Assignment Week 3

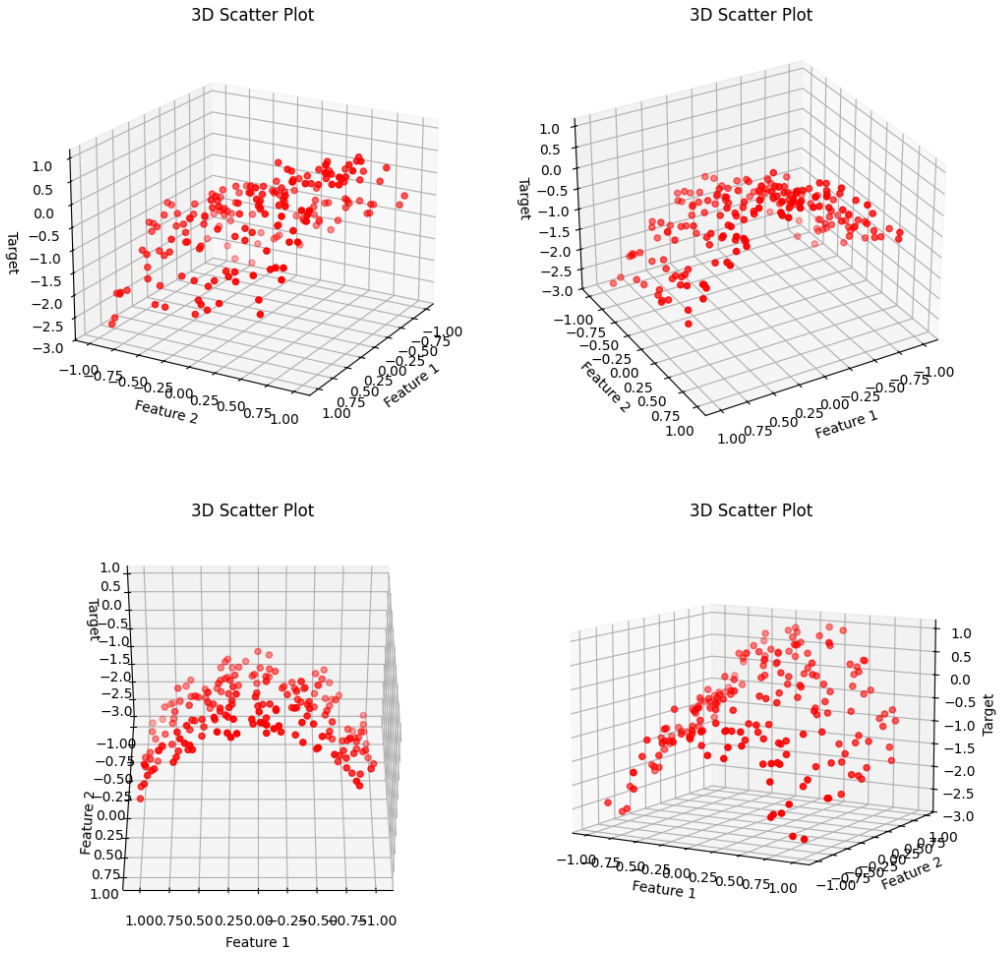
Student Name: Patrick Farmer Student Number: 20331828

Date: 2024-07-10 Dataset: 20–40-20

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

I(A)

The data was read in from the csv into X which held feature 1 and 2 and to Y which held the target. The data was then plotted onto a 3d scatter plot. We can see from the four different angles of the plot below that the data lies on a curve.



I(B)

Features up to the power of 5 were added to the dataset by using the PolynomialFeatures function in sklearn. Using this new data, a lasso regression model was trained with a number of different c values. The C Values chosen were (1, 10, 1000 and 10000). The value 1 was started with because it had all coefficients of 0. The value 10 was then chosen second as it demonstrated a significant improvement in the accuracy of the model with a mean square error of 0.7, there was only 2 coefficients that were non 0 in this case, both of which coefficients from feature 1. The value of C then chosen next was 1000, this is because this is when some more coefficients started to become non 0, all of the coefficients are still

