –Log Dataframe .....	37
Building a LR model .....	37
Evaluate and improve the model.....	38
Summary of Linear Regression .....	38
Decision Tree Regression –Log Dataframe .....	39
Loading of data .....	39
Building a DT model .....	39
Evaluate and improve the model.....	40
Decision Tree Regression – Summary .....	41
Artificial Neural Network - Log Dataframe.....	41
Loading of data .....	41
Building a ANN model .....	41
Evaluate and improve the model.....	41
Random Forest Regression –Log Dataframe .....	43
Loading of data .....	43
Building a RF model .....	43
Evaluate and improve the model.....	43
Random Forest Regression – Summary.....	44
Ada Boost Regression –Log Dataframe .....	45
Loading of data .....	45
Building a Ada Boost model .....	45
Evaluate and improve the model.....	45
Ada Boost Regression – Summary .....	46
XGB – Log Dataframe.....	46
Loading of data .....	46
Building a XGB Boost model .....	46
Evaluate and improve the model.....	47
Summary .....	48
Conclusion.....	49
Reflection .....	50

# Introduction

In this final assignment, I will be utilizing the power of machine learning models to solve two types of problems - classification and regression. My focus will be on using Jupiter notebook to build and train these models.

The two problems that we will be addressing are the HR Analytics problem and the Airbnb problem. The HR Analytics problem is a classification problem where I will use the data to predict if a particular employee will be promoted or not. On the other hand, the Airbnb problem is a regression problem where I will use the data to predict the price of a particular listing.

Our main objective in this assignment is to build machine learning models that can accurately predict the outcomes for these two problems. This will involve selecting the appropriate algorithms and tuning their hyperparameters to achieve the best performance. In the end, I will be evaluating the performance of my models using various metrics and documenting my findings.

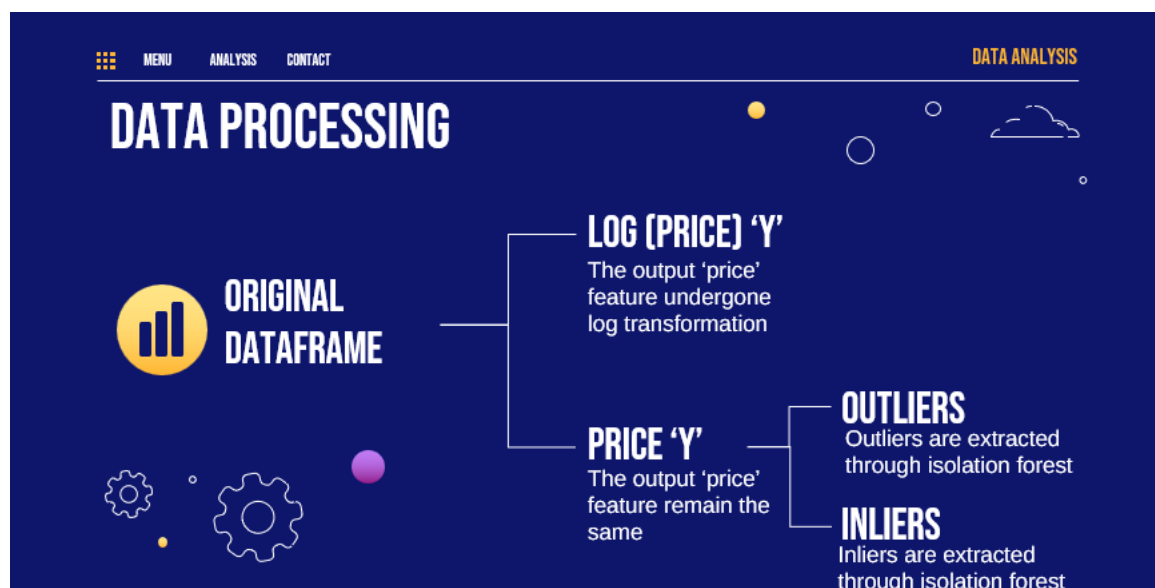
For this assignment, I have already cleaned and explored the datasets for both the HR Analytics and Airbnb problems in the previous assignment. Hence, we will be using the cleansed data for my machine learning modeling purposes.

For this assignment, I will be following a four-step process for each of the classification models I will be building:

- 1) Loading of assignment 1 data
- 2) Building a classification model
- 3) Evaluate and improve that model performance.
- 4) Summarize my finding.

And ending with a overall conclusion.

However, for regression, I am doing a different approach.



For this assignment, I have taken a unique approach. I have created four different dataframes, and I will be training each of them using various models. The first dataframe will be the original dataframe, and from it, I have split it into inliers and outliers. The second dataframe will be the inliers, and the third will be the outliers, which I have identified using

the Isolation Forest approach. The fourth dataframe is particularly noteworthy as I have applied a log function to the price feature. This was done during the data cleaning process in Assignment 1, as I noticed that the price feature was skewed.

Thus, for the regression problem, I will be using 4 separate datasets for training. As a result, I will be following this four-step process four times for each, once for each dataset:

- 1) Loading of assignment 1 data
- 2) Building a classification model
- 3) Evaluate and improve that model performance.
- 4) Summarize my finding.

And ending with overall conclusion.



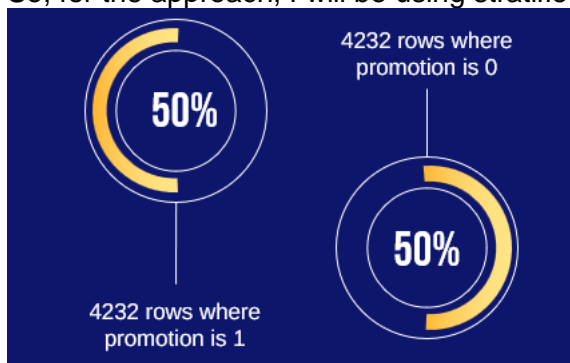
## HR Analysis

The objective of this task is to use machine learning modeling to predict which employees are most likely to be promoted. To do this, I will be using a dataset that includes employee personal information, education, and past performance. I will use these variables to train multiple models and determine if an employee will be promoted.

Models that I will using:

- 1) Logistic regression
- 2) Decision tree
- 3) Artificial natural network ( mlp classifier )
- 4) Random Forest
- 5) Support Vector Machine
- 6) extreme Gradient Boosting
- 7) Voting
- 8) Bagging
- 9) Ada Boost
- 10) TensorFlow\*

So, for the approach, I will be using stratified sampling.



I have employed stratified sampling to balance the data, where both promotion with a value of 1 and promotion with a value of 0 have 4,232 rows each. This sampling is crucial for the classification problem as it helps to ensure that the distribution of the target variable (promotion) is representative in both the training and testing datasets. This helps to prevent potential biases and ensures that the model is trained and evaluated on a representative sample of the data.

And for cleaning the dataframe :

Gender	int64	Gender	int64
Department	int64	Department	int64
Region	int64	Region	int64
Is_promoted	int64	Is_promoted	int64
avg_training_score	int64	avg_training_score	int64
Awards_won?	int64	Awards_won?	int64
KPIs_met >80%	int64	KPIs_met >80%	int64
Length_of_service	int64	Length_of_service	int64
Previous_year_rating	float64	Previous_year_rating	float64
Age	int64	Age	int64
No_of_training	int64	No_of_training	int64
Recruitment_channel	int64	Recruitment_channel	int64
Education	int64	Education	int64
department	object		
gender	object		
region	object		

Next, I will be dropping 3 features (department , gender, region ) as they are object and have already undergone numeric changes. So, my X will be all these features here and my Y would be promotion.

So, the naming of my dataframe will follow df\_hr\_<model\_name>.

## Logistic Regression

### Loading of data

Defining my model Input(X) and Output (y)

```
X = df_hr_Logistic.drop(['is_promoted'],axis=1)
y = df_hr_Logistic["is_promoted"]
```

### Building a logistic regression model

```
# Create the model "lg"
lg = LogisticRegression(solver='lbfgs', # optimization solver
                        max_iter=20000) # maximum iterations: set a big number to make sure the optimization solver will
# run enough iterations to let the model converge

# Fitting the model to the training set
lg.fit(X_train,y_train)
```

So I am building a simple logistic regression model where I am declaring my solver as 'lbfgs'. It is used to optimize the parameters of the logistic regression model by finding the values that minimize the loss function.

```
print(lg.coef_)
print(lg.intercept_)

[[-0.29610977 -0.01770482 -0.24111453 -0.0219064  0.38369054  0.02481377
  1.53041992  2.27257271  0.04532543 -0.01255555 -0.07678355  0.00606306]]
[-3.80285796]
```

So I am printing out the coefficients and the intercept

For example, some feature has a positive coefficient, it means that an increase in that feature is associated with an increase in the probability of the positive class. Conversely, some the coefficient is negative, it means that an increase in that feature is associated with a

decrease in the probability of the positive class.

For example, the coefficient for the feature "education" is -0.29610977, which means that if all other features remain the same, a higher education value would lead to a lower probability of being promoted. On the other hand, the coefficient for the feature "awards\_won?" is 2.27257271, which means that if all other features remain the same, a higher value for awards\_won would lead to a higher probability of being promoted.

Next I will be exploring the `y_fitted` and its probabilities.

```
# Calculated the fitted values for training set
y_fitted = lg.predict(X_train) # returns 0 or 1
y_fitted_prob = lg.predict_proba(X_train)[:,0] # returns probabilities
# printing y_fitted and y_fitted_prob
print(y_fitted)
print(y_fitted_prob)
```

```
[0 1 0 ... 1 1 1]
[0.66610722 0.33729876 0.84533517 ... 0.31900219 0.36913838 0.37016159]
```

As seen here where '`y_fitted`'= [0 1 0 ... 1 1 1], it means the model has predicted that the first observation will not be promoted, the second observation will be promoted, the third observation will not be promoted, and so on.

As seen here where '`y_fitted_prob`' = [0.66610722, 0.33729876, 0.84533517, ..., 0.31900219, 0.36913838, 0.37016159], this means that for the first observation, the model is 66.61% confident that the observation where employee is not promoted, for the second observation, the model is 33.73% confident and so on.

The training and testing accuracy :

```
print(lg.score(X_train, y_train), '(Train Accuracy)')
print(lg.score(X_test, y_test), '(Test Accuracy)')
```

```
0.7223160027008778 (Train Accuracy)
0.7125984251968503 (Test Accuracy)
```

So, the train accuracy is 0.7223, which means the model was able to correctly classify 72.23% of the training data. The test accuracy is 0.7125, which means the model was able to correctly classify 71.26% of the test data.

In this case, the accuracy is relatively high, which suggests that the model is a good fit for the data, but further improvement could still be made.



## Evaluate and improve the model.

So I created another model lg2 using statsmodel.api.

### Summary

Dep. Variable:	is_promoted	No. Observations:	5924
Model:	Logit	Df Residuals:	5912
Method:	MLE	Df Model:	11
Date:	Fri, 27 Jan 2023	Pseudo R-squ.:	0.2112
Time:	19:07:40	Log-Likelihood:	-3239.0
converged:	True	LL-Null:	-4106.1
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P> z	[0.025	0.975]
education	-0.6210	0.064	-9.754	0.000	-0.746	-0.496
recruitment_channel	-0.1133	0.054	-2.081	0.037	-0.220	-0.007
no_of_trainings	-0.4069	0.058	-7.033	0.000	-0.520	-0.294
age	-0.0729	0.004	-16.362	0.000	-0.082	-0.064
previous_year_rating	0.2920	0.027	10.778	0.000	0.239	0.345
length_of_service	0.0475	0.009	5.040	0.000	0.029	0.066
KPIs_met >80%	1.4748	0.063	23.282	0.000	1.351	1.599
awards_won?	2.4641	0.197	12.507	0.000	2.078	2.850
avg_training_score	0.0311	0.002	15.776	0.000	0.027	0.035
Region	-0.0227	0.003	-7.334	0.000	-0.029	-0.017
Department	-0.1107	0.015	-7.545	0.000	-0.139	-0.082
Gender	-0.0922	0.065	-1.409	0.159	-0.220	0.036

Since these two features are the only ones without 0 p-Value. So, I will be removing these features and retrain the model.

```
X = df_hr_Logistic.drop(['is_promoted', 'Gender', 'recruitment_channel'], axis=1)
y = df_hr_Logistic["is_promoted"]
```

### Building the model again

```
lg = LogisticRegression(solver="lbfgs", max_iter=10000)
lg.fit(X_train, y_train)
```

So this is the training and testing accuracy after dropping 2 features.

```
print(lg.score(X_train, y_train), '(Train Accuracy)')
print(lg.score(X_test, y_test), '(Test Accuracy)')

0.7219783929777177 (Train Accuracy)
0.7125984251968503 (Test Accuracy)
```

But more improvement can be made so I did GridSearch.

```
param_grid_lr = {
    'max_iter': [500, 10000, 20000, 50000],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'class_weight': ['balanced']
}
```


So now I have a wide selection of max\_iter and solver. The values that are boxed in blue are the suggested result by the GridSearch. So I trained my model based on these boxed values

## Final Training and testing accuracy

```
print(lg3.score(X_train, y_train), '(Train Accuracy)')
print(lg3.score(X_test, y_test), '(Test Accuracy)')

0.7228224172856178 (Train Accuracy)
0.7165354330708661 (Test Accuracy)
```

Overall the conclusion:

	TRAINING ACCURACY	TESTING ACCURACY	MODEL
Before	0.7223160027008778	0.7125984251968503	0.009717577
After	0.7228224172856178 	0.7165354330708661 	0.006286984 

## Confusion metrics

```
[[902 355]
 [360 923]] : is the confusion matrix

0.718503937007874 : is the accuracy score
0.7222222222222222 : is the precision score
0.7194076383476228 : is the recall score
0.7208121827411168 : is the f1 score

true positive = 902
false positive = 355
false negative = 360
true negative = 923
```

Given the confusion matrix  $\begin{bmatrix} 902 & 355 \\ 360 & 923 \end{bmatrix}$ , accuracy score 0.7185, precision score 0.722, recall score 0.7194 and f1 score 0.72081 of my logistic regression model.

I can conclude that the model performance is good but not excellent. The accuracy score indicates that about 72% of the predictions made by the model are correct, which is a good sign. The precision score of 0.722 means that 72% of the positive predictions made by the model are actually true positives, while the recall score of 0.719 indicates that 72% of the actual positive cases were correctly predicted by the model. Finally, the F1 score of 0.721 is a balance between precision and recall, which also indicates a good performance of the model.

However, there is still room for improvement, so I want to consider trying different algorithms to get better results.

