T4_Processamento_dados_pandas

May 19, 2021

Trabalho #4 - Processamento de dados com Pandas

Nesse trabalho você vai realizar o processamento de dados de pacientes com suspeita de doença cardiovascular e treinar uma RNA para identificar se o paciente tem ou não doença cardiovascular.

O conjunto de dados utilizado nesse trabalho consiste de uma modificação do conjunto de dados original "Cardiovascular Disease Dataset", disponível no Kaggle e que pode ser obtido no seguinte link: https://www.kaggle.com/sulianova/cardiovascular-disease-dataset

Nome: Bruno Rodrigues Silva

1 Descrição dos dados

Nesse conjunto de dados existem 70.000 linhas. Cada linha descreve um paciente e cada coluna descreve uma característica.

O objetivo desse problema é determinar se um paciente tem ou não doença cardiovascular dadas as informações fornecidas pelos paciente e pelos resultados de seus exames.

As características presentes em cada uma das colunas de dados são as seguintes:

1. Idade ("age") em dias
2. Sexo ("gender"): 1 - masculino, 2 - feminino
3. Altura ("height") em cm
4. Peso ("weigth") em kg
5. Pressão systolica ("ap_hi") em mmHg
6. Pressão diastolica ("ap_lo") em mmHg
7. Colesterol ("cholesterol"): normal, high, very_high
8. Gicose ("glucose"): normal, high, very_high
9. Fumante ("smoking"): 0 - não fumante, 1 - fumante
10. Bebe álcool: 0 - não bebe, 1 - bebe
11. Atividade física ("activity"): 0 - não realiza, 1 - realiza
12. Presença de doença cardiovascular: 0 - não, 1 - sim

Ressalta-se que esse conjunto de dados é balanceado.

2 Objetivo do trabalho

O objetivo desse trabalho é processar os dados usando o Pandas e treinar uma RNA para identificar se os pacientes tem ou não doença cardiovascular.

Entrega do notebook:

- O notebook com a sua solução deve ser entregue no fomato pdf
- Os resultados de todas as etapas devem ser apresentados
- As células que devem ser feitas nesse trabalho estão indicados por: "Para você fazer"
- Algumas células de programação estão incluídas para facilitar a verificação do seus cálculos. Esses resutados permitem você saber se realizou a tarefa corretamente ou não, cuidado para não apagá-las (ou salve o enunciado original em outro notebook).

3 Carregar os dados

O conjunto de dados está no arquivo `heart_disease.csv`. Como esse conjunto de dados está em um arquivo tipo CSV use o Pandas para importá-lo e carregar os dados em um dataframe.

O arquivo já possui uma coluna de índice dos pacientes (`id`) , assim, use essa coluna como índice ao importar o arquivo.

[1]:
```
import pandas as pd
data = pd.read_csv("heart_disease.csv", index_col='id')
data
```

[1]:

id	age	gender	height	weight	...	smoke	alco	active	cardio
0	18393	2	168	62.0	...	0	0	1	0
1	20228	1	156	85.0	...	0	0	1	1
2	18857	1	165	64.0	...	0	0	0	1
3	17623	2	169	82.0	...	0	0	1	1
4	17474	1	156	56.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
99993	19240	2	168	76.0	...	1	0	1	0
99995	22601	1	158	126.0	...	0	0	1	1
99996	19066	2	183	105.0	...	0	1	0	1
99998	22431	1	163	72.0	...	0	0	0	1
99999	20540	1	170	72.0	...	0	0	1	0

[70000 rows x 12 columns]

Saída esperada:

id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	18393	2	168	62.0	110	80	normal	normal	0	0	1	0
1	20228	1	156	85.0	140	90	very_high	normal	0	0	1	1
2	18857	1	165	64.0	130	70	very_high	normal	0	0	0	1
3	17623	2	169	82.0	150	100	normal	normal	0	0	1	1
4	17474	1	156	56.0	100	60	normal	normal	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...

70000 rows × 12 columns

4 Analisar os dados

A análise para verificar se os dados são coerentes deve ser realizada para cada característica, ou seja, para cada coluna, tendo em mente valores máximos e mínimos possíveis para cada característica.

Para esse conjunto de dados você deve realizar as seguintes verificações:

- verificar tipo de dados de cada coluna e calcula estaísticas básicas de cada coluna
- Verificar se existem valores não existentes;
- Verificar se as pressões altas ("ap_hi") são maiores do que as pressões baixas ("ap_lo") para todos os pacientes;
- Verificar se a pressão alta ("ap_hi") é menor do que 300 mmHg, pois não existe pressão mais alta do que isso;
- Verificar se a pressão baixa ("ap_lo") é maior do que 30 mmHg, pois não existe pressão mais baixa do que isso;
- Verificar peso máximo e mínimo. O peso dos pacientes deve estar entre 30 kg e 450 kg.
- Verificar altura mínima e máxima. A altura de um adulto não deve ultrapassar 250 cm e não deve ser menor do que 70 cm.

Caso encontre pacientes com dados fora de valores aceitáveis, as linhas correspondentes a esses pacientes devem ser retiradas.

Outras verificações podem e devem ser realizadas e fica a seu critério pensar no que se pode fazer para aprimorar os dados. Observe que dados incoerentes afetam o desempenho da RNA.

Abaixo segue um exemplo de código para verificar se a pressão alta ("ap_hi") é maior do que 300 mmHg e remover os pacientes com esse problema:

```
index = data.index   # separa coluna de índices dos pacientes
condition = data["ap_hi"]>300  # verifica se ap_hi é maior do que 300
indph = index[condition]  # identifica pacientes com pressão maior do que 300
print(data.loc[indph, "ap_hi"]) # apresenta pacientes com dados incoerentes
data = data.drop(indph) # remove linhas correspondentes aos pacientes com dados incoerentes
data # apresenta novo DataFrame
```

4.1 Verificar tipos de dados de cada coluna

Use os métodos `info()` e `describe()` para fazer essa verificação usando o DataFrame completo. Os resultados desses comando fornece muita informação para processamento dos dados.

