NAME: Abhiram Krishnam
NJIT UCID: ak3236
Email Address: ak3236@njit.edu
Professor: Yasser Abduallah
CS 634 - Data Mining

FINAL PROJECT REPORT

ABSTRACT:

This project aims to create a machine learning model to predict the risk of stroke in individuals using health records and demographic data. The dataset included information such as age, medical history, and lifestyle factors. The goal was to determine the likelihood of a person experiencing a stroke based on these details. To achieve this, we used three different algorithms: Random Forest, K-Nearest Neighbors (KNN), and Long Short-Term Memory (LSTM) networks. The process began with cleaning and preparing the data, which involved filling in missing values, converting categorical data into a usable format, and scaling the features to ensure accurate predictions.

INTRODUCTION:

Stroke is a life-threatening condition that impacts millions of people globally, often leaving lasting disabilities or even leading to loss of life. Identifying those at higher risk is critical to ensuring timely interventions and preventive care. This project explored how machine learning can help predict stroke risk using a dataset of patient health records, which included factors like age, gender, hypertension, heart disease, and smoking habits.

To ensure reliable results, the models were tested using a rigorous 10-fold cross-validation process. Since the dataset was imbalanced, with fewer instances of stroke cases, a technique called SMOTE was used to create a more balanced dataset by generating additional examples of the minority class. Among the models tested, the Random Forest classifier, configured with 100 decision trees, stood out, achieving an impressive average accuracy of around 95%.

In addition to accuracy, the model's performance was evaluated using several metrics, including precision, recall, F1-score, and the AUC-ROC curve, which measures the model's ability to distinguish between stroke and non-stroke cases. Other metrics, like the Matthews Correlation Coefficient (MCC), True Positive and Negative Rates (TPR, TNR), False Positive and Negative Rates (FPR, FNR), and specialized scores such as the True Skill Score (TSS), Brier Score (BS), Brier Skill Score (BSS), and

Heidke Skill Score (HSS), were also calculated to ensure a comprehensive evaluation. This holistic approach underscores the potential of machine learning in making stroke risk prediction more accurate and actionable.

WORKFLOW:

1. **Data Preprocessing**

To ensure the dataset was ready for analysis:

- **Handling Missing Values**: Any gaps in the data were filled using a forward-fill method, maintaining the continuity of the information.
- **Encoding Categorical Variables**: Non-numerical data, like categorical features, were converted into numerical values using label encoding to make them suitable for machine learning models.
- **Feature Scaling**: Data was standardized and normalized using tools like StandardScaler and Normalizer, helping the models work more effectively by ensuring consistent scaling of features.

2. **Model Selection and Implementation**

Three different algorithms were chosen to approach the problem:

- **Random Forest Classifier**: A Random Forest model with 100 decision trees was used, specifically adjusted to handle the dataset's class imbalance.
- **K-Nearest Neighbor (KNN)**: The KNN algorithm was included as a simple, yet effective, method for stroke prediction.
- **Long Short-Term Memory (LSTM)**: An LSTM neural network, a specialized type of recurrent model, was implemented to capture patterns and relationships in time-sequenced health data.

3. **Evaluation and Validation**

To ensure reliability and accuracy:

- **Stratified 10-Fold Cross-Validation**: Each model was evaluated by splitting the dataset into 10 parts, training and testing on different subsets to achieve robust results.
- **Performance Metrics**: A wide range of metrics, including accuracy, precision, recall, F1-score, ROC AUC, and more, were calculated to measure each model's effectiveness in predicting stroke risk.

4. **Addressing Class Imbalance**

Since the dataset had fewer stroke cases compared to non-stroke cases:

- **SMOTE**: A technique called Synthetic Minority Over-sampling was used to balance the data by creating additional examples of the minority class, ensuring the models didn’t overlook stroke cases.

5. **Prediction and Results**

- **Predictions on New Data**: The trained models were used to assess stroke risk for unseen patient data, providing valuable insights for preventive measures.
- **Comparative Analysis**: The strengths and limitations of the three algorithms—Random Forest, KNN, and LSTM—were analyzed to understand which approach performed best for this task.

