Bibliotecas em Python

Pandas

INSTITUTO FEDERAL
Sudeste de Minas Gerais

Pandas

- Pandas é uma biblioteca open-source, amplamente utilizada no python
- Fácil manipulação de estrutura de dados
- Alta performance
- Bastante utilizado para realizar a análise de dados
- Para utilizar a biblioteca pandas é necessário importá-la antes de utilizar

```
import pandas as pd
```

Pandas

- 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.

Criando um dataframe

```
d = {‘one’ : pd.Series([1, 2, 3], index=[‘a’, ‘b’, ‘c’]), ‘two’ : pd.Series([1, 2, 3, 4], index=[‘a’, ‘b’, ‘c’, ‘d’])}

df = pd.DataFrame(d)

df
```

	one	two
a	1	1
b	2	2
c	3	3
d	Nan	4

Salvando um dataframe

Podemos salvar o conteúdo de um dataframe em um arquivo, por exemplo, um CSV

df.to_csv('dados.csv', index=False, header=False)

Importando um CSV

- A maneira mais comum de utilizar o pandas é importando um arquivo CSV contendo dados.
- O arquivo CSV também pode ser lido em um programa de planilhas (Excel, Calc, etc)

df = pd.read_csv(‘arquivo.csv’)

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).

Comandos básicos

- Obtendo informações do dataframe

  df.info()

- Obtendo uma amostra dos primeiros valores do dataframe

  df.head(3)

- Número de linhas e colunas

  df.shape

- Nomes das colunas

  df.columns

Comandos básicos

- Número de valores únicos

df ["coluna"].nunique()

- Amostra dos valores únicos

df [“coluna”].unique()

- Contagem dos valores

df [“coluna”].value_counts()

Comandos básicos

- Contagem de valores nulos por colunas

df.isnull().sum()

- Apagar as linhas que contém valor nulo

df.dropna()

- Substituir valor nulo por um valor padrão

df.fillna(-9999)

- Calcular a média de um valor

df.mean()

Comandos básicos

- Selecionando apenas uma coluna

```
df ["sepal_length"]
```

- Selecionando várias colunas e inserindo em outro dataframe

```
X = df [["a", "b", "c"]]
```

- Seleciona a coluna por número

```
df.iloc[:, [1,3,4]]
```

- Apagando uma coluna do dataframe

```
X = df.drop("b", axis=1)
```

- Renomeando colunas

```
df.rename(columns={"a": "novo_a", "b": "novo_b"})
```

Comandos básicos

- Ordenando os valores

```
df.sort_values(by = "a", ascending = True)
```

- Criando uma nova coluna através de outra

```
df ['nova_col'] = df ["a"]*2
```

- Criando uma nova coluna a partir de uma condição

```
df ['nova_col'] = ["valor 1" if i < 3 else "valor 2" for i in df["a"]]
```

Comandos básicos

- Selecionando da linha 3 até a 10

```
df.iloc[3:10,]
```

- Seleciona 10 linhas aleatórias

```
df.sample(10)
```

- Localizando as linhas com um valor específico

```
df [df ["a"].isin(["valor 1"])]
```

Comandos básicos

- Filtrando através de uma condição

```
df [df.a >= 5]
```

```
df [(df.a > 1) & (df.b=="abc") | (df.c < 3)]
```

- Apagando linhas

```
df.drop(dados.index[1])
```

Pandas

- 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.

```
#importa o Pandas como pd
import pandas as pd

#cria um dataframe vazio
df = pd.DataFrame()

#cria um dicionário, por exemplo de alguns resultados
data = {
    'Media': 123.30,
    'DesP': 55.60,
    'CV': 23.50
        }

#agora podemos adicionar o dicionário no DataFrame
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

```
#agora podemos adicionar o dicionário no DataFrame
df = df.append(data, ignore_index=True)

