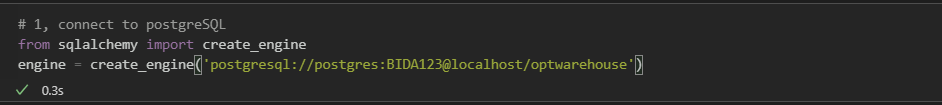
**USER GUIDE**

**MOCKING OF DATA FOR WAREHOUSE OPTIMIZATION PROJECT (Juypter notebook)**

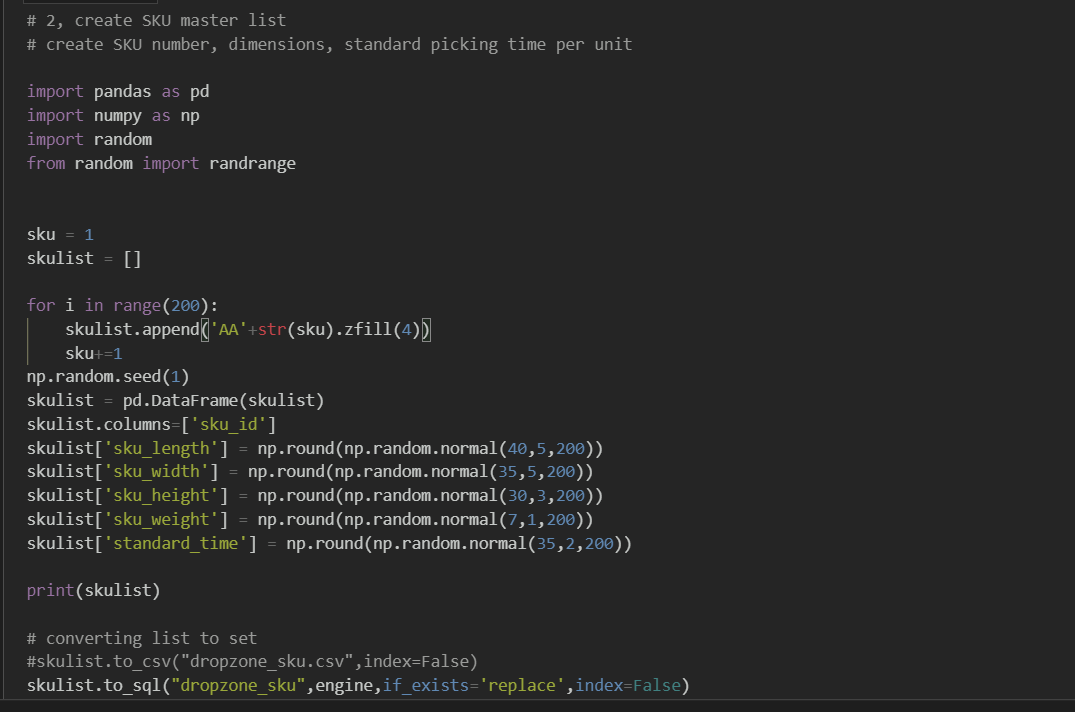
1. **Create a database in pgAdmin**



1. **Create “dropzone\_sku” table**

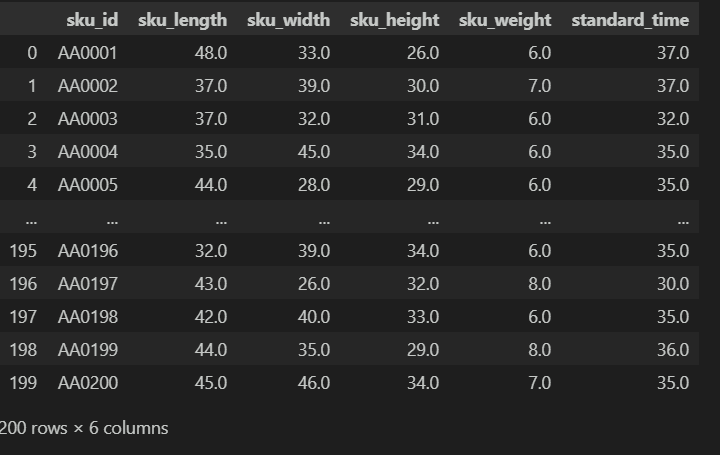
Libraries used in Python: pandas, numpy, random

To create 200 SKU ID.



* Using **pandas**. Create dataframe named **[skulist]**
* Named column as **[sku\_id]**
* Using **random and numpy** library. Gaussian distribution (round into integer as Gaussian return float) to create rest of the columns **[sku\_length,sku\_width,sku\_height,sku\_weight,standard\_time]**
* Set [**sku]** default as 1
* Create an empty list called [**skulist]**
* **For** loop for 200 sku ID
* Convert [**sku]** to string and concat with string ”AA”. Use zfill to add zero in front of an integer to form a SKU ID (AA0001)
* Append to [**skulist]**
* **[sku]** increment of 1 after append
* create random seed
* Import libraries
* To Postgresql

