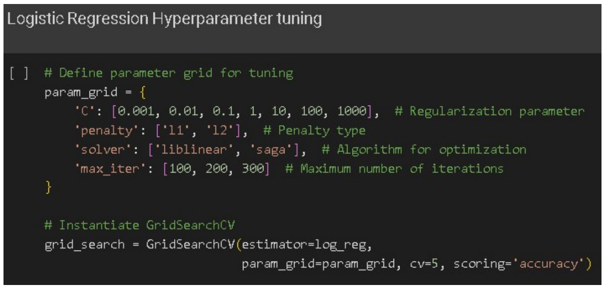
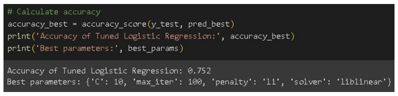
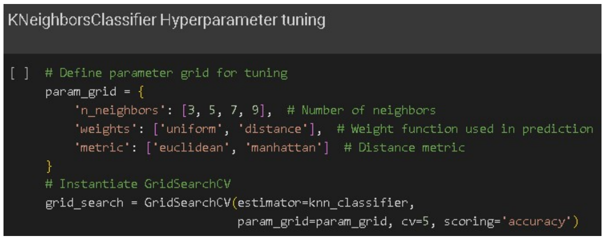
Model Optimization and Tuning Phase Template

Date	15 March 2024
Team ID	738214
Project Title	Predicting Mental Health Illness Of Working Professionals Using Machine Learning.
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (6 Marks):

Model	Tuned Hyperparameters	Optimal Values
Logistic Regression		
KNeighbors Classifier		

Logistic Regression

Logistic Regression Hyperparameter tuning

```
# Define parameter grid for tuning
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],  # Regularization parameter
    'penalty': ['l1', 'l2'],  # Penalty type
    'solver': ['liblinear', 'saga'],  # Algorithm for optimization
    'max_iter': [100, 200, 300]  # Maximum number of iterations
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=log_reg,
                           param_grid=param_grid, cv=5, scoring='accuracy')
```

```
# Calculate accuracy
accuracy_best = accuracy_score(y_test, pred_best)
print('Accuracy of Tuned Logistic Regression:', accuracy_best)
print('Best parameters:', best_params)
```

```
Accuracy of Tuned Logistic Regression: 0.752
Best parameters: {'C': 10, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
```

KNeighbors Classifier

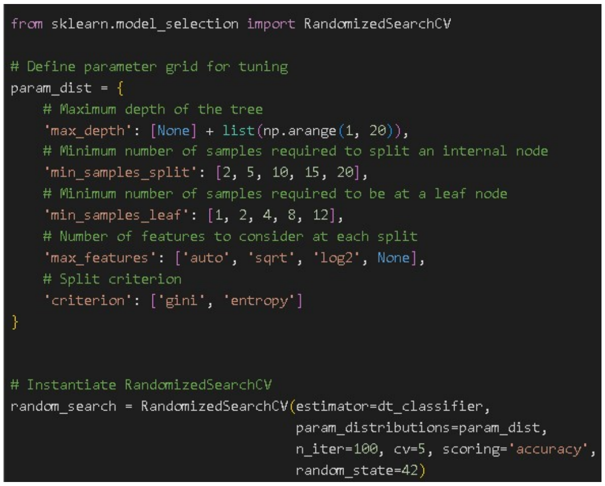
KNeighborsClassifier Hyperparameter tuning

```
# Define parameter grid for tuning
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # Number of neighbors
    'weights': ['uniform', 'distance'],  # Weight function used in prediction
    'metric': ['euclidean', 'manhattan']  # Distance metric
}
# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=knn_classifier,
                          param_grid=param_grid, cv=5, scoring='accuracy')
```

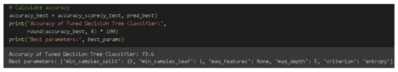
```
# Calculate accuracy
accuracy_best = accuracy_score(y_test, pred_best)
print('Accuracy of Tuned K-Nearest Neighbors:', round(accuracy_best, 4) * 100)
print('Best parameters:', best_params)
```

```
Accuracy of Tuned K-Nearest Neighbors: 70.13000000000001
Best parameters: {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'distance'}
```

