Better Coding Practices for Everyone

ABIGAIL HADDAD

ACCURATE

TRANSPARENT

REPEATABLE

What do we mean by 'better'?

Who I Am

Public Policy Ph.D.

Research/stats/data science for DoD

Lead Data Scientist

https://github.com/abigailhaddad

abigail.Haddad@gmail.com

Problem

-Spreadsheets no one can understand

-Code no one can check

-How did we get that graph? Do people believe us?

End State We Want to Get To

-Someone says "how did you do that" - and you have a git repository already

-Making a change and generating everything over again is easy

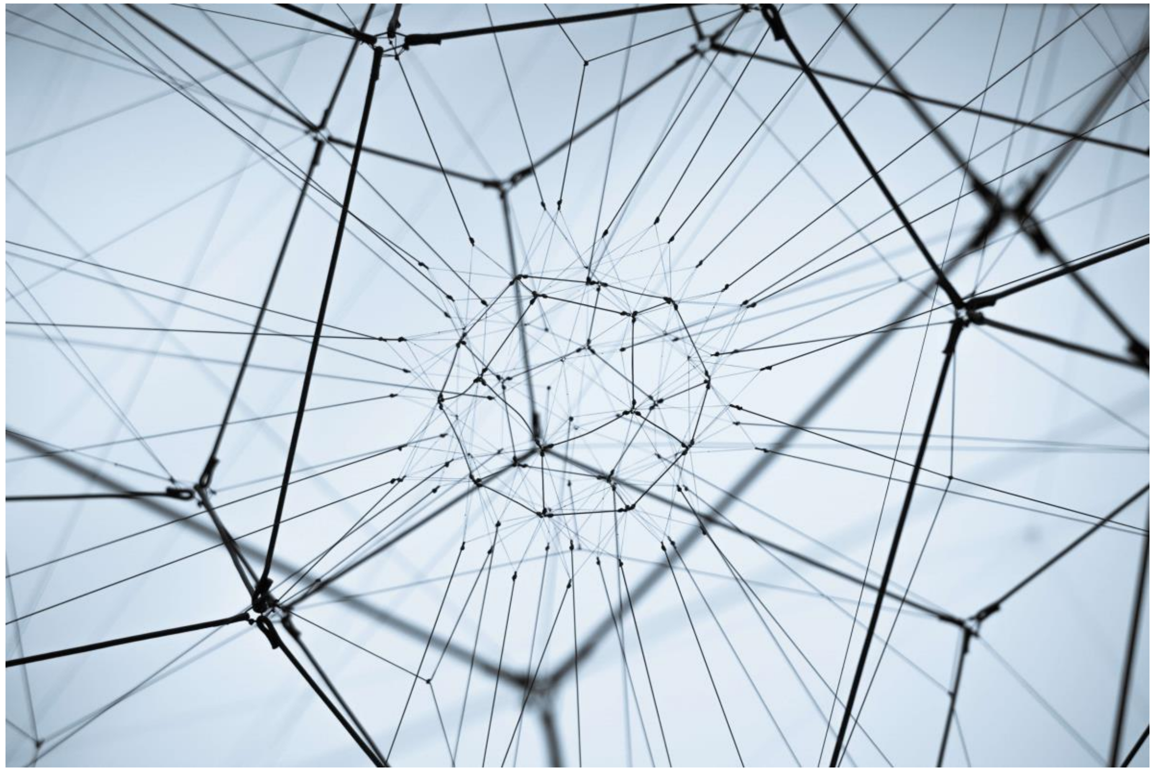
-You can catch your own errors

When is this especially important?

