Data Glacier

Data Science Intern at Data Glacier

Week 4: Deployment on Flask

Name: Aparna Virupakshi

Batch Code: LISUM34

Date: 28 June 2024

Submitted to: Data Glacier

Table of Contents:

1. Introduction

This project aims to deploy a Logistic Regression model using Flask to predict blood donation. The model uses features such as recency, frequency, monetary, and time to make predictions.

2. Data Information

The dataset contains information about blood donations collected from a donor database.

The   features include:

- **Recency**: Months since the last donation
- **Frequency**: Total number of donations
- **Monetary**: Total blood donated in c.c.
- **Time**: Months since the first donation

Table 2.1: Dataset Information

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

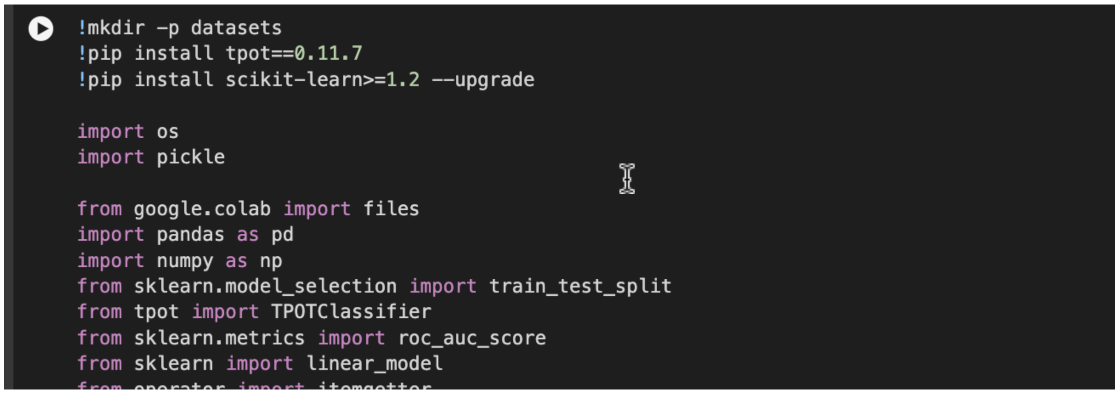
2.1.1 Attribute Information

- **Recency (months)**: This represents the number of months since the donor's most recent donation.
- **Frequency (times)**: This indicates the total number of times the donor has donated blood.
- **Monetary (c.c.)**: This represents the total blood donated in cubic centimeters (c.c.).
- **Time (months)**: This is the number of months since the donor's first donation.
- **Target (1 or 0)**: This binary attribute indicates whether the donor donated blood within a given time window (1 if they donated, 0 if they did not). This attribute is the target variable for the prediction model.

3. Building a Model

3.1.1 Import Required Libraries and Dataset

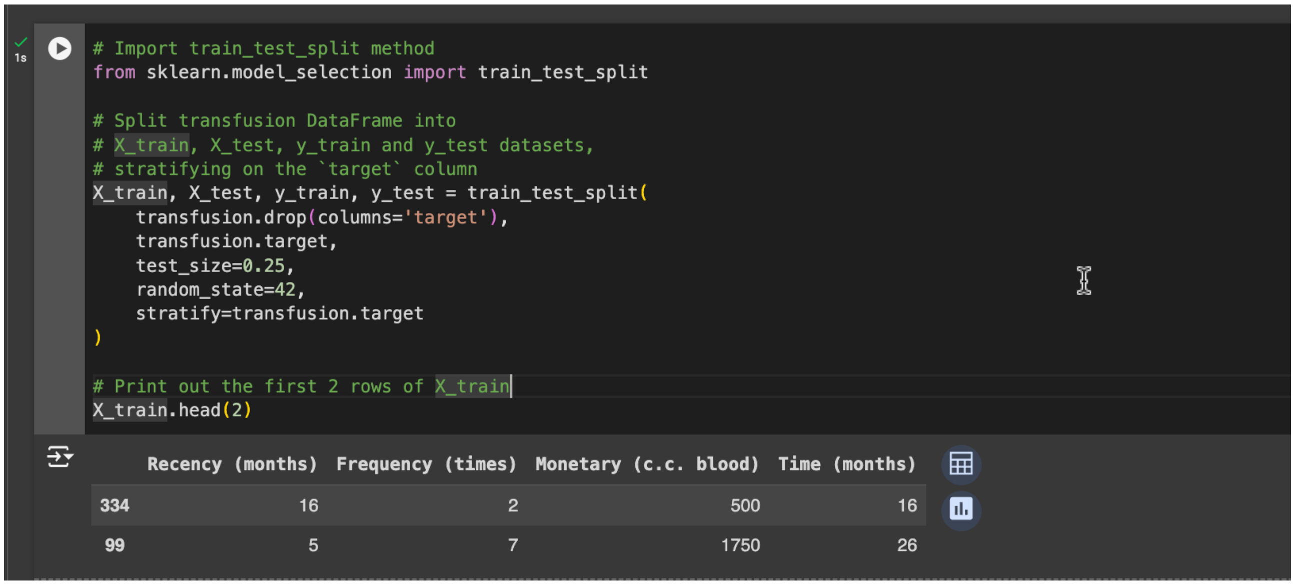
In this part, we import libraries and the dataset containing information about blood donations.



```
!mkdir -p datasets
!pip install tpot==0.11.7
!pip install scikit-learn>=1.2 --upgrade

import os
import pickle

from google.colab import files
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from tpot import TPOTClassifier
from sklearn.metrics import roc_auc_score
from sklearn import linear_model
from operator import itemgetter
```

