Processamento de Linguagens e Compiladores (3º Ano)

# **Trabalho Prático 1**

Relatório de Desenvolvimento


Bruno Fernandes
(a95972)

Nelson Almeida
(a95652)

Nuno Costa
(a97610)


01/11/2022

# Conteúdo

# Capítulo 1

# Introdução

No âmbito da disciplina de Processamentos de Linguagens e Compiladores foi-nos proposto pelo docente Pedro Rangel Henriques um trabalho de pesquisa cujo objetivo principal é consolidar a aprendizagem da utilização de linguagens regulares em programas em linguagem Python.

Decidimos então resolver os exercícios 1 e 2.
O exercício 1 tem como intuito a construção de vários programas em Python para processar o ficheiro de texto "processos.txt"com a intenção de calcular as frequências de alguns elemetos pedidos ao longo do exercício.
O exercício 2 tem como objetivo a construção de vários programas para processar o dataset "emd.csv"e produzir o que é solicitado durante o exercício.

Neste documento apresentamos a nossa resolução para cada um dos problemas.

## Estrutura do Relatório

O relatório está organizado da seguinte forma:
Começamos por fazer uma pequena introdução, capítulo 1, onde referimos o objetivo deste trabalho e explicamos resumidamente o intuito de cada problema que decidimos resolver.
No capítulo 2 aprensentamos o enunciado dos problemas escolhidos.
O capítulo 3 contém a resolução do primeiro problema e no 4 a resolução do segundo.
No capítulo 5 mostramos os resultados obtidos.
E, por fim, o ultimo capítulo contém a conclusão do trabalho realizado.

# Capítulo 2

# Análise e Especificações

## 2.1 Processador de Pessoas listadas nos Róis de Confessados

Construa agora um ou vários programas Python para processar o texto 'processos.txt' com o intuito de calcular frequências de alguns elementos (a ideia é utilizar arrays associativos para o efeito) conforme solicitado a seguir:

a) Calcula a frequência de processos por ano (primeiro elemento da data);

b) Calcula a frequência de nomes próprios (o primeiro em cada nome) e apelidos (o ultimo em cada nome) por séculos;

c) Calcula a frequência dos vários tipos de relação: irmão, sobrinho, etc.

d) Imprimir os 20 primeiros registos num novo cheiro de output mas em formato Json.

## 2.2 Processador de Registos de Exames Médicos Desportivos

Neste exercício pretende-se trabalhar com um dataset gerado no âmbito do registo de exames médicos desportivos. Construa, então, um ou vários programas Python para processar o dataset "emd.csv"e produzir o solicitado nas alíneas seguintes:

1. Página principal: de nome "index.html", contendo os seguintes indicadores estatísticos:
   a) Datas extremas dos registos no dataset;
   b) Distribuição por modalidade em cada ano e no total;
   c) Distribuição por idade e género (para a idade, considera apenas 2 escalões: ¡ 35 anos e ¿= 35);
   d) Distribuição por morada;
   e) Percentagem de aptos e não aptos por ano.
2. Página do indicador: clicando no indicador na página principal, devemos saltar para a página do indicador onde temos a informação que permitiu obter esse indicador. Por exemplo, para a distribuição por morada, a página deverá apresentar uma lista de moradas, ordenada alfabeticamente e para cada morada deverá apresentar uma sublista de registos, ordenada alfabeticamente por nome de atleta (com os dados: nome do atleta, modalidade).

# Capítulo 3

# Resolução do Exercício 1

Inicialmente criamos um ficheiro principal "main.py", onde instalamos os packages necessários para a execução do programa pedindo ao utilizador o caminho (path) para o ficheiro que pretende utilizar, tendo como pré-definido o ficheiro "processos.txt". Neste mesmo ficheiro criamos a função "parser", que recebe como parâmetro o ficheiro pretendido, gerando um conjuntos de arrays com o conteúdo do mesmo que será usado posteriormente para cada uma das 4 alíneas.

a) Para resolver a primeira alínea, desenvolvemos o seguinte código para calcular a frequência de processos por ano.

```
1     def __init__(self, processos):
2       print(self.processesPerTimeSpan(processos))
3
4     def processesPerTimeSpan(self, processos):
5       processosPorData = {}
6       while (processos != []):
7         ano = re.match("(([0-9]{1,4})-*)", processos[0][1]).group(2)
8       processosNoAno, processosNoutrosAnos = self.filtraPorAno(processos, ano)
9       processosPorData[ano] = processosNoAno
10      processos = processosNoutrosAnos
11      return processosPorData
12
13    def filtraPorAno(self, processos, yearToFilter):
14      procPerYear = 0
15      processosNoutrosAnos = []
16      for pessoa in processos:
17        flag = 0
18        for elem in pessoa:
19          if re.search(yearToFilter+r"-[0-9]{2}-[0-9]{1,2}", elem):
20            flag = 1
21        if flag == 1:
22          procPerYear += 1
23        else:
24          processosNoutrosAnos.append(pessoa)
25      return (procPerYear, processosNoutrosAnos)
26
```

Começamos por criar uma função "processesPerTimeSpan"para obter um dicionário com a frequência de processos para cada ano.
Para descobrir cada ano fizemos uso da função match com a expressão regular **"(([0-9]{1,4})-*)"** e invocamos a função auxiliar "filtraPorAno".

```
    while (processos != []):
      ano = re.match("(([0-9]{1,4})-*)", processos[0][1]).group(2)
      processosNoAno, processosNoutrosAnos = self.filtraPorAno(processos, ano)

```

A função auxiliar "filtraPorAno"tem como resultado o número de processos do ano passado como argumento e uma lista contendo todos os processos restantes existentes no ficheiro.
Assim sendo, para cada processo utilizamos a função search com a expressão regular **"-[0-9]{2}-[0-9]{1,2}"** para encontrar todas as datas desse ano passado com argumento.

```
    for pessoa in processos:
      flag = 0
      for elem in pessoa:
        if re.search(yearToFilter+r"-[0-9]{2}-[0-9]{1,2}", elem):
          flag = 1
      if flag == 1:
        procPerYear += 1
      else:
        processosNoutrosAnos.append(pessoa)

