
ydot
Release 0.0.6

Jee Vang, Ph.D.

Dec 10, 2020

CONTENTS

1 Quickstart	3
1.1 Basic	3
1.2 More	4
2 Bibliography	9
3 PySpark Formula	11
3.1 Formula	11
3.2 Spark	13
4 Indices and tables	15
5 About	17
6 Copyright	19
6.1 Documentation	19
6.2 Software	19
6.3 Art	19
7 Citation	21
8 Author	23
9 Help	25
Bibliography	27
Python Module Index	29
Index	31



ydot is a Python API to produce PySpark dataframe models from R-like formula expressions. This project is based on [patsy \[pat\]](#). As a quickstart, let's say you have a Spark dataframe with data as follows.

Table 1: Dummy Data in a Spark Dataframe

a	b	x1	x2	y
left	low	19.945536387662504	3.85214120038979	0.0
left	low	20.674308066353493	4.098585619118175	1.0
right	high	20.346647025958433	2.7107604387194626	1.0
right	mid	18.699653829045985	5.2111542692543065	1.0
left	low	21.51851187887476	2.432390426907621	1.0
right	mid	20.989823705535017	3.6774523253171734	1.0
right	high	20.277680897136328	2.4873300559969604	0.0
right	mid	19.551410645704927	2.3549674965407372	0.0
right	low	20.96196624352397	3.1665930443154995	0.0
right	mid	19.172421360793678	3.562224297579924	1.0

Now, let's say you want to model this dataset as follows.

- $y \sim x_1 + x_2 + a + b$

Then all you have to do is use the `smatrices()` function.

```
1 from ydot.spark import smatrices
2
3 formula = 'y ~ x1 + x2 + a + b'
4 y, X = smatrices(formula, sdf)
```

Observe that `y` and `X` will be Spark dataframes as specified by the formula. Here's a more interesting example where you want a model specified up to all two-way interactions.

- $y \sim (x1 + x2 + a + b)**2$

Then you could issue the code as below.

```
1 from ydot.spark import smatrices
2
3 formula = 'y ~ (x1 + x2 + a + b)**2'
4 y, X = smatrices(formula, sdf)
```

Your resulting `X` Spark dataframe will look like the following.

Table 2: Dummy Data Transformed by Formula

Intercept	a	T	right	T	low	T	mid	a	T	right	T	low	b	T	mid	x2:a	T	right	T	low	b	T	mid	x2
1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	19.9455	0.0	0.0	0.0	0.0	19.9455	0.0	0.0	0.0	19.9455	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	20.6743	0.0	0.0	0.0	0.0	20.6743	0.0	0.0	0.0	20.6743	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	20.3466	1.0	0.0	0.0	0.0	20.3466	1.0	0.0	0.0	20.3466	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	18.6996	1.0	0.0	1.0	0.0	18.6996	1.0	0.0	1.0	18.6996	1.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	21.5185	0.0	1.0	0.0	0.0	21.5185	0.0	1.0	0.0	21.5185	0.0	1.0	0.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	20.9898	1.0	0.0	1.0	0.0	20.9898	1.0	0.0	1.0	20.9898	1.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	20.2776	1.0	0.0	0.0	0.0	20.2776	1.0	0.0	0.0	20.2776	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	19.5514	1.0	0.0	1.0	0.0	19.5514	1.0	0.0	1.0	19.5514	1.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	20.9619	1.0	1.0	0.0	0.0	20.9619	1.0	1.0	0.0	20.9619	1.0	1.0	0.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	19.1724	1.0	0.0	1.0	0.0	19.1724	1.0	0.0	1.0	19.1724	1.0	0.0	1.0	0.0	0.0	0.0	0.0

In general, what you get with `patsy` is what you get with `ydot`, however, there are exceptions. For example, the builtin functions such as `standardize()` and `center()` available with `patsy` will not work against Spark dataframes. Additionally, `patsy` allows for custom transforms, but such transforms (or user defined functions) must be visible. For now, only numpy-based transforms are allowed against continuous variables (or numeric columns).

