Universidad Autónoma de Nuevo León
Facultad De Ciencias Físico Matemática

Etapa 1 Ejercicios de las 4 primeras presentaciones

Minería de Datos

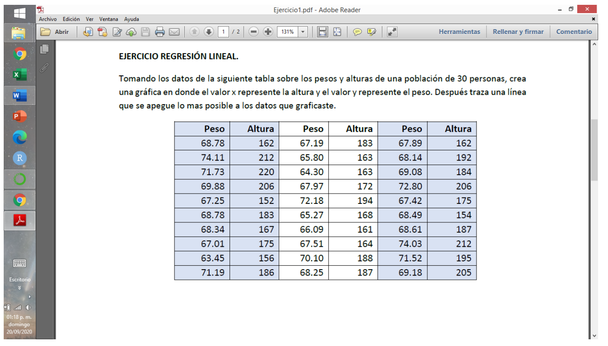
Maestra:
Mayra Cristina Berrones Reyes

Alumnos:	**Matricula:**
Andrea López Solís	#1822031
Daniela Govea Serna	#1722714
Francisco García Sánchez Armáss	#1816358
Jesús Eduardo Valencia González	#1630606
Karyme Mayela Gauna Rodríguez	#1819032

Grupo: 003
Aula: AV12

San Nicolás de los Garza, Nuevo León, México a 25 de septiembre del 2020

Ejercicio 1 Regresión



Peso	Altura	Peso	Altura	Peso	Altura
68.78	162	67.19	183	67.89	162
74.11	212	65.80	163	68.14	192
71.73	220	64.30	163	69.08	184
69.88	206	67.97	172	72.80	206
67.25	152	72.18	194	67.42	175
68.78	183	65.27	168	68.49	154
68.34	167	66.09	161	68.61	187
67.01	175	67.51	164	74.03	212
63.45	156	70.10	188	71.52	195
71.19	186	68.25	187	69.18	205

In [5]:
```
x = [162, 212, 220, 206, 152, 183, 167, 175, 156, 186,
     183, 163, 163, 172, 194, 168, 161, 164, 188, 187,
     162, 192, 184, 206, 175, 154, 187, 212, 195, 205]

y = [68.78, 74.11, 71.73, 69.88, 67.25, 68.78, 68.34, 67.01, 63.45, 71.1
9,
     67.19, 64.80, 64.30, 67.97, 72.18, 65.27, 66.09, 67.51, 70.10, 68.2
5,
     67.89, 68.14, 69.08, 72.80, 67.42, 68.49, 68.61, 74.03, 71.52, 68.1
8]
```

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Número de Items

In [7]:
```
n = len(x)
n
```

Out[7]:
```
30
```

Variables -> vector

In [8]:
```
x = np.array(x)
y = np.array(y)
x
```

Out[8]:
```
array([162, 212, 220, 206, 152, 183, 167, 175, 156, 186, 183, 163, 163,
       172, 194, 168, 161, 164, 188, 187, 162, 192, 184, 206, 175, 154,
       187, 212, 195, 205])
```

Cálculo de datos

In [9]:
```
sumx = sum(x)
sumy = sum(y)
sumx2 = sum(x**2)
sumy2 = sum(y**2)
sumxy = sum(x*y)

promx = sumx/n
promy = sumy/n

sumy
```

Out[9]:
```
2060.3399999999997
```

Cálculo ecuación del modelo lineal

In [10]:
```
m = (sumx*sumy - n*sumxy)/(sumx**2 - n*sumx2)
b = promy - m*promx

m, b
```

Out[10]:
```
(0.10807442840927588, 49.10211853413315)
```

y = -202.92x + 5.5921

Cálculo de R2 ADJ

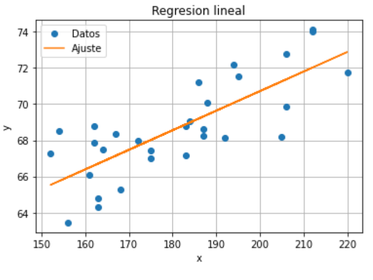
In [11]:
```
sigmax = np.sqrt(sumx2/n - promx**2)
sigmay = np.sqrt(sumy2/n - promy**2)
sigmaxy = sumxy/n - promx*promy

R2 = (sigmaxy/(sigmax*sigmay))**2
R2
```

Out[11]:
```
0.6043677719659506
```

R2 = 60.43%

Gráfico

In [12]:
```
plt.plot(x, y, 'o', label = 'Datos')
plt.plot(x, m*x + b, label = 'Ajuste')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Regresion lineal')
plt.grid()
plt.legend()
plt.show()
```

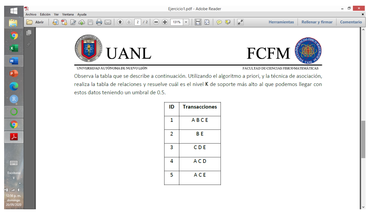


Bibliografía

https://github.com/soloSergioo/Mineria_de_Datos/blob/master/RegresionL_Temp.ipynb

In [ ]:
```
```

Ejercicio 2 Reglas de asociación



In [38]:
```
def Load_data():
    transacciones = [
        ["A", "B", "C", "E"],
        ["B", "E"],
        ["C", "D", "E"],
        ["A", "C", "D"],
        ["A", "C", "E"],
    ]
    return transacciones
```