Decision Tree



```
from sklearn.model_selection import RandomizedSearchCV

# Define parameter grid for tuning
param_dist = {
    # Maximum depth of the tree
    'max_depth': [None] + list(np.arange(1, 20)),
    # Minimum number of samples required to split an internal node
    'min_samples_split': [2, 5, 10, 15, 20],
    # Minimum number of samples required to be at a leaf node
    'min_samples_leaf': [1, 2, 4, 8, 12],
    # Number of features to consider at each split
    'max_features': ['auto', 'sqrt', 'log2', None],
    # Split criterion
    'criterion': ['gini', 'entropy']
}


# Instantiate RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=dt_classifier,
                                   param_distributions=param_dist,
                                   n_iter=100, cv=5, scoring='accuracy',
                                   random_state=42)
```

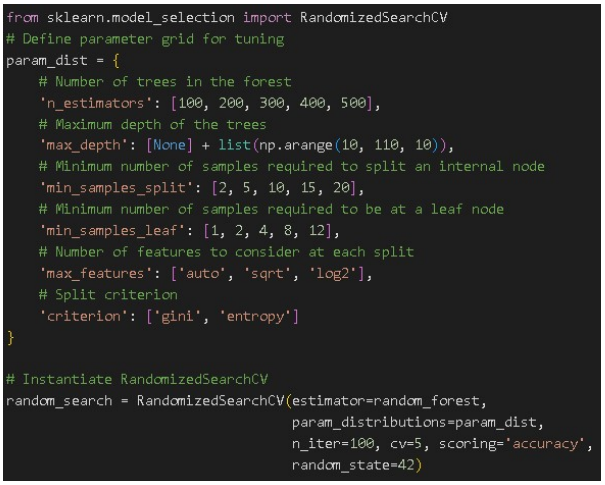


```
# Calculate accuracy
accuracy_best = accuracy_score(y_test, pred_best)
print('Accuracy of Tuned Decision Tree Classifier:',
      round(accuracy_best, 4) * 100)
print('Best parameters:', best_params)

Accuracy of Tuned Decision Tree Classifier: 73.6
Best parameters: {'min_samples_split': 15, 'min_samples_leaf': 1, 'max_features': None, 'max_depth': 5, 'criterion': 'entropy'}
```

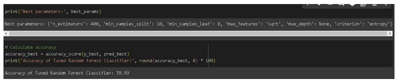
Random

Forest

Classifier



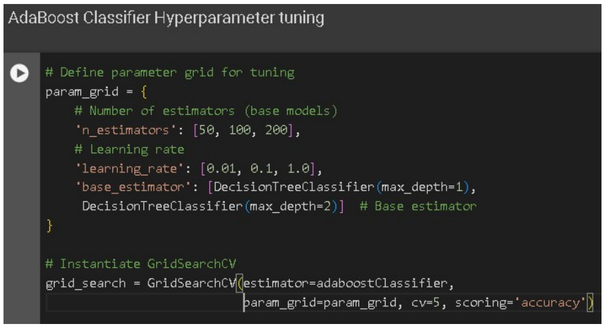
```
from sklearn.model_selection import RandomizedSearchCV
# Define parameter grid for tuning
param_dist = {
    # Number of trees in the forest
    'n_estimators': [100, 200, 300, 400, 500],
    # Maximum depth of the trees
    'max_depth': [None] + list(np.arange(10, 110, 10)),
    # Minimum number of samples required to split an internal node
    'min_samples_split': [2, 5, 10, 15, 20],
    # Minimum number of samples required to be at a leaf node
    'min_samples_leaf': [1, 2, 4, 8, 12],
    # Number of features to consider at each split
    'max_features': ['auto', 'sqrt', 'log2'],
    # Split criterion
    'criterion': ['gini', 'entropy']
}

# Instantiate RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=random_forest,
                                   param_distributions=param_dist,
                                   n_iter=100, cv=5, scoring='accuracy',
                                   random_state=42)
```



```
print('Best parameters:', best_params)

Best parameters: {'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 8, 'max_features': 'sqrt', 'max_depth': None, 'criterion': 'entropy'}

# Calculate accuracy
accuracy_best = accuracy_score(y_test, pred_best)
print('Accuracy of Tuned Random Forest Classifier:', round(accuracy_best, 4) * 100)

Accuracy of Tuned Random Forest Classifier: 78.93
```

