**Run system**

IP for web scraping instant: 192.168.2.57:5901

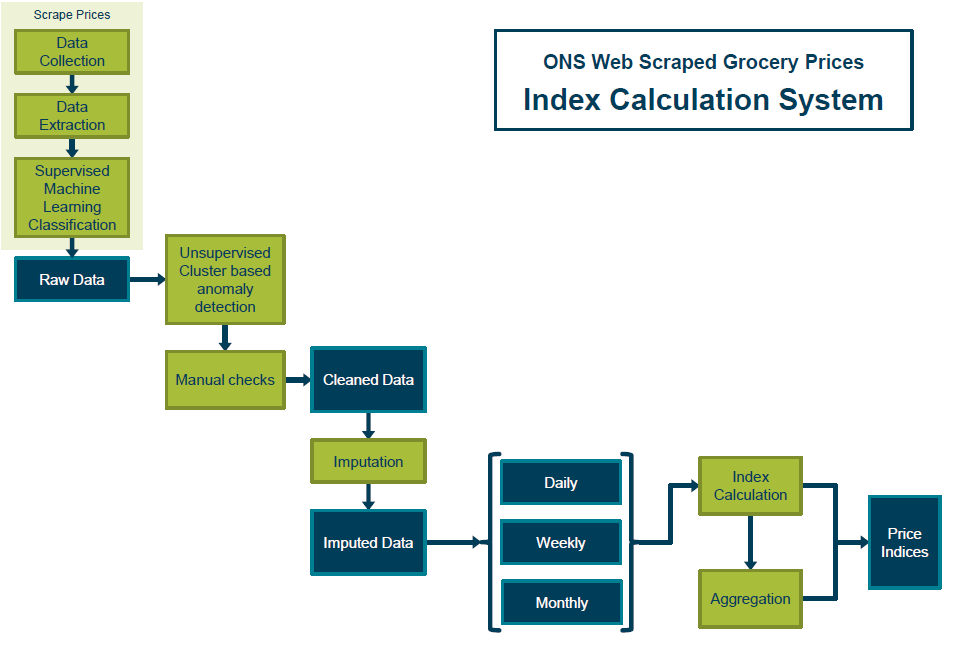
Location of run system cdhimrtode: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system

Hello and welcome to the guidelines for the new WebScraperSystem which functions directly from the NS innovation labs in Newport. This system is designed to make producing indices far quicker and easier with one single centralised collection of code.

The run system is designed to take in the web scraped grocery data which has been through the SVM classification (has a ML\_predict variables), clean it, impute for up to 7 days, store it in a Mongo database, create price indices and put them into a data visualization.

The code is mainly written in python, with a few bits of bash script.

The system is best run through basic python script. To work the code I would suggest using Jupyter notebooks (go to the folder where you want to open the code, open a terminal and type “Juptyer notebook”). I would also suggest making a copy of the code into the “working” folder before working on it and testing it before replacing the original file.



**Storage/backups**

The collections of code (and required data “weights.csv” and “original\_clusters.csv”) which make up this system are stored in several places (just to be on the safe side!!).

* It can be found in the gdrive of the [onswebscraping@gmail.com](mailto:onswebscraping@gmail.com) account (with password: ScrapeMyWeb ) in the folder “WebScrapingSystem” & in the gmail drive that Matt linked me to under the same folder name
* It is stored in the folder “WebScrapingSystem” in the BitBucket account under the account using the above email address/password (I recommend this as the place you source the code from should you want to make edits as the changes are far easier for you and your colleagues to track). There is a document I’ve created @ONSWebscraping Phase 2 Bitbucket Instructions” (attached to this email) which walks you through how to set this up. Please circulate this document as much as possible because I think it will save you all dozens of hours moving forward
* It is stored in two folders in the ONS system
* It can also be found on the same instant in the labs where the web scraping code is. This is the IP address:

**NB The most recent version of the cleaned, imputed daily product level data can also be found in these locations (with the exception of BitBucket as it has quite stingy data allowances). It is called “\_main\_20160918.csv” and runs all the way up until the 18th September 2016.**

**MongoDB**

The data is saved in the Mongo data base at two time points, after scraping and after cleaning. The database is kept in a different instant (called Big Data Mongo database) and therefore if this instant is turned of then you will not be able to access the database.

