Implementing Linear Regression in Python

1. Importing Required Packages

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Color for matplotlib
from matplotlib import cm
import matplotlib as mpl
```

In [2]:
```
%matplotlib inline
#Using tex for matplotlib axis
mpl.rcParams['text.usetex'] = True
```

2. Import Data

Housing price in Portland, Oregon. The data has been taken from the Coursera course "Machine Learning"

In [3]:
```
df_2 = pd.read_csv("./ex1data2.txt", names = ["Size of the house (in sq.
ft.)", "\# of bedrooms", "Price of the house (in \$1000s)"])
```

In [4]:
```
df_2.iloc[:, 2] = df_2.loc[:, "Price of the house (in \$1000s)"].div(1000)
```

In [5]:
```
df_2.head()
```

Out[5]:

	Size of the house (in sq. ft.)	\# of bedrooms	Price of the house (in $1000s)
0	2104	3	399.9
1	1600	3	329.9
2	2400	3	369.0
3	1416	2	232.0
4	3000	4	539.9

In [6]:
```
df_2.tail()
```

Out[6]:

	Size of the house (in sq. ft.)	\# of bedrooms	Price of the house (in $1000s)
42	2567	4	314.0
43	1200	3	299.0
44	852	2	179.9
45	1852	4	299.9

	Size of the house (in sq. ft.)	\# of bedrooms	Price of the house (in $1000s)
46	1203	3	239.5

In [7]:

```
df_2.to_csv('exdata.csv', index = False)
```

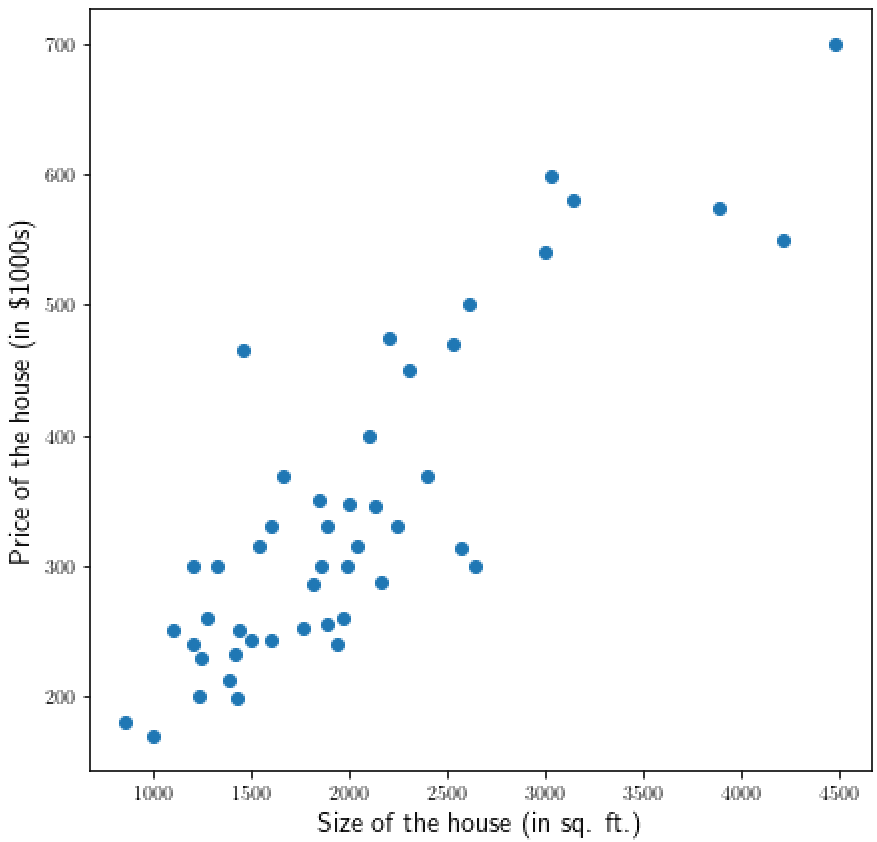
3. Plotting to show the Linear Relationship

In [8]:

```
axis_font_size = 14
```

In [9]:

