FICHA TRABALHO 3

TRATAMENTO DA INFORMAÇÃO

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 para tratar a manipulação dos dados e tratamento da informação

Parte 1 – Funções tratamento de dados com dataframes

Para esta parte deve carregar o ficheiro **dados.csv**

```
#Importar a biblioteca pandas
import pandas as pd

#Carregar o ficheiro com os dados dos clientes
data = pd.read_csv("dados.csv")

#verificar os dados que forma carregados
print(data.info())
```

Como podem ver ao longo do ficheiro, temos alguns erros significativos tais como nomes mal preenchidos (com símbolos de reticências, barras, entre outras situações).

	Nomes	Idades	Locais
0	Anabela	31	Braga
1	_Ana123	27	Porto
2	Car los	24	Viseu
3	rita	22	Coimbra
4	zeLIA	32	faro
5	Luis_	17	Beja
6	Ri_cardo	14	Guarda
7	INES_	12	Lisboa
8	Julia	25	Porto

1 - Função str()

Função **str()** de string (texto em português), oferece várias operações de tratamento de informação textual para trabalhar em ciência de dados, especialmente, quando queremos fazer a mesma operação para vários tratamentos de dados.

1.1 - Função str.upper()

Converter texto em letras maiúsculas para melhor consistência (por exemplo, converter nomes ou categorias em maiúsculas).

```
print("Função .upper():")
data_upper = data["Nomes"].str.upper()
print(data_upper)
```

```
Função .upper():
0     ANABELA
1     _ANA123
2     CAR LOS
3        RITA
4       ZELIA
5       LUIS_
6    RI_CARDO
7       INES_
8       JULIA
```

1.2 - Função str.lower()

Faz o contrário do exercício anterior, convertendo o texto em letras minúsculas.

```
print("Função .lower():")
data_lower = data["Nomes"].str.lower()
print(data_lower)
```

```
Função .lower():
0     anabela
1     _ana123
2     car los
3        rita
4       zelia
5       luis_
6    ri_cardo
7       ines_
8       julia
```

1.3 - Função str.isupper()

Verifica e retorna valores booleanos se encontra texto todo em maiúsculas ou não.

```
print("Função .isupper():")
data_isupper = data["Nomes"].str.isupper()
print(data_isupper)
```

```
Função .isupper():
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7     True
8    False
Name: Nomes, dtype: bool
```

1.4 - Função str.islower()

Faz o contrário do exercício anterior, verificando se encontra texto todo em minúsculas ou não.

```
print("Função .islower():")
data_islower = data["Nomes"].str.islower()
print(data_islower)
```

```
Função .islower():
0    False
1    False
2    False
3     True
4    False
5    False
6    False
7    False
8    False
Name: Nomes, dtype: bool
```

1.5 - Função str.len()

Devolve qual a quantidade de caracteres (comprimento) de cada texto.

```
print("Função .len():")
data_len = data["Nomes"].str.len()
print(data_len)
```

```
Função .len():
0    7
1    7
2    7
3    4
4    5
5    5
6    8
7    5
8    5
```

1.6 - Função str.startswith()

Retorna valores booleanos com base se a expressão **começa com uma determinada sequência de caracteres** ou não. Atenção, pois o python é case-sensitive (sensível com letras maiúsculas e minúsculas)

```
#apenas uma letra/expressão
print("Função .startswith() mais do que uma letra/expressão:")
data_startswith = data["Nomes"].str.startswith("A")
print(data_startswith)

#mais do que uma letra/expressão
print("Função .startswith() mais do que uma letra/expressão:")
data_startswith = data["Nomes"].str.startswith(("A", "L"))
print(data_startswith)
```

```
Função .startswith() apenas uma letra/expressão:
0     True
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
```

```
Função .startswith() mais do que uma letra/expressão:
0     True
1    False
2    False
3    False
4    False
5     True
6    False
7    False
8    False
```

1.7 - Função str.endswith()

Retorna valores booleanos com base se a expressão **termina com uma determinada sequência de caracteres** ou não. Atenção, pois o python é case-sensitive (sensível com letras maiúsculas e minúsculas)

```python
#apenas uma letra/expressão
print("Função .endswith() apenas uma letra/expressão:")
data_endswith = data["Nomes"].str.endswith("a")
print(data_endswith)

#mais do que uma letra/expressão
print("Função .endswith() mais do que uma letra/expressão:")
data_endswith = data["Nomes"].str.endswith(("a", "o"))
print(data_endswith)
```

```
Função .endswith() apenas uma letra/expressão:
0     True
1    False
2    False
3     True
4    False
5    False
6    False
7    False
8     True
```

```
Função .endswith() mais do que uma letra/expressão:
0     True
1    False
2    False
3     True
4    False
5    False
6     True
7    False
8     True
```

