The Jupyter/IPython Architecture

a unified view of computational research from interactive exploration to communication and publication

Min Ragan-Kelley
UC Berkeley

How do we do computational research?

How do we do computational research?

1. write code

How do we do computational research?

1. write code
2. run code

How do we do computational research?

1. write code

2. run code

3. make figures

How do we do computational research?

1. write code
2. run code
3. make figures
4. write paper

How do we do computational research?

1. write code

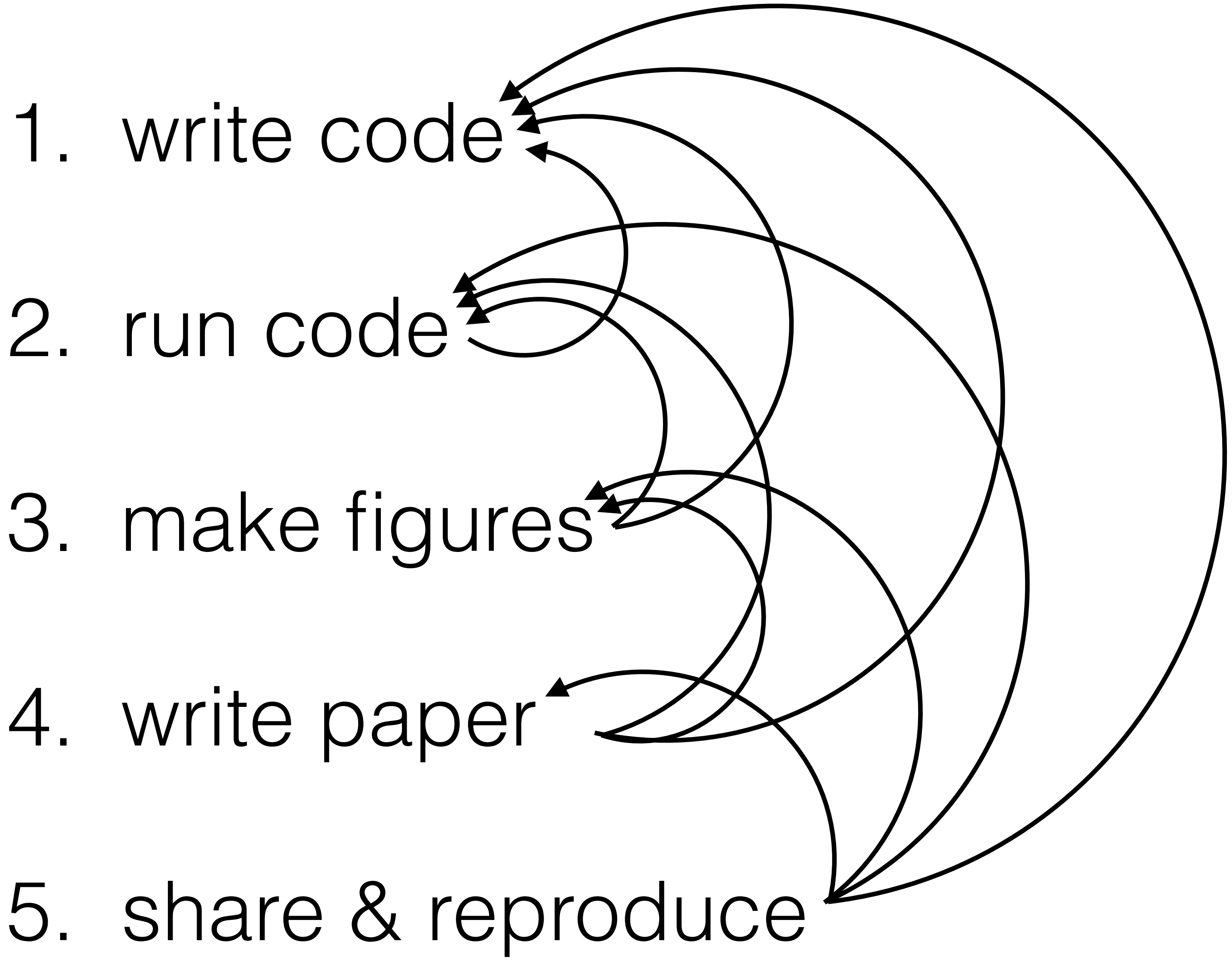
2. run code

3. make figures

4. write paper

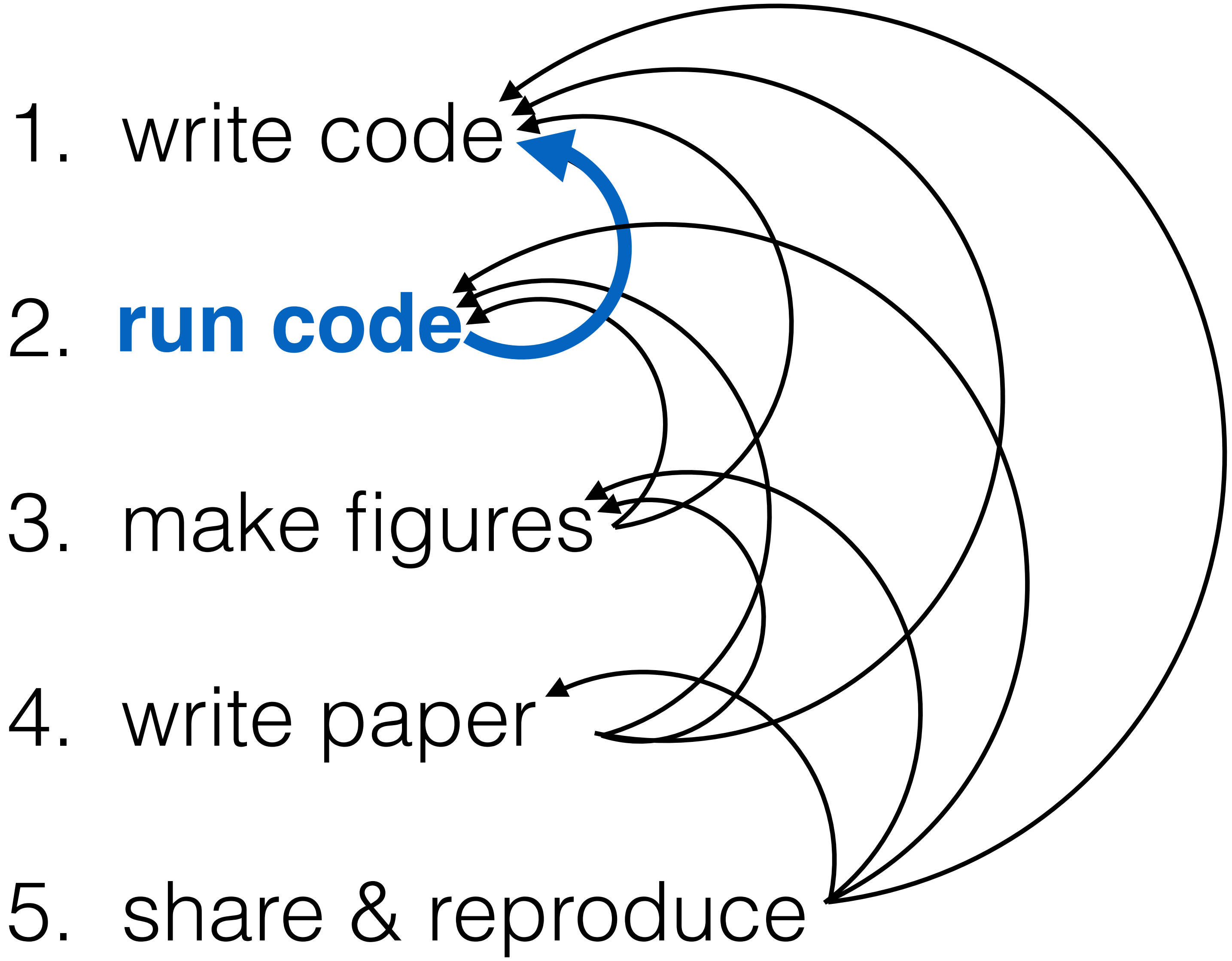
5. share & reproduce

How do we do computational research?



1. write code
2. run code
3. make figures
4. write paper
5. share & reproduce

How do we do computational research?

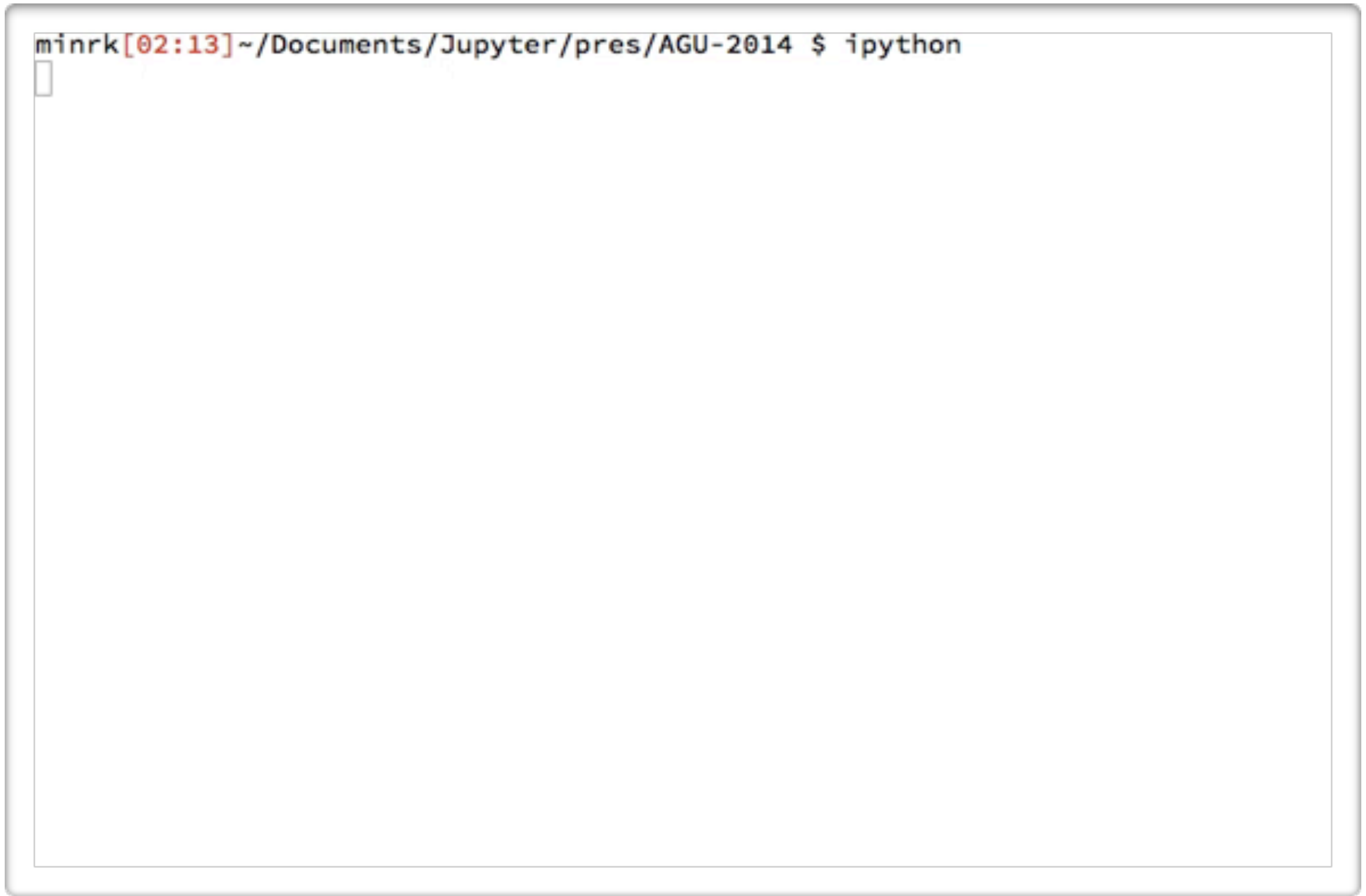


1. write code
2. **run code**
3. make figures
4. write paper
5. share & reproduce

