**Machine Learning Lab** **Kalaiselvan Panneerselvam**

**Exercise sheet-3** **278920**

## 1.1 Data preprocessing (5 Points)

### 1. Convert any non-numeric values to numeric values

```
def handle_non_numerical_data(df):
    columns = df.columns.values
    for column in columns:
        text_digit_vals = {}
        def convert_to_int(val):
            return text_digit_vals[val]

        if df[column].dtype != np.int64 and df[column].dtype != np.float64:
            column_contents = df[column].values.tolist()
            unique_elements = set(column_contents)
            x = 0
            for unique in unique_elements:
                if unique not in text_digit_vals:
                    text_digit_vals[unique] = x
                    x+=1

            df[column] = list(map(convert_to_int, df[column]))

    return df
```

```
air=handle_non_numerical_data(Air_data)
air.head(5)
```

| | City1 | City2 | Average Fare | Distance | Average weekly passengers | market leading airline | market share | Average fare | Low price airline | market share.1 | price |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 33 | 114.47 | 528 | 424.56 | 8 | 70.19 | 111.03 | 10 | 70.19 | 111.03 |
| 1 | 63 | 72 | 122.47 | 860 | 276.84 | 8 | 75.10 | 123.09 | 10 | 17.23 | 118.94 |
| 2 | 26 | 33 | 214.42 | 852 | 215.76 | 8 | 78.89 | 223.98 | 5 | 2.77 | 167.12 |
| 3 | 26 | 27 | 69.40 | 288 | 606.84 | 2 | 96.97 | 68.86 | 3 | 96.97 | 68.86 |
| 4 | 26 | 23 | 158.13 | 723 | 313.04 | 3 | 39.79 | 161.36 | 3 | 15.34 | 145.42 |

### 2. Drop out the rows with missing values or NA

```
winewhite_data.dropna().head(2)
```

| | 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 | 1.001 | 3.0 | 0.45 | 8.8 | 6 |
| 1 | 6.3 | 0.30 | 0.34 | 1.6 | 0.049 | 14.0 | 132.0 | 0.994 | 3.3 | 0.49 | 9.5 | 6 |

```
winered_data.dropna().head(2)
```

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7.4 | 0.70 | 0.0 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 |
| 1 | 7.8 | 0.88 | 0.0 | 2.6 | 0.098 | 25.0 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 | 5 |

3. **Split the data into a train(80%) and test(20%) .**

```
def split_DataSet(Dataset, size):
        msk = np.random.rand(len(Dataset))<size
        Data_train = Dataset[msk]
        Data_test = Dataset[~msk]
        return Data_train,Data_test
```

```
trainSet, testSet = split_DataSet(air, 0.8)
print(trainSet.shape)
print(testSet.shape)
```

```
(800, 7)
(200, 7)
```

## 1.2 Linear Regression with Gradient Descent

### Part A: Implement Linear Regression with Gradient Descent

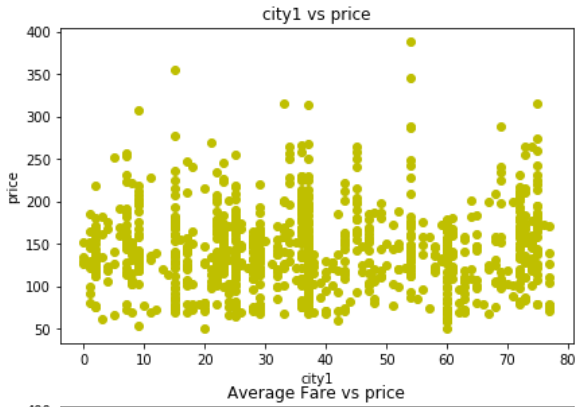
#### 1. Selecting features

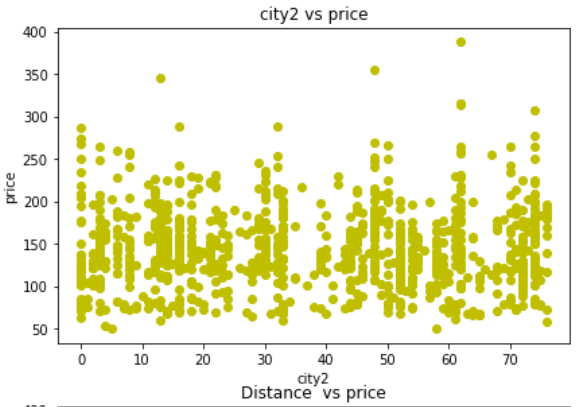
The strength of association is given by

| Strength of Association | Positive | Negative |
|---|---|---|
| Small | .1 to .3 | -0.1 to -0.3 |
| Medium | .3 to .5 | -0.3 to -0.5 |
| Large | .5 to 1.0 | -0.5 to -1.0 |

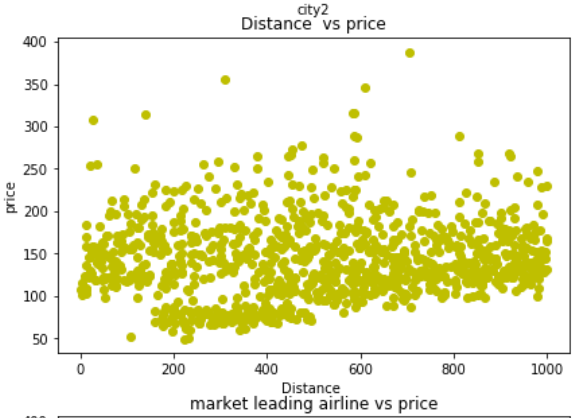
```
The corelation between City1 vs price is 0.04216725451379426
The corelation between City2 vs price is -0.02359031295640774
The corelation between Average Fare vs price is 0.8664103374911549
The corelation between Distance vs price is 0.0936867813163194
The corelation between Average weekly passengers vs price is -0.14231354800543958
The corelation between market leading airline vs price is 0.0923315750323416
The corelation between market share vs price is -0.3076716283706642
The corelation between Average fare vs price is 0.8265105975217341
The corelation between Low price airline vs price is 0.10368176136490836
The corelation between market share.1 vs price is -0.24018595717542673
```

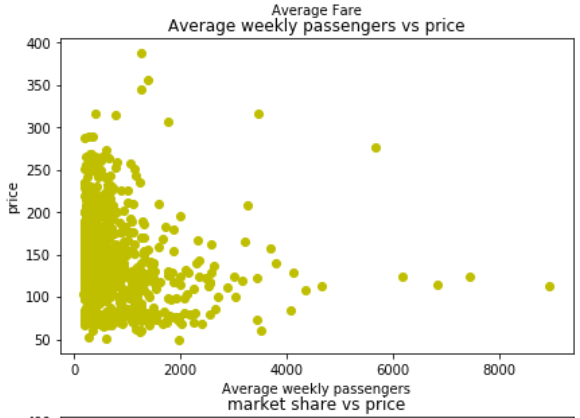
By looking at correlation's coefficient and the plots we can drop city1, city2, market Leading Airline, Low Price Airline as they do not have any correlation with price in the Air fare dataset.


