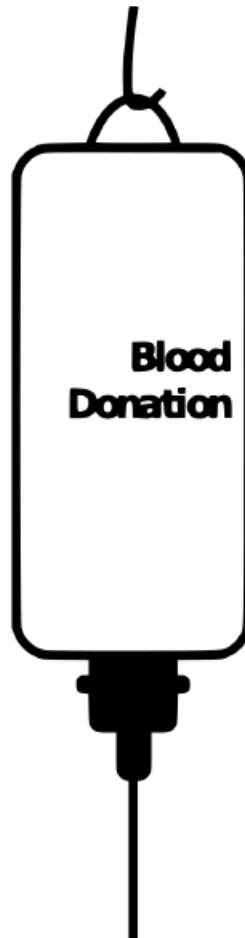1. Inspecting transfusion.data file

Blood transfusion saves lives - from replacing lost blood during majorsurgery or a serious injury to treating various illnesses and blood disorders.Ensuring that there's enough blood in supply whenever needed is a seriouschallenge for the health professionals. According to WebMD([https://www.webmd.com/a-to-z-guides/blood-transfusion-what-to-know#1 (https://www.webmd.com/a-to-z-guides/blood-transfusion-what-to-know#1)](https://www.webmd.com/a-to-z-guides/blood-transfusion-what-to-know#1))

,"about 5 million Americans need a blood transfusion every year". Our dataset is from a mobile blood donation vehicle in Taiwan. The BloodTransfusion Service Center drives to different universities and collects bloodas part of a blood drive. We want to predict whether or not a donor will giveblood the next time the vehicle comes to campus. The data is stored in datasets/transfusion.data and it is structuredaccording to RFMTC marketing model (a variation of RFM). We'll explorewhat that means later in this notebook. First, let's inspect the data.

In [1]:
```python
import os

current_dir = os.getcwd()
print("Present Working Directory:\n", current_dir, sep="")
```

```
Present Working Directory:
C:\myProject\WinPY\WPy64-3830\notebooks
```

```python
# Change the location of current directory to the required locations where datasets resides
os.chdir(r"D:/NYIT (Fall2020-ACADEMICS)/MACHINE LEARNING (DTSC710) [PROFF Xueqing Huang]/PROJECT/dataset")
current_dir = os.getcwd()
print("Changed Directory:\n", current_dir, sep="")

# List all files in the current directory
ls = os.listdir()
print("\nList Of Files:")
for element in range(len(ls)):
    print(ls[element], sep="")

# View the first 20 lines of datasets/transfusion.csv
with open("transfusion.csv") as mysecond_dataset:
    head = [next(mysecond_dataset) for x in range(20)]
print("\nFirst 20 Lines of Dataset transfusion.csv:\n", str(head).strip("[]"), sep="")
```

```
Changed Directory:
D:\NYIT (Fall2020-ACADEMICS)\MACHINE LEARNING (DTSC710) [PROFF Xueqing Huang]\PROJECT\dataset

List Of Files:
DataSet Link.txt
pre-requisite info.txt
transfusion.csv

First 20 Lines of Dataset transfusion.csv:
'Recency (months),Frequency (times),Monetary (c.c. blood),Time (months),"whether he/she donated blood in Mar
ch 2007"\n', '2 ,50,12500,98 ,1\n', '0 ,13,3250,28 ,1\n', '1 ,16,4000,35 ,1\n', '2 ,20,5000,45 ,1\n', '1 ,2
4,6000,77 ,0\n', '4 ,4,1000,4 ,0\n', '2 ,7,1750,14 ,1\n', '1 ,12,3000,35 ,0\n', '2 ,9,2250,22 ,1\n', '5 ,46,
11500,98 ,1\n', '4 ,23,5750,58 ,0\n', '0 ,3,750,4 ,0\n', '2 ,10,2500,28 ,1\n', '1 ,13,3250,47 ,0\n', '2 ,6,1
500,15 ,1\n', '2 ,5,1250,11 ,1\n', '2 ,14,3500,48 ,1\n', '2 ,15,3750,49 ,1\n', '2 ,6,1500,15 ,1\n'
```

2. Loading the blood donations data

We now know that we are working with a typical CSV file (i.e., the delimiter is, etc.). We proceed toloading the data into memory.

