

UNIDADE 3

Infraestrutura para Manipulação e Visualização de Dados

Disciplina: Tópicos Especiais III (DEC7553)

Prof. Alexandre L. Gonçalves

E-mail: a.l.goncalves@ufsc.br

■ Limpeza e Preparação dos Dados

- Durante as fases que antecedem a análise e a modelagem de dados, muito tempo é gasto na fase de preparação;
- A preparação inclui a limpeza, transformação e organização;
- Segundo alguns estudos este tempo pode chegar a 80% do trabalho de determinado analista;
- As vezes o modo como os dados são armazenados em arquivos ou bancos de dados não representam o formato correto para uma tarefa de análise em particular;
- Existem diferentes formas de realizar o processamento dos dados;

■ Limpeza e Preparação dos Dados

- Pode ser através de linguagens de programação como Python, R, C++ ou Java, ou por meio de ferramentas especializadas;
- Em Python uma das ferramentas (bibliotecas) mais utilizada é a Pandas que, juntamente com recursos da linguagem Python, provê um conjunto de funcionalidades de alto nível, com desempenho e flexibilidade.

■ Tratando Dados Ausentes

- Dados ausentes são comuns em muitas aplicações de análise de dados;
- Para tal, a biblioteca Pandas objetiva facilitar o processamento em diversos níveis;
- Todas as estatísticas descritivas em objetos do Pandas excluem dados ausentes;
- Para representar dados ausentes o Pandas utiliza o valor de ponto flutuante NaN (Not a Number);
- Este valor é chamado de **valor de sentinela**, e pode ser facilmente identificado.

■ Tratando Dados Ausentes

- Analisando quais dados em uma série são nulos;

```
string_data = pd.Series(['abacaxi', 'laranja', np.nan, 'abacate'])  
string_data  
string_data.isnull()
```

```
0    False  
1    False  
2     True  
3    False  
dtype: bool
```

■ Tratando Dados Ausentes

- No Pandas é adotada a convenção de dados ausentes como NA, ou seja, *Not Available*;
- Em aplicações estatísticas, dados NA podem representar dados inexistentes ou existentes, contudo, não foram observados possivelmente por problemas na coleta;
- Na fase de limpeza dos dados é importante analisar os dados faltantes a fim de identificar problemas de coleta ou possíveis distorções provocadas por dados ausentes.

■ Tratando Dados Ausentes

- O valor embutido **None** do Python também é tratado como NA em arrays de objetos.

```
string_data[0] = None  
string_data.isnull()
```

```
0    True  
1    False  
2    True  
3    False  
dtype: bool
```

■ Filtrando Dados Ausentes

- Existem algumas maneiras de filtrar dados ausentes;
- Entre as possibilidades está o método **dropna**;
- Em uma série somente os dados diferentes de null e os valores de índices são devolvidos.

```
from numpy import nan as NA
data = pd.Series([1, NA, 3.5, NA, 7])
data.dropna()
```

```
0    1.0
2    3.5
4    7.0
dtype: float64
```


■ Filtrando Dados Ausentes

□ O exemplo anterior é equivalente a:

```
data[data.notnull()]
```

```
0    1.0  
2    3.5  
4    7.0  
dtype: float64
```

■ Filtrando Dados Ausentes

- Com objetos DataFrame a situação é um pouco mais complexa;
- Talvez seja necessário descartar linhas ou colunas que possuam somente NA ou apenas aquelas que possuam algum NA;
- Por padrão, **dropna** descarta qualquer linha contendo um valor ausente.

