



**Principal Component Analysis – PCA**  
**Aplicação em: “Causas de Mortes em países ao redor do mundo no  
ano de 1990”**

**2º Ciclo – Ciência de Dados**  
**Álgebra Linear**

Juliana Câmara Monteiro  
RA 0051352211021

## 1. Dataset:

O dataset usado para a aplicação do PCA neste trabalho foi retirado do site Kaggle. Trata-se de uma base de dados com causas de mortes em países ao redor do mundo entre os anos de 1990 e 2019. Porém, a análise restringiu-se somente ao ano de 1990. O dataset é composto por 6120 linhas e 34 colunas, sendo que, a partir da quarta, encontram-se os registros de mortos por país (dados brutos). Segue relação de colunas:

- 1.1- Country/Territory
- 1.2- Code
- 1.3- Year
- 1.4- Meningitis
- 1.5- Alzheimer's Disease and Other Dementias
- 1.6- Parkinson's Disease
- 1.7- Nutritional Deficiencies
- 1.8- Malaria
- 1.9- Drowning
- 1.10- Interpersonal Violence
- 1.11- Maternal Disorders
- 1.12- HIV/AIDS
- 1.13- Drug Use Disorders
- 1.14- Tuberculosis
- 1.15- Cardiovascular Diseases
- 1.16- Lower Respiratory Infections
- 1.17- Neonatal Disorders
- 1.18- Alcohol Use Disorders
- 1.19- Self-harm
- 1.20- Exposure to Forces of Nature
- 1.21- Diarrheal Diseases
- 1.22- Environmental Heat and Cold Exposure
- 1.23- Neoplasms
- 1.24- Conflict and Terrorism
- 1.25- Diabetes Mellitus
- 1.26- Chronic Kidney Disease
- 1.27- Poisonings
- 1.28- Protein-Energy Malnutrition
- 1.29- Road Injuries
- 1.30- Chronic Respiratory Diseases
- 1.31- Cirrhosis and Other Chronic Liver Diseases
- 1.32- Digestive Diseases
- 1.33- Fire, Heat, and Hot Substances
- 1.34- Acute Hepatitis

Link: <https://www.kaggle.com/datasets/iamsouravbanerjee/cause-of-deaths-around-the-world>

## 2. Aplicando o PCA

A técnica foi aplicada através da linguagem Python, usando as bibliotecas Pandas, Numpy e Scikit-learn, seguindo as seguintes etapas:

- Carregando o DataFrame - Causas de mortes em países ao redor do mundo dos anos de 1990 à 2019.

```
[1] import pandas as pd
import numpy as np

causas_mortes = pd.read_csv('cause_of_deaths.csv')
causas_mortes = pd.DataFrame(causas_mortes)
print(causas_mortes)
```

Após atribuir os registros do arquivo CSV ao DataFrame, excluiremos os valores repetidos da coluna de classificação dos anos, mantendo assim, somente os dados dos anos de 1990 para cada país. Logo após, verificaremos os tipos de variáveis que compõem o dataset para identificar quais colunas participarão da matriz de correlação

```
Removendo valores duplicados da coluna de "Country/Territory". Serão contabilizados somente os registros dos anos de 1990 para cada país.

[ ] causas_mortes = causas_mortes.drop_duplicates(subset= 'Country/Territory')
print(causas_mortes)

causas_mortes.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6120 entries, 0 to 6119
Data columns (total 34 columns):
#   Column                                     Non-Null Count  Dtype
#   ...
```

