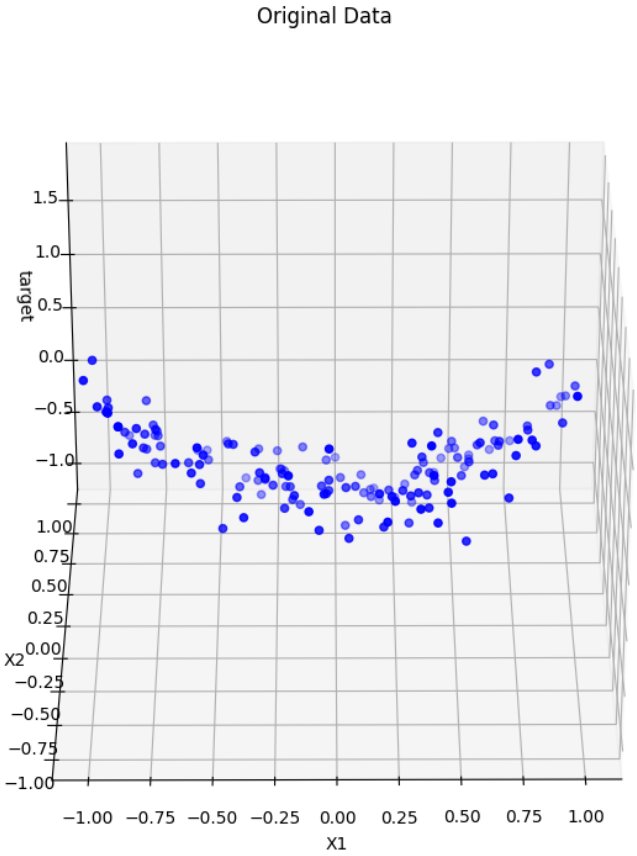
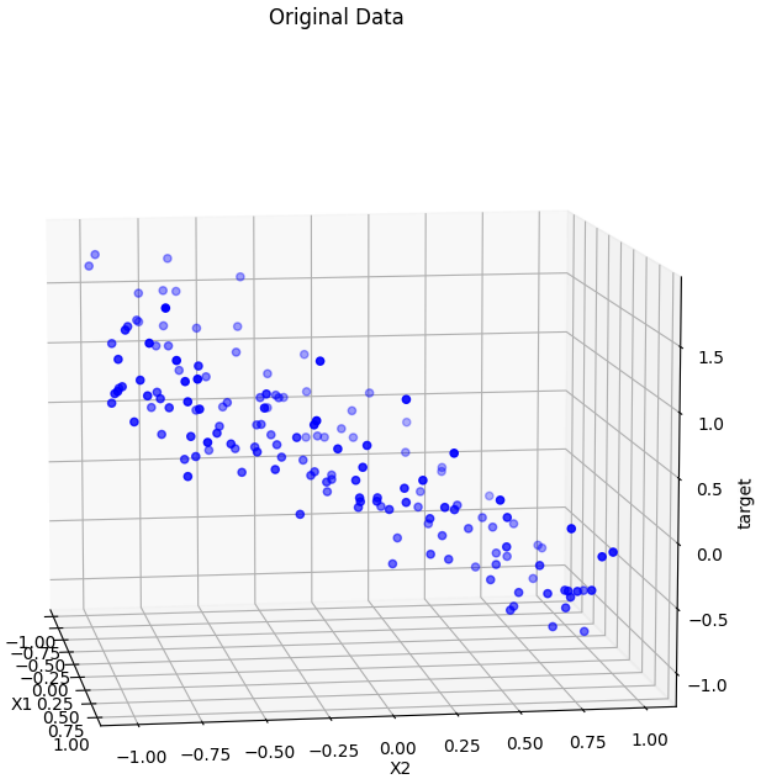
1. **(a)**





To plot the data, I used a subplot set to be in 3d, and then put a scatter plot on this sub plot. Before this however, I split my data into test and training data with sklearn's train_test_split function, I split the data so that 80% was test data and 20% was training data.
It is clear from looking at the data that the data lies on a curved plane.

(b) A quick note: sklearn doesn't use C simply as it is when applied to any model, instead, it uses 1/2C and calls this alpha, because of this, in the code, you will see that alpha is always set to be 1/2C, which is the same as using C as shown in the lectures.

For a C of 1, I got an intercept of 0.42611770777222485, for X1 -0.0, for X2 -0.0, for $X1^2$ 0.0, for $X2^2$ 0.0, for $X1^3$ 0.0, for $X2^3$ -0.0, for $X1^4$ -0.0, for $X2^4$ -0.0, for $X1^5$ -0.0, for $X2^5$ 0.0 and for X1*X2 0.0, at this value of C, it pushes all the parameters to be 0, this will give us a flat, horizontal plane when graphed in 3 dimensions. It got an R2 score of -0.01617634913711674.

For a C of 5, I got an intercept of 0.32912429338560145, for X1 -0.0, for X2 -0.7095716375058277, for $X1^2$ 0.0, for $X2^2$ 0.0, for $X1^3$ 0.0, for $X2^3$ -0.0, for $X1^4$ -0.0, for $X2^4$ -0.0, for $X1^5$ -0.0, for $X2^5$ 0.0 and for X1*X2 0.0, at C = 5, only X2 is not equal to 0, meaning that the data when graphed will be an angled plane, but it will only increase on the X2 axis, and as it is negative, it will rise as X2 decreases. It got an R2 score of 0.6539293006158375.

