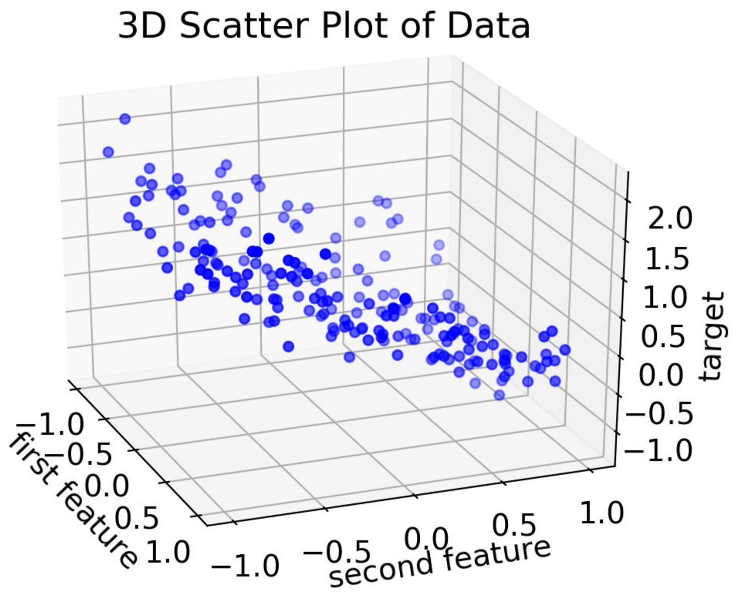
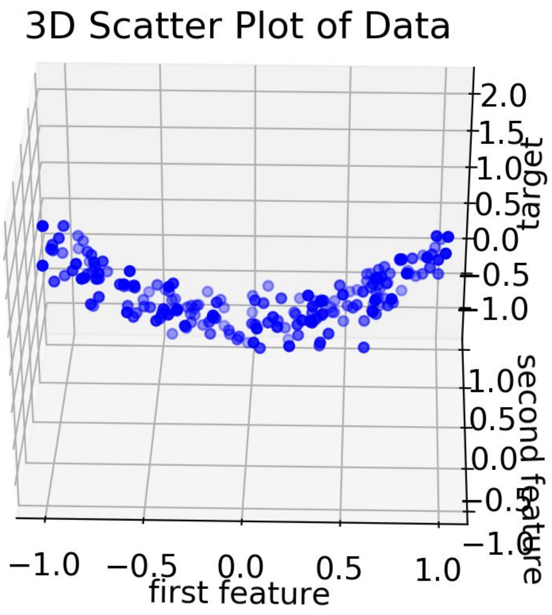
Machine Learning | Week 3 Assignment
17324263 | Stephen Byrne

(i) (a)





To plot the graph I used the code given in the assignment *(Appendix i.a).* The X features are represented on the X and Y axis respectively while the target data is on the z-axis. I scattered the data to show it and will now explain the results.

As we can see by plotting the above graph, there is a linear relationship between the second feature and the target but a more quadratic relationship between the first feature and the target. It looks like the data lies on a slight quadratic curve that linearly increases along a plane.

(b)
To add the squared features I used the PolynomialFeatures function from sklearn.preprocessing that was given in the assignment *(Appendix i.b.1)*. This performs a binomial expansion of the features and I did it per the requested number of additional features. This was then used throughout the assignment for both ridge and lasso regression using the .fit_transform function when fitting to the models *(Appendix i.b.2)*.

Lasso regression adds an absolute value of the magnitude of coefficient as a penalty term to the loss function. As this is an absolute value the key difference between the L1 regularisation of Lasso Regression and the L2 of Ridge Regression is that Lasso will be able to shrink the less important features coefficients to zero, removing some of them altogether. This allows for feature selection with a large number of features, such as the additional polynomial features we have added. As we can see below for the different values of C we get a different number of final parameters for our model because of this. The coefficients below are the parameter values of the model.

To actually generate these values I used a loop which iterates over an array of C values and generates a model for each, prints their parameters, and graphs them. This can be seen at *(Appendix i.b.3)*

Lasso Regression Model where C=1

```
Coef :  [ 0. -0. -0.  0. -0. -0. -0. -0. -0. -0.  0. -0.  0. -0. -0. -0. -0. -0. -0. -0. -0. ]
Intercept :  0.3384288912637509
```

When C=1 the C value is small enough so that all the parameters are 0. This would be equivalent to plotting a plat plane with an intercept at 0.33842… on the target axis.

Lasso Regression Model where C=2

```
Coef : [ 0.  -0.  -0.30972678.  0.  -0.  0.  -0.  -0.  0.  -0.   0.  -0.  0.  -0.  -0.  -0.  -0.  0.
-0. 0. -0. ]
Intercept: 0.3466033259543637
```

When we increase C to 2 we can see a non zero parameter is added as we change the penalty and more non-zero features are added. This would essentially make our model linear and as we are plotting it in 3 dimensions we end up predicting a plane that intercepts the target axis at 0.3466

Lasso Regression Model where C=10

```
 Coef :  [ 0. -0.  -0.89285922.  0.47049658 -0.  0.  -0.  -0.  -0.  -0.  0.  -0.  0.  -0.  0.
-0.  -0.  -0.  -0.  -0.  -0. ]
Intercept :  0.1991350732180508
```

When we reach C=10 we can see we have two parameter values that are non zero. This leads us to plot a quadratic plane with an intercept at 0.199. As our data does seem to have a quadratic shape it is likely this will give an accurate prediction.

Lasso Regression Model where C=1000

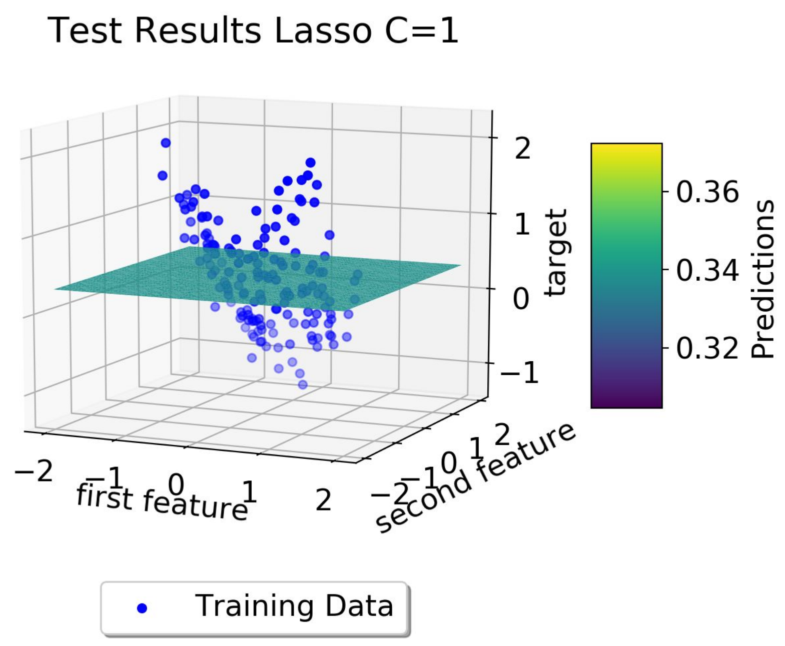
```
Coef : [ 0.  -0.04450203.  -1.05138283.  1.02262127.  0.04119445.  0.06074779
0.16023018.  -0.  -0.13078224.  0.  0.02927762.   -0.06342224.   -0.   -0.  -0.
0.01079366.  0.06103734  0.  -0.04101274.  -0.  -0.]
Intercept :  -0.010701867638647888
```

With C=1000 we have a lot of non zero parameters. It is likely this model is overfitted as our training data visually looked like it increased linearly with some quadratic element to it, while this model is clearly not quadratic in nature.

