



pysat : A Bridge Between Worlds

Russell A. Stoneback
Center for Space Sciences, University of Texas at Dallas
Photon Audio Research Lab

Angeline G. Burrell
Naval Research Lab, Space Science Division

Jeff Klenzing
NASA GSFC

Including Jonathon Smith - Catholic Univ. America



June 2021
Snakes on a Spaceship
CEDAR

- Large variety of data sets in space science
 - Different data formats
 - Different metadata standards
 - Range of dimensionality
 - Variety of data sources
 - Variety of data cleaning standards
 - Variety of supporting data standards
- Lack of specific transparency and validation of code used for research papers
- Solutions to these problems are usually institution/research group specific
 - Historically, these software solutions are not shared
- pysat abstracts away all of these tedious details leaving more time for science!



Python Satellite Data Analysis Toolkit

- PySat implements the general process of space science data analysis
 - Provides a simple, flexible, and consistent interface to any dataset
 - Instrument independent data, metadata, and processing support
 - Design evolved over ~10 years
 - Contains solutions to ~all of the processing problems the group has encountered so far
- Package with support for common problems such as
 - Downloading
 - Organizing Files
 - Loading
 - Cleaning
 - Modifying/Processing
 - Exploit routines from other packages
 - Instrument specific analysis
 - Instrument independent analysis

AGU100 ADVANCING EARTH AND SPACE SCIENCE

JGR

Journal of Geophysical Research: Space Physics

TECHNICAL REPORTS: METHODS

10.1029/2018JA025297

Key Points:

PYSAT: Python Satellite Data Analysis Toolkit

R. A. Stoneback¹, A. G. Burrell¹, J. Klenzing², and M. D. Depew¹

¹W. B. Hanson Center for Space Sciences, Physics Department, University of Texas at Dallas, Richardson, TX, USA,

²NASA Goddard Space Flight Center, Greenbelt, MD, USA



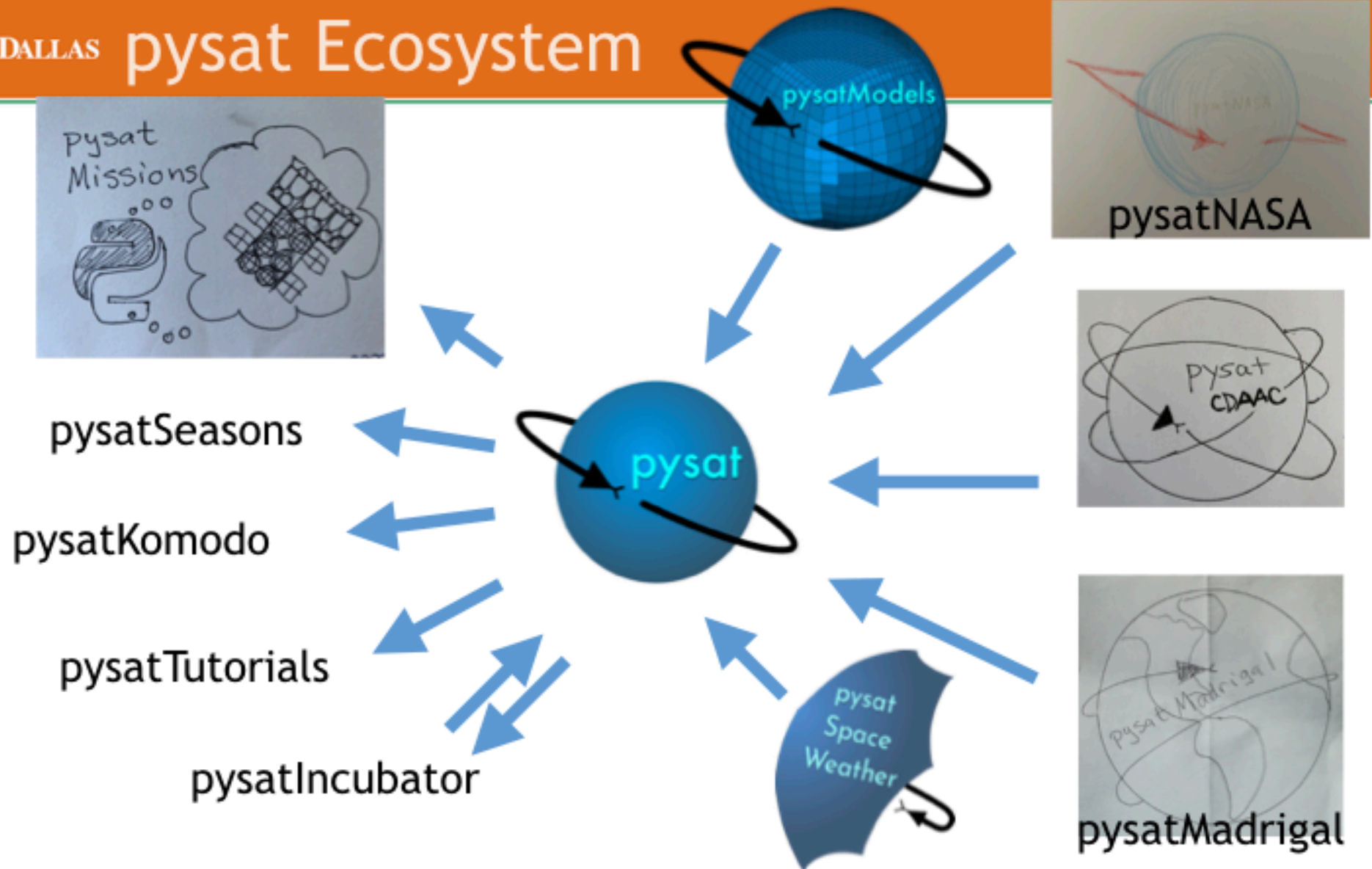
Volume 123, Issue 6

June 2018

Pages 5271-5283



pysat Ecosystem





pysat Ecosystem

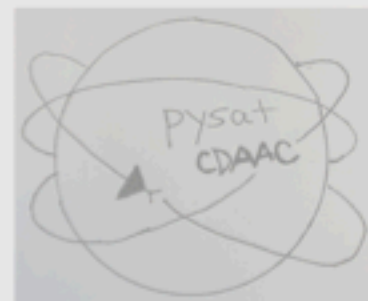


pysatSeasons

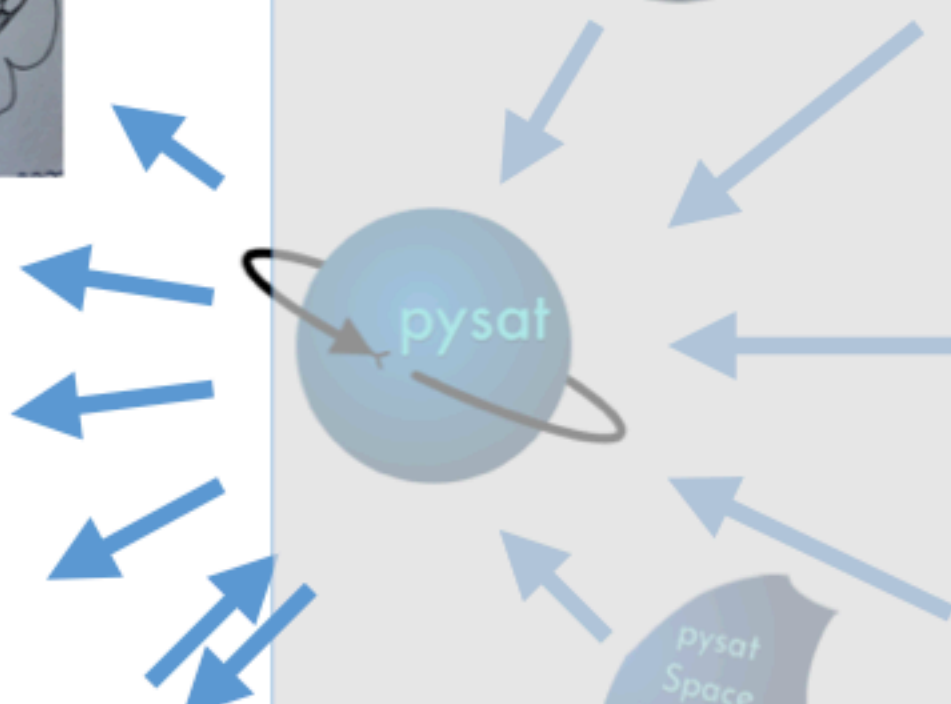
pysatKomodo

pysatTutorials

pysatIncubator

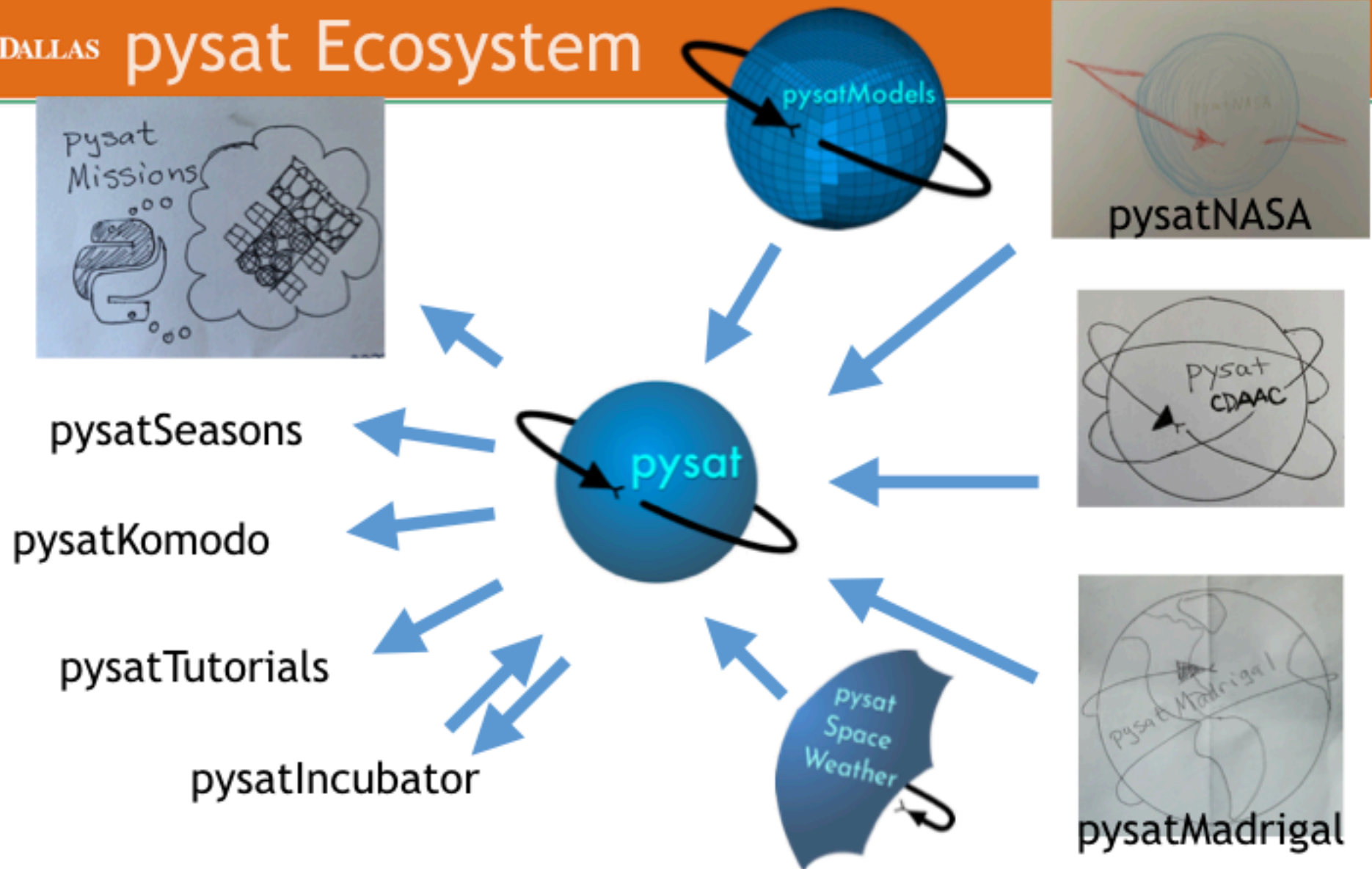


Data Source
Plugins





pysat Ecosystem



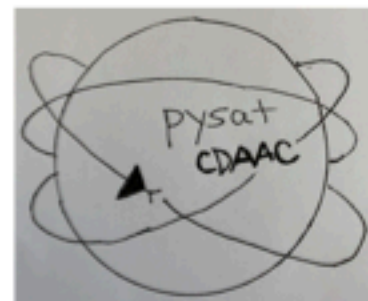
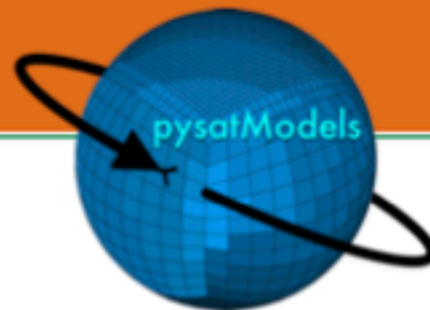
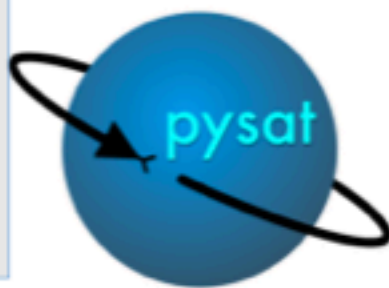
'Use'
Packages