[2]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 70000 entries, 0 to 99999
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          70000 non-null  int64
```

```
 1   gender       70000 non-null  int64
 2   height       70000 non-null  int64
 3   weight       70000 non-null  float64
 4   ap_hi        70000 non-null  int64
 5   ap_lo        70000 non-null  int64
 6   cholesterol  70000 non-null  object
 7   gluc         70000 non-null  object
 8   smoke        70000 non-null  int64
 9   alco         70000 non-null  int64
 10  active       70000 non-null  int64
 11  cardio       70000 non-null  int64
dtypes: float64(1), int64(9), object(2)
memory usage: 6.9+ MB
```

[3]:
```
data.describe().T
```

[3]:

	count	mean	std	...	50%	75%	max
age	70000.0	19468.865814	2467.251667	...	19703.0	21327.0	23713.0
gender	70000.0	1.349571	0.476838	...	1.0	2.0	2.0
height	70000.0	164.359229	8.210126	...	165.0	170.0	250.0
weight	70000.0	74.205690	14.395757	...	72.0	82.0	200.0
ap_hi	70000.0	128.817286	154.011419	...	120.0	140.0	16020.0
ap_lo	70000.0	96.630414	188.472530	...	80.0	90.0	11000.0
smoke	70000.0	0.088129	0.283484	...	0.0	0.0	1.0
alco	70000.0	0.053771	0.225568	...	0.0	0.0	1.0
active	70000.0	0.803729	0.397179	...	1.0	1.0	1.0
cardio	70000.0	0.499700	0.500003	...	0.0	1.0	1.0

[10 rows x 8 columns]

4.2 Exercício #1: Verificar se as pressões altas ("ap_hi") são maiores do que as pressões baixas ("ap_lo") e retirar linhas se necessário

Calcule a diferença entre a pressão alta e pressão baixa.

[4]:
```
# Para você fazer: calcular diferença pressão alta menos pressão baixa
dp = data["ap_hi"]-data["ap_lo"]
dp
```

[4]:
```
id
0        30
1        50
2        60
3        50
4        40
         ..
99993    40
```

```
99995    50
99996    90
99998    55
99999    40
Length: 70000, dtype: int64
```

Remoção das linhas com diferença de pressão negativa

[5]:
```
# Para você fazer: remoção das linhas com diferença de pressão negativa
index = data.index   # separa coluna de índices dos pacientes
indpn = index[dp<0]  # identifica pacientes com pressão maior do que 300

# Insire seu código aqui
#
data = data.drop(indpn) # remove linhas correspondentes aos pacientes com dados
 ↪incoerentes

print('Número de linhas com diferença de pressão negativa:', len(indpn))
data # apresenta novo DataFrame
```

```
Número de linhas com diferença de pressão negativa: 1234
```

[5]:

id	age	gender	height	weight	...	smoke	alco	active	cardio
0	18393	2	168	62.0	...	0	0	1	0
1	20228	1	156	85.0	...	0	0	1	1
2	18857	1	165	64.0	...	0	0	0	1
3	17623	2	169	82.0	...	0	0	1	1
4	17474	1	156	56.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
99993	19240	2	168	76.0	...	1	0	1	0
99995	22601	1	158	126.0	...	0	0	1	1
99996	19066	2	183	105.0	...	0	1	0	1
99998	22431	1	163	72.0	...	0	0	0	1
99999	20540	1	170	72.0	...	0	0	1	0

```
[68766 rows x 12 columns]
```

Saída esperada:

```
Número de linhas com diferença de pressão negativa 123
```

4.3 Exercíco #2: Verificar se a pressão alta ("ap_hi") é maior do que 300 mmHg e retirar linhas se necessário

Observe que não existe pressão mais alta do que isso.

[6]:
```
# Para você fazer:

# Verificar se existem pressões maiores do que 300 mmHg
# Insire seu código aqui
#
index = data.index
condition = data["ap_hi"]>300
indph = index[condition]
# Retirar linhas com pressões maiores do que 300 mmHg
# Insire seu código aqui
#
data = data.drop(indph)

print('Número de linhas com pressão maior do que 300:', len(indph))
data
```

```
Número de linhas com pressão maior do que 300: 40
```

[6]:

id	age	gender	height	weight	...	smoke	alco	active	cardio
0	18393	2	168	62.0	...	0	0	1	0
1	20228	1	156	85.0	...	0	0	1	1
2	18857	1	165	64.0	...	0	0	0	1
3	17623	2	169	82.0	...	0	0	1	1
4	17474	1	156	56.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
99993	19240	2	168	76.0	...	1	0	1	0
99995	22601	1	158	126.0	...	0	0	1	1
99996	19066	2	183	105.0	...	0	1	0	1
99998	22431	1	163	72.0	...	0	0	0	1
99999	20540	1	170	72.0	...	0	0	1	0

[68726 rows x 12 columns]

Saída esperada:

```
Número de linhas com pressão maior do que 300: 40
```

4.4 Exercício #3: Verificar se a pressão baixa ("ap_lo") é menor do que 30 mmHg e retirar linhas se necessário

Observe que não existe pressão mais baixa do que esse valor.

[7]:
```
# Para você fazer:

# Verificar se existem pressões menores do que 30 mmHg
# Insire seu código aqui
#
```

```
index = data.index
condition = data["ap_lo"]<30
indpl = index[condition]
# Retirar linhas com pressões menores do que 30 mmHg
# Insire seu código aqui
#
data = data.drop(indpl)

print('Número de linhas com pressão menor do que 30:', len(indpl))
data
```

```
Número de linhas com pressão menor do que 30: 46
```

[7]:

id	age	gender	height	weight	...	smoke	alco	active	cardio
					...				
0	18393	2	168	62.0	...	0	0	1	0
1	20228	1	156	85.0	...	0	0	1	1
2	18857	1	165	64.0	...	0	0	0	1
3	17623	2	169	82.0	...	0	0	1	1
4	17474	1	156	56.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
99993	19240	2	168	76.0	...	1	0	1	0
99995	22601	1	158	126.0	...	0	0	1	1
99996	19066	2	183	105.0	...	0	1	0	1
99998	22431	1	163	72.0	...	0	0	0	1
99999	20540	1	170	72.0	...	0	0	1	0

[68680 rows x 12 columns]

Saída esperada:

```
Número de linhas com pressão menor do que 30: 40
```

4.5 Exercício #4: Verificar peso máximo e mínimo e remover linhas se necessário

O peso dos pacientes deve estar entre 20 kg e 450 kg.

[8]:
```
# Para você fazer:

# Verificar se existem peso menor do que 20 kg e maior do que 450 kg
# Insire seu código aqui
#
index = data.index
condition = (data["weight"]<20) | (data["weight"]>450)
indwl = index[condition]
# Insire seu código aqui
#
data = data.drop(indwl)
```

```
print('Número de linhas com peso menor do que 20:', len(indwl))
data
```

```
Número de linhas com peso menor do que 20: 1
```

[8]:

id	age	gender	height	weight	...	smoke	alco	active	cardio
0	18393	2	168	62.0	...	0	0	1	0
1	20228	1	156	85.0	...	0	0	1	1
2	18857	1	165	64.0	...	0	0	0	1
3	17623	2	169	82.0	...	0	0	1	1
4	17474	1	156	56.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
99993	19240	2	168	76.0	...	1	0	1	0
99995	22601	1	158	126.0	...	0	0	1	1
99996	19066	2	183	105.0	...	0	1	0	1
99998	22431	1	163	72.0	...	0	0	0	1
99999	20540	1	170	72.0	...	0	0	1	0

[68679 rows x 12 columns]

Saída esperada:

```
Número de linhas com peso menor do que 20: 1
```

4.6 Exercício #5: Verificar altura mínima e máxima e remover linhas se necessário

A altura de um adulto não deve ultrapassar 250 cm e não deve ser menor do que 70 cm.

[9]:
```
# Para você fazer:

# Verificar se existem altura menor do que 70 cm e maior do que 250 cm
# Insire seu código aqui
#

index = data.index
condition = (data["height"]<70) | (data["height"]>250)
indal = index[condition]
# Insire seu código aqui
#
data = data.drop(indal)

print('Número de linhas com altura menor do que 70:', len(indal))
data
```

```
Número de linhas com altura menor do que 70: 12
```

[9]:

id	age	gender	height	weight	...	smoke	alco	active	cardio
					...				
0	18393	2	168	62.0	...	0	0	1	0
1	20228	1	156	85.0	...	0	0	1	1
2	18857	1	165	64.0	...	0	0	0	1
3	17623	2	169	82.0	...	0	0	1	1
4	17474	1	156	56.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
99993	19240	2	168	76.0	...	1	0	1	0
99995	22601	1	158	126.0	...	0	0	1	1
99996	19066	2	183	105.0	...	0	1	0	1
99998	22431	1	163	72.0	...	0	0	0	1
99999	20540	1	170	72.0	...	0	0	1	0

[68667 rows x 12 columns]

Saída esperada:

```
Número de linhas com altura menor do que 70: 14
```

4.7 Visualização das características

Após remover dados inconsistentes, usando o método `describe()` deve-se calcular as estatísticas dos dados de todas as colunas e verifique se estão de acordo com o esperado.

[10]:
```
data.describe().T
```

[10]:

	count	mean	std	...	50%	75%	max
age	68667.0	19464.640162	2468.156460	...	19701.0	21324.0	23713.0
gender	68667.0	1.348625	0.476538	...	1.0	2.0	2.0
height	68667.0	164.378726	8.074608	...	165.0	170.0	250.0
weight	68667.0	74.122328	14.331349	...	72.0	82.0	200.0
ap_hi	68667.0	126.674837	16.696231	...	120.0	140.0	240.0
ap_lo	68667.0	81.303872	9.466903	...	80.0	90.0	182.0
smoke	68667.0	0.087961	0.283240	...	0.0	0.0	1.0
alco	68667.0	0.053344	0.224721	...	0.0	0.0	1.0
active	68667.0	0.803384	0.397442	...	1.0	1.0	1.0
cardio	68667.0	0.494735	0.499976	...	0.0	1.0	1.0

[10 rows x 8 columns]

5 Transformar os dados das colunas

Para transformar os dados deve-se fazer o seguinte:

1. “Age”. Para essa coluna você deve transformar os dias em anos, seguimentar em faixas de idades e depois codificar em vetores “one-hot”.
2. “Gender”. Alterar os valores para 0 e 1.

3. Colunas numéricas “height”, “weight”, “ap_hi”, “ap_lo” devem ser normalizadas para terem média 0 e desvio padrão igual a 1.
4. Colunas categóricas “cholesterol” e “gluc” devem ser codificadas para vetores “one-hot”.
5. Uma informação que pode ser interessante é incluir o índice de massa corporal dos pacientes como sendo uma nova caraterística. Esse índice é igual ao peso (kg) dividido pela altura (metros) ao quadrado.

5.1 Exercíco #6: Coluna “Age”

Para essa coluna você pode transformar os dias em anos, seguimentar em faixas de idades e depois codificar em vetores “one-hot”.

A segmentação das idades pode ser feita em 4 divisões da seguinte forma:

```
age < 40
40 <= age < 50
50 <= age < 60
age >= 60
```

[11]:
```
# Para você fazer: Transformar idades em anos
# Insire seu código aqui
data['age'] = data['age']/365
data['age'].describe()
```

[11]:
```
count    68667.000000
mean        53.327781
std          6.762072
min         29.583562
25%         48.376712
50%         53.975342
75%         58.421918
max         64.967123
Name: age, dtype: float64
```

[12]:
```
# Para você fazer: Criar coluna de idade segmentada
# Insire seu código aqui
#
def segmenta_idades(x):
    if x < 40:
        return 0
    elif x<50:
        return 1
    elif x<60:
        return 2
    else:
        return 3
data['age_seg'] = data['age'].apply(lambda x:segmenta_idades(x))
data
```

[12]:
```
             age  gender  height  weight  ...  alco  active cardio age_seg
id                                        ...
0      50.391781       2     168    62.0  ...     0       1      0       2
1      55.419178       1     156    85.0  ...     0       1      1       2
2      51.663014       1     165    64.0  ...     0       0      1       2
3      48.282192       2     169    82.0  ...     0       1      1       1
4      47.873973       1     156    56.0  ...     0       0      0       1
...          ...     ...     ...     ...  ...   ...     ...    ...     ...
99993  52.712329       2     168    76.0  ...     0       1      0       2
99995  61.920548       1     158   126.0  ...     0       1      1       3
99996  52.235616       2     183   105.0  ...     1       0      1       2
99998  61.454795       1     163    72.0  ...     0       0      1       3
99999  56.273973       1     170    72.0  ...     0       1      0       2

[68667 rows x 13 columns]
```

[13]:
```
data.age_seg.value_counts()
```

[13]:
```
2    34837
1    19290
3    12778
0     1762
Name: age_seg, dtype: int64
```

Saída esperada:

```
2    34839
1    19296
3    12752
0     1770
Name: age_seg, dtype: int6
```

[14]:
```
# Para você fazer:

#Codificação one-hot criando um novo dataframe
# Insire seu código aqui
#
data_age_seg = pd.get_dummies(data['age_seg'])
data = data.drop('age_seg',axis = 1)
data = data.join(data_age_seg)
# União do resultado da codificação one-hot com dataframe original
# Insire seu código aqui
#
data
```

[14]:
```
             age  gender  height  weight  ap_hi  ...  cardio  0  1  2  3
id                                               ...
0      50.391781       2     168    62.0    110  ...       0  0  0  1  0
```

```
1       55.419178       1     156    85.0    140  ...       1  0  0  1  0
2       51.663014       1     165    64.0    130  ...       1  0  0  1  0
3       48.282192       2     169    82.0    150  ...       1  0  1  0  0
4       47.873973       1     156    56.0    100  ...       0  0  1  0  0
...           ...     ...     ...     ...    ...  ...     ... .. .. .. ..
99993   52.712329       2     168    76.0    120  ...       0  0  0  1  0
99995   61.920548       1     158   126.0    140  ...       1  0  0  0  1
99996   52.235616       2     183   105.0    180  ...       1  0  0  1  0
99998   61.454795       1     163    72.0    135  ...       1  0  0  0  1
99999   56.273973       1     170    72.0    120  ...       0  0  0  1  0

[68667 rows x 16 columns]
```

5.2 Exercíco #7: Coluna "Gender"

Alterar os valores para 0 e 1.

[15]:
```
# Para você fazer:

# Transformação da coluna "Sex" em dados categóricos
# Insire seu código aqui
#
data['gender'] = data['gender'].apply(lambda x:0 if x==1 else 1).astype(int)
# Modificação das categorias de strings para números inteiros
# Insire seu código aqui
#
data.head()
```

[15]:
```
           age  gender  height  weight  ap_hi  ap_lo  ... active cardio  0  1  2
3
id                                                    ...
0   50.391781       1     168    62.0    110     80  ...      1      0  0  0  1
0
1   55.419178       0     156    85.0    140     90  ...      1      1  0  0  1
0
2   51.663014       0     165    64.0    130     70  ...      0      1  0  0  1
0
3   48.282192       1     169    82.0    150    100  ...      1      1  0  1  0
0
4   47.873973       0     156    56.0    100     60  ...      0      0  0  1  0
0

[5 rows x 16 columns]
```

5.3 Exercício #8: Colunas numéricas "height", "weight", "ap_hi", "ap_lo"

Essas colunas devem ser normalizadas para terem média 0 e desvio padrão igual a 1.

[16]:
```
# Para você fazer: Normalização das colunas height, weight, ap_hi e ap_lo
# Insire seu código aqui
#
### exercício 10: criação da coluna imc
data['imc'] = data['weight']/((data['height']/100)**2)


for c in ['height', 'weight', 'ap_hi', 'ap_lo', 'imc']:
    data[c] = (data[c]-data[c].mean())/data[c].std()



data
```

[16]:

id	age	gender	height	weight	ap_hi	...	0	1	2	3	imc
0	50.391781	1	0.448477	-0.845861	-0.998719	...	0	0	1	0	-0.963099
1	55.419178	0	-1.037664	0.759012	0.798094	...	0	0	1	0	1.293725
2	51.663014	0	0.076942	-0.706307	0.199157	...	0	0	1	0	-0.694819
3	48.282192	1	0.572322	0.549681	1.397032	...	0	1	0	0	0.211123
4	47.873973	0	-1.037664	-1.264524	-1.597656	...	0	1	0	0	-0.781297
...	...	...	...	...	...	...	..	..	..	..	...
99993	52.712329	1	0.448477	0.131019	-0.399781	...	0	0	1	0	-0.099358
99995	61.920548	0	-0.789973	3.619874	0.798094	...	0	0	0	1	4.000578
99996	52.235616	1	2.306152	2.154554	3.193844	...	0	0	1	0	0.671367
99998	61.454795	0	-0.170748	-0.148090	0.498625	...	0	0	0	1	-0.069440
99999	56.273973	0	0.696167	-0.148090	-0.399781	...	0	0	1	0	-0.450046

[68667 rows x 17 columns]

5.4 Exercício #9: Colunas categóricas "cholesterol" e "gluc"

Essa colunas devem ser codificadas para vetores "one-hot".

[17]:
```
data['gluc'].unique()
```

[17]:
```
array(['normal', 'high', 'very_high'], dtype=object)
```

[18]:
```
# Para você fazer: Coluna "cholesterol"

