RA272746_aula6_ex6-5_cap6

April 25, 2024

0.1 IA376I – Tópicos em Engenharia de Computação VII

0.1.1 Tópico: Análise de Dados Visual (Visual Analytics)

Professora: Wu, Shin - Ting **Aluno:** Luiz Roberto Albano Junior **RA:** 272746

Aula 06 - 19/04/2024

0.1.2 Exercícios 6.5

Exercício: Leia o texto em https://www.kaggle.com/code/evertonsilva/data-wrangling-cleaning e sintetize os principais passos de preparação de dados para aprendizado de máquina. Compare-os como os passos apresentados neste capítulo.

No referido artigo o autor apresenta de forma prática um passo a passo de sua análise de um conjunto de dados a ser trabalhado.

Os principais pontos que destaco no tutorial, são: * Utilização de funções escritivas nos dados para exploração de hipóteses e tendências em relação aos dados * Utilização de gráficos para explorar a quantidade e/ou frequência dos valores, e a partir desta visualização decidir por descartar dados * Padronização dos dados (como nas colunas OS type e tipo de dispositivo)

O que achei interessante no tutorial é que através de um exemplo mais prático é possível enxergar caminhos para realização de uma análise no conjunto de dados a ser explorado e a partir destes passos conseguir enxergar questionamentos que podem ser realizados sobre o conjunto. O texto também apresenta correlação com as teorias apresentadas nas notas de aula, como: limpeza de dados, padronização, além da análise exploratória do conjunto de dados.

Exercício 2: Reproduza os exemplos fornecidos nos Capítulos 6, 7 e 8 em [89] (Python) ou nos Capítulos 9 a 16 em [90] (R). Em ambas as referências, são abordadas diversas funções adicionais de manipulação dos dados, além das apresentadas neste capítulo, proporcionando uma visão mais abrangente das capacidades das respectivas linguagens de programação.

Reprodução dos passos:

Capítulo 6: carregamento de dados

[ ]:
```
import urllib.request
from lxml import objectify

path = base_url + "datasets/mta_perf/Performance_MNR.xml"

with urllib.request.urlopen(path) as f:
    parsed = objectify.parse(f)

root = parsed.getroot()
```

[ ]:
```
import pandas as pd

#Base da URL do repositório com os arquivos de exemplo utilizados no livro
base_url = "https://raw.githubusercontent.com/wesm/pydata-book/3rd-edition/"

df = pd.read_csv(base_url + "examples/ex1.csv")
df
```

[ ]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

[ ]:
```
pd.read_csv(base_url + "examples/ex2.csv", header=None)
```

[ ]:

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

[ ]:
```
pd.read_csv(base_url + "examples/ex2.csv", names=["a", "b", "c", "d",␣
↪"message"])
```

[ ]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

[ ]:
```
names = ["a", "b", "c", "d", "message"]
pd.read_csv(base_url + "examples/ex2.csv", names=names, index_col="message")
```

[ ]:

	a	b	c	d
message				
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

[ ]:
```
parsed = pd.read_csv(base_url + "examples/csv_mindex.csv", index_col=['key1',␣
↪'key2'])
parsed
```

[ ]:
```
           value1  value2
key1 key2
one  a          1       2
     b          3       4
     c          5       6
     d          7       8
two  a          9      10
     b         11      12
     c         13      14
     d         15      16
```

[ ]:
```
result = pd.read_csv(base_url + "examples/ex3.txt", sep="\s+")
result
```

[ ]:
```
            A         B         C
aaa -0.264438 -1.026059 -0.619500
bbb  0.927272  0.302904 -0.032399
ccc -0.264273 -0.386314 -0.217601
ddd -0.871858 -0.348382  1.100491
```

