Title

pmatch —— Pattern matching

Syntax

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

varname is the name of the variable (A) you would like to replace. If the variable does not exist, it will be created.

Variables(*varlist*) contains the list of variables (B) you want to match on.

Body(...) contains the list of replacements you would like to do. It's composed of multiple arms. Each arm includes a *pattern* on the left hand side indicating the conditions of the replacement based on the values of the variables (B). It also contains an expression on the right hand side to replace the values of your variable (A). They are separated by an arrow **=>**.

nocheck skips the checks and directly performs the replacements. This allows to use the syntax of the command, without the performance cost of the verifications.

Description

The **pmatch** command provides an alternative syntax to series of '**replace ... if ...**' statements. It limits repetitions and might feel familiar for users coming from other programming languages with pattern matching.

Beyond the new syntax, the **pmatch** command provides run−time checks for the exhaustiveness and the usefulness of the conditions provided. The exhaustiveness check means that the command will tell you if some levels are not covered and which ones are missing. The usefulness check means that the command will tell you if the conditions you specified in each arm are useful, or if some of them overlap with a previous ones.

The command is inspired by the Rust Programming Language pattern syntax and algorithm.

The different examples illustrate how to use the different patterns detailed in the next section and what kind of information the checks provide.

Patterns

Pattern	Description
Constant *x*	A unique value, either a number or a string.
Range *a~b*	A range from *a* to *b*, with *a* and *b* two numbers. The symbol **~** indicates that both values are included. You can use **!~** to exclude the min, **~!** to exclude the max or **!!** to exclude both. You can use *min* and *max* to refer to the minimum and maximum values of your variable.
Or *pattern* | *...* | *pattern*	A pattern to compose with multiple patterns for a variable.
Wildcard _	A pattern to cover all the possibilities that are not covered by the previous arms.
Tuple (*pattern*, *...*, *pattern*)	A pattern ro use when multiple variables are provided for the matching. Each pattern matches with the corresponding variable.

Note: If a variable is encoded or if you defined label values, you can use these labels to refer to the corresponding value.

Examples

Example 1: Constant patterns

In this example, we use the values of the variable **rep78** to create a new variables using the normal way (**var_1**) and with the **pmatch** command (**var_2**), using Constant patterns with the '**x**' syntax.

```
sysuse auto, clear

* Usual way

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 == .

* With the pmatch command

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

assert var_1 == var_2
```

Example 2: Range patterns

The Constant pattern is simple but not practical once we have many values or decimals. In such cases we can us the Range pattern with the '**a~b**' syntax.

```
sysuse auto, clear

* Usual way

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 == .

* With the pmatch command

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

assert var_1 == var_2
```

Example 3: Or patterns

The Or pattern is used to combine multiple patterns with the '*pattern* **|** ... **|** *pattern*' syntax.

```
sysuse auto, clear

* Usual way

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 == .

* With the pmatch command

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

assert var_1 == var_2
```

Example 4: Wildcard patterns

To define a default value, we can use the wildcard pattern '_'. It covers all the values not included in the previous arms. This means that any value included after a wildcard is ignored.

```
sysuse auto, clear

* Usual way

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

* With the pmatch command

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

assert var_1 == var_2
```

Example 5: Tuple patterns

To pmatch on multiple variables at the same time, we can use the Tuple pattern with the '**(***pattern*, ..., *pattern***)**' syntax.

```
sysuse auto, clear

* Usual way

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 == .

* With the pmatch command

gen var_2 = ""
pmatch var_2, variables(rep78 price) body(  ///
    (min~!3, min~!10000)   => "case 1",      ///
```

```
    (min~!3, 10000~max)    => "case 2",      ///
    (3~max, _)             => "case 3",      ///
    (., _) | (_, .)        => "missing",     ///
)

assert var_1 == var_2
```

Example 6: Exhaustiveness

Coming back to Example 1, if we forgot to include the case where **rep_78** is missing, the command will print a warning.

```
sysuse auto, clear

* Usual way

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

* With the pmatch command

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

// Warning : Missing values
//     .

assert var_1 == var_2
```

Including a Wildcard pattern covers all the remaining cases by default. This should be used with caution, because you might cover some unexpected cases such as missing values.

Example 7: Usefulness

On the other hand, with Example 2, we can also do mistakes with the ranges and cover some cases multiple times.

```
sysuse auto, clear

* Usual way

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 == .

* With the pmatch command

gen var_2 = ""
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

assert var_1 == var_2
```

References

MARANGET L. Warnings for Pattern Matching Journal of Functional Programming. 2007;17(3):387?421. doi:10.1017/S0956796807006223

Package details

Version : **pmatch** version 0.0.1
Source : GitHub

Author : Mael Astruc--Le Souder
E-mail : mael.astruc-le-souder@u-bordeaux.fr

Feedback

Please submit bugs, errors, feature requests on GitHub by opening a new issue, or by sending me an email.

Citation guidelines

Suggested citation for this package:

Astruc--Le Souder, M. (2024). Stata package "pmatch" version 0.0.1. https://github.com/MaelAstruc/stata_match.

```
@software{pmatch,
   author = {Astruc--Le Souder Mael},
   title = {Stata package ‘‘pmatch’’},
   url = {https://github.com/MaelAstruc/stata_match},
   version = {0.0.1},
   date = {2024-08-22}
}
```