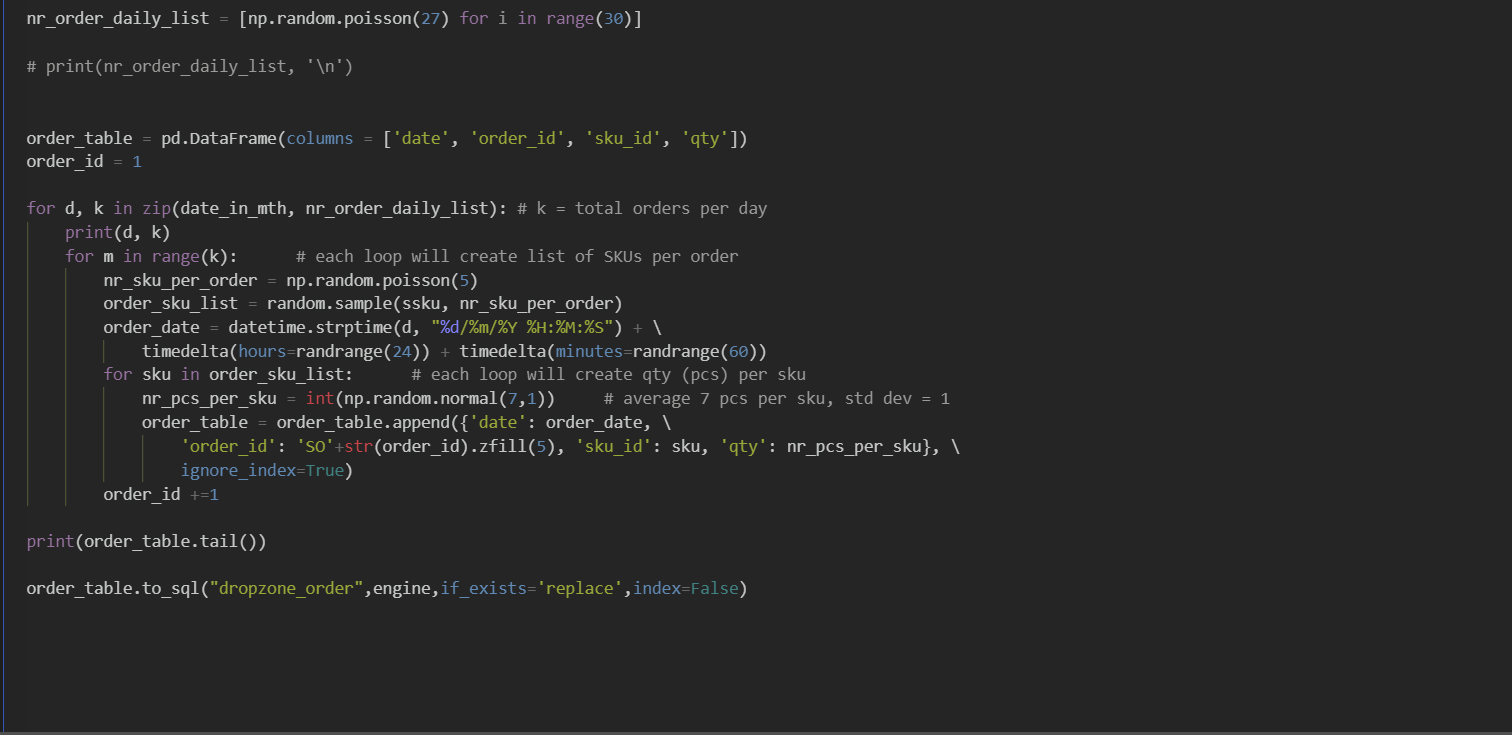
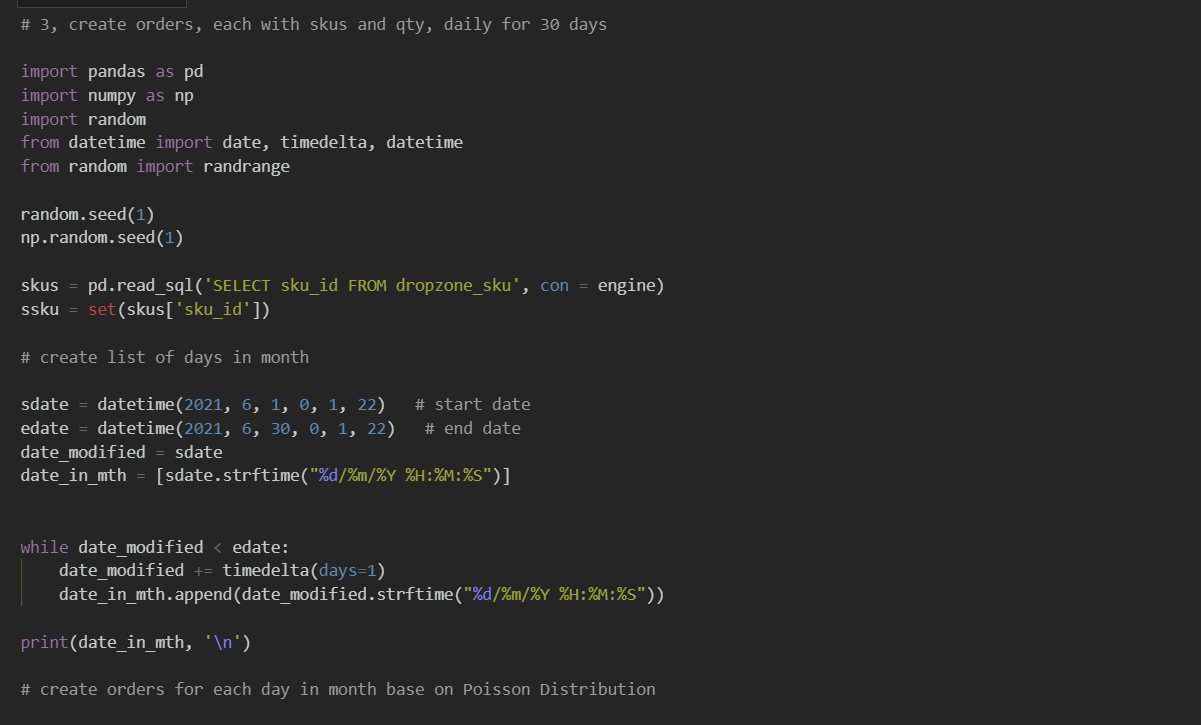
**OUTPUT**  **“dropzone\_sku” table**



1. **Create “dropzone\_order” table**

Libraries used in Python: pandas, numpy, random, datetime

To create orders based on sku\_id and quantity for June 2021



* Create a **while** loop so when **[date\_modified] < [edate] …**
* Add 1 day to **[date\_modified]** using **timedelta**
* Append **[date\_modified] to** **[date\_in\_mth]** until is > **[edate]**
* Using Poisson distribution of 27 to create **[nr\_order\_daily\_list]** for 30 days
* Import libraries
* Create dataframe named **order\_table** with column

