1-a) Visualization of given dataset:

This given data has been plotted on 3-d graph below where axes represent Feature1, Feaure2 and the Target (Output) from the given dataset. Each data point is represented with 'red' colour.

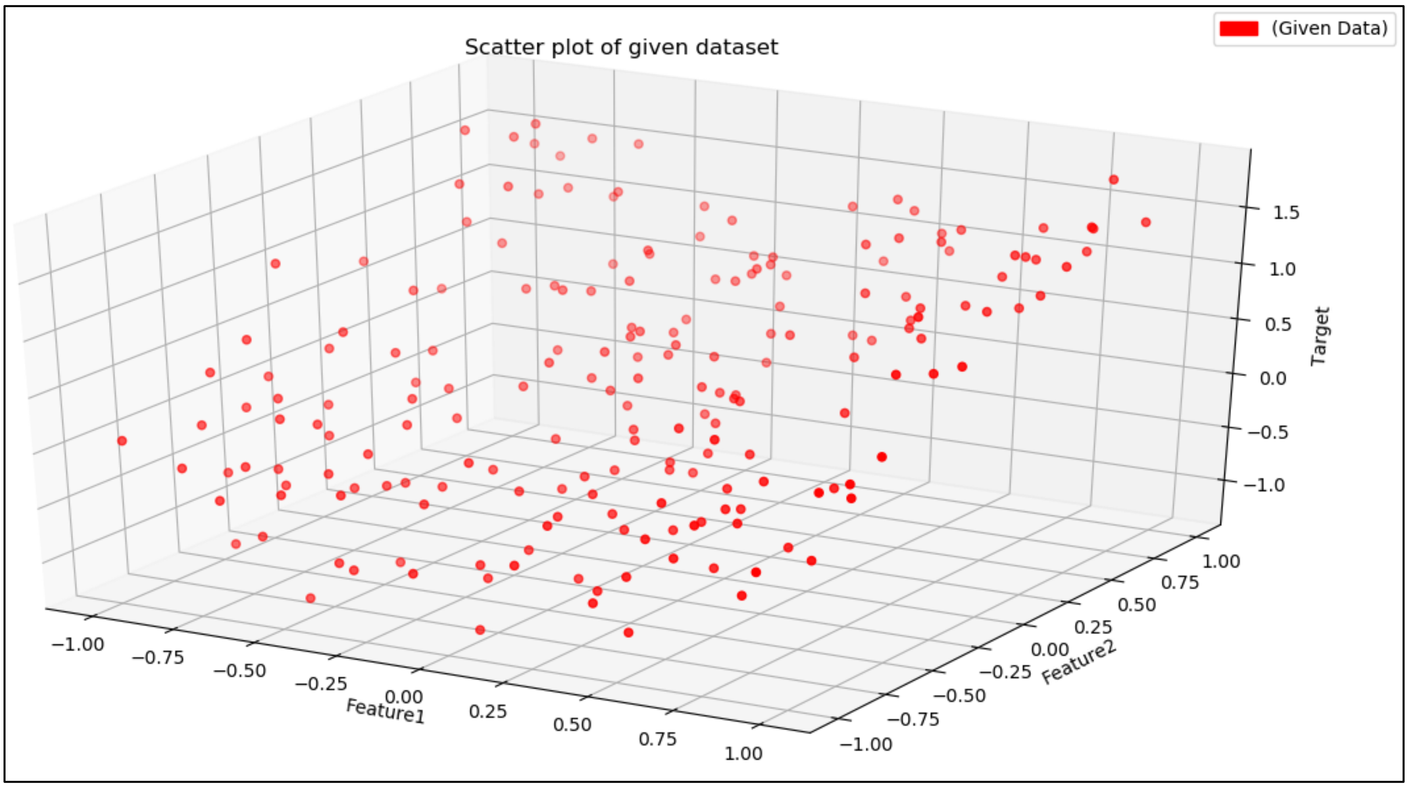
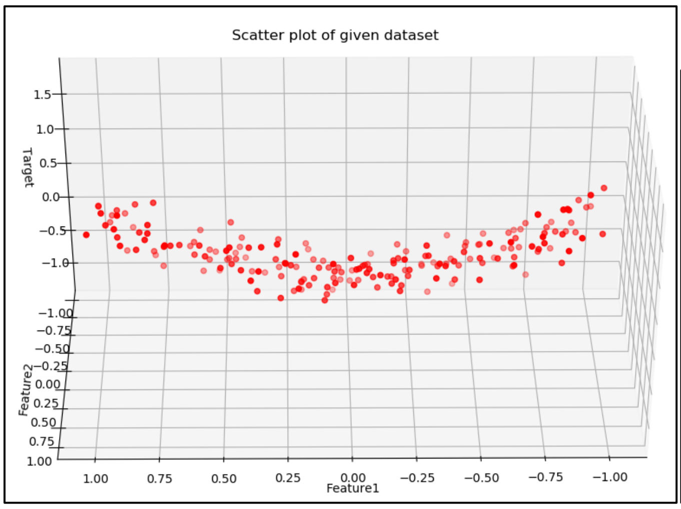


Figure 1-a1: Visualization of training dataset

The given data lies on a **curved surface.** We can confirm this if we look at this data from a different perspective. Figures below help us understand relationship between each feature and the target. Figure 1-a2 suggests that Feature1 and Target are related by a curved (probably quadratic) relationship. Figure 1-a3 represents somewhat linear relationship between Feature2 and the Target. Thus, we can confirm it lies on curved surface.



