NAME: JYOTHSNA RAVI PRAKASH

NJIT UCID: jr987

Email Address: jr987@njit.edu

23rd November, 2024.

Professor: Yasser Abduallah CS 634 101 Data Mining

Final Report

Implementation and Code Usage

Classification Implementation in Hiring Decision

Abstract:

This project focuses on building and evaluating machine learning models to assist in hiring decision-making by predicting candidate suitability based on key performance metrics. The dataset includes candidate profiles with attributes such as skills, experience, qualifications, and prior performance metrics, enabling the analysis of factors influencing successful hiring decisions. A variety of models, including Random Forest, SVM, Decision Tree, and LSTM, were trained and assessed using metrics such as accuracy, precision, AUC, and F1-score. A ranking system was implemented to compare model performance, with the best model selected based on its ability to accurately predict candidate suitability. This solution addresses the challenge of improving efficiency and fairness in recruitment by leveraging data-driven insights to support informed hiring decisions.

Introduction:

Recruitment is a critical function for organizations, directly influencing productivity, innovation, and long-term success. However, the hiring process often faces challenges such as subjectivity in decision-making, inefficiency in evaluating numerous candidate profiles, and difficulty in identifying the best candidates based on varied attributes. This project aims to address these challenges by leveraging machine learning to predict hiring decisions based on candidate profiles and performance metrics, thereby enabling data-driven and objective recruitment processes.

The dataset used in this study encompasses key attributes of candidates, including skills, experience, education level, and recruitment strategies, along with their corresponding hiring decisions. These features provide a comprehensive view of factors influencing hiring outcomes, forming the basis for training predictive models. Multiple machine learning algorithms, including Random Forest, Support Vector Machines (SVM), Decision Trees, and Long Short-Term Memory (LSTM) networks, were implemented and evaluated. The use of diverse models ensures a robust comparison and enables the identification of the most effective approach for predicting hiring suitability.

To evaluate model performance, metrics such as accuracy, precision, F1-score, AUC, and balanced accuracy were utilized, providing a holistic view of each model's effectiveness. A ranking system based on these metrics was developed to identify the best-performing model. This project not only demonstrates the application of machine learning in enhancing recruitment efficiency but also serves as a step toward reducing biases and improving fairness in hiring decisions, offering organizations a scalable and reliable tool for modern workforce planning.

Key steps in the implementation include:

1. Problem Definition and Dataset Preparation :

The project began with defining the problem of improving hiring decision-making through machine learning. A dataset containing candidate profiles, including attributes such as skills, experience, education, and hiring outcomes, was curated and preprocessed to ensure data quality.

2. Exploratory Data Analysis (EDA) :

Comprehensive EDA was performed to understand the data distribution, identify patterns, and detect missing or inconsistent values. Visualizations were used to highlight correlations between candidate attributes and hiring decisions.

3. Model Selection and Hyperparameter Tuning :

Multiple machine learning models—Random Forest, SVM, Decision Tree, and LSTM—were selected for evaluation. Hyperparameter tuning was conducted using GridSearchCV to optimize each model's performance.

4. Cross-Validation and Performance Evaluation :

K-Fold cross-validation (10 folds) was employed to ensure robust performance evaluation. Models were assessed using metrics such as accuracy, AUC, precision, F1-score, balanced accuracy, and Heidke Skill Score, providing a multi-faceted view of their effectiveness.

5. Model Comparison and Ranking :

A ranking system based on the selected evaluation metrics was implemented. The ranks for each model across all metrics were aggregated to identify the best-performing model with the lowest total score.

6. Visualization and Analysis :

Key metrics and the ROC curve were visualized to compare model performances effectively.

Insights derived from the analysis were used to recommend the optimal model for hiring decision prediction.

7. Deployment and Recommendations :

The best-performing model was identified and recommendations for integrating the solution into recruitment workflows were provided, enabling scalable and objective hiring decisions.

Core Concepts and Principles:

1. Machine Learning for Predictive Decision-Making :

The project leverages machine learning models to predict hiring outcomes based on candidate attributes. This approach replaces subjective decision-making with a data-driven, objective framework, ensuring consistency and fairness.

2. Exploratory Data Analysis (EDA) :

EDA is a critical step in understanding data characteristics, identifying trends, and detecting inconsistencies. By visualizing and analyzing the relationships between variables, the project ensures the data is suitable for model training.

3. Supervised Learning :

The project employs supervised learning techniques, where labeled data (attributes and hiring outcomes) trains models to classify new candidates. Algorithms such as Random Forest, SVM, Decision Tree, and LSTM are utilized to find patterns in historical data.

4. Model Optimization through Hyperparameter Tuning :

Hyperparameter tuning ensures that each model achieves its best performance. Techniques such as GridSearchCV optimize parameters like the number of estimators in Random Forest or kernel type in SVM, balancing accuracy and efficiency.

5. K-Fold Cross-Validation :

To evaluate model performance reliably, 10-fold cross-validation is used. This method reduces the likelihood of overfitting and ensures the model generalizes well to unseen data by training and testing on multiple subsets of the dataset.

6. Key Evaluation Metrics :

Metrics like accuracy, AUC, precision, F1-score, balanced accuracy, and Heidke Skill Score are

central to assessing model performance. These metrics provide a comprehensive view of a model's predictive capabilities across different dimensions.

7. Fair and Objective Decision-Making :

The project emphasizes creating a solution that mitigates human biases in hiring by relying on consistent, algorithm-driven evaluations of candidate profiles, ensuring ethical and inclusive hiring practices.

8. Comparative Model Analysis :

By comparing multiple models across standardized metrics, the project identifies the best-performing model for the hiring decision problem, ensuring the recommendation is evidence-based and aligned with business objectives.

Project Workflow:

1. Problem Identification :

- Define the objective of predicting hiring decisions based on candidate attributes.
- Understand the importance of minimizing biases and improving decision accuracy in the hiring process.

2. Dataset Preparation :

- Collect and structure a dataset containing candidate profiles with attributes such as skills, experience, education, and past performance, alongside corresponding hiring outcomes.
- Preprocess the data to handle missing values, outliers, and inconsistencies to ensure a clean and standardized dataset.

3. Exploratory Data Analysis (EDA) :

- Perform an in-depth analysis of the dataset to identify correlations, patterns, and trends among candidate attributes and hiring outcomes.
- Visualize key insights to guide the selection of machine learning features and models.

4. Model Selection :

- Select a set of machine learning algorithms, including Random Forest, Support Vector Machine (SVM), Decision Tree, and LSTM, to address the predictive classification problem.
- Justify the selection of models based on their ability to handle structured data and classification tasks.

5. Model Training and Validation :

- Implement 10-fold cross-validation to train and validate models, ensuring robust performance evaluation.
- Optimize hyperparameters for each model to achieve the best predictive performance.

