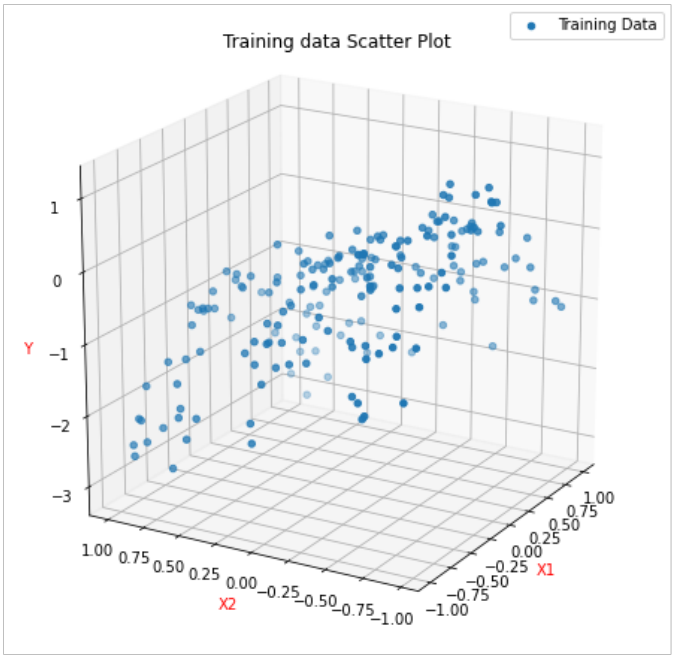
Question i:

(i.a) Answer:

Plotting the data in 3d plot. Placing the feature 1(X1) on X-Axis, feature 2(X2) on Y-Axis and output variable(y) on Z-Axis. Since the data points are not in a straight line or flat, so training data lies on a curve. Training points' scatter plot and the training points converted as a surface of the training points are shown below.



(i.b) Answer:

As specified, PolynomialFeatures function is used to generate a 5th degree equation. This me thod generates all the possible combinations for the given 2 features. Since it is a 5th degree equation there are 21 different features (['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','X 1^2 X2^3','X1 X2^4','X2^5']. A Lasso Regression Model is trained with the obtained 21 featur es with the C value [0.1, 1, 10, 50, 100,1000]. The below are the features and intercepts for each C value.

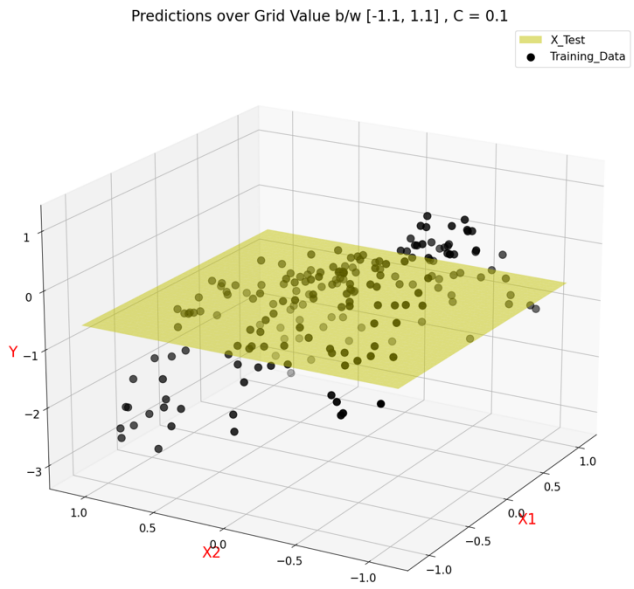
C-Value	Alpha Value	Feature Coefficients	Intercept	Score (R-Squared)
0.1	10	0. -0. -0. -0. 0. -0. -0. -0. 0. -0. -0. 0. -0. 0. -0. -0. -0. 0. -0. 0. -0.	-0.633	0.0
1	1	0. -0. -0. -0. 0. -0. -0. -0. 0. -0. -0. 0. -0. 0. -0. -0. -0. 0 -0. 0. -0.	-0.633	0.0
10	0.1	0. -0. -0.72517176 -0.96503839 -0. -0. -0. -0. -0. -0. -0. -0. -0. 0. -0. -0. -0. -0. -0. -0. -0.	-0.336	0.794
50	0.02	0. -0. -0.92377217 -1.76563578 -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.	-0.078	0.946

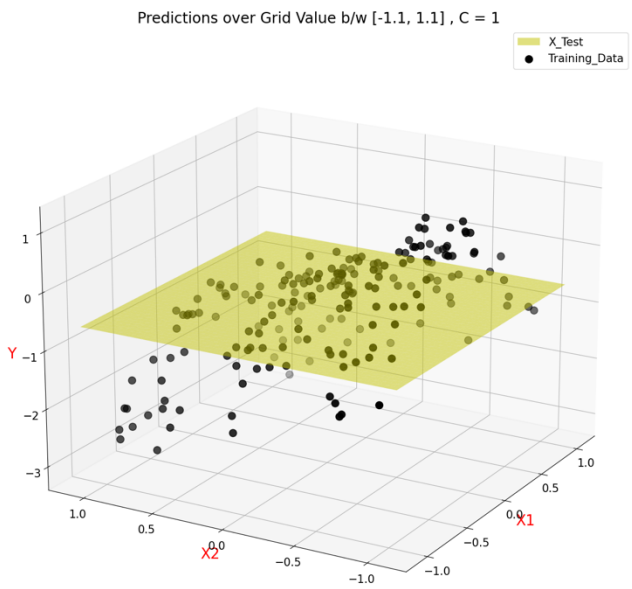
		-0 0. -0. -0. -0. -0.		
100	0.01	0. -0. -0.94855546 -1.85783479 -0. -0. -0. -0. -0. -0. -0.00943706 -0. -0. -0. -0. -0. 0. -0. 0. -0. -0	-0.047	0.951
1000	0.001	0. -0. -0.99135021 -1.78308206 -0.03215026 0.04891166 -0. 0.05994442 -0. -0. -0.18569255 0. -0.03489972 0. -0.07575954 -0.04394047 0. -0.0276597 0. -0. 0.	-0.034	0.954