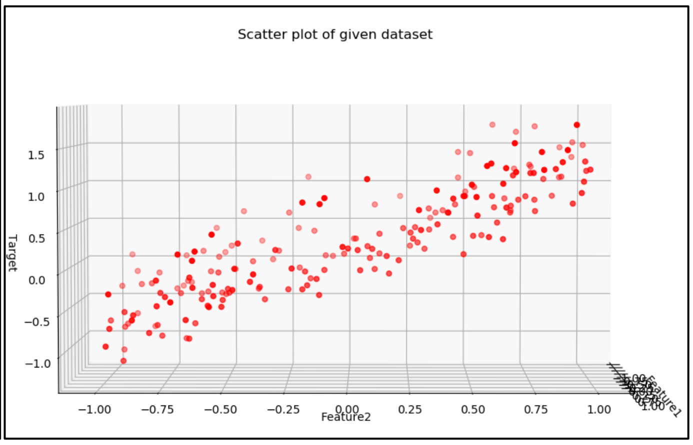


Figure 1-a2: Relation between Feature1 and Target

Figure 1-a3: Relation between Feature2 and Target

1-b) Lasso Regression with different values of C:

The given data set has only two features, but we have created additional features by using PolynomialFeatures function in sklearn.

```
poly = PolynomialFeatures(5)
X = poly.fit_transform(X)
```

This function helps to create additional features of all combinations of powers up to degree five. If **a** and **b** are the two input features, after using this function on degree=5, we will have the following combinations as our feature set:

1	a	b	a^2	ab	b^2	a^3	a^2b	ab^2	b^3	a^4	a^3b	a^2b^2	ab^3	b^4	a^5	a^4b	a^3b^2	a^2b^3	ab^4	b^5

We are using the Lasso Regression model, from sklearn library, where we are using different values of C to train the model. C is the value which helps us define the strength of penalty ($\theta^T\theta/2C$). We are also using train test split in order to judge the performance of the model for different values of C, by using the score function. The **score** values represent the accuracy of the model, i.e., out of all predictions, how many predictions were correct.

```
from sklearn import linear_model
from sklearn.model_selection import train_test_split
C_Values = [1, 5, 10, 50, 100, 500]
lasso_array=[]
for Ci in C_Values:
    lasso_reg = linear_model.Lasso(alpha=1/(2*Ci))

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    lasso_reg.fit(X_train,y_train)
    lasso_array.append(lasso_reg)

    print("\nC value: "+str(Ci))
    print("Lasso coeff: "+str(lasso_reg.coef_))
    print("Lasso score: "+str(lasso_reg.score(X_test,y_test)))
```

Table 1-b: Coefficients and the score of the models for different values of C.

C	1	a	b	a^2	ab	b^2	a^3	a^2b	ab^2	b^3	a^4
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.707	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.849	0.273	0.000	0.000	0.000	0.000	0.000	0.000	0.000
50	0.000	0.000	0.967	0.743	0.000	0.000	0.000	0.000	0.000	0.000	0.000
100	0.000	-0.004	0.982	0.803	0.000	0.000	0.000	0.000	-0.011	0.000	0.000
500	0.000	0.003	0.961	0.894	-0.032	0.000	0.000	0.097	-0.022	0.002	0.000

C	a^3b	a^2b^2	ab^3	b^4	a^5	a^4b	a^3b^2	a^2b^3	ab^4	b^5	Score
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**-0.013**
5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.708**
10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.831**
50	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.920**
100	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.925**
500	0.000	-0.087	0.000	0.000	0.000	0.000	-0.101	0.000	0.000	0.000	**0.921**

