Pattern matching in Stata

Chasing the devil in the details

Mael Astruc–Le Souder

Bordeaux School of Economics, France

2024 Stata UK Conference

Motivation

In research, most of the time is spent preparing the data.

In Stata, most of this can be summarized with two statements:

```
generate x = ... if ...
replace  x = ... if ...
```

Find the bug

Example: You have a variable *like* with the answers for the question:

Do you like Stata ?

	1	2	3	4	5
(Absolutely not)				(Totally)	

You want to summarize it with a new variable *opinion*

```
generate opinion = "negative" if like <= 2
replace  opinion = "neutral"  if like == 3
replace  opinion = "positive" if like >= 4
```

Proposition

The *pmatch* command

- A syntax similar to *switch* / *match* statements in other languages
- Exhaustiveness and usefulness checks

```
pmatch opinion, variables(like) body(
    1~2 => "negative",
    3   => "neutral",
    4~5 => "positive"
)
    Warning : Missing cases
        .
```

Syntax overview

```
pmatch varname, Variables(varlist) Body(
    [pattern => exp,]
    [pattern => exp,]
    ...
) [nocheck]
```

Syntax comparison

```
generate varname = exp1 if conditions1 on varlist
replace  varname = exp2 if conditions2 on varlist
```

```
pmatch varname, variables(varlist) body(
    [pattern1 => exp1,]
    [pattern2 => exp2,]
)
```

Syntax: varname

varname: the name of the variable you want to modify.

```
generate varname = exp1 if conditions1 on varlist
replace  varname = exp2 if conditions2 on varlist
```

```
pmatch varname, variables(varlist) body(
    [pattern1 => exp1,]
    [pattern2 => exp2,]
)
```

Syntax: expressions

exp: the new values you want.

```
generate varname = exp1 if conditions1 on varlist
replace  varname = exp2 if conditions2 on varlist
```

```
pmatch varname, variables(varlist) body(
    [pattern1 => exp1,]
    [pattern2 => exp2,]
)
```

Syntax: conditions

conditions/patterns: the conditions for the replacements.

```
generate varname = exp1 if conditions1 on varlist
replace  varname = exp2 if conditions2 on varlist
```

```
pmatch varname, variables(varlist) body(
    [pattern1 => exp1,]
    [pattern2 => exp2,]
)
```

Syntax: varlist

varlist : the variables that determine the replacement.

```
generate varname = exp1 if conditions1 on varlist
replace  varname = exp2 if conditions2 on varlist
```

```
pmatch varname, variables(varlist) body(
    [pattern1 => exp1,]
    [pattern2 => exp2,]
)
```

Patterns

Pattern	**Syntax**	**Description**
Constant	x	A simple value, a number, or a string.
Range	$a \sim b$	A range between a and b.
Or	*pattern* | ... | *pattern*	The union of multiple patterns for a variable.
Wildcard	_	Any pattern that has not been matched yet.
Tuple	(*pattern*, ..., *pattern*)	The intersection of patterns for different variables.

Example 1: Constant pattern

```
gen var_1 = ""
replace var_1 = "very low"  if rep78 == 1
replace var_1 = "low"       if rep78 == 2
replace var_1 = "mid"       if rep78 == 3
replace var_1 = "high"      if rep78 == 4
replace var_1 = "very high" if rep78 == 5
replace var_1 = "missing"   if rep78 == .
```

```
pmatch var_2, variables(rep78) body( ///
    1 => "very low",                 ///
    2 => "low",                      ///
    3 => "mid",                      ///
    4 => "high",                     ///
    5 => "very high",                ///
    . => "missing",                  ///
)
```

Example 6

Example 2: Range pattern

```
gen var_1 = ""
replace var_1 = "cheap"     if price >= 0    & price <  6000
replace var_1 = "normal"    if price >= 6000 & price <  9000
replace var_1 = "expensive" if price >= 9000 & price <= 16000
replace var_1 = "missing"   if price == .
```

```
pmatch var_2, variables(price) body( ///
    min~!6000   => "cheap",          ///
    6000~!9000  => "normal",         ///
    9000~max    => "expensive",      ///
    .           => "missing",        ///
)
```

Note: the ! excludes the boundary. $a \sim b$ includes a and b, $a \sim !b$ includes a but not b, $a! \sim b$ excludes a and includes b. $a!!b$ excludes both a and b.

