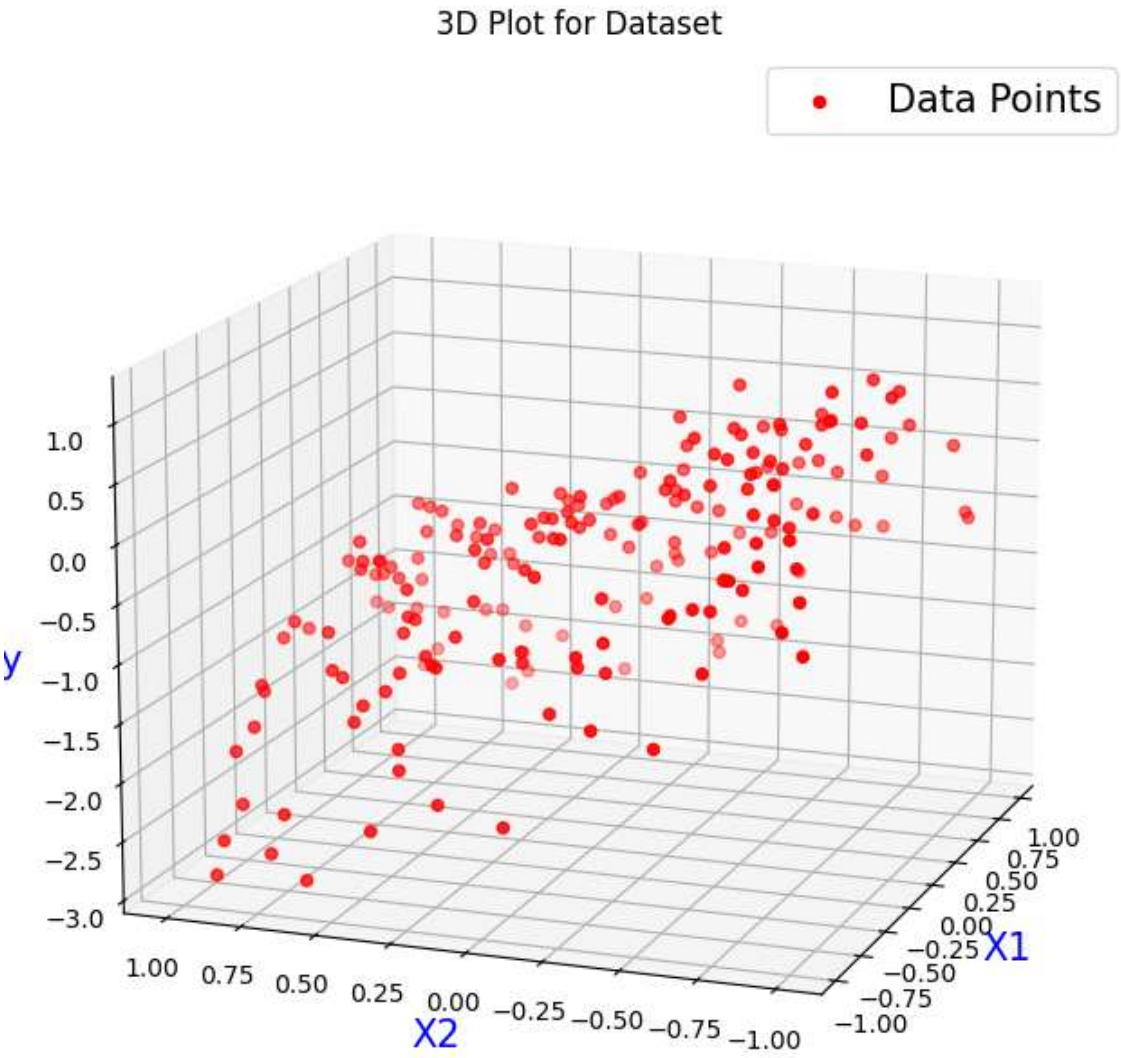
i(a)

The dataset was imported and divided into three data frames: X1 for feature1, X2 for feature2 and y for target. A 3D scatter plot was plotted using matplotlib python library with X1 (feature1) on x-axis, X2 (feature2) on y-axis and y (target) on z-axis. Data points are plotted in red.



From above graph, at first glance, it seems that for training data a curve would be a better fit than a plane.

i(b)

Using PolynomialFeatures library from sklearn.processing, polynomial features up to degree 5 were added using X1 and X2 features. The features now include : *1, x1, x2, x1^2, x1*x2, x2^2, x1^3, x1^2 * x2, x1 * x2^2, x2^3, x1^4, x1^3 * x2, x1^2 * x2^2, x1 * x2^3, x2^4, x1^5, x1^4 * x2, x1^3 * x2^2, x1^2 * x2^3, x1 * x2^4, x2^5.*

