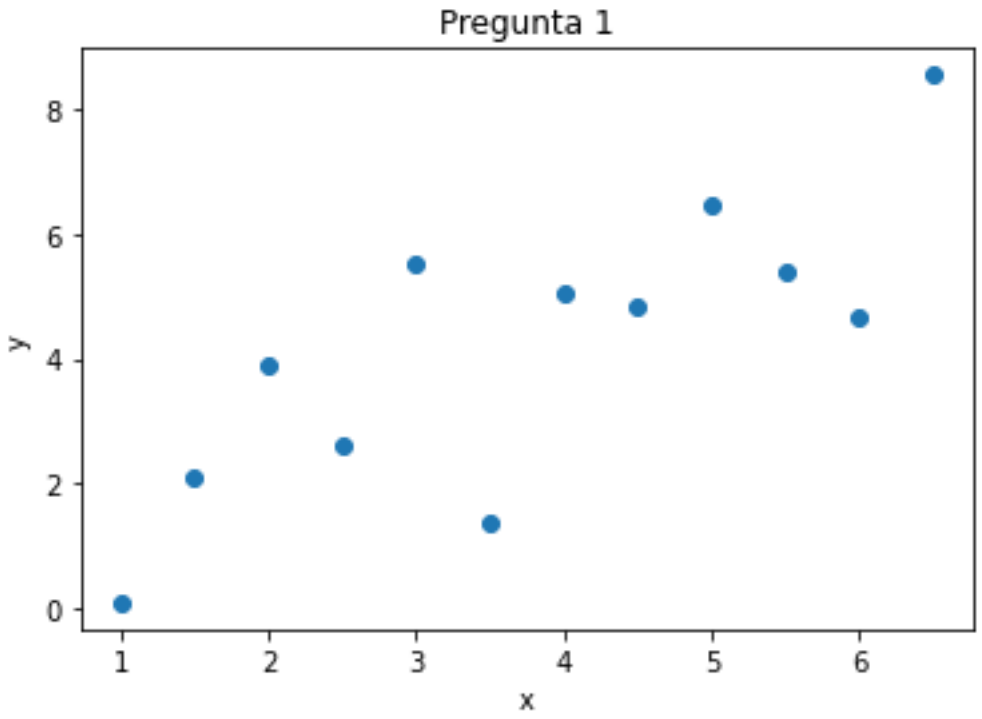
Deber 1

Roberto Alvarado

4 de Septiembre del 2021

- **Pregunta 1**



a) Solamente basándome en la gráfica es posible que las variables tengan un relación lineal.

b) Asumiendo que y se relaciona con x linealmente, utilizamos el método de mínimos cuadrados para encontrar una aproximación de α y β para $\bar{y} = \alpha x + \beta$
Para eso intentamos minimizar la suma de los cuadrados de los residuos.

$$S = \sum_{i=1}^{n} (y_i - \bar{y})^2 = \sum_{i=1}^{n} (y_i - \alpha x_i + \beta)^2$$

Para minimizar este error hacemos uso de la técnica de descenso del gradiente, donde vamos a ir al contrario del gradiente ya que este tiene la dirección del máximo crecimiento. Entonces, calculamos el gradiente con las derivadas parciales de la función S, según α y β. Para eso

$$\nabla S = \begin{bmatrix} \frac{\partial S}{\partial \alpha} & \frac{\partial S}{\partial \beta} \end{bmatrix}$$

$$\nabla S = \begin{bmatrix} -\sum_{i=1}^{n} 2(y_i - \alpha x_i - \beta)(x_i) & -\sum_{i=1}^{n} 2(y_i - \alpha x_i - \beta) \end{bmatrix}$$

