Effective Programming Practices for Economists

16. Data Management

Hans-Martin von Gaudecker

Department of Economics, Universität Bonn

At the end of this lecture you are able to ...

- Convey the message that data management is important and non-trivial
- Understand the benefits and limitations of different tools
- Transform your data to obey the first two normal forms
- Manage your data using pandas DataFrames

Licensed under the Creative Commons Attribution License

Introduction: Importance and definitions

Example and limitations of standard tools

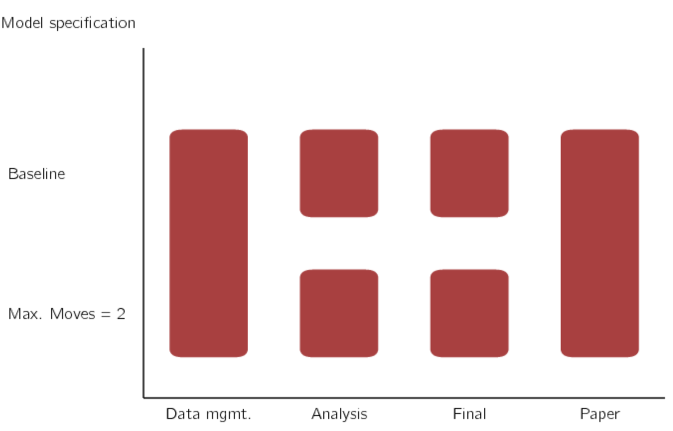
Introducing some relational database logic

First steps with pandas

Structuring a collection of DataFrames

Some fancier conditional statements, aggregating data

Project workflows revisited: Back to Go



Licensed under the Creative Commons Attribution License

Serious data management matters
Case study in Kleiber and Zeileis (2010)

- Grunfeld (1958) data – often used for illustrations of simultaneous equation models (Greene, Baltagi)
  - Several transcription errors
  - At least 5 different versions, authors mostly ignorant
- 5 variables, 220 observations – visual inspection feasible
  - Else have to infer from the code what happens to the data
  - Clarity, minimising errors, replicability are crucial
  - Oreopoulos (2008)

Licensed under the Creative Commons Attribution License

Data management is not trivial . . .

- . . . which is what you might expect given its complete absence from economic curricula

- Most things straightforward, but many steps
  - Little scope for automation – each variable is special . . .
  - Details matter (`‘if x > 0’` in Stata with missings?)

- Some references
  - Long (2008), Inter-University Consortium for Political and Social Research (ICPSR) (2009)
  - Clearer than slides – but don’t forget Git, Waf, etc.

Licensed under the Creative Commons Attribution License

Defining data management

Converting the source data to the format(s) your analysis programs (estimators, simulators, calibrators) expect.

Source data always refers to the original data – as downloaded, collected, or simulated. **Never, ever,** change these data and save it under the same name. The source data should always be the starting point of your workflow.

Similarly, never change the contents of a variable later on in the workflow (minimise state!).

Licensed under the Creative Commons Attribution License

Some of the most common operations

- Appending the data
- Merging data from various sources
- Changing the coding of variables or generating new ones
- Dealing correctly with missing values
- Selecting subsets of the data (columns, rows)
- Aggregating data by higher-level construct
  - Individual $\rightarrow$ household
  - county $\rightarrow$ state
  - ...
- Sorting the data

Licensed under the Creative Commons Attribution License

Excel?

- Probably most wide-spread tool
- Errors equally widespread: Panko (1998/2008)
- Reinhard/Rogoff as a case in point

Licensed under the Creative Commons Attribution License

When Stata is a good fit

- Stata is made for handling economic data
- Specialised tool: Intuitive and works great ...
- ... if your data has 2 (+1) dimensions
  - Individual $\times$ variables
  - Time $\times$ variables
  - Panel data with individual $\times$ time $\times$ variables
- Else: Impossible to stick to normal forms
  - Distinction data / macros bites
- Memory limitations (administrative data)

Licensed under the Creative Commons Attribution License

Example: Gaudecker (2015)
Households' portfolio diversification

1. Households report their holdings of specific assets

2. Connect these with return series from Datastream

3. Calculate diversification measures from Calvet, Campbell, and Sodini (2007)

Licensed under the Creative Commons Attribution License

Dimensions of the data after Step 2

#	Description
1	Cross section of individuals
2	Items in financial portfolios, other variables
3	For each PF item: Return series from Datastream

Trying to handle all in one Stata dataset . . .

