Wendy Dharmawan 2206059591 - Univariate Linear Regression Model : House Price Prediction

Model ini merupakan Model Univariate Linear Regression sederhana yang memperediksi harga rumah berdasarkan satu variable (univariate) yaitu luas dari rumah berdasarkan dataset berikut: https://www.kaggle.com/datasets/umernaeem217/synthetic-house-prices-univariate.

Model ini dibuat sebagai pengenalan dan pembelajaran machine learning, lebih spesifiknya regression model yang merupakan supervised model yang berfungsi untuk memprediksi variasi dari target atau output berdasarkan feature yang diberikan. Implementasi dilakukan melalui numpy, dan pandas.

Referensi pembelajaran dilakukan melalui course berikut:
https://www.coursera.org/learn/machine-learning

Importing

Import library yang digunakan berupa NumPy, pandas, matplotlib, dan tensorflow

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Import dataset yang digunakan menggunakan pandas

```
data = pd.read_csv('house_prices_dataset.csv',names=['Area','Price'])
```

Cek apakah data berhasil diimport

```
data.head()
#Jika muncul 5 buah data, maka data berhasil diimport
```

	Area	Price
0	area	price
1	2231.88	558852.17
2	2524.92	632260.29
3	1527.65	382994.25
4	1986.09	497607.86

Data Cleaning

Data pertama merupakan index sehingga dapat dihilangkan

```
data = data.iloc[1:] #atau data.drop(index=0)
#data iloc akan melakukan slicing dataframe, dalam hal ini hanya
dimulai dari index 1 dan seterusnya
```

Memeriksa data type dari tiap kolom

```
data.dtypes
```

```
Area     object
Price    object
dtype: object
```

Masih dalam bentuk object dan perlu di konversi menjadi float

```
data = data.astype(float)
data.dtypes #memeriksa kembali
```

```
Area     float64
Price    float64
dtype: object
```

Memeriksa kehilangan / ketidaklengkapan data

```
data.isna().sum()
#Dataset yang digunakan masih kecil dan hanya terdiri dari 2 kolom.
Dalam hal ini tidak terdapat data yang tidak lengkap.
```

```
Area     0
Price    0
dtype: int64
```

Memeriksa data duplikat

```
data.duplicated().sum()
#Dalam hal ini tidak terdapat data yang terduplikasi
```

```
0
```

Karena model merupakan supervised model, dataset perlu dibagi menjadi train data dan test data. Train data adalah bagian data yang digunakan untuk melatih model, sedangkan test data adalah data yang nantinya di uji dan dibandingkan predicted target (y hat) dan target sebenarnya (y) untuk memeriksa keakuratan model.

Digunakan rasio train-test sebesar 80:20

```
train_data = data.iloc[:80]

#train_data.head() #periksa apakah index dari data sudah benar
#train_data.tail() #periksa apakah index dari data sudah benar

test_data = data.iloc[80:]

#test_data.head()   #periksa apakah index dari data sudah benar
#test_data.tail()   #periksa apakah index dari data sudah benar
```

Statistik dari training dataset dapat ditampilkan dengan menggunakan data.describe

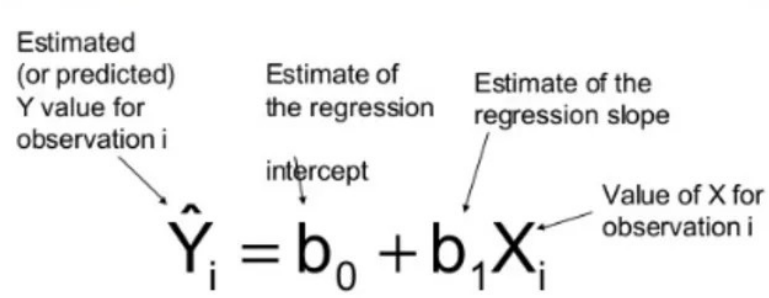
```
train_data.describe(include='all')
```

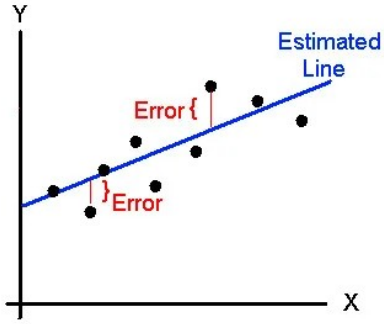
	Area	Price
count	80.000000	80.000000
mean	2261.508000	566369.386875
std	462.365718	115581.870734
min	1522.160000	381657.070000
25%	1824.862500	457256.095000
50%	2306.295000	577703.320000
75%	2656.030000	664948.062500
max	2995.400000	749686.490000

Pada dataset multivariate, normalization penting dilakukan untuk memastikan setiap feature (x) dari data memiliki range yang sama yaitu diantara 0 hingga +- 1. Ini dilakukan agar influence dari tiap feature sebanding dan tidak didominasi feature dengan nilai lebih besar. Namun dalam hal ini, normalization kurang dibutuhkan karena hanya beroperasi dengan 1 buah feature.

