Final Report of Internship Program 2021

On

“Predict Blood Donations”

By:- Parnasree Das

MEDTOUREASY

30th April 2021

ACKNOWLDEGMENTS

The internship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Data Analysis and Different Prediction Models ; 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 internship 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 internship 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 busyschedule.

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 transfusion saves lives - from replacing lost blood during major surgery or a serious injury to treating various illnesses and blood disorders. Ensuring that there's enough blood in supply whenever needed is a serious challenge for the health professionals. According to WebMD, "about 5 million Americans need a blood transfusion every year"

Forecasting blood supply is a serious and recurrent problem for blood collection managers. In this Project, you will work with data collected from the donor database of Blood Transfusion Service Center. The dataset, obtained from the Machine Learning Repository, consists of a random sample of 748 donors. Your task will be to predict if a blood donor will donate within a given time window. You will look at the full model-building process: from inspecting the dataset to using the **tpot** library to automate your Machine Learning pipeline. To complete this Project, you need to know some Python, pandas, and logistic regression

I. 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

In this Project, I work with data collected from the donor database of Blood Transfusion Service Center. The dataset, obtained from the Machine Learning Repository, consists of a random sample of 748 donors. Your task will be to predict if a blood donor will donate within a given time window

Our dataset is from a mobile blood donation vehicle in Taiwan. 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.

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

1.3 Object and Deliverables

This project focuses on creating easily understandable, interactive and dynamic dashboards by gathering data of Blood Donation from the Machine Learning Repository and using coding language Python and packages like pandas, logistic regression, Numpy, Tpot and sklearn. Packages to analyze these statistics which will enable the firm to analyze the situation and find the end conclusive

The project consists of 3 Instructions detailed as follows (3 deliverables):

1:- Selecting model using TPOT

TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. TPOT will automatically explore hundreds of possible pipelines to find the best one for our dataset

2:- 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.

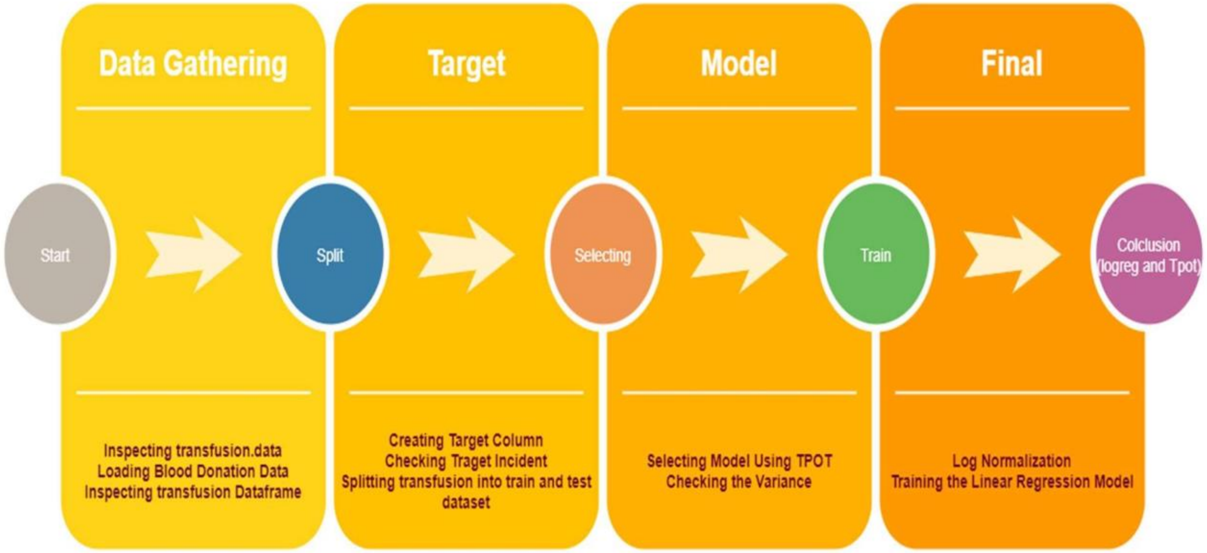
3:- Training the linear regression model

The variance looks much better now. Notice that now `Time (months)` has 4 the largest variance, but it's not the orders of magnitude higher than the rest of the variables