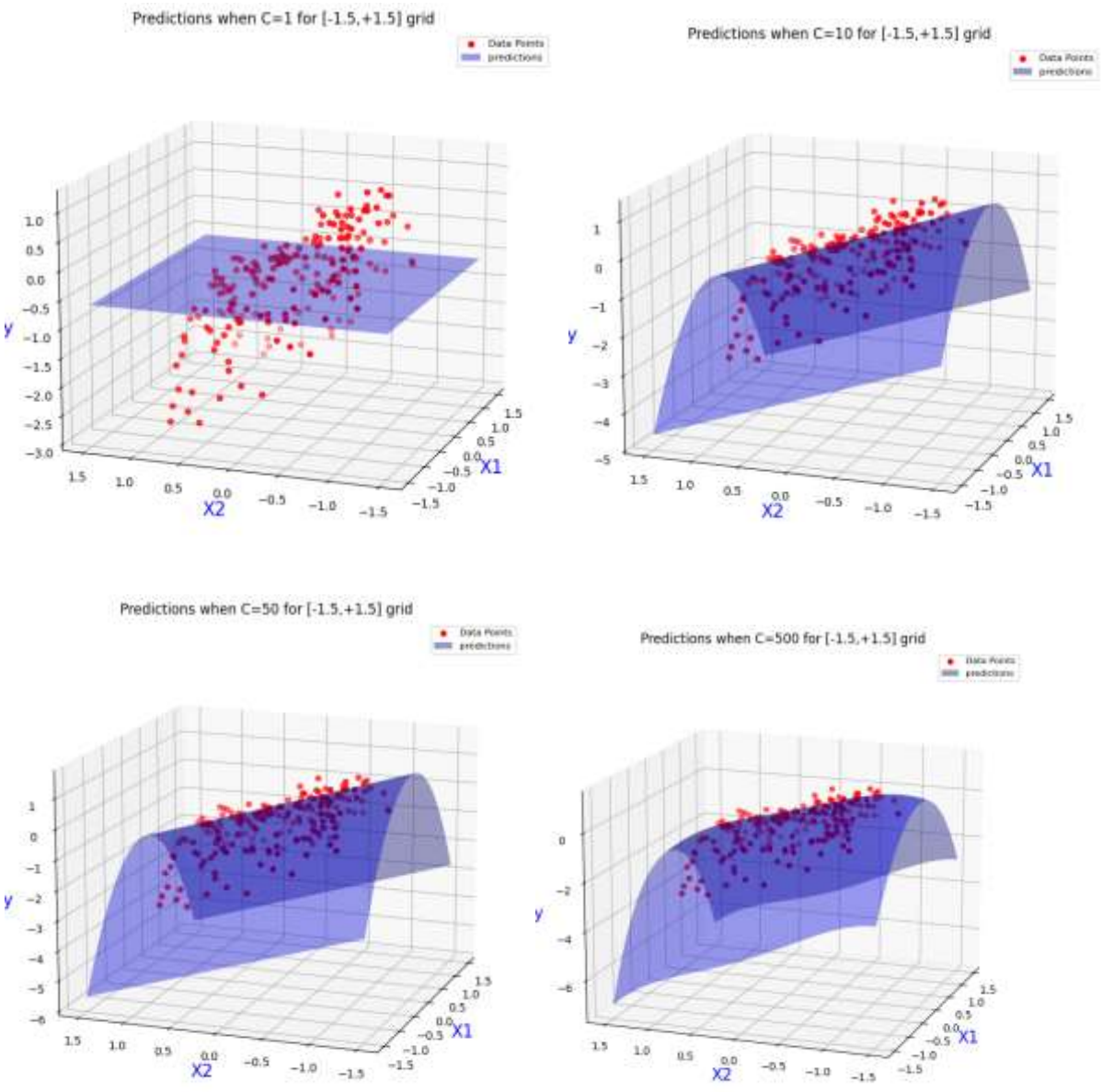
A Lasso regression model was trained with all the features using multiple increasing values of C.
In below table you can see the model intercept and coefficients for different C values.(1 → 1000)

C	Alpha	Intercept	Coefficient
1	0.5	-0.6313	[0. 0. -0. -0. 0. 0. 0. -0. 0. -0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0.]
10	0.05	-0.1780	[0. 0. -0.85714462 -1.4170634 0. 0. 0. -0. 0. -0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0.]
50	0.01	-0.0372	[0. 0. -0.96149013 -1.848706 0. 0. 0. -0. 0. -0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0.]
500	0.001	-0.0182	[0. -0. -0.9691546 -1.89552734 -0. 0.12898278 -0. -0. 0.03225816 -0. -0.09313222 0.0236682 0.11899141 0. -0.20014766 0. -0.06937239 -0. 0. 0. -0.00635412]
1000	0.0005	-0.0228	[0. -0.01891534 -0.95286845 -1.94542898 -0.07219426 0.25477972 -0. -0. 0.10826063 -0. -0.07691299 0.13919468 0.23068882 0. -0.37483672 0.02972276 -0.2609109 -0.09250458 0.26049379 -0. -0.08517118]

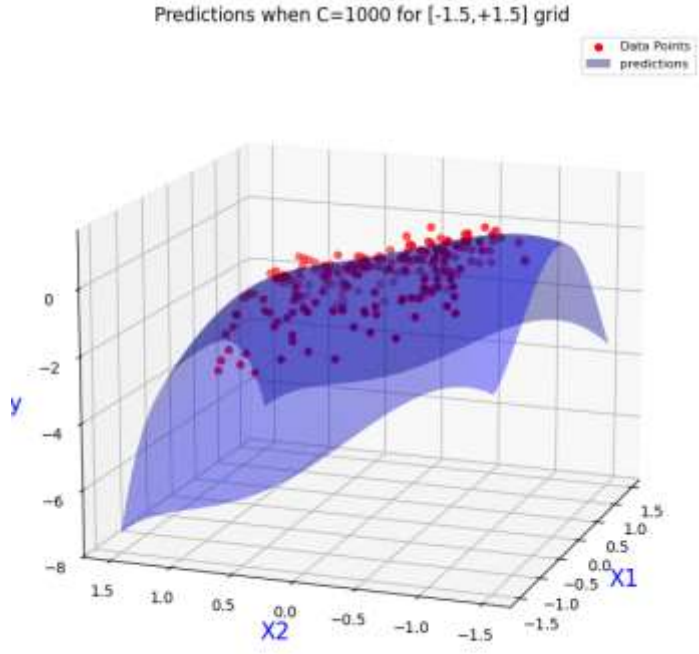
Above table shows values for respective value of C,
When we increase C, alpha is reduced, indicating decrease in regularization, thus the intercept has also moved towards zero trying to fit more data.
When C=1, all model coefficients are zero, meaning model does not consider our training data for making predictions, highly underfit.
As we increase C, we can see that coefficients start turning non-zero one by one, indicating that model starts fitting training data.
For C=1000, most of the features are non-zero. If we map the coefficients with features listed above, x2 (coefficient -0.95286845) and x1^2 (coefficient -1.94542898) will have major impact on predictions.

