Aula 06 - Transformação e visualização de dados

Na última aula vimos sobre como podemos extrair dados e salvá-los em arquivos .json, locais. Na aula de hoje vamos ver o processo de transformação e visualização desses dados.

O processo de transformação de dados é uma etapa essencial para qualquer análise de dados. É nesse processo que o cientista de dados se livra do conteúdo que não será necessário para a análise, além de limpar dados inválidos, ajustar formatos, balancear classes de dados diferentes e identificar valores que possivelmente serão incompatíveis com o modelo que será usado posteriormente.

Normalmente, junto com o processo de transformação de dados também se utilizam técnicas de visualização para observar como os dados estão se comportanto, o que pode facilitar a decisão pelo modelo mais adequado. As técnicas de visualização de dados são usadas não apenas nesta etapa, mas também após o desenvolvimento do modelo, de forma a permitir a análise crítica do modelo, observar correlações e identificar a qualidade geral do modelo.

Veremos como conseguimos realizar algumas operações de transformação e visualização de dados. Como no restante do curso, não é nossa pretensão cobrir todas as ferramentas e funções possíveis. No entanto, o conhecimento apresentado aqui poderá servir de base para o estudo de novas tecnologias disponíveis pelo mercado.

Para o desenvolvimento da aula, precisaremos dos seguintes pacotes em Python:

- pandas
- matplotlib

Durante o resto do curso veremos outros pacotes para nos auxiliar nessas tarefas, como numpy e seaborn, mas hoje vamos nos ater a essas duas.

Trabalhando com notebooks

Antes de prosseguir com o estudo dos pacotes citados, vamos discutir um pouco sobre a ferramenta que estamos utilizando.

Histórico

O Google Colaboratory (ou simplesmente, Colab), é uma ferramenta que permite o desenvolvimento de **notebooks** para a criação de códigos Python com resposta imediata.

O Colab foi inspirado em uma outra ferramenta, chamada **Jupyter Notebook** (caso queira, instale o jupyter no python pelo pip: `python -m pip install jupyter`). O Jupyter é um framework que instala um servidor local na sua máquina, e é capaz de isolar o desenvolvimento de uma aplicação em diversas linguagens, como Python, Javascript, R, entre outras.

Por sua vez, o Jupyter foi inspirado no **IPython**, que é um interpretador interativo capaz de executar diversas linguagens, mas é especialmente focado em Python. É por isso que, até hoje, a extensão utilizada nos notebooks é **.ipynb**, de *IPython Notebook*.

Vantagens

O uso de notebooks é largamente aceito para aplicações de Data Science, Data Mining e Machine Learning, pois:

- É open source, ou seja, é gratuito e permite extensão e evolução própria;
- É eficiente para uma rápida prototipagem e edição dos dados extraídos, sendo possível observar os resultados das chamadas das funções de forma imediata, sem ser necessário finalizar o código para ter resultados parciais;
- Possui uma execução simples em blocos. Percebeu que chamou uma função de forma errada 20 linhas acima? Sem problemas, basta editar o bloco que contém a linha errada e continuar rodando o projeto;
- É muito fácil gerar relatórios com notebooks, já que ele permite incluir imagens ao longo do desenvolvimento do código, bem como incluir textos em markdown para facilitar a explicação do processo de entendimento do conteúdo;
- É um ambiente isolado, favorecendo o trabalho em servidores. Caso seja necessário mais poder computacional, basta subir uma instância do Jupyter em um servidor e acessá-lo remotamente.

Em particular, o Colab possui uma vantagem adicional, que é não exigir nenhuma configuração prévia. A vasta maioria dos pacotes externos utilizados em Ciência de Dados já vem instalada nos servidores, o que permite acessar o site e começar a programar.

