Hands-on Session Preparation

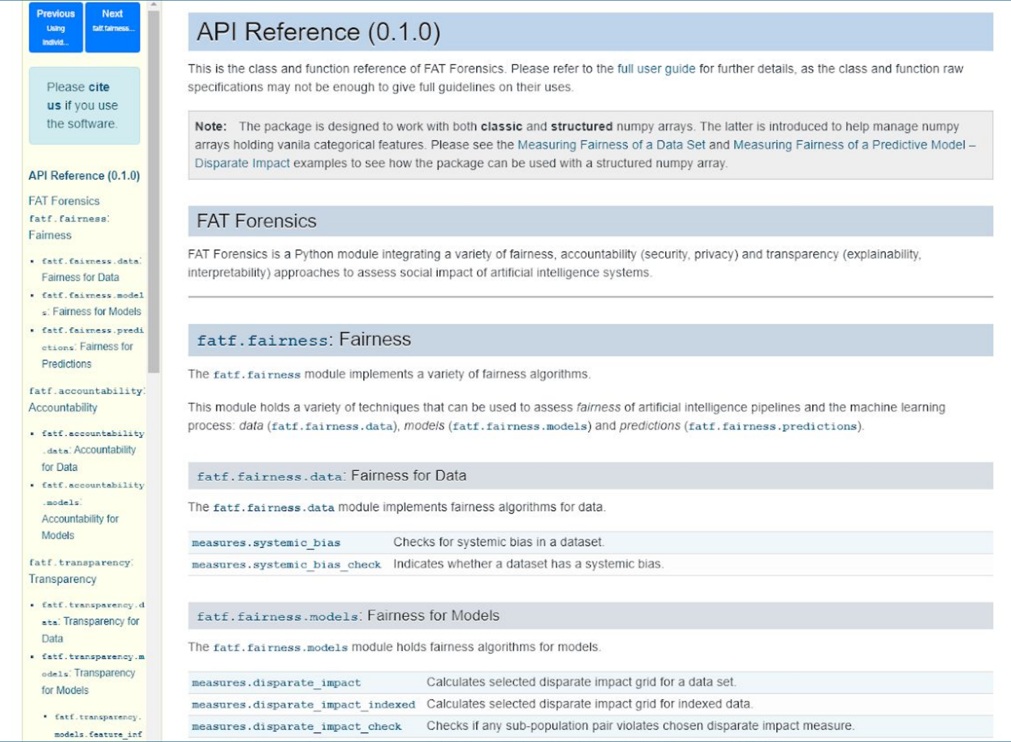
- FAT Forensics documentation and resources.
- Tutorial notebooks.
- Local installation.
- Google Colab.
- My Binder.
- Testing the environment.

Alex Hepburn

Documentation

- API
  - https://fat-forensics.org/api.html
- Code Examples
  - https://fat-forensics.org/sphinx_gallery_auto/
- Tutorials
  - https://fat-forensics.org/tutorials/
- How-to Guide(s)
  - https://fat-forensics.org/how_to/

https://fat-forensics.org/api.html



Previous
Using Individ...

Next
fatf.fairness...

Please **cite us** if you use the software.

API Reference (0.1.0)

FAT Forensics

fatf.fairness: Fairness

- fatf.fairness.data: Fairness for Data
- fatf.fairness.models: Fairness for Models
- fatf.fairness.predictions: Fairness for Predictions

fatf.accountability: Accountability

- fatf.accountability.data: Accountability for Data
- fatf.accountability.models: Accountability for Models

fatf.transparency: Transparency

- fatf.transparency.data: Transparency for Data
- fatf.transparency.models: Transparency for Models
  - fatf.transparency.models.feature_inf

API Reference (0.1.0)

This is the class and function reference of FAT Forensics. Please refer to the full user guide for further details, as the class and function raw specifications may not be enough to give full guidelines on their uses.

Note: The package is designed to work with both **classic** and **structured** numpy arrays. The latter is introduced to help manage numpy arrays holding vanila categorical features. Please see the Measuring Fairness of a Data Set and Measuring Fairness of a Predictive Model – Disparate Impact examples to see how the package can be used with a structured numpy array.

FAT Forensics

FAT Forensics is a Python module integrating a variety of fairness, accountability (security, privacy) and transparency (explainability, interpretability) approaches to assess social impact of artificial intelligence systems.

