O'REILLY®
Strata
CONFERENCE
+
HADOOP
WORLD

Oct. 23–25, 2012
NEW YORK, NY

Co-presented by
O'REILLY® cloudera

Best Practices for Reproducible Research: Vignettes in Quant Finance

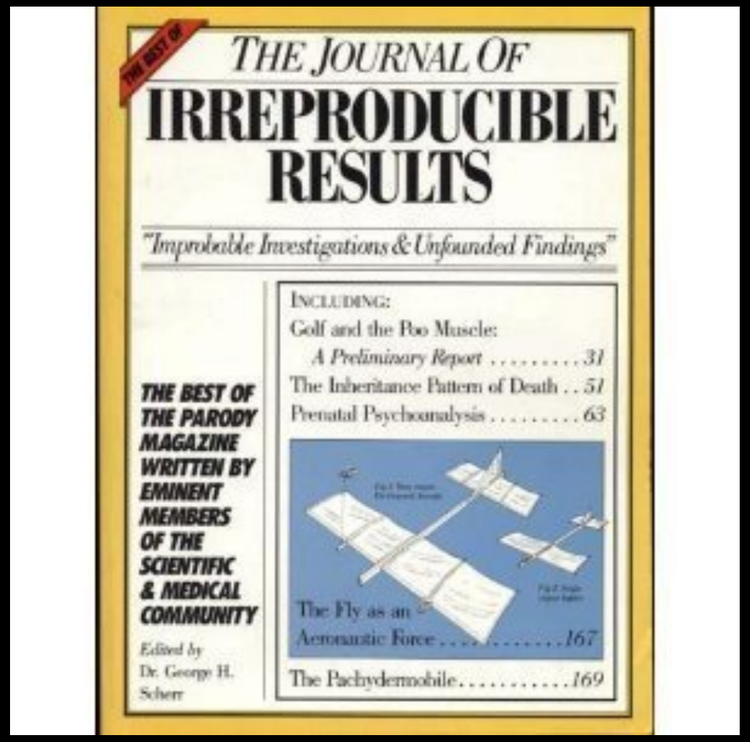
Chang She
Lambda Foundry

About me

- MIT ’05 in CS and Poli Sci
- Quant equities/FX, 2006-2012
- Now Lambda Foundry
- Financial analytics / data tools
- Core dev team for *pandas*

Data *Science*

- applying the scientific method to data
- Reproducibility is a cornerstone of modern science
- Big data => reproducibility is now an organizational effort



Outline

- What is quantitative finance
- Reproducible research
- Conclusions

Outline

- What is quantitative finance
- Reproducible research → Organize
  Version
  Test
- Conclusions

Outline

- What is quantitative finance
- Reproducible research → Organize
  - Version → Code
  - Test
  - Configurations
  - Data
- Conclusions

Relevant questions

- Are my results correct?
- Can anyone produce the same results?
- How do results differ across data environments?
- How quickly can I attribute breakages?

Quantitative finance (marketing literature)