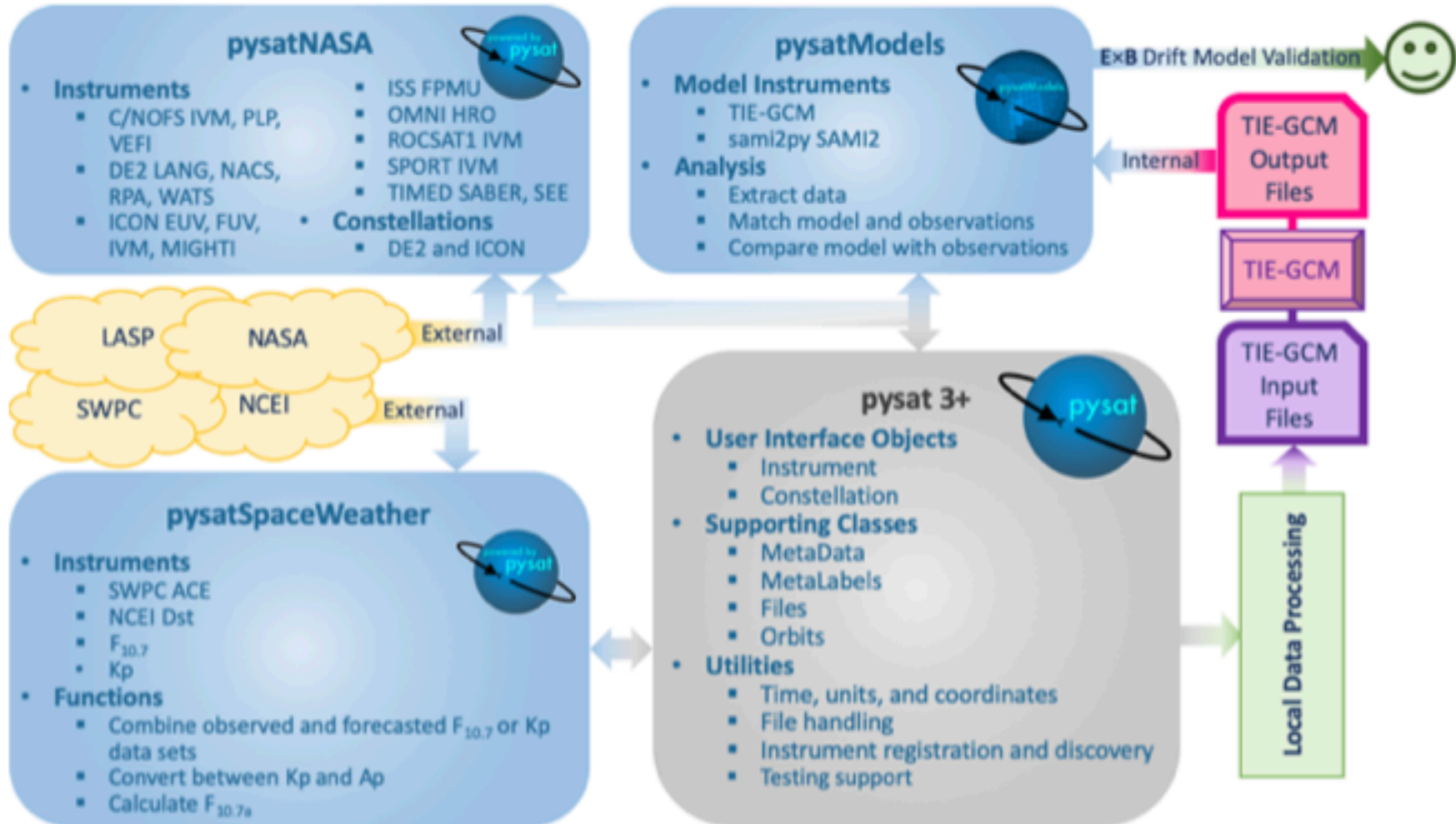
pysatSeasons
pysatKomodo

pysatTutorials

pysatIncubator

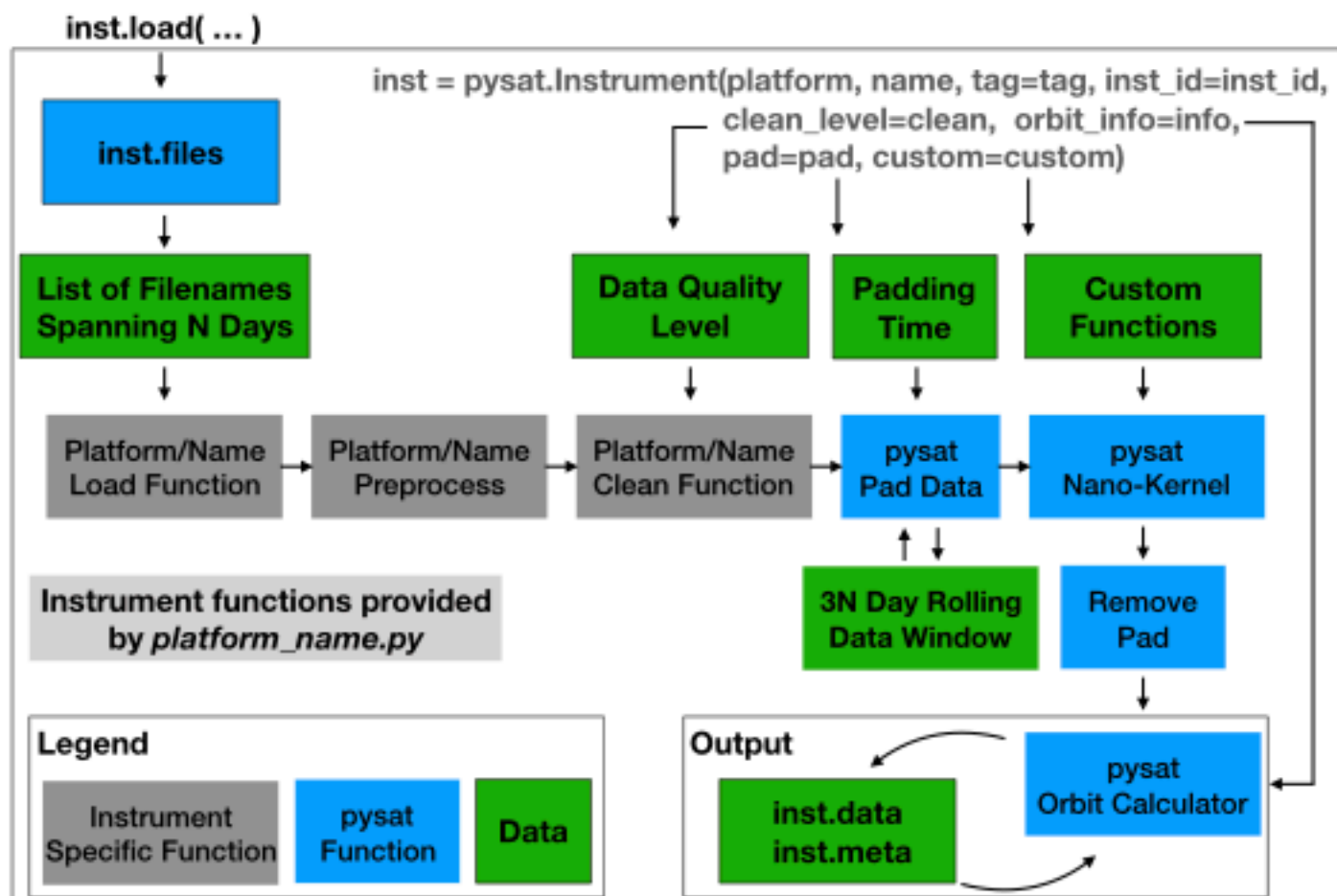


Example Workflow



- Instrument object has built-in data flow
- Supports arbitrary relationships between loaded data and data on disk
- Supports robust and testable processing
- Supports robust and testable analysis

pysat Loading Process and Data Flow



- Processes measurements from the Ion Velocity Meter on
 - NASA ICON
 - NOAA/NSPO COSMIC-2
 - NASA SORTIE CubeSat
 - pysat's `.to_netCDF` used to create public data files
- Used to aggregate data as part of DARPA funded project at NRL to produce next generation space weather predictions
- Used by pysat team for science

- pysat provides an ideal platform for CubeSat and other satellite missions to process, manage, distribute, and analyze data and makes it easy to contribute any improvements back to the community

- pysat is intended to support the development of higher level packages that use pysat as the primary interface for data
 - Requires high code standards
 - All pysat code undergoes review and unit testing
 - v3.0 has ~96% test coverage
 - Documentation, including tutorials, API, etc...
 - Supports pysat.Instruments outside the main package
 - Use and testing
 - Thread and process safe
 - Uses logging
 - Reduced installation requirements (no pysatCDF!)

build passing

docs passing

coverage 96%

DOI [10.5281/zenodo.4660310](https://doi.org/10.5281/zenodo.4660310)



Lots of Documentation!

pysat
latest

Search docs

- Introduction
- Ecosystem
- Citations in the pysat ecosystem
- Installation
- Quick-Start
- Tutorial
 - Transition to v3.0
 - Basics
 - Parameters
 - Verbosity
 - Data Loading
 - Iteration
 - Orbit Support
 - Custom Functions
 - Analysis
- Examples
 - Adding a New Instrument
 - Testing Support
 - Using pysat as a dependency
 - API
 - Contributing
 - Frequently Asked Questions
 - Release Notes

Docs » Tutorial

Tutorial

- [Transition to v3.0](#)
- [Basics](#)
 - [Instrument Discovery](#)
 - [Standard Workflow](#)
 - [Simple Workflow](#)
- [Parameters](#)
- [Verbosity](#)
- [Data Loading](#)
 - [Load Data](#)
 - [Load Data Range](#)
 - [Clean Data](#)
- [Iteration](#)
 - [Daily Iteration](#)
 - [Orbit Iteration](#)
 - [Non-standard Iteration](#)
- [Orbit Support](#)
 - [Ground-Based Instruments](#)
- [Custom Functions](#)
 - [Example Function](#)
 - [Attaching Custom Function](#)
 - [Attaching Custom Function at Instantiation](#)
- [Analysis](#)
 - [Sample Period Mean Function](#)

◀ Previous

pysat
latest

Search docs

- Introduction
- Ecosystem
- Citations in the pysat ecosystem
- Installation
- Quick-Start
- Tutorial
- Examples
- Adding a New Instrument
 - Instrument Libraries
 - Naming Conventions
 - Required Attributes
 - Required Routines
 - Optional Attributes
 - Optional Routines and Support
 - Logging
- Testing Support
- Using pysat as a dependency
- API
- Contributing
- Frequently Asked Questions
- Release Notes

Docs » Adding a New Instrument

[Edit on GitHub](#)

Adding a New Instrument

pysat works by calling modules written for specific instruments that load and process the data consistent with the pysat standard. The name of the module corresponds to the combination 'platform_name' provided when initializing a pysat instrument object. The module should be placed in the pysat instruments directory or registered (see below) for automatic discovery. A compatible module may also be supplied directly using

```
pysat.Instrument(inst_module=python_module_object)
```

A general template has also been included to make starting any instrument module easier at `pysat/instruments/templates/`. Some data repositories have pysat templates prepared to assist in integrating a new instrument. See the associated pysat package for that particular data source, such as `pysatNASA` for supporting additional NASA instruments.

External modules may be registered as part of pysat's user instrument registry using the following syntax:

```
from pysat.utils import registry

# Register single instrument
registry.register('my_package.myInstrument')

# Register all instrument sub-modules
import my_package
registry.register_from_module(my_package, instruments)
```

After registry the instrument module name is stored in the user's home directory in a hidden directory named `~/.pysat`. The instrument may then be instantiated with the instrument's platform and name:

```
inst = Instrument('myPlatform', 'myname')
```

pysat

latest

Search docs

- Introduction
- Ecosystem
- Citations in the pysat ecosystem
- Installation

Quick-Start

- Set the Data Directory
- Load an Instrument
- Explore the Possibilities

Tutorial

- Examples
- Adding a New Instrument
- Testing Support
- Using pysat as a dependency

API

- Contributing
- Frequently Asked Questions
- Release Notes

Get visibility into your Python apps with detailed metrics and end-to-end traces. Try Datadog free!