In [39]:
```
def createC1 (data):
    C1 = []
    for  transacción in datos :
        for item in  transacción:
            if not [item] in C1:
                C1 . append ([item])
    C1 . sort ()

    #crear un conjunto para cada articulo en C1
    return [ set (x) for x in C1]
```

In [40]:
```
def createCk(Lk, k):
    #Creamos una lista con los candidatos con longitud k
        #Argumentos: Lk: es la lista con los itemsets frecuentes
                    #k: the size of the itemsets
    cand_list = []
    len_Lk = len(Lk)
    #Unir conjuntos si los primeros k-2 elementos son iguales
    for i in range(len_Lk):
        for j in range(i+1, len_Lk):
            L1 = list(Lk[i])[:k-2]
            L2 = list(Lk[j])[:k-2]
            L1.sort()
            L2.sort()
            if L1==L2:
                cand_list.append(Lk[i] | Lk[j])

    return cand_list
```

In [41]:
```
def scanD(data, Ck, min_support):
    """
    Analizar a través de los datos de transacciones y devolver una lista
de candidatos que cumplen
    umbral de soporte y datos de apoyo sobre los candidatos actuales.

    Argumentos:
       datos: conjunto de datos,
       Ck: una lista de conjuntos de candidatos
       min_support: el apoyo minimo

    """
    count = {}
    for transaction in data:
        tr = set(transaction)
        for candidate in Ck:
            if candidate.issubset(tr):
                can = frozenset(candidate)
                if can not in count:
                    count[can] = 1
                else:
                    count[can] += 1
                    num_items = float(len(D))

    cand_list = []
    support_data = {}

    # calculamos el soporte por cada itemset
    for key in count:
        support = count[key]/num_items

        #Si el soporte cumple con los requisitos minimos de soporte,
        #agréguelo a la lista de conjuntos de elementos.
        if support >= min_support:
            cand_list.insert(0, key)
        support_data[key] = support

    return cand_list, support_data
```

In [42]:
```
min_support = 0.5
```

In [43]:
```
data = Load_data()
data
```

Out[43]:
```
[['A', 'B', 'C', 'E'],
 ['B', 'E'],
 ['C', 'D', 'E'],
 ['A', 'C', 'D'],
 ['A', 'C', 'E']]
```

In [44]:
```
#K=1
C1 = createC1(data)
C1
```

Out[44]:
```
[{'A'}, {'B'}, {'C'}, {'D'}, {'E'}]
```

In [45]:
```
D = list(map(set, data))
D
```

Out[45]:
```
[{'A', 'B', 'C', 'E'},
 {'B', 'E'},
 {'C', 'D', 'E'},
 {'A', 'C', 'D'},
 {'A', 'C', 'E'}]
```

In [46]:
```
L1, support_data1 = scanD(D, C1, min_support)
L1
```

Out[46]:
```
[frozenset({'E'}), frozenset({'C'}), frozenset({'A'})]
```

In [47]:
```
support_data1
```

Out[47]:
```
{frozenset({'A'}): 0.6,
 frozenset({'B'}): 0.4,
 frozenset({'C'}): 0.8,
 frozenset({'E'}): 0.8,
 frozenset({'D'}): 0.4}
```

In [48]:
```
#K=2
C2 = createCk(L1, k=2)
C2
```

Out[48]:
```
[frozenset({'C', 'E'}), frozenset({'A', 'E'}), frozenset({'A', 'C'})]
```

In [49]:
```
L2, support_data2 = scanD(D, C2, min_support)
L2
```

Out[49]:
```
[frozenset({'A', 'C'}), frozenset({'C', 'E'})]
```

In [50]:
```
support_data2
```

Out[50]:
```
{frozenset({'C', 'E'}): 0.6,
 frozenset({'A', 'E'}): 0.4,
 frozenset({'A', 'C'}): 0.6}
```

In [51]:
```
#K=3
C3 = createCk(L2, k=3)
C3
```

Out[51]:
```
[frozenset({'A', 'C', 'E'})]
```

In [52]:
```
L3, support_data3 = scanD(D, C3, min_support)
L3
```

Out[52]:
```
[]
```

In [59]:
```
support_data3
```

Out[59]:
```
{frozenset({'A', 'C', 'E'}): 0.4}
```

Resultados finales

K=1

In [57]:
```
#Para ver cuales itemsets cumplen con la condicion de que sean >=0.5
support_data1
```

Out[57]:
```
{frozenset({'A'}): 0.6,
 frozenset({'B'}): 0.4,
 frozenset({'C'}): 0.8,
 frozenset({'E'}): 0.8,
 frozenset({'D'}): 0.4}
```

K=2

In [58]:
```
support_data2
```

Out[58]:
```
{frozenset({'C', 'E'}): 0.6,
 frozenset({'A', 'E'}): 0.4,
 frozenset({'A', 'C'}): 0.6}
```

K=3

In [60]:
```
support_data3
```

Out[60]:
```
{frozenset({'A', 'C', 'E'}): 0.4}
```

Como el problema solo nos dice que usemos hasta k-terminos, hasta el k=3 no cumple con la condición y hasta ese itemsets terminamos

** Los unicos que cumple con la condición del umbral de soporte >=5 son:

K=1

frozenset({'A'}): 0.6 >= 0.5

frozenset({'C'}): 0.8 >= 0.5

frozenset({'E'}): 0.8 >= 0.5

K=2

frozenset({'C', 'E'}): 0.6 >=0.5

frozenset({'A', 'C'}): 0.6 >=0.5

Bibliografía

https://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=27202

In [ ]:
```
```