1: Get Data

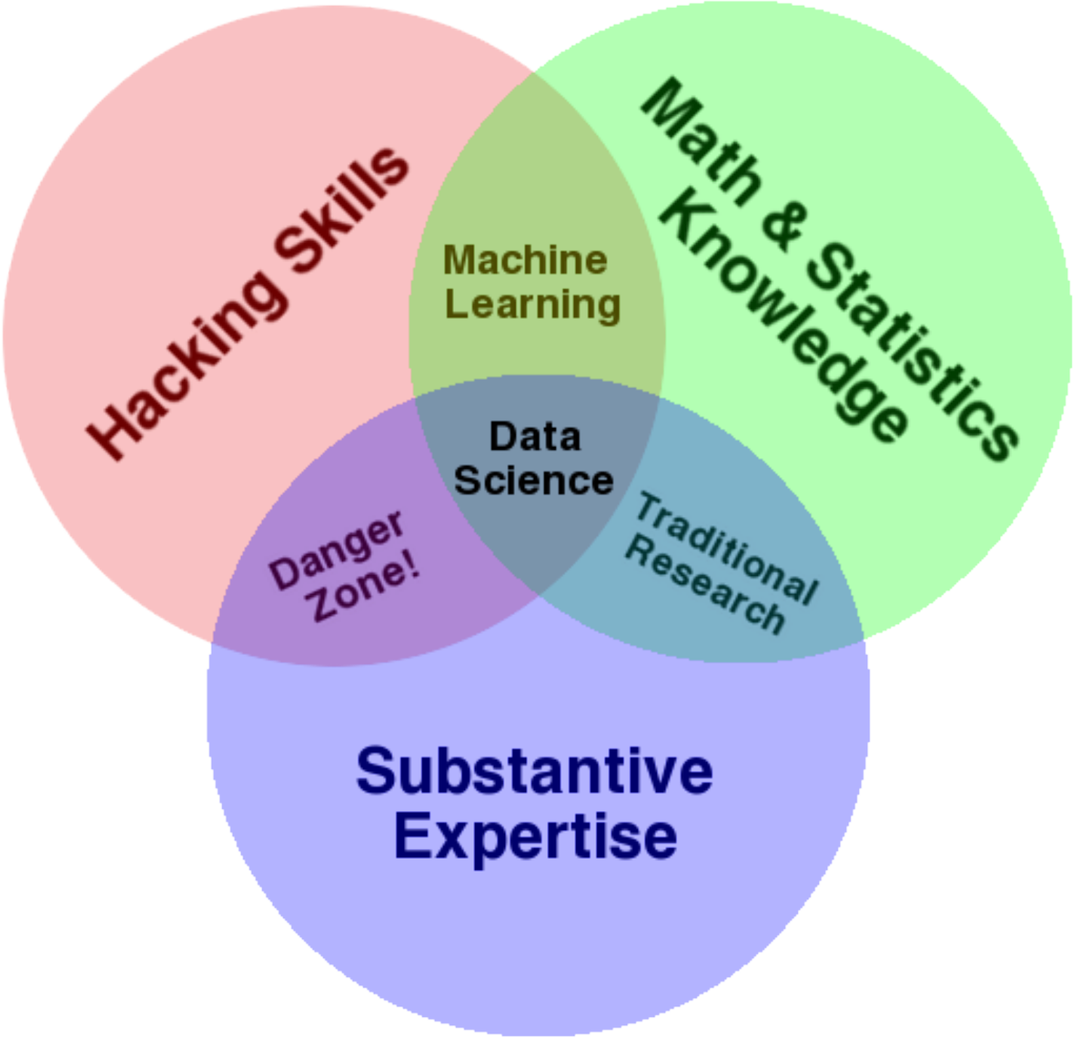
2: Magic Sauce

3: Profit!

