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.**

Hi, I am Nikhil Kumar Mishra, currently working as a Data Scientist for Okcredit, a startup in Bangalore.

2. **What motivated you to compete in this challenge?**

I loved participating in Data Science competitions, since the time I started to learn Data Science, and am always on the lookout for something interesting. I am also fortunate to have finished top 3 in many of them. I think I had never worked seriously on signal data before, so this challenge was a perfect way to hone my skills. Also a competition hosted by NASA on drivendata would surely bring a lot of amazing participants, so I could not pass this opportunity.

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

- I converted the multilabel problem into a binary classification problem. Given a combination of a sample_features + compound, tell if the compound is present in the sample. For a given sample, features were the same for each compound, except the feature telling which compound we were predicting.
- I used lightgbm k fold ensemble model to get the initial predictions, then fed these predictions along with top 5k features to a 31 fold ensemble, catboost model (which acted like a meta model). 2 stage modelling helped quite a lot in this competition, because the presence of one compound could help in the determination of others. Number of folds were high because I felt the data was quite small, so I wanted to have most of the samples in training.
- I will describe the thought process behind feature-engineering now, as it was very crucial in this competition.

- Almost all my feature engineering was done per m/z instead of per sample_id.

- I pre-sorted each sample_id's data mass spectra by abundance in descending order, to help with feature creation, since signal characteristics near the peak were strong indicators of presence of compounds.

- I saw was the signals were quite noisy specially from sam test-bed, so I tried to smoothen the signals and convert them to fix length per m/z, per sample_id, so I could have fixed length features for each sample_id, this alone gave me a lot of boost in my score. After some trial and error I stuck to 25 length sequence, and created features for abundance and temperature using that, both with sorted by abundance and by their original time order

- I followed the following steps, to get aggregation features per m/z, sorting the data in abundance descending order. (Please refer to the code too, as it might be tricky to read)

    1. Firstly, create a list of values of k which basically tells to get top k values for each sample. Since all sample_ids had different length mass spectra, I normalized the length of each of the spectra, and then took top k% of the samples. k had values starting from 0.001 (top 0.1% of the values) to 0.2 (top 20%).
    2. Create features by aggregating abundance (normalized frequency, mean, standard_deviation) per m/z and temp_bin (temperature was binned per sample_id into 5 buckets)
    3. Create features by aggregating temperature and abundance per m/z.
    4. For features created in 2 and 3, I took the ratio of the features generated by current k, to the features generated by previous k.

- I also created sample level features, like max_temperature at highest abundance, range of temperature and top 10 abundance values, etc.

- Finally I created features by aggregating temp on m/z and abundance_bin (I binned abundance of each sample_id into 8 equal parts) for all the values (not top k values).

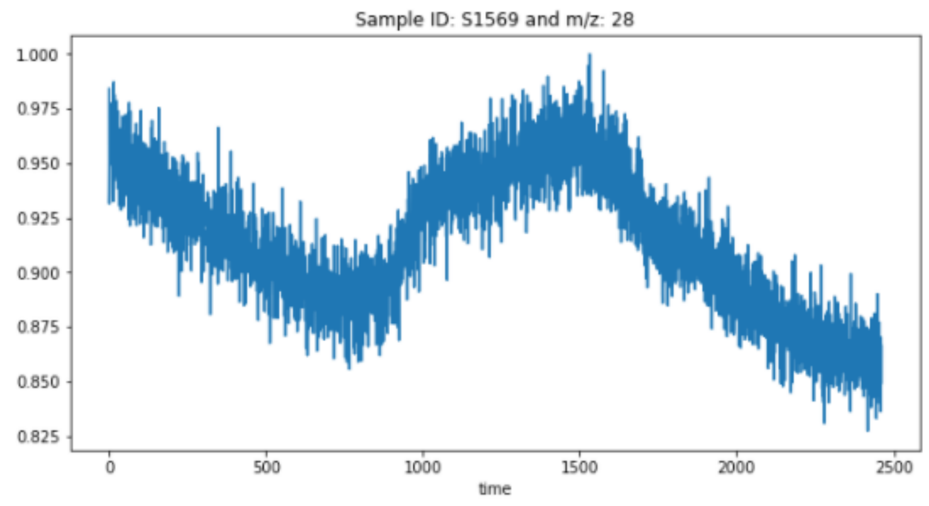
- My general approach to feature engineering and modelling is to add a feature and keep it only if it helps in the model evaluation, and remove it if it doesn't improve the evaluation, feature generation and evaluation of the model must be done together.

