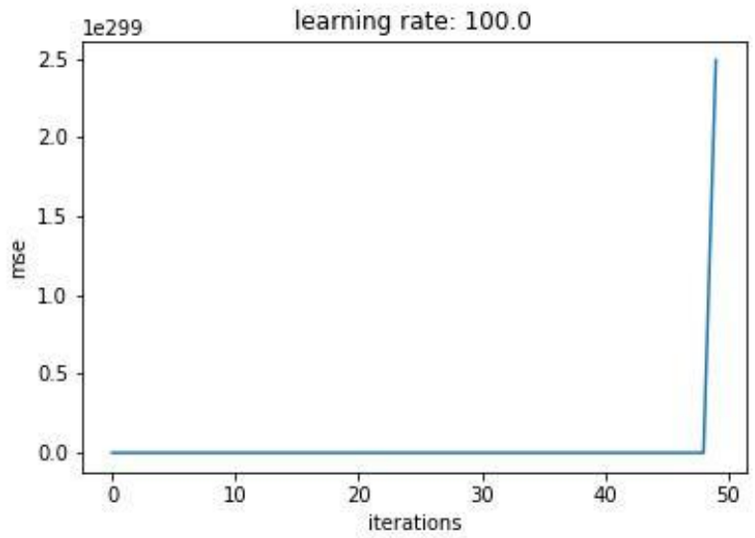
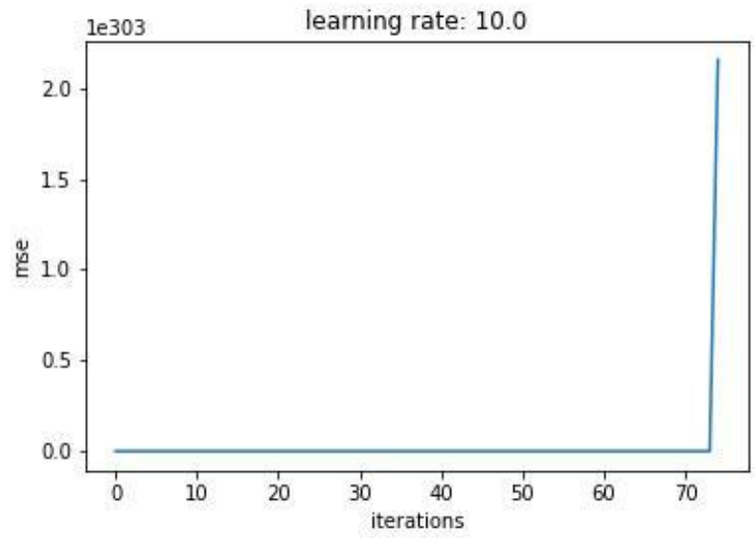
IA1 Report

