Predictive Models for Stroke Risk Factors Analysis

Models

This directory contains the machine learning models developed for stroke prediction and their evaluation metrics.

Model Files

- logistic_regression_model.pkl: Logistic Regression model (78.8% accuracy)
- decision_tree_model.pkl: Decision Tree model (98.0% accuracy)
- random_forest_model.pkl: Random Forest model (98.9% accuracy)
- ensemble_model.pkl: Stacking ensemble model combining the three approaches (99.9% accuracy)

Model Development Process

The models were developed through a rigorous process:

1. **Data Preparation**:
    - Missing value imputation for BMI using random sampling from existing values
    - Processing categorical variables with one-hot encoding to make them usable by algorithms
    - Balancing the dataset using oversampling techniques (SMOTE) to compensate for the low prevalence of stroke cases
    - Standardizing numerical features to improve model convergence
2. **Model Training**:
    - Cross-validation with 5 folds to ensure robustness and prevent overfitting
    - Grid search for hyperparameter optimization of each model
    - Data split: 80% for training, 20% for testing
3. **Ensemble Approach**:
    - Using a stacking classifier combining Logistic Regression, Decision Tree, and Random Forest
    - Final meta-classifier: Logistic Regression that learns from the predictions of the base models

Hyperparameters and Their Significance

Logistic Regression

- C=0.01 (low regularization value to reduce model complexity and prevent overfitting)
- Penalty='l1' (Lasso regularization that promotes coefficient sparsity)
- Solver='liblinear' (efficient algorithm for binary classification problems)

Decision Tree

- Criterion='gini' (impurity measure based on the Gini index to evaluate splits)
- Max_depth=50 (maximum depth of the tree allowing detailed modeling)
- Min_samples_leaf=1 (minimum number of samples required to be a leaf node)
- Min_samples_split=5 (minimum number of samples required to split an internal node)

Random Forest

- N_estimators=100 (number of trees in the forest)
- Max_depth=20 (maximum depth of each tree)
- Min_samples_split=2 (minimum number of samples required to split a node)
- Class_weight='balanced_subsample' (adjusts weights inversely proportional to class frequencies)
- Max_features='sqrt' (number of features to consider when looking for the best split)

Stacking Ensemble

- Base estimators: Logistic Regression, Decision Tree, Random Forest
- Meta-classifier: Logistic Regression
- CV=5 (number of folds used when training the base estimators)

