Technical notes: commodity price index computation

Table of contents

This document contains the technical notes for the commodity price index computation. The R code in this document are consistent with 1\_compute\_validate\_index.R.

The index computation requires the following steps, which are explained in details in the next sections.

1. load raw data and process
2. missing data handling
3. compute index from merged dataset from the new source

The validation and plots are recorded in graphics in the other reproducible document.

First you need to load necessary R packages and set the project directory. The directories are recommended to distinguish source code, metadata and data for validation.

library(ggplot2)  
library(openxlsx)  
library(jsonlite)  
library(httr)  
library(readxl)  
library(zoo)  
library(dplyr)  
library(naniar)  
  
project\_path <- '~/Documents/GitHub/un-commodity-prices/deliverables/'  
  
# source the functions needed  
source(paste0(project\_path, 'rscripts/util.R'))  
  
# set paths  
read\_path <- paste0(project\_path, 'data/')  
dir\_metadata <- 'metadata/'  
dir\_datasource\_2024 <- 'datasource\_2024/'  
dir\_val <- 'validation/'

## 0.1 Metadata

First load the **metadata** that contains information for extracting the relevant columns from the international sources.

The content of metadata:

* index\_sort: 4-digits ID used to compute price index
* series\_id: 8 digits ID for specific commodity
* description\_long: detailed description of commodity for 2024 sources
* unit\_2024: unit used as in the description (2024 sources). This might not be the same as in the 2025 sources.
* data\_source\_2024\_code, data\_source\_2024: code and description for data sources used in 2024 version. For example, 5110 is World Bank Commodity markets.
* data\_source\_2025\_code, data\_source\_2025: code and description for data sources used in 2025 version.
* label\_display: commodity name, used for graphics
* label\_source\_2025: commodity name as in their 2025 data sources. It is important to match the correct names to the latest data.
* check\_two\_sources: indicator of whether the sources are switched from 2024 to 2025. If yes, then quality check graphics will be produced.
* keep: indicator of whether we keep the series in the price index computation.
* within\_product\_weight: weight for products that share the same index\_sort. For now only applies to coffee and oil.
* share\_scale: used only for computing products in the subgroups.

metadata <- read.xlsx(paste0(read\_path, dir\_metadata, 'commodity\_metadata.xlsx'),   
 sheet = 'commodity')  
  
head(metadata, 1)

index\_sort series\_id description\_short  
1 1005 020100.01 beef  
 description\_long unit\_2024  
1 Beef, Australia/New Zealand, frozen, CIF US ports ($/kg) usd\_per\_kg  
 data\_source\_2024\_code data\_source\_2024 data\_source\_2025\_code  
1 5110 World Bank - Commodity-markets 5110  
 data\_source\_2025 label\_display label\_source\_2025  
1 World Bank - Commodity-markets Beef Beef.\*\*  
 check\_two\_sources keep within\_product\_weight share\_scale  
1 <NA> yes 1 1

# 1. Data collection and processing

In this step we gather the data from their original sources, then carry out some processing. The end result at this step is a data frame with all the commodity series ready to enter the computation.

## 1.1 Data collection

### 1.1.1 World Bank

We load the World Bank data from an URL (functional as of 2025.4.29)

Do some processing: convert the prices into numeric values.

wb\_link <- 'https://thedocs.worldbank.org/en/doc/18675f1d1639c7a34d463f59263ba0a2-0050012025/related/CMO-Historical-Data-Monthly.xlsx'  
wb\_raw <- read.xlsx(wb\_link,   
 sheet = "Monthly Prices", startRow = 5)  
  
# print the column names  
wb\_var <- get\_info\_wb(wb\_raw)  
  
# process the raw data  
# make the values numeric  
wb <- process\_data\_wb(data = wb\_raw)

Now select the relevant commodities, as defined by **metadata**. We choose the ones where data\_source\_2025\_code == 5110 (world bank), no missing (!is.na(label\_source\_2025) and keep == 'yes' to indicate whether it’s needed.

