Redução de Dimensionalidade e Métodos de Variáveis Latentes em Aprendizado de Máquina

5.1 PCA: Uma Introdução

Introdução aos Desafios de Dimensionalidade

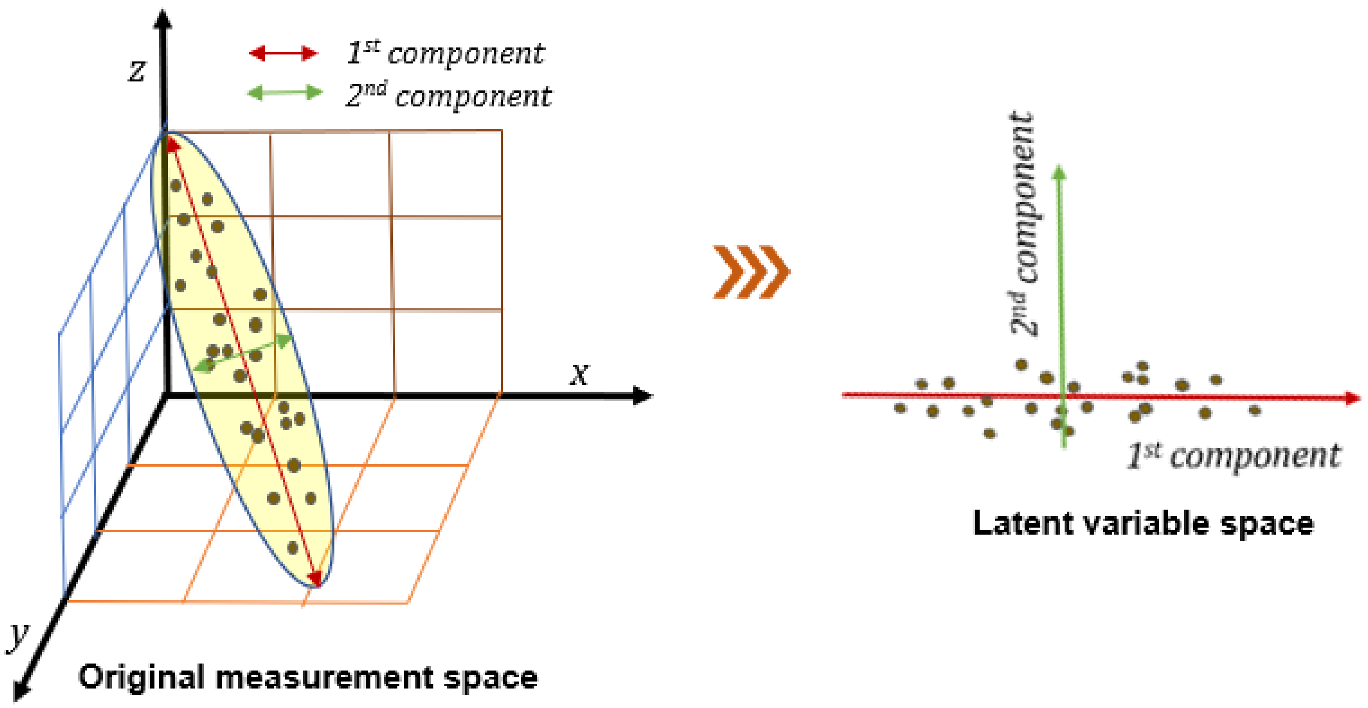
- Dados de alta dimensão apresentam desafios únicos:
- Questões algorítmicas devido à colinearidade
- Dificuldades de visualização
- Altos custos computacionais
- Treinamento lento do modelo
- Conhecido como "maldição da dimensionalidade"

Compreendendo Variáveis Latentes

- Variáveis de processo frequentemente mostram correlações devido a:
  - Leis de conservação de massa
  - Restrições termodinâmicas
  - Especificações do produto
  - Restrições operacionais
- Essas correlações sugerem variáveis ocultas (latentes)
- Métodos de variáveis latentes reduzem dimensionalidade preservando informação

Análise de Componentes Principais (PCA): Fundamentos

- Transforma variáveis correlacionadas de alta dimensão em variáveis não correlacionadas de baixa dimensão
- Preserva o máximo de informação possível
- Cria novas variáveis chamadas Componentes Principais (CPs)
- CP1 corresponde à direção de máxima variância
- CPs subsequentes são ortogonais aos anteriores


1st component
2nd component
z
x
y
Original measurement space
2nd component
1st component
Latent variable space

Fundamento Matemático do PCA

Perguntamos a cinco pessoas quantas horas de celular elas usam por semana

Observação:

1	2	3	4	5
5	7	3	38	7

$$x_1 = 5,\quad x_2 = 7,\ x_3 = 3,\ x_4 = 38,\ x_5 = 7$$

Média:

$$\bar{x} = \frac{\text{Sum of Data}}{n} = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$$

$$= \frac{5 + 7 + 3 + 38 + 7}{5} = \frac{60}{5} = 12\ \textit{Hours}$$

Fundamento Matemático do PCA

Mediana: $3 \; 5 \; 7 \; 7 \; 38$

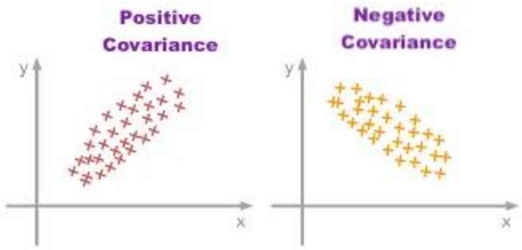
Desvio Padrão: $$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2}$$

$$S = \sqrt{\frac{(3-12)^2+(5-12)^2+(7-12)^2+(7-12)^2+(38-12)^2}{5-1}} = \sqrt{214} = 14.63$$

Fundamento Matemático do PCA

Covariância:

$$Cov\ (x,\ y) = \sum_{i=1}^{n} \frac{(x - \overline{x})(y - \overline{y})}{n-1}$$



Conjunto de dados tridimensionais usando as dimensões x , y e z.
A matriz de covariância tem 3 linhas e 3 colunas, e os valores são:

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

Para este caso de uso de análise exploratória de dados de PCA, vamos usar o conjunto de dados de intenção de compra de compradores on-line do repositório de aprendizado de máquina [da UCI]()

Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType	Weekend	Revenue
0	0	0	1	0	0.2	0.2	0	0	Feb	1	1	1	1	Returning_Visitor	FALSE	FALSE
0	0	0	2	64	0	0.1	0	0	Feb	2	2	1	2	Returning_Visitor	FALSE	FALSE
0	0	0	1	0	0.2	0.2	0	0	Feb	4	1	9	3	Returning_Visitor	FALSE	FALSE
0	0	0	2	2.666666667	0.05	0.14	0	0	Feb	3	2	2	4	Returning_Visitor	FALSE	FALSE
0	0	0	10	627.5	0.02	0.05	0	0	Feb	3	3	1	4	Returning_Visitor	TRUE	FALSE
0	0	0	19	154.2166667	0.01578947	0.0245614	0	0	Feb	2	2	1	3	Returning_Visitor	FALSE	FALSE
0	0	0	1	0	0.2	0.2	0	0.4	Feb	2	4	3	3	Returning_Visitor	FALSE	FALSE
0	0	0	0	0	0.2	0.2	0	0	Feb	1	2	1	5	Returning_Visitor	TRUE	FALSE
0	0	0	2	37	0	0.1	0	0.8	Feb	2	2	2	3	Returning_Visitor	FALSE	FALSE
0	0	0	3	738	0	0.02222222	0	0.4	Feb	2	4	1	2	Returning_Visitor	FALSE	FALSE
0	0	0	3	395	0	0.06666667	0	0	Feb	1	1	3	3	Returning_Visitor	FALSE	FALSE
0	0	0	16	407.75	0.01875	0.02583333	0	0.4	Feb	1	1	4	3	Returning_Visitor	FALSE	FALSE
0	0	0	7	280.5	0	0.02857143	0	0	Feb	1	1	1	3	Returning_Visitor	FALSE	FALSE
0	0	0	6	98	0	0.06666667	0	0	Feb	2	5	1	3	Returning_Visitor	FALSE	FALSE
0	0	0	2	68	0	0.1	0	0	Feb	3	2	3	3	Returning_Visitor	FALSE	FALSE
53	0	0	23	1668.285119	0.00833333	0.01631264	0	0	Feb	1	1	9	3	Returning_Visitor	FALSE	FALSE

Etapa 1: padronizar o conjunto de dados

Padronização ou Z-Score é o processo de padronização de todos os valores em um conjunto de dados de forma que a média de todos os valores seja 0 e o Desvio Padrão (DP) seja 1.

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.multivariate.pca import PCA
from sklearn.preprocessing import LabelEncoder


# Load dataset
dataset = pd.read_csv('online_shoppers_intention.csv')

# Remove unwanted columns
dataset = dataset.drop(['Administrative','Administrative_Duration',
'Informational', 'Informational_Duration'], axis=1)
```

	# Administrative	# Administrative_Duration	# Informational	# Informational_Duration	# ProductRelated	# Proc
0	0	0.0	0	0.0	1	
1	0	0.0	0	0.0	2	
2	0	0.0	0	0.0	1	
3	0	0.0	0	0.0	2	
4	0	0.0	0	0.0	10	
5	0	0.0	0	0.0	19	
6	0	0.0	0	0.0	1	
7	1	0.0	0	0.0	0	
8	0	0.0	0	0.0	2	
9	0	0.0	0	0.0	3	

12,330 rows x 18 cols 10 per page

Page 1 of 1233

```
...  <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 12330 entries, 0 to 12329
     Data columns (total 18 columns):
      #   Column                   Non-Null Count  Dtype  
     ---  ------                   --------------  -----  
      0   Administrative           12330 non-null  int64  
      1   Administrative_Duration  12330 non-null  float64
      2   Informational            12330 non-null  int64  
      3   Informational_Duration   12330 non-null  float64
      4   ProductRelated           12330 non-null  int64  
      5   ProductRelated_Duration  12330 non-null  float64
      6   BounceRates              12330 non-null  float64
      7   ExitRates                12330 non-null  float64
      8   PageValues               12330 non-null  float64
      9   SpecialDay               12330 non-null  float64
      10  Month                    12330 non-null  object 
      11  OperatingSystems         12330 non-null  int64  
      12  Browser                  12330 non-null  int64  
      13  Region                   12330 non-null  int64  
      14  TrafficType              12330 non-null  int64  
      15  VisitorType              12330 non-null  object 
      16  Weekend                  12330 non-null  bool   
      17  Revenue                  12330 non-null  bool   
     dtypes: bool(2), float64(7), int64(7), object(2)
     memory usage: 1.5+ MB
```

```python
# Remove unwanted columns
dataset = dataset.drop(['Administrative','Administrative_Duration',
'Informational', 'Informational_Duration', 'Month', 'VisitorType'], axis=1)
dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ProductRelated           12330 non-null  int64  
 1   ProductRelated_Duration  12330 non-null  float64
 2   BounceRates              12330 non-null  float64
 3   ExitRates                12330 non-null  float64
 4   PageValues               12330 non-null  float64
 5   SpecialDay               12330 non-null  float64
 6   OperatingSystems         12330 non-null  int64  
 7   Browser                  12330 non-null  int64  
 8   Region                   12330 non-null  int64  
 9   TrafficType              12330 non-null  int64  
 10  Weekend                  12330 non-null  bool   
 11  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(5), int64(5)
memory usage: 987.5 KB
```