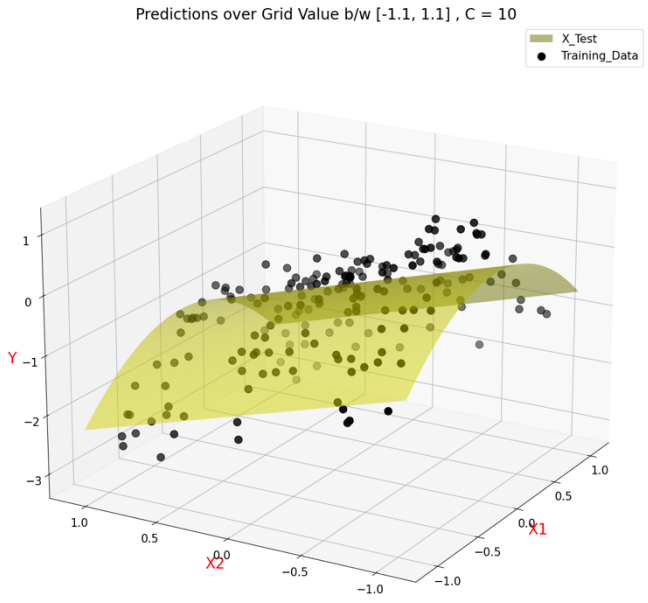
As the C-Value increases, the alpha value reduces as they are inversely proportional. When C is 0.1, all the feature coefficients are 0 and accuracy of the model is 0. When C=10, some of the feature coefficients are non-zero. Ultimately, when C is made to be 1000 almost all the features are non-zero. So, for lower values of C, model tends to set as many as possible parameters to 0. This may cause underfitting of data. But for higher C values, when most of all the features are considered, the model may cause overfitting of the data when there is lesser amount of training data.

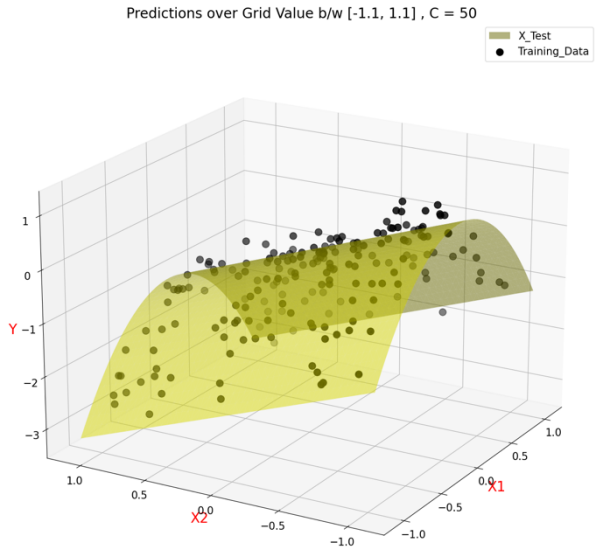
(i.C) Answer:

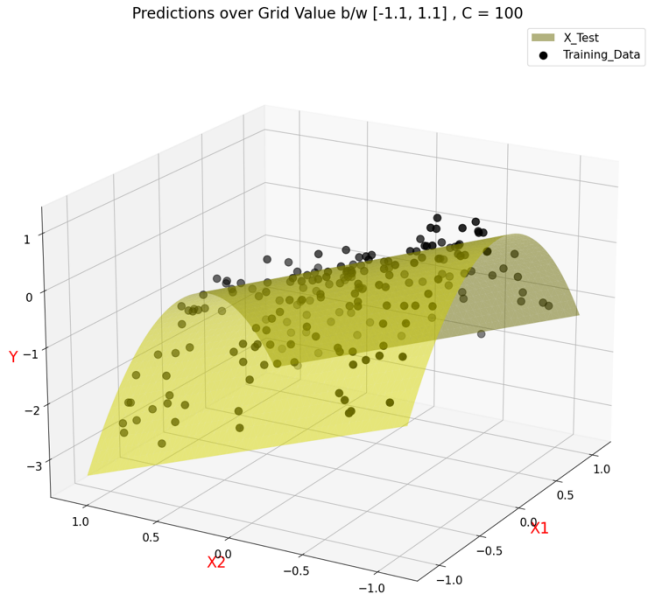
As the training data set mostly lies between -1 and 1, the grid values are also taken between [-1.1,1.1] as taking [-5,5] makes the training points very close to each other and makes the plot difficult to interpret. The scatter plot is used for plotting the training points and plot_trisurf with lower opacity (alpha value) is used to plot the grid values.