Required Python Libraries:

Before starting the project, ensure the following Python libraries are installed in your environment:

1) NumPy: Essential for numerical operations and array manipulation.
   pip install numpy

2) Pandas: Necessary for data manipulation and analysis.
   pip install pandas

3) Scikit-learn: Required for implementing machine learning algorithms and model evaluation.
   pip install scikit-learn
4) Imbalanced-Learn (imblearn): Useful for handling class imbalance in the dataset.
   pip install imbalanced-learn

5) Seaborn: Useful for data visualization and creating attractive statistical graphics.
   pip install seaborn

6) Matplotlib: Essential for creating plots and visualizations.
   pip install matplotlib

7) TensorFlow: Required for building and training deep learning models like LSTM.
   pip install tensorflow

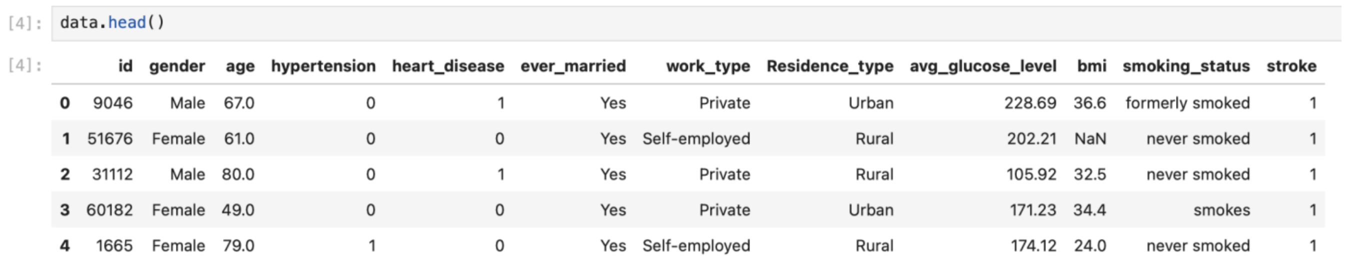
8) Keras: High-level neural networks API (usually installed with TensorFlow).
   pip install keras

9) Warnings: Used to suppress warnings during execution.
   pip install warning

10) Imblearn: For handling imbalanced datasets.
pip install imblearn

SCREENSHOTS:

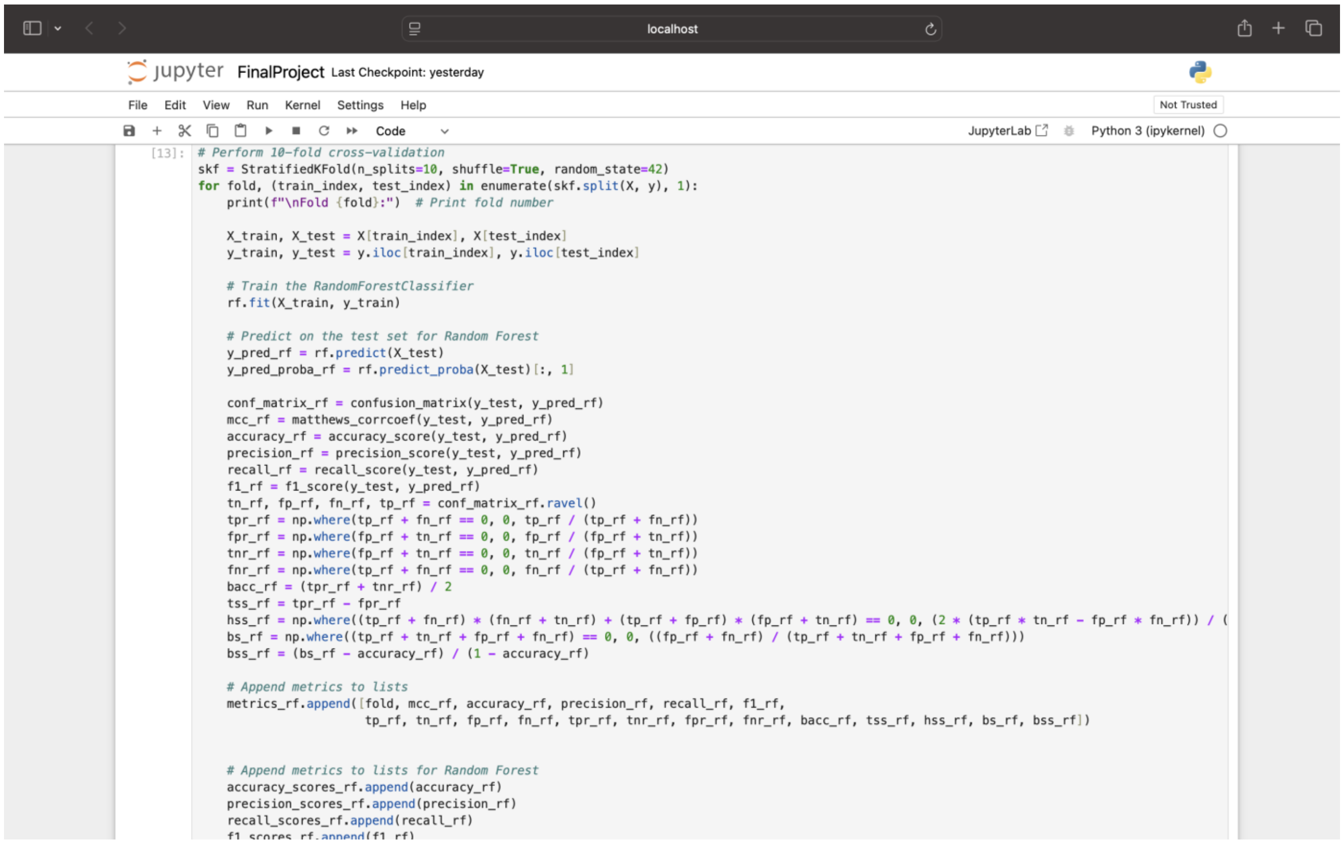
Sample CSV File-



```
[4]: data.head()
```

