

Correction_Examen_S2_POO

May 12, 2026

```
[27]: import numpy as np
import matplotlib.pyplot as plt
```

```
[28]: class Serie :
    def __init__(self , d ) : # d = {'ciment' : [150 , 220 , 370 , ... ] }
        self.label = list( d.keys() )[0]
        self.valeurs = np.array( d[ self.label ] )

    def std( self ) :
        moyX = np.mean( self.valeurs )
        V = self.valeurs - moyX
        S = np.sum(V**2) / len(V)
        return np.sqrt( S )

    def standariser(self) :
        Z = (self.valeurs - np.mean(self.valeurs)) / self.std()
        S = Serie( { self.label : Z.tolist() } )

    def correlation(self, other) :
        X = self.valeurs - np.mean(self.valeurs)
        Y = other.valeurs - np.mean( other.valeurs)

        nume = X @ Y
        deno = np.linalg.norm(X) * np.linalg.norm(Y)

        return nume / deno

    def __eq__(self, val ) :
        return self.valeurs == val

    def __eq__(self , val ) :
        return np.array( [ v == val for v in self.valeurs ] )

    def tracer_Nuage( self , other ) :
        Lx = self.valeurs
        Ly = other.valeurs
```

```

plt.scatter( Lx , Ly)
# ou bien
plt.plot( Lx , Ly , ls = '' , marker = '+') # ls : line style

plt.show()

```

```

[38]: dict_ciment = {'ciment' : np.random.randint(150,380 , size = 100 ) }
dict_eau = {'eau' : np.random.randint(70,150 , size = 100 ) }
dict_gravier = {'gravier' : np.random.randint(800,1150 , size = 100 ) }
dict_sable = {'sable' : np.random.randint(700,950 , size = 100 ) }

S1 = Serie( dict_ciment )
S2 = Serie( dict_eau)

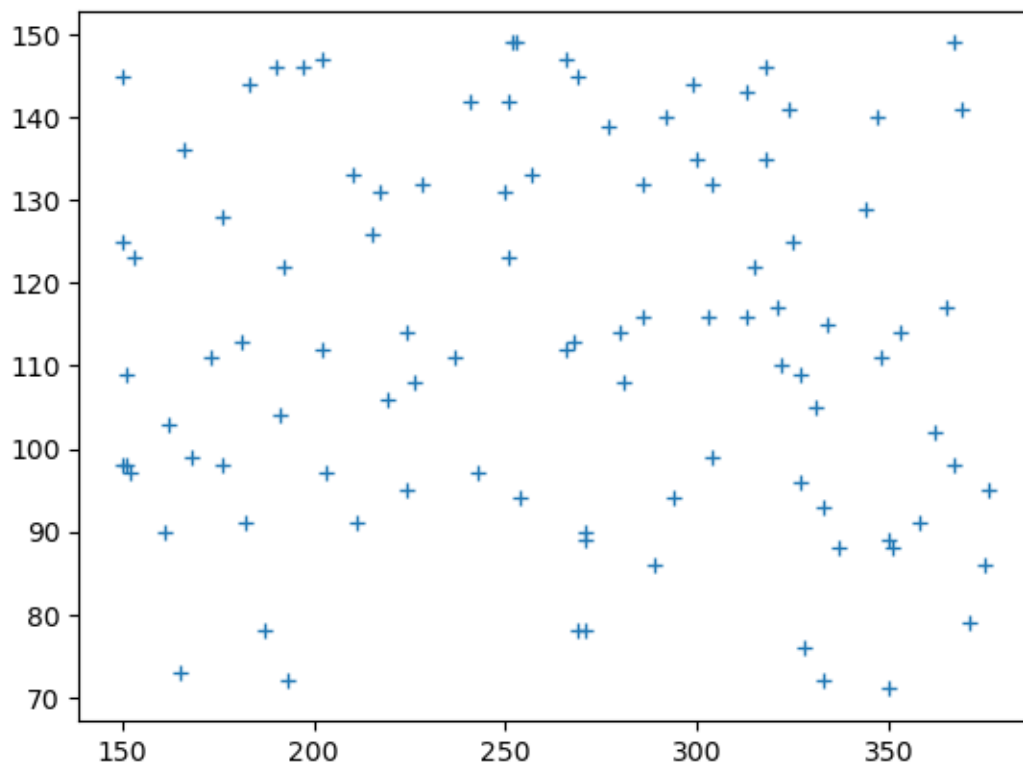
```

```

[39]: print("correlation = " , S1.correlation(S2))
S1.tracer_Nuage(S2)

