Lab 6: Cost Functions and Parameter Space

Author: Nigel Nelson

Introduction:

- This lab is an exploration into cost functions as well as parameter space. This exploration has the purpose of increasing the understanding of these individual spaces, but also to give context to optimizing parameters, specifically through grid search. This lab challenges students to write their own python implementation of Gaussian and Linear models, and also implement cost functions that allow rapid testing of different sets of parameters on the same model that uses identical data. This is done so that students can then analyze the resulting amount of error produced from these models for a specific set of parameters. In order to automate the search for optimized parameters, students use the grid search technique to locate the lowest amount of error from a specified parameter space. Using both Gaussian and Linear models, students explore the pros and cons of the grid search for low dimension parameter spaces, as well as higher dimension parameter spaces.

Results:

- The first experiment ran was parameter optimization for a Gaussian model using grid search. The first part of this experiment was searching for optimum Mu values ranging from 0 to 10, in steps sizes of 0.10, and searching for optimum Sigma values in the range of 0.5 to 2, in step sizes of 0.03. Through visual analysis of the resulting heatmap it was found that the parameters that produced the lowest amount of error were Mu values between 4 to 6.5, and Sigma values between 1.0 to 1.75. To further refine this range another heat plot was created with greater resolution of the parameter space. This second grid search consisted of Mu values ranging from 5 to 6, in steps sizes of 0.01, and searching for optimum Sigma values in the range of 1.0 to 1.75, in step sizes of 0.015. From this heatmap it was visually determined that the parameters that produced the lowest amount of error were Mu values between 5.35 to 5.65, and Sigma values between 1.18 to 1.35. However, by using the numpy.argmin function, it was ultimately determined that for this described grid search, the parameter set that produced the lowest amount of error, an MSE score of **~4.33 * 10^-7**, was a Mu value of **~5.505**, and a Sigma value of **~1.245**. The second experiment was a parameter optimization experiment, however the model used was a linear model that required 4 parameters to tune the model. Again, a grid search technique was used where each of the 4 parameters was tested on a range of values between -1 to 1, in step sizes of 0.04. By using the numpy.argmin function on the resulting MSE scores, it was ultimately determined that for this described grid search, the parameter set that produced the lowest amount of error, an MSE score of **~4.795**, was a B0 value of **~0.143**, a B1 value of **~0.061**, a B2 value of **~0.184**, and a B3 value of **~0.020**.

Questions:

1. By looking at the provided cost_functions.py, use 1-2 sentences to describe in detail the purpose of each of the methods. To guide this description, discuss the method input, method output, and what function each method serves for the cost function.

    - **"__init__"** has the purpose of saving the features and the true responses of the model. It takes the features and true responses as arguments so that it can save them as class variables, this allows the cost function to calculate MSE with only needing the chosen parameters as an argument.
    - **"predict"** has the purpose of calculating the expected response variables in relation to the provided parameters and features. It takes parameters and features as arguments, both of which can be supplied by the cost function, so that in turn in return the predicted responses needed by the cost function.
    - **"_mse"** has the purpose of calculating the Mean Squared Error. It takes the true response variables and the predicted response variables as input in order to return a score that represents the amount of error in the predictions. This is used by the cost to return the error for the parameters the cost function was supplied.
    - **"cost"** has the purpose of accepting parameters as arguments, and returns the amount of error produced by the model with those supplied parameters. This is so that a multitude of parameters can be tried on the same model with the same data in order to draw conclusions on how the parameters affect the amount of error produced.
2. For the heatmaps that you generated for this lab, what do they describe? What do the “valleys” and “peaks” of this heat map represent?