i(c)

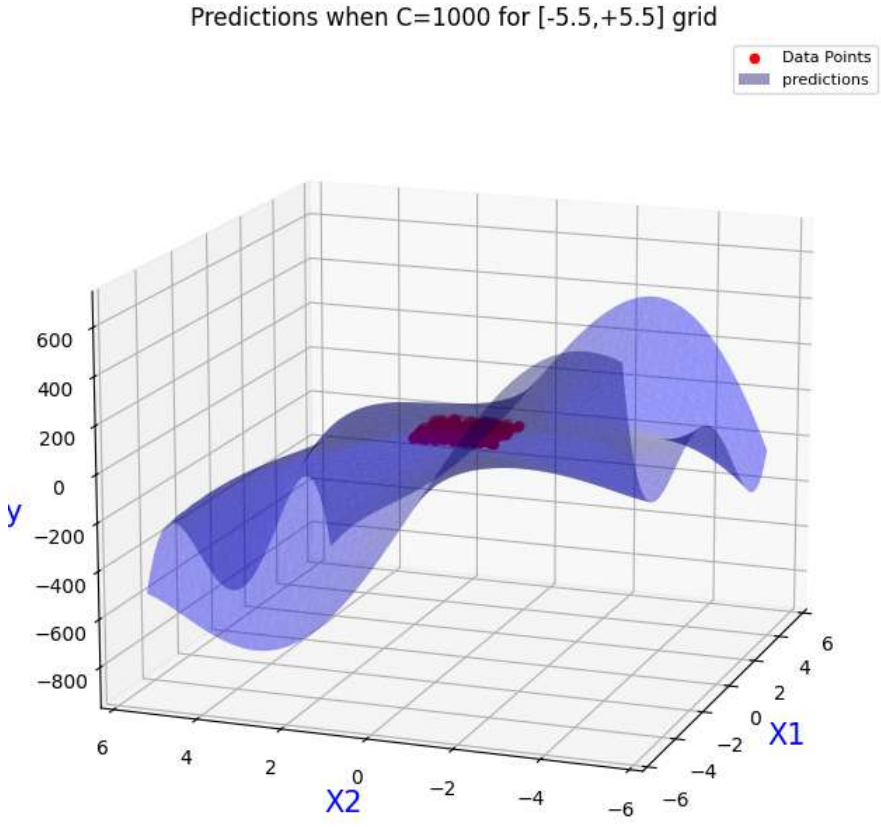
Each of the models was used for making predictions on the target variable. A smaller [-1.5, +1.5] grid (also used [-5.5, +5.5] grid later) was used to make the predictions, making it easier to interpret the fit of the plotted plane to the training data. Below you can see gird plotted against the training data. The grid was plotted using plot_trisurf() with low (0.4) value of alpha for opaque visibility of the plane/curve.



From first graph when C=1, the predictions grid is a simple plane which does not really fit our data, this is because the coefficients for this model are all zero as seen in i(b) table.As you increase C value, the prediction grid starts to fit our training data (as coefficients become non-zero) and the grid changes from a simple plane to a complex curve fitting the data better.



It is also important to note that, as we increase the grid range (-5.5, +5.5) we get the below graph. This is overfitting, for higher values of C (low regularization) the model tries to fit the training data as closely as possible adding noise and thus giving unpredictable predictions for unseen data as shown in the wobbly figure below.



i(d)

Underfitting is when a machine learning model fails to fit the training data, the model is too simple to find any pattern in the dataset. An underfit model gives poor performance not only for new data points but also for the points on which the model was trained on. For example, in our polynomial model when C value is 1, the model has all coefficients as zero i(b), the model is underfit and thus the predictions are a simple plane as shown in i(c).

Overfitting is when a machine learning model is accurate in predicting trained data but has poor performance for new datapoints. Instead of learning the patterns in training data, the model tries to memorize the datapoints which results in highly inaccurate predictions for new points. Using complex models to predict simple patterns in data is one reason for overfitting. For example, in our polynomial model when C value is 1000, most model coefficients are non-zero i(b), the prediction curve perfectly fits in the range of our training data but show unpredictable wavy behaviour outside data range, i(c) last graph.

Thus, C value can be used as a knob to manage a trade-off between underfitting and overfitting thus giving us a model having both, training data fit and good predictions for unseen data.

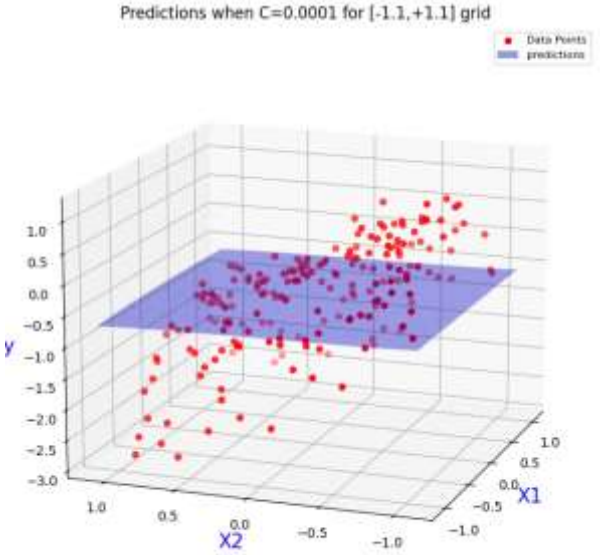
i(e)

