Final Report of Traineeship Program 2024

On

“Predict Blood Donations”

MEDTOUREASY

28th NOVEMBER 2024

ACKNOWLDEGMENTS

The traineeship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Data Visualizations in Data Analytics; and also, for personal as well as professional development. I am very obliged for having a chance to interact with so many professionals who guided me throughout the traineeship project and made it a great learning curve for me.

Firstly, I express my deepest gratitude and special thanks to the Training & Developement Team of MedTourEasy who gave me an opportunity to carry out my traineeship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Data Analytics profile and training me in the same so that I can carry out the project properly and with maximum client satisfaction and also for spearing his valuable time in spite of his busy schedule.

I would also like to thank the team of MedTourEasy and my colleagues who made the working environment productive and very conducive.

TABLE OF CONTENTS

ABSTRACT

Blood donation is a vital component of healthcare systems, with a consistent supply of blood crucial for numerous medical procedures. However, challenges such as donor motivation, scheduling conflicts, and personal circumstances can significantly impact donation rates. To address these challenges, predictive modeling can be employed to anticipate future donor behavior.

This study leverages a dataset collected from a mobile blood donation vehicle in Taiwan to predict the likelihood of a donor's future participation. By analyzing historical donation patterns, demographic information, and other relevant factors, we aim to develop a robust predictive model. This research has the potential to optimize blood donation campaigns, improve resource allocation, and ultimately enhance blood supply management.

Other Information:

Global Blood Supply: According to the World Health Organization (WHO), approximately 118.54 million blood donations are collected worldwide annually. However, there's a significant disparity in blood availability between high-income and low-income countries.

Donor Motivation: Understanding the factors influencing donor motivation is crucial. Factors such as altruism, social norms, and personal satisfaction play significant roles in determining donation behavior.

Impact of COVID-19: The COVID-19 pandemic has had a substantial impact on blood donation drives. Lockdowns, social distancing measures, and health concerns have led to decreased blood donations in many regions.

Predictive Modeling Applications: Predictive modeling techniques, such as machine learning and statistical modeling, can be used to identify potential donors, optimize recruitment strategies, and improve blood supply forecasting.

1.1 About the Company

MedTourEasy, a global healthcare company, provides you the informational resources needed to evaluate your global options. MedTourEasy provides analytical solutions to our partner healthcare providers globally.

1.2 About the Project

Understanding the Challenge

Ensuring a consistent blood supply is a critical challenge faced by healthcare systems worldwide. Blood shortages can have severe consequences for patients in need of transfusions. To address this issue, accurate prediction of future blood donations is essential.

Project Objective

The primary goal of this project is to develop a robust machine learning model capable of predicting whether a blood donor will donate within a specified timeframe. By analyzing historical donation data, demographic information, and other relevant factors, we aim to improve the efficiency and effectiveness of blood donation campaigns.

Dataset and Methodology

The dataset, sourced from the Machine Learning Repository, contains information on 748 blood donors. Key features include demographic data, donation history, and other relevant attributes.

The project will involve the following steps:

1. **Data Exploration and Cleaning:** A thorough analysis of the dataset to identify missing values, outliers, and inconsistencies.
2. **Feature Engineering:** Creating new features or transforming existing ones to improve model performance.
3. **Model Selection and Training:** Employing machine learning algorithms, such as logistic regression, to build and train predictive models.

4. **Model Evaluation:** Assessing the performance of the models using appropriate metrics like accuracy, precision, recall, and F1-score.
5. **Hyperparameter Tuning:** Optimizing model performance through fine-tuning hyperparameters.
6. **Model Deployment:** Preparing the model for real-world applications, such as integration into a blood donation management system.

This project on forecasting blood supply is divided into three major sections to ensure a structured and comprehensive analysis. The main focus is to understand donor behavior and develop a predictive model to address the challenges faced by blood collection managers. The sections are as follows:

- ***Analysis of the donor data*:** This involves a detailed examination of the dataset provided by the Blood Transfusion Service Center. Key statistics, such as donation frequency, recency of last donation, and donor tenure, are analyzed to understand patterns in donor behavior. This step aims to uncover critical factors influencing a donor's likelihood of making a future donation, which will serve as inputs for model development.
- ***Building and evaluating predictive models*:** Using the cleaned and preprocessed data, machine learning techniques are applied to predict whether a donor will donate within a given time window. Logistic regression is used as the primary modeling technique, while the tpot library is leveraged to automate the pipeline and identify the most optimal model configuration. This section focuses on model accuracy, interpretability, and evaluation metrics to assess the performance of the predictive system.
- ***Insights and impact assessment*:** Based on the predictions, actionable insights are generated to assist blood collection managers in making data-driven decisions. This includes understanding donor engagement strategies and estimating future blood supply needs. The insights can help mitigate shortages, reduce wastage, and improve overall resource allocation in blood banks.

The methodologies in this project, supported by Python programming and advanced libraries like pandas and tpot, aim to provide an efficient, automated solution to a pressing issue in healthcare logistics. MedTourEasy's support has been instrumental in driving this project forward, demonstrating their commitment to leveraging predictive analytics to enhance healthcare resource management.

1.3 Objectives and Deliverables

Objectives

This project aims to forecast blood donations by analyzing a dataset from the Blood Transfusion Service Center. The primary objective is to develop a machine learning model capable of predicting whether a donor will donate within a given time window. Key goals include:

- Gaining an in-depth understanding of donor behavior by analyzing donation trends and key influencing factors.
- Building and optimizing a predictive model using logistic regression and the tpot library to automate the machine learning pipeline for efficient model selection and hyperparameter tuning.
- Generating actionable insights to help blood donation organizations improve resource planning, reduce wastage, and maintain a steady blood supply to meet healthcare demands.

Deliverables

The project is divided into three comprehensive deliverables:

1. **Donor Data Analysis:**
    - Perform exploratory data analysis (EDA) to examine the dataset structure, assess data quality, and detect missing or inconsistent values.
    - Identify and analyze key attributes such as:
        - *Recency:* The number of months since the donor's last donation.
        - *Frequency*: The total number of donations made by the donor.
        - *Monetary Contribution*: The total volume of blood donated.
        - *Tenure*: The duration since the first donation.
    - Use descriptive statistics and other techniques to identify trends and correlations between these variables and donation behavior.
    - Summarize findings that will guide feature selection and preprocessing for the predictive model.
2. **Predictive Modeling:**
    - Preprocess the data by normalizing high-variance features, encoding categorical variables, and splitting the dataset into training and testing subsets.
    - Implement logistic regression as the primary predictive model, leveraging its interpretability and suitability for binary classification tasks.

