Assignment 2

Instructions:

- Type your answers in the spaces provided in this Word document. Your submission should not exceed 11 pages, including this page.
- Submit the *Declaration of Academic Integrity* before submitting your assignment.

Introduction

Given a set of data points with at least one predictor and one continuous response variable, we want to construct a linear model to predict the response. This is the aim of **Linear Regression**, which is a supervised learning technique.

In the context of this assignment, the transaction price of 30 flats in the same district of Singapore are collected. The data can be found in the file *housing_price.csv*.

The response variable is *price per square metre* (measured in $ in thousands), and the predictors are *inverse age of flat* (measured in year^{-1}) and *inverse distance to the nearest MRT station* (measured in km^{-1}). The *inverse age of flat* and *inverse distance to the nearest MRT station* are derived fields.

Simple Linear Regression (SLR)

We will first build a SLR model using *inverse distance to the nearest MRT station* as the predictor to predict *price per square metre*.

In SLR notations, let:

x_i = predictor value of the *i*-th data point

y_i = actual response value of the *i*-th data point

$\hat{y}_i$ = predicted response value of the *i*-th data point based on model

Thus, $\hat{y}_i = a + bx_i$, where values of *a* (intercept) and *b* (slope) are to be determined.

The squared-error of the *i*th prediction is $e_i^2 = (y_i - \hat{y}_i)^2$. Errors (also known as residuals) are squared to remove the signs, so that errors of opposite signs do not cancel out each other, giving the false impression of small aggregated errors.

Then, we define **Error function** as the mean of squared-error (of the whole data set):

$$E(a,b) = \frac{1}{n}\sum_{i=1}^{n} e_i^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$= \frac{1}{n}[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \cdots + (y_n - \hat{y}_n)^2]$$

We want to find the values of *a* and *b* such that the Error function is **minimised**.
The resultant equation $\hat{y} = a + bx$ will give the best-fit line that passes through the data points.

MODEL 1: SLR with intercept *a* fixed $\Rightarrow \hat{y}_i = bx_i$ (25 marks)

We will first build a SLR model to predict *price per square metre* (y) using *inverse distance to the nearest MRT station* (x) as the predictor.
Suppose it is believed that price is proportional to inverse distance. This means that $\hat{y}$ is a constant multiple of x and $a = 0$. Then, in the SLR model, we will only need to determine slope b.

(a) Express Error function $E(b)$ in terms of b only. Hence, derive $E'(b)$.

If a $= 0$, $\hat{y} = 0 + bx$, thus $\hat{y} = bx$

Simplifying the function:

$$E(b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$E(b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - bx_i)^2$$

$$E(b) = \frac{1}{n}(y_1 - bx_1)^2 + \frac{1}{n}(y_2 - bx_2)^2 + \cdots + \frac{1}{n}(y_n - bx_n)^2$$

Chain Rule:

$$E'(b) = \frac{-2x_1}{n}(y_1 - bx_1) + \frac{-2x_2}{n}(y_2 - bx_2) + \cdots + \frac{-2x_n}{n}(y_n - bx_n)$$

$$E'(b) = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - bx_i)$$

Express in terms of b only using SymPy:

$$E(b) = 52.9560529666666b^2 - 86.1205782b + 54.4808258$$

$$E'(b) = 105.912105933333b - 86.1205782$$

(b) Use univariate gradient descent algorithm to find the value of b for which $E(b)$ is at its minimum. Write your Python code in a single cell and copy-paste your code below.

```
b = 0.9 # Starting value of b
alpha = 0.01 # Learning rate
epsilon = 0.001 # Stopping criterion
max_iters = 1000 # Maximum number of iterations


def E(b): # calculate error function E(b)
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        summation += (y - b*x)**2
    return 1/len(df) * summation


def E_prime(b): # 1st derivative of E(b)
```