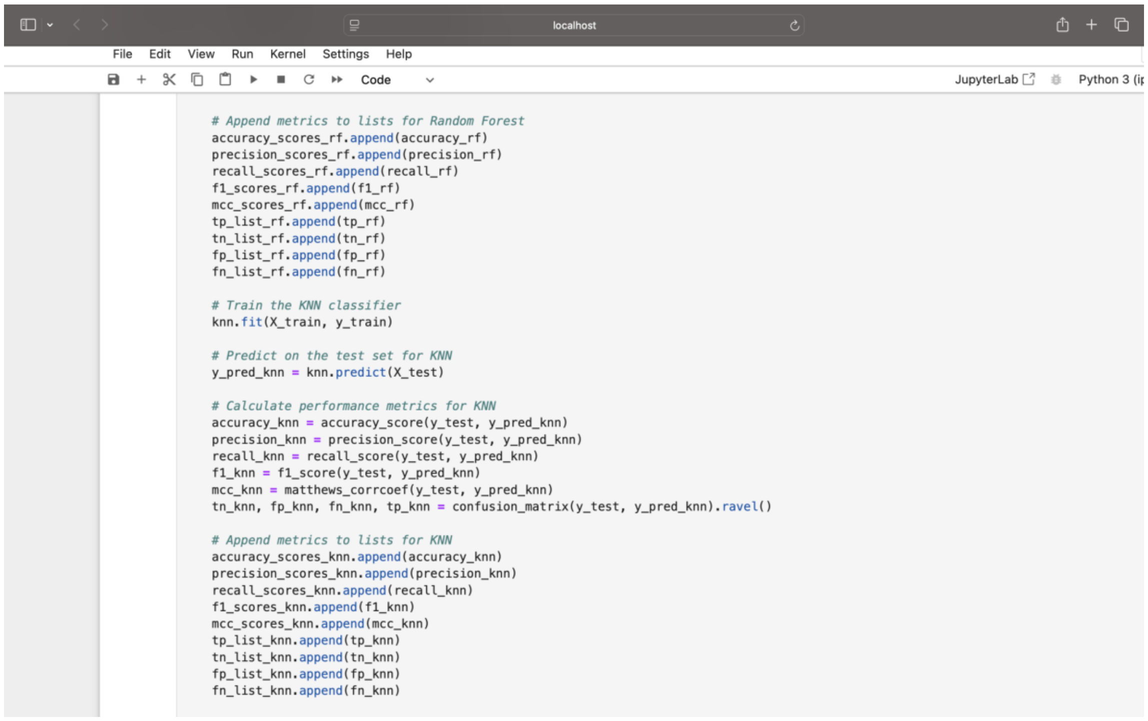
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Random Forest Implementation:


localhost
jupyter FinalProject Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Not Trusted
Code
JupyterLab
Python 3 (ipykernel)

