FICHA TRABALHO 2

MODALIDADE:	Aprendizagem +	Não aplicável	
CURSO:	Técnico/a Especialista em Gestão de Informação e Ciência dos Dados		
UFCD:	Limpeza e transformação de dados em Python	CÓDIGO UFCD:	10808
FORMADOR/A:	Bruno Silva	DATA:	

OBJETIVOS

- Comandos básicos com a biblioteca Pandas

Parte 1 – Introdução aos comandos básicos da biblioteca Pandas

Parte 1.1 – Método Info()

O método **.info()** permite examinar os tipos de dados, os valores ausentes e o tamanho dos dados de um DataFrame.

Para além da invocação do método, temos alguns argumentos:

- **show_counts** com o valor True - fornece alguns valores sobre o total de valores não ausentes em cada coluna;
- **memory_usage** com o valor True - mostra o uso total de memória dos elementos do DataFrame.
- **verbose** com o valor True - imprime o resumo completo do método .info().

```
7  #compressão de dados
8  print(data.info(show_counts=True, memory_usage=True, verbose=True))
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Parte 1.2 – Método Shape()

O número de linhas e colunas de um DataFrame pode ser identificado usando o atributo .shape. Este retorna um tuplo (linha, coluna) e pode ser indexado para obter apenas linhas e apenas colunas:

```
7   #Compreender os dados
8   print(data.shape) #retorna o objeto
9   print(data.shape[0]) #obtem o número de linhas
10  print(data.shape[1]) #obtem o número de colunas
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

(768, 9)
768
9
```

Parte 1.3 – Método len()

Se quiser saber apenas o número de registos de forma rápida, mas invocar a função len() e passar dentro dos parenteses o dataframe:

```
print(f"Numero de registos: {len(data)}")
```

```
Numero de registos: 768
```

Parte 1.4 – Método describe()

O método .describe() imprime as **estatísticas resumidas dos dados numéricos**, como contagem, média, desvio padrão, intervalo e quartis de colunas numéricas: mínimo, (25% 1º Quartil; 50% → mediana ou 2º Quartil; 75% → 3º Quartil e máximo):

```
7   #resumo de dados
8   print(data.describe())
9
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  Python

       Pregnancies     Glucose  BloodPressure  ...  DiabetesPedigreeFunction         Age     Outcome
count   768.000000  768.000000     768.000000  ...                768.000000  768.000000  768.000000
mean      3.845052  120.894531      69.105469  ...                  0.471876   33.240885    0.348958
std       3.369578   31.972618      19.355807  ...                  0.331329   11.760232    0.476951
min       0.000000    0.000000       0.000000  ...                  0.078000   21.000000    0.000000
25%       1.000000   99.000000      62.000000  ...                  0.243750   24.000000    0.000000
50%       3.000000  117.000000      72.000000  ...                  0.372500   29.000000    0.000000
75%       6.000000  140.250000      80.000000  ...                  0.626250   41.000000    1.000000
max      17.000000  199.000000     122.000000  ...                  2.420000   81.000000    1.000000

[8 rows x 9 columns]
```

Também pode **modificar os quartis** usando o argumento **percentiles**. Vamos analisar os percentis de 30%, 50% e 70% das colunas numéricas no DataFrame:

```
7  #resumo de dados (com percentagens personalizadas)
8  print(data.describe(percentiles=[0.3, 0.5, 0.7]))
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

	Pregnancies	Glucose	BloodPressure	...	DiabetesPedigreeFunction	
count	768.000000	768.000000	768.000000	...	768.000000	76
mean	3.845052	120.894531	69.105469	...	0.471876	
std	3.369578	31.972618	19.355807	...	0.331329	
min	0.000000	0.000000	0.000000	...	0.078000	
30%	1.000000	102.000000	64.000000	...	0.259000	
50%	3.000000	117.000000	72.000000	...	0.372500	
70%	5.000000	134.000000	78.000000	...	0.563700	
max	17.000000	199.000000	122.000000	...	2.420000	

[8 rows x 9 columns]

No entanto também podem mostrar todos os campos se desejar, do qual, basta incluir dentro dos parenteses da função describe a opção include="all":

```
print(data.describe(include="all"))
```

Parte 1.5 – Método head()

Podemos visualizar as **primeiras** ou as **últimas** linhas de um DataFrame usando os métodos **.head()** ou **.tail()**, respetivamente.