3.1.1 Data Preprocessing

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

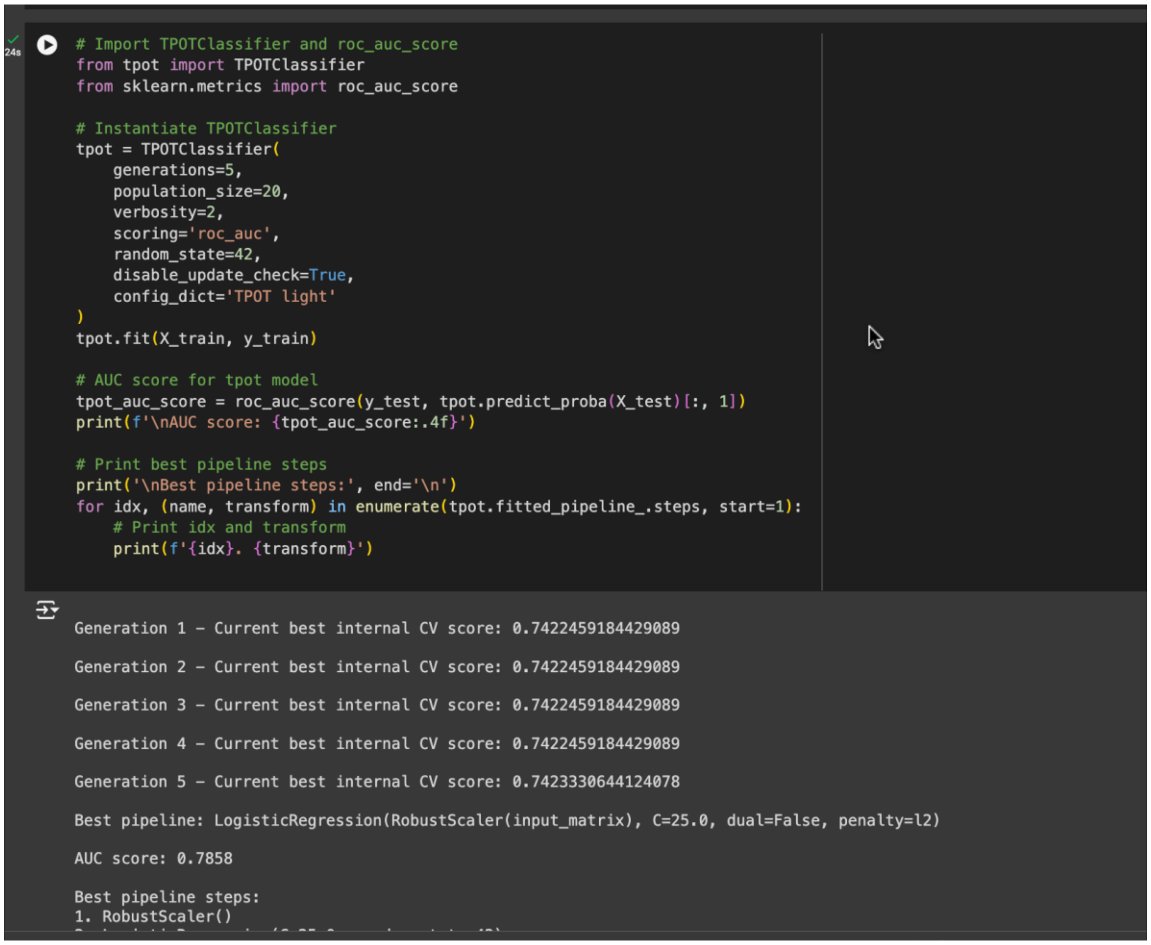


```
# Import train_test_split method
from sklearn.model_selection import train_test_split

# 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)
```

