Analise Componente Principal

November 22, 2022

1 PCA

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from IPython.display import Image
%matplotlib inline
```

1.1 Dados

Dados que apresentam uma grande quantidade de atributos.

[2]:
```
from sklearn.datasets import load_breast_cancer
```

[3]:
```
cancer = load_breast_cancer()
```

[4]:
```
cancer.keys()
```

[4]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

[5]:
```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
```

- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three worst/largest values) of these features were computed for each image, resulting in 30 features. For instance, field 0 is Mean Radius, field 10 is Radius SE, field 20 is Worst Radius.

- class:
    - WDBC-Malignant
    - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058

```
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
```

```
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages
570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning
techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77
(1994)
     163-171.
```

[6]:
```
df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
```

df.head()

1.2 Padronização dos Dados

[8]:
```
from sklearn.preprocessing import StandardScaler
```

[9]:
```
scalar = StandardScaler()
scalar.fit(df)
```

[9]:
```
StandardScaler()
```

[10]:
```
scalar_data = scalar.transform(df)
```

[13]:
```
scalar_data
```

[13]:
```
array([[ 1.09706398, -2.07333501,  1.26993369, …,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, …,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, …,  1.95500035,
         1.152255  ,  0.20139121],
       …,
       [ 0.70228425,  2.0455738 ,  0.67267578, …,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, …,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, …, -1.74506282,
        -0.04813821, -0.75120669]])
```

[14]:
```
from sklearn.decomposition import PCA
```

[15]:
```
pca = PCA(n_components=2)
```

[16]:
```
pca.fit(scalar_data)
```

[16]: PCA(n_components=2)

[18]:
```
x_pca = pca.transform(scalar_data)
```

[19]:
```
scalar_data.shape
```

[19]: (569, 30)

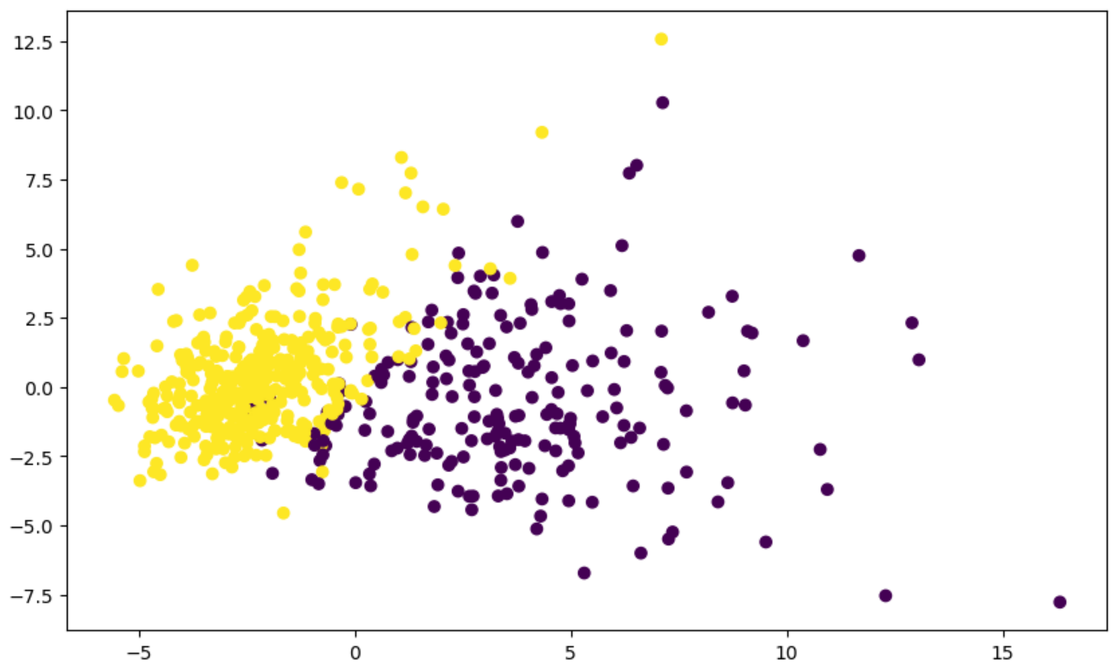
[20]:
```
x_pca.shape
```

[20]: (569, 2)

Reduzimos de 30 colunas para 2 colunas

[23]:
```
plt.figure(figsize=(10,6))
plt.scatter(x_pca[:,0], x_pca[:,1], c=cancer['target'])
```

[23]: <matplotlib.collections.PathCollection at 0x7fbaf19f11e0>



1.3 Interpretando os componente

[24]:
```
pca.components_
```

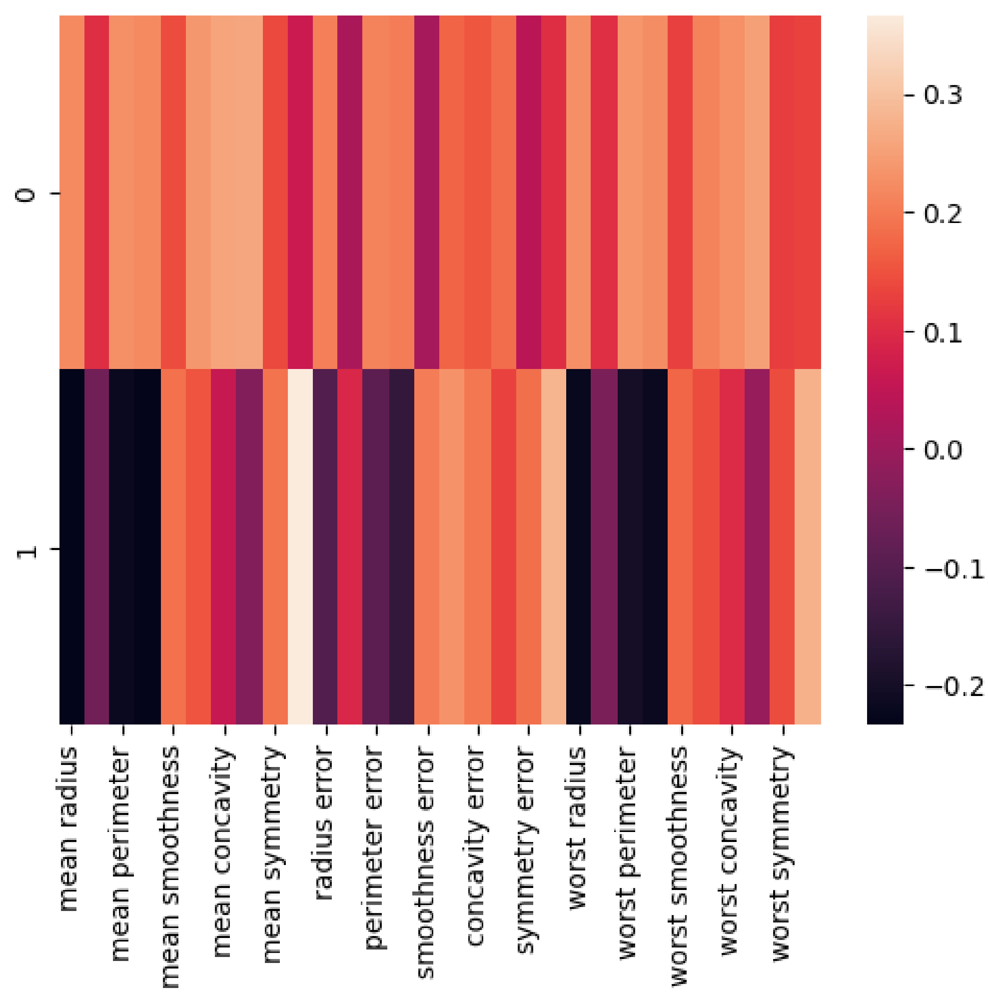
[24]:
```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
```

