Quantification of Water Footprint of Trade

Anav Vora, Hari Dave, Yi-Chia Chang
Sep. 22, 2023

Introduction

Food trade involves the transfer of food from one country to another. According to the water footprint concept introduced by Hoekstra (2003), this transfer is equivalent to transferring the water required for producing that food, termed the virtual water content of a product. The virtual water content of a product depends on the location at which the product is produced. This is because growing a crop in an arid country might require more water than growing the same crop in a humid one.

The water footprint of food trade is the total virtual water transferred across all countries across all food trades. Here, we quantify the water footprint of food aid for the year 2005, which looks specifically at the total volume of virtual water transferred across countries for food transferred in the form of aid. We also examine the water saving achieved by the food aid transfers, i.e., the difference between the virtual water content of a product if it were to be grown in the recipient country vs. the donor country. Lastly, we identify the food products accounting for the largest portion of the water footprint of food aid, the food products resulting in the greatest water saving, and trade links across which large volumes of virtual water are traded.

Methodology

Data collection

The food aid trade data is obtained from the 'Food Aid Information System' (FAIS) portal of the World Food Programme (WFP). The water footprint for the commodities for each nation is retrieved from Mekonnen and Hoekstra (2010a) and Mekonnen and Hoekstra (2010b). From the trade data, only the direct transfer records for the year 2005 are analyzed for this study.

Footprint Calculation

A dictionary file is generated, mapping the description of the commodity traded from the '2005 Tonnage & IRMAt' file to the product descriptions provided by Mekonnen and Hoekstra (2010a, 2010b). In case no appropriate descriptions are found for the products (e.g. biscuits, canned fish, etc.), those particular trades are excluded from the analysis. For the goods with suitable descriptions, The country-average water footprints corresponding to the commodities are retrieved from Mekonnen and Hoekstra (2010a, 2010b) to estimate the water footprints of the trade. This analysis uses the net summation of green and blue water footprints for agricultural goods. In case multiple commodities are matched with the product description of the traded commodity, the median water footprint is considered for further calculations. To find the net water footprint traded, the water footprint of the product for the donor country is multiplied by the tonnage values. The trades involving donor countries with missing footprint information, along with trades having NGOs, "OTHER', WFP, PRIVATE or UNITED NATIONS as donors, are ignored. For the trades having 'European Community' as donors, the average water footprint for 25 European Union countries is considered for the analysis.

We also calculate the aid related water savings as defined in Jackson et al. (2015).

$$GWS = \sum_{i,e,c} T_{c,e\rightarrow i} \times (VWC_{c,i} - VWC_{c,e})$$

Here,
$VWC_{c,i}$: Virtual Water Content for commodity c, in country i
$T_{c, e\rightarrow i}$: Trade amount of the commodity c, exporting from country e to country i

Results and Discussion

The net water footprint traded for the year 2005 is estimated to be **8.72 km^3**, which aligns with the calculated water footprint from Jackson et al. (2015) of **10 km^3**. Fig. 1 shows a pie chart that breaks down the total water footprint by product. We found that wheat transfers correspond to the greatest water footprint.

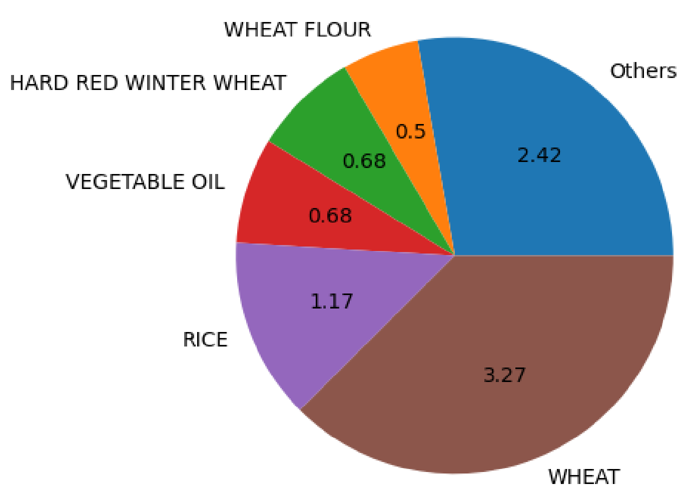

Fig. 1: Virtual Water Trade (in km^3)

Next, we look at the trade links across countries participating in food aid. Fig. 2 shows the trade links that account for the top 5% of the global water footprint of food aid. We note that donations from the US account for large volumes of virtual water transfers. Table 1 summarizes the top 5 exporters and top 5 importers for the year 2005. Table 2 presents the top 5 links with the highest virtual water trade values.