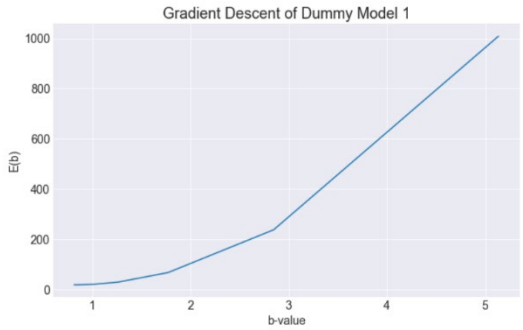
```
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        summation += x * (y - b*x)
    return -2/len(df) * summation

for iter in range(max_iters): # gradient descent
    new_b = b - alpha * E_prime(b) # update of b
    print(f"Iteration {iter + 1}: b-value = {new_b}, E(b) = {E(new_b)}")
    diff = abs(E(new_b) - E(b)) # stopping criterion
    if diff < epsilon: # break if converge
        print(f"The local minimum occurs at b = {new_b}")
        break
    b = new_b
```

(c) Describe the changes and decisions you made on the parameters for your solution to reach convergence.

Parameters: max iterations, epsilon, learning rate (alpha), Initial b-value (slope)

Initially, I tested out the gradient descent algorithm on initial b-value=10, alpha=0.005, epsilon=0.001 and 1000 max iterations. It took 11 iterations to converge and got a Mean-Squared Error (MSE) of 19.467. The visualization of the Gradient Descent process of the initial model is shown below.



I then tried varying starting b-values from -5000 to 5000 to check if the model would converge at different MSE values for varying starting b-values. The top 5 lowest MSE scores are shown below.

alpha	Starting b-value	Iterations	b	E(b)
0.01	500.0	6.0	0.813154	19.467104
0.01	-500.0	6.0	0.813111	19.467104
0.01	5.0	4.0	0.813184	19.467104
0.01	-5.0	4.0	0.813062	19.467104
0.01	0.0	3.0	0.813301	19.467106

It seems that for varying b-values, the model will still converge at MSE of 19.467 (3d.p). I then tried to achieve a lower number of iterations to converge since it is a simple linear regression model and should not take too many iterations. I then tried out starting b-values of 0 to 5 with increments of 0.1 combined with 3 different learning rates (0.02, 0.01, 0.005). The top 5 lowest MSE scores are also shown below.

alpha	Starting b-value	Iterations	b	E(b)
0.01	2.1	4.0	0.813148	19.467104
0.01	2.2	4.0	0.813149	19.467104
0.01	0.9	3.0	0.813115	19.467104
0.01	2.3	4.0	0.813151	19.467104
0.01	2.4	4.0	0.813152	19.467104

The top 5 lowest MSE scores are also the same at 6 decimal places at 19.467104. It seems that a starting b-value of 0.9 and alpha of 0.01 (in the 3rd row) will give the lowest number of iterations of 3, thus its parameters would be used for the final model 1.

Hence, alpha=0.01 epsilon=0.001 and starting b-value=0.9 is used for the final Model 2, which converged in 3 iterations to obtain a MSE of 19.467104.

(d) Describe your MODEL 1 by filling the information below.

Final MODEL 1 equation is: $\hat{y} = 0.81311x$ (5s.f)

Minimum value of Error function is: 19.467 (5s.f)

Number of iterations ran to reach convergence: 3

MODEL 2: SLR $\Rightarrow \hat{y}_i = a + bx_i$ (25 marks)

Now we apply the SLR model where both intercept *a* and slope *b* are to be determined, when predicting *price per square metre* (*y*) using *inverse distance to the nearest MRT station* (*x*) as the predictor.

(a) Express Error function $E(a, b)$ in terms of *a* and *b*. Hence, derive $E_a(a, b)$ and $E_b(a, b)$.