Desvantagens

No entanto, sempre temos algumas desvantagens. Abaixo são apresentadas algumas delas:

- É muito difícil versionar código escrito em Jupyter. Isso acontece porque o arquivo .ipynb mistura o código fonte com os resultados do notebook. Então pode haver uma mudança nos dados analisados e parecerá que o código foi alterado. Para projetos de longo prazo, isso pode tornar a solução inviável;
- Apesar de já haver extensões no VSCode para editar e visualizar notebooks, ainda possui uma fraca integração com IDEs, dificultando a correção de estilo e *linting*. Isso dificulta o trabalho colaborativo, pois pessoas tendem a usar estilos de programação distintos;
- É muito difícil criar testes para notebooks. Como a estrutura de células favorece muito o código estruturado, sem a divisão em funções, é inviável considerar a criação de testes unitários e testes funcionais;
- Os notebooks trabalham com uma estrutura não linear, isto é, é possível trabalhar com as células linearmente, porém também é possível retornar a uma célula anterior e editá-la. O problema é que, caso a pessoa que desenvolveu o código não rode as células seguintes

novamente, a alteração original não terá efeito. Entender isso requer um pouco de prática e hábito na ferramenta;
- Notebooks são ruins para lidar com grandes tarefas assíncronas. Por exemplo, deixar um notebook rodando durante a noite para extrair dados de um site de notícias é algo inviável. O servidor vai terminar, eventualmente, sendo interrompido. No Colab, em particular, há uma limitação de tempo para execução dos *kernels* (que veremos em seguida).

Todas essas desvantagens significam que utilizar um notebook não é uma boa solução? Ao contrário. Não temos uma solução "bala de prata". Conhecendo bem as desvantagens dos notebooks, é possível explorar corretamente as vantagens e obter sucesso nos projetos.

Pessoalmente, utilizo notebooks para prototipar projetos e fazer uma análise exploratória básica dos dados, bem como desenvolver alguns modelos simples de machine learning. Tendo entendido o problema e chegando nos resultados que planejei, eu levo todo o código desenvolvido para um script Python convencional, onde otimizo o projeto, crio testes, rodo o *linting* e faço o versionamento adequado do código.

Usando markdown para a elaboração dos relatórios

Como falamos, uma das vantagens dos notebooks é de permitir o desenvolvimento de relatórios legíveis, intercalando células de código com células de texto.

Para o desenvolvimento dos textos, podemos utilizar uma sintaxe chamada **markdown**. A próxima célula apresenta algumas dicas de sintaxe para formatar o seu texto.

Markdown

Markdown é uma linguagem de markup. HTML (hypertext markup language).

Título de nível 2

Título de nível 3

Esse é um texto de parágrafo.

Parágrafos precisam ser separados por uma linha vazia.

Adicione ênfase com **negrito** ou **negrito**, ou *itálico* ou *itálico*.

- Podemos incluir listas.
    - Que podem ser indentadas.

1. Listas também podem ser numeradas.
2. Para listas ordenadas.

[Também é possível incluir hyperlinks]

Código em linha usa crases: `foo()`, e blocos de códigos usam três crases:

```
print("olá, mundo!")
```

Ou então podem ter um endentação de 4 espaços:

```
foo()
```

- Seguir o site https://daringfireball.net/projects/markdown/syntax para mais informações

Extraindo dados de exemplo

Para podermos aplicar algumas técnicas de transformação de dados, vamos extrair alguns dados de exemplo.

Esses dados são de empresas listadas na Fortune 500. A revista Fortune tem esses dados distribuído de forma bem desigual, e felizmente alguém reuniu tudo em um único repositório no Github.

Como estamos trabalhando com um código que está rodando na nuvem, vamos fazer o download dos arquivos .csv. Acessando o repositório no Github, vemos que os dados estão armazenados em uma pasta `csv`, e os arquivos são nomeados no formato `fortune500-<ano>.csv`. Podemos usar isso ao nosso favor na hora de baixar esses dados.

Navegando pelo Github, conseguimos acessar a URL com os dados "crus", ou seja, com o .csv puro. Essa URL segue o seguinte modelo:

https://raw.githubusercontent.com/cmusam/fortune500/master/csv/fortune500-1955.csv

Para acessar o .csv de outro ano, basta trocar o 1955 pelo ano desejado. Essa base possui dados de 1955 até 2019.

Vamos então rodar um programa que faça o seguinte:

- Baixe os dados de cada ano individualmente;
- Inclua o campo ano em cada linha;
- Agregue todos os dados em uma única lista;
- Salve essa lista em um arquivo .csv.

