Code Documention

By:-

- Mohamed Khaled Ismail - 7000463
- Yahia Ehab - 7037125
- Mariam Amr elDeeb - 7001053

Data Preprocessing

Variables:-

- `df` : Dataframe that contains the csv file's features and labels.
- `scaler` : StandardScaler() variable for scaling the numerical features.
- `label_encoder` : To encode the the categorical features.

Implementation:-

```
df = pd.read_csv("./AMLAss1Datasets/bankloan.csv")
```

Used for reading the csv and suitable for regression.

```
df = df[
[
        "Age", "Experience", "Income",
        "Family", "CCAvg", "Education",
        "Mortgage", "Personal.Loan", "Securities.Account",
        "CD.Account", "Online", "CreditCard",
    ]
]
```

Choosing the only columns that we will use

```
df.columns = [
    "age", "experience", "income", "family",
    "cc_avg","education", "mortgage",
    "personal_loan", "securities_account",
    "cd_account", "online", "credit_card",
]
```

Renaming the columns

```
scaler = StandardScaler()
df[["age", "experience", "income", "cc_avg", "mortgage"]] = scaler.fit_transform(
    df[["age", "experience", "income", "cc_avg", "mortgage"]]
)
```

Scaling the numerical features

```
label_encoder = LabelEncoder()
df["education"] = label_encoder.fit_transform(df["education"])
```

Encoding the categorical features

Bankloan Dataset - Yahia Ehab

1. Training, testing and splitting

```
X = df.drop("personal_loan", axis=1)
y = df["personal_loan"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Random Forest

- Applying the random forest classifier, and evaluating it:-

```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

print("Random Forest Classifier")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred,zero_division=1))
print("Recall:", recall_score(y_test, y_pred,zero_division=1))
print("F1 Score:", f1_score(y_test, y_pred,zero_division=1))
```

- Setting a range of different parameters values:-

```
param_dist = {
    "n_estimators": [100, 200, 300],
    "max_depth": [None, 10, 20],
    "max_features": ["auto", "sqrt"],
}
```

- Applying RandomizedSearchCV to the random forest model with the parametres tuned:-

```
rf_random_search = RandomizedSearchCV(
    rf_model, param_distributions=param_dist, n_iter=50, cv=3)

rf_random_search.fit(X_train, y_train)

rf_random_search.best_params_
```

- Applying GridSearchCV to the random forest model with the parametres tuned:-

```
rf_grid_search = GridSearchCV(rf_model, param_grid=param_dist, cv=3, n_jobs=-1)

rf_grid_search.fit(X_train, y_train)

rf_grid_search.best_params_
```

- Applying Random Forest classifier with the best parametres and printing evaluations:-

```
# Random Forest Classifier with best hyperparameters
MAX_DEPTH = 10
BOOTSTRAP = False
MIN_SAMPLES_LEAF = 1
MIN_SAMPLES_SPLIT = 2
N_ESTIMATORS = 150
MAX_FEATURES = 'sqrt'

rf_model_best = RandomForestClassifier(
    max_depth=MAX_DEPTH,
    bootstrap=BOOTSTRAP,
    min_samples_leaf=MIN_SAMPLES_LEAF,
    min_samples_split=MIN_SAMPLES_SPLIT,
    n_estimators=N_ESTIMATORS,
    max_features=MAX_FEATURES,
)

rf_model_best.fit(X_train, y_train)
y_pred_best = rf_model_best.predict(X_test)

# Model evaluation
print("Random Forest Classifier with best hyperparameters")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred,zero_division=1))
print("Recall:", recall_score(y_test, y_pred,zero_division=1))
print("F1 Score:", f1_score(y_test, y_pred,zero_division=1))
```

AdaBoost

- Applying the AdaBoost and evaluating it:-

```
# AdaBoost Classifier
ada_model = AdaBoostClassifier()
ada_model.fit(X_train, y_train)
y_pred = ada_model.predict(X_test)

# Model evaluation
print("AdaBoost Classifier")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred,zero_division=1))
print("Recall:", recall_score(y_test, y_pred,zero_division=1))
print("F1 Score:", f1_score(y_test, y_pred,zero_division=1))
```

- Setting a range of different parameters values:-

