**Lockdown 3.0 and Unlock 1.0 Restrictions**

After a second extension of the nationwide lockdown, the central government designed the third phase of lockdown to adapt to local rates of viral spread. The center created the following district-wise lockdown parameters:

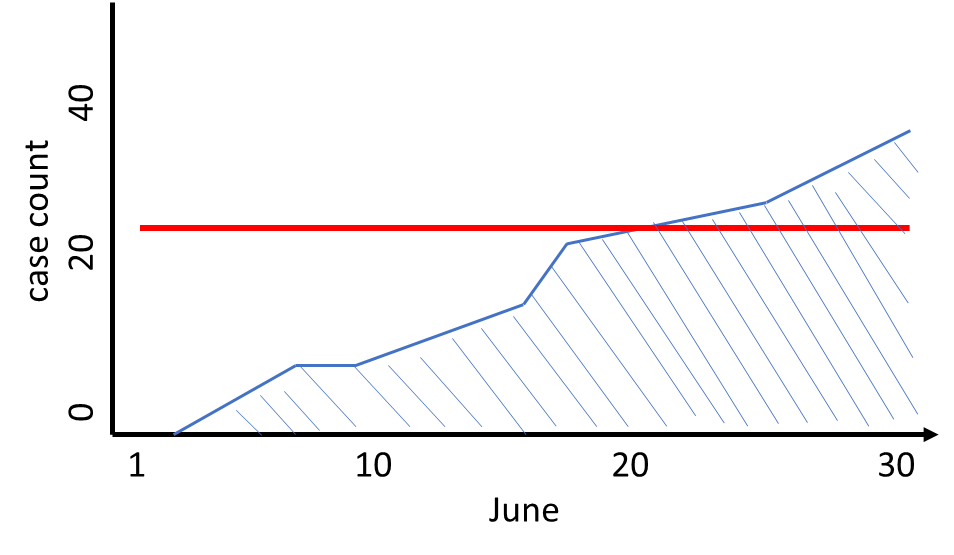
* Green Zones (319 districts):
  + No cases in the past 21 days.
  + All activities except those prohibited nationwide will be allowed. Buses can operate at 50 per cent capacity. [Full List](https://www.indiatoday.in/coronavirus-outbreak/story/lockdown-extended-what-is-allowed-in-green-zones-1673426-2020-05-01)
* Orange Zones (284 districts):
  + Four wheelers, including taxis, with one driver and two passengers will be allowed. Buses will not be allowed. Liquor stores will be allowed to open, and e-commerce will be allowed for both essential and non-essential items. [Full List](https://www.indiatoday.in/india/story/coronavirus-what-s-allowed-in-orange-zones-in-lockdown-3-0-1673449-2020-05-01)
* Red Zones (130 districts):
  + Identified by having a high caseload, contributing to 80 per cent of Covid-19 cases of each state or a district with a doubling rate of less than four days.
  + Cycle and auto rickshaws, taxis and public transport, barbers and salons will remain shut. Four wheelers with a driver and two passengers and two wheelers without pillion riders will be allowed. Offices can open with a third of the staff. E-commerce will be allowed for essential services and all standalone shops, including liquor stores, will be allowed to be open. [Full List](https://www.indiatoday.in/india/story/coronavirus-what-s-allowed-in-red-zone-in-lockdown-3-0-1673434-2020-05-01)
* Containment Areas: A locally defined space where a cluster of cases has been identified.
  + Entry and exit of people will be severely restricted. Movement of people will only be allowed for supply of essential goods and services. Almost everything will remain shut.

These restrictions (largely an improvement from the blanket lockdown of the previous 5 weeks) were in effect from 4 May until 1 June. During this period, 15 state or district governments reclassified district zones to be stricter based on the spread of the virus. After 17 May, state and local authorities had more input in defining district classification and restrictions but there was little change apart from the 15 districts mentioned.

Unlock 1.0 began 1 June and focused on containment and micro-containment zones. Each state set their own policies regarding how they defined a containment area and to what degree the inhabitants (infected or not) could move about freely. States generally followed the Ministry of Health and Family Welfare’s [guidelines](https://www.mohfw.gov.in/pdf/UpdatedContainmentPlanforLargeOutbreaksofCOVID19Version3.0.pdf).

While analysis of lockdown restrictions for the month of May is based on district-wise zones, analysis of June and July restrictions is built upon state-wise policy for the number of days a containment zone is to remain in place from the last positive test and the quarantine period for individuals coming into the state. The state-wise policy analysis is intended to measure the degree of unnecessary restrictions on the populous. The required days of containment or quarantine range from 0-28 and 0-21, respectively.

To convert the containment\_days variable from categorical to continuous, I multiplied the number by a weighted average of the district-level case count for June and July as demonstrated below:



To find the monthly average case load summate the case count for each day (represented here as the shaded region) then divide by the number of days in the month.

