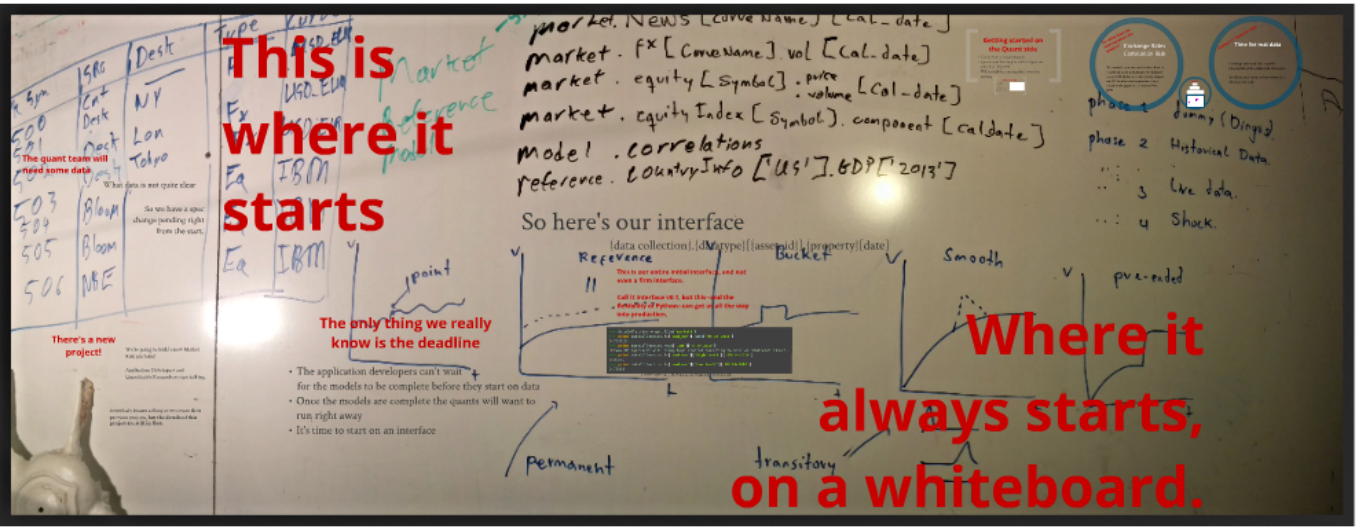
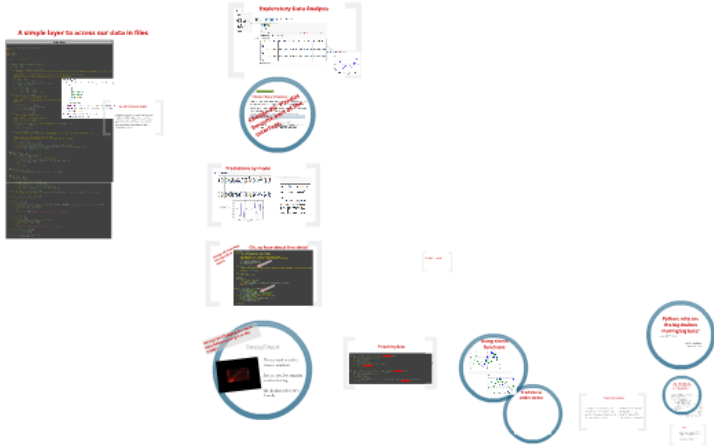

This is where it starts
The quant team will need some data
There's a new project!
The only thing we really know is the deadline
• The application developers can't wait for the models to be complete before they start on data
• Once the models are complete the quants will want to run right away
• It's time to start on an interface
So here's our interface
Reference
Bucket
Smooth
pre-cached
Permanent
transitory
Where it always starts, on a whiteboard.



Python: why are the big dealers making big bets?

QCON NY 2014

Andy Fundinger
Mario Morales

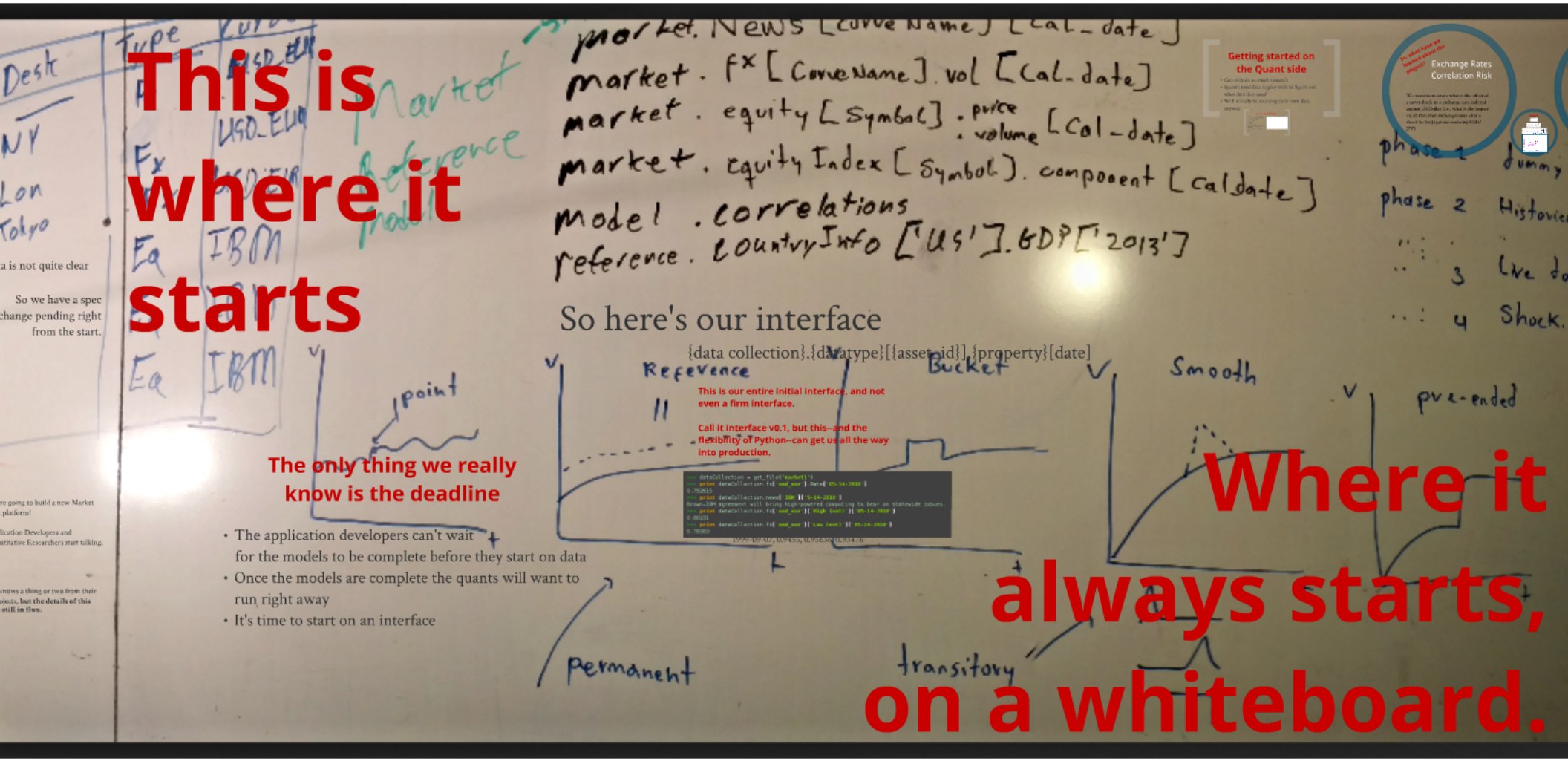
At this point 4 of the top 5 banks not only have Python projects underway but are publicly hiring for them.

