University of Leicester

A
Report on,
CO7093 - Big Data & Predictive Analytics
CW Assignment: Classification & Clustering

Submitted by,
Group ID: 26

Sr.	Name of Student	Student ID no.
1.	Shubham S. Sonawane	239062846
2.	Om Vilas Shimpi	239063265
3.	Vivek Prakash Shah	239062179
4.	Ritika Ritika Karthikeyan	239057165
5.	Dhvanil Alpeshkumar Patel	239064467

Under the guidance of,

Prof. Furqan Aziz

Table of Content

GitHub Repository: https://github.com/mark-3000/C07093_CW_Classification-Clustering/tree/main

CHAPTER 1
INTRODUCTION

The burgeoning field of big data analytics has profoundly impacted healthcare, enabling the synthesis of vast datasets to glean insights that drive better patient outcomes and operational efficiencies. This project harnesses the power of predictive analytics to address a critical challenge in healthcare management: predicting 30-day readmission rates for patients with diabetes. Readmissions not only exacerbate healthcare costs but also indicate potential gaps in care and treatment effectiveness. The dataset, spanning a decade of clinical data from 130 U.S. hospitals, provides a rich foundation for developing a predictive model that can forecast the likelihood of patient readmission within 30 days' post-discharge.

The complexity of diabetes as a chronic condition, coupled with its widespread prevalence and significant impact on patient quality of life, necessitates a nuanced approach to managing patient care transitions from hospital to home. Early identification of patients at high risk of readmission can trigger targeted interventions, improving patient outcomes and reducing unnecessary healthcare expenditures. Therefore, the primary aim of this study is to leverage advanced data analytics techniques to classify patients based on their readmission risk.

This report begins by delineating the project's scope, including the dataset's characteristics and the specific objectives of the analytics task. It then details the methodological approach, starting with data preparation and cleansing to address issues of missing values and data inconsistency. Following this, the report discusses the development of a basic predictive model using machine learning algorithms, emphasizing the handling of imbalanced data and the evaluation of model performance through various metrics. Subsequently, the narrative shifts to the exploration of an enhanced model that integrates unsupervised clustering techniques, aiming to refine the predictive accuracy further.

Through this introduction, we set the stage for a comprehensive examination of the methodologies employed in analyzing the data, the iterative process of model refinement, and the critical evaluation of the models' predictive capabilities. The ultimate goal is to contribute to the broader discourse on using big data analytics in healthcare, particularly in managing chronic diseases like diabetes, by providing actionable insights into patient readmission risks.

CHAPTER 2
REPORT ON PROPOSED SYSTEM AND ITS IMPLEMENTATION

2.1 Proposed System

The system aims to predict 30-day hospital readmission for diabetes patients using a dataset from 130 U.S. hospitals. The implementation process involves loading the data into a Pandas DataFrame, followed by data cleaning, transformation, and analysis. Various Python libraries such as Pandas, Matplotlib, and Seaborn are used for data manipulation and visualization. Machine learning models are developed to predict readmission rates, leveraging the cleaned and processed dataset.

Pre-Requisites:

- **Python version 3.7 or higher**
- **Pandas library version 1.5.3**
  The particular version of Pandas library (!pip install pandas==1.5.3) that is utilised in the our code was probably selected for a few reasons:
  - Compatibility: The code may rely on features or behaviour that was added in a certain version of pandas (in this version, 1.5.3). Some features may not be available if you are using an earlier version, and breaking changes may have been made in a later version.
  - Consistency with the Environment: The code may be a component of a bigger project with a specified Python environment that has version constraints. The pandas version used in this code snippet matches the environment's setup when pip install pandas==1.5.3 is used.
  - Known Issues: By utilising the particular version (1.5.3), the code may be able to circumvent known issues or defects in more recent versions of Pandas. By giving the version, you can be guaranteed that the code will run with a stable and dependable pandas library
- **We use below libraries in our code for a particular reason.**
  - pandas (pd) is used for data loading, cleaning, and exploration. It offers data structures like DataFrames (similar to spreadsheets) and Series (one-dimensional arrays) that efficiently store and handle various data types like numeric values, text strings, and categorical data.
  - NumPy (np) (imported but not explicitly used in our code) is a foundation for numerical computing in Python. It provides efficient arrays for numerical data and mathematical operations.
  - Seaborn (sns) (used for visualizations) builds on top of Matplotlib to create informative and aesthetically pleasing statistical graphics. It leverages pandas DataFrames for data and works seamlessly with pandas data types.

- Matplotlib (plt) (used for visualizations) is a fundamental plotting library in Python. It offers various functionalities to create charts, histograms, and other visual representations of data.
- imbalanced-learn (potentially for later use): Addresses classification problems with imbalanced datasets and was created expressly to deal with classification issues with imbalanced datasets, in which one class has noticeably less samples than the others.
- scikit-learn: Used for feature selection (SelectKBest, f_classif).

- **Also need basic understanding of data cleaning, visualization, and machine learning concepts.**

2.2 Implementation

Part 1: Building up a basic predictive model

1. Data Cleaning and Transformation

We start by adding the required libraries to the dataframe. The Python libraries that have been used throughout the code for their specific purposes are listed below.

```
In [1]: #Load the dataset patients.csv into pandas dataframe
        !pip install pandas==1.5.3
        # import pkg_resources
        # pkg_resources.require("pandas==1.5.3")
        import pandas as pd
        df = pd.read_csv('diabetic_data.csv')
        print(pd.__version__)

        Defaulting to user installation because normal site-packages is not writeable
        Requirement already satisfied: pandas==1.5.3 in c:\users\dhvanil\appdata\roaming\python\python311\site-packages (1.5.3)
        Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib\site-packages (from pandas==1.5.3) (2.8.
        2)
        Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas==1.5.3) (2023.3.post1)
        Requirement already satisfied: numpy>=1.21.0 in c:\programdata\anaconda3\lib\site-packages (from pandas==1.5.3) (1.24.3)
        Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas==1.
        5.3) (1.16.0)
        1.5.3
```

The main csv file, which is the dataframe's main dataset, was imported after the required libraries were added to the dataframe. Using the standard syntax for defining a dataset, we gave the dataset the name "df." Then we started working on the tasks that needed to be done in order to finish data cleaning and transformation. The following are the tasks' implementation and solutions:

1) Show the shape of the data.

```
In [2]: #Show the shape of the data
        df.shape

Out[2]: (101766, 50)
```

The Csv file imported to datafram using pandas library is having 101766 rows and 50 columns in it:
By using the shape in-built function to get the size of the dataframe

2) *Delete the column 'encounter_id'.*

```
In [3]: #Delete the column 'encounter_id' (why axis =1 (whenever  row (axis=0) or a column (axis=1).))
        df = df.drop('encounter_id', axis=1)
```

Explanation: This command drops the column 'encounter_id' from the DataFrame 'df'. The 'axis=1' parameter specifies that the operation should be performed along columns.

3) *Identify missing values in the columns. You will notice that there are some columns with the missing values, but those values are represented by a different character such as '?'. Replace them by NaN.*

```
In [4]: #Identify missing values in the columns. You will notice that there are some columns with the missing values, but those values
        # Identify missing values represented by '?'
        df.replace('?', pd.NA, inplace=True) #inplace needed for editing existing df (if false we have assign new variable for df)
```

Explanation: This command replaces all occurrences of '?' in the DataFrame 'df' with Pandas' missing value representation (pd.NA) in place. It ensures '?' are replaced with NaN values for consistent handling of missing data.

4) *Show a summary of all missing values before and after applying the above operation.*
Before:

```
In [5]: print("Summary of Missing Values Before Replacement:")
        print(df.isnull().sum())
```

After:

```
In [6]: # Display summary of missing values after replacement
        print("\nSummary of Missing Values After Replacement:")
        print(df.isnull().sum())
```

Explanation: This command prints the sum of missing values in each column of the DataFrame 'df'. It provides a quick overview of the number of missing values in each feature.

5) *The response variable, 'readmitted', originally had three levels: '<30', '>30', and 'NO'. Since we are predicting early readmission of the patient within 30 days, we need to convert this feature to a binary feature. Replace '<30' with a 1 and '>30' and 'NO' to 0 respectively.*

```
In [7]: #5th point
        df['readmitted'] = df['readmitted'].replace({'<30': 1, '>30': 0, 'NO': 0})

        # Display the updated 'readmitted' column
        print(df['readmitted'].value_counts())

        0    90409
        1    11357
        Name: readmitted, dtype: int64
```

Explanation: This command replaces the values in the 'readmitted' column of the DataFrame 'df'. '<30' is replaced with 1 indicating early readmission, while '>30' and 'NO' are replaced with 0 indicating no early readmission.

6) *Check the datatype of each column.*

```
In [8]: # Display the data types of each column
        print("Data Types of Each Column:")
        print(df.dtypes)
```

Explanation: This command prints the data types of each column in the DataFrame 'df'. It provides information on the data types of the features, aiding in further data manipulation and analysis.

7) *For each column calculate the percentage of missing values. Drop columns with more than 90% missing values.*

```
In [9]: # Calculate the percentage of missing values for each column
        missing_percentage = (df.isnull().sum() / len(df)) * 100
        print(missing_percentage)
```

Explanation: This command calculates the percentage of missing values for each column in the DataFrame 'df' and assigns the result to the variable 'missing_percentage'. It helps in understanding the extent of missing data in the dataset.

```
In [10]: # Drop columns with more than 90% missing values
         columns_to_drop = missing_percentage[missing_percentage > 90].index
         df = df.drop(columns=columns_to_drop)
```

Explanation: This command identifies columns with missing value percentages greater than 90% and assigns them to the variable 'columns_to_drop'. Then, it drops these columns from the DataFrame 'df', effectively removing features with a high proportion of missing values.

8) *Some columns have no variations. The variables 'examide' and 'citoglipton' have only one value. These columns are not useful in prediction and can be deleted. Delete the following near zero-variance colums: 'repaglinide','nateglinide','chlorpropamide','glimepiride','acetohexamide','tolbutamide','acarbose','miglitol','troglitazone','tolazamide','examide','citoglipton','glyburide-*

metformin','glipizide-metformin','glimepiride-pioglitazone','metformin-rosiglitazone','metformin-pioglitazone'.

```
In [12]: # List of near zero-variance columns to be deleted
columns_to_delete = [
    'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'acetohexamide',
    'tolbutamide', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide', 'examide',
    'citoglipton', 'glyburide-metformin', 'glipizide-metformin', 'glimepiride-pioglitazone',
    'metformin-rosiglitazone', 'metformin-pioglitazone'
]

# Drop the near zero-variance columns
df = df.drop(columns=columns_to_delete)

# Display the updated DataFrame after dropping columns
print("\nDataFrame after Dropping Near Zero-Variance Columns:")
print(df)
```

9) *Drop rows with null values.*

```
In [13]: # Drop rows with null values
df = df.dropna()

# Display the updated DataFrame after dropping rows
print("\nDataFrame after Dropping Rows with Null Values:")
print(df.head())
```

Explanation: This command removes rows with any missing values from the DataFrame 'df'. It ensures that only complete cases are retained for further analysis, potentially improving the quality of the dataset.

prerequisites: We would typically use the IQR method on a DataFrame containing only numerical columns when you suspect the presence of outliers in your dataset. Outliers can skew statistical analyses and machine learning models, leading to biased results. By applying the IQR method, you can identify these outliers and decide whether to remove them or handle them in a different way, such as imputation or transformation.

We choose IQR for Outlier Detection and Boxplots in comparison of Z-Score for below reasons:

- Data Distribution: IQR is generally more robust to outliers in such distributions compared to z-scores. Z-scores can be overly sensitive to outliers in non-normal distributions, potentially removing valid data points.
- Boxplots: Boxplots are a visualization tool that effectively complements IQR for outlier detection. They display the quartiles (Q1, Q3), the IQR (Q3 - Q1), and potential outliers as data points outside the whiskers (typically 1.5 IQR from the quartiles). This visual representation helps identify potential outliers based on their distance from the central tendency of the data.
- As a result, even though z-scores are a popular technique for detecting outliers, IQR might be a better option in this particular case because it is less sensitive to possible data departures from normalcy and pairs well with the boxplot visualisation to help identify outliers.

10) *Display the summary statistics of the numerical columns. Could you identify any outliers in the data? Remove outliers from the data.*

In [23]:
```
import pandas as pd

def remove_outliers(df, columns):
    # Copy the DataFrame to avoid modifying the original
    df_cleaned = df.copy()

    # Iterate over specified columns and remove outliers using IQR method
    for col in columns:
        # Calculate the first and third quartiles
        Q1 = df_cleaned[col].quantile(0.25)
        Q3 = df_cleaned[col].quantile(0.75)

        # Calculate the IQR (Interquartile Range)
        IQR = Q3 - Q1

        # Define the lower and upper bounds to identify outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Identify and remove outliers
        outliers_mask = (df_cleaned[col] < lower_bound) | (df_cleaned[col] > upper_bound)
        df_cleaned = df_cleaned[~outliers_mask]  # Remove rows with outliers

    return df_cleaned
# Columns to consider for outlier removal
columns_to_remove_outliers = ['time_in_hospital', 'num_lab_procedures', 'num_procedures',
                              'num_medications', 'number_outpatient', 'number_emergency',
                              'number_inpatient']

# Call the function with the specified columns
df1 = remove_outliers(df, columns_to_remove_outliers)
print(df1)
```

Explanation: We created the python function to calculate the IQR and remove the outlier (Function Name: Remove_outliers) while we are passing the df we are using above and columns we are passing only numeric values also added the numeric values column that need to calculate, after defining the function we are calling it and passing the df and column in array and defining as df.

11) *Perform feature normalization, if required.*

We are not performing the normalization on the df as it expected the norm function scale up the data and return the adjusted level of the data accordingly to the columns but when we match the data with every aspect we don't need the data to be normalized.

