Importing Libraries

In [1]:

```
# Data manipulation and visualization
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Scikit-learn for machine learning models and preprocessing
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score, brier_score_loss

# TensorFlow/Keras for deep learning models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# For custom base classifier (if needed)
from sklearn.base import BaseEstimator, ClassifierMixin

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

Fetching Dataset

About the Dataset:

The "bank-loan" dataset is focused on credit scoring, crucial for financial decision-making in loan approvals. It includes data on 700 bank customers who received loans and details their repayment outcomes, with a binary "Default" status indicating repayment success (0 for good, 1 for default). The dataset aims to support the development of a predictive credit scoring model that assists banks in assessing loan risks. Key features include:

- **Age**: Customer age in years.
- **Ed**: Education level (1 = No high school, 2 = High school, 3 = Some college, 4 = College degree, 5 = Postgrad).
- **Employ**: Years with the current employer.
- **Address**: Years at current address.
- **Income**: Household income in thousands.
- **Debtinc**: Debt-to-income ratio (x100).
- **Creddebt**: Credit card debt in thousands.
- **Othdebt**: Other debt in thousands.
- **Default**: Target variable, where 1 denotes default status and 0 denotes a good repayment history.

This dataset will be used to build a model to help banks optimize risk management and lending decisions.

In [ ]:
```
# Load the dataset
df = pd.read_csv('Bankloan.csv')
```

Displaying Dataset

In [3]:
```
# Display the first few rows to understand the structure of the data befor
print("Initial Data:")
print(df.head())
```

```
Initial Data:
    age  employ  address  income  debtinc   creddebt   othdebt   ed  defaul
t
0  41.0      17       12   176.0      9.3  11.359392  5.008608  3.0
1
1  27.0      10        6    31.0     17.3   1.362202  4.000798  1.0
0
2  40.0      15        7     NaN      5.5   0.856075  2.168925  1.0
0
3  41.0      15       14   120.0      2.9   2.658720  0.821280  NaN
0
4  24.0       2        0    28.0     17.3   1.787436  3.056564  2.0
1
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       680 non-null    float64
 1   employ    700 non-null    int64  
 2   address   700 non-null    int64  
 3   income    663 non-null    float64
 4   debtinc   700 non-null    float64
 5   creddebt  700 non-null    float64
 6   othdebt   700 non-null    float64
 7   ed        680 non-null    float64
 8   default   700 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 49.3 KB
```

In [5]:

```
df.describe()
```

Out[5]:

	age	employ	address	income	debtinc	creddebt	othde
count	680.000000	700.000000	700.000000	663.00000	700.000000	700.000000	700.0000
mean	34.750000	8.388571	8.268571	45.74359	10.260571	1.553553	3.0582
std	7.973215	6.658039	6.821609	37.44108	6.827234	2.117197	3.2875
min	20.000000	0.000000	0.000000	14.00000	0.400000	0.011696	0.0455
25%	28.000000	3.000000	3.000000	24.00000	5.000000	0.369059	1.0441
50%	34.000000	7.000000	7.000000	34.00000	8.600000	0.854869	1.9875
75%	40.000000	12.000000	12.000000	54.50000	14.125000	1.901955	3.9230
max	56.000000	31.000000	34.000000	446.00000	41.300000	20.561310	27.0336

In [6]:

```
# Check for missing values in the dataset
print("\nMissing Values in Each Column:")
print(df.isnull().sum())
```

```
Missing Values in Each Column:
age         20
employ       0
address      0
income      37
debtinc      0
creddebt     0
othdebt      0
ed          20
default      0
dtype: int64
```

Preprocessing

Handling Missing Values

To ensure the dataset is complete and ready for analysis, we'll address any missing values by following these steps:

1. **Numerical Columns**: Missing values in numerical columns will be filled with the median. This approach helps to mitigate the influence of outliers on imputed values.
2. **Categorical Columns**: Missing values in categorical columns will be filled with the mode (most frequent value), maintaining the most common category for better consistency.

In [7]:

```
# Handle missing values:
# 1. For numerical columns, we'll fill missing values with the median
# 2. For categorical columns, we'll fill missing values with the mode (mos

# Identifying numerical columns
num_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Identifying categorical columns
# Explicitly defining categorical columns based on the data understanding
cat_cols = ['ed']  # 'ed' is categorical, as it is encoded from 1 to 5

# Check the columns identified as categorical
print("\nCategorical Columns:", cat_cols)

# Create an imputer for numerical columns to fill missing values with medi
num_imputer = SimpleImputer(strategy='median')
df[num_cols] = num_imputer.fit_transform(df[num_cols])

# Create an imputer for categorical columns to fill missing values with mo
cat_imputer = SimpleImputer(strategy='most_frequent')
df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])
```

```
Categorical Columns: ['ed']
```

In [8]:

```
# Verify if any missing values remain
print("\nMissing Values After Imputation:")
print(df.isnull().sum())
```

```
Missing Values After Imputation:
age         0
employ      0
address     0
income      0
debtinc     0
creddebt    0
othdebt     0
ed          0
default     0
dtype: int64
```

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       700 non-null    float64
 1   employ    700 non-null    float64
 2   address   700 non-null    float64
 3   income    700 non-null    float64
 4   debtinc   700 non-null    float64
 5   creddebt  700 non-null    float64
 6   othdebt   700 non-null    float64
 7   ed        700 non-null    float64
 8   default   700 non-null    float64
dtypes: float64(9)
memory usage: 49.3 KB
```

Encoding Categorical Variables

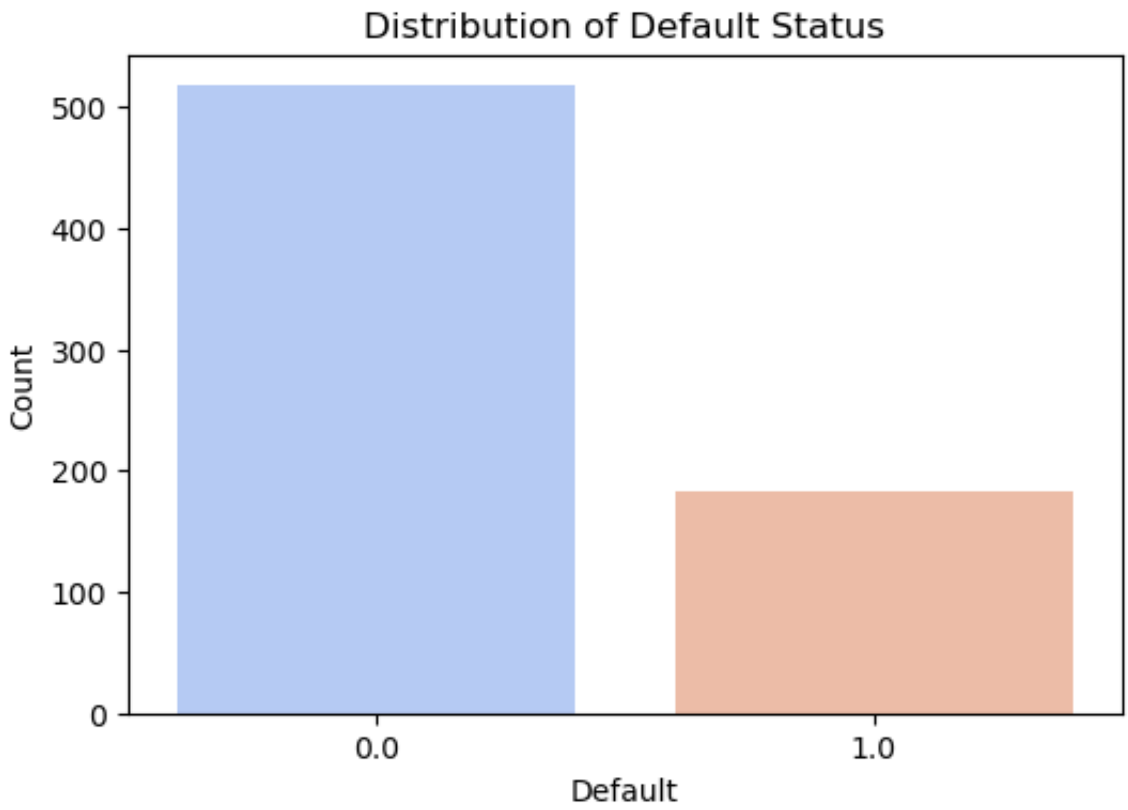
To prepare categorical variables for modeling, we need to encode them into a numerical format:

- The **'Ed'** column, representing education levels, is a categorical variable with values ranging from 1 to 5. We'll use `LabelEncoder` to convert these categories into numerical form.

