Data Mining no dataframe Saúde RS 2022 (Ocorrências de COVID-19 e outras doenças respiratórias)

Autores: João Davi Rigo Mazzarolo e Raul Steinmetz
Professor orientador: Dr. Joaquim Assunção

1. Visualização do Dataset

Visualização inicial do dataset

```
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> import pandas as pd
>>> data = pd.read_csv('./data/saudeRS_2022.csv', sep=';')
>>> data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 513610 entries, 0 to 513609
Data columns (total 30 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   COD_IBGE                513610 non-null  int64
 1   MUNICIPIO               513610 non-null  object
 2   COD_REGIAO_COVID        513610 non-null  int64
 3   REGIAO_COVID            513610 non-null  object
 4   SEXO                    513610 non-null  object
 5   FAIXAETARIA             513610 non-null  object
 6   CRITERIO                513610 non-null  object
 7   DATA_CONFIRMACAO        513610 non-null  object
 8   DATA_SINTOMAS           513610 non-null  object
 9   DATA_INCLUSAO           513610 non-null  object
 10  DATA_EVOLUCAO           34209 non-null   object
 11  EVOLUCAO                513610 non-null  object
 12  HOSPITALIZADO           513610 non-null  object
 13  FEBRE                   513590 non-null  object
 14  TOSSE                   513590 non-null  object
 15  GARGANTA                513590 non-null  object
 16  DISPNEIA                507583 non-null  object
 17  OUTROS                  513590 non-null  object
 18  CONDICOES               74321 non-null   object
 19  GESTANTE                513610 non-null  object
 20  DATA_INCLUSAO_OBITO     10543 non-null   object
 21  DATA_EVOLUCAO_ESTIMADA  481128 non-null  object
 22  RACA_COR                513610 non-null  object
 23  ETNIA_INDIGENA          497654 non-null  object
 24  PROFISSIONAL_SAUDE      513610 non-null  object
 25  BAIRRO                  513539 non-null  object
 26  SRAG                    513610 non-null  object
 27  FONTE_INFORMACAO        513596 non-null  object
 28  PAIS_NASCIMENTO         497306 non-null  object
 29  PES_PRIV_LIBERDADE      513610 non-null  object
dtypes: int64(2), object(28)
memory usage: 117.6+ MB
```

2. Observando dados das colunas

```
>>> data['FEBRE'].unique()
array(['NAO', 'SIM', nan], dtype=object)
>>>
```

```
'''                                     NOTES - all the data bellow is now in lower case
                                              - all the data bellow was collected on python shell using data['COLUMN_NAME'].unique()

++ COD_IBGE: unique values

++ MUNICIPIO: to many values to track

++ COD_REGIAO_COVID: [16, 14,  1, 10, 11,  2, 12, 17,  7, 15, 20, 21,  5, 13,  3, 18,  6, 9,  8,  4, 19]

++ REGIAO_COVID:['BAGE - R22', 'PASSO FUNDO - R17 R18 R19', 'SANTA MARIA - R01 R02',
       'IJUI - R13', 'SANTA ROSA - R14', 'URUGUAIANA - R03',
       'PALMEIRA DAS MISSOES - R15 R20',
       'CAXIAS DO SUL - R23 R24 R25 R26', 'PORTO ALEGRE - R10',
       'PELOTAS - R21', 'LAJEADO - R29 R30', 'GUAIBA - R09',
       'NOVO HAMBURGO - R07', 'ERECHIM - R16', 'CAPAO DA CANOA - R04 R05',
       'CACHOEIRA DO SUL - R27', 'CANOAS - R08', 'CRUZ ALTA - R12',
       'SANTO ANGELO - R11', 'TAQUARA - R06', 'SANTA CRUZ DO SUL - R28']

++ SEXO: ['Feminino', 'Masculino']

++ FAIXA_ETARIA: ['15 a 19', '20 a 29', '40 a 49', '70 a 79', '50 a 59', '80 e mais', '60 a 69', '30 a 39', '10 a 14', '01 a 04', '05 a 09', '<1']

++ CRITERIO: [RT-PCR', 'TESTE RÁPIDO', 'Outros Testes', 'Clínico Epidemiológico', 'Clínico-Imagem', 'Clínico', 'Outros testes']
++ DATA_CONFIRMACAO: unique values
++ DATA_SINTOMAS: unique values
++ DATA_INCLUSAO: unique values
++ DATA_EVOLUCAO: unique values
```

++ EVOLUCAO: ['RECUPERADO', 'OBITO', 'OBITO OUTRAS CAUSAS']

++ HOSPITALIZADO: ['NAO', 'SIM']

++ FEBRE: ['NAO', 'SIM', nan]

++ TOSSE: ['NAO', 'SIM', nan]

++ GARGANTA: ['NAO', 'SIM', nan]

++ DISPNEIA: ['NAO', 'SIM', nan]

++ OUTROS: ['NAO', 'SIM', nan]

++ CONDICOES: ['nan', 'unique combinations']