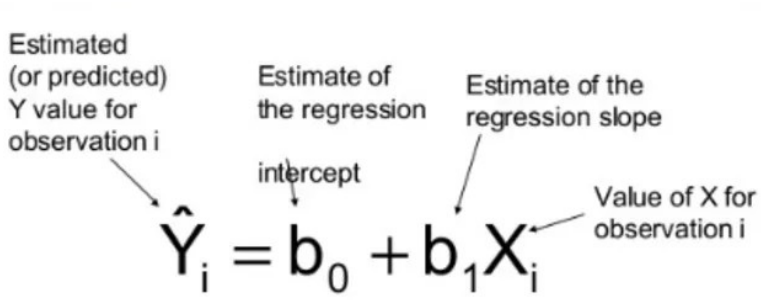
Pembuatan Model

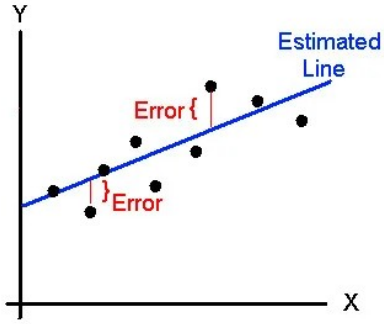
Karena model merupakan univariate model regression, function yang digunakan untuk model adalah Fw,b(x) = wx + b, dengan w sebagai weight dan b sebagai bias. Model bertujuan untuk mencapai loss sekecil mugkin. Nilai ini dapat diukur menggunakan cost function. Untuk model jenis linear regression, dapat digunakan MSE (mean squared errors) untuk mengkalkulasikan cost atau akurasi dari model. Persamaan dari MSE adalah sebagai berikut:





Untuk mencari nilai w dan b yang optimal, yaitu memiliki cost yang mendekati minima. Maka digunakan gradient descent algorithm. Cara kerja gradient descent algorithm adalah memilih initial point dari w dan b, lalu menemukan mengurangi nilai tersebut dengan hasil perkalian learning rate (alpha) dan turunan dari loss pada cost function dengan nilai w,b saat ini. Oleh karena itu, nilai learning rate memiliki pengaruh besar dalam gradient descent.

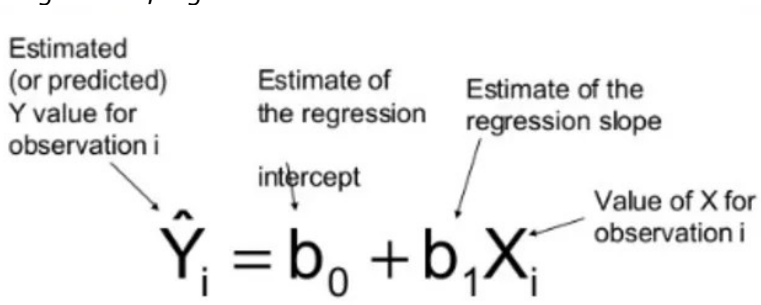


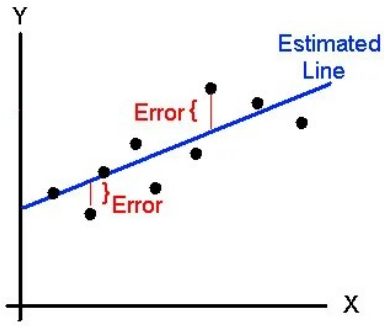


Terlebih dahulu perlu ditentukan nilai dari w dan b initial, beserta jumlah iterasi (epoch) yang akan dilakukan model, beserta learning rate (alpha) dari model. Konsiderasi dari nilai yang ditentukan adalah, bila learning rate terlalu kecil, akan membutuhkan iterasi yang banyak untuk mencapai minima, sedangkan bila learning rate besar, iterasi dapat berpotensi mengakibatkan data salah karena nilainya yang terlalu besar disaat proses gradient descent.

```
#nilai inisial w dan b
w = 0
b = 0

#nilai alpha dan jumlah iterasi (epoch)
learning_rate = 0.00000001
training_epochs = 10000

#jumlah data, diambil dari jumlah row dari train_data
m = train_data.shape[0]
```