`fatf.fairness`: Fairness

The `fatf.fairness` module implements a variety of fairness algorithms.

This module holds a variety of techniques that can be used to assess *fairness* of artificial intelligence pipelines and the machine learning process: *data* (`fatf.fairness.data`), *models* (`fatf.fairness.models`) and *predictions* (`fatf.fairness.predictions`).

`fatf.fairness.data`: Fairness for Data

The `fatf.fairness.data` module implements fairness algorithms for data.

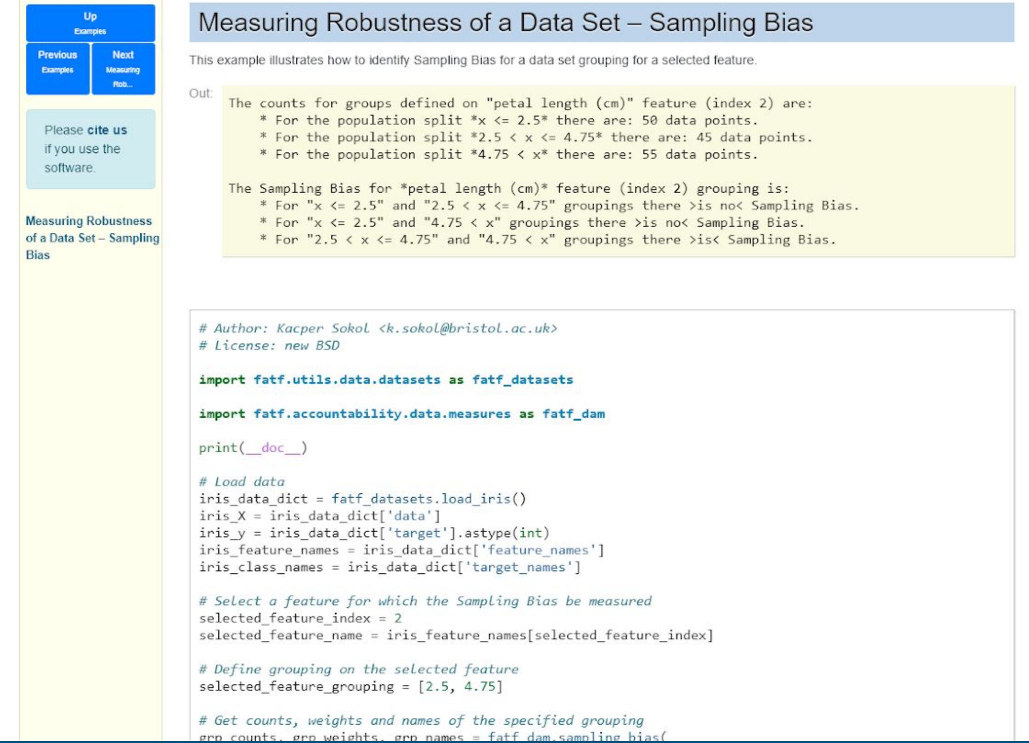
`measures.systemic_bias`	Checks for systemic bias in a dataset.
`measures.systemic_bias_check`	Indicates whether a dataset has a systemic bias.

`fatf.fairness.models`: Fairness for Models

The `fatf.fairness.models` module holds fairness algorithms for models.

`measures.disparate_impact`	Calculates selected disparate impact grid for a data set.
`measures.disparate_impact_indexed`	Calculates selected disparate impact grid for indexed data.
`measures.disparate_impact_check`	Checks if any sub-population pair violates chosen disparate impact measure.

https://fat-forensics.org/sphinx_gallery_auto/



Up
Examples

Previous
Examples

Next
Measuring Rob...

Please **cite us** if you use the software.

Measuring Robustness of a Data Set – Sampling Bias

Measuring Robustness of a Data Set – Sampling Bias

This example illustrates how to identify Sampling Bias for a data set grouping for a selected feature.