12) *Show the shape of resulting data frame.*

```
# Call the function with the specified columns
df1 = remove_outliers(df, columns_to_remove_outliers)
df1.shape
```

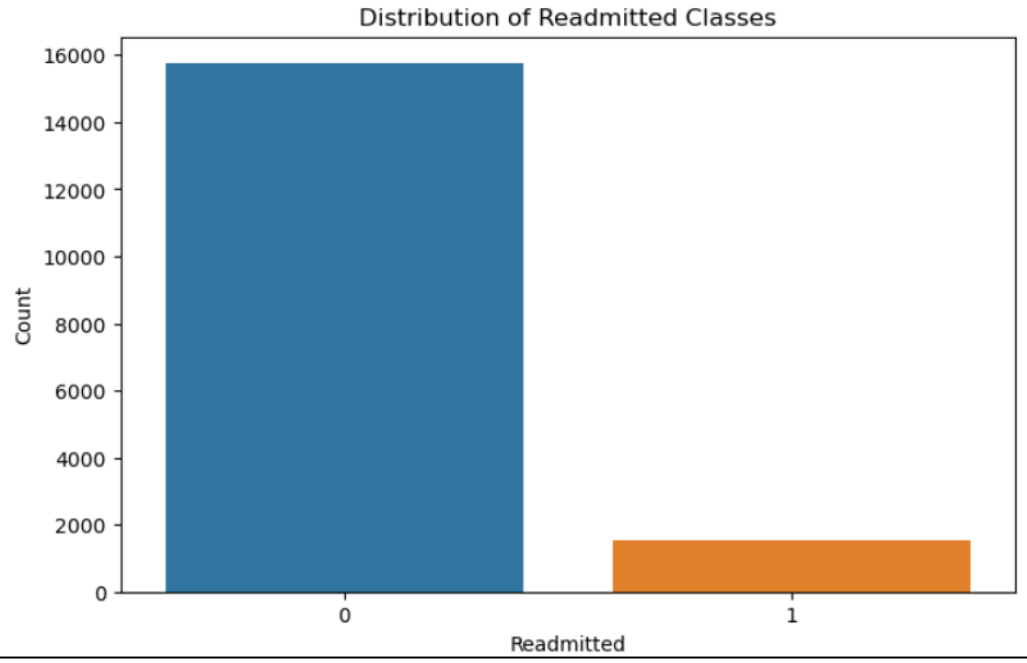
[34]: (17272, 31)

2. Data Visualization

The second step, data exploration, comes after the initial steps of data cleansing and transformation. In this phase, we relate and investigate the characteristics of the condensed dataset using the output from step 1.

1) Plot the distribution of unique classes of the target variable, i.e., readmitted.

```
In [24]: import seaborn as sns
         import matplotlib.pyplot as plt

         plt.figure(figsize=(8, 5))
         sns.countplot(x='readmitted', data=df1)
         plt.title('Distribution of Readmitted Classes')
         plt.xlabel('Readmitted')
         plt.ylabel('Count')
         plt.show()
```

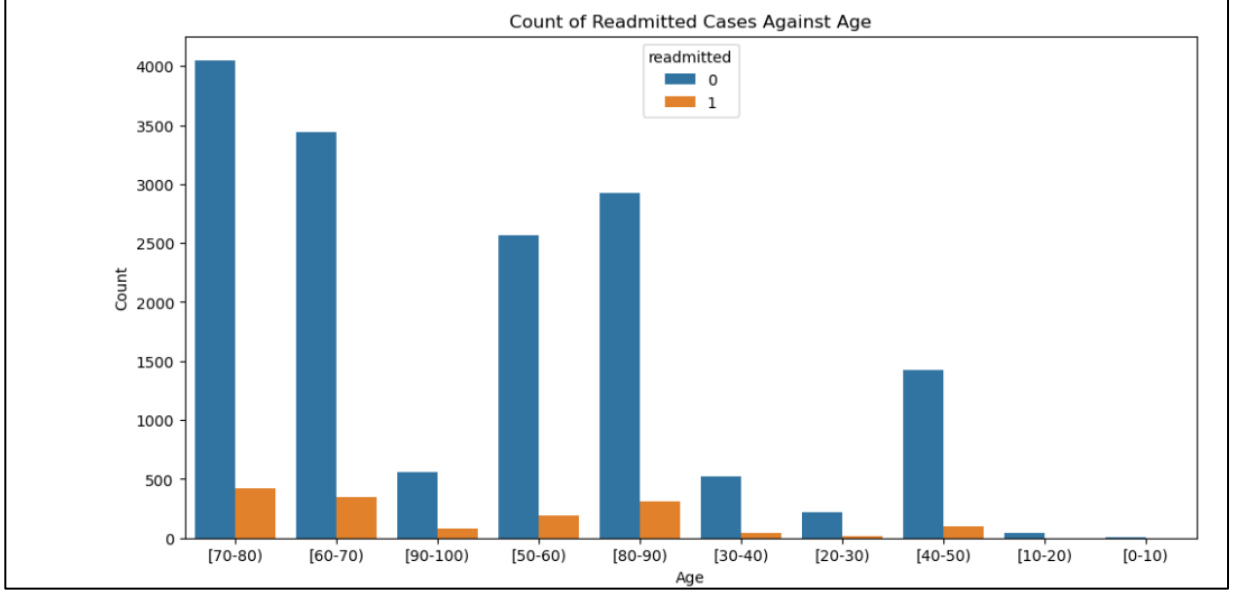


Explanation: This code creates a count plot using Seaborn and Matplotlib libraries to visualize the distribution of the 'readmitted' classes in the DataFrame 'df1'. The count plot displays the frequency of each class of the 'readmitted' variable.

- `plt.figure(figsize=(8, 5))`: Sets the figure size for the plot.
- `sns.countplot(x='readmitted', data=df1)`: Creates the count plot using Seaborn's `countplot` function, where 'readmitted' is the column to plot on the x-axis and `data=df1` specifies the DataFrame to use.
- `plt.title('Distribution of Readmitted Classes')`: Sets the title of the plot.
- `plt.xlabel('Readmitted')`: Sets the label for the x-axis.
- `plt.ylabel('Count')`: Sets the label for the y-axis.
- `plt.show()`: Displays the plot.

2) Plot the count of number of readmitted cases against age.

```
In [24]: #Plot the count of number of readmitted cases against age.
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame with the 'readmitted' and 'age' columns
plt.figure(figsize=(12, 6))
sns.countplot(x='age', hue='readmitted', data=df1)
plt.title('Count of Readmitted Cases Against Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

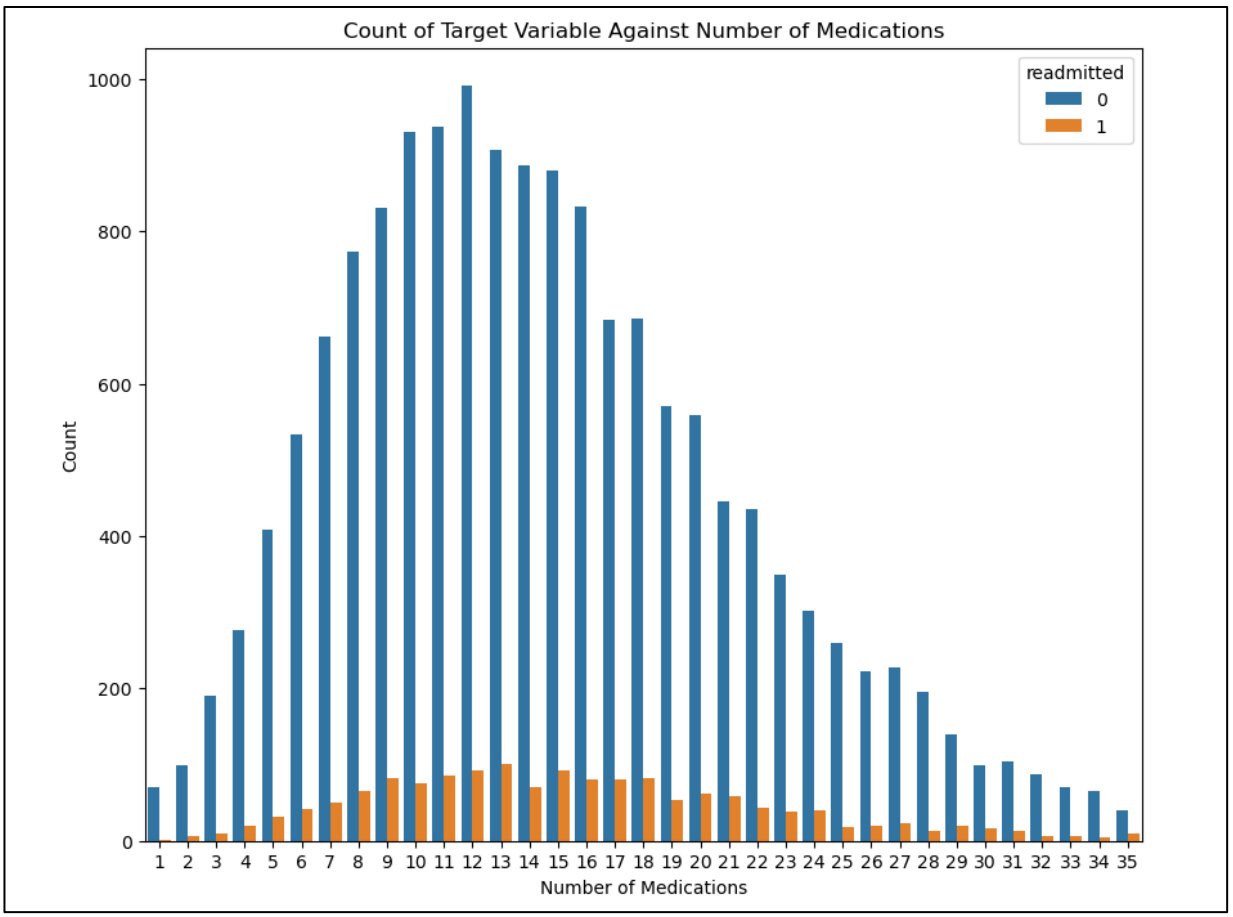


This code creates a count plot using Seaborn and Matplotlib libraries to visualize the count of readmitted cases against different age groups in the DataFrame 'df1'. The count plot is grouped by the 'readmitted' classes for each age group.

Explanation:
- `plt.figure(figsize=(12, 6))`: Sets the figure size for the plot.
- `sns.countplot(x='age', hue='readmitted', data=df1)`: Creates the count plot using Seaborn's `countplot` function, where 'age' is plotted on the x-axis, and the count is grouped by the 'readmitted' classes (represented by different colors).
- `plt.title('Count of Readmitted Cases Against Age')`: Sets the title of the plot.
- `plt.xlabel('Age')`: Sets the label for the x-axis.
- `plt.ylabel('Count')`: Sets the label for the y-axis.
- `plt.show()`: Displays the plot.

3) *Plot a graph that displays the count of target variable against the number of medications.*

