# 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

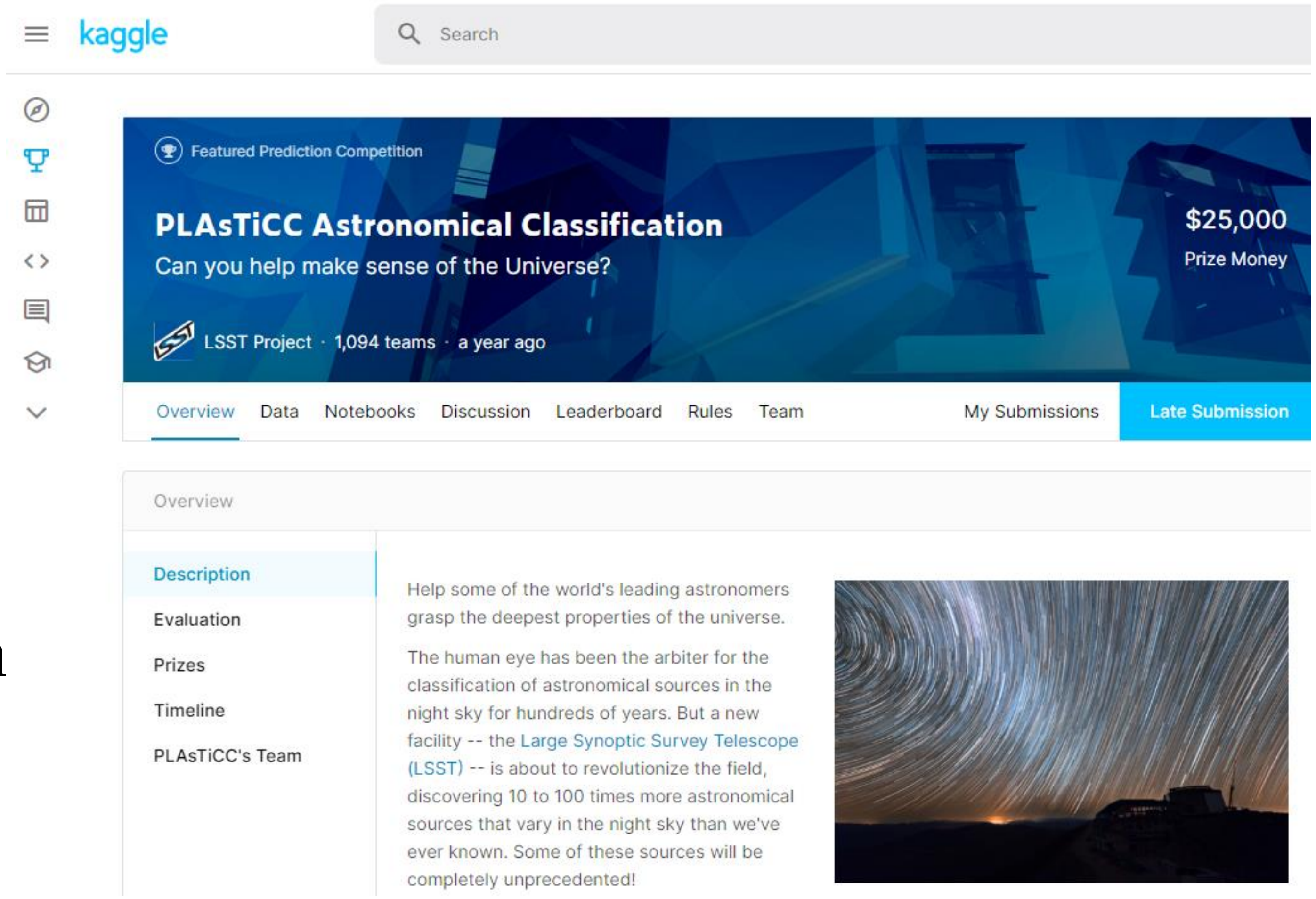
# PLAsTiCC, LSST and Kaggle

- **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



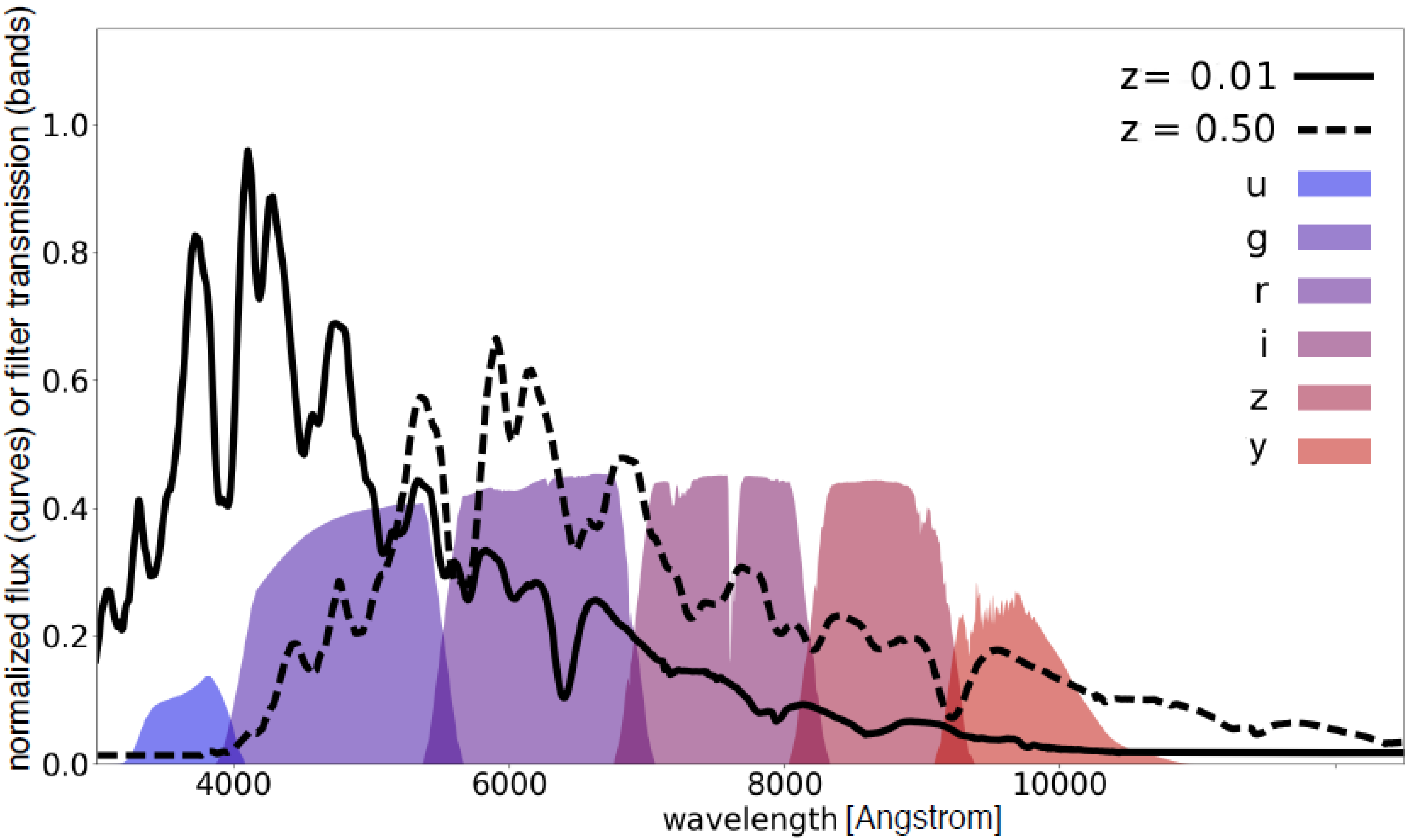
# LSST Data Size/Scale

| | |
|---|---|
| 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




https://arxiv.org/abs/1810.00001

# Template (differential sampling)

A “template” technique is used to measure intensity

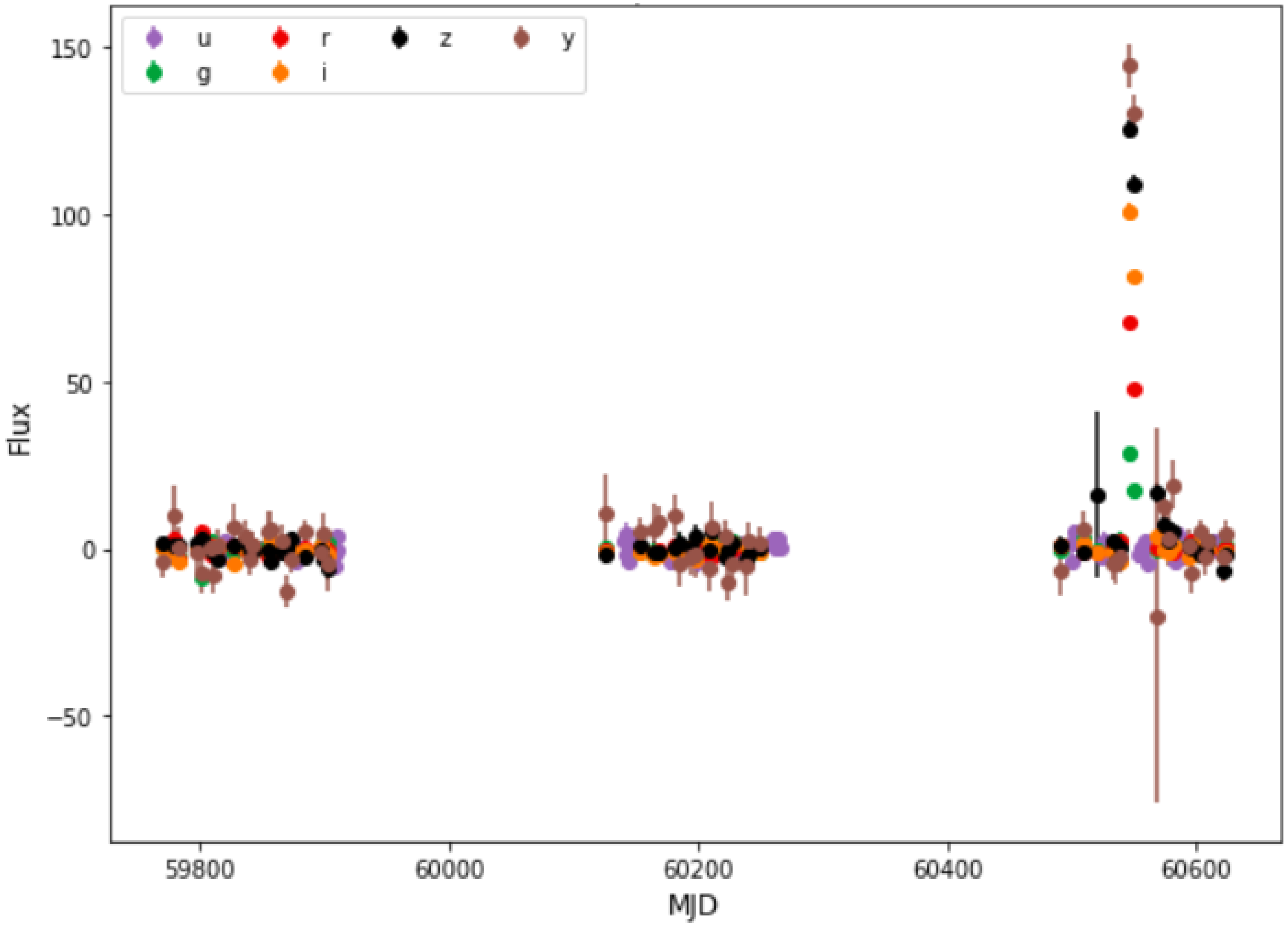
Expressed as “flux” - intensity relative to template

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:   flux = sample – template
- Flux can be negative
- Flux_error is computed (no details here)