```
width, height = [7,7]
fig = plt.figure(figsize=(width,height))
plt.scatter(df_2.iloc[:, 0], df_2.iloc[:, 2])
plt.xlabel(df_2.columns[0], fontsize = axis_font_size )
plt.ylabel(df_2.columns[2], fontsize= axis_font_size )
plt.savefig("./graph/uni_var_relationship", dpi = 300)
plt.show()
```

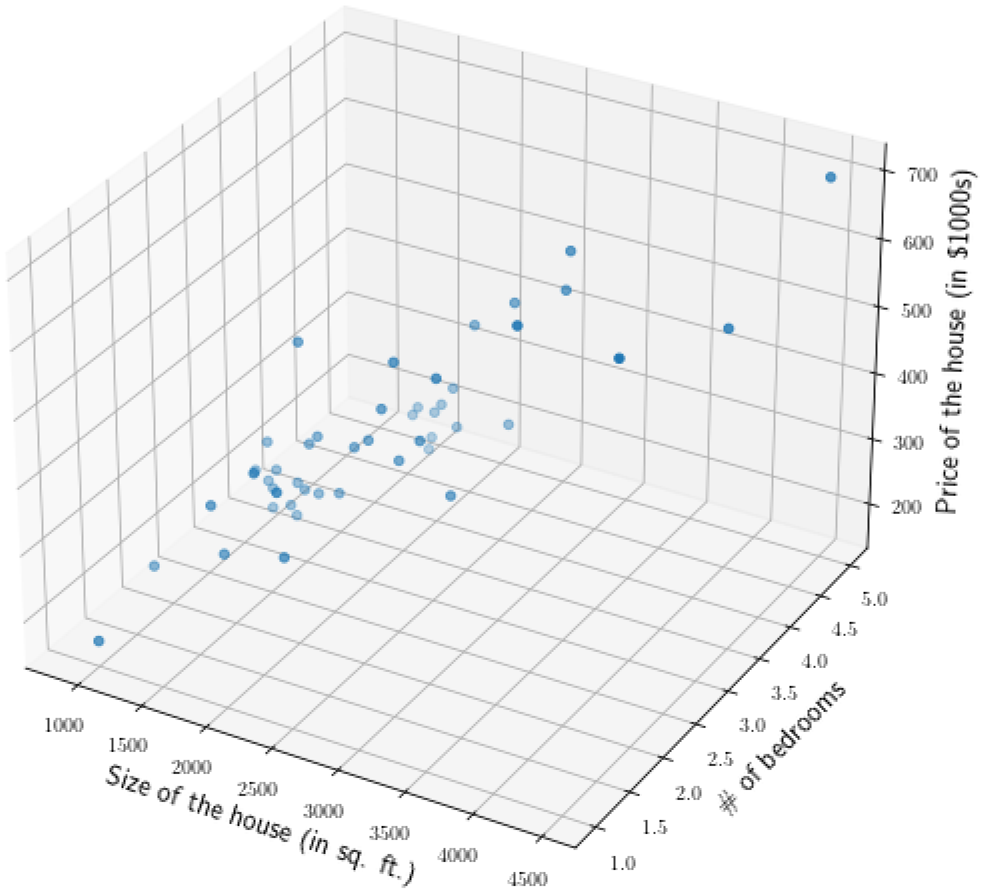


Adding additional feature "# of bathrooms" can be shown as below

In [10]:

```
width, height = [9,9]
fig = plt.figure(figsize=(width,height))
ax = fig.add_subplot(projection='3d')
ax.scatter(df_2.iloc[:, 0], df_2.iloc[:, 1], df_2.iloc[:, 2])


# Labels of axis
```

```python
ax.set_xlabel(df_2.columns[0], fontsize= axis_font_size )
ax.set_ylabel(df_2.columns[1], fontsize = axis_font_size )
ax.set_zlabel(df_2.columns[2], fontsize = axis_font_size )
plt.savefig("./graph/multi_var_relationship", dpi = 300)
plt.show()
```



Simple Linear Model

This will use Size of the house in (sq. ft) as a input variable and price of the house as target variable

3. Define x, θ, y

In [11]:
```python
x = df_2.iloc[:, 0]
theta = [0, 0]
y = df_2.iloc[:, 2]
```

4. Define Hypothesis

$h(x) = \theta_0 + \theta_1 * x$

In [12]:
```python
def hypothesis(x, theta):
    return (theta[0] + theta[1] * x)
```

5. Calculate Cost

Define function J

$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$

In [13]:

```
def uni_mse(x, y, theta):
    squared_sum = 0
    m = y.size
    for i_th in range(x.size):
        squared_sum += np.square(hypothesis(x[i_th], theta) - y[i_th])
    return (1/ (2*m) * squared_sum)
```

6. Visualize the cost function

This gives the appropriate indication and example, that using MSE error gives the global minima, meaning that there is only one minimum point possible.

Inspired by the examplary code from Coursera

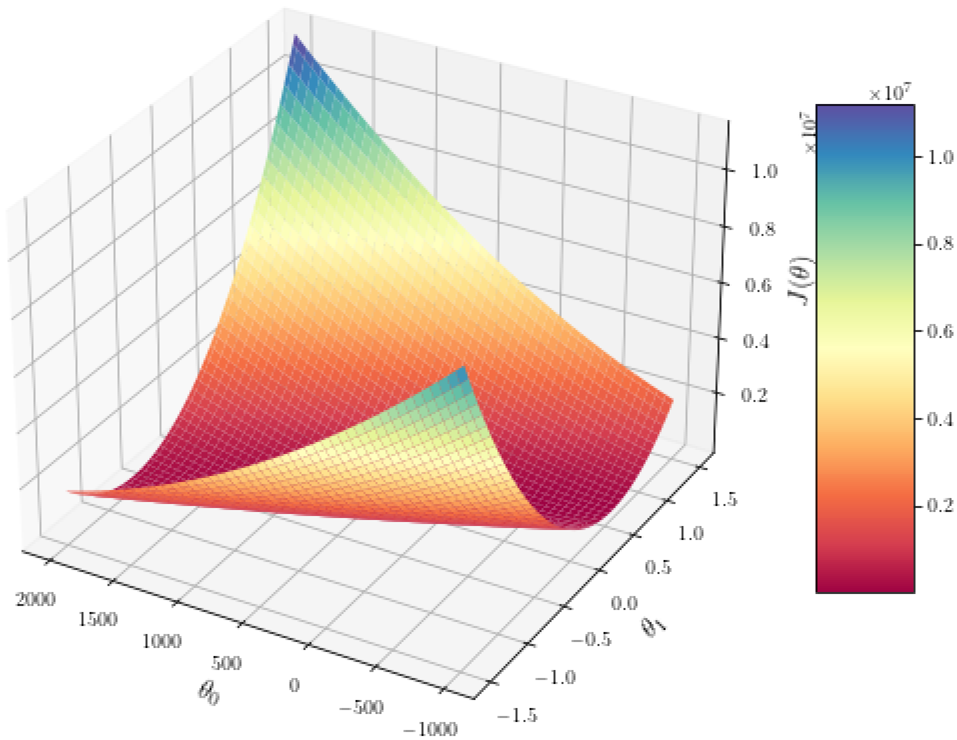
In [14]:

```
# Define the range of theta 0 and theta 1, where the cost will be visualized
theta0_val = np.linspace(-1000, 2000, 100)
theta1_val = np.linspace(-1.5, 1.5, 100)

# Set up zero matrix, so that the combination of each theta 0 and theta 1 can
generate the error
J_vals = np.zeros((theta0_val.size, theta1_val.size))

for i in range(theta0_val.size):
    for j in range(theta1_val.size):
        theta = [theta0_val[i], theta1_val[j]]
        J_vals[i, j] = uni_mse(x, y, theta)
J_vals = np.matrix.transpose(J_vals)
```

In [15]:

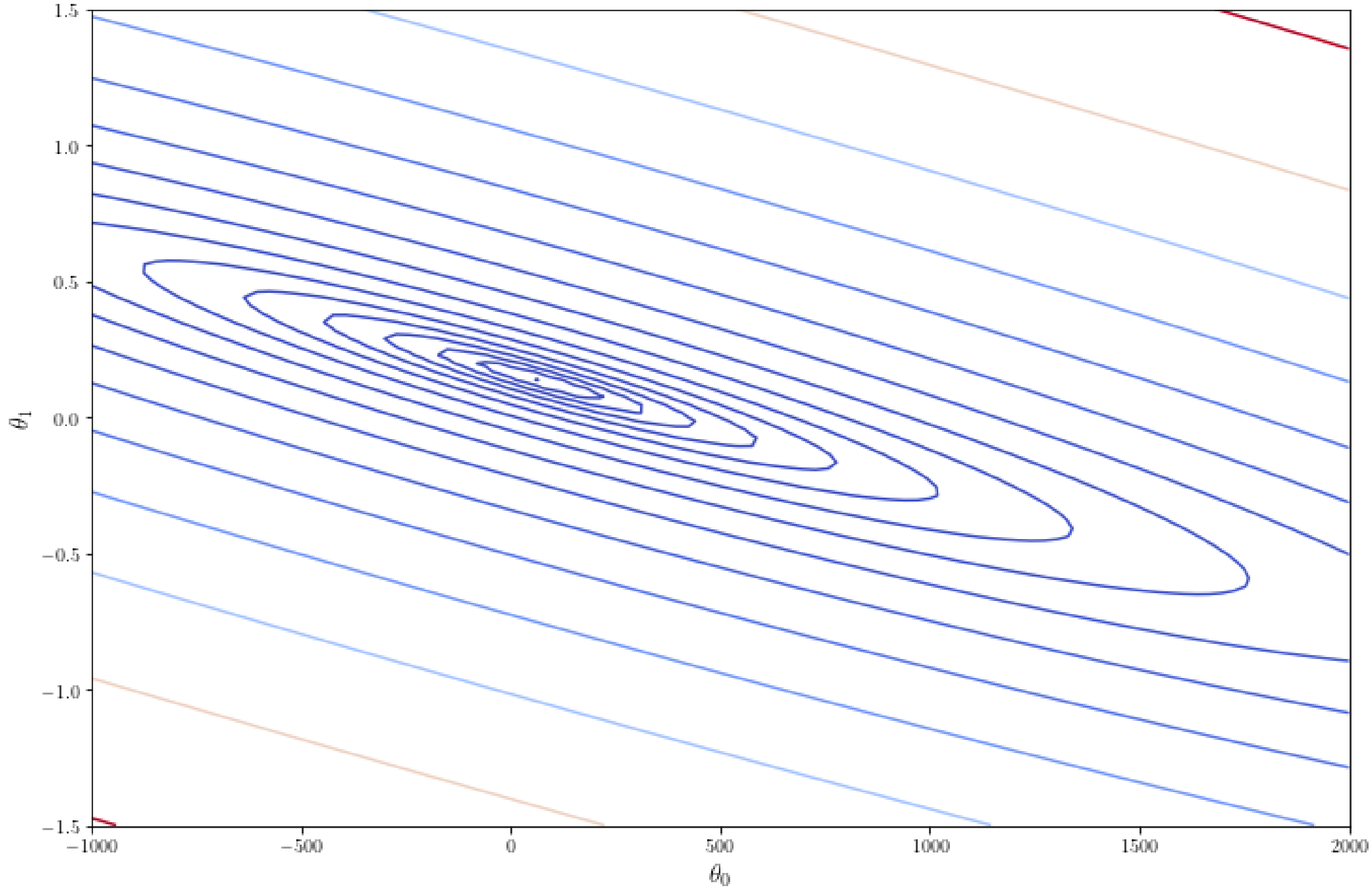
```
fig = plt.figure(figsize = (9,9))
ax = plt.axes(projection='3d')
X, Y = np.meshgrid(theta0_val, theta1_val)

