Linear Regression

This course materials are based on the online material by Andrew Ng

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

Linear Regression with one variable

Data overview - Housing price

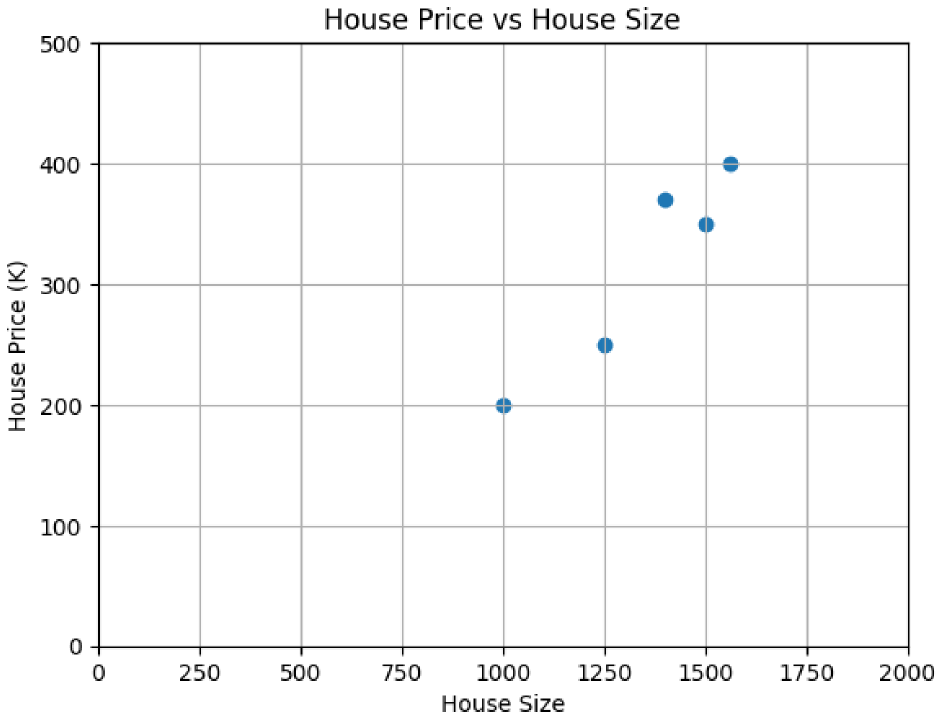
```
house_size = [1000, 1250, 1400, 1500, 1560]
house_price_k = [200, 250, 370, 350, 400]
data_house = pd.DataFrame({'house_size': house_size,
                           'house_price_k': house_price_k})
data_house
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

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