[ ]:
```
pd.read_csv(base_url + "examples/ex4.csv", skiprows=[0, 2, 3])
```

[ ]:
```
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

[ ]:
```
result = pd.read_csv(base_url + "examples/ex5.csv")
result
```

[ ]:
```
  something  a   b     c   d message
0       one  1   2   3.0   4     NaN
1       two  5   6   NaN   8   world
2     three  9  10  11.0  12     foo
```

[ ]:
```
pd.isna(result)
```

[ ]:
```
   something      a      b      c      d  message
0      False  False  False  False  False     True
1      False  False  False   True  False    False
2      False  False  False  False  False    False
```

Não repliquei os demais passos sobre as representações de dados NaN porém realizei a leitura.

Lendo arquivos de texto em pedaços

[ ]:
```
pd.options.display.max_rows = 10
result = pd.read_csv(base_url + "examples/ex6.csv")
result
```

[ ]:
```
           one       two     three      four key
0     0.467976 -0.038649 -0.295344 -1.824726   L
1    -0.358893  1.404453  0.704965 -0.200638   B
2    -0.501840  0.659254 -0.421691 -0.057688   G
3     0.204886  1.074134  1.388361 -0.982404   R
4     0.354628 -0.133116  0.283763 -0.837063   Q
...        ...       ...       ...       ...  ..
9995  2.311896 -0.417070 -1.409599 -0.515821   L
9996 -0.479893 -0.650419  0.745152 -0.646038   E
9997  0.523331  0.787112  0.486066  1.093156   K
9998 -0.362559  0.598894 -1.843201  0.887292   G
9999 -0.096376 -1.012999 -0.657431 -0.573315   0

[10000 rows x 5 columns]
```

[ ]:
```
pd.read_csv(base_url + "examples/ex6.csv", nrows=5)
```

[ ]:
```
        one       two     three      four key
0  0.467976 -0.038649 -0.295344 -1.824726   L
1 -0.358893  1.404453  0.704965 -0.200638   B
2 -0.501840  0.659254 -0.421691 -0.057688   G
3  0.204886  1.074134  1.388361 -0.982404   R
4  0.354628 -0.133116  0.283763 -0.837063   Q
```

Lendo o arquivo em partes

[ ]:
```
chunker = pd.read_csv(base_url + "examples/ex6.csv", chunksize=1000)
type(chunker)
```

[ ]:
```
pandas.io.parsers.readers.TextFileReader
```

[ ]:
```
tot = pd.Series([], dtype='int64')
for piece in chunker:
    tot = tot.add(piece["key"].value_counts(), fill_value=0)

tot = tot.sort_values(ascending=False)

tot[:10]
```

[ ]:
```
key
E    368.0
X    364.0
```

```
L    346.0
O    343.0
Q    340.0
M    338.0
J    337.0
F    335.0
K    334.0
H    330.0
dtype: float64
```

Gravando arquivos

[ ]:
```
data = pd.read_csv(base_url + "examples/ex5.csv")
data
```

[ ]:
```
  something  a   b     c   d message
0       one  1   2   3.0   4     NaN
1       two  5   6   NaN   8   world
2     three  9  10  11.0  12     foo
```

[ ]:
```
data.to_csv("examples/out.csv")
!cat examples/out.csv
```

```
,something,a,b,c,d,message
0,one,1,2,3.0,4,
1,two,5,6,,8,world
2,three,9,10,11.0,12,foo
```

Os demais exemplos para gravação de arquivos foram lidos, porém não replicados.

Outros formatos de delimitadores

[ ]:
```
import csv
f = open("examples/ex7.csv")
reader = csv.reader(f)

for line in reader:
    print(line)

f.close()
```

```
['a', 'b', 'c']
['1', '2', '3']
['1', '2', '3']
```

[ ]:
```
with open("examples/ex7.csv") as f:
    lines = list(csv.reader(f))
    f.close()
```

```
header, values = lines[0], lines[1:]
data_dict = {h: v for h, v in zip(header, zip(*values))}
data_dict
```

[ ]:
```
{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

Dados Json

[ ]:
```
import json

obj = """
{"name": "Wes",
 "cities_lived": ["Akron", "Nashville", "New York", "San Francisco"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 34, "hobbies": ["guitars", "soccer"]},
              {"name": "Katie", "age": 42, "hobbies": ["diving", "art"]}]
}
"""

result = json.loads(obj)
result
```