Sponsored - Ads served ethically

```
print(pysat.params['data_dirs'])
```

Load an Instrument

The best way to see if pysat is working is to load a test instrument. The test instrument will simulate data when it is asked to load data. Loading a day of data will ensure there is no problem with the underlying pandas and xarray installations.

```
# Testing out the pandas installation
inst = pysat.Instrument('pysat', 'testing')
inst.load(2009, 1)
print(inst.data)

# Testing out the xarray installation
inst = pysat.Instrument('pysat', 'testing_xarray')
inst.load(2009, 1)
print(inst.data)
```

Note

pysat will not allow any instrument to be instantiated without a data directory being specified.

Note

Test instruments have a limited date range over which they will simulate data.

Explore the Possibilities

At this point, you are set up to start exploring the tools and methods pysat provides. To start working with some instruments that manage real space science data, check out the [pysat Ecosystem](#). For a more detailed dive into pysat, check out the [Tutorial](#) and [Examples](#).

Previous

Next

pysat

latest

Search docs

- Introduction
- Ecosystem
- Citations in the pysat ecosystem
- Installation
- Quick-Start
- Tutorial
- Examples
- Adding a New Instrument
- Testing Support
- Using pysat as a dependency

API

- Instrument
- Constellation
- Files
- Meta
- MetaLabels
- Orbits
- Parameters
- Instrument Methods
- Utilities
- Core Utilities
- Coordinates
- Files
- Registry
- Time
- Testing

Read the Docs

latest

```
custom_func_3 = {'function': custom_func, 'at_pos': 0,
                'kwargs': {'start': True}}

# Combine all dicts into a list in order of application and execution,
# although this can be modified by specifying 'at_pos'. The actual
# order these functions will run in: 1, 1, 2
custom = [custom_func_1, custom_func_2, custom_func_3]

# Instantiate pysat.Instrument
inst = pysat.Instrument(platform, name, inst_id=inst_id, tag=tag,
                       custom=custom)
```

bounds

Boundaries for iterating over instrument object by date or file.

- Parameters:
- start** (datetime object, filename, or None) - start of iteration, if None uses first data date. file collection also accepted. (default=None)
 - stop** (datetime object, filename, or None) - stop of iteration, inclusive. If None uses last data date. file collection also accepted. (default=None)
 - step** (int, int, or None) - Step size used when iterating from start to stop. Use a Pandas frequency string ('3D', '3M') when setting bounds by date, an integer when setting bounds by file. Defaults to a single day/file (default='1D', 1).
 - width** (pandas.DateOffset, int, or None) - Data window used when loading data within iteration. Defaults to a single day/file if not assigned. (default=dt.timedelta(days=1), 1)

Note

Both start and stop must be the same type (date, or filename) or None. Only the year, month, and day are used for date inputs.

Examples

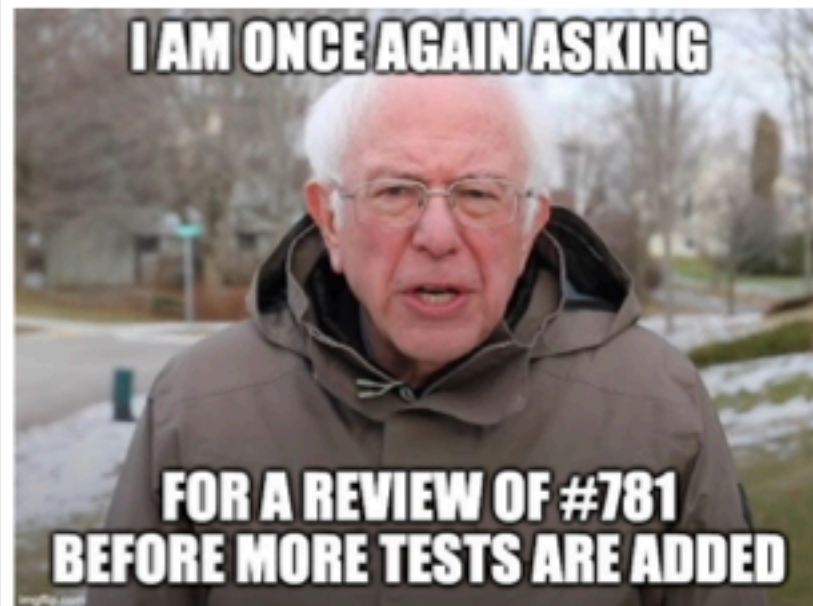
```
import datetime as dt
import pandas as pd
import pysat

inst = pysat.Instrument(platform=platform,
                       name=name,
                       tag=tag)
start = dt.datetime(2009, 1, 1)
stop = dt.datetime(2009, 1, 31)

# Defaults to stepping by a single day and a data loading window
# of one day/file.
```

- pysat is open-source and open for contributions
- pysat penumbra packages would benefit from additional instruments
 - Adding an instrument is easy!*
 - General methods already exist for a given data provider
 - Template and examples are provided
 - Many instruments only require trivial updates
 - Proper development of instrument data cleaning functions requires knowledge or investigation as to what constitutes data quality
 - Many data sets include quality flags!
- pysatSpaceWeather, pysatNASA, and pysatMadriral are great places to start

pysat Ecosystem: <https://github.com/pysat>



- pysat v3.0.1 Release Candidate 1 is out and pysat team open for feedback
<https://github.com/pysat/pysat/pull/821>
<https://test.pypi.org/project/pysat/3.0.1rc1/>
- pysat has the potential for an extremely broad impact, both within and outside of space science
 - The world is challenged by working with a variety of data sets, not just scientists
- Leaving UTDallas to start a research center, Stoneris
 - Enable more focus on research in general and pysat development in particular

Thank you! Questions?

Follow-on slides contain a variety of examples

- Instrument object is the primary user interface

```
import pysat
gps = pysat.Instrument('cosmic', 'gps', tag='ionprf')
```

- Instrument object includes standard functions for working with data
 - Download data
 - Manage files
 - Load data
 - Access data
 - Manage data customization

Identifier	Description
platform	Name of the platform supporting the instrument
name	Name of the instrument
tag	Label for an instrument data product
inst_id	Label for instrument sub-group

- Each Instrument maintains list of its files
 - Compare local list to server
 - Download files with newer version, revision, or cycle
 - Also supports downloading specific date range
-
- With pysat, users never have to ask, ‘How do I get the data?’ or ‘How do I load the data’ or ‘Is my data the most recent?’

```
In [10]: nmax = pysat.Instrument(platform='ses14', name='gold', tag='nmax')
pysat WARNING: Time stamps may be non-unique because Channel A and B are different instruments. An upgrade to the pysat.Constellation object is required to solve this issue. See pysat issue #614 for more info.

In [11]: nmax.files.files
Out[11]:
2018-10-05 gold_l2_nmax_2018_278_v01_r01_c01.nc
2018-10-06 gold_l2_nmax_2018_279_v01_r01_c01.nc
2018-10-07 gold_l2_nmax_2018_280_v01_r01_c01.nc
2018-10-09 gold_l2_nmax_2018_282_v01_r01_c01.nc
2018-10-10 gold_l2_nmax_2018_283_v01_r01_c01.nc
...
2020-12-27 gold_l2_nmax_2020_362_v01_r01_c02.nc
2020-12-28 gold_l2_nmax_2020_363_v01_r01_c02.nc
2020-12-29 gold_l2_nmax_2020_364_v01_r01_c02.nc
2020-12-30 gold_l2_nmax_2020_365_v01_r01_c02.nc
2020-12-31 gold_l2_nmax_2020_366_v01_r01_c02.nc
Length: 795, dtype: object

In [12]: nmax.download_updated_files()

In [13]: nmax.files.files
Out[13]:
2018-10-05 gold_l2_nmax_2018_278_v01_r01_c01.nc
2018-10-06 gold_l2_nmax_2018_279_v01_r01_c01.nc
2018-10-07 gold_l2_nmax_2018_280_v01_r01_c01.nc
2018-10-09 gold_l2_nmax_2018_282_v01_r01_c01.nc
2018-10-10 gold_l2_nmax_2018_283_v01_r01_c01.nc
...
2020-12-27 gold_l2_nmax_2020_362_v02_r02_c01.nc
2020-12-28 gold_l2_nmax_2020_363_v02_r02_c01.nc
2020-12-29 gold_l2_nmax_2020_364_v02_r02_c01.nc
2020-12-30 gold_l2_nmax_2020_365_v02_r02_c01.nc
2020-12-31 gold_l2_nmax_2020_366_v02_r02_c01.nc
Length: 817, dtype: object
```

```
import pysatMadriganal
pysat.utils.registry.register_by_module(pysatMadriganal.instruments)
vtec = pysat.Instrument('gnss', 'tec', 'vtec')
```

```
In [8]: vtec.load(date=dt.datetime(2017, 11, 18))
```

```
In [9]: vtec.data
```

```
Out[9]:
```

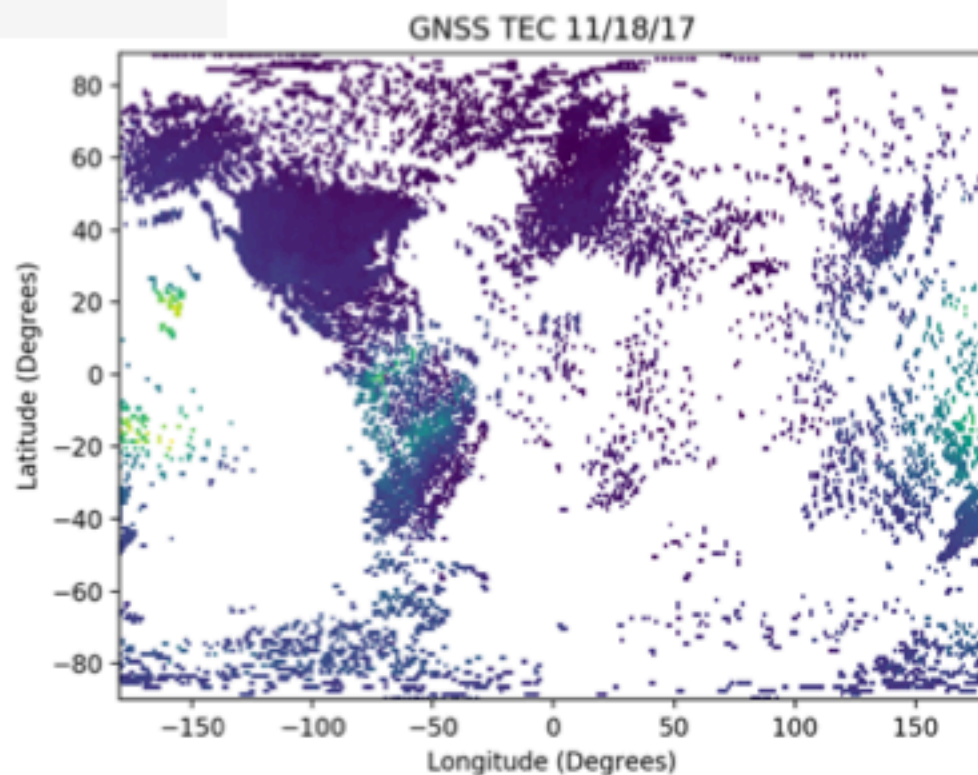
```
<xarray.Dataset>
```

```
Dimensions:  (gdalt: 1, gdlat: 180, glon: 360, time: 1)
```

```
Coordinates:
```

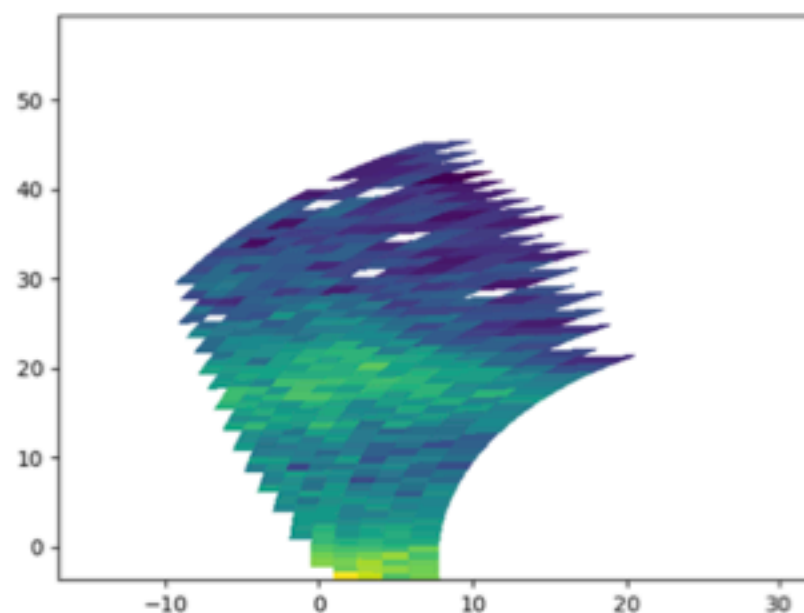
```
* time      (time) datetime64[ns] 2017-11-18T00:02:30.000000000
* gdlat     (gdlat) float64 -90.0 -89.0 -88.0 -87.0 -86.0 -85.0
* glon      (glon) float64 -180.0 -179.0 -178.0 -177.0 -176.0 -175.0
* gdalt     (gdalt) float64 350.0
```

```
Data variables:
```