Before we start running through the code I’ll just give you some tips on how to use MongoDB through the terminal. This will be very useful as it will let you have a look at the data which is collected from the web (before the cleaning) and afterwards as well. You can access the mongoDB “shard” which our data is stored by typing mongo -—host 192.168.2.20 the terminal. Now you are inside mongdb and can access the various databases which exist within it. The following commands are useful:

show dbs - displays all of the databases within the mongoDB shard ( the raw prices are stored in [prices\_db] and the cleaned prices in [cleaned\_imputed\_daily\_data]

use prices\_db - this will put you into the [prices\_db] database

show collections - this will show you the collections within [prices\_db.] You can think of a collection as a database within a database.

db = db.combined - this assigns you to the collection entitled [combined] within the [prices\_db] database which is where all of the web scraped data is stored every day straight from the scrapers

db.count() - will count all of the individual prices within the database, at the moment the number of prices within [prices\_db][combined] is around 6 million

db.findOne() - will show you one price. There are far more commands which will perform similar queries on the mongoDB website

**Running the System**

The best way for you to understand how the system is to refer to the main python modules which initiates the whole system. These are called WS2system\_launcher.py and WS2system\_launcher2.py. The first cleans the data and puts it into the mongo databases, the second creates price indices and puts them into a data visualisation friendly format. There are 9 steps (and the modules it calls to perform them) which are run my these files.

To run these file you would open the terminal in its location (/media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/WebScrapingSystem) and simply write the command python WS2system\_launcher.py, when this has finished write: WS2system\_launcher2.py.

In practice though you do not want to have to run these files manually every day. Instead crontab is used to automate the process.

To change/see the crontab open a terminal from the run\_system folder. Open crontab (type crontab -e to edit). The current automated run files will be present. These are the commands that run the web scrapers so please do not turns these off/change them unless you are sure of the results. The run command looks like this:

37 10 \* \* \* python /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/WebScrapingSystem/WS2system\_launcher.py

If this is not working the path may be wrong.

The first part of this command 37 10 \* \* \* means: min hour day month year. Anything set to \* will run every one of these time periods. So 37 10 \* \* \* will run every day at 10.37am.

**System stages**

The files ‘WS2system\_launcher.py” and “WS2system\_launcher2.py” initiates functions from other python (and R) modules stored in the functions folder. These 9 steps I’m about to explain match well with the descriptions on the code itself.

All python files can be found here: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions

R files here: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/WebScrapingSystem

**WS2system\_launcher.py steps 1 – 5:**

**Stage (1): Pull data from Mongdo database**

Input data: Mongo database

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions

This calls in the function pull\_it() from the first\_mongo\_pull\_v3.py code. This pulls down the previous 7 days worth of data from the mongoDB collections [prices\_db][combined] collection and exports it to a .csv file called last\_7\_days

Output file: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/last\_7\_days.csv

**Stage (2): Anomaly detection (Matt's code)**

Input data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/last\_7\_days.csv

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions

Calls in the function clean\_split() from Matts cleaning code called anomaly\_detection\_v3.py.

This removes prices which don’t match up well to the fixed clusters formed in the base period. The code calls in the clusters from: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/WebScrapingSystem/original\_clusters.csv and then trims the dates from the original clusters back off again, then splitting the data into item level datasets called cleaned\_item.csv.

Output files: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/cleaned/clean\_chunk...

**Stage(3): Run R identify missing days code**

Input data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/cleaned/clean\_chunk...

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/WebScrapingSystem

An R file is used for this step (ask Matt Mayhew for details). This file is stored in the /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/WebScrapingSystem folder as it needs to be stored in the same folder from which it is initiated. This R code (‘identify\_missing\_v3.R”) calls in all of the cleaned\_item.csv datasets and identifies missing days for each item in preparation for imputing, exporting item level datasets called missing\_item.csv.

Output files: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/cleaned/missing\_aug\_todays\_date.csv

**Stage(4): Impute and combine the data into one main file**

Input data location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/cleaned

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions

The python module “pack\_stack\_rack.py” imputes the missing\_item.csv datasets, then exports them into imputed\_item.csv datasets. It then merges all of these imputed datasets together and keeps only the current date of data in ‘\_imputed\_all.csv’. This is why we import the 7 previous days of data from the mongoDB so as the products from that day can be imputed using rules consistent with the rest of the dataset.