Please double check whether the variables are what we need!

Some of the labels might have special characters, hence we replace them.

# select relevant series  
# based on metadata  
# this also has to be within the wb scope  
  
wb\_info <- filter(metadata, data\_source\_2025\_code == 5110 & !is.na(label\_source\_2025) & keep == 'yes')  
  
# process the labels to remove the special characters  
wb\_labels <- wb\_info$label\_source\_2025  
wb\_labels <- gsub(',', '.', wb\_labels) # substitute the commas  
wb\_labels <- gsub('\\\*', '.', wb\_labels) # substitute the star (careful since it's wildcard)  
wb\_labels <- gsub('%', '.', wb\_labels) # substitute the commas  
wb\_labels

[1] "Beef..." "Banana..US"   
 [3] "Coffee..Arabica" "Coffee..Robusta"   
 [5] "Tea..Mombasa" "Wheat..US.HRW"   
 [7] "Maize" "Rice..Thai.5."   
 [9] "Soybeans" "Soybean.meal"   
[11] "Cotton..A.Index" "Soybean.oil"   
[13] "Groundnut.oil..." "Palm.oil"   
[15] "Sunflower.oil" "Coconut.oil"   
[17] "Palm.kernel.oil" "Sugar..world"   
[19] "Cocoa" "Fish.meal"   
[21] "Tobacco..US.import.u.v." "Phosphate.rock"   
[23] "Iron.ore..cfr.spot" "Copper"   
[25] "Nickel" "Aluminum"   
[27] "Lead" "Zinc"   
[29] "Tin" "Silver"   
[31] "Coal..Australian" "Crude.oil..Brent"   
[33] "Crude.oil..Dubai" "Natural.gas.index"   
[35] "Rubber..RSS3" "Logs..Cameroon"   
[37] "Sawnwood..Malaysian" "Plywood"   
[39] "Gold" "Platinum"

Above are the variable names that correspond to the world bank data file. Now we carry out the selection: keep year, period, time, datetime and the commodity labels.

Since some labels are long and have special characters, we set new names that are easier for coding.

# select based on names  
wb\_narrow <- select(wb, year, period, time, datetime, all\_of(wb\_labels))  
  
# reset the colnames  
# colnames(wb\_narrow)[5:ncol(wb\_narrow)]  
colnames(wb\_narrow)[5:ncol(wb\_narrow)] <- wb\_info$description\_short  
colnames(wb\_narrow)[5:ncol(wb\_narrow)]

[1] "beef" "banana\_us" "coffee\_arabica" "coffee\_robusta"   
 [5] "tea\_mombasa" "wheat\_us" "maize" "rice"   
 [9] "soybeans" "soybean\_meal" "cotton" "soybean\_oil"   
[13] "groundnut\_oil" "palm\_oil" "sunflower\_oil" "coconut\_oil"   
[17] "palmkernel\_oil" "sugar" "cocoa" "fish\_meal"   
[21] "tobacco" "phosphate\_rock" "iron\_ore" "copper"   
[25] "nickel" "aluminium" "lead" "zinc"   
[29] "tin" "silver" "coal" "crude\_oil\_brent"   
[33] "crude\_oil\_dubai" "naturalgas\_index" "rubber\_rss3" "logs\_cameroon"   
[37] "sawnwood" "plywood" "gold" "platinum"

### 1.1.2 IMF

Carry out similar tasks for the IMF data.