++ GESTANTE: ['NAO', 'SIM']

++ DATA_INCLUSAO_OBITO: unique values
++ DATA_EVOLUCAO_ESTIMADA: unique values

++ RACA_COR: ['BRANCA', 'NAO INFORMADO', 'PARDA', 'PRETA', 'AMARELA', 'INDIGENA']

++ INDIGENA: [tribes names, 'nan', 'nao encontrado']

++ PROFISSIONAL_SAUDE: ['NAO', 'NAO INFORMADO', 'SIM'

++ BAIRRO: neighborhood names, semi-unique values

++ SRAG: ['NAO', 'SIM']

++ FONTE_INFORMACAO: ['E-SUS', 'SIVEP HOSP', 'SIVEP US', nan]

++ PAIS_NASCIMENTO: [countries names]

++ PES_PRIV_LIBERDADE: ['SIM', 'NAO']

3. Pré-processamento

1. Conversão das strings para lowercase
2. Preenchimento de campos *nan*
3. Conversão de colunas binárias para zeros e uns

```
# converting all strings to lowercase
data = data.apply(lambda x: x.astype(str).str.lower())

# filling missing values with zero
data.fillna(0, inplace=True)

# turning columns into zeros and ones
# note: i want to add .astype(int) in the end
data['SEXO'] = data['SEXO'].map({'masculino': 0, 'feminino': 1}).astype(int)
data['HOSPITALIZADO'] = data['HOSPITALIZADO'].map({'sim': SIM, 'nao': NAO}).astype(int)
data['FEBRE'] = data['FEBRE'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
data['TOSSE'] = data['TOSSE'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
data['GARGANTA'] = data['GARGANTA'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
data['DISPNEIA'] = data['DISPNEIA'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
data['OUTROS'] = data['OUTROS'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
data['GESTANTE'] = data['GESTANTE'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
data['PROFISSIONAL_SAUDE'] = data['PROFISSIONAL_SAUDE'].map({'sim': SIM, 'nao': NAO, 'nao informado': NAO}).astype(int)
data['SRAG'] = data['SRAG'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
data['PES_PRIV_LIBERDADE'] = data['PES_PRIV_LIBERDADE'].map({'sim': SIM, 'nao': NAO, 'nan': NAO}).astype(int)
```

3. Pré-processamento

4. Conversão de Colunas não binárias para zeros e uns
   a. Condições
   b. País de Nascimento
   c. Etnia Indígena

```
# pais_nascimento will be implemented as yes or no
data['BRASILEIRO'] = np.ones(len(data['PAIS_NASCIMENTO'])).astype(int)
itr = 0
for i in data['PAIS_NASCIMENTO']:
       if i != 'brasil':
              data['BRASILEIRO'][itr] = ESTRANGEIRO
       itr += 1

data.drop(['PAIS_NASCIMENTO'], axis=1, inplace=True)
```

3. Pré-processamento

4. One Hot Encoding
   a. Faixa etária
   b. Critério
   c. Raça/Cor
   d. Fonte

```
# FONTE DE INFORMACAO:  ['E-SUS', 'SIVEP HOSP', 'SIVEP US', nan]
data['FONTE_SUS'] = np.zeros(len(data['FONTE_INFORMACAO'])).astype(int)
data['FONTE_HOSP'] = np.zeros(len(data['FONTE_INFORMACAO'])).astype(int)
data['FONTE_US'] = np.zeros(len(data['FONTE_INFORMACAO'])).astype(int)

itr = 0
for i in data['FONTE_INFORMACAO']:
        if i == 'e-sus':
                data['FONTE_SUS'][itr] = 1
        elif i == 'sivep hosp':
                data['FONTE_HOSP'][itr] = 1
        elif i == 'sivep us':
                data['FONTE_US'][itr] = 1
        itr += 1

data.drop(['FONTE_INFORMACAO'], axis=1, inplace=True)
```

3. Pré-processamento

5. Criação de dataframe booleano para apriori

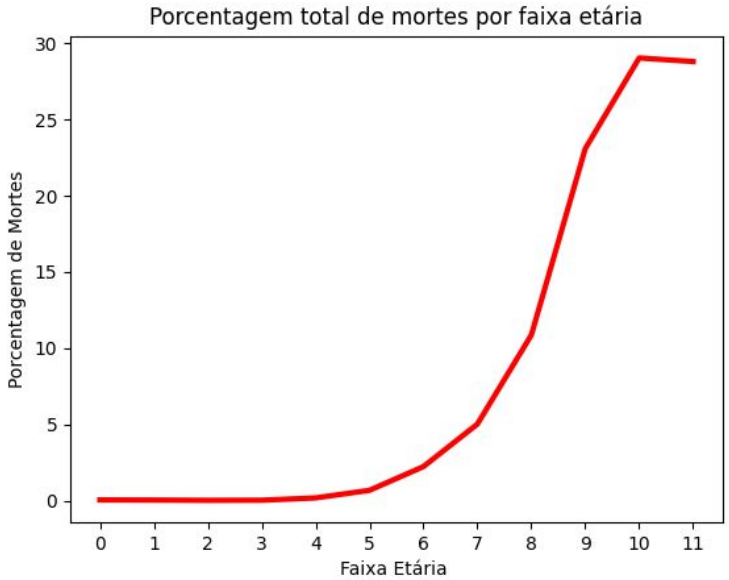
```
import pandas as pd

data = pd.read_csv('./data/final.csv')
df = pd.read_csv('./data/final.csv')
df.drop(['Unnamed: 0', 'COD_IBGE', 'MUNICIPIO', 'COD_REGIAO_COVID', 'REGIAO_COVID', 'BAIRRO'], axis=1, inplace=True)
df = df.apply(lambda x: x == 1)
df.to_csv('./data/bool.csv')
```