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

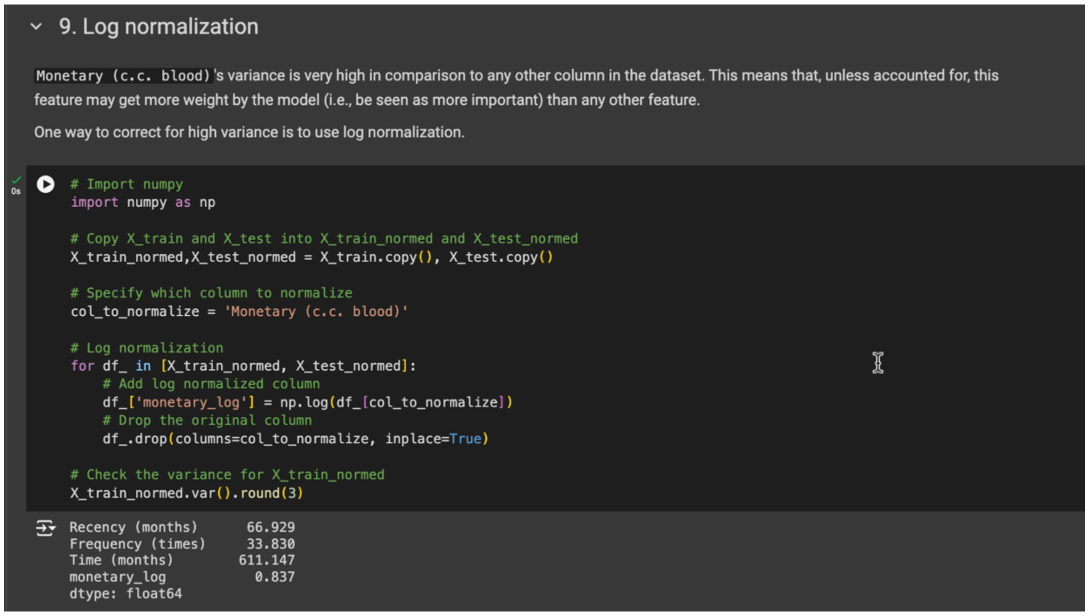
3.1.2 Build Model

We implement a Logistic Regression model using scikit-learn. The model is trained on the training set and evaluated on the test set.



```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: 0.7423330644124078

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

AUC score: 0.7858

Best pipeline steps:
1. RobustScaler()
```



9. Log normalization

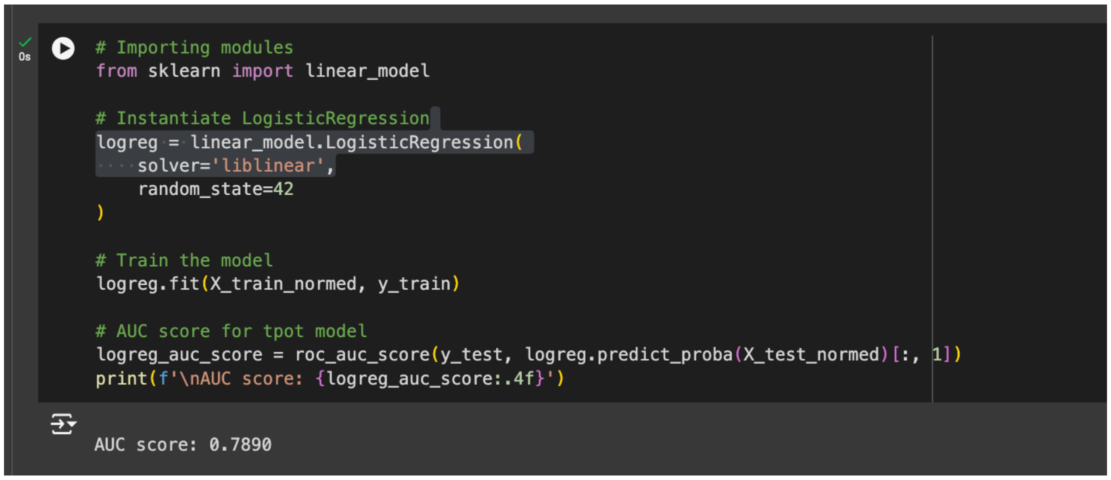
`Monetary (c.c. blood)`'s variance is very high in comparison to any other column in the dataset. This means that, unless accounted for, this feature may get more weight by the model (i.e., be seen as more important) than any other feature.