One thing to note is that IMF data API is subject to change in the near future. Please double check if the link is still functional.

imf\_link <- 'https://www.imf.org/-/media/Files/Research/CommodityPrices/Monthly/external-data.ashx'  
  
# imf\_raw <- read\_excel(paste0('YOUR\_PATH', "imf.xls"))  
  
imf\_loc <- tempfile()  
download.file(imf\_link, imf\_loc)  
# read from temporary path  
imf\_raw <- read\_excel(path = imf\_loc)

New names:  
• `POILAPSP` -> `POILAPSP...17`  
• `POILAPSP` -> `POILAPSP...45`  
• `` -> `...89`  
• `` -> `...90`  
• `` -> `...91`  
• `` -> `...92`  
• `` -> `...93`  
• `` -> `...94`  
• `` -> `...95`

# check variables  
imf\_var <- get\_info\_imf(imf\_raw)  
# View(imf\_var)  
  
# process data, conver to numerics  
imf <- process\_data\_imf(imf\_raw)

The last few columns are a bit messy, we manually set the name for **Manganese**. Please double check if this is what you need!

# need to fill in Manganese  
imf <- fill\_imf\_name(data = imf,   
 keyword = 'Mang',   
 col\_to\_fill = '...92',   
 fill\_name = 'PMANG')

Select the variables defined in **metadata**: where data\_source\_2025\_code == 2311.

# select   
imf\_info <- filter(metadata, data\_source\_2025\_code == 2311 &   
 !is.na(label\_source\_2025) & keep == 'yes')  
imf\_info

index\_sort series\_id description\_short  
1 1010 030212.01 fish\_salmon  
2 1011 030613.01 shrimps\_mex  
3 3002 260200.02 manganese\_99  
4 2006 410100.01 hides  
5 2004 510100.03 wool\_fine  
 description\_long  
1 Salmon, fresh, fish-farm bred, export price, Norway ($/kg)  
2 Shrimps, brown, no. 1, shell-on, headless, Mexico ($/kg)  
3 Manganese 99.7% electrolytic manganese flake, free market, in warehouse ($/t)  
4 Cattle hides, US Chicago packer's heavy native steers, FOB shipping point (¢/lb.)  
5 Fine wool, 19 Micron, AWEX auction price, Australia ($/t)  
 unit\_2024 data\_source\_2024\_code data\_source\_2024  
1 usd\_per\_kg 7801 Statistics Norway  
2 usd\_per\_kg 5110 World Bank - Commodity-markets  
3 usd\_per\_tonne 6801 Metal Bulletin Limited  
4 cent\_per\_lb 2311 IMF - Primary Commodity Prices  
5 usd\_per\_tonne 8001 Australian Wool Innovation (AWI)  
 data\_source\_2025\_code data\_source\_2025 label\_display  
1 2311 IMF - Primary Commodity Prices Fish (Salmon)  
2 2311 IMF - Primary Commodity Prices Shrimps (Thailand)  
3 2311 IMF - Primary Commodity Prices Manganese 99.7  
4 2311 IMF - Primary Commodity Prices Hides  
5 2311 IMF - Primary Commodity Prices Wool (fine)  
 label\_source\_2025 check\_two\_sources keep within\_product\_weight share\_scale  
1 PSALM yes yes 1 1  
2 PSHRI <NA> yes 1 1  
3 PMANG yes yes 1 1  
4 PHIDE yes yes 1 1  
5 PWOOLF yes yes 1 1

# select relevant columns  
imf\_narrow <- select(imf, year, datetime, all\_of(imf\_info$label\_source\_2025))  
  
# reset name   
colnames(imf\_narrow)[3:ncol(imf\_narrow)]

[1] "PSALM" "PSHRI" "PMANG" "PHIDE" "PWOOLF"

colnames(imf\_narrow)[3:ncol(imf\_narrow)] <- imf\_info$description\_short

### 1.1.3 FAO

The mechanism is slightly different for FAO. First grab the name (Jute), then query it based on uuid.