Why are they doing this? What makes this language strong enough to not just use it, but to build new teams around it?

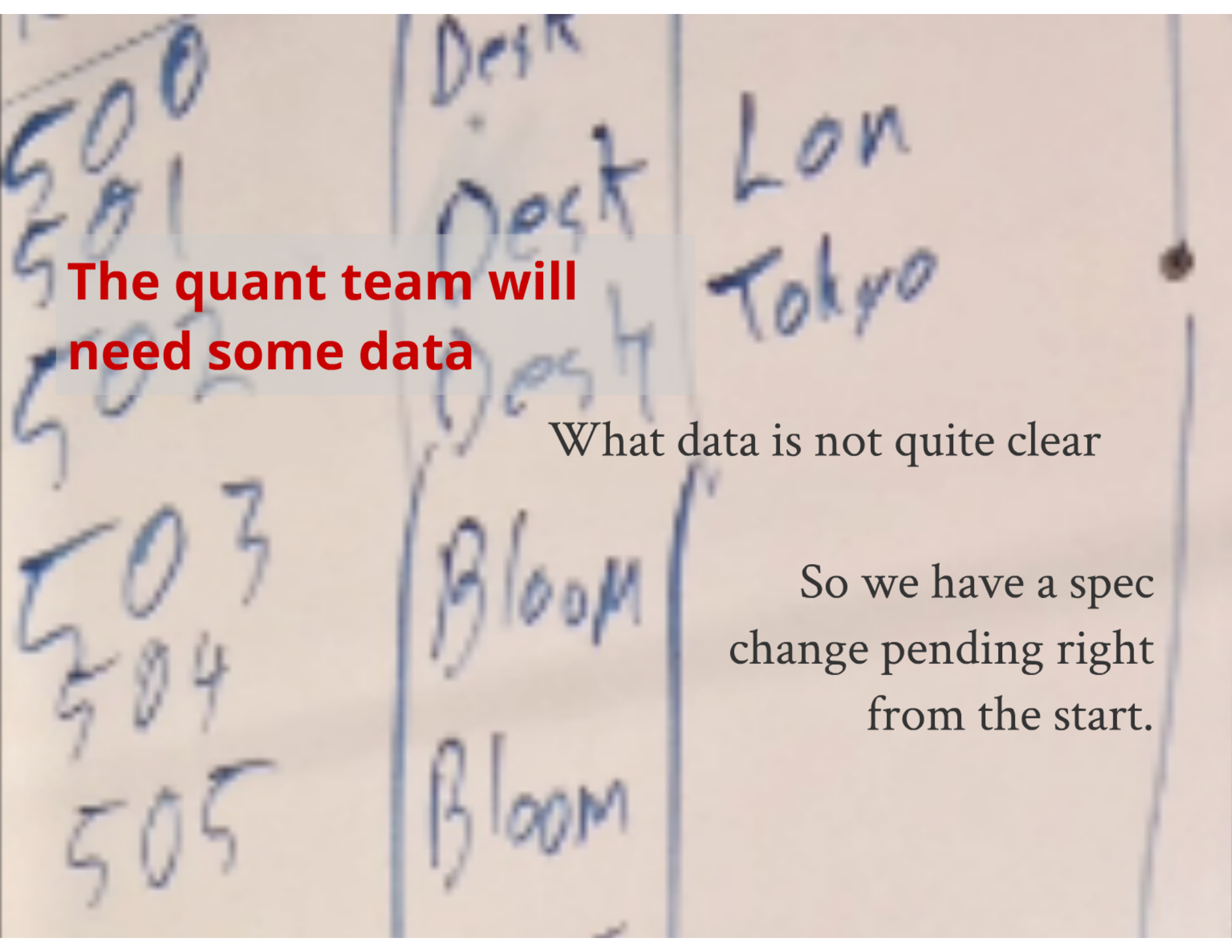
- Extreme flexibility allows for rapid development and customization
- Batteries included - large standard library, even larger scientific python community
- Good language for both quants and application developers

Our v0.1 interface is extended but not superseded.

The built in libraries and others have saved us considerable effort.


This is where it starts
market.News [CurveName] [Cal_date]
market.fx [CurveName].vol [Cal_date]
market.equity [Symbol].price.volume [Cal-date]
market.equityIndex [Symbol].component [caldate]
model.correlations
reference.countryInfo ['US'].GDP['2013']
So here's our interface
{data collection}.{datatype}[{asset id}].{property}[date]
Reference
Bucket
Smooth
This is our entire initial interface, and not even a firm interface.
Call it interface v0.1, but this--and the flexibility of Python--can get us all the way into production.
point
pre-ended
The only thing we really know is the deadline
• The application developers can't wait for the models to be complete before they start on data
• Once the models are complete the quants will want to run right away
• It's time to start on an interface
Permanent
transitory
Where it always starts, on a whiteboard.
Getting started on the Quant side
Exchange Rates Correlation Risk
phase 1
phase 2
Historic
Live da
Shock.

