**Infrastructure used:-**

The entire data pre-processing (Steps 1 -7 mentioned below) was done on a CPU node with 32 CPU and 176 GB RAM. Step1-4 are the most time consuming steps and the time to execute from step 5 onwards reduces drastically once the baseline bands data is created from flattening raster images.

For model development, Tesla V100 GPU is used which came with 6CPU and 16GB GPU RAM). The specifications are mentioned in the beginning of the two notebook in steps 8 and 9.

**Libraries used and external data usage:-**

All latest version of libraries were used till October 1 (saying so because sklearn & some other packages were updated post October 2nd). We have included a requirements.txt file & have also added a code path at the beginning of each notebook to download all the necessary packages. You can either run pip install -r requirements.txt or install the packages while running the notebooks

No external data was used in any stage apart from the competition dataset.

**Steps involved**

1. **Step 1:- Loading of dataset from ML Hub site**The code that will be used to run this step will be the following-

**“step1\_Data\_load\_sentinel2\_updated\_code.ipynb”**

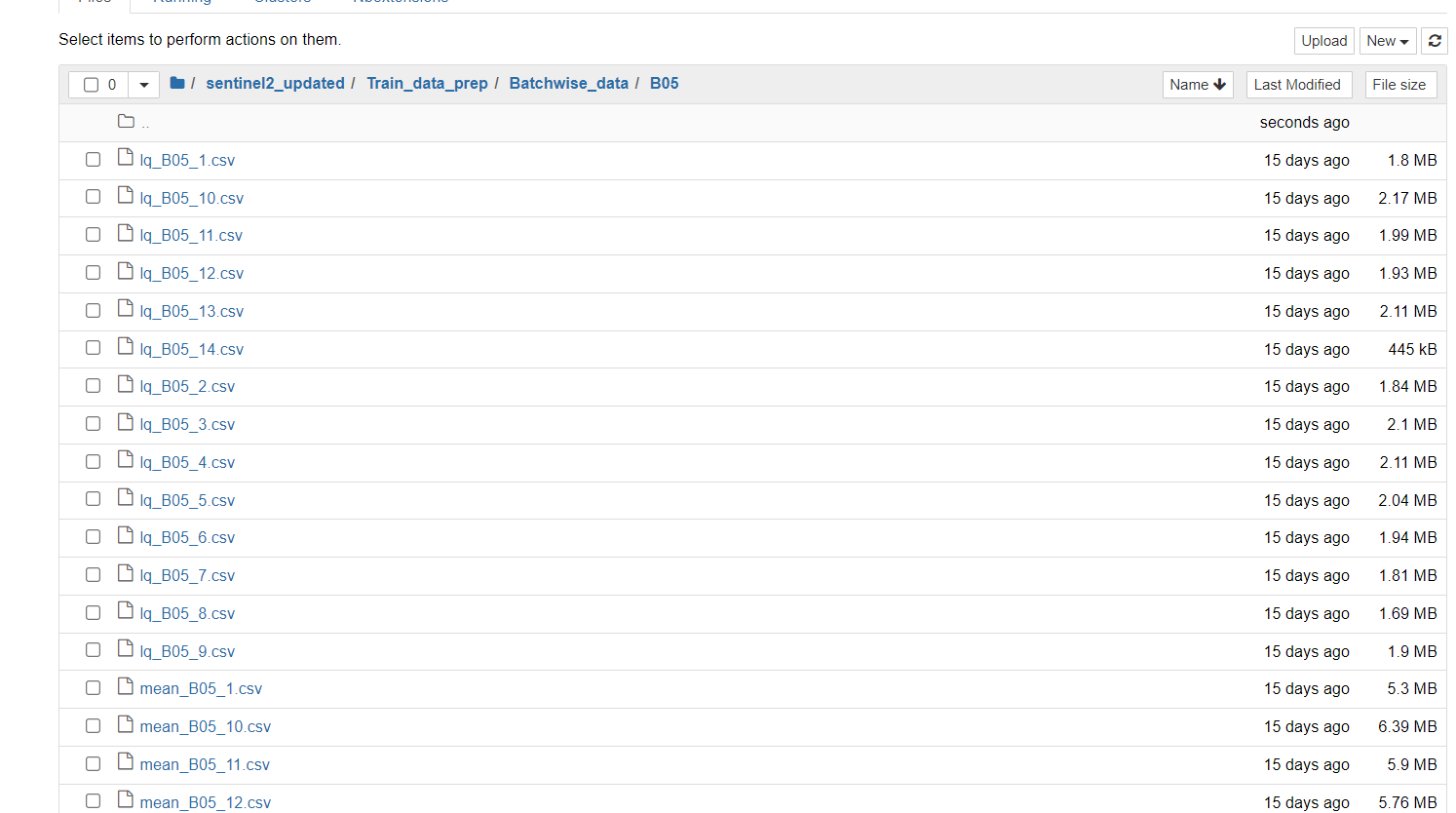
We ran the code on our cloud and the relevant folders were created in the same location where the notebook was present. The two output files be created out of this code and they will be “train\_data\_sentinel2.csv” and “test\_data\_sentinel2.csv”.  
  