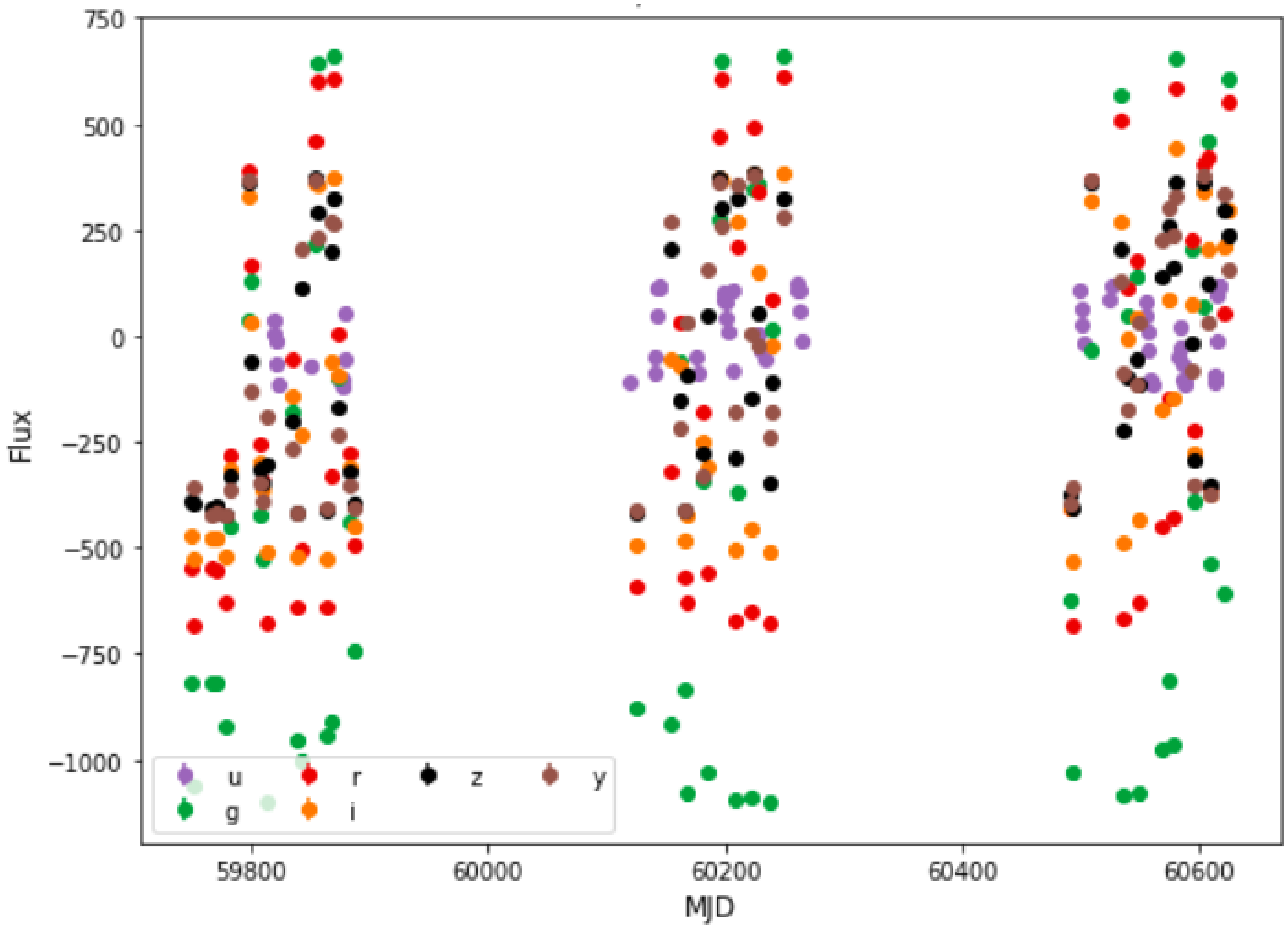
# Impact of Schedule

object_id 3910

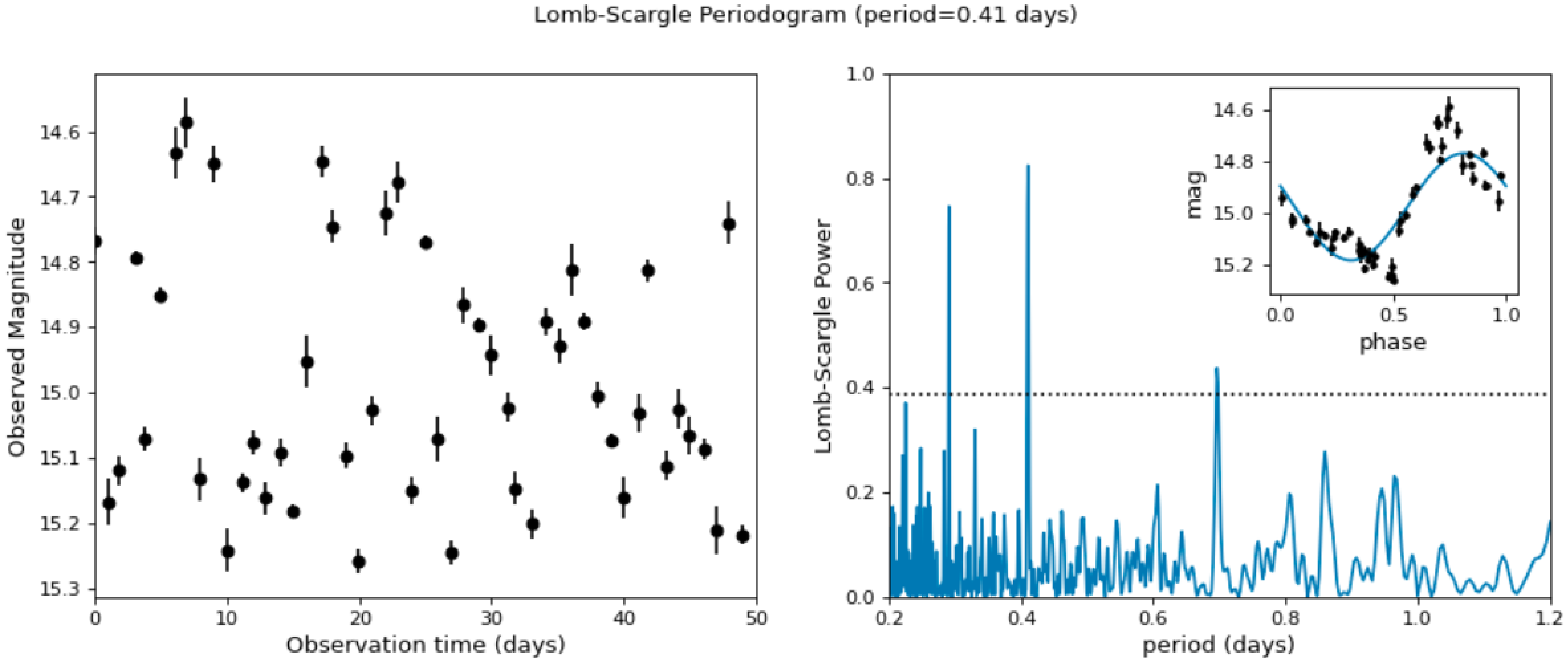


# Impact of Sample Rate < Signal Rate

object_id 615

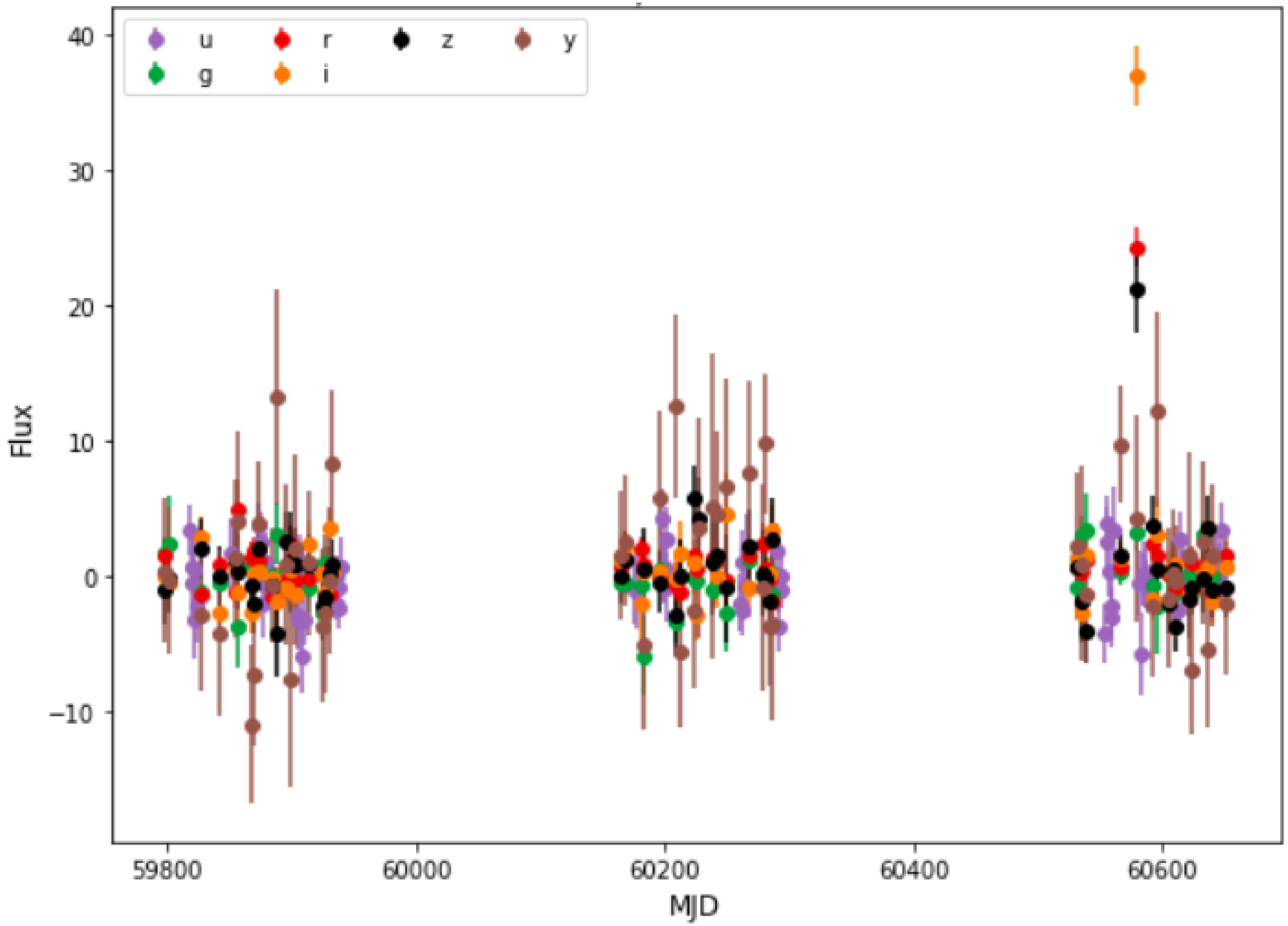


# Periodicity Unrolled

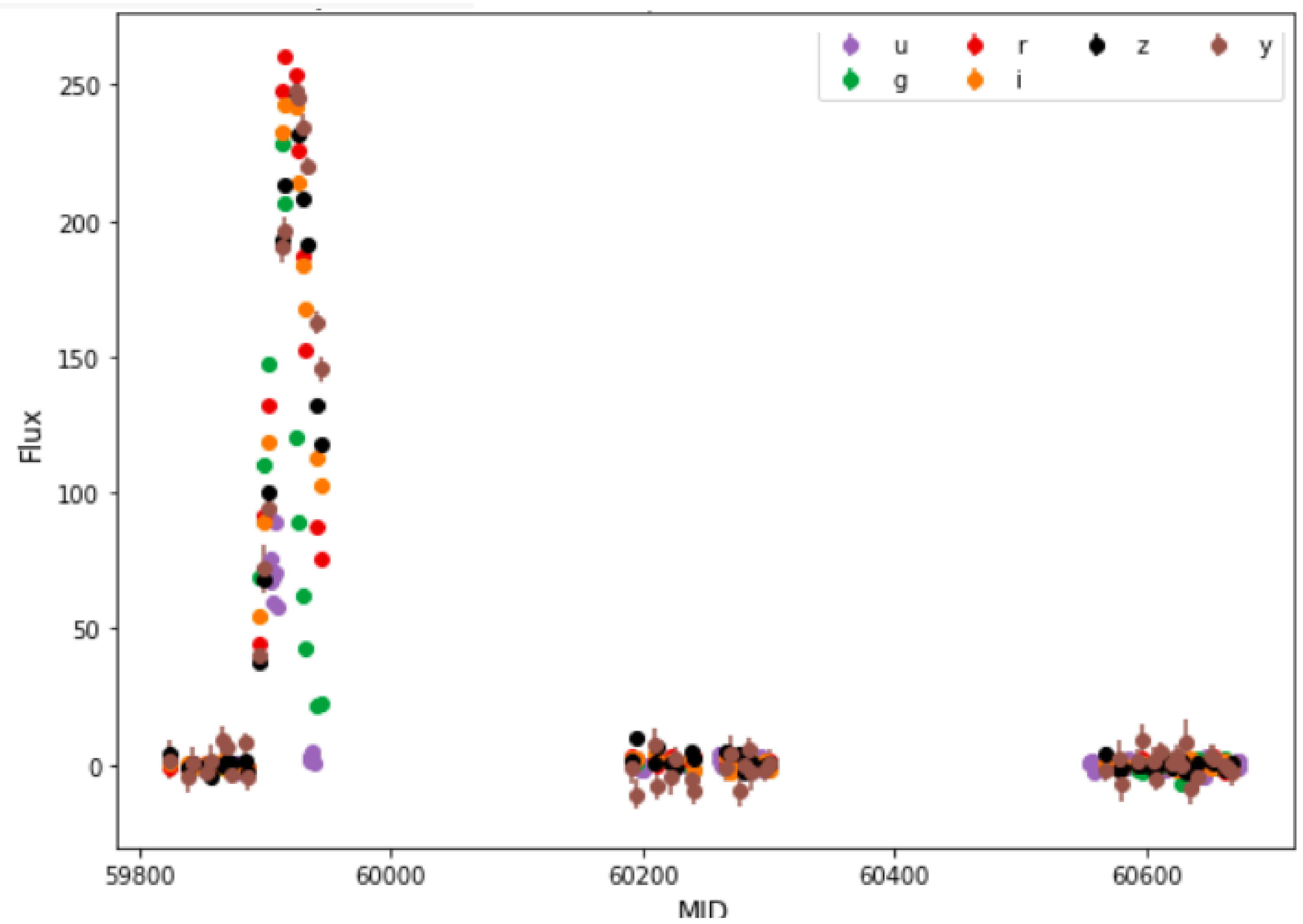



https://docs.astropy.org/en/stable/timeseries/lombscargle.html

# Signal-to-noise
object_id 62187

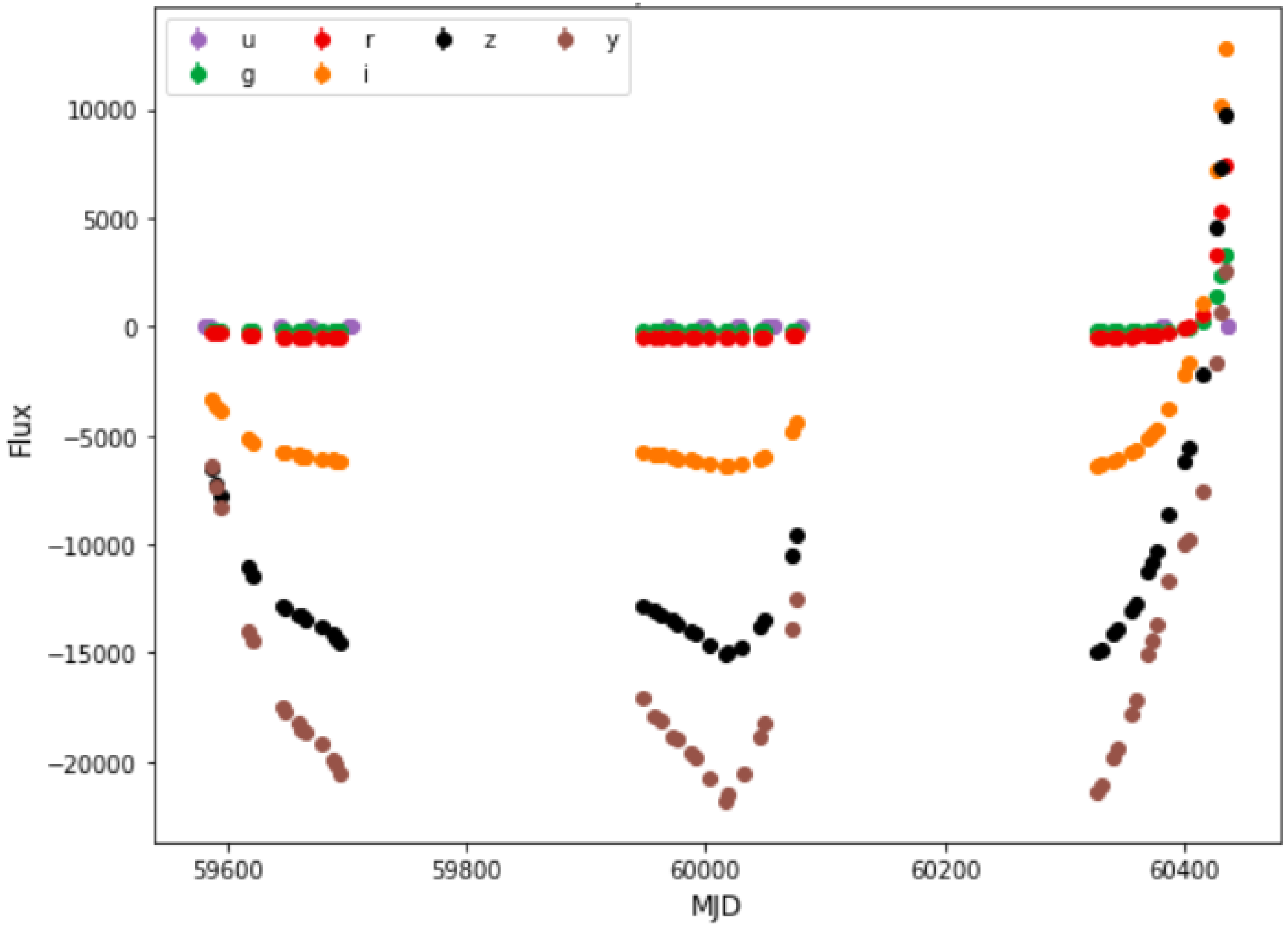


# 10757



u
g
r
i
z
y
Flux
MJD
250
200
150
100
50
0
59800
60000
60200
60400
60600

# 133773



u
r
z
y
g
i
10000
5000
0
−5000
−10000
−15000
−20000
Flux
59600
59800
60000
60200
60400
MJD

# 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

| | object_id | ra | decl | gal_l | gal_b | ddf | hostgal_specz | hostgal_photoz | hostgal_photoz_err | distmod | mwebv | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 |