```
 0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
 0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
 0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
[-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611302,
 0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
-0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
 0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
-0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
 0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.27533947]])
```

[25]:
```
df_comp = pd.DataFrame(pca.components_, columns=cancer['feature_names'])
```

[26]:
```
sns.heatmap(df_comp)
```

[26]: `<AxesSubplot:>`



1.4 Um exemplo de PCA

[27]:

```
Image(filename='Matriz.png')
```

[27]:

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 1 & 3 \\ 2 & 4 \\ 2 & 3 \\ 1 & 4 \end{pmatrix} \quad y = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

[28]:

```
df = pd.DataFrame({'var1':[1,3,1,2,2,1], 'var2':[2,3,3,4,3,4], 'target':
    ↪[0,1,0,1,1,1]})
```

[29]:

```
df.head()
```

[29]:

	var1	var2	target
0	1	2	0
1	3	3	1
2	1	3	0
3	2	4	1
4	2	3	1

[30]:

```
X=df.drop('target', axis=1)
y=df['target']
```

[31]:

```
mean_vec = np.mean(X, axis=0)
print(mean_vec)
```

```
var1    1.666667
var2    3.166667
dtype: float64
```

[32]:
```
Image(filename='Matriz_2.png')
```

[32]:

$$M = \frac{1}{3}\begin{pmatrix} -2 & -4 \\ 4 & 2 \\ -2 & -1 \\ 1 & 2 \\ 1 & -1 \\ -2 & 2 \end{pmatrix}$$

[33]:
```
# Substituindo a média da respectiva coluna
M = X - mean_vec
```

[34]:
```
M
```

[34]:

	var1	var2
0	-0.666667	-1.166667
1	1.333333	-0.166667
2	-0.666667	-0.166667
3	0.333333	0.833333
4	0.333333	-0.166667
5	-0.666667	0.833333

[35]:
```
# calculando a matriz de covariancia

C = M.T.dot(M)/(X.shape[0]-1)
```

[36]:
```
C
```

[36]:

	var1	var2
var1	0.666667	0.066667
var2	0.066667	0.566667

1.5 Determinando os autovalores e autovetores

Matriz identidade I =10

01

[37]:
```python
autovalores, autovetores = np.linalg.eig(C)
```

[38]:
```python
autovalores
```

[38]:
```
array([0.7       , 0.53333333])
```

[39]:
```python
autovetores
```

[39]:
```
array([[ 0.89442719, -0.4472136 ],
       [ 0.4472136 ,  0.89442719]])
```

[40]:
```python
print(autovalores)
```