```

b) Para a resolução desta alínea, desenvolvemos o seguinte código de modo a obter a frequência de nomes próprios e apelidos por século.

```
    def __init__(self, processos):
      print(self.nomesPorSeculo(processos))

    def nomesPorSeculo(self, processos):
      dic = {"firstNames": {}, "lastNames": {}}
      for processo in processos:
        ano = re.match("(([0-9]{1,4})-*)", processo[1]).group(2)
        sec = int(ano)//100 + 1
        if sec not in dic["firstNames"]:
          dic["firstNames"][sec] = {}
          dic["lastNames"][sec] = {}
        i = 2
        while (i <= 4):
          names = re.split(" ", processo[i])
          fstName = names[0]
          lastName = names[-1]
          if fstName not in dic["firstNames"][sec]:
            dic["firstNames"][sec][fstName] = 0
          if lastName not in dic["lastNames"][sec]:
            dic["lastNames"][sec][lastName] = 0
          dic["firstNames"][sec][fstName] += 1
          dic["lastNames"][sec][lastName] += 1
          i += 1
      return dic

```

Começamos por criar uma função "nomesPorSeculo"com o objetivo de obter um dicionário com duas chaves, uma para os nomes próprios e a outra para os apelidos.
Para tal começamos por encontrar o ano em cada processo usando a função match e a expressão regular **"(([0-9]{1,4})-*)"** e deste modo sabemos a que século pertence esse ano adicionando-o,caso nao exista, ao dicionário.

```
      for processo in processos:
        ano = re.match("(([0-9]{1,4})-*)", processo[1]).group(2)
        sec = int(ano)//100 + 1
        if sec not in dic["firstNames"]:
          dic["firstNames"][sec] = {}
          dic["lastNames"][sec] = {}

```

Para obter o nome próprio e o apelido recorremos à separação dos nomes de cada pessoa por utilizando a função split, partindo-os nomes pelos espaços existentes entre cada um.
Após isso adicionamos os nomes próprios e apelidos aos respetivos séculos.

```
    names = re.split(" ", processo[i])
    fstName = names[0]
    lastName = names[-1]
    if fstName not in dic["firstNames"][sec]:
      dic["firstNames"][sec][fstName] = 0
    if lastName not in dic["lastNames"][sec]:
      dic["lastNames"][sec][lastName] = 0
    dic["firstNames"][sec][fstName] += 1
    dic["lastNames"][sec][lastName] += 1

```

c) Para a resolução deste problema, desenvolvemos o código que se segue e calcula a frequência dos vários tipos de relação.

```
    def __init__(self, processos):
      print(self.tiposParentesco(processos))

    def tiposParentesco(self, pessoas):
      tipos = {}
      for pessoa in pessoas:
        for elem in pessoa:
          parent = re.findall(r'(?<=[a-z]\,)([PMTISANB][A-Za-z ]+|Filho|Filhos)(?=\.)',
     elem)
          for x in parent:
            if x in tipos.keys():
              tipos[x] = tipos.get(x, 0) + 1
            else:
              tipos[x] = 1

```

Começamos por criar uma função "tiposParentesco"que tem como retorno um dicionário que a cada tipo de parentesco associa o número de vezes que o mesmo aparece no ficheiro "processos.txt".
Primeiramente começamos por identificar cada grau de parentesco encontrado no ficheiro. E visto que podem existir mais do que um por cada pessoa, recorremos à função findall com a expressão regular **"(?<=[a-z]\,)([PMTISANB][A-Za-z ]+|Filho|Filhos)(?=\.)**".

```
    tipos = {}
    for pessoa in pessoas:
      for elem in pessoa:
        parent = re.findall(r'(?<=[a-z]\,)([PMTISANB][A-Za-z ]+|Filho|Filhos)(?=\.)',
    elem)

```

d) Para a resolução da última alínea, pretendiamos imprimir os 20 primeiros registos num novo ficheiro de output mas em formato Json e para tal desenvolvemos o código abaixo.

```
    def __init__(self, processos, limit):
      if (self.jsonFormatter(processos[0:limit])):
        print("Ficheiro criado com sucesso")
      else:
        print("Ocorreu um erro a criar o ficheiro")

    def jsonFormatter(self, pessoas):
      file = open("./TP1/ex1/rois.json", "w")
      file.write("{\"data\":[")
      templateJson = '''{"folder_id":#,"date":#,"name":#,"father":#,"mother":#,"obs":#}
    '''
      pos = 0
      for pessoa in pessoas:
        pessoa.append("")
        person = templateJson
        for i in pessoa:
          person = re.sub("#", f"\"{i}\"", person, count=1)
        if (pos < len(pessoas)-1):
          file.write(person+",\n")
        else:
          file.write(person)
        pos += 1
      file.write("]}")
      file.close()
      return True

```

Inicialmente criamos a função "jsonFormatter"que irá retornar o novo output em formato Json.
Esta função começa por escrever de acordo com o template Json no ficheiro criado por nós "rois.json"e retornar em caso de sucesso "Ficheiro criado com sucesso".

# Capítulo 4

# Resolução do Exercício 2

Começamos por criar um ficheiro principal "main.py"onde se começa instala os packages necessários para a execução do programa e pede ao utilizador o path para o ficheiro que pretende utilizar, tendo como pré-definido o ficheiro "emd.csv". Com a função "parser"recebendo como parâmetro o ficheiro pretendido, obtemos um conjuntos de arrays com o conteúdo do mesmo que é usado posteriormente para cada uma das 5 alíneas.

1. Para exercício 1 criamos um ficheiro "index.html"que faz referência às restantes alíneas.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ex2 TP1 PLC</title>
  <link rel="stylesheet" href="main.css">
</head>
<body>
  <div className="mainBar" style="padding-top: 50vh;">
    <a href="a.html">A</a>
    <a href="b.html">B</a>
    <a href="c.html">C</a>
    <a href="d.html">D</a>
    <a href="e.html">E</a>
  </div>
</body>
</html>

```

Para as alíneas b), c), d), e) e f) utilizamos a biblioteca "pandas"para fazer a função "htmlGenerator"para através de um dataframe gerar a representação da tabela em html, esta é guardada numa variável para depois ser processada.
Utilizamos também a biblioteca "matplotlib"para podermos gerar gráficos.
Para adicionar o título da pagina, os gráficos gerados e o "inline styling"ao codigo HTML gerado, utilizamos o re.sub que faz a substituição da tag table gerada por o que pretendemos adicionar.

a) Para resolver a primeira alínea, criamos um array com todas as datas do ficheiro selecionado e determinamos o mínimo e o máximo deste.