- Supports loading by file, year/doy, date, as well as corresponding ranges (new in v3)

- Data called from the instrument object directly can be fed into matplotlib or other plotting routines



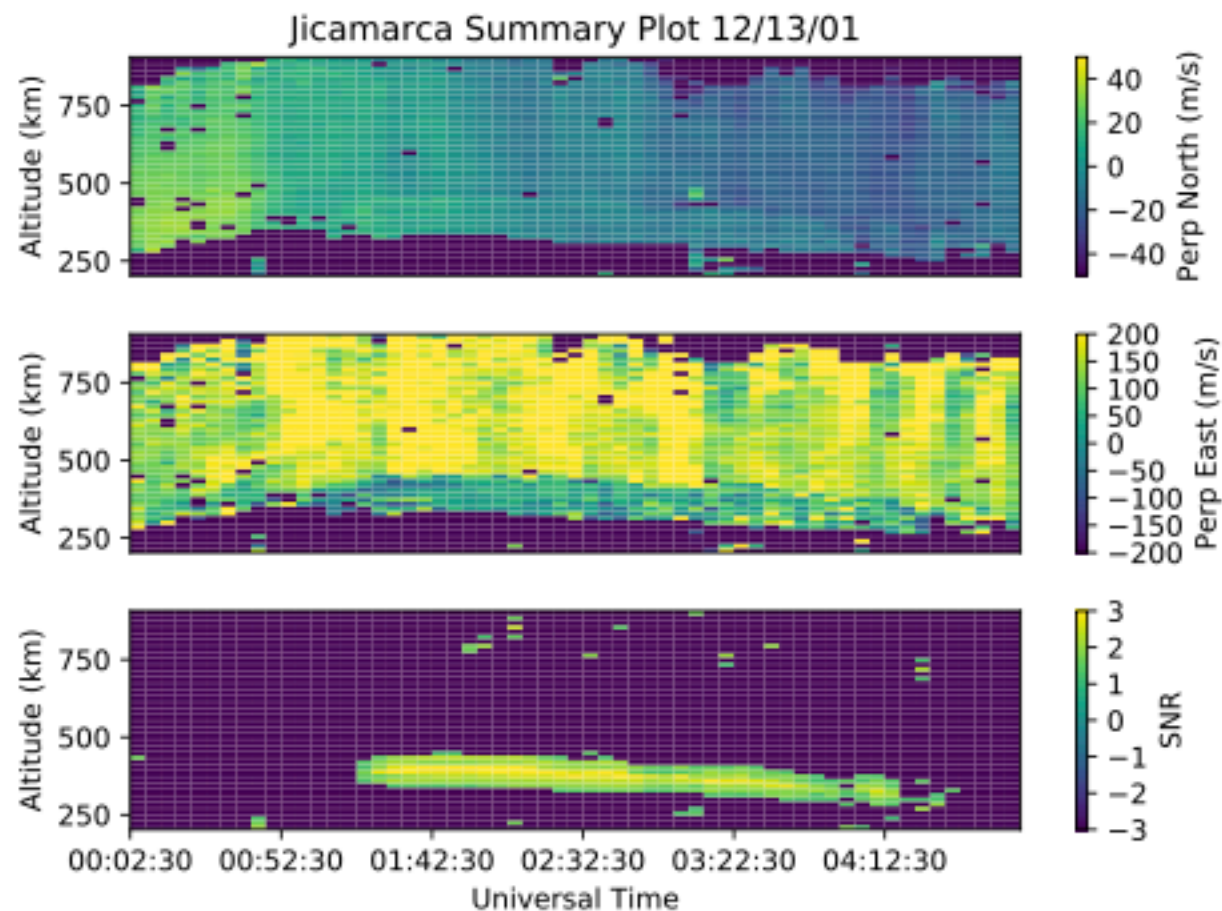
```
[In [36]: import matplotlib.pyplot as plt
```

```
[In [37]: plt.ion()
```

```
[In [38]: plt.pcolor(nmax['longitude'][0,:,:], nmax['latitude'][0,:,:], nmax['nmax'][0,:,:])
```

- Interfaces with Madrigal

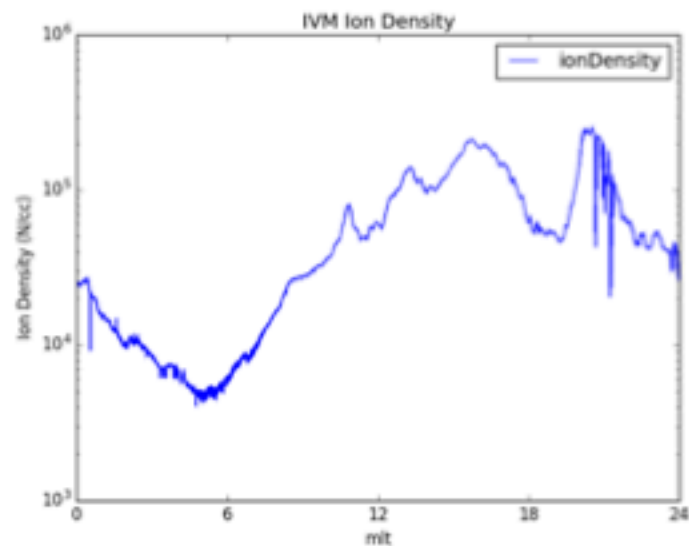
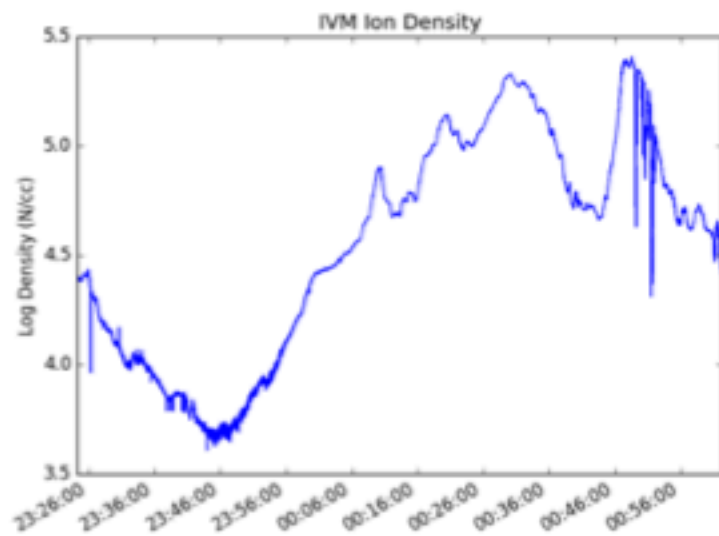
```
jro = pysat.Instrument('jro', 'isr', 'drifts')  
jro.download(date1, date2)  
jro.load(date=date1)  
custom_drifts_plot(jro)
```

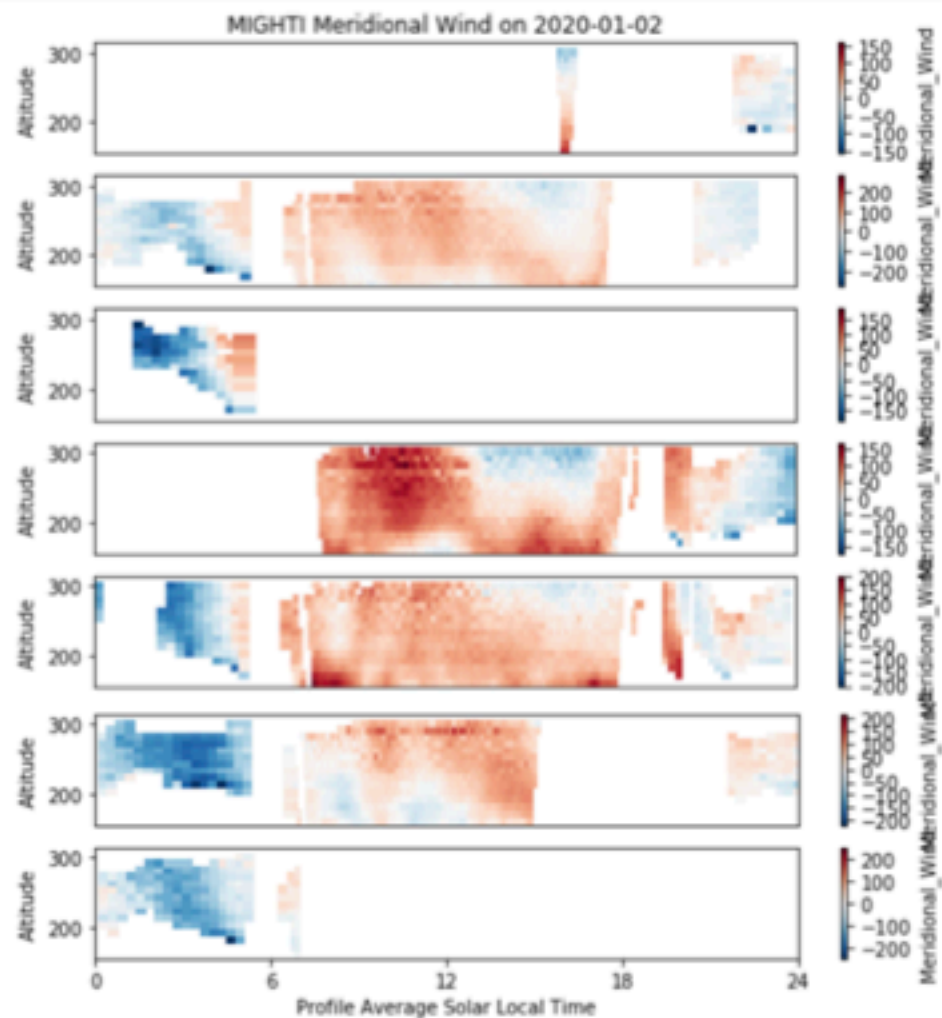
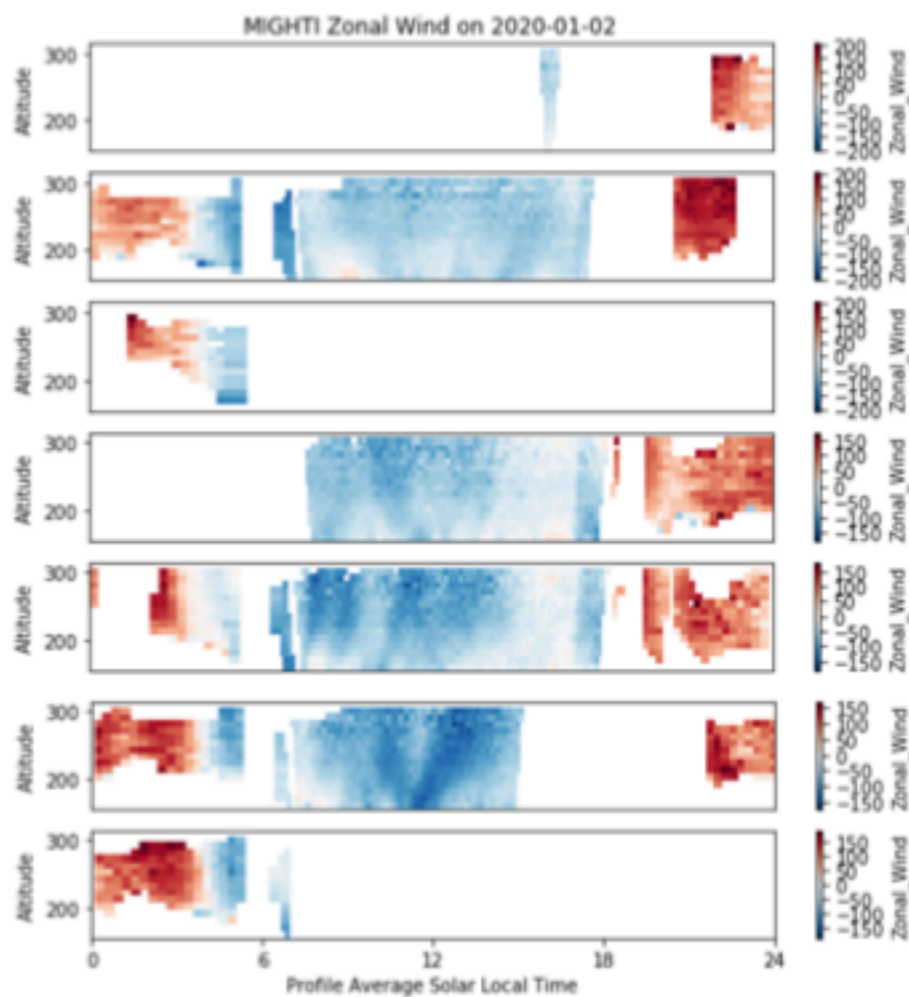


```
In [40]: ivm = pysat.Instrument('cnofs', 'ivm',
...:                             clean_level='dirty',
...:                             orbit_info={'index':'mlt'})
...: ivm.load(2010,2)
...: ivm.orbits.next()
...: np.log10(ivm['ionDensity']).plot(title='IVM Ion Density')
...: plt.ylabel('Log Density (N/cc)')
Returning cnofs ivm data for 01/02/10
Returning cnofs ivm data for 01/01/10
Returning cnofs ivm data for 01/02/10
Loaded Orbit:0
Out[40]: <matplotlib.text.Text at 0x1250fab90>
```

```
In [48]: ivm = pysat.Instrument('cnofs', 'ivm',
...:                             clean_level='dirty',
...:                             orbit_info={'index':'mlt'})
...: ivm.load(2010,2)
...: ivm.orbits.next()
...: ivm.data.plot(x='mlt', y='ionDensity',
...:               title='IVM Ion Density',
...:               logy=True,
...:               xticks=[0,6,12,18,24])
...: plt.ylabel('Ion Density (N/cc)')
Returning cnofs ivm data for 01/02/10
Returning cnofs ivm data for 01/01/10
Returning cnofs ivm data for 01/02/10
Loaded Orbit:0
Out[48]: <matplotlib.text.Text at 0x1284af410>
```