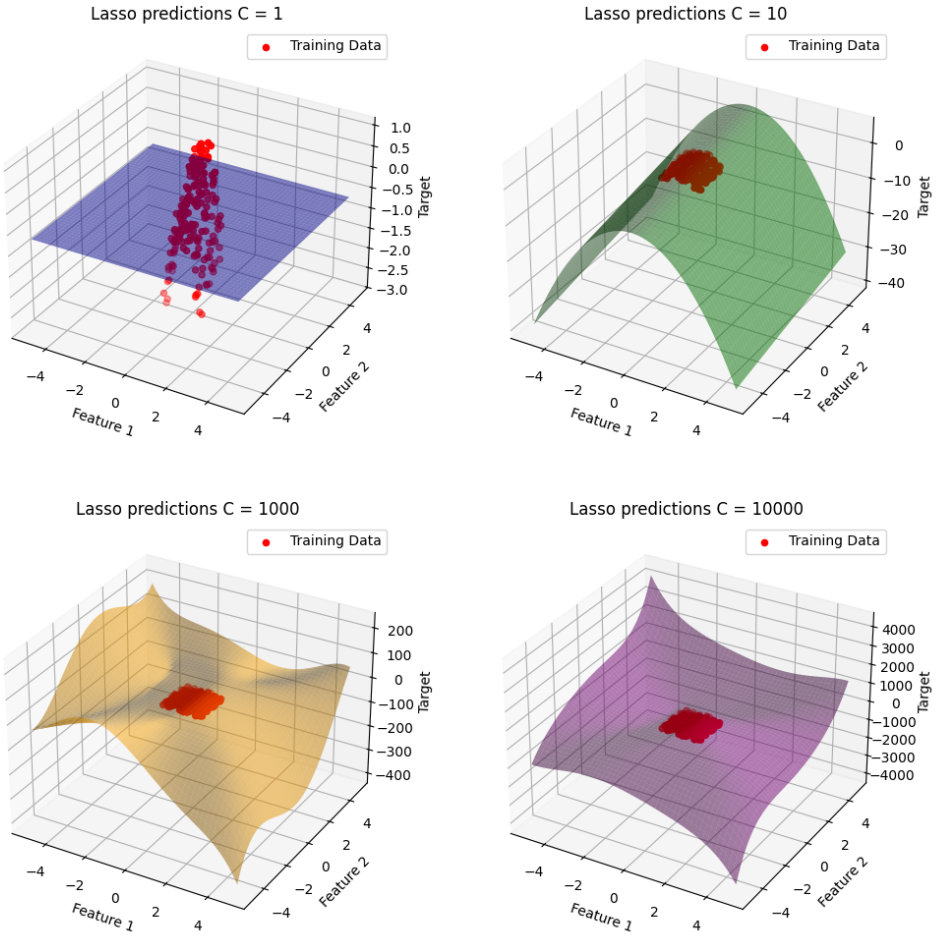
from feature 1 but it is every power of feature 1 (excluding power of 1) that has an impact on the target, there is a substantial difference again in the mean square error, it is half of the MSE for c=10. The final value of C chosen was 10000, this is to demonstrate the effect of over-fitting. There is more of an emphasis on the higher power of feature 1 and also of feature 2 in this case which means that it is likely to not generalise as well to new data. The mean square error is less on the training data (even though marginally) but it is likely to be higher on new data.
The results printed to terminal is as follows:

```
Lasso regression:
C: 1
Coefficients: [ 0. -0.  0. -0.  0.  0. -0.  0. -0.  0. -0. -0. -0.  0. -0. -0.  0. -0.
  0. -0.  0.]
Intercept: -0.6620942711094104
Mean square error: 0.7052321665957307


C: 10
Coefficients: [ 0.         -0.          0.84066564 -1.38608729  0.          0.
 -0.          0.         -0.          0.         -0.          0.
 -0.          0.         -0.         -0.          0.         -0.
  0.         -0.          0.        ]
Intercept: -0.20024920720097728
Mean square error: 0.07923880695629874


C: 1000
Coefficients: [ 0.          0.0360211   0.98093878 -1.91335315  0.02995874 -0.00369941
  0.          0.          0.         -0.          0.         -0.0711885
  0.          0.16497518  0.         -0.08311     0.01187887  0.12169111
  0.03993382 -0.07826324  0.        ]
Intercept: -0.018001936262590612
Mean square error: 0.042884824527337134


C: 10000
Coefficients: [ 0.         -0.03856134  1.13393801 -2.04536396  0.06532606 -0.08089148
  0.31691552 -0.29712296  0.1638204  -0.48722808  0.1330193  -0.21899704
  0.07159164  0.2654704   0.0782562  -0.40746191  0.20309085  0.28945988
  0.32091182 -0.37310009  0.33203645]
Intercept: 0.005803973720044486
Mean square error: 0.041553454785013544
```

