**D603 - Machine Learning**

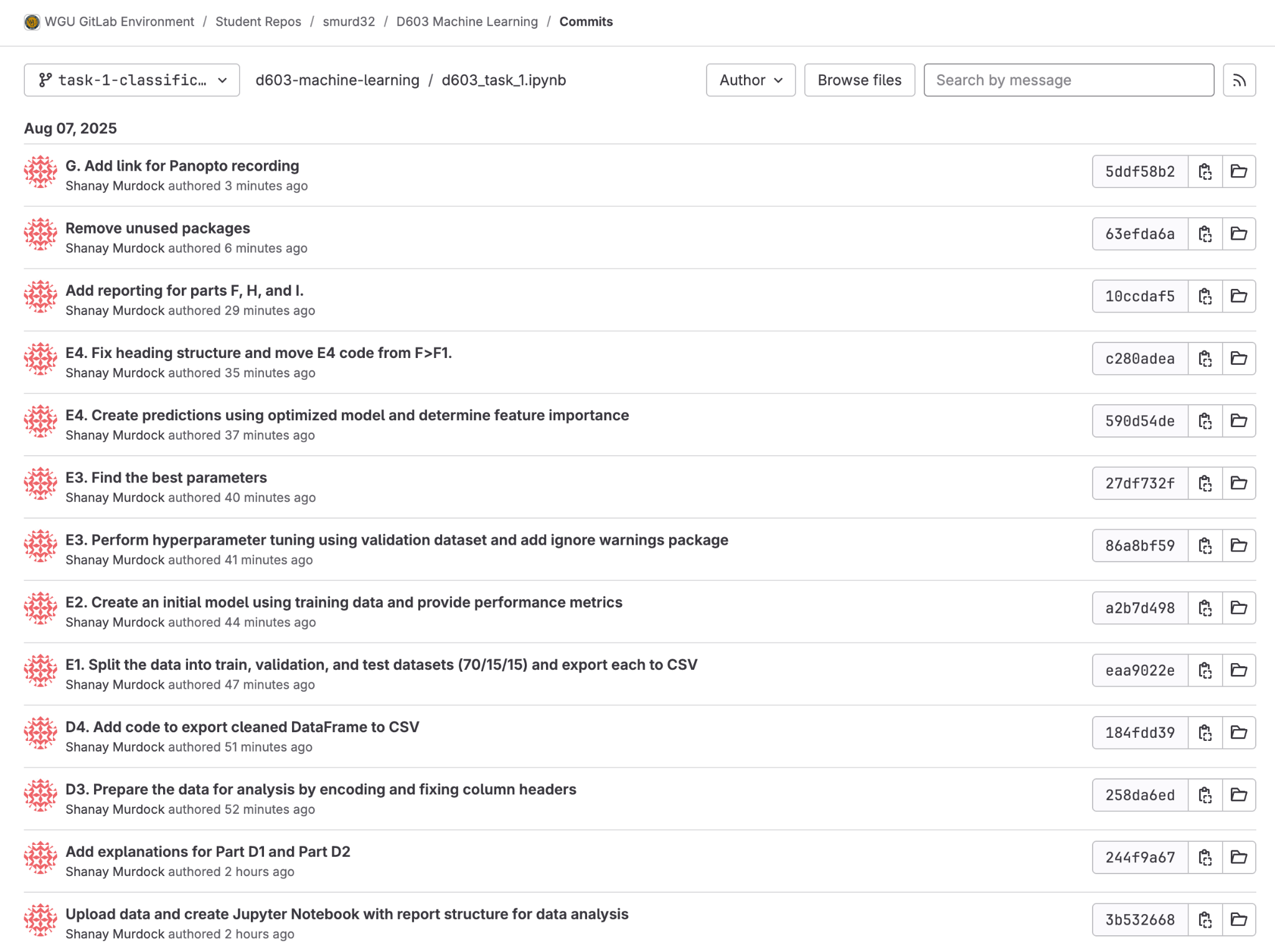
**RFN1 Task 1: Classification Data Mining Models**

Prepared by Shanay Murdock

## A. GitLab Repository

<https://gitlab.com/wgu-gitlab-environment/student-repos/smurd32/d603-machine-learning/-/tree/task-1-classification?ref_type=heads>

Screenshot of commit history:



## B. Purpose of the Data Mining Report

### B1. Proposal of Question

**Business Question:** Can we predict which patients are most likely to be readmitted within 30 days of discharge so that our hospital chain can implement targeted intervention programs to improve patient outcomes and reduce readmission penalties?

### B2. Defined Goal

**Goal:** Develop a Random Forest Classification model that identifies the most important factors influencing the prediction of patient readmission risk, enabling early identification of high-risk patients for targeted intervention programs. By identifying these factors, we can provide targeted intervention initiatives that can improve the patient experience and reduce the financial penalties from the Centers for Medicare and Medicaid Services (CMS). Freeing up that funding can go toward additional hospital resources or medical care initiatives.

## C. Justification of Classification Method

### C1. Explanation of Classification Method

Random Forest Classification works by creating multiple decision trees that each make independent predictions about patient readmission risk. Each tree is trained on a random sample of the data and uses a random subset of available features, such as age, medical conditions, and hospital stay details. The final prediction comes from combining all the individual tree votes--if most trees predict "readmitted," the patient is classified as high-risk for readmission. This ensemble approach reduces errors and provides more reliable predictions than any single decision tree.

Random Forest Classification is the ideal classification method for this medical dataset because it naturally handles the mix of categorical data (like gender and medical conditions) and continuous data (like age and income) without requiring complicated preprocessing. Most importantly for healthcare applications, it provides clear feature importance rankings that show which patient factors most strongly predict readmission risk. This interpretability allows hospital staff to understand why certain patients are high-risk and develop targeted intervention programs.

The expected outcomes include accurate binary predictions (readmitted versus not readmitted), probability scores for risk assessment, and actionable insights about which patient characteristics to monitor most closely.