```
plt.scatter(data_house['house_size'], data_house['house_price_k'])
plt.xlabel('House Size')
plt.ylabel('House Price (K)')
plt.title('House Price vs House Size')
plt.grid(True)
# axis limit
plt.xlim(0, 2000)
plt.ylim(0, 500)
plt.show()
```



Hypothesis

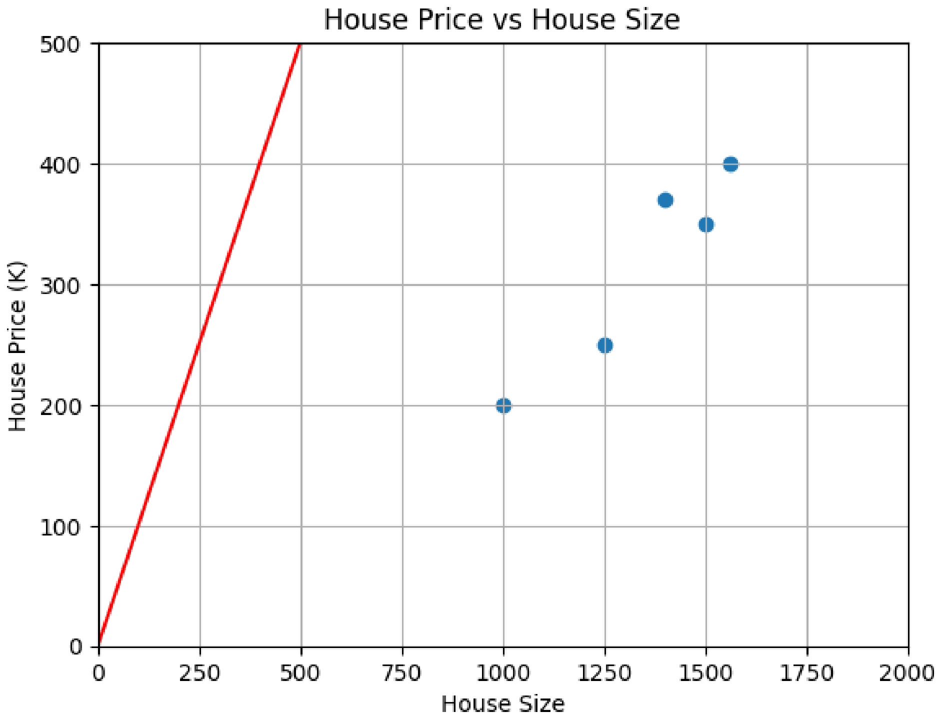
We will apply a supervised learning algorithm (i.e., linear regression in this case) to propose a hypothesis h with a given dataset. The hypothesis h determines an estimated "House Price (k)" (y) based on a given "House Size" (x). In short, The hypothesis h is a function of x that maps x to y.

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

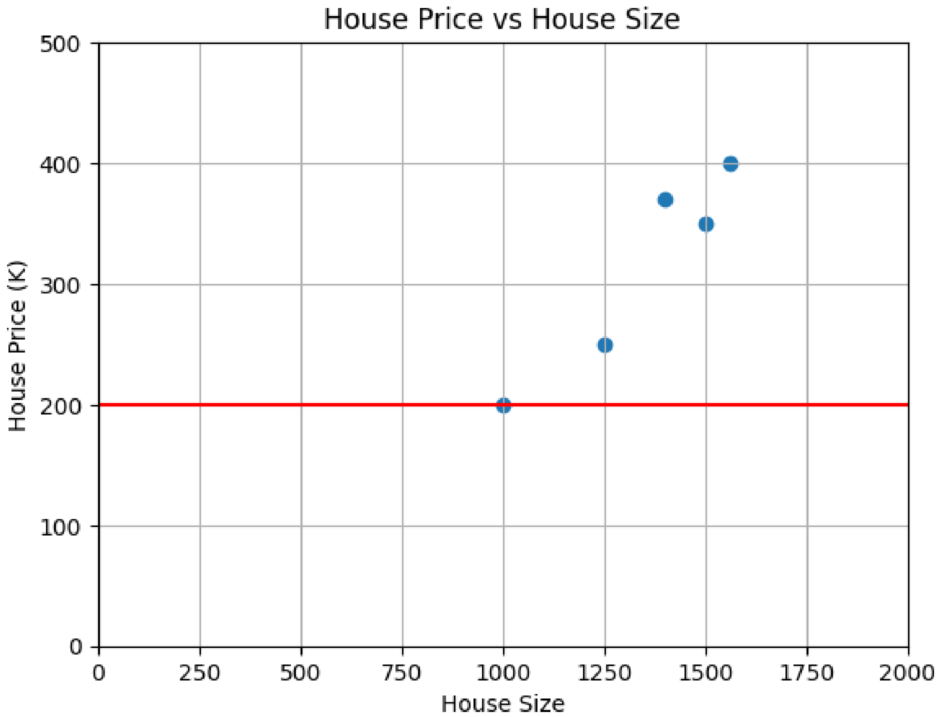
We can try to plug in different values of β_0 and β_1 to see how the hypothesis h line looks like. To simply the code, we need to define a plotting function that takes three parameters: dataset, β_0, and β_1

