FINAL REPORT OF INTERNSHIP PROGRAM 2023 ON "**PREDICTING BLOOD DONATIONS**"

ANJANI UPADHYAY

DATA ANALYTICS INTERN, MEDTOUREASY

ACKNOWLEDGEMENTS

The internship with MedTourEasy was an enriching experience for me which enabled me to learn in-depth about various tools and technologies currently used in the field of data analytics as well as gave me an opportunity to apply my knowledge while working with a real-world dataset to make insightful predictions. I feel grateful to have gotten the opportunity to work with professionals whose guidance has been extremely valuable during the course of my project.

I would like to begin by expressing my heartfelt gratitude towards the Training and Development team at MedTourEasy who believed in my potential and provided me this wonderful opportunity to intern at their esteemed organization. I'm also thankful to them for providing the IBM Data Analytics Professional Course as a part of the training resources which helped me learn the necessary skills for the successful completion of my project.

I would also like to thank the entire team of MedTourEasy and I am glad to have had this internship opportunity which served as a great booster to my personal as well as professional development. Thank you again.

TABLE OF CONTENTS

INTRODUCTION

1.1 About the Company

MedTourEasy, a global healthcare company, provides you the informational resources needed to evaluate your global options. It helps you find the right healthcare solution based on specific health needs, affordable care while meeting the quality standards that you expect to have in healthcare. MedTourEasy improves access to healthcare for people everywhere. It is an easy to use platform and service that helps patients to get medical second opinions and to schedule affordable, high-quality medical treatment abroad.

1.2 About the Project

A blood transfusion is a routine medical procedure in which donated blood is provided through a narrow tube placed within a vein in the arm. This potentially life-saving procedure can help replace blood lost due to surgery or injury. A blood transfusion can also help if an illness prevents the body from making blood or some of the blood's components correctly. Depending on how much blood is needed, a transfusion can take between 1 and 4 hours. About 5 million Americans need a blood transfusion every year and the procedure is usually safe.

Blood has several components, including:
a. Red cells carry oxygen and help remove waste products
b. White cells help the body fight infections
c. Plasma is the liquid part of the blood
d. Platelets help the blood clot properly

A transfusion provides the part or parts of blood needed, with red blood cells being the most commonly transfused. You can also receive whole blood, which contains all the parts, but whole blood transfusions aren't common. Researchers are working on developing artificial blood. So far, no good replacement for human blood is available.

Several cases require blood transfusion. Some of them are listed below :-

a. A major surgery which requires the replacement of lost blood.
b. Bleeding in the digestive tract from an ulcer or some other condition.
c. An illness like leukemia or kidney disease that causes anemia (not enough healthy red blood cells).
d. Cancer treatments like radiation or chemotherapy.
e. Blood disorder or severe liver problems.

Blood transfusion is a crucial challenge in blood supply chain management, especially when there exists a high demand, but not enough blood inventory. By the means of this project, we aim to predict the likelihood of a blood donor donating blood again within a given window of time. The Blood Transfusion Service Center drives to different universities and collects blood as part of a blood drive. We want to predict whether or not a donor will give blood the next time the vehicle comes to campus. The data is structured according to RFMTC marketing model (a variation of RFM, commonly used in marketing for identifying your best customers).

