Initial Hospital Stay Prediction (Random Forest Regression)

Ashley Traore

Research Question:

Can the Random Forest method be used to predict the initial days a patient spends in the hospital with the currently available data?

Goals for Analysis:

The primary goal of this analysis is to develop a Random Forest machine-learning model to help predict the initial number of days a patient is likely to spend in the hospital. The company can then make more informed business decisions aimed at minimizing prolonged hospital stays.

Random Forest Analysis and Expected Outcomes:

The Random Forest technique processes data by constructing multiple decision trees and merging their outputs. It begins by generating multiple subsets of the training data through a method called bootstrap sampling, where random samples are drawn. Each subset is used to train a distinct decision tree. During the training, the algorithm considers only a randomly selected subset of features instead of all the features available. This promotes diversity among the trees and reduces overfitting. The trees, trained independently using their respective bootstrap samples, split the data based on thresholds that minimize prediction errors. For regression tasks, the final output is the average of all tree predictions, while for classification tasks, the most frequently predicted class across the trees determines the outcome.

One expected outcome is highly accurate predictions. The incorporation of randomness in data sampling and feature selection minimized the risk of overfitting, leading to predictions that are both more accurate and dependable.

Assumption:

An important assumption of Random Forest modeling is the availability of adequate data. This means the dataset must contain a sufficient number of observations to support diverse tree training. Additionally, the data should be of high quality, featuring meaningful predictors and minimal noise.

Libraries Used:

Python was chosen as the programming language for this project for two key reasons. Firstly, Python has simple and consistent syntax, which facilitates easier coding and debugging. Secondly, Python has a large and active community, providing an abundance of resources and tutorials for support and knowledge sharing.

Several modeling and data analysis libraries were utilized for this project:

Packages and Libraries	Usage
pandas	Used to handle and prepare the data.
numpy	For performing numerical operations on arrays.
sklearn.preprocessing import LabelEncoder	For encoding the data.
sklearn.model_selection import train_test_split	For splitting the data into train/test sets.
sklearn.neighbors import RandomForestRegressor	To perform Random Forest Regression.
sklearn.metrics import mean_squard_error	To calculate the mean squared error of the model.
sklearn.feature_selection import SelectKBest, f_regression	To select the most significant features.

Data Preprocessing Goal:

When utilizing Random Forest to predict the initial days a patient spends in the hospital, one crucial preprocessing goal is to encode categorical variables into numerical form. Encoding categorical variables is crucial because machine learning algorithms like Random Forest require numerical input to function. Categorical data such as “Married”, “Single”, or “Divorced”, must be converted into numeric representations to be processed effectively. For yes/no variables, a loop and the replace() function were employed to transform them into 1/0. For categorical variables with more than two categories, the LabelEncoder() function was utilized to assign numeric values to each category.

Steps to Prepare the Data:

1. Import the required libraries.

```
# In[2]:

# Import Libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

from sklearn.feature_selection import SelectKBest, f_regression

from sklearn.ensemble import RandomForestRegressor

# Change setting in pandas to display all columns

pd.options.display.max_columns = None
```

2. Load the data into a pandas dataframe and view the data.

```
# ### Data Import

# In[4]:

# Import medical dataset

medicalDF =pd.read_csv(r"C:\Users\ashle\Desktop\MSDA WGU\Data Mining 1 - D209\Medical Data\medical_clean.csv")

# In[5]:

medicalDF
```

3. Print the information for medicalDF to evaluate the data types for each column.

```
# In[6]:
```

```
medicalDF.info()
```

4. Create a copy of the data frame for data cleaning, then, identify any duplicates. No duplicates were identified.

```
# ### Data Cleaning

# In[8]:

# Create a copy of the data frame for cleaning the data.

medicalClean = medicalDF.copy()

# In[9]:

# Indentify duplicates

duplicates = medicalClean[medicalClean.duplicated()]

print(duplicates)
```