-External deliverable

-Complicated analysis

-Need to do this again next year



Primary Set of Solutions

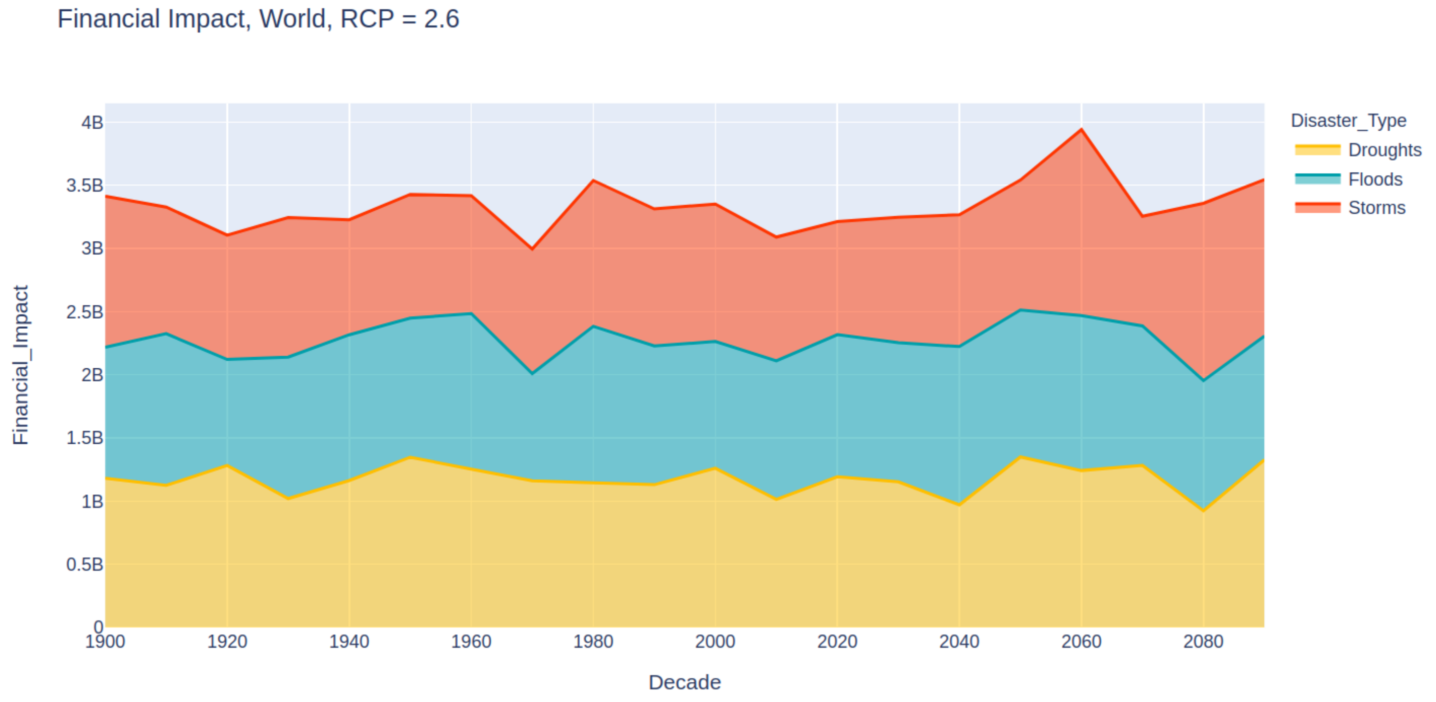
-Automating everything via your code

-Version control

-Documentation

-Writing clean code

Automating Everything



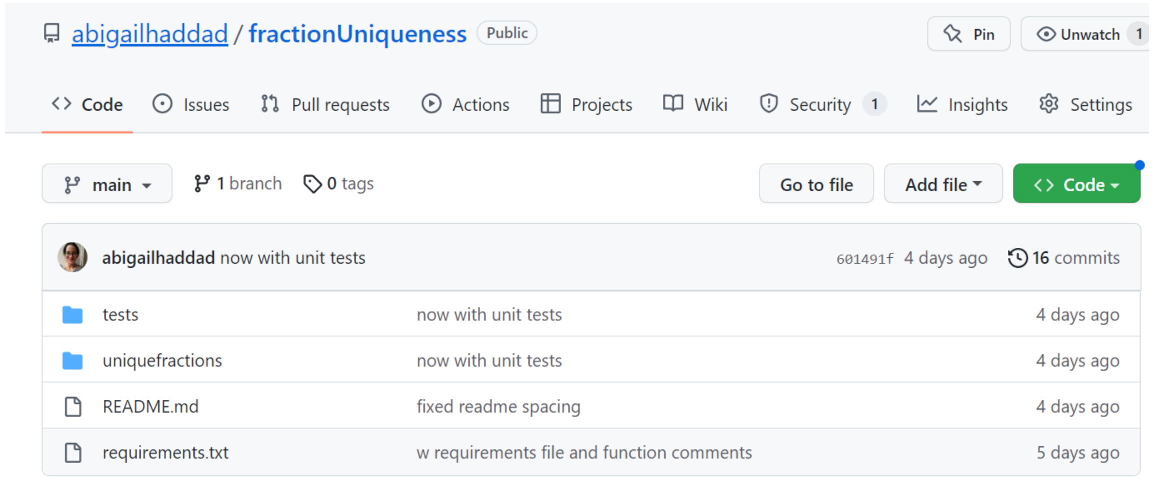
-Pulling the data

-Cleaning the data

-Making the chart

-No hardcoding

Version Control



abigailhaddad now with unit tests 601491f 4 days ago 16 commits

tests	now with unit tests	4 days ago
uniquefractions	now with unit tests	4 days ago
README.md	fixed readme spacing	4 days ago
requirements.txt	w requirements file and function comments	5 days ago