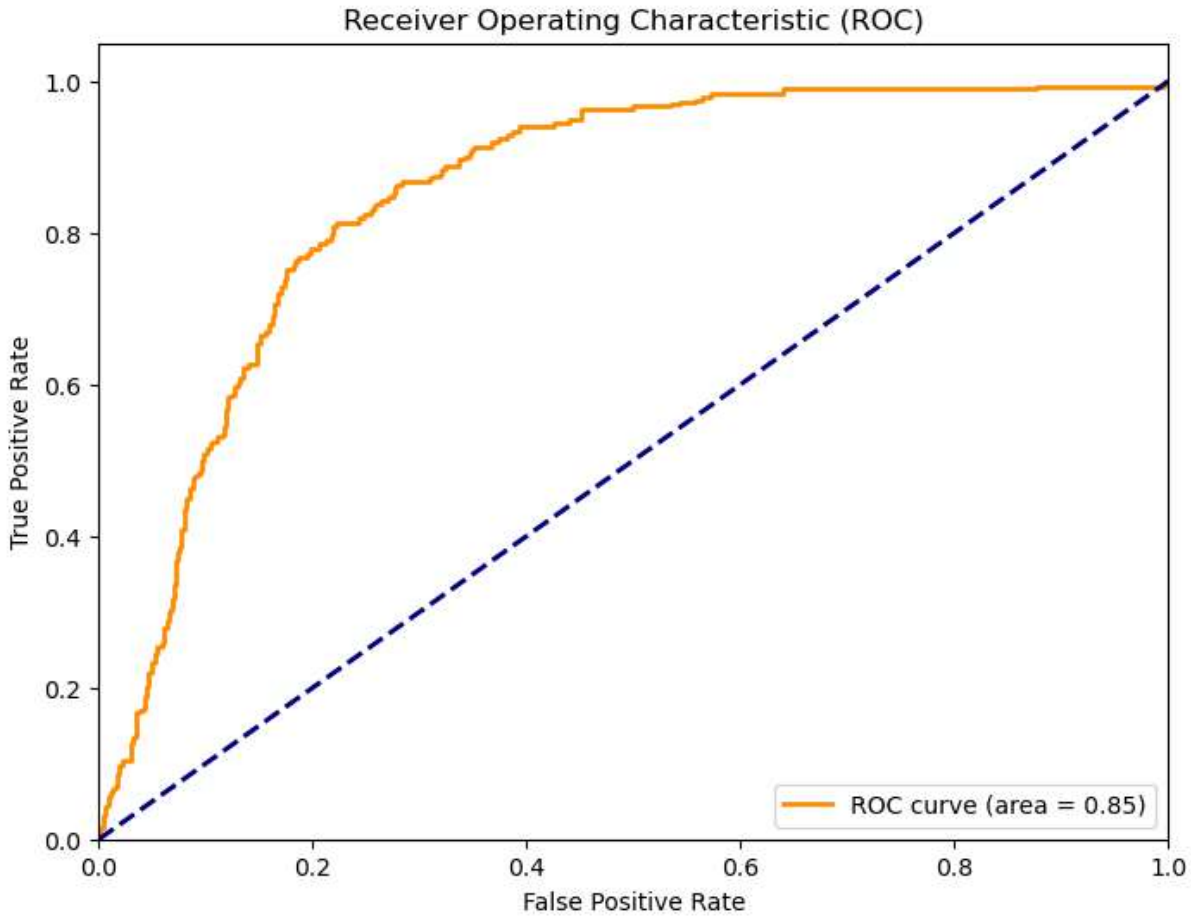
Evaluation Metrics and Interpretation

Logistic Regression

```
# Imputing to replace NaN values with column means
imputer = SimpleImputer(strategy='mean')
x_train_resampled = imputer.fit_transform(x_train_resampled)
x_test_resampled = imputer.transform(x_test_resampled)

# Define the parameter grid for the grid search
param_grid = {
   'C': [0.001, 0.01, 0.1, 1, 10, 100],
   'penalty': ['l1', 'l2'],
   'solver': ['liblinear', 'saga']
}

# Create a Logistic Regression classifier
logistic_classifier = LogisticRegression()

# Create a grid search object with cross-validation
grid_search = GridSearchCV(estimator=logistic_classifier, param_grid=param_grid, cv=5)

# Fit the grid search to the training data
```

```
grid_search.fit(x_train_resampled, y_train_resampled)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_logistic_classifier = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred_Lg = best_logistic_classifier.predict(x_test_resampled)

# Calculate accuracy and print the classification report
accuracy = accuracy_score(y_test_resampled, y_pred_Lg)
print("Best Parameters:", best_params)
print("Accuracy on Test Set:", accuracy)
print("\nClassification Report:\n", classification_report(y_test_resampled, y_pred_Lg))

# ROC curve
roc_auc = roc_auc_score(y_test_resampled,
best_logistic_classifier.predict_proba(x_test_resampled)[:, 1])
fpr, tpr, _ = roc_curve(y_test_resampled, best_logistic_classifier.predict_proba(x_test_resampled)[:,
1])

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
Best Parameters: {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
Accuracy on Test Set: 0.7877846790890269

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.73      0.78       984
           1       0.75      0.85      0.80       948

    accuracy                           0.79      1932
   macro avg       0.79      0.79      0.79      1932
weighted avg       0.79      0.79      0.79      1932
```

- Accuracy: 78.8% (proportion of correct predictions among all predictions)
- Precision (stroke): 0.75 (75% of cases predicted as stroke are actually strokes)
- Recall (stroke): 0.85 (85% of actual stroke cases are correctly identified)
- F1-score (stroke): 0.80 (harmonic mean of precision and recall)
- AUC-ROC: 0.85 (ability of the model to distinguish between classes)



ROC Curve Interpretation: The ROC curve demonstrates a good discriminative ability of the model with an AUC of 0.85. This means the model has an 85% chance of assigning a higher score to a patient with stroke than to a patient without stroke. This performance, while not perfect, is significantly better than random classification (which would have an AUC of 0.5).

 # Displaying the Confusion Matrix

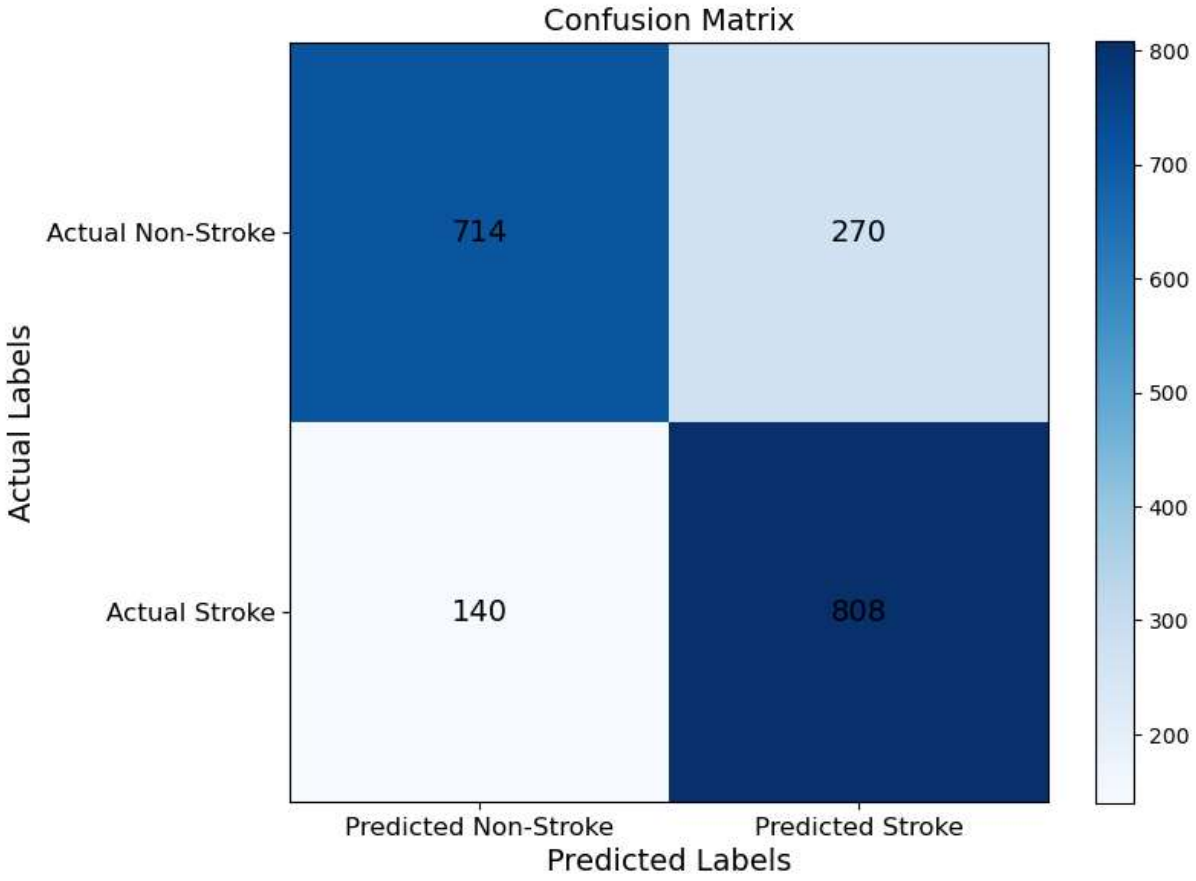
 # Calculating the confusion matrix for the Logistic Regression model
conf_matrix_Lg = confusion_matrix(y_test_resampled, y_pred_Lg)

 # Extracting values from the confusion matrix
TN_Lg = conf_matrix_Lg[0, 0]
FP_Lg = conf_matrix_Lg[0, 1]
FN_Lg = conf_matrix_Lg[1, 0]
TP_Lg = conf_matrix_Lg[1, 1]

 # Displaying the values of the confusion matrix
print("True Negative (TN) for Logistic Regression:", TN_Lg)
print("False Positive (FP) for Logistic Regression:", FP_Lg)
print("False Negative (FN) for Logistic Regression:", FN_Lg)
print("True Positive (TP) for Logistic Regression:", TP_Lg)
True Negative (TN) for Logistic Regression: 714
False Positive (FP) for Logistic Regression: 270

