**Results and Discussions:**

**Performance metrics:**

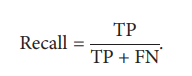
We utilized a variety of metrics to assess algorithm performance. The confusion matrix is used by the majority. The confusion matrix is a tabular representation of the performance of a classification model on the test set, with four parameters: true positive, false positive, true negative, and false negative.

**Accuracy**: Accuracy is a commonly used measure that represents the percentage of properly predicted true or erroneous observations. The following equation can be used to calculate the accuracy of a model's performance:



In most cases, a high accuracy value indicates a good model; however, because we are training a classification model in this case, an article that was predicted as true but was actually false (false positive) can have adverse implications; similarly, if an article was predicted as false but contained verifiable facts, then this can lead to trust issues. As a result, we have utilized three other metrics: precision, recall, and F1-score, to account for the erroneously categorized observation.

**Recall:** The total number of positive classifications out of true class is referred to as recall. It indicates the proportion of articles predicted to be true out of the total number of true articles in our scenario.



**Precision Score:** Precision score, on the other hand, is the ratio of true positives to all true occurrences predicted. Precision in this context refers to the number of articles marked as true out of all the positively predicted (true) articles.



**F1 – Score**: The trade-off between precision and recall is represented by the F1-score. It calculates the harmonic mean of each of the two. As a result, it considers both false positive and false negative observations. The following formula may be used to compute the F1-score.



**Hyperparameter Tuning:**

The practice of tuning the parameters given as tuples when building machine learning models is known as hyperparameter tuning. These parameters are defined by the programmer and can be changed if desired. These parameters are never learned by machine learning algorithms. These have been fine-tuned to ensure that the model performs well. The goal of hyperparameter tuning is to identify the parameters that provide the model the best performance or the best performance with the lowest error rate.

**RandomizedSearchCV:**

RandomizedSearchCV samples a predetermined number of hyperparameter settings from specified probability distributions. To decrease the amount of iterations and to utilize a random mix of parameters, RandomizedSearchCV is used. When you have a lot of parameters to try and your training time is longer, this is a good option. It aids in the reduction of computation costs.

**GridSearchCV:**

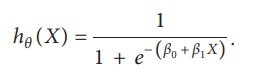
Grid search is the process of fine-tuning hyperparameters to find the best values for a particular model. The value of hyperparameters has a major impact on a model's performance. There is no way to know ahead of time what the optimum values for hyperparameters are, therefore we must attempt all conceivable values to find the best ones. Because manually tweaking hyperparameters would take a significant amount of time and resources, we utilize GridSearchCV to automate the process.

Few machine learning and deep learning models have been selected to test the datasets. This will determine which model will work the best for which dataset. After shortlisting the models further based on accuracies, 4 models are selected and further tuned to give even better results.

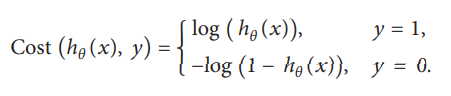
A random seed with value 12 is used throughout the model development and evaluation to reproduce the same results.

**Model Implementation Using Logistic Regression:**

A logistic regression (LR) model is usually used to classify text based on a number of features with a binary output (true/false or true article/fake article), since it gives a straightforward equation to classify problems into binary or multiple classes. The logistic regression hypothesis function can be described mathematically as follows[18]:



The output of logistic regression is transformed into a probability value using a sigmoid function; the goal is to minimize the cost function to get the optimal probability. The cost function is computed as follows:



The results observed were as follows:

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

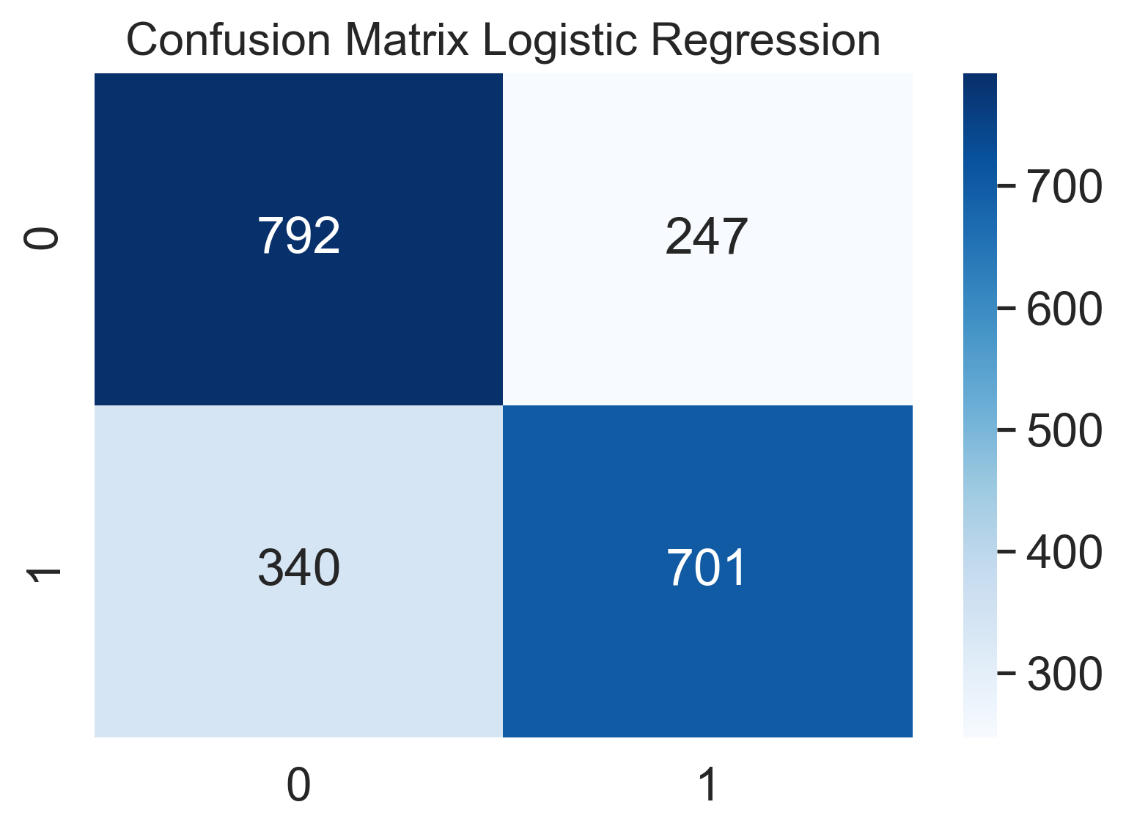
Dataset 1:

0.7177884615384615

0.739451476793249

0.6733909702209414

0.7048768225238813



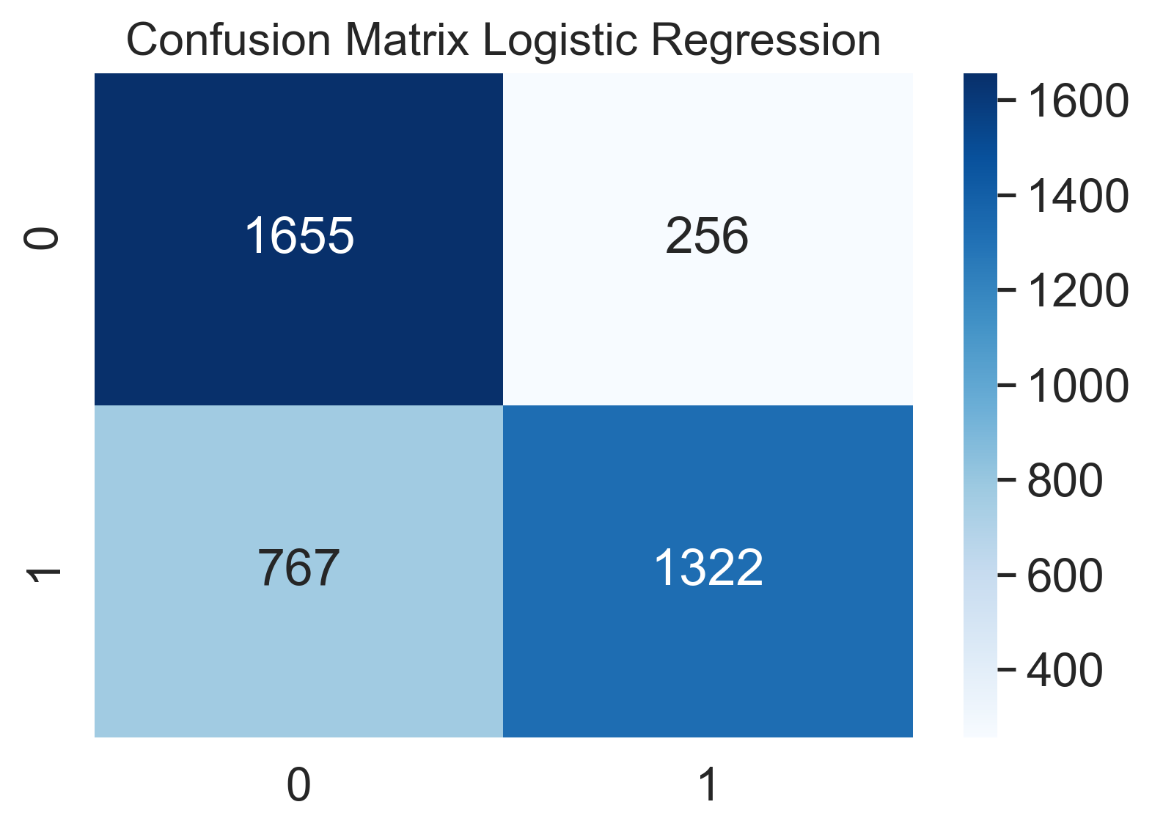
Dataset 2:

0.74425

0.8377693282636248

0.6328386787936812

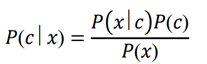
0.7210253613307881

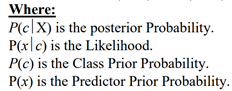


Since, the model does not give satisfying results, it is not further hyperparameter tuned.

**Model Implementation Using Naïve Bayes:**

The Naive Bayes model is a conditional probability model for labeling. The aim is to find a function f: A—>B that can predict the class variable (B) using a vector of independent variables (A). The aim is to find P(B|A), or the probability of B belonging to a given class A. B is a categorical variable with two or more discrete values, according to most definitions. It's mathematically simple to account for a variety of influences when predicting the class of the next data instance in the testing set. The drawback of Naive Bayes is that it assumes that all features are independent of one another. The Naive Bayes rule is based on Bayes' theorem, which is as follows:





The results observed were as follows:

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

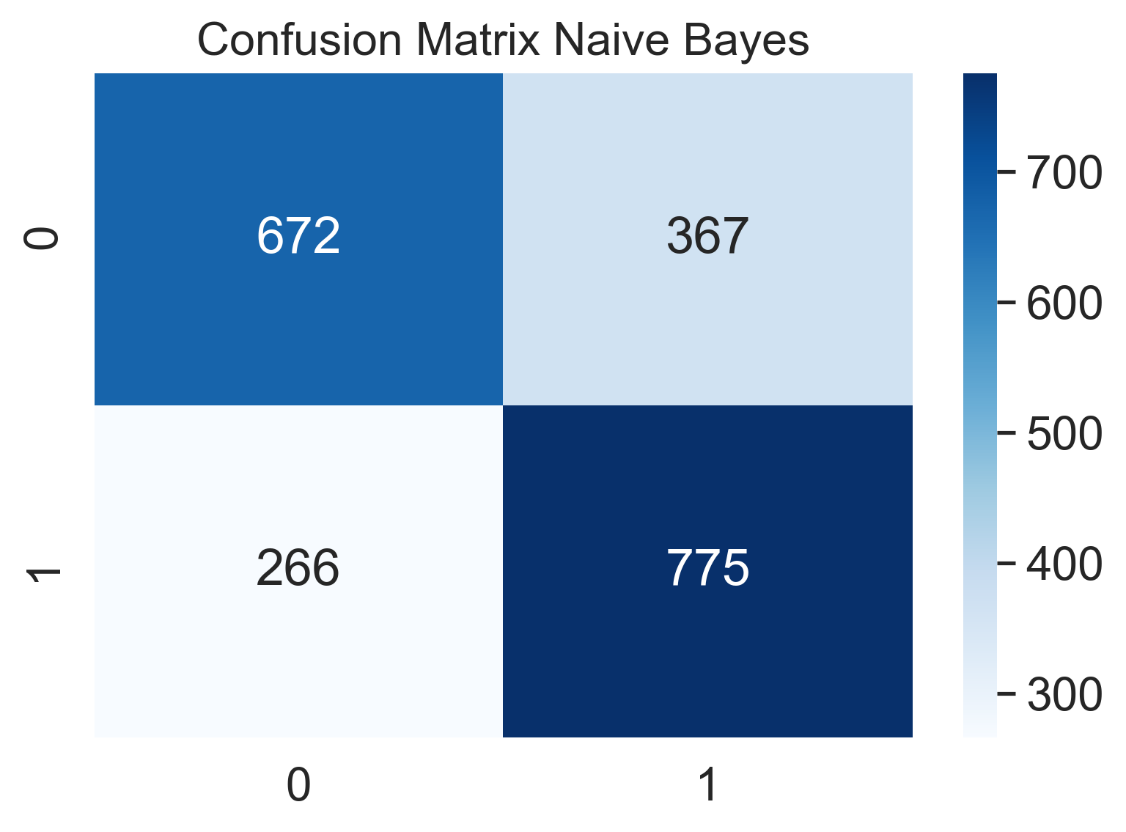
Dataset 1:

0.6956730769230769

0.6786339754816112

0.74447646493756

0.7100320659642693



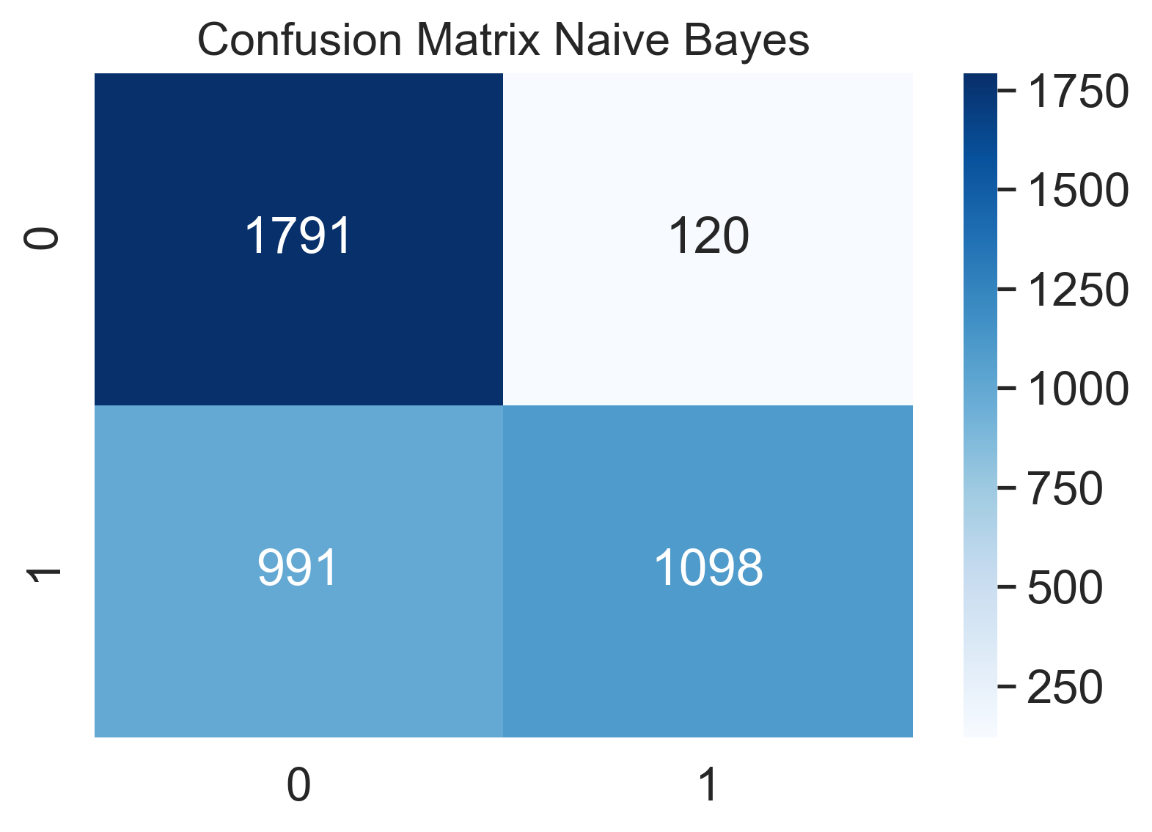
Dataset 2:

0.72225

0.9014778325123153

0.5256103398755385

0.6640459631085577



Since, the model does not give satisfying results, it is not further hyperparameter tuned.