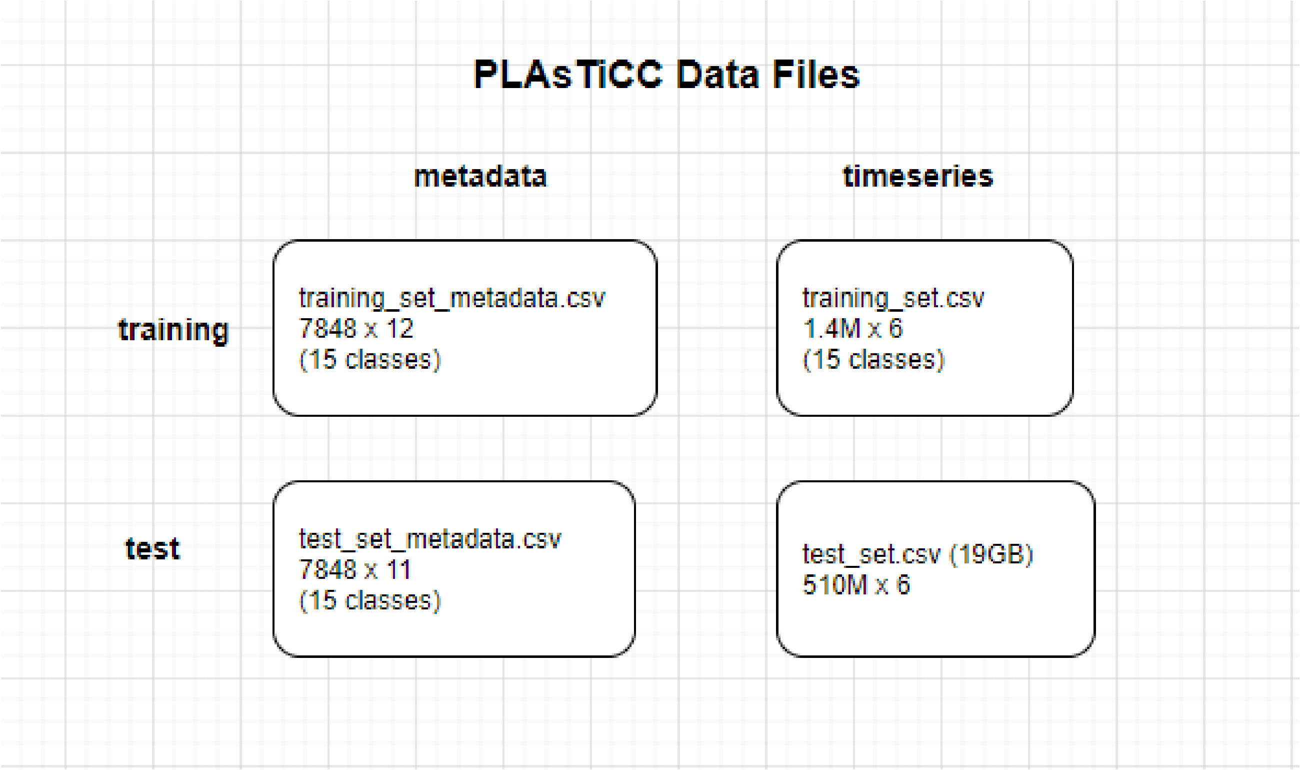
# Timeseries

Intensity (flux) by passband

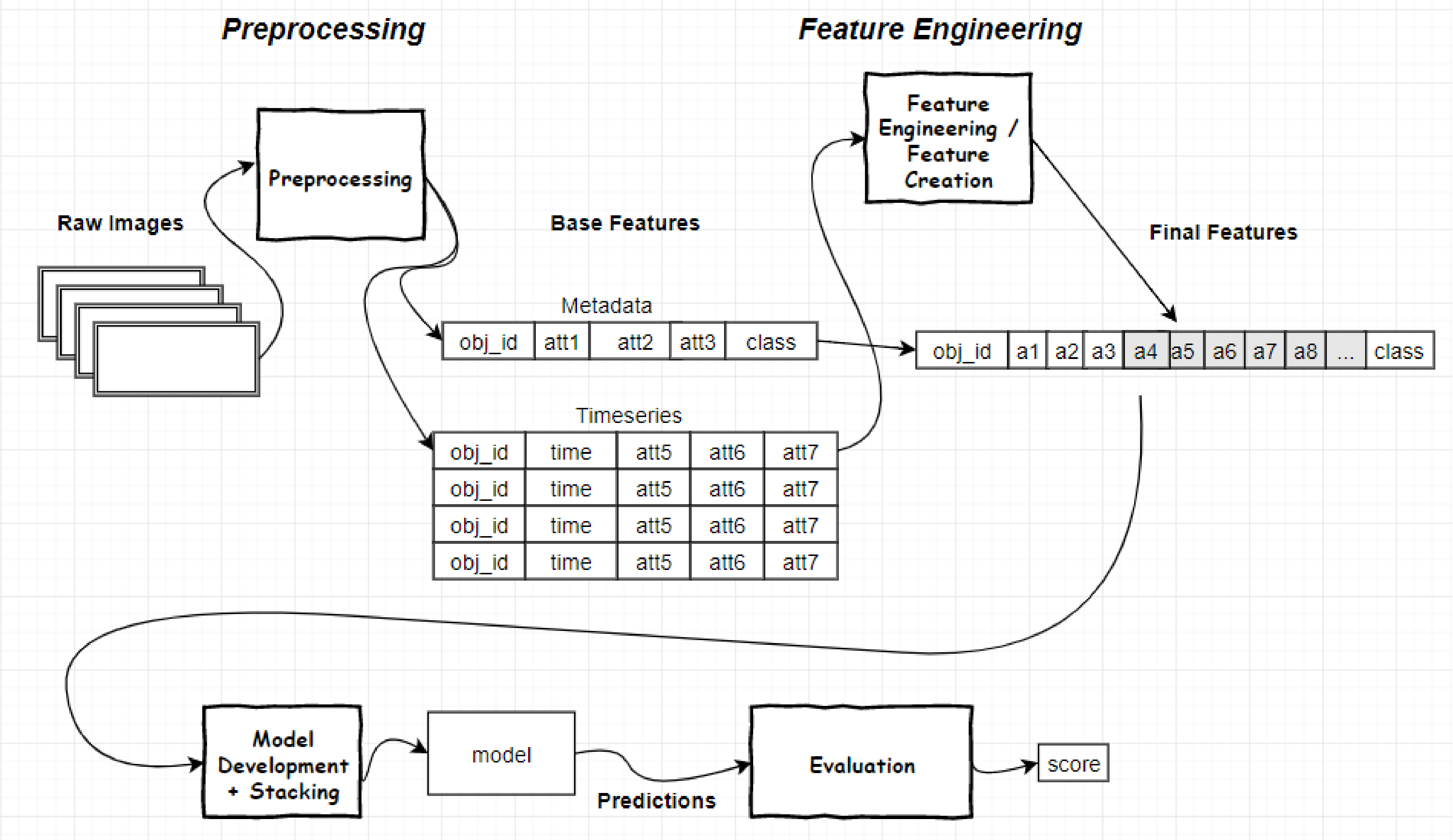
- Object_id = the object id
- MJD = date of sample (Modified Julian Date)
- Passband = frequency band of sample
- Flux = brightness
- Flux_err - = uncertainty on measurement of flux
- Detected = 1 means brightness differs from the "template" by 3 sigma

| | 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, 510M samples



# Data Flow (big picture)



# Approaches Taken

# Variety, and Commonality

| | Rank | Who | Code? | Feature Engineering / Augmentation | Modeling | url |
|---|---|---|---|---|---|---|
| - | 20th | GIBA | no | 250 features<br>***feets(feature extractor for timeseries)***<br>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<br>***catboost(library for categorical data)***<br>@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<br>elementary operations (no curve fitting)<br>***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)<br>did NOT use packages (light gatspy, cesium, tsfresh) "slower" and "not as good"<br>***objective function(?)*** | lightGBM (almost exclusively) | https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75050#latest-447982 |