```

correlation = -0.05766876690173758



```

[73]: class DataFrame :
    def __init__(self , dict_serie ) : # { 'ciment' : [150 , 220 , ...] , 'eau' :
↪: [ ...] }
        self.series = []
        colonne = []

        for nom , Lv in dict_serie.items() :
            colonne.append( nom )
            self.series.append( Serie( {nom : Lv } ) )

        self.columns = np.array( colonne )
        self.shape = ( len(self.series[0].valeurs ) , len(self.series ) )

    def projection(self , nomSerie ) :
        try :
            #assert nomSerie in self.columns
            j = self.columns.tolist().index( nomSerie ) # declenche une erreur
            return self.series[j].valeurs
        except :
            return None

    def to_array(self) :
        n,p = self.shape
        X = np.zeros((n,p))
        for j in range(p) :
            X[ : , j ] = self.projection(self.columns[j] )
        return X

    def __getitem__(self, key) :
        (i , nomSerie) = key
        #assert nomSerie in self.columns
        return self.projection(nomSerie)[i]

    def selection(self, mask) :

        d = {s.label : s.valeurs[mask] for s in self.series }
        df = DataFrame( d )
        return df

    def selection(self, mask) :
        n,p = self.shape
        d = dict()

        X = self.to_array()
        X_mask = np.array( [ X[i] for i in range(n) if mask[i] == True ] )

        d = {self.columns[j] : X_mask[:,j] for j in range(p) }

```

```

df = DataFrame( d )
return df

def orderBy(self , nomSerie , asc = True ) :
    n,p = self.shape
    j = self.columns.tolist().index( nomSerie )
    X = self.to_array()
    L = sorted( X , key = lambda ligne : ligne[j] , reverse = not asc )
    X_sort = np.array( L )

    d = {self.columns[j] : X_sort[ : , j ] for j in range(p) }

    df = DataFrame( d )
    return df

def groupBy( self , nomSerie ) :
    pass

def matrice_correlation(self) :
    n,p = self.shape
    C = np.zeros((p,p))
    for i in range( p ) :
        for j in range(p) :
            C[i,j] = self.series[i].correlation( self.series[j] )
    return C

```

```

[74]: df = DataFrame({'ciment' : np.random.randint(150,380 , size = 100 ) ,
                    'eau' : np.random.randint(70,100 , size = 100 ) ,
                    'gravier' : np.random.randint(800,1150 , size = 100 ) ,
                    'sable' : np.random.randint(700,950 , size = 100 )
                    })

print("Columns = " , df.columns)
print("Shape = ",df.shape)
print("X = \n", df.to_array()[:10])

```

```

Columns = ['ciment' 'eau' 'gravier' 'sable']
Shape = (100, 4)
X =
[[ 207.   93.  926.  887.]
 [ 151.   73.  880.  718.]
 [ 288.   70.  878.  747.]
 [ 336.   78. 1068.  732.]
 [ 218.   98. 1047.  848.]
 [ 329.   72.  962.  931.]
 [ 367.   78. 1141.  903.]