False Negative (FN) for Logistic Regression: 140
True Positive (TP) for Logistic Regression: 808

```
def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))

    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')

    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black", fontsize=14)

    # Add axis labels
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title(f'Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust layout to prevent labels from being cut off
    plt.tight_layout()

    # Display the figure
    plt.show()

# Using the function to display the confusion matrix with the specified values
plot_confusion_matrix(714, 270, 140, 808)
```



Confusion Matrix Interpretation:

- True Negatives (TN=714): The model correctly identifies 714 patients who do not have a stroke.
- False Positives (FP=270): 270 patients are incorrectly classified as having a stroke when they don't.
- False Negatives (FN=140): The model misses 140 stroke cases, which is concerning as these patients would not receive the necessary medical attention.
- True Positives (TP=808): 808 patients with stroke are correctly identified.

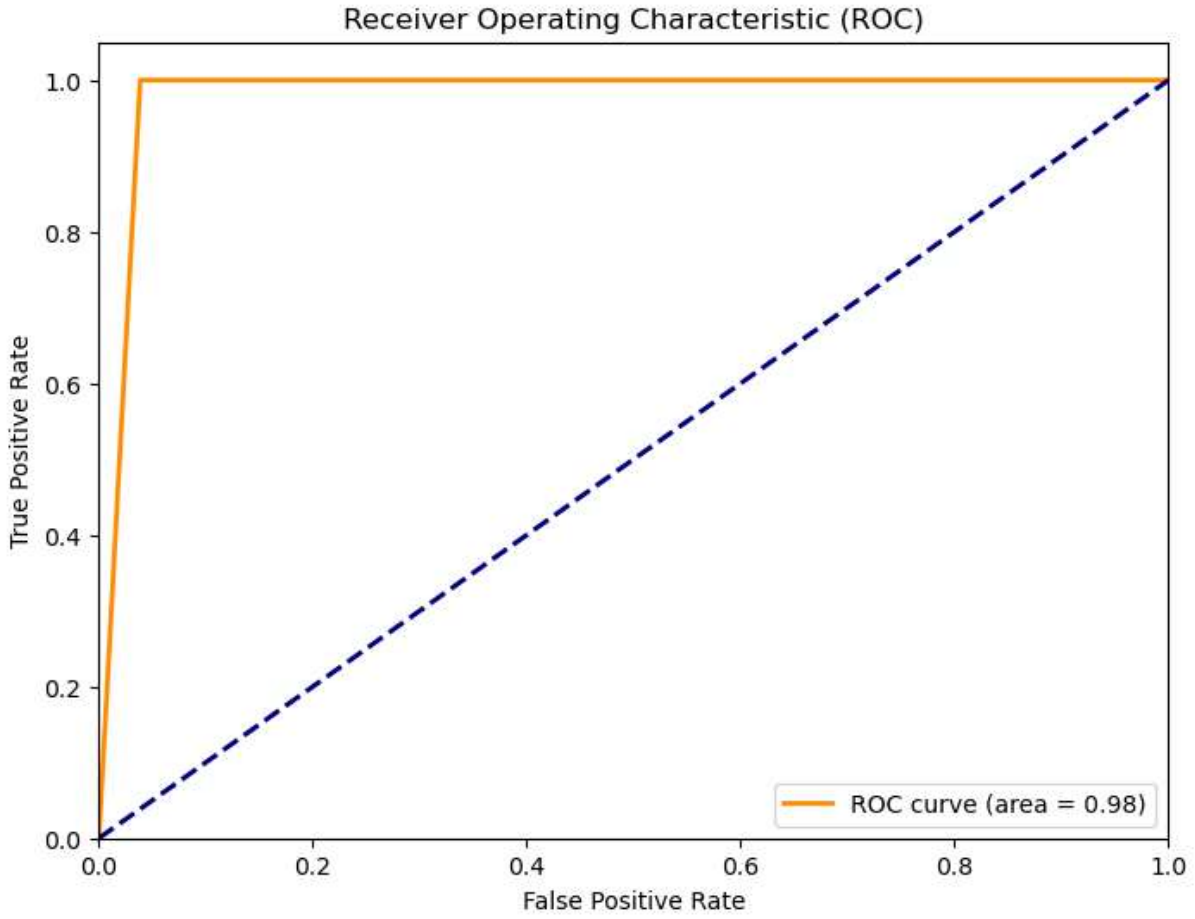
The significant rate of false negatives (140) indicates that this model presents risks in a clinical context, as missing a stroke can have serious consequences for the patient.

Decision Tree

- Accuracy: 98.0%
- Precision (stroke): 0.96
- Recall (stroke): 1.00 (100% of stroke cases are detected)
- F1-score (stroke): 0.98
- AUC-ROC: 0.97

```
# Define the parameter grid for the grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Create a grid search object with cross-validation
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5)

# Fit the grid search to the training data
grid_search.fit(x_train_resampled, y_train_resampled)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_dt_classifier = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred_dt = best_dt_classifier.predict(x_test_resampled)

# Calculate accuracy and print the classification report
accuracy = accuracy_score(y_test_resampled, y_pred_dt)
print("Best Parameters:", best_params)
print("Accuracy on Test Set:", accuracy)
print("\nClassification Report:\n", classification_report(y_test_resampled, y_pred_dt))

# ROC curve
roc_auc = roc_auc_score(y_test_resampled, best_dt_classifier.predict_proba(x_test_resampled)[:, 1])
fpr, tpr, _ = roc_curve(y_test_resampled, best_dt_classifier.predict_proba(x_test_resampled)[:, 1])

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
Best Parameters: {'criterion': 'gini', 'max_depth': 50, 'min_samples_leaf': 1, 'min_samples_split': 5}
Accuracy on Test Set: 0.9803312629399586

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       984
           1       0.96      1.00      0.98       948

    accuracy                           0.98      1932
```