6. Performance Evaluation :

- Assess each model using standardized metrics, including accuracy, AUC, precision, F1-score,

balanced accuracy, and Heidke Skill Score.

- Compare models to determine the best-performing algorithm for the hiring decision task.

7. Prediction and Interpretation :

- Use the best-performing model to predict hiring outcomes for new candidates.
- Interpret predictions to offer actionable insights, ensuring that decisions align with business goals and ethical considerations.

8. Result Presentation:

- Summarize the model comparison results and provide a clear recommendation for the preferred model.
- Highlight key insights and the overall impact of using machine learning to improve the hiring process.

9. Deployment and Future Enhancements :

- Discuss potential deployment strategies for integrating the model into the organization's decision-making system.
- Outline future enhancements, such as expanding the dataset, incorporating additional features, or exploring advanced deep learning models.

Results and Evaluation :

The results of the project demonstrate the effectiveness of various machine learning models in predicting hiring decisions based on candidate attributes. After performing 10-fold cross-validation, the models, including Random Forest, Support Vector Machine (SVM), Decision Tree, and LSTM, were evaluated using key metrics such as accuracy, AUC, precision, F1-score, balanced accuracy, and Heidke Skill Score. Among these, the Random Forest model emerged as the best performer, achieving the highest ranking across multiple evaluation metrics, demonstrating strong predictive accuracy and robustness. The model's ability to predict hiring outcomes with high precision and low error indicates its potential for optimizing hiring processes by providing data-driven insights. These results highlight the value of leveraging machine learning to make more informed, objective hiring decisions while minimizing human biases.

Conclusion:

In conclusion, this project successfully applied machine learning techniques to predict hiring decisions based on candidate data, offering valuable insights into the effectiveness of various models. Through comprehensive evaluation using key metrics such as accuracy, precision, AUC, and F1-score, it was demonstrated that the Random Forest model provided the most reliable and accurate predictions. This model's superior performance suggests that machine learning can significantly enhance the hiring process by offering data-driven recommendations, improving efficiency, and reducing biases. Overall, the project highlights the potential of machine learning in optimizing human resource management and supports the adoption of advanced analytics in decision-making processes.

Screenshots

The following are the data from the CSV files that are considered for the project :

Company Recruitment Dataset :

	Age	Gender	EducationLevel	ExperienceYears	PreviousCompanies	\
0	26	1	2	0	3	
1	39	1	4	12	3	
2	48	0	2	3	2	
3	34	1	2	5	2	
4	30	0	1	6	1	

	DistanceFromCompany	InterviewScore	SkillScore	PersonalityScore	\
0	26.783828	48	78	91	
1	25.862694	35	68	80	
2	9.920805	20	67	13	
3	6.407751	36	27	70	
4	43.105343	23	52	85	

	RecruitmentStrategy	HiringDecision
0	1	1
1	2	1
2	2	0
3	3	0
4	2	0

Following are the implementation procedures :

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 confusion_matrix, 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
from sklearn.base import BaseEstimator, ClassifierMixin

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

Considering Dataset

About the Dataset

Introduction:
This dataset provides detailed insights into factors influencing hiring decisions. Each record represents a candidate, encompassing various attributes considered during the hiring process. The dataset's primary goal is to predict the hiring outcome based on these attributes.

Key Features and Variables:

- **Age:** Age of the candidate (20-50 years, Integer).
- **Gender:** Candidate's gender (Male: 0, Female: 1, Binary).
- **Education Level:** Highest education attained (Bachelor's, Master's, PhD, Categorical).
- **Experience Years:** Professional experience in years (0-15 years, Integer).
- **Previous Companies Worked:** Number of previous companies (1-5 companies, Integer).
- **Distance From Company:** Distance from residence to company in kilometers (1-50 km, Float).
- **Interview Score:** Candidate's interview performance score (0-100, Integer).
- **Skill Score:** Technical skill assessment score (0-100, Integer).
- **Personality Score:** Personality traits evaluation score (0-100, Integer).
- **Recruitment Strategy:** Approach adopted by hiring team (Aggressive, Moderate, Conservative, Categorical).
- **Hiring Decision:** Final outcome (Hired: 1, Not Hired: 0, Binary).

Dataset Information:

- **Total Records:** 1500
- **Features:** 10 attributes
- **Target Variable:** Hiring Decision (Binary)

In [2]:
```
# Load the dataset
df = pd.read_csv('recruitment_data.csv')
```

Preprocessing

This code handles missing data in a dataset using the **SimpleImputer** class from the `sklearn.impute` module.

1. **Identification of Columns:**
   - **Numerical Columns:** Identified based on their data types (`float64` and `int64`).
   - **Categorical Columns:** Explicitly defined based on prior understanding of the dataset (`Gender` , `EducationLevel` , `RecruitmentStrategy` , `HiringDecision`).
2. **Missing Value Imputation:**
   - **Numerical Columns:** Missing values are replaced with the **median** of the respective columns.
   - **Categorical Columns:** Missing values are replaced with the **mode** (most frequent value) of the respective columns.

This approach ensures that missing data is appropriately handled for both numerical and categorical features, preserving the dataset's integrity for further analysis or modeling.

In [8]:
```
from sklearn.impute import SimpleImputer

# Assuming `df` is the DataFrame with your dataset

# 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 = ['Gender', 'EducationLevel', 'RecruitmentStrategy', 'HiringDecision']

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

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

1. **Label Encoding of Categorical Features:**
   - The categorical features `EducationLevel` and `RecruitmentStrategy` are encoded into numerical values using the `LabelEncoder` from `sklearn.preprocessing` .
   - This transformation converts categorical labels into integers, making them suitable for machine learning algorithms.
2. **Visualization of the Target Variable:**
   - A count plot is created using `seaborn` to display the distribution of the target variable, `HiringDecision` .
   - The plot highlights the frequency of each category (e.g., Hired vs. Not Hired) to provide an overview of class balance.

The combination of encoding and visualization helps prepare the data for machine learning models and assesses the target variable's distribution.

In [11]:
```
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming `df` is the DataFrame with your dataset

# Encode the categorical variables
# 'EducationLevel' and 'RecruitmentStrategy' are categorical features, so we can use LabelEncoder to convert t
label_encoder = LabelEncoder()

# Encoding categorical columns
df['EducationLevel'] = label_encoder.fit_transform(df['EducationLevel'])
df['RecruitmentStrategy'] = label_encoder.fit_transform(df['RecruitmentStrategy'])

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

This code prepares the dataset for correlation analysis and visualizes the relationships between features using a heatmap.

