

Bibliotecas em Python

Pandas – Parte III



INSTITUTO FEDERAL
Sudeste de Minas Gerais

Introdução

- O pacote Pandas é uma das ferramentas mais importante para os cientistas de dados.
- As poderosas ferramentas de aprendizado de máquina podem atrair toda a atenção, mas o Pandas pode ser considerado a espinha dorsal da maioria dos modelos de aprendizado de máquina.
- O Pandas será utilizado para limpar, transformar, analisar e visualizar seus dados por diferentes ângulos.
- Isso lhe ajudará a decidir o caminho que deverá seguir na modelagem dos dados.

Pandas

- O Pandas poderá trabalhar com vários formatos de arquivos, por questão de facilidade, recomendamos você trabalhar com arquivos em formato texto separados por vírgulas (CSV).
- Sugerimos ainda, trabalhar com padrão americano, ou seja, ponto para números decimais. Isso poderá evitar muita dor de cabeça futura.
- Para modificar o padrão do seu computador basta ir em Painel de Controle, em opções Região escolha Inglês (Estados Unidos).

Pandas

- Quando importamos um banco de dados no Pandas, se nosso banco de dados contém linhas e colunas ele será um DataFrame.
- Caso tenha apenas uma coluna, com várias linhas, será um Série.
- Poderíamos ter bancos de dados mais complexos, como, um sistema tridimensional, como por exemplo, vários DataFrames empilhados, nesse caso, teríamos um Painel.
- No entanto, por questões de facilidades, vamos trabalhar, basicamente com DataFrame e Séries.
- DataFrame e Séries são bastante similares, muitas funções são compartilhadas entre eles, no entanto, existem algumas diferenças.

Pandas

- Poderíamos criar um DataFrame simplesmente abrindo um arquivo CSV que temos em nosso computador.
- No entanto, isso será feito em vários momentos ao longo do curso. Então, agora, vamos criar um DataFrame utilizando o tipo de variável Dicionários (falamos desse tipo especial de variável do Python em capítulos anteriores).
- Criar um dicionário e então transformá-lo em DataFrame é uma boa maneira de registrar os resultados que iremos obter durante a execução do código, então podemos transformar isso em um DataFrame, então, salvá-lo como um arquivo CSV ou do MS Excel.

Pandas

- Veja o código abaixo.

```
8 #importa o Pandas como pd
9 import pandas as pd
10
11 #cria um dataframe vazio
12 df = pd.DataFrame()
13
14 #cria um dicionário, por exemplo de alguns resultados
15 data = {
16     'Media': 123.30,
17     'DesP': 55.60,
18     'CV': 23.50
19 }
20
21 #agora podemos adicionar o dicionário no DataFrame
22 df = df.append(data, ignore_index=True)
```

Pandas

- Agora vamos avançar na programação. Vamos imaginar que temos um código que executa várias vezes.
- Cada vez que executa gera-se um resultado diferente de média, desvio padrão e coeficiente de variação. Dessa forma, vamos simular um código que executa um cálculo 100 vezes.
- Para isso utilizamos a estrutura FOR. Para gerar resultados diferentes a cada momento, utilizamos a função *random*. Cada vez que o código executa, gera-se um novo dicionário, esse dicionário é adicionado no DataFrame df.
- Ao final do código iremos ter um DataFrame df com 100 linhas e três colunas. Veja o código abaixo.

Pandas

```
21 #agora podemos adicionar o dicionário no DataFrame
22 df = df.append(data, ignore_index=True)
23
24 #importa uma biblioteca para números aleatórios
25 from random import random
26
27 #vamos simular um cálculo realizado 100 vezes.
28 #toda vez que o cálculo é realizado, salvamos no dataframe
29 for i in range(1, 100):
30     media = random()*i
31     dp = random() * i
32     cv = dp/media*100
33
34     data = {
35         'Media': media,
36         'DesP': dp,
37         'CV': cv
38     }
39     df = df.append(data, ignore_index = True)
40
41 # o código acima terminou, o dataframe está cheio de resultados
42 # agora podemos salvar em Excel ou CSV.
43 df.to_excel("resultados_excel.xlsx", index = False)
44
45 #salvar em formato texto separado por vírgula (CSV)
46 df.to_csv("Resultados_csv.csv", index= False)
47
48 #Se a gente quiser abrir o arquivo salvo novamente no código.
49 df2 = pd.read_csv("Resultados_csv.csv")
```


Pandas

- O DataFrame contém várias funções. Está tudo muito bem documentado.
- Iremos apresentar algumas principais que utilizamos no dia a dia. Veja o código abaixo.
- Se quisermos visualizar as primeiras linhas podemos utilizar a função *head()*, conforme linha 52.
- Se você passar o parâmetro dentro do parênteses *head(5)*, por exemplo, será mostrado as 5 primeiras linhas do DataFrame.
- Se quisermos visualizar somente as últimas linhas do DataFrame, utilizamos, conforme linha 55, a função *tail()*.
- O atributo (veja que não é uma função pois não apresenta parênteses) da linha 58 é para apresentar o formato (número de linhas e colunas) do DataFrame. O atributo *shape* irá retornar uma tupla.

