```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report


# Step 1: Load Data
df = pd.read_csv('ecg_features_with_arrhythmia.csv')


# Step 2: Drop ID columns if any
if 'record_id' in df.columns:
    df = df.drop(columns=['record_id'])


# Step 3: Encode target label
le = LabelEncoder()
df['Arrhythmia'] = le.fit_transform(df['Arrhythmia'])


# Step 4: Feature and Target split
X = df.drop(columns=['Arrhythmia'])
y = df['Arrhythmia']


# Step 5: Train-Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)


# Step 6: Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Step 7: Models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'),
    "KNN": KNeighborsClassifier()
}


# Step 7: Models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'),
    "KNN": KNeighborsClassifier()
}

# Instead of using MultiOutputClassifier, fit the model directly:
# model = MultiOutputClassifier(RandomForestClassifier())
# model.fit(X_train, y_train)
# y_pred = model.predict(X_test)

# ... (rest of your code)

# Step 8: Training and Evaluation
for name, model in models.items():
    print(f"\n=== {name} ===")
    model.fit(X_train, y_train)  # This will now work correctly
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"Accuracy: {acc:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred, target_names=le.classes_)) # Assuming 'le' is your LabelEncoder
```

```
                  macro avg       1.00      1.00      1.00      9030
```

```
                  macro avg       1.00      1.00      1.00      9030
               weighted avg       1.00      1.00      1.00      9030


=== SVM ===
Accuracy: 0.9867
F1 Score: 0.9856
Classification Report:
                           precision    recall  f1-score   support

              Bradycardia       0.99      1.00      0.99      3513
Low HRV (Arrhythmia Risk)       1.00      0.43      0.61        92
                   Normal       0.99      0.98      0.98      3011
              Tachycardia       0.99      1.00      0.99      2414

                 accuracy                           0.99      9030
                macro avg       0.99      0.85      0.89      9030