Também pode especificar o **número de linhas** a apresentar por meio do argumento **n** (quando não especificado, o valor por defeito é 5).

```
7  #ler o cabeçalho do ficheiro
8  print(data.head())
9
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	...	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	...	33.6	0.627	50
1	1	85	66	29	...	26.6	0.351	31
2	8	183	64	0	...	23.3	0.672	32
3	1	89	66	23	...	28.1	0.167	21
4	0	137	40	35	...	43.1	2.288	33

[5 rows x 9 columns]

Também pode especificar o **número de linhas** a apresentar por meio do argumento **n** (neste caso mostrar apenas as 3 primeiras linhas):

```
7  #ler o cabeçalho do ficheiro
8  print(data.head(n=3))
9
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

	Pregnancies	Glucose	BloodPressure	SkinThickness	...	BMI	DiabetesPedi
0	6	148	72	35	...	33.6	
1	1	85	66	29	...	26.6	
2	8	183	64	0	...	23.3	

[3 rows x 9 columns]

Parte 1.6 – Método tail()

Podemos visualizar as **últimas** linhas de um DataFrame (por defeito só mostra apenas 5 linhas)

```
7   #ler o cabeçalho do ficheiro
8   print(data.tail())
9
10
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

     Pregnancies  Glucose  BloodPressure  SkinThickness  ...   BMI  DiabetesPedigr
763           10      101             76             48  ...  32.9
764            2      122             70             27  ...  36.8
765            5      121             72             23  ...  26.2
766            1      126             60              0  ...  30.1
767            1       93             70             31  ...  30.4

[5 rows x 9 columns]
```

Também pode especificar o **número de linhas** a apresentar por meio do argumento **n** (neste caso mostrar apenas as 3 últimas linhas):

```
7   #ler o cabeçalho do ficheiro
8   print(data.tail(n=3))
9
10
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

     Pregnancies  Glucose  BloodPressure  SkinThickness  ...   BMI  Dia
765            5      121             72             23  ...  26.2
766            1      126             60              0  ...  30.1
767            1       93             70             31  ...  30.4

[3 rows x 9 columns]
```

Parte 1.7 – Método describe(): Resumo de dados numéricos

Também pode isolar tipos de dados específicos usando o argumento include. Vamos resumir dados apenas nas colunas com números inteiros (int):

```
7   #resumo de dados (com percentagens personalizadas)
8   print(data.describe(include=[int]))
9
10
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

       Pregnancies     Glucose  BloodPressure  SkinThickness
count   768.000000  768.000000     768.000000     768.000000
mean      3.845052  120.894531      69.105469      20.536458
std       3.369578   31.972618      19.355807      15.952218
min       0.000000    0.000000       0.000000       0.000000
25%       1.000000   99.000000      62.000000       0.000000
50%       3.000000  117.000000      72.000000      23.000000
75%       6.000000  140.250000      80.000000      32.000000
max      17.000000  199.000000     122.000000      99.000000
```

Também pode **excluir** tipos de dados específicos usando o argumento **exclude**. Vamos resumir dados apenas nas colunas com números inteiros (int):

```
7   #resumo de dados
8   print(data.describe(exclude=[int]))
9
10
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

                BMI  DiabetesPedigreeFunction
count   768.000000                768.000000
mean     31.992578                  0.471876
std       7.884160                  0.331329
min       0.000000                  0.078000
25%      27.300000                  0.243750
50%      32.000000                  0.372500
75%      36.600000                  0.626250
max      67.100000                  2.420000
```