Licensed under the Creative Commons Attribution License

Dataset in Stata

id	male	t	i_n	i_1_name	i_1_ret	i_1_share	i_2_name	i_2_ret	i_2_share
1	0	1	1	ABN Amro	0.0005	1.0	.	.	.
1	0	2	1	ABN Amro	-0.0002	1.0	.	.	.
1	0	3	1	ABN Amro	0.0014	1.0	.	.	.
1	0	4	1	ABN Amro	-0.0001	1.0	.	.	.
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2	1	1	2	RD/Shell	-0.0012	0.6	Robeco	0.0002	0.4
2	1	2	2	RD/Shell	0.0022	0.6	Robeco	0.0001	0.4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Waste of space: E.g. `male` repeated T_i times

Need to parse variable names all the time

Licensed under the Creative Commons Attribution License

Stata code for calculating within-period returns

```
qui su id
local n_hh = r(max)
gen double portf_ret = 0
forvalues i = 1 / `n_hh' {
    qui su i_n if id == `i'
    local n_xret = r(max)
    forvalues j = 1 / `n_xret' {
        replace portf_ret = portf_ret + i_`j'_ret * i_`j'_share
    }
}
```

Licensed under the Creative Commons Attribution License

Introducing some relational database logic

- Problem: Squeezing multi-dim. object into 2-dim. table
- Relational databases
  - Collections of two-dimensional tables
  - Means to describe relations between them
- SQL + Database implementation (PostgreSQL, Oracle, ...)
- Some relational database logic helps in all tools
  - pandas is built on it
  - Stata 11+'s `merge` syntax borrows heavily from SQL ideas

Licensed under the Creative Commons Attribution License

Example data in a relational DB structure

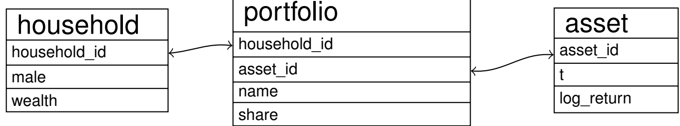
household

portfolio

asset

Licensed under the Creative Commons Attribution License

Relations between the tables



Licensed under the Creative Commons Attribution License

Some theory
The first normal form

- First normal form: Values do not have any internal structure
  - No need for parsing values before using them
  - E.g. store first names and last names separately
- Not too often a problem in economic data
  - X-digit industrial or educational classifiers
  - Store each digit level you need in a separate variable

Licensed under the Creative Commons Attribution License

Some theory
The second normal form

- Tables do not contain redundant information
  - This was severely violated in Stata attempt
  - Everything except `t` and `i_[0-9]{1,2}_ret` redundant
- Violations make things much harder and error-prone:
  - Changes to data
  - Consistency checks
  - Selecting observations
- Find yourself fighting with the second normal form?
  - Discard Stata or use it like a database

Licensed under the Creative Commons Attribution License

No structure in variable names

- There should not be different variables with similar content referring to different time periods etc.
- Example here: `i_1_ret, i_2_ret, ...`
- (In Stata parlance: Use “long” instead of “wide” format)
- (Storing data in “wide” format probably never occurred to relational database people, i.e. no normal form)

Licensed under the Creative Commons Attribution License

pandas

- High-performance, easy-to-use data structures and data analysis tool for Python
- Allows you to carry out your entire data analysis workflow in Python
- Basic features include:
  - Reading and writing data
  - Reshaping, slicing, fancy indexing and subsetting of datasets
  - Merging and joining data
  - Running statistical models (often using statsmodels)

Licensed under the Creative Commons Attribution License

Create first DataFrame with pandas

Create a simple DataFrame object with pandas by importing a from a file with comma separated values

```
>>> # Import numpy and pandas with the usual conventions.
>>> import numpy as np
>>> import pandas as pd


>>> # Import household data and create DataFrame object.
>>> household = pd.read_table('household.csv')

>>> # Display DataFrame.
>>> household

   household_id  male    wealth
0             1     0  196087.3
1             2     1  316478.7
2             3     0  294750.0
```

Licensed under the Creative Commons Attribution License

Create first DataFrame object with pandas

Set `household_id` as the DataFrame's index.

```
>>> # Set household_id as DataFrame index.
>>> household.set_index('household_id', inplace=True)

>>> # Display DataFrame.
>>> household

              male    wealth
household_id
1                0  196087.3
2                1  316478.7
3                0  294750.0
```

