CS 4375
ASSIGNMENT Homework 1

Names of students in your group:
Nicholas Zolton

Number of free late days used: 0

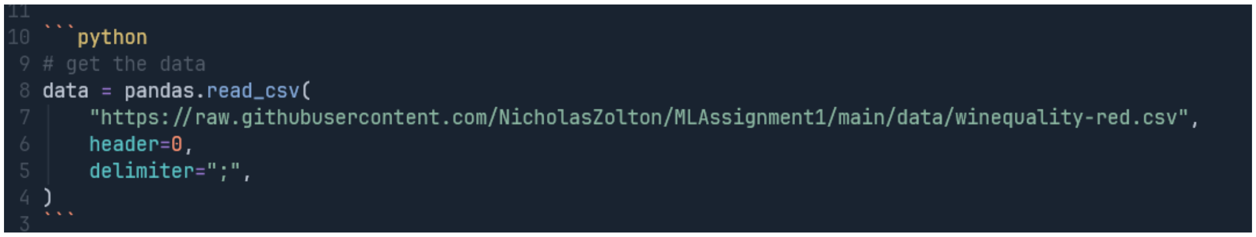
Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.
Seaborn / Pandas / Sci-kit Learn Official Documentation.

Note: All code can be found at https://github.com/NicholasZolton/MLAssignment1

Step 1: Choose a Dataset (and then host it)

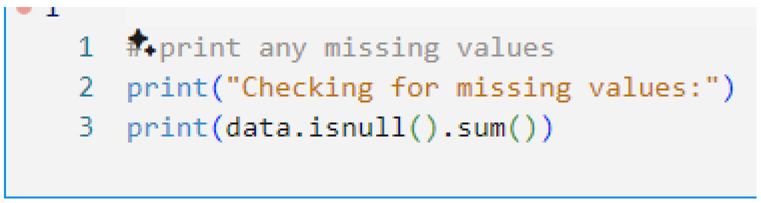
I chose the wine quality dataset:
https://archive.ics.uci.edu/dataset/186/wine+quality



~~~
```python
# get the data
data = pandas.read_csv(
    "https://raw.githubusercontent.com/NicholasZolton/MLAssignment1/main/data/winequality-red.csv",
    header=0,
    delimiter=";",
)
```
~~~

Step 2: Pre-Process the Dataset

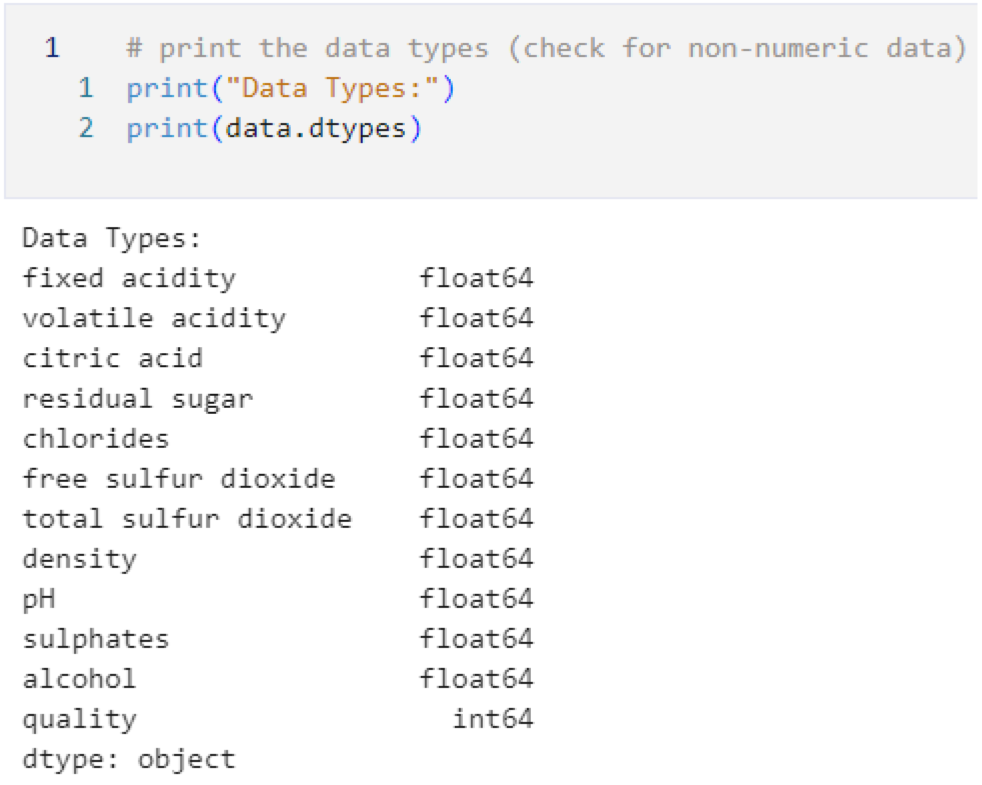
First, I wanted to double check that the data was not missing any values (NaNs, etc.):



```
#print any missing values
print("Checking for missing values:")
print(data.isnull().sum())
```

```
Checking for missing values:
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

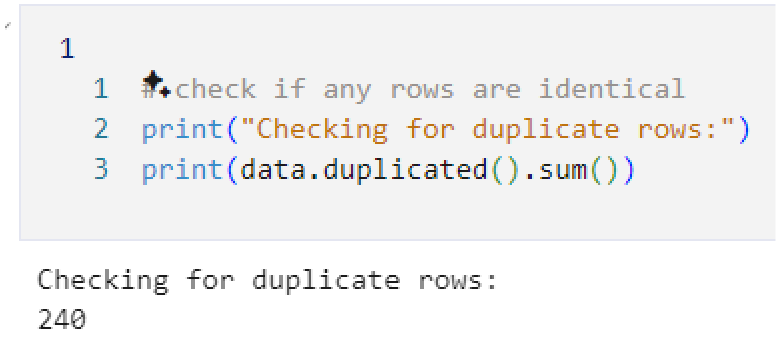
Next I checked that the data types were all numerical:



```
# print the data types (check for non-numeric data)
print("Data Types:")
print(data.dtypes)
```

```
Data Types:
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

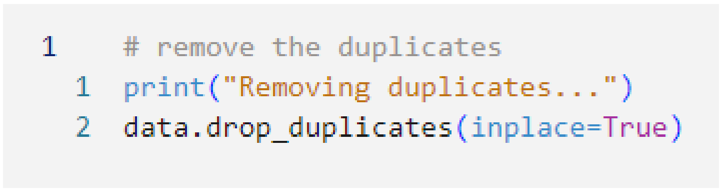
After that I checked for duplicate rows:



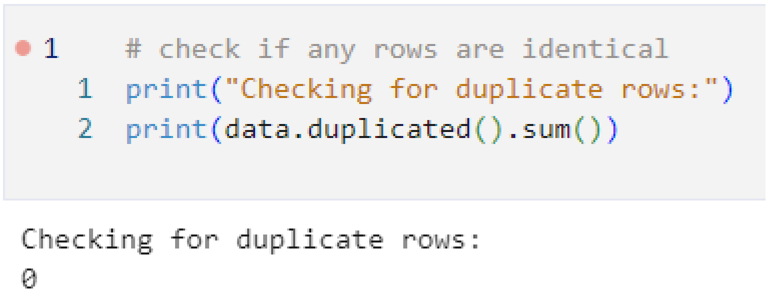
```
#check if any rows are identical
print("Checking for duplicate rows:")
print(data.duplicated().sum())
```

```
Checking for duplicate rows:
240
```

Since there were quite a few of them, I decided to remove them from the dataset so that we only have unique rows:



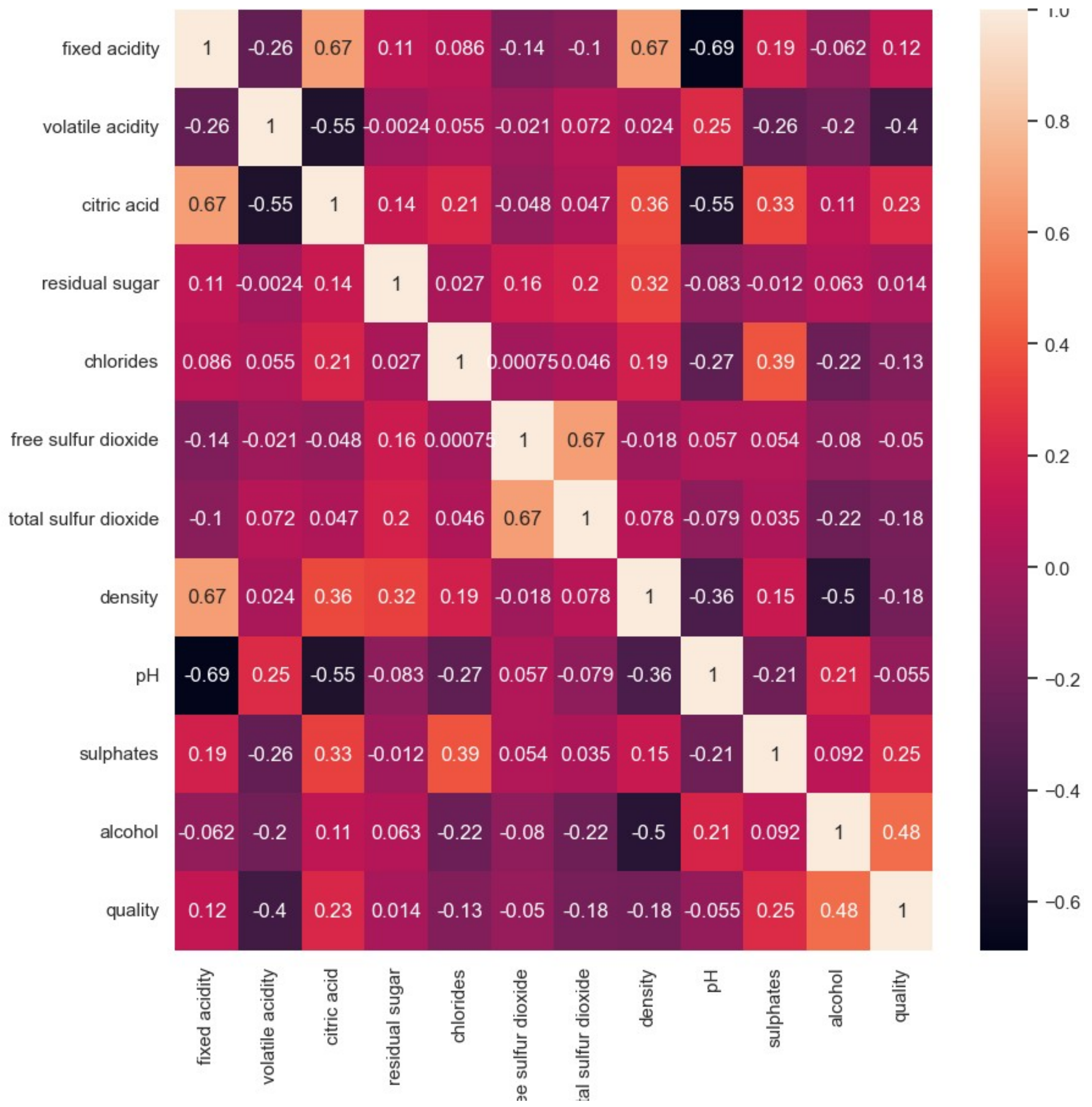
```python
# remove the duplicates
print("Removing duplicates...")
data.drop_duplicates(inplace=True)
```



```python
# check if any rows are identical
print("Checking for duplicate rows:")
print(data.duplicated().sum())
```

```
Checking for duplicate rows:
0
```

