EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF ALGORITHMS AND THEIR APPLICATIONS

An Algorithm to Minimize International Transaction Fees using Graph Theory

Case study on the World Bank's data on Bilateral Remittance Matrices.

Supervisor:

Dr. Szabó László

Associate Professor

Author:

Mohamed Elsaadany

Computer Science BSc

Budapest, 2021

Contents

Chapter 1

Introduction

A money transaction is a process where a sender sends an amount X of money to a receiver. This does not mean that the cash must physically travel from the sender account to the receiver's one, but it means the value of the sender's account would decrease by an amount X and the receiver's account would increase by the same amount X. The transaction shall involve at least three components; sender, receiver, and service provider which is the party that performs that transaction between the sender and receiver such as banks. In addition, other parties can be involved in the process. For example, the card issuer if the transaction is performed using a credit/debit card. Thus, having such transaction services providers involved, a fee must be paid for each transaction made through them.

A local transaction that occurs in the same country in the same currency always has lower fees than an international one as the currency conversion adds up to its fees that are paid to the merchant. In a network of transactions among several countries, if there exists an approach to transform foreign transactions into local transactions avoiding the higher fees of foreign transactions, it will be very beneficial in the money transfer industry either for individuals or companies and even for banks and countries' governments.

My thesis work is an approach to minimize the foreign transaction fees in a network of foreign transactions using graph theory and network flows concepts and algorithms. I am trying to develop an analysis model that can receive a network of foreign transactions and analyses it to check if there exist set of foreign transactions to be replaced by local transactions without affecting the money balances of all the parties of the network. My main topic is developing and documenting the model and creating a demo web application that can perform the model procedure on a specific type of data and store the outcomes. Additionally, to test the model on a network that includes a large number of nodes. I am adding a simple case study on the world bank remittances data where I run the model through the application and generate a report of the analysis as an example of what the algorithm can do.

The web application is developed in Python Django framework for the backend and HTML, CSS, and JavaScript for the frontend. I have chosen Bootstrap and Bootswatch themes to develop a nice responsive user interface. Additionally, I used Highcharts JavaScript library to visualize the output data.

Outline

Chapter 2 Analysis model contains:

- Documentation of the Analysis model approach defining the general idea of the model. See section 2.1.
- Description of the problem that the algorithm tries to solve. See section 2.2.
- A detailed explanation of the algorithm and its inputs, steps, and output. See section 2.3.

Chapter 3 User Documentation contains:

- An overview of the web application “Analysaction”. See section 3.1.
- Description of the running environment of the application. See section 3.1.
- Step by step explanation of application installation procedure. See section 3.2.
- A detailed guide of how to use the application as an end-user and how to create reports. See section 3.3.

Chapter 4 Developer Documentation contains:

- Short description of the application specifications. See section 4.1.
- Detailed illustration of the technologies used to develop the web application/ see sections 4.2 and 4.3.
- full documentation of the web application implementation code for backend and front-end and how the frameworks were integrated. See section 4.4.

Chapter 5 Case study contains:

- Short description of the data source for the case study. See section 5.1.
- Illustration of the input data and the generated report showing the success of running the model on a large number of nodes. See section 5.2.

Chapter 2

Analysis Model

2.1 Approach

The model is trying to find a method to replace as many foreign transactions as possible with local ones while keeping the money flow balanced resulting less transaction fees for all parties in the network.

Given a set of foreign transactions as each transaction is defined by a country A, country B and an amount X to be sent from A to B. As there is a flow of sent and received amounts in each country in this network of transactions, we can introduce a model that finds the cyclic flow where each country within a cycle sends an amount X and receives the same amount X in the cycle.

Case 1 – direct cycle flow match: (two countries)

If an amount X is to be sent from country A to country B and the same amount X is to be sent from country B to country A. Then sender and receiver in each country can exchange the amount X locally and cancel the two foreign transactions.

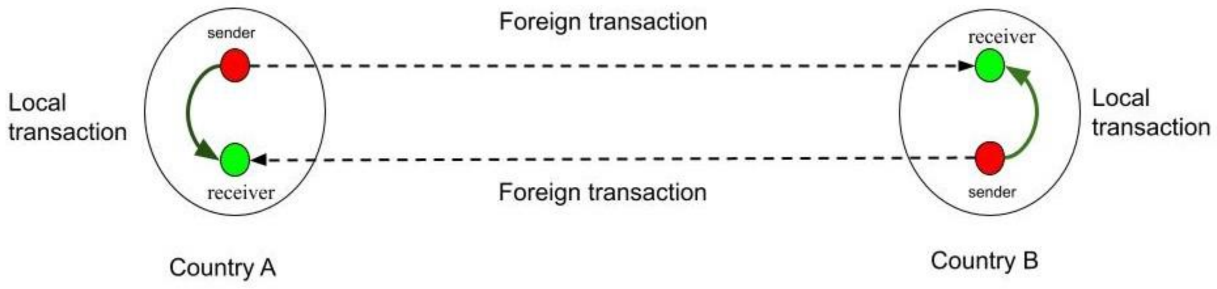


Figure 1 Direct cycle flow match between two countries

Case 2 – indirect cycle flow match: (more than two countries)

If country B does not have an amount X to be sent to country A as well to create a direct match. but, in the network, there exist country C that sends an amount X to country A and receives an amount X from country B so that we would have three local sender/receiver pairs in the three countries resulting in three transactions occurring locally between the sender and receiver in each country without foreign transaction fees.

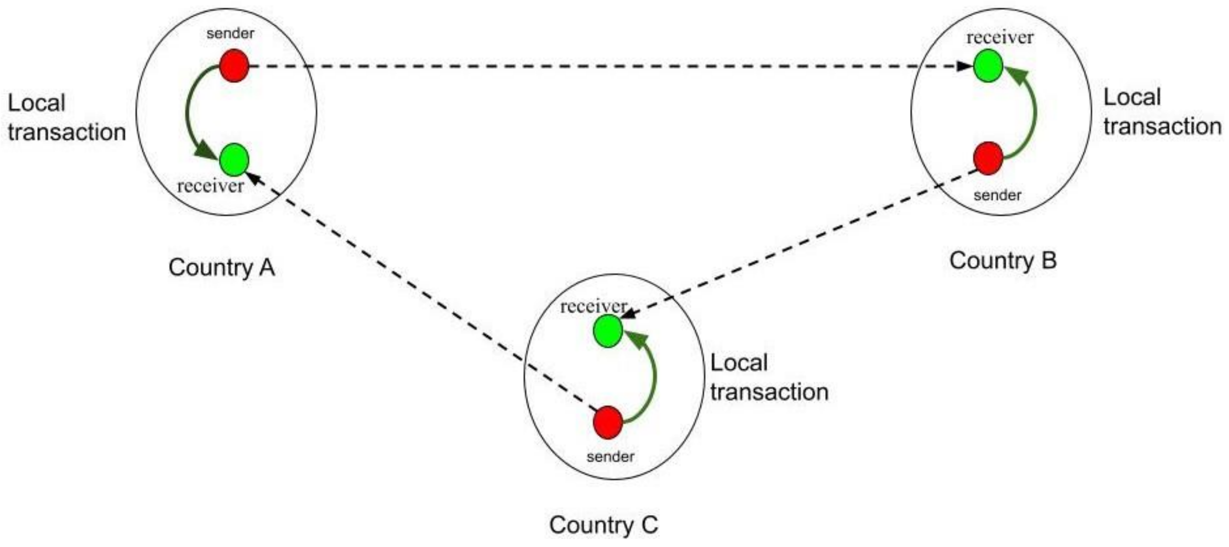


Figure 2 Indirect cycle flow match between more than two countries

Case 3 – General case:

If country C condition does not apply, we keep searching for other countries that can close a longer cycle if any.

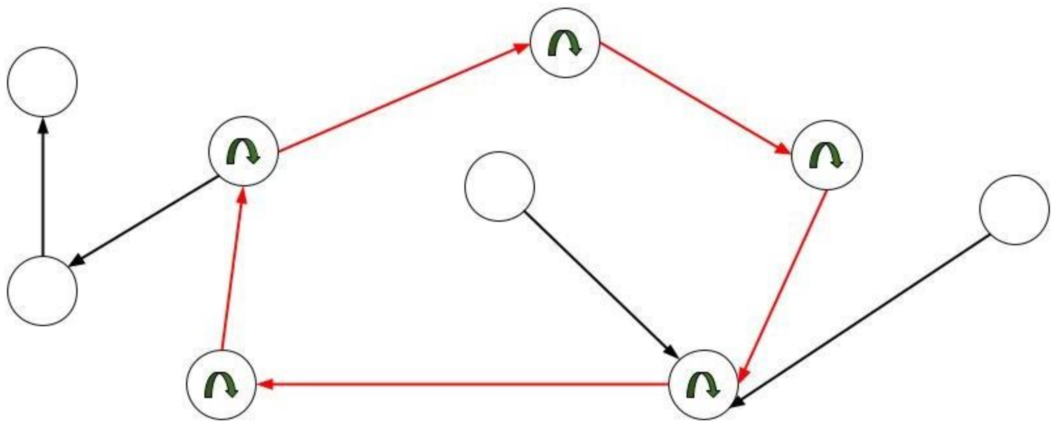


Figure 3 cycle flow match among more than three countries

2.2 Problem description

The problem is a special case of the maximum flow problem [1] in networks. In a regular maximum flow problem:

We let a directed graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. each vertex $\boldsymbol{v} \in \boldsymbol{V}$ represents a country, and each edge $\boldsymbol{e} \in \boldsymbol{E}$ represents a transaction. $\boldsymbol{e}(\boldsymbol{u}, \boldsymbol{v})$ defines sending from vertex $\boldsymbol{u}$ to vertex $\boldsymbol{v}$, and $\boldsymbol{e}(\boldsymbol{u}, \boldsymbol{v})$ capacity $\boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v})$ represents the transaction amount.

- The flow passes from the source $\boldsymbol{s}$ and the sink $\boldsymbol{t}$.

- The flow in one edge is less than or equal to its capacity.

  For any $\boldsymbol{u, v} \;:\; \boldsymbol{there\ exists\ flow(u,v)\ where\ 0 \leq flow(u,v) \leq C(u,v)}$.

- The sum of the flows entering a node must equal the sum of the flows exiting that node, except for the source and the sink.

  For any $\boldsymbol{v \in V \setminus \{s,t\} : \sum_{u:(u,v)\in E} flow(u,v) = \sum_{u:(v,u)\in E} flow(v,u)}$

- The **value of flow** is the amount of flow passing from the source to the sink. Formally for a flow $flow : E \to \mathbb{R}^+$ is:

$$|flow| = \sum_{v:(s,v)} flow(s,v)$$

But this gets the flow from one country to another in the network of transactions and this is not what we seek for. In our model, we have a condition that this flow path starts from a vertex and gets back to it with the same amount as explained in the approach section. Thus, the special case shall hold, **if applicable**, this condition:

$$\boldsymbol{sink\ vertex = source\ vertex}.$$

This means the flow circulates from a vertex to reach itself again forming what we can call a **cycle flow**. This implies **that the graph must include at least one cycle**. However, the maximum flow problem does not satisfy this condition. On the other hand, we can use this condition to describe our problem definition. In the algorithm section, I will demonstrate how we can make a maximum flow problem satisfy it.

problem definition:

Given $G = (V, E)$ is a weighted directed graph that contains at least one cycle, try finding a cycle C that an amount X can flow from vertex $v \in C$ to the same vertex v where the sum of the flows' values on each edge in C is maximum.

2.3 Algorithm

In this section, I will show the algorithm and how to come over the "$Sink = source$" condition. First, let me define some terms to ease my way of explaining the steps.

cycle flow: can be defined as the set of edges that have a positive flow value in the residual graph resulting from calculating the maximum flow of a directed graph G where the sink vertex is the same as the source vertex, if applicable.

cycle flow value: the sum of these positive flows in the resulting residual graph.

Satisfying “Source = sink” condition

Let $G1 = (V, E)$, a weighted cyclic directed graph having the edges shown in the following figure:

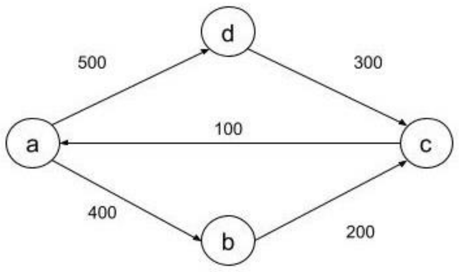


Figure 4 Satisfying "Source = Sink" condition: G1

Since we cannot perform a maximum flow from one vertex to itself, we need to imitate it by modifying the graph:

- Add a node $\boldsymbol{S}$ to $G1$.
- Choose one edge to imitate using the new vertex S. for instance $(c, \boldsymbol{a}, \mathbf{100})$.
- Add an edge $(\boldsymbol{c}, \boldsymbol{S}, \mathbf{100})$ to mimic the chosen edge.

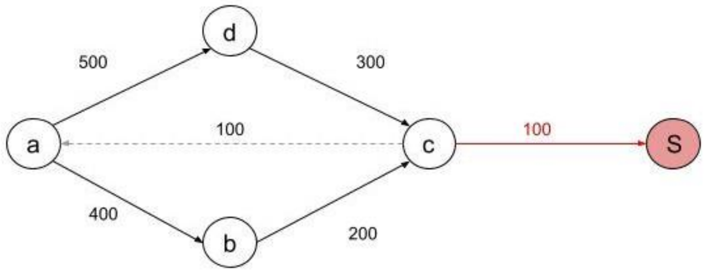


Figure 5 Satisfying "Source = Sink" condition: Step 1

- Get a maximum flow from $(source\ =\ \boldsymbol{a})$ to $(sink\ =\ \boldsymbol{S})$.

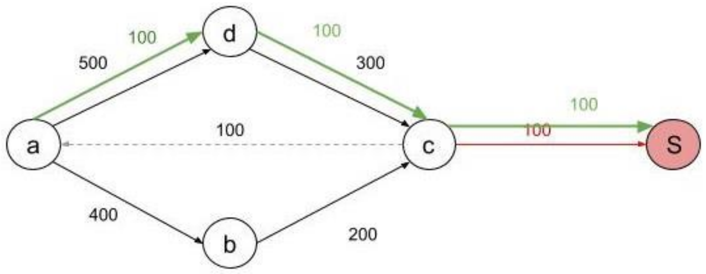


Figure 6 Satisfying "Source = Sink" condition: Step 2

- “$\boldsymbol{S}$” is a mirror to “$\boldsymbol{a}$” in this case and that means the flow augmented path. $[(\boldsymbol{a}, \boldsymbol{d}), (\boldsymbol{d}, \boldsymbol{c}), (\boldsymbol{c}, \boldsymbol{S})]$ is equivalent to $[(\boldsymbol{a}, \boldsymbol{d}), (\boldsymbol{d}, \boldsymbol{c}), (\boldsymbol{c}, \boldsymbol{a})]$. As if we performed a maximum flow from $\boldsymbol{source}\ ==\ \boldsymbol{sink}\ ==\ \boldsymbol{a}$.

However, this gets us the cycle flow and cycle flow value for checking the edge $(\boldsymbol{a}, \boldsymbol{c}, \mathbf{100})$. Our job is to check the maximum cycle flow value to meet our problem definition. Thus, we need to check each edge in the graph the same way to choose the highest cycle flow value as a solution. In a larger graph, with a huge number of edges, it will be very costly to check every single edge.