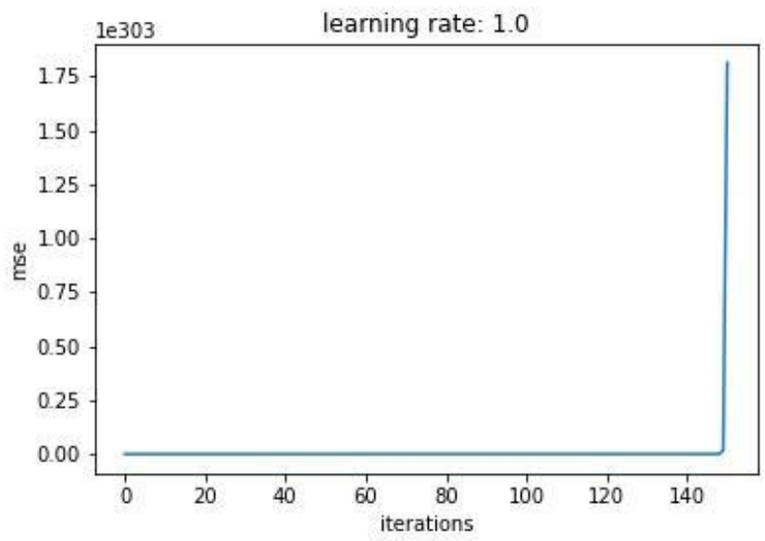
Opeyemi Ajibuwa

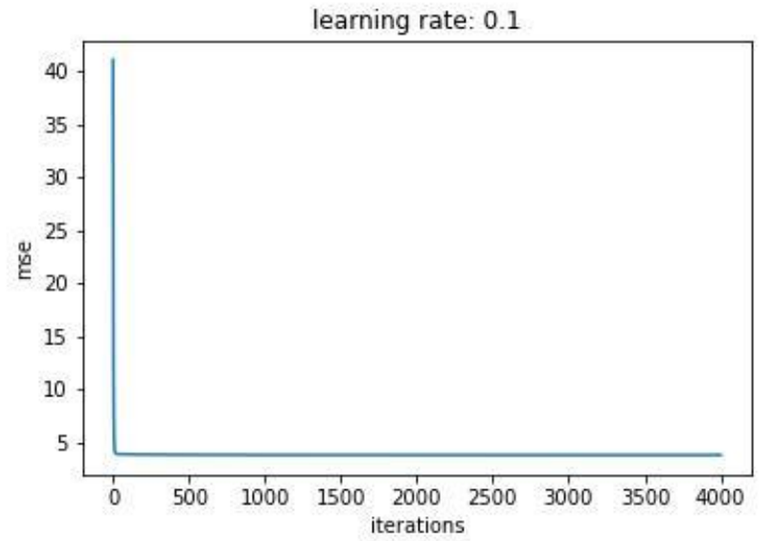
Part 1: Training with pre-processed and normalized data

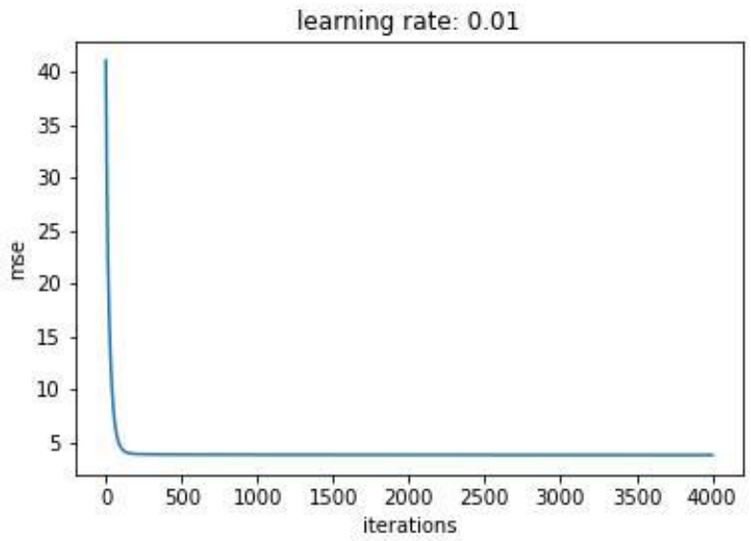
(a) The first three figures with learning rates of 1, 10 and 100 diverged, so an early stop was applied to those models. A stopping threshold of 10^-8 was also applied generally to all trained models in the case of early convergence or divergence. Also, models with learning rate of 10^-5 and above typically did not final convergence state on this normalized dataset.

