

Kaggle PLAsTiCC

The Competition, Dataset and
Top Leaderboard Strategies

PLUS

Code walkthrough of the B. Trotta Submission

Chris Winsor

6/10/2020

https://github.com/cwinsor/kaggle_plasticc

www.cwinsor.us

Agenda:

- Overview of Kaggle PLAsTiCC Competition and LSST
- Review Top Leaderboard (Approaches Taken / Common Themes)
- Detailed Code Walkthrough of B. Trotta submission

PLAsTiCC, LSST and Kaggle



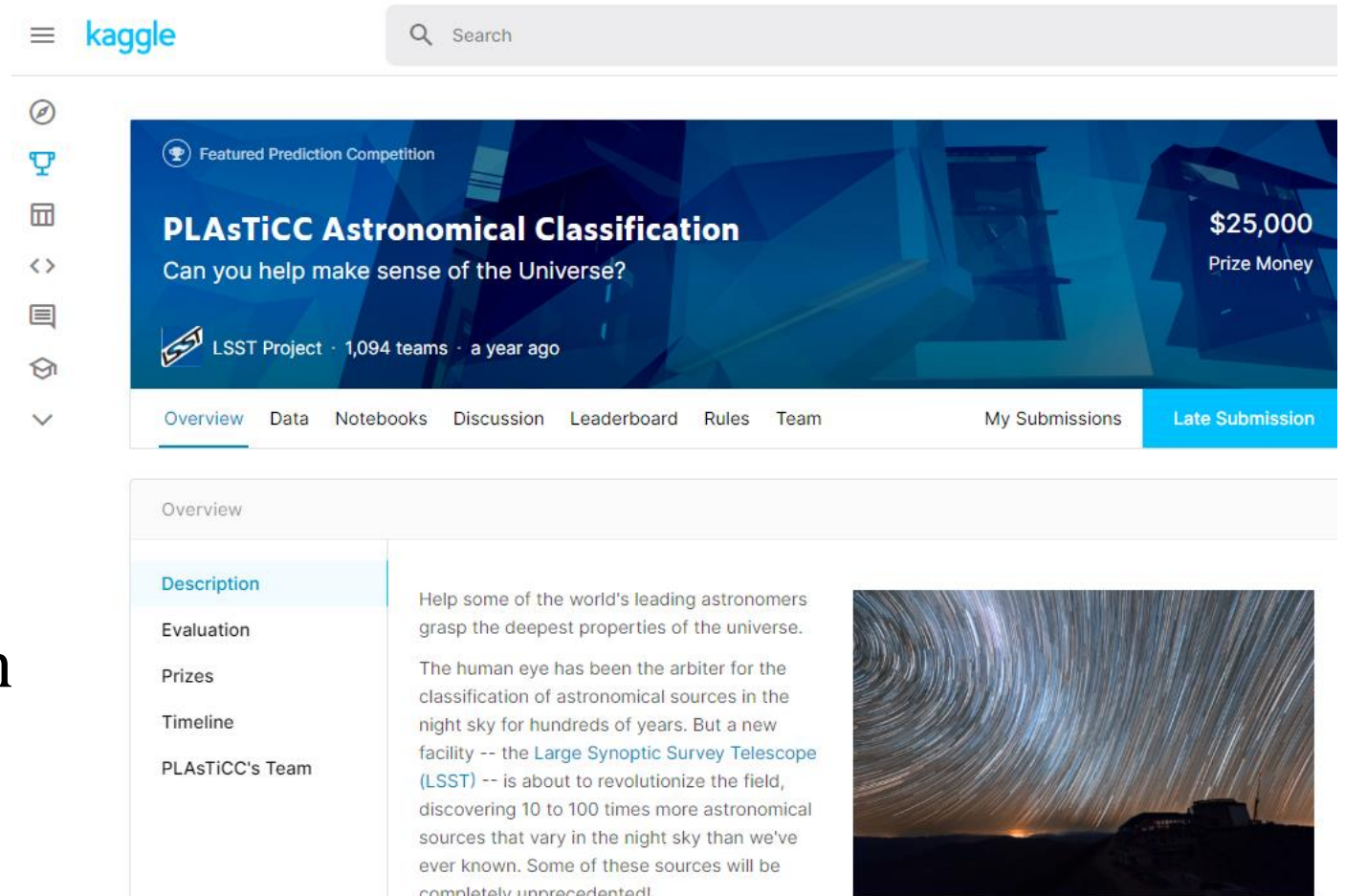
- **LSST**: “Large Synoptic Survey Telescope” [1]
A new telescope focusing on detecting and studying “Transients” - expected completion in 2023
- **PLAsTiCC**: Photometric LSST Astronomical Time Series Classification Challenge” [2]
The Kaggle competition to classify star timeseries data anticipated from LSST
- **Transients**: stars that are actively changing such as:
 - Supernova that explodes over a 100 day period
 - Pulsar that flashes once every 12 hours
 - Lensing Event (a planet goes in front of a star) that occurs... occasionally!

(1) <https://www.lsst.org/>

(2) <https://arxiv.org/abs/1810.00001>

Goals:

- Goal of LSST is to detect transients and notify astronomers
- Goal of PLAsTiCC Kaggle competition is to design classifier to find transients in data stream
- Competition held in 2018 in preparation for first light of LSST



The screenshot shows the Kaggle interface for the PLAsTiCC Astronomical Classification competition. The header includes the Kaggle logo and a search bar. The competition banner features the title "PLAsTiCC Astronomical Classification" with the subtitle "Can you help make sense of the Universe?" and a prize money of "\$25,000". It also mentions "LSST Project · 1,094 teams · a year ago". Below the banner are tabs for "Overview", "Data", "Notebooks", "Discussion", "Leaderboard", "Rules", "Team", "My Submissions", and "Late Submission". The "Overview" tab is selected, showing a "Description" section with text about the competition's goal and a star trail image.

Featured Prediction Competition

PLAsTiCC Astronomical Classification


Can you help make sense of the Universe?

LSST Project · 1,094 teams · a year ago

\$25,000 Prize Money

Overview Data Notebooks Discussion Leaderboard Rules Team My Submissions Late Submission

Overview

Description	Help some of the world's leading astronomers grasp the deepest properties of the universe.	
Evaluation	The human eye has been the arbiter for the classification of astronomical sources in the night sky for hundreds of years. But a new facility -- the Large Synoptic Survey Telescope (LSST) -- is about to revolutionize the field, discovering 10 to 100 times more astronomical sources that vary in the night sky than we've ever known. Some of these sources will be completely unprecedented!	
Prizes		
Timeline		
PLAsTiCC's Team		

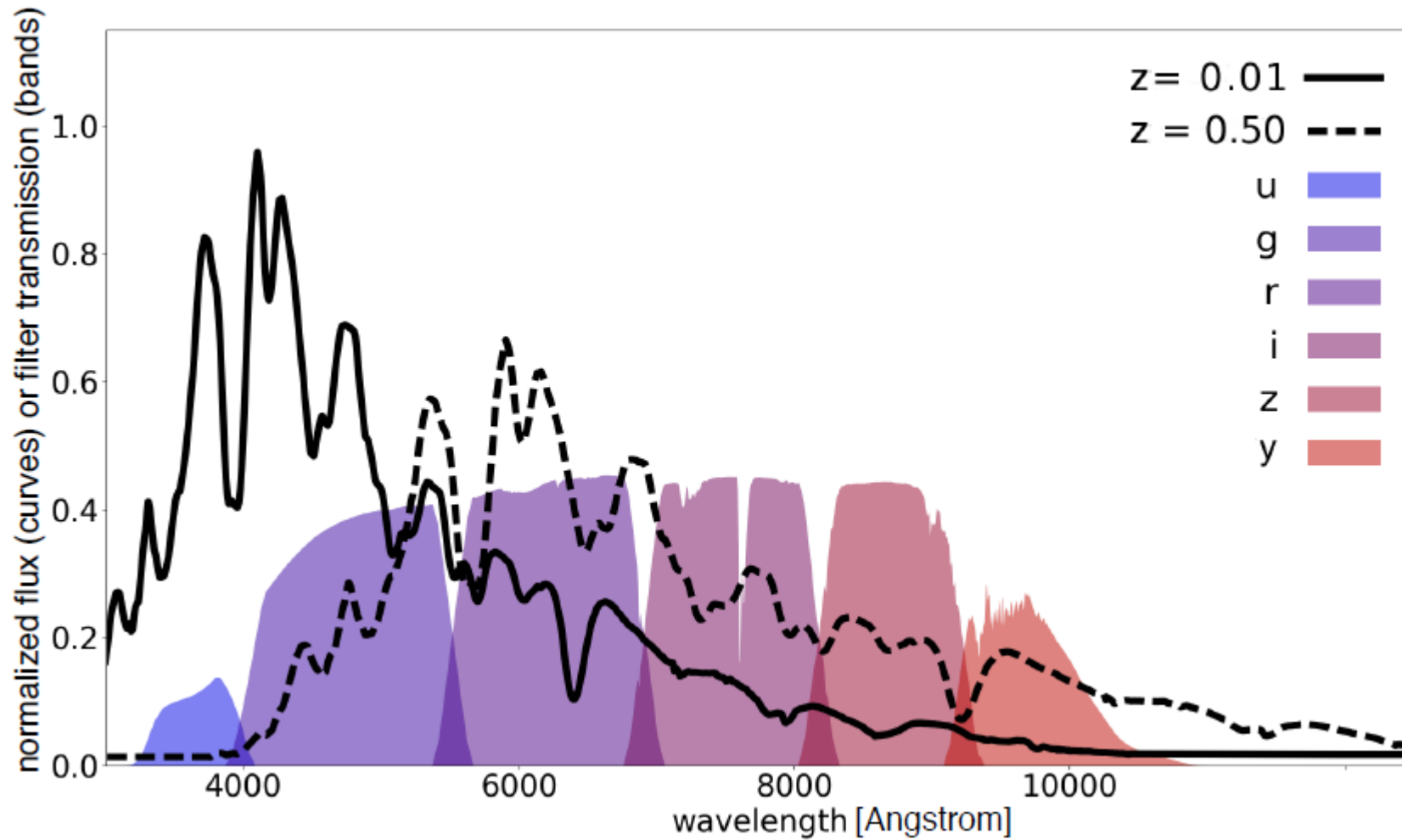
LSST Data Size/Scale

Camera Density	3.2B pixels
Data Rate:	20TB/night
Objects in Database:	37 billion
Class Target:	15 classes
Detection/Notification Rate:	10M/night
Latency Goal (observation to notification):	60 minutes

Additional challenges:

- Observations are aperiodic due to season (Earth's axis) weather and telescope schedule
- Observations are by passband, one band captured per observation.