# first get metadata  
fpma\_api <- GET("https://fpma.fao.org/giews/v4/price\_module/api/v1/FpmaSerieInternational/")  
fpma\_raw <- fromJSON(rawToChar(fpma\_api$content))  
# str(fpma\_raw)  
fpma\_data <- fpma\_raw$results  
  
# get information for jute  
# do the same for other commodity if needed  
jute\_info <- filter(fpma\_data, grepl('Jute', commodity\_name))  
jute\_info

uuid iso3\_country\_code country\_name  
1 5a272e65-e437-41c2-bcb0-f229dc14f47b IPS INTERNATIONAL PRICES  
 periodicity market market\_name market\_info market\_type  
1 monthly, 2024-04-01, 2004-01-01 1750 Bangladesh Export  
 admin\_unit admin\_unit2 commodity  
1 2711  
 commodity\_name  
1 Jute BWD (f.o.b. Mongla, at sight)/from 2006 Jute BTD (f.o.b Bangladesh Port)  
 commodity\_info commodity\_image commodity\_code commodity\_start\_date  
1 NA CMM530300 1  
 alternative\_code alternative\_name source  
1 323  
 source\_name  
1 Bangladesh Jute Mills Corporation/The Public Ledger/Wilhelm G. Clasen (WGC)  
 source\_url price\_type\_id price\_type currency measure\_unit measure\_unit\_label  
1 11 EXPORT USD 3555 tonne  
 conversion\_factor  
1 0.001

We use the uuid to get the jute data. Double check the period where the data is available (it is only from 2004.1 to 2024.1).

jute\_raw <- GET(paste0("https://fpma.fao.org/giews/v4/price\_module/api/v1/FpmaSeriePrice/",jute\_info$uuid,"/"))  
jute <- fromJSON(rawToChar(jute\_raw$content))  
  
# only from 2004.1 to 2024.1  
head(jute$datapoints, 3)

id price\_value price\_value\_real price\_value\_dollar conversion\_factor  
1 5070349 840 NA 840 0.001  
2 5070348 780 NA 780 0.001  
3 5070347 820 NA 820 0.001  
 date periodicity  
1 2024-04-01 monthly  
2 2024-03-01 monthly  
3 2024-02-01 monthly

tail(jute$datapoints, 3)

id price\_value price\_value\_real price\_value\_dollar conversion\_factor  
242 612508 245 NA 245 0.001  
243 612507 245 NA 245 0.001  
244 612506 230 NA 230 0.001  
 date periodicity  
242 2004-03-01 monthly  
243 2004-02-01 monthly  
244 2004-01-01 monthly

Convert to proper format to prepare for merging.

jute <- data.frame(jute$datapoints[, c('date', 'price\_value\_dollar')])  
colnames(jute)[1] <- 'datetime'  
colnames(jute)[2] <- 'jute'  
jute$datetime <- as.Date(jute$datetime)

## 1.2 Merge

We carry out a **left\_join** on the three data sets. The keywords that we join by are the **date times**.

dcommodity <- left\_join(wb\_narrow, imf\_narrow) |>   
 left\_join(jute)

Joining with `by = join\_by(year, datetime)`  
Joining with `by = join\_by(datetime)`

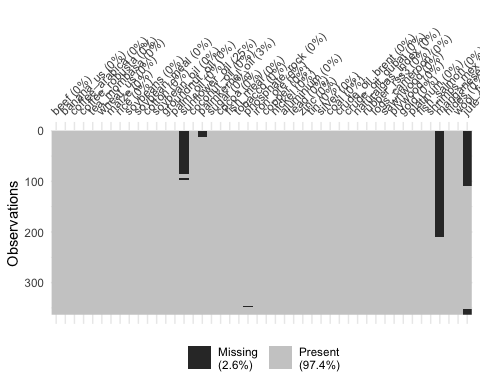
colnames(dcommodity)

[1] "year" "period" "time" "datetime"   
 [5] "beef" "banana\_us" "coffee\_arabica" "coffee\_robusta"   
 [9] "tea\_mombasa" "wheat\_us" "maize" "rice"   
[13] "soybeans" "soybean\_meal" "cotton" "soybean\_oil"   
[17] "groundnut\_oil" "palm\_oil" "sunflower\_oil" "coconut\_oil"   
[21] "palmkernel\_oil" "sugar" "cocoa" "fish\_meal"   
[25] "tobacco" "phosphate\_rock" "iron\_ore" "copper"   
[29] "nickel" "aluminium" "lead" "zinc"   
[33] "tin" "silver" "coal" "crude\_oil\_brent"   
[37] "crude\_oil\_dubai" "naturalgas\_index" "rubber\_rss3" "logs\_cameroon"   
[41] "sawnwood" "plywood" "gold" "platinum"   
[45] "fish\_salmon" "shrimps\_mex" "manganese\_99" "hides"   
[49] "wool\_fine" "jute"

# 2. Quality checks

It can be very convenient to visualize the data missingness.