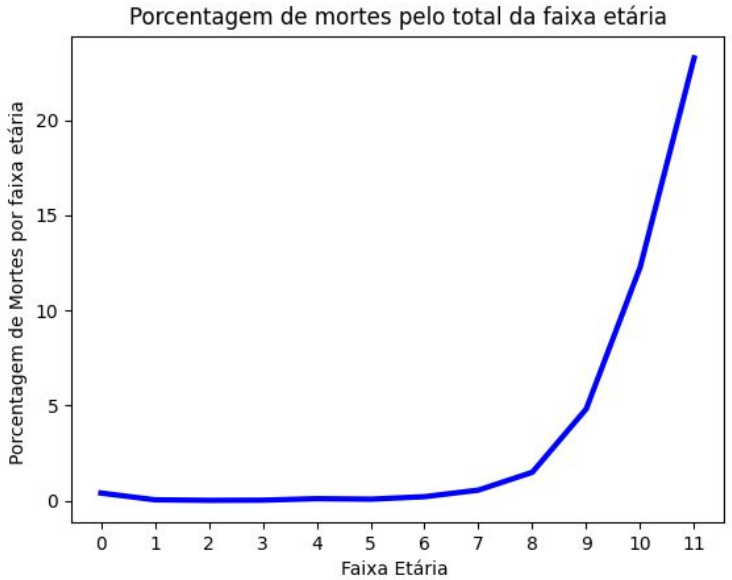
3. Pré-processamento

Estado final do dataframe

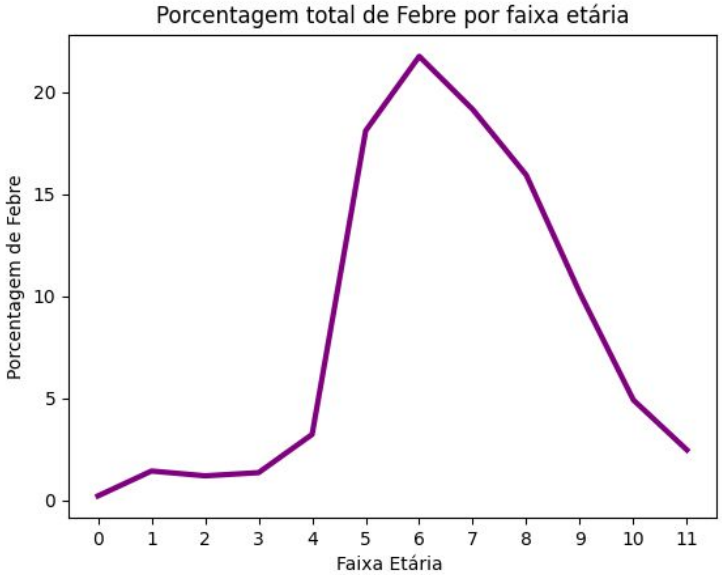
```
>>> data = pd.read_csv('./data/bool.csv')
>>> data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 513610 entries, 0 to 513609
Data columns (total 45 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Unnamed: 0            513610 non-null  int64
 1   SEXO                  513610 non-null  bool
 2   HOSPITALIZADO         513610 non-null  bool
 3   FEBRE                 513610 non-null  bool
 4   TOSSE                 513610 non-null  bool
 5   GARGANTA              513610 non-null  bool
 6   DISPNEIA              513610 non-null  bool
 7   OUTROS                513610 non-null  bool
 8   CONDICOES             513610 non-null  bool
 9   GESTANTE              513610 non-null  bool
 10  ETNIA_INDIGENA        513610 non-null  bool
 11  PROFISSIONAL_SAUDE    513610 non-null  bool
 12  SRAG                  513610 non-null  bool
 13  PES_PRIV_LIBERDADE    513610 non-null  bool
 14  BRASILEIRO            513610 non-null  bool
 15  IDADE_1               513610 non-null  bool
 16  IDADE_1_4             513610 non-null  bool
 17  IDADE_5_9             513610 non-null  bool
 18  IDADE_10_14           513610 non-null  bool
 19  IDADE_15_19           513610 non-null  bool
 20  IDADE_20_29           513610 non-null  bool
 21  IDADE_30_39           513610 non-null  bool
 22  IDADE_40_49           513610 non-null  bool
 23  IDADE_50_59           513610 non-null  bool
 24  IDADE_60_69           513610 non-null  bool
 25  IDADE_70_79           513610 non-null  bool
 26  IDADE_80              513610 non-null  bool
 27  TESTE_RTPCR           513610 non-null  bool
 28  TESTE_RAPIDO          513610 non-null  bool
 29  TESTE_OUTRO           513610 non-null  bool
 30  TESTE_CLINICO_EPI     513610 non-null  bool
 31  TESTE_CLINICO_IMG     513610 non-null  bool
 32  TESTE_CLINICO         513610 non-null  bool
 33  EVOLUCAO_RECUPERADO   513610 non-null  bool
 34  EVOLUCAO_OBITO        513610 non-null  bool
 35  EVOLUCAO_OBITO_OC     513610 non-null  bool
 36  RACA_COR_BRANCA       513610 non-null  bool
 37  RACA_COR_NA           513610 non-null  bool
 38  RACA_COR_PARDA        513610 non-null  bool
 39  RACA_COR_PRETA        513610 non-null  bool
 40  RACA_COR_AMARELA      513610 non-null  bool
 41  RACA_COR_INDIGENA     513610 non-null  bool
 42  FONTE_SUS             513610 non-null  bool
 43  FONTE_HOSP            513610 non-null  bool
 44  FONTE_US              513610 non-null  bool
dtypes: bool(44), int64(1)
memory usage: 25.5 MB
```