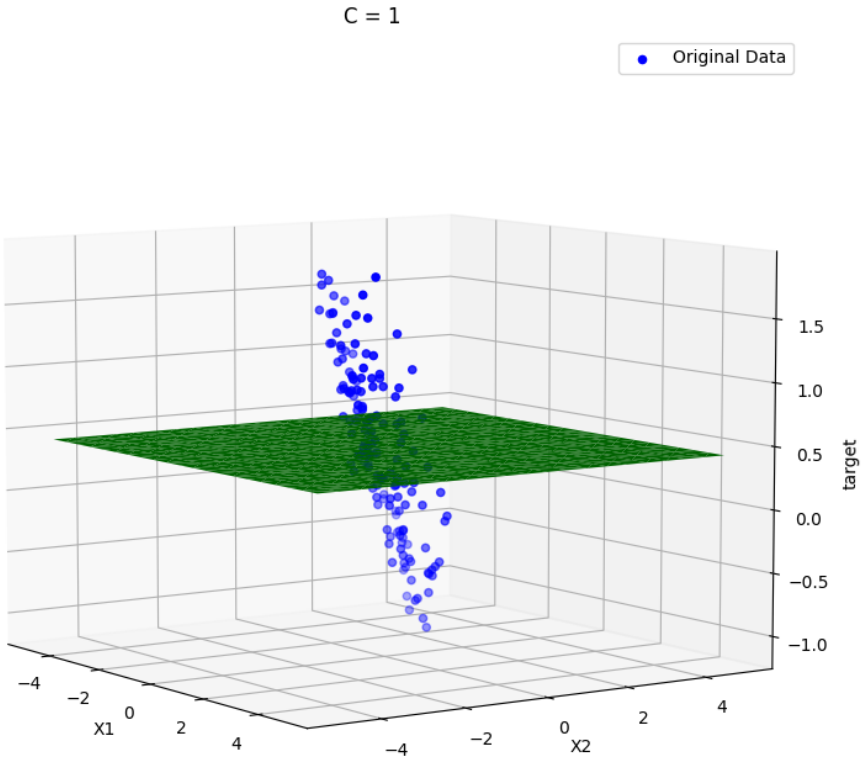
For a C of 10, I got an intercept of 0.1800383634393401, for X1 -0.0, for X2 -0.8437814954273734, for $X1^2$ 0.4088595088390016, for $X2^2$ 0.0, for $X1^3$ 0.0, for $X2^3$ -0.0, for $X1^4$ -0.0, for $X2^4$ -0.0, for $X1^5$ -0.0, for $X2^5$ 0.0 and for X1*X2 0.0, now $X1^2$ is no longer 0, we start to get a curve and we begin to model the trend in the data. It got an R2 score of 0.8198943735903561.

For a C of 50, I got an intercept of 0.01904983069539279, for X1 -0.0, for X2 -0.9476886627684485, for $X1^2$ 0.8807647115210181, for $X2^2$ 0.0, for $X1^3$ 0.0, for $X2^3$ 0.0, for $X1^4$ -0.0, for $X2^4$ -0.0, for $X1^5$ -0.05610982648140133, for $X2^5$ 0.0 and for X1*X2 0.0, with $X1^5$ no longer being 0, we start to see a slight bit of over-fitting. It got an R2 score of 0.919167648624373.

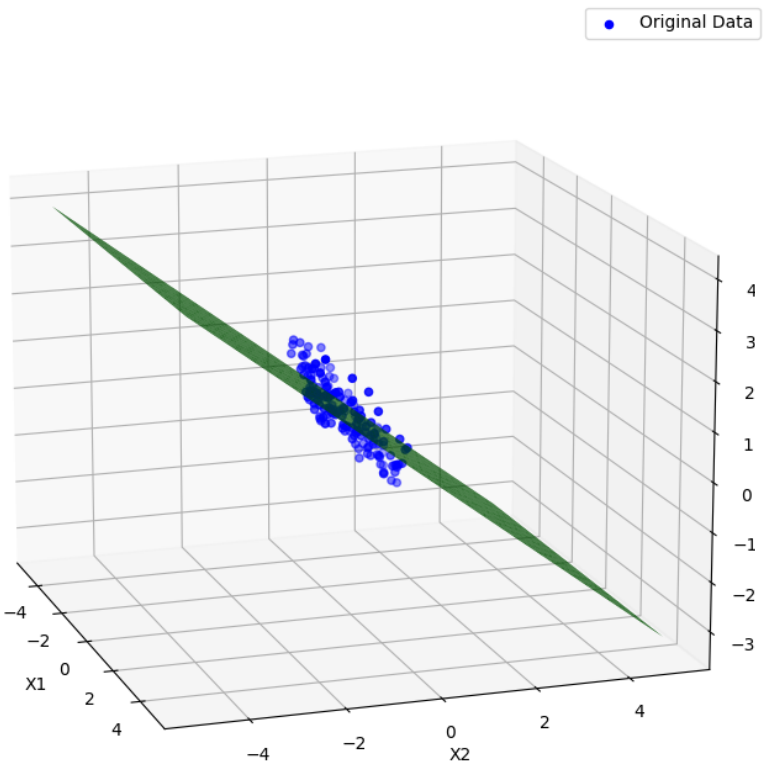
For a C of 1000, I got an intercept of 0.018293855537286074, for X1 -0.050027581215444004, for X2 -0.9193233655865748, for $X1^2$ 0.9683393317047743, for $X2^2$ 0.3292116023180137, for $X1^3$ -0.05928614503753289, for $X2^3$ 0.21306637325523373, for $X1^4$ -0.0, for $X2^4$ -0.11641144542344761, for $X1^5$ 0.0, for $X2^5$ 0.0 and for X1*X2 -0.0, here we see most of our 11 parameters have values, this will give us a large amount of over-fitting because every aspect of the training data is now giving informing the model, regardless if it really represents the trend. It got an R2 score of 0.928946036413391.