## Decision Tree Classifier

### Loading of data

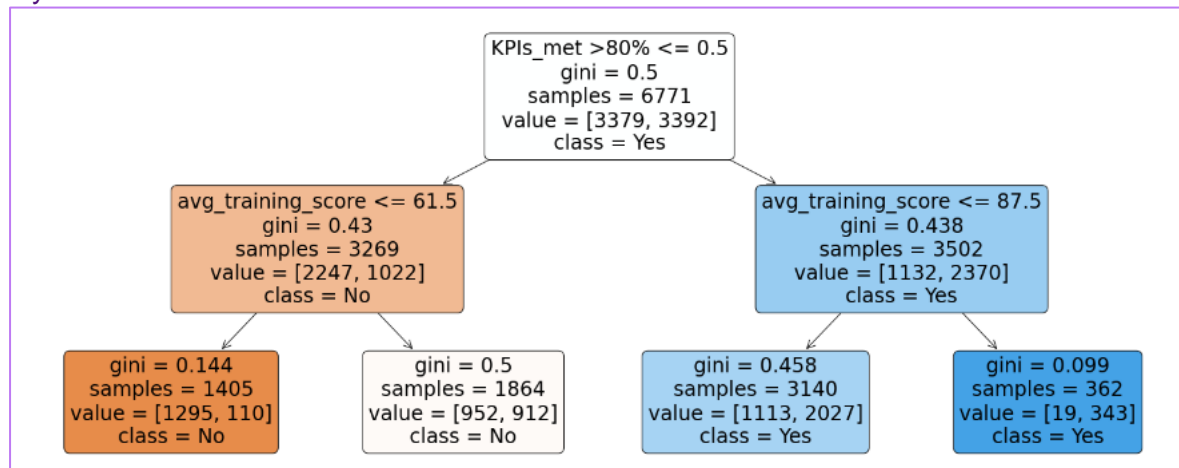
Defining my model Input(X) and Output (y)

```
y_data = df_hr_DecisionTree['is_promoted']
X_data = df_hr_DecisionTree.drop(['is_promoted'], axis=1)
```

### Building a Decision Tree model

```
# Create Decision Tree Model
decision_tree = tree.DecisionTreeClassifier(max_depth = 2)
decision_tree.fit(X_train, y_train)
```

My decision tree:



The first split at the root node of the tree is based on the feature "KPI\_met <= 0.5". The gini impurity of this split is 0.5, which means that roughly half of the employees in the sample have KPI\_met scores below 0.5 and half have scores above. This results in two branches with 6771 total samples, 3379 in the "yes" class and 3393 in the "no" class.

The first branch is based on "avg\_training score <= 61.5". This split has a gini impurity of 0.43 and includes 3269 samples. There are 2247 samples in the "no" class and 1022 samples in the "yes" class. This branch is then split into two further branches with gini impurities of 0.144 and 0.5, and 1405 and 1864 samples, respectively.

The second branch is based on "avg\_training score <= 87.5". This split has a gini impurity of 0.4838 and includes 3502 samples. There are 1132 samples in the "no" class and 2370 samples in the "yes" class. This branch is then split into two further branches with gini impurities of 0.458 and 0.099, and 3140 and 365 samples, respectively.

In this case, it appears to be performing well in terms of classifying samples into the "yes" and "no" classes. The low Gini impurities for some of the branches suggest that the tree is able to split the data into groups with a clear majority of samples belonging to a single class, which is a good sign.

Training and Testing accuracy:

```
#training accuracy
train_acc = decision_tree.score(X_train, y_train)
print('the training accuracy is: ', train_acc)

the training accuracy is:  0.6818785999113868

#testing acc
test_acc = decision_tree.score(X_test, y_test)
print('the testing accuracy is: ', test_acc)

the testing accuracy is:  0.6828115770821028
```

in this case, the difference between the training accuracy and testing accuracy is small, therefore the model is considered to be performing okay.

### Confusion metrics:

```
[[573 280]
 [257 583]] : is the confusion matrix

0.6828115770821028 : is the accuracy score
0.6755504055619931 : is the precision score
0.694047619047619 : is the recall score
0.6846741045214327 : is the f1 score

true positive = 573
false positive = 280
false negative = 257
true negative = 583
```

In conclusion, based on these evaluation metrics, the performance of the decision tree model is moderate. It is not exceptionally good, but also not very bad. Improving the performance may require tweaking the model's hyperparameters.

## Evaluate and improve the model.

### K-fold:

```
results = cross_validate(decision_tree, X_data, y_data, scoring='accuracy', cv=5, return_train_score = True)
print('train_score: ', results['train_score'])
print('test_score: ', results['test_score'])
print("Mean accuracy of training score :", np.mean(results['train_score']))
print("Mean accuracy of testing score :", np.mean(results['test_score']))

train_score: [0.68276473 0.68010634 0.67981096 0.68557082 0.68207324]
test_score: [0.67926757 0.68989959 0.69108092 0.66804489 0.6820331 ]
Mean accuracy of training score : 0.6820652171542531
Mean accuracy of testing score : 0.6820652135968017
```

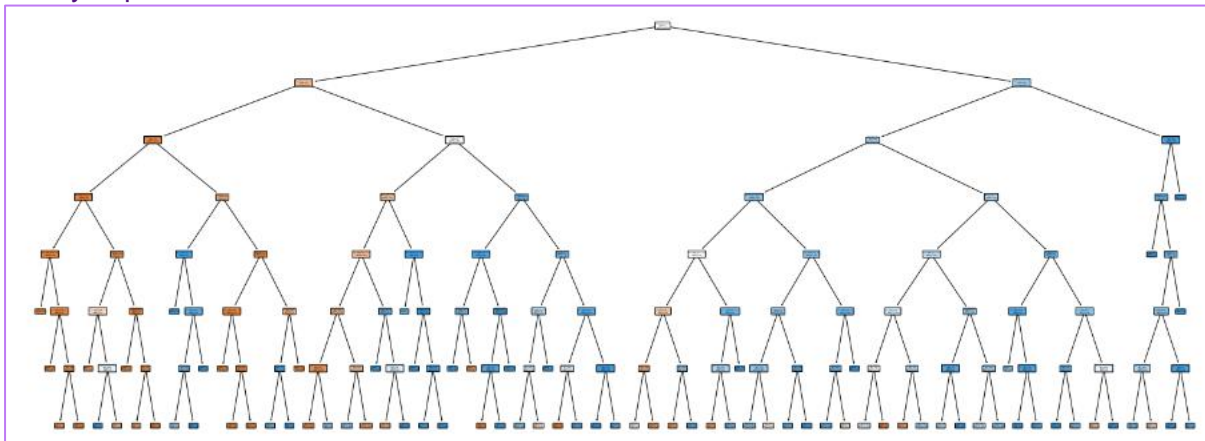
However, the range of accuracy on the testing data is from 66.80% to 69.10%. This variation between different folds suggests that the model might not have a high level of generalization.

### GridSearch:

```
param_grid = { "criterion" : ["gini", "entropy"],
               "min_samples_leaf" : [1, 5, 10],
               "min_samples_split" : [2, 4, 10, 12, 16],
               'max_depth' : [2, 3, 4, 5, 6, 7]}
```

So now I have a wide selection of criterion , min\_samples\_leaf, min\_sample\_split and max\_dept. The values that are boxed in blue are the suggested result my the GridSearch. So I trained my model based on these boxed values.

So my improved decision tree:



Since the dept is given as 7, the decision tree becomes more complex to analysis.

Training and testing accuracy:

```
train_acc = decision_tree.score(X_train, y_train)
print('The training accuracy is: ', train_acc)

The training accuracy is:  0.7984049623393886

#testing acc
test_acc = decision_tree.score(X_test, y_test)
print('The testing accuracy is: ', test_acc)

The testing accuracy is:  0.789722386296515
```

It seems that after using grid-search, the model's performance has improved. The training accuracy has increased from 0.6818 to 0.798 and the testing accuracy has increased from 0.68281 to 0.78972.

The training accuracy is 0.798404 and the testing accuracy is 0.7897, which is a relatively small difference. This suggests that the model is not overfitting and is generalizing well to new data.

Confusion Metrics:



```
[[567 286]
 [ 70 770]] : is the confusion matrix

0.789722386296515 : is the accuracy score
0.7291666666666666 : is the precision score
0.9166666666666666 : is the recall score
0.8122362869198312 : is the f1 score

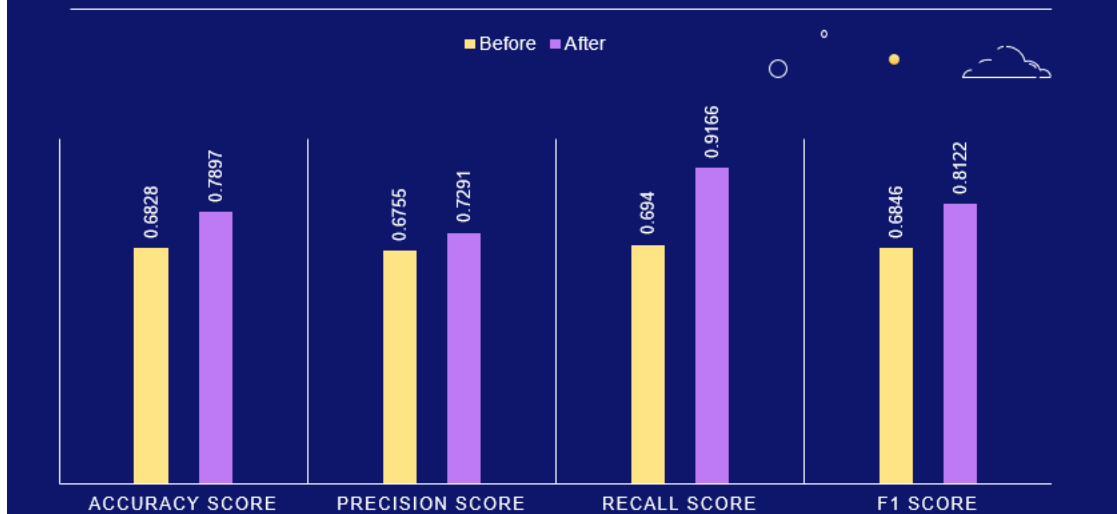
true positive = 567
false positive = 286
false negative = 70
true negative = 770
```

Based on the results, the updated model has a better performance than the previous model.

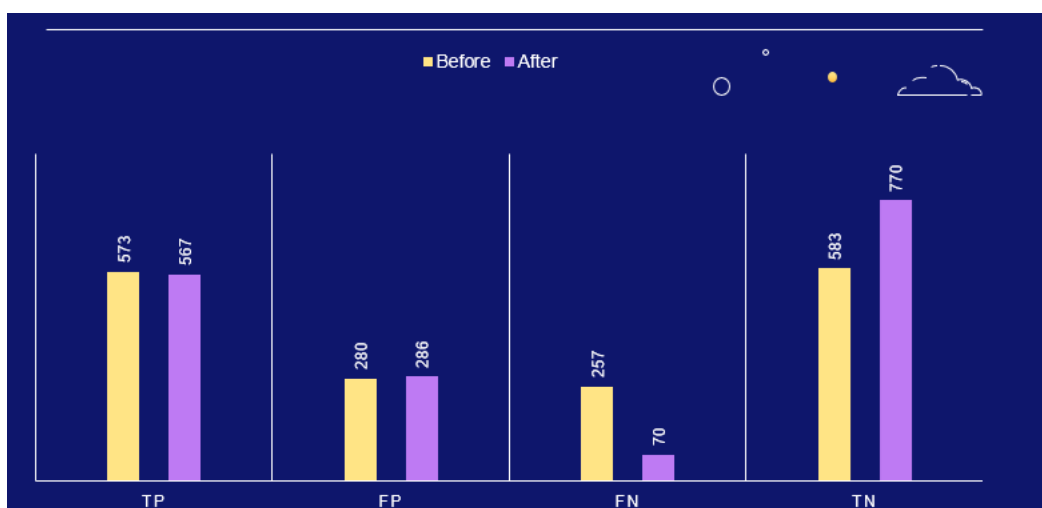
Overall Analysis:

	TRAINING ACCURACY	TESTING ACCURACY	MODEL
Before	0.6818785999113868	0.6828115770821028	0.000932977
After	0.7984049623393886 	0.789722386296515 	0.008682576 

## ANALYSIS RESULTS



This bar graph shows a comparison of scores from the model before improvement and model after improvement.



This bar graph shows a comparison of all cases from the model before improvement and model after improvement.

## Artificial Neural Network

### Loading of data

```
# Define Model Inputs (X) and Output (y)
y = df_hr_ann['is_promoted']
X = df_hr_ann.drop(['is_promoted'], axis=1)

# Split both Inputs (X) and Output (y) into training set (70%) and testing set (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

### Building a ANN model

```
# Create the ann model
mlp = MLPClassifier(activation='logistic', hidden_layer_sizes=(10,), max_iter=2000, solver='sgd', random_state=2)
# Fit the model to the training set
mlp.fit(X_train, y_train)
```

After building the model  
I am going to print their coefficient and intercepts.

```
print(mlp.coefs_)
print(mlp.intercepts_)
```

```
[array([[ -3.69882276e-02,  -2.85853491e-01,   2.99681031e-02,
         6.86169107e-01,   9.85015883e-02,  -1.02464246e-01,
        -4.31463353e-01,   2.55370725e-02,  -1.20053350e-01,
         8.46414795e-02],
        [ 7.37529045e-02,   1.75718843e-02,  -2.20322550e-01,
         1.22655060e-01,  -1.59004769e-01,   1.71963244e-01,
         1.73675624e-01,  -2.74107851e-02,   2.09373880e-01,
        -1.99694708e-01],
        [ 5.55116321e-03,  -2.62120326e-01,  -4.32932734e-02,
         4.25103451e-01,  -9.08483530e-02,   5.80653353e-02,
        -4.08084744e-01,  -3.02217798e-01,  -1.67320178e-01,
         5.67993883e-02],
        [ 4.04518966e-02,  -1.79840661e-01,   8.56467960e-02,
         5.70286866e-02,   1.36008717e-01,  -7.31276581e-02,
        -3.98555586e-02,   7.36368100e-02,  -1.65427518e-01,
         3.96528358e-02],
        [ 2.83356970e-01,   5.04167985e-06,   2.34935001e-01,
        -6.36098070e-01,  -3.91752948e-02,  -4.40659289e-02,
         1.00778636e-01,   2.56907483e-01,   2.37809391e-02,
         3.07968065e-02],
        [ 3.62719724e-02,  -2.51985310e-01,  -8.04068476e-02,
        -1.74927769e-02,  -8.52280408e-02,  -2.85676456e-01,
         5.23242183e-02,  -1.22606797e-01,   3.01876707e-01,
        -2.41723290e-02],
        [ 1.79955134e-01,   6.13929341e-02,   1.59743993e-01,
        -2.14122154e+00,  -4.62795737e-01,   1.46445169e-02,
         5.05265759e-01,  -1.05906308e-01,   2.90988913e-01,
        -6.45897583e-01]])
```

From the table, positive coefficients indicated a positive relationship between the feature and the target variable, and negative coefficients indicated negative relationship. The magnitude of the coefficients can be reflecting the importance of the feature to the model's prediction. However, the interpretation of coefficients in ANNs is more complex than in linear models, as ANNs are non-linear and may have multiple hidden layers that make it difficult to understand the relationship between the features and the target variable.

So next I will be printing the `y_fitted` and `fitted_probability` values.