We can optimize that issue by dropping the edges of the previous cycle flow resulting from the last checked edge. For instance, given the following $G2$ in this figure:

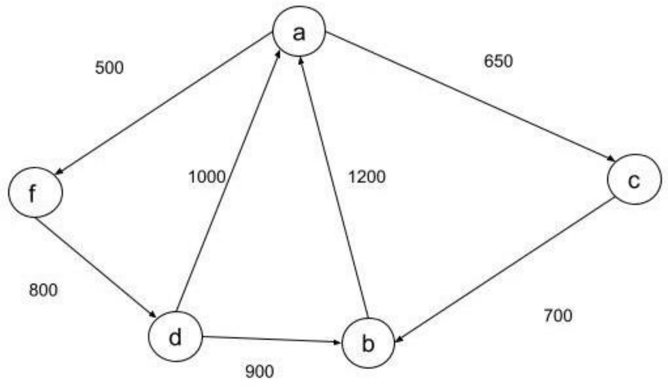


Figure 7 G2: Edge check optimization

$G2$ is a cyclic weighted directed graph. The graph has more than one cycle and some edges share more than one cycle. To find the maximum we need to check the cycle flow value by choosing edges as we did in the previous procedure. Instead of checking every edge in the graph, we choose an arbitrary edge, for example, $(\boldsymbol{a}, \boldsymbol{f}\ , \mathbf{500})$. The resulting cycle flow will be:

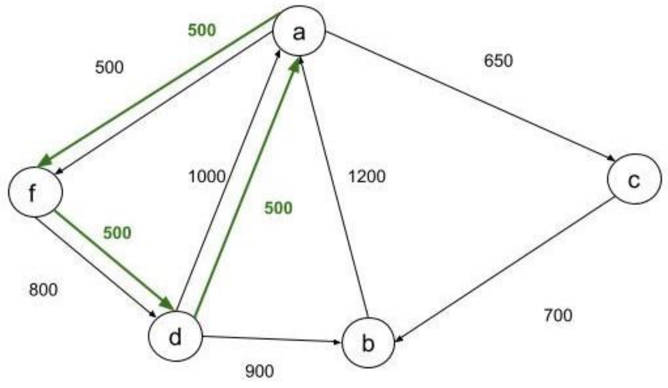


Figure 8 G2: first cycle flow

The first cycle flow is $[(\boldsymbol{a}, \boldsymbol{f}), (\boldsymbol{f}, \boldsymbol{d}), (\boldsymbol{d}, \boldsymbol{a})]$ with flow cycle value **1500** by checking edge $(\boldsymbol{a}, \boldsymbol{f}, \mathbf{500})$**.** Now we will skip the rest of the edges included in this cycle flow: $(\boldsymbol{f}, \boldsymbol{d})$ and $(\boldsymbol{d}, \boldsymbol{a})$**.** and choose a new arbitrary edge to check. Let us take $(\boldsymbol{b}, \boldsymbol{a}, \mathbf{1200})$. The resulting cycle flow is:

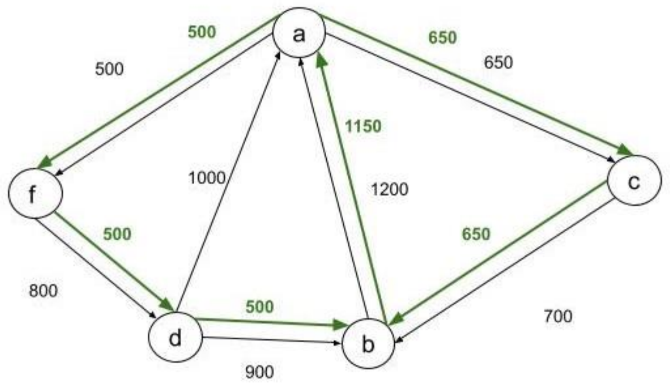


Figure 9 G2: second cycle flow

The second cycle flow is $[(\boldsymbol{a},\boldsymbol{f}),(\boldsymbol{f},\boldsymbol{d}),(\boldsymbol{d},\boldsymbol{b}),(\boldsymbol{b},\boldsymbol{a}),(\boldsymbol{a},\boldsymbol{c}),(\boldsymbol{c},\boldsymbol{b})]$ with flow cycle value of **3950** by checking edge $(\boldsymbol{b},\boldsymbol{a},\mathbf{1200})$**.** Here we can notice there are no more edges left in the graph to check. Then our solution is choosing the second cycle flow as it has the maximum cycle flow value of **3950**.

Important Note: Cycle flow is not exactly the normal directed graph cycle. However, it can be a combination of one or more cycles. By checking the second cycle flow we find it includes 2 cycles sharing the edge $(\boldsymbol{b},\boldsymbol{a},\mathbf{1150})$ and they are:

- $[(\boldsymbol{a},\boldsymbol{f},\mathbf{500}),(\boldsymbol{f},\boldsymbol{d},\mathbf{500}),(\boldsymbol{d},\boldsymbol{b},\mathbf{500}),(\boldsymbol{b},\boldsymbol{a},\mathbf{500})]$
- $[(\boldsymbol{a},\boldsymbol{c},\mathbf{650}),(\boldsymbol{c},\boldsymbol{b},\mathbf{650}),(\boldsymbol{b},\boldsymbol{a},\mathbf{650})]$

That is the reason why I needed to define the cycle flow because it is not just a normal cycle.

Algorithm

In the previous section, we could find the maximum cycle flow in a cyclic weighted directed graph. The algorithm aims to find the total amount of foreign transactions that can be replaced by local ones resulting less transaction fees. To do that, we need to recursively find the maximum cycle flow in the graph by eliminating the values of the flows of the last chosen maximum cycle flow.

Input:

Let the set of foreign transactions be represented as a directed graph $\boldsymbol{G} = (\boldsymbol{V},\boldsymbol{E})$. each vertex $\boldsymbol{v} \in \boldsymbol{V}$ represents a country, and each edge $\boldsymbol{e} \in \boldsymbol{E}$ represents a transaction. $\boldsymbol{e}(\boldsymbol{u},\boldsymbol{v})$ defines sending from vertex $\boldsymbol{u}$ to vertex $\boldsymbol{v}$, and $\boldsymbol{e}(\boldsymbol{u},\boldsymbol{v})$ capacity $\boldsymbol{C}(\boldsymbol{u},\boldsymbol{v})$ represents the transaction amount. And $\boldsymbol{F}(\boldsymbol{u},\boldsymbol{v})$ represents the flow that passes from $\boldsymbol{u}$ to $\boldsymbol{v}$.

Steps and pseudocode:

The algorithm uses Python **NetworkX** [2] algorithms to perform the analysis:

networkx.algorithms.components.kosaraju_strongly_connected_components [3]**:** to find the strongly connected components in G. Cost $O(V+E)$.

networkx.algorithms.flow.dinitz [4]**:** to get the cycle flow in strongly connected components between two vertices. Cost $O(V^2E)$.

First, break down the graph into strongly connected components where the number of vertices is greater than 1. Finding these strongly connected graphs has two benefits:

1. Assure that there is at least one cycle in the strongly connected component.
2. Finding the maximum cycle flow value will check only the strongly connected component edges and not the whole graph edges which optimizes the algorithm cost.

Finding maximum cycle flow in a strongly connected component:

Let $\boldsymbol{SCG}$ a strongly connected component in $\boldsymbol{G}$.

```
#SCG: a strongly connected component.
## find_max_cycle_flow(SCG)##
list cycle_flows := []
maximum_cycle_flow := None
list done_edges := []
for each edge e(u,v) in SCG:
    if e not in done_edges:
        Add node S to SCG.
        add edge e(u,S) where C(u,S) = C(u,v)
        list cycle_flow := []
        find the maximum flow with nx.algorithms.flow.dinitz(SCG, u, S)
        calculate the cycle flow value from the output_residual_graph.
        cycle_flow = [cycle flow value, output_residual_graph, e(u,v)]
        cycle_flows.append[cycle_flow]
        done.append[filter(out_put_residual_graph.edges) where flow(edge)>0]
        remove edge e(u,S)
        remove node S
maximum_cycle_flow = find max(cycle_flows) where cycle flow value is maximum.
return maximum_cycle_flow
```

Figure 10 pseudocode: find maximum cycle flow in SCG.

Update strongly connected component after finding a maximum cycle flow:

Here we update the strongly connected component SCG by eliminating the **flows** resulting in the **maximum cycle flow** edges from the corresponding **capacities** in SCG**.** If any edge in SCG reaches capacity = 0. Then, we remove this edge from SCG and it is considered a foreign transaction that is eliminated completely and has been replaced by a local one.

```
#SCG: strongly connected graph
#cycle_flow: list of edges of last maximum cycle flow calculated from SCG
#finished_edges : edges to be removed as it has no capacity anymore.
##updade_SCG(SCG, cycle_flow)##
finished_edges := []
for e(u,v) in SCG.edges:
    C(u,v) from SCG -= F(u,v) from cycle_flow
    if C(u,v) from SCG == 0:
        finished_edges.append(e(u,v))

for e in finished_edges:
    SCG.remove_edge(e)
return SCG
```

Figure 11 pseudocode: update SCG after finding a maximum flow cycle.

Finding the total amount of foreign transactions that can be reduced:

The final step is to find the total amount of foreign transactions to be replaced by local transactions in the whole graph. And to do that we need to apply these steps:

- Find the first maximum cycle flow in the graph.
- Update the graph by subtracting the flows in maximum cycle flow from the graph and delete the 0 capacity edges.
- If the graph is still strongly connected, we keep finding maximum cycle flows till there are no more cycles.
- The newly updated graph may not be strongly connected. Then we check if it includes new strongly connected components in it. And apply the same procedure recursively.
- Finally, the graph will turn into an acyclic graph having no cycles at all, and when the algorithm is done, and it returns the total amount and the chosen maximum cycle flows. As following pseudocode:

```
#SCG: strongly connected graph.

# new_scg: strongly connected graphs appear in SCG after update.

# total: the total reduced amount.

# chosen_cycle_flows : list of maximum cycle flows chosen to produce the
total amount.

## find_total_reduced_amount(SCG)##


chosen_cycle_flows = []

total reduced_amount = 0

While SCG is a strongly connected graph where number of nodes > 1:

    max_cycle_flow = find_max_cycle_flow(SCG)

    total_reduced_amount += max_cycle_flow[value]

    chosen_cycle_flows.append(max_cycle_flow[cycle_flow])

    update_SCG(SCG)


    for each new_scg in SCG:

        find_total_reduced_amount(SCG) //recursivly

return total_reduced_amount , chosen_cycle_flows
```

Figure 12 pseudocode: find total amount reduced.

Output:

- The **total reduced amount** refers to the amounts of original foreign transactions that can be replaced by local transactions to save its foreign fees.
- The **chosen cycle flows** refer to the cycle flows that existed in the original graph and allowed some foreign transaction fees to be eliminated and replaced with local transactions.

That was a detailed description of the idea of the algorithm and its steps. I did not prove the algorithm and I may have missed some cases that this procedure does not cover. Specifically, the algorithm output is not proven to be an optimal solution. However, I tried it on many different cases, and it got a correct solution, yet I cannot claim if it is optimal or not without proof.

Chapter 3

User Documentation

This chapter a user’s manual of the application. First, I will give an overview of the tool explaining its functionality and what can an end-user do with it. Then I will demonstrate the technical aspects of running the application; the environment requirements needed for running it, and how to install it in few steps and run. Finally, I will explain the step-by-step procedure of creating reports.

3.1 Analaysaction web application

Analysaction is a web application that analyzes foreign transaction networks intending to minimize transaction fees. the tool is based on an analysis model that promotes an alternative to traditional money transfers where foreign transactions can be replaced with local ones which means less fees. The application receives a network of foreign transactions and runs the model’s algorithm on it to produce a full analysis report. The report gives detailed information about the reduced amount, the replaced transactions, and how money flows among countries in such a system. The main purpose of using this tool is just to show that there could be a money transfer system where foreign transaction fees are far less than existing traditional systems.

3.2 Environment

The application is developed in Python Django framework for the backend and HTML, CSS, and JavaScript for the frontend. Bootstrap framework and Boostwatch were integrated to enhance the site styling and make it responsive on different devices. And Finally, Highcharts JavaScript library for the data visualization.

These are the specifications of the development environment:

- Operating System: Windows 10
- Ram: 8.0 GB
- CPU: 2.80 GHz
- IDE: PyCharm
- Python 3.8.5
- Django 3.2

3.3 Application Installation

To install the application, you must have Python and pip installed on the device so that you can run the following commands, if your device does not have these follow these references to install:

- Python: https://www.Python.org/downloads/
- pip: https://pip.pypa.io/en/latest/installing/

The project is provided in a file **my_application.zip** format. Unzip the file and move to my_application directory in your terminal and follow the following steps.

Step 1: setting up a virtual environment:

- Install **virtualenv** with this command:

```
py -m pip install --user virtualenv
```

- Create a **new virtual environment** with this command:

```
py -m venv myproject
```

- **Activate** the virtual environment with this command:

```
.\myproject\Scripts\activate
```

Now our virtual environment is set up and running.

Step 2: installing application's requirement:

The application requirements are the additional packages that the application code utilizes. For instance, this application utilizes Python NetworkX 2.5 to create the network objects. All the requirements are stored in the root directory **my_application** named **requirements.txt**.

- To install all the requirements, run this command:

```
pip install -r requirements.txt
```

Step 3: Collect static files in the root directory:

- Static files are CSS and JavaScript. We need to collect those in the directory named root. to do that run this command:

```
python manage.py collectstatic
```

  If it asked to updated existing files type: yes
- Application is ready to run its server locally by this command:

```
python manage.py runserver
```

Now the server is up and running.

```
Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

May 27, 2021 - 05:33:17

Django version 3.2, using settings 'analysaction.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.
```

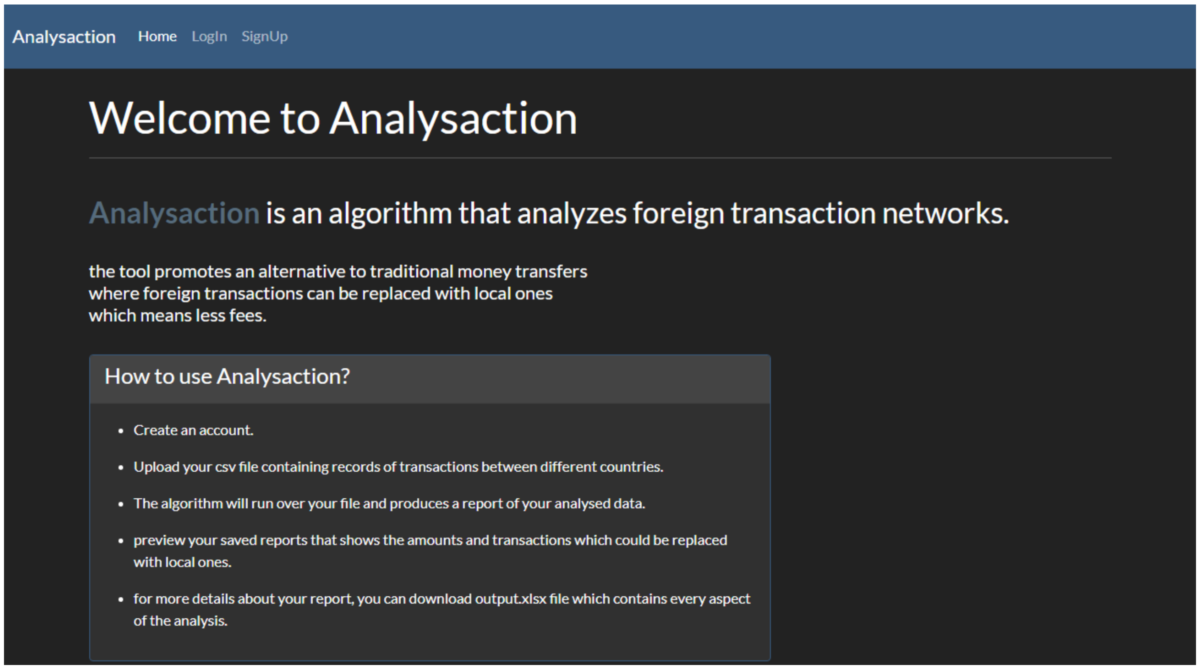
Figure 13 Running server response.

That means the website is ready and we can access the home page with this URL in your browser:

`http://127.0.0.1:8000/app/`

3.4 Application usage guide

The user can use the application in five functions starting from the opening page; creating an account, logging in to profile, uploading a create a new report, preview previous reports, and finally downloading the output analysis file.



Welcome to Analysaction

Analysaction is an algorithm that analyzes foreign transaction networks.

the tool promotes an alternative to traditional money transfers
where foreign transactions can be replaced with local ones
which means less fees.