Both these files created as an output from this code will be used to navigate through the tiff files for sentinel 2 images downloaded after running the code, and help us in creating proper train and test features.

1. **Step 2.1: Creation train dataset corresponding**  
   Using the train\_data\_sentinel2.csv, we create the train dataset for the model. We considered all 76 time points where data was observed. The aggregations suggested in the starter notebook after flattening the image file was mean but we also used median as it was giving better results as compared to mean.   
     
   There were 2650 tile-ids and just to make sure we do not run into memory issues, we processed it in batches of 200 each. We had 14 such batches. The batchwise results for each band was stored in the following way -   
   mean\_df.to\_csv(f'/root/sentinel2\_updated/Train\_data\_prep/Batchwise\_data/{band}/mean\_{band}\_{batchid}.csv',index=False)  
     
   Here, we created a folder named Batchwise\_data and inside that we created 12 folders corresponding to each band. In each of the folders, the outcome corresponding to each aggregation was getting saved. For example, mean\_B01\_4.csv will mean that mean aggregated labels corresponding to band B01 and 4th batch of tile-ids.

This will be the format in which the folders will have to be created to receive the batchwise outputs from the code.



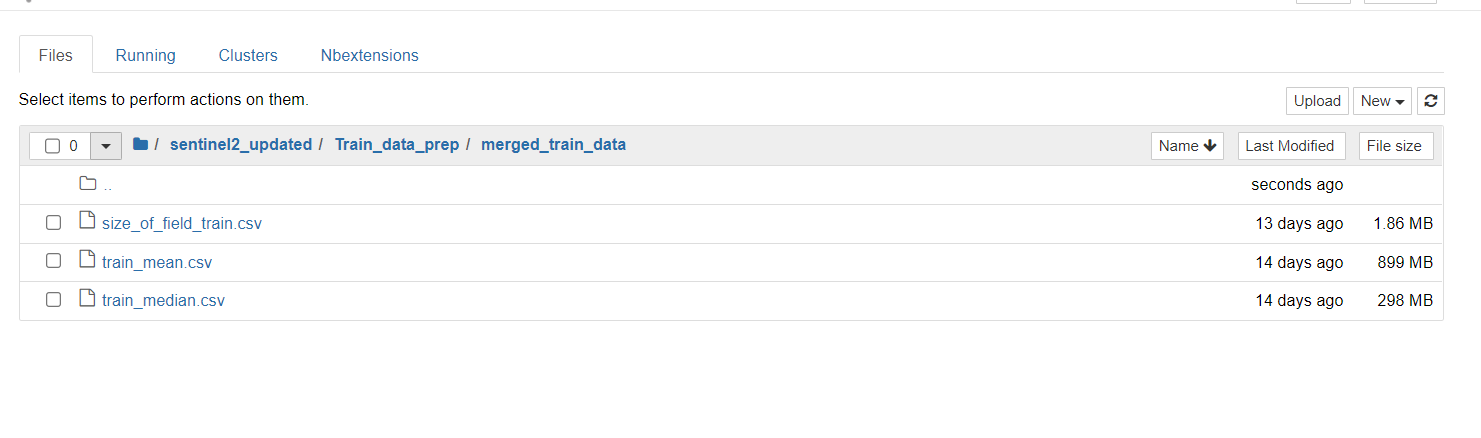
The is how for each band, the aggregates will look like after execution of all the 14 batches of train tile ids.

