

 AlvimPedro / ppge_machine_learning

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

 Code

ppge_machine_learning / lista3 / ex2.ipynb 

Questão 2

```
In [329... import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

X = np.loadtxt("data_pca_question.csv",
               delimiter=",")
```

```
In [330... X
```

```
Out[330... array([[ 2.36377864, -6.42082233, -0.36402731],
        [ 1.97751565, -2.01800454,  3.97859695],
        [ 2.03153387, -4.97005594, -2.18583111],
        ...,
        [ 2.64450005, -4.00466208, -5.14635009],
        [ 2.33976935, -10.24804201, -6.64026619],
        [ 3.04055559, -7.36322362, -7.39808814]])
```

a)

Calculada a média de cada coluna do dataset que tem 3 dimensões

```
In [331... X.mean(axis=0)
```

```
Out[331... array([ 2.52389638, -2.81275819,  0.30387943])
```

Centralizando os dados em média 0 fazendo a subtração dos valores pela média

```
In [332... X_centered = X - X.mean(axis=0)
X_centered
```

```
Out[332... array([[ -0.16011774, -3.60806414, -0.66790673],
        [ -0.54638073,  0.79475365,  3.67471752],
        [ -0.49236251, -2.15729775, -2.48971054],
        ...,
        [  0.12060367, -1.19190389, -5.45022952],
        [ -0.18412703, -7.43528381, -6.94414562],
        [  0.51665921, -4.55046542, -7.70196757]])
```

Por fim, normalizando com variancia em 1 dividindo pelo desvio padrão dos dados

```
In [333... X_normalized = X_centered/X.std(axis=0)
X_normalized
```

```
Out[333...] array([[ -0.11370511, -0.67076356, -0.07132016],
        [ -0.38800373,  0.14775009,  0.39239229],
        [ -0.34964354, -0.40105626, -0.26585532],
        ...,
        [  0.08564481, -0.22158301, -0.58198433],
        [ -0.13075493, -1.38226962, -0.74150711],
        [  0.36689747, -0.84596234, -0.82242857]])
```

Abaixo é demonstrado que a média está bem próxima de 0 e o desvio padrão igual a 1.

```
In [334...] X_normalized.mean(axis=0)
```

```
Out[334...] array([-2.13162821e-17,  7.39186490e-16, -4.03010958e-17])
```

```
In [335...] X_normalized.std(axis=0)
```

```
Out[335...] array([1.,  1.,  1.])
```

b)

Calculada a variância da transposta da matriz de dados

```
In [336...] cov_x = np.cov(X_normalized.transpose())
cov_x
```

```
Out[336...] array([[ 1.001001, -0.53439232, -0.60672834],
        [-0.53439232,  1.001001,  0.94108144],
        [-0.60672834,  0.94108144,  1.001001]])
```

Feito o cálculo dos Autovalores e Autovetores utilizando a matriz de covariância gerada.

```
In [337...] eigenvalues, eigenvectors = np.linalg.eig(cov_x)
```

```
In [338...] eigenvalues
```

```
Out[338...] array([2.40588855, 0.541553,  0.05556146])
```

```
In [339...] eigenvectors
```

```
Out[339...] array([[ 0.49831409, -0.86283557,  0.08484009],
        [-0.60532847, -0.41630393, -0.6784309 ],
        [-0.62069357, -0.28671556,  0.72974905]])
```

Quanto maior o Autovalor (`eigenvalues`), maior está preservada a variância, sendo assim as componentes principais. Em que os Autovetores (`eigenvectors`) são representados por cada coluna da variável, estando na ordem das Componentes principais, primária, secundária e terciária.

c)

Analisando a razão da soma dos Autovalores, é possível identificar se a proporção de variância se mantém acima de 90% em determinada dimensão `k` que é um nível de variância preservado considerado ok.

```
In [340...] eigenvalues[0]/eigenvalues.sum()
```

```
Out[340...] 0.801160886303813
```

```
In [341...] (eigenvalues[0]+eigenvalues[1])/eigenvalues.sum()
```

```
Out[341...] 0.9814980354787338
```

```
In [342...] (eigenvalues[0]+eigenvalues[1]+eigenvalues[2])/eigenvalues.sum()
```

```
Out[342...] 1.0
```

Considerando um threshold de 90%, é possível reduzir a dimensionalidade para 2 sem perder uma quantidade significativa de variância na informação.

d)