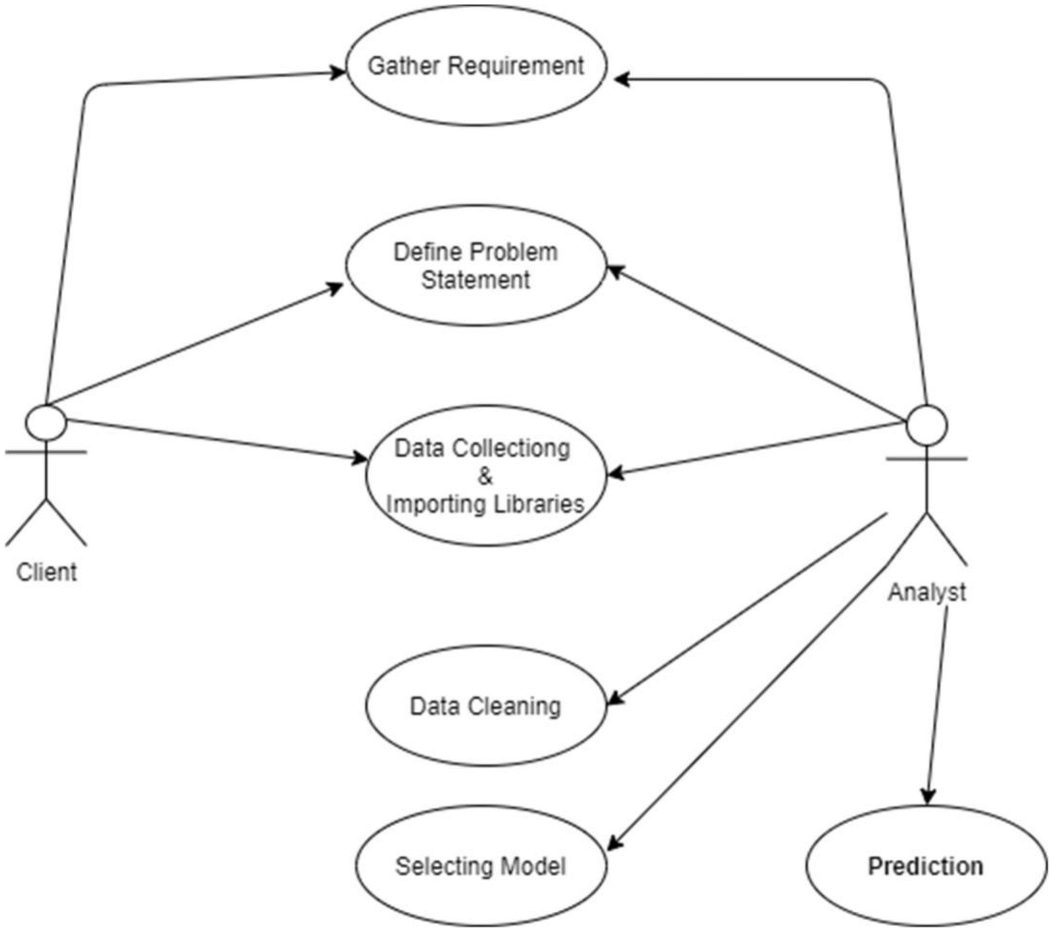
II. METHODOLOGY

2.1 Flow of the Project

The project followed the following steps to accomplish the desired objectives and deliverables.


Data Gathering
Target
Model
Final
Start
Split
Selecting
Train
Colclusion (logreg and Tpot)
Inspecting transfusion.data
Loading Blood Donation Data
Inspecting transfusion Dataframe
Creating Target Column
Checking Traget Incident
Splitting transfusion into train and test dataset
Selecting Model Using TPOT
Checking the Variance
Log Normalization
Training the Linear Regression Model

2.2 Use Case Diagram



Above figure shows the use case of the project. There are two main actors in the same: The Client and Analyst. The analyst will first gather requirements and define the problem statement then collecting the required data and importing it
Then the analyst cleans the data select the model and predict the needed result

2.3 Language and Platform Used

2.3.1 Language: Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

2.3.2 IDE: Google Colaboratory

Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

What Colab Offers You?

As a programmer, you can perform the following using Google Colab.

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g. from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

III. IMPLEMENTATION

3.1 Gathering Requirements and Defining Problem Statement

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.

3.2 Data Collection

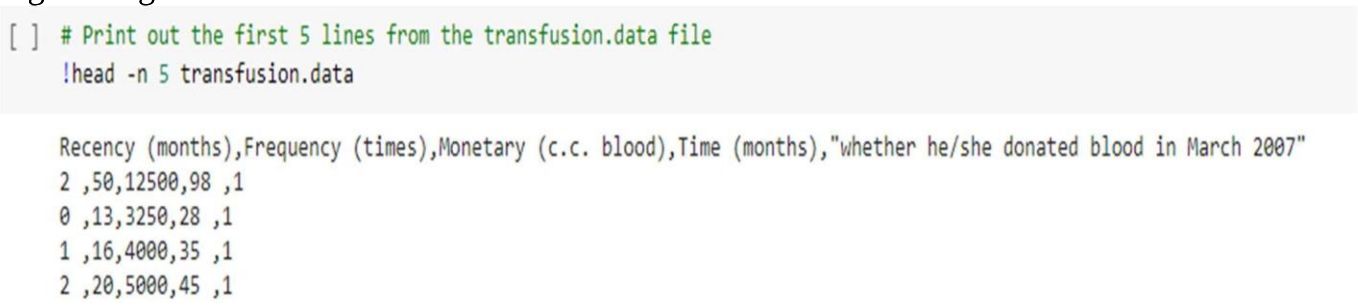
Data collection is a systematic approach for gathering and measuring information from a variety of sources in order to obtain a complete and accurate picture of an interest area. It helps an individual or organization to address specific questions, determine outcomes and forecast future probabilities and patterns.