Passbands



Template (differential sampling)

A “template” technique is used to measure intensity

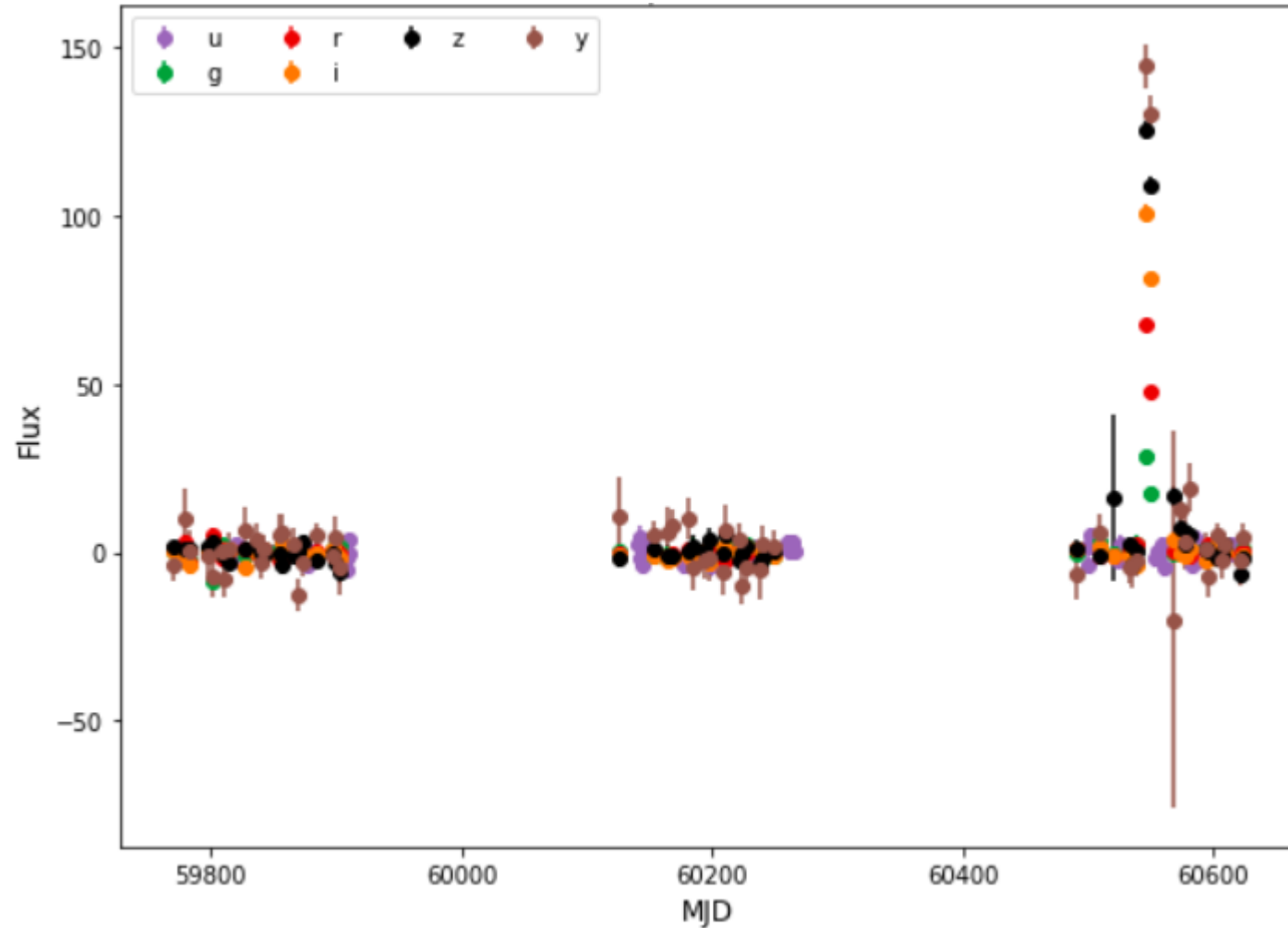
Expressed as “flux” - intensity relative to template

This allows detecting very small changes

- A reference image (template) has been previously established for each star
- A new image (sample) is captured
- A simple difference is computed: $\text{flux} = \text{sample} - \text{template}$
- Flux can be negative
- Flux_error is computed (no details here)