### C2. Packages or Libraries List

The follow packages and libraries were chosen to complete the Random Forest Classification:

* pandas: Allows data manipulation and analysis.
* re: RegEx package allows easy renaming of variables for consistency.
* numpy: Provides numerical computations.
* RandomForestClassifier from Scikit-Learn: Acts as the main classification algorithm.
* train\_test\_split from Scikit-Learn: Provides data splits for training and testing to prevent overfitting.
* GridSearchCV from Scikit-Learn: Finds optimal hyperparameters automatically using cross-validation.
* StratifiedKFold from Scikit-Learn: A cross-validation technique used to split a dataset into training and testing sets, specifically designed to handle similar proportions of damples from each class as the original dataset.
* cross\_val\_score from Scikit-Learn: Provides a more robust performance estimation than single train/test split.
* accuracy\_score from Scikit-Learn: Provides overall correctness but can be misleading if classes are imbalanced.
* precision\_score from Scikit-Learn: Reduces false alarms, important for resource allocation.
* recall\_score from Scikit-Learn: Measures how good the model is at finding all positive cases that actually exist.
* f1\_score from Sckit-Learn: Balances precision and recall.
* roc\_auc\_score from Scikit-Learn: Determines how well the model distinguishes between readmitted and non-readmitted patients.
* confusion\_matrix from Scikit-Learn: Provides detailed breakdown of prediction errors.
* classification\_report from Scikit-Learn: Single function that provides all performance metrics.
* LabelEncoder from Scikit-Learn: Converts categories to numbers for Random Forest needs.
* OrdinalEncoder from Scikit-Learn: Converts ordinal categories to numbers for Random Forest needs.
* matplotlib: Visualizations
* seaborn: Statistical visualizations

## D. Data Preparation

### D1. Data Preprocessing

**Data Preprocessing Goal:** Convert categorical variables to numerical format to create compatibility with Random Forest algorithm requirements.

Random Forest Classification requires all input features to be in numerical format, but the medical\_clean.csv dataset contains several categorical variables such as Gender (Male/Female/Nonbinary), Marital Status (Divorced/Married/Widowed), and several binary medical conditions (Yes/No) for high blood pressure, stroke, diabetes, etc.

To address this requirement, I will use mapping and Scikit-Learn's LabelEncoder and OrdinalEncoder to transform these text-based categorical variables into numeric representations. For example, I will use mapping to replace the 'Yes' and 'No' to '1' and '0' for any binary categories, OrdinalEncoder will be used to encode ordinal categories to preserve meaning in categorical order, and LabelEncoder will be used to encode any nominal categories.

This preprocessing step is essential because Random Forest's underlying decision tree algorithm can only process numerical data when making splits and calculating feature importance scores. By encoding categorical variables appropriately, it is ensured that the Random Forest model can effectively utilize all available patient information to make accurate readmission predictions while maintaining the interpretability of results.

### D2. Dataset Variables

Below are the initial dataset variables retained for the Random Forest Classification analysis to predict patient readmission status:

#### Target Variable:

* ReAdmis: Patient readmitted within 30 days (Yes/No); Categorical (Binary)

#### Predictor Variables:

**Patient Demographics:**

* Age: Patient age in years; Continuous
* Gender: Patient gender (Male/Female/Nonbinary); Categorial
* Income: Annual income of patient; Continuous
* Marital: Marital status of patient (Single/Married/Divorced/Widowed/Never Married); Categorical
* Children: Number of children in household; Continuous
* Area: Area type (Rural/Suburban/Urban); Categorical

**Clinical Measurements:**

* VitD\_levels: Vitamin D levels (ng/mL); Continuous
* Doc\_visits: Number of physician visits during hospitalization; Continous  
  Full\_meals\_eaten: Number of full meals eaten during hospitalization; Continuous
* VitD\_supp: Number of vitamin D supplements administered; Continuous
* Initial\_days: Length of initial hospital stays (days); Continuous

**Medical Conditions (Binary Yes/No):**

* HighBlood: High blood pressure diagnosis; Categorical (Binary)
* Stroke: Previous stroke history; Categorical (Binary)
* Overweight: Patient considered overweight; Categorical (Binary)
* Arthritis: Arthritis diagnosis; Categorical (Binary)
* Diabetes: Diabetes diagnosis; Categorical (Binary)
* BackPain: Chronic back pain diagnosis; Categorical (Binary)
* Anxiety: Anxiety disorder diagnosis; Categorical (Binary)
* Asthma: Asthma diagnosis; Categorical (Binary)

**Operational Factors:**

* Initial\_admin: Type of admission (Emergency/Elective/Observation); Categorical
* Services: Primary service received (Blood work/IV/CT scan/MRI); Categorical
* Complication\_risk: Patient complication risk level (High/Medium/Low); Categorical (Ordinal)
* Soft\_drink: Habitually drinks 3+ sodas daily (Yes/No); Categorical (Binary)

**Financial Variables:**

* TotalCharge: Average daily charge amount; Continuous
* Additional\_charges: Average additional charges; Continuous

Summary: 15 categorical variables; 10 continuous variables

### D3. Steps for Analysis

| *# Core packages:* import pandas as pd import re from sklearn.ensemble import RandomForestClassifier from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, classification\_report from sklearn.preprocessing import LabelEncoder, OrdinalEncoder import matplotlib.pyplot as plt import seaborn as sns import warnings warnings.filterwarnings('ignore')  *# Load the dataset* df = pd.read\_csv('medical\_clean.csv') *# Preview the dataset* df.head() |
| --- |

To prepare the data for the Random Forest classification model, I used the .read\_csv() function from pandas to load the CSV and convert it into a DataFrame, making it easier to analyze. The .head() function loads the first five observations of the DataFrame to verify it loaded correctly and provide a full preview of the data.

| *# Select relevant columns to keep* columns\_to\_keep = ['ReAdmis', 'Age', 'Gender', 'Income', 'Marital',   'Children', 'Area', 'VitD\_levels', 'Doc\_visits',   'Full\_meals\_eaten', 'vitD\_supp', 'Initial\_days',  'HighBlood', 'Stroke', 'Overweight', 'Arthritis',   'Diabetes', 'BackPain', 'Anxiety', 'Asthma',  'Initial\_admin', 'Services', 'Complication\_risk',   'Soft\_drink', 'TotalCharge', 'Additional\_charges']  df = df[columns\_to\_keep] df.info() |
| --- |

Out of the original 50 features, I chose a target variable ('ReAdmis') and 24 potential predictive variables that seemed to have the most to do with what may predict if a patient is readmitted to the hospital after discharge.