Al tener el gradiente podemos tomar el inverso aditivo del mismo, claro que como este gradiente tiende a ser un valor muy grande y si lo sumamos directamente simplemente esta iteración hace que los parámetro divergan, por eso necesitamos un fracción de este gradiente, de igual manera, lo único que necesitamos es la dirección del mismo, por eso se lo multiplica por un γ. Despúes de multiplicar este valor se suma a los parámetros, y volvemos a iterar este proceso las veces que se necesiten según la capacidad computacional, es importante analizar cual es el valor inicial que definimos nuestro parametros (α,β inicial) de forma que igual no divergan. A continuación podemos ver la implementación de la fúncion para la obtención del gradiente

```
# funcion para encontrar el gradiente
def grad(X,y,yt):
  alpha = -2*np.sum((y-yt)*X[0,0,:])
  beta = -2*np.sum((y-yt))
  return(np.array([alpha,beta]))

```

En el siguiente bloque de código se implementa el algoritmo, en este caso utilizamos unos valor iniciales de $\alpha = 3$ y $\beta = 0$ =0, con un $\gamma = 0.004$ ya que cualquier valor superior resulta en divergencia.

```
#valor inicial
alpha = 3
beta = 0
X = np.ones((1,2,len(x)))
X[0,0,:]=x

param = np.array([alpha,beta])

gamma = 0.004

'''
loop para encontrar los parametros que se transmitiran
Si coloco 0.005 como gamma diverge
con este valor el llega a un valor considerable
'''
for i in range(10000):
    yt = np.matmul(param,X[0,:,:])
    param = param-gamma*grad(X,y,yt)
print(param)

```

El código expuesto resulta en los siguientes valores para los parámetros

$$\alpha = 1.03369$$

$$\beta = 0.33968$$

Podemos ver el procedimiento en una hoja de jupyter notebook en la figura 1 con su resultado. El error se analiza en el ejercicio d

Figure 1: Implementacion del algoritmo descrito

```python
#Funcion de calculo de error
def fp(y,yt):
    return np.sum((y-yt)**2)
```

```python
# funcion para encontrar el gradiente para aproximacion lineal
def grad(X,y,yt):
    alpha = -2*np.sum((y-yt)*X[0,0,:])
    beta = -2*np.sum((y-yt))
    return(np.array([alpha,beta]))
```

```python
#b)
#estimulo inicial
alpha = 3
beta = 0
X = np.ones((1,2,len(x)))
X[0,0,:]=x

param = np.array([alpha,beta])

gamma = 0.004

'''
loop para encontrar los parametros que se transmitiran
Si coloco 0.005 como gamma diverge
con este valor el llega a un valor considerable
'''
for i in range(10000):
    yt = np.matmul(param,X[0,:,:])
    param = param-gamma*grad(X,y,yt)
print(param)
fp(y,yt)
```

```
[1.03369434 0.33967874]

22.495390037277975
```

c) Para encontrar el vector $\bar{y}$ simplemente es necesario hacer uso de la siguiente implementación en python

```
#param es el resultado del anterior codigo
malpha = param[0]
mbeta = param[1]
function = lambda x: malpha*x + mbeta
print(function(x))