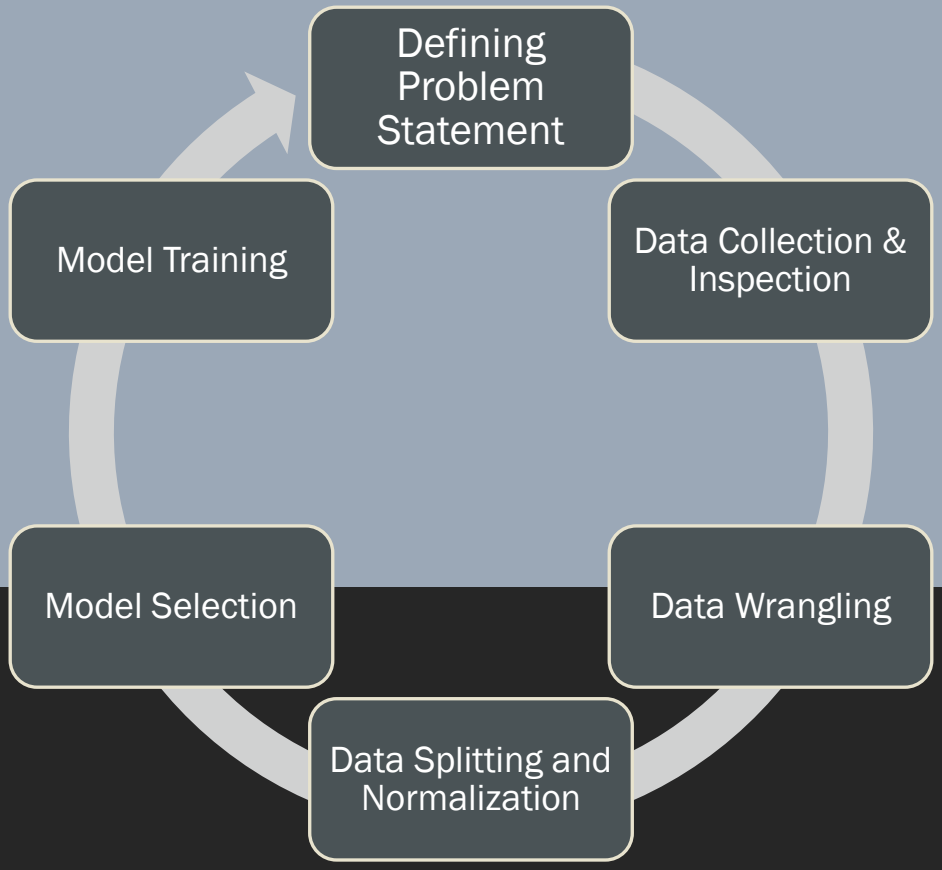
Below is a description of what each column means in our dataset:

R (Recency - months since the last donation)
F (Frequency - total number of donation)
M (Monetary - total blood donated in c.c.)
T (Time - months since the first donation)
C, a binary variable representing whether he/she donated blood in March 2007 (1 stands for donating blood; 0 stands for not donating blood)

METHODOLOGY

2.1 Flow of the Project

The project followed the steps given in the chart below in order to accomplish the desired objectives. Each step has been explained in detail in the next section.



2.2 Languages and Platform used

2.2.1 Language : Python

Python is a high-level general-purpose programming language whose design philosophy mostly emphasizes code readability with the use of significant code. Writing code in Python for both small and large scale projects becomes very easy even for the budding coders of today as the language constructs and Object-Oriented Approach of Python makes the code extremely clear and logical. Python supports a variety of programming paradigms like Object Oriented Programming, Structured Programming, Functional Programming, etc., and has a rich set of libraries like NumPy, Pandas, etc. which makes it a great choice for a variety of technical fields like Data Science, Machine Learning, etc. Key features of Python are :-

1) Easy to learn and readable language
2) Interpreted language
3) Open source and free
4) Large standard library
5) High level language
6) Object Oriented Programming (OOP) Language
7(Platform independent
8) Extensible and embeddable
9) Graphical User Interface (GUI) Support

2.2.2 Integrated Development Environment (IDE) : Jupyter Notebook

Jupyter Notebook allows you to combine code, comments, multimedia, and visualizations in an interactive document — called a notebook, naturally — that can be shared, re-used, and re-worked. As Jupyter Notebook runs via a web browser, the notebook itself could be hosted on your local machine or on a remote server. Originally developed for data science applications written in Python, R, and Julia, Jupyter Notebook is useful in all kinds of ways for all kinds of projects. The most common use cases for Jupyter Notebook are data science, mathematics, and other research projects that involve visualizations of data or formulas. Key features of Jupyter Notebook are :-

1) Data visualization
2) Code sharing
3) Live interactions with code
4) Documenting code samples
5) Multi-language support
6) Ease of use

2.2.3 Package : Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language. It is already well on its way toward this goal. The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. Pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries. Key features of Pandas are :-

1) Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
2) Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
3) Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
4) Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
5) Hierarchical labeling of axes (possible to have multiple labels per tick)
6) Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases.