Additionally, this allows district-wise analysis.

Making it in Stata:

In *district-wise\_cases\_deaths.dta* this calculation has already been caried out for 2020 data. If you desire to repeat the process for new case data, format the data to match that of \raw\covid\_infected\_deaths\_pc11.dta then run gen\_rising\_cases.do in the general folder, specifically the lines:

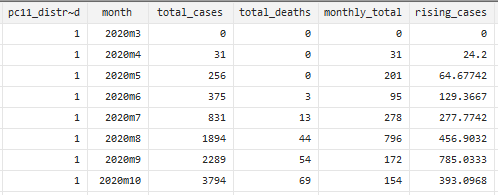
bysort pc11\_district\_id month: egen base\_count = min(total\_cases)

gen monthly\_total = total\_cases - base\_count

bysort pc11\_district\_id month: egen rising\_cases = mean(monthly\_total)

bysort pc11\_district\_id month: keep if \_n==\_N

Below you can see the variables calculations are repeated by month.



The code generating the continuous variable contained\_cases is found in line 128 of assemble\_all.do:

merge 1: pc11\_district\_id month using district-wise\_cases\_deaths, keepusing(rising\_cases)

drop if \_merge!=3

gen contained\_cases =rising\_cases\*containment\_days

drop \_merge rising\_cases light\_perK2 light\_perK2\_19 light\_perCapita\_19 containment\_days

drop if pc11\_district\_id==0

duplicates drop

sort pc11\_state\_id pc11\_district\_id month

order pc11\_state\_id pc11\_district\_id month

save monthly\_all, replace

Containment Zone Restrictions

I created a [list](https://docs.google.com/spreadsheets/d/1tisWrU2MjTdeRdTCL4rIk-Z9hd-GHb897bgFtlcXzGk/edit#gid=58744250) of containment zone restrictions based on online news articles and government releases. Most followed some form of guidance from the MOHFW, but others with limited cases or extenuating circumstances deviated greatly from the centre.

**Google Mobility Data:**

Google has made available [mobility data](https://www.google.com/covid19/mobility/) it has recorded for the purposes of COVID-related research. The “Lights Out” report published by the World Bank uses a similar approach with [Facebook sourced data](https://data.humdata.org/dataset/movement-range-maps). The main difference between the two data-sources is that Facebook data records the number of 600m x 600m square “tiles” an individual(‘s phone) passes through a day while Google data records movement between places of business and homes. As a result, Google data allows the distinction of specific movement patterns, say to grocery vs retail, but may not be as strong in rural areas or where these labeling schemes can break down.

I have used Google’s data so far because I have not been able to access Facebook’s data for India. Checking online for updated data which includes India or procuring the data from research channels will be required for comparative analysis.

Importing New Mobility Data:

Google and Facebook are appending their mobility data daily so analyzing more recent dates is possible. Once downloaded and unzipped into the “raw” folder, you can run the mobility\_painter.do file to import and sort the data.

Before running mobility\_painter.do, a key change needs to be made in line 15. This line drops data after 31 Dec 2020 and will need to be removed.

drop if date2 >= date("20210101","YMD")

Now that the mobility data is updated, run assemble\_all.do.

**Nighttime Light:**

Measures of nighttime light intensities has been used as a proxy for economic activity in several published articles including [Lights Out](https://www.dropbox.com/sh/vid96yr47zd9p65/AACdp9LXR9L8_2uT-0gqogEda?dl=0&preview=Lights-Out-COVID-19-Containment-Policies-and-Economic-Activity.pdf) (World Bank Group, 2020) which will serve as our guide.

Downloading Data:

Satellite images of the Earth at night are provided by the Earth Observation Group at the Colorado School of Mines. I recommend reading the [*file-manipulation*](https://eogdata.mines.edu/products/vnl/#download) section on their website as well as the description for [Monthly Cloud-free DNB Composite](https://eogdata.mines.edu/products/vnl/#monthly) to understand how the data is compiled before trying to unpack it. Download the data:

1. click the [*Download Tiled*](https://eogdata.mines.edu/nighttime_light/monthly/v10/) button
2. click your desired year
3. click your desired month
4. click vcmcfg
5. click the file whose fourth field contains “75N060E”

