Lecture 9-1: Linear Regression

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

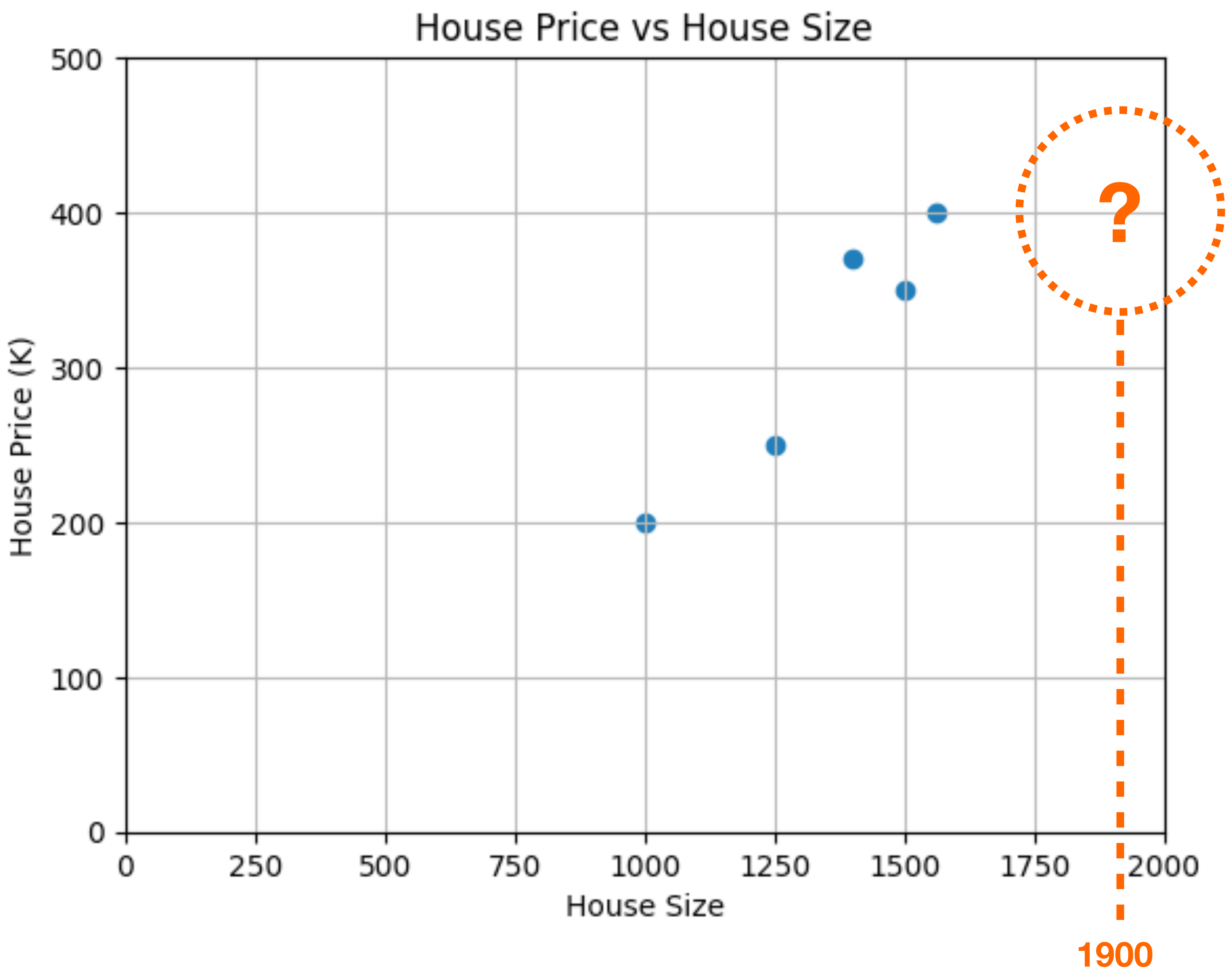
Let’s Start With an Example - Housing Prices

X’s: “input” variable / features / independent variables

Y’s: “output” variable / labels / response variables

	X	Y
	house_size	house_price_k
0	1000	200
1	1250	250
2	1400	370
3	1500	350
4	1560	400
5	1900	???

How do we predict the price?



Supervised Learning

	X house_size	Y house_price_k
0	1000	200
1	1250	250
2	1400	370
3	1500	350
4	1560	400

Training Set

↓

Learning Algorithm

↓

X (House size) → Hypothesis → Y (House price)