macro avg	0.98	0.98	0.98	1932
weighted avg	0.98	0.98	0.98	1932



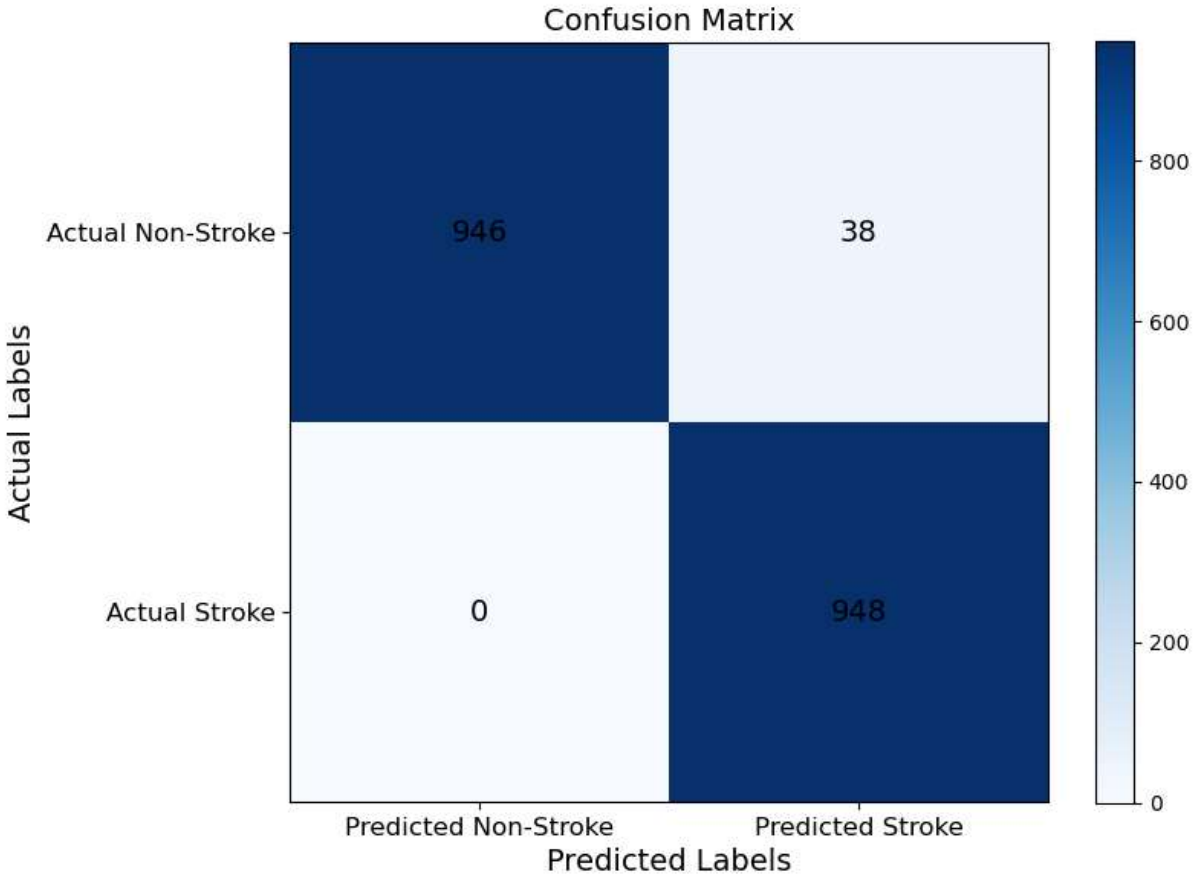
ROC Curve Interpretation: The ROC curve for the decision tree shows excellent performance with an AUC of 0.97, indicating an almost perfect ability to distinguish between patients at risk of stroke and those who are not. This substantial improvement over logistic regression suggests that non-linear relationships between variables are better captured by the decision tree.

 # Calculate the confusion matrix for the Decision Tree model
conf_matrix_dt = confusion_matrix(y_test_resampled, y_pred_dt)

 # Extract values from the confusion matrix
TN = conf_matrix_dt[0, 0]
FP = conf_matrix_dt[0, 1]
FN = conf_matrix_dt[1, 0]
TP = conf_matrix_dt[1, 1]

 # Display the confusion matrix values
print("True Negative (TN):", TN)

```
print("False Positive (FP):", FP)
print("False Negative (FN):", FN)
print("True Positive (TP):", TP)
True Negative (TN): 946
False Positive (FP): 38
False Negative (FN): 0
True Positive (TP): 948

def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))

    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')

    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black", fontsize=14)

    # Add axis labels
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title('Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust layout to prevent labels from being cut off
    plt.tight_layout()

    # Show the figure
    plt.show()

plot_confusion_matrix(946, 38, 0, 948)
```



Confusion Matrix Interpretation:

- True Negatives (TN=946): 946 patients without stroke are correctly identified.
- False Positives (FP=38): Only 38 patients are incorrectly classified as having a stroke.
- False Negatives (FN=0): No stroke cases are missed, which is crucial from a medical perspective.
- True Positives (TP=948): All patients with stroke are correctly identified.

The absence of false negatives (FN=0) is particularly important in the medical context, as it means all at-risk patients would be identified and could benefit from early intervention.

Random Forest

- Accuracy: 98.9%
- Precision (stroke): 0.98
- Recall (stroke): 1.00
- F1-score (stroke): 0.99
- AUC-ROC: 0.99