city1 vs price
price
city1


city2 vs price
price
city2

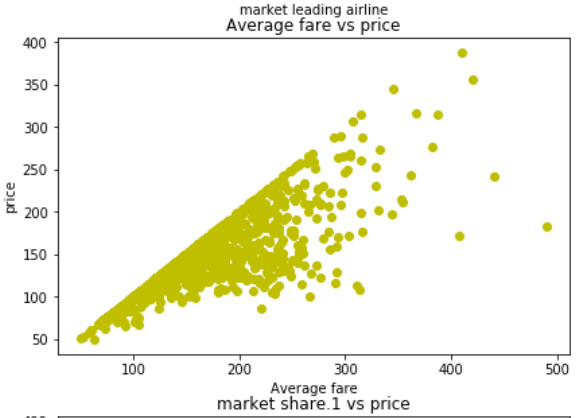

Average Fare vs price
price
Average Fare

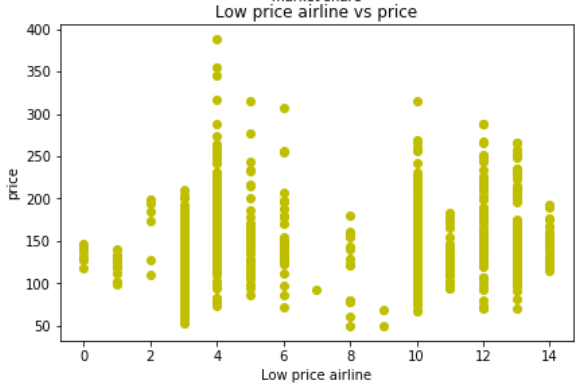

Distance vs price
price
Distance

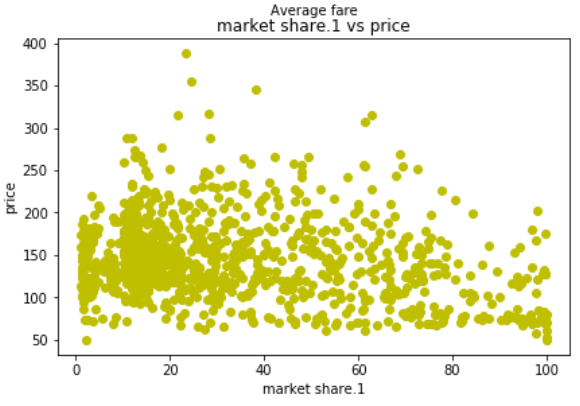

Average weekly passengers vs price
price
Average weekly passengers


market leading airline vs price
price
market leading airline


market share vs price
price
market share


Average fare vs price
price
Average fare


Low price airline vs price
price
Low price airline

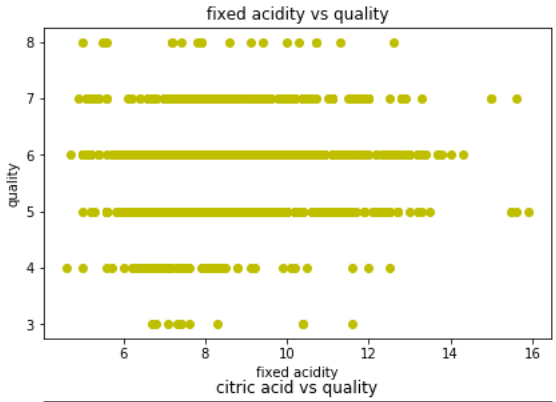

market share.1 vs price
price
market share.1

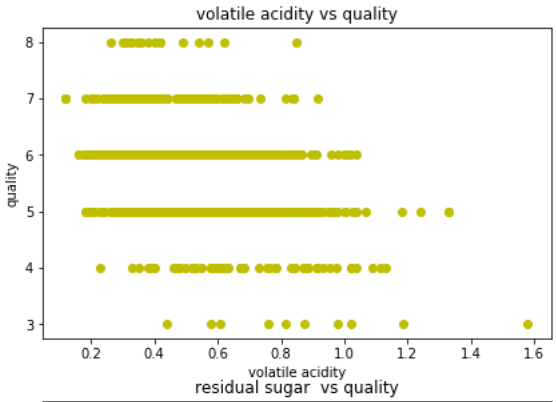
```
For winered dataset,The corelation between fixed acidity vs quality is 0.12405164911322432
For winered dataset,The corelation between volatile acidity vs quality is -0.39055778026400717
For winered dataset,The corelation between citric acid vs quality is 0.2263725143180414
For winered dataset,The corelation between residual sugar vs quality is 0.013731637340066275
For winered dataset,The corelation between chlorides vs quality is -0.1289065599300527
For winered dataset,The corelation between free sulfur dioxide vs quality is -0.05065605724427635
For winered dataset,The corelation between total sulfur dioxide vs quality is -0.18510028892653782
For winered dataset,The corelation between density vs quality is -0.17491922778335012
For winered dataset,The corelation between pH vs quality is -0.05773139120538214
For winered dataset,The corelation between sulphates vs quality is 0.2513970790692614
For winered dataset,The corelation between alcohol vs quality is 0.47616632400113607
```

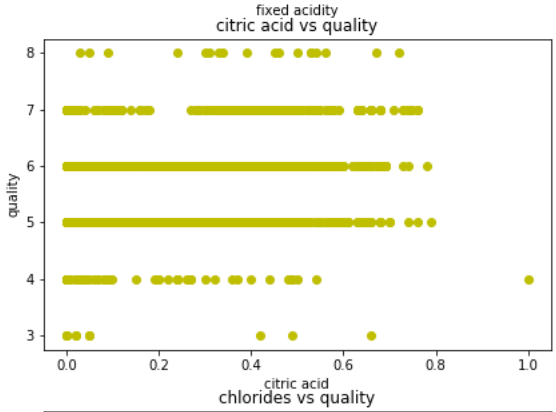
```
For winewhite dataset,The corelation between fixed acidity vs quality is 0.12405164911322432
For winewhite dataset,The corelation between volatile acidity vs quality is -0.39055778026400717
For winewhite dataset,The corelation between citric acid vs quality is 0.2263725143180414
For winewhite dataset,The corelation between residual sugar vs quality is 0.013731637340066275
For winewhite dataset,The corelation between chlorides vs quality is -0.1289065599300527
For winewhite dataset,The corelation between free sulfur dioxide vs quality is -0.05065605724427635
For winewhite dataset,The corelation between total sulfur dioxide vs quality is -0.18510028892653782
For winewhite dataset,The corelation between density vs quality is -0.17491922778335012
For winewhite dataset,The corelation between pH vs quality is -0.05773139120538214
For winewhite dataset,The corelation between sulphates vs quality is 0.2513970790692614
For winewhite dataset,The corelation between alcohol vs quality is 0.47616632400113607
```

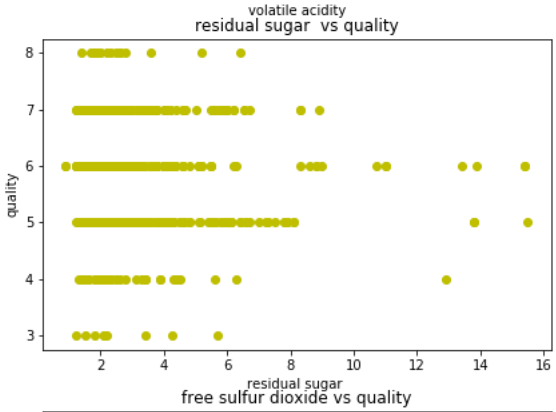
By looking at correlation's coefficient and the plots, I selected volatile acidity, chlorides, density and alcohol in both datasets(Red and white wine datasets)  as they do not have any correlation with quality in the wine dataset.

