Imperial College
London

A fleet of packages for inputting United Kingdom primary care data

Roger B. Newson
r.newson@imperial.ac.uk
http://www.rogernewsonresources.org.uk

Department of Primary Care and Public Health, Imperial College London

To be presented at the 2018 London Stata Conference,
06–07 September, 2018
To be downloadable from the conference website at
http://ideas.repec.org/s/boc/usug18.html

The Clinical Practice Research Datalink (CPRD)

- The **Clinical Practice Research Datalink**[1][2] is a centrally–managed data warehouse, storing data provided by the primary–care sector of the United Kingdom (UK) National Health Service (NHS).
- Primary-care practices (contractors to the NHS) send regular consignments of data to CPRD about what has happened to *consenting* patients registered with them since the previous data consignment, using Vision® software.
- These data are combined by CPRD to form an enormous database, with one enormous dataset for each of a list of classes of *things*, such as patients, practices, or prescriptions.
- Medical researchers may ask for **retrievals**, each comprising a set of text datasets on defined subsets of those *things*.

The Clinical Practice Research Datalink (CPRD)

- The **Clinical Practice Research Datalink**[1][2] is a centrally–managed data warehouse, storing data provided by the primary–care sector of the United Kingdom (UK) National Health Service (NHS).
- Primary-care practices (contractors to the NHS) send regular consignments of data to CPRD about what has happened to *consenting* patients registered with them since the previous data consignment, using Vision® software.
- These data are combined by CPRD to form an enormous database, with one enormous dataset for each of a list of classes of *things*, such as patients, practices, or prescriptions.
- Medical researchers may ask for **retrievals**, each comprising a set of text datasets on defined subsets of those *things*.

The Clinical Practice Research Datalink (CPRD)

- The **Clinical Practice Research Datalink**[1][2] is a centrally–managed data warehouse, storing data provided by the primary–care sector of the United Kingdom (UK) National Health Service (NHS).
- Primary-care practices (contractors to the NHS) send regular consignments of data to CPRD about what has happened to *consenting* patients registered with them since the previous data consignment, using Vision® software.
- These data are combined by CPRD to form an enormous database, with one enormous dataset for each of a list of classes of *things*, such as patients, practices, or prescriptions.
- Medical researchers may ask for **retrievals**, each comprising a set of text datasets on defined subsets of those *things*.

The Clinical Practice Research Datalink (CPRD)

- The **Clinical Practice Research Datalink**[1][2] is a centrally–managed data warehouse, storing data provided by the primary–care sector of the United Kingdom (UK) National Health Service (NHS).
- Primary-care practices (contractors to the NHS) send regular consignments of data to CPRD about what has happened to *consenting* patients registered with them since the previous data consignment, using Vision® software.
- These data are combined by CPRD to form an enormous database, with one enormous dataset for each of a list of classes of *things*, such as patients, practices, or prescriptions.
- Medical researchers may ask for **retrievals**, each comprising a set of text datasets on defined subsets of those *things*.

The Clinical Practice Research Datalink (CPRD)

- The **Clinical Practice Research Datalink**[1][2] is a centrally–managed data warehouse, storing data provided by the primary–care sector of the United Kingdom (UK) National Health Service (NHS).
- Primary-care practices (contractors to the NHS) send regular consignments of data to CPRD about what has happened to *consenting* patients registered with them since the previous data consignment, using Vision® software.
- These data are combined by CPRD to form an enormous database, with one enormous dataset for each of a list of classes of *things*, such as patients, practices, or prescriptions.
- Medical researchers may ask for **retrievals**, each comprising a set of text datasets on defined subsets of those *things*.

The `cprdutil` fleet of packages

- To save time spent re–inventing wheels, I have been developing a flagship package `cprdutil` to input the text files of a CPRD retrieval into Stata.
- `cprdutil` has a module to input each type of text file, using just one line of Stata code.
- Each module is an ado–file, which inputs a text file of the appropriate type into a **primary dataset** in memory, complete with value labels, variable labels, and numeric Stata dates.
- I have also developed a fleet of **satellite packages**, such as `cprdlinkutil` and `cprdhesutil`, to input text files containing non–CPRD data linked to CPRD patients, such as hospitalization records.
- All these Stata datasets can then be used to define **secondary datasets**, possibly with one observation per patient per year, which can be used in a statistical analysis.

The `cprdutil` fleet of packages

- To save time spent re–inventing wheels, I have been developing a flagship package `cprdutil` to input the text files of a CPRD retrieval into Stata.
- `cprdutil` has a module to input each type of text file, using just one line of Stata code.
- Each module is an ado–file, which inputs a text file of the appropriate type into a **primary dataset** in memory, complete with value labels, variable labels, and numeric Stata dates.
- I have also developed a fleet of **satellite packages**, such as `cprdlinkutil` and `cprdhesutil`, to input text files containing non–CPRD data linked to CPRD patients, such as hospitalization records.
- All these Stata datasets can then be used to define **secondary datasets**, possibly with one observation per patient per year, which can be used in a statistical analysis.

The `cprdutil` fleet of packages

- To save time spent re–inventing wheels, I have been developing a flagship package `cprdutil` to input the text files of a CPRD retrieval into Stata.
- `cprdutil` has a module to input each type of text file, using just one line of Stata code.
- Each module is an ado–file, which inputs a text file of the appropriate type into a **primary dataset** in memory, complete with value labels, variable labels, and numeric Stata dates.
- I have also developed a fleet of **satellite packages**, such as `cprdlinkutil` and `cprdhesutil`, to input text files containing non–CPRD data linked to CPRD patients, such as hospitalization records.
- All these Stata datasets can then be used to define **secondary datasets**, possibly with one observation per patient per year, which can be used in a statistical analysis.

The `cprdutil` fleet of packages

- To save time spent re–inventing wheels, I have been developing a flagship package `cprdutil` to input the text files of a CPRD retrieval into Stata.
- `cprdutil` has a module to input each type of text file, using just one line of Stata code.
- Each module is an ado–file, which inputs a text file of the appropriate type into a **primary dataset** in memory, complete with value labels, variable labels, and numeric Stata dates.
- I have also developed a fleet of **satellite packages**, such as `cprdlinkutil` and `cprdhesutil`, to input text files containing non–CPRD data linked to CPRD patients, such as hospitalization records.
- All these Stata datasets can then be used to define **secondary datasets**, possibly with one observation per patient per year, which can be used in a statistical analysis.

The `cprdutil` fleet of packages

- To save time spent re–inventing wheels, I have been developing a flagship package `cprdutil` to input the text files of a CPRD retrieval into Stata.
- `cprdutil` has a module to input each type of text file, using just one line of Stata code.
- Each module is an ado–file, which inputs a text file of the appropriate type into a **primary dataset** in memory, complete with value labels, variable labels, and numeric Stata dates.
- I have also developed a fleet of **satellite packages**, such as `cprdlinkutil` and `cprdhesutil`, to input text files containing non–CPRD data linked to CPRD patients, such as hospitalization records.
- All these Stata datasets can then be used to define **secondary datasets**, possibly with one observation per patient per year, which can be used in a statistical analysis.

The `cprdutil` fleet of packages

- To save time spent re–inventing wheels, I have been developing a flagship package `cprdutil` to input the text files of a CPRD retrieval into Stata.
- `cprdutil` has a module to input each type of text file, using just one line of Stata code.
- Each module is an ado–file, which inputs a text file of the appropriate type into a **primary dataset** in memory, complete with value labels, variable labels, and numeric Stata dates.
- I have also developed a fleet of **satellite packages**, such as `cprdlinkutil` and `cprdhesutil`, to input text files containing non–CPRD data linked to CPRD patients, such as hospitalization records.
- All these Stata datasets can then be used to define **secondary datasets**, possibly with one observation per patient per year, which can be used in a statistical analysis.

