Principal Component Analysis (PCA)

Exercício 01

Implemente o PCA em C / C++ / Java / Python.

- PCA com 2 componentes.
- Teste o funcionamento usando:
  “Alps Water”;
  “US Census Dataset”.

Definição das funções

In [1]:
```
#Definição da biblioteca pandas para importar o dataset
import pandas as pd

#Definição da biblioteca numpy para calculo dos auto-valores
import numpy as np
```

Dataset - Subtração da média

In [2]:
```
#Subtraindo a média para definição do novo Dataset:

#Cálculo da média:
def calculo_mean(x):
    """Retorna a média da entrada."""
    soma=[]
    soma_i = 0
    for i in range(len(x)):
        soma_i= soma_i + x[i]
        media = (soma_i)/len(x)
    return media


def subtract_mean(x,y,z=0):
    new_X = [i - calculo_mean(x) for i in x]
    new_Y = [i - calculo_mean(y) for i in y]

    if z==0:
    #Defining dataset after subtract mean:
        dataset2=[[i,j] for i, j in zip(new_X, new_Y)]
        return dataset2

    else:
        new_Z = [i - calculo_mean(z) for i in z]
        dataset3=[[i,j,z] for i, j, z in zip(new_X, new_Y,new_Z)]
        return dataset3
```

Cálculo da matriz de covariância

In [3]:
```
#Cálculo da matriz de covariância
def cov_matrix(dataset):
     """Retorna a matriz de covariância da entrada."""
    if len(dataset[0])==2:
        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][0]*dataset[i][1]
            diag_cov_element = soma/(len(dataset)-1)
        diag_cov_element

        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][0]*dataset[i][0]
            first_cov_element = soma/(len(dataset)-1)
        first_cov_element

        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][1]*dataset[i][1]
            last_cov_element = soma/(len(dataset)-1)
        last_cov_element

        covariance_matrix = [[first_cov_element,diag_cov_element],[diag_cov_element,last_cov_element]]
        return covariance_matrix

    if len(dataset[0])==3:
        #cov(x,x)
        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][0]*dataset[i][0]
            cov_x_x = soma/(len(dataset)-1)

        #cov(y,y)
        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][1]*dataset[i][1]
            cov_y_y = soma/(len(dataset)-1)

        #cov(z,z)
        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][2]*dataset[i][2]
            cov_z_z = soma/(len(dataset)-1)

        #cov(x,y) = cov(y,x)
        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][0]*dataset[i][1]
            cov_x_y = soma/(len(dataset)-1)


        #cov(x,z) = cov(z,x)
        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][0]*dataset[i][2]
            cov_x_z = soma/(len(dataset)-1)

        #cov(y,z) = cov(z,y)
        soma=0
        for i in range(len(dataset)):
            soma = soma + dataset[i][1]*dataset[i][2]
            cov_y_z = soma/(len(dataset)-1)


        covariance_matrix = [[cov_x_x,cov_x_y,cov_x_z],
                             [cov_x_y,cov_y_y,cov_y_z],
                             [cov_x_z,cov_y_z,cov_z_z]]
        return covariance_matrix
```

Cálculo da matriz transposta

In [4]:
```
#Função para calcular a matriz transposta:

def transposta(X):
    """Retorna a transposta da matriz de entrada"""
    return [[X[j][i] for j in range(len(X))] for i in range(len(X[0]))]
```

Multiplicação de duas matrizes

In [5]:
```
#Multiplicação de duas matrizes:

def multiply_matrix_matrix(mult1,mult2):
    """Retorna o produto escalar de duas matrizes"""
    x = []
    for i in range(0,len(mult1)):
        y=[]
        for j in range(0,len(mult2[0])):
            total = 0
            for k in range(0,len(mult1[0])):
                total = total + mult1[i][k]*mult2[k][j]
            y.append(total)
        x.append(y)

    return x
```