IPython

IPython
Interactive Python

helps run code

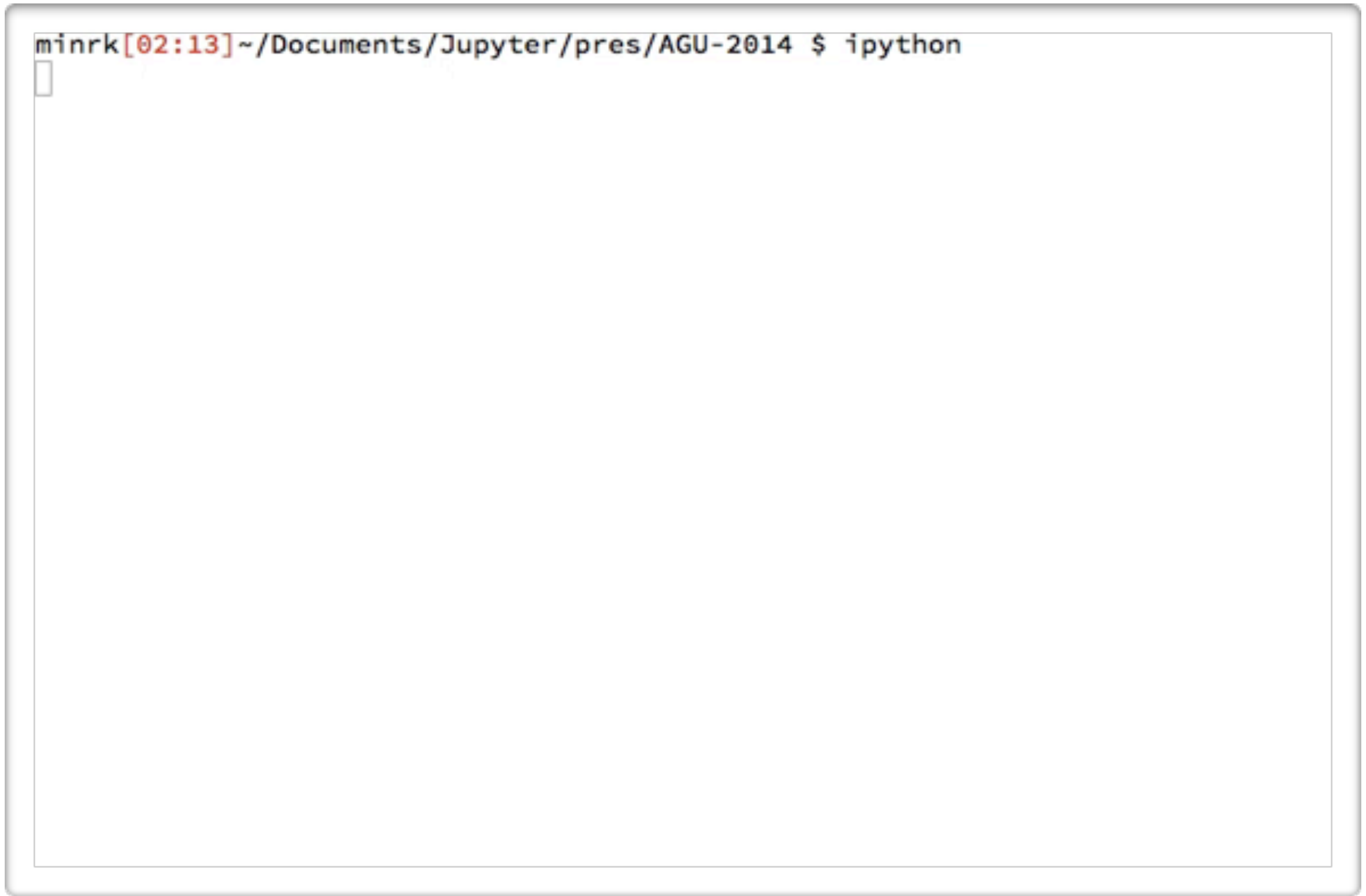


```
minrk[02:13]~/Documents/Jupyter/pres/AGU-2014 $ ipython
```

IPython
Interactive Python

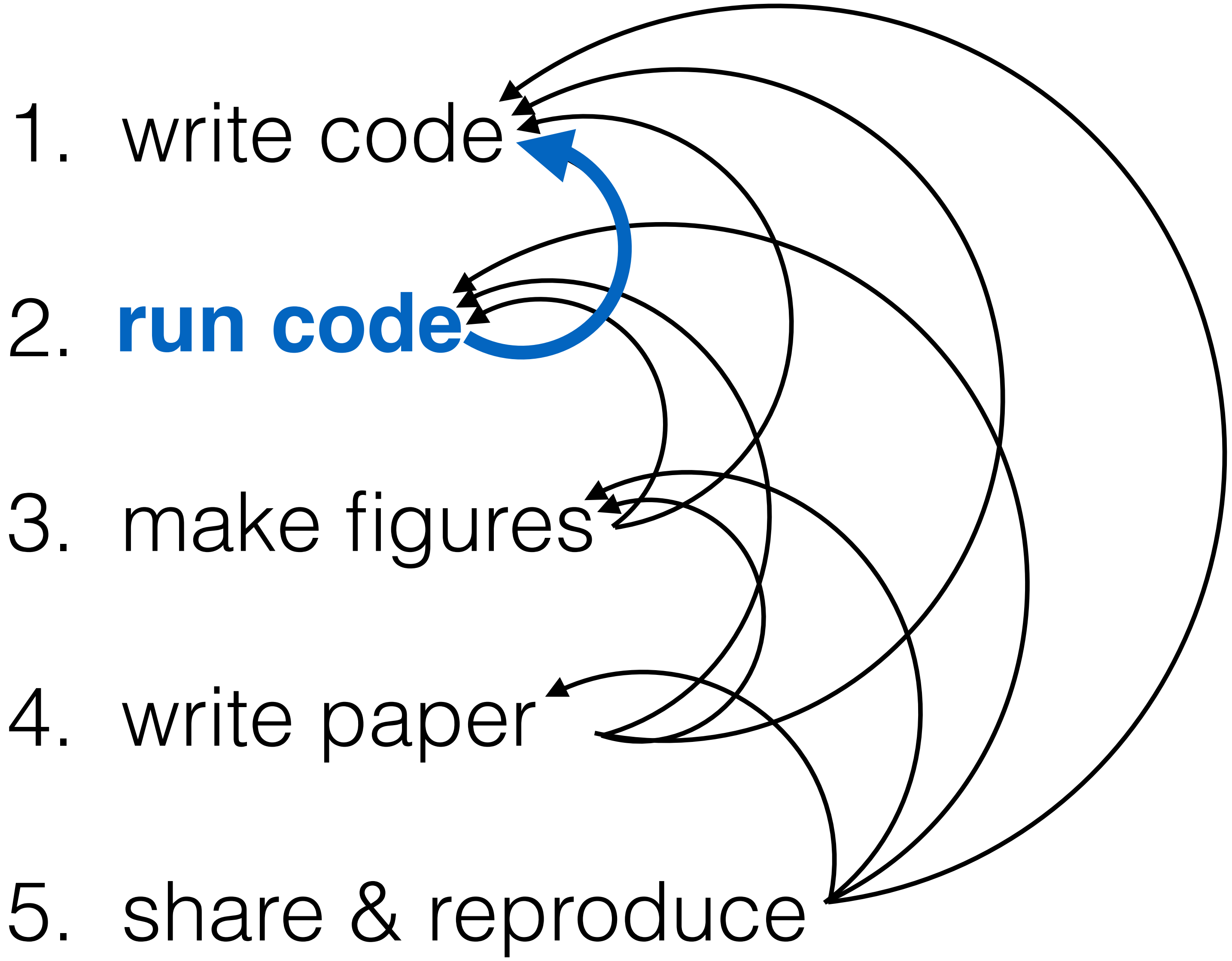
helps run code

- tab completion
- introspection
- %magics



```
minrk[02:13]~/Documents/Jupyter/pres/AGU-2014 $ ipython
```

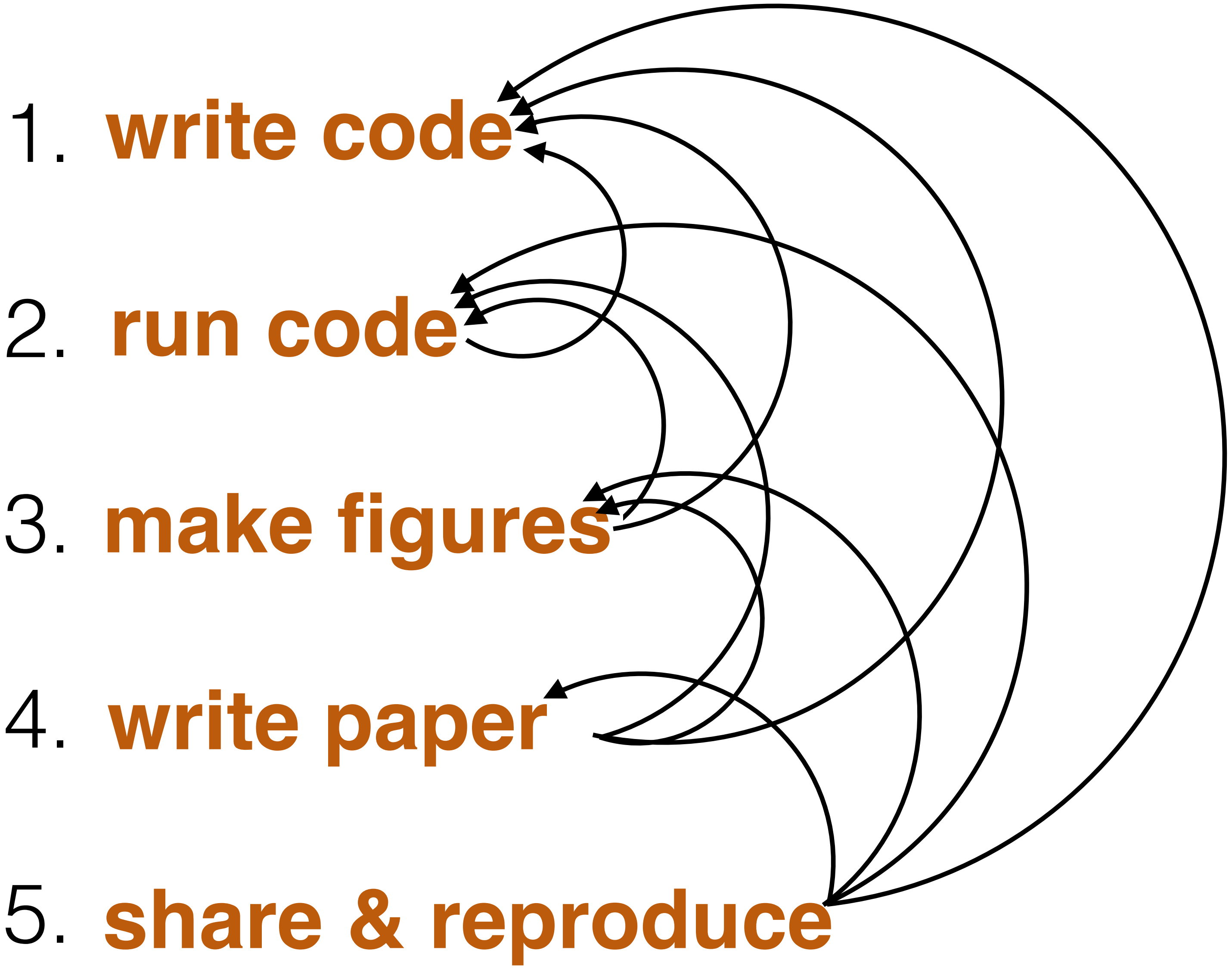
What about Jupyter?



1. write code
2. **run code**
3. make figures
4. write paper
5. share & reproduce

IPython

What about Jupyter?



1. **write code**
2. **run code**
3. **make figures**
4. **write paper**
5. **share & reproduce**

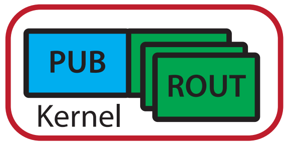
What is Jupyter?

What is Jupyter?

Rich REPL Protocol

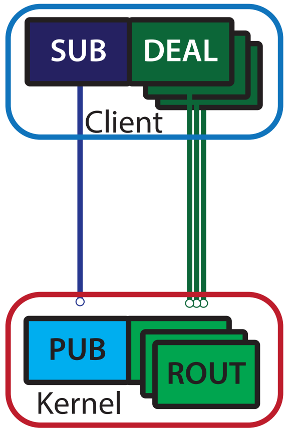
What is Jupyter?

Rich REPL Protocol



What is Jupyter?

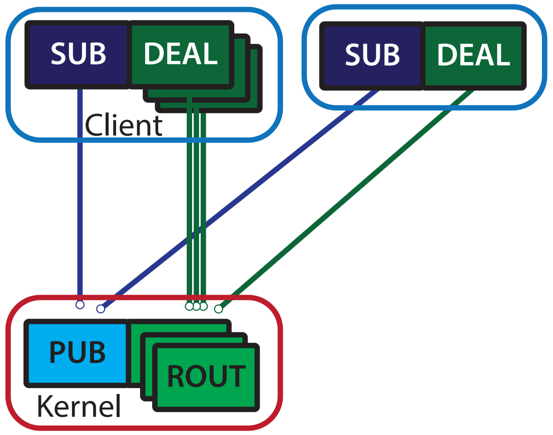
Rich REPL Protocol



ØMQ + JSON

What is Jupyter?

