31250 Introduction to Data Analytics

Assignment 3: Data Mining in Action

DEAN STOKELD - 24923029

UTS Faculty of Engineering and Information Technology

TABLE OF CONTENTS

1A Introduction

1. Data Mining Problem

This assignment is a continuation of the previous assignment, where the first assignment involved data exploration of a dataset whereas this assignment will dive in data mining. In this assignment, the perspective will be from a data scientist at a consultation company with the goal of predicting a missing target variable in dataset (unknown data) by training several models on a dataset that includes the target variable (known data), then selecting the best model using several metrics.

To accomplish this task, Excel will be used for initial data exploration and understanding. Python will then be used to continue data processing then train models on known data then test it on unknown data.

2. Data Goal & Strategy

The main goal of this report is to classify the unknown data using Python using several models. The best performing model will be selected and several techniques will be applied to the data and model to increase the accuracy score of the model on the unknown data. This is an iterative process and finding the correct techniques and configurations will enhance the accuracy of the model.

The strategy used to solve this problem will be to use Python and preprocess the data by cleaning (handling missing values), splitting into train and test sets then encoding the data so it can be used for machine learning models. These models will then be trained and the best performing model(s) will be selected the to perform additional tuning on then submit the results.

3. Data Provided & Submission

The data for this assignment was given through a competition posting on Kaggle.

There are 3 datasets provided on this Kaggle posting:

- Assignment3-Healthcare-Dataset.csv

  This dataset includes the known data and will be the dataset used to train the models on.

- Assignment3-Unknown-Dataset.csv

  This dataset includes the unknown data and will be the dataset used to test the trained models on.

- Assignment3-Kaggle-Submission-Random-Sample.csv

  This dataset is a sample submission of the predictions for the unknown dataset. It is required that the predictions the models produce follow this format for successful Kaggle submissions.

1B Data Preprocessing & Explanation

1. Analysis via Excel

Firstly, Assignment3-Healthcare-Dataset.csv file was opened in Excel for initial data exploration and understanding. Through the filter feature in Excel, it was observed that there were many missing data points in several rows and columns, with the LOSgroupNum column missing many data points - LOSgroupNum are assigned groups 1 to 4 based on total LOSdays (length of stay) of the patient. There was also missing data in 'marital_status', 'NumCallouts', 'age', 'religion', and 'AdmitDiagnosis'.

This process was repeated for the Assignment3-Unknown-Dataset.csv file and found the same missing data trends. Therefore, it was decided that any preprocessing done to the Assignment3-Healthcare-Dataset.csv dataset will also be performed on the Assignment3-Unknown-Dataset.csv dataset, which is common practice.

2. Data Exploration & Imputing Missing Data via Python

Before data exploration and imputation using Python, the .csv files were imported into a Python environment. Using a Jupyter Notebook through the Anaconda environment, pandas was imported and the read_csv() function was used to import both training and unknown data. The training and unknown data were imported as healthcare_data and unknown_healthcare_data, respectively.

```
1  import pandas as pd
2
3  healthcare_data = pd.read_csv('Assignment3-Healthcare-Dataset.csv')
4  unknown_healthcare_data = pd.read_csv('Assignment3-Unknown-Dataset.csv')
```

To view the counts of missing data on every column as well as the percentage of missing data in both the healthcare_data and unknown_ healthcare_data datasets, the built-in function isnull() was used to view this data. Starting with the healthcare_data:

```
1  missing_val = healthcare_data.isnull().sum()
2  missing_val = missing_val[missing_val > 0].sort_values(ascending=False)
3  missing_perc = (missing_val / len(healthcare_data)) * 100
4  missing_data = pd.DataFrame({'Missing Values': missing_val, 'Percentage (%)': missing_perc})
5
6  print(missing_data)
```

```
               Missing Values  Percentage (%)
LOSgroupNum             42447       89.968207
marital_status           8179       17.335735
NumCallouts              2370        5.023315
age                      2339        4.957609
religion                  372        0.788470
AdmitDiagnosis             20        0.042391
```

As seen in the screenshot above, there are:

- 42447 missing values in LOSgroupNum, totalling 89.97% of the data column.
- 8179 missing values in ‘marital_status’, totalling 17.34% of the data column.
- 2370 missing values in ‘NumCallouts’, totalling 5.02% of the data column.
- 2339 missing values in ‘age’, totalling 4.96% of the data column.
- 372 missing values in ‘religion’, totalling 0.79% of the data column.
- 20 missing values in ‘AdmitDiagnosis’, totalling 0.04% of the data column.

This process was also repeated for the unknown_healthcare_data:

```
missing_val = unknown_healthcare_data.isnull().sum()
missing_val = missing_val[missing_val > 0].sort_values(ascending=False)
missing_perc = (missing_val / len(unknown_healthcare_data)) * 100
missing_data = pd.DataFrame({'Missing Values': missing_val, 'Percentage (%)': missing_perc})

print(missing_data)
```

```
               Missing Values  Percentage (%)
LOSgroupNum             10631       90.123771
marital_status           1949       16.522550
age                       609        5.162767
NumCallouts               578        4.899966
religion                   86        0.729061
AdmitDiagnosis              5        0.042387
```

As seen in the screenshot above, there are:

- 10631 missing values in LOSgroupNum, totaling 90.12% of the data column.
- 1949 missing values in ‘marital_status’, totaling 16.52% of the data column.
- 609 missing values in ‘age’, totalling 5.16% of the data column.
- 578 missing values in ‘NumCallouts’, totaling 4.90% of the data column.
- 86 missing values in ‘religion’, totalling 0.73% of the data column.
- 5 missing values in ‘AdmitDiagnosis’, totaling 0.04% of the data column.

Based on the missing data and using logic, it was concluded:

- All cases of ‘NEWBORN’ in the ‘AdmitDiagnosis’ column would have an age of 0.
- All cases of an age less than 18 would have a ‘marital_status’ of ‘SINGLE’.
    - Additionally, it was noticed there were 3 cases of NEWBORNs had a marital status of ‘MARRIED’, while this is an interesting data point, it was decided to not change the marital status of those instances.

Due to some NEWBORNs having a marital status of 'MARRIED' it was decided to label all cases under the age of 18 to marital status 'NA'. Using this logic, some missing values for ‘age’ and 'marital_status' were imputed using a for-loop with a nested if- statement in both the healthcare_data and unknown_healthcare_data datasets:

```
for i, row in healthcare_data.iterrows():
    if row['AdmitDiagnosis'] == 'NEWBORN':
        healthcare_data.at[i, 'age'] = 0

for i, row in unknown_healthcare_data.iterrows():
    if row['AdmitDiagnosis'] == 'NEWBORN':
        unknown_healthcare_data.at[i, 'age'] = 0
```

```
condition = (healthcare_data['age'] < 18) & (healthcare_data['marital_status'].isna())
healthcare_data.loc[condition, 'marital_status'] = 'NA'

condition_unknown = (unknown_healthcare_data['age'] < 18) & (unknown_healthcare_data['marital_status'].isna())
unknown_healthcare_data.loc[condition_unknown, 'marital_status'] = 'NA'
```

To view the remaining missing data, the same code to view the missing data for both datasets was executed, instead of showing the code as it's a repeat from the code before, the results will only be displayed instead:

healthcare_data:

	Missing Values	Percentage (%)
LOSgroupNum	42447	89.968207
NumCallouts	2370	5.023315
age	2036	4.315388
marital_status	1921	4.071641
religion	372	0.788470
AdmitDiagnosis	20	0.042391

As seen in the image above, the missing marital_status data in the healthcare_data dataset reduced by 6258, approx. 13% reduction overall, which is a massive improvement.

For age, the number is missing cases reduced by 303, approx. 0.6% reduction overall, which is a meaningful improvement.

unknown_healthcare_data:

	Missing Values	Percentage (%)
LOSgroupNum	10631	90.123771
NumCallouts	578	4.899966
age	524	4.442184
marital_status	431	3.653781
religion	86	0.729061
AdmitDiagnosis	5	0.042387

As seen above, the missing marital_status data in the unknown_healthcare_data dataset reduced by 1518, approx. 12% reduction overall, which is a massive improvement.

For age, the number is missing cases reduced by 85, approx. 0.7% reduction overall, which is a meaningful improvement.

Moving onto LOSgroupNum, after examining the dataset, following pattern for LOSgroupNum group assignment based on LOSdays was observed:

- 0-4 LOSdays were assigned a LOSgroupNum of 1.
- 4-8 LOSdays were assigned a LOSgroupNum of 2.
- 8-12 LOSdays were assigned a LOSgroupNum of 3.
- 12 and above LOSdays were assigned a LOSgroupNum of 4.

Although LOSdays 0-3, 5-7, 9-11 LOSdays were each to assign a LOSgroupNum to, LOSdays with 4, 8 and 12 were ambiguous, as some datapoints had both the upper and lower group assigned to the same LOSdays.

With this understanding, the NumPy was imported and the LOSgroupNum missing values were imputed using a for- loop and nested if and elif-statements while using this logic:

- 1 for LOSdays less than 4.
- 1.5 for LOSdays equal to 4.
- 2 for LOSdays less than 8.
- 2.5 for LOSdays equal to 8.
- 3 for LOSdays less than 12.
- 3.5 for LOSdays equal to 12.
- 4 for LOSdays greater than 12.

For LOSdays equal to 4, 8 and 12, it is unclear which category they belong to so it was set to .5 between the upper and lower groups.

```
import numpy as np

for i, row in healthcare_data.iterrows():
    if pd.isna(row['LOSgroupNum']):
        if row['LOSdays'] < 4:
            healthcare_data.at[i, 'LOSgroupNum'] = 1
        elif row['LOSdays'] == 4:
            healthcare_data.at[i, 'LOSgroupNum'] = 1.5
        elif row['LOSdays'] < 8:
            healthcare_data.at[i, 'LOSgroupNum'] = 2
        elif row['LOSdays'] == 8:
            healthcare_data.at[i, 'LOSgroupNum'] = 2.5
        elif row['LOSdays'] < 12:
            healthcare_data.at[i, 'LOSgroupNum'] = 3
        elif row['LOSdays'] == 12:
            healthcare_data.at[i, 'LOSgroupNum'] = 3.5
        elif row['LOSdays'] > 12:
            healthcare_data.at[i, 'LOSgroupNum'] = 4

for i, row in unknown_healthcare_data.iterrows():
    if pd.isna(row['LOSgroupNum']):
        if row['LOSdays'] < 4:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 1
        elif row['LOSdays'] == 4:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 1.5
        elif row['LOSdays'] < 8:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 2
        elif row['LOSdays'] == 8:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 2.5
        elif row['LOSdays'] < 12:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 3
        elif row['LOSdays'] == 12:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 3.5
        elif row['LOSdays'] > 12:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 4
```

For the remaining missing ages, there were no obvious discernible patterns to determine these ages, so it was decided to impute the median age of the dataset (which is 58.0 for both datasets) but imputed a number +10 the median age for SINGLE patients and -10 the median age for WIDOWED patients because the median age for SINGLE patients was 51-52 whereas WIDOWED patients were 76. This was achieved using a for-loop and a nested if-statement:

```
median_age = healthcare_data['age'].median()

for i, row in healthcare_data.iterrows():
    if pd.isna(row['age']):
        if row['marital_status'] == 'SINGLE':
            imputed_age = int(median_age - 10)
        elif row['marital_status'] == 'WIDOWED':
            imputed_age = int(median_age + 10)
        else:
            imputed_age = int(median_age)
        healthcare_data.at[i, 'age'] = imputed_age

for i, row in unknown_healthcare_data.iterrows():
    if pd.isna(row['age']):
        if row['marital_status'] == 'SINGLE':
            imputed_age = int(median_age - 10)
        elif row['marital_status'] == 'WIDOWED':
            imputed_age = int(median_age + 10)
        else:
            imputed_age = int(median_age)
        unknown_healthcare_data.at[i, 'age'] = imputed_age
```

For the missing data in NumCallouts, there were no obvious discernible patterns to approximate the NumCallout value for each specific row. Therefore, imputing the median value again (which was 0.0) was the only reasonable approach. Although imputing the median value is effective in resolving any missing numerical values, it is not always the best strategy.

```
healthcare_data['NumCallouts'].fillna(healthcare_data['NumCallouts'].median(), inplace=True)

unknown_healthcare_data['NumCallouts'].fillna(unknown_healthcare_data['NumCallouts'].median(), inplace=True)
```

The remaining missing values were examined in the datasets:

Healthcare_data:

	Missing Values	Percentage (%)
marital_status	1921	4.071641
religion	372	0.788470
AdmitDiagnosis	20	0.042391

Unknown_healthcare_data:

	Missing Values	Percentage (%)
marital_status	431	3.653781
religion	86	0.729061
AdmitDiagnosis	5	0.042387

As seen in the two images above, all missing values were resolved in ‘LOSgroupNum’, ‘age’ and ‘NumCallout’ with only missing values remaining in ‘marital_status’, ‘religion’ and ‘AdmitDiagnosis’.

For the remaining missing values, there were no determinable patterns in the data to impute these values. 'Ethnicity' was inspected for possible relationships to impute the missing values in religion but there were not any consistent trends found. For marital status, it was observed that many older patients were mostly married, widowed, or divorced but some were also single, so there was no strategy for imputing these values without purely guessing. For missing values in AdmitDiagnosis, there were again no consistent trends. Using this information, it was decided to impute these missing values as 'NA' instead of dropping them for both datasets, as important data may be lost which may affect in making accurate predictions.

```
healthcare_data['marital_status'].fillna('NA', inplace=True)
healthcare_data['religion'].fillna('NA', inplace=True)
healthcare_data['AdmitDiagnosis'].fillna('NA', inplace=True)

unknown_healthcare_data['marital_status'].fillna('NA', inplace=True)
unknown_healthcare_data['religion'].fillna('NA', inplace=True)
unknown_healthcare_data['AdmitDiagnosis'].fillna('NA', inplace=True)
```

3. Train, Test & Split

Now that there are no more missing data in our datasets, it is time to train and test our models. But first, to train and test models, splitting the data into training and validation sets is necessary.

The train_test_split function was imported from the sklearn.model.selection module.Then the data is split into X and y, where X will hold all the features of the dataset except for the 'ExpiredHospital' feature, the target variable. And y will hold the target variable.