Parte 1.8 – Método describe().T : Resumo de dados por transposição

Muitas vezes, podemos ter a necessidade de transpor a informação, ou seja, trocar os campos das linhas pelas colunas com o método .describe().T

```
7   #resumo de dados (com os dados transpostos)
8   print(data.describe().T)
9
10
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                Python

              count        mean         std    min       25%       50%
Pregnancies   768.0    3.845052    3.369578  0.000   1.00000    3.0000   6.0
Glucose       768.0  120.894531   31.972618  0.000  99.00000  117.0000  140.2
BloodPressure 768.0   69.105469   19.355807  0.000  62.00000   72.0000   80.0
SkinThickness 768.0   20.536458   15.952218  0.000   0.00000   23.0000   32.0
Insulin       768.0   79.799479  115.244002  0.000   0.00000   30.5000  127.2
BMI           768.0   31.992578    7.884160  0.000  27.30000   32.0000   36.0
```

Parte 1.9 – Método set_index() e inplace

Se quisermos colocar uma coluna **como índice padrão** (que identifica a linha do registo) do Dataframe, podemos usar a função **set_index()**

Dentro da função usamos 2 argumentos:

- Qual a **coluna** que quer colocar como **índice**;
- O argumento **inplace=True** faz com que a modificação de uma operação seja aplicada diretamente ao Dataframe original.

Se quisermos converter a coluna “age” (idade) **no índice padrão** (que identifica a linha do registo) do Dataframe, podemos usar a função **set_index()**

```
#mudar o indice
data.set_index('Age', inplace=True)

#ver os registo com dados
print(data.head(3))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                    Python +

         Pregnancies  Glucose  BloodPressure  ...   BMI  DiabetesPedigreeFunction  Outcome
Age                                           ...
50                 6      148             72  ...  33.6                     0.627        1
31                 1       85             66  ...  26.6                     0.351        0
32                 8      183             64  ...  23.3                     0.672        1

[3 rows x 8 columns]
```

Se quiser reverter a situação, podemos usar o comando **df.reset_index(inplace=True)**

```
#remover o indice anterior
data.reset_index(inplace=True)

#ver os registo com dados
print(data.head(3))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

   Age  Pregnancies  Glucose  BloodPressure  ...  Insulin   BMI  DiabetesPedigre
0   50            6      148             72  ...        0  33.6
1   31            1       85             66  ...        0  26.6
2   32            8      183             64  ...        0  23.3

[3 rows x 9 columns]
```

Parte 1.10 – Operações com colunas:

Parte 1.10.1 – Método .columns()

O atributo **.columns** retorna os nomes das colunas (em forma de um vetor). Para aceder ao nome, basta invocar o atributo com a indicação do índice:

```
#Obter todas as colunas e nomes
print(data.columns) #retorna o vetor com os nomes das colunas
print(data.columns[3]) #ontem a coluna 4
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
SkinThickness
```

Ainda poderá pedir para converter as obtenções das colunas num formato de lista com a função **.list()**

```
7   #Obter todas as colunas e nomes
8   print(list(data.columns))
```

```
['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', '
Age', 'Outcome']
```

Parte 1.10.2 – Obter dados das colunas (forma isolada ou em conjunto)

Para obter os dados de apenas de 1 coluna, basta fazer:

```
#Obter dados colunas
print(data["Glucose"])
```

```
Name: count, dtype: int64
0      148
1       85
2      183
3       89
4      137
      ...
763    101
764    122
```

Para obter os dados de várias colunas, basta fazer:

```
#Obter dados colunas
print(data[["Glucose", "Age"]])
```

```
Name: count, dtype: int64
     Glucose  Age
0        148   50
1         85   31
2        183   32
3         89   21
4        137   33
..       ...  ...
763      101   63
764      122   27
```

Parte 1.11 – Método rename() - Mudar nome das colunas do Dataframe

Caso deseje mudar os nomes das colunas do dataframe, pode usar a função .rename(). **Este vai ter 2 parâmetros:**

- **columns** e dentro das chavetas curvas, deve indicar o nome da coluna original a alterar e qual o novo nome.
- O argumento **inplace=True** faz com que a modificação de uma operação seja aplicada diretamente ao Dataframe original.

```
#mudar o nome da coluna Age para Idade
data.rename(columns={"Age" : "Idade"}, inplace=True )
```