2.2.4 Pipeline Optimization Tool : TPOT

TPOT is an automated machine learning package in python that uses genetic programming concepts to optimize the machine learning pipeline. It automates the most tedious part of machine learning by intelligently exploring thousands of the possible to find the best possible parameter that suits your data. TPOT is built upon the scikit-learn, so its code looks similar to the scikit-learn.

IMPLEMENTATION

3.1 Defining the Problem Statement

The first and foremost step for solving any data analysis problem is defining the problem statement itself so as to have a clearer picture of what questions need to be answered by the means of the analysis performed in the given project. In this project, our problem statement is **predicting whether or not a donor will likely donate blood the next time a mobile blood donation vehicle in Taiwan comes to the university campus as a part of a blood drive**.

3.2 Data Collection and Inspection

In the next step, we inspect the data to get a brief idea about the dataset as well as the different fields and values it contains by using the **with** statement along with the **open** function to open the file. This is followed by storing the contents of the file in a list using the **readlines** function and then iterating the list line by line using a **for loop** to print out the data inside the file.

```
In [10]: # Print out the first 5 lines from the transfusion.data file
with open("C:\\Users\\91989\\Downloads\\Data Analyst (1)\\Give Life_ Predict Blood Donations\\datasets\\transfusion.data", 'r
    linesList= filedata.readlines()
    for line in (linesList[:5]):
        print(line, end ='')

Recency (months),Frequency (times),Monetary (c.c. blood),Time (months),"whether he/she donated blood in March 2007"
2 ,50,12500,98 ,1
0 ,13,3250,28 ,1
1 ,16,4000,35 ,1
2 ,20,5000,45 ,1
```

Now, in order to utilise the wide variety of functions in the pandas library of python and further continue the analysis of our dataset, we need to import the dataset to the jupyter notebook in the form of a **pandas dataframe**. For that, we first import the **pandas** library using the **import** statement. Next, since our dataset is a **CSV (Comma Separated Values) file**, we load the data into memory using the **read_csv** function with the url of the dataset file as the parameter To verify the successful import of our data, we print the first 5 rows of our dataset using the **head** function which displays the dataframe with 5 columns as shown below.

In [15]:
```
# Import pandas
import pandas as pd

# Read in dataset
url="C:\\Users\\91989\\Downloads\\Data Analyst (1)\\Give Life_ Predict Blood Donations\\datasets\\transfusion.data"
transfusion = pd.read_csv(url)

# Print out the first rows of our dataset
# ... YOUR CODE FOR TASK 2 ...
transfusion.head()
```

Out[15]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
0	2	50	12500	98	1
1	0	13	3250	28	1
2	1	16	4000	35	1
3	2	20	5000	45	1
4	1	24	6000	77	0

We can further perform some inspection on our newly created dataframe in order to get the gist of its contents by using the **info** method to print a concise summary of our dataframe including the number of **non-null values** in each column, **datatype** of each column as well as **total memory** occupied by the dataframe

```
In [17]: # Print a concise summary of transfusion DataFrame
         # ... YOUR CODE FOR TASK 3 ...
         transfusion.info()
```

```
<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
```

3.3 Data Wrangling

Data Wrangling is the process of gathering, collecting, and transforming raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging. If data is incomplete, unreliable, or faulty, then analyses will be too, diminishing the value of any insights gleaned. Data wrangling seeks to remove that risk by ensuring data is in a reliable state before it's analyzed and leveraged. This makes it a critical part of the analytical process.

Data wrangling in python deals with the below functionalities :-

1.Data exploration: In this process, the data is studied, analyzed and understood by visualizing representations of data.

2.Dealing with missing values: Most of the datasets having a vast amount of data contain missing values of NaN, they are needed to be taken care of by replacing them with mean, mode, the most frequent value of the column or simply by dropping the row having a NaN value.

3.Reshaping data: In this process, data is manipulated according to the requirements, where new data can be added or pre-existing data can be modified.