**Initial Observations:**

* There are no missing values in any of the columns.
* There is a mix of object, int64, and float64 data types.
* There is one target variable and 24 feature variables out of the initial 50 variables.

| *# Create a function that creates a consistent snake\_case style for all variable names* def to\_snake\_case(name):  """  Convert column name to snake\_case format  """  *# Handle acronyms and special cases first*  name = name.replace('VitD', 'vit\_d')  name = name.replace('UID', 'uid')   *# Insert underscore before uppercase letters that follow lowercase letters*  *# This handles PascalCase like 'HighBlood' -> 'High\_Blood'*  name = re.sub('([a-z0-9])([A-Z])', r'\1\_\2', name)   *# Convert to lowercase*  name = name.lower()   *# Clean up any multiple underscores*  name = re.sub('\_+', '\_', name)   *# Remove leading/trailing underscores*  name = name.strip('\_')   return name  *# Apply snake\_case conversion to all column names* df.columns = [to\_snake\_case(col) for col in df.columns] *# print out new info* df.info() |
| --- |

I found the inconsistency of the column names (despite being a "cleaned" dataset) to be difficult to work with, so I created a function that uses regular expressions to fix the column names and convert them all from a mix of snake case, Pascal case, and other miscellaneous cases to a consistent use of snake\_case.

| *# Categorical variables to label encode* *# (Codecademy, n.d.)* *# Binary variables* binary\_cats = ['re\_admis', 'high\_blood', 'back\_pain', 'stroke', 'overweight', 'arthritis',  'diabetes', 'anxiety', 'asthma', 'soft\_drink']  ordinal\_cats = ['complication\_risk']  nominal\_cats = ['gender', 'marital', 'area', 'initial\_admin', 'services']   *#applying label encoding to each variable by type* *# Binary Categories* for col in binary\_cats:  df[col] = df[col].map({'Yes': 1, 'No': 0})  *# Ordinal Category - order preserved for significance* ordinal\_encoder = OrdinalEncoder(categories=[['Low', 'Medium', 'High']]) df['complication\_risk'] = ordinal\_encoder.fit\_transform(df[['complication\_risk']]).astype(int) print(f"Encoded complication\_risk: Low=0, Medium=1, High=2")  *# Nominal Categories - order not preserved for significance* label\_encoders = {} for col in nominal\_cats:  le = LabelEncoder()  df[col] = le.fit\_transform(df[col])  label\_encoders[col] = le |
| --- |

I separated out the binary, ordinal, and nominal categorical variables and encoded them with a mix of mapping, OrdinalEncoder, and LabelEncoder.

Using DataFrame mapping, I was able to quickly convert binary "Yes" values to 1 and "No" values to 0. OrdinalEncoder allowed me to preserve the order and meaning behind the complication\_risk values as the only ordinal category. Using LabelEncoder for the nominal categorical variables allowed me to maintain interpretability without creating too many variables like pd.get\_dummies() would create.

| df.head() df.info() |
| --- |

**Observations:**

* There are no longer any "object" types. All data has been converted to int64 or float64 as appropriate.

### D4. Cleaned Dataset

| *# Provide a copy of the cleaned dataset* df.to\_csv("cleaned\_medical\_dataset.csv", index=False) |
| --- |

The cleaned dataset is in a separate file named cleaned\_medical\_dataset.csv.

### E. Perform Data Analysis

### E1. Splitting the Data

| *# Separate the features (X) from the target variable (y)* X = df.drop('re\_admis', axis=1) y = df['re\_admis']  print(f"Feature matrix shape: {X.shape}") print(f"Target vector shape: {y.shape}") print(f"Target distribution:\n{y.value\_counts()}") |
| --- |

| *# First split: 70% training data, 30% temporary data (for validation + test)* *# (Codecademy, n.d.) - Training, Validation and Test Dataset* X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(  X, y,  test\_size=0.3, *# 30% will be split equally into validation and test data*  random\_state=42,  stratify=y *# maintain class distribution across splits* )  *# Second split: Split the 30% temp data into 15% validation and 15% test data* X\_val, X\_test, y\_val, y\_test = train\_test\_split(  X\_temp, y\_temp,  test\_size=0.5, *# split original 30% in half*  random\_state=42,  stratify=y\_temp *# maintain class distribution across splits* ) |
| --- |

| *# Verify the splits are correct* total\_samples = len(df) print(f"\nDataset split verification:") print(f"Total samples: {total\_samples}") print(f"Training set: {len(X\_train)} samples ({len(X\_train)/total\_samples:.1%})") print(f"Validation set: {len(X\_val)} samples ({len(X\_val)/total\_samples:.1%})") print(f"Test set: {len(X\_test)} samples ({len(X\_test)/total\_samples:.1%})")  *# Verify class distribution is maintained across splits* print(f"\nClass distribution verification:") print(f"Original: {y.value\_counts(normalize=True).sort\_index()}") print(f"Training: {y\_train.value\_counts(normalize=True).sort\_index()}") print(f"Validation: {y\_val.value\_counts(normalize=True).sort\_index()}") print(f"Test: {y\_test.value\_counts(normalize=True).sort\_index()}") |
| --- |