```
print(y_fitted)
print(y_fitted_prob)
```

```
[1 0 1 ... 0 1 0]
[0.43434843 0.92897417 0.35609313 ... 0.51653423 0.39942794 0.83733208]
```

As seen here where `'y_fitted_prob' = [0.43434843 ,0.92897417 ,0.35609313 ... 0.51653423 , 0.39942794 ,0.83733208 ]`

This means that for the first observation, the model is 43.43% confident that the observation where employee is not promoted, for the second observation, the model is 92.89% confident and so on.

### The training and testing accuracy

```
0.7348075624577988 (Train Accuracy)
```

```
0.7244094488188977 (Test Accuracy)
```

In this case, the difference between the train accuracy (0.73480756) and the test accuracy (0.72440944) is not large, which suggests that the model is not significantly overfitting or underfitting.

However, model seems to have a moderate fit as the difference between train accuracy and test accuracy is small, with train accuracy being slightly higher. The train accuracy of 0.734 and test accuracy of 0.724 indicate that the model has learned some patterns from the training data, but it's not performing exceptionally well on unseen data (test set). There is room for improvement by making the model more complex or by improving the features and data.

## Confusion metrics

```
[[ 751  517]
 [ 183 1089]] : is the confusion matrix

0.7244094488188977 : is the accuracy score
0.678082191780822 : is the precision score
0.8561320754716981 : is the recall score
0.7567755385684504 : is the f1 score

true positive = 751
false positive = 517
false negative = 183
true negative = 1089
```

In conclusion, based on these evaluation metrics, the performance of the decision tree model is moderate. It is not exceptionally good, but also not very bad. Improving the performance may require tweaking the model's hyperparameters.

## Evaluate and improve the model.

Grid search:

```
param_grid = {"activation" : ["logistic", "relu"],
              "hidden_layer_sizes": [(5,), (10,), (20,)],
              "max_iter": [200, 2000, 4000],
              "solver": ["sgd", "adam"]}
```

So now I have a wide selection of activation , hidden\_layers\_size, max\_iter and solver. The values that are boxed in blue are the suggested result my the GridSearch. So, I trained my model based on these boxed values.

The final training and testing data:

```
0.7570898041863605 (Train Accuracy)
0.7480314960629921 (Test Accuracy)
```

The final accuracy values indicates that the model has improved after making some changes.

A train accuracy of 0.7570 and a test accuracy of 0.7480 suggests that the model is not overfitting, as the difference between train and test accuracy is relatively small and is considered to be a reasonably good performance.

Final Confusion metrics:




```
[[ 857  411]
 [ 229 1043]] : is the confusion matrix

0.7480314960629921 : is the accuracy score
0.717331499312242 : is the precision score
0.8199685534591195 : is the recall score
0.7652237710931766 : is the f1 score

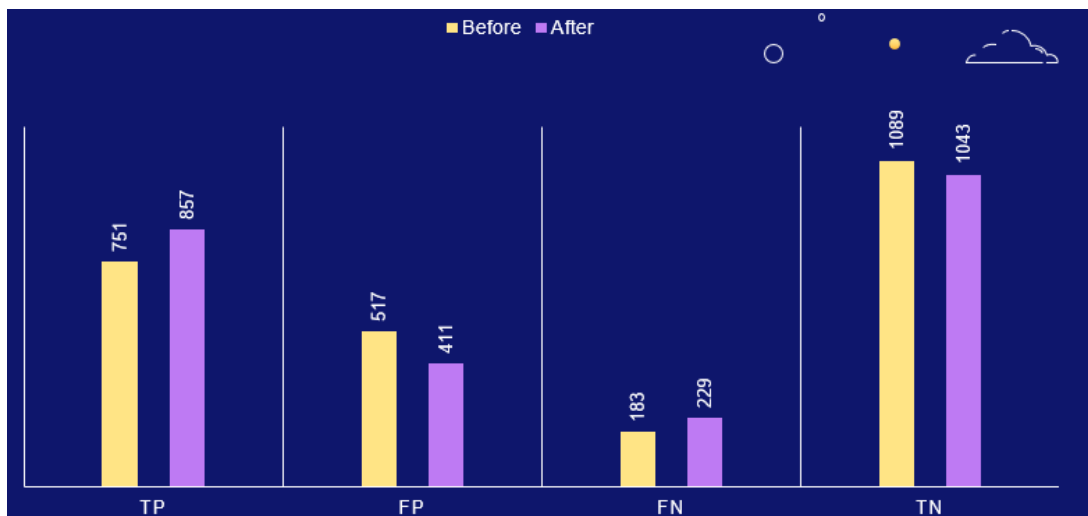
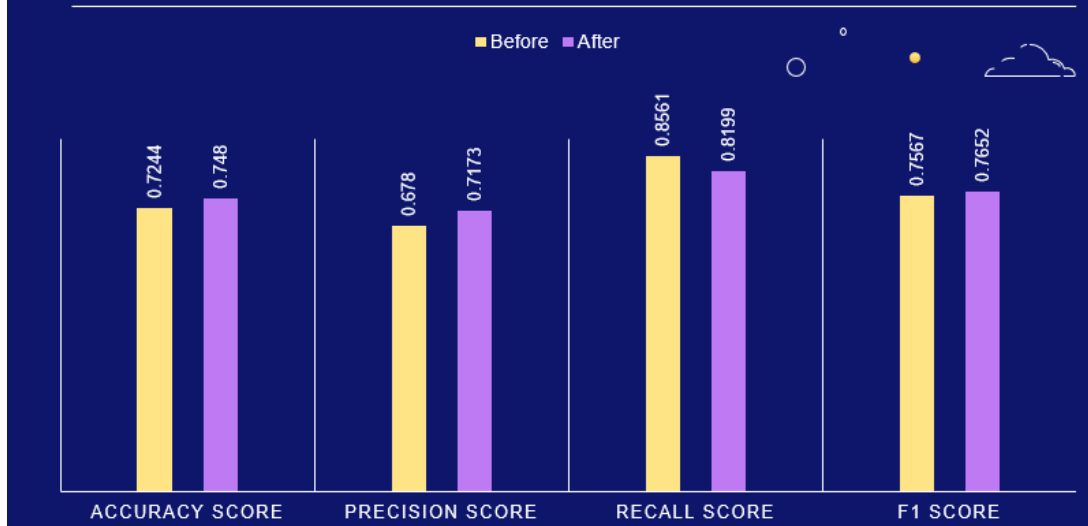
true positive = 857
false positive = 411
false negative = 229
true negative = 1043
```



### Overall Analysis:

	TRAINING ACCURACY	TESTING ACCURACY	MODEL
Before	0.7348075624577988	0.7244094488188977	0.010398113
After	0.7570898041863605 	0.7480314960629921 	0.009058308 

## ANALYSIS RESULTS



We can see that the number so TP increased while the TN have decreased. However, there is a slight increase in the number of FN that leads to a decrease in the number of FP.

# Random Forest

## Loading of data

```
df_y = df_hr_rf['is_promoted']  
df_x = df_hr_rf.drop(['is_promoted'], axis=1)  
  
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state = 4)
```

## Building a Random Forest model

```
rf = RandomForestClassifier(n_estimators = 10, max_depth=4, random_state=2)  
rf.fit(x_train, y_train)
```

The training and testing accuracy:

```
training accuracy is: 0.7548368040171319  
testing accuracy is: 0.7448316597755463
```

In this case, the difference between the train accuracy (0.75483) and the test accuracy (0.74483) is not large, which suggests that the model is not significantly overfitting or underfitting.

However, model seems to have a moderate fit as the difference between train accuracy and test accuracy is small, with train accuracy being slightly higher. The train accuracy and test accuracy indicate that the model has learned some patterns from the training data and is considered to be a reasonable performance.

There is room for improvement by making the model more complex or by improving the features and data.

Confusion Metrics:

```
[[567 308]  
 [124 694]] : is the confusion matrix  
  
0.7448316597755463 : is the accuracy score  
0.6926147704590818 : is the precision score  
0.8484107579462102 : is the recall score  
0.7626373626373626 : is the f1 score  
  
true positive = 567  
false positive = 308  
false negative = 124  
true negative = 694
```

In summary, the model has a decent accuracy and a relatively good balance between precision and recall, but there is room for improvement in both areas.

## Evaluate and improve the model.

K-fold:

```
results = cross_validate(rf, df_x, df_y, scoring='accuracy', cv=5, return_train_score = True)  
print('train_score: ', results['train_score'])  
print('test_score: ', results['test_score'])  
  
train_score: [0.74479397 0.75158765 0.74686162 0.74892926 0.73774365]  
test_score: [0.73892499 0.75487301 0.75723568 0.72888364 0.7393617 ]  
  
Cross val training accuracy is: 0.7459832301388177  
Cross val testing accuracy is: 0.7438558017368136
```

Grid Search:

```
param_grid = { "criterion" : ["gini", "entropy"],  
  "max_depth": [2, 4, 6, 8],  
  "min_samples_leaf" : [1, 5, 10],  
  "n_estimators": [10, 20, 50, 100]}
```

So now I have a wide selection of criterion ,max\_depth, min\_samples\_leaf and n\_estimators. The values that are boxed in blue are the suggested result my the GridSearch. So, I trained my model based on these boxed values.

Training and testing accuracy:

```
training acc: 0.782749963077832  
testing acc: 0.7531010041346722  
cross val training accuracy is: 0.7828745214184777  
cross val testing accuracy is: 0.7657141979420197
```

Feature Importance:

So I have printed all the features and its corresponding importance.

	feature	importance
6	KPIs_met >80%	0.326007
8	avg_training_score	0.265946
4	previous_year_rating	0.157972
7	awards_won?	0.082466
10	Department	0.055563
3	age	0.028377
9	Region	0.027793
5	length_of_service	0.024854
0	education	0.008356
2	no_of_trainings	0.008321
1	recruitment_channel	0.008089
11	Gender	0.006256

So i am going to remove the last three features as they are the least importance.

```
df_x = df_hr_rf.drop(['is_promoted', 'recruitment_channel', 'no_of_trainings', 'Gender'], axis=1)
```

After dropping the 3 features I have re-train the model with the same parameters obtained from grid search.

Final Training and testing accuracy:

```
training acc: 0.7879190666075911  
testing acc: 0.7826343768458358
```

These accuracies indicates that the model is better able to fit the training data and is also performing well on unseen data. An increase in both training accuracy and testing accuracy is a good indicator that the model is generalizing well to new, unseen data.

The gap between training accuracy and testing accuracy has decreased after the improvements, which is a good sign that the model is generalizing well to new data.

## Confusion metrics:

```
[[573 280]
 [ 88 752]] : is the confusion matrix

0.7826343768458358 : is the accuracy score
0.7286821705426356 : is the precision score
0.8952380952380953 : is the recall score
0.8034188034188035 : is the f1 score

true positive = 573
false positive = 280
false negative = 88
true negative = 752
```

The random forest classification model has improved after making some modifications. The confusion matrix shows that the number of true positive (TP) and true negative (TN) predictions have increased, while the number of false positive (FP) and false negative (FN) predictions have decreased.

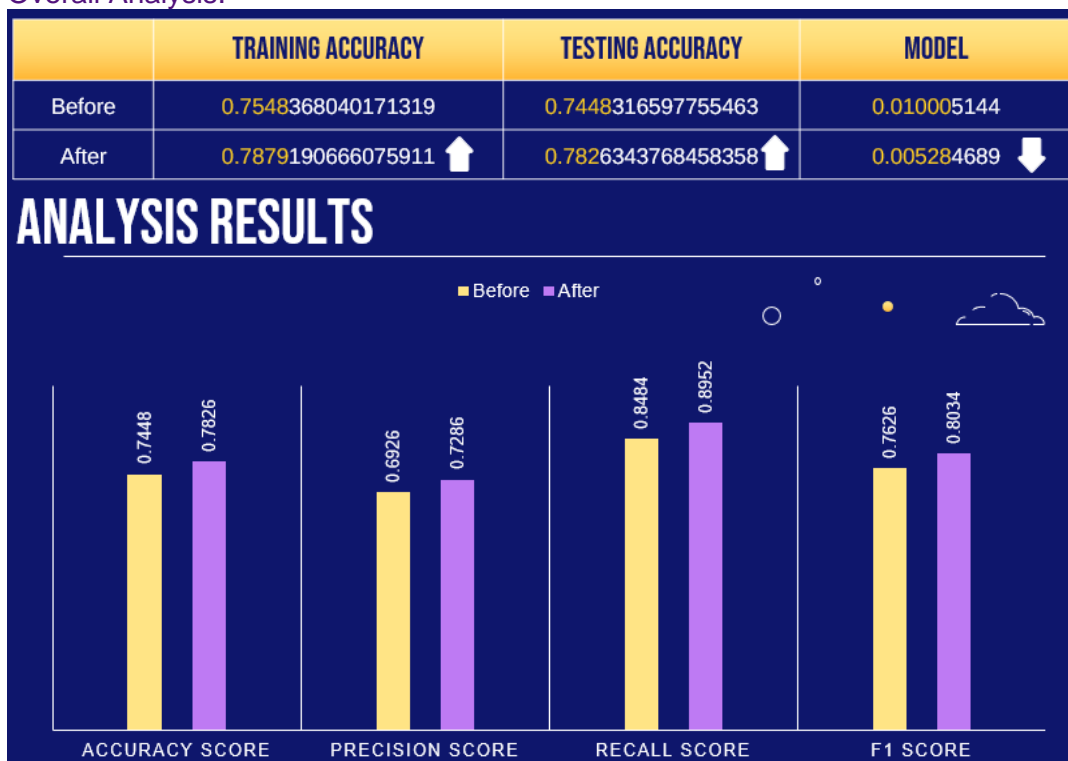
The accuracy score has increased from 0.7448 to 0.7826, indicating that the model is now able to correctly predict the target class for about 78.3% of the instances in the test set, which is a improvement from before.

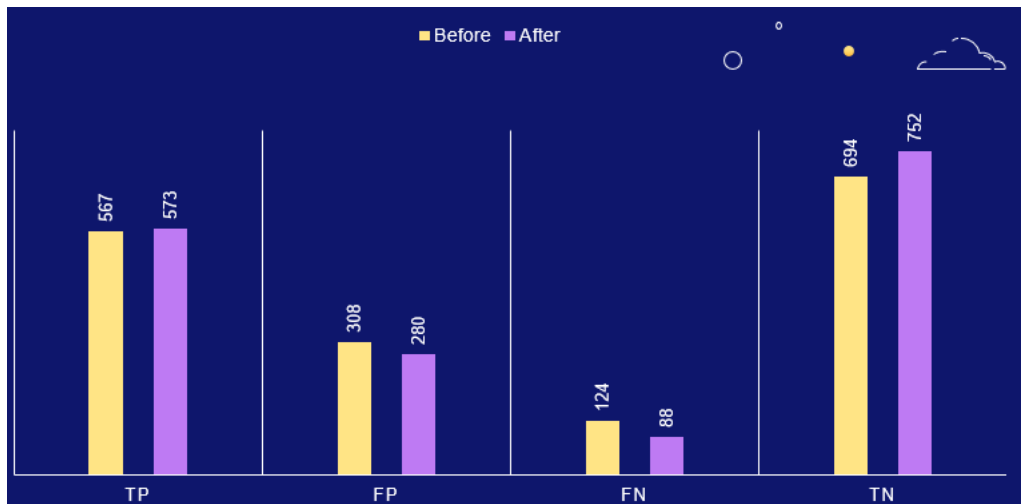
The precision score has increased from 0.6926 to 0.7287, meaning that the model is now more confident in its positive predictions and a larger proportion of its positive predictions are actually positive.

The recall score has increased from 0.8484 to 0.8952, indicating that the model is now able to identify a larger proportion of positive instances in the test set. This means that the model is able to capture more positive instances without sacrificing too much on precision.

The f1 score has increased from 0.7626 to 0.8034, which is a balance between precision and recall and indicates that the model has improved in both areas.

## Overall Analysis:





## Support Vector Machine

### Loading of data

```
# Define Model Inputs (X) and Output (y)
df_y = df_hr_svm['is_promoted']
df_x = df_hr_svm.drop(['is_promoted'], axis=1)

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state = 4)
```

### Building a SVM model

```
svc_model = svm.SVC(kernel='linear', gamma = 0.001, C = 0.1)
svc_model.fit(x_train, y_train)
```

The training and testing accuracy:

```
training accuracy is: 0.7090533156106926
testing accuracy is: 0.6952155936207915
```

In this case, with a training accuracy of 0.709 and a testing accuracy of 0.695, the difference between the two is small. This suggests that the model is not fitting the training data well, and is not generalizing well to unseen data. It could be due to a lack of complexity in the model, limited training data, or noisy or irrelevant features in the data.

So, even though the difference between the training accuracy and testing accuracy is small, the low accuracy scores indicate that the model is underfitting the data, and there is room for improvement.

### Confusion metrics:

```
[[570 305]
 [211 607]] : is the confusion matrix

0.6952155936207915 : is the accuracy score
0.6655701754385965 : is the precision score
0.7420537897310513 : is the recall score
0.7017341040462428 : is the f1 score

true positive = 570
false positive = 305
false negative = 211
true negative = 607
```

The confusion matrix provides additional information about the performance of the model. It shows that the model is making mistakes in both classifying positive cases as negative (false

negatives) and negative cases as positive (false positives). The precision score (0.665) indicates that when the model predicts a positive case, it is correct 66.6% of the time, while the recall score (0.742) indicates that the model correctly identifies 74.2% of the positive cases.

## Evaluate and improve the model.

Grid search :

```
param_grid = {'C': [0.1, 1],  
              'kernel': ['linear', 'poly', 'sigmoid'],  
              'gamma': [0.001]}
```

So now I have a selection of C and kernel. The values that are boxed in blue are the suggested result by the GridSearch. So, I trained my model based on these boxed values.

The final training and testing accuracy:

```
training accuracy is: 0.7353418992763255  
testing accuracy is: 0.7259303012404017
```




After making improvements, the training accuracy increased from 0.7090533156106926 to 0.7353418992763255 and the testing accuracy increased from 0.6952155936207915 to 0.7259303012404017. This indicates that the model has improved in terms of its ability to generalize to unseen data.

Confusion metrics:

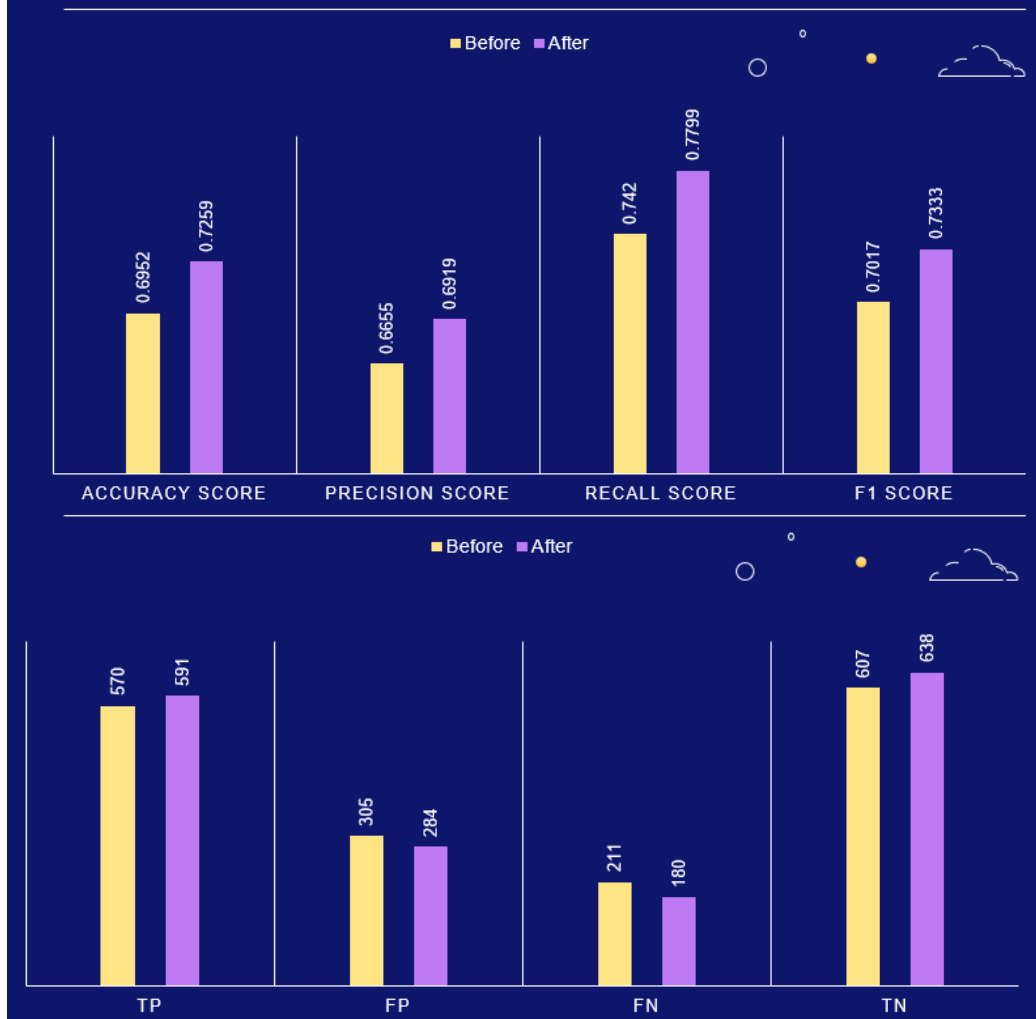
```
[[591 284]  
 [180 638]] : is the confusion matrix  
  
0.7259303012404017 : is the accuracy score  
0.6919739696312365 : is the precision score  
0.7799511002444988 : is the recall score  
0.7333333333333334 : is the f1 score  
  
true positive = 591  
false positive = 284  
false negative = 180  
true negative = 638
```

In this case, the F1-score is around 0.73, which is not very high. The precision score of 0.69 suggests that the model is not very good at identifying positive samples, while the recall score of 0.78 suggests that the model is relatively good at detecting positive samples.

Overall Analysis:

	TRAINING ACCURACY	TESTING ACCURACY	MODEL
Before	0.7090533156106926	0.6952155936207915	0.013837721
After	0.7353418992763255 	0.7259303012404017 	0.009411598 

# ANALYSIS RESULTS



## XGBoost

### Loading of data

```
# Define Model Inputs (X) and Output (y)
df_y = df_hr_XGB['is_promoted']
df_x = df_hr_XGB.drop(['is_promoted'], axis=1)

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state = 4)
```

### Building a XGB model

```
# Build XGBoost Model and Evaluate both training and testing accuracy
xgb = XGBClassifier(n_estimators = 20, learning_rate = 0.1, use_label_encoder = False, eval_metric='logloss')
```

Training and testing accuracy:

```
training accuracy is: 0.8153891596514548
testing accuracy is: 0.7944477259303012
```

In this case, although the gap between the accuracy is small, the fact that training accuracy is higher than the testing accuracy indicates that the model is overfitting. This means that the model is not generalizing well to new data, and is likely too complex for the amount of training data that it is being trained on. A good fit would have seen a smaller gap or similar

performance on both training and testing data.

### Confusion Metrics:

```
[[586 289]
 [ 59 759]] : is the confusion matrix

0.7944477259303012 : is the accuracy score
0.7242366412213741 : is the precision score
0.9278728606356969 : is the recall score
0.8135048231511255 : is the f1 score

true positive = 586
false positive = 289
false negative = 59
true negative = 759
```

However, the additional evaluation metrics do suggest that the model is performing relatively well. The precision, recall, and F1 score all have relatively high values, indicating that the model is making good predictions on the positive class and has a good balance of precision and recall.

## Evaluate and improve the model.

Grid search :

```
param_grid = {
    'max_depth': [5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 500],
    'use_label_encoder': [False],
    'eval_metric': ['logloss']
}
```

So now I have a selection of parameters. The values that are boxed in blue are the suggested result by the GridSearch. So, I trained my model based on these boxed values.

### Final Training and testing accuracy

```
training accuracy is: 0.8961748633879781
testing accuracy is: 0.8038984051978736
```

In this case, the training accuracy is 0.896 (89.6%) and the testing accuracy is 0.803 (80.3%). The gap between the two has increased, which suggests that the model is once again memorizing the training data rather than learning the underlying patterns and relationships in the data.

### Confusion Metrics:

```
[[644 231]
 [101 717]] : is the confusion matrix

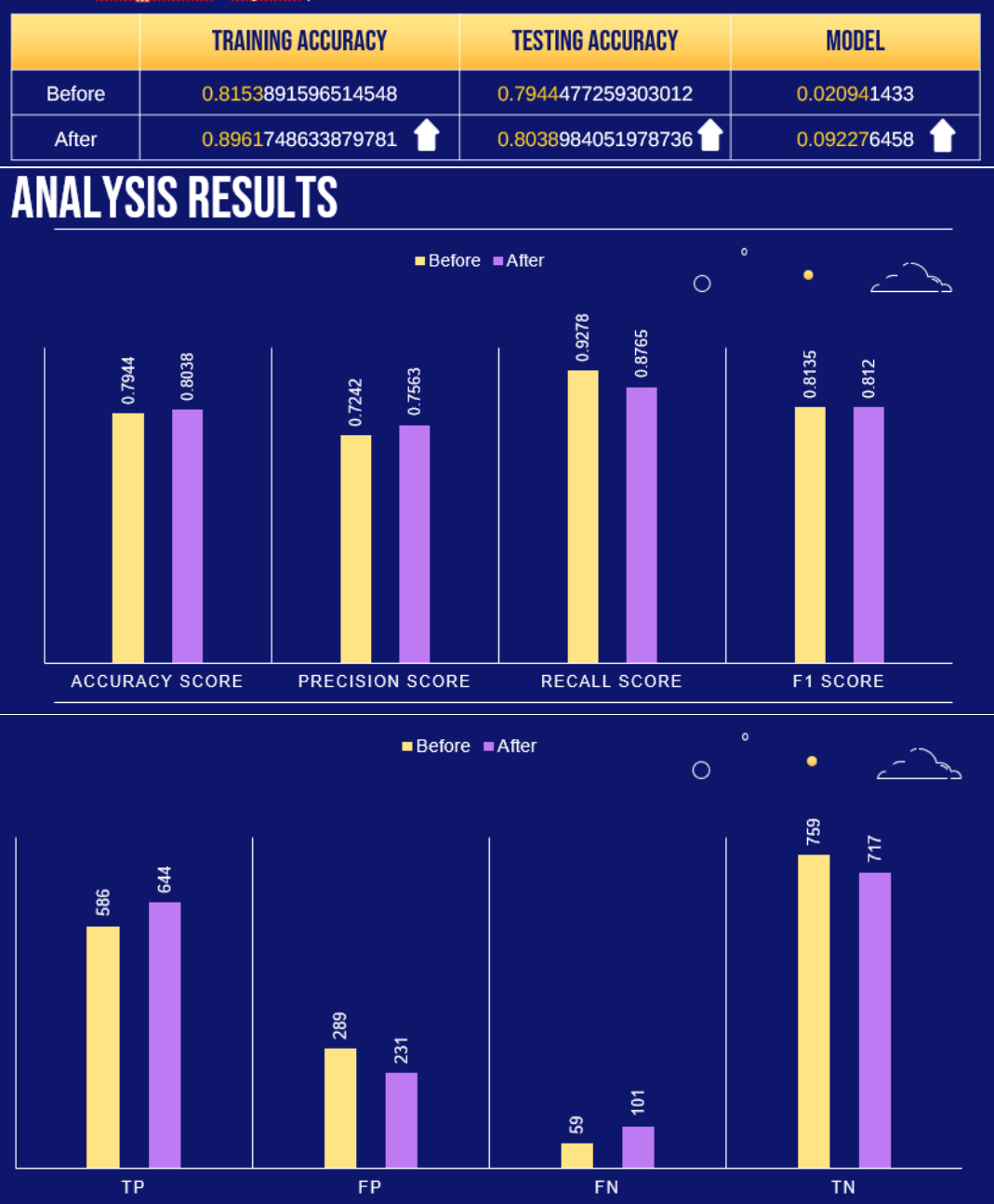
0.8038984051978736 : is the accuracy score
0.7563291139240507 : is the precision score
0.8765281173594132 : is the recall score
0.812004530011325 : is the f1 score

true positive = 644
false positive = 231
false negative = 101
true negative = 717
```



However, other metrics suggest that the model is still performing relatively well in terms of precision, recall, and F1 score. So, it's possible that the overfitting is not as severe as it was before, but it still exists to some degree.

Overall analysis:



## Voting

### Loading of data

```
# Define model inputs (x) and output (y)
df_y = df_hr_voting['is_promoted']
df_x = df_hr_voting.drop(['is_promoted'], axis=1)

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state = 4)
```

## Building a Voting model

```
xgb = XGBClassifier(n_estimators = 100, learning_rate = 0.1, use_label_encoder = False, eval_metric='logloss',max_depth= 7)
dt = DecisionTreeClassifier(criterion='gini', max_depth = 7, min_samples_leaf = 5, min_samples_split =4, random_state=1)
mlp = MLPClassifier(activation = 'logistic', hidden_layer_sizes=(20,), max_iter= 2000, solver = 'adam', random_state=2)

evc=VotingClassifier(estimators= [('xgb',xgb), ('dt',dt), ('mlp', mlp)], voting = 'hard')
```

Voting model is created by using three models: XGB, decision tree and MLP classifier.

Training and Testing accuracy:

```
training accuracy is:  0.8387239698715109
testing accuracy is:  0.7974010632014176
```

The training accuracy is 0.839 (83.9%) and the testing accuracy is 0.797 (79.7%). The gap between the two is relatively small, which suggests that the model is generalizing well to unseen data.

Confusion metrics:

```
[[622 253]
 [ 90 728]] : is the confusion matrix

0.7974010632014176 : is the accuracy score
0.7420998980632009 : is the precision score
0.8899755501222494 : is the recall score
0.8093385214007783 : is the f1 score

true positive = 622
false positive = 253
false negative = 90
true negative = 728
```

This shows that model is making some errors in both the positive and negative classes. However, the additional metrics suggest that the model is performing well overall. The precision score is 0.742, indicating that the model is making accurate positive predictions. The recall score is 0.89, indicating that the model is identifying positive instances well. And the F1 score is 0.809, indicating that the model has a good balance of precision and recall.

Overall, it appears that the hard voting classifier is a good fit for the data.

## Bagging

### Loading of data

```
# copying the dataframe into another dataframe that w
df_hr_bagging = df_hr.copy()

y_data = df_hr_bagging['is_promoted']
X_data = df_hr_bagging.drop(['is_promoted'], axis=1)
```

## Building a Bagging model

```
# Split the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=2)

kfold = model_selection.KFold(n_splits = 10)

bg = BaggingClassifier(DecisionTreeClassifier(criterion='gini', max_depth = 7, min_samples_leaf = 5,
, n_estimators=100 ,random_state = 0)
bg.fit(X_train, y_train)
```

### Training and Testing accuracy:

```
training accuracy is: 0.8000295377344557
testing accuracy is: 0.7974010632014176
```

training accuracy indicates that 80% of the instances in the training data were correctly classified by bagging classifier model. This is a good indication that the model has learned the relationship between the features and the target variable in the training data.

the testing accuracy indicates that bagging classifier model correctly classified about 79.74% of the instances in the testing data. This is a good indication that the model has good generalization ability and can make accurate predictions on new data.

### Confusion metrics:

```
[[563 290]
 [ 53 787]] : is the confusion matrix

0.7974010632014176 : is the accuracy score
0.7307335190343547 : is the precision score
0.9369047619047619 : is the recall score
0.821074595722483 : is the f1 score
```

True positive = 563

False positive = 290

False negative = 53

True negative = 787

Bagging Classifier model seems to be performing well. The accuracy score of 0.7974 is decent, which means that the model is able to correctly predict the target class about 80% of the time.

## Evaluate and improve the model.

### Grid search :

```
param_grid = {'base_estimator__max_depth': [5, 7, 9],
              'base_estimator__min_samples_leaf': [3, 5, 7],
              'base_estimator__min_samples_split': [2, 4, 6],
              'n_estimators': [50, 75, 100]}
```

So now I have a selection of parameters. The values that are boxed in blue are the suggested result by the GridSearch. So, I trained my model based on these boxed values.

### Final Training and testing accuracy