Portanto, vamos começar com o código!

```
import csv  # vamos usar para salvar o arquivo .csv
import requests


MAIN_URL = "https://raw.githubusercontent.com/cmusam/fortune500/master/csv/fortune500-"
```

```
# Vamos fazer alguns testes primeiro...
url = f"{MAIN_URL}1955.csv"
req = requests.get(url)
req.text
```

```
'rank,company,revenue ($ millions),profit ($ millions)\r\n1,General Motors,9823.5,80
il,5661.4,584.8\r\n3,U.S. Steel,3250.4,195.4\r\n4,General Electric,2959.1,212.6\r\n5
1\r\n6,Chrysler,2071.6,18.5\r\n7,Armour,2056.1,1.6\r\n8,Gulf Oil,1705.3,182.8\r\n9,M
\r\n10,DuPont,1687.7,344.4\r\n11,Amoco,1667.4,132.8\r\n12,Bethlehem Steel,1660.3,117
0,84.6\r\n14,Texaco,1574.4,226.1\r\n15,AT&T Technologies,1526.2,55.8\r\n16,Shell Oil
7,Kraft,1210.3,37.4\r\n18,ChevronTexaco,1113.3,211.9\r\n19,Goodyear Tire & Rubber,10
eing,1033.2,37.0\r\n21,Sinclair Oil,1021.5,91.6\r\n22,Navistar International,994.1,3
0,40.5\r\n24,Union Carbide,923.7,89.8\r\n25,Firestone Tire & Rubber,916.0,40.5\r\n26
915.2,36.2\r\n27,Procter & Gamble,911.0,52.3\r\n28,Republic Steel,846.3,52.9\r\n29,C
```

```
# A quebra de linha está sendo feita com os caracteres \r\n. Podemos dividir
# esse texto em uma lista usando a função split().

data = req.text.split("\r\n")
print(data)
print(len(data))
```

```
['rank,company,revenue ($ millions),profit ($ millions)', '1,General Motors,9823.5,8
502
```

```
# Esse tipo de quebra de linha é relativamente comum, mas precisamos ver se
# todos os arquivos estão formatados da mesma forma.
# Podemos fazer isso usando a função split() para dividir na quebra de linha,
# e em seguida verificar se o comprimento da lista é de 502 elementos (as
# 500 empresas, mais o cabeçalho).
# Além disso, uma outra observação importante de se fazer é procurar por
# vírgulas adicionais nos dados. Veja que normalmente as vírgulas aparecem
# como ", ", com um espaço logo em seguida.
# É preciso fazer isso para não termos problema na hora de montar o .csv.

for year in range(1955, 2020):
  url = f"{MAIN_URL}{year}.csv"
  req = requests.get(url)
  data = req.text.split("\r\n")
  if ", " in req.text:
    print(f"Ano {year} com vírgulas no conteúdo!")
  if len(data) != 502:
    print(f"Erro no ano {year}! Total de dados: {len(data)}")
    continue

  print(f"Ano {year} ok!")
```

```
Ano 1955 com vírgulas no conteúdo!
Ano 1955 ok!
Ano 1956 com vírgulas no conteúdo!
Ano 1956 ok!
Ano 1957 com vírgulas no conteúdo!
Ano 1957 ok!
Ano 1958 com vírgulas no conteúdo!
Ano 1958 ok!
Ano 1959 com vírgulas no conteúdo!
Ano 1959 ok!
```

```
Ano 1960 com vírgulas no conteúdo!
Ano 1960 ok!
Ano 1961 com vírgulas no conteúdo!
Ano 1961 ok!
Ano 1962 com vírgulas no conteúdo!
Ano 1962 ok!
Ano 1963 com vírgulas no conteúdo!
Ano 1963 ok!
Ano 1964 com vírgulas no conteúdo!
Ano 1964 ok!
Ano 1965 com vírgulas no conteúdo!
Ano 1965 ok!
Ano 1966 com vírgulas no conteúdo!
Ano 1966 ok!
Ano 1967 com vírgulas no conteúdo!
Ano 1967 ok!
Ano 1968 com vírgulas no conteúdo!
Ano 1968 ok!
Ano 1969 com vírgulas no conteúdo!
Ano 1969 ok!
Ano 1970 com vírgulas no conteúdo!
Ano 1970 ok!
Ano 1971 com vírgulas no conteúdo!
Ano 1971 ok!
Ano 1972 com vírgulas no conteúdo!
Ano 1972 ok!
Ano 1973 com vírgulas no conteúdo!
Ano 1973 ok!
Ano 1974 com vírgulas no conteúdo!
Ano 1974 ok!
Ano 1975 com vírgulas no conteúdo!
Ano 1975 ok!
Ano 1976 com vírgulas no conteúdo!
Ano 1976 ok!
Ano 1977 com vírgulas no conteúdo!
Ano 1977 ok!
Ano 1978 ok!
Ano 1979 ok!
Ano 1980 ok!
Ano 1981 ok!
Ano 1982 ok!
Ano 1983 ok!
Ano 1984 ok!
Ano 1985 ok!
Ano 1986 ok!
Ano 1987 ok!
Ano 1988 ok!
Ano 1989 ok!
Ano 1990 ok!
```

```
# Apenas o ano 2019 está errado. Vamos analisar o seu conteúdo.
url = f"{MAIN_URL}2019.csv"