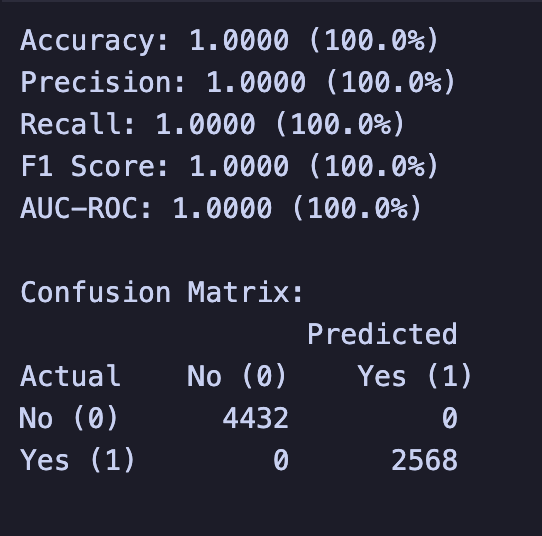
| *# Save the split datasets to separate files* *# Training set* train\_data = pd.concat([X\_train, y\_train], axis=1) train\_data.to\_csv("training\_dataset.csv", index=False)  *# Validation set* val\_data = pd.concat([X\_val, y\_val], axis=1) val\_data.to\_csv("validation\_dataset.csv", index=False)  *# Test set* test\_data = pd.concat([X\_test, y\_test], axis=1) test\_data.to\_csv("test\_dataset.csv", index=False) |
| --- |

### E2. Initial Model Creation

| *# Create initial Random Forest model using the TRAINING dataset and default* *# parameters* rf\_initial = RandomForestClassifier(random\_state=42, n\_jobs=-1)  print(f"Initial Random Forest Parameters:") print(f"- n\_estimators: {rf\_initial.n\_estimators}") print(f"- max\_depth: {rf\_initial.max\_depth}") print(f"- min\_samples\_split: {rf\_initial.min\_samples\_split}") print(f"- min\_samples\_leaf: {rf\_initial.min\_samples\_leaf}") print(f"- max\_features: {rf\_initial.max\_features}") print(f"- bootstrap: {rf\_initial.bootstrap}") |
| --- |

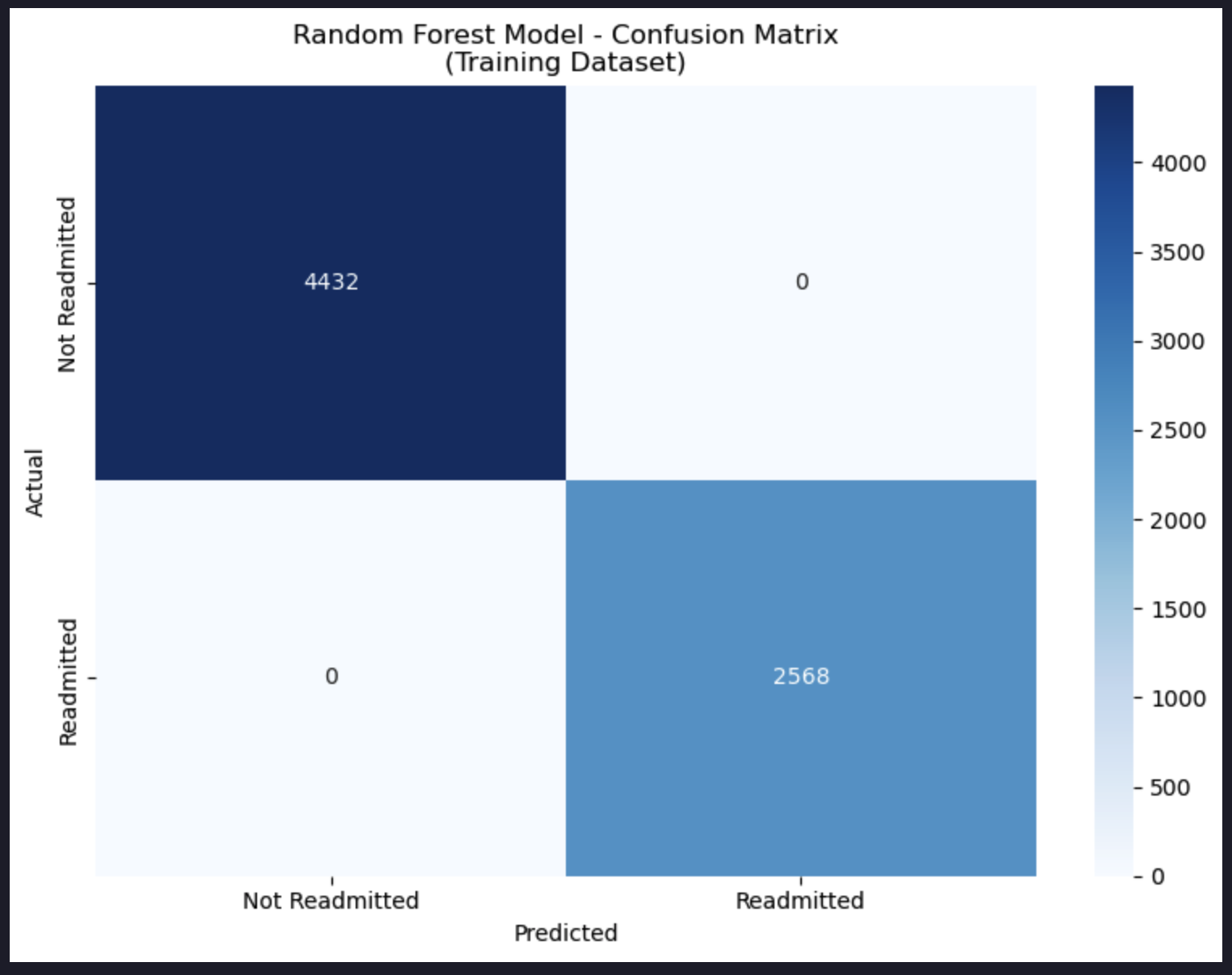
| *# Train the model on the training data* rf\_initial.fit(X\_train, y\_train)  *# Make predictions on the training data* y\_train\_pred = rf\_initial.predict(X\_train) y\_train\_proba = rf\_initial.predict\_proba(X\_train)[:, 1] *# Probabilities for positive class* |
| --- |