In [10]:

```
# Encode the categorical variables if necessary
# 'Ed' is a categorical feature, so we can use LabelEncoder to convert it .
label_encoder = LabelEncoder()
df['ed'] = label_encoder.fit_transform(df['ed'])

# Display the distribution of the target variable
plt.figure(figsize=(6, 4))
sns.countplot(x='default', data=df, palette='coolwarm')
plt.title('Distribution of Default Status')
plt.xlabel('Default')
plt.ylabel('Count')
plt.show()
```

```
/var/folders/xx/d4y5bsbx2210fv9gdbdqc8s80000gn/T/ipykernel_11227/721629432.
py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='default', data=df, palette='coolwarm')
```

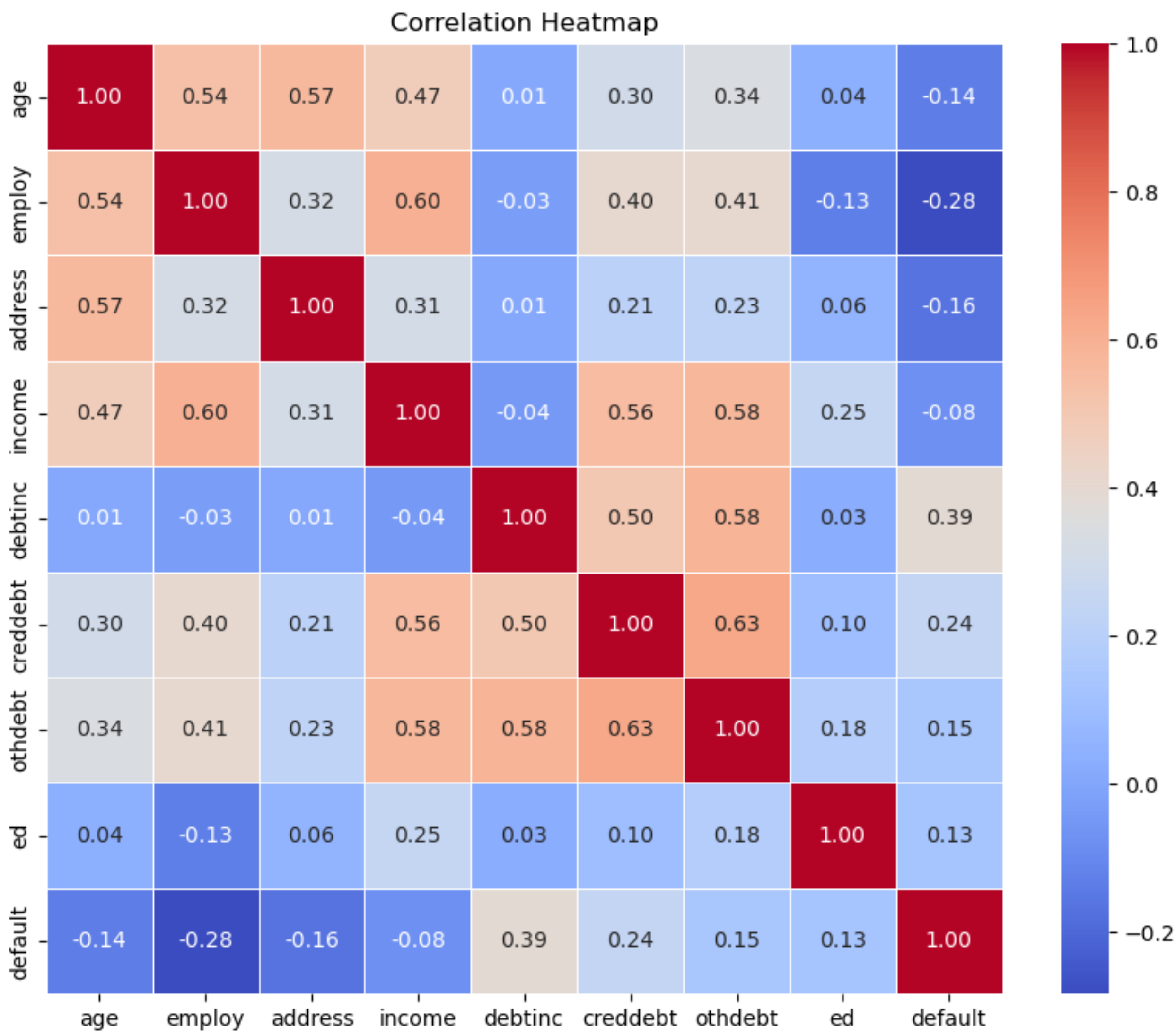


Visualizing the Correlation Matrix

To understand the relationships between variables in the dataset, we will plot a correlation heatmap. This visualization will help identify the strength and direction of correlations between numerical features, which can be useful for feature selection and understanding feature interactions.

In [11]:

```
# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=(
plt.title('Correlation Heatmap')
plt.show()
```

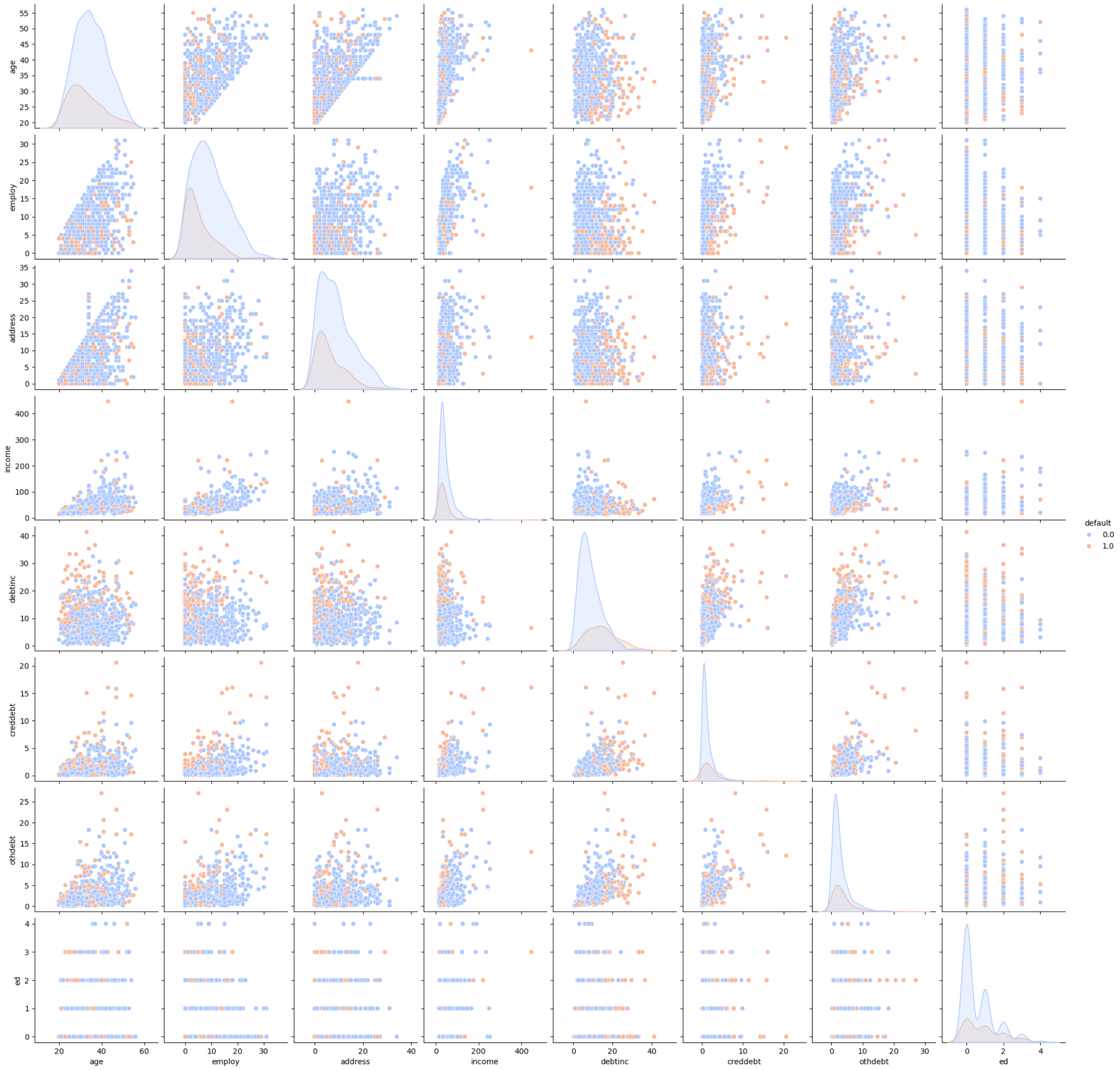


Visualizing Pairwise Relationships

To further explore the relationships between features and understand patterns in the data, we will create a pairplot. This plot displays pairwise relationships between numerical features, color-coded by the target variable **'Default'**. It helps in identifying potential clusters, separations, and distributions that could be relevant for predictive modeling.