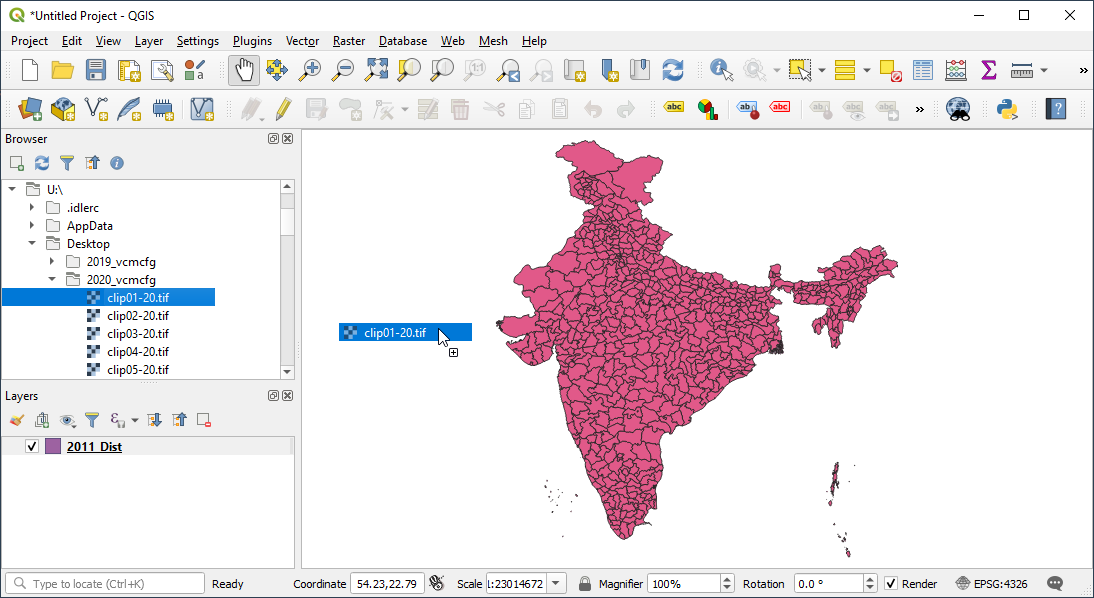


At this point you will be prompted to login to the Earth Observation Group system. If you do not already have an account, you can make one for free.

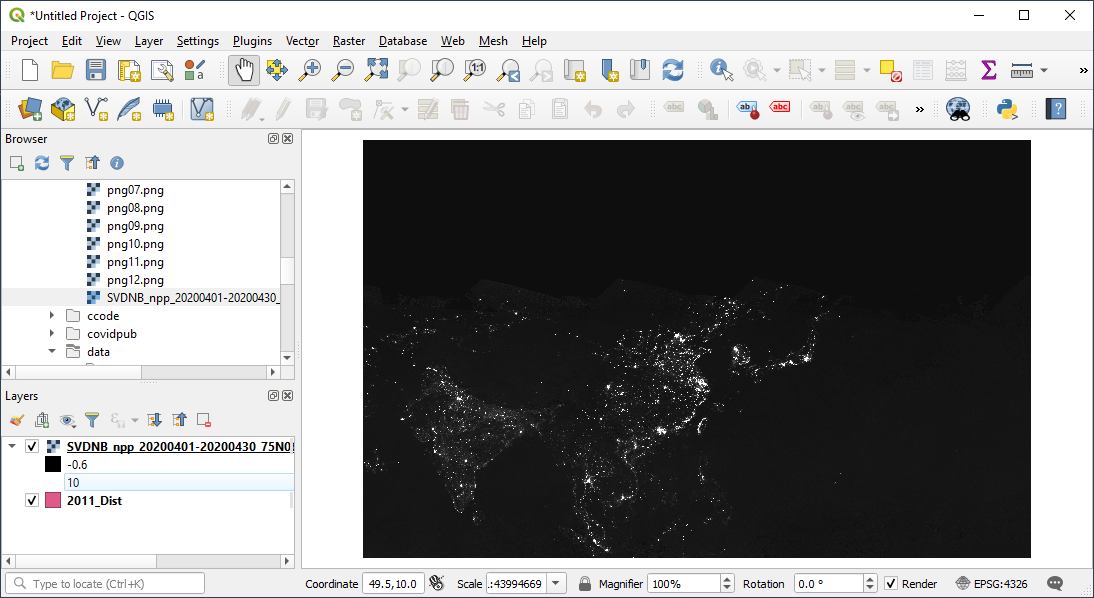
The image we want to access is in a double zipped folder. Use 7-zip to unpack *tgz* file, resulting in a *tar* file. Right-click on the extracted *tar* file and click 7-zip>open archive. Extract only the first file which ends in *avg\_rad9h*. This is the actual GeoTIFF we will now open in QGIS.

QGIS:

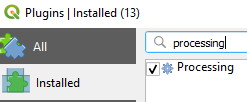
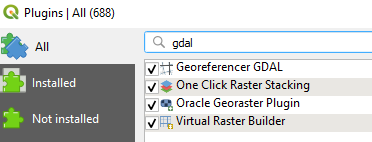
Open QGIS from the windows search bar and click “new empty project.” From the sidebar, navigate to the file where you have extracted the night light. Drag and drop all the GeoTIFF files into the workspace, then navigate to the shape files folder and drag in *2011\_Dist*. Make sure to use the *2011\_Dist* shapefile from GitHub as I have corrected 2 overlap errors.



