colassigner

Endre Márk Borza

Dec 06, 2023

CONTENTS:

fitting somewhat complex, nested data structures into tables, and removing the need to remember the name and construction logic of any column, if you can rely on static analysis

things to think about:

- draw a reliance dag based on calls
- pivot table: data content based columns
  - enum type support
- changing record entity type
- partial inheritance / composite types

CHAPTER

ONE

INSTALLATION:

1.1 using pip

```
pip install colassigner
```

CHAPTER

TWO

QUICKSTART

2.1 Assign Columns

```
import pandas as pd

from colassigner import ColAssigner
```

```
class Cols(ColAssigner):
    def col1(self, df):
        return df.iloc[:, 0] * 2

    def col2(self, df):
        return "added-another"
```

```
df = pd.DataFrame({"a": [1, 2, 3]}).pipe(Cols())
```

```
df
```

```
df.loc[:, Cols.col2]
```

```
0    added-another
1    added-another
2    added-another
Name: col_2, dtype: object
```

2.2 Access Columns

while also documenting datatypes

```
from colassigner import ColAccessor
```

```
class Cols(ColAccessor):

    x = int
    y = float
```

```
df = pd.DataFrame({Cols.x: [1, 2, 3], Cols.y: [0.3, 0.1, 0.9]})
```

```
df
```

```
df.loc[:, Cols.y]
```

```
0    0.3
1    0.1
2    0.9
Name: y, dtype: float64
```

CHAPTER

THREE

NESTED STRUCTURES

3.1 Nested Accessor

```
import pandas as pd
```

```
from colassigner import ColAccessor
```

```
class GrandChildCols(ColAccessor):
    x = str
    y = str


class ChildCols(ColAccessor):
    a = int
    b = float
    grandchild_a = GrandChildCols
    grandchild_b = GrandChildCols


class Cols(ColAccessor):

    fing = int
    assigned_child = ChildCols

    class InheritedChild(ChildCols):
        pass
```

```
pd.DataFrame(
    {
        Cols.fing: [2, 3, 4],
        Cols.assigned_child.grandchild_a.y: ["a", "b", "c"],
        Cols.InheritedChild.b: [0.1, 0.2, 0.3],
    }
)
```

3.2 Nested Assigner

```
from colassigner import ColAssigner
```

```
class SourceCols(ColAccessor):

    x = float
    b = bool

class SepChild(ColAssigner):
    _col = SourceCols.x

    def neg(self, df):
        return -df[self._col]

    def double(self, df):
        return 2 * df[self._col]

class Cols(ColAssigner):
    def col_one(self, df):
        return 1

    class SubCol(ColAssigner):
        def fing(self, df):
            return df.sum(axis=1)

        class SubSubCol(ColAssigner):
            _prefix = "pref_"

            def sub_x(self, df):
                return 0

            def sub_y(self, df):
                return self._prefix + df[Cols.col_one].astype(str)

        class SubSubCol2(SubSubCol):
            _prefix = "pref2_"

    sep_child = SepChild

    class SepChildB(SepChild):
        _col = SourceCols.b
```

```
df = pd.DataFrame({
    SourceCols.x: [1.5, 3.4, 9.1], SourceCols.b: [False, True, True]
}).pipe(Cols())
```

```
df.T
```

```
df.loc[:, [Cols.sep_child.double, Cols.SubCol.SubSubCol2.sub_x]]
```

3.3 Designated Child Assigner

These are designed for information sharing among assigners and **do not** take the dataframe as arguments for their methods but, take both the df and their parent assigner as parameters for their `__init__`

```
import numpy as np

from colassigner import ChildColAssigner
```

```
class RawCols(ColAccessor):

    cat = str
    num = int

class RawCols2(ColAccessor):
    b = str
    c = str

class IntSides(ChildColAssigner):

    # note the type and order of the parameters:
    def __init__(self, df, parent_assigner: "GbReindex") -> None:
        self.arr = parent_assigner.arr

    # note the absence of parameters
    def lower(self):
        return np.floor(self.arr).astype(int)

    def upper(self):
        return np.ceil(self.arr).astype(int)

class GbReindex(ChildColAssigner):
    main_col = ...

    def __init__(self, df, bc: "BaseCols"):
        # note that this reindex needs to be done only once
        # and can be used in many child assigners
        self.arr = bc.base_gb.reindex(df[self.main_col]).values

    def values(self):
        return self.arr

    sides = IntSides

class BaseCols(ColAssigner):
    def __init__(self, base_df):
        self.base_gb = base_df.groupby(RawCols.cat)[RawCols.num].mean()

    class GbB(GbReindex):
        main_col = RawCols2.b

    class GbC(GbReindex):
```

(continues on next page)