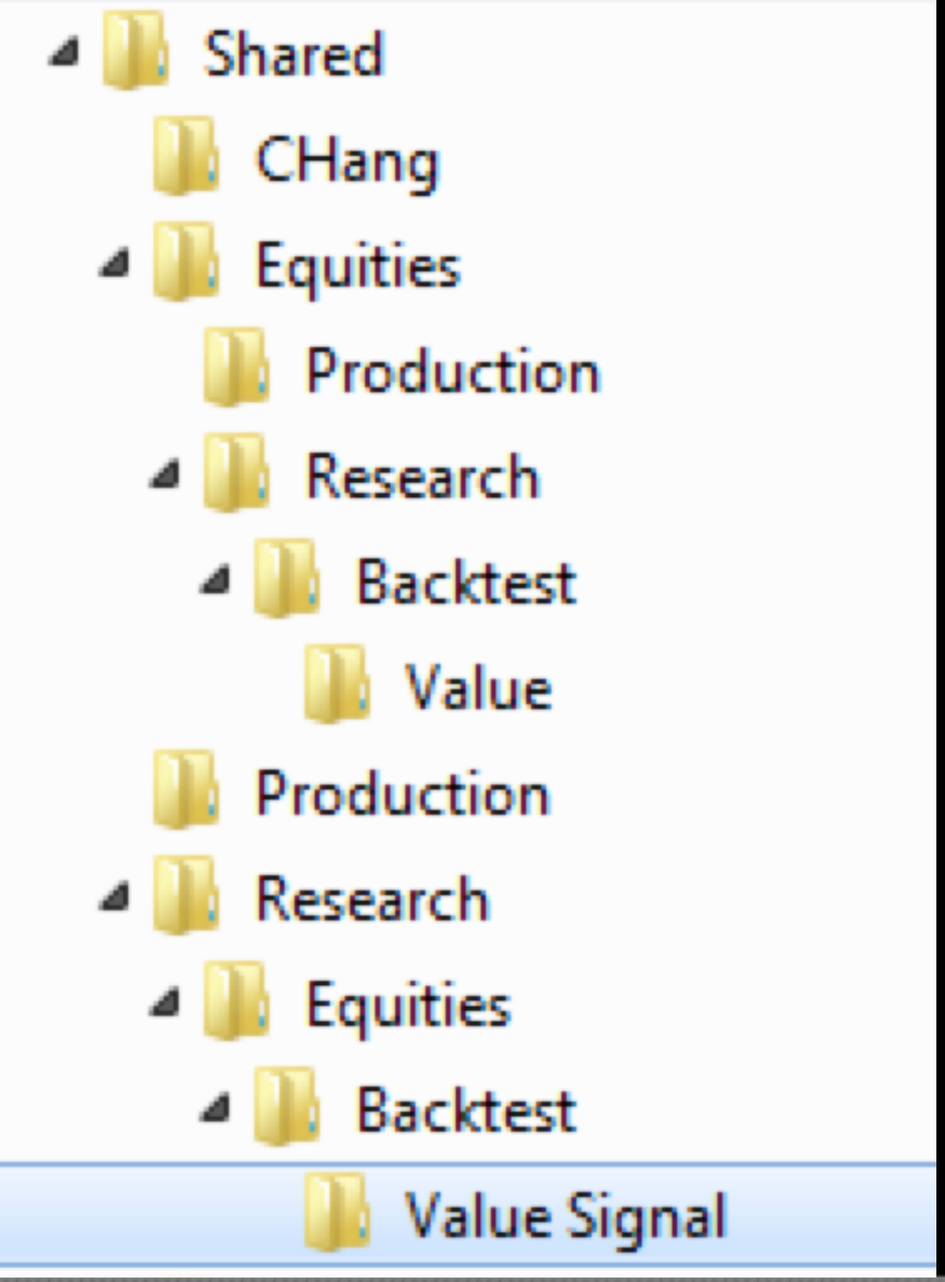
Quantitative finance (according to occupy)

+

→



Courtesy of: Drew Conway



How NOT to organize

- Which folder has the backtest results for the value signal?
- Friends don't let friends organize files like this

Why use the shared network drive?