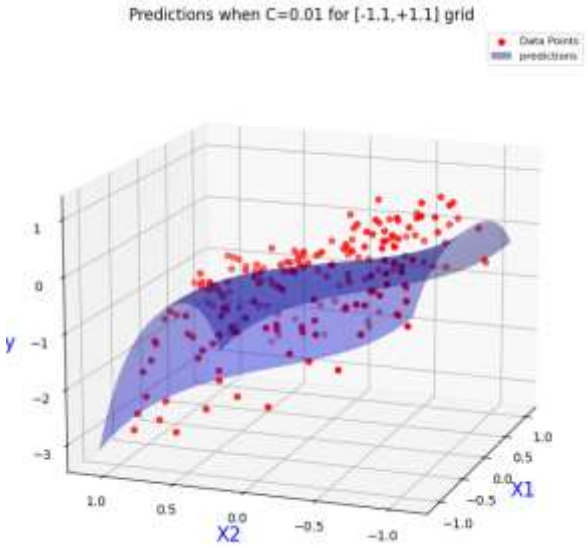
Similar to i(b),
A Ridge regression model was trained with all the polynomial features using multiple increasing values of C.
In below table you can see the model intercept and coefficients for different C values. (0.001 → 100)

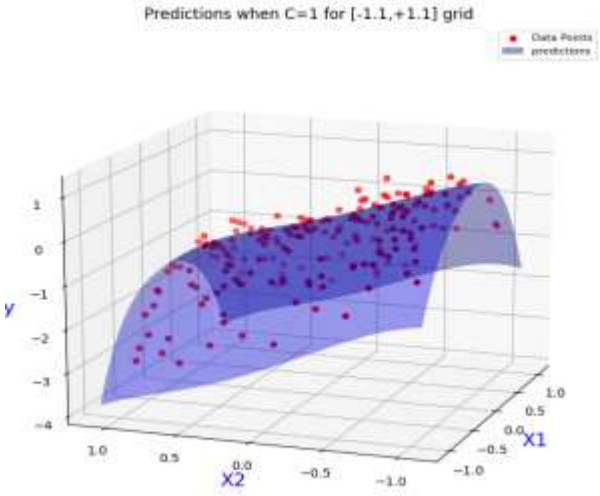
C	Alpha	Intercept	Coefficient
0.0001	5000	-0.6276	[0.00000000e+00 3.04995148e-05 -1.37163490e-02 -7.40392004e-03 1.15291290e-03 1.37639222e-04 2.49702547e-04 -4.77104408e-03 3.07700962e-04 -8.09633141e-03 -6.47344241e-03 8.57243443e-04 -2.33730877e-03 6.39592165e-04 -6.33080493e-05 2.25574236e-04 -3.09913687e-03 5.03808385e-04 -2.79859894e-03 2.91780828e-04 -5.65654025e-03]
0.01	50	-0.4145	[0. -0.00094255 -0.43710487 -0.43065269 0.02607804 0.01503918 0.00125782 -0.1284251 0.01392136 -0.21890857 -0.36537124 0.02026385 -0.12641612 0.01368555 -0.0058528 -0.00154583 -0.08473919 0.01781533 -0.05774855 0.00960031 -0.13290861]
1	0.5	-0.0748	[0. -0.01350432 -0.9007889 -1.53337406 -0.080763 0.24389966 -0.07057241 -0.09138693 0.20079798 -0.0931843 -0.4678648 0.1352151 0.08436467 0.00848853 -0.31014217 0.11893367 -0.1604258 -0.18978473 0.29707521 -0.06021872 -0.05668562]
10	0.05	-0.0357	[0. -0.03064919 -0.94521886 -1.91058389 -0.17642624 0.36153251 -0.25788181 -0.08286721 0.47254004 0.03229393 -0.15088387 0.25749248 0.2958294 0.03824389 -0.50726052 0.36314013 -0.33069679 -0.4590845 0.51061322 -0.13225401 -0.18200391]

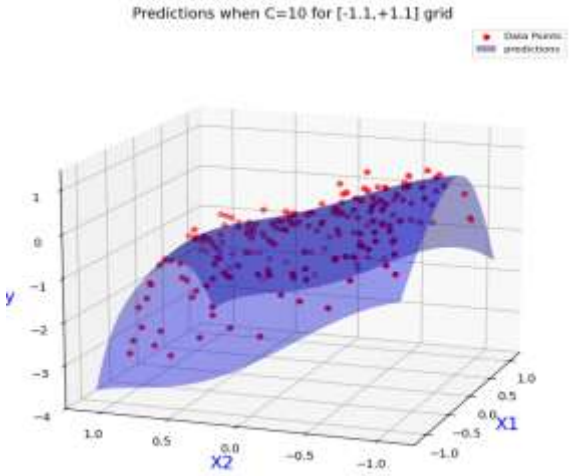
200	0.0025	-0.0288	[0. -0.02708787 -0.96044813 -1.98166169 -0.19393885 0.38376634 -0.34671087 -0.06780571 0.55302279 0.08003518 -0.08965833 0.27938757 0.33446629 0.04328748 -0.54361362 0.46270808 -0.37599132 -0.53614404 0.54787102 -0.15862009 -0.22622185]

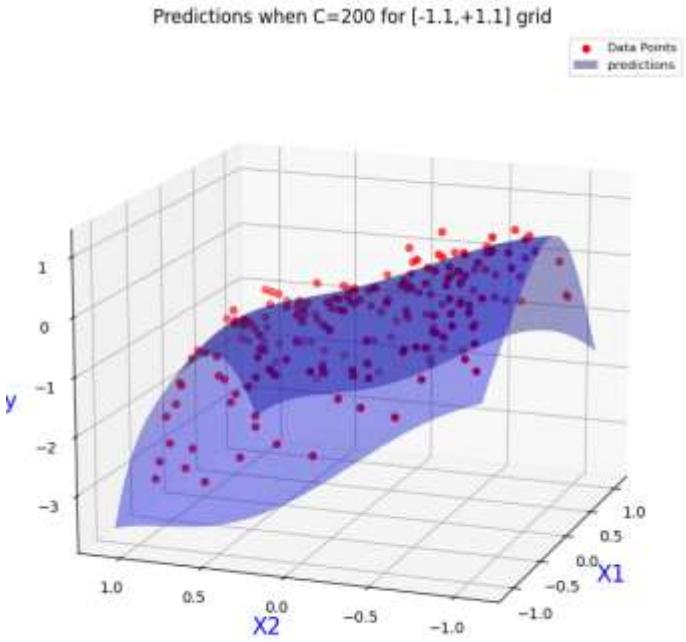
The coefficients for lowest value of C (0.0001) are all almost nearing zero. This indicates underfitting. Unlike Lasso regularization, even for small C the **coefficients are not zero**, but nearing zero. Ridge refression tries to include all the parameters, but decreases their magnitudes.
As C value increases, alpha decreases, decreasing regularization, and thus the coefficients start getting bigger as the model tries to fit training data. For higher values of C the coefficients are much bigger, probably indicating overfitting.