Next I looked at the correlation of the features:

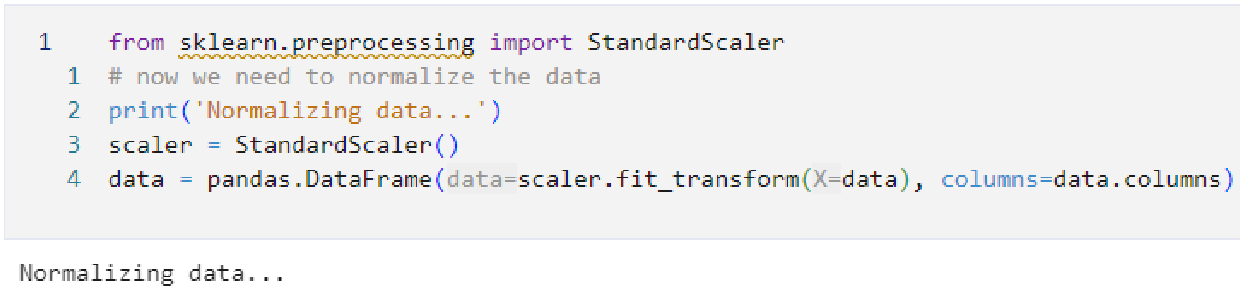


	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1	-0.26	0.67	0.11	0.086	-0.14	-0.1	0.67	-0.69	0.19	-0.062	0.12
volatile acidity	-0.26	1	-0.55	-0.0024	0.055	-0.021	0.072	0.024	0.25	-0.26	-0.2	-0.4
citric acid	0.67	-0.55	1	0.14	0.21	-0.048	0.047	0.36	-0.55	0.33	0.11	0.23
residual sugar	0.11	-0.0024	0.14	1	0.027	0.16	0.2	0.32	-0.083	-0.012	0.063	0.014
chlorides	0.086	0.055	0.21	0.027	1	0.00075	0.046	0.19	-0.27	0.39	-0.22	-0.13
free sulfur dioxide	-0.14	-0.021	-0.048	0.16	0.00075	1	0.67	-0.018	0.057	0.054	-0.08	-0.05
total sulfur dioxide	-0.1	0.072	0.047	0.2	0.046	0.67	1	0.078	-0.079	0.035	-0.22	-0.18
density	0.67	0.024	0.36	0.32	0.19	-0.018	0.078	1	-0.36	0.15	-0.5	-0.18
pH	-0.69	0.25	-0.55	-0.083	-0.27	0.057	-0.079	-0.36	1	-0.21	0.21	-0.055
sulphates	0.19	-0.26	0.33	-0.012	0.39	0.054	0.035	0.15	-0.21	1	0.092	0.25
alcohol	-0.062	-0.2	0.11	0.063	-0.22	-0.08	-0.22	-0.5	0.21	0.092	1	0.48
quality	0.12	-0.4	0.23	0.014	-0.13	-0.05	-0.18	-0.18	-0.055	0.25	0.48	1

Since pH, free sulfur dioxide, and residual sugar had a fairly low correlation (<0.1), I decided to drop them from the data:

```
# remove the fields with little correlation
print("Removing fields with little correlation...")
data: DataFrame = data.drop(columns=["pH", "free sulfur dioxide", "residual sugar"])
```

After that, I chose to normalize the data to prevent certain features from being far larger than others:

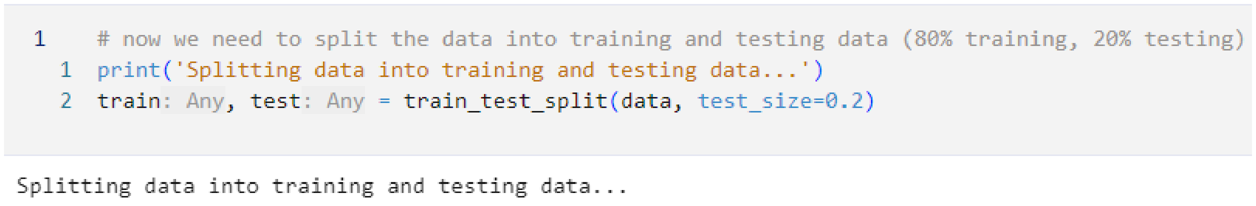


```
from sklearn.preprocessing import StandardScaler
# now we need to normalize the data
print('Normalizing data...')
scaler = StandardScaler()
data = pandas.DataFrame(data=scaler.fit_transform(X=data), columns=data.columns)
```

```
Normalizing data...
```

Step 3: Splitting the Data

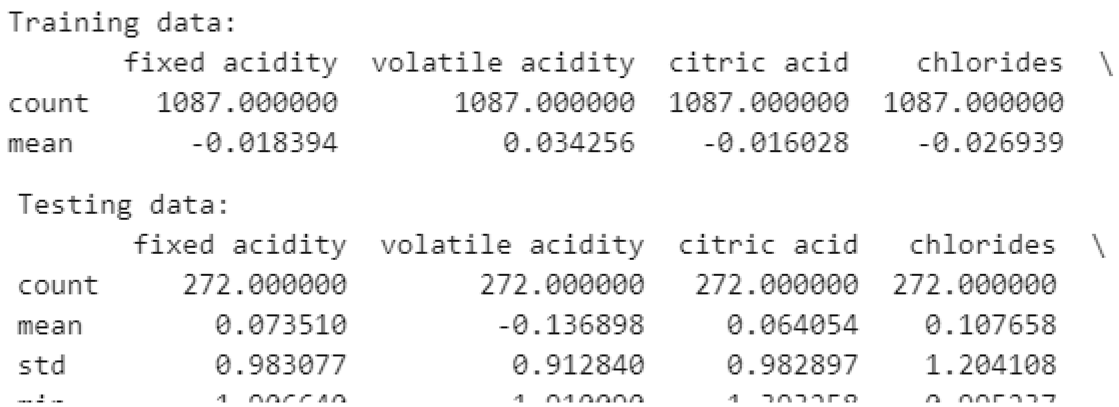
To split the data I decided to use the sklearn train_test_split() function:



```
# now we need to split the data into training and testing data (80% training, 20% testing)
print('Splitting data into training and testing data...')
train: Any, test: Any = train_test_split(data, test_size=0.2)
```

```
Splitting data into training and testing data...
```