Saída:

```

Data columns (total 34 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Country/Territory                         6120 non-null   object
1   Code                                       6120 non-null   object
2   Year                                       6120 non-null   int64
3   Meningitis                               6120 non-null   int64
4   Alzheimer's Disease and Other Dementias  6120 non-null   int64
5   Parkinson's Disease                      6120 non-null   int64
6   Nutritional Deficiencies                 6120 non-null   int64
7   Malaria                                  6120 non-null   int64
8   Drowning                                 6120 non-null   int64
9   Interpersonal Violence                   6120 non-null   int64
10  Maternal Disorders                       6120 non-null   int64
11  HIV/AIDS                                 6120 non-null   int64
12  Drug Use Disorders                       6120 non-null   int64
13  Tuberculosis                             6120 non-null   int64
14  Cardiovascular Diseases                  6120 non-null   int64
15  Lower Respiratory Infections             6120 non-null   int64
16  Neonatal Disorders                       6120 non-null   int64
17  Alcohol Use Disorders                    6120 non-null   int64
18  Self-harm                                6120 non-null   int64
19  Exposure to Forces of Nature              6120 non-null   int64
20  Diarrheal Diseases                       6120 non-null   int64
21  Environmental Heat and Cold Exposure      6120 non-null   int64
22  Neoplasms                                6120 non-null   int64
23  Conflict and Terrorism                   6120 non-null   int64
24  Diabetes Mellitus                       6120 non-null   int64
25  Chronic Kidney Disease                   6120 non-null   int64
26  Poisonings                               6120 non-null   int64
27  Protein-Energy Malnutrition              6120 non-null   int64
28  Road Injuries                            6120 non-null   int64
29  Chronic Respiratory Diseases              6120 non-null   int64
30  Cirrhosis and Other Chronic Liver Diseases 6120 non-null   int64
31  Digestive Diseases                       6120 non-null   int64
32  Fire, Heat, and Hot Substances            6120 non-null   int64
33  Acute Hepatitis                          6120 non-null   int64
dtypes: int64(32), object(2)
memory usage: 1.6+ MB

```

Entendendo que as três primeiras colunas não participarão do cálculo de correlação, pois não geram valores significativos para a análise, criaremos uma nova lista a partir desta condição:

▼ Criando a matriz numérica para o cálculo da correlação.

```

[8] variaveis_numericas = causas_mortes.drop(columns= ['Country/Territory', 'Code', 'Year'])
    variaveis_numericas.corr()

```

	Meningitis	Alzheimer's Disease and Other Dementias	Parkinson's Disease	Nutritional Deficiencies	Malaria	Drowning	Interpersonal Violence
Meningitis	1.000000	0.216713	0.351668	0.760851	0.755261	0.576347	0.447242
Alzheimer's Disease and Other Dementias	0.216713	1.000000	0.950785	0.193209	0.031290	0.599403	0.429622
Parkinson's Disease	0.351668	0.950785	1.000000	0.313033	0.084109	0.753663	0.485528
Nutritional Deficiencies	0.760851	0.193209	0.313033	1.000000	0.411149	0.596367	0.407065
Malaria	0.755261	0.031290	0.084109	0.411149	1.000000	0.195839	0.184469
Drowning	0.576347	0.599403	0.753663	0.596367	0.195839	1.000000	0.539339
Interpersonal Violence	0.447242	0.429622	0.485528	0.407065	0.184469	0.539339	1.000000

Agora, iremos importar a biblioteca `sklearn.preprocessing` e aplicaremos o método `StandardScaler` para padronizar as variáveis. Como a função `StandardScaler()` retorna um array, iremos criar uma nova variável para armazenar as colunas da lista "variaveis numéricas" e, posteriormente, devolvê-las em forma de dataframe com os dados já padronizados. O método `StandardScaler` aplicará o procedimento de Z Score que se trata de um método para equalizar a escala dos dados. Consiste em selecionar cada registro, subtrair da média da variável e

dividir pelo desvio padrão da respectiva coluna. Tendo feito isso, teremos nossa tabela com os dados padronizados.