```
pca = PCA(dataset, standardize=True, method='eig')
normalized_dataset = pca.transformed_data
```

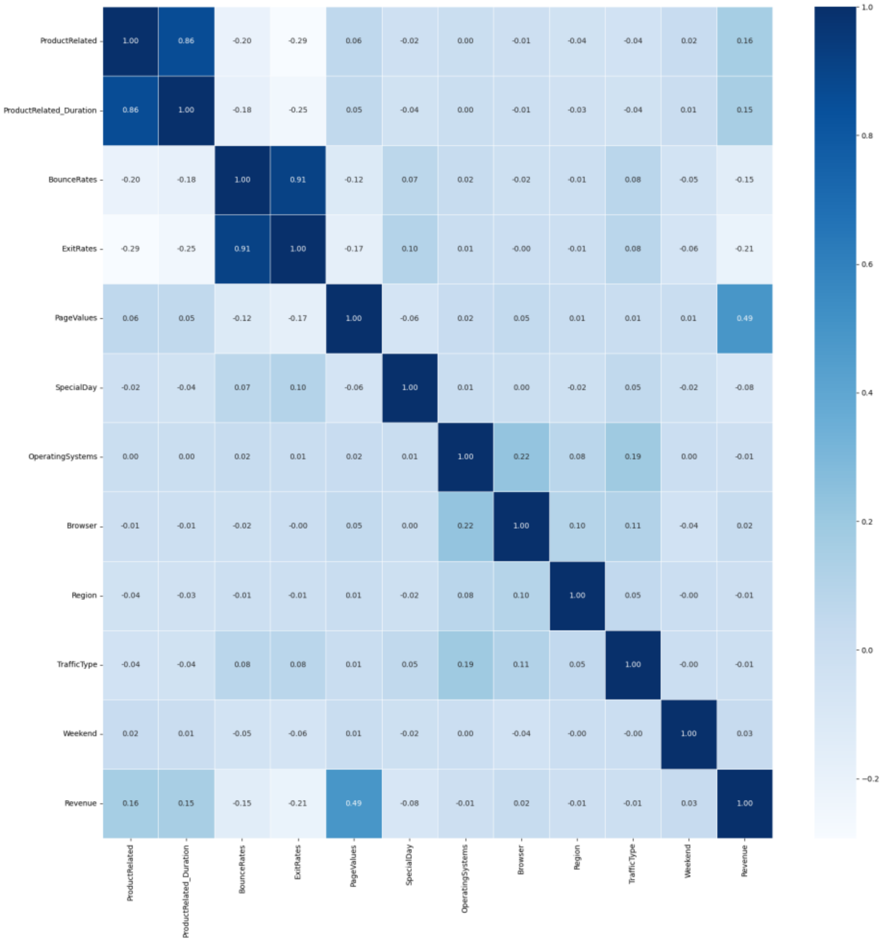
✓ 0.0s

```
# Covariance Matrix
covariance_df = pd.DataFrame(data=np.cov(normalized_dataset, bias=True, rowvar=False),
columns=dataset.columns)
```

✓ 0.0s

```
# Plot Covariance Matrix
plt.subplots(figsize=(20, 20))
sns.heatmap(covariance_df, cmap='Blues', linewidths=.7, annot=True, fmt='.2f',
yticklabels=dataset.columns)
plt.show()
```

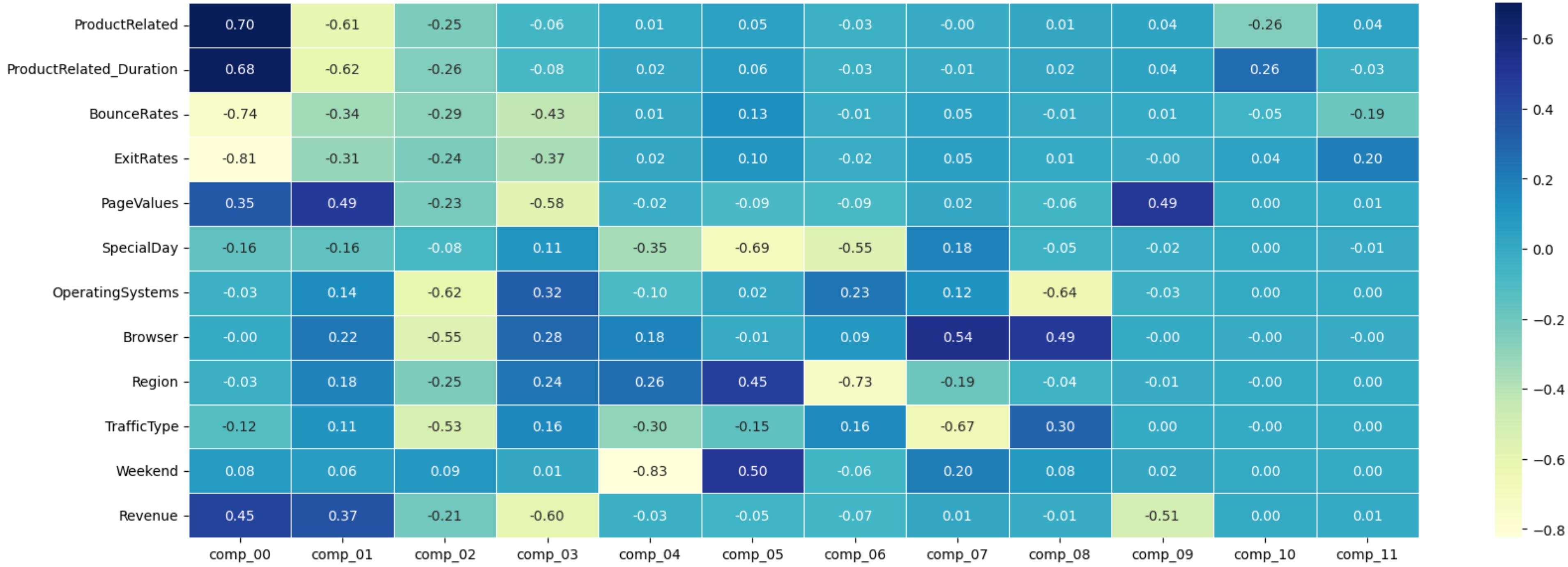
✓ 0.4s


ProductRelated 1.00 0.86 -0.20 -0.29 0.06 -0.02 0.00 -0.01 -0.04 -0.04 0.02 0.16
ProductRelated_Duration 0.86 1.00 -0.18 -0.25 0.05 -0.04 0.00 -0.01 -0.03 -0.04 0.01 0.15
BounceRates -0.20 -0.18 1.00 0.91 -0.12 0.07 0.02 -0.02 -0.01 0.08 -0.05 -0.15
ExitRates -0.29 -0.25 0.91 1.00 -0.17 0.10 0.01 -0.00 -0.01 0.08 -0.06 -0.21
PageValues 0.06 0.05 -0.12 -0.17 1.00 -0.06 0.02 0.05 0.01 0.01 0.01 0.49
SpecialDay -0.02 -0.04 0.07 0.10 -0.06 1.00 0.01 0.00 -0.02 0.05 -0.02 -0.08
OperatingSystems 0.00 0.00 0.02 0.01 0.02 0.01 1.00 0.22 0.08 0.19 0.00 -0.01
Browser -0.01 -0.01 -0.02 -0.00 0.05 0.00 0.22 1.00 0.10 0.11 -0.04 0.02
Region -0.04 -0.03 -0.01 -0.01 0.01 -0.02 0.08 0.10 1.00 0.05 -0.00 -0.01
TrafficType -0.04 -0.04 0.08 0.08 0.01 0.05 0.19 0.11 0.05 1.00 -0.00 -0.01
Weekend 0.02 0.01 -0.05 -0.06 0.01 -0.02 0.00 -0.04 -0.00 -0.00 1.00 0.03
Revenue 0.16 0.15 -0.15 -0.21 0.49 -0.08 -0.01 0.02 -0.01 -0.01 0.03 1.00
ProductRelated ProductRelated_Duration BounceRates ExitRates PageValues SpecialDay OperatingSystems Browser Region TrafficType Weekend Revenue
1.0 0.8 0.6 0.4 0.2 0.0 −0.2


ProductRelated 1.00 0.86 -0.20 -0.29
ProductRelated_Duration 0.86 1.00 -0.18 -0.25
BounceRates -0.20 -0.18 1.00 0.91
ExitRates -0.29 -0.25 0.91 1.00

```python
components_df = pca.factors
combined_df = pd.concat([dataset, components_df], axis=1)
correlation = combined_df.corr()
# This matrix will have the correlation between:
# We're removing part of the output to keep only the correlation between features and principal c
correlation_plot_data = correlation[:-len(components_df.columns)].loc[:, 'comp_00':]

# plot correlation matrix
fig, ax = plt.subplots(figsize=(20, 7))
sns.heatmap(correlation_plot_data, cmap='YlGnBu', linewidths=.7, annot=True, fmt='.2f')
plt.show()
```

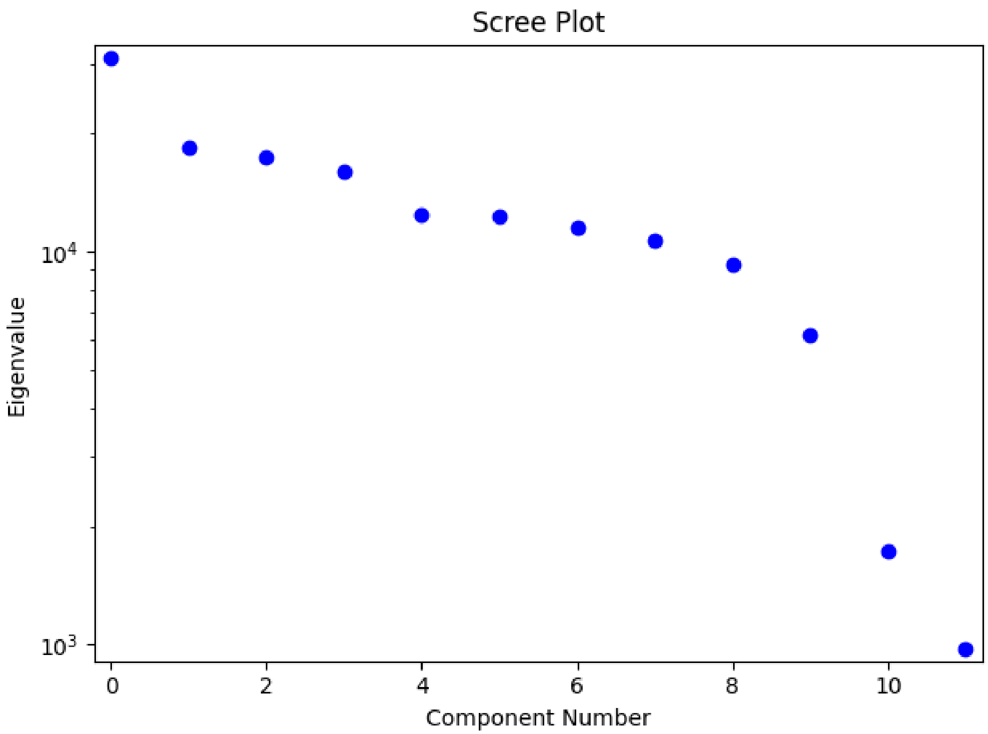
correlation matrix



```python
pca = PCA(dataset, standardize=True, method='eig')
eigen_values = pd.DataFrame(data=pca.eigenvals.values, columns=['eigenvalue'])
print(eigen_values)
```

[37] ✓ 0.0s Open 'eigen_values' in Data Wrangler

```
      eigenvalue
0   31240.652358
1   18353.544889
2   17445.149119
3   15949.459728
4   12378.968807
5   12282.187981
6   11564.604285
7   10640.461155
8    9262.959522
9    6139.765529
10   1729.047207
11    973.199421
```



Scree Plot
Eigenvalue
10^4
10^3
0
2
4
6
8
10
Component Number

Etapas de Implementação do PCA

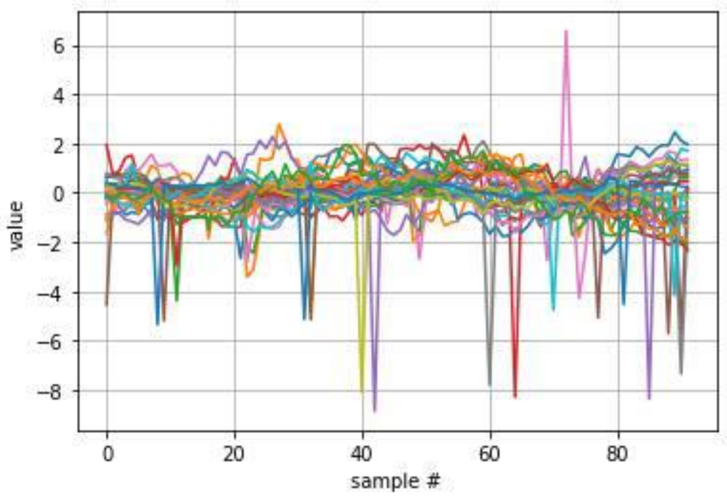
- Normalizar dados (média zero, variância unitária)
- Calcular matriz de covariância
- Realizar decomposição de autovalores
- Selecionar número de componentes a reter
- Transformar dados para espaço CP
- Reconstruir dados se necessário

Aplicações do PCA

- Monitoramento de processo
- Detecção de falhas
- Visualização de dados
- Redução de dimensionalidade
- Regressão por Componentes Principais (PCR)
- Reconhecimento de padrões em dados de processo

Redução de dimensionalidade para processo de fabricação de polímeros

Dados de uma fábrica de polímeros. O conjunto de dados contém 33 variáveis e 92 amostras horárias. da planta).



O processo começou a se comportar de forma anormal por volta da amostra 70 e, eventualmente, teve que ser encerrado.
Portanto, usamos as amostras 1 a 69 para treinar o modelo de ACP usando o código abaixo.
O restante dos dados será utilizado para ilustrar o monitoramento do processo posteriormente.