[ ]:
```
{'name': 'Wes',
 'cities_lived': ['Akron', 'Nashville', 'New York', 'San Francisco'],
 'pet': None,
 'siblings': [{'name': 'Scott', 'age': 34, 'hobbies': ['guitars', 'soccer']},
  {'name': 'Katie', 'age': 42, 'hobbies': ['diving', 'art']}]}
```

[ ]:
```
asjson = json.dumps(result)
asjson
```

[ ]:
```
'{"name": "Wes", "cities_lived": ["Akron", "Nashville", "New York", "San
Francisco"], "pet": null, "siblings": [{"name": "Scott", "age": 34, "hobbies":
["guitars", "soccer"]}, {"name": "Katie", "age": 42, "hobbies": ["diving",
"art"]}]}'
```

[ ]:
```
siblings = pd.DataFrame(result["siblings"], columns=["name", "age"])
siblings
```

[ ]:
```
    name  age
0  Scott   34
1  Katie   42
```

[ ]:
```
data = pd.read_json(base_url + "examples/example.json")
data
```

[ ]:
```
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

Web scraping

[ ]:
```
tables = pd.read_html(base_url + "examples/fdic_failed_bank_list.html")
len(tables)
```

[ ]:
```
1
```

[ ]:
```
failures = tables[0]
failures.head()
```

[ ]:
```
                     Bank Name             City  ST   CERT  \
0                   Allied Bank          Mulberry  AR     91
1  The Woodbury Banking Company          Woodbury  GA  11297
2        First CornerStone Bank  King of Prussia  PA  35312
3            Trust Company Bank          Memphis  TN   9956
4    North Milwaukee State Bank        Milwaukee  WI  20364

                 Acquiring Institution        Closing Date       Updated Date
0                          Today's Bank  September 23, 2016  November 17, 2016
1                           United Bank     August 19, 2016  November 17, 2016
2  First-Citizens Bank & Trust Company         May 6, 2016  September 6, 2016
3            The Bank of Fayette County      April 29, 2016  September 6, 2016
4  First-Citizens Bank & Trust Company      March 11, 2016      June 16, 2016
```

[ ]:
```
close_timestamps = pd.to_datetime(failures["Closing Date"])
close_timestamps.dt.year.value_counts()
```

[ ]:
```
Closing Date
2010    157
2009    140
2011     92
2012     51
2008     25
       ...
2004      4
2001      4
2007      3
2003      3
2000      2
Name: count, Length: 15, dtype: int64
```

Analisando XML

[ ]:
```
import urllib.request
from lxml import objectify

path = base_url + "datasets/mta_perf/Performance_MNR.xml"

with urllib.request.urlopen(path) as f:
    parsed = objectify.parse(f)

root = parsed.getroot()
```

[ ]:
```
data = []

skip_fields = ["PARENT_SEQ", "INDICATOR_SEQ",
               "DESIRED_CHANGE", "DECIMAL_PLACES"]

for elt in root.INDICATOR:
    el_data = {}
    for child in elt.getchildren():
        if child.tag in skip_fields:
            continue
        el_data[child.tag] = child.pyval
    data.append(el_data)
```

[ ]:
```
perf = pd.DataFrame(data)
perf.head()
```

[ ]:

	AGENCY_NAME	INDICATOR_NAME	DESCRIPTION	PERIOD_YEAR
0	Metro-North Railroad	On-Time Performance (West of Hudson)	Percent of commuter trains that arrive at thei…	2008
1	Metro-North Railroad	On-Time Performance (West of Hudson)	Percent of commuter trains that arrive at thei…	2008
2	Metro-North Railroad	On-Time Performance (West of Hudson)	Percent of commuter trains that arrive at thei…	2008
3	Metro-North Railroad	On-Time Performance (West of Hudson)	Percent of commuter trains that arrive at thei…	2008
4	Metro-North Railroad	On-Time Performance (West of Hudson)	Percent of commuter trains that arrive at thei…	2008

	PERIOD_MONTH	CATEGORY	FREQUENCY	INDICATOR_UNIT	YTD_TARGET
0	1	Service Indicators	M	%	95.0
1	2	Service Indicators	M	%	95.0
2	3	Service Indicators	M	%	95.0
3	4	Service Indicators	M	%	95.0
4	5	Service Indicators	M	%	95.0