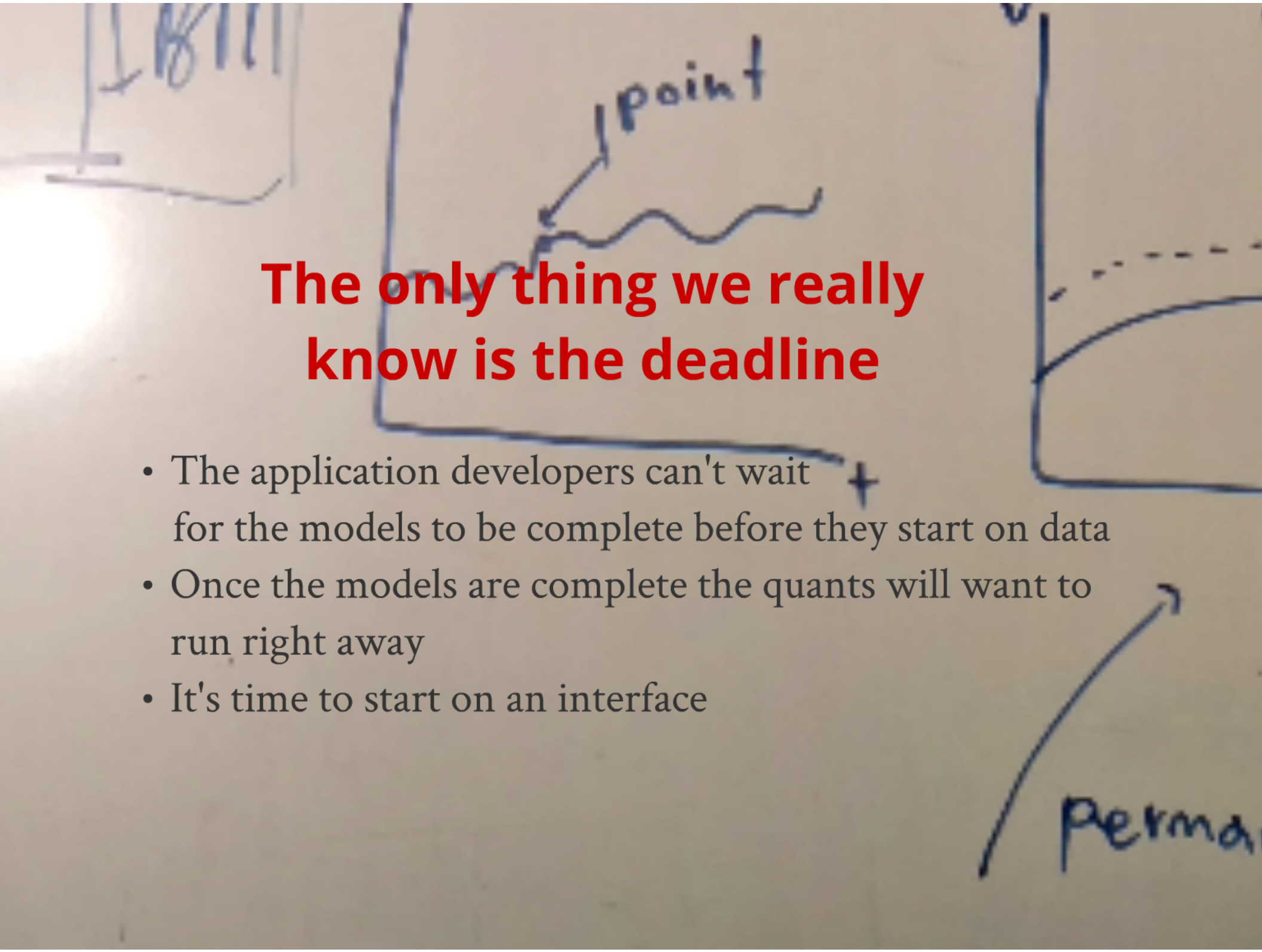

There's a new project!
We're going to build a new Market Risk platform!
Application Developers and Quantitative Researchers start talking.
Everybody knows a thing or two from their previous projects, **but the details of this project are still in flux.**



The quant team will need some data

What data is not quite clear

So we have a spec change pending right from the start.


point
The only thing we really know is the deadline
• The application developers can't wait for the models to be complete before they start on data
• Once the models are complete the quants will want to run right away
• It's time to start on an interface

market.News [CurveName] [Cal_date]

market.fx [CurveName].vol [Cal_date]

market.equity [Symbol] : price / volume [Cal_date]

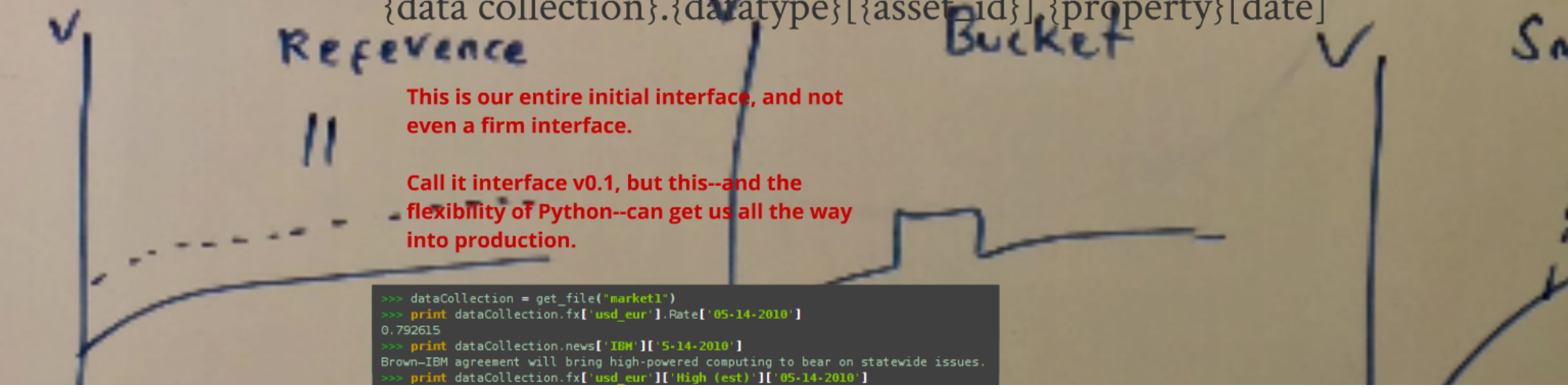
market.equityIndex [Symbol].component [

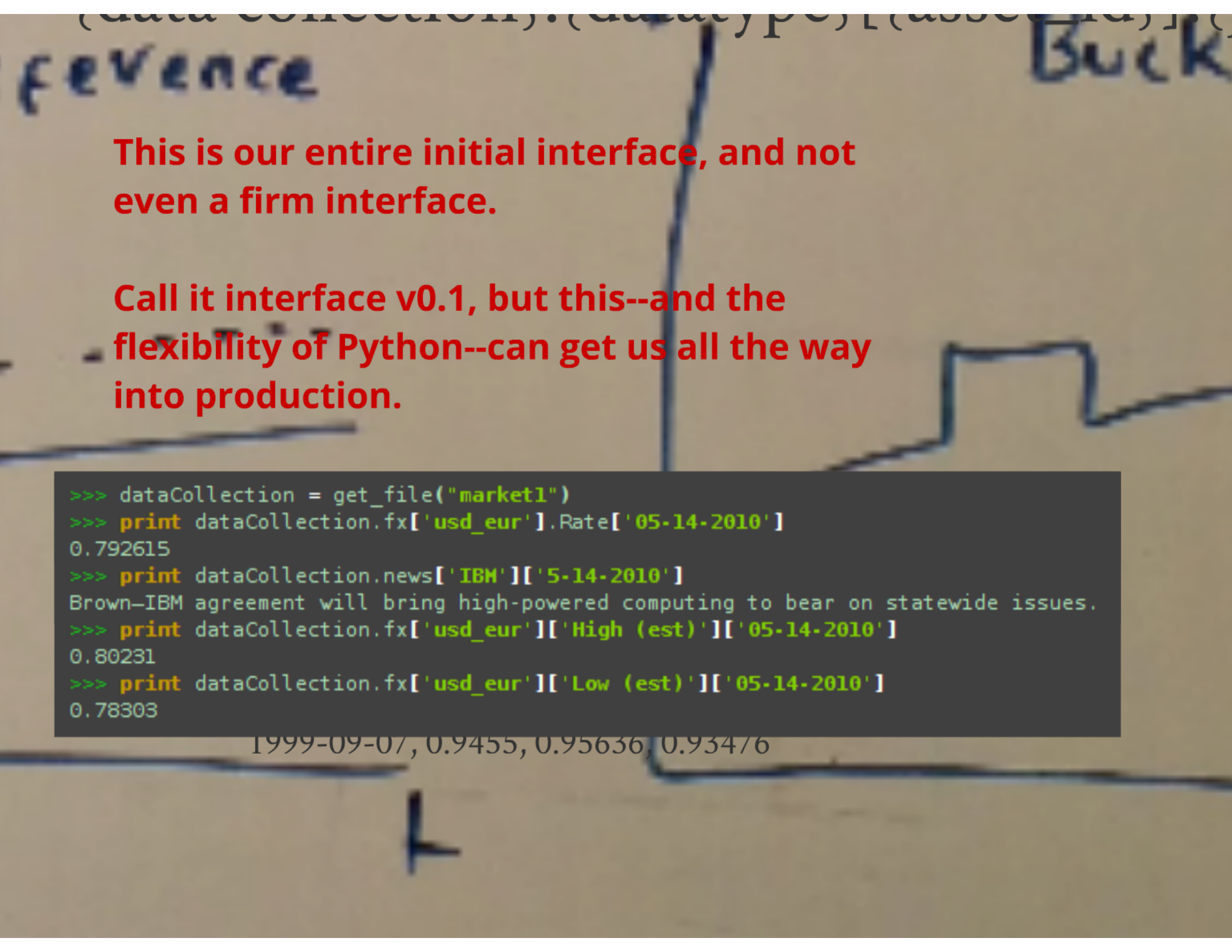
model.correlations

reference.countryInfo ['US'].GDP ['2013']

So here's our interface

{data collection}.{datatype}[{asset_id}].{property}[date]




This is our entire initial interface, and not even a firm interface.
Call it interface v0.1, but this--and the flexibility of Python--can get us all the way into production.
>>> dataCollection = get_file("market1")
>>> print dataCollection.fx['usd_eur'].Rate['05-14-2010']
0.792615
>>> print dataCollection.news['IBM']['5-14-2010']
Brown–IBM agreement will bring high-powered computing to bear on statewide issues.
>>> print dataCollection.fx['usd_eur']['High (est)']['05-14-2010']
0.80231
>>> print dataCollection.fx['usd_eur']['Low (est)']['05-14-2010']
0.78303

