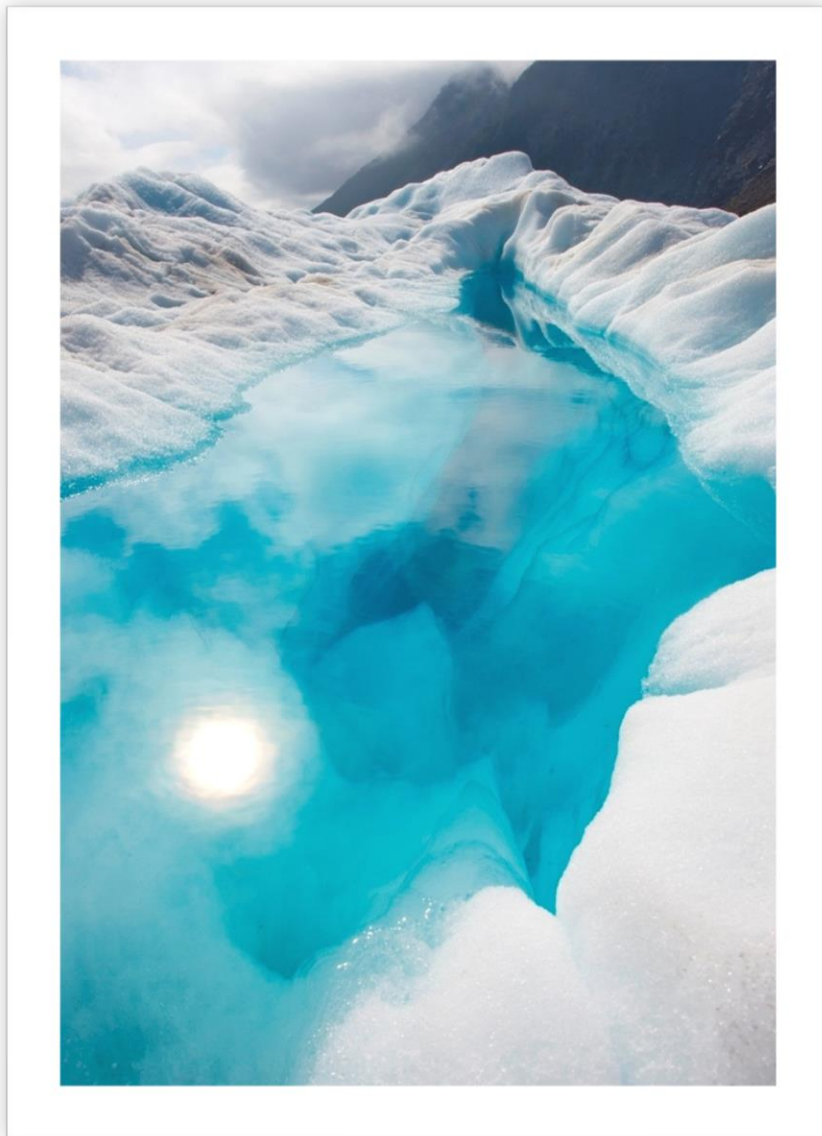


GRADO EN INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE HUELVA



Regresión Lineal

MEMORIA DE PRÁCTICA

Udad sidelgaum liman

Contenido

1. Introducción.....	2
2. Representación gráfica de los datos:	2
3. Aplicación Externa: Excel.....	3
4. Ecuaciones normales (matrices):	3
5. Descenso por gradiente:.....	4
6. Descenso por gradiente estocástico:	8
7. Función de predicción:.....	9
8. Representación de punto con las líneas de los tres algoritmos:	10
9. Pregunta 10 de la practica:.....	11