## ▼ Padronizando as variáveis numéricas

```

[9] from sklearn.preprocessing import StandardScaler

[32] colunas_numericas = variaveis_numericas.columns
     colunas_numericas

[33] padronizar = StandardScaler()
     variaveis_numericas = padronizar.fit_transform(variaveis_numericas)
     variaveis_numericas

dados_padronizados = pd.DataFrame(variaveis_numericas, columns= colunas_numericas)
dados_padronizados

```

Saída:

	Meningitis	Alzheimer's Disease and Other Dementias	Parkinson's Disease	Nutritional Deficiencies	Malaria	Drowning	Interpersonal Violence
0	0.057004	-0.173062	-0.145985	0.005270	-0.191593	-0.036522	-0.050031
1	-0.261658	-0.216862	-0.207291	-0.344914	-0.196091	-0.202428	-0.282897
2	-0.148138	-0.160258	-0.130989	-0.273852	-0.195656	-0.058634	-0.229172
3	-0.278119	-0.237545	-0.227579	-0.350893	-0.196091	-0.221827	-0.312668
4	-0.278274	-0.237371	-0.227359	-0.351234	-0.196091	-0.222369	-0.313010
...	...	...	...	...	...	...	...
199	-0.174849	-0.143341	-0.165612	-0.212015	-0.188788	-0.096075	0.172739
200	-0.089748	0.497955	0.429363	-0.099785	-0.174183	0.663863	-0.037712
201	-0.168637	-0.174800	-0.171345	-0.028040	-0.019910	-0.097974	-0.211720
202	0.430174	-0.221265	-0.208393	0.373732	0.203762	-0.157391	-0.235502
203	-0.102482	-0.211242	-0.200675	0.015349	-0.020635	-0.147488	-0.177329

204 rows × 31 columns

Com os dados padronizados, aplicaremos o método do PCA à matriz já balanceada. Importamos o método PCA da biblioteca `sklearn.decomposition`. Primeiramente, identificaremos todos os possíveis fatores que explicam os dados.

### ▼ Aplicando o PCA para todos os possíveis fatores

```
✓ [36] from sklearn.decomposition import PCA
```

```
✓ n_fatores = dados_padronizados.shape[1]  
n_fatores
```

```
31
```

```
[ ] pca = PCA(n_components= n_fatores)  
pca.fit(dados_padronizados)
```

```
PCA(n_components=31)
```

Agora, identificaremos os Autovalores, Autovetores e a Variância dos Fatores.

### ▼ Checando os Autovalores, Autovetores e a Variância explicada pelos fatores

```
✓ print("Auto-valores:")  
print(pca.explained_variance_, '\n')  
  
print("Auto-vetores:")  
print(pca.components_, '\n')  
  
print("Variância explicada:")  
print(pca.explained_variance_ratio_, '\n')
```

Saída:

31 autovalores:

```
Auto-valores:  
[1.99300611e+01 4.74314119e+00 1.22898295e+00 1.09824170e+00  
1.00989189e+00 8.29332013e-01 7.87385610e-01 5.04676150e-01  
3.17184379e-01 2.40795831e-01 1.40508114e-01 1.02866631e-01  
9.47869658e-02 2.78972100e-02 1.99389945e-02 1.83110228e-02  
1.29231371e-02 9.72009844e-03 7.72984349e-03 7.04018311e-03  
5.39136064e-03 4.21260567e-03 3.53495272e-03 2.60846074e-03  
1.88005946e-03 1.29350076e-03 9.82436491e-04 7.05879334e-04  
3.75837843e-04 1.99646016e-04 1.09623355e-04]
```