(c)

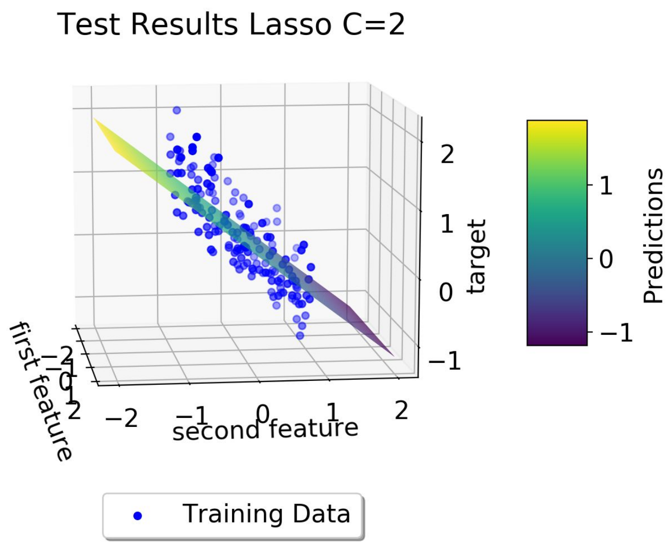
To graph the values for the predictions, it happens within the previously explained function which can be seen at *(Appendix i.b.3).* The graphing function can be seen at *(Appendix i.c.1).* This function creates a test space *(Appendix i.c.2)* which was given in the assignment. The test space is used to plot a surface which may relate to a polynomial function or other plane. The test space is used in the graph and plotted against the predicted target values from the model. The graphing function then plots this surface for the predicted values along with the training data as requested in the question.

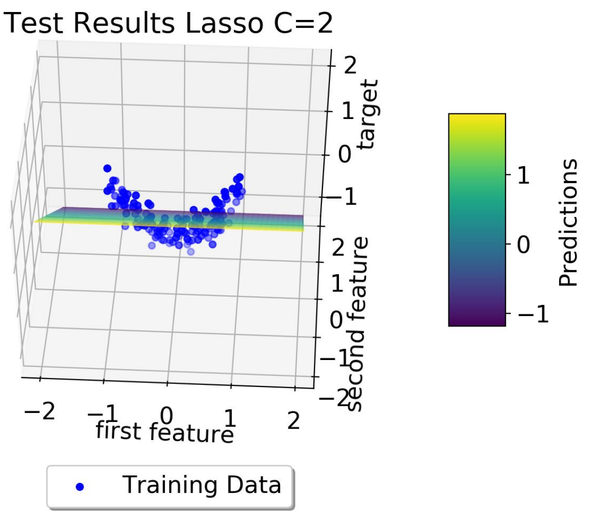
Lasso Regression Model where C=1



As we can see when C=1 the model parameters are all 0 as mentioned above so we get a flat plane with an intercept at 0.338.. This does not fit our data very well and would be similar to predicting the mean value of the target.

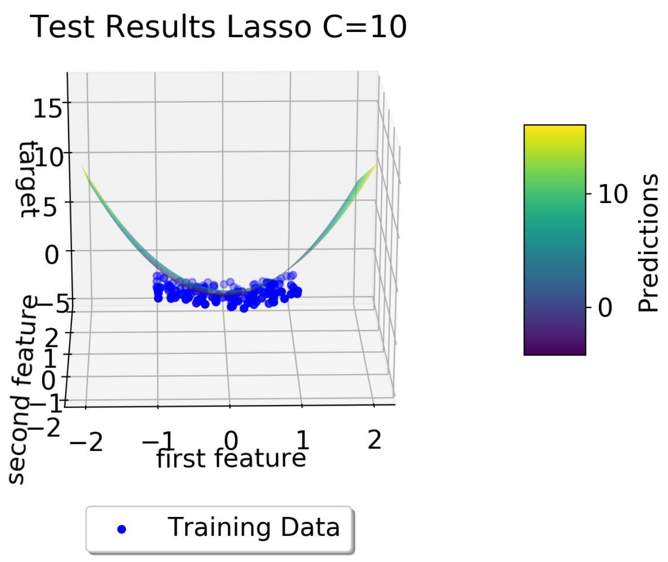
Lasso Regression where C=2

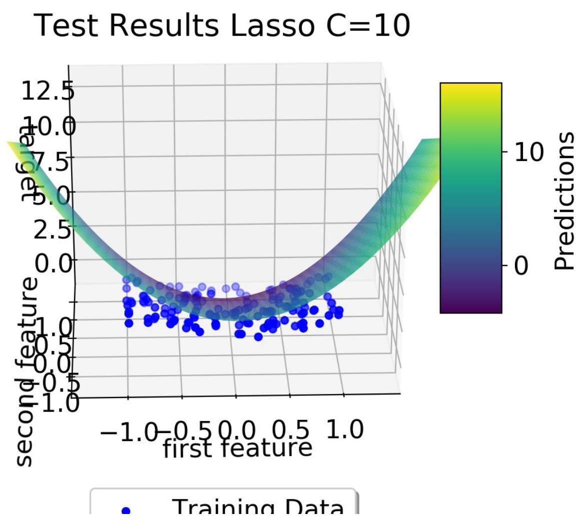




As we can see above when C=2 we get a plane that linearly follows our data quite well. This can be seen on the left. However, when looking at the predicted data plane from another angle, we can see it does not capture the quadratic nature of the data. The parameter would represent the slope along with the intercept in the equation of a line y=mx+c

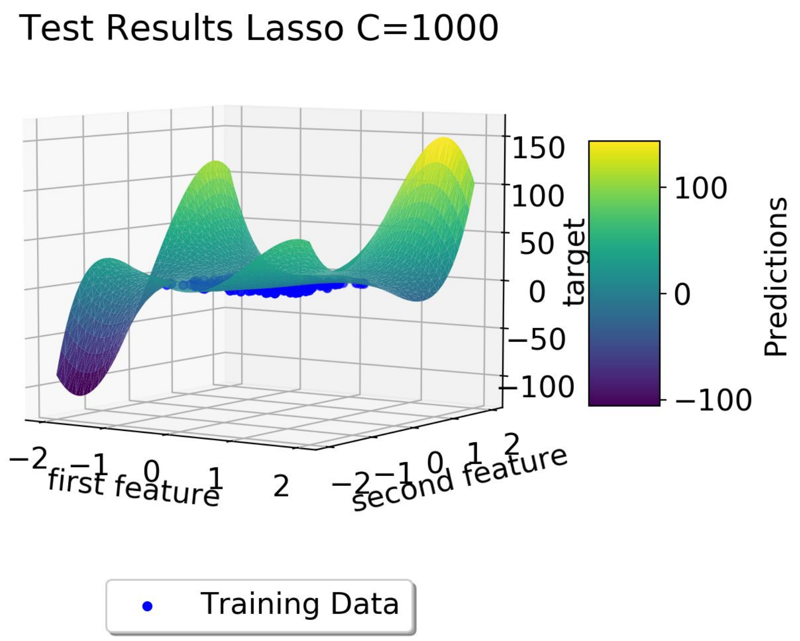
Lasso Regression Model where C=10





As we can see on the left and right, the quadratic nature of the curve is taken into account when training the model for this penalty, however, it does seem to be quite a steeply sloping curve so may not be perfect, however, it does a good job. As we can see in the coefficients above in (b) we have two non-zero which would be quadratic.

Lasso Regression Model where C=1000



As we can see, the ridge model is obviously overfitted. We know this as the data had a linear/quadratic shape to it while the predictions for C=1000 leave us with many non-zero coefficients which gives a polynomial shape to the predictions, this is due to more noise being fitted.

(d)
Underfitting:
Underfitting occurs when our model is so simple that it fails to capture the behavior of the data, such as fitting a straight line (or plane in our case) to quadratic natured data. This occurs when C=1, as can be seen in (b) all the parameter values are 0 and we simply have an intercept meaning we get a flat plane at the intercept which is nowhere near what our data looks like. This is as all the features have been reduced through the L1 Regularisation to 0. It can be visually seen in (c) on the left. When we move to C=2 we end up in a more interesting situation. As can be seen in (b) where C=2, we have one parameter value which corresponds to the slope. The plane we fit to predict the data now has a slope and matches the general direction of our data quite nicely, however, when we turn the 3D plot (visualized on the right where C=2) we can see it is not matching the quadratic nature of our data at all. This would lead to the conclusion that the model is still under-fit.