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

```python
# 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
```

The variance looks much better now. Notice that now Time (months) has the largest variance, but it's not the orders of magnitude higher than the rest of the variables, so we'll leave it as is. We are now ready to train the logistic regression model.

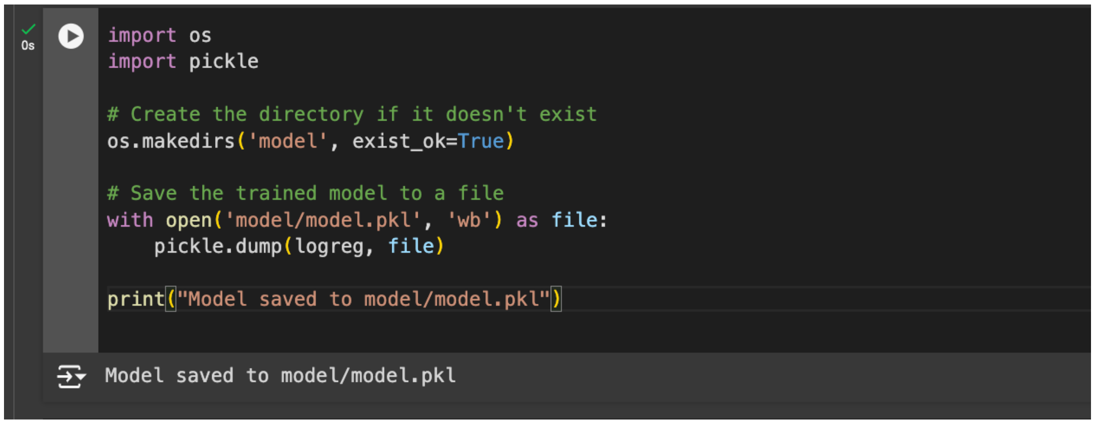


```
# Importing modules
from sklearn import linear_model

# 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
```

3.1.3 Save the Model

The trained model is saved using the `pickle` module.



```
import os
import pickle

# Create the directory if it doesn't exist
os.makedirs('model', exist_ok=True)

# Save the trained model to a file
with open('model/model.pkl', 'wb') as file:
    pickle.dump(logreg, file)

print("Model saved to model/model.pkl")
```

```
Model saved to model/model.pkl
```