```
training accuracy is: 0.8399054792497416
testing accuracy is: 0.819846426461902
```

The improved model has a better fit than the previous model, with a training accuracy of 0.8399054792497416 and a testing accuracy of 0.819846426461902. The improvement in accuracy can be attributed to the changes made in the Decision Tree Classifier, such as increasing the max\_depth and reducing the min\_samples\_leaf and min\_samples\_split, which can help the model generalize better to the test data.

### Confusion metrics:

```
[[596 257]
 [ 48 792]] : is the confusion matrix

0.819846426461902 : is the accuracy score
0.7550047664442326 : is the precision score
0.9428571428571428 : is the recall score
0.8385389094759131 : is the f1 score
```

True positive = 596

False positive = 257

False negative = 48

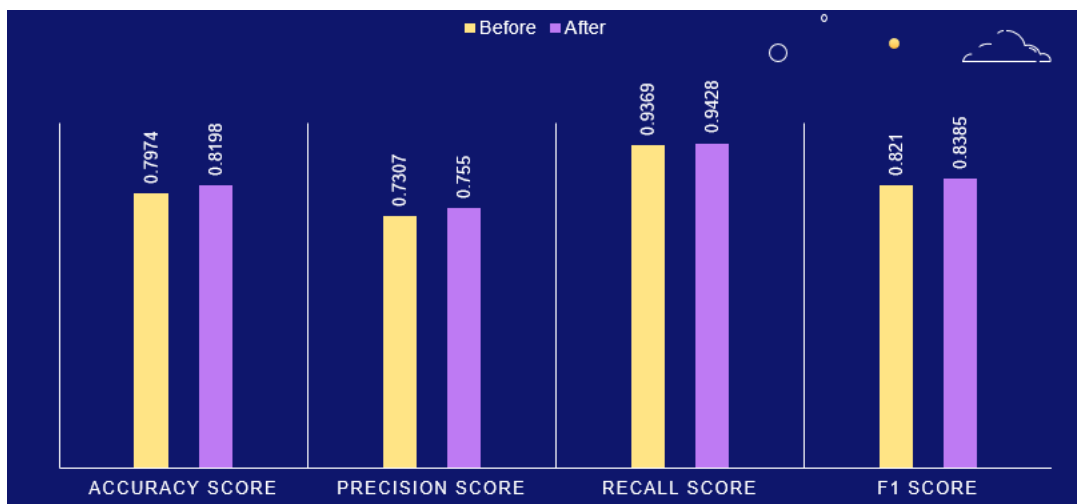
True negative = 792

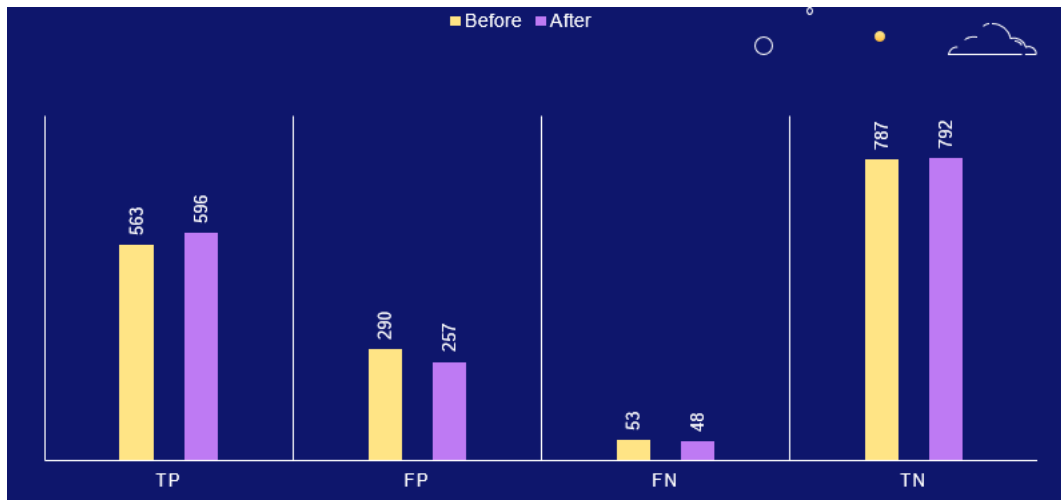
The precision score also increased, which indicates that the improved model has a better balance between the TP and FP , while still maintaining a high recall score, indicating that the model has high TP rate.

Overall, the improved model has a better fit than the previous model and seems to be a better classifier based on the evaluation metrics.

### Overall analysis:

	TRAINING ACCURACY	TESTING ACCURACY	MODEL
Before	0.80002	0.79740	0.00262
After	0.83990	0.81984	0.02005





## Ada Boost

### Loading of data

```
#copying the dataframe into another dataframe that
df_hr_adaBoost = df_hr.copy()

df_y = df_hr_adaBoost['is_promoted']
df_x = df_hr_adaBoost.drop(['is_promoted'], axis=1)
```

### Building a ADB model

```
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state = 4)

abd = AdaBoostClassifier(DecisionTreeClassifier(max_depth=3), n_estimators = 20, learning_rate =0.1)
abd.fit(x_train, y_train)

AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                    learning_rate=0.1, n_estimators=20)
```

### Training and testing accuracy

```
training accuracy is: 0.7669472751439964
testing accuracy is: 0.7531010041346722
```

### Confusion metrics:

```
[[574 301]
 [117 701]] : is the confusion matrix

0.7531010041346722 : is the accuracy score
0.6996007984031936 : is the precision score
0.8569682151589242 : is the recall score
0.7703296703296704 : is the f1 score

True positive = 574
False positive = 301
False negative = 117
True negative = 701
```

The difference between the training accuracy (0.7669) and the testing accuracy (0.7531) is small, which is indicative of a model that may not be overfitting the data.

In terms of the evaluation metrics, the accuracy of 0.7531 is relatively low, which means that the model is only able to correctly predict the target variable around 75% of the time. So it is considered a decent accuracy.

## Evaluate and improve the model.

Grid search :

```
param_grid = {  
    'n_estimators': [50, 100, 150],  
    'learning_rate': [0.1, 0.5, 1],  
    'base_estimator': [DecisionTreeClassifier(max_depth=2),  
                       DecisionTreeClassifier(max_depth=3),  
                       DecisionTreeClassifier(max_depth=4)]  
}
```

So now I have a selection of parameters. The values that are boxed in blue are the suggested result by the GridSearch. So, I trained my model based on these boxed values.

## Final Training and testing accuracy

```
training accuracy is: 0.8373947718210013  
testing accuracy is: 0.7968103957471944
```

## Confusion metrics:

```
[[604 271]  
 [ 73 745]] : is the confusion matrix  
  
0.7968103957471944 : is the accuracy score  
0.7332677165354331 : is the precision score  
0.910757946210269 : is the recall score  
0.8124318429661942 : is the f1 score  
  
True positive = 604  
  
False positive = 271  
  
False negative = 73  
  
True negative = 745
```

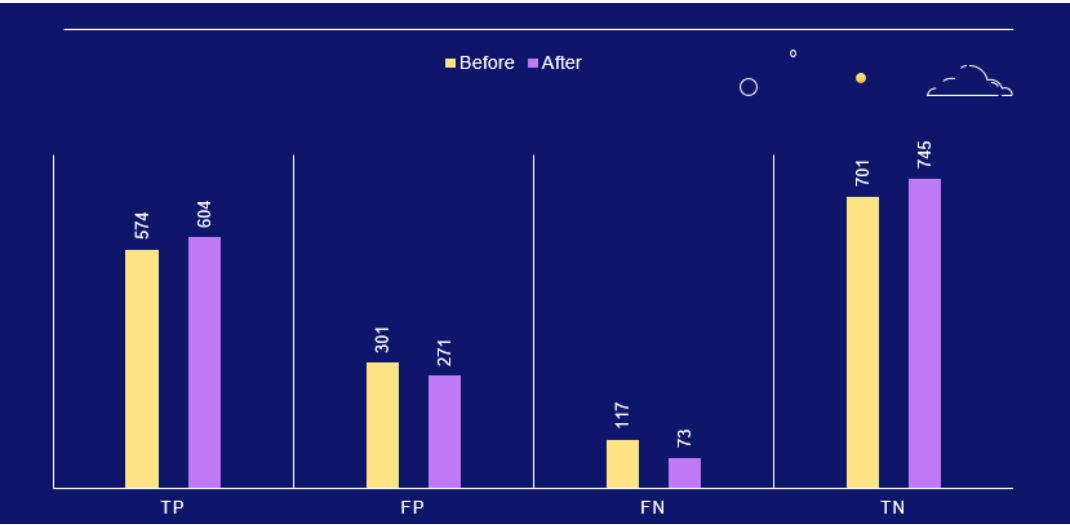
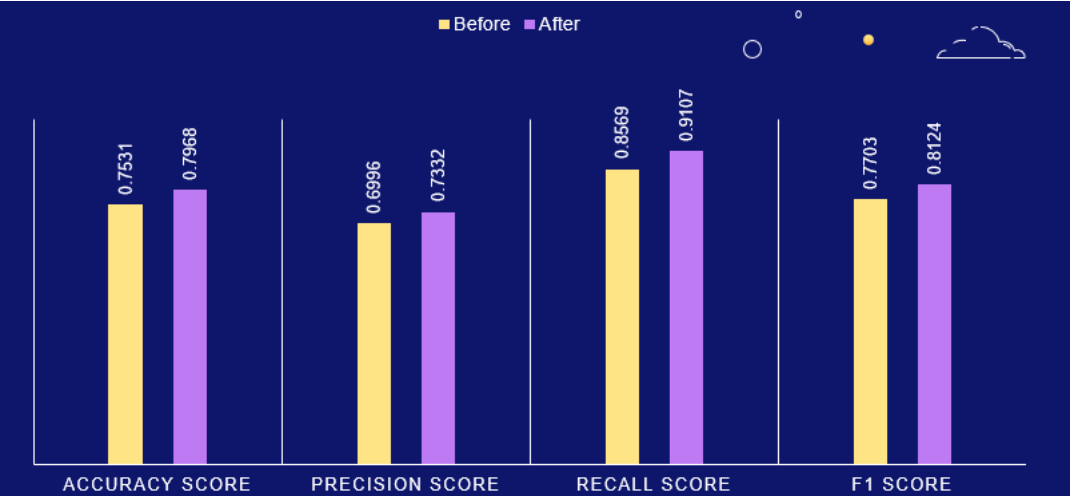
By increasing the maximum depth of the decision tree base estimator and the number of estimators in the AdaBoost algorithm, I have provided the model with more opportunities to learn from the training data. As a result, the training accuracy has improved, which indicates that the model has learned the patterns in the training data more effectively.

In turn, this has also led to an improvement in the testing accuracy, which suggests that the model is now making more accurate predictions on unseen data. The confusion matrix shows that there are fewer false positives and false negatives, indicating that the model is making fewer incorrect predictions.

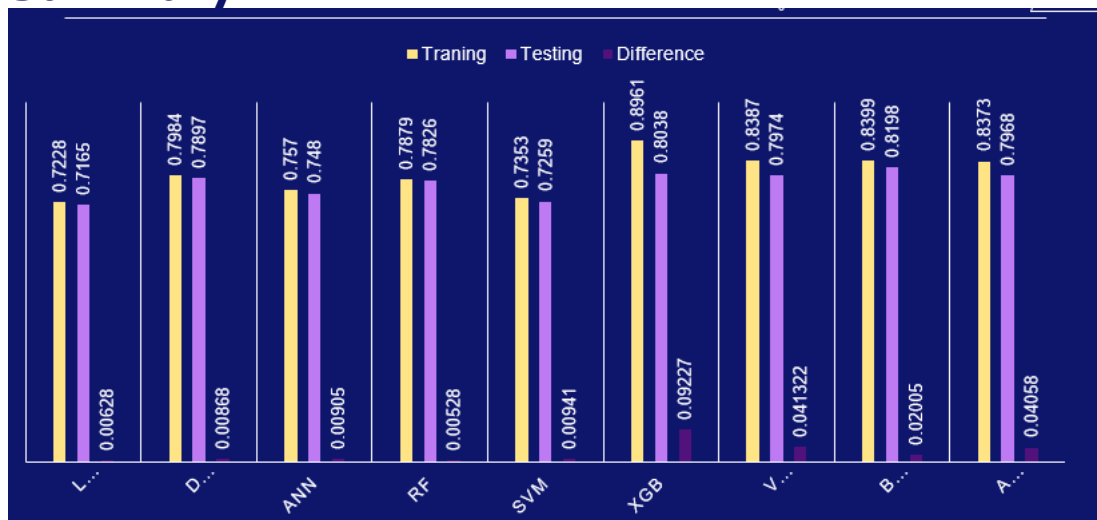
Overall, the improvement in accuracy and the confusion matrix suggest that the model is now a better fit for the data.

Overall analysis:

	TRAINING ACCURACY	TESTING ACCURACY	MODEL
Before	0.76694	0.75310	0.01384
After	0.83739	0.79681	0.04058



# Summary



## The top three models are:

XGBoost: has the highest testing accuracy (0.8038) among all the models, and it is also the model with the highest difference between training and testing accuracy (0.09227), which indicates that it has low overfitting.

Voting Classifier: has the second highest testing accuracy (0.7974) among all the models and has a relatively low difference between the training and testing accuracy (0.041322), which also indicates low overfitting.

Ada Boost: has a testing accuracy of 0.7968 and a difference between the training and testing accuracy of 0.04058, which makes it the third best model in terms of performance.

## Other metrics:

When comparing these three models, XGBoost has the highest accuracy score (0.803898), precision score (0.756329), and F1 score (0.8120045) among all the models. This indicates that XGBoost has a good balance between precision and recall and is able to make accurate predictions while avoiding false negatives and false positives.

The voting classifier has a relatively high accuracy score (0.797401), precision score (0.742099), and F1 score (0.809338), which indicates that it also has a good balance between precision and recall.

Ada Boost has a relatively high recall score (0.910757), which indicates that it is able to identify a high number of positive instances, but with a lower precision score compared to the other two models.

## Ranking:

Based on these metrics, the XGB would be considered the best fit, followed by Voting, and then the Ada Boost.





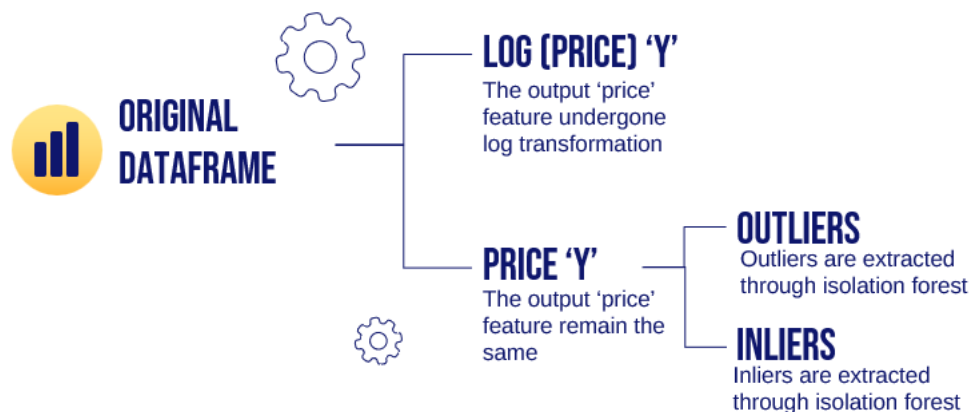
## AirBnb

The objective is to apply machine learning modelling in predicting the rental price of the listed properties. Using the dataset that contains listing activity and metrics from year 2013 to 2019. The data file includes the hosts information, the condition of listed properties, the reviews and etc. . I will use these variables to train multiple models and determine the pricing.

Models that I will using:

- 1) Linear regression
- 2) Decision tree
- 3) Artificial natural network ( mlp regressor)
- 4) Random Forest
- 5) AdaBoost
- 6) extreme Gradient Boosting

### Splitting the dataframe



For this, I have chosen a unique approach. I have created a 4 different dataframe and trained then through all the models. The first dataframe would be my original dataframe and from here I have split it into inliers which is the second dataframe and outliers that was the third. I have used a approach called isolation forest to help me identify outliers and inliers. The last dataframe is quite special in a sense that I have used a log function on the price feature. I have done this for my assignment one when cleaning the data as I realised that price was skewed.

# Isolation Forest

## Build the model