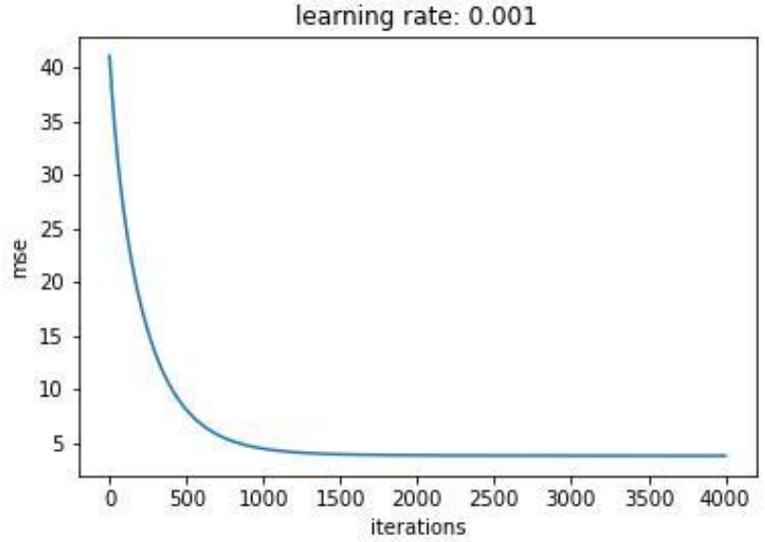


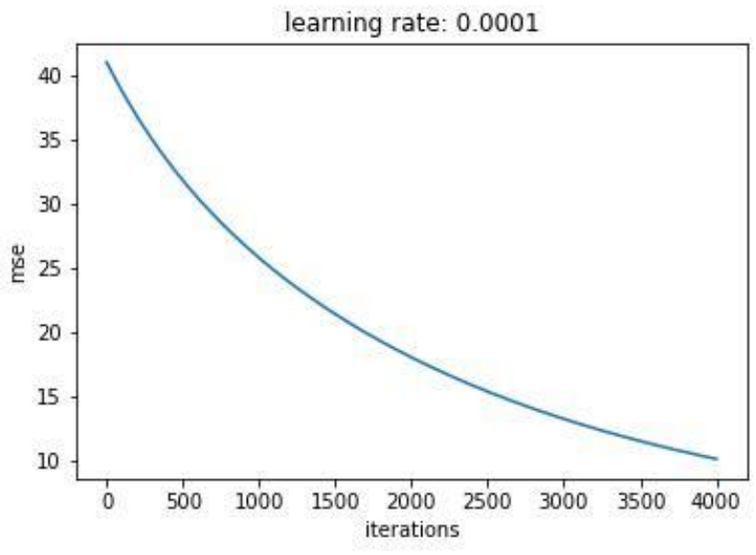


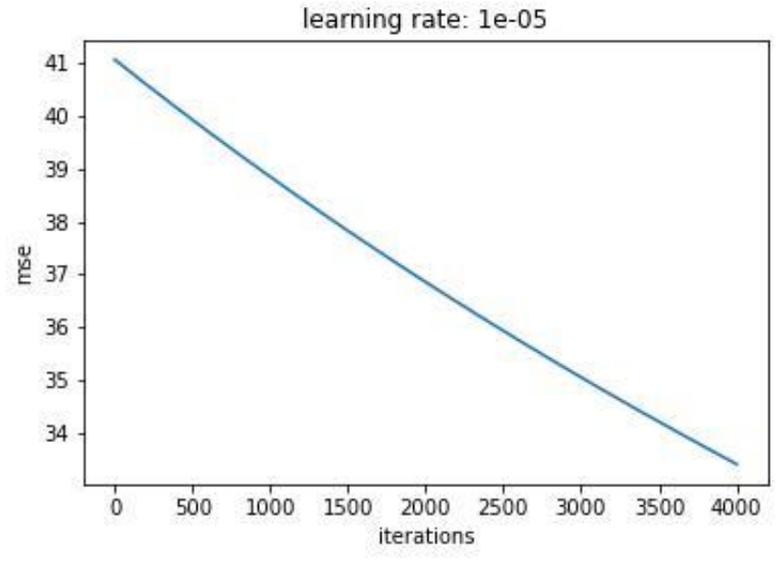












Question: Which learning rate or learning rates did you observe to be good for this particular dataset? What learning rates (if any) make the gradient descent diverge?

Learning rates 10^-1, 10^-2, 10^-3 are all good for this dataset, since these make the model converge. The learning rates of 100, 10, and 1 make the gradient descent diverge, while the learning rates of 10^-4 and 10^-5 did not reach final convergence.

(b) Since the learning rates of (>=10^-5) did not reach final convergence, I only use the three good learning rates to compute the MSE on the validation data. Below is the result:

```
learning rate=0.1, MSE=4.356874833670252
learning rate=0.01, MSE=4.461375006628152
learning rate=0.001, MSE=4.632524467113898
```

From the results, it appears the learning rate of 0.1 leads to the best validation MSE.

Under the situation of nearly identical MSE with different convergent rates, we should the model with the smallest learning rate. From the figures above, the curve of the plot with learning rate of 10^-3 is smoother, since smaller learning rate implies smaller update steps, which can avoid missing the local minima to some degrees.

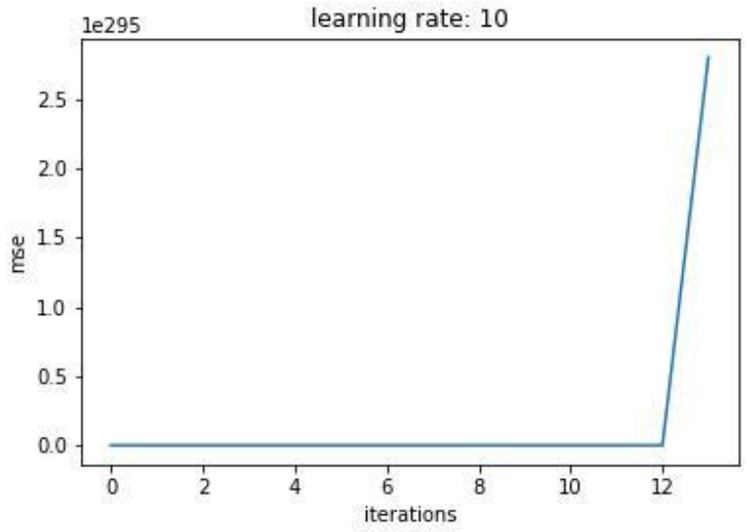
(c) The solution that best minimizes the validation MSE is the model with learning rate of 0.1, according to the MSE values listed in part 1b above. The learned weights for each feature are:

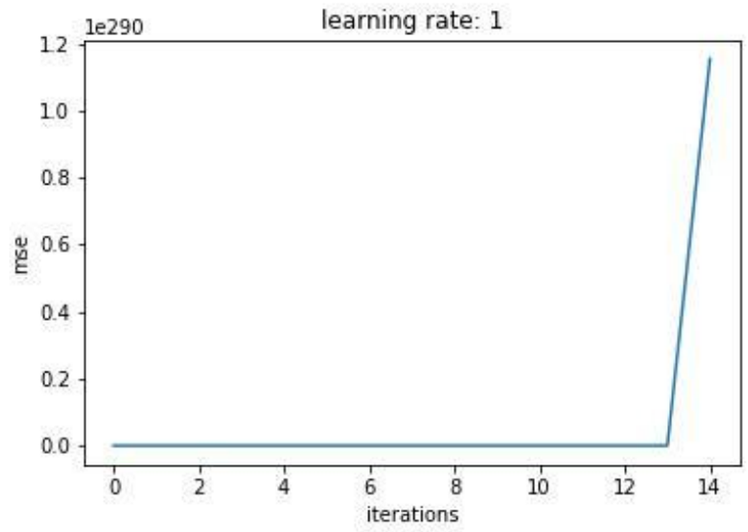
Feature	**Learned Weight**	**Feature**	**Learned Weight**
Dummy	**5.334911**	grade	1.115143
Month	0.054940	sqft_above	0.756488
Day	-0.050598	sqft_basement	0.155253
Year	0.173781	yr_built	-0.883467
Bedrooms	-0.281475	age_since_renovated	-0.102649
Bathrooms	0.339022	zipcode	-0.263433
Sqft_living	0.763553	Lat	0.836586
Sqft_lot	0.058188	Long	-0.303795
Floors	0.018143	Sqft_living15	0.143490
waterfront	3.965003	Sqft_lot15	-0.099304
view	0.448146		
condition	0.199874		

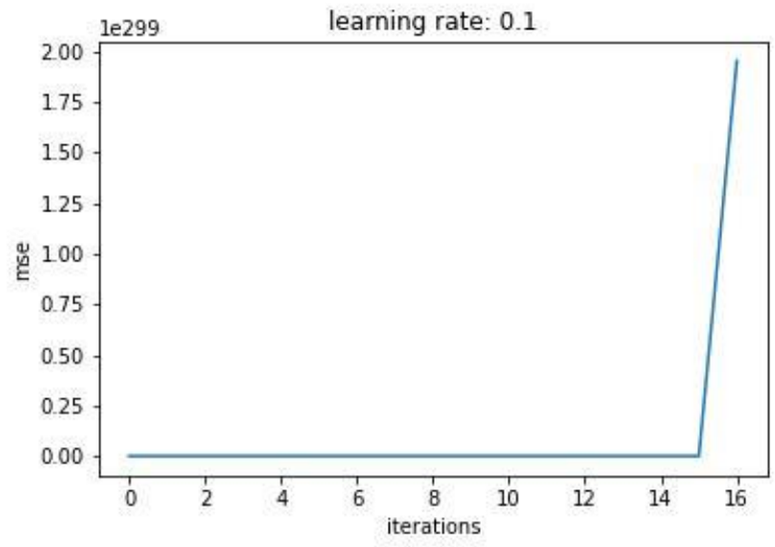
The features that are the most important besides the dummy variable is “waterfront” and followed by “grade”

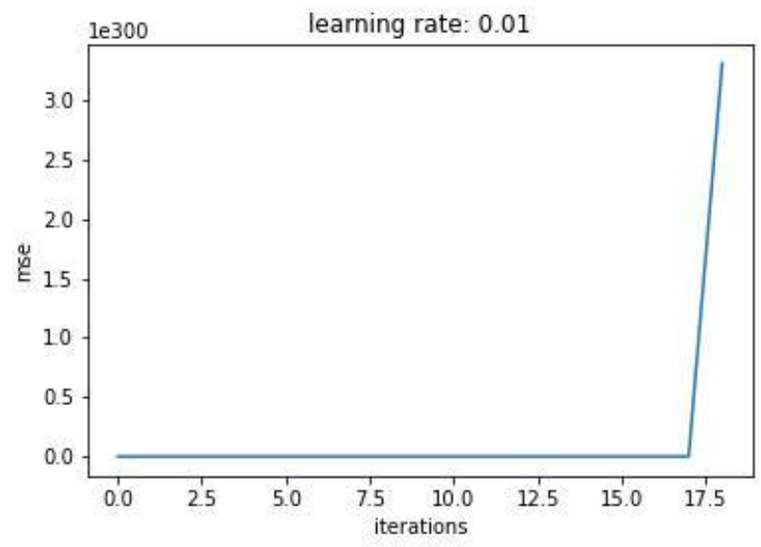
Part 2: Training with non-normalized data