QUICKSTART

1.1 Basic

The best way to learn R-style formula syntax with `ydot` is to head on over to [patsy \[pat\]](#) and read the documentation. Below, we show very simple code to transform a Spark dataframe into two design matrices (these are also Spark dataframes), `y` and `X`, using a formula that defines a model up to two-way interactions.

```
1 import random
2 from random import choice
3
4 import numpy as np
5 import pandas as pd
6 from pyspark.sql import SparkSession
7
8 from ydot.spark import smatrices
9
10 random.seed(37)
11 np.random.seed(37)
12
13
14 def get_spark_dataframe(spark):
15     n = 100
16     data = {
17         'a': [choice(['left', 'right']) for _ in range(n)],
18         'b': [choice(['high', 'mid', 'low']) for _ in range(n)],
19         'x1': np.random.normal(20, 1, n),
20         'x2': np.random.normal(3, 1, n),
21         'y': [choice([1.0, 0.0]) for _ in range(n)]
22     }
23     pdf = pd.DataFrame(data)
24
25     sdf = spark.createDataFrame(pdf)
26     return sdf
27
28
29 if __name__ == '__main__':
30     try:
31         spark = (SparkSession.builder
32                 .master('local[4]')
33                 .appName('local-testing-pyspark')
34                 .getOrCreate())
35         sdf = get_spark_dataframe(spark)
36
37         y, X = smatrices('y ~ (x1 + x2 + a + b)**2', sdf)
```

(continues on next page)

(continued from previous page)

```

38     y = y.toPandas()
39     X = X.toPandas()
40
41     print(X.head(10))
42     X.head(10).to_csv('two-way-interactions.csv', index=False)
43 except Exception as e:
44     print(e)
45 finally:
46     try:
47         spark.stop()
48         print('closed spark')
49     except Exception as e:
50         print(e)

```

1.2 More

We use the code below to generate the models (data) below.

```

1  import random
2  from random import choice
3
4  import numpy as np
5  import pandas as pd
6  from pyspark.sql import SparkSession
7
8  from ydot.spark import smatrices
9
10 random.seed(37)
11 np.random.seed(37)
12
13
14 def get_spark_dataframe(spark):
15     n = 100
16     data = {
17         'a': [choice(['left', 'right']) for _ in range(n)],
18         'b': [choice(['high', 'mid', 'low']) for _ in range(n)],
19         'x1': np.random.normal(20, 1, n),
20         'x2': np.random.normal(3, 1, n),
21         'y': [choice([1.0, 0.0]) for _ in range(n)]
22     }
23     pdf = pd.DataFrame(data)
24
25     sdf = spark.createDataFrame(pdf)
26     return sdf
27
28
29 if __name__ == '__main__':
30     try:
31         spark = (SparkSession.builder
32                 .master('local[4]')
33                 .appName('local-testing-pyspark')
34                 .getOrCreate())
35         sdf = get_spark_dataframe(spark)
36

```

(continues on next page)

(continued from previous page)

```

37     formulas = [
38         {
39             'f': 'y ~ np.sin(x1) + np.cos(x2) + a + b',
40             'o': 'transformed-continuous.csv'
41         },
42         {
43             'f': 'y ~ x1*x2',
44             'o': 'star-con-interaction.csv'
45         },
46         {
47             'f': 'y ~ a*b',
48             'o': 'star-cat-interaction.csv'
49         },
50         {
51             'f': 'y ~ x1:x2',
52             'o': 'colon-con-interaction.csv'
53         },
54         {
55             'f': 'y ~ a:b',
56             'o': 'colon-cat-interaction.csv'
57         },
58         {
59             'f': 'y ~ (x1 + x2) / (a + b)',
60             'o': 'divide-interaction.csv'
61         },
62         {
63             'f': 'y ~ x1 + x2 + a - 1',
64             'o': 'no-intercept.csv'
65         }
66     ]
67
68     for item in formulas:
69         f = item['f']
70         o = item['o']
71
72         y, X = smatrices(f, sdf)
73         y = y.toPandas()
74         X = X.toPandas()
75
76         X.head(5).to_csv(o, index=False)
77
78         s = f"""
79         .. csv-table:: {f}
80            :file: _code/{o}
81            :header-rows: 1
82         """
83         print(s.strip())
84     except Exception as e:
85         print(e)
86     finally:
87         try:
88             spark.stop()
89             print('closed spark')
90         except Exception as e:
91             print(e)

