Introdução ao PYTHON para Inteligência Artificial – PLP – IMT (2024.1)

1. Pandas

O Pandas é uma biblioteca para a manipulação de dados em Python. É uma biblioteca que fornece uma série de recursos de leitura e manipulação de grandes bases de dados com uma grande variedade de comandos.

Inicialmente aprenderemos como abrir bases de dados (datasets) armazenados em Excel (.xlsx) e CSV (.csv) e depois criaremos nosso próprio arquivo de dados como um DataFrame.

Para utilizar o Pandas dentro de nossos projetos é necessário que seja feita a importação da biblioteca. Ela já se encontra pré-instalada, então para realizar a importação devemos apenas escrever o código abaixo:

```
import pandas as pd
```

Note que demos o apelido de pd para o Pandas. Isso facilitará a utilização frequente de seus comandos. A primeira importação de dados que faremos será uma planilha Excel. Para isso vá até o drive da disciplina e faça download do arquivo "dados.xlsx", sendo .xlsx a extensão padrão do Excel hoje em dia. A figura 1 mostra os dados contidos neste arquivo.

Figura 1. Planilha Excel básica

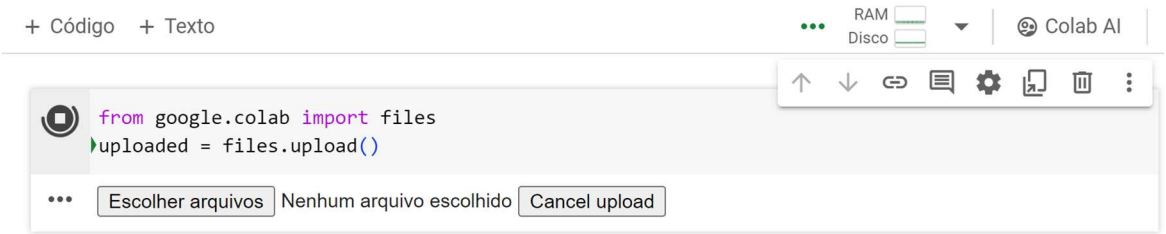
	A	B
1	numero	letras
2	1	a
3	2	b
4	3	c
5	4	d
6	5	e
7	6	f
8	7	g
9	8	h
10	9	j
11	10	k

Agora que temos nossa planilha criada vamos carregá-la no Colab. Esse processo é um pouco diferente de outros ambientes. As instruções para isso no Colab são:

```
from google.colab import files
uploaded = files.upload()
```

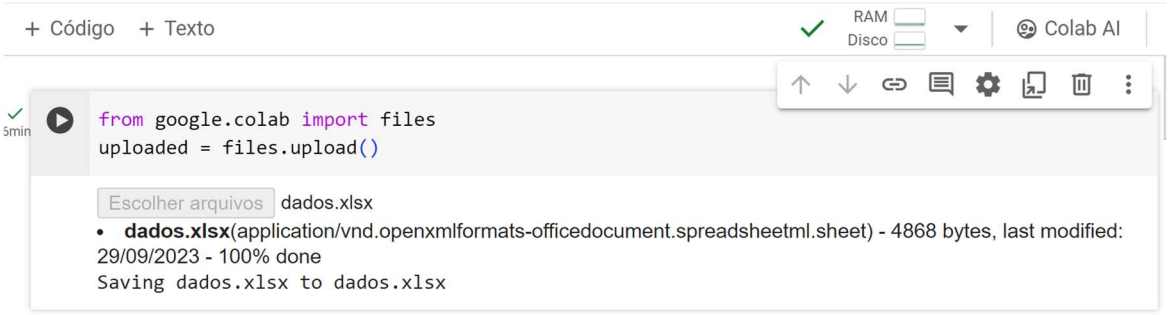
Assim que o algoritmo for executado, será solicitado que você faça a seleção dos arquivos para upload. Neste caso, selecione "dados.xlsx". Na figura 2, podemos ver um exemplo dessa interação.