```
[0.7        0.53333333]
```

[41]:
```python
print(autovetores)
```

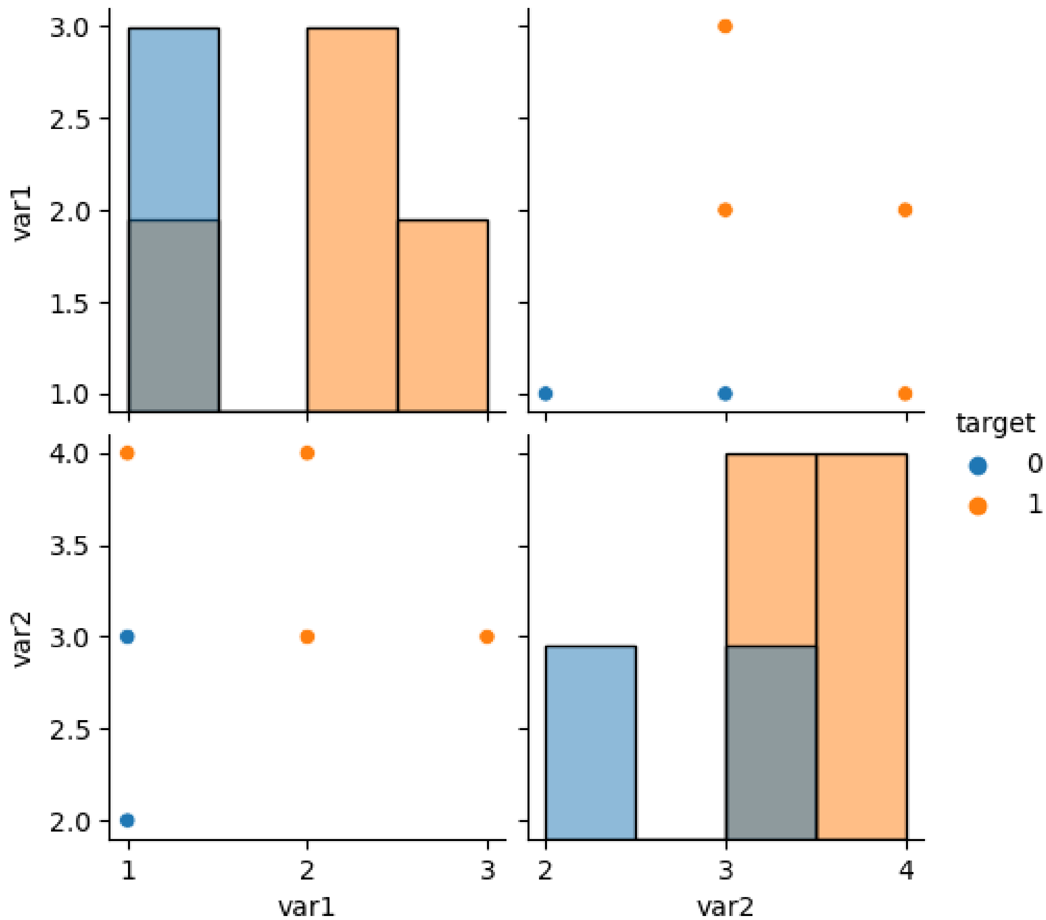
```
[[ 0.89442719 -0.4472136 ]
 [ 0.4472136   0.89442719]]
```

[44]:
```python
# ordenando em ordem descrecente
pares_de_autos = [
    (
        np.abs(autovalores[i]),
        autovetores[:,i]
    )for i in range(len(autovalores))
]
pares_de_autos.sort()
pares_de_autos.reverse()
```

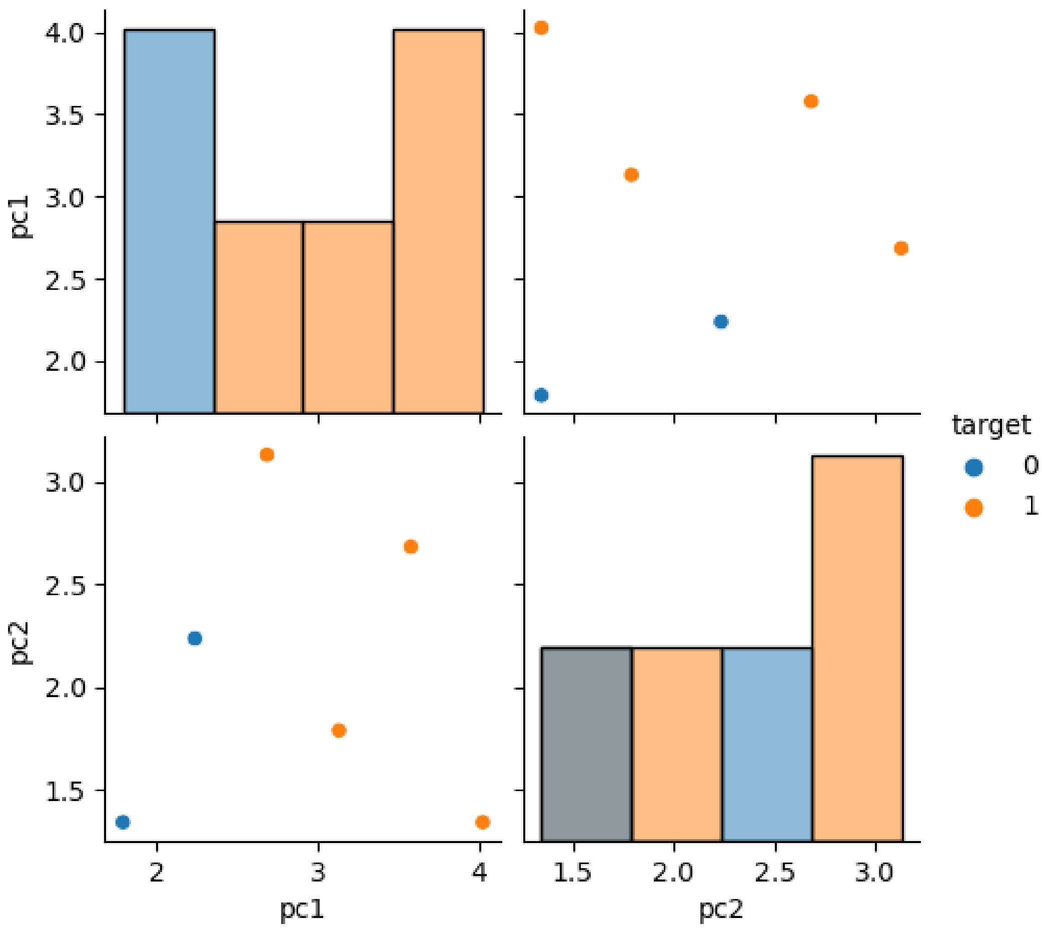
[45]:
```python
# calculando a variância e a variância acumulada
total = sum(autovalores)

var_exp = [
    (i/total)*100 for i in sorted(
        autovalores, reverse=True
    )
]
cum_var_exp=np.cumsum(var_exp)
```

[46]:
```python
# Visualizando os dados graficamente
sns.pairplot(
    df, vars=['var1', 'var2'], hue='target', diag_kind='hist'
)
```