$$y = h(x) = \beta_0 + \beta_1 x$$

The hypothesis that maps x to y

e.g., **Linear regression** is a type of **supervised learning algorithm:**

(x, y) - one training example

feature **label**

Hypothesis

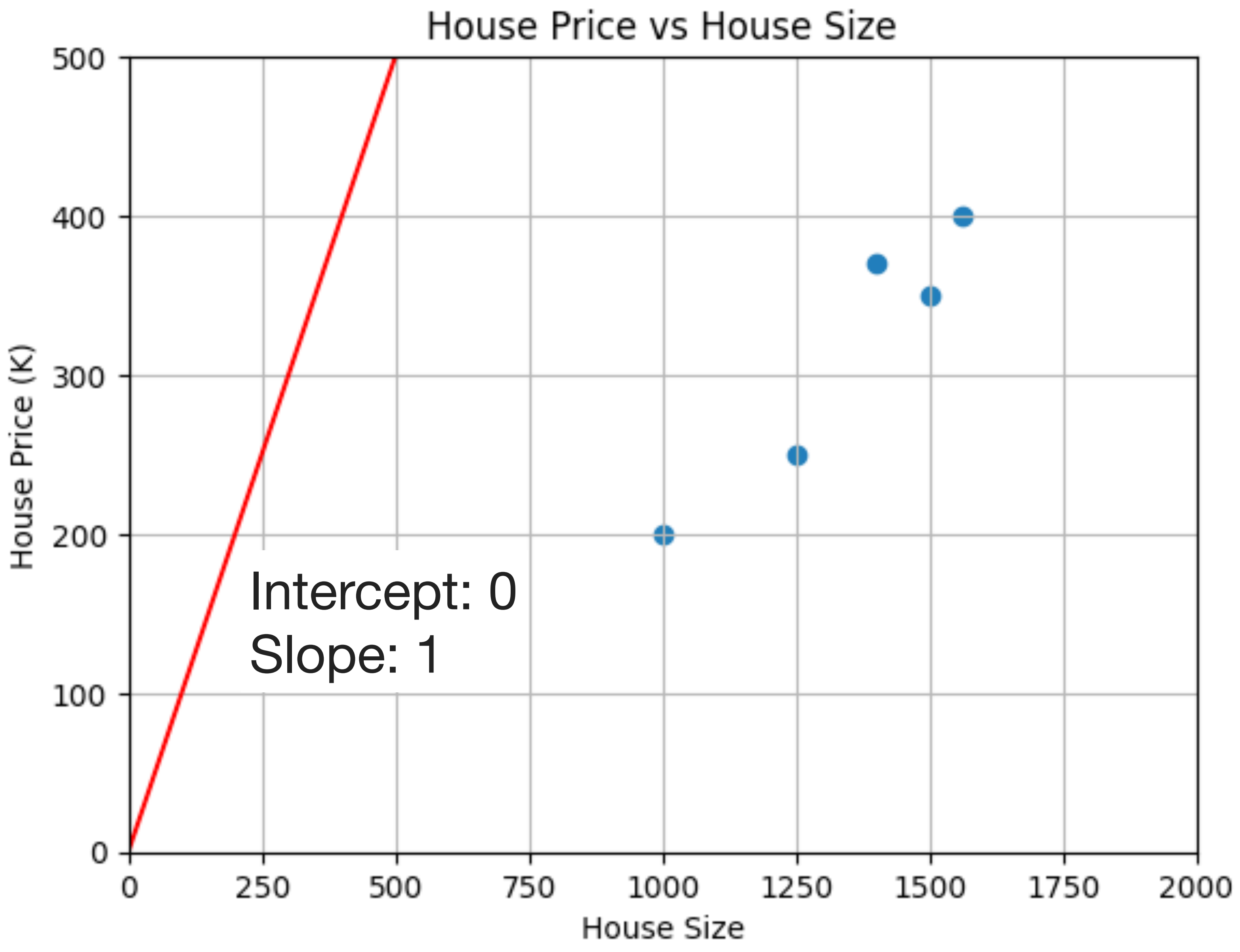
$$y = h(x) = \beta_0 + \beta_1 x$$

The hypothesis that maps x to y (predicted)

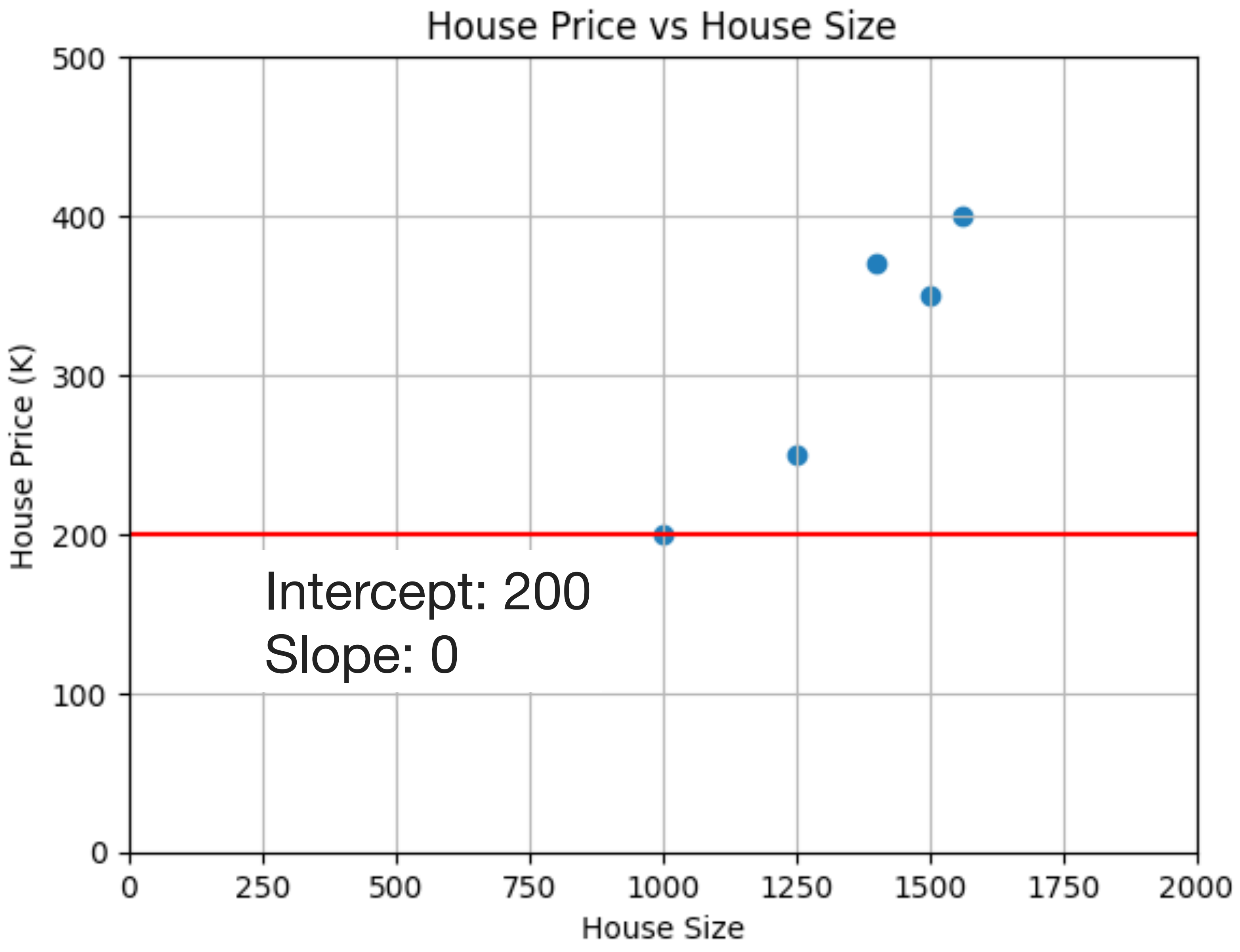
Parameter β_0: intercept
Parameter β_1: slope

Is it the best hypothesis?