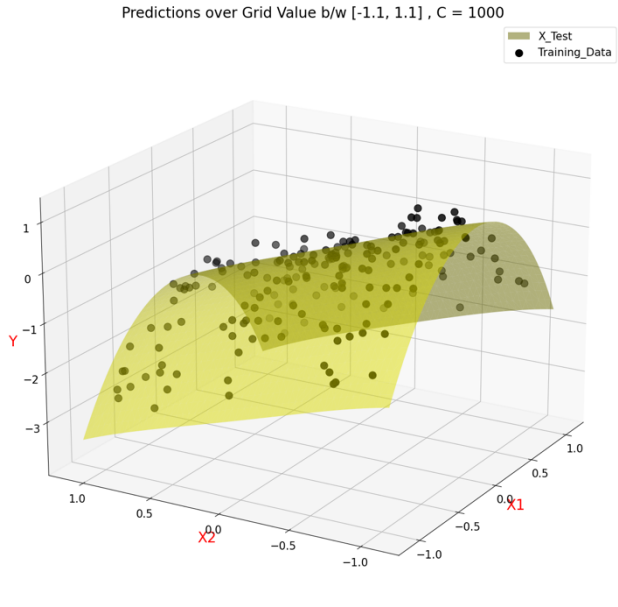












From the above plots we can see that, when the C=1, the plot is a flat plane as all the parameters for the model are zero (ref. i(b)). As the C value is increased, the number of feature coefficients are increased, and the model become more complex. The noise of the model is also increased for an increased value of C.

(i.d) Answer:

Under-Fitting: A machine learning model is said to be under-fitting when it cannot identify the general trend in the data. The model usually neither predicts the training data nor the new data accurately. This can happen when the model is too simple for the training data. For example, if we use a linear model for predicting a quadratic line, the model is underfitted.

Over-Fitting: A machine learning model is said to be over-fitting when it cannot make accurate predictions on the new data (test data). This usually happens when we have high model complexity (Many features) compared to the size of the training dataset.

We compare our models by using the plots and the feature coefficients above and following are the observations.

When C=0.1 and 1, we can see that all the feature coefficients values are zero, so none of them are used to predict the data. From the above we can see that; the plots are a straight plane, and it does not match our training data properly. This is a classic example of underfitting.

When C=1000, we can see that most of the feature coefficients are non-zero and model is trying to form intricate relations for predicting the output which ends up to overfit the model. From the plot, we can see that the surface is more convex and catches more noise which is not desired.

(i.e) Answer:

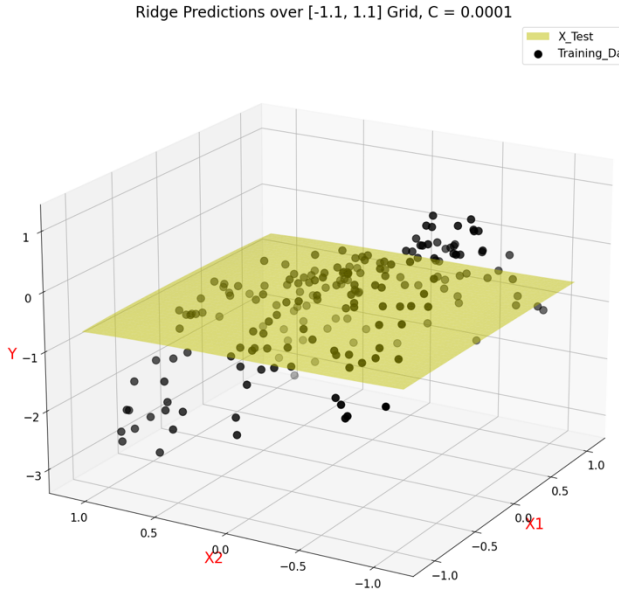
Like the Lasso regression above, PolynomialFeatures is used to create the features up to a power of 5. We selected a C value ranging from 0.0001 to 200 to see the impact of very small C vs very large value of C. Later we train the Ridge regression in like the Lasso Regression above and obtained the following coefficients and scores.

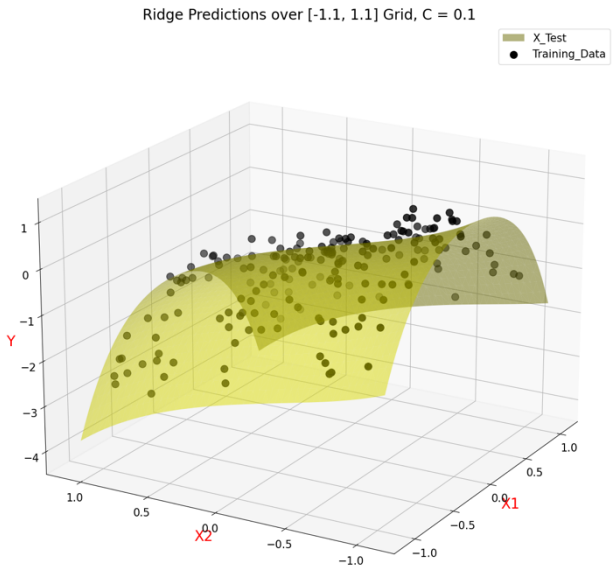
C-Value	Alpha Value	Feature Coefficients	Intercept	Score (R-Squared)
0.0001	5000	0.00000000e+00 -6.88696781e-05 -1.40566797e-02 -8.03484631e-03 4.22948486e-04 -1.67551473e-03 -7.79420010e-05 -5.16258256e-03 2.43241489e-04 -8.53515032e-03 -6.94952219e-03 6.02973037e-05 -3.94486831e-03 5.78073929e-04 -1.67909845e-03 -1.71034689e-04 -3.50445284e-03 1.15307658e-04 -3.30598942e-03 3.10361239e-04 -6.12314736e-03	-0.6278077803821528	0.03165749286518171
0.1	5.0	0. 0.00324933 -0.76159255 -0.99228497 -0.05173174 0.06119372 -0.00126088 -0.05698634 -0.00850075 -0.21487046 -0.70462431 0.01752987 -0.29437762 0.02235597 -0.01581767 -0.03765692 -0.00222661 -0.03145923 0.04250154 -0.01347477 -0.0399763	-0.16407592227769224	0.9397535361278373
1	0.1	0. -0.11824151 -0.89645063 -1.67943021 -0.21520302 0.33492205 0.37644134 0.0967881 0.12665613 -0.4626145 -0.29775907 0.1208166 -0.09094122 0.13983383 -0.3781769 5 -0.29756563 -0.02417576 -0.20028069 0.03540945 -0.03493384 0.40908608]	-0.07305484814666008	0.9554257194764436
10	0.05	0. -0.15575417 -0.86898751 -1.72672815 -0.2282066 0.36552157		0.9576289667716362