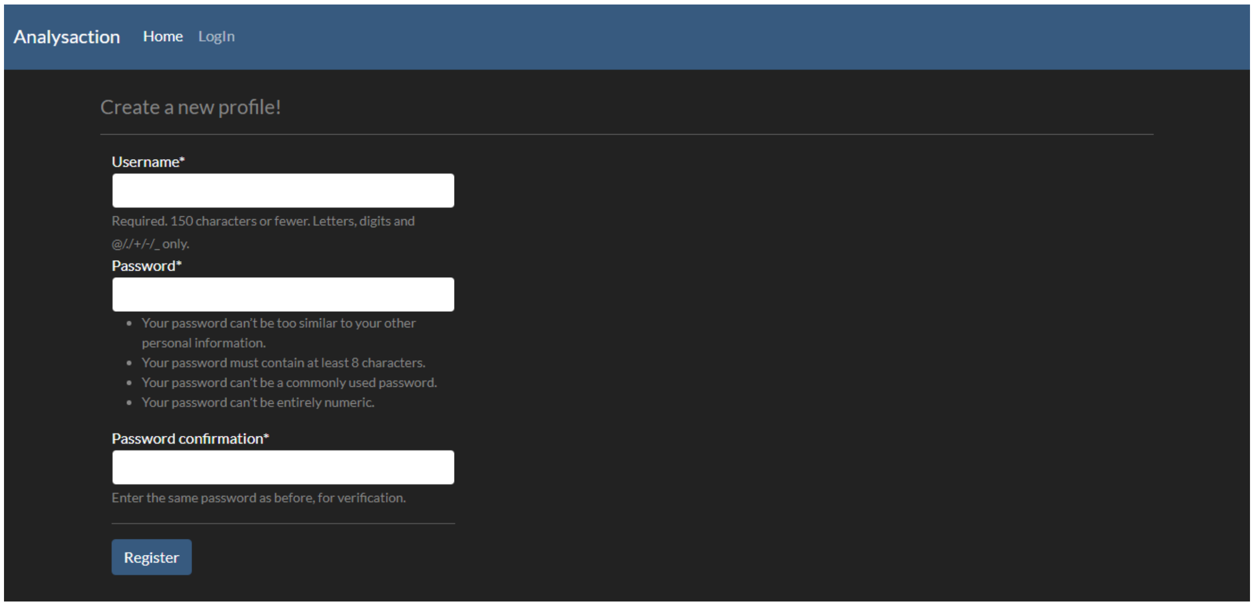
How to use Analysaction?

- Create an account.
- Upload your csv file containing records of transactions between different countries.
- The algorithm will run over your file and produces a report of your analysed data.
- preview your saved reports that shows the amounts and transactions which could be replaced with local ones.
- for more details about your report, you can download output.xlsx file which contains every aspect of the analysis.

Figure 14 Analysaction app: openong page

1. **Create an account:**

On the home page, click the **SignUp** button. This will redirect to the registration page.



Create a new profile!

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

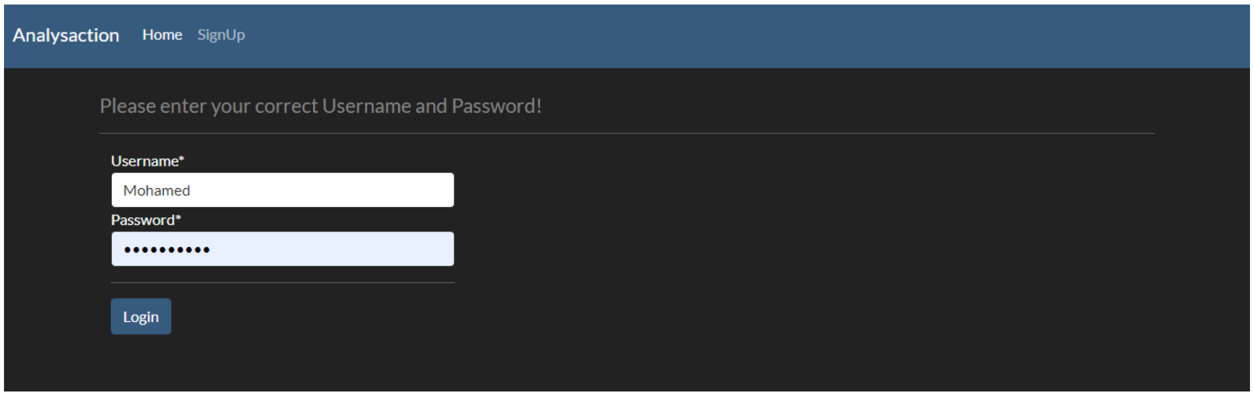
Enter the same password as before, for verification.

Figure 15 Analysaction app: Registration page

Fill the form with a valid username and password and click **Register**. This will create your account and redirect you to the login page.

2. **Log in to your profile:**

On the login page, fill the form with your correct username password click **Login**.



Please enter your correct Username and Password!

Username*: Mohamed

Password*: ••••••••••

Figure 16 Analysaction app: Login page

This will redirect you to your profile.

3. **Create a new report:**

In your profile, click the green button **+ Create a new report.**

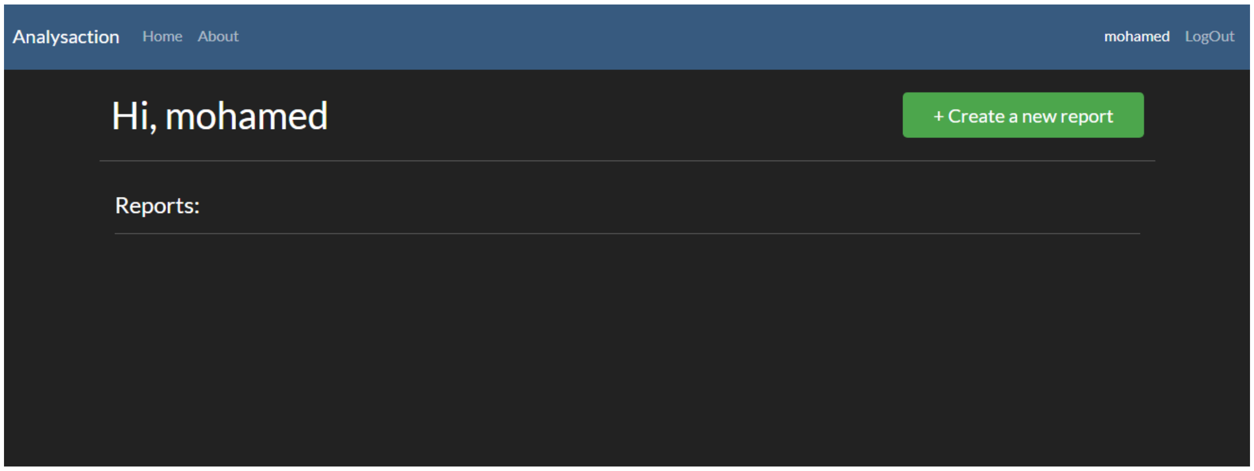
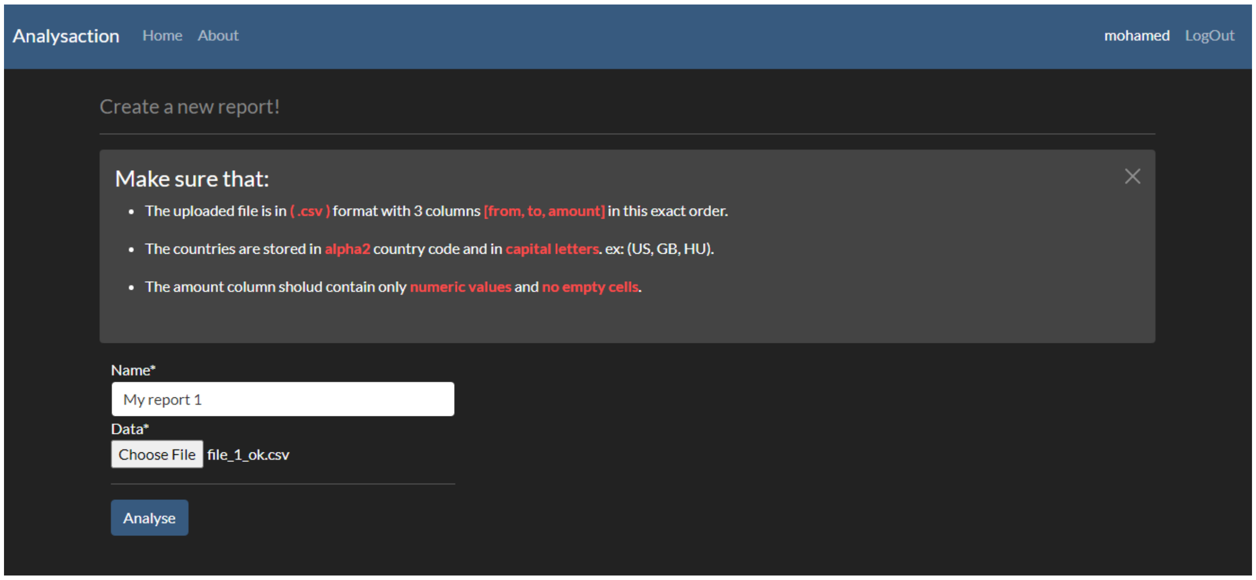


Figure 17Analysaction app: Profile page - no reports

This redirects to the report form page to upload your CSV file.



Make sure that:

- The uploaded file is in (.csv) format with 3 columns [from, to, amount] in this exact order.
- The countries are stored in alpha2 country code and in capital letters. ex: (US, GB, HU).
- The amount column sholud contain only numeric values and no empty cells.

Name* My report 1

Data* Choose File file_1_ok.csv

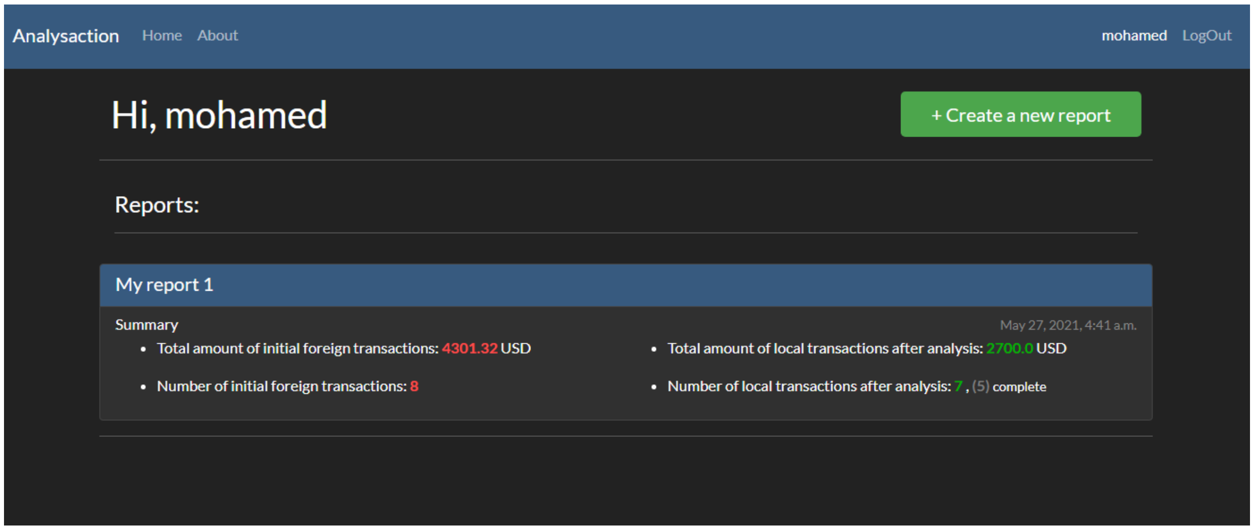
Figure 18Analysaction app: Report form page

On the report form page, you are required to name your report and upload a CSV file that meets the validation requirements. A warning would pop up if your file is not valid informing, you of the requirement that your file did not meet. The requirements are:

1. The uploaded file extension can only be **CSV**.
2. The file contains, only, three columns named respectively **“from”**, **“to”**, **“amount”**.
3. **“from”**, **“to”** columns records shall be filled with **ISO 3166-1 alpha-2** [20] country code in capital letters.
4. **“amount”** column records shall be filled with, only, **numeric values.**
5. **No missing rows or missing values** in the entire file.

If there is any duplication in the records of the file it will be aggregated on **from – to** columns.

Click **Analyze** and this will redirect you to your profile page again to preview your created Report summary card. The report summary card gives general info about your report. Click on it to redirect to the report page.



Hi, mohamed

Reports:

My report 1

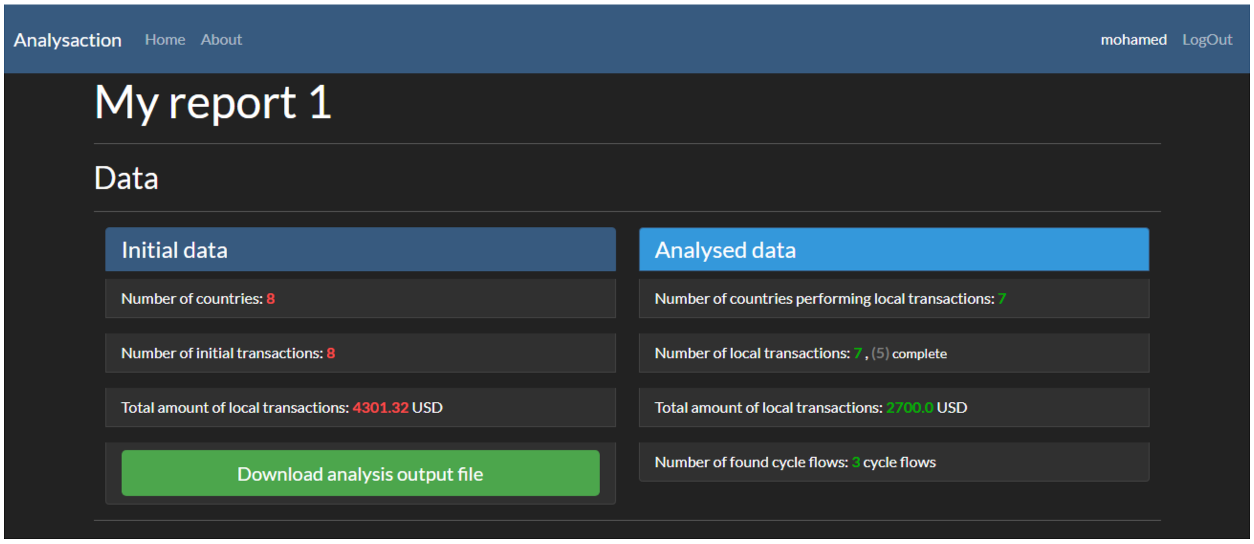
Summary May 27, 2021, 4:41 a.m.

- Total amount of initial foreign transactions: 4301.32 USD
- Number of initial foreign transactions: 8
- Total amount of local transactions after analysis: 2700.0 USD
- Number of local transactions after analysis: 7 , (5) complete

Figure 19 Analysaction app: profile page - with reports cards

4. Preview your saved reports

The report page has two sections. The first is data where you can find more details about your report's initial data and analysis output: the number of initial countries and countries that performed local transactions after the analysis, the initial number of transactions, and the number of transactions after the analysis. and the number of cycle flows detected in the initial network. And finally, the total amount of the initial transactions and the total amount that can be transacted locally after the analysis.