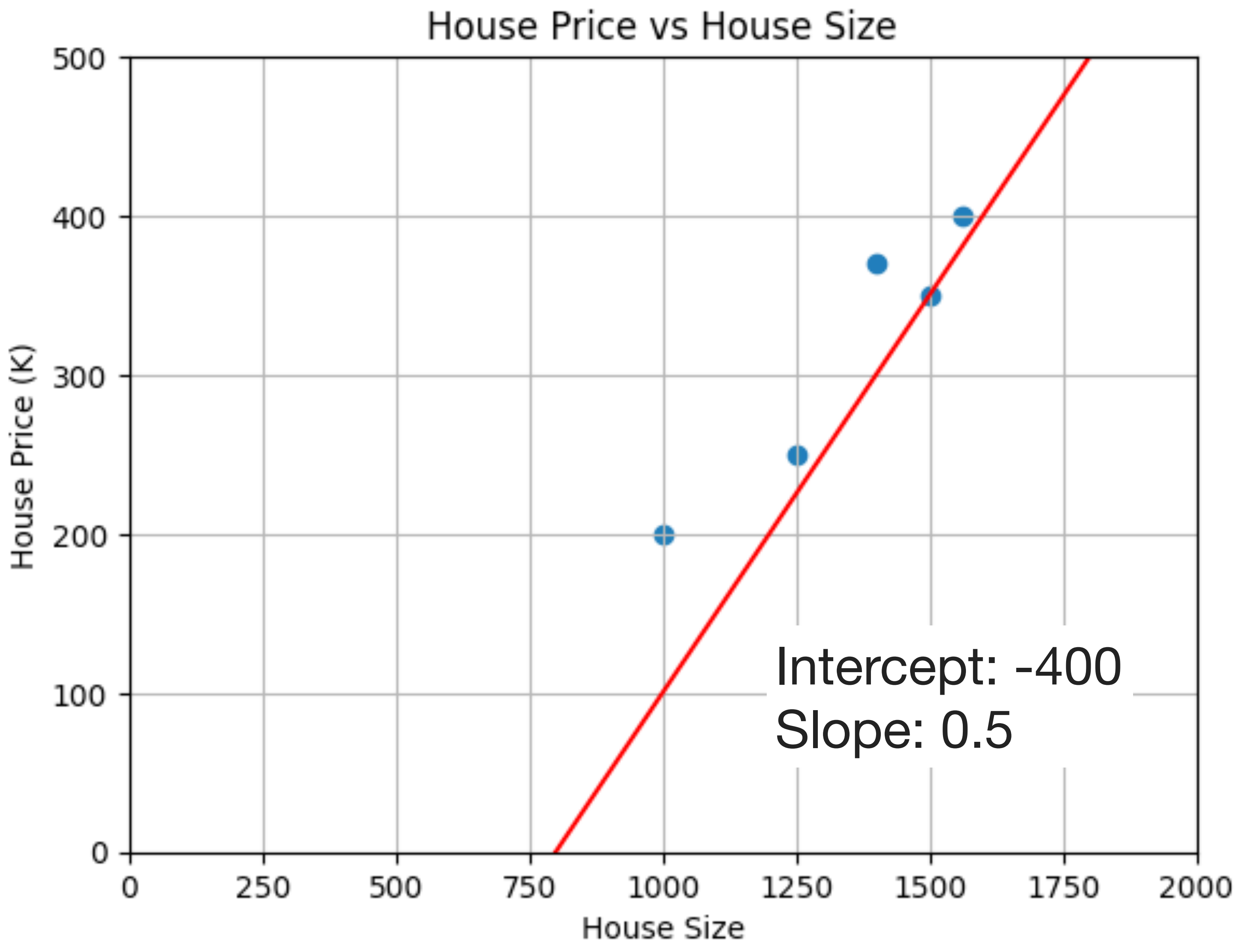
β: $(\beta_0, \beta_1) = (0, 1)$



β: $(\beta_0, \beta_1) = (200, 0)$



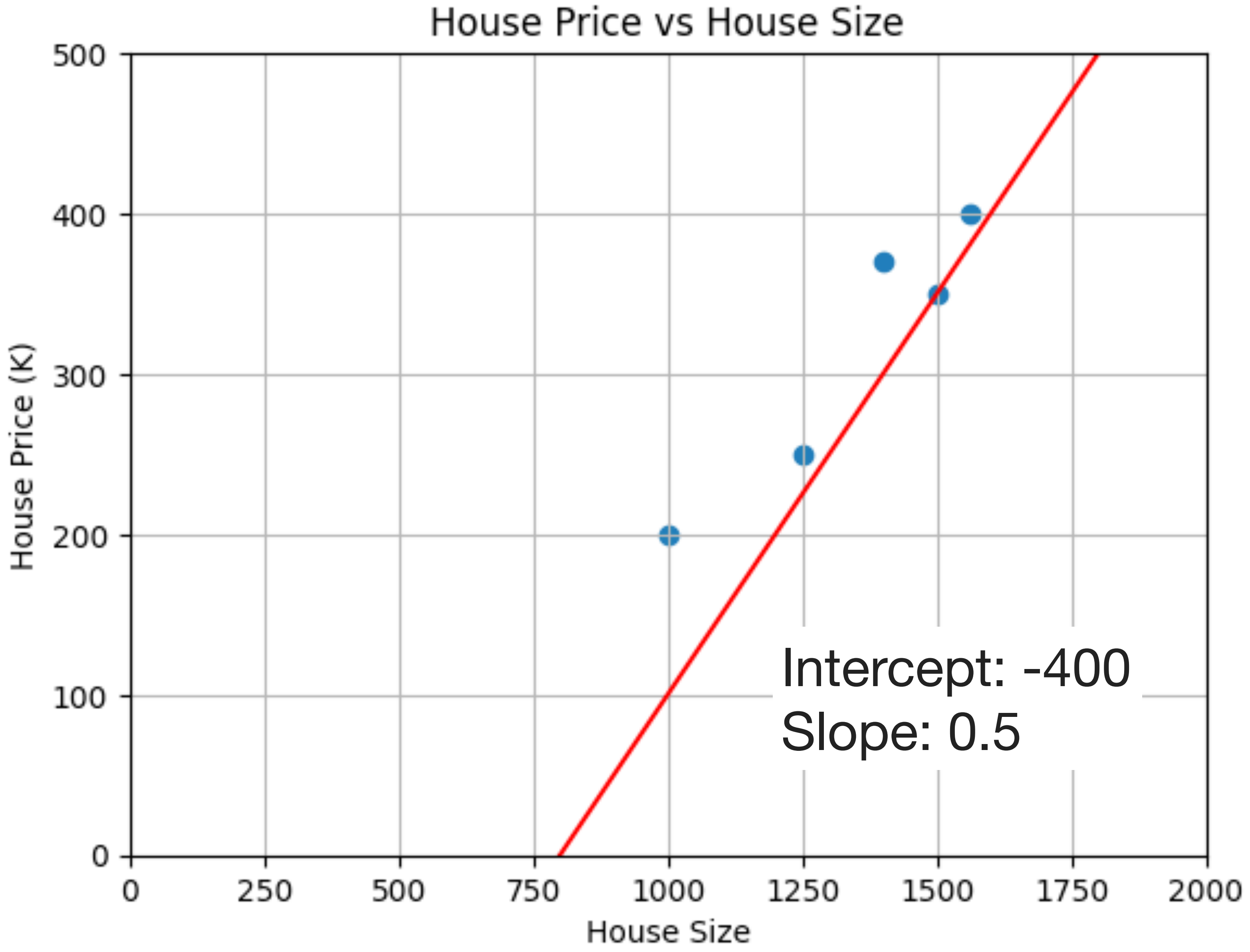
β: $(\beta_0, \beta_1) = (-400, 0.5)$



Loss Function

Is it the best hypothesis?

β: (β_0, β_1) = (-400, 0.5)



Let's define the loss function *f*: mean squared error (MSE)

MSE

$$f(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(\underbrace{h(x_i)}_{\text{Predicted y}} - \underbrace{y_i}_{\text{Observed y}})^2$$

Predicted y

$$h(x_i) = \underbrace{\beta_0}_{\text{Intercept}} + \underbrace{\beta_1}_{\text{Slope}} x_i$$

