Assignment 2 – Week 3

Pallavit Aggarwal – Ms IS
ID: 22333721

Q1.

i)

The data corresponds #id:24--24-24 ; It is plotted using scatter3D with elevation set to 5 and azimuth of the z axis at 35 degrees in Fig 1.

Fig 2 shows the same data with elevation 5 and azimuth set to 120.

As I can observe by rotating this plot, the data doesn't lie on any of the X,Y,Z planes. In the given system, it appears to be lying on a curve.

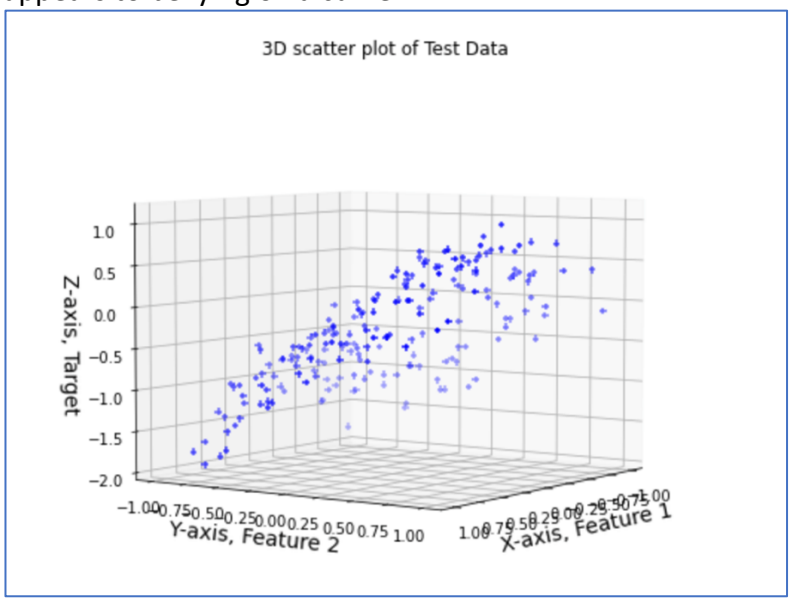


Fig 1

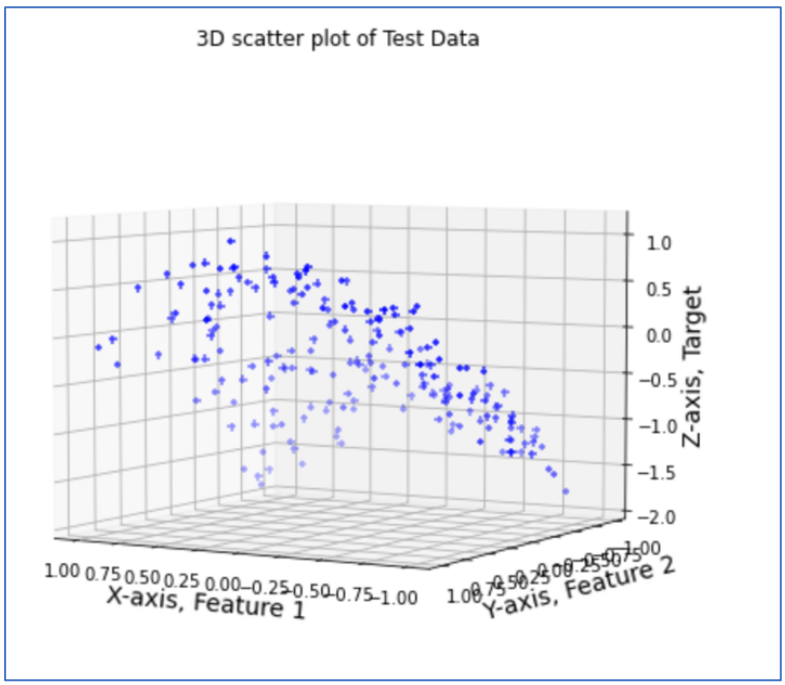


Fig 2

ii)

Using Polynomial Features from sklearn.preprocessing an instance of it is created with the following parameters. PolynomialFeatures(degree=5,include_bias=False).

The parameters for polynomial features are as follows , degree gives the degree value up to which the parameters would be raised to. There will also be combination of the parameters involved in this. For degree 5 we get a total of 21 features. I have set include_bias to false as this is a constant parameter “1” which gets added to features.

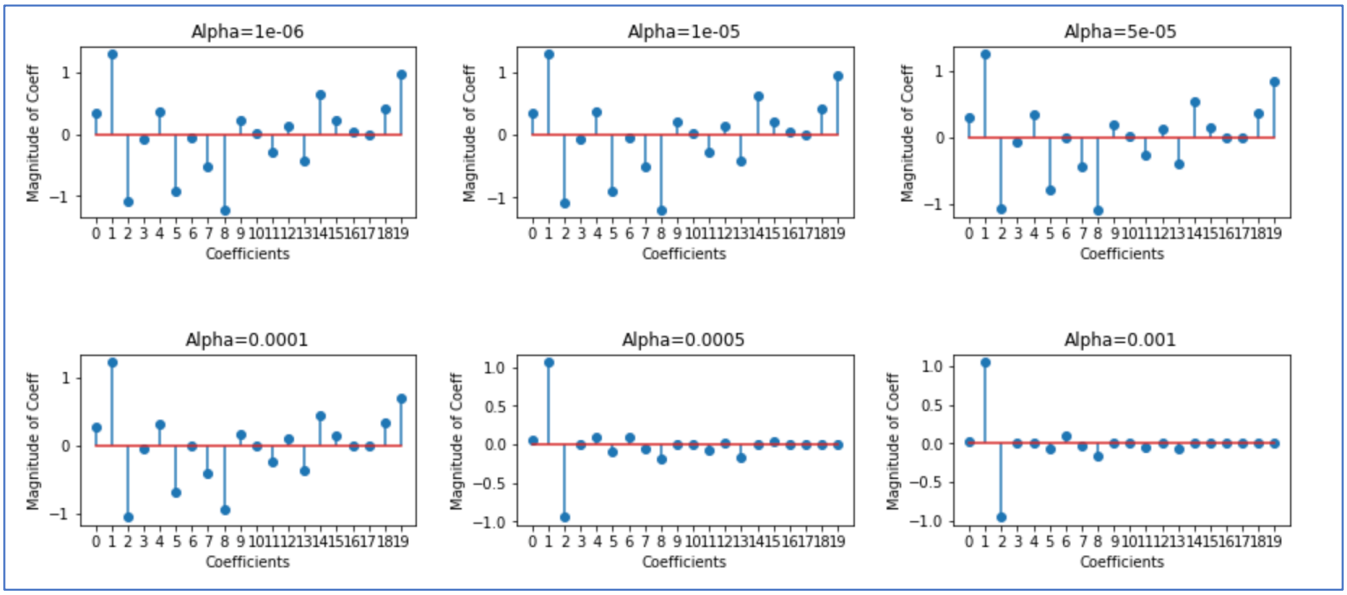
(Note: I had tried other permutation combinations while playing with Polynomial features by setting interaction_only to false which means no combinations would be added and only squares would be returned, and stacked them together and trained the model, but this approach was finally failing while plotting the plane against the scatter.)