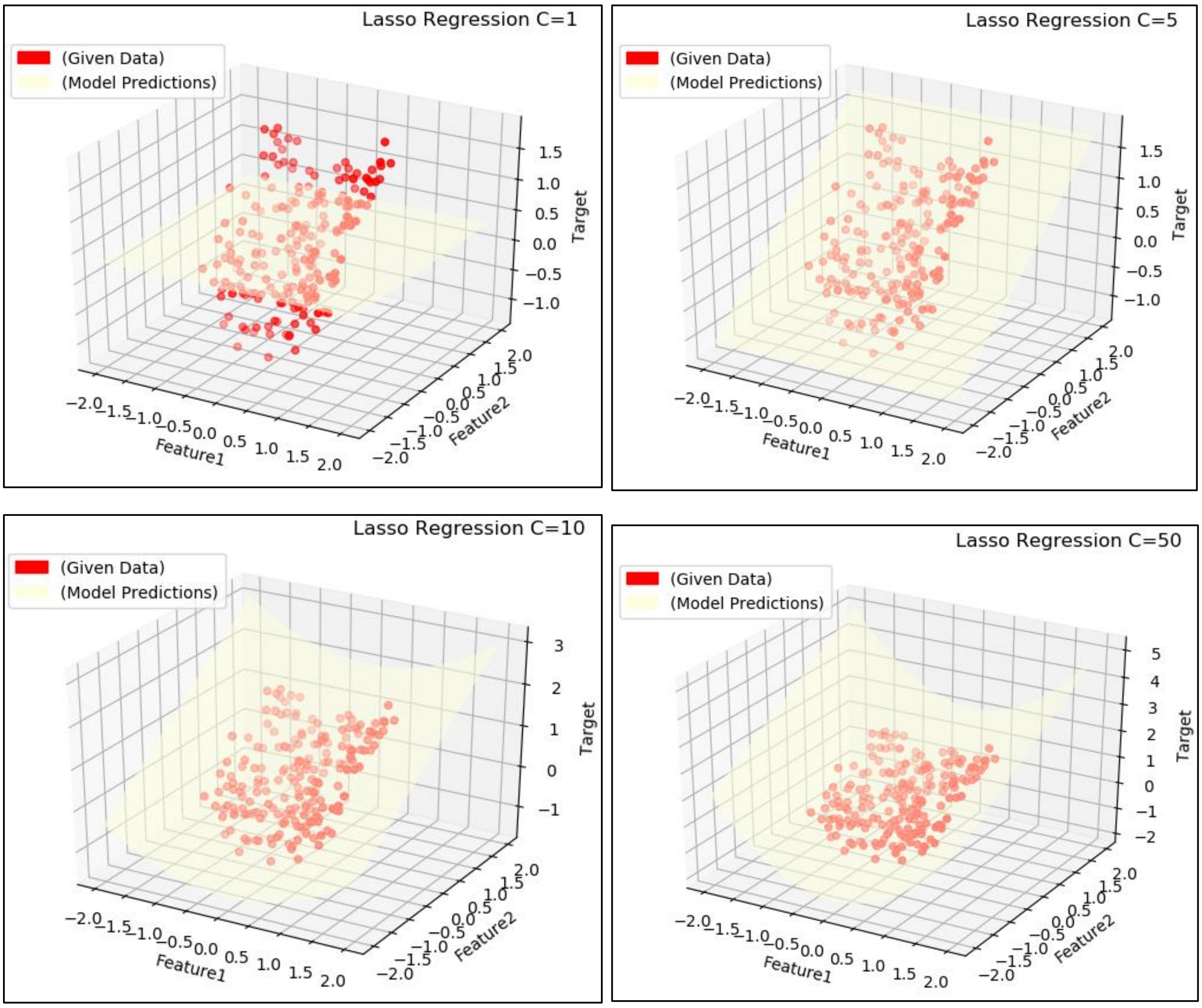
The model has been trained on different C values, i.e., **1, 5, 10, 50, 100, 500**. From the table above we can infer **C=1**, is very strong penalty and all the coefficients of the model are 0. The score is also 0 which means the model is not able to predict anything. When **C=5**, Only one coefficient is present which is related to feature2 (b), the score has increased to 0.7 which means this model is 70% correct. For **C=10 & 50**, the model has picked up two coefficients for **b** and **a²**, which we were expecting as per figure 1-a2 and 1-a3. The score also increases for these values of C. For **C=100 & 500**, the number of non-zero coefficients increase. Many of these might not even be necessary as there is no significant increase in the score. This confirms these higher values of C results in very weak penalty.

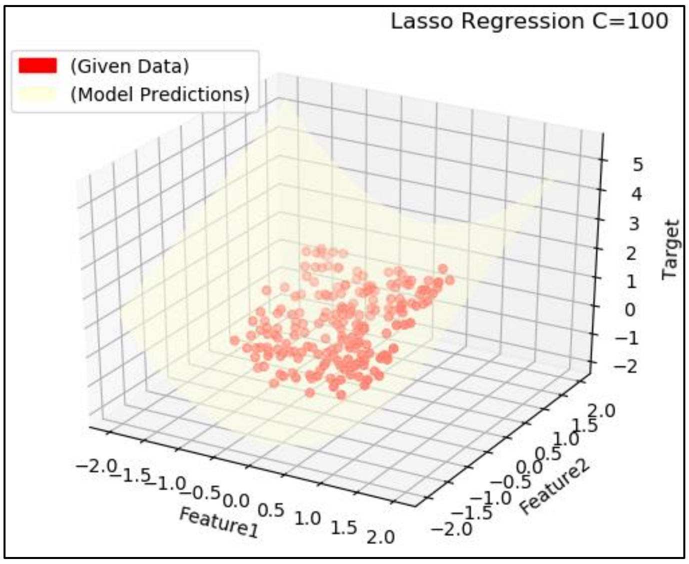
Thus, we can infer that **after increasing the value of C, the penalty becomes weaker and the number of non-zero coefficients increase**.