req = requests.get(url)
req.text
```

```
'rank,company,revenue ($ millions),profit ($ millions)\n1,Walmart,514405.0,6670.0\n2
2.0,20840.0\n3,Apple,265595.0,59531.0\n4,Berkshire Hathaway,247837.0,4021.0\n5,Amazo
```

```
# A quebra de linha é \n, e não \r\n. Portanto, nosso código precisa prever
# esse cenário.
# Veja também que o conteúdo não termina com \n, portanto não há uma quebra de
# linha no final. Ou seja, ao rodarmos a função split() a lista terá apenas
# 501 elementos.
len(req.text.split("\n"))
```

```
501
```

```
# Sendo assim, vamos elaborar uma função que leia o ano e uma dada lista,
# extraia os dados e os inclua na lista, fazendo os tratamentos necessários.
def collect_year_data(year, fortune_data):
  print(f"Coletando dados do ano {year}...")
  url = f"{MAIN_URL}{year}.csv"
  req = requests.get(url)
  # Não queremos as linhas de título, por isso vamos usar o slicing:
  raw_data = req.text.replace("\r", "").split("\n")[1:]

  # Se a última linha for vazia, vamos usar a função pop():
  if raw_data[-1] == "":
    raw_data.pop()

  # Vamos usar uma compreensão de lista para adicionar os anos no início de
  # cada linha.
  fortune_data.extend([f"{year},{data}" for data in raw_data])
  print(f"{len(raw_data)} dados extraídos!")


# Vamos também criar uma função para salvar a dada lista no formato .csv.
def save_to_csv(data):
  print("\nSalvando arquivo...")

  # Adicionando um cabeçalho:
  data = ["year,rank,company,revenue ($ millions),profit ($ millions)"] + data

  # Para usarmos a função que escreve no arquivo .csv, vamos precisar substituir
  # as entradas que possuam ", " ou aspas duplas por outros caracteres.
  data = [row.replace(", ", "; ").replace("\"", "") for row in data]

  with open("fortune500.csv", "w", newline="", encoding="utf-8") as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=",")
    for row in data:
      spamwriter.writerow(row.split(","))


# Agora vamos criar o código para rodar as funções anteriores.
full_data = []
for year in range(1955, 2020):
  collect_year_data(year, full_data)

save_to_csv(full_data)
print("Pronto!")
```

```
Coletando dados do ano 1955...
500 dados extraídos!
Coletando dados do ano 1956...
500 dados extraídos!
Coletando dados do ano 1957...
```

```
500 dados extraídos!
Coletando dados do ano 1958...
500 dados extraídos!
Coletando dados do ano 1959...
500 dados extraídos!
Coletando dados do ano 1960...
500 dados extraídos!
Coletando dados do ano 1961...
500 dados extraídos!
Coletando dados do ano 1962...
500 dados extraídos!
Coletando dados do ano 1963...
500 dados extraídos!
Coletando dados do ano 1964...
500 dados extraídos!
Coletando dados do ano 1965...
500 dados extraídos!
Coletando dados do ano 1966...
500 dados extraídos!
Coletando dados do ano 1967...
500 dados extraídos!
Coletando dados do ano 1968...
500 dados extraídos!
Coletando dados do ano 1969...
500 dados extraídos!
Coletando dados do ano 1970...
500 dados extraídos!
Coletando dados do ano 1971...
500 dados extraídos!
Coletando dados do ano 1972...
500 dados extraídos!
Coletando dados do ano 1973...
500 dados extraídos!
Coletando dados do ano 1974...
500 dados extraídos!
Coletando dados do ano 1975...
500 dados extraídos!
Coletando dados do ano 1976...
500 dados extraídos!
Coletando dados do ano 1977...
500 dados extraídos!
Coletando dados do ano 1978...
500 dados extraídos!
Coletando dados do ano 1979...
500 dados extraídos!
Coletando dados do ano 1980...
500 dados extraídos!
Coletando dados do ano 1981...
500 dados extraídos!
Coletando dados do ano 1982...
500 dados extraídos!
Coletando dados do ano 1983...
500 dados extraídos!
Coletando dados do ano 1984...
```