Alps Water

In [6]:
```
#1° Passo - Pegar os dados / Importar o dataset
alps_water = pd.read_csv('DataSets/alpswater1.txt',sep='\t',header=None)
alps_water.columns= ['Row','Pressure','Boiling']
alps_water.drop(columns='Row', inplace=True)
alps_water.head()

# 1° Passo - Transformando as entradas em listas
x_alps = [i for i in alps_water['Boiling']]
y_alps = [i for i in alps_water['Pressure']]

#Defining dataset to apply PCA
dataset_alps=[[i,j] for i, j in zip(x_alps, y_alps)]

#2° Passo - Subtraindo a média
new_dataset_alps = subtract_mean(x_alps,y_alps)

#3° Passo - Cálculo da matriz de covariância
covariance_matrix_alps = cov_matrix(new_dataset_alps)

#4° Passo - Encontrar os autovalores, checar se estão na ordem
autovalores_alps = np.linalg.eig(covariance_matrix_alps)[0]

#4° Passo - Encontrar os autovetores
autovetores_alps = np.linalg.eig(covariance_matrix_alps)[1]

#5° Passo - Escolher os componentes
FeatureVector = autovetores_alps

#6° Passo - Derive the new
#Multiplicar os autovetores pela matriz após subtração da média:
pca_alps = multiply_matrix_matrix(transposta(FeatureVector),transposta(new_dataset_alps))
pca_alps

#Organizando a resposta
pca_alps_answer = [[pca_alps[0][i],pca_alps[1][i]] for i in range(len(pca_alps[0]))]
pca_alps_answer
```

Out[6]:
```
[[-9.46867735263238, 0.13862918119196577],
 [-9.64586459136409, 0.23139060868544403],
 [-5.709764802870609, -0.011957814406917233],
 [-5.141568778925107, -0.004658610852802614],
 [-4.033005159282153, -0.043216375364096615],
 [-3.4972756349593705, -0.0979327053660839],
 [-2.3608835870683698, -0.0833342982578602],
 [-2.1373156345899202, -0.08750210638548084],
 [-1.8576205623683082, -0.2000661618159525],
 [-1.9508522531088361, -0.16254481000579746],
 [0.6109030576440994, -0.22819332734663744],
 [2.1600834578811017, 0.5748882921177365],
 [7.3916805107456645, 0.003230810351890767],
 [6.2557587261017105, -0.22607618729820267],
 [8.709897868743049, -0.0660728480967765],
 [10.162619296606008, 0.1215449896155345],
 [10.511885439447706, 0.1418713632338635]]
```

In [7]:
```
#Checando a resposta
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(dataset_alps)
pca.explained_variance_ratio_
pca.singular_values_
Transformed_X = pca.transform(dataset_alps)
Transformed_X
```

Out[7]:
```
array([[-9.46867735e+00,  1.38629181e-01],
       [-9.64586459e+00,  2.31390609e-01],
       [-5.70976480e+00, -1.19578144e-02],
       [-5.14156878e+00, -4.65861085e-03],
       [-4.03300516e+00, -4.32163754e-02],
       [-3.49727563e+00, -9.79327054e-02],
       [-2.36088359e+00, -8.33342983e-02],
       [-2.13731563e+00, -8.75021064e-02],
       [-1.85762056e+00, -2.00066162e-01],
       [-1.95085225e+00, -1.62544810e-01],
       [ 6.10903058e-01, -2.28193327e-01],
       [ 2.16008346e+00,  5.74888292e-01],
       [ 7.39168051e+00,  3.23081035e-03],
       [ 6.25575873e+00, -2.26076187e-01],
       [ 8.70989787e+00, -6.60728481e-02],
       [ 1.01626193e+01,  1.21544990e-01],
       [ 1.05118854e+01,  1.41871363e-01]])
```