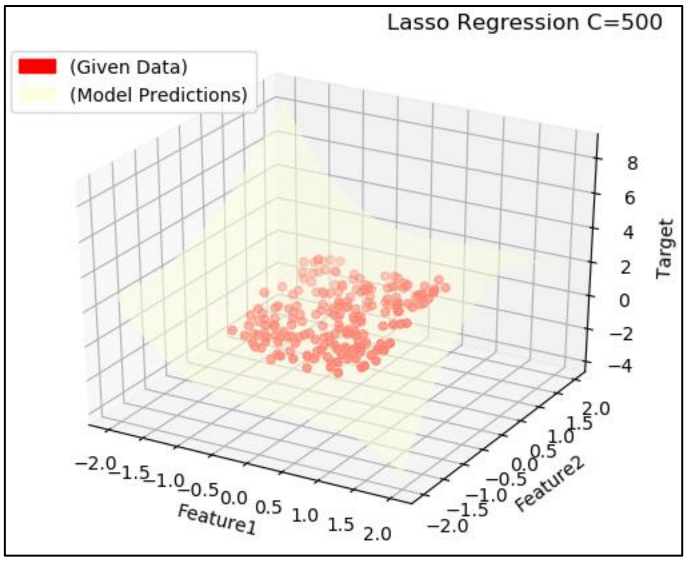
1-c) Prediction data visualization along with given data.

We have created test data features ranging from -2 to 2. The models with different values of C are used to make predictions on this test data. These predictions are plotted as a *'lightyellow'* coloured surface in the graphs below. While the given dataset is plotted as a scatter plot in *'red'* colour.

Figure 1-c: Visualization of Lasso Regression for different values of C







In the plots above, when **C=1**, the predictions lie on a horizontal plane which does not match the given data. When **C=5**, the predictions still lie on a plane, but the model is successful in capturing the linear relationship between Feature2 and the Target. When **C=10 and C=50**, the predictions lie on curved surface which closely matches the given data. When **C=100 and C=500**, the surface looks too curved which might be a sign of overfitting.

1-d) Under-fitting and Over-fitting.

Under-fitting occurs when the model is too simple to capture the underlying relationships and patterns in the data. It fails to give correct predictions even for training data. **Over-fitting** occurs when the model modifies itself too much to accommodate all the training data, even the noise. When tested against training data it gives excellent accuracy but fails to perform when tested against completely unseen data.

With reference to plots in Figure 1-c and Table 1-b, we can see that when **C=1 & 5,** the model shows signs of underfitting. The surface in the plots is a simple plane which does not represent the training data correctly. The number of non-zero coefficients and the score is also very less. When **C=10 & 50**, the model looks appropriate as the number of coefficients is only 2 and the scores are good as well. Also, the plot shows a proper curved surface against the training data. When **C=100 & 500,** the number of coefficients increase but the score does not jump too much. Also, the plots show too much curvature on the prediction surface. This suggests the model is overfitting for C=100 & 500. So, we can vary the values of C in such way that the accuracy of model increases without increasing the number of parameters, i.e., less non-zero coefficients.

1-e) Ridge Regression.

Using the same training set, we have trained Ridge Regression model for different values of C **[0.0001, 0.001, 0.01, 0.1, 1]**. Following table provides the coefficients and accuracy score (no. of correct predictions out of total) for the trained regression model for the given C values:

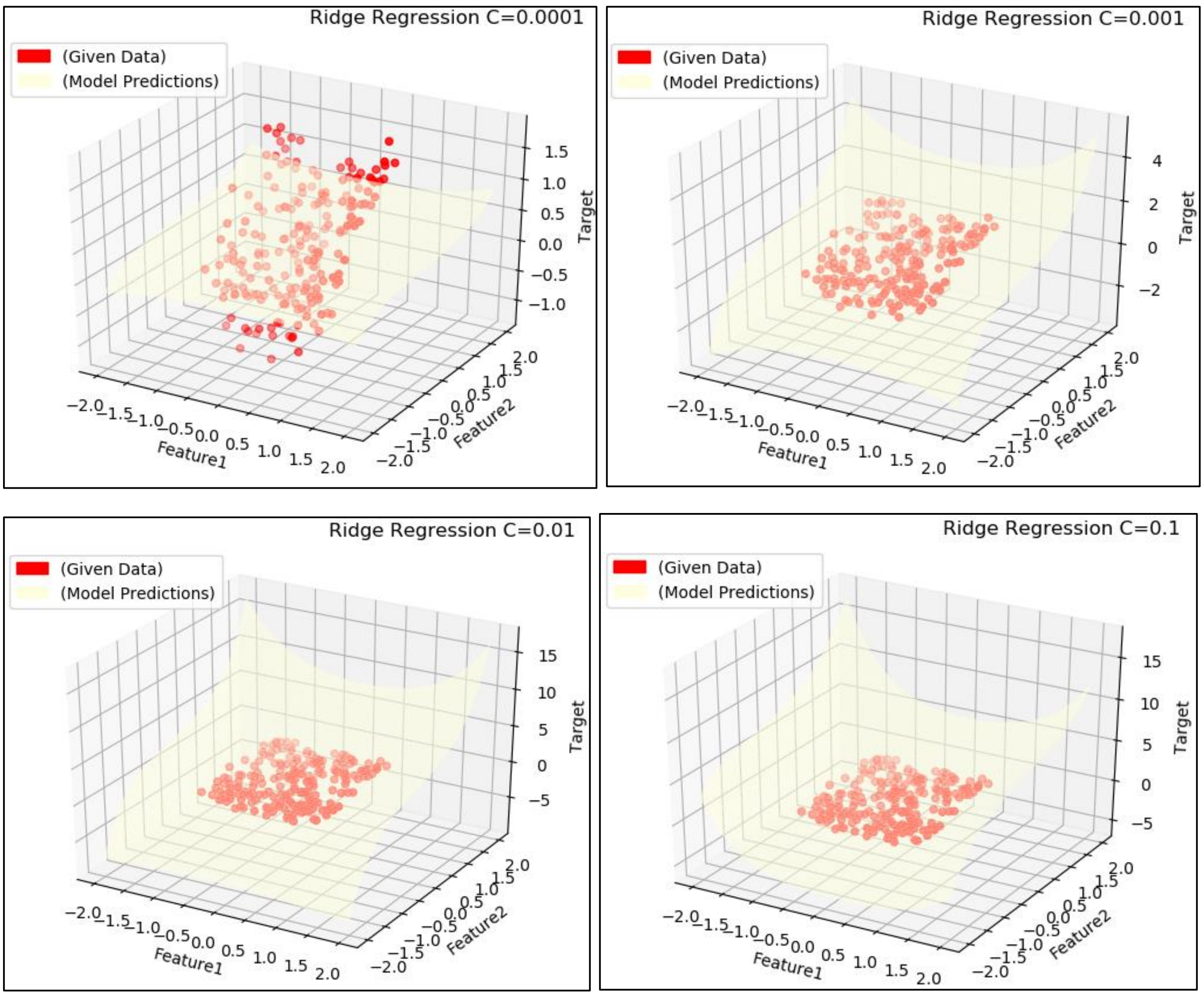
Table 1-e1: Coefficients of Ridge Regression for different values of C

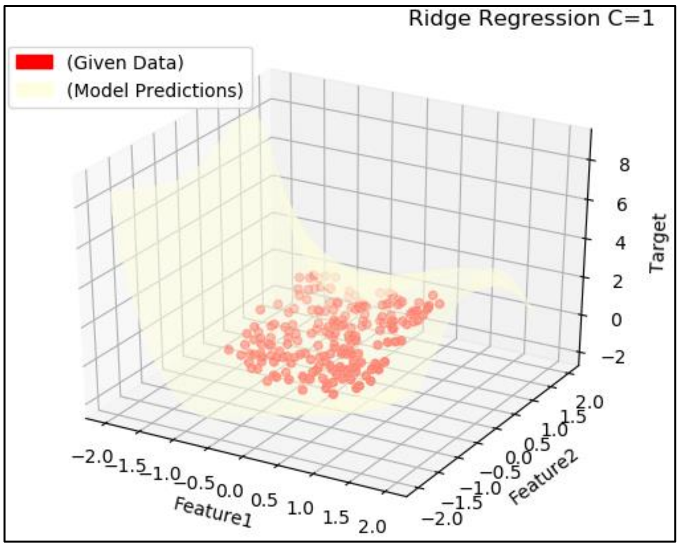
C	**1**	**a**	**b**	**a^2**	**ab**	**b^2**	**a^3**	**a^2b**	**ab^2**	**b^3**	**a^4**
0.0001	0.0000	0.0001	0.0111	0.0022	0.0000	0.0005	0.0004	0.0041	0.0000	0.0067	0.0018
0.001	0.0000	0.0006	0.0941	0.0209	-0.0002	0.0047	0.0028	0.0342	-0.0005	0.0558	0.0176
0.01	0.0000	-0.0071	0.3911	0.1489	-0.0051	0.0191	0.0065	0.1282	-0.0101	0.2078	0.1239
0.1	0.0000	-0.0064	0.7077	0.4136	-0.0342	0.0157	0.0027	0.1655	-0.0446	0.2213	0.3052
1	0.0000	0.0153	0.8618	0.6955	-0.0900	0.0814	0.0156	0.2609	-0.1893	0.1830	0.2480