31 autovetores, cada um com 31 elementos:

```
Auto-vetores:
[[ 1.64461007e-01  1.55374077e-01  1.87791159e-01  1.81975191e-01
   7.38994117e-02  2.08991807e-01  1.64893359e-01  1.87456007e-01
   1.47464687e-01  1.12899741e-01  1.94307159e-01  1.94594677e-01
   2.10773825e-01  1.91457511e-01  1.70387251e-01  2.16127379e-01
   9.53307154e-02  1.81553618e-01  1.56611072e-01  1.69248378e-01
   1.00503425e-02  2.15336899e-01  2.16034816e-01  1.58309621e-01
   1.68703876e-01  1.98585985e-01  2.04568147e-01  2.18193507e-01
   2.21031542e-01  2.17714636e-01  1.87054867e-01]
 [-2.49462564e-01  2.85813742e-01  2.37920308e-01 -2.12549602e-01
  -2.15634606e-01  1.00001759e-01 -4.70192126e-03 -2.43500374e-01
  -2.25710792e-01  2.03856365e-01 -1.99568265e-01  2.19671840e-01
  -1.31284373e-01 -2.13431730e-01  1.63834801e-01  7.79670446e-02
  -3.74356311e-02 -2.54652134e-01  2.99555696e-02  2.87395688e-01
  -2.47342315e-02  3.63909470e-02  7.94913163e-02  2.52992438e-01
  -2.12195215e-01  1.78279485e-01  1.44199665e-01  2.09896776e-02
  2.58388626e-02 -3.74408850e-02 -2.00175250e-01]
```

Variância explicada pelos fatores:

```
Variância explicada:
[6.39753700e-01 1.52254532e-01 3.94502748e-02 3.52534892e-02
 3.24174659e-02 2.66215052e-02 2.52750283e-02 1.62000725e-02
 1.01815985e-02 7.72953096e-03 4.51030157e-03 3.30201236e-03
 3.04265561e-03 8.95498678e-04 6.40040463e-04 5.87782672e-04
 4.14831885e-04 3.12014545e-04 2.48127487e-04 2.25989433e-04
 1.73062336e-04 1.35224375e-04 1.13471759e-04 8.37314247e-05
 6.03497898e-05 4.15212926e-05 3.15361492e-05 2.26586820e-05
 1.20643710e-05 6.40862448e-06 3.51890275e-06]
```

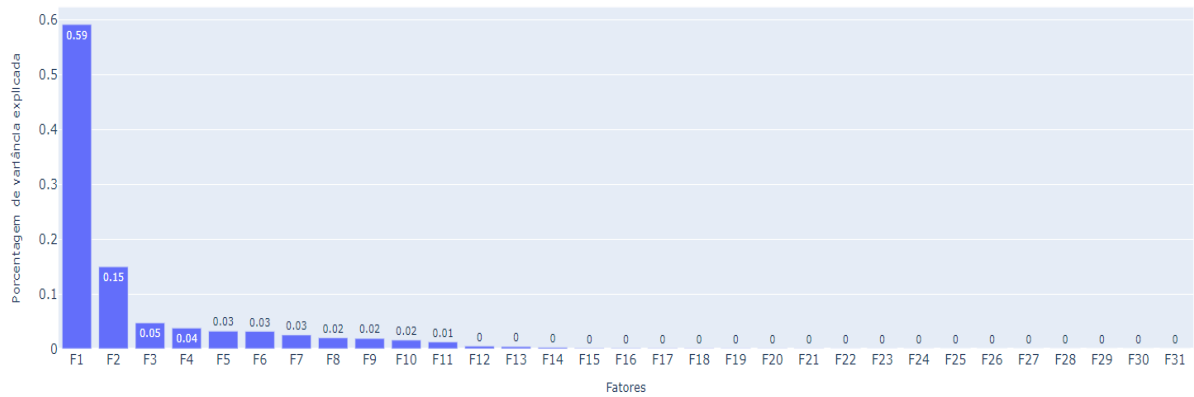
Agora iremos plotar esses valores através da biblioteca plotly.express

```
import plotly.express as px
import numpy as np

[ ] fatores = [f'F{i+1}' for i in range(n_fatores)]
    fatores

[ ] fig = px.bar(x=fatores, y=pca.explained_variance_ratio_, text=np.around(pca.explained_variance_ratio_, decimals=2), title='Scree Plot')
    fig.update_layout(yaxis={'title': 'Porcentagem de variância explicada', 'tickfont': {'size': 15}},
                      xaxis={'title': 'Fatores', 'tickfont': {'size': 15}},
                      title={'font': {'size': 25}})
    fig.show()
```