■ Filtrando Datos Ausentes

```
data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],  
                    [NA, NA, NA], [NA, 6.5, 3.]])
```

```
cleaned = data.dropna()
```

data

```
   0    1    2  
0 1.0  6.5  3.0  
1 1.0  NaN NaN  
2 NaN  NaN NaN  
3 NaN  6.5  3.0
```

cleaned

	0	1	2
0	1.0	6.5	3.0

■ Filtrando Dados Ausentes

- Utilizar `how='all'` descartará apenas as linhas que contenham somente NAs;

```
data.dropna(how='all')
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

■ Filtrando Dados Ausentes

- Para descartar colunas do mesmo modo deve-se utilizar a propriedade axis (0=linha, 1=coluna);

```
data[3] = NA
print(data)
print()
res = data.dropna(axis=0, how='all')
print(res)
final = res.dropna(axis=1, how='all')
print()
print(final)
```

	0	1	2	3
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

	0	1	2	3
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

■ Filtrando Dados Ausentes

- Para manter linhas contendo pelo menos n observações não nulas utilizamos o atributo **thresh**;

```
df = pd.DataFrame(np.random.randn(7, 3))
df.iloc[4, 1] = NA
df.iloc[:2, 2] = NA
df
df.dropna()
df.dropna(thresh=2)
```

■ Filtrando Datos Ausentes

	0	1	2
0	0.107657	NaN	NaN
1	-0.017007	NaN	NaN
2	1.634736	NaN	0.457940
3	0.555154	NaN	-0.440554
4	-0.301350	0.498791	-0.823991
5	1.320566	0.507965	-0.653438
6	0.186980	-0.391725	-0.272293

`df.dropna()`

`df.dropna(thresh=2)`

	0	1	2
2	0.910983	NaN	-1.413416
3	1.296608	NaN	1.127481
4	-0.568363	0.309362	-0.577385
5	-1.168634	-0.825020	-2.644409
6	-0.152986	-0.751921	-0.132609

■ Preenchendo Dados Ausentes

- Em vez de filtrar dados ausentes e, possivelmente, descartar dados necessários, pode-se preencher as lacunas de várias maneiras;
- Na maioria dos casos, o método fillna cumpre o papel.

df.fillna(0)

	0	1	2
0	-0.289436	0.000000	0.000000
1	0.838775	0.000000	0.000000
2	0.910983	0.000000	-1.413416
3	1.296608	0.000000	1.127481
4	-0.568363	0.309362	-0.577385
5	-1.168634	-0.825020	-2.644409
6	-0.152986	-0.751921	-0.132609

■ Preenchendo Dados Ausentes

- Ao chamar `fillna` com um dicionário, pode-se utilizar um valor de preenchimento diferente para cada coluna;

```
df.fillna({1: 0.5, 2: 0})
```

	0	1	2
0	1.457300	0.500000	0.000000
1	1.239980	0.500000	0.000000
2	-0.846852	0.500000	1.263572
3	-0.255491	0.500000	0.468367
4	-0.961604	-1.824505	0.625428
5	1.022872	1.107425	0.090937
6	-0.350109	0.217957	-0.894813

■ Preenchendo Dados Ausentes

- fillna devolve um novo objeto, mas o objeto existente pode ser alterado in-place;

```
df.fillna(0, inplace=True)
```

```
df
```

	0	1	2
0	1.457300	0.000000	0.000000
1	1.239980	0.000000	0.000000
2	-0.846852	0.000000	1.263572
3	-0.255491	0.000000	0.468367
4	-0.961604	-1.824505	0.625428
5	1.022872	1.107425	0.090937
6	-0.350109	0.217957	-0.894813

■ Preenchendo Dados Ausentes

- Utilizando o atributo *method* no método `fillna()` com o valor `ffill` irá preencher os valores nulos com base no valor anterior de cada coluna;