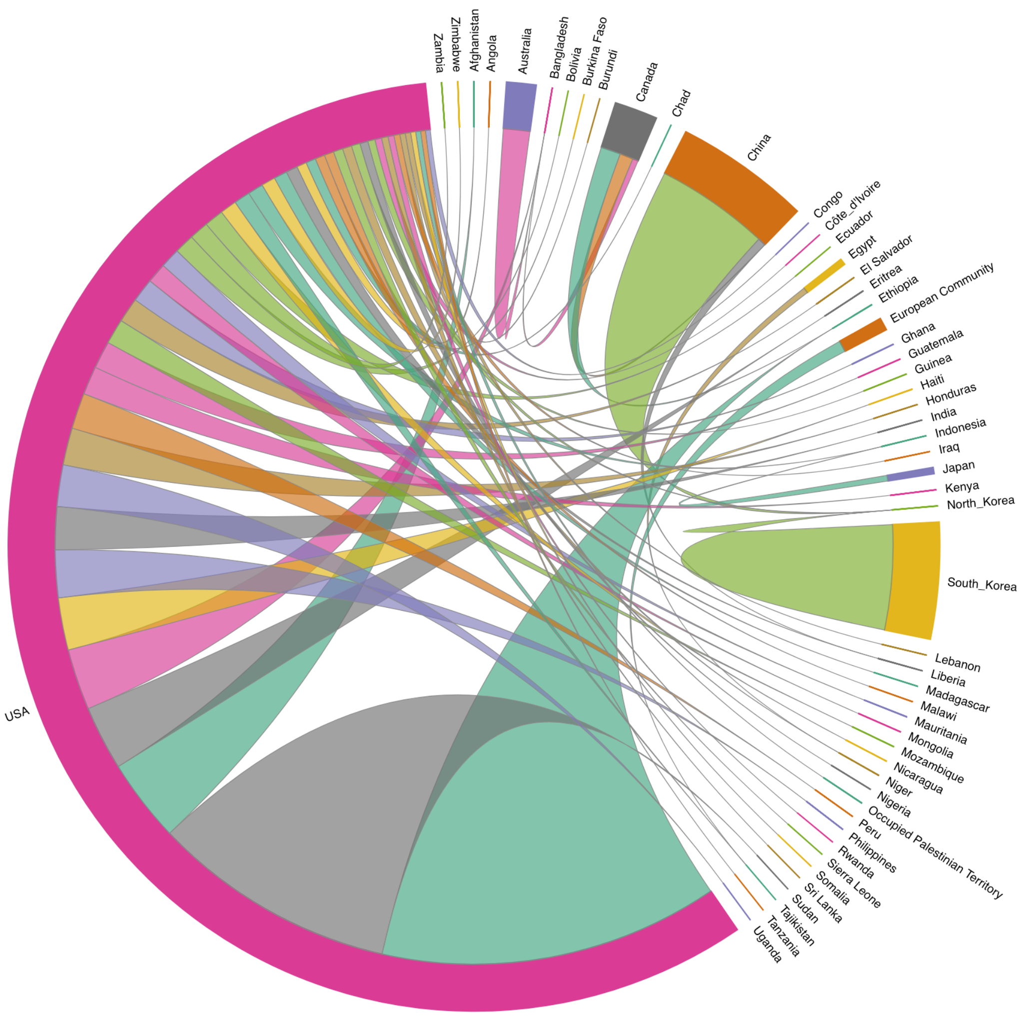

Fig. 2: Trade links across which the top 5% of the virtual water flux occurs

Table 1: Summary of top donors and recipients in terms of virtual water volume

Top 5 Donors			Top 5 Recipients		
Country	Virtual Water Exported (km^3)	Percent (%)	Country	Virtual Water Imported (km^3)	Percent (%)
US	6.78	77.76	Ethiopia	1.58	18.17
China	0.54	6.25	Sudan	1.09	12.47
South Korea	0.46	5.28	North Korea	1.02	11.68
Canada	0.33	3.77	Bangladesh	0.42	4.86
Japan	0.23	2.69	Afghanistan	0.38	4.3