Licensed under the Creative Commons Attribution License

Basic operations on a DataFrame object

```
>>> # Display row labels.
>>> household.index

Int64Index([1, 2, 3], dtype=int64)
```

```
>>> # Display column labels.
>>> household.columns

Index(['male', 'wealth'], dtype=object)
```

Licensed under the Creative Commons Attribution License

Basic operations on a DataFrame object

```
>>> # Access column 'wealth'.
>>> household['wealth']

household_id
1              196087.3
2              316478.7
3              294750.0
Name: wealth, dtype: float64
```

```
>>> # Alternative way of achieving the same thing.
>>> household.wealth

household_id
1              196087.3
2              316478.7
3              294750.0
Name: wealth, dtype: float64
```

Licensed under the Creative Commons Attribution License

Basic operations on a DataFrame object

```
>>> # Access row with household_id == 2.
>>> household.loc[2]

male           1.0
wealth    316478.7
Name: 2, dtype: float64


>>> # Access second row by integer location.
>>> household.iloc[1]

male           1.0
wealth    316478.7
Name: 2, dtype: float64
```

Licensed under the Creative Commons Attribution License

Basic operations on a DataFrame object

```
>>> # Obtain 'male' for household_id == 3.
>>> household.loc[3]['male']

0


>>> # Or the other way around.
>>> household['male'].loc[3]

0
```

Licensed under the Creative Commons Attribution License

Insert an observation into the DataFrame

First create a new DataFrame object with a `household_id` that has not been used before.

```
>>> # Create a new observation.
>>> new_obs = pd.DataFrame([
...     {
...         'household_id': 4,
...         'male': 1,
...         'wealth': 261534.3
...     }
... ])
```

```
>>> new_obs
```

```
   household_id  male    wealth
0             4     1  261534.3
```

Licensed under the Creative Commons Attribution License

Insert an observation into the DataFrame

Now append to the main DataFrame.

```
>>> # Append new observation, get a new DataFrame object.
>>> household.append(new_obs)

   household_id  male    wealth
1           NaN     0  196087.3
2           NaN     1  316478.7
3           NaN     0  294750.0
0             4     1  261534.3
```

Lots of missing values …

Licensed under the Creative Commons Attribution License

Insert an observation into the DataFrame

Correct the index of the DataFrame with the new observation.

```
>>> # Set index to id column for new observation.
>>> new_obs.set_index('household_id', inplace=True)
>>> new_obs

              male    wealth
household_id
4                1  261534.3


>>> # Append new observation to household.
>>> household = household.append(new_obs)
>>> household

              male    wealth
household_id
1                0  196087.3
2                1  316478.7
3                0  294750.0
4                1  261534.3
```

Licensed under the Creative Commons Attribution License

Excerpts of the other two tables

Import portfolio, set composite index.

```
>>> # Import portfolio data, set a MultiIndex
>>> portfolio = pd.read_table(
...     'portfolio.csv',
...     index_col=['household_id', 'asset_id']
... )

>>> portfolio

                                                     name  share
household_id asset_id

1            nl0000301109                        ABN Amro   1.00
2            nl0000289783                          Robeco   0.40
             gb00b03mlx29              Royal Dutch Shell   0.60
3            gb00b03mlx29              Royal Dutch Shell   0.15
             lu0197800237  AAB Eastern Europe Equity Fund   0.60
             nl0000289965          Postbank BioTech Fonds   0.25
4            NaN                                      NaN   1.00
```

Licensed under the Creative Commons Attribution License

Excerpts of the other two tables

Import asset data, again with composite index

```
>>> # Import asset data and create DataFrame object.
>>> asset = pd.read_table('asset.txt', index_col=['asset_id', 't'])
```

```
>>> # Display 10 rows of asset returns.
>>> asset.iloc[230:240]
```

```
                 log_return
asset_id     t

gb00b03mlx29 231  -0.026809
             232   0.037107
             233   0.096050
             234  -0.065241
             235   0.035324
lu0197800237 180   0.030254
             181   0.036997
             182   0.128663
             183  -0.068121
             184  -0.054926
```

Licensed under the Creative Commons Attribution License

Selecting data: The very basics

Select the `wealth` variable from the `household` DataFrame

```
>>> # Select 'wealth' as a Series.
>>> household['wealth']

household_id
1               196087.3
2               316478.7
3               294750.0
4               261534.3
Name: wealth, dtype: float64


>>> # Select 'wealth' as a DataFrame.
>>> household[['wealth']]

                wealth
household_id
1             196087.3
2             316478.7
3             294750.0
4             261534.3
```