Using the 75-25 split method, 25% of the data will be used for the validation set ('X_val' and 'y_val') and the remaining 75% of the data will be used for the training set ('X_train' and 'y_train'). Additionally, the stratify parameter and set it to 'y', this ensures that the distribution of the target variable will be consistent in both the training and validation sets, which is very important to do for imbalanced datasets, as around 10-11% of the rows have a positive case for ExpiredHospital, whereas the negative case has 89-90% of all rows.

The models will be trained on the X_train and y_train sets, where the model is trained on X_train then compares its predictions with the true values in y_train then adjusts its parameters accordingly during training to minimize false predictions. The model is then evaluated on the X_val and y_val sets, where the model is tested on X_val to make predictions then uses y_val set to assess the model's performance on unseen data.

```
from sklearn.model_selection import train_test_split

X = healthcare_data.drop('ExpiredHospital', axis=1)
y = healthcare_data['ExpiredHospital']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)

(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```

4. Encoding Categorical Variables

Most machine learning models require numerical input and can't handle non-numeric categorical data; therefore, it is necessary to encode this data into a format that can be handled by the models.

There two main options for encoding data, one-hot encoding and label-encoding. This decision depends on the type of data; if the data has ordinal categorical data, where there is a clear ordering or hierarchy, one-hot encoding is recommended and when categorical variables have no inherent order, label-encoding is recommended.

Label-encoding encodes each unique variable with a unique number, starting at 1, whereas one-hot encoding creates a new column for each unique variable and assigns a 1 or 0 to the row in question, 0 meaning false and 1 meaning true.

While label-encoding doesn't increase the dimensionality of the dataset, machine learning models can interpret one variable as being less than or greater than another variable, which is not desired if there isn't an order to the data. This ordering problem doesn't occur with one-hot encoding, however it greatly increases the dimensionality of the dataset, depending on the number of unique non-numeric variables each column has.

Something to note, tree-based models can interpret label-encoded data without assuming any order on the label-encoded data, whereas other models will interpret label-encoded data as ordered.

Initially, the data was one-hot encoded for all the non-tree based machine learning models to keep the data consistent between those models so they can be fairly compared between each other, although this will result in a high dimensionality of datasets which will be computationally expensive train and test models on.

Using the pandas function get_dummies, the data is one-hot encoded:

```
1  X_train_hot = pd.get_dummies(X_train)
2  X_val_hot = pd.get_dummies(X_val)
3  unknown_healthcare_data_hot = pd.get_dummies(unknown_healthcare_data)
4
5  X_val_hot = X_val_hot.reindex(columns=X_train_hot.columns)
6
7  X_train_hot = X_train_hot.fillna(0)
8  X_val_hot = X_val_hot.fillna(0)
```

In the 5th line, X_val_hot columns order was matched with the columns in X_train_hot to minimize any data mismatch.

In the 7th and 8th line, the NaN values is set to 0 as models were throwing errors, this was probably due to unseen data in either 'AdmitDiagnosis' or 'AdmitProcedure' between the training and evaluation subsets, resulting in NaN values.

Using the following code, the data is also label-encoded for tree-based models:

```
1  from sklearn.preprocessing import LabelEncoder
2
3  combined = pd.concat([X_train, X_val, unknown_healthcare_data], axis=0)
4
5  categorical_cols = combined.select_dtypes(include=[object, 'category']).columns
6  for column in categorical_cols:
7      combined[column] = LabelEncoder().fit_transform(combined[column])
8
9  train_len = len(X_train)
10 val_len = len(X_val)
11
12 X_train = combined.iloc[:train_len]
13 X_val = combined.iloc[train_len:train_len+val_len]
14 unknown_healthcare_data = combined.iloc[train_len+val_len:]
```

In the code above, in the 1st line first imported LabelEncoder from the sklearn.preprocessing module.

The datasets that will be encoded was combined using the panda's concatenation function in the 3rd line. This was done because the categorical columns between the datasets may be unique, and if a label was assigned to one variable and the same label to a different variable in the other datasets, it will result in a mismatch of data which may affect the accuracy of the model.

In the 5th line of code, the categorical columns were specified to be encoded into numeric form, as the label-encoder will not only encode the categorical variables, but the numerical ones too. Specifying the categorical columns, it ensures the numerical data isn't affected as they don't need to be encoded prior to machine learning.

In the 6th line, a for loop was used to encode the categorical data in each column containing categorical data, while ensuring the labels are consistent throughout the datasets by using the combined variable.

The combined data was then split back into the original dataframes so that they may be used for training and testing using the original data length and .iloc to split the data based on the length of the original datasets.

1C Training Classifiers and Interpreting Results

This section will include training several classifiers then interpreting and discussing the results from the accuracy scores, F1 scores, classification reports and ROC-AUC Curves.

The following models will be used to train on the data:

Random Forest (Ensemble – Bagging)
- Creates multiple decision trees during training then merges the predictions of each tree.

Decision Tree (Decision-based)
- Flowchart tree structure where each node is a decision, and each branch is an outcome.

LightGBM (Light Gradient Boosting)
- An efficient and scalable gradient boosting model that uses tree-based algorithms and has been optimized for speed and accuracy.

XGBoost (Extreme Gradient Boosting)
- A highly efficient, versatile, and optimized gradient boosting model designed for speed and performance.

Catboost (Categorical Gradient Boosting)
- Gradient boosting model which focuses on categorical features.

Support Vector Machine (Margin-based)
- Supervised model that aims to find a hyperplane in an N-dimensional space to classify data points.

Gaussian Naive Bayes (Probabilistic)
- Probabilistic model based on Bayes' Theorem.

Logistic Regression (Regression)
- Statistical model used for binary classification problems.

Neural Network (Deep Learning)
- Interconnected nodes or 'neurons' that processes information in layers, like the human brain.

K-Nearest Neighbors (Instance-based)
- Non-parametric method used for classification and regression. Finds the labeled data points closest to the point that needs a prediction.

Stacking Classifier (RF, DT, LGBM, XGB & Catboost) (Ensemble – Stacking)
- Ensemble model that combines predictions from multiple models, attempting to combine the strengths of the models to increase the overall performance.

Note: The tree-based, gradient boosting & stacking models will be trained on the label-encoded and one-hot encoded and will only present the better performing model whereas the other models will be trained on the one-hot encoded data only due to the nature of their interactions with label-encoded data.
A thorough explanation will be given for the first model then the explanations will be brief for the following models as explanations of code and what the results represent are already given. This is to avoid repetition.

1. Random Forest (RT)

Trained on label-encoded data:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score

rf_model = RandomForestClassifier(random_state=42)

rf_model.fit(X_train, y_train)

predictions = rf_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

The necessary libraries were first imported then the Random Forest model was initialised with a random state of 42 (for reproducibility). The model was then trained on the X_train and y_train sets. Afterwards, the model was evaluated on the X_val evaluation set.

Using the sklearn.metrics library, the accuracy_score, f1_score, classification_report confusion_matrix functions were called then printed out the results using the print function.

Results

Results for the Random Forest model with label-encoded data:

```
Accuracy: 0.9363
F1 Score: 0.5894
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.97     10608
           1       0.84      0.45      0.59      1187

    accuracy                           0.94     11795
   macro avg       0.89      0.72      0.78     11795
weighted avg       0.93      0.94      0.93     11795

Confusion Matrix:
[[10505   103]
 [  648   539]]
```

Key Metrics

Accuracy (93.63%):
This model correctly predicted 93.63% of all cases in the evaluation set. However, since the data is severely imbalanced; accuracy can be misleading as the model can correctly predict the majority class but incorrectly predict the minority class.

F1 Score (58.94%):
The F1 score combines both the precision and recall into 1 metric for the class 1 (the minority/positive class). For the minority class, the model in 58.94% effective in the model's ability to correctly identify the minority class instances while minimizing false positives and false negatives. This low score indicates room for improvement, whether through resampling methods, hyperparameter tuning or something else.

Classification Report

Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.94	0.84
Recall	0.99	0.45
F1-Score	0.97	0.59

Other metrics:

Precision: The precision is the accuracy of correctly identifying a specific class, i.e., out of all the instances the model predicted as class 0, 94% of them were class 0.

Recall: The recall is the percentage of correctly identified class, i.e., out of all the actual class 1 instance, the model correctly identified 45% of them, which is quite low.

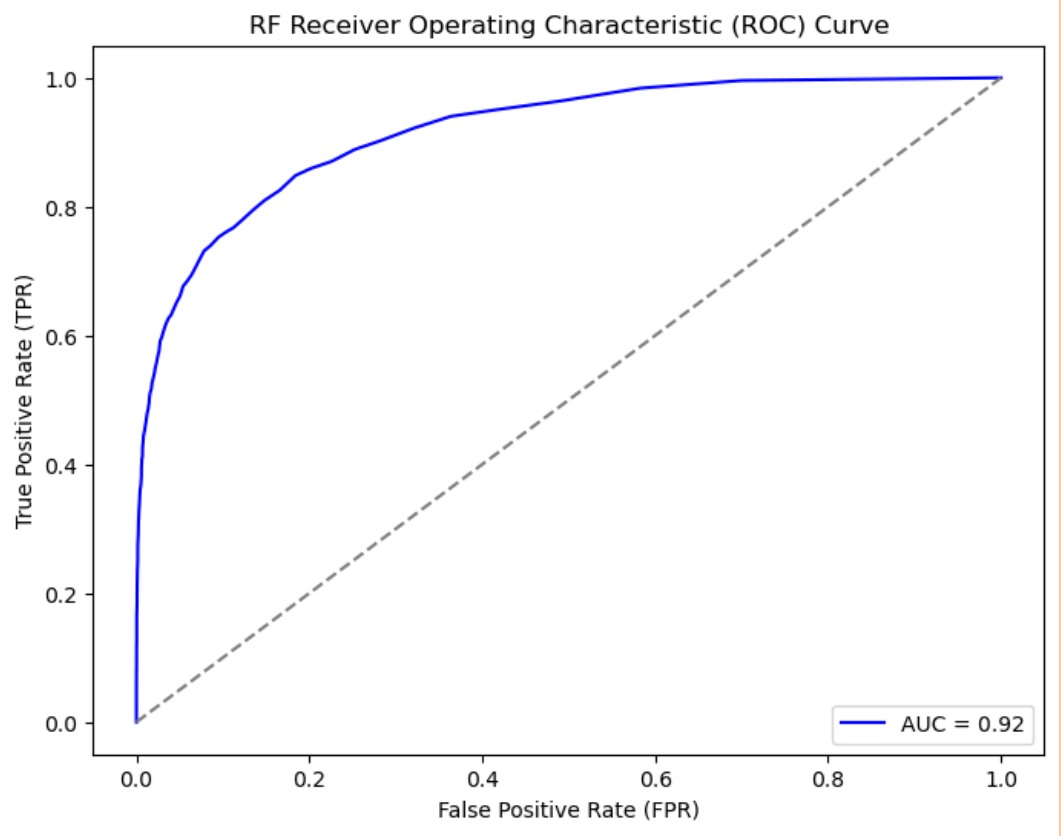
Confusion Matrix

The confusion matrix displays all the true & false positives and true & false negatives.

[10505, 103] ← True Negative, False Positive
[648, 539] ← False Negative, True Positive

The model predicted 10505 true negatives, 539 true positives, 648 false negatives and 103 false positives.

ROC-AUC Curve



Curve Shape:

The blue curve on the graph above represents the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FRP) at various thresholds. The closer the curve is to the top left of the graph the better the model is performing.

Area Under the Curve (AUC):

The AUC is 0.92, as seen in the legend at the bottom right of the graph. This value ranges from 0 to 1, where the value closer to 1 indicates a well performing model. The graph above is 0.92, which indicates a very good performing model. However, since the data is imbalanced, this could be misleading without additional information. However, due the low recall for positive class (0.45), the model isn't performing very well at correctly classifying instances of the positive class.

Comparison with Random Chance:

The dotted line along the middle represents a model that is performing the same as random guessing without any understanding of the data. If the blue curve is above the dotted line, which it is, that represents a model that is performing better than random guessing.

2. Decision Tree (DT)

Trained on one-hot encoded data:

```python
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_hot, y_train)

predictions = dt_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the Decision Tree model with one-hot encoded data (which performed better than label):

```
Accuracy: 0.9089
F1 Score: 0.5173
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     10608
           1       0.55      0.49      0.52      1187

    accuracy                           0.91     11795
   macro avg       0.75      0.72      0.73     11795
weighted avg       0.90      0.91      0.91     11795

Confusion Matrix:
[[10144   464]
 [  611   576]]
```

Key Metrics

Accuracy (90.89%):
This model correctly predicted 90.89% of all cases in the evaluation set.

F1 Score (51.73%):
The model is 51.73% effective in correctly identifying minority class instances.

Classification Report

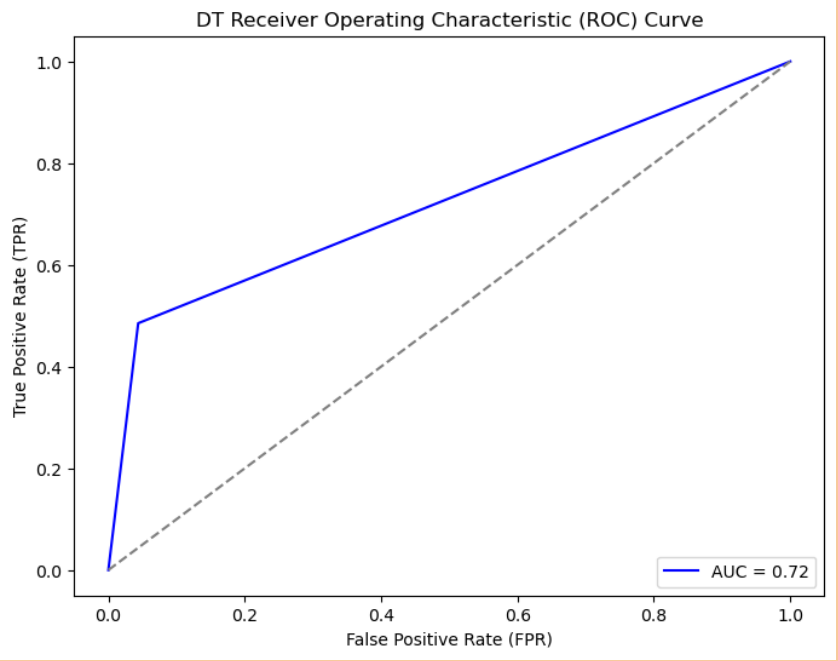
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.94	0.55
Recall	0.96	0.49
F1-Score	0.95	0.52

Confusion Matrix

[10144, 464] ← True Negative, False Positive
[611, 576] ← False Negative, True Positive

The model predicted 10144 true negatives, 576 true positives, 611 false negatives and 464 false positives.