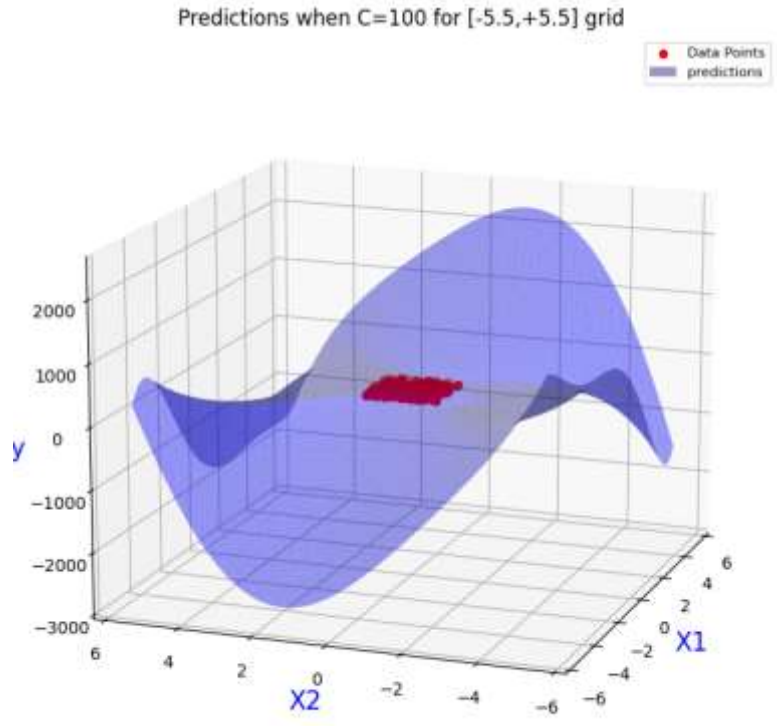




From above graphs for ridge regression, when C=0.0001, the predictions grid is a simple plane which does not really fit our data, this is because the coefficients for this model are very close to zero as seen in table.

As you increase C value, the prediction grid starts to fit our training data (as coefficients increase in magnitude) and the grid changes from a simple plane to a complex curve fitting the data better.

It is also important to note that, as we increase the grid range (-5.5, +5.5) we get the below graph. This is overfitting, for higher values of C (low regularization) the model tries to fit the training data as closely as possible adding noise and thus giving unpredictable predictions for unseen data.

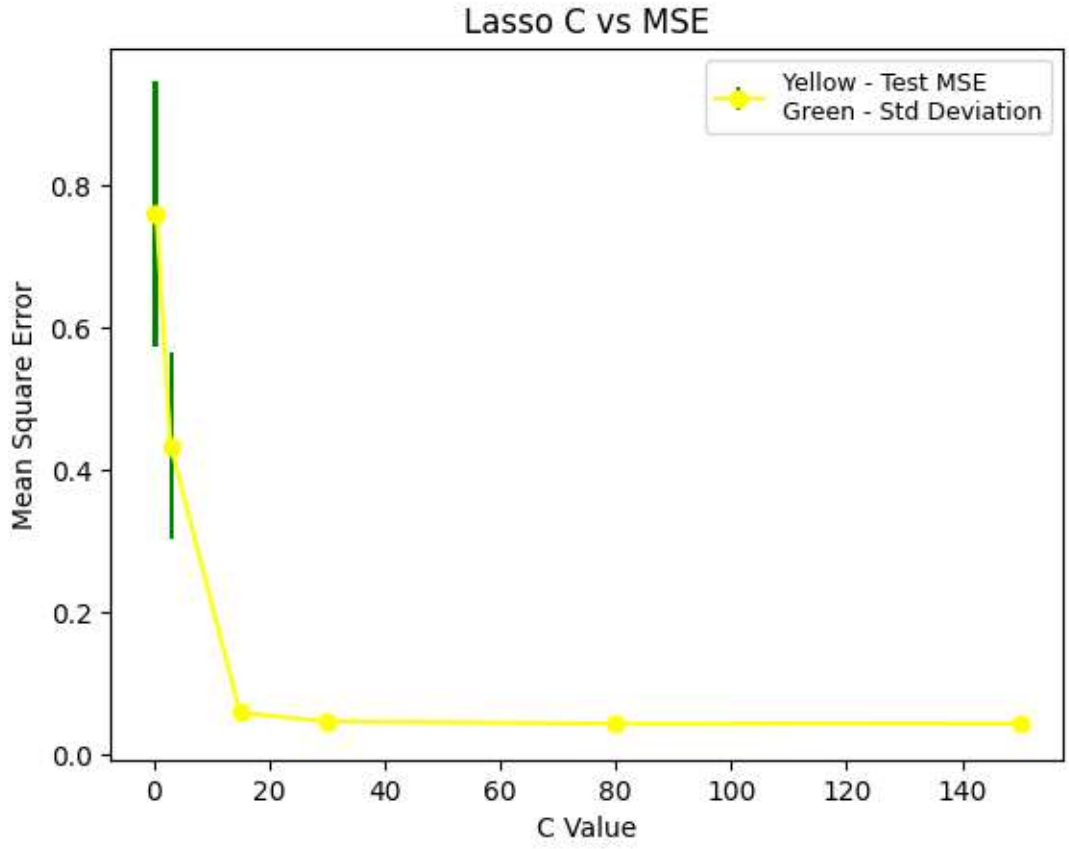


ii(a)