The overall trend is that, as C increases, we have less and less parameters that are equal to 0, increasing the complexity of our model.

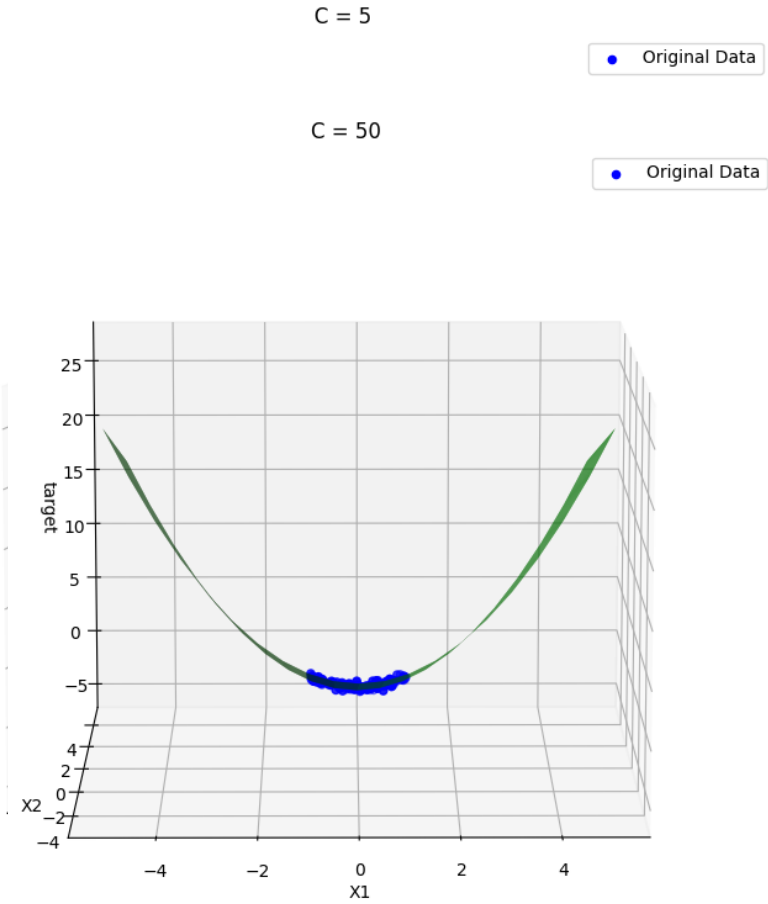
(c)