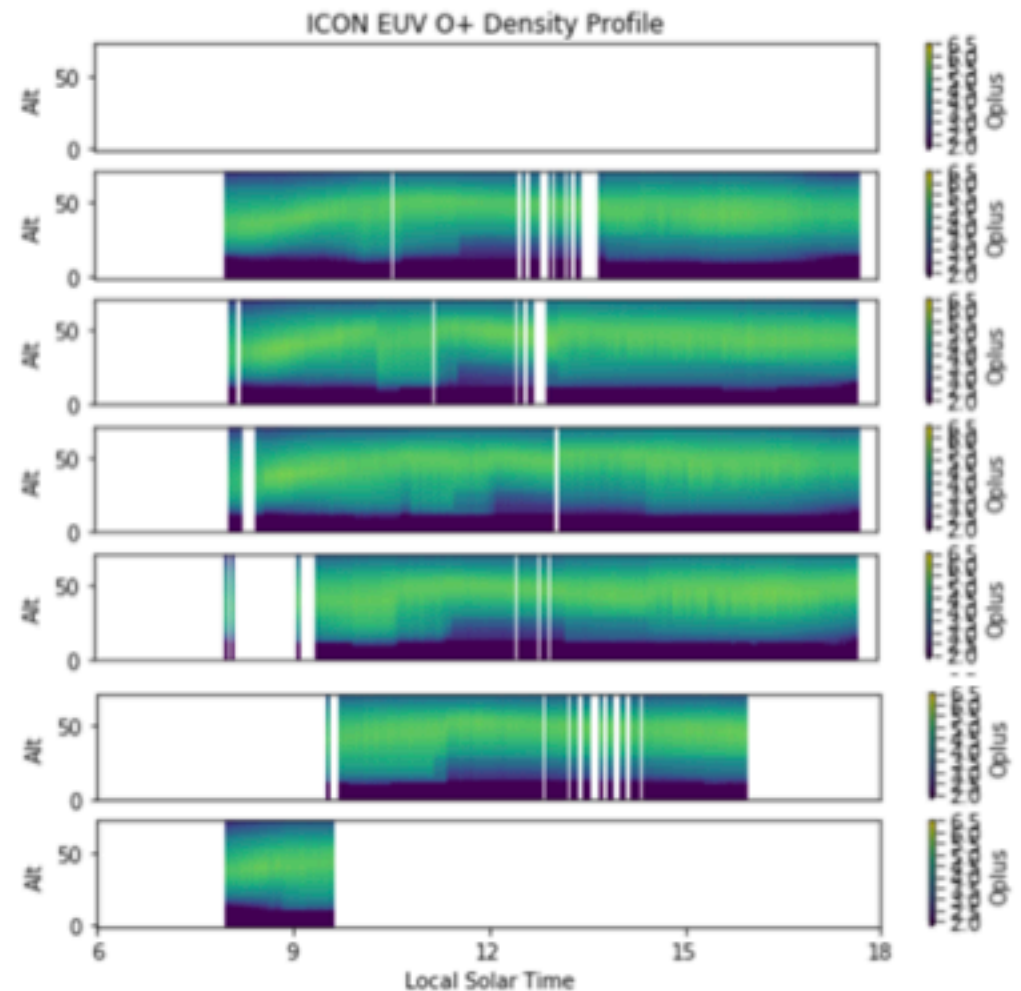
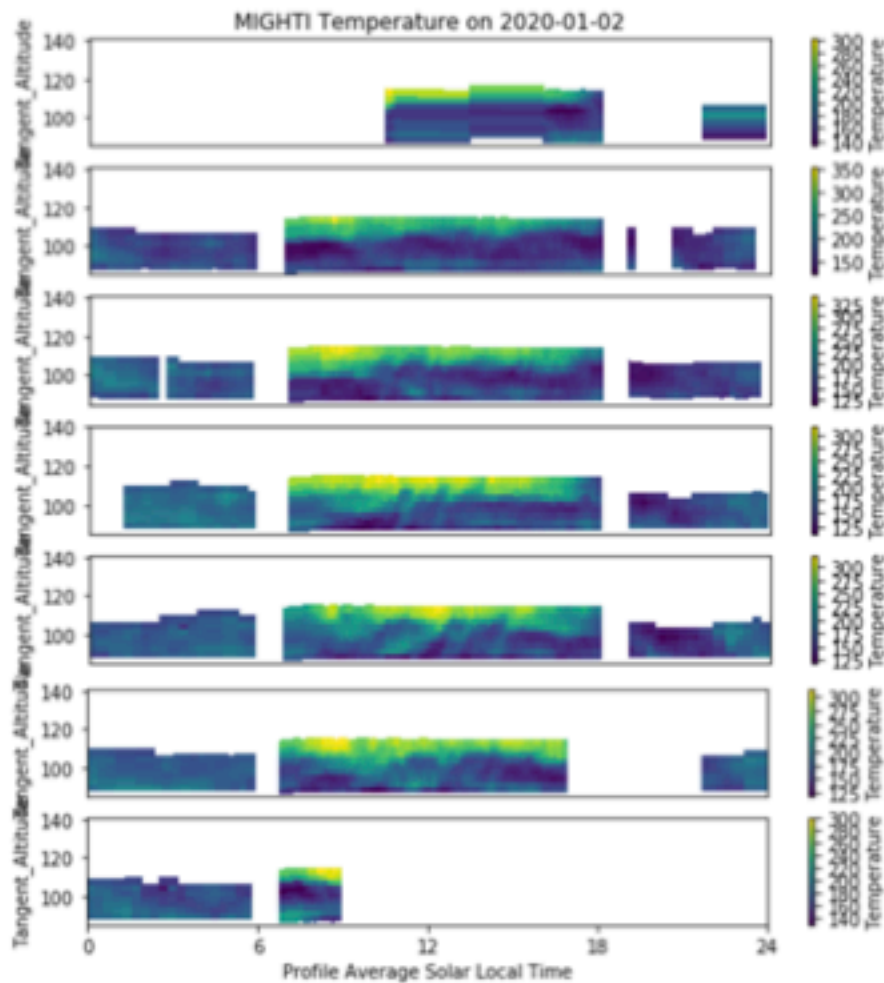
Constructs full orbits across file/day breaks!







More ICON plots via orbit



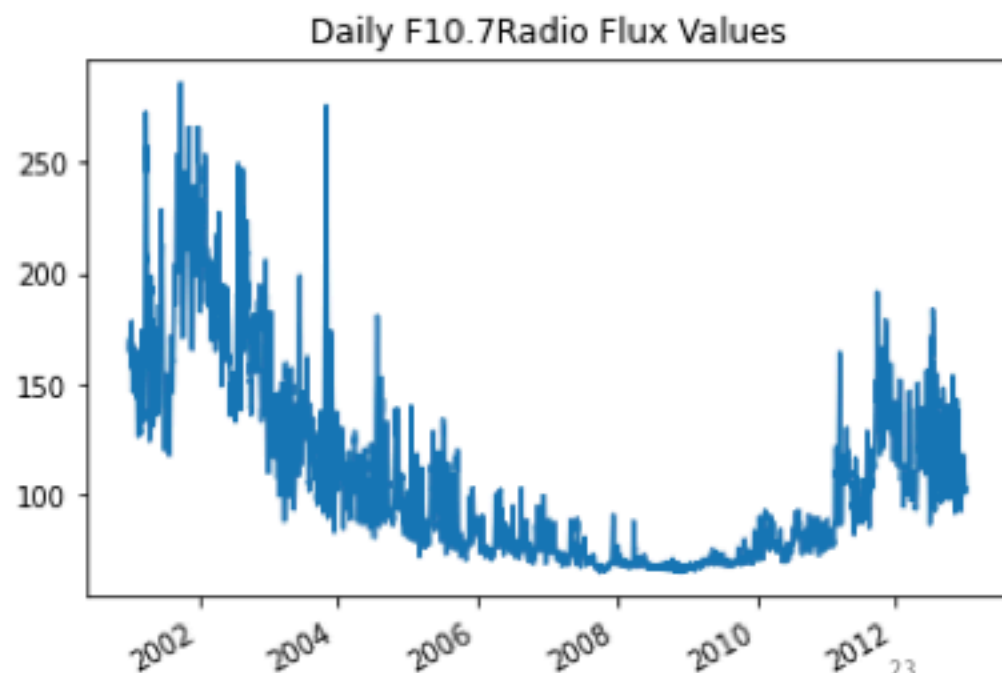
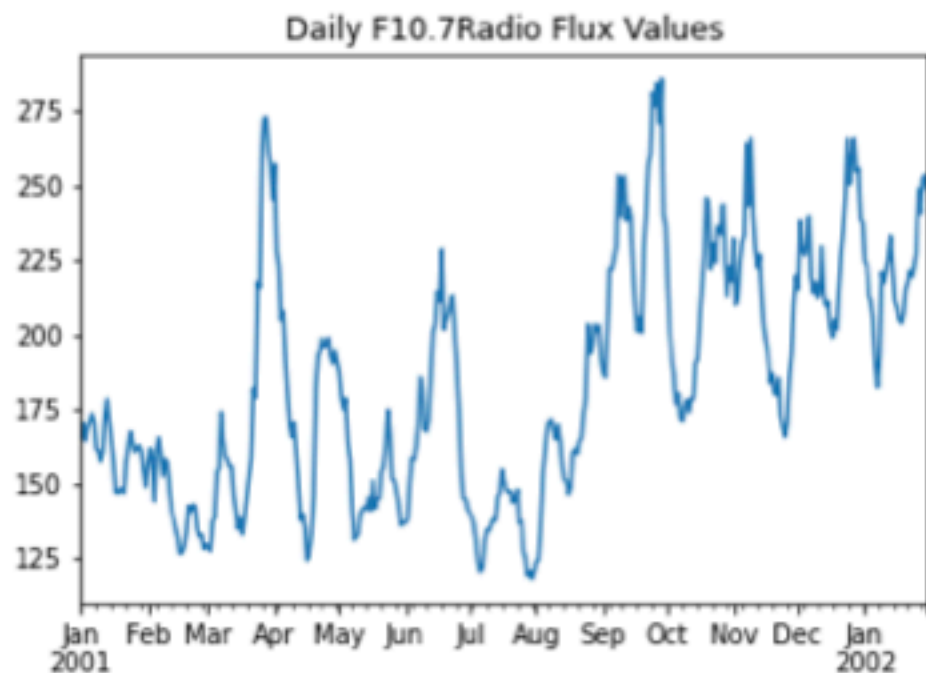
Loading Data Ranges

```
f107 = pysat.Instrument('sw', 'f107')
```

```
f107.load(2001, 1, end_yr=2002, end_doy=31)
```

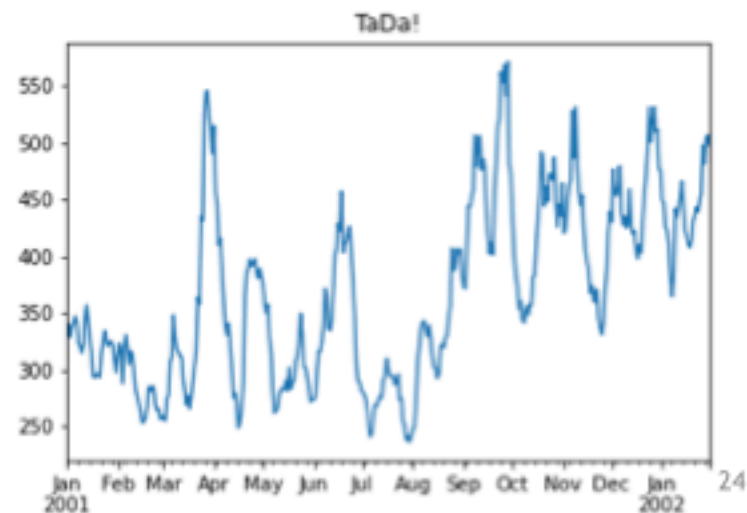
```
f107['f107'].plot(title='Daily F10.7Radio Flux Values')
```

```
import datetime as dt
f107.load(date=dt.datetime(2001, 1, 1),
          end_date=dt.datetime(2012, 12, 31))
f107['f107'].plot(title='Daily F10.7Radio Flux Values')
```



