Dataset.- <https://www.kaggle.com/datasets/harlfoxem/housesalesprediction>

Mediante las características de una casa, se puede llegar a predecir el precio estimado de una casa.

**Variables de entrada(X):**

* número de dormitorios
* número de baños
* pies cuadrados living
* Lote pies cuadrados
* pisos
* frente al mar
* vista
* condición
* calificación
* pies cuadrados arriba
* pies cuadrados sótano
* año construcción
* año renovado
* código postal
* latitud
* largo
* pies cuadrados living
* pies cuadrados lote

**Variable de salida(Y):**

* Predicción del precio de una vivienda

**Especificación de cada columna.-**

* **Primera columna(id).-** Un id que identifica a cada casa dentro del dataset.
* **Segunda columna (date).-** Fecha en la que se añadió la casa al dataset.
* **Tercera columna (bedrooms).-** Especifica el número de dormitorios que contiene la casa.
* **Cuarta columna (bathrooms).-**Especifica el número de dormitorios que contiene la casa.
* **Quinta columna (sqft\_living).-** Pies cuadrados habitables. Esta expresión se utiliza para indicar la medida del espacio interior utilizable en una vivienda o edificación.
* **Sexta columna (sqft\_lot).-** Pies cuadrados del lote. Este término se utiliza para indicar la medida del terreno o parcela en la que se encuentra una propiedad.
* **Séptima columna (floors).-** El número de pisos que contine la casa.
* **Octava columna (waterfront).-** significa que se encuentra en la línea de costa o a orillas de un cuerpo de agua. 1 para sí, y 0 para no.
* **Novena columna (view).-** La presencia de una buena vista puede tener un impacto significativo en el valor y atractivo de una propiedad. 1 si 0 no.
* **Decima columna (condition).-** Se refiere al estado general de una propiedad, es decir, a su condición física y estructural, del 1 al 5.
* **Onceava columna(grade).-** Se refiere al grado de posición de la casa.
* **Doceava columna (sqft\_above**).- Pies cuadrados arriba. Este término se utiliza para describir la medida del espacio habitable en una propiedad que está ubicado por encima del nivel del suelo
* **Treceava columna (sqft\_basement).-** Pies cuadrados del sótano. Este término se utiliza para describir la medida del espacio habitable que se encuentra en el sótano de una propiedad.
* **Catorceava columna (yr\_built).-** Refiere al año de construcción de una propiedad. Este término se utiliza para indicar el año en que la estructura principal de una casa, edificio o propiedad fue construida.
* **Quinceava columna (yr\_renovated).-** refiere al año en que una propiedad fue renovada.
* **Dieciseisava columna (zipcode).-** Es un término utilizado en los Estados Unidos para referirse al código postal de una determinada área geográfica.
* **Diecisieteava columna (lat).-** Generalmente se refiere a la latitud de una ubicación geográfica específica.
* **Dieciochoava columna (long).-** Generalmente se refiere a la longitud de una ubicación geográfica específica.
* **Diecinueveava columna (sqft\_living15).-** es un término utilizado en bienes raíces para describir la medida del espacio habitable de una propiedad específica, sugiere que la medida se refiere a la superficie habitable específica para el año 2015.
* **Veinteava columna (sqft\_lot15).-** es un término utilizado en bienes raíces para describir la medida del tamaño del terreno de una propiedad específica, se refiere al tamaño del terreno específico para el año 2015.

1. **regresión lineal multivariable**

Se implementa la regresión lineal multivariable para predecir el precio de las casas en USA. El archivo **`kc\_house\_data.csv`** contiene un conjunto de entrenamiento de precios de casas en el condado de King, Estado Unidos.

data = pd.read\_csv('kc\_house\_data.csv', *delimiter*=',')

Se retiro las columnas irrelevantes del dataset, como “date”, “id”.

dataset = data.drop(['id', 'date'], *axis*=1).astype(*float*)

Y se separo el precio en un vector aparte.