```
# Initiate isolation forest
isolation = IsolationForest(n_estimators=100,
                             contamination='auto',
                             max_features=1.0
                             )

# Fit and predict
isolation.fit(df_Airbnb_ir)
outliers_predicted = isolation.predict(df_Airbnb_ir)

# Address outliers in a new column
df_Airbnb_ir['outlier'] = outliers_predicted
```

## Split the data

```
df_Airbnb_ir['outlier'].value_counts()

1      6688
-1     1202
Name: outlier, dtype: int64
```

So the -1 indicates outliers while 1 indicates inliers.

## Assigning them to individual dataframe

```
# assigning df_Airbnb_inliers as inliers is 1
df_Airbnb_inliers=df_Airbnb_ir[df_Airbnb_ir['outlier']==1]
df_Airbnb_inliers.head()

# assigning df_Airbnb_outliers as outliers is -1
df_Airbnb_outliers=df_Airbnb_ir[df_Airbnb_ir['outlier']==-1]
df_Airbnb_outliers.head()
```

# Linear Regression

## Loading of data

```
# Define Model Inputs (X) and Output (y)
X = df_Airbnb_linear.drop('price', axis =1)
y = df_Airbnb_linear["price"]

# Split both Inputs (X) and Output (y) into training set (70%) and testing set (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
```

# Normal Dataframe

## Building a LR model

```
lm = LinearRegression()
lm.fit(X_train, y_train)
```

Building a very simple model.

## RMSE and R<sup>2</sup> Value:

```
153.76323838705198 (Train RMSE)
0.2897691614451978 (Train R^2 Value)

180.98589418820453 (Test RMSE)
0.32252147754467586 (Test R^2 Value)
```

Here, the train RMSE is 153.76, which means that on average, the model's predictions deviate by 153.76 units from the actual values. The test RMSE of 180.99 is a bit higher, which suggests that the model is not performing as well on the test data compared to the training data.

An  $R^2$  value of 0.29 for the training data suggests that the model explains 29% of the variance in the dependent variable. The  $R^2$  value of 0.32 for the test data is slightly higher, which is good, but still not a very high value.

In conclusion, based on these values, it appears that the model is not performing very well. Improvements have to be made.

## Evaluate and improve the model.

### Summary

	coef	std err	t	P> t	[0.025	0.975]
minimum_nights	-0.6945	0.057	-12.115	0.000	-0.807	-0.582
number_of_reviews	-0.3672	0.095	-3.876	0.000	-0.553	-0.181
reviews_per_month	3.7464	2.702	1.387	0.166	-1.550	9.043
calculated_host_listings_count	-0.2340	0.036	-6.502	0.000	-0.305	-0.163
availability_365	0.1163	0.015	7.822	0.000	0.087	0.145
DaysOfMonth	1.1801	0.238	4.955	0.000	0.713	1.647
DaysOfWeek_int	0.5270	1.193	0.442	0.659	-1.811	2.865
year	0.0328	0.005	6.078	0.000	0.022	0.043
Month_Int	0.3492	1.052	0.332	0.740	-1.713	2.411
Total_price	0.0030	8.45e-05	35.821	0.000	0.003	0.003
Neighbourhood_group	-5.5242	1.828	-3.023	0.003	-9.107	-1.942
Room_type	59.0500	2.349	25.141	0.000	54.445	63.654

Since these two features are the only ones with the highest p value. So, I will be removing these features and retrain the model.

```
X = df_Airbnb_linear.drop(['Month_Int', 'DaysOfWeek_int', 'price'], axis = 1)
y = df_Airbnb_linear["price"]
```

### Final RSME and $R^2$ Value:

153.76323838705198 (Train RMSE)  
0.289709464741989 (Train  $R^2$  Value)

180.95129051867542 (Test RMSE)  
0.289709464741989 (Test  $R^2$  Value)

slight improvements in the performance of the model. In conclusion, while the improvements I have made to the model are small, they are still positive and indicate that the model is making slightly better predictions on the test data.

The high RSME values suggest that the model cannot capture the underlying pattern in the data. We can say that it has a high bias.

Overall, the linear regression model seems to have a poor fit with the data, as the RMSE values and R-squared values are relatively high. This might indicate that the model is not capturing the underlying pattern in the data or that the data is not linear.

# Linear Regression –Inliers and Outliers

## Dataframe

### Building a LR model

```
# Define Model Inputs (X) and Output (y)
X = df_Airbnb_inliers_linear.drop(['price', 'Month_Int', 'DaysOfWeek_int'], axis =1)
y = df_Airbnb_inliers_linear["price"]

1.1.8 Build the Model

# Split both Inputs (X) and Output (y) into training set (70%) and testing set (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

# train the model using training set
lm = LinearRegression()
lm.fit(X_train, y_train)
```

For this, I have removed the same 2 features and train the model.

Inliers:

RMSE and R<sup>2</sup> Value:

```
129.49437113500667 (Train RMSE)
0.2211761327390489 (Train R^2 Value)
```

```
118.22475405952777 (Test RMSE)
0.24309145456262782 (Test R^2 Value)
```

The values after removing outliers suggest that the performance of the model has improved.

Train RMSE value of 129.49 is considered high and suggests that the model is not fitting the training data well. This is generally an indicator that the model is not capturing the underlying patterns in the data accurately.

Outliers:

RMSE and R<sup>2</sup> Value:

```
309.6093147808511 (Train RMSE)
0.38830184167409865 (Train R^2 Value)
```

```
169.02713036735102 (Test RMSE)
0.6655171595192176 (Test R^2 Value)
```

Train RMSE value of 309.60 is much lower than the test RMSE value of 169.02, and the train R-squared value of 0.30 is much higher than the test R-squared value of 0.66. This suggests that the model is memorizing the training data and not generalizing well to new, unseen data. In summary, it appears that the model is overfitting.

## Evaluate and improve the model.

### Summary

	coef	std err	t	P> t	[0.025	0.975]
minimum_nights	-0.9834	0.176	-5.577	0.000	-1.329	-0.637
number_of_reviews	-0.8041	0.292	-2.757	0.006	-1.377	-0.232
reviews_per_month	-1.4266	9.860	-0.145	0.885	-20.781	17.928
calculated_host_listings_count	-0.4892	0.149	-3.278	0.001	-0.782	-0.196
availability_365	0.0153	0.082	0.188	0.851	-0.145	0.176
DaysOfMonth	4.1342	1.159	3.566	0.000	1.858	6.410
year	0.1053	0.018	6.005	0.000	0.071	0.140
Total_price	0.0031	0.000	18.847	0.000	0.003	0.003
Neighbourhood_group	-33.1495	7.158	-4.631	0.000	-47.199	-19.100
Room_type	53.6596	12.506	4.291	0.000	29.112	78.208

Since these two features are the only ones with the highest p value. So, I will be removing these features and retrain the model.

### Final RMSE and R<sup>2</sup> Value outliers:

309.6093147808511 (Train RMSE)  
0.3806382486574922 (Train R<sup>2</sup> Value)

168.93475122693337 (Test RMSE)  
0.6658826723629008 (Test R<sup>2</sup> Value)

It appears that the model is still overfitting, although the improvement in the model has reduced the degree of overfitting.

The reason for the high RMSE is caused by heavily skewed output feature (the dependent variable or "y") can also contribute to a high RMSE in a linear regression model.

## Linear Regression –Log Dataframe

### Building a LR model

```
lm = LinearRegression()  
lm.fit(X_train, y_train)
```

Building a very simple model.

### RMSE and R<sup>2</sup> Value:

0.5907564859304365 (Train RMSE)  
0.39374051607746985 (Train R<sup>2</sup> Value)

0.6078035784224346 (Test RMSE)  
0.37572340022854656 (Test R<sup>2</sup> Value)

It appears that the linear regression model after log transformation has a decent fit.

## Evaluate and improve the model.

### Summary

minimum_nights	-0.0024	0.000	-10.954	0.000	-0.003	-0.002
number_of_reviews	-0.0017	0.000	-4.634	0.000	-0.002	-0.001
reviews_per_month	0.0301	0.010	2.899	0.004	0.010	0.050
calculated_host_listings_count	-0.0005	0.000	-3.753	0.000	-0.001	-0.000
availability_365	0.0005	5.71e-05	8.646	0.000	0.000	0.001
DaysOfMonth	0.0049	0.001	5.310	0.000	0.003	0.007
DaysOfWeek_int	0.0005	0.005	0.114	0.909	-0.008	0.010
year	0.0021	2.08e-05	98.927	0.000	0.002	0.002
Month_int	0.0055	0.004	1.350	0.177	-0.002	0.013
Total_price	3.491e-06	3.24e-07	10.759	0.000	2.86e-06	4.13e-06
Neighbourhood_group	-0.0408	0.007	-5.812	0.000	-0.055	-0.027
Room_type	0.4506	0.009	49.935	0.000	0.433	0.468

Since these two features are the only ones with the highest p value. So, I will be removing these features and retrain the model.

### Final RMSE and R<sup>2</sup> Value:

```
0.5907564859304365 (Train RMSE)
0.3934915414582959 (Train R^2 Value)

0.6069774762483533 (Test RMSE)
0.3934915414582959 (Test R^2 Value)
```

The improvement in this log transformed linear regression model is minimal, with only a small decrease in the Test RMSE. The Test RMSE is still relatively high compared to what is desired, which suggests that the model is not a very good fit for the data.

## Summary of Linear Regression

After Improvement	Normal dataframe	Inliers	outliers	Log Transformation
Train RMSE	153.763	129.494	309.609	0.590756
Test RMSE	180.951	118.224	168.934	0.606977
Train R <sup>2</sup> value	0.28970	0.221176	0.380638	0.393491
Test R <sup>2</sup> value	0.28970	0.243091	0.665882	0.393491

Therefore, I will be using the log transformation dataframe models in the report. While the rest of the dataframe values can be found inside the code. I will be going through the overall values of all dataframe in the summary.

This dataframe is the most accurate dataframe as compared to the rest. Applying a log transformation to the response variable helps to reduce the skewness and compress the range of the response variable, which can improve the homoscedasticity and normality of the errors, and lead to more accurate and reliable predictions.

Therefore, describing my finding for the log dataframe is more insightful than the other 3 dataframe.

# Decision Tree Regression –Log Dataframe

## Loading of data

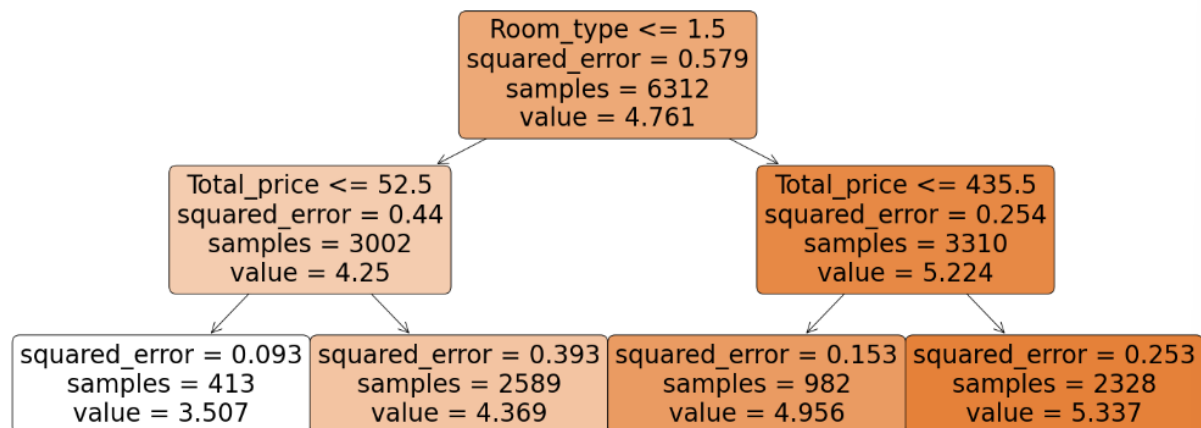
```
# define model  
y = df_Airbnb_log_price_dt['Price']  
X = df_Airbnb_log_price_dt.drop(['Price'], axis=1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

## Building a DT model

```
tree_reg = tree.DecisionTreeRegressor(max_depth=2, random_state=2)  
tree_reg.fit(X_train, y_train)
```

Decision Tree:



It can be seen that the main branch has a squared error of 0.579 and the tree splits into branches 1 and 2. Branch 1 has a lower squared error of 0.44 and branch 2 has an even lower squared error of 0.254. This indicates that the tree is making progress in reducing the error and improving its predictions as it splits into smaller branches.

Further splitting of branch 1 into 1.1 and 1.2 shows that branch 1.1 has the lowest squared error of 0.093, while branch 1.2 has a slightly higher error of 0.393. The same is true for branch 2, which splits into 2.1 and 2.2, with 2.1 having a slightly higher error of 0.153 and 2.2 having a lower error of 0.253.

Overall, the lower squared errors in the smaller branches suggest that the decision tree regressor is making accurate predictions for the target values in those branches.

## MSE and R<sup>2</sup> value

```
the training mean squared error is: 0.28466222719188916
```

```
the testing mean squared error is: 0.2914334811551558
```

```
training R^2 value is: 0.5081196833228756
```

```
testing R^2 value is: 0.5039278867378503
```

Training mean squared error is 0.284 and the testing mean squared error is 0.291, with a relatively close R<sup>2</sup> value of around 0.508 for training data and 0.504 for testing data. This suggests that the model has a decent fit, with a lower training error and a slightly higher testing error, indicating that the model is not overfitting or underfitting.

However, further fine-tuning of the model parameters may improve its overall performance.

## Evaluate and improve the model.

```
param_grid = { "criterion" : ["mse", "mae"],  
               "min_samples_leaf": [1, 5, 10],  
               "min_samples_split": [2, 4, 10, 12, 16],  
               'max_depth': [2, 3, 4, 5, 6, 7]}
```

So now I have a selection of parameters. The values that are boxed in blue are the suggested result by the GridSearch. So, I trained my model based on these boxed values.