Se desejar **alterar mais que um elemento**, deve utilizar a vírgula (delimitador que separa as informações). **Para este exemplo, vamos mudar duas colunas específicas:**

- Mudar a coluna com o nome "**Age**" por "**Idade**"
- Mudar a coluna com o nome "**BloodPressure**" por "**Pressão**"

```
#ver informações do dataframe
print(data.head(n=3))

#mudar o nome da coluna Age para Idade
data.rename(columns={"Age" : "Idade", "BloodPressure" : "Pressão"}, inplace=True )

#ver informações do dataframe
print(data.head(n=3))
```

```
[3 rows x 9 columns]
   Pregnancies  Glucose  Pressão  SkinThickness  Insulin   BMI  DiabetesPedigreeFunction  Idade
0            6      148       72             35        0  33.6                     0.627     50
1            1       85       66             29        0  26.6                     0.351     31
2            8      183       64              0        0  23.3                     0.672     32
```

Parte 1.12 – Adicionar colunas do Dataframe

Parte 1.12.1 – Adicionar colunas individualmente e o método assign()

Exemplo 1: Para adicionar uma nova coluna com o valor vazio, basta invocar o nome do dataframe e dentro de chavetas colocar o novo nome e a frente inicializar com o valor a nulo:

```
#adicionar uma nova coluna no dataframe
data["teste"] = 0
print(data)
```

```
DiabetesPedigreeFunction  Age  Outcome  teste
                   0.627   50        1      0
                   0.351   31        0      0
                   0.672   32        1      0
                   0.167   21        0      0
                   2.288   33        1      0
                     ...  ...      ...    ...
                   0.171   63        0      0
```

Exemplo 2: usar a função **assign()** e preencher linhas das informações com dados vazios. No entanto temos de criar uma nova variável para armazenar a nova informação:

```
#adicionar uma nova coluna no dataframe
nova_data = data.assign(Teste = 0)
print(nova_data)
```

```
DiabetesPedigreeFunction  Age  Outcome  teste
                   0.627   50        1      0
                   0.351   31        0      0
                   0.672   32        1      0
                   0.167   21        0      0
                   2.288   33        1      0
                     ...  ...      ...    ...
                   0.171   63        0      0
```

Exemplo 3: Adicionar mais que uma coluna (“Tipo_Sangue” e “Cardinalidade”). Deve utilizar a vírgula entre os novos campos para separar as colunas. No entanto temos de criar uma nova variável para armazenar a nova informação:

```
#Exemplo 2: Adicionar mais que uma coluna
#deve utilizar a virgula entre os novos campos para separar as colunas
nova_data2 = data.assign(Tipo_Sangue="Sem info", Cardinalidade="+/-")
#mostrar a nova dataframe
print(nova_data2.head())
```

```
come  Tipo_Sangue  Cardinalidade
   1     Sem info           +/-
   0     Sem info           +/-
   1     Sem info           +/-
   0     Sem info           +/-
   1     Sem info           +/-
```

Parte 1.12.2 – Método assign() com números aleatórios

Exemplo: Adicionar apenas uma nova coluna ("Tipo_Diabetes") e preencher linhas das informações com dados aleatórios

```
#Exemplo 1: Adicionar apenas uma nova coluna (Tipo_Diabetes)
# e gerar uma lista de numeros aleatorios para preencher
nova_data = data.assign(Tipo_Diabetes = list(np.random.randint(1, 3, len(data))))
#mostrar a nova dataframe
print(nova_data.head())
```

	Pregnancies	Glucose	BloodPressure	...	Age	Outcome	Tipo_Diabetes
0	6	148	72	...	50	1	2
1	1	85	66	...	31	0	2
2	8	183	64	...	32	1	2
3	1	89	66	...	21	0	1
4	0	137	40	...	33	1	1