C = 5

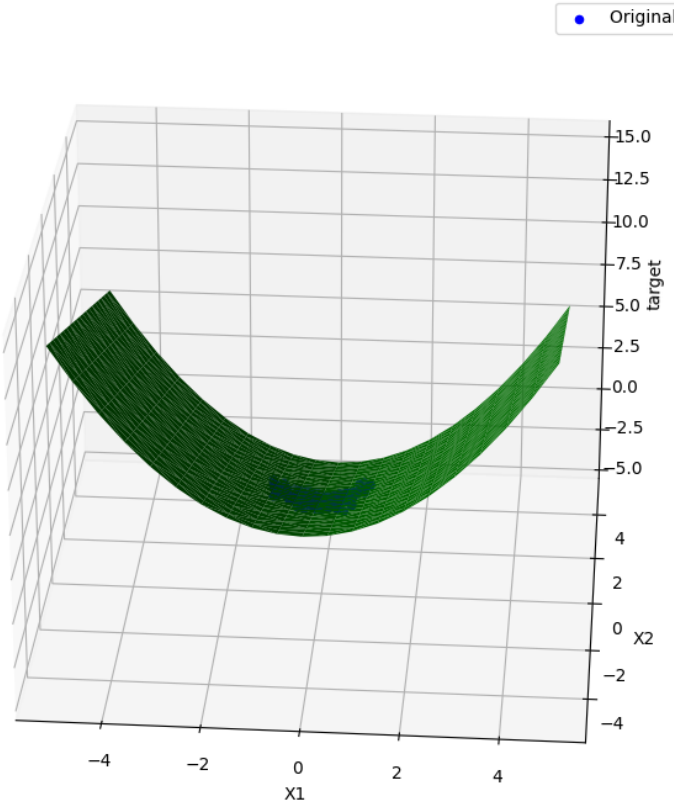


C = 5

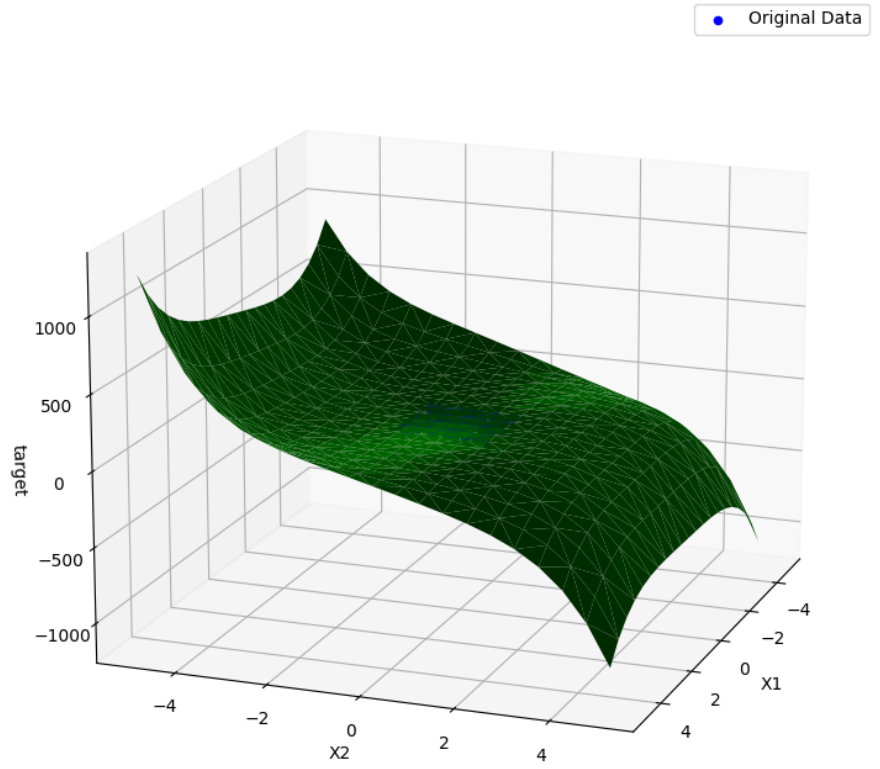


C = 50

C = 10



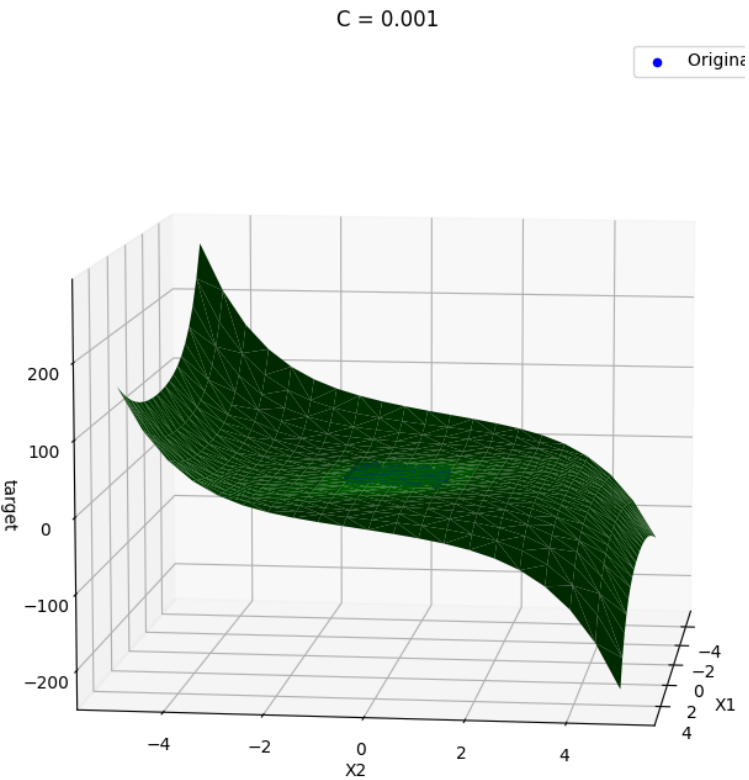
C = 1000

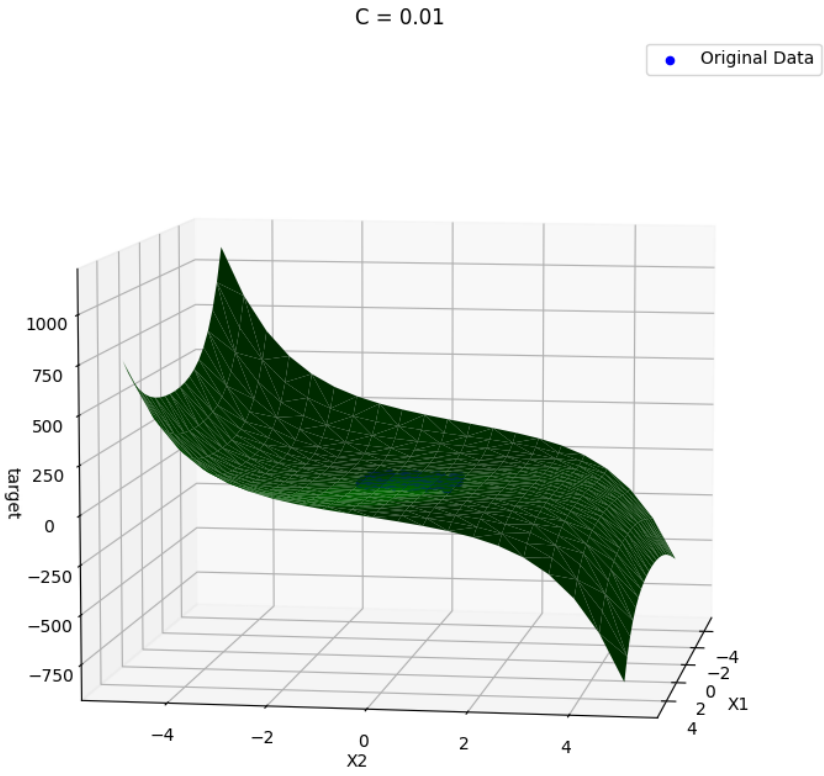


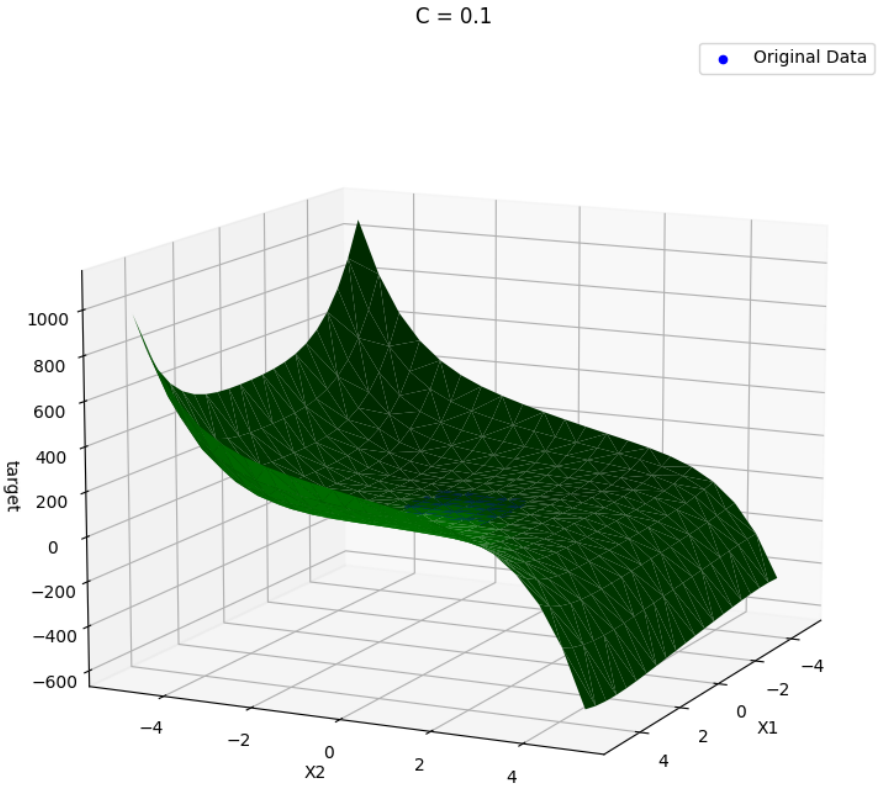
From these plots we can see that as C is increased, the complexity of the plane increases, while the increased complexity is welcome from the first 2 models, as the planes were not modeling the trend in the data at all, as we continue to increase C, we notice that the shapes being created slowly appear to no longer be representative of the trend of the data, but rather, just seems to represent the