In [12]:

```
# Visualize pairwise relationships for the features
sns.pairplot(df, hue='default', diag_kind='kde', palette='coolwarm')
plt.show()
```



Data Splitting and Preprocessing

To prepare the dataset for modeling, we perform the following steps:

1. **Splitting Features and Target**: The dataset is divided into features (X) and the target variable (y), where 'default' serves as the target.

2. **Training and Testing Sets**: We split the data into training and testing sets, with 70% of the data for training and 30% for testing, ensuring an unbiased model evaluation.

3. **Standardization**: The features are standardized using `StandardScaler` to transform them into a common scale. This step helps in improving model performance and convergence, especially for algorithms sensitive to feature scaling.

4. **Summary of Preprocessing**: After scaling, the data is ready for modeling, ensuring a consistent and optimized input for machine learning algorithms.

Now that preprocessing is complete, we can proceed to model development.

In [13]:

```
# Splitting the dataset into features (X) and target (y)
X = df.drop(columns=['default'])
y = df['default']

# Splitting into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Display the scaled data
print("\nScaled Data (First 5 rows of X_train_scaled):")
print(X_train_scaled[:5])

# Now, the data is preprocessed, and we can move on to model development
# For now, we'll just show the summary of the preprocessing
print("\nPreprocessing Complete. Data is ready for modeling.")
```

```
Scaled Data (First 5 rows of X_train_scaled):
[[-1.32755167 -1.2442356  -0.92295568 -0.80935606  0.84829132 -0.60284836
  -0.28727213  0.2724169 ]
 [-0.70231107 -0.49043312 -0.19773393 -0.46656363  1.22900191  0.27873052
   0.02017597  0.2724169 ]
 [-0.20211859  0.11260886 -0.05268958 -0.36108904 -0.70383646 -0.51729771
  -0.5383796  -0.77042903]
 [ 1.42350696 -1.09347511  0.96262086 -0.65114417  0.65793603 -0.08715123
  -0.39697112 -0.77042903]
 [-0.82735919 -0.18891213 -0.92295568 -0.30835174 -0.48419573 -0.58419453
  -0.30941012  0.2724169 ]]

Preprocessing Complete. Data is ready for modeling.
```

Splitting the dataset - train and test

In [14]:

```
# Load and preprocess your dataset
X_features = df.drop('default', axis=1)  # Features (predictors)
y_target = df['default']  # Target variable (default)

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_features)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_target, tes
y_train = y_train.to_numpy()
```

Models Considered for Comparison :

1) Random Forest

2) SVM

3) Decision tree

4) LSTM

Random Forest Classifier

Model Definition and Hyperparameter Tuning

In [15]:

```
# Define the model and parameter grid for hyperparameter tuning
rf_model = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
}

# Hyperparameter tuning with GridSearchCV
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy',
grid_search.fit(X_train, y_train)
best_rf_model = grid_search.best_estimator_
print(f"Best parameters: {grid_search.best_params_}")
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimator
s': 50}
```

KFold Cross-Validation

In [16]:

```
# KFold Cross-validation setup
cv_strategy = KFold(n_splits=10, shuffle=True, random_state=42)

# Prepare to store results in a DataFrame
metrics_per_fold = []

# Perform 10-Fold Cross-validation
for fold_number, (train_idx, val_idx) in enumerate(cv_strategy.split(X_tra
    X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
    y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

    # Fit model on the fold
    best_rf_model.fit(X_train_fold, y_train_fold)
    y_pred = best_rf_model.predict(X_val_fold)
    y_proba = best_rf_model.predict_proba(X_val_fold)[:, 1]

    # Initialize counts
    tp = tn = fp = fn = 0
```

```
    # Calculate manually
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1  # True Positive
        elif true == 0 and pred == 0:
            tn += 1  # True Negative
        elif true == 0 and pred == 1:
            fp += 1  # False Positive
        elif true == 1 and pred == 0:
            fn += 1  # False Negative

    # Calculate metrics
    true_positive_rate = tp / (tp + fn) if (tp + fn) > 0 else 0
    true_negative_rate = tn / (tn + fp) if (tn + fp) > 0 else 0
    false_positive_rate = fp / (fp + tn) if (fp + tn) > 0 else 0
    false_negative_rate = fn / (fn + tp) if (fn + tp) > 0 else 0
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    f1_score = 2 * precision * true_positive_rate / (precision + true_posi
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    error_rate = 1 - accuracy
    balanced_accuracy = (true_positive_rate + true_negative_rate) / 2
    true_skill_statistic = true_positive_rate + true_negative_rate - 1
    heidke_skill_score = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) +
    brier_score = brier_score_loss(y_val_fold, y_proba)
    auc_score = roc_auc_score(y_val_fold, y_proba)

    # Append metrics for each fold
    metrics_per_fold.append([fold_number, tp, tn, fp, fn, true_positive_rat
                             false_negative_rate, precision, f1_score, accu
                             true_skill_statistic, heidke_skill_score, brie

# Create DataFrame with fold metrics
metrics_rf = pd.DataFrame(metrics_per_fold, columns=[
    'Fold', 'TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR',
    'Precision', 'F1_measure', 'Accuracy', 'Error_rate', 'BACC', 'TSS', 'HS
    'Brier_score', 'AUC'
])

# Display results per fold and calculate average metrics across all folds
metrics_rf.loc['Average'] = metrics_rf.mean(numeric_only=True)
print(metrics_rf)
```

	Fold	TP	TN	FP	FN	TPR	TNR	FPR	FNR
0	1.0	3.0	32.0	2.0	19.0	0.136364	0.941176	0.058824	0.863636
1	2.0	4.0	42.0	4.0	6.0	0.400000	0.913043	0.086957	0.600000
2	3.0	4.0	41.0	4.0	7.0	0.363636	0.911111	0.088889	0.636364
3	4.0	12.0	27.0	6.0	11.0	0.521739	0.818182	0.181818	0.478261
4	5.0	6.0	37.0	7.0	6.0	0.500000	0.840909	0.159091	0.500000
5	6.0	6.0	35.0	6.0	9.0	0.400000	0.853659	0.146341	0.600000
6	7.0	8.0	38.0	4.0	6.0	0.571429	0.904762	0.095238	0.428571
7	8.0	3.0	42.0	5.0	6.0	0.333333	0.893617	0.106383	0.666667
8	9.0	9.0	38.0	1.0	8.0	0.529412	0.974359	0.025641	0.470588
9	10.0	6.0	41.0	3.0	6.0	0.500000	0.931818	0.068182	0.500000
Average	5.5	6.1	37.3	4.2	8.4	0.425591	0.898264	0.101736	0.574409