Selanjutnya, membuat function yang digunakan oleh model. Karena model merupakan model linear regression, digunakan function berikut





Dari function ini, akan didapatkan predicted output (y^). Nilai dari predicted output dapat dibandingkan dengan target (y) menggunakan MSE untuk mendapatkan nilai cost. Persamaan fungsi dimasukkan melalui fungsi tensorflow.

```
def calculate_cost(train_data,w,b):
        cost = 0
        for i in range(m):
                f_wb = w * train_data['Area'].loc[data.index[i]] + b
                cost = cost + (f_wb -
train_data['Price'].loc[data.index[i]])**2
```

```
        total_cost = (1 / (2 * m)) * cost
        return total_cost
```

Setelahnya perlu dibuat function untuk menerapkan gradient descent. Output dari function berikut akan menghasilkan nilai w dan b yang baru setelah diapply gradient function.

```
def gradient_descent(train_data, w, b):

    dj_dw = 0
    dj_db = 0

    for i in range(m):
        f_wb = w * train_data['Area'].loc[data.index[i]] + b
        dj_dw_i = (f_wb - train_data['Price'].loc[data.index[i]]) *
train_data['Area'].loc[data.index[i]] #rumusan turunan parsial untuk
mendapatkan turunan j terhadap w
        dj_db_i = f_wb - train_data['Price'].loc[data.index[i]]
#rumusan turunan parsial untuk mendapatkan turunan j terhadap b

    #mencari rata-rata turunan dj terhadap dw dan db
        dj_db += dj_db_i
        dj_dw += dj_dw_i
    dj_dw = dj_dw / m
    dj_db = dj_db / m

    b = b - dj_db * learning_rate
    w = w - dj_dw * learning_rate

    return w , b
```

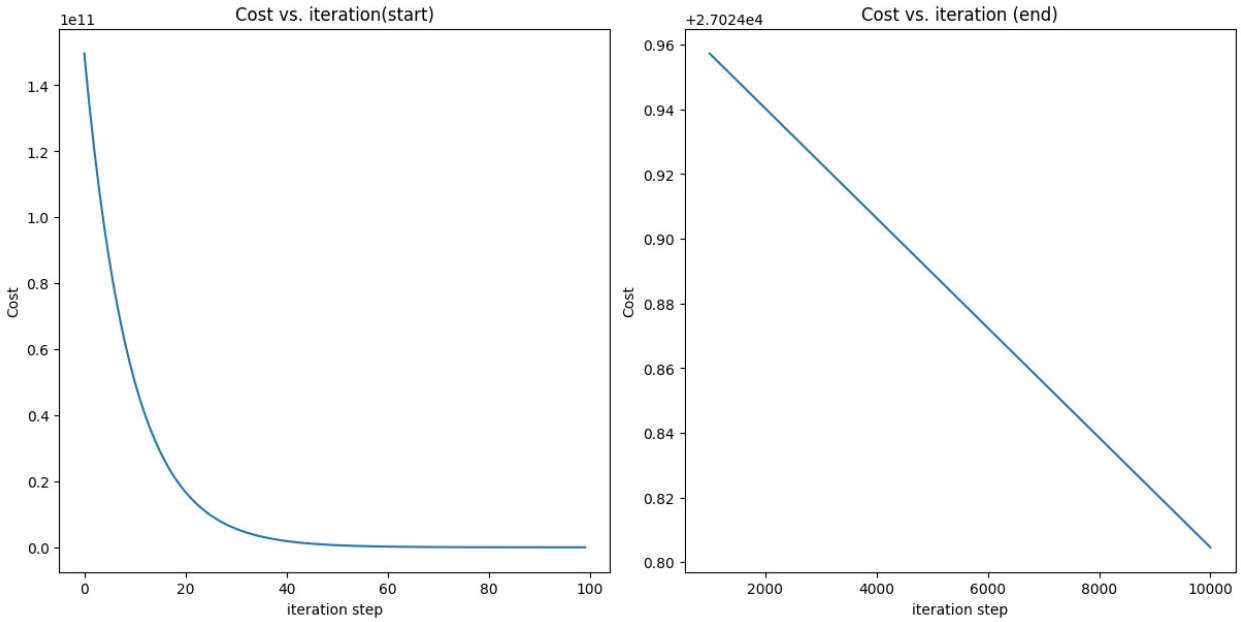
Lalu apply gradient descent sebanyak jumlah iterasi (epoch) yang ditentukan. Di setiap iterasi, masukkan cost dari setiap function dalam list agar dapat divisualisasikan dengan grafik nantinya.

```
cost_list = []
w_b_list = []

iteration = 1000
cost_list=[]
w_b_list=[]
for i in range(training_epochs):
    w,b = gradient_descent(train_data,w,b)
    cost = calculate_cost(train_data,w,b)
    cost_list.append(cost)

    #print cost dan nilai w,b setiap iterasi ke 'iteration' epoch
    if i % iteration == 0:
            print(f"Iteration {i:4}: Cost {cost:0.2e} ", f"w: {w:
0.3e}, b:{b: 0.5e}")
```

```
Iteration    0: Cost 1.50e+11  w:  1.334e+01, b: 5.66369e-03
Iteration 1000: Cost 2.70e+04  w:  2.504e+02, b: 1.06755e-01
Iteration 2000: Cost 2.70e+04  w:  2.504e+02, b: 1.07167e-01
Iteration 3000: Cost 2.70e+04  w:  2.504e+02, b: 1.07578e-01
Iteration 4000: Cost 2.70e+04  w:  2.504e+02, b: 1.07990e-01
Iteration 5000: Cost 2.70e+04  w:  2.504e+02, b: 1.08402e-01
Iteration 6000: Cost 2.70e+04  w:  2.504e+02, b: 1.08814e-01
Iteration 7000: Cost 2.70e+04  w:  2.504e+02, b: 1.09226e-01
Iteration 8000: Cost 2.70e+04  w:  2.504e+02, b: 1.09638e-01
Iteration 9000: Cost 2.70e+04  w:  2.504e+02, b: 1.10050e-01
```