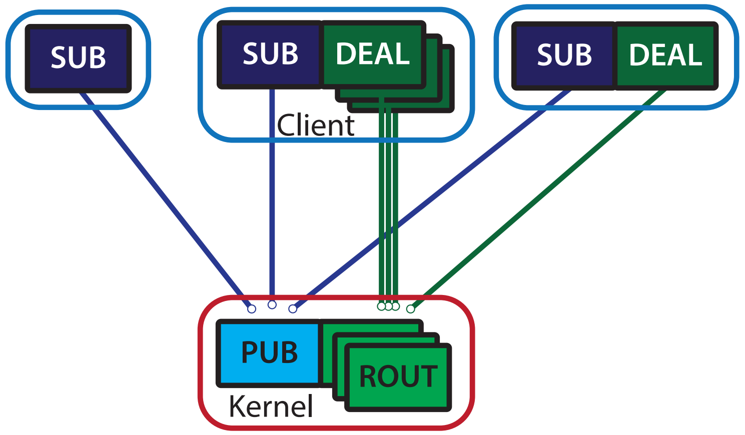
Rich REPL Protocol



ØMQ + JSON

What is Jupyter?

Rich REPL Protocol

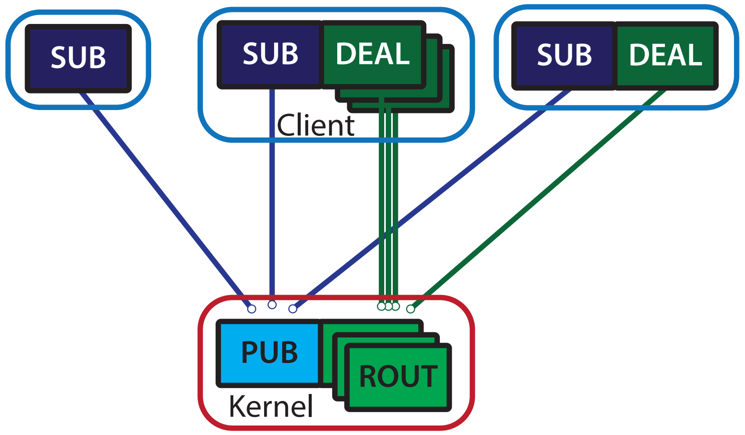


ØMQ + JSON

What is Jupyter?

Rich REPL Protocol

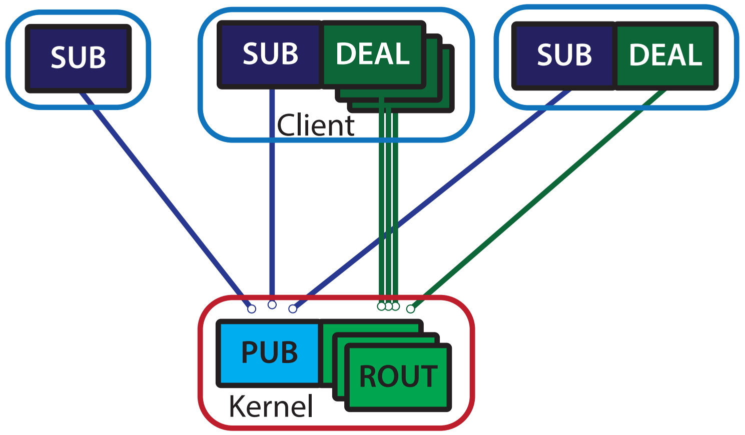
Document Format



ØMQ + JSON

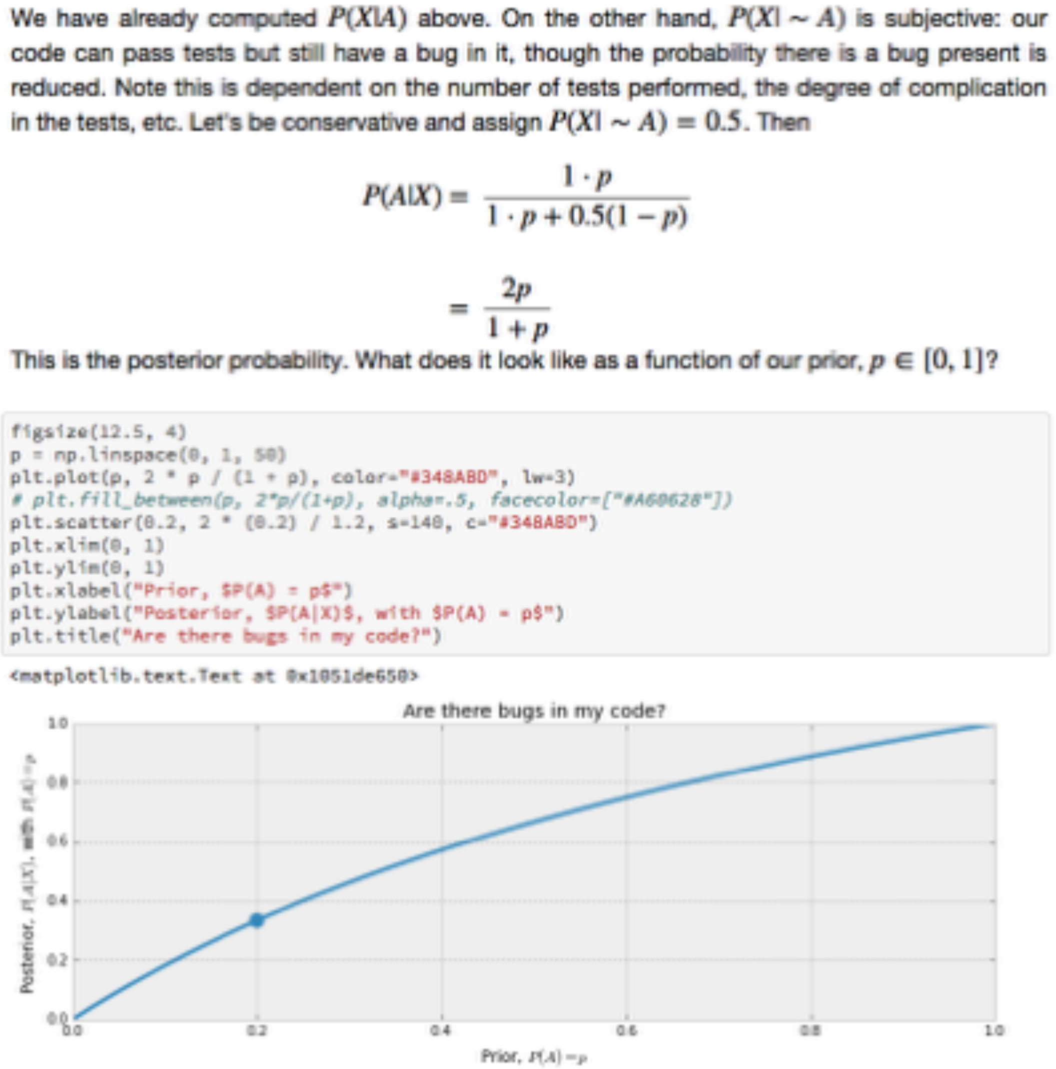
What is Jupyter?

Rich REPL Protocol



ØMQ + JSON

Document Format



We have already computed $P(X|A)$ above. On the other hand, $P(X|\sim A)$ is subjective: our code can pass tests but still have a bug in it, though the probability there is a bug present is reduced. Note this is dependent on the number of tests performed, the degree of complication in the tests, etc. Let's be conservative and assign $P(X|\sim A) = 0.5$. Then

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1-p)}$$

$$= \frac{2p}{1+p}$$

This is the posterior probability. What does it look like as a function of our prior, $p \in [0, 1]$?

```
figsize(12.5, 4)
p = np.linspace(0, 1, 50)
plt.plot(p, 2 * p / (1 + p), color="#348ABD", lw=3)
# plt.fill_between(p, 2*p/(1+p), alpha=.5, facecolor=["#A60628"])
plt.scatter(0.2, 2 * (0.2) / 1.2, s=140, c="#348ABD")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel("Prior, $P(A) = p$")
plt.ylabel("Posterior, $P(A|X)$, with $P(A) = p$")
plt.title("Are there bugs in my code?")
```

```
<matplotlib.text.Text at 0x1051de650>
```

Are there bugs in my code?
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Posterior, $P(A|X)$, with $P(A) = p$
Prior, $P(A) = p$

https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers

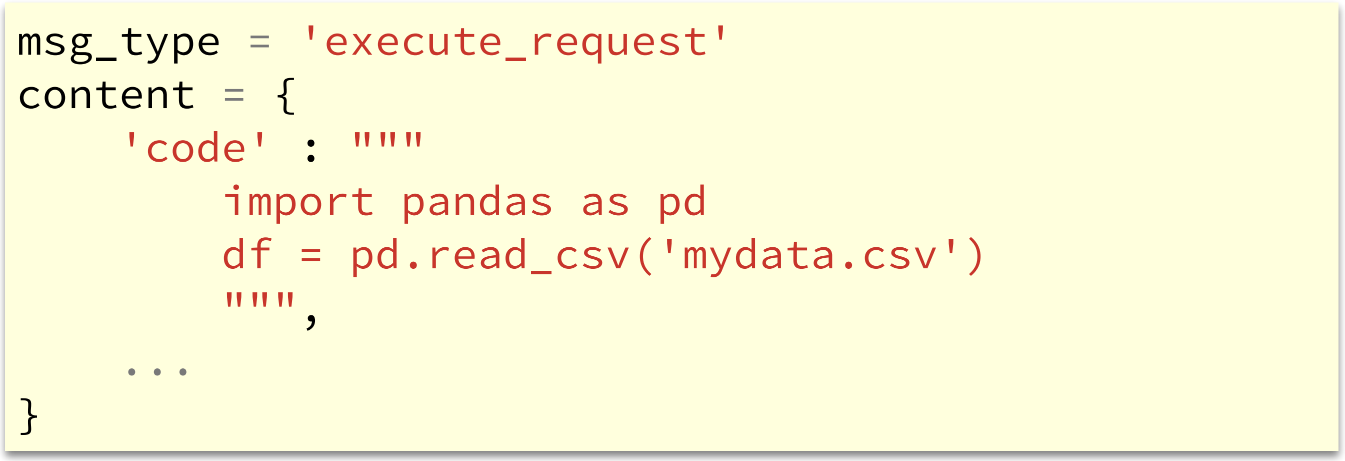
Jupyter Protocol

REP*L over JSON + ØMQ

Jupyter Protocol

REP*L over JSON + ØMQ

- Read



```
msg_type = 'execute_request'
content = {
    'code' : """
        import pandas as pd
        df = pd.read_csv('mydata.csv')
        """,
    ...
}
```

Jupyter Protocol

REP*L over JSON + ØMQ

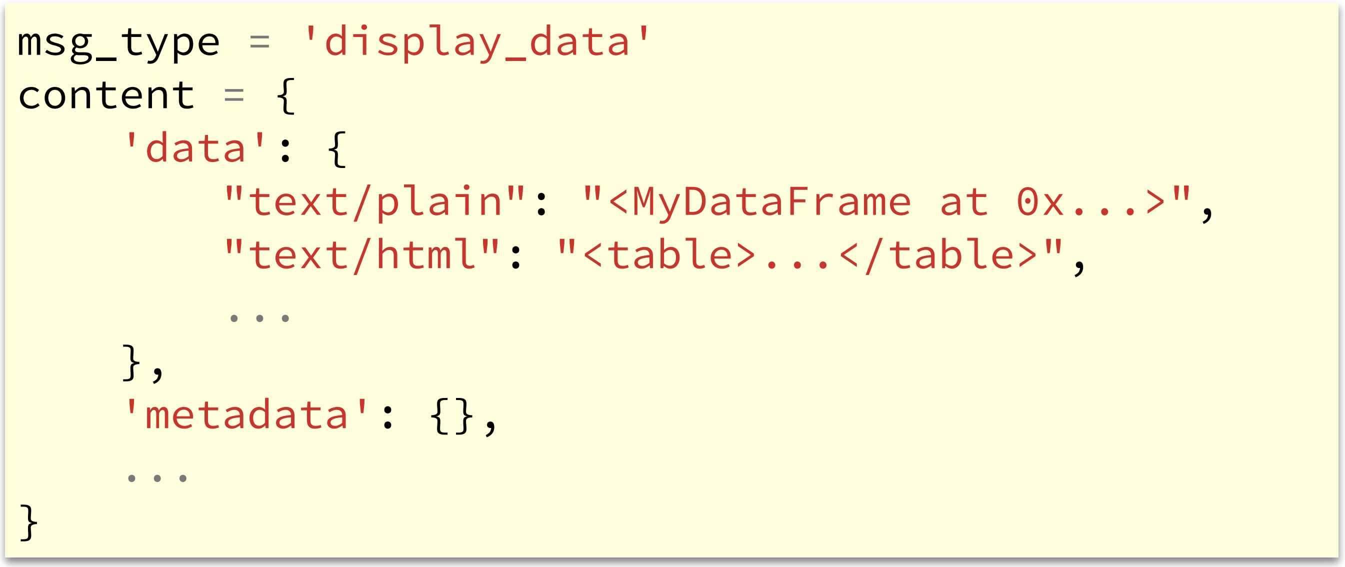
- Read
- Eval

```
msg_type = 'execute_reply'
content = {
    'execution_count': 3,
    'status': 'ok',
    ...
}
```

