CS 6375
ASSIGNMENT ____1_____

Names of students in your group:
Nikhil Manda, nxm180037
Shanmukha Bodapati, ssb180006

Number of free late days used: _____0_____

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.

- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor
- https://scikit-learn.org/stable/modules/classes.html#
- https://www.geeksforgeeks.org/vectorization-of-gradient-descent/
- In-class lectures, slides, and code examples

Linear Regression using Gradient Descent Results Analysis

Dataset Information:
The dataset is called Wine Quality. It includes two csv files, one for red wines and one for white wines. We ended up using the white wines dataset which had 4898 instances and 12 attributes of the data. In addition, there is a class imbalance in both files where there are more than normal wines compared to excellent or poor wines. For that reason since there was more data about white wines than red wines as well as a stronger class imbalance in the red wines, we decided to go investigate the feature selection of white wines.

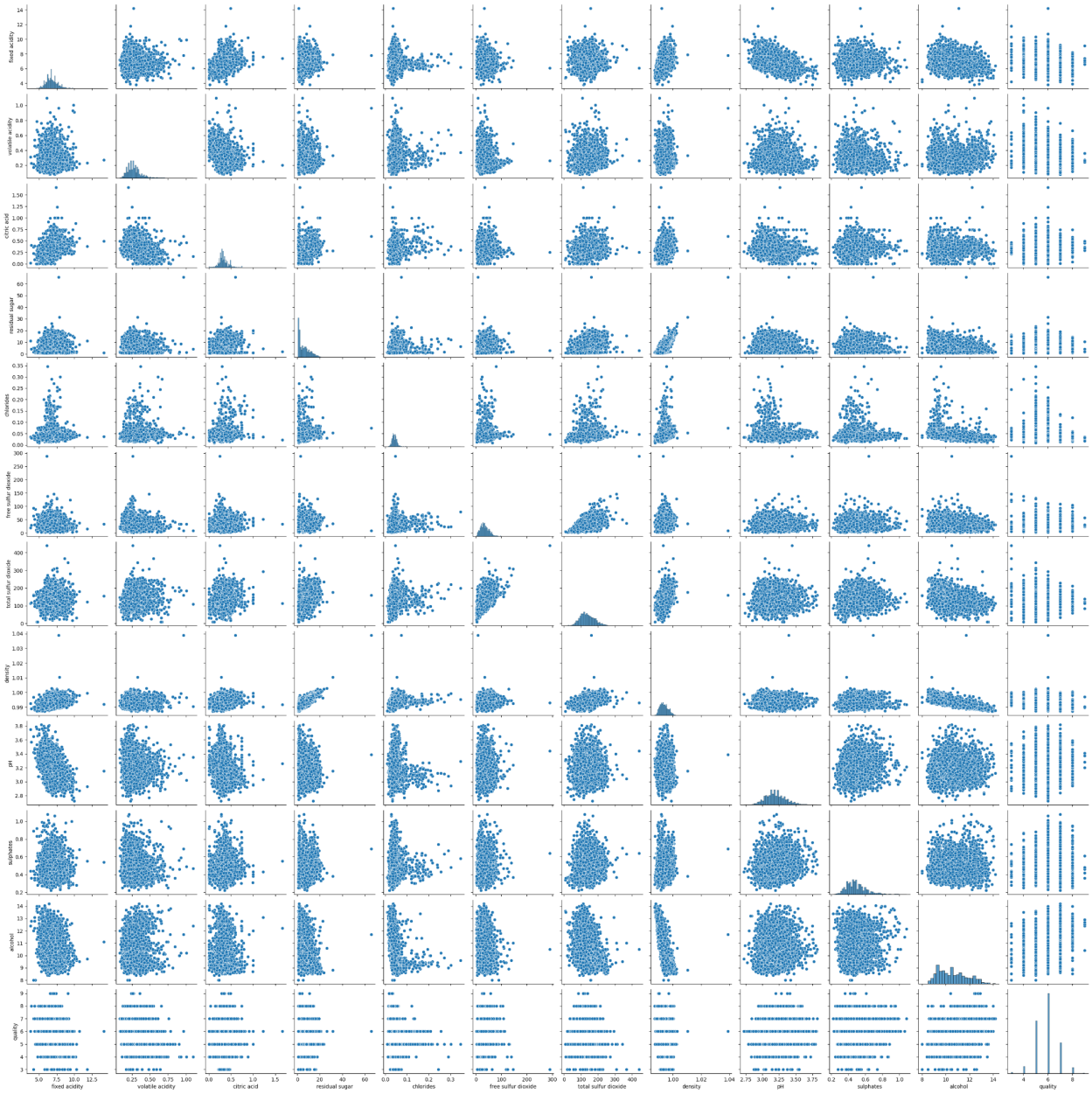
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3961 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         3961 non-null   float64