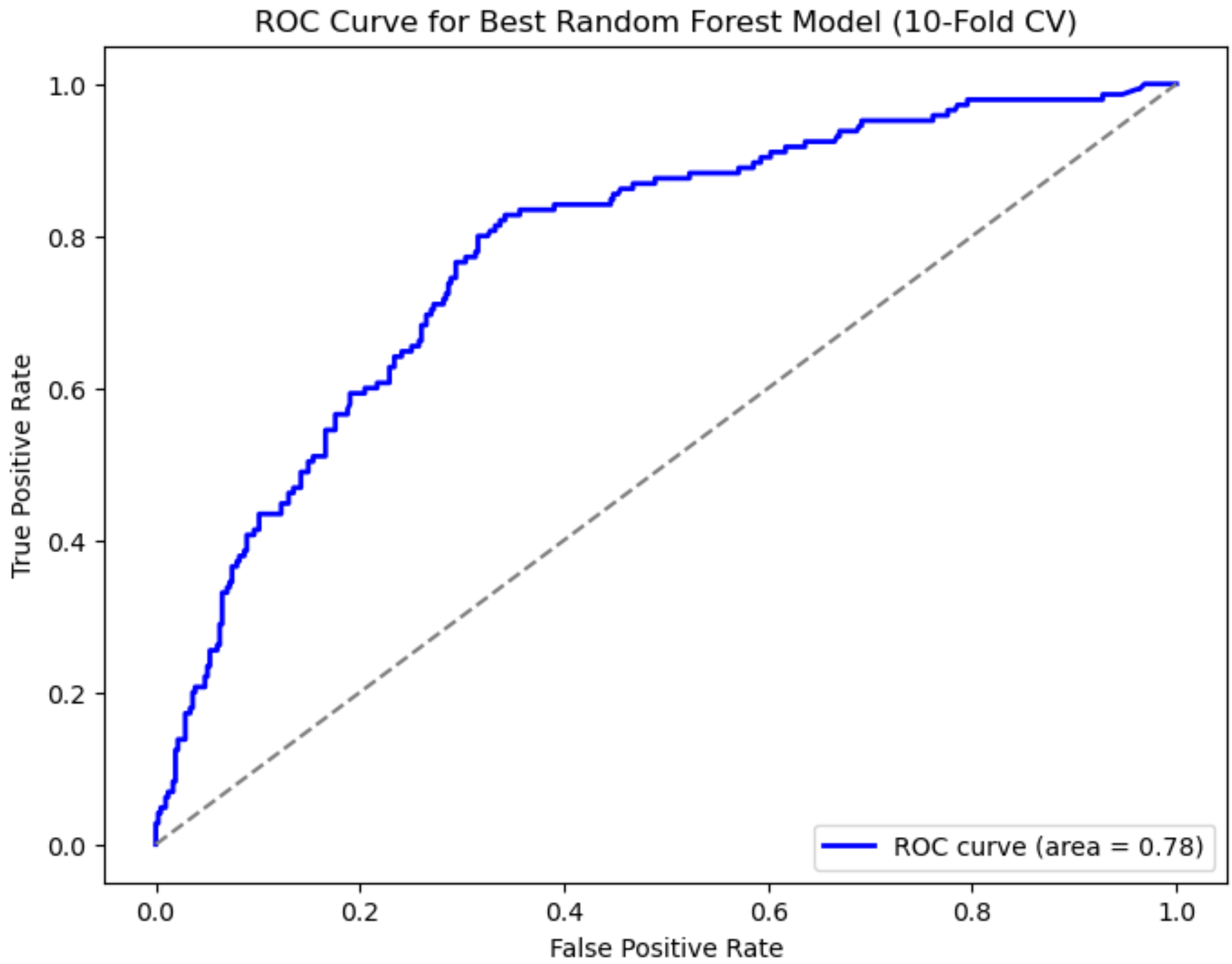
	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS
0	0.600000	0.222222	0.625000	0.375000	0.538770	0.077540
1	0.500000	0.444444	0.821429	0.178571	0.656522	0.313043
2	0.500000	0.421053	0.803571	0.196429	0.637374	0.274747
3	0.666667	0.585366	0.696429	0.303571	0.669960	0.339921
4	0.461538	0.480000	0.767857	0.232143	0.670455	0.340909
5	0.500000	0.444444	0.732143	0.267857	0.626829	0.253659
6	0.666667	0.615385	0.821429	0.178571	0.738095	0.476190
7	0.375000	0.352941	0.803571	0.196429	0.613475	0.226950
8	0.900000	0.666667	0.839286	0.160714	0.751885	0.503771
9	0.666667	0.571429	0.839286	0.160714	0.715909	0.431818
Average	0.583654	0.480395	0.775000	0.225000	0.661927	0.323855

	HSS	Brier_score	AUC
0	0.089783	0.232632	0.794118
1	0.339623	0.142026	0.671739
2	0.306306	0.127965	0.856566
3	0.351499	0.181543	0.820817
4	0.330882	0.144092	0.801136
5	0.270833	0.165738	0.778862
6	0.500000	0.150885	0.801020
7	0.237624	0.165757	0.567376
8	0.569966	0.121881	0.873303
9	0.475000	0.117996	0.873106
Average	0.347152	0.155052	0.783804

ROC Curve and AUC Score for Model Evaluation

In [17]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Aggregate true labels and predicted probabilities across all folds
y_true_all = []
y_proba_all = []

for train_idx, val_idx in cv_strategy.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
    y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

    # Fit the model and predict probabilities
    best_rf_model.fit(X_train_fold, y_train_fold)
    y_proba = best_rf_model.predict_proba(X_val_fold)[:, 1]

    # Append the results to the lists
    y_true_all.extend(y_val_fold)
    y_proba_all.extend(y_proba)

# Convert lists to numpy arrays for further calculations
y_true_all = np.array(y_true_all)
y_proba_all = np.array(y_proba_all)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_true_all, y_proba_all)

# Calculate the ROC AUC score
roc_auc = roc_auc_score(y_true_all, y_proba_all)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Best Random Forest Model (10-Fold CV)')
plt.legend(loc='lower right')
plt.show()

print(f"Average ROC AUC Score across folds: {roc_auc:.2f}")
```




```
Average ROC AUC Score across folds: 0.78
```

SVM

Hyperparameter Tuning

In [18]:

```
# Define the model and parameter grid for hyperparameter tuning
svm = SVC(probability=True, random_state=42)
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

# Hyperparameter tuning
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jol
grid_search.fit(X_train, y_train)
best_svm = grid_search.best_estimator_
print("Best parameters: ", grid_search.best_params_)
```

```
Best parameters:  {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
```

Model Creation, Kfold cross validation and Calculate metrics

In [19]:

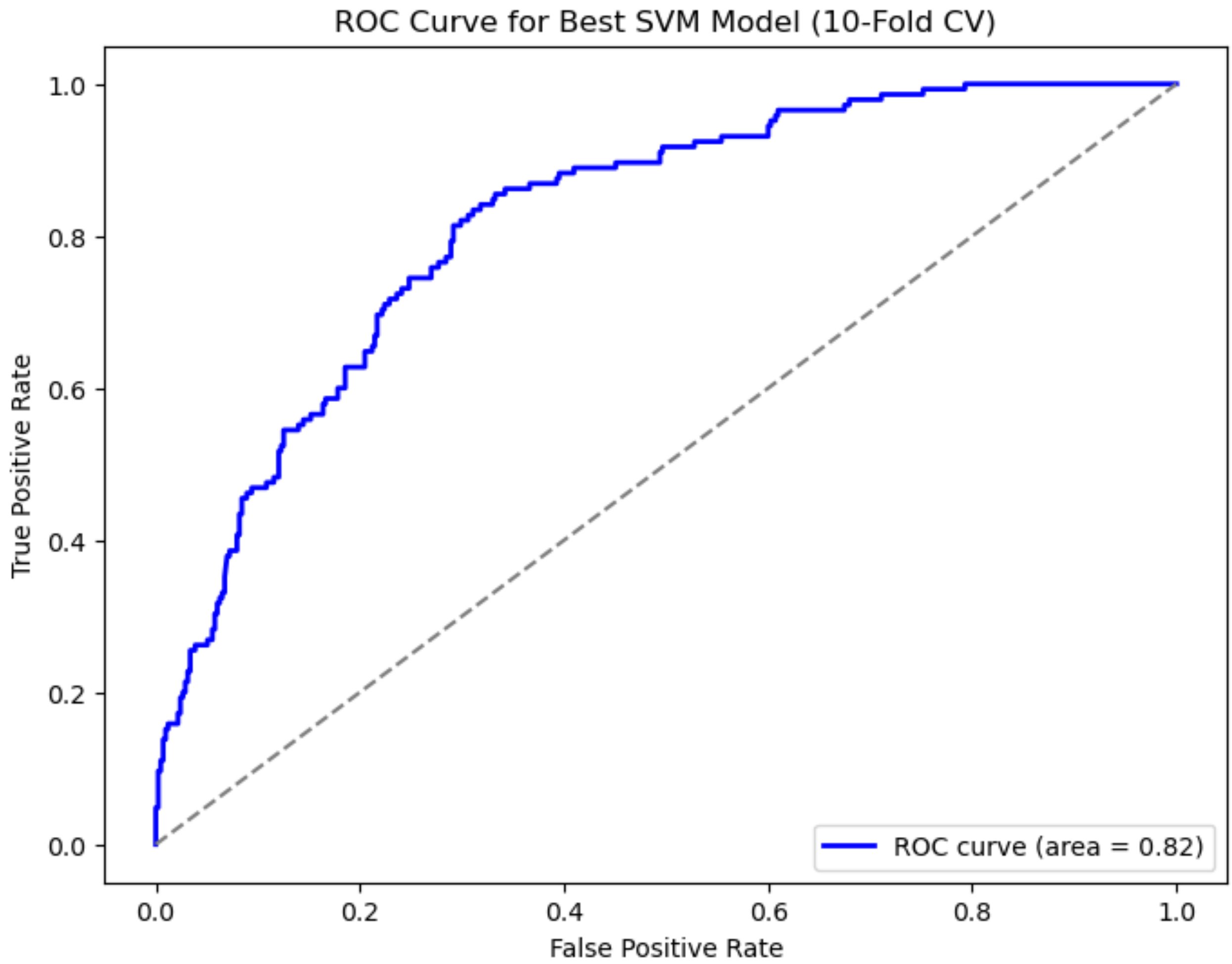
```python
# KFold Cross-validation setup
cv = KFold(n_splits=10, shuffle=True, random_state=42)

