

# Introdução ao Python

MBA USP/ESALQ

Anotações por Lucas Rezende

## Manipulação de base de dados

Utilização do numpy e pandas

### Importando dados de um arquivo .csv

```
In [1]: import pandas as pd
import numpy as np

df = pd.read_csv('(2) notas_pisa.csv', sep=";", decimal=".") # lê um arquivo .csv
df['country'] # podemos nos referenciar a uma coluna do data frame como se fosse u
```

```
Out[1]: 0          Australia
1          Austria
2          Belgium
3          Canada
4          Chile
...
91  Ukraine (18 of 27 Regions)
92      United Arab Emirates
93          Uruguay
94      Uzbekistan
95      Viet Nam
Name: country, Length: 96, dtype: object
```

```
In [2]: df['country'][0]
```

```
Out[2]: 'Australia'
```

`read_csv` has many parameters that allow you to customize how the CSV is read. Some of the most commonly used ones are:

- `sep`: Specify a custom delimiter, e.g., `sep=';'` if your CSV uses semicolons.
- `header`: Specify which row to use as the column names. For example, `header=0` uses the first row.
- `index_col`: Specify which column to use as the row labels of the DataFrame. For example, `index_col=0` uses the first column.
- `usecols`: Specify a subset of columns to read. For example, `usecols=['col1', 'col2']`.

- dtype: Specify the data type for the columns. For example, dtype={'col1': int, 'col2': float}. na\_values: Specify additional strings to recognize as NA/NaN. For example, na\_values=['NA', '?'].

## Importando dados de um arquivo xlsx

```
In [3]: df = pd.read_excel('(2) receita_empresas.xlsx') # podemos utilizar a função read_e
```

## Criação de Data Frames

Podemos criar o nosso próprio data frame, seguindo a estrutura de um dicionário:

```
In [4]: varA = pd.Series(['a', 'b', 'c', 'd', 'e', 'f']) # criando uma série, que será uma colu
varB = pd.Series([1, 2, 3, 4, 5, None])
varC = np.arange(1, 7) # podemos utilizar o numpy também

df2 = pd.DataFrame({'varA': varA, # criando o Dataframe no estilo de um dicionário
                    'varB': varB,
                    'varC': varC})

df2
```

```
Out[4]:
```

	varA	varB	varC
0	a	1.0	1
1	b	2.0	2
2	c	3.0	3
3	d	4.0	4
4	e	5.0	5
5	f	NaN	6

```
In [5]: df3 = pd.DataFrame({'VarA': ['Brasil', 'Estados Unidos', 'Peru', 'Colômbia', 'Urugu
      'VarB': [1, 2, 3, 4, 5],
      'varC': ['Inativo', 'Ativo', 'Ativo', None, 'Ativo']})

df3
```

```
Out[5]:
```

	VarA	VarB	varC
0	Brasil	1	Inativo
1	Estados Unidos	2	Ativo
2	Peru	3	Ativo
3	Colômbia	4	None
4	Uruguai	5	Ativo

# Salvando Data Frames (Exportar base de dados)

## Salvando em CSV

Utilizamos a função `to_csv`

- `path_or_buf`: String or file handle, default `None`. The file path or object to write. If not specified, the result is returned as a string.
- `sep`: String, default `'.'`. The delimiter to use. For example, `sep='\t'` for tab-separated values.
- `na_rep`: String, default `''`. The string representation of missing values.
- `float_format`: String, default `None`. Format string for floating-point numbers. For example, `float_format='%.2f'` will format numbers with two decimal places.
- `columns`: Sequence, optional. Columns to write. If not specified, all columns are written.
- `header`: Boolean or list of strings, default `True`. Whether to write column names. If a list of strings is given, it is used as the column names.
- `index`: Boolean, default `True`. Whether to write row names (index).
- `index_label`: String or sequence, default `None`. Column label for the index column(s). If `None`, the index names are used.
- `mode`: String, default `'w'`. The file mode to use (`'w'` for write, `'a'` for append).
- `encoding`: String, default `None`. The encoding to use for the output file, e.g., `encoding='utf-8'`.
- `compression`: String or dict, default `None`. For on-the-fly compression of the output file. Possible values: `'infer'`, `'gzip'`, `'bz2'`, `'zip'`, `'xz'`, or a dict with specific options.
- `quoting`: Constant from `csv` module, default `csv.QUOTE_MINIMAL`. Controls when quotes should be generated by the CSV writer.
- `quotechar`: String, default `''''`. Character to use for quoting fields.
- `line_terminator`: String, default `None`. The newline character or sequence to use in the output file.
- `chunksize`: Integer, default `None`. Write file in chunks of this size.
- `date_format`: String, default `None`. Format string for datetime objects.
- `doublequote`: Boolean, default `True`. Control whether quoting is done with double quotes.
- `escapechar`: String, default `None`. Character to use for escaping the `sep` and `quotechar` if quoting is set to `csv.QUOTE_NONE`.
- `decimal`: String, default `'.'`. Character recognized as decimal separator. E.g., use `','` for European data.

```
In [6]: df3.to_csv('Data Frame 3.csv', index=False) # essa função salva o arquivo em CSV
# index = False diz para não escrever os índices no arquivo.
```