Table 2: Links with the most virtual water trade

Link	Commodity	Virtual Water Traded (km^3)
US → Ethiopia	Wheat	0.97
US → Sudan	Wheat	0.75
South Korea → North Korea	Rice	0.34
US → Ethiopia	Hard Red Winter Wheat	0.30
China → North Korea	Wheat Flour	0.28

The aid-related water savings is estimated to be 11.69 km^3. Fig. 3 shows the commodity-wise water savings occurring as a result of food aid transfers. Table 3 summarizes the trade links resulting in maximum water saving as well as trade links across which maximum virtual water loss occurs. The export of Soya Oil from the European Union to Tajikistan saves the most water. While, the export of wheat from Australia and the United States and Australia to Sudan and Bangladesh shows the most negative water saving.

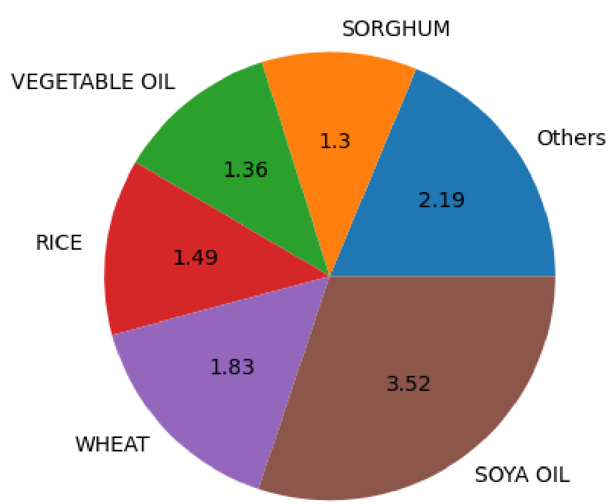

Fig. 3: Virtual Water Savings (in km^3)

Table 3: Summary of top positive and negative water saving trades

Top 5 Positive Water Saving Trades		
Link	Commodity	Virtual Water Savings (km^3)
EU → Tajikistan	Soya Oil	3.49
US → Ethiopia	Wheat	1.12
US → Tajikistan	Vegetable Oil	0.98
US → Sudan	Sorghum	0.82
US → Ethiopia	Hard Red Winter Wheat	0.35

Top 5 Negative Water Saving Trades		
Link	Commodity	Virtual Water Savings (km^3)
US → Sudan	Wheat	-0.058
Australia → Bangladesh	Wheat	-0.038
US → Bangladesh	Common Wheat	-0.036
US → Bangladesh	Hard Red Winter Wheat	-0.023
US → Mozambique	Wheat	-0.015

References

- Food Aid Information System Online Database. Available online: http://www.wfp.org/fais/
- Hoekstra, A. Y. (2003). Virtual water trade: proceedings of the international expert meeting on virtual water trade, Delft, The Netherlands, 12-13 December 2002, Value of Water Research Report Series No. 12.
- Jackson, N.; Konar, M.; Hoekstra, A.Y. The Water Footprint of Food Aid. Sustainability 2015, 7, 6435-6456. https://doi.org/10.3390/su7066435
- Mekonnen, M.M. & Hoekstra, A.Y. (2010a) The green, blue and grey water footprint of farm crops and derived crop products. Value of Water, 47:
- Mekonnen, M.M. & Hoekstra, A.Y. (2010b) The green, blue and grey water footprint of farm animals and animal products. Value of Water, 48:

Appendix

All codes and figures can be found in the GitHub repository:
https://github.com/yichiac/CEE598_Globalization_of_Water