```

```
[ 366.   90.  923.  910.]
[ 245.   77. 1125.  785.]
[ 379.   72. 1070.  927.]]
```

```
[75]: df.projection('gravier') [ : 10 ]
```

```
[75]: array([ 926,  880,  878, 1068, 1047,  962, 1141,  923, 1125, 1070])
```

```
[76]: mask = (df.series[1] == 97) # appel a __eq__ de la classe Serie
df2 = df.selection ( mask )
df2.to_array()
```

```
[76]: array([[ 327.,   97., 1098.,  940.],
             [ 345.,   97., 1000.,  880.],
             [ 348.,   97.,  981.,  702.],
             [ 293.,   97.,  967.,  792.]])
```

```
[78]: df3 = df2.orderBy( 'ciment' , asc = False)
df3.to_array()
```

```
[78]: array([[ 348.,   97.,  981.,  702.],
             [ 345.,   97., 1000.,  880.],
             [ 327.,   97., 1098.,  940.],
             [ 293.,   97.,  967.,  792.]])
```

```
[81]: df.matrice_correlation().round(2)
```

```
[81]: array([[ 1.   ,  0.02, -0.01, -0.05],
             [ 0.02,  1.   ,  0.04,  0.07],
             [-0.01,  0.04,  1.   , -0.01],
             [-0.05,  0.07, -0.01,  1.   ]])
```

```
[91]: class RegressionLineaire :
        def __init__( self ) :
            self.constante = 0
            self.V_coef = None

        def matrice_augmentee(self, objDataFrame) :
            n,p = objDataFrame.shape
            X = objDataFrame.to_array()
            X1 = np.ones( (n,p+1) , dtype = float)
            X1[ : , 1 : ] = X
            return X1

        def fit(self, objDataFrame, Serie_resistance) :
            X1 = self.matrice_augmentee(objDataFrame)
            y = Serie_resistance.valeurs
```

```

A = np.linalg.inv(X1.T @ X1) @ X1.T @ y

self.constante = A[0]
self.V_coef = A[ 1 : ]

```

```
[92]: modele_RG = RegressionLineaire()
```

```
[95]: X1 = modele_RG.matrice_augmentee(df)
X1[ : 10 ]
```

```
[95]: array([[1.000e+00, 2.070e+02, 9.300e+01, 9.260e+02, 8.870e+02],
 [1.000e+00, 1.510e+02, 7.300e+01, 8.800e+02, 7.180e+02],
 [1.000e+00, 2.880e+02, 7.000e+01, 8.780e+02, 7.470e+02],
 [1.000e+00, 3.360e+02, 7.800e+01, 1.068e+03, 7.320e+02],
 [1.000e+00, 2.180e+02, 9.800e+01, 1.047e+03, 8.480e+02],
 [1.000e+00, 3.290e+02, 7.200e+01, 9.620e+02, 9.310e+02],
 [1.000e+00, 3.670e+02, 7.800e+01, 1.141e+03, 9.030e+02],
 [1.000e+00, 3.660e+02, 9.000e+01, 9.230e+02, 9.100e+02],
 [1.000e+00, 2.450e+02, 7.700e+01, 1.125e+03, 7.850e+02],
 [1.000e+00, 3.790e+02, 7.200e+01, 1.070e+03, 9.270e+02]])
```

```
[96]: dict_resistance = {'Resistance' : np.random.randint(25,37 , size = 100 ) }
S_resistance = Serie( dict_resistance )
```

```
[97]: modele_RG.fit(df , S_resistance)
```

```
[99]: print(modele_RG.constante)
```

```
30.118689009240615
```

```
[100]: print(modele_RG.V_coef)
```

```
[ 0.00467938  0.02629002 -0.00496167  0.00196412]
```

```
[103]: # Exemple d'essai : {ciment:250 , 'eau' :120, 'gravier' : 900 , 'sable' : 700}
resistance_predit = modele_RG.constante + np.array([250 , 120 , 900 , 700 ]) @
↳modele_RG.V_coef
print("Resistance predite par AI : ", resistance_predit )
```

```
Resistance predite par AI : 31.352721329600204
```