5. Identify any null values. No missing data was identified.

```
# In[10]:

# Identify null values in the dataframe.

missingValues = medicalClean.isnull().sum().sum()

print(missingValues)
```

6. Identify and cap outliers using Z-score capping with a threshold set at 3 standard deviations from the mean.

```
# In[11]:

#Identifying Outliers

#Define column variable with all the numeric columns in the data frame

columns = ['Lat', 'Lng', 'Children', 'Age', 'VitD_levels', 'Doc_visits',
'Full_meals_eaten', 'vitD_supp', 'Population', 'Income']

print("\n\n\033[1m Count of Outliers: ")
```

```
#Loop through each numeric column to find outliers

for i in columns:

    #print('')

    # Calculating the first quartile, the third quartile, and the interquartile range. [In-Text Citation: (Hackers Realm)]

    Q1 = medicalClean[i].quantile(0.25)

    Q3 = medicalClean[i].quantile(0.75)

    IQR = Q3 - Q1

        # Define outlier bounds

    upper_bound = Q3 + 1.5 * IQR

    lower_bound = Q1 - 1.5 * IQR

        # Identify outliers

    outliers = medicalClean[(medicalClean[i] < lower_bound) | (medicalClean[i] > upper_bound)]

    #Printing the count and range of outliers

    print("\n\033[1m " + i + ": \033[0m" + "Count of outliers: " + str(len(outliers)))

# Cleaning Outliers

# Z-score capping to cap all columns with outliers with a threshold set at 3 standard deviations from the mean.

outlierColumns = ['Lat', 'Lng', 'Children', 'VitD_levels', 'Full_meals_eaten', 'vitD_supp', 'Population', 'Income']

print("\n\n\033[1m Upper and Lower Threshold for Each Column: ")

# Loop through each column that has outliers.

for i in outlierColumns:

    # Calculate the upper and lower threshold using the Interquartile Range (IQR) method.

    upperThreshold = medicalClean[i].mean() + 3*medicalClean[i].std()

    lowerThreshold = medicalClean[i].mean() - 3*medicalClean[i].std()
```

```
    #Print upper and lower threshold for each column
    print("\n\033[1m" + i + ": \033[0m Upper: " + str(upperThreshold) + ' Lower: ' +
str(lowerThreshold))
        #Capping outliers for each column on upper and lower limit
    medicalClean.loc[(medicalClean[i]>=upperThreshold), i] = int(upperThreshold)
    medicalClean.loc[(medicalClean[i]<=lowerThreshold), i] = int(lowerThreshold)
# Print a count of outliers after cleaning.
print("\n\n\033[1m Count of Outliers After Cleaning: ")
for i in columns:
    outliers_ = medicalClean[(medicalClean[i] < lowerThreshold) | (medicalClean[i]
> upperThreshold)]
    print("\n\033[1m " + i + ": \033[0m" + "Count of Outliers: " + str(len(outliers_)))
```

7. Transforming the yes/no categorical variables into binary format represented as 1/0, and applying LabelEncoder to encode all categorical columns with multiple categories. Additionally, removing all ID columns that are irrelevant to the analysis.

```
# In[12]:
# Re-express all yes/no categorical variables to be expressed as 1/0.
# Create a variable for categorical columns that need to be replaced.
yes_no_Columns = ['ReAdmis', 'Soft_drink', 'HighBlood', 'Stroke', 'Arthritis',
'Diabetes', 'Hyperlipidemia', 'BackPain',
                  'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Overweight',
'Anxiety']
# Loop through each column and use replace to change yes/no values to 1/0.
for i in yes_no_Columns:
    medicalClean[i].replace(to_replace =['Yes','No'], value = [1,0], inplace =
True)
columns = ['Initial_admin', 'Complication_risk', 'Gender', 'Services', 'City', 'State',
'County', 'Area', 'TimeZone', 'Job', 'Marital', ]
```

```
for col in columns:
    label_encoder = LabelEncoder()
    encodedLabels = label_encoder.fit_transform(medicalClean[col])
    medicalClean[col] = encodedLabels
medicalClean.drop(['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'Item1',
'Item2', 'Item3',        'Item4', 'Item5', 'Item6', 'Item7', 'Item8'],axis=1,inplace=True)
# In[13]:
medicalClean
```