Parte 1.12.3 – Método assign() e cálculos com base em outras colunas

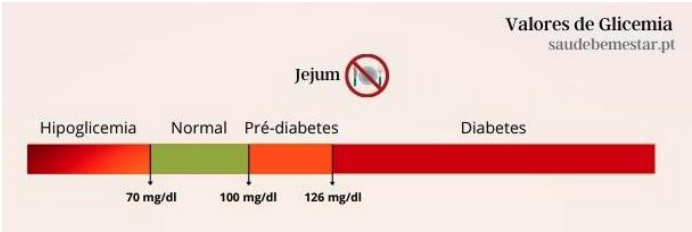
Exemplo: Podemos pegar em dados de outras colunas no Dataframe e realizar cálculos adicionais ou colocar texto. Devem usar a palavra reservada lambda e atribuir uma variável para aceder aos campos da dataframe:

```
nova_data3 = data.assign(Acerto_Glucose=lambda x: x['Glucose'] * 0.10)
#mostrar a nova dataframe
print(nova_data3.head())
```

	Pregnancies	Glucose	BloodPressure	...	Age	Outcome	Acerto_Glucose
0	6	148	72	...	50	1	14.8
1	1	85	66	...	31	0	8.5
2	8	183	64	...	32	1	18.3

Parte 1.13 – Método apply(): Adicionar colunas com base numa função

Para este caso, vamos ter de fazer uma pequena operação. Vamos pegar nos valores da coluna Glucose e vamos quer classificar o valor numérico para o respetivo estado, baseando na seguinte tabela:



Exemplo: Adicionar coluna Estado_Glucose com resultado de uma função

Como tal, vamos desenvolver a seguinte função (com base na tabela anterior):

```
# função para retornar o resultado do valor da glucose
def classificacao_IMC(imc):
    if imc < 70 :
        return "Hipoglicemia (Baixo)"
    elif imc >= 70 and imc <= 100 :
        return "Normal"
    elif imc > 100 :
        return "Hiperglicemia (Alto)"
    else:
        return "erro"
```

Com isto feito, vamos usar a função **.apply()** para conseguir usar a coluna que desejamos trabalhar (neste caso a Glucose) e aplicar a função que foi feita anteriormente).

```
data["Estado_Glucose"] = data["Glucose"].apply(classificacao_IMC)
#mostrar a nova dataframe
print(data.head())
```

Repare que a função não precisa de passar o parâmetro de entrada, pois a função apply() já passa o campo da coluna que estiver a percorrer naquele momento para dentro da função:

Glucose	BloodPressure	...	Age	Outcome	Estado_Glucose
148	72	...	50	1	Hiperglicemia (Alto)
85	66	...	31	0	Normal
183	64	...	32	1	Hiperglicemia (Alto)
89	66	...	21	0	Normal
137	40	...	33	1	Hiperglicemia (Alto)

A partir deste momento, vamos mudar os dados de para o ficheiro nba_duplicados.csv

Como tal, no Jupyter deve criar um novo ficheiro com o nome "jogadores_nba.ipynb" e carregue o ficheiro csv com a biblioteca pandas

Parte 1.14 – Método drop(): Retirar colunas do dataframe

Da mesma forma existe função para adicionar colunas, também podemos remover colunas. Como tal, deve usar a função **.drop()** e **dentro dos parênteses** deve colocar o que deseja eliminar:

Eliminar apenas 1 coluna:

```
nova_data = nova_data.drop(columns="Number")
```

Eliminar apenas 2 colunas:

```
nova_data = nova_data.drop(columns=["Number", "Age"])
```

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Name             457 non-null    object
 1   Team             457 non-null    object
 2   Position         457 non-null    object
 3   Height           457 non-null    object
 4   Weight           457 non-null    float64
 5   College          373 non-null    object
 6   Salary           446 non-null    float64
 7   Aumento_Salario  446 non-null    float64
```

Parte 1.15 – *Função isnull(): Verificação de valores ausentes*

O DataFrame de amostra não tem nenhum valor ausente. Como tal, **vamos pegar num ficheiro com erros de processamento (ficheiro nba.csv)**, mais especificamente, usar a função **.isnull()** para verificar valores ausentes.