Out:

```
The counts for groups defined on "petal length (cm)" feature (index 2) are:
    * For the population split *x <= 2.5* there are: 50 data points.
    * For the population split *2.5 < x <= 4.75* there are: 45 data points.
    * For the population split *4.75 < x* there are: 55 data points.

The Sampling Bias for *petal length (cm)* feature (index 2) grouping is:
    * For "x <= 2.5" and "2.5 < x <= 4.75" groupings there >is no< Sampling Bias.
    * For "x <= 2.5" and "4.75 < x" groupings there >is no< Sampling Bias.
    * For "2.5 < x <= 4.75" and "4.75 < x" groupings there >is< Sampling Bias.
```

```python
# Author: Kacper Sokol <k.sokol@bristol.ac.uk>
# License: new BSD

import fatf.utils.data.datasets as fatf_datasets

import fatf.accountability.data.measures as fatf_dam

print(__doc__)

# Load data
iris_data_dict = fatf_datasets.load_iris()
iris_X = iris_data_dict['data']
iris_y = iris_data_dict['target'].astype(int)
iris_feature_names = iris_data_dict['feature_names']
iris_class_names = iris_data_dict['target_names']

# Select a feature for which the Sampling Bias be measured
selected_feature_index = 2
selected_feature_name = iris_feature_names[selected_feature_index]

# Define grouping on the selected feature
selected_feature_grouping = [2.5, 4.75]

# Get counts, weights and names of the specified grouping
grp_counts, grp_weights, grp_names = fatf_dam.sampling_bias(
```

https://fat-forensics.org/tutorials/

Up
Tutorials

Previous
Explaining a ...

Next
Examples

Please **cite us** if you use the software.

Explaining Machine Learning Predictions: LIME and Counterfactuals

Counterfactual Explanations

- Explanation Reliability
- Counterfactual Fairness

LIME

Relevant FAT Forensics Examples

LIME

In addition to counterfactual explanations we also have an implementation of the Local Interpretable Model-agnostic Explanations (LIME). Let us see what LIME can tell us about our predictive model's behaviour in the neighbourhood of our setosa data point:

```
>>> fatf.setup_random_seed(42)
>>> import fatf.transparency.predictions.surrogate_explainers as surrogates

>>> iris_lime = surrogates.TabularBlimeyLime(
...     iris_data,
...     clf,
...     feature_names=iris_feature_names.tolist(),
...     class_names=iris_target_names.tolist())
>>> lime_explanation = iris_lime.explain_instance(
...     x_setosa, samples_number=500)
```

Let us first have a look at the text version of our LIME explanation for the setosa data point:

```
>>> pprint(lime_explanation)
{'setosa': {'*petal width (cm)* <= 0.30': 0.03698295125607895,
            '*sepal length (cm)* <= 5.10': 0.013734230852263654,
            '1.60 < *petal length (cm)* <= 4.35': -0.18301541432210996,
            '3.30 < *sepal width (cm)*': 0.08821105886209096},
 'versicolor': {'*petal width (cm)* <= 0.30': 0.05771468585613034,
                '*sepal length (cm)* <= 5.10': 0.025733500528816115,
                '1.60 < *petal length (cm)* <= 4.35': 0.4694710470975027,
                '3.30 < *sepal width (cm)*': -0.00246315406456463},
 'virginica': {'*petal width (cm)* <= 0.30': -0.09469763711220923,
               '*sepal length (cm)* <= 5.10': -0.03946773138107974,
               '1.60 < *petal length (cm)* <= 4.35': -0.2864556327753925
4,
               '3.30 < *sepal width (cm)*': -0.08574790479752634}}
```

With all these numbers it may actually be easier to interpret their visualisation, which we can generate using the built-in `fatf.vis.lime.plot_lime` plotting function:

```
>>> import fatf.vis.lime as fatf_vis_lime

>>> lime_fig_setosa = fatf_vis_lime.plot_lime(lime_explanation)
```

https://fat-forensics.org/how_to/

Up
How-to Guides

Previous
How-to Guides

Next
News

Please **cite us** if you use the software.

How to build LIME yourself (bLIMEy) – Surrogate Tabular Explainers

Setup

Surrogate Linear Model (LIME)

- Data Augmentation
- Interpretable Representation
- Explanation Generation

Surrogate Tree

How to build LIME yourself (bLIMEy) – Surrogate Tabular Explainers

How-to Guide Contents

This how-to guide illustrates how to construct a local surrogate model on top of a black-box model and use it to generate explanations of selected predictions of the black-box model.