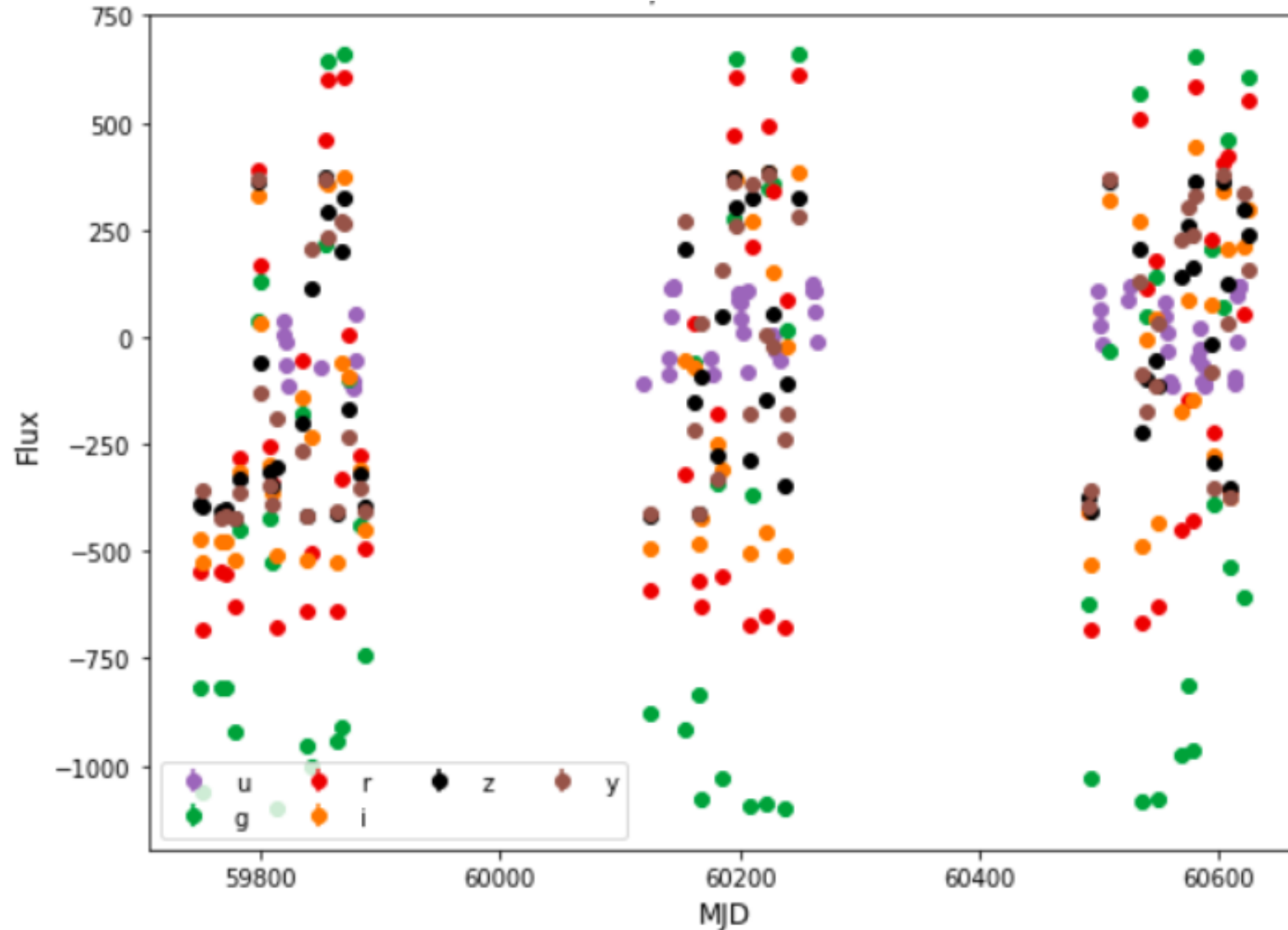
Impact of Schedule

object_id 3910



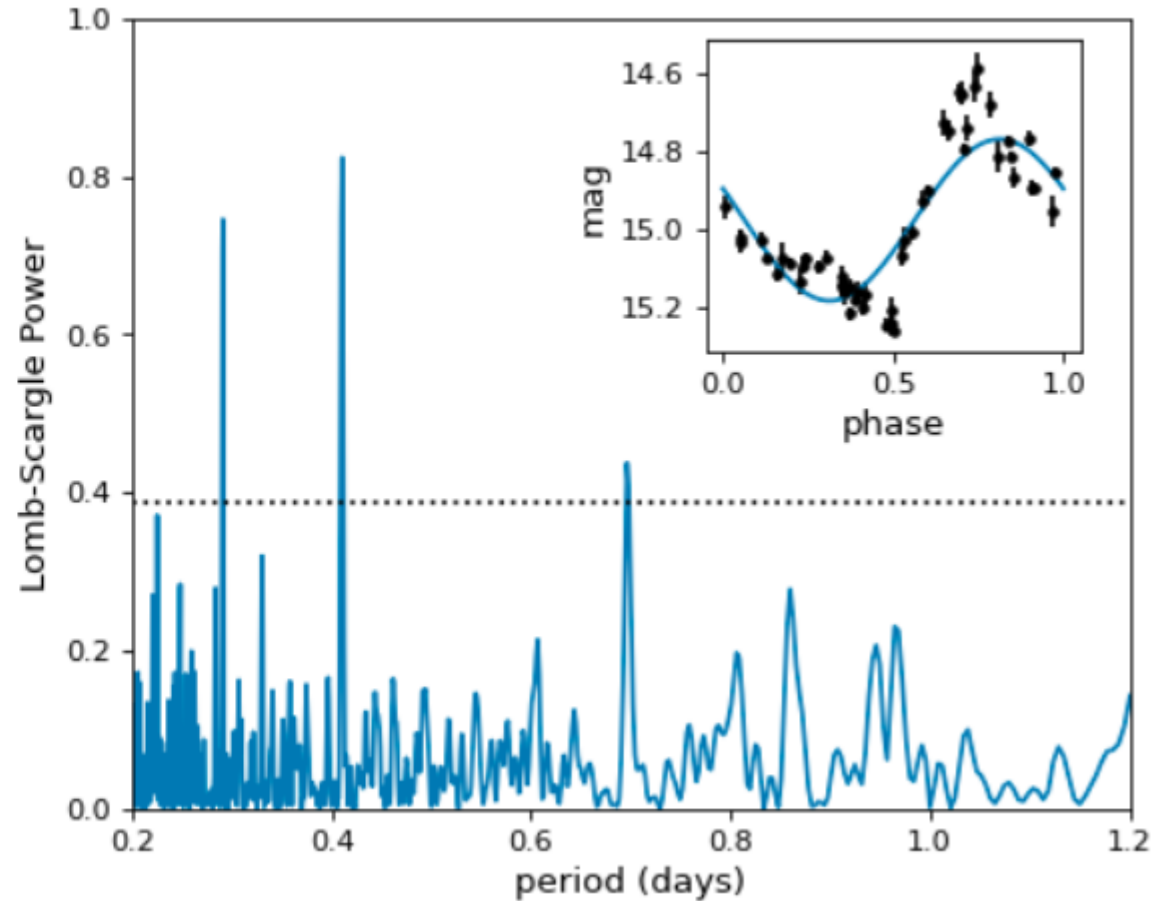
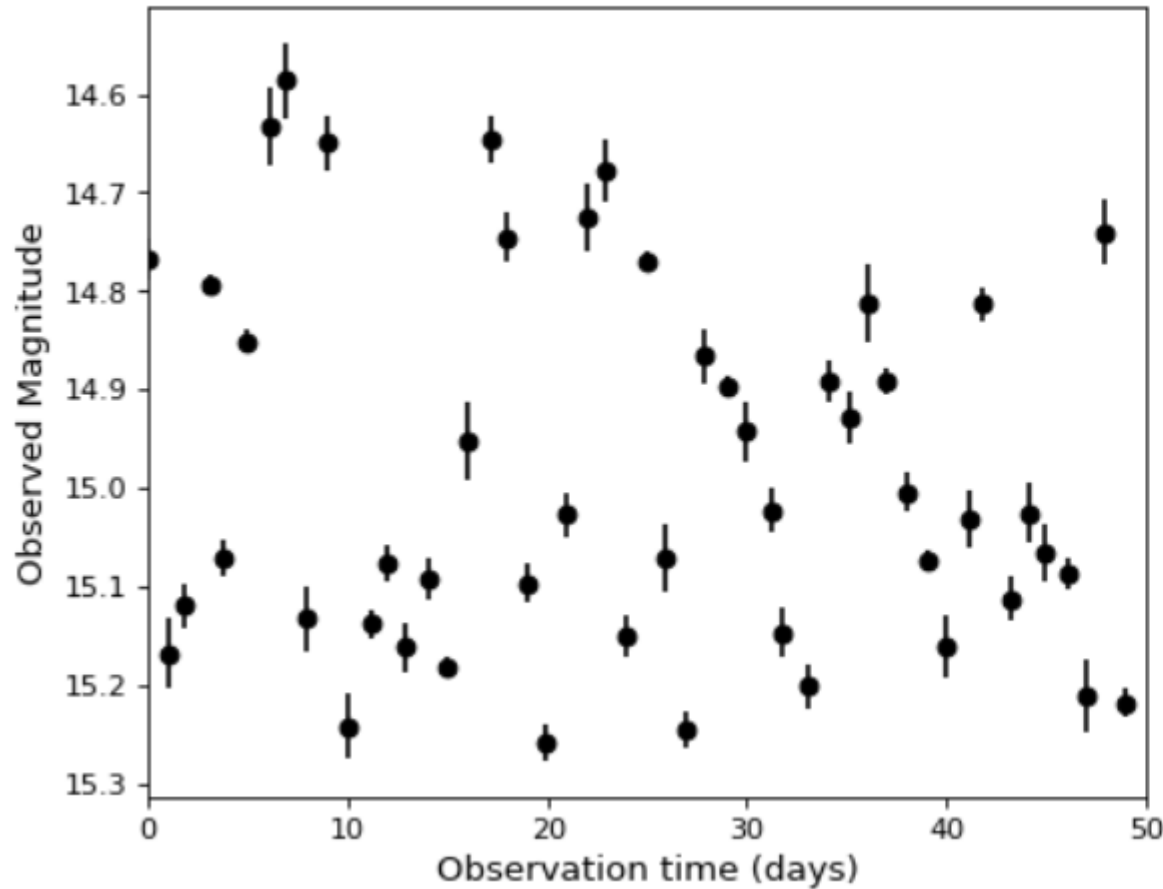
Impact of Sample Rate < Signal Rate

object_id 615



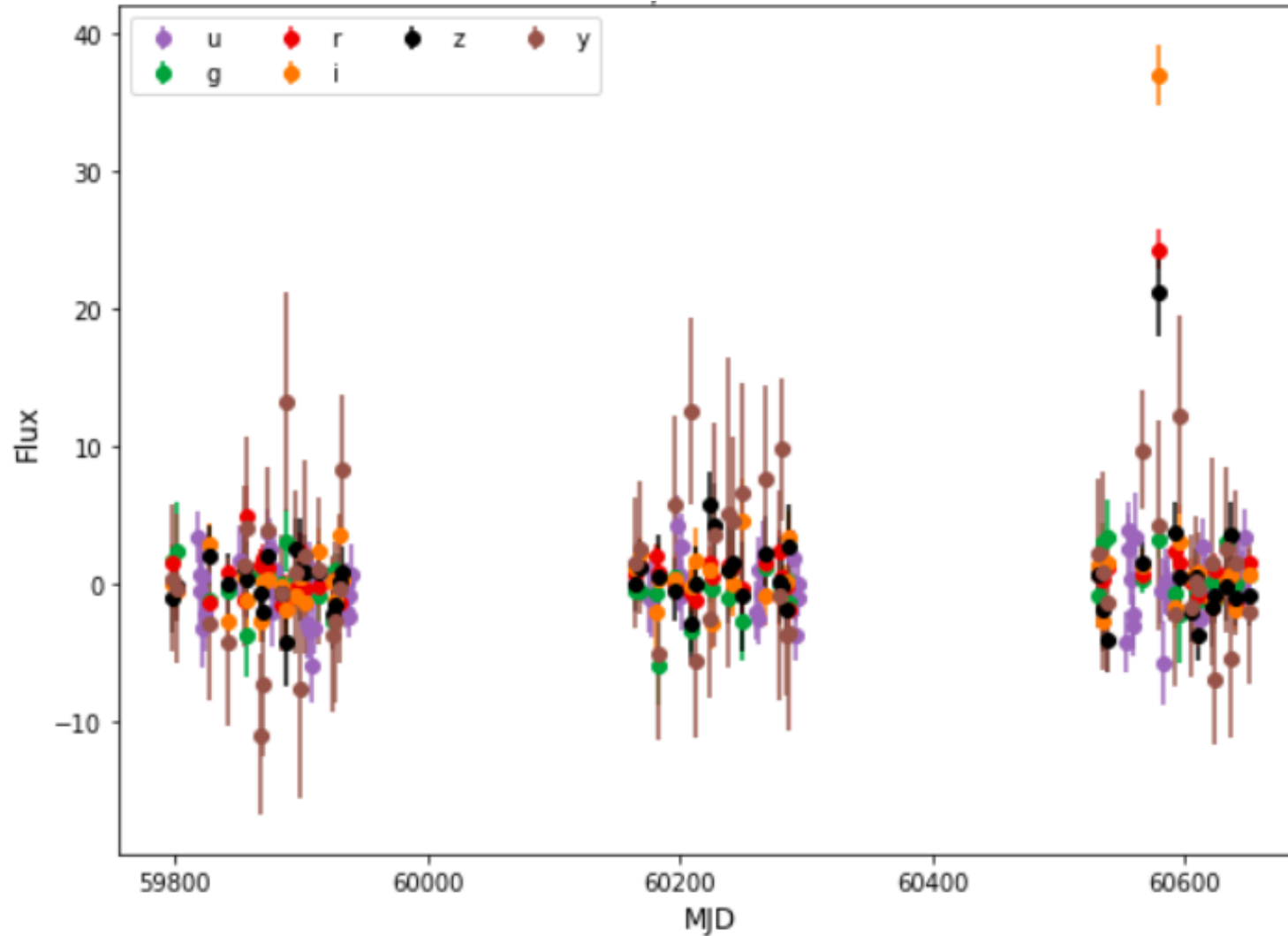
Periodicity Unrolled

Lomb-Scargle Periodogram (period=0.41 days)