Loss Function

MSE

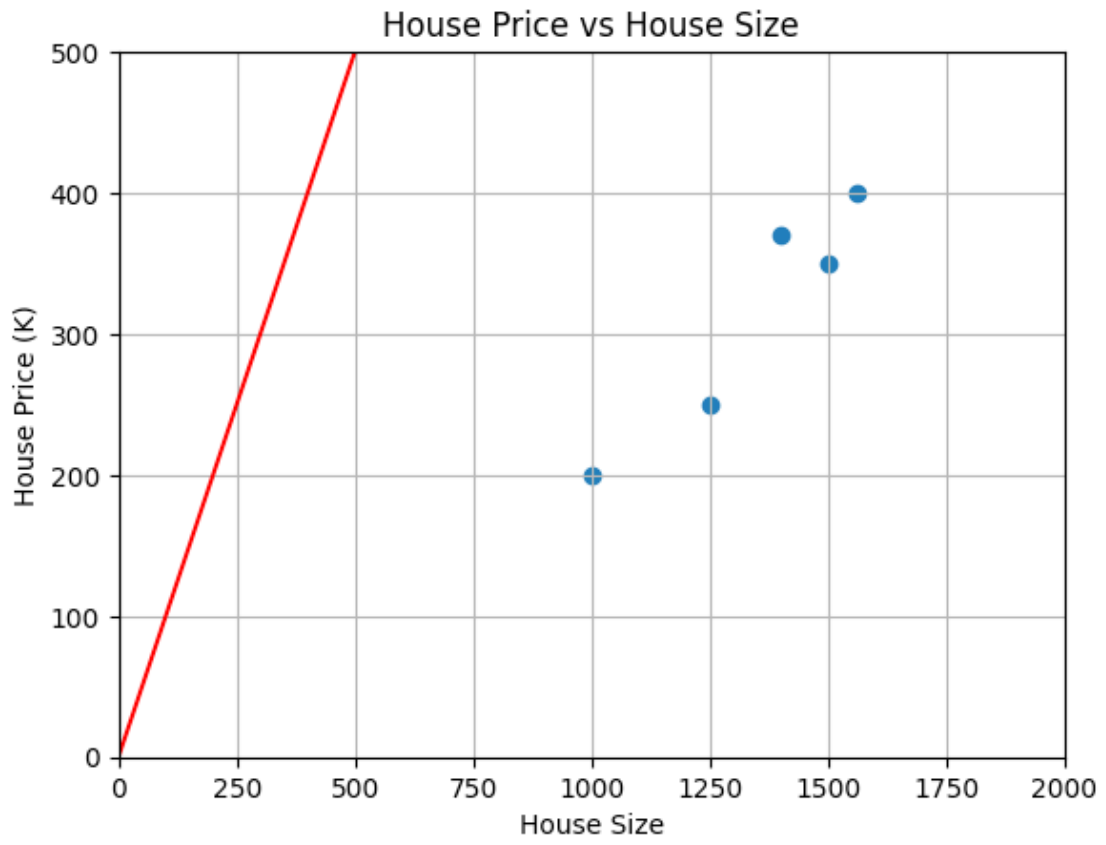
$$f(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(\underbrace{h(x_i)}_{\text{Predicted y}} - \underbrace{y_i}_{\text{Observed y}})^2$$

```python
def loss_func(data, b0, b1):
    # calculate loss
    loss = 0
    for i in range(len(data)):
        x = data.loc[i, 'house_size']
        y = data.loc[i, 'house_price_k']
        loss += (y - (b0 + b1 * x)) ** 2
    return loss / len(data)
```

```python
loss_func(data_house, b0=0, b1=1)
```

```
1073800.0
```