# visualise which series are missing  
commodity\_only <- select(dcommodity, -c(year, period, time, datetime))  
vis\_miss(commodity\_only)



We can see that sunflower\_oil, palmkernel\_oil, manganese\_99 and jute have quite some missing values. We should be careful about what to do for each case.

* historical missing can be handled by using already existing data from other sources. If multiple data sources have similar values for most of the time periods, they can be used directly (sunflower oil, palm kernel oil, manganese).
* discontinued data: depending on the importance of the product in the index computation, it can be completely dropped (jute).

The missingness of prices should be monitored constantly as data sources might change.

## 2.1 Fill missing period

We create a copy of the data for backfilling, in this way we keep track of the original data for comparison.

dcommodity\_filled <- dcommodity

### 2.1.1 Manganese

For manganese, we use the historical data from **Metal Bulletin** (6801). This is already processed and saved in prices\_2024\_compare.rds, so we load the data and retrieve the proper column.

When it comes to backfilling, we use the function rows\_patch from dplyr. This function conveniently matches the common column (date time) and fills the missing.

We save it in the dcommodity\_filled dataframe.

# missing period in the wb data  
manga\_mp <- check\_missing\_period(data = dcommodity, tag = 'manganese\_99')  
manga\_missing\_period <- manga\_mp$missing\_datetime  
  
# load comparison data  
dcompare <- readRDS(paste0(read\_path, dir\_val, 'prices\_2024\_compare.rds'))  
  
# manganese, use the one for 260200.02  
manga\_compare <- filter(  
 dcompare, CommodityProduct == '260200.02' & dtime %in% manga\_missing\_period  
)|> select(manganese\_99 = Value,   
 datetime = dtime)  
  
  
# select the price in the original data  
manga\_dcommodity <- select(dcommodity\_filled,   
 manganese\_99, datetime)  
  
# patch the same period in the comparison data  
manga\_filled <- rows\_patch(manga\_dcommodity, manga\_compare, by = 'datetime')  
# plot(manga\_filled$manganese\_99)  
# replace the filled series in dcommodity  
dcommodity\_filled$manganese\_99 <- manga\_filled$manganese\_99

### 2.1.2 Sunflower oil

For sunflower oil, we use a comparable historical data from IMF.

sunflower\_dcommodity <- select(dcommodity\_filled,   
 sunflower\_oil, datetime)  
  
sunflower\_mp <- check\_missing\_period(data = dcommodity, tag = 'sunflower\_oil')  
sunflower\_missing\_period <- sunflower\_mp$missing\_datetime  
  
# select sunflower oil in the imf data  
sunflower\_imf <- select(imf, sunflower\_oil = PSUNO, datetime) |>   
 filter(datetime %in% sunflower\_missing\_period)  
  
# patch the same period in the comparison data  
  
sunflower\_filled <- rows\_patch(sunflower\_dcommodity, sunflower\_imf, by = 'datetime')  
# plot(sunflower\_filled$sunflower\_oil)  
# replace the filled series in dcommodity  
dcommodity\_filled$sunflower\_oil <- sunflower\_filled$sunflower\_oil

### 2.1.3 Palm kernel oil

For this product, we do not have any product that is directly replaceable. We can use palm oil from IMF, which is close enough.