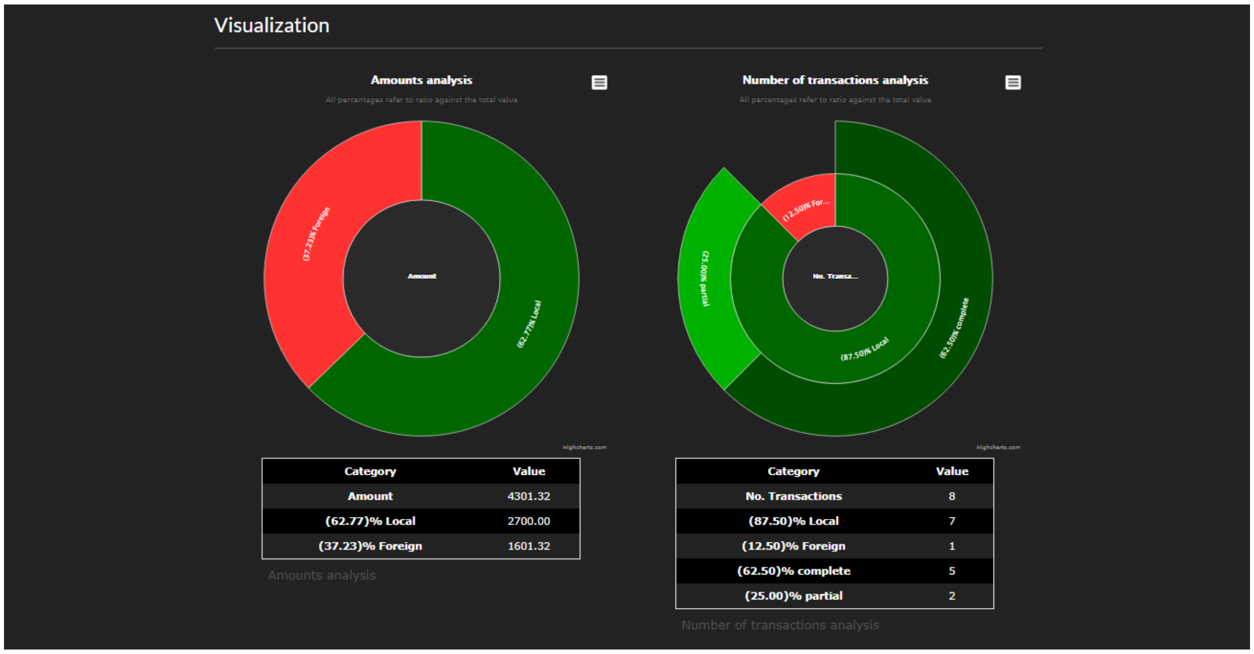
My report 1

Data

Initial data	Analysed data
Number of countries: 8	Number of countries performing local transactions: 7
Number of initial transactions: 8	Number of local transactions: 7 , (5) complete
Total amount of local transactions: 4301.32 USD	Total amount of local transactions: 2700.0 USD
	Number of found cycle flows: 3 cycle flows

Figure 20 Analysaction app: report page: Data section

The second section is the visualization of the data analyzed in a sunburst chart to give a visual sense of the ratio between the initial and analyzed local amounts and for the number of transactions participating in the analysis network. By hovering over the chart, it gives the exact amounts or number of transactions of the portions in the chart. also, the number of transaction charts is interactive, and by clicking it shows the ratio between complete and partial local transactions.



Category	Value
Amount	4301.32
(62.77)% Local	2700.00
(37.23)% Foreign	1601.32

Amounts analysis

Category	Value
No. Transactions	8
(87.50)% Local	7
(12.50)% Foreign	1
(62.50)% complete	5
(25.00)% partial	2

Number of transactions analysis

Figure 21 Analysaction app: report page: Visualization section

Also, a table of the chart can be downloaded in different formats (PNG, JPG, …) by clicking the white menu button in the right upper corner of each chart corner. Additionally. There is an option to show the percentages in a table under the chart from the same button.

5. Download the analysis output file

For deeper details about your report, click the **Download analysis output file** green button to download the output.xlsx file which contains every aspect of the analysis. the exact cycle flow paths. The local amount in every single transaction or every single.

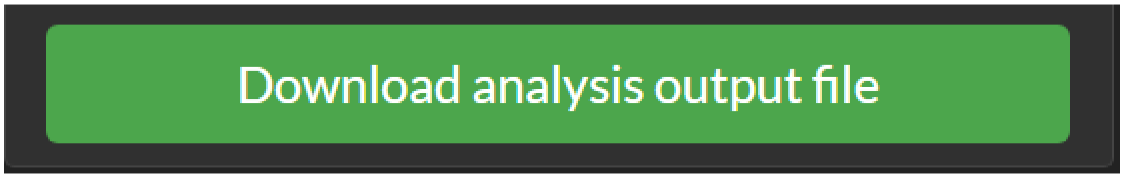
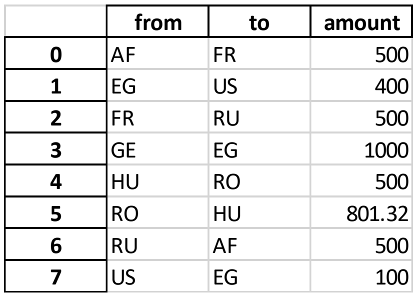


Figure 22 Analysaction app: report page: download output button.

The excel file contains four sheets.

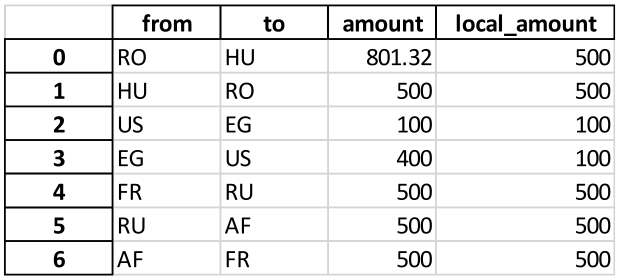
Initial transactions: the records of the initial data. Aggregated on from -to if there are any duplicate transactions.



	from	to	amount
0	AF	FR	500
1	EG	US	400
2	FR	RU	500
3	GE	EG	1000
4	HU	RO	500
5	RO	HU	801.32
6	RU	AF	500
7	US	EG	100

Figure 23 analysis output file: initial transactions

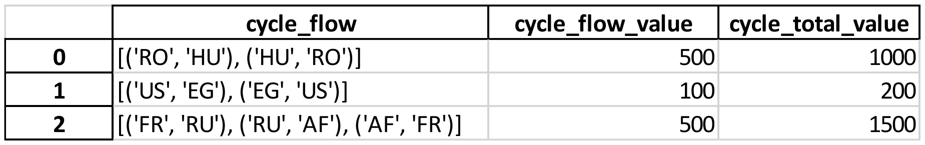
Local transactions: the amount transacted locally in every single transaction.



	from	to	amount	local_amount
0	RO	HU	801.32	500
1	HU	RO	500	500
2	US	EG	100	100
3	EG	US	400	100
4	FR	RU	500	500
5	RU	AF	500	500
6	AF	FR	500	500

Figure 24 analysis output file: Local amounts on each transaction

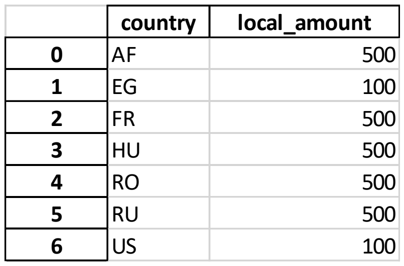
Cycle flows: the exact cycle paths of each cycle flow detected.



	cycle_flow	cycle_flow_value	cycle_total_value
0	[('RO', 'HU'), ('HU', 'RO')]	500	1000
1	[('US', 'EG'), ('EG', 'US')]	100	200
2	[('FR', 'RU'), ('RU', 'AF'), ('AF', 'FR')]	500	1500

Figure 25 analysis output file: cycle flows

Local countries: the local amount to be transacted within every country in the analyzed network.



	country	local_amount
0	AF	500
1	EG	100
2	FR	500
3	HU	500
4	RO	500
5	RU	500
6	US	100

Figure 26 analysis output file: countries local transactions

You can test the application with the provided sample.ok and sample.error in directory:

`Mohamed_Elsaadany_Thesis_work/sample`

Chapter 4

Developer Documentation

Overview

This chapter is an overall developer guide for the web application. It will contain every detail about the development process and the technologies used to produce the web application. The web application is developed in Python using the **Django** framework as it is one of the best for such purposes. Using its MVT (Model -View-Template) architecture eases and speeds the development process. For the front-end, the program uses HTML, CSS, and JavaScript for the user interface. **Bootstrap** and **Bootswatch** themes are added for the styling. JavaScript **Highcharts** library also was integrated for visualizing the report's data in neat colorful charts. In Addition to the Analysis algorithm python script, the main component of the program. the documentation also includes specifications of the program to demonstrate what the program does.

4.1 Specifications

The web application functionality is quite simple. It allows the user to analyze his foreign transaction data in few steps so that they can get an overview of how much they could save if such an alternative money transfer system exists.

- A user needs to create an account on the web application so that he can perform his analysis and preview them individually. They can log in to their account and create a new analysis report. Simply, they give a name to their report and upload a CSV file containing their foreign transaction records in a specific order.

- The analysis algorithm runs over their data and creates a new report that they can access from their profile to visualize the results and know how much they can save transaction fees if they use this algorithm. Also, they can download the .xlsx file containing all the details about the analysis and the difference between the initial data they uploaded and the analysis result.

4.2 Django

Django[5] is a very commonly used Python-based framework for web development as it allows developers to create their web applications at a very rapid pace while taking into consideration the code cleanness and the very organized structure. Developers can always build their web applications from scratch. However, that may take an enormously long time and it will never be as organized and structured as what Django provides. Utilizing Django MVT architecture besides

Django forms, URLs, sqlite3 database, Admin site, and Django user authentication gives the developer all that they need to start implementing their websites.

4.2.1 MVT architecture

MVT is Django's design pattern consisting of three main components are Models, Views, and Templates. It is mainly the logical architecture of the web application code.

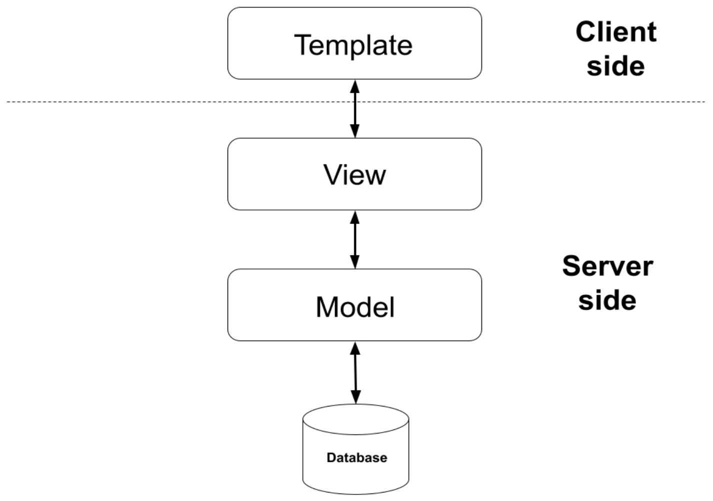


Figure 27 Model - View - Template architecture

- **Models** [6]:

  are the logical data structure of the whole application which means a code representation of the web application database tables. It is simply a python class wrapping the data fields in a specific object and maintains the relations between the various models of your web application. All the models are written in **models.py**.

- **Views** [7]**:**

  are python functions that handle the web application's requests. It receives a request and returns a response that can be a specific HTML page or errors or a file or anything you would like to receive when you call this function depending on the URL you assign to the view. Views can also access the models and perform operations on them like creating an instance or update or delete it. All views are written in **views.py**.

- **Templates** [8]**:**

  are representing the user interface as HTML pages. The view's job is to describe how shall the data be represented. Templates' job is to receive these data from views dynamically and show it to the user on the page as the view describes. Templates are saved as HTML pages in the **Templates** directory.

4.2.2 Django structure

In addition to the three main components, Django offers a lot more this design pattern. Django structure contains various more components to facilitate the developer work and keeping them away from repetitive tasks. This section will demonstrate some of the components which were used to develop this web application.

- **Django URLs** [9]**:**

  URLs are the link between a view and a template through a URLconf. URL configuration is a set of patterns that matches the requested URL and finds the correct view. their main job is to tell the view to redirect to or render which template and at the same time it can be used inside the template itself to trigger a specific view according to the code logic.
- **Django Form** [10]:

  Django Form class job is to create forms in an HTML page so that a user can fill and save their entries. The form content describes the fields to be filled that can be any of HTML input tags and it can represent the fields defined in a specific Django model. Additionally, at form creation on the page, we can assign the desired action method such as GET or POST.
- **Django Admin site** [11]:

  This is one of the strongest Django features. Django automatically creates an Admin site page where you can manage all the web application content. Once you migrate the models' data. The Admin page creates a tap for each class, and you can preview, edit, or even delete any of their records. However, to access this site you need a superuser account.
- **Django database** [12]:

  Django supports various databases for instance: Oracle, PostgreSQL, and MySQL. On the other hand, Django uses SQLite as a default database that is a very good solution for the development stage while working on the local host. This program uses the default SQLite3 as a database because the program is working on the local host server. When the web application extends and requires the deployment and hosting for real users, it is needed to set up one of the previously mentioned databases.

As a summary for the Django structure, those are Django elements that are used to develop this web application and that is how it looks like in the directory tree in a Django project:

app

- __init__.py
- admin.py
- apps.py
- forms.py
- models.py
- views.py
- urls.py
- templates/
- migrations/

4.3 Front-end frameworks

Web application front-end is not an easy task for developers. It is the most important part of the website because it is the interface that the end-user interacts with. There are plenty of libraries and frameworks that can help with this crucial element. In this application, I choose to use Bootstrap and Bootswatch themes for styling and organizing the site pages' structures. I choose these frameworks particularly since they are and open source. Furthermore, they are straightforward to integrate into my site and gave me a variety of styling options.

4.3.1 Bootstrap and Bootswatch

The application front-end utilizes **Bootstrap 5** [13] as a major CSS framework due to its variety of colors and grid systems. We add adding the style sheet link in the head tag of the HTML page and Besides the JavaScript links at the end of the body tag as following:

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.css
" rel="stylesheet"
      integrity="sha384-
wEmeIV1mKuiNpC+IOBjI7aAzPcEZeedi5yW5f2yOq55WWLwNGmvvx4Um1vskeMj0"
crossorigin="anonymous">
```

```
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js
"
        integrity="sha384-
IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
        crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/js/bootstrap.min.js"
        integrity="sha384-
lpyLfhYuitXl2zRZ5Bn2fqnhNAKOAaM/0Kr9laMspuaMiZfGmfwRNFh8HlMy49eQ"
        crossorigin="anonymous"></script>
```

Afterward, all Bootstrap classes and ids are accessible on the HTML page.

Similarly for **Bootswatch** [14] is a CSS themes library built on top of Bootstrap. I choose the **Bootswatch Darkly** [15] theme for this web application. Here is how it is integrated:

```
<link rel="stylesheet"
href='https://bootswatch.com/5/darkly/bootstrap.min.css'>
<link rel="stylesheet" href='https://bootswatch.com/5/darkly/bootstrap.css'>
```

4.3.2 Highcharts JavaScript

To visualize the analysis report data to the end-user, the application takes an advantage of the Highcahrts library[16]. Highcharts is an outstanding pure JavaScript library providing plenty of interactive charts' demos that can be used for a wide variety of data visualization purposes. On the application report's page, where the report data is visualized, a Highcharts sunburst [17] chart is chosen to picture the differences between the initial data uploaded and the analysis output. Sunburst chart brings out the data in the hierarchical form as following:

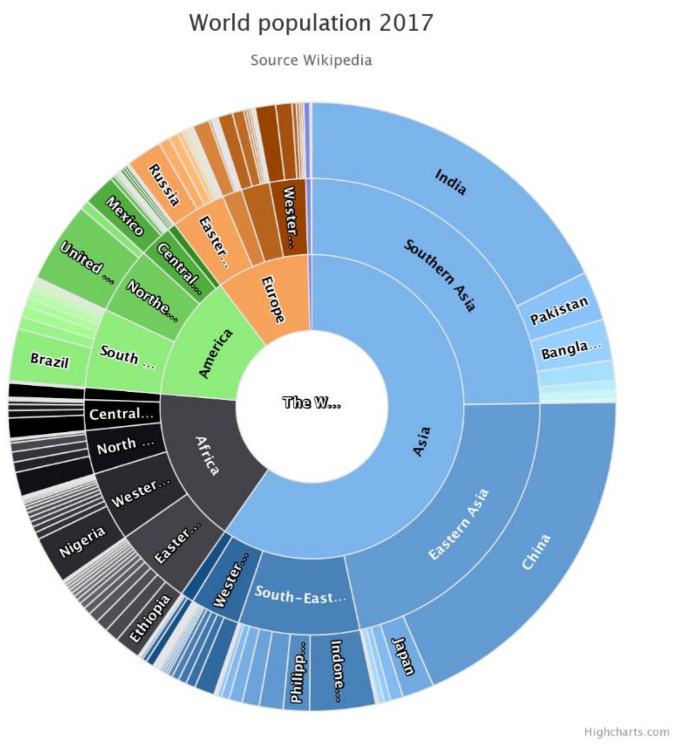


Figure 28 Sunburst chart example from Highcharts

For integrating Highcharts sunburst chart in an HTML page. It is needed to add these scripts:

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/sunburst.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
<script src="https://code.highcharts.com/modules/export-data.js"></script>
<script src="https://code.highcharts.com/modules/accessibility.js"></script>
```

After that, I can write my script that defines the wanted chart characteristics in terms of section representations, colors, texts, levels, and values as I will explain in the code implementation section.