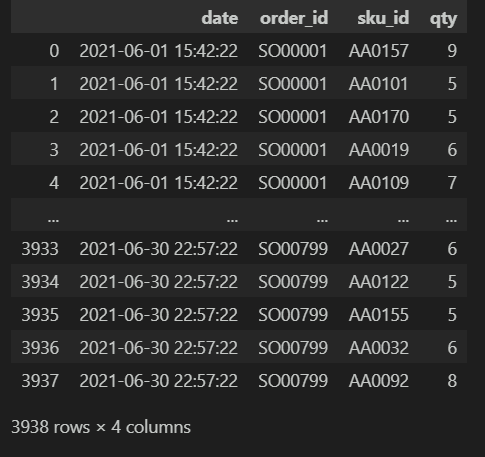
**[date, order\_id, sku\_id, qty]**

* Set [**order\_id]** default as 1
* Using **for** loop to loop through **d** **[date\_in\_mth]**, **k** **[nr\_order\_daily\_list]**
* In range of **k ,** using Poisson distribution of 5 to create **[nr\_sku\_per\_order] ,**

Using **random sample**  **[ssku, nr\_sku\_per\_order]** create **[order\_sku\_list]** and generate datetime with random hours and minutes called **[order\_date]**

* **for** **sku in order\_sku\_list,** usingGaussian to createrandom **[nr\_pcs\_per\_sku]**  Append **[order\_date**], **[order\_id]** + zfill and concat with “SO”, **[sku]**, **[nr\_pcs\_per\_sku]** to **order\_table**
* **[order\_id] + 1**
* To Postgresql
* Declare a startdate named **[sdate]**
* Declare an enddate named **[edate]**
* Declare **[date\_modified]** = **[sdate]**
* convert **[sdate]** to string using **strftime** and declare in list as **[date\_in\_mth]**
* Read and Select **[sku\_id]** from **dropzone\_sku** table in PostgreSQL
* Create a set with**[sku\_id]** named **[ssku]**
* Create Random seed

**OUTPUT**  **“dropzone\_order” table**



1. **Create “dropzone\_batch\_order” table**

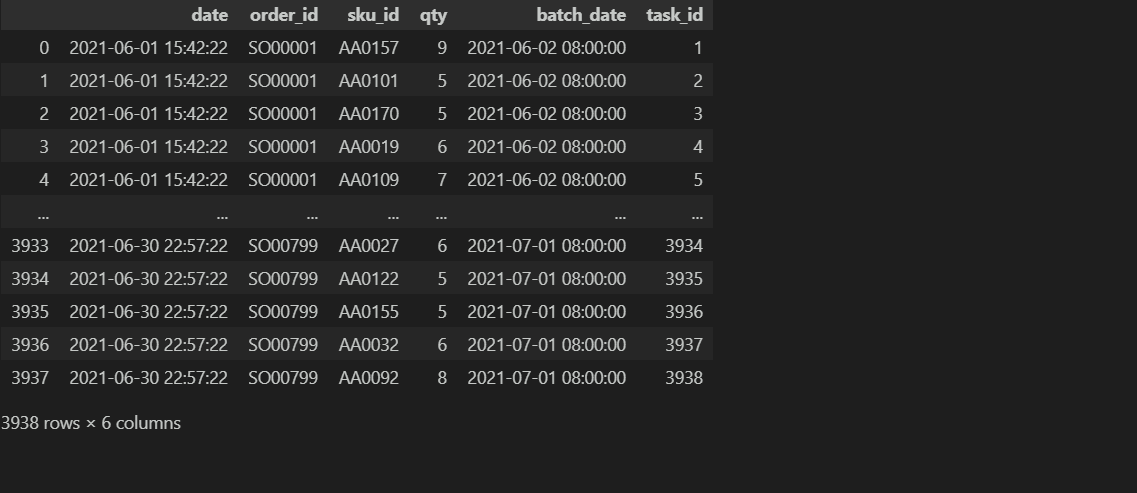
Libraries used in Python: pandas, datetime, numpy

Batching the orders based on hour of day. Setting condition where orders beyond the working hour of the day will move to the next day.



* To Postgresql
* Create **[batch\_date]** by combining **[ date\_1]** & **[timestamp]**
* Do a reset of the index
* Create **[task\_id]** byadding +1 to index
* Drop unnecessary columns **[timestamp,date\_1, date\_hour]**
* Using pandas, read and Select **ALL(\*)** from **dropzone\_order** table in Postgresql
* Create a condition **nextday** for **[date\_hour] if [date\_hour] above 15 and less than 24 ,** return value **date** = **([date ]+ 1 day)**, else **[date\_hour] above 0 and less than 15** return value **date** = **[date ]**
* Stored the values as **[date\_1]**
* Create a condition **roundingtime** for batching to batch sku\_id at different **[date\_hour]** and return the **values**
* Create **[timestamp]** to store the **values**
* Convert **[date]** for string to datetime
* Create **[date\_hour]** to extract hour from **[date]** using **pd.DatetimeIndex**
* Import libraries

**OUTPUT**  **“dropzone\_batch\_order” table**



1. **Create “dropzone\_workers\_all” & dropzone\_workers tables**

Libraries used in Python: pandas, faker, numpy

Create a base 20 max number of worker. Have to create different amount of workers to run simulation. 

* To Postgresql to run different amount of workers simulation
* Change the no of worker in **num\_worker** circled in blue
* Change the **“dropzone\_worker”** table name with the num worker added behind etc **(dropzone\_worker\_3)**
* To Postgresql for fixed 20 workers
* Create a list called **df**
* Set **serial** as 1
* Using **for** loop , create 20 **[worker\_id]** using **serial** and **zfill** and concat with “W” , 20 names using **faker**
* **serial** + 1
* create pandas dataframe **[df]**
* create **[salaryperhr]** to randomly assign the worker salary from 7 to 14 using **np.random. randint**
* set **[df1]** as **df[:numworker]**
* Declare **num\_worker** to amount of worker needed to run the simulation
* Create a faker seed and random seed
* Import libraries

