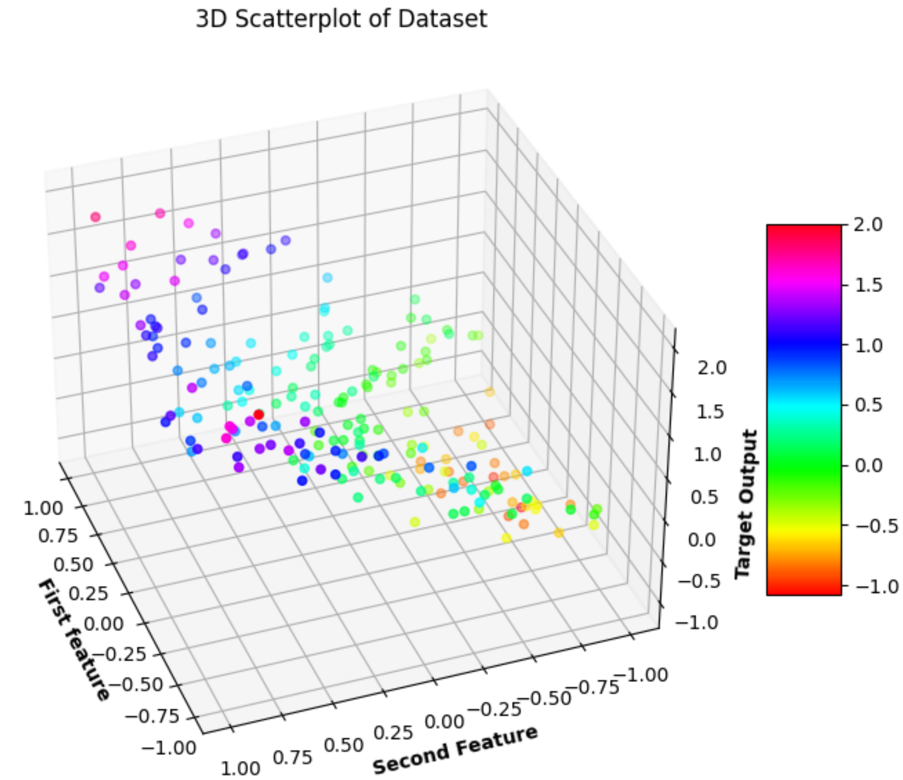
Dataset ID: 7-7-7
Q1a.



The training data looks like it lies along a curved plane.

Q1b.

	C	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
0	1	0.352934	0.0	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.0	0.0
1	10	0.137264	0.0	-0.000000	0.849900	0.526620	-0.000000	0.000000	-0.000000	0.000000	-0.0	0.0
2	100	-0.036367	0.0	-0.000000	0.992433	1.011042	-0.000754	0.000000	-0.000000	0.000000	-0.0	0.0
3	500	-0.050579	0.0	-0.002159	1.029724	1.055176	-0.037633	-0.000000	-0.010402	-0.000000	0.0	-0.0
4	1000	-0.062617	0.0	-0.005039	1.049500	1.169622	-0.039731	0.000000	-0.006309	-0.000000	0.0	-0.0
5	2000	-0.073307	0.0	0.005275	1.055322	1.261005	-0.073603	0.001865	-0.128686	-0.000000	0.0	-0.0
6	5000	-0.085944	0.0	0.082696	1.068722	1.334968	-0.134064	0.009884	-0.488390	-0.135795	0.0	-0.0

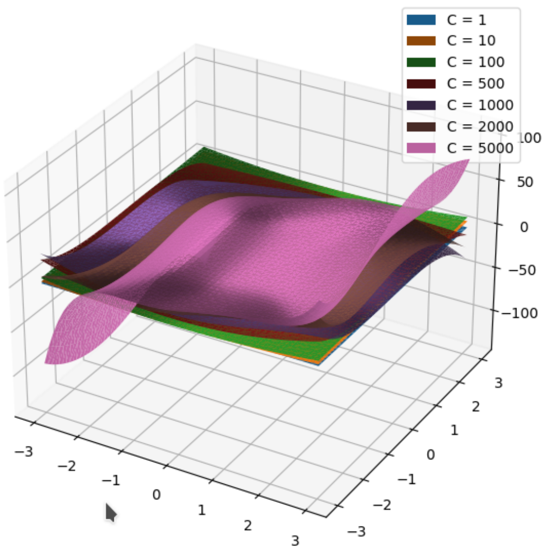
	B11	B12	B13	B14	B15	B16	B17	B18	B19	B20	B21
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000	0.000000	-0.000000	0.000000
1	0.000000	-0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000	-0.000000	0.000000	-0.000000	0.000000
2	0.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.000000
3	-0.000000	-0.000000	-0.000000	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.051650	0.000000	-0.031858
4	-0.116049	0.000000	-0.021984	0.000000	0.000000	0.000000	-0.000000	-0.014490	-0.104438	0.014471	-0.050076
5	-0.208896	0.030396	-0.078460	0.018499	0.015993	0.156666	0.000000	-0.170443	-0.131208	0.134911	-0.052218
6	-0.284978	0.054407	-0.128443	0.104301	0.041561	0.472412	0.169997	-0.189706	-0.145143	0.148536	-0.057154