1. Introducción

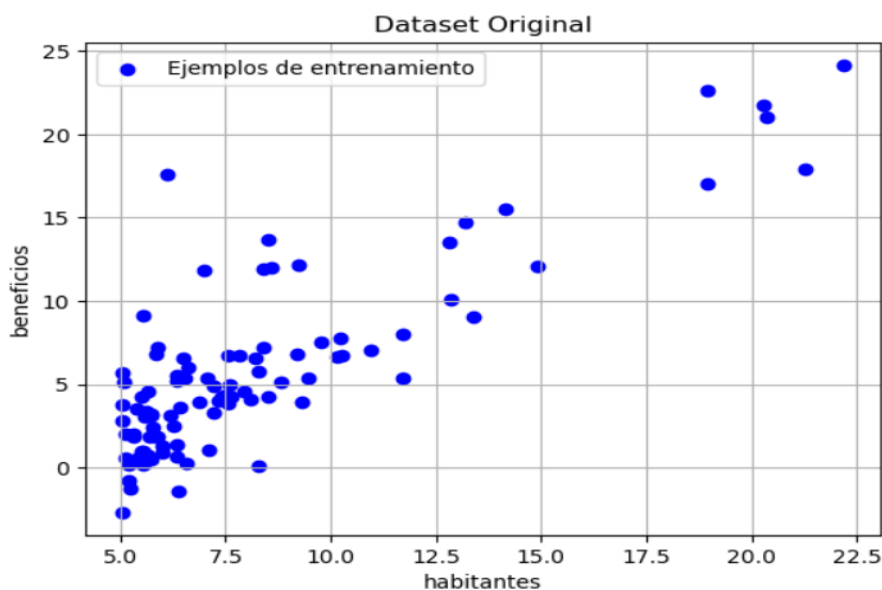
Esta práctica consiste en implementar los métodos de regresión lineal visto en clase, usando el contenido del fichero (regresión_1.csv).

Los métodos que usamos para el calculo de los coeficientes de la regresión lineal son:

- **Aplicación Externa:** Utilizando la aplicación Excel calculamos los coeficientes de la recta de regresión lineal para tener una primera aproximación.
- **Ecuaciones normales (matrices):** Utilizamos el cálculo de matrices mediante la librería numpy de Python para obtener los valores de ajuste de la regresión lineal.
- **Descenso por Gradiente:** Obtenemos los coeficientes mediante la implementación del algoritmo descenso por gradiente en Python.
- **Descenso por Gradiente Estocástico:** Mediante la implementación del algoritmo descenso por gradiente Estocástico en Python obtenemos los coeficientes de la regresión lineal.

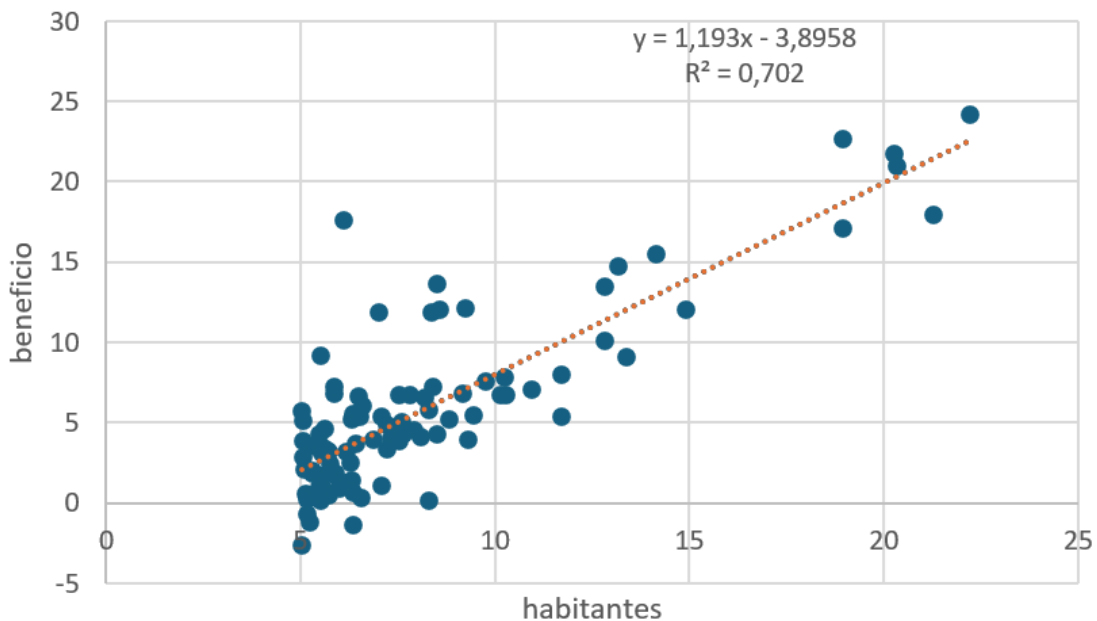
2. Representación gráfica de los datos:

Para representar gráficamente los ejemplos hemos hecho uso de la librería matplotlib de Python.



3. Aplicación Externa: Excel

Mediante la aplicación podemos obtener los valores de los coeficientes de regresión lineal y la representación grafica de los datos y la línea de tendencia.



