Name: Shane Bowen
ID: R00149085
Class: SDH4-A

Task 1

```
Type of Cuts: ['Ideal' 'Premium' 'Good' 'Very Good' 'Fair']
Type of Colors: ['E' 'I' 'J' 'H' 'F' 'G' 'D']
Type of Clarity: ['SI2' 'SI1' 'VS1' 'VS2' 'VVS2' 'VVS1' 'I1' 'IF']
('Ideal', 'D', 'VS2')
Num Datapoints: 920
('Ideal', 'E', 'VS2')
Num Datapoints: 1136
('Ideal', 'F', 'VS2')
Num Datapoints: 879
('Ideal', 'G', 'VS1')
Num Datapoints: 953
('Ideal', 'G', 'VS2')
Num Datapoints: 910
```

In Task 1, I display all the unique types of cuts, colors and clarity in the data. I group the data by the unique combinations and store the data points in a dictionary. I do a count on the number of datapoints for each unique combination and if it is greater than 800, then I use if for further processing.

```
******** Features *******
{('Ideal', 'D', 'VS2'): [[0.7, 61.0, 57.0], [0.7, 61.0, 57.0], [0.7,
62.4, 57.0], [0.7, 61.3, 57.0], [0.7, 60.3, 60.0], [0.7, 62.8, 57.0],
[0.71, 60.4, 53.0], [0.74, 61.9, 56.0], [0.7, 61.6, 58.0], [0.7, 62.3,
55.0], [0.72, 61.8, 57.0], [0.7, 61.5, 56.0], [0.71, 61.2, 56.0],
[0.71, 60.4, 57.0], [0.71, 62.3, 56.0], [0.71, 62.0, 57.0], [0.71,
62.5, 55.0], [0.74, 61.1, 54.0], [0.72, 60.8, 56.0], [0.72, 62.1,
57.0], [0.72, 62.8, 57.0], [0.73, 62.1, 56.0], [0.71, 62.7, 57.0],
[0.71, 62.5, 55.0], [0.72, 62.7, 56.0], [0.72, 62.7, 56.0], [0.71,
61.4, 57.0], [0.71, 61.4, 56.0], [0.72, 60.9, 55.0], [0.71, 63.0,
56.0], [0.76, 63.0, 56.0], [0.73, 61.9, 56.0], [0.72, 60.5, 57.0],
[0.76, 60.2, 56.0], [0.71, 61.5, 55.0], [0.76, 61.8, 57.0], [0.73,
61.8, 54.0], [0.71, 62.1, 56.0], [0.71, 61.2, 57.0], [0.73, 61.5,
```

```
******** Targets *******
{('Ideal', 'D', 'VS2'): [2833.0, 2859.0, 2867.0, 2960.0, 2991.0,
2998.0, 3020.0, 3087.0, 3092.0, 3092.0, 3099.0, 3129.0, 3135.0, 3135.0,
3137.0, 3153.0, 3161.0, 3177.0, 3179.0, 3179.0, 3179.0, 3182.0, 3212.0,
3217.0, 3219.0, 3219.0, 3222.0, 3234.0, 3236.0, 3245.0, 3248.0, 3255.0,
3275.0, 3293.0, 3299.0, 3306.0, 3319.0, 3321.0, 3321.0, 3346.0, 3352.0,
3360.0, 3372.0, 3384.0, 3401.0, 3412.0, 3440.0, 3442.0, 3443.0, 3448.0,
3450.0, 3454.0, 3456.0, 3464.0, 3467.0, 3487.0, 3495.0, 3528.0, 3540.0,
3543.0, 3562.0, 3595.0, 3625.0, 3668.0, 3673.0, 3679.0, 3721.0, 3735.0,
3736.0, 3755.0, 3763.0, 3788.0, 3811.0, 3818.0, 3821.0, 3837.0, 3838.0,
3858.0, 3864.0, 3980.0, 3980.0, 4029.0, 4029.0, 4029.0, 4029.0, 4029.0,
4082.0, 4082.0, 4087.0, 4128.0, 4134.0, 4134.0, 4170.0, 4183.0, 4315.0,
4327.0, 4378.0, 4628.0, 4628.0, 4781.0, 5897.0, 606.0, 606.0, 6444.0,
6513.0, 6689.0, 6856.0, 6981.0, 6999.0, 7127.0, 7220.0, 616.0, 616.0,
7602.0, 7602.0, 7602.0, 7697.0, 7792.0, 621.0, 7916.0, 7965.0, 8008.0,
```

I split the data into features and target, in the features it contains the carat, depth, table and in the target it contains the price. This data is stored in a dictionary where the key is the combination, so it can make for easy access later on.