This how-to guide requires `scikit-learn` package as it uses ridge regression and decision tree predictors (implemented therein) as local surrogate models.

Each surrogate explainer is composed of three main parts:

- interpretable data representation;
- data sampling; and
- explanation generation.

Choosing a particular algorithm for each of these components shapes the type of surrogate explanations that can be generated with the final explainer. (The theoretical considerations for each component can be found in Surrogate Transparency User Guide, [SOKOL2019BLIMEY] and the Jupyter Notebook distributed with the latter manuscript.) In this how-to guide we will show how to build the tabular LIME explainer [RIBEIRO2016WHY] (with fixed sampling procedure [SOKOL2019BLIMEY] and the sampling algorithm replaced with MixuP – `fatf.utils.data.augmentation.Mixup`) and a simple tree-based surrogate.

Two similar surrogate explainer are already distributed with this package: `fatf.transparency.predictions.surrogate_explainers.TabularBlimeyLime` and `fatf.transparency.predictions.surrogate_explainers.TabularBlimeyTree`. However, the LIME explainer implementation is the exact replica of its official implementation, hence it does the "reverse sampling", which introduces randomness to the explainer. Both of these classes provide usage convenience – no need to build the explainers from scratch – in exchange for lack of flexibility – none of the three aforementioned components can be customised.

Note: Deploying Surrogate Explainers
You may want to consider using the abstract `fatf.transparency.predictions.surrogate_explainers.SurrogateTabularExplainer` class to implement a custom surrogate explainer for tabular data. This abstract class implements a series of input validation steps and internal attribute computation that make implementing a custom surrogate considerably easier.

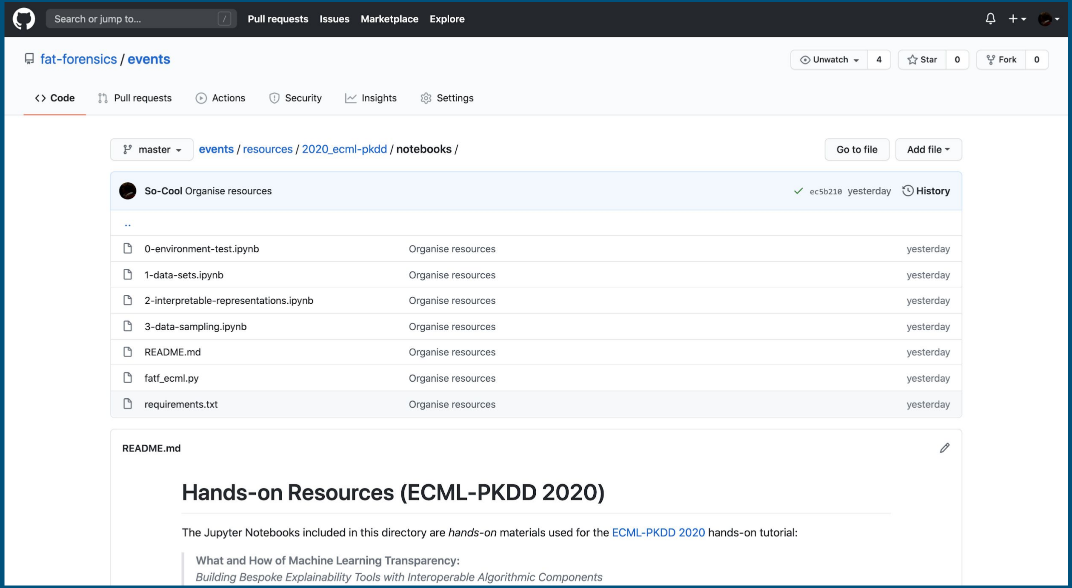
SOKOL2019BLIMEY(1,2) Sokol, K., Hepburn, A., Santos-Rodriguez, R. and Flach, P., 2019. bLIMEy: Surrogate Prediction Explanations Beyond LIME. 2019 Workshop on Human-Centric Machine Learning (HCML 2019). 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada. arXiv preprint arXiv:1910.13016. URL https://arxiv.org/abs/1910.13016.