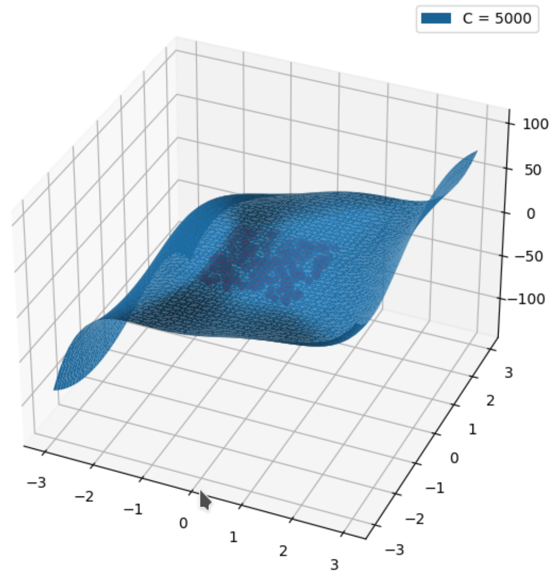
**you might need to zoom in to see the parameters.*

The above pandas dataframe captures all the parameters for each polynomial feature for each model, as well as the C-value used for each model. As the value increases the params begin to gain weight. Some parameters don't begin to gain any weight until well into C-values of 2000+ it seems.

Q1c.

The two plots below show the model predictions under different C-value constraints, and the target outputs from the initial dataset. I have separated one surface plot on the right for brevity. Clearly as the C-value increases, the model will more accurately predict the parameters. I chose -3,3 as my grid space because -5,5 didn't show the dataset as well.



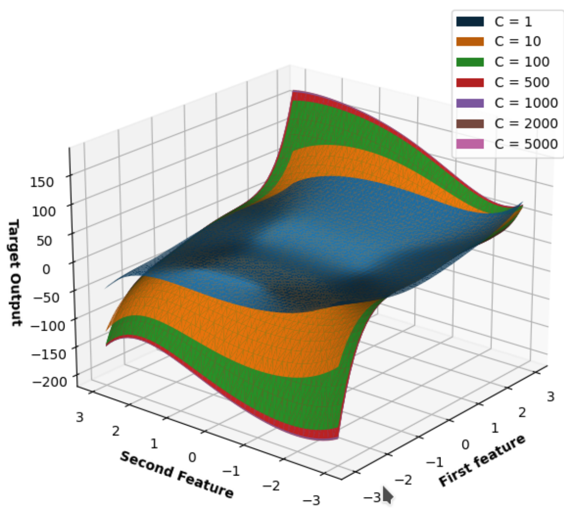


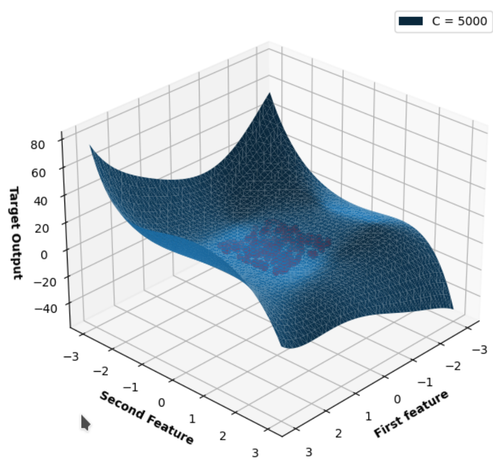
Q1e. (b)-(c)

	C	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
0	1	-0.031150	0.0	-0.003349	0.983522	0.918205	-0.101168	-0.006636	-0.091462	0.013535	0.024851	0.132780
1	10	-0.080048	0.0	0.062220	1.050498	1.282602	-0.164213	0.015060	-0.412815	-0.152006	0.012575	0.085713
2	100	-0.092640	0.0	0.124912	1.076822	1.371775	-0.178405	0.013197	-0.691649	-0.296744	-0.001747	0.050890
3	500	-0.094074	0.0	0.135164	1.080857	1.381611	-0.180008	0.012106	-0.736350	-0.320731	-0.004512	0.045766
4	1000	-0.094259	0.0	0.136544	1.081397	1.382874	-0.180215	0.011948	-0.742355	-0.323970	-0.004892	0.045092
5	2000	-0.094352	0.0	0.137243	1.081670	1.383509	-0.180320	0.011866	-0.745396	-0.325611	-0.005085	0.044752
6	5000	-0.094408	0.0	0.137665	1.081835	1.383891	-0.180382	0.011816	-0.747232	-0.326603	-0.005202	0.044547

