Módulo de Programação Python

Trilha Python - Aula 22/23: Revisão e Resolução de Problemas

Objetivo:

Revisar e consolidar o conteúdo de **Pandas** avançado.

Revisão sobre persistência da dados em Pandas.

Nas Instruções práticas desta semana trabalhamos com informação armazenada em arquivos **CSV** obtidos de diferentes fontes.

Para análise de dados e aprendizado de máquina, é uma prática comum armazenar os dados e modelos em formato **CSV**.

Embora o formato **CSV** ajude a armazenar dados em um formato tabular retangular, ele nem sempre é o mais adequado para guardar todos os `Dataframe` s do **Pandas**.

Os arquivos **CSV** tendem a ser lentos para leitura e gravação, ocupam mais memória e espaço e, o mais importante, os CSVs não armazenam informações sobre os tipos de dados.

Como o CSV lida com diferentes formatos de arquivo?

O **Pandas** oferece suporte a uma ampla variedade de formatos e subformatos de dados para facilitar o trabalho com grandes conjuntos de dados.

Alguns dos formatos mais populares são `object`, `string`, `timedelta`, `int`, `float` etc.

Pandas usam arrays **NumPy** para armazenar os dados. Portanto, cada tipo de dados como `int` é armazenado na forma de `int8`, `int16` etc, para maior eficiência.

Porém, o **CSV** não armazena informações sobre os tipos de dados.

O usuário precisa especificá-lo com o método `read_csv` e tipos de dados como `timedelta` são armazenados como strings.

In [1]:
```
# Vamos ler um csv com Pandas
import pandas as pd
import numpy as np
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions
4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated.
Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Adva
nced Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions
4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated.
Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Adva
nced Vector Extensions (Intel(R) AVX) instructions.
```

In [2]:
```
import random
import string
import datetime
import os
```

In [3]:
```
def gera_strings(tamanho):
    '''
    Função para gerar palavras aleatória de determinado tamanho
    '''
    letras = string.ascii_letters
    palavra = ''.join([random.choice(letras)for i in range(random.r
    return palavra

def get_sentence(tamanho):
    '''
    Função para gerar frases aleatórias de determinado tamanho
    '''
    sentence = ' '.join([gera_strings(random.randint(3,10)) for i i
    return sentence

def gera_data(inicio, fim, formato):
```

```
    '''
    Função para gerar datas aleatórias em um intervalo
    '''
    inicio = datetime.datetime.strptime(inicio, formato)
    fim  = datetime.datetime.strptime(fim, formato)
    total = fim - inicio
    int_total = (total.days * 24 * 60 * 60) + total.seconds
    random_second = random.randrange(int_total)
    return(inicio + datetime.timedelta(seconds=random_second)).strf

def random_data(size):
    data = []
    for i in range(size):
        data.append(
            [random.randint(-127,127),
             random.randint(-32768,32767),
             random.randint(-2147483648,2147483647),
             random.randint(-9223372036854775808 ,92233720368547758
             random.randint(0,255),
             round(random.uniform(0,10000),2),
             round(random.uniform(0,1000000),2),
             gera_strings(10),
             get_sentence(5),
             random.choice([gera_strings(10) for i in range(25)]),
             gera_data("1900-01-01","2020-05-01","%Y-%m-%d"),
             gera_data("1900-01-01T00:00:00","2020-05-01T23:59:59",'
             random.choice([True,False])])
    return data

def gera_indice(size):
    data = []
    cont = 0
    while cont < size:
        strIndex = gera_strings(5)
        strIndex = strIndex.upper() + str(random.randint(0,1000))
        while strIndex in data:
            strIndex = gera_strings(5)
            strIndex = strIndex.upper() + str(random.randint(0,1000
        data.append(strIndex)
        cont += 1
    return data

def test_data(size, drop_timedelta, drop_timezone):
    data = random_data(size)
    index = gera_indice(size)
    labels = ["Int8", "Int16", "Int32", "Int64", "UInt8",
              "Float32", "Float64", "String", "Sentence",
              "Category", "Date", "DateTime", "Bool"]

    df = pd.DataFrame(data,index=index, columns=labels)

    df["Int8"] = df["Int8"].astype("int8")
    df["Int16"] = df["Int16"].astype("int16")
    df["Int32"] = df["Int32"].astype("int32")
```

```
    df["UInt8"] = df["UInt8"].astype("uint8")
    df["Float32"] = df["Float32"].astype("float32")
    df["Category"] = df["Category"].astype("category")
    df["Date"] = pd.to_datetime(df["Date"], format="%Y-%m-%d")
    df["DateTime"] = pd.to_datetime(df["DateTime"], format="%Y-%m-%
    if not drop_timedelta:
        df["TimeDelta"] = df["DateTime"]-df["Date"]
    if not drop_timezone:
        df["DateTime+Zone"] = df["DateTime+Zone"].dt.tz_localize('E
    return df
```