# Prepare to store results in a DataFrame
metrics_list = []

# Perform 10-Fold Cross-validation
for fold, (train_idx, val_idx) in enumerate(cv.split(X_train, y_train), 1)
    X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
    y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

    # Fit model on the fold
    best_svm.fit(X_train_fold, y_train_fold)
    y_pred = best_svm.predict(X_val_fold)
    y_proba = best_svm.predict_proba(X_val_fold)[:, 1]

    # Initialize counts
    tp = tn = fp = fn = 0

    # Calculate manually
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1  # True Positive
        elif true == 0 and pred == 0:
            tn += 1  # True Negative
        elif true == 0 and pred == 1:
            fp += 1  # False Positive
        elif true == 1 and pred == 0:
            fn += 1  # False Negative

    # Calculate metrics
    TPR = tp / (tp + fn) if (tp + fn) > 0 else 0
    TNR = tn / (tn + fp) if (tn + fp) > 0 else 0
    FPR = fp / (fp + tn) if (fp + tn) > 0 else 0
    FNR = fn / (fn + tp) if (fn + tp) > 0 else 0
    Precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    F1 = 2 * Precision * TPR / (Precision + TPR) if (Precision + TPR) > 0 e
    Accuracy = (tp + tn) / (tp + tn + fp + fn)
    Error_rate = 1 - Accuracy
    BACC = (TPR + TNR) / 2
    TSS = TPR + TNR - 1
    HSS = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (t
    Brier_score = brier_score_loss(y_val_fold, y_proba)
    AUC = roc_auc_score(y_val_fold, y_proba)

    # Append metrics for each fold
    metrics_list.append([fold, tp, tn, fp, fn, TPR, TNR, FPR, FNR, Precisio

# Create DataFrame with fold metrics
metrics_svm = pd.DataFrame(metrics_list, columns=[
    'Fold', 'TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR',
    'Precision', 'F1_measure', 'Accuracy', 'Error_rate', 'BACC',
    'TSS', 'HSS', 'Brier_score', 'AUC'
])

# Display results per fold and calculate average metrics across all folds
metrics_svm.loc['Average'] = metrics_svm.mean(numeric_only=True)
print(metrics_svm)
```

```
         Fold   TP    TN   FP    FN       TPR       TNR       FPR       FNR  \
0         1.0  2.0  31.0  3.0  20.0  0.090909  0.911765  0.088235  0.909091
1         2.0  5.0  45.0  1.0   5.0  0.500000  0.978261  0.021739  0.500000
2         3.0  5.0  43.0  2.0   6.0  0.454545  0.955556  0.044444  0.545455
3         4.0  9.0  30.0  3.0  14.0  0.391304  0.909091  0.090909  0.608696
4         5.0  4.0  39.0  5.0   8.0  0.333333  0.886364  0.113636  0.666667
5         6.0  5.0  37.0  4.0  10.0  0.333333  0.902439  0.097561  0.666667
6         7.0  3.0  40.0  2.0  11.0  0.214286  0.952381  0.047619  0.785714
7         8.0  3.0  43.0  4.0   6.0  0.333333  0.914894  0.085106  0.666667
8         9.0  6.0  39.0  0.0  11.0  0.352941  1.000000  0.000000  0.647059
9        10.0  3.0  41.0  3.0   9.0  0.250000  0.931818  0.068182  0.750000
Average   5.5  4.5  38.8  2.7  10.0  0.325399  0.934257  0.065743  0.674601

         Precision  F1_measure  Accuracy  Error_rate      BACC       TSS  \
0         0.400000    0.148148  0.589286    0.410714  0.501337  0.002674
1         0.833333    0.625000  0.892857    0.107143  0.739130  0.478261
2         0.714286    0.555556  0.857143    0.142857  0.705051  0.410101
3         0.750000    0.514286  0.696429    0.303571  0.650198  0.300395
4         0.444444    0.380952  0.767857    0.232143  0.609848  0.219697
5         0.555556    0.416667  0.750000    0.250000  0.617886  0.235772
6         0.600000    0.315789  0.767857    0.232143  0.583333  0.166667
7         0.428571    0.375000  0.821429    0.178571  0.624113  0.248227
8         1.000000    0.521739  0.803571    0.196429  0.676471  0.352941
9         0.500000    0.333333  0.785714    0.214286  0.590909  0.181818
Average   0.622619    0.418647  0.773214    0.226786  0.629828  0.259655

              HSS  Brier_score       AUC
0        0.003096     0.216868  0.799465
1        0.567010     0.105880  0.789130
2        0.475410     0.114426  0.858586
3        0.323864     0.175672  0.872200
4        0.241667     0.158262  0.744318
5        0.270019     0.165512  0.786992
6        0.212121     0.132442  0.870748
7        0.272727     0.138199  0.640662
8        0.431734     0.119583  0.914027
9        0.222222     0.108218  0.903409
Average  0.301987     0.143506  0.817954
```

ROC Curve

In [20]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Aggregate true labels and predicted probabilities across all folds
y_true_all = []
y_proba_all = []

for train_idx, val_idx in cv.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
    y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

    # Fit the model and predict probabilities for each fold
    best_svm.fit(X_train_fold, y_train_fold)
    y_proba = best_svm.predict_proba(X_val_fold)[:, 1]

    # Append the results to the lists
    y_true_all.extend(y_val_fold)
    y_proba_all.extend(y_proba)

# Convert lists to numpy arrays for further calculations
y_true_all = np.array(y_true_all)
y_proba_all = np.array(y_proba_all)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_true_all, y_proba_all)

# Calculate the ROC AUC score
roc_auc = roc_auc_score(y_true_all, y_proba_all)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Best SVM Model (10-Fold CV)')
plt.legend(loc='lower right')
plt.show()

print(f"Average ROC AUC Score across folds: {roc_auc:.2f}")
```




```
Average ROC AUC Score across folds: 0.82
```

Decision Tree

Hyperparameter Tuning

In [21]:

```
# Define the model and parameter grid for hyperparameter tuning
dt = DecisionTreeClassifier(random_state=42)
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10]
}

# Hyperparameter tuning
grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='accuracy', n_jobs
grid_search.fit(X_train, y_train)
best_dt = grid_search.best_estimator_
print(f"Best Decision Tree Model: {best_dt}")
```

```
Best Decision Tree Model: DecisionTreeClassifier(min_samples_leaf=5, min_sa
mples_split=20,
                       random_state=42)
```

Model Creation, Kfold cross validation and metrics

In [22]:

```
# KFold Cross-validation setup
cv = KFold(n_splits=10, shuffle=True, random_state=42)

# Prepare to store results in a DataFrame
metrics_list = []

# Perform 10-Fold Cross-validation
for fold, (train_idx, val_idx) in enumerate(cv.split(X_train, y_train), 1)
    X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
    y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

    # Fit model on the fold
    best_dt.fit(X_train_fold, y_train_fold)
    y_pred = best_dt.predict(X_val_fold)
    y_proba = best_dt.predict_proba(X_val_fold)[:, 1]

    # Initialize counts
    tp = tn = fp = fn = 0

    # Loop through true and predicted labels
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1  # True Positive
        elif true == 0 and pred == 0:
            tn += 1  # True Negative
        elif true == 0 and pred == 1:
            fp += 1  # False Positive
        elif true == 1 and pred == 0:
            fn += 1  # False Negative

    # Calculate metrics
    TPR = tp / (tp + fn) if (tp + fn) > 0 else 0
    TNR = tn / (tn + fp) if (tn + fp) > 0 else 0
    FPR = fp / (fp + tn) if (fp + tn) > 0 else 0
    FNR = fn / (fn + tp) if (fn + tp) > 0 else 0
    Precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    F1 = 2 * Precision * TPR / (Precision + TPR) if (Precision + TPR) > 0 e
    Accuracy = (tp + tn) / (tp + tn + fp + fn)
    Error_rate = 1 - Accuracy
    BACC = (TPR + TNR) / 2
    TSS = TPR + TNR - 1
    HSS = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (t
    Brier_score = brier_score_loss(y_val_fold, y_proba)
    AUC = roc_auc_score(y_val_fold, y_proba)

    # Append metrics for each fold
    metrics_list.append([fold, tp, tn, fp, fn, TPR, TNR, FPR, FNR, Precisio

# Create DataFrame with fold metrics
metrics_dt = pd.DataFrame(metrics_list, columns=[
    'Fold', 'TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR',
    'Precision', 'F1_measure', 'Accuracy', 'Error_rate', 'BACC',
    'TSS', 'HSS', 'Brier_score', 'AUC'
])

# Display results per fold and calculate average metrics across all folds
metrics_dt.loc['Average'] = metrics_dt.mean(numeric_only=True)
print(metrics_dt)
```