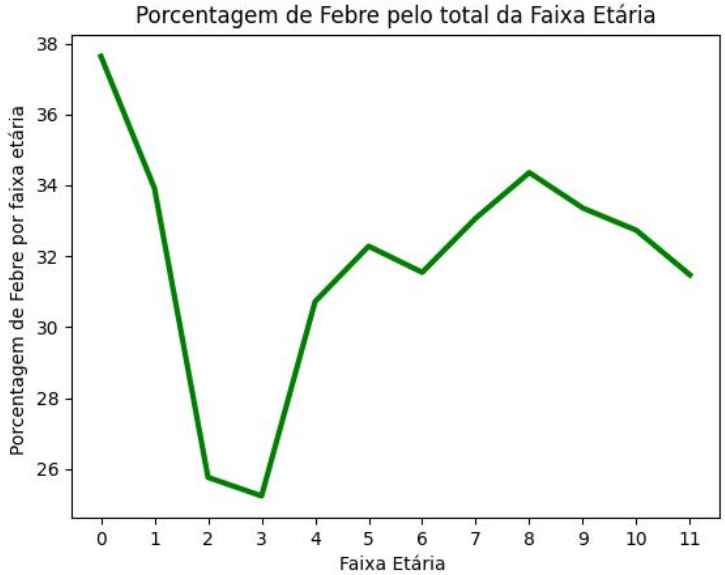
4. Mineração de Dados - Alguns gráficos interessantes





4. Mineração de Dados - Alguns gráficos interessantes





4. Mineração de Dados - Associação

Começando por tirar colunas menos relevantes e converter colunas de inteiros para booleanos para melhorar o desempenho na hora de rodar o Apriori.

```
df.drop(['Unnamed: 0', 'COD_IBGE', 'MUNICIPIO', 'COD_REGIAO_COVID', 'REGIAO_COVID', 'BAIRRO',
         'TESTE_RTPCR', 'TESTE_RAPIDO', 'TESTE_OUTRO', 'TESTE_CLINICO_EPI', 'TESTE_CLINICO_IMG',
         'TESTE_CLINICO', 'PES_PRIV_LIBERDADE', 'BRASILEIRO', 'RACA_COR_BRANCA', 'RACA_COR_NA',
         'RACA_COR_PARDA', 'RACA_COR_PRETA', 'RACA_COR_AMARELA', 'RACA_COR_INDIGENA', 'FONTE_SUS',
         'FONTE_HOSP', 'FONTE_US'], axis=1, inplace=True)
```

4. Mineração de Dados - Associação

Criando o modelo do apriori, passando seus parâmetros e salvando as regras de associação geradas em um arquivo CSV.

```
# Building the model
frq_items = apriori(df, min_support = 0.01, use_colnames = True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
index = np.arange(len(rules.index))
rules['index'] = index
rules.set_index('index', inplace=True)
rules.to_csv("./data/rules.csv")
```

4. Mineração de Dados - Associação

Separando os resultados em diferentes CSVs para poder analisá-los melhor

```
df = pd.read_csv('./data/rules.csv')

print(df.info())

deathdf = df[(df["consequents"] == "frozenset({'EVOLUCAO_OBITO'})") | (df["consequents"] == "frozenset({'EVOLUCAO_OBITO_OC'})")]

deathdf.to_csv('./data/deathRules.csv')

recoverydf = df[(df["consequents"] == "frozenset({'EVOLUCAO_RECUPERADO'})")]

recoverydf.to_csv('./data/recoveryRules.csv')

feverdf = df[(df["consequents"] == "frozenset({'FEBRE'})")]

feverdf.to_csv('./data/feverRules.csv')
```