4. Ecuaciones normales (matrices):

En ecuaciones normales calculamos los coeficientes aplicando la definición analítica de tita, la formula matricial de las ecuaciones normales para obtener las distintas titas es la siguiente:

$$\theta = (X^t X)^{-1} X^t y$$

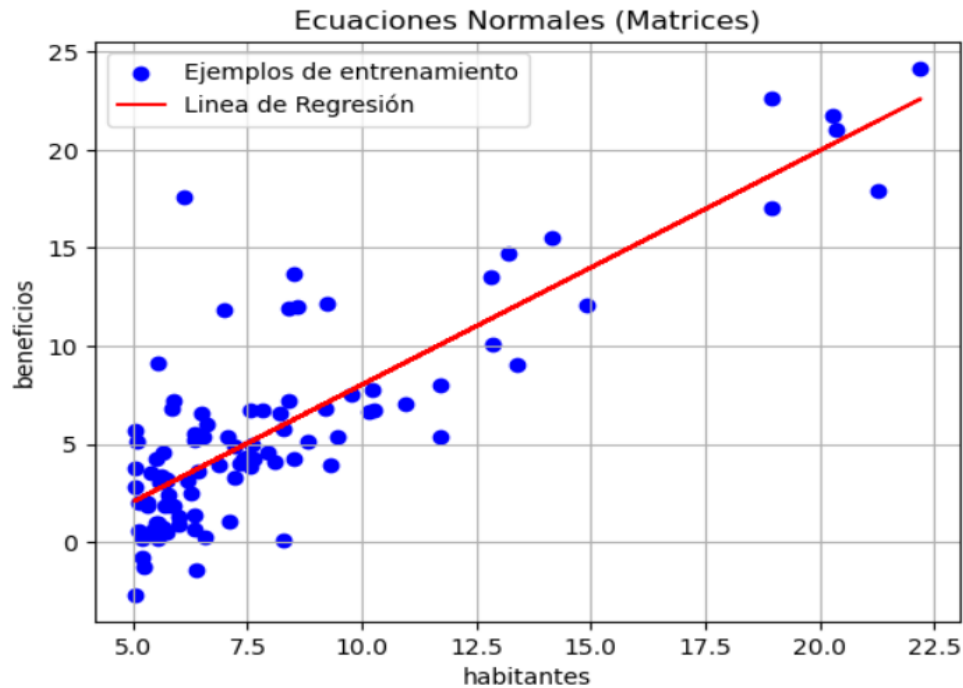
A continuación, vemos el desarrollo de la formula anterior en lenguaje Python:

```
# leemos los datos del fichero
datos=panda.read_csv("regresion_1.csv",header=None)
# extraemos los valores de atributos (habitantes) y los objetivos (beneficio)
x=datos.iloc[:,0]
y= datos.iloc[:,1]
# obtener los valores de entrada y salida en formato columna para hacer el calculo de la ecuacion normal
X = np.matrix(x).T
y = np.matrix(y).T
# añadir una columna de unos al matriz X para el termino independiente (tita_0)
X_b = np.hstack([np.ones((X.shape[0], 1)), X])
tita = np.linalg.inv(X_b.T * X_b) * (X_b.T * y)
```

La representación grafica de los datos con la línea de regresión es la siguiente:

Coefficientes de la regresión:

Intercepto (tita_0): -3.8958, Pendiente (tita_1): 1.1930



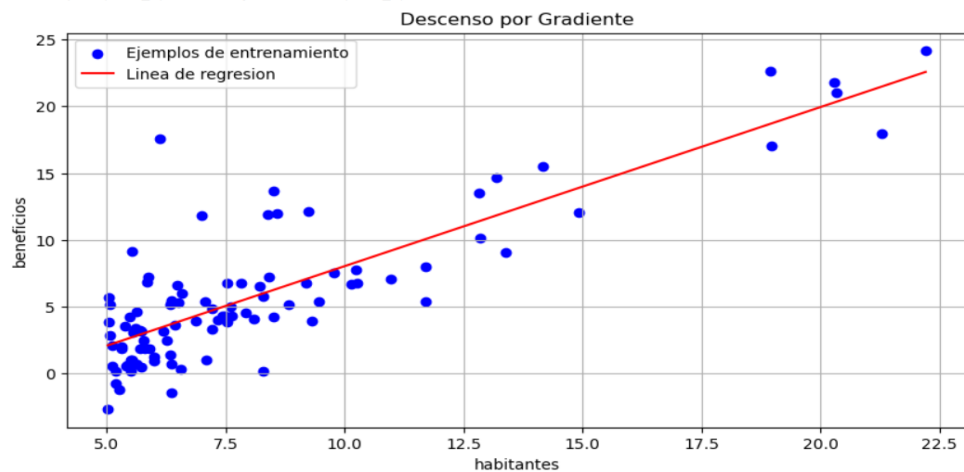
5. Descenso por gradiente:

Este algoritmo se trata de un algoritmo iterativo que consiste en encontrar los valores de los coeficientes de la regresión mediante la minimización del error.