Code calculating the virtual water footprints

```
import pandas as pd
import numpy as np
import os
import pickle as pkl
import matplotlib.pyplot as plt

def read_WF_dataset(file_name, is_agri_data = 0):   ##### Reading and
saving data to numpy arrays
    df_read = pd.read_excel(file_name, header=None, sheet_name=1)

    if(is_agri_data == 1):
        data_arr = df_read.iloc[6:, :].to_numpy()
        country_row = df_read.iloc[3, :].to_numpy()
        label_row = df_read.iloc[4, :].to_numpy()
    else:
        data_arr = df_read.iloc[4:, :].to_numpy()
        country_row = df_read.iloc[2, :].to_numpy()
        label_row = df_read.iloc[3, :].to_numpy()


    return data_arr, country_row, label_row

def find_WF(product, country_export, country_import, data_arr,
country_row, product_type):
    if(product in list(data_arr[:, 3]) or product in list(data_arr[:,
2])): ##### Check if the product esists in the description
        product_found = 1   ############ product_type=1: Agri products,
product_type=0: Animal products
        if(product_type == 1):
            product_idx = np.argwhere(data_arr[:,
3]==product).reshape(-1)[0]   ### Matching the description with the
dictionary term
        else:
```

```
            product_idx = np.argwhere(data_arr[:,
2]==product).reshape(-1)[0]

        ### Exporters Footprint
        country_export_idx =
np.argwhere(country_row==country_export).reshape(-1)[-1] ### Last Column
is country average

        if(product_type==1): #### Crops and Agri
            WF_export = np.nansum([data_arr[product_idx,
country_export_idx],data_arr[product_idx+1, country_export_idx]]) ###
Adding green and blue
        else:
            WF_export = data_arr[product_idx, country_export_idx+3] ###
Taking Weighted avg

        ### Importers Footprint
        if(product_type==0 and country_import == "Occupied Palestinian
Territory"):
            country_import_temp = "Israel"
            country_import_idx =
np.argwhere(country_row==country_import_temp).reshape(-1)[-1] ### Last
Column is country average
        else:
            country_import_idx =
np.argwhere(country_row==country_import).reshape(-1)[-1] ### Last Column
is country average

        if(product_type==1): #### Crops and Agri
            WF_import = data_arr[product_idx,
country_import_idx]+data_arr[product_idx+1, country_import_idx] ### Adding
green and blue
        else:
            WF_import = data_arr[product_idx, country_import_idx+3] ###
Taking Weighted

    else:
        product_found = 0
        WF_export, WF_import = 0, 0
    return product_found, WF_export, WF_import
```

```python
def append_EU_data(data_arr, product_type, country_row, EU_countries):
########## Appending EU average data to the datasets

    column_arr = np.full((len(data_arr[:, 0]), len(EU_countries)), np.nan)

    for i in range(len(EU_countries)):
        nation_idx = np.argwhere(country_row ==
EU_countries[i]).reshape(-1)[-1]
        if(product_type == 1):
            column_arr[:, i] = np.copy(data_arr[:, nation_idx])
        else:
            column_arr[:, i] = np.copy(data_arr[:, nation_idx+3])

    if(product_type==1):    ############ product_type=1: Agri products,
product_type=0: Animal products
        data_arr = np.hstack((data_arr, np.nanmean(column_arr, axis =
1).reshape(-1, 1)))
    else:
        data_arr = np.hstack((data_arr, np.nanmean(column_arr, axis =
1).reshape(-1, 1)*np.ones((len(data_arr[:, 0]), 4))))
    country_row = np.hstack((country_row, np.array(['European Community'],
dtype=object)))

    return data_arr, country_row

def pie_chart(sizes, legends, save_file_name, save_loc = '.\\'): ### To
plot the product-wise pie-chart

    temp_arr = np.hstack((sizes.reshape(-1, 1), legends.reshape(-1, 1)))

    temp_arr = temp_arr[temp_arr[:, 0].argsort()]

    # print(np.nansum(temp_arr[:, 0]))

    temp_arr[-6, 1] = 'Others'
    temp_arr[-6, 0] = np.nansum(temp_arr[:-5, 0])

    def absolute_value(val):
        a  = np.round((val/100)*np.nansum(temp_arr[-6:, 0]), 2)
```

```
        return a
    # print(temp_arr[-6:, :])

    plt.figure()
    plt.pie(temp_arr[-6:, 0], labels=temp_arr[-6:, 1],
autopct=absolute_value) #autopct='%1.1f%%'
    plt.savefig(save_loc+save_file_name+'.png')
    plt.close()

##########################################################################
###########

agri_data =
'.\\Report47-Appendix\\Report47-Appendix-II\\Report47-Appendix-II.xlsx'
animal_data = '.\\Report48-Appendix-V\\Report48-Appendix-V.xlsx'

trade_yr = [2005] #### Considering 2005 data

trade_file_loc = '.\\WFP-0000018924\\Historical Data Files\\Tonnage &
IRMAs IRMA Energy IRMAt Energy FINAL\\'

