LABORATORIO2

MATERIA: INTELIGENCIA ARITIFICIAL

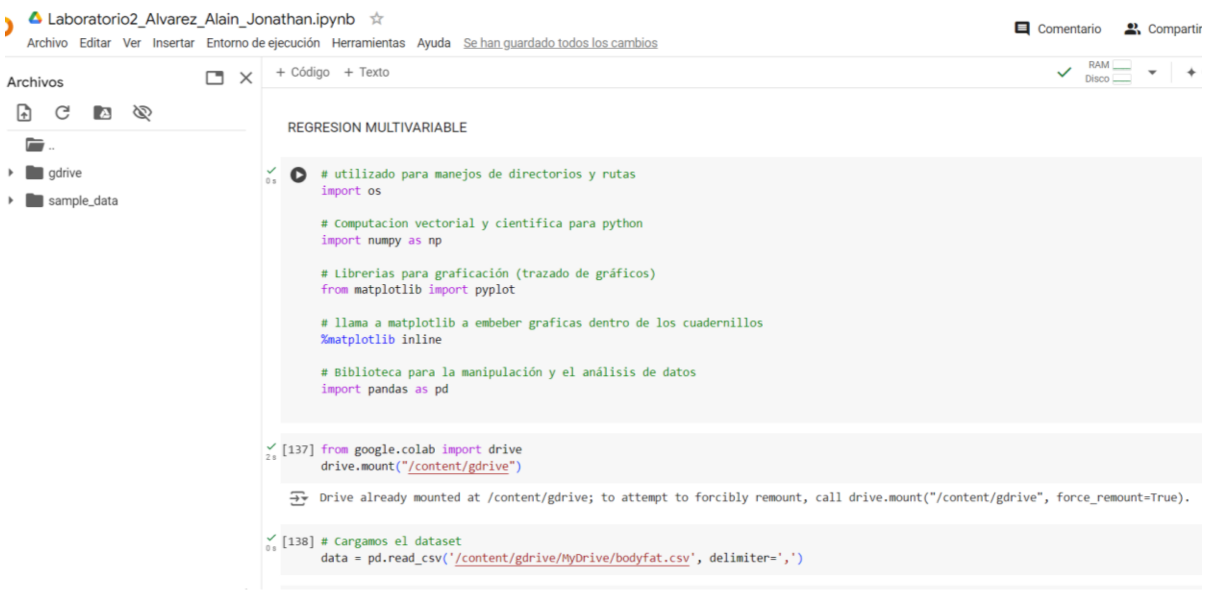
ESTUDIANTE: Alvarez Alain Jonathan

AÑO:2024

Carrera: Ing. en Diseño y Animación Digital

1.- RESUMEN EXPLICANDO LO TRABAJADO

Primero que nada, se importa las librerías que usare y comenzare desarrollando lo que es regresión lineal múltiple



Me aseguro también de importar mi dataset desde mi drive con el código que estaba debajo de la importación de librerías, copiando la ruta de mi dataset lo que hago es importarlo.

LINK DE MI DATASET :

https://www.kaggle.com/datasets/fedesoriano/body-fat-prediction-dataset

Con el siguiente código lo que hago es seleccionar las columnas que usaré.

```
[140] # Seleccionamos las columnas independientes (X) y dependiente(y)
X = data.iloc[:, 1:]
y = data.iloc[:, 0]
m = y.size
```