4.4 Implementation

This section is covering the code implementation of the web application project. The project was developed using the **PyCharm** integrated development environment. We will go through each written code script and explain how each framework was used and for what purpose. Starting from Django components explained in the previous section. The project has two models, **Report**, **Profile**. In views, there are **home**, **register**, **show_profile**, **create_report**, **show_report**. All views matching with the correct **URLs** to redirect or render the wanted template including and the built-in **login** and **logout** from Django authentication. **forms.py** defining the forms used in the project and their validation functions. **admin.py** handles the admin site migrations. In addition to the **Templates** directory containing the user interface pages. In templates, we will go through the implementations of the front-end frameworks and libraries, Bootstrap, Bootswatch, and Highcharts. At last, I will explain the analysis algorithm implementation step by step in the **Analysis.py** Python script. Finally, I will show unit tests for the whole project.

4.4.1 Models

models.py contains two classes; each class has some attributes where the class data is saved in the required data structures:

1. **Profile class:** representing a specific user profile.

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, null=True)
    id = models.UUIDField(default=uuid.uuid4, primary_key=True)

    def __str__(self):
        return self.user.username
```

Figure 29 Profile class code

Profile fields definitions:

id:	A unique id in the form of **UUID4** [18] acting as a primary key for each profile so that it would be easier to filter the profile objects through its unique id.
user:	A Django User object one-to-one field. This means each user will be created by Django will be associated with a unique profile object that represents it.

Table 1 Profile fields

Profiles are linked to User objects so that they take an advantage of Django user authentication and this part will be explained in the URLs section. However, we need to set up this action to happen and the following code snippet added at the end of the models.py to play this role:

```
@receiver(post_save, sender=User)
def create_user_person(instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)


post_save.connect(create_user_person, sender=User)
```

Figure 30 linking user to profile code

2. **Report class:** representing a specific report analysis created by a user.

```
class Report(models.Model):
    id = models.UUIDField(default=uuid.uuid4, primary_key=True)
    name = models.CharField(max_length=250, unique=False, default="")
    Creator = models.ForeignKey(settings.AUTH_USER_MODEL, 
related_name='creator', on_delete=models.CASCADE, null=True)
    data = models.FileField(upload_to="files/", 
validators=[FileExtensionValidator(allowed_extensions=['csv'])])
    output_data = models.FileField(upload_to="files/output")
    timestamp = models.DateTimeField(auto_now_add=True)
    total_local_transactions = models.FloatField(default=0.0)
    total_initial_transactions = models.FloatField(default=0.0)
    no_of_initial_transactions = models.IntegerField(default=0)
    no_of_initial_countries = models.IntegerField(default=0)
    no_of_local_countries = models.IntegerField(default=0)
    no_of_cycles_flows = models.IntegerField(default=0)
    no_complete_local_transactions = models.IntegerField(default=0)
    no_of_local_transactions = models.IntegerField(default=0)

    def __str__(self):
        return self.name
```

Figure 31 Report class code

Report fields definitions:

id:	A unique id in the form of **UUID4** acting as a primary key for each report so that it would be easier to filter the report objects through its unique id.
name:	A **CharField** that stores the report name.
Creator:	A **ForeignKey field** with a User parameter represents the creator of the report. The **ForeignKey** in Django represents a **many-to-one relationship** [19]. This means, in this case, each User can have as many reports as their creator. However, each report can have only one User as a creator.
data:	A **FileField** stores the report uploaded file and uploads it to the "**files/**" directory. data field accepts only **CSV** files as it is a requirement for **Analysis.py** to run.
output_data:	A **FileField** stores the after-analysis generated **XLSX** file and uploads it to the "**files/output**" directory to allow the user to download it later.
timestamp:	A **DateTimeField** stores the exact time of the report creation using the parameter (auto_now_add=True).
total_local_transaction:	A **FloatField** stores the total amount of transactions that can be performed locally after running the analysis.
total_initial_transactions:	A **FloatField** stores the total amount of initial transactions from the uploaded CSV file before running the analysis.
no_of_initial_transactions:	An **IntegerField** stores the number of initial transactions from the uploaded CSV file before running the analysis.
no_of_initial_countries:	An **IntegerField** stores the number of initial countries from the uploaded CSV file before running the analysis.
no_of_local_countries:	An **IntegerField** stores the number of countries that can perform local transactions after running the analysis.

no_of_cycles_flows:	An **IntegerField** stores the number of best cycle flows that are found in the network of initial transactions after running the analysis.
no_of_complete_local _transactions:	An **IntegerField** stores the number of transactions that could be performed locally with its full amount after running the analysis.
no_of_local_transactions:	An **IntegerField** stores the number of transactions that could be performed locally amount after running the analysis (either with its full amount or with partial amount).

Table 2 Report fields definitions

4.4.2 Views

Views.py contains five function-based views. Each view takes a request and returns a response as rendering a specific template or redirecting to a specific URL as following:

home (request):

```
def home(request):
    return render(request, 'home.html')
```

Figure 32 home view

home view, Simply, it takes the user request and renders the **'home.html'** template.

register (request):

```
def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('/accounts/login')
    else:
        form = UserCreationForm()

    return render(request, 'registration/register.html', {'form': form})
```

Figure 33 register view.

register view role is handling the registration page requests. Using Django built-in forms, it creates the User creation form, and on form submission, it validates its input values and saves the User object if the form is valid then redirects to the login URL. Otherwise, it raises a form validation error then renders the `'registration/register.html'` page with a new form and asks the user to input valid values.

The next 3 view functions require a user that is stored in the database and currently authenticated to be called. For that this code snippet is added:

```
@login_required
@transaction.atomic
```

Figure 34 login required code.

show_profile (request):

```
def show_profile(request):

    my_reports = 
list(reversed((Report.objects.filter(Creator=request.user))))
    return render(request, 'profile.html', {'my_reports': my_reports})
```

Figure 35 show_profile view

show_profile view is the one that handles rendering the user's profile showing their specific reports. In the models' section, it was explained that the User-Profile relationship is one-to-many and the foreign-key is the report creator. Given that, the show_profile view simply filters all the report objects on this condition: **the report creator = = the current user**. Afterward, it renders the `'profile.html'` page only with the user's reports context.

show_report (request, id):

```
def show_report(request, id):

    report = Report.objects.get(id=id)

    return render(request, 'Report.html', {'report': report})
```

Figure 36 show_report view

show_report takes the user request and the report id as arguments and it renders the report page. The id here argument has two roles: first, filtering the Report objects on the unique report id. Second, defining the view URL configuration to match the required report exactly on calling this view (more details in URLs section).

create_report(request):

```
def create_report(request):
    form = ReportForm

    if request.method == 'POST':
        form = ReportForm(request.POST, request.FILES)
        if form.is_valid():
            my_report = form.save(commit=False)
            my_report.Creator = request.user
            my_report.save()
            analyse_response = Analysis.analyse(my_report.data.file)

            saved_report = Report.objects.get(id=my_report.id)

            output_file_path = "media/files/output/" + request.user.username + "_" +
saved_report.name + "_" +\
                                   str(saved_report.id) + "_output.xlsx"

            with pd.ExcelWriter(output_file_path) as writer:
                analyse_response[4].to_excel(writer, sheet_name='initial_transactions')
                analyse_response[1].to_excel(writer, sheet_name='local_transactions')
                analyse_response[0].to_excel(writer, sheet_name='cycles_flows')
                analyse_response[5].to_excel(writer, sheet_name='local_countries')

            saved_report.output_data.name = "files/output/" + request.user.username + "_" +
saved_report.name + "_" \
                                                  + str(saved_report.id) + "_output.xlsx"

            saved_report.total_local_transactions = analyse_response[2]
            saved_report.total_initial_transactions = analyse_response[3]
            saved_report.no_of_initial_transactions = analyse_response[6]
            saved_report.no_of_initial_countries = analyse_response[7]
            saved_report.no_of_local_countries = analyse_response[8]
            saved_report.no_of_cycles_flows = analyse_response[9]
            saved_report.no_of_local_transactions = analyse_response[10]
            saved_report.no_complete_local_transactions = analyse_response[11]

            saved_report.save()

            return redirect('show_profile')

    return render(request, 'Report_form.html', {'form': form})
```

Figure 37 create_report view.

create_report view is the most crucial view in the project since handles the report form page.

1. It creates a report form and waits to receive the input values (report name, transactions CSV file) from the user.

If the form input values are valid, it:

2. Runs the analysis over the uploaded file.
3. Receives Analysis returns and saves each value in the correct report object's attribute.

4. Creates the data output .xlsx file and fills its sheets with the analysis result data frames.
5. Saves the report as an object in the database.
6. Redirects to show_profile URL.

If the form input values are invalid, it raises a form validation error and renders the report form page asking the user to input valid data.

4.4.3 URLs

URLs are the paths that views use to perform a request. In a Django project, there are two different **urls.py** files. The first is the project-level URLs (**analysaction/urls.py**). The second is the application-level URLs (**analysaction/app/urls.py**).

Project-level URLs:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('app/', include('app.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
]
if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT)
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

Figure 38 Project-level URLs

These URLs handles the overall project paths, and it should include each application URLs in the project. In this project, we have one application called app. The URL patterns include the **admin** path that redirects to the Django admin site, the **app** path that contains all our application URLs, and the **accounts** path that includes all built-in Django authentication URLs. Admin and accounts paths are built-in features in Django while the app path is the application custom paths that we will demonstrate in the following section.

Finally, in the development stage, the URL patterns add two paths for the static files. **STATIC_ROOT** is the location where any project additional files will be added. For instance, any JavaScript or CSS files for styling the templates. **MEDIA_ROOT** is the location where all the application's uploaded files, and downloaded files can be found. For example, the user's uploaded CSV file or the generated .xlsx output file that the user downloads after creating a report.

app-level URLs:

```
urlpatterns = [

    path('', views.home, name='home'),
    path('register', views.register, name='register'),
    path('profile', views.show_profile, name='show_profile'),
    path('report/<uuid:id>', views.show_report, name='show_report'),
    path('new_report', views.create_report, name='create_report'),
 ]
```

Figure 39 Application-Level URLs

These URLs were created to handle the application's views. The paths are very descriptive but let us have a deep look at the show_report path. The path describes which view it triggers which is **views.show_report**. Also, the path content which is `'report/<uuid:id>'` and that defines how this page address will look like (**app/report/ <some report id>)**. The id is added in this path specifically because the assigned view takes a report id as an argument.

4.4.4 Forms

Forms feature is one of the most beneficial gems in the Django framework. It takes care of creating all types of forms a developer may need to add to their application in few lines of code. In this application, I created one form called **ReportForm**. Additionally, I used two built-in forms using their URLs for logging in and creating a new account.

ReportForm:

```
class ReportForm(forms.ModelForm):
    class Meta:
        model = Report
        fields = ['name', "data"]

>    def clean_data(self): ...
```

Figure 40Report Form

Creating the form class only needs to define the model and the data fields that the form accepts.

This form allows the user to create a new report by assigning a name and uploading a file to save them in the fields name and data in the newly created report object. And here comes the value of the Django form where we need to validate the user's entries.

Data validation:

```
def clean_data(self):
    countries = []
    for country in pycountry.countries:
        countries.append(country.alpha_2)

    data_passed = self.cleaned_data.get("data")

    if not data_passed.name.endswith('.csv'):
        raise forms.ValidationError(_("Please upload a .csv file. this is the only
format allowed"))

    df = pd.read_csv(data_passed.file)

    if not (df.columns[0] == "from" and df.columns[1] == "to" and df.columns[2] ==
"amount"):
        raise forms.ValidationError(_("Please fix the columns titles to the form (from
, to , amount)"
                                      " in the exact order"))

    if not df['from'].isin(countries).all():
        raise forms.ValidationError(_("Please fix from countries to match the alpha2
country code "
                                      "(US, GB, HU, ...) in capital letters."))

    if not df['to'].isin(countries).all():
        raise forms.ValidationError(_("Please fix to countries to match the alpha2
country code "
                                    "(US, GB, HU, ...) in capital letters"))

    if pd.isna(df['amount']).any() or not np.issubdtype(df['amount'].dtype,
np.number):

        raise forms.ValidationError(_("Please fix amounts to be all numbers and make
sure there are "
                                      "no empty cells"))

    return data_passed
```

Figure 41 Data validation: clean_data function

4.4.5 Templates

In this section, we will go through the application pages and the integrated scripts. The application has six pages, and its location is (**app/templates**). The registration pages have a separate directory named **registration** and the rest are in the parent directory **templates**.

As mentioned in the **Front-end and frameworks** section all HTML pages utilities Bootstrap and Bootswatch integrations for styling. Here we will go through the content of each template. Each page body has the following structure:

```
<!doctype html>
<html lang="en">
    <head>
          ...
    </head>
    <body>
        <header>
            <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
                   ...
            </nav>
        </header>
        <main>
                   ...
        </main>
        <script> ... </script>
    </body>
</html>
```

Figure 42 HTML page structure

The page content is what is between the <**body**> tag. <**header**> tag is the upper part of the page and this is where the navigation bar is located. <**main**> tag is the rest of the page content.

Navigation bar:

The navigation bar is very similar in all pages although the differences occur in the URLs assigned to the clickable blocks that redirect to the required pages. This is home page navigation bar as:

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
    <div class="container-fluid">
        <a class="navbar-brand">Analysaction</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarColor01"
                aria-controls="navbarColor01" aria-expanded="false" aria-label="Toggle
navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarColor01">
            <ul class="navbar-nav me-auto">
                <li class="nav-item">
                    <a class="nav-link active" href="{% url 'home' %}">Home
                        <span class="visually-hidden">(current)</span>
                    </a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'login' %}">LogIn</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'register'%}">SignUp</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
```

Figure 43 Navigation bar HTML code

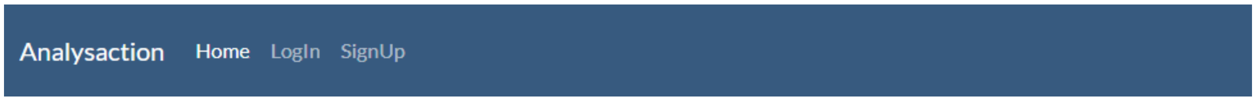


Figure 44 Navigation bar preview

The home navigation bar contains three clickable blocks that are Home, LogIn, and SignUp. Each one redirects to the assigned page by setting "`href="{% url '`<required URL>`' %}"`".