```

You can use numpy functions against continuous variables.

Table 1: $y \sim np.\sin(x1) + np.\cos(x2) + a + b$

Intercept	a[T.right]	b[T.low]	b[T.mid]	np.sin(x1)	np.cos(x2)
1.0	0.0	1.0	0.0	0.8893769205406579	-0.758004200582313
1.0	0.0	1.0	0.0	0.9679261582216445	-0.5759807266894401
1.0	1.0	0.0	0.0	0.9972849995254774	-0.9086185088676886
1.0	1.0	0.0	1.0	-0.14934132364604816	0.4783416124776783
1.0	0.0	1.0	0.0	0.45523550315103734	-0.7588816501987654

The * specifies interactions and keeps lower order terms.

Table 2: $y \sim x1*x2$

Intercept	x1	x2	x1:x2
1.0	19.945536387662504	3.85214120038979	76.83302248278848
1.0	20.674308066353493	4.098585619118175	84.73542172597531
1.0	20.346647025958433	2.7107604387194626	55.154885818557126
1.0	18.699653829045985	5.2111542692543065	97.44678088481062
1.0	21.51851187887476	2.432390426907621	52.341422295472896

Table 3: $y \sim a*b$

Intercept	a[T.right]	b[T.low]	b[T.mid]	a[T.right]:b[T.low]	a[T.right]:b[T.mid]
1.0	0.0	1.0	0.0	0.0	0.0
1.0	0.0	1.0	0.0	0.0	0.0
1.0	1.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	1.0	0.0	1.0
1.0	0.0	1.0	0.0	0.0	0.0

The : specifies interactions and drops lower order terms.

Table 4: $y \sim x1:x2$

Intercept	x1:x2
1.0	76.83302248278848
1.0	84.73542172597531
1.0	55.154885818557126
1.0	97.44678088481062
1.0	52.341422295472896

Table 5: $y \sim a:b$

Intercept	b[T.low]	b[T.mid]	a[T.right]:b[high]	a[T.right]:b[low]	a[T.right]:b[mid]
1.0	1.0	0.0	0.0	0.0	0.0
1.0	1.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	1.0	0.0	0.0
1.0	0.0	1.0	0.0	0.0	1.0
1.0	1.0	0.0	0.0	0.0	0.0

The / is **quirky** according to the patsy documentation, but it is shorthand for $a / b = a + a:b$.

Table 6: $y \sim (x1 + x2) / (a + b)$

Intercept	x1	x2	x1:x2:a[left]	x1:x2:a[right]	x1:x2:b[T.low]	x1:x2:b[T.mid]
1.0	19.945536387662504	3.85214120038979	76.83302248278803	76.83302248278803	76.83302248278803	76.83302248278803
1.0	20.674308066353493	4.098585619118175	84.73542172597501	84.73542172597501	84.73542172597501	84.73542172597501
1.0	20.346647025958433	2.7107604387194626	55.154885818557026	55.154885818557026	55.154885818557026	0.0
1.0	18.699653829045985	5.2111542692543065	97.44678088481062	97.44678088481062	97.44678088481062	97.44678088481062
1.0	21.51851187887476	2.432390426907621	52.341422295472096	52.341422295472096	52.341422295472096	52.341422295472096

If you need to drop the Intercept, add - 1 at the end. Note that one of the dummy variables for a is not dropped. This could be a bug with patsy.