This ended up with 1087 training rows, and 272 test rows:



```
Training data:
       fixed acidity  volatile acidity  citric acid    chlorides  \
count    1087.000000       1087.000000  1087.000000  1087.000000
mean       -0.018394          0.034256    -0.016028    -0.026939

Testing data:
       fixed acidity  volatile acidity  citric acid   chlorides  \
count     272.000000        272.000000   272.000000  272.000000
mean        0.073510         -0.136898     0.064054    0.107658
std         0.983077          0.912840     0.982897    1.204108
```

Step 4+5: Creating the Model and Training It

(I combined these steps since I did them both a number of times)

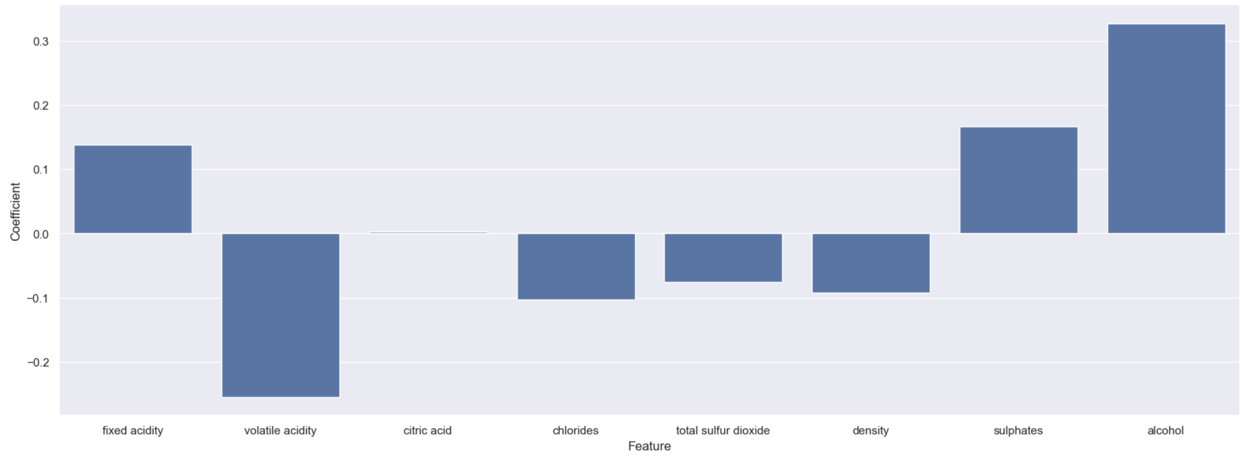
First, I tried making and training a model with the default parameters:

```
# create the model
print('Creating model...')
base_model = SGDRegressor(random_state=0)

# train the model with default parameters
print('Training model...')
base_model.fit(X=train.drop(columns='quality'), y=train['quality'])
```

This ended up with the following scores:

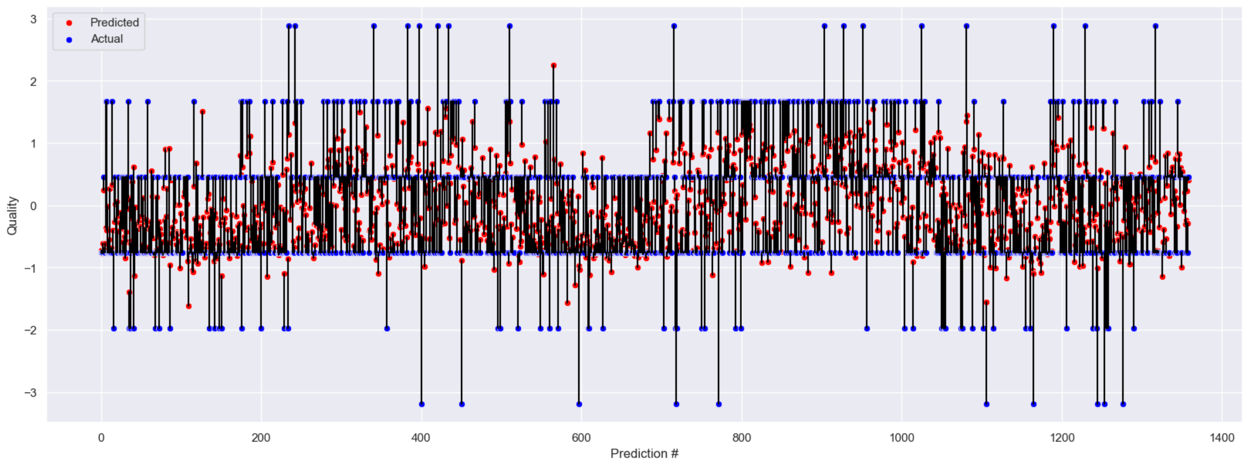
```
Creating model...
Training model...
Model coefficients: [ 0.13780703 -0.25422835  0.0032703  -0.10260775 -0.07545125 -0.09210162
  0.16655288  0.32717622]
Model MSE (training data): 0.6414873858982681
Model MSE (testing data): 0.6427455483439493
Model R^2 (training data): 0.3598796235009144
Model R^2 (testing data): 0.35725445165605074
```