ROC-AUC Curve



The AUC is 0.72 and exhibits a sharp change at a low threshold which is typical for decision trees.

3. XGBoost (XGB)

Trained on label-encoded data:

```python
from xgboost import XGBClassifier

xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train)

predictions = xgb_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the XGBoost model with label-encoded data:

```
Accuracy: 0.9401
F1 Score: 0.6449
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     10608
           1       0.80      0.54      0.64      1187

    accuracy                           0.94     11795
   macro avg       0.87      0.76      0.81     11795
weighted avg       0.94      0.94      0.93     11795

Confusion Matrix:
[[10446   162]
 [  545   642]]
```

Key Metrics

Accuracy (94.01%):
This model correctly predicted 94.01% of all cases in the evaluation set.

F1 Score (64.49%):
The model is 64.49% effective in correctly identifying minority class instances.

Classification Report

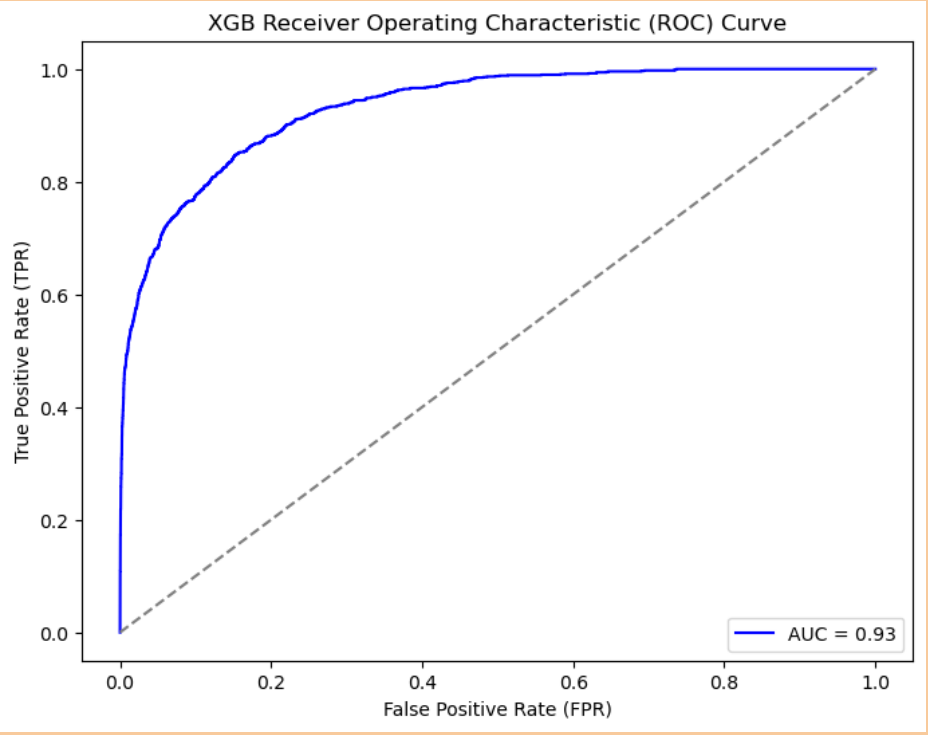
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.95	0.80
Recall	0.98	0.54
F1-Score	0.97	0.64

Confusion Matrix

[10446, 162] ← True Negative, False Positive
[545, 642] ← False Negative, True Positive

The model predicted 10446 true negatives, 642 true positives, 545 false negatives and 162 false positives.

ROC-AUC Curve



The AUC is 0.93 and performing well but then plateaus at 0.4 threshold.

4. LightGBM (LGBM)

Trained on label-encoded data:

```python
from lightgbm import LGBMClassifier

lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_train, y_train)

predictions = lgbm_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the LightGBM model with label-encoded data:

```
Accuracy: 0.9423
F1 Score: 0.6517
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     10608
           1       0.83      0.54      0.65      1187

    accuracy                           0.94     11795
   macro avg       0.89      0.76      0.81     11795
weighted avg       0.94      0.94      0.94     11795

Confusion Matrix:
[[10477   131]
 [  550   637]]
```

Key Metrics

Accuracy (94.23%):
This model correctly predicted 94.23% of all cases in the evaluation set.

F1 Score (65.17%):
The model is 65.17% effective in correctly identifying minority class instances.

Classification Report

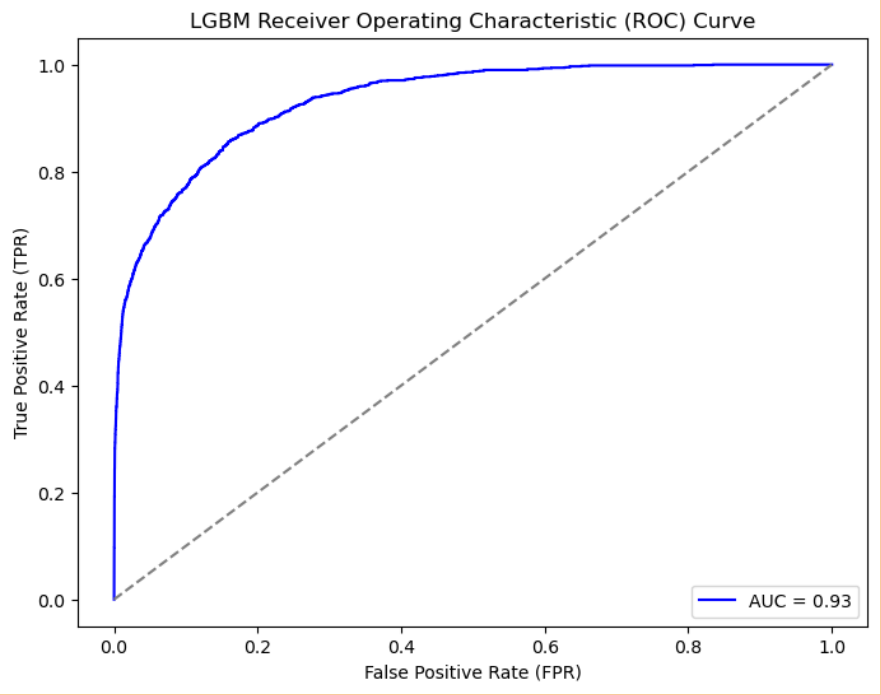
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.95	0.83
Recall	0.99	0.54
F1-Score	0.97	0.65

Confusion Matrix

[10477, 131] ← True Negative, False Positive
[550, 637] ← False Negative, True Positive

The model predicted 10477 true negatives, 637 true positives, 550 false negatives and 131 false positives.

ROC-AUC Curve



The AUC is 0.93 and performing well but then plateaus at 0.4 threshold.

5. CATBOOST

Trained on label-encoded data:

```python
from catboost import CatBoostClassifier

catboost_model = CatBoostClassifier(random_state=42)
catboost_model.fit(X_train, y_train)

predictions = catboost_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

RESULTS

Results for the Catboost model with label-encoded data:

```
Accuracy: 0.9413
F1 Score: 0.6440
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     10608
           1       0.83      0.53      0.64      1187

    accuracy                           0.94     11795
   macro avg       0.89      0.76      0.81     11795
weighted avg       0.94      0.94      0.94     11795

Confusion Matrix:
[[10477   131]
 [  561   626]]
```

KEY METRICS

Accuracy (94.13%):
This model correctly predicted 94.13% of all cases in the evaluation set.

F1 Score (64.40%):
The model is 64.40% effective in correctly identifying minority class instances.

Classification Report

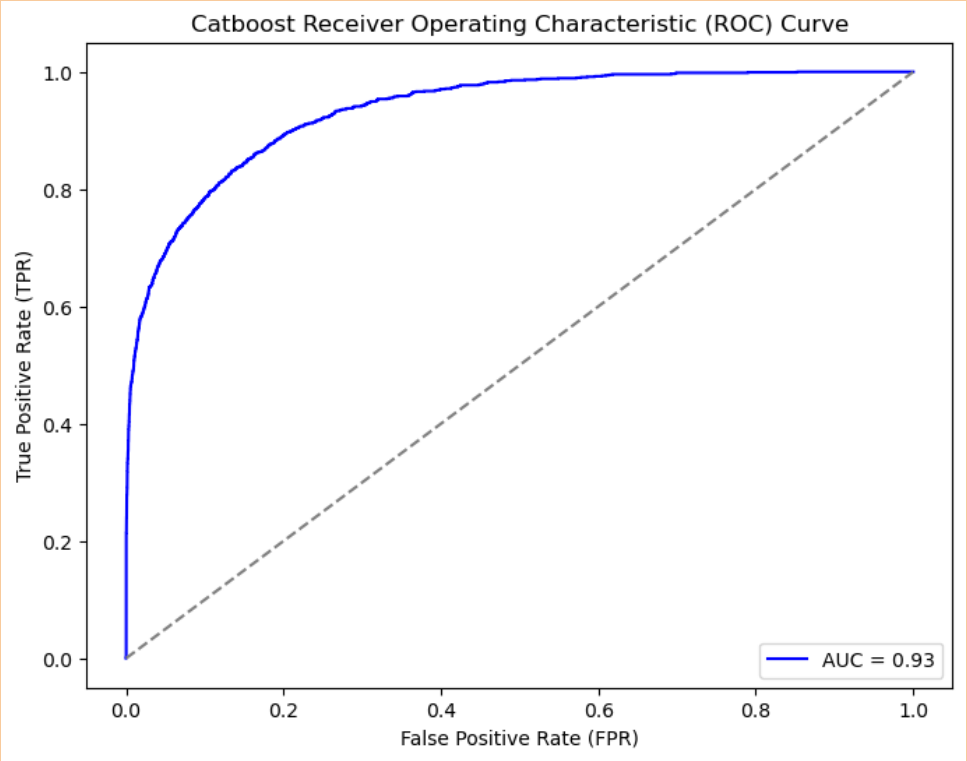
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.95	0.83
Recall	0.99	0.53
F1-Score	0.97	0.64

Confusion Matrix

[10477, 131] ← True Negative, False Positive
[561, 626] ← False Negative, True Positive

The model predicted 10477 true negatives, 626 true positives, 561 false negatives and 131 false positives.

ROC-AUC Curve



The AUC is 0.93 and performing well but then plateaus at 0.4 threshold.

6. Support Vector Machine (SVM)

Trained on one-hot encoded data:

```python
from sklearn.svm import SVC

svm_model = SVC(random_state=42)
svm_model.fit(X_train_hot, y_train)

predictions = svm_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the SVM model with one-hot encoded data:

```
Accuracy: 0.9072
F1 Score: 0.1762
Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     10608
           1       0.83      0.10      0.18      1187

    accuracy                           0.91     11795
   macro avg       0.87      0.55      0.56     11795
weighted avg       0.90      0.91      0.87     11795

Confusion Matrix:
[[10584    24]
 [ 1070   117]]
```

Key Metrics

Accuracy (90.72%):
This model correctly predicted 90.72% of all cases in the evaluation set.

F1 Score (17.62%):
The model is 17.62% effective in correctly identifying minority class instances, which is extremely low.

Classification Report

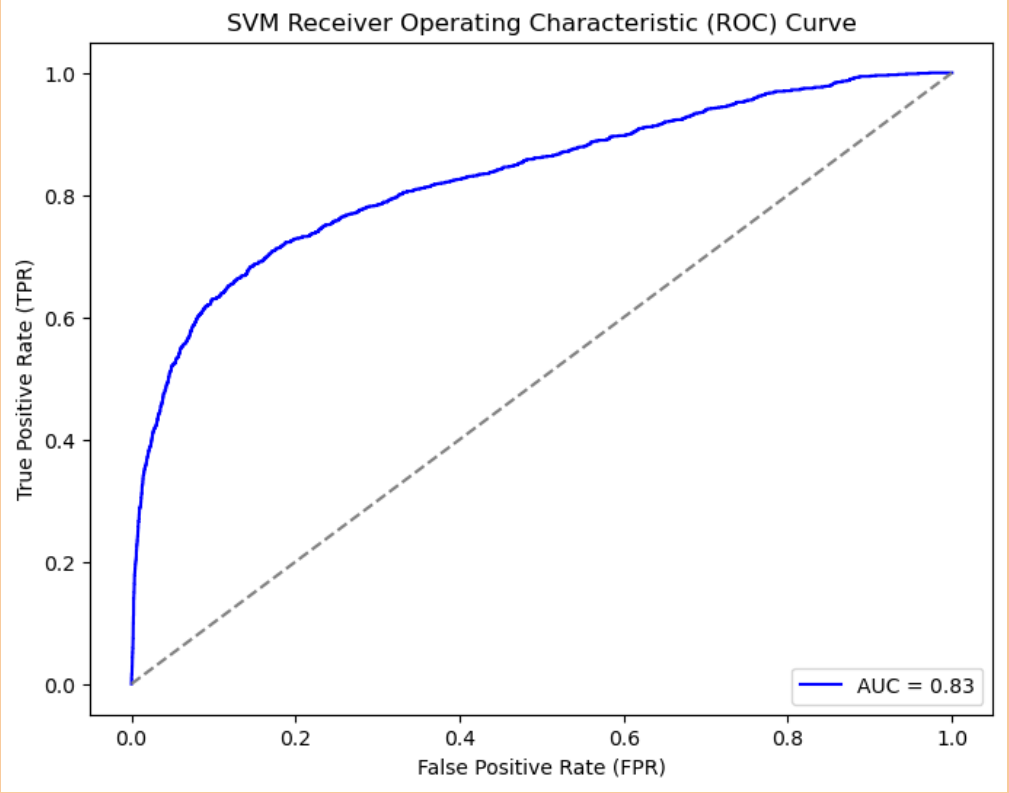
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.91	0.83
Recall	1.00	0.10
F1-Score	0.95	0.18

Confusion Matrix

[10584, 24] ← True Negative, False Positive
[1070, 117] ← False Negative, True Positive

The model predicted 10584 true negatives, 117 true positives, 1070 false negatives and 24 false positives.