 1   volatile acidity      3961 non-null   float64
 2   citric acid           3961 non-null   float64
 3   residual sugar        3961 non-null   float64
 4   chlorides             3961 non-null   float64
 5   free sulfur dioxide   3961 non-null   float64
 6   total sulfur dioxide  3961 non-null   float64
 7   density               3961 non-null   float64
 8   pH                    3961 non-null   float64
 9   sulphates             3961 non-null   float64
 10  alcohol               3961 non-null   float64
 11  quality               3961 non-null   int64  
dtypes: float64(11), int64(1)
```

In our case, we used the density as the target variable, and residual sugar, total sulfur dioxide, and alcohol as the predictor variables. All attributes were numeric data types. Below is the first few rows of data:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
6	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.9949	3.18	0.47	9.6	6

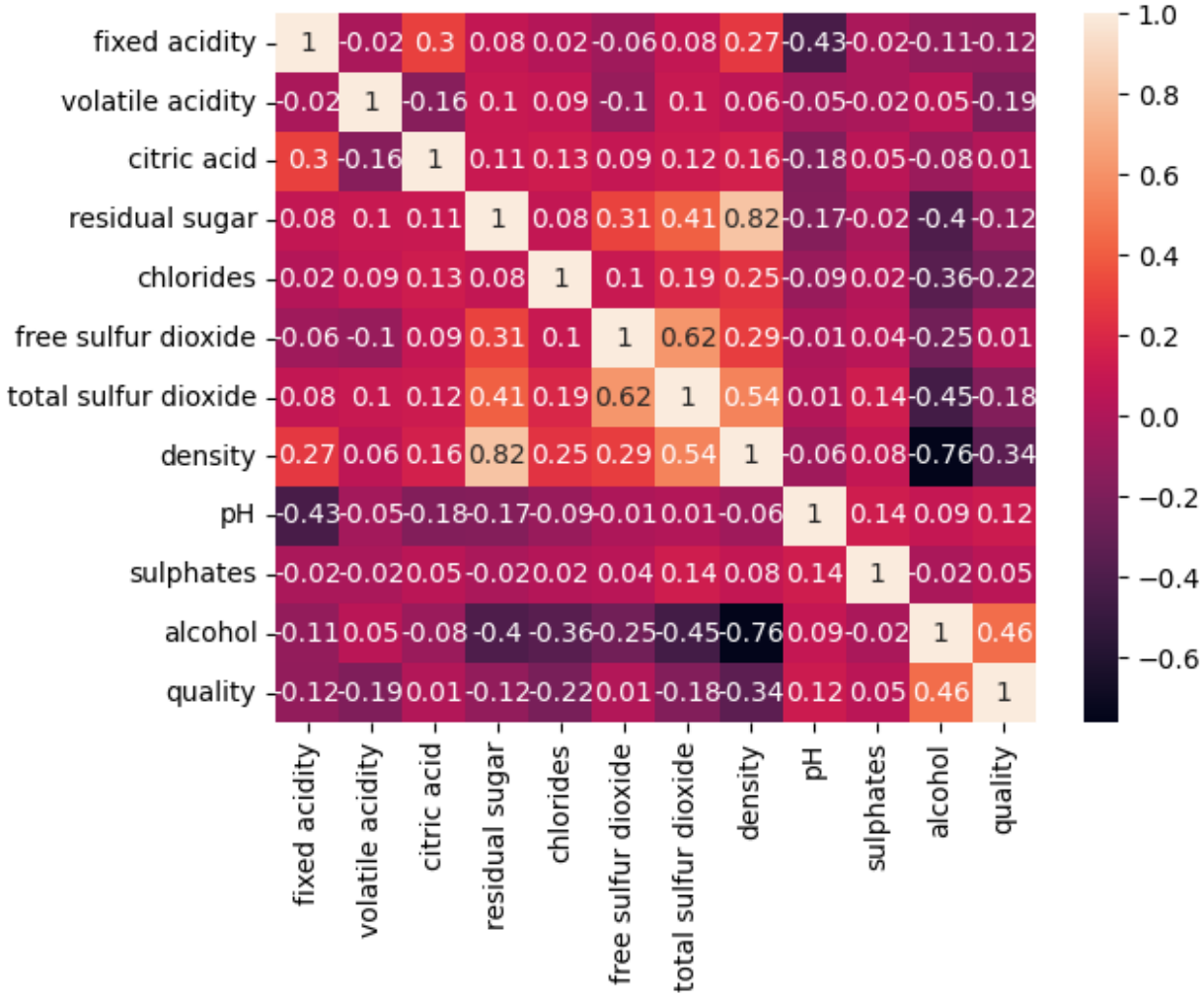
Data Preprocessing:
The first thing we did was to check for missing, N/A, and null values. None existed. Next, we removed duplicate rows, bringing the number of instances down to 3961.