I(C)

The previously trained models were then used to predict target values and plotted onto a 3d scatter plot with bounds exceeding the range of the training data. This was done using the plot_surface function in the matplotlib library. As mentioned with the coefficients we can see that the lower values of C have a more generalised model and the higher values of C have a more overfitted model. We can also see in the graph how with the final 2 models the model predictions get very erratic towards the bounds of the graph due to the higher emphasis on the higher powers of the features. We can also see that the graphs themselves change shape significantly as the value of C changes. The shape of C=1000 has much more peaks and troughs than c=10.

The graphs produced are as follows: **If hard to read, zoom in, quality should scale up**



I(D)

Under-fitting is when the model is too simple to capture the underlying structure of the data. This can be due to the model being too simple or over-regularised. The appropriate amount of regularisation will vary for each data set. In our dataset we can see an example of under-fitting with the model with C=1. This model has all coefficients set to 0 which means that it is just a flat plane at the intercept. This is not able to capture the underlying structure of the data.

Over-fitting is when the model is too complex and captures the noise in the data. This can be due to the model being too complex or under-regularised. The appropriate amount of regularisation will vary for each data set. In our dataset we can see an example of over-fitting with the model with C=10000 and to a lesser extent with C=1000. We can see over-fitting in the graph itself with the complex plan

but we can also see it in the coefficients when the higher power features become non zero. This is likely to not generalise well to new data.

C is used to manage the balance of over-fitting and under-fitting. Generally the best option is to choose the lowest value of C that matches the trend of the data. When it is difficult to determine the trend a good option is to pick a value of C where the increase is accuracy begins to slow down. This is because the model is likely to be over-fitting at this point.

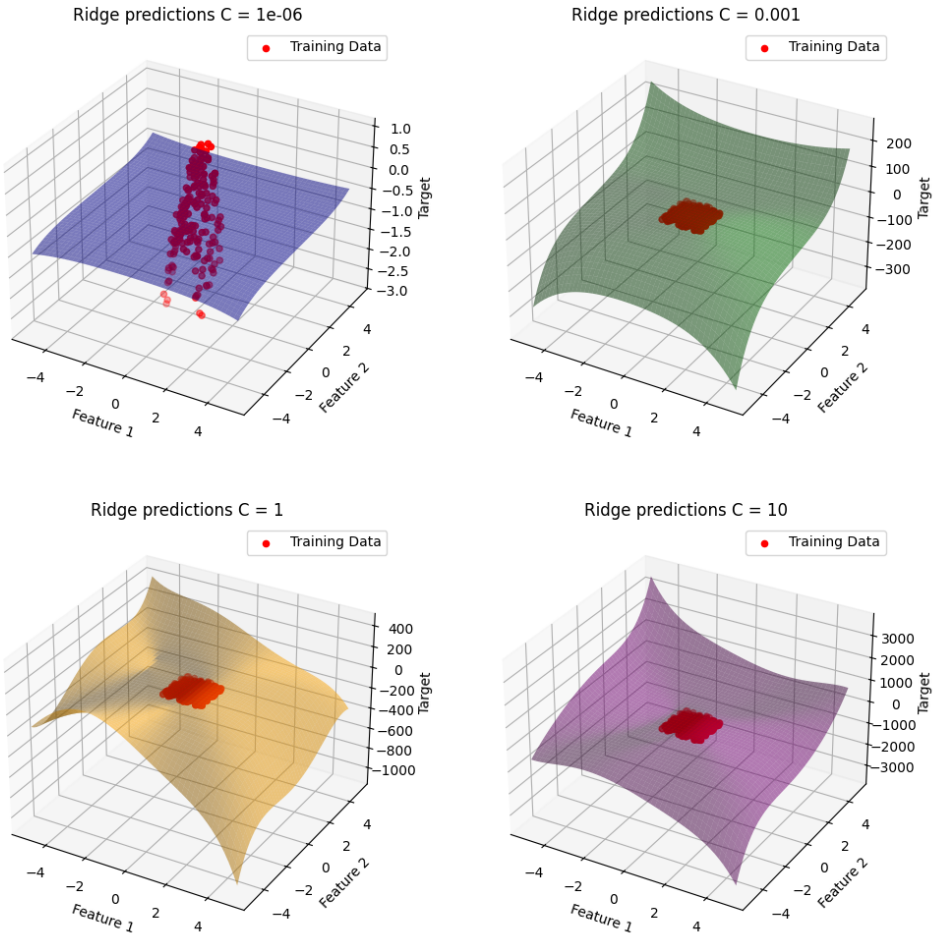
I(E)