- pysat allows users to preselect several levels of cleaning
 - clean, dusty, dirty, none
 - Actual cleaning performed by instrument specific function
 - Cleaning performed as part of the loading process
- pysat also supports easy modification of data while loading
- Any user accessible data is presented in a 'known' state

```
def custom_function(inst):  
    """Modify data in inst to make it custom """  
    inst['new_variable'] = inst['f107'] * 2.  
    return  
f107 = pysat.Instrument('sw', 'f107')  
f107.custom_attach(custom_function)  
f107.load(2001, 1, end_yr=2002, end_doy=31)  
f107['new_variable'].plot(title='TaDa!')
```



- pysat supports both pandas DataFrames and xarray DataSets

f107.data

	f107
2001-01-01	165.3
2001-01-02	170.2
2001-01-03	164.2
2001-01-04	168.8
2001-01-05	170.5
...	...
2012-12-26	106.2
2012-12-27	103.2
2012-12-28	102.3
2012-12-29	100.9
2012-12-30	103.2

4378 rows x 1 columns

```
import pysat
gps = pysat.Instrument('cosmic', 'gps', tag='ionprf',
                      strict_time_flag=False)
```

```
gps.load(2008, 1)
```

```
<ipython-input-9-bb6939bfe4e3>:1: UserWarning: Metadata set to defaults, as they were missing.
gps.load(2008, 1)
<ipython-input-9-bb6939bfe4e3>:1: UserWarning: Loaded data is not unique.
gps.load(2008, 1)
```

gps.data

xarray.Dataset

This particular dataset still in development

> Dimensions: (RO: 669, time: 1199)

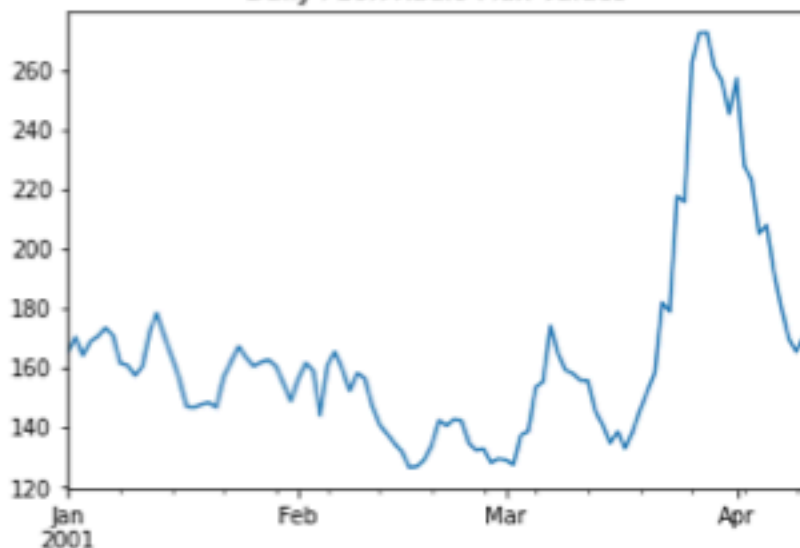
▼ Coordinates:

time	(time)	datetime64[ns]	2008-01-01T00:08:16 ... 2008-01-...	
MSL_alt	(time, RO)	float64	51.46 54.22 56.97 ... nan nan nan	
GEO_lat	(time, RO)	float64	34.3 34.32 34.33 ... nan nan nan	
GEO_lon	(time, RO)	float64	-85.61 -85.62 -85.63 ... nan nan	
OCC_azi	(time, RO)	float64	131.1 131.0 130.9 ... nan nan nan	

> Data variables: (53)

```
f107[0:100, 'f107'].plot(title='Daily F10.7Radio
```

Daily F10.7Radio Flux Values



```
In [30]: nmax['nmax']
Out[30]:
<xarray.DataArray 'nmax' (time: 24, nlats: 100, nlons: 52)>
[124800 values with dtype=float32]
Coordinates:
  channel      (time) I550 ...
  hemisphere   (time) I550 ...
  * time       (time) datetime64[ns] 2020-01-02T20:10:48 ... 2020-01-04T20:10:48
  * nlats      (nlats) int64 0 1 2 3 4 5 6 7 8 9 ... 91 92 93 94 95 96
  * nlons      (nlons) int64 0 1 2 3 4 5 6 7 8 9 ... 43 44 45 46 47 48

In [31]: nmax['nmax'][0, :, :]
Out[31]:
<xarray.DataArray 'nmax' (nlats: 100, nlons: 52)>
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]], dtype=float32)
Coordinates:
  channel      I550 ...
  hemisphere   I550 ...
  time         datetime64[ns] 2020-01-02T20:10:48
  * nlats      (nlats) int64 0 1 2 3 4 5 6 7 8 9 ... 91 92 93 94 95 96
  * nlons      (nlons) int64 0 1 2 3 4 5 6 7 8 9 ... 43 44 45 46 47 48
```

- metadata will be exported to netcdf as attributes when using `pysat.utils.to_netcdf()`
- Tracks 'units', 'long_name', etc.
- Supports programmatic access to metadata
 - Different datasets can call metadata different things and it still gets mapped to the same user determined set of labels
- User centric view

```
In [20]: nmax.meta.data
Out[20]:
```

	units	long_name	...	DISPLAY_TYPE	TIME_BASE
mask_oi_1356		mask_oi_1356	...	NaN	NaN
mask_wavelength		mask_wavelength	...	NaN	NaN
radiance_oi_1356	Rayleighs	radiance_oi_1356	...	image	NaN
channel		channel	...	NaN	NaN
nmax_unc_ran	electrons/cm ³	nmax_unc_ran	...	image	NaN
nmax_unc_mod	electrons/cm ³	nmax_unc_mod	...	image	NaN
nmax_unc_sys	electrons/cm ³	nmax_unc_sys	...	image	NaN
latitude	deg	latitude	...	image	NaN
oi_1356_unc_ran	Rayleighs	oi_1356_unc_ran	...	image	NaN
counts_oi_1356		counts_oi_1356	...	image	NaN
dqi		dqi	...	NaN	NaN
scan_start_time	s	scan_start_time	...	NaN	J2000
nmax_dqi		nmax_dqi	...	image	NaN
solar_zenith_angle	deg	solar_zenith_angle	...	image	NaN
longitude	deg	longitude	...	image	NaN
time_utc	s	time_utc	...	NaN	J2000
nmax	electrons/cm ³	nmax	...	image	NaN
input_llc_file		input_llc_file	...	NaN	NaN
emission_angle	deg	emission_angle	...	image	NaN
scan_stop_time	s	scan_stop_time	...	NaN	J2000
oi_1356_unc_sys	Rayleighs	oi_1356_unc_sys	...	image	NaN
hemisphere		hemisphere	...	NaN	NaN
time		time	...	NaN	NaN
nlats		nlats	...	NaN	NaN
nlons		nlons	...	NaN	NaN
nmask		nmask	...	NaN	NaN

[26 rows x 23 columns]

- Each instrument also includes references and acknowledgements

```
In [27]: nmax.acknowledgements
```

```
Out[27]: 'This is a data product from the NASA Global-scale Observations of the Limb and Disk (GOLD) mission, an Helio physics Explorer mission of opportunity launched in January 2018.\nResponsibility of the mission science falls to the Principal Investigator, Dr. Richard Eastes at University of Colorado/LASP.\nValidation of the L1B data products falls to the instrument lead investigators/scientists.\n* EUV Dr. Bill McClintock\nValidation of the L2 data products falls to Computational Physics, Inc.\n* Dr. Jerry Lumpe\n(https://gold.cs.ucf.edu/).\nOverall validation of the products is overseen by the GOLD Project Scientist Dr. Alan Burns.\nUsers of these data should contact and acknowledge the Principal Investigator Dr. Richard Eastes and the party directly responsible for the data product and the NASA Explorers Project Office.'
```

```
In [28]: nmax.references
```

```
Out[28]: 'Eastes, R.W., McClintock, W.E., Burns, A.G. et al. The Global-Scale Observations of the Limb and Disk (GOLD) Mission. Space Sci Rev 212, 383-408 (2017). https://doi.org/10.1007/s11214-017-0392-2'
```



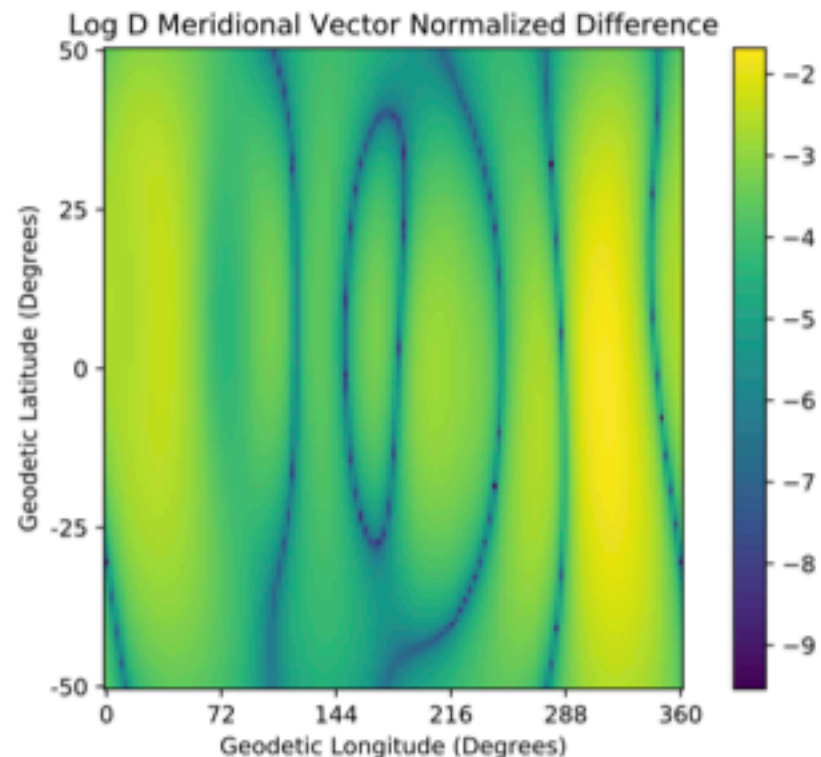
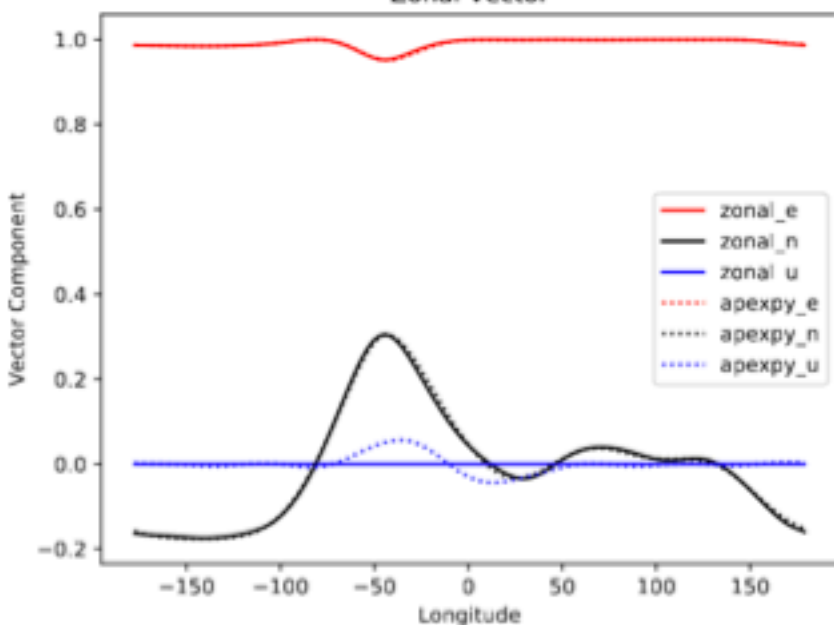
Many Supported Instruments (partial list)