US Census Dataset

In [8]:
```
#1° Passo - Pegar os dados / Importar o dataset
us_census = pd.read_csv('DataSets/US-Census.txt', sep='\t', header=None)
us_census.columns = ['x','y'] #atribuição de nomes às colunas

# 1° Passo - Transformando as entradas em listas
x_us = [i for i in us_census['x']]
y_us = [i for i in us_census['y']]
dataset_us = [[i,j] for i,j in zip(y_us,x_us)]

#2° Passo - Subtraindo a média
new_dataset_us = subtract_mean(y_us,x_us)
new_dataset_us

#3° Passo - Cálculo da matriz de covariância
covariance_matrix_us = cov_matrix(new_dataset_us)
covariance_matrix_us

#4° Passo - Encontrar os autovalores
autovalores_us = np.linalg.eig(covariance_matrix_us)[0]
autovalores_us

#4° Passo - Encontrar os autovetores
autovetores_us = np.linalg.eig(covariance_matrix_us)[1]
autovetores_us

#5° Passo - Escolher os componentes
FeatureVector_us = autovetores_us

#6° Passo - Derive the new dataset
pca_us = multiply_matrix_matrix(transposta(FeatureVector_us),transposta(new_dataset_us))
pca_us

#Organizando a resposta
pca_us_answer = [[pca_us[0][i],pca_us[1][i]] for i in range(len(pca_us[0]))]
pca_us_answer
```

Out[8]:
```
[[-102.26303932718267, -5.880648264018603],
 [-83.52185636156509, -3.8725172799329215],
 [-66.79343152132931, -0.8858932050646011],
 [-46.68972462365029, 0.45985304598454846],
 [-34.70360255157288, 5.751925765936422],
 [-13.218485251487028, 6.426105713392109],
 [16.898648352180082, 2.903870698679783],
 [42.75553385936203, 1.4527356710262147],
 [68.07640335102677, 0.2621823601315576],
 [93.24887914712045, -0.8562300392442808],
 [126.21067492709801, -5.761384466890277]]
```

In [9]:
```
#Checando a resposta
import numpy as np
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(dataset_us)
pca.explained_variance_ratio_
pca.singular_values_
Transformed_X = pca.transform(dataset_us)
Transformed_X
```

Out[9]:
```
array([[-102.26303933,   -5.88064826],
       [ -83.52185636,   -3.87251728],
       [ -66.79343152,   -0.88589321],
       [ -46.68972462,    0.45985305],
       [ -34.70360255,    5.75192577],
       [ -13.21848525,    6.42610571],
       [  16.89864835,    2.9038707 ],
       [  42.75553386,    1.45273567],
       [  68.07640335,    0.26218236],
       [  93.24887915,   -0.85623004],
       [ 126.21067493,   -5.76138447]])
```

Exercício 02

Implemente o PCA em C / C++ / Java Python.

- PCA com 3 componentes
- Pode usar um solucionador para os autovalores: `numpy.linalg.eig`
- Teste o funcionamento usando: “Books x Grades”

