

Pacote FDTH em Python

DEX083 – Probabilidade e Estatística.

Prof. José Cláudio Faria.

Discentes: Emyle Silva, Lucas Gabriel, Maria Clara Simões e Yuri Coutinho

2024.2



Contextualização

Pacote FDTH

Tabela de distribuição de Frequências, Histogramas e Polígonos (FDTH);

Ferramenta que possibilita a criação de tabelas de distribuição de frequências (TDF) e representações gráficas no ambiente R.

Objetivo

- Ampliar o suporte do R para análises baseadas em TDF, facilitando a análise exploratória de dados (AED).

Histórico de desenvolvimento

- Iniciado em 2003/2004;
- Na época, o R oferecia ferramentas de baixo nível, tornando as análises mais trabalhosas

Pacote FDTH

Principais características

- Redução univariada via TDF;
- TDFs simultâneas de dados multivariados (data.frame e matrizes);
- Diversas opções de apresentações gráficas das TDF;
- Aceita como entrada objetos das classes: vetores, matrizes e data frames.

Autores

- **José Cláudio Faria** -
Brasil/UDESC/DCET (Principal autor e mantenedor)
- **Enio Galinkin Jelihovschi** -
Brasil/UDESC/DCET (Co-autor)
- **Ivan Bezerra Allaman** -
Brasil/UDESC/DCET (Co-autor)

Pacote FDTH

Para utilizar o pacote, basta realizar a instalação e carregar a biblioteca ao iniciar o código;

```
> install.packages("fdth")
```

```
> library("fdth")
```

Também é possível acessar a implementação das funções através do repositório do pacote no github (versão de desenvolvimento):

[<https://github.com/jcfaria/fdth.git>](https://github.com/jcfaria/fdth.git)

Ou fazer o download através do link (versão estável):

[<https://cran.r-project.org/package=fdth>](https://cran.r-project.org/package=fdth)



Implementação em Python

Funções default

O pacote possui funções para vetores, matrizes e dataframes;

Inicialmente, foi priorizada a implementação das funções default do pacote.

- `summary_fdt_default`
- `summary_fdt_cat_default`
- `print_fdt_default`
- `print_fdt_cat_default`
- `fdt_default`
- `fdt_cat_default`
- `var_default`
- `sd_default`
- `plot_fdt_cat_default`
- Entre outras

Funções matrix e dataframe

Funções para entrada no formato de matrizes:

- `fdt_matrix`
- `fdt_cat_matrix`

Funções para entrada no formato de dataframes:

- `fdt_dataframe`
- `fdt_cat_dataframe`

Funções make

Funções que são utilizadas dentro de outras funções (como as default) para criar tabelas de distribuição de frequências (FDT) ou realizar formatações;

As funções implementadas podem ser acessadas no repositório do github:

[<https://github.com/yuriccosta/fdth-python.git>](https://github.com/yuriccosta/fdth-python.git)

- `make_fdt_simple;`
- `make_fdt_multiple;`
- `make_fdt_cat_simple;`
- `make_fdt_cat_multiple;`
- `make_fdt_format_classes;`

Estratégias Utilizadas no Desenvolvimento

- Padronização de declaração;
- Nomenclatura em snake_case;
- Preparação para Integração por meio de importação para garantir modularização;
- Documentação dos scripts ;
- Scripts de testes para as funções criadas;
- Entre outras.