ROC-AUC Curve



The AUC is 0.83 and performing well but then starts to taper off at the 0.1 threshold.

7. Gaussian Naïve Bayes (GNB)

Trained on label-encoded data:

```python
from sklearn.naive_bayes import GaussianNB

gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)

predictions = gnb_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the GNB model with label-encoded data (which performed better than one-hot):

```
Accuracy: 0.8998
F1 Score: 0.4685
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.95      0.94     10608
           1       0.50      0.44      0.47      1187

    accuracy                           0.90     11795
   macro avg       0.72      0.70      0.71     11795
weighted avg       0.89      0.90      0.90     11795

Confusion Matrix:
[[10092   516]
 [  666   521]]
```

Key Metrics

Accuracy (89.98%):
This model correctly predicted 89.98% of all cases in the evaluation set.

F1 Score (46.85%):
The model is 46.85% effective in correctly identifying minority class instances.

Classification Report

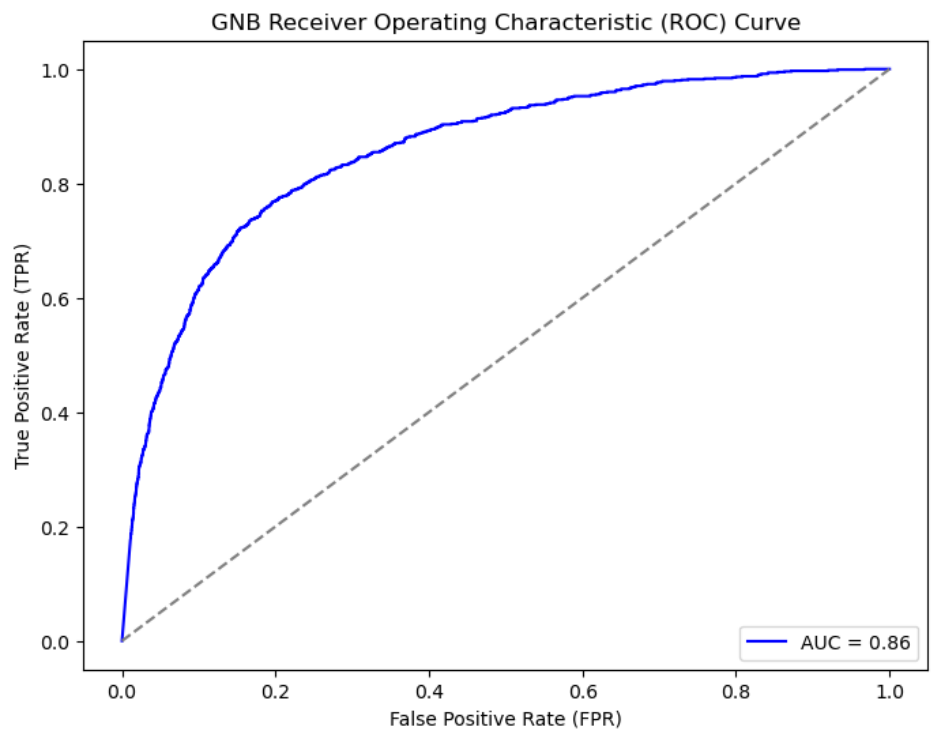
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.94	0.50
Recall	0.95	0.44
F1-Score	0.94	0.47

Confusion Matrix

[10092, 516] ← True Negative, False Positive
[666, 521] ← False Negative, True Positive

The model predicted 10092 true negatives, 521 true positives, 666 false negatives and 516 false positives.

ROC-AUC Curve



The AUC is 0.86 and performing well but then starts to taper off at 0.2-0.25 threshold.

8. Logistic Regression (LogReg)

Trained on one-hot encoded data:

```python
from sklearn.linear_model import LogisticRegression

logreg_model = LogisticRegression(random_state=42)
logreg_model.fit(X_train_hot, y_train)

predictions = logreg_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the LogReg model with label-encoded data (which performed better than one-hot):

```
Accuracy: 0.9178
F1 Score: 0.4164
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     10608
           1       0.73      0.29      0.42      1187

    accuracy                           0.92     11795
   macro avg       0.83      0.64      0.69     11795
weighted avg       0.91      0.92      0.90     11795

Confusion Matrix:
[[10479   129]
 [  841   346]]
```

Key Metrics

Accuracy (91.78%):
This model correctly predicted 91.78% of all cases in the evaluation set.

F1 Score (41.64%):
The model is 41.64% effective in correctly identifying minority class instances.

Classification Report

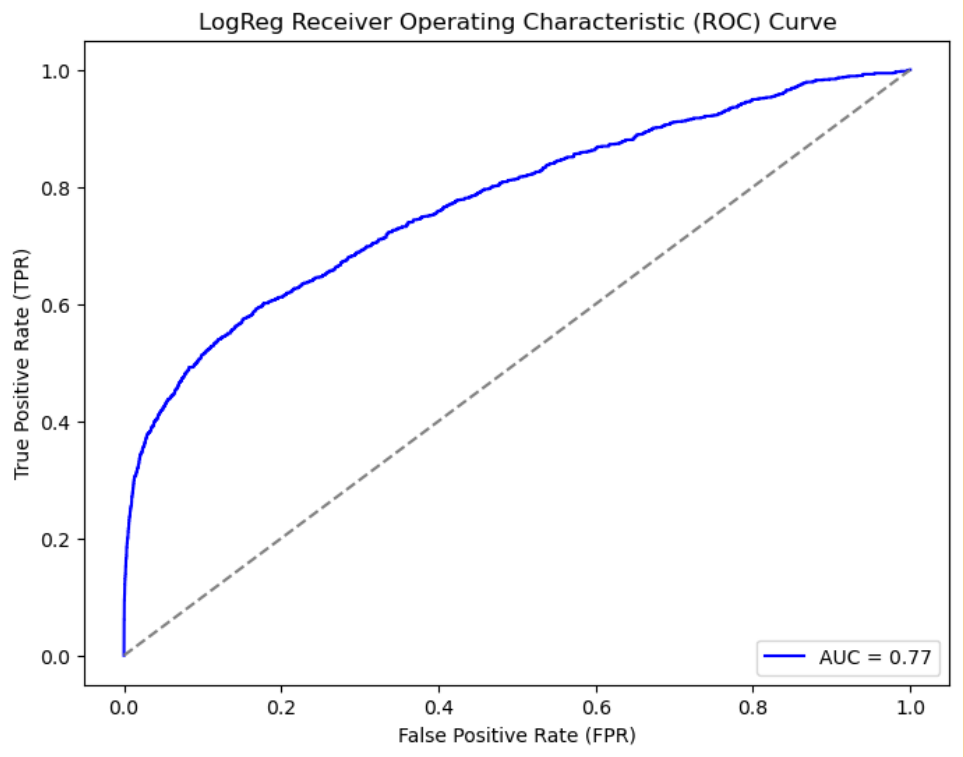
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.93	0.73
Recall	0.99	0.29
F1-Score	0.96	0.42

Confusion Matrix

[10479, 129] ← True Negative, False Positive
[841, 346] ← False Negative, True Positive

The model predicted 10479 true negatives, 346 true positives, 841 false negatives and 129 false positives.

ROC-AUC Curve



The AUC is 0.77 and performing well but then starts to taper off at 0.1 threshold.

9. Neural Network (NN)

Trained on one-hot encoded data:

```python
from sklearn.neural_network import MLPClassifier

nn_model = MLPClassifier(random_state=42)
nn_model.fit(X_train_hot, y_train)

predictions = nn_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the NN model with one-hot encoded data:

```
Accuracy: 0.9380
F1 Score: 0.6238
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     10608
           1       0.80      0.51      0.62      1187

    accuracy                           0.94     11795
   macro avg       0.87      0.75      0.80     11795
weighted avg       0.93      0.94      0.93     11795

Confusion Matrix:
[[10458   150]
 [  581   606]]
```

Key Metrics

Accuracy (93.90%):
This model correctly predicted 93.90% of all cases in the evaluation set.

F1 Score (62.38%):
The model is 62.38% effective in correctly identifying minority class instances.

Classification Report

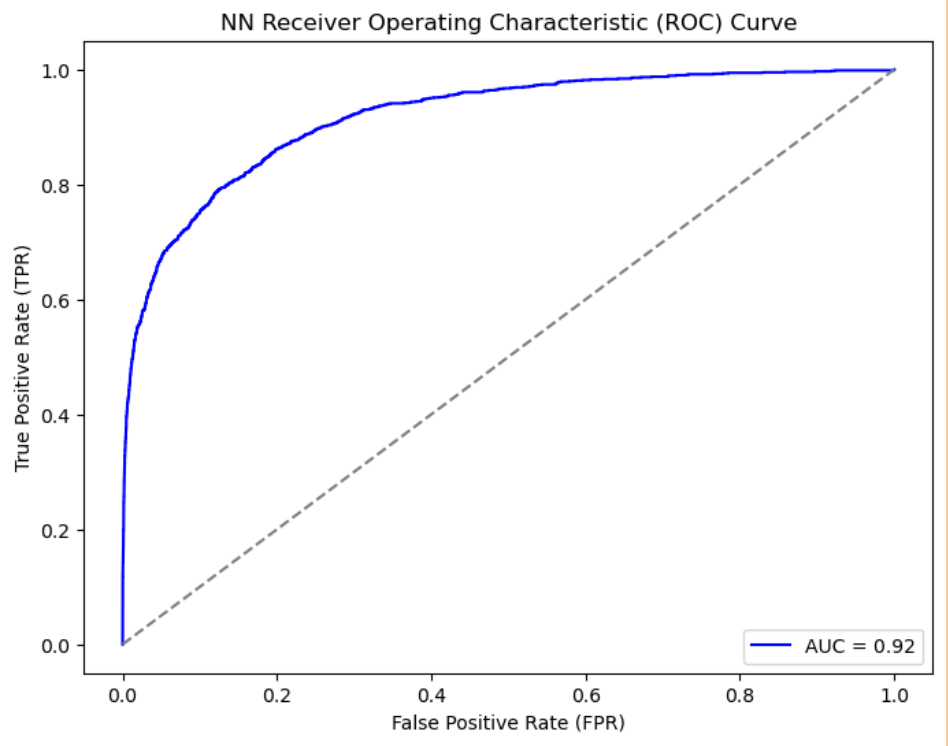
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.95	0.80
Recall	0.99	0.51
F1-Score	0.97	0.62

Confusion Matrix

[10458, 150] ← True Negative, False Positive
[581, 606] ← False Negative, True Positive

The model predicted 10458 true negatives, 606 true positives, 581 false negatives and 150 false positives.

ROC-AUC Curve



The AUC is 0.92 and performing well but then starts to plateau at 0.4 threshold.

10. K-Nearest Neighbors (KNN)

Trained on one-hot encoded data:

```python
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(algorithm='brute')
knn_model.fit(X_train_hot, y_train)

predictions = knn_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the KNN model with one-hot encoded data:

```
Accuracy: 0.9201
F1 Score: 0.5011
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96     10608
           1       0.67      0.40      0.50      1187

    accuracy                           0.92     11795
   macro avg       0.81      0.69      0.73     11795
weighted avg       0.91      0.92      0.91     11795

Confusion Matrix:
[[10380   228]
 [  714   473]]
```

Key Metrics

Accuracy (92.01%):
This model correctly predicted 92.01% of all cases in the evaluation set.

F1 Score (50.11%):
The model is 50.11% effective in correctly identifying minority class instances.

Classification Report

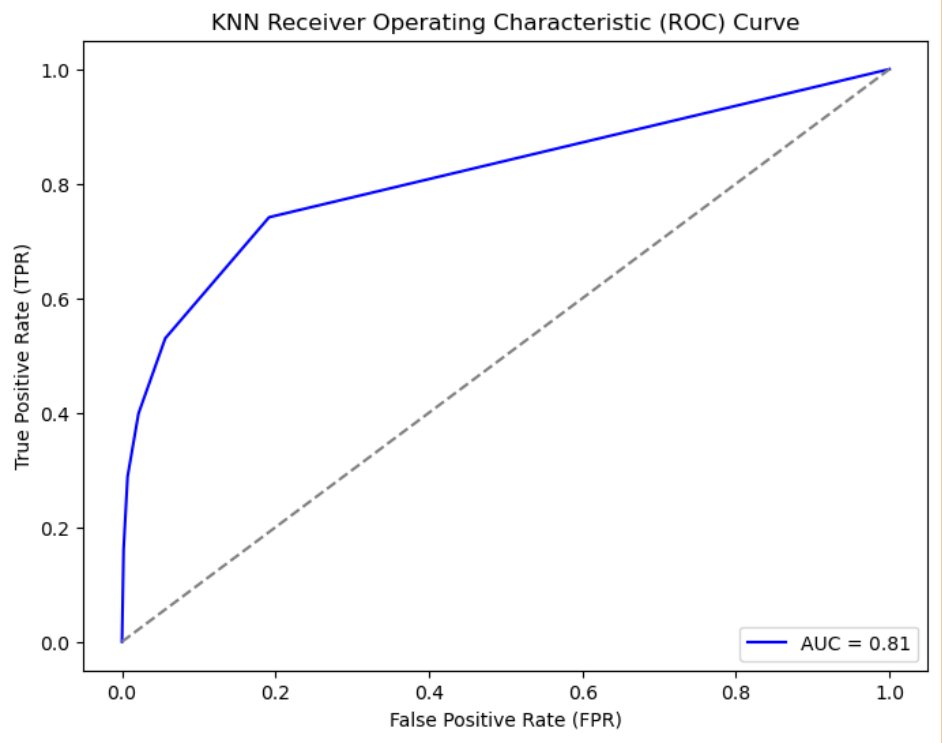
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.94	0.67
Recall	0.98	0.40
F1-Score	0.96	0.50

Confusion Matrix

[10380, 228] ← True Negative, False Positive
[714, 473] ← False Negative, True Positive

The model predicted 10380 true negatives, 473 true positives, 714 false negatives and 228 false positives.