    - The heatmaps generated for this lab describe the error that results for a given set of parameters. The valleys correspond to parameters that result in a lower amount of error, and the peaks correspond to parameters that result in a higher amount of error.
3. For experiment 2, you increased the number of samples within the specified range.
    A. Describe how the heatmap representation changed due to this increase in sampling.
        - In experiment 2, an area that corresponded to low amounts of error in experiment 1 was focused on, and the heatmap for experiment 2 essentially represents zooming in on that area of low error, but with higher resolution to further dissect that area of interest to further refine precisely what parameters result in the lowest amount of error.
    B. What benefit did this higher sampling rate have for finding the set of parameters with the minimum error?
        - The benefit of further refining this area of relatively low error, is that it allowed greater precision in determining the exact parameter values that led to the lowest amount of error.
    C. Was this sampling rate high enough? Defend your answer!
        - Whether or not the sampling rate was high enough depends on how the resulting model is intended to be used. If models must be produced rapidly and it is acceptable to have predictions that are "close enough" then this sampling rate is acceptable. However, if this model is intended to be used in a problem space where accuracy is of the utmost importance, then it would be advantageous to further increase the sampling rate interactively until the amount of error reduced for each iteration converges to a consistent amount of minimum error.
4. The Gaussian distribution model is limited to two dimensions while the multivariate linear model implemented for this lab is 4 dimensional.

A. Describe a limitation of the grid search method as you add additional dimensions. Hint: Think about the time complexity required for the grid search as you add additional dimensions.
B. The limitation of the grid search method is that as dimensions are added, the time complexity increases exponentially. This results in greater and greater times to determine the optimum parameters to be used. Ultimately this means that if dozens of parameters are used, this method of determining optimum parameters is extremely time inefficient.
C. With time complexity in mind, can you derive a rule (mathematical expression) to estimate how many grid points are needed to evaluate all combination of parameters based on the number of dimensions.
D. If each dimension d is to be tested in a range of n values, then the resulting time complexity would be n^d. This means that for each dimension added, the time to find the optimum set of parameters increases exponentially.
E. With this rule, compare 2-dimensional models with 4-dimensional models. 10-dimensional? 100-dimensional?
F. 2 dimensional model: n^2
G. 4 dimensional model: n^4
H. 10 dimensional model: n^10
I. 100 dimensional model: n^100

5. In experiment 3 you plotted the line of identity in the figure that compared the given response variable to the model prediction.
   A. What does this line represent and how is it useful?
   B. The line of identity represents a perfect 1:1 correlation between the predicted sale values, and the true sale values. In other words, it represents where a point would lie if a prediction point was perfectly accurate in predicting the sale value.
   C. What does it mean for a value to lie above the line? Below the line?
   D. If a point lies above the line, this means that the prediction value predicted less sales than there actually were. If the point lied below the line, this means that the prediction value predicted more sales than there actually were.
   E. How would predictions that perfectly replicate the given data appear in this plot?
   F. If the predictions perfectly replicated the given data, all points would fall exactly on the line of identity.
6. What are the weaknesses of grid search? Why wouldn't we want to use it?

- The first weakness of the grid search is the time required to approximate the best set of parameters. Without any heuristic used, every possible combination of parameters must be iterated through and tested. This process is much more time consuming that search methods that use a heuristic in order to only test parameter values that trend towards a lower amount of error. Another weakness of grid search is that it is limited to the resolution of grid points searched. If points used in the grid search are only whole values, and the true optimum set of parameters lie on half values, then that true optimum will not be found, and instead only a relative optimum will be discovered. Lastly, a weakness of grid search is that it limits the range of values tested. If parameter values between 0 and 10 are only tested, but the optimum values actually occur in a range of values greater than 10, grid search does not find this set of points and instead only finds the approximate best set of parameters within the specified range.

Imports:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from cost_functions import *
from test_cost_functions import *
```

Validating cost_functions.py implementation:

In [2]:
```
!python test_cost_functions.py
```

```
......
----------------------------------------------------------------------
Ran 6 tests in 0.039s