```
  YTD_ACTUAL MONTHLY_TARGET MONTHLY_ACTUAL
0       96.9           95.0           96.9
1       96.0           95.0           95.0
2       96.3           95.0           96.9
3       96.8           95.0           98.3
4       96.6           95.0           95.8
```

[ ]:
```
perf2 = pd.read_xml(path)
perf2.head()
```

[ ]:
```
   INDICATOR_SEQ  PARENT_SEQ           AGENCY_NAME  \
0          28445         NaN  Metro-North Railroad
1          28445         NaN  Metro-North Railroad
2          28445         NaN  Metro-North Railroad
3          28445         NaN  Metro-North Railroad
4          28445         NaN  Metro-North Railroad

                         INDICATOR_NAME  \
0  On-Time Performance (West of Hudson)
1  On-Time Performance (West of Hudson)
2  On-Time Performance (West of Hudson)
3  On-Time Performance (West of Hudson)
4  On-Time Performance (West of Hudson)

                                        DESCRIPTION  PERIOD_YEAR  \
0  Percent of commuter trains that arrive at thei…         2008
1  Percent of commuter trains that arrive at thei…         2008
2  Percent of commuter trains that arrive at thei…         2008
3  Percent of commuter trains that arrive at thei…         2008
4  Percent of commuter trains that arrive at thei…         2008

   PERIOD_MONTH            CATEGORY FREQUENCY DESIRED_CHANGE INDICATOR_UNIT  \
0             1  Service Indicators         M              U              %
1             2  Service Indicators         M              U              %
2             3  Service Indicators         M              U              %
3             4  Service Indicators         M              U              %
4             5  Service Indicators         M              U              %

   DECIMAL_PLACES YTD_TARGET YTD_ACTUAL MONTHLY_TARGET MONTHLY_ACTUAL
0               1      95.00      96.90          95.00          96.90
1               1      95.00      96.00          95.00          95.00
2               1      95.00      96.30          95.00          96.90
3               1      95.00      96.80          95.00          98.30
4               1      95.00      96.60          95.00          95.80
```

Formato de Dados Binários

[ ]:
```
frame = pd.read_csv(base_url + "examples/ex1.csv")
frame
```

[ ]:
```
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

[ ]:
```
frame.to_pickle("examples/frame_pickle")
pd.read_pickle("examples/frame_pickle")
```

[ ]:
```
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

Lendo arquivos do formato Excel

[ ]:
```
xlsx = pd.ExcelFile(base_url + "examples/ex1.xlsx")
xlsx.sheet_names
```

[ ]:
```
['Sheet1']
```

[ ]:
```
xlsx.parse(sheet_name="Sheet1")
```

[ ]:
```
   Unnamed: 0  a   b   c   d message
0           0  1   2   3   4   hello
1           1  5   6   7   8   world
2           2  9  10  11  12     foo
```

[ ]:
```
xlsx.parse(sheet_name="Sheet1", index_col=0)
```

[ ]:
```
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

[ ]:
```
frame = pd.read_excel(base_url + "examples/ex1.xlsx", sheet_name="Sheet1")
frame
```

[ ]:
```
   Unnamed: 0  a   b   c   d message
0           0  1   2   3   4   hello
1           1  5   6   7   8   world
2           2  9  10  11  12     foo
```

[ ]:
```
writer = pd.ExcelWriter("examples/ex2.xlsx")
frame.to_excel(writer, "Sheet1")
writer.close()
```

```
/tmp/ipykernel_24963/4135277578.py:2: FutureWarning: Starting with pandas
version 3.0 all arguments of to_excel except for the argument 'excel_writer'
will be keyword-only.
  frame.to_excel(writer, "Sheet1")
```