Next, to determine which attributes we would use as the independent variables and which variable would be the target, we created a pairplot, correlation matrix, and heatmap of the correlation matrix. The pairplot is shown below:



(Note: due to the amount of attributes it is slightly hard to see the names of each attribute).

We noticed that there seemed to be a positive linear correlation between density and residual sugar, density and total sulfur dioxide, and a negative correlation between density and alcohol. To corroborate our thinking, we used a correlation matrix and heatmap.



As the heatmap shows, our thinking is correct. Alcohol and density seem to be strongly, negatively linear correlated while residual sugar and total sulfur dioxide are moderately, positively linear correlated. So, we decided to use **alcohol, residual sugar, and total sulfur dioxide** as our independent variables and **density** as the target variable.

Next, we removed any outliers (defined as points outside 1.5*IQR), standardized the attributes so that they are all on a similar scale using the StandardScaler, and finally randomly split the data into training and testing sets with 80% for training and 20% for testing.

Part 1 (Linear Regression using Gradient Descent from Scratch)

Our gradient descent algorithm will be looking to minimize the given error function:

$$J = \frac{1}{2n}\sum_{i=1}^{n}(\text{Predicted Value} - \text{Actual Value})^2$$

Our predicted value will use a linear function of the following form:
$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$
Therefore, our error function will look like this:

$$J = \frac{1}{2n}\sum_{i=1}^{n}[(w_0 + w_1x_1 + w_2x_2 + w_3x_3) - y_a]^2$$

where w_o is the bias term representing our y-intercept, $w_{1..3}$ are weights for each variable $x_{1...3}$. Technically, x_0 is also a variable for the bias term, but is just 1 so we don't explicitly show it. To compute the gradients using the error equation, we use the following partial derivatives:

$$\frac{\partial J}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n}[(w_0 + w_1x_1 + w_2x_2 + w_3x_3) - y_a]$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n}x_1[(w_0 + w_1x_1 + w_2x_2 + w_3x_3) - y_a]$$

$$\frac{\partial J}{\partial w_2} = \frac{1}{n}\sum_{i=1}^{n}x_2[(w_0 + w_1x_1 + w_2x_2 + w_3x_3) - y_a]$$

$$\frac{\partial J}{\partial w_3} = \frac{1}{n}\sum_{i=1}^{n}x_3[(w_0 + w_1x_1 + w_2x_2 + w_3x_3) - y_a]$$

Before we do the gradient descent algorithm, we convert our data into numpy arrays (really matrices) so we can do vectorized operations. Vectorizing the gradient descent algorithm greatly speeds up the process. We also add a column of ones to the X data representing $x_0 = 1$.

We then create 2 functions, one to run the gradient descent algorithm and one to calculate the MSE. Our gradient descent algorithm takes in the X data (attributes), Y data (actual target values), the initial weights (coefficients), learning rate, number of iterations to run the algorithm for, and tolerance threshold to stop the algorithm. The function to calculate MSE takes in X data, actual Y, and the final weights (coefficients) which then calculates the predicted values and MSE.

We define multiple values for each parameter to determine and run the algorithm with each combination to find the best value. We also find the training and testing MSE for each combination. We write the parameters and MSE to a log file which looks like below.