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

One helpful graph I felt was resampling the signal abundance values, an example of this is shown below. Y-axis is the **abundance.** (Refer to nbs/eda.ipynb for this example)

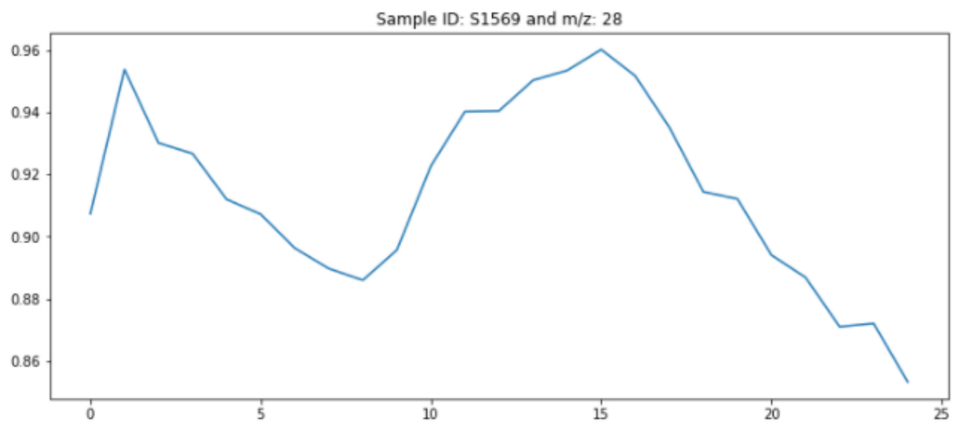
Before Sampling:

The signal is quite noisy and is represented by 2500 numbers for a single m/z



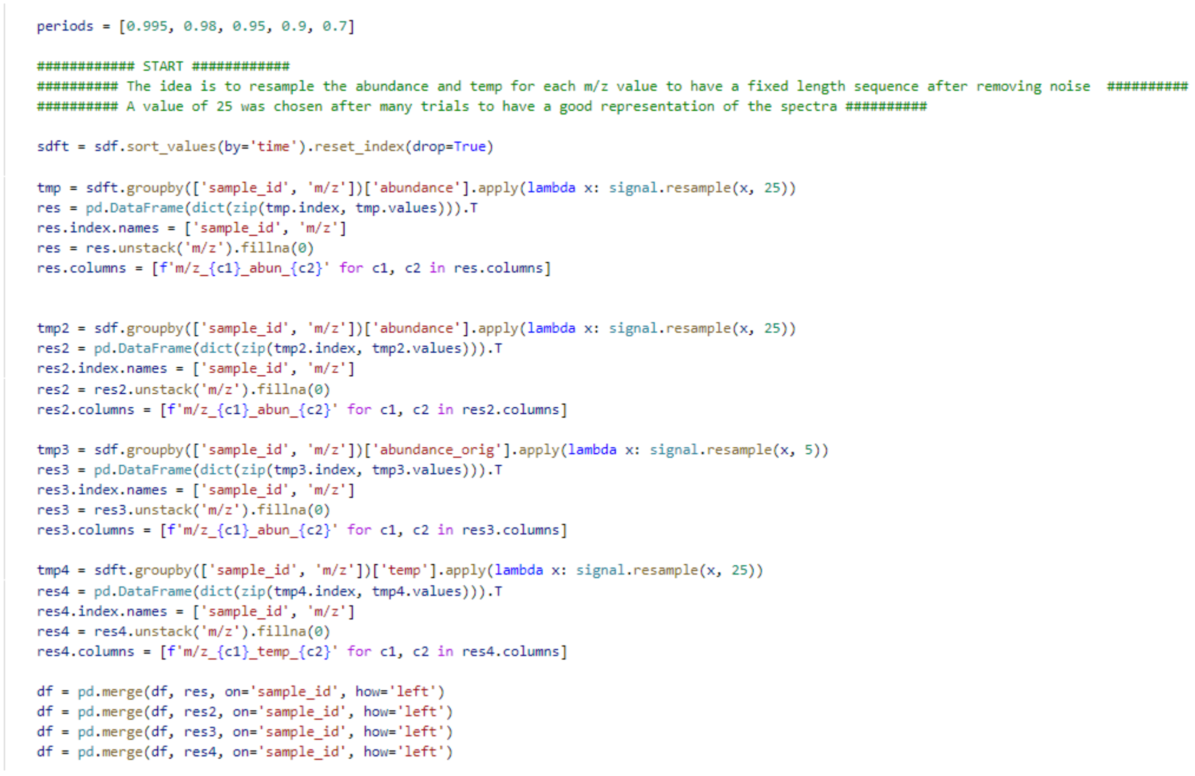
After Sampling

The signal is represented by only 25 numbers and is less noisy. These 25 numbers can be easily used as features directly to the model.



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

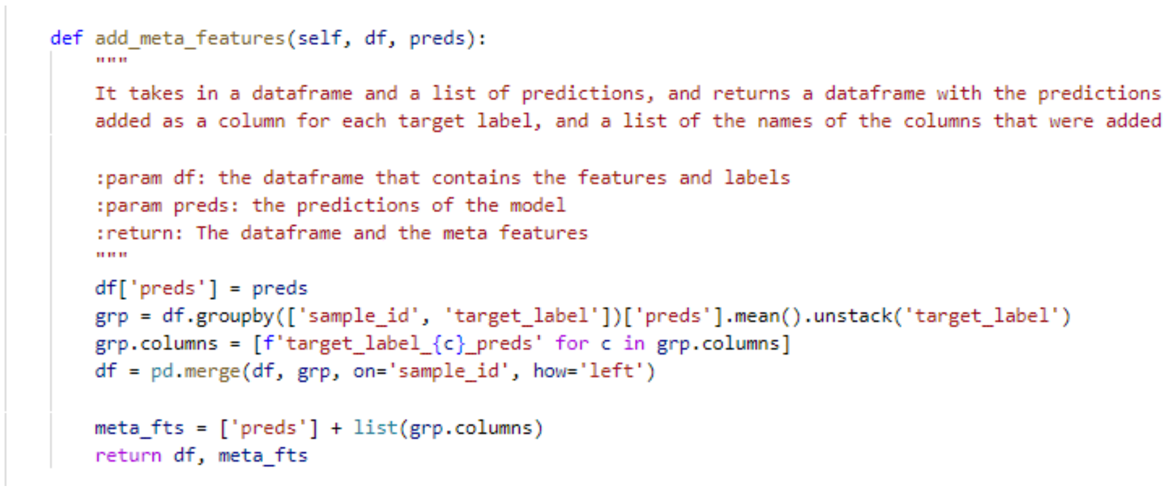
A. Code for Signal Resampling



```
periods = [0.995, 0.98, 0.95, 0.9, 0.7]

############ START ############
########## The idea is to resample the abundance and temp for each m/z value to have a fixed length sequence after removing noise  ##########
########## A value of 25 was chosen after many trials to have a good representation of the spectra ##########

sdft = sdf.sort_values(by='time').reset_index(drop=True)

tmp = sdft.groupby(['sample_id', 'm/z'])['abundance'].apply(lambda x: signal.resample(x, 25))
res = pd.DataFrame(dict(zip(tmp.index, tmp.values))).T
res.index.names = ['sample_id', 'm/z']
res = res.unstack('m/z').fillna(0)
res.columns = [f'm/z_{c1}_abun_{c2}' for c1, c2 in res.columns]


tmp2 = sdf.groupby(['sample_id', 'm/z'])['abundance'].apply(lambda x: signal.resample(x, 25))
res2 = pd.DataFrame(dict(zip(tmp2.index, tmp2.values))).T
res2.index.names = ['sample_id', 'm/z']
res2 = res2.unstack('m/z').fillna(0)
res2.columns = [f'm/z_{c1}_abun_{c2}' for c1, c2 in res2.columns]

tmp3 = sdf.groupby(['sample_id', 'm/z'])['abundance_orig'].apply(lambda x: signal.resample(x, 5))
res3 = pd.DataFrame(dict(zip(tmp3.index, tmp3.values))).T
res3.index.names = ['sample_id', 'm/z']
res3 = res3.unstack('m/z').fillna(0)
res3.columns = [f'm/z_{c1}_abun_{c2}' for c1, c2 in res3.columns]

tmp4 = sdft.groupby(['sample_id', 'm/z'])['temp'].apply(lambda x: signal.resample(x, 25))
res4 = pd.DataFrame(dict(zip(tmp4.index, tmp4.values))).T
res4.index.names = ['sample_id', 'm/z']
res4 = res4.unstack('m/z').fillna(0)
res4.columns = [f'm/z_{c1}_temp_{c2}' for c1, c2 in res4.columns]

df = pd.merge(df, res, on='sample_id', how='left')
df = pd.merge(df, res2, on='sample_id', how='left')
df = pd.merge(df, res3, on='sample_id', how='left')
df = pd.merge(df, res4, on='sample_id', how='left')
```