# Transformação da coluna "cholesterol" em dados categóricos
# Insire seu código aqui
#
def cat_to_int(x):
    if x == 'normal':
        return 0
    elif x=='high':
        return 1
    elif x=='very_high':
        return 2

data['cholesterol'] = data['cholesterol'].apply(lambda x : cat_to_int(x)).
 ↪astype(int)
# Modificação das categorias de strings para números inteiros
# Insire seu código aqui
#

# Codificação one-hot criando um novo dataframe
# Insire seu código aqui
#
data_one = pd.get_dummies(data['cholesterol'])
data = pd.concat([data, data_one], axis=1)
# União do resultado da codificação one-hot com dataframe original
# Insire seu código aqui
#

data
```

[18]:

	age	gender	height	weight	ap_hi	...	3	imc	0	1	2
id						...					
0	50.391781	1	0.448477	-0.845861	-0.998719	...	0	-0.963099	1	0	0
1	55.419178	0	-1.037664	0.759012	0.798094	...	0	1.293725	0	0	1
2	51.663014	0	0.076942	-0.706307	0.199157	...	0	-0.694819	0	0	1
3	48.282192	1	0.572322	0.549681	1.397032	...	0	0.211123	1	0	0
4	47.873973	0	-1.037664	-1.264524	-1.597656	...	0	-0.781297	1	0	0
...	...	...	...	...	...	...	...	...	..	..	..

```
99993  52.712329       1  0.448477  0.131019 -0.399781  ...   0 -0.099358  1  0
0
99995  61.920548       0 -0.789973  3.619874  0.798094  ...   1  4.000578  0  1
0
99996  52.235616       1  2.306152  2.154554  3.193844  ...   0  0.671367  0  0
1
99998  61.454795       0 -0.170748 -0.148090  0.498625  ...   1 -0.069440  1  0
0
99999  56.273973       0  0.696167 -0.148090 -0.399781  ...   0 -0.450046  0  1
0

[68667 rows x 20 columns]
```

[19]:
```
# Para você fazer: Coluna "gluc"

# Transformação da coluna "gluc" em dados categóricos
# Insire seu código aqui
#
# data['gluc'] = data['gluc'].apply(lambda x : cat_to_int(x)).astype(int)
# Modificação das categorias de strings para números inteiros
# Insire seu código aqui
#

# Codificação one-hot criando um novo dataframe
# Insire seu código aqui
#
data_one = pd.get_dummies(data['gluc'])
data = pd.concat([data, data_one], axis=1)

# União do resultado da codificação one-hot com dataframe original
# Insire seu código aqui
#

data
```

[19]:

id	age	gender	height	weight	...	2	high	normal	very_high
0	50.391781	1	0.448477	-0.845861	...	0	0	1	0
1	55.419178	0	-1.037664	0.759012	...	1	0	1	0
2	51.663014	0	0.076942	-0.706307	...	1	0	1	0
3	48.282192	1	0.572322	0.549681	...	0	0	1	0
4	47.873973	0	-1.037664	-1.264524	...	0	0	1	0
...	...	...	...	...	...	...	...	...	...
99993	52.712329	1	0.448477	0.131019	...	0	0	1	0
99995	61.920548	0	-0.789973	3.619874	...	0	1	0	0
99996	52.235616	1	2.306152	2.154554	...	1	0	1	0
99998	61.454795	0	-0.170748	-0.148090	...	0	1	0	0

```
99999  56.273973       0  0.696167 -0.148090  ...  0     0       1          0

[68667 rows x 23 columns]
```

5.5 Exercício #10: Índice de massa corporal (IMC)

OBS: Esse exercício foi feito antes da normalização, pois é mais simples. O índice de massa coprora é igual ao peso (kg) dividido pela altura (metros) ao quadrado.

Essa nova coluna deve ser incorporada aos dados.

[20]:
```
# Para você fazer: Incluir coluna de IMC
# Insire seu código aqui
#

data
```

[20]:
```
             age  gender    height    weight  ...  2  high  normal very_high
id                                            ...
0      50.391781       1  0.448477 -0.845861  ...  0     0       1          0
1      55.419178       0 -1.037664  0.759012  ...  1     0       1          0
2      51.663014       0  0.076942 -0.706307  ...  1     0       1          0
3      48.282192       1  0.572322  0.549681  ...  0     0       1          0
4      47.873973       0 -1.037664 -1.264524  ...  0     0       1          0
...          ...     ...       ...       ...  ... ..   ...     ...        ...
99993  52.712329       1  0.448477  0.131019  ...  0     0       1          0
99995  61.920548       0 -0.789973  3.619874  ...  0     1       0          0
99996  52.235616       1  2.306152  2.154554  ...  1     0       1          0
99998  61.454795       0 -0.170748 -0.148090  ...  0     1       0          0
99999  56.273973       0  0.696167 -0.148090  ...  0     0       1          0