```
df = pd.DataFrame(np.random.randn(6, 3))
```

```
df.iloc[2:, 1] = NA
```

```
df.iloc[4:, 2] = NA
```

```
print(df)
```

```
df.fillna(method='ffill')
```

```
df.fillna(method='ffill', limit=2)
```

	0	1	2
0	0.700428	2.092852	-0.136972
1	-0.930489	0.327497	1.303013
2	-1.409402	0.327497	-0.716414
3	0.103614	0.327497	-1.174894
4	2.613999	NaN	-1.174894
5	0.636281	NaN	-1.174894

■ Preenchendo Dados Ausentes

- Com `fillna` pode-se executar outras tarefas, por exemplo, passar o valor da média (*mean*) ou da mediana (*median*) de uma série;

```
data = pd.Series([1., NA, 3.5, NA, 7])  
data.fillna(data.mean())
```

0	1.0	0	1.000000
1	NaN	1	3.833333
2	3.5	2	3.500000
3	NaN	3	3.833333
4	7.0	4	7.000000
dtype: float64		dtype: float64	

■ Transformação de Dados (Removendo Duplicatas)

- Linhas duplicadas podem ser encontradas em um DataFrame por diversos motivos;

```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],  
                    'k2': [1, 1, 2, 3, 3, 4, 4]})
```

data

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

■ Transformação de Dados (Removendo Duplicatas)

- O método **deduplicated** de DataFrame informa em uma série se determinada linha é uma duplicata;

```
data.duplicated()
```

```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

■ Transformação de Dados (Removendo Duplicatas)

- As duplicatas são removidas através do método `drop_duplicates()`;

`data.drop_duplicates()`

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

■ Transformação de Dados (Removendo Duplicatas)

- Os dois métodos anteriores consideram todas as colunas. Contudo, pode-se especificar qualquer subconjunto de colunas na detecção de duplicatas;

```
data['v1'] = range(7)
```

```
print(data)
```

```
data.drop_duplicates(['k1'])
```

```
   k1 k2 v1
0 one  1  0
1 two  1  1
2 one  2  2
3 two  3  3
4 one  3  4
5 two  4  5
6 two  4  6
```

	k1	k2	v1
0	one	1	0
1	two	1	1

■ Transformação de Dados (Removendo Duplicatas)

- `deduplicated()` e `drop_duplicates()`, por padrão, mantêm a primeira combinação de valores observados. Para considerar a última utiliza-se a propriedade `keep='last'`;

```
data.drop_duplicates(['k1'], keep='last')
```

	k1	k2	v1
4	one	3	4
6	two	4	6

- **Transformação de Dados** (Transformando dados usando uma função ou um mapeamento)
 - Dependendo da situação pode ser requerida transformações com base em valores de um array, uma série ou uma coluna de uma DataFrame;

```
data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',  
                             'Pastrami', 'corned beef', 'Bacon',  
                             'pastrami', 'honey ham', 'nova lox'],  
                    'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
```

data

■ Transformação de Dados (Transformando dados usando uma função ou um mapeamento)

	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

- **Transformação de Dados** (Transformando dados usando uma função ou um mapeamento)
 - Suponha que seja necessário adicionar uma coluna informando o tipo de animal a partir do qual cada alimento é proveniente;

```
meat_to_animal = {  
    'bacon': 'pig',  
    'pulled pork': 'pig',  
    'pastrami': 'cow',  
    'corned beef': 'cow',  
    'honey ham': 'pig',  
    'nova lox': 'salmon'  
}
```

■ Transformação de Dados (Transformando dados usando uma função ou um mapeamento)

```
lowercased = data['food'].str.lower()
```

```
lowercased
```

```
data['animal'] = lowercased.map(meat_to_animal)
```

```
data
```

	food	ounces	animal
0	Bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	Pastrami	6.0	cow
4	corned beef	7.5	cow
5	Bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

- Transformação de Dados (Transformando dados usando uma função ou um mapeamento)
 - Poderíamos também ter passado uma função que fizesse todo o trabalho;

```
data['animal'] = data['food'].map(lambda x: meat_to_animal[x.lower()])
```

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	Pastrami	6.0	cow
4	corned beef	7.5	cow
5	Bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