particular data that we have currently, and so will most likely not represent new data from the same source that well.

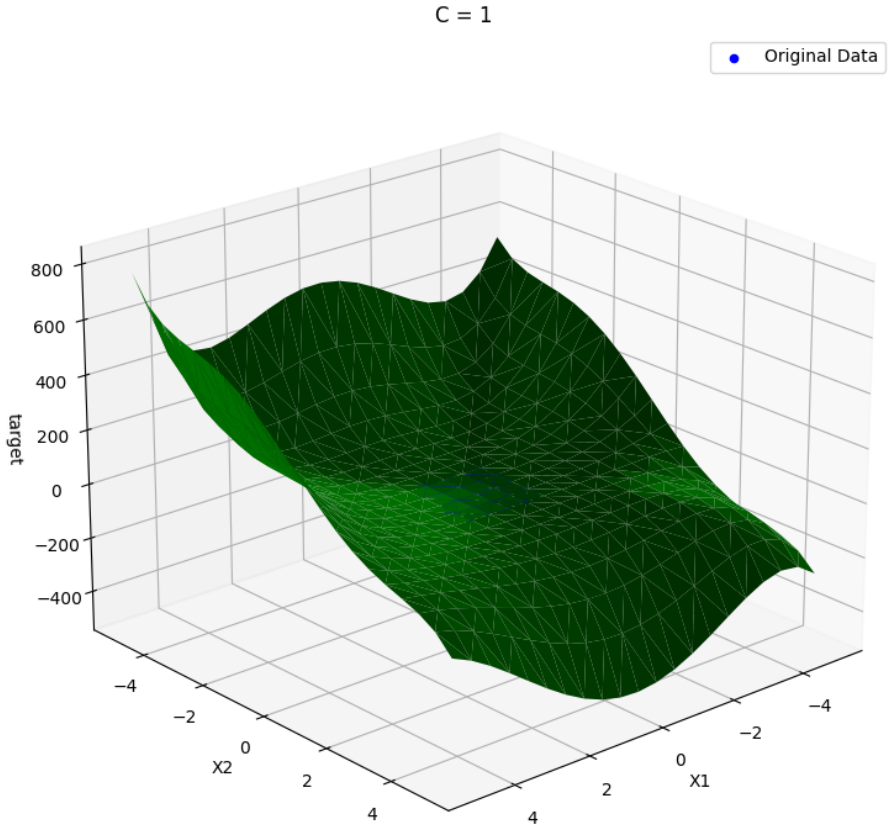
(d) Under-fitting is when the data does not model the trend of the data, it is quite easy to see as it will clearly no align to the trend, this can be seen with C=1 and C=5 above. Over-fitting, likewise, also does not model the trend of the data, but rather begins to fit the noise of the test data we have, this can be seen from C=1000 above, it can be hard to see if we are looking to closely at the test data itself, and don't take a step back to look at what is happening later on with our model.