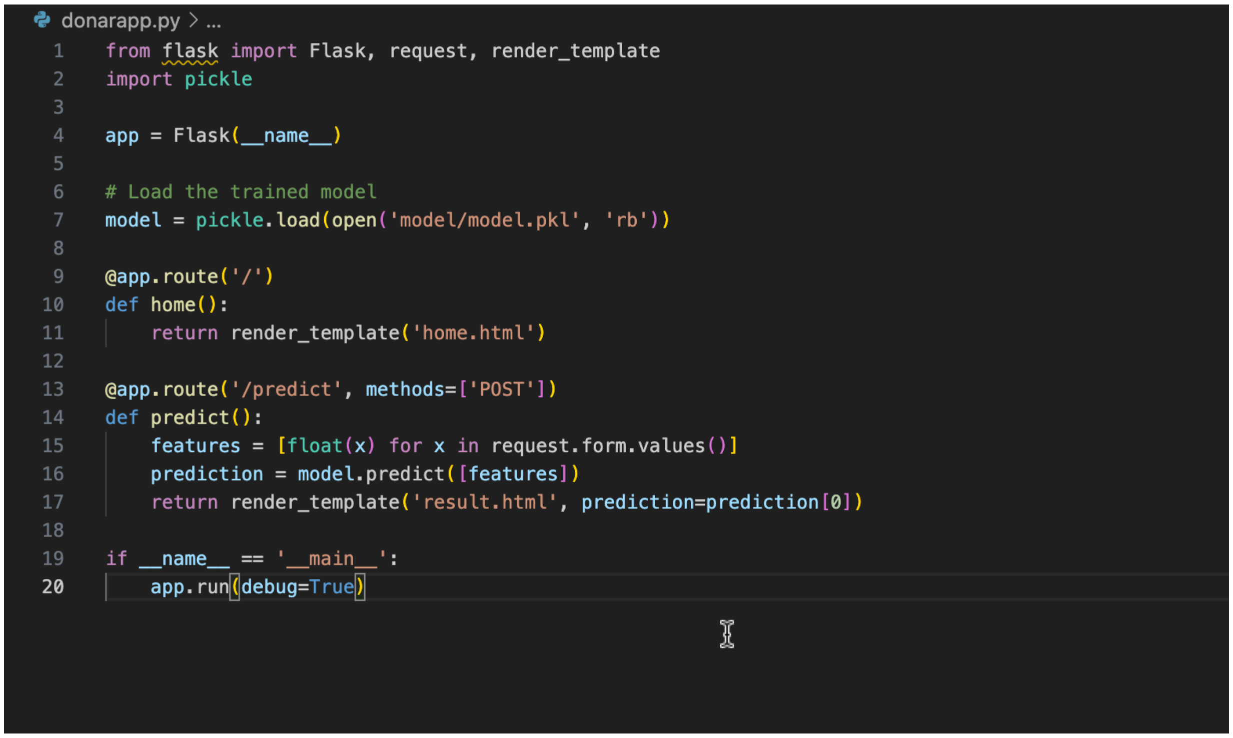
4. Turning Model into Web Application

We develop a web application that allows users to input features and get predictions on

Blood donation.

⇒ 4.1 donarapp.py :

The donarapp`.py` file contains the main code for running the Flask web application. It includes routes for the home page and prediction.



donarapp.py > ...

```python
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

# Load the trained model
model = pickle.load(open('model/model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict', methods=['POST'])
def predict():
    features = [float(x) for x in request.form.values()]
    prediction = model.predict([features])
    return render_template('result.html', prediction=prediction[0])

if __name__ == '__main__':
    app.run(debug=True)
```

- **Import Libraries**:
    - `Flask`, `request`, and `render_template` from the `flask` library are used to handle web requests and render HTML templates.
    - `pickle` is used to load the saved machine learning model.
    - `numpy` is used to handle numerical operations and arrays.

- **Initialize Flask Application**:
    - The Flask application is initialized by creating an instance of the `Flask` class.

- **Load the Trained Model**:

  - The trained Logistic Regression model is loaded from the `model/model.pkl` file using the `pickle.load` function.

- **Home Page Route**:

  - The route for the home page (`/`) is defined using the `@app.route('/')` decorator.
  - The `home` function renders the `home.html` template.

- **Prediction Route**:

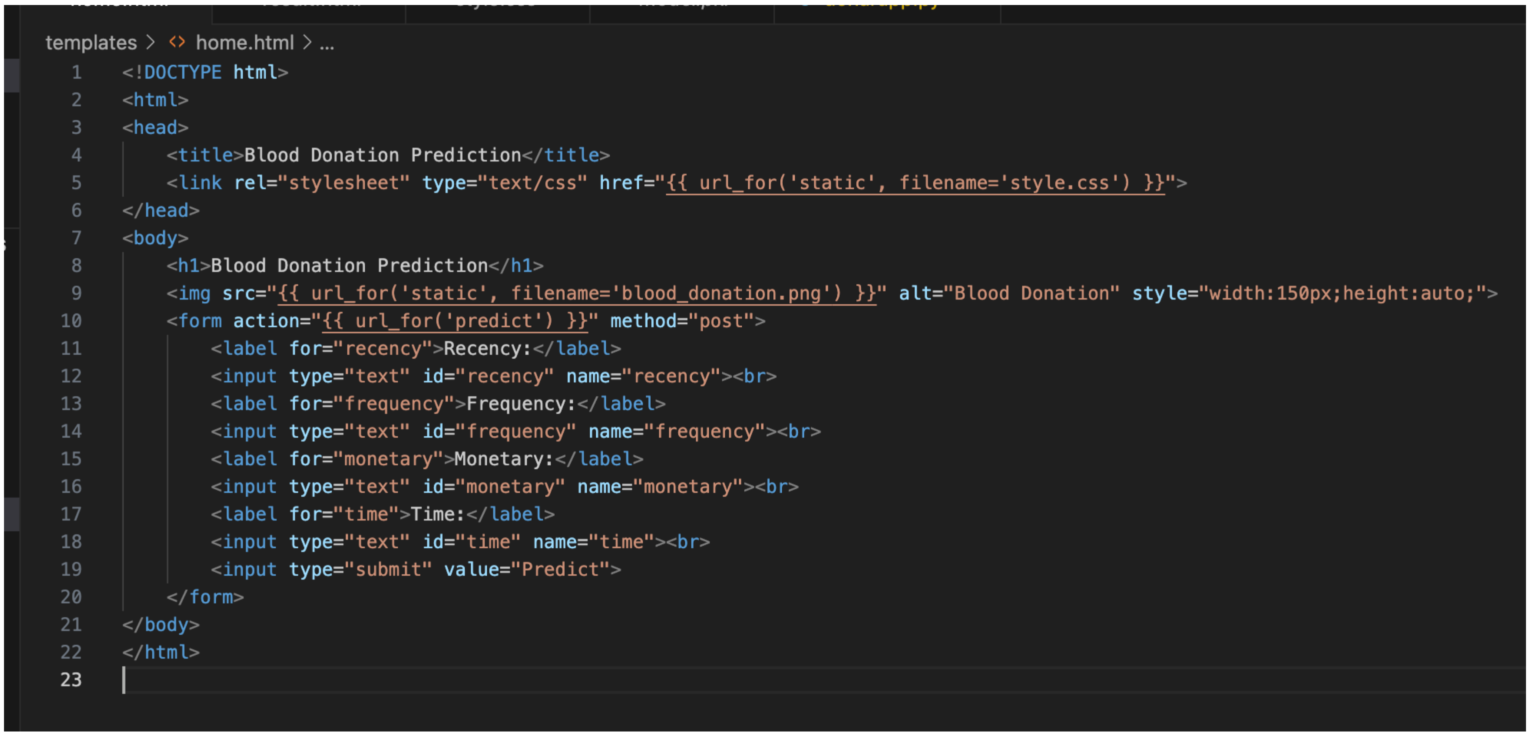
  - The route for handling form submission and making predictions (`/predict`) is defined using the `@app.route('/predict', methods=['POST'])` decorator.
  - The `predict` function:
    - Extracts the input features from the form submission.
    - Converts the input features to a NumPy array and reshapes it for the model.
    - Uses the loaded model to make a prediction.
    - Renders the `result.html` template with the prediction result.

- **Run the Flask Application**:

  - The `if __name__ == '__main__':` block ensures that the Flask application runs when the script is executed directly.
  - The `app.run(debug=True)` line runs the Flask application in debug mode, allowing for easier debugging during development.

⇒ 4.2 Home.html :

The following are the contents of the *`home.html`* file that will render a text form where a user can enter a message.



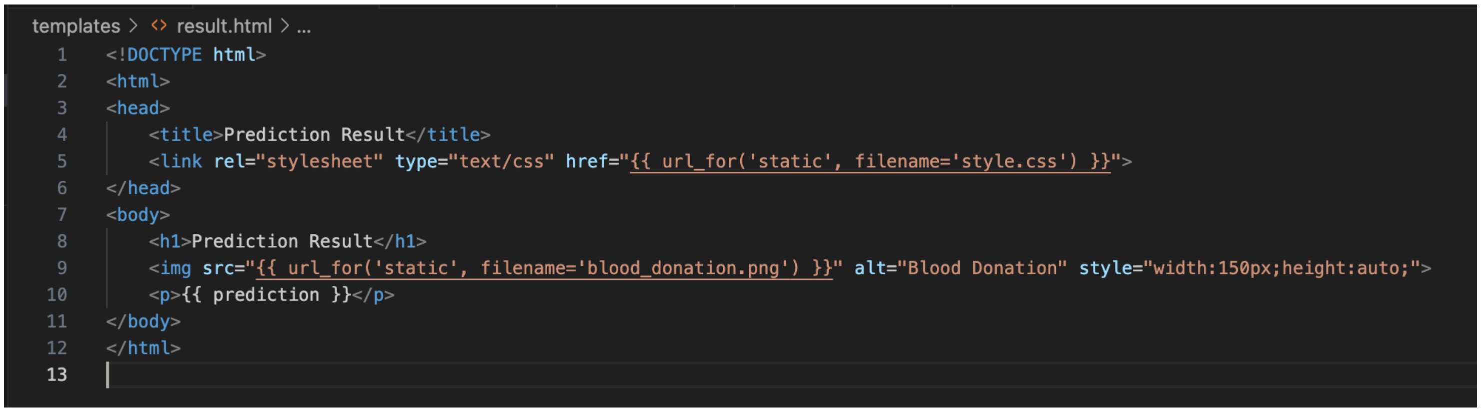
```html
<!DOCTYPE html>
<html>
<head>
    <title>Blood Donation Prediction</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Blood Donation Prediction</h1>
    <img src="{{ url_for('static', filename='blood_donation.png') }}" alt="Blood Donation" style="width:150px;height:auto;">
    <form action="{{ url_for('predict') }}" method="post">
        <label for="recency">Recency:</label>
        <input type="text" id="recency" name="recency"><br>
        <label for="frequency">Frequency:</label>
        <input type="text" id="frequency" name="frequency"><br>
        <label for="monetary">Monetary:</label>
        <input type="text" id="monetary" name="monetary"><br>
        <label for="time">Time:</label>
        <input type="text" id="time" name="time"><br>
        <input type="submit" value="Predict">
    </form>
</body>
</html>
```