Luego visualizo lo que es mi dataset por un código

```
display(data)
```

	Density	BodyFat	Age	Weight	Height	Neck	Chest	Abdomen	Hip	Thigh	Knee	Ankle	Biceps	Forearm	Wrist
0	1.0708	12.3	23	154.25	67.75	36.2	93.1	85.2	94.5	59.0	37.3	21.9	32.0	27.4	17.1
1	1.0853	6.1	22	173.25	72.25	38.5	93.6	83.0	98.7	58.7	37.3	23.4	30.5	28.9	18.2
2	1.0414	25.3	22	154.00	66.25	34.0	95.8	87.9	99.2	59.6	38.9	24.0	28.8	25.2	16.6
3	1.0751	10.4	26	184.75	72.25	37.4	101.8	86.4	101.2	60.1	37.3	22.8	32.4	29.4	18.2
4	1.0340	28.7	24	184.25	71.25	34.4	97.3	100.0	101.9	63.2	42.2	24.0	32.2	27.7	17.7
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
247	1.0736	11.0	70	134.25	67.00	34.9	89.2	83.6	88.8	49.6	34.8	21.5	25.6	25.7	18.5
248	1.0236	33.6	72	201.00	69.75	40.9	108.5	105.0	104.5	59.6	40.8	23.2	35.2	28.6	20.1
249	1.0328	29.3	72	186.75	66.00	38.9	111.1	111.5	101.7	60.3	37.3	21.5	31.3	27.2	18.0
250	1.0399	26.0	72	190.75	70.50	38.9	108.3	101.3	97.8	56.0	41.6	22.7	30.5	29.4	19.8
251	1.0271	31.9	74	207.50	70.00	40.8	112.4	108.5	107.1	59.3	42.2	24.6	33.7	30.0	20.9

252 rows × 15 columns

Pasos siguientes: Generar código con data | Ver gráficos recomendados | New interactive sheet

Luego me dispongo de usar los códigos y las fórmulas enseñadas en clases:

ain_Jonathan.ipynb

torno de ejecución Herramientas Ayuda Se han guardado todos los cambios

Comentario Compartir

+ Código + Texto RAM Disco Gemini

```
[145] # a la ecuación de predicción para ajustar la línea de
      # regresión de manera que se alinee mejor con los datos observados
      X = np.concatenate([np.ones((m, 1)), X_norm], axis=1)
```

```
[146] # Esta funcion calcula el costo, el costo mide la diferencia entre las predicciones realizadas por el modelo y los
      # valores reales observados en los datos.
      def computeCostMulti(X, y, theta):
          # almacenamos en m la cantidad de filas que tiene y
          m = y.shape[0]
          # variable que almacenara el valor del costo
          J = 0
          J = (1/(2 * m)) * np.sum(np.square(np.dot(X, theta) - y))
          return J
```