En la siguiente grafica observamos los valores de los coeficientes con la recta de la regresión:

Coefficientes de la regresión:

Intercepto (tita_0): -3.8780, Pendiente (tita_1): 1.1913

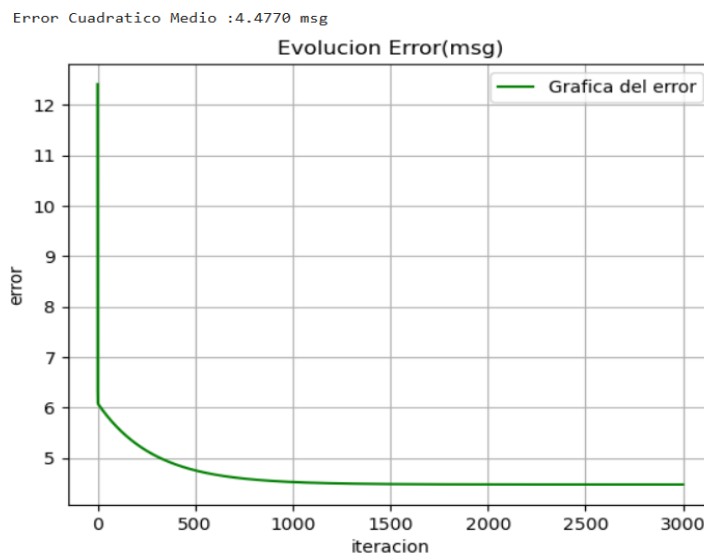


En la gráfica anterior, vemos los resultados de la regresión, calculados a partir de una ratio de aprendizaje equivalente a 0.0001 escogido a mano entre 3 distintos valores y un criterio de convergencia de 3000 iteraciones.

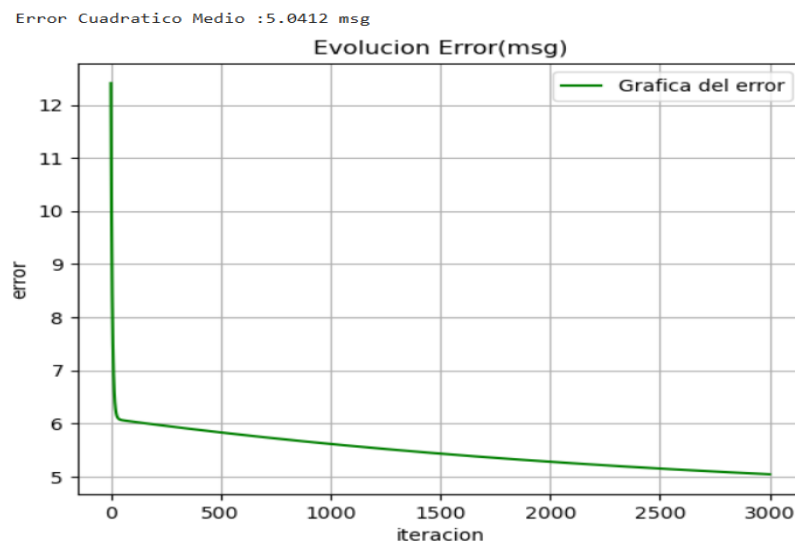
Por otro lado, en las siguientes graficas observamos la evolución del error con distintos valores de la tasa de aprendizaje, para ello, se ha hecho el uso de la siguiente fórmula para el cálculo del error total cuadrático.