```
#Passo 2 - Verificão de valores ausentes
print(data.isnull())
```

Este por sua vez verifica os dados e devolve **true** se encontrar dados vazios:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	True
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	True	False
..	...	...	...	...	...	...	...	...	...

Mas como tal, não queremos exibir a informação neste estado. Seria mais agradável, saber a contagem dos elementos.

Como geralmente é mais útil saber a **quantidade de dados ausentes**, também podemos combinar o método **.isnull()** com **.sum()** para contar o número de nulos em cada coluna:

```
#Passo 2 - Verificar e fazer o sumatorio de valores ausentes
print(data.isnull().sum())
```

```
Name          1
Team          1
Number        1
Position      1
Age           1
Height        1
Weight        1
College      85
Salary       12
```

Como remover os dados nulos?

Uma maneira de lidar com dados ausentes é eliminá-los. Isso é particularmente útil nos casos em que possa ter muitos dados e a perda de uma pequena parte não afetará a análise.

Poderá usar um método .dropna() e este onde detetar que existe campo vazio, não vai mostrar a linha que contém esse registo:

```
#Eliminar dados nulos (e colocar a nova informação na variavel data2)
data2 = data.dropna()
```

Resultado após a utilização do dropna():

```
Name          0
Team          0
Number        0
Position      0
Age           0
Height        0
Weight        0
College       0
Salary        0
dtype: int64
```

Parte 1.16 – *Função duplicated(): Verificação de valores duplicados*

Dentro do DataFrame, podemos ter dados duplicados. Como tal, **vamos pegar num ficheiro com erros de processamento e dados duplicados (ficheiro nba_duplicados.csv)**, mais especificamente, usar a função **.duplicated()** para verificar valores duplicados.

```
#Passo 2 - Verificar e fazer o sumatorio das linhas duplicadas
print(data.duplicated())
```

Este por sua vez verifica os dados e devolve **true** se encontrar dados duplicados:

```
1       False
2        True
3       False
4       False
       ...
458     False
459     False
460     False
```

Mas como tal, não queremos exibir a informação neste estado. Seria mais agradável, saber a contagem dos elementos.

Como geralmente é mais útil saber a **quantidade de dados duplicados**, também podemos combinar o método **.duplicated()** com **.sum()** para contar os registos duplicados:

```
#Passo 2 - Verificar e fazer o sumatorio das linhas duplicadas
print(data.duplicated().sum())
```

```
10
```

Como remover os dados duplicados?

Poderá usar um método .drop_duplicates() e este onde detetar que existe registos duplicados, não vai mostrar a linha que contém esse registo:

```
#Eliminar dados duplicados (e colocar a nova informação na variavel data3)
data3 = data2.drop_duplicates()
```

Parte 1.17 – *Gravar dados em excel (simples ou múltiplas folhas de cálculo)*

Após a realização do trabalho, pode gravar os dados numa folha de cálculo (mas pode ser noutro formato em causa).

Como tal, podemos usar a ***função .to_excel()*** para gravar os dados numa folha de cálculo simples ou gravar várias fontes de dados em múltiplas folhas de cálculo.

Imagine que a variável **data3** tem a informação de **um DataFrame** com todos os tratamentos de informação. Agora podemos invocar a seguir a variável a função ***.to_excel()*** e dentro dos parênteses e aspas, **indicar o caminho (deve clicar com o lado direito do rato em cima da pasta de onde está a trabalhar)** e qual **o nome do ficheiro excel** (não esquecer de dar a extensão **.xlsx** para o sistema assumir como ficheiro excel):