palmkernel\_dcommodity <- select(dcommodity\_filled,   
 palmkernel\_oil, datetime)  
  
palmkernel\_mp <- check\_missing\_period(data = dcommodity, tag = 'palmkernel\_oil')  
palmkernel\_missing\_period <- palmkernel\_mp$missing\_datetime  
  
  
# select palm oil in the imf data  
palm\_imf <- select(imf, palmkernel\_oil = PPOIL, datetime) |>   
 filter(datetime %in% palmkernel\_missing\_period)  
  
# patch the same period in the comparison data  
palmkernel\_filled <- rows\_patch(palmkernel\_dcommodity, palm\_imf, by = 'datetime')  
# replace the filled series in dcommodity  
dcommodity\_filled$palmkernel\_oil <- palmkernel\_filled$palmkernel\_oil

# 3. Price index computation

This is the step for computing the prices using the weights. One important thing to note is that the **weight matrix does not have the 8 digits code** for individual series, so we need to have one additional step of linking these two tables: **metadata** and **weights**.

## 3.1 Prepare weights

### 3.1.1 Weights

weights <- read.xlsx(paste0(read\_path, dir\_metadata, 'weights.xlsx'))  
head(weights)

index\_sort group subgroup index\_description w s  
1 1001 ALL FOOD FOOD Wheat 2206573845 0.001966377  
2 1002 ALL FOOD FOOD Maize 10035445721 0.008943037  
3 1003 ALL FOOD FOOD Rice 12086634062 0.010770943  
4 1004 ALL FOOD FOOD Sugar 18309075031 0.016316040  
5 1005 ALL FOOD FOOD Bovine meat 10935950196 0.009745517  
6 1006 ALL FOOD FOOD Bananas 7314470514 0.006518253  
 available  
1 yes  
2 yes  
3 yes  
4 yes  
5 yes  
6 yes

The content of weight data:

* index\_sort: 4-digits ID used to compute price index. Link to **metadata**.
* group: first level of grouping of commodity
* subgroup: second level of grouping
* index\_description: description of the index. This does not necessarily link to metadata; however it is indicative to the product used.
* w and s: numeric values used to compute the weighted sum of the index. s sum up to 1, while w for each product divided by the total sum of w equals to s (hence equivalent).
* available: indicator of whether this product is available in the data source in 2025.

# shares sum up to 1  
weights$s |> sum()

[1] 1

### 3.1.2 Combine weight with metadata

This step produces a table that provides information that links price series to their weights.

m <- dplyr::filter(metadata, keep == 'yes') |>   
 select(index\_sort,   
 series\_id,   
 description\_short,   
 within\_product\_weight,  
 share\_scale)  
  
m |> head()

index\_sort series\_id description\_short within\_product\_weight share\_scale  
1 1005 020100.01 beef 1.0 1  
2 1010 030212.01 fish\_salmon 1.0 1  
3 1011 030613.01 shrimps\_mex 1.0 1  
4 1006 080300.01 banana\_us 1.0 1  
5 1101 090100.03 coffee\_arabica 0.4 1  
6 1101 090100.05 coffee\_robusta 0.6 0

ws <- dplyr::filter(weights, available == 'yes') |>   
 select(index\_sort,   
 group,   
 subgroup,  
 index\_description,  
 s)  
  
ws |> head()

index\_sort group subgroup index\_description s  
1 1001 ALL FOOD FOOD Wheat 0.001966377  
2 1002 ALL FOOD FOOD Maize 0.008943037  
3 1003 ALL FOOD FOOD Rice 0.010770943  
4 1004 ALL FOOD FOOD Sugar 0.016316040  
5 1005 ALL FOOD FOOD Bovine meat 0.009745517  
6 1006 ALL FOOD FOOD Bananas 0.006518253

### 3.1.3 Within product weights

Double check how many rows there are. They might be different, but it is not a big problem: we have a mechanism to cope with products that share the same **index sort code**, as explained below.

c(nrow(m), nrow(ws))

[1] 46 43

The within-product weight is defined in the **metadata**. For most products this value is 1, exceptions apply to two products:

* index sort code 1101: **Coffee** Arabica (090100.03) and Robusta (090100.05) takes up 40% and 60%
* index sort code 4201: **Crude oil** - Brent (270900.01) and Dubai (270900.02) takes up 50% each

Special case:

* index sort code 2010: **Rubber** RSS3 (400100.02) and TSR20 (400100.01). We use RSS3 as it is complete, while TSR20 is only available from 1999 in World Bank Data.