4.Filtering data: Some times datasets are comprised of unwanted rows or columns which are required to be removed or filtered

5.Other: After dealing with the raw dataset with the above functionalities we get an efficient dataset as per our requirements and then it can be used for a required purpose like data analyzing, machine learning, data visualization, model training etc.

As per the summary obtained about our dataset using the info method in the previous step, there are **no null or missing values** in the dataset. Hence, that issue is not something that needs to be taken care of and we further proceed with our data wrangling process by renaming the last column **“whether he/she donated blood in March 2007”** to **“”target”** using the **rename function** for the sake of our convenience as the target column has the values which we need to predict. We also set the **inplace** parameter to **True** so that the changes we are making are reflected directly in the original dataframe. We confirm the same by printing out the first two rows.

In [18]:
```
# Rename target column as 'target' for brevity
transfusion.rename(columns={'whether he/she donated blood in March 2007': 'target'}, inplace=True)

# Print out the first 2 rows
# ... YOUR CODE FOR TASK 4 ...
transfusion.head(2)
```

Out[18]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	target
0	2	50	12500	98	1
1	0	13	3250	28	1

As the model given for this dataset is a **binary classifier**, there are only 2 possible outcomes :-

0 - the donor will not give blood
1 - the donor will give blood

Thus finally, in order to check the **target incidence proportions**, i.e. the number of cases of each individual target value in a dataset which conveys the number of 0s that are present in the target column as compared to the number of 1s so that we get an idea of how balanced or imbalanced our dataset is, we use the **value_counts** method on the target column while also setting the parameter **normalize** to **True** and round off the resultant values to 3 decimal places for the sake of convenience using the **round** function.

```
In [21]: # Print target incidence proportions, rounding output to 3 decimal places
         # ... YOUR CODE FOR TASK 5 ...
         transfusion['target'].value_counts(normalize=True).round(3)

Out[21]: 0    0.762
         1    0.238
         Name: target, dtype: float64
```

3.4 Data Splitting

After cleaning and refining our dataset, the next step is to make the dataframe ready for model training by splitting into training as well as testing sets. For this task, we use the **train_test_split** method from the scikit learn library. The train_test_split method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train, X_test, y_train, and y_test. X_train and y_train sets are used for training and fitting the model. The X_test and y_test sets are used for testing the model if it's predicting the right outputs/labels. We can explicitly test the size of the train and test sets. It is suggested to keep our train sets larger than the test sets.

1) Train set: The training dataset is a set of data that was utilized to fit the model. The dataset on which the model is trained. This data is seen and learned by the model.
2) Test set: The test dataset is a subset of the training dataset that is utilized to give an accurate evaluation of a final model fit.
3) Validation set: A validation dataset is a sample of data from your model's training set that is used to estimate model performance while tuning the model's hyperparameters.
4) Underfitting: A data model that is under-fitted has a high error rate on both the training set and unobserved data because it is unable to effectively represent the relationship between the input and output variables.
5) Overfitting: When a statistical model matches its training data exactly but the algorithm's goal is lost because it is unable to accurately execute against unseen data, it is called overfitting.