OK
```

Experiment 1: Coarse Grid Search - Gaussian Distribution

Loading gaussdist.csv:

In [3]:
```
data = np.loadtxt(open("gaussdist.csv", "rb"), delimiter=",")
data[:5]
```

Out[3]:
```
array([[6.99000000e+00, 1.56842355e-01],
       [8.90000000e+00, 7.89692304e-03],
       [9.58000000e+00, 1.55088416e-03],
       [5.46000000e+00, 3.18990459e-01],
       [1.38000000e+00, 1.39635489e-03]])
```

- **Identifying response variable and feature variable:**
  - Based from the Gaussian Distribution data that is loaded in the above cell, it is clear that the second column is not monotonic, which is a characteristic of the output for a Gaussian Distribution. As such, the first column are the feature variable values, and the second column are the response variable values.

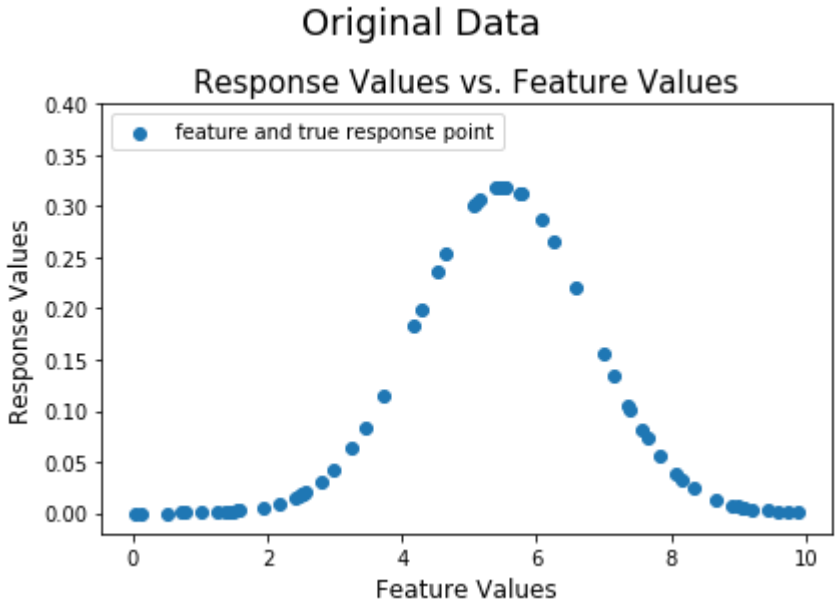
Instantiating GaussianCostFunction using the provided data:

In [4]:
```
features = data[:, 0]
responses = data[:, 1]

gaussian_func = GaussianCostFunction(features, responses)
```

Plotting the provided data:

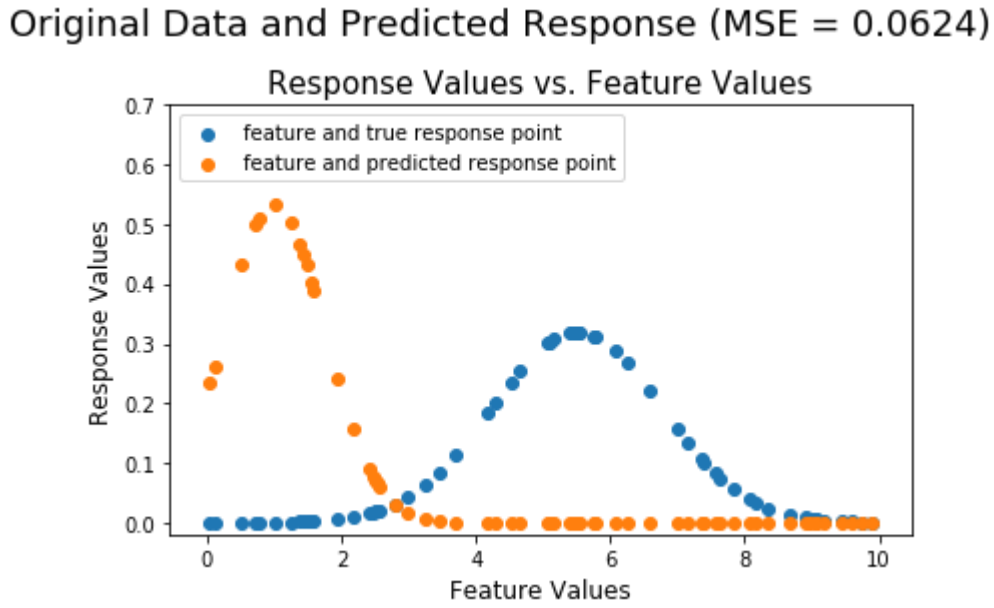
In [5]:
```
plt.scatter(features, responses, label='feature and true response point')
plt.title("Response Values vs. Feature Values", fontsize=15)
plt.xlabel("Feature Values", fontsize=12)
plt.ylabel("Response Values", fontsize=12)
plt.suptitle('Original Data', fontsize=18, y=1.07)
plt.legend(loc=2)
plt.ylim(-.02, .4)
plt.tight_layout()
plt.tight_layout()
```



Plotting the provided data alongside predicted output:

In [6]:
```
pred_y = gaussian_func.predict(features, np.array([1, 0.75]))
mse_score = gaussian_func.cost(np.array([1, 0.75]))