$$J(\theta) = \frac{1}{2} \sum_{j=1}^m (h_{\theta}(x^j) - y^j)^2$$

1. Grafica del error con un valor de alfa equivalente a 0.0001:

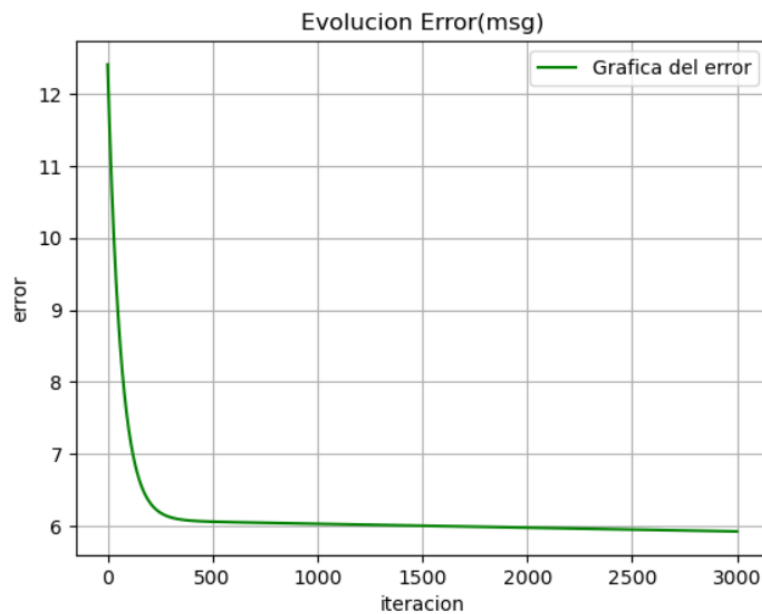


2. Grafica del error con un valor de alfa equivalente a 0.00001:



3. Grafica del error con un valor de alfa equivalente a 0.000001:

Error Cuadratico Medio :5.9271 msg



Como hemos visto en los resultados de las gráficas anteriores, la evolución del error cuadrático medio según el numero de iteraciones del criterio de parada, también observamos que no todos los valores de la tasa de aprendizaje proporcionan buena aproximación, por lo que hemos escogido el primero ya que es el que obtiene un ajuste mejor de los tres.

A continuación, adjuntamos el código que desarrolla el algoritmo:

```
alfa = 0.0001 # inicializamos la tasa de aprendizaje
conj=[123456789,234567890,345678900,456789000,567890000] # conjuntos que contiene las semillas que usaremos
resultados_error = []
resultados_tita_0 = []
resultados_tita_1 = []
for k in conj:
    # diferentes semillas
    random.seed(k)
    # inicializamos las titas a valores aleatorios entre 0 y 1
    tita_0 = random.random()
    tita_1 = random.random()
    conjErrores = []
    conj_tita_0 = []
    conj_tita_1 = []
    for i in range(3000): # numero de iteraciones para el criterio de parada (3000 iteraciones hasta que converga)
        sum_0, sum_1,error = 0, 0, 0
        for j in range(len(x)): # calculamos el sumatorio del gradiente y lo multiplicamos por el alpha
            h_x = tita_0 + tita_1 * x.iloc[j]
            sum_0 += (h_x - y.iloc[j])
            sum_1 += ((h_x - y.iloc[j]) * x.iloc[j])
            error += (h_x - y.iloc[j]) ** 2
        # modificamos los coeficientes restando el valor obtenido para que decrezca las titas en sentido contrario
        tita_0 -= alfa * sum_0
        tita_1 -= alfa * sum_1
        # añadimos los datos (valores de los coeficientes y el error) a los conjuntos
        conjErrores.append(0.5 * error / len(x))
        conj_tita_0.append(tita_0)
        conj_tita_1.append(tita_1)
    # calculamos el valor que minimiza el error para obtener los coeficientes que le corresponden
    mini = min(conjErrores)
    ind = conjErrores.index(mini)
    resultados_error.append(mini)
    resultados_tita_0.append(conj_tita_0[ind])
    resultados_tita_1.append(conj_tita_1[ind])
# obtenemos los valores de los coeficientes que minimizan el error de los 5 valores correspondientes a las distintas semillas
ind_2 = resultados_error.index(min(resultados_error))
coeficiente_0 = resultados_tita_0[ind_2]
coeficiente_1 = resultados_tita_1[ind_2]
pintar_error(conjErrores)
print(f"Coeficientes de la regresión: \nIntercepto (tita_0): {coeficiente_0:.4f}, Pendiente (tita_1): {coeficiente_1:.4f}")
plt.figure(figsize=(10, 5))
plt.scatter(x, y, color='blue', label='Ejemplos de entrenamiento')
x_linea = np.linspace(min(x), max(x), 1000)
y_linea = coeficiente_0 + coeficiente_1 * x_linea
plt.plot(x_linea, y_linea, color='red', label='Línea de regresión')

plt.title('Descenso por Gradiente')
plt.xlabel('habitantes')
plt.ylabel('beneficios')
plt.legend()
plt.grid(True)
plt.show()
```