8. Applying SelectKBest to identify the features with the highest p-values associated with readmission. These selected features will then serve as input variables for the KNN model.

```
# ## Selecting  Significant Variables
# In[15]:
# Using SelectKBest to identify the features with the highest p values
X = medicalClean.drop(["Initial_days"],axis=1)
y = medicalClean['Initial_days']
feature_names = X.columns
skbest = SelectKBest(score_func=f_regression, k=5)
X_new = skbest.fit_transform(X, y)
print(X_new.shape)
# Finding p-values greater than .05 to select statiscially significant columns
p_values = pd.DataFrame({'Feature':X.columns,
'p_value':skbest.pvalues_}).sort_values('p_value')
p_values[p_values['p_value']<.05]
features_to_keep = p_values['Feature'][p_values['p_value']<.05]
#print(features_to_keep)
```

```
predictors = medicalClean[features_to_keep]

# Print features to keep and the associated p values

features_to_keep_df = p_values[p_values['p_value'] < 0.05]

print(features_to_keep_df)

# In[16]:

predictors
```

9. Exporting the cleaned data.

```
# In[17]:

# Export cleaned data

predictors.to_csv(r'C:\Users\ashle\Desktop\MSDA WGU\Data Mining 1 - D209\PA Task 2 - Python\cleaned datasets\CleanedData.csv')
```

10. Splitting the data into training and testing sets and exporting the sets to csv files.

```
# ## Splitting the data

# In[19]:

# Splitting the data into train/test sets

X = predictors

y = medicalClean['Initial_days']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=24)

# In[20]:

#Export train and test sets

X_train.to_csv(r'C:\Users\ashle\Desktop\MSDA WGU\Data Mining 1 - D209\PA Task 2 - Python\cleaned datasets\X_train.csv')

X_test.to_csv(r'C:\Users\ashle\Desktop\MSDA WGU\Data Mining 1 - D209\PA Task 2 - Python\cleaned datasets\X_test.csv')
```

```
y_train.to_csv(r'C:\Users\ashle\Desktop\MSDA WGU\Data Mining 1 - D209\PA Task 2 - Python\cleaned datasets\y_train.csv')

y_test.to_csv(r'C:\Users\ashle\Desktop\MSDA WGU\Data Mining 1 - D209\PA Task 2 - Python\cleaned datasets\y_test.csv')
```

Analysis Technique and Calculations:

The analysis technique used to predict the initial number of days a patient spends in the hospital is Random Forest Regression. First, data preprocessing was performed to handle missing values, duplicate values, address outliers, and encode categorical variables into numerical format.

Following preprocessing, feature selection was performed using SelectKBest, which identifies the most significant features based on their p-values in relation to initial days. This helps to identify the five most relevant predictor variables, ensuring only significant features were maintained. The data was then split into training and testing sets using an 80/20 split to ensure a reliable evaluation of the model performance while avoiding overfitting.

Next, a RandomForestRegressor was initialized and trained on the training dataset, allowing the model to identify patterns and relationships within the features. Upon completion of the training phase, predictions were made on the test set.

Finally, model evaluation was carried out using the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared metrics. The model achieved an MSE score of 13.55, an RMSE score of 3.68, and a R-squared value of 0.98.

```
Mean Squared Error: 13.552251364171294
Root Mean Squared Error: 3.6813382572335422
R-squared: 0.9807
```

Mean Squared Error (MSE):

Mean Squared Error (MSE) is a metric commonly used to assess the performance of a regression model. It calculates the average squared difference between the predicted and actual values (GeeksforGeeks, 2025). The closer the MSE value is to 0, the more accurate the model's predictions are. Due to MSE being in squared units, it can be less intuitive to interpret without taking the square root. To make the error easier to interpret Root Mean Squared Error (RMSE) can be calculated, which provides the error in the same units as the target variable.