```
plt.show()
```



[48]:
```
n_componentes = 2
autovetores = [p[1] for p in pares_de_autos]
A = autovetores[0:n_componentes]
X = np.dot(X,np.array(A).T)
new_df = pd.DataFrame(X, columns=['pc1', 'pc2'])
new_df['target'] = df['target']
sns.pairplot(
    new_df, vars=['pc1', 'pc2'], hue='target', diag_kind='hist'
)
plt.show()
```

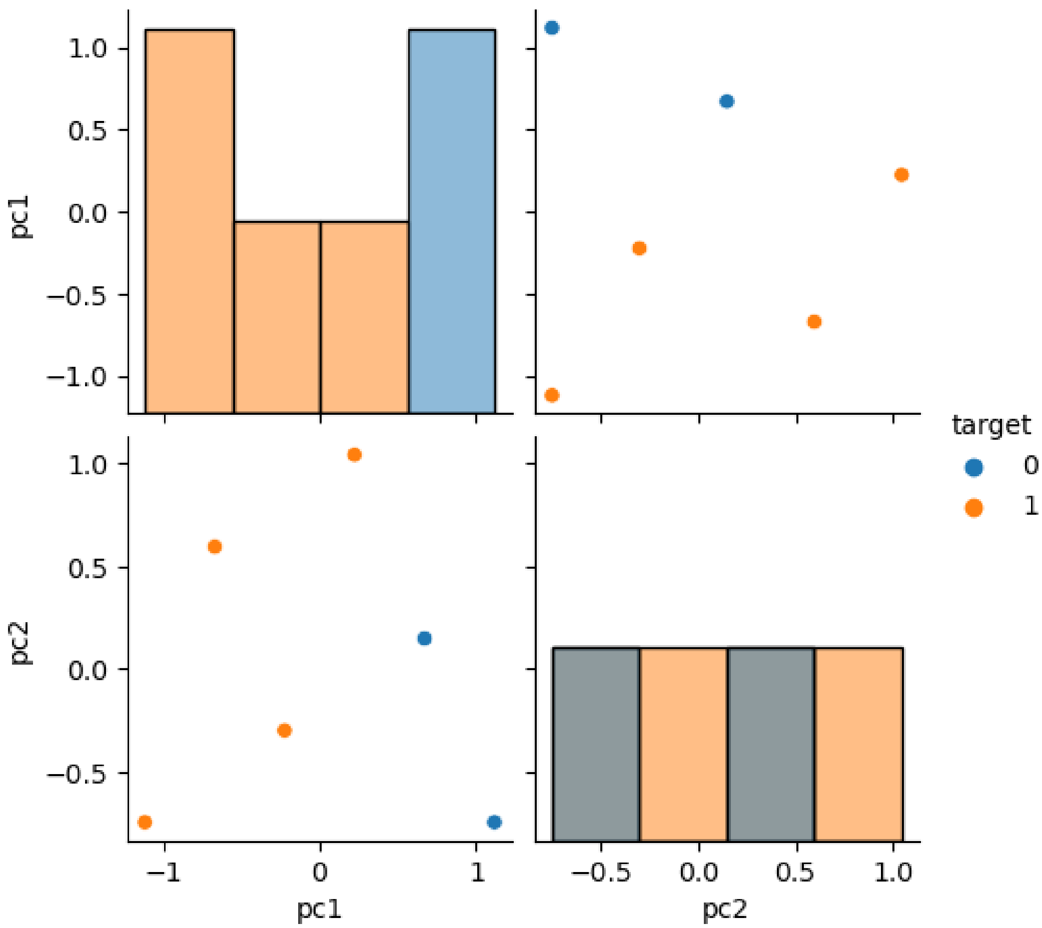


[49]:
```
from sklearn.decomposition import PCA
X = df.drop('target', axis=1)
y = df['target']

pca = PCA(n_components=2)
pca.fit(X)
```

[49]: PCA(n_components=2)

[50]:
```
X=pca.transform(X)
new_df2 = pd.DataFrame(X, columns=['pc1', 'pc2'])
new_df2['target'] = df['target']
sns.pairplot(
    new_df2, vars=['pc1', 'pc2'], hue='target', diag_kind='hist'
)
```

[50]: <seaborn.axisgrid.PairGrid at 0x7fbade967550>



1.6 PCA no conjunto de dados Iris

Link para baixar a base de dados

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

[54]:
```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
# carregando a base de dados
df = pd.read_csv(url, names=['sepal length in cm','sepal width in cm',
                             'petal length in cm','petal width in cm',␣
  ↪'target'])
```

[55]:
```
df.head()
```

[55]:
```
   sepal length in cm  sepal width in cm  petal length in cm  \
0                 5.1                3.5                 1.4
1                 4.9                3.0                 1.4
2                 4.7                3.2                 1.3
```