ROC-AUC Curve



The AUC is 0.81 and performing well but then starts to taper off at 0.1-0.2 threshold.

11. Stacking Classifier

Trained on label-encoded data:

```python
from sklearn.ensemble import StackingClassifier

stacking_model = StackingClassifier(estimators=[
    ('xgb', xgb_model),
    ('lgbm', lgbm_model),
    ('catboost', catboost_model)], final_estimator=LogisticRegression())
stacking_model.fit(X_train, y_train)
predictions = stacking_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results

Results for the Stacking model with label-encoded data:

```
Accuracy: 0.9421
F1 Score: 0.6535
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     10608
           1       0.82      0.54      0.65      1187

    accuracy                           0.94     11795
   macro avg       0.89      0.76      0.81     11795
weighted avg       0.94      0.94      0.94     11795

Confusion Matrix:
[[10468   140]
 [  543   644]]
```

Key Metrics

Accuracy (94.21%):
This model correctly predicted 94.21% of all cases in the evaluation set.

F1 Score (65.35%):
The model is 65.35% effective in correctly identifying minority class instances.

Classification Report

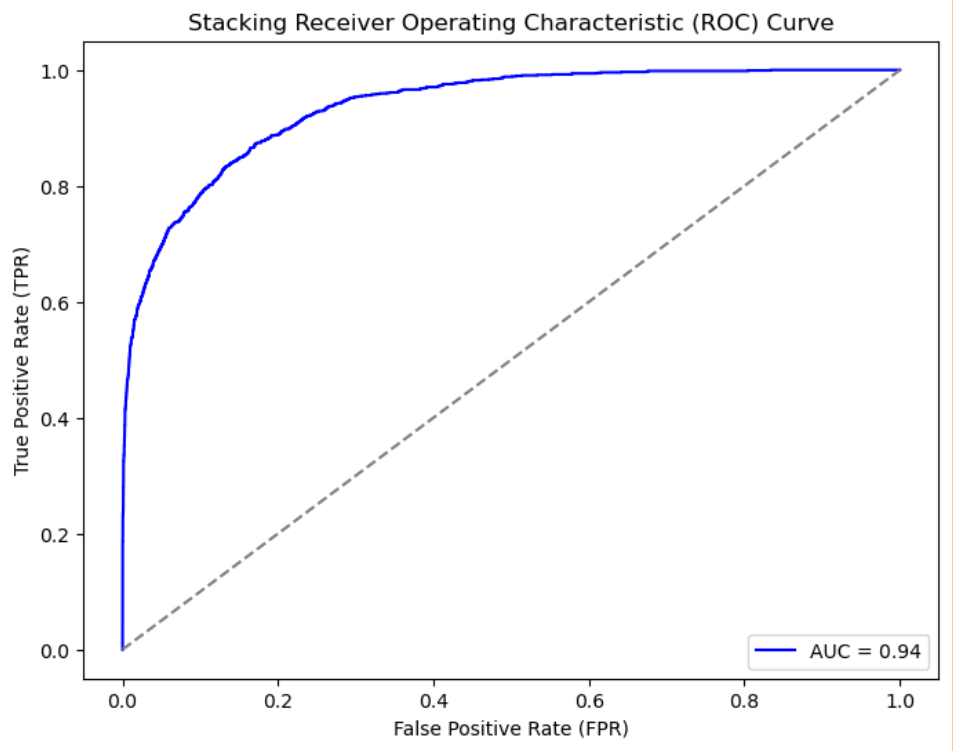
Metric	Class 0 (majority/negative class)	Class 1 (minority/negative class)
Precision	0.95	0.82
Recall	0.99	0.54
F1-Score	0.97	0.65

Confusion Matrix

[10468, 140] ← True Negative, False Positive
[543, 644] ← False Negative, True Positive

The model predicted 10468 true negatives, 644 true positives, 543 false negatives and 140 false positives.

ROC-AUC Curve



The AUC is 0.94 and performing well but then starts to plateau at 0.4 threshold.

1D Best Classifier Selection, Tuning & Conclusion

1. Selecting the Best Classifier & Further Tuning

The best performing classifier was Light Gradient Boosting Model (LightGBM) with an accuracy of 94.23% and an F1 Score of 65.17% which was the highest of all the singular models. For this reason, the LightGBM model was selected as the best classification model and will now perform fine tuning on this model to yield the best results. Additionally, LGBM is particularly well suited for binary classification tasks, especially when dealing with imbalanced datasets and is designed to be efficient on large datasets.

The stacking classifier did perform slightly better than the LGBM model, therefore additional tuning will be conducted on the XGB and Catboost models to be included into the Stacking classsifier with the LGBM model.

A dimensionality technique, Principle Component Analysis (PCA), was applied on the data (and tested it on other models) to reduce noise but it only reduced the models' performances.
For example, on the neural network model, applying PCA reduced the accuracy from 93.80% to 92.60% and F1 score from 62.28% to 49.33%.

Additionally, re-sampling the data by oversampling using SMOTE, ADASYN and under-sampling the majority class on the data, however it only resulted in a lesser performing models.

For example, on the LGBM model, SMOTE reduced the model's accuracy from 94.23% to 93.56% and F1 score from 65.17% to 64.25%. This was similar results for ADASYN, however under-sampling reduced the accuracy by ~9% and F1 score by ~13%.

Early Stopping, Feature Selection, Decision Threshold , Tuning Hyperparameters & Pseudo Labelling

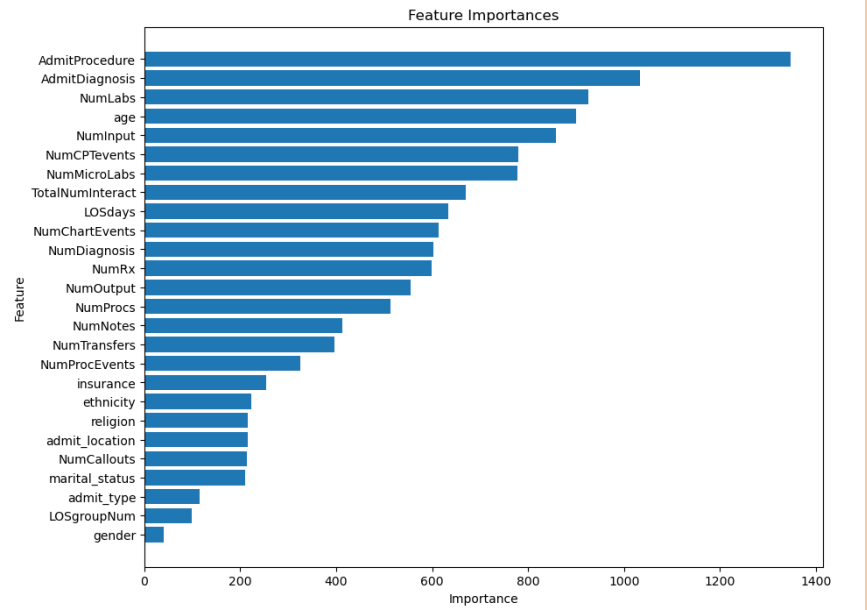
Early Stopping

Early stopped was introduced into the model set up which is a regularization technique so that if the model does not improve after a specific number of rounds, then the model will halt the training early and move on. The main goal of early stopping is to prevent over-fitting which is a common problem in machine learning. Over-fitting is when the model learns the training data too closely, so it performs exceptionally well on the training data but poorly on validation or unseen data. In this case, 50 rounds for early stopping was selected as it produced the best results.

Feature Selection

Additionally, feature selection was used in the model to further reduce noise. This is done by measuring the importance of each feature during training. This was tested several top features to train the data on and found that using the top 24 features of the dataset during training yielded the best results. In the images below, the features ‘LOSgroupNum’, and ‘gender’ were the two least important features and removing them reduced the noise of the data resulting in an increase in the model's performance.

	Feature	Importance
4	AdmitProcedure	1347
0	AdmitDiagnosis	1034
1	NumLabs	925
2	age	899
3	NumInput	857
9	NumCPTevents	779
5	NumMicroLabs	777
7	TotalNumInteract	670
12	LOSdays	633
6	NumChartEvents	613
8	NumDiagnosis	602
10	NumRx	598
11	NumOutput	555
14	NumProcs	513
13	NumNotes	413
15	NumTransfers	397
16	NumProcEvents	325
22	insurance	254
18	ethnicity	223
17	religion	216
20	admit_location	216
21	NumCallouts	214
19	marital_status	210
23	admit_type	115
24	LOSgroupNum	98
25	gender	41



Decision Threshold

The default decision threshold for models is 0.5, meaning that if the predicted probability for a positive case is above 50% then the instance will be classified as positive and will be predicted negative if the predicted probably is less than 50%.

Since the datasets has a severe class imbalance for the target variable, adjusting the decision threshold will be beneficial. Although there are some disadvantages of adjusting the decision threshold, as lowering the threshold (to be in favour of the minority/positive class) will decrease the precision of the model, however, this also increases the recall, which is a tradeoff worth testing.

Additionally, it's always good practice to apply context of the data to the tuning of the model. For this healthcare scenario, it is far more important to predict a true positive than a false positive when it comes to predicting the mortality of a patient.

Several decision thresholds below 0.5 were tested and found that a threshold of 0.4 produced the best results.

Hyperparameter Tuning

Hyperparameter tuning is a process of searching for the best model configuration to yield the best resultswhether that is accuracy or another metric. Each hyperparameter has a default setting and to determine the best configurations for several hyperparameters 'HalvingGridSearch', a variant of the GridSearch function from the sklearn.model_selection module, will be used.

HalvingGridSearch takes a grid of parameters with varying values and performs multiple iterations by selecting a subsample of these combinations and evaluates them while allocating more resources to the most promising parameters, then sample is then 'halved', keeping the best performing parameters and discarding the others. This process continues until the best performing parameters is found for the model.

Although this approach isn't as accurate, a large grid search using the traditional GridSearch function is computationally expensive as it tests all possible combinations, the larger the grid the more possible combinations. Therefore, HalvingGridSearch was selected to search for the best performing parameters. In addition to this, a Cross-Validation of 3 folds was included during training with the intention of limiting overfitting and underfitting during training on the hyperparameters.

Pseudo-labeling

Pseudo-labeling, also known as self-training, is where the high probability predictions is extracted then integrate it with the original training data then train the model again, this is an iterative approach but can be highly effective if performed under the correct circumstances, for example this assignment only requires making predictions of one set of unknown data and no other unseen data, making this approach quite convenient.

Putting it all together

The LightGBM model was the best performing model on both the training data and the unknown data (determined via Kaggle submissions). Therefore, the LGBM will be included in this report. The others with similar performance were configured identically with different parameters, see appendix for others).

The necessary libraries from sklearn were imported:

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
```

The parameter grid was defined that the HalvingGridSearch will be using to train the model on:

```
param_grid = {
    'objective': ['binary'],
    'class_weight': ['balanced', None],
    'num_leaves': [31, 40, 50, 60, 70],
    'learning_rate': [0.005, 0.01, 0.05, 0.1],
    'max_depth': [-1, 6, 8, 10],
    'min_child_samples': [10, 20, 30, 40],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
    'boosting_type': ['gbdt', 'dart'],
    'reg_alpha': [0, 0.1, 0.5, 1],
    'reg_lambda': [0, 1, 5, 10, 15],
    'n_estimators': [500, 1000, 2000, 3000]
}
```

After some research, these parameters were selected to test as they were popular parameters to adjust for the LightGBM model. 'Binary' for the objective was selected as this is a binary classification problem and selected values around the default values of these parameters as well as including the default values.
The model was initialised and set the evaluation metric to 'logloss' as it's the common metric used for binary classification tasks. The random_state to '42' was chosen for reproducibility and set n_jobs to '-1' so that it will use all available CPU cores to speed up training.

```
lgbm_model = lgb.LGBMClassifier(metric='logloss',
                                random_state=42,
                                n_jobs=-1)
```

The HalvingGridSearch was then set up on the model using the parameters and cross-validation set to 3. The model was then fit on the dataset to determine the best combination of parameters:

```
grid_search = HalvingGridSearchCV(lgbm_model,
                                  param_grid,
                                  cv=3,
                                  n_jobs=-1,
                                  factor=2)
grid_search.fit(X_train, y_train)
```

Afterwards, through pseudo labeling, predictions were on the unknown data and the high decision threshold predictions were fed as pseudo-labeled data back to the model to be trained again:

```
best_model = grid_search.best_estimator_
unseen_probs = best_model.predict_proba(unknown_healthcare_data)[:, 1]
high_confidence_indices = np.where(unseen_probs > 0.65)[0]
pseudo_labels = (unseen_probs[high_confidence_indices] > 0.5).astype(int)
pseudo_labeled_data = unknown_healthcare_data.iloc[high_confidence_indices]
X_train_augmented = pd.concat([X_train, pseudo_labeled_data])
y_train_augmented = np.concatenate([y_train, pseudo_labels])
best_model.fit(X_train_augmented, y_train_augmented)
```

The 24 best performing features in descending order from the best performing parameter combination was added to a list to then train the model on as well as set early stopping on the model to prevent over-fitting:

```
top_24 = pd.DataFrame({
    'feature': X_train.columns,
    'importance': best_model.feature_importances_
}).nlargest(24, 'importance')['feature'].tolist()

X_train_ = X_train[top_24]
X_val_ = X_val[top_24]

best_model.fit(X_train_, y_train,
               eval_set=[(X_val_, y_val)],
               eval_metric='binary_logloss',
               early_stopping_rounds=50,
               verbose=True)
```

To include the decision threshold of 0.4, the probabilities of all positive instances were determined and assigned to 'probs'. The decision threshold of the predictions were adjusted to 0.4 using an if-statement, if an instance the positive class is 40% or high then it will be assigned as a positive outcome, if not then assigned as a negative outcome:

```
probs = best_model.predict_proba(X_val_)[:, 1]
predictions_adjusted = [1 if prob > 0.4 else 0 for prob in probs]
```