Una vez cargado los datos, y separando el 80% para entrenamiento y el 20% restante para las pruebas. Se procede a la muestra de datos:

train\_dataset, test\_dataset = train\_test\_split(dataset, *test\_size*=0.2, *random\_state*=42)

X\_regre\_test = test\_dataset.drop(['price'], *axis*=1).values

y\_regre\_test = test\_dataset['price'].values

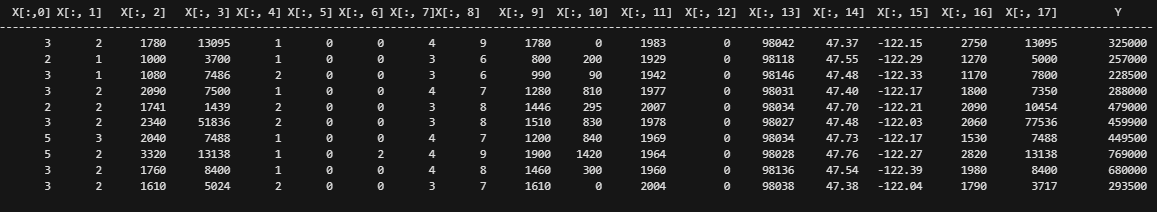
# Seleccionamos las columnas para X y la columna 'price' para y

X\_regre = train\_dataset.drop(['price'], *axis*=1).values

y\_regre = train\_dataset['price'].values

m\_regre = len(y\_regre)

En este caso solo se imprimió 10 ejemplos, esto debido a la gran cantidad de ejemplos.



* 1. **Normalización de características**

Al visualizar los datos se puede observar que las características tienen diferentes magnitudes, por lo cual se debe transformar cada valor en una escala de valores similares, esto con el fin de que el descenso por el gradiente pueda converger más rápidamente.

Se crea la función para normalizar:

*def*  featureNormalize(*X*):

    X\_norm = X.copy()

    mu = np.zeros(X.shape[1])

    sigma = np.zeros(X.shape[1])

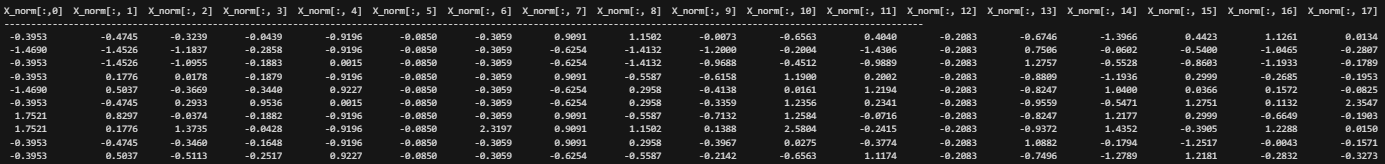
    mu = np.mean(X, *axis* = 0)

    sigma = np.std(X, *axis* = 0)