```
    def a(self, file):
      datas = [i[2] for i in file]
      file = open("./TP1/ex2/1/website/a.html", "w")
      templateText =
        '''<!DOCTYPE html >
        <html lang = "en" >
        <head >
          <meta charset = "UTF-8" >
          <meta http-equiv = "X-UA-Compatible"  content = "IE=edge" >
          <meta name = "viewport" content = "width=device-width, initial-scale
    =1.0" >
          <title >a</title >
          <link rel="stylesheet" href="./main.css">
        </head >
        <body >
        </body >
        </html >'''
      newText = re.sub(
        "<body >", f'''\n<body >\n<p class="title">Datas extremas dos registos no
    dataset</p>\n<p class="minData">Extremo inferior do dataset: {min(datas)}</p>\n
    <p class="maxData">Extremo superior do dataset: {max(datas)}</p>''',
    templateText)
      file.write(newText)
      file.close()

```

b) Para a resolução da alínea b) e assim determinar a distribuição por modalidade em cada ano e no total, desenvolvemos o seguinte código.

```
    def b(self, data):
      sportsPerYear, totalSports = self.sportsPerYearandTotal(data)
      indexes = sorted(sportsPerYear)
      sportsPerYear = {i: sportsPerYear[i] for i in indexes}
      totalSports = dict(sorted(totalSports.items(), key=lambda x: x[1]))
      self.plotter(sportsPerYear, totalSports)
      self.htmlGenerator(sportsPerYear, totalSports)

    def sportsPerYearandTotal(self, data):
      sportsPerYear = {}
      totalSports = {}
      while (data):
        year = re.match("(([0-9]{1,4})-*)", data[0][2]).group(2)
        sportsYear, updatedData = self.sportsPerYear(data, year)
        sportsPerYear[year] = sportsYear
        data = updatedData

      for year in sportsPerYear:
        for sport in sportsPerYear[year]:
          if sport not in totalSports:
            totalSports[sport] = 0
          totalSports[sport] += sportsPerYear[year][sport]
      return sportsPerYear, totalSports

    def sportsPerYear(self, data, yearToFilter):
      sportsPerY = {}
      newData = []
      for person in data:
        if re.match(yearToFilter+r"(-[0-9]{1,2}){2}", person[2]):
          sport = person[8]
          if sport not in sportsPerY:
            sportsPerY[sport] = 0
          sportsPerY[sport] += 1
        else:
          newData.append(person)
      return sportsPerY, newData

    def htmlGenerator(self, data1, data2):
      file = open(f"./TP1/ex2/1/website/b.html", "w")
      templateText1 = pd.DataFrame(data1).to_html()
      templateText1 = re.sub(
        r'''<table border="1" class="dataframe">''', '''<h1 class="title">
Distribui  o por modalidade em cada ano e no total</h1>\n<div class="images"
style="display: flex; width: 100%; padding-bottom: 2rem;"><img src="./src/b1.
png" alt="" style="width:50%">\n<img src="./src/b2.png" alt="" style="width
:50%">\n</div>\n<link rel="stylesheet" href="./main.css">\n<table border="1"
class="dataframe">''', templateText1
      )
      file.write(templateText1)
      templateText2 = pd.DataFrame(data2, index=["Praticantes"]).to_html()
      file.write(templateText2)
      file.close()

    def plotter(self, data1, data2):
      df1 = pd.DataFrame(data1)
```

```
51          df1.plot(kind="bar")
52          plt.savefig("./TP1/ex2/1/website/src/b1.png")
53          df2 = pd.DataFrame(data2, index=["Praticantes"])
54          df2.plot(kind="bar")
55          plt.savefig("./TP1/ex2/1/website/src/b2.png")
56
```

Começamos por criar a função auxiliar "sportsPerYearandTotal"para obtermos dois dicionários, um com os desportos distribuídos por anos e outro com todos os desportos existentes no ficheiro. Para cada pessoa existente no ficheiro utilizamos o match com a expressão regular **"(([0-9]1,4)-*)"** para encontrar o ano que vai ser utilizado como parâmetro em outra função auxiliar "sportsPerYear".

```
1       year = re.match("(([0-9]{1,4})-*)", data[0][2]).group(2)
2       sportsYear, updatedData = self.sportsPerYear(data, year)
3
```

Na função "sportsPerYear"começamos por criar um dicionário com os desportos existentes nesse ano e a sua frequência.

```
1       if re.match(yearToFilter+r"(-[0-9]{1,2}){2}", person[2]):
2         sport = person[8]
3         if sport not in sportsPerY:
4           sportsPerY[sport] = 0
5         sportsPerY[sport] += 1
6
```

E um array onde vão ser adicionados apenas os dados relativos aos restantes anos (diferentes do ano passado como parâmetro), para assim, evitar repetições e garantir um melhor controlo.

```
1       else:
2         newData.append(person)
3
```

No fim vão ser retornados o dicionário "sportsPerY"e o array "newData"que são utilizados para atualizar o dicionário "sportsPerYear"e os dados do array "data"do método anterior.

```
1       sportsPerYear[year] = sportsYear
2       data = updatedData
3
```

Por último, com os dados colecionados no dicionário "sportsPerYear", soma o número de desportistas para cada ano e para cada desporto e coleciona no dicionário de "totalSports".

```
1       for year in sportsPerYear:
2         for sport in sportsPerYear[year]:
3           if sport not in totalSports:
4             totalSports[sport] = 0
5           totalSports[sport] += sportsPerYear[year][sport]
6
```

c) Para a resolução da alínea c) e assim determinar a distribuição por idade e género, desenvolvemos o seguinte código.

```
    def c(self, processos):
      data = processos[1:]  # aqui ficam tudo menos os indicadores
      data = self.distByAgeAndGender(data)
      self.plotter(data)
      self.htmlGenerator(data)

    def distByAgeAndGender(self, data):
      dist = {"<35": {}, ">=35": {}}
      ageIndex = 5
      genderIndex = 6
      for person in data:
        ageRange = ">=35"
        if int(person[ageIndex]) < 35:
          ageRange = "<35"
        if person[genderIndex] not in dist[ageRange]:
          dist[ageRange][person[genderIndex]] = 0
        dist[ageRange][person[genderIndex]] += 1
    return dist

    def htmlGenerator(self, data1):
      file = open(f"./TP1/ex2/1/website/c.html", "w")
      templateText = pd.DataFrame(data1).to_html()
      templateText = re.sub(
        r'''<table border="1" class="dataframe">''', '''<p class="title">
Distribui  o por idade e g nero</p>\n<div class="images" style="padding-
bottom: 2rem;"><img src="./src/c.png" alt="">\n</div>\n<link rel="stylesheet"
href="./main.css">\n<table border="1" class="dataframe">''', templateText)
      file.write(templateText)
      file.close()

    def plotter(self, data):
      df1 = pd.DataFrame(data)
      df1.plot(kind="bar")
      plt.savefig("./TP1/ex2/1/website/src/c.png")

```

Criamos a função auxiliar "distByAgeAndGender"que começa por criar um dicionário onde vão ser guardadas as informações, vai percorrer cada pessoa do ficheiro, verificar se a sua idade é maior ou menor que 35 anos, verificar o género dessa pessoa e incrementar na contação.

d) Para a resolução da alínea d) e assim determinar a distribuição por morada, desenvolvemos o seguinte código.