### Final MSE and R<sup>2</sup> value

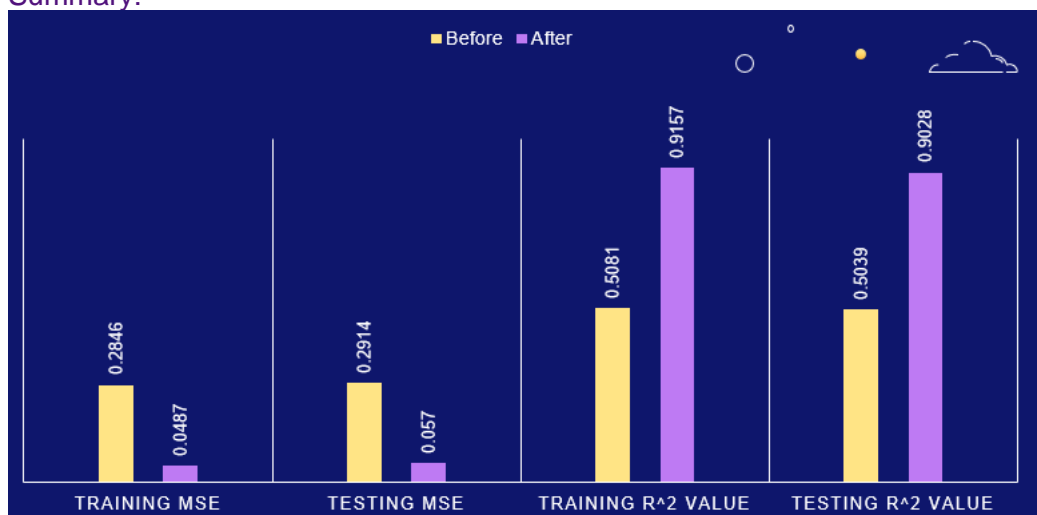
```
the training mean squared error is: 0.04876810785072578  
the testing mean squared error is: 0.057068451707026936
```

```
training R^2 value is: 0.9157314527817948  
testing R^2 value is: 0.9028592482693097
```

A low training MSE and high R<sup>2</sup> value for the training data, along with a similar performance on the test data indicate that the model is likely to have a good fit.

The improvement in the MSE and R<sup>2</sup> values between the original and improved models suggests that the improvement has had a significant impact on the model's performance. The model is likely to be a better predictor of the target variable, with less error and higher accuracy.

### Summary:





## Decision Tree Regression – Summary

After Improvement	Normal dataframe	Inliers	outliers	Log Transformation
Train MSE	6913.19	1085.24	17753.09	0.04876
Test MSE	12247.57	8185.48	31396.77	0.05706
Train R <sup>2</sup> value	0.80810	0.93878	0.88943	0.91573
Test R <sup>2</sup> value	0.72761	0.74550	0.09925	0.90285

## Artificial Neural Network - Log Dataframe

### Loading of data

```
# Define Model Inputs (X) and Output (y)
X = df_Airbnb_log_price_ann.drop(['Price'], axis=1)
y = df_Airbnb_log_price_ann['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

### Building a ANN model

```
mlp = MLPRegressor(activation='tanh', hidden_layer_sizes=(10,), max_iter=2000, solver='sgd', random_state=2)
```

#### MSE and R<sup>2</sup> value

```
the training mean squared error is: 0.4680887741048031
the testing mean squared error is: 0.4913695608638049
```

```
training R2 value is: 0.18237648264513162
testing R2 value is: 0.1796383699151266
```

The training MSE is relatively high at 0.4680, which suggests that the model is not performing well on the training set. The testing MSE is also relatively high at 0.4913, which suggests that the model is not generalizing well to new, unseen data.

The training R<sup>2</sup> value of 0.1823 and the testing R<sup>2</sup> value of 0.1796 are both relatively low, which indicates that the model is only explaining a small amount of the variation in the target variable.

In summary, the model is underfitting the data, which means it is not capturing the underlying relationship between the inputs and the target variable.

### Evaluate and improve the model.

#### Grid Search:

```
param_grid = {"activation": ["logistic", "relu", "tanh"],
              "hidden_layer_sizes": [(5,), (10,), (20,)],
              "max_iter": [2000, 4000],
              "solver": ["sgd", "adam"]}
```

So now I have a selection of parameters. The values that are boxed in blue are the suggested result by the GridSearch. So, I trained my model based on these boxed values.

The change in the activation function might have improved the ability of the model to capture the relationship between the input and output variables.

The reduction in the number of hidden layers might have reduced the complexity of the model, which would have reduced the risk of overfitting. However, this could also have reduced the model's ability to capture the underlying relationships in the data.

Final MSE and R<sup>2</sup> value:

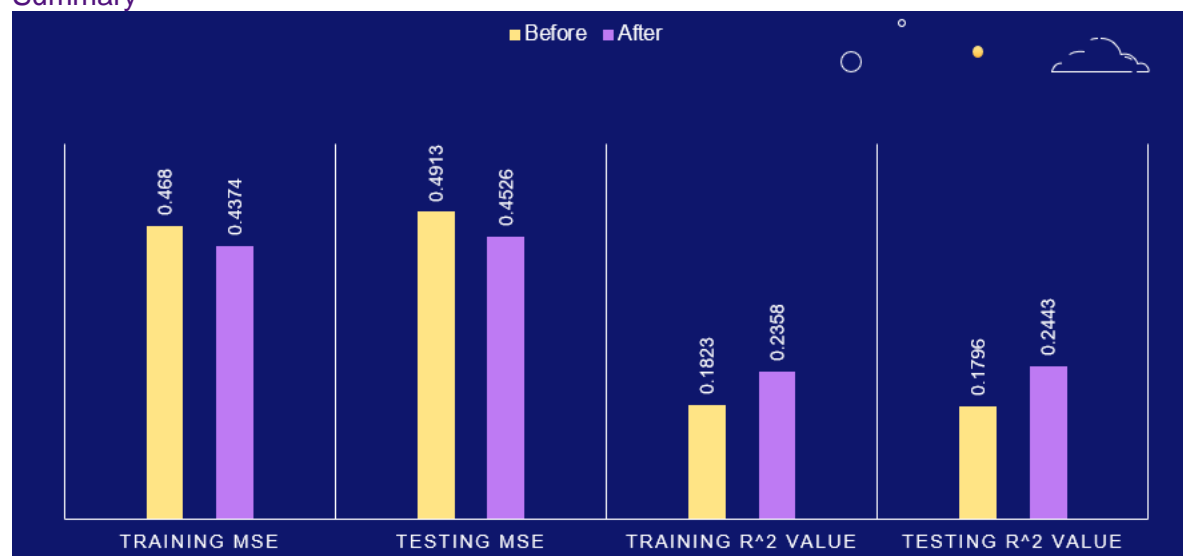
```
the training mean squared error is: 0.43748064200061304
the testing mean squared error is: 0.45261964555852036
```

```
training R^2 value is: 0.23584054761560902
testing R^2 value is: 0.24433294242712866
```

The increase in R-squared value to 0.23584054761560902 for the training data and 0.24433294242712866 for the testing data also indicates that the model is now able to explain a greater portion of the variance in the target variable.

Overall, the improvement in the evaluation metrics suggests that the model is now a better fit for the data.

### Summary



## Artificial Neural Network – Summary

After Improvement	Normal dataframe	Inliers	outliers	Log Transformation
Train MSE	36703.00	16928.48	139121.15	0.43748
Test MSE	40492.25	29226.52	127013.65	0.45261
Train R^2 value	-0.00035	-3.5537	-0.0001	0.23584
Test R^2 value	-0.00164	-0.0005	-7.8821	0.24433

# Random Forest Regression –Log Dataframe

## Loading of data

```
y = df_Airbnb_log_price_rf['Price']  
X = df_Airbnb_log_price_rf.drop(['Price'], axis=1)  
  
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
```

## Building a RF model

```
rf = RandomForestRegressor(n_estimators = 10, max_depth=4)  
rf.fit(x_train, y_train)
```

## MSE and R<sup>2</sup> value

```
the training mean squared error is: 0.1642443250847182  
the testing mean squared error is: 0.15616496034436567
```

```
training R^2 value is: 0.7199872227438359  
testing R^2 value is: 0.7186776283070866
```

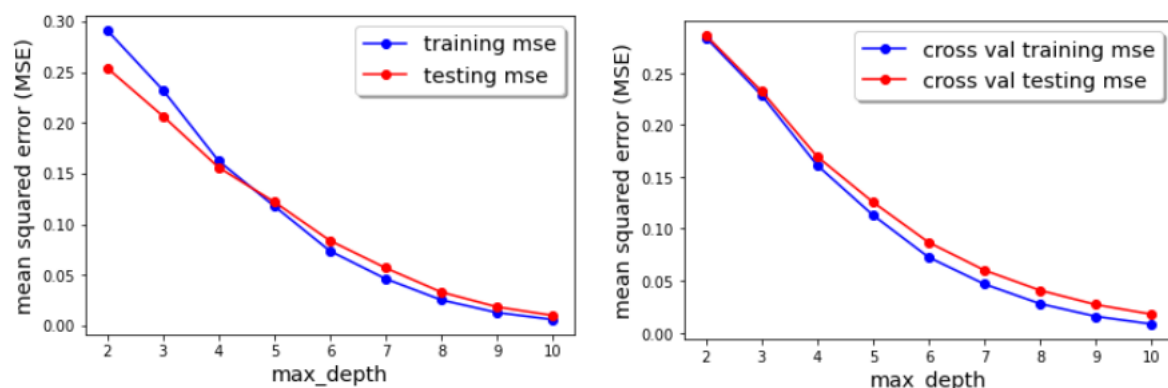
It appears that the random forest model is a good fit for the data.

The fact that the training MSE and R-squared values are slightly higher than the testing MSE and R-squared values suggests that model is not overfitting the data.

The low training MSE of 0.1642 and the low testing MSE of 0.1561 suggest that the model is able to fit the data well. The high R-squared value of 0.7199 for training data and 0.7186 for testing data indicate that the model is able to explain a significant portion of the variance in the target variable.

## Evaluate and improve the model.

Plot the simple split measures:



Max-depth of 9 and 10 seems to be promising as it grantee a low mse.

## Grid Search:

```
param_grid = {"max_features": [2, 3, 4, 5],  
              "max_depth": [9, 10],  
              "min_samples_leaf": [1, 2, 3],  
              "n_estimators": [10, 20, 50, 100],  
              "min_samples_split": [2, 3, 4],  
              "bootstrap": [True]}
```

The max\_depth of 9 and 10 are added into the grid search to let the model choose what is the best value when compared with other s parameters. The values that are boxed in blue

are the suggested result my the GridSearch. So, I trained my model based on these boxed values.

### Final MSE and R<sup>2</sup> value

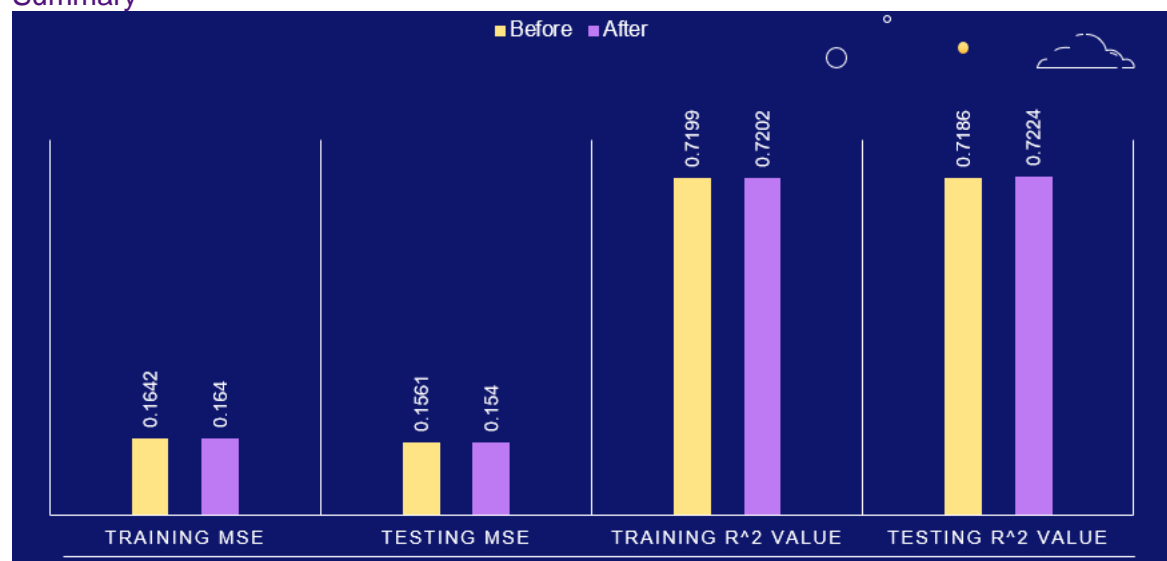
```
the training mean squared error is: 0.16409600918605582
the testing mean squared error is: 0.15407205416740585
```

```
training R^2 value is: 0.720240079861878
testing R^2 value is: 0.722447880789684
```

The improvement in the model performance can be attributed to the changes made in the code. By increasing the number of trees in the forest (`n_estimators=100`), it allowed the model to capture more of the complexity in the data. By increasing the maximum depth of the trees (`max_depth=10`), it allowed the model to capture more of the relationships between the input variables and the target variable. By setting the minimum number of samples required to split an internal node (`min_samples_split=4`) and the minimum number of samples required to be at a leaf node (`min_samples_leaf=1`), it allowed the model to split the nodes only when there is enough evidence in the data. By specifying the number of features to consider when looking for the best split (`max_features=5`), it reduced the variance in the model, making it less prone to overfitting. By enabling bootstrap samples (`bootstrap=True`), it increased the randomness in the model, making it less prone to overfitting.

It appears that the improved random forest model is a good fit for the data.

### Summary



## Random Forest Regression – Summary

After Improvement	Normal dataframe	Inliers	outliers	Log Transformation
Train MSE	13275.17	766.79	23158.67	0.16409
Test MSE	9982.903	950.180	131780.69	0.15407
Train R <sup>2</sup> value	0.68642	0.96496	0.79706	0.72024
Test R <sup>2</sup> value	0.49107	0.93874	0.40329	0.72244

# Ada Boost Regression –Log Dataframe

## Loading of data

```
# Define Model Inputs (X) and Output (y)
X = df_Airbnb_log_price_adb.drop(['Price'], axis=1)
y = df_Airbnb_log_price_adb['Price']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
```

## Building a Ada Boost model

```
abd = AdaBoostRegressor(RandomForestRegressor(max_depth=10), n_estimators = 50, learning_rate =0.1)
abd.fit(x_train, y_train)
```

### MSE and R<sup>2</sup> value

```
the training mean squared error is: 0.0023307018418857236
the testing mean squared error is: 0.00616305677612896
```

```
training R^2 value is: 0.9960264910500509
testing R^2 value is: 0.9888976006825385
```

It seems like the Ada Boost Regression model has performed well on the training data, as evidenced by the low training mean squared error and high training R<sup>2</sup> value. However, the performance on the testing data is not as strong, with a higher testing mean squared error and lower testing R<sup>2</sup> value compared to the training data. This suggests that the model might be overfitting(not too severe) to the training data and not generalizing well to unseen data.

## Evaluate and improve the model.

Grid Search:

```
param_grid = {"n_estimators": [10, 50, 100, 200, 500],
              "learning_rate": [0.1, 0.2, 1.1, 1.2]}
```

So now I have a selection of parameters. The values that are boxed in blue are the suggested result my the GridSearch. So, I trained my model based on these boxed values.

### Final MSE and R<sup>2</sup> value