**OUTPUT**  **“dropzone\_workers\_all”**



**OUTPUT dropzone\_workers\_7 table**



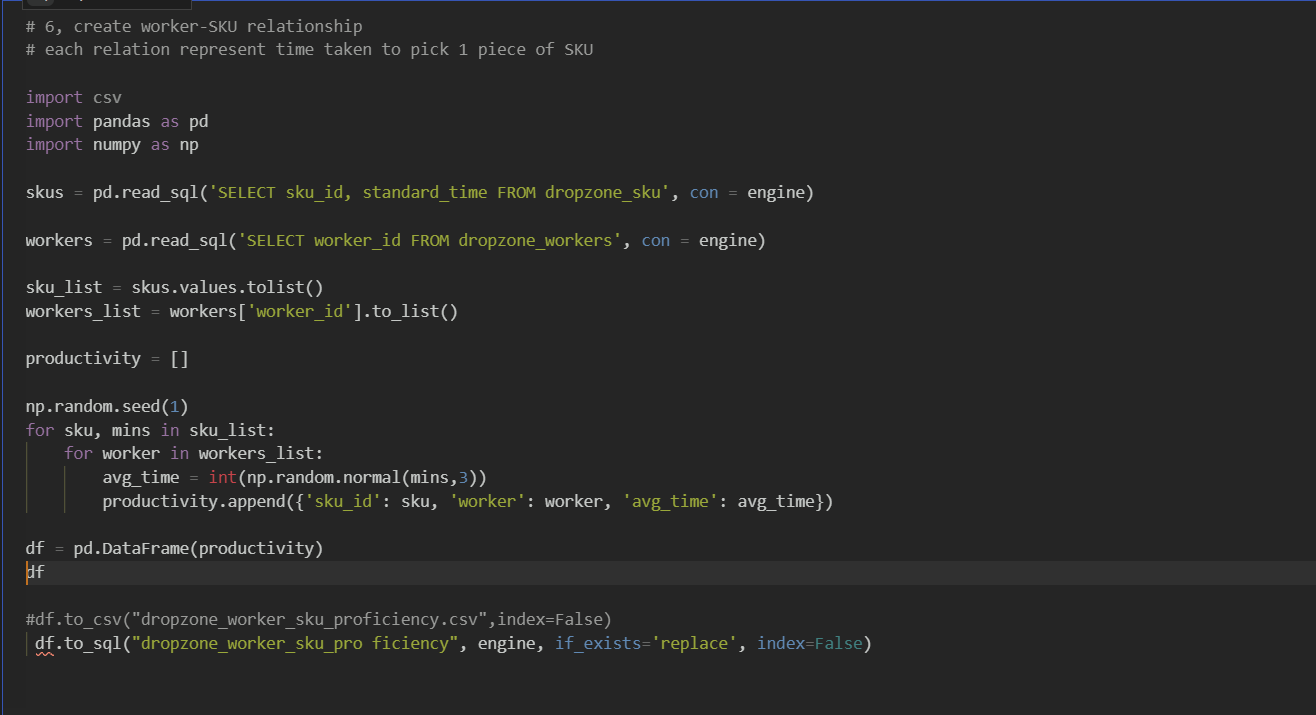
**OUTPUT dropzone\_workers\_3 table**



1. **Create “dropzone\_workers\_sku\_proficiency” table**

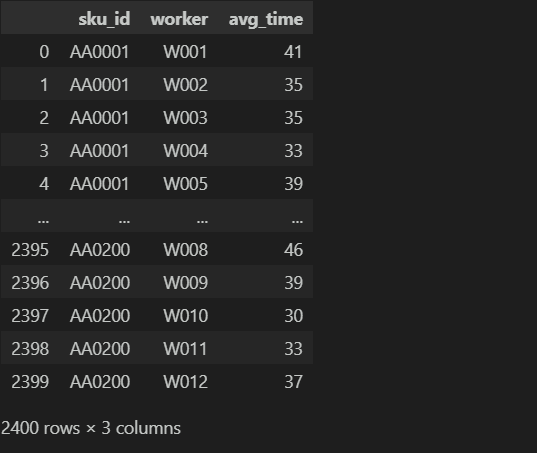
Libraries used in Python: pandas, faker, numpy

Create each worker’s average time to pick up each sku using [standard\_time] in dropzone\_sku



* To Postgresql
* Create random seed
* Using **for** loop , using Gaussian distribution to create an average time for each worker to pick for each sku.
* Append to **[productivity]** and put in dataframe **[df]**
* Create **sku\_list** to store **skus** values and **worker\_list** to store workers**[‘worker\_id’]** values
* Create an empty list named **productivity**
* Using pandas, read and Select **[sku\_id],[standard\_time]** from **dropzone\_sku** table in Postgresql name as **skus**
* Using pandas, read and Select **[worker\_id]** from **dropzone\_workers** table in Postgresql name as **workers**
* Import libraries