```
# import requisite libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns
```

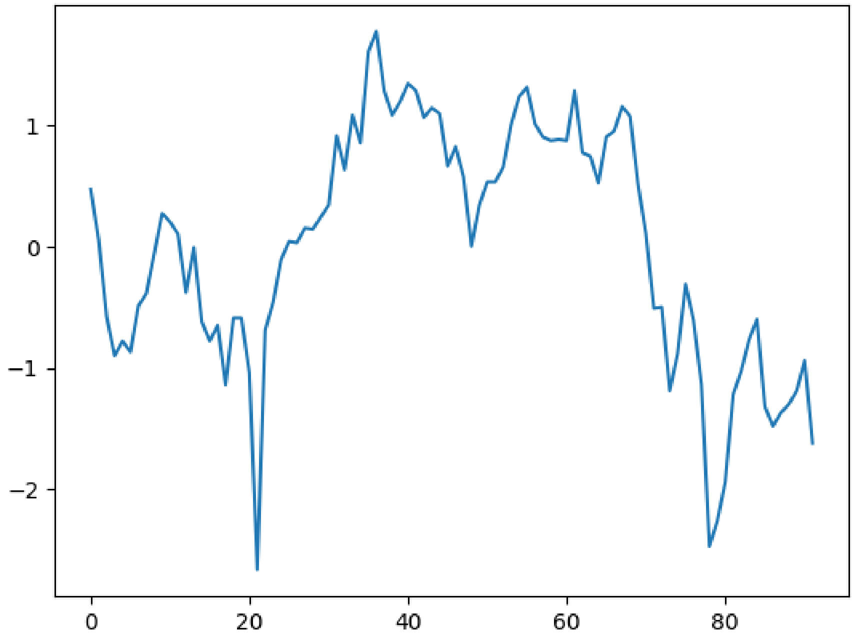
```
# fetch data and separate training data
data = pd.read_excel('proc1a.xls', skiprows = 1, usecols = 'C:AI')
data_train = data.iloc[0:69,]
```

```
data.info()
```

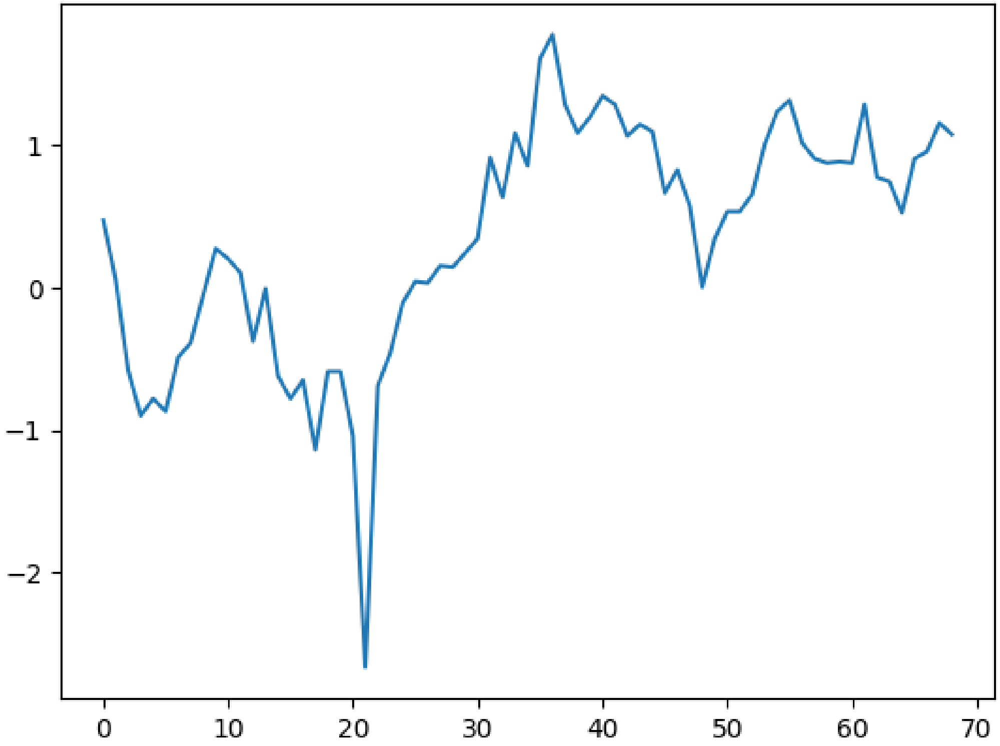
```
...   <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 92 entries, 0 to 91
      Data columns (total 33 columns):
       #   Column  Non-Null Count  Dtype  
      ---  ------  --------------  -----  
       0   x1in    92 non-null     float64
       1   x2in    92 non-null     float64
       2   x3in    92 non-null     float64
       3   x4in    92 non-null     float64
       4   x5in    92 non-null     float64
       5   x6in    92 non-null     float64
       6   x7in    92 non-null     float64
       7   y1      92 non-null     float64
       8   y2      92 non-null     float64
       9   y3      92 non-null     float64
       10  y4      92 non-null     float64
       11  y5      92 non-null     float64
       12  y6      92 non-null     float64
       13  y7      92 non-null     float64
       14  y8      92 non-null     float64
       15  x8md    92 non-null     float64
       16  x9md    92 non-null     float64
       17  xamd    92 non-null     float64
       18  xbmd    92 non-null     float64
       19  xcmd    92 non-null     float64
      ...
       31  xoen    92 non-null     float64
       32  xpen    92 non-null     float64
```

[17]

```
import matplotlib.pyplot as plt
plt.plot(data.index, data["x1in"])
```



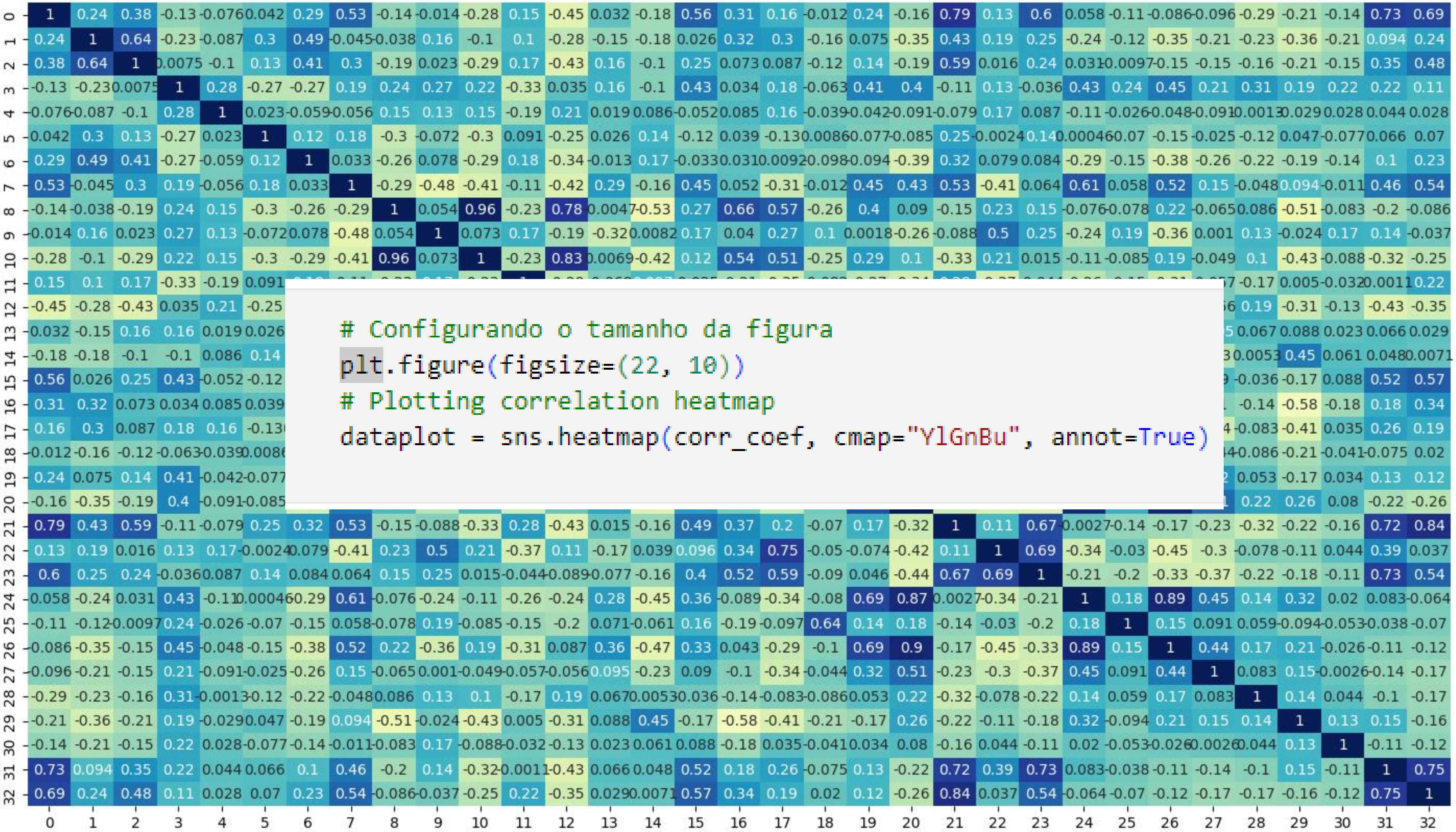
```
import matplotlib.pyplot as plt
plt.plot(data_train.index, data_train["x1in"])
```



```python
# normalize data
scaler = StandardScaler()
data_train_normal = scaler.fit_transform(data_train)
```

```python
# confirm correlation
corr_coef = np.corrcoef(data_train_normal, rowvar = False)
print('Correlation matrix: \n', corr_coef[0:3,0:3]) # printing only a portion
```

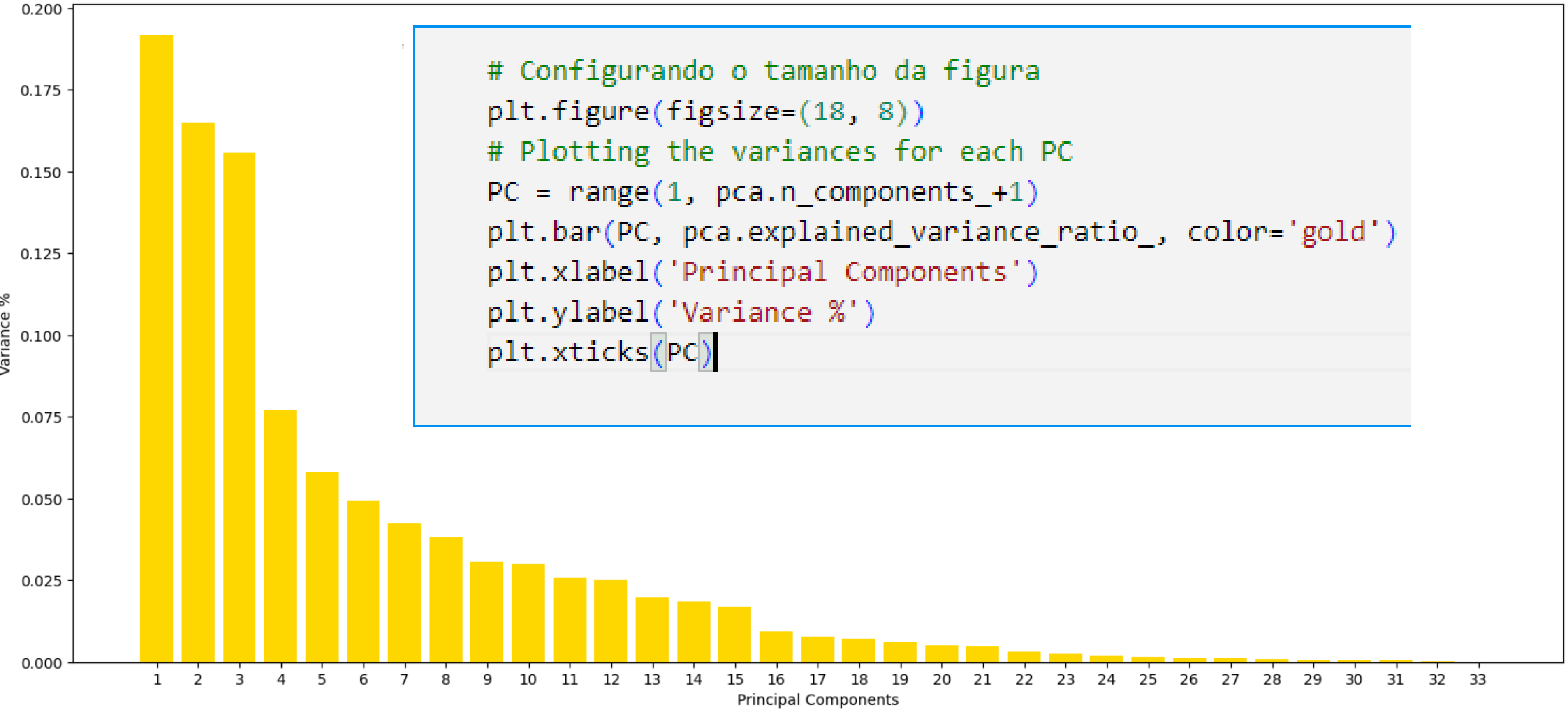
```
Correlation matrix:
 [[1.         0.23697456 0.38232242]
 [0.23697456 1.         0.64229595]
 [0.38232242 0.64229595 1.        ]]
```



# Configurando o tamanho da figura
plt.figure(figsize=(22, 10))
# Plotting correlation heatmap
dataplot = sns.heatmap(corr_coef, cmap="YlGnBu", annot=True)