Books x Grades

In [10]:
```
#1° Passo - Pegar os dados / Importar o dataset
books_attend_grade = pd.read_csv('DataSets/Books_attend_grade.txt',sep='\t',header=None)
books_attend_grade.columns=['Books', 'Attend', 'Grade']

#1° Passo - Transformando as features em linhas
books = [i for i in books_attend_grade['Books']]
attend = [i for i in books_attend_grade['Attend']]
grade = [i for i in books_attend_grade['Grade']]
dataset_books=[[i,j,k] for i, j, k in zip(books,attend,grade)]

#2° Passo - Subtraindo a média
new_dataset_books = subtract_mean(x=books,y=attend,z=grade)

#3° Passo - Cálculo da matriz de covariância
covariance_matrix_books = cov_matrix(new_dataset_books)

#4° Passo - Encontrar os autovalores
autovalores_books = np.linalg.eig(covariance_matrix_books)[0]

#Ordenando os autovalores
autovalores_books_order=[autovalores_books[0],autovalores_books[2],autovalores_books[1]]
autovalores_books_order

#4° Passo - Encontrar os autovetores
autovetores_books = np.linalg.eig(covariance_matrix_books)[1]

#Ordenando (para seguir a ordem dos autovalores)
autovetores_books_order=[[autovetores_books[0][0],autovetores_books[0][2],autovetores_books[0][1]],
                         [autovetores_books[1][0],autovetores_books[1][2],autovetores_books[1][1]],
                         [autovetores_books[2][0],autovetores_books[2][2],autovetores_books[2][1]]]


#5° Passo - Escolher os componentes
Feature = autovetores_books_order

#6° Passo - Derive the new dataset
pca_books = multiply_matrix_matrix(transposta(Feature),transposta(new_dataset_books))
pca_books_answer = [[pca_books[0][i],pca_books[1][i],pca_books[2][i]] for i in range(len(pca_books[0
]))]
pca_books_answer
```

Out[10]:
```
[[-19.120804656379264, 2.764724624002969, -0.929315364134587],
 [-6.415968932866218, -1.6608246798150847, -0.8885403960499472],
 [-18.991919282209103, 1.7780490661827764, -1.028612413059671],
 [-12.18896168749004, -3.5365507956094957, 0.18579949131701012],
 [0.993327687092036, 4.048313636919347, 2.353073905169412],
 [25.06930570477028, -2.772788839906002, 0.6738349427458701],
 [-19.81118936814235, 0.5644214506876208, -0.10346132420851861],
 [24.078561171032163, -2.905208539965853, 0.703672702449031],
 [25.373034294022887, 2.387541006869924, 0.14587204959505362],
 [-4.477068939024697, -1.3014529220802702, -1.9428262935293443],
 [1.641123325220414, 6.343149167849807, 0.5326094871702681],
 [1.2522165886983854, 1.371884036304108, -0.9286483758250665],
 [-6.892290123189728, -5.03686812618085, 1.8272373510971882],
 [-16.967841141098166, 1.9483561086873658, -0.09367755439291742],
 [1.5559451779509954, 6.532213883080033, -1.4566112689758828],
 [-22.912308343526856, 1.1538379083282595, 0.08534900382604826],
 [-7.192649945164756, -2.9689846529253527, 1.0312210708742808],
 [-26.78899017794387, -0.2679840921162242, -0.8892073843594677],
 [-17.78936184104296, -7.1965633118140655, -0.9276754753123517],
 [-5.684113300734452, 1.1480246208600056, 2.3633441252413707],
 [-16.71118854602388, 0.6780834780218363, 2.790856530901225],
 [0.9921952263465159, -4.5860566530617835, -2.4892032877454855],
 [33.2564014902931, 3.541430964963844, -1.0874404061033094],
 [-8.31227985307303, -2.1147287951650116, 1.160355879502526],
 [-11.716009264444114, -7.388720669275152, -1.2059990824564015],
 [-3.6555482390798986, 7.8434664984211615, -1.1088283726099102],
 [5.5581436192605835, -0.2608529943270972, 0.7426272436582726],
 [15.981130453322372, -2.883358225009356, 0.04706145092632702],
 [7.3681582389319455, 1.0851943212279096, -0.21236160489604078],
 [-1.6774279388812507, 0.8800925785094424, 0.15547528135749178],
 [23.649315974886974, 0.1493504911098369, 0.006953471151207591],
 [-8.701186589595059, -7.085993926710712, -0.3009019834928088],
 [-20.75934482824576, 0.3374693930126571, 0.9209868135677178],
 [28.98969476608803, -2.148577682051485, -0.4401264741398493],
 [20.115583036079702, -3.434887340205257, 0.8230237412616748],
 [31.01377290719897, -1.9782706395468956, 0.49480838452690434],
 [-4.131869429403409, 4.467423052055397, 1.6069493745372252],
 [-8.308911085795447, 5.11348452486641, -0.16362329387119795],
 [24.468586060820225, 1.3629781066049929, -0.9181976176999449],
 [-2.149262208661147, 4.029183967200244, -1.5358543280134072]]
```