4. Mineração de Dados - Associação

Regras que levam ao óbito:

Antecedents	Consequents	Antecedent su	Consequent su	Support	Confidence	Lift	Leverage	Conviction
frozenset({'SRAG', 'CONDICOES', 'DISPNEIA'})	frozenset({'EVOLUCAO_OBITO'})	0.04	0.02	0.02	0.39	18.84	0.01	1.6
frozenset({'CONDICOES', 'HOSPITALIZADO', 'DISPNEIA'})	frozenset({'EVOLUCAO_OBITO'})	0.04	0.02	0.01	0.38	18.56	0.01	1.58
frozenset({'CONDICOES', 'SRAG', 'HOSPITALIZADO', 'DISPNEIA'})	frozenset({'EVOLUCAO_OBITO'})	0.04	0.02	0.01	0.38	18.56	0.01	1.58
frozenset({'SRAG', 'CONDICOES'})	frozenset({'EVOLUCAO_OBITO'})	0.05	0.02	0.02	0.36	17.74	0.02	1.54
frozenset({'CONDICOES', 'HOSPITALIZADO'})	frozenset({'EVOLUCAO_OBITO'})	0.05	0.02	0.02	0.36	17.39	0.02	1.52
frozenset({'SRAG', 'CONDICOES', 'HOSPITALIZADO'})	frozenset({'EVOLUCAO_OBITO'})	0.05	0.02	0.02	0.36	17.39	0.02	1.52
frozenset({'CONDICOES', 'TOSSE', 'SRAG'})	frozenset({'EVOLUCAO_OBITO'})	0.03	0.02	0.01	0.33	15.94	0.01	1.46
frozenset({'SRAG', 'DISPNEIA'})	frozenset({'EVOLUCAO_OBITO'})	0.05	0.02	0.02	0.33	15.91	0.02	1.45
frozenset({'HOSPITALIZADO', 'DISPNEIA'})	frozenset({'EVOLUCAO_OBITO'})	0.05	0.02	0.02	0.32	15.64	0.02	1.44
frozenset({'SRAG', 'HOSPITALIZADO', 'DISPNEIA'})	frozenset({'EVOLUCAO_OBITO'})	0.05	0.02	0.02	0.32	15.64	0.02	1.44
frozenset({'TOSSE', 'CONDICOES', 'HOSPITALIZADO'})	frozenset({'EVOLUCAO_OBITO'})	0.03	0.02	0.01	0.32	15.64	0.01	1.44
frozenset({'CONDICOES', 'TOSSE', 'SRAG', 'HOSPITALIZADO'})	frozenset({'EVOLUCAO_OBITO'})	0.03	0.02	0.01	0.32	15.64	0.01	1.44

4. Mineração de Dados - Associação

Regras que levam à recuperação:

Antecedents	Consequents	Antecedent Sup	Consequent Su	Support	Confidence	Lift	Leverage	Conviction
frozenset({'PROFISSIONAL_SAUDE'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.05	0.98	0.05	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'SEXO'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.04	0.98	0.04	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'FEBRE'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.01	0.98	0.01	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'TOSSE'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.02	0.98	0.02	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'GARGANTA'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.02	0.98	0.02	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'OUTROS'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.03	0.98	0.03	1	1.02	0	inf
frozenset({'IDADE_30_39', 'PROFISSIONAL_SAUDE'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.02	0.98	0.02	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'IDADE_40_49'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.01	0.98	0.01	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'SEXO', 'FEBRE'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.01	0.98	0.01	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'TOSSE', 'SEXO'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.02	0.98	0.02	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'SEXO', 'GARGANTA'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.01	0.98	0.01	1	1.02	0	inf
frozenset({'PROFISSIONAL_SAUDE', 'SEXO', 'OUTROS'})	frozenset({'EVOLUCAO_RECUPERADO'})	0.02	0.98	0.02	1	1.02	0	inf

4. Mineração de Dados - Associação

Regras que levam à febre:

Antecedents	Consequents	Antecedent su	Consequent su	Support	Confidence	Lift	Leverage	Conviction
frozenset({'GARGANTA', 'TOSSE', 'CONDICOES', 'DISPNEIA'})	frozenset({'FEBRE'})	0.02	0.32	0.01	0.67	2.06	0.01	2.03
frozenset({'TOSSE', 'HOSPITALIZADO', 'EVOLUCAO_RECUPERADO'})	frozenset({'FEBRE'})	0.03	0.32	0.02	0.67	2.05	0.01	2.02
frozenset({'TOSSE', 'SRAG', 'HOSPITALIZADO', 'EVOLUCAO_RECUPERADO'})	frozenset({'FEBRE'})	0.03	0.32	0.02	0.67	2.05	0.01	2.02
frozenset({'TOSSE', 'SRAG', 'EVOLUCAO_RECUPERADO'})	frozenset({'FEBRE'})	0.03	0.32	0.02	0.67	2.05	0.01	2.02
frozenset({'TOSSE', 'HOSPITALIZADO', 'EVOLUCAO_RECUPERADO', 'DISPNEIA'})	frozenset({'FEBRE'})	0.03	0.32	0.02	0.66	2.05	0.01	2.02
frozenset({'TOSSE', 'SRAG', 'EVOLUCAO_RECUPERADO', 'DISPNEIA', 'HOSPITALIZADO'})	frozenset({'FEBRE'})	0.03	0.32	0.02	0.66	2.05	0.01	2.02
frozenset({'TOSSE', 'SRAG', 'EVOLUCAO_RECUPERADO', 'DISPNEIA'})	frozenset({'FEBRE'})	0.03	0.32	0.02	0.66	2.05	0.01	2.02
frozenset({'TOSSE', 'HOSPITALIZADO', 'OUTROS'})	frozenset({'FEBRE'})	0.02	0.32	0.01	0.66	2.02	0.01	1.97
frozenset({'TOSSE', 'SRAG', 'HOSPITALIZADO', 'OUTROS'})	frozenset({'FEBRE'})	0.02	0.32	0.01	0.66	2.02	0.01	1.97
frozenset({'TOSSE', 'HOSPITALIZADO'})	frozenset({'FEBRE'})	0.05	0.32	0.03	0.66	2.02	0.02	1.96
frozenset({'TOSSE', 'SRAG', 'HOSPITALIZADO'})	frozenset({'FEBRE'})	0.05	0.32	0.03	0.66	2.02	0.02	1.96
frozenset({'TOSSE', 'SRAG', 'OUTROS'})	frozenset({'FEBRE'})	0.02	0.32	0.01	0.66	2.02	0.01	1.96

4. Mineração de Dados - Classificação

A começo foram feitas algumas modificações no DF

```
df.drop(['Unnamed: 0', 'COD_IBGE', 'MUNICIPIO', 'COD_REGIAO_COVID', 'REGIAO_COVID', 'BAIRRO'], axis=1, inplace=True)
df.drop(['PES_PRIV_LIBERDADE', 'BRASILEIRO', 'TESTE_RTPCR', 'TESTE_RAPIDO',
             'TESTE_OUTRO', 'TESTE_CLINICO_EPI', 'TESTE_CLINICO_IMG', 'TESTE_CLINICO', 'RACA_COR_BRANCA',
                 'RACA_COR_NA', 'RACA_COR_PARDA', 'RACA_COR_PRETA', 'RACA_COR_AMARELA', 'RACA_COR_INDIGENA',
                    'FONTE_SUS', 'FONTE_HOSP', 'FONTE_US'], axis=1, inplace=True)
```

```
# creating morreu column
df['MORREU'] = np.zeros(len(df['EVOLUCAO_RECUPERADO'])).astype(int)
itr = 0
for i in df['EVOLUCAO_RECUPERADO']:
    if i != 1:
        df['MORREU'][itr] = 1
    itr += 1

df.drop(['EVOLUCAO_RECUPERADO', 'EVOLUCAO_OBITO', 'EVOLUCAO_OBITO_OC'], axis=1, inplace=True)
```

4. Mineração de Dados - Classificação

Divisão do DF

```
data = df.drop(['MORREU'], axis=1)
target = df['MORREU']

# print(df.info())

X_train_full, X_test, y_train_full, y_test  = train_test_split(data, target)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full)
```

4. Mineração de Dados - Classificação

Função de cálculo de accuracy

```
def precision_score(y_true, y_pred):
    correct_preds = 0
    itr = 0
    for i in y_true:
        if y_pred[itr] == i:
            correct_preds += 1
        itr += 1
    return(correct_preds/len(y_true))
```

4. Mineração de Dados - Classificação

- O primeiro preditor testado foi uma rede neural, com arquitetura de multilayer perceptron (todos os neurônios de uma camada são conectados com todos os neurônios da camada que provêm seu input e da camada que recebe seu output)
- Construída com *tensorflow* e *keras*
- 3 camadas densas com 50, 30 e 2 neurônios respectivamente

4. Mineração de Dados - Classificação

MLP

```
if RUNALL:
    model = keras.models.Sequential()
    model.add(keras.layers.Dense(50, activation='relu'))
    model.add(keras.layers.Dense(30, activation='relu'))
    model.add(keras.layers.Dense(2, activation='softmax'))

    model.compile(loss=keras.losses.sparse_categorical_crossentropy,
                  optimizer=keras.optimizers.SGD(), metrics=keras.metrics.sparse_categorical_accuracy)
    history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid))
```