```

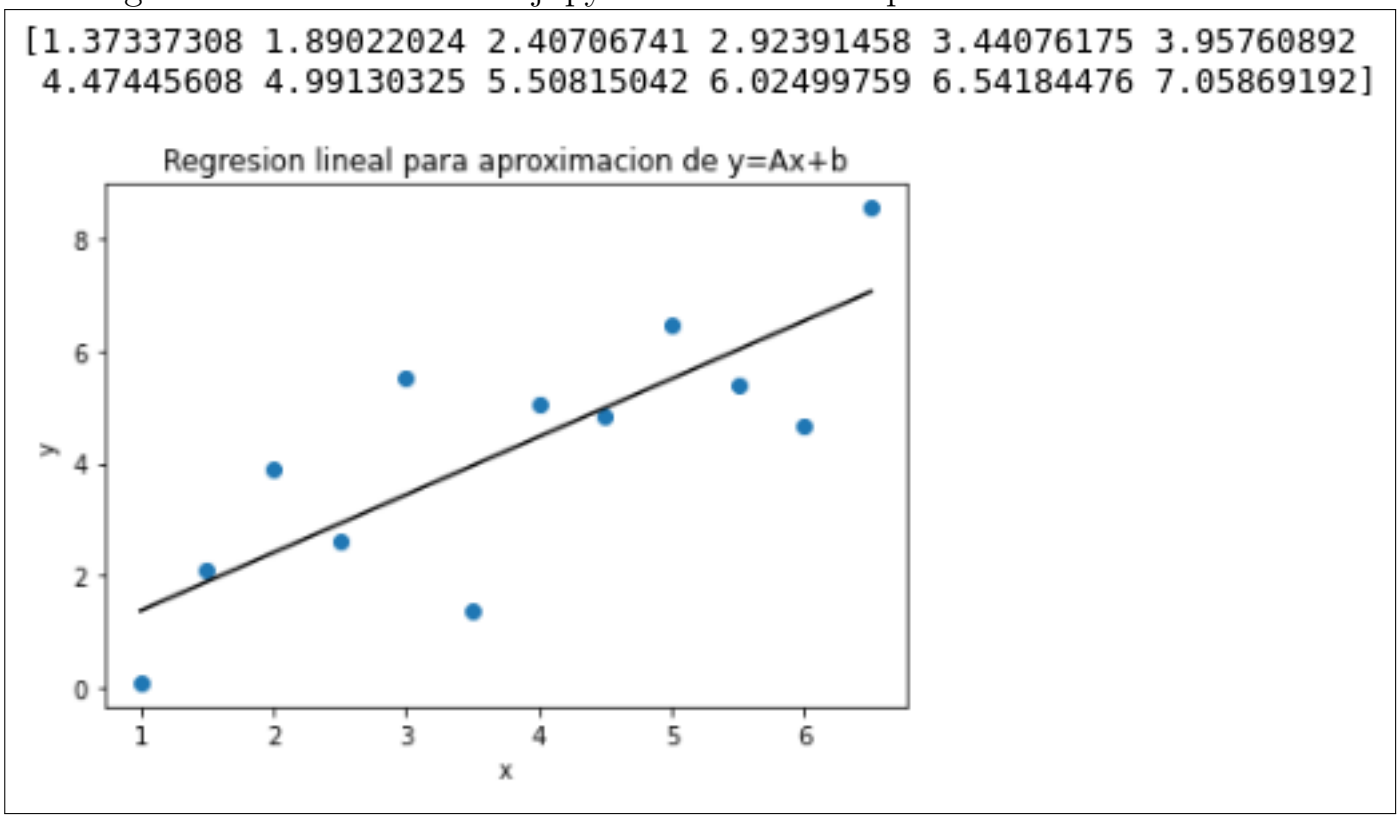
Con lo que resulta que

$$\bar{y} = [1.373373081.890220242.407067412.923914583.440761753.95760892$$

$$4.474456084.991303255.508150426.024997596.541844767.05869192]$$

En este caso podemos hacer uso de matplotlib para visualizar la recta

Figure 2: Resultado de un jupyter notebook del procedimiento



d) El vector $e = y - \bar{y}$ en este caso se calcula en python de la siguiente forma, además que encontramos a su vez su magnitud. Por medio del codigo

```
#yt esta definido como el vector resultante del literal c
e = y - yt
print(e)

#magnitud error
magnitudE = np.sqrt(e.dot(e))