| | 13th | Blonde | no | Parametric curve fittings (***Bazin paper***)<br>***cesium*** (ratios, std, skew)<br>***feets***<br>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)<br>***log-transformed*** (gives mult,div to NN)<br>***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<br>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<br>200 features<br>**George** (Gaussian Process Regression)<br>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 $\mu_p$ and $\sigma_p$ 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,

# $14^{th}$ 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 (about 15 minutes)
  - Run “calculate_features.py” generates 3 features files (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)

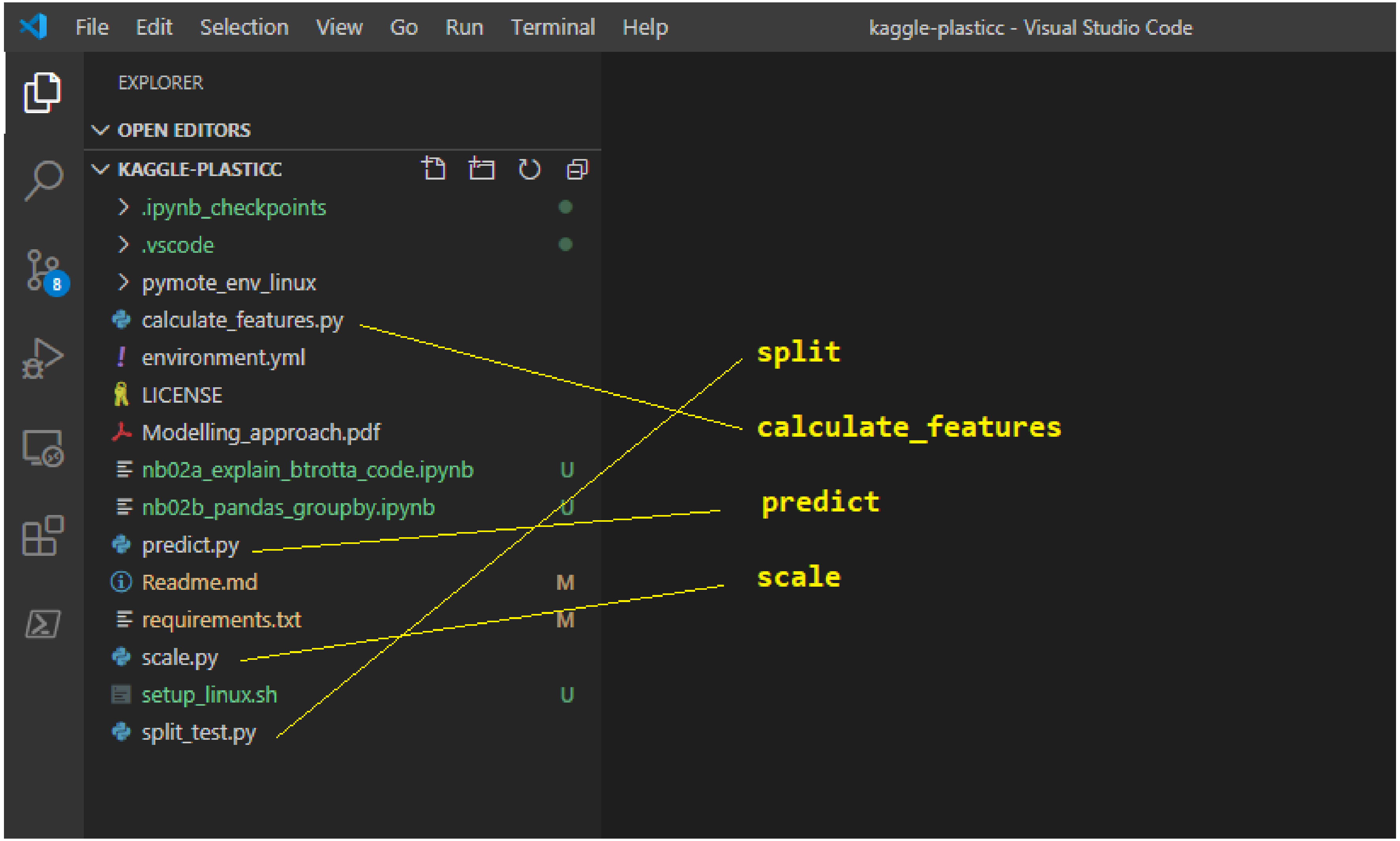
# If you want to follow along..

- A Jupyter Notebook is available that "code walks" B.Trotta. To use this a git of B.Trotta is populated within the cwinsor git workarea.

- Git clone https://github.com/cwinsor/kaggle_plasticc.git
- cd kaggle_plasticc/code_kaggle_plasticc_btrotta/
- You will see
  - Jupyter notebook NB99_EXPLAIN_BTROTTA_CODE_WALK.ipynb
  - requirments.txt
  - setu_linux_btrotta.sh
  - Use them
- From there, git clone https://github.com/btrotta/kaggle-plasticc.git
- Run the Jupyter Notebook – you will find it uses files in the B.Trotta sub-git. It should work!