The Lasso regression model was trained using 5-fold cross validation for different values of C. For each value of C, the model was split in 5 parts. 1 Fold = 4 parts for training and 1 part for testing. This was repeated until each part was used as a testing part.
Prediction error was measured using the cost function used for training, mean squared error (MSE).
Started with C = 0.1 and kept increasing C approximately as x5 the previous C.
C value was increased slowly till 30 to find the right value where both mean square error and standard deviation come closer to zero.
After 30 as there was not much change in these values, C value was increased significantly to 80 and then to 150. Mean square error and standard deviation for testing was found.
Below is the data for all values of C used.

C	MSE	Std D
0.1	0.758	0.186
0.5	0.758	0.186
3	0.433	0.131
15	0.058	0.009
30	**0.046**	**0.003**
80	0.043	0.004
150	0.043	0.005

For each C the mean MSE and deviation for all folds was calculated and stored in temp variable. This was plotted using matplotlib errorbar function and below graph was obtained.
Yellow line represents the mean square error, green line represents standard deviation for particular value of C.



ii(b)

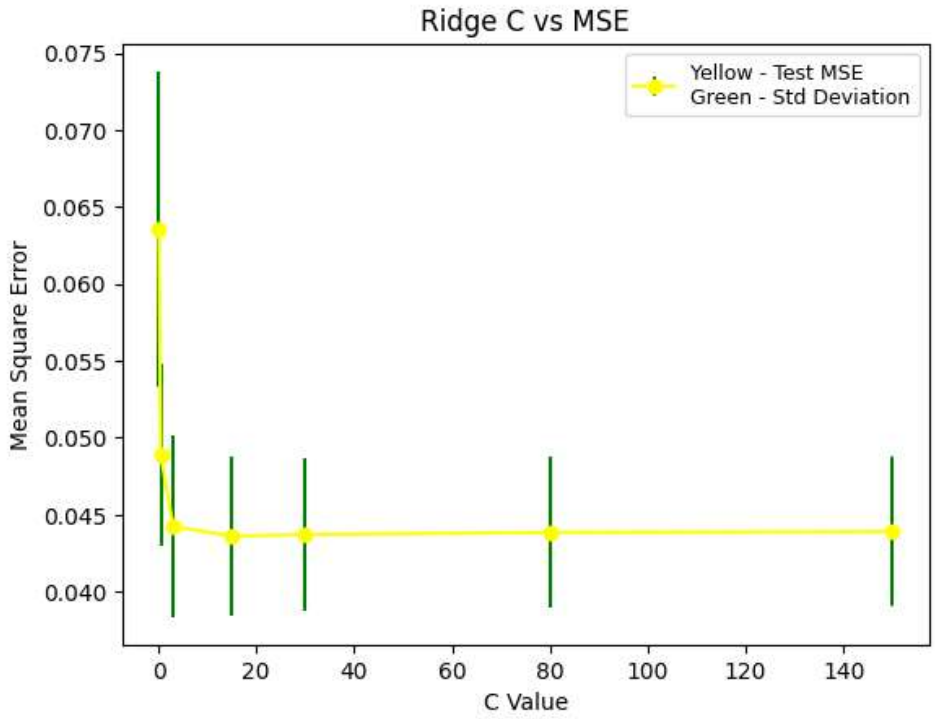
For lower C values, till C is 3, error is high (around 0.7) and standard deviation is also high (around 0,2). Thus, these values of C are not recommended as they do not fit our data correctly, indicating underfitting.
As C is increase, both error and deviation start decreasing.
For C as 30, the mean square error (0.05849) and standard deviation (0.00909) both are very low. This model is a good fit for our data.
Further the value of C is increased rapidly, but there is minimal change/decrease in mean square error and standard deviation.

Thus, value of C=30 seems to be the most optimal.

ii(c)

Using the same 5-fold technique used in ii(b), a ridge regression model was used for cross validation across different values of C. Below table show the findings.