```

Lo que resulta en un valor del vector e de

$$e = [-1.28207308 0.20117976 1.48563259 - 0.32131458 2.09383825 - 2.58250892$$

$$0.58944392 - 0.15350325 0.97074958 - 0.62069759 - 1.87504476 1.49429808]$$

y el valor de la magnitud de este vector es de

$$|e| = 4.74293$$

- **Pregunta2**

  a) Realizando el procedimiento anterior, con la misma definicion de la funcion grad(x,y,yt), para el conjunto de datos cars.csv tenemos los mismos resultados para los parametros Siendo en-

```python
import pandas as pd
import numpy as np

df = pd.read_csv("cars.csv")
x = df.speed
y = df.dist
x = np.array(x)
y = np.array(y)

#Realizando el anterior procedimiento

#estimulos iniciales
alpha=4
beta=0
gamma=0.00001

X = np.ones((1,2,len(x)))
X[0,0,:]=x

param = np.array([alpha,beta])

for i in range(100000):
    yt = np.matmul(param,X[0,:,:])
    param = param-gamma*grad(X,y,yt)
print(param)

```

```
[  3.93237521 -17.57851881]
```

tonces los parametros

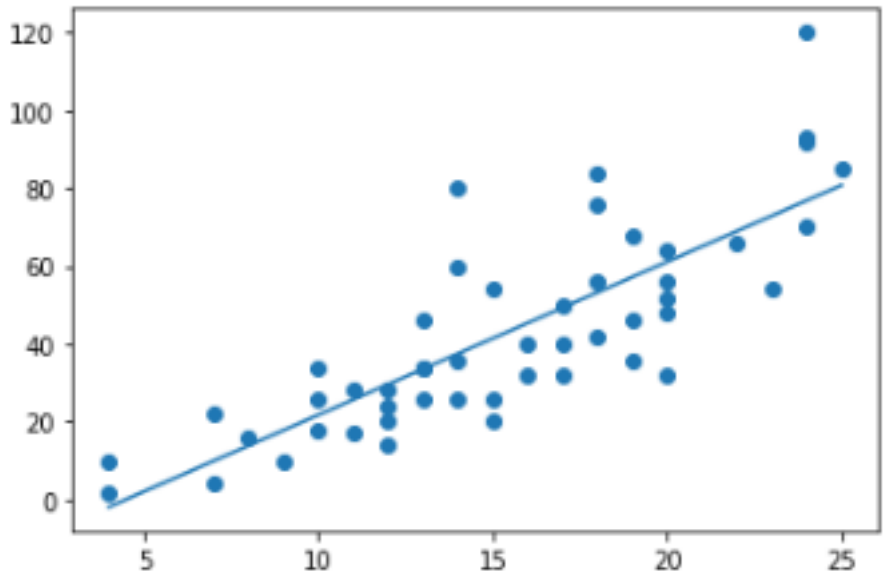
$$\alpha = 3.93237$$

$$\beta = -17.57851$$

podemos ver dentro de la figura, que el calculo del error resulta en 106.55, un valor bastante alto. Entonces por eso asumir que es una relacion lineal es un error

```
plt.scatter(x,y)
f =lambda x: param[0]*x + param[1]
plt.plot(x,f(x))
e = y - f(x)
np.sqrt(e.dot(e))
```

```
106.55290260154084
```



b) Para este proceso tenemos que comenzar de nuevo con el análisis de estímulos procesos y transmisión. Para eso notamos primeramente que es lo que estamos viendo. Utilizamos nuestro vector de estimulos podemos ver que.

$$\begin{bmatrix} xi^2 \\ xi \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_2 \\ \alpha_1 \\ \alpha \end{bmatrix}$$

Lo que resulta que podamos conseguir un vector de estimulos de la siguiente forma

$$\begin{bmatrix} \alpha_2 & \alpha_1 & \alpha \end{bmatrix} \cdot \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 & ... & x_n^2 \\ x_1 & x_2 & x_3 & ... & x_n \\ 1 & 1 & 1 & ... & 1 \end{bmatrix}$$