# Procedure

Parts 1,2,3,4



# Part 1: split

split_test.py > ...

```python
"""Split the test data into chunks."""

import numpy as np
import pandas as pd
import os

n_chunks = 100
test = pd.read_csv(os.path.join('data', 'test_set.csv'), dtype=col_dict)
test.sort_values('object_id', inplace=True)
test = test.reset_index()
test_len = len(test)
id_diff = test.loc[test['object_id'].diff() != 0].index
chunk_starts = [id_diff[int(len(id_diff) * i / n_chunks)] for i in range(n_chunks)]
for i in range(n_chunks):
    if i == n_chunks - 1:
        end = len(test)
    else:
        end = chunk_starts[i + 1]
    test.iloc[chunk_starts[i]: end - 1].to_hdf(os.path.join('data', 'split_{}'.format(n_chunks),
                                                            'chunk_{}.hdf5'.format(i)), key='file0')

```

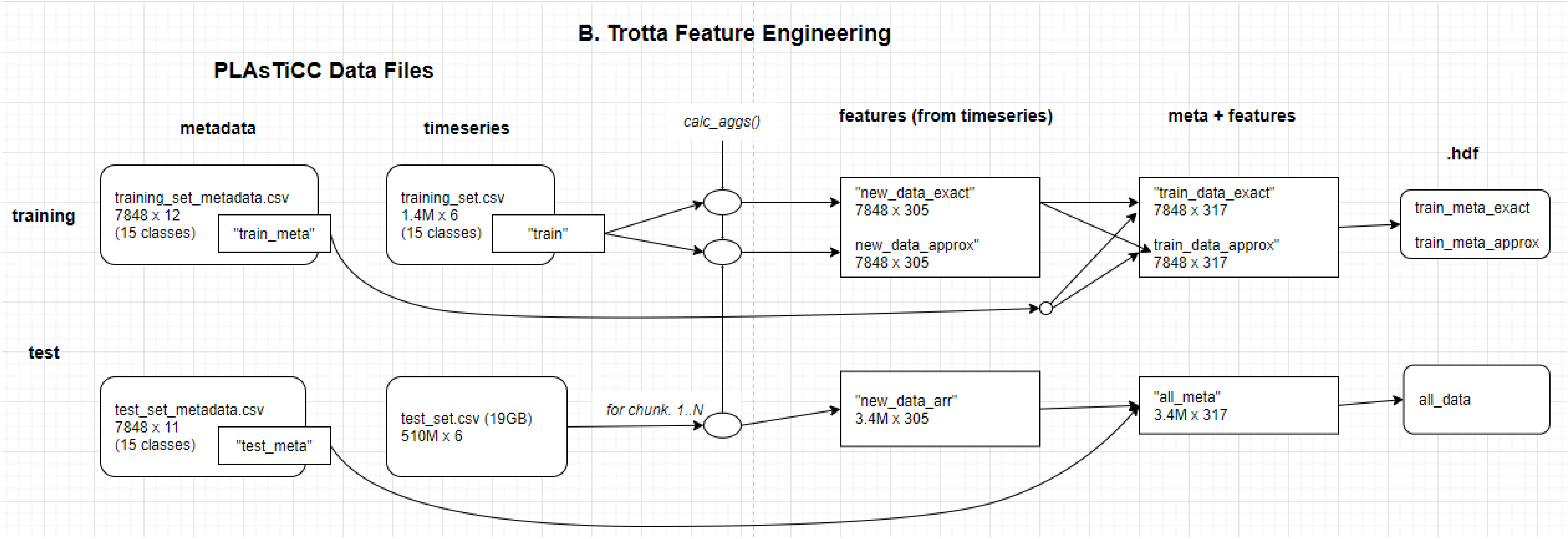
# Part 2: calculate_features

a.k.a. “feature engineering”

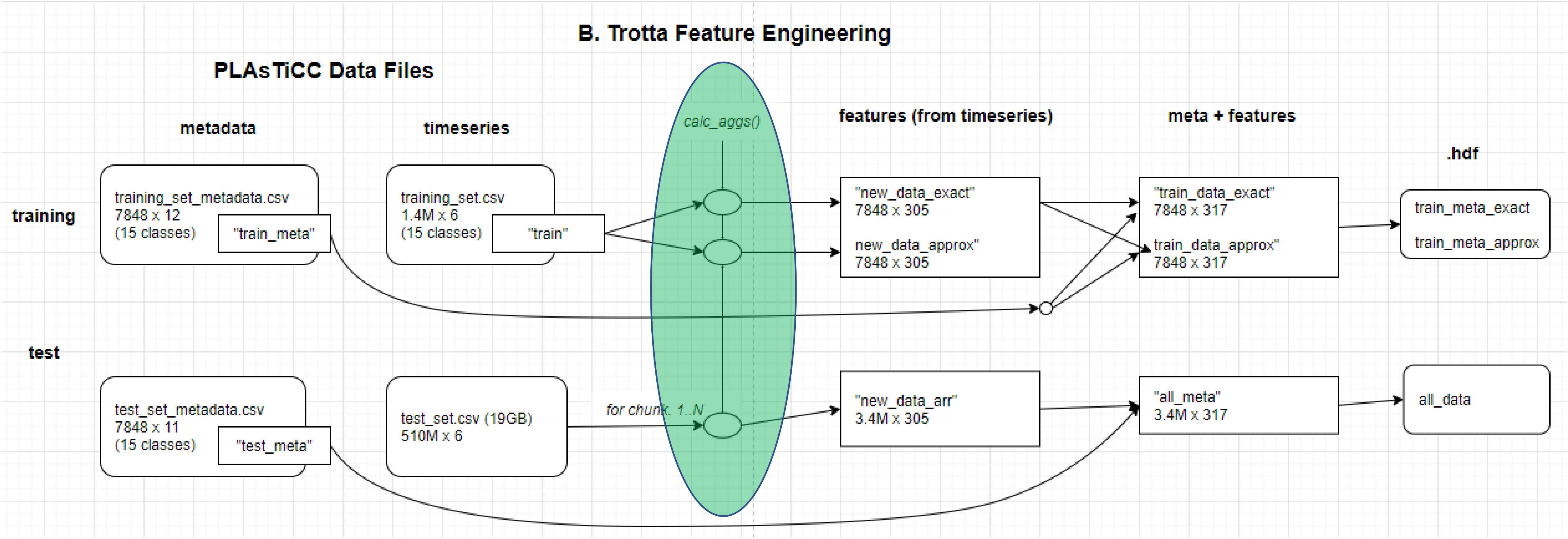
- For chunk 1..N
  - get data from file
  - calc_aggs()
- Merge w/metadat
- Write .hdf

```python
# get the metadata
test_meta = pd.read_csv(os.path.join('data', 'test_set_metadata.csv'))
all_meta = pd.concat([train_meta, test_meta], axis=0, ignore_index=True, sort=True).reset_index()
all_meta.drop('index', axis=1, inplace=True)
n_chunks = 100

# calculate features
new_data_exact = calc_aggs(train.copy(), True)
new_data_approx = calc_aggs(train.copy(), False)
train_meta_exact = pd.merge(train_meta, new_data_exact, 'left', left_on='object_id', right_index=True)
train_meta_approx = pd.merge(train_meta, new_data_approx, 'left', left_on='object_id', right_index=True)

# process training set (not actually used, just to get right shape of dataframe)
new_data_arr = []
new_data_arr.append(calc_aggs(train.copy(), True))
# process test set
for i in range(n_chunks):
    df = pd.read_hdf(os.path.join('data', 'split_{}'.format(n_chunks), 'chunk_{}.hdf5'.format(i)), key='file0')
    df.drop('index', axis=1, inplace=True)
    print('Read chunk {}'.format(i))
    new_data_arr.append(calc_aggs(df.copy(), True))
    print('Calculated features for chunk {}'.format(i))
del df
gc.collect()
new_data = pd.concat(new_data_arr, axis=0, sort=True)

# merge
all_meta = pd.merge(all_meta, new_data, 'left', left_on='object_id', right_index=True)

# write output
dir_name = 'features'
if not os.path.exists(os.path.join('data', dir_name)):
    os.mkdir(os.path.join('data', dir_name))
all_meta.to_hdf(os.path.join('data', dir_name, 'all_data.hdf5'), key='file0')
train_meta_exact.to_hdf(os.path.join('data', dir_name, 'train_meta_exact.hdf5'), key='file0')
train_meta_approx.to_hdf(os.path.join('data', dir_name, 'train_meta_approx.hdf5'), key='file0')
```