3. Inspecting transfusion DataFrame

Let's briefly return to our discussion of RFM model. RFM stands for Recency, Frequency and Monetary Valueand it is commonly used in marketing for identifying your best customers. In our case, our customers areblood donors. RFMTC is a variation of the RFM model. 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) a binary variable representing whether he/she donated blood in March 2007 (1 stands for donatingblood; 0 stands for not donating blood) It looks like every column in our DataFrame has the numeric type, which is exactly what we want whenbuilding a machine learning model. Let's verify our hypothesis.

```
In [3]:   # Import the `pandas` module as "pd"
          import pandas as pd

          # Read in `transfusion.csv`
          transfusion = pd.read_csv("D:/NYIT (Fall2020-ACADEMICS)/MACHINE LEARNING (DTSC710) [PROFF Xueqing Huang]/PROJE

          # Save the number of rows columns as a tuple
          rows_and_cols = transfusion.shape
          print("Brief Description Of Dataset transfusion.csv Stored In DataFrame Object transfusion:\n")
          print('There are {} rows and {} columns.\n'.format(rows_and_cols[0], rows_and_cols[1]))

          # Generate an overview of the DataFrame
          transfusion_information = transfusion.info()
          print(transfusion_information)

          # Check whether there are Nan values or not
          print("\nBrief Description Whether DataFrame Object transfusion Consists Any NaN Values:\n", transfusion.isna(

          # Display the first five rows of the DataFrame
          print("\nFollowing Is The Generic Overview Of DataFrame Object transfusion:\n")
          transfusion.head()
```

```
Brief Description Of Dataset transfusion.csv Stored In DataFrame Object transfusion:

There are 748 rows and 5 columns.

<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
None

Brief Description Whether DataFrame Object transfusion Consists Any NaN Values:
Recency (months)                              0
Frequency (times)                             0
Monetary (c.c. blood)                         0
Time (months)                                 0
whether he/she donated blood in March 2007    0
dtype: int64

Following Is The Generic Overview Of DataFrame Object transfusion:
```

Out[3]:

	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

```
# Compute the summary statistics of all columns in the `transfusion` DataFrame
sum_stat_transfusion = transfusion.describe()
print("Summary Statistics Of DataFrame Object transfusion:")
sum_stat_transfusion
```

Summary Statistics Of DataFrame Object transfusion:

Out[4]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
count	748.000000	748.000000	748.000000	748.000000	748.000000
mean	9.506684	5.514706	1378.676471	34.282086	0.237968
std	8.095396	5.839307	1459.826781	24.376714	0.426124
min	0.000000	1.000000	250.000000	2.000000	0.000000
25%	2.750000	2.000000	500.000000	16.000000	0.000000
50%	7.000000	4.000000	1000.000000	28.000000	0.000000
75%	14.000000	7.000000	1750.000000	50.000000	0.000000
max	74.000000	50.000000	12500.000000	98.000000	1.000000

4. Creating target column

We are aiming to predict the value in whether he/she donated blood in March 2007 column. Let'srename this it to target so that it's more convenient to work with.

In [5]:

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

# Display the first five rows of the DataFrame
print("\nFollowing Is The Generic Overview Of DataFrame Object transfusion:\n")
transfusion.head()
```

Following Is The Generic Overview Of DataFrame Object transfusion:

Out[5]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	target
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

In [6]:

```
# Compute the summary statistics of all columns in the `transfusion` DataFrame
sum_stat_transfusion = transfusion.describe()
print("Summary Statistics Of DataFrame Object transfusion:")
sum_stat_transfusion
```

Summary Statistics Of DataFrame Object transfusion:

Out[6]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	target
count	748.000000	748.000000	748.000000	748.000000	748.000000
mean	9.506684	5.514706	1378.676471	34.282086	0.237968
std	8.095396	5.839307	1459.826781	24.376714	0.426124
min	0.000000	1.000000	250.000000	2.000000	0.000000
25%	2.750000	2.000000	500.000000	16.000000	0.000000
50%	7.000000	4.000000	1000.000000	28.000000	0.000000
75%	14.000000	7.000000	1750.000000	50.000000	0.000000
max	74.000000	50.000000	12500.000000	98.000000	1.000000

5. Checking target incidence

We want to predict whether or not the same donor will give blood the next time the vehicle comes to campus.The model for this is a binary classifier, meaning that there are only 2 possible outcomes: 0

- the donor will not give blood 1

- the donor will give blood Target incidence is defined as the number of cases of each individual target value in a dataset. That is, howmany 0s in the target column compared to how many 1s? Target incidence gives us an idea of how balanced(or imbalanced) is our dataset.

```
In [7]: ▶  # Print target incidence proportions, rounding output to 3 decimal places
            transfusion.target.value_counts(normalize=True).round(3)
```

```
Out[7]: 0    0.762
        1    0.238
        Name: target, dtype: float64
```

6. Splitting transfusion into train and test datasets

We'll now use train_test_split() method to split transfusion DataFrame. Target incidence informed us that in our dataset 0 s appear 76% of the time. We want to keep the samestructure in train and test datasets, i.e., both datasets must have 0 target incidence of 76%. This is very easyto do using the train_test_split() method from the scikit learn library - all we need to do isspecify the stratify parameter. In our case, we'll stratify on the target column.