In [4]:
```
dataset_size = 50000
df= test_data(dataset_size, True, True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 50000 entries, AKT643 to WSKY494
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Int8      50000 non-null  int8
 1   Int16     50000 non-null  int16
 2   Int32     50000 non-null  int32
 3   Int64     50000 non-null  int64
 4   UInt8     50000 non-null  uint8
 5   Float32   50000 non-null  float32
 6   Float64   50000 non-null  float64
 7   String    50000 non-null  object
 8   Sentence  50000 non-null  object
 9   Category  50000 non-null  category
 10  Date      50000 non-null  datetime64[ns]
 11  DateTime  50000 non-null  datetime64[ns]
 12  Bool      50000 non-null  bool
dtypes: bool(1), category(1), datetime64[ns](2), float32(1), float
64(1), int16(1), int32(1), int64(1), int8(1), object(2), uint8(1)
memory usage: 4.9+ MB
```

In [5]:
```
df.head()
```

Out[5]:

	Int8	Int16	Int32	Int64	UInt8	Float32	Float64
AKT643	-15	17384	302470034	2061692868925798585	112	1585.810059	89400.15
IQXX645	-106	-5531	-1851242249	-2890108420681993231	247	3873.760010	928189.79
DRLT443	124	-31021	-1967907470	2784788926253526540	211	3612.439941	225114.03
TOE630	118	-16523	2117301076	-9181007834582809404	49	1444.910034	476708.12
BYBJ467	22	12378	672685054	3866551225166128599	49	4889.140137	597670.18

In [6]:
```
%timeit -n 1 -r 1  df.to_csv("test_data.csv", index=True, index_lab
%timeit -n 1 -r 1  pd.read_csv("test_data.csv", index_col="Index")
dfCSV = pd.read_csv("test_data.csv")
dfCSV.info()
dfCSV.head()
```

```
320 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
122 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Index     50000 non-null  object
 1   Int8      50000 non-null  int64
 2   Int16     50000 non-null  int64
 3   Int32     50000 non-null  int64
 4   Int64     50000 non-null  int64
 5   UInt8     50000 non-null  int64
 6   Float32   50000 non-null  float64
 7   Float64   50000 non-null  float64
 8   String    50000 non-null  object
 9   Sentence  50000 non-null  object
 10  Category  50000 non-null  object
 11  Date      50000 non-null  object
 12  DateTime  50000 non-null  object
 13  Bool      50000 non-null  bool
dtypes: bool(1), float64(2), int64(5), object(6)
memory usage: 5.0+ MB
```

Out[6]:

	Index	Int8	Int16	Int32	Int64	UInt8	Float32	Float64
0	AKT643	-15	17384	302470034	2061692868925798585	112	1585.81	89400.15
1	IQXX645	-106	-5531	-1851242249	-2890108420681993231	247	3873.76	928189.79
2	DRLT443	124	-31021	-1967907470	2784788926253526540	211	3612.44	225114.03
3	TOE630	118	-16523	2117301076	-9181007834582809404	49	1444.91	476708.12
4	BYBJ467	22	12378	672685054	3866551225166128599	49	4889.14	597670.18