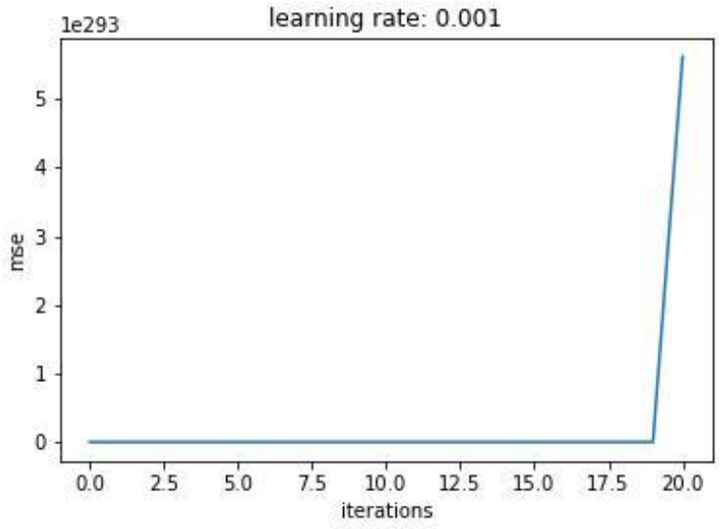
(a) The learning rates experimented with are the following: [10, 1, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14, 1e-15, 1e-16]. As we can see from the figures plotted below. Only learning rates of 1e-11, 1e-12 and 1e-13 converged, the others diverged.

