**Readmission prediction for heart failure patients(AI/ML)**

**1. Project Overview**

This project aims to develop a predictive model for heart failure readmission using hospital patient data. The process involves data extraction, preprocessing, feature engineering, and model building.

**Team Members:**

1. Kavya Chaudhary
2. Khushi Tomer
3. Khushi Srivastav
4. Paarth Sarthi

**2. Data Sources**

The analysis uses the following datasets:

* admissions\_202208161605.csv: Admission details.
* patients\_202208161605.csv: Patient demographics.
* diagnoses\_icd\_202208161605.csv: Diagnosis codes (ICD-9).
* procedures\_icd\_202208161605.csv: Procedure codes (ICD-9).
* labevents\_202208161605.csv: Lab test results.
* cptevents\_202208161605.csv: CPT (Current Procedural Terminology) codes.
* drgcodes\_202208161605.csv: Diagnosis Related Group (DRG) codes.

**3. Data Preprocessing ("DataPREP.ipynb")**

**3.1. Libraries**

The following Python libraries are used:

* pandas: Data manipulation.
* numpy: Numerical operations.
* matplotlib.pyplot: Plotting.
* seaborn: Enhanced plotting.
* pickle: Saving objects.
* warnings: Handling warnings.
* sklearn.model\_selection: Data splitting.
* sklearn.preprocessing: Scaling, encoding.
* sklearn.ensemble: Random Forest.
* sklearn.metrics: Evaluation metrics.

**3.2. Data Loading**

The datasets are loaded into pandas DataFrames.

**3.3. Initial Data Inspection**

* drgcodes.isnull().sum(): Missing value counts in DRG codes.
* admissions.shape, patients.shape: Dimensions of DataFrames.
* Counting total rows, unique subject IDs, and duplicate subject IDs in the admissions data.

**3.4. Heart Failure Case Selection**

* A list of relevant ICD-9 codes (hf\_icd9) is defined to identify heart failure cases.
* The diagnoses DataFrame is filtered to include only heart failure-related diagnoses (hf\_diagnoses).
* The admissions DataFrame is filtered to include admissions corresponding to these heart failure diagnoses (hf\_admissions).

**3.5. Data Merging and Feature Engineering**

1. **Merging Demographic Data:** hf\_admissions is merged with patient demographics (patients) on subject\_id.
2. **Merging Diagnosis Data:** The result is merged with hf\_diagnoses on subject\_id and hadm\_id.
3. **Merging Lab Data:**
   * hf\_labevents is filtered to include only relevant admissions.
   * The average lab value (avg\_lab\_value) is calculated by grouping by hadm\_id.
   * hf\_cohort is merged with lab\_avg.
4. **Merging Procedure Count:**
   * hf\_cptevents is filtered.
   * The count of unique CPT codes (cpt\_code\_count) is calculated per admission.
   * hf\_cohort is merged with cpt\_code\_count.
5. **Merging DRG Severity:**
   * hf\_drgcodes is filtered.
   * drg\_severity is extracted (handling duplicates).
   * hf\_cohort is merged with drg\_severity.
6. **Ethnicity Grouping:** A binary ethnicity\_group feature is created ('Caucasian' vs. 'Non-Caucasian').
7. **Data Inspection:** Shape, unique IDs, and duplicates are checked, similar to the initial inspection.

**3.6. Missing Value Analysis**

The code calculates and prints the number of missing values for each column in the merged DataFrame (hf\_cohort).

**3.7. Final Data Preparation**

* **Date and Time Conversion:** admittime, dischtime, and dob are converted to datetime objects.
* **Admission Hour and Day:** admit\_hour and admit\_day are extracted from admittime.
* **Length of Stay:** length\_of\_stay is calculated as the difference between dischtime and admittime.

**4. Data Transformation and Feature Selection ("MODELLING.ipynb")**

**4.1. Feature Selection**

A list of features (features) is defined for the model: age, gender, ethnicity\_group, insurance\_type, admit\_hour, admit\_day, admission\_type, avg\_lab\_value, cpt\_code\_count, drg\_severity, and icd9\_category.