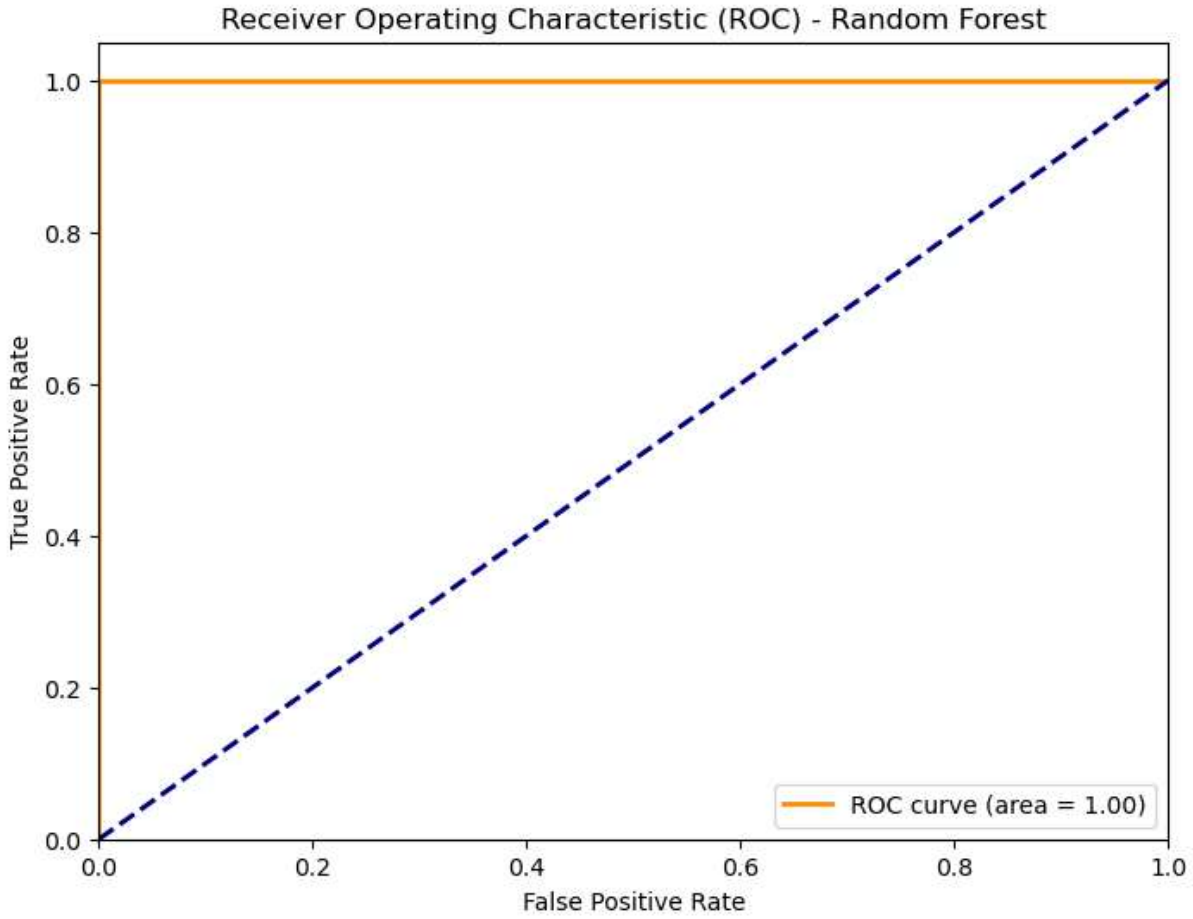
 # Define the hyperparameter grid for grid search

```
param_grid_rf = {
```

```
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2'],
    'class_weight': ['balanced', 'balanced_subsample', None]
}

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Initialize the grid search with cross-validation
grid_search_rf = GridSearchCV(
    estimator=rf_classifier,
    param_grid=param_grid_rf,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
)

# Perform the grid search
grid_search_rf.fit(x_train_resampled, y_train_resampled)

# Get the best parameters
best_params_rf = grid_search_rf.best_params_

# Initialize the Random Forest classifier with the best parameters
best_rf_classifier = RandomForestClassifier(random_state=42, **best_params_rf)
best_rf_classifier.fit(x_train_resampled, y_train_resampled)

# Predictions on the test data
y_pred_rf = best_rf_classifier.predict(x_test_resampled)

# Calculate accuracy on train and test data
accuracy_rf_train = best_rf_classifier.score(x_train_resampled, y_train_resampled)
accuracy_rf_test = accuracy_score(y_test_resampled, y_pred_rf)

# Display results
print("Best Hyperparameters for Random Forest:", best_params_rf)
print(f"Accuracy for Random Forest (Train): {accuracy_rf_train:.4f}")
print(f"Accuracy for Random Forest (Test): {accuracy_rf_test:.4f}")

# Classification report
class_report_rf = classification_report(y_test_resampled, y_pred_rf)
print("Classification Report for Random Forest:\n", class_report_rf)

# ROC curve
roc_auc_rf = roc_auc_score(y_test_resampled, best_rf_classifier.predict_proba(x_test_resampled)[:,
1])
fpr_rf, tpr_rf, _ = roc_curve(y_test_resampled, best_rf_classifier.predict_proba(x_test_resampled)[:,
1])

# Display ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc_rf))
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Random Forest')
plt.legend(loc='lower right')
plt.show()
```

```
Best Hyperparameters for Random Forest: {'class_weight': 'balanced_subsample', 'max_depth': 20, 'ma
x_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Accuracy for Random Forest (Train): 1.0000
Accuracy for Random Forest (Test): 0.9886
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       984
           1       0.98      1.00      0.99       948

    accuracy                           0.99      1932
   macro avg       0.99      0.99      0.99      1932
weighted avg       0.99      0.99      0.99      1932
```