Overfitting:
Overfitting occurs as we add more parameters to the model, we start to fit the "noise" This is what happens with the model where C=1000, at points where we don't have training data the predictions of the model get really bad. So the model for C=1000 generalizes poorly. When we look at the parameters for this model in (b) there are many features that are non zero, however, it is clear that the data we are trying to fit is quadratic in nature. This can be visually seen in (c) where we have a visual polynomial plane trying to predict our target values, which is overfit as the data is visually quadratic in nature. When we use Lasso Regression, by modifying the C value we can shrink less important features leaving only the parameters that best fit the model. In this case, C=10 performs relatively accurately however without knowing the underlying nature of the data we could not tell for certain.

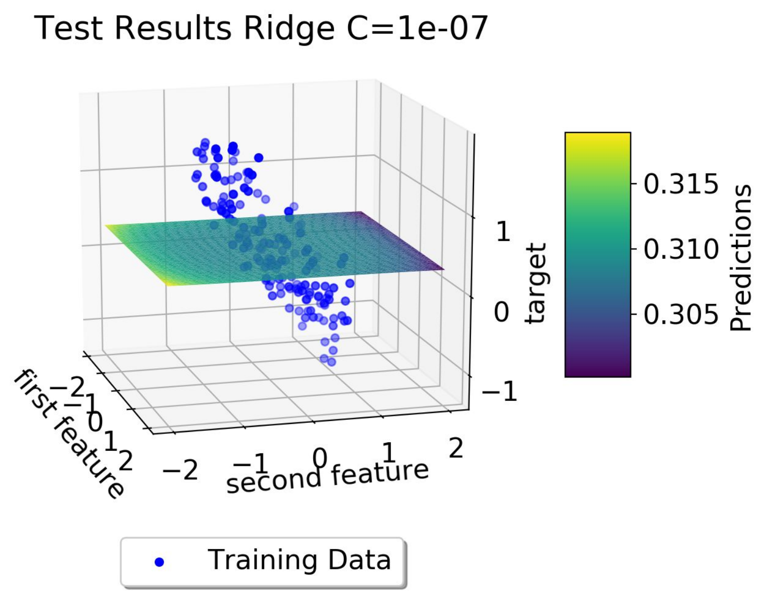
(e) - repeat (b) & (c) for Ridge Regression

I initially used the same C values as with Lasso Regression because to compare the effect of changing them it would make sense to compare the same values. Ridge Regression works using L2 Regularisation which uses the square magnitude of the coefficients. This means that no coefficients are eliminated as they are all shrunk by the same factor. This means we end up fitting polynomial models to our quadratic data below which leads to numerous overfit models. I also tried drastically reducing the size of C to reduce the model parameters and then try to slowly increase them as done in part (c). I was able to reduce the model parameters and plot a flat plane (as shown below) however when it came to increasing the parameter size the model quickly became overfit as it is fitting polynomial features to a quadratic model. To plot this I use essentially the same function as *(Appendix i.c.1)* so I will not explain it again, refer to previous explanation. The only difference changed is the model. It can be seen at *(Appendix i.e.1)*

Ridge Regression Model where C=0.00000001

```
Coef :  [ 0.00000000e+00 -5.46539190e-07 -1.07913369e-05  2.71061039e-06
 -4.41936777e-07 -2.16316820e-07 -3.16107467e-07 -3.35587209e-06
  5.28251855e-08 -6.69516321e-06  2.36986266e-06 -5.29193364e-07
  7.54212571e-07 -4.41768527e-07 -2.86497713e-07 -1.73260107e-07
 -1.88666730e-06  2.46104561e-07 -1.98023783e-06  1.18133046e-07
 -4.97284386e-06]
Intercept :  0.3086442691343684
```

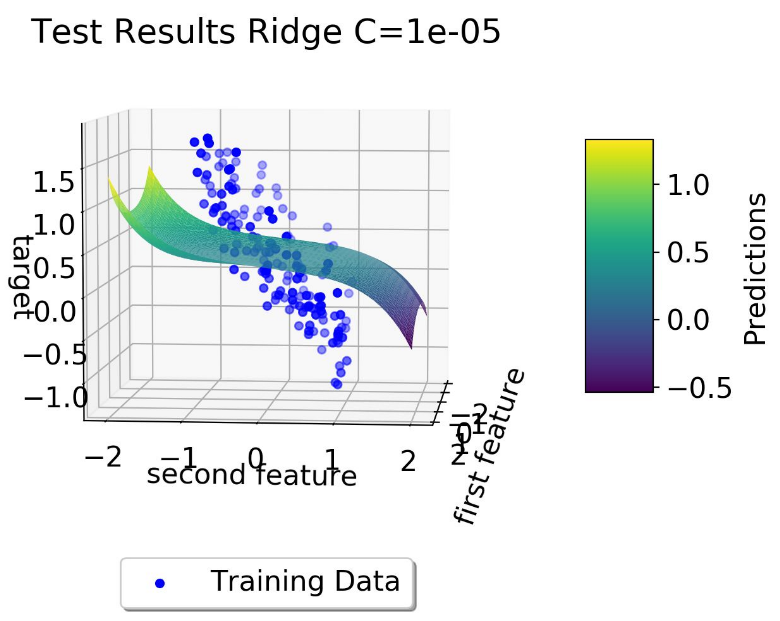
As we can see above all the values are so small they are approaching 0, therefore when plotted (below) we end with a flat plane, which cuts the target axis at the intercept.



Ridge Regression Model where C=0.000001

```
Coef :  [ 0.00000000e+00 -5.45000792e-05 -1.07709350e-03  2.70977344e-04
 -4.40547485e-05 -2.14956436e-05 -3.15418665e-05 -3.34846989e-04
  5.29843073e-06 -6.68139740e-04  2.36928371e-04 -5.28082005e-05
  7.54721367e-05 -4.40694262e-05 -2.85113133e-05 -1.72926721e-05
 -1.88206780e-04  2.45764400e-05 -1.97531261e-04  1.18035172e-05
 -4.96215764e-04]
Intercept :  0.30859613599607105
```

Parameter values here are still very small, however, when plotted they still lead to an overfit model that generalizes poorly outside of our training data, as do the rest of the ridge regression models.

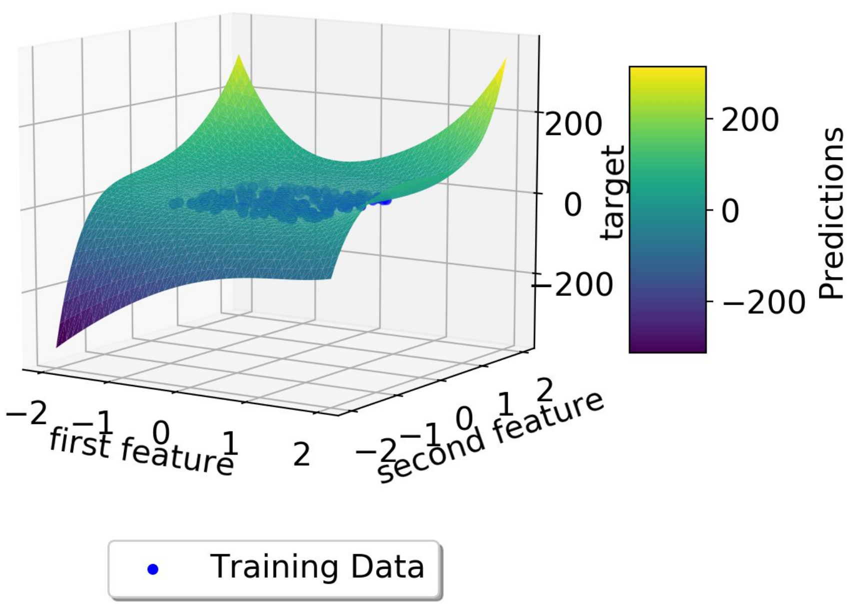


Ridge Regression Model where C=1

```
Coef :  [ 0.   -0.01251227 -0.98855674  0.75623807  0.12284561  0.20700655
  0.06869271 -0.14966456 -0.1410586  -0.09474584  0.24004226 -0.14610606
  0.08301728 -0.06083658 -0.18726659  0.01389276  0.03674076 -0.04179784
  0.13639714  0.10540867  0.10298927]
Intercept :  0.007718646265874984
```

When we get to C=1 (and for the rest of the graphs) they are all overfitted due to the high parameter values on polynomial features attempting to fit quadratic data. I won't repeat the same comment for the next 4 graphs however it is the same situation for all and progressively as the parameters get bigger they become more overfit and generalize worse.