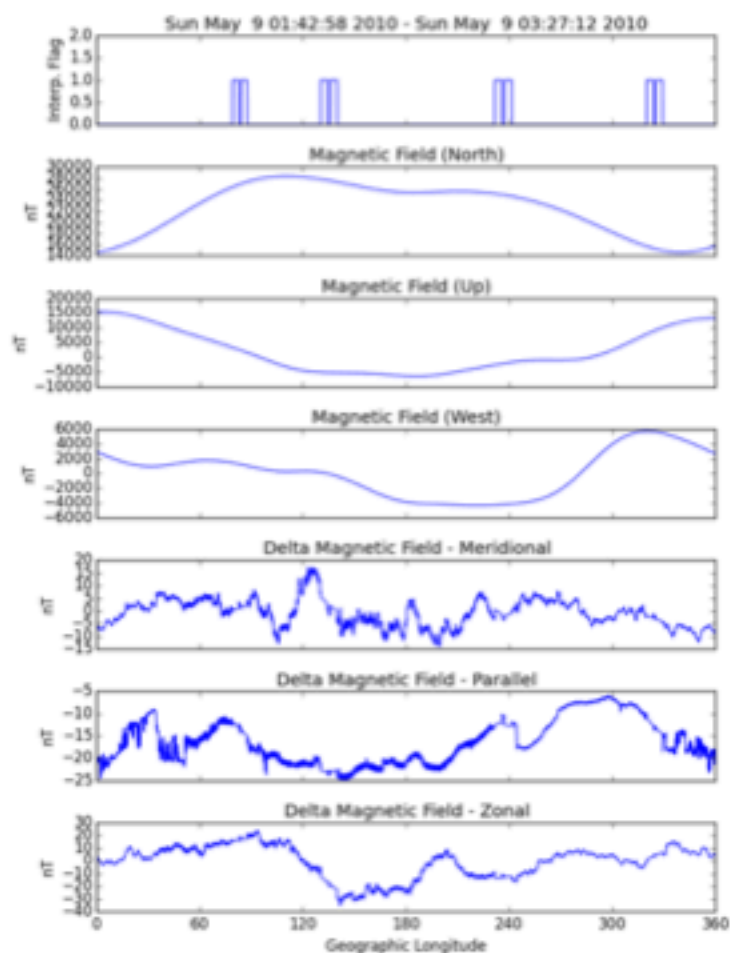
```
pysat.utils.display_available_instruments()
```

Platform	Name	[Tag	Inst_ID]	Description
'ace'	'epam'	['realtime'	'']	Real-time data from the SWPC
		['historic'	'']	Historic data from the SWPC
'ace'	'mag'	['realtime'	'']	Real-time data from the SWPC
		['historic'	'']	Historic data from the SWPC
'ace'	'sis'	['realtime'	'']	Real-time data from the SWPC
		['historic'	'']	Historic data from the SWPC
'ace'	'swepam'	['realtime'	'']	Real-time data from the SWPC
		['historic'	'']	Historic data from the SWPC
'cnofs'	'ivm'	[''	'']	
'cnofs'	'plp'	[''	'']	
'cnofs'	'vefi'	['dc_b'	'']	DC Magnetometer data - 1 second
'cosmic'	'gps'	['ionprf'	'']	Ionospheric Profiles
		['wetprf'	'']	
		['atmprf'	'']	Atmospheric Profiles
		['eraprf'	'']	
		['gfsprf'	'']	
		['ionphs'	'level_1b']	
		['podtec'	'level_1b']	
		['scnlv1'	'level_1b']	
'de2'	'lang'	[''	'']	500 ms cadence Langmuir Probe data
'de2'	'nacs'	[''	'']	1 s cadence Neutral Atmosphere Composition Spectrometer
'de2'	'rpa'	[''	'']	2 sec cadence RPA data
'de2'	'wats'	[''	'']	2 s cadence Wind and Temperature Spectrometer data
'dnsp'	'ivm'	['utd'	'f11']	UTDallas DMSP data processing
		[''	'f11']	Level 2 data processing
		['utd'	'f12']	UTDallas DMSP data processing
		[''	'f12']	Level 2 data processing
		['utd'	'f13']	UTDallas DMSP data processing
		[''	'f13']	Level 2 data processing
		['utd'	'f14']	UTDallas DMSP data processing
		[''	'f14']	Level 2 data processing
		['utd'	'f15']	UTDallas DMSP data processing
		[''	'f15']	Level 2 data processing
		[''	'f15']	Level 2 data processing
'gnss'	'tec'	['vtec'	'']	vertical TEC
'icon'	'euv'	[''	'']	Level 2 public geophysical data
'icon'	'fuv'	['day'	'']	Level 2 daytime O/N2
		['night'	'']	Level 2 nighttime O profile
'icon'	'ivm'	[''	'a']	Level 2 public geophysical data
		[''	'b']	Level 2 public geophysical data
'icon'	'mighti'	['vector_wind_green'	'']	Vector wind data -- Green Line
		['vector_wind_red'	'']	Vector wind data -- Red Line
		['los_wind_green'	'a']	Line of sight wind data -- Green Line
		['los_wind_red'	'a']	Line of sight wind data -- Red Line
		['temperature'	'a']	Neutral temperature data
		['los_wind_green'	'b']	Line of sight wind data -- Green Line
		['los_wind_red'	'b']	Line of sight wind data -- Red Line
		['temperature'	'b']	Neutral temperature data
'iss'	'fpmu'	[''	'']	
'jro'	'isr'	['drifts'	'']	Drifts and wind
		['drifts_ave'	'']	Averaged drifts
		['oblique_stan'	'']	Standard Faraday rotation double-pulse
		['oblique_rand'	'']	Randomized Faraday rotation double-pulse
		['oblique_long'	'']	Long pulse Faraday rotation
'omni'	'hro'	['1min'	'']	1-minute time averaged data
		['5min'	'']	5-minute time averaged data
'pydineof'	'dineof'	[''	'']	pydineof output file
		['test'	'']	Standard output of pydineof for benchmarking
'pysat'	'ephen'	[''	'']	Satellite simulation data set
'pysat'	'sgp4'	[''	'']	Satellite simulation data set
'sami2py'	'sami2'	[''	'']	sami2py output file
		['test'	'']	Standard output of sami2py for benchmarking
'ses14'	'gold'	['nmax'	'']	Level 2 Nmax data for the GOLD instrument
'sport'	'ivm'	[''	'']	Level-2 IVM Files
		['l1'	'']	Level-1 IVM Files
		['l0'	'']	Level-0 IVM Files
'sw'	'dst'	[''	'']	
'sw'	'f107'	[''	'']	Daily LASP value of F10.7
		['all'	'']	All LASP F10.7 values (deprecated, use '' instead)
		['prelim'	'']	Preliminary SWPC daily solar indices

- First **orthogonal** geomagnetic coordinate system for multi-pole magnetic fields (drift mapping accurate to ~2%)
- Supports mapping observed electric fields along geomagnetic field-lines
- Used by ICON, COSMIC-2, SORTIE

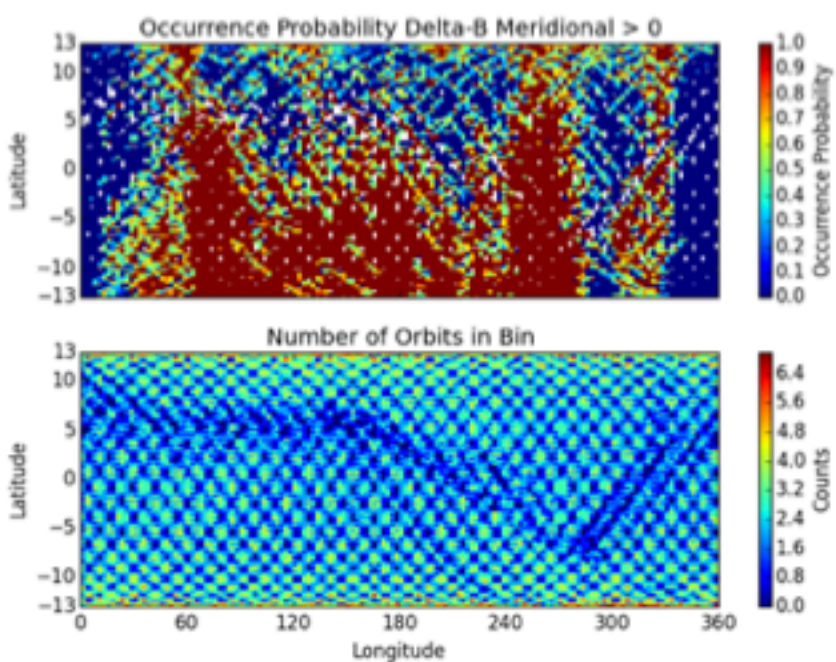
Zonal Vector



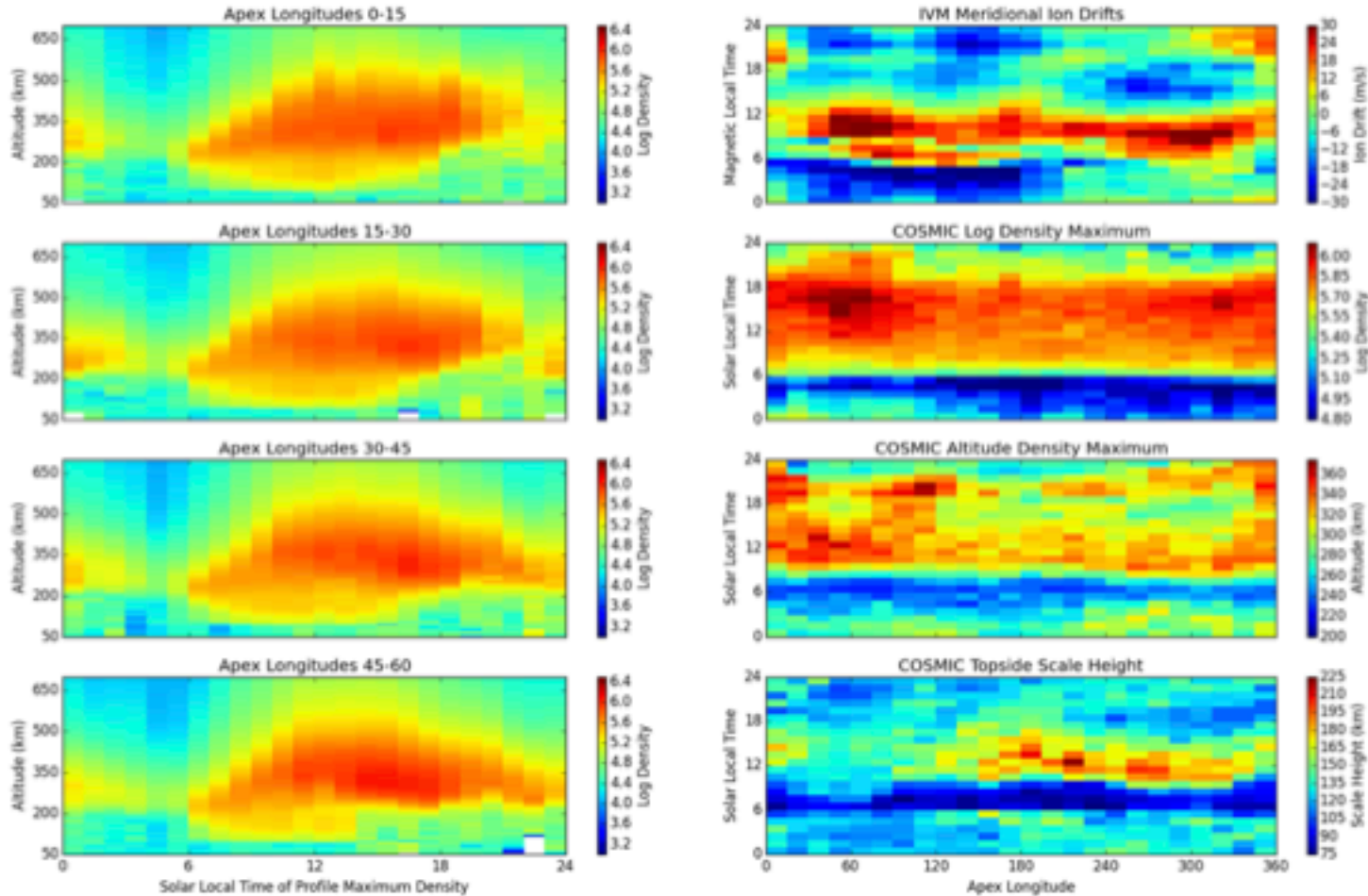


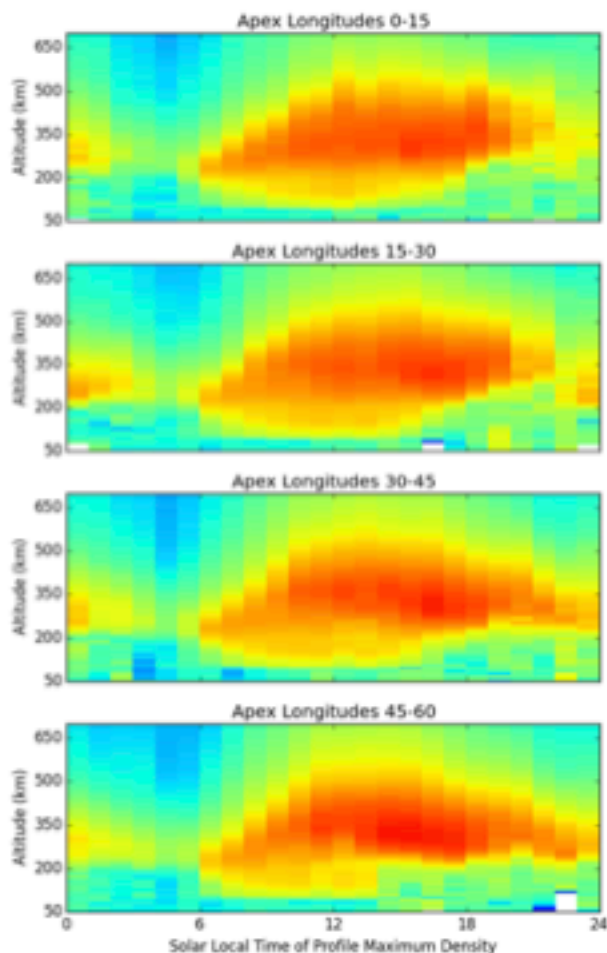
```
# select vefi dc magnetometer data, use longitude to determine where
# there are changes in the orbit (local time info not in file)
orbit_info = {'index':'longitude', 'kind':'longitude'}
vefi = pysat.Instrument(platform='cnofs', name='vefi', tag='dc_b',
                        clean_level=None, orbit_info=orbit_info)

# perform occurrence probability calculation
# any data added by custom functions is available within routine below
ans = pysat.ssnl.occu_prob.by_orbit2D(vefi, [0,360,144], 'longitude',
                                     [-13,13,104], 'latitude', ['dB_mer'], [0.], returnBins=True)
```