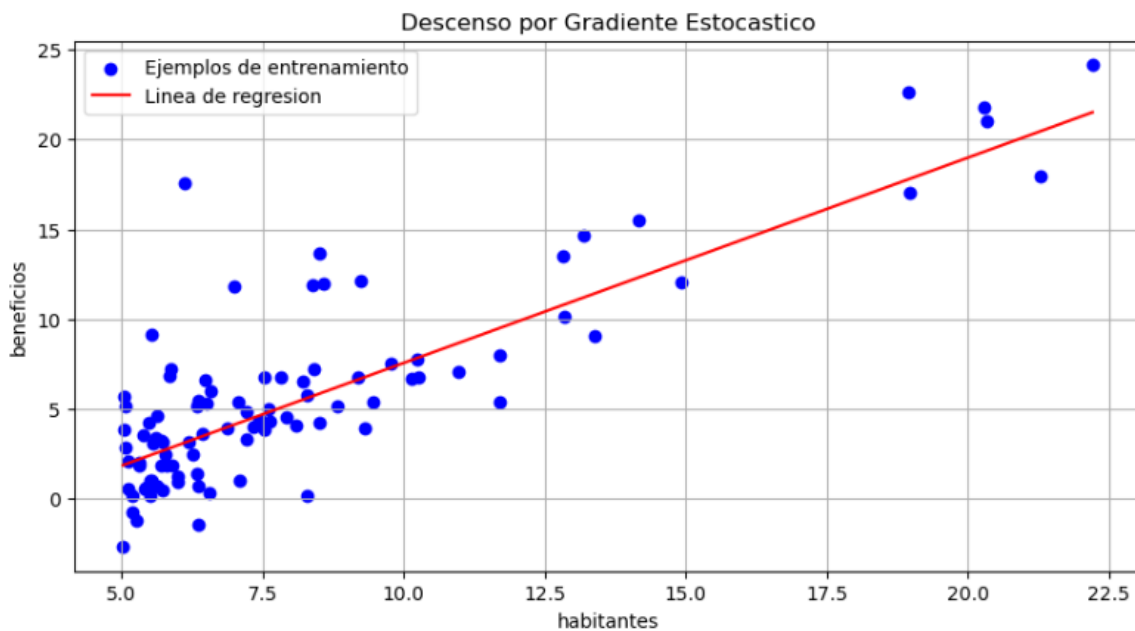

6. Descenso por gradiente estocástico:

El concepto estocástico consiste en calcular mediante la formula del gradiente los valores de la regresión sobre un numero de ejemplos (batch) elegidos al azar no sobre todos los ejemplos, eso implica que alcanza la convergencia mucho antes que el descenso por gradiente, este algoritmo nos propone un solución, pero no la óptima dado que escoge los datos al azar.

En la siguiente grafica observamos los valores de los coeficientes con la recta de la regresión:

Coefficientes de la regresión:

Intercepto (tita_0): -3.8755, Pendiente (tita_1): 1.1433



El código desarrollado es el siguiente:

```

# Descenso por gradiente estocastico
alfa = 0.0001 # inicializamos la tasa de aprendizaje
conj=[123456789,234567890,345678900,456789000,567890000] # conjuntos que contiene las semillas que usaremos
resultados_error = []
resultados_tita_0 = []
resultados_tita_1 = []
batch = 50 # numero de valores que usamos para el calculo de la formula del gradiente estocastico
for k in conj:
    # diferentes semillas
    random.seed(k)
    # inicializamos las titas a valores aleatorios entre 0 y 1
    tita_0 = random.random()
    tita_1 = random.random()
    conjErrores = []
    conj_tita_0 = []
    conj_tita_1 = []
    error = 0
    for i in range(5000): # numero de iteraciones para el criterio de parada (hasta que converga) al ser
        # aleatorio aumentamos el numero de iteraciones para ver si se puede tener mejor solucion
        sum_0, sum_1, error = 0, 0, 0
        conj_batch=[]
        for h in range(batch): # obtener 10 datos de los ejemplos para el calculo del gradiente estocastico
            num = random.randint(0,len(x)-1)
            conj_batch.append((x.iloc[num], y.iloc[num]))
        for j in conj_batch: # calcular el sumatorio de gradiente estocastico del conjunto obtenido antes
            h_x = tita_0 + tita_1 * j[0]
            sum_0 += (h_x - j[1])
            sum_1 += ((h_x - j[1]) * j[0])
            error += (h_x - j[1]) ** 2
        # modificamos los coeficientes restando el valor obtenido para que decrezca las titas en sentido contrario
        tita_0 -= alfa * sum_0
        tita_1 -= alfa * sum_1
        # añadimos los datos(valores de los coeficientes y el error) a los conjuntos
        conjErrores.append(0.5 * error/batch)
        conj_tita_0.append(tita_0)
        conj_tita_1.append(tita_1)
    # calculamos el valor que minimiza el error para obtener los coeficientes que le corresponden
    mini = min(conjErrores)
    ind = conjErrores.index(mini)
    resultados_error.append(mini)
    resultados_tita_0.append(conj_tita_0[ind])
    resultados_tita_1.append(conj_tita_1[ind])
# obtenemos los valores de los coeficientes que minimizan el error de los 5 valores correspondientes a las distintas semillas
ind_2 = resultados_error.index(min(resultados_error))
coeficiente_0 = resultados_tita_0[ind_2]
coeficiente_1 = resultados_tita_1[ind_2]