## Salvando em xlsx

Utilizamos a função `to_excel`

### Key Parameters

- `excel_writer`: This can be a string representing the file path, a `pandas.ExcelWriter` object, or a file-like object.
- `sheet_name`: Specifies the name of the sheet where the DataFrame will be written. The default is 'Sheet1'.
- `na_rep`: String representation for missing values. For example, `na_rep='N/A'`.
- `float_format`: Format string for floating-point numbers. For example, `float_format='%.2f'`.
- `columns`: A list of column names to write. If not specified, all columns are written.
- `header`: Boolean or a list of strings. If True, the column names are written. If False, no header is written. You can also pass a list of column names to be used as the header.
- `index`: Boolean. If True, the index is written to the file. The default is True.
- `index_label`: A string or a sequence. The column label(s) for the index column(s). If None, the index name(s) are used.
- `startrow`: The upper left cell row to start writing the DataFrame data. The default is 0.
- `startcol`: The upper left cell column to start writing the DataFrame data. The default is 0.
- `engine`: Specify the Excel writer engine to use. Options are None, 'xlwt', 'xlsxwriter', 'openpyxl', 'odf', 'pyxlsb'. The default is None, meaning pandas will choose the appropriate engine based on the file extension.
- `merge_cells`: Boolean. Write MultiIndex and Hierarchical Rows as merged cells. The default is True.
- `encoding`: The encoding of the output file. The default is utf-8.
- `inf_rep`: Representation for infinity values (inf). The default is inf.
- `verbose`: If True, prints additional information. The default is False.
- `freeze_panes`: A tuple (row, col) for the topmost row and leftmost column to be frozen. For example, `freeze_panes=(1, 1)`.

```
In [7]: df2.to_excel('Data Frame 2.xlsx', index=False)
```

## Aula III - Introdução ao Python

### Métodos do dataframe

#### `df.info()`

Traz informações sobre o objeto em questão

```
In [8]: import pandas as pd
import numpy as np
```

```
df = pd.read_csv('(2) notas_pisa.csv', delimiter=",")
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   country               96 non-null    object
1   group                 96 non-null    object
2   mathematics_2022      96 non-null    object
3   reading_2022          96 non-null    object
4   science_2022          96 non-null    object
5   mathematics_2018      96 non-null    object
6   reading_2018          96 non-null    object
7   science_2018          96 non-null    object
dtypes: object(8)
memory usage: 6.1+ KB
None
```

## df.iloc()

Localiza um elemento com base na posição, as contagens se iniciam no 0, tanto em linha quanto coluna

```
In [9]: df.iloc[0, 3]; # imprime o elemento da linha 0, coluna 3 (Linha 1 coluna D)
```

```
In [10]: df.iloc[0:7]; # imprime da linha 0 a 7 (Linha 1 a 8)
```

```
In [11]: df.iloc[:, [0, 2, 5]]; # imprime todas as linhas mostrando apenas as colunas 0, 2,
```

```
In [12]: df.iloc[:, 0:3]; # imprime todas as linhas, das colunas 0 até 2 (A até C)
```

## df.reindex()

Troca a ordem dos índices da base de dados

## pd.to\_numeric(df['coluna'])

Transforma a coluna em dados números

**arg** : O argumento a ser convertido. Pode ser uma lista, série, ou um array-like (incluindo colunas de um DataFrame).

**errors** : Define como lidar com erros durante a conversão. Pode assumir três valores:

- raise: (padrão) Levanta uma exceção se houver um erro na conversão.
- coerce: Converte os valores que não puderem ser convertidos para NaN.
- ignore: Retorna o argumento original sem alterações se houver um erro.

**downcast** : Uma tentativa de converter valores para o tipo numérico de menor tamanho possível, para economizar memória. Pode assumir os valores:

- integer: Converte para o tipo de inteiro de menor tamanho possível.
- signed: Converte para o tipo de inteiro assinado de menor tamanho possível.
- unsigned: Converte para o tipo de inteiro não assinado de menor tamanho possível.
- float: Converte para o tipo de ponto flutuante de menor tamanho possível.

```
In [13]: df['mathematics_2018'] = pd.to_numeric(df['mathematics_2018'], errors='coerce')
df['reading_2018'] = pd.to_numeric(df['reading_2018'], errors='coerce')
df['science_2018'] = pd.to_numeric(df['science_2018'], errors='coerce')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   country                96 non-null    object
1   group                  96 non-null    object
2   mathematics_2022       96 non-null    object
3   reading_2022           96 non-null    object
4   science_2022           96 non-null    object
5   mathematics_2018       78 non-null    float64
6   reading_2018           77 non-null    float64
7   science_2018           78 non-null    float64
dtypes: float64(3), object(5)
memory usage: 6.1+ KB
```

Repare como que a coluna de dados mathematics\_2018 contém dados do tipo float

## df.drop()

Podemos excluir dados utilizando linhas ou colunas como parâmetros, bem como o índice

```
In [14]: df.drop(df.index[18:96]); # excluindo todos os dados da linha 18 até a linha 95
```

## df.dropna()

Exclui observações que continham valores faltantes

```
In [15]: df_na = df.dropna()
df_na;
```