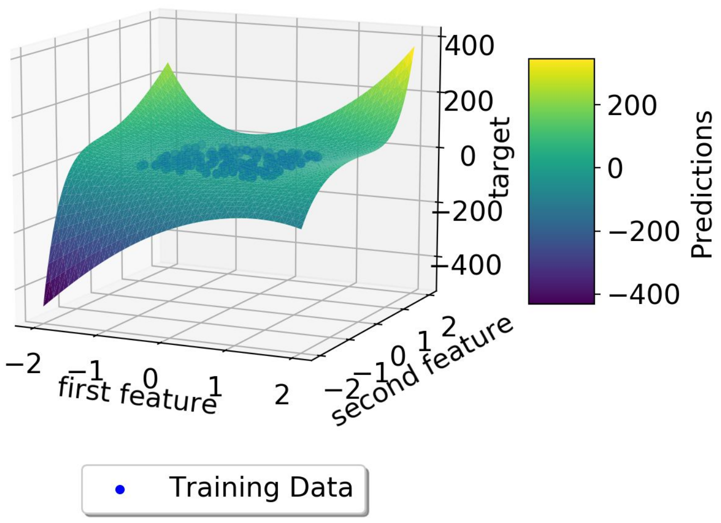
Test Results Ridge C=1



Ridge Regression Model where C=2

```
Coef :  [ 0.         -0.01280719 -1.0155661   0.83820458  0.14114793  0.27555564
  0.10110981 -0.16317908 -0.21015405 -0.06009222  0.17796168 -0.16219314
  0.04694569 -0.07957452 -0.24955862 -0.01048017  0.04404397 -0.04620445
  0.16747022  0.18193831  0.0971394 ]
Intercept :  -0.013344854453158583
```

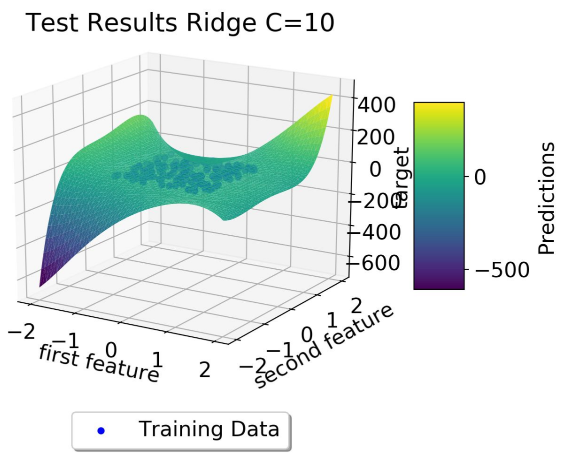
Test Results Ridge C=2



(Explanation same as previous model)

Ridge Regression Model where C=10

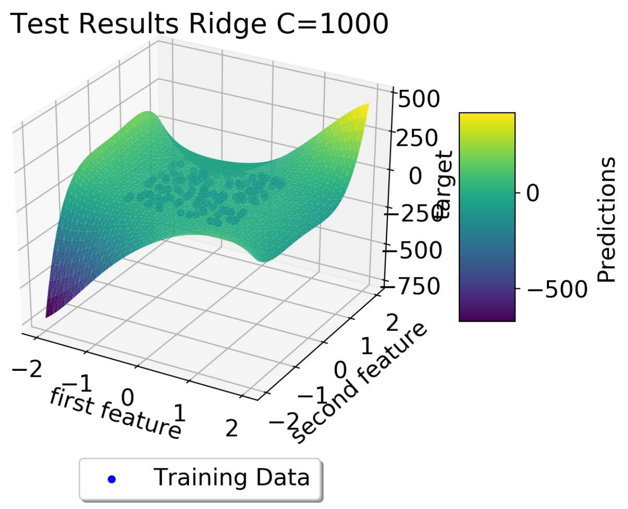
```
Coef :  [ 0.         -0.01159719 -1.05045221  0.94580714  0.15889463  0.37715781
  0.18338358 -0.18809183 -0.3853054   0.02732849  0.09529962 -0.17663778
 -0.00436187 -0.10330787 -0.34421705 -0.08237628  0.03121131 -0.03068217
  0.2440984   0.35917192  0.03573025]
Intercept :  -0.0415922520655036
```



(Explanation same as previous model)

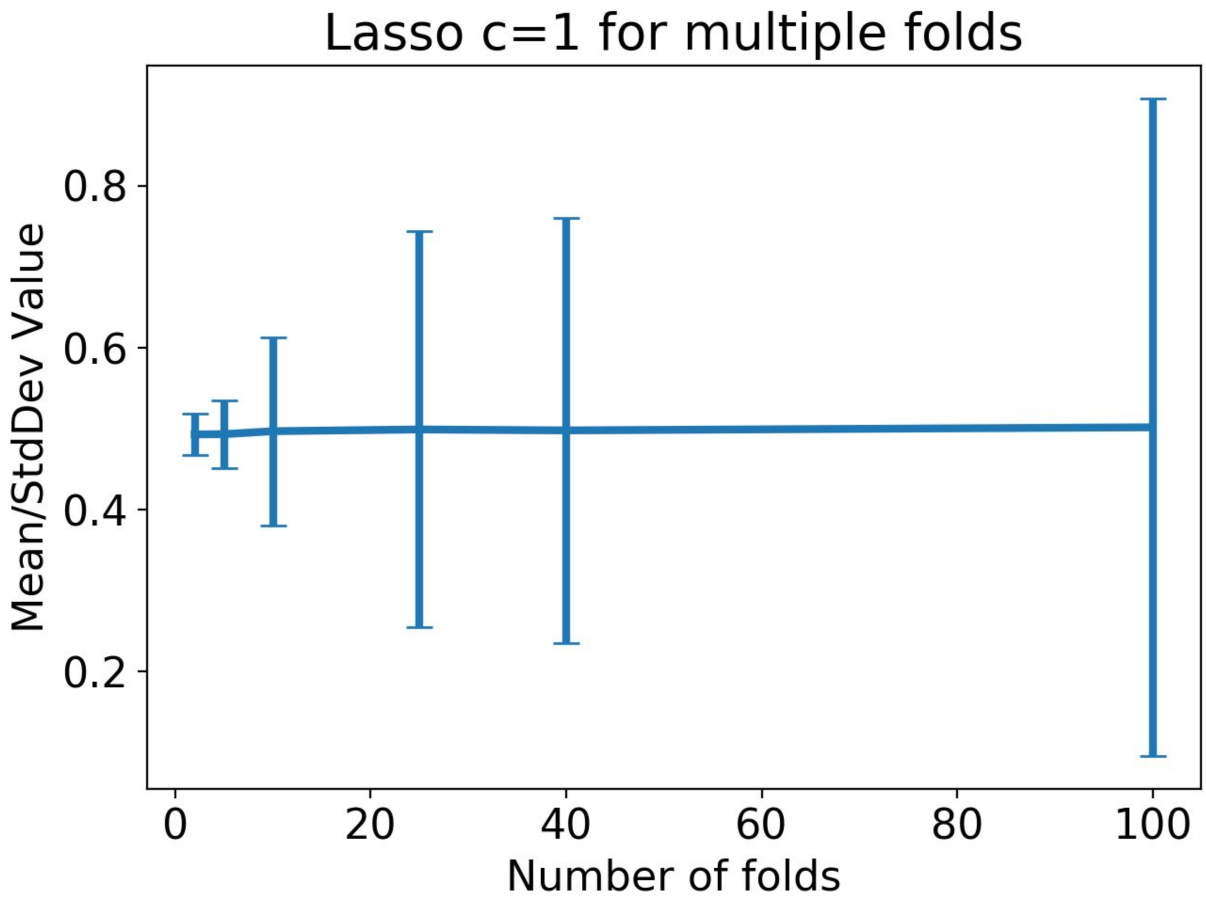
Ridge Regression Model where C=1000

```
Coef :  [ 0.         -0.01054479 -1.06991434  0.98334295  0.16429497  0.41651004
  0.23768639 -0.19500698 -0.49825569  0.09736936  0.06734745 -0.18092901
 -0.023557   -0.11251818 -0.38211312 -0.13244216  0.00889529 -0.01109782
  0.29218118  0.46619834 -0.02531341]
Intercept :  -0.05185127921689098
```