plt.scatter(features, responses, label='feature and true response point')
plt.scatter(features, pred_y, label='feature and predicted response point')
plt.title("Response Values vs. Feature Values", fontsize=15)
plt.xlabel("Feature Values", fontsize=12)
plt.ylabel("Response Values", fontsize=12)
plt.suptitle(f'Original Data and Predicted Response (MSE = {round(mse_score,
plt.legend(loc=2)
plt.ylim(-.02, .7)
plt.tight_layout()
```



Grid Search - Coarse:

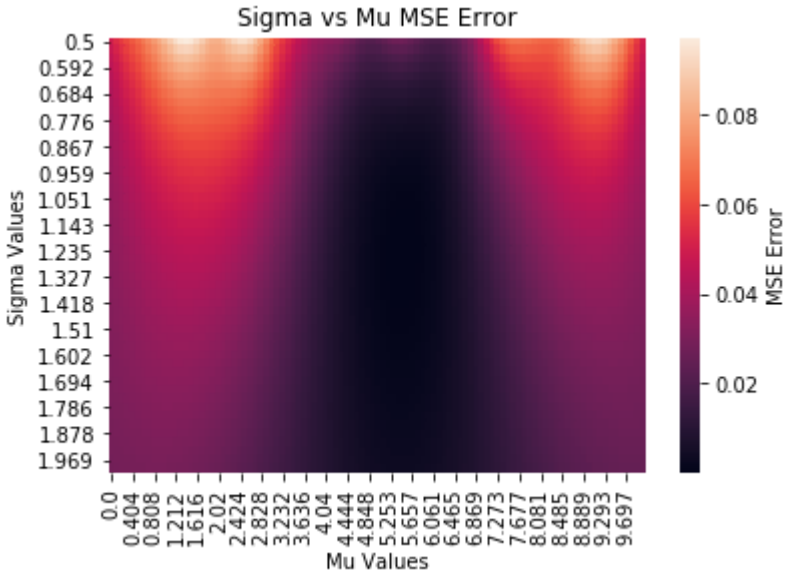
In [7]:
```
mu = np.linspace(0, 10, 100)
sigma = np.linspace(0.5, 2, 50)
mus, sigmas = np.meshgrid(mu, sigma)
params = np.array([mus.flatten(), sigmas.flatten()]).T
mse = np.zeros(params[:, 0].shape)