- Transformação de Dados (Substituindo valores)
 - Além de fillna e map, pode-se utilizar o método replace para substituição de valores como uma forma mais simples e flexível;

```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
```

```
data
```

```
0    1.0  
1  -999.0  
2    2.0  
3  -999.0  
4 -1000.0  
5    3.0  
dtype: float64
```

■ Transformação de Dados (Substituindo valores)

```
data.replace(-999, np.nan)
```

```
0    1.0  
1    NaN  
2    2.0  
3    NaN  
4 -1000.0  
5    3.0  
dtype: float64
```

```
data.replace([-999, -1000], np.nan)
```

```
0    1.0  
1    NaN  
2    2.0  
3    NaN  
4    NaN  
5    3.0  
dtype: float64
```

```
data.replace([-999, -1000], [np.nan, 0])
```

```
0    1.0  
1    NaN  
2    2.0  
3    NaN  
4    0.0  
5    3.0  
dtype: float64
```

```
data.replace({-999: np.nan, -1000: 0})
```

```
0    1.0  
1    NaN  
2    2.0  
3    NaN  
4    0.0  
5    3.0  
dtype: float64
```


■ Transformação de Dados (Renomeando os Índices dos Eixos)

- Assim com os valores em uma série, os rótulos dos eixos podem ser transformados de maneira similar por uma função ou uma forma de mapeamento;

```
data = pd.DataFrame(np.arange(12).reshape((3, 4)),  
                    index=['Paraná', 'Santa Catarina', 'Rio Grande do Sul'],  
                    columns=['um', 'dois', 'tres', 'quatro'])
```

data

	um	dois	tres	quatro
Paraná	0	1	2	3
Santa Catarina	4	5	6	7
Rio Grande do Sul	8	9	10	11

■ Transformação de Dados (Renomeando os Índices dos Eixos)

- Assim como em uma série, os índices de um eixo possuem um método map;

```
transform = lambda x: x.upper()  
data.index.map(transform)
```

```
Index(['PARANÁ', 'SANTA CATARINA', 'RIO GRANDE DO SUL'],  
      dtype='object')
```


■ Transformação de Dados (Renomeando os Índices dos Eixos)

- É possível realizar uma atribuição para index, modificando o DataFrame in-place;

```
data.index = data.index.map(transform)
```

```
data
```

	um	dois	tres	quatro
PARANÁ	0	1	2	3
SANTA CATARINA	4	5	6	7
RIO GRANDE DO SUL	8	9	10	11

- 
- **Transformação de Dados** (Discretização e Compartimentalização)
 - Frequentemente tem-se a necessidade de discretizar dados contínuos em compartimentos (bins) para que análises sejam realizadas;
 - Como exemplo, vamos considerar um grupo de pessoas em um estudo em que a intenção é agrupar por idades discretizadas;

■ Transformação de Dados (Discretização e Compartimentalização)

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

```
bins = [18, 25, 35, 60, 100]
```

```
cats = pd.cut(ages, bins)
```

```
cats
```

```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60],  
(35, 60], (25, 35]] Length: 12 Categories (4, interval[int64]): [(18, 25] < (25,  
35] < (35, 60] < (60, 100]]
```

cats.codes	(18, 25]	5
cats.categories	(35, 60]	3
	(25, 35]	3
pd.value_counts(cats)	(60, 100]	1
	dtype: int64	

■ Transformação de Dados (Detectando e filtrando valores discrepantes)

- Em geral, filtrar ou transformar valores discrepantes (outliers) é uma questão de aplicar operações de array. Considere um DataFrame com dados normalmente distribuídos;

```
data = pd.DataFrame(np.random.randn(1000, 4))  
data.describe()
```

■ Transformação de Dados (Detectando e filtrando valores discrepantes)

- Caso fosse requerido encontrar os valores que excedem 3 em valor absoluto em uma das colunas;

```
col = data[2]  
col[np.abs(col) > 3]
```

```
41    -3.399312  
136    -3.745356  
Name: 2, dtype: float64
```