(Explanation same as previous model)

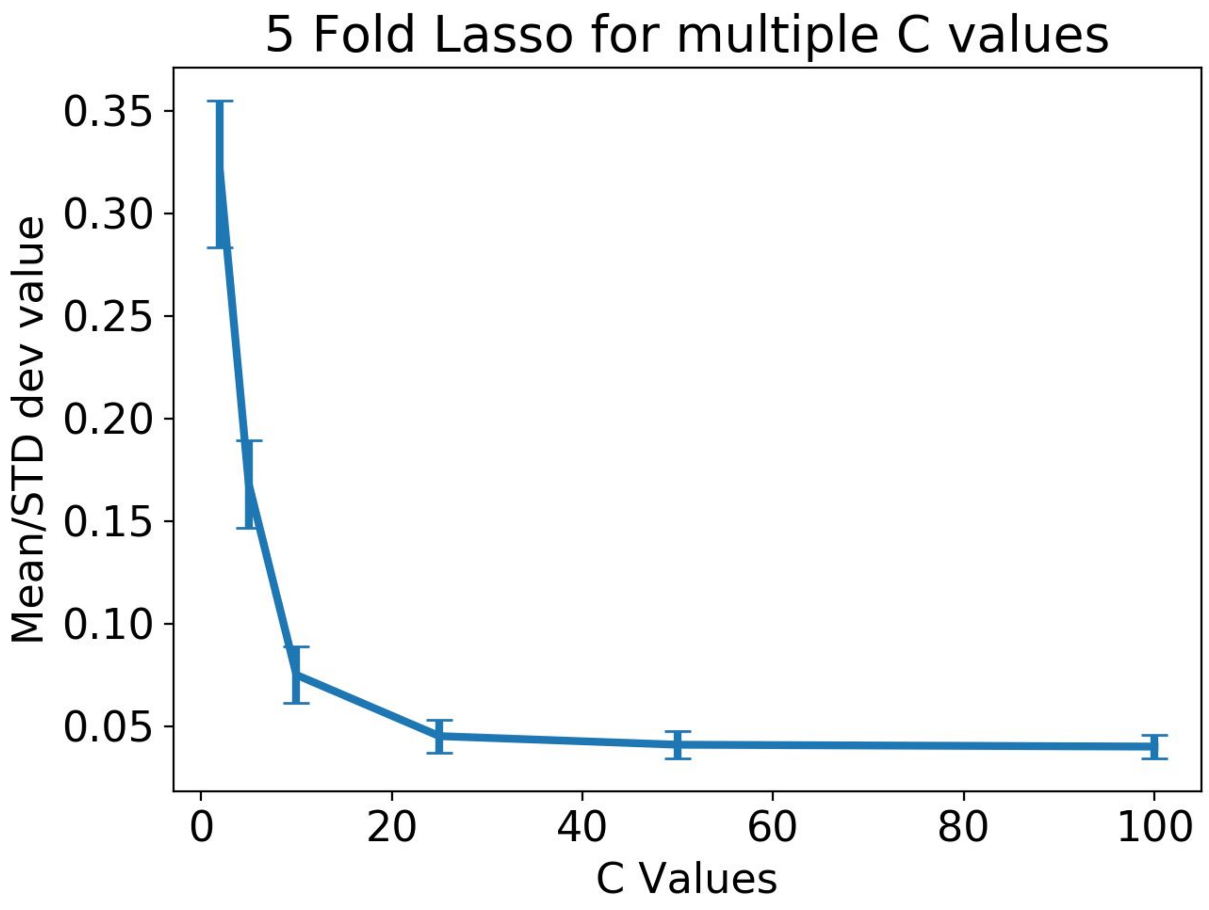
(ii) (a)



Explanation of graphing code seen at *(Appendix ii.a)*: To graph this I used a loop that iterates over multiple fold values and for each of them calculates the mean prediction error estimates using the same model for C=1. I then calculated the square root of the variances to get the standard deviations and added this and the means to my plot.

If we use too many folds we also start to model more noise as there may be a bigger spread in the smaller data sets. This means as we move from one set to another for large folds (eg 20, 40, or 100 above) we get big fluctuations leading to a big variance. When we go with a small number of folds we get a small MSE mean however we don’t get as many good estimates for the model. There is also a trade-off with computational time to complete a very large number of folds. As we can see from above, 5 folds doesn't give a massive variance however also provides enough estimates for our model and isn’t too computationally taxing so I would pick 5 folds. It is worth noting that the mean doesn’t change much either while the variances fluctuates wildly for larger numbers of folds due to the spread in the data.

(b)



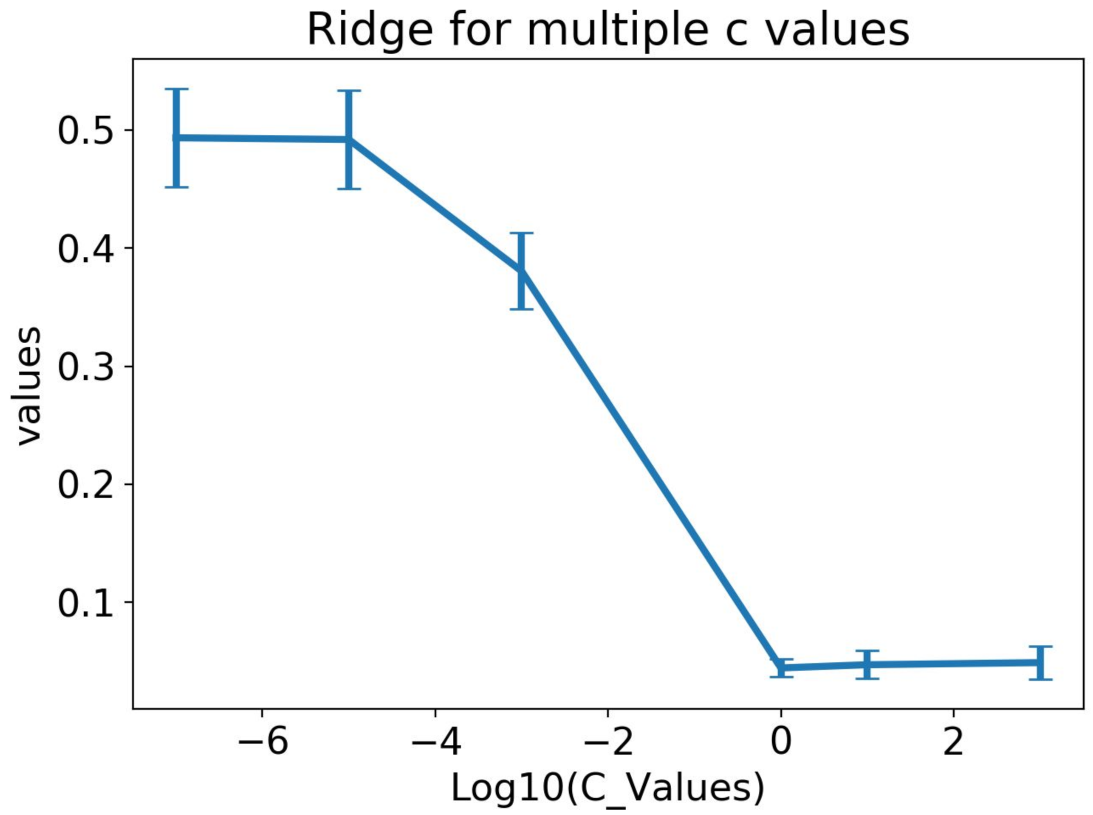
Explanation of code seen at *(Appendix ii.b)* : To plot the C values I iterated over an array similarly to the previous question, but for C values not folds. I then trained a model for each C value and used kFold for 5 splits to get the prediction estimates for each model. The range of C values chosen was based off of the results I got for the graphs in part (c). I was aware larger values (above say 40) would give overfit models yet they also give good results for the mean and std deviation. Working backwards from this I tried some smaller C values however they gave very high mean and variances for the mean and standard deviation of the prediction errors. Judging off the good fit from C=10 in (C) the optimal value will be around this. This can be seen above as the mean and standard deviation decrease to very low amounts from there. The goal would be to pick the simplest mode (closest to the smallest C value here) that also has a low mean and std deviation of the prediction error. A good model here would be somewhere between 10 to 25.