Text files supplied in a CPRD data retrieval

- These fall into 3 classes (supplied in 3 distinct folders), namely **data files**, **non–*XYZ* lookup files**, and ***XYZ*** **lookup files**.
- **Data files** (or non–lookup files) have 1 row for each of a class of *things* known to UK primary–care practitioners, such as patients, practices, prescriptions, tests, or clinical diagnoses.
- These *things* may or may not be identifiable uniquely using a **primary key** of variables, such as a patient ID or a practice ID.
- **Non–*XYZ* lookup files** have 1 row for each value of a **code variable**, and descriptive data on what that code value means.
- Examples include a `product` file with 1 row per CPRD prescribed–product code (`prodcode`), and a `medical` file with 1 row per CPRD medical–term code (`medcode`).
- ***XYZ*** **lookup files** have 3–character filenames (as in *XYZ*`.txt`), have 1 numeric key variable `code` and 1 other string variable, and are used to define Stata **value labels**.

Text files supplied in a CPRD data retrieval

- These fall into 3 classes (supplied in 3 distinct folders), namely **data files**, **non–*XYZ* lookup files**, and ***XYZ* lookup files**.
- **Data files** (or non–lookup files) have 1 row for each of a class of *things* known to UK primary–care practitioners, such as patients, practices, prescriptions, tests, or clinical diagnoses.
- These *things* may or may not be identifiable uniquely using a **primary key** of variables, such as a patient ID or a practice ID.
- **Non–*XYZ* lookup files** have 1 row for each value of a **code variable**, and descriptive data on what that code value means.
- Examples include a `product` file with 1 row per CPRD prescribed–product code (`prodcode`), and a `medical` file with 1 row per CPRD medical–term code (`medcode`).
- ***XYZ* lookup files** have 3–character filenames (as in *XYZ*`.txt`), have 1 numeric key variable `code` and 1 other string variable, and are used to define Stata **value labels**.

Text files supplied in a CPRD data retrieval

- These fall into 3 classes (supplied in 3 distinct folders), namely **data files**, **non–*XYZ* lookup files**, and ***XYZ* lookup files**.
- **Data files** (or non–lookup files) have 1 row for each of a class of *things* known to UK primary–care practitioners, such as patients, practices, prescriptions, tests, or clinical diagnoses.
- These *things* may or may not be identifiable uniquely using a **primary key** of variables, such as a patient ID or a practice ID.
- **Non–*XYZ* lookup files** have 1 row for each value of a **code variable**, and descriptive data on what that code value means.
- Examples include a `product` file with 1 row per CPRD prescribed–product code (`prodcode`), and a `medical` file with 1 row per CPRD medical–term code (`medcode`).
- ***XYZ* lookup files** have 3–character filenames (as in *XYZ*`.txt`), have 1 numeric key variable `code` and 1 other string variable, and are used to define Stata **value labels**.

Text files supplied in a CPRD data retrieval

- These fall into 3 classes (supplied in 3 distinct folders), namely **data files**, **non–*XYZ* lookup files**, and ***XYZ* lookup files**.
- **Data files** (or non–lookup files) have 1 row for each of a class of *things* known to UK primary–care practitioners, such as patients, practices, prescriptions, tests, or clinical diagnoses.
- These *things* may or may not be identifiable uniquely using a **primary key** of variables, such as a patient ID or a practice ID.
- **Non–*XYZ* lookup files** have 1 row for each value of a **code variable**, and descriptive data on what that code value means.
- Examples include a `product` file with 1 row per CPRD prescribed–product code (`prodcode`), and a `medical` file with 1 row per CPRD medical–term code (`medcode`).
- ***XYZ* lookup files** have 3–character filenames (as in *XYZ*`.txt`), have 1 numeric key variable `code` and 1 other string variable, and are used to define Stata **value labels**.

Text files supplied in a CPRD data retrieval

- These fall into 3 classes (supplied in 3 distinct folders), namely **data files**, **non–*XYZ* lookup files**, and ***XYZ* lookup files**.
- **Data files** (or non–lookup files) have 1 row for each of a class of *things* known to UK primary–care practitioners, such as patients, practices, prescriptions, tests, or clinical diagnoses.
- These *things* may or may not be identifiable uniquely using a **primary key** of variables, such as a patient ID or a practice ID.
- **Non–*XYZ* lookup files** have 1 row for each value of a **code variable**, and descriptive data on what that code value means.
- Examples include a `product` file with 1 row per CPRD prescribed–product code (`prodcode`), and a `medical` file with 1 row per CPRD medical–term code (`medcode`).
- ***XYZ* lookup files** have 3–character filenames (as in *XYZ*`.txt`), have 1 numeric key variable `code` and 1 other string variable, and are used to define Stata **value labels**.

Text files supplied in a CPRD data retrieval

- These fall into 3 classes (supplied in 3 distinct folders), namely **data files**, **non–*XYZ* lookup files**, and ***XYZ* lookup files**.
- **Data files** (or non–lookup files) have 1 row for each of a class of *things* known to UK primary–care practitioners, such as patients, practices, prescriptions, tests, or clinical diagnoses.
- These *things* may or may not be identifiable uniquely using a **primary key** of variables, such as a patient ID or a practice ID.
- **Non–*XYZ* lookup files** have 1 row for each value of a **code variable**, and descriptive data on what that code value means.
- Examples include a `product` file with 1 row per CPRD prescribed–product code (`prodcode`), and a `medical` file with 1 row per CPRD medical–term code (`medcode`).
- ***XYZ* lookup files** have 3–character filenames (as in *XYZ*`.txt`), have 1 numeric key variable `code` and 1 other string variable, and are used to define Stata **value labels**.

Text files supplied in a CPRD data retrieval

- These fall into 3 classes (supplied in 3 distinct folders), namely **data files**, **non–*XYZ* lookup files**, and ***XYZ*** **lookup files**.
- **Data files** (or non–lookup files) have 1 row for each of a class of *things* known to UK primary–care practitioners, such as patients, practices, prescriptions, tests, or clinical diagnoses.
- These *things* may or may not be identifiable uniquely using a **primary key** of variables, such as a patient ID or a practice ID.
- **Non–*XYZ* lookup files** have 1 row for each value of a **code variable**, and descriptive data on what that code value means.
- Examples include a `product` file with 1 row per CPRD prescribed–product code (`prodcode`), and a `medical` file with 1 row per CPRD medical–term code (`medcode`).
- ***XYZ*** **lookup files** have 3–character filenames (as in ***XYZ*****.**`txt`), have 1 numeric key variable `code` and 1 other string variable, and are used to define Stata **value labels**.

`cprdutil` modules for inputting data files

These ado–files input a text data file with 1 row per *thing*, and create a dataset in memory, with 1 observation per *thing*. These observations may be **keyed** (sorted and uniquely identified) by numeric ID variables.

Module:	*Creates a dataset with 1 observation per:*	*With key variables:*
`cprd_patient`	Patient	`patid`
`cprd_practice`	Practice	`pracid`
`cprd_staff`	NHS staff member	`staffid`
`cprd_consultation`	Consultation event	`patid`, `consid`
`cprd_clinical`	Medical–history event (e.g. a diagnosis)	Unkeyed
`cprd_additional`	Additional data on a medical–history event	`patid`, `adid`
`cprd_referral`	Referral event (e.g. to a hospital specialist)	Unkeyed
`cprd_immunisation`	Immunisation event	Unkeyed
`cprd_test`	Test event (e.g. recording blood pressures)	Unkeyed
`cprd_therapy`	Prescription event	Unkeyed

The variables `patid`, `pracid`, `staffid`, `consid`, and `adid` are *anonymized* numeric ID variables for patient, practice, staff member, consultation, and additional clinical data record, respectively. Events *should* always have dates, which are converted to numeric Stata dates.