■ Transformação de Dados (Detectando e filtrando valores discrepantes)

- Para selecionar todas as linhas em que os valores excedam 3 ou -3, pode-se utilizar o método `any` em `DataFrame`;

```
data[(np.abs(data) > 3).any(1)]
```

	0	1	2	3
246	-3.398512	0.561477	-0.995473	-0.557069
258	-3.098856	-0.281689	-0.918874	-0.935530
442	3.069109	1.085835	1.355198	0.763169
477	1.088333	3.199487	-3.167749	-0.156923
664	-3.049587	-1.516184	0.565160	-1.447332
705	0.723983	3.080926	0.838979	2.910516
719	-1.417417	-0.987457	3.041182	-0.249577
927	0.252296	3.089328	-0.080754	-0.919374

■ Transformação de Dados (Detectando e filtrando valores discrepantes)


	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.022520	-0.039360	0.059577	0.013372
std	0.991972	0.994021	1.003896	1.008059
min	-3.745356	-3.428254	-3.645860	-3.184377
25%	-0.623928	-0.738859	-0.599807	-0.641675
50%	0.001781	-0.083097	0.087191	-0.023333
75%	0.680673	0.623776	0.772359	0.670265
max	3.927528	3.366626	2.653656	3.525865

■ Transformação de Dados (Detectando e filtrando valores discrepantes)

- O exemplo a seguir ajusta os valores que estejam fora do intervalo de -3 e 3 ;

```
data[np.abs(data) > 3] = np.sign(data) * 3  
data.describe()
```

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.057862	-0.015234	0.007739	-0.005076
std	0.988097	1.004301	0.979386	0.969187
min	-3.000000	-2.995410	-3.000000	-2.676416
25%	-0.609083	-0.676141	-0.663171	-0.648887
50%	0.063611	-0.007445	-0.003778	-0.042938
75%	0.666164	0.608836	0.661509	0.645200
max	3.000000	3.000000	3.000000	2.932621

- 
- Transformação de Dados (Calculando variáveis indicadoras (dummy))
 - Uma transformação importante para modelagem estatística ou aplicações de aprendizado de máquina consiste em converter uma variável de categorias em uma matriz indicadora ou “dummy”;
 - Se uma coluna em um DataFrame tiver k valores distintos, pode-se derivar uma matriz ou um DataFrame com k colunas contendo somente 1s e 0s;
 - Pandas tem um método `get_dummies` para esta funcionalidade;

■ Transformação de Dados (Calculando variáveis indicadoras (dummy))

```
df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],  
                  'data1': range(6)})  
  
print(df)  
pd.get_dummies(df['key'])
```

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

	a	b	c
0	0	1	0
1	0	1	0
2	1	0	0
3	0	0	1
4	1	0	0
5	0	1	0

- Transformação de Dados (Calculando variáveis indicadoras (dummy))
 - Se uma linha de um conjunto de dados possuir várias categorias, o processamento para separar as categorias torna-se mais complicado;

```
mnames = ['movie_id', 'title', 'genres']  
movies = pd.read_csv('../datasets/movielens/movies.csv', sep=':',  
                    header=None, names=mnames)  
movies[:10]
```

■ Transformação de Dados (Calculando variáveis indicadoras (dummy))

	movie_id	title	genres
0	1	Toy Story (1995)	Animation Children's Comedy
1	2	Jumanji (1995)	Adventure Children's Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action Crime Thriller
6	7	Sabrina (1995)	Comedy Romance
7	8	Tom and Huck (1995)	Adventure Children's
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action Adventure Thriller