Equation $\hat{y} = a + bx$

Express error function in terms of a and *b*,

$$E(a,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$E(a,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (a + bx_i))^2$$

$$E(a,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - a - bx_i)^2$$

$$E(a,b) = \frac{1}{n}(y_1 - a - bx_1)^2 + \frac{1}{n}(y_2 - a - bx_2)^2 + \cdots + \frac{1}{n}(y_n - a - bx_n)^2$$

Derive $E_a(a, b)$, Chain Rule:

$$E_a(a,b) = \frac{-2}{n}(y_1 - a - bx_1) + \frac{-2}{n}(y_2 - a - bx_2) + \cdots + \frac{-2}{n}(y_n - a - bx_n)$$

$$E_a(a,b) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - a - bx_i)$$

Derive $E_b(a,b)$, Chain Rule:

$$E_b(a,b) = \frac{-2x_1}{n}(y_1 - a - bx_1) + \frac{-2x_2}{n}(y_2 - a - bx_2) + \cdots + \frac{-2x_n}{n}(y_n - a - bx_n)$$

$$E_b(a,b) = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - a - bx_i)$$

Express in terms of a and b only:

$$E(a,b) = a^2 + 9.48166666666667ab - 14.3246666666667a + 52.9560529666666b^2 - 86.1205782b + 54.4808258$$

$$E_a(a,b) = 2a + 9.48166666666667b - 14.3246666666667$$

$$E_b(a,b) = 9.48166666666667a + 105.912105933333b - 86.1205782$$

(b) Use gradient descent algorithm to find the values of a and b for which $E(a,b)$ is at its minimum. Write your Python code in a single cell and copy-paste your code below.

```
a = 5.8 # Starting value of a
b = 0.9 # Starting value of b
alpha = 0.01 # Learning rate
epsilon = 0.0001 # Stopping criterion
max_iters = 1000 # Maximum number of iterations

def E(a, b): # calculate error function E(a, b)
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        summation += (y - a - b*x)**2
    return 1/len(df) * summation

def partial_E_a(a, b): # 1st derivative of E(a, b) with respect to a
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        summation += y - a - b*x
    return -2/len(df) * summation

def partial_E_b(a, b): # 1st derivative of E(a, b) with respect to b
    summation = 0
```

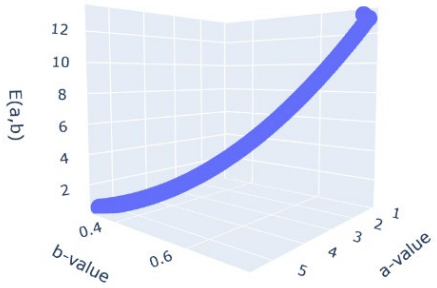
```
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        summation += x * (y - a - b*x)
    return -2/len(df) * summation

for iter in range(max_iters): # Gradient Descent
    new_a = a - alpha * partial_E_a(a, b) # update of a
    new_b = b - alpha * partial_E_b(a, b) # update of b
    diff = abs(E(new_a, new_b) - E(a, b)) # stopping criterion
    print(f"Iteration {iter+1}: a = {new_a}, b = {new_b}, E(a,b) = {E(new_a, new_b)}")
    if diff < epsilon:
        print(f"The local minimum occurs at a = {new_a}, b = {new_b}")
        break
    a = new_a
    b = new_b
```

(c) Describe the changes and decisions you made on the parameters for your solution to reach convergence.

Parameters: max iterations, epsilon, learning rate (alpha), initial a-value (intercept), initial b-value (slope)

For the initial model, I tried starting a and b-values of 1, alpha=0.01, epsilon=0.001 and max iterations of 1000. It took 249 iterations to converge with a MSE of 0.505 (3s.f). As it is a Simple Linear Regression (SLR) model, it should not take too many iterations to converge, so I proceeded to try different parameters to achieve a lower MSE as well as lower number of iterations. The gradient descent visualization of the initial model is shown below.