```
def plot_house(data, b0, b1):
    # scatter plot
    plt.scatter(data['house_size'], data['house_price_k'])
    # hypothesis line
    x_series = np.linspace(0, 2000, 100)
    y_series = b0 + b1 * x_series
    plt.plot(x_series, y_series, 'r')
    # plot labels
    plt.title('House Price vs House Size')
    plt.xlabel('House Size')
    plt.ylabel('House Price (K)')
    # axis limit
    plt.xlim(0, 2000)
    plt.ylim(0, 500)
    # show plot
    plt.grid(True)
    plt.show()
```

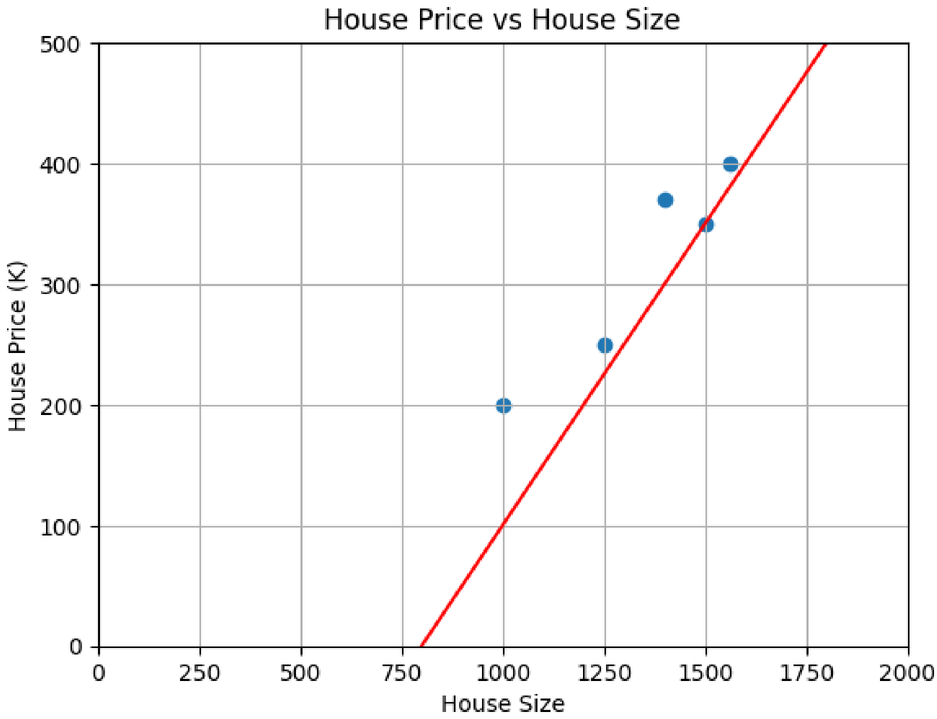
```
plot_house(data=data_house, b0=0, b1=1)
```




```
plot_house(data=data_house, b0=200, b1=0)
```




```
plot_house(data=data_house, b0=-400, b1=0.5)
```



Loss function

To evaluate the hypothesis h, we need to define a loss function. The loss function measures the difference between the estimated value ($h(x)$) and the actual value (y). The loss function is defined as

```
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)
```

And you can check the loss function by plugging in different values of β_0 and β_1.

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

```
1073800.0
```

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

```
18780.0
```

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

```
3185.0
```

We can generate a loss surface by plotting the loss function with respect to β_0 and β_1. The loss surface is a 3D plot that shows the loss function with respect to β_0 and β_1. The loss surface is shown below.

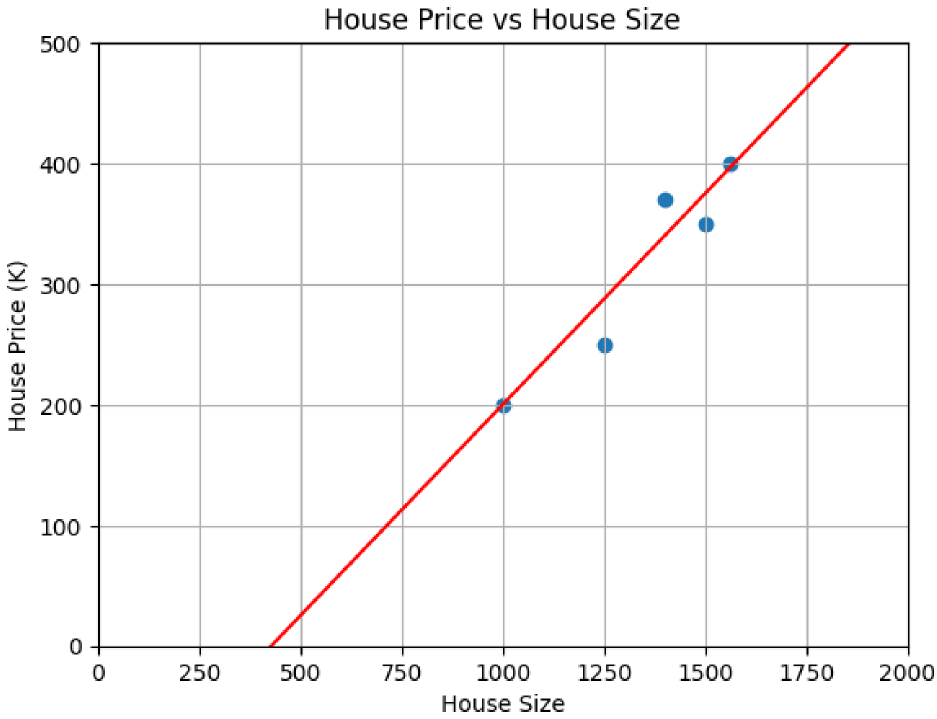
```
# generate loss surface
b0_series = np.linspace(-1000, 1000, 100)
b1_series = np.linspace(-1, 1, 100)
b0_grid, b1_grid = np.meshgrid(b0_series, b1_series)
loss_grid = np.zeros((len(b0_series), len(b1_series)))
for i in range(len(b0_series)):
    for j in range(len(b1_series)):
        loss_grid[i, j] = loss_func(data=data_house,
                                   b0=b0_series[i],
                                   b1=b1_series[j])
# plot loss surface
import plotly.graph_objs as go
trace = go.Surface(x=b0_series, y=b1_series, z=loss_grid.T)
layout = go.Layout(
                title='Loss Surface',
                scene=dict(
                        xaxis=dict(title='b0'),
                        yaxis=dict(title='b1'),
                        zaxis=dict(title='SSE')),
                width=900, height=900,
                margin=dict(r=20, l=10, b=10, t=50))
fig = go.Figure(data=[trace], layout=layout)
fig.show()
```

Let's try βs that seems to be the minimum of the loss surface.