             weighted avg       0.99      0.99      0.99      9030


=== XGBoost ===
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [18:02:33] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Accuracy: 0.9978
F1 Score: 0.9978
Classification Report:
                           precision    recall  f1-score   support

              Bradycardia       1.00      1.00      1.00      3513
Low HRV (Arrhythmia Risk)       1.00      0.99      0.99        92
                   Normal       1.00      1.00      1.00      3011
              Tachycardia       1.00      1.00      1.00      2414

                 accuracy                           1.00      9030
                macro avg       1.00      1.00      1.00      9030
             weighted avg       1.00      1.00      1.00      9030


=== KNN ===
Accuracy: 0.9932
F1 Score: 0.9932
Classification Report:
                           precision    recall  f1-score   support

              Bradycardia       1.00      0.99      0.99      3513
Low HRV (Arrhythmia Risk)       0.96      0.86      0.91        92
                   Normal       0.99      0.99      0.99      3011
              Tachycardia       1.00      1.00      1.00      2414

                 accuracy                           0.99      9030
                macro avg       0.99      0.96      0.97      9030
             weighted avg       0.99      0.99      0.99      9030
```

```
# Step 8: Training and Evaluation
for name, model in models.items():
    print(f"\n=== {name} ===")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"Accuracy: {acc:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred, target_names=le.classes_))
```

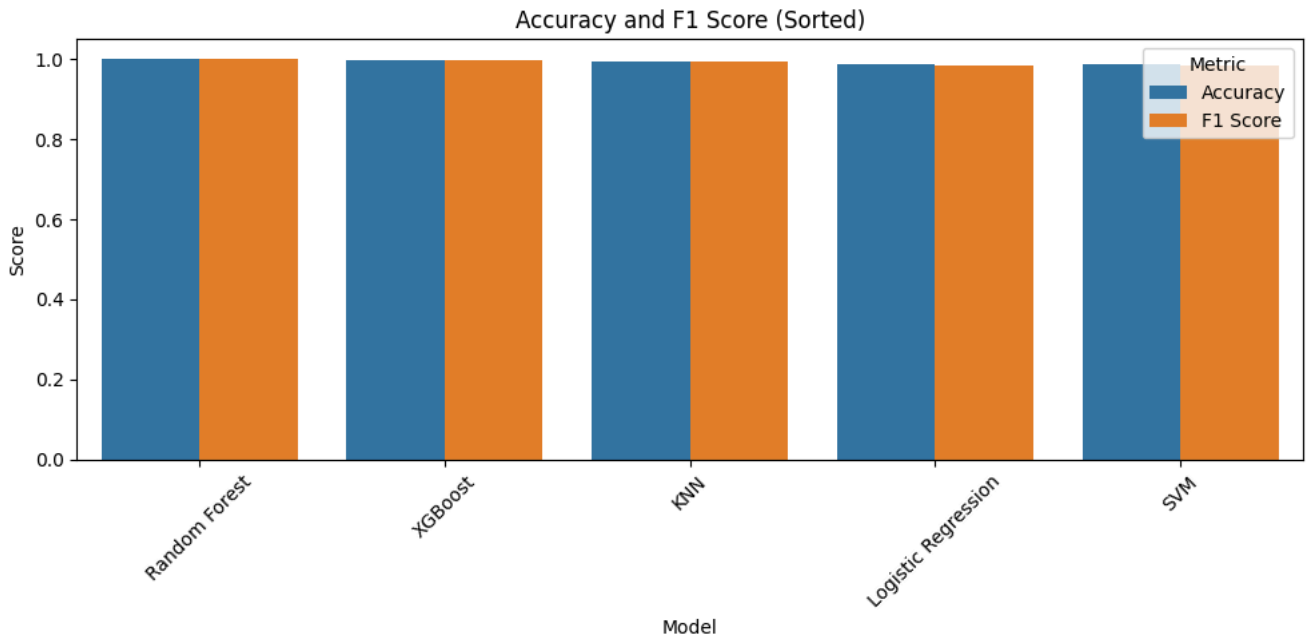
```
=== XGBoost ===
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [18:12:43] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Accuracy: 0.9978
F1 Score: 0.9978
Classification Report:
                           precision    recall  f1-score   support

              Bradycardia       1.00      1.00      1.00      3513
Low HRV (Arrhythmia Risk)       1.00      0.99      0.99        92
                   Normal       1.00      1.00      1.00      3011