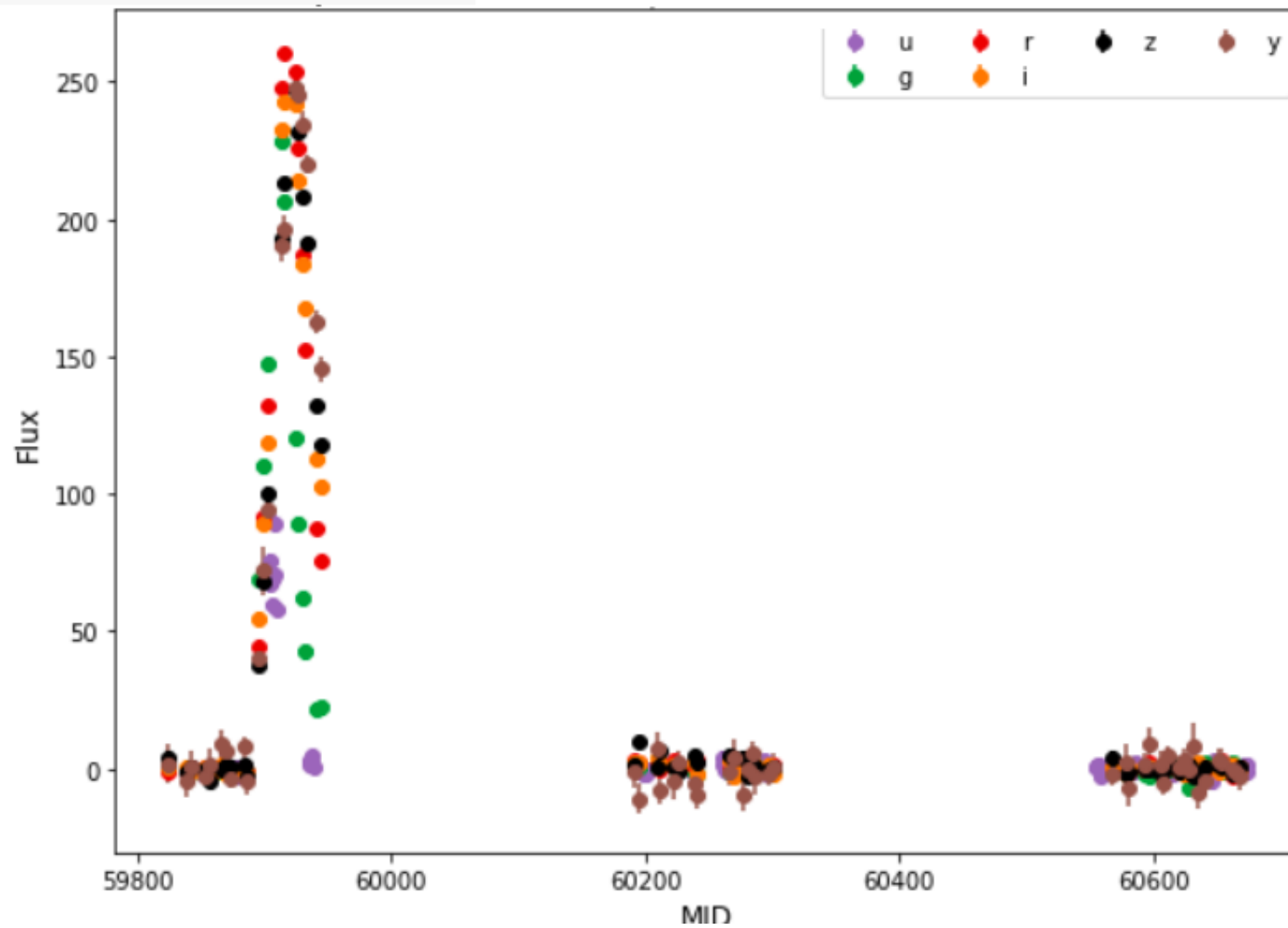


Signal-to-noise

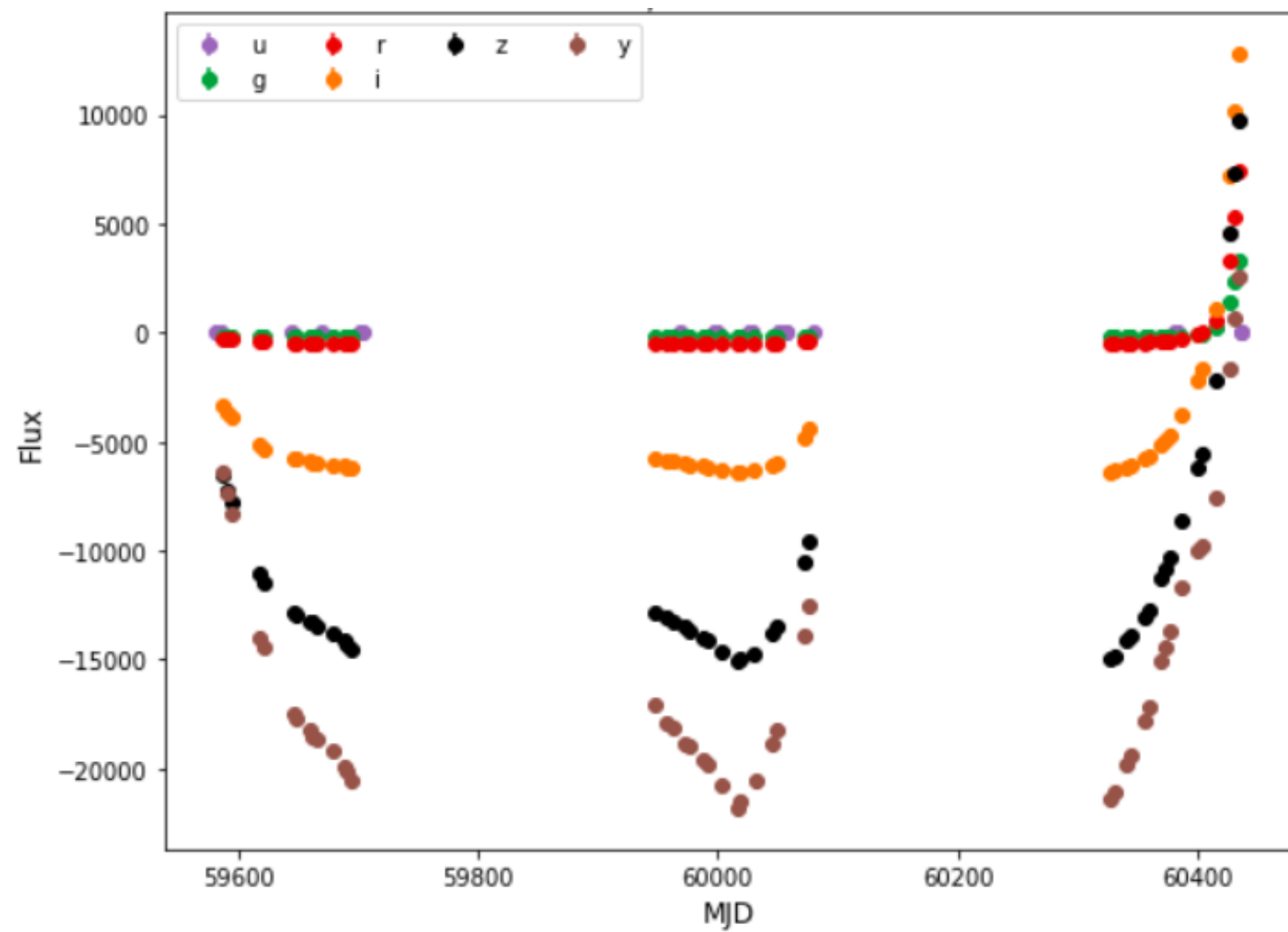
object_id 62187



10757



133773



The Data

- Two tables – metadata, timeseries
- Two datasets – training, test

Metadata

information about the object that doesn't change

- `object_id` - unique identifier
- `ra` - right ascension
- `decl` - declination
- `gal_l` - galactic longitude
- `gal_b` - galactic latitude
- `ddf` - flag that indicates data is from ddf survey (otherwise WFD)
- `hostgal_specz` - spectrographic redshift of the source - accurate measure of redshift - available in training and small part of test set
- `hostgal_photoz` - photometric redshift - meant as proxy for `hostgal_specz` but less accurate
- `hostgal_photoz_err` - uncertainty in above
- `distmod` - distance to source calculated from `hostgal_photoz`
- `mwebv` - extinction of light property along line of sight to milky way
- `target` = class of astronomical source