Getting started on the Quant side

- Can only do so much research
- Quants need data to play with to figure out what data they need
- Will initially be sourcing their own data anyway

Get started with Dingus

- An extremely flexible mocking library
- This is a truly universal mock, using operator overloading every operation is mocked including:
  - indexing
  - evaluation
  - attribute access
- Mario can insert any data he needs to get started directly into the objects
- This will be a template for the later data

```
In [85]: # Dingus Example
         from data_access import get_dummy
         print(repr(get_dummy("test")))
         dummyCollection = get_dummy("market1")
         print(repr(dummyCollection))
         print(repr(dummyCollection.news))
         print(repr(dummyCollection.news['IBM']))
         print(repr(dummyCollection.news['IBM']['5-14-2010']))
         print(repr(dummyCollection.fx.usd_eur['High (est)']['2010-05-14']))
         print(repr(dummyCollection.fx.usd_eur['Low (est)']['2010-05-14']))
         print(repr(dummyCollection.fx.usd_eur.Rate['2010-05-14']))

         <Dingus test_env>
         <Dingus market1_env>
         <Dingus market1_env.news>
         <Dingus market1_env.news[IBM]>
         <Dingus market1_env.news[IBM][5-14-2010]>
         <Dingus market1_env.fx.usd_eur['High (est)']['2010-05-14']>
         <Dingus market1_env.fx.usd_eur['Low (est)']['2010-05-14']>
         <Dingus market1_env.fx.usd_eur.Rate['2010-05-14']>
```

Get started with Dingus

- An extremely flexible mocking library
- This is a truly universal mock, using operator overloading every operation is mocked including:
  - indexing
  - evaluation
  - attribute access
- Mario can insert any data he needs to get started directly into the objects
- This will be a template for the later data to follow

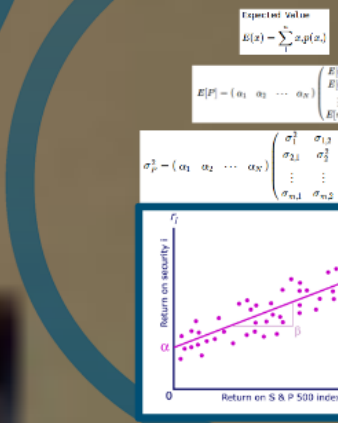
```
In [85]: # Dingus Example
         from data_access import get_dummy
         print(repr(get_dummy("test")))
         dummyCollection = get_dummy("market1")
         print(repr(dummyCollection))
         print(repr(dummyCollection.news))
         print(repr(dummyCollection.news['IBM']))
         print(repr(dummyCollection.news['IBM']['5-14-2010']))
         print(repr(dummyCollection.fx.usd_eur['High (est)']['2010-05-14']))
         print(repr(dummyCollection.fx.usd_eur['Low (est)']['2010-05-14']))
         print(repr(dummyCollection.fx.usd_eur.Rate['2010-05-14']))

         <Dingus test_env>
         <Dingus market1_env>
         <Dingus market1_env.news>
         <Dingus market1_env.news[IBM]>
         <Dingus market1_env.news[IBM][5-14-2010]>
         <Dingus market1_env.fx.usd_eur[High (est)][2010-05-14]>
         <Dingus market1_env.fx.usd_eur[Low (est)][2010-05-14]>
         <Dingus market1_env.fx.usd_eur.Rate[2010-05-14]>
```

So, what have we learned about the project?

Exchange Rates Correlation Risk

We want to measure what is the effect of a news shock in a exchange rate indexed against US Dollar (i.e., what is the impact on all the other exchange rates after a shock to the Japanese economy USD/ JPY)

Model



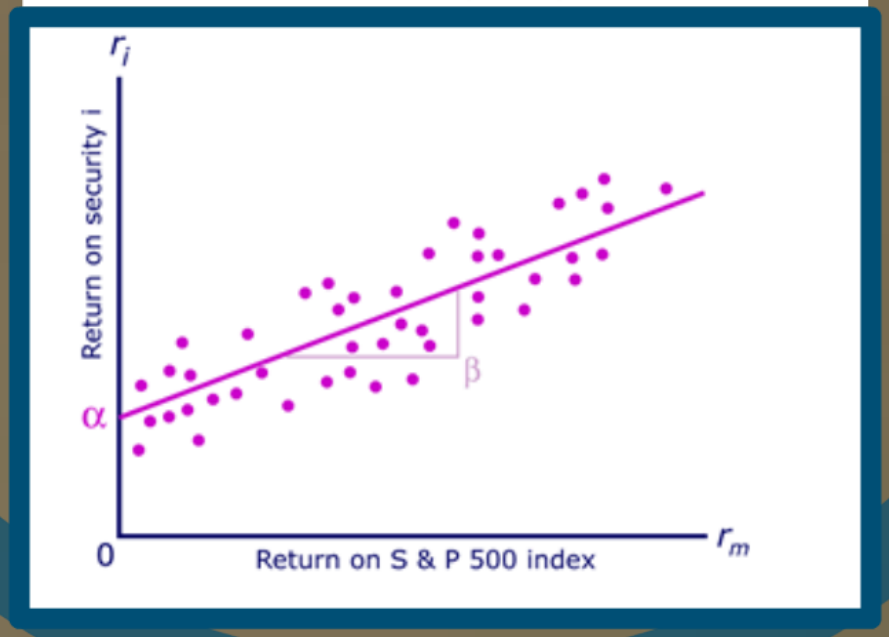
Model

Expected Value

$$E(x) = \sum_{1}^{n} x_i p(x_i)$$

$$E[P] = \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_N \end{pmatrix} \begin{pmatrix} E[r_1] \\ E[r_2] \\ \vdots \\ E[r_N] \end{pmatrix}$$

$$\sigma_P^2 = \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_N \end{pmatrix} \begin{pmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,n} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{m,1} & \sigma_{m,2} & \cdots & \sigma_N^2 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{pmatrix}$$

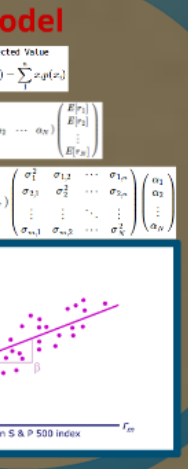


Change #1: Time for data.