| *# Calculate and print all required metrics* *# 1. Accuracy* accuracy = accuracy\_score(y\_train, y\_train\_pred) print(f"Accuracy: {accuracy:.4f} ({accuracy:.1%})")  *# 2. Precision* precision = precision\_score(y\_train, y\_train\_pred) print(f"Precision: {precision:.4f} ({precision:.1%})")  *# 3. Recall* recall = recall\_score(y\_train, y\_train\_pred) print(f"Recall: {recall:.4f} ({recall:.1%})")  *# 4. F1 Score* f1 = f1\_score(y\_train, y\_train\_pred) print(f"F1 Score: {f1:.4f} ({f1:.1%})")  *# 5. AUC-ROC* auc\_roc = roc\_auc\_score(y\_train, y\_train\_proba) print(f"AUC-ROC: {auc\_roc:.4f} ({auc\_roc:.1%})")  *# 6. Confusion Matrix* cm = confusion\_matrix(y\_train, y\_train\_pred) print(f"\nConfusion Matrix:") print(f" Predicted") print(f"Actual No (0) Yes (1)") print(f"No (0) {cm[0,0]:6d} {cm[0,1]:6d}") print(f"Yes (1) {cm[1,0]:6d} {cm[1,1]:6d}") |
| --- |



| *# Additional detailed breakdown* tn, fp, fn, tp = cm.ravel() print(f"\nDetailed Breakdown:") print(f"True Negatives (TN): {tn:,}") print(f"False Positives (FP): {fp:,}") print(f"False Negatives (FN): {fn:,}") print(f"True Positives (TP): {tp:,}")  *# Comprehensive Classification Report* print(f"\nClassification Report:") print(classification\_report(y\_train, y\_train\_pred,  target\_names=["Not Readmitted", "Readmitted"])) |
| --- |

| *# Visualize confusion matrix for TRAINING data* plt.figure(figsize=(8, 6)) sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',  xticklabels=['Not Readmitted', 'Readmitted'],  yticklabels=['Not Readmitted', 'Readmitted']) plt.title('Random Forest Model - Confusion Matrix\n(Training Dataset)') plt.xlabel('Predicted') plt.ylabel('Actual') plt.tight\_layout() plt.show() |
| --- |



### E3. Hyperparameter Turning

**Hyperparameters Selected for Tuning:**

1. n\_estimators: Number of trees in the forest
   * More trees generally improve performance but increase computation time
   * Testing: [50, 100, 200] to balance performance vs efficiency
2. max\_depth: Maximum depth of individual trees
   * Controls overfitting - deeper trees can memorize training data
   * Testing: [10, None] to find optimal complexity
3. min\_samples\_split: Minimum samples required to split internal node
   * Higher values prevent overfitting by requiring more samples per split
   * Testing: [5, 10] from very flexible to more conservative
4. min\_samples\_leaf: Minimum samples required at leaf node
   * Controls overfitting by ensuring leaves have sufficient samples
   * Testing: [2, 4] to balance model complexity
5. max\_features: Number of features considered for best split
   * Controls randomness and prevents overfitting
   * Testing: ['sqrt', 'log2'] - standard Random Forest approaches

| *# Define parameter grid for hyperparameter tuning* param\_grid = {  'n\_estimators': [50, 100, 200], *# Number of trees*  'max\_depth': [10, None], *# Tree depth (None = unlimited)*  'min\_samples\_split': [5, 10], *# Min samples to split node*  'min\_samples\_leaf': [2, 4], *# Min samples in leaf*  'max\_features': ['sqrt', 'log2'] *# Features per tree* } |
| --- |

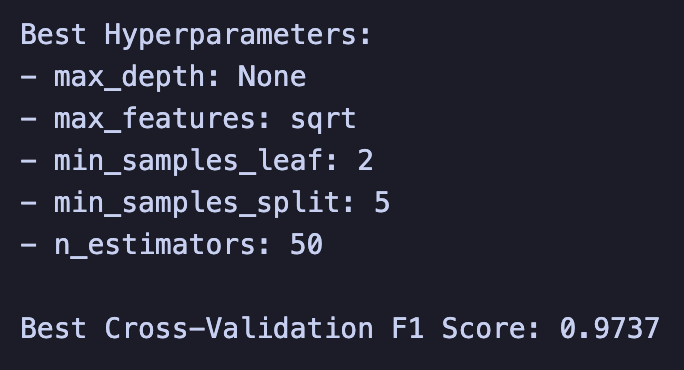
These hyperparameters were chosen because they directly control:

* Model complexity and overfitting (max\_depth, min\_samples\_split, min\_samples\_leaf)
* Ensemble diversity and performance (n\_estimators, max\_features)
* Computational efficiency vs accuracy trade-offs
* Are most impactful for Random Forest performance in medical prediction

| *# Create Random Forest for hyperparameter tuning* rf\_tuning = RandomForestClassifier(random\_state=42, n\_jobs=-1)  *# Define the 5-fold cross-validation strategy* cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42) |
| --- |

| *# Perform GridSearchCV with 5-fold cross validation* print(f"Total parameter combinations to test: {len(param\_grid['n\_estimators']) \* len(param\_grid['max\_depth']) \* len(param\_grid['min\_samples\_split']) \* len(param\_grid['min\_samples\_leaf']) \* len(param\_grid['max\_features'])}")  grid\_search = GridSearchCV(  estimator=rf\_tuning,  param\_grid=param\_grid,  cv=cv, *# 5-fold cross validation*  scoring='f1', *# Optimize for F1-score (balances precision/recall)*  n\_jobs=-1,  verbose=1 ) |
| --- |

| *# Fit the grid search on VALIDATION data* grid\_search.fit(X\_val, y\_val) *# Display best parameters* print("Best Hyperparameters:") best\_params = grid\_search.best\_params\_ for param, value in best\_params.items():  print(f"- {param}: {value}")  print(f"\nBest Cross-Validation F1 Score: {grid\_search.best\_score\_:.4f}") |
| --- |

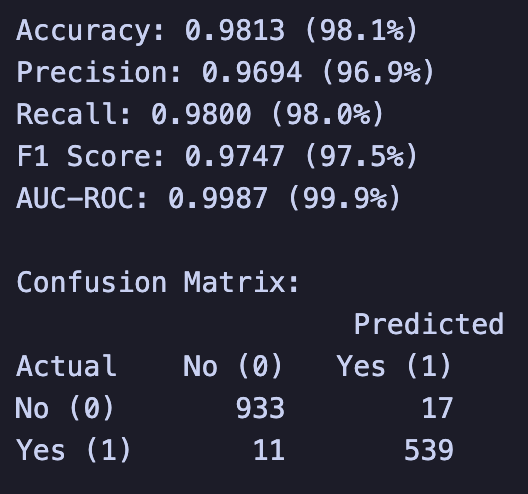


| *# Get the best model* rf\_optimized = grid\_search.best\_estimator\_ |
| --- |