```python
# PCA
pca = PCA()
score_train = pca.fit_transform(data_train_normal)
print(score_train)
#print(pca.explained_variance_ratio_)
```

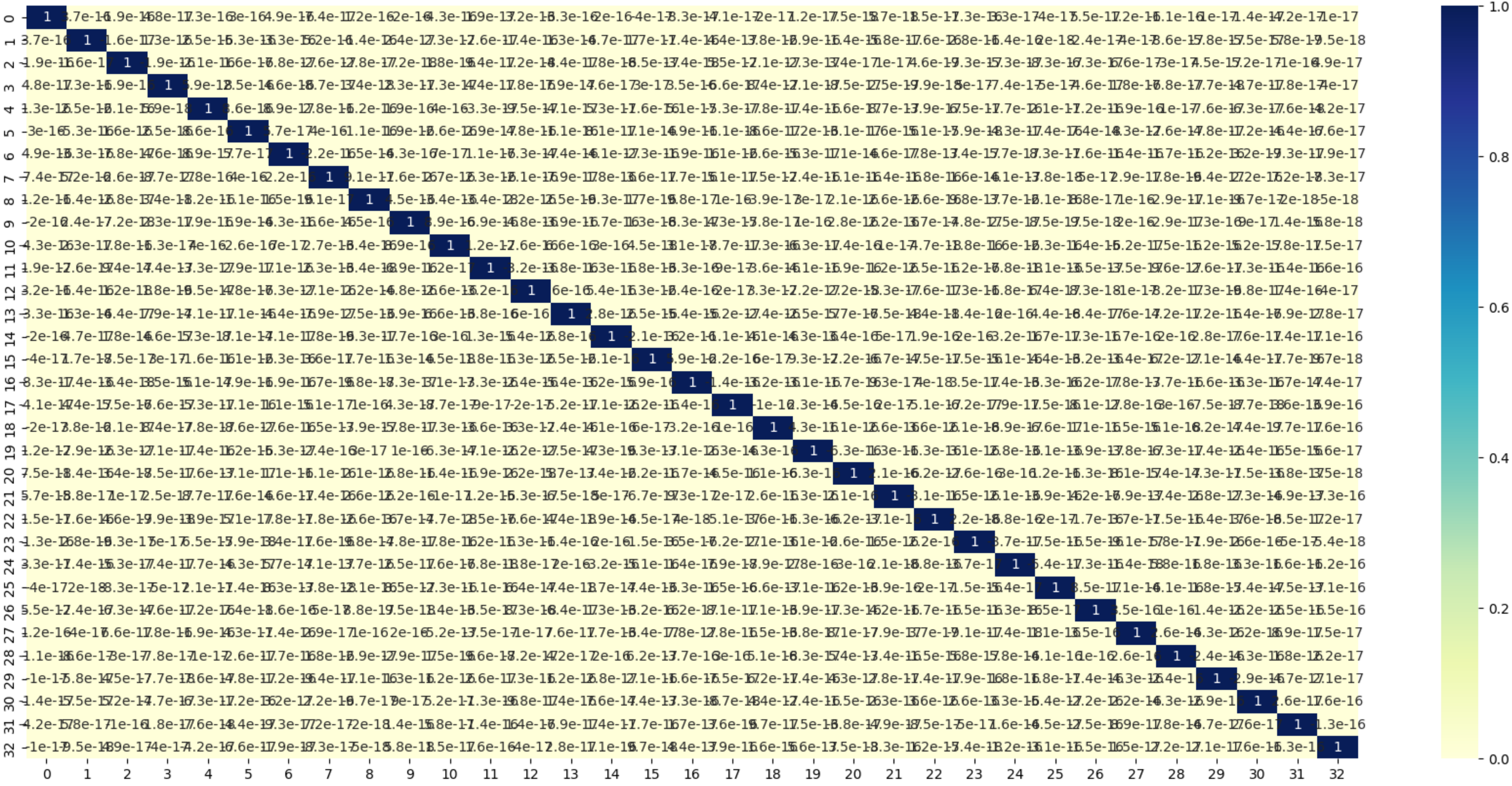
```
[[-3.13567818e+00  6.98524669e-01  4.21512436e+00 ...  6.55633603e-03
  -1.63656288e-02 -4.44619796e-04]
 [-2.41859631e+00  2.03755715e-01  3.19052028e+00 ...  2.19162255e-01
  -6.65076407e-02  2.42339528e-03]
 [-1.84872816e+00  3.21696864e-01  3.74358250e+00 ...  1.21199952e-01
   9.14625408e-02 -2.85745514e-03]
 ...
 [ 2.11264293e+00 -2.37124334e+00 -8.24050777e-02 ... -1.14989382e-01
   9.33187839e-02  2.70253860e-03]
 [ 1.98584160e+00 -2.27975777e+00  2.24309407e-01 ...  7.81995031e-02
   1.05277838e-01 -2.85030836e-04]
 [ 1.52808935e+00 -2.93512763e+00  8.17495002e-02 ... -2.47544074e-02
  -1.46165630e-02 -8.51068689e-04]]
```



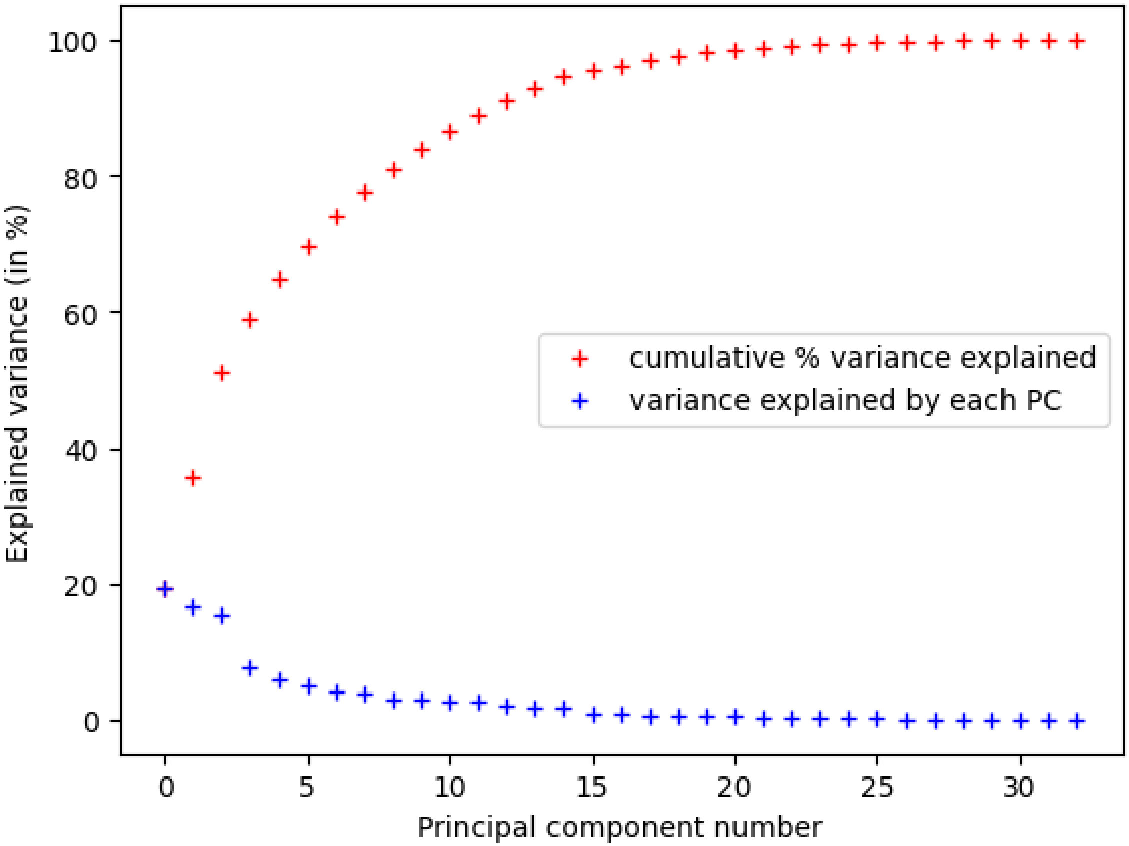

```python
# Configurando o tamanho da figura
plt.figure(figsize=(18, 8))
# Plotting the variances for each PC
PC = range(1, pca.n_components_+1)
plt.bar(PC, pca.explained_variance_ratio_, color='gold')
plt.xlabel('Principal Components')
plt.ylabel('Variance %')
plt.xticks(PC)
```

```python
# confirm no correlation
corr_coef = np.corrcoef(score_train, rowvar = False)
print('Correlation matrix: \n', corr_coef[0:3,0:3]) # printing only a portion

# Configurando o tamanho da figura
plt.figure(figsize=(22, 10))
# Plotting correlation heatmap
dataplot = sns.heatmap(corr_coef, cmap="YlGnBu", annot=True)
```

```
Correlation matrix:
 [[ 1.00000000e+00  3.71544006e-16 -1.93566117e-16]
 [ 3.71544006e-16  1.00000000e+00 -1.63473731e-17]
 [-1.93566117e-16 -1.63473731e-17  1.00000000e+00]]
```


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
1.0
0.8
0.6
0.4
0.2
0.0