AdaBoost

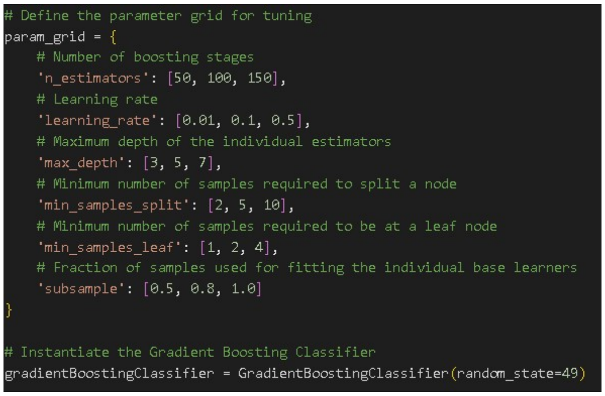
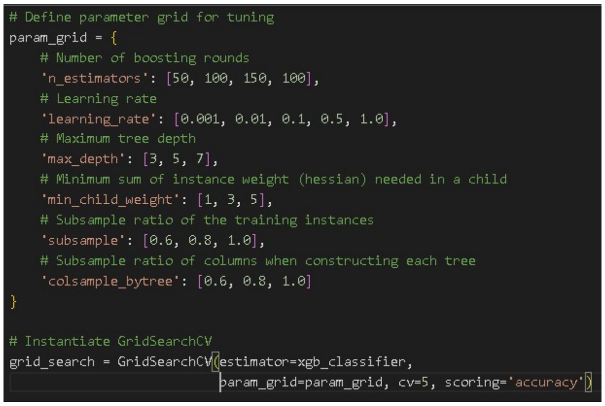
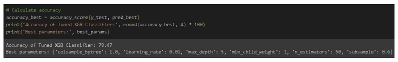
Classifier



AdaBoost Classifier Hyperparameter tuning

```
# Define parameter grid for tuning
param_grid = {
    # Number of estimators (base models)
    'n_estimators': [50, 100, 200],
    # Learning rate
    'learning_rate': [0.01, 0.1, 1.0],
    'base_estimator': [DecisionTreeClassifier(max_depth=1),
     DecisionTreeClassifier(max_depth=2)]  # Base estimator
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=adaboostClassifier,
                           param_grid=param_grid, cv=5, scoring='accuracy')
```

```
# Calculate accuracy
accuracy_best = accuracy_score(y_test, pred_best)
print('Accuracy of Tuned AdaBoost Classifier:', round(accuracy_best, 4) * 100)
print('Best parameters:', best_params)

Accuracy of Tuned AdaBoost Classifier: 78.93
Best parameters: {'base_estimator': DecisionTreeClassifier(max_depth=2), 'learning_rate': 0.01, 'n_estimators': 200}
```

Gradient Boosting Classifier		
XGB Classifier		

Gradient Boosting Classifier:

```
# Define the parameter grid for tuning
param_grid = {
    # Number of boosting stages
    'n_estimators': [50, 100, 150],
    # Learning rate
    'learning_rate': [0.01, 0.1, 0.5],
    # Maximum depth of the individual estimators
    'max_depth': [3, 5, 7],
    # Minimum number of samples required to split a node
    'min_samples_split': [2, 5, 10],
    # Minimum number of samples required to be at a leaf node
    'min_samples_leaf': [1, 2, 4],
    # Fraction of samples used for fitting the individual base learners
    'subsample': [0.5, 0.8, 1.0]
}

# Instantiate the Gradient Boosting Classifier
gradientBoostingClassifier = GradientBoostingClassifier(random_state=49)
```