`cprdutil` modules for inputting non–*XYZ* lookup files

Each of these ado–files inputs a text data file with 1 row per value of an **internal CPRD code variable**, usually used as a **foreign key** in at least 1 other CPRD file type. It creates a **lookup dataset** in memory, with 1 observation per code–variable value, and descriptive data on that code value.

Module:	*Creates a dataset with 1 observation per:*	*With key variable:*
`cprd_medical`	Medical–term code	`medcode`
`cprd_product`	Prescribed product or therapy code	`prodcode`
`cprd_entity`	Entity type (format for reading string fields)	`enttype`
`cprd_scoremethod`	Scoring methodology	`code`
`cprd_packtype`	Pack type for prescribed products	`packtype`
`cprd_bnfcodes`	British National Formulary code	`bnfcode`
`cprd_common_dosages`	Common dosage for prescribed products	`dosageid`
`cprd_batchnumber`	Immunisation batch number	`batch`

Alternatively, the user may use the module `cprd_nonxyzlookup`, which calls *all* these modules in sequence, outputting the Stata datasets to disk files in a user–specified directory.

`cprdutil` modules for inputting *XYZ* lookup files to create value labels

- There is only one module for this, whose name is `cprd_xyzlookup`.
- It inputs a directory to search for the *XYZ*`.txt` lookup files, and outputs a **generated Stata do–file** (specified by the `dofile()` option).
- This generated do–file will contain a long list of `label define` commands, defining the whole set of CPRD value labels specified by the *XYZ*`.txt` files (nearly 100 of them).
- The modules for inputting *data* files also have a `dofile()` option, this time specifying an *input* do–file, to be run to define the value labels.
- *So*, if we run `cprd_xyzlookup` first to generate the do–file, then we can run the modules for inputting data files afterwards, specifying the generated do–file as their input do–file.

`cprdutil` modules for inputting *XYZ* lookup files to create value labels

- There is only one module for this, whose name is `cprd_xyzlookup`.
- It inputs a directory to search for the *XYZ*`.txt` lookup files, and outputs a **generated Stata do–file** (specified by the `dofile()` option).
- This generated do–file will contain a long list of `label define` commands, defining the whole set of CPRD value labels specified by the *XYZ*`.txt` files (nearly 100 of them).
- The modules for inputting *data* files also have a `dofile()` option, this time specifying an *input* do–file, to be run to define the value labels.
- *So*, if we run `cprd_xyzlookup` first to generate the do–file, then we can run the modules for inputting data files afterwards, specifying the generated do–file as their input do–file.

`cprdutil` modules for inputting *XYZ* lookup files to create value labels

- There is only one module for this, whose name is `cprd_xyzlookup`.
- It inputs a directory to search for the *XYZ*`.txt` lookup files, and outputs a **generated Stata do–file** (specified by the `dofile()` option).
- This generated do–file will contain a long list of `label define` commands, defining the whole set of CPRD value labels specified by the *XYZ*`.txt` files (nearly 100 of them).
- The modules for inputting *data* files also have a `dofile()` option, this time specifying an *input* do–file, to be run to define the value labels.
- *So*, if we run `cprd_xyzlookup` first to generate the do–file, then we can run the modules for inputting data files afterwards, specifying the generated do–file as their input do–file.

`cprdutil` modules for inputting *XYZ* lookup files to create value labels

- There is only one module for this, whose name is `cprd_xyzlookup`.
- It inputs a directory to search for the *XYZ*`.txt` lookup files, and outputs a **generated Stata do–file** (specified by the `dofile()` option).
- This generated do–file will contain a long list of `label define` commands, defining the whole set of CPRD value labels specified by the *XYZ*`.txt` files (nearly 100 of them).
- The modules for inputting *data* files also have a `dofile()` option, this time specifying an *input* do–file, to be run to define the value labels.
- *So*, if we run `cprd_xyzlookup` first to generate the do–file, then we can run the modules for inputting data files afterwards, specifying the generated do–file as their input do–file.

`cprdutil` modules for inputting *XYZ* lookup files to create value labels

- There is only one module for this, whose name is `cprd_xyzlookup`.
- It inputs a directory to search for the *XYZ*`.txt` lookup files, and outputs a **generated Stata do–file** (specified by the `dofile()` option).
- This generated do–file will contain a long list of `label define` commands, defining the whole set of CPRD value labels specified by the *XYZ*`.txt` files (nearly 100 of them).
- The modules for inputting *data* files also have a `dofile()` option, this time specifying an *input* do–file, to be run to define the value labels.
- *So*, if we run `cprd_xyzlookup` first to generate the do–file, then we can run the modules for inputting data files afterwards, specifying the generated do–file as their input do–file.

`cprdutil` modules for inputting *XYZ* lookup files to create value labels

- There is only one module for this, whose name is `cprd_xyzlookup`.
- It inputs a directory to search for the *XYZ*`.txt` lookup files, and outputs a **generated Stata do–file** (specified by the `dofile()` option).
- This generated do–file will contain a long list of `label define` commands, defining the whole set of CPRD value labels specified by the *XYZ*`.txt` files (nearly 100 of them).
- The modules for inputting *data* files also have a `dofile()` option, this time specifying an *input* do–file, to be run to define the value labels.
- *So*, if we run `cprd_xyzlookup` first to generate the do–file, then we can run the modules for inputting data files afterwards, specifying the generated do–file as their input do–file.

Sensible programming using master and servant do–files with `dolog`

- We have seen that creating even a simple CPRD database may seem complicated to most people.
- To minimize these complications, it helps to follow sensible programming practices.
- In particular, it makes sense to write a **master do–file**, which calls a sequence of **servant do–files**, in the correct order of execution.
- A useful tool is the SSC package `dolog`, which executes a named do–file *ABC*`.do` to create a log file *ABC*`.log` with the same filename in the same directory.
- The master do–file typically is itself executed using a `do` command, and contains a sequence of `dolog` commands, one to execute each servant do–file (in the correct order).
- *So*, we can test the servant do–files one by one, in the correct order, and be able to re–execute the whole sequence later, using a `do` command to call the master do–file to call the servant do–files (in the correct order), to create the log files and the datasets.

Sensible programming using master and servant do–files with `dolog`

- We have seen that creating even a simple CPRD database may seem complicated to most people.
- To minimize these complications, it helps to follow sensible programming practices.
- In particular, it makes sense to write a **master do–file**, which calls a sequence of **servant do–files**, in the correct order of execution.
- A useful tool is the SSC package `dolog`, which executes a named do–file *ABC*`.do` to create a log file *ABC*`.log` with the same filename in the same directory.
- The master do–file typically is itself executed using a `do` command, and contains a sequence of `dolog` commands, one to execute each servant do–file (in the correct order).
- *So*, we can test the servant do–files one by one, in the correct order, and be able to re–execute the whole sequence later, using a `do` command to call the master do–file to call the servant do–files (in the correct order), to create the log files and the datasets.

Sensible programming using master and servant do–files with `dolog`

- We have seen that creating even a simple CPRD database may seem complicated to most people.
- To minimize these complications, it helps to follow sensible programming practices.
- In particular, it makes sense to write a **master do–file**, which calls a sequence of **servant do–files**, in the correct order of execution.
- A useful tool is the SSC package `dolog`, which executes a named do–file *ABC*`.do` to create a log file *ABC*`.log` with the same filename in the same directory.
- The master do–file typically is itself executed using a `do` command, and contains a sequence of `dolog` commands, one to execute each servant do–file (in the correct order).
- *So*, we can test the servant do–files one by one, in the correct order, and be able to re–execute the whole sequence later, using a `do` command to call the master do–file to call the servant do–files (in the correct order), to create the log files and the datasets.