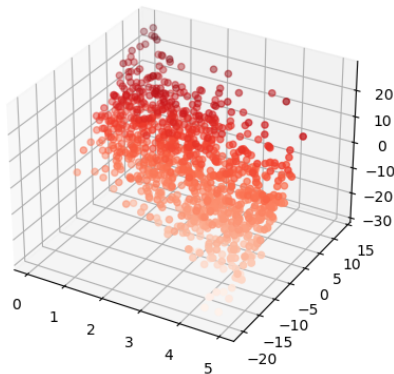
Primeiramente os dados originais plotados para ter uma noção dos dados apresentados

```
In [343... fig = plt.figure()
ax = plt.axes(projection='3d')

xdata = X[:, 0]
ydata = X[:, 1]
zdata = X[:, 2]

ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='Reds')
```

Out[343... <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fa23b0c10f0>



Feita a redução de dimensionalidade PCA para uma e duas dimensões fazendo o produto das matrizes dos dados originais normalizados com os autovetores.

```
In [344... pca_data_1d = X_normalized.dot(eigenvectors[:,0:1])
pca_data_2d = X_normalized.dot(eigenvectors[:,0:2])
```

Abaixo, o plot dos dados originais normalizados com os dados reduzidos pelo PCA, mostrando o 2D atuando em duas dimensões sendo um plano, e o de 1 dimensão sendo uma reta.

```
In [350... fig = plt.figure()
ax = plt.axes(projection='3d')

xdata = X_normalized[:, 0]
ydata = X_normalized[:, 1]
zdata = X_normalized[:, 2]

ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='Reds', s=2, label='X Normalized')

xdata = pca_data_2d[:, 0]
ydata = pca_data_2d[:, 1]
zdata = 0

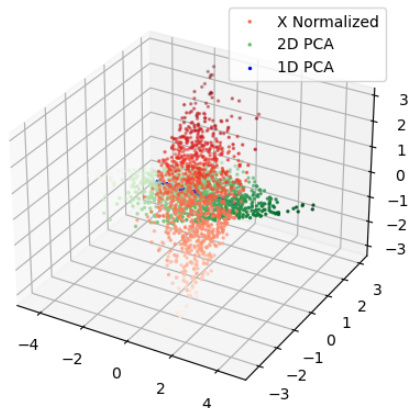
ax.scatter3D(xdata, ydata, zdata, c=xdata, cmap='Greens', s=2, label='2D PCA')

xdata = X_normalized[:, 0]
ydata = 0
zdata = 0

ax.scatter3D(xdata, ydata, zdata, color='blue', s=2, label='1D PCA')

ax.legend(loc='upper right')
```

Out[350... <matplotlib.legend.Legend at 0x7fa23af79270>



Aqui foi feita a reconstrução dos dados, pegando os dados reduzidos por PCA multiplicando pelos Autovetores com o número de componentes principais de acordo com a dimensionalidade dos dados, foi adicionado a média que foi retirada no início para a normalização, assim como a multiplicação pelo desvio padrão.

```
In [351]: reconstructed_2d = (pca_data_2d.dot(eigenvectors[0:2]) + X.mean(axis=0))*X.std(axis=0)
reconstructed_1d = (pca_data_1d.dot(eigenvectors[0:1]) + X.mean(axis=0))*X.std(axis=0)

# reconstructed_2d = (pca_data_2d.dot(eigenvectors[0:2]))
# reconstructed_1d = (pca_data_1d.dot(eigenvectors[0:1]))
```

Em seguida, foi realizado o plot dos dados originais e as reconstruções em 2D e 1D do PCA.

```
In [353]: fig = plt.figure()
ax = plt.axes(projection='3d')

xdata = X[:, 0]
ydata = X[:, 1]
zdata = X[:, 2]

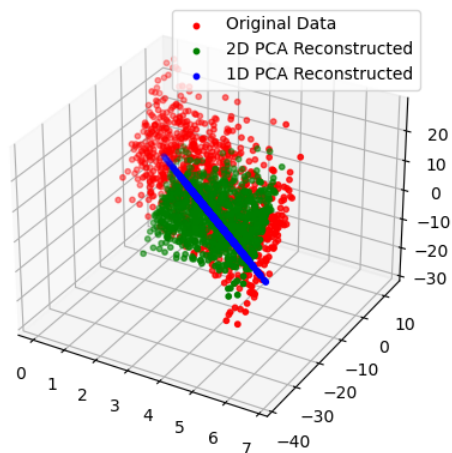
ax.scatter3D(xdata, ydata, zdata, color='red', s=10, label='Original Data')

xdata = reconstructed_2d[:, 0]
ydata = reconstructed_2d[:, 1]
zdata = reconstructed_2d[:, 2]

ax.scatter3D(xdata, ydata, zdata, color='green', s=10, label='2D PCA Reconstructed')

xdata = reconstructed_1d[:, 0]
ydata = reconstructed_1d[:, 1]
zdata = reconstructed_1d[:, 2]

ax.scatter3D(xdata, ydata, zdata, color='blue', s=10, label='1D PCA Reconstructed')
ax.legend(loc='upper right')
```



e)

Com o PCA é possível fazer a redução da dimensionalidade, sendo útil em diversas análises, como casos de simplificar os dados sem perder muita informação. Nesse caso reduzir para 2 dimensões manteve um bom nível de informação, enquanto que 1 dimensão houve grande perda, como pode ser visto na reconstrução.