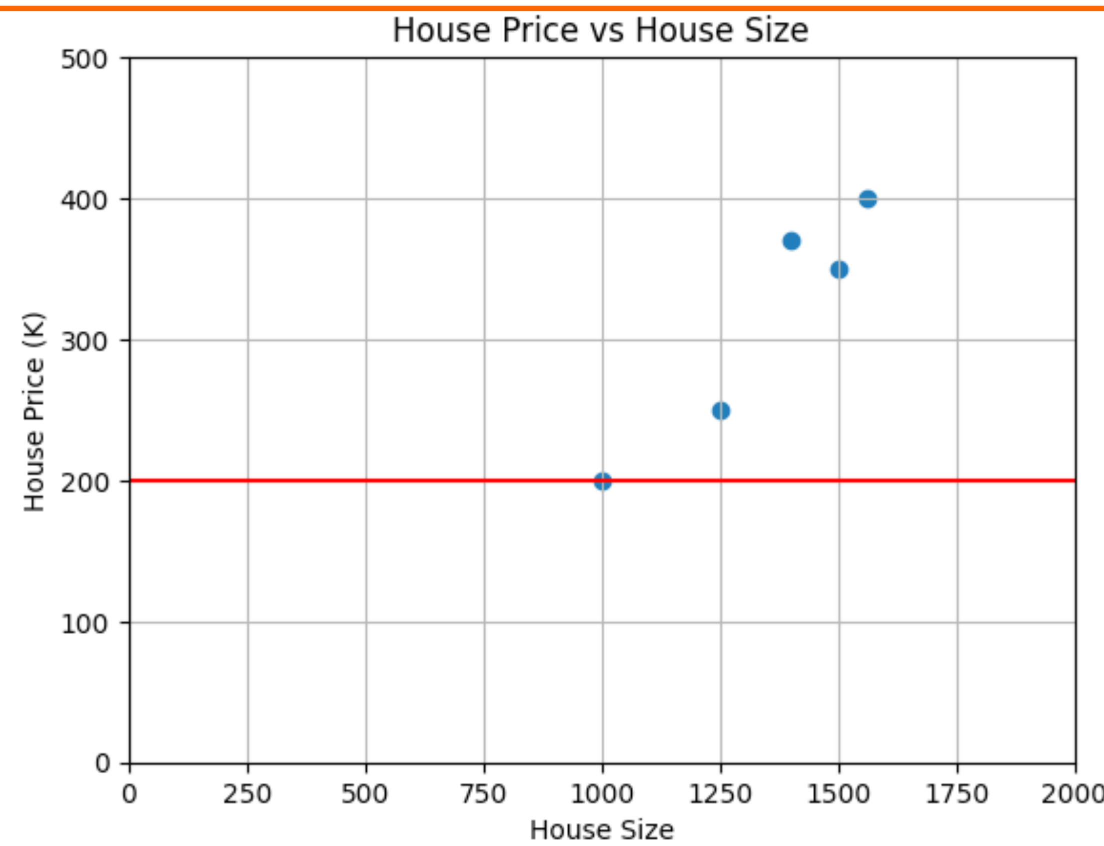
Intercept: 0
Slope: 1




```python
loss_func(data_house, b0=200, b1=0)
```

```
18780.0
```

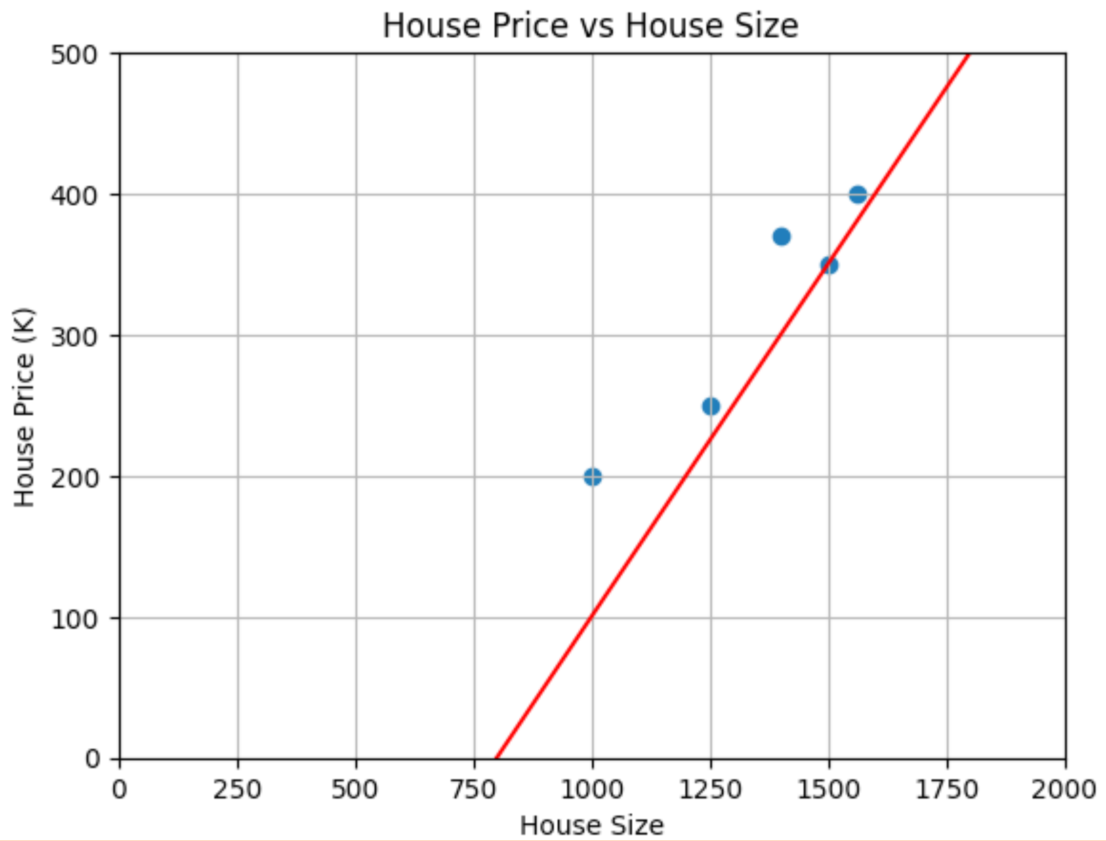
Intercept: 200
Slope: 0




```python
loss_func(data_house, b0=-400, b1=0.5)
```

```
3185.0
```