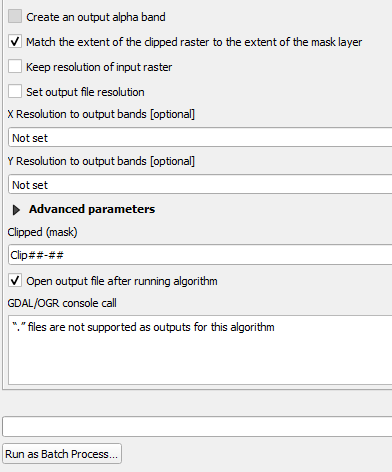
The GeoTIFF raster images should appear black. If you wish to see the lights you can change the color gradient down to a max of 10. Doing this has no effect of the data stored in the GeoTIFF and would only be for the aesthetical appreciation of the one preforming these steps.



Before we progress, navigate to Plugins> Manage and Install Plugins and make sure all GDAL and Processing Plugins are installed and enabled.



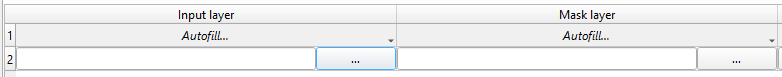
What you want to do now is trim the raster (image) to the outline of India (to decrease file size you) and then export the light data. Navigate to Raster> Extraction> Clip Raster by Mask Layer…. To clip a single raster image select it as the Input Layer. The Mask Layer should be the *2011\_Dist* shapefile provided in the GitHub repository. Other options are displayed below:

<check “Create an output alpha band” to have the raster area outside of India be transparent. Left unchecked, blackness will fill the square cropped around India. This is purely aesthetical; choose the option that suits your preference.

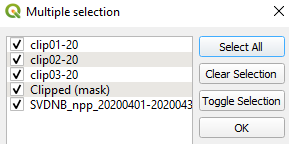
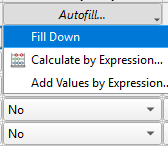
<type the name of the file you want to save the clipped image as. Leaving empty will create a temporary file that will be able to continue the process with, but it will be unnamed so you may confuse which month is which.

<Trim multiple months…

If you are working with multiple months’ worth of image data then you will probably want to run the extraction as a batch process. Click “Run as Batch Process” at the bottom left of the Clip Raster window.

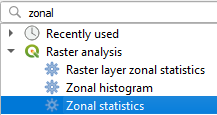
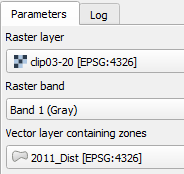
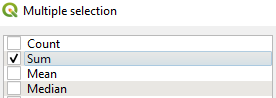


Click the “…” under the column to select the layers you want to reference. For input Layer, click “select all”> “OK.” For Mask layer, select *2011\_Dist*> “OK” then click the Autofill dropdown and select “Fill Down.” Follow these guidelines as you fill in Alpha Band option and name of the clipped files:

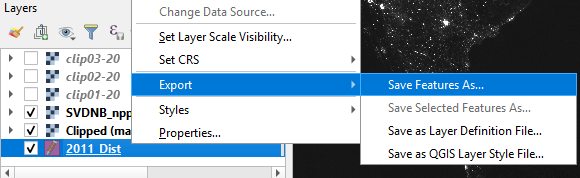
 

Once the rasters are clipped, we need to extract the data stored within. Navigate Processing> Toolbox and search ‘zonal’ and select “Zonal statistics.”

For extracting data from one month select it as the raster layer with *2011\_Dist* as the Vector layer. Leave the “output column prefix” bar as the default underscore. For multiple months, use a batch process.

To export the data, right click on *2011\_Dist* in the Layers window navigating Export> Save Features As. The data we just extracted is stored in the attribute table of this shapefile.



CSV to Stata

The current *import\_dist\_light* do-file imports data from a csv that contains radiance sums for each month of 2019 and 2020. Following this do-file’s format you can copy and paste the sections that need repeating for whichever years/months you want to add. Let me explain the code:

After importing the delimited csv data, rename the variables to display their month and year:

rename (\_sum \_sum\_1 \_sum\_2 \_sum\_3 …) (sum01\_20 sum02\_20 sum03\_20 …)

rename (v18 v19 v20 v21 …) (sum01\_19 sum02\_19 sum03\_19 …)

Note how the variables have sub, sub\_1, sub\_2, etc. after the underscore we left as the prefix in QGIS. Also note that in the second row the variables are initially labeled v18, v19, etc. This is because I extracted the radiance data for all of months 1-12 of 2020 – corresponding to sum-sum\_11 – then did the same thing months 1-12 of 2019. This resulted in months 1-12 of 2019 having the same variable name (sum-sum\_11) which state converted to variable column numbers to avoid repetition. If you extract data from 5 years at once your suffixes will run from sum to sum\_59 and you will not need to worry about v## in stata. If you want to extract in smaller batches, be aware this will happen.