(e)









I am going to simply print the data that I got for Ridge Regression as what is actually important is the difference in how the L1 and L2 regularisation affect the parameters. It's very clear when looking at the data, that there is no parameter that is ever set to 0, this is the key difference with L1 and L2 regularisation, L1 regularisation deals with the absolute value of $\theta^T\theta$, whereas L2 deals with the square of these values. Because of this, L2 will simply try to

make the parameters as small as possible but will never make them equal to 0, but L1 will try to make as many parameters as possible equal to 0. Because of this, and the fact that we are dealing with a large number of features, our data is always over-fitting, this can be clearly seen because our planes take on very strange shapes, and contort more and more as C increases.

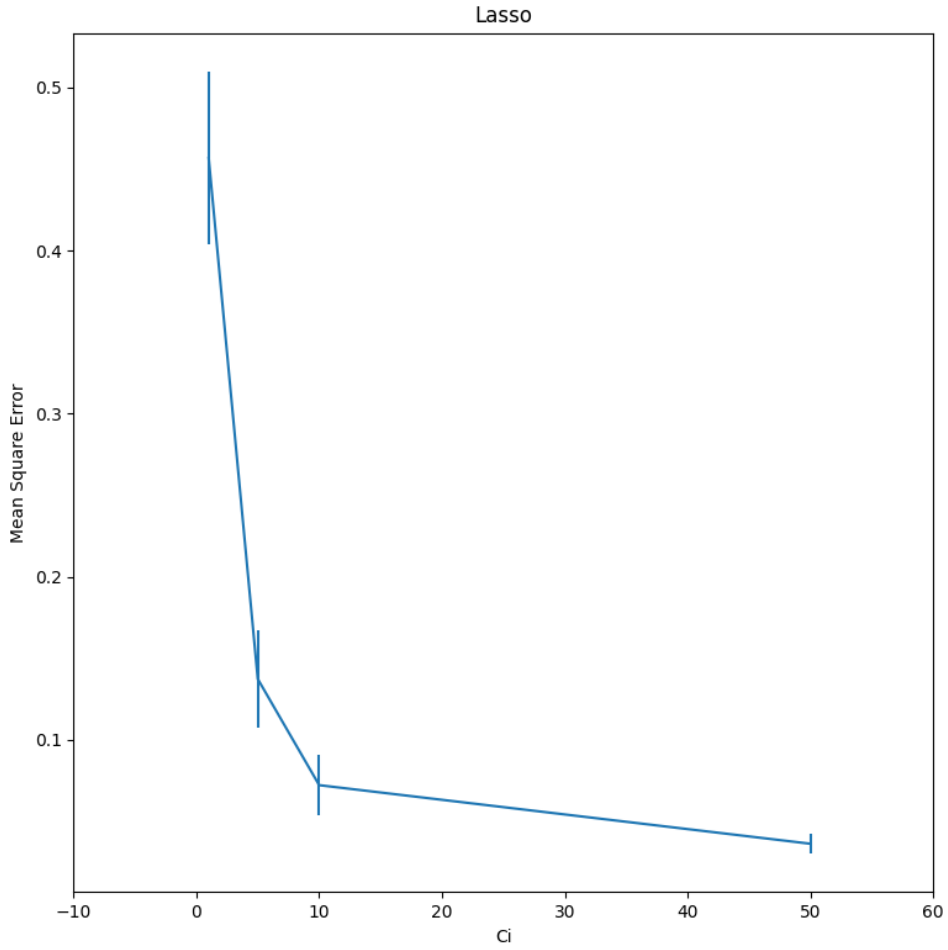
For a C of 0.001, I got an intercept of 0.36460908183450824, for X1 -0.00981245583089054, for X2 -0.06975003075223145, for $X1^2$ 0.022481309719468663, for $X2^2$ 0.009307877759249595, for $X1^3$ 0.006880685712014678, for $X2^3$ -0.008254948681968773, for $X1^4$ -0.02518006312304125, for $X2^4$ -0.0033824827873220357, for $X1^5$ -0.04047406018365284, for $X2^5$ 0.020290490700569364 and for X1*X2 0.008290805788780888. It got an R2 score of 0.18706579136702317.