Memvisualisasikan cost per setiap iterasi menggunakan matplotlib

```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True,
figsize=(12,6))
ax1.plot(cost_list[:100])
ax2.plot(iteration + np.arange(len(cost_list[iteration:])),
cost_list[iteration:])
ax1.set_title("Cost vs. iteration(start)");  ax2.set_title("Cost vs.
iteration (end)")
ax1.set_ylabel('Cost')
ax2.set_ylabel('Cost')
ax1.set_xlabel('iteration step')
ax2.set_xlabel('iteration step')
plt.show()
```



Pada percobaan training model dengan

- learning_rate = 0.00000001

- training_epochs = 10000 masih menghasilkan cost yang besar dinilai 2.70e+04. Dari grafik dapat dilihat bahwa cost menurun drastis pada 40 epoch pertama.

Dari percobaan training ini juga, telah didapatkan nilai dan b yang dapat mendapatkan cost local minimum yaitu

- w:2.504e+02
- b: 1.10050e-01 Dengan nilai ini, kita dapat melakukan prediksi output dengan fitur yang diberikan.

Prediksi

Kita dapat menggunakan sisa 20 data pada dataset yang ada untuk melakukan prediksi nilai y^ dan membandingkannya dengan nilai y sebenarnya untuk menguji keakuratan model