Jupyter Protocol

REP*L over JSON + ØMQ

- Read
- Eval
- Print*

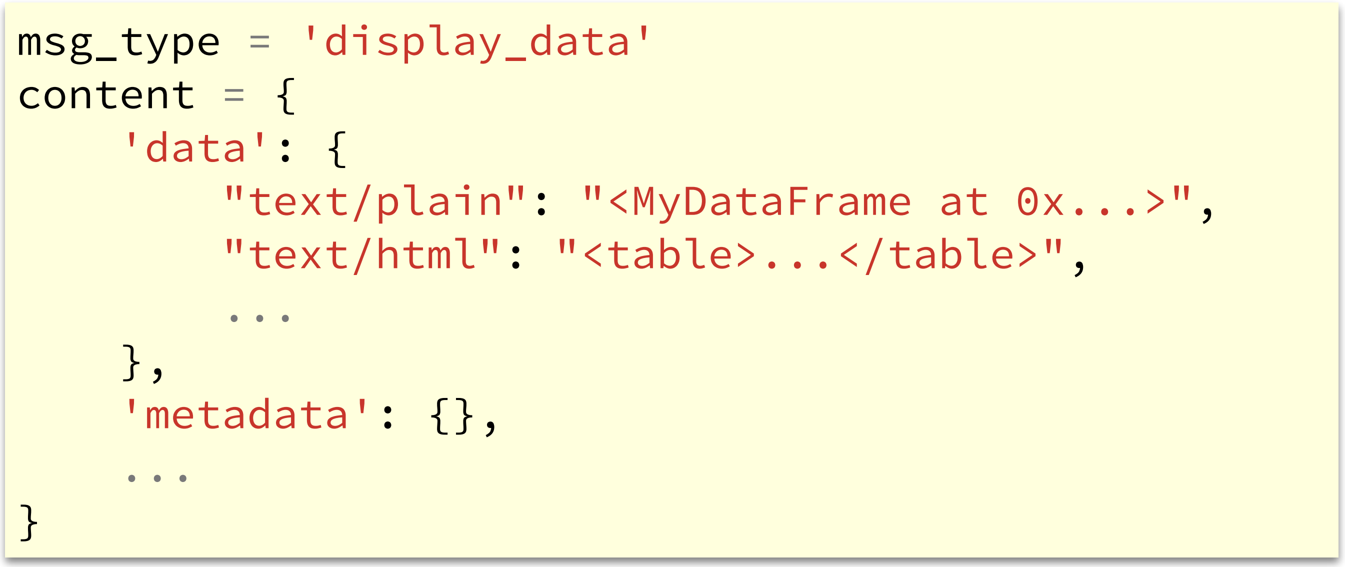


```
msg_type = 'display_data'
content = {
    'data': {
        "text/plain": "<MyDataFrame at 0x...>",
        "text/html": "<table>...</table>",
        ...
    },
    'metadata': {},
    ...
}
```

Jupyter Protocol

REP*L over JSON + ØMQ

- Read
- Eval
- Print*
- Loop



```
msg_type = 'display_data'
content = {
    'data': {
        "text/plain": "<MyDataFrame at 0x...>",
        "text/html": "<table>...</table>",
        ...
    },
    'metadata': {},
    ...
}
```

Jupyter Protocol

supercharge the P in REP*L

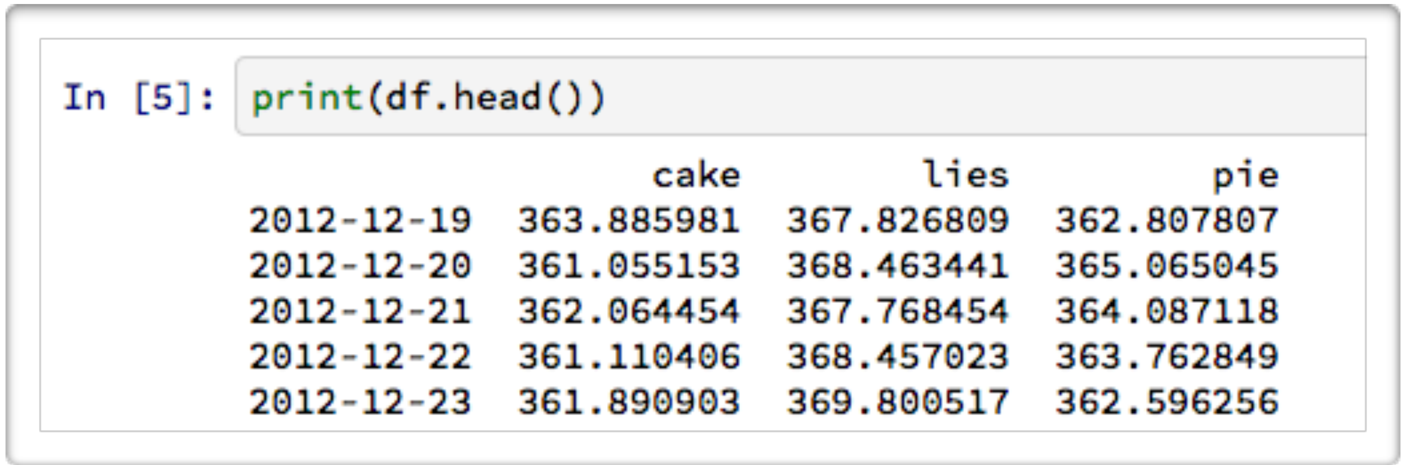
any mime-type output

Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text



```
In [5]: print(df.head())
```

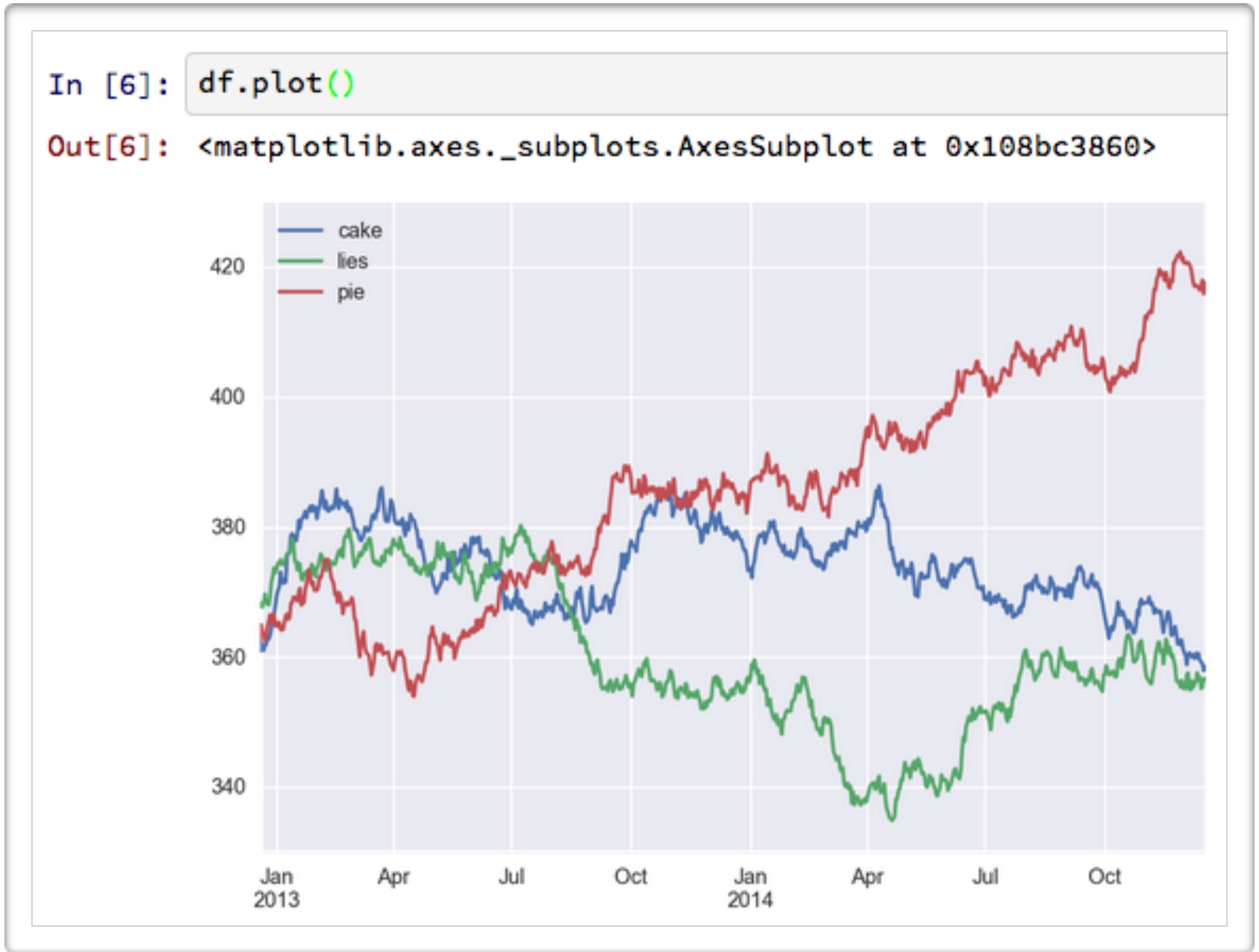
```
                  cake        lies         pie
2012-12-19  363.885981  367.826809  362.807807
2012-12-20  361.055153  368.463441  365.065045
2012-12-21  362.064454  367.768454  364.087118
2012-12-22  361.110406  368.457023  363.762849
2012-12-23  361.890903  369.800517  362.596256
```

Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg



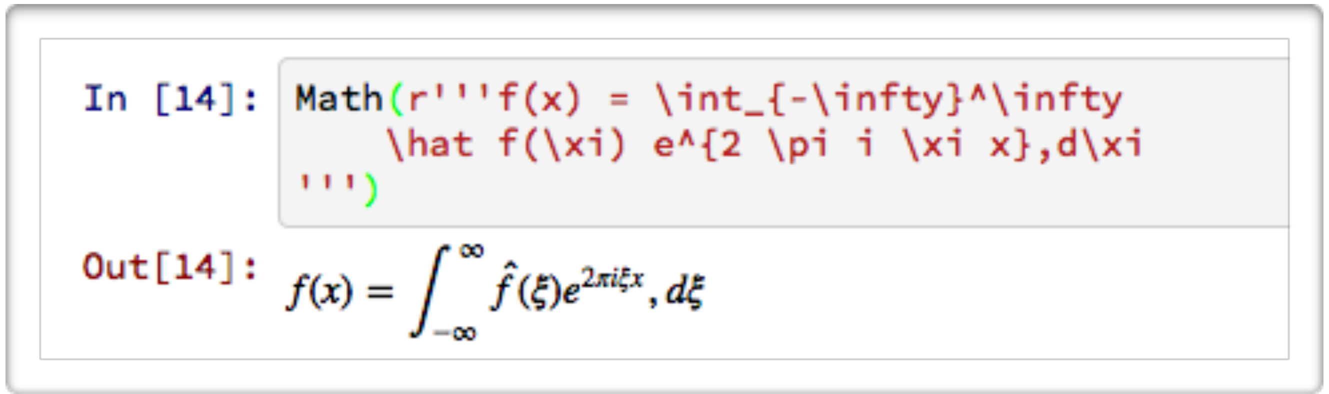
```
In [6]: df.plot()
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x108bc3860>
```

Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg
- latex, pdf



```
In [14]: Math(r'''f(x) = \int_{-\infty}^\infty
             \hat f(\xi) e^{2 \pi i \xi x},d\xi
         ''')
```

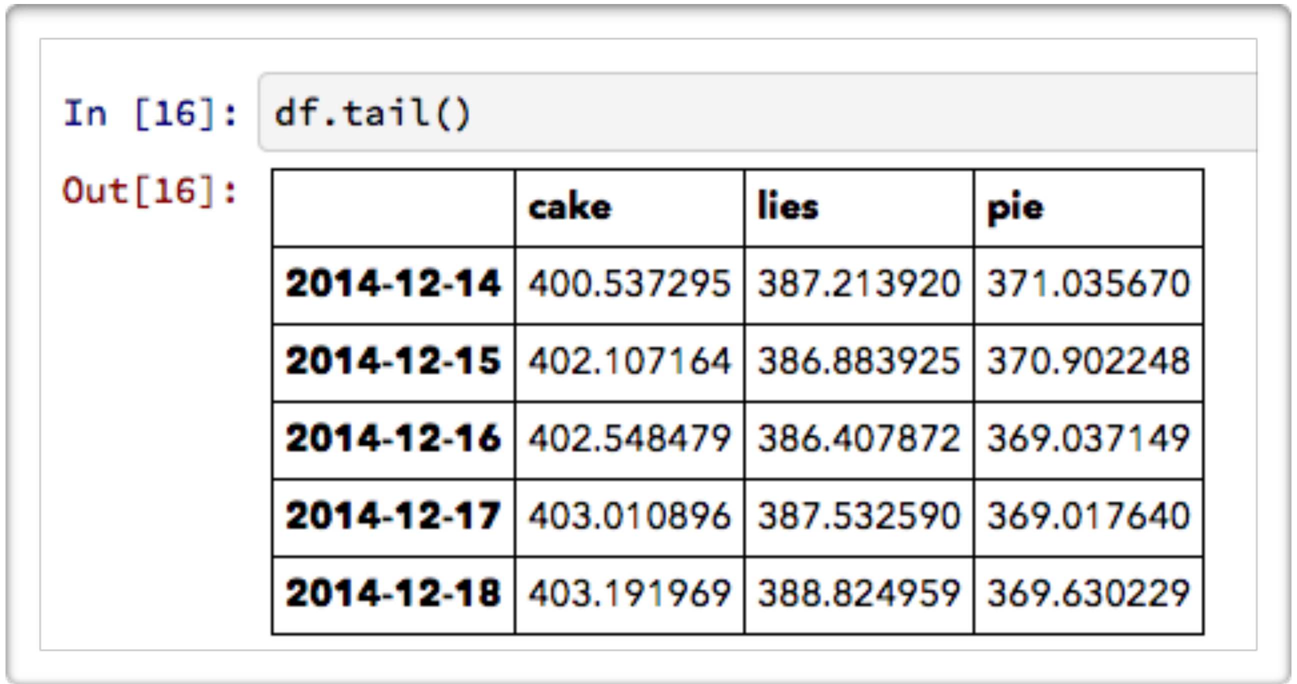
Out[14]: $f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x}, d\xi$

Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg
- latex, pdf
- html, javascript



```
In [16]: df.tail()
```

Out[16]:

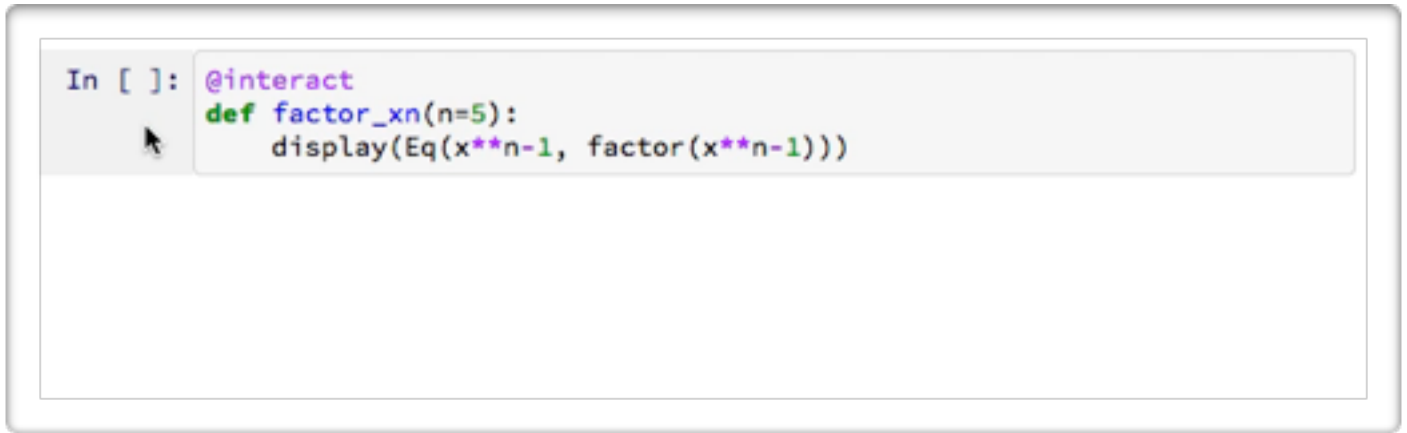
	cake	lies	pie
2014-12-14	400.537295	387.213920	371.035670
2014-12-15	402.107164	386.883925	370.902248
2014-12-16	402.548479	386.407872	369.037149
2014-12-17	403.010896	387.532590	369.017640
2014-12-18	403.191969	388.824959	369.630229

Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg
- latex, pdf
- html, javascript
- interactive widgets



```
In [ ]: @interact
        def factor_xn(n=5):
            display(Eq(x**n-1, factor(x**n-1)))
```