product_dict = '.\\Dictionary_Products.csv' ### Product Dictionary file

product_dict_data = pd.read_csv(product_dict, header=None).to_numpy()[1:,
1:]

ignore_donors = ['NGOs', 'OTHER', 'WFP', 'PRIVATE', 'UNITED NATIONS'] ##
Ignoring these donors

########### Saving the datasets into binary files to speed up the code

save_binary = 0
if(save_binary==1):
    os.makedirs('.\\binary_files\\', exist_ok=True)
    agri_data_arr, agri_country_row, agri_label_row =
read_WF_dataset(agri_data, is_agri_data=1)
    animal_data_arr, animal_country_row, animal_label_row =
read_WF_dataset(animal_data, is_agri_data=0)

    with open('.\\binary_files\\agri_data_arr.pkl', 'wb') as pkl_file:
```

```
        pkl.dump(agri_data_arr, pkl_file)
    with open('.\\binary_files\\agri_country_row.pkl', 'wb') as pkl_file:
        pkl.dump(agri_country_row, pkl_file)
    with open('.\\binary_files\\agri_label_row.pkl', 'wb') as pkl_file:
        pkl.dump(agri_label_row, pkl_file)

    with open('.\\binary_files\\animal_data_arr.pkl', 'wb') as pkl_file:
        pkl.dump(animal_data_arr, pkl_file)
    with open('.\\binary_files\\animal_country_row.pkl', 'wb') as
pkl_file:
        pkl.dump(animal_country_row, pkl_file)
    with open('.\\binary_files\\animal_label_row.pkl', 'wb') as pkl_file:
        pkl.dump(animal_label_row, pkl_file)

######## Reads the binary file

with open('.\\binary_files\\agri_data_arr.pkl', 'rb') as pkl_file:
    agri_data_arr = pkl.load(pkl_file)
with open('.\\binary_files\\agri_country_row.pkl', 'rb') as pkl_file:
    agri_country_row = pkl.load(pkl_file)
with open('.\\binary_files\\agri_label_row.pkl', 'rb') as pkl_file:
    agri_label_row = pkl.load(pkl_file)

with open('.\\binary_files\\animal_data_arr.pkl', 'rb') as pkl_file:
    animal_data_arr = pkl.load(pkl_file)
with open('.\\binary_files\\animal_country_row.pkl', 'rb') as pkl_file:
    animal_country_row = pkl.load(pkl_file)
with open('.\\binary_files\\animal_label_row.pkl', 'rb') as pkl_file:
    animal_label_row = pkl.load(pkl_file)

########## Taking average for the European Union countries

EU_countries = ['Germany', 'France', 'Italy', 'Netherlands', 'Belgium',
'Luxembourg',
                'Denmark', 'Ireland', 'United Kingdom', 'Greece', 'Spain',
'Portugal',
                'Austria', 'Finland', 'Sweden', 'Czech Republic',
'Cyprus', 'Estonia', 'Latvia',
                'Lithuania', 'Hungary', 'Malta', 'Poland', 'Slovenia',
'Slovakia']
```

```
agri_data_arr, agri_country_row = append_EU_data(agri_data_arr, 1,
agri_country_row, EU_countries)
animal_data_arr, animal_country_row = append_EU_data(animal_data_arr, 0,
animal_country_row, EU_countries)

###############

for i in range(len(trade_yr)):
    trade_data = pd.read_csv(trade_file_loc+str(trade_yr[i])+' Tonnage &
IRMAt.csv', header=0).to_numpy()

    write_str = np.array(['year', 'country_export', 'country_import',
'Product', 'Commodity Cereals Non Cereals', 'Tonnage',
                          'Exporter Footprint (per unit)', 'Importers
Footprint (per unit)', 'Water Footprint Traded (m3)',
                          'Water Saved (m3)'], dtype=object).reshape(1,
-1)
    for ii in range(len(trade_data[:, 0])):

        if(trade_data[ii, 5] == 'Direct Transfer' and trade_data[ii, 1]
not in ignore_donors):
            country_export = trade_data[ii, 1]
            country_import = trade_data[ii, 2]

            print(country_export, country_import)

            country_export_temp = country_export.split(',') ####
Extracting countries which are named as 'Country_name, the'
            country_import_temp = country_import.split(',') ####
Extracting countries which are named as 'Country_name, the'
            try:
                if(country_export_temp[1] == ' the'):
                    country_export = country_export_temp[0]
            except:
                pass

            try:
                if(country_import_temp[1] == ' the'):
                    country_import = country_import_temp[0]
```

```
            except:
                pass

            ##### Matching the names of the countries across the databases

            if(country_import == "Democratic People's Republic of Korea
(DPRK)"):
                country_import = "Korea, Democratic People's Republic of"
            elif(country_import == "Democratic Republic of the Congo
(DRC)"):
                country_import = "Congo, Democratic Republic of the"
            elif(country_import == "São Tomé and Principe"):
                country_import = "Sao Tome and Principe"
            elif(country_import == "Central African Republic "):
                country_import = "Central African Republic"
            elif(country_import == "Timor-Leste"):