(c)

Explain what c value to use and reason?

Based on the cross-validation data, the value of c I would recommend using would be between 10 to 25. As we want to avoid overfitting and introducing more parameters, we want to use the simplest model that also produces a reasonable value. As we can see in (b) this would be in the range of 10 to 25.

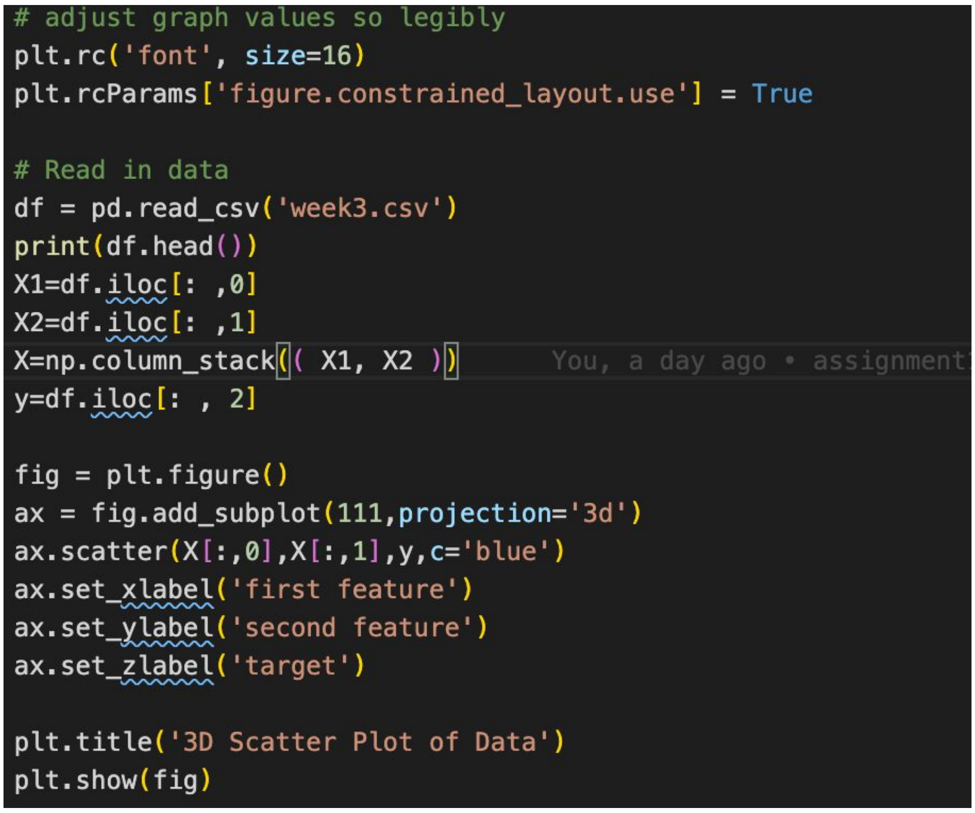
(d)



This question was completed the same as the question seen at (ii b) except using a ridge regression model and the loop again altering the C values, however they are different values for this model *(Appendix ii.d)* The values chosen for C with ridge were also based on the earlier results in the graph. As we are using very small values (1x10e-7 and 1x10e-5) I plotted the values on a logarithmic scale to space them out to make them more distinguishable. As we can see the values for 1x10e-7 and 1x10e-5 have quite large values for mean and standard deviation of the prediction error. This is due to the fact both essentially represent a flat plane around our training data, although 1x10e-5 is a poorer generalisation as we move to the edges of the data. As the ridge regression fits all the parameters we try to fit a polynomial to quadratic data which leads to overfit models once we approach the higher values (c=1 and above, values after 0 here). Every model for this will have a poor generalization when moving outside the training data once C increases so there is no best C value, only ones that perform well within the ranges of the training data.

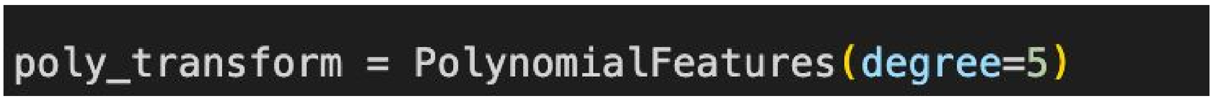
Appendix

(i.a)



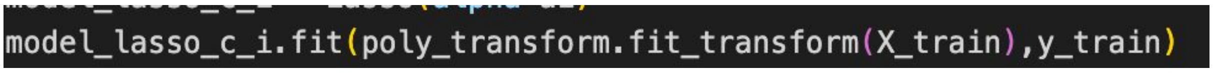
```
# adjust graph values so legibly
plt.rc('font', size=16)
plt.rcParams['figure.constrained_layout.use'] = True

# Read in data
df = pd.read_csv('week3.csv')
print(df.head())
X1=df.iloc[: ,0]
X2=df.iloc[: ,1]
X=np.column_stack(( X1, X2 ))        You, a day ago • assignment
y=df.iloc[: , 2]

fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
ax.scatter(X[:,0],X[:,1],y,c='blue')
ax.set_xlabel('first feature')
ax.set_ylabel('second feature')
ax.set_zlabel('target')

plt.title('3D Scatter Plot of Data')
plt.show(fig)
```

(i.b.1)



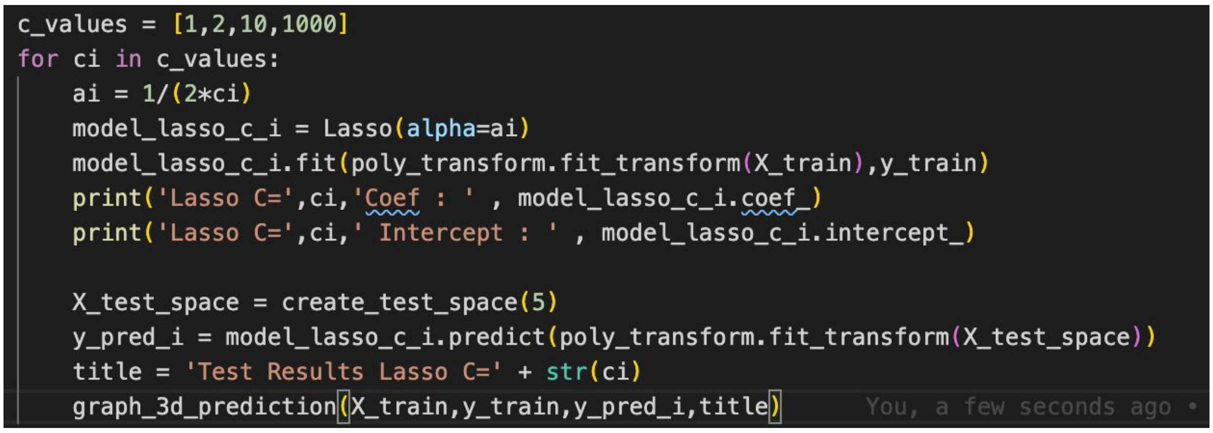
```
poly_transform = PolynomialFeatures(degree=5)
```

(i.b.2)