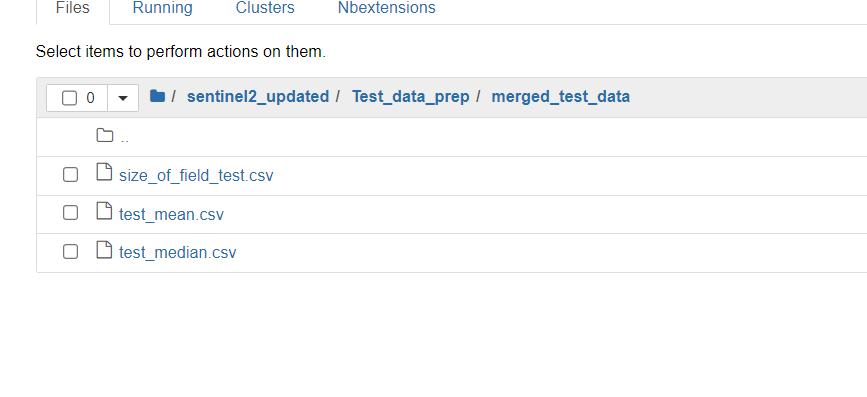


We considered lower quantiles and upper quantiles too but those can be ignored while execution of the code. The quantiles did give a little bit of boost but it came at the cost of increasing the number of features.  
  
The batchwise data will be combined together to a single data frame in the next step (step3).

1. **Step 2.2 Creation of test data**The code to run for this step is **step2.2\_Final\_test\_sentinel2.ipynb**  
   The test data was created in the same way as the train data. The tile ids for test were around 1137 and the number of batches were 6 where each batch had 200 tile ids.

Like in previous step(step2.1), we will merge all the batches in the test data in the next steps.

1. **Step 3.1: Merging all train data batches to one**  
   All the data will be merged to one over here and the code to run in this step is **step3.1\_Train\_data\_merging\_final.ipynb**Here all the data prepared in step 2.1 batchwise will be merged together to get a single data-frame. There will be two files created **train\_mean.csv and train\_median.csv.**The train\_mean.csv will have labels and features. The train\_median.csv will have only features.   
   The feature names are self-explanatory. For example, the feature name – “**B12\_month\_11\_day\_15”** means that it is a feature corresponding to Band 12 for month 11 corresponding to 15th day of the month (the observing date).   
     
   The merged data which will be used for model development is stored in the following location- ****  
   The files saved will be used to access create the model.
2. **Step 3.2: Merging all test data batches to one**All the test data will be merged to one over here and the code to run in this step is **“step3.2\_Final\_test\_merge.ipynb”**

The same steps will be followed as done in previous step. And the final test data will be saved in the location mentioned in the code.   


The test\_mean.csv and test\_median.csv will be used for model development.

1. **Step 4.1: Size of field creation (train data)**We incorporate the size of field information over here. We followed the introductory video (<https://www.youtube.com/watch?v=JphCl0jOXUA&t=1394s>) and was able to look at the time step 23:10 and understand that along with spectral information, we can also use the shape of the field to get information about the crop that could possibly grow in a region.

The code that needs to be run in this case is ‘step4.1\_Parallel\_processed\_train\_bands\_size\_of\_field.ipynb’ and this will output a file - size\_of\_field\_train.csv in the merged\_train\_data folder (check images in step 3.1)   
The size of a field is basically the 1000 multiplied by count of pixels divided by 256x256.

1. **Step 4.2: Size of field creation (test data)**We incorporate the size of field information over here. We followed the introductory video (<https://www.youtube.com/watch?v=JphCl0jOXUA&t=1394s>) and was able to look at the time step 23:10 and understand that along with spectral information, we can also use the shape of the field to get information about the crop that could possibly grow in a region.

The code that needs to be run in this case is ‘step4.2\_Parallel\_processed\_test\_bands\_size\_of\_field.ipynb’ and this will output a file - size\_of\_field\_train.csv in the merged\_train\_data folder (check images in step 3.1)   
The size of a field is basically the 1000 multiplied by count of pixels divided by 256x256.