Figura 2. Seleção dos arquivos para upload.



Se tudo der certo será apresentado o processo de carregamento até os 100% e aparecerá a informação que o arquivo foi carregado conforme figura 3.

Figura 3: Arquivo 100% Carregado



Cuidado! Não crie nome de arquivos com espaço em branco. Eles podem dar problema ao carregar ou na hora de indicar a leitura. Além disso, sempre verifique o nome assumido ao salvar o arquivo. Se um arquivo com o mesmo nome já existia, outro nome automaticamente é dado com sufixo " (n)" dependendo de quantos arquivos com o mesmo nome já existem.

Assim como usamos o "upload", podemos usar o "download" para baixar os arquivos do Colab em nosso computador:

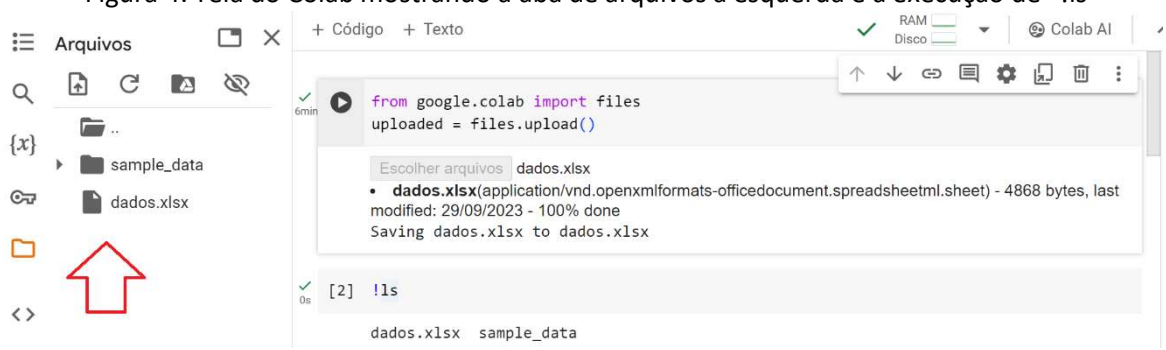
```
files.download("dados.xlsx")
```

A partir do momento que os arquivos estão no Colab alguns comandos simples e úteis são "!ls" (mostra os arquivos armazenados) "!rm" <nome do arquivo> (remove o arquivo).

Além disso você pode clicar no símbolo de pasta à esquerda da tela (seta vermelha em destaque na figura 4) que apresentará os arquivos por onde também é possível gerenciar os arquivos de dados.

A figura 4 mostra também o comando "!ls" executado mostrando os arquivos existentes e a aba de arquivos aberta à esquerda.

Figura 4. Tela do Colab mostrando a aba de arquivos a esquerda e a execução de "!ls"



Depois de carregar o arquivo no Colab, precisamos preparar o arquivo para leitura. A instrução a seguir é responsável por isso:

```
dados1 = pd.read_excel('dados.xlsx')
```

Sempre que criamos um novo dataset, podemos utilizar dois comandos para conhecer um pouco sobre o arquivo aberto. O atributo "shape" mostra uma tupla com o número de linhas e colunas do arquivo e a função "head" mostra os primeiros registros do dataset.

Veja o uso do “shape”:

```
dados1.shape
```

```
(10, 2)
```

Veja o uso do “head”:

```
dados1.head()
```

	números	letras
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e

O padrão da função “head” é mostrar os primeiros 5 registros, mas se você passar como argumento a quantidade de registros ele apresentará a quantidade solicitada:

```
dados1.head(8)
```

	números	letras
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e
5	6	f
6	7	g
7	8	h

Esses comandos iniciais nos permitem ter uma ideia da base de dados que abrimos e por isso vale a pena executá-los.