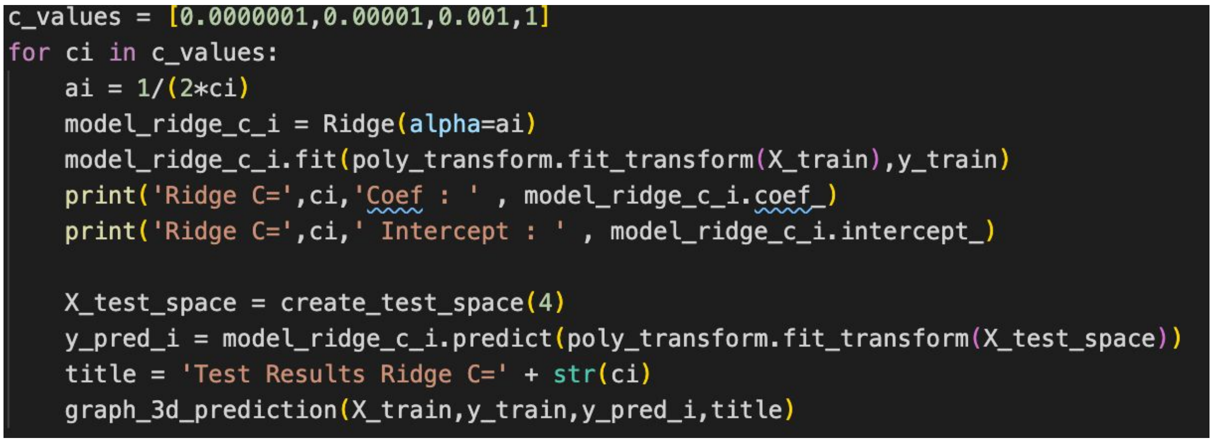
```
model_lasso_c_i.fit(poly_transform.fit_transform(X_train),y_train)
```

(i.b.3)



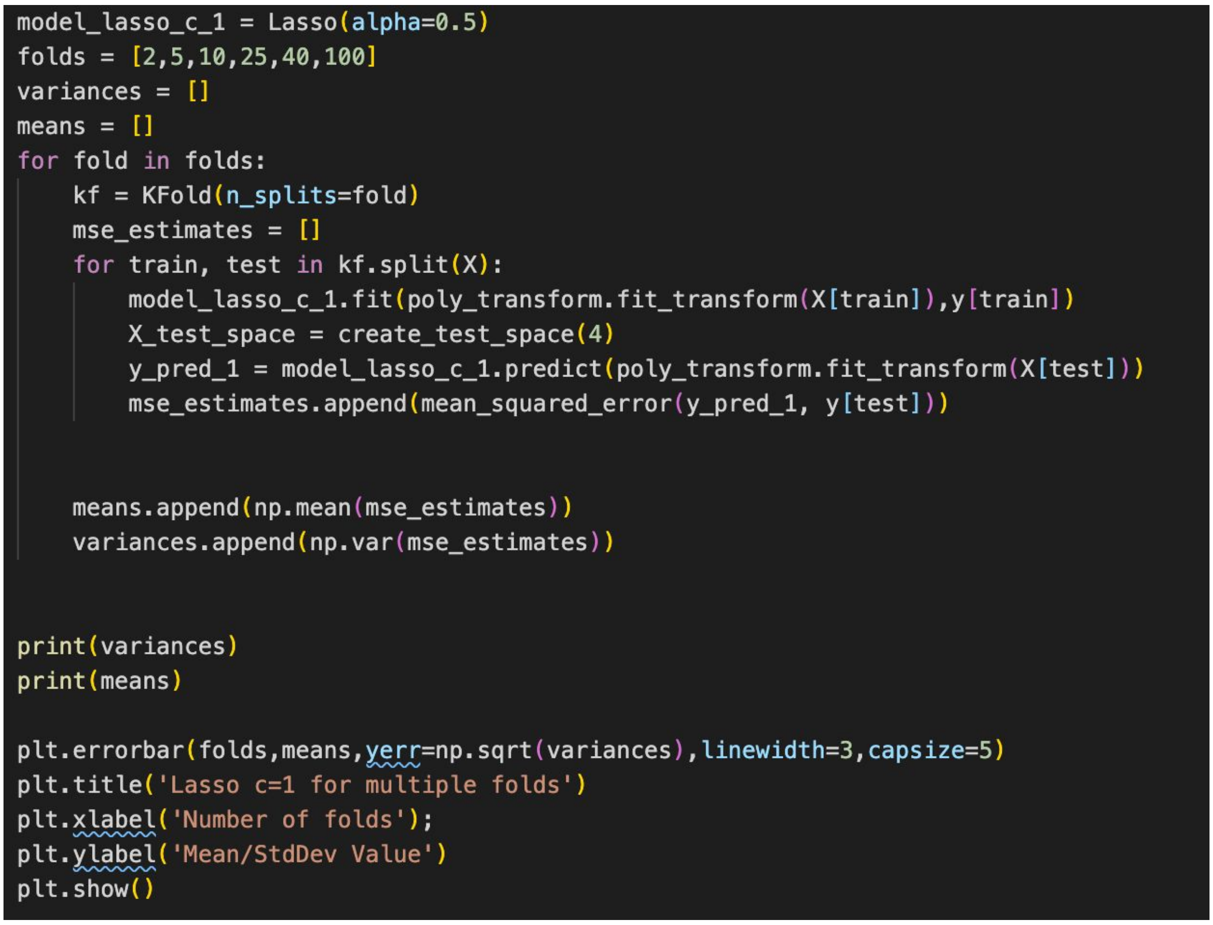
```
c_values = [1,2,10,1000]
for ci in c_values:
    ai = 1/(2*ci)
    model_lasso_c_i = Lasso(alpha=ai)
    model_lasso_c_i.fit(poly_transform.fit_transform(X_train),y_train)
    print('Lasso C=',ci,'Coef : ' , model_lasso_c_i.coef_)
    print('Lasso C=',ci,' Intercept : ' , model_lasso_c_i.intercept_)

    X_test_space = create_test_space(5)
    y_pred_i = model_lasso_c_i.predict(poly_transform.fit_transform(X_test_space))
    title = 'Test Results Lasso C=' + str(ci)
    graph_3d_prediction(X_train,y_train,y_pred_i,title)        You, a few seconds ago •
```

(i.e.1)



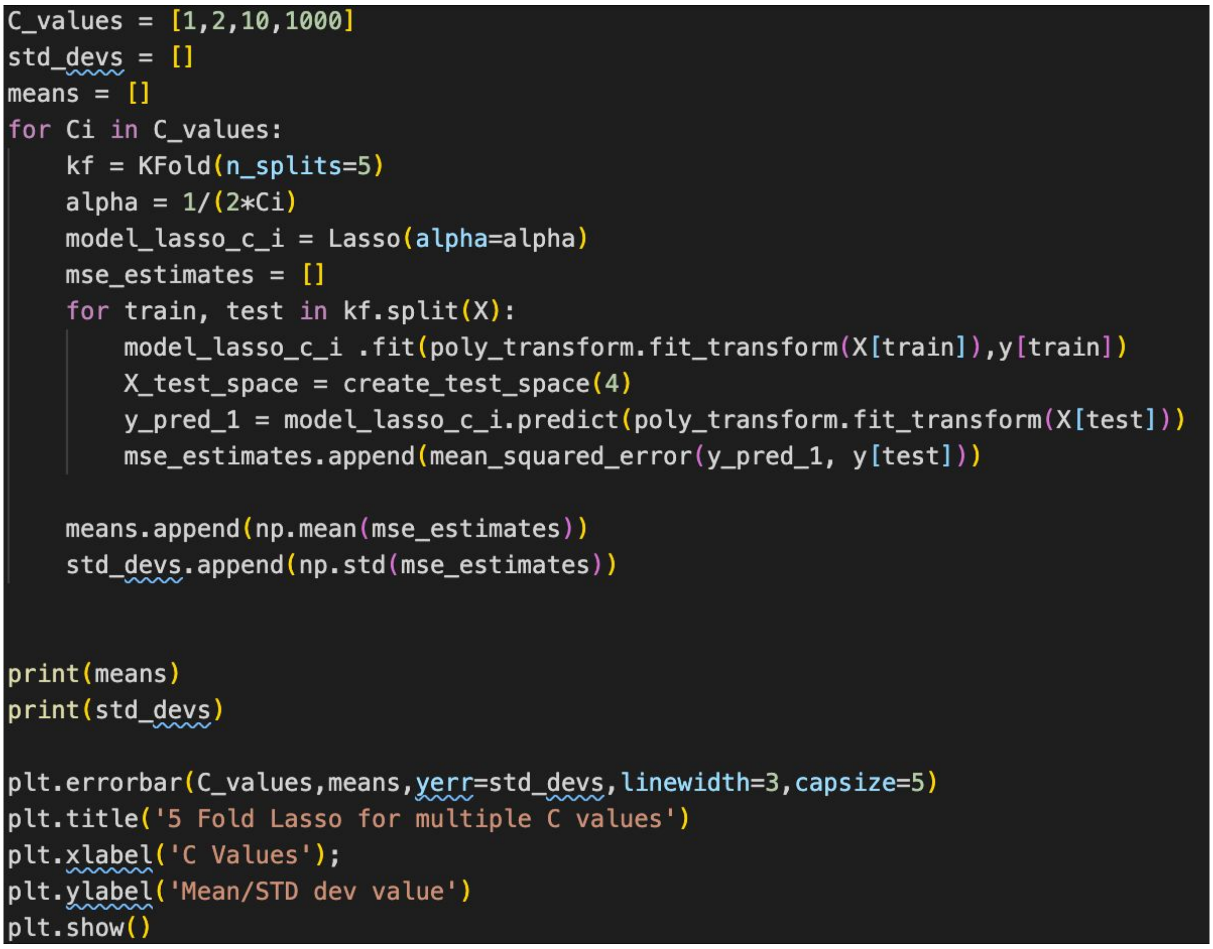
```
c_values = [0.0000001,0.00001,0.001,1]
for ci in c_values:
    ai = 1/(2*ci)
    model_ridge_c_i = Ridge(alpha=ai)
    model_ridge_c_i.fit(poly_transform.fit_transform(X_train),y_train)
    print('Ridge C=',ci,'Coef : ' , model_ridge_c_i.coef_)
    print('Ridge C=',ci,' Intercept : ' , model_ridge_c_i.intercept_)

    X_test_space = create_test_space(4)
    y_pred_i = model_ridge_c_i.predict(poly_transform.fit_transform(X_test_space))
    title = 'Test Results Ridge C=' + str(ci)
    graph_3d_prediction(X_train,y_train,y_pred_i,title)
```