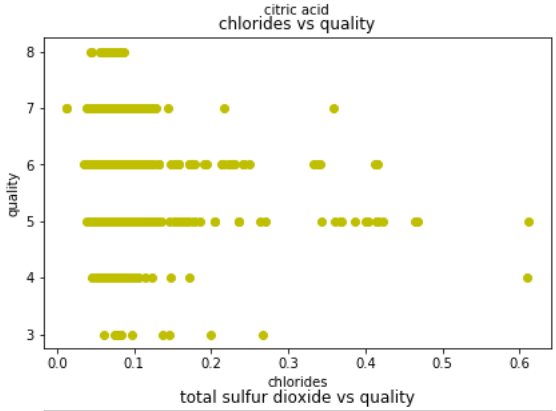
**Wine Red  & Wine White Datasets:**

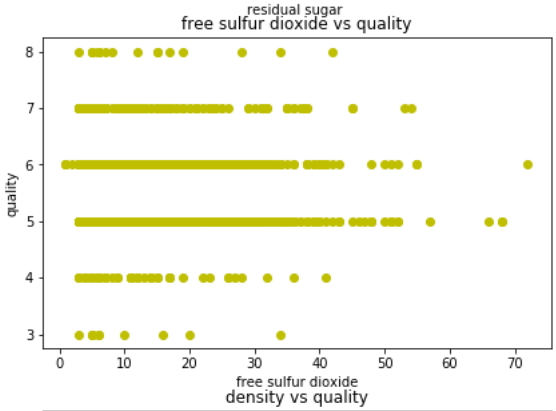

fixed acidity vs quality
quality
fixed acidity

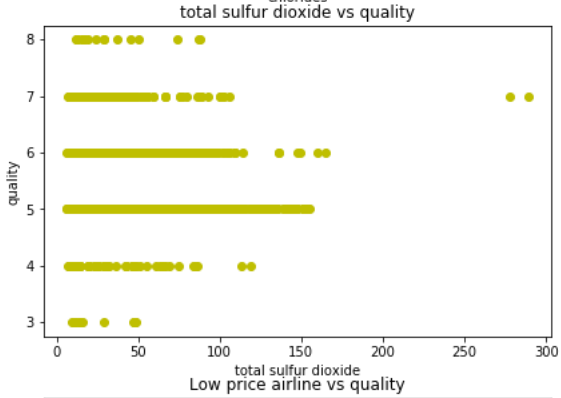

volatile acidity vs quality
quality
volatile acidity

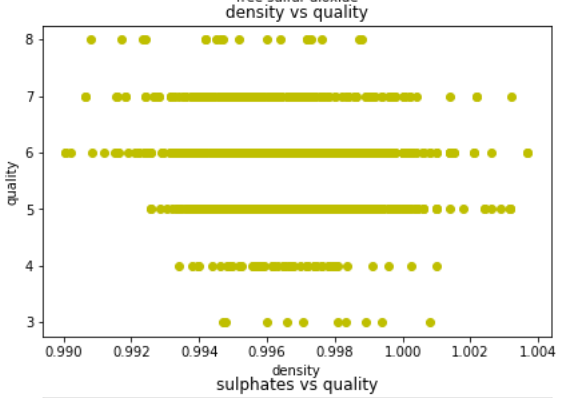

citric acid vs quality
quality
citric acid

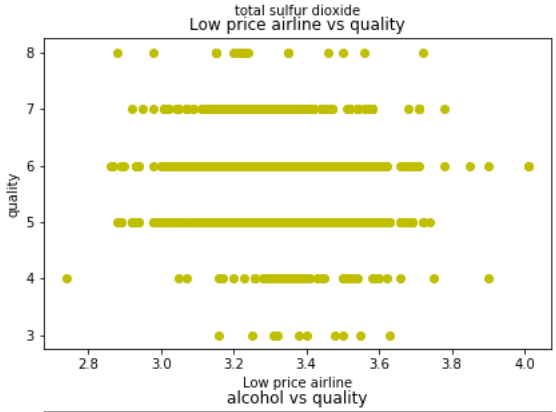

residual sugar vs quality
quality
residual sugar

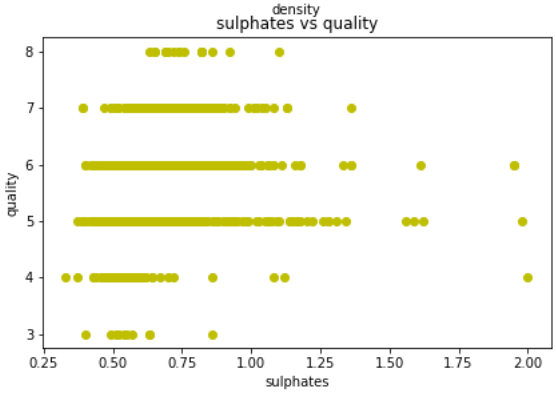

chlorides vs quality
quality
chlorides

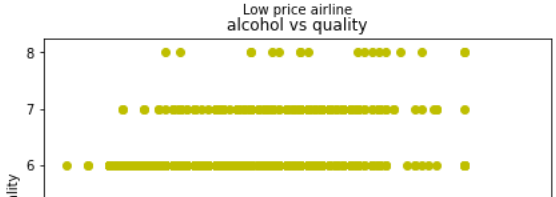

free sulfur dioxide vs quality
quality
free sulfur dioxide

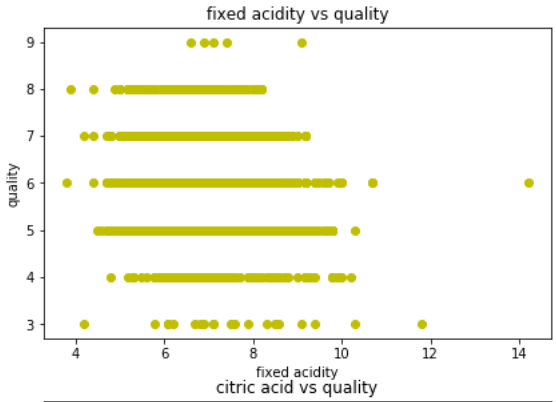

total sulfur dioxide vs quality
quality
total sulfur dioxide

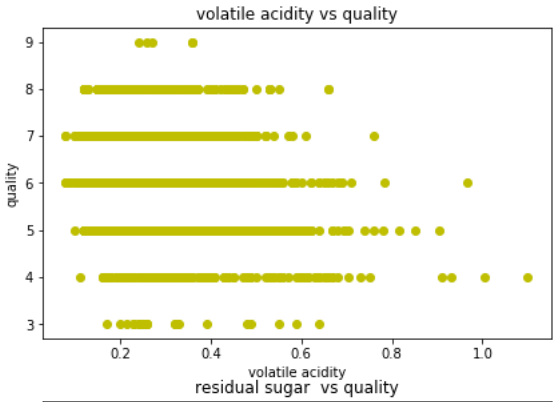

density vs quality
quality
density

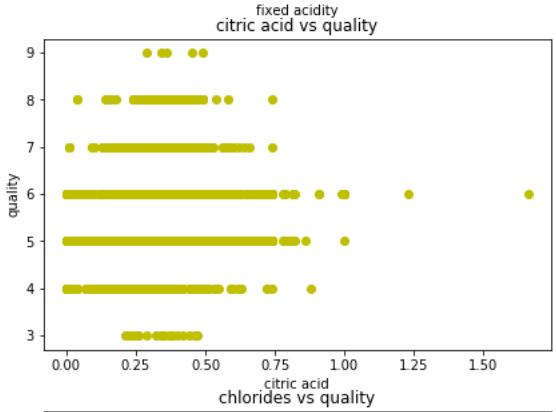

Low price airline vs quality
quality
Low price airline

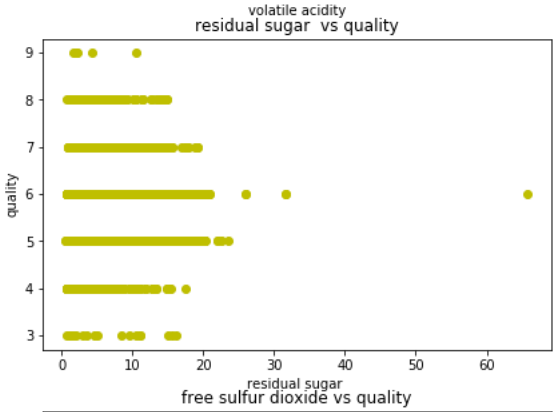

sulphates vs quality
quality
sulphates

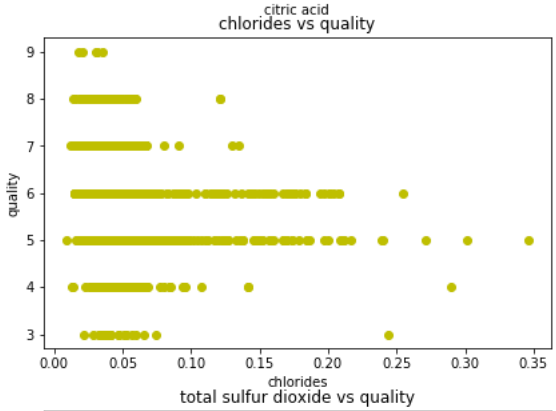

alcohol vs quality

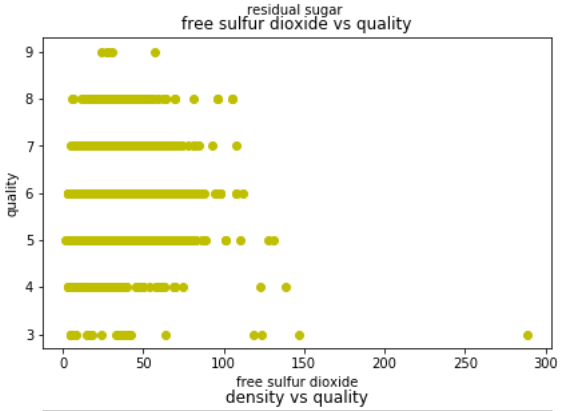

fixed acidity vs quality
quality
fixed acidity

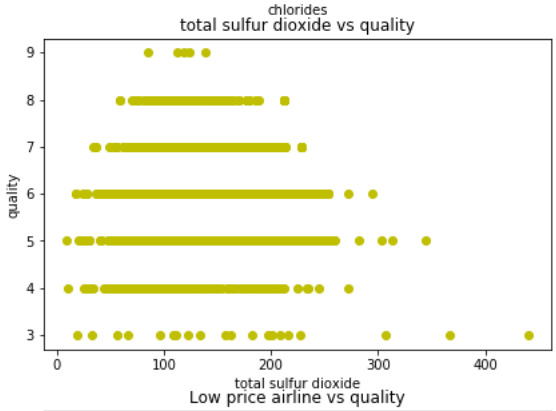

volatile acidity vs quality
quality
volatile acidity

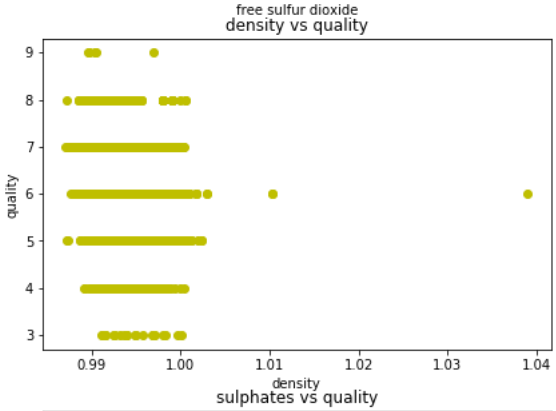

citric acid vs quality
quality
citric acid

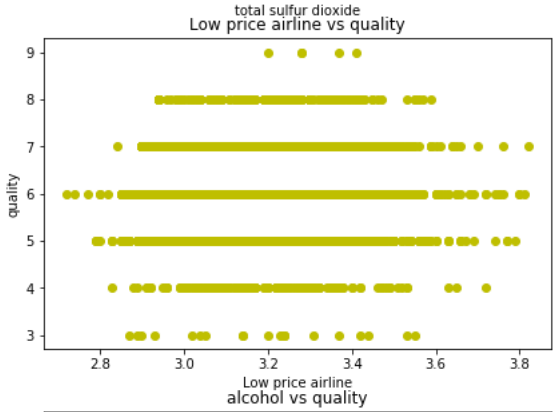

residual sugar vs quality
quality
residual sugar

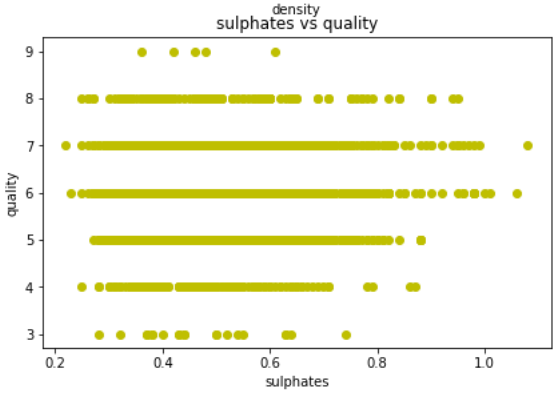

chlorides vs quality
quality
chlorides

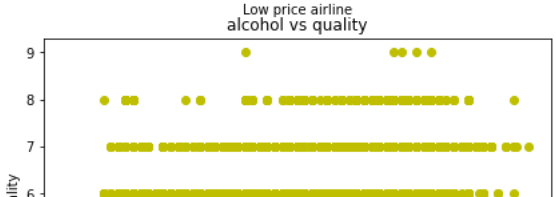

free sulfur dioxide vs quality
quality
free sulfur dioxide


total sulfur dioxide vs quality
quality
total sulfur dioxide


density vs quality
quality
density


Low price airline vs quality
quality
Low price airline


sulphates vs quality
quality
sulphates


alcohol vs quality

Linear Regression model is given as $\hat{y}^n = \sum_{m=1}^{M} \beta_m \boldsymbol{x}_m^n$

```
def Regression_Line(x, betas):
    x = np.insert(x, 0, 1, axis=1)
    yPrediction = np.dot(betas, x.T)
    return yPrediction
```