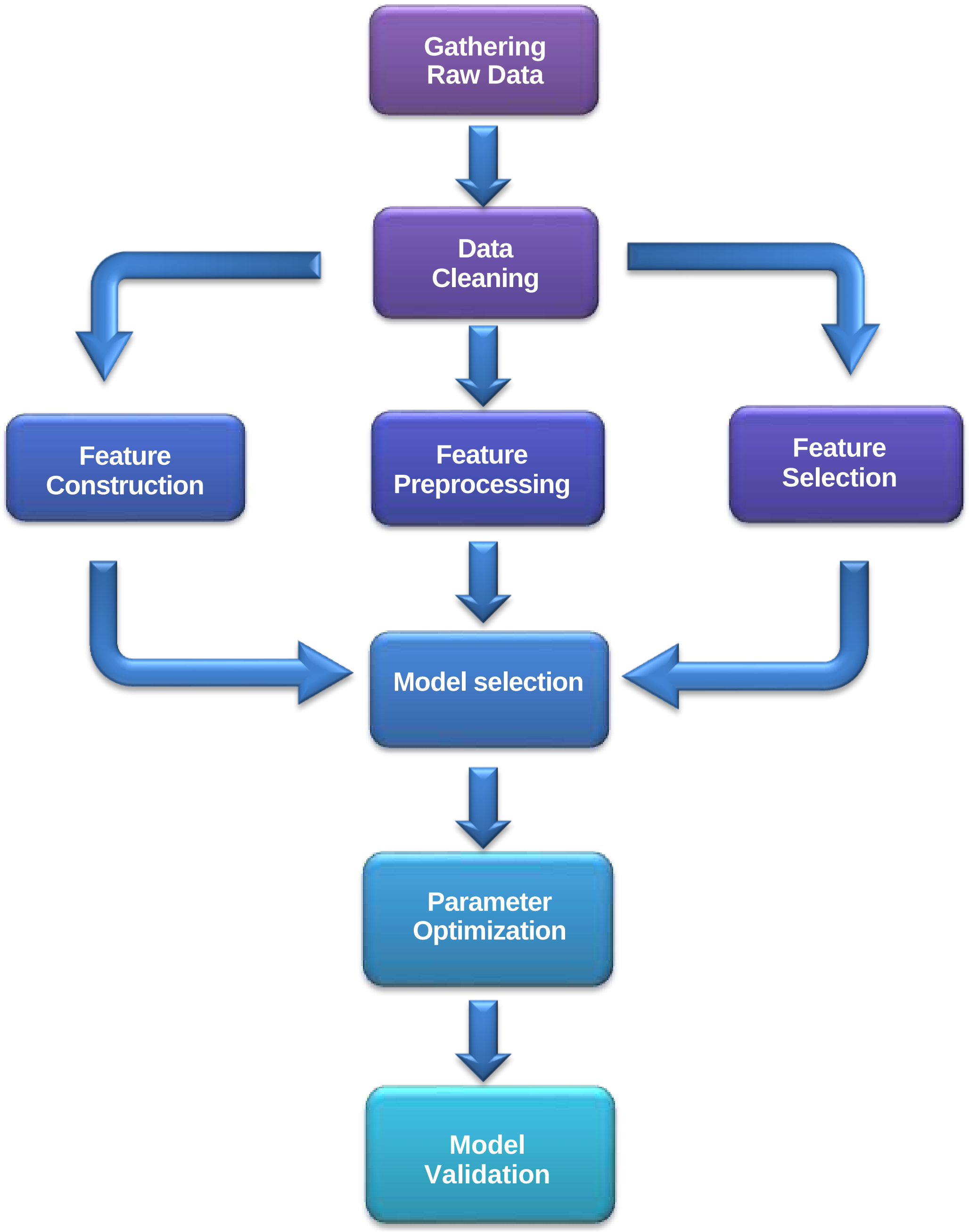
- Use the tpot library to automate and optimize the machine learning pipeline, including feature engineering, model selection, and hyperparameter tuning.
- Evaluate the model's performance using metrics such as accuracy, precision, recall, F1-score, and area under the curve (AUC).
- Compare the performance of multiple pipelines generated by tpot to select the most effective one.

3. **Insights and Recommendations:**
    - Analyze the model's predictions to identify donors most likely to donate within a specified time window.
    - Provide data-driven recommendations to blood collection managers on:
        - Strategies to retain high-frequency donors.
        - Timing and targeting of donation campaigns to maximize turnout.
        - Estimating future blood supply needs to avoid shortages or overstocking.
    - Highlight potential areas for improvement in donor engagement and retention programs based on predictive insights.
    - Deliver a comprehensive report summarizing findings, methodologies, and actionable strategies for implementation by blood donation organizations.

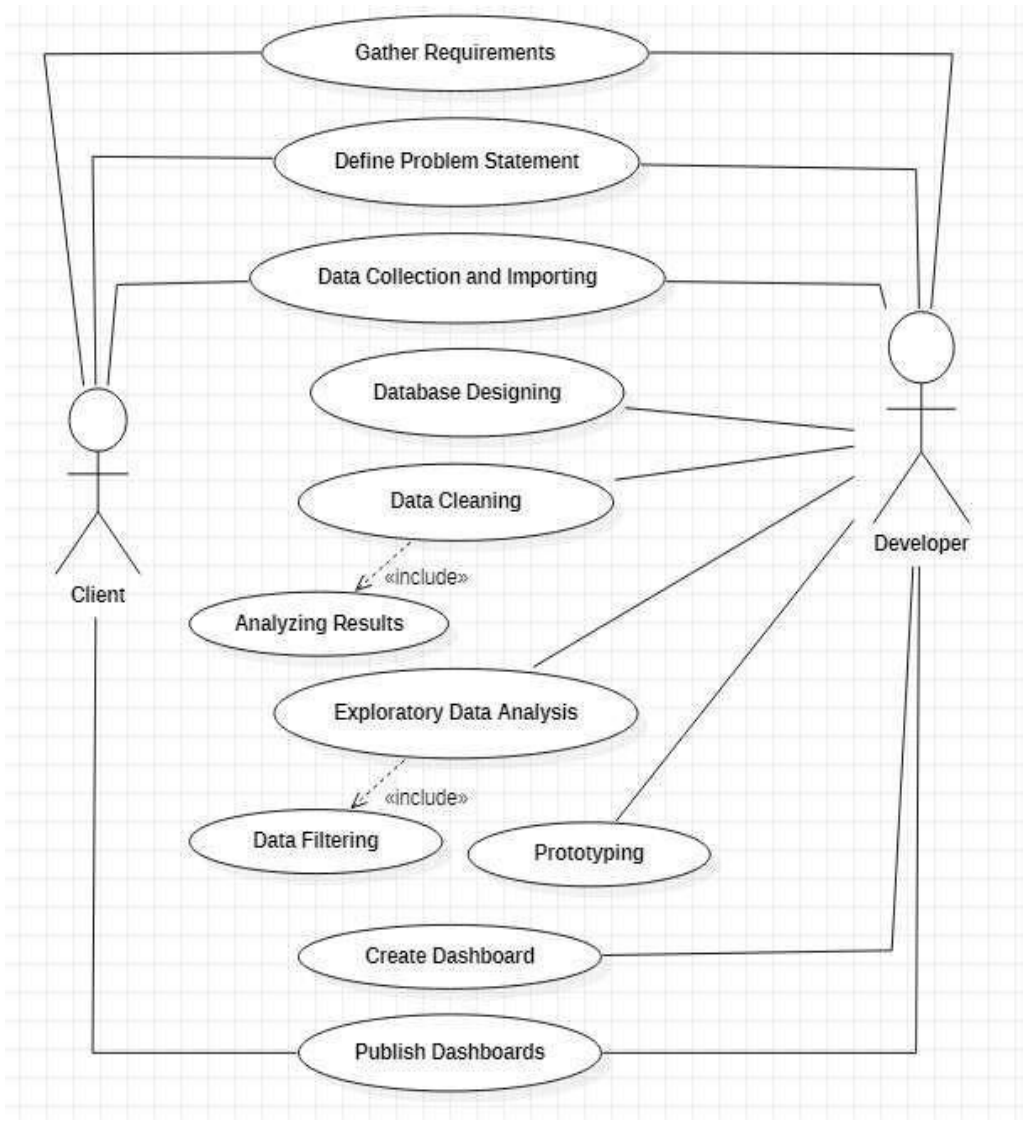
I. METHODOLOGY

2.1 Flow of the Project

The project followed the following steps to accomplish the desired objectives and deliverables. Each step has been explained in detail in the following section.



2.2 Use Case Diagram



Above figure shows the use case of this project involves the client and the developer, where the developer gathers requirements, defines the problem, and preprocesses the blood donation dataset to address issues like missing values and inconsistencies. key donor behaviors such as frequency and recency are analyzed through exploratory data analysis. the cleaned data is then used to build and optimize predictive models using logistic regression and the tpot library. finally, the developer delivers actionable insights to the client, enabling data-driven strategies for improving blood donation management and forecasting supply.