1.8 - Função str.find()

Esta função faz a pesquisa de um carácter ou sequência de caracteres, e se encontrar alguma coisa, devolve a posição de onde está a informação. Ou devolve a posição ou então -1 (não encontrou nada).

```python
print("Função Find com a letra 'o':")
data_find = data["Nomes"].str.find('o')
print(data_find)
```

```
Função Find com a letra 'o':
0   -1
1   -1
2    5
3   -1
4   -1
5   -1
6    7
7   -1
8   -1
Name: Nomes, dtype: int64
```

1.11 - Função str.count()

Esta função retorna a contagem de quantos elementos existem no DataFrame, do qual, temos de indicar o carácter ou a expressão a encontrar. Neste caso, retorna o total de letras 'a' por cada registo.

```
print("Função count():")
data_count = data["Nomes"].str.count('a')
print(data_count)
```

```
Função count():
0    2
1    1
2    1
3    1
4    0
5    0
6    1
7    0
8    1
Name: Nomes, dtype: int64
```

1.9 - Função str.strip()

A função strip() é algo que temos de ter cuidado. Como tal, vamos sempre indicar qual a coluna que desejamos trabalhar, antes de trabalhar com a função.

A função strip() tem algumas funcionalidades adicionais tais como:

- strip() -> apenas retira espaços em branco do início e do fim;
- lstrip() -> limpa as irregularidades que estão à esquerda da informação;
- rstrip() -> limpa as irregularidades que estão à direita da informação;

```
print("Função strip (e variantes):")
data["Nomes"] = data["Nomes"].str.strip()
data["Nomes"] = data["Nomes"].str.lstrip("...")
data["Nomes"] = data["Nomes"].str.lstrip("/")
data["Nomes"] = data["Nomes"].str.rstrip("_")
print(data["Nomes"])
```

```
Função strip (e variantes):
0     Anabela
1      Ana123
2     Car los
3        rita
4       zÉlIA
5        Luis
6    Ri_cardo
7        INÊS
8       Júlia
Name: Nomes, dtype: object
```

Mas, também podemos eliminar todos os casos de uma só vez com a função strip(). Como tal, vamos correr o seguinte comando e excluir os outros símbolos e os números 123.

```
data["Nomes"] = data["Nomes"].str.strip("123._/")
print(data["Nomes"])
```

1.10 - Função str.replace()

Esta função ajuda a remover certas sequências de caracteres às vezes que estão presentes em todas as cadeias e são indesejáveis.

```
print("Função replace():")
data["Nomes"] = data["Nomes"].str.replace(' ','')
data["Nomes"] = data["Nomes"].str.replace('_','')
print(data["Nomes"])
```

```
Função replace():
0    Anabela
1     Ana123
2     Carlos
3       rita
4      zÉlIA
5       Luis
6    Ricardo
7       INÊS
8      Júlia
Name: Nomes, dtype: object
```

Parte 2 – Funções tratamento de dados avançado com dataframes

Para esta parte deve **carregar o ficheiro Lista_Clientes.csv**

```
#Importar a biblioteca pandas
import pandas as pd

#Carregar o ficheiro com os dados dos clientes
data = pd.read_csv("Lista_Clientes.csv", sep=";")
#verificar os dados que forma carregados
print(data.info())
#mostrar os dados
print(data)
```

Como podem ver ao longo do ficheiro, temos alguns erros significativos tais como nomes mal preenchidos (com símbolos de reticencias, barras, entre outros números de telefone mal preenchidos).

	CustomerID	First_Name	Last_Name	...	Paying Customer	Do_Not_Contact	Not_Useful_Columr
0	1001	Frodo	Baggins	...	Yes	No	VERDADEIRO
1	1002	Abed	Nadir	...	No	Yes	FALSO
2	1003	Walter	/White	...	N	NaN	VERDADEIRO
3	1004	Dwight	Schrute	...	Yes	Y	VERDADEIRO
4	1005	Jon	Snow	...	Y	No	VERDADEIRO
5	1006	Ron	Swanson	...	Yes	Yes	VERDADEIRO
6	1007	Jeff	Winger	...	No	No	FALSO
7	1008	Sherlock	Holmes	...	N	No	FALSO
8	1009	Gandalf	NaN	...	Yes	NaN	FALSO