surface = ax.plot_surface(X, Y, J_vals, cmap = cm.Spectral)
ax.set_xlabel(r'$\theta_0$', fontsize=axis_font_size)
ax.set_ylabel(r'$\theta_1$', fontsize=axis_font_size)
ax.set_zlabel(r'$J(\theta)$', fontsize=axis_font_size)
ax.invert_xaxis()

fig.colorbar(surface, shrink=0.5, aspect=5)
```

```
plt.savefig("./graph/surface_cost_actual", dpi = 300)
plt.show()
```



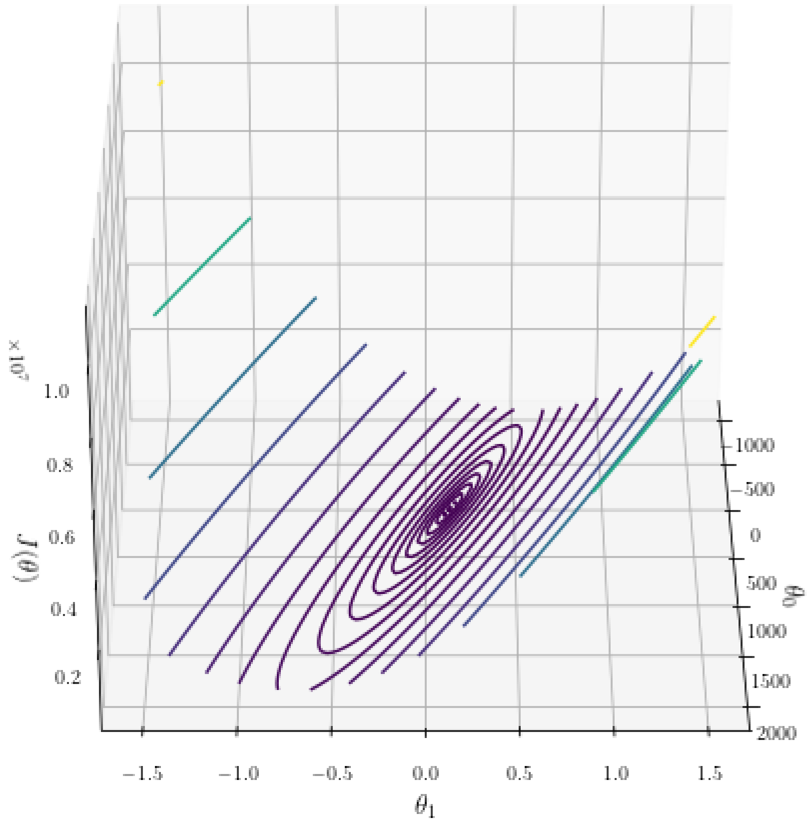
In [16]:

```
X, Y = np.meshgrid(theta0_val, theta1_val)
fig,ax=plt.subplots(1,1, figsize=(12,8))
contour = ax.contour(X, Y, J_vals, np.logspace(-5, 7),  cmap = cm.coolwarm)
ax.set_xlabel(r'$\theta_0$', fontsize=axis_font_size)
ax.set_ylabel(r'$\theta_1$', fontsize=axis_font_size)
plt.savefig("./graph/contour_cost_actual", dpi = 300)
plt.show()
```



In [17]:

```
X, Y = np.meshgrid(theta0_val, theta1_val)


fig = plt.figure(figsize = (9,9))
ax = plt.axes(projection='3d')

# ax.contour3D is used plot a contour graph
ax.contour3D(X, Y, J_vals, np.logspace(-4,7))



ax.set_xlabel(r'$\theta_0$', fontsize=axis_font_size)
ax.set_ylabel(r'$\theta_1$', fontsize=axis_font_size)
ax.set_zlabel(r'$J(\theta)$', fontsize=axis_font_size)
ax.view_init(azim=0)#
plt.savefig("./graph/contour_cost_actual_3d", dpi = 300)
plt.show()
```



Multiple Linear regression

3. Redefine X, θ, and y

In [18]:

```
X = df_2.iloc[:, :2]

#normalization
for column in X.columns:
    data = X[column]
    mean = data.mean()
    std = data.std()
    X[column] = (data - mean)/std

X = np.column_stack((np.ones(X.shape[0]), X))

theta = np.zeros(X.shape[1])
y = df_2.iloc[:, 2]
```

4. Define Hypothesis

$h(x) = \theta_0 x_0 + \theta_1 * x_1 + \ldots + \theta_n x_n = X \cdot \theta$

In [19]:

```
def multi_hyp(X, theta):
    return X @ theta
```

5. Calculate Cost

Define function J

$$J(\theta) = \frac{1}{2m}(\theta^T X^T X\theta - 2(X\theta)^T y + y^T y)$$

In [20]:
```
def multi_cost(X, y, theta):
    m = y.size
    return(1/(2*m) * (np.transpose(theta)@np.transpose(X)@X@theta - 2 *
np.transpose(X@theta)@y + np.transpose(y)@y))
```

6. Gradient Descent

1. Take gradient of the cost function
2. Repeat for specified amount of times (epoch). The size of each step is controlled by the value of alpha.

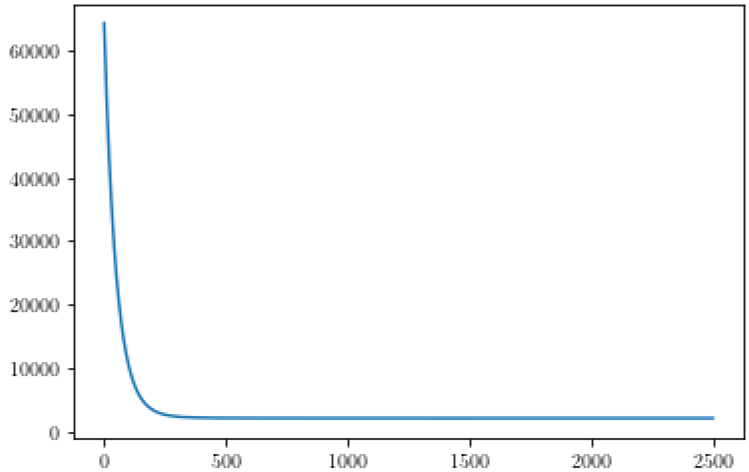
In [21]:
```
def gradient_descent(X, y, theta, alpha, num_iter):

    m = y.size
    # Save error value for each iteration to keep track of the improvement
    J_history = np.zeros(num_iter)

    for i in range(num_iter):
        theta -= alpha * (1/m) * np.transpose(X) @ (multi_hyp(X, theta) - y)
        J_history[i] = multi_cost(X, y, theta)
    return J_history
```

In [22]:
```
num_iter = 2500
alpha = 1e-2
J_history = gradient_descent(X, y, theta, alpha, num_iter)
```

In [23]:
```
#show theta
theta
```

Out[23]:
```
array([340.41265957, 110.62984204,  -6.64826603])
```

In [24]:
```
plt.plot([i for i in range(J_history.size)], J_history)
plt.show()
```



Visualization

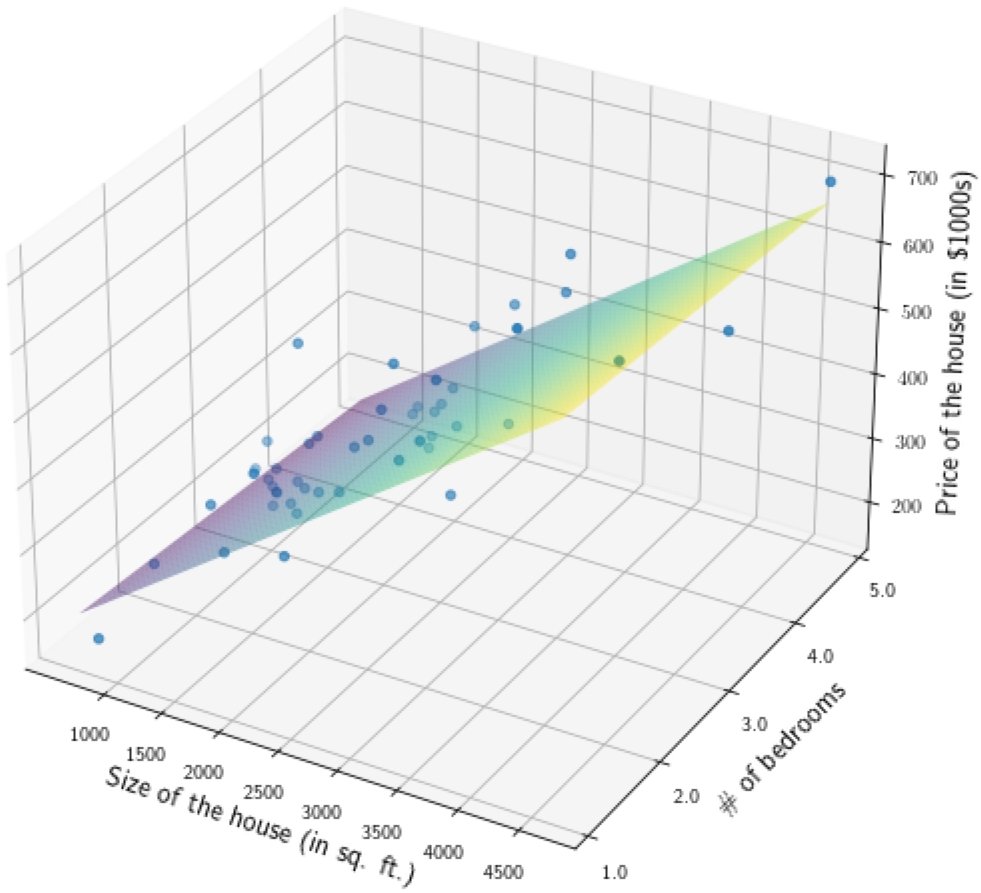
In [25]:

```
num_example = 100

x_data = np.linspace(X[:, 1].min(), X[:, 1].max(), num_example)
y_data = np.linspace(X[:, 2].min(), X[:, 2].max(), num_example)
total_data = np.column_stack((x_data, y_data))

x_pred = total_data[:, 0]
y_pred = total_data[:, 1]

x_pred = np.tile(x_pred, (num_example, 1))
y_pred = np.tile(y_pred, (num_example, 1)).T
z_pred = (x_pred * theta[1] + y_pred*theta[2] + theta[0])#/1000
```

In [26]:

```
width, height = [9,9]
fig = plt.figure(figsize=(width,height))
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x_pred,
                y_pred,
                z_pred,
                cmap=cm.viridis,
                alpha=0.5)



x = X[:, 1]

y = X[:, 2]
z = df_2.iloc[:, 2]
ax.scatter(x, y, z)



# Labels of axis
```

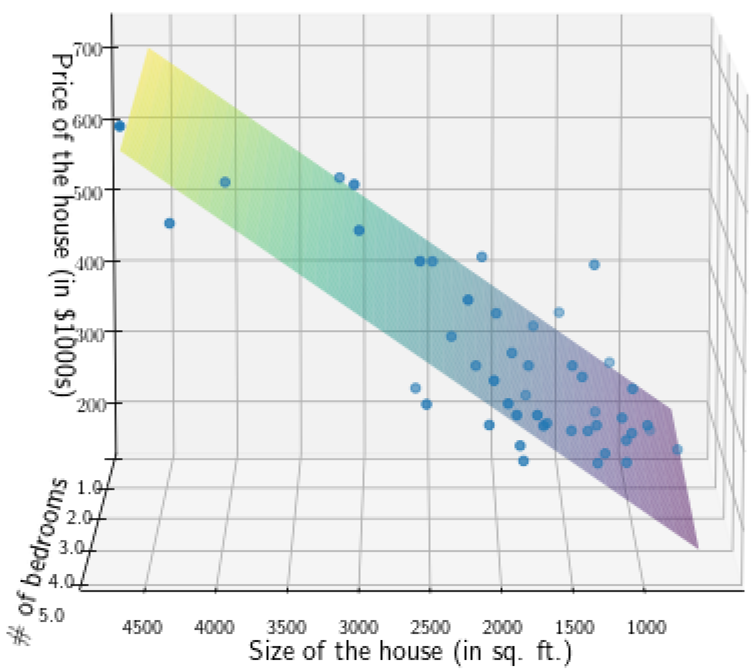
```python
ax.set_xlabel(df_2.columns[0], fontsize= axis_font_size )
ax.set_ylabel(df_2.columns[1], fontsize = axis_font_size )
ax.set_zlabel(df_2.columns[2], fontsize = axis_font_size )




# ytick
y_tick_loc = np.arange(-3, 4, 7/5)
y_tick = np.arange(1.0, 6)
ax.set_yticks(y_tick_loc)
ax.set_yticklabels(y_tick)

# xtick
x_tick_loc = np.arange(-1, 3.5, 4.5/8)
x_tick = np.arange(1000, 5000, 500)
ax.set_xticks(x_tick_loc)
ax.set_xticklabels(x_tick)



plt.savefig("./graph/multi_var_regression_result", dpi = 300)



plt.show()
```



In [27]:

```
width, height = [9,9]
fig = plt.figure(figsize=(width,height))
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x_pred,
                y_pred,
                z_pred,
                cmap=cm.viridis,
                alpha=0.5)

ax.scatter(x, y, z)



# Labels of axis
ax.set_xlabel(df_2.columns[0], fontsize= axis_font_size )
ax.set_ylabel(df_2.columns[1], fontsize = axis_font_size )
ax.set_zlabel(df_2.columns[2], fontsize = axis_font_size )
#plt.savefig("multi_var_regression_result", dpi = 300)

ax.view_init(elev = 10, azim=90)

# ytick
y_tick_loc = np.arange(-3, 4, 7/5)
```

```
y_tick = np.arange(1.0, 6)
ax.set_yticks(y_tick_loc)
ax.set_yticklabels(y_tick)

# xtick
x_tick_loc = np.arange(-1, 3.5, 4.5/8)
x_tick = np.arange(1000, 5000, 500)
ax.set_xticks(x_tick_loc)
ax.set_xticklabels(x_tick)