    X\_norm = (X - mu) / sigma

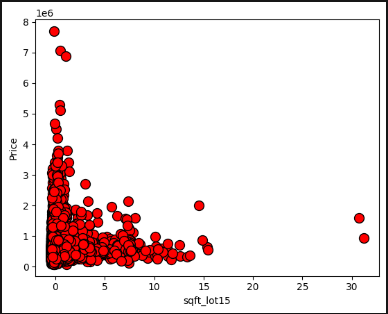
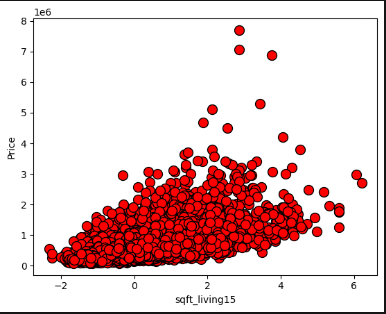
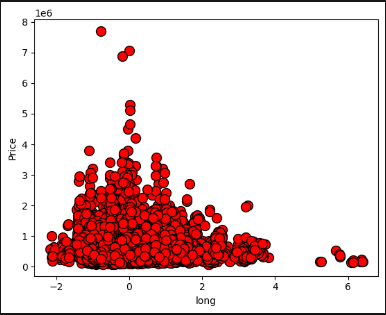
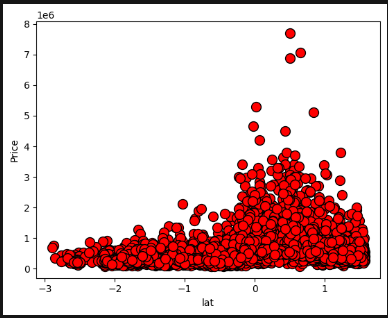
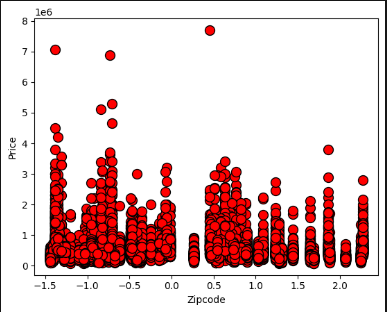
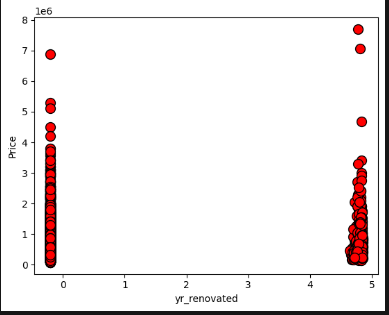
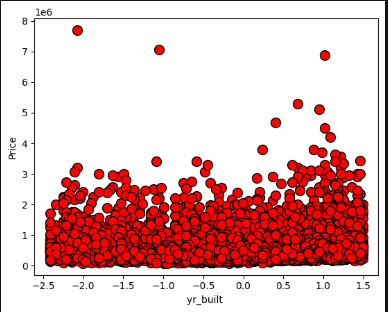
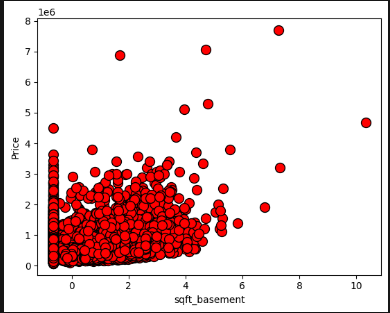
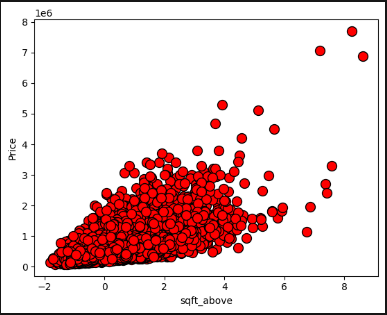
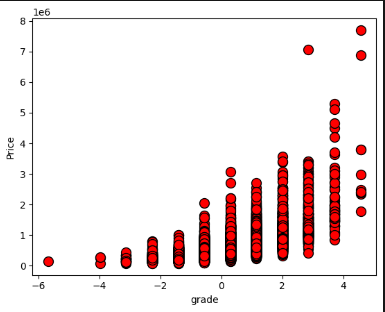
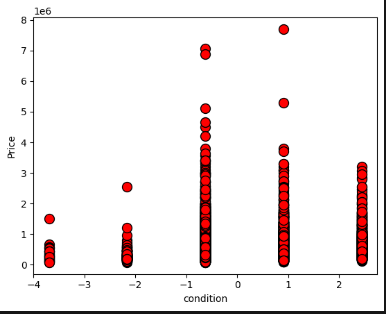
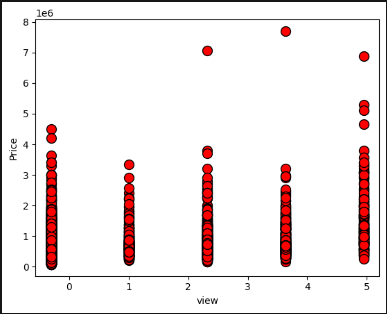
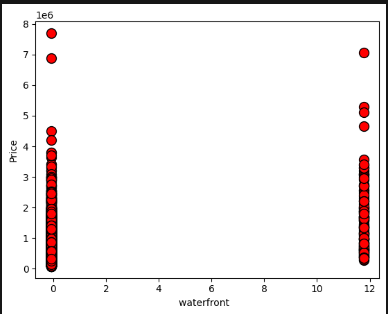
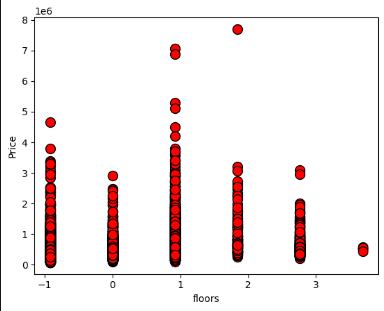
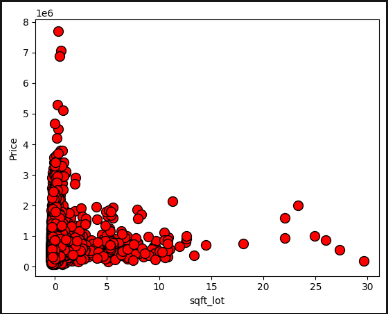
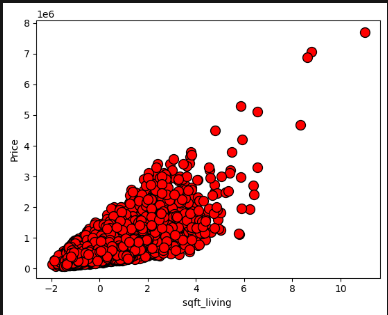
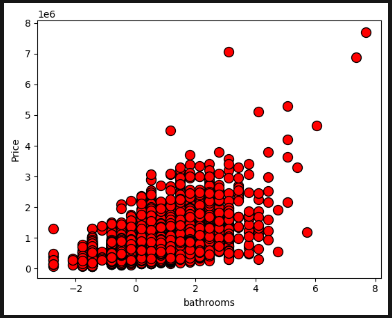
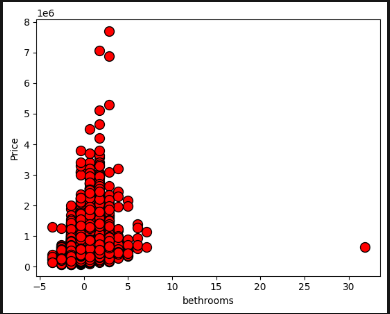
    return X\_norm, mu, sigma

