

# Prize recipient solution documentation guide

**Congratulations!** You've gone up against dataheads from around the globe and emerged victorious! Laugh, dance, brush your shoulders off. You demonstrated serious skills, and helped make this world a better place in the process. Awesome job. Now you've finished in one of the top spots of the private leaderboard, which makes you eligible to receive a monetary prize. You're almost there.

In accordance with the official competition rules, the DrivenData terms of use, and applicable State and Federal law, we both have some due diligence to take care of before we can announce winners and disburse prizes.

You will receive a separate document to submit your [legal documentation](#) so that we can verify your legal identity. We will use this to verify your eligibility to participate and then review the specific laws and rules about giving out prizes based upon your nationality and our tax reporting obligations.

*Note: we are required by US Federal Law to withhold 30% of prize winnings for non-US individuals who do not already pay US taxes, unless exempted under an [applicable income tax treaty](#).*

This document details the steps required to submit your solution materials.

I. **Code submission and result reproducibility**

You package up and send us your code, documentation of dependencies, and any other assets you used. We review your package and make sure that it works and that we can fully reproduce the workflow from raw data to a submission comparable to your best submission. See the [Competition prize winner submission template](#) to get started.

II. **Basic information for winner announcement**

Provide your preferred information for use in announcing the winners of the competition.

III. **Model documentation and write-up**

You write up answers to our questionnaire, providing important context and documentation so that the beneficiary and the community get the most out of your work.

Please read this document carefully. Each section details exactly what is needed from you—the faster we can check all the boxes for our mutual responsibilities, the faster we can disburse your prize!

Thanks for your hard work, and congratulations for making it this far.

Best,  
The DrivenData Team

## I. Code submission and result reproducibility

You will need to submit a compressed archive of your code. You don't need to include the raw data we provided, but everything else should be included and organized. If the files are too large to be e-mailed, a Google Drive or Dropbox share (or other comparable method of transferring data) works.

**Note: please follow these instructions carefully.** The spirit and purpose of the competition (and the reason for offering prizes) is to give our beneficiary organizations the best possible solution *along with working code they can actually use*. In accordance with the competition rules, if we can't get your code working and reproduce your results with a reasonable effort, or if your entry is too disorganized to be practically usable, then your entry may be disqualified!

The overall concept is to **set up this archive as if it were a finished open source project**, with clear instructions, dependencies and requirements identified, and code structured logically with an obvious point of entry. Check out the competition prize winner [README template](#) to get started. We also have a [data science project template which may be helpful](#).

At a minimum, your solution must include **an extremely clear README** that details all of the steps necessary to get to your submission from a fresh system with no dependencies (e.g., a brand new Linux, Mac OS X, or Windows installation depending on what environment you choose to develop under) and no other data aside from the raw data you downloaded from us.

This will probably entail the following:

- Necessary tools and requirements (e.g. “You must install Word2Vec 0.1c” or “Install the required Python packages in `requirements.txt`”).
  - **All requirements should be clearly documented**, for instance in either a `requirements.txt` file with versions specified or `environment.yml` file.
- The series of commands, in order, that would get a reasonably experienced and savvy user from your code to a finished submission.
  - **Ideally, you will have a main point of entry to your code** such as an executable script that runs all steps of the pipeline in a deterministic fashion. A well-constructed Jupyter notebook or R script meets this standard.
  - **The next best thing is a list of specific, manual steps** outlining what to do. For example, “First, open Word2Vec and set these parameters. [...] Take the output file and run the script `src/make_preds.R` with the following parameters [...]”. (*The limitations of this approach should be clear to any experienced data scientist!*)
- **Make sure to provide access to all trained model weights necessary to generate predictions from new data samples** without needing to retrain your model from scratch. Note that model weights can be contained in your archive or shared via a cloud storage service. The solution should provide clear instructions to perform inference on a new data point, whether or not it is included in the test set.
- Any other instructions necessary to end up with your winning submission file (or comparable — we understand that certain parts of model fitting are stochastic and won't result in exactly the same parameters every time).

## II. Basic information for winner announcement.

Please provide your preferred information for use in announcing the winners of the competition.

- Name (first and last name or first name and last initial): Raphael Kiminya
- Hometown: Meru, Kenya
- A recent picture of yourself or digital avatar (feel free to attach separately): *attached*

## III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

In my other life, I worked various roles as part of a data analytics team - as a systems admin, DBA, data warehouse architect and reports developer. I now work as a freelance data scientist. Over the past 2 years I've been studying machine learning through data competitions, solving all sorts of fascinating challenges.