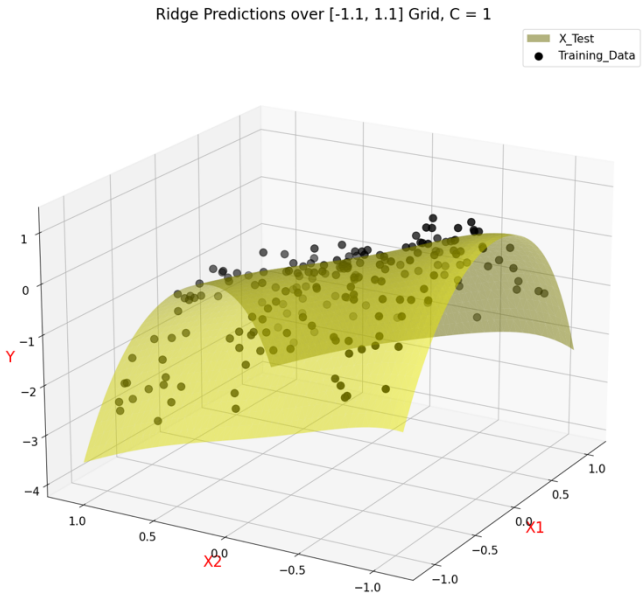
		0.50050252 0.11552576 0.15124031 -0.6018865 -0.25767167 0.1223282 -0.07236999 0.1544083 -0.42257778 -0.38620567 -0.03855569 -0.22845054 0.03067328 -0.04146711 0.53732803	-0.06943834808636595	
50	0.01	0. -0.2075871 -0.8224907 -1.77181418 -0.24341262 0.39723281 0.68238336 0.13383968 0.17044575 -0.81986237 -0.22043877 0.12389018 -0.05286107 0.17111919 -0.46991905 -0.51993409 -0.05594729 -0.26115479 0.03249291 -0.03805203 0.73186875] Intercept: -0.06588188034024056	-0.06588188034024056	0.9578287269316436
200	0.0025	0. -0.22152602 -0.80865937 -1.78133068 -0.24725522 0.40428532 0.73309302 0.13753696 0.17281077 -0.88267917 -0.21281256 0.12435999 -0.04838665 0.1751813 -0.48074675 -0.5577561 -0.06016984 -0.26897191 0.03435161 -0.03465879 0.78715681	-0.06508537944314596	0.9578287269316436

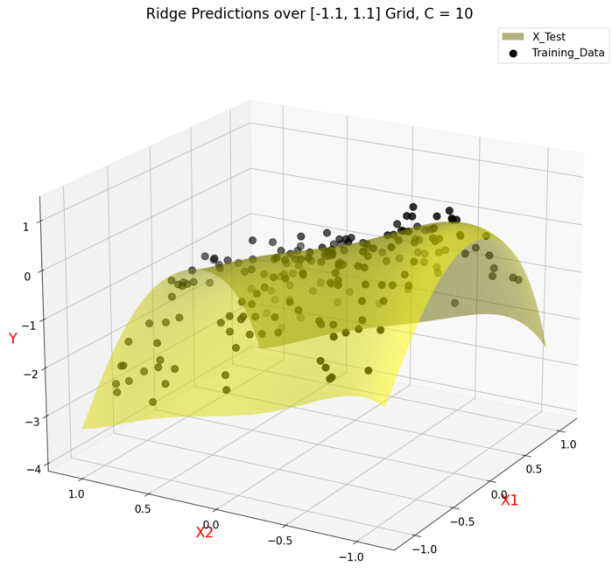
For the ridge model, we can see that most of the features are being used for the model however small the value of C is. And the higher the value of C, there is less impact of the C value on the model coefficients. So, there is not much difference between C=1 and C=200.

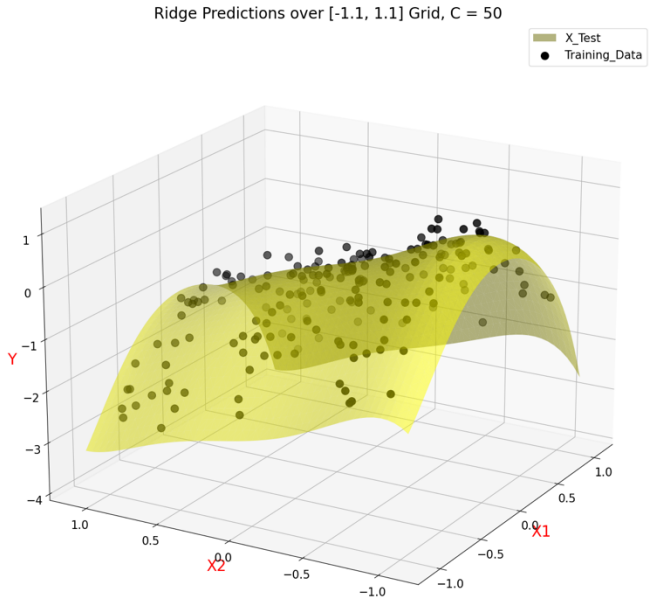
Below are the plots that are obtained for the Ridge Regression when C values range from 0.0001 to 200. The Grid Value is taken between [-1.1,1.1] like the previous Lasso Regression as making it [-5,5] would be hard to interpret the graph. The plotting methods are like the previous question.