```
loss_func(data_house, b0=-150, b1=0.35)
```

```
589.4500000000007
```

```
plot_house(data=data_house, b0=-150, b1=0.35)
```



Matrix representation

So how do we calculate the best hypothesis h that can minimize the loss? The hypothesis h can be written in a matrix form as follows:

$$ \begin{bmatrix} y_1 \ y_2 \ \vdots \ y_m \end{bmatrix}

\begin{bmatrix} 1 & x_1 \ 1 & x_2 \ \vdots & \vdots \ 1 & x_m \end{bmatrix} \begin{bmatrix} \beta_0 \ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \ \epsilon_2 \ \vdots \ \epsilon_m \end{bmatrix} $$

where ϵ is the error term. The matrix form of the hypothesis h is as follows:

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

$$

where $\mathbf{y}$ is a $m \times 1$ vector, $\mathbf{X}$ is a $m \times 2$ matrix, $\mathbf{\beta}$ is a 2×1 vector, and $\mathbf{\epsilon}$ is a $m \times 1$ vector.

In our dataset, the formula should look like this:

$$ \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} $$

This is equivalent to the following linear equations:

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

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:

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

Or in matrix form:

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

The minimization problem can be extended as

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

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

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

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

Solve the OLS problem on our dataset

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)
```

```
'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])
```

Obtain the β based on the formula:

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

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

```
[-169.78139905    0.36049285]
```

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
```

Should return the same β as the previous computation.

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

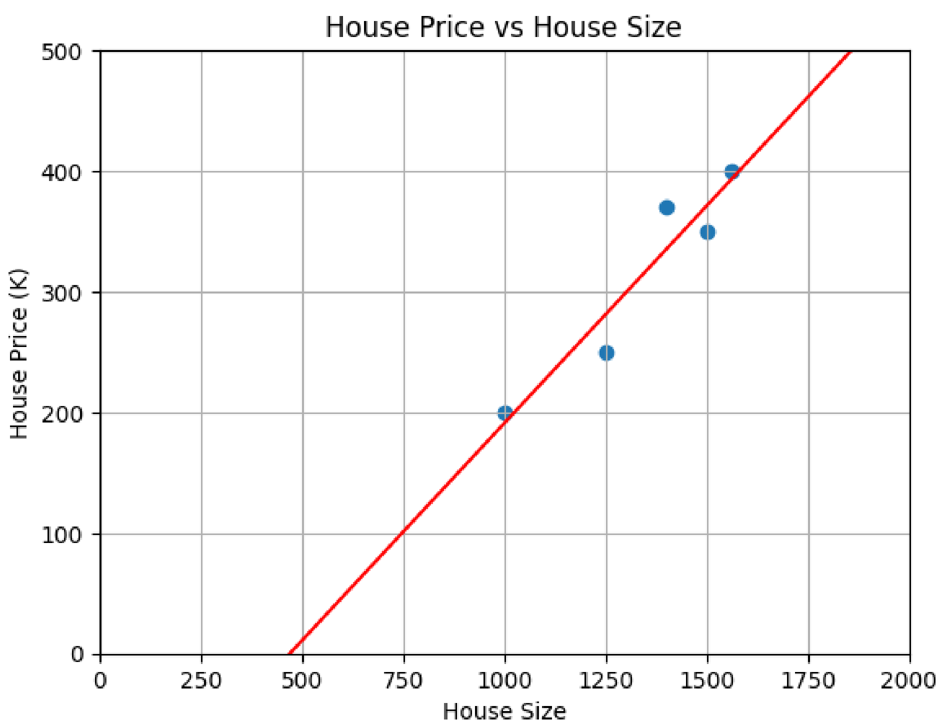
```
array([-169.78139905,    0.36049285])
```

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])
```

```
552.5278219395883
```

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



Sklearn - Linear Regression

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)
```

```
▾ 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])
```

Multi-variable Linear Regression

Medical insurance dataset

In this section, we will use a real-world dataset to demonstrate the multi-variable linear regression. The dataset contains information about medical insurance costs for 1338 people. The dataset is available on Kaggle.