```
#Guardar dados num ficheiro excel (ficheiro simples)
data3.to_excel("Ficha 3 - Funções básicas Pandas/Relatorio_Ficha_3.xlsx")
```

Mas, no entanto, pode ter várias Dataframes com vários tipos de tratamentos de dados. Imagine a seguinte situação:

É pretendido guardar a informação das variáveis que contém as informações:

- **data** → contém as informações originais que recebeu;
- **data2** → variável que foram eliminados os dados nulos;
- **data3** → variável que foram eliminados os dados nulos e duplicados;

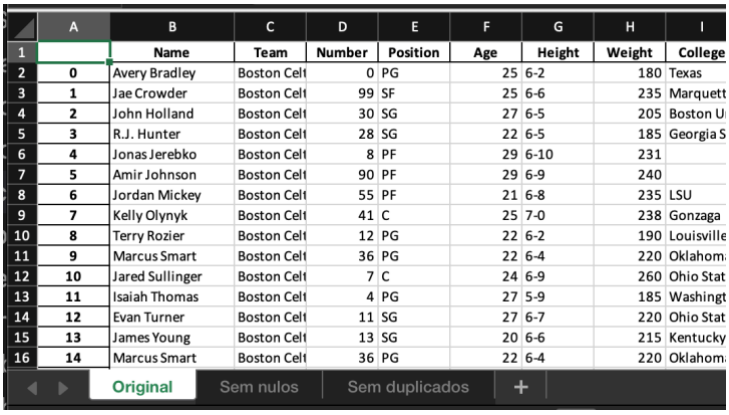
Como tal e quando queremos armazenar vários Dataframes ao mesmo tempo e cada uma das Dataframes separados por folhas de cálculo (para depois realizar comparações de dados e afins), vamos ter de invocar ***outra funcionalidade que já vem embutida no Pandas***.

A funcionalidade ***with pd.ExcelWriter*** permite **indicar o caminho (deve clicar com o lado direito do rato em cima da pasta de onde está a trabalhar)** seguido **do nome do ficheiro excel** (não esquecer de dar a extensão **.xlsx** para o sistema assumir como ficheiro excel) e logo a seguir temos o operador ***as*** para **dar um nome/acrónimo abreviado das instruções escritas mais atrás**, que neste caso é o nome **ficheiro**.

De seguida, basta indicar qual das variáveis DataFrame que deseja trabalhar e de seguida invocar a função ***to_excel()*** e dentro dos parênteses indicar 2 argumentos: o **nome do ficheiro da variável** e qual o **nome dessa folha de cálculo (sheet_name)**:

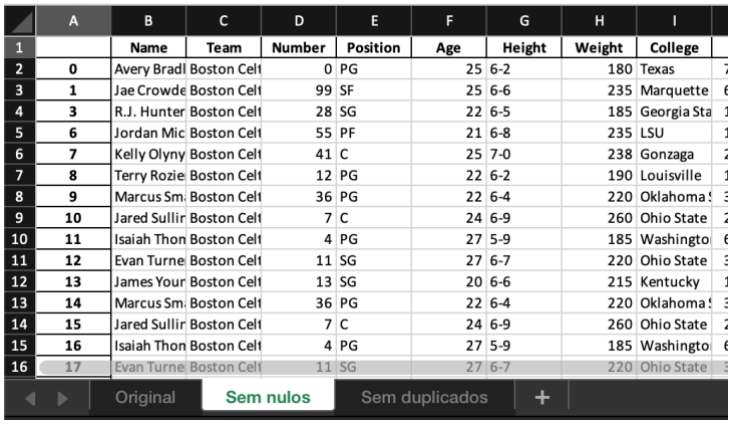
```
#Guardar os dados num ficheiro excel (com várias folhas)
with pd.ExcelWriter("Ficha 3 - Funções básicas Pandas/Relatorio_Ficha_3.xlsx") as ficheiro:
    data.to_excel(ficheiro, sheet_name="Original")
    data2.to_excel(ficheiro, sheet_name="Sem Nulos")
    data3.to_excel(ficheiro, sheet_name="Sem Nulos e Duplicados")
```

Como pode observar, foi criado o ficheiro excel e dentro do mesmo, foram criadas as folhas de cálculo Original, Sem Nulos e Sem Nulos ou Duplicados, do qual, cada uma das folhas tem a representação dos dados de cada variável DataFrame:



		Name	Team	Number	Position	Age	Height	Weight	College
0	0	Avery Bradley	Boston Celt	0	PG	25	6-2	180	Texas
1	1	Jae Crowder	Boston Celt	99	SF	25	6-6	235	Marquett
2	2	John Holland	Boston Celt	30	SG	27	6-5	205	Boston U
3	3	R.J. Hunter	Boston Celt	28	SG	22	6-5	185	Georgia S
4	4	Jonas Jerebko	Boston Celt	8	PF	29	6-10	231	