	intrcpt	Coef_0	Coef_1	Coef_2	Coef_3	Coef_4	Coef_5	Coef_6	Coef_7	Coef_8	..
0.000001	-0.010987	0.345776	1.288321	-1.080963	-0.075533	0.371321	-0.916035	-0.064452	-0.516508	-1.217716	..
0.000010	-0.01062	0.337641	1.282422	-1.077551	-0.073828	0.365385	-0.892088	-0.053975	-0.503511	-1.193874	..
0.000050	-0.00891	0.301505	1.254925	-1.063031	-0.065948	0.33859	-0.785397	-0.004031	-0.446294	-1.085674	..
0.000100	-0.006693	0.265116	1.226529	-1.046961	-0.050469	0.306857	-0.667676	0.0	-0.404306	-0.941632	..
0.000500	0.009545	0.065044	1.06872	-0.944004	-0.0	0.093511	-0.100545	0.088555	-0.054804	-0.194257	..
0.001000	0.021256	0.035213	1.055063	-0.954638	0.0	0.0	-0.069872	0.100366	-0.026195	-0.170072	..
0.005000	0.002517	-0.0	0.98585	-0.945276	0.0	-0.0	-0.019185	0.0	-0.0	-0.0	..
0.010000	-0.019528	-0.0	0.970894	-0.89214	0.0	-0.0	-0.0	0.0	-0.0	-0.0	..
0.050000	-0.177094	-0.0	0.844837	-0.471403	0.0	-0.0	-0.0	0.0	-0.0	0.0	..
0.100000	-0.354714	-0.0	0.686627	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	..
0.500000	-0.398389	0.0	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	..
1.000000	-0.398389	0.0	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	..

..	Coef_10	Coef_11	Coef_12	Coef_13	Coef_14	Coef_15	Coef_16	Coef_17	Coef_18	Coef_19
..	0.023236	-0.291082	0.12914	-0.423145	0.631418	0.220804	0.041182	-0.004173	0.417652	0.965592
..	0.021744	-0.285551	0.128141	-0.417865	0.613738	0.208167	0.033832	-0.002039	0.409701	0.942333
..	0.015103	-0.260679	0.123204	-0.393847	0.535007	0.152837	0.001018	-0.0	0.375207	0.839319
..	0.0	-0.235265	0.110619	-0.364216	0.440126	0.145707	0.0	-0.0	0.339371	0.703239
..	-0.0	-0.082184	0.024452	-0.166151	0.0	0.032608	-0.0	0.0	0.0	0.0
..	0.0	-0.047076	0.011184	-0.067368	-0.0	0.0	-0.0	0.0	0.0	-0.0
..	0.0	-0.0	0.0	-0.013452	-0.0	0.0	-0.0	0.0	-0.0	-0.0
..	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	-0.0	-0.0
..	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	-0.0	0.0
..	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	-0.0	0.0
..	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	0.0	0.0
..	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	0.0	0.0

The coefficients with changing values of C are listed in the above table, split vertically into two images.

Next, I plot the coefficients on a stem plot to visualize the change better. As we can see, when Alpha increases, i.e., C decreases in Alpha = 1/2C, the coefficients are penalized more.



For the same alpha values as above I report the training error and the testing error as mean squared errors of training data and predictions on training data, and test data and predictions on test data.

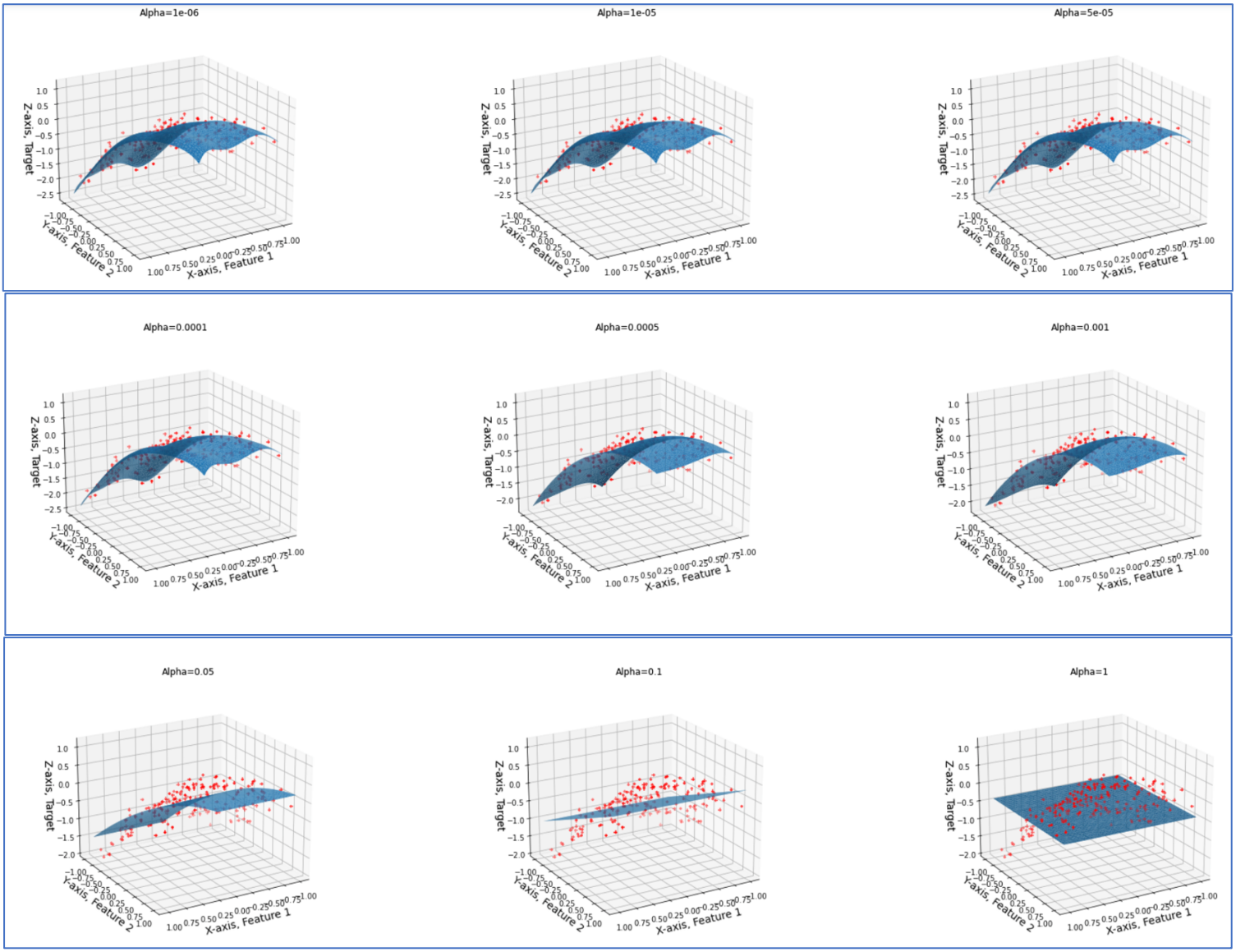
```
alpha:    1e-06 | train error: 0.031 | test error: 0.041
alpha:    1e-05 | train error: 0.031 | test error: 0.041
alpha:    5e-05 | train error: 0.031 | test error: 0.039
alpha:   0.0001 | train error: 0.031 | test error: 0.038
alpha:   0.0005 | train error: 0.034 | test error: 0.034
alpha:    0.001 | train error: 0.034 | test error: 0.034
```

iii)

Lasso Regression is a type of regularized linear regression that includes an L1 penalty. This has the effect of shrinking the coefficients for those input variables that do not contribute much to the prediction task. It allows some coefficient values to go to the value of zero, allowing input variables to be effectively removed from the model, providing a type of automatic feature selection.