```
# visualize explained variance
import matplotlib.pyplot as plt
explained_variance = 100*pca.explained_variance_ratio_ # in percentage
cum_explained_variance = np.cumsum(explained_variance) # cumulative % variance explained
plt.figure()
plt.plot(cum_explained_variance, 'r+', label = 'cumulative % variance explained')
plt.plot(explained_variance, 'b+', label = 'variance explained by each PC')
plt.ylabel('Explained variance (in %)'), plt.xlabel('Principal component number'), plt.legend()
```



cumulative % variance explained
variance explained by each PC
Explained variance (in %)
Principal component number
0
20
40
60
80
100
0
5
10
15
20
25
30

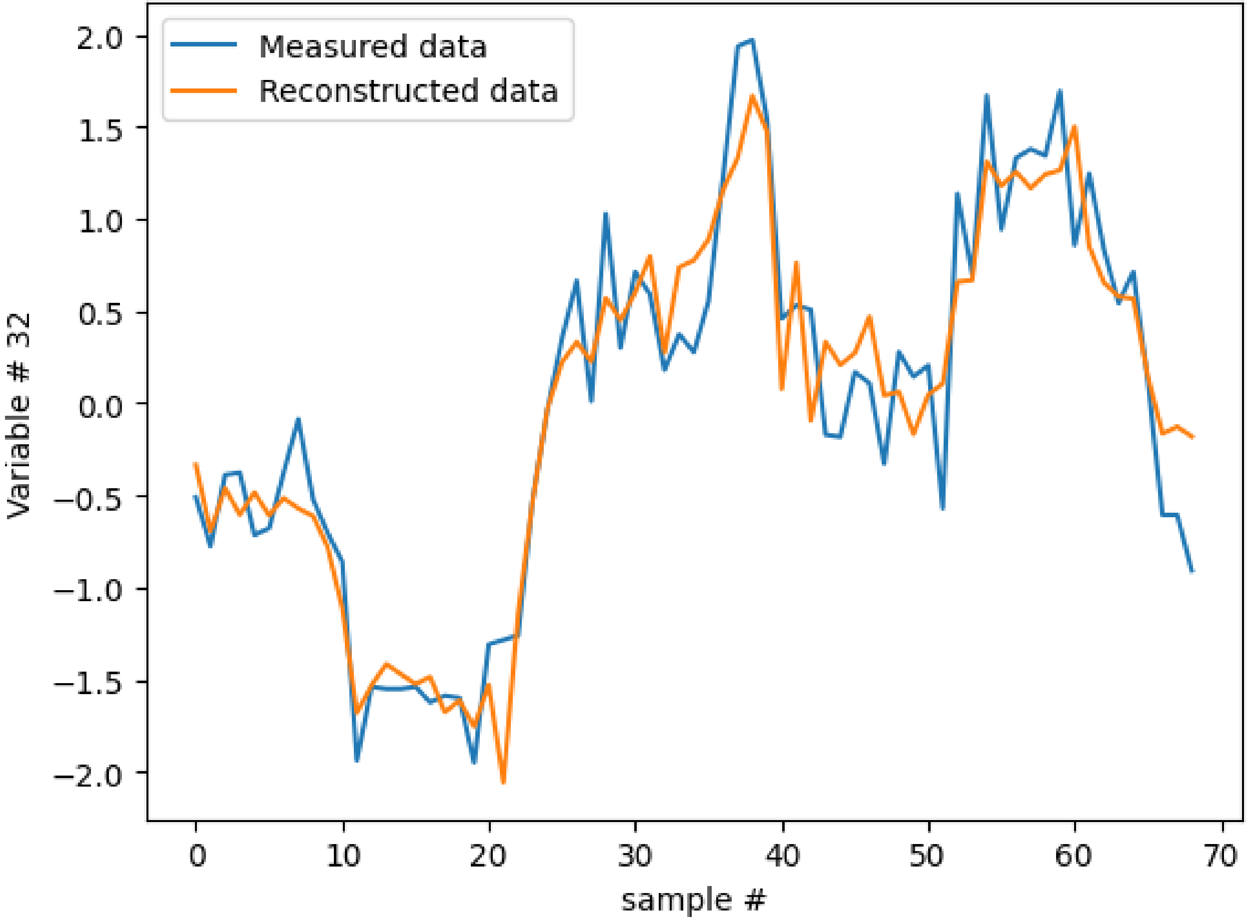
```
# decide # of PCs to retain and compute reduced data in PC space
n_comp = np.argmax(cum_explained_variance >= 90) + 1
score_train_reduced = score_train[:,0:n_comp]
print('Number of PCs cumulatively explaining atleast 90% variance: ', n_comp)
```

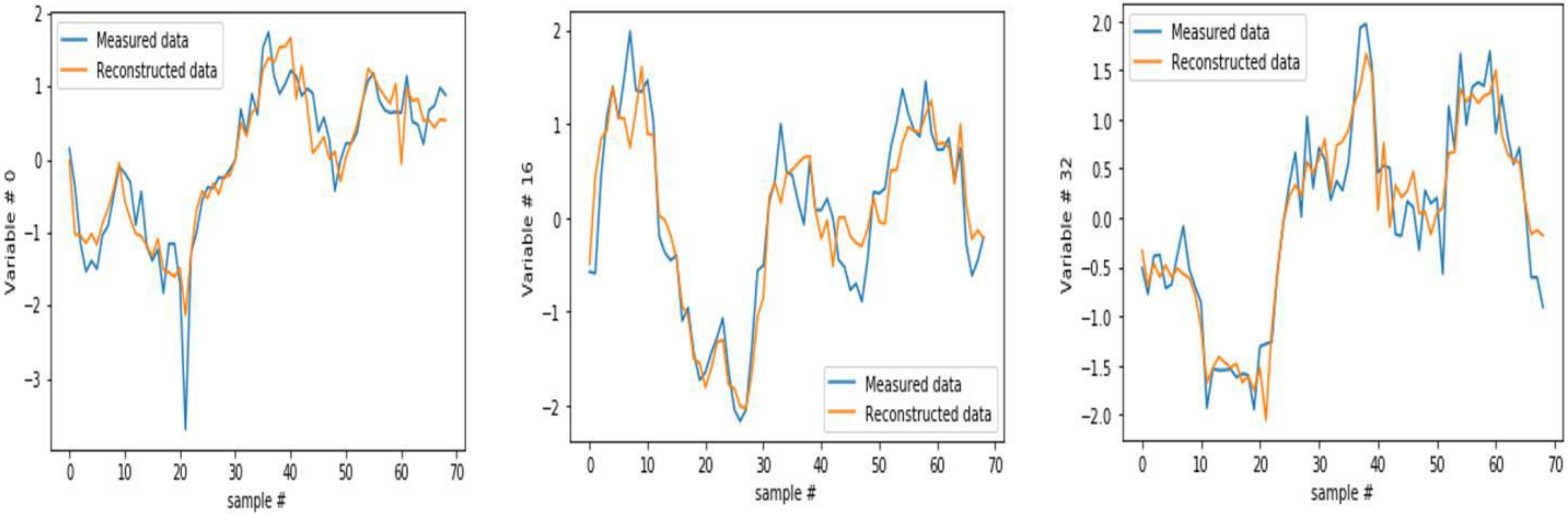
```
Number of PCs cumulatively explaining atleast 90% variance:  13
```

```
# confirm that only about 10% of original information is lost
from sklearn.metrics import r2_score
V_matrix = pca.components_.T
P_matrix = V_matrix[:,0:n_comp]
data_train_normal_reconstruct = np.dot(score_train_reduced, P_matrix.T)
R2_score = r2_score(data_train_normal, data_train_normal_reconstruct)
print('% information lost = ', 100*(1-R2_score))
```

```
% information lost =  9.046972754471994
```

```
# plot to compare original and reconstructed variables
var = 32
plt.figure()
plt.plot(data_train_normal[:,var],label = 'Measured data')
plt.plot(data_train_normal_reconstruct[:,var],label = 'Reconstructed data')
plt.ylabel('Variable # '+ str(var))
plt.xlabel('sample #')
plt.legend()
plt.show()
```

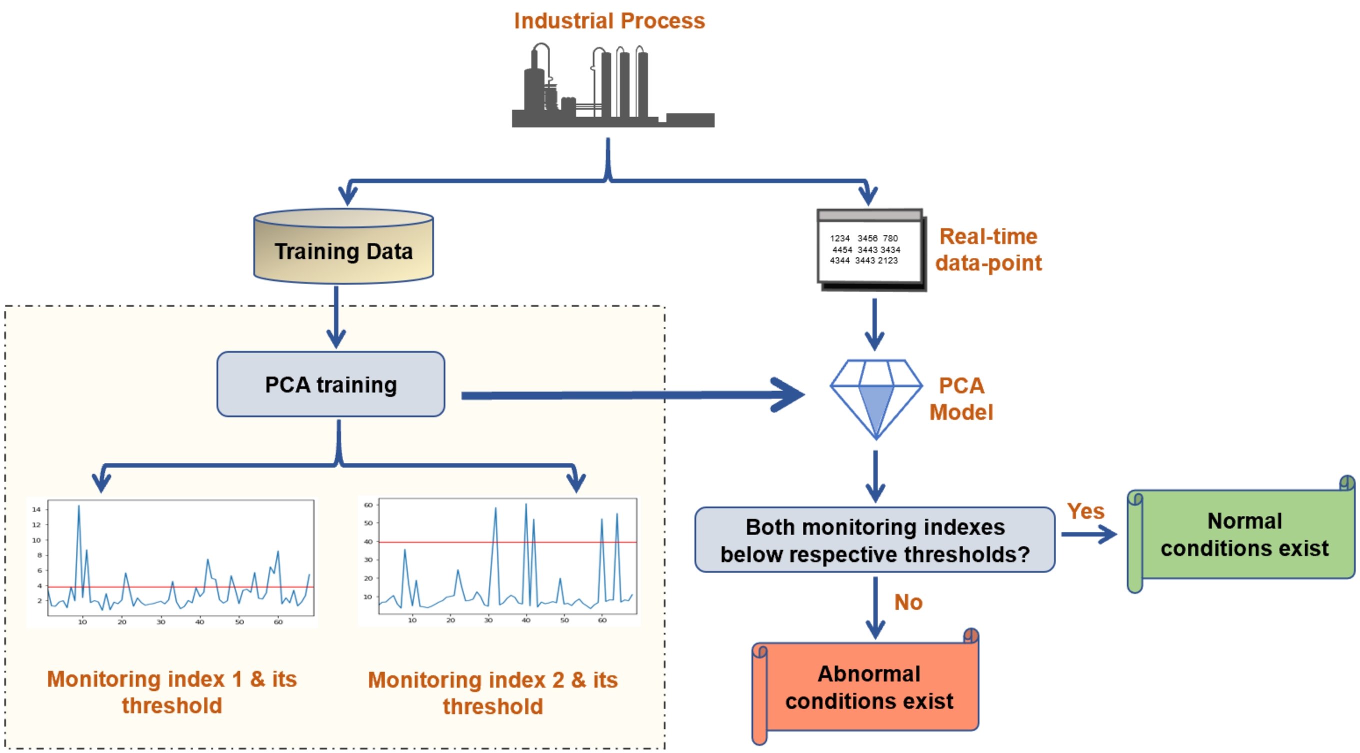

Measured data
Reconstructed data
Variable # 32
sample #
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
10
20
30
40
50
60
70



Measured data
Reconstructed data
Variable # 0
sample #
Variable # 16
sample #
Variable # 32
sample #

5.2 Monitoramento de Processo via PCA para Processo de Fabricação de Polímeros

Monitoramento de Processo de Fabricação de Polimero via PCA



Monitoramento de Processo via PCA

- Dois índices principais de monitoramento:
  - Estatística T^2 de Hotelling
  - Estatística SPE (Q) (Erro de predição quadrática)
- Limiares estatísticos determinam condições anormais
- Ambos índices necessários para monitoramento abrangente

$$T^2 = \sum_{j=1}^{k} \frac{t_{i,j}^2}{\lambda_j} = \boldsymbol{t}_i \boldsymbol{\Lambda}_k^{-1} \boldsymbol{t}_i^T \qquad \text{eq. 5}$$

$$\boldsymbol{\Lambda_k} = diag\{\lambda_1, \lambda_2, \ldots, \lambda_k\}.$$

$$Q = \sum_{j=1}^{m} e_{i,j}^2$$

$$T^2 = \sum_{j=1}^{k} \frac{t_{i,j}^2}{\lambda_j} = \boldsymbol{t}_i \boldsymbol{\Lambda}_k^{-1} \boldsymbol{t}_i^T \qquad \text{eq. 5}$$

```
# calculate T² for training data
lambda_k = np.diag(pca.explained_variance_[0:n_comp]) # eigenvalue = explained variance
lambda_k_inv = np.linalg.inv(lambda_k)

T2_train = np.zeros((data_train_normal.shape[0],))
for i in range(data_train_normal.shape[0]):
    T2_train[i] = np.dot(np.dot(score_train_reduced[i,:],lambda_k_inv),score_train_reduced[i,:].T)

# calculate Q for training data
error_train = data_train_normal - data_train_normal_reconstruct
Q_train = np.sum(error_train*error_train, axis = 1)
```

$$Q = \sum_{j=1}^{m} e_{i,j}^2$$

$$\boldsymbol{\Lambda_k} = diag\{\lambda_1, \lambda_2, \ldots, \lambda_k\}.$$

$$T_{CL}^2 = \frac{k(N^2-1)}{N(N-k)} F_{k,N-k}(\alpha)$$

$F_{k,N-k}(\alpha)$ Percentil de distribuição F

$T^2 \leq T_{CL}^2$ Limite das fronteira de uma elipsoide

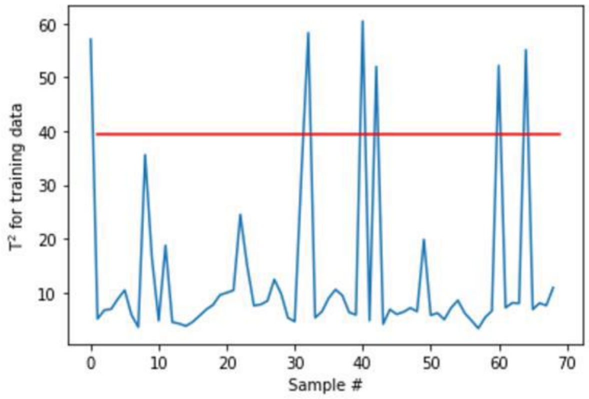
```
# T2 control limit
import scipy.stats

