REPORT

1. **Objective Definition**:
    - Develop a machine learning model to predict the presence of a disease using the provided dataset.

2. **Dataset Description**:
    - Two datasets provided:

    a) **Training Dataset (Disease_train.csv)**: The training dataset contains 4000 entries with 12 columns. There are 10 feature columns of type float64 and 2 columns of type int64 (patient _id and diagnosis).

    b) **Test Dataset (Disease_test.csv)**: The test dataset contains 1000 entries with 11 columns. Like the training dataset, it consists of 10 feature columns of type float64 and 1 column of type int64 (patient _id)

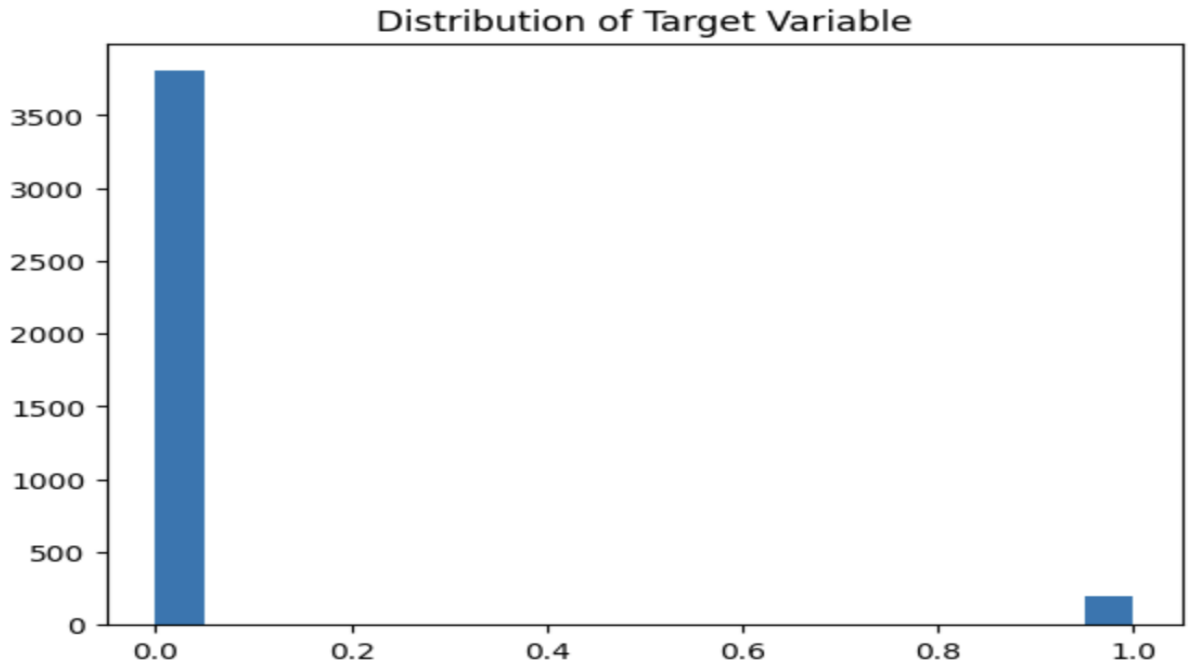
Data Preprocessing:

3. **Handling Missing Values**:
    - The missing value percentages for each column in the training dataset are displayed using the is null (). Mean () function. All columns have 0% missing values, indicating that there are no missing values in any of the features or target variable.

```
1 # Display the missing value percentages
2 print(train_df.isnull().mean())
```

```
feature_1     0.0
feature_2     0.0
feature_3     0.0
feature_4     0.0
feature_5     0.0
feature_6     0.0
feature_7     0.0
feature_8     0.0
feature_9     0.0
feature_10    0.0
patient_id    0.0
diagnosis     0.0
dtype: float64
```

4. **Plotting Histogram:** Visualizing the target variable distribution through a histogram highlights a significant class imbalance in the dataset.



- **Imbalanced Classes**: The target variable (likely representing the presence or absence of a disease) exhibits a significant class imbalance.
- **Majority Class (0):** This class comprises most instances, with over 3500 occurrences. It likely represents the 'healthy' or 'no disease' class.
- **Minority Class (1)**: This class has a much lower frequency, with fewer than 200 instances. It likely represents the 'diseased' or 'positive' class.