Lasso Regression is an extension of linear regression that adds a regularization penalty to the loss function during training

In Model tuning we try to decrease the complexity of the model, and one way to do this is through regularization.



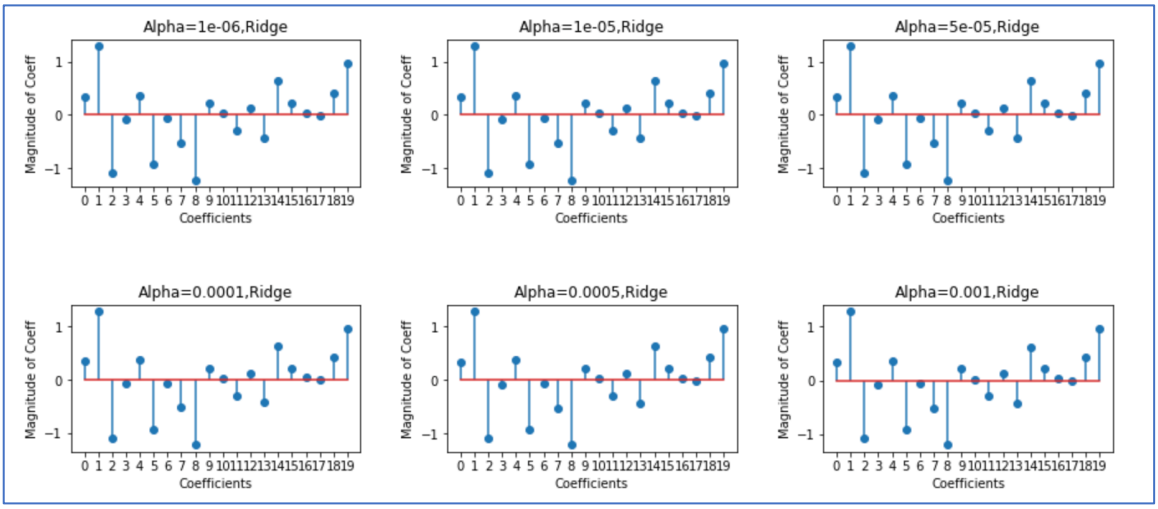
iv)

In the above figures, the blue plane is the predictions depicted as a surface and the red dots are the data points scattered in a 3d plane.
We can observe that when alpha is very small, the coefficients have significant weights corresponding to them in the model, and this leads to a wavy plane and a possible scenario of over-fitting.
When the value of the hyper-parameter becomes large, we start penalizing the model for having weights to multiple coefficients or features and we start restricting this. As the coefficients go towards 0, we can see the plane, doesn't fit the data properly.

As we add more parameters, we start to fit the "noise" in the training data, called overfitting. (ML Slides-Features1.pdf)

Over-fitting is when we rely too heavily on our training data and give too much importance to features, so much so that we adapt our model (visually thinking) completely to the data points. This leads to errors with test-data or live data as it may not adhere to the training data. Under-fitting is when it cannot model the training data nor generalize to new data.

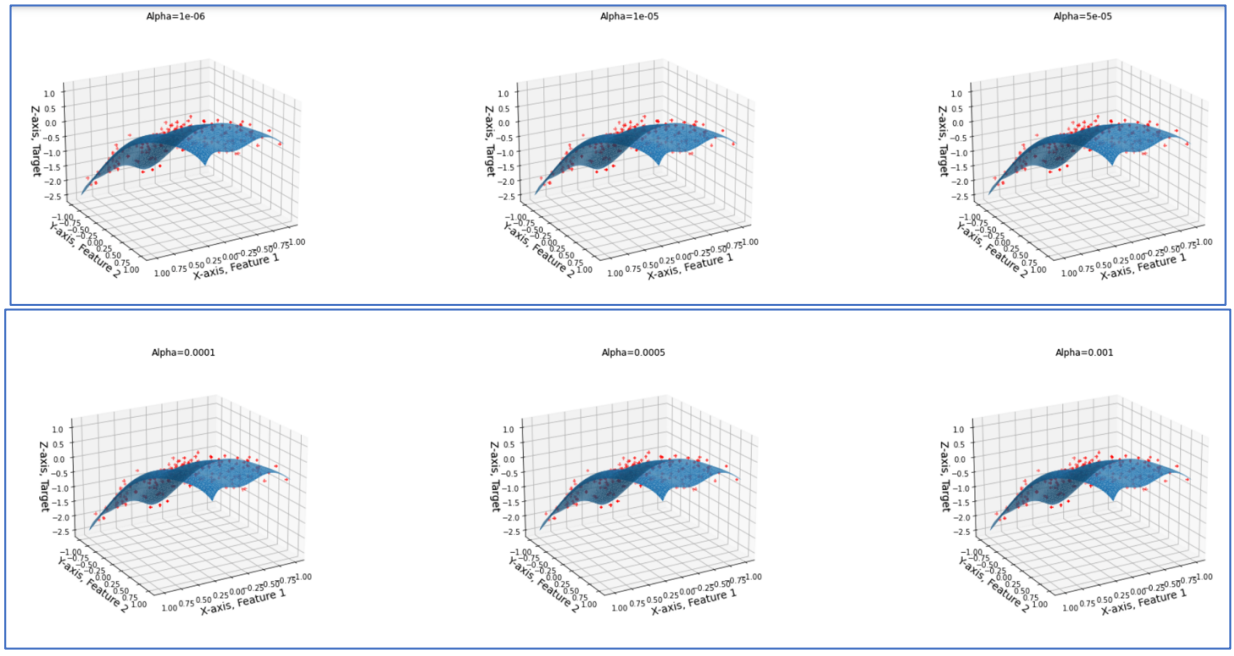
v)
Ridge Regression which uses L2 penalty

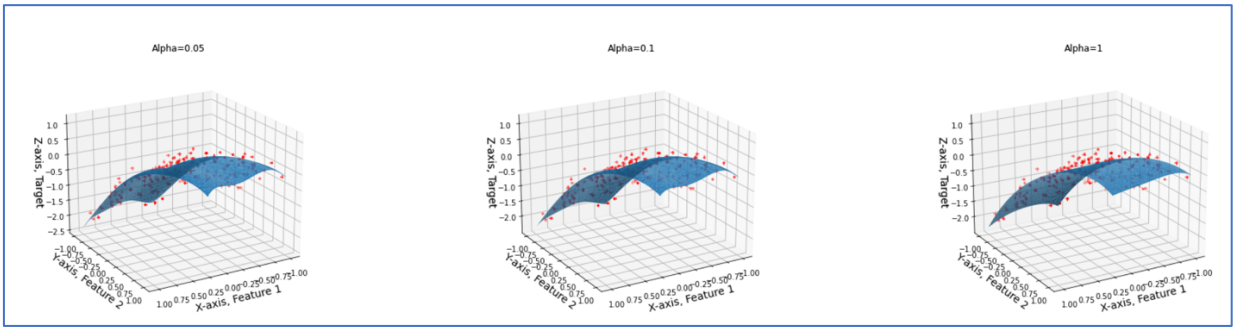


For Alpha values [0.000001, 0.00001, 0.00005, 0.0001, 0.0005, 0.001], the coefficients are visualized above.

```
alpha:    1e-06 | train error: 0.031 | test error: 0.041
alpha:    1e-05 | train error: 0.031 | test error: 0.041
alpha:    5e-05 | train error: 0.031 | test error: 0.041
alpha:   0.0001 | train error: 0.031 | test error: 0.041
alpha:   0.0005 | train error: 0.031 | test error: 0.041
alpha:    0.001 | train error: 0.031 | test error: 0.041
```

The training errors and test error as mean squared error of training and test target values against predicted values are show in the above figure.