Scree Plot



Podemos também analisar a variância acumulada entre os fatores:

```

▶ variância_acumulada = [sum(pca.explained_variance_ratio_[0:i+1]) for i in range(n_fatores)]
variância_acumulada

[0.6397537000892095,
0.7920082323249253,
0.8314585071514891,
0.8667119963309233,
0.8991294621877256,
0.9257509673534609,
0.9510259956474102,
0.9672260681424226,
0.9774076666366699,
0.985137197592408,
0.9896474991659573,
0.9929495115253922,
0.9959921671313814,
0.9968876658092681,
0.9975277062718975,
0.9981154889442506,
0.9985303208288818,
0.9988423353738827,
0.9990904628611288,
0.9993164522937183,
0.999489514629319,
0.9996247390047608,
0.9997382107634321,
0.9998219421881727,
0.9998822919780096,
0.999923813270599,
0.9999553494198146,
0.9999780081017727,
0.999990072472769,
0.9999964810972525,
0.9999999999999993]

```



Por fim, iremos plotar os dois maiores fatores que explicam, juntos, aproximadamente 74% dos dados.

#### ▼ Verificando os dois maiores Fatores

```
[ ] pca = PCA(n_components= 2)
pca.fit(dados_padronizados)

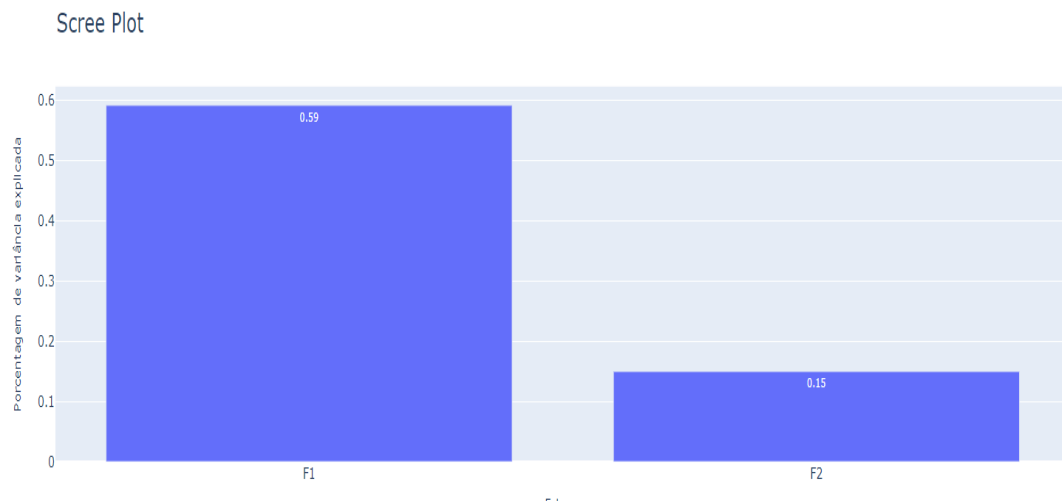
PCA(n_components=2)

[ ] fatores2 = [f'F{i+1}' for i in range(pca.n_components)]
fatores2

['F1', 'F2']

[ ] fig = px.bar (x= fatores2, y= pca.explained_variance_ratio_, text= np.around(pca.explained_variance_ratio_, decimals= 2), title= 'Scree Plot' )
fig.update_layout(yaxis= { 'title': 'Porcentagem de variância explicada', 'tickfont' : { 'size' : 15}},
                  xaxis= { 'title' : 'Fatores', 'tickfont' : { 'size' : 15}},
                  title = { 'font' : { 'size' : 25}})
fig.show()
```

Visualização do gráfico:



Link do repositório no Github: <https://github.com/julianacamara98/Fatec>