A seguir carregaremos um arquivo CSV. Para isso selecionamos a base de dados “growth.csv”, também disponibilizada como material para essa aula. Ela é uma base de dados pública disponível para download no Kaggle (pronuncia-se “kégol”), um lugar interessante para pegarmos bases de dados públicas para fazermos nosso estudo:

https://www.kaggle.com/

A base de dados que escolhemos, chamada “growth.csv”, guarda informações sobre o crescimento populacional mundial ano a ano, desde 1951 até 2023. Além de já ter sido disponibilizada no material desta aula, também pode ser encontrada no link abaixo:

https://www.kaggle.com/datasets/maheshmani13/world-population-growth?resource=download

Para carregar a base CSV, utilizamos o comando abaixo, da mesma forma que no exemplo anterior do Excel.

```
uploaded = files.upload()
```

Com o arquivo carregado basta usarmos uma função bem parecida com a usada para arquivos Excel agora para o arquivo CSV:

```
dados2 = pd.read_csv('growth.csv')
```

Para conhecer melhor o arquivo, como anteriormente usamos o “shape” e o “head”:

```
dados2.shape
```

```
(73, 5)
```

```
dados2.head()
```

	Year	Population	Yearly Growth %	Number	Density (Pop/km2)
0	1951	2,543,130,380	1.75%	43,808,223	17
1	1952	2,590,270,899	1.85%	47,140,519	17
2	1953	2,640,278,797	1.93%	50,007,898	18
3	1954	2,691,979,339	1.96%	51,700,542	18
4	1955	2,746,072,141	2.01%	54,092,802	18

Temos então as 5 colunas com os 5 primeiras linhas e podemos ter uma ideia do *dataset* que estamos utilizando. A partir daí é possível excluir colunas que não nos interessam, repartir,e manipular o *dataset* de forma geral.

A terceira forma de trabalhar com *datasets* é criarmos ele a partir de um dicionário. O dicionário é utilizado pois apresenta, para cada posição, um valor e uma chave que aponta para esse valor. Veja o exemplo de um dicionário de alunos:

```
alunos = {'Nome':['Marcelo','Lucas','Cynthia','Helena'],
          'Media':[4,7,5.5,9],
          'Status':['Reprovado','Aprovado','Reprovado','Aprovado']}
```

Para transformar este dicionário em um dataframe utilizamos a função “DataFrame”. Veja:

```
df1 = pd.DataFrame(alunos)
```

A partir de agora podemos usar “alunos” como os outros *datasets*:

```
df1.shape
```

```
(4, 3)
```

Ou ainda:

```
df1.head()
```

	Nome	Media	Status
0	Marcelo	4.0	Reprovado
1	Lucas	7.0	Aprovado
2	Cynthia	5.5	Reprovado
3	Helena	9.0	Aprovado

Além do “shape” e “head()” existem algumas outras funções interessantes para explorarmos nossas bases de dados. A primeira delas é a “describe()” que trás de uma vez uma série de informações importantes (quantidade, média, valor máximo, valor mínimo, desvio padrão e percentis) sobre as colunas numéricas do *dataframe*:

```
df1.describe()
```

	Media
count	4.000000
mean	6.375000
std	2.136001
min	4.000000
25%	5.125000
50%	6.250000
75%	7.500000
max	9.000000

Para filtrar colunas específicas basta indicar o nome do *dataframe* com a coluna entre parênteses:

```
df1[['Nome']]
```

	Nome
0	Marcelo
1	Lucas
2	Cynthia
3	Helena

Para filtrar linhas podemos usar o atributo “loc” indicando a numeração das linhas entre colchetes:

```
df1.loc[1:3]
```

	Nome	Media	Status
1	Lucas	7.0	Aprovado
2	Cynthia	5.5	Reprovado
3	Helena	9.0	Aprovado

Podemos ainda estabelecer condições no “loc”. Veja os exemplos:

```
df1.loc[df1['Status'] == 'Aprovado']
```