Though Ridge and Lasso might appear to work towards a common goal, the inherent properties and practical use cases differ substantially.

Ridge performs L2 regularization, i.e., adds penalty equivalent to square of the magnitude of coefficients.

We can observe that at alpha = 0.001, in Lasso Regression the coefficients were tending to 0, but in Ridge regression they still have some weight.

```
alpha:   1e-06 | train error: 0.032 | test error: 0.042
alpha:   1e-05 | train error: 0.032 | test error: 0.042
alpha:   5e-05 | train error: 0.032 | test error: 0.041
alpha:  0.0001 | train error: 0.032 | test error: 0.041
alpha:  0.0005 | train error: 0.033 | test error: 0.037
alpha:   0.001 | train error: 0.034 | test error: 0.034
alpha:   0.005 | train error: 0.038 | test error: 0.033
alpha:    0.01 | train error: 0.039 | test error: 0.034
alpha:    0.05 | train error: 0.074 | test error: 0.074
alpha:     0.1 | train error: 0.157 | test error: 0.165
alpha:     0.5 | train error: 0.446 | test error: 0.473
alpha:       1 | train error: 0.446 | test error: 0.473
```

Again, training and test errors are mean squared errors of training and test target values against predicted values.
I have selected the same alpha values as above, and added more values at the bottom after 0.001. We can see that the magnitude of testing and training errors increase as the alpha increase (C decreases). The test/train error percentage is minimum at alpha – 0.001 and 0.05. This makes it the most favorable values to go for.

Q2.

i)
Here using the percentage for each k-fold, we can see the test-error/train-error. A negative percentage means that training error exceeds testing error by that percentage. We would want to keep the difference as small as possible.

```
For k =  0
alpha: 1e-06 | test-error/train-error 33.0%
alpha: 1e-05 | test-error/train-error 32.0%
alpha: 5e-05 | test-error/train-error 30.0%
alpha: 0.0001 | test-error/train-error 28.0%
alpha: 0.0005 | test-error/train-error 13.0%
alpha: 0.001 | test-error/train-error 0.0%
alpha: 0.005 | test-error/train-error -13.0%
alpha: 0.01 | test-error/train-error -13.0%
alpha: 0.05 | test-error/train-error 1.0%
alpha: 0.1 | test-error/train-error 5.0%
alpha: 0.5 | test-error/train-error 6.0%
alpha: 1 | test-error/train-error 6.0%
```

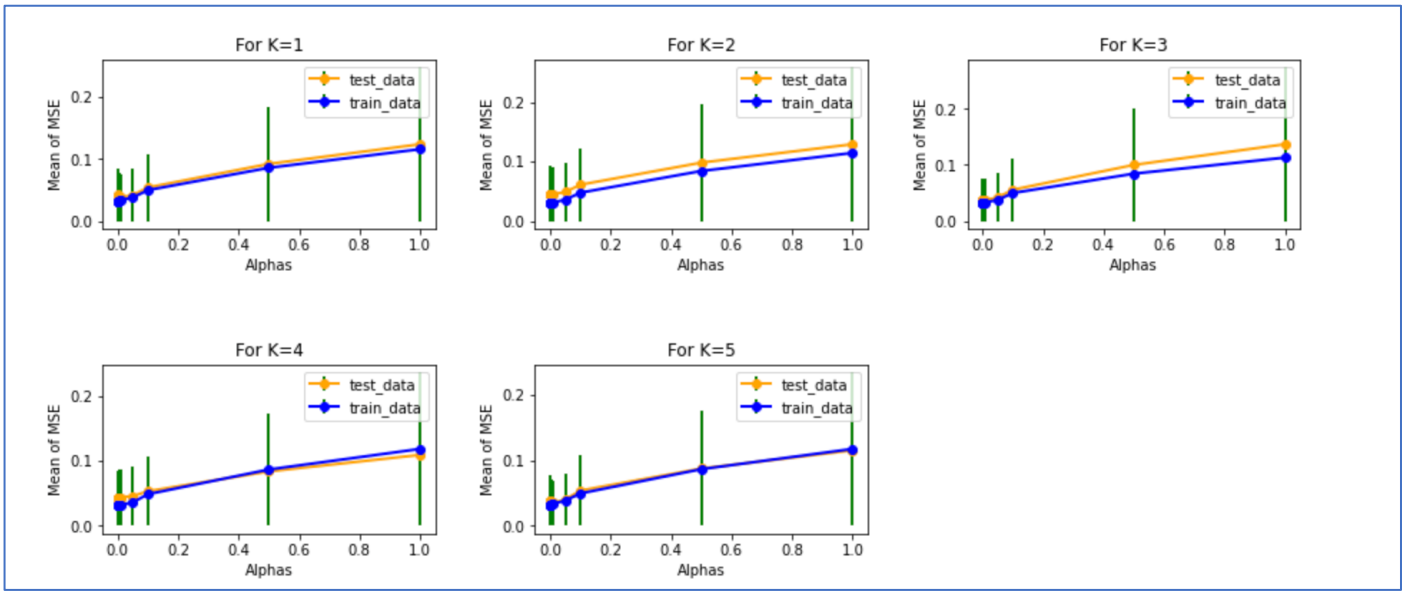
```
For k =  1
alpha: 1e-06 | test-error/train-error 54.0%
alpha: 1e-05 | test-error/train-error 54.0%
alpha: 5e-05 | test-error/train-error 54.0%
alpha: 0.0001 | test-error/train-error 54.0%
alpha: 0.0005 | test-error/train-error 49.0%
alpha: 0.001 | test-error/train-error 42.0%
alpha: 0.005 | test-error/train-error 29.0%
alpha: 0.01 | test-error/train-error 23.0%
alpha: 0.05 | test-error/train-error 9.0%
alpha: 0.1 | test-error/train-error 11.0%
alpha: 0.5 | test-error/train-error 5.0%
alpha: 1 | test-error/train-error 5.0%
```

```
For k =  2
alpha: 1e-06 | test-error/train-error 18.0%
alpha: 1e-05 | test-error/train-error 18.0%
alpha: 5e-05 | test-error/train-error 17.0%
alpha: 0.0001 | test-error/train-error 16.0%
alpha: 0.0005 | test-error/train-error 14.0%
alpha: 0.001 | test-error/train-error 9.0%
alpha: 0.005 | test-error/train-error 1.0%
alpha: 0.01 | test-error/train-error 4.0%
alpha: 0.05 | test-error/train-error 13.0%
alpha: 0.1 | test-error/train-error 11.0%
alpha: 0.5 | test-error/train-error 26.0%
alpha: 1 | test-error/train-error 26.0%
```