The dataset, obtained from the Machine Learning Repository, consists of a random sample of 748 donors.

3.3 Code

1. Inspecting transfusion.data file

Inspect the file that contains the dataset. • Print out the first 5 lines from datasets/transfusion.data using the head shell command. Make sure to first read the narrative for each task in the notebook on the right before reading the more detailed instructions here. To complete this Project, you need to know some Python, pandas, and logistic regression. We recommend one is familiar with the content

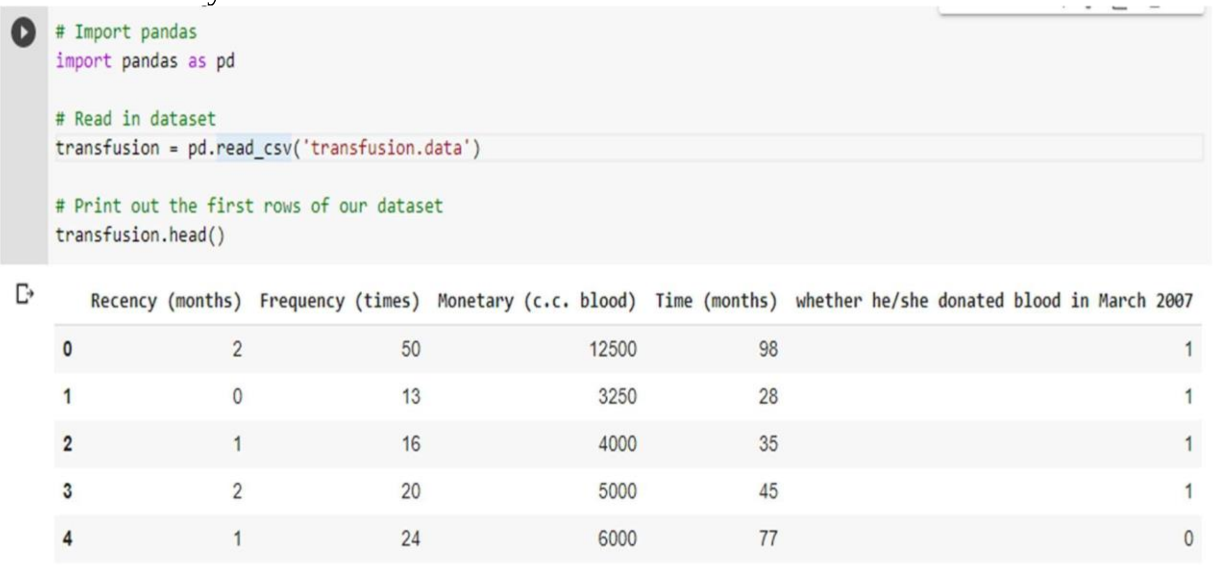


```
[ ] # Print out the first 5 lines from the transfusion.data file
    !head -n 5 transfusion.data
```

```
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
```

2. Loading the blood donations data

• Load the dataset.
• Import pandas library.
• Load the transfusion.data file from datasets/transfusion.data and assign it to the transfusion variable.
• Display the first rows of the DataFrame with the head() method to verify the file was loaded correctly.

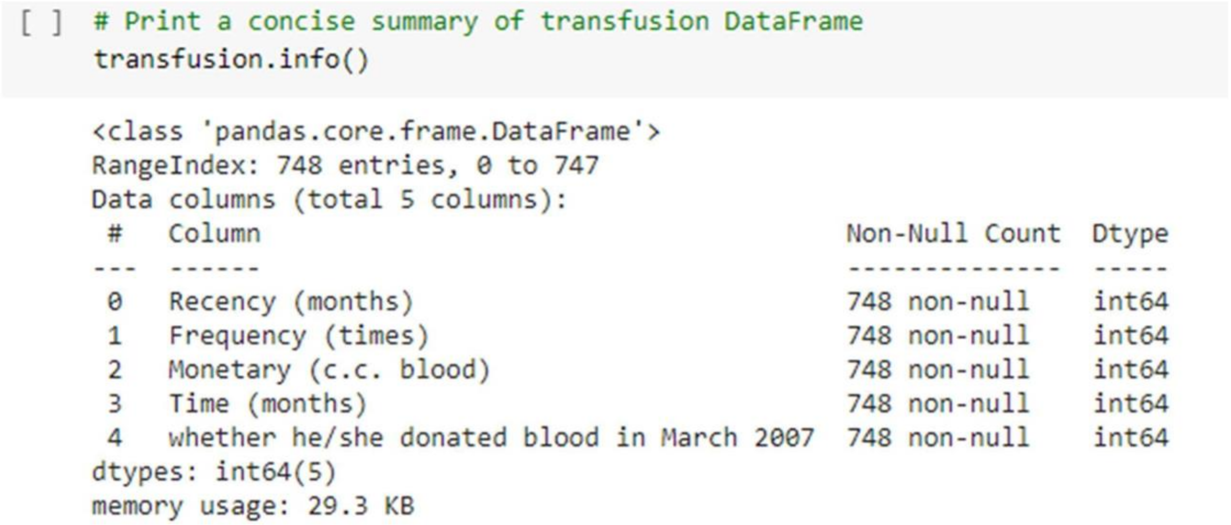


```python
# Import pandas
import pandas as pd

# Read in dataset
transfusion = pd.read_csv('transfusion.data')

# Print out the first rows of our dataset
transfusion.head()
```