Now join the selected columns of weight and metadata together by index\_sort.

# only 1101 (coffee), 4201 (crude oil) have more frequency  
table(m$index\_sort) |> sort()

1001 1002 1003 1004 1005 1006 1008 1010 1011 1102 1103 1201 1202 1203 1204 1206   
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   
1207 1208 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 3001 3002 3003 3004   
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   
3005 3006 3007 3008 3009 3101 3102 3103 4001 4101 1101 4201   
 1 1 1 1 1 1 1 1 1 1 2 2

M <- full\_join(m, ws, by = 'index\_sort')  
head(M,3)

index\_sort series\_id description\_short within\_product\_weight share\_scale  
1 1005 020100.01 beef 1 1  
2 1010 030212.01 fish\_salmon 1 1  
3 1011 030613.01 shrimps\_mex 1 1  
 group subgroup index\_description s  
1 ALL FOOD FOOD Bovine meat 0.009745517  
2 ALL FOOD FOOD Fish 0.002242046  
3 ALL FOOD FOOD Crustaceans 0.010791602

## 3.2 Compute index

Now we compute the index. First we need to base the prices at 2015 as 100: compute the average price for each product for 2015, and merge it back to **M** matrix. These 2015 basis values are used as the denominator when we compute the index.

# call it a different name  
dprices <- dcommodity\_filled  
  
basis\_2015 <- filter(dprices, year == 2015) |>   
 select(-c(year, period, time, datetime)) |>   
 apply(MARGIN = 2, mean)   
basis\_2015 <- data.frame(basis\_2015)  
basis\_2015$description\_short <- rownames(basis\_2015)  
head(basis\_2015)

basis\_2015 description\_short  
beef 4.5591542 beef  
banana\_us 0.9569142 banana\_us  
coffee\_arabica 3.5260692 coffee\_arabica  
coffee\_robusta 1.9411679 coffee\_robusta  
tea\_mombasa 2.9646806 tea\_mombasa  
wheat\_us 204.4491286 wheat\_us

# merge it to M  
M2 <- left\_join(M, basis\_2015, by = 'description\_short')

Select the relevant columns from the prices (by dropping year, period, time, datetime).

# select only relevant columns  
dcwide <- select(dprices, -c(year, period, time, datetime))  
  
# still want to keep track of the time information  
rownames(dcwide) <- dprices$datetime

Divide the original values by the 2015 basis, then multiply by their weights. For the special cases (coffee, crude oil), the *within\_product\_weight* is used to combine the weighted sum of the two sub-products.

### 3.2.1 Index for one product group

UNCTAD has many distinct subgroups for products, hence we need to match the variables for each group.

# grouping information is in the validation\_unctad tab  
  
grouping <- read.xlsx(paste0(read\_path, dir\_metadata, 'weights.xlsx'),   
 sheet = 'validation\_unctad')  
  