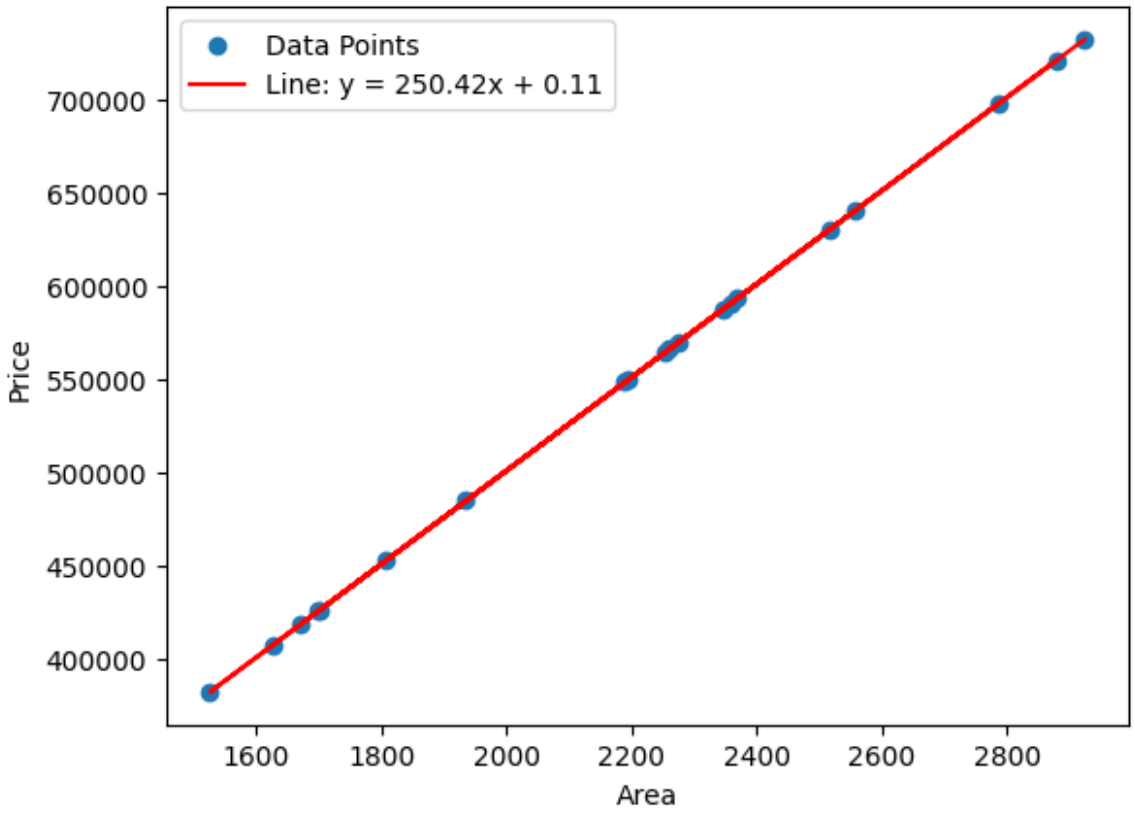
```
area = []
prediction =[]
target = []

n = test_data.shape[0]
for i in range (n):
    x = test_data['Area'].loc[data.index[m+i]]
    yhat = w*x + b
    y = test_data['Price'].loc[data.index[m+i]]
    area.append(x)
    prediction.append(yhat)
    target.append(y)
    print(f"{i}. Prediksi harga rumah seluas {x}: {yhat}, harga
aktual: {y}")
```

```
0. Prediksi harga rumah seluas 1626.72: 407364.22858412546, harga
aktual: 407559.15
1. Prediksi harga rumah seluas 2189.35: 548258.3429855529, harga
aktual: 548516.57
2. Prediksi harga rumah seluas 2253.37: 564290.266636939, harga
aktual: 564535.09
3. Prediksi harga rumah seluas 1698.64: 425374.47458549525, harga
aktual: 425571.13
4. Prediksi harga rumah seluas 1935.77: 484756.6997565854, harga
aktual: 484909.92
5. Prediksi harga rumah seluas 2557.51: 640453.1729058016, harga
aktual: 640315.12
6. Prediksi harga rumah seluas 1526.64: 382302.1398769914, harga
aktual: 382699.89
7. Prediksi harga rumah seluas 2785.27: 697488.9575174808, harga
aktual: 697484.01
8. Prediksi harga rumah seluas 2260.8: 566150.8913281262, harga
aktual: 566293.29
9. Prediksi harga rumah seluas 2367.9: 592970.9323006887, harga
aktual: 593131.88
```

```
10. Prediksi harga rumah seluas 2195.95: 549911.1186197164, harga
aktual: 550014.72
11. Prediksi harga rumah seluas 1806.84: 452469.97816375166, harga
aktual: 452748.03
12. Prediksi harga rumah seluas 1669.9: 418177.387960365, harga
aktual: 418496.26
13. Prediksi harga rumah seluas 2359.56: 590882.4249084275, harga
aktual: 590800.94
14. Prediksi harga rumah seluas 2517.89: 630531.5106898078, harga
aktual: 630290.91
15. Prediksi harga rumah seluas 2345.42: 587341.4783225075, harga
aktual: 587342.25
16. Prediksi harga rumah seluas 1701.95: 426203.3666080833, harga
aktual: 426321.81
17. Prediksi harga rumah seluas 2880.39: 721308.9602934859, harga
aktual: 721159.46
18. Prediksi harga rumah seluas 2273.6: 569356.2743762007, harga
aktual: 569238.36
19. Prediksi harga rumah seluas 2923.85: 732192.237423902, harga
aktual: 732054.33
```

Visualisasi harga prediksi dan harga aktual menggunakan matplotlib

```
plt.scatter(area,target, label='Data Points')
plt.plot(area, prediction, color='red', label='Line: y = {:.2f}x +
{:.2f}'.format(w,b))
plt.xlabel('Area')
plt.ylabel('Price')
plt.legend()
plt.show()
```



Terlihat bahwa model sudah cukup akurat dalam melakukan prediksi harga, meskipun memiliki nilai cost yang cukup tinggi. Terdapat kemungkinan bahwa ini disebabkan oleh underfitting atau overfitting data, dikarenakan dataset berukuran kecil dan bersifat univariate, atau algoritma MSE masih belum tepat untuk digunakan pada dataset tersebut dikarenakan adanya outlier data.