[RIBEIRO2016WHY] Ribeiro, M.T., Singh, S. and Guestrin, C., 2016, August. Why should i trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144). ACM

Executing the Notebooks

Hands-on Jupyter Notebooks

https://github.com/fat-forensics/events/tree/master/resources/2020_ecml-pkdd/notebooks/



Usage Dependencies

Module	Dependencies	Install command
fatf	numpy >= 1.10.0 scipy >= 0.13.3	$ pip install fat-forensics
fatf.transparency.predictions.surrogate_explainers fatf.transparency.sklearn fatf.utils.data.feature_selection.sklearn	scikit-learn >= 0.19.2	$ pip install fat-forensics[ml]
fatf.vis	matplotlib >= 3.0.0	$ pip install fat-forensics[vis]
fatf.*	↥	$ pip install fat-forensics[all]

Development Dependencies

Module	Dependencies	Install command

`development tools and packages`	`codecov == 2.1.0`
	`flake8 == 3.8.1`
	`mypy == 0.770`
	`nbval == 0.9.1`
	`numpydoc == 0.8.0`
	`pylint == 2.3.0`
	`pytest == 3.9.1`
	`pytest-cov == 2.6.0`
	`sphinx == 2.0`
	`sphinx-gallery == 0.3.1`
	`twine == 1.14.0`
	`yapf == 0.26.0`
		`$ pip install fat-forensics[dev]`

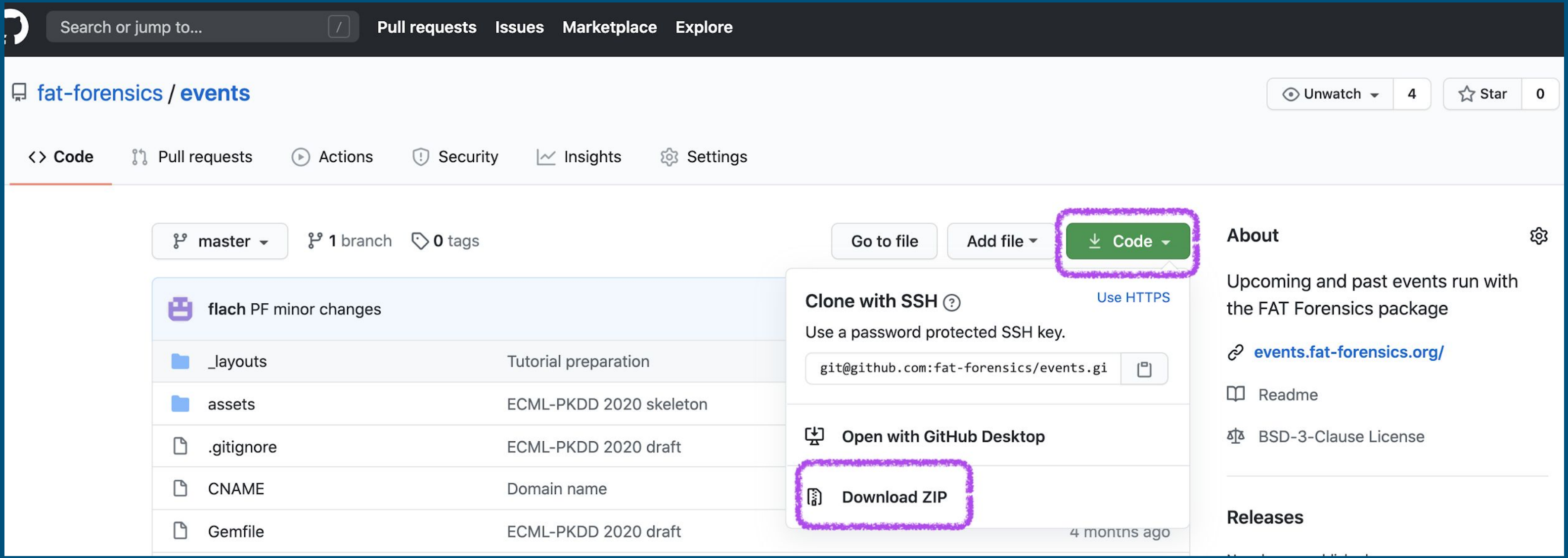
Local Installation (`$ pip install ...`)