2. What motivated you to compete in this challenge?

Satellites can 'see' the air that we breathe? It's ingenious. And I love a good challenge. Whenever I come across such a challenge, I'm reminded of the capacity of machine learning to alleviate the broader issues facing our society today. We can build solutions that affect millions of people, hopefully change their lives for the better.

3. High level summary of your approach: what did you do and why?

My approach was almost entirely data-centric. Initially I experimented with the original competition dataset and a variety of models. I quickly hit a plateau where local CV and LB scores didn't improve.

I shifted my focus to the external datasets, and settled for a simple gradient boosting model to speed up my experiments. I used the MAIAC satellite data and GFS forecasts for the PM2.5 model. For the NO2 model, I used OMI and TROPOMI satellite data, GFS forecasts and GEOS-CF NO2 hindcasts.

Choice of GFS variables was based on relevant literature and availability of data throughout the training and testing period i.e. Jan 2017 to Aug 2021. Generally, I selected parameters affecting or similar to air humidity, soil temperature, soil humidity, air temperature, wind velocity, wind direction and rainfall/ precipitation.

GFS forecasts from upto 3 days preceding the date of interest seemed to improve the predictions, so I created multiple datasets with different lookback periods. I also realized that separate models for each location performed better than a single model. Since the test set

includes dataset from the past, I decided to use a k-fold (without shuffle) cross validation strategy instead of time-based splits. The final solution is an average ensemble of 45 models (3 datasets x 5 folds x 3 locations) for the PM2.5 model and 30 models (2 datasets x 5 folds x 3 locations) for the NO2 model.

4. Do you have any useful charts, graphs, or visualizations from the process?

No.

5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

```
##impute missing MAIAC data by expanding the region of interest with  
small increments. The same idea is applied for TROPOMI data  
bounds=row.geometry.bounds  
for OFF in [i*0.01 for i in range(1,5)]:  
    #increment area with a small offset  
    ixs=np.where((lon>=bounds[0]-OFF) & (lon<=bounds[2]+OFF) &  
(lat>=bounds[1]-OFF) & (lat<=bounds[3]+OFF))  
    d_mean=np.nanmean(data[ixs])  
    d_median=np.nanmedian(data[ixs])  
    if not np.isnan(d_mean):  
        break
```

6. Please provide the machine specs and time you used to run your model.

- CPU (model): Intel Xeon @ 2.30GHz, 4vCPUs
- GPU (model or N/A): N/A
- Memory (GB): 32
- OS: Ubuntu 21.04
- Train duration: 2 hours (see *runtime.html* for details)
- Inference duration: 10 minutes

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

Downloading GFS data is the biggest bottleneck in the pipeline. The NCAR data archive server (<https://rda.ucar.edu>) has a limit of 10 concurrent requests, and seems to process only 2 requests at a time while the rest are queued. Try using separate NCAR accounts if running the NO2 and PM2.5 preprocessing scripts concurrently.

It's also worth noting that requests for GFS data sometimes fail for no good reason. Retrying the failed request usually works, so my current solution is a loop which could potentially run forever :(.

Refer to *src/data/extract\_gfs.py*

```
while True:  
    REQUESTS,PARAM_PATHS = generate_requests()  
    results = pqdm(REQUESTS, download_data,  
n_jobs=MAX_CONCURRENT_REQUESTS,argument_type=None)  
    completed = [res for res in results if res['success']==True]  
    if len(completed)>0:
```

```
print('Completed requests: ',completed)
failed = [res for res in results if res['success']==False]
if len(failed)>0:
    print('The following requests failed. Retrying...')
    print(failed)
else:
    break
```

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

No

9. How did you evaluate the performance of the model other than the provided metric, if at all?

I optimized RMSE for the PM2.5 model and Huber loss for the NO2 model. I tracked the best R2 validation score for both models.

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

RNNs/ Transformers with a rolling time window, but sequential data didn't seem to improve forecasts. I also briefly experimented with segmentation models but with the low spatial resolution and missing data it was a long shot to begin with.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

Feature selection, tuning - I ran out of time before I could identify the most useful features and properly tune the model parameters before my last submission. With around 300 features it's likely that the models could be improved or at least simplified with comprehensive feature selection.

Higher resolution data - I didn't get a chance to explore the ECMWF dataset, which has a higher spatial resolution than GFS. Integrating the dataset in the model could improve forecasts.

Redundancy - Sporadic outages of the [GEOS-CF data server](#) got me thinking about the resilience of this solution in a production environment due to its dependence on external datasets. I would consider building different models that work with different combinations of the available datasets and at different scales e.g. using 3,6,12,24 hour forecasts.