For a C of 0.01, I got an intercept of 0.2472555179781723, for X1 -0.02499776985214944, for X2 -0.344754057934643, for $X1^2$ 0.15024208411154008, for $X2^2$ 0.036146137173342974, for $X1^3$ 0.01258933613108173, for $X2^3$ -0.018262560437356346, for $X1^4$ -0.11321580395544119, for $X2^4$ -0.006953029123389737, for $X1^5$ -0.185721261720643, for $X2^5$ 0.13220509919188256 and for X1*X2 0.03390471659821302. It got an R2 score of 0.6981003741409171.

For a C of 0.1, I got an intercept of 0.11346272866181228, for X1 -0.029110679717600597, for X2 -0.6735017132337592, for $X1^2$ 0.4365127155967094, for $X2^2$ 0.0535576484566027, for $X1^3$ -0.044909710968635476, for $X2^3$ .0.036277458717524136, for $X1^4$ -0.13381307334628803, for $X2^4$ -0.028106350065118937, for $X1^5$ -0.2521954026480163, for $X2^5$ 0.334509387087964 and for X1*X2 0.031170957750012527. It got an R2 score of 0.9156521619435319.

For a C of 1, I got an intercept of 0.04965907186027729, for X1 -0.04665288119659398, for X2 -0.854486429647797, for $X1^2$ 0.724522191230877, for $X2^2$ 0.20798505057673997, for $X1^3$ -0.07031184447688392, for $X2^3$ 0.1972583272844475, for $X1^4$ -0.07335726765896403, for $X2^4$ -0.18137120769380088, for $X1^5$ -0.1419027726885862, for $X2^5$ 0.23872430768658784 and for X1*X2 -0.007352949911218527. It got an R2 score of 0.9309286830176814.

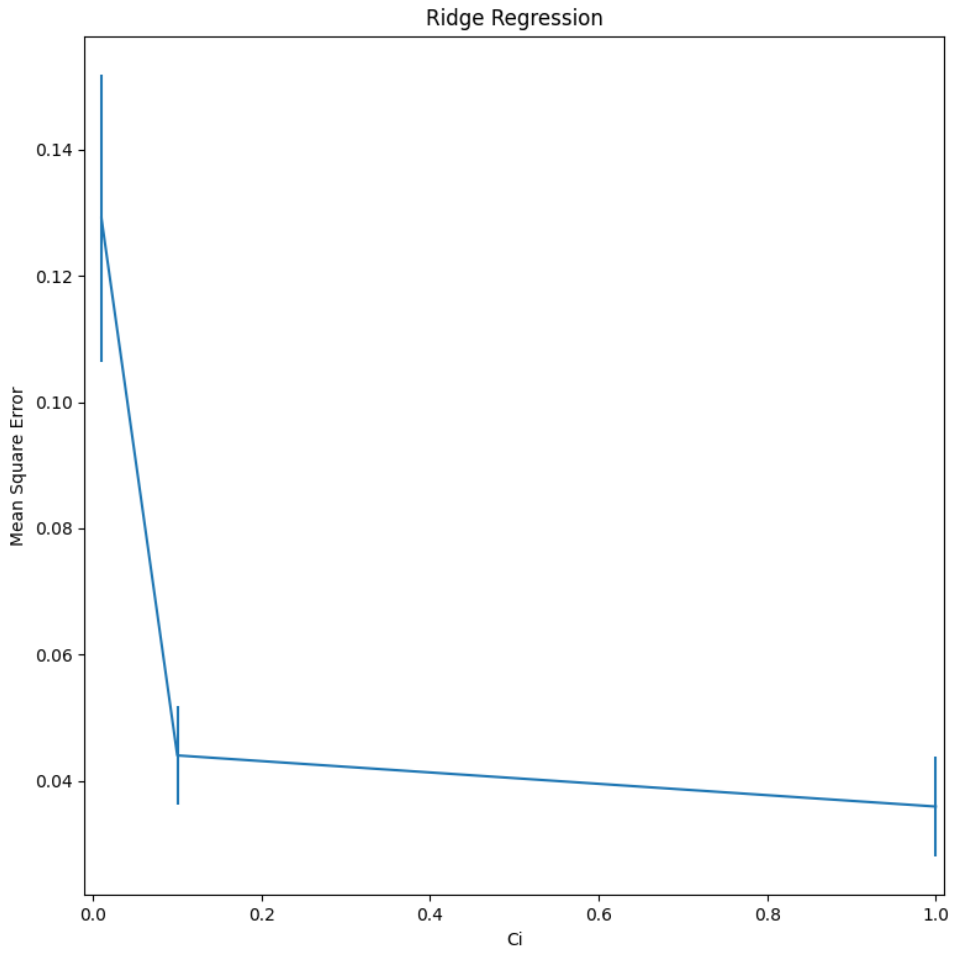
2. **(a)**



I have used the same C values as before apart from 1000 as they give context, along with the graph, as to the balance between mean square error and over-fitting that is going on when we choose a particular C value. For example, we can see from the graph, that a value of around 10 is probably optimal, as it is very simple, and the mean square error doesn't drop significantly within it's local vicinity.