```
3                 4.6                3.1                 1.5
4                 5.0                3.6                 1.4

   petal width in cm       target
0                0.2  Iris-setosa
1                0.2  Iris-setosa
2                0.2  Iris-setosa
3                0.2  Iris-setosa
4                0.2  Iris-setosa
```

[56]:
```
X = df.drop('target', axis=1)
y = df['target']
```

[57]:
```
mean_vec = np.mean(X, axis=0)
M = X - mean_vec
C = M.T.dot(M)/(X.shape[0]-1)
autovalores, autovetores = np.linalg.eig(C)
pares_de_autos = [
    (
        np.abs(autovalores[i]),
        autovetores[:,i]
    )for i in range(len(autovalores))
]
pares_de_autos.sort()
pares_de_autos.reverse()
total = sum(autovalores)

var_exp = [
    (i/total)*100 for i in sorted(
        autovalores, reverse=True
    )
]
cum_var_exp=np.cumsum(var_exp)
```

[58]:
```
x = ['PC %s' %i for i in range(1, len(autovetores)+1)]
```

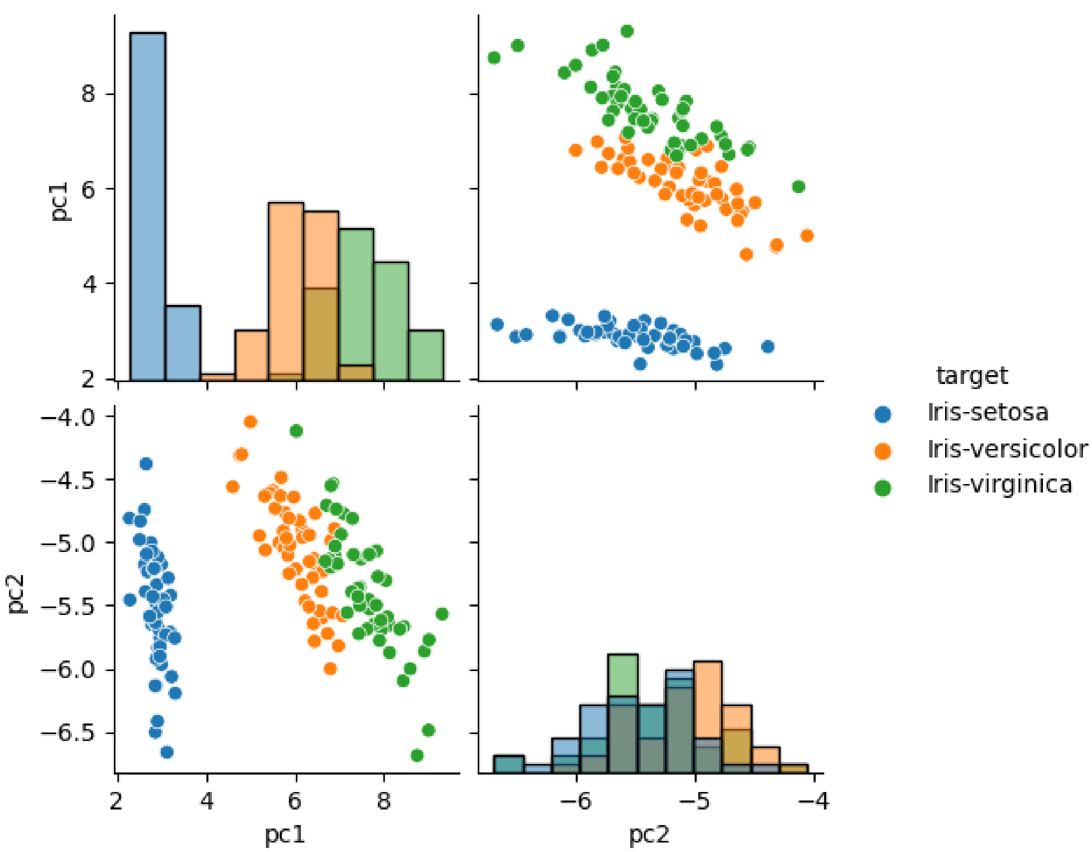
[59]:
```
df_temp = pd.DataFrame({'auto_valores': autovalores, 'cum_var_exp': cum_var_exp,
                       'var_exp':var_exp, 'Componente':x})
print(df_temp)
```

```
   auto_valores  cum_var_exp    var_exp Componente
0      4.224841    92.461621  92.461621       PC 1
1      0.242244    97.763178   5.301557       PC 2
2      0.078524    99.481691   1.718514       PC 3
3      0.023683   100.000000   0.518309       PC 4
```

[60]:
```
print('Auto-vetores')
for autovetor in [p[1] for p in pares_de_autos]:
    print(autovetor)
print()
```