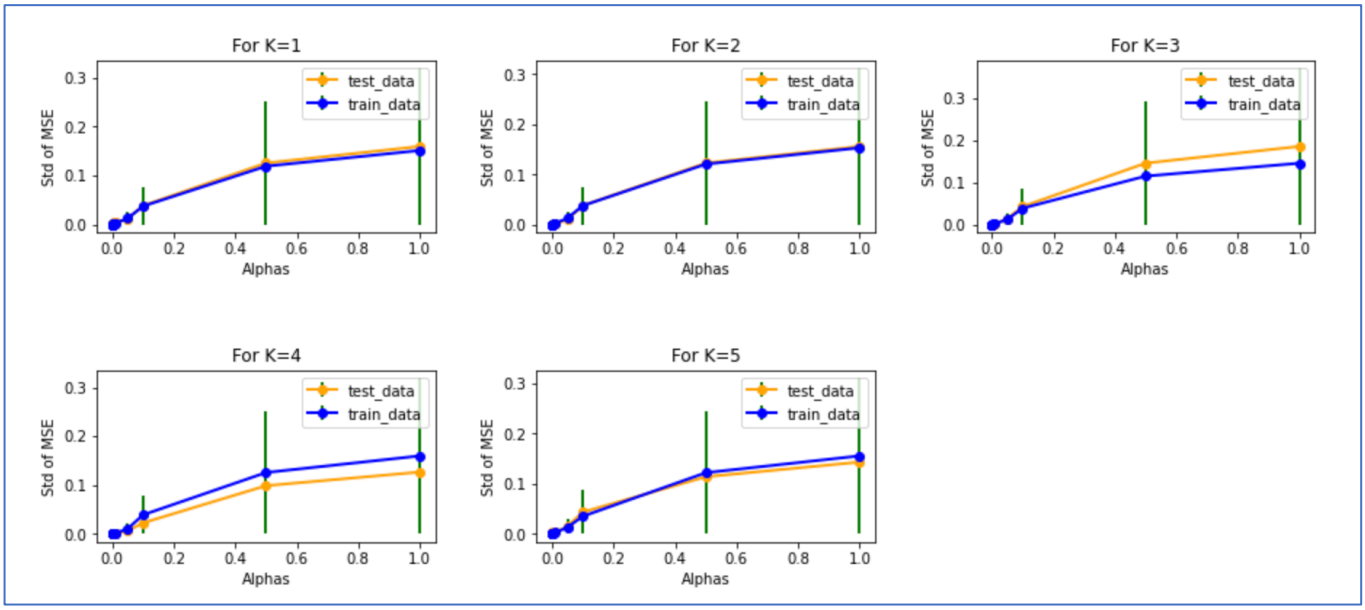
```
For k =  3
alpha: 1e-06 | test-error/train-error 35.0%
alpha: 1e-05 | test-error/train-error 35.0%
alpha: 5e-05 | test-error/train-error 35.0%
alpha: 0.0001 | test-error/train-error 35.0%
alpha: 0.0005 | test-error/train-error 36.0%
alpha: 0.001 | test-error/train-error 38.0%
alpha: 0.005 | test-error/train-error 37.0%
alpha: 0.01 | test-error/train-error 33.0%
alpha: 0.05 | test-error/train-error -5.0%
alpha: 0.1 | test-error/train-error -27.0%
alpha: 0.5 | test-error/train-error -17.0%
alpha: 1 | test-error/train-error -17.0%
```

```
For k =  4
alpha: 1e-06 | test-error/train-error 22.0%
alpha: 1e-05 | test-error/train-error 22.0%
alpha: 5e-05 | test-error/train-error 18.0%
alpha: 0.0001 | test-error/train-error 15.0%
alpha: 0.0005 | test-error/train-error -4.0%
alpha: 0.001 | test-error/train-error -7.0%
alpha: 0.005 | test-error/train-error -15.0%
alpha: 0.01 | test-error/train-error -18.0%
alpha: 0.05 | test-error/train-error 9.0%
alpha: 0.1 | test-error/train-error 20.0%
alpha: 0.5 | test-error/train-error -8.0%
alpha: 1 | test-error/train-error -8.0%
```

I am also reporting the mean and std errors of the training and test data.



This figure above shows mean error with green line showing yerr (mean of mse)



This figure above shows std deviation error with green line showing yerr(std of mse)

ii)

Based on the cross validation data and the test and training error percentages for different k and different alphas, along with considering the prediction plane, Alpha at 0.001 or 0.005 works most optimally in predicting correct values based on training data.

The plane for Alpha = 0.001 and Alpha = 0.005 is most accurately covering all the points.

iii)

For Ridge Regression

```
For k =  0
alpha: 1e-06 | test-error/train-error 33.0%
alpha: 1e-05 | test-error/train-error 33.0%
alpha: 5e-05 | test-error/train-error 33.0%
alpha: 0.0001 | test-error/train-error 33.0%
alpha: 0.0005 | test-error/train-error 33.0%
alpha: 0.001 | test-error/train-error 33.0%
alpha: 0.005 | test-error/train-error 32.0%
alpha: 0.01 | test-error/train-error 31.0%
alpha: 0.05 | test-error/train-error 27.0%
alpha: 0.1 | test-error/train-error 23.0%
alpha: 0.5 | test-error/train-error 11.0%
alpha: 1 | test-error/train-error 7.0%


For k =  1
alpha: 1e-06 | test-error/train-error 54.0%
alpha: 1e-05 | test-error/train-error 54.0%
alpha: 5e-05 | test-error/train-error 54.0%
alpha: 0.0001 | test-error/train-error 54.0%
alpha: 0.0005 | test-error/train-error 54.0%
alpha: 0.001 | test-error/train-error 54.0%
alpha: 0.005 | test-error/train-error 54.0%
alpha: 0.01 | test-error/train-error 54.0%
alpha: 0.05 | test-error/train-error 54.0%
alpha: 0.1 | test-error/train-error 53.0%
alpha: 0.5 | test-error/train-error 48.0%
alpha: 1 | test-error/train-error 45.0%
```