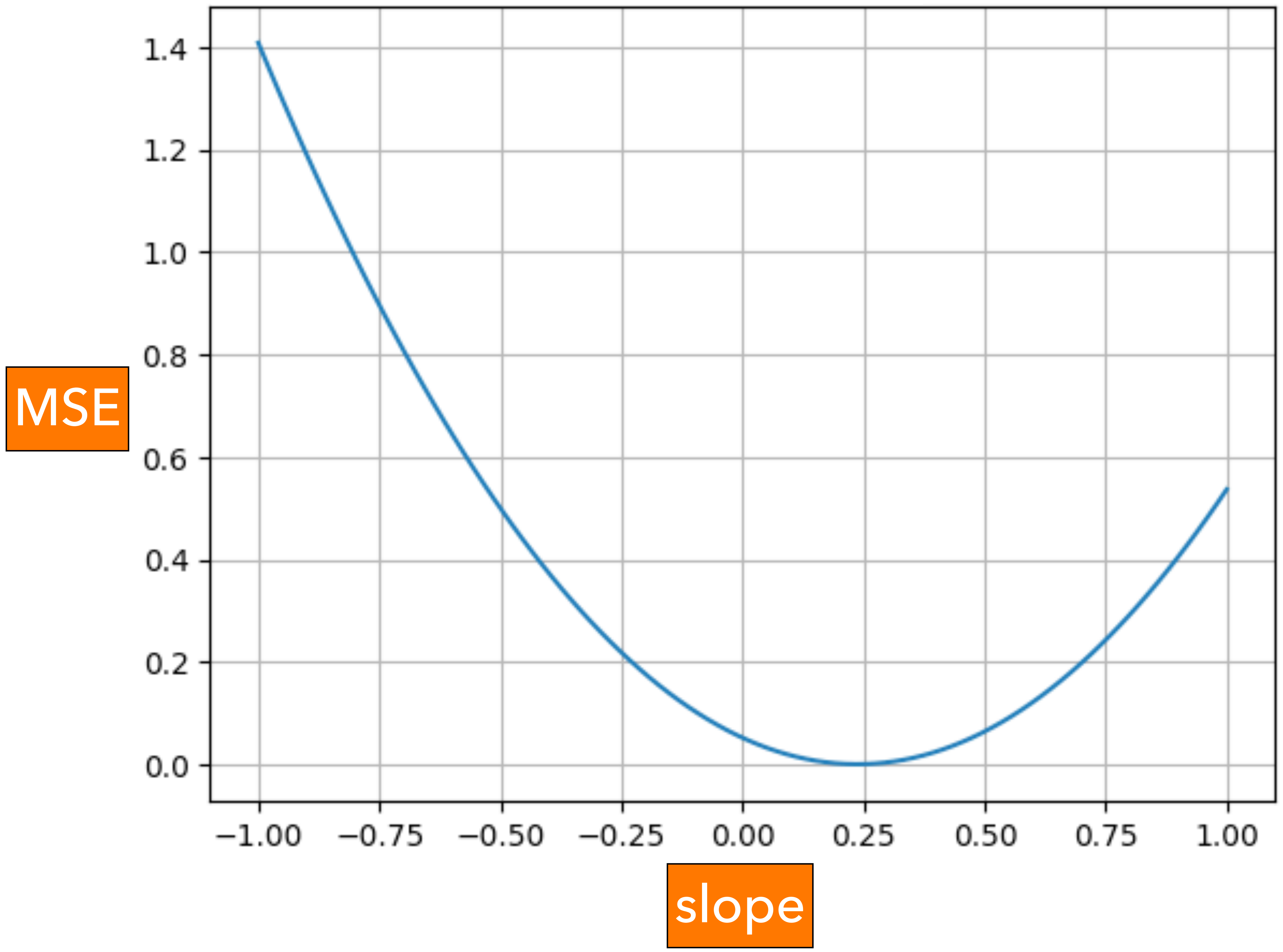
Intercept: -400
Slope: 0.5

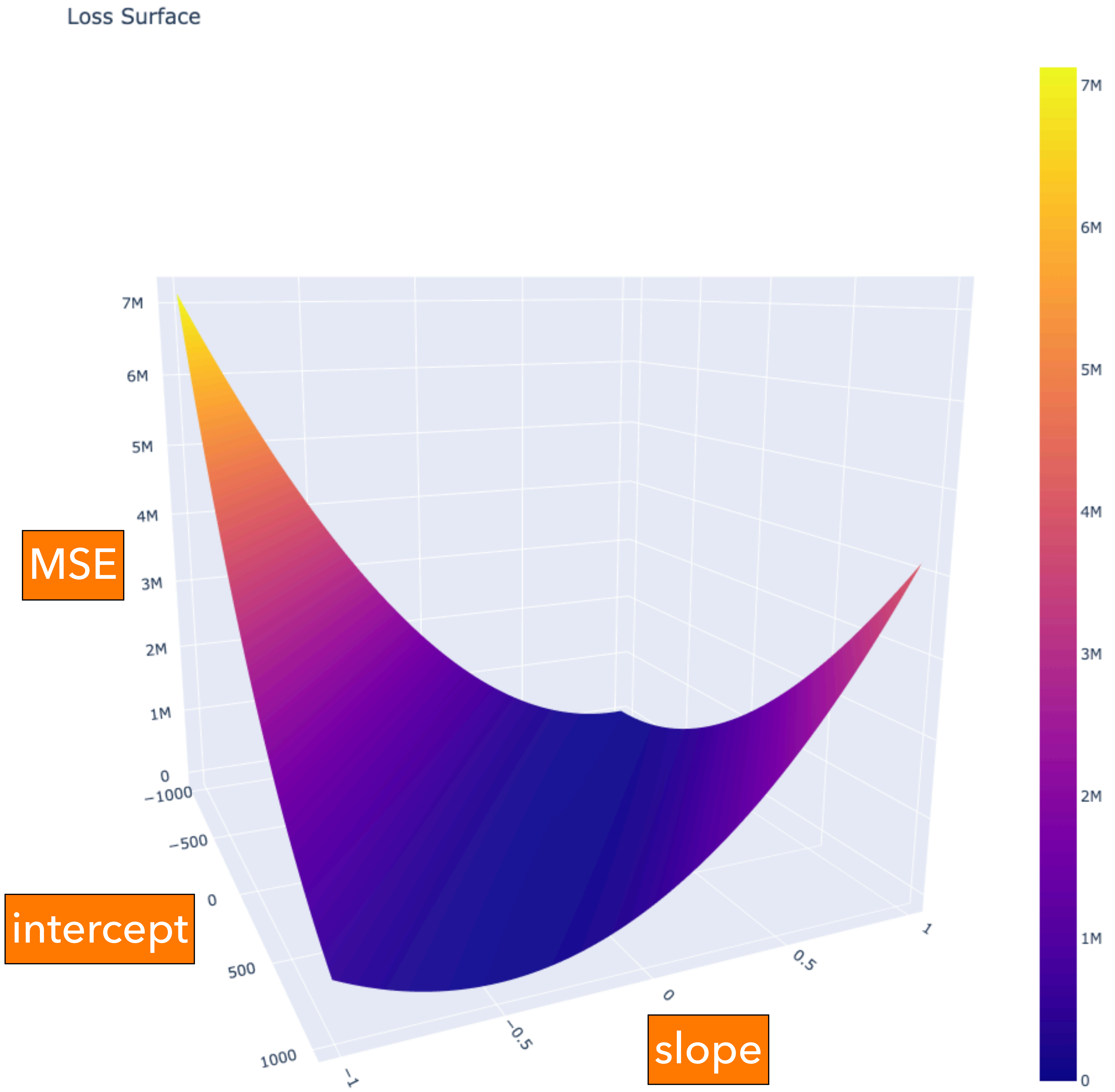


Loss Curve and Surface

How do we find a hypothesis that minimizes the loss? Differentiation!

We fixed the intercept to 0





Linear Algebra

Matrix multiplication

Rule 1

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \begin{bmatrix} g & h \\ i & j \end{bmatrix} = \begin{bmatrix} ag+bi & ah+bj \\ cg+di & ch+dj \\ eg+fi & eh+fj \end{bmatrix}$$

Rule 2

$$AB \neq BA$$

Matrix transpose

Rule 1

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

Rule 2

$$(A+B)^T = A^T + B^T$$

Rule 3

$$(AB)^T = B^T A^T$$

Rule 4

$$(ABC)^T = C^T B^T A^T$$

Matrix inverse

Rule 1

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$A^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Rule 2

$$A^{-1}A = I$$
$$AA^{-1} = I$$

Matrix derivative

$$\frac{\partial \beta^T A \beta}{\partial \beta} = 2\beta^T A$$
$$= 2A\beta$$

Matrix Representation

Data frame → Linear equations → Matrix representation

	X	Y
	house_size	house_price_k
0	1000	200
1	1250	250
2	1400	370
3	1500	350
4	1560	400

$$200 = \beta_0 + 1000\beta_1 + \epsilon_1$$
$$250 = \beta_0 + 1250\beta_1 + \epsilon_2$$
$$370 = \beta_0 + 1400\beta_1 + \epsilon_3$$
$$350 = \beta_0 + 1500\beta_1 + \epsilon_4$$
$$400 = \beta_0 + 1560\beta_1 + \epsilon_5$$

$$\begin{bmatrix} 200 \\ 250 \\ 370 \\ 350 \\ 400 \end{bmatrix} = \begin{bmatrix} 1 & 1000 \\ 1 & 1250 \\ 1 & 1400 \\ 1 & 1500 \\ 1 & 1560 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}$$

We want to minimize the error term

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

Ordinary Least Squares

The Ordinary Least Squares (OLS) is a method to estimate the parameters β_0 and β_1 in the hypothesis h by minimizing the sum of squared errors (SSE). The SSE is defined as follows:

$$\text{SSE} = \sum_{i=1}^{m} (y_i - h(x_i))^2$$

MSE (Recap)

$$f(\beta_0, \beta_1) = \frac{1}{n} \sum_{i=1}^{n} (h(x_i) - y_i)^2$$

Or in matrix form:

$$\text{SSE} = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$$

Ordinary Least Squares

The minimization problem can be extended as

$$
\begin{aligned}
arg\min \mathrm{SSE} = arg\min_{\beta} f(\beta) &= arg\min_{\beta} (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) \\
&= arg\min_{\beta} (\mathbf{y}^T - \beta^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\beta) \\
&= arg\min_{\beta} \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\beta - \beta^T\mathbf{X}^T\mathbf{y} + \beta^T\mathbf{X}^T\mathbf{X}\beta \\
&= arg\min_{\beta} \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X}\beta
\end{aligned}
$$

Find the partial derivatives of $f(\beta)$ with respect to β and set it to zero:

$$\frac{\partial f}{\partial \beta} = \beta^T\mathbf{X}^T\mathbf{X} - \mathbf{X}^T\mathbf{y} = 0$$

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

This is the learning algorithm!

Organize the Matrices in Python

Now, we can plug in our dataset to this formula to obtain the optmium β. First, we need to define X and y in Python.

```
X = data_house['house_size']
y = data_house['house_price_k']
X = np.array(X)
X = np.vstack([np.ones(len(X)), X]).T # add intercept
y = np.array(y)
# show
display("X:", X)
display("y:", y)
```

Python representation

```
'X:'

array([[1.00e+00, 1.00e+03],
       [1.00e+00, 1.25e+03],
       [1.00e+00, 1.40e+03],
       [1.00e+00, 1.50e+03],
       [1.00e+00, 1.56e+03]])

'y:'

array([200, 250, 370, 350, 400])
```

Math representation

$$\begin{bmatrix} 200 \\ 250 \\ 370 \\ 350 \\ 400 \end{bmatrix} = \begin{bmatrix} 1 & 1000 \\ 1 & 1250 \\ 1 & 1400 \\ 1 & 1500 \\ 1 & 1560 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}$$