```
Auto-vetores
[ 0.36158968 -0.08226889  0.85657211  0.35884393]
[-0.65653988 -0.72971237  0.1757674   0.07470647]
[-0.58099728  0.59641809  0.07252408  0.54906091]
[ 0.31725455 -0.32409435 -0.47971899  0.75112056]
```

[61]:
```
n_componentes = 2
autovetores = [p[1] for p in pares_de_autos]
A=autovetores[0:n_componentes]
X = np.dot(X,np.array(A).T)
new_df = pd.DataFrame(X, columns=['pc1', 'pc2'])
new_df['target'] = df['target']
sns.pairplot(
    new_df, vars=['pc1', 'pc2'], hue='target', diag_kind='hist'
)
plt.show()
```



[62]:
```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
# carregando a base de dados
df = pd.read_csv(url, names=['sepal length in cm','sepal width in cm',
                             'petal length in cm','petal width in cm',␣
 ↪'target'])
```

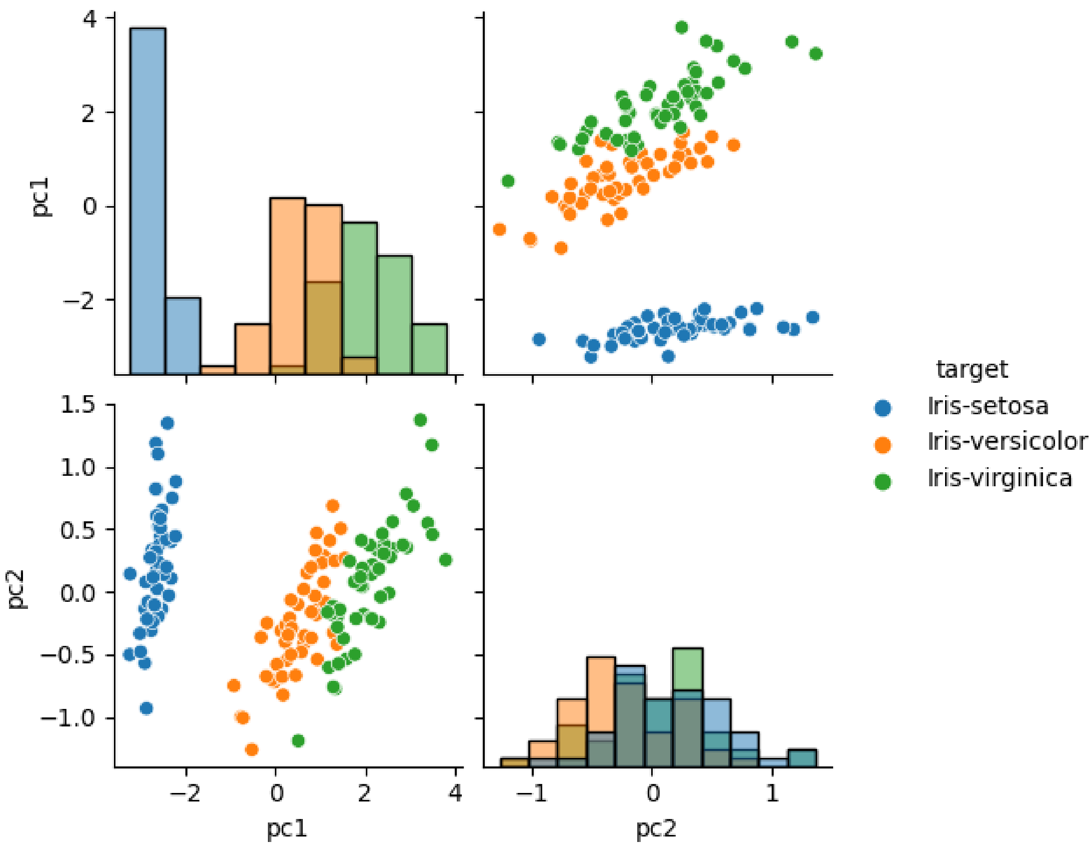
[63]:
```
X = df.drop('target', axis=1)
y = df['target']
```

[67]:
```
pca = PCA(n_components=2)
pca.fit(X)
```

[67]: PCA(n_components=2)

[66]:
```
X=pca.transform(X)
```

[68]:
```
new_df = pd.DataFrame(X, columns=['pc1', 'pc2'])
new_df['target'] = df['target']
sns.pairplot(
    new_df, vars=['pc1', 'pc2'], hue='target', diag_kind='hist'
)
plt.show()
```



1.7 PCA para redução de dimensões de imagens

[69]:
```
# carregando a imagem
X = plt.imread('bird.jpg')
X.shape
```

[69]: (1920, 1899)

[ ]:
```
# aplicando PCA  na imagem
pca = PCA(0.99) # variância 0.99
menor_dimensao = pca.fit_transform(X)
```

[71]:
```
menor_dimensao.shape
```

[71]: (1920, 145)

A variância representa que queremos que o numero de componentes garantia 99% de variância cumulativa

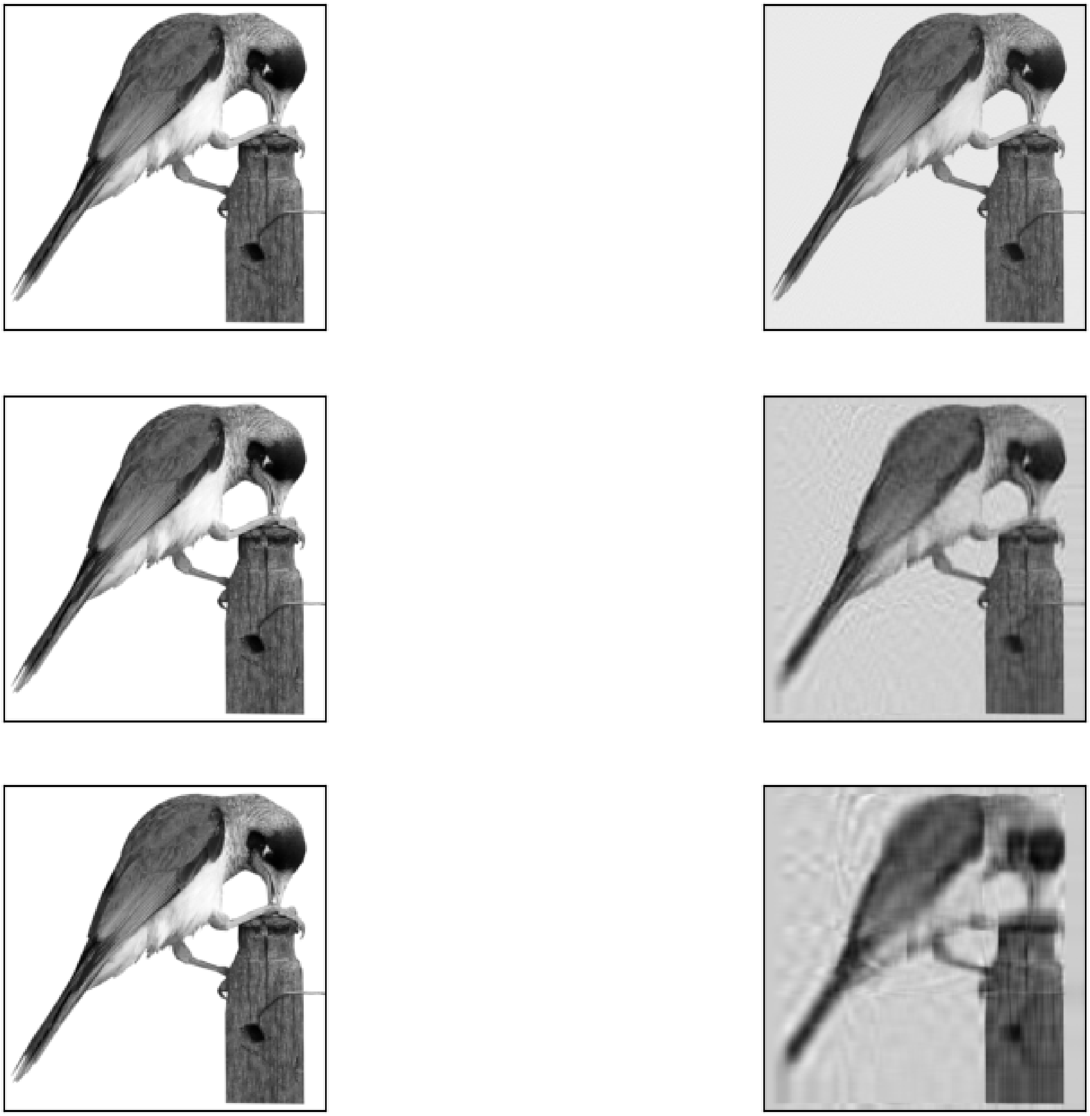
[72]:
```
def pca_com_var(X, var_exp=0.99):
    pca = PCA(var_exp)
    lower_dimension_data = pca.fit_transform(X)
    print(lower_dimension_data.shape)
    approximation = pca.inverse_transform(lower_dimension_data)
    return approximation
```

[76]:
```
def plot_subplot(X, i):
    plt.subplot(3,2,i)
    plt.imshow(X, cmap='gray')
    plt.xticks([])
    plt.yticks([])
```

[74]:
```
img_1 = pca_com_var(X, var_exp=0.99)
img_2 = pca_com_var(X, var_exp=0.95)
img_3 = pca_com_var(X, var_exp=0.90)
```