Least square loss function is given as $l(x, y) = \sum_{n=1}^{N} (y^n - \hat{y}^n)^2$

```
def learnlinregGD(x, y, xTest, yTest, alpha, beta , max_itr=1000, epsilon=1.1e-20,
                  stepLengthController = None, stepLengthControllerParameters = None):
    x = np.insert(x, 0, 1, axis=1)
    x = x * 1.0
    y = y * 1.0

    plotX = []
    plotY_diff = []
    plotY_RMSE = []

    y_pred = np.dot(beta,x.T)
    residual = y_pred - y
    lossfunction = np.dot(residual.T, residual)
    rmse = RMSE(yTest, Regression_Line(xTest, beta))
    plotY_RMSE.append(rmse)
    plotY_diff.append(lossfunction)
    plotX.append(0)

    for i in range(1, max_itr):

        gradient = np.dot(x.T, residual) * 2
        if stepLengthController != None:
            alpha = stepLengthController(lossfunction = lossfunction, alpha = alpha, x = x, y = y,
                                         beta = beta, gradient = gradient, **stepLengthControllerParameters)
        beta = beta - (alpha * gradient)
        y_pred = np.dot(beta,x.T)
        residual = y_pred - y
        lossfunction_updated = np.dot(residual.T, residual)
        rmse = RMSE(yTest, Regression_Line(xTest, beta))
        plotY_RMSE.append(rmse)
        plotY_diff.append(abs(lossfunction - lossfunction_updated))
        plotX.append(i)
        if abs(lossfunction - lossfunction_updated) < epsilon:
            print("Converged in " + str(i) + " iterations")
            return beta, plotX, plotY_diff, plotY_RMSE, lossfunction, rmse
        lossfunction = lossfunction_updated
    print("Algorithm does not converge in " + str(max_itr) + " interations")
    return beta, plotX, plotY_diff, plotY_RMSE, lossfunction, rmse
```

**Minimize the loss function l(x, y) using Gradient Descent algorithm**

**For Air Fare Dataset:**

```
When alpha =0.1
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:
Algorithm does not converge in 1000 interations
RMSE nan
When alpha =1.3e-09
Algorithm does not converge in 1000 interations
RMSE inf
When alpha =1.37e-11
Algorithm does not converge in 1000 interations
RMSE 25.7828447131188
```

**For White wine Dataset:**

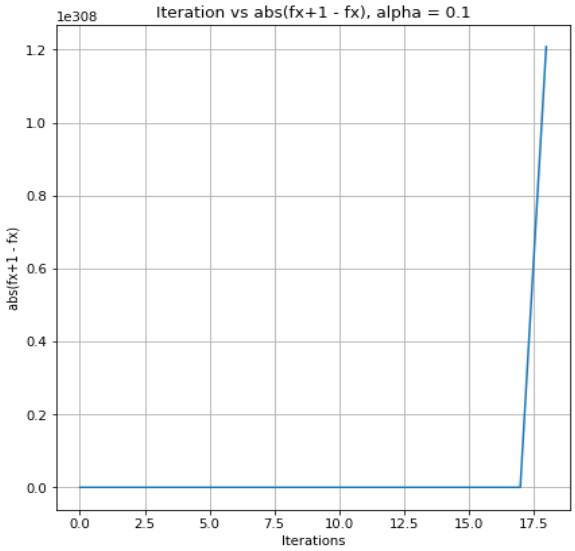
```
When alpha =0.1
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:69: Ru
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:71: Ru
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:63: Ru
Algorithm does not converge in 1000 interations
RMSE nan
When alpha =1.7e-05
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:59: Ru
Algorithm does not converge in 1000 interations
RMSE nan
When alpha =1.7e-11
Algorithm does not converge in 1000 interations
RMSE 1.4667118421940806
```

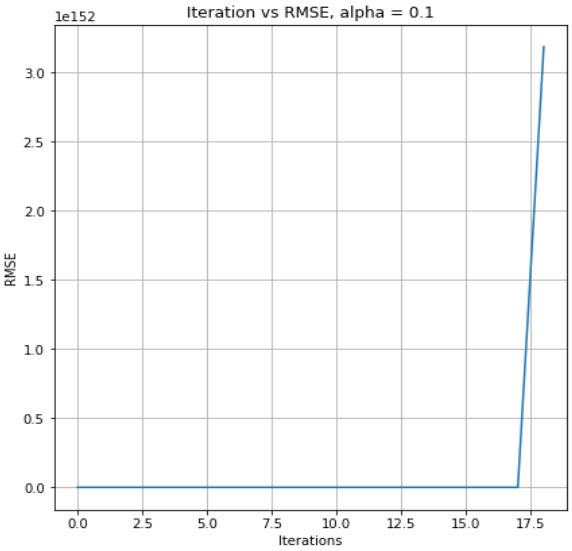
**For White wine Dataset:**

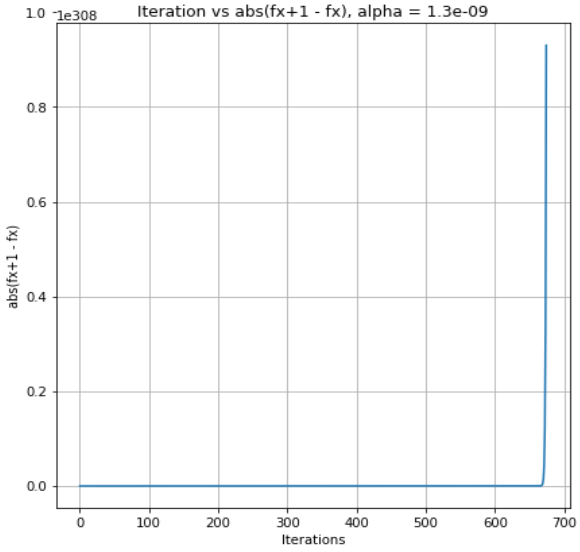
```
When alpha =0.1
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:6
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:7
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:6
Algorithm does not converge in 1000 interations
RMSE nan
When alpha =0.007
C:\Users\HP\Anaconda\lib\site-packages\ipykernel_launcher.py:5
Algorithm does not converge in 1000 interations
RMSE nan
When alpha =1.17e-11
Algorithm does not converge in 1000 interations
RMSE 1.9016040003593744
```

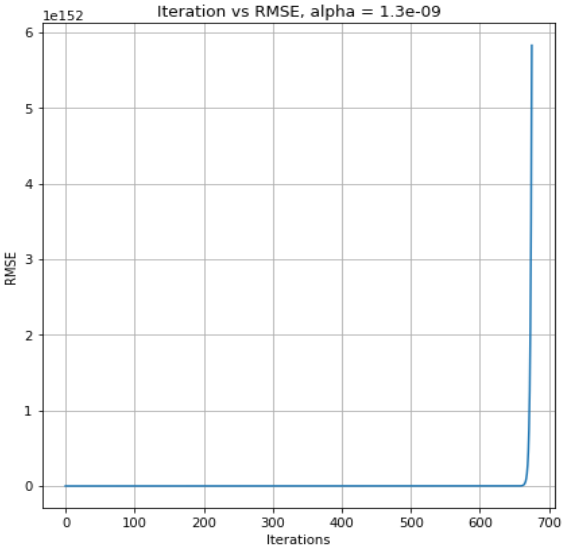
**Plots for Air Fare, White wine and Red wine Dataset:**

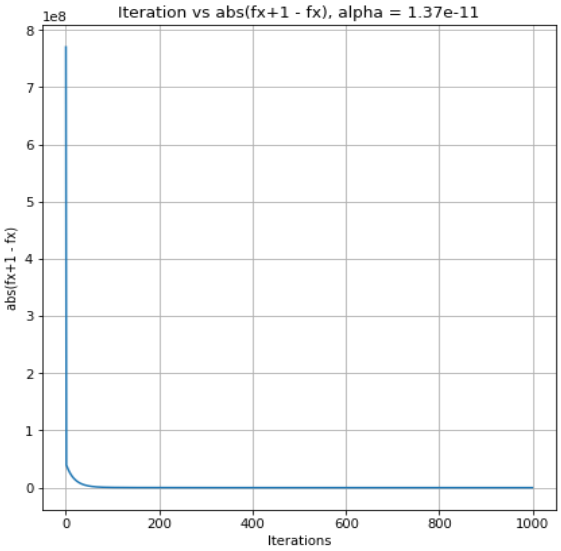
## Air Fare:

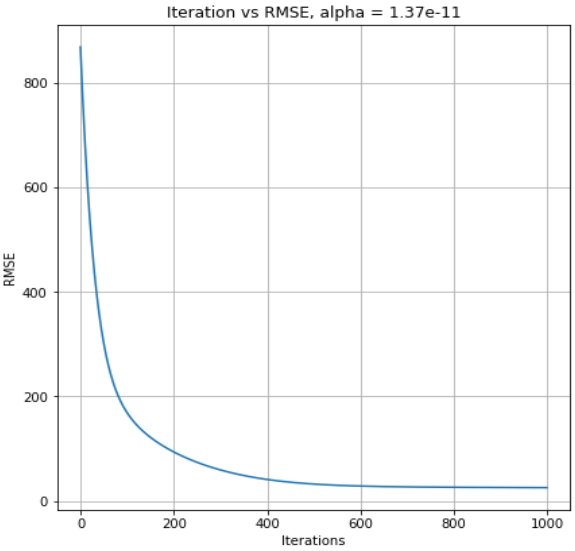

Iteration vs abs(fx+1 - fx), alpha = 0.1
1e308
abs(fx+1 - fx)
Iterations


Iteration vs RMSE, alpha = 0.1
1e152
RMSE
Iterations

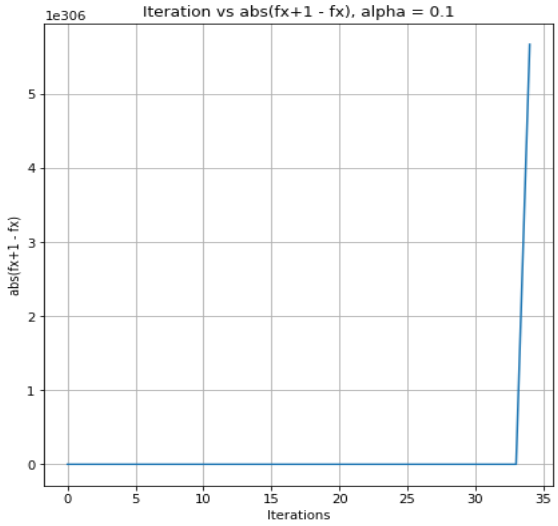

Iteration vs abs(fx+1 - fx), alpha = 1.3e-09
1e308
abs(fx+1 - fx)
Iterations

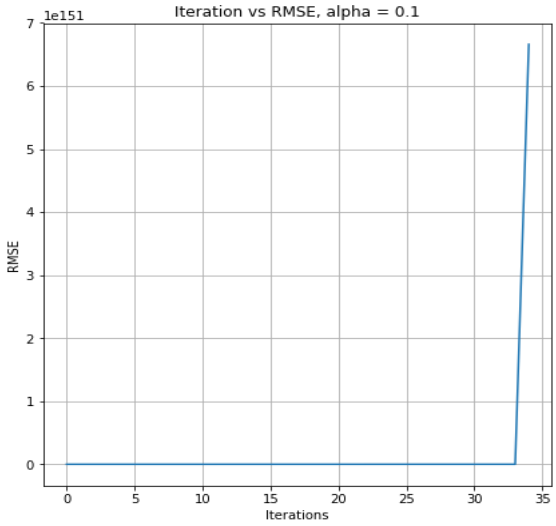

Iteration vs RMSE, alpha = 1.3e-09
1e152
RMSE
Iterations

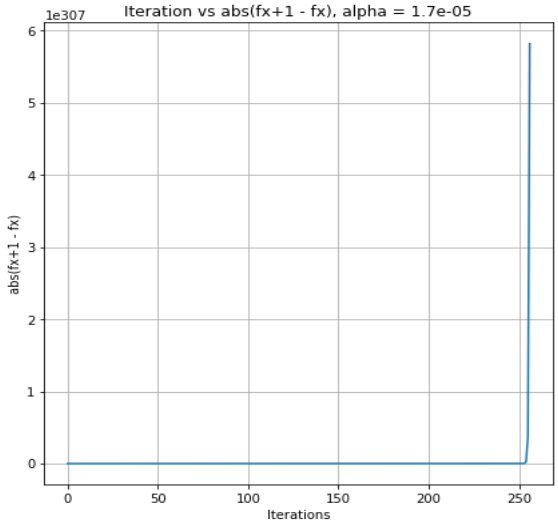

Iteration vs abs(fx+1 - fx), alpha = 1.37e-11
1e8
abs(fx+1 - fx)
Iterations

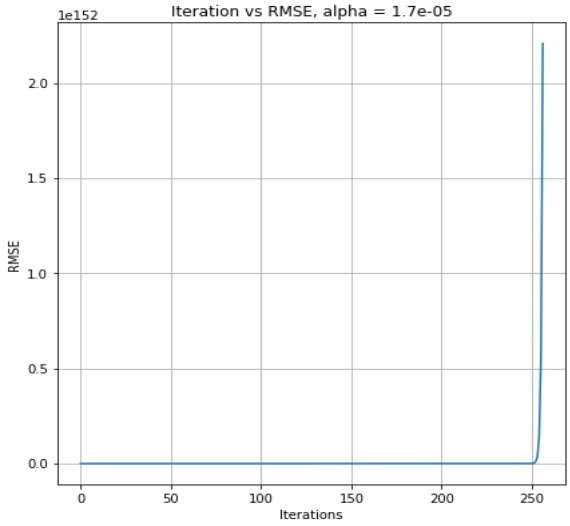

Iteration vs RMSE, alpha = 1.37e-11
RMSE
Iterations

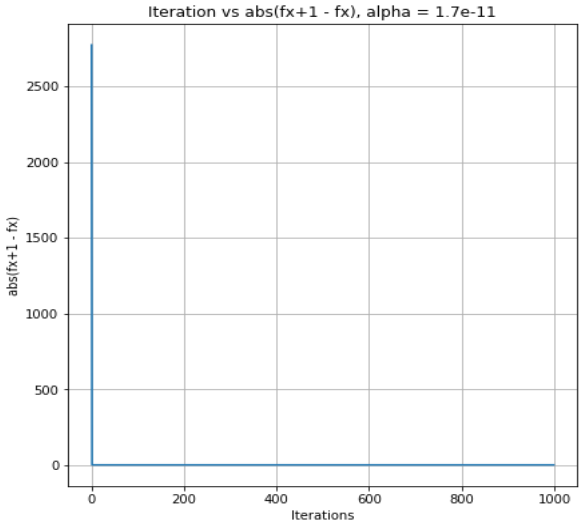
## White wine:

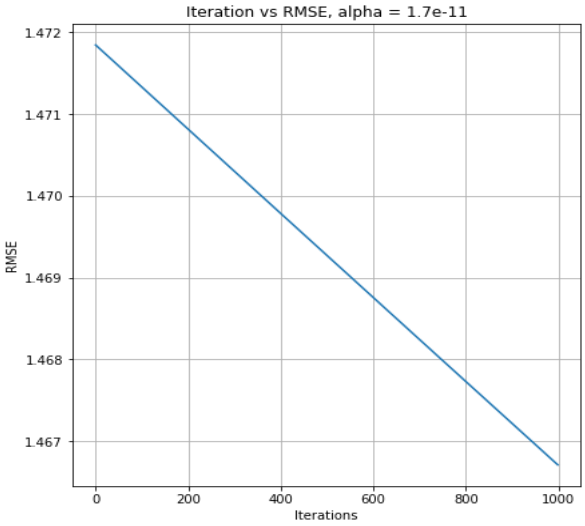

Iteration vs abs(fx+1 - fx), alpha = 0.1
1e306
abs(fx+1 - fx)
Iterations


Iteration vs RMSE, alpha = 0.1
1e151
RMSE
Iterations

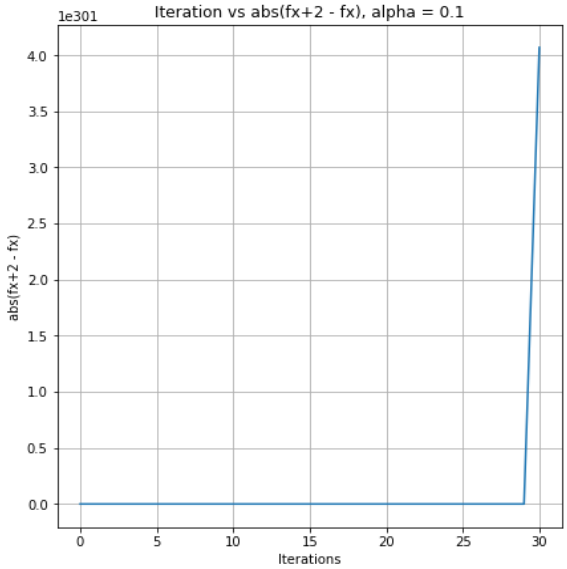

Iteration vs abs(fx+1 - fx), alpha = 1.7e-05
1e307
abs(fx+1 - fx)
Iterations

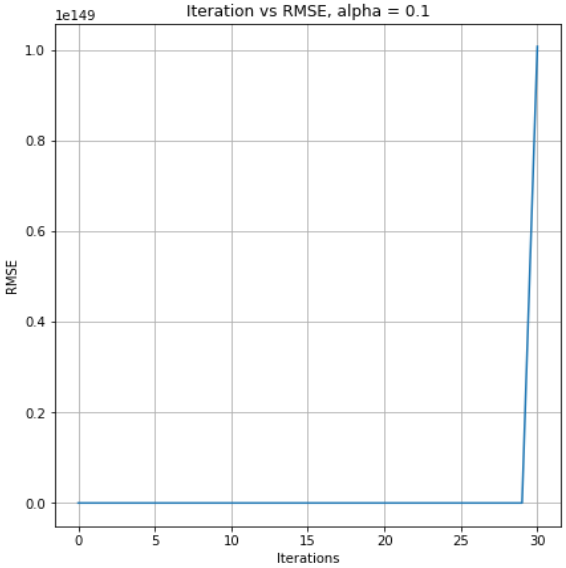

Iteration vs RMSE, alpha = 1.7e-05
1e152
RMSE
Iterations

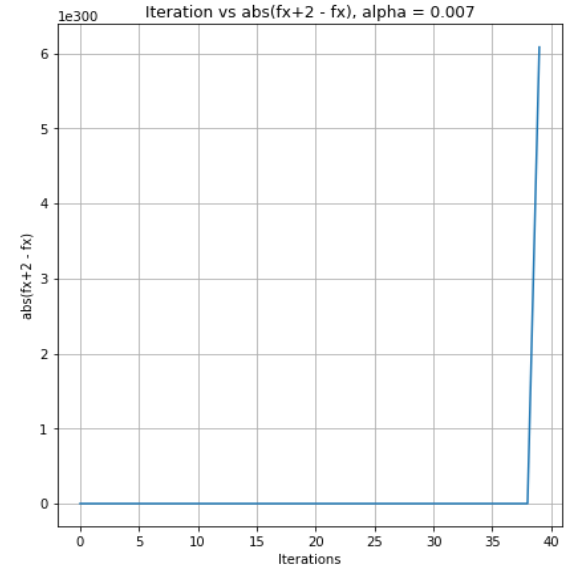

Iteration vs abs(fx+1 - fx), alpha = 1.7e-11
abs(fx+1 - fx)
Iterations

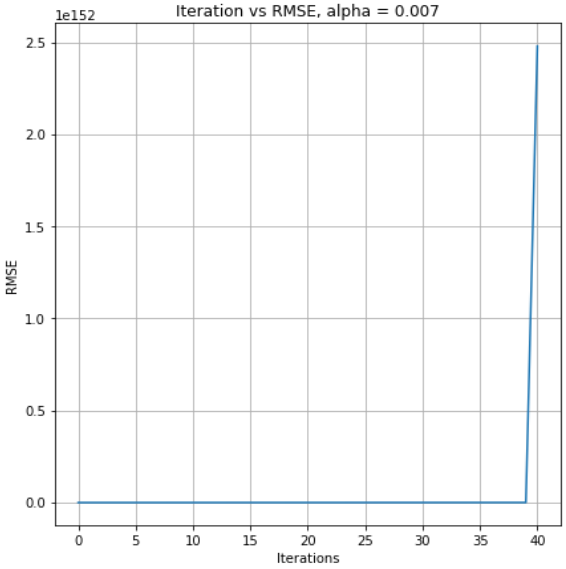

Iteration vs RMSE, alpha = 1.7e-11
RMSE
Iterations

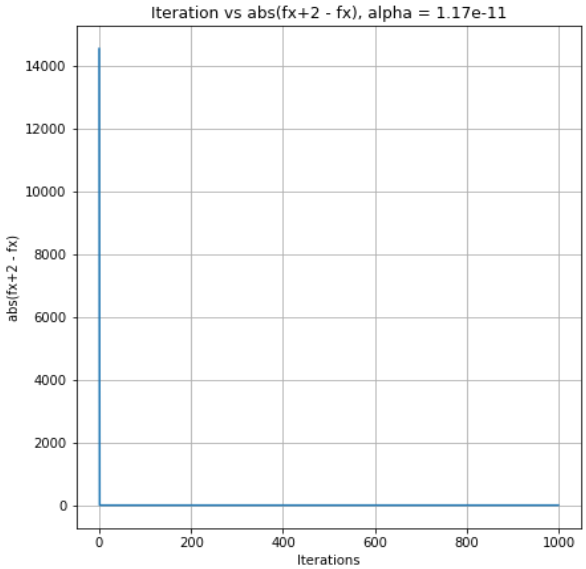
**Red Wine:**

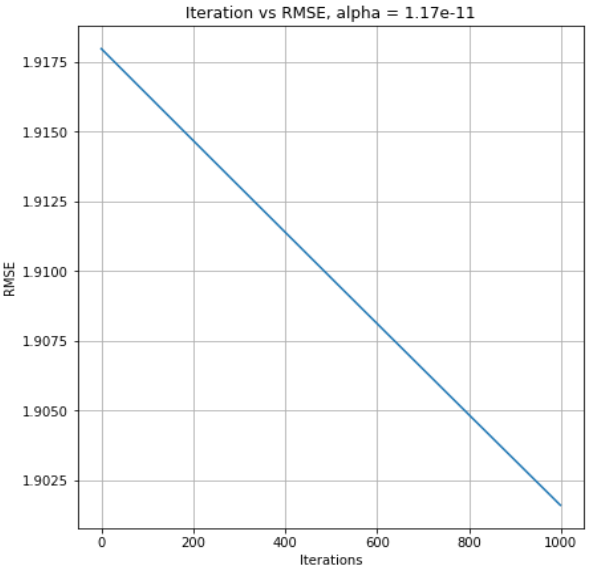

Iteration vs abs(fx+2 - fx), alpha = 0.1
1e301
abs(fx+2 - fx)
Iterations


Iteration vs RMSE, alpha = 0.1
1e149
RMSE
Iterations


Iteration vs abs(fx+2 - fx), alpha = 0.007
1e300
abs(fx+2 - fx)
Iterations


Iteration vs RMSE, alpha = 0.007
1e152
RMSE
Iterations

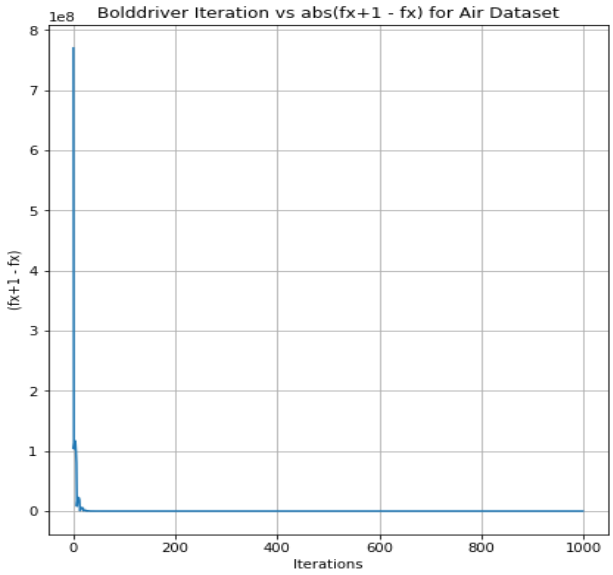

Iteration vs abs(fx+2 - fx), alpha = 1.17e-11
abs(fx+2 - fx)
Iterations

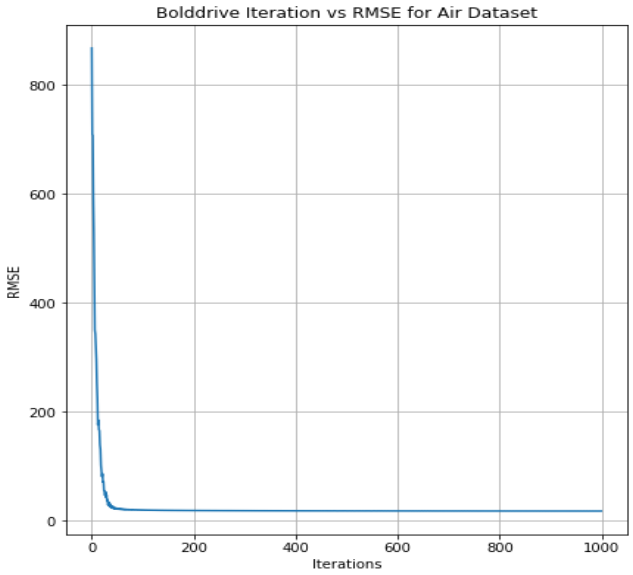

Iteration vs RMSE, alpha = 1.17e-11
RMSE
Iterations

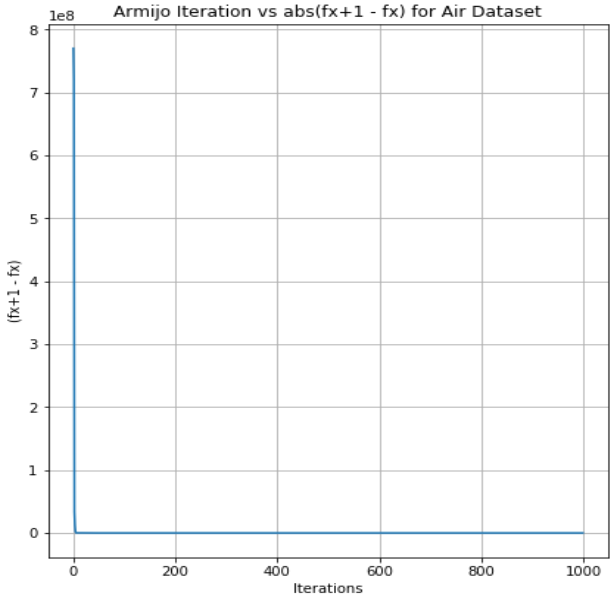
## Part B: (7 Points): Step Length for Gradient Descent