```
    def d(self, processos):
      data = processos[1:]
      data = self.distByAddress(data)
      self.plotter(data)
      self.htmlGenerator(data)

    def distByAddress(self, data):
      distByAddress = {}
      for person in data:
        address = person[7]
        if address not in distByAddress:
          distByAddress[address] = 0
          distByAddress[address] += 1
      return distByAddress

    def htmlGenerator(self, data1):
      file = open(f"./TP1/ex2/1/website/d.html", "w")
      templateText = pd.DataFrame(data1, index=[""]).to_html()
      templateText = re.sub(
        r'''<table border="1" class="dataframe">''', '''<p class="title">
Distribui  o por morada</p>\n<div class="images" style="padding-bottom: 2rem
;"><img src="./src/d.png" alt="">\n</div>\n<link rel="stylesheet" href="./main.
css">\n<table border="1" class="dataframe">''', templateText
      )
      file.write(templateText)
      file.close()

    def plotter(self, data):
      if len(data) >= 50:
        df1 = pd.DataFrame(data, index=[""])
        df1.plot(kind="bar")
        plt.savefig("./TP1/ex2/1/website/src/d.png")
      else:
        df1 = pd.DataFrame(data, index=[""])
        df1.plot(kind="bar")
        plt.savefig("./TP1/ex2/1/website/src/d.png")

```

Criamos a função auxiliar "distByAddress"que começa por criar um dicionário onde vão ser guardadas as informaçõe, vai percorrer cada pessoa do ficheiro, adicionar ao dicionário a sua morada caso esta não exista e incrementar o número de pessoas que vivem nessa morada.

e) Para a resolução da alínea e) e assim determinar a percentagem de pessoas aptas e não aptas por ano, desenvolvemos o seguinte código.

```
    def e(self, data):
      data = data[1:]
      readinessStats = {}
      while (data):
        year = re.match("(([0-9]{1,4})-*)", data[0][2]).group(2)
        (ready, total), newData = self.readinessPerYear(data, year)
        readinessPerc = (ready/total)
        readinessStats[year] = {
          "ready": float("{:.2f}".format(100*readinessPerc)),
          "not_ready": float("{:.2f}".format(100*(1-readinessPerc)))
        }
        data = newData
      self.plotter(readinessStats)
      self.htmlGenerator(readinessStats)

    def readinessPerYear(self, data, year):
      ready = 0
      total = 0
      newData = []
      for person in data:
        if re.match(year+r"(-[0-9]{1,2}){2}", person[2]):
          if person[-1] == "true":
            ready += 1
          total += 1
        else:
          newData.append(person)
      return (ready, total), newData

    def htmlGenerator(self, data1):
      file = open(f"./TP1/ex2/1/website/e.html", "w")
      templateText = pd.DataFrame(data1).to_html()
      templateText = re.sub(
        r'''<table border="1" class="dataframe">''', '''<p class="title">
Percentagem de aptos e n o aptos por ano</p>\n<div class="images" style="
padding-bottom: 2rem;"><img src="./src/e.png" alt="">\n</div>\n<link rel="
stylesheet" href="main.css">\n<table border="1" class="dataframe">''',
templateText
    )
      file.write(templateText)
      file.close()

    def plotter(self, data):
      df1 = pd.DataFrame(data)
      df1.plot(kind="bar")
      plt.savefig("./TP1/ex2/1/website/src/e.png")

```

Começamos por percorrer cada pessoa do ficheiro e utilizando o match com a expressão regular **"(([0-9]1,4)-*)"** encontrar o ano que vai ser utilizado como parâmetro na função auxiliar "readinessPerYear".

```
    year = re.match("(([0-9]{1,4})-*)", data[0][2]).group(2)
    (ready, total), newData = self.readinessPerYear(data, year)

```

Esta função "readinessPerYear"começa por criar duas variáveis "ready"e "total"que vão servir para saber o número de pessoas aptas e o total de pessoas, respetivamente, e um array onde

vão ser adicionados apenas os dados relativos aos restantes anos (diferentes do ano passado como parâmetro), para assim, evitar repetições e garantir um melhor controlo.
Começamos por percorrer cada pessoa, verificar se o ano coincide com ano passado como parâmetro, incrementar na variavél "ready"se tal se verificar e essa pessoa estiver apta.

```
      if re.match(year+r"(-[0-9]{1,2}){2}", person[2]):
        if person[-1] == "true":
          ready += 1
        total += 1

```

Na função principal, depois de receber o número de pessoas aptas e no total nesse ano, são calculadas as porcentagens de atletas aptos e não aptos e colocadas no dicionário. Posteriormente os dados são atualizados.

```
      readinessPerc = (ready/total)
      readinessStats[year] = {
        "ready": float("{:.2f}".format(100*readinessPerc)),
        "not_ready": float("{:.2f}".format(100*(1-readinessPerc)))
      }
      data = newData

```

# Capítulo 5

# Testes e Resultados obtidos

## 5.1 Problema 1

### 5.1.1 Frequência de Processos Por Ano