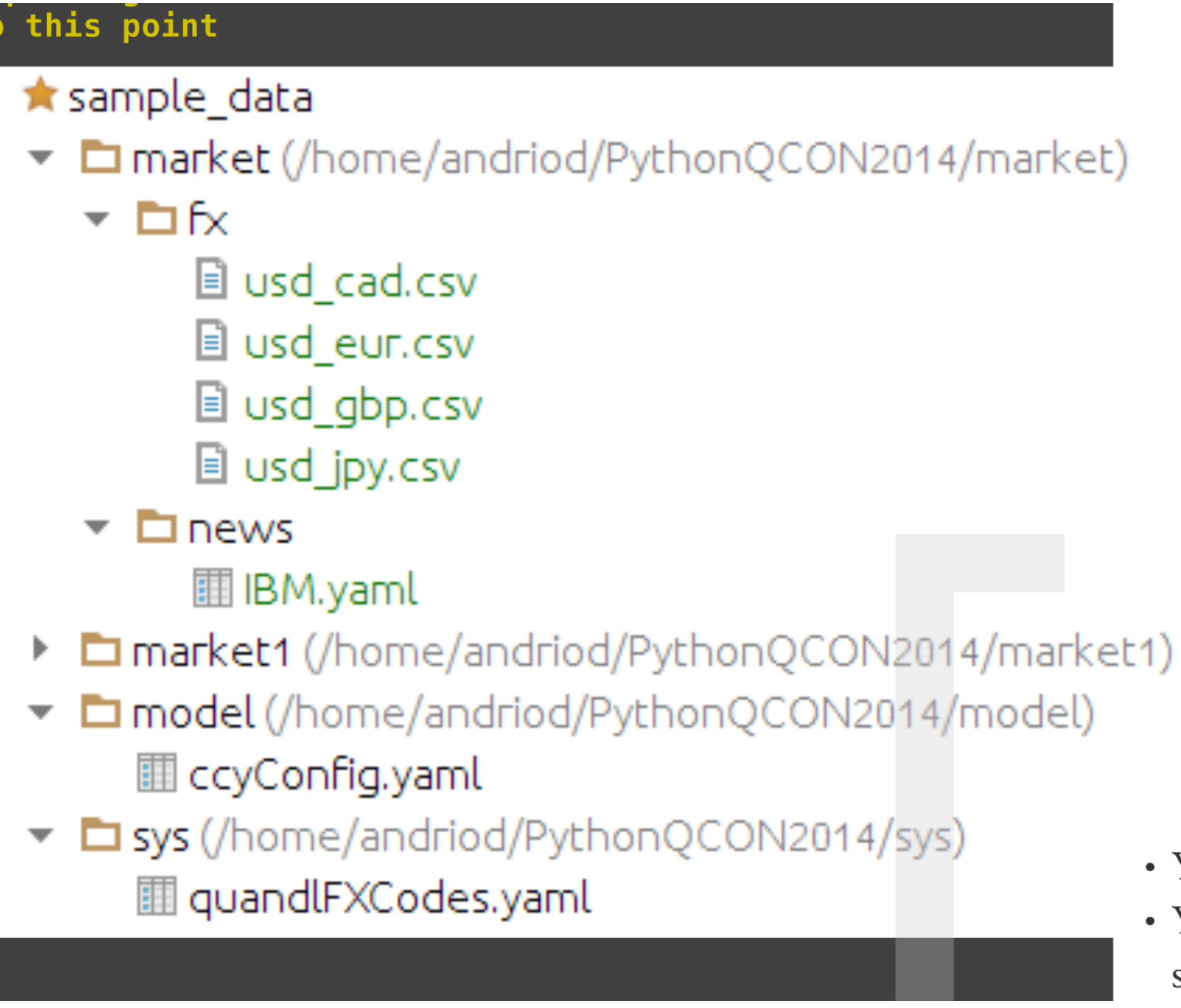
Time for real data

Exchange rates available in public repositories or in commercial data stores

We'll just grab some and store them in a directory for now



this point



Se

- You'll not
- YAML is
  serializati

2014/market1)
14/model)

sys)

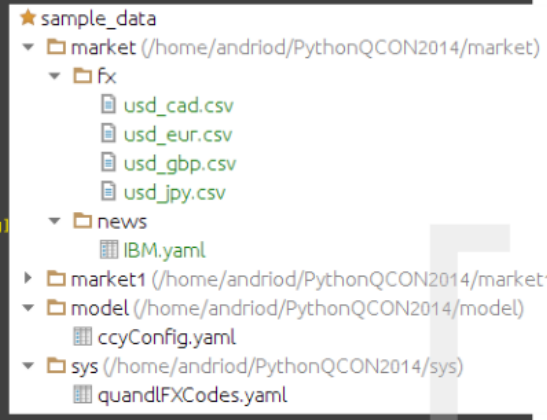
Serialization & YAML

- You'll notice that there are some YAML files here
- YAML is not Python specific, just a nice text serialization format capable of serializing graphs
- However, built into the format, we can serialize any Python instance, though this may limit deserialization options

A simple layer to access our data in files

simple_files.py

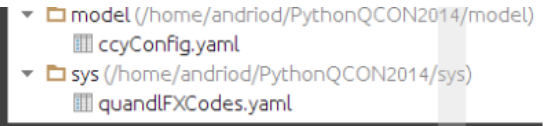
```
from collections import MutableMapping
import os

import pandas
import yaml


__author__ = 'andriod'


class FileHolder(MutableMapping, object):
    next_type = None

    def __init__(self, name, *prev_path):
        """FileHolders hold a section of the data and are matched in the file system with a matching directory or file

        :type name: str - name of the file or directory corresponding to this instance
        :param prev_path: - tuple of the FileHolders in the to to this point
        """
        self.name = name
        self.path = prev_path + (self,)
        self.is_dir = os.path.isdir(self.file_path)
        self._yaml_obj = None

        self._cache = {}
        if self.next_type is None:
            self.next_type = type(self)

    def __getitem__(self, item):
        """Overriding the [indexing] operation

        :type item: str - the key being accessed by [indexing]
        :return:
        """
        if item not in self._cache:
            ret = self.create_sub_obj(item)
            self._cache[item] = ret
        return self._cache[item]

    def __getattr__(self, item):
        """Overriding attribute access

        :type item: str - attribute requested
        :return: :raise AttributeError:
        """
        if item not in ['yaml_dict', 'file_path'] and not item[0] == "_":
            return self[item]
        else:
            raise AttributeError

    def create_sub_obj(self, item):
        """In both cases of .attribute and [indexing] we actually just continue to walk the tree

        :type item: str - the name of the next object
        :return: a newly created object, caller is responsible for caching
        """
        if self.yaml_obj:
            return self.yaml_obj[item]
        elif os.path.isfile(os.path.join(self.file_path, item) + ".csv"):
```



Serialization & YAML

- You'll notice that there are some YAML files here
- YAML is not Python specific, just a nice text serialization format capable of serializing graphs
- However, built into the format, we can serialize any Python instance, though this may limit deserialization options

```
            self._cache[item] = ret
        return self._cache[item]

    def __getattr__(self, item):
        """Overriding attribute access

        :type item: str - attribute requested
        :return: :raise AttributeError:
        """
        if item not in ['yaml_dict', 'file_path'] and not item[0] == "_":
            return self[item]
        else:
            raise AttributeError

    def create_sub_obj(self, item):
        """In both cases of .attribute and [indexing] we actually just continue to walk the tree

        :type item: str - the name of the next object
        :return: a newly created object, caller is responsible for caching
        """
        if self.yaml_obj:
            return self.yaml_obj[item]
        elif os.path.isfile(os.path.join(self.file_path, item) + ".csv"):
            return pandas.DataFrame.from_csv(os.path.join(self.file_path, item) + ".csv")
        return self.next_type(item, *self.path)

    @property
    def yaml_obj(self):
        if self._yaml_obj is not None:
            return self._yaml_obj
        elif os.path.isfile(self.file_path + ".yaml"):
            self._yaml_obj = yaml.load(open(self.file_path + ".yaml"))
            return self._yaml_obj
        else:
            return None

    @property
    def file_path(self):
        return str(os.path.join(*[x.name for x in self.path]))

    @property
    def value(self):
        if self.is_dir:
            return "Directory, no value"
        elif os.path.isfile(self.file_path):
            return open(self.file_path).read()
        elif os.path.isfile(self.file_path + ".yaml"):
            return yaml.load(open(self.file_path + ".yaml"))

    def __repr__(self, *args, **kwargs):
        return "< {path} - {value} ".format(path=",".join(x.name for x in self.path), value=self.value)

    def __str__(self, *args, **kwargs):
        return str(self.value)

    def __iter__(self):
        if self.is_dir:
            return (self[os.path.splitext(os.path.basename(path))[0]] for path in os.listdir(self.file_path))
        elif self.yaml_obj is not None:
            return iter(self.yaml_obj)
        else:
            return iter([]) #empty iter, we have no case for this now


    def __len__(self):
        if self.is_dir:
            return len(os.listdir(self.file_path))
        elif self.yaml_obj is not None:
            return len(self.yaml_obj)
        else:
            return 0  #empty iter, we have no case for this now

    def __delitem__(self, key):
        del self._cache[key]

    def __setitem__(self, key, value):
        self._cache[key] = value
```