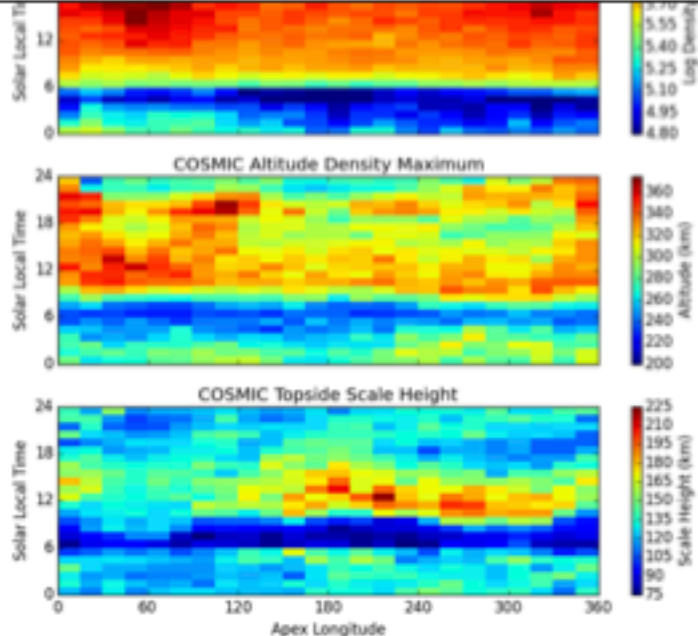


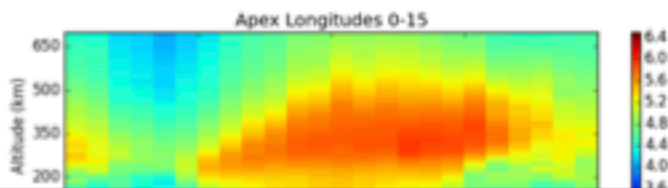
Full Code in Demo Area of Repo





```
# instantiate IVM Object
ivm = pysat.Instrument(platform='cnofs',
                        name='ivm', tag='',
                        clean_level='clean')
# restrict measurements to those near geomagnetic equator
ivm.custom.add(restrictMLAT, 'modify', maxMLAT=25.)
# perform seasonal average
ivm.bounds = (startDate, stopDate)
ivmResults = pysat.ssn1.avg.median2D(ivm, [0,360,24], 'alon',
                                     [0,24,24], 'mlt', ['ionVelmeridional'])
```



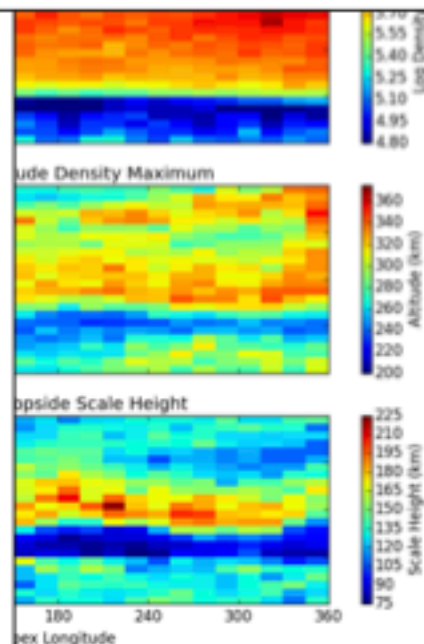


```
# create COSMIC instrument object
cosmic = pysat.Instrument(platform='cosmic2013',
                          name='gps', tag='ionprf',
                          clean_level='clean',
                          altitude_bin=3)

# apply custom functions to all data that is loaded through cosmic
cosmic.custom.add(addApexLong, 'add')
# select locations near the magnetic equator
cosmic.custom.add(filterMLAT, 'modify', mlatRange=(0.,10.) )
# take the log of NmF2 and add to the dataframe
cosmic.custom.add(addlogNm, 'add')
# calculates the height above hmF2 to reach Ne < NmF2/e
cosmic.custom.add(addTopsideScaleHeight, 'add')

# do an average of multiple COSMIC data products
# from startDate through stopDate
# a mixture of 1D and 2D data is averaged
cosmic.bounds = (startDate, stopDate)
cosmicResults = pysat.ssn1.avg.median2D(cosmic, [0,360,24], 'apex_long',
                                         [0,24,24], 'edmaxlct', ['profiles', 'edmaxalt', 'lognm', 'thf2'])
```

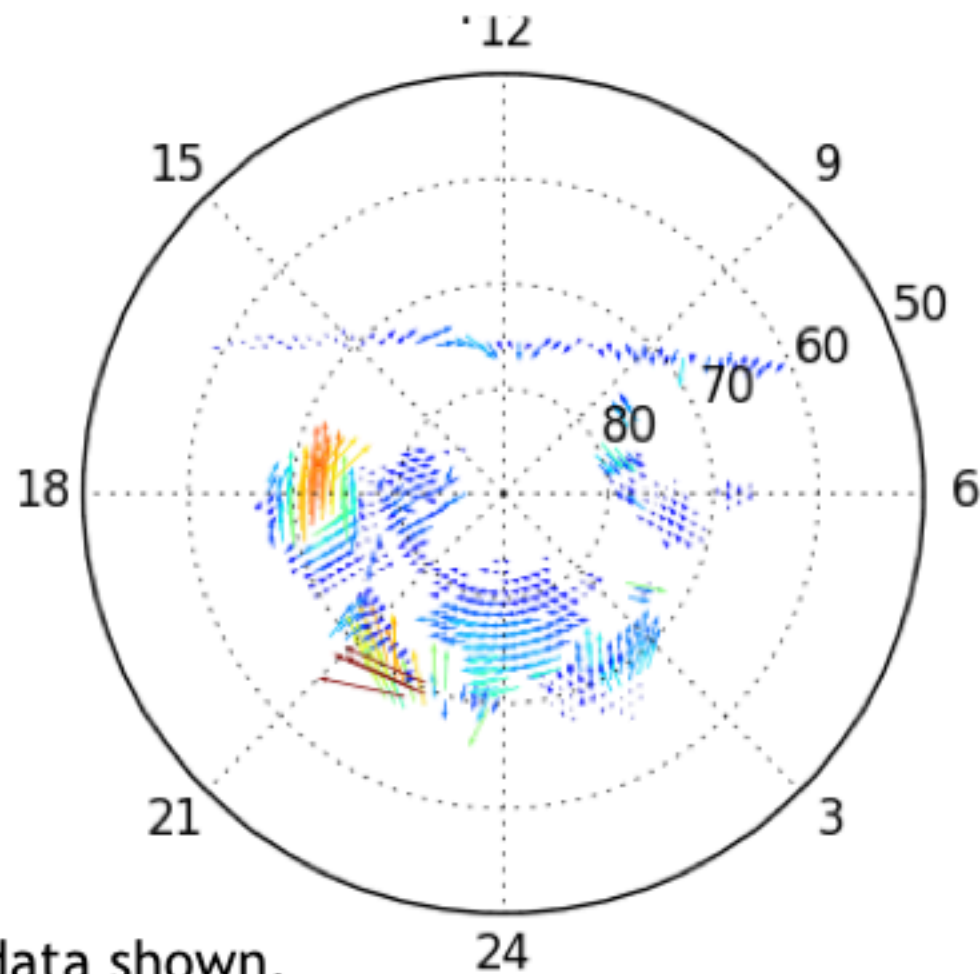
```
# instantiate IVM Object
ivm = pysat.Instrument(platform='cnofs',
                       name='ivm', tag='',
                       clean_level='clean')
# restrict measurements to those near geomagnetic equator
ivm.custom.add(restrictMLAT, 'modify', maxMLAT=25.)
# perform seasonal average
ivm.bounds = (startDate, stopDate)
ivmResults = pysat.ssn1.avg.median2D(ivm, [0,360,24], 'alon',
                                     [0,24,24], 'mlt', ['ionVelmeridional'])
```



- Interfaces with VT servers (shutdown)

```
sdarn = pysat.Instrument('superdarn', 'grdex', 'north')
sdarn.download(date1, date2)
sdarn.load(date=date1)
```

Support for SuperDARN uses community SuperDARN package



Plot including both DMSP and SuperDARN data shown.

The research performed by A.G. Burrell was funded by the Chief of Naval Research

pysat source code available at <https://github.com/pysat/pysat>

Or 'pip install pysat'

Version 3.0 Recently Released!

pysat ecosystem available at <https://github.com/pysat>

OMMBV available at <https://github.com/rstoneback>

main 18 branches 32 tags

Go to file Add file Code

About

rstoneback Merge pull request #676 from pysat/v3-0-rc1 988dead 12 days ago 5,857 commits

.github	Merge branch 'main' into v3-0-rc1	2 months ago
docs	Apply suggestions from code review	12 days ago
pysat	STY: Adjusted imports	12 days ago
.gitignore	first round of pycodestyle complete	2 years ago
.readthedocs.yml	Update .readthedocs.yml	8 months ago
.travis.yml	STY: Removed data dir path creation on TravisCI	last month
.zenodo.json	ENH: added references, keywords to zenodo	3 months ago
CHANGELOG.md	Apply suggestions from code review	14 days ago
CODE_OF_CONDUCT.md	Update CODE_OF_CONDUCT.md	3 years ago
CONTRIBUTING.md	Apply suggestions from code review	13 days ago
LICENSE	Updated year	5 years ago
MANIFEST.in	ENH: First pass citation.txt	2 months ago

Generalized satellite and space science data processing and file management.

- python plasma nasa space
- space-science magnetometer
- netcdf cdf measurements
- satellite-data cubesat ionosphere
- electric-fields nasa-data
- science-research thermosphere
- madrigal radar-measurements
- magnetosphere

- Readme
- BSD-3-Clause License

Releases 32

v3.0.0 Latest 12 days ago

https://github.com/pysat/pysat

Label issues and pull requests for new contributors Dismiss

Now, GitHub will help potential first-time contributors [discover issues](#) labeled with **good first issue**

Filters Labels 15 Milestones 3 New issue

<input type="checkbox"/>	40 Open ✓ 250 Closed	Author ▾	Label ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	Common Constellation operations enhancement #764 opened 5 days ago by aburrell 0 of 2 3.1.0 Release						
<input type="checkbox"/>	ENH: Additional keyword support parsing filenames #763 opened 6 days ago by rstoneback 3.1.0 Release						
<input type="checkbox"/>	ENH: Review and document file parsing code in pysat.utils.files #762 opened 6 days ago by rstoneback 3.1.0 Release						1
<input type="checkbox"/>	BUG: data access via slicing if data is missing bug #761 opened 8 days ago by jklenzing 3.0.1 Release						4
<input type="checkbox"/>	STY/TST: caplog usage in unit tests style testing #751 opened 19 days ago by jklenzing 3.0.1 Release						1



Search or jump to...

[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



[pysat / pysat](#)

[Unwatch](#) 9

[Unstar](#) 74

[Fork](#) 30

[Code](#) [Issues 40](#) [Pull requests](#) [Actions](#) [Projects 6](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

[Labels](#) [Milestones](#)

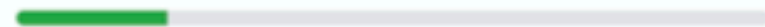
[New milestone](#)

3 Open ✓ 7 Closed

Sort

3.0.1 Release

No due date Last updated 5 days ago

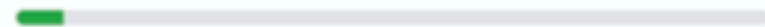


20% complete 4 open 1 closed

[Edit](#) [Close](#) [Delete](#)

3.1.0 Release

No due date Last updated 5 days ago



6% complete 30 open 2 closed

[Edit](#) [Close](#) [Delete](#)

Major changes that need 3.0.0 changes to be implemented first, but should not hold up the final version of 3.0.0.

Future

No due date Last updated 15 days ago



16% complete 5 open 1 closed

[Edit](#) [Close](#) [Delete](#)

Milestones for future releases