## df.describe()

Extraí estatísticas descritivas das variáveis métricas

```
In [16]: df[['mathematics_2018', 'reading_2018', 'science_2018']].describe()
```

```
Out[16]:
```

	mathematics_2018	reading_2018	science_2018
count	78.000000	77.000000	78.000000
mean	458.671675	453.101877	457.915065
std	56.456544	53.131971	51.941188
min	325.100547	339.691591	335.629947
25%	417.470322	412.295085	417.326482
50%	467.964344	465.950148	468.155673
75%	499.941059	498.279257	498.277673
max	591.393598	555.236244	590.452641

## df.value\_counts()

Conta os valores em uma determinada coluna

```
In [17]: df['group'].value_counts()
```

```
Out[17]: group
PARTNERS    58
OECD        38
Name: count, dtype: int64
```

## Filtros

É possível filtrar observações por meio dos operadores: Alguns operadores úteis para realizar filtros:

- == igual
- > maior
- >= maior ou igual
- < menor
- <= menor ou igual
- != diferente
- & indica e
- | indica ou

```
In [18]: df[df['mathematics_2018'] > 430]; # filtra todas as linhas que contém notas de mate
df[df['mathematics_2018'] > 430][0:4] # podemos mostrar apenas os primeiros 4 resul
```

```
Out[18]:
```

	country	group	mathematics_2022	reading_2022	science_2022	mathematics_2022
0	Australia	OECD	487.08425354734698	498.05093978372099	507.000869417628	4
1	Austria	OECD	487.26749908879401	480.40584721829902	491.27095877058503	4
2	Belgium	OECD	489.48681680184899	478.852668384217	490.57834610681402	5
3	Canada	OECD	496.94789438732198	507.13289582796602	515.01667592948297	5

```
In [19]: df[df['group'] == 'OECD']; # mostra apenas os valores do data frame que tem OECD na
```

## Mesclando critérios

```
In [20]: df['science_2022'] = pd.to_numeric(df['science_2022'], errors = 'coerce');
df[(df['group'] == 'OECD') & (df['science_2022'] <= 493)]; # utilizando a notação
df[(df['group'] == 'OECD') | (df['science_2022'] <= 493)]; # utilizando a notação
```

## Agrupando banco de dados

### df.groupby(by['coluna'])

```
In [21]: df_agrupado = df.groupby(by=['group']) # df_agrupado não pode ser manipulado da mesma forma
df_agrupado.describe()
```

```
Out[21]:
```

	count	mean	std	min	25%	50%	75%
group							
OECD	37.0	484.646614	30.989428	409.886598	477.463366	491.270959	499.964883
PARTNERS	44.0	415.147382	52.110795	347.104162	375.607585	406.984551	433.148079

2 rows × 32 columns

## Transpondo uma tabela de dados

```
In [22]: df_agrupado.describe().T # podemos transpor a tabela
```



Out[22]:

	group	OECD	PARTNERS
science_2022	count	37.000000	44.000000
	mean	484.646614	415.147382
	std	30.989428	52.110795
	min	409.886598	347.104162
	25%	477.463366	375.607585
	50%	491.270959	406.984551
	75%	499.964883	433.148079
	max	546.634453	561.433275
mathematics_2018	count	38.000000	40.000000
	mean	486.998322	431.761360
	std	33.563272	60.824943
	min	390.932273	325.100547
	25%	481.241544	394.197636
	50%	495.766955	421.875066
	75%	507.878070	451.291620
	max	526.973250	591.393598
reading_2018	count	37.000000	40.000000
	mean	485.487406	423.145262
	std	27.108257	53.920935
	min	412.295085	339.691591
	25%	473.974317	391.847911
	50%	491.800785	413.926415
	75%	503.928109	441.658446
	max	523.017018	555.236244
science_2018	count	38.000000	40.000000
	mean	486.740274	430.531117
	std	28.796369	54.416865
	min	413.322993	335.629947
	25%	475.460431	397.513395
	50%	492.807054	424.917405

	group	OECD	PARTNERS
75%	503.285090	444.052998	
max	530.108005	590.452641	

## df.sort\_values(by=['coluna'])

In [23]: `df['mathematics_2018'].sort_values()[0: 5] # podemos ordenar apenas uma coluna`

Out[23]:

55	325.100547
80	352.567068
77	352.846256
65	365.884947
74	367.727353

Name: mathematics\_2018, dtype: float64

In [24]: `df.sort_values(by=['mathematics_2018'])[0: 5] # ordenando a base de dados inteira`

Out[24]:

	country	group	mathematics_2022	reading_2022	science_2022	mathematics_2018
55	Dominican Republic	PARTNERS	339.10724671520001	351.31228660018098	360.426060	325.100547
80	Philippines	PARTNERS	354.71971267011003	346.547387252531	356.167293	352.567068
77	Panama	PARTNERS	356.57350389273603	391.953355235988	387.767624	352.846256
65	Kosovo	PARTNERS	354.96244390817299	342.19442056258799	357.024052	365.884947
74	Morocco	PARTNERS	364.766240669848	339.36492493384299	365.398076	367.727353

In [25]: `df.sort_values(by=['mathematics_2018'], ascending= False)[0:5] # Aqui eu não colocaria o nome da coluna`