N = data_train_normal.shape[0]
k = n_comp

alpha = 0.01 # 99% control limit
T2_CL = k*(N**2-1)*scipy.stats.f.ppf(1-alpha,k,N-k)/(N*(N-k))
```

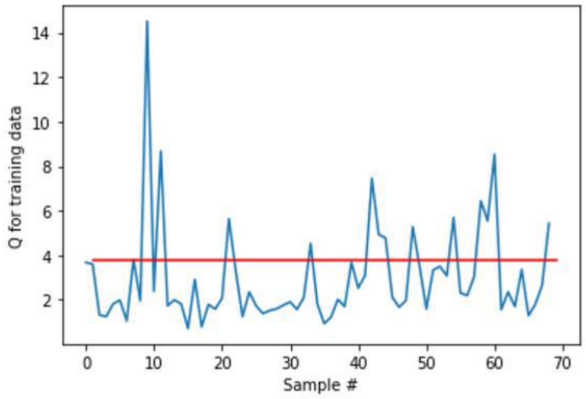
$$Q_{CL} = \theta_1 \left(\frac{z_\alpha\sqrt{2\theta_2 h_0^2}}{\theta_1} + 1 + \frac{\theta_2 h_0(1-h_0)}{\theta_1^2}\right)^2$$

$$h_0 = 1 - \frac{2\theta_1\theta_3}{3\theta_2^2} \text{ and } \theta_r = \sum_{j=k+1}^{m} \lambda_j^r \qquad ; r=1,2,3$$

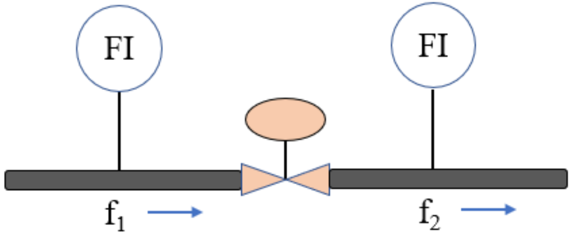
```
# Q control limit
eig_vals = pca.explained_variance_
m = data_train_normal.shape[1]

theta1 = np.sum(eig_vals[k:])
theta2 = np.sum([eig_vals[j]**2 for j in range(k,m)])
theta3 = np.sum([eig_vals[j]**3 for j in range(k,m)])
h0 = 1-2*theta1*theta3/(3*theta2**2)

z_alpha = scipy.stats.norm.ppf(1-alpha)
Q_CL = theta1*(z_alpha*np.sqrt(2*theta2*h0**2)/theta1+ 1 + theta2*h0*(1-h0)/theta1**2)**2
```

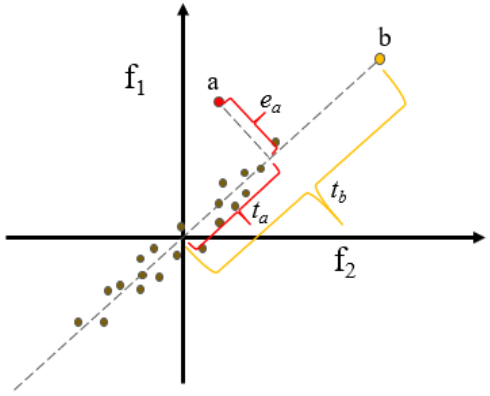
```
# Q_train plot with CL
plt.figure()
plt.plot(Q_train)
plt.plot([1,len(Q_train)],[Q_CL,Q_CL], color='red')
plt.xlabel('Sample #'), plt.ylabel('Q for training data')

# T2_train plot with CL
plt.figure()
plt.plot(T2_train)
plt.plot([1,len(T2_train)],[T2_CL,T2_CL], color='red')
plt.xlabel('Sample #'), plt.ylabel('T$^2$ for training data')
```





Importancia das estatísticas T^2 e Q





T^2 Mede a distancia dos dados em relação ao espaço PC

Q Mede a quantidade de dados que não são explicados pelo PC

Detecção de falhas

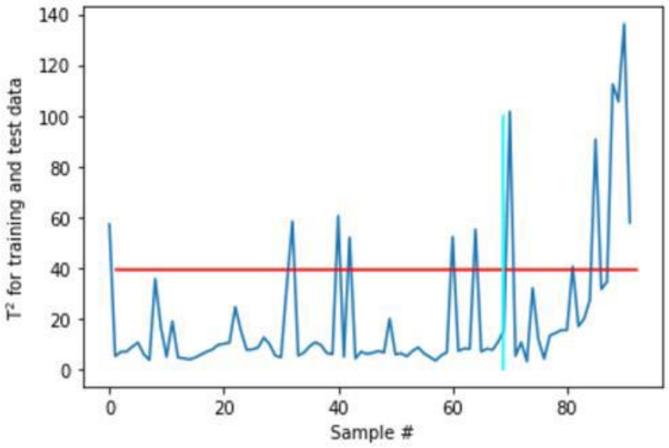
Verificar se nossos gráficos T2 e Q podem nos ajudar a detectar a presença de anormalidades de processo

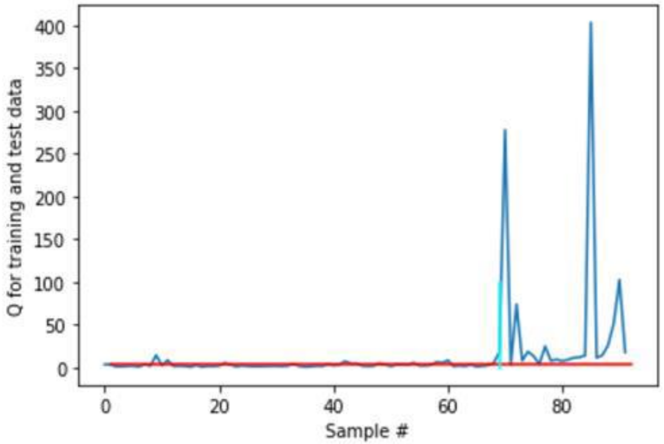
Em dados de teste (amostras 70 em diante).

Para isso, calcularemos as estatísticas de monitoramento para os dados de teste.

```
# get test data, normalize it
data_test = data.iloc[69:,]
data_test_normal = scaler.transform(data_test) # using scaling parameters from training data

# compute scores and reconstruct
score_test = pca.transform(data_test_normal)
score_test_reduced = score_test[:,0:n_comp]
data_test_normal_reconstruct = np.dot(score_test_reduced, P_matrix.T)
```

```
# calculate T2_test
T2_test = np.zeros((data_test_normal.shape[0],))
for i in range(data_test_normal.shape[0]): # eigenvalues from training data are used
    T2_test[i] = np.dot(np.dot(score_test_reduced[i,:],lambda_k_inv),score_test_reduced[i,:].T)

# calculate Q_test
error_test = data_test_normal_reconstruct - data_test_normal
Q_test = np.sum(error_test*error_test, axis = 1)
```





Diagnóstico de falhas

$$T^2 \text{ contribution of variable } j \;=\; j^{th} \text{ element of } \left(\boldsymbol{D}^{1/2}\boldsymbol{x}\right)^2$$

$$\boldsymbol{D} \;=\; \boldsymbol{P}\boldsymbol{\Lambda}_k^{-1}\boldsymbol{P}^T$$

```
# T² contribution
sample = 85 - 69
data_point = np.transpose(data_test_normal[sample-1,])

D = np.dot(np.dot(P_matrix,lambda_k_inv),P_matrix.T)
T2_contri = np.dot(scipy.linalg.sqrtm(D),data_point)**2 # vector of contributions

plt.figure()
plt.plot(T2_contri), plt.ylabel('T$^2$ contribution plot')
```

Diagnóstico de falhas

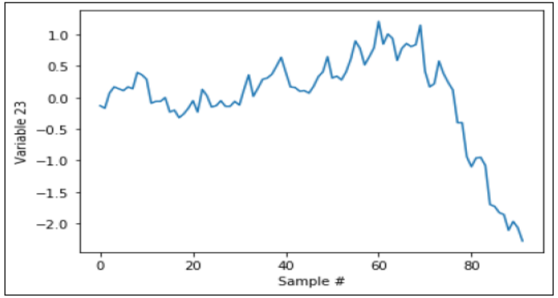
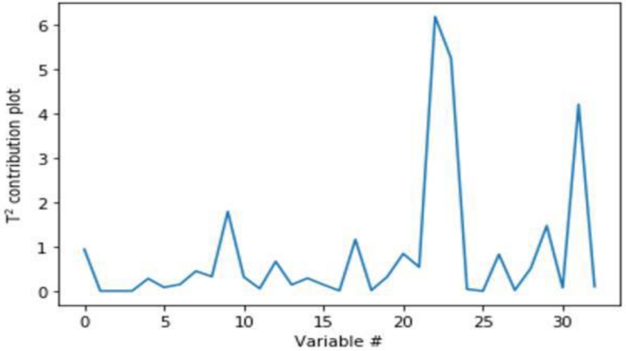
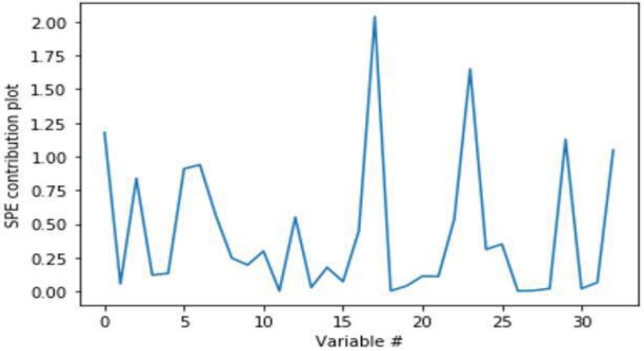


Figure 5.10: Temporal evolution of variable # 23

$$SPE = \sum_{j=1}^{m} e_j^2 = \sum_{j=1}^{m} SPE_j$$

```
# SPE contribution
error_test_sample = error_test[sample-1,]
SPE_contri = error_test_sample*error_test_sample # vector of contributions