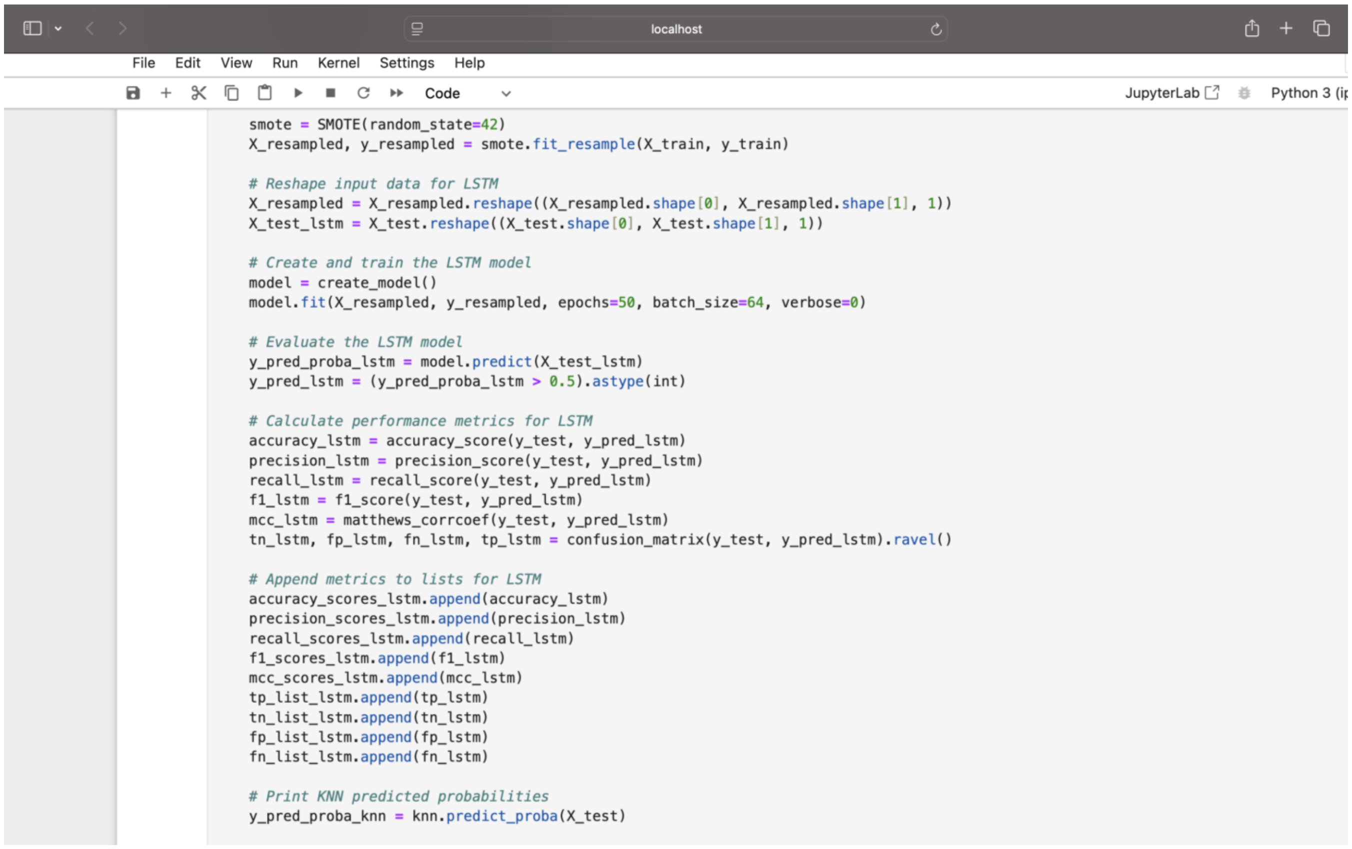
```
[13]: # Perform 10-fold cross-validation
      skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
      for fold, (train_index, test_index) in enumerate(skf.split(X, y), 1):
          print(f"\nFold {fold}:")  # Print fold number

          X_train, X_test = X[train_index], X[test_index]
          y_train, y_test = y.iloc[train_index], y.iloc[test_index]

          # Train the RandomForestClassifier
          rf.fit(X_train, y_train)

          # Predict on the test set for Random Forest
          y_pred_rf = rf.predict(X_test)
          y_pred_proba_rf = rf.predict_proba(X_test)[:, 1]

          conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
          mcc_rf = matthews_corrcoef(y_test, y_pred_rf)
          accuracy_rf = accuracy_score(y_test, y_pred_rf)
          precision_rf = precision_score(y_test, y_pred_rf)
          recall_rf = recall_score(y_test, y_pred_rf)
          f1_rf = f1_score(y_test, y_pred_rf)
          tn_rf, fp_rf, fn_rf, tp_rf = conf_matrix_rf.ravel()
          tpr_rf = np.where(tp_rf + fn_rf == 0, 0, tp_rf / (tp_rf + fn_rf))
          fpr_rf = np.where(fp_rf + tn_rf == 0, 0, fp_rf / (fp_rf + tn_rf))
          tnr_rf = np.where(fp_rf + tn_rf == 0, 0, tn_rf / (fp_rf + tn_rf))
          fnr_rf = np.where(tp_rf + fn_rf == 0, 0, fn_rf / (tp_rf + fn_rf))
          bacc_rf = (tpr_rf + tnr_rf) / 2
          tss_rf = tpr_rf - fpr_rf
          hss_rf = np.where((tp_rf + fn_rf) * (fn_rf + tn_rf) + (tp_rf + fp_rf) * (fp_rf + tn_rf) == 0, 0, (2 * (tp_rf * tn_rf - fp_rf * fn_rf)) / (
          bs_rf = np.where((tp_rf + tn_rf + fp_rf + fn_rf) == 0, 0, ((fp_rf + fn_rf) / (tp_rf + tn_rf + fp_rf + fn_rf)))
          bss_rf = (bs_rf - accuracy_rf) / (1 - accuracy_rf)

          # Append metrics to lists
          metrics_rf.append([fold, mcc_rf, accuracy_rf, precision_rf, recall_rf, f1_rf,
                             tp_rf, tn_rf, fp_rf, fn_rf, tpr_rf, tnr_rf, fpr_rf, fnr_rf, bacc_rf, tss_rf, hss_rf, bs_rf, bss_rf])


          # Append metrics to lists for Random Forest
          accuracy_scores_rf.append(accuracy_rf)
          precision_scores_rf.append(precision_rf)
          recall_scores_rf.append(recall_rf)
          f1_scores_rf.append(f1_rf)
```