Style.css :

In the `<head>` section of the `home.html` file, we include a link to the `styles.css` file. This CSS file controls the appearance and layout of the HTML elements on the web page. To ensure Flask can locate and apply the CSS styles correctly, the `styles.css` file must be placed in a folder named `static`. The `static` folder is Flask's designated directory for storing static files, such as CSS, JavaScript, and images.

4.2.1 Result.html

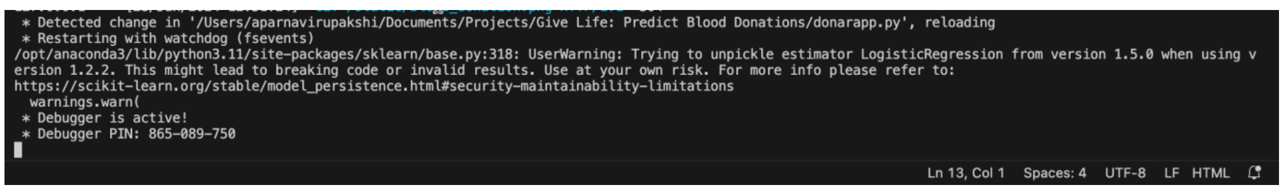
The prediction result is passed from the Flask backend to the `result.html` template using the `render_template` function. The result is then accessed using Jinja2 templating syntax `{{ prediction }}`.



```html
<!DOCTYPE html>
<html>
<head>
    <title>Prediction Result</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Prediction Result</h1>
    <img src="{{ url_for('static', filename='blood_donation.png') }}" alt="Blood Donation" style="width:150px;height:auto;">
    <p>{{ prediction }}</p>
</body>
</html>
```

5. Running Procedure

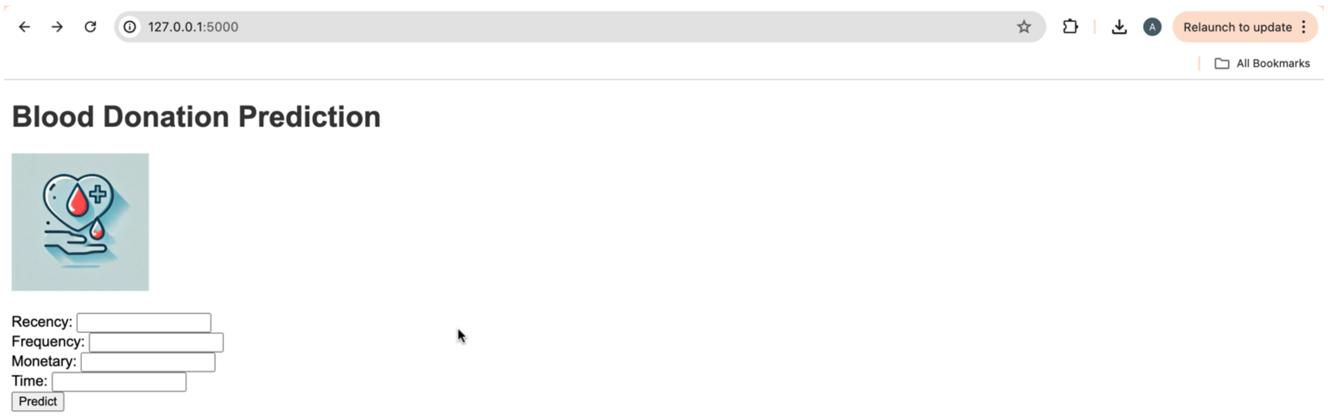
Once we have done all of the above, we can start running the API by using terminal:



```
 * Detected change in '/Users/aparnavirupakshi/Documents/Projects/Give Life: Predict Blood Donations/donarapp.py', reloading
 * Restarting with watchdog (fsevents)
/opt/anaconda3/lib/python3.11/site-packages/sklearn/base.py:318: UserWarning: Trying to unpickle estimator LogisticRegression from version 1.5.0 when using v
ersion 1.2.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Debugger is active!
 * Debugger PIN: 865-089-750
```