1. Install FAT Forensics with auxiliary dependencies:
   - `$ pip install fat-forensics[all]`
2. Install Jupyter Lab:
   - `$ pip install jupyterlab`
3. Download the fat-forensics/events GitHub repository with *git* or *zip* (https://github.com/fat-forensics/events):
   - `$ git clone https://github.com/fat-forensics/events.git`
   - https://github.com/fat-forensics/events/archive/master.zip
4. Launch Jupyter Lab in the `resources/2020_ecml-pkdd/notebooks/` folder with:
   - `$ jupyter lab`

Software	license BSD-3-Clause	release v0.1.0	pypi v0.1.0	python 3.5

Local Installation (`$ pip install ...`)

https://github.com/fat-forensics/events



Colab (Google Account Required)

- Cold start: `!pip install fat-forensics[all]`
- Dedicated function: `fatf_ecml.initialise_colab()`

\+ Code \+ Text

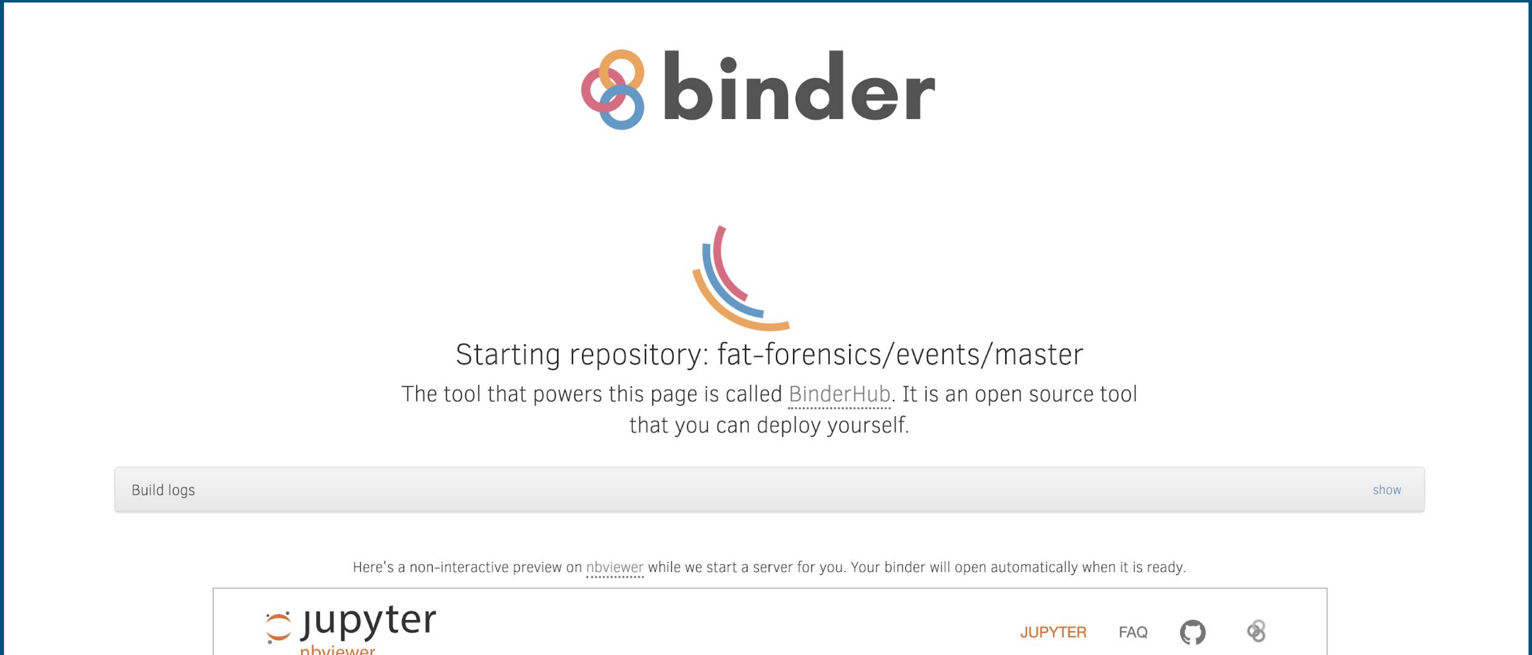
```
[1]  !pip install fat-forensics[all]

Collecting fat-forensics[all]
  Downloading https://files.pythonhosted.org/p
     |████████████████████████████████| 184kB
Requirement already satisfied: scipy>=0.13.3 i
Requirement already satisfied: numpy>=1.10.0 i
Requirement already satisfied: matplotlib>=3.0
Requirement already satisfied: scikit-learn>=0
Requirement already satisfied: python-dateutil
Requirement already satisfied: cycler>=0.10 in
Requirement already satisfied: pyparsing!=2.0.
Requirement already satisfied: kiwisolver>=1.0
Requirement already satisfied: joblib>=0.11 in
Requirement already satisfied: six>=1.5 in /us
Installing collected packages: fat-forensics
Successfully installed fat-forensics-0.1.0
```

```
[1]
LIBRARY_URL = 'https://github.com/fat-forensics/even

try:
    import google.colab
    !wget $LIBRARY_URL -O fatf_ecml.py
except ImportError:
    pass

import fatf_ecml

import matplotlib.pyplot as plt
import numpy as np

from IPython.display import display

%matplotlib inline
```

```
[2]  fatf_ecml.initialise_colab()

Installing FAT Forensics.
Collecting fat-forensics[all]
  Downloading https://files.pythonhosted.org/packages/3
Requirement already satisfied: numpy>=1.10.0 in /usr/lo
Requirement already satisfied: scipy>=0.13.3 in /usr/lo
```