	Fold	TP	TN	FP	FN	TPR	TNR	FPR	FNR
0	1.0	5.0	32.0	2.0	17.0	0.227273	0.941176	0.058824	0.772727
1	2.0	4.0	39.0	7.0	6.0	0.400000	0.847826	0.152174	0.600000
2	3.0	6.0	39.0	6.0	5.0	0.545455	0.866667	0.133333	0.454545
3	4.0	9.0	28.0	5.0	14.0	0.391304	0.848485	0.151515	0.608696
4	5.0	4.0	37.0	7.0	8.0	0.333333	0.840909	0.159091	0.666667
5	6.0	6.0	34.0	7.0	9.0	0.400000	0.829268	0.170732	0.600000
6	7.0	8.0	35.0	7.0	6.0	0.571429	0.833333	0.166667	0.428571
7	8.0	3.0	40.0	7.0	6.0	0.333333	0.851064	0.148936	0.666667
8	9.0	10.0	32.0	7.0	7.0	0.588235	0.820513	0.179487	0.411765
9	10.0	3.0	38.0	6.0	9.0	0.250000	0.863636	0.136364	0.750000
Average	5.5	5.8	35.4	6.1	8.7	0.404036	0.854288	0.145712	0.595964

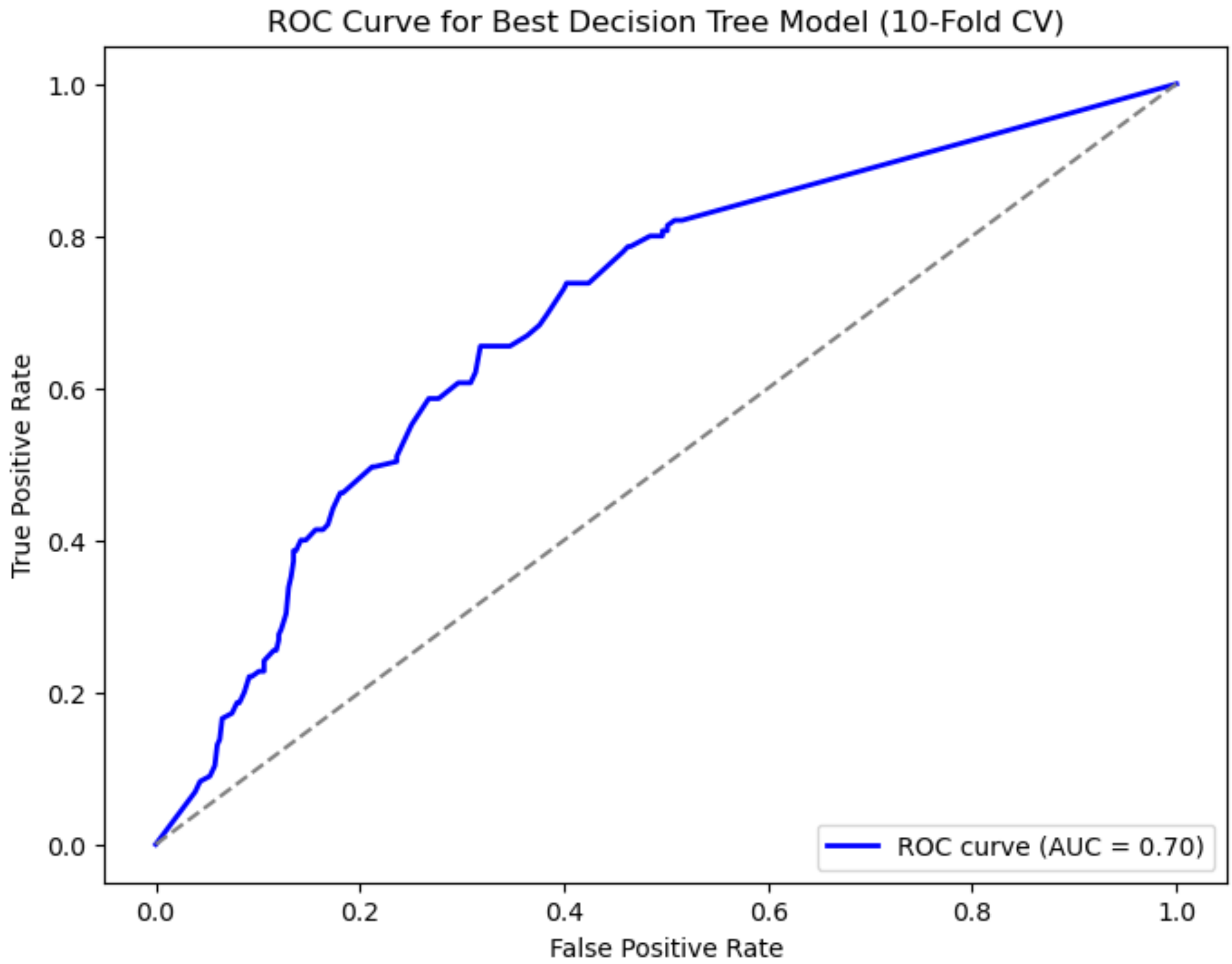
	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS
0	0.714286	0.344828	0.660714	0.339286	0.584225	0.168449
1	0.363636	0.380952	0.767857	0.232143	0.623913	0.247826
2	0.500000	0.521739	0.803571	0.196429	0.706061	0.412121
3	0.642857	0.486486	0.660714	0.339286	0.619895	0.239789
4	0.363636	0.347826	0.732143	0.267857	0.587121	0.174242
5	0.461538	0.428571	0.714286	0.285714	0.614634	0.229268
6	0.533333	0.551724	0.767857	0.232143	0.702381	0.404762
7	0.300000	0.315789	0.767857	0.232143	0.592199	0.184397
8	0.588235	0.588235	0.750000	0.250000	0.704374	0.408748
9	0.333333	0.285714	0.732143	0.267857	0.556818	0.113636
Average	0.480086	0.425187	0.735714	0.264286	0.629162	0.258324

	HSS	Brier_score	AUC
0	0.191489	0.247200	0.728610
1	0.238494	0.174039	0.665217
2	0.398438	0.164395	0.746465
3	0.254902	0.241597	0.752306
4	0.179688	0.227260	0.722538
5	0.239389	0.236043	0.673984
6	0.395349	0.217166	0.716837
7	0.176471	0.205775	0.596927
8	0.408748	0.204872	0.713424
9	0.125000	0.183755	0.725379
Average	0.260797	0.210210	0.704169

ROC Curve

In [23]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Aggregate true labels and predicted probabilities across all folds
y_true_all = []
y_proba_all = []

# Perform 10-Fold Cross-validation with the best Decision Tree model
for train_idx, val_idx in cv.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
    y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

    # Fit the model on the training fold
    best_dt.fit(X_train_fold, y_train_fold)
    y_proba = best_dt.predict_proba(X_val_fold)[:, 1]  # Get probability fo

    # Store the true labels and predicted probabilities for the validation
    y_true_all.extend(y_val_fold)
    y_proba_all.extend(y_proba)

# Convert lists to numpy arrays for calculations
y_true_all = np.array(y_true_all)
y_proba_all = np.array(y_proba_all)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_true_all, y_proba_all)

# Calculate the ROC AUC score
roc_auc = roc_auc_score(y_true_all, y_proba_all)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Best Decision Tree Model (10-Fold CV)')
plt.legend(loc='lower right')
plt.show()

print(f"Average ROC AUC Score across folds: {roc_auc:.2f}")
```




```
Average ROC AUC Score across folds: 0.70
```

LSTM

Model Creation, Hyperparameter tuning, kfold cross validation and metrics

In [24]:

```python
# Separate features and target
X = df.drop('default', axis=1).values
y = df['default'].values

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Reshape data for LSTM [samples, time steps, features]
X = X.reshape((X.shape[0], 1, X.shape[1]))

# Define custom Keras model wrapper
class KerasLSTMClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, learning_rate=0.001, dropout_rate=0.2, epochs=20, ba
        self.learning_rate = learning_rate
        self.dropout_rate = dropout_rate
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = None

    def fit(self, X, y):
```

```
        self.model = self.create_lstm_model()
        self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size
        return self

    def predict(self, X):
        return (self.model.predict(X) > 0.5).astype("int32").flatten()

    def create_lstm_model(self):
        model = Sequential()
        model.add(LSTM(units=50, activation='relu', input_shape=(X.shape[1
        model.add(Dropout(self.dropout_rate))
        model.add(Dense(1, activation='sigmoid'))
        optimizer = Adam(learning_rate=self.learning_rate)
        model.compile(optimizer=optimizer, loss='binary_crossentropy', metr
        return model