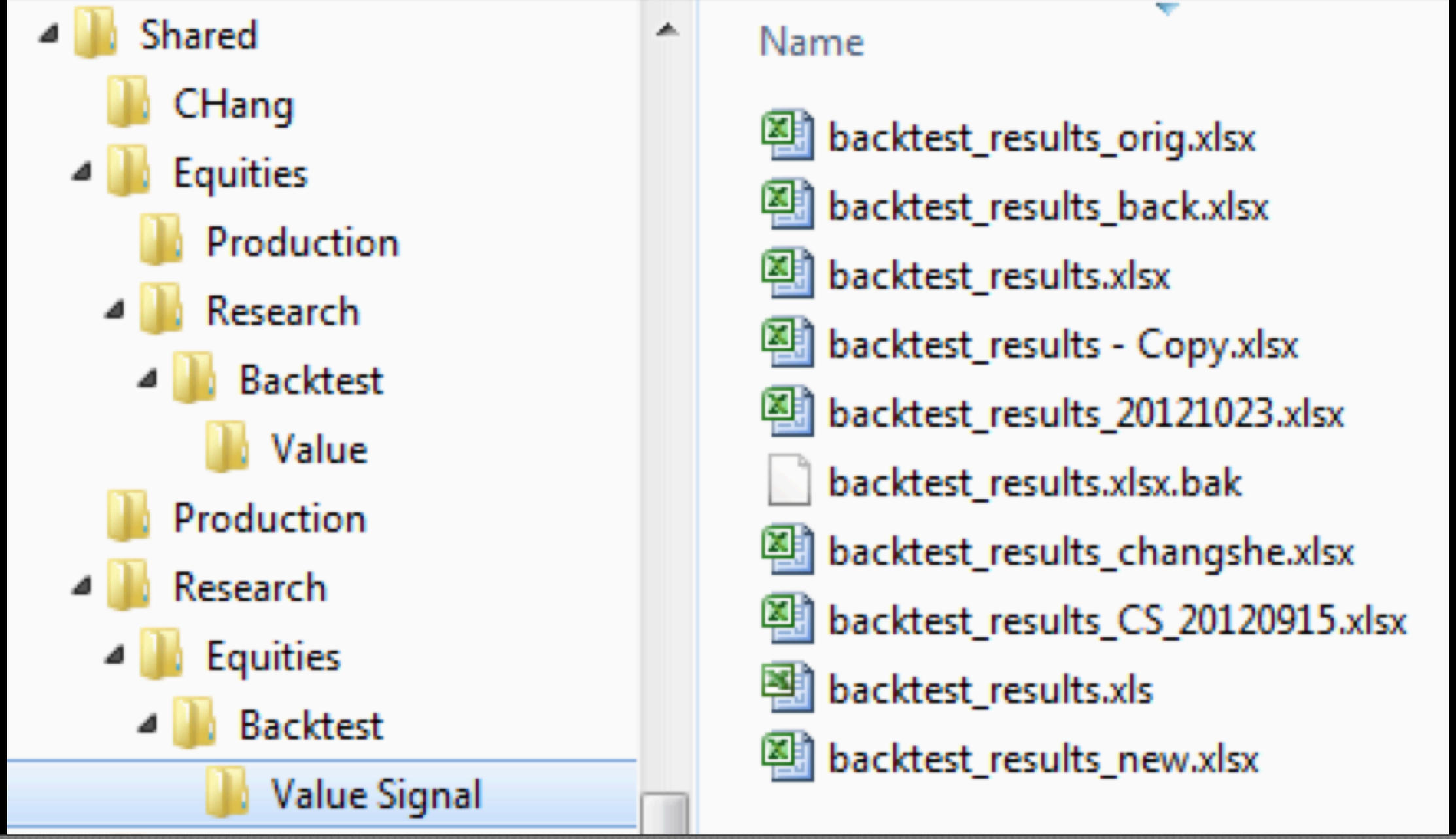
- Backup
- Collaboration
- Presentation
- Separate personal and “master” copies

Solutions

- Take away the demand for primitive drive sharing with the right tools
- Keep the script together with the results -- need good IDE or shell that has easy history saving (e.g., IPython shell or notebook)
- Save version numbers for key libraries

What are “the right tools”?

- Automatic and non-blocking sync/backup
- Easy sharing - give public read access and collaborator read/write access
- Versioning to help reduce multiple modified copies


Shared
CHang
Equities
Production
Research
Backtest
Value
Production
Research
Equities
Backtest
Value Signal
Name
backtest_results_orig.xlsx
backtest_results_back.xlsx
backtest_results.xlsx
backtest_results - Copy.xlsx
backtest_results_20121023.xlsx
backtest_results.xlsx.bak
backtest_results_changshe.xlsx
backtest_results_CS_20120915.xlsx
backtest_results.xls
backtest_results_new.xlsx

(Version) Control all the things!

