

PCA-Sem-PBS-Eletrodos-HIV-ReviewMarli

August 13, 2022

```
[1]: # 1. PCA dataset
import pandas as pd

df = pd.read_csv('../mestrado/ReviewMarli/Eletrodo/allHIVData.csv')
print(df)

# df['Sensor'] = df['Sensor'].astype('string')
df.head()
```

	Eletrodo	Freq(Hz)	Z'(a)	Z''(b)	antiHIVmicrog/ml \
0	6.0	1000000.00	72.6	11.30	0.001
1	6.0	794000.00	72.8	7.04	0.001
2	6.0	631000.00	72.9	3.16	0.001
3	6.0	501000.00	73.1	-0.51	0.001
4	6.0	398000.00	73.4	-4.17	0.001
...
1215	4.0	2.51	59700.0	-193000.00	1.000
1216	4.0	2.00	68900.0	-234000.00	1.000
1217	4.0	1.58	80500.0	-285000.00	1.000
1218	4.0	1.26	93500.0	-346000.00	1.000
1219	4.0	1.00	109000.0	-421000.00	1.000

	antiHCVmicrog/ml
0	0.01
1	0.01
2	0.01
3	0.01
4	0.01
...	...
1215	0.00
1216	0.00
1217	0.00
1218	0.00
1219	0.00

[1220 rows x 6 columns]

```
[1]: Eletrodo    Freq(Hz)  Z'(a)  Z''(b)  antiHIVmicrog/ml  antiHCVmicrog/ml
      0         6.0  1000000.0  72.6   11.30             0.001             0.01
      1         6.0   794000.0  72.8    7.04             0.001             0.01
      2         6.0   631000.0  72.9    3.16             0.001             0.01
      3         6.0   501000.0  73.1   -0.51             0.001             0.01
      4         6.0   398000.0  73.4   -4.17             0.001             0.01
```

```
[2]: print(df.dtypes)
```

```
Eletrodo          float64
Freq(Hz)          float64
Z'(a)             float64
Z''(b)           float64
antiHIVmicrog/ml  float64
antiHCVmicrog/ml  float64
dtype: object
```

```
[3]: df_features = df.iloc[:,1:].copy()
      X = df_features.to_numpy()
      # X = df.values
      X
```

```
[3]: array([[ 1.00e+06,  7.26e+01,  1.13e+01,  1.00e-03,  1.00e-02],
             [ 7.94e+05,  7.28e+01,  7.04e+00,  1.00e-03,  1.00e-02],
             [ 6.31e+05,  7.29e+01,  3.16e+00,  1.00e-03,  1.00e-02],
             ...,
             [ 1.58e+00,  8.05e+04, -2.85e+05,  1.00e+00,  0.00e+00],
             [ 1.26e+00,  9.35e+04, -3.46e+05,  1.00e+00,  0.00e+00],
             [ 1.00e+00,  1.09e+05, -4.21e+05,  1.00e+00,  0.00e+00]])
```

```
[4]: Y = df.iloc[:, 0].to_numpy()
      Y
```

```
[4]: array([6., 6., 6., ..., 4., 4., 4.])
```

```
[5]: X.shape
```

```
[5]: (1220, 5)
```

```
[6]: Y.shape
```

```
[6]: (1220,)
```

```
[7]: # 2. PCA analysis
      # 2.1 Load library

      from sklearn.preprocessing import scale # Data scaling
```

```
from sklearn import decomposition # PCA
import pandas as pd # pandas
```

```
[8]: # 2.2 Data scaling
x = scale(X)

# Standardize the Data
# PCA is effected by scale so you need to scale the features in your data
→ before applying PCA. Use StandardScaler to help you standardize the
→ dataset's features onto unit scale (mean = 0 and variance = 1) which is a
→ requirement for the optimal performance of many machine learning algorithms.
x
```

```
[8]: array([[ 4.71762672, -0.36758838,  0.29776146, -0.59910015,  0.20652852],
 [ 3.66164406, -0.3675857 ,  0.29774615, -0.59910015,  0.20652852],
 [ 2.82608496, -0.36758436,  0.2977322 , -0.59910015,  0.20652852],
 ...,
 [-0.4084946 ,  0.70946422, -0.72676567,  1.98845304, -0.25242374],
 [-0.40849624,  0.88355519, -0.94604173,  1.98845304, -0.25242374],
 [-0.40849758,  1.09112519, -1.21564345,  1.98845304, -0.25242374]])
```

```
[9]: # 2.3 Perform PCA analysis
pca = decomposition.PCA(n_components=3)
pca.fit(x) # when build the model we use pca.fit function. Where by the
→ argument will be the input data, which essentially is the x variable
# we are using the x variable because pca is an unsupervised learning approach,
→ meaning that it does not need the y variable or the class label in order to
→ learn
# it'll goind to cluster the data based on similarity and differences, based on
→ the eigenvalue that are inherently present in the data set
```

```
[9]: PCA(n_components=3)
```

```
[10]: # 2.4 compure the scores value
# scores value will essentially be represented by the data samples so we're
→ gonna use the pca.transform function
scores = pca.transform(x)
```

```
[11]: scores
```

```
[11]: array([[ -1.7817368 , -0.42485418,  4.38176347],
 [ -1.49942515, -0.2138078 ,  3.40529599],
 [ -1.27604194, -0.0468147 ,  2.63265453],
 ...,
 [ 1.26259044, -1.38958513, -0.59952051],
 [ 1.52735236, -1.36436812, -0.52132011],
 [ 1.84849243, -1.33371842, -0.42627322]])
```

```
[12]: scores_df = pd.DataFrame(scores, columns=['PC1', 'PC2', 'PC3']) # dataframe is
      ↪to make more readable
      scores_df
```

```
[12]:
```