Vamos a utilizar el metodo de minimos cuadrados, entonces vamos a minimizar la funcion S tal que S

$$S = \sum_{i=1}^{n} (y_i - \bar{y})^2 = \sum_{i=1}^{n} (y_i - \alpha_2 x_i^2 - \alpha_1 x_i - \alpha)$$

y volvermos a buscar el gradiente

$$\nabla S = \begin{bmatrix} \frac{\partial S}{\partial \alpha_2} & \frac{\partial S}{\partial \alpha_1} & \frac{\partial S}{\partial \alpha} \end{bmatrix}$$

$$\nabla S = \begin{bmatrix} -\sum_{i=1}^{n} 2 * (y_i - \alpha_2 x_i^2 - \alpha_1 x_i - \alpha)(x_i^2) & -\sum_{i=1}^{n} 2 * (y_i - \alpha_2 x_i^2 - \alpha_1 x_i - \alpha)(x_i) & -\sum_{i=1}^{n} 2 * (y_i - \alpha_2 x_i^2 - \alpha_1 x_i - \alpha) \end{bmatrix}$$

Ahora realizamos el mismo algoritmo para estos nuevos parametros. La definicion de nuestra funcion grad esta dada por

```
def grad(X,y,yt):
  alpha2 = -2*np.sum((y-yt)*(X[0,0,:]))
  alpha1 = -2*np.sum((y-yt)*(X[0,1,:]))
  alpha = -2*np.sum((y-yt))
  return np.array([alpha2,alpha1,alpha])

```

Y finalmente aplicamos el algoritmo con la siguiente implementacion en Python

```
alpha = 0
alpha1 = 1
alpha2 = 0.2
gamma= 0.0000001

#array x,y
x = np.array(df.speed)
```

```
y = np.array(df.dist)

#tensor de estimulos
X = np.ones((1,3,len(x)))
X[0,1,:]=x
X[0,0,:]=x**2

#parametros
param = np.array([alpha2,alpha1,alpha])

#iteracion algorimica
for i in range(100000):
    yt = np.matmul(param,X[0,:,:])

    param = param - gamma*grad(X,y,yt)
print(param)

```

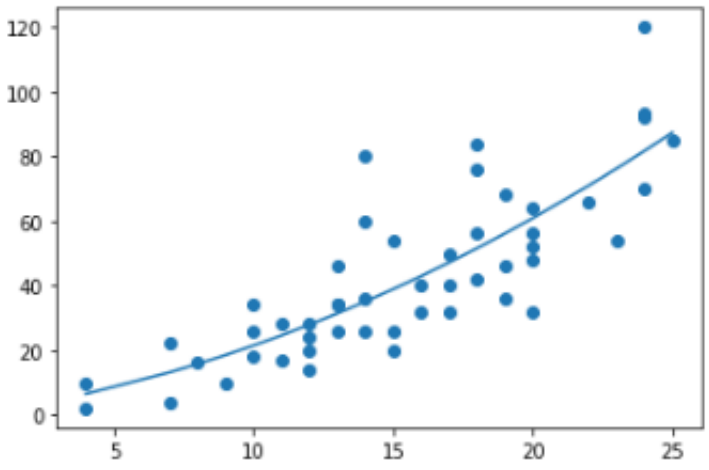
Tenemos el siguiente resultado

$$\alpha_2 = 0.09043885 \quad \alpha_1 = 1.22876285 \quad \alpha = 0.08128858$$

En la siguiente figura podemos ver el vector aproximado

```
#grafico
fun = lambda x : param[0]*(x**2) + param[1]*x+param[0]
plt.plot(x,fun(x))
plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x7f5b25ae40a0>
```



Para el calculo del error seguimos utilizando el proceso que ya conociamos, restando el vector aproximado al vector y sacando el modulo

```
#param es el resultado del anterior proceso
fun = lambda x : param[0]*(x**2) + param[1]*x+param[0]
e = y - fun(x)
np.sqrt(e.dot(e))

```

teniendo como resultado que la magnitud del error es de

$$e_{cuadratico} = 104.0704644391518$$

para el proceso lineal, el error que se obtuvo

$$e_{lineal} = 107.59129863991082$$

Por lo que en conclusion, el modelo cuadratico es mejor para interpretar este conjunto de datos.

- **Pregunta 3**