	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

3. Inspecting transfusion DataFrame

Inspect the DataFrame's structure.
• Print a concise summary of the transfusion DataFrame with the info() method.
DataFrame's info() method prints some useful information about a DataFrame:
• index type
• column types
• non-null values
• memory usageincluding the index dtype and column dtypes, non-null values and memory usage.

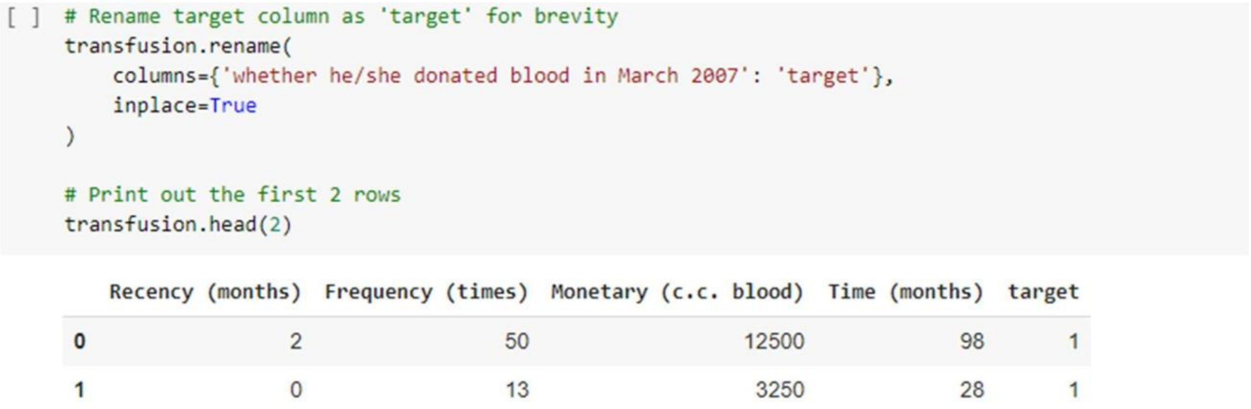


```
[ ]  # Print a concise summary of transfusion DataFrame
     transfusion.info()

     <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
```

4. Creating target column

Rename a column.
• Rename whether he/she donated blood in March 2007 to target for brevity.
• Print the first 2 rows of the DataFrame with the head() method to verify the change was done correctly.

By setting the inplace parameter of the rename() method to True, the transfusion DataFrame is changed in-place, i.e., the transfusion variable will now point to the updated DataFrame as you'll verify by printing the first 2 rows.



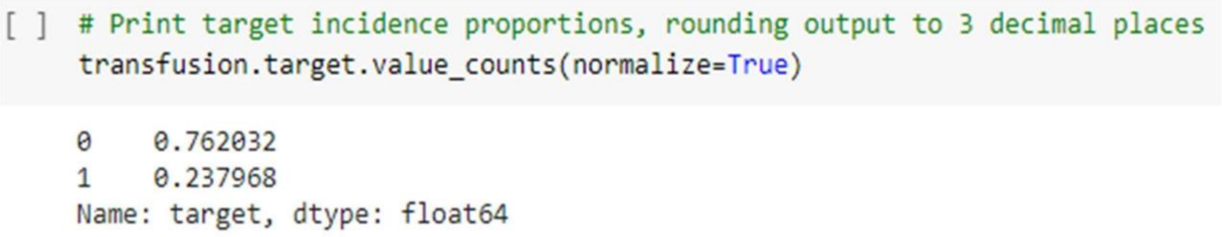
```
[ ] # Rename target column as 'target' for brevity
    transfusion.rename(
        columns={'whether he/she donated blood in March 2007': 'target'},
        inplace=True
    )

    # Print out the first 2 rows
    transfusion.head(2)
```

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

5. Checking target incidence

Print target incidence.
- Use value_counts() method on transfusion.target column to print target incidence proportions, setting normalize=True and rounding the output to 3 decimal places.

By default, value_counts() method returns counts of unique values. By setting normalize=True, the value_counts() will return the relative frequencies of the unique values instead.