plt.figure()
plt.plot(SPE_contri), plt.ylabel('SPE contribution plot')
```

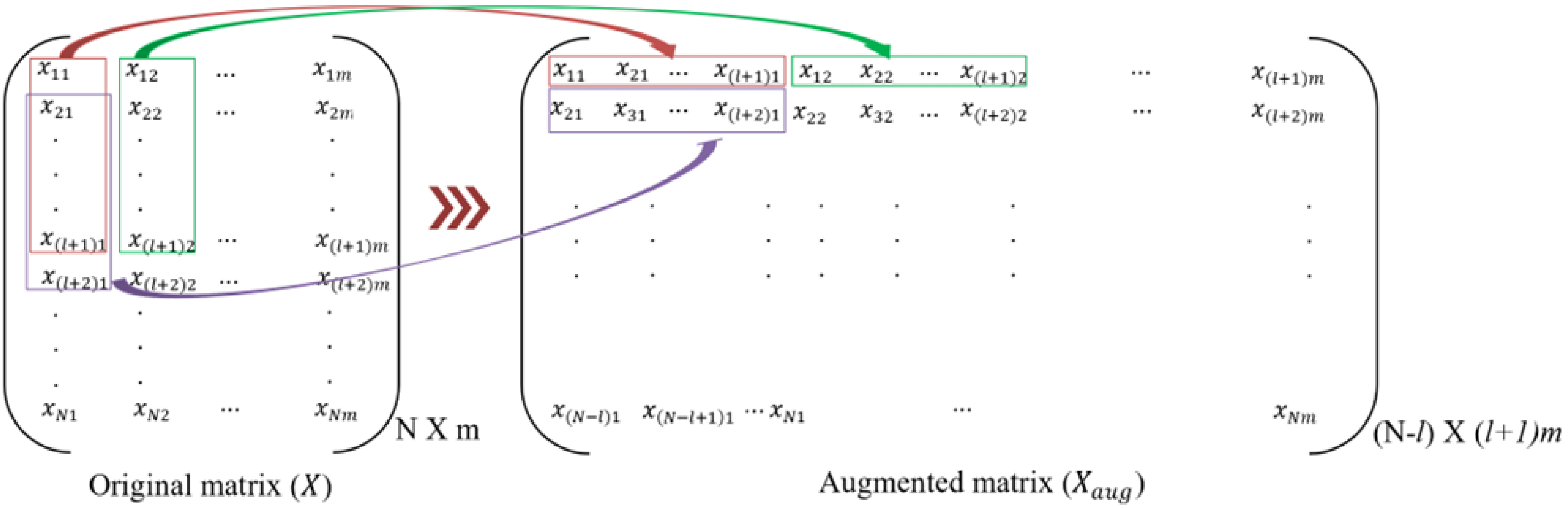




5.3 Variantes da PCA Clássica

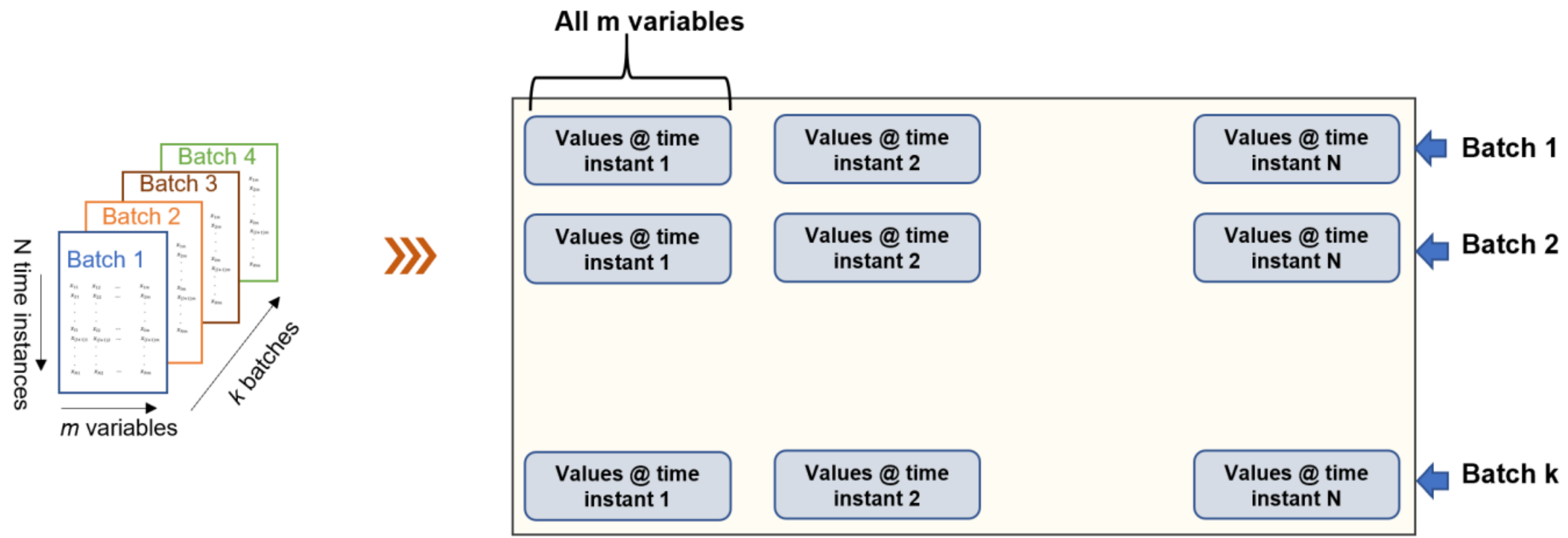
PCA Dinâmico (DPCA)

- Lida com dados autocorrelacionados
- Incorpora relações de séries temporais
- Aumenta matriz de dados com valores defasados
- Mais adequado para dados sequenciais temporais
- Captura relações dinâmicas no processo


N X m
(N-l) X (l+1)m
Original matrix (X)
Augmented matrix (X_{aug})

PCA Multiway

- Projetado para processos em lote
- Lida com dados tridimensionais
- Desdobra dados para duas dimensões
- Captura variabilidade entre lotes
- Útil em aplicações de manufatura


Batch 4
Batch 3
Batch 2
Batch 1
N time instances
m variables
k batches
All m variables
Values @ time instant 1
Values @ time instant 2
Values @ time instant N
Batch 1
Values @ time instant 1
Values @ time instant 2
Values @ time instant N
Batch 2
Values @ time instant 1
Values @ time instant 2
Values @ time instant N
Batch k

PCA Kernel

- Lida com relações não lineares
- Mapeia dados para espaço de maior dimensão
- Torna relações não lineares em lineares
- Usa truque do kernel para computação
- Efetivo para relações complexas de processo

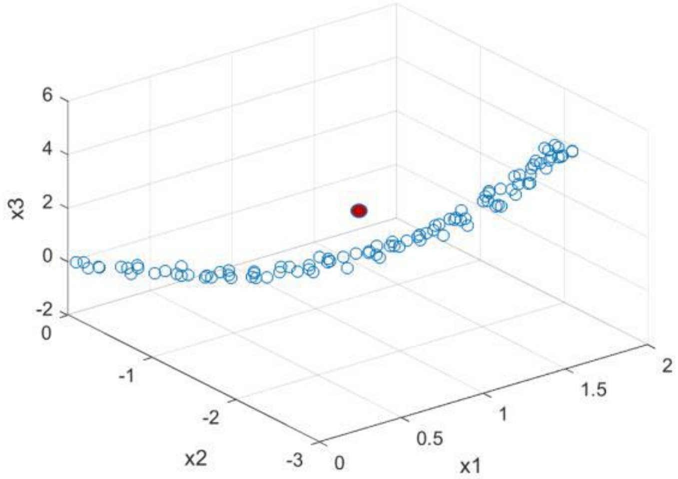
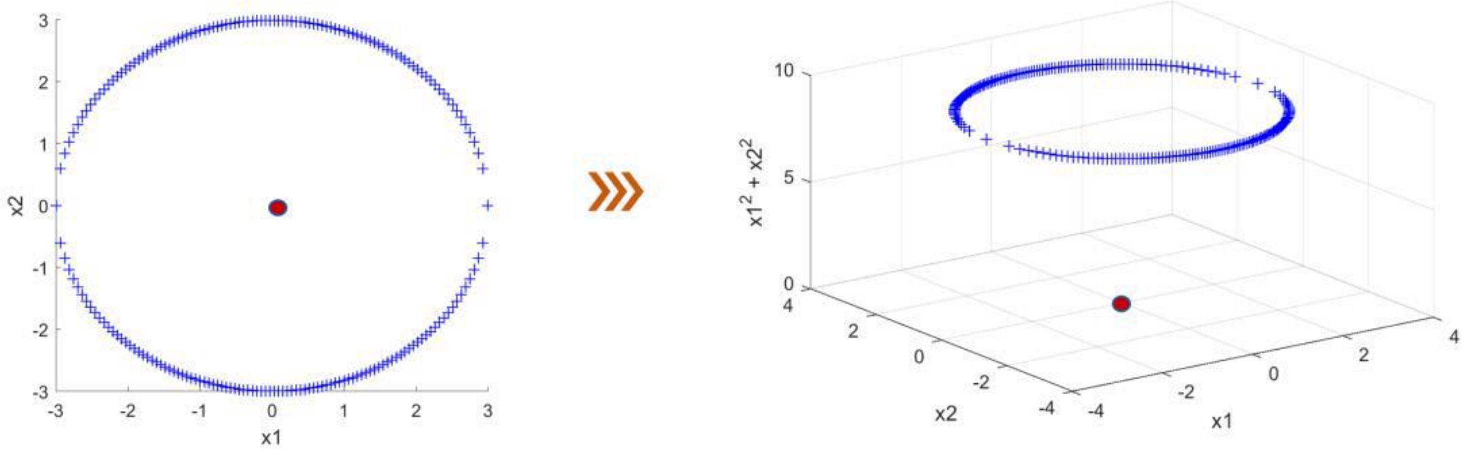


Figura 5.12: Dados não linearmente relacionados com um ponto de dados anormal (vermelho)

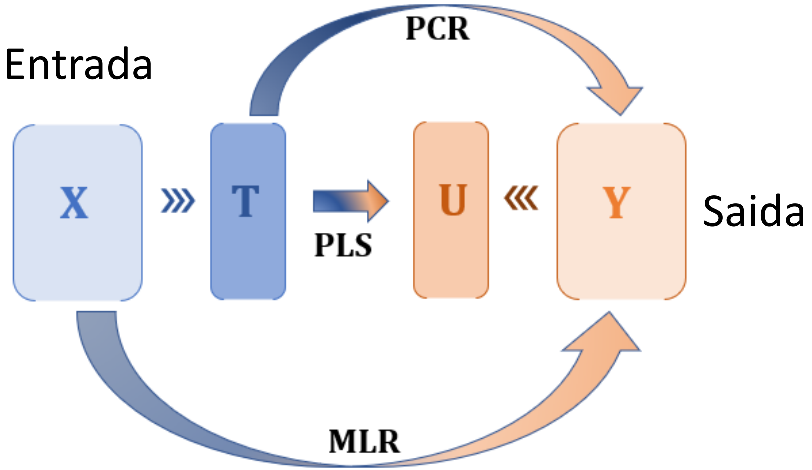


x2
3
2
1
0
-1
-2
-3
-3 -2 -1 0 1 2 3
x1
$x1^2 + x2^2$
10
5
0
4 2 0 -2 -4
x2
-4 -2 0 2 4
x1

Aplicações do PLS na Indústria

PCR:
Colinearidade,
Alta correlação
Ruidos de medição
Dataset reduzido

- Sensoriamento virtual
- Monitoramento de processo em tempo real
- Classificação de falhas
- Previsão de qualidade
- Controle de processo
- Otimização de desempenho



PLS: Minimo quadrado parcial

PCR: Regressão de componente principal

MLR: Regressão linear multivariavel

PCR: Variaveis latentes computadas
Independentes da saida

PLS Dinâmico

- Incorpora valores defasados no tempo
- Duas abordagens principais:
  - FIR (Resposta ao Impulso Finito)
  - ARX (Autoregressivo com variáveis Exógenas)
- Melhor para processos dependentes do tempo

Manutenção do Modelo

- Modelos degradam com o tempo devido a:
  - Envelhecimento do equipamento
  - Mudança nas condições do processo
- Duas abordagens de atualização:
  - Atualização recursiva
  - Atualização por janela móvel

Mínimos Quadrados Parciais (PLS): Introdução

- Técnica de regressão supervisionada
- Estima relações lineares entre:
  - Variáveis de entrada (X)
  - Variáveis de saída (Y)
- Transforma X e Y em componentes latentes
- Maximiza covariância entre entrada e saída

Formação em matemática

O PLS realiza 3 tarefas simultâneas:

- ➢ Captura a máxima variabilidade em X
- ➢ Captura a máxima variabilidade em Y
- ➢ Maximiza a correlação entre X e Y

Formação em matemática

Considere a matriz de dados: $\boldsymbol{X} \in \mathbb{R}^{N \times m}$

N observações de ***m*** variáveis de entrada

também tem uma matriz de dados de saída: $\boldsymbol{Y} \in \mathbb{R}^{N \times p}$

$p \ (\geq 1)$

Assume-se que cada coluna é normalizada para média zero e variância unitária em ambas as matrizes

Formação em matemática

As primeiras pontuações dos componentes latentes são dadas por

$$\boldsymbol{t}_1 = \mathbf{X}\boldsymbol{w}_1 \text{ and } \boldsymbol{u}_1 = \mathbf{Y}\boldsymbol{c}_1$$

Os vetores *w1* e *c1*, denominados vetores de peso, são calculados de forma que a covariância entre *t1* e *u1* seja maximizada

$$Cov(\boldsymbol{t}_1, \boldsymbol{u}_1) = Correlation(\boldsymbol{t}_1, \boldsymbol{u}_1) * \sqrt{Var(\boldsymbol{t}_1)} * \sqrt{Var(\boldsymbol{u}_1)}$$

Formação em matemática

Na próxima etapa, os vetores de carga, p1 e q1, são encontrados

$$\boldsymbol{X} = \boldsymbol{t}_1 \boldsymbol{p}_1^T + \boldsymbol{E}_1 \text{ and } \boldsymbol{Y} = \boldsymbol{u}_1 \boldsymbol{q}_1^T + \boldsymbol{F}_1$$

E e *F* são chamadas matrizes residuais e representam a parte de X e Y que ainda não foi capturada

Para encontrar as próximas pontuações dos componentes, as três etapas acima são repetidas com as matrizes *E1* e *F1* substituindo *X* e *Y*.

Formação em matemática

A decomposição final do PLS se parece com o seguinte

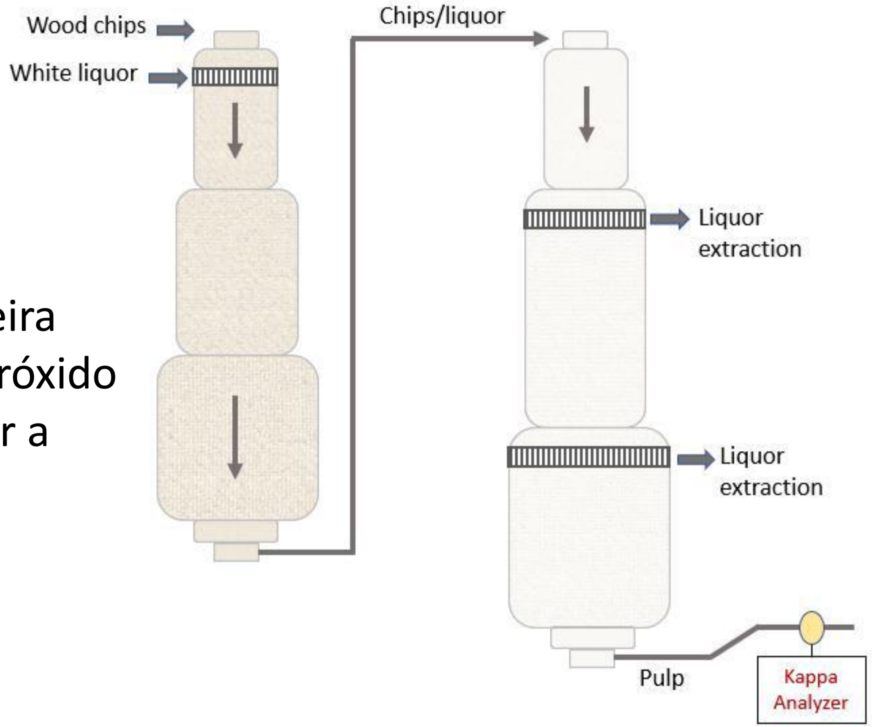
$$X = \mathbf{TP}^T + \mathbf{E} = \sum_{i=1}^{k} t_i p_i^T + E$$

$$Y = UQ^T + F = \sum_{i=1}^{k} u_i q_i^T + F$$

Soft Sensor via PLS para processo de fabricação de celulose e papel

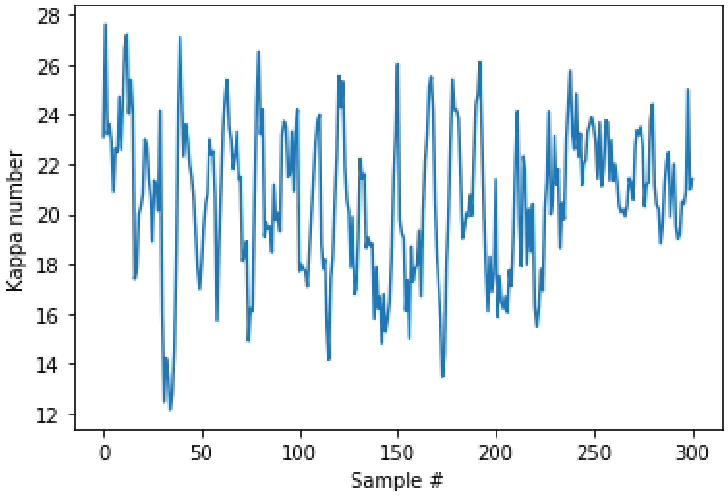
O conjunto de dados contém 301 amostras horárias de 21 variáveis de processo de um digestor Kamyr

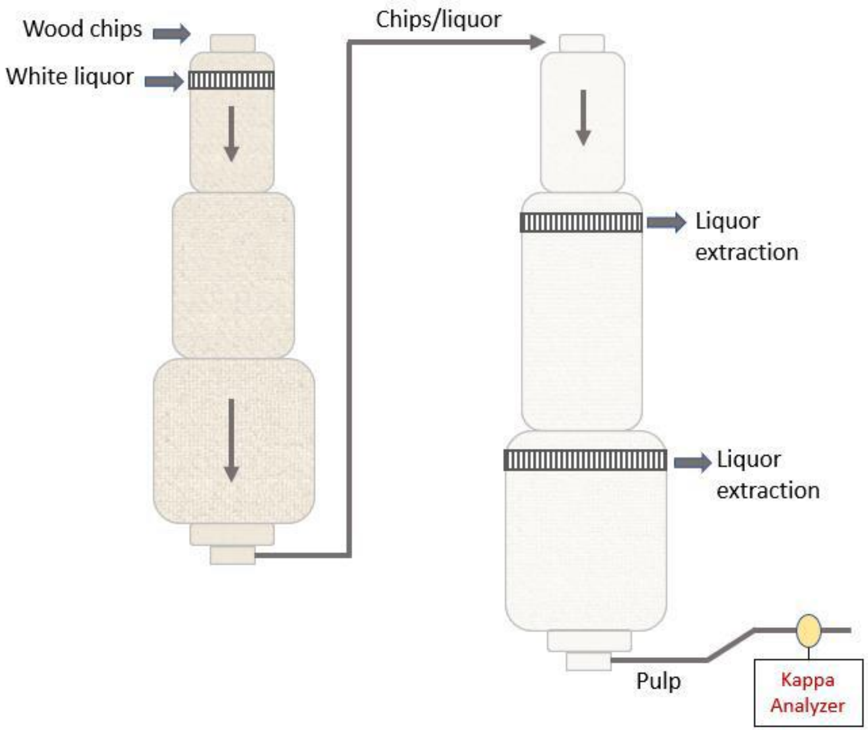
um reator tubular onde cavacos de madeira reagem com licor branco (solução de hidróxido de sódio e sulfeto de sódio) para remover a lignina das fibras de celulose



Soft Sensor via PLS para processo de fabricação de celulose e papel

O número kappa é a variável crítica de qualidade (fornecida no conjunto de dados) neste processo e quantifica o teor de lignina na polpa.





PLS vs Outros Métodos de Regressão

- Regressão Linear Multivariada (MLR)
    - Ajuste direto por mínimos quadrados
- Regressão por Componentes Principais (PCR)
    - PCA seguido de regressão
- Vantagens do PLS:
    - Lida com colinearidade
    - Considera X e Y na transformação

Estrutura Matemática do PLS

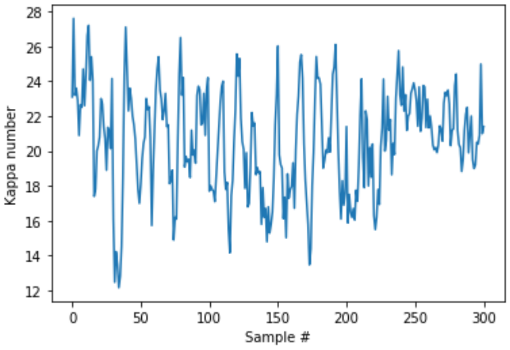
- Calcula matrizes de score T e U
- Maximiza covariância entre T e U
- Realiza três tarefas simultâneas:
  - Maximizar captura de variância X
  - Maximizar captura de variância Y
  - Maximizar correlação X-Y

Seleção do Número de Componentes

- Métodos incluem:
  - Limiar de variância explicada (90-95%)
  - Validação cruzada
  - Testes scree
  - Critério AIC
- Equilíbrio entre complexidade e precisão

Soft Sensing via PLS para processo de fabricação de celulose e papel

- Conjunto de dados ‘Kamyr digester’ de um processo de fabricação de celulose e papel.
- Cavacos de madeira são processados em celulose cuja qualidade é quantificada pelo número Kappa.
- No conjunto de dados, 301 amostras horárias do número Kappa e 21 outras variáveis de processo são fornecidas.

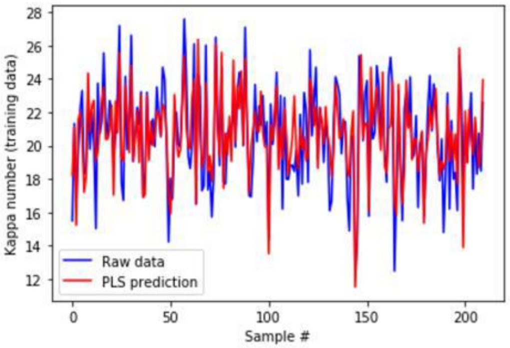


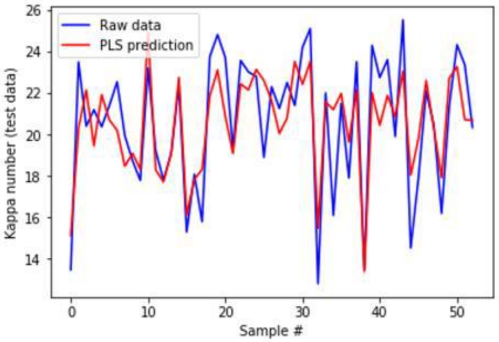
```
# import required packages
import numpy as np, pandas as pd
from sklearn.cross_decomposition import PLSRegression