The same code that was applied for the lasso regression was applied to the ridge regression. The main difference between the two is that the lasso regression uses the L1 penalty and the ridge regression uses the L2 penalty. The effect of these two penalties is that the L1 penalty will set some coefficients to 0 whereas the L2 penalty will make the coefficients very small but not 0. This is why we will see more non zero coefficients in the ridge regression log than the lasso regression log. The values for C chosen were (0.000001, 0.001, 1, 10). For C=0.000001 we can see that the model looks very much like the c=1 model for the lasso regression, there is a slight curve in the model though due to the higher power coeffcients not being set 0 like the l1 penalty. For C=0.001 we can see that the model is very similar to the c=10 model for the lasso regression, the model captures the underlying structure of the data well which is shown by the more simple shape of the model and the lower values for the higher power coefficients printed to terminal. For C=1 and C=10 we can see the model begin to over-fit with the higher power coefficients become larger.
The results printed to terminal is as follows:

```
C: 1e-06
Coefficients: [ 0.00000000e+00 -3.68539326e-06  1.25467042e-04 -7.16607284e-05
  4.04327809e-06  1.97488256e-06 -3.94764156e-06  3.87222468e-05
 -5.80318550e-06  7.35108561e-05 -6.18100722e-05 -9.90618677e-07
 -2.17470833e-05  6.37818285e-06 -4.53024537e-07 -3.26097580e-06
  2.25031329e-05 -3.59609353e-06  2.40383736e-05 -6.66391248e-06
  5.19219772e-05]
Intercept: -0.6620566744425684
Mean square error: 0.7050510098216829


C: 0.001
Coefficients: [ 0.         -0.00193413  0.10402134 -0.06686939  0.00308726  0.00230748
 -0.00304114  0.03132703 -0.00414035  0.06017536 -0.05759894 -0.001298
 -0.0198678   0.00541079  0.00012678 -0.00281455  0.0180354  -0.00269429
  0.019386   -0.00497366  0.04219861]
Intercept: -0.6271765744013319
Mean square error: 0.5622856599311535


C: 1
Coefficients: [ 0.          0.03473047  0.95358618 -1.54597304  0.06691228 -0.00833979
  0.02532275 -0.00498428  0.05975658  0.01886483 -0.34373703 -0.09511995
 -0.10553073  0.14488172  0.04376759 -0.1236849   0.04996693  0.16170927
  0.03019935 -0.1776871   0.01671556]
Intercept: -0.06702152065433697
Mean square error: 0.044051787163365094
```

```
C: 10
Coefficients: [ 0.         -0.02648752  1.09561279 -1.97639628  0.07103039 -0.08391659
  0.26208839 -0.24880538  0.16975252 -0.35747644  0.06456599 -0.21047785
  0.04975984  0.24879627  0.08657142 -0.35688757  0.18496206  0.27626565
  0.25645857 -0.37932223  0.23825872]
Intercept: -0.0022578456765575128
Mean square error: 0.04166857632646202
```