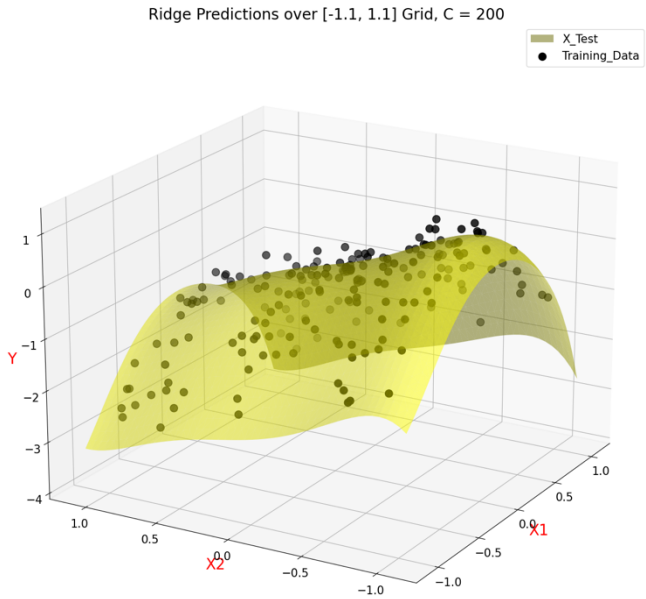












From the above plots, we can see that when C value is 0.0001, the plot is a flat plane as most of the features are close to 0 and do not affect the model much. So, this is a classic case of underfitting as explained in the previous question. As we increase the value of C, the Ridge regression takes lesser account of penalty term. When this happens, the model tries to fit the noise in the data as well and it goes into overfitting.

When we compare, Lasso and Ridge regression from the above two questions, both models gave similar performance when it comes to prediction. When we look at the coefficients, Lasso tries to take the coefficients that are only important and tries to keep the less important terms to zero. Ridge regression tries to take all the coefficients and plays around on impact for each coefficient. From the above, we can say that Lasso Regression is more useful when we have high number of features when we select a reasonable value of C. We have 21 features in this problem, so we can say that Lasso Regression is more useful in the above.

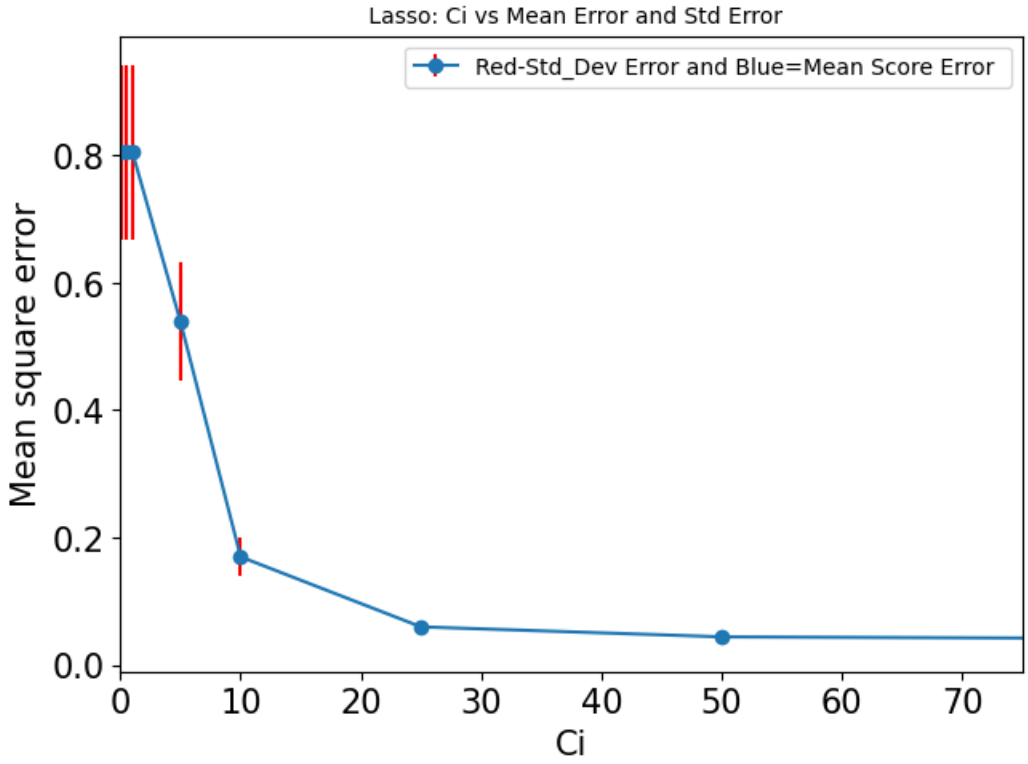
Question ii:

(ii.a) Answer:

Using 5-fold Cross-Validation, we trained the Lasso Regression with polynomial features (5) with the following values of C. As a rule of thumb, we started the C value at 0.1 and multiplied it by 5 until 100 so we can quickly scan across a large range. Over 100, the penalty will be very small that it becomes nearly equal to linear regression. In 5-fold Cross Validation, the training data set is divided into 5 parts. 1 part for the validation and the 4 parts for training. So, there would be 5 iterations along with the number of iterations for the values of C.

For each value of Ci:
For each iteration of Cross Validation, we train the regression model on 4 parts of the training data and get the mean squared error score based on the 1 part of training data (which is validation data) and store it in a temp list. We then take the mean and standard deviation of the mean squared errors obtained over 5 iterations and stored in the corresponding mean_error and std_error lists.