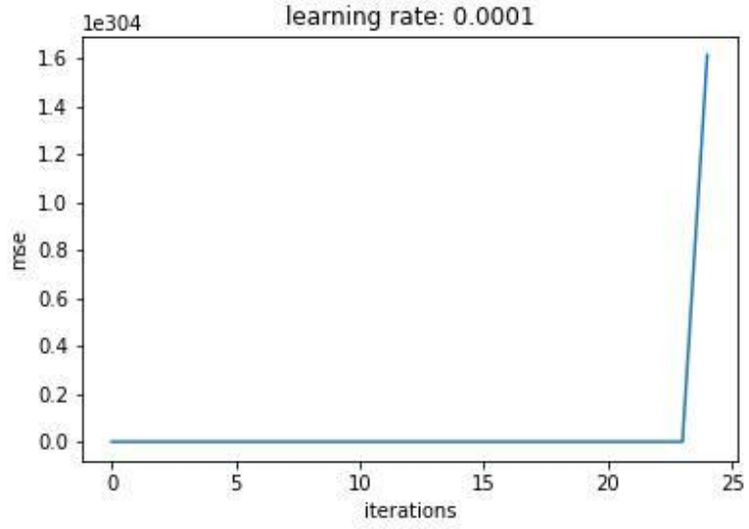

learning rate: 10
1e295
mse
iterations

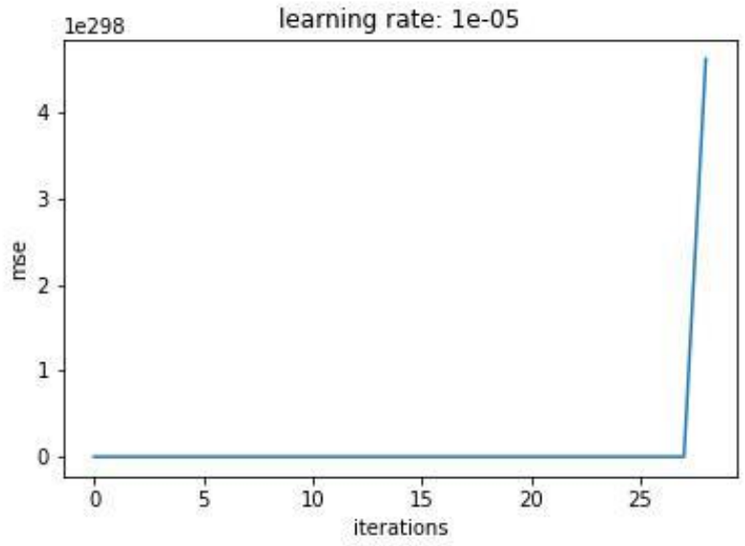

learning rate: 1
1e290
mse
iterations

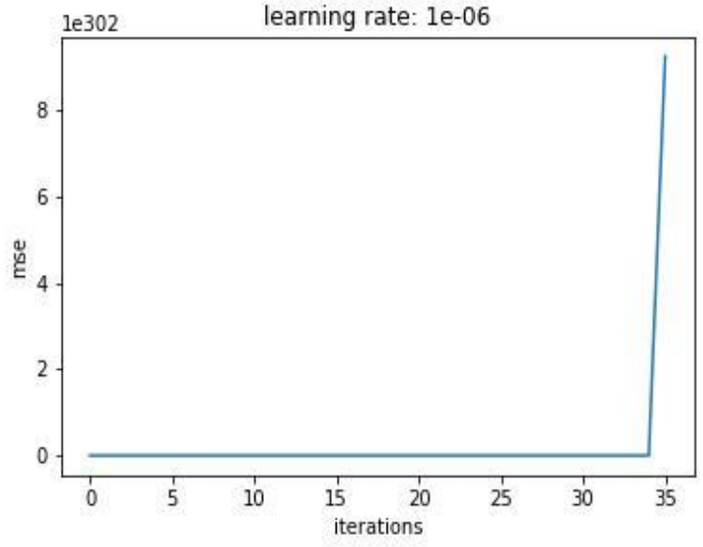

learning rate: 0.1
1e299
mse
iterations

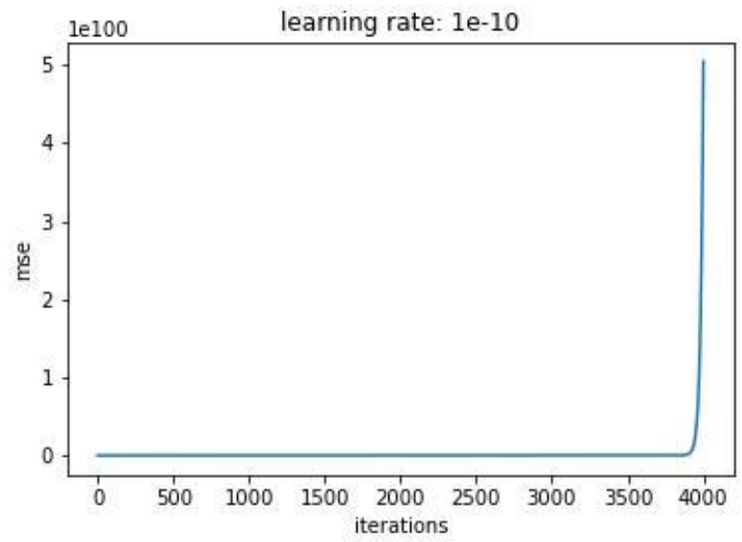

learning rate: 0.01
1e300
mse
iterations

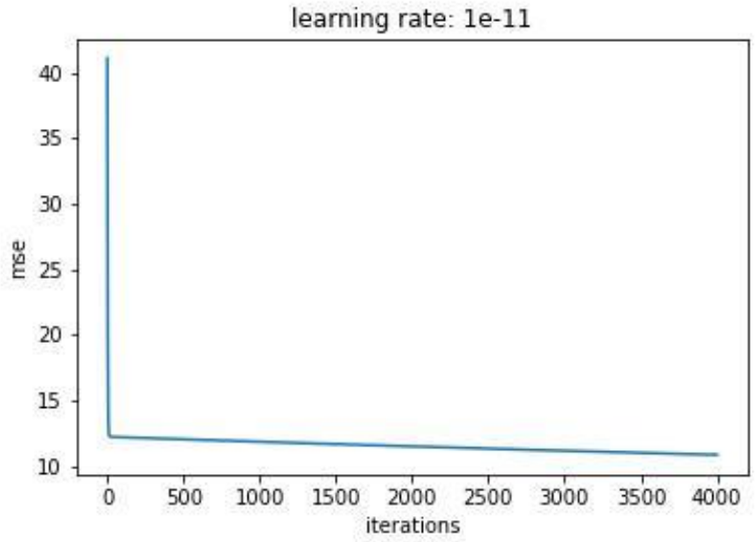

learning rate: 0.001
1e293
mse
iterations

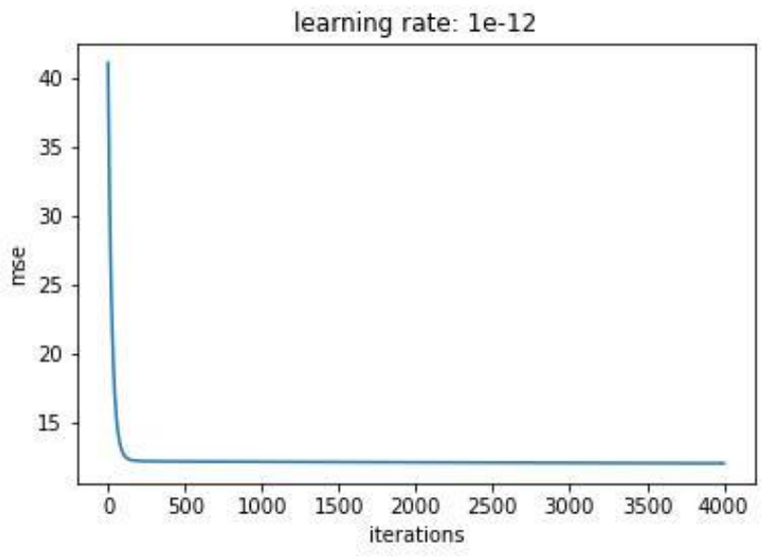

learning rate: 0.0001
1e304
mse
iterations

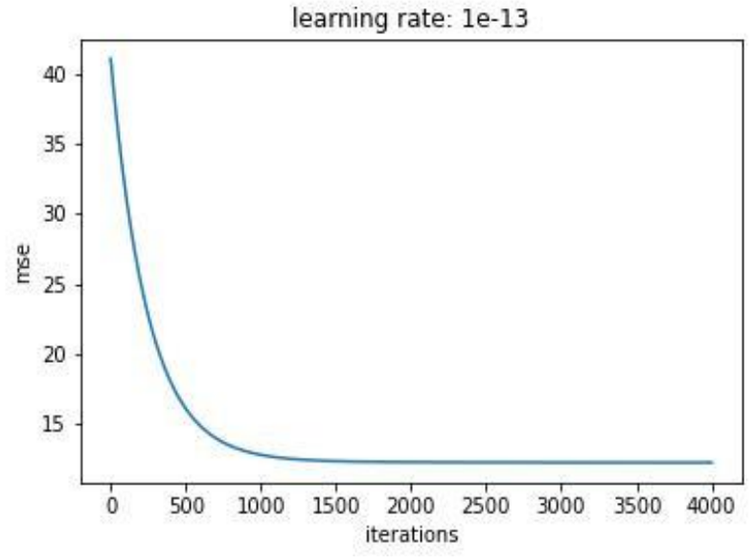

learning rate: 1e-05
1e298