```
data = pd.read_csv("insurance.csv")
data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400

	age	sex	bmi	children	smoker	region	charges
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

The regression problem

Say, we want to know how the listed factors, which include `age`, `sex`, `bmi`, `children`, `smoker`, `region`, affect the `charges` of medical insurance. We can define the hypothesis h as follows:

$$ h(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7 + \beta_8 x_8 $$

where

- x_1 = `age`
- x_2 = `sex`; 0 for male, 1 for female
- x_3 = `bmi`
- x_4 = `children`
- x_5 = `smoker`; 0 for non-smoker (no), 1 for smoker (yes)
- x_6 = `region` is `southwest` (1) or not (0)
- x_7 = `region` is `southeast` (1) or not (0)
- x_8 = `region` is `northwest` (1) or not (0)

Note that we turned a categorical variable `region` into three binary variables. This is called one-hot encoding.

Solve the OLS problem

Now, let's try to solve this problem using the formula (1). We need to define the matrix X and y before we can compute the β.

```
X = data.iloc[:, :-1]
y = data["charges"]
display(X)
display(y)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	sex	bmi	children	smoker	region
0	19	female	27.900	0	yes	southwest
1	18	male	33.770	1	no	southeast
2	28	male	33.000	3	no	southeast
3	33	male	22.705	0	no	northwest
4	32	male	28.880	0	no	northwest
...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest
1334	18	female	31.920	0	no	northeast
1335	18	female	36.850	0	no	southeast
1336	21	female	25.800	0	no	southwest
1337	61	female	29.070	0	yes	northwest

1338 rows × 6 columns

```
0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
           ...
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030
Name: charges, Length: 1338, dtype: float64
```

```
# turn categorical variable into dummy variables
X = pd.get_dummies(X, drop_first=True) # very handy function!
display(X)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
0	19	27.900	0	0	1	0	0	1
1	18	33.770	1	1	0	0	1	0
2	28	33.000	3	1	0	0	1	0
3	33	22.705	0	1	0	1	0	0

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
4	32	28.880	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...
1333	50	30.970	3	1	0	1	0	0
1334	18	31.920	0	0	0	0	0	0
1335	18	36.850	0	0	0	0	1	0
1336	21	25.800	0	0	0	0	0	1
1337	61	29.070	0	0	1	1	0	0

1338 rows × 8 columns

Add intercept to the X matrix and turn both X and y into numpy arraies.

```
# add intercept
X = np.array(X)
X = np.hstack([np.ones((len(X), 1)), X])
y = np.array(y)
```

Check the dimensions of X and y.

```
print("X shape: ", X.shape)
display(X)
print("y shape: ", y.shape)
display(y)
```

```
X shape:  (1338, 9)



array([[ 1.   , 19.   , 27.9 , ...,  0.   ,  0.   ,  1.   ],
       [ 1.   , 18.   , 33.77, ...,  0.   ,  1.   ,  0.   ],
       [ 1.   , 28.   , 33.   , ...,  0.   ,  1.   ,  0.   ],
       ...,
       [ 1.   , 18.   , 36.85, ...,  0.   ,  1.   ,  0.   ],
       [ 1.   , 21.   , 25.8 , ...,  0.   ,  0.   ,  1.   ],
       [ 1.   , 61.   , 29.07, ...,  1.   ,  0.   ,  0.   ]])


y shape:  (1338,)



array([16884.924 ,  1725.5523,  4449.462 , ...,  1629.8335,  2007.945 ,
       29141.3603])
```

Plug in the X and y into the formula (1) to obtain the β.

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

```
array([-11938.53857617,    256.85635254,    339.19345361,    475.50054515,
         -131.3143594 ,  23848.53454191,   -352.96389942,  -1035.02204939,
         -960.0509913 ])
```

We can solve the same problem using the `sklearn` library

```
X = data.iloc[:, :-1]
X = pd.get_dummies(X, drop_first=True)
y = data["charges"]
model = LinearRegression()
model.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

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

```
coef.:  [   256.85635254    339.19345361    475.50054515   -131.3143594
 23848.53454191   -352.96389942 -1035.02204939   -960.0509913 ]
intercept:  -11938.53857616715
```