Jupyter Protocol

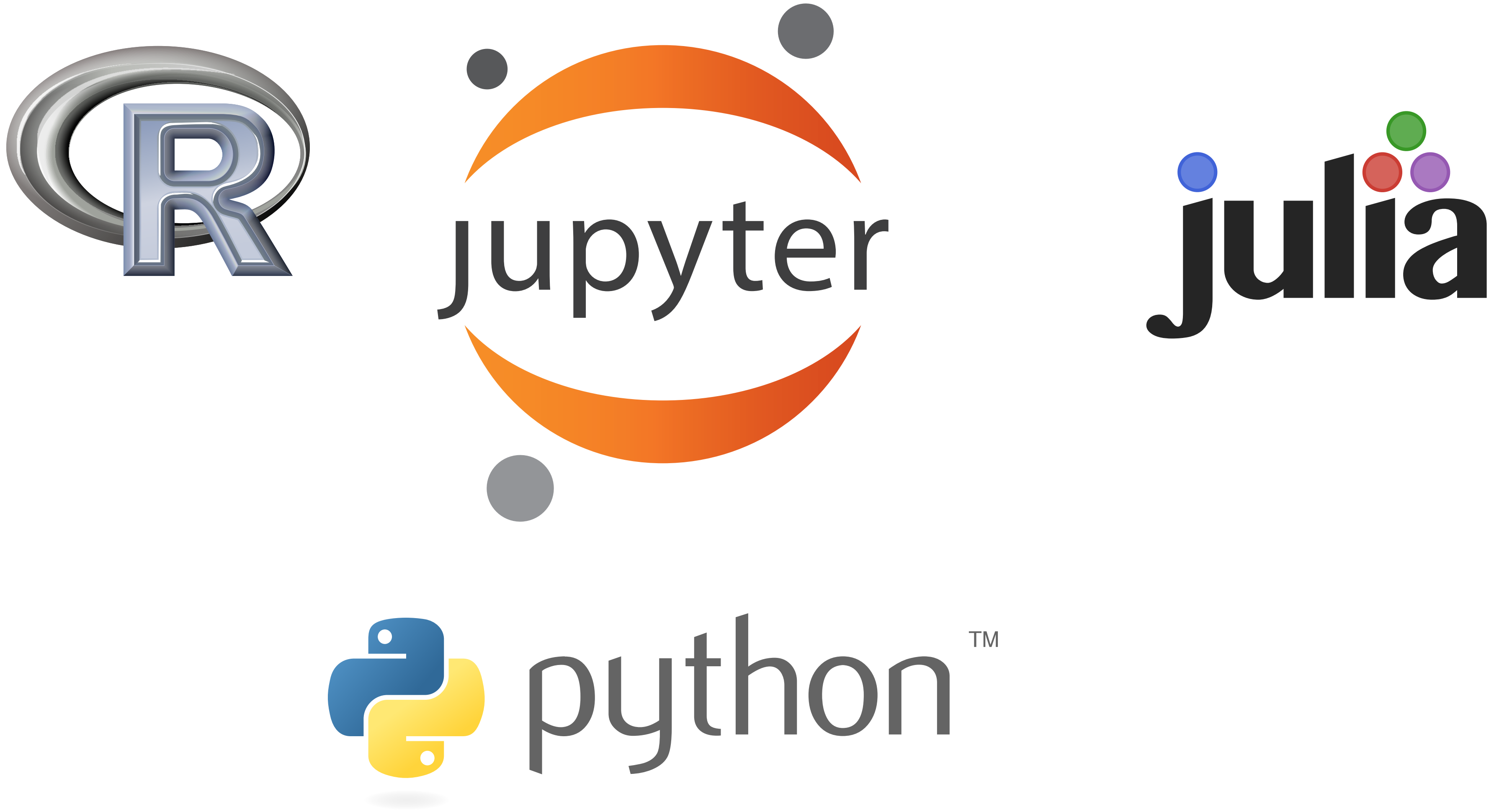
is language agnostic

Jupyter Protocol

is language agnostic

Jupyter Protocol

is language agnostic



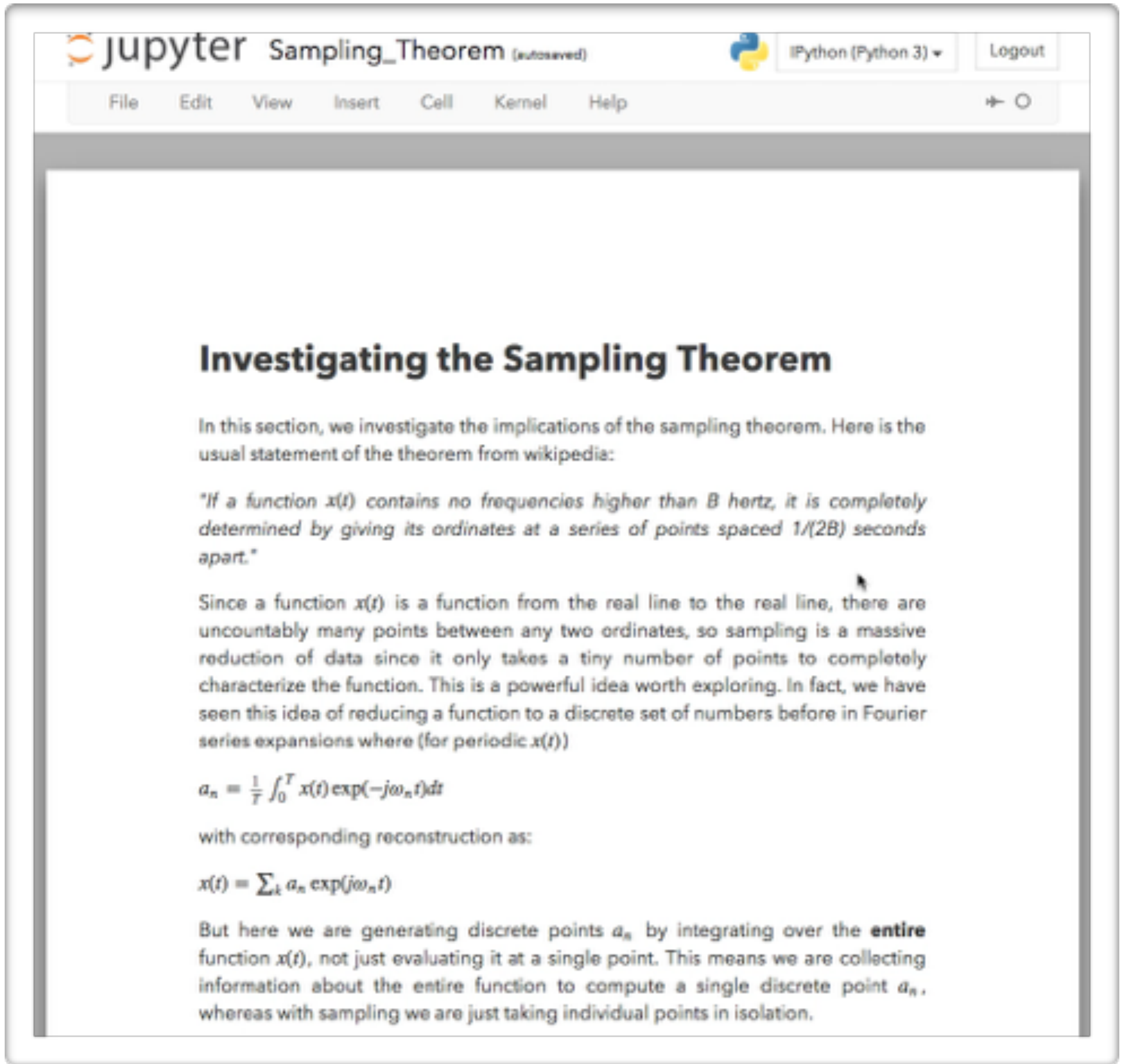
Jupyter Protocol

is language agnostic

Scala
perl
R
jupyter
julia
Spark
ERLANG
python ™

Jupyter Notebooks

- notebook = sequence of cells



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

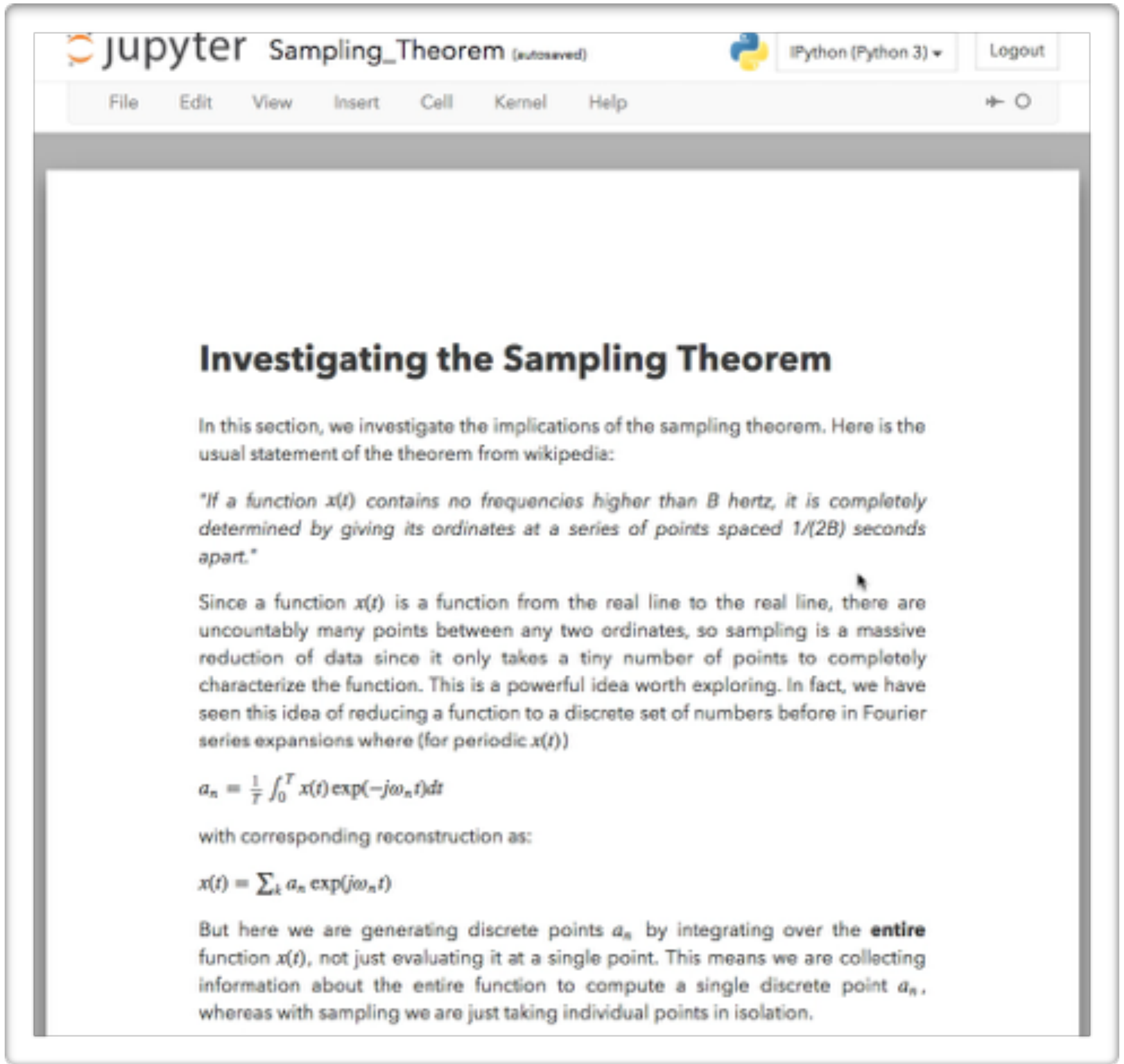
$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- notebook = sequence of cells
- text cell = markdown + latex)



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

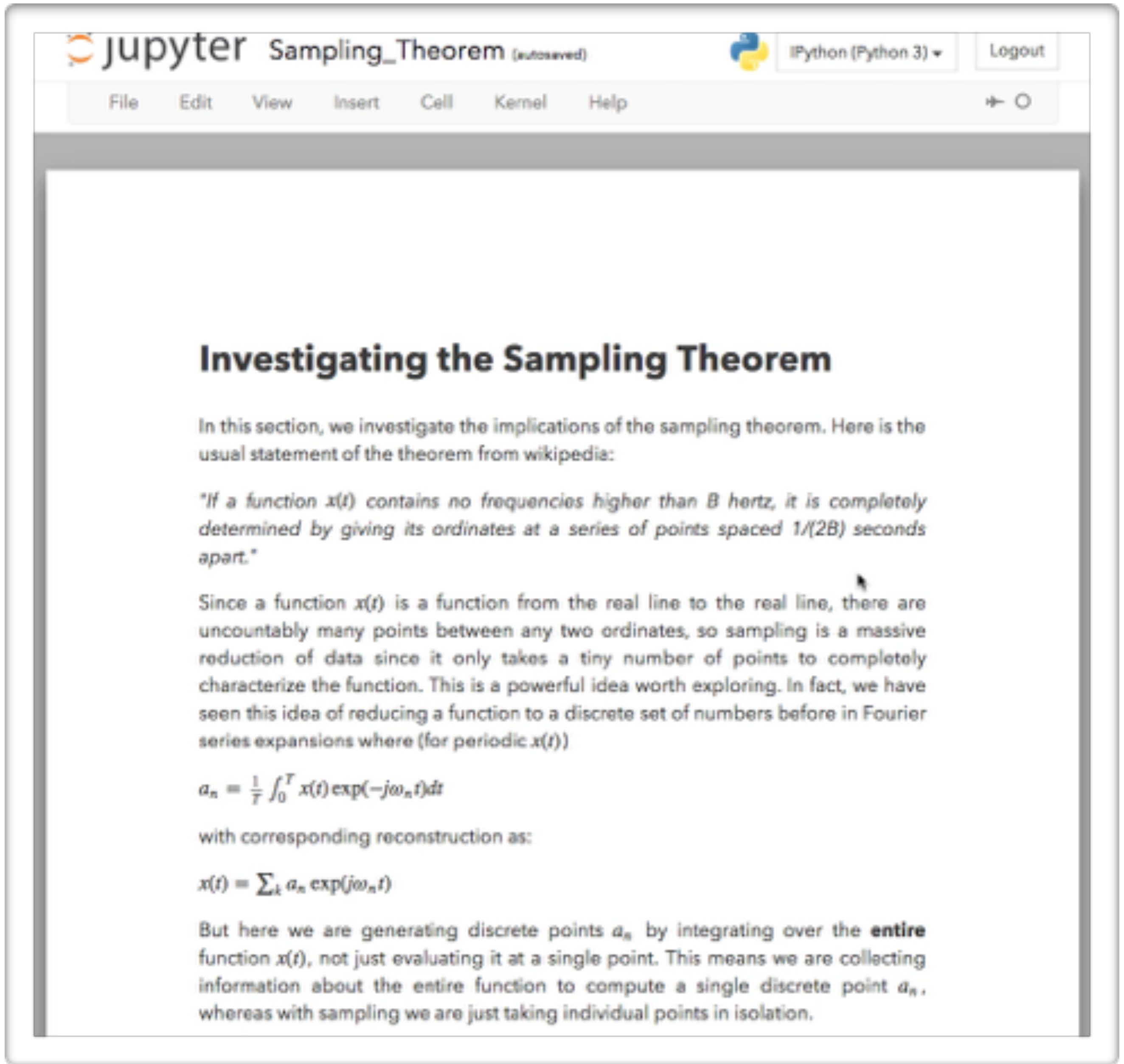
$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- notebook = sequence of cells
- text cell = markdown + latex)
- code cell = REP (input + output)



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

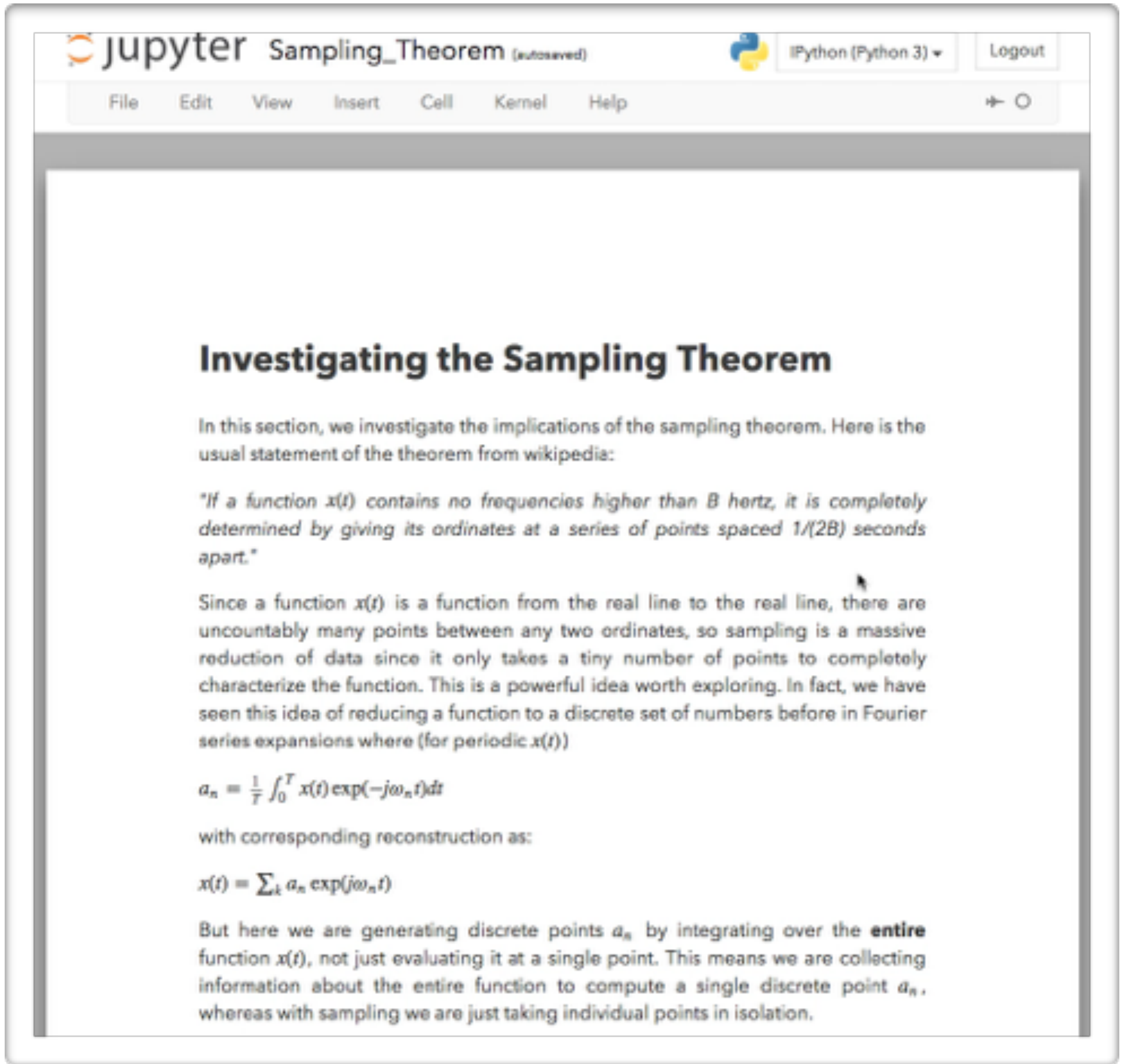
$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- notebook = sequence of cells
- text cell = markdown + latex)
- code cell = REP (input + output)
- metadata everywhere



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

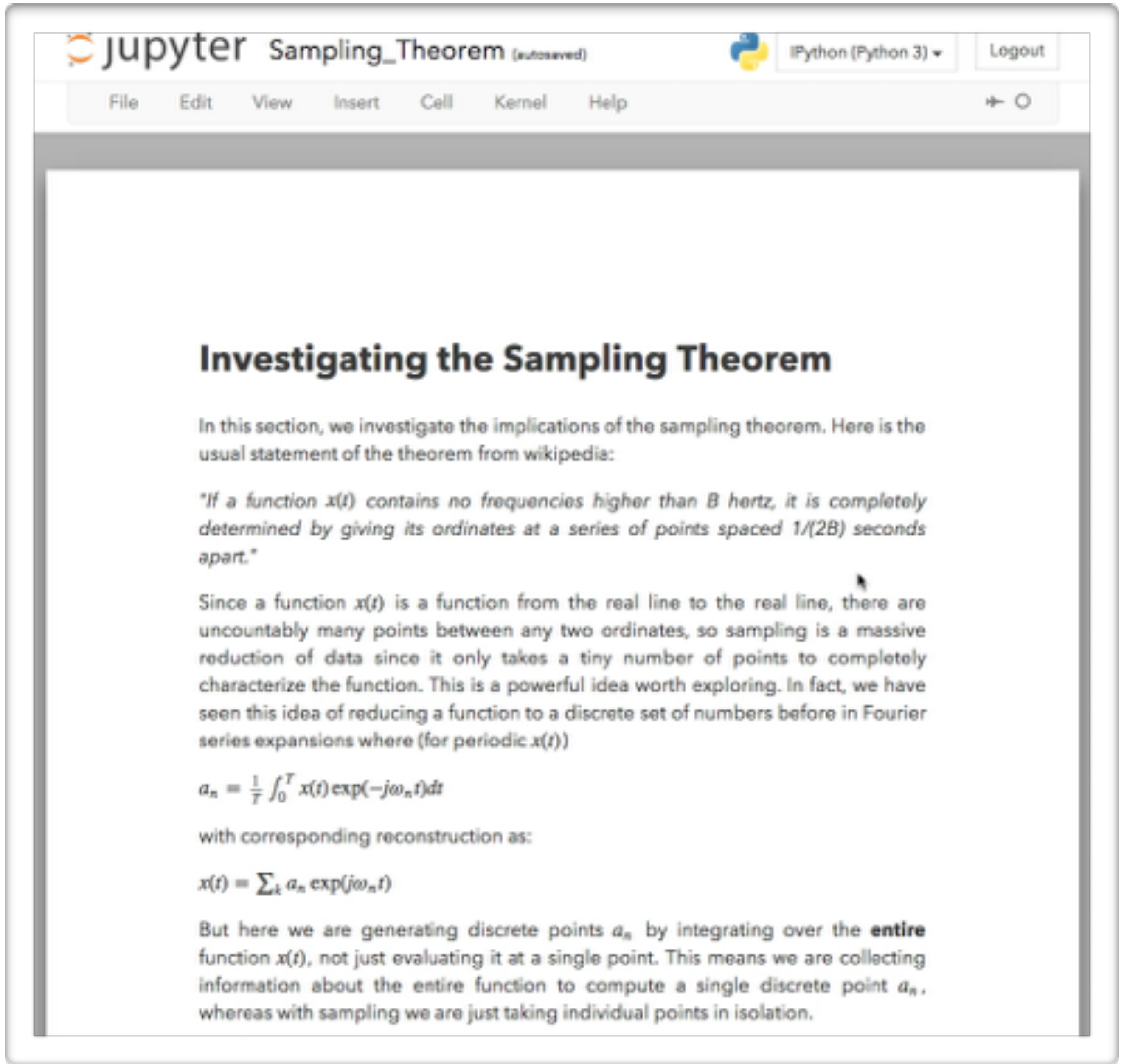
$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- Plain Text (JSON)