```
In [28]: import seaborn as sns
         import matplotlib.pyplot as plt

         # Assuming 'df' is your DataFrame with the 'readmitted' and 'num_medications' columns
         plt.figure(figsize=(20, 16))
         sns.countplot(x='num_medications', hue='readmitted', data=df1)
         plt.title('Count of Target Variable Against Number of Medications')
         plt.xlabel('Number of Medications')
         plt.ylabel('Count')
         plt.show()
```



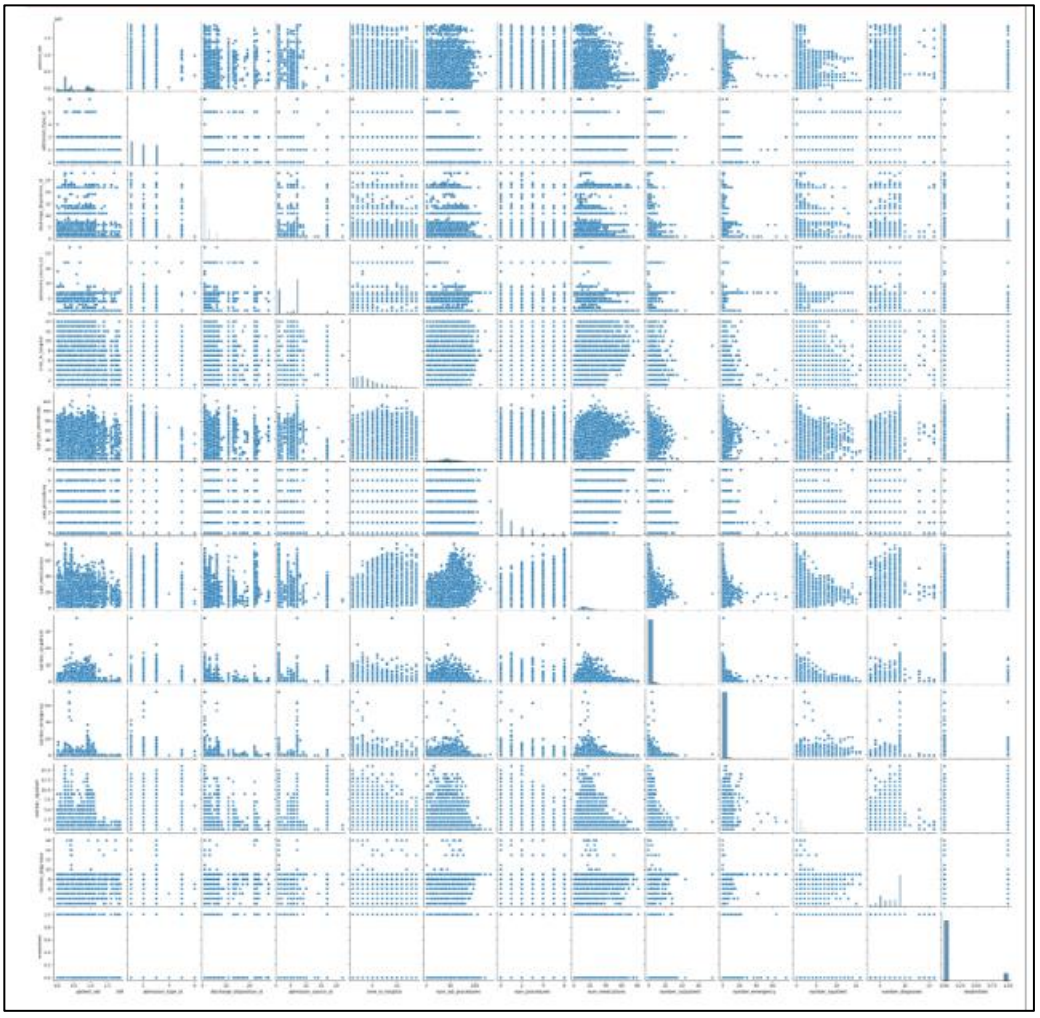
This code creates a count plot using Seaborn and Matplotlib libraries to visualize the count of the target variable ('readmitted') against the number of medications ('num_medications') in the DataFrame 'df1'. The count plot is grouped by the 'readmitted' classes for each number of medications.

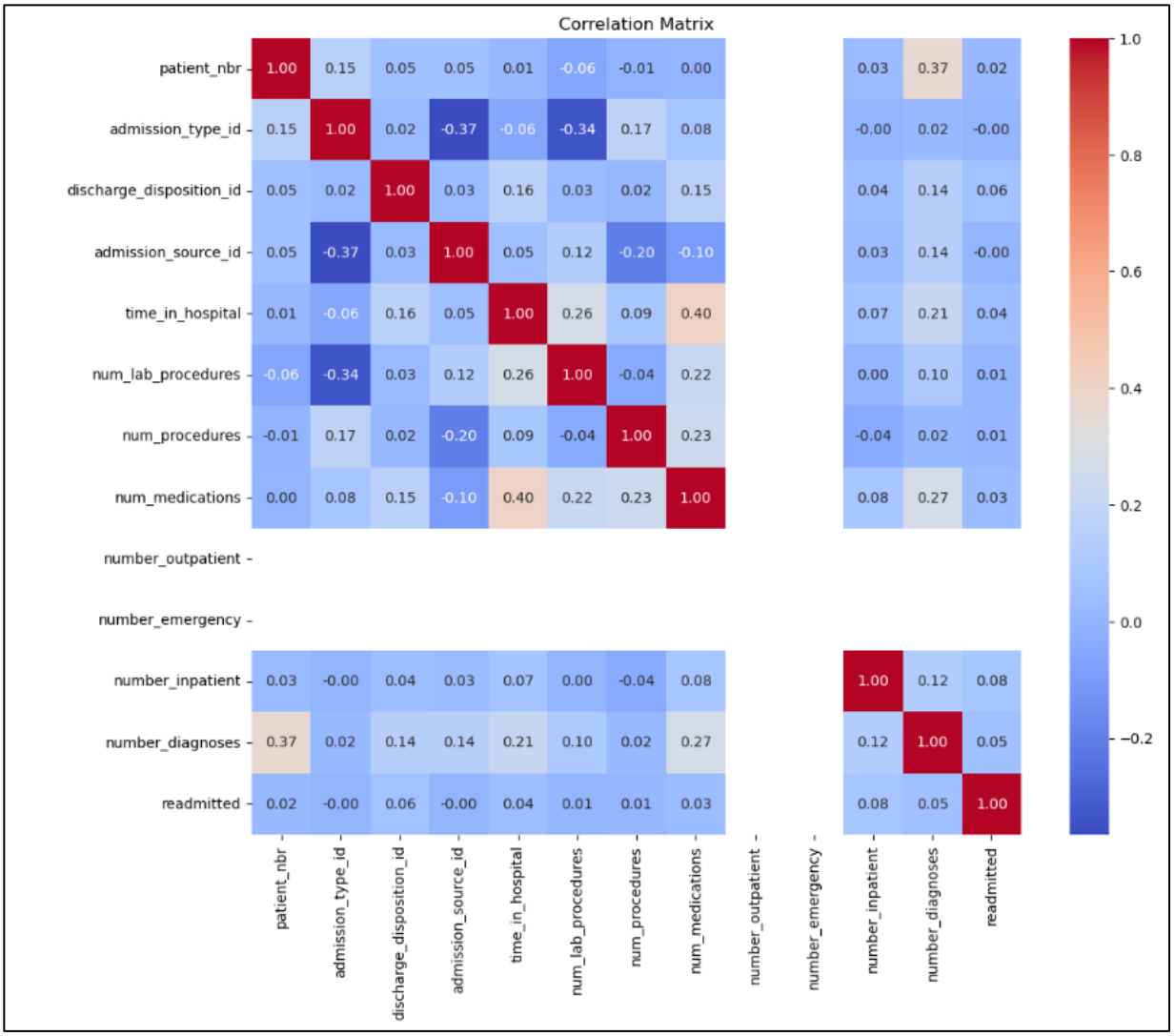
Explanation:
- `plt.figure(figsize=(12, 6))`: Sets the figure size for the plot.

- `sns.countplot(x='num_medications', hue='readmitted', data=df1)`: Creates the count plot using Seaborn's `countplot` function, where 'num_medications' is plotted on the x-axis, and the count is grouped by the 'readmitted' classes (represented by different colors).
- `plt.title('Count of Target Variable Against Number of Medications')`: Sets the title of the plot.
- `plt.xlabel('Number of Medications')`: Sets the label for the x-axis.
- `plt.ylabel('Count')`: Sets the label for the y-axis.
- `plt.show()`: Displays the plot.

4) *Show the scatter matrix plot and the correlation matrix. This should be a very large matrix and you might find it difficult to analyse. Which pair of features are highly correlated?*

In [29]:
```
# Show the scatter matrix plot and the correlation matrix. This should be a very large matrix and you might find it difficult to

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame containing the dataset

# Calculate correlation matrix
corr_matrix = df1.corr()

# Create a scatter matrix plot
sns.pairplot(df)
plt.show()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```





With the use of scatter plots and a heatmap, this code shows the relationships between the various features in the dataset visually, making it easier to find feature pairings that are highly connected.

Explnation:

- corr_matrix = df1.corr(): Calculates the correlation matrix for the DataFrame 'df1'. The correlation matrix shows the pairwise correlations between all the numerical columns in the DataFrame.
- sns.pairplot(df): Creates a scatter matrix plot using Seaborn's pairplot function. This plot visualizes pairwise relationships between variables by plotting scatterplots for each pair of features against each other.
- plt.show(): Displays the scatter matrix plot.
- plt.figure(figsize=(12, 10)): Sets the figure size for the heatmap plot.
- sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f"): Creates a heatmap of the correlation matrix using Seaborn's heatmap function. The heatmap visualizes the correlation values between pairs of features. The 'annot=True' parameter adds numerical annotations to the heatmap, 'cmap='coolwarm" sets the color map for the heatmap, and 'fmt=".2f"' formats the annotations to display correlation values with two decimal places.
- plt.title('Correlation Matrix'): Sets the title for the heatmap plot.
- plt.show(): Displays the correlation matrix heatmap.

Now we would get to know which pair of features are highly correlated?

```
In [31]: import pandas as pd

         # Calculate correlation matrix
         corr_matrix = df1.corr(numeric_only=True)

         # Print the correlation matrix
         print("Correlation Matrix:")
         print(corr_matrix)

         # Identify pairs of features that are highly correlated
         highly_correlated_pairs = []
         threshold = 0.9  # Adjust the threshold as needed
         for i in range(len(corr_matrix.columns)):
             for j in range(i+1, len(corr_matrix.columns)):
                 if abs(corr_matrix.iloc[i, j]) > threshold:
                     highly_correlated_pairs.append((corr_matrix.columns[i], corr_matrix.columns[j]))

         # Print highly correlated pairs
         print("\nHighly Correlated Feature Pairs:")
         for pair in highly_correlated_pairs:
             print(pair)
```

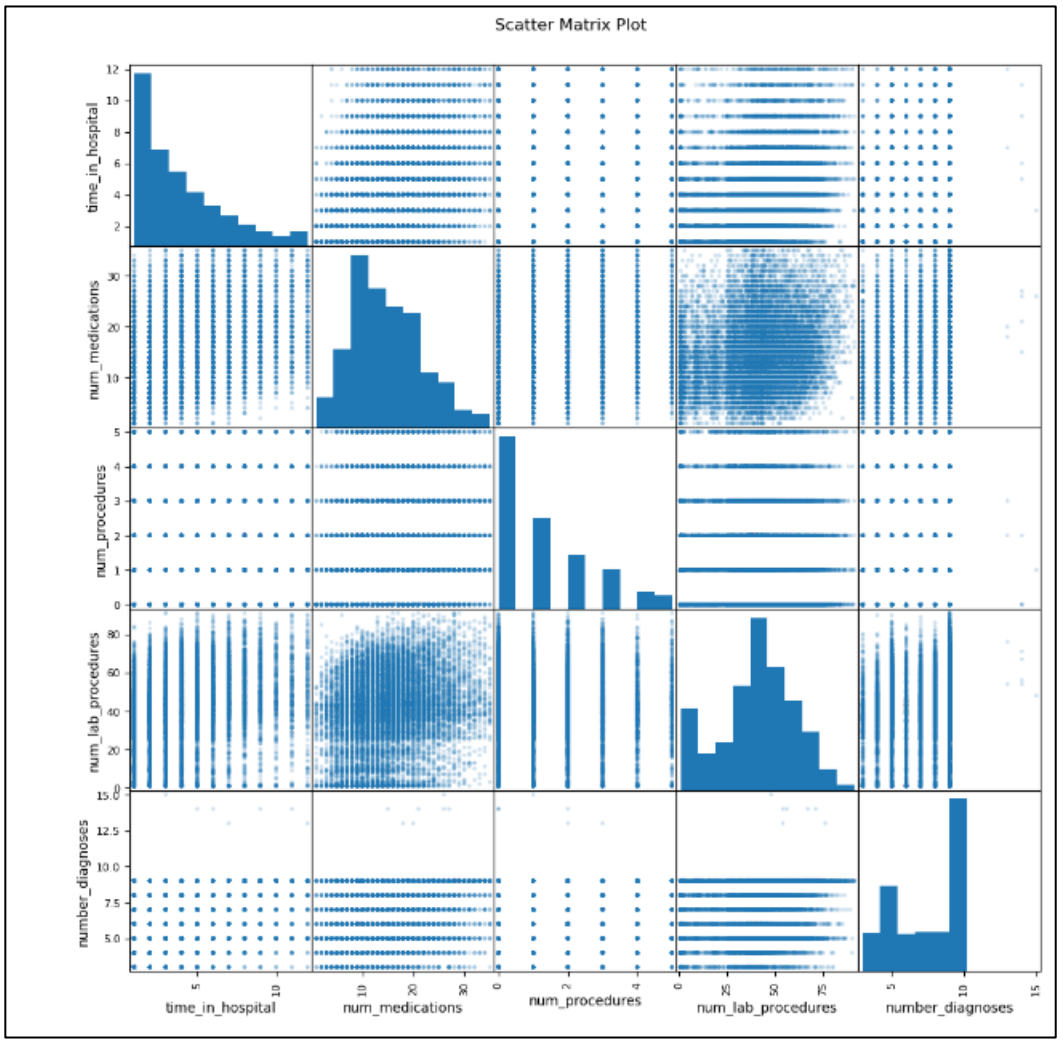
This code is useful for identifying pairs of features that are highly correlated based on a specified threshold, which can help in feature selection or multicollinearity detection in statistical modeling tasks.

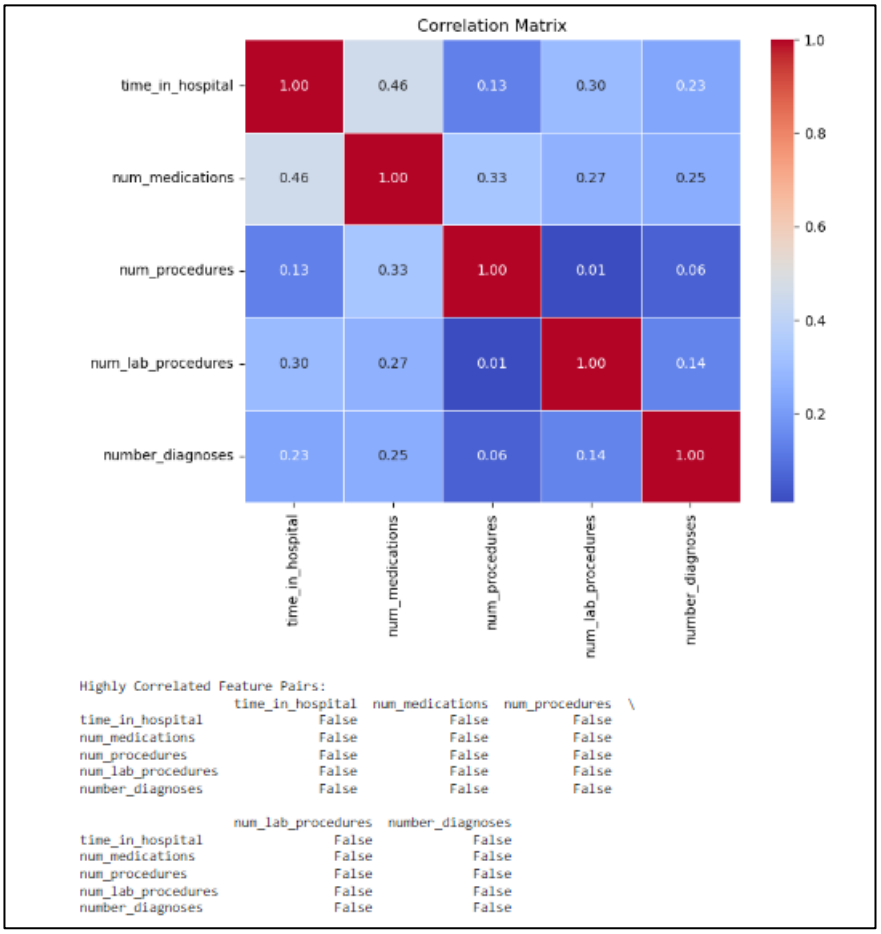
Explanation:

- corr_matrix = df1.corr(numeric_only=True): Calculates the correlation matrix for the DataFrame 'df1'. The numeric_only=True parameter ensures that only numerical columns are considered for calculating correlations.
- threshold = 0.9: Sets the threshold value for identifying highly correlated features. You can adjust this threshold based on your requirements.
- abs(corr_matrix.iloc[i, j]) > threshold: Checks if the absolute correlation value between the pair of features (at position [i, j] in the correlation matrix) exceeds the defined threshold.
- If the condition is met, the pair of feature names (corr_matrix.columns[i], corr_matrix.columns[j]) is appended to the list highly_correlated_pairs.