Table 7: $y \sim x1 + x2 + a - 1$

a[left]	a[right]	x1	x2
1.0	0.0	19.945536387662504	3.85214120038979
1.0	0.0	20.674308066353493	4.098585619118175
0.0	1.0	20.346647025958433	2.7107604387194626
0.0	1.0	18.699653829045985	5.2111542692543065
1.0	0.0	21.51851187887476	2.432390426907621

BIBLIOGRAPHY

PYSPARK FORMULA

3.1 Formula

The `formula` module contains code to extract values from a record (e.g. a Spark dataframe Record) based on the model definition.

class `ydot.formula.CatExtractor` (*record, term*)

Bases: `ydot.formula.Extractor`

Categorical extractor (no levels).

`__init__` (*record, term*)

ctor.

Parameters

- **record** – Dictionary.
- **term** – Model term.

Returns None.

property value

Gets the extracted value.

class `ydot.formula.ConExtractor` (*record, term*)

Bases: `ydot.formula.Extractor`

Continuous extractor (no functions).

`__init__` (*record, term*)

ctor.

Parameters

- **record** – Dictionary.
- **term** – Model term.

Returns None.

property value

Gets the extracted value.

class `ydot.formula.Extractor` (*record, term, term_type*)

Bases: `abc.ABC`

Extractor to get value based on model term.

`__init__` (*record, term, term_type*)

ctor.

Param Dictionary.

Term Model term.

Term_type Type of term.

Returns None

abstract property value

Gets the extracted value.

class `ydot.formula.FunExtractor` (*record, term*)

Bases: `ydot.formula.Extractor`

Continuous extractor (with functions defined).

__init__ (*record, term*)

ctor.

Parameters

- **record** – Dictionary.
- **term** – Model term.

Returns None.

property value

Gets the extracted value.

class `ydot.formula.IntExtractor` (*record, term*)

Bases: `ydot.formula.Extractor`

Intercept extractor. Always returns 1.0.

__init__ (*record, term*)

ctor.

Parameters

- **record** – Dictionary.
- **term** – Model term.

Returns None.

property value

Gets the extracted value.

class `ydot.formula.InteractionExtractor` (*record, terms*)

Bases: `object`

Interaction extractor for interaction effects.

__init__ (*record, terms*)

ctor.

Parameters

- **record** – Dictionary.
- **terms** – Model term (possibly with interaction effects).

Returns None.

property value

class `ydot.formula.LvlExtractor` (*record*, *term*)

Bases: `ydot.formula.Extractor`

Categorical extractor (with levels).

__init__ (*record*, *term*)
ctor.

Parameters

- **record** – Dictionary.
- **term** – Model term.

Returns None.

property value

Gets the extracted value.

class `ydot.formula.TermEnum` (*value*)

Bases: `enum.IntEnum`

Term types.

- CAT: categorical without levels specified
- LVL: categorical with levels specified
- CON: continuous
- FUN: continuous with function transformations
- INT: intercept

CAT = 1

CON = 3

FUN = 4

INT = 5

LVL = 2

static **get_extractor** (*record*, *term*)

Gets the associated extractor based on the specified term.

Parameters

- **record** – Dictionary.
- **term** – Model term.

Returns Extractor.

3.2 Spark

The `spark` module contains code to transform a Spark dataframe into design matrices as specified by a formula.

`ydot.spark.get_columns` (*formula*, *sdf*, *profile=None*)

Gets the expanded columns of the specified Spark dataframe using the specified formula.

Parameters

- **formula** – Formula (R-like, based on patsy).

- **sdf** – Spark dataframe.
- **profile** – Profile. Default is *None* and profile will be determined empirically.

Returns Tuple of columns for y, X.

`ydot.spark.get_profile(sdf)`

Gets the field profiles of the specified Spark dataframe.

Parameters **sdf** – Spark dataframe.

Returns Dictionary.

`ydot.spark.smatrices(formula, sdf, profile=None)`

Gets tuple of design/model matrices.