# fetch data
data = pd.read_csv('kamyr-digester.csv', usecols = range(1,23))
# find the # of nan entries in each column
na_counts = data.isna().sum(axis = 0)

# remove columns that have a lot of nan entries
data_cleaned = data.drop(columns = ['AAWhiteSt-4 ','SulphidityL-4 '])

# remove any row that have any nan entry
data_cleaned = data_cleaned.dropna(axis = 0)

# separate X, y
y = data_cleaned.iloc[:,0].values[:, np.newaxis] # StandardScaler requires 2D array
X = data_cleaned.iloc[:,1:].values

print('Number of samples left: ', X.shape[0])
```

```python
# separate training and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)

# scale data
from sklearn.preprocessing import StandardScaler
X_scaler = StandardScaler()
X_train_normal, X_test_normal = X_scaler.fit_transform(X_train), X_scaler.transform(X_test)

y_scaler = StandardScaler()
y_train_normal, y_test_normal = y_scaler.fit_transform(y_train), y_scaler.transform(y_test)
```

```
# PLS model
pls = PLSRegression(n_components = 9)
pls.fit(X_train_normal, y_train_normal)

# Training vs Test accuracy
y_train_normal_predict = pls.predict(X_train_normal)
y_test_normal_predict = pls.predict(X_test_normal)

print('Accuracy over training data: ', pls.score(X_train_normal, y_train_normal))
print('Accuracy over test data: ', pls.score(X_test_normal, y_test_normal))
```

```
>>> Accuracy over training data: 0.6615
>>> Accuracy over test data: 0.6812
```





Numero de variáveis latentes

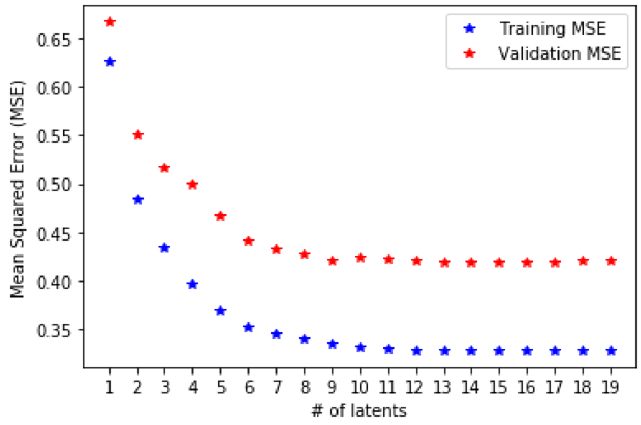
Usamos 9 componentes latentes em nosso modelo PLS.

Isso foi determinado por meio do procedimento de validação cruzada K-fold.

```
# import required packag
from sklearn.model_sele
from sklearn.metrics imp

scaler = StandardScaler()
fit_MSE = []
validate_MSE = []
```

```
kfold = KFold(n_splits = 10, shuffle = True, random_state = 100)
for fit_index, validate_index in kfold.split(y_train):

    X_fit_normal = scaler.fit_transform(X_train[fit_index])
    X_validate_normal = scaler.transform(X_train[validate_index])

    y_fit_normal = scaler.fit_transform(y_train[fit_index])
    y_validate_normal = scaler.transform(y_train[validate_index])

    pls = PLSRegression(n_components = n_comp)
    pls.fit(X_fit_normal, y_fit_normal)

    local_fit_MSE.append(mean_squared_error(y_fit_normal, pls.predict(X_fit_normal)))
    local_validate_MSE.append(mean_squared_error(y_validate_normal,
                                                 pls.predict(X_validate_normal)))
```

```
kfold = KFold(n_splits = 10, shuffle = True, random_state = 100)
for fit_index, validate_index in kfold.split(y_train):

    X_fit_normal = scaler.fit_transform(X_train[fit_index])
    X_validate_normal = scaler.transform(X_train[validate_index])

    y_fit_normal = scaler.fit_transform(y_train[fit_index])
    y_validate_normal = scaler.transform(y_train[validate_index])

    pls = PLSRegression(n_components = n_comp)
    pls.fit(X_fit_normal, y_fit_normal)

    local_fit_MSE.append(mean_squared_error(y_fit_normal, pls.predict(X_fit_normal)))
    local_validate_MSE.append(mean_squared_error(y_validate_normal,
                                                 pls.predict(X_validate_normal)))

fit_MSE.append(np.mean(local_fit_MSE))
validate_MSE.append(np.mean(local_validate_MSE))
```



Considerações de Implementação

- Requisitos de pré-processamento de dados
- Tratamento de valores ausentes
- Detecção de outliers
- Escalonamento e normalização
- Abordagens de validação do modelo

Melhores Práticas e Diretrizes

- Escolher modelo mais simples efetivo
- Atualizar modelos regularmente
- Validar suposições
- Monitorar desempenho do modelo
- Considerar conhecimento do processo
- Documentar parâmetros do modelo

Resumo e Direções Futuras

- Ferramentas poderosas para monitoramento de processo
- Essencial para dados de alta dimensão
- Desenvolvimentos contínuos em:
  - Métodos não lineares
  - Aplicações dinâmicas
  - Monitoramento em tempo real
  - Algoritmos adaptativos