### E4. Predictions

| *# Make predictions on TEST dataset* y\_test\_pred = rf\_optimized.predict(X\_test) y\_test\_proba = rf\_optimized.predict\_proba(X\_test)[:, 1] |
| --- |

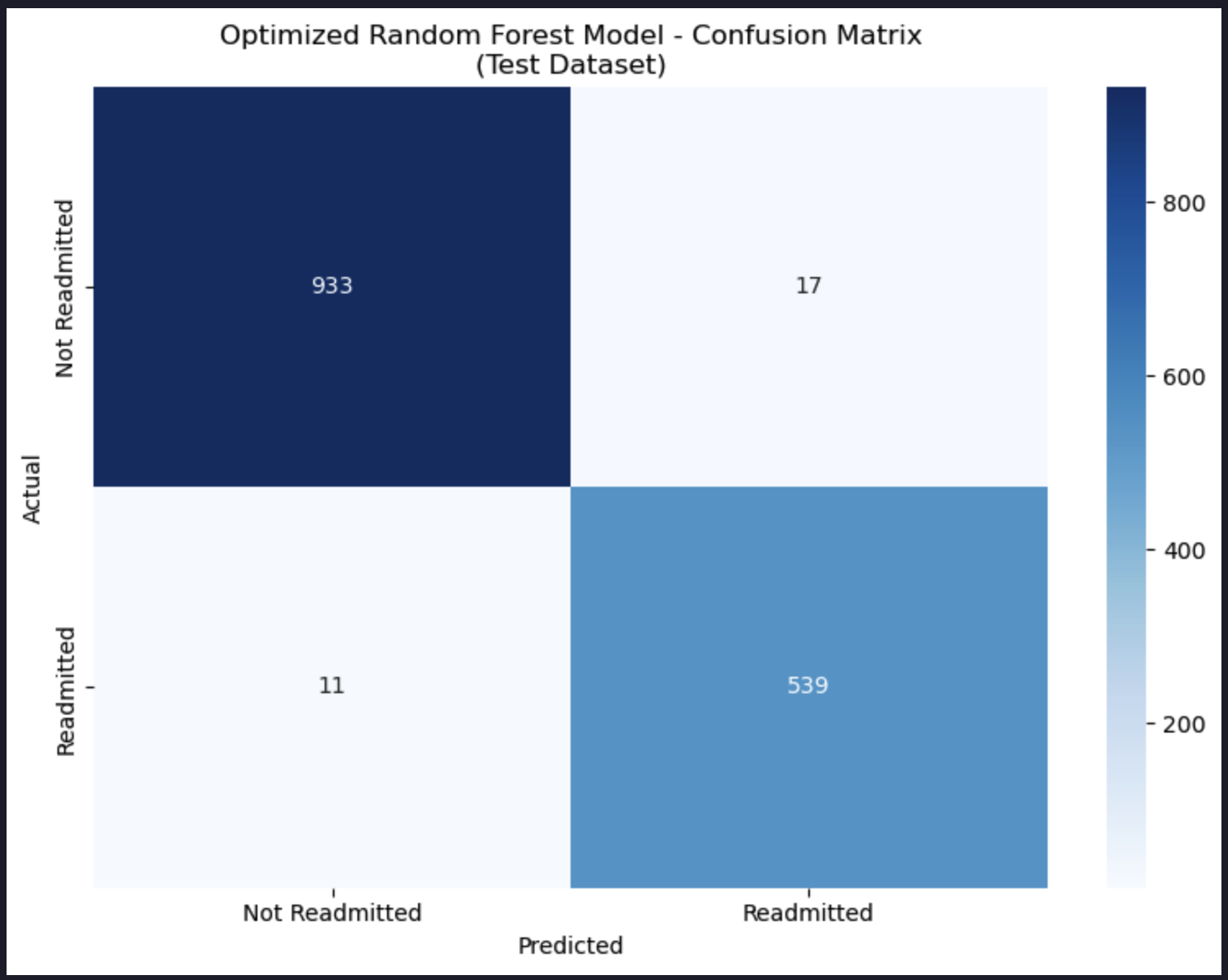
| *# Calculate and print all required metrics on TEST data* test\_accuracy = accuracy\_score(y\_test, y\_test\_pred) test\_precision = precision\_score(y\_test, y\_test\_pred) test\_recall = recall\_score(y\_test, y\_test\_pred) test\_f1 = f1\_score(y\_test, y\_test\_pred) test\_auc\_roc = roc\_auc\_score(y\_test, y\_test\_proba)  *# Calculate and print all required metrics* *# 1. Accuracy* test\_accuracy = accuracy\_score(y\_test, y\_test\_pred) print(f"Accuracy: {test\_accuracy:.4f} ({test\_accuracy:.1%})")  *# 2. Precision* test\_precision = precision\_score(y\_test, y\_test\_pred) print(f"Precision: {test\_precision:.4f} ({test\_precision:.1%})")  *# 3. Recall* test\_recall = recall\_score(y\_test, y\_test\_pred) print(f"Recall: {test\_recall:.4f} ({test\_recall:.1%})")  *# 4. F1 Score* test\_f1 = f1\_score(y\_test, y\_test\_pred) print(f"F1 Score: {test\_f1:.4f} ({test\_f1:.1%})")  *# 5. AUC-ROC* test\_auc\_roc = roc\_auc\_score(y\_test, y\_test\_proba) print(f"AUC-ROC: {test\_auc\_roc:.4f} ({test\_auc\_roc:.1%})")  *# 6. Confusion Matrix for TEST* test\_cm = confusion\_matrix(y\_test, y\_test\_pred) print(f"\nConfusion Matrix:") print(f" Predicted") print(f"Actual No (0) Yes (1)") print(f"No (0) {test\_cm[0,0]:6d} {test\_cm[0,1]:6d}") print(f"Yes (1) {test\_cm[1,0]:6d} {test\_cm[1,1]:6d}") |
| --- |