Estratégias Utilizadas no Desenvolvimento

```
import numpy as np

def mean_fdt(x):
    # Definir intervalos de classe com base nos valores 'start', 'end' e 'h'
    breaks = np.arange(x['breaks']['start'], x['breaks']['end'] + x['breaks']['h'], x['breaks']['h'])

    # Calcular pontos médios dos intervalos de classe
    mids = 0.5 * (breaks[:-1] + breaks[1:])

    # Frequências das classes
    y = x['table'][:, 1]

    # Calcular a média ponderada dos pontos médios
    res = np.sum(y * mids) / np.sum(y)

    # Retornar a média
    return res
```

```
import numpy as np
from mean import mean_fdt
# Função sd_fdt
def sd_fdt(x):
    # Definir intervalos de classe com base nos valores 'start', 'end' e 'h'
    breaks = np.arange(x['breaks']['start'], x['breaks']['end'] + x['breaks']['h'], x['breaks']['h'])

    # Calcular pontos médios dos intervalos de classe
    mids = 0.5 * (breaks[:-1] + breaks[1:])

    # Frequências das classes
    y = x['table'][:, 1]

    # Calcular média usando a função mean_fdt
    mean_fdt_value = mean_fdt(x)

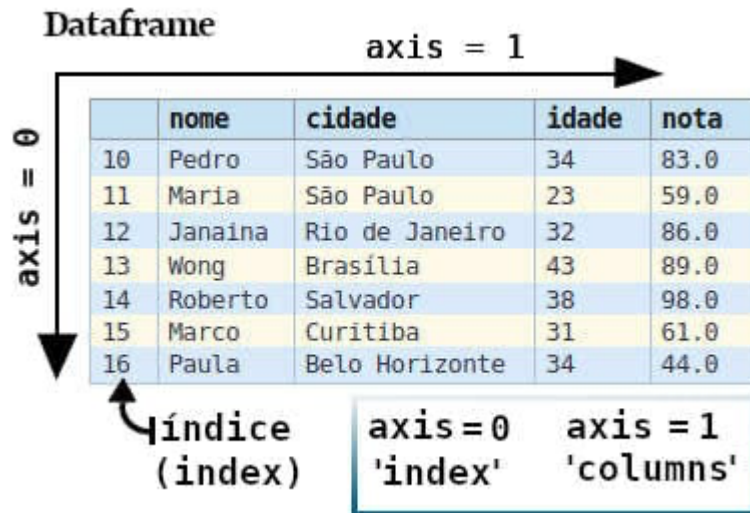
    # Calcular variância ponderada
    res = np.sum((mids - mean_fdt_value) ** 2 * y) / (np.sum(y) - 1)

    # Retornar o desvio padrão
    return np.sqrt(res)
```

Diferenças entre R e Python

Estrutura dos Dados

- **R:** Muitas funções no pacote FDTH em R aceitam e retornam `data.frames` nativos, que possuem suporte integrado a operações como indexação por nome de coluna.
- **Python:** A versão Python utiliza objetos do `pandas`, como `DataFrame`, que são equivalentes ao `data.frame` de R. Contudo, essas estruturas precisam ser importadas explicitamente e configuradas.



Diferenças entre R e Python

Bibliotecas e Dependências

- **R:** As funções FDTH geralmente dependem apenas das bibliotecas padrão do R.
- **Python:** Exige bibliotecas como **pandas** e **numpy** para replicar a funcionalidade do FDTH.

Observação: Pacote FDTH versão Python precisa adaptar certas funcionalidades (estruturas de dados, manipulação de categorias) que são automáticas no R.

Aspecto	R	Python
Vetores	Estrutura básica de dados, homogênea (todos os elementos devem ter o mesmo tipo).	Não há uma estrutura nativa equivalente; listas ou arrays do NumPy são usados.
Matrizes	Estruturas tabulares bidimensionais criadas com <code>matrix()</code>	Usam arrays bidimensionais do NumPy, amplamente usados em operações numéricas.
Data Frames	Estruturas bidimensionais com suporte nativo (<code>data.frame</code>). As colunas podem ter tipos de dados diferentes (numérico, categórico, caractere, etc.). Porém, dentro de uma coluna, todos os valores devem ser do mesmo tipo.	Representados por <code>pandas.DataFrame</code> , poderoso para manipulação
Indexação	Indexação começa em 1	Indexação começa em 0
Operações com Vetores	Suporta operações vetorizadas nativamente, como soma ou multiplicação direta entre vetores.	Usa NumPy para operações vetorizadas; listas nativas precisam de loops explícitos
Gráficos	Ferramentas nativas como <code>plot()</code> para gráficos avançados.	Usa bibliotecas como <code>matplotlib</code> , <code>seaborn</code> , e <code>plotly</code> para gráficos.