- Transformação de Dados (Calculando variáveis indicadoras (dummy))
 - Adicionar variáveis indicadoras para cada gênero exige alguma manipulação nos dados;

```
all_genres = []  
for x in movies.genres:  
    all_genres.extend(x.split('|'))  
genres = pd.unique(all_genres)
```

```
array(['Animation', 'Children's', 'Comedy', 'Adventure', 'Fantasy',  
      'Romance', 'Drama', 'Action', 'Crime', 'Thriller',  
      'Dead and Loving It (1995)', 'When Nature Calls (1995)',  
      'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery',  
      'Beyond Cyberspace (1996)', 'Horror', 'Hollywood Madam (1995)',  
      'Film-Noir', 'With a Vengeance (1995)', .... ], dtype=object)
```

- Transformação de Dados (Calculando variáveis indicadoras (dummy))
 - Para construir o DataFrame indicador pode-se iniciar com uma estrutura contendo apenas zeros;

```
zero_matrix = np.zeros((len(movies), len(genres)))  
dummies = pd.DataFrame(zero_matrix, columns=genres)  
dummies
```

Out -> será criada uma matriz com a quantidade de linhas do dataset e com o Número de colunas do array genres.

- **Transformação de Dados** (Calculando variáveis indicadoras (dummy))
 - Após isso deve-se iterar pelos filmes para definir entradas para cada linha em dummies com 1;
 - E então utilizar `iloc` para definir valores com base nesses índices;

```
for i, gen in enumerate(movies.genres):  
    indices = dummies.columns.get_indexer(gen.split('|'))  
    dummies.iloc[i, indices] = 1
```

```
movies_windic = movies.join(dummies.add_prefix('Genre_'))  
movies_windic.iloc[0]
```

■ Transformação de Dados (Calculando variáveis indicadoras (dummy))

movie_id	1
title	Toy Story (1995)
genres	Animation Children's Comedy
Genre_Animation	1
Genre_Children's	1
Genre_Comedy	1
Genre_Adventure	0
Genre_Fantasy	0
Genre_Romance	0
Genre_Drama	0

■ Transformação de Dados (Eliminando colunas)

- Colunas específicas podem ser excluídas indicando-as em um array

```
to_drop = ['Coluna 1',  
           'Coluna 2']  
data.drop(to_drop, inplace=True, axis=1)
```



■ Manipulação de Strings (Métodos de Objetos String)

- Entre os motivos da popularidade do Python reside a facilidade de uso no processamento de strings e texto;
- A maior parte das operações em texto são simplificadas com os métodos da classe *String*;
- Contudo, para manipular padrões complexos em texto o uso de expressões regulares pode ser adequado.

■ Manipulação de Strings (Métodos de Objetos *String*)

- `split()` – separa *strings* com base em algum conteúdo
- `strip()` – remove espaços em branco
- `join()` – permite concatenar elementos de uma lista ou uma tupla
- `index()` – realiza buscas na *string* retornando a posição caso seja encontrado. Em caso contrário, uma exceção será lançada
- `find()` – idem a `index` mas retorna -1 quando determinado conteúdo não é localizado
- `count()` – conta as ocorrências de determinado conteúdo em *strings*
- `replace()` – substitui conteúdos em uma *string*

■ Manipulação de Strings (Expressões Regulares)

- Fornecem uma maneira flexível para realizar pesquisas ou correspondências de padrões de string em um texto;
- O módulo embutido **re** do Python permite aplicar expressões regulares em strings;
- As funções do módulo **re** são divididas em três categorias: correspondências, substituição e separação de padrões.

■ Manipulação de Strings (Expressões Regulares)

- `compile()` – cria um objeto regex reutilizável;
- `findall()` – obtém uma lista de todos os padrões que correspondam à regex;
- `search()` – devolve um objeto especial de correspondência para o primeiro padrão no texto;
- `groups()` – devolve uma tupla dos componentes do padrão;



■ Conclusão

- Uma preparação de dados eficiente pode melhorar de modo significativo a produtividade, possibilitando investir mais tempo na análise.



Bons Estudos!