Learning Rate	Number of Iterations	Tolerance	Initial Coefficients	Training MSE	Testing MSE
0.001	10	1e-05	[[0.0], [0], [0.0], [0.0]]	0.0000023	0.0000022
0.001	10	1e-05	[[0.0001], [0.0001], [0.0001], [0.0001]]	0.1201006	0.1186574
0.001	10	1e-05	[[0.001], [0.001], [0.001], [0.001]]	0.4794791	0.4736510
0.001	10	1e-06	[[0.0], [0], [0.0], [0.0]]	0.0000023	0.0000022
0.001	10	1e-06	[[0.0001], [0.0001], [0.0001], [0.0001]]	0.1201006	0.1186574

There are 2 additional steps we added to the gradient descent algorithm in order to obtain better results. Initially, we faced a massive exploding gradient problem. To remedy it, we applied a batch normalization technique and normalized the theta (weight or coefficient) values after every step. Furthermore, due to the normalization, the weight for the bias (y-intercept) tended to be weighted more than necessary so regularization terms were applied at the very end to account for these differences.

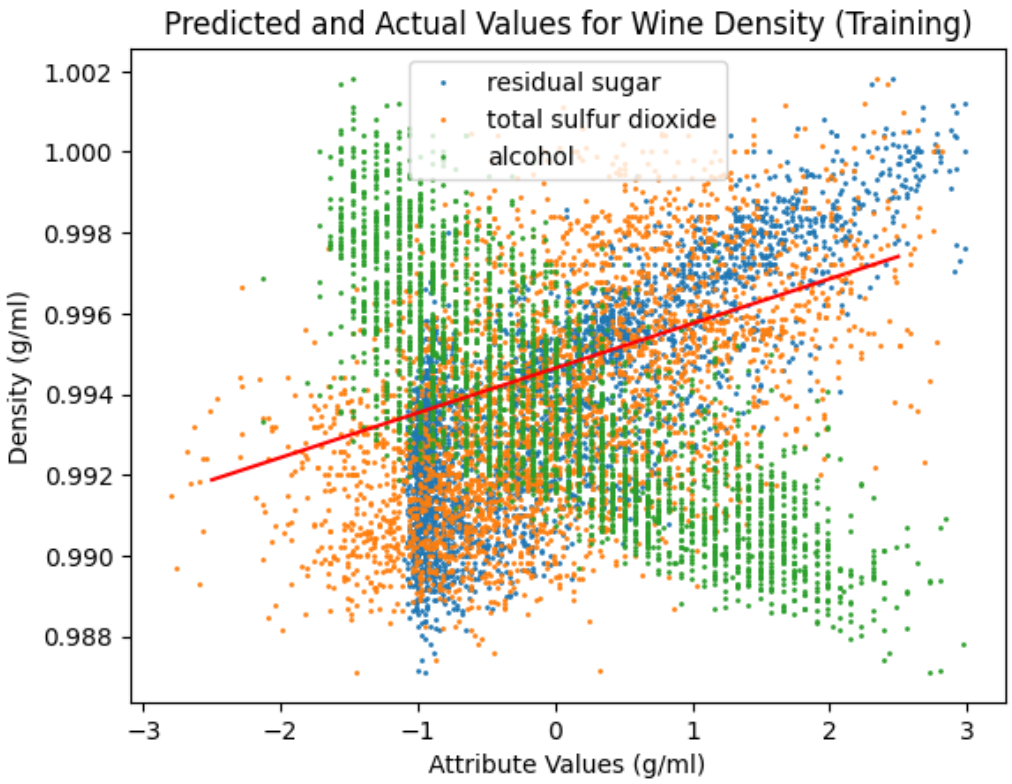
Out of all the iterations, we find the best combination of parameters by finding the lowest average MSE (of training and testing MSE). We find the average because we don't only rely on the training error. We found that the best model has the following parameters:
Learning rate: 0.001
Num iterations: 1000
Tolerance: 1e-05
Initial weights: [[0.0], [0], [0], [0]]
Training MSE: 1.652e-06
Testing MSE: 1.587e-06

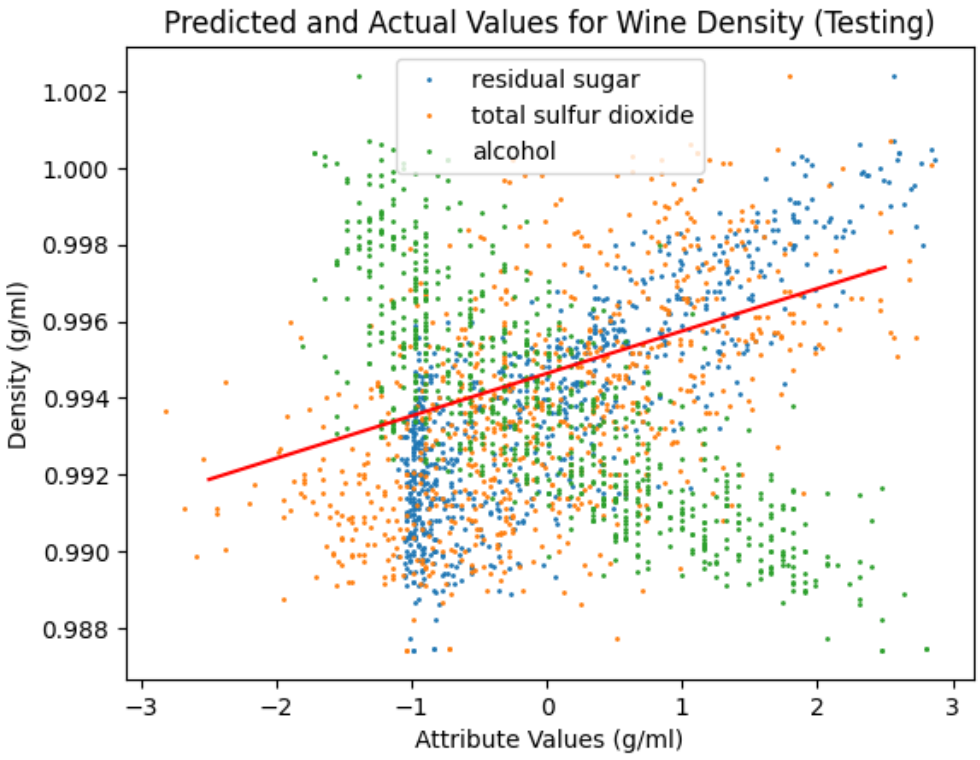
Then we applied these parameters to the algorithm, but now also calculated the training and testing MSE at every step:

Clearly, the MSE decreases significantly at each iteration and starts to flatten out around 1000 iterations. It is better we stop at 1000 iterations so we don't overtrain the model. The coefficients for the best model turned out to be 0.994641290, -0.00188987, 0.000477033, -0.00125997189 meaning our equation is as follows:

$y = 0.9946 - 0.00189x_1 + 0.000477x_2 - 0.00126x_3$

where x_1 is the residual sugar, x_2 is total sulfur dioxide and x_3 is alcohol.





The plots above show the predicted (as the red line) and actual values for Wine Density based on the attributes. The plot of the data clearly shows a positive linear correlation between density and residual sugar and total sulfur dioxide and a negative linear correlation between