(b) I would recommend using a C value of 10, as the difference in error is very small between it and the larger values of C and, as it's a simpler model and so avoids over-fitting much better than the other values of C. And while the difference in R^2 score is 0.1, this is not that large of a difference and what I belive is that it is likely that outside of the range of -1 to 1, the smaller C value of 10 will generalise the data better.

(c)



Once again I've used almost the same values of C apart from the most extreme value of 0.001

I would recommend a C value of 0.1, as it is simpler than a value of 1, but has a roughly equal mean square error.

Appendix:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import sklearn as sk
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
data = pd.read_csv('data.csv', comment='#')
X1 = data.iloc[:,0]
X2 = data.iloc[:,1]
X = np.column_stack((X1, X2))
Y = data.iloc[:,2]

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y,
test_size=0.2)
X1 = np.array([x[0] for x in Xtrain])
X2 = np.array([x[1] for x in Xtrain])
X = np.column_stack((X1, X2))
Y = np.array(Ytrain)

def plotOriginalData(ax=None):
if ax == None:
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.scatter(X1, X2, Y, color="blue", label='Original
Data')
ax.set_xlabel('X1'); ax.set_ylabel('X2');
ax.set_zlabel('target')
plt.title("Original Data")

def plotPredictions(model, features, X, ax, name):
predictions = model.predict(features)
ax.plot_trisurf(X[:, 0], X[:, 1], predictions, color='g')

def oneA():
plotOriginalData()
plt.show()

def oneB(lasso=True):
# First get the extra polynomial features
trans = PolynomialFeatures(degree=5)
# If we just want the combinations and want to ignore the
stand alone powers
#trans = PolynomialFeatures(degree=5,
interaction_only=True)
data = trans.fit_transform(X)
testData = trans.fit_transform(Xtest)

models = []
if lasso == True: Ci_range = [1, 5, 10, 50, 1000]
else: Ci_range = [0.001, 0.01, 0.1, 1]
for Ci in Ci_range:
if lasso: model = Lasso(alpha=1/(2*Ci))
else: model = Ridge(alpha=1/(2*Ci))
bestModel = [Lasso(), 10000]
```

```
kf = KFold(n_splits=5)
for train, test in kf.split(data):
model.fit(data[train], Y[train])
ypred = model.predict(data[test])
mean = mean_squared_error(Y[test], ypred)
if mean < bestModel[1]:
bestModel = [model, mean]
model = bestModel[0]
pred = model.predict(testData)
print(f'For a C of {Ci}, I got an intercept of
{model.intercept_}, for X1 {model.coef_[1]}, for X2
{model.coef_[2]}, for X1^2 {model.coef_[3]}, for X2^2
{model.coef_[4]}, for X1^3 {model.coef_[5]}, for X2^3
{model.coef_[6]}, for X1^4 {model.coef_[7]}, for X2^4
{model.coef_[8]}, for X1^5 {model.coef_[9]}, for X2^5
{model.coef_[10]} and for X1*X2 {model.coef_[11]}\n')
print(f'It got an R2 score of {sk.metrics.r2_score(Ytest,
pred)}\n')
models.append([bestModel[0], Ci])

return models

def oneC(models):
# this simply makes a grid from -5 to 5, making a jump of
.5 every time
# the grid will have a combination of every possibel X1
and X2
grid = np.mgrid[-5:5:21j, -5:5:21j].reshape(2, -1).T
# This is the grid with polynomial features of degree 5
Xtest = PolynomialFeatures(degree=5).fit_transform(grid)
for m, name in models:
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
plotPredictions(m, Xtest, grid, ax, name)
plotOriginalData(ax)
plt.title(f"C = {name}")
plt.legend()
plt.show()

def twoA(lasso=True):
trans = PolynomialFeatures(degree=5)
data = trans.fit_transform(X)

mean_error=[]; std_error=[]
if lasso == True: Ci_range = [1, 5, 10, 50]
else: Ci_range = [0.01, 0.1, 1]
for Ci in Ci_range:
if lasso: model = Lasso(alpha=1/(2*Ci))
```

```
else: model = Ridge(alpha=1/(2*Ci))
temp=[]
kf = KFold(n_splits=5)
for train, test in kf.split(data):
model.fit(data[train], Y[train])
ypred = model.predict(data[test])
mean = mean_squared_error(Y[test], ypred)
temp.append(mean)

mean_error.append(np.array(temp).mean())
std_error.append(np.array(temp).std())
plt.errorbar(Ci_range, mean_error, yerr=std_error)
plt.xlabel('Ci')
plt.ylabel('Mean Square Error')
if lasso == True:
plt.xlim((-10,60))
plt.title('Lasso')
else:
plt.xlim((-0.01, 1.01))
plt.title('Ridge Regression')
plt.show()

def part1():
print('(i)')

# (a)
oneA()

# (b)
models = oneB()
# (c)
oneC(models)

# (e)
models = oneB(False)
oneC(models)

def part2():
print('(ii)')

# (a)
twoA()

# (c)
twoA(False)

part1()
part2()
```