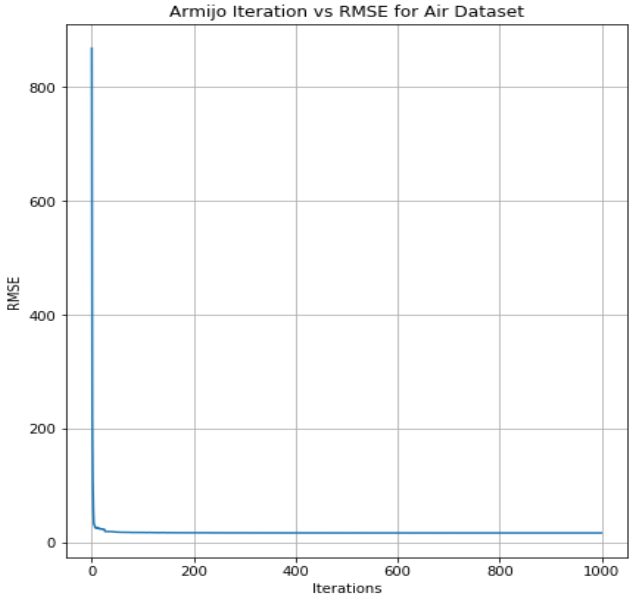
```
def steplength_armijo(lossfunction, alpha, x, y, beta, gradient, delta, max_itr = 1000):
    alpha = 1.0
    gradSquare = np.dot(gradient, gradient)
    for i in range(0, max_itr):
        alpha = alpha/2
        residual = y - np.dot((beta - (alpha * gradient)), x .T)
        lossfunction_gradient = np.dot(residual.T, residual)
        if lossfunction_gradient < lossfunction - (alpha * delta * gradSquare):
            break;
    return alpha

def steplength_bolddriver(lossfunction, alpha, x, y, beta, gradient, max_itr = 1000,alphaMinus = 0.5, alphaPlus = 1.1):
    alpha = alpha * alphaPlus
    for i in range(0, max_itr):
        alpha = alpha * alphaMinus
        residual = y - np.dot((beta - (alpha * gradient)), x .T)
        lossfunction_gradient = np.dot(residual.T, residual)
        if lossfunction - lossfunction_gradient > 0:
            break;
    return alpha
```