Output data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/\_imputed\_all.csv

**Stage (5): Add todays data to the main file and the mongoDB (clean dataset)**

Input data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/\_imputed\_all.csv

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions

This stage has caused a lot of problems and if indices are not being created this is the first place I would look.

The first step is to combine to todays today with the main (the back series dataset in: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions\_full\_data\_run/data) and then save over the \_main\_.csv file. This file will be later used to create price indices's. The follow command funs the python script where this happens:

os.system('python /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions/add\_to\_databases.py')

The second step of this stage is to stack the current days data on top of the mongoDB collection where all of the other cleaned/imputed data sits. Command to run bash code:

os.system('/media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions/load\_csv\_single.sh')

This step also gave a lot of problems and can be checked by running directly from the terminal. When you read this code in through the terminal it will feed the .csv file targeted in the code into the [cleaned\_imputed\_daily\_data][test] collection. It will stack the data “on top” of any previous data which lies in the collection. If you run it twice, it will leave 2 copies of that day in the database etc. If you adjust the [collection] or [test] names on this code it will automatically create a new one which is a useful feature. As it is easy to drop duplicates of data do not worry about re-adding data to the database. We are better of having copies of data than no data.

Output data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions\_full\_data\_run/data

Mongo database

**WS2system\_launcher2.py steps 6 – 9:**

**Stage (6) : Averaging data over time period**

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions/averaging\_v3.py

Input data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions\_full\_data\_run/data/\_main\_.csv

This code “averaging\_v3” creates averaged daily, weekly and monthly datasets for each product in preparation for creating indices. The main dataset updated in step 5 is used to create the averages. The averages are taken across all of the products which are the same within each time period using a arithmetic mean.

Output data location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/averages

**Stage (7): Price index creation**

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions

Input data location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/averages

This produces GEKS, UnitPrice, DailyChained (timestamped\*\* for that day) and CLIP indices with each indices having its own module of python code which is called in. Any additional indices code (eg LCPI, FEWS) which is written should be incorporated into the system at this stage.

The Unit price and CLIP indices are double chain linked at the aggregate level, single chain linked at the item level. The Geks is single chained linked.

Double chain linking is a very simple concept which is impossible to get your head around and code. This is why there are multiple files with very similar code in.

Price indices are created with todays date in the name of the file.

The price index code for each index are as follows and are annotated within the code:

GEKS: geks.py

Aggregation (single chain link): aggregation\_v3.py

Unit Price: unitprice.py, unitprice-weekly.py

Aggregation (double chain link): UNIT\_double\_chain\_link\_itemlevel.py, UNIT\_double\_chain\_link\_itemlevel\_week.py, UNIT\_double\_chain\_link\_weekly.py, UNIT\_double\_chain\_link.py,

Dailychained: dailychained.py

CLIP: CLIP.py, CLIP\_weekly.py

Aggregation (double chain link): CLIP\_double\_chain\_link.py, CLIP\_double\_chain\_link\_itemlevel.py, CLIP\_double\_chain\_link\_itemlevel\_week.py, CLIP\_double\_chain\_link\_week.py

Output data location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/price\_indices

**Stage (8): Data visualisation**

Input data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/price\_indices

Stage 8 puts the create price indices into a structured json format.

### Data structure

```

.

+-- Monthly

+-- Level 0

+-- Overwiew

+-- Level 1

+-- Food

+-- Alcoholic

+-- Weekly

+-- Level 0

+-- Overwiew

+-- Level 1

+-- Food

+-- Alcoholic

```

This is the input format for the Prices data visualisation that can be found here: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/PricesTool/index\_v1.html. There is also a readme file within this location.

The code, and updates to the code can be found here:

# https://github.com/ONSPrices/PricesTool

# password: importio2015

#here is the tool https://onsprices.github.io/PricesTool/index.html

Output data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/price\_indices/test.json

/media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/PricesTool/data/test.json

/media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/PricesTool/index\_v1.html

**Stage (9): Automated email**

Input data: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/price\_indices

Code location: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/Data/price\_indices/emailer\_v3.py

Stage 9 sends out a daily email containing the price indices upto, and including todays. To edit the mailing list you can add/remove email addresses from the ‘emailto’ list inside the function emailer(). You can also add price indices to the email by putting the data into the PriceIndex folder, and adding the name to csvfiles (does not have to be a csv).