```
param_dist = {
    "n_estimators": [50, 100, 150],
    "learning_rate": [0.01, 0.1, 1],
}
```

- Applying RandomizedSearchCV to the AdaBoost model with the parametres tuned:-

```
ada_random_search = RandomizedSearchCV(
    ada_model, param_distributions=param_dist, n_iter=50, cv=3)

ada_random_search.fit(X_train, y_train)

ada_random_search.best_params_
```

- Applying GridSearchCV to the AdaBoost model with the parametres tuned:-

```
ada_grid_search = GridSearchCV(ada_model, param_grid=param_dist, cv=3, n_jobs=-1)

ada_grid_search.fit(X_train, y_train)

ada_grid_search.best_params_
```

- Applying AdaBoost classifier with the best parametres and printing evaluations:-

```
# AdaBoost Classifier with best hyperparameters
N_ESTIMATORS = 150
LEARNING_RATE = 1

ada_model_best = AdaBoostClassifier(
    n_estimators=N_ESTIMATORS,
    learning_rate=LEARNING_RATE,
)

ada_model_best.fit(X_train, y_train)
y_pred_best = ada_model_best.predict(X_test)

# Model evaluation
print("AdaBoost Classifier with best hyperparameters")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred,zero_division=1))
print("Recall:", recall_score(y_test, y_pred,zero_division=1))
print("F1 Score:", f1_score(y_test, y_pred,zero_division=1))
```

Gradient Boost

- Applying the Gradient Boost and evaluating it:-

```
 # Gradient Boosting Classifier
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)
y_pred = gb_model.predict(X_test)

print("Gradient Boosting Classifier")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred,zero_division=1))
```

```
print("Recall:", recall_score(y_test, y_pred,zero_division=1))
print("F1 Score:", f1_score(y_test, y_pred,zero_division=1))
```

- Setting a range of different parameters values:-

```
# Hyperparameters
param_dist = {
    "n_estimators": [50, 100, 150],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.1, 1],
}
```

- Applying RandomizedSearchCV to the Gradient Boost model with the parametres tuned:-

```
gb_random_search = RandomizedSearchCV(
    gb_model, param_distributions=param_dist, n_iter=50, cv=3)

gb_random_search.fit(X_train, y_train)

gb_random_search.best_params_
```

- Applying GridSearchCV to the Gradient Boost model with the parametres tuned:-

```
gb_grid_search = GridSearchCV(gb_model, param_grid=param_dist, cv=3, n_jobs=-1)

gb_grid_search.fit(X_train, y_train)

gb_grid_search.best_params_
```

- Applying Gradient Boost classifier witht the best parametres and printing evaluations:-

```
# Gradient Boosting Classifier with best hyperparameters
N_ESTIMATORS = 150
MAX_DEPTH = 3
LEARNING_RATE = 0.1

gb_model_best = GradientBoostingClassifier(
    n_estimators=N_ESTIMATORS,
    max_depth=MAX_DEPTH,
    learning_rate=LEARNING_RATE,
)

gb_model_best.fit(X_train, y_train)
y_pred_best = gb_model_best.predict(X_test)

print("Gradient Boosting Classifier with best hyperparameters")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred,zero_division=1))
print("Recall:", recall_score(y_test, y_pred,zero_division=1))
print("F1 Score:", f1_score(y_test, y_pred,zero_division=1))
```

Final Comparison

- Table to compare the performance of the models on the dataset:-

