Pandas – Limpeza de Dados

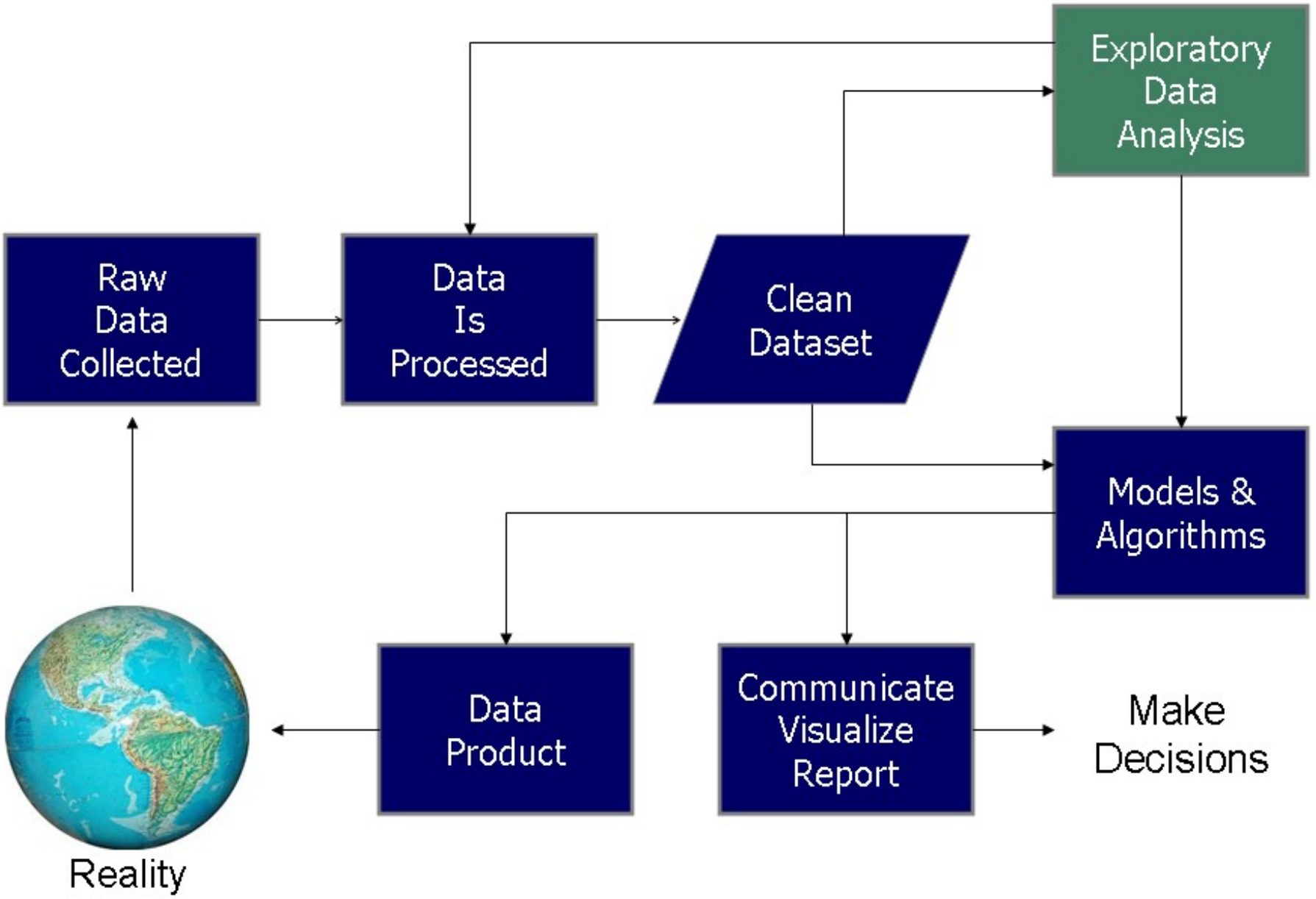
Luiz Celso Gomes-Jr

Limpeza de Dados

- Processo fundamental no ciclo de ciência de dados
- "models are what they eat (garbage in, garbage out)"

http://www.weeklystandard.com/how-to-build-a-senate-election-model-step-1/article/2011716