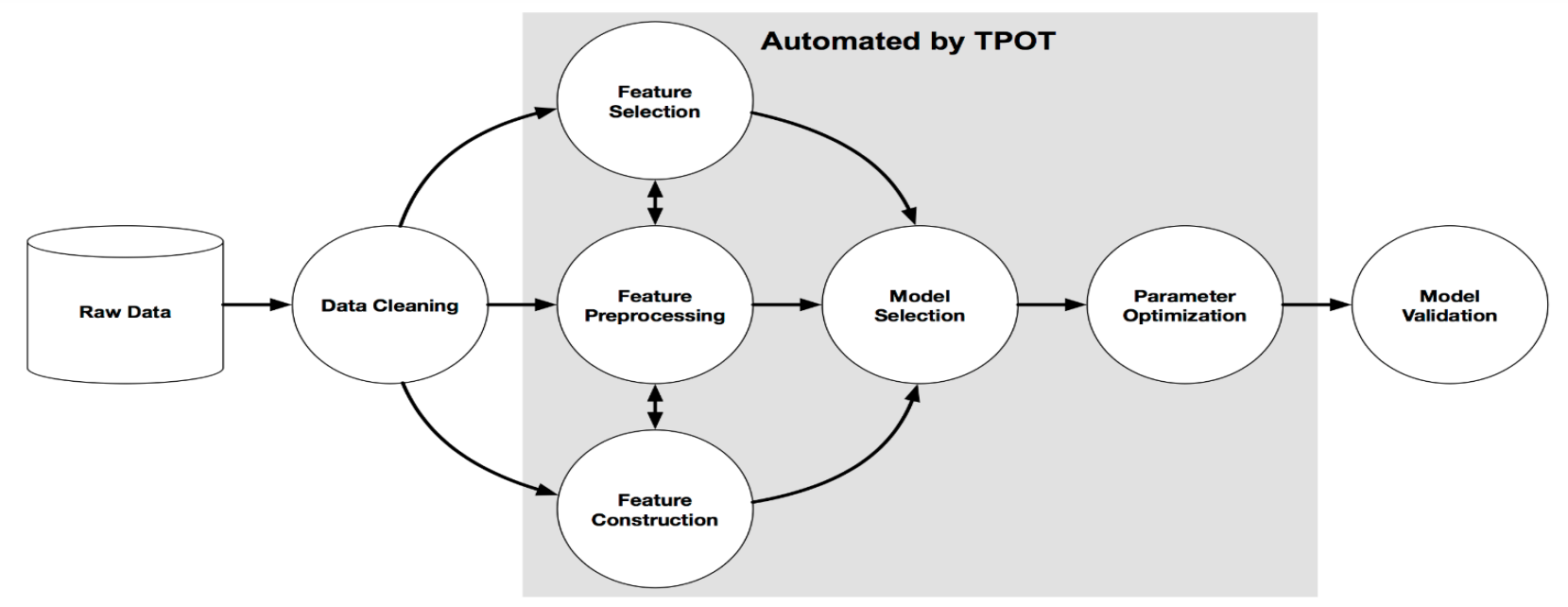
In [25]:
```
# Import train_test_split method
from sklearn.model_selection import train_test_split

# Split transfusion DataFrame into
# 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'), transfusion.target, test_size=0.25,
                                                    random_state=42, stratify=transfusion.target)

# Print out the first 2 rows of X_train
# ... YOUR CODE FOR TASK 6 ...
X_train.head(2)
```

Out[25]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)
334	16	2	500	16
99	5	7	1750	26

3.5 Model Selection and Normalization

The last step before training our model is selecting the appropriate type of model according to our dataset as well as the problem at-hand that needs to be solved. In this project, we make the use of **TPOT library** to find the **best machine learning pipeline**. TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. TPOT will automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for your data. Once TPOT is finished searching, it provides you with the Python code for the best pipeline it found so you can tinker with the pipeline from there. TPOT has the same API as scikit-learn, i.e., you first instantiate a model and then you train it, using the **fit** method. Data pre-processing affects the model's performance, and tpot's **fitted_pipeline_** attribute will allow you to see what pre-processing (if any) was done in the best pipeline.



In [37]:

```python
# Import TPOTClassifier and roc_auc_score
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'
)
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}. {transform}')
```

```
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: 0.7456308339276876

Best pipeline: MultinomialNB(Normalizer(input_matrix, norm=l2), alpha=0.001, fit_prior=True)

AUC score: 0.7637

Best pipeline steps:
1. Normalizer()
2. MultinomialNB(alpha=0.001)
```

As depicted in the previous slide, the TPOT Classifier selects **MultinomialNB** as the best pipeline for our dataset with an **AUC score** of **0.7637**. The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. AUC is a common abbreviation for **Area Under the Receiver Operating Characteristic Curve (ROC AUC)**. It is a metric used to assess the performance of classification machine learning models. The ROC is a graph which maps the relationship between true positive rate (TPR) and the false positive rate (FPR), showing the TPR that we can expect to receive for a given trade-off with FPR. The AUC score is the area under this ROC curve, meaning that the resulting score represents in broad terms the model's ability to predict classes correctly. The AUC score ranges from 0 to 1, where 1 is a perfect score and 0.5 means the model is as good as random.