Licensed under the Creative Commons Attribution License

Select conditional subsets of data

```
>>> # Note the '&' instead of 'and'.
>>> household[
...     (household.index < 3)
...     &
...     (household['wealth'] < 250 * 1000)
... ]

              male    wealth
household_id
1                0  196087.3


>>> # Only keep a single Series.
>>> portfolio[portfolio['share'] > 0.5]['name']

household_id  asset_id
1             nl0000301109                          ABN Amro
2             gb00b03mlx29                Royal Dutch Shell
3             lu0197800237    AAB Eastern Europe Equity Fund
4             NaN                                        NaN
Name: name, dtype: object
```

Licensed under the Creative Commons Attribution License

The `query` method

Reduce verbosity via DataFrames' `query` method:

```
>>> household.query('household_id < 3 & wealth < 250000')

              male    wealth
household_id
1                0  196087.3
```

Licensed under the Creative Commons Attribution License

Joining data from several tables: One-to-one

Prepare for adding some socio-economic data to `household`:

```
>>> # Create a new DataFrame.
>>> hh_socio = pd.DataFrame(
...     {
...         'kids': [0, 1, 1, 0],
...         'married': [1, 0, 1, 0]
...     },
...     index=pd.Index([4, 1, 3, 2], name='household_id')
... )
```

```
>>> hh_socio
```

```
              kids  married
household_id
4                0        1
1                1        0
3                1        1
2                0        0
```

Licensed under the Creative Commons Attribution License

Joining data from several tables: One-to-one

```
>>> # Join both DataFrames one-to-one on household_id.
>>> household.join(hh_socio)

              male    wealth  kids  married
household_id
              
1                0  196087.3     1        0
2                1  316478.7     0        0
3                0  294750.0     1        1
4                1  261534.3     0        1
```

Licensed under the Creative Commons Attribution License

Joining data from several tables: Many-to-one

Add `wealth` variable from `household` to `portfolio`.

Start with some preparations for more concise display

```
>>> # Select portfolio where household_id < 3.
... pf_selection = portfolio[
...     portfolio.index.get_level_values('household_id') < 3
... ]
>>> pf_selection
```

```
                                           name  share
household_id asset_id
1            nl0000301109          ABN Amro    1.0
2            nl0000289783            Robeco    0.4
             gb00b03mlx29  Royal Dutch Shell    0.6
```

Licensed under the Creative Commons Attribution License

Joining data from several tables: Many-to-one

```
>>> household_portfolios = household.join(pf_selection)

>>> household_portfolios

                          male    wealth                name  share
household_id asset_id
                                                               
1            nl0000301109    0  196087.3            ABN Amro    1.0
2            nl0000289783    1  316478.7              Robeco    0.4
             gb00b03mlx29    1  316478.7  Royal Dutch Shell    0.6
```

Licensed under the Creative Commons Attribution License

Joining data from several tables: Many-to-many

Merge `asset` and `portfolio` data on `asset_id`.

Currently not possible with `join` keyword, see explanation ↗

```
>>> mtm = pd.merge(
...     portfolio.reset_index(),
...     asset.reset_index(),
...     on=['asset_id']
... ).set_index(['household_id', 'asset_id', 't'])
```

Licensed under the Creative Commons Attribution License

Joining data from several tables: Many-to-many

```
>>> mtm.iloc[:5].append(mtm.iloc[300:305]).append(mtm.iloc[500:505])
                                                name  share  log_return
household_id asset_id     t
1            nl0000301109 9             ABN Amro    1.0   -0.108815
                          10            ABN Amro    1.0    0.009683
                          11            ABN Amro    1.0   -0.012898
                          12            ABN Amro    1.0    0.025632
                          13            ABN Amro    1.0    0.087747
2            nl0000289783 89              Robeco    0.4    0.072231
                          90              Robeco    0.4    0.074398
                          91              Robeco    0.4    0.119277
                          92              Robeco    0.4   -0.105887
                          93              Robeco    0.4    0.036558
             gb00b03mlx29 54  Royal Dutch Shell    0.6   -0.038197
                          55  Royal Dutch Shell    0.6    0.061536
                          56  Royal Dutch Shell    0.6    0.003005
                          57  Royal Dutch Shell    0.6   -0.046255
                          58  Royal Dutch Shell    0.6    0.046955
```

Licensed under the Creative Commons Attribution License

Altering the structure of the data

- Add some more asset characteristics
  - Type $\in$ `{stock, fund}`
- Where to put it?
- Column `name` in the table `portfolio`
  1. Unclearly labelled
  2. Violates 2nd normal form
- The name for the table `asset`
  - Misnomer – `monthly_return` much better