mostrando la tabla con los datos normalizados:



* 1. Graficar las características

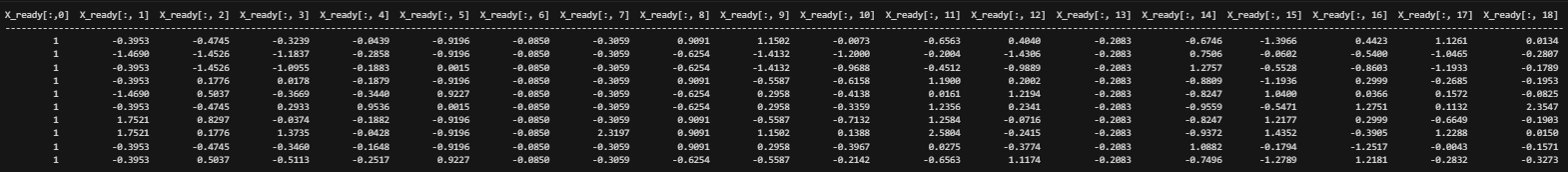
graficamos cada característica respecto a Y que es el precio, esto para ver la relacion entre estos.



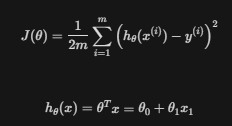
* 1. **Descenso por el gradiente**
     1. **Cálculo del costo(Jθ)**

Primero se añadió unos a la matriz X

Imprimiendo los datos normalizados con la columna de unos añadidos.



Creamos la función para el cálculo del costo:



*def* calcularCosto(*X*, *y*, *theta*):

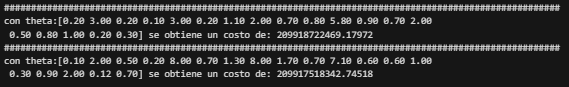
  m = y.size

  J = 0

  J = (1/(2 \* m)) \* np.sum(np.square(np.dot(X, theta) - y))

  return J

hacemos las pruebas para ver el funcionamiento de la función.



* + 1. **Descenso por el gradiente**

El costo Jθ esta parametrizado por el vector θ, no X e Y. Donde hay que minimizar el valor de Jθ cambiando los valores del vector θ. Una buena manera de verificar si el descenso por el gradiente está trabajando correctamente es ver los valores de Jθ y verificar si estos decrecen en cada paso.

Creamos la función para calcular el descenso por la gradiente y obtener un theta y J\_historico.