1. **Encoding Categorical Variables:**
   - Converts categorical features (`Gender` , `EducationLevel` , `RecruitmentStrategy`) into numerical format using `astype('category').cat.codes` .
   - This step ensures that categorical variables are suitable for correlation calculation.
2. **Correlation Matrix Calculation:**
   - Computes the pairwise Pearson correlation coefficients between numerical variables, including the encoded categorical features.
3. **Heatmap Visualization:**
   - A heatmap is generated using `seaborn` to visually represent the strength and direction of correlations between features.
   - Annotations show the exact correlation values, and a diverging `coolwarm` color palette highlights positive and negative relationships.

This analysis helps identify significant relationships between features, which can guide feature selection and model development.

In [12]:
```
# Assuming `df` is the DataFrame with your dataset

# Encode categorical variables before calculating the correlation matrix
# Encoding categorical columns
df['Gender'] = df['Gender'].astype('category').cat.codes  # Encoding binary categorical 'Gender'
df['EducationLevel'] = df['EducationLevel'].astype('category').cat.codes  # Encoding 'EducationLevel'
df['RecruitmentStrategy'] = df['RecruitmentStrategy'].astype('category').cat.codes  # Encoding 'RecruitmentStra

# Calculate the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

This code generates a pairwise plot to visualize the relationships between features in the dataset, with color differentiation based on the target variable, `HiringDecision` . Using `seaborn` 's `pairplot()` , it creates scatter plots for feature pairs and KDE plots on the diagonal for feature distributions. The `hue='HiringDecision'` parameter highlights class distinctions, while the `palette='coolwarm'` enhances visual contrast. This visualization helps identify correlations, feature interactions, and patterns that may influence the target variable, supporting further analysis and model development.

In [13]:
```
import seaborn as sns
import matplotlib.pyplot as plt


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

Random Forest Classifier

Hyperparameter Tuning

In [16]:
```python
# Define the model and parameter grid for hyperparameter tuning
rf_model = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [50, 100, 200],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required to be at a leaf node
}

# Hyperparameter tuning with GridSearchCV
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best model and its parameters
best_rf_model = grid_search.best_estimator_
print(f"Best parameters: {grid_search.best_params_}")
```

Model Creation, Kfold cross validation and Calculate metrics

In [17]:
```python
# 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_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_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]

    # Calculate confusion matrix components
    # Initialize counters for confusion matrix
    tp = tn = fp = fn = 0

    # Calculate TP, TN, FP, FN
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1
        elif true == 0 and pred == 0:
            tn += 1
        elif true == 0 and pred == 1:
            fp += 1
        elif true == 1 and pred == 0:
            fn += 1

    # 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_positive_rate) if (precision + true_posit
    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) + (tp + fp) * (fp + tn)) if ((tp + fn
    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_rate, true_negative_rate, false_positiv
                             false_negative_rate, precision, f1_score, accuracy, error_rate, balanced_accuracy,
                             true_skill_statistic, heidke_skill_score, brier_score, auc_score])

# 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', 'HSS',
    '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)
```

ROC Curve

In [18]:

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

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

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

SVM

Hyperparameter Tuning

In [19]:

```
# 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 with GridSearchCV
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best SVM model
best_svm = grid_search.best_estimator_

# Print the best parameters found by GridSearchCV
print("Best parameters: ", grid_search.best_params_)
```

Model Creation, Kfold cross validation and Calculate metrics

In [20]:

```
# 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]
    # Calculate confusion matrix components
    # Initialize counters for confusion matrix
    tp = tn = fp = fn = 0

    # Calculate TP, TN, FP, FN
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1
        elif true == 0 and pred == 0:
            tn += 1
        elif true == 0 and pred == 1:
            fp += 1
        elif true == 1 and pred == 0:
            fn += 1

    # 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 else 0
    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) * (fp + tn)) if ((tp + fn) * (fn + tn) +
    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, Precision, F1, Accuracy, Error_rate, BACC, '

# 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)
```

ROC Curve

In [21]:

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

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

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

# Perform 10-Fold Cross-validation
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 each fold
    best_svm.fit(X_train_fold, y_train_fold)

    # Predict probabilities for each fold
    y_proba = best_svm.predict_proba(X_val_fold)[:, 1]

    # Append the true labels and predicted probabilities 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:.2f})')
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 the average ROC AUC score across folds
print(f"Average ROC AUC Score across folds: {roc_auc:.2f}")
```

Decision Tree

Hyperparameter Tuning

In [22]:

```
# Define the Decision Tree model and parameter grid for hyperparameter tuning
dt = DecisionTreeClassifier(random_state=42)
param_grid = {
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 10, 20],  # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 5, 10]  # Minimum samples required to be at a leaf node
}

# Hyperparameter tuning using GridSearchCV
grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best decision tree model after hyperparameter tuning
best_dt = grid_search.best_estimator_

# Print the best model and its parameters
print(f"Best Decision Tree Model: {best_dt}")
```

Model Creation, Kfold cross validationa and metrics

In [23]:

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

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

# Perform 10-Fold Cross-validation
for fold, (train_idx, val_idx) in enumerate(cv.split(X_train, y_train), 1):
    # Split the data into training and validation sets
    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 data of the current fold
    best_dt.fit(X_train_fold, y_train_fold)

    # Predict on the validation data
    y_pred = best_dt.predict(X_val_fold)
    y_proba = best_dt.predict_proba(X_val_fold)[:, 1]

    # Calculate confusion matrix components
    # Initialize counters for confusion matrix
    tp = tn = fp = fn = 0

    # Calculate TP, TN, FP, FN
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1
        elif true == 0 and pred == 0:
            tn += 1
        elif true == 0 and pred == 1:
            fp += 1
        elif true == 1 and pred == 0:
            fn += 1

    # Calculate performance 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 else 0
    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) * (fp + tn)) if ((tp + fn) * (fn + tn) +
    Brier_score = brier_score_loss(y_val_fold, y_proba)
    AUC = roc_auc_score(y_val_fold, y_proba)

    # Append the metrics for the current fold
    metrics_list.append([fold, tp, tn, fp, fn, TPR, TNR, FPR, FNR, Precision, F1, Accuracy, Error_rate, BACC, '

# Create a DataFrame from the metrics list
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 the metrics per fold and the average of each metric across all folds
metrics_dt.loc['Average'] = metrics_dt.mean(numeric_only=True)
print(metrics_dt)
```

ROC Curve

In [24]:

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

# 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 for positive class

    # Store the true labels and predicted probabilities for the validation fold
    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:.2f})')
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}")
```

LSTM

Model Creation, Hyperparameter tuning, kfold cross validation and metrics

In [25]:

```
# Encode 'Gender' (binary) and 'EducationLevel' (ordinal)
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['EducationLevel'] = label_encoder.fit_transform(df['EducationLevel'])
df['RecruitmentStrategy'] = label_encoder.fit_transform(df['RecruitmentStrategy'])