[68667 rows x 23 columns]
```

5.6 Exercício #11: Remover colunas originais que possam permanecer após a etapa de transformação

As colunas originais “age”, " cholesterol" e " gluc" devem ser removidas após serem transfromadas.

[21]:
```
# Para você fazer: Remoção das colunas
# Insire seu código aqui
#
data = data.drop(['age', 'cholesterol', 'gluc'], axis=1)
```

5.7 Visualização dos resultados

[22]:
```
# Visualização das estatísticas usando decribe()
data.describe().T
```

[22]:

	count	mean	std	...	50%	75%	max
gender	68667.0	3.486245e-01	0.476538	...	0.000000	1.000000	1.000000
height	68667.0	1.570371e-15	1.000000	...	0.076942	0.696167	10.603768
weight	68667.0	-9.057048e-16	1.000000	...	-0.148090	0.549681	8.783379
ap_hi	68667.0	-3.821698e-16	1.000000	...	-0.399781	0.798094	6.787470
ap_lo	68667.0	-2.265070e-16	1.000000	...	-0.137730	0.918582	10.636650
smoke	68667.0	8.796074e-02	0.283240	...	0.000000	0.000000	1.000000
alco	68667.0	5.334440e-02	0.224721	...	0.000000	0.000000	1.000000
active	68667.0	8.033844e-01	0.397442	...	1.000000	1.000000	1.000000
cardio	68667.0	4.947355e-01	0.499976	...	0.000000	1.000000	1.000000
0	68667.0	2.566007e-02	0.158120	...	0.000000	0.000000	1.000000
1	68667.0	2.809210e-01	0.449452	...	0.000000	1.000000	1.000000
2	68667.0	5.073325e-01	0.499950	...	1.000000	1.000000	1.000000
3	68667.0	1.860865e-01	0.389179	...	0.000000	0.000000	1.000000
imc	68667.0	-3.893871e-15	1.000000	...	-0.200518	0.456455	47.218628
0	68667.0	7.499090e-01	0.433068	...	1.000000	1.000000	1.000000
1	68667.0	1.354799e-01	0.342238	...	0.000000	0.000000	1.000000
2	68667.0	1.146111e-01	0.318554	...	0.000000	0.000000	1.000000
high	68667.0	7.379090e-02	0.261432	...	0.000000	0.000000	1.000000
normal	68667.0	8.502337e-01	0.356845	...	1.000000	1.000000	1.000000
very_high	68667.0	7.597536e-02	0.264961	...	0.000000	0.000000	1.000000

[20 rows x 8 columns]

[23]:
```
# Visualização dos dados transformados
data
```

[23]:

id	gender	height	weight	ap_hi	...	2	high	normal	very_high
0	1	0.448477	-0.845861	-0.998719	...	0	0	1	0
1	0	-1.037664	0.759012	0.798094	...	1	0	1	0
2	0	0.076942	-0.706307	0.199157	...	1	0	1	0
3	1	0.572322	0.549681	1.397032	...	0	0	1	0
4	0	-1.037664	-1.264524	-1.597656	...	0	0	1	0
...	...	...	...	...	...	..	...	...	...
99993	1	0.448477	0.131019	-0.399781	...	0	0	1	0
99995	0	-0.789973	3.619874	0.798094	...	0	1	0	0
99996	1	2.306152	2.154554	3.193844	...	1	0	1	0
99998	0	-0.170748	-0.148090	0.498625	...	0	1	0	0
99999	0	0.696167	-0.148090	-0.399781	...	0	0	1	0

[68667 rows x 20 columns]

6 Preparação dos conjuntos de dados

6.1 Exercício #12: Divisão dos dados

Os dados devem ser divididos em conjuntos de treinamento, validação e teste. Sugere-se usar a seguinte divisão:

- Dados de treinamento: 70%
- Dados de validação: 15%
- Dados de teste: 15%

Ao separar os dados deve-se escolher as linhas aleatóriamente. Use a função `train_test_split()` da biblioteca SciKitLearn para fazer a divisão.

[25]:
```
# Para você fazer: Divisão dos dados
from sklearn.model_selection import train_test_split

# Insire seu código aqui
#
data_train, data_val = train_test_split(data, train_size=0.7)
data_val, data_test = train_test_split(data_val, train_size=0.5)
print("Numero de exemplos de treinamemto =", len(data_train))
print("Numero de exemplos de validação =", len(data_val))
print("Numero de exemplos de teste =", len(data_test))
```

```
Numero de exemplos de treinamemto = 48066
Numero de exemplos de validação = 10300
Numero de exemplos de teste = 10301
```

Saída esperada:

```
Numero de exemplos de treinamemto = 46657
Numero de exemplos de validação = 11000
Numero de exemplos de teste = 11000
```

6.2 Exercício #13: Separação das saídas

As saídas devem ser separadas das entradas e devem ser criados dados de entrada e de saída para os conjuntos de treinamento, validação e teste.

[26]:
```
# Para você fazer: Separação das saídas e criação dos dados de entrada e de␣
 ↪saída de treinamento, validação e teste

# Dados de treinamento
# Insire seu código aqui
#
y_train = data_train.pop('cardio')
x_train = data_train.copy()
# Dados de validação
# Insire seu código aqui
```

```
#
y_val = data_val.pop('cardio')
x_val = data_val.copy()

# Dados de teste
# Insire seu código aqui
#
y_test = data_test.pop('cardio')
x_test = data_test.copy()
```

DataFrame de dados de treinamento

[27]:
```
x_train
```

[27]:
```
        gender    height    weight     ap_hi  ...  2  high  normal  very_high
id                                          ...
34032        1 -0.170748 -0.078313 -0.399781  ...  0     0       1          0
20541        0 -1.285354  0.061241  0.798094  ...  0     0       1          0
81004        0 -1.161509  1.666115 -0.399781  ...  0     0       1          0
95691        1 -0.418438 -0.217867 -0.399781  ...  0     0       1          0
14347        0 -1.161509 -0.357421 -0.998719  ...  0     0       1          0
...        ...       ...       ...       ...  ... ..   ...     ...        ...
99686        1  0.696167  0.200796  1.397032  ...  0     0       1          0
93812        0 -0.789973 -1.194746 -1.597656  ...  0     0       1          0
46478        0 -0.542283 -0.148090 -0.399781  ...  0     0       1          0
35971        1  2.182307  1.107898  0.199157  ...  1     0       1          0
49232        1  1.934617  0.131019 -0.399781  ...  0     1       0          0

[48066 rows x 19 columns]
```

[28]:
```
y_train
```

[28]:
```
id
34032    1
20541    1
81004    0
95691    0
14347    0
        ..
99686    1
93812    0
46478    0
35971    1
49232    1
Name: cardio, Length: 48066, dtype: int64
```

Saída eseprada:

```
id
76349    0
64327    0
26232    0
80070    0
83461    0
        ..
42747    1
91562    1
36913    1
31436    0
2054     1
Name: cardio, Length: 46657, dtype: int6
```

6.3 Exercício #14: Transformar os DataFrames em tensores Numpy

Finalmente deve-se transformar os DataFrames em tensores Numpy para poderem ser usados por uma RNA.

[29]:
```
# Para você fazer:

# Transforma dados de entrada em tensores Numpy
# Insire seu código aqui
#
x_train = x_train.values
x_val = x_val.values
x_test = x_test.values
# Transforma dados de saída em tensores Numpy
# Insire seu código aqui
#
y_train = y_train.ravel()
y_val = y_val.ravel()
y_test = y_test.ravel()

# Apresenta dimensões dos dados de entrada
print("Dimensões dos dados de entrada")
print("Dados de treinamemto =", x_train.shape)
print("Dados de validação =", x_val.shape)
print("Dados de teste =", x_test.shape)

# Apresenta dimensões dos dados de saída
print(" ")
print("Dimensão dos dados e saída")
print("Dados de treinamemto =", y_train.shape)
print("Dados de validação =", y_val.shape)
print("Dados de teste =", y_test.shape)
```

```
Dimensões dos dados de entrada
Dados de treinamemto = (48066, 19)
Dados de validação = (10300, 19)
Dados de teste = (10301, 19)

Dimensão dos dados e saída
Dados de treinamemto = (48066,)
Dados de validação = (10300,)
Dados de teste = (10301,)
```

Saída esperada:

```
Dimensões dos dados de entrada
Dados de treinamemto = (46657, 19)
Dados de validação = (11000, 19)
Dados de teste = (11000, 19)

Dimensão dos dados e saída
Dados de treinamemto = (46657,)
Dados de validação = (11000,)
Dados de teste = (11000,)
```

6.4 Exercício #15: Muitos dos dados de entrada são números inteiros que precisam ser transformados em números reais.

[33]:
```
# Para você fazer: Tranformação dos dados de entrada em números reais
x_train = x_train.astype(float)
x_val = x_val.astype(float)
x_test = x_test.astype(float)
# Insire seu código aqui
#
```

7 Configuração, compilação e treinamento da RNA

7.1 Exercício #16: Configuração da RNA

Para resolver esse problema de calcular a probabilidade de um passageiro do navio Titanic sobreviver ao naufrágio, vamos utilizar uma RNA simples com três camadas densas.

Com certeza você vai ter problema de "overfitting" dos dados. Assim, para tentar minimizar esse problema inclua na sua RNA alguma forma de regularização, tal como, regularização L2 ou "dropout".

[57]:
```
# Para você fazer: Configuração da RNA
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential
# Cria RNA com 3 camadas densas
# Insire seu código aqui
```

```
#
rna = Sequential([
    Dense(64, activation='relu', input_shape=(19,)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(8, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid'),
])

# Apresenta sumário da RNA
rna.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 64)                1280
_________________________________________________________________
dropout_15 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_21 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_16 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_22 (Dense)             (None, 8)                 264
_________________________________________________________________
dropout_17 (Dropout)         (None, 8)                 0
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 9
=================================================================
Total params: 3,633
Trainable params: 3,633
Non-trainable params: 0
_________________________________________________________________
```

Saída esperada:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                1280
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
```

```
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 264
_________________________________________________________________
dropout_2 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 3,633
Trainable params: 3,633
Non-trainable params: 0
_________________________________________________________________
```

7.2 Exercício #16: Compilação da RNA

Vamos compilar essa RNA vamos com o método de otimização Adam.

Com é um problema de classificação binária, a função de custo mais indicada é a `binary_crossentropy` (função logística). Como métrica vamos utlizar a exatidão (`accuracy`).

[58]:
```
# Para você fazer: Definição do otimizador e compilação da RNA
from tensorflow.keras.optimizers import Adam
# Define método de otimização
# Insire seu código aqui
#
opt = Adam()
# Compila RNA
rna.compile(opt, loss='binary_crossentropy', metrics='accuracy')
# Insire seu código aqui
#
```

7.3 Exercício #17: Treinamento da RNA

Para treinar a sua RNA use 100 épocas e verbose=0.

Inclua os gráficos do valor da função de custo e da métrica em função das épocas de treinamento para os dados de treinamento e validação. Esses gráficos são importantes para verificar se estão ocorrendo problema de "overfitting" ou de "underfitting".

Casso esteja ocorrendo "underfitting", aumente o número de camadas da RNA e/ou o número de neurônios das camadas.

Caso esteja ocorrendo "overfitting", aumente o parâmetro de regularização L2 ou a taxa de "dropout", dependo do que você estiver utilizando para minimixar esse problema.

[59]:
```
# Para você fazer:

# Realiza o treinamento usando os dados de treinamento e validação
# Insire seu código aqui
```

```
#
history = rna.fit(x_train, y_train, batch_size=1,verbose=0, epochs=100,␣
 ↪validation_data=(x_val, y_val))
```