Example 7

Example 3: Or pattern

```
gen var_1 = ""
replace var_1 = "low"     if rep78 == 1 | rep78 == 2
replace var_1 = "mid"     if rep78 == 3
replace var_1 = "high"    if rep78 == 4 | rep78 == 5
replace var_1 = "missing" if rep78 == .
```

```
pmatch var_2, variables(rep78) body( ///
    1 | 2  => "low",                 ///
    3      => "mid",                 ///
    4 | 5  => "high",                ///
    .      => "missing",             ///
)
```

Example 4: Wildcard pattern

```
gen var_1 = "other"
replace var_1 = "very low" if rep78 == 1
replace var_1 = "low"      if rep78 == 2
```

```
pmatch var_2, variables(rep78) body( ///
    1 => "very low",                 ///
    2 => "low",                      ///
    _ => "other",                    ///
)
```

Example 5: Tuple pattern

```
gen var_1 = ""
replace var_1 = "case 1"  if rep78 <  3 & price <  10000
replace var_1 = "case 2"  if rep78 <  3 & price >= 10000
replace var_1 = "case 3"  if rep78 >= 3
replace var_1 = "missing" if rep78 == . | price == .
```

```
pmatch var_2, variables(rep78 price) body( ///
    (min~!3, min~!10000) => "case 1",       ///
    (min~!3, 10000~max)  => "case 2",       ///
    (3~max,  _)          => "case 3",       ///
    (., _) | (_, .)      => "missing",      ///
)
```

Checks

Convenient syntax, but that's not the main benefit.

- Exhaustiveness
    - Did you forgot some cases ?
- Usefulness
    - Are all the conditions useful ?
    - Are there some overlaps between them ?

No time for the algorithm, straight to the results.

Example 6: Exhaustiveness

```
gen var_1 = ""
replace var_1 = "very low"  if rep78 == 1
replace var_1 = "low"       if rep78 == 2
replace var_1 = "mid"       if rep78 == 3
replace var_1 = "high"      if rep78 == 4
replace var_1 = "very high" if rep78 == 5
```

```
pmatch var_2, variables(rep78) body( ///
    1 => "very low",                 ///
    2 => "low",                      ///
    3 => "mid",                      ///
    4 => "high",                     ///
    5 => "very high",                ///
)
    Warning : Missing values
          .
```

Example 1

Example 7: Overlaps

```
gen var_1 = ""
replace var_1 = "cheap"     if price >= 0    & price <= 6000
replace var_1 = "normal"    if price >= 6000 & price <= 9000
replace var_1 = "expensive" if price >= 9000 & price <= 16000
replace var_1 = "missing"   if price == .
```

```
pmatch var_2, variables(price) body( ///
    min~6000  => "cheap",            ///
    6000~9000 => "normal",           ///
    9000~max  => "expensive",        ///
    .         => "missing",          ///
)
    Warning :  Arm 2 has overlaps
        Arm 1:  6000
    Warning :  Arm 3 has overlaps
        Arm 2:  9000
```

Example 2

Example 8: Usefulness

```
gen var_1 = ""
replace var_1 = "cheap"     if price >= 0    & price <  6000
replace var_1 = "normal"    if price >= 6000 & price <  9000
replace var_1 = "expensive" if price >= 9000 & price <= 16000
replace var_1 = "missing"   if price == .
```

```
pmatch var_2, variables(price) body( ///
    min~!6000  => "cheap",           ///
    6000~!9000 => "normal",          ///
    9000~max   => "expensive",       ///
    min~max    => "oops",            ///
    .          => "missing",         ///
)
    Warning :  Arm 4 is not useful
    Warning :  Arm 4 has overlaps
        Arm 1:  3291~5999
        Arm 2:  6000~8999
        Arm 3:  9000~15906
```

Limitations

What does it cost compare to ‘replace . . . if . . . ‘ statements ?

- It depends on your data
- The command has 4 steps
  - Checking the variables
  - Parsing the body
  - Checking the conditions
  - Evaluating each arm
- $<$ 1M observations, it’s less than 0.1s
- $\geq$ 1M observations, checking levels becomes costly

Next steps

Supports byte, integer, long, float, double, and strings

- Already supports using label values instead of values Example 9
- Plan to add support for dates
- Plan to add *missing* and *nonmissing* patterns
- Plan to add examples in the warnings
- Plan to add possibility to ignore impossible cases with tuples

Conclusion

This project is still young, this is my first time presenting it

- Tell me if you find it interesting, or what you think are the issues
- Comments on the syntax, features, or anything else are welcomed

You can find the project and the installation command on GitHub

`https://github.com/MaelAstruc/stata_match`

You can contact me by email

`mael.astruc-le-souder@u-bordeaux.fr`

Thank you for your attention!

Example 9: Label values

```
drop _all
set obs 100
gen int color = runiform(1, 4)
label define color_label 1 "Red" 2 "Green" 3 "Blue"
label values color color_label
```

```
pmatch color_hex, variables(color) body( ///
    1      => "#FF0000",                 ///
    2      => "#00FF00",                 ///
   "Blue"  => "#0000FF",                 ///
)
```

Next steps