KNN Implementation-



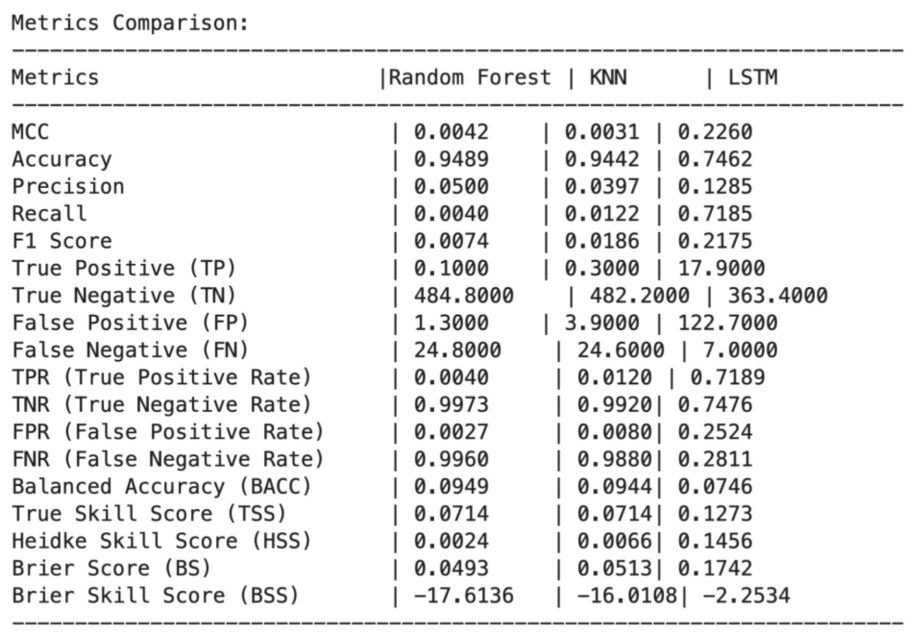
```
    # Append metrics to lists for Random Forest
    accuracy_scores_rf.append(accuracy_rf)
    precision_scores_rf.append(precision_rf)
    recall_scores_rf.append(recall_rf)
    f1_scores_rf.append(f1_rf)
    mcc_scores_rf.append(mcc_rf)
    tp_list_rf.append(tp_rf)
    tn_list_rf.append(tn_rf)
    fp_list_rf.append(fp_rf)
    fn_list_rf.append(fn_rf)

    # Train the KNN classifier
    knn.fit(X_train, y_train)

    # Predict on the test set for KNN
    y_pred_knn = knn.predict(X_test)

    # Calculate performance metrics for KNN
    accuracy_knn = accuracy_score(y_test, y_pred_knn)
    precision_knn = precision_score(y_test, y_pred_knn)
    recall_knn = recall_score(y_test, y_pred_knn)
    f1_knn = f1_score(y_test, y_pred_knn)
    mcc_knn = matthews_corrcoef(y_test, y_pred_knn)
    tn_knn, fp_knn, fn_knn, tp_knn = confusion_matrix(y_test, y_pred_knn).ravel()

    # Append metrics to lists for KNN
    accuracy_scores_knn.append(accuracy_knn)
    precision_scores_knn.append(precision_knn)
    recall_scores_knn.append(recall_knn)
    f1_scores_knn.append(f1_knn)
    mcc_scores_knn.append(mcc_knn)
    tp_list_knn.append(tp_knn)
    tn_list_knn.append(tn_knn)
    fp_list_knn.append(fp_knn)
    fn_list_knn.append(fn_knn)
```

LSTM Implementation:



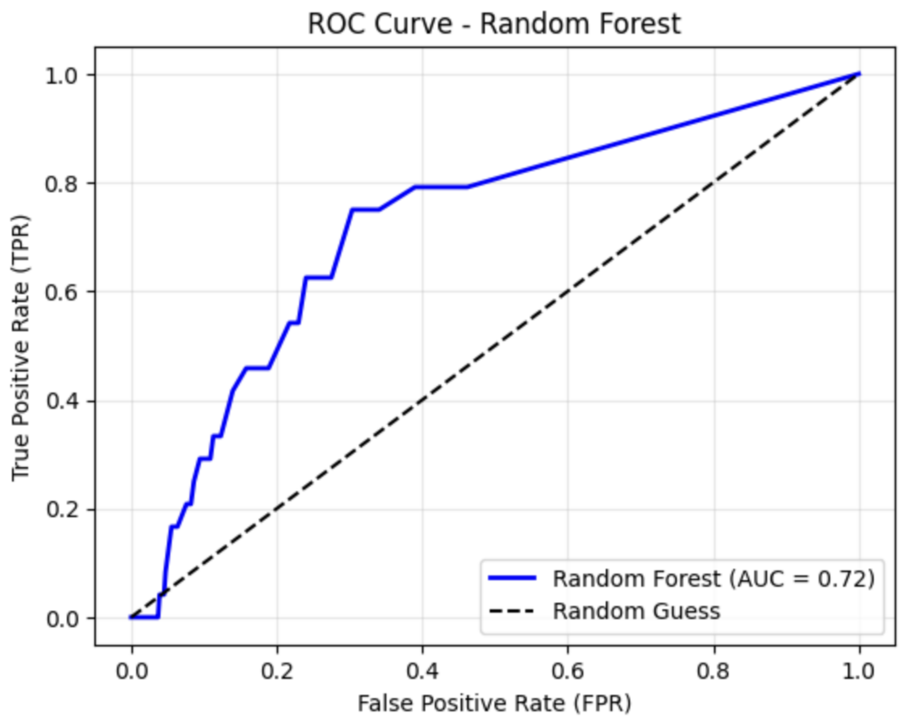
```python
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Reshape input data for LSTM
X_resampled = X_resampled.reshape((X_resampled.shape[0], X_resampled.shape[1], 1))
X_test_lstm = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Create and train the LSTM model
model = create_model()
model.fit(X_resampled, y_resampled, epochs=50, batch_size=64, verbose=0)

# Evaluate the LSTM model
y_pred_proba_lstm = model.predict(X_test_lstm)
y_pred_lstm = (y_pred_proba_lstm > 0.5).astype(int)

# Calculate performance metrics for LSTM
accuracy_lstm = accuracy_score(y_test, y_pred_lstm)
precision_lstm = precision_score(y_test, y_pred_lstm)
recall_lstm = recall_score(y_test, y_pred_lstm)
f1_lstm = f1_score(y_test, y_pred_lstm)
mcc_lstm = matthews_corrcoef(y_test, y_pred_lstm)
tn_lstm, fp_lstm, fn_lstm, tp_lstm = confusion_matrix(y_test, y_pred_lstm).ravel()

# Append metrics to lists for LSTM
accuracy_scores_lstm.append(accuracy_lstm)
precision_scores_lstm.append(precision_lstm)
recall_scores_lstm.append(recall_lstm)
f1_scores_lstm.append(f1_lstm)
mcc_scores_lstm.append(mcc_lstm)
tp_list_lstm.append(tp_lstm)
tn_list_lstm.append(tn_lstm)
fp_list_lstm.append(fp_lstm)
fn_list_lstm.append(fn_lstm)

# Print KNN predicted probabilities
y_pred_proba_knn = knn.predict_proba(X_test)
```