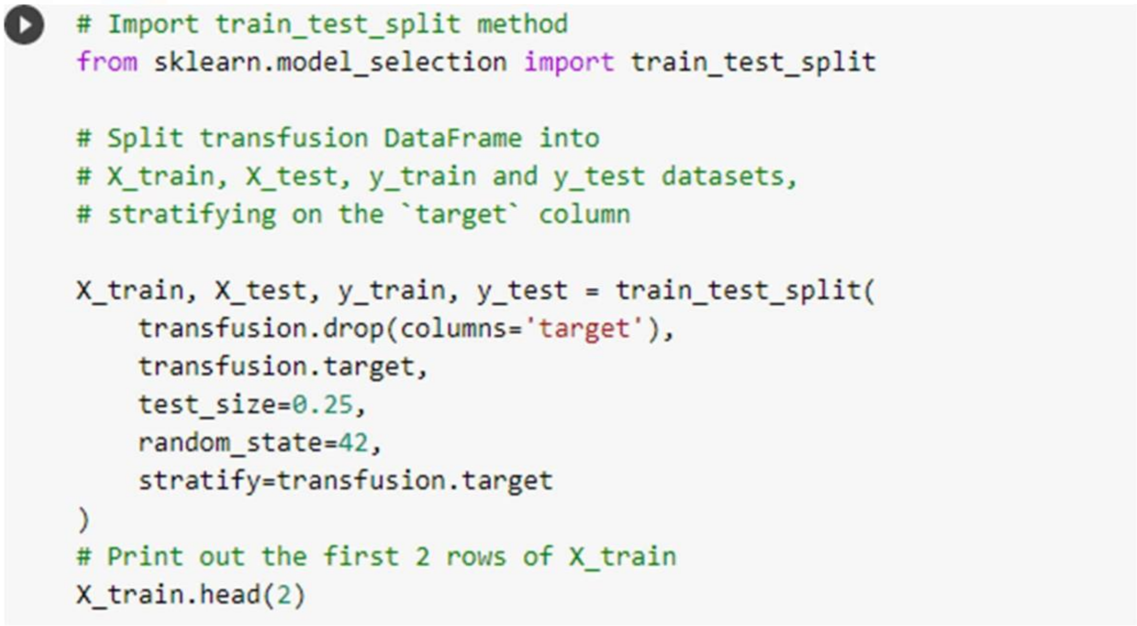
```
[ ] # Print target incidence proportions, rounding output to 3 decimal places
    transfusion.target.value_counts(normalize=True)
```

```
0    0.762032
1    0.237968
Name: target, dtype: float64
```

6. Splitting transfusion into train and test datasets

Split the transfusion DataFrame into train and test datasets.
- Import train_test_split from sklearn.model_selection module.
- Split transfusion into X_train, X_test, y_train and y_test datasets, stratifying on the target column.
- Print the first 2 rows of the X_train DataFrame with the head() method.

Writing the code to split the data into the 4 datasets needed would require a lot of work. Instead, you will use the train_test_split() method in the scikit-learn library



```
# 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
X_train.head(2)
```