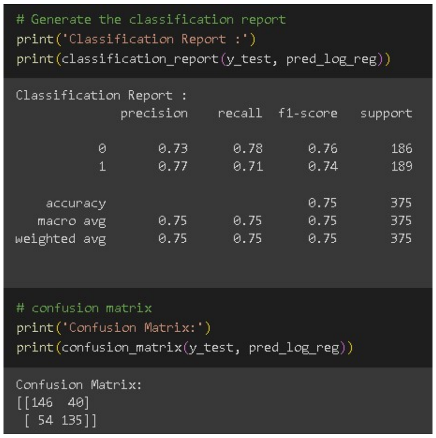
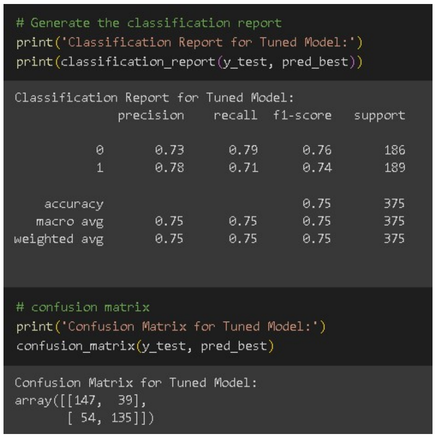
```
# Calculate accuracy
accuracy_best = accuracy_score(y_test, pred_best)
print('Accuracy of Tuned Gradient Boosting Classifier:', round(accuracy_best, 4) * 100)
print('Best parameters:', best_params)

Accuracy of Tuned Gradient Boosting Classifier: 78.93
Best parameters: {'learning_rate': 0.01, 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100, 'subsample': 0.5}
```

XGB Classifier:

```
# Define parameter grid for tuning
param_grid = {
    # Number of boosting rounds
    'n_estimators': [50, 100, 150, 100],
    # Learning rate
    'learning_rate': [0.001, 0.01, 0.1, 0.5, 1.0],
    # Maximum tree depth
    'max_depth': [3, 5, 7],
    # Minimum sum of instance weight (hessian) needed in a child
    'min_child_weight': [1, 3, 5],
    # Subsample ratio of the training instances
    'subsample': [0.6, 0.8, 1.0],
    # Subsample ratio of columns when constructing each tree
    'colsample_bytree': [0.6, 0.8, 1.0]
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=xgb_classifier,
                           param_grid=param_grid, cv=5, scoring='accuracy')
```

```
# Calculate accuracy
accuracy_best = accuracy_score(y_test, pred_best)
print('Accuracy of Tuned XGB Classifier:', round(accuracy_best, 4) * 100)
print('Best parameters:', best_params)

Accuracy of Tuned XGB Classifier: 79.47
Best parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 50, 'subsample': 0.6}
```

Performance Metrics Comparison Report (2 Marks):

Model	Baseline Metric	Optimized Metric
Logistic Regression		

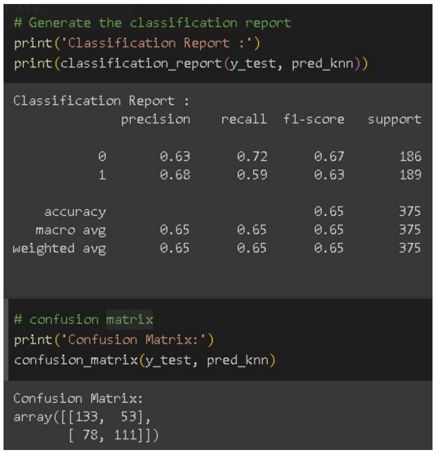
Logistic Regression — Baseline Metric:

```
# Generate the classification report
print('Classification Report :')
print(classification_report(y_test, pred_log_reg))

Classification Report :
              precision    recall  f1-score   support

           0       0.73      0.78      0.76       186
           1       0.77      0.71      0.74       189

    accuracy                           0.75       375
   macro avg       0.75      0.75      0.75       375
weighted avg       0.75      0.75      0.75       375

# confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, pred_log_reg))

Confusion Matrix:
[[146  40]
 [ 54 135]]
```