Registration and Login pages:

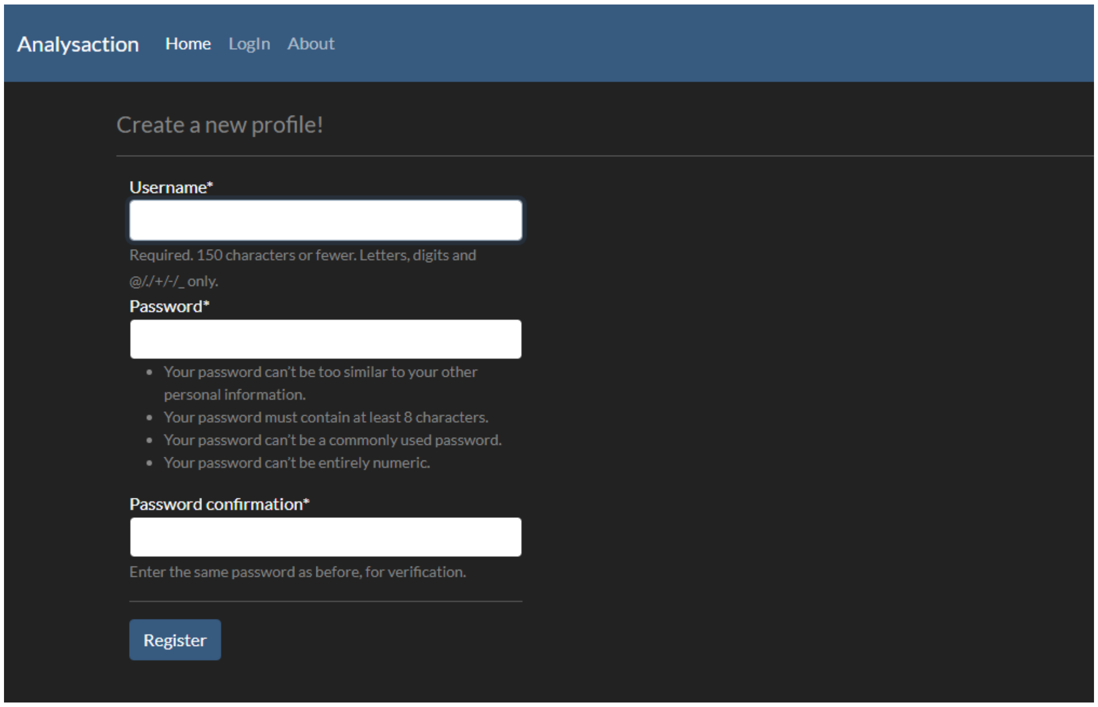
The two pages have very similar templates since each contains a Django built-in form so that a user can create an account on the registration page and log in to their profile on the login page. That is how the main tag in the **register.html** page code is written:

```
<main>
    <div class="container">
        {% load crispy_forms_tags %}
        <br>
        <h4>
            <small class="text-muted"> Create a new profile!</small>
        </h4>
        <hr>
        <div id="content-container" class="container center-block">
            <div class="lgn-container col-lg-4">
                <form id="login-form" method="post"
action="{%url'register'%}"%}>
                    {% csrf_token %}
                    {{ form | crispy}}

                    <hr>
                    <input type="submit" value="Register"
                           class="btn btn-primary pull-right" />
                    <input type="hidden" name="next" value="{{ next }}" />
                </form>
            </div>
        </div>
    </div>
</main>
```

Figure 45 Register page main HTML code

The two forms in the two pages run the built-in Django forms functionality and that means it validates and stores the newly created user in the database or authenticates a user to log in.


Analysaction Home LogIn About

Create a new profile!

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password*

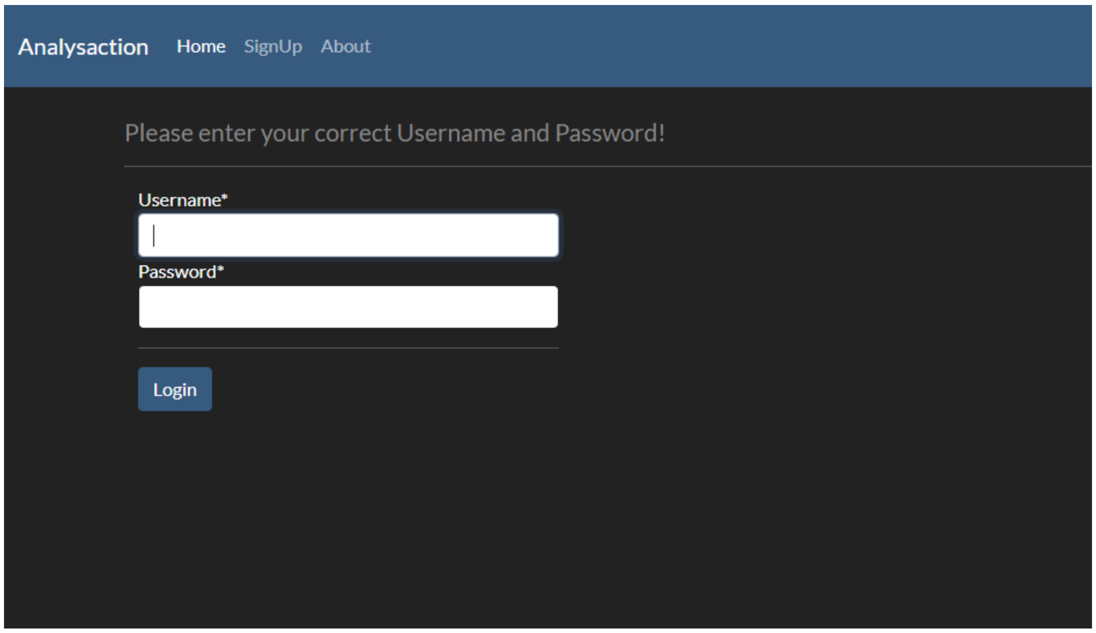
- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

Enter the same password as before, for verification.

Register

Figure 46 Registration page preview


Analysaction Home SignUp About

Please enter your correct Username and Password!

Username*

Password*

Login

Figure 47 Login page preview

Profile page:

The profile page is the user's main page that allows them to access their data or create new records. Its main tag has two elements: New report creation and showing user's previously stored reports. Here is how they are implemented:

```
<div class="container">
<br>
<div class="row  mx-auto">
        <div class="col-md-9">
            <h2>Hi, {{ request.user }}</h2>
        </div>
        <div class="col-md-3 center-block">
             <div class="d-grid gap-2">
                <a class="btn btn-lg" style="background: #4ca64c" href="{% url 'create_report' %}">
                    + Create a new report
                </a>
             </div>
        </div>
</div>
```

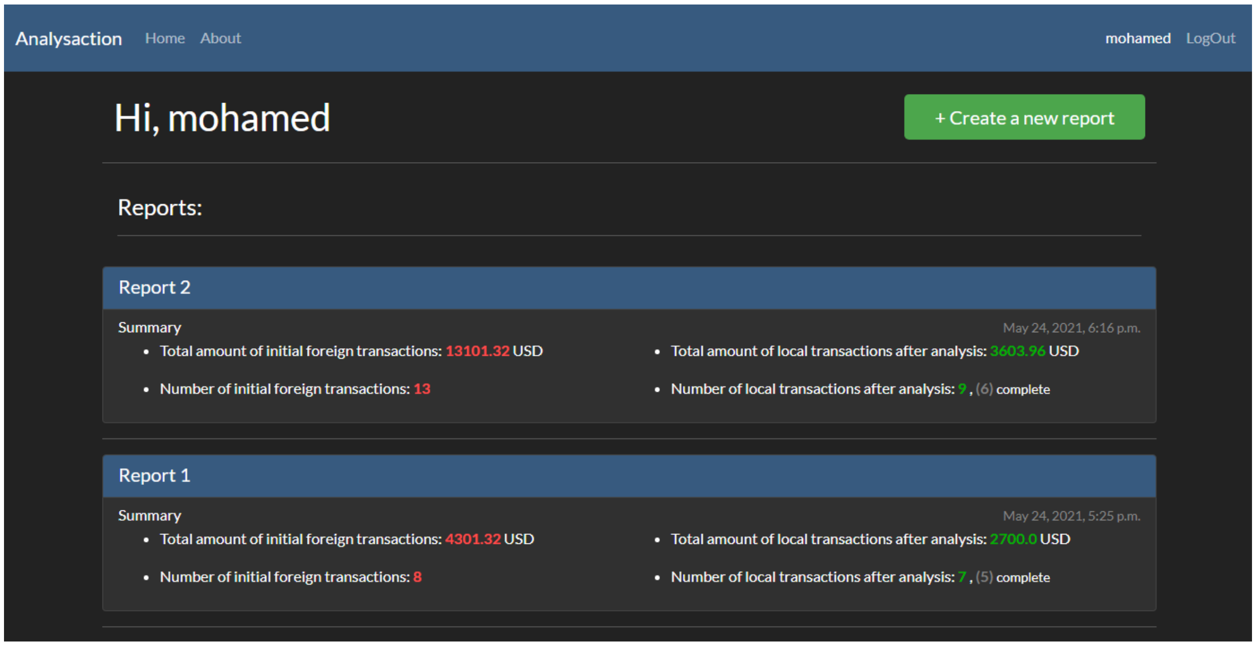
Figure 48 Profile page: New report creation HTML code

The URL **create_report** redirects to the **Report_form** page to upload their file. On the other hand, showing user reports uses a for loop over the context variable "**my_reports**" that is received from the **show_profile** view. That is how it is implemented:

```
{% for r in my_reports %}
        ...
{% endfor %}
```

Figure 49 Profile page: looping over user's reports.

Within the loop, a card is created for each report and it can access any of the report's data fields to show and it shows the report summary:



Report 2

Summary — May 24, 2021, 6:16 p.m.

- Total amount of initial foreign transactions: 13101.32 USD
- Number of initial foreign transactions: 13
- Total amount of local transactions after analysis: 3603.96 USD
- Number of local transactions after analysis: 9 , (6) complete

Report 1

Summary — May 24, 2021, 5:25 p.m.

- Total amount of initial foreign transactions: 4301.32 USD
- Number of initial foreign transactions: 8
- Total amount of local transactions after analysis: 2700.0 USD
- Number of local transactions after analysis: 7 , (5) complete

Figure 50 Profile page preview

Report page:

This is the largest page of the application. It uses the view **show_report** that gives the template access to the required report object. It shows the report data similarly to the report cards in the profile page shows the report summary. However, it has two more components.

Analysis out_put file download button:

```
<a class="d-grid gap-2 list-group-item list-group-item-action"
href="{{report.output_data.url}}">
    <button class="btn btn-lg " style="background: #4ca64c"
type="button">Download analysis output file</button>
</a>
```

Figure 51 Report page: Analysis out_put file download button.

The out_put file is saved as a field in the report object. To download the file, we need to define a clickable `href` that points to this field's URL property. The button is assigned to this value.

Charts' visualization:

To help the user to visualize the difference between their initial data and what the analysis produces, I used the Highcharts library in **JavaScript** and **CSS**. The chosen chart is the **sunburst chart**[17] and we have already mentioned how we can integrate it into the HTML page. Here we will explain how we structure it to suit our represented data and how it receives the data from a report object. I needed to write a custom JavaScript for that purpose but to integrate a custom file we need to add it to the static file location and then let the HTML page access it. The static file location is (**app/boot/js/sunburst_chart.js**). I used the exact sunburst chart documentation implementation. The only difference is the data dictionary that the script receives but first I needed to store the wanted report data in JavaScript variables as following:

```
<script>
    let local_amount = {{ report.total_local_transactions }};
    let total = {{report.total_initial_transactions}};
    let no_all_transactions = {{ report.no_of_initial_transactions }};
    let no_local_transactions = {{ report.no_of_local_transactions }};
    let no_complete_transactions = {{ report.no_complete_local_transactions
}};
</script>
```

Figure 52 Report page: storing report data in JavaScript variables.

Then in the **sunburst_chart.js,** I edited the data dictionary to visualize data in that format:

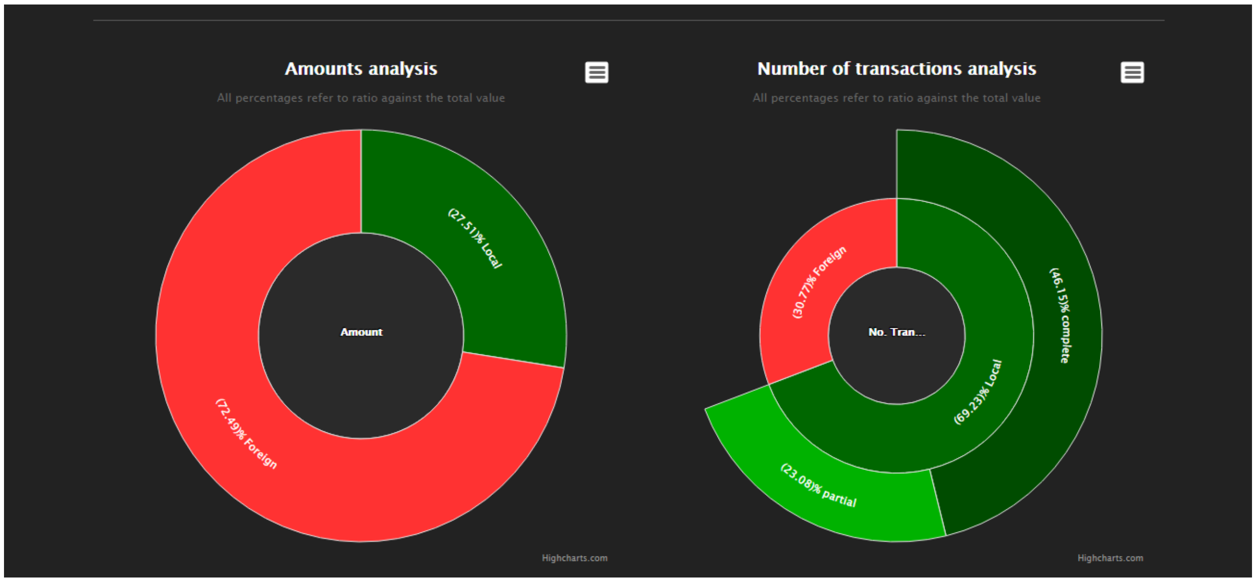


Figure 53Report page: charts preview

```
let data_2 = [
  {
  id: 'all transactions',
  name: 'No. Transactions',
  color: "#2a2a2a",
  value: no_all_transactions
},
  {
  id: 'Local transactions',
  name: `(${((no_local_transactions ) * 100 /
no_all_transactions).toFixed(2)})% Local`,
  parent: 'all transactions',
  color: '#006700',
  value: no_local_transactions
}, {
  id: 'Foreign',
  name: `(${((no_all_transactions - (no_local_transactions )) * 100 /
no_all_transactions).toFixed(2)})% Foreign`,
  parent: 'all transactions',
  color: "#ff3232",
  value: no_all_transactions - (no_local_transactions )
},{
  id: 'complete',
  name: `(${(no_complete_transactions * 100 /
no_all_transactions).toFixed(2)})% complete`,
  parent: 'Local transactions',
  color: "#004c00",
  value: no_complete_transactions
     },{
  id: 'partial',
  name: `(${(((no_local_transactions ) - no_complete_transactions) * 100 /
no_all_transactions).toFixed(2)})% partial`,
  parent: 'Local transactions',
  color: "#00b200",
  value:(no_local_transactions ) - no_complete_transactions
     }];
```

Figure 54 Data visualization: Sunburst data dictionary in JavaScript

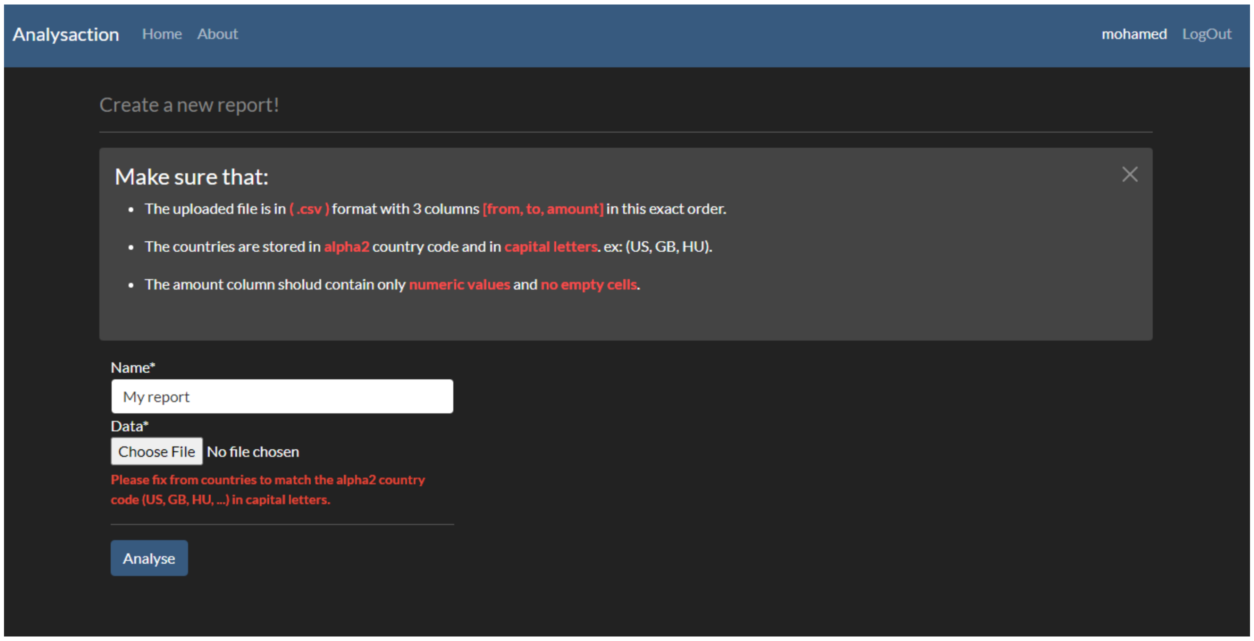
Important note: after any edits in static files, we need to run this command in the terminal so that it can be available for use a static file: `python manag.py collectstatic`

Report_form page:

This template allows the user to create his report using the **ReportForm** object. The template main has a warning message informing the user of the CSV file criteria. Also, because it pops up the form error messages if there were any invalid data entered.

```
<div id="content-container" class="container center-block">
    <div class="lgn-container col-lg-4">
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {{ form | crispy}}
            <hr>
            <input type="submit" value="Analyse"
                   class="btn btn-primary pull-right" />
            <input type="hidden" name="next" value="{{ next }}" />
        </form>
    </div>
</div>
```

Figure 55 Report_form page: form styling



Make sure that:

- The uploaded file is in (.csv) format with 3 columns [from, to, amount] in this exact order.
- The countries are stored in alpha2 country code and in capital letters. ex: (US, GB, HU).
- The amount column sholud contain only numeric values and no empty cells.