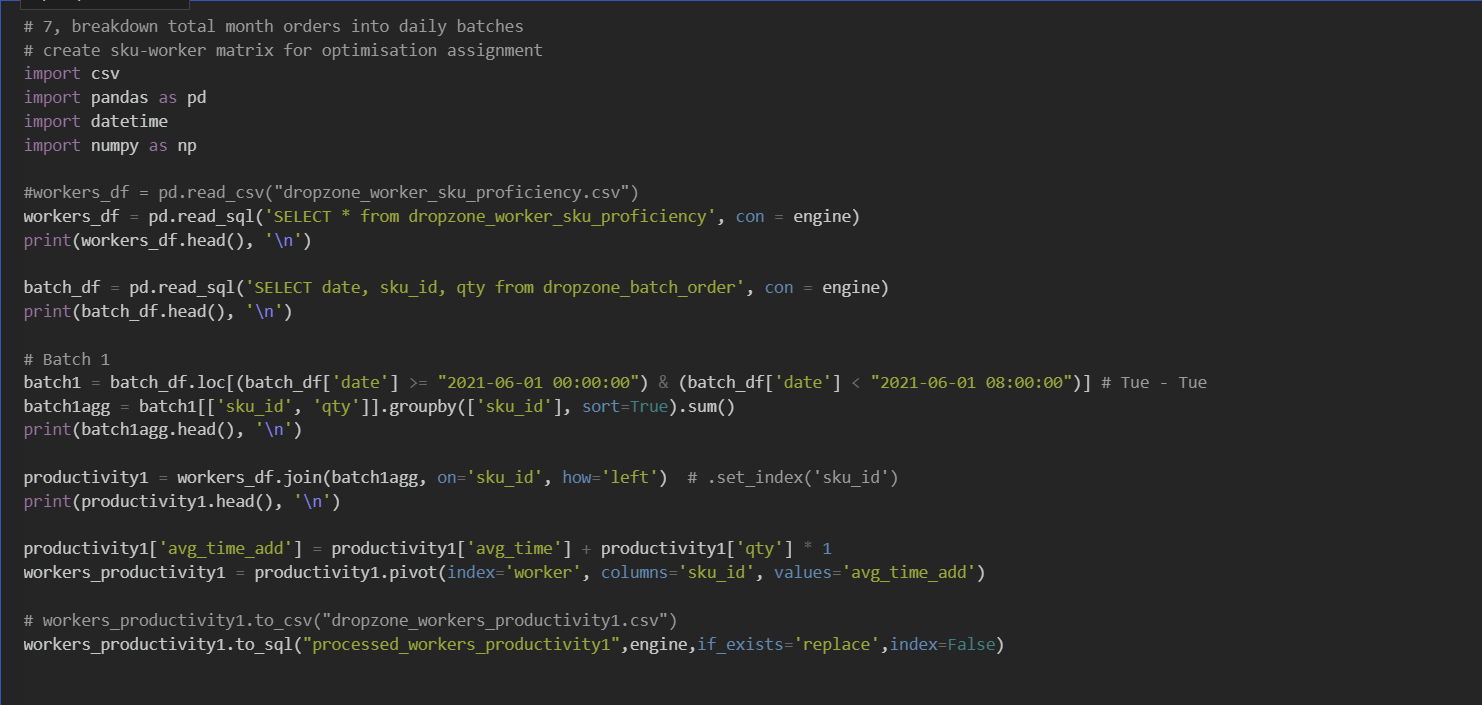
**OUTPUT**  **“dropzone\_workers\_sku\_proficiency” table**



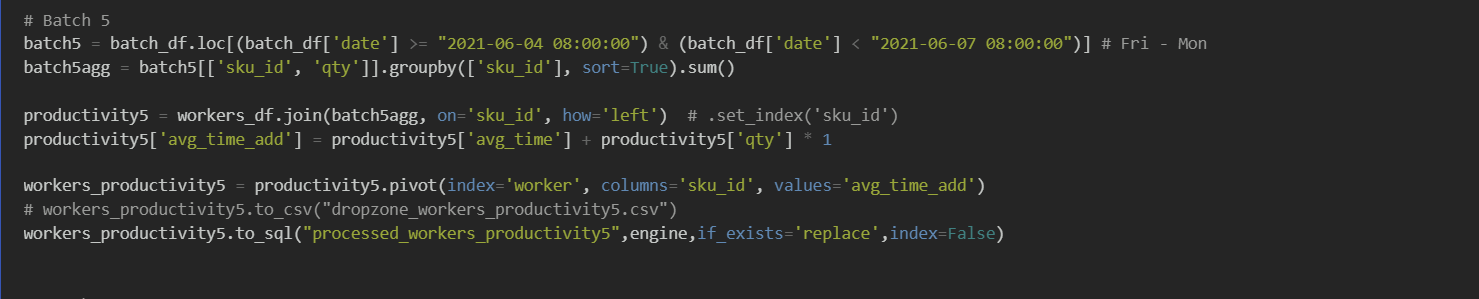
1. **Create “processed\_workers\_productivity” tables**

Libraries used in Python: csv, pandas, datetime, numpy

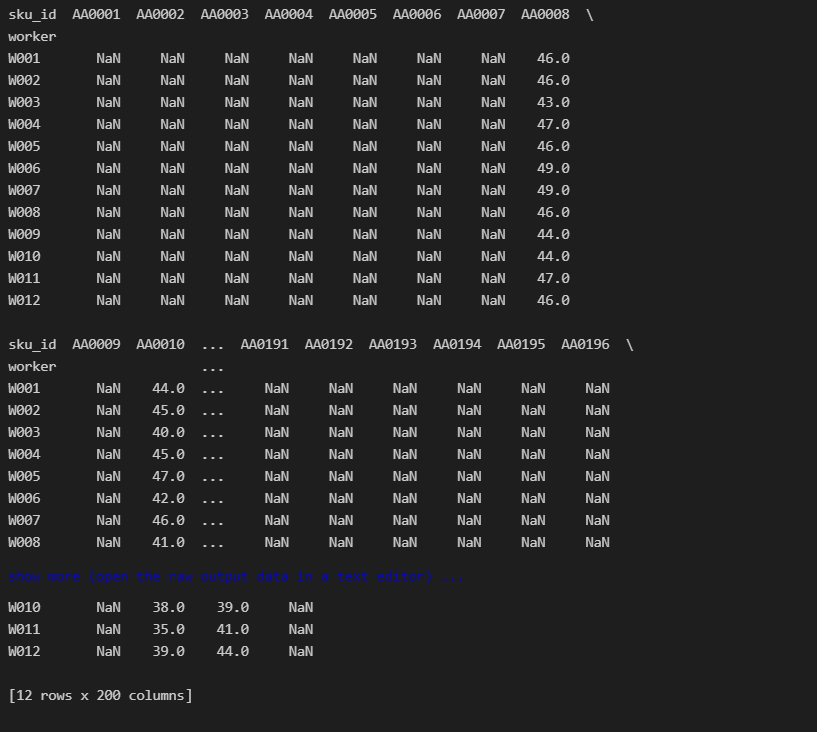
To create different tables/batches according to day of month and day of week needed to run in Google OR tools.