2.3 Language and Platform Used

2.3.1 Language: Python

Python is a versatile, high-level programming language widely used for data analysis, machine learning, and predictive modeling. Its simplicity and rich ecosystem of libraries make it a preferred choice for projects involving data-driven tasks like blood donation forecasting. Python's key features include:

- *Ease of Use*: Readable syntax and extensive documentation.
- *Data Handling*: Libraries such as pandas and NumPy enable efficient data processing and storage.
- *Machine Learning*: Supports libraries like scikit-learn and tpot for model building and optimization.
- *Visualization*: Integration with Matplotlib and Seaborn for creating insightful visualizations.
- *Community Support*: A large user base and active community provide a wealth of resources and solutions.

2.3.2 IDE: Jupyter Notebook

Jupyter Notebook is an open-source IDE for interactive computing. It allows combining code, visualizations, and explanatory text in a single document, making it ideal for data analysis and machine learning projects. Major features include:

- *Interactive Coding*: Supports Python and integrates well with libraries used for data science.
- *Visualization*: Facilitates inline display of charts and graphs.
- *Documentation*: Enables embedding markdown text for explanation and reporting.
- *Ease of Sharing*: Outputs can be saved and shared as HTML or PDF.

2.3.3 Libraries Used

- **Pandas** is a versatile library for data manipulation and analysis. It simplifies handling structured data with tools like DataFrames, enabling tasks such as data cleaning, exploration, and aggregation. It was used to load the dataset, inspect its structure, and perform operations like renaming columns and checking variance.
- **NumPy** is a core library for numerical computing in Python. It provides support for multi-dimensional arrays and essential mathematical operations, making it a foundation for data manipulation. In this project, it complemented pandas and scikit-learn by enabling efficient numerical calculations.
- **scikit-learn** is a widely-used library for machine learning, offering tools for model building, evaluation, and data preprocessing. It was used for splitting data into training and testing sets, applying logistic regression, and evaluating performance using metrics

like ROC-AUC.

- **TPOT** (Tree-Based Pipeline Optimization Tool) is an automated machine learning library that uses genetic programming to optimize machine learning pipelines. It was utilized in this project to identify the best-performing model and preprocessing steps, saving time and effort in model selection and hyperparameter tuning.
- **Matplotlib** is a powerful visualization library in Python. It was used to create plots for exploratory data analysis, enabling insights into the distribution and relationships in the dataset, such as visualizing target incidence and trends in donor behavior.
- **LogisticRegression (from scikit-learn)** is a statistical method for binary classification tasks. In this project, it was implemented using scikit-learn to predict whether a donor would donate blood within a specific time frame, serving as a baseline for evaluating model performance.
- **train_test_split (from scikit-learn)** was employed to divide the dataset into training and testing subsets. This step ensured that the model's performance was evaluated on unseen data, preventing overfitting and improving generalizability.
- **roc_auc_score (from scikit-learn)** was used to evaluate model performance, focusing on the area under the ROC curve. This metric provided insights into the balance between sensitivity and specificity of the predictions, crucial for binary classification tasks.

Intallation:

```
pip install pandas matplotlib seaborn scikit-learn tpot
```

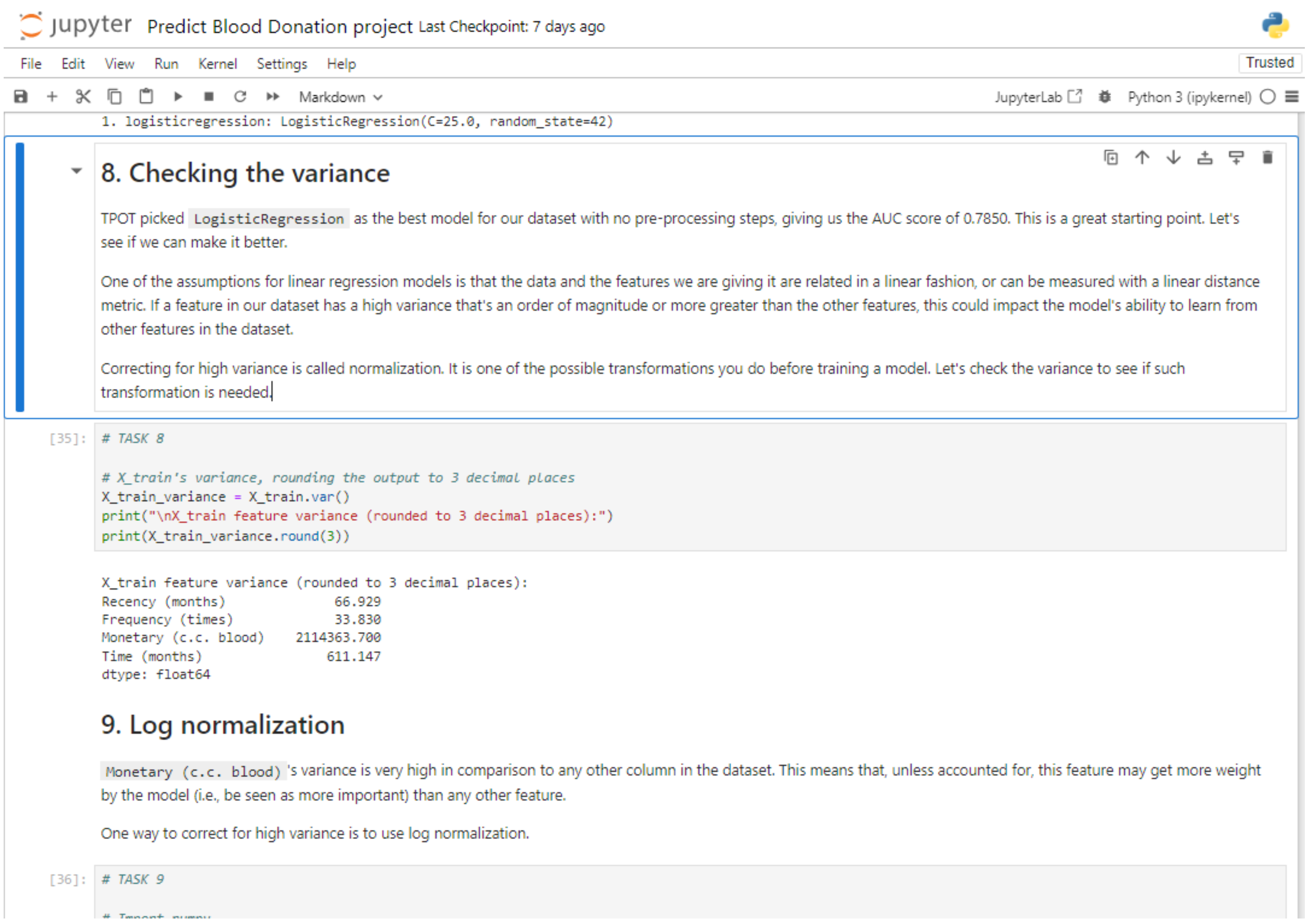
Importing:

```
# Data handling
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# Automated Machine Learning
from tpot import TPOTClassifier
```


jupyter Predict Blood Donation project Last Checkpoint: 7 days ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
1. logisticregression: LogisticRegression(C=25.0, random_state=42)
```

8. Checking the variance

TPOT picked `LogisticRegression` as the best model for our dataset with no pre-processing steps, giving us the AUC score of 0.7850. This is a great starting point. Let's see if we can make it better.

One of the assumptions for linear regression models is that the data and the features we are giving it are related in a linear fashion, or can be measured with a linear distance metric. If a feature in our dataset has a high variance that's an order of magnitude or more greater than the other features, this could impact the model's ability to learn from other features in the dataset.

Correcting for high variance is called normalization. It is one of the possible transformations you do before training a model. Let's check the variance to see if such transformation is needed.