                country_import = "East Timor"
            elif(country_import == "Republic of Moldova"): #, the
                country_import = "Moldova"

            if(country_export == "Lybian Arab Jamahiriya"):
                country_export = "Libyan Arab Jamahiriya"
            elif(country_export=="Taiwan, Province of China"):
                country_export = "China"
            elif(country_export == "Democratic Republic of the Congo
(DRC)"):
                country_export = "Congo, Democratic Republic of the"
            elif(country_export == "Republic of Korea"): ## , the
                country_export = "Korea, Republic of"

            ###########################

            tonnage = trade_data[ii, 9]

            cereal_non_cereal = trade_data[ii, 7]

            Product_name = trade_data[ii, 6]
```

```
            idx_product = np.argwhere(product_dict_data[:,
0]==Product_name).reshape(-1)[0] ##### Matching the Product names with the
Annexes

            ##### Removing nan values from the dictionary

            idx_nan =
np.argwhere(pd.isna(product_dict_data[idx_product])==True).reshape(-1)

            product_idx_temp = np.copy(product_dict_data[idx_product])
            product_idx_temp = np.delete(product_idx_temp, idx_nan)

            ###################

            ######### Finding the water footprint of the commodities

            WF_export_temp = []
            WF_import_temp = []

            for kk in range(1, len(product_idx_temp)):

                product_found, WF_export, WF_import =
find_WF(product_idx_temp[kk], country_export, country_import,
                                                                        agri_data_arr,
agri_country_row, 1)

                if(product_found==0):
                    product_found, WF_export, WF_import =
find_WF(product_idx_temp[kk], country_export, country_import,

animal_data_arr, animal_country_row, 0)
                WF_export_temp.append(WF_export)
                WF_import_temp.append(WF_import)

            try:
                WF_export = np.nanmedian(WF_export_temp) ### Taking the
median to account for skewed data
                WF_import = np.nanmedian(WF_import_temp) ### Taking the
median to account for skewed data
```

```
                # WF_export = np.nanmean(WF_export_temp) ### mean can also
be considered
                # WF_import = np.nanmean(WF_import_temp) ### mean can also
be considered
            except:
                product_found, WF_export, WF_import = 0, 0, 0

            ##### Saving data into an array

            write_str = np.vstack((write_str, np.array([trade_yr[i],
country_export, country_import, Product_name,
                                                        cereal_non_cereal,
tonnage, WF_export, WF_import, WF_export*tonnage,

(WF_import-WF_export)*tonnage], dtype=object).reshape(1, -1)))

    ######## Outputting array as a csv file
    os.makedirs('.\\calculated_trade_footprint\\', exist_ok = True)

pd.DataFrame(write_str).to_csv('.\\calculated_trade_footprint\\Trade_footp
rint_'+str(trade_yr[i])+'.csv', header=None, index=None)

    footprint_data =
pd.read_csv('.\\calculated_trade_footprint\\Trade_footprint_'+str(trade_yr
[i])+'.csv', header=0).to_numpy()

    #### Creating the product-wise footprint and water savings pie-chart
    unique_products = np.unique(footprint_data[:, 3])

    product_water_footprint = np.full((len(unique_products)), np.nan)
    product_water_savings = np.full((len(unique_products)), np.nan)

    zero_footprints = np.argwhere(np.logical_or(footprint_data[:, -3]==0,
footprint_data[:, -4]==0)).reshape(-1)
    nan_footprints = np.argwhere(np.logical_or(pd.isna(footprint_data[:,
-3])==True, pd.isna(footprint_data[:, -4])==True)).reshape(-1)

    #### Removing the missing values from the data
    water_saved_data = np.copy(footprint_data[:, -1]).reshape(-1, 1)
    water_saved_data[zero_footprints, :] = np.nan
```

```
    water_saved_data[nan_footprints, :] = np.nan

    for iii in range(len(unique_products)):
        idx_unique_product = np.argwhere(footprint_data[:, 3] ==
unique_products[iii]).reshape(-1)

        product_water_footprint[iii] =
np.nansum(footprint_data[idx_unique_product, -2])
        product_water_savings[iii] =
np.nansum(water_saved_data[idx_unique_product, 0])

    # print(unique_products, product_water_footprint/1e9,
product_water_savings/1e9)

    pie_chart(product_water_footprint/1e9, unique_products,
'water_footprint_products', '.\\calculated_trade_footprint\\')
    pie_chart(product_water_savings/1e9, unique_products,
'water_savings_products', '.\\calculated_trade_footprint\\')
```