El cual de acuerdo al numero de iteraciones, y con un θ iniciado con ceros, este retornara J\_historico en de cada iteración, junto el vector de valores θ

*def* calcularDescensoGradiente(*X*, *y*, *theta*, *alpha*, *numero\_iteraciones*):

  m = y.shape[0]

  theta = theta.copy()

  J\_historico = []

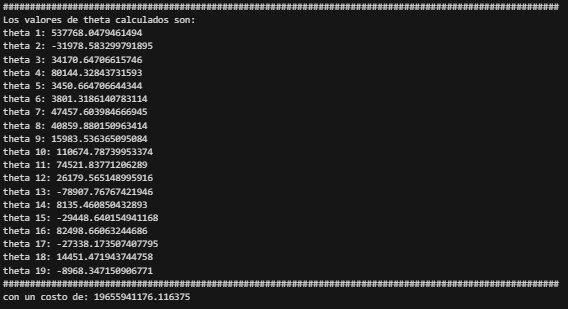
  for i in range(numero\_iteraciones):

    theta = theta - (alpha / m) \* (np.dot(X, theta) - y).dot(X)

    J\_historico.append(calcularCosto(X, y, theta))

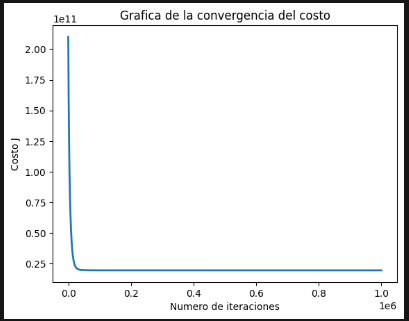
  return theta, J\_historico

Se inicializan los parámetros θ con 0 y la taza de aprendizaje α con 0.00009.



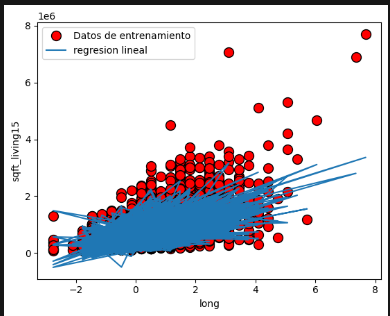
* 1. **Grafica de la convergencia del costo**

graficamos el costo



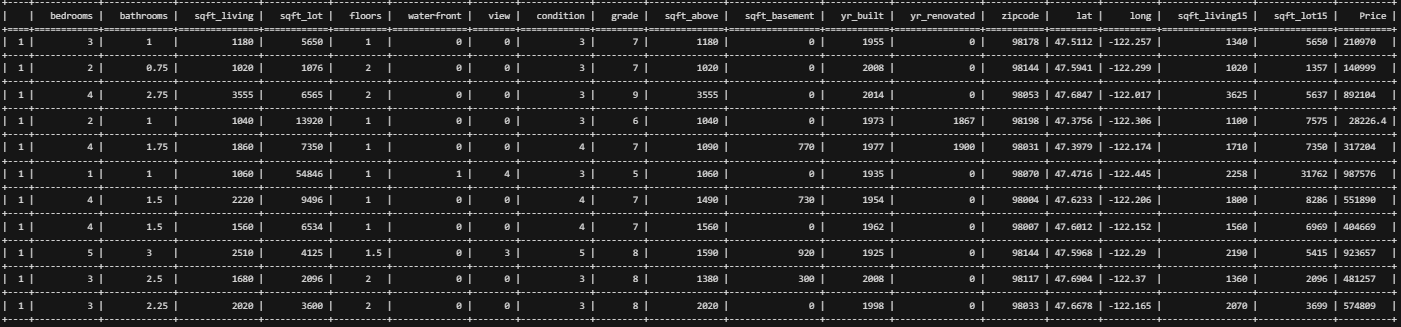
* + 1. **Grafica de la regresión lineal**

La grafica no muestra una linea recta, esto debido a los valores de θ, ya que en un cierto punto se disminuyen y luego incrementan.

****

* 1. **Ejemplos de Predicciones**

Se creo una matriz con 11 ejemplos, donde se hace las predicciones correspondientes:



* 1. **Validaciones**

Para hacer las validaciones correspondientes, primero se crea la función del **Mean squeared error**.

Siguiendo el consejo de 80/20, donde 80% es para la fase de entrenamiento, y 20% es para la fase de prueba.