	PC1	PC2	PC3
0	-1.781737	-0.424854	4.381763
1	-1.499425	-0.213808	3.405296
2	-1.276042	-0.046815	2.632655
3	-1.097881	0.086370	2.016438
4	-0.956720	0.191894	1.528206
...
1215	0.852451	-1.428493	-0.720179
1216	1.034600	-1.411205	-0.666566
1217	1.262590	-1.389585	-0.599521
1218	1.527352	-1.364368	-0.521320
1219	1.848492	-1.333718	-0.426273

[1220 rows x 3 columns]

```
[13]: y_label = []

for i in Y:
    if i == 1:
        y_label.append('Eletrodo 1')
    elif i == 2:
        y_label.append('Eletrodo 2')
    elif i == 3:
        y_label.append('Eletrodo 3')
    elif i == 4:
        y_label.append('Eletrodo 4')
    elif i == 5:
        y_label.append('Lignina')
    else:
        y_label.append('AntiHCV Eletrodo 1')

sensors = pd.DataFrame(y_label, columns=['Sensor'])
```

```
[14]: df_scores = pd.concat([scores_df, sensors], axis=1) # combine dataframes with
      ↪concat
      df_scores
```

```
[14]:
```

	PC1	PC2	PC3	Sensor
0	-1.781737	-0.424854	4.381763	AntiHCV Eletrodo 1
1	-1.499425	-0.213808	3.405296	AntiHCV Eletrodo 1
2	-1.276042	-0.046815	2.632655	AntiHCV Eletrodo 1
3	-1.097881	0.086370	2.016438	AntiHCV Eletrodo 1
4	-0.956720	0.191894	1.528206	AntiHCV Eletrodo 1

```

...      ...      ...      ...      ...
1215  0.852451 -1.428493 -0.720179      Eletrodo 4
1216  1.034600 -1.411205 -0.666566      Eletrodo 4
1217  1.262590 -1.389585 -0.599521      Eletrodo 4
1218  1.527352 -1.364368 -0.521320      Eletrodo 4
1219  1.848492 -1.333718 -0.426273      Eletrodo 4

```

```
[1220 rows x 4 columns]
```

```
[15]: feature_names = df.columns[1:]
      feature_names
```

```
[15]: Index(['Freq(Hz)', 'Z'(a)', 'Z''(b)', 'antiHIVmicrog/ml', 'antiHCVmicrog/ml'],
          dtype='object')
```

```
[16]: # 2.5 retrieve the loading values (remember PCA scores and loadings)
      # loadings value tell about the descriptor and scores value tell about the data
      ↪ samples
      # 150 flowers 150 score values 4 descriptors 4 loading values (1 for each
      ↪ descriptor)
      loadings = pca.components_.T
      df_loadings = pd.DataFrame(loadings, columns=['PC1', 'PC2', 'PC3'],
      ↪ index=feature_names)
      df_loadings
```

```
[16]:
```