5) *Generate additional plots that demonstrate your understanding of the problem and the data. You are free to select the plot and features for visualisation. For better visualisation and understanding of data, consider using seaborn library.*

In [33]:
```
#Generate additional plots that demonstrate your understanding of the problem and the data. You are free to select the plot and f
import pandas as pd
import seaborn as sns
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt



# Select a subset of columns for demonstration (adjust as needed)
subset_columns = ['time_in_hospital','num_medications', 'num_procedures', 'num_lab_procedures','number_diagnoses']

# Create a scatter matrix plot
scatter_matrix(df1[subset_columns], alpha=0.2, figsize=(12, 12), diagonal='hist')
plt.suptitle('Scatter Matrix Plot', y=0.92)
plt.show()

# Display the correlation matrix with seaborn heatmap
correlation_matrix = df[subset_columns].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()

# Identify highly correlated pairs of features #why we select 0.7 threshold
highly_correlated_pairs = (correlation_matrix.abs() > 0.7) & (correlation_matrix.abs() < 1)
print("Highly Correlated Feature Pairs:")
print(highly_correlated_pairs)
```





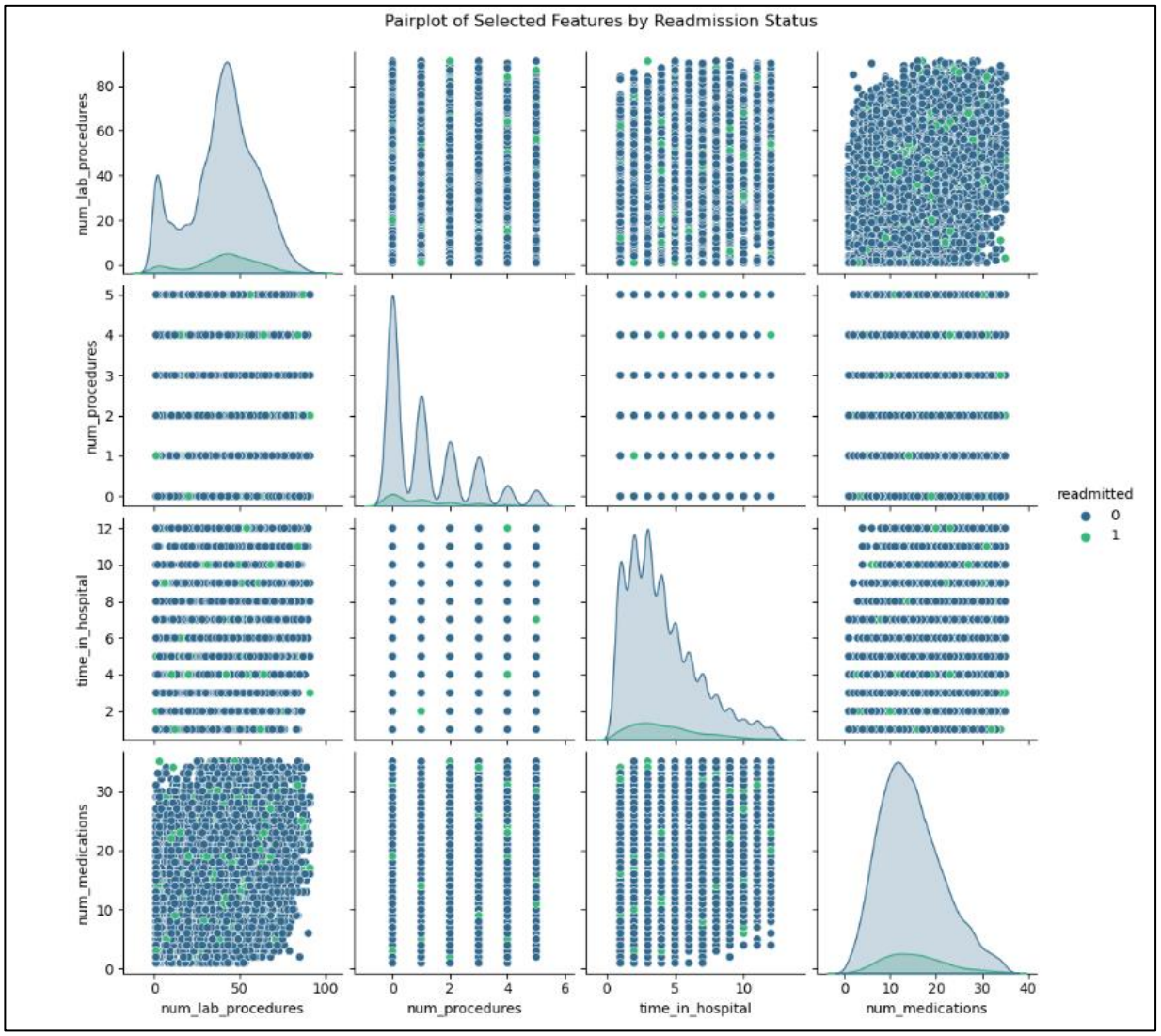
This code provides visualizations and analysis to understand relationships and correlations between different features in the dataset, aiding in exploratory data analysis and feature selection processes.

Explanation:

- subset_columns: Defines a subset of columns from the DataFrame 'df1' for visualization purposes. These columns are chosen to demonstrate relationships between different features in the dataset.
- scatter_matrix(df1[subset_columns], alpha=0.2, figsize=(12, 12), diagonal='hist'): Creates a scatter matrix plot using pandas' scatter_matrix function. This plot displays pairwise relationships between features by plotting scatterplots for each pair of features against each other. The alpha=0.2 parameter adjusts the transparency of the points, and diagonal='hist' specifies that histograms should be drawn on the diagonal.
- plt.suptitle('Scatter Matrix Plot', y=0.92): Sets the super title for the scatter matrix plot.
- correlation_matrix = df[subset_columns].corr(): Calculates the correlation matrix for the subset of columns defined earlier. This matrix shows the pairwise correlations between the selected features.
- sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5): Creates a heatmap of the correlation matrix using Seaborn's heatmap function. The heatmap visualizes the correlation values between pairs of features. Parameters like annot=True add numerical annotations to the heatmap, cmap='coolwarm' sets the color map for the heatmap, fmt=".2f" formats the annotations to display correlation values with two decimal places, and linewidths=0.5 adjusts the width of the lines separating cells.

About pair plot to visualize relation between multiple variables:

In [36]:
```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# Pairplot to visualize relationships between multiple variables
# Using seaborn's pairplot to create a grid of scatterplots for selected numeric features
# 'num_lab_procedures', 'num_procedures', 'time_in_hospital', 'num_medications'
# 'readmitted' is used to color points based on the 'readmitted' status
sns.pairplot(df1[['num_lab_procedures', 'num_procedures', 'time_in_hospital', 'num_medications', 'readmitted']],
             hue='readmitted', diag_kind='kde', palette='viridis')

# Adding a title to the pairplot
plt.suptitle('Pairplot of Selected Features by Readmission Status', y=1.02)


# Display the pairplot
plt.show()
```



The visual examination of the correlations between specific attributes in the dataset is made easier by this code, which focuses in particular on how these associations change depending on the readmission status.

Explanation:

- sns.pairplot(): Generates a grid of scatterplots, with each numeric feature plotted against every other numeric feature. It's a convenient way to visualize relationships between multiple variables simultaneously. In this case, the pairplot is created for the DataFrame 'df1' with selected features: 'num_lab_procedures', 'num_procedures', 'time_in_hospital', 'num_medications'. The 'hue' parameter is set to 'readmitted', which colors the points based on the 'readmitted' status.
- hue='readmitted': Colors the points in the pairplot based on the 'readmitted' status, allowing for easy differentiation between readmission categories.
- diag_kind='kde': Specifies that the diagonal elements of the pairplot should be kernel density estimate (KDE) plots instead of histograms. KDE plots provide a smooth estimate of the distribution of each variable.
- palette='viridis': Sets the color palette for the plot to 'viridis', which is a visually pleasing color scheme.
- plt.suptitle('Pairplot of Selected Features by Readmission Status', y=1.02): Adds a super title to the pairplot with an adjusted vertical position (y=1.02) to prevent overlapping with plot elements.

3. Model Building

The act of developing a mathematical model from the data in a dataframe so that it can be used to make predictions or learn more about the data is known as "model building" in a dataframe. To create accurate predictions or learn more about new data, a model must accurately reflect the underlying patterns and relationships in the data. This is the aim of model building.

1) Select the predictors that would have impact in predicting readmission.

In [27]:
```
import numpy as np
# Selecting the numeric columns from dataframe
numeric_df = df1.select_dtypes(include=[np.number])

# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Print correlation of each feature with the target variable 'readmitted'
print(correlation_matrix['readmitted'].sort_values(ascending=False))
```

```
readmitted                 1.000000
number_inpatient           0.079551
discharge_disposition_id   0.059539
number_diagnoses           0.051088
time_in_hospital           0.038489
num_medications            0.033826
patient_nbr                0.020128
num_procedures             0.009319
num_lab_procedures         0.005348
admission_type_id         -0.000857
admission_source_id       -0.001672
number_outpatient               NaN
number_emergency                NaN
Name: readmitted, dtype: float64
```

This code performs the task such as, select numeric columns, Calculate correlation matrix and printing correlation and target variable ‘readmitted’.

Explanation:

- df1.select_dtypes(include=[np.number]): Selects numeric columns from the DataFrame df1 using the select_dtypes method, ensuring that only columns with numeric data types are included.
- numeric_df: Stores the DataFrame containing only the numeric columns.
- numeric_df.corr(): Calculates the correlation matrix for the numeric columns in numeric_df.
- correlation_matrix['readmitted']: Selects the correlation values of each feature with the target variable 'readmitted'.
- .sort_values(ascending=False): Sorts the correlation values in descending order to show the highest positive correlations first.

2) *Build up a first linear model with appropriate predictors and evaluate it. Split the data into a training and test sets. Evaluate your model by using a cross-validation procedure.*

In [29]:
```
# Selecting the categorial columns from dataframe
cat_df = df1.select_dtypes(include=[object])
```

In [31]:
```
# One-hot encode categorical variables
df2 = pd.get_dummies(df1, columns=cat_df.columns)
```

In [32]:
```
df2.shape
```

Out[32]:
```
(17272, 1727)
```

In [33]:
```
# Separate predictors and target variable
X = df2.drop('readmitted', axis=1)
y = df2['readmitted']
```

In [34]:
```
# Importing necessary libraries
from sklearn.feature_selection import SelectKBest, f_classif

# Selecting the top 5 features using Univariate feature selection
# Here, f_classif is used as the score function, which computes the ANOVA F-value for the provided dataset
selector = SelectKBest(score_func=f_classif, k=7)

# Transforming the input features (X) to get the selected features (X_selected)
X_selected = selector.fit_transform(X, y)

# Getting the names of the selected features
# This gets the column names of X (the original dataset) that were selected by the selector
selected_features = X.columns[selector.get_support()]

# Printing the selected feature names
print(selected_features)
```

```
Index(['discharge_disposition_id', 'number_inpatient', 'number_diagnoses',
       'diag_1_V58', 'change_No', 'diabetesMed_No', 'diabetesMed_Yes'],
      dtype='object')
```

```
In [35]: from sklearn.model_selection import train_test_split, cross_val_score
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error

In [37]: X2 = df2[['insulin_No','insulin_Steady','change_No','diabetesMed_No', 'diabetesMed_Yes','number_inpatient', 'number_diagnoses']]
         y2 = df2['readmitted']

In [39]: X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=42)

In [40]: # Create the linear regression model
         model = LinearRegression()

         # Train the model
         model.fit(X_train, y_train)

Out[40]: LinearRegression()
         In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
         On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

Using the training data, this code creates a linear regression model with the proper predictors, trains it, and then uses cross-validation to assess the model's performance.

This code performs following tasks:

- Selecting Categorical Columns: The pandas select_dtypes function is used to pick just the categorical columns from the DataFrame df1. Only columns with object data types—typically categorical columns—are included thanks to the include=[object] parameter.One
- Hot Encoding: Using pd.get_dummies, it one-hot encodes the categorical variables in the DataFrame df1. By doing this, binary variables that were previously category are now eligible for use in machine learning models.
- Separating Predictors and Target Variable: It takes the DataFrame df2 and separates the predictors (X) and target variable (y), where X has all the features except the target variable "readmitted," and y has just the target variable "readmitted."
- Feature Selection: The univariate feature selection approach is used to carry out feature selection. Choose KBest using scikit-learn. In this case, the score function that determines the ANOVA F-value for the given dataset is f_classif. Rank-wise, the top 7 traits (k=7) are chosen according to their significance.
- Training and Evaluating the Linear Regression Model: First, a linear regression model is trained on the training data after the data is divided into training and test sets using train_test_split. Lastly, it uses the cross-validation process that cross_val_score provides to assess the model.

Explanation:

- from sklearn.feature_selection import SelectKBest, f_classif: Brings in the libraries required for the feature selection process.
- selector = SelectKBest(score_func=f_classif, k=7): Chooses the top 7 features by initialising the SelectKBest feature selection method with f_classif as the scoring function.
- X_selected = selector.fit_transform(X, y): Performs feature selection on the predictor variables X using the target variable y, retaining only the top 7 selected features.
- selected_features = X.columns[selector.get_support()]: Retrieves the names of the selected features.
- from sklearn.model_selection import train_test_split, cross_val_score: imports the libraries required for data partitioning and cross-validation.

- X2, y2, test_size=0.2, random_state=42) = train_test_split(X_train, X_test, y_train, y_test): divides the data into test and training sets.
- model.fit(X_train, y_train): Trains the linear regression model on the training data.

3) *Use different performance metrics to evaluate the performance of your model. You might have noticed that the data is imbalanced. The number of positive examples are roughly 10% of the total dataset. Choose appropriate performance metrics to evaluate the performance of your model.*

In [41]:
```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE) on the test set
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error on Test Set:", mse)
```