```
{'1894': 74, '1909': 39, '1867': 54, '1896': 75, '1904': 52, '1901': 59, '1883': 34, '1900': 50, '1902': 83, '1880': 62, '1889': 71, '1908': 51, '1869': 39,
'1862': 40, '1906': 63, '1856': 72, '1875': 15, '1892': 57, '1733': 1188, '1778': 1066, '1691': 1084, '1730': 1200, '1899': 80, '1898': 91, '1877': 47,
'1910': 27, '1881': 64, '1907': 47, '1884': 44, '1879': 55, '1895': 78, '1897': 71, '1707': 116, '1689': 628, '1713': 266, '1824': 251, '1703': 143, '1720': 172,
'1890': 44, '1732': 2217, '1683': 142, '1863': 30, '1729': 47, '1694': 49, '1765': 9, '1754': 303, '1690': 278, '1755': 303, '1823': 213, '1708': 150, '1757': 32,
'1699': 97, '1759': 90, '1712': 70, '1687': 97, '1738': 213, '1717': 267, '1684': 238, '1704': 358, '1688': 110, '1888': 70, '1734': 888, '1786': 352, '1798': 56,
'1773': 482, '1821': 312, '1822': 303, '1809': 292, '1722': 457, '1680': 158, '1695': 44, '1728': 440, '1716': 238, '1849': 120, '1777': 1142, '1851': 66,
'1785': 797, '1857': 80, '1686': 189, '1784': 361, '1780': 218, '1727': 169, '1788': 416, '1719': 387, '1847': 130, '1799': 103, '1829': 162, '1787': 743,
'1805': 109, '1819': 242, '1844': 196, '1891': 73, '1731': 900, '1760': 280, '1741': 32, '1725': 272, '1802': 90, '1827': 97, '1885': 8, '1807': 602, '1710': 199,
'1692': 246, '1706': 117, '1858': 46, '1739': 74, '1826': 171, '1714': 317, '1762': 342, '1743': 53, '1724': 110, '1697': 28, '1852': 74, '1740': 13, '1855': 86,
'1723': 129, '1859': 87, '1811': 237, '1817': 207, '1685': 235, '1905': 42, '1893': 78, '1865': 40, '1848': 125, '1911': 16, '1882': 50, '1735': 153, '1871': 35,
'1903': 20, '1850': 115, '1825': 155, '1843': 296, '1860': 77, '1812': 259, '1846': 86, '1845': 127, '1701': 108, '1746': 49, '1868': 22, '1715': 90, '1803': 167,
'1830': 174, '1761': 381, '1766': 9, '1672': 17, '1876': 40, '1698': 153, '1726': 85, '1679': 83, '1750': 53, '1711': 75, '1810': 47, '1756': 75, '1721': 135,
'1873': 46, '1831': 193, '1781': 208, '1818': 229, '1816': 352, '1783': 309, '1752': 80, '1866': 33, '1747': 58, '1771': 26, '1700': 82, '1682': 106, '1794': 57,
'1742': 22, '1887': 42, '1836': 3, '1878': 76, '1886': 46, '1702': 139, '1767': 15, '1709': 148, '1779': 198, '1751': 107, '1681': 101, '1748': 46, '1678': 12,
'1839': 85, '1770': 27, '1673': 19, '1792': 46, '1796': 86, '1828': 172, '1736': 47, '1737': 65, '1696': 42, '1853': 90, '1772': 122, '1718': 92, '1776': 36,
'1804': 123, '1774': 38, '1832': 301, '1806': 106, '1820': 253, '1808': 311, '1854': 112, '1861': 62, '1833': 194, '1841': 47, '1705': 76, '1872': 42, '1874': 46,
'1782': 155, '1693': 28, '1838': 10, '1797': 46, '1660': 1, '1864': 56, '1870': 12, '1744': 41, '1800': 107, '1655': 2, '1795': 55, '1840': 38, '1764': 24,
'1763': 29, '1676': 9, '1768': 27, '1753': 37, '1842': 36, '1671': 14, '1801': 58, '1814': 11, '1815': 45, '1758': 22, '1749': 27, '1775': 30, '1745': 47,
'1834': 13, '1769': 16, '1793': 45, '1665': 2, '1813': 12, '1635': 1, '1616': 1, '1791': 6, '1631': 2, '1658': 2, '1664': 5, '1789': 12, '1623': 3, '1675': 9,
'1677': 6, '1666': 6, '1663': 3, '1674': 3, '1668': 5, '1661': 1, '1630': 4, '1628': 7, '1670': 3, '1633': 2, '1632': 3, '1669': 3, '1636': 1, '1667': 1,
'1634': 1, '1625': 2, '1656': 2, '1620': 1, '1629': 1, '1622': 1, '1627': 2, '1790': 1, '1650': 1, '1662': 1}
```


### 5.1.2 Frequência de nomes próprios e apelidos por séculos


```
{'firstNames': {19: {'Aarao': 1, 'Antonio': 3618, 'Francisca': 311, 'Abel': 3, 'Maria': 3695, 'Francisco': 1779, 'Antonia': 428, 'Joao': 2626, 'Cecilia': 5,
'Abelardo': 2, 'Jose': 3810, 'Leopoldina': 9, 'Abilio': 8, 'Sebastiao': 91, 'Teresa': 509, 'Ana': 1270, 'Flora': 4, 'Acacio': 3, 'Albina': 18, 'Adelino': 13,
'Custodia': 333, 'Bernardo': 203, 'Rosa': 660, 'Manuel': 3469, 'Rosalia': 25, 'Joana': 328, 'Adolfo': 3, 'Carlota': 18, 'Custodio': 241, 'Emilia': 29,
'Adriano': 9, 'Miguel': 157, 'Adriao': 1, 'Bernardina': 29, 'Afonso': 6, '': 171, 'Agostinho': 71, 'Luisa': 345, 'Domingos': 918, 'Helena': 40, 'Benta': 53,
'Margarida': 65, 'Mateus': 19, 'Gregorio': 17, 'Jeronima': 40, 'Joaquina': 249, 'Cleto': 6, 'Bernardino': 76, 'Aires': 2, 'Luis': 478, 'Albano': 6, 'Albertino':
1, 'Alberto': 14, 'Florinda': 23, 'Gracinda': 5, 'Benedita': 2, 'Henrique': 38, 'Albino': 26, 'Engracia': 29, 'Mariana': 233, 'Violante': 22, 'Cipriana': 7,
'Inacia': 38, 'Josefa': 318, 'Alexandre': 99, 'Marinha': 8, 'Perpetua': 20, 'Caetana': 39, 'Teodora': 8, 'Feliciana': 10, 'Bento': 353, 'Isabel': 202,
'Senhorinha': 49, 'Catarina': 75, 'Martina': 1, 'Lucia': 1, 'Alexandrino': 3, 'Vicente': 38, 'Clarina': 1, 'Caetano': 114, 'Alfredo': 15, 'Apolonia': 2,
'Lidorio': 1, 'Alvaro': 21, 'Vitorino': 25, 'Julio': 26, 'Rita': 82, 'Lourenco': 56, 'Amador': 1, 'Amaro': 6, 'Gaspar': 68, 'Ambrosio': 7, 'Americo': 1,
'Lino': 16, 'Anacleto': 10, 'Estevao': 20, 'Ana,Exposta': 2, 'Anastacio': 10, 'Narciso': 36, 'Andre': 27, 'Joaquim': 768, 'Inacio': 73, 'Angelo': 4, 'Anibal': 3,
'Ludovina': 9, 'Aniceto': 3, 'Anselmo': 6, 'Eugenia': 15, 'Escolastica': 3, 'Antao': 2, 'Andresa': 5, 'Antonino': 2, 'Rosendo': 3, 'Angelica': 35, 'Simao': 42,
'Balbina': 9, 'Pedro': 156, 'Jeronimo': 95, 'Genoveva': 25, 'Domingas': 46, 'Ventura': 7, 'Felizarda': 7, 'Vitoria': 33, 'Claudina': 10, 'Brigida': 6,...},...},