Next we import census names and id’s

merge 1:1 pc11\_state\_name pc11\_district\_id using \\rschfs1x\userrs\a-e\bp257\_RS\Desktop\data\general/2011\_Dist\_1

merge 1:1 pc11\_state\_id pc11\_district\_id using \\rschfs1x\userrs\a-e\bp257\_RS\Desktop\data\general/census\_data

then create the variables for light per Capita and per square-Kilometer year by year

drop sum01\_20 sum02\_20 sum03\_20 sum04\_20 sum05\_20 sum06\_20 sum07\_20 sum08\_20 sum09\_20 sum10\_20 sum11\_20 sum12\_20

foreach month in sum01\_19 sum02\_19 sum03\_19 sum04\_19 sum05\_19 sum06\_19 sum07\_19 sum08\_19 sum09\_19 sum10\_19 sum11\_19 sum12\_19 {

gen `month'\_perCapita = `month'/pc11\_pop

gen `month'\_perK2 = `month'/pc11\_area

}

save dist\_sum\_19, replace

Copy and paste the above section for the years/months you need.

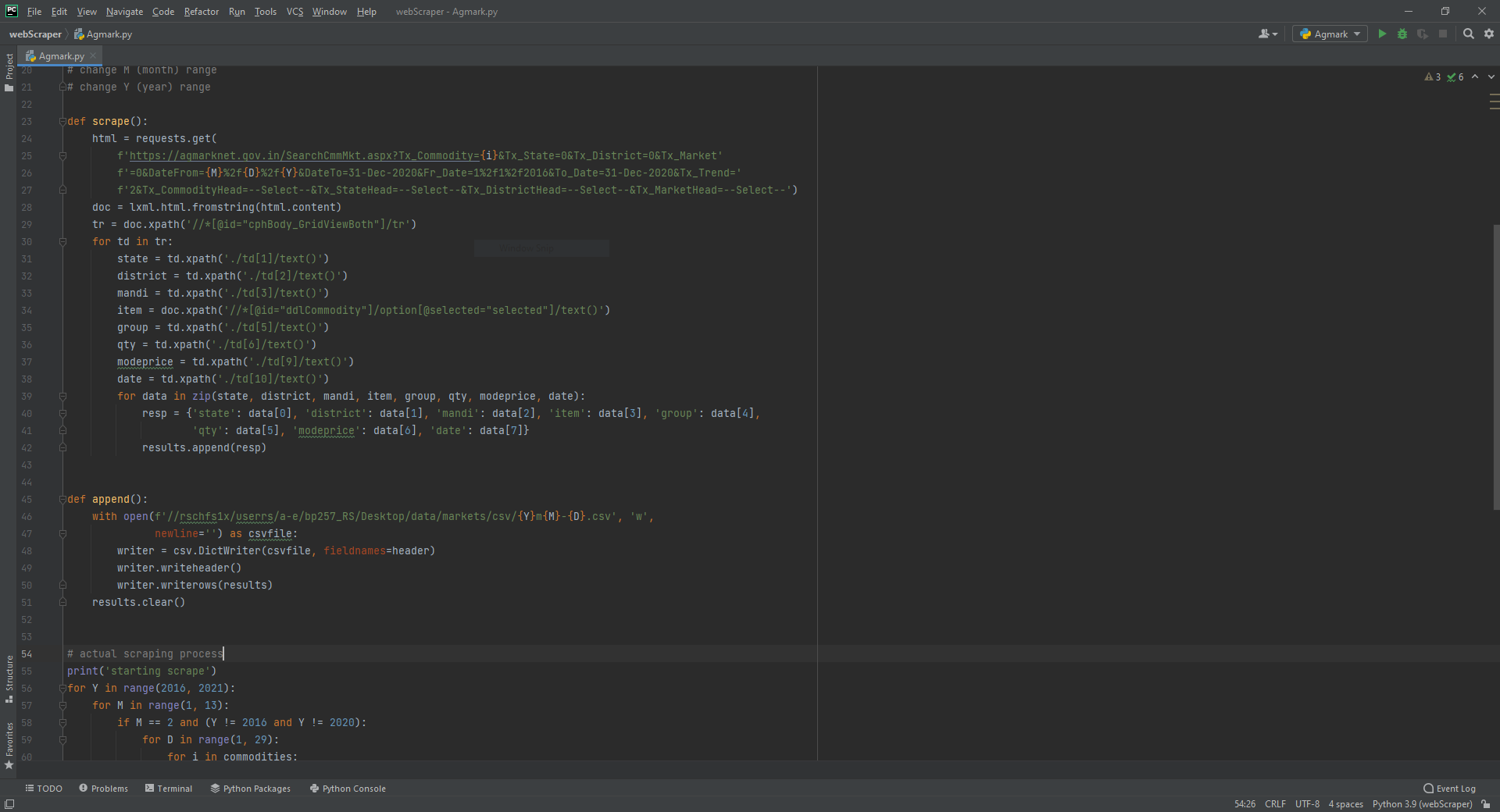
In *assemble\_monthly\_light* the nightlight data is rearranged and saved as *monthly\_lights* before being merged into *monthly\_all*.