```
Mean Squared Error on Test Set: 0.08130072802473882
```

In [42]:
```
# Perform cross-validation (let's say with 5 folds)
cv_scores = cross_val_score(model, X2, y2, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_scores  # Convert negative MSE to positive

# Calculate the average MSE from cross-validation
avg_mse = np.mean(cv_mse_scores)
print("Average MSE from Cross-Validation:", avg_mse)
```

```
Average MSE from Cross-Validation: 0.07946897452181771
```

In [43]:
```
# Importing necessary libraries for performance metrics
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, accuracy_score, classifica

# Converting predicted probabilities to binary predictions based on a threshold of 0.5
threshold = 0.7  # You can adjust this
y_pred_binary = (y_pred > threshold).astype(int)

# Calculating Precision
precision = precision_score(y_test, y_pred_binary)

# Calculating Recall
recall = recall_score(y_test, y_pred_binary)

# Calculating F1 Score
f1 = f1_score(y_test, y_pred_binary)

# Calculating Accuracy
accuracy = accuracy_score(y_test, y_pred_binary)

# Calculating ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_pred)

# Calculating Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_binary)

# Generating Classification Report
clf_report = classification_report(y_test, y_pred_binary)

# Printing the calculated performance metrics
print("Precision:", precision)
print("Accuracy:", accuracy)
print("Recall:", recall)
print("F1 Score:", f1)
print("ROC-AUC Score:", roc_auc)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(clf_report)
```

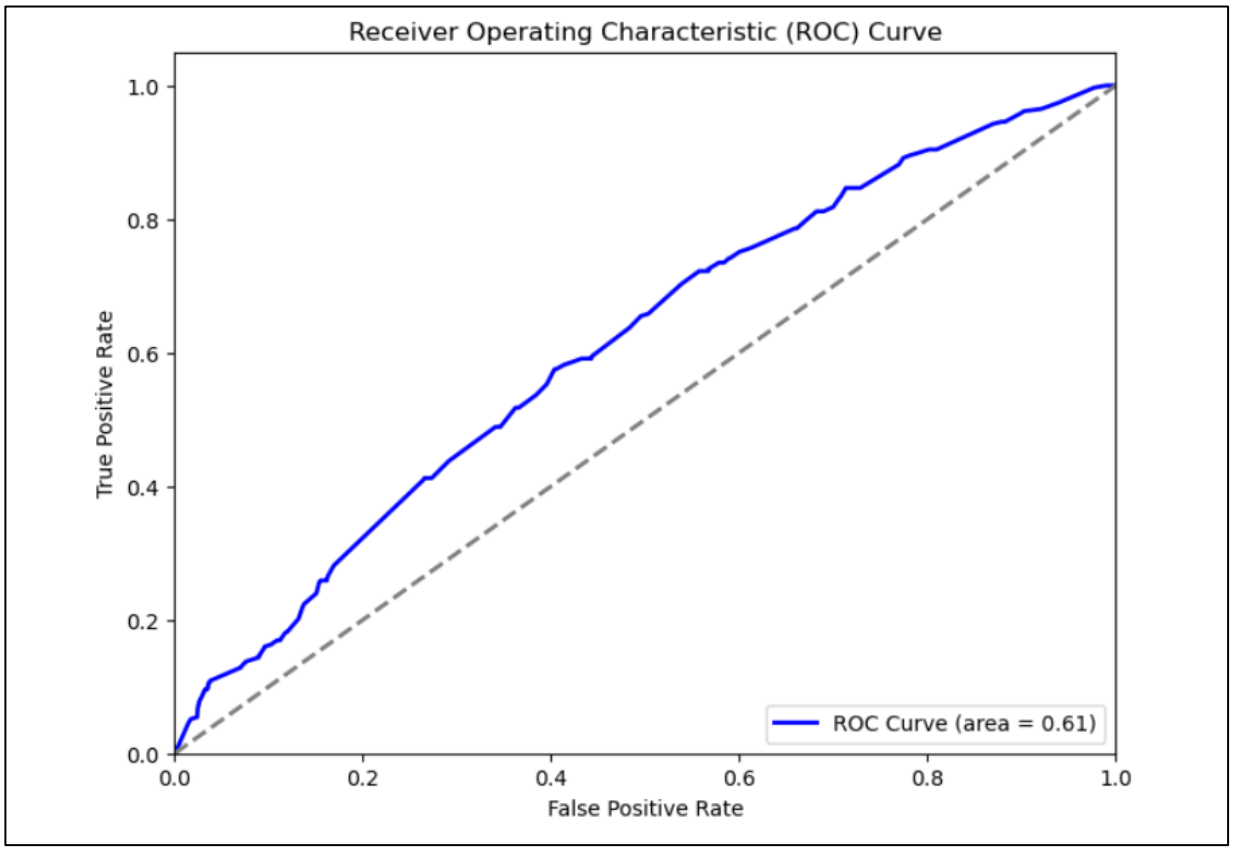
```
Precision: 0.0
Accuracy: 0.9094066570188133
Recall: 0.0
F1 Score: 0.0
ROC-AUC Score: 0.6104661567589884
Confusion Matrix:
[[3142    0]
 [ 313    0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95      3142
           1       0.00      0.00      0.00       313

    accuracy                           0.91      3455
   macro avg       0.45      0.50      0.48      3455
weighted avg       0.83      0.91      0.87      3455
```

In [44]:

```
# Importing necessary libraries
from sklearn.metrics import roc_curve, auc, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Splitting data into train and test data
# X2 is your feature matrix, y2 is your target vector
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Building a Logistic Regression model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Predicting probabilities on the testing data
y_pred_proba = model.predict_proba(X_test_scaled)

# Importing necessary libraries for performance metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, classifica

# Converting predicted probabilities to binary predictions
# np.argmax(y_pred_proba, axis=1) will convert the predicted probabilities to the index of the maximum value along the second ax
# effectively giving you the predicted class labels.
y_pred_binary = np.argmax(y_pred_proba, axis=1)

# Calculating Precision
precision = precision_score(y_test, y_pred_binary)

# Calculating Recall
recall = recall_score(y_test, y_pred_binary)

# Calculating F1 Score
f1 = f1_score(y_test, y_pred_binary)

# Calculating Accuracy
accuracy = accuracy_score(y_test, y_pred_binary)

# Calculating ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_pred_proba[:, 1])

# Calculating Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_binary)

# Generating Classification Report
clf_report = classification_report(y_test, y_pred_binary)

# Printing the calculated performance metrics
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Accuracy:", accuracy)
print("ROC-AUC Score:", roc_auc)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(clf_report)
```

```
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
Accuracy: 0.9094066570188133
ROC-AUC Score: 0.6107447689044442
Confusion Matrix:
[[3142    0]
 [ 313    0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95      3142
           1       0.00      0.00      0.00       313

    accuracy                           0.91      3455
   macro avg       0.45      0.50      0.48      3455
weighted avg       0.83      0.91      0.87      3455
```

In [45]:
```
# Compute ROC curve and ROC area for each class
# y_pred_proba[:, 1] selects the predicted probabilities for the positive class
fpr, tpr, _ = roc_curve(y_test, y_pred_proba[:, 1])

# Compute Area Under the Curve (AUC) for ROC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC Curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Plot Confusion Matrix
# Predicting labels on the test set
y_pred = model.predict(X_test_scaled)

# Calculating the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



This code snippet evaluates the performance of a binary classification model using different performance metrics, considering the imbalanced nature of the data. It performs the following tasks:

- Mean Squared Error (MSE): Calculates the MSE on the test set to measure the average squared difference between the actual and predicted values.
- Cross-Validation (CV): Determines the average MSE by doing five folds of cross-validation.
- Performance Metrics: Calculates various performance metrics including Precision, Recall, F1 Score, Accuracy, and ROC-AUC Score.
- Confusion Matrix: To comprehend the model's performance in terms of true positive, true negative, false positive, and false negative predictions, compute and visualise the confusion matrix.
- Receiver Operating Characteristic (ROC) Curve: Plots the ROC curve to visualize the trade-off between true positive rate and false positive rate at various threshold settings.

Explanation:

- Mean Squared Error (MSE):
  It measures the average squared difference between the actual and predicted values. And It's a widely used statistic in regression analysis.
- Cross-Checking (CV):
  It's a method for evaluating how well a prediction model applies to a different dataset. And Here, it's used to calculate the average MSE from cross-validation.
- Performance Metrics:
  Precision: It measures the ratio of correctly predicted positive observations to the total predicted positives.
  Recall: It measures the ratio of correctly predicted positive observations to the all observations in actual class.
  F1 Score: It is the weighted average of Precision and Recall.
  Accuracy: It measures the ratio of correctly predicted observations to the total observations.
  ROC-AUC Score: It measures the area under the ROC curve, which indicates the model's ability to discriminate between positive and negative classes.
- Confusion Matrix:
  It is a table used to describe the performance of a classification model.
  It shows the number of true positive, true negative, false positive, and false negative predictions.
- Receiver Operating Characteristic (ROC) Curve:
  It is a graphical plot that illustrates the diagnostic ability of a binary classification model across different threshold settings.
  It is a common tool for evaluating the performance of binary classification models.

4. *Balance your data using undersampling or oversampling data balance technique. Train your model again and evaluate its performance. Did you achieve better prediction accuracies with more balanced data?*

In [46]:
```
from imblearn.under_sampling import RandomUnderSampler
# Create the RandomUnderSampler object
undersampler = RandomUnderSampler(random_state=42)

# Resample the data
X_undersampled, y_undersampled = undersampler.fit_resample(X2, y2)

# Check the new class distribution
print("Undersampled Data Distribution:")
print(pd.Series(y_undersampled).value_counts())
```

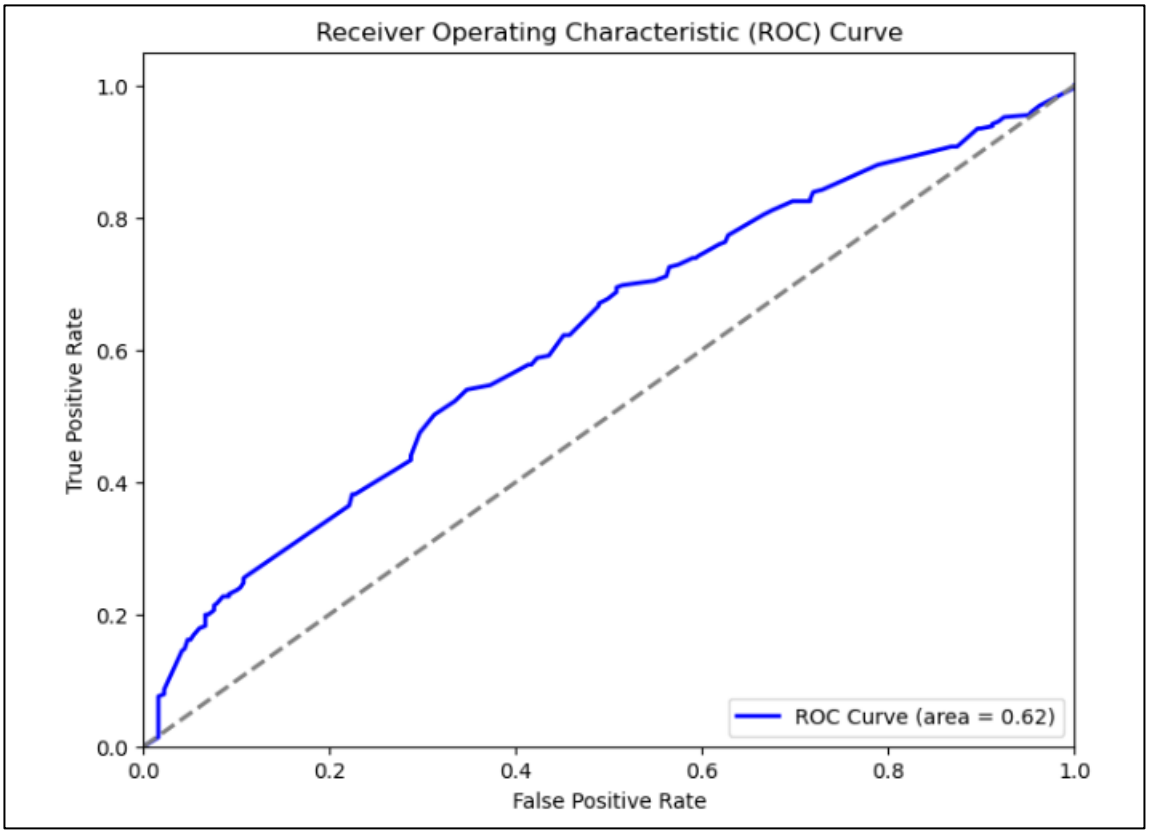
```
Undersampled Data Distribution:
0    1519
1    1519
Name: readmitted, dtype: int64
```

In [47]:
```
# Choose either X_undersampled, y_undersampled based on the method you want to use

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_undersampled, y_undersampled, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [48]:
```
# Build the Logistic Regression model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Predict probabilities on the testing data
y_pred_proba = model.predict_proba(X_test_scaled)