for i in range(mse.size):
    mse[i] = gaussian_func.cost(params[i])
```

Heatmap of MSE error for coarse grid search:

In [8]:
```
df = pd.DataFrame(data=mse.reshape(mus.shape), columns=mu.round(3), index=sig
sns.heatmap(df, cbar_kws={'label':'MSE Error'})
plt.xlabel("Mu Values")
plt.ylabel("Sigma Values")
plt.title("Sigma vs Mu MSE Error")
```

Out[8]:
```
Text(0.5, 1, 'Sigma vs Mu MSE Error')
```



Visually selecting optimized parameters:

In [9]:
```
chosen_params = np.array([5, 1.25])
mse_score = gaussian_func.cost(chosen_params)
print(f'Error for mu=5, sigma=1.8: {mse_score}')
```

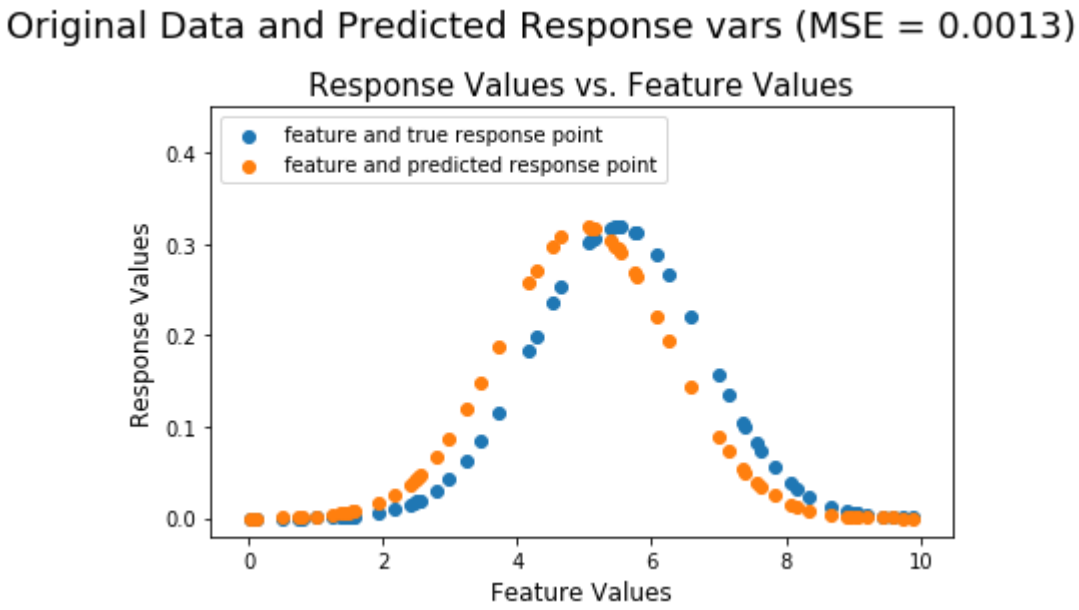
```
Error for mu=5, sigma=1.8: 0.0013108572886184819
```

Plotting provided data alongside predicted output for selected parameters:

In [10]:
```
pred_y = gaussian_func.predict(features, chosen_params)

plt.scatter(features, responses, label='feature and true response point')
plt.scatter(features, pred_y, label='feature and predicted response point')
plt.title("Response Values vs. Feature Values", fontsize=15)
plt.xlabel("Feature Values", fontsize=12)
plt.ylabel("Response Values", fontsize=12)
plt.legend()

plt.suptitle(f'Original Data and Predicted Response vars (MSE = {round(mse_sc
plt.legend(loc=2)
plt.ylim(-.02, .45)
plt.tight_layout()
```



Experiment 2: Refined Grid Search - Gaussian Distribution

Grid Search - Refinement Pass

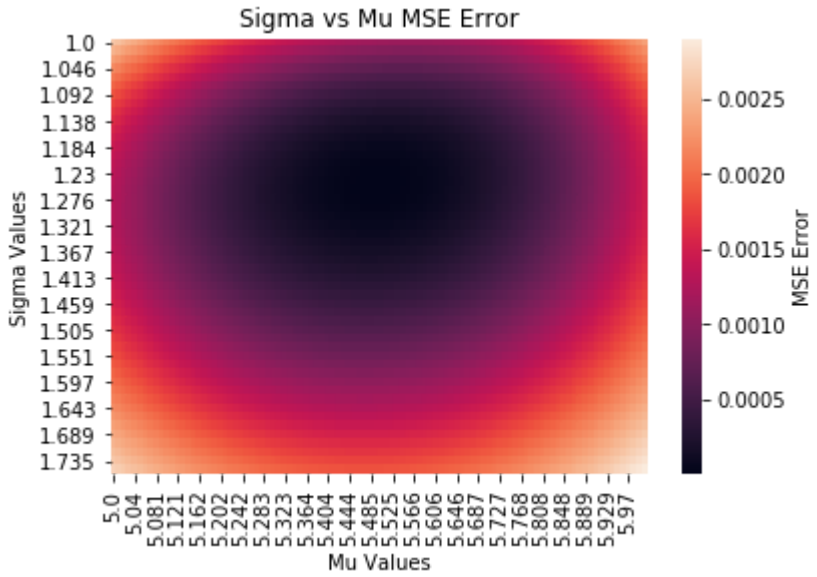
In [11]:
```
mu = np.linspace(5, 6, 100)
sigma = np.linspace(1, 1.75, 50)
mus, sigmas = np.meshgrid(mu, sigma)
params = np.array([mus.flatten(), sigmas.flatten()]).T
mse = np.zeros(params[:, 0].shape)