Conseguimos baixar todos os dados! Percebam que os dados ainda estão crus, então precisamos passar por um processo de transformação para formatá-los e deixá-los prontos para as análises.

Observe também que fizemos várias etapas antes de criarmos de fato as funções. Essa é a principal vantagem no uso de notebooks. Podemos ir explorando os dados aos poucos e, depois que estivermos confortáveis, podemos elaborar as funções que vão, de fato, realizar o trabalho que precisamos.

Transformando dados com pandas

Vamos ver agora algumas funções importantes na hora de trabalhar com transformação de dados. Como tudo em programação, sempre temos várias formas de resolver um mesmo problema, então o que vai ser apresentado aqui é apenas uma parte do potencial desse pacote. Ao longo do curso veremos novas funções e aprofundaremos o conhecimento necessário para transformar dados.

Para essa etapa do processo, vamos usar um pacote chamado **pandas**. Esse pacote possui inúmeras funções que manipulam e organizam massas de dados. Os dados ficam armazenados em uma estrutura chamada *DataFrame*. Um DataFrame é basicamente uma tabela com vários recursos.

Vamos começar então o nosso processo.

```
import pandas as pd

# Vamos criar um DataFrame com os nossos dados.
df = pd.read_csv("fortune500.csv")

# Usamos as funções head() e tail() para imprimir as primeiras ou as últimas
# linhas, respectivamente.
df.head()
```

	year	rank	company	revenue ($ millions)	profit ($ millions)
0	1955	1	General Motors	9823.5	806.0
1	1955	2	Exxon Mobil	5661.4	584.8
2	1955	3	U.S. Steel	3250.4	195.4
3	1955	4	General Electric	2959.1	212.6
4	1955	5	Esmark	2510.8	19.1

```
# Por padrão, o pandas exibe as primeiras ou últimas 5 linhas do DataFrame.
# Podemos alterar isso passando um parâmetro numérico
df.tail(10)
```

	year	rank	company	revenue ($ millions)	profit ($ millions)
32490	2019	491	First American Financial	5747.8	474.5
32491	2019	492	Harley-Davidson	5716.9	531.5
32492	2019	493	Windstream Holdings	5713.1	-723.0
32493	2019	494	Yum Brands	5688.0	1542.0
32494	2019	495	Williams-Sonoma	5671.6	333.7
32495	2019	496	Simon Property Group	5657.9	2440.1
32496	2019	497	Navient	5610.0	395.0

```
# Vamos renomear as colunas. O ideal é não colocar espaços ou caracteres
# especiais.
df.columns = ["year", "rank", "company", "revenue", "profit"]

# Vamos ver quantas entradas temos no DataFrame:
len(df)
```

```
32500
```

```
# Ok, esse tamanho é o que esperamos (65 anos, 500 empresas por ano).
# Vamos verificar se os nossos dados foram importados com os tipos adequados:
df.dtypes
```

```
year          int64
rank          int64
company      object
revenue     float64
profit       object
dtype: object
```

```
# Veja que "revenue" está com um formato numérico, mas "profit", não.
# Era esperado que "profit" também fosse um float. Isso pode indicar que
# provavelmente há um valor não inteiro.
# Vamos forçar converter os dados da coluna profit para numéricos, e vamos
# definir que eventuais erros sejam convertidos para NaN (Not a Number):
df.profit = pd.to_numeric(df.profit, errors="coerce")

# Depois disso, vamos remover novamente todos os
df = df.dropna()
len(df)
```

```
32124
```

```
# Tivemos 376 linhas removidas. Não foram poucas, mas considerando o nosso
# volume de dados, é razoável.
# Existem outras técnicas possíveis. Poderíamos substituir os valores não
# numéricos por 0 ou pela média daquele ano, ou usar alguma outra técnica.
# Tudo vai depender do projeto. Durante o curso vamos ver alguns outros
# tipos de tratamento de dados.
df.dtypes
```

```
year         int64
rank         int64
company     object
revenue    float64
profit     float64
dtype: object
```

```
# Ok, nossos dados estão formatados. A partir daí podemos fazer análises ou
# exibir esses dados em gráficos, como vamos fazer logo mais.
# Vamos ver agora mais algumas funções no pandas.

# Vamos criar alguns dados para podermos utilizar no pandas como um DataFrame.
# Vamos começar criando um dicionário:

data = {
    "Nome": ["Ana", "Pedro", "João", "Renato", "Alice"],
    "Idade": [22, 23, 20, 19, 22]
}

# Agora vamos criar um dataframe a partir dos dados:

students = pd.DataFrame(data)
students
```