**4.2. Handling Missing Values**

* Categorical columns (cat\_cols) are imputed using the most frequent value.
* Numerical columns (num\_cols) are imputed using the mean.

**4.3. Encoding and Scaling**

* Categorical features are one-hot encoded.
* Numerical features are standardized (scaled).

**4.4. Final Feature Set**

The processed numerical and categorical features are combined into a final DataFrame (X\_final).

**5. Model Building and Evaluation ("MODELLING.ipynb")**

**5.1. Model Training**

* The data is split into training and testing sets.
* A Random Forest Classifier (rf) is trained.
* Model performance is evaluated using metrics like the confusion matrix and classification report.

**5.2. Model Saving**

The trained Random Forest model is saved to a pickle file (random\_forest\_model.pkl).

**6. Test Cases**

**I. Data Preprocessing ("DataPREP.ipynb") Test Cases**

The focus here is on validating the data transformations and aggregations.

**1. Heart Failure Case Selection**

* **Test Case 1.1:**
  + Input: diagnoses DataFrame with various ICD-9 codes, including and excluding heart failure codes.
  + Expected Output: hf\_diagnoses DataFrame contains *only* rows with the specified hf\_icd9 codes.
  + Validation:
    - Check if all icd9\_code values in hf\_diagnoses are present in hf\_icd9.
    - Check if the number of rows in hf\_diagnoses is correct based on the filter.
* **Test Case 1.2:**
  + Input: admissions DataFrame and hf\_diagnoses DataFrame.
  + Expected Output: hf\_admissions contains only admissions whose hadm\_id is present in hf\_diagnoses.
  + Validation: Verify that all hadm\_id values in hf\_admissions exist in hf\_diagnoses.
* **Test Case 1.3:**
  + Input: diagnoses DataFrame with null values in the icd9\_code column.
  + Expected Output: The code should handle null values without errors, and hf\_diagnoses should not contain any null values in the icd9\_code column.
  + Validation: Verify that there are no null values in the icd9\_code column of the hf\_diagnoses DataFrame.

**2. Data Merging and Feature Engineering**

* **Test Case 2.1:** (Merging demographics)
  + Input: hf\_admissions, patients DataFrames.
  + Expected Output: hf\_cohort contains all columns from hf\_admissions and dob, gender from patients. The number of rows should remain the same as hf\_admissions.
  + Validation:
    - Check for the presence of expected columns.
    - Verify the number of rows.
    - Check if subject\_id values match correctly after the merge.
* **Test Case 2.2:** (Merging lab values)
  + Input: hf\_cohort, labevents
  + Expected Output: hf\_cohort has a new avg\_lab\_value column.
  + Validation:
    - Check if avg\_lab\_value is calculated correctly by comparing it to manual calculations on a subset.
    - Handle cases where a patient has no lab events (expect nulls or a default value).
* **Test Case 2.3:** (Counting procedures)
  + Input: hf\_cohort, cptevents
  + Expected Output: hf\_cohort has a cpt\_code\_count column.
  + Validation:
    - Verify the counts against manual counts.
    - Test cases with zero procedures.
* **Test Case 2.4:** (DRG severity)
  + Input: hf\_cohort, drgcodes
  + Expected Output: hf\_cohort has drg\_severity.
  + Validation:
    - Check if the correct severity is extracted.
    - Handle duplicate DRG codes appropriately (as the code drops duplicates).
* **Test Case 2.5:** (Ethnicity grouping)
  + Input: hf\_cohort with various ethnicity values.
  + Expected Output: hf\_cohort has ethnicity\_group with only 'Caucasian' and 'Non-Caucasian'.
    - Validation: Check that no other values exist in ethnicity\_group.

**3. Missing Value Analysis**

* **Test Case 3.1:**
  + Input: hf\_cohort with artificially injected missing values in various columns.
  + Expected Output: The code correctly counts and reports the number of missing values per column.
  + Validation: Compare the reported counts with the actual number of injected missing values.