```
# Comparing the performance of the models
models = [
    "Random Forest Classifier",
    "Random Forest Classifier with best hyperparameters",
    "AdaBoost Classifier",
    "AdaBoost Classifier with best hyperparameters",
    "Gradient Boosting Classifier",
    "Gradient Boosting Classifier with best hyperparameters",
]

accuracies = [
    accuracy_score(y_test, rf_model.predict(X_test)),
    accuracy_score(y_test, rf_model_best.predict(X_test)),
    accuracy_score(y_test, ada_model.predict(X_test)),
    accuracy_score(y_test, ada_model_best.predict(X_test)),
    accuracy_score(y_test, gb_model.predict(X_test)),
    accuracy_score(y_test, gb_model_best.predict(X_test)),
]

precisions = [
    precision_score(y_test, rf_model.predict(X_test), zero_division=1),
    precision_score(y_test, rf_model_best.predict(X_test), zero_division=1),
    precision_score(y_test, ada_model.predict(X_test), zero_division=1),
    precision_score(y_test, ada_model_best.predict(X_test), zero_division=1),
    precision_score(y_test, gb_model.predict(X_test), zero_division=1),
    precision_score(y_test, gb_model_best.predict(X_test), zero_division=1),
]

recalls = [
    recall_score(y_test, rf_model.predict(X_test), zero_division=1),
    recall_score(y_test, rf_model_best.predict(X_test), zero_division=1),
    recall_score(y_test, ada_model.predict(X_test), zero_division=1),
    recall_score(y_test, ada_model_best.predict(X_test), zero_division=1),
    recall_score(y_test, gb_model.predict(X_test), zero_division=1),
    recall_score(y_test, gb_model_best.predict(X_test), zero_division=1),
]

f1_scores = [
    f1_score(y_test, rf_model.predict(X_test), zero_division=1),
    f1_score(y_test, rf_model_best.predict(X_test), zero_division=1),
    f1_score(y_test, ada_model.predict(X_test), zero_division=1),
    f1_score(y_test, ada_model_best.predict(X_test), zero_division=1),
    f1_score(y_test, gb_model.predict(X_test), zero_division=1),
    f1_score(y_test, gb_model_best.predict(X_test), zero_division=1),
]

model_comparison = pd.DataFrame({
    "Model": models,
```

```
    "Accuracy": accuracies,
    "Precision": precisions,
    "Recall": recalls,
    "F1 Score": f1_scores,
})

model_comparison
```

- Creating a barchart for each of the model:-

```
model_comparison.plot(kind="bar", x="Model", figsize=(10, 6))

plt.title("BankLoan Dataset Model Comparison")
plt.xlabel("Model")
plt.ylabel("Score")
plt.xticks(rotation=90)
plt.show()
```

Banknote Dataset - Mohamed Ismail

1. Training and testing the data:-

```
# Train test split
X = df.drop("class", axis=1)
y = df["class"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Random Forest

- Applying the Random Forest and evaluating it:-

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

- Setting a range of different parameters values:-

```
# Having a range of parameters to test
n = [64, 100, 128, 200]
```

```
max_features = ["sqrt", "log2"]
bootstrap = [True, False]
```

- Applying GridSearchCV() with Random Forest:-

```
param_grid = {
    "n_estimators": n,
    "max_features": max_features,
    "bootstrap": bootstrap,
}

clf = GridSearchCV(rf, param_grid, cv=10, scoring="accuracy")
clf.fit(X_train, y_train)
print("Best Parameters:", clf.best_params_)
print("Best Accuracy:", clf.best_score_)
```

- Training the model with the best parameters

```
# Training the model with the best parameters
rf_best = RandomForestClassifier(
    n_estimators=clf.best_params_["n_estimators"],
    max_features=clf.best_params_["max_features"],
    bootstrap=clf.best_params_["bootstrap"],
)

rf_best.fit(X_train, y_train)
```

- Testing and evalutating the model:-

```
# Predictions
y_pred = rf_best.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Displaying the prediction evals
print("Random Forest Classifier")
print("Accuracy:", accuracy)
print("Confusion Matrix:", cm)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

AdaBoost

- Applying the Random Forest and evaluating it:-

```
# AdaBoost
ada = AdaBoostClassifier()
ada.fit(X_train, y_train)
```

```
y_pred = ada.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

- Having a range of parametres:-

```
# Having a range of parameters to test
n_ada = [64, 100, 200, 500, 700, 1000]
learning_rate = [0.1, 0.5, 1, 1.5, 2.0]
random = [None, 42]
algorithm = ["SAMME"]
```