plt.savefig("./graph/multi_var_regression_result_diffangle", dpi = 300)
plt.show()
```



6.1 Testing with other learning rate (α)

Test will be conducted with 1e-3, 1e-2, 1e-1 for 400 iteration. The result will be graphed.

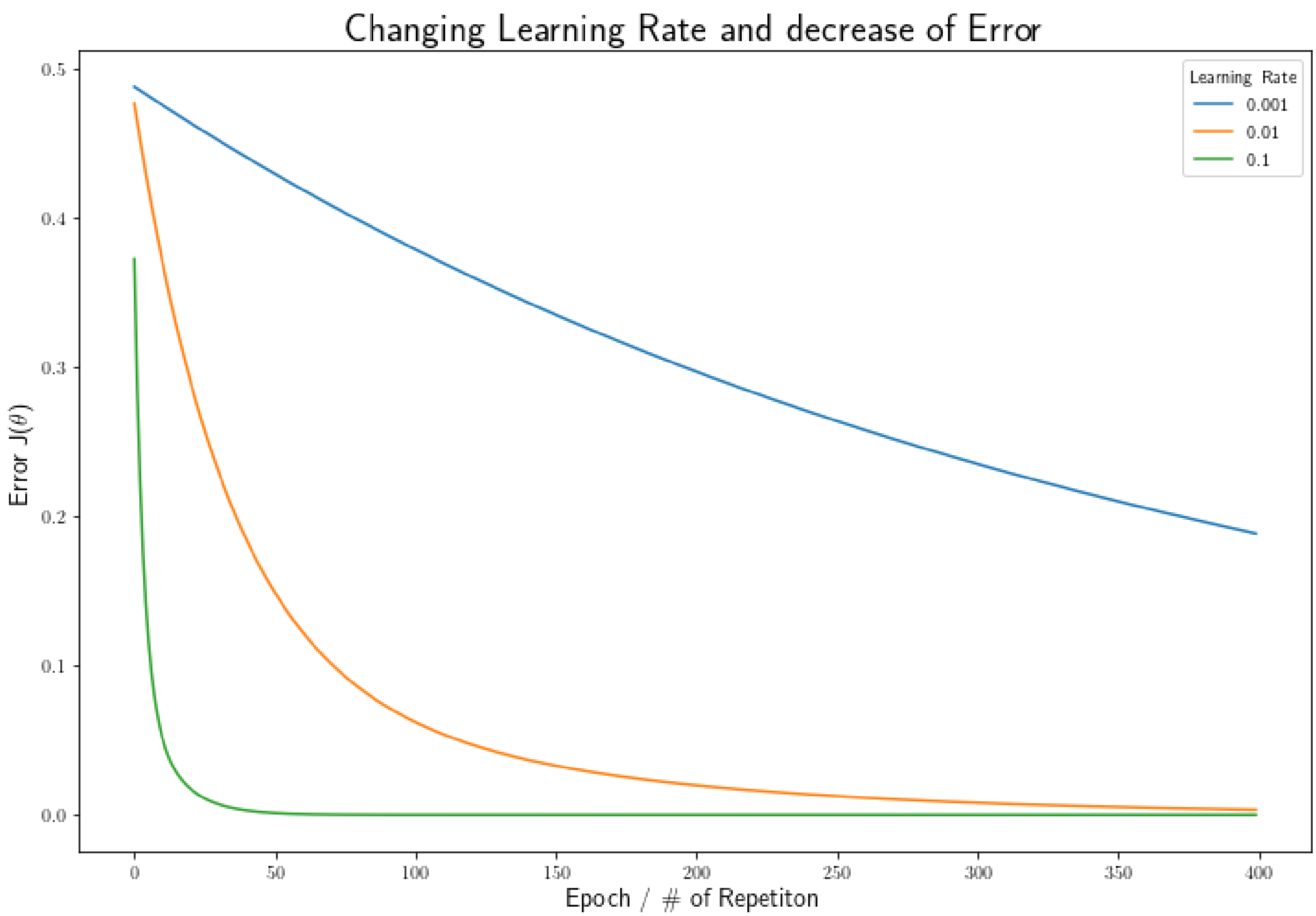
In [28]:

```
num_iter = 400
alphas = [1e-3, 1e-2, 1e-1]
result = []
```

```python
for alpha in alphas:
    theta = np.zeros(X.shape[1])
    descent_result = gradient_descent(X, y, theta, alpha, num_iter)

    result.append(descent_result)
```

In [29]:

```python
plotting_range = [i for i in range(num_iter)]

width, height = [12,8]
fig,ax = plt.subplots(figsize=(width,height))
for trials in range(len(alphas)):
    ax.plot(plotting_range, result[trials])
ax.legend(alphas, title = "Learning Rate")
ax.set_xlabel("Epoch / \# of Repetiton", fontsize= axis_font_size )
ax.set_ylabel(r"Error J($\theta$)", fontsize = axis_font_size )
ax.set_title("Changing Learning Rate and decrease of Error", fontsize =
axis_font_size * 1.5)
plt.savefig("./graph/learning_rate_error", dpi = 300)
plt.show()
```



7. Normal Equation

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

In [30]:
```
X = df_2.iloc[:, :2]
#X = (X - X.mean())/X.std()

X = np.column_stack((np.ones(X.shape[0]), X))