5. **Extracting Features and Target Variables**:

- Features (X) are extracted from the training data by dropping the 'patient _id' and 'diagnosis' columns.
- The target variable (y) is extracted as the 'diagnosis' column.

```
# Extract features and target from training data
X = train_df.drop(['patient_id', 'diagnosis'], axis=1)
y = train_df['diagnosis']

# Drop 'patient_id' from test set
X_test = test_df.drop('patient_id', axis=1)
```

6. **Standardizing the Features**:

- The features are standardized using a scaler to ensure uniformity and improve model performance.

```
# Standard scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X)
X_test_scaled = scaler.transform(X_test)
```

7. **Applying Borderline-SMOTE to Handle Class Imbalance**:

- Borderline-SMOTE is applied to address the significant class imbalance in the target variable. This oversampling technique generates synthetic samples for the minority class while avoiding the risk of overfitting.

```
# Apply BorderlineSMOTE to handle class imbalance
borderline_smote = BorderlineSMOTE(random_state=42)
X_resampled, y_resampled = borderline_smote.fit_resample(X_train_scaled, y)
```

8. **Splitting Resampled Data into Training and Validation Sets**:

- The resampled data is split into training and validation subsets to facilitate model development and evaluation.

```
# Split the resampled data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

9. **Setting Parameter Grid for Hyperparameter Tuning**:

- A parameter grid is defined to specify the hyperparameters for optimization during model training. This grid typically includes parameters like the number of estimators, maximum depth, minimum samples split, and leaf nodes, among others.

- **Training the RandomForestClassifier with GridSearchCV**:

The RandomForestClassifier is trained using GridSearchCV, which performs an exhaustive search over the specified parameter grid to find the best combination of hyperparameters.

```
# Set the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)

# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3, scoring='roc_auc', n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(f"Best parameters found: {best_params}")
```

10. Training the Optimized RandomForestClassifier:

- The optimized RandomForestClassifier is trained using the best hyperparameters obtained from the GridSearchCV process.
- **Evaluating the Model on the Validation Set**:
  The performance of the optimized model is evaluated on the validation set using metrics such as ROC-AUC score, accuracy, precision, recall, and F1 score.
- **Making Predictions on the Test Dataset**:
  The trained model is used to make predictions on the test dataset to assess its performance on unseen data.

```
# Initialize and train the optimized RandomForestClassifier
optimized_rf_model = RandomForestClassifier(**best_params, random_state=42)
optimized_rf_model.fit(X_train, y_train)

# Evaluate the optimized model on the validation set
y_val_pred = optimized_rf_model.predict(X_val)
roc_auc = roc_auc_score(y_val, y_val_pred)
print(f' ROC-AUC Score with Optimized RandomForestClassifier: {roc_auc}')

# Calculate other metrics
accuracy = accuracy_score(y_val, y_val_pred)
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred)
conf_matrix = confusion_matrix(y_val, y_val_pred)

# Print the metrics
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'Confusion Matrix: \n{conf_matrix}')

# Create a DataFrame with actual and predicted values
validation_results = pd.DataFrame({
    'actual': y_val,
    'predicted': y_val_pred
})

# Print the DataFrame
print("\nValidation Actual vs Predicted:")
print(validation_results.head())

# Make predictions on the test dataset
test_predictions = optimized_rf_model.predict(X_test_scaled)
```

11. Saving the Predictions to a CSV File:

- The predictions made on the test dataset are saved to a CSV file named according to the specified format. This file contains two columns: 'patient _id' and 'prediction', representing the unique identifiers of patients and their corresponding predicted outcomes, respectively.

```
# Make predictions on the test dataset
test_predictions = optimized_rf_model.predict(X_test_scaled)

# Create a DataFrame with the predictions
predictions_df = pd.DataFrame({
    'patient_id': test_df['patient_id'],
    'prediction': test_predictions
})

# Print the predictions
print("\nTest Predictions:")
print(predictions_df.head())

# Save the predictions to a CSV file
output_path = '/content/drive/MyDrive/SE22UCSE193_predictions.csv'
predictions_df.to_csv(output_path, index=False)

# Confirm the file is saved
if os.path.exists(output_path):
    print(f"File saved successfully at: {output_path}")
else:
    print("File not found.")
```

12. Cross-validation is employed to assess the performance and generalization ability of the model. It involves partitioning the dataset into multiple subsets, performing training and evaluation iteratively, and then aggregating the results. In this case, we use 5-fold Stratified Cross-Validation, which ensures that each fold maintains the same class distribution as the original dataset. The model is trained and evaluated five times, with each fold serving as the validation set once. Finally, the average performance metrics, including accuracy, precision, recall, F1 score, and ROC-AUC score, are computed and reported for an overall assessment of the model's performance.

Conclusion:

13. Summary:

- Successfully developed a machine learning model for disease prediction.
- Achieved the objective of predicting disease presence based on patient features.

RESULTS:

METRIC	VALIDATIONSET	CROSS-VALIDATION
ROC-AUC Score	0.9776	0.9482
Accuracy	0.9777	0.9482
Precision	0.9813	0.9287
Recall	0.9735	0.9713
F1 Score	0.9774	0.9495