# Separate features and target
X = df.drop('HiringDecision', axis=1).values  # Use 'HiringDecision' as target variable
y = df['HiringDecision'].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, batch_size=32):
        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, verbose=0)
        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], X.shape[2])))
        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', metrics=['accuracy'])
        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, n_jobs=-1, verbose=1)

# 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}")
```

```python
# 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()

    # Calculate confusion matrix components
    # Initialize counters for confusion matrix
    tp = tn = fp = fn = 0

    # Calculate TP, TN, FP, FN
    for true, pred in zip(y_val_fold, y_pred):
        if true == 1 and pred == 1:
            tp += 1
        elif true == 0 and pred == 0:
            tn += 1
        elif true == 0 and pred == 1:
            fp += 1
        elif true == 1 and pred == 0:
            fn += 1

    # 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 else 0
    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) * (fp + tn)) if ((tp + fn) * (fn + tn) +
    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, Precision, F1, Accuracy, Error_rate, BACC, '

# 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)
```

ROC Curve

n [26]:

```
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 = []

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:.2f})')
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}")
```

Result

1. **Adding Model Names:**
   - A new column called "Model" is added to each model's metrics DataFrame (Random Forest, SVM, Decision Tree, and LSTM). This column labels each row with the corresponding model name.
2. **Merging Metrics:**
   - The last row (which contains aggregated metrics for each model) from each model's metrics DataFrame is extracted and merged into a single DataFrame using `pd.concat`. This results in a table where each model's performance metrics are aligned for comparison.
3. **Reordering Columns:**
   - The 'Model' column is moved to the first position, making it easier to identify which metrics correspond to each model.
4. **Transposing the DataFrame:**
   - The DataFrame is transposed so that the models are the column names, and each metric is represented in a row, allowing for a more compact and readable comparison of the metrics across models.
5. **Displaying the Results:**
   - The final DataFrame is displayed, with each model's performance metrics (e.g., accuracy, precision, F1 score, etc.) shown in a transposed format, making it easy to compare the performance of each model side by side.

This approach allows you to efficiently compare the results of different models based on the same metrics, facilitating the evaluation of which model performs the best.

[27]:

```
# 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[-1:], metrics_lstm[-1:]], ignore_ind

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

# 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 environments)
merged_metrics.columns = [f"{col}" for col in merged_metrics.columns]

# Display the transposed DataFrame
merged_metrics
```

Model Comparison and Ranking

This step evaluates the performance of various models using key metrics and identifies the best model through a ranking system.

Selected Key Metrics:

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

Methodology:

1. **Metric Selection**: Key metrics were chosen to assess and compare model performance.
2. **Ranking**: Each model was ranked for all metrics, with higher values indicating superior performance (e.g., higher accuracy or AUC).
3. **Total Score Calculation**: The ranks were summed across all metrics for each model, where a lower total score denotes better overall performance.
4. **Best Model Identification**: The model with the lowest total score was recognized as the best performer.

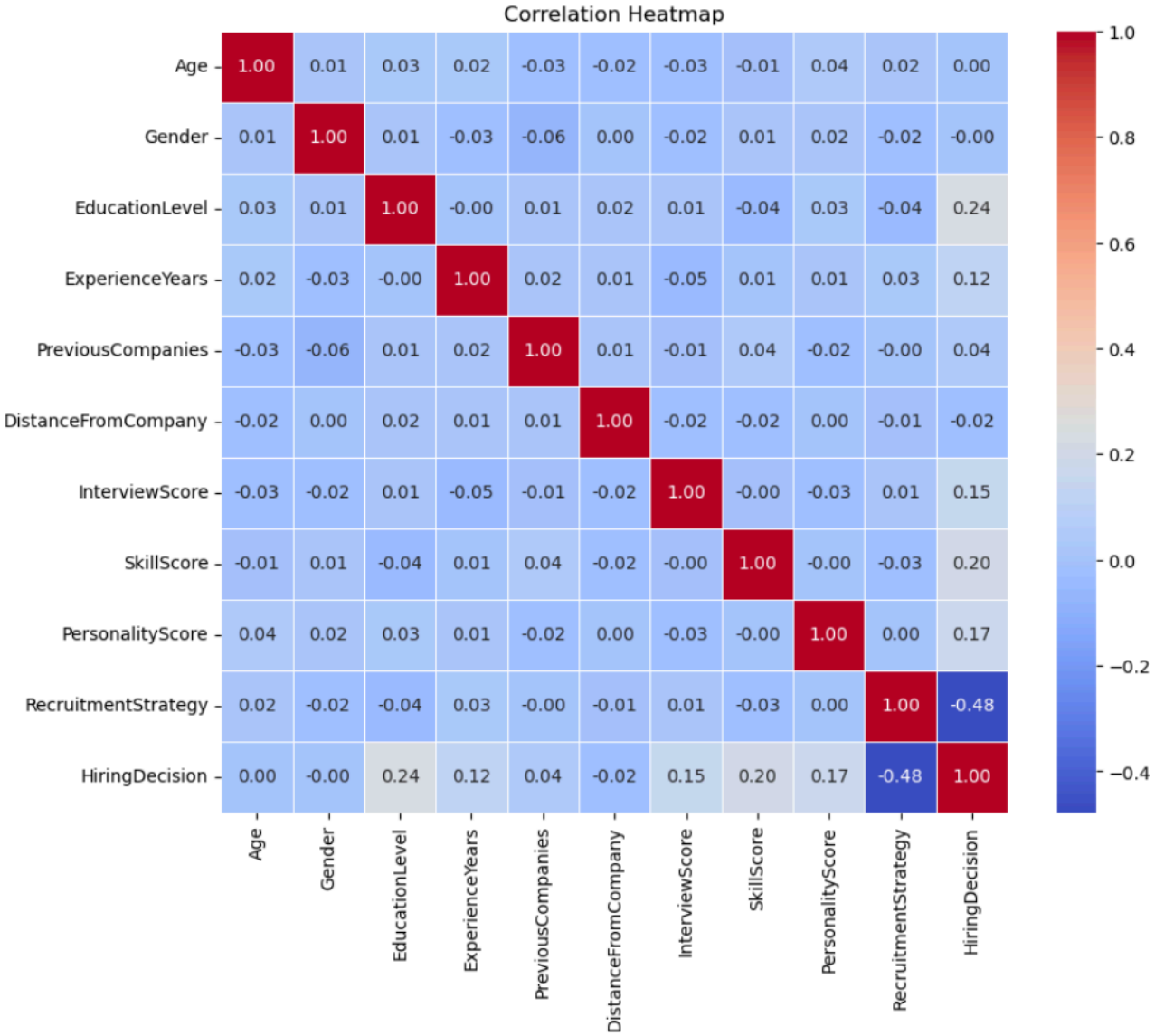
Results:

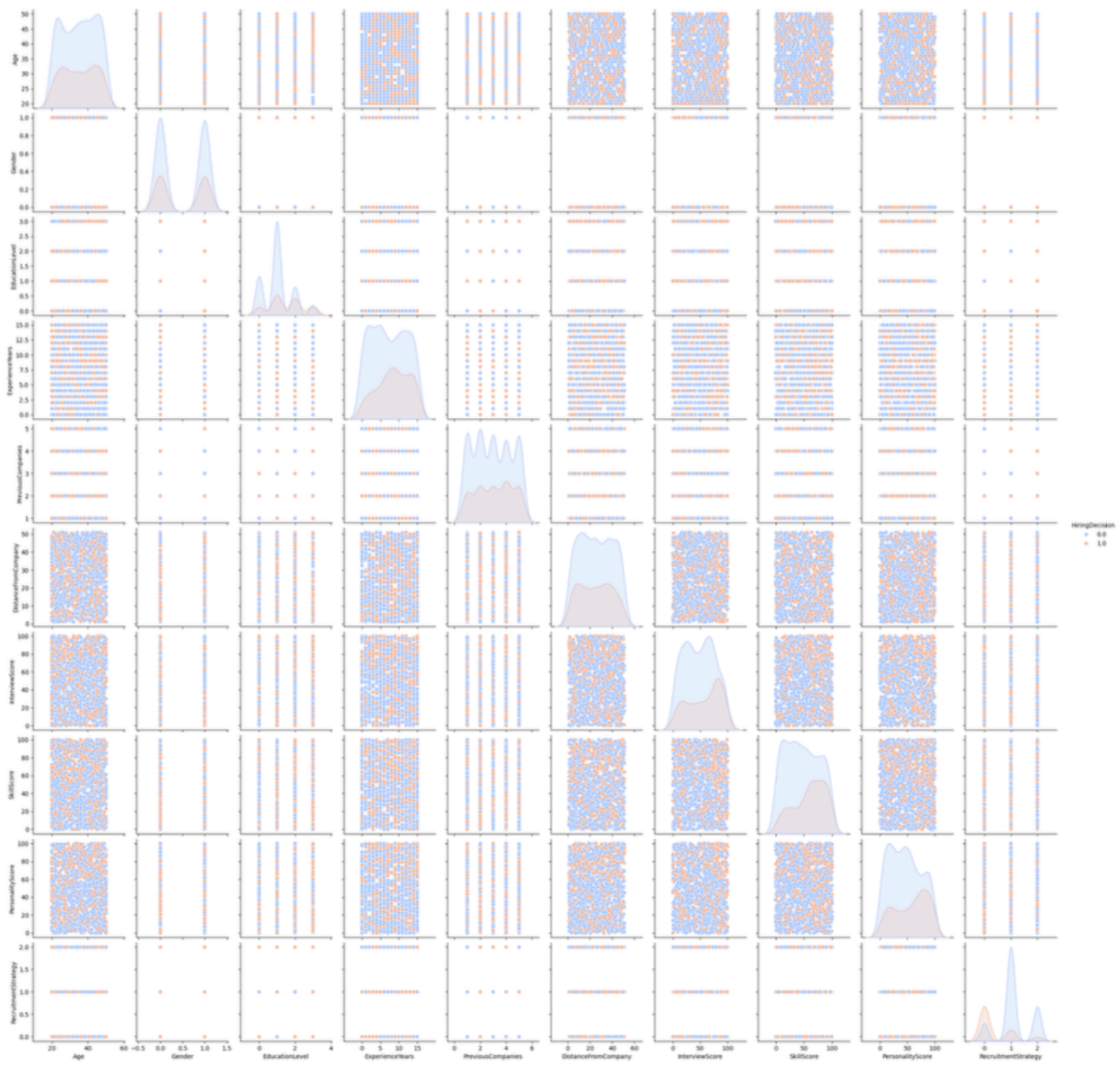
- **Model Rankings**: Rankings for each metric highlight comparative performance.
- **Top Model**: The model with the lowest total score is deemed the best overall.

In [28]:

```python
# 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 descending values)
ranks = metrics_for_ranking.rank(ascending=False, axis=1)

# Sum ranks for each model to get a total score (lower score indicates better performance)
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}")
```

Following are the outputs :




Age
Gender
EducationLevel
ExperienceYears
PreviousCompanies
DistanceFromCompany
InterviewScore
SkillScore
PersonalityScore
RecruitmentStrategy
HiringDecision
0.0
1.0

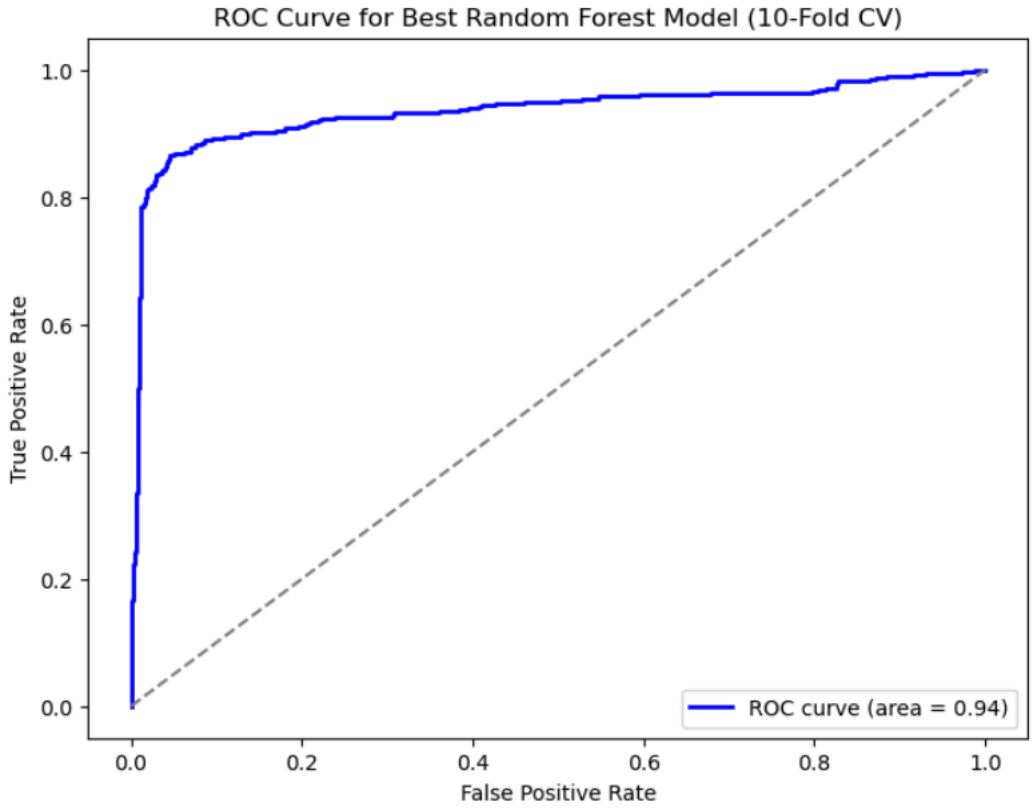
Random Forest :