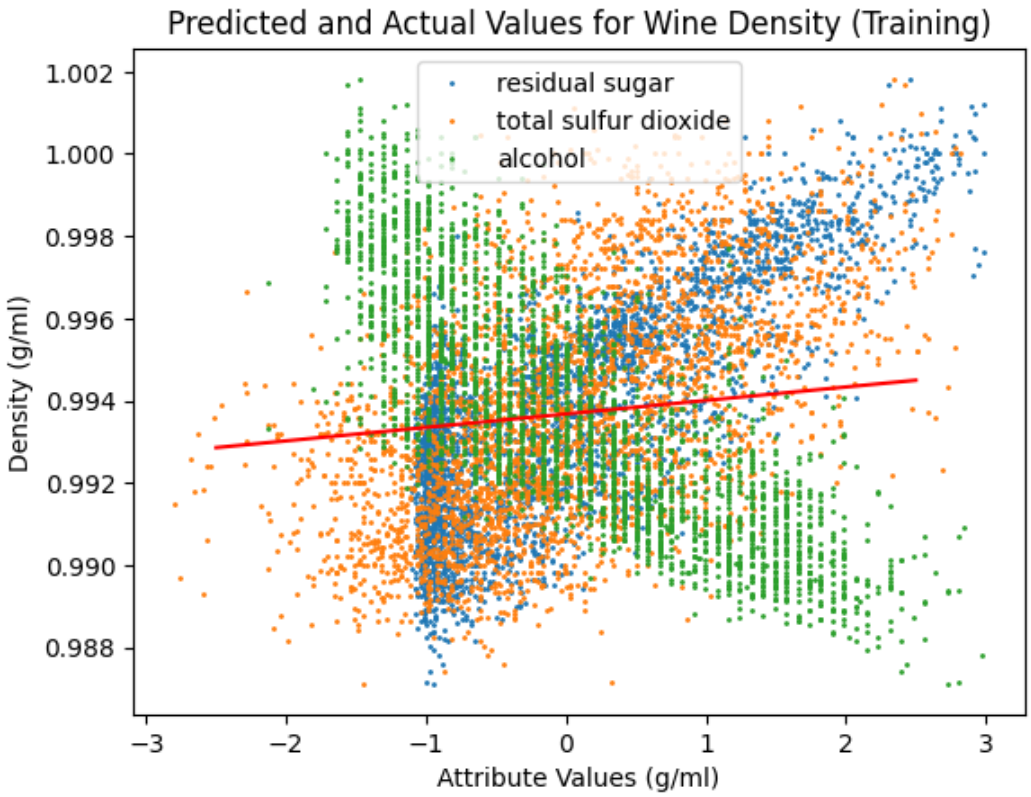
alcohol and density. Our predicted model has a slightly positive linear correlation. This could be because the 2 attributes with the positive correlations are more strongly influencing the model than alcohol which is negatively linearly correlated. From the picture alone, it seems our predicted model (while having a very low MSE) may not be the best predictor. However, compared to the other models, this combination of parameters is the best so we are satisfied with our model and we have found the best solution according to the algorithm we implemented. The R squared values for training is about 0.784 while testing is about 0.802, which indicates the model is a decently good fit for both our training and testing datasets. Our model performs well for the given dataset for a more naive implementation of gradient descent. Therefore, we are satisfied with our model for Part 1.

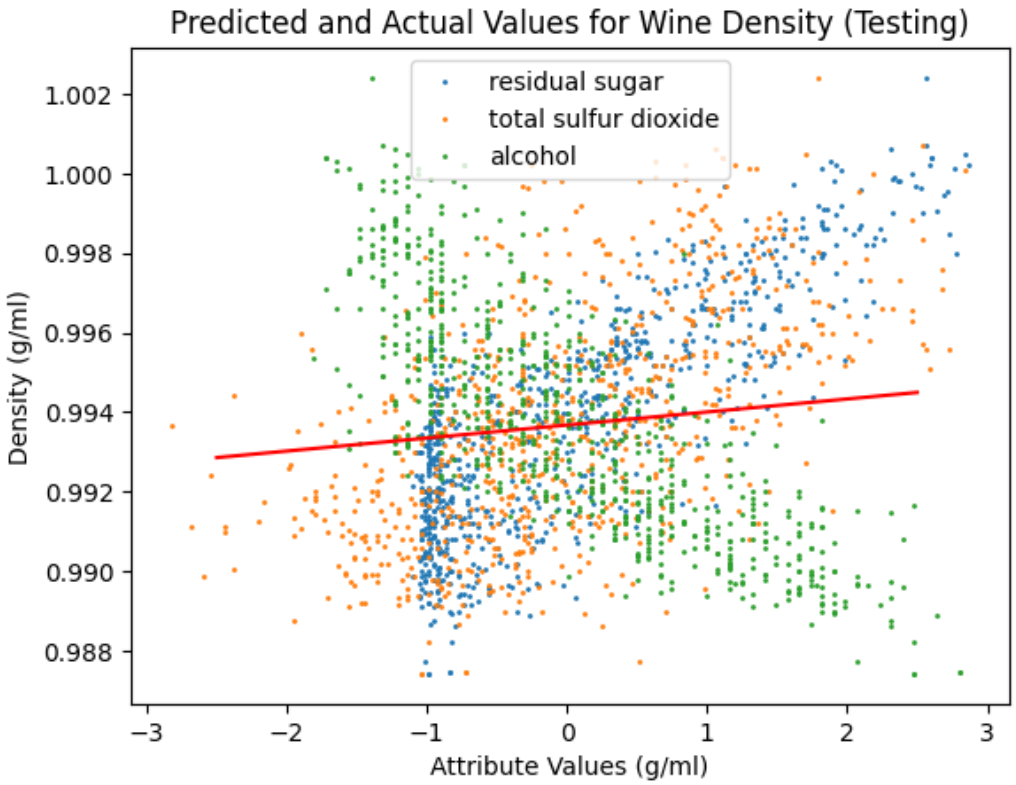
Note, we found similar results in Part 1 without using alcohol as an attribute.