Code to analyze the footprints

```
import numpy as np
import pandas as pd

read_file_data =
pd.read_csv('.\\calculated_trade_footprint\\Trade_footprint_2005.csv',
header=0).to_numpy()

def find_net_links(read_file_data):
    out_df = np.array(('Donor', 'Recipient', 'Commodity', 'VWT', 'Water
Saved'), dtype=object).reshape(1, -1)

    for i in range(len(read_file_data)):
        # print(out_df)
        idx_trade = np.argwhere(np.logical_and(np.logical_and(out_df[:, 0]
== read_file_data[i, 1],
                                                                  out_df[:, 1]
== read_file_data[i, 2]),
```

```
                                                                          out_df[:, 2]
== read_file_data[i, 3])).reshape(-1)
        if(len(idx_trade)==0):
            temp_array = read_file_data[i, 1:4].reshape(1, -1)
            temp_array = np.hstack((temp_array, read_file_data[i,
-2:].reshape(1, -1)))
            out_df = np.vstack((out_df, temp_array.reshape(1, -1)))
        else:
            out_df[idx_trade, -2] = np.nansum([out_df[idx_trade,-2],
read_file_data[i, -2]])
            out_df[idx_trade, -1] = np.nansum([out_df[idx_trade,-1],
read_file_data[i, -1]])


pd.DataFrame(out_df).to_csv('.\\calculated_trade_footprint\\Link_commodity
_analysis.csv', header=None, index=None)

def find_countrywise_export(read_file_data):
    out_df = np.array(('Donor','VWT'), dtype=object).reshape(1, -1)

    for i in range(len(read_file_data)):
        # print(out_df)
        idx_trade = np.argwhere(out_df[:, 0] == read_file_data[i,
1]).reshape(-1)
        if(len(idx_trade)==0):
            temp_array = np.array(['country', 0.0], dtype=object)
            temp_array[0] = read_file_data[i, 1]
            temp_array[1] = read_file_data[i, -2]
            out_df = np.vstack((out_df, temp_array.reshape(1, -1)))
        else:
            try:
                out_df[idx_trade, -1] = np.nansum([out_df[idx_trade,-1],
read_file_data[i, -2]])
            except:
                if(out_df[idx_trade, -1] == 'nan'):
                    out_df[idx_trade, -1] = read_file_data[i, -2]

    percent_column = np.copy(out_df[:, -1])
    percent_column[1:] = out_df[1:, -1]*100/np.nansum(out_df[1:, -1])
```

```
    out_df = np.hstack((out_df, percent_column.reshape(-1, 1)))


pd.DataFrame(out_df).to_csv('.\\calculated_trade_footprint\\Donors_VWT.csv
', header=None, index=None)

def find_countrywise_import(read_file_data):
    out_df = np.array(('Recipient','VWT'), dtype=object).reshape(1, -1)

    for i in range(len(read_file_data)):
        # print(out_df)
        idx_trade = np.argwhere(out_df[:, 0] == read_file_data[i,
2]).reshape(-1)
        if(len(idx_trade)==0):
            temp_array = np.array(['country', 0.0], dtype=object)
            temp_array[0] = read_file_data[i, 2]
            temp_array[1] = read_file_data[i, -2]
            out_df = np.vstack((out_df, temp_array.reshape(1, -1)))
        else:
            try:
                out_df[idx_trade, -1] = np.nansum([out_df[idx_trade,-1],
read_file_data[i, -2]])
            except:
                if(out_df[idx_trade, -1] == 'nan'):
                    out_df[idx_trade, -1] = read_file_data[i, -2]

    percent_column = np.copy(out_df[:, -1])
    percent_column[1:] = out_df[1:, -1]*100/np.nansum(out_df[1:, -1])

    out_df = np.hstack((out_df, percent_column.reshape(-1, 1)))


pd.DataFrame(out_df).to_csv('.\\calculated_trade_footprint\\Recipient_VWT.
csv', header=None, index=None)

find_net_links(read_file_data)
find_countrywise_export(read_file_data)
find_countrywise_import(read_file_data)
```

