In [1]:

```
# Basic Libraries
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt # we only need pyplot
sb.set() # set the default Seaborn style for graphics
```

In [4]:

```
prodata = pd.read_csv('ProgrammingQuiz_dataset.csv')
prodata.head()
```

Out[4]:

	Height	Weight	Diameter	Length
0	0.135	0.6770	0.420	0.530
1	0.150	0.7775	0.415	0.530
2	0.125	0.7680	0.425	0.545
3	0.150	0.8945	0.440	0.550
4	0.140	0.6065	0.380	0.525

Problem 1

Exploratory analysis

In [8]:

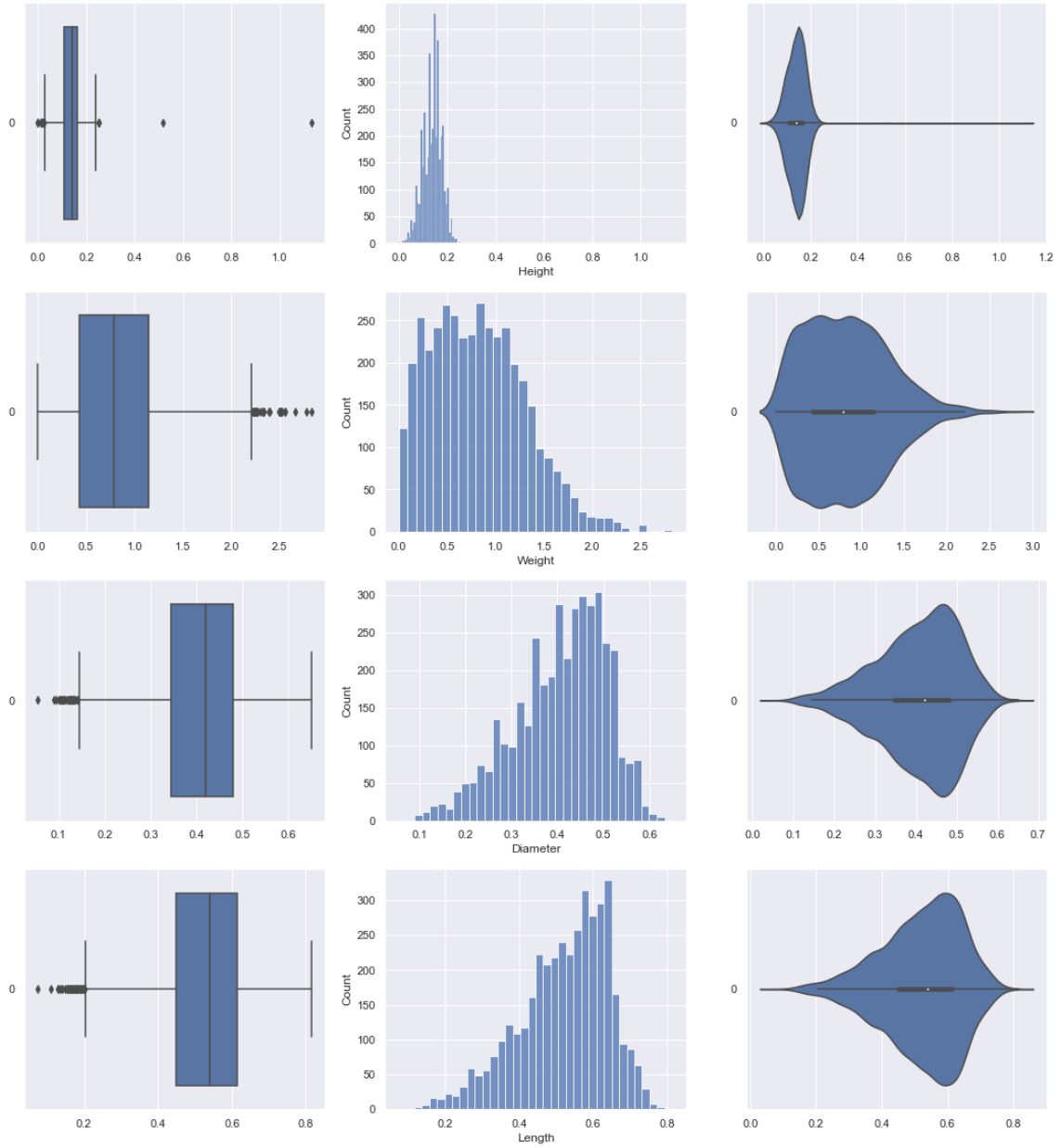
```
prodata.describe()
```

Out[8]:

	Height	Weight	Diameter	Length
count	4000.000000	4000.000000	4000.000000	4000.000000
mean	0.138874	0.819623	0.406095	0.521759
std	0.042092	0.489400	0.099668	0.120595
min	0.000000	0.002000	0.055000	0.075000
25%	0.110000	0.433500	0.345000	0.450000
50%	0.140000	0.787750	0.420000	0.540000
75%	0.165000	1.144500	0.480000	0.615000
max	1.130000	2.825500	0.650000	0.815000

In [10]:

```
# Draw the distributions of all variables
f, axes = plt.subplots(4, 3, figsize=(18, 20))

count = 0
for var in prodata:
    sb.boxplot(data = prodata[var], orient = "h", ax = axes[count,0])
    sb.histplot(data = prodata[var], ax = axes[count,1])
    sb.violinplot(data = prodata[var], orient = "h", ax = axes[count,2])
    count += 1
```



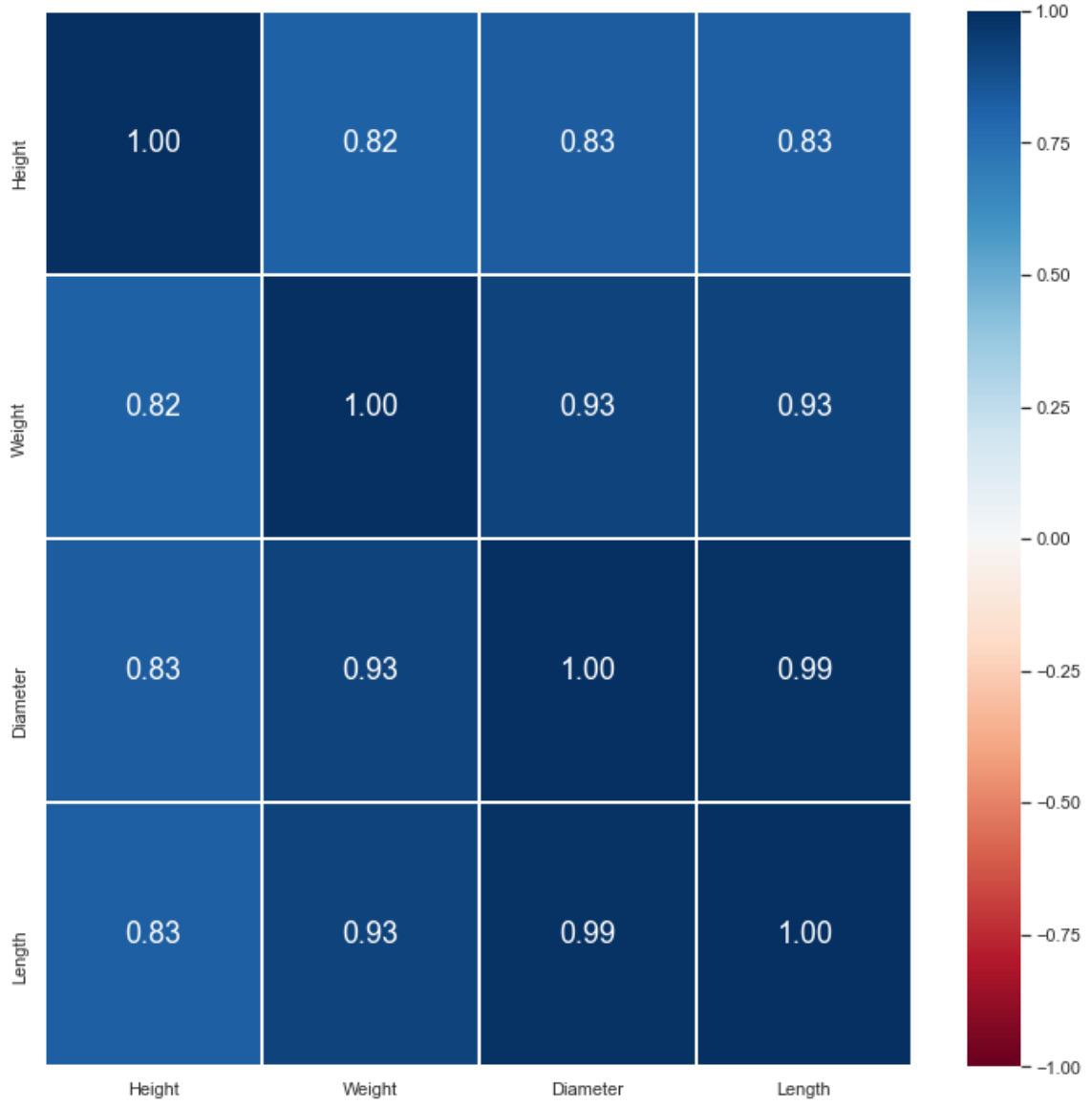
In [11]:

```
# Correlation Matrix
print(prodata.corr())

# Heatmap of the Correlation Matrix
f = plt.figure(figsize=(12, 12))
sb.heatmap(prodata.corr(), vmin = -1, vmax = 1, linewidths = 1,
           annot = True, fmt = ".2f", annot_kws = {"size": 18}, cmap = "RdBu"
```

```
              Height    Weight  Diameter    Length
Height      1.000000  0.817782  0.831975  0.825734
Weight      0.817782  1.000000  0.925831  0.925393
Diameter    0.831975  0.925831  1.000000  0.987054
Length      0.825734  0.925393  0.987054  1.000000
```