Manually Solve the OLS Problem

Obtain the β based on the formula:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (1)$$

```python
XtX = np.dot(X.T, X)
XtXi = np.linalg.inv(XtX)
Xy = np.dot(X.T, y)
b = np.dot(XtXi, Xy)
print(b)
```

MagicPython

`XtX = np.dot(X.T, X)`	Matrix multiplication	$\mathbf{X}^T\mathbf{X}$
`XtXi = np.linalg.inv(XtX)`	Matrix inverse	$(\mathbf{X}^T\mathbf{X})^{-1}$
`Xy = np.dot(X.T, y)`	Matrix multiplication	$\mathbf{X}^T\mathbf{y}$
`b = np.dot(XtXi, Xy)`	Matrix multiplication	$(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

```
[-169.78139905    0.36049285]
```

Parameter β_0 intercept

Parameter β_1 slope

Wrap Up the Solver Into a Function

We can wrap the computation as a solver function, which takes X and yas input parameters.

```
def solve_OLS(X, y):
    XtX = np.dot(X.T, X)
    XtXi = np.linalg.inv(XtX)
    Xy = np.dot(X.T, y)
    b = np.dot(XtXi, Xy)
    return b
```

MagicPython

Should return the same β as the previous computation.

```
solve_OLS(X, y)
```

MagicPython

```
array([-169.78139905,    0.36049285])
```

Parameter β_0
intercept

Parameter β_1
slope

Validate the Solution

Great! Now we have the optimal β_0 = -169.78 and β_1 = 0.36. Let's validate this result.

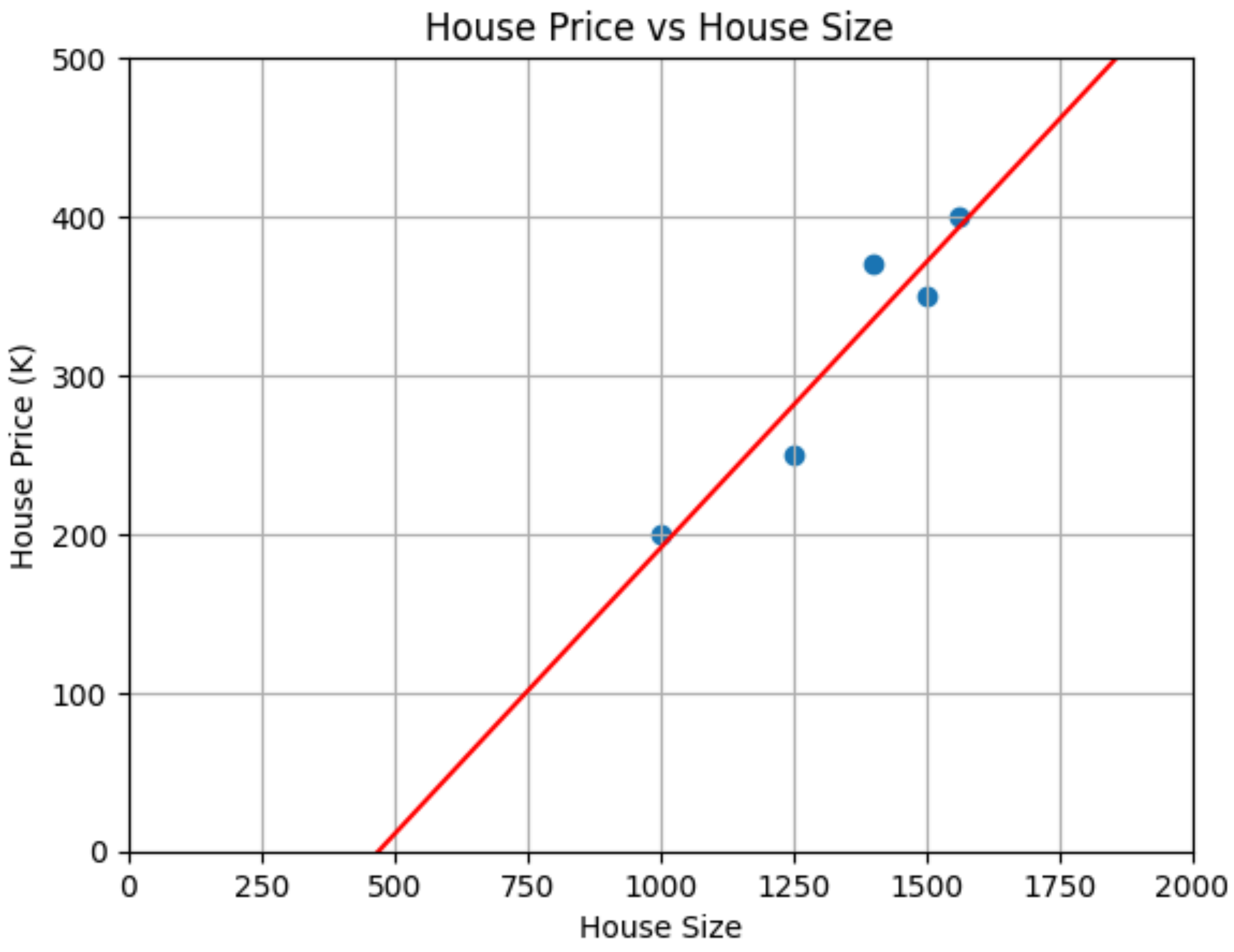
```
loss_func(data_house, b0=b[0], b1=b[1])
```

MagicPython

```
552.5278219395883
```

```
plot_house(data=data_house, b0=b[0], b1=b[1])
```

MagicPython



Sklearn

The Python package `sklearn` provides a function to solve the OLS problem. It's noted that we don't need to add intercepts to the X matrix.

```
from sklearn.linear_model import LinearRegression
X = np.array(data_house['house_size']).reshape((-1, 1))
y = np.array(data_house['house_price_k'])
model = LinearRegression()
model.fit(X, y)
```

MagicPython

```
▾ LinearRegression
LinearRegression()
```

```
print("coef.: ", model.coef_)
print("intercept: ", model.intercept_)
```

```
coef.:  [0.36049285]
intercept:  -169.78139904610492
```

We can also use this model to predict a new data point.

```
new_x = [[750]] # predict house price for 750 sqft house
model.predict(new_x)
```

```
array([100.58823529])
```