- Applying GridSearchCV() with the model:-

```
param_grid = {
    "n_estimators": n_ada,
    "learning_rate": learning_rate,
    "random_state": random,
    "algorithm": algorithm,
}

clf = GridSearchCV(ada, param_grid, cv=10, scoring="accuracy")
clf.fit(X_train, y_train)
print("Best Parameters:", clf.best_params_)
print("Best Accuracy:", clf.best_score_)
```

- Training the model with the best parameters:-

```
# Training the model with the best parameters
ada_best = AdaBoostClassifier(
    n_estimators=clf.best_params_["n_estimators"],
    learning_rate=clf.best_params_["learning_rate"],
    random_state=clf.best_params_["random_state"],
    algorithm=clf.best_params_["algorithm"],
)

ada_best.fit(X_train, y_train)
```

- Testing and evaluating the model:-

```
# Predictions
y_pred = ada_best.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Displaying the prediction evals
print("AdaBoost Classifier")
print("Accuracy:", accuracy)
print("Confusion Matrix:", cm)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Gradient Boost

- Applying and evaluating the model:-

```
# Gradient Boosting
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

- Having a range of parameters to test

```
n_gb = [64, 100, 200, 500, 700, 1000]
learning_rate = [0.1, 0.5, 1, 1.5, 2.0]
loss = ["log_loss", "exponential"]
```

- Applying GridSearchCV() with the model:-

```
param_grid = {
    "n_estimators": n_gb,
    "learning_rate": learning_rate,
    "loss": loss,
}

clf = GridSearchCV(gb, param_grid, cv=10, scoring="accuracy")
clf.fit(X_train, y_train)
print("Best Parameters:", clf.best_params_)
print("Best Accuracy:", clf.best_score_)
```

- Training the model with the best parameters

```
# Training the model with the best parameters
gb_best = GradientBoostingClassifier(
    n_estimators=clf.best_params_["n_estimators"],
    learning_rate=clf.best_params_["learning_rate"],
    loss=clf.best_params_["loss"],
)

gb_best.fit(X_train, y_train)
```

- Testing and evaluating the trained model with best parametres:-

```
# Predictions
y_pred = gb_best.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Displaying the prediction evals
print("Gradient Boosting Classifier")
print("Accuracy:", accuracy)
print("Confusion Matrix:", cm)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Final Comparison

- Table with the comparisons:-

```
models = [
    "Random Forest Classifier",
    "Random Forest Classifier with best hyperparameters",
    "AdaBoost Classifier",
    "AdaBoost Classifier with best hyperparameters",
    "Gradient Boosting Classifier",
    "Gradient Boosting Classifier with best hyperparameters",
]

accuracies = [
    accuracy_score(y_test, rf.predict(X_test)),
    accuracy_score(y_test, rf_best.predict(X_test)),
    accuracy_score(y_test, ada.predict(X_test)),
    accuracy_score(y_test, ada_best.predict(X_test)),
    accuracy_score(y_test, gb.predict(X_test)),
    accuracy_score(y_test, gb_best.predict(X_test)),
]

precisions = [
```

```
    precision_score(y_test, rf.predict(X_test), zero_division=1),
    precision_score(y_test, rf_best.predict(X_test), zero_division=1),
    precision_score(y_test, ada.predict(X_test), zero_division=1),
    precision_score(y_test, ada_best.predict(X_test), zero_division=1),
    precision_score(y_test, gb.predict(X_test), zero_division=1),
    precision_score(y_test, gb_best.predict(X_test), zero_division=1),
]

recalls = [
    recall_score(y_test, rf.predict(X_test), zero_division=1),
    recall_score(y_test, rf_best.predict(X_test), zero_division=1),
    recall_score(y_test, ada.predict(X_test), zero_division=1),
    recall_score(y_test, ada_best.predict(X_test), zero_division=1),
    recall_score(y_test, gb.predict(X_test), zero_division=1),
    recall_score(y_test, gb_best.predict(X_test), zero_division=1),
]