- Code
- Configurations
- Data

Code version control

- Choose the right tool for your organization / workflow
- High activation energy for financial researchers

Code version control

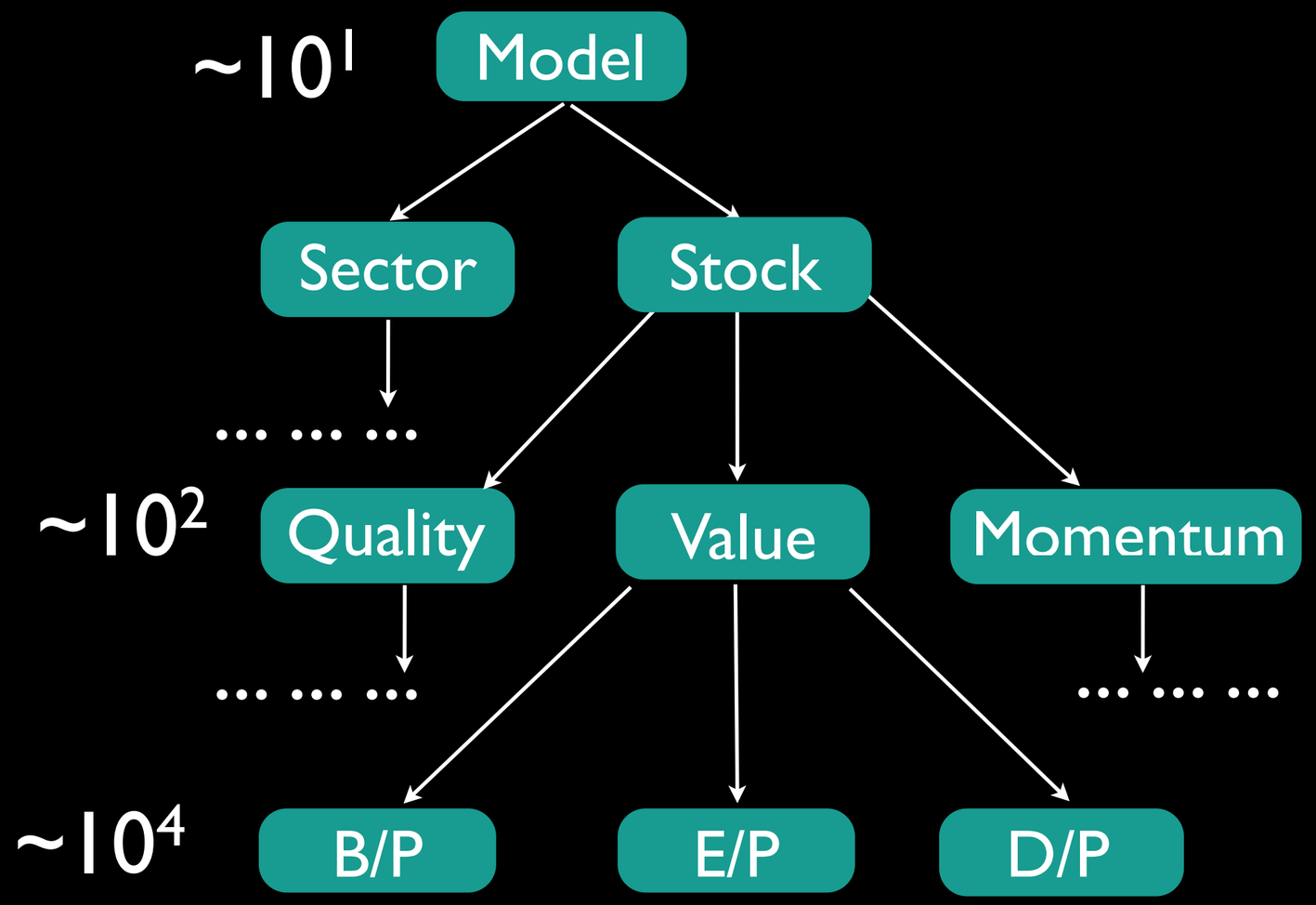
- Stop building models in Excel
  - Traceable code vs untraceable hot-key sequence
  - VBA, or “I don’t want to live on this planet anymore”
- Research code should be in languages like Python, R, Matlab
- Production code should be in Python, C++, Java, etc

Code versioning tools

- Distributed vs Centralized
- How do you collaborate?
- What's least disruptive to your workflow?