The above piece of code resamples the values per sample_id and abundance to a fixed length, which are then directly used as a features to the model

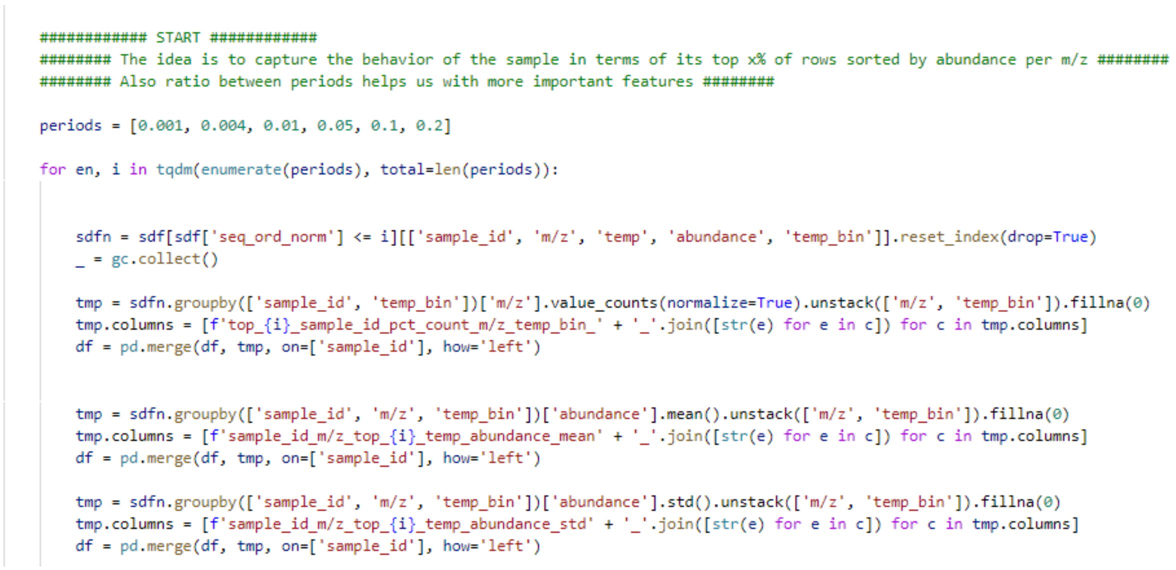
B. Code for Adding Meta Features



```python
def add_meta_features(self, df, preds):
    """
    It takes in a dataframe and a list of predictions, and returns a dataframe with the predictions
    added as a column for each target label, and a list of the names of the columns that were added

    :param df: the dataframe that contains the features and labels
    :param preds: the predictions of the model
    :return: The dataframe and the meta features
    """
    df['preds'] = preds
    grp = df.groupby(['sample_id', 'target_label'])['preds'].mean().unstack('target_label')
    grp.columns = [f'target_label_{c}_preds' for c in grp.columns]
    df = pd.merge(df, grp, on='sample_id', how='left')

    meta_fts = ['preds'] + list(grp.columns)
    return df, meta_fts
```