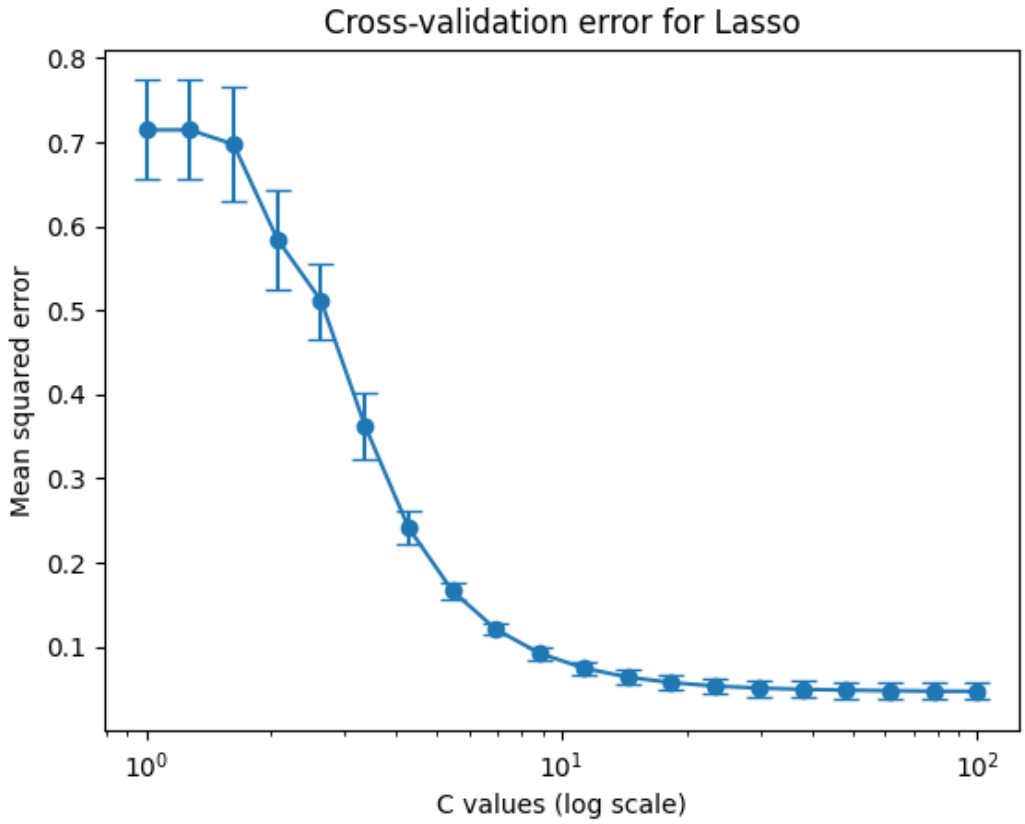
The graphs produced are as follows:



II(A)

For this section Cross-Validation was used to plot the mean and standard deviation of the mean squared error of a range of values for C for the lasso regression. The prediction error chosen is mean squared error. The range of values for C chosen was 1 to 100 with 20 data points taken at logarithmic intervals. This produced a clear graph where we can a clear drop off in the rate of decrease of the mean squared error as C increases. Another observation to be made is as C increases the standard deviation of the mean squared error also increases. This is another byproduct of over-fitting.

The graph produced is as follows:



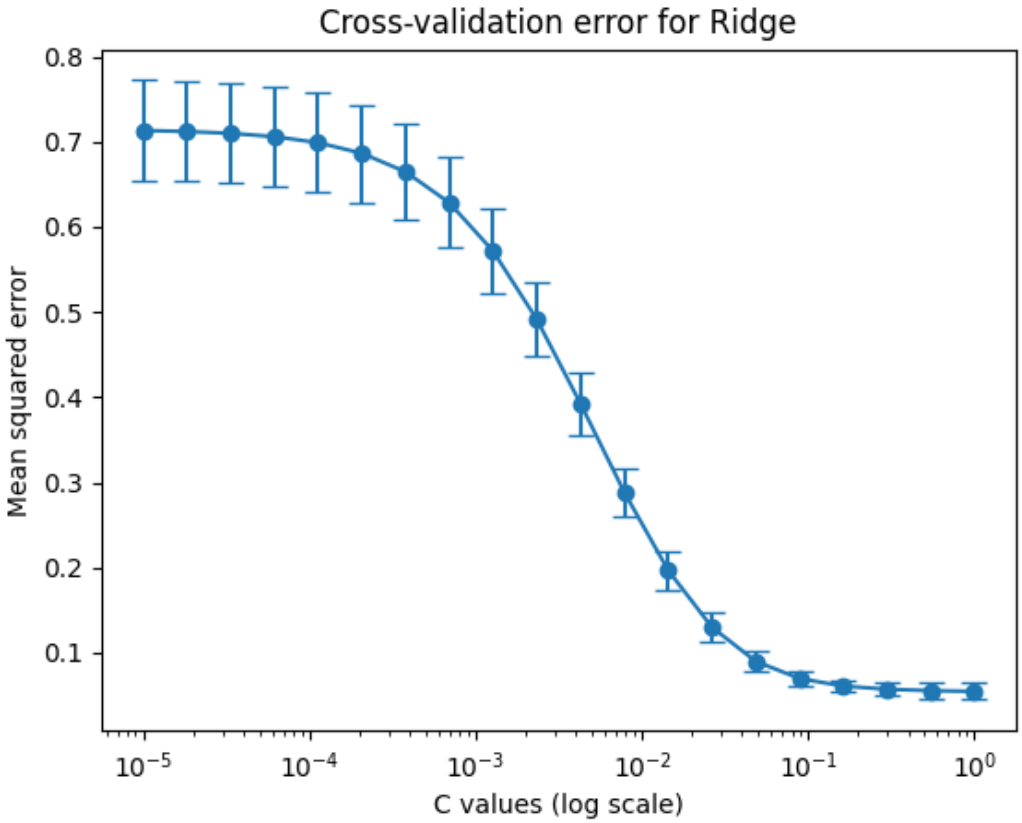
II(B)

Taking the cross validation error we can see a range of values of C that will be likely to produce a good model. The recommendation that I would make is to choose 10.5 as the value of C. This is because it is the lowest value of C that has a mean squared error that is only marginally higher than the lowest. It also maintains a low standard deviation of the mean squared error. This is likely to generalise well to new data.

II(C)

The same code that was applied for the lasso regression was applied to the ridge regression. The same prediction error scoring was chosen as the mean squared error. The range of values for C chosen was 10^{-5} to 1 as this shows the most significant change in the mean squared error and provides a clear

graph for choosing an optimal value for C. The graph produced is as follows:



From this graph I would recommend choosing a value of C of 0.11 for the same reasons as the lasso regression. This is because it is the lowest value of C that has a mean squared error that is only marginally higher than the lowest. It also maintains a low standard deviation of the mean squared error.

Appendix

I(A)

```
def load_data(file_name):
    file_path = os.path.join(os.path.dirname(__file__), file_name)
    column_names = ['Feature1', 'Feature2', 'Target']
    return pd.read_csv(file_path, names=column_names, skiprows=1)

def plot_3d_scatter(data, angles, output_file):
    fig, axes = plt.subplots(2, 2, subplot_kw={'projection': '3d'}, figsize=(12, 12))
    for ax, (elev, azim) in zip(axes.flatten(), angles):
        ax.scatter(data['Feature1'], data['Feature2'], data['Target'], color='red')
        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_zlabel('Target')
        ax.set_title(f'3D Scatter Plot')
        ax.view_init(elev=elev, azim=azim)
```

```
    plt.savefig(output_file)
    plt.close()

data = load_data('week3.csv')
angles = [(20, 30), (30, 60), (40, 90), (10, -60)]
plot_3d_scatter(data, angles, 'i(a).png')
```