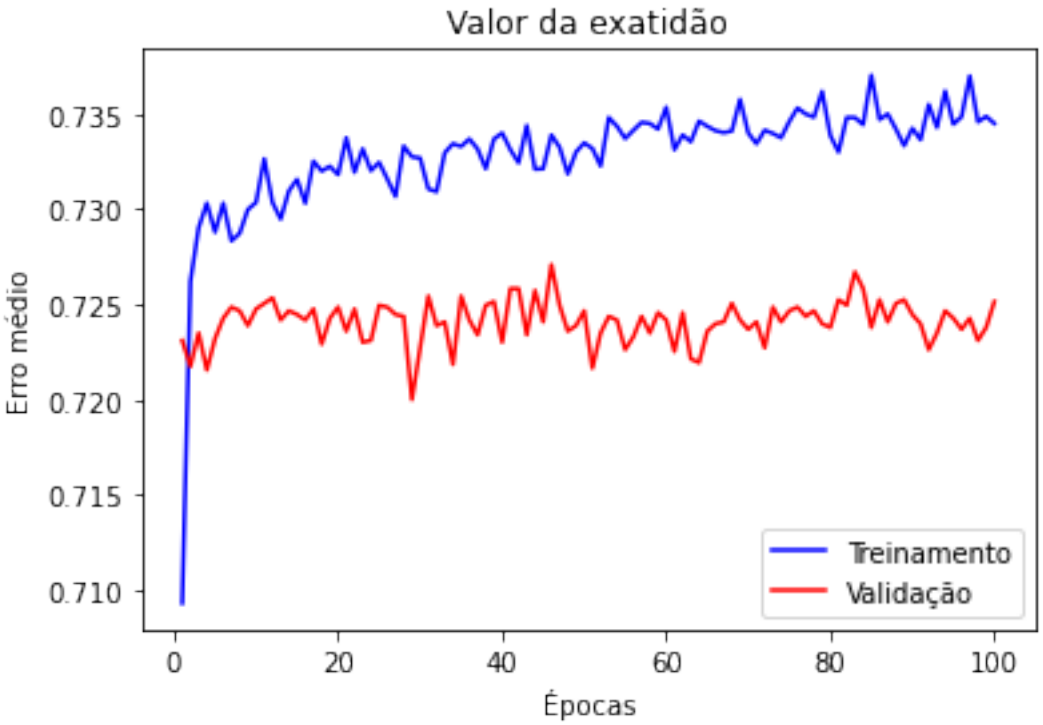
Saída eseprada:

```
Train on 46657 samples, validate on 11000 samples
Epoch 1/100
46657/46657 [==============================] - 3s 66us/sample - loss: 0.6406 - accuracy: 0.6295

Epoch 100/100
46657/46657 [==============================] - 0s 6us/sample - loss: 0.5494 - accuracy: 0.7360 -
```

Gráficos do processo de treinamento.

[60]:
```
import matplotlib.pyplot as plt

# Definir vetores com valores da função de custo e da métrica para os dados de␣
 ↪treinamento e de validação
history_dict = history.history
custo = history_dict['loss']
exatidao = history_dict['accuracy']
custo_val = history_dict['val_loss']
exatidao_val = history_dict['val_accuracy']

# Criar vetor de épocas
epocas = range(1, len(custo) + 1)

# Fazer o gráfico dos valores da função de custo
plt.plot(epocas, custo, 'b', label='Treinamento')
plt.plot(epocas, custo_val, 'r', label='Validação')
plt.title('Valor da função de custo')
plt.xlabel('Épocas')
plt.ylabel('Custo')
plt.legend()
plt.show()

# Fazer o gráfico dos valores da métrica
plt.plot(epocas, exatidao, 'b', label='Treinamento')
plt.plot(epocas, exatidao_val, 'r', label='Validação')
plt.title('Valor da exatidão')
plt.xlabel('Épocas')
plt.ylabel('Erro médio')
plt.legend()
plt.show()
```


Valor da função de custo
Treinamento
Validação
Custo
0.585
0.580
0.575
0.570
0.565
0.560
0.555
0.550
0
20
40
60
80
100
Épocas


Valor da exatidão
Erro médio
0.735
0.730
0.725
0.720
0.715
0.710
Treinamento
Validação
0
20
40
60
80
100
Épocas

8 Avaliação e teste da RNA

8.1 Execício #18: Avaliação da RNA

Apresente a avaliação de desempenho geral para os dados de treinamento, validação e teste (use o método `evaluate`).

[63]:
```
# Para você fazer:

# Avalia desempenho da RNA para os dados de treinamento, validação e teste
# Insire seu código aqui
#
eval_train = rna.evaluate(x_train, y_train, verbose=0)
eval_val = rna.evaluate(x_val, y_val, verbose=0)
eval_test = rna.evaluate(x_test, y_test, verbose=0)

# Apresenta resultados
print('Dados de treinamento: Função de custo =', eval_train[0], '- Exatidão =',␣
 ↪eval_train[1])
print('Dados de validação: Função de custo =', eval_val[0], '- Exatidão =',␣
 ↪eval_val[1])
print('Dados de teste: Função de custo =', eval_test[0], '- Exatidão =',␣
 ↪eval_test[1])
```

```
Dados de treinamento: Função de custo = 0.5332759618759155 - Exatidão =
0.7422086000442505
Dados de validação: Função de custo = 0.5570616722106934 - Exatidão =
0.7251456379890442
Dados de teste: Função de custo = 0.5456126928329468 - Exatidão =
0.7311911582946777
```

Saída esperada:

```
Dados de treinamento: Função de custo = 0.5353194371969623 - Exatidão = 0.73892945
Dados de validação: Função de custo = 0.5486114514524286 - Exatidão = 0.73072726
Dados de teste: Função de custo = 0.5468874634179202 - Exatidão = 0.7303636
```

Observa-se que a RNA acerta se o passageiro tem ou não doença cardiovascular em cerca de 73% dos casos.

8.2 Exercício #19: Teste da RNA

Apresente alguns exemplos de resultados do conjunto de teste (use o método `predict`).

Faça um gráfico com as classes reais e previstas sobrepostas com cores diferentes para cerca de 150 exemplos é interessante para verificar os resultados.

[66]:
```
# Para você fazer: Calcula previsão da RNA para os dados de teste

# Cálculo da probabilidade de sobrevivência usando a RNA
```

```
# Insire seu código aqui
#
classe_prev = rna.predict(x_test)
# Verifica se sobreviveu ou não
# Insire seu código aqui
#
classe_prev = (classe_prev > 0.5).astype("int32")
# Fazer o gráfico das classes reais e previstas
plt.figure(figsize=(18, 5))
plt.plot(classe_prev[:150], 'bo', label='Classe prevista')
plt.plot(y_test[:150], 'ro', label='Classe real')
plt.title('Classes reais e preivistas - Conjunto de teste')
plt.xlabel('Exemplos')
plt.ylabel('Classe')
plt.legend()
plt.show()
```