Sensible programming using master and servant do–files with `dolog`

- We have seen that creating even a simple CPRD database may seem complicated to most people.
- To minimize these complications, it helps to follow sensible programming practices.
- In particular, it makes sense to write a **master do–file**, which calls a sequence of **servant do–files**, in the correct order of execution.
- A useful tool is the SSC package `dolog`, which executes a named do–file *ABC*`.do` to create a log file *ABC*`.log` with the same filename in the same directory.
- The master do–file typically is itself executed using a `do` command, and contains a sequence of `dolog` commands, one to execute each servant do–file (in the correct order).
- *So*, we can test the servant do–files one by one, in the correct order, and be able to re–execute the whole sequence later, using a `do` command to call the master do–file to call the servant do–files (in the correct order), to create the log files and the datasets.

Sensible programming using master and servant do–files with `dolog`

- We have seen that creating even a simple CPRD database may seem complicated to most people.
- To minimize these complications, it helps to follow sensible programming practices.
- In particular, it makes sense to write a **master do–file**, which calls a sequence of **servant do–files**, in the correct order of execution.
- A useful tool is the SSC package `dolog`, which executes a named do–file *ABC*`.do` to create a log file *ABC*`.log` with the same filename in the same directory.
- The master do–file typically is itself executed using a `do` command, and contains a sequence of `dolog` commands, one to execute each servant do–file (in the correct order).
- *So*, we can test the servant do–files one by one, in the correct order, and be able to re–execute the whole sequence later, using a `do` command to call the master do–file to call the servant do–files (in the correct order), to create the log files and the datasets.

Sensible programming using master and servant do–files with `dolog`

- We have seen that creating even a simple CPRD database may seem complicated to most people.
- To minimize these complications, it helps to follow sensible programming practices.
- In particular, it makes sense to write a **master do–file**, which calls a sequence of **servant do–files**, in the correct order of execution.
- A useful tool is the SSC package `dolog`, which executes a named do–file *ABC*`.do` to create a log file *ABC*`.log` with the same filename in the same directory.
- The master do–file typically is itself executed using a `do` command, and contains a sequence of `dolog` commands, one to execute each servant do–file (in the correct order).
- *So*, we can test the servant do–files one by one, in the correct order, and be able to re–execute the whole sequence later, using a `do` command to call the master do–file to call the servant do–files (in the correct order), to create the log files and the datasets.

Sensible programming using master and servant do–files with `dolog`

- We have seen that creating even a simple CPRD database may seem complicated to most people.
- To minimize these complications, it helps to follow sensible programming practices.
- In particular, it makes sense to write a **master do–file**, which calls a sequence of **servant do–files**, in the correct order of execution.
- A useful tool is the SSC package `dolog`, which executes a named do–file *ABC*`.do` to create a log file *ABC*`.log` with the same filename in the same directory.
- The master do–file typically is itself executed using a `do` command, and contains a sequence of `dolog` commands, one to execute each servant do–file (in the correct order).
- *So*, we can test the servant do–files one by one, in the correct order, and be able to re–execute the whole sequence later, using a `do` command to call the master do–file to call the servant do–files (in the correct order), to create the log files and the datasets.

A master do–file `create.do` to create a minimal CPRD database

This master do–file calls 3 servant do–files. If CPRD has sent us a retrieval, then the servant do–file `lookups.do` can input the lookups data, the servant do–file `practice.do` can input the practice data, and the servant do–file `patient.do` can input the patient data. All these data are saved to disk in forms that can be understood by Stata, and by Stata users.

```
#delim ;
version 13.1;
*
 Create a minimal CPRD database
*;

dolog lookups;
dolog practice;
dolog patient;

exit;
```

Note that the order of execution matters, because `lookups.do` generates a do–file, which is used by `practice.do` and `patient.do` to define the value labels.

A servant do–file `lookups.do` to create the lookup do–file and datasets

This servant do–file uses `cprd_xyzlookup` to create a do–file `xyzlookuplabs.do`. It then uses `cprd_nonxyzlookup` to create the non–*XYZ* lookup datasets in the subfolder `./dta`.

```
#delim ;
version 13.1;
*
 Create lookups for a CPRD database
*;

* Folder containing input text files *;
global CPRDDATA "../../../../cprddata";

* Create do-file and datasets *;
cprd_xyzlookup, txtdirspec($CPRDDATA/Lookups/TXTFILES) dofile(xyzlookuplabs.do, replace);
cprd_nonxyzlookup, txtdirspec($CPRDDATA/Lookups) dtadirspec(./dta) replace;

exit;
```

Note that we store the name of the **root folder** containing the CPRD text data in a global macro `CPRDDATA`. This is also good programming practice, in case the directory tree is rearranged.

A few lines of the generated do–file `xyzlookuplabs.do`

This do–file was generated by the servant do–file `lookups.do`, using the module `cprd_xyzlookup`. It contains enough `label define` commands to define nearly 100 value labels.

```
label define aar 0 `"Data Not Entered"', modify
label define aar 1 `"Not at risk"', modify
label define aar 2 `"Previous history of severe attack"', modify
label define aar 3 `"On 3 or more drugs"', modify
label define aar 4 `"Night symptoms"', modify
label define aar 5 `"Recent hospital admission"', modify
label define aar 6 `"Other reason"', modify
label define abo 0 `"Data Not Entered"', modify
label define abo 1 `"A"', modify
label define abo 2 `"A+"', modify
label define abo 3 `"A-"', modify
label define abo 4 `"B"', modify
label define abo 5 `"B+"', modify
label define abo 6 `"B-"', modify
label define abo 7 `"O"', modify
label define abo 8 `"O+"', modify
label define abo 9 `"O-"', modify
label define abo 10 `"AB"', modify
label define abo 11 `"AB+"', modify
label define abo 12 `"AB-"', modify
label define abo 13 `"Rhesus +"', modify
label define abo 14 `"Rhesus -"', modify
```

Note that we only have enough space to show the first few lines!

A servant do–file `practice.do` to create the practice dataset

This servant do–file uses `cprd_practice` to input a practice text data file `practice.txt`, using the `dofile()` option to execute the generated do–file `xyzlookuplabs.do` that we saw in the previous frame.

```
#delim ;
version 13.1;
*
 Create dataset practice with 1 obs per practice
*;

* Folder containing input text files *;
global CPRDDATA "../../../../cprddata";

* Create and save practice dataset *;
cprd_practice using $CPRDDATA/Data/practice.txt, clear dofile(xyzlookuplabs.do);
save ./dta/practice, replace;

exit;
```

The generated dataset in memory is then saved to a disk file `practice.dta` in the subfolder `./dta`.

The practice dataset saved to `practice.dta`

And here is a `describe` of the new dataset, with 1 observation per practice. Note that the dataset is keyed by the variable `pracid`, and contains a variable `region` with an automatically–generated value label `prg`, and 2 numeric Stata date variables `lcd_n` and `uts_n`, computed from the string date variables `lcd` and `uts`, respectively.

```
Contains data
  obs:            96
 vars:             6
 size:         2,592
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
pracid          int     %10.0g                Practice Identifier
region          byte    %22.0g     prg        Region
lcd             str10   %10s                  Last Collection Date
uts             str10   %10s                  Up To Standard Date
lcd_n           int     %td..                 Last collection date for practice
uts_n           int     %td..                 Up to standard date for practice
-------------------------------------------------------------------------------
Sorted by: pracid
     Note: Dataset has changed since last saved.
```

`uts_n` contains the first date when the practice was considered by CPRD to be sending in up–to–standard data. `lcd_n` contains the date of the most recent data consignment sent to CPRD by the practice.