I then created a graph of the model coefficients to visualize which features are considered the most important:

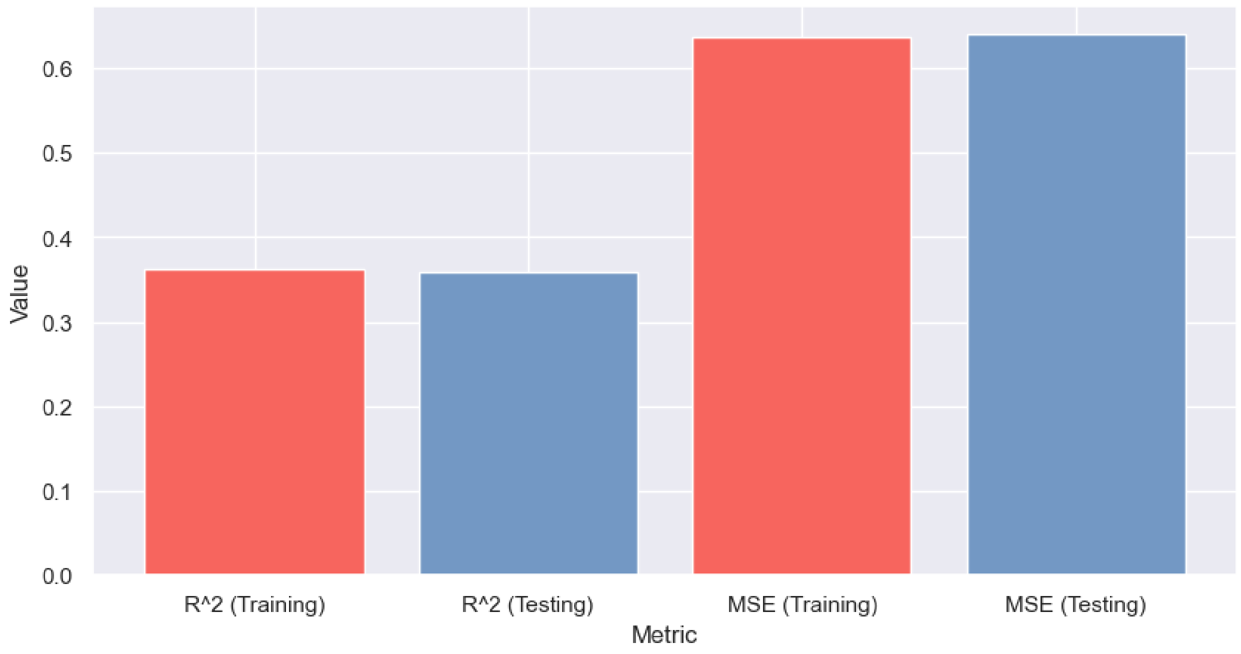


This ended up with alcohol and sulphates having the highest effect, with volatile acidity being negatively correlated and citric acid doing almost nothing.

I also created a graph to visualize how the predictions are working in comparison to the test data (the red dots are predictions, blue are actuals, and the black lines are the error):



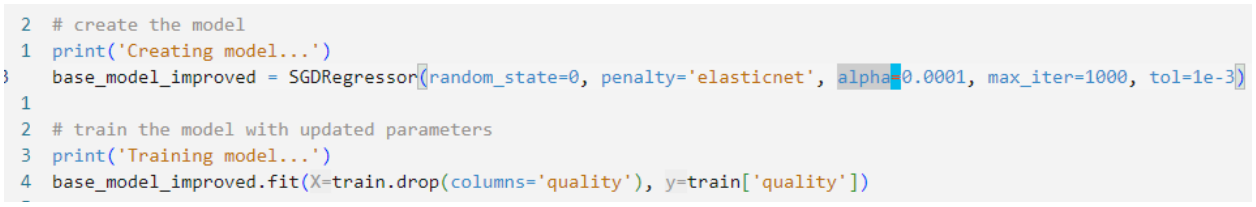
Finally, I made a graph of the difference in R^2/MSE between the training and testing (to see if overfitting is occurring):



Since my results were not incredible (R^2 of ~0.36 is not great), I decided to change the hyperparameters and see if I could achieve a higher result.

Attempt 2:

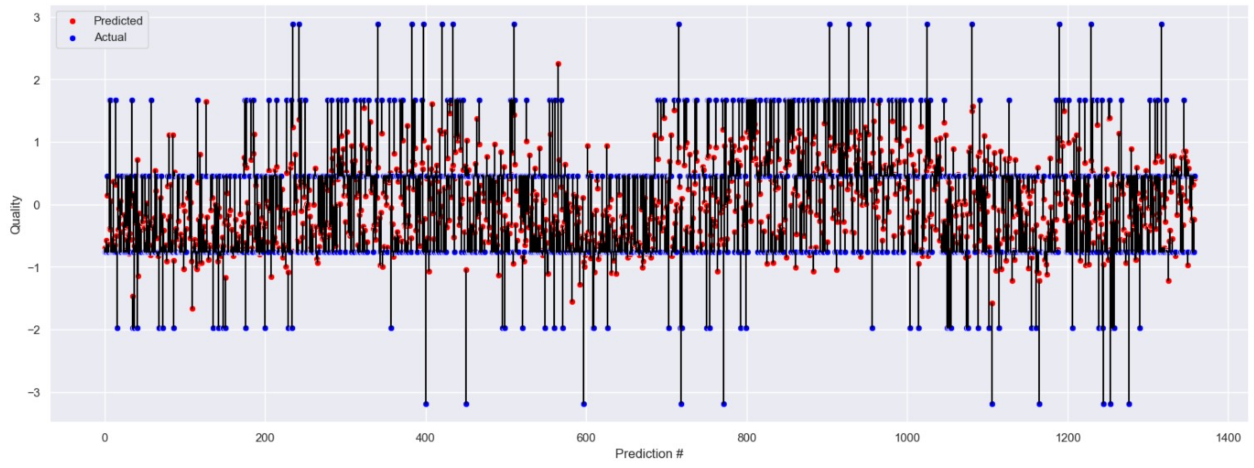
First, I started by creating the model with higher max_iterations, lower tolerance, and a lower alpha:

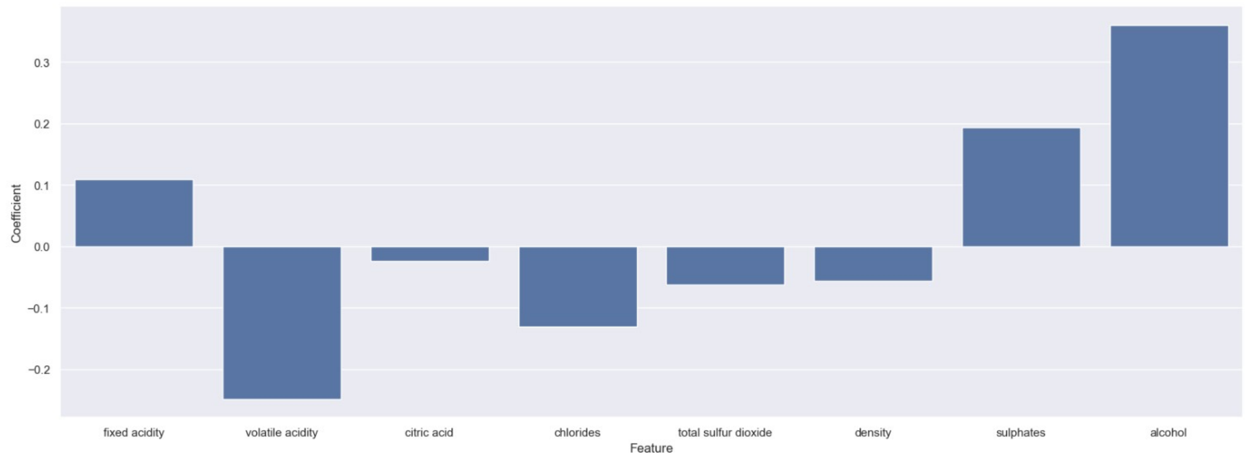


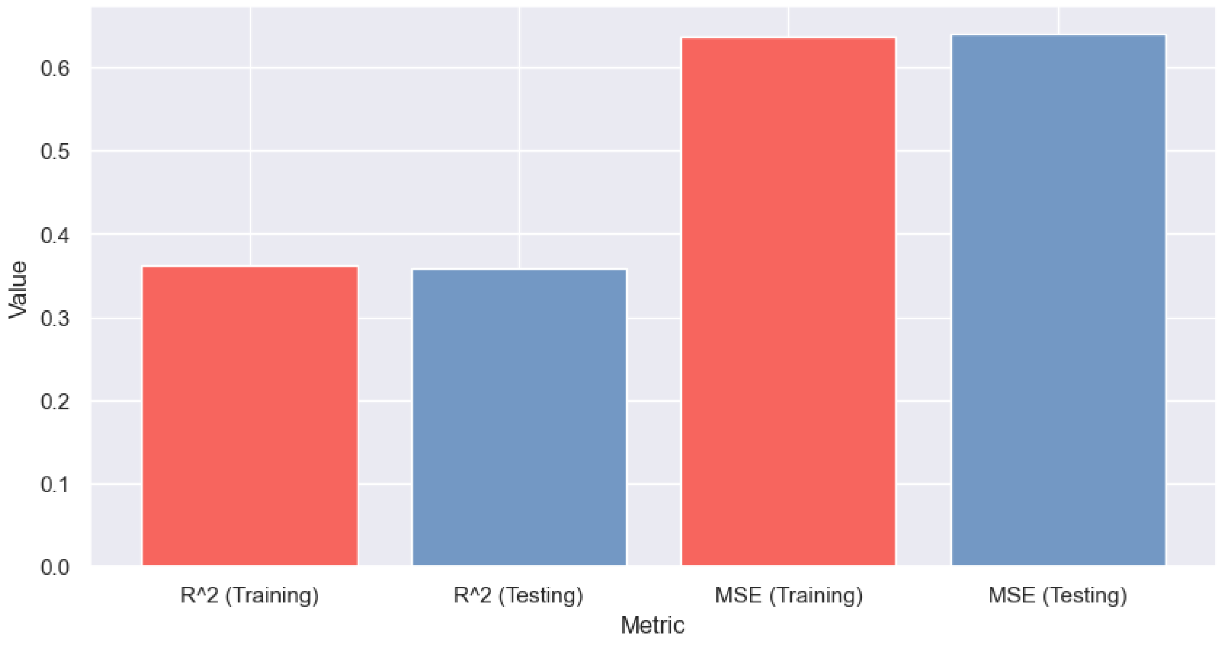
```
# create the model
print('Creating model...')
base_model_improved = SGDRegressor(random_state=0, penalty='elasticnet', alpha=0.0001, max_iter=1000, tol=1e-3)

# train the model with updated parameters
print('Training model...')
base_model_improved.fit(X=train.drop(columns='quality'), y=train['quality'])
```

This ended up with the following scores and graphs:

```
Creating model...
Training model...
Model coefficients: [ 0.10919198 -0.24846191 -0.02449635 -0.13107773 -0.06204708 -0.05570814
  0.19295417  0.36031124]
Model MSE (training data): 0.636305556885284
Model MSE (testing data): 0.6408143771860415
Model R^2 (training data): 0.36138369527650605
Model R^2 (testing data): 0.3591856228139585
```