The model's performance was then evaluated using the best combination of parameters, the top 24 features and a decision threshold of 0.4 in the form of accuracy, f1 score, a classification report, and a confusion matrix:

```
accuracy = accuracy_score(y_val, predictions_adjusted)
f1 = f1_score(y_val, predictions_adjusted)
classification_rep = classification_report(y_val, predictions_adjusted)
confusion_mat = confusion_matrix(y_val, predictions_adjusted)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

Results for the tuned LGBM model

```
Accuracy: 0.9415
F1 Score: 0.6683
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     10608
           1       0.78      0.59      0.67      1187

    accuracy                           0.94     11795
   macro avg       0.87      0.78      0.82     11795
weighted avg       0.94      0.94      0.94     11795

Confusion Matrix:
 [[10410   198]
 [  492   695]]
```

Accuracy

The tuned LGBM model produced an overall accuracy of 94.15%, indicating that the model correctly predicted 94.15% of the evaluation data. However, due to the imbalanced nature of the dataset, accuracy is not the most important metric in determining the performance of a model.

F1-Score

The F1 score for the positive class was 68.83%, which is considerably lower than the overall accuracy score.

Precision and Recall

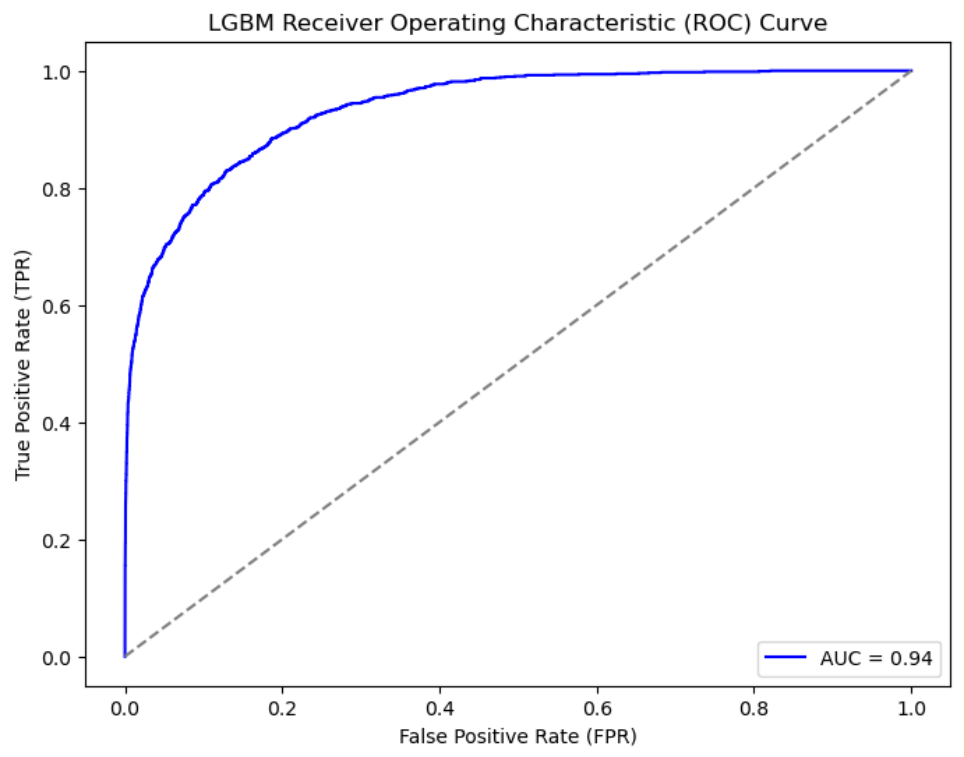
For the negative (majority) class, the model has a high precision and recall of 0.98, leading to a high F1 score for the negative class.

For the positive (minority) class, the model has a much lower precision and recall compared to the negative class. Which means it the model correctly predicts the positive class 78% of the time but only detects 59% (recall) of all actual positive class instances.

Classification Report

The macro average F1 score (which gives equal weight to both the positive and negative class) is 0.82, and the weighted average (which accounts for the support of each class) is 0.94. These scores suggest that overall the model performs well but is biased towards the negative class.

ROC Curve for the tuned LGBM model



The area under the curve (AUC) is 0.94, indicating that the model is performing quite well at distinguishing positive class instances to negative class instances.

However, the ROC curve does not directly address the class imbalance problem, making the curve misleading to the model's performance as F1 scores cannot be directly shown in a traditional ROC curve.

The ExpiredHospital 0.4 decision threshold predictions were then extracted as a .csv file for Kaggle submission:

```
best_lgbm_model = best_model
trained_feature_order = best_lgbm_model.feature_name_
unknown_healthcare_data_reordered = unknown_healthcare_data[trained_feature_order]

lgbm_probabilities = best_lgbm_model.predict_proba(unknown_healthcare_data_reordered)[:, 1]
lgbm_predictions = (lgbm_probabilities > 0.4).astype(int)

lgbm_predictions_df = pd.DataFrame({
    "row ID": ['Row' + str(i) for i in unknown_healthcare_data.index],
    "Predicted-ExpiredHospital": lgbm_predictions
})

lgbm_predictions_df.to_csv("lgbm_predictions_24f_0.4_threshold_early50_ps0.65_halvinggrid_0.25.csv", index=False)
```

Using these parameters, the XGBoost model was trained and tested with the same set up as the LGBM model:

```
param_grid = {
    'objective': ['binary:logistic', 'binary:hinge'],
    'scale_pos_weight': [1, 10, 50],
    'max_depth': [3, 6, 9, 12],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'min_child_weight': [1, 5, 10],
    'subsample': [0.5, 0.7, 0.9],
    'colsample_bytree': [0.5, 0.7, 0.9],
    'booster': ['gbtree', 'dart'],
    'alpha': [0, 1, 5],
    'lambda': [0, 1, 5],
    'n_estimators': [1000, 2000, 3000],
    'gamma': [0, 0.1, 0.5]
```

Results Summary for tuned XGBoost model

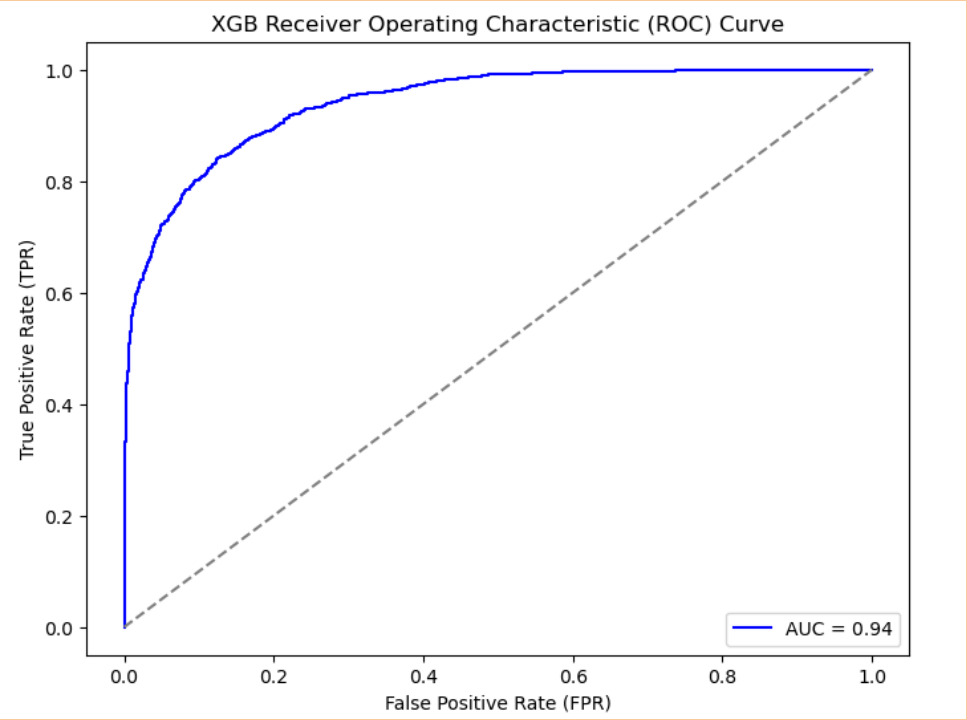
```
Accuracy: 0.9413
F1 Score: 0.6683
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     10608
           1       0.78      0.59      0.67      1187

    accuracy                           0.94     11795
   macro avg       0.87      0.78      0.82     11795
weighted avg       0.94      0.94      0.94     11795

Confusion Matrix:
 [[10406   202]
 [  490   697]]
```

ROC Curve for tuned XGBoost model



Using these parameters, the Catboost model was trained and tested with the same set up as the LGBM model:

```
param_grid = {
    'loss_function': ['Logloss', 'CrossEntropy'],
    'learning_rate': [0.005, 0.01, 0.05, 0.1, 0.2],
    'depth': [4, 6, 8, 10],
    'min_child_samples': [10, 20, 30],
    'subsample': [0.6, 0.7, 0.8, 0.9],
    'rsm': [0.6, 0.7, 0.8, 0.9],
    'boosting_type': ['Ordered', 'Plain'],
    'l2_leaf_reg': [0.1, 1, 3, 5, 10],
    'iterations': [1000, 2000, 3000],
    'border_count': [32, 64, 128, 254]
}
```

Results Summary for tuned Catboost model

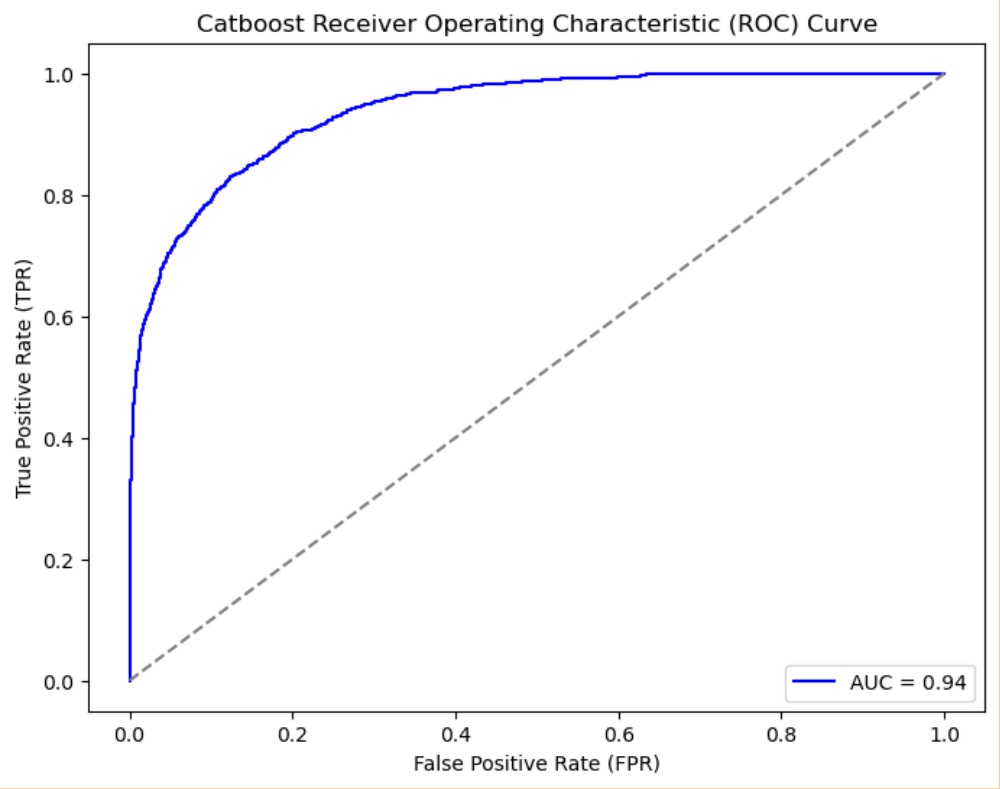
```
Accuracy: 0.9411
F1 Score: 0.6595
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     10608
           1       0.79      0.57      0.66      1187

    accuracy                           0.94     11795
   macro avg       0.87      0.77      0.81     11795
weighted avg       0.94      0.94      0.94     11795

Confusion Matrix:
 [[10427   181]
 [  514   673]]
```

ROC Curve for tuned Catboost model



Stacking Model

Using a stacking ensemble classifier with the tuned gradient boosting models as base leaders and a logistic regression as a meta learner, the ensemble was trained using cross-validation. In the training phase, the three base learners are trained on the training subsets and transform the original features into a prediction space where the meta learner combines these transformed predictions and determines which prediction is most trustworthy in different instances then selects those predictions as the result.