So, for each value of C in Ci there would be a corresponding mean_error and std_error value. Using the error bar function in matplotlib we plot the mean error and std error vs values of C in Ci.



Mean= [0.8054340300183614, 0.8054340300183614, 0.8054340300183614, 0.5389051716751627, 0.16961134779298043, **0.059650251905207854**, 0.043979774006332836, 0.040188941201302286]

Std_Error= [0.1373060452206968, 0.1373060452206968, 0.1373060452206968, 0.0933625804338741, 0.029989855220849595, **0.006817673700891519**, 0.004930095762668943, 0.005509454661490701]

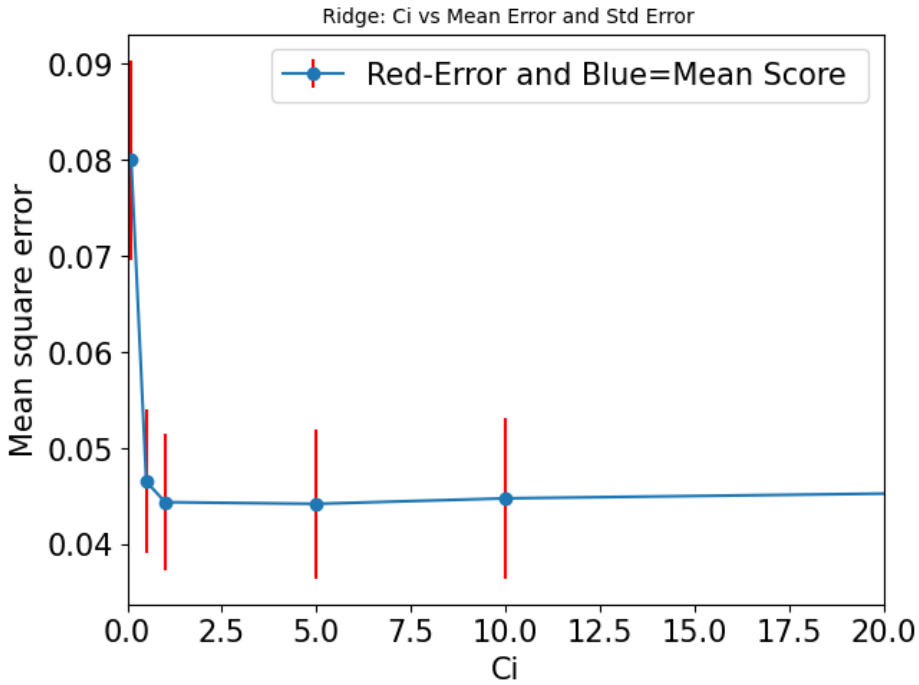
(ii.b) Answer:

From the plot above, we can see that for a low value of C(C=0.1) the mean of the error is 0.8 which is quite high because all the coefficients are set to 0. When the C value is very large, the penalty is very small, and it becomes similar to Linear Regression. So, from the plot above we can infer values above 25(C>=25) yields the least error. So, to avoid overfitting of data, we use the least possible value of C, which is 25 in this case. So, C value of 25 is the most optimal value.

Mean_Error at C=25 --- **0.059650251905207854**
StdDev_Error at C=25 --- **0.006817673700891519**

(ii.c) Answer:

Using the similar method above, we plot the graph of values of Ci vs mean of error and StdDev of error.



Mean= [0.08002735506248294, 0.046505135775350044, **0.044360056728321294**, 0.04418593133367131, 0.04475709118530048, 0.04549987484750961, 0.04595160705191762, 0.046276253251247686]
Std_Error= [0.010392420746177997, 0.007488330850628127, **0.007140638773810939**, 0.007774497104589593, 0.008309199207054412, 0.008923477741634512, 0.009250743688710632, 0.009466602511859799]

From the above plot we can see that, for low values of C(C=0.1), the mean error is highest at 0.09(highest in this plot) as the coefficients are close to 0. When C is very large(C=100) the penalty is negligible, and the model would be similar to a linear regression, and it would be overfitting. So, from the plot above we can infer values above (C>=1) yields the least error. So, to avoid overfitting, we take the least possible value of C, which is 1 in this case. So, the C value of 1 is the most optimal.
Mean_Error at C=1 --- **0.044360056728321294**
StdDev_Error at C=1 --- **0.007140638773810939**

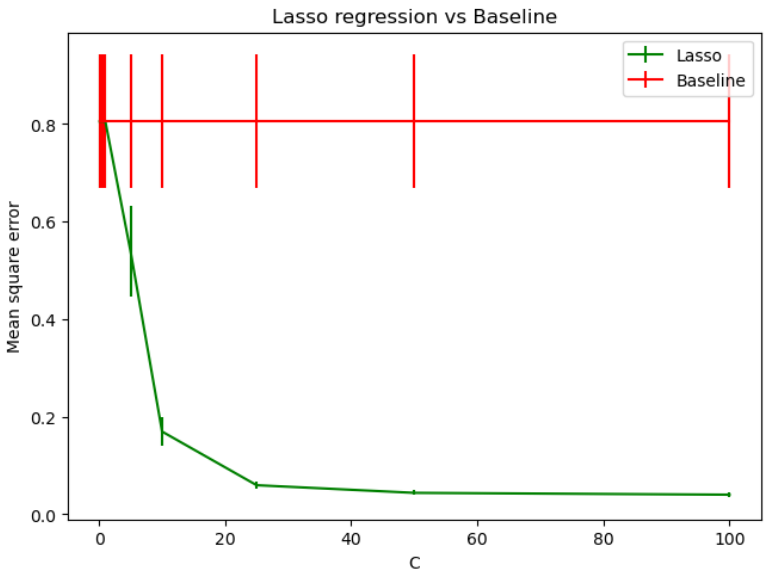
Baseline Prediction:

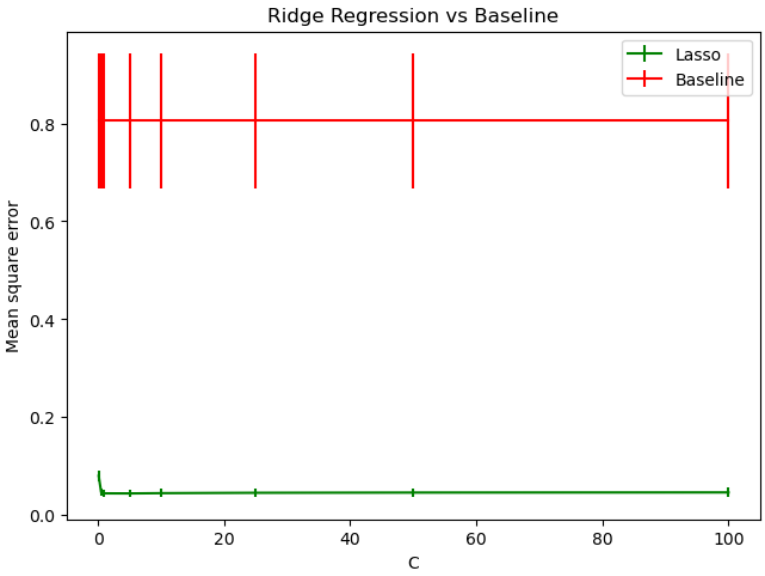
To compare our model to the baseline model, I created a dummy regressor which trains on all the C values. From the below values of Mean Error, we can say that our both models does a better job at predicting the data than the baseline models. So our model is acceptable. The Plots b/w Lasso and Baseline, Ridge and Baseline and also the Mean Error values are given below.

Mean Error of Lasso: [0.8054340300183614, 0.8054340300183614, 0.8054340300183614, 0.5389051716751627, 0.16961134779298043, 0.059650251905207854, 0.043979774006332836, 0.040188941201302286]

Mean Error of Ridge: [0.08002735506248294, 0.046505135775350044, 0.044360056728321294, 0.04418593133367131, 0.04475709118530048, 0.04549987484750961, 0.04595160705191762, 0.046276253251247686]

Mean Error of Baseline: [0.8054340300183614, 0.8054340300183614, 0.8054340300183614, 0.8054340300183614, 0.8054340300183614, 0.8054340300183614, 0.8054340300183614, 0.8054340300183614]





Appendix:

Data Reading:

```
# id:14--28--14
import numpy as np
import pandas as pd
df=pd.read_csv("week 3.csv")
print(df.head())
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]
```

Q(i.a):

```
#i(a)
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111,projection ='3d')
ax.scatter(X[:,0],X[:,1],y,label="Training Data",s=100) #Scatter Plot fot
the Training Data.
ax.set_xlabel('X1',size='20',color='red')
ax.set_ylabel('X2',size='20',color='red')
ax.set_zlabel('Y',size='20',color='red')
plt.rcParams['figure.constrained_layout.use'] = True
plt.title("Training data Scatter Plot",fontsize=20)
ax.view_init(20,210)
ax.legend(fontsize=20)
```

Q(i.b,c):

```
#i(b&c)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=5) #Taking Polynomial Features upto 5
X_poly=poly.fit_transform(X)
# Generate the [-1.1,1.1] grid for predictions since -5,5 makes the plot
difficult to interpret
Xtest = []
grid = np.linspace(-1.1, 1.1)
for i in grid:
    for j in grid:
```

```
        Xtest.append([i, j])
# Grab the powers up to 5 for the grid data.
Xtest = np.array(Xtest)
Xtest_poly = poly.fit_transform(Xtest) #i(b&c)

C=[0.1, 1, 10, 50, 100,1000]

for C_Value in C:
    alpha_val=1/(C_Value)
    linlasso=Lasso(alpha=alpha_val,max_iter=10000).fit(X_poly,y) #Training
the Lasso Regression on the Poly features, with C in the above range
    print("C_Value:", C_Value,"Alpha_val: ",alpha_val,"Lasso Coefficients:
",linlasso.coef_,"Intercept: ",linlasso.intercept_)
    print()
    print("R squared", linlasso.score(X_poly,y))
    print()

    ypred=linlasso.predict(Xtest_poly)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    fig.set_size_inches(12,12)
    ax1=ax.plot_trisurf(Xtest[:, 0], Xtest[:, 1], ypred, color="yellow",
alpha=0.5,label='X_Test') #Plot for the test data
    ax.scatter(X[:, 0], X[:, 1],
y,color='black',label="Training_Data",s=100) #Plot for training dta
    ax1._edgecolors2d = ax1._edgecolor3d
    ax1._facecolors2d = ax1._facecolor3d
    ax.set_xlabel('X1',size='20',color='red')
    ax.set_ylabel('X2',size='20',color='red')
    ax.set_zlabel('Y',size='20',color='red')
    ax.view_init(20,210)
    ax.legend()
    ax.set_title("Predictions over Grid Value b/w [-1.1, 1.1] , C = " +
str(C_Value),fontsize=20)
```