**Model Implementation Using KNN:**

The results observed were as follows:

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

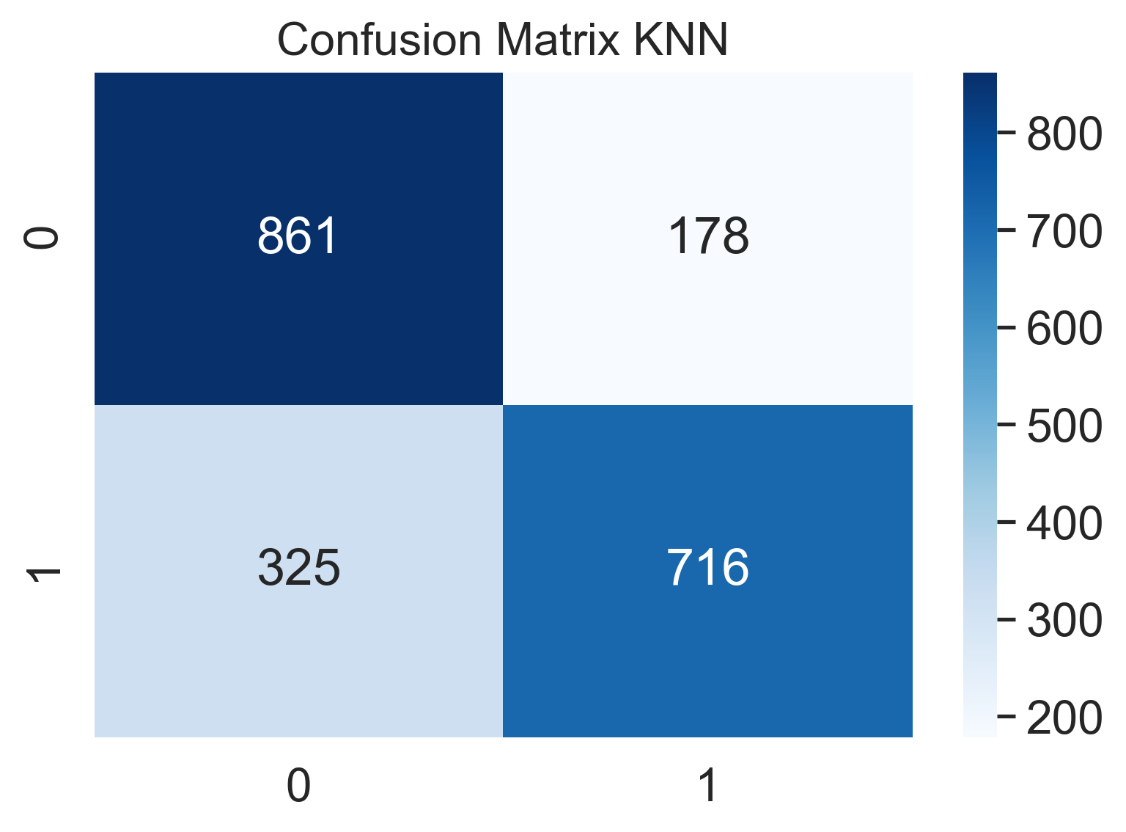
Dataset 1:

0.7581730769230769

0.8008948545861297

0.6878001921229587

0.7400516795865634



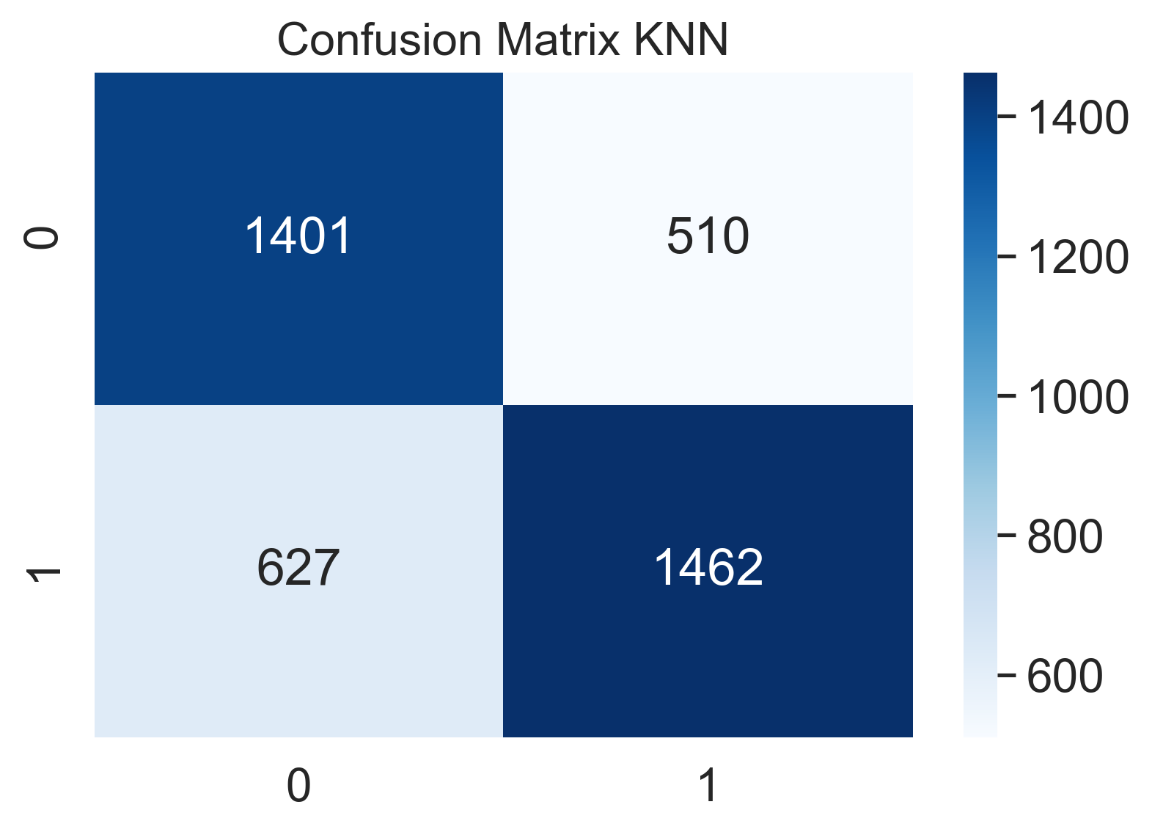
Dataset 2:

0.71575

0.7413793103448276

0.6998563906175204

0.720019699581384



Since, the model does not give satisfying results, it is not further hyperparameter tuned.

**Model Implementation Using Decision Trees:**

A decision tree is a useful tool that is built on a flowchart-like structure and is mostly used to solve classification problems. Each decision tree internal node defines a condition or "test" on an attribute, and branching is based on the test conditions and results. Finally, the leaf node has a class label that is determined once all attributes have been computed. The classification rule is represented by the distance between the root and the leaf. The fact that it can operate with both a category and a dependent variable is incredible. They are effective in identifying the most significant features and depicting the relationships between them. They are important in the creation of new variables and features that are beneficial for data exploration and accurately forecast the target variable.

Predictive models that use supervised learning methods frequently use tree-based learning algorithms to achieve high accuracy. They excel at mapping non-linear relationships. They are also known as CART and are very good at solving classification or regression problems.

The results observed are as follows:

**Baseline Accuracy Dataset 1:** 0.894231

**Baseline Accuracy Dataset 2:** 0.7915

Since, the model does give satisfying results, it is further hyperparameter tuned.