'lastNames': {19: {'Silva': 1318, 'Barroso': 138, 'Oliveira': 456, 'Rebelo': 121, 'Freitas': 194, 'Pereira': 1197, 'Araujo': 719, 'Guerreiro': 47, 'Monteiro':
164, 'Alves': 245, 'Rocha': 235, 'Arantes': 28, 'Sousa': 804, 'Santos': 244, 'Jesus': 156, 'Leite': 199, 'Fernandes': 626, 'Guimaraes': 161, 'Borges': 94,
'Teixeira': 443, 'Carneiro': 142, 'Barbosa': 357, 'Matos': 160, 'Coelho': 201, 'Dias': 356, 'Ferreira': 592, 'Peixoto': 155, 'Costa': 995, 'Cardoso': 143,
'Eiras': 10, 'Goncalves': 789, 'Afonso': 179, 'Almeida': 256, 'Campos': 123, 'Pimenta': 65, 'Miranda': 135, 'Pedrosa': 16, 'Balazeiro': 1, 'Barros': 309, 'Abreu':
253, 'Vieira': 266, 'Meireles': 65, 'Moura': 161, 'Segura': 4, 'Vaz': 128, 'Casimira': 2, 'Pinheiro': 126, 'Freire': 20, 'Catalao': 2, '': 171, 'Sanfins': 1,
'Soares': 218, 'Penteado': 5, 'Brito': 173, 'Azevedo': 320, 'Reis': 119, 'Mendes': 113, 'Antunes': 158, 'Nogueira': 76, 'Botelho': 53, 'Vale': 99, 'Ribeiro': 456,
'Sotomaior': 48, 'Felizarda': 3, 'Rodrigues': 741, 'Alvares': 381, 'Cunha': 519, 'Domingues': 178, 'Teresa': 161, 'Passos': 37, 'Macedo': 180, 'Meneses': 138,
'Queiros': 80, 'Carvalho': 667, 'Lopes': 345, 'Castro': 382, 'Martins': 470, 'Marques': 152, 'Coimbra': 12, 'Gomes': 495, 'Flores': 16, 'Conceicao': 70, 'Fontes':
20, 'Basto': 39, 'Salgado': 52, 'Pedreira': 10, 'Lobo': 126, 'Chaves': 66, 'Sepulveda': 18, 'Machado': 360, 'Pinto': 267, 'Vasconcelos': 143, 'Junior': 84,
'Brandao': 61, 'Josefa': 84, 'Correia': 203, 'Faria': 214, 'Joaquina': 201, 'Novais': 57, 'Assuncao,Solteira': 2, 'Carreira,Viuva': 1, 'Braganca': 11, 'Coutinho':
130, 'Moreira': 115, 'Maria': 412, 'Dantas': 71, 'Pequeno': 5, 'Pimentel': 35, 'Homem': 1, 'Camelo': 31, 'Lima': 340, 'Joao': 23, 'Pires': 192}...}}
```

### 5.1.3 Frequência dos vários tipos de relação


```
{'Tio Paterno': 2245, 'Tio Materno': 2463, 'Irmao': 13168, 'Primo Paterno': 205, 'Sobrinho Materno': 1698, 'Pai': 525, 'Filho': 346, 'Sobrinho Paterno': 1642,
'Irmaos': 686, 'Sobrinhos Maternos': 98, 'Irmao Paterno': 497, 'Neto Materno': 41, 'Sobrinhos Paternos': 57, 'Sobrinho Neto Paterno': 97, 'Primo': 673,
'Primo Materno': 283, 'Tio Avo Paterno': 154, 'Tio Avo Materno': 230, 'Irmao Materno': 55, 'Sobrinho Bisneto Paterno': 3, 'Tios Maternos': 20,
'Irmaos Paternos': 21, 'Sobrinho Neto Materno': 145, 'Avo Materno': 48, 'Bisavo Materno': 2, 'Filhos': 27, 'Avo Paterno': 11, 'Neto Paterno': 8,
'Tios Paternos': 12, 'Tio Bisavo Materno': 5, 'Primos': 13, 'Parente': 4, 'Primos Paternos': 1, 'Tio Bisavo Paterno': 6, 'Irmaos Maternos': 4,
'Sobrinhos Netos Paternos': 2, 'Sobrinho Neto': 2, 'Tio Avo': 3, 'Tio': 5, 'Sobrinhos Netos Maternos': 5, 'Meio Irmao': 3, 'Sobrinho Bisneto Materno': 3,
'Primos Maternos': 1}
```


### 5.1.4 Imprimir os 20 primeiros registos em formatos Json