Name*

My report

Data*

No file chosen

Please fix from countries to match the alpha2 country code (US, GB, HU, ...) in capital letters.

Figure 56 Report form page preview

4.4.6 Analysis.py

This section describes how the analysis model is implemented in the program. **Analysis.py** is the python script that contains all the needed functions to perform analysis on the desired file and it is called on the report creation stage. Once the uploaded CSV file is valid and saved in the report object, the report creation's view calls **Analysis.py** to run over the file and receive the rest of the report fields values, and saves them in the matching report object field.

The script imports **Pandas** and **NetworkX** [2] python libraries to maintain the required data structures and methods to perform the analysis. NetworkX is a great solution when it comes to graph theory and network flows and those exactly are what I used to produce this program.

4.4.6.1 Script step by step functionality:

1. Takes a CSV file as an input.
2. Transforms the file into panda's data frame.
3. Aggregates any duplicates on the columns "from" and "to" by summing the amounts, if any duplicates exist, and saves it in a new data frame with the name **initial_transactions_df**.
4. Stores the number of the initial transactions data frame in a variable **no_of_initial_transactions**.
5. Gets the total sum of amounts in the initial transactions data frame and stores it in a variable **total_initial_amount**.
6. Creates a NetworkX directed graph from the transactions data frame values.
7. Counts the number of nodes and stores it in **no_of_initial_countries**.
8. Breaks down the directed graph into strongly connected components.
9. For each strongly connected component, it finds the maximum cycle flow as a set of edges with amounts and:
    - Saves the cycle flow in **cycle_df**
    - Updates the strongly connected graph by eliminating the previous cycle flow positive edges values.
    - Checks the newly updated graph if it contains any smaller strongly connected graphs and it runs again from step **8** recursively until there are no more cycles in the graph. Then it moves to the next strongly connected graph.
10. Once there are no more strongly connected components left. It starts to store the rest of the variables to return them all. The rest of the variables:

Local _transactions_df: the amount transacted locally in every single transaction.

local_countries_df: the local amount to be transacted within every country in the analyzed network.

total_local_transactions: the total amount of transactions replaced to be local and the number of these transactions **as no_of_local_transactions.**

no_complete_local_transactions: the number of transactions that eliminated the full amount of a foreign transaction.

no_of_cycle_flows: the number of cycle flows detected in the network.

4.4.6.2 Code functions

The script contains seven functions. the main function named **analyse(file)** and six helper functions:

create_directed_graph(dataframe)

takes a data frame of the initial transactions and returns a NetworkX directed graph with the amounts as a capacity attribute of the edges.

```
def create_directed_graph(dataframe):
    dataframe = dataframe.values.tolist()
    directed_graph = nx.DiGraph()
    for i in dataframe:
        directed_graph.add_edge(i[0], i[1], capacity=i[2])
    return directed_graph
```

Figure 57code: create_directed_graph function

find_strongly_connected_components(directed_graph)

Takes a directed graph and breaks it into strongly connected components having more than one node to assure having a cycle at least and returns a list of strongly connected components.

```
def find_strongly_connected_components(directed_graph):
    strongly_connected_components =
list(nx.kosaraju_strongly_connected_components(directed_graph))
    for component in strongly_connected_components:
        if len(component) == 1:
            directed_graph.remove_node(list(component)[0])
    filtered_strongly_connected_components = filter(lambda a: len(a) > 1,
strongly_connected_components)
    return list(filtered_strongly_connected_components)
```

Figure 58 code: find_strongly_connected_components function

get_subgraphs(graph):

Takes each of the strongly connected components and copies it into a new graph so that the updates would not affect the original graph. And returns a list of copied strongly connected components.

```
def get_subgraphs(graph):
    list_strongly_connected_graphs = []
    for i in list(graph.subgraph(c)
                  for c in find_strongly_connected_components(graph)):
        subgraph = graph.subgraph(list(i))
        strongly_connected_graph = graph.__class__()
        strongly_connected_graph.add_edges_from((n, nbr, d)
                                                for n, nbrs in graph.adj.items() if n
in subgraph
                                                for nbr, d in nbrs.items() if nbr in
subgraph)
        strongly_connected_graph.graph.update(graph.graph)
        list_strongly_connected_graphs.append(strongly_connected_graph)
    return list_strongly_connected_graphs
```

Figure 59 code: get_subgraphs function

find_max_cycle_flow_sum(strongly_connected_graph)

Takes a strongly connected graph and returns the first maximum cycle flow in it and it returns the maximum cycle flow in a list form contains:

[max_flow_value, residual graph produced the maximum, edge checked to find this maximum]

```
def find_max_cycle_flow_sum(strongly_connected_graph):
    flows = []
    done = []
    strongly_connected_graph.add_node("S")
    for edge in strongly_connected_graph.edges:
        if edge not in done:
            strongly_connected_graph.add_edge(edge[0], "S",

capacity=strongly_connected_graph.get_edge_data(*edge)["capacity"])
            cycle_flow_sum = 0
            residual = nx.algorithms.flow.dinitz(strongly_connected_graph,
edge[1], "S")
            if residual.graph['flow_value'] != 0:
                cycle_flow_sum = sum(map(lambda e:
residual.get_edge_data(*e)["flow"] if
                residual.get_edge_data(*e)["flow"] > 0 else 0,
                                         strongly_connected_graph.edges))
            flows.append([cycle_flow_sum, residual, edge])
            strongly_connected_graph.remove_edge(edge[0], "S")
    max_cycle_flow = max(flows, key=lambda x: float(x[0]))
    return max_cycle_flow
```

Figure 60 code: find_max_cycle_flow_sum function

choose_maximum_flow(strongly_connected_graph)

this function takes the strongly connected graph and finds the maximum cycle flow and updates the graph and returns the maximum cycle flow.

```
def choose_max_cycle_flow(strongly_connected_graph):
    max_cycle_flow = find_max_cycle_flow_sum(strongly_connected_graph)
    update_directed_graph(strongly_connected_graph, max_cycle_flow)
    return max_cycle_flow
```

update_directed_graph(directed_graph, max_cycle_flow)

This function takes a strongly connected graph and the last cycle flow produced from it. And updates the capacities of the strongly connected graph by subtracting the positive flows in cycle flow from the corresponding graph edge capacity. And it returns the updated graph.

```
def update_directed_graph(directed_graph, max_cycle_flow):
    directed_graph[max_cycle_flow[2][0]][max_cycle_flow[2][1]]["capacity"] -=
max_cycle_flow[1].graph["flow_value"]
    if directed_graph[max_cycle_flow[2][0]][max_cycle_flow[2][1]]["capacity"] == 0:
        directed_graph.remove_edge(max_cycle_flow[2][0], max_cycle_flow[2][1])
    done = []
    for k in directed_graph.edges:
        if max_cycle_flow[1][k[0]][k[1]]["flow"] > 0 and "S" not in k:
            directed_graph[k[0]][k[1]]["capacity"] -=
max_cycle_flow[1][k[0]][k[1]]["flow"]
            if directed_graph[k[0]][k[1]]["capacity"] == 0:
                done.append((k[0], k[1]))
    for e in done:
        directed_graph.remove_edge(e[0], e[1])
    directed_graph.remove_node("S")
    return directed_graph
```

Figure 61 code:Update_directed_graph function

Analyse(file)

It takes the CSV file uploaded by the user and using the helper functions it performs the steps demonstrated in section 4.4.6.1 Script step by step functionality and it returns the 12 variables defined at the end of the same section to be stored in the report object fields.

```
def analyse(file):
    cycles_df = pd.DataFrame(columns=["cycle_flow", "cycle_flow_value",
"cycle_total_value"])
    total_local_transactions = 0
    local_transactions_df = pd.DataFrame(columns=["from", "to", "local_amount"])

    initial_transactions_df = pd.read_csv(file)
    initial_transactions_df = initial_transactions_df.groupby(['from',
'to'])['amount'].sum().reset_index()

    no_of_initial_transactions = len(initial_transactions_df.index)
    total_initial_amount = initial_transactions_df['amount'].sum()
    graph = create_directed_graph(initial_transactions_df)
    no_of_initial_countries = graph.number_of_nodes()

    for i in get_subgraphs(graph):
        while len(get_subgraphs(i)) > 0:
            chosen_cycle_flow_result = choose_max_cycle_flow(i)
            total_local_transactions += chosen_cycle_flow_result[0]

            for e in chosen_cycle_flow_result[1].edges:
                if "S" not in e and
chosen_cycle_flow_result[1].get_edge_data(*e)['flow'] > 0:
                    local_transactions_df = local_transactions_df.append({"from":
e[0], "to": e[1],

"local_amount":

chosen_cycle_flow_result[1].

get_edge_data(*e)['flow']},
ignore_index=True)
            local_transactions_df = local_transactions_df.append(
                {"from": chosen_cycle_flow_result[2][0], "to":
chosen_cycle_flow_result[2][1],
                 "local_amount": chosen_cycle_flow_result[1].graph['flow_value'], },
                ignore_index=True)
            local_transactions_df = local_transactions_df.groupby(['from',
'to'])['local_amount'].sum().reset_index()

            chosen_cycle_flow_edges = list(filter(lambda a: "S" not in a and

chosen_cycle_flow_result[1].get_edge_data(*a)[
                                                              'flow'] > 0,
                                                  chosen_cycle_flow_result[1].edges))

            chosen_cycle_flow_edges.append(chosen_cycle_flow_result[2])
            cycle = {"cycle_flow": chosen_cycle_flow_edges,
                     "cycle_flow_value":
chosen_cycle_flow_result[1].graph['flow_value'],
                     "cycle_total_value": chosen_cycle_flow_result[0]}
            cycles_df = cycles_df.append(cycle, ignore_index=True)

    no_of_local_countries = len(pd.unique(local_transactions_df[["from",
"to"]].values.ravel()))
    no_of_local_transactions = len(local_transactions_df.index)
    local_transactions_df = pd.merge(initial_transactions_df, local_transactions_df,
how='right',
                                     left_on=['from', 'to'],
                                     right_on=['from', 'to'])

    local_countries_df = local_transactions_df[['to',
'local_amount']].groupby(['to']).sum().reset_index()
    local_countries_df = local_countries_df.rename(columns={'to': 'country'})

    no_of_cycle_flows = len(cycles_df.index)
    no_complete_local_transactions =
local_transactions_df[local_transactions_df.amount ==

local_transactions_df.local_amount].shape[0]

    return cycles_df, local_transactions_df, total_local_transactions,
total_initial_amount, initial_transactions_df, \
           local_countries_df, no_of_initial_transactions, no_of_initial_countries,
no_of_local_countries, \
           no_of_cycle_flows, no_of_local_transactions, no_complete_local_transactions
```

4.4.7 Testing

Testing is a very crucial task in any software development process. It helps the developer challenge their code and understand the weak points in the structure and easily figure out any possible bug in the code. It also helps with the software scalability in the future as tests run when any new feature is added to the program to check if that affects the existing feature's functionality.

Unit-Tests is one of the most common testing types. In unit tests, we test each function individually and check if it returns what it is expected to do or not. In this application, I created unit tests for all the functions I wrote in the code. The test files are in **app/test.** To run the tests, you need to run this command in the terminal at the main directory **my_application:**

```
python manage.py test
```

There are 24 unit-tests in four files in the project and a directory **test_files**. As the application mainly handling CSV files, I needed to create this directory to provide test cases so that I can test my functions. the four files are **test_Analysis.py**, **test_forms.py**, **test_views.py**, and **test_urls.py**.

test_urls.py:

This class inherits from django.test SimpleTestcase. And it checks if the link between the URLs and views is working correctly. It simply resolves a URL and asserts the result to the view assigned to this URL as follows:

```
class TestUrls(SimpleTestCase):

    def test_home_url_resolves(self):
        url = reverse('home', args=[])
        self.assertEquals(resolve(url).func, views.home)

    def test_register_url_resolves(self):
        url = reverse('register', args=[])
        self.assertEquals(resolve(url).func, views.register)

    def test_show_profile_url_resolves(self):
        url = reverse('show_profile', args=[])
        self.assertEquals(resolve(url).func, views.show_profile)

    def test_create_report_url_resolves(self):
        url = reverse('create_report', args=[])
        self.assertEquals(resolve(url).func, views.create_report)

    def test_show_report_url_resolves(self):
        url = reverse('show_report', args=[uuid.uuid4()])
        self.assertEquals(resolve(url).func
```

Figure 62 code: URL tests

The class tests five URLs and it passed all of them.

test_forms.py

test_forms.py class is a very important unit test as the report form clean function is the one responsible for the uploaded file validation. To test the forms class, we need to pass files as an argument to be a test case and assert that to the form errors value. If the file is not valid the form shall have an error. Otherwise, the form errors shall be empty.

I created few test files in the directory with “ok” or fail” extensions so that we can test if the form validates correctly. Let us look at two examples:

Valid file:

```
def test_report_form_valid(self):
    form = ReportForm(data={})
    assert form.is_valid() is False

    file_path = os.path.join('app\\test\\test_files', 'file_1_ok.csv')

    with open(file_path, 'rb') as f:
        form = ReportForm(data={'name': "my_report"}, files={'data':
SimpleUploadedFile('file_1_ok.csv', f.read())})
    assert form.is_valid(), 'Invalid form, errors: {}'.format(form.errors)
```

Figure 63 test: testing forms with a valid file

Invalid file:

```
def test_report_form_invalid_not_csv(self):
    file_path = os.path.join('app\\test\\test_files', 'file_6_fail')

    with open(file_path, 'rb') as f:
        form = ReportForm(data={'name': "my_report"},
                          files={'data': SimpleUploadedFile('file_6_fail',
f.read())})
    self.assertFalse(form.is_valid())
    self.assertEquals(len(form.errors), 1)
```

Figure 64 test: testing forms with invalid file

test_Analysis.py

This class is the most complicated one in the program since it implements the main algorithm. To test its functions, I had to create mock values in JSON files imitating the expected output to see if it returns a good result. The mock values are stored as **JSON** and **CSV** in **test_files** directory. The setUp function loads the valid file and a JSON file stores the correct return graph for this CSV file. Then the test function transforms the JSON into a network graph and checks if it was isomorphic to the return value.

```
def setUp(self):

    self.file_path = os.path.join('app\\test\\test_files', 'file_1_ok.csv')
    self.d_graph_path = os.path.join('app\\test\\test_files', 'graph.json')

    with open(self.d_graph_path) as f:
       self.js_graph = json.load(f)
    f.close()

    self.main_graph = json_graph.node_link_graph(self.js_graph)

def test_create_directed_graph(self):
    self.assertTrue(nx.is_isomorphic(self.main_graph,
Analysis.create_directed_graph(pd.read_csv(self.file_path))))
```

Figure 65 Test: Analysis.py class: testing create_directed _graph function.

test_views.py

This class tests if the views redirect or render the correct HTML pages. It checks the response code and the page viewed. Some views require login authentication to get the correct page, and some do not. here are two examples for each type.