Q(i.e):

```
#i(E)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures

C=[0.0001, 0.1, 1, 10, 50, 200]

for C_Value in C:
    alpha_val=1/(2*C_Value)
    linridge=Ridge(alpha=alpha_val,max_iter=10000).fit(X_poly,y) #Training
Ridge Regression on Poly Features with C value in the above range.
    print("C_Value:", C_Value,"Alpha_val: ",alpha_val,"Ridge Coefficients:
",linridge.coef_,"Intercept: ",linridge.intercept_)
    print()
```

```
    print("R squared", linridge.score(X_poly,y))
    print()

    ypred=linridge.predict(Xtest_poly)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    fig.set_size_inches(12,12)
    ax1=ax.plot_trisurf(Xtest[:, 0], Xtest[:, 1], ypred, color="yellow",
alpha=0.5,label='X_Test') #Plot for Test Data
    ax.scatter(X[:, 0], X[:, 1],
y,color='black',label="Training_Data",s=100)#Plot for the Training Data
    ax1._edgecolors2d = ax1._edgecolor3d
    ax1._facecolors2d = ax1._facecolor3d
    ax.set_xlabel('X1',size='20',color='red')
    ax.set_ylabel('X2',size='20',color='red')
    ax.set_zlabel('Y',size='20',color='red')
    ax.view_init(20,210)
    ax.legend()
    ax.set_title("Ridge Predictions over [-1.1, 1.1] Grid, C = " +
str(C_Value),fontsize=20)
```

Q(ii.a,b):

```
#ii.a,b
from sklearn.model_selection import KFold
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
mean_error=[]; std_error=[]
Ci_range = [0.1, 0.5, 1, 5, 10,25, 50, 100]
for Ci in Ci_range:
    model = Lasso(alpha=1/(Ci)) #Defining the model with the C values
above.
    temp=[]

    kf = KFold(n_splits=5)
    for train, test in kf.split(X_poly):
        model.fit(X_poly[train], y[train]) #Training the Lasso Model with 5
fold Crossvalidation
        ypred = model.predict(X_poly[test])

        temp.append(mean_squared_error(y[test],ypred)) #Calculating Mean
Squared Error
    mean_error.append(np.array(temp).mean()) #Calculating the mean of MSE
    std_error.append(np.array(temp).std()) #Calculating the Standard
Deviation of MSE
import matplotlib.pyplot as plt
plt.errorbar((Ci_range),mean_error,yerr=std_error,ecolor='red',fmt='-o')
#Plotting the Error Bar
plt.xlabel('Ci'); plt.ylabel('Mean square error')
plt.legend(["Red-Std_Dev Error and Blue=Mean Score Error
"],loc=1,fontsize=10)
plt.title(" Lasso: Ci vs Mean Error and Std Error", fontsize=10)
```

```
plt.xlim((0,75))
plt.show()
print("Mean=",mean_error)
print("Std_Error=",std_error)
```

Q(ii.c):

```
#ii.c
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
mean_error_R=[]; std_error_R=[]
Ci_range = [0.1, 0.5, 1, 5, 10,25, 50, 100]
for Ci in Ci_range:

    model = Ridge(alpha=1/(Ci))#Defining the model with the C values above.
    temp=[]

    kf = KFold(n_splits=5)
    for train, test in kf.split(X_poly):
        model.fit(X_poly[train], y[train])#Training the Lasso Model with 5
fold Crossvalidation
        ypred = model.predict(X_poly[test])

        temp.append(mean_squared_error(y[test],ypred))#Calculating Mean
Squared Error
    mean_error_R.append(np.array(temp).mean())#Calculating the mean of MSE
    std_error_R.append(np.array(temp).std())#Calculating the Standard
Deviation of MSE
import matplotlib.pyplot as plt
plt.errorbar(Ci_range,mean_error_R,yerr=std_error_R,ecolor='red',fmt='-
o')#Plotting the Error Bar
plt.xlabel('Ci'); plt.ylabel('Mean square error')
plt.legend(["Red-Error and Blue=Mean Score "],loc=1)
plt.title(" Ridge: Ci vs Mean Error and Std Error", fontsize=10)
plt.xlim((0,20))
plt.show()
print("Mean=",mean_error)
print("Std_Error=",std_error)
```

Dummy Regressor for Baseline Prediction:

```
#Dummy Regressor
from sklearn.dummy import DummyRegressor
%matplotlib inline
kf = KFold(n_splits = 5)
mean_error1=[]; std_error1=[];

#Loop through each k fold
for Ci in Ci_range:
    mse_temp1=[]
        #Model
```

```
    model1=DummyRegressor(strategy="mean")

    for train, test in kf.split(X_poly):
        model1.fit(X_poly[train],y[train])
        ypred2 = model1.predict(X_poly[test])
        mse1= mean_squared_error(y[test],ypred2)
        mse_temp1.append(mse1)
        #Get mean & std
    mean_error1.append(np.array(mse_temp1).mean())
    std_error1.append(np.array(mse_temp1).std())
```

Plot Lasso vs Baseline:

```
    #Plot Lasso vs Baseline
plt.errorbar((Ci_range),mean_error,yerr=std_error, color
='green',label='Lasso')
plt.errorbar(Ci_range, mean_error1, yerr=std_error1, color
='red',label='Baseline')
plt.xlabel('C')
plt.ylabel('Mean square error')
plt.title('Lasso regression & baseline C vs MSE(C=[1,5,10,15,50])')
plt.legend()
plt.show()
print("Baseline Mean Error=",mean_error1)
print("Lasso Mean Error=",mean_error)
```

Plot Ridge vs Baseline:

```
    #Plot Ridge vs Baseline
plt.errorbar((Ci_range),mean_error_R,yerr=std_error_R, color
='green',label='Lasso')
plt.errorbar(Ci_range, mean_error1, yerr=std_error1, color
='red',label='Baseline')
plt.xlabel('C')
plt.ylabel('Mean square error')
plt.title('Lasso regression & baseline C vs MSE(C=[1,5,10,15,50])')
plt.legend()
plt.show()
print("Baseline Mean Error=",mean_error1)
print("Ridge Mean Error=",mean_error_R)
```