1. **Step 4.3 Slope creation for both train and test data**We will have to run the code – “step4.3\_Slope calculation train test.ipynb”. Make sure the python file – “indices\_creation.py” is in the same location because it is used to create the indices.   
   The input to the code is all the output file from step2 to step 4.2 and the output will be the merged data with slope calculated for each band values (B01-B12). We calculate few indices over here as well that were selected after following the PhD thesis – “MACHINE LEARNING AND HIGH SPATIAL RESOLUTION MULTITEMPORAL SENTINEL-2 IMAGERY FOR CROP TYPE CLASSIFICATION” By MMAMOKOMA GRACE MAPONYAThe outcome of the code is “**train\_with\_slopes.csv**” and “**test\_with\_slopes.csv”**

**All important information that we could capture and what was mentioned in the video** (<https://www.youtube.com/watch?v=JphCl0jOXUA&t=1394s>)**, ends here only. We are able to capture the spectral bands, temporal phenology of crops and size of fields for which an estimate needs to be given.**   
  
No geospatial information was extracted at this stage. We could have stopped here only and ideally for any proper real world application of the problem, everyone should stop here.   
But there was a last week confirmation on usage of features derived from geo-spatial locations. **Even though, we pointed out several times that it was not allowed and it was clearly mentioned not to use in the video, we did not get a proper justification as to whether one needs to use it or not and hence, we went ahead with using it.**

1. **Step 5.1: Extracting latitude and longitudes from train data**The code to run for this stage is - **step5.1\_extract\_lat\_long\_train.ipynb**This code captures the X,Y coordinates for each pixel median and the corresponding latitude and longitude where the field is present. The latitude and longitudes for each field will help us in finding the distance between the fields in Step6 and also to know what type of crops are grown in neighbouring fields. (This was done purely as per the Zindi discussion mentioned in this link - <https://zindi.africa/competitions/radiant-earth-spot-the-crop-challenge/discussions/7105>)   
   The output will have 3 columns – field ids , latitude and longitude. The corresponding csv file **will be “train\_coordinates\_lat\_lon.csv”**
2. **Step 5.1: Extracting latitude and longitudes from test data**The code to run for this stage is - **step5.1\_extract\_lat\_long\_test.ipynb**This code captures the X,Y coordinates for each pixel median and the corresponding latitude and longitude where the field is present. The latitude and longitudes for each field will help us in finding the distance between the fields in Step6 and also to know what type of crops are grown in neighbouring fields. (This was done purely as per the Zindi discussion mentioned in this link - <https://zindi.africa/competitions/radiant-earth-spot-the-crop-challenge/discussions/7105>)   
   The output will have 3 columns – field ids , latitude and longitude. The corresponding csv file **will be “test\_coordinates\_lat\_lon.csv”**
3. **Step 6: Create geo-spatial features.**The code that needs to run is **“step6\_geospatial\_features.ipynb”.**This codes takes into account the latitude and longitudes captures in the previous step, creates 7 clusters of the data.

Further, for each cluster, it calculates the proportion of crop types in each cluster.   
The output from this step will be two files – pivottable.csv and seven\_cluster.csv  
Both the files have features generated from latitude and longitudes. The latitude and longitude are removed from the data and we have only the absolute features from them.

1. **Step7: Creating points within certain radius of a given fields**The code to run is “step7\_nearest\_points.ipynb”.   
   This code find out the percentage of points within a certain radius of a field. The points within a radius is determined by considering the haversine distance between the latitude ad longitude of the points.

The output to this code will be two files – “full\_nearest\_radius\_0.25.csv” and “full\_nearest\_radius\_0.4.csv”.

1. **Model development (Model 1) by Akash**The code to run for this step is **“step8\_model1\_akash.ipynb”**

The input files that will go into running it are as follows -   
a. train\_mean.csv (generated from step 3.1)

b. test\_mean.csv (generated from step 3.2)

c. train\_median.csv (generated from step 3.1)

d. test\_median.csv (generated from step 3.2)

e. seven\_cluster.csv ((generated from step 6)

f. full\_nearest\_radius\_0.25.csv (generated from step 7)

g. full\_nearest\_radius\_0.4.csv (generated from step 7)

h. train\_with\_slopes.csv (generated from step 4.3)

i. test\_with\_slopes.csv (generated from step 4.3)

j. pivotable.csv (generated from step 6)  
  
Supporting python file - indices\_creation.py (to create indices)

All the files must be in the same location as the codes. (Since we ran our codes on servers, we organized all files in one place after the end of step 7).

The outcome of the code will be “trial1\_sep\_akash.csv”  
The code is pretty self-explanatory and I guess it is trivial as well

1. **Model development (Model2) by Salim**

This code will include the same input files as the previous one. Just that its parameters of the models are different.

1. **Last step :- Blending code :**

Eventually the outcomes from both the models are taken and blended in the code- final\_blend.ipynb