(ii.a)



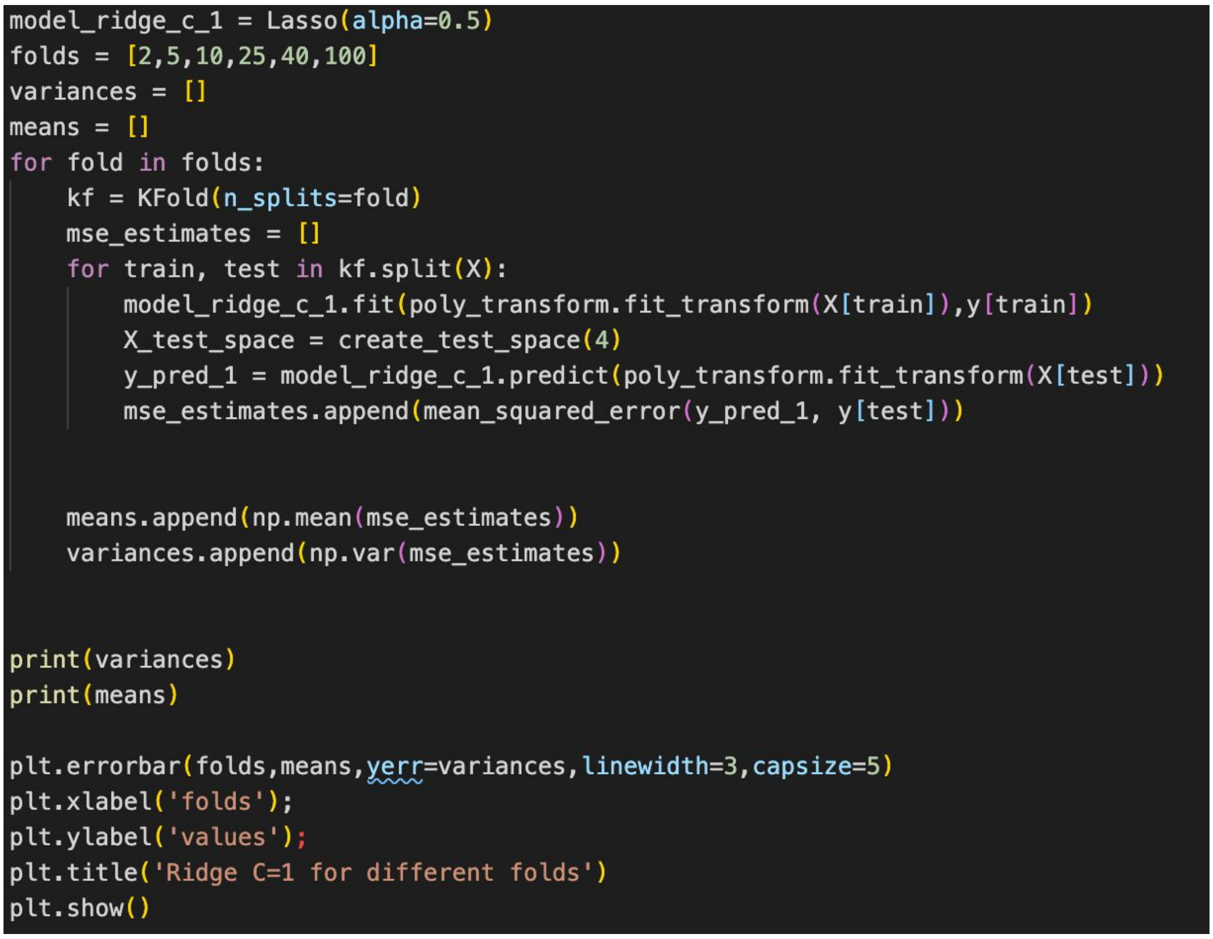
```
model_lasso_c_1 = Lasso(alpha=0.5)
folds = [2,5,10,25,40,100]
variances = []
means = []
for fold in folds:
    kf = KFold(n_splits=fold)
    mse_estimates = []
    for train, test in kf.split(X):
        model_lasso_c_1.fit(poly_transform.fit_transform(X[train]),y[train])
        X_test_space = create_test_space(4)
        y_pred_1 = model_lasso_c_1.predict(poly_transform.fit_transform(X[test]))
        mse_estimates.append(mean_squared_error(y_pred_1, y[test]))


    means.append(np.mean(mse_estimates))
    variances.append(np.var(mse_estimates))


print(variances)
print(means)

plt.errorbar(folds,means,yerr=np.sqrt(variances),linewidth=3,capsize=5)
plt.title('Lasso c=1 for multiple folds')
plt.xlabel('Number of folds');
plt.ylabel('Mean/StdDev Value')
plt.show()
```

(ii.b)



```
C_values = [1,2,10,1000]
std_devs = []
means = []
for Ci in C_values:
    kf = KFold(n_splits=5)
    alpha = 1/(2*Ci)
    model_lasso_c_i = Lasso(alpha=alpha)
    mse_estimates = []
    for train, test in kf.split(X):
        model_lasso_c_i .fit(poly_transform.fit_transform(X[train]),y[train])
        X_test_space = create_test_space(4)
        y_pred_1 = model_lasso_c_i.predict(poly_transform.fit_transform(X[test]))
        mse_estimates.append(mean_squared_error(y_pred_1, y[test]))

    means.append(np.mean(mse_estimates))
    std_devs.append(np.std(mse_estimates))


print(means)
print(std_devs)

plt.errorbar(C_values,means,yerr=std_devs,linewidth=3,capsize=5)
plt.title('5 Fold Lasso for multiple C values')
plt.xlabel('C Values');
plt.ylabel('Mean/STD dev value')
plt.show()
```

(ii.d)



```python
model_ridge_c_1 = Lasso(alpha=0.5)
folds = [2,5,10,25,40,100]
variances = []
means = []
for fold in folds:
    kf = KFold(n_splits=fold)
    mse_estimates = []
    for train, test in kf.split(X):
        model_ridge_c_1.fit(poly_transform.fit_transform(X[train]),y[train])
        X_test_space = create_test_space(4)
        y_pred_1 = model_ridge_c_1.predict(poly_transform.fit_transform(X[test]))
        mse_estimates.append(mean_squared_error(y_pred_1, y[test]))


    means.append(np.mean(mse_estimates))
    variances.append(np.var(mse_estimates))


print(variances)
print(means)

plt.errorbar(folds,means,yerr=variances,linewidth=3,capsize=5)
plt.xlabel('folds');
plt.ylabel('values');
plt.title('Ridge C=1 for different folds')
plt.show()
```