4. Mineração de Dados - Classificação

A precisão da classificação no final do treinamento foi de 0.9819 (98,19%), como mostra na figura a seguir.

```
Epoch 1/10
9029/9029 [==============================] - 8s 903us/step - loss: 0.0532 - sparse_categorical_accuracy: 0.9787 - val_loss: 0.0377 - val_sparse_categorical_accuracy: 0.9806
Epoch 2/10
9029/9029 [==============================] - 7s 803us/step - loss: 0.0370 - sparse_categorical_accuracy: 0.9810 - val_loss: 0.0373 - val_sparse_categorical_accuracy: 0.9807
Epoch 3/10
9029/9029 [==============================] - 7s 791us/step - loss: 0.0366 - sparse_categorical_accuracy: 0.9812 - val_loss: 0.0367 - val_sparse_categorical_accuracy: 0.9814
Epoch 4/10
9029/9029 [==============================] - 7s 785us/step - loss: 0.0362 - sparse_categorical_accuracy: 0.9813 - val_loss: 0.0364 - val_sparse_categorical_accuracy: 0.9815
Epoch 5/10
9029/9029 [==============================] - 7s 779us/step - loss: 0.0360 - sparse_categorical_accuracy: 0.9813 - val_loss: 0.0362 - val_sparse_categorical_accuracy: 0.9814
Epoch 6/10
9029/9029 [==============================] - 7s 791us/step - loss: 0.0358 - sparse_categorical_accuracy: 0.9816 - val_loss: 0.0361 - val_sparse_categorical_accuracy: 0.9814
Epoch 7/10
9029/9029 [==============================] - 7s 780us/step - loss: 0.0356 - sparse_categorical_accuracy: 0.9818 - val_loss: 0.0360 - val_sparse_categorical_accuracy: 0.9812
Epoch 8/10
9029/9029 [==============================] - 7s 769us/step - loss: 0.0355 - sparse_categorical_accuracy: 0.9817 - val_loss: 0.0358 - val_sparse_categorical_accuracy: 0.9817
Epoch 9/10
9029/9029 [==============================] - 7s 788us/step - loss: 0.0353 - sparse_categorical_accuracy: 0.9818 - val_loss: 0.0359 - val_sparse_categorical_accuracy: 0.9813
Epoch 10/10
9029/9029 [==============================] - 8s 927us/step - loss: 0.0353 - sparse_categorical_accuracy: 0.9819 - val_loss: 0.0355 - val_sparse_categorical_accuracy: 0.9817
```

4. Mineração de Dados - Classificação

O segundo preditor testado foi um simples regressor. Os parâmetros usados foram os padrões da biblioteca *sklearn*. O código a seguir mostra o processo. Esse regressor obteve precisão de 0.9798(97,98%).

```
# PREDICTOR 2 - LINEAR REGRESSION
if RUNALL:
    model = LinearRegression()
    model.fit(X_train_full, y_train_full)

    linear_model_pred = model.predict(X_test)
    linear_model_pred_bool = np.zeros(len(X_test))
    itr = 0
    for i in linear_model_pred:
        if 1 - i > 0.5:
            linear_model_pred_bool[itr] = 0
        else:
            linear_model_pred_bool[itr] = 1
        itr += 0
```

```
0.9798369196981379
```

4. Mineração de Dados - Classificação

O terceiro preditor usado foi o método de ensemble learning Random Forest. Os parâmetros para tal foram os padrões do *sklearn*. O código a seguir mostra o processo. Esse regressor obteve precisão de 0.9819 (98,19%) assim como o MLP.

```
# PREDICTOR 3 - RANDOM FOREST
if RUNALL:
    model = RandomForestClassifier()
    model.fit(X_train_full, y_train_full)
    print(precision_score(y_test, model.predict(X_test)))
```

```
0.9819163103665802
```

4. Mineração de Dados - Classificação

Dumb Predictor

```
# PREDICTOR 4 - DUMB PREDICTOR
if RUNALL:
    class DumbPred:
        def __init__(self):
            print("I don't need parameters!")
        def predict(self, a):
            return np.zeros(len(a))

    model = DumbPred()
    print(precision_score(y_test, model.predict(X_test)))
```

```
I don't need parameters!
0.9798369196981379
```