	Nome	Media	Status
1	Lucas	7.0	Aprovado
3	Helena	9.0	Aprovado

```
df1.loc[df1['Media'] < 7.0]
```

	Nome	Media	Status
0	Marcelo	4.0	Reprovado
2	Cynthia	5.5	Reprovado

Além de filtrar é possível também construir novos *dataframes* a partir de condições específicas. Para isso basta atribuir a um novo *dataframe*. Veja:

```
exame = df1.loc[df1['Media'] < 7.0]
print(exame)
```

	Nome	Media	Status
0	Marcelo	4.0	Reprovado
2	Cynthia	5.5	Reprovado

Ou ainda selecionando colunas:

```
exame = df1[['Nome','Media']].loc[df1['Media'] <
7.0]print(exame)
```

```
      Nome  Media
0  Marcelo    4.0
2  Cynthia    5.5
```

Para renomear uma coluna, por exemplo, basta usar o “rename”. Você indica entre parênteses os nomes atuais das colunas e após “:” o novo nome. Consideraremos o dataframe “dados1”, que já criamos anteriormente, para este teste.

```
print(dados1.rename(columns={"numero":"nr","letras":"ltr"}))
```

Para excluir uma coluna que pode não ser de nosso interesse, utilizamos a funções “drop”. Temos que indicar qual a coluna alvo (no caso a “numero”), que nosso alvo é uma coluna (“axis=1”) e que os dados devem ser excluídos também em dados1 (“Inplace=True”):

```
dados1.drop('numero',axis=1,inplace=True)
print(dados1)
```

Podemos também realizar **operações aritméticas** com valores numéricos nas colunas/linhas de um data frame. Para estes exemplos, criaremos um data frame com dados fictícios de vendas:

```
# Dados fictícios de vendas
data = {'Produto': ['Camisa', 'Calça', 'Tênis', 'Boné', 'Meia'],
        'Preço_Unitário': [25.99, 49.99, 89.99, 19.99, 9.99],
        'Quantidade_Vendida': [100, 50, 30, 80, 120]}

# Criando o DataFrame
df_vendas = pd.DataFrame(data)
```

Para calcular o total de vendas da empresa para cada produto, podemos criar uma nova coluna a partir da multiplicação entre os elementos da coluna “Preço Unitário” e “Quantidade Vendida”:

```
# Calculando o total de vendas para cada produto
df_vendas['Total_Vendas'] = df_vendas['Preço_Unitário'] *
df_vendas['Quantidade_Vendida']
```

Para calcular o total de vendas da empresa (a soma do total vendido para todos os produtos), podemos utilizar a função sum() nesta nova coluna, como mostrado abaixo:

```
# Calculando o total de vendas da empresa
```

```
print("Total de Vendas da Empresa:", df_vendas['Total_Vendas'].sum())
```

Para calcular a média dos valores de uma coluna, podemos utilizar a função mean():

```
# Calculando a média do preço unitário, quantidade vendida e total de vendas
print("Média do Preço Unitário:", df_vendas['Preço_Unitário'].mean())
print("Média da Quantidade Vendida:", df_vendas['Quantidade_Vendida'].mean())
print("Média do Total de Vendas:", df_vendas['Total_Vendas'].mean())
```

Por fim, testaremos as funções idxmax() e idxmin(), que retornam o maior e o menor valor, respectivamente, dado um conjunto de valores numéricos:

```
# Obtendo o produto mais vendido (índice do valor máximo na coluna
'Quantidade_Vendida')
produto_mais_vendido = df_vendas.loc[df_vendas['Quantidade_Vendida'].idxmax(),
'Produto']
print("Produto Mais Vendido: ", produto_mais_vendido)

# Obtendo o produto menos vendido (índice do valor mínimo na coluna
'Quantidade_Vendida')
produto_menos_vendido =
df_vendas.loc[df_vendas['Quantidade_Vendida'].idxmin(), 'Produto']
print("Produto Menos Vendido:", produto_menos_vendido)
```