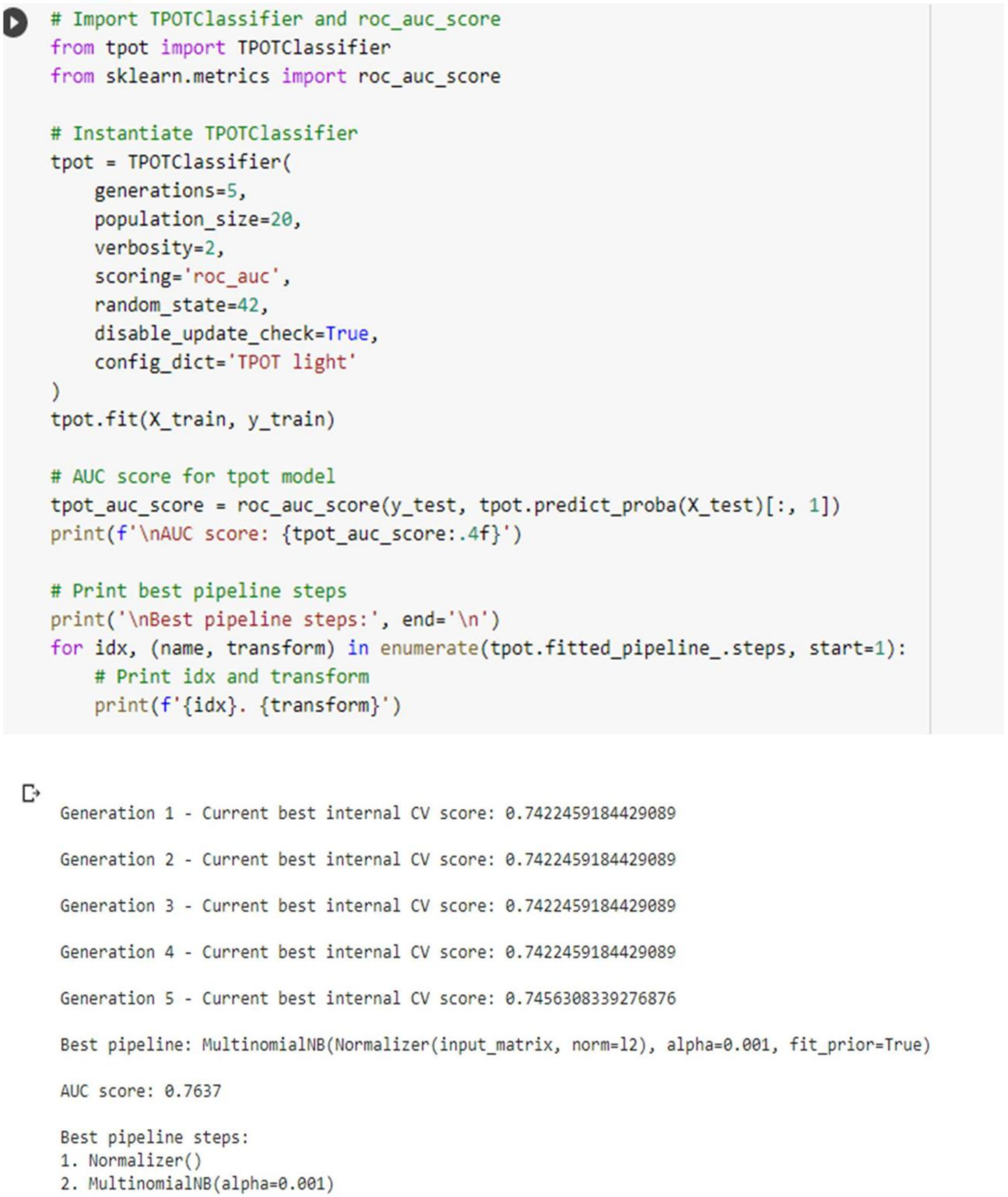
7. Selecting model using TPOT

Use the TPOT library to find the best machine learning pipeline.

- Import TPOTClassifier from tpot and roc_auc_score from sklearn.metrics.
- Create an instance of TPOTClassifier and assign it to tpot variable.
- Print tpot_auc_score, rounding it to 4 decimal places.
- Print idx and transform in the for-loop to display the pipeline steps.

You will adapt the classification example from the TPOT's documentation. In particular, you will specify scoring='roc_auc' because this is the metric that you want to optimize for and add random_state=42 for reproducibility. You'll also use TPOT light configuration with only fast models and preprocessors.

The nice thing about TPOT is that it 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



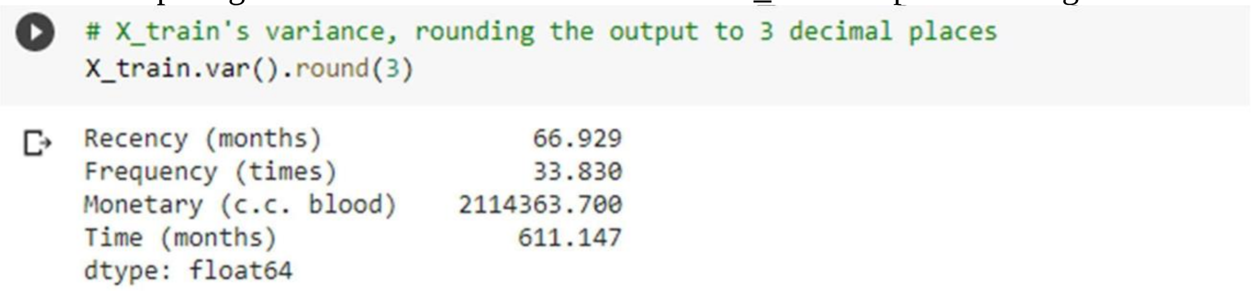
```python
# Import TPOTClassifier and roc_auc_score
from tpot import TPOTClassifier
from sklearn.metrics import roc_auc_score

# 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)
```

8. Checking the variance

Check the variance.
• Print X_train's variance using var() method and round it to 3 decimal places.

pandas.DataFrame.var() method returns column-wise variance of a DataFrame, which makes comparing the variance across the features in X_train simple and straightforward



```
# X_train's variance, rounding the output to 3 decimal places
X_train.var().round(3)
```