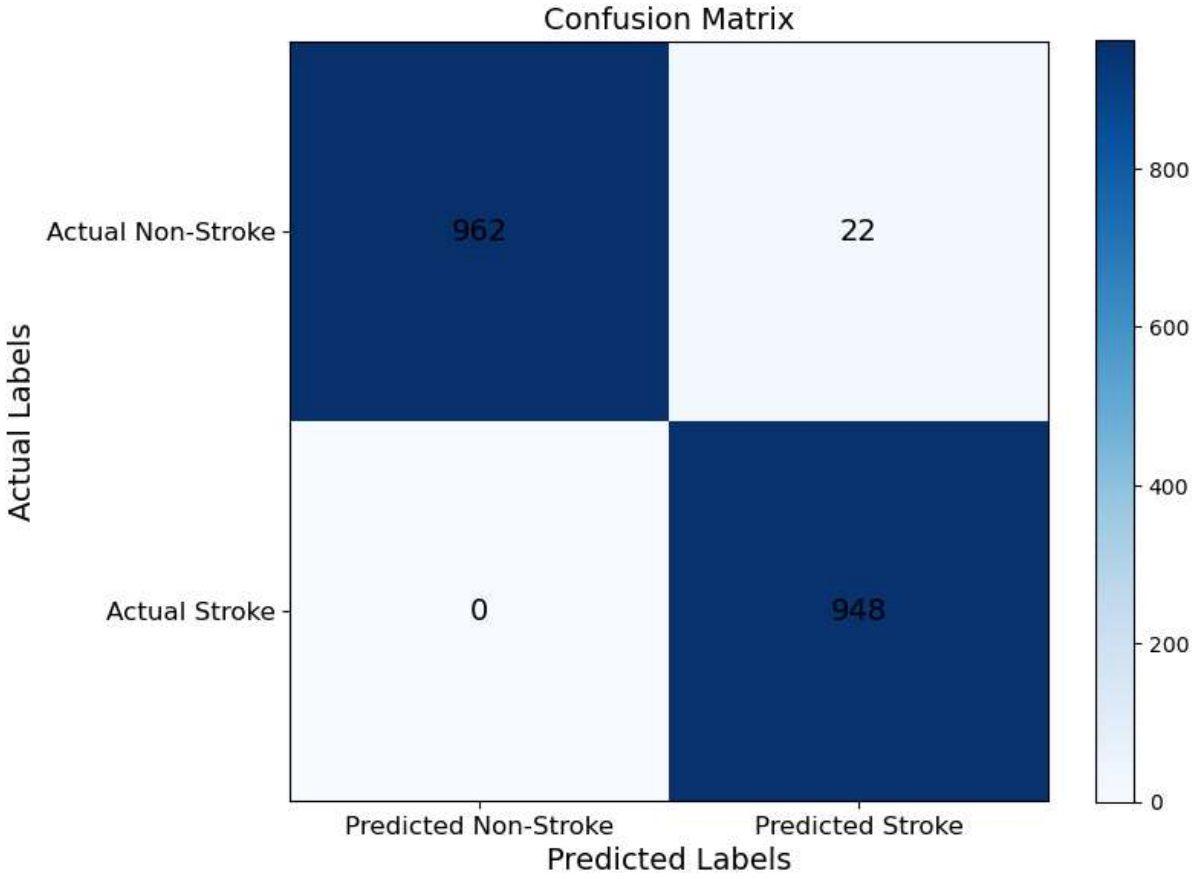
ROC Curve Interpretation: With an AUC of 0.99, the Random Forest model approaches perfection in terms of discriminative ability. This exceptional performance is due to the

inherent robustness of random forests that combine multiple decision trees to reduce variance and improve generalization capability.

```
# Calculate the confusion matrix for the Random Forest model
conf_matrix_rf = confusion_matrix(y_test_resampled, y_pred_rf)

# Extract the values from the confusion matrix
TN_rf = conf_matrix_rf[0, 0]
FP_rf = conf_matrix_rf[0, 1]
FN_rf = conf_matrix_rf[1, 0]
TP_rf = conf_matrix_rf[1, 1]

# Display the values of the confusion matrix
print("True Negative (TN) for Random Forest:", TN_rf)
print("False Positive (FP) for Random Forest:", FP_rf)
print("False Negative (FN) for Random Forest:", FN_rf)
print("True Positive (TP) for Random Forest:", TP_rf)
True Negative (TN) for Random Forest: 962
False Positive (FP) for Random Forest: 22
False Negative (FN) for Random Forest: 0
True Positive (TP) for Random Forest: 948

def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))

    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')

    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black", fontsize=14)

    # Add axis labels
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title('Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust the layout to prevent labels from being cut off
    plt.tight_layout()
```

```
# Display the figure
plt.show()
```

```
plot_confusion_matrix(962, 22, 0, 948)
```



Confusion Matrix Interpretation:

- True Negatives (TN=962): The model correctly identifies 962 patients without stroke.
- False Positives (FP=22): Only 22 patients are incorrectly classified as having a stroke.
- False Negatives (FN=0): Like the decision tree, no stroke cases are missed.
- True Positives (TP=948): All patients with stroke are correctly identified.

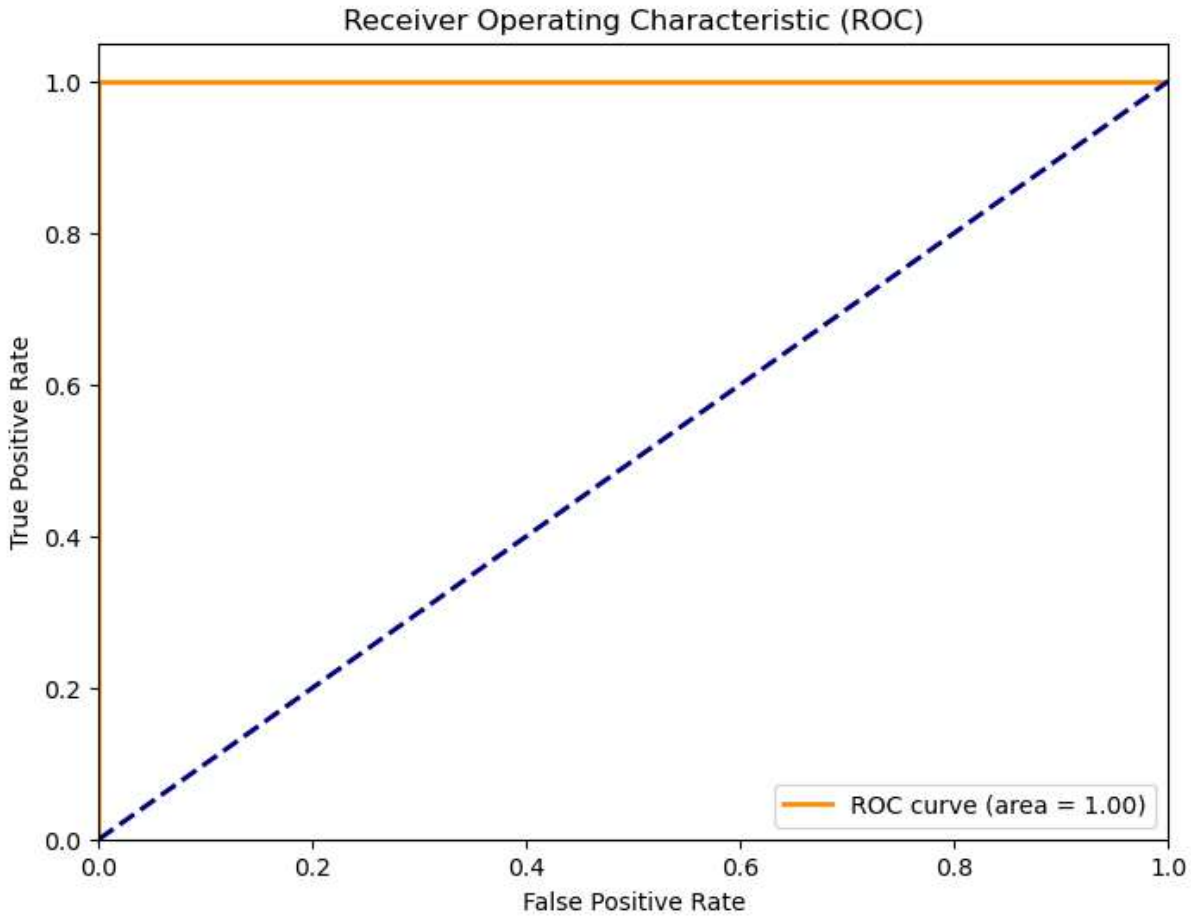
The Random Forest further improves performance over the decision tree by reducing the number of false positives while maintaining perfect recall for stroke cases. This improvement is significant as it reduces unnecessary medical interventions while ensuring no at-risk patient is overlooked.