**Web Scraping Agmarknet.gov.in Using Python:**

[Agmarknet.gov.in](https://cornellprod-my.sharepoint.com/personal/bp257_cornell_edu/Documents/agmarknet.gov.in) is a website that stores the daily prices (high, low, and mode) and arrival quantity of 347 commodities in every mandi in India. Mode-price and quantity data are used in our analysis of the COVID-19 lockdown. We already have prepared Agmark data from the DDL repository for the years 2018-2020 however we want to expand our data to include years since 2016, or even 2014. We will accomplish this through automated web scraping using Python code. Here I will walk you through how to operate the operate the code and how to troubleshoot any issues.

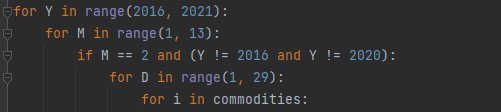
Parts of the code:

The code consists of the functions scraping() and append() nested within a series of loops that account for the varying days across different months.

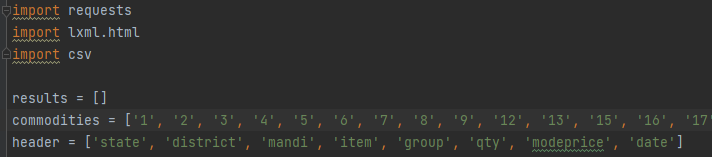


This code **(A)** opens the webpage of the target item and date, **(B)** saves the target data to a list, **(C)** orders and appends data to a list which will be printed after each month **(D)**.

To set the time range in which to scrape Agmarknet, change the Y and M range to the desired year and month(s). Note how Python reads a list of numbers, *range(5)* will be read as *(0, 1, 2, 3, 4)* and *range(1, 4)* will read as *(1, 2, 3)*. In other words, Python will count from the first number (or 0) up to, but not including, the later number.



The first part of the code tells Python to import certain libraries whose stored commands will allow web scraping functionalities. Underneath, we define the variable lists holding results, important commodities, and headers which are used when appending and exporting our data.



Installing necessary programs:

To run the code on ciser we need to operate within a virtual environment to bypass restrictions on installing external programs like the libraries we need. ([steps](https://ciser.cornell.edu/anaconda-install-packages-create-new-virtual-environment/))

In the windows search bar open ANACONDA PROMPT and type the following lines one at a time

U:

cd Documents

mkdir "conda\_dir"

cd "conda\_dir"

conda create --prefix ./myenv python=3.8

When prompted, respond “y” then wait for the process to complete.

Now install lxml and requests libraries.