```
{
  "data": [
  {
    "folder_id": "575",
    "date": "1894-11-08",
    "name": "Aarao Pereira Silva",
    "father": "Antonio Pereira Silva",
    "mother": "Francisca Campos Silva",
    "obs": ""
  },
  {
    "folder_id": "582",
    "date": "1909-05-12",
    "name": "Abel Almeida",
    "father": "Antonio Manuel Almeida",
    "mother": "Teresa Maria Sousa",
    "obs": ""
  },
  {
    "folder_id": "569",
    "date": "1867-05-23",
    "name": "Abel Alves Barroso",
    "father": "Antonio Alves Barroso",
    "mother": "Maria Jose Alvares Barroso",
    "obs": "Bento Alvares Barroso,Tio Paterno. Proc.32057.   Domingos Jose Alvares
  Barroso,Tio Materno. Proc.32235."
  },
  {
    "folder_id": "576",
    "date": "1896-11-28",
    "name": "Abel Augusto Oliveira",
    "father": "Francisco Jose Oliveira",
    "mother": "Antonia Rosa Rebelo",
    "obs": "Jose Antonio Oliveira,Irmao. Proc.5020."
  },
  {
    "folder_id": "579",
    "date": "1904-05-27",
    "name": "Abel Gomes Abreu Reis",
    "father": "Antonio Gomes Abreu",
    "mother": "Ana Sequeira Reis",
    "obs": ""
  },
  {
    "folder_id": "579",
    "date": "1904-05-21",
```

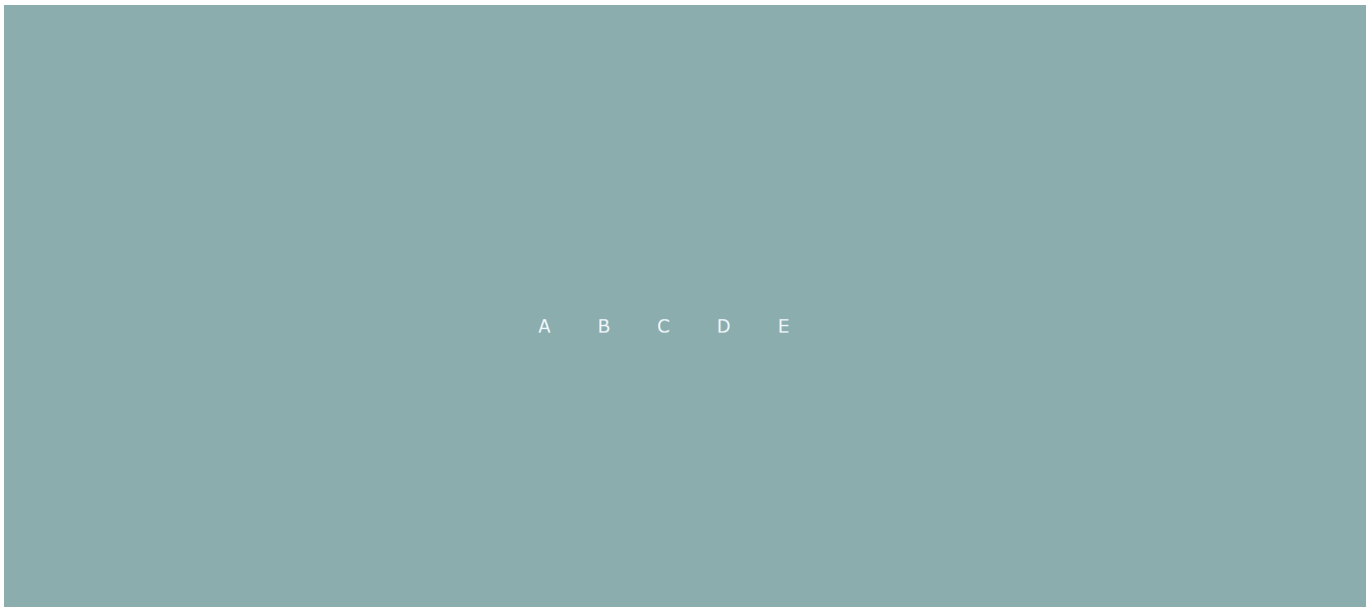
```
      "name": "Abel Marques Reis",
      "father": "Jose Joaquim Marques Reis",
      "mother": "Bernardina Dantas",
      "obs": ""
    },
    {
      "folder_id": "578",
      "date": "1901-11-12",
      "name": "Abel Martins Pereira",
      "father": "Serafim Martins Pereira",
      "mother": "Emilia Goncalves",
      "obs": ""
    },
    {
      "folder_id": "572",
      "date": "1883-02-01",
      "name": "Abel Pedro Pereira Freitas",
      "father": "Joao Freitas Oliveira",
      "mother": "Cecilia Rosa Pereira",
      "obs": ""
    },
    {
      "folder_id": "578",
      "date": "1900-08-30",
      "name": "Abel Silva Carvalho",
      "father": "Constantino Silva Rego",
      "mother": "Margarida Rosa Carvalho",
      "obs": ""
    },
    {
      "folder_id": "575",
      "date": "1894-04-30",
      "name": "Abelardo Jose Cerqueira Araujo",
      "father": "Jose Maria Araujo",
      "mother": "Leopoldina Cerqueira Ribeiro Araujo",
      "obs": ""
    },
    {
      "folder_id": "575",
      "date": "1894-04-30",
      "name": "Abelardo Jose Cerqueira Araujo",
      "father": "Jose Maria Araujo",
      "mother": "Leopoldina Cerqueira Ribeiro Araujo",
      "obs": ""
    },
    {
      "folder_id": "572",
      "date": "1883-11-24",
      "name": "Abilio Acacio Conceicao Guerreiro",
      "father": "Jose Antonio Pereira Dantas Guerreiro",
      "mother": "Maria Rita Pereira Monteiro",
      "obs": ""
    },
    {
      "folder_id": "579",
      "date": "1902-10-23",
      "name": "Abilio Aires Sousa Pereira Guimaraes",
      "father": "Joaquim Aires Sousa Pereira Guimaraes",
      "mother": "Josefa Rosa Gomes",
```

```
      "obs": ""
    },
    {
      "folder_id": "571",
      "date": "1880-01-24",
      "name": "Abilio Antonio Alves",
      "father": "Joao Francisco Alves",
      "mother": "Maria Jesus Rocha",
      "obs": ""
    },
    {
      "folder_id": "573",
      "date": "1889-12-03",
      "name": "Abilio Augusto Arantes",
      "father": "Sebastiao Arantes",
      "mother": "Maria Sousa",
      "obs": ""
    },
    {
      "folder_id": "581",
      "date": "1908-08-11",
      "name": "Abilio Augusto Galvao",
      "father": "Antonio Augusto Galvao",
      "mother": "Perpetua Duarte Galvao",
      "obs": ""
    },
    {
      "folder_id": "581",
      "date": "1908-05-20",
      "name": "Abilio Augusto Magalhaes",
      "father": "",
      "mother": "Maria Jesus Magalhaes",
      "obs": ""
    },
    {
      "folder_id": "581",
      "date": "1908-05-20",
      "name": "Abilio Augusto Magalhaes",
      "father": "",
      "mother": "Maria Jesus Magalhaes",
      "obs": ""
    },
    {
      "folder_id": "581",
      "date": "1908-05-20",
      "name": "Abilio Augusto Magalhaes",
      "father": "",
      "mother": "Maria Jesus Magalhaes",
      "obs": ""
    },
    {
      "folder_id": "569",
      "date": "1869-12-02",
      "name": "Abilio Augusto Santos",
      "father": "Jose Joaquim Santos",
      "mother": "Teresa Jesus",
      "obs": "Antonio Jose Adao,Tio Materno. Proc.12530.   Albino Antonio Ribeiro,Primo
    Paterno. Proc.12721."
    }
```