  
grouping

unctad\_name  
1 All.groups\_Index\_Base\_2015\_Value  
2 All.food\_Index\_Base\_2015\_Value  
3 Food\_Index\_Base\_2015\_Value  
4 Tropical.beverages\_Index\_Base\_2015\_Value  
5 Vegetable.oilseeds.and.oils\_Index\_Base\_2015\_Value  
6 Agricultural.raw.materials\_Index\_Base\_2015\_Value  
7 Minerals..ores.and.metals\_Index\_Base\_2015\_Value  
8 Minerals..ores.and.non.precious.metals\_Index\_Base\_2015\_Value  
9 Precious.metals\_Index\_Base\_2015\_Value  
10 Fuels\_Index\_Base\_2015\_Value  
11 Tropical.beverages.and.food\_Index\_Base\_2015\_Value  
12 All.groups.excl..fuels\_Index\_Base\_2015\_Value  
13 All.groups.excl..precious.metals\_Index\_Base\_2015\_Value  
14 All.groups.excl..precious.metals.and.fuels\_Index\_Base\_2015\_Value  
 group\_name select\_group\_from  
1 all level\_1  
2 all\_food level\_1  
3 food level\_2  
4 tropical\_beverages level\_2  
5 vegetable\_oilseeds\_oil level\_2  
6 agricultural\_raw\_material level\_1  
7 minerals\_ore\_metal level\_1  
8 minerals\_ore\_metal\_non\_precious\_metal level\_2  
9 precious\_metal level\_2  
10 fuels level\_1  
11 tropical\_beverages\_food level\_2  
12 all\_excl\_fuels level\_1  
13 all\_excl\_precious\_metal level\_2  
14 all\_excl\_precious\_metal\_fuels level\_2

Let us test it for the first combination: **all products**.

# select one combination  
# from 1 to 14  
cg <- query\_commodity\_group(target\_group\_info = grouping[1,])  
cg

$target\_group\_info  
 unctad\_name group\_name select\_group\_from  
1 All.groups\_Index\_Base\_2015\_Value all level\_1  
  
$commodity\_groups  
[1] "ALL FOOD" "AGRICULTURAL RAW MATERIALS"   
[3] "ALL MINERALS, ORES AND METALS" "FUELS"   
  
$weight\_ref\_column  
[1] "group"

The function compile\_index takes in three arguments:

* d\_price: the prices for individual products
* d\_weight\_unscaled: the weight matrix (after combining metadata)
* commodity\_group: group information used to select and rescale the index

index\_onegroup <- compile\_index(d\_price = dcwide,  
 d\_weight\_unscaled = M2,  
 commodity\_group = cg)

compute index for group: all   
processing product 1 : beef   
processing product 2 : fish\_salmon   
processing product 3 : shrimps\_mex   
processing product 4 : banana\_us   
processing product 5 : coffee\_arabica   
processing product 6 : tea\_mombasa   
processing product 7 : wheat\_us   
processing product 8 : maize   
processing product 9 : rice   
processing product 10 : soybeans   
processing product 11 : soybean\_meal   
processing product 12 : cotton   
processing product 13 : soybean\_oil   
processing product 14 : groundnut\_oil   
processing product 15 : palm\_oil   
processing product 16 : sunflower\_oil   
processing product 17 : coconut\_oil   
processing product 18 : palmkernel\_oil   
processing product 19 : sugar   
processing product 20 : cocoa   
processing product 21 : fish\_meal   
processing product 22 : tobacco   
processing product 23 : phosphate\_rock   
processing product 24 : iron\_ore   
processing product 25 : manganese\_99   
processing product 26 : copper   
processing product 27 : nickel   
processing product 28 : aluminium   
processing product 29 : lead   
processing product 30 : zinc   
processing product 31 : tin   
processing product 32 : silver   
processing product 33 : coal   
processing product 34 : crude\_oil\_brent   
processing product 35 : naturalgas\_index   
processing product 36 : rubber\_rss3   
processing product 37 : hides   
processing product 38 : logs\_cameroon   
processing product 39 : sawnwood   
processing product 40 : plywood   
processing product 41 : wool\_fine   
processing product 42 : gold   
processing product 43 : platinum   
processing product 44 : coffee\_robusta   
processing product 45 : crude\_oil\_dubai

head(index\_onegroup$index)

1995-01-01 1995-02-01 1995-03-01 1995-04-01 1995-05-01 1995-06-01   
 45.86073 46.50581 46.63123 47.93610 47.44628 46.59248

### 3.2.2 Index for other groups

Please read through the R code in 1\_compute\_valdiate\_index.R section 4.