Pandas

- Então se quiser acessar somente o primeiro elemento da tupla (que é o número de linhas do DataFrame), basta informar por exemplo `df.shape[0]`.
- Veja, a regra é a mesma para acessar elementos de uma tupla. Por isso é importante entender a base do Python. Assim os códigos Python farão mais sentido.
- Na linha 61 é usada uma função no DataFrame `df` para remover linhas duplicadas. O resultado é armazenado no `df2`. Veja bem, já existia o `df2`, no entanto, agora ele foi substituído pelo resultado dessa operação.
- Na linha 62 imprimimos novamente o formado, agora para o `df2`.

Pandas

```
51 #ler as primeiras linhas
52 df.head() #se especificar um número dentro do parênteses será o n. de Linhas
53
54 #ler as últimas linhas
55 df.tail() #se especificar um número dentro do parênteses será o n. de Linhas
56
57 #tamanho do dataframe
58 df.shape
59
60 #remove linhas duplicadas
61 df2 = df.drop_duplicates()
62 df2.shape
63
64 #se usar o argumento inplace=True então a remoção será no df atual
65 #df.drop_duplicates(inplace = True)
66
67 #imprimir as colunas do df
68 df.columns
69
70 #renomear colunas
71 df2.rename(columns={'Media': 'MediaNew', 'CV': 'CVNew'}, inplace=True)
```

Pandas

- Na linha 68 mostramos as colunas presentes no DataFrame. Essa função é importante para a gente visualizar de forma rápida os nomes das colunas do DataFrame.
- O nome da coluna é utilizado para acessar valores do DataFrame, por exemplo. Na linha 71 iremos renomear os nomes das colunas. Veja você mesmo o que vai acontecer.
- Nessa linha é passado um parâmetro importante, o *inplace*. O parâmetro é passado como *True*. Se nada for passado ele será considerado *False* (default). Como o parâmetro é *True*, então, ele gera a modificação e passa a modificação para o *df2*.
- Entenda isso como uma atualização do *df2*. Caso fosse *False*, ele realiza a operação, mas não atualizar o *df2*. Então teríamos que atualizar manualmente da seguinte forma: *df2 = df2.rename(...)*. Ou seja, teríamos que substituir o *df2* pelo resultado da operação realizada no *df2*.

Pandas

- Para acessar elementos pelo índice da linha e coluna do DataFrame iremos utilizar a função *iloc*. Veja o código abaixo com vários exemplos de acesso, veja o resultado no console de cada uma delas.

```
73 #Função ILOC
74 x = df.iloc[1,1] #Linha 1 e coluna 1
75
76 x = df.iloc[1,1:3] #Linha 1 e coluna de 1 a 2
77
78 x = df.iloc[0] #retorna uma Serie com a primeira linha
79
80 x = df.iloc[[0]] #retorna um dataframe com a primeira linha
81
82 x = df.iloc[[1,2]] #retorna as linhas 1 e 2
83
84 x = df.iloc[:10] #retorna a Linha de 0 a 9
85
86 x = df.iloc[[1,2],[0,1]] #retorna a linha 1 e 2 e colunas 0 e 1
87
88 x = df.iloc[:9,:2] #retona de linha 0 a 8 e coluna 0 a 1
```

Pandas

- No código acima vale destacar o resultado da linha 78 e 80. Na linha 78 retorna como resultado uma Série e na linha 80 um DataFrame. Isso poderá fazer diferença como irá utilizar esses resultados posteriormente.
- Outra forma de acessar valores do DataFrame é utilizando a Função loc.
- A função loc é mais poderosa, pois além de acessar valores pelos nomes das colunas, também aceitar estrutura lógica comparadora. Então podemos utilizar a função loc como um filtro. Veja alguns exemplos abaixo.

```

90 #Função LOC
91 x = df.loc[:,['Media']] #toda coluna média
92
93 x = df.loc[:,['Media', 'DesP']] #todas as colunas media e desp
94
95 x = df.loc[[1], ['Media', 'DesP']] #linha 1 e colunas Media e Desp
96
97 x = df.loc[1:10, ['Media', 'DesP']] #linha de 1 a 10 e colunas media e desp
98
99 x = df.loc[:10, ['Media', 'DesP']] #linha de 0 a 10.
100
101 #Outra alternativa
102 x = df['Media'] #lista toda a coluna Media
103
104 #Retorna True or False
105 x1 = df['Media'] > 50
106
107 #retorna True or False com AND(&)
108 x2 = (df['Media'] > 50) & (df['DesP'] > 10)
109 |
110 #retorna True or False com OR()
111 x3 = (df['Media'] > 50) | (df['DesP'] > 10)
112
113 #Operações com dataframe usando LOC
114 x = df.loc[df.Media > 50]
115
116 #ou
117 x = df.loc[x1]
118
119 #outra operação
120 x = df.loc[x2]