# Create the KerasLSTMClassifier
model = KerasLSTMClassifier()

# Define hyperparameters grid to tune
param_grid = {
    'learning_rate': [0.001, 0.01],
    'dropout_rate': [0.2, 0.3],
}

# Set up GridSearchCV with 10-fold cross-validation
cv = KFold(n_splits=10, shuffle=True, random_state=42)

grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, r

# Fit GridSearchCV
grid_result = grid_search.fit(X, y)

# Get the best hyperparameters
best_params = grid_result.best_params_
print(f"Best Hyperparameters: {best_params}")

# Best model with the best hyperparameters
best_model = grid_result.best_estimator_

# Evaluate the best model with 10-fold cross-validation
metrics_list = []

for fold, (train_idx, val_idx) in enumerate(cv.split(X, y), 1):
    X_train_fold, X_val_fold = X[train_idx], X[val_idx]
    y_train_fold, y_val_fold = y[train_idx], y[val_idx]

    # Train the model
    best_model.fit(X_train_fold, y_train_fold)

    # Predict on the validation fold
    y_pred = best_model.predict(X_val_fold)
    y_proba = best_model.model.predict(X_val_fold).flatten()

    # Initialize counts
    tp = tn = fp = fn = 0

    # Manually compute confusion matrix components
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1  # True Positive
```

```
        elif true == 0 and pred == 0:
            tn += 1  # True Negative
        elif true == 0 and pred == 1:
            fp += 1  # False Positive
        elif true == 1 and pred == 0:
            fn += 1  # False Negative

    # Calculate metrics
    TPR = tp / (tp + fn) if (tp + fn) > 0 else 0
    TNR = tn / (tn + fp) if (tn + fp) > 0 else 0
    FPR = fp / (fp + tn) if (fp + tn) > 0 else 0
    FNR = fn / (fn + tp) if (fn + tp) > 0 else 0
    Precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    F1 = 2 * Precision * TPR / (Precision + TPR) if (Precision + TPR) > 0 e
    Accuracy = (tp + tn) / (tp + tn + fp + fn)
    Error_rate = 1 - Accuracy
    BACC = (TPR + TNR) / 2
    TSS = TPR + TNR - 1
    HSS = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (
    Brier_score = brier_score_loss(y_val_fold, y_proba)
    AUC = roc_auc_score(y_val_fold, y_proba)

    # Append metrics for each fold
    metrics_list.append([fold, tp, tn, fp, fn, TPR, TNR, FPR, FNR, Precisio

# Create DataFrame with fold metrics
metrics_lstm = pd.DataFrame(metrics_list, columns=[
    'Fold', 'TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR',
    'Precision', 'F1_measure', 'Accuracy', 'Error_rate', 'BACC',
    'TSS', 'HSS', 'Brier_score', 'AUC'
])

# Display results for only the best hyperparameters
metrics_lstm.loc['Average'] = metrics_lstm.mean(numeric_only=True)
print(metrics_lstm)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/stepp
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/joblib/externa
ls/loky/process_executor.py:752: UserWarning: A worker stopped while some j
obs were given to the executor. This can be caused by a too short worker ti
meout or by a memory leak.
  warnings.warn(
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/stepp
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/stepp
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
```

```
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16b5c9a8
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16b5c9a8
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16eff9a8
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16eff9a8
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
```

```
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x17166f2e
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x17166f2e
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/stepp
```

```
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x171be32e
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x171be32e
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16f6fb10
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16f6fb10
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16f585a8
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16b2c9a8
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTraine
r.make_predict_function.<locals>.one_step_on_data_distributed at 0x16f585a8
0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflo
w.org/guide/function#controlling_retracing and https://www.tensorflow.org/a
pi_docs/python/tf/function for  more details.
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/stepp
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x16b2c9a80> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x17176da80> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x17176da80> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Best Hyperparameters: {'dropout_rate': 0.3, 'learning_rate': 0.01}
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 931us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/stepWARNING:tensorflow:5 out of the last 1
3 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_s
tep_on_data_distributed at 0x310140220> triggered tf.function retracing. Tr
acing is expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with differ
ent shapes, (3) passing Python objects instead of tensors. For (1), please 
define your @tf.function outside of the loop. For (2), @tf.function has red
uce_retracing=True option that can avoid unnecessary retracing. For (3), pl
ease refer to https://www.tensorflow.org/guide/function#controlling_retraci
ng and https://www.tensorflow.org/api_docs/python/tf/function for  more det
ails.
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 608us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/stepWARNING:tensorflow:5 out of the last 1
3 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_s
tep_on_data_distributed at 0x30b4f3c40> triggered tf.function retracing. Tr
acing is expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with differ
ent shapes, (3) passing Python objects instead of tensors. For (1), please 
define your @tf.function outside of the loop. For (2), @tf.function has red
uce_retracing=True option that can avoid unnecessary retracing. For (3), pl
ease refer to https://www.tensorflow.org/guide/function#controlling_retraci
ng and https://www.tensorflow.org/api_docs/python/tf/function for  more det
ails.
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 592us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 565us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 607us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 590us/step
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

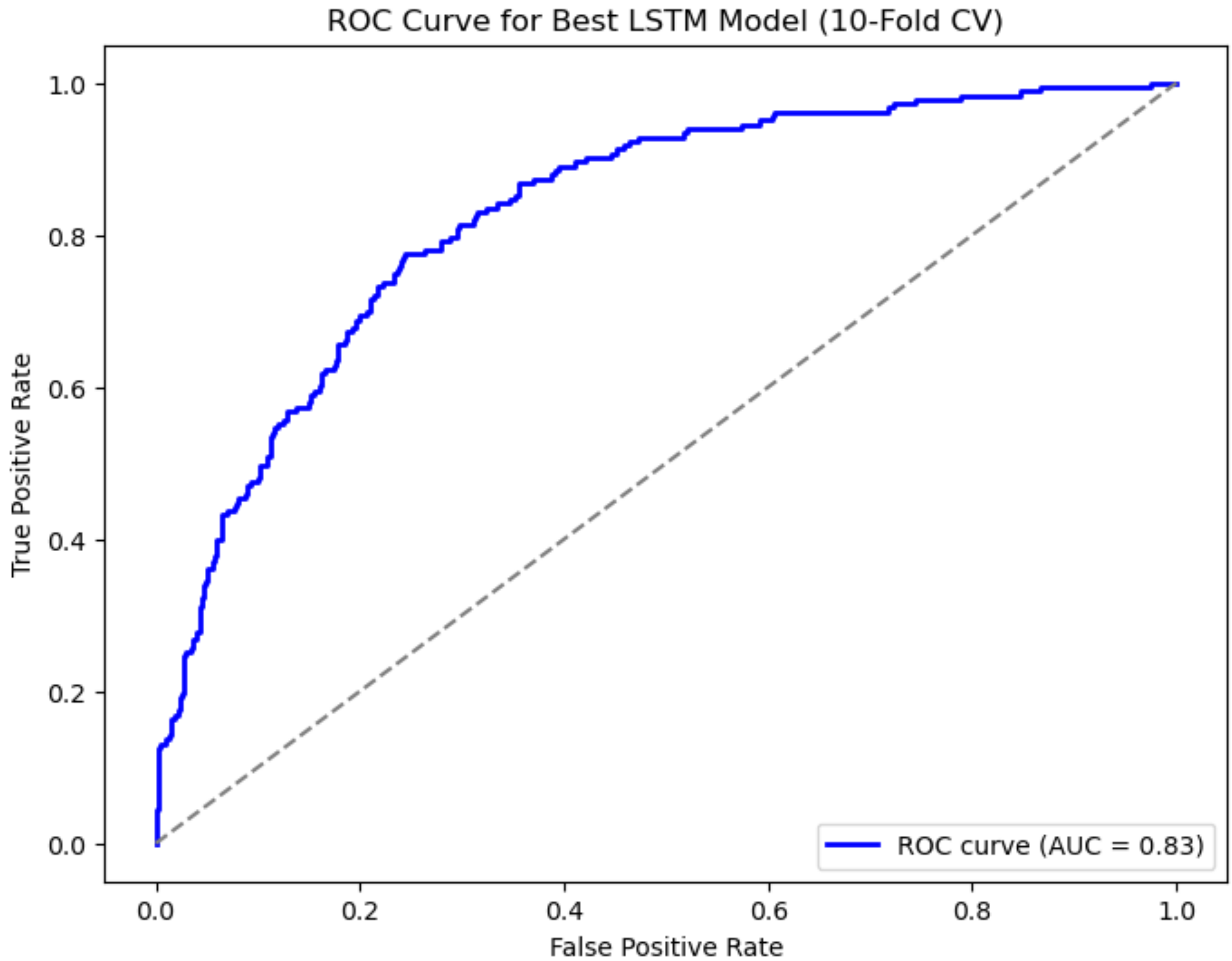
```
3/3                      0s 43ms/step
3/3                      0s 600us/step
         Fold    TP    TN   FP    FN       TPR       TNR       FPR       FN
R  \
0         1.0  12.0  48.0  4.0   6.0  0.666667  0.923077  0.076923  0.33333
3
1         2.0  10.0  49.0  1.0  10.0  0.500000  0.980000  0.020000  0.50000
0
2         3.0   6.0  54.0  5.0   5.0  0.545455  0.915254  0.084746  0.45454
5
3         4.0   8.0  42.0  5.0  15.0  0.347826  0.893617  0.106383  0.65217
4
4         5.0   8.0  43.0  6.0  13.0  0.380952  0.877551  0.122449  0.61904
8
5         6.0   7.0  49.0  3.0  11.0  0.388889  0.942308  0.057692  0.61111
1
6         7.0   9.0  50.0  4.0   7.0  0.562500  0.925926  0.074074  0.43750
0
7         8.0   7.0  48.0  4.0  11.0  0.388889  0.923077  0.076923  0.61111
1
8         9.0   9.0  51.0  2.0   8.0  0.529412  0.962264  0.037736  0.47058
8
9        10.0   6.0  44.0  5.0  15.0  0.285714  0.897959  0.102041  0.71428
6
Average   5.5   8.2  47.8  3.9  10.1  0.459630  0.924103  0.075897  0.54037
0

         Precision  F1_measure  Accuracy  Error_rate      BACC       TSS  \
0         0.750000    0.705882  0.857143    0.142857  0.794872  0.589744
1         0.909091    0.645161  0.842857    0.157143  0.740000  0.480000
2         0.545455    0.545455  0.857143    0.142857  0.730354  0.460709
3         0.615385    0.444444  0.714286    0.285714  0.620722  0.241443
4         0.571429    0.457143  0.728571    0.271429  0.629252  0.258503
5         0.700000    0.500000  0.800000    0.200000  0.665598  0.331197
6         0.692308    0.620690  0.842857    0.157143  0.744213  0.488426
7         0.636364    0.482759  0.785714    0.214286  0.655983  0.311966
8         0.818182    0.642857  0.857143    0.142857  0.745838  0.491676
9         0.545455    0.375000  0.714286    0.285714  0.591837  0.183673
Average   0.678367    0.541939  0.800000    0.200000  0.691867  0.383734

              HSS  Brier_score       AUC
0        0.611973     0.111189  0.891026
1        0.554913     0.130682  0.865000
2        0.460709     0.108178  0.852080
3        0.271592     0.196074  0.737280
4        0.285714     0.166721  0.810496
5        0.387500     0.149098  0.786325
6        0.522924     0.127961  0.839120
7        0.357405     0.133081  0.832265
8        0.558638     0.124127  0.839068
9        0.212598     0.165930  0.819242
Average  0.422397     0.141304  0.827190
```