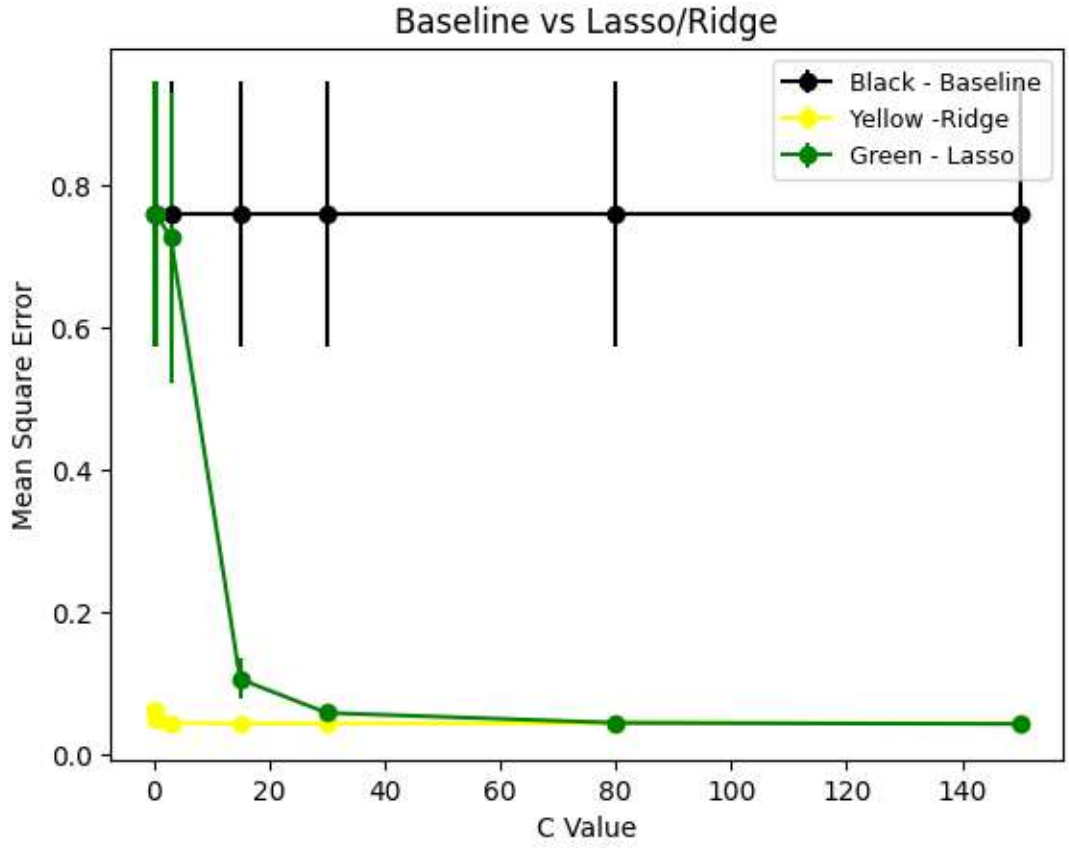
C	MSE	Std D
0.1	0.0635	0.0102
0.5	0.0488	0.0059
3	**0.0442**	**0.0059**
15	0.0436	0.0051
30	0.0437	0.0049
80	0.0438	0.0048
150	0.0438	0.0048



From the above graph we can see that for lower values of C the mean square error is high.
As C increases the error and deviation decrease as model fits the data better.
After C=3, there no significant decrease in error and deviation value.
Thus, C=3 seems the most optimal value for Ridge regression.

<u>Baseline Model:</u>

A baseline predictor was built using DummyRegressor library from sklearn dummy.
The dummy model was trained on the training data using mean strategy and 5-fold cross validation was performed.
Below graph shows the baseline model mean square error and standard deviation as compared to the Lasso and Ridge regression models.



We can see that error and deviation for baseline is much higher than that of the other two models. Thus, our Lasso and Ridge regression models have done a much better job in predicting data points compared to baseline.