	<code>object_id</code>	<code>ra</code>	<code>decl</code>	<code>gal_l</code>	<code>gal_b</code>	<code>ddf</code>	<code>hostgal_specz</code>	<code>hostgal_photoz</code>	<code>hostgal_photoz_err</code>	<code>distmod</code>	<code>mwebv</code>	<code>target</code>
0	615	349.046051	-61.943836	320.796530	-51.753706	1	0.0000	0.0000	0.0000	NaN	0.017	92
1	713	53.085938	-27.784405	223.525509	-54.460748	1	1.8181	1.6267	0.2552	45.4063	0.007	88
2	730	33.574219	-6.579593	170.455585	-61.548219	1	0.2320	0.2262	0.0157	40.2561	0.021	42
3	745	0.189873	-45.586655	328.254458	-68.969298	1	0.3037	0.2813	1.1523	40.7951	0.007	90
4	1124	352.711273	-63.823658	316.922299	-51.059403	1	0.1934	0.2415	0.0176	40.4166	0.024	90

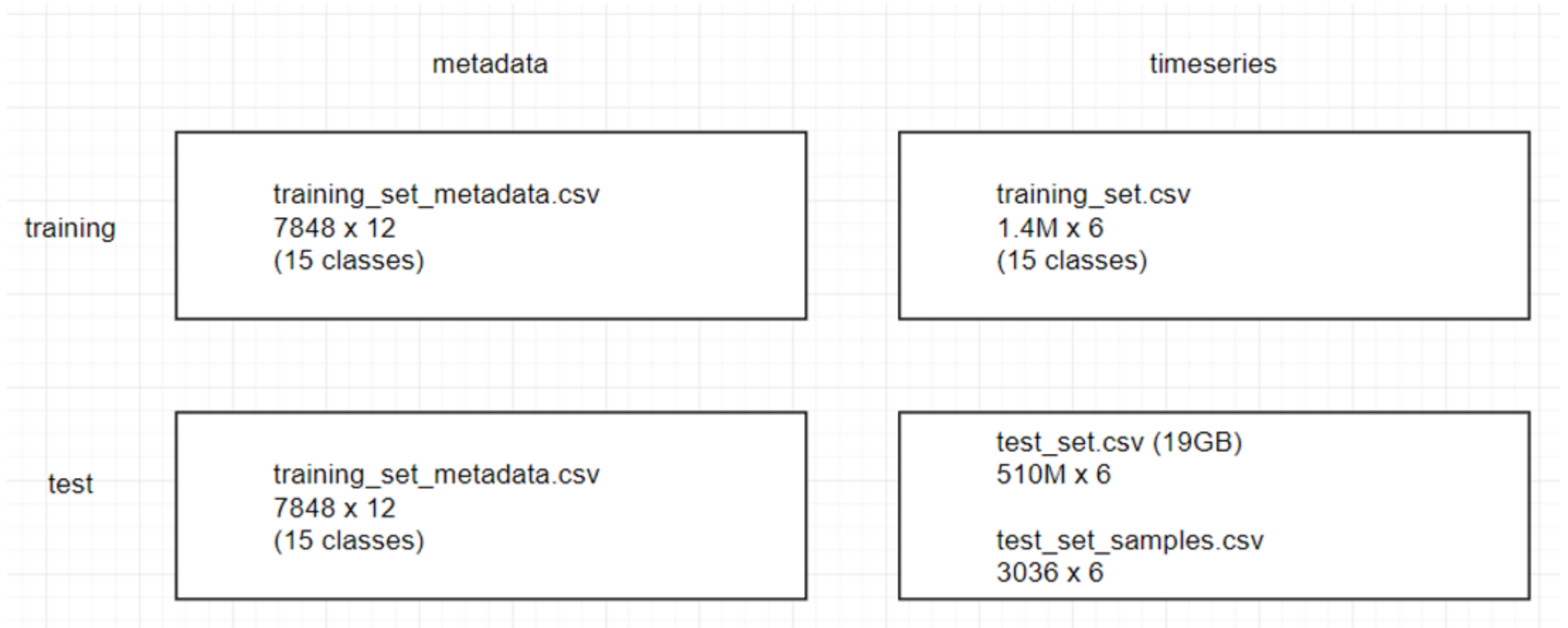
Timeseries

Intensity (flux) by passband

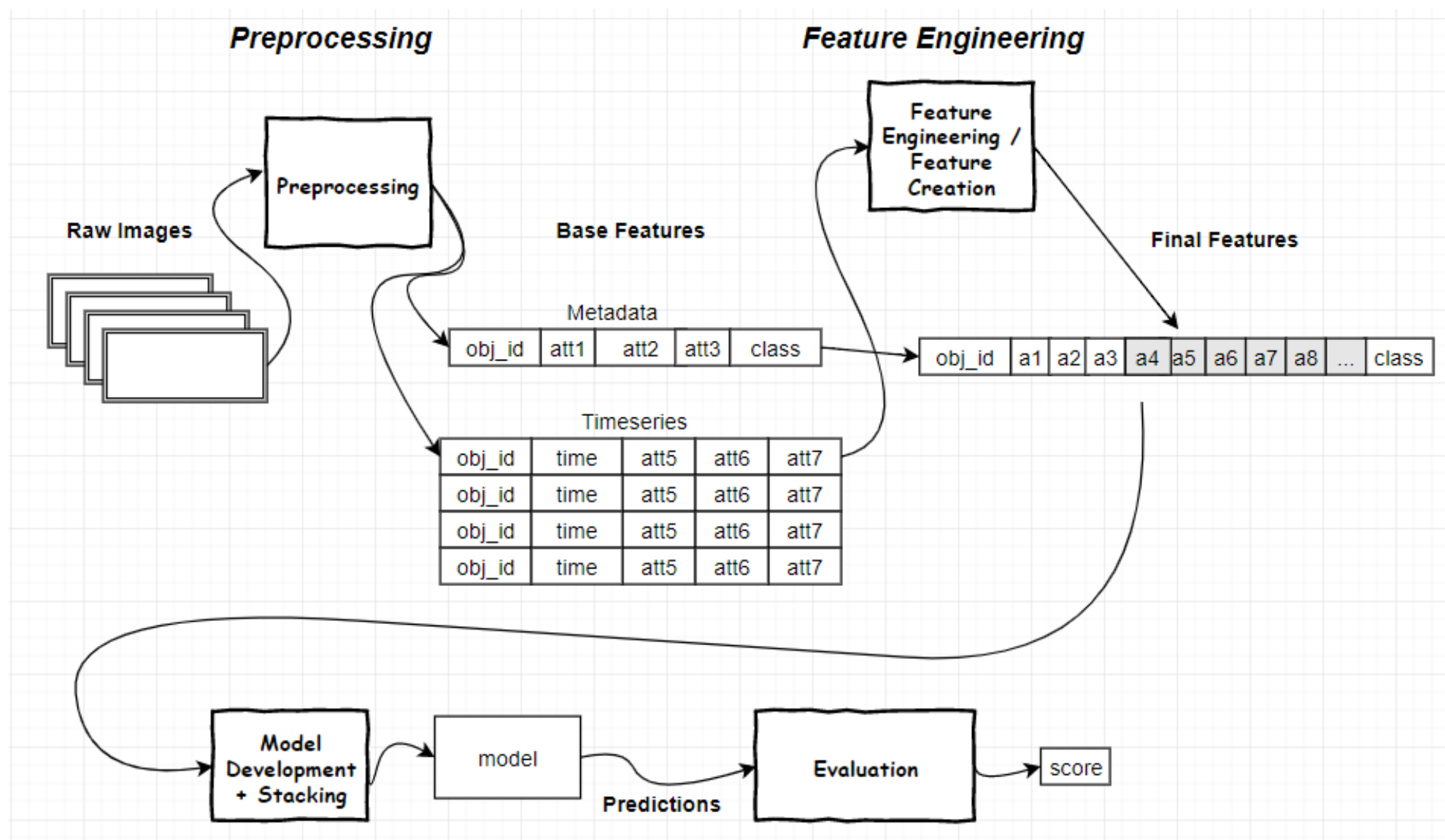
- Object_id = the object id
- MJD = date of sample (Modified Julian Date)
- Passband = frequency band of sample
- Flux = brightness
- Flux_err = uncertainty on measurement of flux
- Detected = 1 means brightness differs from the “template” by 3 sigma

	object_id	mjd	passband	flux	flux_err	detected
0	615	59750.4229	2	-544.810303	3.622952	1
1	615	59750.4306	1	-816.434326	5.553370	1
2	615	59750.4383	3	-471.385529	3.801213	1
3	615	59750.4450	4	-388.984985	11.395031	1
4	615	59752.4070	2	-681.858887	4.041204	1
5	615	59752.4147	1	-1061.457031	6.472994	1