              Tachycardia       1.00      1.00      1.00      2414

                 accuracy                           1.00      9030
                macro avg       1.00      1.00      1.00      9030
             weighted avg       1.00      1.00      1.00      9030


=== KNN ===
Accuracy: 0.9932
F1 Score: 0.9932
Classification Report:
                           precision    recall  f1-score   support

              Bradycardia       1.00      0.99      0.99      3513
Low HRV (Arrhythmia Risk)       0.96      0.86      0.91        92
                   Normal       0.99      0.99      0.99      3011
              Tachycardia       1.00      1.00      1.00      2414

                 accuracy                           0.99      9030
                macro avg       0.99      0.96      0.97      9030
             weighted avg       0.99      0.99      0.99      9030
```

```
# Train and evaluate models
from sklearn.metrics import confusion_matrix
results = []
conf_matrices = {}
precision_list = []
recall_list = []

# Before the loop, scale X_train and X_test:
# Step 6: Normalize features
scaler = StandardScaler() # Assuming StandardScaler has been imported
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    report = classification_report(y_test, y_pred, output_dict=True)

    results.append({'Model': name, 'Accuracy': acc, 'F1 Score': f1})
    precision_list.append({'Model': name, 'Precision': report['weighted avg']['precision']})
    recall_list.append({'Model': name, 'Recall': report['weighted avg']['recall']})
    conf_matrices[name] = confusion_matrix(y_test, y_pred) # Make sure 'confusion_matrix' is imported

results_df = pd.DataFrame(results)
precision_df = pd.DataFrame(precision_list)
recall_df = pd.DataFrame(recall_list)
```

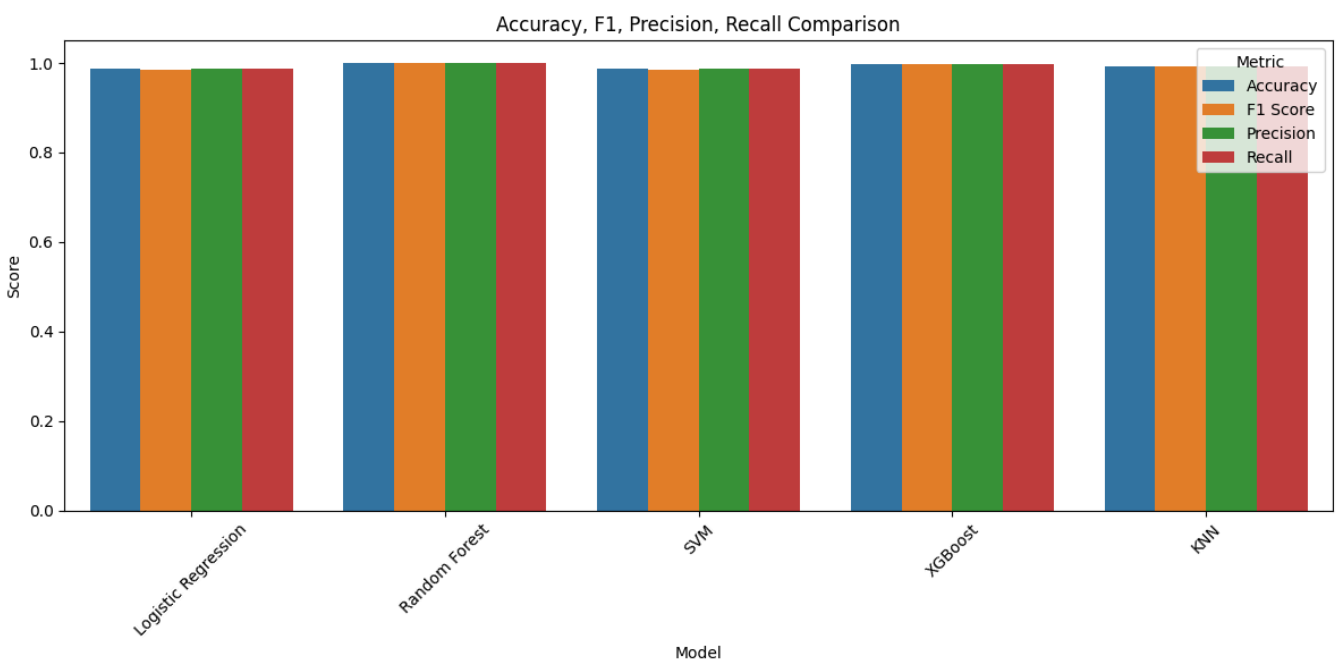
```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [18:15:56] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
# === Plot Accuracy and F1 Score ===
melted = results_df.melt(id_vars='Model', var_name='Metric', value_name='Score')
melted = melted.sort_values(by='Score', ascending=False)

plt.figure(figsize=(10, 5))