for i in range(mse.size):
    mse[i] = gaussian_func.cost(params[i])
```

Heatmap of MSE error for refinement pass grid search:

In [12]:
```
df = pd.DataFrame(data=mse.reshape(mus.shape), columns=mu.round(3), index=sig
sns.heatmap(df, cbar_kws={'label':'MSE Error'})
plt.xlabel("Mu Values")
plt.ylabel("Sigma Values")
plt.title("Sigma vs Mu MSE Error")
```

Out[12]: Text(0.5, 1, 'Sigma vs Mu MSE Error')



Visually selecting optimized parameters:

In [13]:
```
chosen_params = np.array([5.485, 1.276])
mse_score = gaussian_func.cost(chosen_params)
print(f'Error for Mu=5, Sigma=1.8: {mse_score}')
```

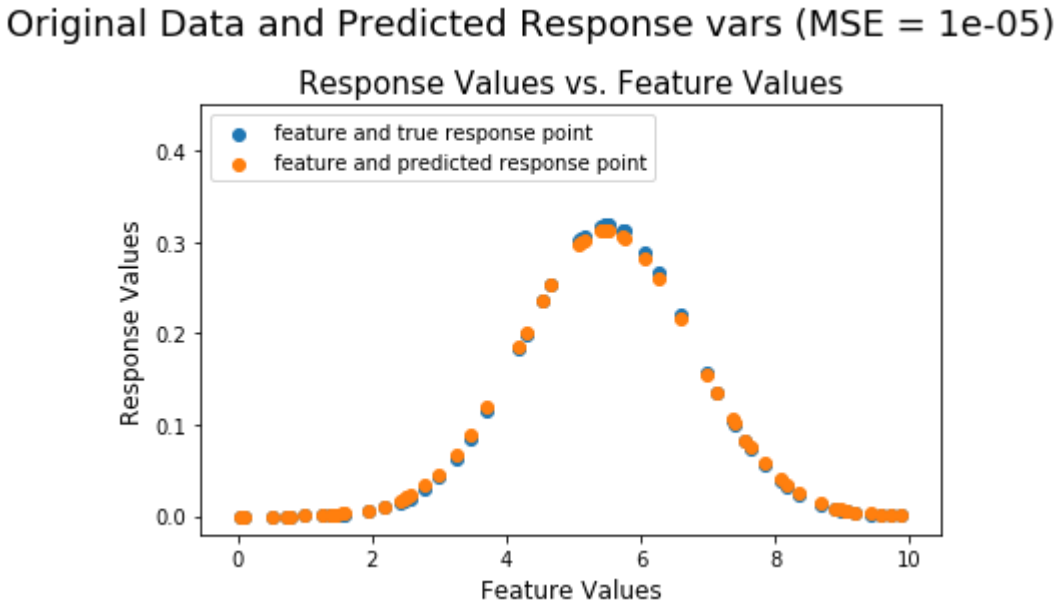
```
Error for Mu=5, Sigma=1.8: 9.028286276230895e-06
```

Plotting provided data alongside predicted output for selected parameters:

In [14]:
```
pred_y = gaussian_func.predict(features, chosen_params)