```
163     ]
164   }
```

## 5.2 Problema 2

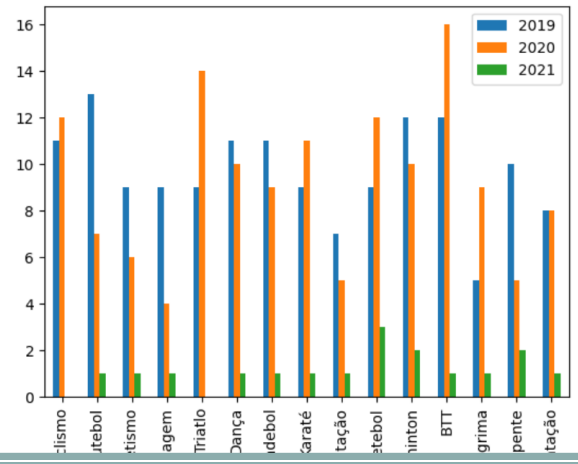
### 5.2.1 index.html

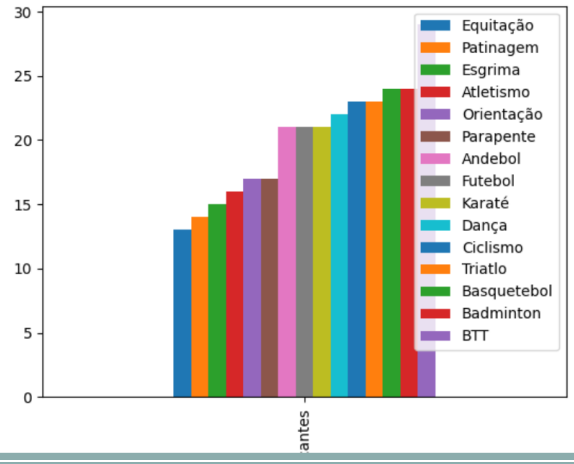



### 5.2.2 Datas extremas dos registos no dataset

### 5.2.3 Distribuição por modalidade em cada ano e no total

**Distribuição por modalidade em cada ano e no total**



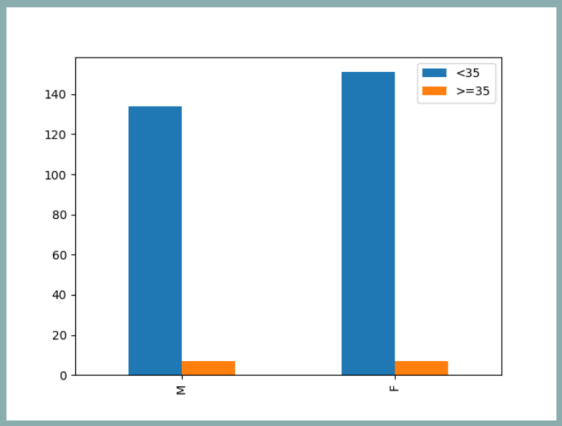




| | 2019 | 2020 | 2021 |
|---|---|---|---|
| **Ciclismo** | 11 | 12 | NaN |
| **Futebol** | 13 | 7 | 1.0 |
| **Atletismo** | 9 | 6 | 1.0 |
| **Patinagem** | 9 | 4 | 1.0 |
| **Triatlo** | 9 | 14 | NaN |
| **Dança** | 11 | 10 | 1.0 |
| **Andebol** | 11 | 9 | 1.0 |
| **Karaté** | 9 | 11 | 1.0 |
| **Equitação** | 7 | 5 | 1.0 |
| **Basquetebol** | 9 | 12 | 3.0 |
| **Badminton** | 12 | 10 | 2.0 |
| **BTT** | 12 | 16 | 1.0 |
| **Esgrima** | 5 | 9 | 1.0 |
| **Parapente** | 10 | 5 | 2.0 |
| **Orientação** | 8 | 8 | 1.0 |

| | Equitação | Patinagem | Esgrima | Atletismo | Orientação | Parapente | Andebol | Futebol | Karaté | Dança | Ciclismo | Triatlo | Basquetebol | Badminton | BTT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Praticantes** | 13 | 14 | 15 | 16 | 17 | 17 | 21 | 21 | 21 | 22 | 23 | 23 | 24 | 24 | 29 |

### 5.2.4 Distribuição por ano e género

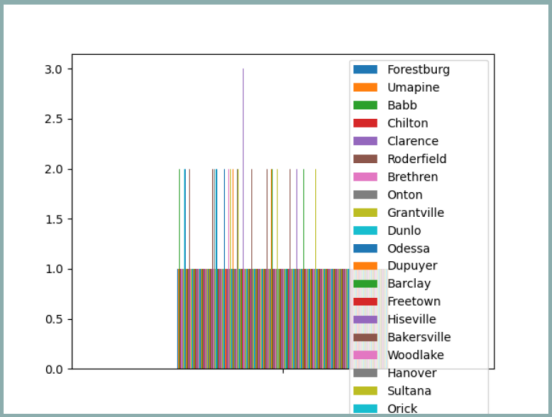
Distribuição por idade e género




| | <35 | >=35 |
|---|---|---|
| M | 134 | 7 |
| F | 151 | 7 |

### 5.2.5 Distribuição por morada

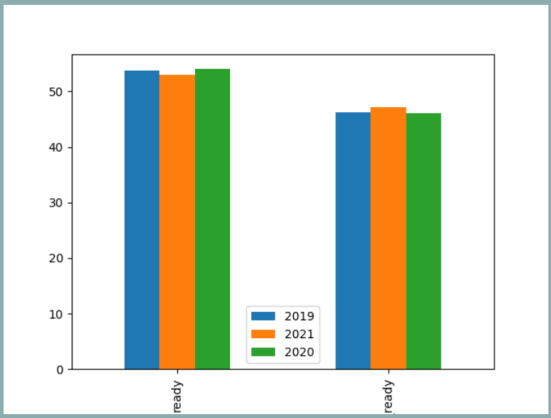
Distribuição por morada




| Forestburg | Umapine | Babb | Chilton | Clarence | Roderfield | Brethren | Onton | Grantville | Dunlo | Odessa | Dupuyer | Barclay | Freetown | Hiseville | Baker |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 2 |

### 5.2.6 Porcentagem de aptos e não aptos por ano

Percentagem de aptos e não aptos por ano




| | 2019 | 2021 | 2020 |
|---|---|---|---|
| ready | 53.79 | 52.94 | 54.01 |
| not_ready | 46.21 | 47.06 | 45.99 |

# Capítulo 6

# Conclusão

Com a realização deste trabalho deparamo-nos com vários problemas de programação e raciocínio encontrados anteriormente e que sem o recurso/conhecimento das expressões regulares levavam soluções muito mais complexas, o que se traduziu portanto numa valiosa adição à nossa bagagem. Numa perspetiva geral julgamos estar a entregar um trabalho que cumpria a maioria das nossa metas para o mesmo, no entanto é de salientar que certos aspetos do exercício 2 ficaram à quem das nossas expectativas, nuances tais que serão melhoradas e corrigidas em trabalhos futuros.

## 6.1 Bibliografia

Pandas Documentation
Matplotlib Documentation