In [11]:
```
#Checando a resposta
import numpy as np
from sklearn.decomposition import PCA
pca = PCA(n_components = 3)
pca.fit(dataset_books)
pca.explained_variance_ratio_
pca.singular_values_
Transformed_X = pca.transform(dataset_books)
Transformed_X
```

Out[11]:
```
array([[-1.91208047e+01,  2.76472462e+00, -9.29315364e-01],
       [-6.41596893e+00, -1.66082468e+00, -8.88540396e-01],
       [-1.89919193e+01,  1.77804907e+00, -1.02861241e+00],
       [-1.21889617e+01, -3.53655080e+00,  1.85799491e-01],
       [ 9.93327687e-01,  4.04831364e+00,  2.35307391e+00],
       [ 2.50693057e+01, -2.77278884e+00,  6.73834943e-01],
       [-1.98111894e+01,  5.64421451e-01, -1.03461324e-01],
       [ 2.40785612e+01, -2.90520854e+00,  7.03672702e-01],
       [ 2.53730343e+01,  2.38754101e+00,  1.45872050e-01],
       [-4.47706894e+00, -1.30145292e+00, -1.94282629e+00],
       [ 1.64112333e+00,  6.34314917e+00,  5.32609487e-01],
       [ 1.25221659e+00,  1.37188404e+00, -9.28648376e-01],
       [-6.89229012e+00, -5.03686813e+00,  1.82723735e+00],
       [-1.69678411e+01,  1.94835611e+00, -9.36775544e-02],
       [ 1.55594518e+00,  6.53221388e+00, -1.45661127e+00],
       [-2.29123083e+01,  1.15383791e+00,  8.53490038e-02],
       [-7.19264995e+00, -2.96898465e+00,  1.03122107e+00],
       [-2.67889902e+01, -2.67984092e-01, -8.89207384e-01],
       [-1.77893618e+01, -7.19656331e+00, -9.27675475e-01],
       [-5.68411330e+00,  1.14802462e+00,  2.36334413e+00],
       [-1.67111885e+01,  6.78083478e-01,  2.79085653e+00],
       [ 9.92195226e-01, -4.58605665e+00, -2.48920329e+00],
       [ 3.32564015e+01,  3.54143096e+00, -1.08744041e+00],
       [-8.31227985e+00, -2.11472880e+00,  1.16035588e+00],
       [-1.17160093e+01, -7.38872067e+00, -1.20599908e+00],
       [-3.65554824e+00,  7.84346650e+00, -1.10882837e+00],
       [ 5.55814362e+00, -2.60852994e-01,  7.42627244e-01],
       [ 1.59811305e+01, -2.88335823e+00,  4.70614509e-02],
       [ 7.36815824e+00,  1.08519432e+00, -2.12361605e-01],
       [-1.67742794e+00,  8.80092579e-01,  1.55475281e-01],
       [ 2.36493160e+01,  1.49350491e-01,  6.95347115e-03],
       [-8.70118659e+00, -7.08599393e+00, -3.00901983e-01],
       [-2.07593448e+01,  3.37469393e-01,  9.20986814e-01],
       [ 2.89896948e+01, -2.14857768e+00, -4.40126474e-01],
       [ 2.01155830e+01, -3.43488734e+00,  8.23023741e-01],
       [ 3.10137729e+01, -1.97827064e+00,  4.94808385e-01],
       [-4.13186943e+00,  4.46742305e+00,  1.60694937e+00],
       [-8.30891109e+00,  5.11348452e+00, -1.63623294e-01],
       [ 2.44685861e+01,  1.36297811e+00, -9.18197618e-01],
       [-2.14926221e+00,  4.02918397e+00, -1.53585433e+00]])
```

Exercício 03

Compare PCA com o método dos mínimos quadrados.