```
(1920, 145)
(1920, 31)
(1920, 14)
```

[78]:
```
plt.figure(figsize=(10,8))
plot_subplot(X,1)
plot_subplot(img_1, 2)

plot_subplot(X,3)
plot_subplot(img_2, 4)

plot_subplot(X,5)
plot_subplot(img_3, 6)
```



1.8 PCA de conjunto de ações

[80]:
```
preco = pd.read_csv('stock_prices.csv', index_col=[0,1],
                    parse_dates=True, squeeze=True)
dji_all = pd.read_csv('DJI.csv', index_col=0, parse_dates=True)
```

```
/tmp/ipykernel_148504/1485819936.py:1: FutureWarning: The squeeze argument has
been deprecated and will be removed in a future version. Append
.squeeze("columns") to the call to squeeze.


  preco = pd.read_csv('stock_prices.csv', index_col=[0,1],
```

[81]:
```
calc_return = lambda x : np.log(x/x.shift(1))[1:]
scale = lambda x: (x-x.mean())/x.std()
```

[83]:
```
import statsmodels.api as sm
def OLSreg(y, Xmat):
    return sm.OLS(y, sm.add_constant(Xmat, prepend=True)).fit()
```

[84]:
```
preco.shape
```

[84]:
```
(59150,)
```

[86]:
```
precos = preco.unstack()
dates = list(precos.index)
```

[90]:
```
import datetime as dt
dates.remove(dt.datetime(2002, 2, 1))
precos = precos.drop('DDR', axis=1)
precos = precos.drop([dt.datetime(2002, 2, 1)])
```

[91]:
```
precos.shape
```

[91]:
```
(2366, 24)
```

[92]:
```
precos
```

[92]:
```
Stock         ADC    AFL   ARKR   AZPN  CLFD    DTE   ENDP  FLWS     FR  \
Date
2002-01-02  17.70  23.78   8.15  17.10  3.19  42.37  11.54  15.77  31.16
2002-01-03  16.14  23.52   8.15  17.41  3.27  42.14  11.48  17.40  31.45
2002-01-04  15.45  23.92   7.79  17.90  3.28  41.79  11.60  17.11  31.46
2002-01-07  16.59  23.12   7.79  17.49  3.50  41.48  11.90  17.38  31.10
2002-01-08  16.76  25.54   7.35  17.89  4.24  40.69  12.41  14.62  31.40
…             …    …     …     …    …     …     …     …     …
2011-05-19  22.51  50.36  16.33  16.74  6.18  51.91  41.35   2.84  11.94
2011-05-20  22.52  49.57  16.90  16.79  6.19  51.90  41.02   2.79  11.77
2011-05-23  22.40  48.82  16.45  16.18  5.99  51.47  40.36   2.78  11.70
2011-05-24  22.12  49.23  16.64  16.10  6.02  51.51  40.24   2.81  11.74
2011-05-25  22.76  49.30  16.68  16.48  6.01  51.12  40.40   2.92  11.87

Stock       GMXR  …    KSS   MTSC    NWN   ODFL  PARL  RELV   SIGM    STT  \
Date              …
2002-01-02  4.50  …  70.23  10.03  26.20  13.40  1.92  1.30   1.75  52.11
2002-01-03  4.37  …  69.65  10.85  26.25  13.00  1.94  1.22   2.11  52.90
2002-01-04  4.45  …  70.21  10.34  26.46  13.00  1.98  1.26   2.20  54.16
2002-01-07  4.38  …  70.17   9.99  26.84  13.32  1.94  1.28   2.11  55.14
2002-01-08  4.30  …  69.90  10.35  27.35  13.75  1.94  1.27   2.25  54.44
…            …  …    …      …      …      …     …     …      …      …
2011-05-19  4.76  …  56.57  39.82  45.14  36.27  3.19  1.95  11.84  47.47
```