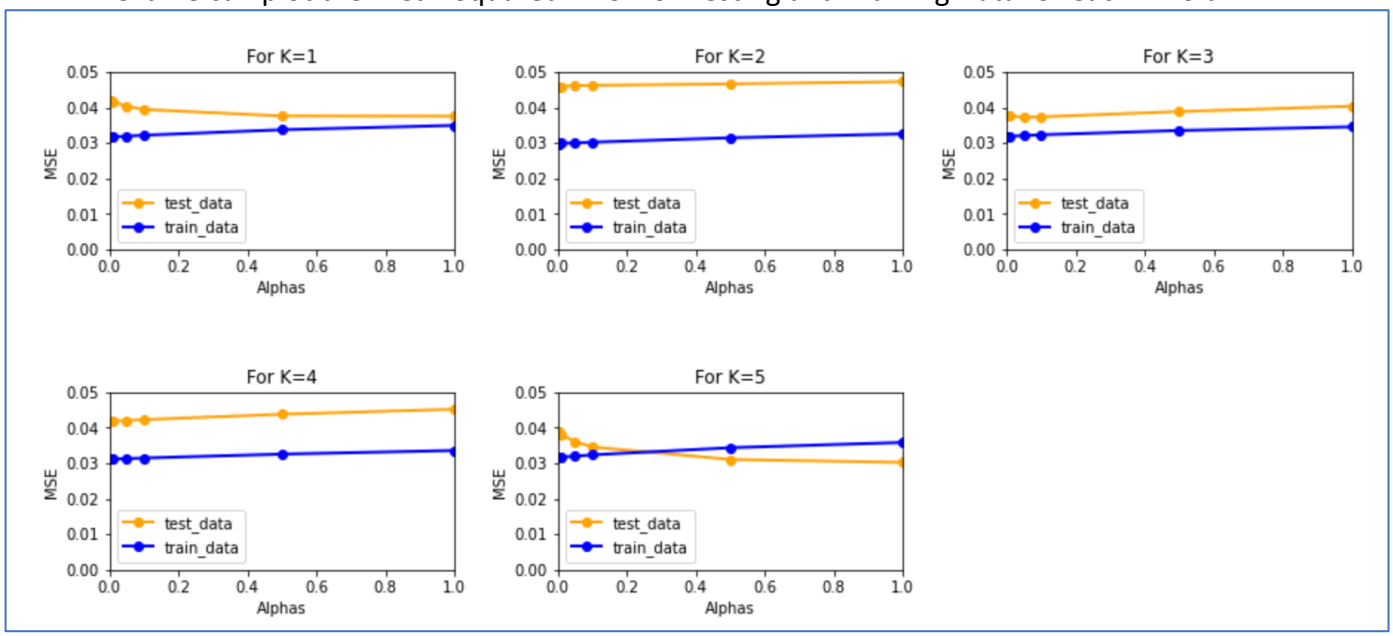
```
For k =  2
alpha: 1e-06 | test-error/train-error 18.0%
alpha: 1e-05 | test-error/train-error 18.0%
alpha: 5e-05 | test-error/train-error 18.0%
alpha: 0.0001 | test-error/train-error 18.0%
alpha: 0.0005 | test-error/train-error 18.0%
alpha: 0.001 | test-error/train-error 18.0%
alpha: 0.005 | test-error/train-error 18.0%
alpha: 0.01 | test-error/train-error 18.0%
alpha: 0.05 | test-error/train-error 16.0%
alpha: 0.1 | test-error/train-error 16.0%
alpha: 0.5 | test-error/train-error 16.0%
alpha: 1 | test-error/train-error 17.0%


For k =  3
alpha: 1e-06 | test-error/train-error 35.0%
alpha: 1e-05 | test-error/train-error 35.0%
alpha: 5e-05 | test-error/train-error 35.0%
alpha: 0.0001 | test-error/train-error 35.0%
alpha: 0.0005 | test-error/train-error 35.0%
alpha: 0.001 | test-error/train-error 35.0%
alpha: 0.005 | test-error/train-error 35.0%
alpha: 0.01 | test-error/train-error 35.0%
alpha: 0.05 | test-error/train-error 34.0%
alpha: 0.1 | test-error/train-error 34.0%
alpha: 0.5 | test-error/train-error 34.0%
alpha: 1 | test-error/train-error 35.0%
```

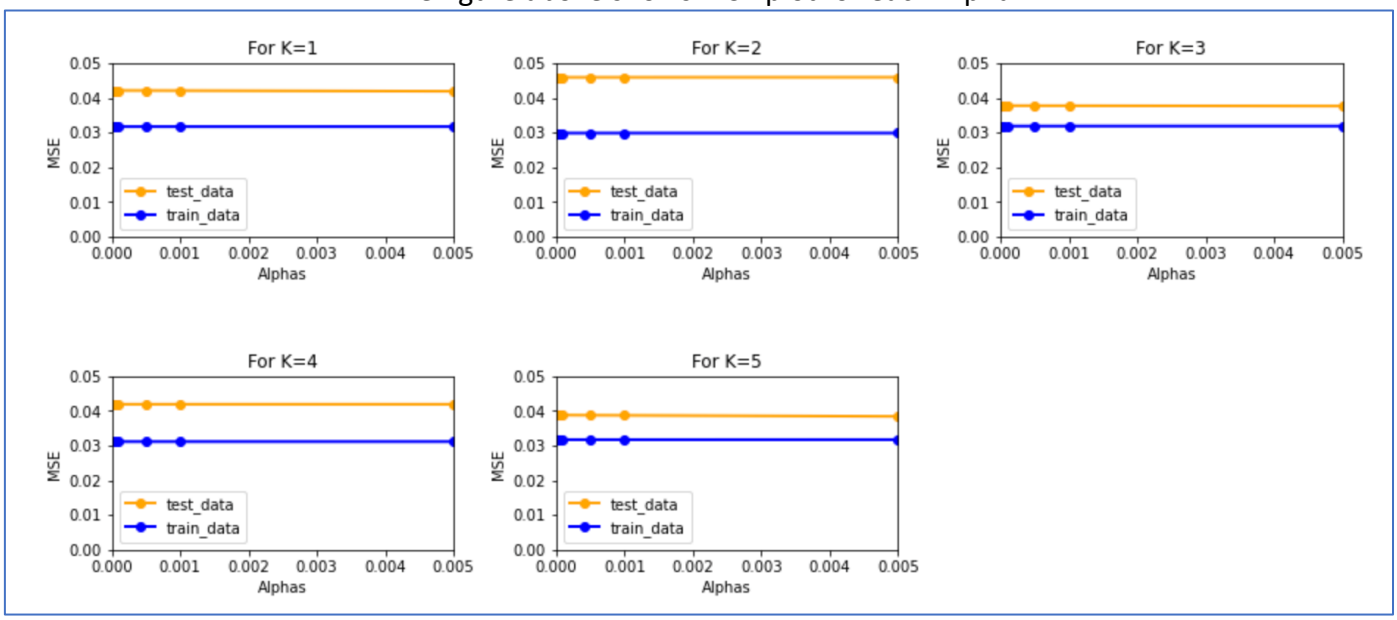
```
For k =  4
alpha: 1e-06 | test-error/train-error 22.0%
alpha: 1e-05 | test-error/train-error 22.0%
alpha: 5e-05 | test-error/train-error 22.0%
alpha: 0.0001 | test-error/train-error 22.0%
alpha: 0.0005 | test-error/train-error 22.0%
alpha: 0.001 | test-error/train-error 22.0%
alpha: 0.005 | test-error/train-error 21.0%
alpha: 0.01 | test-error/train-error 20.0%
alpha: 0.05 | test-error/train-error 13.0%
alpha: 0.1 | test-error/train-error 7.0%
alpha: 0.5 | test-error/train-error -10.0%
alpha: 1 | test-error/train-error -16.0%
```

I first report the test/train error percentage as the mean-squared-error of test targets vs predicted targets on test data and the train targets vs predicted targets on training data. A negative percentage means that training error exceeds testing error by that percentage. We would want to keep the difference as small as possible.