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

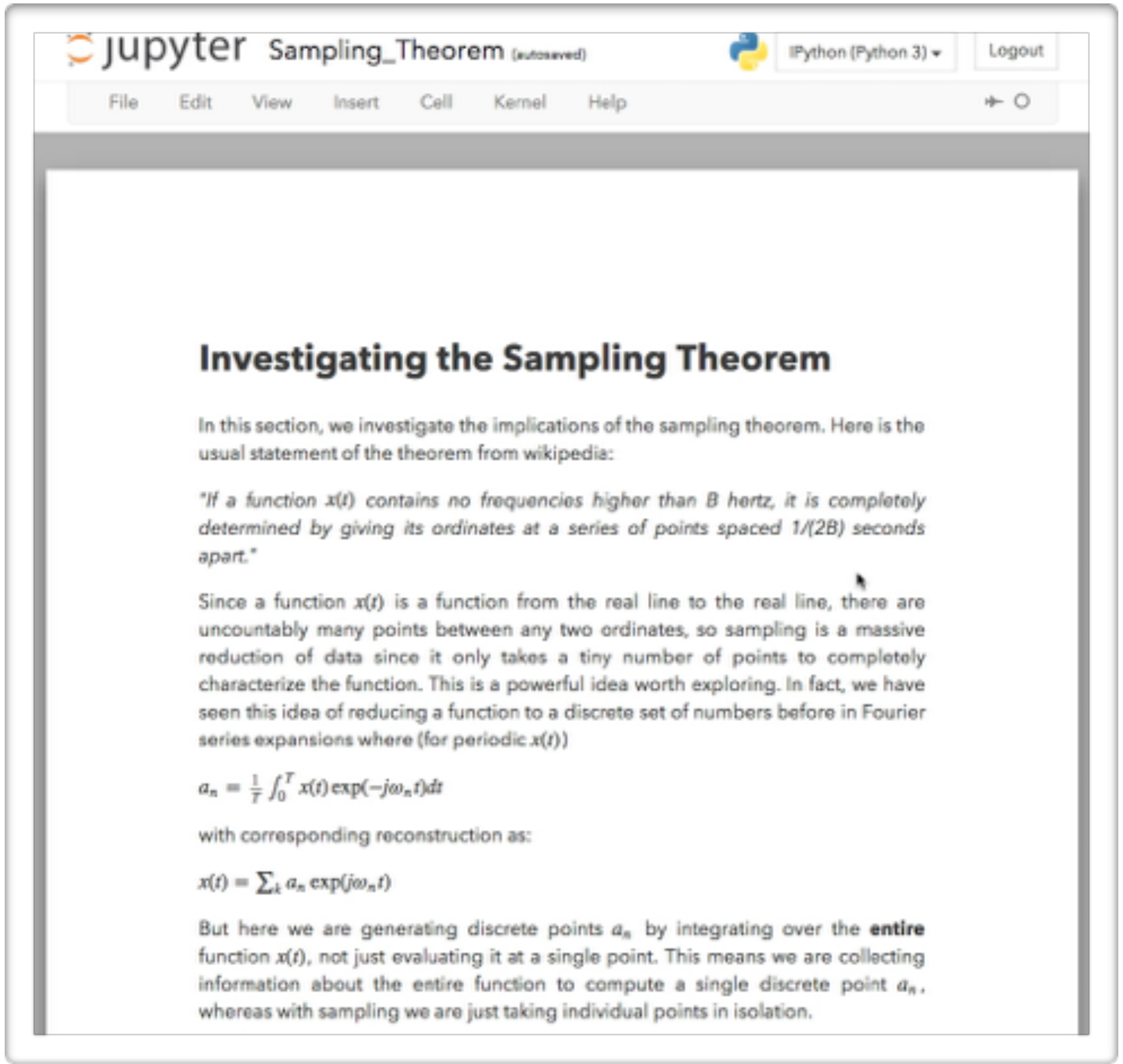
$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- Plain Text (JSON)
- Publicly documented schema



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

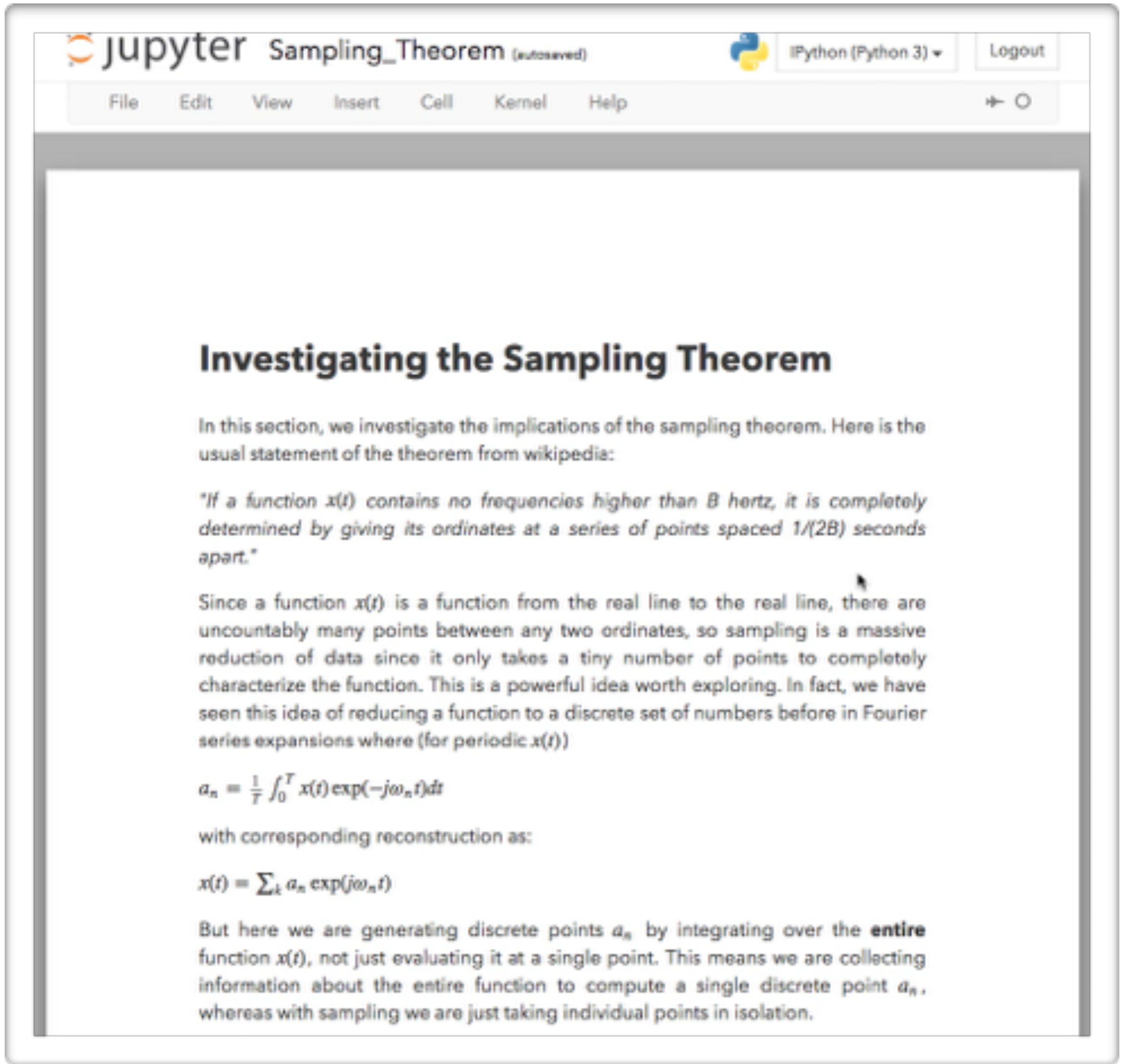
$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- Plain Text (JSON)
- Publicly documented schema
- Machine readable, easy to understand



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function x(t) contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

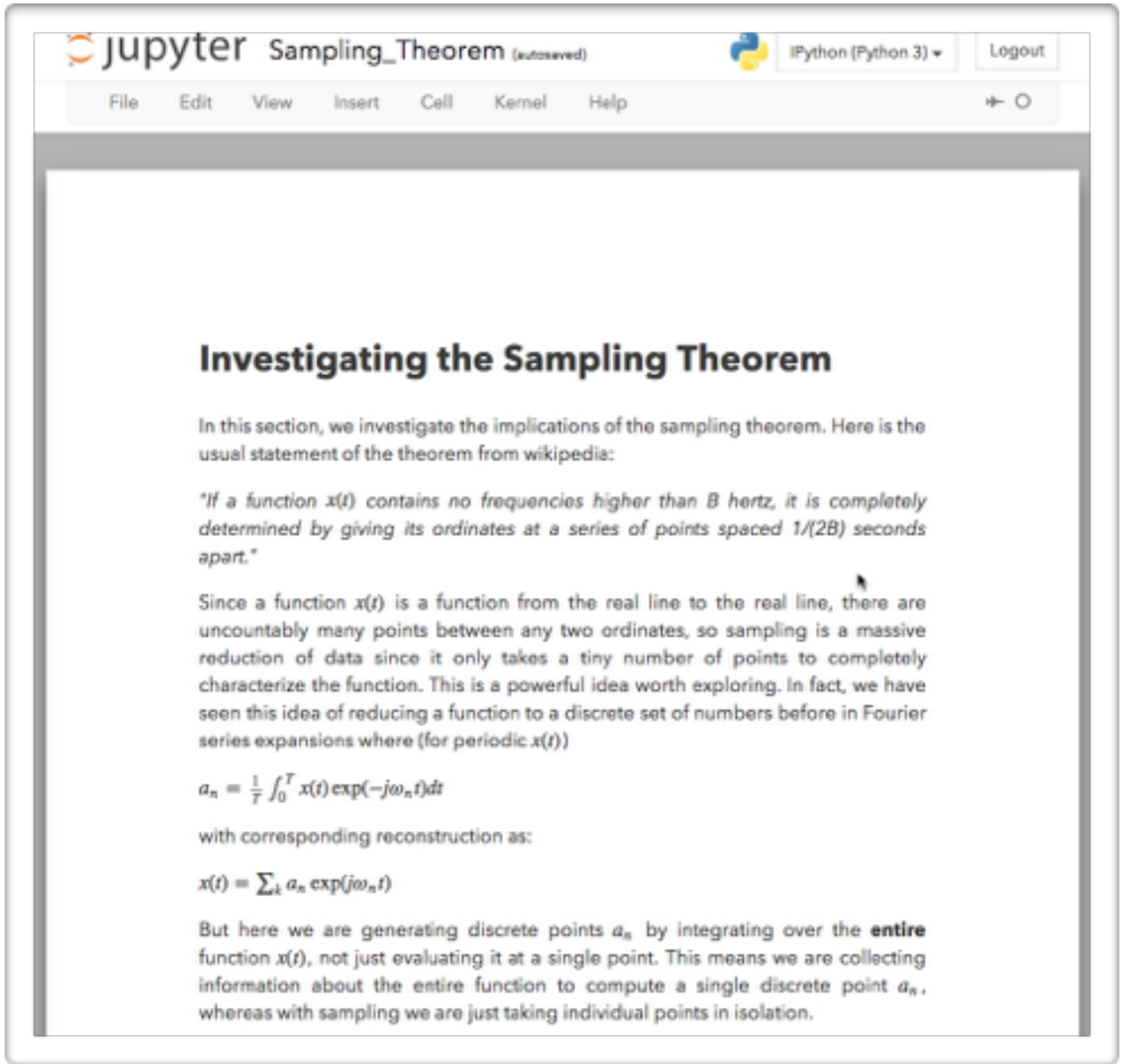
$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- Plain Text (JSON)
- Publicly documented schema
- Machine readable, easy to understand
- Transformable (nbconvert)