Appendix:

```
# Imports and Dataset - #id:21--42—21
import pandas as pd
import numpy as np
dataframe = pd.read_csv("week3.csv", header=None)
print(dataframe.head())
X1=dataframe.iloc[:,0]
X2=dataframe.iloc[:,1]
X = np.column_stack((X1,X2))
y = dataframe.iloc[:,2]


#i(a)

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# figure = plt.figure(figsize=(10,8), facecolor='lightgrey')
figure = plt.figure(figsize=(10,8))
axes = figure.add_subplot(111,projection='3d')
axes_color='blue'
axes_size=15
axes.scatter(X1,X2,y, s=20 , label='Data Points', color='red')
axes.set_xlabel("X1",color=axes_color, fontsize=axes_size)
axes.set_ylabel("X2",color=axes_color,fontsize=axes_size)
axes.set_zlabel("y",color=axes_color,fontsize=axes_size)
axes.view_init(15,200)
axes.set_title("3D Plot for Dataset")
plt.legend(fontsize=8)
plt.show()


#i(b)

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
poly_5 = PolynomialFeatures(degree=5)
X_poly_5= poly_5.fit_transform(X)
C=[1,10,500,1000]
print("* Lasso *")
for c in C:
  alpha = (1/(2*c))
  lasso = Lasso(alpha=alpha)
  lasso.fit(X_poly_5,y)
  print("C = " , c)
  print("alpha = " ,alpha )
  print("Intercept = " , lasso.intercept_)
  print("Coefficients = " , lasso.coef_)
  print("Score = " , lasso.score(X_poly_5,y))
  print("----------------------------------------------------------------------------------------------------")


# i(c)

Xtest = []
grid = np.linspace(-1.5,+1.5)
for i in grid:
  for j in grid:
     Xtest.append([i,j])
     Xtest = np.array(Xtest)
```

```
Xtest_poly_5 = poly_5.fit_transform(Xtest)
def predictLasso(C,range):
  for c in C:
    alpha = (1/(2*c))
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_poly_5,y)
    predictions=lasso.predict(Xtest_poly_5)
    figure = plt.figure(figsize=(10,8))
    axes = figure.add_subplot(111,projection='3d')
    axes_color='blue'
    axes_size=15
    axes.scatter(X1,X2,y, s=20 , label='Data Points', color='red')
    axes.set_xlabel("X1",color=axes_color, fontsize=axes_size)
    axes.set_ylabel("X2",color=axes_color,fontsize=axes_size)
    axes.set_zlabel("y",color=axes_color,fontsize=axes_size)
    axes.view_init(15,200)
    axes.set_title(f"Predictions when C={c} for [-{range},+{range}] grid")
    axes.plot_trisurf(Xtest[:,0],Xtest[:,1], predictions, label='predictions',color='blue' , alpha=0.4)
    plt.legend(fontsize=8)
    plt.show()
predictLasso(C,1.5)


# i(c)
Xtest = []
grid = np.linspace(-5.5,+5.5)
for i in grid:
  for j in grid:
     Xtest.append([i,j])
Xtest = np.array(Xtest)
Xtest_poly_5 = poly_5.fit_transform(Xtest)
predictLasso([1000],5.5)


# i(e)
from sklearn.linear_model import Ridge
C_ridge=[0.0001, 0.01,1,10,200]
print("* Ridge *")
for c in C_ridge:
  alpha = (1/(2*c))
  ridge = Ridge(alpha=alpha)
  ridge.fit(X_poly_5,y)
  print("C = " , c)
  print("alpha = " ,alpha )
  print("Intercept = " , ridge.intercept_)
  print("Coefficients = " , ridge.coef_)
  print("Score = " , ridge.score(X_poly_5,y))
  print("-------------------------------------------------------------------------------------------------------")
Xtest_ridge = []
grid_size = 1.1
grid = np.linspace(-grid_size,+grid_size)
for i in grid:
  for j in grid:
    Xtest_ridge.append([i,j])
```

```
Xtest_ridge = np.array(Xtest_ridge)
Xtest_poly_5 = poly_5.transform(Xtest_ridge)
def predictRidge(C,range):
   for c in C:
    alpha = (1/(2*c))
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_poly_5,y)
    predictions=ridge.predict(Xtest_poly_5)
    figure = plt.figure(figsize=(10,8))
    axes = figure.add_subplot(111,projection='3d')
    axes_color='blue'
    axes_size=15
    axes.scatter(X1,X2,y, s=20 , label='Data Points', color='red')
    axes.set_xlabel("X1",color=axes_color, fontsize=axes_size)
    axes.set_ylabel("X2",color=axes_color,fontsize=axes_size)
    axes.set_zlabel("y",color=axes_color,fontsize=axes_size)
    axes.view_init(15,200)
    axes.set_title(f"Predictions when C={c} for [-{range},+{range}] grid")
    axes.plot_trisurf(Xtest_ridge[:,0],Xtest_ridge[:,1], predictions, label='predictions',color='blue' ,
alpha=0.4)
    plt.legend(fontsize=8)
    plt.show()

predictRidge(C_ridge,grid_size)

# i(e)

Xtest_ridge = []
grid_size = 5.5
grid = np.linspace(-grid_size,+grid_size)
for i in grid:
  for j in grid:
     Xtest_ridge.append([i,j])

Xtest_ridge = np.array(Xtest_ridge)
Xtest_poly_5 = poly_5.fit_transform(Xtest_ridge)

predictRidge([100],grid_size)


# # ii(a)(b)

from sklearn.model_selection import KFold

from sklearn.metrics import mean_squared_error


mean_err=[]

std_err=[]

C_Lasso = [0.1, 0.5, 3, 15, 30, 80,150]


for c_lasso in C_Lasso:

  lasso_model = Lasso(alpha=1/(2*c_lasso))

  temp=[]


  k_fold = KFold(n_splits=5)
```

```
 for train,test in k_fold.split(X_poly_5):
   lasso_model.fit(X_poly_5[train], y[train])
   pred = lasso_model.predict(X_poly_5[test])
   temp.append(mean_squared_error(y[test],pred))

 mean_err.append(float(np.array(temp).mean()))
 std_err.append(float(np.array(temp).std()))

 print("C -> " , c_lasso)
 print("Mean square error ->",float(np.array(temp).mean()) )
 print("Std Deviation ->",float(np.array(temp).std()) )
 print("_____")


plt.errorbar(C_Lasso,mean_err,yerr=std_err,ecolor='green',fmt='-o' , label="Yellow - Test
MSE\nGreen - Std Deviation",color="yellow")
plt.xlabel("C Value")
plt.ylabel("Mean Square Error")
plt.legend(fontsize=9)
plt.title("Lasso C vs MSE")
plt.show()



# ii(c)


mean_err_R=[];
std_err_R=[]
C_Ridge = [0.1, 0.5, 3, 15, 30, 80,150]

for c_ridge in C_Ridge:
 ridge_model = Ridge(alpha=1/(2*c_ridge))
 temp=[]
 k_fold = KFold(n_splits=5)

 for train,test in k_fold.split(X_poly_5):
   ridge_model.fit(X_poly_5[train], y[train])
   pred = ridge_model.predict(X_poly_5[test])
   temp.append(mean_squared_error(y[test],pred))
```

```
  mean_err_R.append(float(np.array(temp).mean()))
  std_err_R.append(float(np.array(temp).std()))

  print("C -> " , c_ridge)
  print("Mean square error ->",float(np.array(temp).mean()) )
  print("Std Deviation ->",float(np.array(temp).std()) )
  print("_____")


plt.errorbar(C_Ridge,mean_err_R,yerr=std_err_R,ecolor='green',fmt='-o' , label="Yellow - Test MSE\nGreen - Std Deviation",color="yellow")
plt.xlabel("C Value")
plt.ylabel("Mean Square Error")
plt.legend(fontsize=9)
plt.title("Ridge C vs MSE")
plt.show()

# Baseline
from sklearn.dummy import DummyRegressor

mean_err_B=[];
std_err_B=[]

for c_base in C_Ridge:
  base_model = DummyRegressor(strategy='mean')
  temp=[]
  k_fold = KFold(n_splits=5)

  for train,test in k_fold.split(X_poly_5):
    base_model.fit(X_poly_5[train], y[train])
    pred = base_model.predict(X_poly_5[test])
    temp.append(mean_squared_error(y[test],pred))

  mean_err_B.append(float(np.array(temp).mean()))
  std_err_B.append(float(np.array(temp).std()))

plt.errorbar(C_Ridge,mean_err_B,yerr=std_err_B,ecolor='black',fmt='-o' , label="Black - Baseline",color="black")
```

```
plt.errorbar(C_Ridge,mean_err_R,yerr=std_err_R,ecolor='yellow',fmt='-o' , label="Yellow - Ridge",color="yellow")

plt.errorbar(C_Lasso,mean_err,yerr=std_err,ecolor='green',fmt='-o' , label="Green - Lasso",color="green")


plt.xlabel("C Value")

plt.ylabel("Mean Square Error")

plt.legend(fontsize=9)

plt.title("Baseline vs Lasso/Ridge")

plt.show()
```