I then proceeded to try various a and b values from -5000 to 5000, alpha values of 0.02, 0.01, 0.005 and epsilon values of 0.001 and 0.0001. The top 5 lowest MSE scores all used alpha=0.01 and epsilon=0.0001. All the top 5 scores have an MSE of 0.4663 (4s.f). However, the number of iterations is still high for a SLR model, with the lowest number of iterations being 340 in the top 5 scores.

alpha	epsilon	iterations	starting a	starting b	E(a,b)
0.01	0.0001	763.0	-500.0	500.0	0.466296
0.01	0.0001	745.0	10.0	5000.0	0.466299
0.01	0.0001	600.0	0.0	-1000.0	0.466300
0.01	0.0001	340.0	10.0	0.0	0.466301
0.01	0.0001	611.0	0.0	1000.0	0.466302

To try to bring down the number of iterations, I used alpha=0.01 and epsilon=0.0001 to test out different starting a and b values. The top 5 lowest MSE results are shown below. All 5 different models took 4 iterations to converge, hence I will be taking the parameters of the model with the lowest MSE

score as my final model, with a starting a=5.8 and b=0.9.

alpha	epsilon	iterations	starting a	starting b	E(a,b)
0.01	0.0001	4.0	5.8	0.9	0.462090
0.01	0.0001	4.0	5.8	0.8	0.462127
0.01	0.0001	4.0	5.8	0.7	0.462248
0.01	0.0001	4.0	5.7	0.0	0.462279
0.01	0.0001	4.0	5.8	0.6	0.462454

Thus, alpha=0.01 epsilon=0.0001, starting a-value=5.8 and starting b-value=0.9 is used for the final Model 2, which converged in 4 iterations to obtain a MSE of 0.462090.

(d) Describe your MODEL 2 by filling the information below.

Final MODEL 2 equation is: $\hat{y} = 5.7456 + 0.29878x$ (5s.f)

Minimum value of Error function is: 0.46209 (5s.f)

Number of iterations ran to reach convergence: 4

Conclusion on SLR (15 marks)

(a) Using Python (or other software), in a single figure, plot the data points (scatterplot) together with the linear lines representing the two models. Insert the figure below and describe what you observe regarding the location of the data and the linear lines.

For Model 1, since the intercept is fixed at $a = 0$, the linear line only can pivot around the origin point to find the best fit. The line is the closest to the middle data point in the scatterplot, and the distance between the line and the points increases as the line is further away from the data point in the middle.
For Model 2, The distance between each point and the linear line is smaller in general compared to Model 1, which shows that Model 2 fit better to the dataset.

(b) In a linear regression model, the constant a is commonly interpreted as the value of the response variable when the predictor variable is zero. In your Model 2, can you interpret your value of a as such? Explain.

For Model 2, it would be impossible to interpret the response variable as the constant a when the predictor variable is zero.

The reason is that the predictor variable in Model 2 is defined as the inverse distance to nearest MRT station, and as $b^{-1} = 0$ does not exist, it is wrong to interpret the constant a as the response variable when the predictor is zero as it does not make sense.

MODEL 3: MLR $\Rightarrow \hat{y}_i = a + bx_i + cw_i$ (25 marks)

We can extend the SLR model to include more predictors. A linear regression model with more than 1 predictor is called **Multiple Linear Regression** (MLR) model.
Apply the MLR model where intercept a, and slopes b and c are to be determined, when predicting *price per square metre* (y) using *inverse distance to the nearest MRT station* (x) and *inverse age of flat* (w) as the predictors.

(a) Explain how gradient descent algorithm can be extended for MODEL 3.