```
[35]: # TASK 8

# X_train's variance, rounding the output to 3 decimal places
X_train_variance = X_train.var()
print("\nX_train feature variance (rounded to 3 decimal places):")
print(X_train_variance.round(3))
```

```
X_train feature variance (rounded to 3 decimal places):
Recency (months)             66.929
Frequency (times)            33.830
Monetary (c.c. blood)    2114363.700
Time (months)               611.147
dtype: float64
```

9. Log normalization

`Monetary (c.c. blood)`'s variance is very high in comparison to any other column in the dataset. This means that, unless accounted for, this feature may get more weight by the model (i.e., be seen as more important) than any other feature.

One way to correct for high variance is to use log normalization.

```
[36]: # TASK 9
```

This file contains the following parts:

- Markdown Cells: Used for descriptive headers (e.g., "8. Checking the Variance" or "9. Log Normalization") and explanations of tasks, providing context for the code.
- Code Cells: Python code for specific tasks is written and executed, such as variance checking and normalization (e.g., importing libraries, calculating variance).
- Code Output: Outputs from code execution, like variance values or log-normalized results, appear directly below the code cells.
- Formatting: Markdown formatting is applied for readability, with headers, bullet points, or code syntax highlighting.
- Interactive Execution: Cells are executed sequentially, combining code, analysis, and outputs in a single notebook interface.

II. IMPLEMENTATION & OBSERVATION

3.1 Gathering Requirements and Inspecting transfusion.data file

This is the first step wherein the requirements are collected from the clients to understand the deliverables and goals to be achieved after which a problem statement is defined which has to be adhered to while development of the project.

Blood transfusion is vital for saving lives, from surgery and injuries to treating illnesses and blood disorders. Ensuring sufficient blood supply is a critical challenge for health professionals, According to WebMD, "about 5 million Americans need a blood transfusion every year." Blood transfusion can be necessary after significant blood loss or to address conditions like anemia, blood disorders, or cancer treatments.

Why Blood Transfusions Are Necessary

- Replacing Blood Loss: To compensate for blood loss from accidents, surgeries, or internal bleeding.
- Treating Blood Disorders: To manage conditions like anemia, hemophilia, and sickle cell disease.
- Supporting Cancer Treatment: To counteract the effects of chemotherapy and radiation therapy.
- Types of Blood Transfusions
- Whole Blood: Contains all blood components: red blood cells, white blood cells, platelets, and plasma.
- Packed Red Blood Cells: Primarily used to increase oxygen-carrying capacity in the blood.
- Platelets: Used to improve blood clotting.
- Plasma: Contains proteins essential for blood clotting and other functions.

Blood Types and Compatibility

Blood types (A, B, AB, and O) and Rh factor (positive or negative) determine blood compatibility. Type O-negative blood is considered a universal donor, while type AB+ is a universal recipient.

Risks and Complications

- While blood transfusions are generally safe, potential risks include:
- Allergic Reactions: Mild to severe allergic reactions can occur.
- Infection: There's a very low risk of contracting bloodborne infections like HIV or hepatitis.
- Acute Lung Injury: A rare but serious complication that can cause lung damage.
- Iron Overload: Repeated transfusions can lead to excess iron buildup, which can damage organs.

To minimize risks, blood banks rigorously screen donated blood for infections and ensure compatibility with the recipient's blood type.

The dataset comes from a mobile blood donation vehicle in Taiwan, managed by the Blood Transfusion Service Center. The vehicle visits various universities as part of a blood donation drive. The objective is to predict whether a donor will donate blood during the next campus visit.

The data, stored in (datasets/transfusion.data), follows the RFMTC marketing model a variation of the RFM model. This structure will be explored further in the project.

3.2 Data Collection and Importing

Data collection is a crucial step in any predictive modeling project as it involves gathering relevant and reliable information from various sources. In this project, we aim to forecast blood supply, a critical issue for blood collection managers. The data for this project has been collected from the donor database of the Blood Transfusion Service Center. The dataset, sourced from the Machine Learning Repository from GitHub, includes a random sample of 748 donors.

Data importing involves loading the collected data into the coding environment, where it can be manipulated and analyzed. For this project, the dataset is imported into Python using the pandas library, which allows for easy manipulation, cleaning, and transformation of the data. After importing, the dataset is prepared for further analysis, including splitting it into training and testing sets, performing exploratory data analysis, and applying machine learning techniques to build a predictive model using libraries like TPOT and logistic regression.

Packages Used:

- Pandas: Pandas is a powerful Python library primarily used for data manipulation and analysis. It provides efficient data structures like DataFrame for handling large datasets and allows for various operations such as reading, cleaning, and analyzing data. It is particularly useful for handling tabular data (like CSV files) and supports a wide range of file formats for reading and writing data, including CSV, Excel, and SQL.

Functions Used:

- open(): The open() function is used to open a file and return a file object. It is essential for reading and writing files in Python. The file can be opened in different modes such as 'r' (read), 'w' (write), and 'a' (append). In the provided code, it opens the file at the specified path (file_path) and allows reading its contents.
- readline(): The readline() function reads a single line from the file at a time. In the provided code, it is used to print the first 5 lines of the file. It reads until it encounters a line break, making it useful for iterating over each line in a file.
- pd.read_csv(): This function from the pandas library reads a CSV file and converts it into a DataFrame, which is a 2D table-like structure with rows and columns. It is highly customizable, allowing you to specify parameters like delimiter, column names, and data types. In the provided code, it reads the transfusion.data file and stores it in a DataFrame called transfusion, which can then be manipulated or analyzed further.

```
Sample Code:
 Code 1:
file_path = 'datasets/transfusion.data'

with open(file_path, 'r') as file:
for i in range(5):  # Read and print the first 5 lines
print(file.readline().strip())

Code 2:
import pandas as pd
transfusion = pd.read_csv("datasets/transfusion.data")
print(transfusion.head(5))
```

3.3 Designing DataFrame

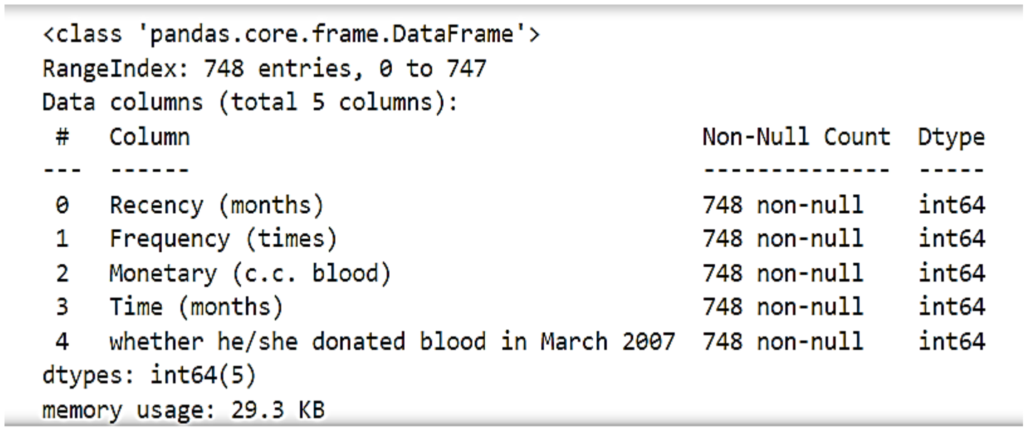
Once the data has been collected and imported into the Python environment, it is structured into a DataFrame for easy manipulation and analysis. The dataset in this project uses a variation of the RFM model (Recency, Frequency, Monetary, and Time) to predict whether a blood donor will donate within a given time window. Here's a brief description of the columns in the dataset:

- **R (Recency)**: Months since the last donation.
- **F (Frequency)**: Total number of donations made.
- **M (Monetary)**: Total blood donated (in c.c.).
- **T (Time)**: Months since the first donation.
- A binary **target variable** that indicates whether the donor donated blood in March 2007 (1 for donated, 0 for not donated).

Each column in the DataFrame is numeric, which is ideal for building machine learning models. After importing the data, we verify the data types and ensure the dataset is ready for preprocessing and model building.

RangeIndex: 748 entries, 0 to 747
Data columns (total 5 columns):

Columns	Data type	Size	Extra
Recency (months)	INT64	748	Not Null
Frequency (times)	INT64	748	Not Null
Monetary (c.c. blood)	INT64	748	Not Null
whether he/she donated blood in March 2007	INT64	748	Not Null



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 748 entries, 0 to 747
Data columns (total 5 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
 0   Recency (months)                            748 non-null    int64
 1   Frequency (times)                           748 non-null    int64
 2   Monetary (c.c. blood)                       748 non-null    int64
 3   Time (months)                               748 non-null    int64
 4   whether he/she donated blood in March 2007  748 non-null    int64
dtypes: int64(5)
memory usage: 29.3 KB
```