# Convert predicted probabilities to binary predictions
y_pred_binary = np.argmax(y_pred_proba, axis=1)

# Calculate performance metrics
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
accuracy = accuracy_score(y_test, y_pred_binary)
roc_auc = roc_auc_score(y_test, y_pred_proba[:, 1])
conf_matrix = confusion_matrix(y_test, y_pred_binary)

# Print performance metrics
print("Performance Metrics on Balanced Data:")
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Accuracy:", accuracy)
print("ROC-AUC Score:", roc_auc)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Performance Metrics on Balanced Data:
Precision: 0.5606557377049181
Recall: 0.5876288659793815
F1 Score: 0.5738255033557048
Accuracy: 0.5822368421052632
ROC-AUC Score: 0.6212180341908137
Confusion Matrix:
[[183 134]
 [120 171]]
```

In [49]:
```
# Compute ROC curve and ROC area for each class
fpr, tpr, _ = roc_curve(y_test, y_pred_proba[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC Curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Plot Confusion Matrix
y_pred = model.predict(X_test_scaled)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
In [50]: from imblearn.over_sampling import RandomOverSampler
         # Create the RandomOverSampler object
         oversampler = RandomOverSampler(random_state=42)

         # Resample the data
         X_oversampled, y_oversampled = oversampler.fit_resample(X2, y2)

         # Check the new class distribution
         print("Oversampled Data Distribution:")
         print(pd.Series(y_oversampled).value_counts())

         Oversampled Data Distribution:
         0    15753
         1    15753
         Name: readmitted, dtype: int64
```

This code snippet performs the following tasks:

- Undersampling: It balances the class distribution by randomly undersampling the majority class using the RandomUnderSampler from imblearn.under_sampling. The class distribution is then checked and printed.

- Model Training and Evaluation after Undersampling: It splits the undersampled data into training and test sets, standardizes the features, builds a logistic regression model, and evaluates its performance using various metrics including Precision, Recall, F1 Score, Accuracy, ROC-AUC Score, and Confusion Matrix. Additionally, it plots the ROC curve and the Confusion Matrix.

- Oversampling: Similarly, it balances the class distribution by randomly oversampling the minority class using the RandomOverSampler from imblearn.over_sampling. The new class distribution is then checked and printed.

Explanation:

Undersampling:

- undersampler = RandomUnderSampler(random_state=42): Initializes the RandomUnderSampler object.
- X_undersampled, y_undersampled = undersampler.fit_resample(X2, y2): Undersamples the data by fitting and resampling.
- Prints the new class distribution.

Model Training and Evaluation after Undersampling:

- Splits the undersampled data into training and test sets.
- Standardizes the features using StandardScaler.
- Builds a logistic regression model and evaluates its performance using various metrics.
- Plots the ROC curve and the Confusion Matrix.

Oversampling:

- oversampler = RandomOverSampler(random_state=42): Initializes the RandomOverSampler object.
- X_oversampled, y_oversampled = oversampler.fit_resample(X2, y2): Oversamples the data by fitting and resampling.
- Prints the new class distribution.

Part 2: Improved Model

1) *Consider the entire datasets again. Develop an improved classification model that predicts the patient readmission. Validate your model. You should aim for a model with a higher performance while using a maximum of data points. This implies treating missing values differently for example through imputation rather than dropping them. You may also consider retaining some of the near-zero variance columns.*

In [1]:
```
import pandas as pd
df = pd.read_csv('diabetic_data.csv')
print(pd.__version__)
```

```
2.2.1
```

In [2]:
```
df.shape
```

Out[2]:
```
(101766, 50)
```

In [3]:
```
# Replaces "?" with null values in the dataframe
df.replace("?",pd.NA,inplace=True)
```

In [4]:
```
# Here the issue is that actual dataframe contain the a value named 'None' which is string but I think jupyter is considering it

# Deleting all the forward and backward spaces if present
df['max_glu_serum'] = df['max_glu_serum'].str.strip()

# So we replaced Null values with string named 'None'
df['max_glu_serum'] = df['max_glu_serum'].fillna('None')
```

In [5]:
```
# Applying the same thing to column named 'AC1result' which contains the same issue in the dataframe

# Deleting all the forward and backward spaces if present
df['A1Cresult'] = df['A1Cresult'].str.strip()

# So we replaced Null values with string named 'None'
df['A1Cresult'] = df['A1Cresult'].fillna('None')
```

Fig 2.1. 1

In [6]:
```
missing_percentage = (df.isnull().sum() / len(df)) * 100
print(missing_percentage)
```

```
encounter_id                0.000000
patient_nbr                 0.000000
race                        2.233555
gender                      0.000000
age                         0.000000
weight                     96.858479
admission_type_id           0.000000
discharge_disposition_id    0.000000
admission_source_id         0.000000
time_in_hospital            0.000000
payer_code                 39.557416
medical_specialty          49.082208
num_lab_procedures          0.000000
num_procedures              0.000000
num_medications             0.000000
number_outpatient           0.000000
number_emergency            0.000000
number_inpatient            0.000000
diag_1                      0.020636
diag_2                      0.351787
diag_3                      1.398306
number_diagnoses            0.000000
max_glu_serum               0.000000
A1Cresult                   0.000000
metformin                   0.000000
repaglinide                 0.000000
nateglinide                 0.000000
chlorpropamide              0.000000
glimepiride                 0.000000
acetohexamide               0.000000
glipizide                   0.000000
glyburide                   0.000000
tolbutamide                 0.000000
pioglitazone                0.000000
rosiglitazone               0.000000
acarbose                    0.000000
miglitol                    0.000000
troglitazone                0.000000
tolazamide                  0.000000
examide                     0.000000
citoglipton                 0.000000
insulin                     0.000000
glyburide-metformin         0.000000
glipizide-metformin         0.000000
glimepiride-pioglitazone    0.000000
metformin-rosiglitazone     0.000000
metformin-pioglitazone      0.000000
change                      0.000000
diabetesMed                 0.000000
readmitted                  0.000000
dtype: float64
```

In [7]:
```
# Drop columns with more than 90% missing values
columns_to_drop = missing_percentage[missing_percentage > 90].index
df.drop(columns=columns_to_drop,inplace=True)
```

In [8]:
```
df.shape
```

Out[8]:
```
(101766, 49)
```

In [9]:
```
import numpy as np
df_num = df.select_dtypes(np.number)
```

Fig 2.1. 2

```
In [10]: variance = df_num.var()
         print(variance)
```

```
encounter_id                1.053503e+16
patient_nbr                 1.497408e+15
admission_type_id           2.089189e+00
discharge_disposition_id    2.788015e+01
admission_source_id         1.651675e+01
time_in_hospital            8.910868e+00
num_lab_procedures          3.870805e+02
num_procedures              2.909777e+00
num_medications             6.605733e+01
number_outpatient           1.605961e+00
number_emergency            8.657786e-01
number_inpatient            1.594824e+00
number_diagnoses            3.738810e+00
dtype: float64
```

```
In [11]: df2 = ['examide','citoglipton','repaglinide','nateglinide','chlorpropamide','glimepiride','acetohexamide','tolbutamide','acarbose
               'troglitazone','tolazamide','examide','citoglipton','glyburide-metformin','glipizide-metformin','glimepiride-pioglitazone
               'metformin-rosiglitazone','metformin-pioglitazone']
```

```
In [12]: def check_zero_variance(df, column_name):
             unique_values = df[column_name].nunique()
             if unique_values <= 2:
                 return column_name

         # List to store zero variance columns
         zero_variance_columns = []

         # Check for each categorical column
         for column in df2:
             result = check_zero_variance(df, column)
             if result:
                 zero_variance_columns.append(result)

         print("Zero Variance Columns:")
         print(zero_variance_columns)
```

```
Zero Variance Columns:
['examide', 'citoglipton', 'acetohexamide', 'tolbutamide', 'troglitazone', 'examide', 'citoglipton', 'glipizide-metformin', 'gl
imepiride-pioglitazone', 'metformin-rosiglitazone', 'metformin-pioglitazone']
```

```
In [13]: df.drop(columns=zero_variance_columns, inplace=True)
```

```
In [14]: df.shape
```

```
Out[14]: (101766, 40)
```

Fig 2.1. 3

```
In [15]: missing_percentage = (df.isnull().sum() / len(df)) * 100
         print(missing_percentage)
```

```
encounter_id                0.000000
patient_nbr                 0.000000
race                        2.233555
gender                      0.000000
age                         0.000000
admission_type_id           0.000000
discharge_disposition_id    0.000000
admission_source_id         0.000000
time_in_hospital            0.000000
payer_code                 39.557416
medical_specialty          49.082208
num_lab_procedures          0.000000
num_procedures              0.000000
num_medications             0.000000
number_outpatient           0.000000
number_emergency            0.000000
number_inpatient            0.000000
diag_1                      0.020636
diag_2                      0.351787
diag_3                      1.398306
number_diagnoses            0.000000
max_glu_serum               0.000000
A1Cresult                   0.000000
metformin                   0.000000
repaglinide                 0.000000
nateglinide                 0.000000
chlorpropamide              0.000000
glimepiride                 0.000000
glipizide                   0.000000
glyburide                   0.000000
pioglitazone                0.000000
rosiglitazone               0.000000
acarbose                    0.000000
miglitol                    0.000000
tolazamide                  0.000000
insulin                     0.000000
glyburide-metformin         0.000000
change                      0.000000
diabetesMed                 0.000000
readmitted                  0.000000
dtype: float64
```

```
In [16]: df['race'].ffill(inplace=True)
         df['payer_code'].ffill(inplace=True)
         df['medical_specialty'].ffill(inplace=True)
         df['diag_3'].ffill(inplace=True)
         df['diag_1'].ffill(inplace=True)
         df['diag_2'].ffill(inplace=True)
```

Fig 2.1. 4

```
In [17]: missing_percentage = (df.isnull().sum() / len(df)) * 100
         print(missing_percentage)
```

```
encounter_id                0.000000
patient_nbr                 0.000000
race                        0.000000
gender                      0.000000
age                         0.000000
admission_type_id           0.000000
discharge_disposition_id    0.000000
admission_source_id         0.000000
time_in_hospital            0.000000
payer_code                 20.091190
medical_specialty           0.000000
num_lab_procedures          0.000000
num_procedures              0.000000
num_medications             0.000000
number_outpatient           0.000000
number_emergency            0.000000
number_inpatient            0.000000
diag_1                      0.000000
diag_2                      0.000983
diag_3                      0.000983
number_diagnoses            0.000000
max_glu_serum               0.000000
A1Cresult                   0.000000
metformin                   0.000000
repaglinide                 0.000000
nateglinide                 0.000000
chlorpropamide              0.000000
glimepiride                 0.000000
glipizide                   0.000000
glyburide                   0.000000
pioglitazone                0.000000
rosiglitazone               0.000000
acarbose                    0.000000
miglitol                    0.000000
tolazamide                  0.000000
insulin                     0.000000
glyburide-metformin         0.000000
change                      0.000000
diabetesMed                 0.000000
readmitted                  0.000000
dtype: float64
```

```
In [18]: df.dropna(inplace=True)
```

```
In [19]: df.shape
```

```
Out[19]: (81320, 40)
```

```
In [20]: # Replacing '<30' with a 1 and '>30' and 'NO' to 0 respectively
         df["readmitted"] = df["readmitted"].replace({'<30': 1, '>30': 0, 'NO': 0})
```

Fig 2.1. 5

```
In [22]: from sklearn.preprocessing import LabelEncoder
         # Encoding categorical variables
         categorical_cols = df.select_dtypes(include='object').columns.tolist()
         label_encoders = {}
         for col in categorical_cols:
             label_encoders[col] = LabelEncoder()
             df[col] = label_encoders[col].fit_transform(df[col])
```

```
In [23]: # Splitting the dataset into features and target variable
         X = df.drop(columns=['readmitted'])
         y = df['readmitted']
```

Fig 2.1. 6

In [24]:
```
from sklearn.model_selection import train_test_split

# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fig 2.1. 7

In [25]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

# Pipeline for preprocessing and modeling
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)])
```

In [26]:
```
# Append classifier to preprocessing pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(max_iter=1000))])
```

In [27]:
```
# Train the model
clf.fit(X_train, y_train)
```

Out[27]:

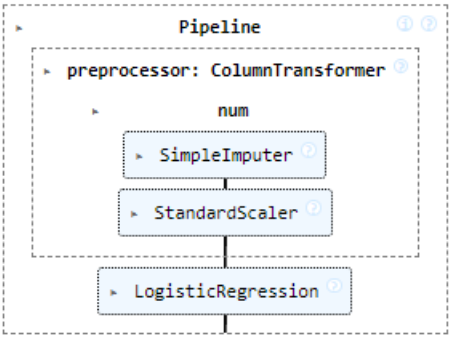


Fig 2.1. 8

In [28]:
```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, f1_score, recall_score, precision_score
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming clf is your classifier model trained earlier

y_pred = clf.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)
clf_report = classification_report(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
roc_auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1], average='weighted', multi_class='ovr')

print('precision:')
print(precision)
print("\n")
print('recall:')
print(recall)
print("\n")
print('f1')
print(f1)
print("\n")
print('roc_auc')
print(roc_auc)
print("\n")
print("Confusion Matrix:")
print(conf_matrix)
print("\n")
print("Classification Report:")
print(clf_report)


# Plotting the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

Fig 2.1. 9

```
precision:
0.8406154354391767
recall:
0.8896950319724545
f1
0.8416778245748018
roc_auc
0.6490835511483289
Confusion Matrix:
[[14441    39]
 [ 1755    29]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     14480
           1       0.43      0.02      0.03      1784

    accuracy                           0.89     16264
   macro avg       0.66      0.51      0.49     16264
weighted avg       0.84      0.89      0.84     16264
```