Exercício 1 – Vamos verificar se existe dados duplicados (fazendo o somatório). Se tiver, deve remover os dados duplicados com a função **drop_duplicates()**;

Exercício 2 – Vamos verificar se existe dados nulos (fazendo o somatório). Se tiver, deve remover os dados nulos com a função **dropna()**;

Exercício 3 – Eliminar colunas desnecessárias, neste caso, a coluna "**Not_Useful_Column**";

Exercício 4 – Os campos **CostumerID** e **First_Name** parecem ter os dados em condições, mas, quando passamos para o campo **Last_Name**, temos algumas irregularidades:

- reticências atras dos nomes;
- underscores (_) antes e depois dos nomes;
- barras invertidas;
- entre outros pontos;

A função strip() é algo que temos de ter cuidados. Se não especificar a coluna que seja trabalhar, pode fazer as mesmas ações para as outra colunas. Como tal, vamos sempre indicar qual a coluna que desejamos trabalhar, antes de trabalhar com a função.

A função strip() tem algumas funcionalidades adicionais tais como:

- strip() -> apenas retira espaços em branco do início e do fim
- lstrip() -> limpa as irregularidades que estão à esquerda da informação
- rstrip() -> limpa as irregularidades que estão à direita da informação

Exercício 4.1 – Use as funções lstrip() e rstrip() para remover irregularidade atras dos nomes como por exemplo reticências, barras invertidas, underscores.

P.S.: Só queremos apenas alterar os dados da coluna. Coloque atrás do nome da coluna a trabalhar e depois a frente é que colocamos o igual e a operação a trabalhar.

Exercício 4.2 – Use a função strip() para remover todos os símbolos e os números 123.

Exercício 5 – Funções para tratar dos dados telefónicos.

Exercício 5.1 – Função replace() - vamos tratar os dados dos contatos telefónicos (colocar apenas números). Se reparar nos dados, estes estão muito inconsistentes e queremos colocar o formato de apenas números

Como tal vamos utilizar a função **replace** permite uma funcionalidade adicional chamada de Regex (expressões regulares). No primeiro parâmetro podemos colocar apenas uma instrução ou combinação de várias instruções e arranjar os dados sem que:

- \d – devolve tudo o que não são dígitos;
- \D – devolve os dígitos equivalente no intervalo entre 0 até 9.
- \s – devolve tudo o que não são espaços;
- \S – devolve espaços;

```
data["Phone_Number"] = data["Phone_Number"].str.replace('\D', '', regex=True)
print(data["Phone_Number"])
```

Exercício 6 – Função Split() - Tratar moradas (coluna Address)

O objetivo será dividir a morada em partes. Muitas vezes as pessoas utilizam virgulas ou outros símbolos para separar informações, como rua, número da porta, concelhos, cidade, código-postal, país, entre outras situações.

Neste ficheiro, os dados das moradas estão separados por virgulas (,). Vamos usar a função split para separar o texto em porções e verificar a resposta.

A função split() tem 3 parâmetros em causa:

- **Parâmetro 1** → qual o símbolo a encontrar (neste caso estamos a assumir a ,);
- **Parâmetro 2** → qual o limite de repetições para fazer a operação (pois temos moradas com mais de uma vírgula);
- **Parâmetro 3** → se permite expandir a operação do split nas diferentes colunas (como indicados no início da expressão);

```
data[["Street", "State", "Zip Code"]] = data["Address"].str.split(',', n=2, expand=True)
print(data.head())
```

Exercício 7 – Funções para colocar valores com o mesmo formato

Se analisar a coluna **Paying Customer** ou **Do_Not_Contact** ambos tem irregularidade de informação, isto é, alguns dados têm Y outros Yes e o oposto N outros NO. O objetivo será fazer uma regra para colocar tudo com o mesmo formato.

Exercício 7.1 – Primeiro convertemos as palavras extensas para letra simples e fica tudo igual:

```
data["Paying Customer"] = data["Paying Customer"].str.replace('Yes', 'Y')
print(data["Paying Customer"])
```

Faça o mesmo para o No:

```
data["Paying Customer"] = data["Paying Customer"].str.replace('No', 'N')
print(data["Paying Customer"])
```

Pode continuar com as mesmas operações para a coluna “Do_Not_Contact”:

```
data["Do_Not_Contact"] = data["Do_Not_Contact"].str.replace('Yes', 'Y')
data["Do_Not_Contact"] = data["Do_Not_Contact"].str.replace('No', 'N')
print(data["Do_Not_Contact"])
```

Exercício 8 – Gere um novo ficheiro excel com os dados tratados.