pip install lxml requests

If you install a wrong library due to a typo, simply use

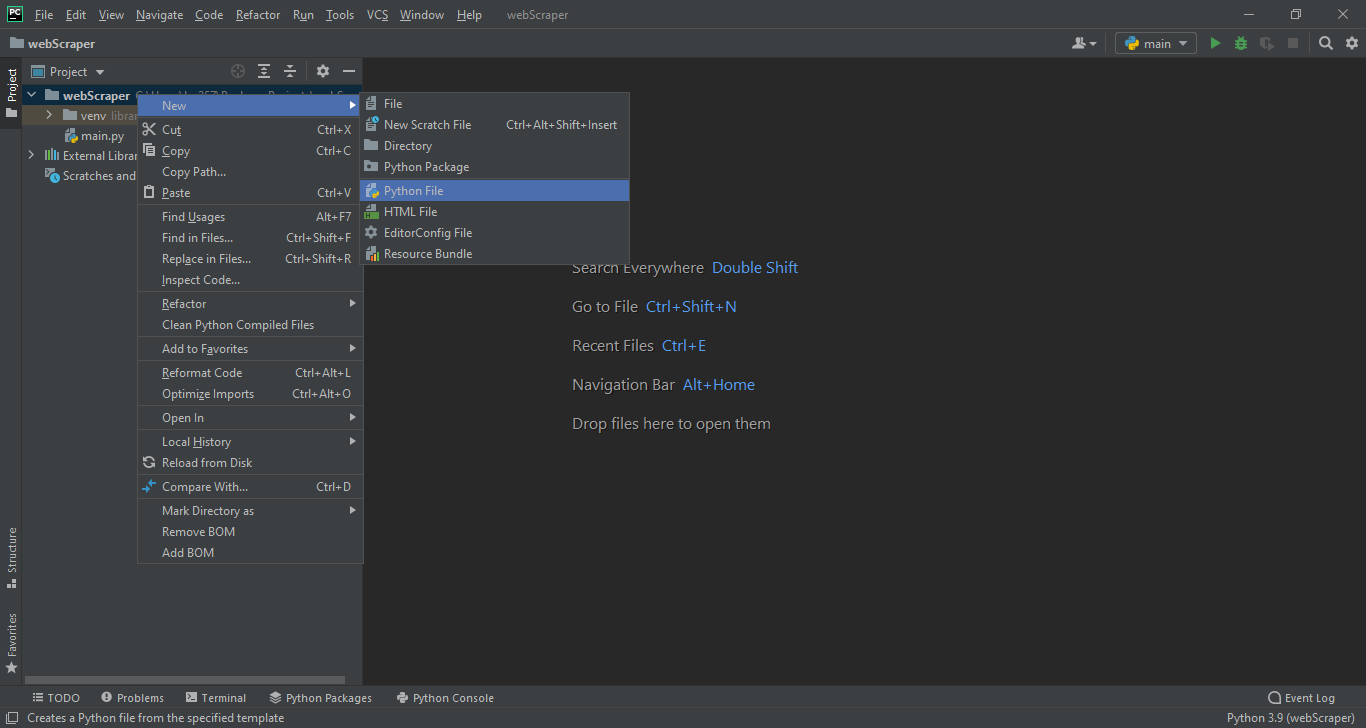
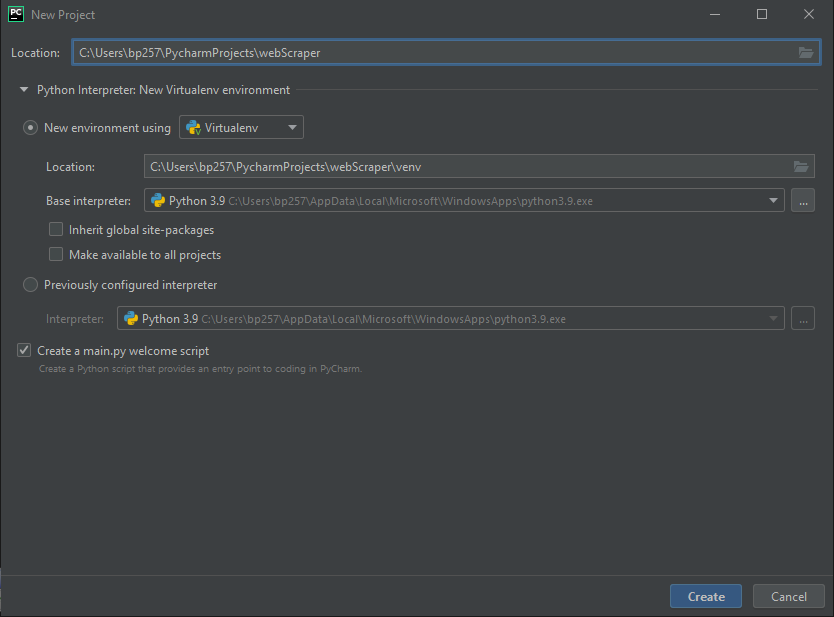
pip uninstall *typo*

Running the code:

I recommend using PyCharm (open from windows search bar) or IDLE (right-click>edit with IDLE) to edit the Agmark code. Pycharm is built into ciser and easy to use.

The first time you open PyCharm open a new project using Python called webScraper. Then Right-Click the webScraper tab in the *projects* sidebar. Click New> Python File and name it *Agmark*.

Copy and paste the code from the Agmark script from the repository into the new Agmark file you just created. This seemingly roundabout method will ensure that the necessary files in the .idea and env folders correspond with Agmark.



Once you have reached this part, scrapping Agmarknet for a desired year and month should be straightforward. ([steps](https://stackoverflow.com/questions/39995380/how-to-use-anaconda-python-to-execute-a-py-file))

If you have not already, open the Anaconda Prompt window and enter the following lines to 1) activate your virtual environment, 2) set directory to your *PythonProject*, 3) run *Agmark*

activate "U:\Documents\conda\_dir\myenv"

cd U:\\Users\NetID\PythonProjects\webScrape

python Agmark.py

The Process takes 30 minutes for all items in a day, or about 14.5 hours for a month. Utilizing ciser’s remote servers to collect multiple months at once is crucial to expediting to scraping process.