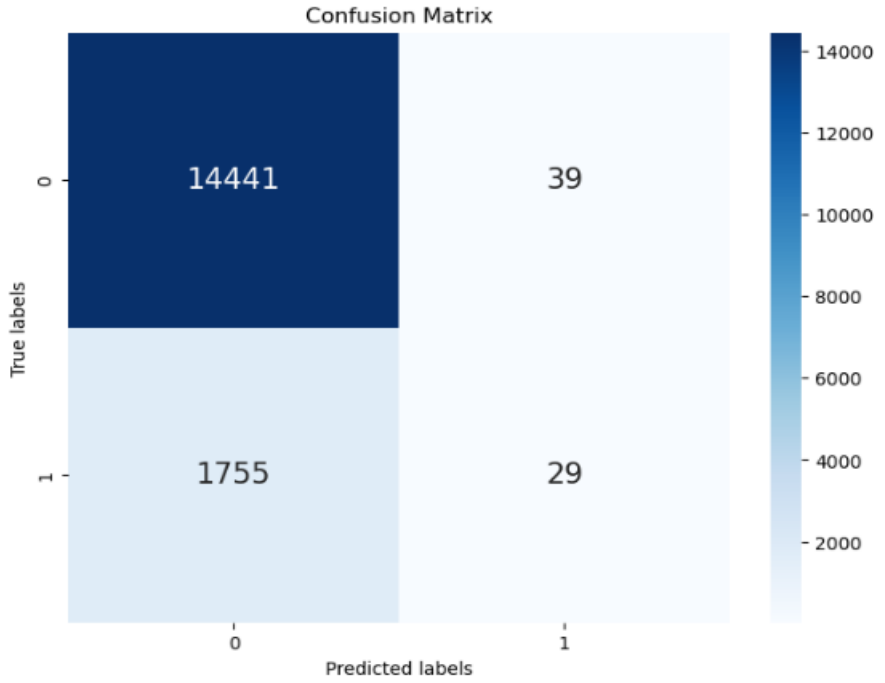


Fig 2.1. 10

To address the challenge of creating an improved classification model that maximises data point utilisation and handles missing values differently (e.g., by imputation) in order to predict patient readmission.

let's break down the steps used in the provided code snippet:

1) Data Loading and Initial Inspection:
   - The code begins by loading the dataset diabetic_data.csv into a pandas DataFrame (df).
   - The version of pandas library used here is latest other than which we have used in part 1.

2) Handling Missing Values:
   - The code replaces '?' with null values (pd.NA) in the DataFrame using df.replace().
   - Jupyter Notebook with pandas version 2.2.1 was considering 'None' values in columns 'max_glu_serum' and 'A1Cresult' as null values so they were handled by stripping spaces and replacing null values with the string 'None'. (Ref. **Fig 2.1. 1**)
   - Missing values are then checked and printed as a percentage using df.isnull().sum() and columns with > 90% missing values are dropped. (Ref. Fig 2.1. 2)

3) Handling Near-Zero Variance Columns:
   - The variance of numerical columns (df_num.var()) is calculated to identify near-zero variance columns.
   - As there are no near-zero variance numerical columns we have identified categorial columns only with two or less unique values such as 'examide', 'citoglipton', etc., which we identified as near-zero variance columns and dropped them from the DataFrame using df.drop(). (Ref. **Fig 2.1. 3**)

4) Handling Remaining Missing Values:
   - Missing values in categorical columns ('race', 'payer_code', 'medical_specialty', 'diag_1', 'diag_2', 'diag_3') are filled using forward fill (ffill) to propagate the last valid observation forward. (Ref. **Fig 2.1. 4**)
   - Remaining missing values are dropped using df.dropna().
   - Values in ‘readmitted’ column are replaced ‘<30’ with a 1 and ‘>30’ and ‘NO’ to 0 respectively. (Ref. **Fig 2.1. 5**)

5) Encoding Categorical Variables:
   - Categorical variables are encoded using LabelEncoder to convert them into numerical format suitable for training our models.
   - Then we split the data into predictors and target variables. (Ref. **Fig 2.1. 6**)

6) Train-Test Split:
   - The dataset is split into features (X) and target variable (y) using df.drop() and selecting the 'readmitted' column.
   - The data is then split into training and testing sets (80% for training, 20% for testing) using train_test_split(). (Ref. **Fig 2.1. 7**)

7) Preprocessing Pipeline:
   - A preprocessing pipeline is created using ColumnTransformer, which applies imputation (mean strategy) and scaling (using StandardScaler) to numeric features. (Ref. **Fig 2.1. 8**)

8) Model Training:
   - A logistic regression model is trained using the training data within a pipeline that includes preprocessing steps. (Ref. **Fig 2.1. 8**)

9) Model Evaluation:
   - Predictions are made on the test set using the trained model (clf.predict()).
   - Classification report and accuracy score are printed to evaluate the model's performance (classification_report, confusion_matrix, roc_auc_score, f1_score, recall_score, precision_score). (Ref. **Fig 2.1. 9**)
   - A confusion matrix is plotted to visualize the model's performance (seaborn.heatmap()). (Ref. Fig 2.1. 10)

2) *Use the K-Means algorithm to cluster your cleansed dataset and compare the obtained clusters with the distribution found in the data. Justify your clustering and visualize your clusters as appropriate.*

```
In [29]: # Using K-Means algorithm to cluster the cleansed dataset
         kmeans = KMeans(n_clusters=3, random_state=42)
         kmeans.fit(X)

Out[29]: KMeans(n_clusters=3, random_state=42)
         In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
         On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [30]: from sklearn.compose import ColumnTransformer

In [31]: # Compare the obtained clusters with the distribution found in the data
         df['cluster'] = kmeans.labels_
         cluster_counts = df['cluster'].value_counts(normalize=True).sort_index()
         print("Cluster distribution:")
         print(cluster_counts)

         Cluster distribution:
         cluster
         0    0.312236
         1    0.130866
         2    0.556899
         Name: proportion, dtype: float64
```

Fig 2.2. 1

```
In [30]: # Compare the obtained clusters with the distribution found in the data
         df['cluster'] = kmeans.labels_
         cluster_counts = df['cluster'].value_counts(normalize=True).sort_index()
         print("Cluster distribution:")
         print(cluster_counts)

         Cluster distribution:
         cluster
         0    0.312236
         1    0.130866
         2    0.556899
         Name: proportion, dtype: float64
```

Fig 2.2. 2

```
In [33]: # Visualizing cluster distribution
         plt.figure(figsize=(8, 6))
         sns.countplot(x='cluster', data=df)
         plt.title('Cluster Distribution')
         plt.xlabel('Cluster')
         plt.ylabel('Count')
         plt.show()
```

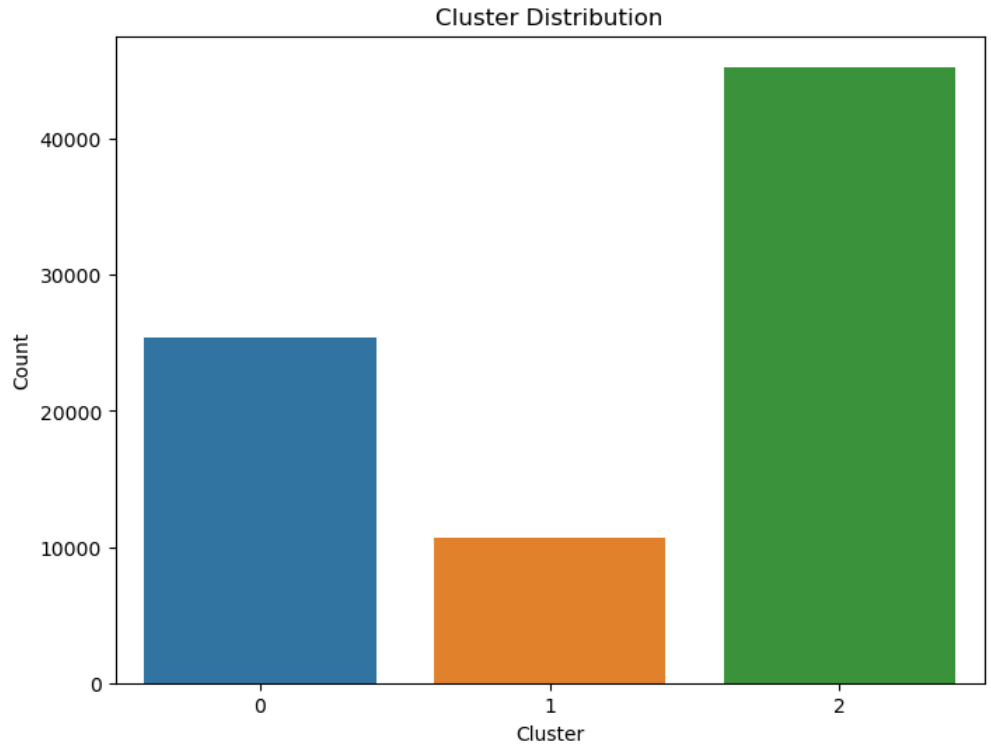


Fig 2.2. 3

Using the cleaned dataset, the algorithm simply applies K-Means clustering, analyses the cluster distribution, trains local classifiers for each cluster, and displays the cluster distribution. This method makes it possible to comprehend the traits of every cluster and how well they reflect various dataset segments. Furthermore, it permits the measurement of classification performance inside each cluster independently through the training of local classifiers.

1) K-Means Clustering:
   - kmeans = KMeans(n_clusters=3, random_state=42): This initializes the K-Means clustering algorithm with 3 clusters and sets the random state for reproducibility.
   - kmeans.fit(X): This fits the K-Means model to the feature matrix X, clustering the data points into three clusters based on their attributes. (Ref. **Fig 2.2. 1**)

2) Cluster Distribution Comparison:
   - df['cluster'] = kmeans.labels_: This assigns cluster labels obtained from K-Means clustering to the DataFrame.
   - cluster_counts = df['cluster'].value_counts(normalize=True).sort_index(): This calculates the distribution of clusters in the data. The value_counts() method counts the occurrences of each cluster label, and normalize=True normalizes the counts to proportions.
   - Printing and analyzing cluster_counts provide insights into how data points are distributed among the clusters. (Ref. Fig 2.2. 2)

3) Visualizing Cluster Distribution:
   - sns.countplot(x='cluster', data=df): This creates a count plot to visualize the distribution of data points among the clusters.

- The count plot helps in understanding how well the clusters represent the overall data distribution. (Ref. Fig 2.2. 3)

3) *Build up local classifiers based on your clustering and discuss how this clusters-based classification compares to your model obtained in the first part of Improved model.*

```
In [32]: # from sklearn.model_selection import train_test_split
         # from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import confusion_matrix, classification_report, precision_score, recall_score, f1_score, roc_auc_score

         # Assuming df is your DataFrame containing the data

         # Building local classifiers based on the clustering
         for cluster_label in range(3):
             cluster_data = df[df['cluster'] == cluster_label]
             X_cluster = cluster_data.drop(columns=['readmitted', 'cluster'])
             y_cluster = cluster_data['readmitted']

             X_cluster_train, X_cluster_test, y_cluster_train, y_cluster_test = train_test_split(X_cluster, y_cluster,
                                                                                                 test_size=0.2, random_state=42)

             # Train and evaluate local classifier
             local_clf = LogisticRegression(max_iter=1000)
             local_clf.fit(X_cluster_train, y_cluster_train)
             y_cluster_pred = local_clf.predict(X_cluster_test)

             # Calculate metrics for the current cluster
             conf_matrix = confusion_matrix(y_cluster_test, y_cluster_pred)
             clf_report = classification_report(y_cluster_test, y_cluster_pred)
             precision = precision_score(y_cluster_test, y_cluster_pred, average='weighted')
             recall = recall_score(y_cluster_test, y_cluster_pred, average='weighted')
             f1 = f1_score(y_cluster_test, y_cluster_pred, average='weighted')
             roc_auc = roc_auc_score(y_cluster_test, local_clf.predict_proba(X_cluster_test)[:, 1], average='weighted', multi_class='ovr')

             print(f"Metrics for Cluster {cluster_label}:")
             print('Precision:', precision)
             print('Recall:', recall)
             print('F1-score:', f1)
             print('ROC AUC:', roc_auc)
             print("Confusion Matrix:")
             print(conf_matrix)
             print("Classification Report:")
             print(clf_report)
             print("\n")

             # Plotting the confusion matrix as a heatmap
             plt.figure(figsize=(8, 6))
             sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})
             plt.xlabel('Predicted labels')
             plt.ylabel('True labels')
             plt.title('Confusion Matrix')
             plt.show()
```

Fig 2.3. 1

```
Metrics for Cluster 0:
Precision: 0.7933937351569944
Recall: 0.8907265209686946
F1-score: 0.8392474811751275
ROC AUC: 0.4987406504647884
Confusion Matrix:
[[4524    0]
 [ 555    0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      4524
           1       0.00      0.00      0.00       555

    accuracy                           0.89      5079
   macro avg       0.45      0.50      0.47      5079
weighted avg       0.79      0.89      0.84      5079
```