4. Mineração de Dados - Classificação

O mesmo processo foi feito para um preditor de febre

```
# PREPARING TRAIN/VALID/TEST

df.drop(['Unnamed: 0', 'COD_IBGE', 'MUNICIPIO', 'COD_REGIAO_COVID', 'REGIAO_COVID', 'BAIRRO'], axis=1, inplace=True)
df.drop(['PES_PRIV_LIBERDADE', 'BRASILEIRO', 'TESTE_RTPCR', 'TESTE_RAPIDO',
             'TESTE_OUTRO', 'TESTE_CLINICO_EPI', 'TESTE_CLINICO_IMG', 'TESTE_CLINICO', 'RACA_COR_BRANCA',
                 'RACA_COR_NA', 'RACA_COR_PARDA', 'RACA_COR_PRETA', 'RACA_COR_AMARELA', 'RACA_COR_INDIGENA',
                    'FONTE_SUS', 'FONTE_HOSP', 'FONTE_US', 'EVOLUCAO_RECUPERADO', 'EVOLUCAO_OBITO',
                         'EVOLUCAO_OBITO_OC', 'PROFISSIONAL_SAUDE'], axis=1, inplace=True)

print(df.info())

# target = fever?
target = df['FEBRE']
data = df.drop(['FEBRE'], axis=1)

X_train_full, X_test, y_train_full, y_test  = train_test_split(data, target)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full)
```

```
if RUNALL:
    model = keras.models.Sequential()
    model.add(keras.layers.Dense(256, activation='relu'))
    model.add(keras.layers.Dense(100, activation='relu'))
    model.add(keras.layers.Dense(2, activation='softmax'))

    model.compile(loss=keras.losses.sparse_categorical_crossentropy,
                  optimizer=keras.optimizers.SGD(), metrics=keras.metrics.sparse_categorical_accuracy)
    history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid))


    model.save('./saved_models/mlp_covid_febre.h5')
```

```
# PREDICTOR 2 - LINEAR REGRESSION AND CLASSIFICATION

if RUNALL:
    model = LinearRegression()
    model.fit(X_train_full, y_train_full)

    linear_model_pred = model.predict(X_test)
    linear_model_pred_bool = np.zeros(len(X_test))
    itr = 0
    for i in linear_model_pred:
        if 1 - i > 0.5:
            linear_model_pred_bool[itr] = 0
        else:
            linear_model_pred_bool[itr] = 1
        itr += 0

    print(precision_score(y_test, linear_model_pred_bool))


# PREDICTOR 3 - RANDOM FOREST
if RUNALL:
    model = RandomForestClassifier()
    model.fit(X_train_full, y_train_full)
    print(precision_score(y_test, model.predict(X_test)))
```

```
Epoch 1/10
9029/9029 [==============================] - 10s 1ms/step - loss: 0.5605 - sparse_categorical_accuracy: 0.6992 - val_loss: 0.5538 - val_sparse_categorical_accuracy: 0.7051
Epoch 2/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5535 - sparse_categorical_accuracy: 0.7030 - val_loss: 0.5531 - val_sparse_categorical_accuracy: 0.7045
Epoch 3/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5529 - sparse_categorical_accuracy: 0.7038 - val_loss: 0.5526 - val_sparse_categorical_accuracy: 0.7046
Epoch 4/10
9029/9029 [==============================] - 10s 1ms/step - loss: 0.5527 - sparse_categorical_accuracy: 0.7045 - val_loss: 0.5532 - val_sparse_categorical_accuracy: 0.7043
Epoch 5/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5525 - sparse_categorical_accuracy: 0.7042 - val_loss: 0.5522 - val_sparse_categorical_accuracy: 0.7070
Epoch 6/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5524 - sparse_categorical_accuracy: 0.7045 - val_loss: 0.5518 - val_sparse_categorical_accuracy: 0.7071
Epoch 7/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5523 - sparse_categorical_accuracy: 0.7044 - val_loss: 0.5525 - val_sparse_categorical_accuracy: 0.7053
Epoch 8/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5522 - sparse_categorical_accuracy: 0.7048 - val_loss: 0.5520 - val_sparse_categorical_accuracy: 0.7068
Epoch 9/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5522 - sparse_categorical_accuracy: 0.7052 - val_loss: 0.5519 - val_sparse_categorical_accuracy: 0.7072
Epoch 10/10
9029/9029 [==============================] - 9s 1ms/step - loss: 0.5521 - sparse_categorical_accuracy: 0.7046 - val_loss: 0.5518 - val_sparse_categorical_accuracy: 0.7072
0.6754904480424913
0.7045006736602727
```

4. Mineração de Dados - Regressão

Outro preditor feito foi um regressor linear, que a partir de certas colunas da o output de um número float, quanto maior, maior a chance de mortalidade.

```
# this linear regressor was used to output float results - the biggest output will
# be the higher death probability


model = LinearRegression()
model.fit(X_train_full, y_train_full)

print("GENERAL PREDICTIONS: ")
print(model.predict(X_test))
print(f'The avarege prediction on general cases is {np.average(model.predict(X_test))}')
print()

df = df[df['MORREU'] == 1]
data = df.drop(['MORREU'], axis=1)
print("PREDICTIONS FOR DEATH CASES")
print(model.predict(data))
print(f'The avarege prediction on the cases where death happened is {np.average(model.predict(data))}')
print()
```

4. Mineração de Dados - Regressão

```
GENERAL PREDICTIONS:
[ 0.39550781  0.00048828  0.04553223 ... -0.00073242 -0.00146484
  0.33215332]
The avarege prediction on general cases is 0.020655678594992427

PREDICTIONS FOR DEATH CASES
[0.39196777 0.3861084  0.39404297 ... 0.31604004 0.29589844 0.28466797]
The avarege prediction on the cases where death happened is 0.34309796055656416
```

Obrigado pela atenção!