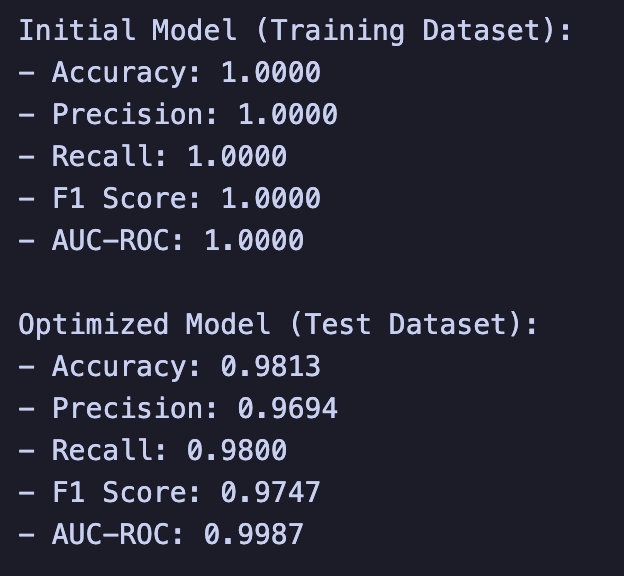
| *# Detailed breakdown* tn\_test, fp\_test, fn\_test, tp\_test = test\_cm.ravel() print(f"\nDetailed Breakdown:") print(f"True Negatives (TN): {tn\_test:,}") print(f"False Positives (FP): {fp\_test:,}") print(f"False Negatives (FN): {fn\_test:,}") print(f"True Positives (TP): {tp\_test:,}") |
| --- |

| print(f"\nClassification Report:") print(classification\_report(y\_test, y\_test\_pred,  target\_names=['Not Readmitted', 'Readmitted'])) |
| --- |

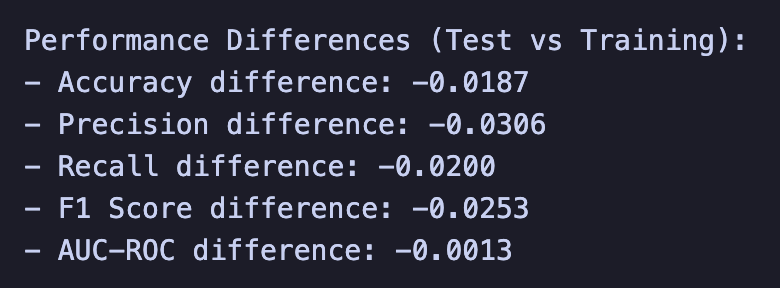
| *# Visualize confusion matrix for TEST data* plt.figure(figsize=(8, 6)) sns.heatmap(test\_cm, annot=True, fmt='d', cmap='Blues',  xticklabels=['Not Readmitted', 'Readmitted'],  yticklabels=['Not Readmitted', 'Readmitted']) plt.title('Optimized Random Forest Model - Confusion Matrix\n(Test Dataset)') plt.xlabel('Predicted') plt.ylabel('Actual') plt.tight\_layout() plt.show() |
| --- |



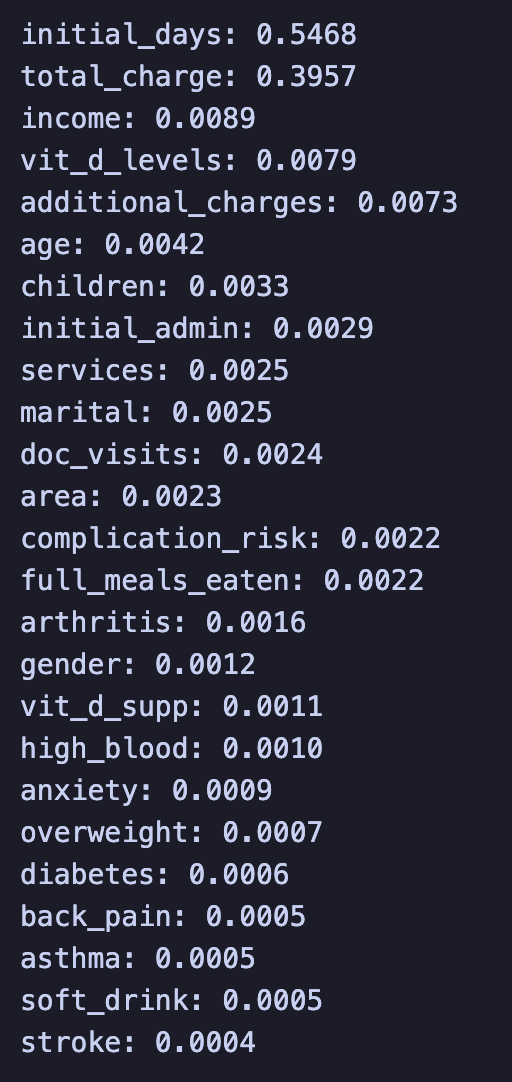
| *# Compare initial model (training data) vs optimized model (test data)* print(f"Initial Model (Training Dataset):") print(f"- Accuracy: {accuracy:.4f}") print(f"- Precision: {precision:.4f}") print(f"- Recall: {recall:.4f}") print(f"- F1 Score: {f1:.4f}") print(f"- AUC-ROC: {auc\_roc:.4f}")  print(f"\nOptimized Model (Test Dataset):") print(f"- Accuracy: {test\_accuracy:.4f}") print(f"- Precision: {test\_precision:.4f}") print(f"- Recall: {test\_recall:.4f}") print(f"- F1 Score: {test\_f1:.4f}") print(f"- AUC-ROC: {test\_auc\_roc:.4f}") |
| --- |



| *# Performance differences* print(f"\nPerformance Differences (Test vs Training):") print(f"- Accuracy difference: {test\_accuracy - accuracy:.4f}") print(f"- Precision difference: {test\_precision - precision:.4f}") print(f"- Recall difference: {test\_recall - recall:.4f}") print(f"- F1 Score difference: {test\_f1 - f1:.4f}") print(f"- AUC-ROC difference: {test\_auc\_roc - auc\_roc:.4f}") |
| --- |



| *# Selecting most important features* importances = rf\_optimized.feature\_importances\_  *# Pair features with importance values* feature\_importances = list(zip(X.columns, importances))  *# Sort features by importance* feature\_importances.sort(key=lambda x: x[1], reverse=True) for feature, importance in feature\_importances:  print(f"{feature}: {importance:.4f}") |
| --- |



## F. Analysis Summary

### F1. Model Evaluation

The comparison between the initial model performance on training data and the optimized performance model on test data revealed several key insights about the Random Forest Classifier.