mse
iterations

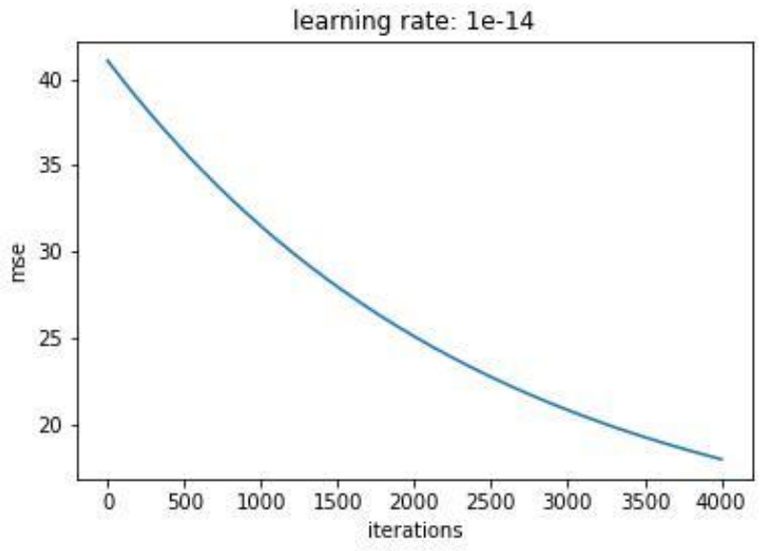

learning rate: 1e-06
1e302
mse
iterations

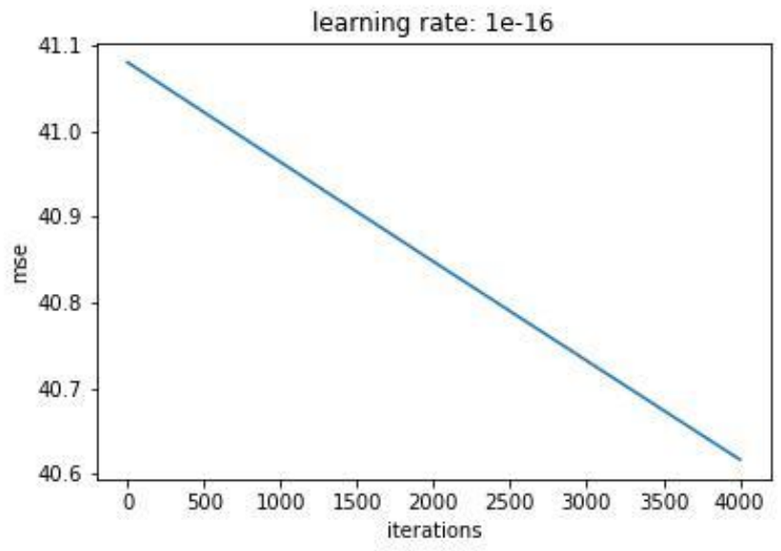

learning rate: 1e-10
1e100
mse
iterations


learning rate: 1e-11
mse
iterations


learning rate: 1e-12
mse
iterations


learning rate: 1e-13
mse
iterations





(b) The best learning rate found was 1e-12 since it has lowest MSE among the three models that converged as seen below.

```
itr=4000, lr=1e-11, loss=10.85034565905031
itr=4000, lr=1e-12, loss=12.071950445691549
itr=4000, lr=1e-13, loss=12.214437783245735
```