- You'll notice that there are some YAML files here
- YAML is not Python specific, just a nice text serialization format capable of serializing graphs
- However, built into the format, we can serialize any Python instance, though this may limit deserialization options

Exploratory Data Analysis

Required Libraries

-Numpy
-SciPy
-Pandas
-SciKit-Learn
-StatsModels
-Patsy
-Matplotlib
-Ipython
-Quandl
-Dingus

IP[y]: Notebook Qcon2014VaR

```
In [92]: from data_access import get_file
         import pandas as pd
         import numpy as np

         market = get_file('market1')
         model = get_file('model')

         def make_ccy_matrix(market, model):
             return pd.DataFrame({ccy:curve.Rate for ccy,curve in [(ccy,market.fx[ccy+'_usd']) for ccy in model.ccyConfig]})

         ccy_matrix=make_ccy_matrix(market, model)
```

```
In [93]: ccy_matrix.tail()
```

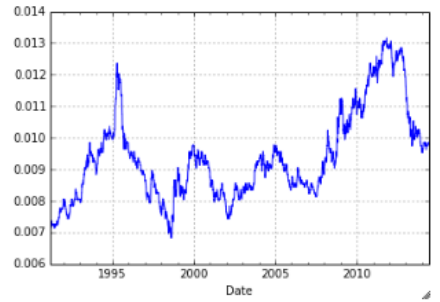
Out[93]:

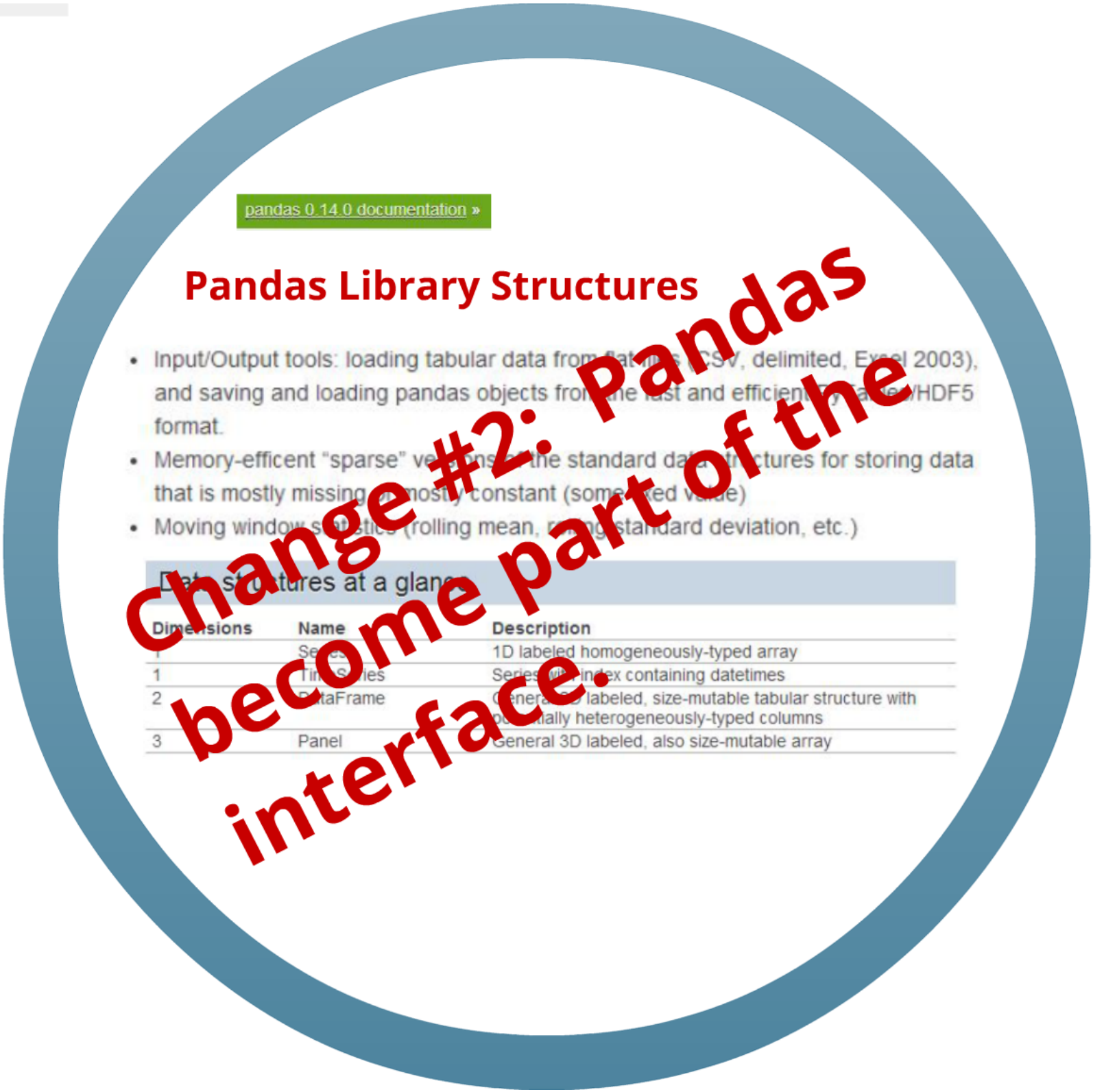
	aed	afn	all	ang	aoa	ars	aud	awg	bbd	bdt	bgn	bhd	bif
Date													
2014-05-20	0.272375	0.017439	0.009838	0.556593	0.010238	0.124287	0.935728	0.560754	0.500429	0.012864	0.700729	2.65412	0.000636
2014-05-21	0.272375	0.017439	0.009838	0.556593	0.010238	0.124038	0.930881	0.560754	0.500246	0.012864	0.700741	2.65292	0.000636
2014-05-22	0.272375	0.017439	0.009838	0.556593	0.010238	0.124063	0.925006	0.560754	0.499910	0.012864	0.700100	2.65292	0.000636
2014-05-23	0.272375	0.017439	0.009838	0.556593	0.010238	0.124063	0.923839	0.559834	0.500046	0.012864	0.699248	2.65292	0.000636
2014-05-26	0.272375	0.017439	0.009736	0.556593	0.010238	0.124063	0.923627	0.560754	0.500327	0.012864	0.697697	2.65208	0.000636

5 rows × 148 columns

```
In [34]: ccy_matrix['jpy'].plot()
```

Out[34]: <matplotlib.axes.AxesSubplot at 0xb5312ac>




pandas 0.14.0 documentation »
Pandas Library Structures
Input/Output tools: loading tabular data from flat files (CSV, delimited, Excel 2003), and saving and loading pandas objects from the fast and efficient PyTables/HDF5 format.
Memory-efficent "sparse" versions of the standard data structures for storing data that is mostly missing or mostly constant (some fixed value)
Moving window statistics (rolling mean, rolling standard deviation, etc.)
Dimensions
Name
Description
1
Series
1D labeled homogeneously-typed array
1
TimeSeries
Series with index containing datetimes
2
DataFrame
General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed columns
3
Panel
General 3D labeled, also size-mutable array
Change #2: Pandas become part of the interface.