f1_scores = [
    f1_score(y_test, rf.predict(X_test), zero_division=1),
    f1_score(y_test, rf_best.predict(X_test), zero_division=1),
    f1_score(y_test, ada.predict(X_test), zero_division=1),
    f1_score(y_test, ada_best.predict(X_test), zero_division=1),
    f1_score(y_test, gb.predict(X_test), zero_division=1),
    f1_score(y_test, gb_best.predict(X_test), zero_division=1),
]

model_comparison = pd.DataFrame({
    "Model": models,
    "Accuracy": accuracies,
    "Precision": precisions,
    "Recall": recalls,
    "F1 Score": f1_scores,
})

model_comparison
```

- Plotting a bar chart to visualize the comparison:-

```
model_comparison.plot(kind="bar", x="Model", figsize=(10, 6))

plt.title("BankLoan Dataset Model Comparison")
plt.xlabel("Model")
plt.ylabel("Score")
plt.xticks(rotation=90)
plt.show()
```

Glass Type Dataset - Mariam El Deeb

1. Training and evaluating the model:-

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
```

Random Forest

- Testing and evaluating the model:-

```
accuracy_rf = accuracy_score(y_test, y_pred)
precision_rf = precision_score(y_test, y_pred, average="macro",zero_division=1)
recall_rf = recall_score(y_test, y_pred, average="macro",zero_division=1)
f1_rf = f1_score(y_test, y_pred, average="macro",zero_division=1)

print("Random Forest Classifier default parameters:")
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1 Score:", f1_rf)
```

- Having a range of parametres to choose from in testing:-

```
params={
    'n_estimators': [50, 100, 150, 200],
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5,10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['sqrt', 'log2']
}
```

- Applying RandomizedSearchCV() with the model:-

```
rf_random = RandomizedSearchCV(rf_model, param_distributions=params, n_iter=100, cv=5)

rf_random.fit(X_train, y_train)

print("Best parameters for Random Forest Classifier:")
print(rf_random.best_params_)
```

- Applying GridSearchCV() with the model:-

```
rf_grid = GridSearchCV(rf_model, param_grid=params, cv=5, n_jobs=-1)
rf_grid.fit(X_train, y_train)

print("Best parameters for Random Forest Classifier from Grid Search:")
print(rf_grid.best_params_)
```

- Comparison between default and tuned model:-

```
#Final Comparison between initial and tuned models

#Tuned model
```

```
rf_model_tuned = RandomForestClassifier(
    n_estimators=150,
    criterion='gini',
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    max_features='sqrt',
    random_state=42,
)
rf_model_tuned.fit(X_train, y_train)
y_pred_tuned = rf_model_tuned.predict(X_test)

accuracy_rf_tuned = accuracy_score(y_test, y_pred_tuned)
precision_rf_tuned = precision_score(y_test, y_pred_tuned,
average="macro",zero_division=1)
recall_rf_tuned = recall_score(y_test, y_pred_tuned, average="macro",zero_division=1)
f1_rf_tuned = f1_score(y_test, y_pred_tuned, average="macro",zero_division=1)

print("Random Forest Classifier tuned parameters:")
print("Accuracy:", accuracy_rf_tuned)
print("Precision:", precision_rf_tuned)
print("Recall:", recall_rf_tuned)
print("F1 Score:", f1_rf_tuned)

#Initial model
print("\n")
print("Random Forest Classifier default parameters:")
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1 Score:", f1_rf)
```

AdaBoost

- Training and testing and evaluating the model:-

```
ab_model = AdaBoostClassifier(random_state=42)
ab_model.fit(X_train, y_train)
y_pred_ab = ab_model.predict(X_test)

#Performance
accuracy_ab = accuracy_score(y_test, y_pred_ab)
precision_ab = precision_score(y_test, y_pred_ab, average="macro",zero_division=1)
recall_ab = recall_score(y_test, y_pred_ab, average="macro",zero_division=1)
f1_ab = f1_score(y_test, y_pred_ab, average="macro",zero_division=1)

print("AdaBoost Classifier default parameters performance:")
print("Accuracy:", accuracy_ab)
print("Precision:", precision_ab)
print("Recall:", recall_ab)
print("F1 Score:", f1_ab)
```