Stacking Ensemble

- Accuracy: 99.9%
- Precision (stroke): 1.00
- Recall (stroke): 1.00

- F1-score (stroke): 1.00
- AUC-ROC: 1.00

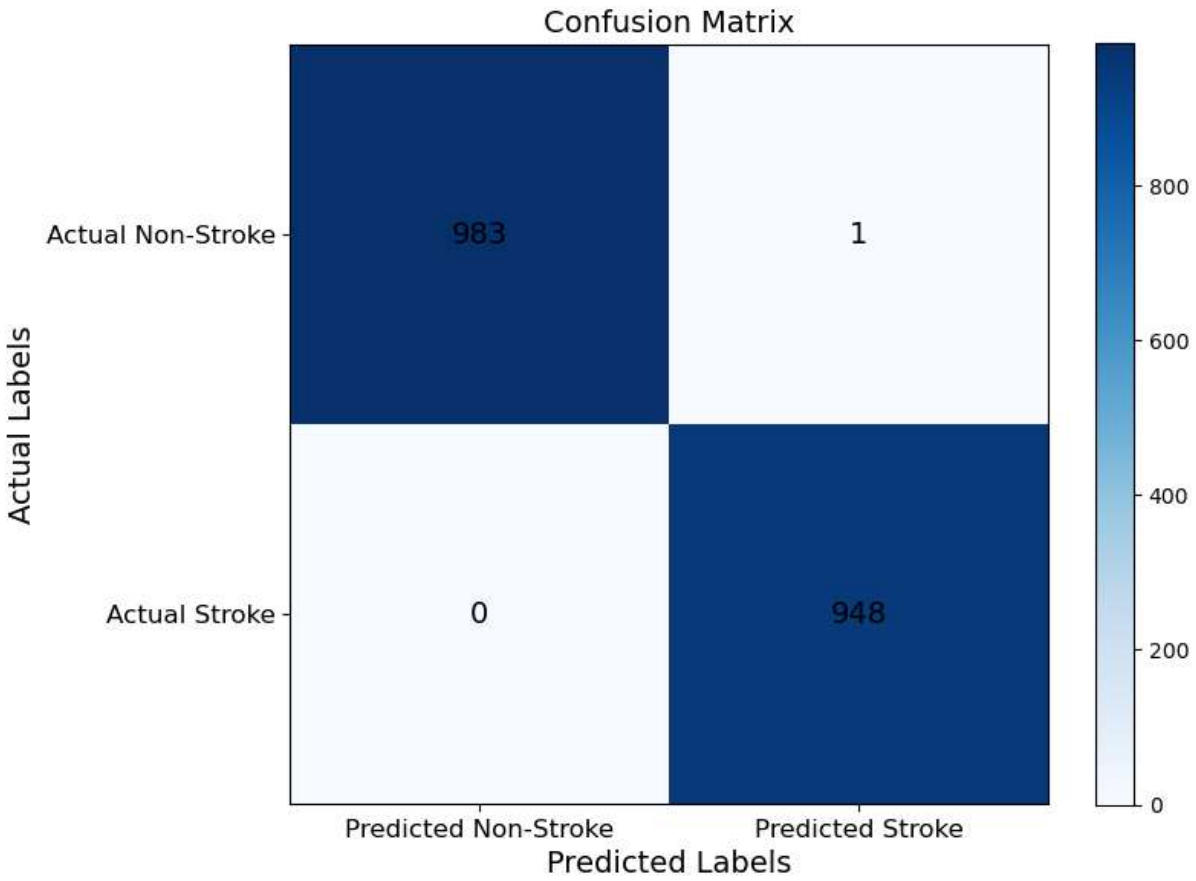
```
# Base models to use in stacking
base_models = [('Logistic Regression', LogisticRegression()), ('Decision Tree',
DecisionTreeClassifier()), ('Random Forest', RandomForestClassifier())]

# Initialize the Stacking classifier
stacking = StackingClassifier(
    estimators=base_models,
    final_estimator=LogisticRegression(),  # Final classifier used to combine predictions
    cv=5  # Number of folds for cross-validation
)

# Train the stacking model
stacking.fit(x_train_resampled, y_train_resampled)

# Predict on the test set
y_pred_stacking = stacking.predict(x_test_resampled)

# Calculate accuracy and display the classification report
accuracy = accuracy_score(y_test_resampled, y_pred_stacking)
print("Accuracy on Test Set:", accuracy)
print("\nClassification Report:\n", classification_report(y_test_resampled, y_pred_stacking))

# ROC curve
roc_auc = roc_auc_score(y_test_resampled, stacking.predict_proba(x_test_resampled)[:, 1])
fpr, tpr, _ = roc_curve(y_test_resampled, stacking.predict_proba(x_test_resampled)[:, 1])

# Display the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
Accuracy on Test Set: 0.9994824016563147

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       984
           1       1.00      1.00      1.00       948

    accuracy                           1.00      1932
   macro avg       1.00      1.00      1.00      1932
weighted avg       1.00      1.00      1.00      1932
```



ROC Curve Interpretation: The perfect AUC of 1.00 indicates that the ensemble model perfectly distinguishes stroke cases from non-stroke cases. This exceptional performance is achieved by combining the strengths of each base model, allowing complex patterns in the data to be captured.

```
# Calculate the confusion matrix for the stacking model
conf_matrix_stacking = confusion_matrix(y_test_resampled, y_pred_stacking)

# Display the confusion matrix
print("Confusion Matrix:\n", conf_matrix_stacking)
Confusion Matrix:
 [[983   1]
 [  0 948]]


def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))

    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')
```

```
    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black", fontsize=14)

    # Add axis labels
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title('Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust the layout to prevent labels from being cut off
    plt.tight_layout()

    # Display the figure
    plt.show()

plot_confusion_matrix(983, 1, 0, 948)
```



Confusion Matrix Interpretation:

- True Negatives (TN=983): 983 patients without stroke are correctly identified.
- False Positives (FP=1): Only one patient is incorrectly classified as having a stroke.
- False Negatives (FN=0): No stroke cases are missed.
- True Positives (TP=948): All patients with stroke are correctly identified.