Fig 2.3. 2

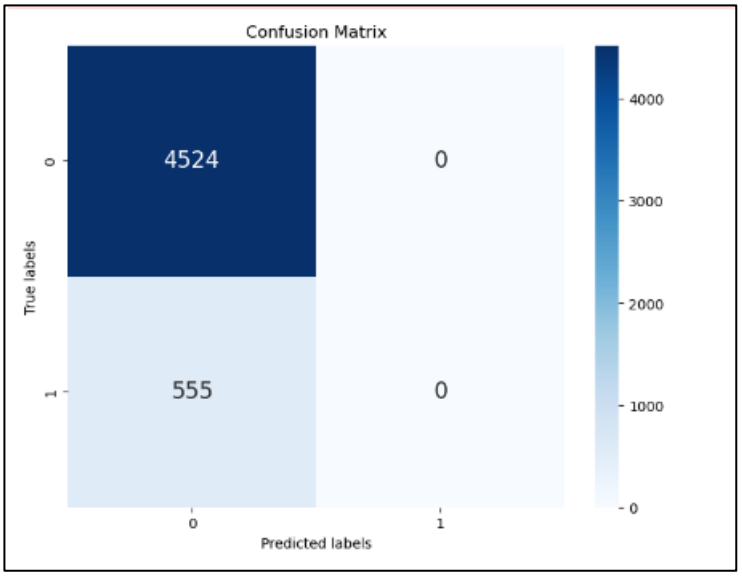


Fig 2.3. 3

```
Metrics for Cluster 1:
Precision: 0.7981227721321852
Recall: 0.8933771723813997
F1-score: 0.8430679145965876
ROC AUC: 0.5147584040912185
Confusion Matrix:
[[1902    0]
 [ 227    0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      1902
           1       0.00      0.00      0.00       227

    accuracy                           0.89      2129
   macro avg       0.45      0.50      0.47      2129
weighted avg       0.80      0.89      0.84      2129
```

Fig 2.3. 4

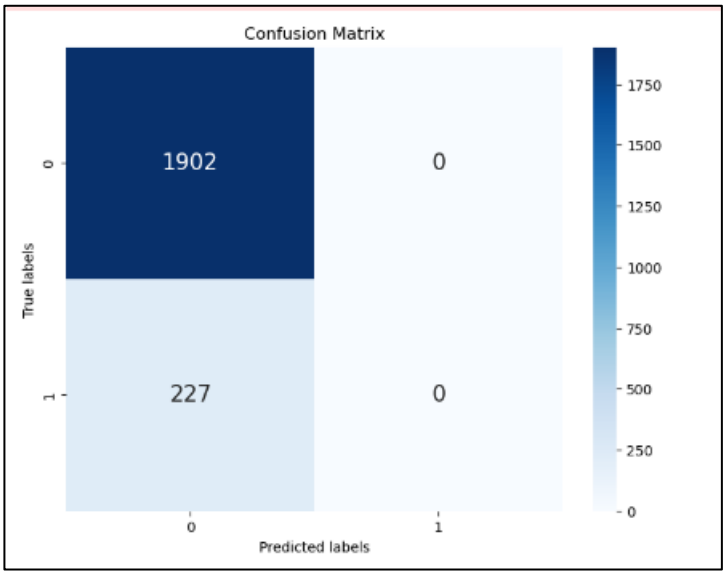


Fig 2.3. 5

```
Metrics for Cluster 2:
Precision: 0.7796472778820781
Recall: 0.8829763744756017
F1-score: 0.8281009665872259
ROC AUC: 0.49915863399812216
Confusion Matrix:
[[7998    0]
 [1060    0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      1.00      0.94      7998
           1       0.00      0.00      0.00      1060

    accuracy                           0.88      9058
   macro avg       0.44      0.50      0.47      9058
weighted avg       0.78      0.88      0.83      9058
```

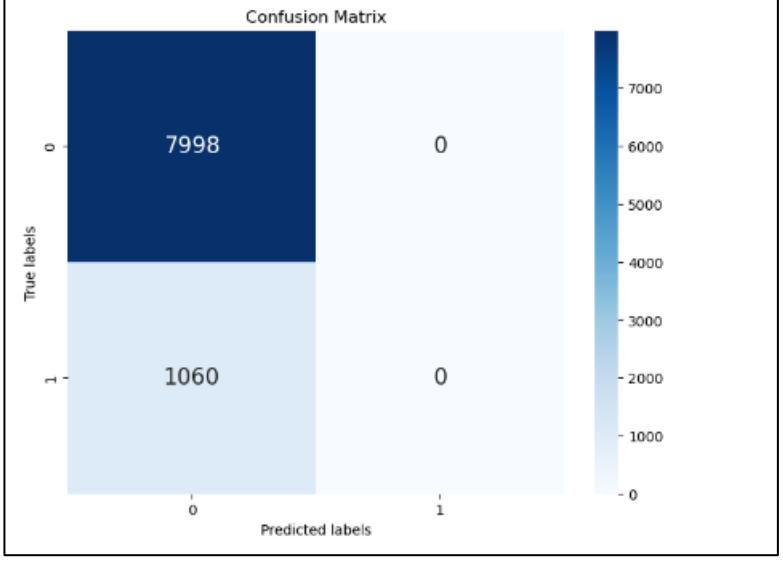


Fig 2.3. 6

Local Classifier Training:

A loop iterates over each cluster:

- for cluster_label in range(3):: This loop iterates over each cluster label.
- cluster_data = df[df['cluster'] == cluster_label]: This selects data points belonging to the current cluster.
- X_cluster and y_cluster are defined as features and target variable for the current cluster, respectively.
- train_test_split() splits the cluster data into training and testing sets.
- LogisticRegression(max_iter=1000) initializes a logistic regression classifier for the current cluster.
- local_clf.fit(X_cluster_train, y_cluster_train) trains the logistic regression classifier on the training data.
- local_clf.predict(X_cluster_test) predicts the target variable using the trained classifier on the test data. (Ref. Fig 2.3. 1)
- Performance metrics such as precision, recall, F1-score, ROC AUC, confusion matrix, and classification report are calculated for each cluster. (Ref. Fig 2.3. 2 & Fig 2.3. 4 & Fig 2.3. 6)
- A confusion matrix is plotted to visualize the model's performance (seaborn.heatmap()). (Ref. Fig 2.3. 3 & Fig 2.3. 5 & Fig 2.3. 6)

4) *As in Part1, balance the data and train and test your model with the balanced data.*

UnderSampling

In [37]:
```
from imblearn.under_sampling import RandomUnderSampler
# Create the RandomUnderSampler object
undersampler = RandomUnderSampler(random_state=42)

# Resample the data
X_undersampled, y_undersampled = undersampler.fit_resample(X, y)

# Check the new class distribution
print("Undersampled Data Distribution:")
print(pd.Series(y_undersampled).value_counts())
```

```
Undersampled Data Distribution:
readmitted
0    9012
1    9012
Name: count, dtype: int64
```

In [38]:
```
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_undersampled, y_undersampled, test_size=0.2, random_state=42)
```

In [39]:
```
from sklearn.compose import ColumnTransformer

# Pipeline for preprocessing and modeling
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)])
```

Fig 2.4. 1

In [40]:
```
# Append classifier to preprocessing pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(max_iter=1000))])
```

In [41]:
```
# Train the model
clf.fit(X_train, y_train)
```

Out[41]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImputer()),
                                                                  ('scaler',
                                                                   StandardScaler())]),
                                                  Index(['encounter_id', 'patient_nbr', 'admission_type_id',
       'discharge_disposition_id', 'admission_source_id', 'time_in_hospital',
       'num_lab_procedures', 'num_procedures', 'num_medications',
       'number_outpatient', 'number_emergency', 'number_inpatient',
       'number_diagnoses'],
      dtype='object'))])),
                ('classifier', LogisticRegression(max_iter=1000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Fig 2.4. 2

In [42]:
```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, f1_score, recall_score, precision_score

# Assuming clf is your classifier model trained earlier

y_pred = clf.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)
clf_report = classification_report(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
roc_auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1], average='weighted', multi_class='ovr')

print('precision:')
print(precision)
print("\n")
print('recall:')
print(recall)
print("\n")
print('f1')
print(f1)
print("\n")
print('roc_auc')
print(roc_auc)
print("\n")
print("Confusion Matrix:")
print(conf_matrix)
print("\n")
print("Classification Report:")
print(clf_report)
```

```
precision:
0.6041501083892604


recall:
0.5983356449375867


f1
0.5917884642973714


roc_auc
0.6468846648215271


Confusion Matrix:
[[1312  501]
 [ 947  845]]


Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.72      0.64      1813
           1       0.63      0.47      0.54      1792

    accuracy                           0.60      3605
   macro avg       0.60      0.60      0.59      3605
weighted avg       0.60      0.60      0.59      3605
```

Fig 2.4. 3

OverSampling

In [43]:
```
from imblearn.over_sampling import RandomOverSampler
# Create the RandomOverSampler object
oversampler = RandomOverSampler(random_state=42)

# Resample the data
X_oversampled, y_oversampled = oversampler.fit_resample(X, y)

# Check the new class distribution
print("Oversampled Data Distribution:")
print(pd.Series(y_oversampled).value_counts())
```

```
Oversampled Data Distribution:
readmitted
0    72308
1    72308
Name: count, dtype: int64
```

In [44]:
```
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_oversampled, y_oversampled, test_size=0.2, random_state=42)
```

In [45]:
```
from sklearn.compose import ColumnTransformer

# Pipeline for preprocessing and modeling
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)])
```

In [46]:
```
# Append classifier to preprocessing pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(max_iter=1000))])
```

Fig 2.4. 4

In [47]:
```
# Train the model
clf.fit(X_train, y_train)
```

Out[47]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImputer()),
                                                                  ('scaler',
                                                                   StandardScaler())]),
                                                  Index(['encounter_id', 'patient_nbr', 'admission_type_id',
       'discharge_disposition_id', 'admission_source_id', 'time_in_hospital',
       'num_lab_procedures', 'num_procedures', 'num_medications',
       'number_outpatient', 'number_emergency', 'number_inpatient',
       'number_diagnoses'],
      dtype='object'))])),
                ('classifier', LogisticRegression(max_iter=1000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Fig 2.4. 5

```
In [48]: from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, f1_score, recall_score, precision_score

         # Assuming clf is your classifier model trained earlier

         y_pred = clf.predict(X_test)

         conf_matrix = confusion_matrix(y_test, y_pred)
         clf_report = classification_report(y_test, y_pred)
         precision = precision_score(y_test, y_pred, average='weighted')
         recall = recall_score(y_test, y_pred, average='weighted')
         f1 = f1_score(y_test, y_pred, average='weighted')
         roc_auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1], average='weighted', multi_class='ovr')

         print('precision:')
         print(precision)
         print("\n")
         print('recall:')
         print(recall)
         print("\n")
         print('f1')
         print(f1)
         print("\n")
         print('roc_auc')
         print(roc_auc)
         print("\n")
         print("Confusion Matrix:")
         print(conf_matrix)
         print("\n")
         print("Classification Report:")
         print(clf_report)

         precision:
         0.6029428408335409


         recall:
         0.5977389019499377


         f1
         0.5922750982296896


         roc_auc
         0.6436840920886038


         Confusion Matrix:
         [[10335  4160]
          [ 7475  6954]]


         Classification Report:
                       precision    recall  f1-score   support

                    0       0.58      0.71      0.64     14495
                    1       0.63      0.48      0.54     14429

             accuracy                           0.60     28924
            macro avg       0.60      0.60      0.59     28924
         weighted avg       0.60      0.60      0.59     28924
```

Fig 2.4. 6

Overall, this code illustrates how to balance data using approaches such as undersampling and oversampling, and how to assess the effectiveness of the trained model on the balanced datasets.

Let's break down the provided code and explain each part:

1) Undersampling Technique:

- RandomUnderSampler: This method randomly selects samples from the majority class (the class with more instances) to balance the class distribution.
- undersampler = RandomUnderSampler(random_state=42): Initializes the RandomUnderSampler object with a specified random state.
- X_undersampled, y_undersampled = undersampler.fit_resample(X, y): Resamples the data by reducing the number of instances in the majority class to match the minority class.
- Prints the distribution of classes in the undersampled dataset.

2) Oversampling Technique:

- RandomOverSampler: This method randomly replicates samples from the minority class to balance the class distribution.
- oversampler = RandomOverSampler(random_state=42): Initializes the RandomOverSampler object with a specified random state.

- X_oversampled, y_oversampled = oversampler.fit_resample(X, y): Resamples the data by increasing the number of instances in the minority class to match the majority class.
- Prints the distribution of classes in the oversampled dataset.

3) Model Training and Evaluation (for both undersampled and oversampled data):
- Pipeline Construction: A pipeline is constructed to preprocess the data and train a logistic regression classifier. And preprocessing includes imputation of missing values and scaling of numerical features.
- Model Training: The pipeline is trained on the resampled training data (X_train, y_train).
- Model Evaluation: Predictions are made on the test data (X_test) using the trained model. Performance metrics such as precision, recall, F1-score, ROC AUC, confusion matrix, and classification report are computed and printed. These metrics help assess the effectiveness of the logistic regression model trained on balanced data.

Overall explanation:
- Techniques such as undersampling and oversampling are used to adjust the dataset's class distribution.
- To compare performance, the logistic regression model is trained independently on both the oversampled and undersampled datasets.
- The model can learn from the unbalanced data more effectively by employing these resampling strategies, which may enhance its capacity to identify occurrences of the minority class.
- Evaluation of the model's generalisation and predictive power on balanced data is possible through performance indicators.

CHAPTER 3
CONCLUSION

This project successfully employed big data analytics to develop a predictive model for 30-day hospital readmission among diabetes patients, leveraging a comprehensive dataset from 130 U.S. hospitals. Through meticulous data preprocessing and exploratory analysis, significant predictors influencing readmission rates were identified and analyzed. The adoption of the Pandas library facilitated efficient data manipulation, proving indispensable for handling and transforming the large dataset to suit the modeling needs.

The selection of the Interquartile Range (IQR) method for outlier removal was justified by its robustness in managing skewed distributions and extreme data values, thus enhancing the model's accuracy and reliability. The thoughtful choice of data columns for graphical analysis, based on their impact on readmission rates, and the preference for IQR over the z-score method for outlier detection, were underpinned by the data's well-distributed nature, as evidenced through detailed boxplot visualizations.

This investigation not only underscores the potential of machine learning and big data in transforming healthcare analytics but also provides a scalable and effective framework for predicting hospital readmissions. The insights gained from this study can inform healthcare strategies and interventions, aiming to reduce readmission rates and improve the quality of patient care for those with diabetes. Moving forward, the project sets a precedent for future research in healthcare analytics, advocating for a data-driven approach in understanding and addressing the complexities of patient re admission.