```python
base_learners = [
    ('catboost', best_catboost_model),
    ('lgbm', best_lgbm_model),
    ('xgb', best_xgb_model)
]

stacking_classifier = StackingClassifier(
    estimators=base_learners,
    final_estimator=LogisticRegression(),
    cv=5,
    stack_method='predict_proba',
    n_jobs=-1,
    passthrough=False
)

stacking_classifier.fit(X_train, y_train)

stacking_predictions = stacking_classifier.predict(X_val)
stacking_probs = stacking_classifier.predict_proba(X_val)[:, 1]

stacking_predictions_adjusted = (stacking_probs > 0.4).astype(int)

stacking_accuracy = accuracy_score(y_val, stacking_predictions_adjusted)
stacking_f1 = f1_score(y_val, stacking_predictions_adjusted)
stacking_classification_rep = classification_report(y_val, stacking_predictions_adjusted)
stacking_confusion_mat = confusion_matrix(y_val, stacking_predictions_adjusted)

print(f'Stacking Accuracy: {stacking_accuracy:.4f}')
print(f"Stacking F1 Score: {stacking_f1:.4f}")
print('Stacking Classification Report: \n', stacking_classification_rep)
print('Stacking Confusion Matrix: \n', stacking_confusion_mat)
```

Results Summary for Stacking model

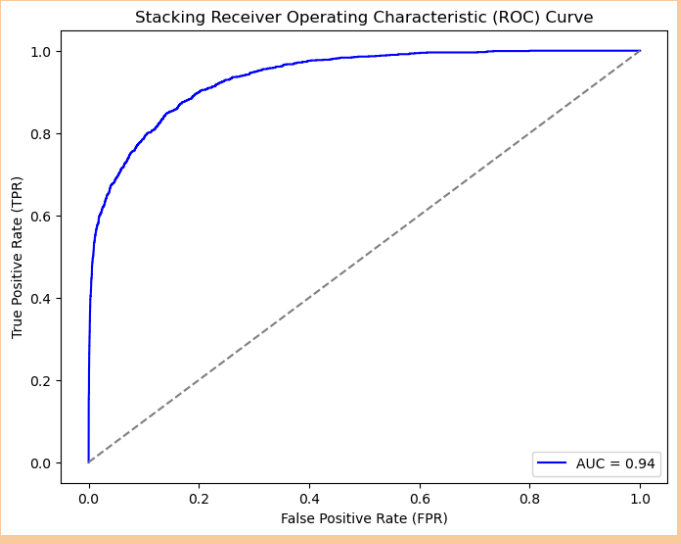
```
Stacking Accuracy: 0.9438
Stacking F1 Score: 0.6693
Stacking Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     10608
           1       0.82      0.57      0.67      1187

    accuracy                           0.94     11795
   macro avg       0.89      0.78      0.82     11795
weighted avg       0.94      0.94      0.94     11795

Stacking Confusion Matrix:
 [[10461   147]
 [  516   671]]
```

ROC Curve for Stacking model



2. Conclusion & Kaggle Submission

In conclusion, after preprocessing the data via filling in missing values and encoding the categorical data into numerical data and training on several classifiers, here is a summary of all the model's performances:

	Accuracy %	Precision (Negative class) %	Precision (Positive class) %	Recall (Negative class) %	Recall (Positive class) %	ROC-AUC Score %	F1-Score %
RF	93.63	94	84	99	45	92	58.94
DT	90.89	94	55	96	49	72	51.73
XGB	94.01	95	80	98	54	93	64.49
LGBM	94.23	95	83	99	54	93	65.17
Catboost	94.13	95	83	99	53	93	64.40
SVM	90.72	91	83	100	10	83	17.62
GNB	89.98	94	50	95	44	86	46.85
LogReg	91.78	93	73	99	29	77	41.64
NN	93.80	95	80	99	51	92	62.38
KNN	92.01	94	67	98	40	81	50.11
Stacking	94.21	95	82	99	54	94	65.35
XGB**	94.13	96	78	98	59	94	66.83
LGBM**	**94.15**	**95**	**78**	**98**	**59**	**94**	**66.83**
Catboost**	94.11	95	79	98	57	94	65.95
Stacking**	**94.38**	**95**	**82**	**99**	**57**	**94**	**66.93**

** refers to tuned models.

The Stacking classifier, incorporating the LGBM, XGB and Catboost models, performed the best on the training and validation sets determined by the accuracy and F1-score metrics.

Re-sampling (to reduce class imbalance and dimensionality reduction techniques (to reduce the noise in data) only resulted in lesser performing models.

Using the the tuned XGB, LGBM, Catboost & Stacking models, predictions were made for the ExpiredHospital on the unknown dataset to then submit to the Kaggle competition. The highest score achieved was 0.84615 (84.615%) accuracy on the unknown dataset using the LightGBM model:

For reference, the tuned XGBoost model scored a 0.8375 (83.75%) accuracy, the tuned Catboost model scored a 0.83015(83.015%) accuracy and the stacking model scored a 0.83207 (83.207%) accuracy for their predictions on the unknown dataset on their Kaggle Submissions.

3. APPENDIX

```python
import pandas as pd

healthcare_data = pd.read_csv('Assignment3-Healthcare-Dataset.csv')
unknown_healthcare_data = pd.read_csv('Assignment3-Unknown-Dataset.csv')
```

```python
missing_val = healthcare_data.isnull().sum()
missing_val = missing_val[missing_val > 0].sort_values(ascending=False)
missing_perc = (missing_val / len(healthcare_data)) * 100
missing_data = pd.DataFrame({'Missing Values': missing_val, 'Percentage (%)': missing_perc})

print(missing_data)
```

```python
missing_val = unknown_healthcare_data.isnull().sum()
missing_val = missing_val[missing_val > 0].sort_values(ascending=False)
missing_perc = (missing_val / len(unknown_healthcare_data)) * 100
missing_data = pd.DataFrame({'Missing Values': missing_val, 'Percentage (%)': missing_perc})

print(missing_data)
```

```python
for i, row in healthcare_data.iterrows():
    if row['AdmitDiagnosis'] == 'NEWBORN':
        healthcare_data.at[i, 'age'] = 0

for i, row in unknown_healthcare_data.iterrows():
    if row['AdmitDiagnosis'] == 'NEWBORN':
        unknown_healthcare_data.at[i, 'age'] = 0
```

```python
healthcare_data.loc[healthcare_data['age'] < 18, 'marital_status'] = 'SINGLE'

unknown_healthcare_data.loc[unknown_healthcare_data['age'] < 18, 'marital_status'] = 'SINGLE'
```

```python
import numpy as np

for i, row in healthcare_data.iterrows():
    if pd.isna(row['LOSgroupNum']):
        if row['LOSdays'] < 4:
            healthcare_data.at[i, 'LOSgroupNum'] = 1
        elif row['LOSdays'] == 4:
            healthcare_data.at[i, 'LOSgroupNum'] = 1.5
        elif row['LOSdays'] < 8:
            healthcare_data.at[i, 'LOSgroupNum'] = 2
        elif row['LOSdays'] == 8:
            healthcare_data.at[i, 'LOSgroupNum'] = 2.5
        elif row['LOSdays'] < 12:
            healthcare_data.at[i, 'LOSgroupNum'] = 3
        elif row['LOSdays'] == 12:
            healthcare_data.at[i, 'LOSgroupNum'] = 3.5
        elif row['LOSdays'] > 12:
            healthcare_data.at[i, 'LOSgroupNum'] = 4

```

```
for i, row in unknown_healthcare_data.iterrows():
    if pd.isna(row['LOSgroupNum']):
        if row['LOSdays'] < 4:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 1
        elif row['LOSdays'] == 4:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 1.5
        elif row['LOSdays'] < 8:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 2
        elif row['LOSdays'] == 8:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 2.5
        elif row['LOSdays'] < 12:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 3
        elif row['LOSdays'] == 12:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 3.5
        elif row['LOSdays'] > 12:
            unknown_healthcare_data.at[i, 'LOSgroupNum'] = 4
```

```
widowed_median_age = healthcare_data[healthcare_data['marital_status'] == 'WIDOWED']['age'].median()
print('WIDOWED: ', widowed_median_age)
widowed_median_age = unknown_healthcare_data[unknown_healthcare_data['marital_status'] == 'WIDOWED']['age'].median()
print('WIDOWED-Unknown: ', widowed_median_age)
single_median_age = healthcare_data[healthcare_data['marital_status'] == 'SINGLE']['age'].median()
print('SINGLE: ', single_median_age)
single_median_age = unknown_healthcare_data[unknown_healthcare_data['marital_status'] == 'SINGLE']['age'].median()
print('SINGLE-Unknown: ', single_median_age)
```

```
median_age = healthcare_data['age'].median()

for i, row in healthcare_data.iterrows():
    if pd.isna(row['age']):
        if row['marital_status'] == 'SINGLE':
            imputed_age = int(median_age - 10)
        elif row['marital_status'] == 'WIDOWED':
            imputed_age = int(median_age + 10)
        else:
            imputed_age = int(median_age)
        healthcare_data.at[i, 'age'] = imputed_age

for i, row in unknown_healthcare_data.iterrows():
    if pd.isna(row['age']):
        if row['marital_status'] == 'SINGLE':
            imputed_age = int(median_age - 10)
        elif row['marital_status'] == 'WIDOWED':
            imputed_age = int(median_age + 10)
        else:
            imputed_age = int(median_age)
        unknown_healthcare_data.at[i, 'age'] = imputed_age
```

```
healthcare_data['NumCallouts'].fillna(healthcare_data['NumCallouts'].median(), inplace=True)

unknown_healthcare_data['NumCallouts'].fillna(unknown_healthcare_data['NumCallouts'].median(), inplace=True)
```

```
healthcare_data['marital_status'].fillna('NA', inplace=True)
healthcare_data['religion'].fillna('NA', inplace=True)
healthcare_data['AdmitDiagnosis'].fillna('NA', inplace=True)

unknown_healthcare_data['marital_status'].fillna('NA', inplace=True)
unknown_healthcare_data['religion'].fillna('NA', inplace=True)
unknown_healthcare_data['AdmitDiagnosis'].fillna('NA', inplace=True)
```

```
healthcare_data['AdmitDiagnosis'] = healthcare_data['AdmitDiagnosis'].str.strip()
unknown_healthcare_data['AdmitDiagnosis'] = unknown_healthcare_data['AdmitDiagnosis'].str.strip()
healthcare_data.columns = [col.replace(' ', '_').replace(':', '').replace('[', '').replace(
    ']', '') for col in healthcare_data.columns]
unknown_healthcare_data.columns = [col.replace(' ', '_').replace(':', '').replace('[', '').replace(
    ']', '') for col in unknown_healthcare_data.columns]
```

```python
from sklearn.model_selection import train_test_split

X = healthcare_data.drop('ExpiredHospital', axis=1)
y = healthcare_data['ExpiredHospital']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)