```



Pandas

- A melhor forma de entender como funciona o código é realizar as operações e visualizar o que aconteceu.
- Para isso você poderá executar o código linha a linha e verificar o que foi armazenado nas variáveis. No entanto, vou destacar as linhas 105, 108 e 111.
- Nessas linhas são realizadas operações comparadoras no DataFrame. O resultado de cada operação dessa é uma Série com valores verdadeiros ou falsos.
- Na linha 114 é utilizado a função *loc* com a mesma operação realizada na linha 105.
- O resultado da linha 114 será o mesmo da linha 117.
- Isso foi feito para entender como funciona a lógica da função *loc*.



Pandas

- Veja na linha 129 que a função `range` foi utilizada para gerar uma lista de valores de 1 a 9. Então esses valores serão utilizados para serem removidos utilizando a função `drop`.
- Na linha 131 a 135 é realizada uma filtragem, depois na linha 133 gera-se o DataFrame utilizando o filtro.
- Depois gera-se a lista com os valores dos índices que foram filtrados.
- Depois aplica a lista na função `drop` na linha 135.
- Na linha 138 e 139 é simulado a presença de alguns valores ausentes (NaN), então esses valores serão removidos com a função `dropna`, veja na linha 140 e 141 que `axis = 0` indica que será removido a linha inteira e `axis = 1` indica que será removida a coluna inteira com a presença do valor ausente.

Pandas

```
122 #remover colunas inteiras
123 x = df.drop(['Media'], axis = 1) #nome da coluna e axis = 1 (remover coluna)
124
125 #remover uma linha específica
126 x = df.drop([1,2,4,6,7]) #lista dos valores dos índices a remover
127
128 #remover linhas inteiras
129 x = df.drop(range(1,10)) #Lista com os valores dos índices da linha
130
131 #remover linhas com valores de certas condições
132 x3 = (df['Media'] > 5) | (df['DesP'] > 10)
133 x = df.loc[x3]
134 lista_indice = x.index
135 x = df.drop(lista_indice)
136
137 #remover linhas inteiras com valores NAN (vazio)
138 df.iloc[1,2] = float('NaN') #simular um valor vazio
139 df.iloc[5,1] = float('NaN') #simular um valor vazio
140 x = df.dropna(axis = 0) #axis = 0 remove a linha com nan
141 x= df.dropna(axis = 1) #remove coluna inteira
```

Pandas

- Podemos ainda realizar operações estatísticas nas colunas, veja o código:

```
143 #algumas estatísticas
144 media = df['Media'].mean()
145 soma = df['Media'].sum()
146 minimo = df['Media'].min()
147 maximo = df['Media'].max()
149 #criar novas colunas
150 df['classeCV'] = 0
151
152 #modificar as linhas com base em uma condição
153 x = df['CV'] > 40
154 df.loc[x, ['classeCV']] = 1
155
156 #outra opção usar Numpy
157 import numpy as np
158 df['classeCV'] = np.where(df['CV'] >= 30,1, df['classeCV'])
159 df['classeCV'] = np.where(df['CV'] > 60,2, df['classeCV'])
```

Pandas

- Podemos acrescentar novas colunas e modificar colunas com base em regras.
- Na linha 150 é criado uma coluna 'classeCV' com todos as linhas igual a 0.
- Na linha 153 e 154 é realizado uma filtragem e sem seguida a coluna classe onde a operação é verdadeira é modificado para 1.
- Outra forma de fazer, praticamente a mesma operação anterior, é utilizar a biblioteca Numpy, conforme apresentado na linha 158 e 159.
- Na linha 158 onde os valores de df['CV'] é maior ou igual a 30, a coluna df['classeCV'] assume valor 1, caso contrário, o valor permanece igual a df['classeCV'], ou seja, não altera.
- Então, dessa forma, podemos aplicar várias condições no DataFrame sem realizar nenhuma estrutura FOR e IF.

Pandas

- Além dessas funções, temos também função de visualização gráfica de dados.
- O objetivo aqui é apenas fazer uma visualização rápida.
- Para plotar gráficos com mais recursos gráficos, sugerimos a utilização da biblioteca do Matplotlib, veja abaixo um exemplo para visualização gráfica de dados do Pandas.

```
170 #gráficos
171 df['Media'].hist()
172
173 df['Media'].plot()
```

Pandas

- O Pandas não morre por aqui, temos uma infinidade de métodos e atributos que poderão ser utilizados, fora a integração o Pandas com Numpy.
- O que sugiro é, sempre que precisar fazer algo no seu banco de dados, provavelmente existirá uma função para tal.
- Basta fazer uma pesquisa rápida no Google que, provavelmente irá encontrar algo.
- O que você precisa saber é especificar a palavra chave correta da função que deseja.



INSTITUTO FEDERAL
Sudeste de Minas Gerais