y = df_2.iloc[:, 2]
theta = np.linalg.inv((np.transpose(X)@X))@np.transpose(X)@y
```

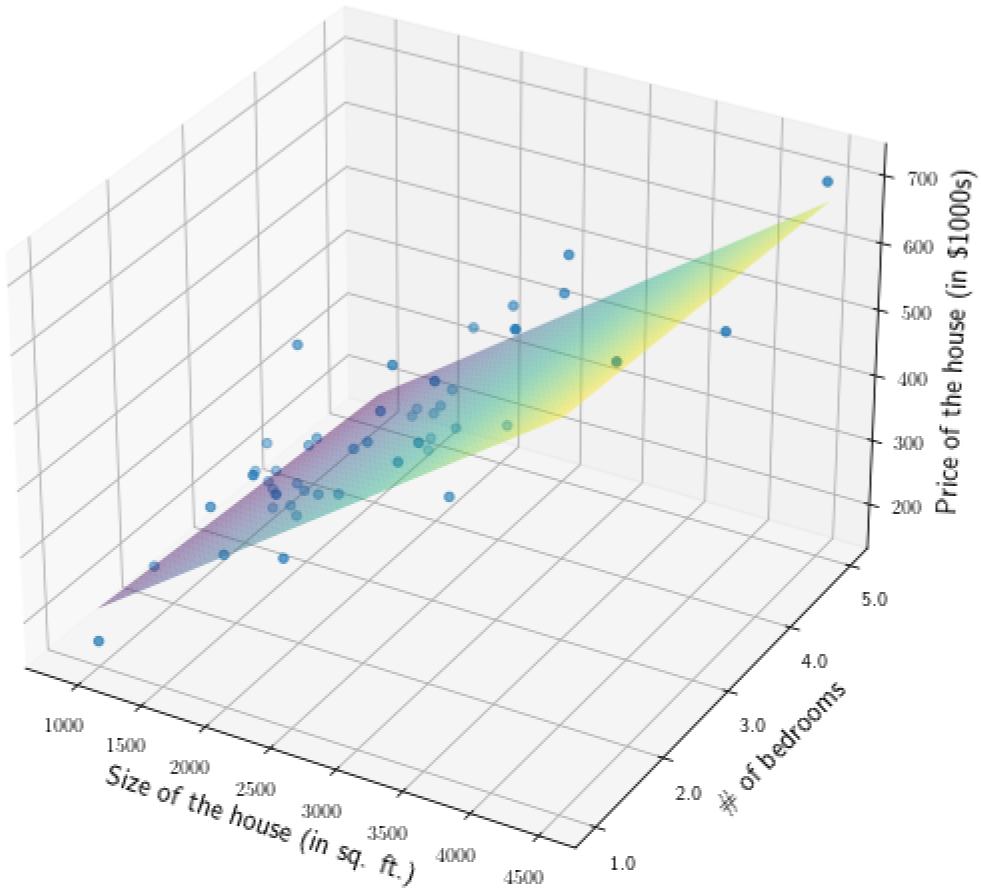
In [31]:
```
theta
```

Out[31]:
```
array([89.59790954,  0.13921067, -8.73801911])
```

7.1 Graphing the found theta

In [32]:
```
num_example = 100

x_data = np.linspace(1000, 4500, num_example)
y_data = np.linspace(1, 5, num_example)
total_data = np.column_stack((x_data, y_data))



x_pred = total_data[:, 0]
y_pred = total_data[:, 1]

x_pred = np.tile(x_pred, (num_example, 1))
y_pred = np.tile(y_pred, (num_example, 1)).T
z_pred = (x_pred * theta[1] + y_pred*theta[2] + theta[0])
width, height = [9,9]
fig = plt.figure(figsize=(width,height))
ax = fig.add_subplot(projection='3d')

ax.plot_surface(x_pred,y_pred,z_pred, alpha=0.5, cmap=cm.viridis,)
ax.scatter(df_2.iloc[:, 0], df_2.iloc[:, 1], df_2.iloc[:, 2])

# ytick
y_tick_loc = np.arange(1, 6)
y_tick = np.arange(1.0, 6)
ax.set_yticks(y_tick_loc)
ax.set_yticklabels(y_tick)

# Labels of axis
ax.set_xlabel(df_2.columns[0], fontsize= axis_font_size )
ax.set_ylabel(df_2.columns[1], fontsize = axis_font_size )
ax.set_zlabel(df_2.columns[2], fontsize = axis_font_size )
```

```
plt.savefig("./graph/normaleq_result", dpi = 300)

plt.show()
```



In [33]:
```
num_example = 100

x_data = np.linspace(1000, 4500, num_example)
y_data = np.linspace(1, 5, num_example)
total_data = np.column_stack((x_data, y_data))


x_pred = total_data[:, 0]
y_pred = total_data[:, 1]

x_pred = np.tile(x_pred, (num_example, 1))
y_pred = np.tile(y_pred, (num_example, 1)).T
z_pred = (x_pred * theta[1] + y_pred*theta[2] + theta[0])
width, height = [9,9]
fig = plt.figure(figsize=(width,height))
ax = fig.add_subplot(projection='3d')
```

```
ax.plot_surface(x_pred,y_pred,z_pred, alpha=0.5, cmap=cm.viridis,)
ax.scatter(df_2.iloc[:, 0], df_2.iloc[:, 1], df_2.iloc[:, 2])
ax.view_init(elev = 10, azim=90)

# ytick
y_tick_loc = np.arange(1, 6)
y_tick = np.arange(1.0, 6)
ax.set_yticks(y_tick_loc)
ax.set_yticklabels(y_tick)

# Labels of axis
ax.set_xlabel(df_2.columns[0], fontsize= axis_font_size )
ax.set_ylabel(df_2.columns[1], fontsize = axis_font_size )
ax.set_zlabel(df_2.columns[2], fontsize = axis_font_size )
plt.savefig("./graph/normaleq_result_diff_ang", dpi = 300)
plt.show()
```