(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```

```python
X_train_hot = pd.get_dummies(X_train)
X_val_hot = pd.get_dummies(X_val)
unknown_healthcare_data_hot = pd.get_dummies(unknown_healthcare_data)

X_val_hot = X_val_hot.reindex(columns=X_train_hot.columns)
```

```python
from sklearn.preprocessing import LabelEncoder

combined = pd.concat([X_train, X_val, unknown_healthcare_data], axis=0)

categorical_cols = combined.select_dtypes(include=[object, 'category']).columns
for column in categorical_cols:
    combined[column] = LabelEncoder().fit_transform(combined[column])

train_len = len(X_train)
val_len = len(X_val)

X_train = combined.iloc[:train_len]
X_val = combined.iloc[train_len:train_len+val_len]
unknown_healthcare_data = combined.iloc[train_len+val_len:]
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score

rf_model = RandomForestClassifier(random_state=42)

rf_model.fit(X_train, y_train)

predictions = rf_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

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

probabilities = rf_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('RF Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_hot, y_train)

predictions = rf_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```
probabilities = rf_model.predict_proba(X_val_hot)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('RF Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(random_state=42)

dt_model.fit(X_train, y_train)

predictions = dt_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```
probabilities = dt_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('DT Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_hot, y_train)

predictions = dt_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```
probabilities = dt_model.predict_proba(X_val_hot)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('DT Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from xgboost import XGBClassifier

xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train)

predictions = xgb_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = xgb_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('XGB Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from lightgbm import LGBMClassifier

lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_train, y_train)

predictions = lgbm_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = lgbm_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('LGBM Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from catboost import CatBoostClassifier

catboost_model = CatBoostClassifier(random_state=42)
catboost_model.fit(X_train, y_train)

predictions = catboost_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = catboost_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('Catboost Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
catboost_model = CatBoostClassifier(random_state=42)
catboost_model.fit(X_train_hot, y_train)

predictions = catboost_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = catboost_model.predict_proba(X_val_hot)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('Catboost Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from sklearn.svm import SVC

svm_model = SVC(random_state=42)
svm_model.fit(X_train_hot, y_train)

predictions = svm_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
distances = svm_model.decision_function(X_val_hot)

fpr, tpr, thresholds = roc_curve(y_val, distances)

auc = roc_auc_score(y_val, distances)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('SVM Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from sklearn.naive_bayes import GaussianNB

gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)

predictions = gnb_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = gnb_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('GNB Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from sklearn.linear_model import LogisticRegression

logreg_model = LogisticRegression(random_state=42)
logreg_model.fit(X_train_hot, y_train)

predictions = logreg_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = logreg_model.predict_proba(X_val_hot)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('LogReg Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from sklearn.neural_network import MLPClassifier

nn_model = MLPClassifier(random_state=42)
nn_model.fit(X_train_hot, y_train)

predictions = nn_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = nn_model.predict_proba(X_val_hot)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('NN Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(algorithm='brute')
knn_model.fit(X_train_hot, y_train)

predictions = knn_model.predict(X_val_hot)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = knn_model.predict_proba(X_val_hot)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('KNN Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from sklearn.ensemble import StackingClassifier

stacking_model = StackingClassifier(estimators=[
    ('xgb', xgb_model),
    ('lgbm', lgbm_model),
    ('catboost', catboost_model)], final_estimator=LogisticRegression())
stacking_model.fit(X_train, y_train)
predictions = stacking_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = stacking_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('Stacking Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_hot_scaled = scaler.fit_transform(X_train_hot)
X_val_hot_scaled = scaler.transform(X_val_hot)

X_train_hot_scaled = np.nan_to_num(X_train_hot_scaled, copy=False)
X_val_hot_scaled = np.nan_to_num(X_val_hot_scaled, copy=False)

n_components = 10
pca = PCA(n_components=n_components)

X_train_hot_pca = pca.fit_transform(X_train_hot_scaled)
X_val_hot_pca = pca.transform(X_val_hot_scaled)
```

```python
from sklearn.neural_network import MLPClassifier

nn_model = MLPClassifier(random_state=42)
nn_model.fit(X_train_hot_pca, y_train)

predictions = nn_model.predict(X_val_hot_pca)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
probabilities = nn_model.predict_proba(X_val_hot_pca)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('NN Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled_smote, y_resampled_smote = smote.fit_resample(X_train, y_train)

lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_resampled_smote, y_resampled_smote)

predictions = lgbm_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```
probabilities = lgbm_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('LGBM Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```
from imblearn.over_sampling import ADASYN

adasyn = ADASYN(random_state=42)
X_resampled_ada, y_resampled_ada = adasyn.fit_resample(X_train, y_train)

lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_resampled_ada, y_resampled_ada)

predictions = lgbm_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```
probabilities = lgbm_model.predict_proba(X_val)[:, 1]

fpr, tpr, thresholds = roc_curve(y_val, probabilities)

auc = roc_auc_score(y_val, probabilities)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('LGBM Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```
from imblearn.under_sampling import RandomUnderSampler

ros = RandomUnderSampler(random_state=42)
X_resampled_ros, y_resampled_ros = ros.fit_resample(X_train, y_train)

lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_resampled_ros, y_resampled_ros)

predictions = lgbm_model.predict(X_val)

accuracy = accuracy_score(y_val, predictions)
f1 = f1_score(y_val, predictions)
classification_rep = classification_report(y_val, predictions)
confusion_mat = confusion_matrix(y_val, predictions)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print(f'Classification Report: \n{classification_rep}')
print(f'Confusion Matrix: \n{confusion_mat}')
```

```python
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV

param_grid = {
    'objective': ['binary'],
    'class_weight': ['balanced', None],
    'num_leaves': [31, 40, 50, 60, 70],
    'learning_rate': [0.005, 0.01, 0.05, 1.0],
    'max_depth': [-1, 6, 8, 10],
    'min_child_samples': [10, 20, 30, 40],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
    'boosting_type': ['gbdt', 'dart'],
    'reg_alpha': [0, 0.1, 0.5, 1],
    'reg_lambda': [0, 1, 5, 10, 15],
    'n_estimators': [500, 1000, 2000, 3000]
}

lgbm_model = LGBMClassifier(metric='binary_logloss',
                            random_state=42,
                            n_jobs=-1)


grid_search = HalvingGridSearchCV(lgbm_model,
                                  param_grid,
                                  cv=3,
                                  verbose=2,
                                  n_jobs=-1,
                                  factor=2)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
unseen_probs = best_model.predict_proba(unknown_healthcare_data)[:, 1]
high_confidence_indices = np.where(unseen_probs > 0.65)[0]
pseudo_labels = (unseen_probs[high_confidence_indices] > 0.5).astype(int)
pseudo_labeled_data = unknown_healthcare_data.iloc[high_confidence_indices]
X_train_augmented = pd.concat([X_train, pseudo_labeled_data])
y_train_augmented = np.concatenate([y_train, pseudo_labels])
best_model.fit(X_train_augmented, y_train_augmented)

top_24 = pd.DataFrame({
    'feature': X_train.columns,
    'importance': best_model.feature_importances_
}).nlargest(24, 'importance')['feature'].tolist()

X_train_ = X_train[top_24]
X_val_ = X_val[top_24]

best_model.fit(X_train_, y_train,
               eval_set=[(X_val_, y_val)],
               eval_metric='binary_logloss',
               early_stopping_rounds=50,
               verbose=True)

probs = best_model.predict_proba(X_val_)[:, 1]
predictions_adjusted = (probs > 0.4).astype(int)

accuracy = accuracy_score(y_val, predictions_adjusted)
f1 = f1_score(y_val, predictions_adjusted)
classification_rep = classification_report(y_val, predictions_adjusted)
confusion_mat = confusion_matrix(y_val, predictions_adjusted)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print('Classification Report: \n', classification_rep)
print('Confusion Matrix: \n', confusion_mat)
```

```python
fpr, tpr, thresholds = roc_curve(y_val, probs)

auc = roc_auc_score(y_val, probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('LGBM Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
feature_importances = best_model.feature_importances_

features = pd.DataFrame({
    'Feature': X_train_.columns,
    'Importance': feature_importances
})

features = features.sort_values(by='Importance', ascending=False)
print(features)

plt.figure(figsize=(10, 8))
plt.barh(features['Feature'], features['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances')
plt.gca().invert_yaxis()
plt.show()
```

```python
best_lgbm_model = best_model
trained_feature_order = best_lgbm_model.feature_name_
unknown_healthcare_data_reordered = unknown_healthcare_data[trained_feature_order]

lgbm_probabilities = best_lgbm_model.predict_proba(unknown_healthcare_data_reordered)[:, 1]
lgbm_predictions = (lgbm_probabilities > 0.4).astype(int)

lgbm_predictions_df = pd.DataFrame({
    "row ID": ['Row' + str(i) for i in unknown_healthcare_data.index],
    "Predicted-ExpiredHospital": lgbm_predictions
})

lgbm_predictions_df.to_csv("lgbm_predictions_24f_0.4_threshold_early50_ps0.65_halvinggrid_0.25.csv", index=False)
```

```python
param_grid = {
    'loss_function': ['Logloss', 'CrossEntropy'],
    'learning_rate': [0.005, 0.01, 0.05, 0.1, 0.2],
    'depth': [4, 6, 8, 10],
    'min_child_samples': [10, 20, 30],
    'subsample': [0.6, 0.7, 0.8, 0.9],
    'rsm': [0.6, 0.7, 0.8, 0.9],
    'boosting_type': ['Ordered', 'Plain'],
    'l2_leaf_reg': [0.1, 1, 3, 5, 10],
    'iterations': [1000, 2000, 3000],
    'border_count': [32, 64, 128, 254]
}

catboost_model = CatBoostClassifier(loss_function='Logloss',
                                    eval_metric='Logloss',
                                    verbose=200,
                                    random_seed=42,
                                    thread_count=-1)

grid_search = HalvingGridSearchCV(catboost_model,
                                  param_grid,
                                  cv=3,
                                  verbose=2,
                                  n_jobs=-1,
                                  factor=2)

grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
unseen_probs = best_model.predict_proba(unknown_healthcare_data)[:, 1]
high_confidence_indices = np.where(unseen_probs > 0.65)[0]
pseudo_labels = (unseen_probs[high_confidence_indices] > 0.5).astype(int)
pseudo_labeled_data = unknown_healthcare_data.iloc[high_confidence_indices]
X_train_augmented = pd.concat([X_train, pseudo_labeled_data])
y_train_augmented = np.concatenate([y_train, pseudo_labels])
best_model.fit(X_train_augmented, y_train_augmented)

top_24 = pd.DataFrame({
    'feature': X_train.columns,
    'importance': best_model.get_feature_importance()
}).nlargest(24, 'importance')['feature'].tolist()

X_train_ = X_train[top_24]
X_val_ = X_val[top_24]

best_model.fit(X_train_, y_train,
               eval_set=[(X_val_, y_val)],
               early_stopping_rounds=50,
               verbose=True)

probs = best_model.predict_proba(X_val_)[:, 1]
predictions_adjusted = (probs > 0.4).astype(int)

accuracy = accuracy_score(y_val, predictions_adjusted)
f1 = f1_score(y_val, predictions_adjusted)
classification_rep = classification_report(y_val, predictions_adjusted)
confusion_mat = confusion_matrix(y_val, predictions_adjusted)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print('Classification Report: \n', classification_rep)
print('Confusion Matrix: \n', confusion_mat)
```

```python
feature_importances = best_model.feature_importances_

features = pd.DataFrame({
    'Feature': X_train_.columns,
    'Importance': feature_importances
})

features = features.sort_values(by='Importance', ascending=False)
print(features)

plt.figure(figsize=(10, 8))
plt.barh(features['Feature'], features['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Catboost Feature Importances')
plt.gca().invert_yaxis()
plt.show()
```

```python
fpr, tpr, thresholds = roc_curve(y_val, probs)

auc = roc_auc_score(y_val, probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('Catboost Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
best_catboost_model = best_model
trained_feature_order = best_catboost_model.feature_names_
unknown_healthcare_data_reordered = unknown_healthcare_data[trained_feature_order]

catboost_probabilities = best_catboost_model.predict_proba(unknown_healthcare_data_reordered)[:, 1]
catboost_predictions = (catboost_probabilities > 0.4).astype(int)

catboost_predictions_df = pd.DataFrame({
    "row ID": ['Row' + str(i) for i in unknown_healthcare_data.index],
    "Predicted-ExpiredHospital": catboost_predictions
})

catboost_predictions_df.to_csv("catboost_predictions_24f_0.4_threshold_early50_ps0.65_halvinggrid_0.25.csv", index=False)
```

```python
param_grid = {
    'objective': ['binary:logistic', 'binary:hinge'],
    'scale_pos_weight': [1, 10, 50],
    'max_depth': [3, 6, 9, 12],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'min_child_weight': [1, 5, 10],
    'subsample': [0.5, 0.7, 0.9],
    'colsample_bytree': [0.5, 0.7, 0.9],
    'booster': ['gbtree', 'dart'],
    'alpha': [0, 1, 5],
    'lambda': [0, 1, 5],
    'n_estimators': [1000, 2000, 3000],
    'gamma': [0, 0.1, 0.5]
}

xgb_model = XGBClassifier(objective='binary:logistic',
                          random_state=42,
                          n_jobs=-1,
                          eval_metric='logloss')

grid_search = HalvingGridSearchCV(xgb_model,
                                  param_grid,
                                  cv=3,
                                  verbose=2,
                                  n_jobs=-1,
                                  factor=2)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
unseen_probs = best_model.predict_proba(unknown_healthcare_data)[:, 1]
high_confidence_indices = np.where(unseen_probs > 0.65)[0]
pseudo_labels = (unseen_probs[high_confidence_indices] > 0.5).astype(int)
pseudo_labeled_data = unknown_healthcare_data.iloc[high_confidence_indices]
X_train_augmented = pd.concat([X_train, pseudo_labeled_data])
y_train_augmented = np.concatenate([y_train, pseudo_labels])
best_model.fit(X_train_augmented, y_train_augmented)

top_24 = pd.DataFrame({
    'feature': X_train.columns,
    'importance': best_model.feature_importances_
}).nlargest(24, 'importance')['feature'].tolist()

X_train_ = X_train[top_24]
X_val_ = X_val[top_24]

best_model.fit(X_train_, y_train,
               eval_set=[(X_val_, y_val)],
               early_stopping_rounds=50,
               verbose=True)

probs = best_model.predict_proba(X_val_)[:, 1]
predictions_adjusted = (probs > 0.4).astype(int)

accuracy = accuracy_score(y_val, predictions_adjusted)
f1 = f1_score(y_val, predictions_adjusted)
classification_rep = classification_report(y_val, predictions_adjusted)
confusion_mat = confusion_matrix(y_val, predictions_adjusted)

print(f'Accuracy: {accuracy:.4f}')
print(f"F1 Score: {f1:.4f}")
print('Classification Report: \n', classification_rep)
print('Confusion Matrix: \n', confusion_mat)
```

```python
feature_importances = best_model.feature_importances_

features = pd.DataFrame({
    'Feature': X_train_.columns,
    'Importance': feature_importances
})

features = features.sort_values(by='Importance', ascending=False)
print(features)

plt.figure(figsize=(10, 8))
plt.barh(features['Feature'], features['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('XGBoost Feature Importances')
plt.gca().invert_yaxis()
plt.show()
```

```python
fpr, tpr, thresholds = roc_curve(y_val, probs)

auc = roc_auc_score(y_val, probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('XGB Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
best_xgb_model = best_model
trained_feature_order = best_xgb_model.get_booster().feature_names
unknown_healthcare_data_reordered = unknown_healthcare_data[trained_feature_order]

xgb_probabilities = best_xgb_model.predict_proba(unknown_healthcare_data_reordered)[:, 1]
xgb_predictions = (xgb_probabilities > 0.4).astype(int)

xgb_predictions_df = pd.DataFrame({
    "row ID": ['Row' + str(i) for i in unknown_healthcare_data.index],
    "Predicted-ExpiredHospital": xgb_predictions
})

xgb_predictions_df.to_csv("xgb_predictions_24f_0.4_threshold_early50_ps0.65_halvinggrid_0.25.csv", index=False)
```

```python
base_learners = [
    ('catboost', best_catboost_model),
    ('lgbm', best_lgbm_model),
    ('xgb', best_xgb_model)
]

meta_learner = LogisticRegression()

stacking_classifier = StackingClassifier(
    estimators=base_learners,
    final_estimator=LogisticRegression(),
    cv=5,
    stack_method='predict_proba',
    n_jobs=-1,
    passthrough=False
)

stacking_classifier.fit(X_train, y_train)

stacking_predictions = stacking_classifier.predict(X_val)
stacking_probs = stacking_classifier.predict_proba(X_val)[:, 1]

stacking_predictions_adjusted = (stacking_probs > 0.4).astype(int)

stacking_accuracy = accuracy_score(y_val, stacking_predictions_adjusted)
stacking_f1 = f1_score(y_val, stacking_predictions_adjusted)
stacking_classification_rep = classification_report(y_val, stacking_predictions_adjusted)
stacking_confusion_mat = confusion_matrix(y_val, stacking_predictions_adjusted)

print(f'Stacking Accuracy: {stacking_accuracy:.4f}')
print(f"Stacking F1 Score: {stacking_f1:.4f}")
print('Stacking Classification Report: \n', stacking_classification_rep)
print('Stacking Confusion Matrix: \n', stacking_confusion_mat)
```

```python
fpr, tpr, thresholds = roc_curve(y_val, stacking_probs)

auc = roc_auc_score(y_val, stacking_probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.title('Stacking Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.show()
```

```python
stacking_probs = stacking_classifier.predict_proba(unknown_healthcare_data)[:, 1]
stacking_predictions_adjusted = (stacking_probs > 0.4).astype(int)

stacking_predictions_df = pd.DataFrame({
    "row ID": ['Row' + str(i) for i in unknown_healthcare_data.index],
    "Predicted-ExpiredHospital": stacking_predictions_adjusted
})

stacking_predictions_df.to_csv("stacking_predictions_0.4_threshold.csv", index=False)
```

4. Attribute Summaries

Attribute	Summary
Gender	Gender of the patient (e.g., Male or Female)
Age	Age of the patient
LOSdays	Length of Stay in days
admit_type	Type of admission
admit_location	Location from which the patient was admitted
AdmitDiagnosis	Diagnosis upon admission
Insurance	Type of insurance the patient has
religion	Patient's religion.
marital_status	Marital status of the patient.
ethnicity	Ethnic background of the patient.
NumCallouts	Number of callouts
NumDiagnosis	Number of diagnoses
NumProcs	Number of procedures
AdmitProcedure	Procedure upon admission
NumCPTevents	Number of CPT events
NumInput	Number of inputs.
NumLabs	Number of lab tests
NumMicrolabs	Number of microbiology labs.
NumNotes	Number of notes.
NumOutput	Number of outputs.
NumRx	Number of prescriptions.
NumProcEvents	Number of procedure events.
NumTransfers	Number of transfers.
NumChartEvents	Number of chart events.
ExpiredHospital	Indicates if the patient expired in the hospital (0 = No, 1 = Yes).
TotalNumInteract	Total number of interactions.
LOSgroupNum	Group number based on the length of stay.