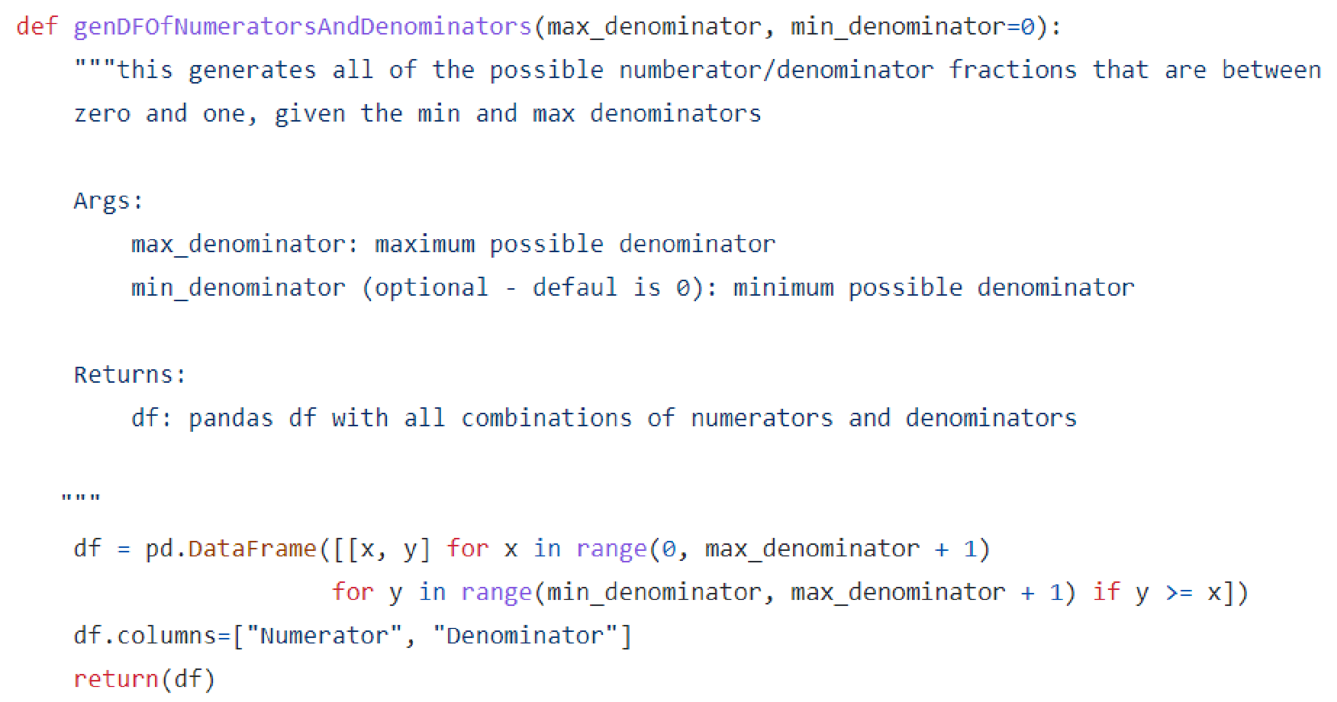
-Avoid multiple copies floating around

-Let multiple work on the same thing at once

-Easily revert back if you break something

-Compare versions

-Benefit of working in coding tools



```
def genDFOfNumeratorsAndDenominators(max_denominator, min_denominator=0):
    """this generates all of the possible numberator/denominator fractions that are between
    zero and one, given the min and max denominators

    Args:
        max_denominator: maximum possible denominator
        min_denominator (optional - defaul is 0): minimum possible denominator

    Returns:
        df: pandas df with all combinations of numerators and denominators

   """
    df = pd.DataFrame([[x, y] for x in range(0, max_denominator + 1)
                      for y in range(min_denominator, max_denominator + 1) if y >= x])
    df.columns=["Numerator", "Denominator"]
    return(df)