ROC Curve

In [25]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Aggregating true labels and predicted probabilities across all folds
y_true_all = []
y_proba_all = []

# Perform 10-Fold Cross-validation with the best LSTM model
for train_idx, val_idx in cv.split(X, y):
    X_train_fold, X_val_fold = X[train_idx], X[val_idx]
    y_train_fold, y_val_fold = y[train_idx], y[val_idx]

    # Fit the best model on the training fold
    best_model.fit(X_train_fold, y_train_fold)

    # Get the predicted probabilities for the validation fold
    y_proba = best_model.model.predict(X_val_fold).flatten()

    # Store the true labels and predicted probabilities
    y_true_all.extend(y_val_fold)
    y_proba_all.extend(y_proba)

# Convert lists to numpy arrays for calculations
y_true_all = np.array(y_true_all)
y_proba_all = np.array(y_proba_all)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_true_all, y_proba_all)

# Calculate the ROC AUC score
roc_auc = roc_auc_score(y_true_all, y_proba_all)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Best LSTM Model (10-Fold CV)')
plt.legend(loc='lower right')
plt.show()

print(f"Average ROC AUC Score across folds: {roc_auc:.2f}")
```

```
/Users/krkaushikkumar/anaconda3/lib/python3.11/site-packages/keras/src/laye
rs/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(**kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
```




```
Average ROC AUC Score across folds: 0.83
```

Result

In [26]:

```python
# Add a new column to each DataFrame for the model name
metrics_rf['Model'] = 'Random Forest'
metrics_svm['Model'] = 'SVM'
metrics_dt['Model'] = 'Decision Tree'
metrics_lstm['Model'] = 'LSTM'

# Select the last row from each DataFrame and merge them vertically
merged_metrics = pd.concat([metrics_rf[-1:], metrics_svm[-1:], metrics_dt[-

# Move the 'Model' column to the first position
merged_metrics = merged_metrics[['Model'] + [col for col in merged_metrics

# Transpose the DataFrame to have models as column names
merged_metrics = merged_metrics.set_index('Model').T

# Display the DataFrame with bold model names in the first row
# Formatting the model names as bold for display (works in Jupyter environm
merged_metrics.columns = [f"{col}" for col in merged_metrics.columns]

# Display the transposed DataFrame
merged_metrics
```

Out[26]:

	Random Forest	SVM	Decision Tree	LSTM
Fold	5.500000	5.500000	5.500000	5.500000
TP	6.100000	4.500000	5.800000	8.200000
TN	37.300000	38.800000	35.400000	47.800000
FP	4.200000	2.700000	6.100000	3.900000
FN	8.400000	10.000000	8.700000	10.100000
TPR	0.425591	0.325399	0.404036	0.459630
TNR	0.898264	0.934257	0.854288	0.924103
FPR	0.101736	0.065743	0.145712	0.075897
FNR	0.574409	0.674601	0.595964	0.540370
Precision	0.583654	0.622619	0.480086	0.678367
F1_measure	0.480395	0.418647	0.425187	0.541939
Accuracy	0.775000	0.773214	0.735714	0.800000
Error_rate	0.225000	0.226786	0.264286	0.200000
BACC	0.661927	0.629828	0.629162	0.691867
TSS	0.323855	0.259655	0.258324	0.383734
HSS	0.347152	0.301987	0.260797	0.422397
Brier_score	0.155052	0.143506	0.210210	0.141304
AUC	0.783804	0.817954	0.704169	0.827190

Model Comparison and Ranking

In this step, the performance of different models is compared using key evaluation metrics, and the best model is determined based on a ranking system.

Key Metrics Selected for Comparison:

- **Accuracy**
- **AUC (Area Under the Curve)**
- **Precision**
- **F1-Score**
- **Balanced Accuracy (BACC)**
- **Heidke Skill Score (HSS)**

Process:

1. **Metric Selection**: The relevant metrics were selected for comparison.
2. **Ranking**: Models were ranked based on each of the key metrics, with higher values indicating better performance (e.g., higher accuracy and AUC are favorable).
3. **Total Score Calculation**: The ranks for each model across all key metrics were summed. A lower total score indicates better overall performance.
4. **Best Model Identification**: The model with the lowest total score was identified as the best model.

Outcome:

- **Ranking Results**: The ranking of models across all metrics is provided.
- **Best Model**: The model with the lowest total score is determined to be the best performer overall.

In [27]:

```
# Assuming merged_metrics is the transposed DataFrame with model metrics
# Select key metrics for comparison
key_metrics = ['Accuracy', 'AUC', 'Precision', 'F1_measure', 'BACC', 'HSS'

# Filter the merged metrics DataFrame to only key metrics
metrics_for_ranking = merged_metrics.loc[key_metrics]

# Rank each model for each metric (higher is better, so we rank by descend
ranks = metrics_for_ranking.rank(ascending=False, axis=1)

# Sum ranks for each model to get a total score (lower score indicates bet
total_scores = ranks.sum()

# Find the model with the lowest total score
best_model = total_scores.idxmin()

# Display the ranking results and the best model
print("Ranking of Models by Metrics:\n", ranks)
print("\nTotal Scores for Each Model:\n", total_scores)
print(f"\nBest Model Overall: {best_model}")
```

```
Ranking of Models by Metrics:
              Random Forest  SVM  Decision Tree  LSTM
Accuracy               2.0  3.0            4.0   1.0
AUC                    3.0  2.0            4.0   1.0
Precision              3.0  2.0            4.0   1.0
F1_measure             2.0  4.0            3.0   1.0
BACC                   2.0  3.0            4.0   1.0
HSS                    2.0  3.0            4.0   1.0

Total Scores for Each Model:
 Random Forest    14.0
SVM              17.0
Decision Tree    23.0
LSTM              6.0
dtype: float64

Best Model Overall: LSTM
```