#importa uma biblioteca para números aleatórios
from random import random

#vamos simular um cálculo realizado 100 vezes.
#toda vez que o cálculo é realizado, salvamos no dataframe
for i in range(1, 100):
    media = random()*i
    dp = random() * i
    cv = dp/media*100

    data = {
    'Media': media,
    'DesP': dp,
    'CV': cv
        }
    df = df.append(data, ignore_index = True)

# o código acima terminou, o dataframe está cheio de resultados
# agora podemos salvar em Excel ou CSV.
df.to_excel("resultados_excel.xlsx", index = False)

#salvar em formato texto separado por vírgula (CSV)
df.to_csv("Resultados_csv.csv", index= False)

#Se a gente quiser abrir o arquivo salvo novamente no código.
df2 = pd.read_csv("Resultados_csv.csv")
```

INSTITUTO FEDERAL
Sudeste de Minas Gerais

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.

Pandas

```
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 columa 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

```
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

- 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

- 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.

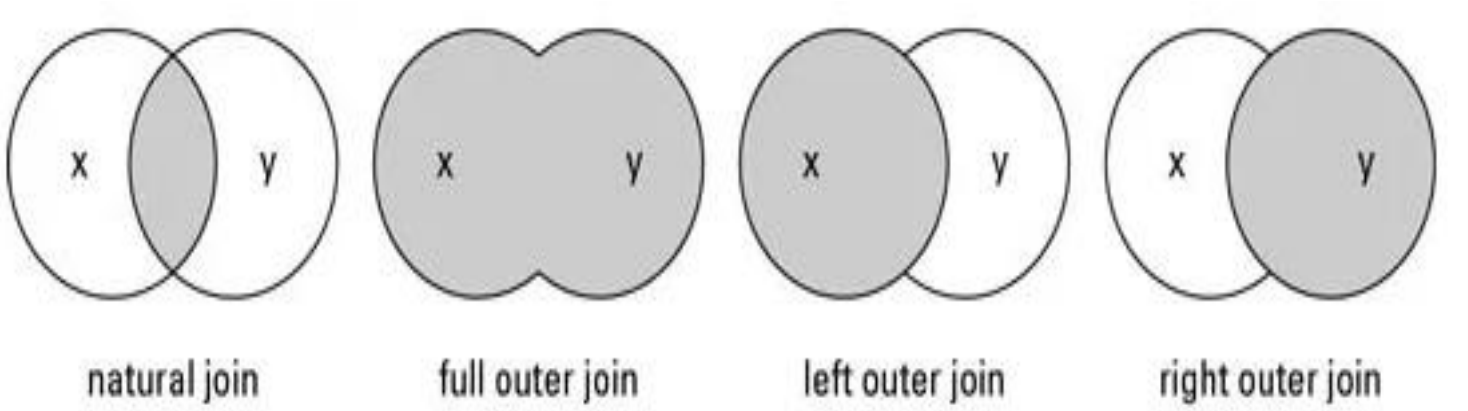
```
170 #gráficos
171 df['Media'].hist()
172
173 df['Media'].plot()
```

Combinação de dataframes

- Os principais comandos para combinar dataframe são: merge() e concat()
- Os métodos merge() e concat() são utilizados para unirmos data frames de maneira objetiva e com poucos dados.
- Com pequenos comandos podemos unir grandes quantidades de dados.

Tipos de Combinação de dataframes

- Assim como em bancos de dados, podemos combinar conjuntos de dados de maneiras diferentes, sendo elas:



merge()

- Interseção entre os conjuntos de dados
  m = pd.merge(a, b, how = 'inner', on = 'key')
- União entre os conjuntos de dados
  m = pd.merge(a, b, how = 'outer')
- Conjunto da esquerda ou direita
  m = pd.merge(a, b, how = 'left', on = 'key')
  m = pd.merge(a, b, how = 'right', on = 'key')

concat()

- O método concat é utilizado para concatenar linearmente o dados, ou seja, os dados serão novas linhas no dataframe.
- Bastante utilizado quando o dataframe está particionado em diversos pedaços, sendo assim, ele poderá se tornar apenas um único dataframe.
  pd.concat([a_df, b_df])

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