To start off, gradient descent is an algorithm to find a minimum for a function, which in this case is the error function defined in the 1st page of this assignment. In a univariate gradient descent, as displayed in model 1, it is easy to understand as there is only 1 variable, b-value, and the goal was to find the b-value where E(b) is close or equal to 0, which is also called the local minima.
However, in a multivariate gradient descent as seen in model 3, the goal is to find the local minima of E(a, b, c) and find the corresponding a, b and c values. This is where partial derivatives can be used. Partial derivatives can be helps us to find the gradient of the tangent of E(a, b, c) at a given a, b or c value. However, 1 variable can only be focused at a time and the other 2 variables must be constant. So the gradient descent algorithm can be extended for model 3 by having partial derivatives for a, b and c, then iterate and update all a, b and c values simultaneously until E(a, b, c) reaches local minima.

(b) Use gradient descent algorithm to find the values of a, b and c for which Error function is at its minimum. Write your Python code in a single cell and copy-paste your code below.

```
a = 4.5 # Starting value of a
b = 1.4 # Starting value of b
c = 17 # Starting value of c
alpha = 0.01 # Learning rate
epsilon = 0.0001 # Stopping criterion
max_iters = 1000 # Maximum number of iterations

def E(a, b, c): # calculate error function E(a, b, c)
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        w = df.loc[rownum, 'inverse age of flat (year-1)']
        summation += (y - a - b*x - c*w)**2
    return 1/len(df) * summation

def partialE_a(a, b, c): # 1st derivative of E(a, b, c) with respect to a
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
```

```
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        w = df.loc[rownum, 'inverse age of flat (year-1)']
        summation += y - a - b*x - c*w
    return -2/len(df) * summation

def partialE_b(a, b, c): # 1st derivative of E(a, b, c) with respect to b
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        w = df.loc[rownum, 'inverse age of flat (year-1)']
        summation += x * (y - a - b*x - c*w)
    return -2/len(df) * summation

def partialE_c(a, b, c): # 1st derivative of E(a, b, c) with respect to c
    summation = 0
    for rownum in np.arange(len(df)):
        y = df.loc[rownum, 'price per square metre ($ in thousands)']
        x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
        w = df.loc[rownum, 'inverse age of flat (year-1)']
        summation += w * (y - a - b*x - c*w)
    return -2/len(df) * summation

for iter in range(max_iters): # Gradient Descent
    new_a = a - alpha * partialE_a(a, b, c) # update of a
    new_b = b - alpha * partialE_b(a, b, c) # update of b
    new_c = c - alpha * partialE_c(a, b, c) # update of c
    diff = abs(E(new_a, new_b, new_c) - E(a, b, c)) # stopping criterion
    print(f"Iteration {iter+1}: a = {new_a}, b = {new_b}, c= {new_c}, E(a,b,c) = {E(new_a, new_b,
new_c)}")
    if diff < epsilon:
        print(f"The local minimum occurs at a = {new_a}, b = {new_b}, c={new_c}")
        break
    a = new_a
    b = new_b
    c = new_c
```

(c) Describe the changes and decisions you made on the parameters for your solution to reach convergence.

Parameters: max iterations, epsilon, learning rate (alpha), initial a-value (intercept), initial b-value (slope 1), initial c-value (slope 2)

For the initial model, I tried using initial a, b and c-values of 1, with alpha=0.01, epsilon=0.001 and max iterations of 1000. The model converged in 247 iterations and had a MSE of 0.45428 (5s.f). To try to lower the number of iterations and reduce the MSE, I tried out different combinations of parameter values to tune the model.

Initially, I tried using varying a, b and c values of -500 to 500, alpha of 0.01 and 0.005, and epsilon of 0.001 and 0.0001. The top 5 lowest MSE and the parameters used are shown below. The MSE significantly went down after this round of tuning, with the lowest MSE being 0.165746 using alpha=0.01 and epsilon=0.0001. However, the number of iterations is still high, with the lowest number of iterations in the top 5 results being 441 iterations.

alpha	epsilon	iterations	starting a	starting b	starting c	E(a,b,c)
0.010	0.0001	509.0	-10.0	-500.0	10.0	0.165746
0.005	0.0001	959.0	-10.0	-500.0	10.0	0.170130
0.010	0.0001	533.0	0.0	-500.0	10.0	0.170162
0.010	0.0001	552.0	10.0	-500.0	10.0	0.175843
0.010	0.0001	441.0	-10.0	-10.0	10.0	0.194464