~~parameter hell~~ Configurations

- Factor tree
- Model settings
- Factor settings
- Risk model
- Trading cost model
- Constraints



Configuration version control

- Too many knobs to turn
- Version changes often unrecorded
- Multiple time dimensions

```
def get_factor_config(factor, date):
    sqlstr = """\
        SELECT * FROM Factors
        WHERE FactorName='%s' AND
            '%s' BETWEEN StartDate and EndDate
            """
    rs = database_query(sqlstr % (factor, date), CON)
    return FactorConfig.from_sql(rs)


class Factor(object):


    def config(self, date):
        value = get_factor_config(self.id, date)
        return value
```

call database

wrapper

```
def get_factor_config(factor, date):
    sqlstr = """\
        SELECT * FROM Factors
        WHERE FactorName='%s' AND
            '%s' BETWEEN StartDate and EndDate
            """
    rs = database_query(sqlstr % (factor, date), CON)
    return FactorConfig.from_sql(rs)


class Factor(object):

    _config = None

    def config(self, date):
        if (self._config is None or
            self._config.StartDate > date or
            self._config.EndDate <= date):
            self._config = get_factor_config(self.id, date)
        return self._config
```

memoize

Configuration versioning

- Part of the problem can be solved with additional date information
- But lots of ad-hoc wrapper code. (“Wait...which tables have start/end dates again?”)
- What if a particular research study requires changing historical values?

Designing parameter version control

- Centralized entry point for configuration querying
- Get and set version date / tag / hash
- Runs in different anchoring modes
  - Single anchor - use current configurations
  - Multiple anchor - switch at pre-defined dates
  - Floating anchor - match data date


IF DATA CHANGES AND NO ONE KNOWS ABOUT IT
DID IT REALLY CHANGE?
memegenerator.net

Data

- Scale problems
- Boundaries of control
- Proprietary point-in-time databases

Data versioning

- Distribution
- Diff / Merge
- Performance

Data versioning design issues

- Store locally only as needed
- Need a structure aware diff tool
- Scale of data means pure git is too space intensive
- Need for quick access means pure HG too slow

Data versioning design issues

- Need some full save-points (e.g., per day or month)
- Just store change-sets for other points
- A way to “upgrade” partial save-points to full if accessed very frequently

Dependency hell

- Firm-wide research platform
  - Must have full scientific computing stack
  - Cloneable VMs

- New research tools: http://www.pgbovine.net/cde.html


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
memegenerator.net

Testing

- Versioning is useless if you don’t test
- Testing is difficult/impossible if you don’t version
- Testing is worth the time
- Continuous integration

Testing in the financial industry

- Cavalier attitude
- Lack of good processes
- Over-reliance on compile time checks
- Reinventing the wheel, on purpose!

Unit testing

- Independent of configuration and data versions
- Data loading/cleaning/munging code
- Data transformation/computation code

Loading/cleaning/munging

```
assert data.name == expected

assert (isnull(data) == exp).all()

assert data.shape == expected

assert I am using Python + pandas
```

Core computations

```
assert result.std() == expected

assert calc() == alternate(ddof=1)

assert ar_reg(test_data) == exp
```

Data testing

```
assert has_dataitems(expected)

assert data.dtype == expected

assert data.count() == expected
```

Data testing

```
assert abs(returns) < expected

assert sector_code in GIC_CODES

assert problem_data == expected
```

Model testing

- Compute models on simple data with known answer
- Compare real model output with historical values
  - Make sure you're running on the same data (versioning)
- Need to be finish in a reasonable amount of time (where reasonable depends on frequency of changes and the workflow of your team)

Conclusion

- Take home messages
  - Organize
  - Version
  - Test
- Organization, testing, and code versioning is “easier”
- Need better tools for parameter and data versioning