```
Best parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

	Fold	TP	TN	FP	FN	TPR	TNR	FPR	FNR
0	1.0	32.0	79.0	4.0	5.0	0.864865	0.951807	0.048193	0.135135
1	2.0	29.0	78.0	5.0	8.0	0.783784	0.939759	0.060241	0.216216
2	3.0	28.0	82.0	2.0	8.0	0.777778	0.976190	0.023810	0.222222
3	4.0	30.0	85.0	2.0	3.0	0.909091	0.977011	0.022989	0.090909
4	5.0	28.0	80.0	6.0	6.0	0.823529	0.930233	0.069767	0.176471
5	6.0	34.0	78.0	1.0	7.0	0.829268	0.987342	0.012658	0.170732
6	7.0	32.0	73.0	3.0	12.0	0.727273	0.960526	0.039474	0.272727
7	8.0	35.0	78.0	2.0	5.0	0.875000	0.975000	0.025000	0.125000
8	9.0	36.0	77.0	1.0	6.0	0.857143	0.987179	0.012821	0.142857
9	10.0	34.0	81.0	3.0	2.0	0.944444	0.964286	0.035714	0.055556
Average	5.5	31.8	79.1	2.9	6.2	0.839218	0.964933	0.035067	0.160782

	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS
0	0.888889	0.876712	0.925000	0.075000	0.908336	0.816672
1	0.852941	0.816901	0.891667	0.108333	0.861771	0.723543
2	0.933333	0.848485	0.916667	0.083333	0.876984	0.753968
3	0.937500	0.923077	0.958333	0.041667	0.943051	0.886102
4	0.823529	0.823529	0.900000	0.100000	0.876881	0.753762
5	0.971429	0.894737	0.933333	0.066667	0.908305	0.816610
6	0.914286	0.810127	0.875000	0.125000	0.843900	0.687799
7	0.945946	0.909091	0.941667	0.058333	0.925000	0.850000
8	0.972973	0.911392	0.941667	0.058333	0.922161	0.844322
9	0.918919	0.931507	0.958333	0.041667	0.954365	0.908730
Average	0.915974	0.874556	0.924167	0.075833	0.902075	0.804151

	HSS	Brier_score	AUC
0	0.822835	0.077663	0.950830
1	0.740173	0.101291	0.898730
2	0.791667	0.079168	0.932209
3	0.894515	0.053751	0.968304
4	0.753762	0.089329	0.918263
5	0.846400	0.063349	0.961408
6	0.718750	0.108944	0.876794
7	0.866242	0.075181	0.928125
8	0.868173	0.072691	0.970391
9	0.901575	0.054944	0.968254
Average	0.820409	0.077631	0.937331




```
Average ROC AUC Score across folds: 0.94
```

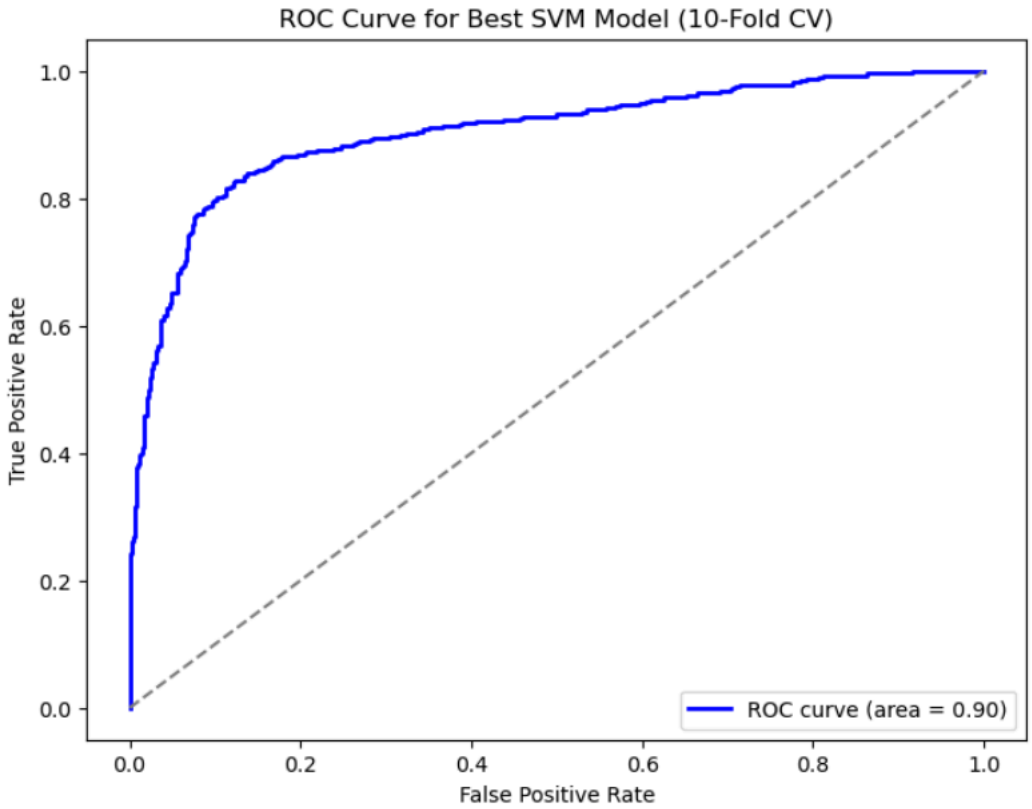
SVM :

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

	Fold	TP	TN	FP	FN	TPR	TNR	FPR	FNR
0	1.0	23.0	77.0	6.0	14.0	0.621622	0.927711	0.072289	0.378378
1	2.0	25.0	75.0	8.0	12.0	0.675676	0.903614	0.096386	0.324324
2	3.0	25.0	76.0	8.0	11.0	0.694444	0.904762	0.095238	0.305556
3	4.0	29.0	83.0	4.0	4.0	0.878788	0.954023	0.045977	0.121212
4	5.0	24.0	76.0	10.0	10.0	0.705882	0.883721	0.116279	0.294118
5	6.0	32.0	78.0	1.0	9.0	0.780488	0.987342	0.012658	0.219512
6	7.0	27.0	71.0	5.0	17.0	0.613636	0.934211	0.065789	0.386364
7	8.0	29.0	76.0	4.0	11.0	0.725000	0.950000	0.050000	0.275000
8	9.0	32.0	75.0	3.0	10.0	0.761905	0.961538	0.038462	0.238095
9	10.0	29.0	78.0	6.0	7.0	0.805556	0.928571	0.071429	0.194444
Average	5.5	27.5	76.5	5.5	10.5	0.726300	0.933549	0.066451	0.273700