Data Science Process



Data Science Cycle – The importance of good data management

CrowdFlower
2015 Data Scientist Report

Lendo dados de um arquivo

- Usaremos o dataset ‘aluguel-com-erros.csv’ nos exemplos

```
# lê o arquivo CSV
df = pd.read_csv('../data/aluguel-com-erros.csv')
df.head()
```

	codigo	endereco	quartos	suite	area	vaga	aluguel	condominio	data
0	34	Rua Desembargador Westphalen	2	0	90	0	900	371	11/10/17
1	167	Rua Jose Loureiro	2	0	64	0	650	428	15/07/17
2	6784	Rua Jose Loureiro	2	0	81	0	1100	1100	23/08/17
3	82	Rua Lourenço Pinto	2	0	50	0	1350	300	19/09/17
4	2970	Rua Lourenço Pinto	2	0	63	0	1300	300	05/08/17

Verificando tipos

O método **info** é útil para vermos quais colunas do DataFrame tiveram seus tipos de dados corretamente identificados pelo Pandas. No caso abaixo já é possível identificar alguns problemas. As colunas quartos, suite, area, aluguel e condomínio foram interpretadas como objetos genéricos enquanto deveriam ser numéricas. A coluna data deveria ser do tipo *datetype*.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 9 columns):
codigo        20 non-null int64
endereco      19 non-null object
quartos       20 non-null int64
suite         20 non-null object
area          20 non-null object
vaga          20 non-null int64
aluguel       20 non-null object
condominio    20 non-null object
data          20 non-null object
dtypes: int64(3), object(6)
memory usage: 1.5+ KB
```

Localizando e tratando valores inválidos

Uma forma prática de identificar potenciais problemas é exibir todas as linhas com valores em branco (NaN). Abaixo podemos verificar que uma das linhas não tem o valor para a coluna endereco.

```
df[df.isna().any(axis=1)]
```

	codigo	endereco	quartos	suite	area	vaga	aluguel	condominio	data
7	469	NaN	1	0	30	0	550	210	03/07/17

Localizando e tratando valores inválidos

Uma forma prática de identificar potenciais problemas é exibir todas as linhas com valores em branco (NaN). Abaixo podemos verificar que uma das linhas não tem o valor para a coluna endereco.

```
df[df.isna().any(axis=1)]
```

	codigo	endereco	quartos	suite	area	vaga	aluguel
7	469	NaN	1	0	30	0	550

Encadeamento de métodos

Mais detalhes sobre este tipo de operação serão dados na próxima aula. Mas é possível ter uma ideia do que está acontecendo executando partes do comando, por exemplo:
df.isna()

Preenchendo valores vazios (NaN)

Podemos usar o método **fillna** para substituir os valores em branco por um valor definido, neste caso a String "Desconhecido". Após a substituição não há mais linhas com valores em branco.

```
df['endereco'] = df['endereco'].fillna("Desconhecido")

df[df.isna().any(axis=1)]
```

codigo	endereco	quartos	suite	area	vaga	aluguel	condominio	data

Substituindo substrings (str.replace)

Podemos também fazer substituições de partes das strings no DataFrame. Por exemplo, podemos padronizar ruas e avenidas com as abreviações R. e Av. O código abaixo faz isto.