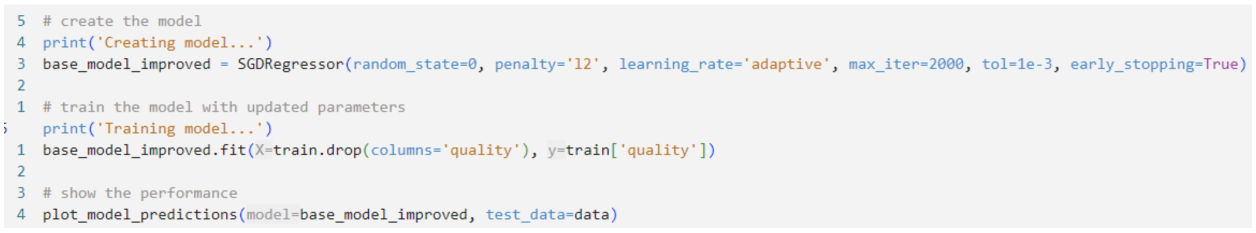




This had a few interesting results. First, citric acid went from being weighted at nearly 0 to being weighted negatively. Second, density lost some of its weight. Third, we can see that the performance on the training data increased while the performance on the testing data decreased. This leads me to believe that we have begun overfitting.

Attempt 3:

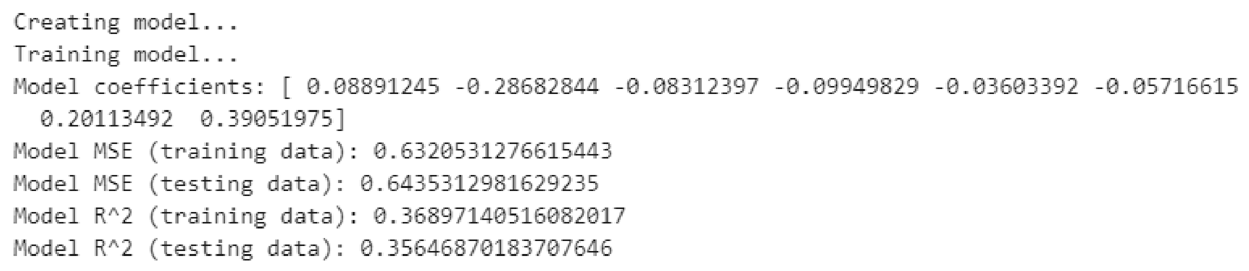
For my final attempt I created the model with the following parameters:



```python
# create the model
print('Creating model...')
base_model_improved = SGDRegressor(random_state=0, penalty='l2', learning_rate='adaptive', max_iter=2000, tol=1e-3, early_stopping=True)

# train the model with updated parameters
print('Training model...')
base_model_improved.fit(X=train.drop(columns='quality'), y=train['quality'])

# show the performance
plot_model_predictions(model=base_model_improved, test_data=data)
```

This uses a different penalty, learning rate, and number of iterations. I also enabled early stopping to see if this would increase the overfitting (purely out of curiosity).

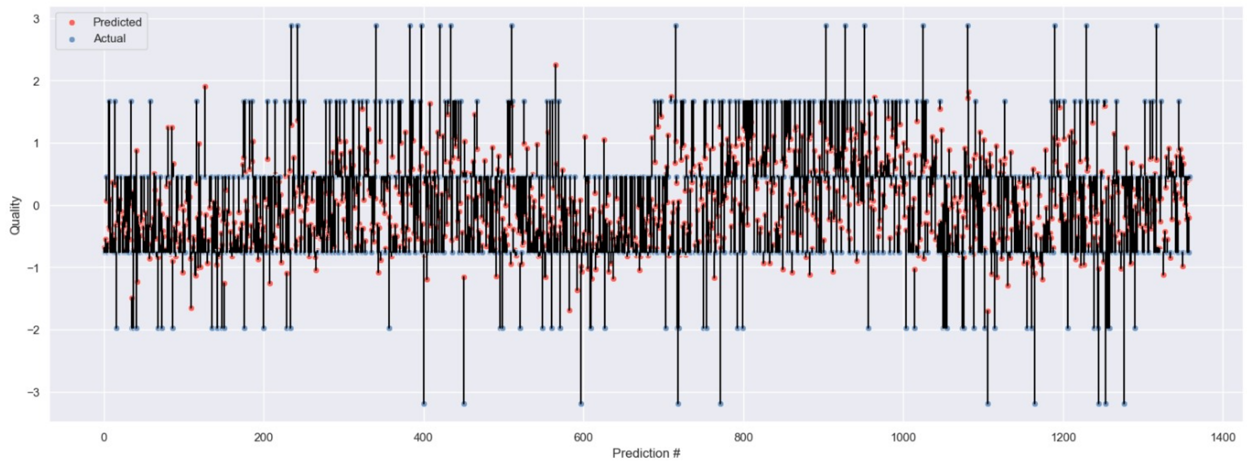
This resulted in the following scores:

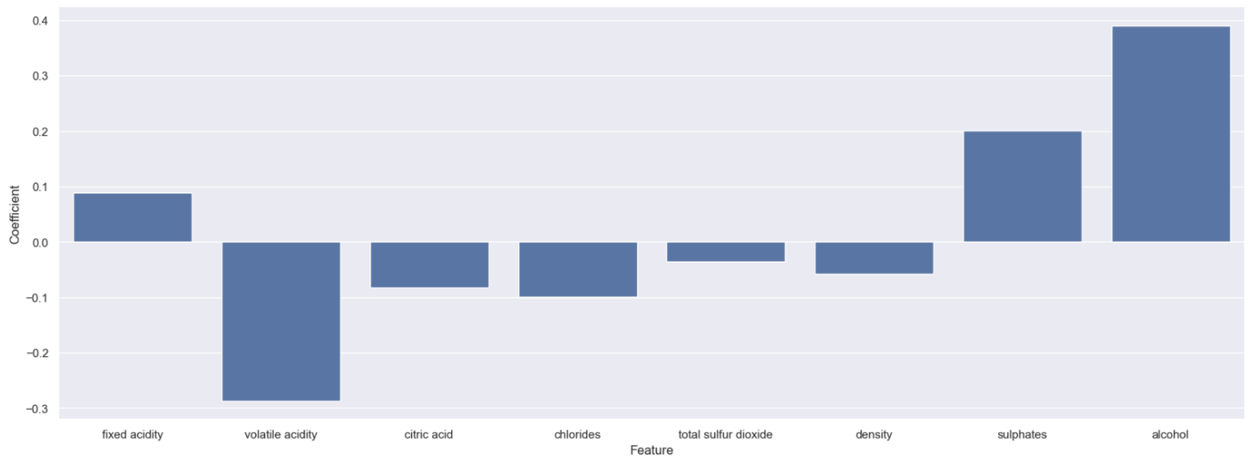


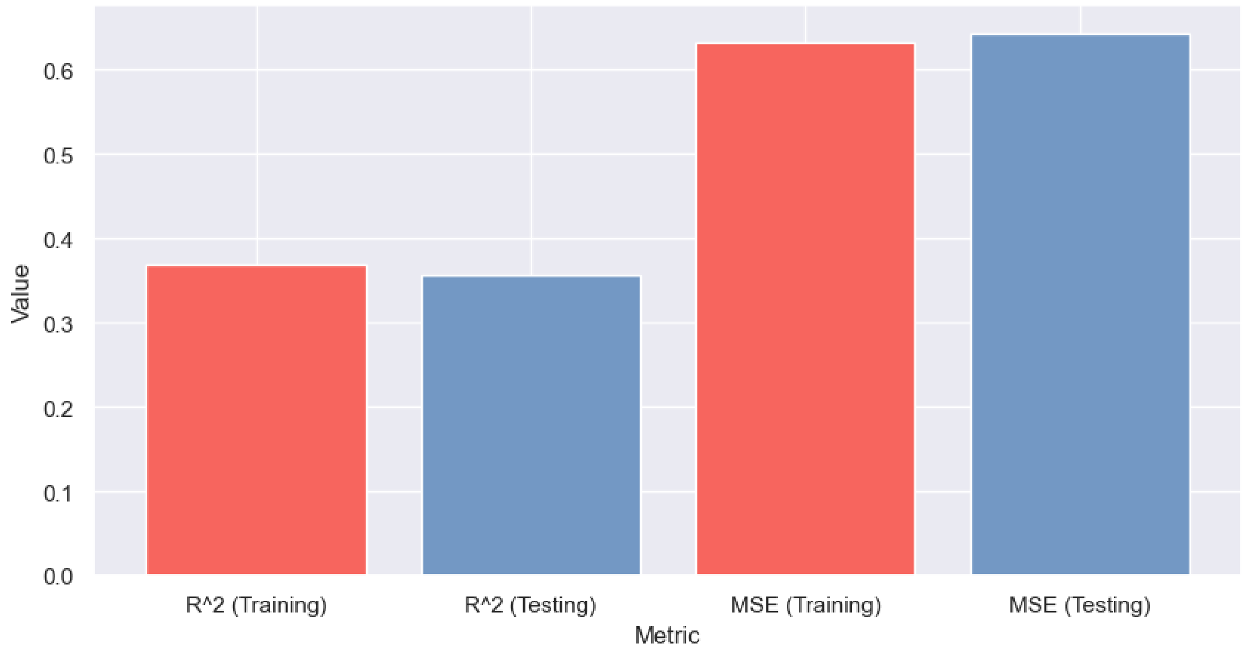
```
Creating model...
Training model...
Model coefficients: [ 0.08891245 -0.28682844 -0.08312397 -0.09949829 -0.03603392 -0.05716615
  0.20113492  0.39051975]
Model MSE (training data): 0.6320531276615443
Model MSE (testing data): 0.6435312981629235
Model R^2 (training data): 0.36897140516082017
Model R^2 (testing data): 0.35646870183707646
```

This, once again, resulted in a higher performance on the training data and a lower performance on the testing data, which is likely an example of overfitting.

This also generated the following graphs:







Interestingly enough, this time the model seemed to "double down" and focus a lot more on the sulphates and alcohol being positive weights, and also focus a lot more on the negative correlation for quality and citric acid, chlorides, and many of the other features.

Overall, since we saw the same trend of the performance improving on the training set and decreasing on the testing set, this seems to be another attempt that has fallen more towards overfitting.

Step 6: Explain: Are you satisfied that you have found the best solution?

It appears that no matter what we do, there is a non-linear relationship that is difficult to capture with a linear model. We seem to be stuck at this score no matter what we change with the parameters. It seems likely to me that no matter what we do we will almost always fall into a similar local optima.

I do not think I have achieved the greatest result possible, but I do think I achieved near the greatest result possible using a simple linear model. Perhaps using a non-linear model it would be possible to improve on this (to me this seems likely). One reason for this might be due to the fact that, just because having more than a small amount of residual sugar is good, having a ton of residual sugar in the wine can likely ruin it as well - thus the linear relationship is not maintained.

This seems especially likely considering that the highest weighted coefficients are alcohol and sulphates, which are likely to have a non-linear relationship with quality.

So, in summary, I am not satisfied that I have found the best solution, but I am satisfied that I have found a near-optimal solution using the linear model.