C	**a^3b**	**a^2b^2**	**ab^3**	**b^4**	**a^5**	**a^4b**	**a^3b^2**	**a^2b^3**	**ab^4**	**b^5**	**Score**
0.0001	0.0001	0.0010	0.0000	0.0003	0.0004	0.0024	0.0001	0.0025	-0.0001	0.0047	**0.0184**
0.001	0.0013	0.0093	-0.0002	0.0029	0.0033	0.0200	0.0009	0.0205	-0.0009	0.0394	**0.2367**
0.01	0.0045	0.0569	0.0001	0.0082	0.0099	0.0669	-0.0027	0.0686	-0.0076	0.1341	**0.7376**
0.1	-0.0001	0.0742	0.0105	-0.0225	0.0020	0.0313	-0.0474	0.0284	0.0053	0.0496	**0.9039**
1	0.0366	-0.1608	0.0570	-0.0518	0.0420	-0.1395	-0.2203	0.0001	0.2233	-0.0990	**0.9162**

The major difference in Lasso regression and Ridge regression is use of regularization. Lasso uses L1 regularization whereas Ridge uses L2 regularization. The L1 regularization encourages the sparsity of solution (few non-zero coefficients) as compared to L2. Looking at both the tables 1-b and 1-e1, we can clearly see that the **number of non-zero coefficients in Lasso regression are very less as compared to the number of non-zero coefficients in Ridge regression.** For both models the **number of non-zero coefficients increase as we increase the value of C** just the difference is Ridge has multiple non-zero coefficients whereas Lasso has only a selected few.

Figure 1-e2: Prediction plots for Ridge Regression for different values of C





Referring the visualizations above, Ridge regression also plots a plane as the prediction surface for very small values of **C=0.0001**. The curvature of the prediction surface increases as we increase the value of C and clearly shows overfitting when **C=1**. But when compared to Lasso, these prediction surfaces are much more curved on the edges and corners. This might be due to the fact that Ridge has multiple non-zero coefficients which might be creating these curves on the edges and corners.

2-a) 5-fold Cross Validation for Lasso Regression

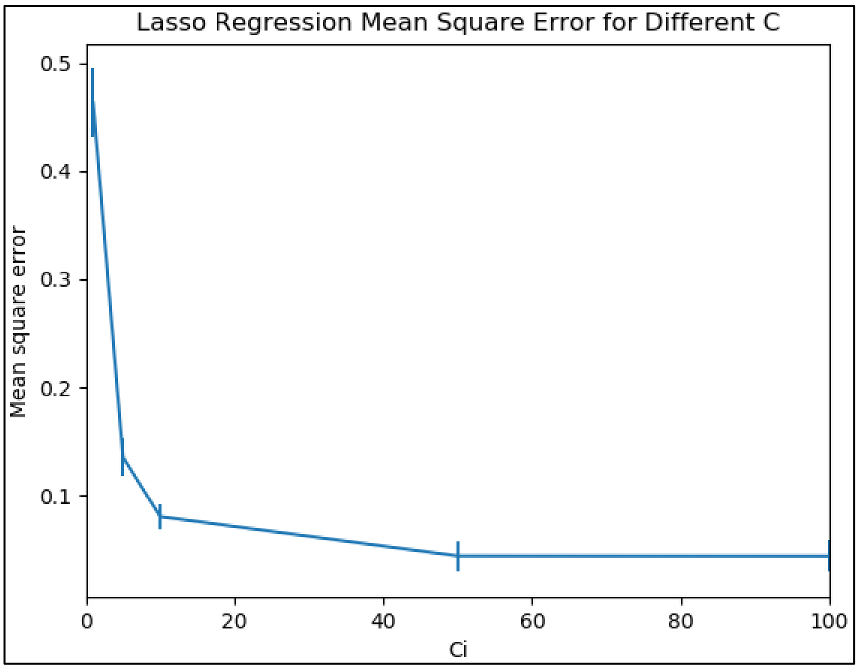
For all different models (different C) of Lasso Regression, we have used K-Fold cross validation with 5 splits of data. Trained the model on all splits and recorded the error and standard deviations in arrays mean_error and std_error respectively.