Logistic Regression — Optimized Metric:

```
# Generate the classification report
print('Classification Report for Tuned Model:')
print(classification_report(y_test, pred_best))

Classification Report for Tuned Model:
              precision    recall  f1-score   support

           0       0.73      0.79      0.76       186
           1       0.78      0.71      0.74       189

    accuracy                           0.75       375
   macro avg       0.75      0.75      0.75       375
weighted avg       0.75      0.75      0.75       375

# confusion matrix
print('Confusion Matrix for Tuned Model:')
confusion_matrix(y_test, pred_best)

Confusion Matrix for Tuned Model:
array([[147,  39],
       [ 54, 135]])
```

KNeighbors Classifier

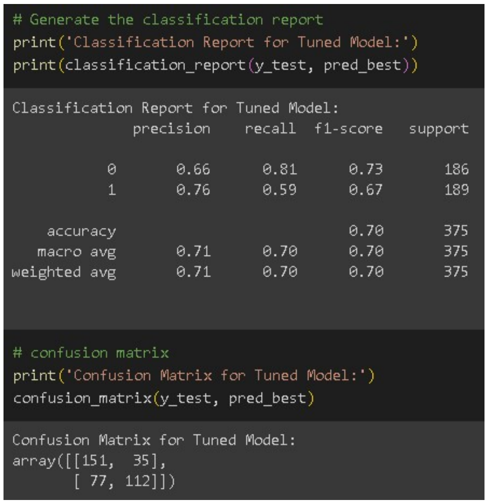


```
# Generate the classification report
print('Classification Report :')
print(classification_report(y_test, pred_knn))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.63      0.72      0.67       186
           1       0.68      0.59      0.63       189

    accuracy                           0.65       375
   macro avg       0.65      0.65      0.65       375
weighted avg       0.65      0.65      0.65       375
```

```
# confusion matrix
print('Confusion Matrix:')
confusion_matrix(y_test, pred_knn)
```

```
Confusion Matrix:
array([[133,  53],
       [ 78, 111]])
```



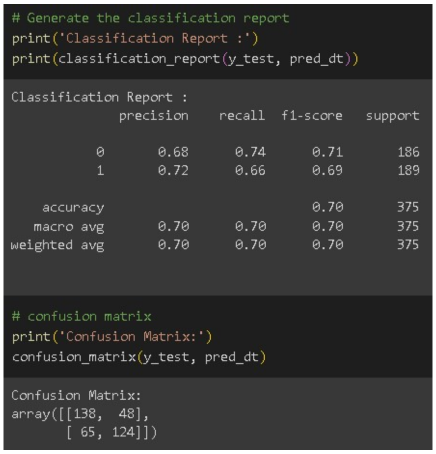
```
# Generate the classification report
print('Classification Report for Tuned Model:')
print(classification_report(y_test, pred_best))
```

```
Classification Report for Tuned Model:
              precision    recall  f1-score   support

           0       0.66      0.81      0.73       186
           1       0.76      0.59      0.67       189

    accuracy                           0.70       375
   macro avg       0.71      0.70      0.70       375
weighted avg       0.71      0.70      0.70       375
```

```
# confusion matrix
print('Confusion Matrix for Tuned Model:')
confusion_matrix(y_test, pred_best)
```

```
Confusion Matrix for Tuned Model:
array([[151,  35],
       [ 77, 112]])
```

Decision Tree

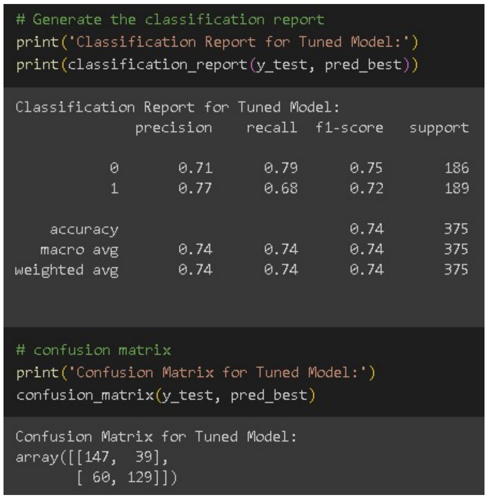


```
# Generate the classification report
print('Classification Report :')
print(classification_report(y_test, pred_dt))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.68      0.74      0.71       186
           1       0.72      0.66      0.69       189

    accuracy                           0.70       375
   macro avg       0.70      0.70      0.70       375
weighted avg       0.70      0.70      0.70       375
```

```
# confusion matrix
print('Confusion Matrix:')
confusion_matrix(y_test, pred_dt)
```

```
Confusion Matrix:
array([[138,  48],
       [ 65, 124]])
```