plt.scatter(features, responses, label='feature and true response point')
plt.scatter(features, pred_y, label='feature and predicted response point')
plt.title("Response Values vs. Feature Values", fontsize=15)
plt.xlabel("Feature Values", fontsize=12)
plt.ylabel("Response Values", fontsize=12)
plt.legend()

plt.suptitle(f'Original Data and Predicted Response vars (MSE = {round(mse_sc
plt.legend(loc=2)
plt.ylim(-.02, .45)
plt.tight_layout()
```



Finding minimum MSE error for refinement pass grid search:

In [15]:
```
mse_idx = np.argmin(mse)
min_mse = mse[mse_idx]
min_mu = params[mse_idx, 0]
min_sigma = params[mse_idx, 1]
print(f'Minimum MSE score in current parameter space: {min_mse}')
print(f'Coresponding Mu value: {min_mu}')
print(f'Coresponding Sigma value: {min_sigma}')
```

```
Minimum MSE score in current parameter space: 4.330705127067271e-07
Coresponding Mu value: 5.505050505050505
Coresponding Sigma value: 1.2448979591836735
```

Experiment 3: “Blind” Grid Search – Multivariate Linear Model

Loading advertising.csv data:

In [16]:
```
data_E2 = np.loadtxt(open("advertising.csv", "rb"), delimiter=",", skiprows=1
data_column_labels = np.loadtxt(open("advertising.csv", "rb"), delimiter=",",
print("Advertising data shape: " + str(data_E2.shape))
print("Advertising data column labels: " + str(data_column_labels))
```

```
Advertising data shape: (200, 6)
Advertising data column labels: [b'' b'Offset' b'TV' b'radio' b'newspaper'
b'sales']
```

- **Identifying response variable and feature variables:**
  - After viewing the labels for the Advertising data set, it is clear that the feature variables are the second, third, fourth, and fith columns, and the response variable is the final 'sales' column.

Instantiating LinearCostFunction using the provided data:

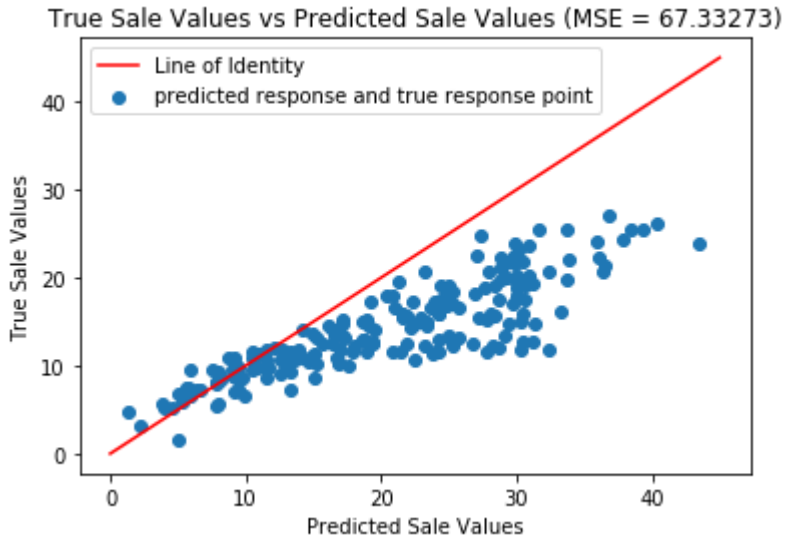
In [17]:
```
features = data_E2[:, 1:5]
sale_values = data_E2[:, 5]

linear_model = LinearCostFunction(features, sale_values)
```

Plotting model predictions versus true response:

In [18]:
```
params = np.array([0.1, 0.1, 0.1, 0.1])
pred_y = linear_model.predict(features, params)
mse_score = linear_model.cost(params)
plt.scatter(pred_y, sale_values, label='predicted response and true response
plt.plot([0, 45], [0, 45], c = 'r', label = 'Line of Identity')
plt.xlabel('Predicted Sale Values')
plt.ylabel('True Sale Values')
plt.title(f'True Sale Values vs Predicted Sale Values (MSE = {round(mse_score
plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x7fb936c98c88>