Now we could open a web browser and navigate to http://127.0.0.1:5000/,

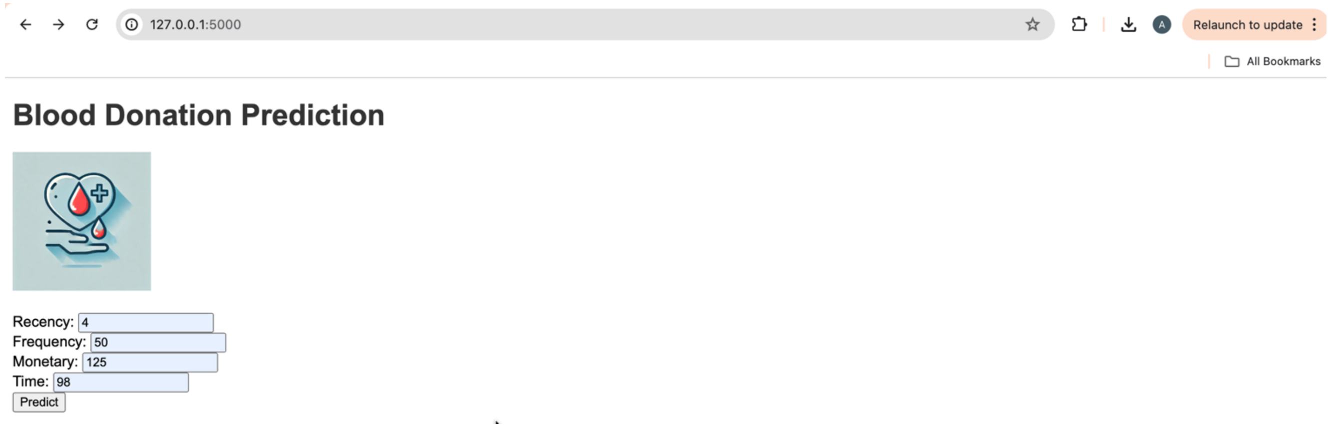
So that we can see a simple website with containing information.



Blood Donation Prediction

Recency:
Frequency:
Monetary:
Time:
Predict

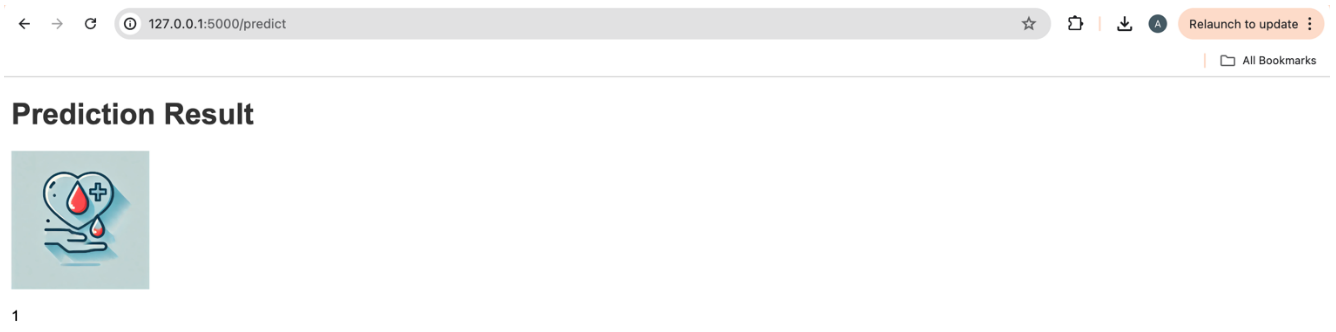
We can input the values accordingly


127.0.0.1:5000

Blood Donation Prediction

Recency: 4
Frequency: 50
Monetary: 125
Time: 98

After entering the input click the predict button now, we can the result of our input.


127.0.0.1:5000/predict

Prediction Result

1