Patient datasets and added observation–window variables

- The module `cprd_patient` is used to input a text patient dataset on disk into a Stata patient dataset in memory.
- The module `cprd_patientobs` is then used to add **observation–window variables** to the Stata patient dataset in memory, using an existing Stata practice dataset on disk.
- The variable `entrydate` is added as the maximum of the practice's up–to–standard date, the date when the patient joined the practice, and the patient's earliest–possible birth date.
- The variable `exitdate` is added as the *minimum* of the practice's most recent data–collection date, the date when the patient left the practice, and the patient's death date.
- The patient's **observation window** with CPRD is the span of days from the patient's `entrydate` value to the patient's `exitdate` value (inclusively).
- Note that this observation window can be empty, as a patient's `exitdate` value can be *before* the patient's `entrydate` value.

Patient datasets and added observation–window variables

- The module `cprd_patient` is used to input a text patient dataset on disk into a Stata patient dataset in memory.
- The module `cprd_patientobs` is then used to add **observation–window variables** to the Stata patient dataset in memory, using an existing Stata practice dataset on disk.
- The variable `entrydate` is added as the maximum of the practice's up–to–standard date, the date when the patient joined the practice, and the patient's earliest–possible birth date.
- The variable `exitdate` is added as the *minimum* of the practice's most recent data–collection date, the date when the patient left the practice, and the patient's death date.
- The patient's **observation window** with CPRD is the span of days from the patient's `entrydate` value to the patient's `exitdate` value (inclusively).
- Note that this observation window can be empty, as a patient's `exitdate` value can be *before* the patient's `entrydate` value.

Patient datasets and added observation–window variables

- The module `cprd_patient` is used to input a text patient dataset on disk into a Stata patient dataset in memory.
- The module `cprd_patientobs` is then used to add **observation–window variables** to the Stata patient dataset in memory, using an existing Stata practice dataset on disk.
- The variable `entrydate` is added as the maximum of the practice's up–to–standard date, the date when the patient joined the practice, and the patient's earliest–possible birth date.
- The variable `exitdate` is added as the *minimum* of the practice's most recent data–collection date, the date when the patient left the practice, and the patient's death date.
- The patient's **observation window** with CPRD is the span of days from the patient's `entrydate` value to the patient's `exitdate` value (inclusively).
- Note that this observation window can be empty, as a patient's `exitdate` value can be *before* the patient's `entrydate` value.

Patient datasets and added observation–window variables

- The module `cprd_patient` is used to input a text patient dataset on disk into a Stata patient dataset in memory.
- The module `cprd_patientobs` is then used to add **observation–window variables** to the Stata patient dataset in memory, using an existing Stata practice dataset on disk.
- The variable `entrydate` is added as the maximum of the practice's up–to–standard date, the date when the patient joined the practice, and the patient's earliest–possible birth date.
- The variable `exitdate` is added as the *minimum* of the practice's most recent data–collection date, the date when the patient left the practice, and the patient's death date.
- The patient's **observation window** with CPRD is the span of days from the patient's `entrydate` value to the patient's `exitdate` value (inclusively).
- Note that this observation window can be empty, as a patient's `exitdate` value can be *before* the patient's `entrydate` value.

Patient datasets and added observation–window variables

- The module `cprd_patient` is used to input a text patient dataset on disk into a Stata patient dataset in memory.
- The module `cprd_patientobs` is then used to add **observation–window variables** to the Stata patient dataset in memory, using an existing Stata practice dataset on disk.
- The variable `entrydate` is added as the maximum of the practice's up–to–standard date, the date when the patient joined the practice, and the patient's earliest–possible birth date.
- The variable `exitdate` is added as the *minimum* of the practice's most recent data–collection date, the date when the patient left the practice, and the patient's death date.
- The patient's **observation window** with CPRD is the span of days from the patient's `entrydate` value to the patient's `exitdate` value (inclusively).
- Note that this observation window can be empty, as a patient's `exitdate` value can be *before* the patient's `entrydate` value.

Patient datasets and added observation–window variables

- The module `cprd_patient` is used to input a text patient dataset on disk into a Stata patient dataset in memory.
- The module `cprd_patientobs` is then used to add **observation–window variables** to the Stata patient dataset in memory, using an existing Stata practice dataset on disk.
- The variable `entrydate` is added as the maximum of the practice's up–to–standard date, the date when the patient joined the practice, and the patient's earliest–possible birth date.
- The variable `exitdate` is added as the *minimum* of the practice's most recent data–collection date, the date when the patient left the practice, and the patient's death date.
- The patient's **observation window** with CPRD is the span of days from the patient's `entrydate` value to the patient's `exitdate` value (inclusively).
- Note that this observation window can be empty, as a patient's `exitdate` value can be *before* the patient's `entrydate` value.

Patient datasets and added observation–window variables

- The module `cprd_patient` is used to input a text patient dataset on disk into a Stata patient dataset in memory.
- The module `cprd_patientobs` is then used to add **observation–window variables** to the Stata patient dataset in memory, using an existing Stata practice dataset on disk.
- The variable `entrydate` is added as the maximum of the practice's up–to–standard date, the date when the patient joined the practice, and the patient's earliest–possible birth date.
- The variable `exitdate` is added as the *minimum* of the practice's most recent data–collection date, the date when the patient left the practice, and the patient's death date.
- The patient's **observation window** with CPRD is the span of days from the patient's `entrydate` value to the patient's `exitdate` value (inclusively).
- Note that this observation window can be empty, as a patient's `exitdate` value can be *before* the patient's `entrydate` value.

A servant do–file `patient.do` to create the patient dataset

This servant do–file uses `cprd_patient` to input a patient text data file `patient.txt`, using the `dofile()` option again to define the value labels. We then use `cprd_patientobs` to add the additional observation–window variables to the patient dataset in memory, using the practice dataset that we made earlier.

```
#delim ;
version 13.1;
*
 Create dataset patient with 1 obs per patient
*;

* Folder containing input text files *;
global CPRDDATA "../../../../cprddata";

* Create and save patient dataset *;
cprd_patient using $CPRDDATA/Data/patient.txt, clear dofile(xyzlookuplabs.do);
cprd_patientobs using ./dta/practice, accept;
save ./dta/patient, replace;

exit;
```

The extended patient dataset is then saved to a file `patient.dta` in the `./dta` subfolder.

The patient dataset saved to `patient.dta`

When we `describe` the patient dataset, we see that it has 33 variables in total. Here are the first 10 of them.

```
Contains data from ./dta/patient.dta
  obs:           441
 vars:            33                          9 Oct 2017 17:21
 size:        38,808
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
patid           long    %10.0g                Patient Identifier
vmid            long    %10.0g                VAMP Identifier
gender          byte    %16.0g     sex        Patient Gender
yob             int     %10.0g                Birth Year
mob             byte    %10.0g                Birth Month
marital         byte    %19.0g     mar        Marital Status
famnum          long    %10.0g                Family Number
chsreg          byte    %16.0g     y_n        CHS Registered
chsdate         str1    %9s                   CHS Registration Date
prescr          byte    %60.0g     pex        Prescription Exemption
```

Note that these variables are complete with variable labels, and *sometimes* value labels (with 3–letter names), generated automatically from the *XYZ*`.txt` lookups of the same names provided by CPRD.