	Nome	Idade
0	Ana	22
1	Pedro	23
2	João	20
3	Renato	19
4	Alice	22

```
# Selecionando uma coluna, retornando um DataFrame:

students[["Nome"]]
```

	Nome
0	Ana
1	Pedro
2	João
3	Renato
4	Alice

```
# Selecionando uma linha:

row = students.loc[1]
row
```

```
Nome      Pedro
```

```
Idade       23
Name: 1, dtype: object
```

```
# Você pode tratar um DataFrame de forma semântica, como um dicionário.
# Recuperar, definir e excluir colunas funciona com a mesma sintaxe de
# dicionários:

students["Idade"]
```

```
0    22
1    23
2    20
3    19
4    22
Name: Idade, dtype: int64
```

```
del students["Idade"]
students
```

	Nome
0	Ana
1	Pedro
2	João
3	Renato
4	Alice

```
# Posicao, Nome_Coluna, Valor
students.insert(1, "Idade", [22, 23, 20, 19, 22])
students
```

	Nome	Idade
0	Ana	22
1	Pedro	23
2	João	20
3	Renato	19
4	Alice	22

```
# Com muita frequência vamos encontrar tabelas com valores ausentes.
# Podemos identificar esses conjuntos usando a função `isnull()`.
import numpy as np  # Só vamos usar a numpy aqui para simular um valor nulo.

data = {
    "Nome": ["Ana", np.nan],
    "Idade": [22, np.nan]
}

students = pd.DataFrame(data)
```

```
students
```

	Nome	Idade
0	Ana	22.0
1	NaN	NaN

```
students.isnull()
```

	Nome	Idade
0	False	False
1	True	True

```
students.fillna(0)
```

	Nome	Idade
0	Ana	22.0
1	0	0.0

```
# Renomeando as colunas de um DataFrame:
data = {
    "NOMe": ["Ana", "Pedro"],
    "IDD": [22, 23]
}

students = pd.DataFrame(data)
students
```

	NOMe	IDD
0	Ana	22
1	Pedro	23

```
newcols = {
    "NOMe": "Nome",
    "IDD": "Idade"
}

students.rename(columns=newcols, inplace=True)
students
```

	Nome	Idade
0	Ana	22
1	Pedro	23

```
students.columns = ["name", "age"]
students
```

```
students
```

	name	age
0	Ana	22
1	Pedro	23

Visualizando dados com matplotlib

O matplotlib é um pacote extremamente poderoso para a elaboração de gráficos. Ele possui fácil integração com pandas, e várias extensões, como o seaborn (que vamos ver brevemente na aula de hoje, e será mais explorado ao longo do curso).

Assim como o pandas e os demais pacotes para manipulação de dados, existem inúmeros pacotes voltados para a elaboração de gráficos, muitas vezes mais sofisticados que o matplotlib.

Como fizemos com o pandas, vamos fazer um caso aplicado com os dados da Fortune 500, e depois veremos mais alguns gráficos possíveis. Ao longo do curso veremos outras aplicações interessantes da matplotlib.

[ ] ↳ *17 células ocultas*