# as a picture...


B. Trotta Feature Engineering
PLAsTiCC Data Files
metadata
timeseries
calc_aggs()
features (from timeseries)
meta + features
.hdf
training
training_set_metadata.csv
7848 x 12
(15 classes)
"train_meta"
training_set.csv
1.4M x 6
(15 classes)
"train"
"new_data_exact"
7848 x 305
new_data_approx"
7848 x 305
"train_data_exact"
7848 x 317
train_data_approx"
7848 x 317
train_meta_exact
train_meta_approx
test
test_set_metadata.csv
7848 x 11
(15 classes)
"test_meta"
test_set.csv (19GB)
510M x 6
for chunk. 1..N
"new_data_arr"
3.4M x 305
"all_meta"
3.4M x 317
all_data

# all the work...


B. Trotta Feature Engineering
PLAsTiCC Data Files
metadata
timeseries
calc_aggs()
features (from timeseries)
meta + features
.hdf
training
training_set_metadata.csv
7848 x 12
(15 classes)
"train_meta"
training_set.csv
1.4M x 6
(15 classes)
"train"
"new_data_exact"
7848 x 305
new_data_approx"
7848 x 305
"train_data_exact"
7848 x 317
train_data_approx"
7848 x 317
train_meta_exact
train_meta_approx
test
test_set_metadata.csv
7848 x 11
(15 classes)
"test_meta"
test_set.csv (19GB)
510M x 6
for chunk: 1..N
"new_data_arr"
3.4M x 305
"all_meta"
3.4M x 317
all_data

# 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

Pandas Gold

- 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(),   gd.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...

Pandas Gold

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.

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

get the value at that point

for each object, find the max distance from mean
idmax 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')
```

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)
```

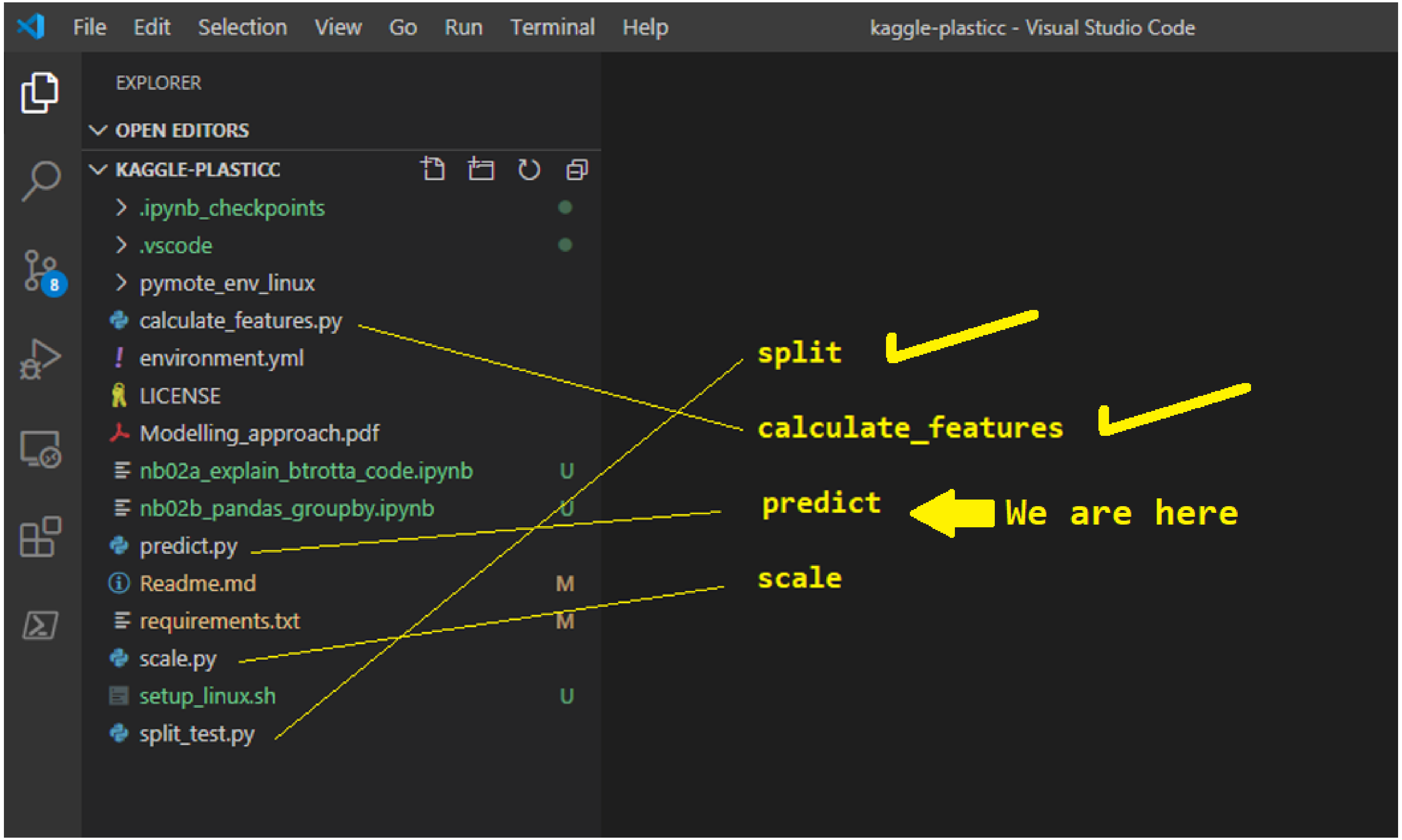
# Periodicity identification

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

```python
# 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']
```

# Where we are…

We engineered features.  Now create model using LGB.



# Part 3: Train
(in predict.py)

- This is where we:
  - Train
  - Validate
  - (optionally) save the model for later use in predicting
  - Make predictions

B.Trotta does not save the model.  For StarChaser we will need this.

# Train [1]

Read features data from file

Will be creating separate models (galactic, non-galactic) using “hostgal” as delimiter

```
import pandas as pd
import numpy as np
from sklearn import metrics, model_selection
import lightgbm as lgb
import os
```

```
test_mode = False
```

```
# read data
# warning - this may take 30 seconds or so (3.5M x 317)
all_meta = pd.read_hdf(os.path.join('all_data.hdf5'), key='file0')
train_meta_approx = pd.read_hdf(os.path.join('train_meta_approx.hdf5'), key='file0')
train_meta_exact = pd.read_hdf(os.path.join('train_meta_exact.hdf5'), key='file0')
```

```
print(all_meta.shape)
print(train_meta_approx.shape)
print(train_meta_exact.shape)
```

```
(3500738, 317)
(7848, 317)
(7848, 317)
```

# Train [2]

Map classes to integer range

```
# map classes to range [0, 14]
# Train separate models for galatic and extra-galactic, since these classes contain disjoint sets of objects,
# and can be distinguished by whether hostgl_photoz == 0, as observed here:
classes = np.sort(all_meta.loc[all_meta['target'].notnull(), 'target'].unique().astype(int))
galactic_bool = all_meta['hostgal_photoz'] == 0
galactic_classes = np.sort(     all_meta.loc[all_meta['target'].notnull() & galactic_bool, 'target'].unique().astype(int))
non_galactic_classes = np.sort( all_meta.loc[all_meta['target'].notnull() & ~galactic_bool, 'target'].unique().astype(int))
```

```
print(classes)
print(galactic_classes)
print(non_galactic_classes)
```

```
[ 6 15 16 42 52 53 62 64 65 67 88 90 92 95]
[ 6 16 53 65 92]
[15 42 52 62 64 67 88 90 95]
```

# Train [3]

Prepare target_trans_ for galactic, non-galactic models

```
# transform the target so the classes are the integers range(num_classes)
# CW: I removed "all_meta" to avoid long calculation time - we will only have train_meta_* to work with
#for df in [all_meta, train_meta_approx, train_meta_exact]:
for df in [train_meta_approx, train_meta_exact]:
    df['target_trans'] = np.nan
    df['target_trans_galactic'] = np.nan
    df['target_trans_non_galactic'] = np.nan
    for k, class_list in enumerate([classes, galactic_classes, non_galactic_classes]):
        if k == 0:
            suffix = ''
        elif k == 1:
            suffix = '_galactic'
        else:
            suffix = '_non_galactic'
        for i in range(len(class_list)):
            df.loc[df['target'] == class_list[i], 'target_trans' + suffix] = i
```

```
print(all_meta.shape)
print(train_meta_approx.shape)
print(train_meta_exact.shape)
```

```
(3500738, 317)
(7848, 320)
(7848, 320)
```

```
train_meta_approx[['object_id','target','target_trans','target_trans_galactic','target_trans_non_galactic']].head(10)
```

| | object_id | target | target_trans | target_trans_galactic | target_trans_non_galactic |
|---|---|---|---|---|---|
| **0** | 615 | 92 | 12.0 | 4.0 | NaN |
| **1** | 713 | 88 | 10.0 | NaN | 6.0 |
| **2** | 730 | 42 | 3.0 | NaN | 1.0 |
| **3** | 745 | 90 | 11.0 | NaN | 7.0 |
| **4** | 1124 | 90 | 11.0 | NaN | 7.0 |
| **5** | 1227 | 65 | 8.0 | 3.0 | NaN |
| **6** | 1598 | 90 | 11.0 | NaN | 7.0 |
| **7** | 1632 | 42 | 3.0 | NaN | 1.0 |
| **8** | 1920 | 90 | 11.0 | NaN | 7.0 |