Função mfv_default

Python

```

1  from collections import Counter
2
3  def mfv_default(x, **kwargs):
4      # Se o conjunto de dados estiver vazio, retorne None
5      if len(x) == 0:
6          return None
7
8      # Conta a frequência de cada elemento no conjunto de dados
9      frequency = Counter(x)
10
11     # Encontra a frequência máxima
12     max_freq = max(frequency.values())
13
14     # Seleciona o primeiro elemento que tem a frequência máxima
15     mode = next(key for key, value in frequency.items() if value == max_freq)
16
17     return mode

```

R

[illegible]

Função print_fdt_cat_default

Python

```
print_fdt_cat_default.py > ...
1 import pandas as pd
2
3 def print_fdt_cat_default(x, columns=range(6), round=2, row_names=False, right=True,
4 # Concatenate line 0 with lines 1 to 5 (rounded)
5 res = pd.concat([x.iloc[:, [0]], x.iloc[:, 1:6].round(round)], axis=1)
6 # Filters the columns
7 res = res.iloc[:, columns]
8
9 # Name the columns
10 columns_names = columns_names = ['Category', 'f', 'rf', 'rf(%)', 'cf', 'cf(%)']
11 res.columns = [columns_names[i] for i in columns]
12
13 if row_names:
14     print(res, **kwargs)
15 else:
16     print(res.to_string(index=False), **kwargs)
17
```

R

```
mfv.default.R
1 print.fdt_cat.default <- function (x,
2 # columns=1:6,
3 # round=2,
4 # row.names=FALSE,
5 # right=TRUE, ...)
6 {
7     res <- cbind(x[, 1],
8                 round(x[, 2:6],
9                     round))[columns]
10
11     names(res) <- c('Category',
12                   'f',
13                   'rf',
14                   'rf(%)',
15                   'cf',
16                   'cf(%)')[columns]
17
18     print.data.frame(res,
19                     row.names=row.names,
20                     right=right, ...)
21 }
```

Função median_fdt

Python

```
median_fdt.py > ...
1 import numpy as np
2 import pandas as pd
3
4 def median_fdt(data):
5     # Extrai a tabela de frequência do objeto data
6     fdt = data['table']
7
8     # Número total de observações
9     n = fdt.iloc[-1, 4]
10
11     # Posição da classe mediana
12     posM = (n / 2 <= fdt.iloc[:, 4]).idxmax()
13
14     # Intervalos de classe
15     breaks = data['breaks']
16     brk = np.arange(breaks['start'], breaks['end'] + breaks['h'], breaks['h'])
17
18     # Limite inferior da classe mediana
19     liM = brk[posM]
20
21     # Frequência acumulada anterior à classe mediana
22     if posM - 1 < 0:
23         sfaM = 0
24     else:
25         sfaM = fdt.iloc[posM - 1, 4]
26
27     # Frequência da classe mediana
28     fm = fdt.iloc[posM, 1]
29
30     # Largura da classe
31     h = breaks['h']
32
33     # Cálculo da mediana
34     res = liM + (((n / 2) - sfaM) * h) / fm
35
36     return res
```

R

```
testes.R
1 median.fdt <- function(x, ...)
2 {
3     fdt <- with(x,
4               table)
5
6     n <- fdt[nrow(fdt), 5]
7
8     posM <- grep(TRUE,
9               n / 2 <= fdt[, 5])[1]
10
11     brk <- with(x,
12               seq(breaks['start'],
13                 breaks['end'],
14                 breaks['h']))
15
16     liM <- brk[posM]
17
18     # It is important if 'posM' is inside of the first class
19     if (posM - 1 <= 0)
20         sfaM <- 0
21     else
22         sfaM <- fdt[(posM - 1), 5]
23
24     fm <- fdt[posM, 2]
25
26     h <- as.vector(with(x,
27                       breaks['h']))
28
29     res <- liM + (((n / 2) - sfaM) * h) / fm
30
31     return(res)
32 }
```



Scripts de Testes