```

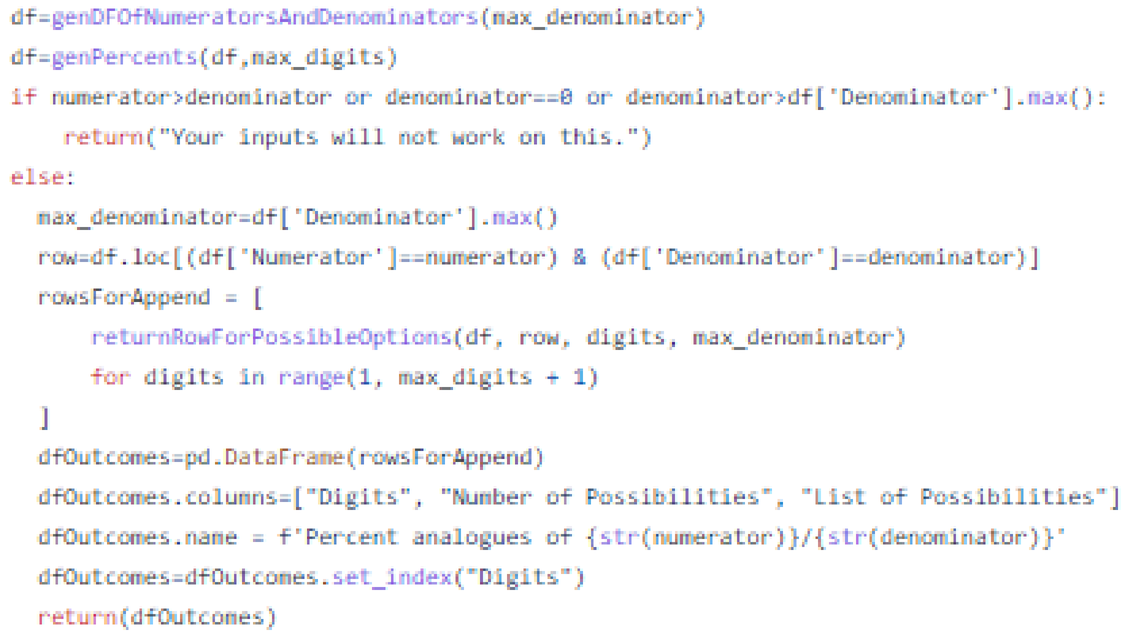
Documentation

-What did you do and why?

-Readme files

-Inputs/outputs

-Function-level code



```
df=genDFOfNumeratorsAndDenominators(max_denominator)
df=genPercents(df,max_digits)
if numerator>denominator or denominator==0 or denominator>df['Denominator'].max():
    return("Your inputs will not work on this.")
else:
  max_denominator=df['Denominator'].max()
  row=df.loc[(df['Numerator']==numerator) & (df['Denominator']==denominator)]
  rowsForAppend = [
      returnRowForPossibleOptions(df, row, digits, max_denominator)
      for digits in range(1, max_digits + 1)
  ]
  dfOutcomes=pd.DataFrame(rowsForAppend)
  dfOutcomes.columns=["Digits", "Number of Possibilities", "List of Possibilities"]
  dfOutcomes.name = f'Percent analogues of {str(numerator)}/{str(denominator)}'
  dfOutcomes=dfOutcomes.set_index("Digits")
  return(dfOutcomes)
```