- Having a range of parametres to choose from while testing:=

```
params={
    'n_estimators': [50, 100, 150, 200],
    'learning_rate': [0.01, 0.1, 1, 10]
}
```

- Applying RandomSearchCV() with the model:-

```
ab_random = RandomizedSearchCV(ab_model, param_distributions=params, n_iter=100, cv=5)
ab_random.fit(X_train, y_train)

print("Best parameters for AdaBoost Classifier with Random Search:")
print(ab_random.best_params_)
```

- Applying GridSearchCV() with the model:-

```
ab_grid = GridSearchCV(ab_model, param_grid=params, cv=5)
ab_grid.fit(X_train, y_train)

print("Best parameters for AdaBoost Classifier from Grid Search:")
print(ab_grid.best_params_)
```

- Comparison between default and tuned model:-

```
#Final Comparison between initial and tuned models

#Tuned model
ab_model_tuned = AdaBoostClassifier(
    n_estimators=150,
    learning_rate=0.1,
    random_state=42,
)
ab_model_tuned.fit(X_train, y_train)
ab_y_pred_tuned = ab_model_tuned.predict(X_test)

accuracy_ab_tuned = accuracy_score(y_test, ab_y_pred_tuned)
precision_ab_tuned = precision_score(y_test, ab_y_pred_tuned,
average="macro",zero_division=1)
recall_ab_tuned = recall_score(y_test, ab_y_pred_tuned,
average="macro",zero_division=1)
f1_ab_tuned = f1_score(y_test, ab_y_pred_tuned, average="macro",zero_division=1)

print("AdaBoost Classifier tuned parameters:")
print("Accuracy:", accuracy_ab_tuned)
print("Precision:", precision_ab_tuned)
print("Recall:", recall_ab_tuned)
print("F1 Score:", f1_ab_tuned)

#Initial model
print("\n")
print("AdaBoost Classifier default parameters performance:")
```

```
print("Accuracy:", accuracy_ab)
print("Precision:", precision_ab)
print("Recall:", recall_ab)
print("F1 Score:", f1_ab)
```

Gradient Boost

- Training and testing and evaluating the model:-

```
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)
gb_y_pred = gb_model.predict(X_test)

#Performance
accuracy_gb = accuracy_score(y_test, gb_y_pred)
precision_gb = precision_score(y_test, gb_y_pred, average="macro",zero_division=1)
recall_gb = recall_score(y_test, gb_y_pred, average="macro",zero_division=1)
f1_gb = f1_score(y_test, gb_y_pred, average="macro",zero_division=1)

print("Gradient Boosting Classifier default parameters performance:")
print("Accuracy:", accuracy_gb)
print("Precision:", precision_gb)
print("Recall:", recall_gb)
print("F1 Score:", f1_gb)
```

- Having a range of parametres to choose from while testing:-

```
params={
    'n_estimators': [50, 100, 150, 200],
    'learning_rate': [0.01, 0.1, 1,1.5],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['sqrt', 'log2'],
    'learning_rate': [0.01, 0.1, 1, 10]
}
```

- Applying RandomizedSearchCV() with the model:-

```
gb_random = RandomizedSearchCV(gb_model, param_distributions=params, n_iter=100, cv=5)
gb_random.fit(X_train, y_train)

print("Best parameters for Gradient Boosting Classifier with Random Search:")
print(gb_random.best_params_)
```

- Applying GridSearchCV() with the model:-

```
gb_grid = GridSearchCV(gb_model, param_grid=params, cv=5)
gb_grid.fit(X_train, y_train)

print("Best parameters for Gradient Boosting Classifier from Grid Search:")
print(gb_grid.best_params_)
```