sns.barplot(data=melted, x='Model', y='Score', hue='Metric')
plt.title("Accuracy and F1 Score (Sorted)")
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()
```

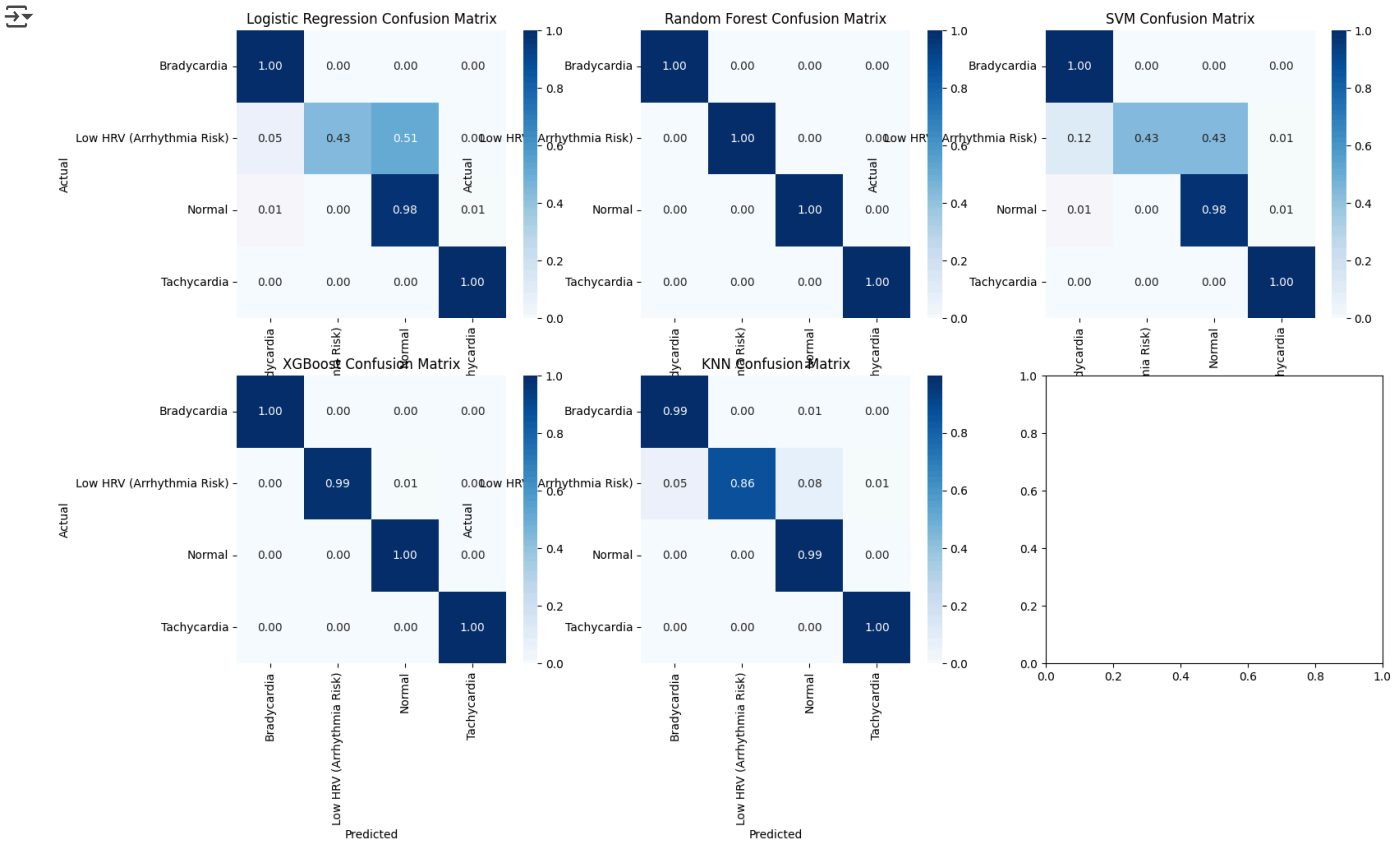



```
# === Plot Precision and Recall along with others ===
combined_metrics = results_df.merge(precision_df, on='Model').merge(recall_df, on='Model')
metrics_melted = combined_metrics.melt(id_vars='Model', var_name='Metric', value_name='Score')

plt.figure(figsize=(12, 6))
sns.barplot(data=metrics_melted, x='Model', y='Score', hue='Metric')
plt.title("Accuracy, F1, Precision, Recall Comparison")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
# === Plot Confusion Matrices ===
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.flatten()
class_names = le.classes_

for i, (model_name, cm) in enumerate(conf_matrices.items()):
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    sns.heatmap(cm_normalized, annot=True, fmt='.2f', cmap='Blues',
```

```
                xticklabels=class_names, yticklabels=class_names, ax=axes[i])
    axes[i].set_title(f"{model_name} Confusion Matrix")
    axes[i].set_xlabel("Predicted")
    axes[i].set_ylabel("Actual")
```




```
# Hide unused subplots if any
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()

# === Feature Importances ===
feature_names = X.columns

# Random Forest
rf_model = models["Random Forest"]
rf_importances = rf_model.feature_importances_
rf_df = pd.DataFrame({'Feature': feature_names, 'Importance': rf_importances})
rf_df = rf_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=rf_df.head(15), y='Feature', x='Importance')
plt.title("Top 15 Feature Importances - Random Forest")
plt.tight_layout()
plt.show()

# XGBoost
xgb_model = models["XGBoost"]
xgb_importances = xgb_model.feature_importances_
xgb_df = pd.DataFrame({'Feature': feature_names, 'Importance': xgb_importances})
xgb_df = xgb_df.sort_values(by='Importance', ascending=False)
```

```
plt.figure(figsize=(10, 6))
sns.barplot(data=xgb_df.head(15), y='Feature', x='Importance')
plt.title("Top 15 Feature Importances - XGBoost")
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```