The current daily email contains the following files: 'month\_'+ todays\_date + '\_GEKS\_item.csv','week\_'+ todays\_date + '\_GEKS\_item.csv',

'month\_'+ todays\_date + '\_GEKS\_agg.csv', 'week\_'+ todays\_date + '\_GEKS\_agg.csv',

'DC\_daily\_itemlevel\_'+ todays\_date +'\_.csv', 'DC\_weekly\_itemlevel\_'+ todays\_date +'\_.csv',

'DC\_monthly\_itemlevel\_'+ todays\_date +'\_.csv', 'DC\_daily\_agg\_'+todays\_date+'\_.csv','DC\_weekly\_agg\_'+ todays\_date +'\_.csv',

'DC\_monthly\_agg\_'+ todays\_date +'\_.csv', 'unitdoublechained\_'+ todays\_date +'\_.csv',

'unitdoublechaineditemlevel\_'+ todays\_date +'\_.csv', 'unitdoublechaineditemlevelweek\_'+ todays\_date +'\_.csv',

'unitdoublechainedweek\_'+ todays\_date +'\_.csv','CLIPdoublechained'+todays\_date'+.csv', 'CLIPdoublechaineditemlevel'+todays\_date+'.csv',

'CLIPdoublechainedweek'+todays\_date+'.csv','CLIPdoublechaineditemlevelweek'+todays\_date+'.csv','test.json'

There does exist code on the onsbigdata bitbucket account which will automatically send the data up to a Gdrive (see (3) in Next Steps)…..potentially a better option moving forward so as not to clog up mailboxes.

\*\*The only datasets which are marked with the current date which they are produced are the indices datasets……all of the intermediate datasets (cleaned, imputed, missing) are overwritten each day. This can be adjusted if you would rather keep all the data but you would eventually run into memory problems.

**Stage 0: Pull down full dataset from Mongdb and clean**

If the system is down for a few days/something goes wrong with the data a new backseries will be needed to be pulled down from the Mongo database. The files to do this can be found here: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions\_full\_data\_run.

The steps follow the same structure as steps 1-5 explained above.

second\_mongo\_pull\_v3.py will pull down the data (this can take a while)

anomaly\_detection\_v3.py will check for anomalies

impute\_pack\_stack\_rack\_v3.py will impute and stack (this can take quite a while)

The new dataset should be put here: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions\_full\_data\_run/data and called \_main\_.csv.

**Next Steps**

There are a number of improvements that could be made to this system. Please take a copy of the system from the gdrive or the bitbucket account to make and test these changes separately to the production system.

(1) Optomise the price index so that is only calculates the latest time periods price index instead of the current approach of re-calculating the whole index each time.

(2) At the moment the code sends out the indices as an attachment to an email to “[onswebscraping@gmail.com](mailto:onswebscraping@gmail.com)” (plus any additional email addresses you choose to add). While this provides a quick and easy way to view the indices it will quickly fill up the memory of your email account. One way around this is to incorporate the existing code which exports the data to the google accounts Gdrive. The code is in the folder “gdrive\_upload” in the main folder. Alternatively this code can be cloned from the BitBucket account from the folder “web-scrapers-for-bitbucket-liz”

3) Update the weights for 2017 after February.

4) Update the double chaining to the new methodology which following the CPI

5) Update to run at COICOP5 level rather than at item level

6) Use LCF or mysupermarket data as train data

7) Update data\_vis.py file (in function folder) to show all of the item level price indices. The otherput is in:/media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/PricesTool. The index.html code

8) Pull down full dataset, add newest to clean data (most recent cut), and put this is the folder: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions\_full\_data\_run/data. Call is \_main­\_.csv. The code to do this is in the folder: /media/mint/e834712c-23da-4cbe-a4ec-3a35d416877b/Run\_system/functions\_full\_data\_run. However, it doesn't quite work as yet, just some little bits and pieces though so very do able.