3.4 Data Cleaning

"Quality data beats fancy algorithms"

Data is the most imperative aspect of Analytics and Machine Learning. Everywhere in computing or business, data is required. However, real-world data often contains issues like incompleteness, inconsistency, or missing values. Corrupted or inaccurate data can impede the model-building process or lead to misleading results. Hence, **Data cleaning** is considered a foundational element of data science.

Data cleaning refers to the process of identifying and correcting or removing incorrect, incomplete, inaccurate, irrelevant, or missing data. It ensures the data is in a usable form, enabling accurate and efficient analysis.

In this project, the **Blood Transfusion Service Center dataset** may contain errors or missing values, which are common issues in real-world data. Various methods and functions are used to clean this dataset before proceeding with the analysis.

Libraries Used:

Pandas: Pandas is a powerful Python library for data manipulation and analysis. It provides data structures like DataFrames that allow efficient handling of structured data. Pandas is widely used for cleaning, filtering, transforming, and analyzing data. It provides functions to handle missing values, perform data transformations, and load data from various file formats.

Functions Used:

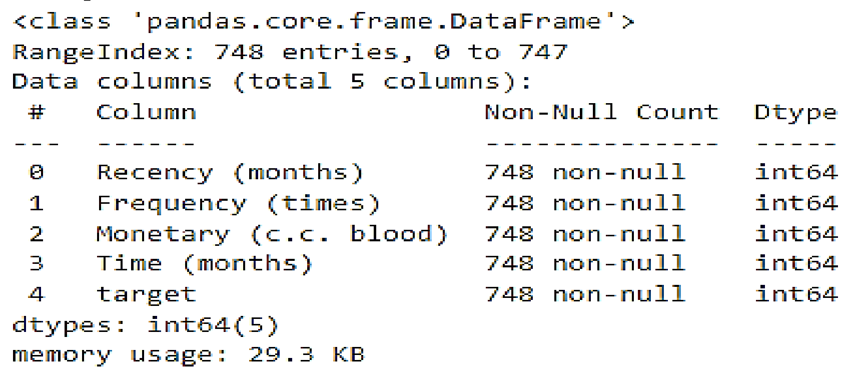
- **`pd.read_csv()`**: This function is used to load the dataset into a DataFrame. It reads the data from a CSV file and allows further manipulation. In this project, we used it to load the transfusion dataset.
- **`transfusion.info()`**: This function provides a concise summary of the DataFrame, including the data types, number of non-null values, and memory usage. It helps to identify missing or incorrect data by providing insights into the structure of the dataset.
- **`rename()`**: This function is used to rename a column for clarity and brevity. In this project, the column "whether he/she donated blood in March 2007" was renamed to simply "target". Renaming helps in making the column names more manageable for further analysis.

These functions ensure that the dataset is clean, structured, and ready for analysis. The next steps involve further preprocessing and feature engineering to prepare the data for machine learning modeling.

Sample Code:

```
#LOADING
transfusion = pd.read_csv("datasets/transfusion.data")
print(transfusion.head(5))
#DATAFRAME SUMMARY
transfusion.info()
#RENAME
transfusion.rename(columns={'whether he/she donated blood in March 2007':
'target'}, inplace=True)
print(transfusion.head(2))
```

Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 748 entries, 0 to 747
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Recency (months)       748 non-null    int64
 1   Frequency (times)      748 non-null    int64
 2   Monetary (c.c. blood)  748 non-null    int64
 3   Time (months)          748 non-null    int64
 4   target                 748 non-null    int64
dtypes: int64(5)
memory usage: 29.3 KB
```

OBSERVATION:

- The column "whether he/she donated blood in march 2007" has renamed to simply "target".

3.5 Data Exploration

In our project, the target variable represents whether a donor will donate blood in the next time window. This is a binary classification problem with two possible outcomes:
- 0 - The donor will not give blood.
- 1 - The donor will give blood.

Target incidence refers to the distribution of these two classes within the dataset, showing the proportion of 0s versus 1s in the target column. Understanding target incidence is critical to identifying whether the dataset is balanced or imbalanced, which can significantly affect the performance of the machine learning model.
Data exploration is the process of analyzing and summarizing data to understand its structure, distribution, and key characteristics. It helps identify trends, patterns, anomalies, or relationships within the dataset. This phase is essential for gaining insights and making informed decisions about data preprocessing, feature engineering, and model selection.
Data exploration involved analyzing the distribution of the target variable to understand its incidence in the dataset. This step provides critical insights into whether the dataset is imbalanced, which can affect the performance of machine learning models.

Package Used

Pandas: The Pandas library is widely used for data exploration. It provides powerful tools for summarizing and analyzing data, such as counting unique values, calculating proportions, and generating descriptive statistics.

Functions Used

- **`value_counts()`**

  The value_counts() function is used to count the occurrences of unique values in a DataFrame column. It is commonly employed for summarizing categorical or numerical data. By setting normalize=True, it converts the counts into proportions, which are useful for understanding relative frequencies. The function helps quickly analyze the distribution of data. It is especially useful in exploratory data analysis to identify patterns or imbalances in datasets.

- **`round()`**

  The `round()` function rounds numeric values to a specified number of decimal places. It ensures data is presented in a concise and easy-to-read format, which is crucial for clear reporting and visualization. This function is often used after performing calculations like proportions or averages to improve interpretability. It supports both series and DataFrame operations in Pandas. By reducing unnecessary detail, it simplifies data interpretation and enhances readability.

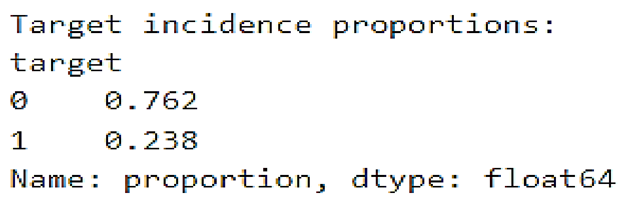
Operators Used

- *Dot Operator* (.):
  Used to access attributes or methods of an object. For example, transfusion['target'].value_counts accesses the value_counts method of the target column in the transfusion DataFrame.
- *Indexing Operator* ([]):
  Used to select specific columns or rows in a DataFrame or Series.
  For example,transfusion['target'] selects the target column.
- *Assignment Operator* (=):
  Used to assign a value to a variable. For example, target_proportions = ... assigns the output of the computation to the variable target_proportions.
- *Method Chaining* (.):
  Allows sequential operations to be performed in a single line by chaining methods together. For example, .value_counts(normalize=True).round(3) applies the value_counts method first and then the round method on its output.
- *Parentheses* (()):
  Used to call functions or methods with arguments. For example, value_counts(normalize=True) passes the normalize=True argument to the value_counts function.

Sample Code:

```
target_proportions =
transfusion['target'].value_counts(normalize=True).
round(3)
print("Target incidence proportions:")
print(target_proportions)
```

Output:



```
Target incidence proportions:
target
0    0.762
1    0.238
Name: proportion, dtype: float64
```

OBSERVATION:

The output shows a class imbalance: **76.2%** of donors are unlikely to give blood again (target = 0), while only **23.8%** are likely to donate again (target = 1). This imbalance can affect the model's performance, favoring the majority class and requiring careful handling during training and evaluation.

3.6 Feature Preprocessing

Feature preprocessing involves preparing the dataset for model training by splitting it into training and testing subsets. This ensures that the model can learn from one set and be evaluated on another, promoting generalizability. In this project, the dataset was split using a stratified approach to maintain the proportion of target classes in both subsets.

Packages Used:

- **scikit-learn:** A widely-used Python library for machine learning, scikit-learn provides a range of efficient tools for data preprocessing, model training, evaluation, and validation. It simplifies tasks such as splitting datasets, handling missing data, and applying machine learning algorithms.
- **pandas:** Although primarily used for data manipulation, pandas supports key preprocessing tasks like removing columns or rows, creating subsets of data, and ensuring that only relevant features are passed into the model.

Functions Used:

- **train_test_split()**: This function from scikit-learn makes splitting data into training and testing subsets efficient. It requires parameters such as:
- **test_size**: Defines the proportion of the dataset to be used as the test set (e.g., 0.25 means 25% testing data).
- **random_state**: Ensures reproducibility by setting a seed for random splitting.
- **stratify**: Maintains the same class distribution in both the training and testing datasets by sampling proportionately.

Operators Used:

- **drop():** Removes unwanted columns (like the target column) from the features to ensure that only input variables are used for training the model.

Explaining Training and Testing:
In machine learning, the **training dataset** is used to teach the model by identifying patterns and relationships between features and the target variable. The **testing dataset**, on the other hand, is used to evaluate the model's performance on unseen data, ensuring that the model generalizes well and avoids overfitting. By using a stratified split, the class imbalance in the target variable (if any) is preserved, which is critical for accurate performance evaluation.

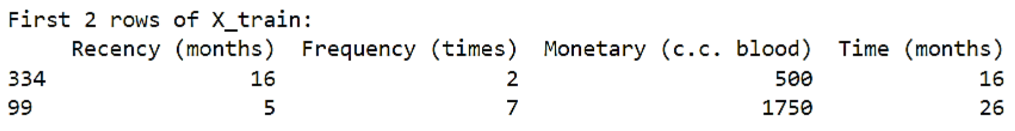
Why use train_test_split()?
Manually splitting a dataset into four parts (features and target for both training and testing) involves writing repetitive and error-prone code. The train_test_split() method automates this process with minimal effort, ensuring that the splits are accurate, random, and reproducible. This function is both time-saving and reliable, making it an essential tool in any data preprocessing pipeline.

Sample Code:

```
                    #IMPORTING
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train and y_test datasets,
      # stratifying on the `target` column
X_train, X_test, y_train, y_test =
train_test_split(transfusion.drop(columns=['target']),transfus
ion['target'],test_size=0.25,random_state=42,stratify=transfus
ion['target'])
print("\nFirst 2 rows of X_train:")
print(X_train.head(2))
```

Output:



```
First 2 rows of X_train:
     Recency (months)  Frequency (times)  Monetary (c.c. blood)  Time (months)
334                16                  2                    500             16
99                  5                  7                   1750             26
```

Observation:

1. Columns in the Dataset:
   - Recency (months): Indicates how many months ago the donor last donated blood.
   - Frequency (times): Total number of times the donor has donated blood in the past.
   - Monetary (c.c. blood): The total amount of blood donated by the donor (in cubic centimeters). Typically, each donation contributes 500 c.c.
   - Time (months): Represents how many months ago the donor first donated blood.
2. Data:

   The first donor (334):
   - Donated blood 16 months ago (Recency = 16).
   - Donated 2 times in total (Frequency = 2).
   - Donated 500 c.c. in total (Monetary = 500).
   - Made their first donation 16 months ago (Time = 16).

   The second donor (99):
   - Donated blood 5 months ago (Recency = 5).
   - Donated 7 times in total (Frequency = 7).
   - Donated 1,750 c.c. in total (Monetary = 1750).
   - Made their first donation 26 months ago (Time = 26).
3. Purpose:

   The train_test_split function has split the original dataset into training (X_train and y_train) and testing (X_test and y_test) subsets.

   These training rows (X_train) will be used to train the machine learning model.
4. Significance of the Output:

   It shows that the data split has been successful.

   The X_train dataset contains the input features (excluding the target column), which are essential for training the model.

3.7 Model Selection

Packages used:

TPOT: TPOT (Tree-based Pipeline Optimization Tool) is a Python library designed for Automated Machine Learning (AutoML). It simplifies the process of building machine learning models by automatically testing and optimizing different machine learning pipelines using genetic programming—a method inspired by biological evolution to find the most suitable solutions.

TPOT Machine Learning Pipeline
TPOT explores hundreds of possible combinations of pre-processing steps, feature transformations, and machine learning models to determine the best pipeline for the dataset.
Example components in a pipeline may include:

- **Data Preprocessing**: Scaling, normalization, or handling missing values.
- **Feature Engineering**: Selecting or creating new features.
- **Model Training**: Fitting algorithms like logistic regression, decision trees, or random forests.

Scikit-learn (sklearn)
Scikit-learn is a popular Python library for machine learning, offering tools for data preprocessing, model training, evaluation, and pipeline creation. It provides the necessary functions for metric evaluation (e.g., **roc_auc_score**) and dataset handling.

Functions Used

1. TPOTClassifier()

- *Purpose:* The core class of TPOT used to create an automated pipeline for classification problems.
- *Features:*
    - Allows customization of genetic programming parameters like the number of generations and population size.
    - Includes built-in support for different scoring metrics (e.g., AUC, accuracy).
    - Outputs the best machine learning pipeline after optimization.

2. fit()

- *Purpose*: Trains the TPOTClassifier on the training data.
- *Functionality*: Iteratively evaluates different combinations of preprocessing steps, algorithms, and hyperparameters, ultimately selecting the pipeline with the best performance metric.

3. **predict_proba()**

- *Purpose*: Outputs probabilities for each class rather than just the predicted class.
- *Use in this context*: The probability of the positive class is used to calculate the AUC-ROC score.

4. **roc_auc_score()**

- *Purpose*: Evaluates the quality of the model by calculating the Area Under the Curve (AUC) for the Receiver Operating Characteristic (ROC) curve.
- *Key Aspect*: It helps measure how well the model can distinguish between the two target classes.

5. **fitted_pipeline_**

- *Purpose*: Retrieves the best machine learning pipeline selected by TPOT.
- *Use in this context*: Displays the step-by-step pipeline, including preprocessing and modeling steps.

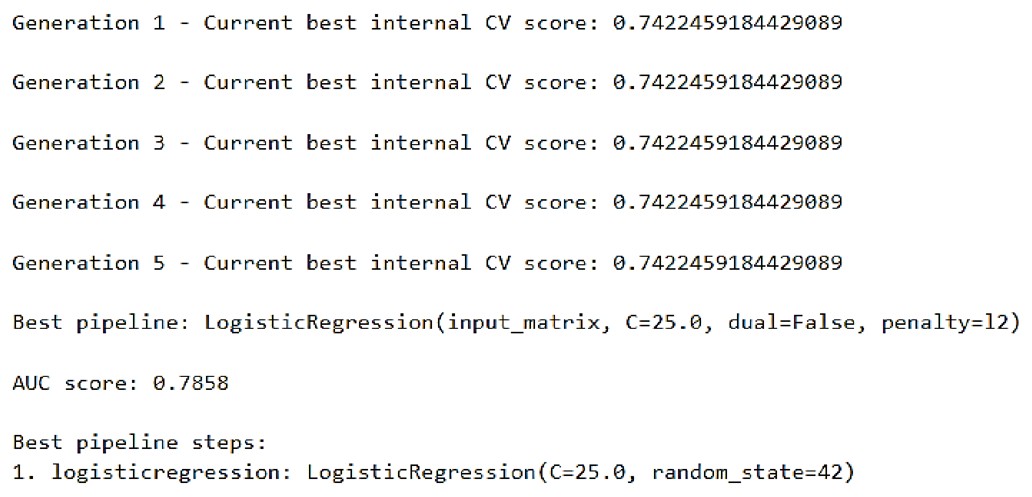
Operators Used

- **Colon (:)**: Used to separate items in key-value pairs or for slicing arrays.
- **Comma (,)**: Separates arguments in function calls or elements in lists.
- **Square Brackets ([])**: Used for indexing or slicing arrays.
- **Dot (.)**: Accesses attributes or methods of objects.

Sample Code:

```
#IMPORTING
from tpot import TPOTClassifier
from sklearn.metrics import roc_auc_score
# Instantiate TPOTClassifier
tpot = TPOTClassifier(generations=5,population_size=20,
verbosity=2,scoring='roc_auc',random_state=42,
disable_update_check=True,config_dict='TPOT light')
# Train the TPOT model
tpot.fit(X_train, y_train)
# AUC score for tpot model
tpot_auc_score = roc_auc_score(y_test, tpot.predict_proba(X_test)[:, 1])
print(f'\nAUC score: {tpot_auc_score:.4f}')
# Print best pipeline steps
print('\nBest pipeline steps:', end='\n')
for idx, (name, transform) in enumerate(tpot.fitted_pipeline_.steps, start=1):
# Print idx and transform
print(f'{idx}. {name}: {transform}')
```

Output:



```
Generation 1 - Current best internal CV score: 0.7422459184429089

Generation 2 - Current best internal CV score: 0.7422459184429089

Generation 3 - Current best internal CV score: 0.7422459184429089

Generation 4 - Current best internal CV score: 0.7422459184429089

Generation 5 - Current best internal CV score: 0.7422459184429089

Best pipeline: LogisticRegression(input_matrix, C=25.0, dual=False, penalty=l2)

AUC score: 0.7858

Best pipeline steps:
1. logisticregression: LogisticRegression(C=25.0, random_state=42)
```

Observation:

1. **Generations**: TPOT iterates over 5 generations, optimizing pipelines, but the best CV score (0.7422) remains unchanged across all generations.
2. **Best Pipeline**: Logistic Regression with hyperparameters (C=25.0, penalty=l2, dual=False, random_state=42) is the best pipeline found by TPOT.

    - **C=25.0**: This is the regularization strength. A higher value for C means less regularization (i.e., the model will focus more on fitting the training data, potentially leading to overfitting). In this case, a value of 25.0 suggests the model is allowed some flexibility in fitting the data.
    - **penalty=l2**: This refers to the type of regularization applied to the model. "L2" regularization penalizes large coefficients, helping prevent overfitting by shrinking the model's parameters toward zero.
    - **dual=False**: This hyperparameter specifies whether to solve the primal or dual optimization problem. In most cases, setting it to False works well for Logistic Regression.
    - **random_state=42**: This ensures reproducibility. It initializes the random number generator so that results can be replicated exactly each time the code is run.

3. **AUC Score**: The final model's AUC score is **0.7858**, indicating the model's performance in distinguishing between the classes. An **AUC score of 0.7858** means that the model has an 78.58% probability of correctly distinguishing between a randomly chosen positive instance and a randomly chosen negative instance. Here's how to interpret it:
    - **0.5** indicates no discrimination (random guessing).
    - **1.0** indicates perfect classification (no errors).
    - **0.7858** is a good score, meaning the model performs better than random guessing and is able to distinguish between the positive and negative classes with reasonable accuracy. The closer the score is to 1, the better the model is at making accurate predictions.

4. **Pipeline Steps**: The best pipeline includes a Logistic Regression model as the final step, showing the specific regularization and optimization settings used.

3.8 Variance Checking & Log normalization

Checking the Variance:

- *Purpose*: Variance in features can impact the performance of models, particularly linear models like Logistic Regression. If one feature has a significantly higher variance than others, the model may overemphasize its importance, skewing the learning process.

- *Why It Matters*: Logistic Regression assumes a linear relationship between features and the target. High variance in one feature can dominate the optimization process and reduce the model's ability to learn from other features effectively.

- *Normalization*: This is a preprocessing technique used to adjust features to be on a similar scale. If high variance is detected, normalization ensures that no single feature disproportionately influences the model.

Log Normalization:

- *Why It's Needed*: The *Monetary (c.c. blood)* feature has significantly higher variance compared to the other features in the dataset. Without correction, the model might treat this feature as overly important, which could lead to biased or suboptimal predictions.

- *What It Does*: Log normalization reduces the scale of a feature by applying a logarithmic transformation, which compresses large values while maintaining the overall distribution. This transformation reduces the variance of the feature, aligning it better with others in the dataset.

- *Benefit*: Log normalization helps prevent the model from assigning undue importance to features with large numerical ranges, ensuring a fair contribution from all features to the learning process.

This step ensures that the model remains balanced and can interpret relationships between features more effectively.

Packages and Functions Used

1. **Pandas (`var()` method)**:
   - **Description**: Used to calculate the variance of each column in the DataFrame. The `var()` method measures how data points are spread out in each feature of the dataset.
   - **Purpose**: Helps identify features with high variance that may require normalization.
2. **Numpy (`np.log()` function)**:
   - **Description**: This function computes the natural logarithm of a column's values to reduce its range and address high variance.
   - **Purpose**: Used for log normalization, ensuring all features are on a comparable scale for better model performance.
3. **`idxmax()`**:
   - **Description**: Finds the name of the column (feature) with the maximum variance.
   - **Purpose**: Pinpoints which column needs log normalization due to high variance.

4. **`drop()`**:
    - **Description**: Removes the column with the highest variance after it has been log-transformed.
    - **Purpose**: Ensures only the normalized version of the column is retained in the dataset.
5. **`inplace=True`**:
    - **Description**: Modifies the DataFrame directly without creating a new copy.
    - **Purpose**: Used in `drop()` to save memory and simplify code.

Operators Used

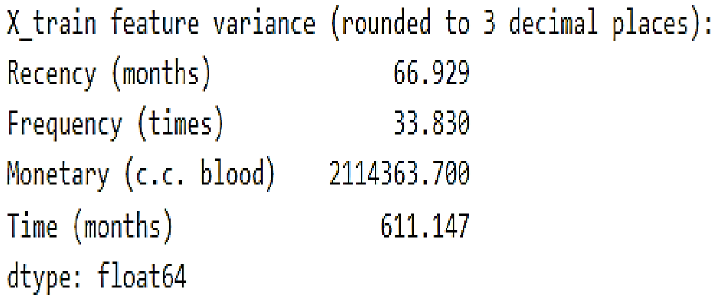
- **`[]`**: Indexing to select columns for transformation or normalization.
- .: Accessing methods or attributes of objects like DataFrames (`var()`, `copy()`)

Sample Code:

```
#VARIANCE CHECKING
X_train_variance = X_train.var()
print("\nX_train feature variance (rounded to 3 decimal places):")
print(X_train_variance.round(3))
# Log normalization
import numpy as np
X_train_normed,X_test_normed = X_train.copy(), X_test.copy()
col_to_normalize = X_train.var().idxmax()
for df_ in [X_train_normed, X_test_normed]
df_['monetary_log'] = np.log(df_[col_to_normalize])
df_.drop(columns=[col_to_normalize], inplace=True)
print("\nX_train_normed variance:")
print(X_train_normed.var().round(3))
```

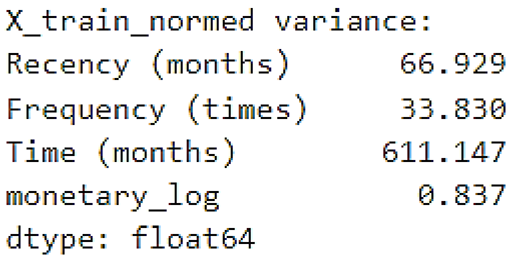
Output:

#VARIANCE CHECKING



```
X_train feature variance (rounded to 3 decimal places):
Recency (months)               66.929
Frequency (times)              33.830
Monetary (c.c. blood)     2114363.700
Time (months)                 611.147
dtype: float64
```

# Log normalization



```
X_train_normed variance:
Recency (months)      66.929
Frequency (times)     33.830
Time (months)        611.147
monetary_log           0.837
dtype: float64
```

Observation:

- Variance of Features Before Normalization
  - Observation: The feature Monetary (c.c. blood) has a significantly higher variance (2,114,363.7) compared to other features like Recency (66.929) and Frequency (33.83).
  - Implication: This high variance could bias the model by giving undue importance to this feature, making normalization necessary.
- Variance of Features After Normalization
  - Observation: After log normalization, the variance of monetary_log has reduced to 0.837, bringing it closer to the variance of other features. The variances of the other features remain unchanged.
  - Implication: Normalization has successfully reduced the impact of the previously high variance feature, ensuring a more balanced dataset for model training.

3.9 Training the linear regression model

The variance of features has been adjusted, with *Monetary (c.c. blood)* normalized using log transformation to bring its variance in line with other features. Now, *Time (months)* has the highest variance, but since it is not drastically larger than the others, no further adjustment is necessary. With the features balanced, the dataset is ready for training the linear regression model.

Packages Used:

- *Scikit - linear_model*:
  This package is part of the scikit-learn library and provides implementations of various linear models like LogisticRegression, LinearRegression, Ridge, and others. In this case, LogisticRegression is used for binary classification tasks, where the goal is to predict one of two outcomes.
- *Scikit - roc_auc_score:*
  This package provides various metrics for evaluating the performance of machine learning models. roc_auc_score calculates the Area Under the Receiver Operating Characteristic Curve (AUC-ROC), which is a common evaluation metric for classification models, especially when dealing with imbalanced datasets.

Functions Used:

- *linear_model.LogisticRegression(solver='liblinear', random_state=42):*
  This function creates an instance of the Logistic Regression model. The solver='liblinear' parameter specifies the optimization algorithm used to fit the model. random_state=42 ensures that the results are reproducible across different runs of the code.
- *logreg.fit(X_train_normed, y_train):*
  The fit() function trains the Logistic Regression model using the training data (X_train_normed for features and y_train for target labels). It adjusts the model parameters to best fit the training data.

- *logreg.predict_proba(X_test_normed)*
  This function generates probability predictions for each class (0 and 1) for the test data (X_test_normed). It returns a 2D array where each row contains the predicted probabilities for each class.
- *roc_auc_score(y_test, logreg.predict_proba(X_test_normed)[:, 1]):*
  This function calculates the AUC score by comparing the true target values (y_test) with the predicted probabilities for the positive class (class 1). The AUC score helps assess the model's ability to discriminate between the classes.
- *print(f'\nAUC score: {logreg_auc_score:.4f}'):*
  This function is used to print the AUC score of the model, formatted to 4 decimal places. It provides a measure of the model's performance in terms of classification accuracy.

Sample Code:

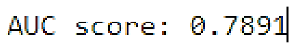
```
# Importing
from sklearn import linear_model

# Instantiate LogisticRegression
logreg
linear_model.LogisticRegression(solver='liblinear',random_state=42)

# Train the model
logreg.fit(X_train_normed, y_train)

# AUC score for tpot model
logreg_auc_score = roc_auc_score(y_test,
logreg.predict_proba(X_test_normed)[:, 1])
print(f'\nAUC score: { logreg_auc_score:.4f}')
```
Output:

Output:



```
AUC score: 0.7891
```

Observation:

- AUC score: 0.7891:
  This value indicates the performance of the logistic regression model. An AUC score of 0.7891 means that the model has a good ability to distinguish between the two classes (donor vs. non-donor).
- AUC Range:
  0.5 indicates random guessing (no model discrimination).
  1.0 indicates perfect classification.
  An AUC score of 0.7891 suggests the model performs well, though there is room for improvement in model tuning.

III Performance Observations: AUC Score Analysis

Libraries and Functions Used:

OPERATOR:

- Library: operator
- Function: itemgetter
- Description: The code imports itemgetter from Python's operator module. itemgetter is a function that extracts an element from a collection (like a list or tuple) based on the given index. It is commonly used to sort collections..

SORTED():

- The list [(‘tpot’, tpot_auc_score), (‘logreg’, logreg_auc_score)] contains tuples, each representing a model (tpot and logreg) and its corresponding AUC score.
- **key=itemgetter(1)**: This means the sorting will be done based on the second element of each tuple (the AUC score), which is at index 1.
- **reverse=True**: This ensures that the sorting is done in descending order, meaning the higher AUC score will appear first.

Sample Code:

```
# Importing itemgetter
from operator import itemgetter

# Sort models based on their AUC score from highest to lowest
sorted([('tpot', tpot_auc_score), ('logreg',
logreg_auc_score)], key=itemgetter(1),reverse=True)
```

Output:

```
[('logreg', 0.7890972663699937), ('tpot', 0.7857596948506039)]
```

Observation:

- **('logreg', 0.7891):** This means the Logistic Regression model (logreg) has the higher AUC score of 0.7891, making it the better-performing model based on AUC.
- **('tpot', 0.7858):** This means the TPOT model (tpot) has a slightly lower AUC score of 0.7858 compared to Logistic Regression.

In summary, **Logistic Regression (logreg) outperforms the TPOT model (tpot)** based on the AUC score, as shown by the sorted output.

IV CONCLUSION AND FUTURE SCOPE

The demand for blood fluctuates throughout the year, with periods like busy holiday seasons seeing a slowdown in donations. An accurate forecast of future blood supply allows timely actions, helping to save more lives. In this notebook, we used TPOT for automatic model selection, and the best AUC score achieved was 0.7850. This outperforms a baseline model that always predicts '0' (no donation), which would have a 76% success rate based on the target incidence. By applying log normalization to the training data, we managed to improve the AUC score by 0.5%. In machine learning, even small improvements in accuracy can make a significant difference, depending on the application. Furthermore, the logistic regression model offers interpretability, allowing us to assess how much of the target variable's variance can be explained by other features in the dataset.

V. REFERENCES

Data Collection

The following websites have been referred to obtain the input data and statistics:

- https://github.com/irshadbaruah/Predict-Blood-Donations.git
- https://github.com/satyadeepS7/Predict-Blood-Donation-for-Future-Expectancy.git
- https://www.webmd.com/a-to-z-guides/blood-transfusion-what-to-know#1
- https://www.mayoclinic.org/tests-procedures/blood-transfusion/about/pac-20385168

Programming References

The following websites have been referred for Python tutorials:

- https://datascienceplus.com/category/programming/?tdo_tag=Python
- https://docs.python.org/3/tutorial/index.html
- https://www.w3schools.com/python/
- https://www.geeksforgeeks.org/python-programming-language-tutorial/
- https://www.codecademy.com/catalog/language/python
- https://www.javatpoint.com/python-tutorial