19GB, 500M samples



Data Flow



Approaches Taken

Variety, and Commonality

Rank	Who	Code?	Feature Engineering / Augmentation	Modeling	url	
-	20th	GIBA	no	250 features feets(feature extractor for timeseries) aggregations, statistics by hand	LGBs, SVCs -> "stack ensemble"	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75262#latest-527064
	9th	Garreta	no	8000 pruned to 180 catboost(library for categorical data) @manugangler's kernel (light curves to microlensing event)	lgb + catboost + nn -> stacking	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75316#latest-495584
	14th	BTrotta	YES +pdf	~200(x4) features elementary operations (no curve fitting) bayes for removing noise from flux	LGB	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75054#latest-448552
	5th	CPMP	YES (well documented)	hand crafted from light curves (used Pandas) did NOT use packages (light gatspy, cesium, tsfresh) "slower" and "not as good" objective function(?)	lightGBM (almost exclusively)	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75050#latest-447982
	13th	Blonde	no	Parametric curve fittings (Bazin paper) cesium (ratios, std, skew) feets augmentation on flux	50/50 blend of two LGBM models	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75134#latest-445370
	4th	Ahmet Erdem	YES	Ratios (passband / all passbands) log-transformed (gives mult,div to NN) Sub-models to create features	LGB + NN + Stacking	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75011#latest-444878
	12th	Daniel Bi	no	non-frequency (light curves): hand-gen inspired by FATS = 50-60 features frequency: Lomb-Scargle (detect periodicity in unevenly spaced observations) with curve fitting based on "Bazin" paper	Three LGM + ensemble	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75237#latest-446568
	1st	Kyle Boone	YES	STRATEGY = focus on what ML needs -> most effort in separating super-novae because everything else was fairly easy to tell apart 200 features George (Gaussian Process Regression) augment training set by "degrading...to match test set"	single LGBM model with 5-fold cross-validation	https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75033#latest-457546

Takeaways:

- Feature Engineering is The Task. Between 200 and 8000 features into ML
- Focusing on what ML needs is key:
 - “most effort in separating Super-Novae (types) because everything else was fairly easy to tell apart” (K. Boone #1)
 - “log-transformed to allow CNN to do multiply, divide” (A. Erdem #4)
- Libraries...
 - 100% use LGB to stack lower-level models, some exclusively. A few CNNs.
 - Feature extraction libraries: feets, catboost, cesium, george
 - Periodicity: Lomb-Scargle
 - Much hand-crafting of features

14th Place Solution

B. Trotta



Pandas
Gold

Kaggle Plasticc Challenge

Belinda Trotta

January 8, 2019

This challenge requires us to classify objects in outer space into one of 15 categories based on the light they emit at various frequencies, and some basic metadata. Fourteen of the categories are present in the training data; the other category is for objects of types which have not yet been observed.

My solution is implemented in Python and uses LightGBM gradient boosted classification tree models. It scores 0.84070 on the private leaderboard, and runs in around 5.5 hours on a 24 Gb laptop (including calculating features, training, and prediction). It uses only elementary operations to calculate the features: there's no curve fitting or optimisation, which helps keep the runtime down. Apart from the hints revealed in the forum discussions, my original insights that gave the most improvement in score are: the Bayesian approach to removing noise from the flux measurements, adding features based on scaled flux values, adding features to capture the behaviour around the peak, and understanding how to optimise the metric (including for class 99). All these are described in more detail below.

1 Feature engineering

1.1 Removing noise from flux

Since some of the flux values have large errors, we use a Bayesian approach to estimate the most likely true value. We assume a prior distribution given by the flux observations for the same object and passband, and assume the observed value comes from a distribution whose mean is the true value of the flux, and which has standard deviation `flux_err`, since we are told in the Starter Kit kernel [1] that this is a 68% confidence interval.

We then calculate the mean of this posterior distribution from which we assume the observed value is drawn, using the formula in [2]. Note that we have only a single observed value from the posterior distribution, that is, $n = 1$ in the above calculation. This gives an estimate of

$$\frac{\mu_p/\sigma_p^2 + f/f_{err}^2}{1/\sigma_p^2 + 1/f_{err}^2}$$

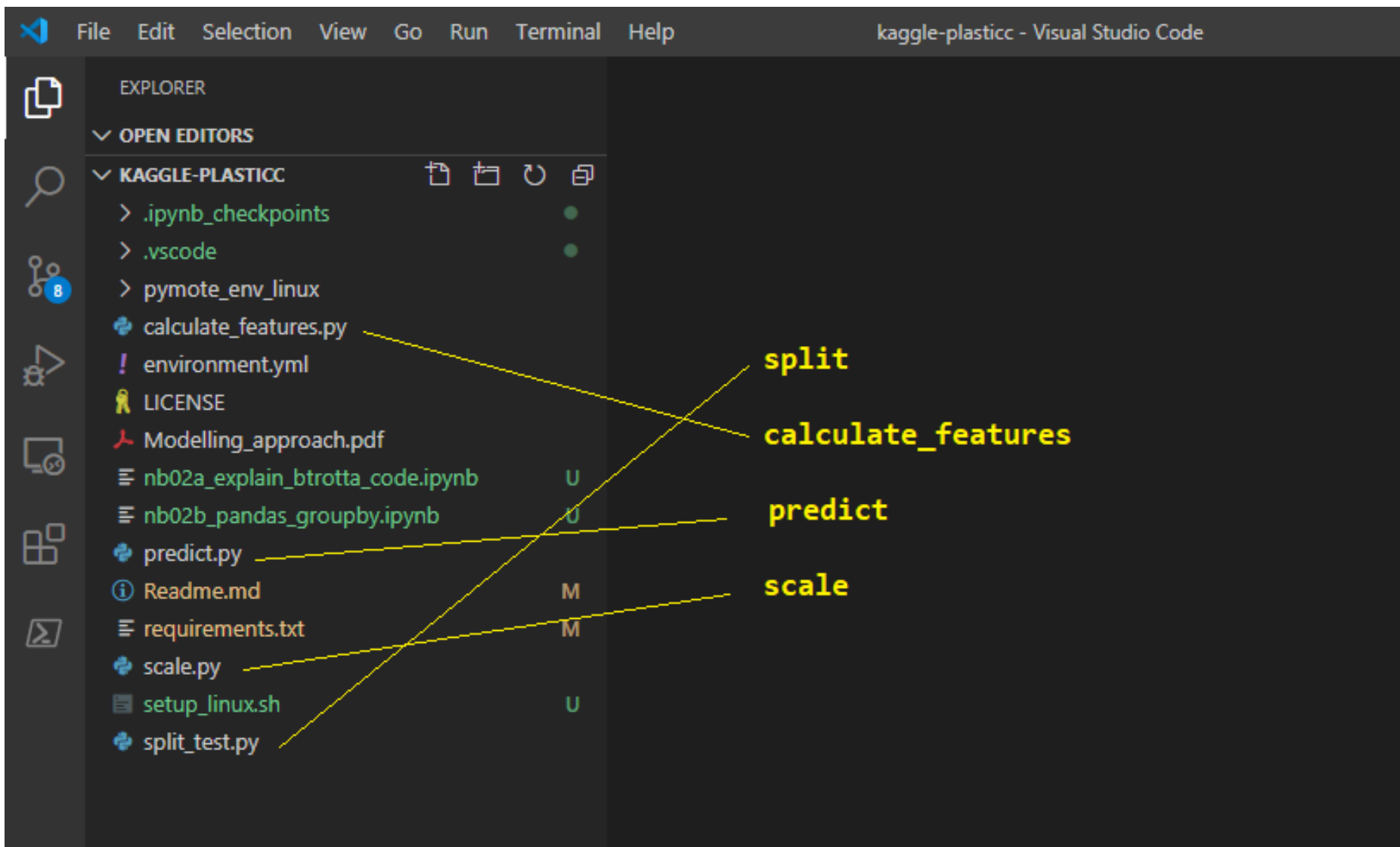
where μ_p and σ_p are the assumed prior mean and standard deviation (i.e. the mean and standard deviation of the flux for the given object and passband), f is the observed flux,

14th Place Solution

B. Trotta

<https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75054>

- Exactly 4 python files
- Procedure:
 - Download from git
 - Create “data” folder and download challenge data there
 - Run “split_test.py” Splits data into 100 .hdf5 files - takes about 15 minutes
 - Run “calculate_features.py” generates 3 features files – takes about 3.5 hours
 - Run “predict.py” train the model and make predictions (1.5 hours)
 - Run “scale.py” applies regularization and creates submission file (couple minutes)



split

split_test.py > ...

```
1  """Split the test data into chunks."""
2
3  import numpy as np
4  import pandas as pd
5  import os
6
7  n_chunks = 100
13 test = pd.read_csv(os.path.join('data', 'test_set.csv'), dtype=col_dict)
14 test.sort_values('object_id', inplace=True)
15 test = test.reset_index()
16 test_len = len(test)
18 id_diff = test.loc[test['object_id'].diff() != 0].index
19 chunk_starts = [id_diff[int(len(id_diff) * i / n_chunks)] for i in range(n_chunks)]
20 for i in range(n_chunks):
21     if i == n_chunks - 1:
22         end = len(test)
23     else:
24         end = chunk_starts[i + 1]
25     test.iloc[chunk_starts[i]: end - 1].to_hdf(os.path.join('data', 'split_{}'.format(n_chunks),
26                                                             'chunk_{}.hdf5'.format(i)), key='file0')
27
```

calculate_features.py

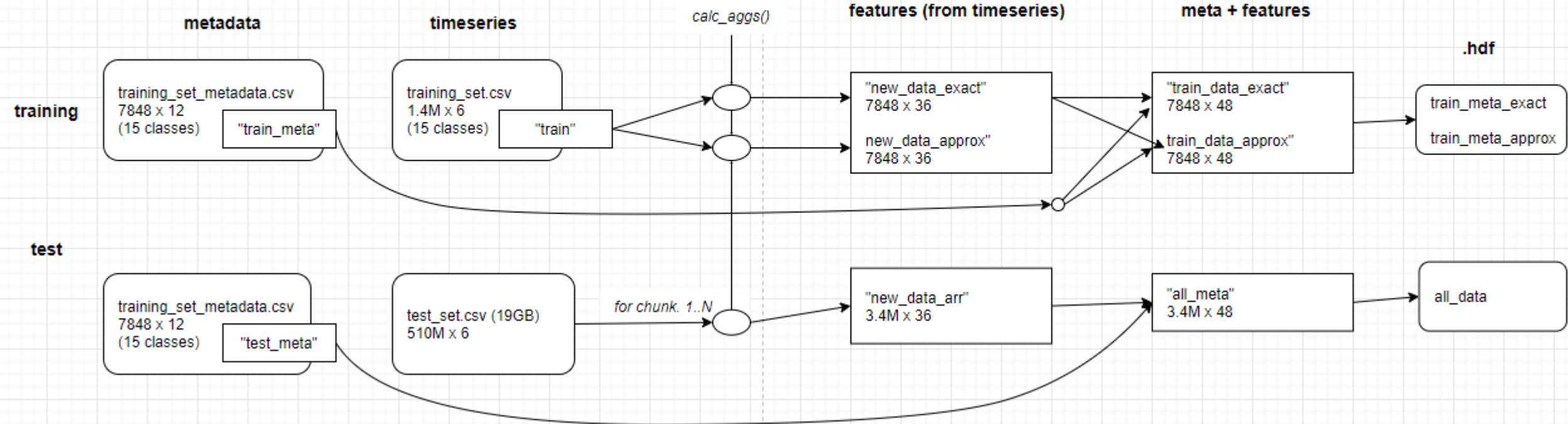
a.k.a. “feature engineering”