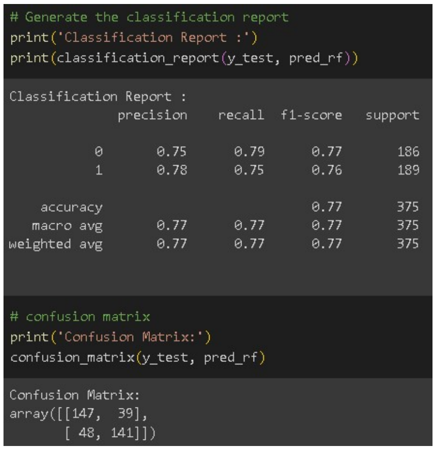
```
# Generate the classification report
print('Classification Report for Tuned Model:')
print(classification_report(y_test, pred_best))
```

```
Classification Report for Tuned Model:
              precision    recall  f1-score   support

           0       0.71      0.79      0.75       186
           1       0.77      0.68      0.72       189

    accuracy                           0.74       375
   macro avg       0.74      0.74      0.74       375
weighted avg       0.74      0.74      0.74       375
```

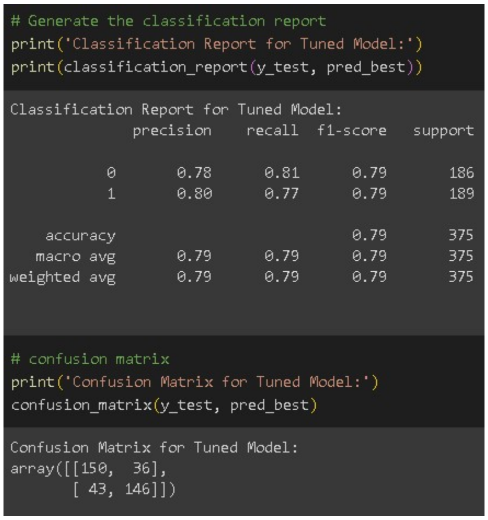
```
# confusion matrix
print('Confusion Matrix for Tuned Model:')
confusion_matrix(y_test, pred_best)
```

```
Confusion Matrix for Tuned Model:
array([[147,  39],
       [ 60, 129]])
```

Random Forest Classifier

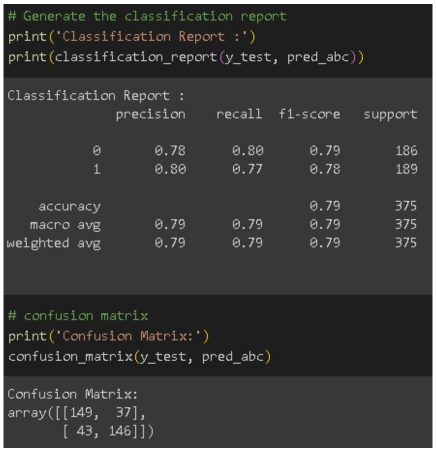


```
# Generate the classification report
print('Classification Report :')
print(classification_report(y_test, pred_rf))

Classification Report :
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       186
           1       0.78      0.75      0.76       189

    accuracy                           0.77       375
   macro avg       0.77      0.77      0.77       375
weighted avg       0.77      0.77      0.77       375

# confusion matrix
print('Confusion Matrix:')
confusion_matrix(y_test, pred_rf)

Confusion Matrix:
array([[147,  39],
       [ 48, 141]])
```