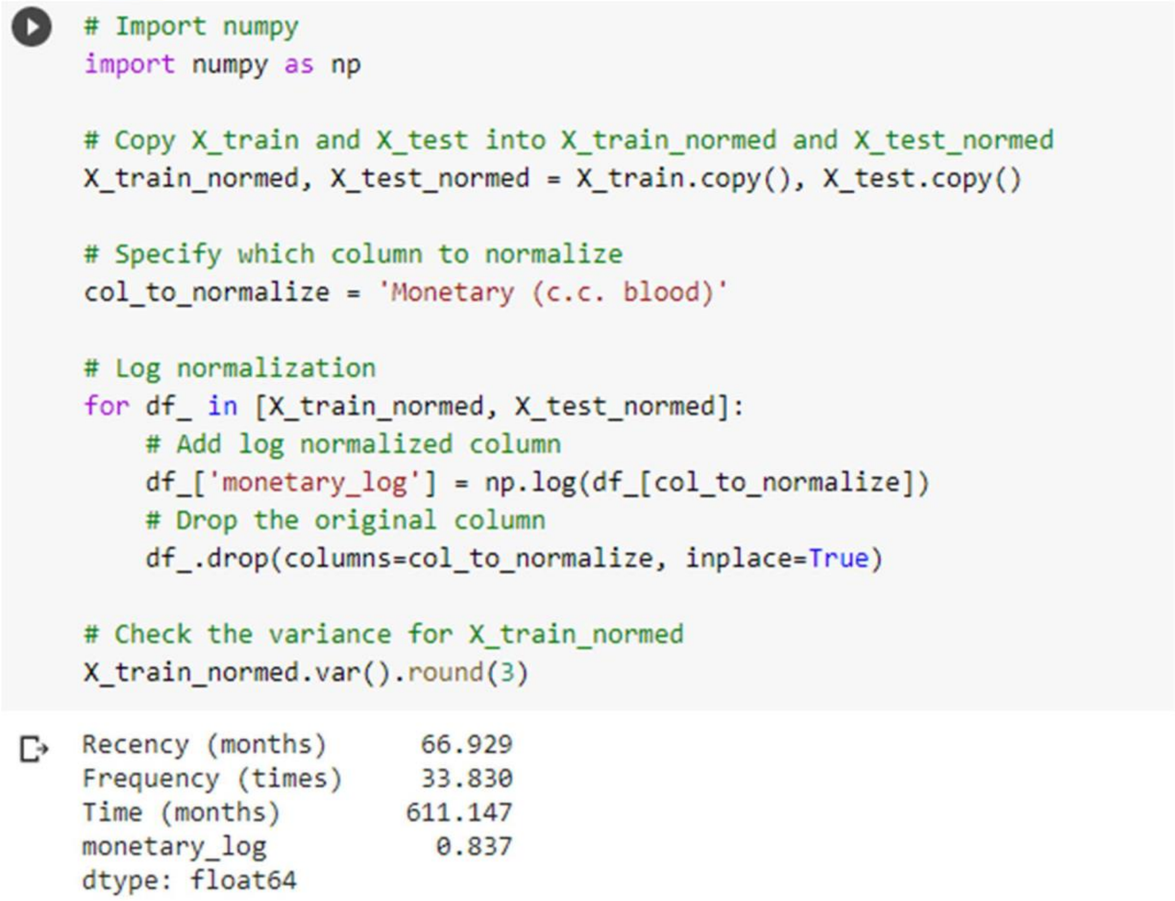
```
Recency (months)             66.929
Frequency (times)            33.830
Monetary (c.c. blood)   2114363.700
Time (months)               611.147
dtype: float64
```

9. Log normalization

Correct for high variance.
• Copy X_train and X_test into X_train_normed and X_test_normed respectively.
• Assign the column name (a string) that has the highest variance to col_to_normalize variable.
• For X_train and X_test DataFrames:.
• Log normalize col_to_normalize to add it to the DataFrame.
• Drop col_to_normalize. • Print X_train_normed variance using var() method and round it to 3 decimal places.

X_train and X_test must have the same structure. To keep your code "DRY" (Don't Repeat Yourself), you are using a for-loop to apply the same set of transformations to each of the DataFrames.

Normally, you'll do pre-processing before you split the data (it could be one of the steps in machine learning pipeline). Here, you are testing various ideas with the goal to improve model performance, and therefore this approach is fine.



```
# 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
X_train_normed.var().round(3)
```