Task 2

```python
def num_coefficients_3(d):
    t = 0
    for n in range(d+1):
        for i in range(n+1):
            for j in range(n+1):
                for k in range(n+1):
                    if i+j+k==n:
                        t = t+1
    return t

def calculate_model_function3(deg, feature, p):
    result = np.zeros(feature.shape[0])
    m=0
    for n in range(deg+1):
        for i in range(n+1):
            for j in range(n+1):
                for k in range(n+1):
                    if i+j+k==n:
                        result += p[m]*(feature[:,0]**i)*(feature[:,1]**j)*(feature[:,2]**k)
                        m+=1
    return result
```

In Task 2, I create my polynomial model function that will calculate the estimated target vector. It pass in the parameters degree, the features and parameter vector. I return the result. I also calculate the number of coefficients for a three variate coefficient. I pass in the degree and return the result.

Task 3

```
def linearize3(deg, feature, p0):
    f0 = calculate_model_function3(deg, feature, p0)
    J = np.zeros((len(f0), len(p0)))
    epsilon = 1e-6

    for i in range(len(p0)):
        p0[i] += epsilon
        fi = calculate_model_function3(deg, feature, p0)
        p0[i] -= epsilon
        di = (fi - f0)/epsilon
        J[:,i] = di

    return f0,J
```

In Task 3, I created a linearization function which calculates the value of the model and the jacobian. I pass in the degree, features and the coefficient as parameters.

Task 4

```
#task4
def calculate_update(y,f0,J):
    l=1e-2
    N = np.matmul(J.T,J) + l*np.eye(J.shape[1])
    r = y-f0
    n = np.matmul(J.T,r)
    dp = np.linalg.solve(N,n)
    return dp
```

In Task 4, I calculate the optimal update from the training target vector and estimated target vector. I calculate the normal equation matrix, calculate the residual, calculate normal equation system and then solve for the optimal parameter update vector

Task 5

```
#task5
def regression(deg, feature, target):
    max_iter = 5
    p0 = np.zeros(num_coefficients_3(deg))
    for i in range(max_iter):
        f0,J = linearize3(deg,feature, p0)
        dp = calculate_update(target,f0,J)
        p0 += dp

        test_target = calculate_model_function3(deg,feature, p0)
        return(test_target)
```

In Task 5, I create a regression function that will iteratively find the best fit coefficient vector in the training data. I pass in degree, feature and target as the parameters. I return the best fitting coefficient.

Task 6

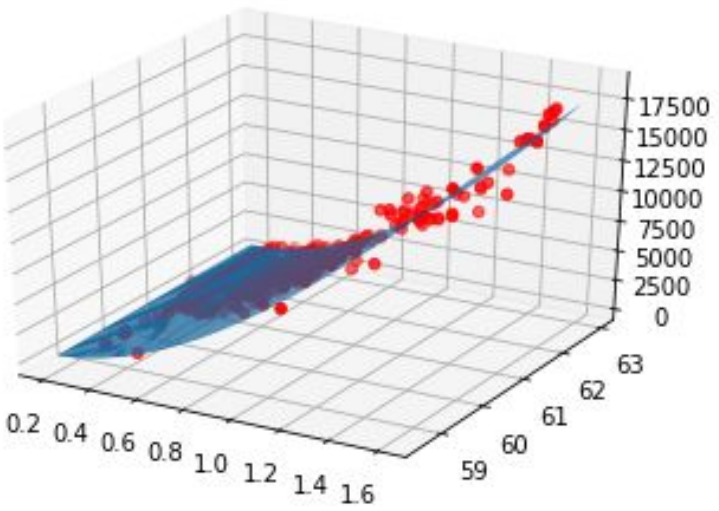
```
Actual:  [1854.]
Prediction:  [1853.99726769]
Difference:  [0.00273231]


******** ('Ideal', 'D', 'VS2') *********
Mean Diff Deg  0 : 5.229368329783647
Mean Diff Deg  1 : 5.229013104184544
Mean Diff Deg  2 : 5.228657848196922
Mean Diff Deg  3 : 5.2283026404626725
The optimal degree is: 3
```

In Task 6, I determine the best polynomial degree for each different combination. I create a k-fold that iterates through the data, I then iterate through the degrees from 0 to 3, I use mean absolute price difference to get the difference between the prediction and actual price, the lowest absolute mean difference is the most optimal degree.

Task 7

```
******** Price Estimation *******
p[0]= -64.42443916611809  ± 3296.2966246916226
p[1]= -753.0718174103856  ± 881.3399092120457
p[2]= -321.212306967004  ± 3297.5587293020467
p[3]= 23.197467951965844  ± 28.33431615157102
p[4]= 380.4765269118831  ± 145.44924942478113
p[5]= -13219.15712348857  ± 3128.896996276477
p[6]= -0.1731864789669284  ± 0.22997571718747242
p[7]= -7.272514319337588  ± 1.8167549428092913
p[8]= 411.6901218993471  ± 52.703574640048885
p[9]= -1940.6005421853774  ± 358.84867263278727
```



In Task 7, I visualize the results, I print the price estimates by using the most optimal degree I found in task 6. It shows the estimated price differential from that price. You can see from the screenshot that p[6] is the most closest estimate to the actual value. I then plot the data-points in a 3d graph. I show the estimated prices against the true sale prices and showing the relation of the target vector and the feature vectors.