Out[25]:

	country	group	mathematics_2022	reading_2022	science_2022	mathematics_2018
49	B-S-J-Z (China)	PARTNERS	—	—	NaN	551.92315308368097
86	Singapore	PARTNERS	574.663819557029	542.55332240961297	561.433275	561.433275
67	Macao (China)	PARTNERS	551.92315308368097	510.40512187447899	543.096281	551.92315308368097
59	Hong Kong (China)	PARTNERS	540.35180120700602	499.701340798594	520.418680	551.92315308368097
52	Chinese Taipei	PARTNERS	547.09416419952095	515.16728997095902	537.380381	551.92315308368097

## Renomeando o data frame

```
In [26]: nomes = ["pais",
                  "grupo",
                  "matematica_2022",
                  "leitura_2022",
                  "ciencias_2022",
                  "matematica_2018",
                  "leitura_2018",
                  "ciencias_2018"]

df.columns = nomes # mudando o nome das variáveis

df[0:1]
```

```
Out[26]:
```

	pais	grupo	matematica_2022	leitura_2022	ciencias_2022	matematica_2018
0	Australia	OECD	487.08425354734698	498.05093978372099	507.000869	491.3600

```
In [27]: df.columns.array[1] = 'grupos_países' # mudando o nome de apenas uma coluna
```

## Visualização de dados

```
In [28]: #!pip install matplotlib;
         #!pip install seaborn;
         #!pip install plotly;
```

```
In [29]: import matplotlib.pyplot as plt
         import seaborn as sns
         import plotly.express as px
         import plotly.io as pio
         pio.renderers.default = 'browser'
         import plotly.graph_objects as go
```

## seaborn

```
In [30]: comercio = pd.read_excel('(2) comercio_global.xlsx')
```

### seaborn.countplot(data = df, x= 'coluna')

Consegue contar dados categóricos e fazer um gráfico de contagens

Parameters

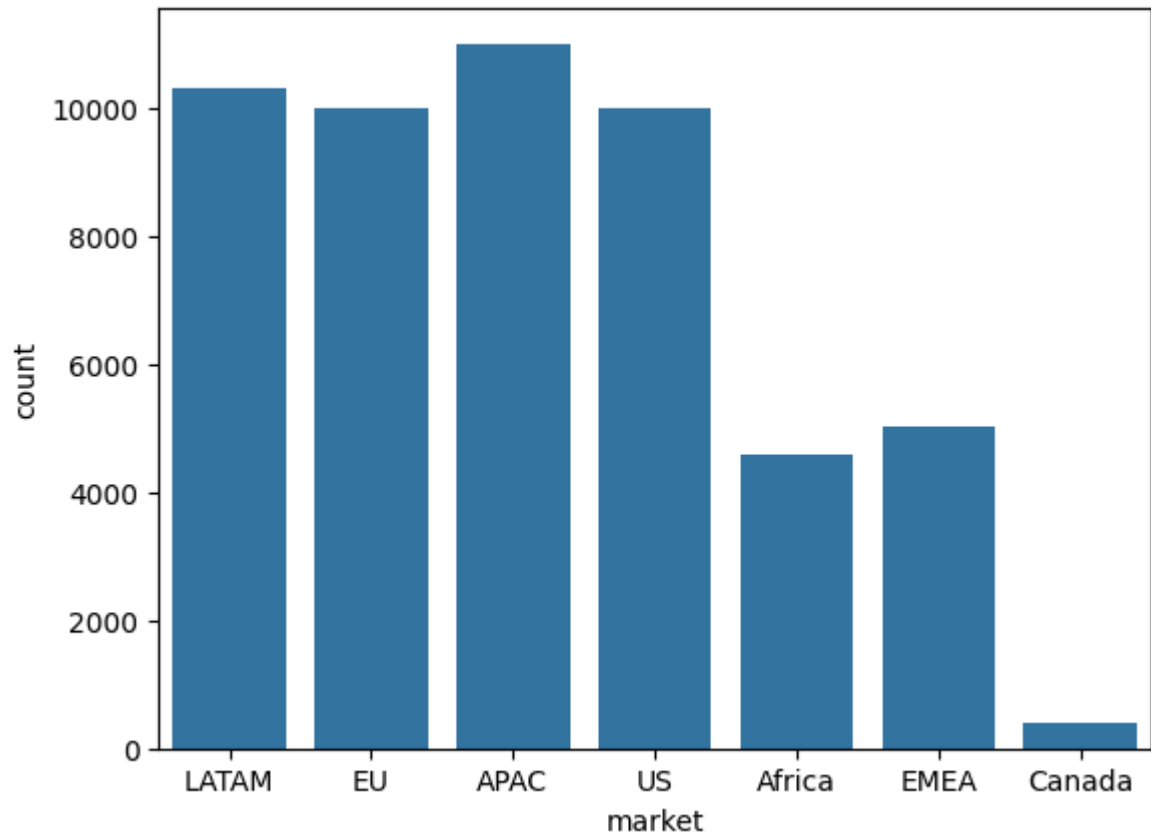
- **x, y** : Names of variables in data or vector data, optional. One of these variables should be specified to determine the position of the bars.
- **hue** : Name of variable in data or vector data, optional. If specified, will produce a multi-plot grid by different levels of the hue variable.
- **data** : DataFrame, array, or list of arrays, optional. The data source.