[ ]:
```
frame.to_excel("examples/ex2.xlsx")
frame
```

[ ]:
```
   Unnamed: 0  a   b   c   d message
0           0  1   2   3   4   hello
1           1  5   6   7   8   world
2           2  9  10  11  12     foo
```

Usando formato HDF5

[ ]:
```
import numpy as np
frame = pd.DataFrame({"a": np.random.standard_normal(100)})
store = pd.HDFStore("examples/mydata.h5")
store["obj1"] = frame
store["obj1_col"] = frame["a"]
store
```

[ ]:
```
<class 'pandas.io.pytables.HDFStore'>
File path: examples/mydata.h5
```

[ ]:
```
store["obj1"]
```

[ ]:
```
           a
0   1.498700
1   0.397916
2   0.055675
3  -0.426357
4  -0.169815
..       …
95  0.650061
96 -0.096914
97 -1.235860
98 -0.867650
99 -0.498083

[100 rows x 1 columns]
```

[ ]:
```
store.put("obj2", frame, format="table")
store.select("obj2", where=["index >= 10 and index <= 15"])
```

[ ]:
```
           a
10  0.404044
11  0.276968
```

```
12 -3.175140
13  0.229887
14  0.843530
15  0.101231
```

[ ]:
```
store.close()
```

[ ]:
```
frame.to_hdf("examples/mydata.h5", "obj3", format="table")
```

```
/tmp/ipykernel_24963/1151319590.py:1: FutureWarning: Starting with pandas
version 3.0 all arguments of to_hdf except for the argument 'path_or_buf' will
be keyword-only.
  frame.to_hdf("examples/mydata.h5", "obj3", format="table")
```

[ ]:
```
pd.read_hdf("examples/mydata.h5", "obj3", where=["index < 5"])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_24963/3028457229.py in <module>
----> 1 pd.read_hdf("examples/mydata.h5", "obj4", where=["index < 5"])

~/.local/lib/python3.10/site-packages/pandas/io/pytables.py in␣
 ↪read_hdf(path_or_buf, key, mode, errors, where, start, stop, columns,␣
 ↪iterator, chunksize, **kwargs)
    424             raise FileNotFoundError(f"File {path_or_buf} does not exist")
    425
--> 426         store = HDFStore(path_or_buf, mode=mode, errors=errors, **kwargs)
    427         # can't auto open/close if we are using an iterator
    428         # so delegate to the iterator

~/.local/lib/python3.10/site-packages/pandas/io/pytables.py in __init__(self,␣
 ↪path, mode, complevel, complib, fletcher32, **kwargs)
    583         self._fletcher32 = fletcher32
    584         self._filters = None
--> 585         self.open(mode=mode, **kwargs)
    586
    587     def __fspath__(self) -> str:

~/.local/lib/python3.10/site-packages/pandas/io/pytables.py in open(self, mode,␣
 ↪**kwargs)
    743             raise ValueError(msg)
    744
--> 745         self._handle = tables.open_file(self._path, self._mode, **kwargs)
    746
    747     def close(self) -> None:

/usr/local/lib/python3.10/dist-packages/tables/file.py in open_file(filename,␣
 ↪mode, title, root_uep, filters, **kwargs)
```

```
    276             # 'r' is incompatible with everything except 'r' itself
    277             if mode == 'r' and omode != 'r':
--> 278                 raise ValueError(
    279                     "The file '%s' is already opened, but "
    280                     "not in read-only mode (as requested)." % filename)

ValueError: The file 'examples/mydata.h5' is already opened, but not in␣
 ↪read-only mode (as requested).
```

O erro acima acredito que seja causado por versão de bibliotecas ou de gerenciamento no manejo de arquivos em meu computador (estou utilizando Linux).