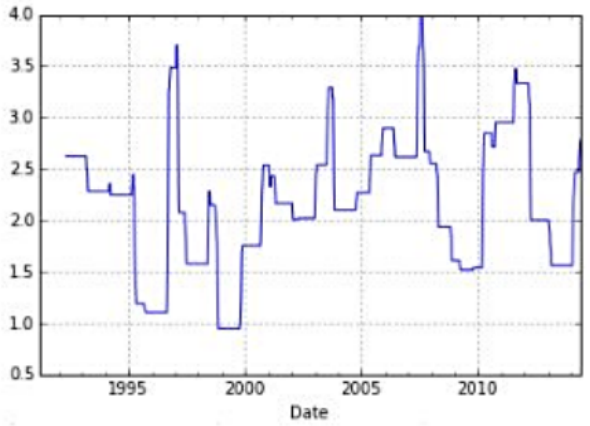
Predictions by model

```
In [25]: corr=ccy_matrix.corr()

In [26]: corr
```

Out[26]:

	aed	afn	all	ang	aoa	ars	aud	awg	bbd	bdt	bgn	bhd	bif
aed	1.000000	-0.387602	-0.215562	0.318342	0.150919	0.345806	-0.429843	0.204882	-0.036890	0.421365	-0.294120	0.061025	0.296765
afn	-0.387602	1.000000	0.850153	-0.658482	-0.386002	-0.880705	0.730158	-0.116045	0.191257	-0.870072	0.844934	0.124149	-0.748667
all	-0.215562	0.850153	1.000000	-0.493636	-0.391092	-0.726466	0.660349	-0.001542	0.059841	-0.695829	0.753363	0.164095	-0.607221
ang	0.318342	-0.658482	-0.493636	1.000000	-0.064658	0.713985	-0.389470	-0.009043	-0.568945	0.827816	-0.782344	-0.048932	0.880685
aoa	0.150919	-0.386002	-0.391092	-0.064658	1.000000	0.391747	-0.487989	0.197351	0.212811	0.183662	-0.108598	-0.041409	-0.093057
ars	0.345806	-0.880705	-0.726466	0.713985	0.391747	1.000000	-0.605686	0.082945	-0.289172	0.885343	-0.882117	-0.092293	0.829553
aud	-0.429843	0.730158	0.660349	-0.389470	-0.487989	-0.605686	1.000000	-0.289410	-0.033362	-0.660589	0.519028	0.094448	-0.321689
awg	0.204882	-0.116045	-0.001542	-0.009043	0.197351	0.082945	-0.289410	1.000000	0.124027	0.098901	-0.022411	0.009140	-0.041812
bbd	-0.036890	0.191257	0.059841	-0.568945	0.212811	-0.289172	-0.033362	0.124027	1.000000	-0.340147	0.308121	0.074803	-0.499189
bdt	0.421365	-0.870072	-0.695829	0.827816	0.183662	0.885343	-0.660589	0.098901	-0.340147	1.000000	-0.908591	-0.109724	0.911894
bgn	-0.294120	0.844934	0.753363	-0.782344	-0.108598	-0.882117	0.519028	-0.022411	0.308121	-0.908591	1.000000	0.108716	-0.921780
bhd	0.061025	0.124149	0.164095	-0.048932	-0.041409	-0.092293	0.094448	0.009140	0.074803	-0.109724	0.108716	1.000000	-0.085641
bif	0.296765	-0.748667	-0.607221	0.880685	-0.093057	0.829553	-0.321689	-0.041812	-0.499189	0.911894	-0.921780	-0.085641	1.000000

Rolling R^2




```
In [36]: from statsmodels.formula.api import ols
         lm1 = ols('jpy~aed', ccy_matrix).fit()
         lm1.summary()
```

Out[36]:

OLS Regression Results

Dep. Variable:	jpy	**R-squared:**	0.263
Model:	OLS	**Adj. R-squared:**	0.263
Method:	Least Squares	**F-statistic:**	1825.
Date:	Wed, 28 May 2014	**Prob (F-statistic):**	0.00
Time:	04:33:35	**Log-Likelihood:**	26976.
No. Observations:	5115	**AIC:**	-5.395e+04
Df Residuals:	5113	**BIC:**	-5.393e+04
Df Model:	1		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	4.3032	0.101	42.816	0.000	4.106 4.500
aed	-15.7668	0.369	-42.721	0.000	-16.490 -15.043

Omnibus:	2582.831	**Durbin-Watson:**	0.082
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	69611.799
Skew:	1.855	**Prob(JB):**	0.00
Kurtosis:	20.688	**Cond. No.**	2.29e+04

Ok, so how about live data?

Change #3: Switching the input data source.

```
class QuandlAsset(object):
    def __init__(self, quandl_name, authtoken=None):
        """Uses the Quandl service to get live data

        :type quandl_name: str - an asset name assigned by Quandl
        :type authtoken: builtins.NoneType - Quandl auth token if available
        """
        self._authtoken = authtoken
        self.quandl_name = quandl_name
        self._value = None

    def __getattr__(self, item):
        """Redirect any unknown attribute access to the data retrieved from Quandl, this constitutes a proxy
        """
        return getattr(self.value, item)

    def __len__(self):
        return len(self.value)

    @property
    def value(self):
        if self._value is not None:
            return self._value
        self._value = Quandl.get(self.quandl_name, authtoken=self._authtoken)
        return self._value


def get_live(coll_name, sys_coll=None):
    if coll_name == 'market2':
        coll = FileHolder('market1')
        coll.etf.usd_eur = QuandlAsset("GOOG/NYSE_ERO")
        coll.fx.usd_jpy = QuandlAsset("QUANDL/USDJPY")
    elif coll_name == 'market3' and sys_coll is not None:
        coll = FileHolder('market1')
        for name, code in sys_coll.quandlFXCodes.items():
            asset = QuandlAsset(code)
            coll.fx[name] = asset

    return coll
```

Change #4: Changing the input data before passing it to the model.

Stress/Shock

The we need to build a stress framework

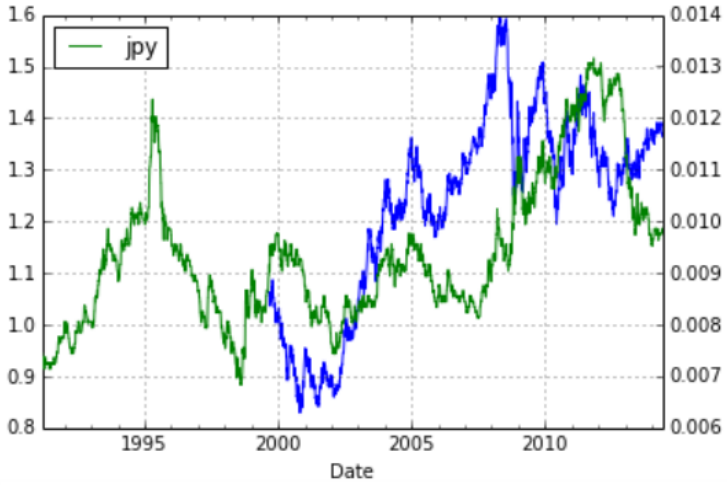
Just as a test, lets consider an alien landing

On the plus side they're friendly

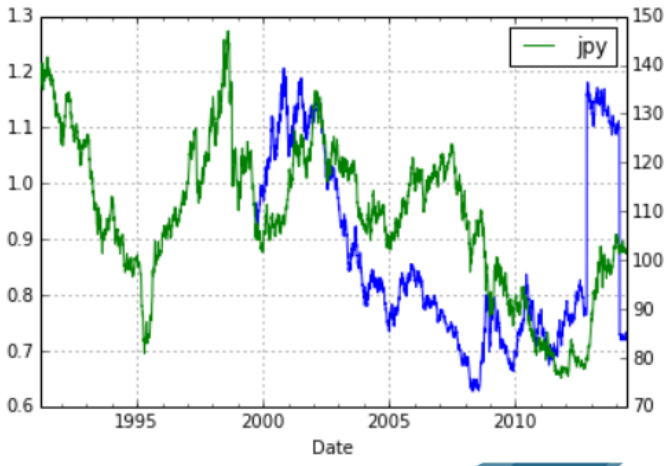
Tweaking data

```
def point_tweak(collection, curve, date, operation, amount):
    path = [x for x in re.split("[\.[\]]", curve) if x]
    collection = deepcopy(collection)
    curr = collection
    for element in path:
        curr = curr[element]
    curr[date] = operation(curr[date], amount)
    return collection


def step_tweak(collection, curve, start_date, end_date, operation, amount):
    path = [x for x in re.split("[\.[\]]", curve) if x]
    collection = deepcopy(collection)
    curr = collection
    for element in path:
        curr = curr[element]
    curr[start_date:end_date] = curr[start_date:end_date].apply(operation, args=(amount,))
    return collection
```