- For chunk 1..N
 - get data from file
 - calc_aggs()
- Merge w/metadat
- Write .hdf

```
205 # get the metadata
206 test_meta = pd.read_csv(os.path.join('data', 'test_set_metadata.csv'))
207 all_meta = pd.concat([train_meta, test_meta], axis=0, ignore_index=True, sort=True).reset_index()
208 all_meta.drop('index', axis=1, inplace=True)
209 n_chunks = 100
210
211 # calculate features
212 new_data_exact = calc_aggs(train.copy(), True)
213 new_data_approx = calc_aggs(train.copy(), False)
214 train_meta_exact = pd.merge(train_meta, new_data_exact, 'left', left_on='object_id', right_index=True)
215 train_meta_approx = pd.merge(train_meta, new_data_approx, 'left', left_on='object_id', right_index=True)
216
217 # process training set (not actually used, just to get right shape of dataframe)
218 new_data_arr = []
219 new_data_arr.append(calc_aggs(train.copy(), True))
220 # process test set
221 for i in range(n_chunks):
222     df = pd.read_hdf(os.path.join('data', 'split_{}'.format(n_chunks), 'chunk_{}.hdf5'.format(i)), key='file0')
223     df.drop('index', axis=1, inplace=True)
224     print('Read chunk {}'.format(i))
225     new_data_arr.append(calc_aggs(df.copy(), True))
226     print('Calculated features for chunk {}'.format(i))
227 del df
228 gc.collect()
229 new_data = pd.concat(new_data_arr, axis=0, sort=True)
230
231 # merge
232 all_meta = pd.merge(all_meta, new_data, 'left', left_on='object_id', right_index=True)
233
234 # write output
235 dir_name = 'features'
236 if not os.path.exists(os.path.join('data', dir_name)):
237     os.mkdir(os.path.join('data', dir_name))
238 all_meta.to_hdf(os.path.join('data', dir_name, 'all_data.hdf5'), key='file0')
239 train_meta_exact.to_hdf(os.path.join('data', dir_name, 'train_meta_exact.hdf5'), key='file0')
240 train_meta_approx.to_hdf(os.path.join('data', dir_name, 'train_meta_approx.hdf5'), key='file0')
241
```

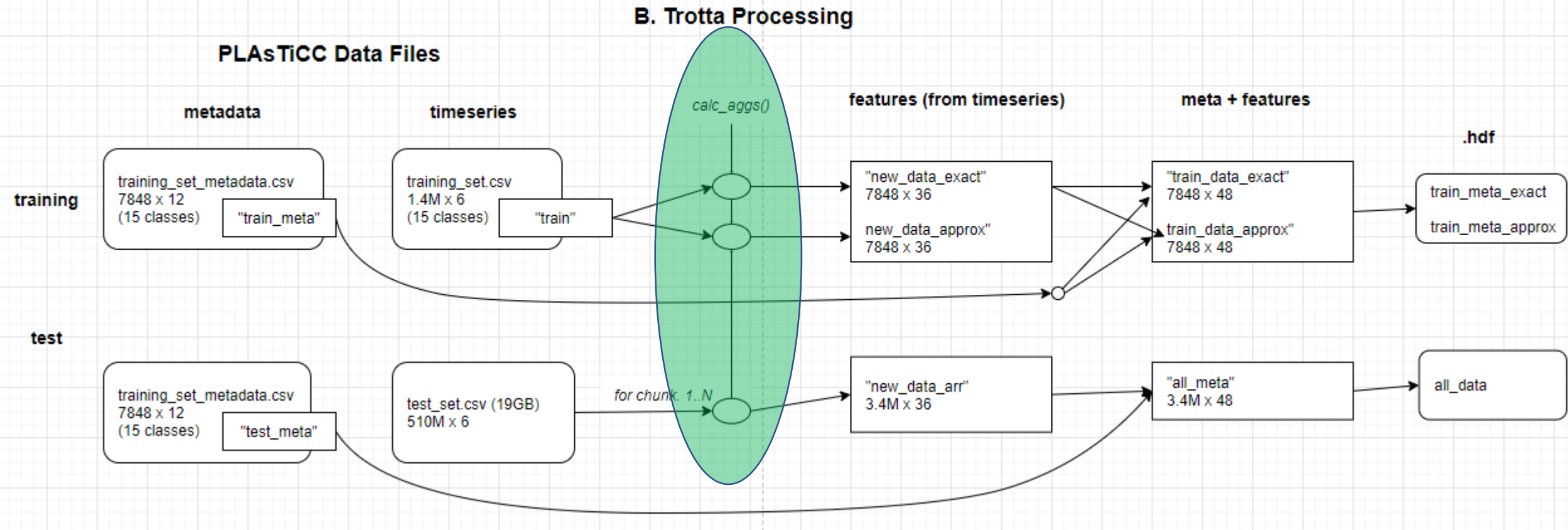
as a picture...

PLAsTiCC Data Files



all the work...

PLAsTiCC Data Files



Normalize the flux

For each [object, passband], calculate reductions mean and std

Use that to scale the flux (Bayes calculation)

Add “bayes_flux” as new feature to timeseries and overwrite “flux”

```
# Normalise the flux, following the Bayesian approach here:
# https://www.statlect.com/fundamentals-of-statistics/normal-distribution-Bayesian-estimation
# Similar idea (but not the same) as the normalisation done in the Starter Kit
# https://www.kaggle.com/michaelapers/the-plasticc-astronomy-starter-kit?scriptVersionId=6040398
prior_mean = all_data.groupby(['object_id', 'passband'])['flux'].transform('mean')
prior_std = all_data.groupby(['object_id', 'passband'])['flux'].transform('std')
prior_std.loc[prior_std.isnull()] = all_data.loc[prior_std.isnull(), 'flux_err']
obs_std = all_data['flux_err'] # since the above kernel tells us that the flux error is the 68% confidence interval
all_data['bayes_flux'] = (all_data['flux'] / obs_std**2 + prior_mean / prior_std**2) \
    / (1 / obs_std**2 + 1 / prior_std**2)
all_data.loc[all_data['bayes_flux'].notnull(), 'flux'] \
    = all_data.loc[all_data['bayes_flux'].notnull(), 'bayes_flux']
```

before

	object_id	mjd	passband	flux	flux_err	detected
702	730	59798.3205	2	1.177371	1.364300	0
703	730	59798.3281	1	2.320849	1.159247	0
704	730	59798.3357	3	2.939447	1.771328	0
705	730	59798.3466	4	2.128097	2.610659	0
706	730	59798.3576	5	-12.809639	5.380097	0

after

	object_id	mjd	passband	flux	flux_err	detected	bayes_flux
702	730	59798.3205	2	1.246867	1.364300	0	1.246867
703	730	59798.3281	1	1.685412	1.159247	0	1.685412
704	730	59798.3357	3	2.952700	1.771328	0	2.952700
705	730	59798.3466	4	2.250392	2.610659	0	2.250392
706	730	59798.3576	5	-10.380242	5.380097	0	-10.380242

DataFrame.groupby()

https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html



- Split -> Apply -> Combine
- Split (grouping) establishes “a mapping of labels to group names” (keys)
 - Can be done via function, list, dict, string indicating df column
- Apply can be:
 - Aggregation: (compute reduction statistic and apply to the group)
 - Filtration: (compute reduction True/False and discard some groups based on it)
 - Transformation (compute element-wide function - returns a like-index object)

Example:

```
df = pd.DataFrame({'A': ['one', 'one', 'two', 'three', 'three', 'one'], 'B': range(6)})
```

```
gb = df.groupby('A')
```

```
gb.display(), gb.count(), gb.transform('mean') gb.transform(lambda x:x+1)
```