In [7]:
```
file_stats = os.stat("test_data.csv")
print("Tamanho do arquivo: {} bytes".format(file_stats.st_size))
```

```
Tamanho do arquivo: 7238155 bytes
```

In [8]:
```
%timeit -n 1 -r 1  df.to_csv("test_data.zip", index=True, index_lab
%timeit -n 1 -r 1  pd.read_csv("test_data.zip", index_col="Index",
dfZIP = pd.read_csv("test_data.zip")
dfZIP.info()
dfZIP.head()
```

```
549 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
145 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Index     50000 non-null  object
 1   Int8      50000 non-null  int64
 2   Int16     50000 non-null  int64
 3   Int32     50000 non-null  int64
 4   Int64     50000 non-null  int64
 5   UInt8     50000 non-null  int64
 6   Float32   50000 non-null  float64
 7   Float64   50000 non-null  float64
 8   String    50000 non-null  object
 9   Sentence  50000 non-null  object
 10  Category  50000 non-null  object
 11  Date      50000 non-null  object
 12  DateTime  50000 non-null  object
 13  Bool      50000 non-null  bool
dtypes: bool(1), float64(2), int64(5), object(6)
memory usage: 5.0+ MB
```

Out[8]:

	Index	Int8	Int16	Int32	Int64	UInt8	Float32	Float64
0	AKT643	-15	17384	302470034	2061692868925798585	112	1585.81	89400.15
1	IQXX645	-106	-5531	-1851242249	-2890108420681993231	247	3873.76	928189.79
2	DRLT443	124	-31021	-1967907470	2784788926253526540	211	3612.44	225114.03
3	TOE630	118	-16523	2117301076	-9181007834582809404	49	1444.91	476708.12
4	BYBJ467	22	12378	672685054	3866551225166128599	49	4889.14	597670.18

In [9]:
```
file_stats = os.stat("test_data.zip")
print("Tamanho do arquivo: {} bytes".format(file_stats.st_size))
```

```
Tamanho do arquivo: 4340902 bytes
```

Outros métodos de armazenamento diferentes de CSV

1. **Pickle**: Trata-se do formato nativo de **Python**, popular para serialização de objetos. A vantagem do **Pickle** é que ele permite que o código **Python** implemente qualquer tipo de melhoria. É muito mais rápido quando comparado aos arquivos **CSV** e reduz o tamanho do arquivo para quase metade. se comparado ao arquivo **CSV**, usando suas técnicas de compactação. Além disso, não há necessidade de especificar vários parâmetros para cada coluna de dados. A maneira de implementar isso é simples.

   ```
   dataframe.to_pickle(caminho)
   ```

   ```
   Caminho: onde os dados serão armazenados
   ```

In [11]:
```
%timeit -n 1 -r 1  df.to_pickle("test_data.pkl")
%timeit -n 1 -r 1  pd.read_pickle("test_data.pkl")
dfPickle = pd.read_pickle("test_data.pkl")
dfPickle.info()
dfPickle.head()
```

```
34.8 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
15 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
<class 'pandas.core.frame.DataFrame'>
Index: 50000 entries, AKT643 to WSKY494
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Int8      50000 non-null  int8
 1   Int16     50000 non-null  int16
 2   Int32     50000 non-null  int32
 3   Int64     50000 non-null  int64
 4   UInt8     50000 non-null  uint8
 5   Float32   50000 non-null  float32
 6   Float64   50000 non-null  float64
 7   String    50000 non-null  object
 8   Sentence  50000 non-null  object
 9   Category  50000 non-null  category
 10  Date      50000 non-null  datetime64[ns]
 11  DateTime  50000 non-null  datetime64[ns]
 12  Bool      50000 non-null  bool
dtypes: bool(1), category(1), datetime64[ns](2), float32(1), float
64(1), int16(1), int32(1), int64(1), int8(1), object(2), uint8(1)
memory usage: 3.9+ MB
```