Há casos em que a redução é feita para 2 ou 3 dimensões para fazer uma análise visual dos dados, conseguindo retirar insights.

Para esse dataset, a redução para 2 dimensões é possível, mantendo cerca de 98% da variância. Podendo ser útil para insights ou utilizar em duas dimensões para outros algoritmos de classificação ou clusterização como exemplo.

Questão 3

```
In [81]: %matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

#Dataset classico do Iris
from sklearn.datasets import load_iris
iris = load_iris()
```

```
In [94]: #Transformando em Dataframe Pandas para manipulação
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
In [83]: df
```

```
Out[83]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

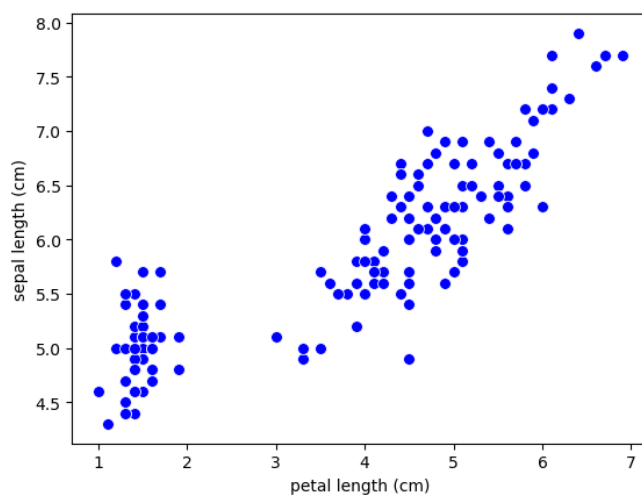
150 rows × 4 columns

Dataset Iris plotando as características:

- Petal length
- Sepal length

```
In [84]: sns.scatterplot(x='petal length (cm)', y='sepal length (cm)',
                        data=df, s=50, color='blue')
```

```
Out[84]: <AxesSubplot:xlabel='petal length (cm)', ylabel='sepal length (cm)'>
```



```
In [95]: #Variável contendo os valores dos atributos selecionados
X = df[['petal length (cm)', 'sepal length (cm)']].values
```

Método do cotovelo

Primeiramente foi feita a escolha do número de clusters para realizar o clustering. Para isso,

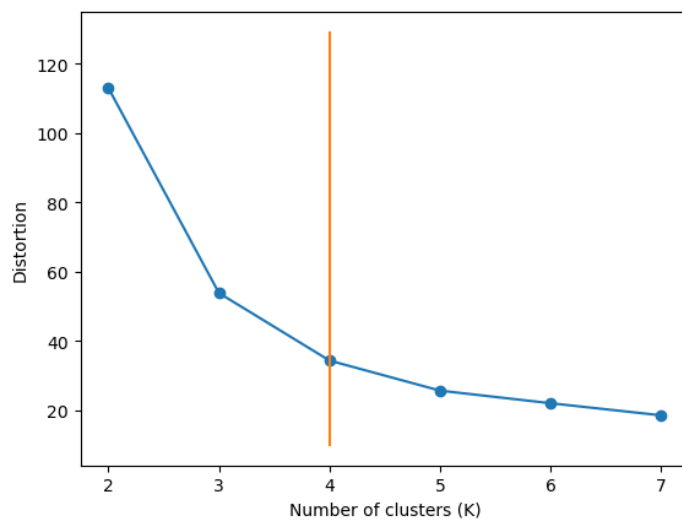
- Foram treinados os modelos do K-means para diversos valores de clusters (2-7).
- Em seguida verificado o parâmetro `inertia_` em que consiste na soma das distancias ao quadrado para o Centroid mais próximo.