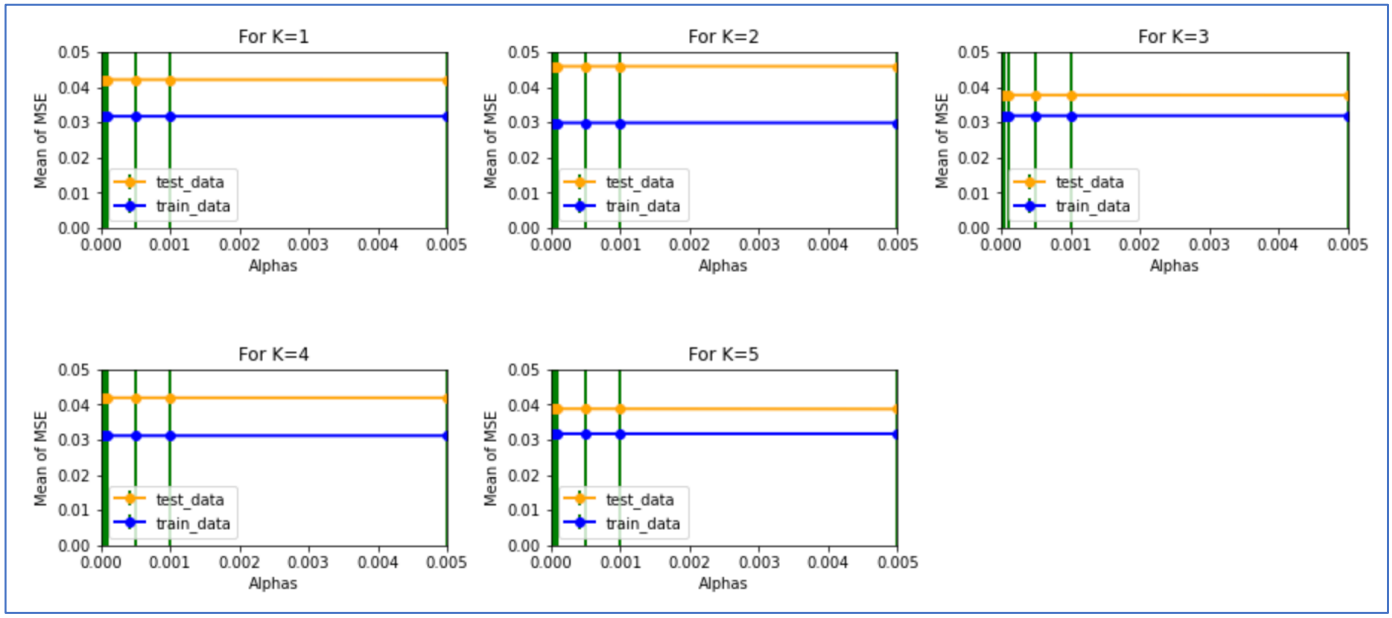
Next we can plot the Mean Squared Error for Testing and Training Data for each K-Fold



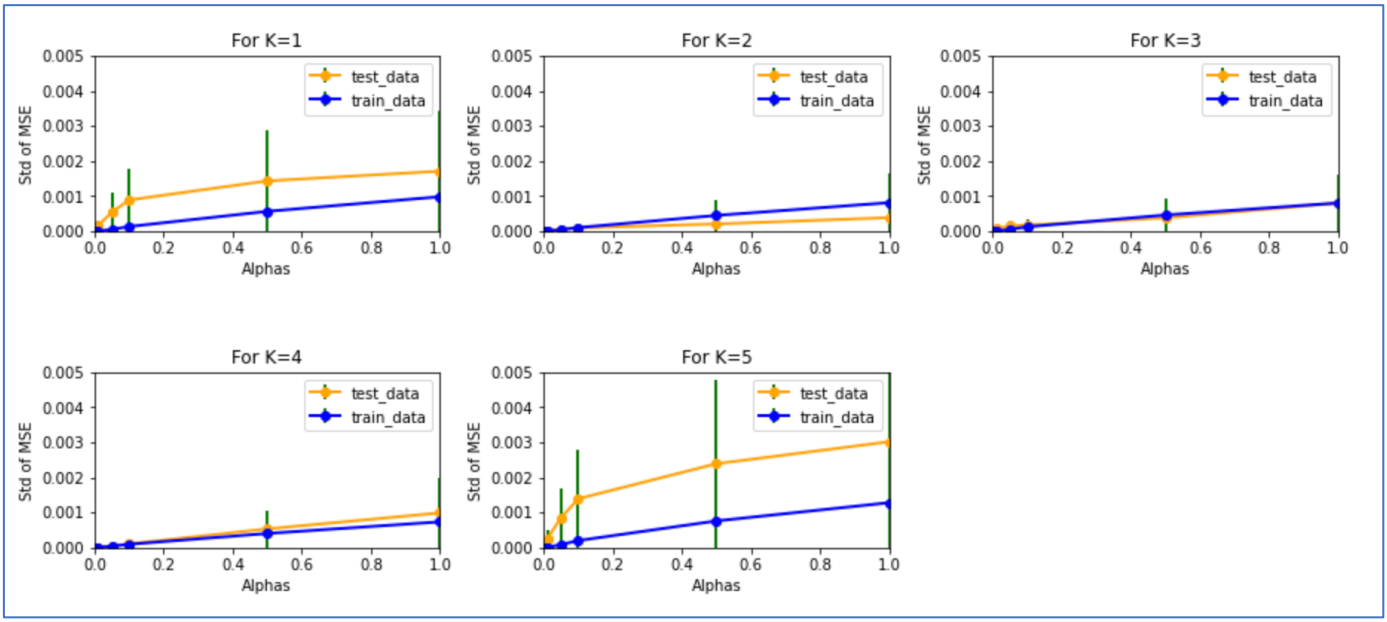
The figure above shows MSE plot for each Alpha



The figure above shows MSE plot with Alphas zoomed into 0 to 0.005



This figure above shows mean error with yerr given in green (mean of mse)



This figure above shows std deviation error with yerr in green (std of mse)

Based on the cross-validation data and the test and training error percentages for different k and different alphas, along with considering the prediction plane, Alpha at 0.001 or 0.005 works most optimally in predicting correct values based on training data.

The plane for Alpha = 0.001 and Alpha = 0.005 is most accurately covering all the points.
Also,

The performance of Lasso and Ridge with Alphas as 0.001 and 0.005 against a dummy regressor are reported below

```
Square Error- Lasso(Alpha=0.005): 0.033990 , Dummy: 0.476904, Lasso(0.005) with K-Fold=5: 0.031560
Square Error- Ridge(Alpha=0.005): 0.040171 , Dummy: 0.476904, Ridge(0.005) with K-Fold=5: 0.037638
```

```
Square Error- Lasso(Alpha=0.001): 0.034167 , Dummy: 0.476904, Lasso(0.001) with K-Fold=5: 0.032030
Square Error- Ridge(Alpha=0.001): 0.040734 , Dummy: 0.476904, Ridge(0.001) with K-Fold=5: 0.037736
```

Appendix

Q1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, PolynomialFeatures

df = pd.read_csv('week3.csv')

X1=df.iloc[:,0]
X2=df.iloc[:,1]
y=df.iloc[:,2]

X = np.column_stack((X1,X2))
---
fig = plt.figure(figsize = (10, 7))
ax = fig.add_subplot(111,projection='3d')

ax.view_init(5, 35)
ax.scatter3D(X[:,0],X[:,1],y,marker='+',c='b',cmap='binary')
ax.set_title('3D scatter plot of Test Data')
ax.set_xlabel('X-axis, Feature 1',fontsize=14)
ax.set_ylabel('Y-axis, Feature 2',fontsize=14)
ax.set_zlabel('Z-axis, Target',fontsize=14)
---
fig = plt.figure(figsize = (10, 7))
ax = fig.add_subplot(111,projection='3d')
ax.set_title('3D scatter plot of Test Data')
ax.view_init(5, 120)
ax.scatter3D(X[:,0],X[:,1],y,marker='+',c='b',cmap='binary')
ax.set_xlabel('X-axis, Feature 1',fontsize=14)
ax.set_ylabel('Y-axis, Feature 2',fontsize=14)
ax.set_zlabel('Z-axis, Target',fontsize=14)
---
trans5 = PolynomialFeatures(degree=5,include_bias=False)
X1_poly5 = trans5.fit_transform(X)
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X1_poly5,y,test_size=0.20,random_state =
0)
---
from sklearn.linear_model import Lasso
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error
from IPython.display import display
model_params = []
print('--- Lasso Regression ---')
C_values = [0.000001, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1]