Result Comparison:

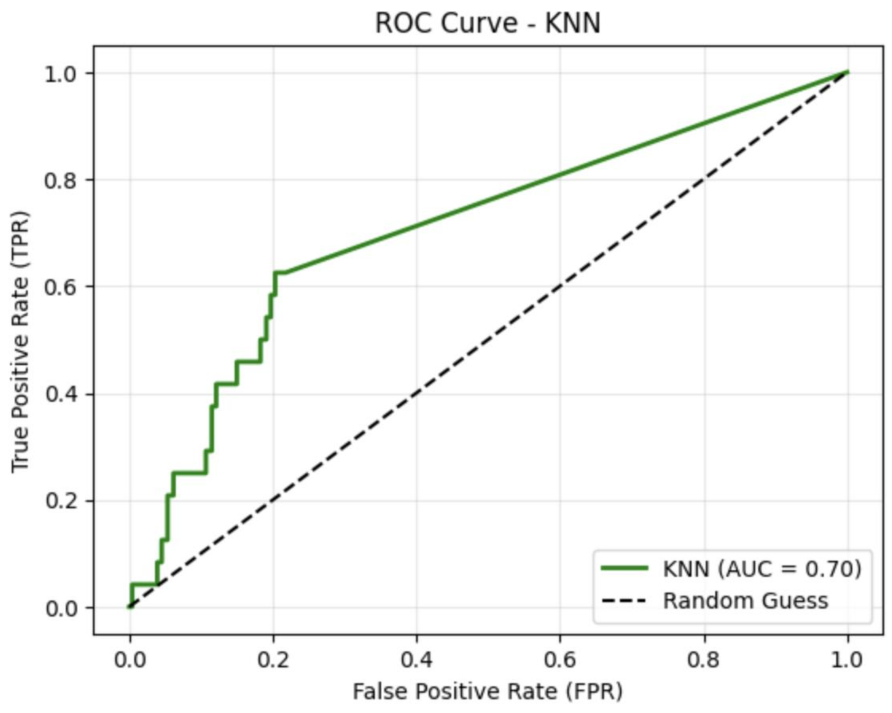


```
Metrics Comparison:
-----------------------------------------------------------------------
Metrics                      |Random Forest | KNN      | LSTM
-----------------------------------------------------------------------
MCC                           | 0.0042     | 0.0031 | 0.2260
Accuracy                      | 0.9489     | 0.9442 | 0.7462
Precision                     | 0.0500     | 0.0397 | 0.1285
Recall                        | 0.0040     | 0.0122 | 0.7185
F1 Score                      | 0.0074     | 0.0186 | 0.2175
True Positive (TP)            | 0.1000     | 0.3000 | 17.9000
True Negative (TN)            | 484.8000     | 482.2000 | 363.4000
False Positive (FP)           | 1.3000     | 3.9000 | 122.7000
False Negative (FN)           | 24.8000     | 24.6000 | 7.0000
TPR (True Positive Rate)      | 0.0040      | 0.0120 | 0.7189
TNR (True Negative Rate)      | 0.9973      | 0.9920| 0.7476
FPR (False Positive Rate)     | 0.0027      | 0.0080| 0.2524
FNR (False Negative Rate)     | 0.9960      | 0.9880| 0.2811
Balanced Accuracy (BACC)      | 0.0949      | 0.0944| 0.0746
True Skill Score (TSS)        | 0.0714      | 0.0714| 0.1273
Heidke Skill Score (HSS)      | 0.0024      | 0.0066| 0.1456
Brier Score (BS)              | 0.0493      | 0.0513| 0.1742
Brier Skill Score (BSS)       | -17.6136    | -16.0108| -2.2534
-----------------------------------------------------------------------
```