	PC1	PC2	PC3
Freq(Hz)	-0.267334	-0.199857	0.924704
Z'(a)	0.677350	0.054767	0.170125
Z''(b)	-0.669663	-0.071518	-0.221568
antiHIVmicrog/ml	0.077959	-0.699177	-0.256949
antiHCVmicrog/ml	-0.123306	0.680510	-0.029402

```
[17]: # 2.6 explained variance
      # what's the contribution to the percent variance of the entire model
      ↪ contributed by each of the principal components
      explained_variance = pca.explained_variance_ratio_
      explained_variance
```

```
[17]: array([0.33664974, 0.21689237, 0.18867443])
```

```
[18]: # 3. Scree plot
      import numpy as np
      import plotly.express as px
```

```
[19]: # 3.1 preparing the explained variance data
      # add origin value, x and y are going to have origin zero, the subsequent lines
      ↪ of code are going to create the scree plot
```

```
# the scree plot does not start from zero, then we need to manually create the
↪zero origin
explained_variance = np.insert(explained_variance, 0, 0)
explained_variance
```

```
[19]: array([0.          , 0.33664974, 0.21689237, 0.18867443])
```

```
[20]: # 3.2 Preparing cumulative variance data
cumulative_variance = np.cumsum(np.round(explained_variance, decimals=3))
```

```
[21]: # 3.3 Combining dataframe
pc_df = pd.DataFrame(['', 'PC1', 'PC2', 'PC3'], columns=['PC'])
explained_variance_df = pd.DataFrame(explained_variance, columns=['Explained_
↪Variance'])
cumulative_variance_df = pd.DataFrame(cumulative_variance, columns=['Cumulative_
↪Variance'])
```

```
[22]: df_explained_variance = pd.concat([pc_df, explained_variance_df,
↪cumulative_variance_df], axis=1)
df_explained_variance
```

```
[22]:
```

	PC	Explained Variance	Cumulative Variance
0		0.000000	0.000
1	PC1	0.336650	0.337
2	PC2	0.216892	0.554
3	PC3	0.188674	0.743

```
[23]: # 3.4 Creating Scree Plot
# https://plotly.com/python/bar-charts/

fig = px.bar(df_explained_variance, x='PC', y='Explained Variance',
↪text='Explained Variance', width=800)
```

```
[24]: fig.update_traces(texttemplate='%{text:.3f}', textposition='outside') # limit_
↪decimal cases and text outside the bar
fig.show() #explained variance
```

```
[25]: # explained variance + cumulative variance
# https://plotly.com/python/creating-and-updating-figures

import plotly.graph_objects as go
fig = go.Figure()

fig.add_trace(
    go.Scatter(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Cumulative Variance'],
```

```

        marker=dict(size=15, color='LightSeaGreen')
    ))

fig.add_trace(
    go.Bar(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Explained Variance'],
        marker=dict(color='RoyalBlue')
    ))

fig.show()

```

[26]: *# Explained variance + cumulative variance (Separate Plot)*

```

from plotly.subplots import make_subplots
import plotly.graph_objects as go

fig = make_subplots(rows=1, cols=2)

fig.add_trace(
    go.Scatter(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Cumulative Variance'],
        marker=dict(size=15, color='LightSeaGreen')
    ), row=1, col=1
)

fig.add_trace(
    go.Bar(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Explained Variance'],
        marker=dict(color='RoyalBlue')
    ), row=1, col=2
)

fig.show()

```

[27]: *# 4. Scores plot*

```

# check API documentation for plotly.express https://plotly.com/python/
→ 3d-scatter-plots

import plotly.express as px

fig = px.scatter_3d(df_scores, x='PC1', y='PC2', z='PC3', color='Sensor')
fig.show()

```

```
[28]: # 4.1 Customize 3D Scatter Plot

fig = px.scatter_3d(df_scores, x='PC1', y='PC2', z='PC3', color='Sensor',
    ↪symbol='Sensor', opacity=0.4)

# tight layout
fig.update_layout(margin=dict(l=0, r=0, b=0, t=0)) # left, right, bottom, top
    ↪with zero margin

fig.show()

# https://plotly.com/python/templates/
# fig.update_layout(template='plotly_white')
# plotly, plotly_white, plotly_dark, ggplot2, seaborn, simple_white, none
```

```
[29]: # 5. Loadings Plot

loadings_label = df_loadings.index
# loadings_label = df_loadings.index.str.strip(' (cm)')

fig = px.scatter_3d(df_loadings, x='PC1', y='PC2', z='PC3', text=loadings_label)
fig.show()
```

```
[30]: loadings_label
```

```
[30]: Index(['Freq(Hz)', 'Z'(a)', 'Z''(b)', 'antiHIVmicrog/ml', 'antiHCVmicrog/ml'],
dtype='object')
```

```
[31]: df_loadings
```

```
[31]:
```

	PC1	PC2	PC3
Freq(Hz)	-0.267334	-0.199857	0.924704
Z'(a)	0.677350	0.054767	0.170125
Z''(b)	-0.669663	-0.071518	-0.221568
antiHIVmicrog/ml	0.077959	-0.699177	-0.256949
antiHCVmicrog/ml	-0.123306	0.680510	-0.029402

```
[ ]:
```

```
[ ]:
```