Out[11]:

	Int8	Int16	Int32	Int64	UInt8	Float32	Float64
AKT643	-15	17384	302470034	2061692868925798585	112	1585.810059	89400.15
IQXX645	-106	-5531	-1851242249	-2890108420681993231	247	3873.760010	928189.79
DRLT443	124	-31021	-1967907470	2784788926253526540	211	3612.439941	225114.03
TOE630	118	-16523	2117301076	-9181007834582809404	49	1444.910034	476708.12
BYBJ467	22	12378	672685054	3866551225166128599	49	4889.140137	597670.18

In [12]:
```
file_stats = os.stat("test_data.pkl")
print("Tamanho do arquivo: {} bytes".format(file_stats.st_size))
```

```
Tamanho do arquivo: 5149469 bytes
```

In [13]:
```
%timeit -n 1 -r 1  df.to_pickle("test_data.pkl.bz2", compression="b
%timeit -n 1 -r 1  pd.read_pickle("test_data.pkl.bz2")
dfPickle = pd.read_pickle("test_data.pkl.bz2")
dfPickle.info()
dfPickle.head()
```

```
430 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
196 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
<class 'pandas.core.frame.DataFrame'>
Index: 50000 entries, AKT643 to WSKY494
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Int8      50000 non-null  int8
 1   Int16     50000 non-null  int16
 2   Int32     50000 non-null  int32
 3   Int64     50000 non-null  int64
 4   UInt8     50000 non-null  uint8
 5   Float32   50000 non-null  float32
 6   Float64   50000 non-null  float64
 7   String    50000 non-null  object
 8   Sentence  50000 non-null  object
 9   Category  50000 non-null  category
 10  Date      50000 non-null  datetime64[ns]
 11  DateTime  50000 non-null  datetime64[ns]
 12  Bool      50000 non-null  bool
dtypes: bool(1), category(1), datetime64[ns](2), float32(1), float
64(1), int16(1), int32(1), int64(1), int8(1), object(2), uint8(1)
memory usage: 3.9+ MB
```

Out[13]:

	Int8	Int16	Int32	Int64	UInt8	Float32	Float64
AKT643	-15	17384	302470034	2061692868925798585	112	1585.810059	89400.15
IQXX645	-106	-5531	-1851242249	-2890108420681993231	247	3873.760010	928189.79
DRLT443	124	-31021	-1967907470	2784788926253526540	211	3612.439941	225114.03
TOE630	118	-16523	2117301076	-9181007834582809404	49	1444.910034	476708.12
BYBJ467	22	12378	672685054	3866551225166128599	49	4889.140137	597670.18

In [14]:
```
file_stats = os.stat("test_data.pkl.bz2")
print("Tamanho do arquivo: {} bytes".format(file_stats.st_size))
```

```
Tamanho do arquivo: 3708530 bytes
```

2. **Parquet**: Este é um formato de armazenamento compactado usado no ecossistema **Hadoop**. Ele permite serializar estruturas aninhadas complexas, suporta compactação e codificação por coluna e oferece leituras rápidas. A vantagem, em relação ao **CSV**, é que ele armazena informações sobre todos os tipos de dados, é mais rápido e oferece amplo suporte no ecossistema **Hadoop** permitindo uma filtragem rápida.