Observation–window variables in the patient dataset in `patient.dta`

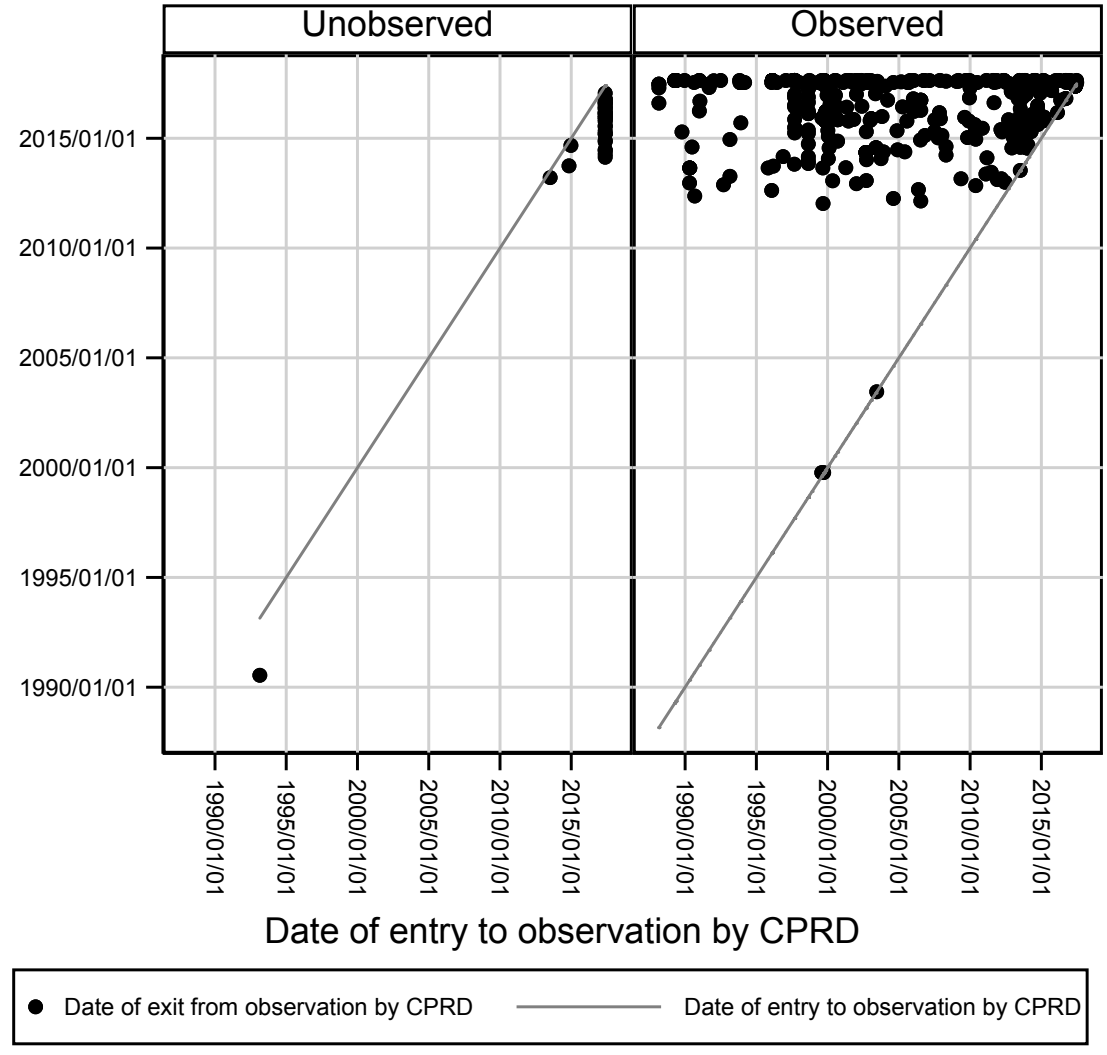
These are the last 6 of the variables in the patient dataset. They were also generated automatically, by the `cprd_patientobs` module, using the practice dataset that we made earlier.

```
obscalc         byte    %23.0g     obscalc    Observation window calculated
entrystat       byte    %23.0g     entrystat
                                                Entry status to observation by CPRD
entrydate       int     %td..                  Date of entry to observation by CPRD
exitstat        byte    %21.0g     exitstat   Exit status from observation by CPRD
exitdate        int     %td..                  Date of exit from observation by CPRD
observed        byte    %10.0g     observed   Patient observed by CPRD
------------------------------------------------------------------------------------
Sorted by: patid
```

The binary variable `obscalc` indicates that the observation window was calculated for a patient. The variables `entrystat` and `entrydate` give the mode and date, respectively, of patient entry to CPRD observation. The variables `exitstat` and `exitdate` give the mode and date, respectively, of patient exit from CPRD observation. And the binary variable `observed` indicates whether the patient has a non–empty observation window.

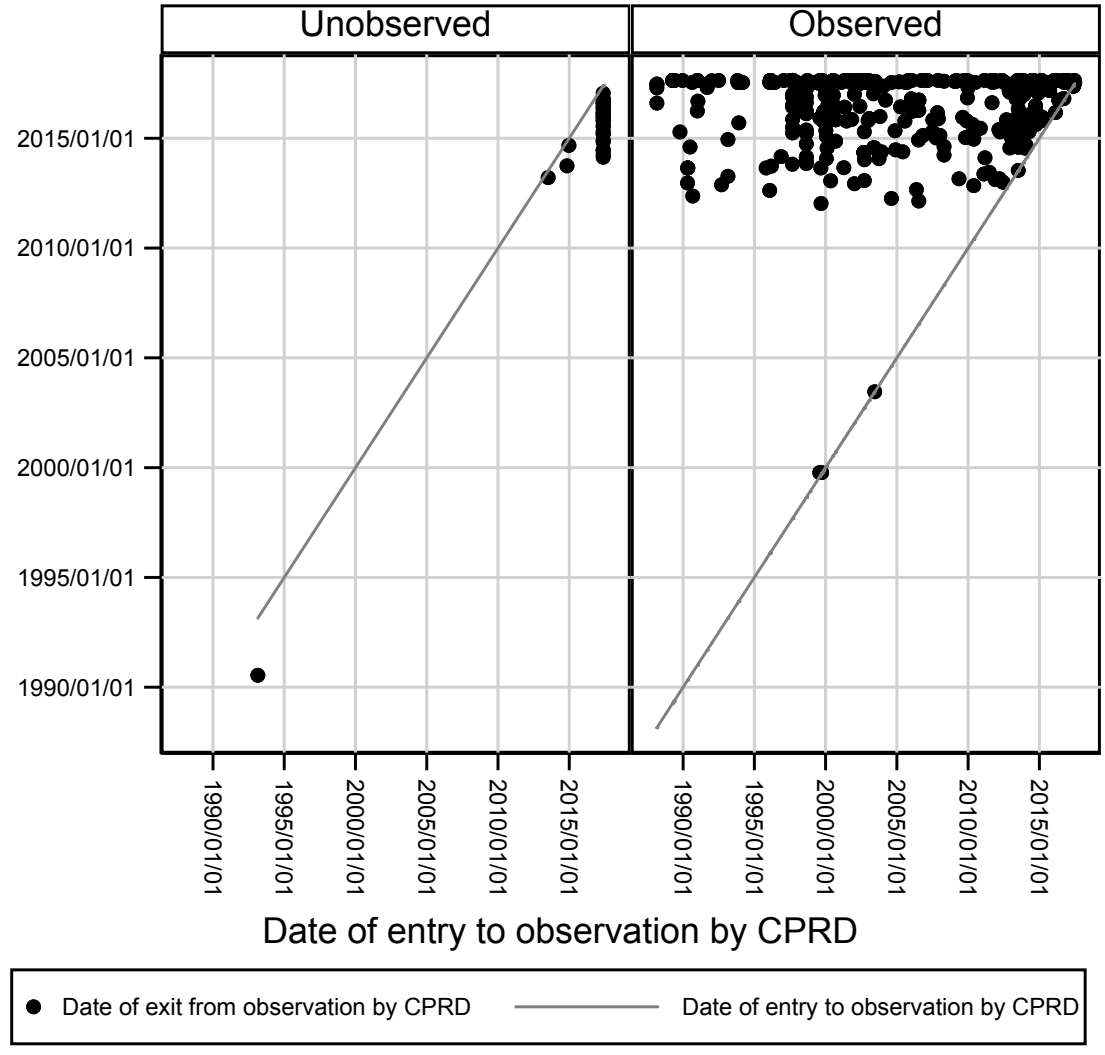
Plot of exit date against entry date for unobserved and observed patients

- And here are plots of exit date (on the vertical axis) against entry date (on the horizontal axis), with diagonal **equality lines**.
- The two graphs show patients with empty and non–empty observation windows, respectively.
- The "unobserved" patients, with pre–entry exit dates and empty observation windows, should be excluded from observational analyses.