- Comparison between default and tuned model:-

```
#Final Comparison between initial and tuned models

#Tuned model
gb_model_tuned = GradientBoostingClassifier(
    n_estimators=150,
    learning_rate=0.1,
    max_depth=3,
    min_samples_split=5,
    min_samples_leaf=1,
    max_features='sqrt',
    random_state=42,
)
gb_model_tuned.fit(X_train, y_train)
gb_y_pred_tuned = gb_model_tuned_gs.predict(X_test)

accuracy_gb_tuned = accuracy_score(y_test, gb_y_pred_tuned)
precision_gb_tuned = precision_score(y_test, gb_y_pred_tuned,
average="macro",zero_division=1)
recall_gb_tuned = recall_score(y_test, gb_y_pred_tuned,
average="macro",zero_division=1)
f1_gb_tuned = f1_score(y_test, gb_y_pred_tuned, average="macro",zero_division=1)

print("\n")
print("Gradient Boosting Classifier tuned parameters:")
print("Accuracy:", accuracy_gb_tuned)
print("Precision:", precision_gb_tuned)
print("Recall:", recall_gb_tuned)
print("F1 Score:", f1_gb_tuned)

#Initial model
print("\n")
print("Gradient Boosting Classifier default parameters performance:")
print("Accuracy:", accuracy_gb)
print("Precision:", precision_gb)
print("Recall:", recall_gb)
print("F1 Score:", f1_gb)
```

Final Comparison

- Table to compare:-

```
# Comparing the performance of the models
models = [
    "Random Forest Classifier",
    "Random Forest Classifier with best hyperparameters",
    "AdaBoost Classifier",
    "AdaBoost Classifier with best hyperparameters",
    "Gradient Boosting Classifier",
    "Gradient Boosting Classifier with best hyperparameters",
]
```

```
accuracies = [
    accuracy_score(y_test, rf_model.predict(X_test)),
    accuracy_score(y_test, rf_model_tuned.predict(X_test)),
    accuracy_score(y_test, ab_model.predict(X_test)),
    accuracy_score(y_test, ab_model_tuned.predict(X_test)),
    accuracy_score(y_test, gb_model.predict(X_test)),
    accuracy_score(y_test, gb_model_tuned.predict(X_test)),
]

precisions = [
    precision_score(y_test, rf_model.predict(X_test),
average="macro",zero_division=1),
    precision_score(y_test, rf_model_tuned.predict(X_test),
average="macro",zero_division=1),
    precision_score(y_test, ab_model.predict(X_test),
average="macro",zero_division=1),
    precision_score(y_test, ab_model_tuned.predict(X_test),
average="macro",zero_division=1),
    precision_score(y_test, gb_model.predict(X_test),
average="macro",zero_division=1),
    precision_score(y_test, gb_model_tuned.predict(X_test),
average="macro",zero_division=1),
]

recalls = [
    recall_score(y_test, rf_model.predict(X_test), average="macro",zero_division=1),
    recall_score(y_test, rf_model_tuned.predict(X_test),
average="macro",zero_division=1),
    recall_score(y_test, ab_model.predict(X_test), average="macro",zero_division=1),
    recall_score(y_test, ab_model_tuned.predict(X_test),
average="macro",zero_division=1),
    recall_score(y_test, gb_model.predict(X_test), average="macro",zero_division=1),
    recall_score(y_test, gb_model_tuned.predict(X_test),
average="macro",zero_division=1),
]

f1_scores = [
    f1_score(y_test, rf_model.predict(X_test), average="macro",zero_division=1),
    f1_score(y_test, rf_model_tuned.predict(X_test), average="macro",zero_division=1),
    f1_score(y_test, ab_model.predict(X_test), average="macro",zero_division=1),
    f1_score(y_test, ab_model_tuned.predict(X_test), average="macro",zero_division=1),
    f1_score(y_test, gb_model.predict(X_test), average="macro",zero_division=1),
    f1_score(y_test, gb_model_tuned.predict(X_test), average="macro",zero_division=1),
]

model_comparison = pd.DataFrame({
    "Model": models,
    "Accuracy": accuracies,
    "Precision": precisions,
    "Recall": recalls,
    "F1 Score": f1_scores,
```

```
})

model_comparison
```

- Visualizing results via barchart:

```
model_comparison.plot(kind="bar", x="Model", figsize=(10, 6))

plt.title("Glass Type Prediction Model Comparison")
plt.xlabel("Model")
plt.ylabel("Score")
plt.xticks(rotation=90)
plt.show()
```