**4. Final Data Preparation**

* **Test Case 4.1:** (Date/time conversion)
  + Input: hf\_cohort with admittime, dischtime, dob as strings.
  + Expected Output: These columns are converted to datetime objects.
  + Validation:
    - Check the data types of the columns after conversion.
    - Verify that the conversion is accurate.
* **Test Case 4.2:** (Length of stay)
  + Input: hf\_cohort with admittime, dischtime.
  + Expected Output: hf\_cohort has length\_of\_stay calculated correctly.
  + Validation: Compare calculated length\_of\_stay with manual calculations.
* **Test Case 4.3:** (Admission hour and day)
  + Input: hf\_cohort with admittime.
  + Expected Output: admit\_hour and admit\_day columns are extracted correctly.
  + Validation: Verify the extracted hour and day values.

**II. Modeling ("MODELLING.ipynb") Test Cases**

Here, we test the feature processing and model training.

**1. Feature Selection**

* **Test Case 1.1:**
  + Input: DataFrame df
  + Expected Output: DataFrame X contains only the columns specified in the features list.
  + Validation: Check if X has the correct columns.

**2. Handling Missing Values**

* **Test Case 2.1:**
  + Input: DataFrame X with missing values in categorical and numerical columns.
  + Expected Output: Missing values are imputed correctly. Categorical columns with the most frequent value and numerical columns with the mean.
  + Validation:
    - Verify that there are no remaining missing values in X.
    - Check if the imputed values are correct.
* **Test Case 2.2:**
  + Input: DataFrame X where one or more categorical columns have only one unique value.
  + Expected Output: The imputer should still work without errors.
  + Validation: The code doesn't raise exceptions.

**3. Encoding and Scaling**

* **Test Case 3.1:** (One-hot encoding)
  + Input: DataFrame X with categorical columns.
  + Expected Output: Categorical columns are one-hot encoded; X\_encoded has new columns.
  + Validation:
    - Check the number of columns in X\_encoded.
    - Verify that the encoding is correct (binary values).
    - Test cases with categories not seen during training ('unknown' handling).
* **Test Case 3.2:** (Scaling)
  + Input: DataFrame X with numerical columns.
  + Expected Output: Numerical columns are standardized; X\_scaled has scaled values.
  + Validation: Check the mean and standard deviation of scaled columns (should be close to 0 and 1).

**4. Final Feature Set**

* **Test Case 4.1:**
  + Input: X\_scaled, X\_encoded
  + Expected Output: X\_final is created by correctly concatenating scaled and encoded features.
  + Validation:
    - Check the shape and column names of X\_final.
    - Verify that data types are appropriate.

**5. Model Training**

* **Test Case 5.1:**
  + Input: X\_final, target variable y
  + Expected Output: The Random Forest model trains without errors.
  + Validation: The code executes successfully.
* **Test Case 5.2:**
  + Input: A trained model and test data.
  + Expected Output: The model produces predictions.
  + Validation:
    - Check the format of predictions.
    - Evaluate predictions using metrics (confusion matrix, etc.).
* **Test Case 5.3:**
  + Input: X\_final with highly imbalanced classes in the target variable.
  + Expected Output: The model trains, but performance metrics (especially recall for the minority class) are captured.
  + Validation: Examine the classification report to confirm model behavior under class imbalance.

**6. Model Saving**

* **Test Case 6.1:**
  + Input: Trained Random Forest model.
  + Expected Output: Model is saved to a pickle file.
  + Validation:
    - Check if the file is created.
    - Load the model from the file and make sure it's the same as the original.

**7. Important Considerations for Test Cases**

* **Data Variety:** Use diverse data subsets to cover different scenarios (e.g., edge cases, outliers, specific value combinations).
* **Boundary Conditions:** Test with minimum and maximum values, empty datasets, etc.
* **Error Handling:** Consider how the code should behave when encountering unexpected input (e.g., wrong data types, missing files).
* **Reproducibility:** Ensure that test cases are designed to produce consistent results.
* **Automation:** Ideally, test cases should be automated using a testing framework (like pytest in Python) to ensure continuous validation as the code evolves.

**Result:**