* Batch it according to different day of week for 1 month. **(circled in blue)**
* Concat Weekend batches to Monday **(example of batch 5 below)**
* To Postgresql
* Create a condition by day of week and date named **batch1**
* Groupby **[sku\_id]** and sum up the **[qty]** named as **batch1agg**
* Join **worker\_df** and **batch1agg** on **[sku\_id]** named as **productivity1**
* **[avg\_time]** add and set 1 min to pick up per **[qty]** and named as **[avg\_time\_add]**
* Pivot the dataframe to match the GoogleOR algorithm format and named as **[workers\_productivity1]**
* Using pandas, read and Select **[All(\*)]** from **dropzone\_worker\_sku\_proficiency** table in Postgresql name as **workers\_df**
* Using pandas, read and Select **[date,sku\_id,qty]** from **dropzone\_batch\_order** table in Postgresql name as **batch\_df**
* Import libraries



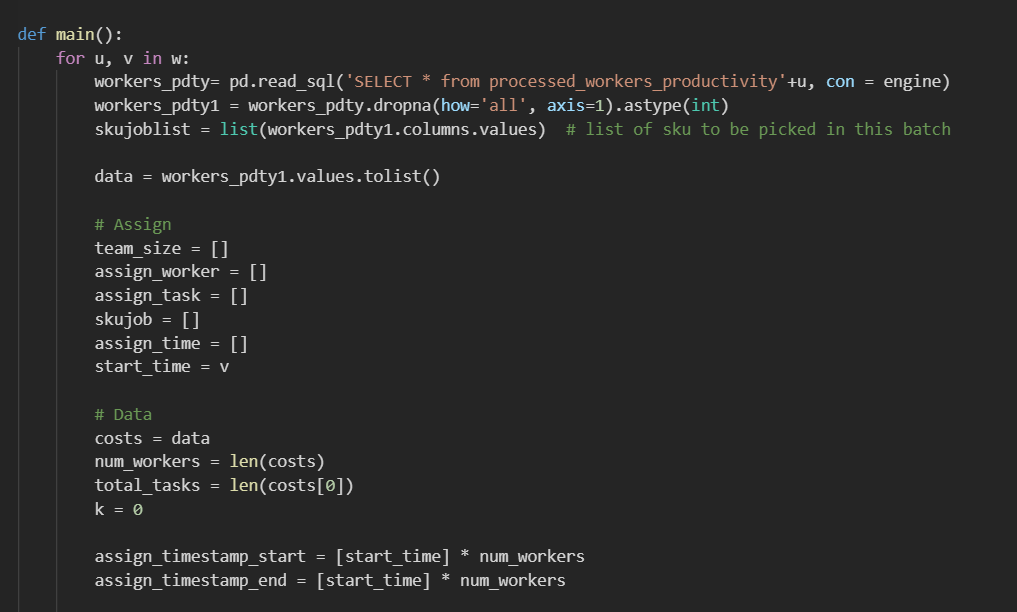
**OUTPUT**  **“processed\_workers\_productivity1” table**



1. **Run OR (Operational Research) Optimization to calculate optimal job assignments**



* This is the start of the OR Solver code, adapted from Google OR.
* OR stands for Operational Research
* Import libraries
* Make a list of tuples consisting of suffix id of tables (created in step 7) and the workday and start time.



Initialize list to track start and end time of each job by each worker.

Shape of dataframe is determined to for OR algorithm to plan its calculation.

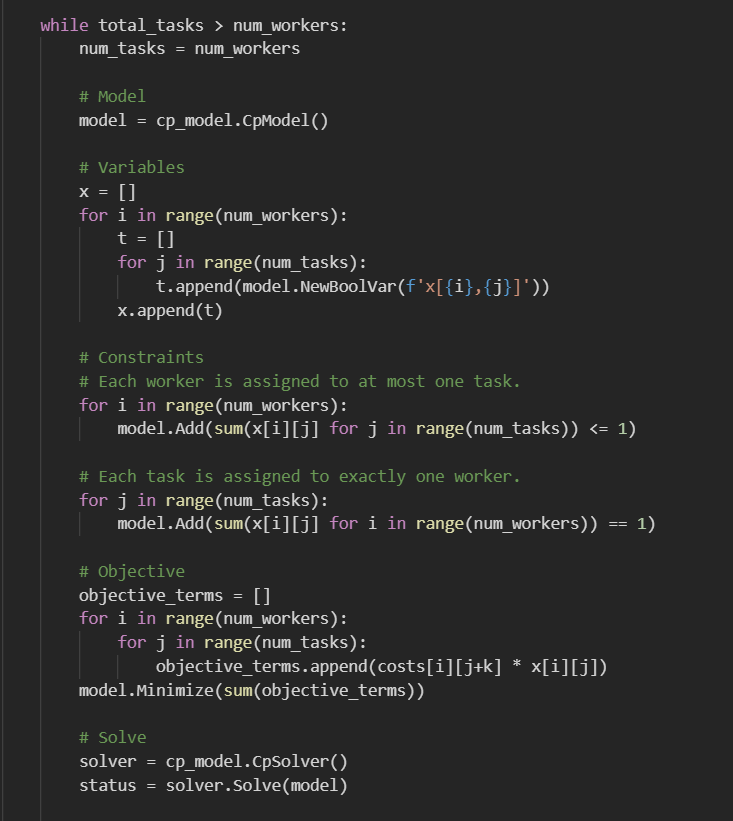
The dataframe is an array consisting of time taken (in mins).

It is assigned to the variable “costs” because time\_taken can be regarded as “costs”.

Create empty list to track each variable.

List of SKU is extracted so that it is tracked for each job assigned to each worker.

* This is for-loop to run the optimization code on each day batched orders.
* Empty columns in dataframe has to be removed before feeding it into the OR algorithm.



Logic check in algorithm.

Number of workers must always be more than or equal to number of tasks, otherwise there is no feasible solution, and the Solver will end.

On a typical day in the warehouse operation, there is always more tasks than workers. So, upon completion of a task, workers will have to come back and perform the next task.

This repeats until all tasks are completed.