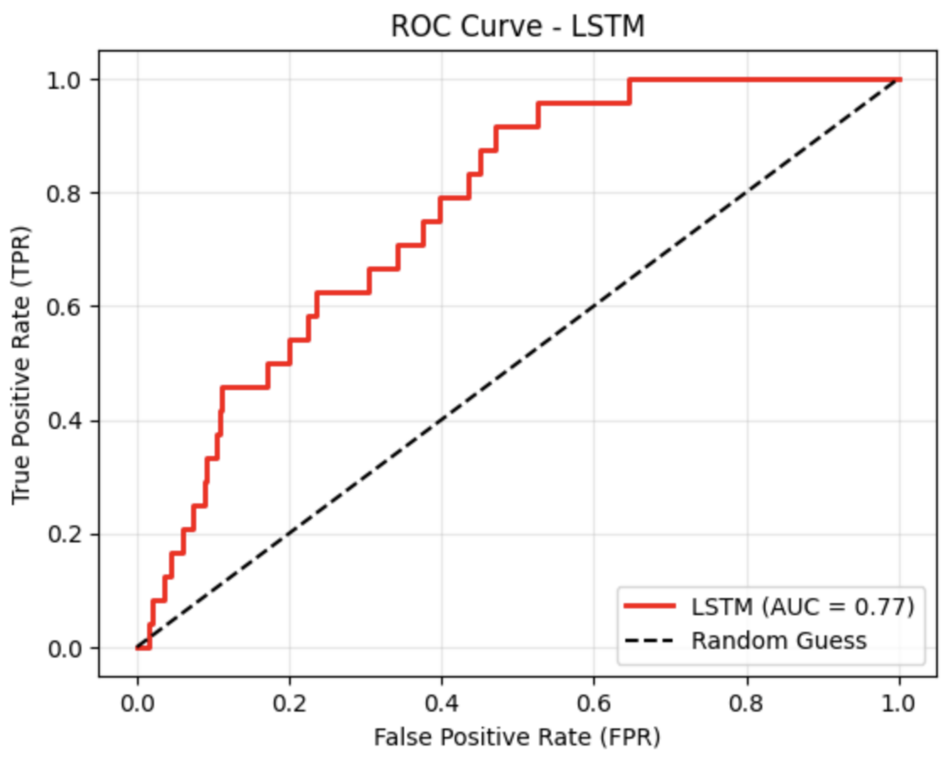
Random Forest ROC:



KNN ROC Curve:



LSTM ROC Curve:



CONCLUSION:

The results show significant differences in model performance for stroke risk prediction. Random Forest and KNN achieved high accuracy (94%) but performed poorly in detecting true stroke cases, with low recall and F1 scores, making them unsuitable for imbalanced datasets.

In contrast, LSTM excelled in identifying true positives, with a recall of 71.85% and a higher F1 score (0.2175), despite a lower overall accuracy of 74.62%. While it is better suited for detecting strokes, its higher false positive rate indicates the need for further optimization to improve precision.

GitHub Link: https://github.com/ak3236/FinalProject