The initial model, trained on the training dataset in Part E2, achieved perfect performance across all metrics, receiving a 100% on accuracy, precision, recall, F1 Score, and AUC-ROC. This indicates the model successfully learned patterns in the training dataset, but overfitting occurred. Perfect scores are both expected and normal for Random Forest Classification models when used on training data, again as a result of overfitting.

When splitting the data in Part E1, I took careful measure to use the stratify=y parameter when splitting the initial dataset into training, validation, and testing splits in order to keep the appropriate proportions of readmitted and non-readmitted patients for accurate testing of the model's predictive powers.

The optimized model was developed in Part E4 using the validation training dataset and then used on the testing dataset. All metrics performed at a 96% or above, showing that the hyperparameter tuning solved for overfitting while representing the model's true predictive capabilities on unseen data. The slight drop in performance again suggests that the initial model was suffering from overfitting.

In the hyperparameter tuning state (Part E3) used 5-fold cross-validation on the validation dataset to find optimal parameters, balancing model complexity and generalization of its predictive capabilities. Optimization of the F1 Score ensured balanced precision and recall for medical applications.

The small performance drop between the initial model and the optimized model is normal and expected when moving from training to testing datasets. The hyperparameter tuning using k-fold cross-validation successfully optimized the model for true predictive performance rather than the over-estimations that the overfitting was causing. The AUC-ROC scores indicate good discrimination ability between readmitted and non-readmitted patients and the optimized model generalizes well to unseen data despite the small drop in metric scores. This results in a better identification of true positives and fewer false positives.

### F2. Results and Implications

The optimized Random Forest Classification model successfully predicts patient readmission risk with approximately 96.9% accuracy, providing actionable insights for reducing patient readmission and building or improving upon hospital intervention programs.

**Clinical Implications:**

1. Early risk identification: The model enables early identification of high-risk patients before discharge, allowing medical staff to implement targeted interventions.
2. Resource allocation: Hospitals can prioritize limited resources toward patients most likely to be readmitted.
3. Personalized care plans: Feature importance rankings reveal which patient characteristics most strongly predict readmission, enabling personalized care strategies.

**Feature Importance Insights:**

The most significant features demonstrate that length of initial hospital stay (initial\_days) and total charges (total\_charges) are key predictors, likely due to longer stays and higher charges indicating more severe and/or more involved treatment needs.

**Business Impact:**

1. Cost reduction: Preventing readmissions reduces CMS penalties and associated costs.
2. Improved patient outcomes: Targeted interventions improve patient health and satisfaction.
3. Operational efficiency: Predictive insights enable better discharge planning and resource management.

**Healthcare Quality:**

The model supports evidence-based medicine by providing data-driven insights into readmission risk factors, ultimately improving the quality of patient care.

### F3. Limitation

Perhaps the biggest limitation of this particular analysis is that the model was trained and tested on data from a single hospital chain, limiting its generalizability to other healthcare systems with different patient populations, treatment approaches, or geographic locations.

Factors such as regional health disparities, varying socioeconomic conditions, patient access to hospitals and healthcare centers, and different hospital protocols could significantly impact the model's performance when applied to other healthcare organizations or hospital chains.

The model would need to be validated before deployment using different healthcare settings to ensure continued accuracy and relevance.

### F4. Course of Action

The original question asked is: Can we predict which patients are most likely to be readmitted within 30 days of discharge so that our hospital chain can implement targeted intervention programs to improve patient outcomes and reduce readmission penalties?

My initial recommended course of action is to focus on high-risk predictor variables like length of initial stay (measured in days) and total charge.

I would recommend deploying the model's use in 2-3 test hospitals and determining if the predictions can be generalized beyond the initial dataset provided.

I would also recommend training discharge planning teams and patient case managers on interpreting the model predictions and feature importance scores; this can provide flags on which patients are most at risk of readmission.

Finally, after further testing, I would recommend integrating the model with the electronic health records (EHR) system to automatically generate risk scores during patient stays.

With these things in mind, further courses of action that can be rolled out over time include developing standardized protocols for high-risk patients, establishing dedicated care coordination teams, hospital chain-wide rollout, performance monitoring to check for intervention effectiveness, and financial impact tracking.

Over the long term, model maintenance will need to be addressed to account for performance degradation and retraining, a consideration for broader model expansion to predict other patient outcomes, and improved financial impact tracking to determine return on investment through reduced penalties and improved patient satisfaction.

## G. Panopto Recording

​​<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=718e590c-a72a-4279-84d8-b33200208e80>

## H. Sources for Third-Party Code

Codecademy (n.d.) *Encoding Categorical Variables*.<https://www.codecademy.com/journeys/data-scientist-ml/paths/dsmlcj-22-machine-learning-foundations/tracks/dsmlcj-22-machine-learning-fun/modules/dsml-data-transformations-for-feature-analysis-2f0d46b3-b281-4d68-a8ac-30a198aae004/articles/fe-encoding-categorical-variables>

Codecademy (n.d.). *Training, Validation and Test Dataset*.<https://www.codecademy.com/journeys/data-scientist-ml/paths/dsmlcj-22-machine-learning-i/tracks/dsmlcj-22-supervised-learning-i/modules/mle-evaluation-metrics-98226eca-cdc3-42e7-98ee-9336cdb65a8c-a92db6b9-8ef8-4381-95d3-1e806194d196/articles/mlfun-training-test-validation>

## I. Sources

WGU Course Materials