- `order` : List of strings, optional. Order to plot the categorical levels in; otherwise, the levels are inferred from the data objects.
- `hue_order` : List of strings, optional. Order to plot the levels of the hue variable in; otherwise, the levels are inferred from the data objects.
- `orient` : "v" or "h", optional. Orientation of the plot (vertical or horizontal). This is usually inferred based on the type of the input variables, but it can be used to resolve ambiguity when both x and y are numeric or when plotting wide-form data.
- `color` : Matplotlib color, optional. Color for all elements, or seed for a gradient palette.
- `palette` : Palette name, list, or dict, optional. Colors to use for the different levels of the hue variable.
- `saturation` : Float, optional. Proportion of the original saturation to draw colors at.
- `dodge` : bool, optional. When hue nesting is used, whether elements should be shifted along the categorical axis.
- `ax` : Matplotlib Axes, optional. Axes object to draw the plot onto, otherwise uses the current Axes.
- `kwargs` : Other keyword arguments are passed through to `seaborn.catplot`

**Dica:** `hue` é utilizado para produzir plots divididos em categorias. Por exemplo, produzir plots de contagens de frequência de alunos pelo dia da semana (variável x) separados por gênero (`hue='gênero'`)

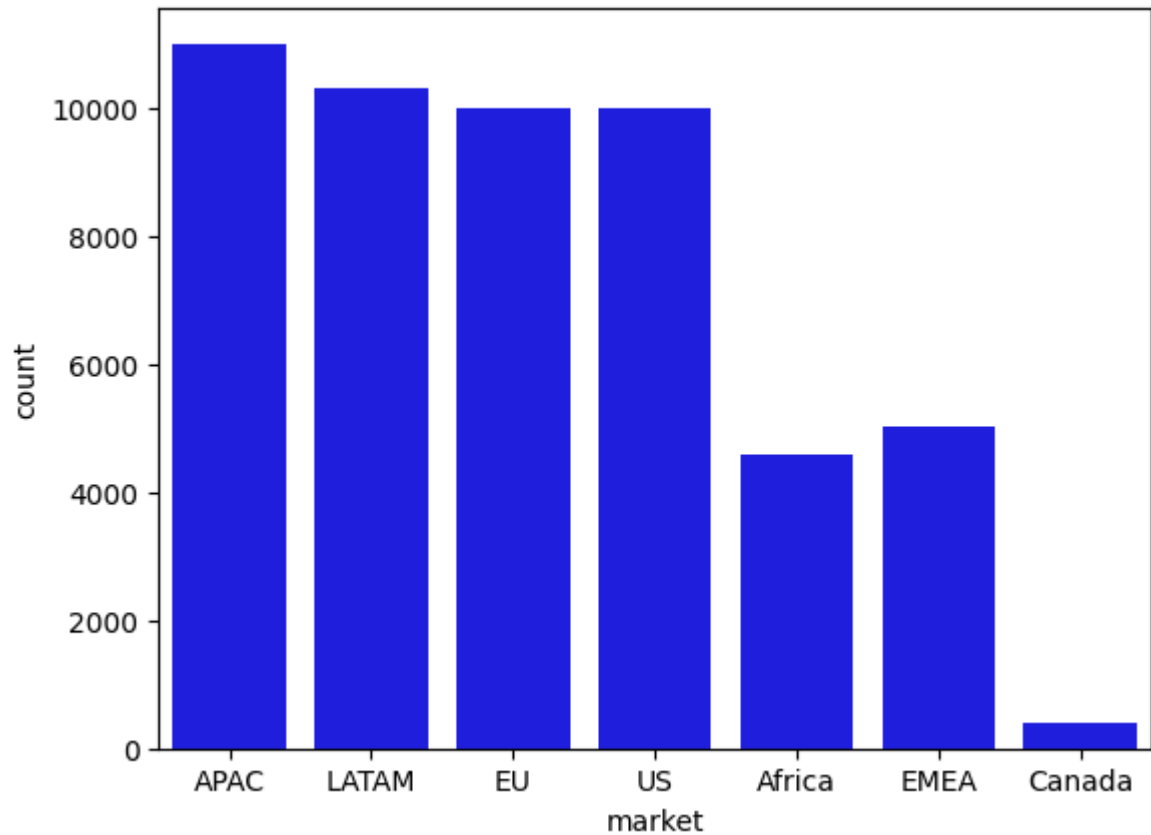
```
In [43]: sns.countplot(data = comercio, x='market')
```

```
Out[43]: <Axes: xlabel='market', ylabel='count'>
```



```
In [32]: # podemos ordenar os dados e implementar cores
sns.countplot(data = comercio, x='market', order=["APAC", "LATAM", "EU", "US", "Afr

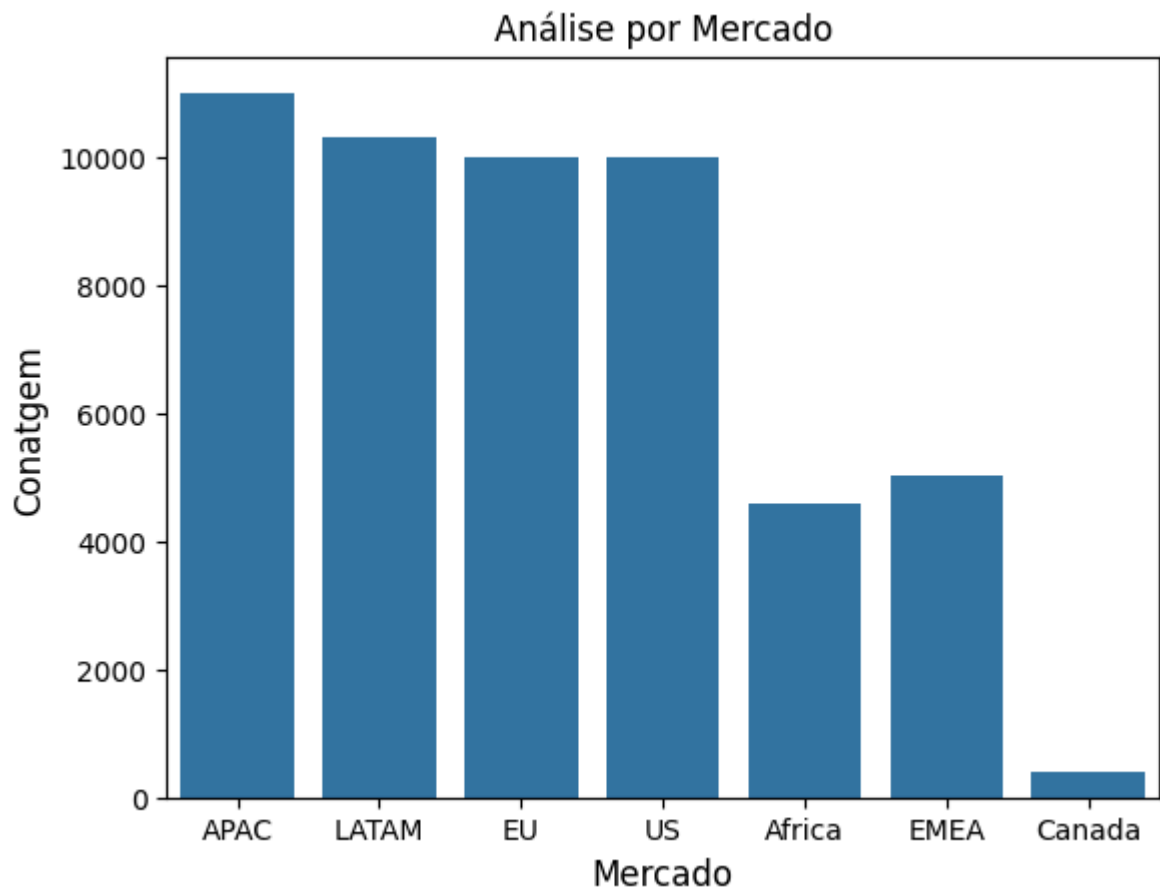
Out[32]: <Axes: xlabel='market', ylabel='count'>
```