Licensed under the Creative Commons Attribution License

Altering the structure of the data

Create `monthly_return` and construct useful `asset` table.

```
>>> monthly_return = asset

>>> # Select unique 'asset_id' from portfolio, i.e. drop duplicates.
>>> asset = portfolio.reset_index().drop_duplicates(['asset_id'])
>>> # Keep only 'asset_id' and 'name' in DataFrame, set index.
>>> asset = asset[['asset_id', 'name']]
>>> asset = asset.set_index('asset_id', inplace=False)
>>> asset
                                             name
asset_id

nl0000301109                             ABN Amro
nl0000289783                               Robeco
gb00b03mlx29                    Royal Dutch Shell
lu0197800237  AAB Eastern Europe Equity Fund
nl0000289965             Postbank BioTech Fonds
NaN                                           NaN
```

Licensed under the Creative Commons Attribution License

Altering the structure of the data

Select **unique** values of `asset_id` from `portfolio` as a basis for a new table `asset`

```
>>> # Drop missing values
>>> asset.dropna(inplace=True)

>>> # Set the type of the asset.
... asset['type'] = pd.Categorical(
...     ['stock', 'fund', 'stock', 'fund', 'fund'],
...     ordered=False
... )
>>> asset
```

```
                                          name   type
asset_id

nl0000301109                          ABN Amro  stock
nl0000289783                            Robeco   fund
gb00b03mlx29                Royal Dutch Shell  stock
lu0197800237  AAB Eastern Europe Equity Fund   fund
nl0000289965          Postbank BioTech Fonds   fund
```

Licensed under the Creative Commons Attribution License

Altering the structure of the data

Remove the `name` column from `portfolio` — the resulting table does little more than describing the relation between `household` and `asset`

```
>>> # Drop column 'name'.
>>> portfolio = portfolio.drop('name', axis=1)
>>> portfolio
```

```
                            share
household_id asset_id

1            nl0000301109   1.00
2            nl0000289783   0.40
             gb00b03mlx29   0.60
3            gb00b03mlx29   0.15
             lu0197800237   0.60
             nl0000289965   0.25
4            NaN            1.00
```

Licensed under the Creative Commons Attribution License

Select all stocks in portfolio, show wealth alongside

```
>>> # Boolean Series to select only stocks.
>>> stocks = asset['type'] == 'stock'

>>> # Merge portfolio and asset tables.
>>> # (NB: join has issues with missing values)
>>> df = pd.merge(
>>>     portfolio.reset_index(),
>>>     asset[stocks][['type', 'name']].reset_index(),
>>>     on=['asset_id']
>>> ).set_index(['household_id', 'asset_id'])

>>> df
```

```
                          share   type                 name
household_id asset_id

1            nl0000301109  1.00  stock             ABN Amro
2            gb00b03mlx29  0.60  stock   Royal Dutch Shell
3            gb00b03mlx29  0.15  stock   Royal Dutch Shell
```

Licensed under the Creative Commons Attribution License

Select all stocks in portfolio, show wealth alongside

```
>>> # Merge relevant parts of portfolio and asset tables.
>>> df = pd.merge(
>>>     df[['name']].reset_index(),
>>>     household[['wealth']].reset_index(),
>>>     on=['household_id']
>>> ).set_index(['household_id', 'asset_id'])

>>> df

                                         name    wealth
household_id asset_id

1            nl0000301109            ABN Amro  196087.3
2            gb00b03mlx29  Royal Dutch Shell  316478.7
3            gb00b03mlx29  Royal Dutch Shell  294750.0
```

Licensed under the Creative Commons Attribution License

Select returns of ABN Amro in the first year of data

```
>>> # Get the identifier of ABN Amro, safeguard against duplicates.
>>> abn_isin = asset[asset['name'] == 'ABN Amro'].index
>>> assert len(abn_isin) == 1
>>> abn_isin = abn_isin[0]
>>> abn_isin

'nl0000301109'
```

```
>>> # Merge monthly return with asset in a new DataFrame object.
>>> abn_ret = monthly_return.loc[abn_isin]
>>> abn_ret.loc[:12]

    log_return
t
9    -0.108815
10    0.009683
11   -0.012898
12    0.025632
```

Licensed under the Creative Commons Attribution License

#obs with wealth info, average wealth

```
>>> # Count observations with wealth information.
>>> household['wealth'].count()

4
```

```
>>> # Average wealth for non-missing observations.
>>> household['wealth'].mean()

267212.57500000001
```