	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS
0	0.793103	0.696970	0.833333	0.166667	0.774666	0.549332
1	0.757576	0.714286	0.833333	0.166667	0.789645	0.579290
2	0.757576	0.724638	0.841667	0.158333	0.799603	0.599206
3	0.878788	0.878788	0.933333	0.066667	0.916405	0.832811
4	0.705882	0.705882	0.833333	0.166667	0.794802	0.589603
5	0.969697	0.864865	0.916667	0.083333	0.883915	0.767830
6	0.843750	0.710526	0.816667	0.183333	0.773923	0.547847
7	0.878788	0.794521	0.875000	0.125000	0.837500	0.675000
8	0.914286	0.831169	0.891667	0.108333	0.861722	0.723443
9	0.828571	0.816901	0.891667	0.108333	0.867063	0.734127
Average	0.832802	0.773855	0.866667	0.133333	0.829924	0.659849

	HSS	Brier_score	AUC
0	0.584344	0.109516	0.900684
1	0.597180	0.126414	0.876587
2	0.613821	0.112754	0.880291
3	0.832811	0.078500	0.940091
4	0.589603	0.121483	0.862517
5	0.805637	0.073118	0.941031
6	0.581218	0.135241	0.863337
7	0.705882	0.096465	0.918750
8	0.752381	0.077626	0.947192
9	0.740000	0.084602	0.934854
Average	0.680288	0.101572	0.906533




```
Average ROC AUC Score across folds: 0.90
```

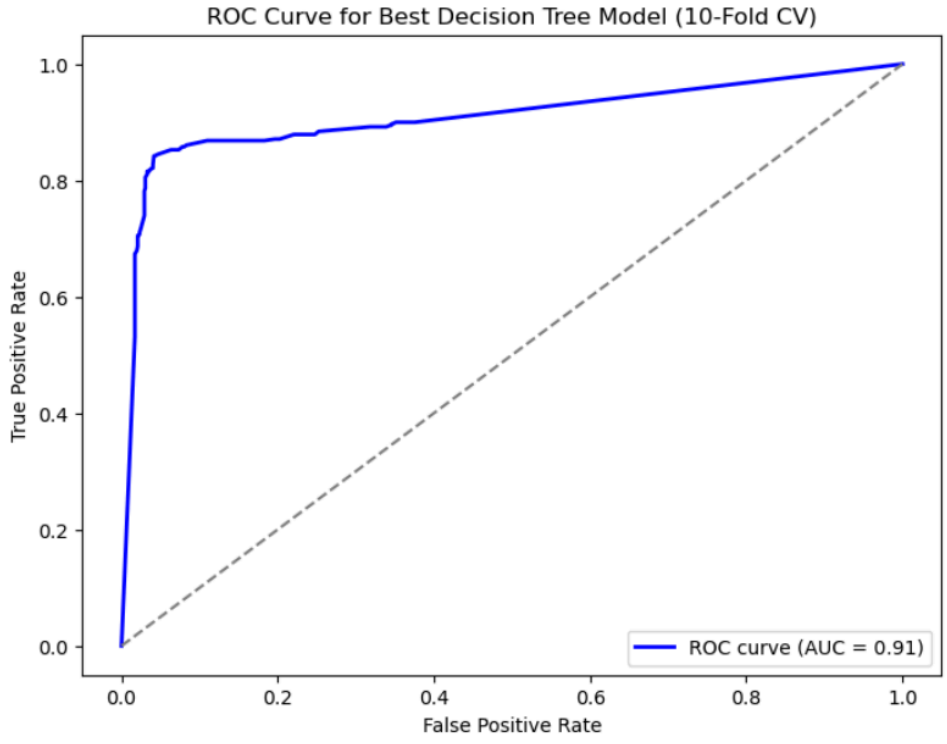
Decision tree :

```
Best Decision Tree Model: DecisionTreeClassifier(min_samples_leaf=10, random_state=42)
```

	Fold	TP	TN	FP	FN	TPR	TNR	FPR	FNR
0	1.0	32.0	78.0	5.0	5.0	0.864865	0.939759	0.060241	0.135135
1	2.0	28.0	80.0	3.0	9.0	0.756757	0.963855	0.036145	0.243243
2	3.0	30.0	80.0	4.0	6.0	0.833333	0.952381	0.047619	0.166667
3	4.0	27.0	84.0	3.0	6.0	0.818182	0.965517	0.034483	0.181818
4	5.0	29.0	78.0	8.0	5.0	0.852941	0.906977	0.093023	0.147059
5	6.0	35.0	78.0	1.0	6.0	0.853659	0.987342	0.012658	0.146341
6	7.0	36.0	73.0	3.0	8.0	0.818182	0.960526	0.039474	0.181818
7	8.0	33.0	77.0	3.0	7.0	0.825000	0.962500	0.037500	0.175000
8	9.0	31.0	76.0	2.0	11.0	0.738095	0.974359	0.025641	0.261905
9	10.0	33.0	83.0	1.0	3.0	0.916667	0.988095	0.011905	0.083333
Average	5.5	31.4	78.7	3.3	6.6	0.827768	0.960131	0.039869	0.172232

	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS
0	0.864865	0.864865	0.916667	0.083333	0.902312	0.804624
1	0.903226	0.823529	0.900000	0.100000	0.860306	0.720612
2	0.882353	0.857143	0.916667	0.083333	0.892857	0.785714
3	0.900000	0.857143	0.925000	0.075000	0.891850	0.783699
4	0.783784	0.816901	0.891667	0.108333	0.879959	0.759918
5	0.972222	0.909091	0.941667	0.058333	0.920500	0.841000
6	0.923077	0.867470	0.908333	0.091667	0.889354	0.778708
7	0.916667	0.868421	0.916667	0.083333	0.893750	0.787500
8	0.939394	0.826667	0.891667	0.108333	0.856227	0.712454
9	0.970588	0.942857	0.966667	0.033333	0.952381	0.904762
Average	0.905618	0.863409	0.917500	0.082500	0.893950	0.787899