### Adicionando mais formatação

Podemos utilizar a `matplotlib.pyplot` para adicionar mais informações ao gráfico

```
In [33]: sns.countplot(data = comercio, x='market', order=["APAC", "LATAM", "EU", "US", "Africa", "EMEA", "Canada"])
plt.title("Análise por Mercado")
plt.xlabel('Mercado', fontsize=12)
plt.ylabel('Conatgem', fontsize=12)
plt.show()
```



## Paleta de cores

A biblioteca seaborn vem com algumas paletas de cores

### Paleta bright

```
In [34]: # Paleta bright  
  
palette = sns.color_palette("bright")  
sns.palplot(palette)
```



### Paleta viridis

```
In [35]: palette = sns.color_palette("viridis")  
sns.palplot(palette)
```



### Paleta Paired

```
In [36]: palette = sns.color_palette("Paired")
sns.palplot(palette)
```



### Paleta Rocket

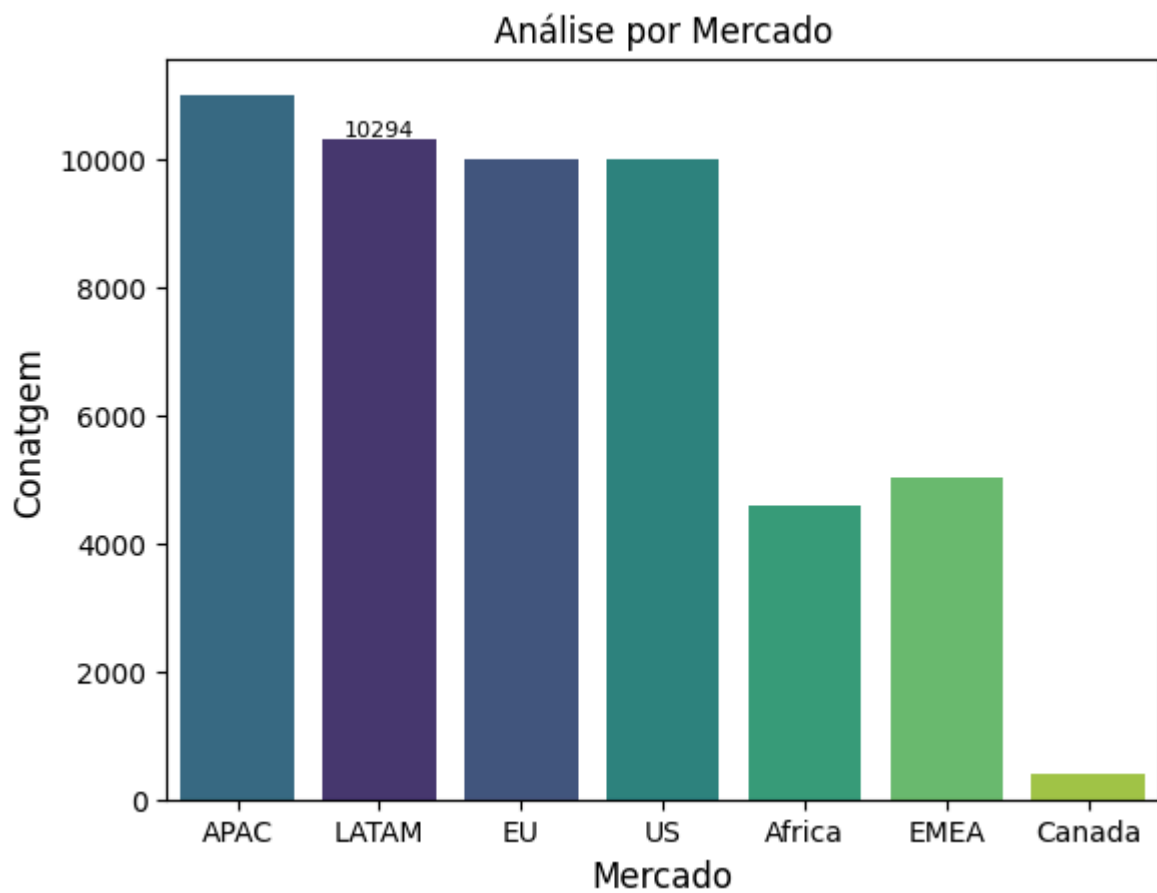
```
In [37]: palette = sns.color_palette("rocket")
sns.palplot(palette)
```



Podemos utilizar a paleta de cores no nosso plot

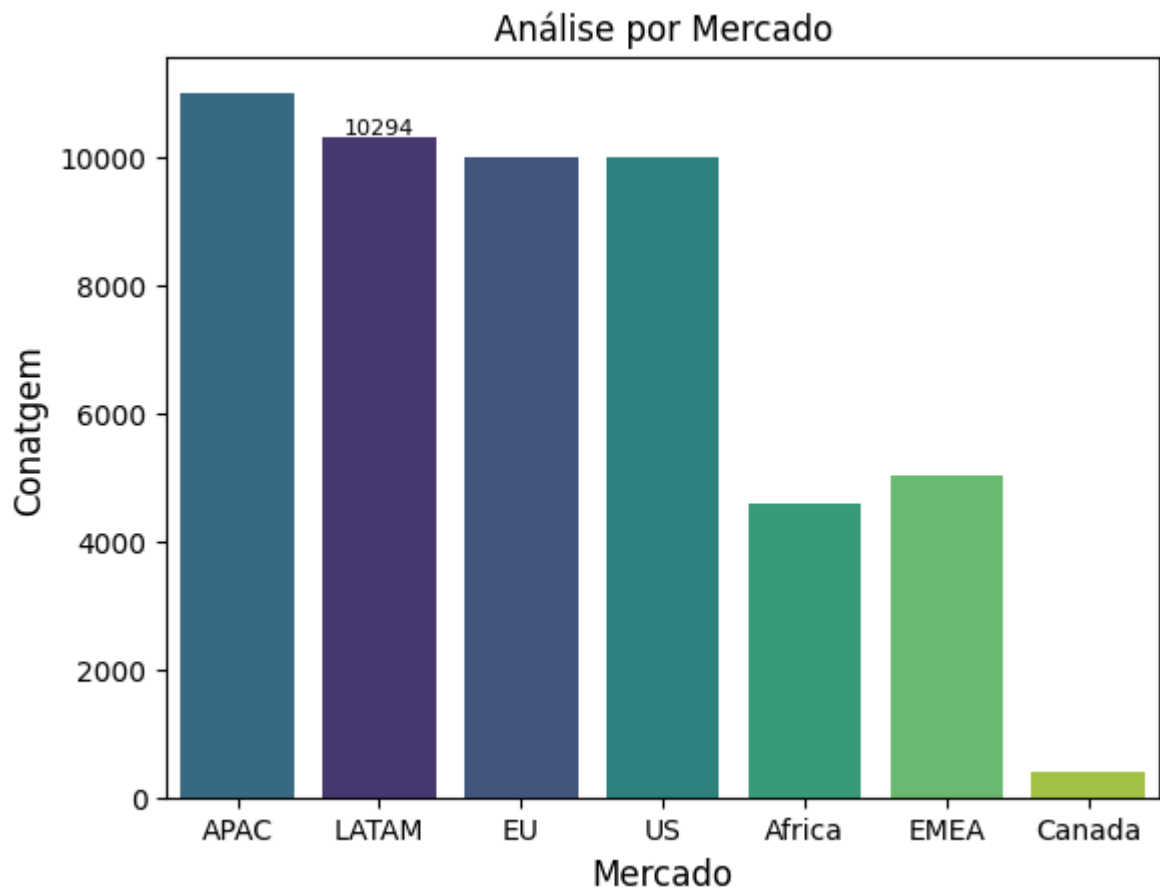
```
In [42]: ax = sns.countplot(data=comercio, x="market", hue="market", order=["APAC", "LATAM", "
ax.bar_label(ax.containers[0], fontsize=8)
plt.title("Análise por Mercado")
plt.xlabel('Mercado', fontsize=12)
plt.ylabel('Conatgem', fontsize=12)
plt.show()
```