```
df['endereco'] = df['endereco'].str.replace('Rua', 'R.')
df['endereco'] = df['endereco'].str.replace('Avenida', 'Av.')

df.head()
```

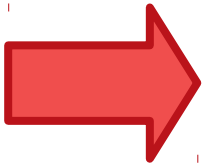
	codigo	endereco	quartos	suite	area	vaga	aluguel	condominio	data
0	34	R. Desembargador Westphalen	2	0	90	0	900	371	11/10/17
1	167	R. Jose Loureiro	2	0	64	0	650	428	15/07/17
2	6784	R. Jose Loureiro	2	0	81	0	1100	1100	23/08/17
3	82	R. Lourenço Pinto	2	0	50	0	1350	300	19/09/17
4	2970	R. Lourenço Pinto	2	0	63	0	1300	300	05/08/17

Tradando *strings*

O pacote str possui diversos métodos para tratar strings como:

- str.lower() - Converte para caixa baixa
- str.upper() - Caixa alta
- str.strip() - Elimina espaços no início e final da string
- str.split('_') - Separa a string de acordo com o caractere
- str.split('_').str.get(1) – Retorna valor separado

	nome	idade	genero
0	Celso	23	M
1	Josie	32	F
2	Luiz	11	M
3	Helen	18	F



```
df['nome'] = df['nome'].str.lower()
df
```

	nome	idade	genero
0	celso	23	M
1	josie	32	F
2	luiz	11	M
3	helen	18	F

Verificando valores inválidos

Uma forma de averiguar valores inválidos é listar todos os valores não repetidos de uma coluna. O método **unique** é útil neste caso.

```
df['aluguel'].unique()
```

```
array(['900', '650', '1100', '1350', '1300', '550', '800', '1800',
'600',
       '750', '560', '?'], dtype=object)
```

Verificando valores inválidos

Outra forma de averiguar valores inválidos é listar todos os valores que não são do tipo esperado. O código abaixo usa o método **str.isnumeric** para isto.

```
df[df['aluguel'].str.isnumeric() == False]
```

	codigo	endereco	quartos	suite	area	vaga	aluguel	condominio	data
19	44803	R. Rockefeller	2	?	?	1	?	?	19/07/17

Excluindo valores inválidos

Podemos decidir retirar a linha com problemas do DataFrame. Para isso, usamos o método **drop**.

```
# Exclui linhas de índice = 19
df1 = df.drop(19)
# Exclui todas as linhas que satisfazem a condição
df2 = df.drop(df[df['aluguel'].str.isnumeric() == False].index)
# Exclui todas as linhas com algum valor igual a "?" na coluna aluguel
df3 = df.drop(df[df['aluguel'] == "?"].index)

# Impreme o número final de linhas em cada um dos dataframes limpos
print("Total df: {}, Total df1: {}, Total df2:{}, Total df3:{}"\
            .format(len(df), len(df1), len(df2), len(df3)))
```

```
Total df: 20, Total df1: 19, Total df2:19, Total df3:19
```

Conversão de tipos

Para converter as colunas para os tipos adequados, usamos o método **astype**. Podemos tanto aplicar a uma coluna por vez ou em várias colunas representadas por um dicionário. O código abaixo exemplifica as duas abordagens.

```
df['suite'] = df['suite'].astype(int)

df = df.astype({'aluguel': float, 'condominio': float})
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19 entries, 0 to 18
Data columns (total 9 columns):
codigo        19 non-null int64
endereco      19 non-null object
quartos       19 non-null int64
suite         19 non-null int64
area          19 non-null int64
vaga          19 non-null int64
aluguel       19 non-null float64
condominio    19 non-null float64
data          19 non-null object
```

Conversão de tipos

Para uma coluna com datas, precisamos usar o comando **to_datetime**.

```
df['data'] = pd.to_datetime(df['data'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19 entries, 0 to 18
Data columns (total 9 columns):
codigo        19 non-null int64
endereco      19 non-null object
quartos       19 non-null int64
suite         19 non-null int64
area          19 non-null int64
vaga          19 non-null int64
aluguel       19 non-null float64
condominio    19 non-null float64
data          19 non-null datetime64[ns]
```

Exercícios!

- Revise o conteúdo e faça os exercícios do notebook:
  03d-Pandas_Limpeza de Dados.ipynb
- Faça os exercícios do notebook:
  03d1-Exercício-Pandas_Limpeza de Dados.ipynb