(continued from previous page)

```
        main_col = RawCols2.c

    def prod(self, df):
        return df.loc[
            :, [BaseCols.GbB.sides.lower, BaseCols.GbC.values]
        ].prod(axis=1)
```

```
df1 = pd.DataFrame({RawCols.cat: ["x", "y", "y"], RawCols.num: [2, 3, 4]})
```

```
assigner = BaseCols(df1)
```

```
df2 = pd.DataFrame({"b": ["x", "y", "x"], "c": ["y", "y", "x"]}).pipe(assigner)
```

```
df2
```

CHAPTER

FOUR

API

4.1 colassigner Package

Helper for assigning and accessing pandas columns

4.1.1 Functions

camel_to_snake(cc_str)	
experiment(assigner_inst, base_df, col_to_treat)	
get_all_cols(cls)	returns a list of strings of all columns given by the type
get_att_value(accessor, attname)	get the true assigned value for the class attribute
get_dag(cls)	generates a dag of the reliances of columns based on the ast of a colassigner
get_new_cols(cls)	
measure_effect(assigner, base_df, cause_col, ...)	

camel_to_snake

colassigner.**camel_to_snake**(*cc_str*)

experiment

colassigner.**experiment**(*assigner_inst*, *base_df*, *col_to_treat*, *treat_rate=0.2*, *seed=742*)

get_all_cols

colassigner.**get_all_cols**(*cls: ColMeta*)

returns a list of strings of all columns given by the type

can also be used for nested structues of columns

get_att_value

colassigner.**get_att_value**(*accessor: ColMeta*, *attname: str*)

get the true assigned value for the class attribute

get_dag

colassigner.**get_dag**(*cls: Type[*ColAssigner*]*)

generates a dag of the reliances of columns based on the ast of a colassigner

Parameters

cls : Type[ColAssigner]

Returns

list of edges of a dag

BETA! - WIP

rules: - explicitly access columns within other functions - no external function referencing column (only method of assigner class) - no common source in as attributes - self should always be named self

get_new_cols

colassigner.**get_new_cols**(*cls: ColMeta*)

measure_effect

colassigner.**measure_effect**(*assigner*, *base_df*, *cause_col*, *effect_col*, *treat_rate=0.2*, *seed=742*)

4.1.2 Classes

ChildColAssigner(df, parent_assigner)	assigner specifically for nested structures
Col()	
ColAccessor()	describe and access raw columns
ColAssigner()	define functions that create columns in a dataframe

ChildColAssigner

class colassigner.**ChildColAssigner**(*df: DataFrame*, *parent_assigner:* ColAssigner)

Bases: *ColAssigner*

assigner specifically for nested structures

methods of these are not called with parameters

the dataframe and the parent assigner are passed to the __init__ method as parameters

Col

class colassigner.**Col**

Bases: Generic[T]

ColAccessor

class colassigner.**ColAccessor**

Bases: object

describe and access raw columns

useful for - getting column names from static analysis - documenting types - dry describing nested structures

e. g.

```
>>> class LocationCols(ColAccessor):
...     lon = float
...     lat = float
```

```
>>> class TableCols(ColAccessor):
...     col1 = int
...     col2 = str
...     foreign_key1 = "name_of_key"
...
...     class NestedCols(ColAccessor):
...         s = str
...         x = float
...
...     start_loc = LocationCols
...     end_loc = LocationCols
```

```
>>> TableCols.start_loc.lat
'start_loc__lat'
```

ColAssigner

class colassigner.**ColAssigner**

Bases: *ColAccessor*

define functions that create columns in a dataframe

later the class attributes can be used to access the column can be used to created nested structures of columns

either by assigning or inheriting within:

class MyStaticChildAssigner(ColAssigner):

    pass

class MyAssigner(ColAssigner):

    class MySubAssigner(ColAssigner):
        pass

    chass1 = MyStaticChildAssigner

Methods Summary

__call__(df[, carried_prefixes])	Call self as a function.

Methods Documentation

__call__(*df: DataFrame*, *carried_prefixes=()*) → DataFrame

Call self as a function.

4.1.3 Class Inheritance Diagram

CHAPTER

FIVE

RELEASE NOTES

5.1 v0.0.0

- first release of colassigner, yay!!

5.2 v0.0.1

- first release of colassigner, yay!!

5.3 v0.0.2

- points of whats new

5.4 v0.0.3

- add calling graph feature

5.5 v0.0.4

add colaccessor

5.6 v0.0.5

extend allcols

5.7 v0.0.6

unify accessor and assigner

5.8 v0.1.0

change base structure to dry it a bit add col type retention

5.9 v0.2.0

- rethink nested structures
- do away with .assign(CA()) pattern
- replace with .pipe(CA())
- put aside graph generation for a while
- add proper docs

5.10 v0.2.1

multidigit number fix

5.11 v0.2.2

5.12 v0.3.0

5.13 v0.3.1

5.14 v0.4.0

cleanup

5.15 v0.4.1

minor extension and simplification

5.16 v0.4.2

getitem

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

C

E

G

M