Interagindo com APIs Web

[ ]:
```
import requests

url = "https://api.github.com/repos/pandas-dev/pandas/issues"
resp = requests.get(url)
resp.raise_for_status()
resp
```

[ ]:
```
<Response [200]>
```

[ ]:
```
data = resp.json()
data[0]["title"]
```

[ ]:
```
'DOC: fixing RT03 erros for Index: duplicated and nunique'
```

[ ]:
```
issues = pd.DataFrame(data, columns=["number", "title", "labels", "state"])
issues
```

[ ]:
```
    number                                              title  \
0    58432  DOC: fixing RT03 erros for Index: duplicated a…
1    58431       DOC: Fix DataFrame.reorder_levels SA01 error
2    58430      DOC: fixing SA01 error for Index: T and empty
3    58429  DOC: Fix RT03 errors for DataFrame.infer_objec…
4    58428  BUG: Adding two series with pd.Series.add can …
..     …                                                 …
25   58395  BUG: Losing information when handling of `None…
26   58394        ENH: Add support for numpy 2's string dtype
27   58392         BUG: identity checking NA in map incorrect
28   58391  PERF: df.unstack() is 500 times slower since p…
29   58388  DOC: pandas.Grouper should not accept args and…

                                               labels state
0                                                  []  open
1                                                  []  open
```

```
2                                                    []  open
3                                                    []  open
4   [{'id': 76811, 'node_id': 'MDU6TGFiZWw3NjgxMQ=…  open
..                                                  …   …
25  [{'id': 76811, 'node_id': 'MDU6TGFiZWw3NjgxMQ=…  open
26  [{'id': 76812, 'node_id': 'MDU6TGFiZWw3NjgxMg=…  open
27                                                   []  open
28  [{'id': 8935311, 'node_id': 'MDU6TGFiZWw4OTM1M…  open
29  [{'id': 134699, 'node_id': 'MDU6TGFiZWwxMzQ2OT…  open

[30 rows x 4 columns]
```

Interagindo com Banco de Dados

[ ]:
```
import sqlite3

query = """
CREATE TABLE test (
    a VARCHAR(20),
    b VARCHAR(20),
    c REAL,
    d INTEGER
);"""

con = sqlite3.connect("mydata.sqlite")

con.execute(query)
```

[ ]:
```
<sqlite3.Cursor at 0x7ff8d0aa3340>
```

[ ]:
```
con.commit()
```

[ ]:
```
data = [
    ("Atlanta", "Georgia", 1.25, 6),
    ("Tallahassee", "Florida", 2.6, 3),
    ("Sacramento", "California", 1.7, 5)
]

stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"
con.executemany(stmt, data)
```

[ ]:
```
<sqlite3.Cursor at 0x7ff8d09d7cc0>
```

[ ]:
```
con.commit()
```

[ ]:
```
cursor = con.execute("SELECT * FROM test")
rows = cursor.fetchall()
```

```
rows
```

```
[ ]: [('Atlanta', 'Georgia', 1.25, 6),
      ('Tallahassee', 'Florida', 2.6, 3),
      ('Sacramento', 'California', 1.7, 5)]
```

```
[ ]: cursor.description
```

```
[ ]: (('a', None, None, None, None, None, None),
      ('b', None, None, None, None, None, None),
      ('c', None, None, None, None, None, None),
      ('d', None, None, None, None, None, None))
```

```
[ ]: pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
```

```
[ ]:              a           b     c  d
     0      Atlanta     Georgia  1.25  6
     1  Tallahassee     Florida  2.60  3
     2   Sacramento  California  1.70  5
```

```
[ ]: import sqlalchemy as sqla

     db = sqla.create_engine("sqlite:///mydata.sqlite", echo=False)
     conn = db.raw_connection()
     pd.read_sql("SELECT * FROM test", conn)
```

```
/tmp/ipykernel_24963/1309403950.py:5: UserWarning: pandas only supports
SQLAlchemy connectable (engine/connection) or database string URI or sqlite3
DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using
SQLAlchemy.
  pd.read_sql("SELECT * FROM test", conn)
```

```
[ ]:              a           b     c  d
     0      Atlanta     Georgia  1.25  6
     1  Tallahassee     Florida  2.60  3
     2   Sacramento  California  1.70  5
```