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 we can perform before training a model. Hence, we check the variance in the X_train dataset to see if any such transformation is needed.

```
In [41]: # X_train's variance, rounding the output to 3 decimal places
         # ... YOUR CODE FOR TASK 8 ...
         pd.DataFrame.var(X_train).round(3)

Out[41]: Recency (months)              66.929
         Frequency (times)             33.830
         Monetary (c.c. blood)    2114363.700
         Time (months)                611.147
         dtype: float64
```

From the previous results, we observe that the column **“Monetary (c.c. blood)”** has a very high variance in comparison to the other columns in the dataset. Hence, unless accounted for, this feature may get more weight by the model than any other feature resulting in inaccurate predictions and results. One way to correct for high variance is to use **log normalization**. Log normalization applies a logarithmic transformation to our values, which transforms them onto a scale that approximates normality - an assumption that many models make. To keep the code "DRY" (Don't Repeat Yourself), we are using a for-loop to apply the same set of transformations to each of the dataframes.

In [42]:
```
# Import numpy
import numpy as np

# Copy X_train and X_test into X_train_normed and X_test_normed
X_train_normed, X_test_normed = X_train.copy(), X_test.copy()

# Specify which column to normalize
col_to_normalize = 'Monetary (c.c. blood)'

# Log normalization
for df_ in [X_train_normed, X_test_normed]:
    # Add log normalized column
    df_['monetary_log'] = np.log(df_[col_to_normalize])
    # Drop the original column
    df_.drop(columns=col_to_normalize, inplace=True)

# Check the variance for X_train_normed
# ... YOUR CODE FOR TASK 9 ...
pd.DataFrame.var(X_train_normed).round(3)
```

Out[42]:
```
Recency (months)      66.929
Frequency (times)     33.830
Time (months)        611.147
monetary_log           0.837
dtype: float64
```

3.6 Model Training

In the final step, we are now ready to train our model using the training datasets. We do so by importing the **linear_model** module from **sklearn** and instantiating **Logistic Regression** by creating its object and assigning it to **logreg** variable. We then use the **fit** method to train the logreg model. The scikit-learn library has a consistent API when it comes to fitting a model: Create an instance of a model you want to train and train it on your train datasets using the fit method. Finally, we print the AUC score using **roc_auc_score** method to get a score of **0.7891** which is a great improvement over our previous AUC score. We can also sort the models on the basis of their AUC score using the **sorted** method as shown.

In [43]:
```
# Importing modules
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}')
```

```
AUC score: 0.7891
```

In [50]:
```
# 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 )
```

Out[50]:
```
[('logreg', 0.7890972663699937), ('tpot', 0.7637476160203432)]
```

LINK TO THE JUPYTER NOTEBOOK :-

http://localhost:8888/notebooks/Downloads/Data%20Analyst%20(1)/Give%20Life_%20Predict%20Blood%20Donations/notebook.ipynb

CONCLUSION

The demand for blood fluctuates throughout the year. As one prominent example, blood donations slow down during busy holiday seasons. An accurate forecast for the future supply of blood allows for an appropriate action to be taken ahead of time and therefore saving more lives.

In this notebook, we explored automatic model selection using TPOT and AUC score we got was 0.7637. This is better than simply choosing 0 all the time (the target incidence suggests that such a model would have 76% success rate). We then log normalized our training data and improved the AUC score to 0.7891. In the field of machine learning, even small improvements in accuracy can be important, depending on the purpose.

REFERENCES

https://scikit-learn.org/

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#:~:text=A%20blood%20transfusion%20is%20a,due%20to%20surgery%20or%20injury.

https://www.interviewbit.com/blog/features-of-python/

https://www.infoworld.com/article/3347406/what-is-jupyter-notebook-data-analysis-made-easier.html

https://www.g2.com/products/the-jupyter-notebook/features

https://pandas.pydata.org/docs/getting_started/overview.html

https://www.geeksforgeeks.org/

http://epistasislab.github.io/tpot/