The model achieved an MSE value of 13.55, an RMSE value of 3.68, and an R-squared value of 0.98. An MSE value of 13.55 means that the average squared

difference between the model's predictions and actual values is 13.55. This suggest that on average there is some error in the model predictions. An RMSE value of 3.68, means the model's predictions, on average, deviate from the actual values by approximately 3 days. The range of the target variable, initial days, is quite large, ranging from 1 to 71 days, an average error rate of 3 days indicates that the model performs well in capturing the patterns and variability in the data. An R-squared value of 0.98 means that 98% of the variability in initial days is explained by the model's features. This is an exceptionally high R-squared value demonstrating that the model captures the underlying patterns in the data with great accuracy. These metrics demonstrate that the Random Forest model performs well, providing accurate predictions and actionable insights.

Results and Implications:

The Random Forest Regression model showcases impressive predictive performance with notable implications. With an MSE score of 13.55 and an RMSE score of 3.68, the model's predictions deviate from actual values by an average of 3.68 days. Given the target variable (initial days) spans from 1 to 71 days, this represents a relatively small margin of error, highlighting the model's accuracy in capturing data patterns and variability. Additionally, the R-squared value of 0.98 demonstrates the model accounts for 98% of the variability in the target variable, leaving only 2% unexplained. This underscores the model's effectiveness in identifying relationships between predictions and the target.

Through feature selection, the five most influential predictors of initial days were identified, offering actionable insights into key factors affecting the target variable. The model's predictions can aid the hospital's administrators in resource allocation, such as anticipating extended patient stays and optimizing staff scheduling. Additionally, by pinpointing factors strongly linked to hospital stays, medical professionals can design targeted interventions or treatments to reduce prolonged hospitalizations. These insights can also inform policy updates or initiatives to enhance patient outcomes in hospital efficiency.

Limitation of Method Used:

One notable limitation of a Random Forest Regression model is its inability to extrapolate. For example, in this case where the target variable in the data set ranges from 1 to 71 days, the model cannot reliably predict values beyond this range. When

faced with such situations, it assigns predictions based on the nearest observed region in the training data, which can lead to inaccurate or even nonsensical results. This occurs because Random Forest models do not use mathematical equations or explicit functional relations for predictions. Instead, they rely on patterns and data points encountered during training, which restricts their ability to generalize relationships outside the scope of the observed data (Lyashenko, 2021).

Recommendations:

Based on the results of the Random Forest Regression model, it is recommended that the hospital use the predictions to better plan staffing and resource distribution. For example, knowing the likely length of the patient's stay can help ensure availability of staff, equipment, or hospital beds. These predictions can also help in planning discharges or admissions, minimizing bottlenecks. Additionally, the hospital can develop specialized patient care plans based on their predicted length of stay, for example patients with longer predicted stays may benefit from more personalized care or counseling. Lastly, the model can be further enhanced by fine-tuning the algorithm or incorporating additional features to enhance performance.

Code Sources:

Selectkbest. scikit. (n.d.-e). https://scikit-learn.org/dev/modules/generated/sklearn.feature_selection.SelectKBest.html

Randomforestregressor. scikit. (n.d.-e). https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Pandas. pandas. (n.d.). https://pandas.pydata.org/

Sources:

GeeksforGeeks. (2025c, January 16). *Random Forest algorithm in machine learning*. https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

GeeksforGeeks. (2025d, March 3). *Mean squared error: Definition, formula, interpretation and examples%%page%% %%Sep%% %%sitename%%*. https://www.geeksforgeeks.org/mean-squared-error/

Lyashenko, V. (2021, February 3). *Random Forest regression - the definitive guide: Intel® tiberTM ai studio*. cnvrg. https://cnvrg.io/random-forest-regression/#:~:text=The%20Random%20Forest%20Regressor%20is%20unable%20to%20discover,Forest%20is%20less%20interpretable%20than%20a%20Decision%20tree.