5	5	Amir Johnson	Boston Celt	90	PF	29	6-9	240	
6	6	Jordan Mickey	Boston Celt	55	PF	21	6-8	235	LSU
7	7	Kelly Olynyk	Boston Celt	41	C	25	7-0	238	Gonzaga
8	8	Terry Rozier	Boston Celt	12	PG	22	6-2	190	Louisville
9	9	Marcus Smart	Boston Celt	36	PG	22	6-4	220	Oklahom
10	10	Jared Sullinger	Boston Celt	7	C	24	6-9	260	Ohio Stat
11	11	Isaiah Thomas	Boston Celt	4	PG	27	5-9	185	Washingt
12	12	Evan Turner	Boston Celt	11	SG	27	6-7	220	Ohio Stat
13	13	James Young	Boston Celt	13	SG	20	6-6	215	Kentucky
14	14	Marcus Smart	Boston Celt	36	PG	22	6-4	220	Oklahom

Original | Sem nulos | Sem duplicados | +



		Name	Team	Number	Position	Age	Height	Weight	College
0	0	Avery Bradl	Boston Celt	0	PG	25	6-2	180	Texas
1	1	Jae Crowde	Boston Celt	99	SF	25	6-6	235	Marquette
2	3	R.J. Hunter	Boston Celt	28	SG	22	6-5	185	Georgia Sta
3	6	Jordan Mic	Boston Celt	55	PF	21	6-8	235	LSU
4	7	Kelly Olyny	Boston Celt	41	C	25	7-0	238	Gonzaga
5	8	Terry Rozie	Boston Celt	12	PG	22	6-2	190	Louisville
6	9	Marcus Sm	Boston Celt	36	PG	22	6-4	220	Oklahoma S
7	10	Jared Sullir	Boston Celt	7	C	24	6-9	260	Ohio State
8	11	Isaiah Thon	Boston Celt	4	PG	27	5-9	185	Washingto
9	12	Evan Turne	Boston Celt	11	SG	27	6-7	220	Ohio State
10	13	James Your	Boston Celt	13	SG	20	6-6	215	Kentucky
11	14	Marcus Sm	Boston Celt	36	PG	22	6-4	220	Oklahoma S
12	15	Jared Sullir	Boston Celt	7	C	24	6-9	260	Ohio State
13	16	Isaiah Thon	Boston Celt	4	PG	27	5-9	185	Washingto
14	17	Evan Turne	Boston Celt	11	SG	27	6-7	220	Ohio State

Original | Sem nulos | Sem duplicados | +

		Name	Team	Number	Position	Age	Height	Weight	College
0	0	Avery Bradl	Boston Celt	0	PG	25	6-2	180	Texas
1	1	Jae Crowde	Boston Celt	99	SF	25	6-6	235	Marquett
2	3	R.J. Hunter	Boston Celt	28	SG	22	6-5	185	Georgia S
3	6	Jordan Mic	Boston Celt	55	PF	21	6-8	235	LSU
4	7	Kelly Olyny	Boston Celt	41	C	25	7-0	238	Gonzaga
5	8	Terry Rozie	Boston Celt	12	PG	22	6-2	190	Louisville
6	9	Marcus Sm	Boston Celt	36	PG	22	6-4	220	Oklahom
7	10	Jared Sullir	Boston Celt	7	C	24	6-9	260	Ohio Stat
8	11	Isaiah Thon	Boston Celt	4	PG	27	5-9	185	Washingt
9	12	Evan Turne	Boston Celt	11	SG	27	6-7	220	Ohio Stat
10	13	James Your	Boston Celt	13	SG	20	6-6	215	Kentucky
11	19	Tyler Zeller	Boston Celt	44	C	26	7-0	253	North Ca
12	21	Markel Bro	Brooklyn N	22	SG	24	6-3	190	Oklahom
13	22	Wayne Ellir	Brooklyn N	21	SG	28	6-4	200	North Ca
14	23	Rondae Ho	Brooklyn N	24	SG	21	6-7	220	Arizona

Original | Sem nulos | Sem duplicados | +