Creando la funcion:

*def* mean\_squared\_error(*y\_pred*, *y\_actual*):

    resta = y\_pred - y\_actual

    err\_cuadrado = np.sum(resta \*\* 2)

    return err\_cuadrado / len(y\_pre)

haciendo cálculo del error cuadrático medio: se puede ver que el error cuadrático es elevado, esto debido a los valores calculados de θ



1. **Regresión Polinómica**

Se hizo **PolynomialFeatures** es una clase en **scikit-learn** que se utiliza para generar características polinómicas a partir de un conjunto de características existente. En el contexto de regresión polinómica, puedes usar **PolynomialFeatures** para generar nuevas características que son combinaciones polinómicas de las características originales.

Siguiendo los mismos pasos anteriores:

data = pd.read\_csv('kc\_house\_data.csv', *delimiter*=',')

#retiramos el id y la fecha de las columnas, quitamos el precio ya que ira a el vector Y, tambien quitamos los títulos

dataset = data.drop(['id', 'date'], *axis*=1).astype(*float*)

#Separando el 80% para entranamiento y el 20% para pruebas

train\_dataset, test\_dataset = train\_test\_split(dataset, *test\_size*=0.2, *random\_state*=42)

X\_poli\_test = test\_dataset.drop(['price'], *axis*=1).values

y\_poli\_test = test\_dataset['price'].values

en esta parte hacemos uso de la librería **PolynomialFeatures**:

#Indicamos de que grado será nuestro polinomio, en este caso de segundo grado

poly = PolynomialFeatures(*degree*=2)

# Seleccionamos las columnas para X y la columna 'price' para y, transformando la X a polinomial cuadrado

X\_poli = poly.fit\_transform(train\_dataset.drop(['price'], *axis*=1).values)

* 1. **Normalización de características**

Al visualizar los datos se puede observar que las características tienen diferentes magnitudes, por lo cual se debe transformar cada valor en una escala de valores similares, esto con el fin de que el descenso por el gradiente pueda converger más rápidamente.

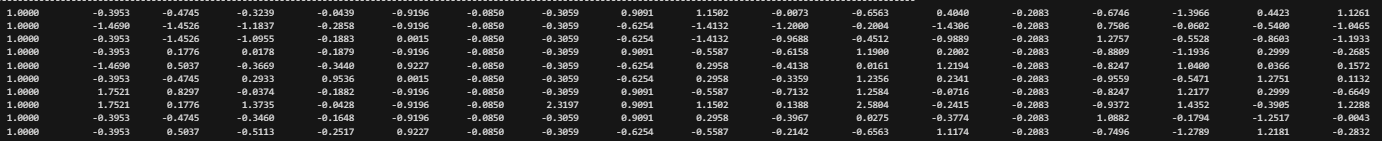
Se hizo uso de la función **featureNormalize().**

Solo se normalizo de la columna 1 en adelante, ya que estos ya cuentan con 1 en su primera columna, y no es necesario normalizar la primera columna.

X\_norm\_poli, mu\_poli, sigma\_poli = featureNormalize(X\_poli[:, 1:])

X\_poli\_ready = np.concatenate([np.ones((m\_poli, 1)), X\_norm\_poli], *axis*=1)

Mostrando algunas columnas, ya que este cuenta con 190 columnas, y más de 20000 ejemplos:



* 1. **Descenso por el gradiente**

Una vez ya tenido los datos listos y normalizados, se procede al cálculo del costo.

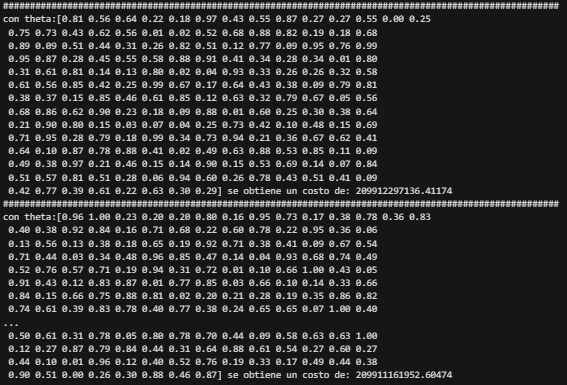
* + 1. **Cálculo del costo Jθ**

Ejemplo de funcionamiento de la función **calcularCosto** con dos valores diferentes de θ

theta\_poli = np.random.rand(len(X\_poli\_ready[1]))