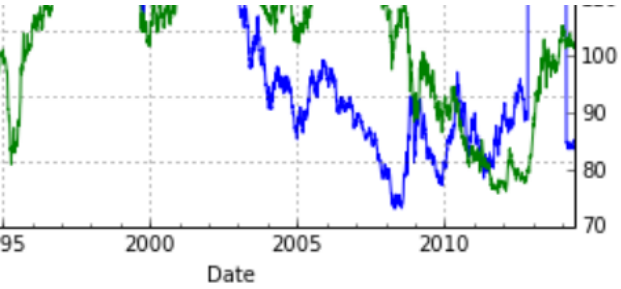
Using stress functions




```
from stress import step_tweak
from operator import add, mul
steptweakedCollection = step_tweak(market, 'fx.usd_eur.Rate', '2012-10-31', '2014-02-15', mul, 1.5)
```

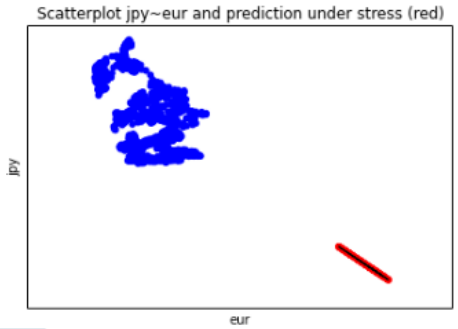


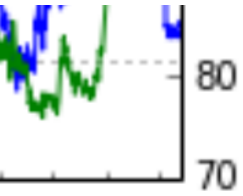
Prediction under stre



Predictions under stress

```
X=np.asarray(steptweakedCollection.fx.usd_eur.Rate['2007-10-31':'2012-10-30'])[:, np.newaxis]
Y=np.asarray(steptweakedCollection.fx.usd_jpy.Rate['2007-10-31':'2012-10-30'])

lr=LinearRegression()
lr.fit(X,Y)

# Testing Data
test_X=np.asarray(steptweakedCollection.fx.usd_eur.Rate['2012-10-31':'2014-02-15'])[:, np.newaxis]
test_Y=np.asarray(steptweakedCollection.fx.usd_jpy.Rate['2012-10-31':'2014-02-15'])

# Plot Prediction
pred = lr.predict(test_X)

plt.scatter(X, Y, color='blue')
plt.scatter(test_X, pred, color='red')
plt.plot(test_X, lr.predict(test_X), color='black',linewidth=1)
plt.xlabel('eur')
plt.ylabel('jpy')
plt.title('Scatterplot jpy~eur and prediction under stress (red)')
plt.grid(True)
plt.xticks(())
plt.yticks(())
plt.show()
```

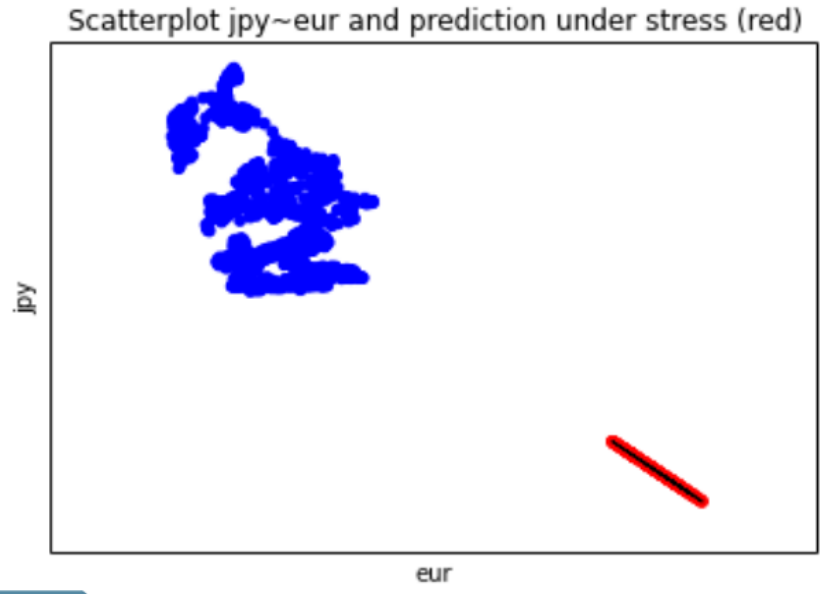




Predictions under stress

```
X=np.asarray(steptweakedCollection.fx.usd_eur.Rate['2007-10-31':'2012-10-30'])[:, np.newaxis]
Y=np.asarray(steptweakedCollection.fx.usd_jpy.Rate['2007-10-31':'2012-10-30'])

lr=LinearRegression()
lr.fit(X,Y)

# Testing Data
test_X=np.asarray(steptweakedCollection.fx.usd_eur.Rate['2012-10-31':'2014-02-15'])[:, np.newaxis]
test_Y=np.asarray(steptweakedCollection.fx.usd_jpy.Rate['2012-10-31':'2014-02-15'])

# Plot Prediction
pred = lr.predict(test_X)

plt.scatter(X, Y, color='blue')
plt.scatter(test_X, pred, color='red')
plt.plot(test_X, lr.predict(test_X), color='black',linewidth=1)
plt.xlabel('eur')
plt.ylabel('jpy')
plt.title('Scatterplot jpy~eur and prediction under stress (red)')
plt.grid(True)
plt.xticks(())
plt.yticks(())
plt.show()
```

```
lr.fit(X,Y)

# Testing Data
test_X=np.asarray(steptweakedCollection.fx.usd_eur.Rate['2012-10-31':'2014-02-15'])[:, np.newaxis]
test_Y=np.asarray(steptweakedCollection.fx.usd_jpy.Rate['2012-10-31':'2014-02-15'])

# Plot Prediction
pred = lr.predict(test_X)

plt.scatter(X, Y, color='blue')
plt.scatter(test_X, pred, color='red')
plt.plot(test_X, lr.predict(test_X), color='black',linewidth=1)
plt.xlabel('eur')
plt.ylabel('jpy')
plt.title('Scatterplot jpy~eur and prediction under stress (red)')
plt.grid(True)
plt.xticks(())
plt.yticks(())
plt.show()
```



Future Directions

- Extending the stress analysis outside to understand different market conditions
- Converting performance sensitive code to C
- Adding tests and limiting flexibility to what we actually need going forward
- Match with the full regulatory requirements
- Backtesting/Model review
- Simulation based modeling
- Apply model to pricing portfolio

uild new teams around it?

- Extreme flexibility allows for rapid development and customization
- Batteries included - large standard library, even larger scientific python community
- Good language for both quants and application developers

Our v0.1 interface is extended but not superseded.

The built in libraries and others have saved us considerable effort.

The language has supported statistical modeling and development of enterprise features.

Links

- **Code:** https://github.com/riskfocus/PythonQCON2014
- **Prezi:** http://prezi.com/fvdmyfkudohp/?utm_campaign=share&utm_medium=copy&rc=ex0share

- Andy Fundinger: Andy.Fundinger@riskfocus.com
- Mario Morales: emetricz@gmail.com