Performing grid search over 4 dimensional parameter space:

In [19]:
```
weight = np.linspace(-1, 1, 50)
w0, w1, w2, w3 = np.meshgrid(weight, weight, weight, weight)
params = np.array([w0.flatten(), w1.flatten(), w2.flatten(), w3.flatten()]).T
mse = np.zeros(params[:, 0].shape)

for i in range(mse.size):
    mse[i] = linear_model.cost(params[i])
```

Finding minimum MSE error for 4 dimensional grid search:

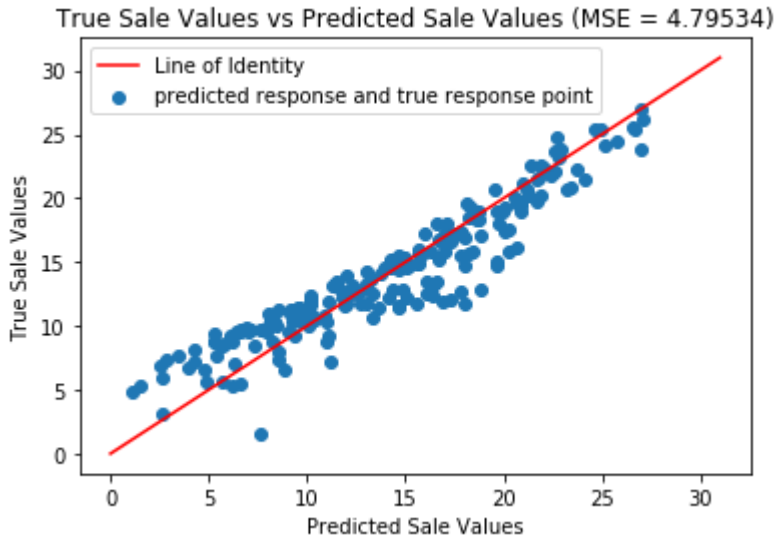
In [20]:
```
mse_idx = np.argmin(mse)
min_mse = mse[mse_idx]
min_B0 = params[mse_idx, 0]
min_B1 = params[mse_idx, 1]
min_B2 = params[mse_idx, 2]
min_B3 = params[mse_idx, 3]
print(f'Minimum MSE score in current parameter space: {min_mse}')
print(f'Coresponding B0 value: {min_B0}')
print(f'Coresponding B1 value: {min_B1}')
print(f'Coresponding B2 value: {min_B2}')
print(f'Coresponding B3 value: {min_B3}')
```

```
Minimum MSE score in current parameter space: 4.795343148688013
Coresponding B0 value: 0.1428571428571428
Coresponding B1 value: 0.06122448979591821
Coresponding B2 value: 0.18367346938775508
Coresponding B3 value: 0.020408163265306145
```

Plotting gird search optimized model predictions versus true response:

In [21]:
```
pred_y = linear_model.predict(features, params[mse_idx, :])
mse_score = linear_model.cost(params[mse_idx, :])
plt.scatter(pred_y, sale_values, label='predicted response and true response
plt.plot([0, 31], [0, 31], c = 'r', label = 'Line of Identity')
plt.xlabel('Predicted Sale Values')
plt.ylabel('True Sale Values')
plt.title(f'True Sale Values vs Predicted Sale Values (MSE = {round(mse_score
plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x7fb936c88898>



Conclusion:

- This lab was an exploration into cost functions as well as parameter space. This exploration had the purpose of increasing the understanding of these individual spaces, but also gave context to optimizing parameters, specifically through grid search. Through challenging students to implement their own models and cost functions, a greater understanding of the benefits and possible use cases for cost functions was gained. In addition, through exploring parameter optimization using grid search, vast benefits were seen over non-automated parameter optimization, however, major downsides were also uncovered such as the time complexity of this method and its inability to truly search all of parameter space.