The data file is saved as “{Y}m{M}-{D}.csv”, ea. “2017m2-15.csv”. The month is split into two files indicated by the last day in the data. This more frequent saving will reduce the memory demand and the amount of data loss in the case of disconnection.

Importing Newly Scrapped Data:

Incorporating new data is straightforward. Run the *assemble\_&append* then the *ordering\_*agmark do-files to put all the scraped data into a workable format. Add additional years/months by pasting in the following code to line 50. Change the year of the data you want to incorporate. If you are appending a leap year, you will also need to change to '2-28' to '2-29.

Additionally, if you are not adding an entire year of observations, you will need to remove unscraped months from the foreach list.

foreach ds in 1-31 2-15 2-28 3-15 4-15 4-30 5-15 5-31 6-15 6-30 7-15 7-31 8-15 8-31 9-15 9-30 10-15 10-31 11-15 11-30 12-15 12-31{

clear

import delimited \\rschfs1x\userrs\a-e\bp257\_RS\Desktop\data\markets\csv/2016m`ds'.csv, varnames(1)

drop if date == "-"

append using agmark\_append

}

The first section of code in *assemble\_&append* differs slightly than the above. This creates the *agmark\_append* data-file with the first half-month’s data then appends the remaining months of the first year to the bottom of *agmark\_append*. The sections of code for following years can append all the months in one loop.

The code following this section renames the variables and creates new variables that will be used in analyses:

gen month= mofd(date)

bysort date pc11\_district\_id group: egen qtyperday =mean(qty)

bysort date pc11\_district\_id group: egen priceperday =mean(modeprice)

bysort month pc11\_district\_id group: egen monthly\_qty =mean(qtyperday)

bysort month pc11\_district\_id group: egen monthly\_price =mean(priceperday)

Moving to *ordering­\_agmark.do* we will align the data to allow comparison across different years. Establishing the format with the year 2020, we use the code:

keep if month>tm(2019m12)&month<tm(2021m1)

rename (monthly\_qty monthly\_price) (qty\_20 price\_20)

save agmark\_monthly, replace

We merge the other years one at a time, in a process similar to how we appended months in *assemble­\_&append.do*:

use agmark\_append, clear

keep if month>tm(2018m12)&month<tm(2020m1)

rename (monthly\_qty monthly\_price) (qty\_19 price\_19)

replace month =720 if month==tm(2019m1)

replace month =721 if month==tm(2019m2)

...

replace month =731 if month==tm(2019m12)

merge 1:1 month pc11\_district\_id group using agmark\_monthly

drop \_merge

save agmark\_monthly, replace

Replacing month values will allow January 2019 (in this example code) to align with January 2020 in the using data. After this process is complete, the comparison of the price and quantity of commodity groups in a mandi can be analyzed across years.

The last section of code creates variables based on market trends over the imported years. We implore a difference in differences approach that compares the expect price- and quantity growth rates to the actual differences between the years 2019-2020.

//calculating difference in differences of growth rates

//price

gen AGR16 = (price\_17 - price\_16)/price\_16 + 1

gen AGR17 = (price\_18 - price\_17)/price\_17 + 1

gen AGR18 = (price\_19 - price\_18)/price\_18 + 1

gen diff20 = price\_20 - price\_19

egen AARG = rowmean(AGR16 AGR17 AGR18)

gen deviation\_price20 = diff20 - (AARG \* price\_19)

//for 2021:

\*gen diff21 = price\_21 - price\_20

\*gen deviation\_price21 = diff21 - (AARG \* price\_20)

The code above establishes an average annual growth rate (AAGR) for price by averaging the growth rate between each year. Variable “diff20” represents the difference between 2019 and 2020 price values. “AAGR” is then multiplied by the price values for 2019 before being deducted from the 2020 price values. The resulting variable “deviation\_price20” is the difference between the counterfactual expected price-increase between 2019-2020, and the change that actually occurred. The last 2 lines in the sample code can be un-commented to calculate the DID for 2021 using the AAGR from 2020 prices as the counterfactual.

Merge With *monthly­\_all*:

From This point, you can merge the agmark data into the *monthly\_all* data-file for regressions. I have not saved a merged *monthly\_all* to the GitHub folder because the file would be large and to make easier the possibility to append more years/months to *agmark­\_monthly* before a merge. The line below should work for merging either in the console or line 134 of *assemble\_all* in the general folder:

Merge 1:m pc11\_district\_id month using agmark\_monthly, keepusing(group deviation\_price20 deviation\_qty20)

Dropping observations that are outside of the window of months you are looking at, or non-important commodity groups might save time in the merge, however using the *if* attribute when running regressions will provide more flexibility in what you can include in regressions or visualizations. The choice is yours.