```
2011-05-20  4.94  …  54.66  39.19  45.08  36.43  3.25  1.98  11.83  46.84
2011-05-23  4.73  …  54.98  38.62  44.64  36.32  3.30  2.03  11.42  45.65
2011-05-24  4.77  …  54.53  37.87  44.48  35.75  3.41  1.97  11.22  44.96
2011-05-25  4.97  …  54.21  38.78  44.79  36.12  3.50  1.92  11.00  45.18

Stock        TRIB    UTR
Date
2002-01-02   1.50  39.34
2002-01-03   1.55  39.49
2002-01-04   1.54  39.38
2002-01-07   1.55  38.55
2002-01-08   1.58  38.98
…             …     …
2011-05-19  10.00  30.74
2011-05-20  10.04  30.85
2011-05-23  10.04  29.71
2011-05-24  10.28  29.26
2011-05-25  10.42  29.47

[2366 rows x 24 columns]
```

[93]:

```
dji_all =dji_all.sort_index()
dji = dji_all['Close'].reindex(index = dates)
dji_ret = calc_return(dji)
```

[94]:

```
dji_all
```

[94]:

```
                Open      High       Low     Close      Volume  Adj Close
Date
1928-10-01    239.43    242.46    238.24    240.01     3500000     240.01
1928-10-02    240.01    241.54    235.42    238.14     3850000     238.14
1928-10-03    238.14    239.14    233.60    237.75     4060000     237.75
1928-10-04    237.75    242.53    237.72    240.00     4330000     240.00
1928-10-05    240.00    243.08    238.22    240.44     4360000     240.44
…                …         …      …         …          …          …
2011-05-19  12561.46  12673.78  12506.67  12605.32  3626110000   12605.32
2011-05-20  12604.64  12630.11  12453.96  12512.04  4066020000   12512.04
2011-05-23  12511.29  12511.29  12292.49  12381.26  3255580000   12381.26
2011-05-24  12381.87  12465.80  12315.42  12356.21  3846250000   12356.21
2011-05-25  12355.45  12462.28  12271.90  12394.66  4109670000   12394.66

[20756 rows x 6 columns]
```

[95]:

```
dji
```

[95]:

```
Date
2002-01-02    10073.40
```

```
2002-01-03    10172.14
2002-01-04    10259.74
2002-01-07    10197.05
2002-01-08    10150.55
                …
2011-05-19    12605.32
2011-05-20    12512.04
2011-05-23    12381.26
2011-05-24    12356.21
2011-05-25    12394.66
Name: Close, Length: 2366, dtype: float64
```

[96]:
```
dji_ret
```

[96]:
```
Date
2002-01-03    0.009754
2002-01-04    0.008575
2002-01-07   -0.006129
2002-01-08   -0.004571
2002-01-09   -0.005578
                …
2011-05-19    0.003587
2011-05-20   -0.007428
2011-05-23   -0.010507
2011-05-24   -0.002025
2011-05-25    0.003107
Name: Close, Length: 2365, dtype: float64
```

[100]:
```
def make_pca_index(data, scale_data = True):
    if scale_data:
        data_std = data.apply(scale)
    else:
        data_std = data
    corrs = np.asarray(data_std.cov())
    pca = PCA(n_components=1).fit(corrs)
    mkt_index = -scale(pca.transform(data_std))
    return mkt_index
```

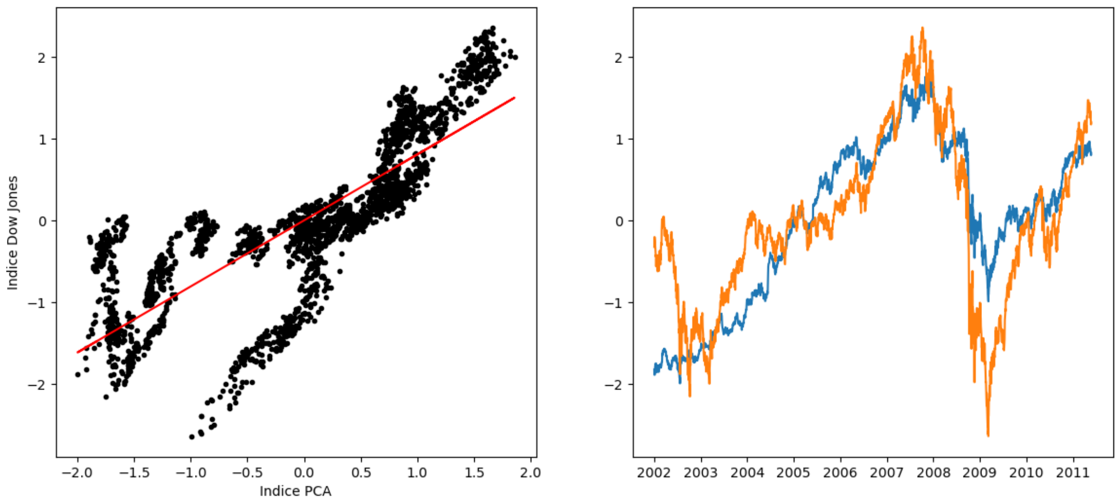
[101]:
```
indice_precos = make_pca_index(precos)
```

```
/home/hefesto/.local/lib/python3.10/site-packages/sklearn/base.py:443:
UserWarning: X has feature names, but PCA was fitted without feature names
  warnings.warn(
```

[103]:
```
plt.figure(figsize=(14,6))
plt.subplot(121)
plt.plot(indice_precos, scale(dji), 'k.')
plt.xlabel('Indice PCA')
```

```python
plt.ylabel('Indice Dow Jones')
ols_fit = OLSreg(scale(dji), indice_precos)
plt.plot(indice_precos, ols_fit.fittedvalues, 'r-')
plt.subplot(122)
plt.plot(dates, indice_precos, label='indice PCA')
plt.plot(dates, scale(dji), label = 'Indice Dow Jones')
```

[103]: [<matplotlib.lines.Line2D at 0x7fbadac7d480>]




```python
[ ]:
```