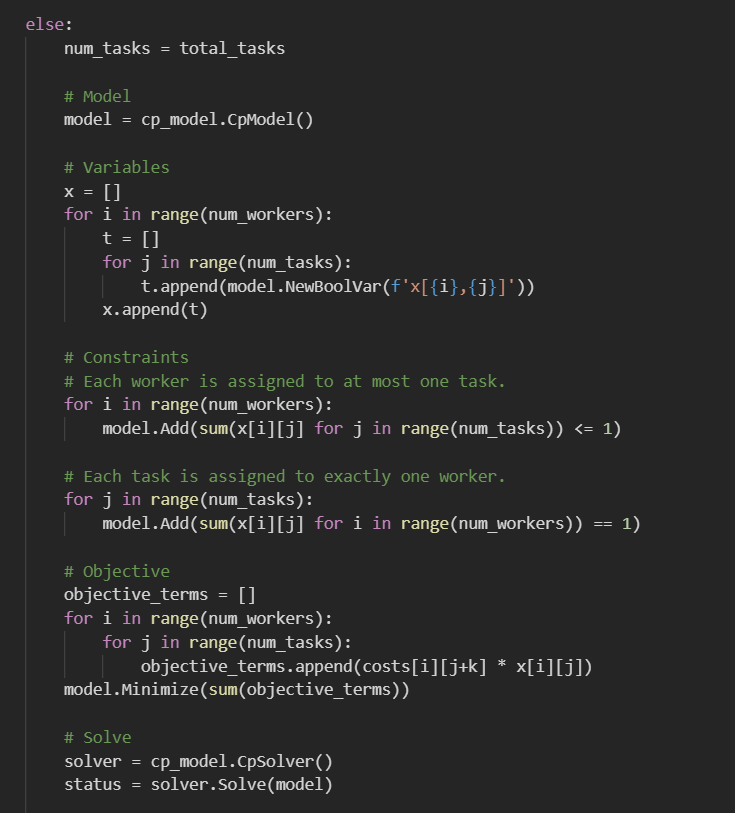
So, for each loop of job assignment, we set number of tasks (maximum) as equal to number of workers.

In the final loop, the remaining tasks will be less than number of workers. This will be processed in the Else-loop in the code below.

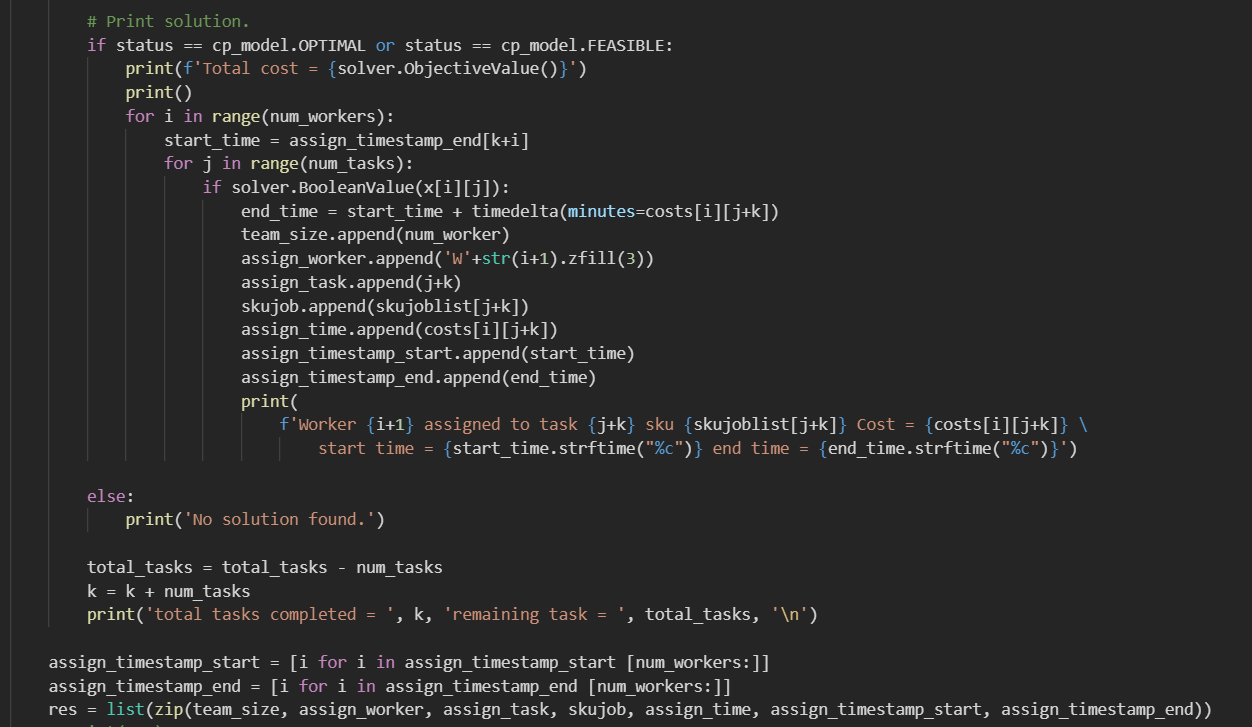


Update counter and prepare for next loop (job assignment).

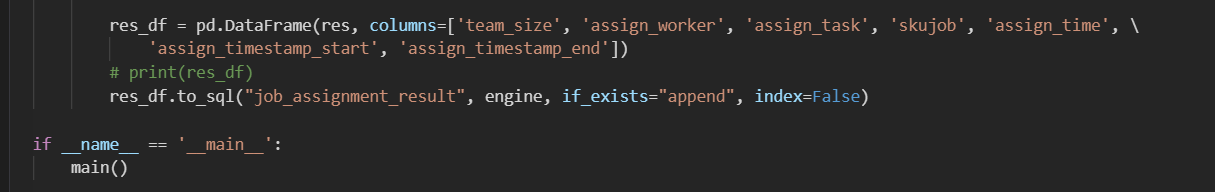
Register the job assignments, track the time stamps of each job.