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

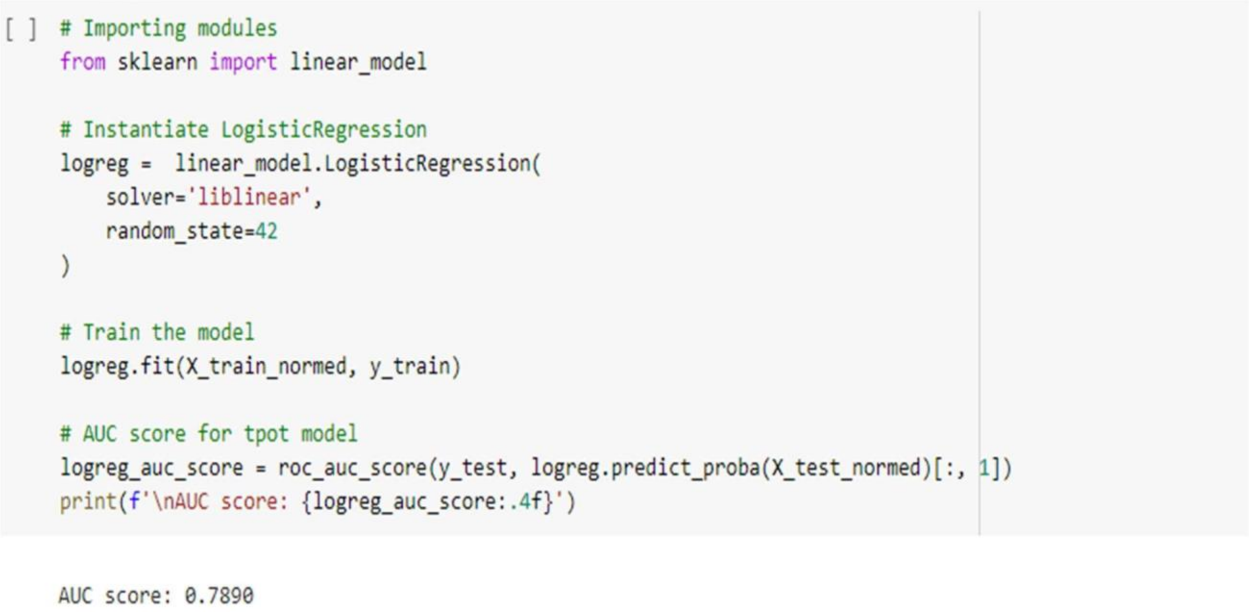
10. Training the linear regression model

Train the logistic regression model.

- Import linear_model from sklearn.
- Create an instance of linear_model.LogisticRegression and assign it to logreg variable.
- Train logreg model using the fit() method.
- Print logreg_auc_score.

The scikit-learn library has a consistent API when it comes to fitting a model:
1. Create an instance of a model you want to train.
2. Train it on your train datasets using the fit method.

You may recognise this pattern from when you trained TPOT model. This is the beauty of the scikit-learn library: you can quickly try out different models with only a few code changes.



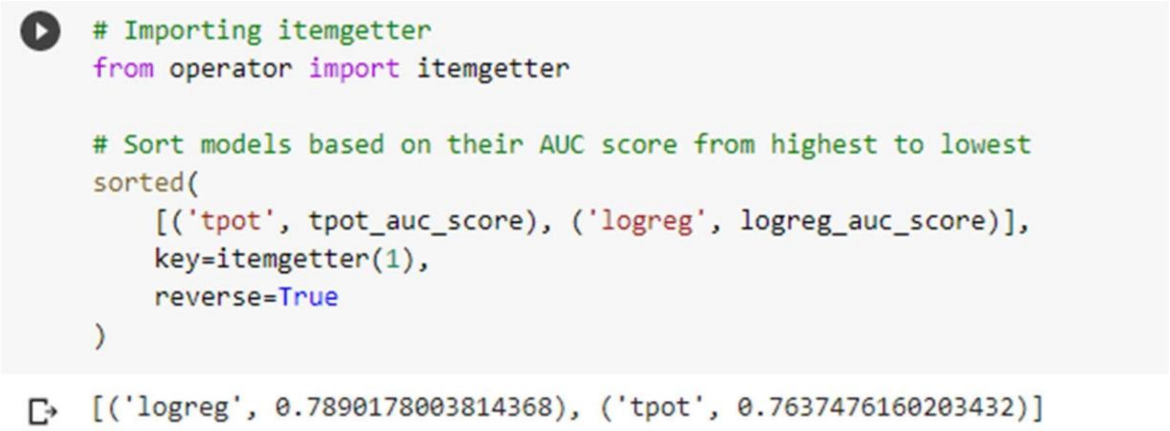
```
[ ]  # Importing modules
     from sklearn import linear_model

     # 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: 0.7890
```

11. Conclusion

Sort your models based on their AUC score from highest to lowest.

- Import itemgetter from operator module.
- Sort the list of (model_name, model_score) pairs from highest to lowest using reverse=True parameter.



```
# 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
)
```

```
[('logreg', 0.7890178003814368), ('tpot', 0.7637476160203432)]
```

CONCLUSION AND FUTURE SCOPE

The entire world is in midst of a serious pandemic which has affected more than 200 countries causing more than 7 million infected cases and 0.4 million deaths. This pandemic has taken its economic and financial toll on most of the major economies of the world.

This project aimed at analyzing the current situation of the pandemic by creating intuitive and user interactive dashboards and drawing conclusions on the impact it will have on the world. Currently, the project is in its last stage of development with the dashboards been developed and submitted for review and feedback.

With regards to the future work, the firm aims at regularly updating the dashboards with time and integrating it with their systems so as to continually draw conclusions and analyze the results. This will enable them to predict future business opportunities and provide a basis on which they can plan on increasing their market presence and capacity planning.