Code for Circos plots
Python

```
import pandas as pd
import numpy as np

# read data
df = pd.read_csv('AWF_2005.csv')
df.dropna(inplace=True)

# aggregate actual water footprint with same paris of countries
df_group = df.groupby(['country_export',
'country_import']).agg('sum').reset_index()

# drop trade flux under the threshold to get clear a chord plot
awf = df['Actual WF (m3)']
awf = awf.to_numpy()
threshold = np.quantile(awf, 0.95)
df_group = df_group[df_group['Actual WF (m3)'] > threshold]

# get the list of countries meeting trade flux threshold
exporters = df_group['country_export'].unique()
importers = df_group['country_import'].unique()
all_countries = np.unique(np.concatenate((exporters, importers),
axis=0))

# create trade flux matrix
matrix = pd.DataFrame(0, index=all_countries,
columns=all_countries)

for index, row in df_group.iterrows():
export_country = row['country_export']
import_country = row['country_import']
```

```python
value = row['Actual WF (m3)']
matrix.at[export_country, import_country] = value

matrix.to_csv('output_matrix.csv')
```

R

```r
library(chorddiag)

data = read.csv("output_matrix.csv")
m = as.matrix(data)

countries <- c('Afghanistan', 'Angola', 'Australia', 'Bangladesh', 'Bolivia',
               'Burkina Faso', 'Burundi', 'Canada', 'Chad', 'China',
               'Congo', "Côte_d'Ivoire", 'Ecuador',
               'Egypt', 'El Salvador', 'Eritrea', 'Ethiopia',
               'European Community', 'Ghana', 'Guatemala', 'Guinea', 'Haiti',
               'Honduras', 'India', 'Indonesia', 'Iraq', 'Japan', 'Kenya',
               "North_Korea", 'South_Korea',
               'Lebanon', 'Liberia', 'Madagascar', 'Malawi', 'Mauritania',
               'Mongolia', 'Mozambique', 'Nicaragua', 'Niger', 'Nigeria',
               'Occupied Palestinian Territory', 'Peru', 'Philippines',
               'Rwanda', 'Sierra Leone', 'Somalia', 'Sri Lanka', 'Sudan',
               'Tajikistan', 'Tanzania', 'Uganda', 'USA', 'Zambia', 'Zimbabwe')

dimnames(m) <- list(have = countries,
                    prefer = countries)

# Build the chord diagram:
png(file="chord_plot.png")
p <- chorddiag(m, groupnamePadding = 6, showTicks = F, groupnameFontsize = 8)
p
dev.off()
```