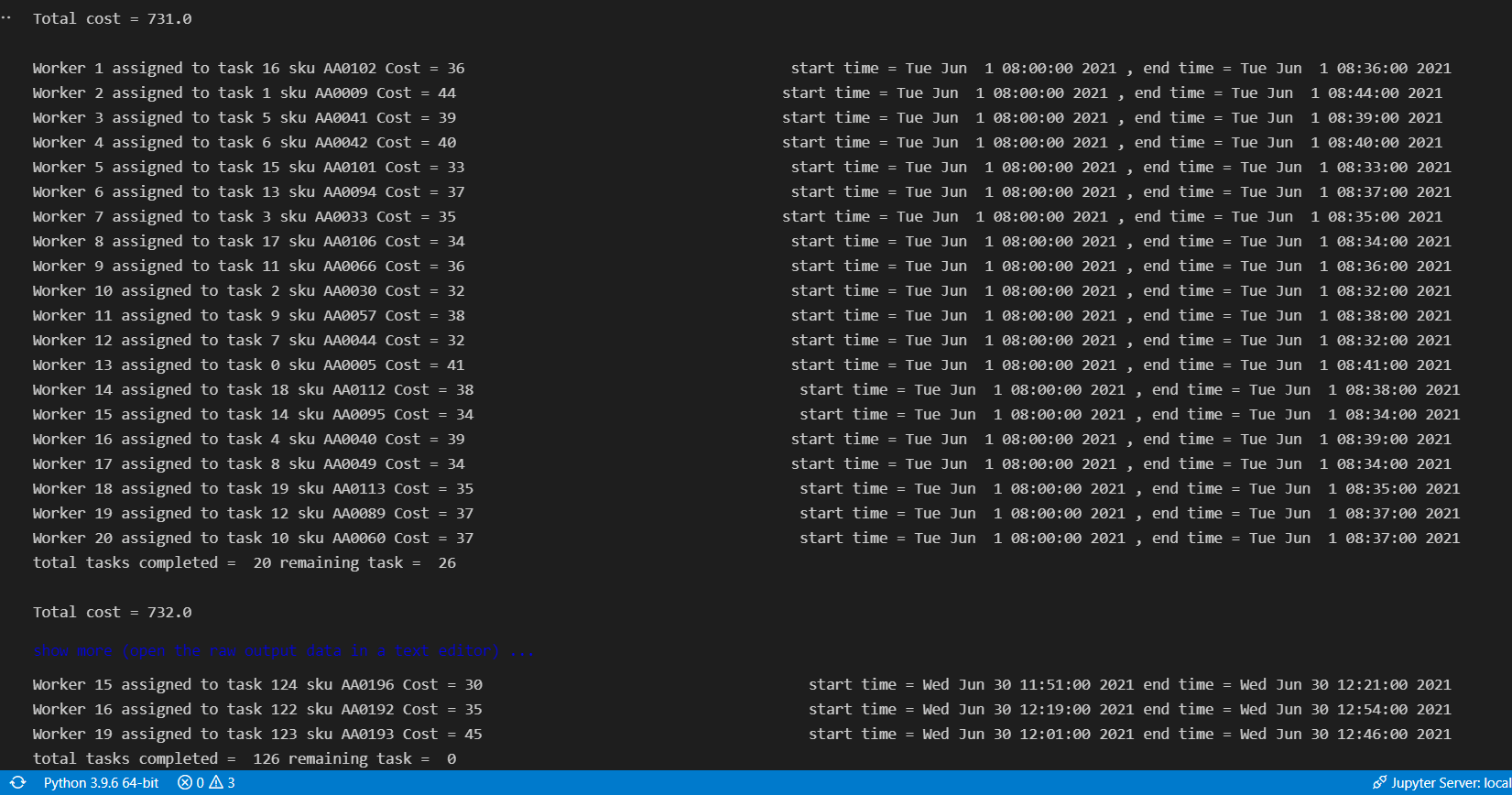
This is last loop to handle remaining jobs which is less than number of workers.



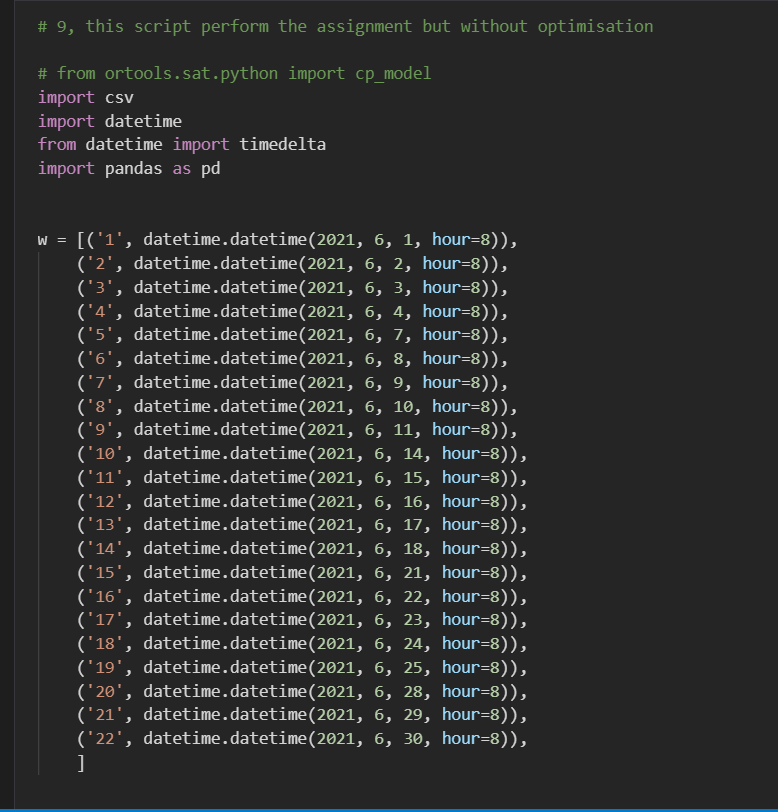
* Zip up all trackers into a list of tuples.
* Then, convert it into dataframe
* Then, transfer into table in PostgreSQL.



* Below is the output that display how each job is assigned to the workers, the time stamp for start and end of job.



1. **Job assignment but without OR (Operational Research) Optimization.**



* Below python code is the same as in previous section except that the OR algorithm has been removed.
* This means, the jobs are assigned to workers randomly without OR optimization.