I then tried another round of tuning, this time fixing the alpha=0.01 and epsilon=0.0001 and tune only the starting a, b and c values in order to lower the number of iterations. The top 5 lowest MSE and the parameters used are shown below. The top 5 models all converged in 4 iterations, thus I will be using the parameters of the model with the lowest MSE.

alpha	epsilon	iterations	starting a	starting b	starting c	E(a,b,c)
0.01	0.0001	4.0	4.5	1.4	17.0	0.156181
0.01	0.0001	4.0	4.5	1.3	17.0	0.156193
0.01	0.0001	4.0	4.5	1.5	17.0	0.156255
0.01	0.0001	4.0	4.5	1.2	17.0	0.156292
0.01	0.0001	4.0	4.5	1.6	17.0	0.156416

Hence, alpha=0.01 epsilon=0.0001, starting a-value=4.5, starting b-value=1.4 and starting c-value=17 is used for the final Model 3, which converged in 4 iterations to obtain a MSE of 0.156181.

(d) Describe your MODEL 3 by filling the information below.

Final MODEL 3 equation is: $\hat{y} = 4.3896 + 0.17842x + 16.983w$ (5s.f)

Minimum value of Error function is: 0.15618 (5s.f)

Number of iterations ran to reach convergence: 4

Conclusion (10 marks)

(a) David used gradient descent algorithm to find the 3 models. Next, he computed the predicted housing prices using the 3 models for all the data points in the dataset. He noticed that for one of the data points, the error of the predicted housing price in Model 1 from the actual housing price is the smallest, compared to the other 2 models. Is this possible, assuming he has done his gradient descent algorithm correctly? Explain.

If he has done gradient descent algorithm correctly and tuned his model, it would not be possible. In the Python code screenshot below, I calculated the Absolute Error (to make negative values positive) of each data point in the model and then compared the Absolute Error of model 1 with the Absolute Error of model 2 and 3. The results show that there is no data point where the error of predicted price from actual price is smallest in model 1 compared to the 2 other models.

```
correct = False
for rownum in np.arange(len(df)):
    y = df.loc[rownum, 'price per square metre ($ in thousands)']
    x = df.loc[rownum, 'inverse distance to the nearest MRT station (km-1)']
    w = df.loc[rownum, 'inverse age of flat (year-1)']
    error_model1 = abs(y - 0.81311*x)
    error_model2 = abs(y - (5.7456 + 0.29878*x))
    error_model3 = abs(y - (4.3896 + 0.17842*x + 16.983*w))

    if error_model1 < error_model2 and error_model1 < error_model3:
        print("David is correct. For 1 of the data points, Error is the smallest, compared to the other 2 m
        correct = True

if correct == False:
    print("David is incorrect")
```

✓ 0.1s

```
David is incorrect
```

(b) Compare the 3 models. Which model will you use to predict housing price in this context? Explain.

To find the best model to predict housing price, the Error function results can be compared as the Mean Squared Error (MSE) for each model tells us how well the linear line has fit to the dataset compared to the actual housing prices.
In this case, model 1 would be eliminated since our optimized model 1 has a MSE of 19.467, which is way higher than MSE of 0.46209 for model 2 and MSE of 0.15618 for model 3. The high MSE also tells us that Model 1 does not fit well to the dataset as average distance between the data points and the linear line is 19.467 which is quite high compared to the MSE of models 2 and 3.
To choose between model 2 and 3, I would pick model 3 for predicting over model 2. The MSE of model 3 is around 0.3 lower than model 2, which indicates that the distance between data points and the linear line is smaller for model 3 compared to model 2. This can result in model 3 giving better predictions of house prices compared to using model 2, which has a higher MSE. Thus, I would use model 3 to predict housing price in this context.