	B11	B12	B13	B14	B15	B16	B17	B18	B19	B20	B21
0	0.122916	0.051876	0.002722	0.034161	0.012724	0.097841	0.012407	-0.121050	-0.126531	0.098521	-0.105497
1	-0.229987	0.081496	-0.131269	0.128123	0.039708	0.412875	0.193094	-0.222001	-0.157082	0.160354	-0.127479
2	-0.323068	0.068818	-0.161308	0.173671	0.064767	0.653472	0.337825	-0.192368	-0.117110	0.148581	-0.118166
3	-0.333580	0.065916	-0.164501	0.180687	0.069469	0.691102	0.360682	-0.183824	-0.108271	0.144749	-0.117485
4	-0.334934	0.065514	-0.164910	0.181629	0.070114	0.696144	0.363752	-0.182619	-0.107044	0.144204	-0.117417
5	-0.335615	0.065309	-0.165116	0.182106	0.070442	0.698696	0.365305	-0.182004	-0.106420	0.143926	-0.117384
6	-0.336024	0.065185	-0.165239	0.182394	0.070640	0.700237	0.366244	-0.181630	-0.106042	0.143758	-0.117365

The parameters obtained from the Ridge regression are shown above, with each C-value displayed on the left-most column. The Ridge regression model seems to tend to a value much faster than the Lasso regression model, probably because the L2 regularization is more volatile than L1 since instead of summing the absolute values of the parameters, it sums the squares of all parameters. Below are shown the 3d plots for the model in the same manner as for the Lasso regression. On the right, a surface that would predict more extreme values given larger features is shown.





APPENDIX

```
from re import X
import numpy as np
import pandas as pd
from sklearn.linear_model import Lasso
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from matplotlib.colors import ListedColormap
from mpl_toolkits import mplot3d
import json

data = open("data","r").read().split("\n")
dataset_id = data[0]
data = data[1:]
data = data[:-1]
print("DATASET ID: {}".format(dataset_id))
data = [a.split(",") for a in data]

# -- FORMAT DATA FOR FUTURE USE
f1 = []
f2 = []
features = []
target_outputs = []

for row in data:
   f1.append(float(row[0]))
   f2.append(float(row[1]))
   target_outputs.append(float(row[2]))
   features.append([float(row[0]),float(row[1])])
train_features,test_features,train_output,test_output =
train_test_split(features,target_outputs,test_size=0.25,random_state=0)

# --

# --- QUESTION ONE ---
#  -- a --
# Create 3D scatter plot of features on two axes with z axis being the
target output of feature vector.

# Creating figure
fig = plt.figure(figsize = (10, 7))
```

```
ax = plt.axes(projection ="3d")

# Creating colormap
colormap = plt.get_cmap("hsv")
# Creating plot
scatter_3d = ax.scatter3D(f1, f2, target_outputs, c=(target_outputs),
cmap=colormap)
fig.colorbar(scatter_3d, ax = ax, shrink = 0.5, aspect = 5)
ax.set_xlabel('First feature', fontweight ='bold')
ax.set_ylabel('Second Feature', fontweight ='bold')
ax.set_zlabel('Target Output', fontweight ='bold')
plt.title("3D Scatterplot of Dataset")

# show plot
# plt.show()

#  -- b --
# Add extra polynomial features equal to all combinations of powers up
to 5.


# --
poly_transform = PolynomialFeatures(5)
train_new_features = poly_transform.fit_transform(train_features)

# Set weights of C-value
C_weights = [1,10,100,500,1000,2000,5000]

# Train models with different C-values
def extend_list(a, b):
   a.extend(b)
   return a
parameters_and_model_from_lasso_regression =
[[x,extend_list([x[0]],extend_list([x[1].intercept_], x[1].coef_))] for
x in [[c,Lasso(alpha=1/(2*c)).fit(train_new_features,train_output)] for
c in C_weights]]

print(pd.DataFrame([x[1] for x in
parameters_and_model_from_lasso_regression],
columns=["C","B0","B1","B2","B3","B4","B5","B6",

"B7","B8","B9","B10","B11","B12",
```

```
"B13","B14","B15","B16","B17","B18",

"B19","B20","B21",]))

#  -- c --
#
models = [x[0] for x in parameters_and_model_from_lasso_regression]

Xtest = []
grid = np.linspace(-3,3)

for i in grid:
   for j in grid:
       Xtest.append([i,j])
Xtest_poly = poly_transform.fit_transform(Xtest)
predictions = [[model[0], model[1].predict(Xtest_poly)] for model in
models]

plt.clf()
plt.cla()

fig = plt.figure(figsize = (10, 7))
ax = fig.add_subplot(111, projection="3d")
ax.scatter3D(f1, f2, target_outputs, color="red")

index = 0
for model in predictions:
   label = "C = {}".format(model[0])
   surf = ax.plot_trisurf([x[0] for x in Xtest],[x[1] for x in
Xtest],predictions[index][1], label=label)
   surf._edgecolors2d = surf._edgecolor3d
   surf._facecolors2d = surf._facecolor3d
   index += 1

ax.legend()
plt.show()
```