## Salvando o objeto plt

```
In [46]: ax = sns.countplot(data=comercio, x="market", hue="market", order=["APAC", "LATAM",  
ax.bar_label(ax.containers[0], fontsize=8)  
plt.title("Análise por Mercado")  
plt.xlabel('Mercado', fontsize=12)  
plt.ylabel('Conatgem', fontsize=12)  
plt.show()
```



### Invertendo a ordem do gráfico

```
In [54]: comercio['market'].value_counts(ascending=False) # contagem de dados
```

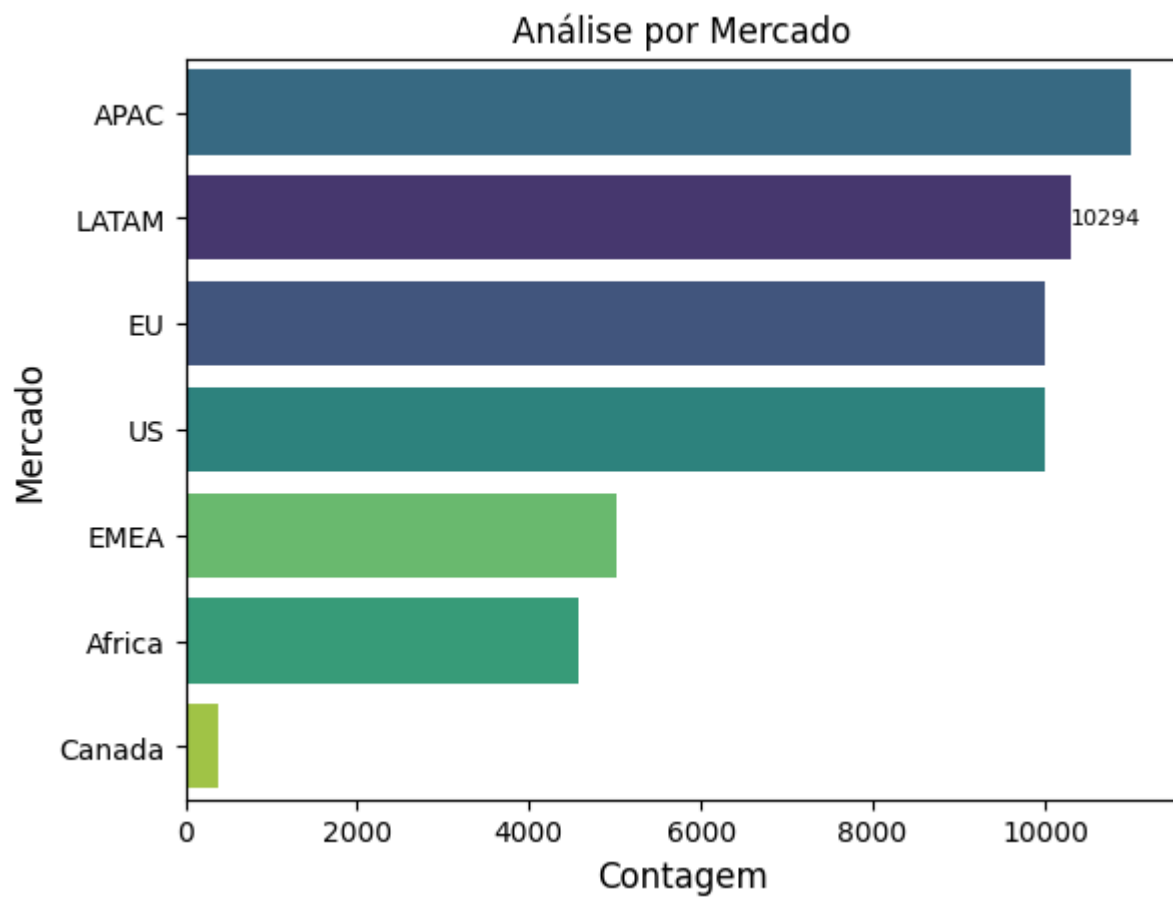
```
Out[54]: market
APAC      11002
LATAM     10294
EU         10000
US          9994
EMEA        5029
Africa     4587
Canada      384
Name: count, dtype: int64
```

```
In [56]: comercio['market'].value_counts(ascending=False).index # ordena os labels em ordem
```

```
Out[56]: Index(['APAC', 'LATAM', 'EU', 'US', 'EMEA', 'Africa', 'Canada'], dtype='object', name='market')
```

```
In [49]: ax = sns.countplot(data=comercio, y="market",
                             order = comercio['market'].value_counts(ascending=False).index,
                             palette = 'viridis',
                             hue = "market")
ax.bar_label(ax.containers[0], fontsize=8)
plt.title("Análise por Mercado")
plt.xlabel('Contagem', fontsize=12)
```

```
plt.ylabel('Mercado',fontsize=12)  
plt.show()
```



```
In [ ]: !pip install playwright  
        !jupyter nbconvert --to webpdf --allow-chromium-download "Introdução ao python.ipyn"
```