```

7. Función de predicción:

Como resultado final íbamos buscando la predicción de beneficios en función de la población, para ello hemos diseñado la siguiente función donde le pasamos los valores de la regresión obtenidas mediante alguno de los apartados anteriores y el número de la población.

```

def calcular_Predicado(coeficiente1, coeficiente2, valor_entrada):
    return coeficiente1 + coeficiente2 * valor_entrada

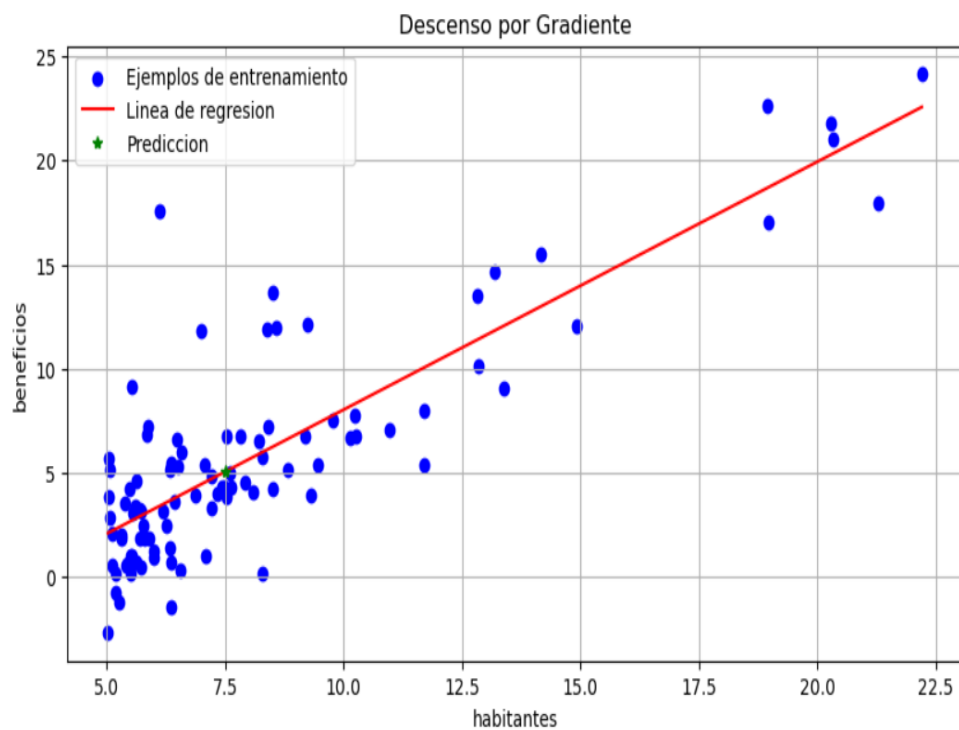
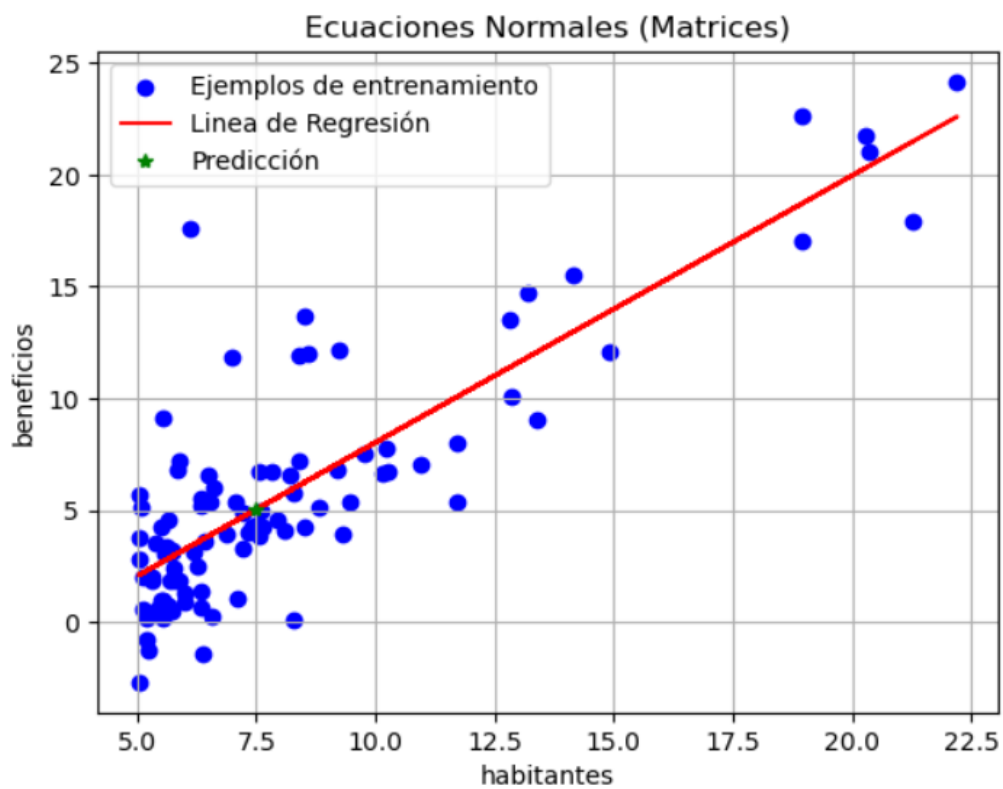
```