Licensed under the Creative Commons Attribution License

#obs with wealth info, average wealth if wealth is below 300k

```
>>> # Condition on wealth below 300,000.
>>> below_300k = household['wealth'] < 300000
```

```
>>> # Count observations with wealth information.
>>> household[below_300k]['wealth'].count()

3
```

```
>>> # Average wealth for non-missing observations.
>>> household[below_300k]['wealth'].mean()

250790.53333333333
```

Licensed under the Creative Commons Attribution License

Average return for each asset, # obs. in time series, display the result nicely

```
>>> # Merge monthly_return with asset to obtain assets' name.
>>> return_by_name = monthly_return.join(asset[['name']]).reset_index()

>>> # Keep only relevant columns, group by name.
>>> return_grouped = return_by_name.groupby('name')

>>> # Aggregate columns
... return_grouped.aggregate({
...     't': np.count_nonzero,
...     'log_return': np.mean
... })

                                  t  log_return
name

AAB Eastern Europe Equity Fund   48   -0.000769
ABN Amro                        212    0.014442
Postbank BioTech Fonds           64   -0.000929
Robeco                          235    0.003650
Royal Dutch Shell               235    0.006768
```

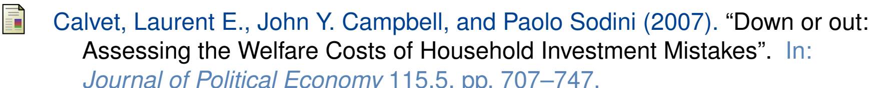
Licensed under the Creative Commons Attribution License

At the end of this lecture you are able to ...

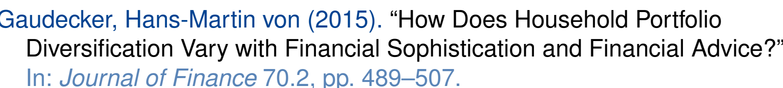
- Convey the message that data management is important and non-trivial
- Understand the benefits and limitations of different tools
- Transform your data to obey the first two normal forms
- Manage your data using pandas DataFrames

Licensed under the Creative Commons Attribution License

References I



Calvet, Laurent E., John Y. Campbell, and Paolo Sodini (2007). “Down or out: Assessing the Welfare Costs of Household Investment Mistakes”. In: *Journal of Political Economy* 115.5, pp. 707–747.



Gaudecker, Hans-Martin von (2015). “How Does Household Portfolio Diversification Vary with Financial Sophistication and Financial Advice?” In: *Journal of Finance* 70.2, pp. 489–507.

Grunfeld, Yehuda (1958). “The Determinants of Corporate Investment”. Ph.D. thesis, Department of Economics, University of Chicago.

Inter-University Consortium for Political and Social Research (ICPSR) (2009). *Guide to Social Science Data Preparation and Archiving: Best Practice throughout the Data Life Cycle*. 4th ed. Available at `http://www.icpsr.umich.edu/files/ICPSR/access/dataprep.pdf`. Ann Arbor, MI.

Kleiber, Christian and Achim Zeileis (2010). “The Grunfeld Data at 50”. In: *German Economic Review* 11.4, pp. 404–417.

Long, J. Scott (2008). *The Workflow of Data Analysis Using Stata*. Stata Press.

Licensed under the Creative Commons Attribution License

References II

Oreopoulos, Philip (2008). “Estimating Average and Local Average Treatment Effects of Education when Compulsory Schooling Laws Really Matter: Corrigendum”. Available at http://dx.doi.org/10.1257/000282806776157641.

Panko, Raymond R. (1998/2008). “What We Know about Spreadsheet Errors”. In: *Journal of End User Computing* 10.2. Revised version 2008 available at http://panko.shidler.hawaii.edu/ssr/Mypapers/whatknow.htm, pp. 15–21.

Licensed under the Creative Commons Attribution License

Acknowledgements

- This course is designed after and borrows a lot from the `Software Carpentry` course designed by `Greg Wilson` for scientists and engineers.

- The Software Carpentry course material is made available under a `Creative Commons Attribution License`, as is this course's material.

License for the course material

[Links to the full `legal` text and the source text for `this` page.] You are free:

- **to Share** to copy, distribute and transmit the work
- **to Remix** to adapt the work

Under the following conditions:

- **Attribution** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

- **Waiver** Any of the above conditions can be waived if you get permission from the copyright holder.
- **Public Domain** Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- **Other Rights** In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to `this` web page.