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5)
temp = []

for train, test in kf.split(X):
    lasso_reg.fit(X[train],y[train])
    ypred = lasso_reg.predict(X[test])
    from sklearn.metrics import mean_squared_error
    temp.append(mean_squared_error(y[test],ypred))
mean_error.append(np.array(temp).mean())
std_error.append(np.array(temp).std())
```

Plotting C for values between 0.1 and 100 as there was significant change in accuracy of model in this range. So, visualizing the mean errors in this range would of importance. The plot for Lasso Regression mean square errors is shown below:

Figure 2-a: Mean square error for Lasso Regression



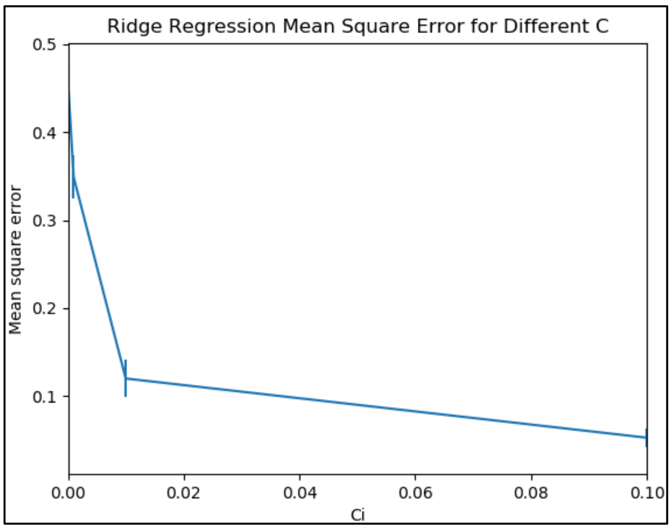
2-b) Value of C:

Based on the cross-validation data and the mean square error graph, I would choose **C value=10.** As seen from the figure the error is much less when compared to **C < 10.** For **C > 10**, the error starts decreasing but forms an asymptote. If we move more towards this side there is a chance of overfitting. That is why C=10 seems like a reasonable choice.

2-c) 5-fold Cross validation for Ridge Regression.

Similar to Lasso regression we, we have used K-fold cross validation for Ridge Regression and recorded the error and standard deviation. The error graph is shown below:

Figure 2-c: Mean square error for Ridge Regression



We can see that the mean square error decreases as the value of C increases. For Ridge regression the optimal value of C would be **0.01** as the error is not too high or too low.

Appendix:

Code referred from lecture slides and official documentation of sklearn and matplotlib.

Code for using colour boxes in legends has been referred from: https://www.geeksforgeeks.org/how-to-manually-add-a-legend-with-a-color-box-on-a-matplotlib-figure/

```
# Name: Omkar Pramod Padir
# Student Id: 20310203
# Dataset id:4-4-4
# Course: Machine Learning CS7CS4
# Week 3 Assignment


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import matplotlib.patches as mpatches


# Part 1-A starts here

# Load data and create arrays of input and output

df = pd.read_csv("ML_W3_DATA.csv")

X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]

from mpl_toolkits.mplot3d import Axes3D
fig=plt.figure()
ax=fig.add_subplot(111,projection='3d')
ax.scatter(X1,X2,y, c='red')

# Used to add color legends in the plots
# Code referred from: https://www.geeksforgeeks.org/how-to-manually-add-a-legend-with-
a-color-box-on-a-matplotlib-figure/

red_patch = mpatches.Patch(color='red', label='(Given Data)')
yellow_patch = mpatches.Patch(color='lightyellow', label='(Model Predictions)')

# add labels and display
ax.set_xlabel('Feature1')
ax.set_ylabel('Feature2')
ax.set_zlabel('Target')
plt.legend(handles=[red_patch])
plt.title('Scatter plot of given dataset')
plt.show()

# Part 1-B starts here

from sklearn.preprocessing import PolynomialFeatures

# this function will provide all combinations of inputs to degrree given in parameters
# eg. degree 3 for inputs a and b will give 1, a, b, a^2, a*b, b^2, a^3, (a^2)*b ,
a*(b^2), b^3.
# https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
```

```
poly = PolynomialFeatures(5)
X = poly.fit_transform(X)


from sklearn import linear_model
from sklearn.model_selection import train_test_split
C_Values = [1, 5, 10, 50, 100, 500]         # Different C values for changing penalty 
strength
lasso_array=[]

for Ci in C_Values:
    lasso_reg = linear_model.Lasso(alpha=1/(2*Ci))

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 
random_state=42)
    lasso_reg.fit(X_train,y_train)
    lasso_array.append(lasso_reg)

    print("\nC value: "+str(Ci))
    print("Lasso coeff: "+str(lasso_reg.coef_))
    print("Lasso score: "+str(lasso_reg.score(X_test,y_test)))  # Shows the accuracy of 
the model