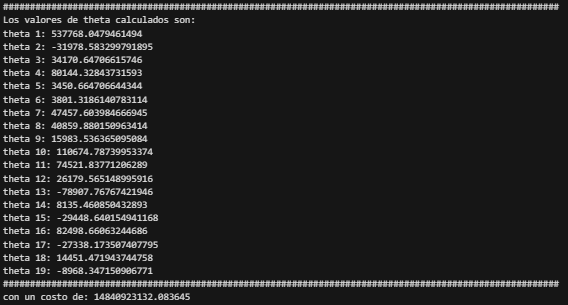
theta\_poli2 = np.random.rand(len(X\_poli\_ready[1]))

print(theta\_poli.shape[0])

****

* + 1. **Descenso por el gradiente**

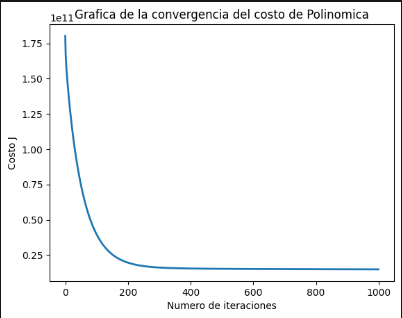
El costo Jθ esta parametrizado por el vector θ, no X y Y. Donde hay que minimizar el valor de θ cambiando los valores del vector θ. Una buena manera de verificar si el descenso por el gradiente está trabajando correctamente es ver los valores de Jθ y verificar si estos decrecen en cada paso.



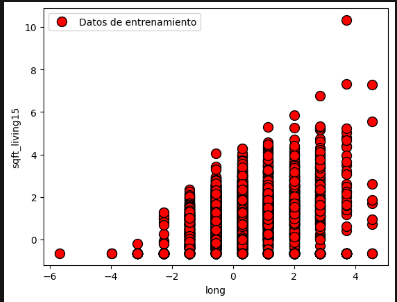
Se utilizan los parámetros finales para graficar la linea.

* 1. **Grafica de la convergencia del costo**

graficamos el costo:

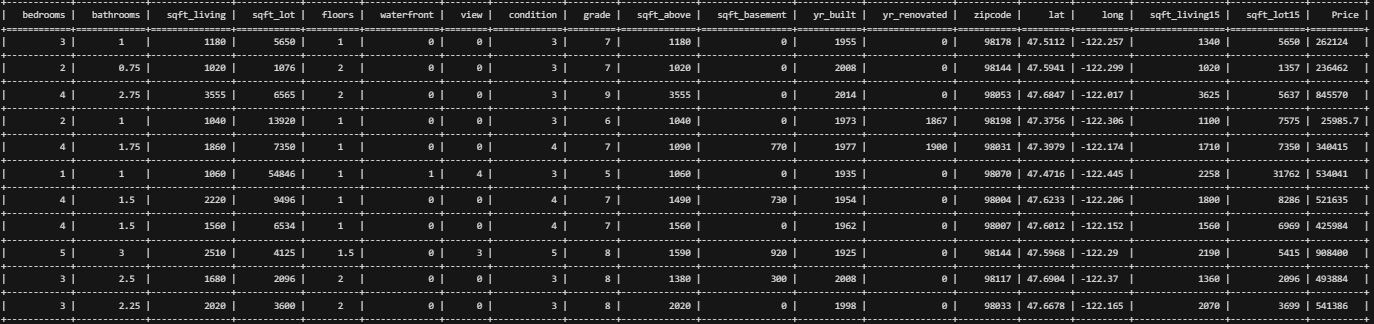


* 1. **Grafica de la regresión Polinómica**

****

* 1. **Ejemplos de Predicciones**

Se creo una matriz con 11 ejemplos, donde se hace las predicciones correspondientes:



2.6 Validaciones

Para hacer las validaciones correspondientes, primero se crea la función del \*\*Mean squeared error\*\*

Siguiendo el consejo de 80/20, donde 80% es para la fase de entrenamiento, y 20% es para la fase de prueba.

haciendo cálculo del error cuadrático medio: se puede ver que el error cuadrático es elevado, esto debido a los valores calculados de θ



1. **Ecuación de la normal**

Hacemos uso de la función de la Normal, haremos uso de los datos cargados en X para garantizar que las variables no estén modificadas, se debe agregar la columna de unos a la matriz $X$ para tener el termino de intersección