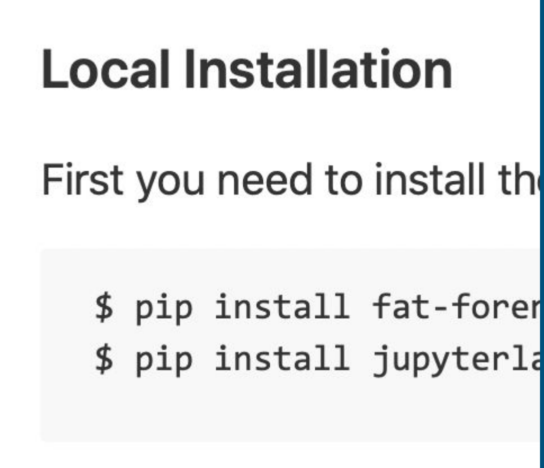
My Binder

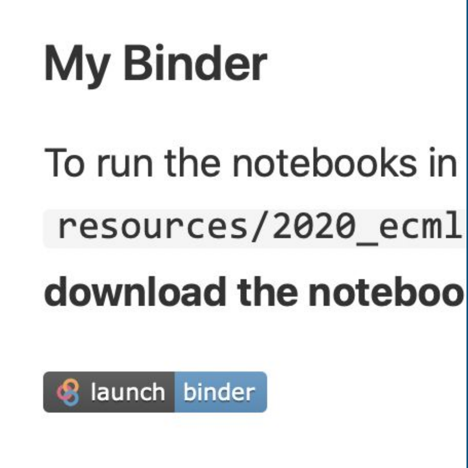
Go to the `resources/2020_ecml-pkdd/notebooks/` directory.

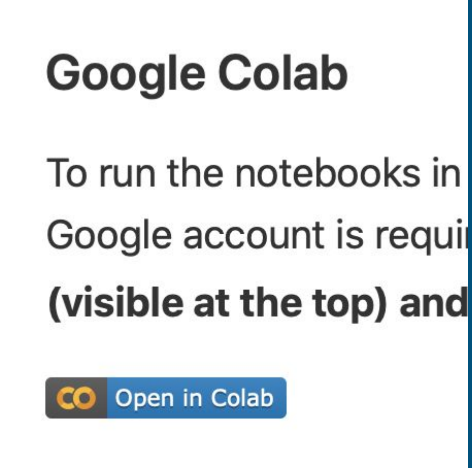


Testing the Hands-on Environment

- https://github.com/fat-forensics/events/tree/master/resources/2020_ecml-pkdd/notebooks/
- Execute all cells of the `0-environment-test.ipynb` notebook to test your environment.







Next Up

Break

Get in touch via Slack for questions and troubleshooting.

https://fatforensicsevents.slack.com/

(Alex Hepburn & Kacper Sokol)

Next Next Up

Part 3 (Hands-on): Building Bespoke Surrogate Explainers

Introduction to the Hands-on Resources

(Alex Hepburn)