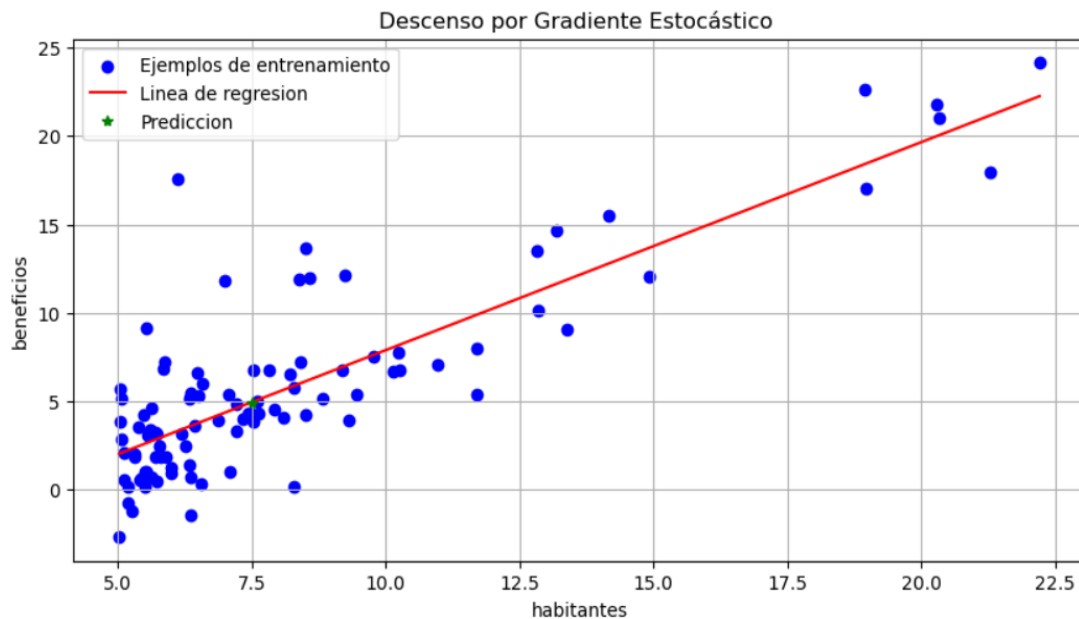
Ejemplos de la predicción con distintos valores:

El beneficio que corresponde a una población de 9.5 personas es : 7.4380 €

El beneficio que corresponde a una población de 67.8 personas es : 76.9919 €

8. Representación de punto con las líneas de los tres algoritmos:





Para $x = 7.5$, los tres algoritmos nos proporcionan un valor de predicción equivalente a 5 como se muestra en las figuras anteriores.

9. Pregunta 10 de la practica:

Para una población de 9.5 miles de personas el restaurante obtendría un beneficio equivalente es de 7.438 €.