Cols = ['intrcpt']
```

```
for i in range(0,20):
   name = 'Coef_'+str(i)
   Cols.append(name)

coeff = pd.DataFrame(index=C_values, columns=Cols)
i=0
for c_value in C_values:
   model = Lasso(alpha=c_value, max_iter=10000)
   model.fit(X_train, Y_train)
   ret = [model.intercept_]
   ret.extend(model.coef_)
   coeff.iloc[i,] = ret
   i=i+1


display(coeff)
---
model_params = []
C_values = [0.000001, 0.00001, 0.00005, 0.0001, 0.0005, 0.001]

Cols = []
for i in range(0,len(model.coef_)):
   name = str(i)
   Cols.append(name)

data=[]
for c_value in C_values:
   model = Lasso(alpha=c_value, max_iter=10000)
   model.fit(X_train, Y_train)
   data.append(model.coef_)

fig = plt.figure(figsize=(15, 10))
fig.subplots_adjust(hspace=0.8, wspace=0.3)

for i in range(0,len(data)):
   ax = fig.add_subplot(3, 3, i+1)
   ax.stem(Cols,data[i])
   ax.set_title("Alpha="+str(C_values[i]))
   ax.set_xlabel("Coefficients")
   ax.set_ylabel("Magnitude of Coeff")
---
model_params = []
C_values = [0.000001, 0.00001, 0.00005, 0.0001, 0.0005, 0.001]

Cols = []
for i in range(0,len(model.coef_)):
   name = str(i)
   Cols.append(name)

data=[]
for c_value in C_values:
   model = Lasso(alpha=c_value, max_iter=10000)
```

```
    model.fit(X_train, Y_train)
    new_train_error = mean_squared_error(Y_train, model.predict(X_train))
    new_test_error = mean_squared_error(Y_test, model.predict(X_test))
    print('alpha: {:7} | train error: {:5} | test error: {}'
      .format(c_value,round(new_train_error,3),round(new_test_error,3)))
---
Xtest = []

grid = np.linspace(-max(X[:,0]),max(X[:,0]))

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

Xtest=np.array(Xtest)
Xtest = trans5.fit_transform(Xtest)

fig = plt.figure(figsize=(30, 30))
fig.subplots_adjust(hspace=0.8, wspace=0.3)

i=0
C_values = [0.000001, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.05, 0.1, 1]
for c_value in C_values:
    model = Lasso(alpha=c_value, max_iter=10000)
    model.fit(X_train, Y_train)
    y_pred = model.predict(Xtest)
    ax = fig.add_subplot(3, 3, i+1, projection='3d')
    ax.view_init(20, 60)
    ax.set_title('Alpha='+str(c_value))
    ax.set_xlabel('X-axis, Feature 1',fontsize=14)
    ax.set_ylabel('Y-axis, Feature 2',fontsize=14)
    ax.set_zlabel('Z-axis, Target',fontsize=14)
    ax.scatter3D(X[:,0],X[:,1],y,marker='+',c='red',cmap='binary')
    ax.plot_trisurf(Xtest[:,0], Xtest[:,1], y_pred, linewidth = 0.2, antialiased = True)
    i+=1

---

Q2

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, PolynomialFeatures

df = pd.read_csv('week3.csv')

X1=df.iloc[:,0]
X2=df.iloc[:,1]
y=df.iloc[:,2]

X = np.column_stack((X1,X2))
```

```
---
trans5 = PolynomialFeatures(degree=5,include_bias=False)
X1_poly5 = trans5.fit_transform(X)
---
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold

model_params = []
C_values = [0.000001, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1]
kf = KFold(n_splits=5)
i=0
fig = plt.figure(figsize=(15, 10))
fig.subplots_adjust(hspace=0.8, wspace=0.3)
fig2 = plt.figure(figsize=(15, 10))
fig2.subplots_adjust(hspace=0.8, wspace=0.3)
for train, test in kf.split(X1_poly5,y):
   print('For k = ',i)
   mean_error=[]
   std_error=[]
   temp=[]
   mean_error2=[]
   std_error2=[]
   temp2=[]
   for c_value in C_values:
      model = Ridge(alpha=c_value, max_iter=10000)
      model.fit(X1_poly5[train], y[train])
      predictions_test = model.predict(X1_poly5[test])
      predictions_train = model.predict(X1_poly5[train])
      temp.append(mean_squared_error(y[test],predictions_test))
      mean_error.append(np.array(temp).mean())
      std_error.append(np.array(temp).std())
      temp2.append(mean_squared_error(y[train],predictions_train))
      mean_error2.append(np.array(temp2).mean())
      std_error2.append(np.array(temp2).std())
#       new_train_error = mean_squared_error(y[train], model.predict(X1_poly5[train]))
#       new_test_error = mean_squared_error(y[test], model.predict(X1_poly5[test]))
#       print('alpha: {} | test-error/train-error {}%'
#             .format(c_value,round(((round(new_test_error/new_train_error,2)-1))*100,2)))
   i+=1
   print('\n')
   ax = fig.add_subplot(3, 3, i)
   ax2 = fig2.add_subplot(3, 3, i)
   ax.set_title('For K='+str(i))
   ax.errorbar(C_values,mean_error,yerr=mean_error,color='orange',linewidth=2,fmt='-
o',ecolor='green',label='test_data')
   ax.errorbar(C_values,mean_error2,yerr=mean_error2,color='blue',linewidth=2,fmt='-
o',ecolor='green',label='train_data')
   ax.set_xlabel('Alphas')
   ax.set_ylabel('Mean of MSE')
   ax.legend()
```

```
   ax2.set_title('For K='+str(i))
   ax2.errorbar(C_values,std_error,yerr=std_error,color='orange',linewidth=2,fmt='-
o',ecolor='green',label='test_data')
   ax2.errorbar(C_values,std_error2,yerr=std_error2,color='blue',linewidth=2,fmt='-
o',ecolor='green',label='train_data')
   ax2.set_xlabel('Alphas')
   ax2.set_ylabel('Std of MSE')
   ax2.legend()
---

Comparison Logic
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

kf = KFold(n_splits=5)

X_train, X_test, Y_train, Y_test = train_test_split(X1_poly5,y,test_size=0.20,random_state =
0)

model = Lasso(alpha=0.001, max_iter=10000).fit(X_train, Y_train)
ypred = model.predict(X_test)

ridge = Ridge(alpha=0.001, max_iter=10000).fit(X_train, Y_train)
ypredR = ridge.predict(X_test)

from sklearn.dummy import DummyRegressor
dummy = DummyRegressor(strategy="mean").fit(X_train, Y_train)
ydummy = dummy.predict(X_test)

temp=[]
temp2=[]
for train, test in kf.split(X1_poly5,y):
   model = Ridge(alpha=0.001, max_iter=10000)
   model.fit(X1_poly5[train], y[train])
   predictions_test = model.predict(X1_poly5[test])
   temp.append(mean_squared_error(y[test],predictions_test))
   model2 = Lasso(alpha=0.001, max_iter=10000)
   model2.fit(X1_poly5[train], y[train])
   predictions_test2 = model2.predict(X1_poly5[test])
   temp2.append(mean_squared_error(y[test],predictions_test2))



print("Square Error- Lasso(Alpha=0.001): %f , Dummy: %f, Lasso(0.001) with K-Fold=5: %f"
    %(mean_squared_error(Y_test,ypred),mean_squared_error(Y_test,ydummy),
min(temp2)))
print("Square Error- Ridge(Alpha=0.001): %f , Dummy: %f, Ridge(0.001) with K-Fold=5: %f"
    %(mean_squared_error(Y_test,ypredR),mean_squared_error(Y_test,ydummy),
min(temp)))
```