Para implementar em python:

```
dataframe.to_parquet(caminho, mecanismo, compression, inde
x, partição_cols)

Caminho: onde os dados são armazenados

Motor: motor pyarrow ou fastparquet

Compressão: permitindo escolher vários métodos de compressã
o

Índice: salva o índice do dataframe

Partition_cols: especifique a ordem de particionamento da c
oluna
```

In [16]:
```
%timeit -n 1 -r 1  df.to_parquet("test_data.parquet", index=True)
%timeit -n 1 -r 1  pd.read_parquet("test_data.parquet")
dfParquet = pd.read_parquet("test_data.parquet")
dfParquet.info()
dfParquet.head()
```

```
66.8 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
36.8 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
<class 'pandas.core.frame.DataFrame'>
Index: 50000 entries, AKT643 to WSKY494
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Int8      50000 non-null  int8
 1   Int16     50000 non-null  int16
 2   Int32     50000 non-null  int32
 3   Int64     50000 non-null  int64
 4   UInt8     50000 non-null  uint8
 5   Float32   50000 non-null  float32
 6   Float64   50000 non-null  float64
 7   String    50000 non-null  object
 8   Sentence  50000 non-null  object
 9   Category  50000 non-null  category
 10  Date      50000 non-null  datetime64[ns]
 11  DateTime  50000 non-null  datetime64[ns]
 12  Bool      50000 non-null  bool
dtypes: bool(1), category(1), datetime64[ns](2), float32(1), float
64(1), int16(1), int32(1), int64(1), int8(1), object(2), uint8(1)
memory usage: 4.9+ MB
```

Out[16]:

	Int8	Int16	Int32	Int64	UInt8	Float32	Float64
AKT643	-15	17384	302470034	2061692868925798585	112	1585.810059	89400.15
IQXX645	-106	-5531	-1851242249	-2890108420681993231	247	3873.760010	928189.79
DRLT443	124	-31021	-1967907470	2784788926253526540	211	3612.439941	225114.03
TOE630	118	-16523	2117301076	-9181007834582809404	49	1444.910034	476708.12
BYBJ467	22	12378	672685054	3866551225166128599	49	4889.140137	597670.18

In [17]:
```
file_stats = os.stat("test_data.parquet")
print("Tamanho do arquivo: {} bytes".format(file_stats.st_size))
```

```
Tamanho do arquivo: 5632134 bytes
```

In [15]:
```
%timeit -n 1 -r 1  df.to_parquet("test_data.parquet.gzip", index=Tr
%timeit -n 1 -r 1  pd.read_parquet("test_data.parquet.gzip")
dfParquet = pd.read_parquet("test_data.parquet.gzip")
dfParquet.info()
dfParquet.head()
```

```
1.75 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
2.44 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
<class 'pandas.core.frame.DataFrame'>
Index: 50000 entries, AKT643 to WSKY494
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Int8      50000 non-null  int8
 1   Int16     50000 non-null  int16
 2   Int32     50000 non-null  int32
 3   Int64     50000 non-null  int64
 4   UInt8     50000 non-null  uint8
 5   Float32   50000 non-null  float32
 6   Float64   50000 non-null  float64
 7   String    50000 non-null  object
 8   Sentence  50000 non-null  object
 9   Category  50000 non-null  category
 10  Date      50000 non-null  datetime64[ns]
 11  DateTime  50000 non-null  datetime64[ns]
 12  Bool      50000 non-null  bool
dtypes: bool(1), category(1), datetime64[ns](2), float32(1), float
64(1), int16(1), int32(1), int64(1), int8(1), object(2), uint8(1)
memory usage: 4.9+ MB
```

Out[15]:

	Int8	Int16	Int32	Int64	UInt8	Float32	Float64
AKT643	-15	17384	302470034	2061692868925798585	112	1585.810059	89400.15
IQXX645	-106	-5531	-1851242249	-2890108420681993231	247	3873.760010	928189.79
DRLT443	124	-31021	-1967907470	2784788926253526540	211	3612.439941	225114.03
TOE630	118	-16523	2117301076	-9181007834582809404	49	1444.910034	476708.12
BYBJ467	22	12378	672685054	3866551225166128599	49	4889.140137	597670.18

In [19]:
```
file_stats = os.stat("test_data.parquet.gzip")
print("Tamanho do arquivo: {} bytes".format(file_stats.st_size))
```

```
Tamanho do arquivo: 4545539 bytes
```