Without authentication:

```
def test_home(self):
    response = self.client.get(reverse("home"))
    self.assertEquals(response.status_code, 200)
    self.assertTemplateUsed(response, 'home.html')
```

Figure 66 test views: test home view

With authentication:

```
def test_show_profile(self):
    self.user = User.objects.create_user(username='testuser',
password='12345')
    login = self.client.login(username='testuser', password='12345')
    response = self.client.get(reverse("show_profile"))
    self.assertEquals(response.status_code, 200)
    self.assertTemplateUsed(response, 'profile.html')
```

Figure 67 test: views: test show profile view

Finally, when running the whole test, we get the program tested 24 unit-test over the four test classes and it responds with OK as every single unit test passes.

The testing output:

```
----------------------------------------------------------------------

Ran 24 tests in 1.790s


OK

Destroying test database for alias 'default'...

----------------------------------------------------------------------
```

Figure 68 Successful test output

Chapter 5

Case study

To implement the Algorithm, we need records of transaction that contains sender country, receiver country, amount so that we can find the matches among different records and be able to calculate the amount of possible local transactions in the network. In this chapter, we will implement the model on sample data from the World Bank Remittance Database.

5.1 The World Bank Remittance Database

Remittances are those transactions that migrants send back to their home countries from where they live and work. The world bank keeps track of the costs of worldwide remittances in the last 10 years. The data is surveyed and estimated based on the world bank measures (number of migrants, wages, work sectors, etc.)

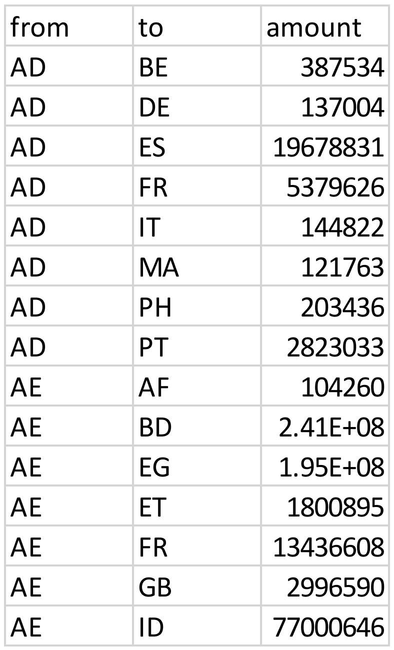
5.2 Case study: World Bank Bilateral Remittance Matrices 2017

The data is saved as a bilateral coloration matrix among all countries around the globe demonstrates the total amounts sent and received between each corridor of two countries. The sheet describes the Bilateral Remittance Estimates for 2017 using Migrant Stocks, Host Country Incomes, and Origin Country Incomes. The transaction values are in (**millions of US$**).

Remittance-receiving country (across) - Remittance-sending country (down)	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Antigua and Barbuda	Argentina	Armenia	Aruba	Australia	Austria	Azerbaijan
Afghanistan	0	0	0	0	0	0	0	0	0	0	0	0	0
Albania	0	0	0	0	0	0	0	0	0	0	0	0	0
Algeria	0	0	0	0	0	0	0	0	0	0	1	1	0
American Samoa	0	0	0	0	0	0	0	0	0	0	0	0	0
Andorra	0	0	0	0	0	0	0	1	0	0	0	0	0
Angola	0	0	0	0	0	0	0	0	0	0	0	0	0
Antigua and Barbuda	0	0	0	0	0	0	0	0	0	0	0	0	0
Argentina	0	0	0	0	0	0	0	0	2	0	5	9	0
Armenia	0	0	0	0	0	0	0	0	0	0	0	0	174
Aruba	0	0	0	0	0	0	0	0	0	0	0	0	0
Australia	3	4	2	0	0	0	0	11	3	0	0	121	0
Austria	1	4	2	0	0	0	0	1	7	0	10	0	1
Azerbaijan	0	0	0	0	0	0	0	0	7	0	0	0	0
Bahamas, The	0	0	0	0	0	0	0	0	0	0	0	0	0
Bahrain	0	0	0	0	0	0	0	0	0	0	0	0	0
Bangladesh	0	0	0	0	0	0	11	0	0	0	0	0	0
Barbados	0	0	0	0	0	0	0	0	0	0	0	0	0
Belarus	0	0	0	0	0	0	0	0	11	0	0	0	11
Belgium	0	7	30	0	0	0	0	0	13	0	6	29	1
Belize	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3 Bilateral Remittance Estimates for 2017

I took a sample from this table and transformed it into a CSV file matching the application validation requirements. The sample included **2500** transactions and **183** random countries. This sample is not accurate. It is only for the sake of testing the algorithm with a large number of nodes and see what it can do. The following table shows how new CSV file:



from	to	amount
AD	BE	387534
AD	DE	137004
AD	ES	19678831
AD	FR	5379626
AD	IT	144822
AD	MA	121763
AD	PH	203436
AD	PT	2823033
AE	AF	104260
AE	BD	2.41E+08
AE	EG	1.95E+08
AE	ET	1800895
AE	FR	13436608
AE	GB	2996590
AE	ID	77000646

Table 4 Remittance Estimates CSV file structure matching the validation requirements.

I used the application to create a report, and these were the initial data key points:

- The total amount of foreign transactions before running the analysis was **26201638783 USD**.
- The number of countries in the network of transactions was **183 countries**.
- The number of foreign transactions in the original network was **2500 transactions**.

The algorithm ran over the file and detecting the possible cycle flows in this network of transactions that allows replacing foreign transactions with local transactions and calculate the total of these local transactions. And these were the results:

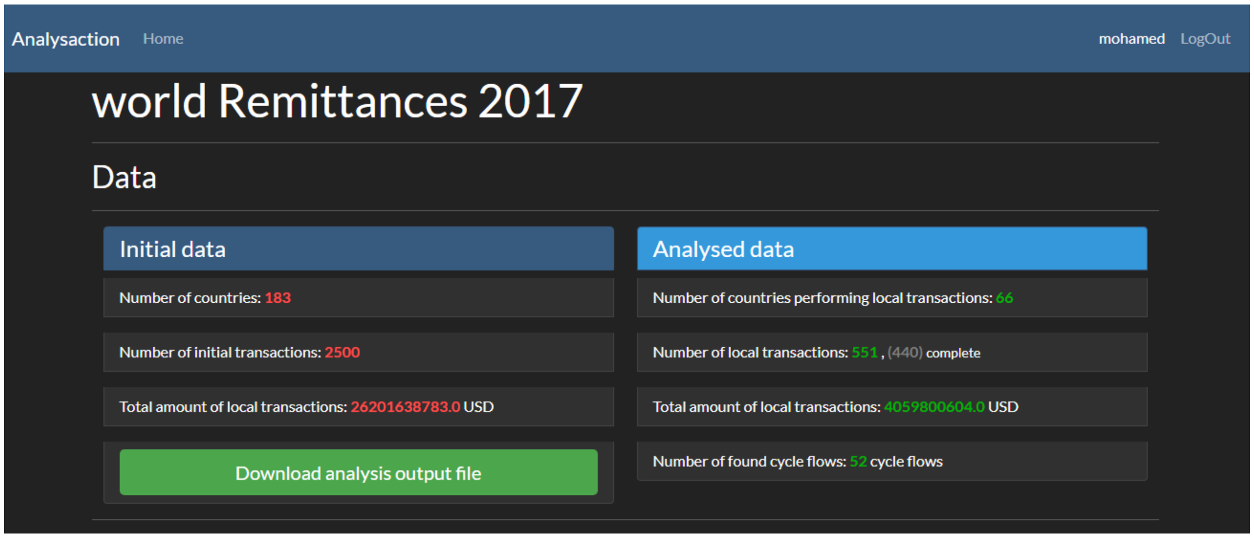
- The number of cycle flows detected in the network was **52 cycle flow**.

- The number of transactions that could participate in the cycle flows as local transactions were **551 transactions (22.04% of the initial number of transactions)**. This means there were **551** foreign transactions in the initial network that could be replaced, completely or partially, by local transactions and save the fees.

- Out of the **551 local transactions**. There were **440 complete local transactions (17.60% of the initial number of transactions)** that could replace the full amount of the corresponding foreign transaction.

- the number of countries that could perform local transactions was **66 countries (35.75% of the initial number of transactions)**.

- The total amount of local transactions according to the analysis **4059800604 USD (15.49% of the initial total amount of foreign transactions).** With an average transaction fee of **3%,** the tool can save the global transaction network around **121794018.12 USD.**



world Remittances 2017

Data

Initial data	Analysed data
Number of countries: 183	Number of countries performing local transactions: 66
Number of initial transactions: 2500	Number of local transactions: 551 , (440) complete
Total amount of local transactions: 26201638783.0 USD	Total amount of local transactions: 4059800604.0 USD
	Number of found cycle flows: 52 cycle flows

Figure 69 Screenshot of World Remittances 2017 sample report

The application also provides an output excel file including all the needed analysis characteristics describing in detail the initial transactions data and what are the detected cycles in the network and the amounts reduced in every single transaction. And the amounts to be transacted locally in each country. These report files are attached with my thesis work:

- **bilateralremittancematrix2017Apr2018.xlsx**
  The original world bank bilateral remittance matrix file.

- **world Remittances 2017_ intial_transactions_sample.csv**
  sample of the remittance's transactions transformed into CSV format matching the application validation requirements. You can create a report on the application using this file. But as it has a large number of nodes it may take some time to load (around 4 minutes)

- **world Remittances 2017_output.xlsx**
  The application's report output describing the analysis outcomes in detail.

The files will be in the directory:

```
Mohamed_Elsaadany_Thesis_work/world_bank_remittances_case_study_files/
```

Chapter 6

Conclusion

The aim of this thesis work was trying to find an alternative approach to money transfers that minimizes the transaction fees and develop an analysis model that can solve the problem. In addition to developing a web application that performs this model analysis on arbitrary input data and checks if the approach is effective. During working on this thesis project, I encountered a lot of challenges. Especially in theoretical aspects. However, I could grasp a quite decent amount of knowledge in advanced topics such as graph theory and network flows. On the other hand, technically, I learned a lot about the frameworks and libraries to develop the application such as Django, Bootstrap, and Highcharts libraries.

I propose an approach that can solve the problem of minimizing the foreign transaction fees by replacing foreign transactions with local transactions if it does not affect senders' and receivers' balances. The idea was in a network of foreign transactions if there exists a flow amount that circulates in the network from one country to the same country again. It means that each country in this cycle can perform this transaction locally and save the transaction fees.

I designed the model and tried it on several test cases and got good results in each. However, the model is not complete and has a lot of limitations.

Limitations

- The algorithm is not proven to produce an optimal solution. Although it gets good results, I need to have solid proof that it works on any arbitrary network of transactions. There might be some test cases that break the model's logic.
- Some enhancements are needed in terms of the input, output, and model specifications. For instance:

    - The input shall include the fees of each transaction so that we can estimate more accurate fees reduction.

    - The model shall handle more than one currency and be flexible in calculating real-time exchange rates without additional fees.

    - The transaction time factor shall be strongly in consideration. A transaction shall not wait for days till it finds a match in a cycle flow in a dynamic network. The case study I tried in this thesis project was on estimated historical data spread in almost one year time. That was only a demo for the sake of explanation.

Future work

For future work, I need to start coming over the limitations in the last section to have a solid proven scalable solution that can be implemented in the industry in real life. At the same time, I have a less complex application idea that can use the model as it is now.

The new application would do the same purpose but with individual interest of one transaction and not serving the whole network. In other words, trying to find one cycle flow that serves only this transaction not the best cycles in the whole network.

Users can publish their transactions on a public pool and define sending and receiving country and the amount of the transaction. The application would match the users whose transactions can form a cycle flow and inform them of possible flow. Once all parties in the cycle flow confirm performing the cycle flow. The transactions occur locally in each country free of foreign charge.

Bibliography

[1] "Maximum flow problem," *Wikipedia*. May 03, 2021. Accessed: May 25, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Maximum_flow_problem&oldid=1021194371

[2] "Reference — NetworkX 2.5 documentation." https://networkx.org/documentation/stable/reference/index.html (accessed May 25, 2021).

[3] "networkx.algorithms.components.kosaraju_strongly_connected_components — NetworkX 2.5 documentation." https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.kosaraju_strongly_connected_components.html (accessed May 26, 2021).

[4] "networkx.algorithms.flow.dinitz — NetworkX 2.5 documentation." https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.dinitz.html (accessed May 26, 2021).

[5] "Django documentation | Django documentation | Django." https://docs.djangoproject.com/en/3.2/ (accessed May 22, 2021).

[6] "Models | Django documentation | Django." https://docs.djangoproject.com/en/3.2/topics/db/models/ (accessed May 22, 2021).

[7] "Writing views | Django documentation | Django." https://docs.djangoproject.com/en/3.2/topics/http/views/ (accessed May 22, 2021).

[8] "Templates | Django documentation | Django." https://docs.djangoproject.com/en/3.2/topics/templates/ (accessed May 22, 2021).

[9] "URL dispatcher | Django documentation | Django." https://docs.djangoproject.com/en/3.2/topics/http/urls/ (accessed May 22, 2021).

[10] "Forms | Django documentation | Django." https://docs.djangoproject.com/en/3.2/ref/forms/ (accessed May 22, 2021).

[11] "The Django admin site | Django documentation | Django." https://docs.djangoproject.com/en/3.2/ref/contrib/admin/ (accessed May 22, 2021).

[12] "Databases | Django documentation | Django." https://docs.djangoproject.com/en/3.2/ref/databases/ (accessed May 22, 2021).

[13] M. O. contributors Jacob Thornton, and Bootstrap, "Bootstrap." https://getbootstrap.com/ (accessed May 22, 2021).

[14] "Bootswatch: Free themes for Bootstrap." https://bootswatch.com/ (accessed May 22, 2021).

[15] "Bootswatch: Darkly." https://bootswatch.com/darkly/ (accessed May 22, 2021).

[16] "Highcharts Documentation | Highcharts." https://highcharts.com/docs/index (accessed May 22, 2021).

[17] "Sunburst | Highcharts." https://highcharts.com/docs/chart-and-series-types/sunburst-series (accessed May 22, 2021).

[18] "uuid — UUID objects according to RFC 4122 — Python 3.9.5 documentation." https://docs.python.org/3/library/uuid.html (accessed May 22, 2021).

[19] “Many-to-one relationships | Django documentation | Django.” https://docs.djangoproject.com/en/3.2/topics/db/examples/many_to_one/ (accessed May 22, 2021).
[20] “ISO 3166-1 alpha-2,” *Wikipedia*. May 17, 2021. Accessed: May 24, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ISO_3166-1_alpha-2&oldid=1023620933
[21] C. Theune, *pycountry: ISO country, subdivision, language, currency and script definitions and their translations*.
[22] “Migration and Remittances Data,” *World Bank*. https://www.worldbank.org/en/topic/migrationremittancesdiasporaissues/brief/migration-remittances-data (accessed Mar. 01, 2021).

List of Figures

List of Tables