```
In [86]: distortions = []

for k in range(2, 8):
    KMeans_model = KMeans(n_clusters=k, random_state=14, n_init=10)
    KMeans_model.fit(X)
    distortions.append(KMeans_model.inertia_)

plt.plot(range(2, 8), distortions, marker='o')
plt.plot(np.ones(120)*4, range(10,130))
plt.xlabel('Number of clusters (K)')
plt.ylabel('Distortion')
```

Out[86]: Text(0, 0.5, 'Distortion')



Os dados da distorção foram plotados no gráfico em função do número de clusters.

Foi identificado visualmente, utilizando o método do cotovelo que onde começa a ter uma diferença de distorção menor é quando o número de cluster é 4.

Dessa forma foi escolhido o valor de 4 cluster para prosseguir com a clusterização dos dados como indica a figura.

K-Means

Anteriormente foi realizado o procedimento de treinamento dos modelos para verificar as distâncias para a decisão do K (número de clusters), porém nessa seção será explicado de forma mais detalhada.

Foi criado uma instância do modelo utilizado a função `KMeans()` do Scikit Learn `sklearn.cluster` em que é passado como parâmetros o número de clusters (`n_clusters`), o `random_state` para um valor fixo de aleatoriedade para dar o mesmo resultado independente do momento que a função é executada e o parâmetro `n_init` que é o número de vezes que o algoritmo do K-Means é rodado para diferentes valores de Centroids, onde é escolhido o melhor caso no final.

Em seguida treinado o modelo de fato com a função `fit()`

```
In [96]: model = KMeans(n_clusters=4, random_state=14, n_init=10)
          model.fit(X)
```

Out[96]: KMeans(n_clusters=4, n_init=10, random_state=14)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Os 4 centroides são encontrados no atributo `cluster_centers_` e mostrado a posição espacial nos parâmetros do dataset de tamanho de pétalas e sépalas.

```
In [88]: centroids = model.cluster_centers_
centroids
```

```
Out[88]: array([[6.03181818, 7.12272727],
                [1.462      , 5.006      ],
                [4.94       , 6.292      ],
                [3.96071429, 5.53214286]])
```

É possível visualizar as separações por clusters dos dados de acordo com a distância ao centroid mais próximo pelo atributo `labels`.

```
In [89]: model.labels_
```

[illegible]

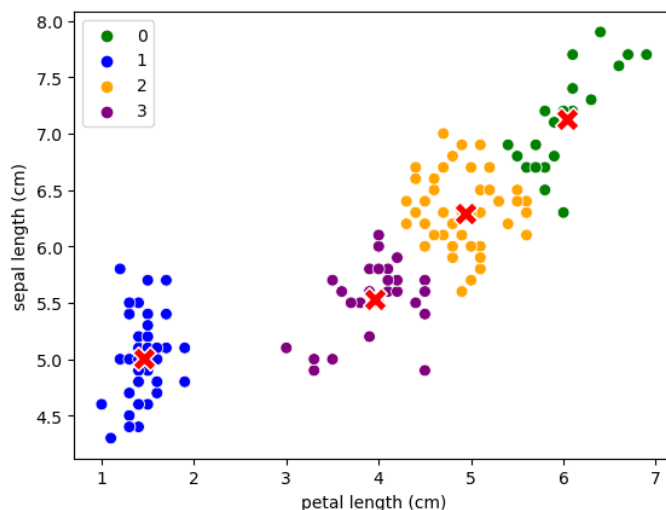
Em seguida, com essas informações, foram plotados novamente os pontos do dataset, porém separados pelos Labels separados com o K-Means de 4 Clusters, indicados pela cor dos pontos.

Também foi indicado a posição com um "X" vermelho a posição dos centroides escolhidos.

```
In [93]: sns.scatterplot(x='petal length (cm)', y='sepal length (cm)', data=df,
                    s=50, hue=model.labels_, palette=['green', 'blue', 'orange', 'purple'])

sns.scatterplot(x=centroids[:, 0], y=centroids[:, 1], marker='X',
                s=200, color='red')
```

```
Out[93]: <AxesSubplot:xlabel='petal length (cm)', ylabel='sepal length (cm)'>
```



Devido a utilização do dataset do Iris, já conhecido, se sabe que na realidade, existem 3 espécies de flores Iris e não 4 como se deu a separação do número de clusters pelo método do cotovelo. Indicando que pode não ser o modelo mais apropriado para determinados datasets devido sua simplicidade.

A escolha do número de clusters é importante e existem diversos métodos para sua escolha.

Para comparação, será indicado o algoritmo do K-Means para o número de 3 clusters, que pelo dataset, seria a escolha correta.

```
In [97]: model = KMeans(n_clusters=3, random_state=14, n_init=10)
model.fit(X)
centroids = model.cluster_centers_
sns.scatterplot(x='petal length (cm)', y='sepal length (cm)', data=df,
                s=50, hue=model.labels_, palette=['green', 'blue', 'orange'])

sns.scatterplot(x=centroids[:, 0], y=centroids[:, 1], marker='X',
                s=200, color='red')
```

```
Out[97]: <AxesSubplot:xlabel='petal length (cm)', ylabel='sepal length (cm)'>
```