The above piece of code adds meta features (predictions of each kind of compound as features to the meta-model)

C. Feature Engineering of Top X% of the rows



```python
############# START #############
######## The idea is to capture the behavior of the sample in terms of its top x% of rows sorted by abundance per m/z ########
######## Also ratio between periods helps us with more important features ########

periods = [0.001, 0.004, 0.01, 0.05, 0.1, 0.2]

for en, i in tqdm(enumerate(periods), total=len(periods)):


    sdfn = sdf[sdf['seq_ord_norm'] <= i][['sample_id', 'm/z', 'temp', 'abundance', 'temp_bin']].reset_index(drop=True)
    _ = gc.collect()

    tmp = sdfn.groupby(['sample_id', 'temp_bin'])['m/z'].value_counts(normalize=True).unstack(['m/z', 'temp_bin']).fillna(0)
    tmp.columns = [f'top_{i}_sample_id_pct_count_m/z_temp_bin_' + '_'.join([str(e) for e in c]) for c in tmp.columns]
    df = pd.merge(df, tmp, on=['sample_id'], how='left')


    tmp = sdfn.groupby(['sample_id', 'm/z', 'temp_bin'])['abundance'].mean().unstack(['m/z', 'temp_bin']).fillna(0)
    tmp.columns = [f'sample_id_m/z_top_{i}_temp_abundance_mean' + '_'.join([str(e) for e in c]) for c in tmp.columns]
    df = pd.merge(df, tmp, on=['sample_id'], how='left')

    tmp = sdfn.groupby(['sample_id', 'm/z', 'temp_bin'])['abundance'].std().unstack(['m/z', 'temp_bin']).fillna(0)
    tmp.columns = [f'sample_id_m/z_top_{i}_temp_abundance_std' + '_'.join([str(e) for e in c]) for c in tmp.columns]
    df = pd.merge(df, tmp, on=['sample_id'], how='left')
```

The above code shows how features were added using top x% of the rows (sorted by abundance in descending order) for each m/z and temp_bin of a sample_id

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

- CPU (model): Intel 16 cores
- GPU (model or N/A): N/A
- Memory (GB): 64 GB RAM
- OS: Ubuntu
- Train duration: 2 hours
- Inference duration: Around 6 minutes for 804 (train + val samples), can be reduced if we precompute the features.

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.)?**

   No

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 performance of the model other than the provided metric, if at all?**

   My goal was to use the validation data as test data and only use train_labels to train the models, till the end of the competition. Once I saw my validation metrics improved significantly, for the competition metric, I reran the pipeline using validation and train data to train the model, and then predicted according to the competition requirements.

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

    I think with more samples deep learning could have beat my lightgbm + catboost feature engineering approach. I had tried CNNs and transformers, but they obviously couldn't come par with the winning methods, due to the small data. I would love to rerun my deep learning approaches with a large data if possible.

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?**

    I think instead of just predicting a certain fixed compounds in the sample, if we can predict more compounds in the sample it would definitely improve the score (the reason why meta-modelling was working quite well).

    I think I created a lot of features in the end, to give me a winning edge in the competition, but with some more effort I could easily reduce the number of features by 10x or 50x.