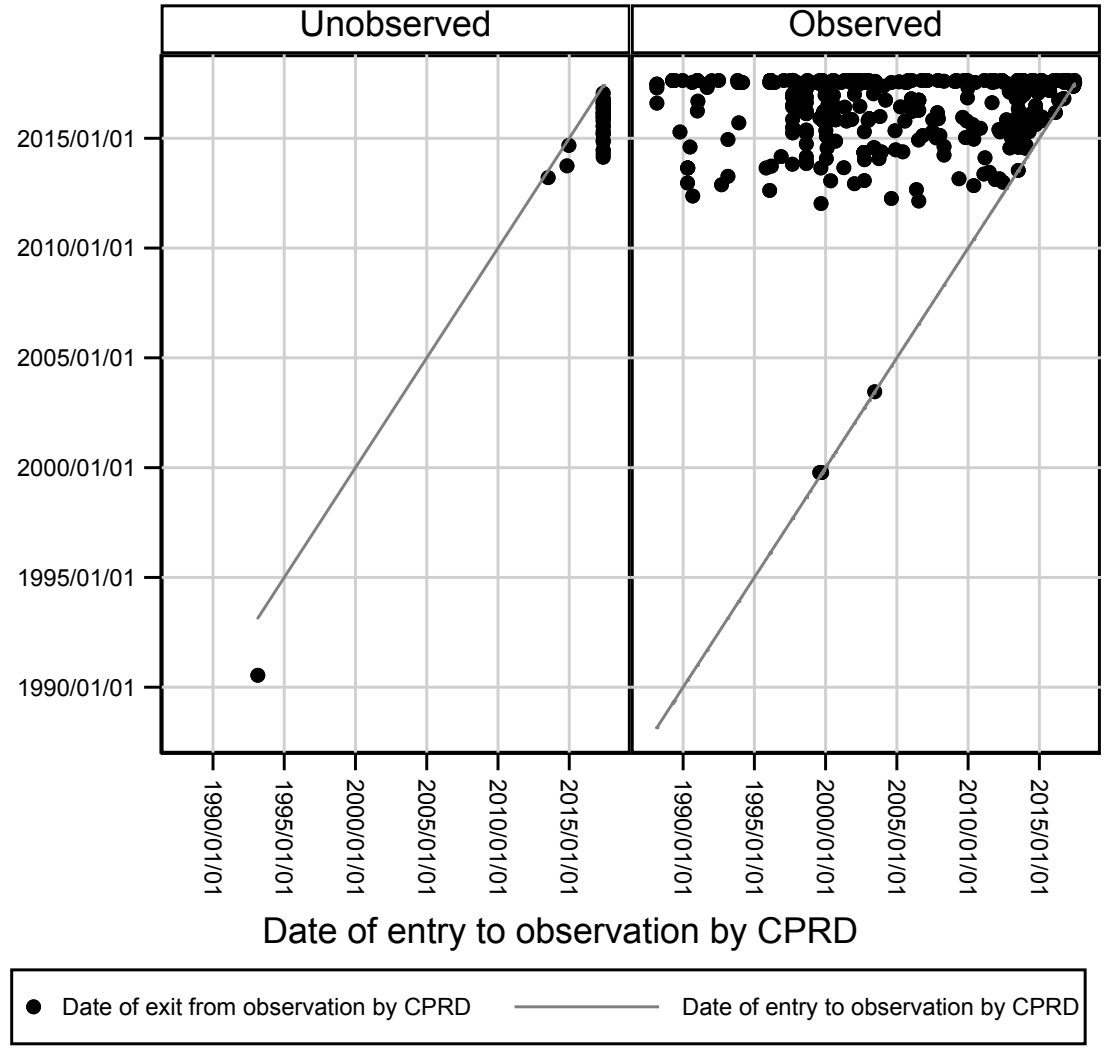
Plot of exit date against entry date for unobserved and observed patients

- And here are plots of exit date (on the vertical axis) against entry date (on the horizontal axis), with diagonal **equality lines**.
- The two graphs show patients with empty and non–empty observation windows, respectively.
- The "unobserved" patients, with pre–entry exit dates and empty observation windows, should be excluded from observational analyses.



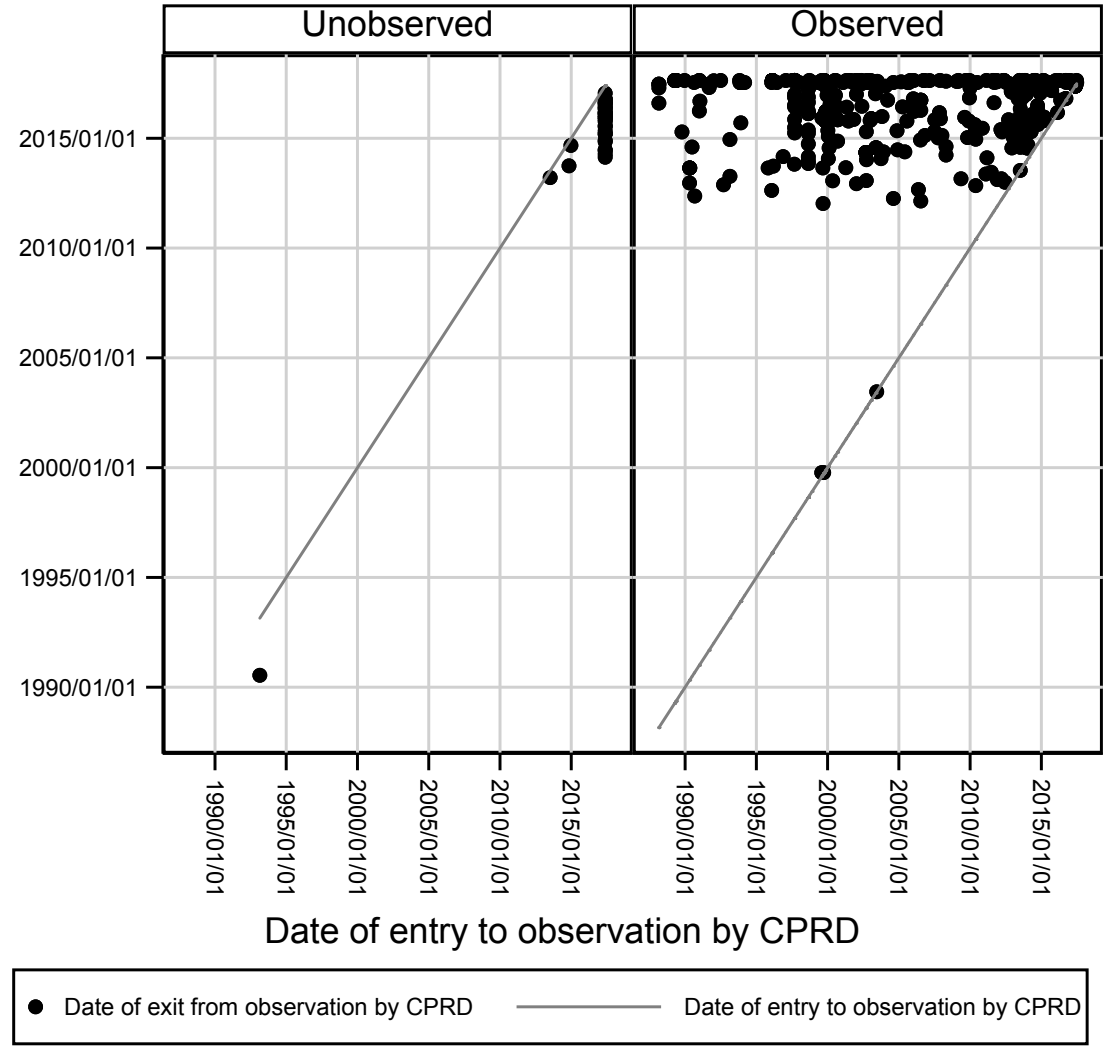
Plot of exit date against entry date for unobserved and observed patients

- And here are plots of exit date (on the vertical axis) against entry date (on the horizontal axis), with diagonal **equality lines**.
- The two graphs show patients with empty and non–empty observation windows, respectively.
- The "unobserved" patients, with pre–entry exit dates and empty observation windows, should be excluded from observational analyses.