- Compare a reta gerada pelos mínimos quadrados e a gerada pelo PCA.

> To solve this exercise I'll use `Alps Water Dataset`

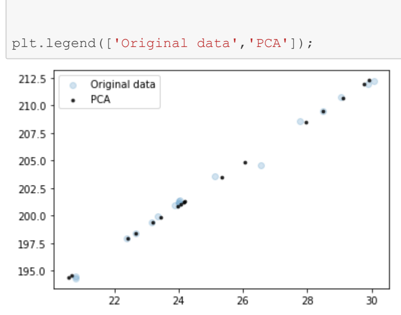
Obs: All the resolution was solved without libraries (except linalg.eig - autorized by teacher). Now, I will use libraries just to plot and compare the results of last exercise (Least Squares) with these (PCA) in a faster way.

Import Libraries

In [12]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression #for plot the result of least squares
import pandas as pd
```

Original Data

In [49]:
```
# Plot Original Data
plt.scatter(y_alps, x_alps, alpha=0.2)
plt.legend(['Original data']);
```

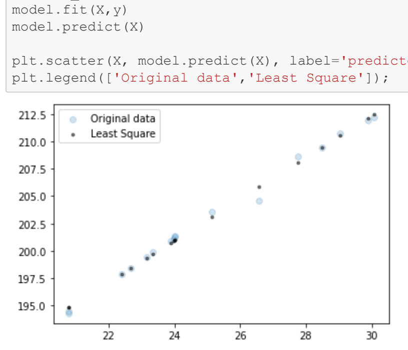


PCA x Original Data

In [50]:
```
#Using library just to compare the plots
pca = PCA(n_components=1)
X= alps_water
pca.fit(X)
X_pca = pca.transform(X)
X_new = pca.inverse_transform(X_pca)

plt.scatter(X.iloc[:, 0], X.iloc[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8, s=8, c='k')
model = LinearRegression()
X=alps_water[['Boiling']]
y=alps_water['Pressure']


plt.legend(['Original data','PCA']);
```

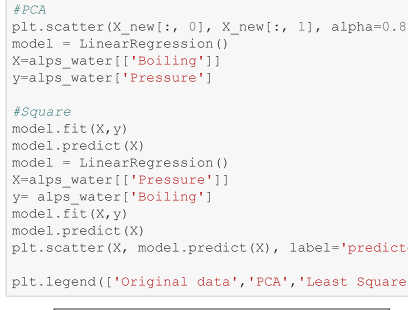


Least Square x Original Data

In [51]:
```
#Using library just to compare the plots
plt.scatter(alps_water.iloc[:, 0], alps_water.iloc[:, 1], alpha=0.2)

model.fit(X,y)
model.predict(X)
model = LinearRegression()
X=alps_water[['Pressure']]
y= alps_water['Boiling']
model.fit(X,y)
model.predict(X)

plt.scatter(X, model.predict(X), label='predicted', alpha=0.5, s=8, c='k')
plt.legend(['Original data','Least Square']);
```



Original Dat x PCA x Least Square

In [52]:
```
#Using library just to compare the plots
#Original
plt.scatter(alps_water.iloc[:, 0], alps_water.iloc[:, 1], alpha=0.2)

#PCA
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8, s=8, c='k')
model = LinearRegression()
X=alps_water[['Boiling']]
y=alps_water['Pressure']

#Square
model.fit(X,y)
model.predict(X)
model = LinearRegression()
X=alps_water[['Pressure']]
y= alps_water['Boiling']
model.fit(X,y)
model.predict(X)
plt.scatter(X, model.predict(X), label='predicted', alpha=0.5, s=8, c='k')

plt.legend(['Original data','PCA','Least Square']);
```