Estimate Flux at Source

- Redshift is in the metadata (not timeseries) so no groupby needed. Just copy specz or photoz
- Apply the inverse-square calculation to “flux”

```
# Estimate the flux at source, using the fact that light is proportional
# to inverse square of distance from source.
# This is hinted at here: https://www.kaggle.com/c/PLAsTiCC-2018/discussion/70725#417195
redshift = all_meta.set_index('object_id')[['hostgal_specz', 'hostgal_photoz']]
if exact:
    redshift['redshift'] = redshift['hostgal_specz']
    redshift.loc[redshift['redshift'].isnull(), 'redshift'] \
        = redshift.loc[redshift['redshift'].isnull(), 'hostgal_photoz']
else:
    redshift['redshift'] = redshift['hostgal_photoz']
all_data = pd.merge(all_data, redshift, 'left', 'object_id')
nonzero_redshift = all_data['redshift'] > 0
all_data.loc[nonzero_redshift, 'flux'] = all_data.loc[nonzero_redshift, 'flux'] \
    * all_data.loc[nonzero_redshift, 'redshift']**2
```


	object_id	mjd	passband	flux	flux_err	detected	bayes_flux
702	730	59798.3205	2	1.246867	1.364300	0	1.246867
703	730	59798.3281	1	1.685412	1.159247	0	1.685412
704	730	59798.3357	3	2.952700	1.771328	0	2.952700
705	730	59798.3466	4	2.250392	2.610659	0	2.250392
706	730	59798.3576	5	-10.380242	5.380097	0	-10.380242

	object_id	mjd	passband	flux	flux_err	detected	bayes_flux	hostgal_specz	hostgal_photoz	redshift
702	730	59798.3205	2	0.067111	1.364300	0	1.246867	0.232	0.2262	0.232
703	730	59798.3281	1	0.090716	1.159247	0	1.685412	0.232	0.2262	0.232
704	730	59798.3357	3	0.158926	1.771328	0	2.952700	0.232	0.2262	0.232
705	730	59798.3466	4	0.121125	2.610659	0	2.250392	0.232	0.2262	0.232
706	730	59798.3576	5	-0.558706	5.380097	0	-10.380242	0.232	0.2262	0.232

pd.merge()

super powerful...



DataFrame.merge(self, right, how, on, suffixes, validate)

- Right: other DataFrame
- How: inner, left, right, outer
- On: merge key
- Suffixes: if merge results in duplicate column names
- Validate: optional check for 1-1, 1:m, m:1

Aggregate Features

- Mean, STD, Max and Min for each [object,passband]
- 25% and 75% quantiles

```
# aggregate features
band_aggs = all_data.groupby(['object_id', 'passband'])['flux'].agg(['mean', 'std', 'max', 'min']).unstack(-1)
band_aggs.columns = [x + '_' + str(y) for x in band_aggs.columns.levels[0]]
for y in band_aggs.columns.levels[1]]
all_data.sort_values(['object_id', 'passband', 'flux'], inplace=True)
# this way of calculating quantiles is faster than using the pandas quantile builtin on the groupby object
all_data['group_count'] = all_data.groupby(['object_id', 'passband']).cumcount()
all_data['group_size'] = all_data.groupby(['object_id', 'passband'])['flux'].transform('size')
q_list = [0.25, 0.75]
for q in q_list:
    all_data['q_' + str(q)] = all_data.loc[
        (all_data['group_size'] * q).astype(int) == all_data['group_count'], 'flux']
quantiles = all_data.groupby(['object_id', 'passband'])[['q_' + str(q) for q in q_list]].max().unstack(-1)
quantiles.columns = [str(x) + '_' + str(y) + '_quantile' for x in quantiles.columns.levels[0]]
for y in quantiles.columns.levels[1]]
```

```
all_data[all_data["object_id"]==730]
```

ject_id	mjd	passband	flux	flux_err	detected	bayes_flux	hostgal_specz	hostgal_photoz	redshift	group_count	group_size	q_0.25	q_0.75
730	60643.0521	0	-0.095862	1.682890	0	-1.781029	0.232	0.2262	0.232	0	72	NaN	NaN
730	60290.0761	0	-0.077771	1.929932	0	-1.444906	0.232	0.2262	0.232	1	72	NaN	NaN
730	59938.0647	0	-0.076970	2.015928	0	-1.430028	0.232	0.2262	0.232	2	72	NaN	NaN
730	60558.2332	0	-0.066238	2.511074	0	-1.230635	0.232	0.2262	0.232	3	72	NaN	NaN
730	60646.0636	0	-0.029485	1.798218	0	-0.547804	0.232	0.2262	0.232	18	72	-0.029485	NaN
730	59906.0562	0	-0.029049	2.043133	0	-0.539706	0.232	0.2262	0.232	19	72	NaN	NaN

DataFrame.agg()

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.agg.html>



Pandas Gold

- “Aggregate using one or more operations over the specified axis”
- `DataFrame.agg(self, func, axis=0, *args, **kwargs)`
- **Func:** function, list of functions, dictionary of label->function
- **Axis:** 0-> apply to columns, 1-> apply to rows

“most_extreme()”

- Find the "most extreme" time for each object and each band
- Retrieve the k data points on either side
- Procedure:
 - for each passband - translate to it's median
 - find the date of the peak (largest value)
 - for each sample identify the number of days to/from the peak
 - sort by days before/after in order to find the n preceding, and n following

most_extreme (1)

find the max value

compute the mean and
distance from the
mean

for each [object,passband] compute the median
add it to the dataframe as column "object_passband_mean"

```
df['object_passband_mean'] = df.groupby(['object_id', 'passband'])['flux'].transform('median')
```

```
df['dist_from_mean'] = (df['flux'] - df['object_passband_mean'])
```

find the index at
the max point

for each object, find the max distance from mean
idxmax returns the index of the max instance
create a new dataframe "max_time" with this

```
max_time = df.loc[df['detected'] == 1].groupby('object_id')['dist_from_mean'].idxmax().to_frame('max_ind')
```

get the value at
that point

from the max_time dataframe get the index of the max
and then get the value of that data point.
Add it to the max_time as "mjd_max/min"

```
max_time['mjd_max' + suffix] = df.loc[max_time['max_ind'].values, 'mjd'].values
```

“most_extreme()” (2)

get first K after max

sort by object_id, passband, time_after_max
group by [object_id, passband] and number the entries as "row_num_after"
create dataframe with k entries for each group
unstack (k rows to k columns)
name the columns (point)1_(object)10_after

```
# first k after event
df.sort_values(['object_id', 'passband', 'time_after_mjd_max'], inplace=True)
df['row_num_after'] = df.loc[df['time_after_mjd_max'] >= 0].groupby(
    ['object_id', 'passband']).cumcount()
first_k_after = df.loc[(df['row_num_after'] < k) & (df['time_after_mjd_max'] <= 50),
    ['object_id', 'passband', 'flux', 'row_num_after']]
first_k_after.set_index(['object_id', 'passband', 'row_num_after'], inplace=True)
first_k_after = first_k_after.unstack(level=-1).unstack(level=-1)
first_k_after.columns = [str(x) + '_' + str(y) + '_after' for x in first_k_after.columns.levels[1]
    for y in first_k_after.columns.levels[2]]
```


“most_extreme()” (3)

calculate mean flux for “time bands” around the max

create a list of “time bands” and iterate
get entries that are within [start, end]
for each [object, passband] group compute mean flux, unstack to passband columns
name the columns “10_start_20_end

```
extreme_data = first_k_after
time_bands = [[-50, -20], [-20, -10], [-10, 0], [0, 10], [10, 20], [20, 50], [50, 100], [100, 200], [200, 500]]
if include_interval:
    interval_arr = []
    for start, end in time_bands:
        band_data = df.loc[(start <= df['time_after_mjd_max']) & (df['time_after_mjd_max'] <= end)]
        interval_agg = band_data.groupby(['object_id', 'passband'])['flux'].mean().unstack(-1)
        interval_agg.columns = ['{}_start_{}_end_{}'.format(c, start, end) for c in interval_agg.columns]
        interval_arr.append(interval_agg)
    interval_data = pd.concat(interval_arr, axis=1)
    extreme_data = pd.concat([extreme_data, interval_data], axis=1)
```

Periodicity identification

- Strategy - do not need period, only a flag (periodic vs episodic/cataclysmic)
- Roll-your own (vs library) satisfies w/ much lower computes

```
# add the feature mentioned here, attempts to identify periodicity:
# https://www.kaggle.com/c/PLAsTiCC-2018/discussion/69696#410538
time_between_detections = all_data.loc[all_data['detected'] == 1].groupby('object_id')['mjd'].agg(['max', 'min'])
time_between_detections['det_period'] = time_between_detections['max'] - time_between_detections['min']
# same feature but grouped by passband
time_between_detections_pb \
    = all_data.loc[all_data['detected'] == 1].groupby(['object_id', 'passband'])['mjd'].agg(['max', 'min'])
time_between_detections_pb['det_period'] = time_between_detections_pb['max'] - time_between_detections_pb['min']
time_between_detections_pb = time_between_detections_pb['det_period'].unstack(-1)
time_between_detections_pb.columns = ['det_period_pb_' + str(i) for i in range(6)]
# similar feature based on high values
all_data['threshold'] = all_data.groupby(['object_id'])['flux'].transform('max') * 0.75
all_data['high'] = ((all_data['flux'] >= all_data['threshold']) & (all_data['detected'] == 1)).astype(int)
time_between_highs = all_data.loc[all_data['high'] == 1].groupby('object_id')['mjd'].agg(['max', 'min'])
time_between_highs['det_period_high'] = time_between_highs['max'] - time_between_highs['min']
```

In Summary

- PLAsTiCC is a great example of a timeseries dataset
 - Challenging in size and sampling (periodicity and passbands)
 - Rich in underlying structure (types of stars and characteristics)
 - Tractable in concept
- Competitors show strong consistency (LGB), with differences in underlying ML models, approach and libraries
- B.Trotta code is wonderfully structured and documented
- A rich source for Pandas examples

Thank You