Plot of exit date against entry date for unobserved and observed patients

- And here are plots of exit date (on the vertical axis) against entry date (on the horizontal axis), with diagonal **equality lines**.
- The two graphs show patients with empty and non–empty observation windows, respectively.
- The "unobserved" patients, with pre–entry exit dates and empty observation windows, should be excluded from observational analyses.



Distribution of entry status in observed patients

Focussing on patients with a non–empty observation window, we now `tabulate` the distribution of entry status in these patients.

```
. tab entrystat if observed==1, miss;

        Entry status to |
    observation by CPRD |      Freq.     Percent        Cum.
------------------------+-----------------------------------
         First UTS date |        233       56.97       56.97
Patient joined practice |        176       43.03      100.00
------------------------+-----------------------------------
                  Total |        409      100.00
```

As CPRD is a growing concern and continues to recruit practices, we are not surprised to find that more patients enter observation when their practices become up–to–standard (UTS) in their data collection than when they register with an existing up–to–standard practice.

Distribution of exit status in observed patients

And now, we `tabulate` the distribution of exit status in the same observed patients.

```
. tab exitstat if observed==1, miss;

     Exit status from |
  observation by CPRD |      Freq.     Percent        Cum.
----------------------+-----------------------------------
 Last collection date |        283       69.19       69.19
Patient left practice |         61       14.91       84.11
         Patient died |         65       15.89      100.00
----------------------+-----------------------------------
                Total |        409      100.00
```

Unsurprisingly, most observed patients end their observation windows at the most recent date on which their practices sent a data consignment to CPRD. Only a minority leave their practices or die.

The master do–file `create.do` (revisited)

We have seen that this master do–file calls 3 servant do–files to create the lookup do–file and datasets, the practice dataset, and the patient dataset, respectively.

```
#delim ;
version 13.1;
*
 Create a minimal CPRD database
*;

dolog lookups;
dolog practice;
dolog patient;

exit;
```

This database is, of course, *very* minimal. A real–world master do–file would call many more servant do–files, using `cprdutil` modules to create primary datasets for clinical events and/or referrals and/or tests and/or prescriptions. Other servant do–files might create **secondary datasets**. *For instance*, these datasets might contain one observation per patient per year of observation, and data on hospitalization counts.

Summary: Concluding remarks and tips

- The `cprdutil` package uses the SSC packages `keyby`, `addinby`, `lablist`, `chardef`, and `intext`, which need to be installed in order for `cprdutil` to work.
- It may be a good idea to install *all* my packages, using one of the `instasisay_`*X* do-files, downloadable from my website, to install the latest versions compatible with the user's Stata Version *X*. (In Stata, type `findit instasisay`.)
- Hospitalization data can be merged into a `cprdutil` database, using the satellite SSC packages `cprdlinkutil` and `cprdhesutil`.
- And lists of interesting medical codes and product codes can be created using the **CPRD browser**, which creates output text datafiles, which can also be input into Stata, using the `cprdutil` modules `cprd_browser_medical` and `cprd_browser_product`.

Summary: Concluding remarks and tips

- The `cprdutil` package uses the SSC packages `keyby`, `addinby`, `lablist`, `chardef`, and `intext`, which need to be installed in order for `cprdutil` to work.
- It may be a good idea to install *all* my packages, using one of the `instasisay_`*X* do-files, downloadable from my website, to install the latest versions compatible with the user's Stata Version *X*. (In Stata, type `findit instasisay`.)
- Hospitalization data can be merged into a `cprdutil` database, using the satellite SSC packages `cprdlinkutil` and `cprdhesutil`.
- And lists of interesting medical codes and product codes can be created using the **CPRD browser**, which creates output text datafiles, which can also be input into Stata, using the `cprdutil` modules `cprd_browser_medical` and `cprd_browser_product`.

Summary: Concluding remarks and tips

- The `cprdutil` package uses the SSC packages `keyby`, `addinby`, `lablist`, `chardef`, and `intext`, which need to be installed in order for `cprdutil` to work.
- It may be a good idea to install *all* my packages, using one of the `instasisay_`*X* do-files, downloadable from my website, to install the latest versions compatible with the user's Stata Version *X*. (In Stata, type `findit instasisay`.)
- Hospitalization data can be merged into a `cprdutil` database, using the satellite SSC packages `cprdlinkutil` and `cprdhesutil`.
- And lists of interesting medical codes and product codes can be created using the **CPRD browser**, which creates output text datafiles, which can also be input into Stata, using the `cprdutil` modules `cprd_browser_medical` and `cprd_browser_product`.

Summary: Concluding remarks and tips

- The `cprdutil` package uses the SSC packages `keyby`, `addinby`, `lablist`, `chardef`, and `intext`, which need to be installed in order for `cprdutil` to work.
- It may be a good idea to install *all* my packages, using one of the `instasisay_`*X* do-files, downloadable from my website, to install the latest versions compatible with the user's Stata Version *X*. (In Stata, type `findit instasisay`.)
- Hospitalization data can be merged into a `cprdutil` database, using the satellite SSC packages `cprdlinkutil` and `cprdhesutil`.
- And lists of interesting medical codes and product codes can be created using the **CPRD browser**, which creates output text datafiles, which can also be input into Stata, using the `cprdutil` modules `cprd_browser_medical` and `cprd_browser_product`.

Summary: Concluding remarks and tips

- The `cprdutil` package uses the SSC packages `keyby`, `addinby`, `lablist`, `chardef`, and `intext`, which need to be installed in order for `cprdutil` to work.
- It may be a good idea to install *all* my packages, using one of the `instasisay_`*X* do-files, downloadable from my website, to install the latest versions compatible with the user's Stata Version *X*. (In Stata, type `findit instasisay`.)
- Hospitalization data can be merged into a `cprdutil` database, using the satellite SSC packages `cprdlinkutil` and `cprdhesutil`.
- And lists of interesting medical codes and product codes can be created using the **CPRD browser**, which creates output text datafiles, which can also be input into Stata, using the `cprdutil` modules `cprd_browser_medical` and `cprd_browser_product`.

References

[1] Herrett E. M., Gallagher A. M., Bhaskaran K., Forbes H., Mathur R., van Staa T., and Smeeth L. 2015. Data Resource Profile: Clinical Practice Research Datalink (CPRD). *International Journal of Epidemiology* **44 (3)**: 827–836.

[2] Kousoulis, A., Rafi, I., and de Lusignan, S. 2015. The CPRD and the RCGP: building on research success by enhancing benefits for patients and practices. *British Journal of General Practice* **65(631)**: 54–55.

This presentation, and the do–files producing the examples, can be downloaded from the conference website at *http://ideas.repec.org/s/boc/usug18.html*

The packages used in this presentation can be downloaded from SSC, using the `ssc` command.