Writing really clean code

-Code is in functions

-Each function does one thing

-Things are named based on what they are/do

-No complicated workflows

-Don't repeat yourself

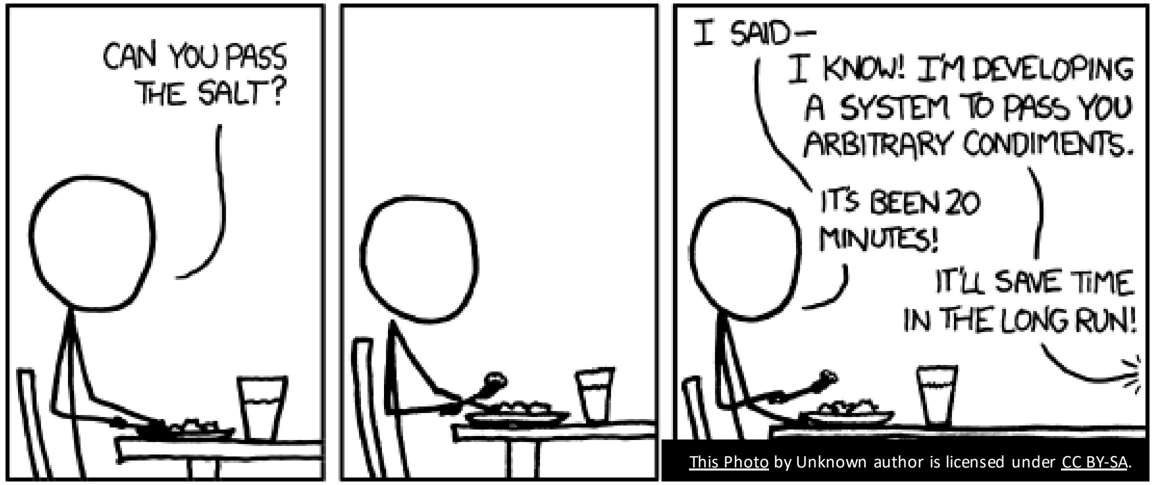
More practices

Isolation/containerization

Code reuse

Test cases

When is this not overengineering?


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

-Need to reuse code

-Need people outside your technical environment to be able to replicate it

-Want to be able to seamlessly hand something off

Isolation/Containerization

-virtual environments

-%pip install 'pandas=1.5.3'

-docker

Code Reuse

-import .py or workbooks

-modularization

Test Cases

```
class testgenDFOfNumeratorsAndDenominators(unittest.TestCase):
    def testLength(self):
        min_denominator=5
        max_denominator=50
        df=uniquefractions.genDFOfNumeratorsAndDenominators(max_denominator, min_denominator)
        self.assertEqual(len(df), 1311)
    def testAverageNumerator(self):
        min_denominator=5
        max_denominator=50
        df=uniquefractions.genDFOfNumeratorsAndDenominators(max_denominator, min_denominator)
        self.assertEqual(df['Numerator'].mean(), 16.842105263157894)
    def testAverageDenominator(self):
        min_denominator=5
        max_denominator=50
        df=uniquefractions.genDFOfNumeratorsAndDenominators(max_denominator, min_denominator)
        self.assertEqual(df['Denominator'].mean(), 33.68421052631579)
```

-Make sure it's doing what you want it to do

-When something breaks in a different way, write a new test case

DATABRICKS FUNCTIONALITY

VERSION CONTROL

- Integrate with GitHub
- Branch, clone, pull requests
- Branch protection rules

FILESTORE

- Write outputs to a folder
- Use the CLI to pull them down locally

(LIKE) COMMAND LINE

- Pip for versions
- Jobs/dbutils for cron/passing parameters
- Re-use code via importing from other workbooks, installing custom packages
- **Coming soon: new IDE**

Small Steps To Better Practices

-Do something BOTH in your current tool and using more coding-type one to make sure you're getting the right results

-Version control

-Documentation

Questions