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

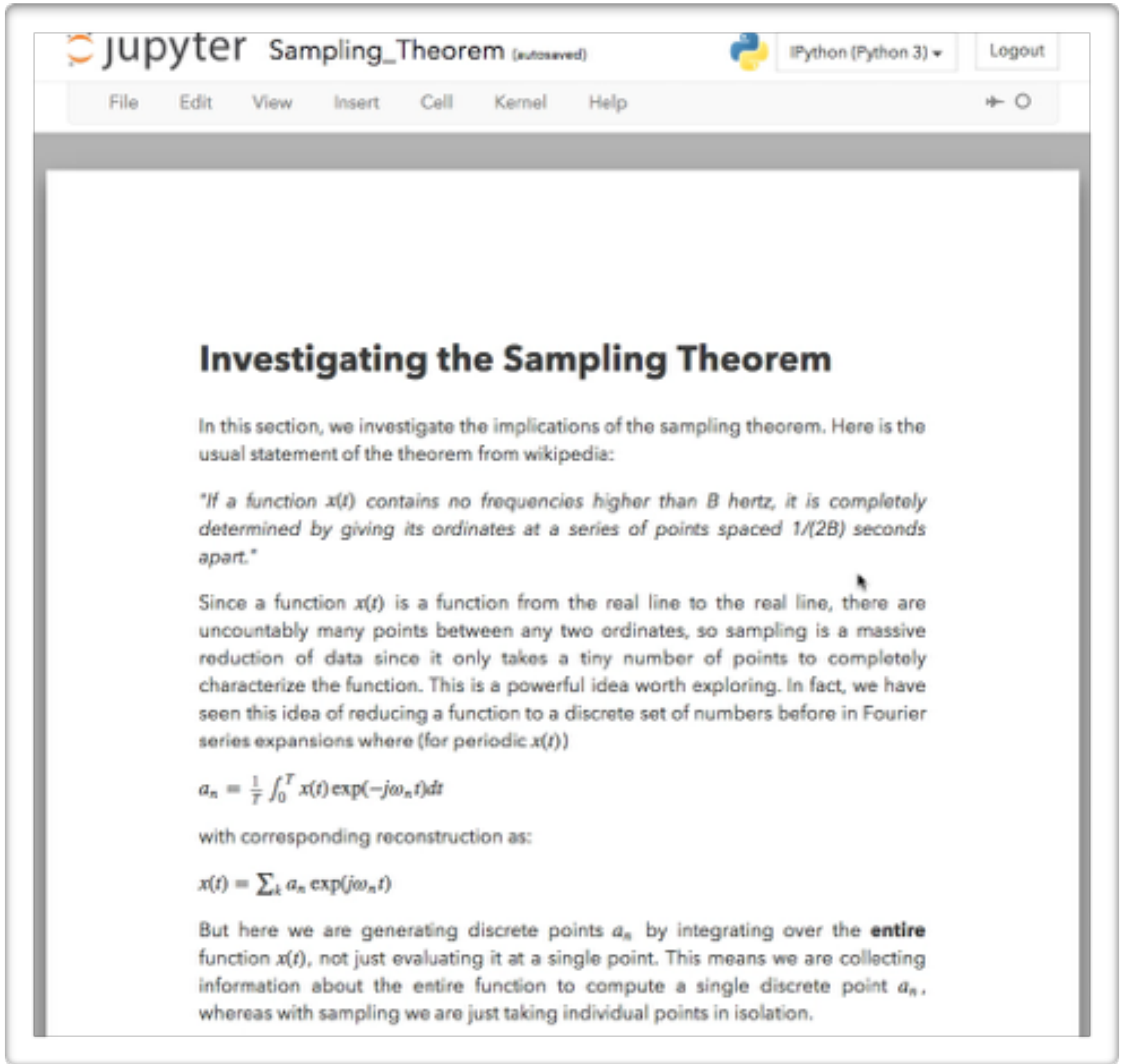
$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- interactive environment



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

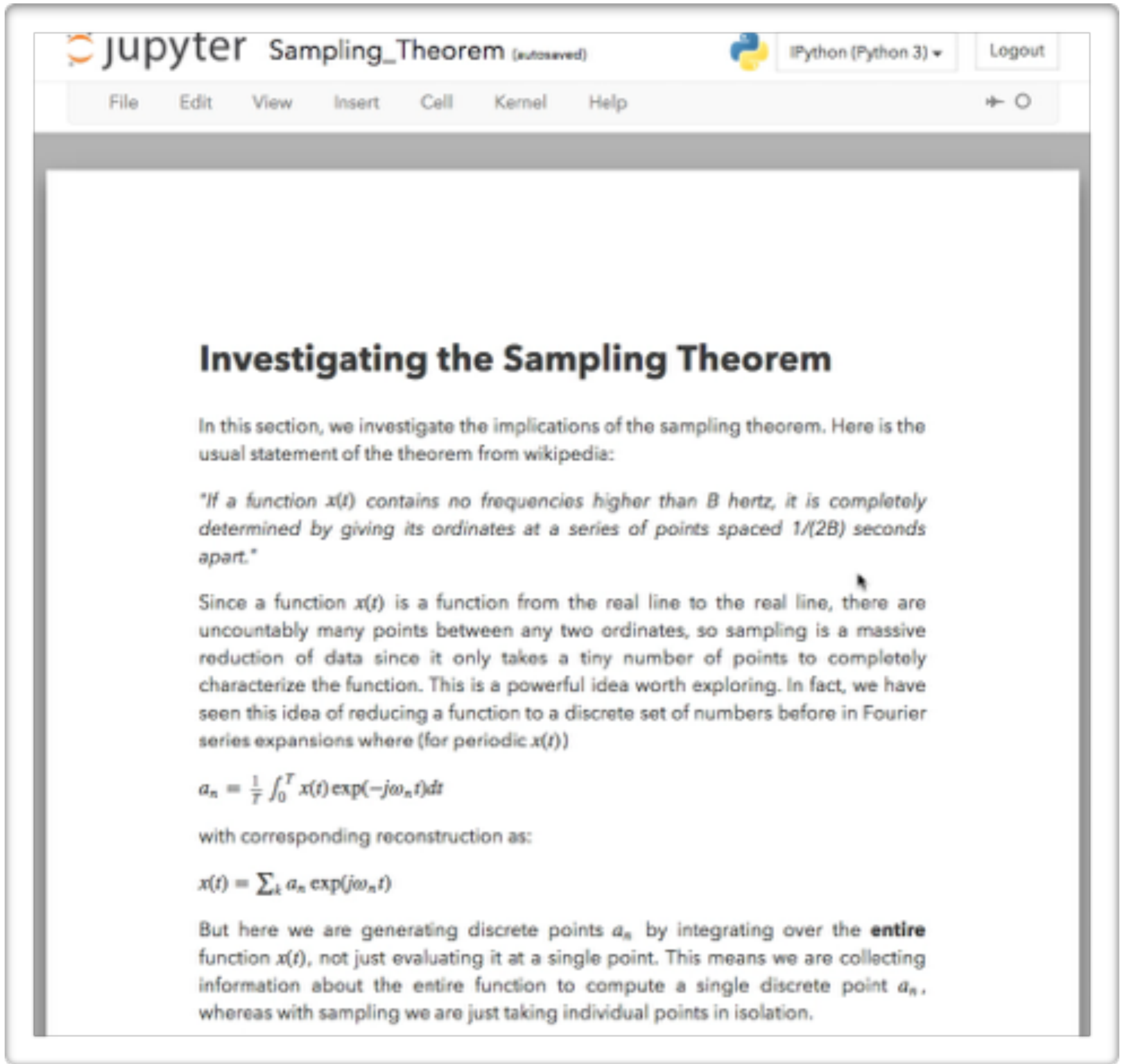
$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- interactive environment
- input format



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

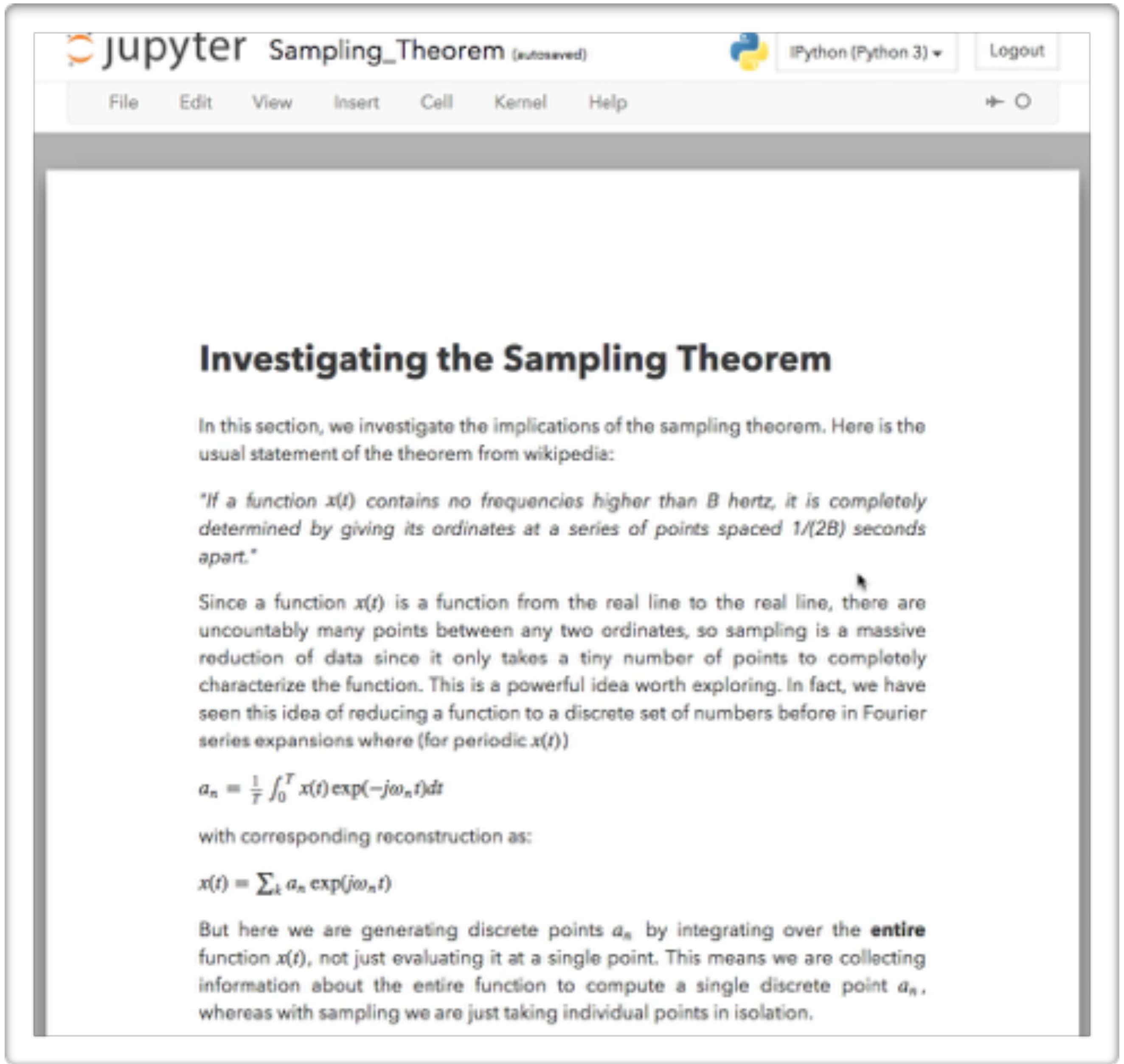
$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Jupyter Notebooks

- interactive environment
- input format
- output format



Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$

with corresponding reconstruction as:

$x(t) = \sum_k a_n \exp(j\omega_n t)$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n, whereas with sampling we are just taking individual points in isolation.

https://github.com/unpingco/Python-for-Signal-Processing

Lifecycle of a Computational Idea

Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook

Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook

2. *Build/add to a library based on what you learn*

Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook
2. *Build/add to a library based on what you learn*
3. Record and collaborate on analyses in Notebooks

Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook

2. Build/add to a library based on what you learn

3. Record and collaborate on analyses in Notebooks

4. Document, demonstrate, and share in Notebooks

Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook
2. *Build/add to a library based on what you learn*
3. Record and collaborate on analyses in Notebooks
4. Document, demonstrate, and share in Notebooks
5. Computational companions, reproducible papers

Applications of Jupyter Notebooks

- **nbconvert** - convert notebooks to other formats (rst, html, latex/pdf, markdown, script, reveal.js slides)
- **nbviewer** - nbconvert to html on the web
- **nbgrader** - automated grading of notebooks
- **tmpnb** - containerized (docker) transient deployments of notebooks
- **thebe** - transient kernels on the web, without notebooks
- **dexy** - reproducible document-based workflows
- **jupyterhub** - multi-user notebook server for classes, groups

Brian Granger

Jonathan Frederic
Kyle Kelley

Fernando Perez

Matthias Bussonier
Jessica Hamrick
Thomas Kluyver

Brian Granger

Fernando Perez

Jonathan Frederic
Kyle Kelley

Matthias Bussonier
Jessica Hamrick
Thomas Kluyver