# Train [4]

## Choose columns to be used in training

```
### note from CW: the author excludes columns that are assumed not good predictors to save computes
### also exclude 'target' and 'target_trans*' which are CLASS not FEATURES

# train 2 models for each class, one for when we have exact redshift, and another for when we don't
train_cols_exact_redshift \
    = [c for c in train_meta_exact.columns if
      c not in ['object_id', 'ra', 'decl', 'gal_l', 'gal_b', 'target', 'target_trans', 'target_trans_galactic',
                'target_trans_non_galactic', 'ddf',
                'distmod', 'mwebv', 'hostgal_photoz', 'hostgal_photoz_err', 'index']]
train_cols_approx_redshift \
    = [c for c in train_meta_approx.columns if
      c not in ['object_id', 'ra', 'decl', 'gal_l', 'gal_b', 'target', 'target_trans', 'target_trans_galactic',
                'target_trans_non_galactic', 'ddf',
                'distmod', 'mwebv', 'hostgal_specz', 'index']]
```

```
# separate parameters for galactic and non-galactic
params_galactic = {'boosting_type': 'gbdt', 'application': 'binary', 'num_leaves': 32, 'seed': 0, 'verbose': -1,
                   'min_data_in_leaf': 1, 'bagging_fraction': 0.8, 'bagging_freq': 1, 'lambda_l1': 0, 'lambda_l2': 1,
                   'learning_rate': 0.02}
params_non_galactic = {'boosting_type': 'gbdt', 'application': 'binary', 'num_leaves': 16, 'seed': 0, 'verbose': -1,
                       'min_data_in_leaf': 1, 'bagging_fraction': 0.8, 'bagging_freq': 1, 'lambda_l1': 0,
                       'lambda_l2': 1, 'learning_rate': 0.02}
```

```
num_rounds = 3000
```

```
len(train_cols_exact_redshift)
```

306

# Train [5]

Prepare cross-validation infrastructure.
This uses sklearn (Scikit-learn) model_selection.Kfold
nfolds=5

```
# cross-validate on train set, and measure distribution of out-of-sample predicted values

# CW Note: author is setting up validation infrastructure - 5-fold validation
# there are 2 models: galactic ("exact") and extra-galactic ("approx")
# there are 14 predictions made

train_err_exact = []
test_err_exact = []
train_err_approx = []
test_err_approx = []
cv = model_selection.KFold(5, shuffle=True, random_state=4)
galactic_bool_train = train_meta_exact['hostgal_photoz'] == 0
train_meta_exact['predict_max_exact'] = 0
train_meta_exact['predict_max_approx'] = 0
train_meta_approx['predict_max_exact'] = 0
train_meta_approx['predict_max_approx'] = 0
predict_cols = ['class_' + str(c) for c in classes]
train_prediction_exact \
    = pd.DataFrame(np.zeros((len(train_meta_exact), 14)), index=train_meta_exact.index, columns=predict_cols)
train_prediction_approx \
    = pd.DataFrame(np.zeros((len(train_meta_exact), 14)), index=train_meta_exact.index, columns=predict_cols)
eval_prediction_exact \
    = pd.DataFrame(np.zeros((len(train_meta_exact), 14)), index=train_meta_exact.index, columns=predict_cols)
eval_prediction_approx \
    = pd.DataFrame(np.zeros((len(train_meta_exact), 14)), index=train_meta_exact.index, columns=predict_cols)
importance = {}
best_iter_exact = {c: [] for c in classes}
best_iter_approx = {c: [] for c in classes}
```

# Train (6)

- The training set is not representative of the test set.
- Author resamples training set to reflect test.

```
# Evaluate accuracy on resampled training set having similar distribution to test. The data note says
# "The training data are mostly composed of nearby, low-redshift, brighter objects while the test data contain
# more distant (higher redshift) and fainter objects."  So we resample to achieve a similar distribution of
# hostgal_photoz.

# CW Note: Author is resampling so training set distribution is similar to
# test distribution. I did not drill down to verify this code.

train_bool = all_meta['target'].notnull()
ddf = all_meta['ddf'] == 1
w = pd.DataFrame(index=train_meta_exact.index, columns=['galactic', 'non_galactic'])
w['galactic'] = galactic_bool_train.astype(int)
w['non_galactic'] = np.nan
bands = np.arange(all_meta.loc[~train_bool, 'hostgal_photoz'].min(),
                  all_meta.loc[~train_bool, 'hostgal_photoz'].max() + 0.00001, 0.1)
for i in range(len(bands[:-1])):
    band_bool = ~galactic_bool_train & ~ddf & (train_meta_exact['hostgal_photoz'] >= bands[i]) \
            & (train_meta_exact['hostgal_photoz'] <= bands[i + 1])
    train_prop = band_bool.sum() / (~galactic_bool_train & ~ddf).sum()
    test_prop = ((all_meta.loc[~train_bool & ~galactic_bool, 'hostgal_photoz'] >= bands[i])
             & (all_meta.loc[~train_bool & ~galactic_bool, 'hostgal_photoz'] <= bands[i + 1])).sum() \
            / (~train_bool & ~galactic_bool).sum()
    w.loc[band_bool, 'non_galactic'] = test_prop / train_prop
w.loc[ddf] = 0
```

# Train [7]

(train and evaluate)
library = lgb (lightGBM by Microsoft)

A "cv" was previously created

For each train/test split and for each class:

prepare lgb dataset (training and validation)

train model "est"
from the iterations, take the best and append to best_iter[c]

make prediction using test data ("train_prediction")

evaluate performance

```
for train_ind, test_ind in list(cv.split(train_meta_exact.index, train_meta_exact['target_trans'])):
    train_bool = train_meta_exact.index.isin(train_ind)
    ddf = train_meta_exact['ddf'] == 1

    for i, c in enumerate(classes):
        g = c in galactic_classes
        gal_bool_train_curr = galactic_bool_train == g
        params = params_galactic if g else params_non_galactic
        col = 'class_' + str(c)
        weight_col = 'galactic' if g else 'non_galactic'

        # exact redshift model
        lgb_train = lgb.Dataset(train_meta_exact.loc[train_bool & gal_bool_train_curr, train_cols_exact_redshift],
                                label=(train_meta_exact.loc[train_bool & gal_bool_train_curr, 'target'] == c).astype(int))
        lgb_valid = lgb.Dataset(train_meta_exact.loc[(~train_bool) & gal_bool_train_curr & ~ddf, train_cols_exact_redshift],
                                label=(train_meta_exact.loc[(~train_bool) & gal_bool_train_curr & ~ddf, 'target'] == c).astype(ir
                                weight=w.loc[(~train_bool) & gal_bool_train_curr & ~ddf, weight_col])
        est = lgb.train(train_set=lgb_train, valid_sets=[lgb_train, lgb_valid], valid_names=['train', 'valid'],
                        params=params, num_boost_round=num_rounds, early_stopping_rounds=100)
        best_iter_exact[c].append(est.best_iteration)
        train_prediction_exact.loc[~train_bool & gal_bool_train_curr, col] = est.predict(
            train_meta_exact.loc[(~train_bool) & gal_bool_train_curr, train_cols_exact_redshift],
            num_iteration=est.best_iteration)
        # measure errors on train and test
        eval_prediction_exact.loc[gal_bool_train_curr, col] \
            = est.predict(train_meta_exact.loc[gal_bool_train_curr, train_cols_exact_redshift],
                          num_iteration=est.best_iteration)
```

```
[1]	train's binary_logloss: 0.674	valid's binary_logloss: 0.675019
Training until validation scores don't improve for 100 rounds.
[2]	train's binary_logloss: 0.655598	valid's binary_logloss: 0.657552
[3]	train's binary_logloss: 0.637851	valid's binary_logloss: 0.640828
[4]	train's binary_logloss: 0.620838	valid's binary_logloss: 0.62469
[5]	train's binary_logloss: 0.604492	valid's binary_logloss: 0.609301
[6]	train's binary_logloss: 0.588657	valid's binary_logloss: 0.594182
[7]	train's binary_logloss: 0.573377	valid's binary_logloss: 0.579447
[8]	train's binary_logloss: 0.558579	valid's binary_logloss: 0.565713
[9]	train's binary_logloss: 0.544272	valid's binary_logloss: 0.55225
[10]	train's binary_logloss: 0.530479	valid's binary_logloss: 0.539341
[11]	train's binary_logloss: 0.517194	valid's binary_logloss: 0.527043
```

# 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