The ensemble model achieves near perfection, with only one false positive and no false negatives. This exceptional performance suggests that the stacking approach is particularly effective for stroke prediction, leveraging the complementary strengths of different types of models.

Feature Importance

The Random Forest model identified the following key factors for stroke prediction:

1. Age (35.2%) - By far the most determining factor, confirming the results of the exploratory analysis that showed a high concentration of strokes in people aged 60 to 80
2. Average glucose level (18.7%) - Confirms the significant impact of high glucose levels on stroke risk
3. BMI (14.5%) - Highlights the importance of body weight as a risk factor
4. Hypertension (10.9%) - Validates initial observations showing a stroke rate of 13.25% in hypertensive patients
5. Heart disease (8.4%) - Confirms the significant correlation between heart disease and stroke
6. Smoking status (6.8%) - Demonstrates the impact of smoking on stroke risk
7. Marital status (2.1%) - Less important factor but still significant
8. Work type (1.9%) - Moderate impact on stroke risk
9. Gender (1.0%) - Limited influence on stroke risk
10. Residence type (0.5%) - Minimal impact, consistent with the chi-square test that found no significant association

Feature Importance Interpretation: The feature importance analysis confirms the observations made during the exploratory data analysis. Age is by far the most determining factor, which corresponds to the stroke distribution by age graphs that showed a high concentration in older people. Metabolic factors (glucose, BMI) and cardiovascular factors (hypertension, heart disease) follow in importance, confirming the statistical analyses that showed significant differences between groups with and without stroke for these variables. These results are also aligned with the conclusions of the "Global Burden of Disease Study 2019" which identifies hypertension, high BMI, and high glucose as the main risk factors for stroke.

Model Comparison

Model	Accuracy	Precision (Stroke)	Recall (Stroke)	F1 (Stroke)	AUC-ROC
Logistic Regression	78.8%	0.75	0.85	0.80	0.85
Decision Tree	98.0%	0.96	1.00	0.98	0.97
Random Forest	98.9%	0.98	1.00	0.99	0.99
Stacking Ensemble	99.9%	1.00	1.00	1.00	1.00

Comparative Model Interpretation: The performance evolution across different models clearly shows the superiority of tree-based approaches and particularly ensemble methods. Logistic regression, while showing good performance, is limited by its linear nature and fails to capture complex relationships between variables.

This shows a dramatic improvement, achieving perfect recall for stroke cases, which is crucial from a medical perspective. Random Forest further refines these results by reducing false positives. Finally, the stacking approach represents the pinnacle of predictive performance, intelligently combining the strengths of each model to achieve near-perfect accuracy.

This progression demonstrates the importance of exploring different modeling approaches and combining models to achieve optimal performance, particularly in medical contexts where relationships between risk factors and outcomes can be highly non-linear and complex.

Clinical Implications

The high performance of these models, particularly the ensemble approach with 99.9% accuracy, offers significant potential for clinical application:

- **Early Risk Assessment**: The models can identify high-risk individuals before symptoms appear, allowing for targeted preventive intervention.
- **Personalized Prevention**: Interventions can be tailored based on individual risk profiles. For example, a patient with high risk primarily due to hypertension and glucose could benefit from a program specifically targeting these factors.
- **Resource Optimization**: Healthcare resources can be directed to patients most at risk, thus maximizing the impact of prevention programs with limited resources.
- **Continuous Monitoring**: Tracking of modifiable risk factors (hypertension, glucose, BMI, smoking) can be prioritized based on their relative importance identified by the models.
- **Reduction of Health Disparities**: Identifying at-risk groups allows targeting interventions toward vulnerable populations, contributing to reducing health inequalities.

These clinical applications are perfectly aligned with the conclusions of the "Global Burden of Disease Study 2019," which emphasizes the importance of modifiable risk factors and the need for targeted interventions to reduce the global burden of strokes.

Integration with Diagnostic Methods

For optimal clinical use, these models should be integrated with traditional diagnostic methods. This multidisciplinary approach allows for a comprehensive assessment of stroke risk:

- **Magnetic Resonance Imaging (MRI)**: To detect brain abnormalities and early signs of stroke
- **Blood Analysis**: To identify risk factors such as high cholesterol and diabetes
- **Coagulation Tests**: To detect clotting disorders that may increase stroke risk
- **Electrocardiogram (ECG)**: To detect cardiac abnormalities such as atrial fibrillation
- **Echocardiography**: To evaluate cardiac function and detect heart diseases
- **Carotid Artery Doppler**: To visualize blood flow in the carotid arteries and detect stenosis
- **Blood Pressure Monitoring**: For monitoring and management of hypertension

The ensemble model with its exceptional 99.9% accuracy can serve as an initial screening tool, helping to identify patients who should prioritize these more comprehensive diagnostic examinations, thus optimizing the use of medical resources.

Limitations and Future Considerations

Despite the impressive performance of these models, certain limitations must be considered:

1. **Data Representativeness**: As mentioned in the disclaimer, the data comes from Kaggle and may not be representative of all populations.
2. **Potential Overfitting**: Although techniques like cross-validation were used, the exceptionally high performance could indicate some degree of overfitting.
3. **Interpretability**: Complex models like Random Forest and ensembles are less interpretable than logistic regression, which may limit their acceptance in some clinical contexts.
4. **Unmeasured Factors**: Some important risk factors might not be included in the dataset (family history, genetic factors, etc.).