The best hyperparameter results for dataset 1 are:

'criterion': 'entropy',

'max\_depth': 80,

'max\_leaf\_nodes': 95,

'min\_samples\_leaf': 3,

'min\_samples\_split': 25

The best hyperparameter results for dataset 2 are:

'criterion': 'gini',

'max\_depth': 78,

'max\_leaf\_nodes': 85,

'min\_samples\_leaf': 25,

'min\_samples\_split': 14

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

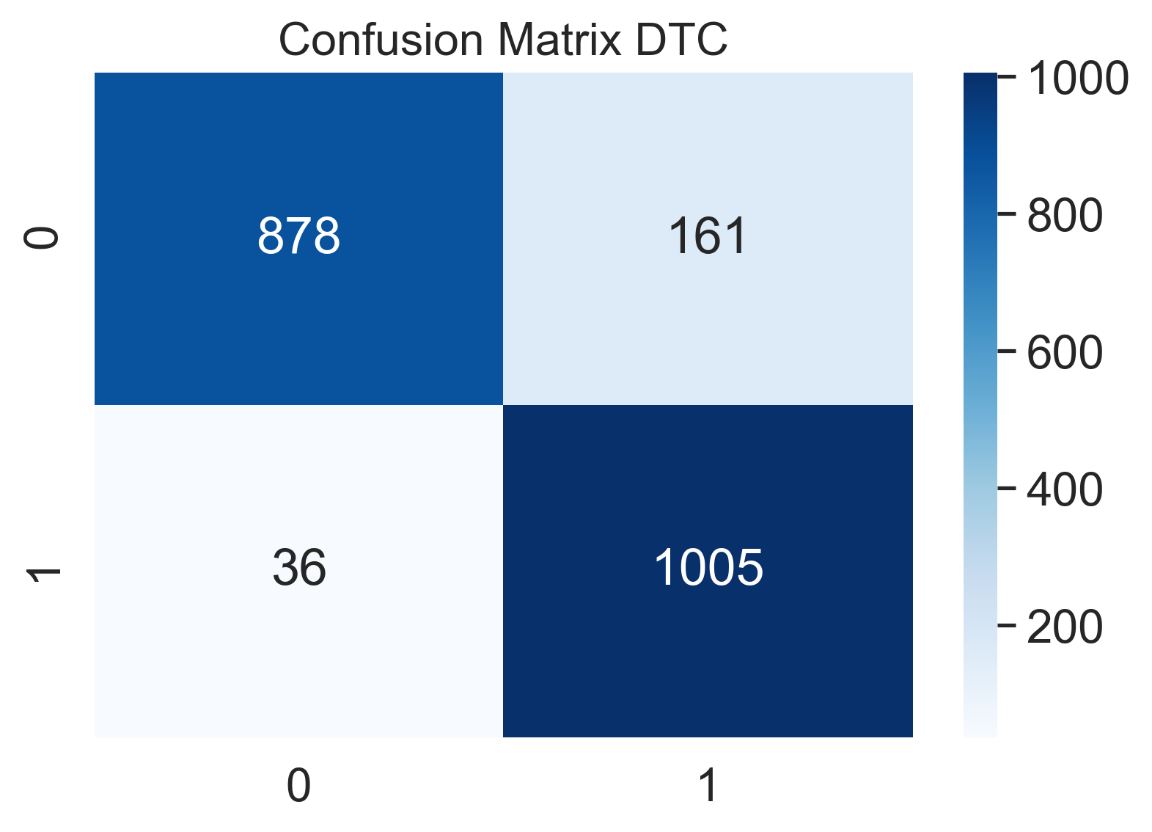
Dataset 1:

0.9052884615384615

0.8619210977701544

0.9654178674351584

0.9107385591300409



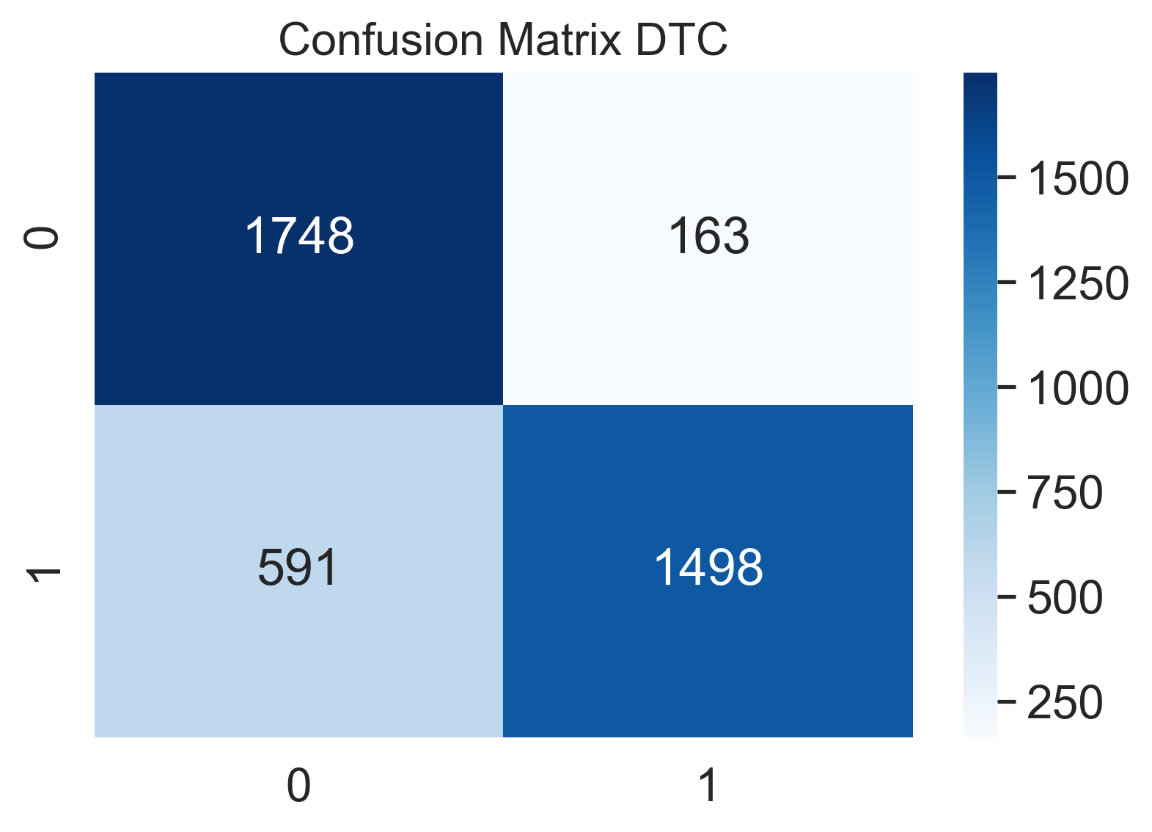
Dataset 2:

0.8115

0.9018663455749548

0.717089516515079

0.7989333333333334



**Model Implementation Using Random Forest:**

Random forest is a supervised learning model that is an enhanced variant of decision trees. RF is made up of a huge number of decision trees that work together to predict the outcome of a class, with the final prediction based on the class that obtained the most votes. Due to minimal correlation across trees, the error rate in random forest is low when compared to other models[19]. Random forest is most useful when applied to uncorrelated decision trees. The final outcome will be more or less comparable to a single decision tree if applied to similar trees. Bootstrapping and feature randomness can be used to create uncorrelated decision trees.

The primary distinction between Random Forest and Naive Bayes is their model size. Because Naive Bayes models are not effective at capturing complicated behavior, they have a small model size and are only useful for a specific constant type of data. The model size for the Random Forest model, on the other hand, is quite big, which may lead to overfitting. When new data is entered, Naive Bayes is ideal for it since it can quickly be molded, but Random Forest may need a forest rebuild every time a modification is made.

The results observed are as follows:

**Baseline Accuracy Dataset 1:** 0.909615

**Baseline Accuracy Dataset 2:** 0.85625

Since, the model does give satisfying results, it is further hyperparameter tuned.

The best hyperparameter results for dataset 1 are:

'bootstrap': False,

'max\_depth': None,

'min\_samples\_leaf': 1,

'min\_samples\_split': 6,

'n\_estimators': 100

The best hyperparameter results for dataset 2 are:

'bootstrap': True,

'max\_depth': None,

'min\_samples\_leaf': 1,

'min\_samples\_split': 5,

'n\_estimators': 900

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

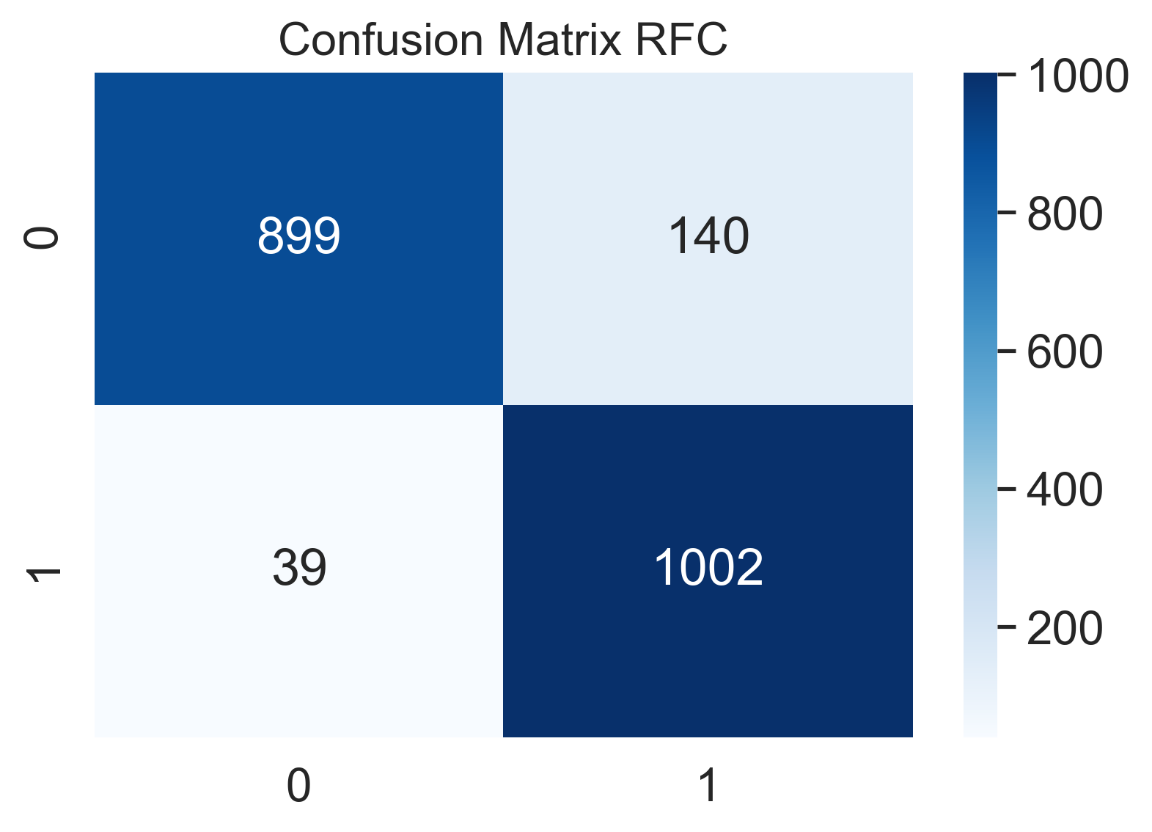
Dataset 1:

0.9139423076923077

0.8774080560420315

0.962536023054755

0.9180027485112231



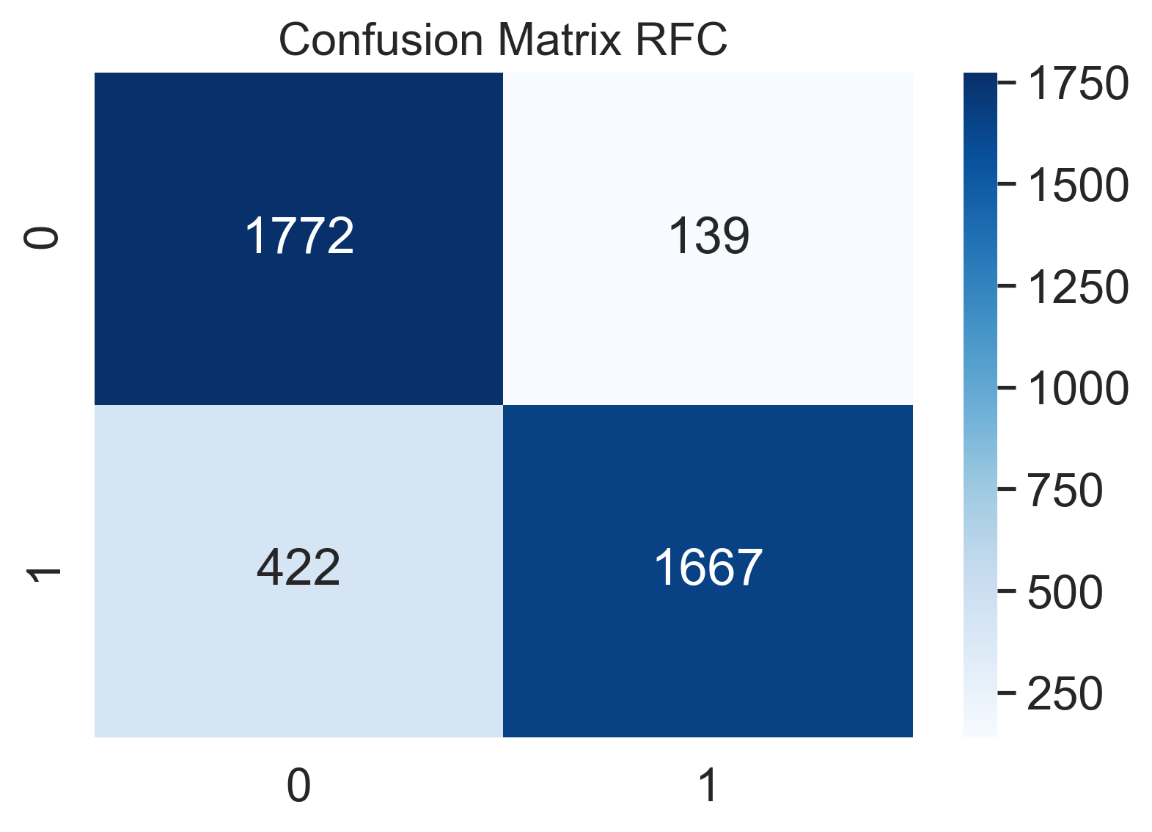
Dataset 2:

0.85975

0.9230343300110742

0.7979894686452849

0.85596919127086



**Model Implementation Using Boosting Ensemble Classifiers:**

A common ensemble approach for training weak models to become strong learners is boosting. A forest of randomized trees is trained for this purpose, and the final prediction is based on each tree's majority vote outcome. This method uses an incremental approach to assist weak learners to correctly classify data points that are frequently misclassified. To classify an issue, equal weighted coefficients are utilized at first for all data points. In subsequent rounds, the weighted coefficients are reduced for properly classified data points and increased for incorrectly classified data points[20]. Each round's subsequent tree learns to minimize the previous round's errors and improve overall accuracy by properly classifying data points that were misclassified in prior rounds. One of the primary drawbacks of boosting ensemble is that it may overfit to the training data, resulting in inaccurate predictions for unknown cases[21]. There are a variety of boosting techniques available that may be utilized for classification and regression. In our experiment, we have employed XGBoost[22] and CatBoost[23] algorithms for classification purposes.

**XGBOOST:**

The results observed are as follows:

**Baseline Accuracy Dataset 1:** 0.910577

**Baseline Accuracy Dataset 2:** 0.8115

Since, the model does give satisfying results, it is further hyperparameter tuned.

The best hyperparameter results for dataset 1 are:

'colsample\_bytree': 0.5,

'learning\_rate': 0.12,

'max\_depth': 74,

'min\_child\_weight': 1,

'n\_estimators': 850,

'subsample': 0.5,

'verbosity': 1

The best hyperparameter results for dataset 2 are:

'colsample\_bytree': 0.5,

'learning\_rate': 0.1,

'max\_depth': 74,

'min\_child\_weight': 1,

'n\_estimators': 850,

'subsample': 0.5,

'verbosity': 1

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

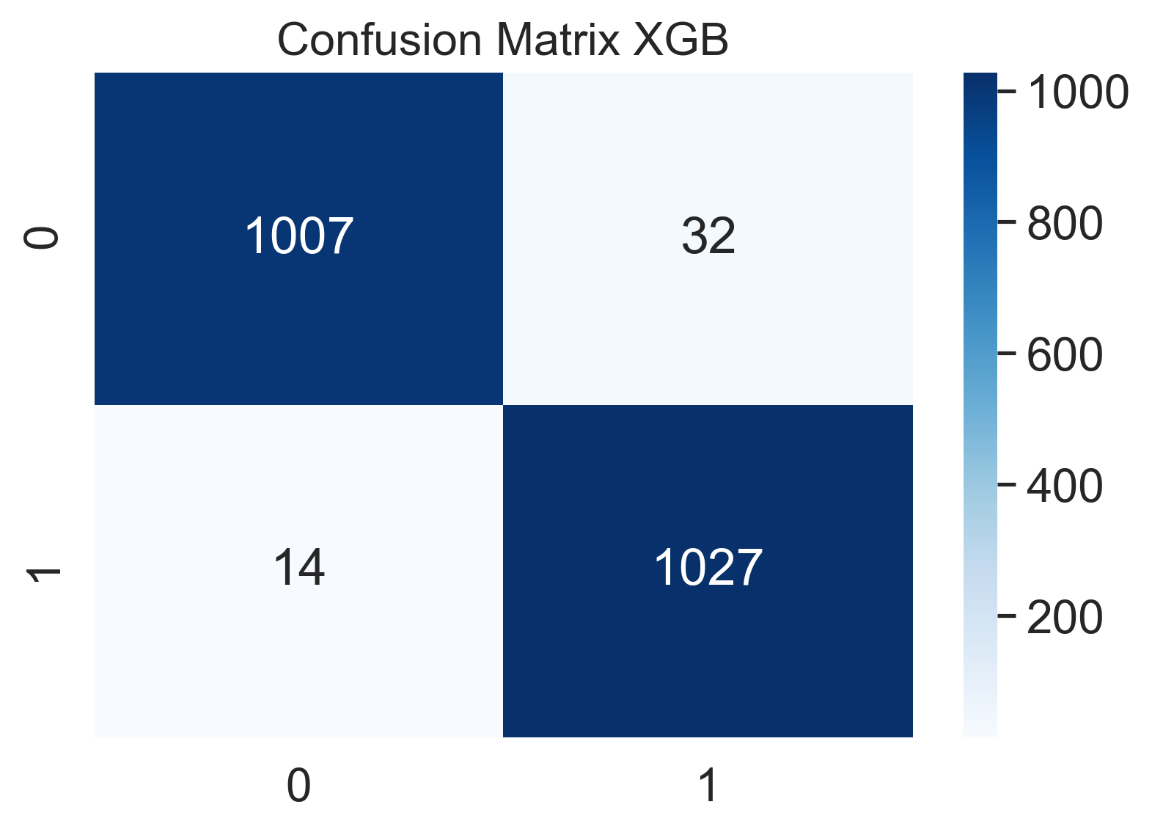
Dataset 1:

0.9778846153846154

0.9697828139754485

0.9865513928914506

0.9780952380952381



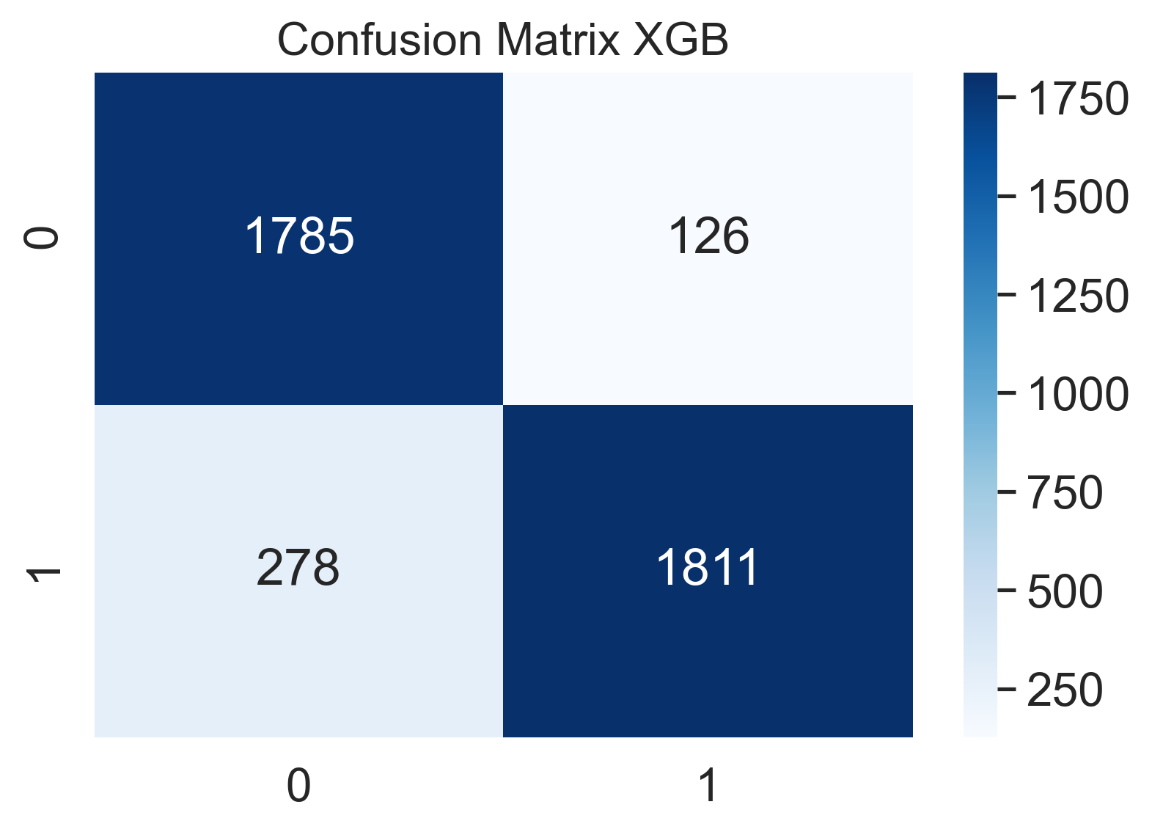
Dataset 2:

0.899

0.9349509550851832

0.8669219722355194

0.8996522603079979



**CATBOOST:**

The results observed are as follows:

**Baseline Accuracy Dataset 1:** 0.975481

**Baseline Accuracy Dataset 2:** 0.861

Since, the model does give satisfying results, it is further hyperparameter tuned.

The best hyperparameter results for dataset 1 are:

'border\_count': 200,

'depth': 8,

'iterations': 500,

'l2\_leaf\_reg': 10,

'learning\_rate': 0.2,

'thread\_count': 4

The best hyperparameter results for dataset 2 are:

'border\_count': 110,

'depth': 8,

'iterations': 500,

'l2\_leaf\_reg': 2,

'learning\_rate': 0.3,

'thread\_count': 4

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

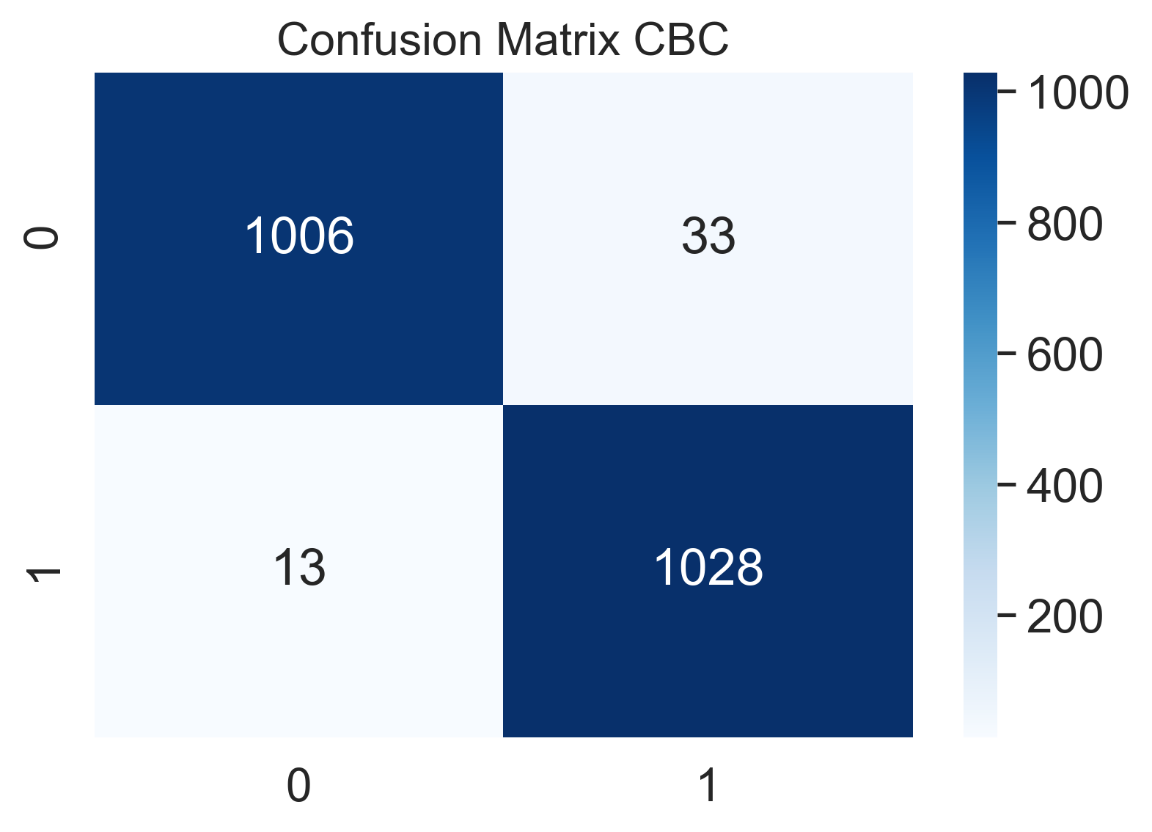
Dataset 1:

0.9778846153846154

0.9688972667295005

0.9875120076849183

0.978116079923882



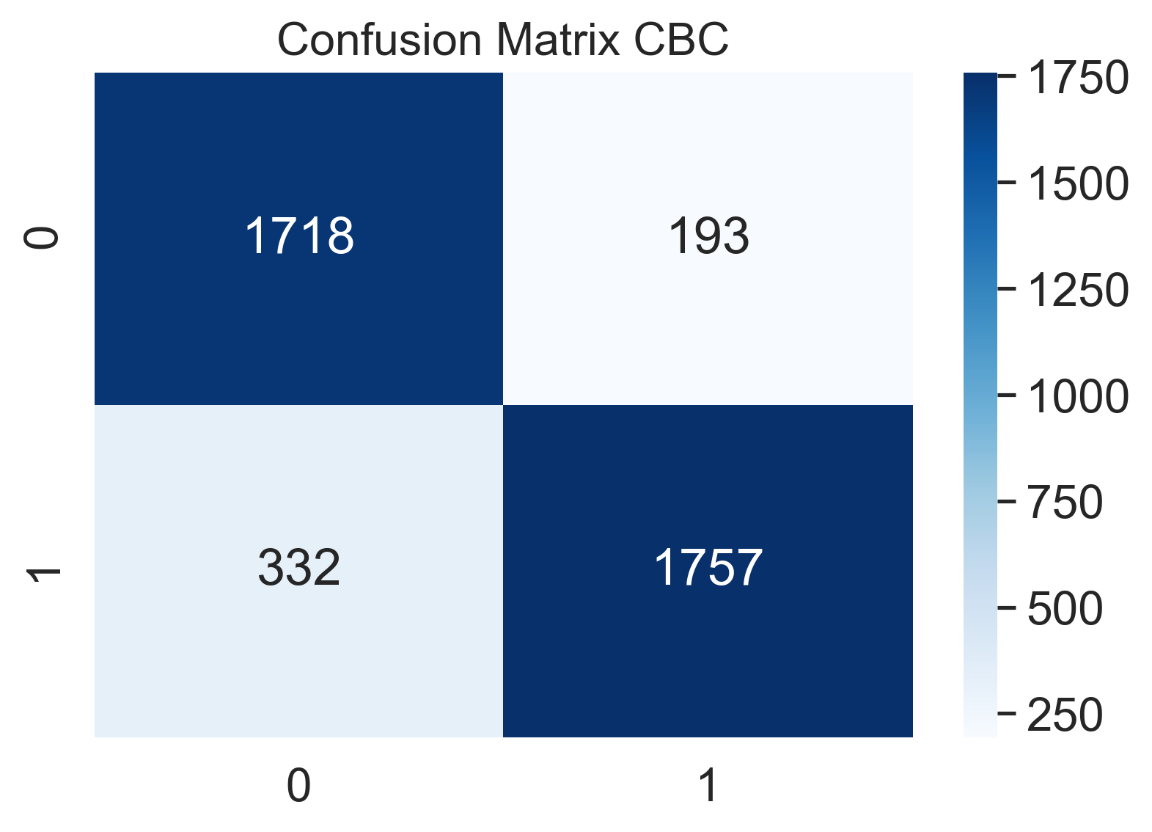
Dataset 2:

0.86875

0.901025641025641

0.8410722833891814

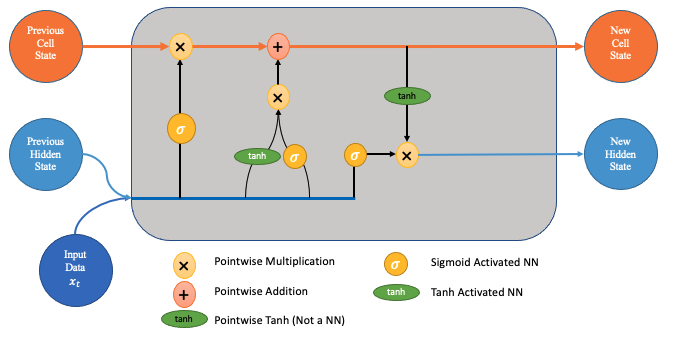
0.8700173310225303



**Model Implementation Using LSTM**

RNN (Recurrent Neural Networks) have a problem: vanishing gradient problem. This problem is basically when the gradient of the loss function tries to reach an optimum value which makes the neural network harder to train. This particularly happens due to change in the value of the sigmoid activation function which makes the derivative smaller. A solution to this could be using simply different activation function like ReLU because it does not cause a smaller derivative. Also, RNN suffers from memory loss.

LSTM (Long Short Term Memory) solves the vanishing gradient problem, addresses the memory issue and allows data to be persisted. The network uses an architecture of gates and does data handling. It is also capable of learning long-term dependencies i.e, a network which can remember information for a longer period of time.



Long Short Term Memory Networks (LSTMs) are a kind of RNN that can learn long-term dependencies. All recurrent neural networks are made up of a chain of repeated neural network modules. This repeating module in conventional RNNs will have a relatively basic structure, such as a single tanh layer. LSTMs have a chain-like structure as well, but the repeating module is different. Instead of a single neural network layer, there are four, each of which interacts in a unique way. Only the key sequences of words are remembered by LSTM, whereas unimportant words that do not contribute value to the prediction are forgotten.

The input gate ik, output gate ok, and forget gate fk are the three gates that make up the LSTM. Based on the dropout value, these gates determine which information is relevant for classification and which information is forgettable. Previous input is saved in cell memory block CK, which is required for prediction.

A dropout layer to filter out 30% of units in between and then go to the LSTM layer of 100 units. Post LSTM layer, another dropout layer is used and finally, a fully connected layer containing one unit with the sigmoid activation function is used binary classification i.e, classifying the news.

In our model,

1.   The embedding feature vectors, which are target feature vectors for the embedding layer, have a value of 40.

2.   A single 100-node LSTM layer is utilized.

3.   Because this is a binary classification task, a Dense Layer with 1 neuron and sigmoid activation function is applied.

4.   To avoid overfitting, the dropout technique is utilized, and the Adam optimizer is used to optimize the loss function.

**Classification Report: (Accuracy, Precision, Recall, F1 score)**

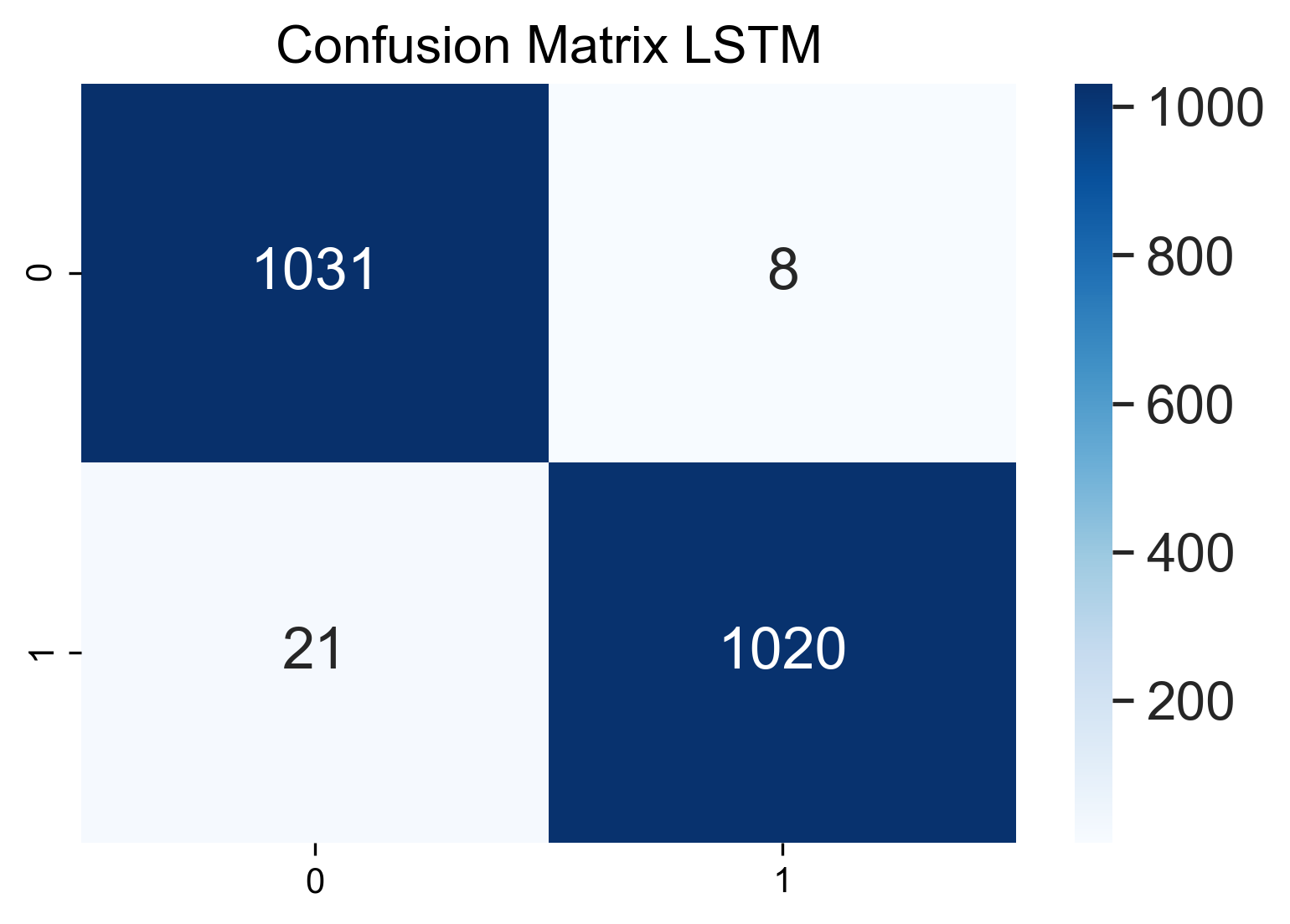
Dataset 1:

0.9860576923076924

0.9922178988326849

0.9798270893371758

0.985983566940551



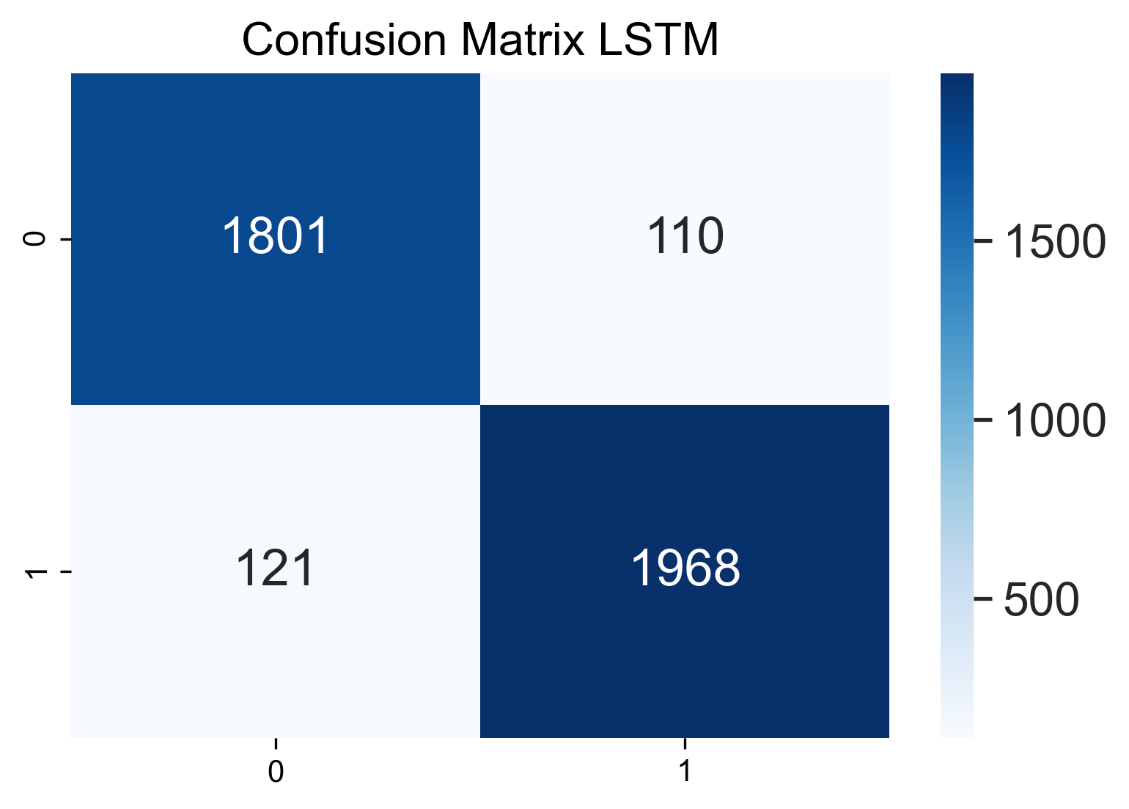
Dataset 2:

0.94225

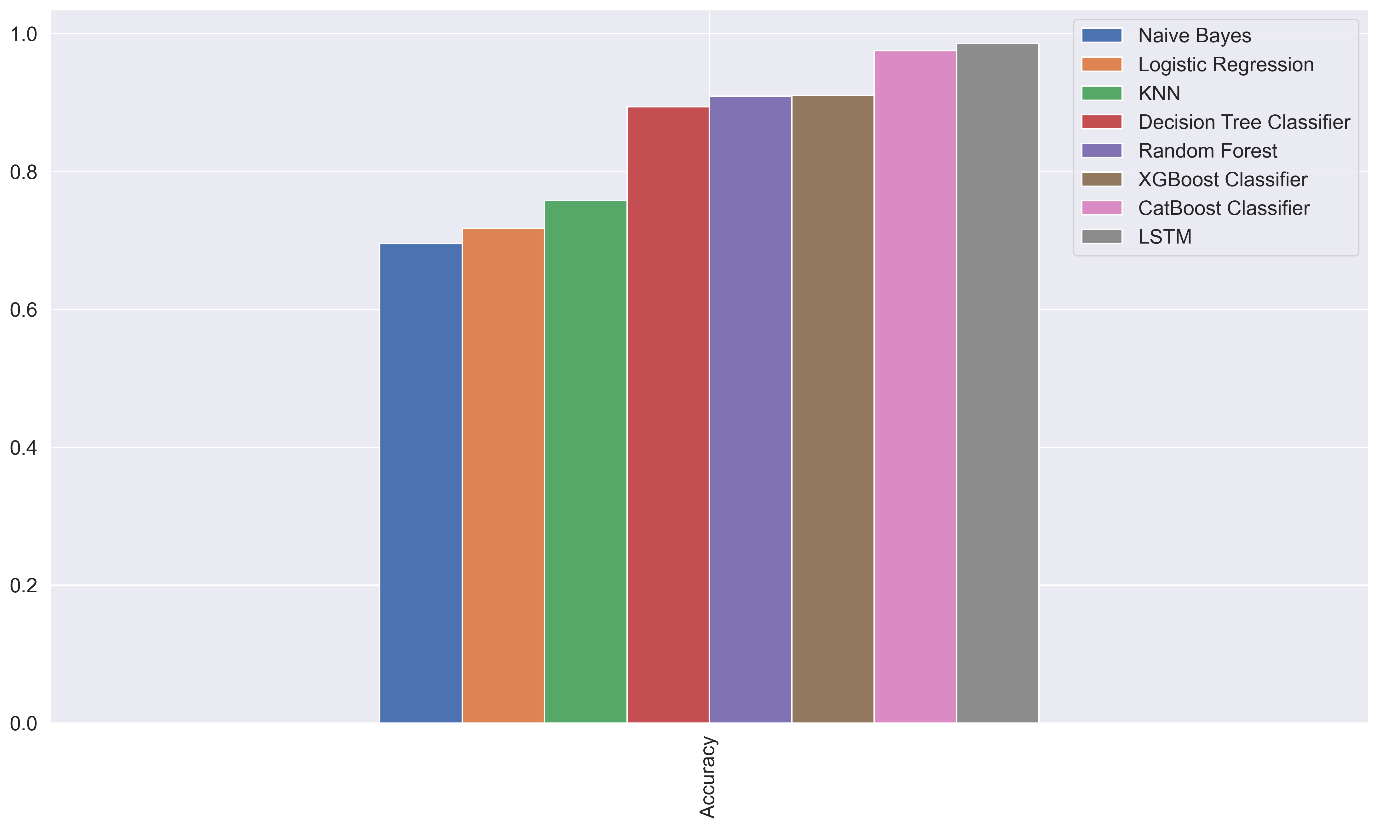
0.9470644850818094

0.9420775490665391

0.9445644348452124



**Model Comparison For Dataset 1 in Ascending Order:**



**Model Comparison For Dataset 2 in Ascending Order:**