	HSS	Brier_score	AUC
0	0.804624	0.085852	0.906219
1	0.754518	0.108841	0.849560
2	0.798387	0.074148	0.912533
3	0.806452	0.073073	0.907872
4	0.740173	0.095542	0.902702
5	0.866412	0.044389	0.965113
6	0.797794	0.094496	0.881579
7	0.807692	0.083098	0.893750
8	0.749518	0.092355	0.911020
9	0.919355	0.033688	0.974041
Average	0.804493	0.078548	0.910439




```
Average ROC AUC Score across folds: 0.91
```

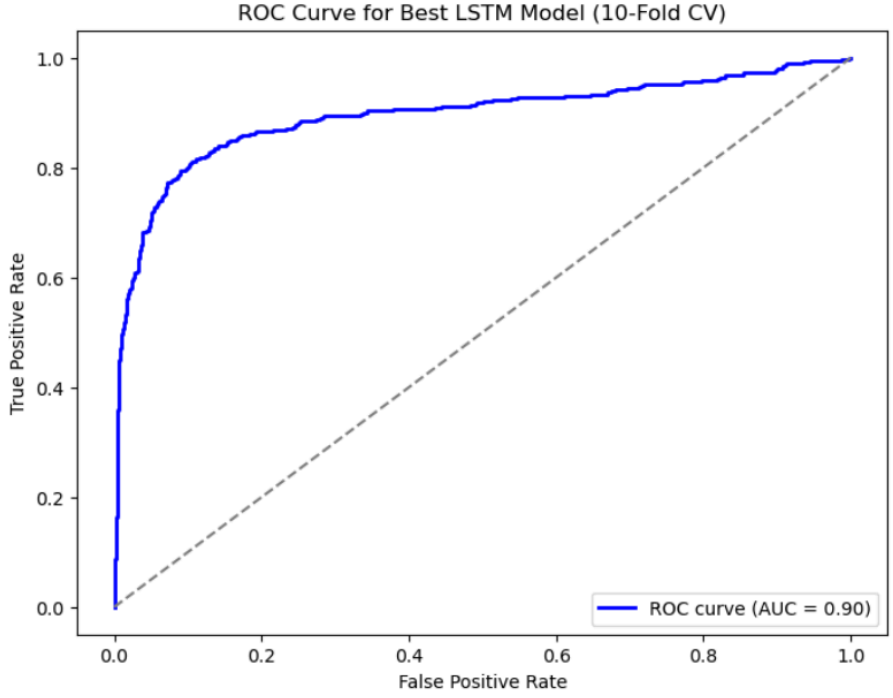
LSTM :

```
Best Hyperparameters: {'dropout_rate': 0.2, 'learning_rate': 0.01}
```

	Fold	TP	TN	FP	FN	TPR	TNR	FPR
0	1.0	29.0	107.0	5.0	9.0	0.763158	0.955357	0.044643
1	2.0	38.0	98.0	5.0	9.0	0.808511	0.951456	0.048544
2	3.0	30.0	96.0	10.0	14.0	0.681818	0.905660	0.094340
3	4.0	38.0	96.0	5.0	11.0	0.775510	0.950495	0.049505
4	5.0	39.0	91.0	9.0	11.0	0.780000	0.910000	0.090000
5	6.0	29.0	105.0	5.0	11.0	0.725000	0.954545	0.045455
6	7.0	34.0	103.0	5.0	8.0	0.809524	0.953704	0.046296
7	8.0	34.0	97.0	4.0	15.0	0.693878	0.960396	0.039604
8	9.0	33.0	93.0	7.0	17.0	0.660000	0.930000	0.070000
9	10.0	32.0	89.0	5.0	24.0	0.571429	0.946809	0.053191
Average	5.5	33.6	97.5	6.0	12.9	0.726883	0.941842	0.058158

	FNR	Precision	F1_measure	Accuracy	Error_rate	BACC
0	0.236842	0.852941	0.805556	0.906667	0.093333	0.859258
1	0.191489	0.883721	0.844444	0.906667	0.093333	0.879983
2	0.318182	0.750000	0.714286	0.840000	0.160000	0.793739
3	0.224490	0.883721	0.826087	0.893333	0.106667	0.863003
4	0.220000	0.812500	0.795918	0.866667	0.133333	0.845000
5	0.275000	0.852941	0.783784	0.893333	0.106667	0.839773
6	0.190476	0.871795	0.839506	0.913333	0.086667	0.881614
7	0.306122	0.894737	0.781609	0.873333	0.126667	0.827137
8	0.340000	0.825000	0.733333	0.840000	0.160000	0.795000
9	0.428571	0.864865	0.688172	0.806667	0.193333	0.759119
Average	0.273117	0.849222	0.781270	0.874000	0.126000	0.834362

	TSS	HSS	Brier_score	AUC
0	0.718515	0.744401	0.068393	0.939850
1	0.759967	0.777966	0.078973	0.908283
2	0.587479	0.603524	0.117225	0.852487
3	0.726005	0.749635	0.089074	0.897151
4	0.690000	0.696970	0.088698	0.931400
5	0.679545	0.713604	0.078814	0.949318
6	0.763228	0.780257	0.076340	0.921958
7	0.654274	0.694403	0.108398	0.896747
8	0.590000	0.621053	0.117546	0.864600
9	0.518237	0.556394	0.154080	0.829217
Average	0.668725	0.693821	0.097754	0.899101




```
Average ROC AUC Score across folds: 0.90
```

Out[27]:

	Random Forest	SVM	Decision Tree	LSTM
Fold	5.500000	5.500000	5.500000	5.500000
TP	31.800000	27.500000	31.400000	33.600000
TN	79.100000	76.500000	78.700000	97.500000
FP	2.900000	5.500000	3.300000	6.000000
FN	6.200000	10.500000	6.600000	12.900000
TPR	0.839218	0.726300	0.827768	0.726883
TNR	0.964933	0.933549	0.960131	0.941842
FPR	0.035067	0.066451	0.039869	0.058158
FNR	0.160782	0.273700	0.172232	0.273117
Precision	0.915974	0.832802	0.905618	0.849222
F1_measure	0.874556	0.773855	0.863409	0.781270
Accuracy	0.924167	0.866667	0.917500	0.874000
Error_rate	0.075833	0.133333	0.082500	0.126000
BACC	0.902075	0.829924	0.893950	0.834362
TSS	0.804151	0.659849	0.787899	0.668725
HSS	0.820409	0.680288	0.804493	0.693821
Brier_score	0.077631	0.101572	0.078548	0.097754
AUC	0.937331	0.906533	0.910439	0.899101

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

Total Scores for Each Model:
 Random Forest     6.0
SVM               23.0
Decision Tree     12.0
LSTM              19.0
dtype: float64

Best Model Overall: Random Forest
```

Other

The source code (.py and .pynb files) and data sets (.csv files) will be attached to the zip file.

Link to Git Repository

https://github.com/jr987-NJIT/RaviPrakash_Jyothsna_finalproj.git