## Air Fare:

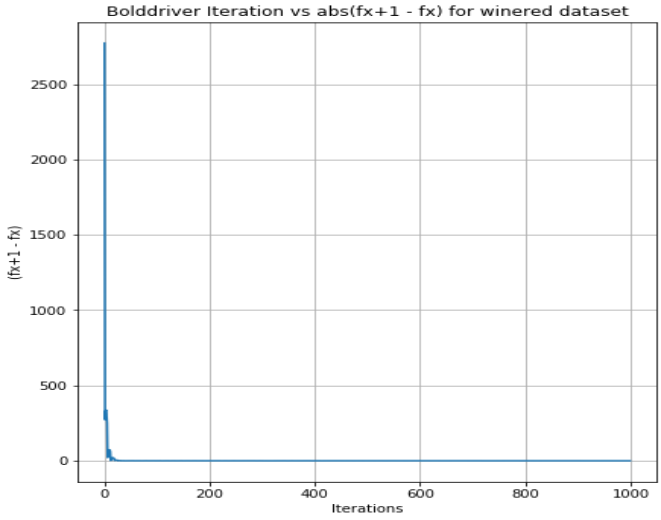


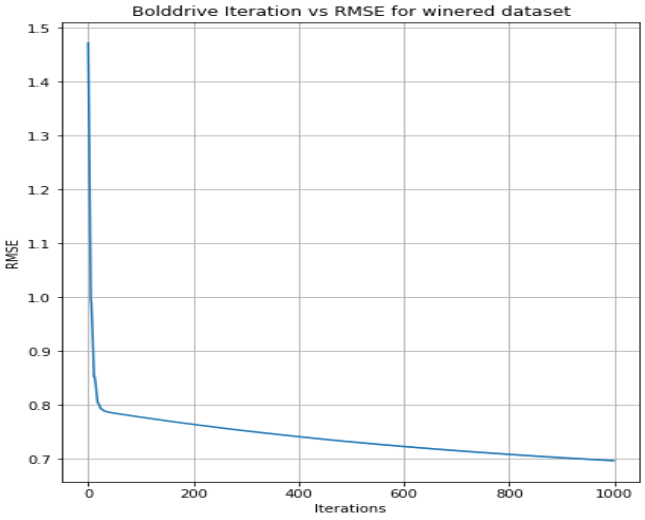



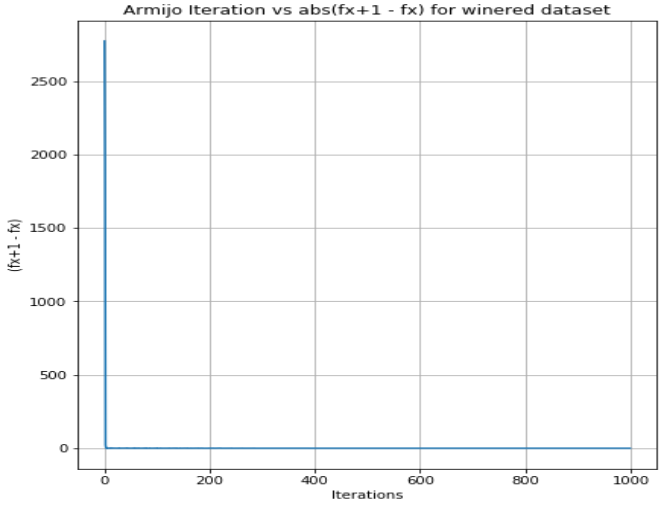



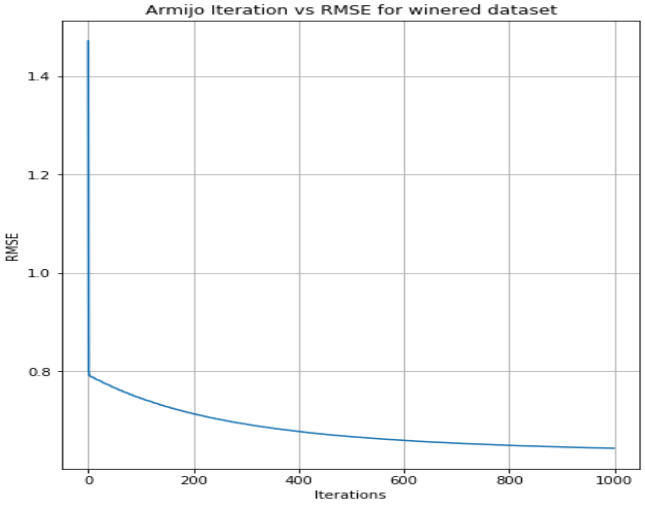



## Wine Red:

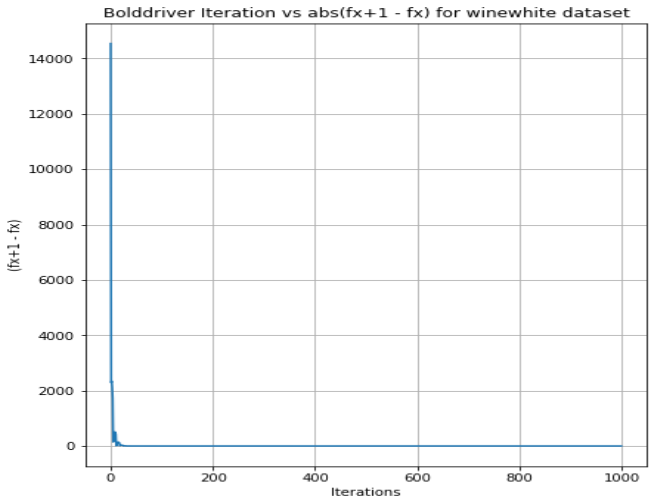


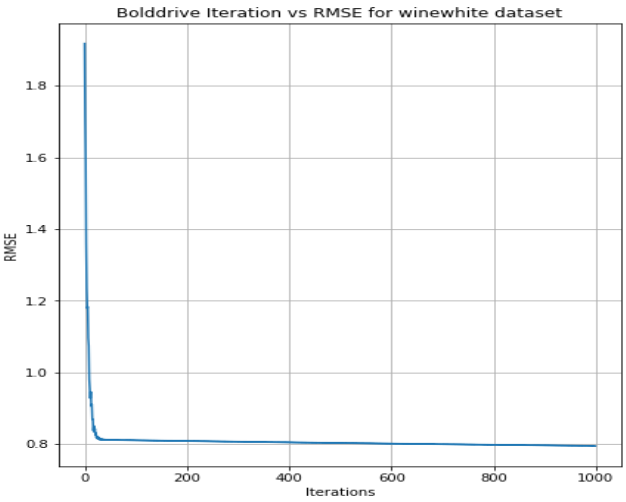



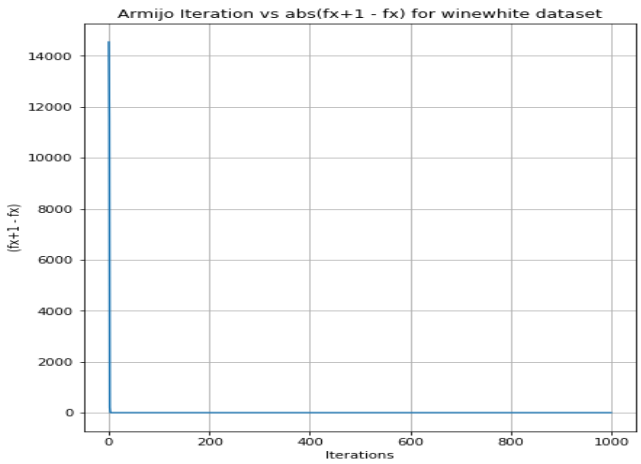



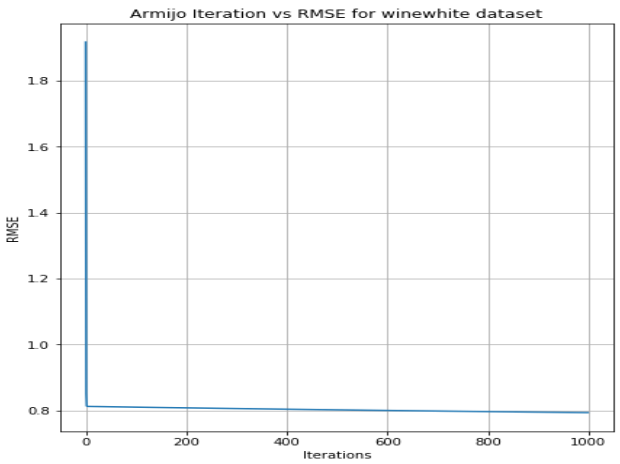




## Wine White:

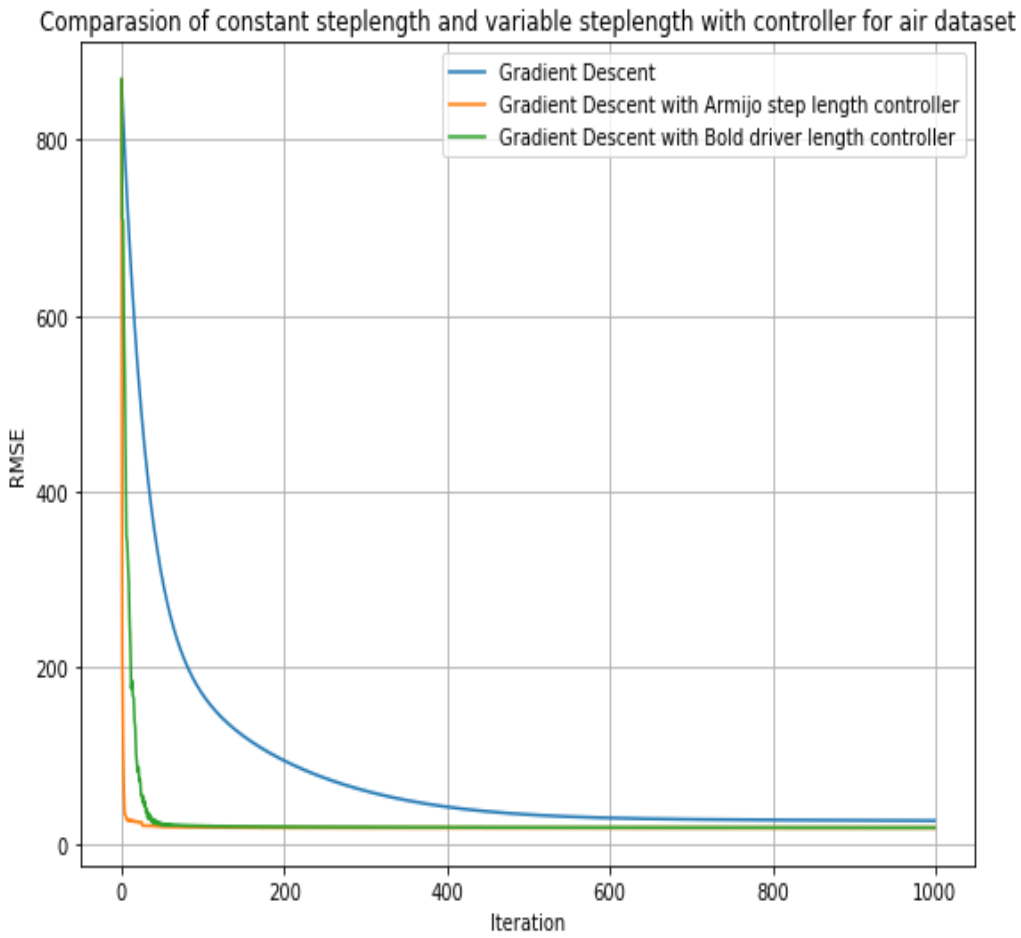


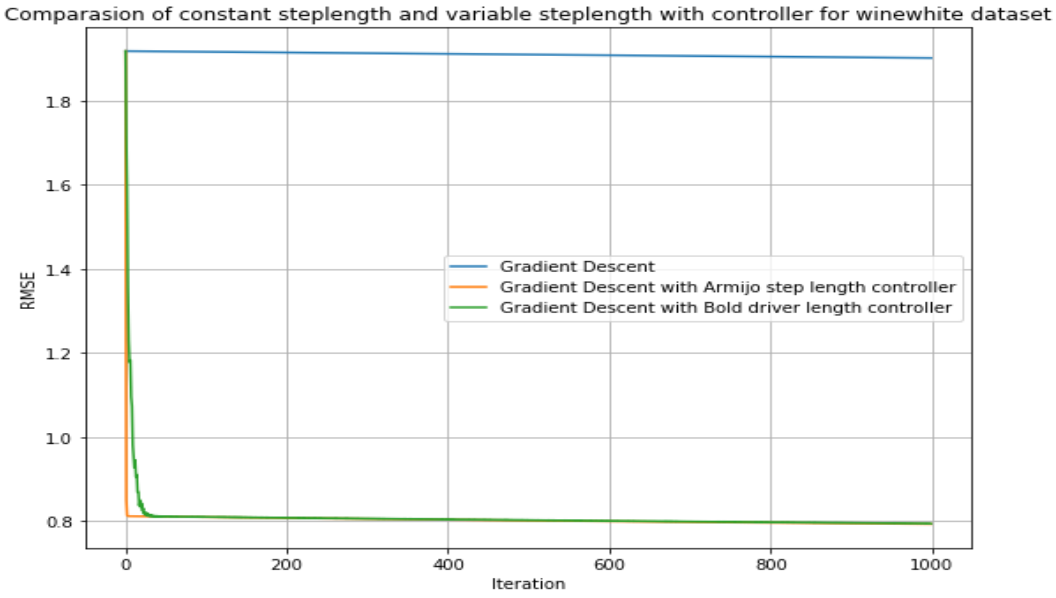



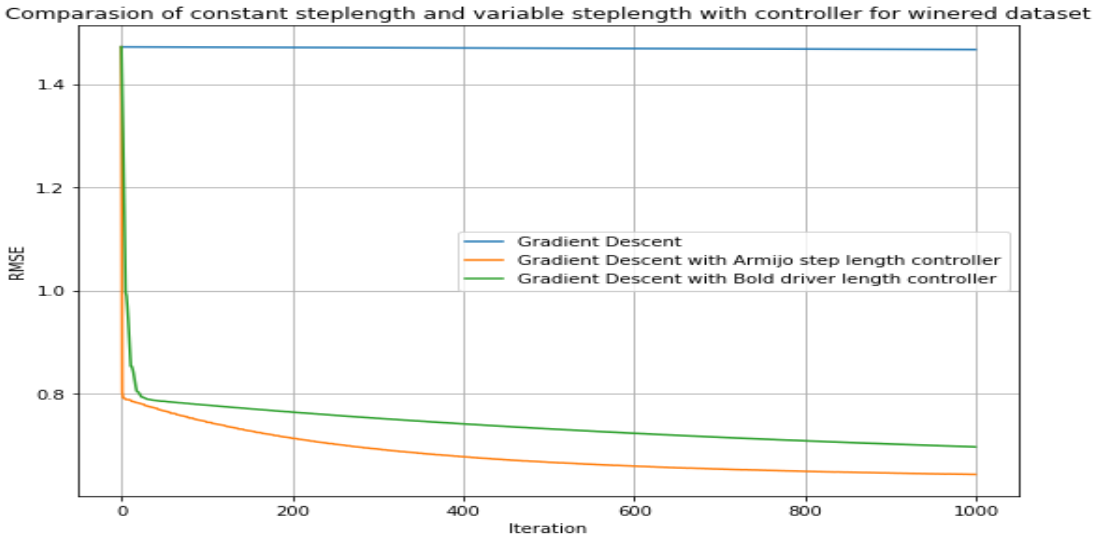



**Comparision of different step length algorithms:**






**Prediction samples for all three dataset:**

```
yPrediction = Regression_Line(X_test,betas)
yPrediction1 = Regression_Line(X_test1,betas2)
yPrediction2 = Regression_Line(X_test2,betas2)
```

| | Air_Dataset_Actual | Air_Dataset_Prediction | Winered_Dataset_Actual | Winered_Dataset_Prediction | Winewhite_Dataset_Actual | Winewhite_Dataset_Prediction |
|---|---|---|---|---|---|---|
| 597 | 231.52 | 177.225430 | 6.0 | 4.371153 | 6.0 | 4.242872 |
| 695 | 140.69 | 112.779825 | 6.0 | 4.885669 | 5.0 | 4.000467 |