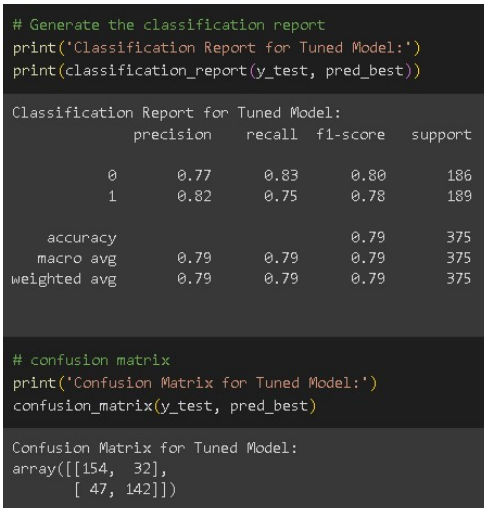


```
# Generate the classification report
print('Classification Report for Tuned Model:')
print(classification_report(y_test, pred_best))

Classification Report for Tuned Model:
              precision    recall  f1-score   support

           0       0.78      0.81      0.79       186
           1       0.80      0.77      0.79       189

    accuracy                           0.79       375
   macro avg       0.79      0.79      0.79       375
weighted avg       0.79      0.79      0.79       375

# confusion matrix
print('Confusion Matrix for Tuned Model:')
confusion_matrix(y_test, pred_best)

Confusion Matrix for Tuned Model:
array([[150,  36],
       [ 43, 146]])
```

AdaBoost Classifier

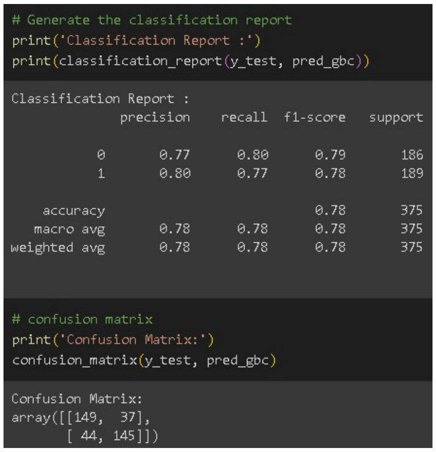


```
# Generate the classification report
print('Classification Report :')
print(classification_report(y_test, pred_abc))

Classification Report :
              precision    recall  f1-score   support

           0       0.78      0.80      0.79       186
           1       0.80      0.77      0.78       189

    accuracy                           0.79       375
   macro avg       0.79      0.79      0.79       375
weighted avg       0.79      0.79      0.79       375

# confusion matrix
print('Confusion Matrix:')
confusion_matrix(y_test, pred_abc)

Confusion Matrix:
array([[149,  37],
       [ 43, 146]])
```



```
# Generate the classification report
print('Classification Report for Tuned Model:')
print(classification_report(y_test, pred_best))

Classification Report for Tuned Model:
              precision    recall  f1-score   support

           0       0.77      0.83      0.80       186
           1       0.82      0.75      0.78       189

    accuracy                           0.79       375
   macro avg       0.79      0.79      0.79       375
weighted avg       0.79      0.79      0.79       375

# confusion matrix
print('Confusion Matrix for Tuned Model:')
confusion_matrix(y_test, pred_best)

Confusion Matrix for Tuned Model:
array([[154,  32],
       [ 47, 142]])
```

Gradient Boosting Classifier

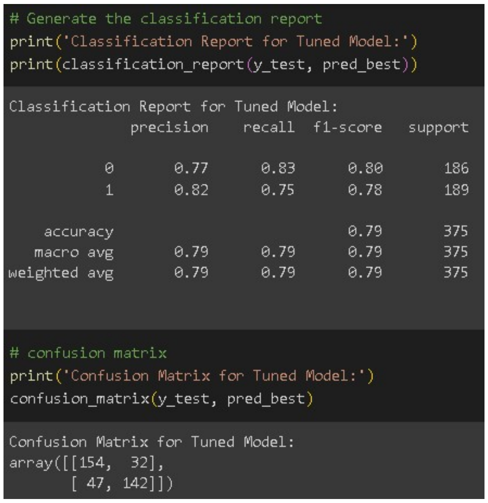


```
# Generate the classification report
print('Classification Report :')
print(classification_report(y_test, pred_gbc))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.77      0.80      0.79       186
           1       0.80      0.77      0.78       189

    accuracy                           0.78       375
   macro avg       0.78      0.78      0.78       375
weighted avg       0.78      0.78      0.78       375
```

```
# confusion matrix
print('Confusion Matrix:')
confusion_matrix(y_test, pred_gbc)
```

```
Confusion Matrix:
array([[149,  37],
       [ 44, 145]])
```