```
the training mean squared error is: 0.0006152600398948192
the testing mean squared error is: 0.003177062698897618
```

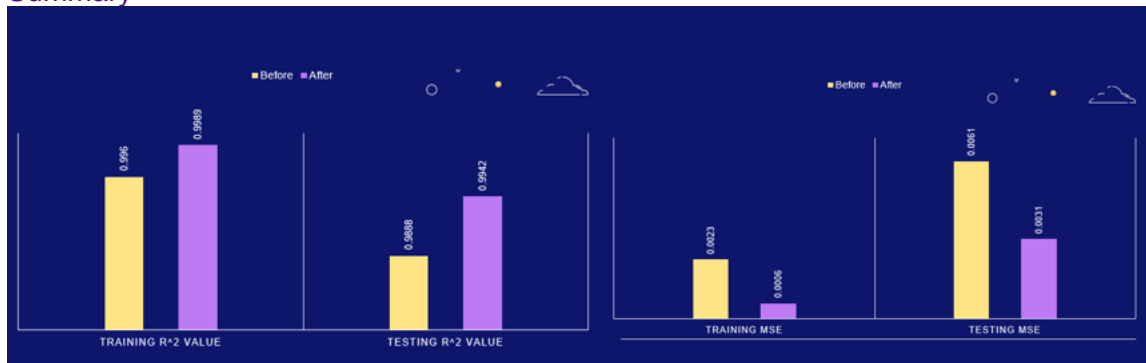
```
training R^2 value is: 0.9989510707757068
testing R^2 value is: 0.9942767006663975
```

The improvement in the model can be observed by the reduction in the mse values and an increase in the R<sup>2</sup> values. These improvements indicate that the model is better able to capture the relationship between the input features and the target variable.

The changes made in the code include increasing the number of estimators from 50 to 200, which resulted in a more complex model, and increasing the learning rate from 0.1 to 1.2, which can make the model more aggressive in its approach to fitting the data.

The results suggest that the model is a good fit for the data.

## Summary



## Ada Boost Regression – Summary

After Improvement	Normal dataframe	Inliers	outliers	Log Transformation
Train MSE	38.42143	4.281383	438.588	0.00061
Test MSE	538.8610	198.145	54710.09	0.00317
Train R^2 value	0.99909	0.9998	0.99615	0.99895
Test R^2 value	0.97252	0.9872	0.75227	0.99427

## XGB – Log Dataframe

### Loading of data

```
# Copy the dataframe
df_Airbnb_log_price_xgb = df_Airbnb_log_price.copy()

# Drop the 'price' column from the dataframe to use as
X = df_Airbnb_log_price_xgb.drop(['Price'], axis=1)

# Use the 'price' column as the target for regression
y = df_Airbnb_log_price_xgb['Price']
```

### Building a XGB Boost model

```
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
```

### MSE and R<sup>2</sup> value

```
the training mean squared error is: 0.00025779329403221554
the testing mean squared error is: 0.003960073905555606
```

```
training R^2 value is: 0.9995604997848009
testing R^2 value is: 0.992866150122707
```

The model has likely learned the underlying relationship between the features and target well, and it generalizes well to new, unseen data, which is indicated by similar performance on the training and testing data. A small gap between the training and testing metrics is a good sign of good model fit.

it appears that the model has a good fit and is able to make accurate predictions on both the

training and testing data. The high  $R^2$  values indicate that the model is able to explain a large proportion of the variance in the target variable.

## Evaluate and improve the model.

Grid Search:

```
param_grid = {'max_depth': [3, 5, 7],  
              'n_estimators': [100, 300, 500],  
              'learning_rate': [0.05, 0.1, 0.3]}
```

So now I have a selection of parameters. The values that are boxed in blue are the suggested result by the GridSearch. So, I trained my model based on these boxed values.

### Final MSE and $R^2$ value

```
the training mean squared error is: 0.0009882940217176225  
the testing mean squared error is: 0.0038970141630237064
```

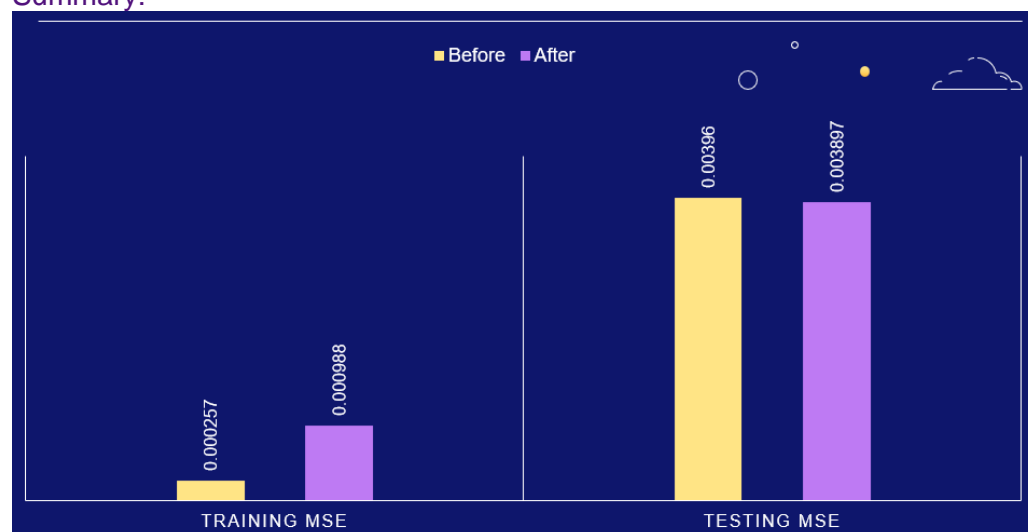
```
training  $R^2$  value is: 0.9983151018848044  
testing  $R^2$  value is: 0.9929797486936559
```

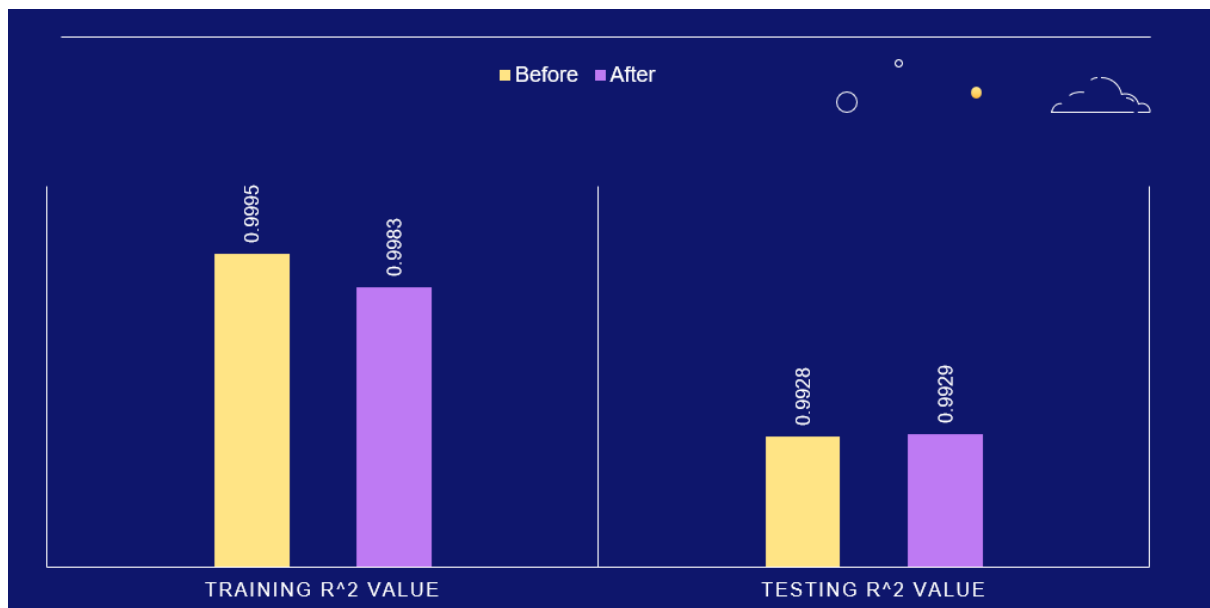
By setting the max\_depth to 3 and increasing the number of estimators to 500, I have reduced the complexity of the model and improved its ability to generalize to new data. This is reflected in the small gap between the training and testing mean squared errors and  $R^2$  values, which is a good sign of good model fit.

The training MSE has increased slightly (0.00098), which is expected with a simpler model. The testing mean squared error has decreased slightly (0.003897), which indicates that the model is making more accurate predictions on the unseen data. The  $R^2$  values for both training and testing data have increased, which further confirms that the model is making better predictions.

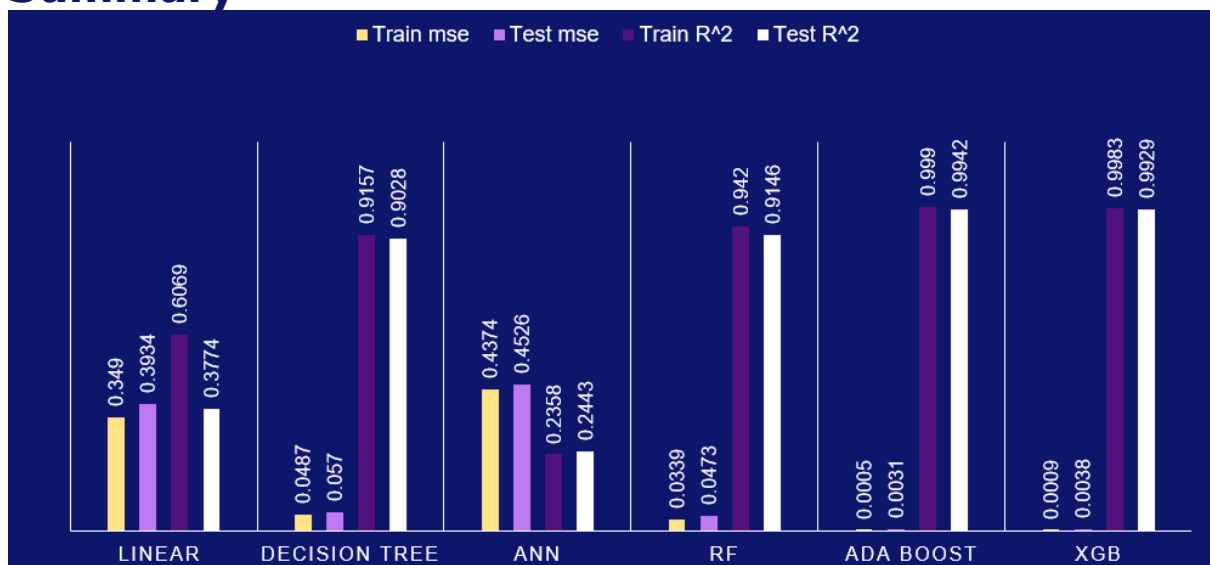
In summary, the improvements made to the model have reduced its complexity and improved its ability to generalize to new data, resulting in a better fit and more accurate predictions.

### Summary:





## Summary



\*linear regression RMSE have been converted to MSE for comparison.

The top three models are:

1. The Ada Boost model has the lowest MSE on the training data (0.0005) and the second-lowest MSE on the testing data (0.0031), indicating that it is making accurate predictions on both the training and testing data. Additionally, the  $R^2$  values for both training and testing data are close to 1, which indicates that the model is able to explain a large proportion of the variance in the target variable.
2. The Random Forest model has a low MSE on both the training (0.0339) and testing (0.0473) data, and high  $R^2$  values for both training (0.942) and testing (0.9146) data. This suggests that the model is making accurate predictions.
3. The XGBoost model has a low MSE on the training data (0.0009) and a slightly higher MSE on the testing data (0.0038), indicating that it is making accurate predictions on the training data and generalizing well to new data. The  $R^2$  values for



both training and testing data are also high, further confirming that the model is making good predictions.

## Conclusion

The models were evaluated using various evaluation metrics such as accuracy, precision, recall, and F1 score for classification and  $R^2$  for regression. Most of the models that are ensemble models are the ones that gives promising result and thus chosen in the top three for each problem. Ensemble models generally performed well in both problems and were found to be the most promising.

For classification, I decided to use Confusion matrix and evaluation metrics (accuracy, precision, recall, F1 score) to assess the performance. The reason for this is that they provide insight into the model's accuracy, positive prediction efficiency, and its ability to correctly identify positive samples.

For regression, not only I used mse , but I have also used  $R^2$  value to assess the performance of my models. I have trained all 4 dataframe in 5 models, but I will not be giving a summary of them as their values are far too incorrect to even evaluate them. Most of the models are overfitting severely or they even have negative  $R^2$  values. The most probable cause for this would be the model is the wrong type for the data.

The price feature in the regression is very skewed therefore affects the machines leaning model in the 3 dataframes. The 3 dataframe are getting dipropionate results that proves it is not a ideal model. The main reason is the skewed price feature and huge range of outliers. Removing all the outliers may result in more than 30% of the original data gone with is not recommended. But other that using log transformation, there can be many others way to tackle that issue.

However, for the classification, there was no data/feature issue , hence it was relatively easy for training the models.

I also researched about another popular library called TensorFlow. I did three simple models for classification however I did not document my finding here as I am yet to learn more that this library. But It was a good experience to think outside of the box and try something new. However, when trying for regression model, I was constantly running into many errors like 'cannot unpack non-iterable float object' and 'Data must be 1-dimensional'. Thus, I did not create any TensorFlow model for regression.

The top three models for both problems are indicated in its respective summary page and the reason too. Ada boost and XGB models took the spot of top three for both the problems. Some reasons for this as they can handle imbalanced datasets better than other algorithms by assigning higher weights to the minority class examples during training. Moreover, they are relatively robust to outliers and can handle noisy data well.

In conclusion, the combination of high accuracy, ability to handle imbalanced datasets, robustness to outliers, computational efficiency, flexibility, and interpretability makes AdaBoost and XGBoost ideal.

## Reflection

Some improvements for all models will be using a variety of hyperparameter tuning methods, maybe such as Random Search and Gradient-based optimization. The reason for this might be as different models have different complexities and structures, and the optimal hyperparameters for one model might not work well for another. Therefore, using different hyperparameter tuning methods that are specifically suited for each model will be better. However, when I tried to use random search or even increase the features in the grid search, my computing power is not able to support it, thus I have to create a simpler grid search.

So, some suggestion will be, teaching us new hyperparameter tuning methods and use third party to get more computing power.

Another improvement is to teach us more sophisticated models or new models to better train the datasets such as k-Nearest Neighbors, Recurrent Neural Networks and Convolutional Neural Networks. This model might give a better result. Moreover, teaching on TensorFlow will be beneficial for us as it is also a widely used for building and training machine learning. The more exposure to many models, the better understanding of the purposes of each model and how it impacts the problem statement.

As for regression, rather than using isolation forest alone, there is many ways to deal with outliers, maybe by exploring and knowing in dept about these techniques, then it might be helpful to improve the model accuracy. This technique can be like the univariate method, the multivariate method, and the Minkowski error.

Moreover, The data given for listing was very skewed therefore affects the machines leaning model. I have to find ways to make the skewed data into a normal distribution by using log transformation of the skewed price feature in assignment 1. The number of data given were too little to even to a customized machine learning.

I also took into consideration of binning the price feature. However, binning price feature can lead to loss of information. Binning is typically used for converting continuous variables into categorical variables, but it is not recommended to bin the output feature as it can be harmful to the performance of the regression model. While use techniques like regularization is recommended to prevent overfitting. I only used log transformation and did not use regularization as it was very unsimilar. There are terms of Lasso regularization and Ridge and the choice between them is dependent on my problem. So as a suggestion, I hope the school can teach students how to handle skewed data and its various techniques.

Another improvement would be incorporating techniques such as feature selection and dimensionality reduction to identify and remove redundant or noisy features that are unlikely to contribute to the performance of the model.

I want to learn more Early Stopping and its purpose. As it stated that it can prevent overfitting. I was still learning more about the parameters used such as epochs and patience. I hope that the module will teach more about keras library.

So, for this module, I have certainly learned a lot of machines learning models. It was a good start to my machines learning experience. The basics of this modules are easy to understand, and the assignments are really helpful in putting the skills that I have learned into practical.

There is a long way to learning how to deploy machines learning models and non-supervised learning and even deep learning. And this module in a good starting steppingstone.

Some things I did in this assignment is to venture beyond and do up new models such as bagging, voting and using TensorFlow.