I(B/C/D/E)

```
def regression_analysis(model_class, C_values, png_name):
    data = load_data('week3.csv')
    X = data[['Feature1', 'Feature2']].values
    y = data['Target'].values

    poly = PolynomialFeatures(degree=5)
    X_poly = poly.fit_transform(X)

    for C in C_values:
        model = model_class(alpha=1/(2*C), max_iter=10000)
        model.fit(X_poly, y)

        print(f'C: {C}')
        print(f'Coefficients: {model.coef_}')
        print(f'Intercept: {model.intercept_}')
        error = mean_squared_error(y, model.predict(X_poly))
        print(f'Mean square error: {error}')
        print("\n")

    plot_regression_results(model_class, C_values, X, y, X_poly, poly, png_name)

def plot_regression_results(model_class, C_values, X, y, X_poly, poly, png_name):
    X_test = []
    for i in np.linspace(-5, 5):
        for j in np.linspace(-5, 5):
            X_test.append([i, j])
    X_test = np.array(X_test)
    X_test_poly = poly.transform(X_test)

    fig, axes = plt.subplots(1, 4, subplot_kw={'projection': '3d'}, figsize=(24, 6))
    colors = ['blue', 'green', 'orange', 'purple']

    for ax, C, color in zip(axes, C_values, colors):
        model = model_class(alpha=1/(2*C), max_iter=10000)
        model.fit(X_poly, y)
        predictions = model.predict(X_test_poly).reshape((50, 50))

        ax.plot_surface(X_test[:, 0].reshape((50, 50)), X_test[:, 1].reshape((50, 50)), predictions,
        ax.scatter(X[:, 0], X[:, 1], y, color='red', label='Training Data')

        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_zlabel('Target')
```

```
        ax.set_title(f'{model_class.__name__} predictions C = {C}')
        ax.legend()

print("\033[91m" + "Lasso regression:" + "\033[0m")
C_values = [1, 10, 1000, 10000]
regression_analysis(Lasso, C_values, 'i(c).png')

print("\033[91m" + "Ridge regression:" + "\033[0m")
C_values = [0.000001, 0.001, 1, 10]
regression_analysis(Ridge, C_values, 'i(e).png')
```

II(A)

```
def cross_validation_analysis(model_class, C_values, png_name):
    data = load_data('week3.csv')
    X = data[['Feature1', 'Feature2']].values
    y = data['Target'].values

    poly = PolynomialFeatures(degree=5)
    X_poly = poly.fit_transform(X)

    initial_c, final_c = find_meaningful_c_range(model_class, C_values, X_poly, y)
    if initial_c and final_c:
        spaced_c_values = np.logspace(np.log10(initial_c), np.log10(final_c), 20)
        plot_cross_validation_error(model_class, spaced_c_values, X_poly, y, png_name)

def find_meaningful_c_range(model_class, C_values, X_poly, y):
    previous_mean_error = None
    threshold = 0.01
    meaningful_data_found = False
    initial_c = None
    final_c = None

    for C in C_values:
        model = model_class(alpha=1/(2*C), max_iter=10000)
        scores = cross_val_score(model, X_poly, y, cv=5, scoring='neg_mean_squared_error')
        mean_error = -scores.mean()

        if previous_mean_error is not None:
            error_difference = abs(previous_mean_error - mean_error)
            if error_difference < threshold:
                if meaningful_data_found:
                    final_c = previous_c
            else:
                if meaningful_data_found:
                    pass
                else:
                    initial_c = previous_c
                meaningful_data_found = True
        previous_c = C
        previous_mean_error = mean_error
```

```
    return initial_c, final_c

def plot_cross_validation_error(model_class, C_values, X_poly, y, png_name):
    mean_errors = []
    std_errors = []

    for C in C_values:
        model = model_class(alpha=1/(2*C), max_iter=10000)
        scores = cross_val_score(model, X_poly, y, cv=5, scoring='neg_mean_squared_error')
        mean_errors.append(-scores.mean())
        std_errors.append(scores.std())

    plt.figure()
    plt.errorbar(C_values, mean_errors, yerr=std_errors, fmt='o-', capsize=5)
    plt.xscale('log')
    plt.xlabel('C values (log scale)')
    plt.ylabel('Mean squared error')
    plt.title(f'Cross-validation error for {model_class.__name__}')

    plt.savefig(os.path.join(os.path.dirname(__file__), png_name))
    plt.close()

C_values = [10**i for i in range(-7, 5)]
cross_validation_analysis(Lasso, C_values, 'ii(a).png')
cross_validation_analysis(Ridge, C_values, 'ii(c).png')
```