Part 2: Linear Regression using Stochastic Gradient Descent from Sklearn library

Alpha	Number of Iterations	Tolerance	Learning Rate	Eta0	Training MSE	Testing MSE	Training R2	Testing R2	Coefficients
0.001	10	1e-05	constant	0.01	6.76565e-07	6.26436e-07	0.911544	0.921634	[0.00148378 0.00021036 -0.00146828]
0.001	10	1e-05	constant	0.001	6.67922e-07	6.09941e-07	0.912674	0.923698	[0.00153221 0.00024782 -0.00147039]

After performing analysis of a stochastic gradient descent linear regression model from scratch, we then chose the SGDRegressor function from the scikit learn library to create the model with some fine-tuned hyperparameters for us. The main hyperparameters that we focused on were alpha which is the numeric learning rate value, the tolerance for the gradient, the type of learning rate such as adaptive, optimal, etc., a random state, the number of iterations, and the eta0 numeric value for the initial learning rate. The rest of the required parameters for the SGDRegressor function were kept at the default values provided by the library and didn't change between each trial. Most of the trials showed that the testing value for the coefficient of determination was around 92% and the training value for the coefficient of determination was around 91% for the best models. The best model had a learning rate of 0.00001, was trained for 1000 iterations, and had a tolerance of 0.0000001. This best scoring model also had an adaptive learning rate where eta = eta0 and in this case eta0 was assigned 0.0001. The testing r squared value was around 92.2% whereas the training r squared value was around 91.2% and the training and testing mean squared errors were both very small further indicating how good the model is. There was never a case where the training R-squared value was above the testing R-squared value, meaning that our SGDRegressor models were pretty good and didn't overfit. I would say that I'm satisfied that this model provided the best combination of hyperparameters and returned the best scoring metrics when using the package.


Predicted and Actual Values for Wine Density (Training)
residual sugar
total sulfur dioxide
alcohol
Density (g/ml)
1.002
1.000
0.998
0.996
0.994
0.992
0.990
0.988
−3
−2
−1
0
1
2
3
Attribute Values (g/ml)


Predicted and Actual Values for Wine Density (Testing)
residual sugar
total sulfur dioxide
alcohol
Density (g/ml)
1.002
1.000
0.998
0.996
0.994
0.992
0.990
0.988
−3
−2
−1
0
1
2
3
Attribute Values (g/ml)

```
alpha: 1e-05
Number of Iterations: 1000
Tolerance: 1e-07
Learning Rate: adaptive
Eta0: 0.0001
Training MSE: 0.000000676
Testing MSE: 0.000000621
Training R2: 0.912
Testing R2: 0.922
Coefficients: [ 0.00148561  0.0002946  -0.00145259]
Coefficients: [0.99367539]
```

As mentioned earlier in the report, we chose the residual sugar, total sulfur dioxide, and alcohol to be the model predictor variables for the wine density variable. Using the sklearn library package, the equation for the best model was

$y = 0.9937 + 0.00148x_1 + 0.000295x_2 - 0.00145x_3$

where x_1 is the residual sugar, x_2 is total sulfur dioxide and x_3 is alcohol. This equation is seen in the two plots above. The plots above show the predicted (as the red line) and actual values for Wine Density based on the attributes. The plot of the data clearly shows a positive linear correlation between density and residual sugar and total sulfur dioxide and a negative linear correlation between alcohol and density. Our predicted model has a slightly positive linear correlation as the slope of the line appears to be positive. This could be because the 2 attributes with the positive correlations are more strongly influencing the model compared to alcohol.

Comparison:

Obviously, the library's method outperformed our implementation of gradient descent. This is due to a number of things. First, the library contains a more sophisticated implementation with better techniques to prevent exploding gradients, better stopping conditions, and more parameters to help tune the gradient descent algorithm. Secondly, the library is designed to be optimized compared to our implementation which could lead to better results. However, the gradient descent model also performed really well. The model from part 1 had R squared values of 0.784 and 0.80 for the training and testing while model from part 2 had r squared values of 0.912 and 0.922. We are satisfied with the performance of both the models.