The training and validation MSE for the best learning rate of 1e-12 are listed below:

```
Training dataset, lr=1e-12, MSE=12.071950445691549
Validation dataset, lr=1e-12, MSE= 14.178753778371513
```

(c) The weights under the best learning rate of 10^-12 are listed below:

Feature	**Learned Weight**	**Feature**	**Learned Weight**
Dummy	5.352510e-10	grade	2.556549e-08
Month	3.640779e-09	sqft_above	1.477215e-05
Day	2.363169e-10	sqft_basement	3.587229e-06
Year	1.078074e-06	yr_built	1.087632e-06
Bedrooms	9.793863e-09	age_since_renovated	-4.478279e-08
Bathrooms	1.206408e-08	zipcode	5.245070e-05
Sqft_living	1.835938e-05	Lat	2.669476e-08
Sqft_lot	4.504619e-06	Long	-6.543665e-08
Floors	4.860859e-09	Sqft_living15	1.197049e-05
waterfront	4.500020e-10	Sqft_lot15	4.451010e-06
view	8.150734e-09		
condition	2.776913e-09		

Questions:

1. **Why do you think the learning rate needs to be smaller for the non-normalized data?**
   Non-normalized data requires the use of smaller learning rates to converge as non-normalized input features has the risk of exploding gradients. As some of the features have much larger values compared to others, the gradients for weights related to feature with large values will be significantly larger than the gradients for the other features during training and this can hinder the gradient descent algorithm from coverging to a minimal. This is where the use of small learning rates can help to ensure convergence on the non-normalized features of our dataset.

2. **Compare between using the normalized and the non-normalized versions of the data, which one is easier to train and why?**
   Comparing both versions of the trained data, the normalized version was easier to train. One reason may be because after normalization, the data now resembles a gaussian distribution compared to the un-normalized data, with a lot of skewed features.

3. **Compare the learned weights to those of 1(c), what key differences do you observe? How do you think this impacts the validity of using weights as the measure of feature importance**.
   Comparing to the weights to part 1(c), the learned weights here are very small. The explanation for this phenomenon could be the magnitude imbalance of values for different features. There are very large values like features "sqft_lot", versus small values like features "bathrooms". When updating weights during training, the large values will contribute much more to the weight update than features with smaller values, so the weights of these large features must be very small to counteract the domination of large values, trying to balance contributions from small values. Therefore, as a measure of feature importance, learned weights alone cannot be used to indicate the importance of a feature.

(b) I used the best model with learning rate of 0.1 which we have previously obtained. The validation MSE using this learning rate is 4.52114487957 6144. The learned weights for each feature are reported as follows:

Feature	**Learned Weight**	**Feature**	**Learned Weight**
Dummy	5.335026	grade	1.157613
Month	0.055064	sqft_above	0.797647
Day	-0.050384	sqft_basement	0.160526
Year	0.172789	yr_built	-0.874220
Bedrooms	-0.283117	age_since_renovated	-0.090588
Bathrooms	0.333352	zipcode	-0.272844
Sqft_living	0.803578	Lat	0.841366
Sqft_lot	0.051960	Long	-0.286782
Floors	0.004418	Sqft_lot15	-0.095519
waterfront	3.947974		
view	0.462525		
condition	0.195855		

Questions:

1. **How does this new model compare to the one in part 1(c)?**
   The new model performs almost the same as the one in part 1(c), since the MSE on validation set is almost the same.

2. **What do you observe when comparing the weight for sqft_living in both versions?**
   The weight of feature “sqft_living” changes from 0.7 to 0.8 from the model in part 1(c) to the current model without feature “sqft_living15”, which increases just a little bit.

3. **In the situation when two features x1 and x2 are redundant, what do you expect to happen to the weights (*w1 and w2*) when learning with both features, in comparison with w1 which is learned with just x1? How does this phenomenon influence the interpretation of feature importance?**
   When two features x1 and x2 are redundant, I expect the weights (w1 and w2) learned by both features to be very similar (same) or slightly smaller than w1 learned with just x1. The reason is that the two features are redundant, so the information that one feature carries is basically very similar to the other feature. Then the weight learned by both features or single feature should be very similar. I think the slightly higher weight learned by only one feature x1 is caused by removing the other feature x2. Due to this removal of x2, even though x2 is redundant, it still carries slightly different information compared to x1. So the weight of x1 will be slightly larger as a kind of compensation to weight x2.