Parameters

- **formula** – Formula.
- **sdf** – Spark dataframe.
- **profile** – Dictionary of data profile.

Returns y, X Spark dataframes.

INDICES AND TABLES

- genindex
- modindex
- search

ABOUT



One-Off Coder is an educational, service and product company. Please visit us online to discover how we may help you achieve life-long success in your personal coding career or with your company's business goals and objectives.

-
-
-
-
-
-

COPYRIGHT

6.1 Documentation

6.2 Software

Copyright 2020 One-Off Coder

Licensed under the Apache License, Version 2.0 (the "License");
you may **not** use this file **except in** compliance **with** the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law **or** agreed to **in** writing, software
distributed under the License **is** distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express **or** implied.
See the License **for** the specific language governing permissions **and**
limitations under the License.

6.3 Art

Copyright 2020 Daytchia Vang

CITATION

```
@misc{oneoffcoder_ydot_2020,  
title={ydot, R-like formulas for Spark Dataframes},  
url={https://github.com/oneoffcoder/pyspark-formula},  
author={Jee Vang},  
year={2020},  
month={Dec}}
```

**CHAPTER
EIGHT**

AUTHOR

Jee Vang, Ph.D.

-
-

HELP

-
-

BIBLIOGRAPHY

[pat] patsy. Patsy - describing statistical models in python. URL: <https://patsy.readthedocs.io/en/latest/index.html>.

PYTHON MODULE INDEX

y

`ydot.formula`, 11

`ydot.spark`, 13

Symbols

[__init__\(\)](#) (*ydot.formula.CatExtractor* method), 11
[__init__\(\)](#) (*ydot.formula.ConExtractor* method), 11
[__init__\(\)](#) (*ydot.formula.Extractor* method), 11
[__init__\(\)](#) (*ydot.formula.FunExtractor* method), 12
[__init__\(\)](#) (*ydot.formula.IntExtractor* method), 12
[__init__\(\)](#) (*ydot.formula.InteractionExtractor* method), 12
[__init__\(\)](#) (*ydot.formula.LvlExtractor* method), 13

C

[CAT](#) (*ydot.formula.TermEnum* attribute), 13
[CatExtractor](#) (*class in ydot.formula*), 11
[CON](#) (*ydot.formula.TermEnum* attribute), 13
[ConExtractor](#) (*class in ydot.formula*), 11

E

[Extractor](#) (*class in ydot.formula*), 11

F

[FUN](#) (*ydot.formula.TermEnum* attribute), 13
[FunExtractor](#) (*class in ydot.formula*), 12

G

[get_columns\(\)](#) (*in module ydot.spark*), 13
[get_extractor\(\)](#) (*ydot.formula.TermEnum* static method), 13
[get_profile\(\)](#) (*in module ydot.spark*), 14

I

[INT](#) (*ydot.formula.TermEnum* attribute), 13
[InteractionExtractor](#) (*class in ydot.formula*), 12
[IntExtractor](#) (*class in ydot.formula*), 12

L

[LVL](#) (*ydot.formula.TermEnum* attribute), 13
[LvlExtractor](#) (*class in ydot.formula*), 12

M

module
 [ydot.formula](#), 11

[ydot.spark](#), 13

S

[smatrices\(\)](#) (*in module ydot.spark*), 14

T

[TermEnum](#) (*class in ydot.formula*), 13

V

[value\(\)](#) (*ydot.formula.CatExtractor* property), 11
[value\(\)](#) (*ydot.formula.ConExtractor* property), 11
[value\(\)](#) (*ydot.formula.Extractor* property), 12
[value\(\)](#) (*ydot.formula.FunExtractor* property), 12
[value\(\)](#) (*ydot.formula.InteractionExtractor* property), 12
[value\(\)](#) (*ydot.formula.IntExtractor* property), 12
[value\(\)](#) (*ydot.formula.LvlExtractor* property), 13

Y

[ydot.formula](#)
 module, 11
[ydot.spark](#)
 module, 13