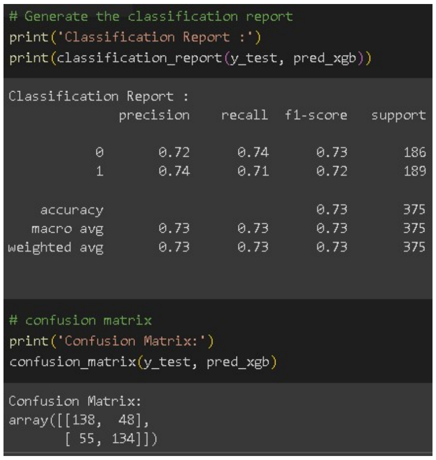
```
# Generate the classification report
print('Classification Report for Tuned Model:')
print(classification_report(y_test, pred_best))
```

```
Classification Report for Tuned Model:
              precision    recall  f1-score   support

           0       0.77      0.83      0.80       186
           1       0.82      0.75      0.78       189

    accuracy                           0.79       375
   macro avg       0.79      0.79      0.79       375
weighted avg       0.79      0.79      0.79       375
```

```
# confusion matrix
print('Confusion Matrix for Tuned Model:')
confusion_matrix(y_test, pred_best)
```

```
Confusion Matrix for Tuned Model:
array([[154,  32],
       [ 47, 142]])
```

XGB Classifier

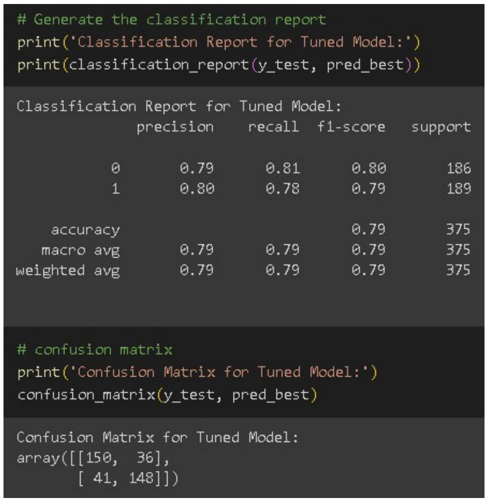


```
# Generate the classification report
print('Classification Report :')
print(classification_report(y_test, pred_xgb))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.72      0.74      0.73       186
           1       0.74      0.71      0.72       189

    accuracy                           0.73       375
   macro avg       0.73      0.73      0.73       375
weighted avg       0.73      0.73      0.73       375
```

```
# confusion matrix
print('Confusion Matrix:')
confusion_matrix(y_test, pred_xgb)
```

```
Confusion Matrix:
array([[138,  48],
       [ 55, 134]])
```



```
# Generate the classification report
print('Classification Report for Tuned Model:')
print(classification_report(y_test, pred_best))
```

```
Classification Report for Tuned Model:
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       186
           1       0.80      0.78      0.79       189

    accuracy                           0.79       375
   macro avg       0.79      0.79      0.79       375
weighted avg       0.79      0.79      0.79       375
```

```
# confusion matrix
print('Confusion Matrix for Tuned Model:')
confusion_matrix(y_test, pred_best)
```

```
Confusion Matrix for Tuned Model:
array([[150,  36],
       [ 41, 148]])
```

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
XGB Classifier	XGB Model was selected for its superior performance, exhibiting high accuracy during hyper parameter tuning. Optimized predictive accuracy aligns with project objectives, justifying its selection as the final model.