# 1-C starts here

# Create grid of inputs ranging from -2 to 2
X_test =[]
grid=np.linspace(-2,2)

for i in grid:
    for j in grid:
        X_test.append([i,j])

X_test = np.array(X_test)

# Use polynomialfeatures on the newly created test data.

poly = PolynomialFeatures(5)
X_test_poly = poly.fit_transform(X_test)

# Plot all predictions and training data for all models
i=0
for l in lasso_array:

    y_pred=l.predict(X_test_poly)

    #https://matplotlib.org/2.0.2/mpl_toolkits/mplot3d/tutorial.html

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # Plot the surface.
    cmap, norm = mcolors.from_levels_and_colors([-1, 0, 1], ['lightyellow', 
'lightyellow'])

    surf = ax.plot_trisurf(X_test[:,0],X_test[:,1],y_pred, cmap=cmap, alpha=0.6)

    ax.scatter(X1,X2,y,c='red')

    ax.set_xlabel('Feature1')
    ax.set_ylabel('Feature2')
    ax.set_zlabel('Target')
    plt.legend(handles=[red_patch,yellow_patch],loc=2)
    plt.title('Lasso Regression C='+str(C_Values[i]),loc='right')
    plt.show()
    i=i+1


# 1-e Ridge Regression
C_Values_ridge=[0.0001,0.001,0.01,0.1,1]
Ridge_array=[]
for Ci in C_Values_ridge:
    ridge_reg = linear_model.Ridge(alpha=1/(2*Ci))

    X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2,
random_state=42)
    ridge_reg.fit(X_train1,y_train1)
    Ridge_array.append(ridge_reg)

    print("\nC value: "+str(Ci))
    print("Ridge coeff: "+str(ridge_reg.coef_))
    print("Ridge score: "+str(ridge_reg.score(X_test1,y_test1)))

i=0
# Plot all predictions and training data for all models
for r in Ridge_array:

    y_pred=r.predict(X_test_poly)

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # Plot the surface.

    cmap, norm = mcolors.from_levels_and_colors([-1, 0, 1], ['lightyellow',
'lightyellow'])

    surf = ax.plot_trisurf(X_test[:,0],X_test[:,1],y_pred, cmap=cmap, alpha=0.6)

    ax.scatter(X1,X2,y,c='red')

    ax.set_xlabel('Feature1')
    ax.set_ylabel('Feature2')
    ax.set_zlabel('Target')
    plt.legend(handles=[red_patch,yellow_patch],loc=2)
    plt.title('Ridge Regression C='+str(C_Values_ridge[i]),loc='right')
    plt.show()
    i=i+1


# Part 2 Lasso Regression K-fold Cross Validation

mean_error=[]; std_error=[]

#Loop for all models
for Ci in C_Values:
    lasso_reg = linear_model.Lasso(alpha=1/(2*Ci))

    from sklearn.model_selection import KFold
    kf = KFold(n_splits=5)
    temp = []

    # Loop for all k-fold splits
    for train, test in kf.split(X):
        lasso_reg.fit(X[train],y[train])
        ypred = lasso_reg.predict(X[test])
        print("\nC value: "+str(Ci))
        print("Lasso coeff: "+str(lasso_reg.coef_))
        from sklearn.metrics import mean_squared_error
        temp.append(mean_squared_error(y[test],ypred))
    mean_error.append(np.array(temp).mean())
    std_error.append(np.array(temp).std())

# Plot the error bar for Lasso Regression
import matplotlib.pyplot as plt
plt.errorbar(C_Values,mean_error,yerr=std_error)
plt.xlabel('Ci'); plt.ylabel('Mean square error')
plt.xlim((0.1,100))
plt.title('Lasso Regression Mean Square Error for Different C')
plt.show()


# Part 2 for Ridge Regression K-fold Cross Validation

mean_error=[]; std_error=[]
#Loop for all models
for Ci in C_Values_ridge:
    ridge_reg = linear_model.Ridge(alpha=1/(2*Ci))

    from sklearn.model_selection import KFold
    kf = KFold(n_splits=5)
    temp = []

    # Loop for all k-fold splits
    for train, test in kf.split(X):
        ridge_reg.fit(X[train],y[train])
        ypred = ridge_reg.predict(X[test])
        print("\nC value: "+str(Ci))
        print("Ridge coeff: "+str(ridge_reg.coef_))

        from sklearn.metrics import mean_squared_error
        temp.append(mean_squared_error(y[test],ypred))
    mean_error.append(np.array(temp).mean())
    std_error.append(np.array(temp).std())

# Plot the error bar for Ridge Regression
import matplotlib.pyplot as plt
plt.errorbar(C_Values_ridge,mean_error,yerr=std_error)
plt.xlabel('Ci'); plt.ylabel('Mean square error')
plt.xlim((0.0001,0.1))
plt.title('Ridge Regression Mean Square Error for Different C')
plt.show()
```