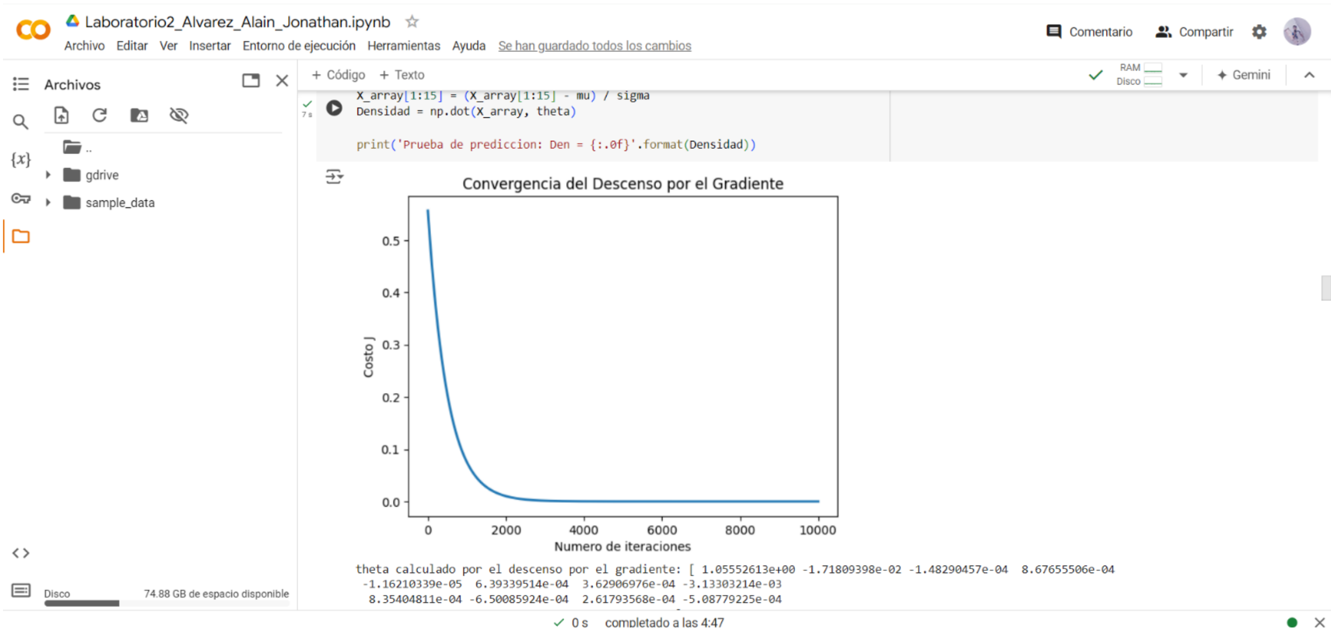
```
[147] # Funcion descenso de la gradiente, ajusta los parametros theta del modelo de regresion de manera que se minimice la funcion de costo
      def gradientDescentMulti(X, y, theta, alpha, num_iters):
          # almacenamos en m la cantidad de filas que tiene y
          m = y.shape[0]
          # se realiza una copia de theta para evitar modificar el vector original fuera de la funcion
          # las thetas son los parametros o coeficientes que determinan la relacion entre variables dependientes e independientes
          # Este valor representa cuánto cambia el precio de la casa por cada unidad adicional de tamaño.
          # Si θ1 es positivo, significa que a medida que aumenta el tamaño de la casa, también aumenta el precio.
          # el objetivo es encontrar una relación lineal entre las características de entrada y la salida
          theta = theta.copy()
          # una lista que almacenara el valor de la funcion de costo en cada iteracion
          J_history = []
          for i in range(num_iters):
              theta = theta - (alpha / m) * (np.dot(X, theta) - y).dot(X)
              # Después de actualizar theta, se calcula el costo actual con computeCostMulti y se guarda en J_history.
              J_history.append(computeCostMulti(X, y, theta))
```

ponible

Como fue enseñado en clases cambio lo que son las variables “X_Array” en relación a mis datos que tengo de mi dataset:

```python
# Elegir algun valor para alpha
alpha = 0.001
num_iters = 10000

# inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(15)
theta, J_history = gradientDescentMulti(X, y, theta, alpha, num_iters)

# Grafica la convergencia del costo
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Numero de iteraciones')
pyplot.ylabel('Costo J')
pyplot.title('Convergencia del Descenso por el Gradiente')
pyplot.show()

# Muestra los resultados del descenso por el gradiente
print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta)))
X_array = [1,12.3,23,154.25,67.75,36.2,93.1,85.2,94.5,59.0,37.3,21.9,32.0,27.4,17.1]
X_array[1:15] = (X_array[1:15] - mu) / sigma
Densidad = np.dot(X_array, theta)

print('Prueba de prediccion: Den = {:.0f}'.format(Densidad))
```

Luego grafico lo que seria el descenso por el gradiente



PARA LA NORMAL:

En aquí hacemos lo mismo hasta donde seleccionamos nuestras X y Y para luego usar una formula diferente aprendida en clases que sería la siguiente

```
[109] X = np.concatenate([np.ones((m, 1)), X], axis=1)
```

```
[111] # Funcion de la ecuacion de la normal, obtenemos valores optimos
      def normalEqn(X, y):
          theta = np.zeros(X.shape[1])
          theta = np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)),X.T),y)
          return theta
```

```
[114] # Calcula los parametros con la ecuación de la normal
      theta = normalEqn(X, y)

      # Muestra los resultados obtenidos a partir de la aplicación de la ecuación de la normal
      print('Theta calculado a partir de la ecuación de la normal: {:s}'.format(str(theta)))
      X_array = [1,12.3,23,154.25,67.75,36.2,93.1,85.2,94.5,59.0,37.3,21.9,32.0,27.4,17.1]
      price = np.dot(X_array, theta)

      print('Densidad predecido para una persona (usando la ecuación de la normal): den = {:.0f}'.format(price))
```

```
Theta calculado a partir de la ecuación de la normal: [ 1.09814633e+00 -2.21954247e-03  1.74382153e-05  4.32112085e-05
 -4.64369070e-06  3.06376944e-05  7.01838282e-05 -1.57402034e-04
  9.06335692e-05 -9.08373185e-05 -1.45223508e-05 -2.45248574e-04
 -1.72391330e-04 -1.35475763e-05  3.61663447e-04]
Densidad predecido para una persona (usando la ecuación de la normal): den = 1
```

Como en el anterior modificamos X_Array con las variables de acuerdo a nuestro dataset.

PARA LA REGRESION POLINOMICA

De igual forma hacemos lo mismo hasta donde seleccionamos nuestras X y Y para luego utilizar las fórmulas correspondientes aprendidas en clases

```
[129] def gradientDescentPoly(X, y, theta, alpha, num_iters):

          # Inicializa algunos valores
          m = y.shape[0] # numero de ejemplos de entrenamiento

          # realiza una copia de theta, el cual será acutalizada por el descenso por el gradiente
          theta = theta.copy()

          J_history = []

          for i in range(num_iters):
              theta = theta - (alpha / m) * (np.dot(X, theta) - y).dot(X)
              J_history.append(computeCostMulti(X, y, theta))

          return theta, J_history
```

```
[130] num_filas, num_columnas = X.shape
      print("Número de filas:", num_filas)
      print("Número de columnas:", num_columnas)
```

```
Número de filas: 252
Número de columnas: 25
```

```
[131] # Elegir algun valor para alpha (probar varias alternativas)
      alpha = 0.001
      num_iters = 10000

      # inicializa theta y ejecuta el descenso por el gradiente
      theta = np.zeros(25)
```

cio disponible

0 s completado a las 4:47

Luego en X_Array y en theta como en la múltiple debemos ponerlo de acuerdo a nuestro dataset solo que en este caso las variables cambiaran por las formulas y debemos acomodarlo, en esta parte me confundí pero logre guiarme.

```
# inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(25)
theta, J_history = gradientDescentPoly(X, y, theta, alpha, num_iters)

# Grafica la convergencia del costo
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Numero de iteraciones')
pyplot.ylabel('Costo J')

# Muestra los resultados del descenso por el gradiente
print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta)))

X_array = [1,12.3,23,154.25,67.75,36.2,93.1,85.2,94.5,59.0,37.3,21.9,32.0,151.29,529,23793.0625,4590.0625,1310.44,8667.61,
           7259.04,8930.25,3481,1391.29,479.61,1024]
X_array[1:25] = (X_array[1:25] - mu) / sigma
Densidad = np.dot(X_array, theta)

print('Densidad predecido para una persona (usando el descenso por el gradiente): den = {:.0f}'.format(Densidad))
```

```
theta calculado por el descenso por el gradiente: [44.88289303 -2.41876276 -0.59806214  0.6874983   0.64490901  5.491902
 -1.97939158 -4.38860981  2.0371774  -1.37231466  0.27306516 -0.90252847
  2.71074692 -2.02889973 -1.72407072  0.45685474  0.47428072  4.92427571
 -1.9505534  -4.43698106  2.05906967 -0.10407199  0.3897767  -1.50134015
  2.43956934]
Densidad predecido para una persona (usando el descenso por el gradiente): den = 276
```

Luego por último mostramos la grafica por el descenso de la gradiante