```
In [8]: ▶  # 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(5)
```
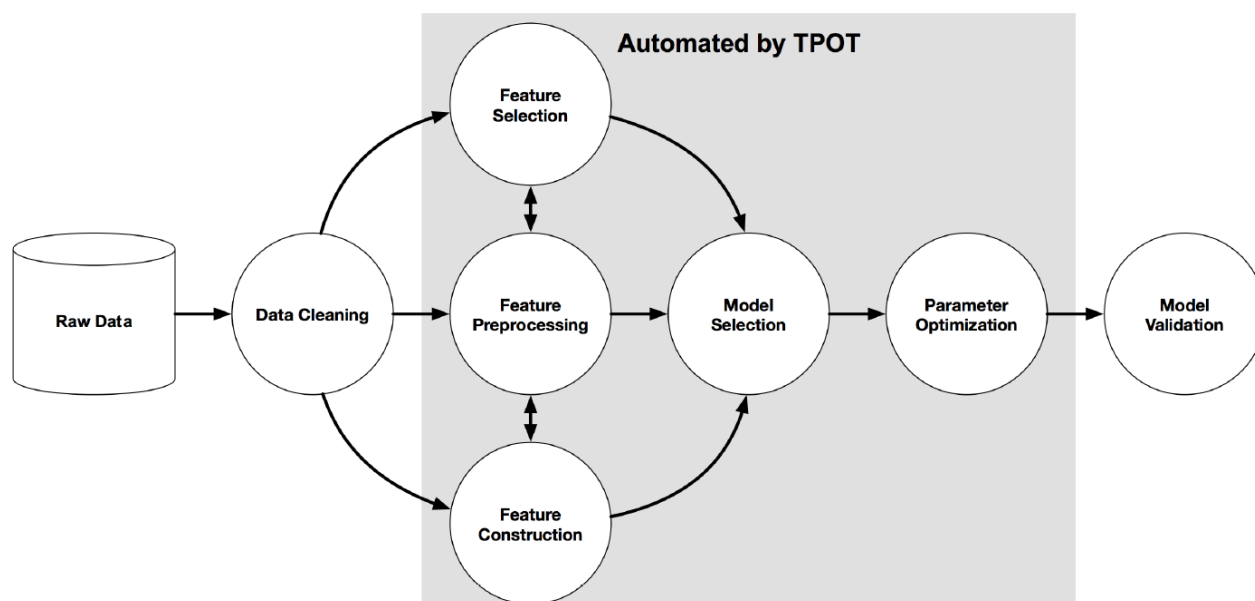
Out[8]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)
334	16	2	500	16
99	5	7	1750	26
116	2	7	1750	46
661	16	2	500	16
154	2	1	250	2

7. Selecting model using TPOT

TPOT (https://github.com/EpistasisLab/tpot (https://github.com/EpistasisLab/tpot)) is a Python Automated Machine Learning tool that optimizesmachine learning pipelines using genetic programming.

TPOT will automatically explore hundreds of possible pipelines to find the best one for our dataset. Note, theoutcome of this search will be a scikit-learn pipeline ([https://scikitlearn (https://scikitlearn)](https://scikitlearn). org/stable/modules/generated/sklearn.pipeline.Pipeline.html) , meaning it will include any pre-processing steps as well as the model. We are using TPOT to help us zero in on one model that we can then explore and optimize further.

In [9]:
```python
# Import TPOTClassifier
from tpot import TPOTClassifier
# Import roc_auc_score
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

TPOT picked LogisticRegression as the best model for our dataset with no pre-processing steps, givingus 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 arerelated in a linear fashion, or can be measured with a linear distance metric. If a feature in our dataset has ahigh variance that's an order of magnitude or more greater than the other features, this could impact themodel'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 beforetraining a model. Let's check the variance to see if such transformation is needed.

In [10]:
```python
# X_train's variance, rounding the output to 3 decimal places
X_train.var().round(3)
```

Out[10]:
```
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. Thismeans that, unless accounted for, this feature may get more weight by the model (i.e., be seen as moreimportant) than any other feature. One way to correct for high variance is to use log normalization.

```
In [11]:    ▶|   # 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
                 X_train_normed.var().round(3)
```

```
Out[11]:   Recency (months)        66.929
           Frequency (times)       33.830
           Time (months)          611.147
           monetary_log             0.837
           dtype: float64
```

10. Training the logistic regression model module of linear_model imported from sklearn library

The variance looks much better now. Notice that now Time (months) has the largest variance, but it's notthe orders of magnitude (https://en.wikipedia.org/wiki/Order_of_magnitude (https://en.wikipedia.org/wiki/Order_of_magnitude)) higher than the rest of thevariables, so we'll leave it as is. We are now ready to train the linear regression model.

```
In [12]:    ▶|   # 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
```

11. Conclusion

The demand for blood fluctuates throughout the year. As one prominent (https://www.kjrh.com/news/localnews/ (https://www.kjrh.com/news/localnews/) red-cross-in-blood-donation-crisis) 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 timeand therefore saving more lives. In this notebook, we explored automatic model selection using TPOT and AUC score we got was 0.7850.This is better than simply choosing 0 all the time (the target incidence suggests that such a model wouldhave 76% success rate). We then log normalized our training data and improved the AUC score by 0.5%. Inthe field of machine learning, even small improvements in accuracy can be important, depending on thepurpose. Another benefit of using logistic regression model is that it is interpretable. We can analyze how much of thevariance in the response variable (target) can be explained by other variables in our dataset.

```
In [13]:    ▶|   # 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[13]:   [('logreg', 0.7890972663699937), ('tpot', 0.7637476160203432)]
```