Out[11]:

<AxesSubplot:>



In [ ]:

```
```

Problem 2

Uni-variate linear regression

In [14]:

```
# Import essential models and functions from sklearn
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Create a Linear Regression object
linreg = LinearRegression()
```

In [15]:

```
# Datasets
Length = pd.DataFrame(prodata['Length'])
Height = pd.DataFrame(prodata['Height'])
Weight = pd.DataFrame(prodata['Weight'])
Diameter = pd.DataFrame(prodata['Diameter'])
```

In [16]:

```
# Split the Dataset into Train and Test
Le_train, Le_test, He_train, He_test = train_test_split(Length, Height, test_
Le_train, Le_test, We_train, We_test = train_test_split(Length, Weight, test_
Le_train, Le_test, Di_train, Di_test = train_test_split(Length, Diameter, tes


# Check the sample sizes
print("Train Set :", Le_train.shape, He_train.shape, We_train.shape)
print("Test Set  :", Le_test.shape, He_test.shape, We_test.shape)
```

```
Train Set : (2800, 1) (2800, 1) (2800, 1)
Test Set  : (1200, 1) (1200, 1) (1200, 1)
```

Le vs He

In [29]:

```
# Train the Linear Regression model
linreg.fit(Le_train, He_train)
```

Out[29]:

```
LinearRegression()
```

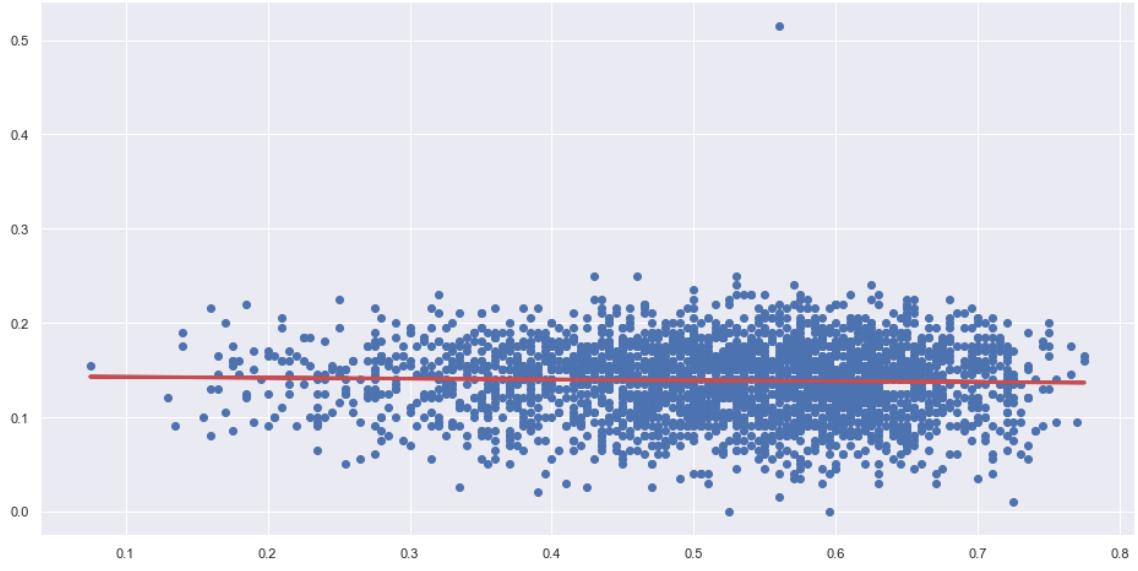
In [30]:

```
# Coefficients of the Linear Regression line
print('Intercept \t: b = ', linreg.intercept_)
print('Coefficients \t: a = ', linreg.coef_)
```

```
Intercept       : b =  [0.14329314]
Coefficients    : a =  [[-0.00889905]]
```

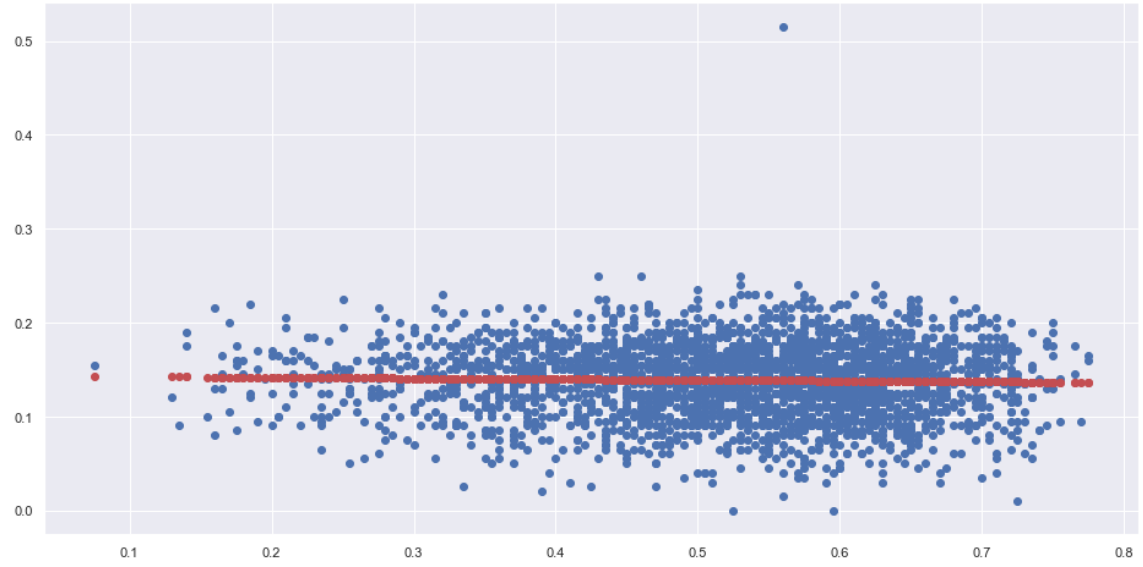
In [31]:

```
# Formula for the Regression line
regline_x = Le_train
regline_y = linreg.intercept_ + linreg.coef_ * Le_train

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_train, He_train)
plt.plot(regline_x, regline_y, 'r-', linewidth = 3)
plt.show()
```



In [32]:

```
# Predict Total values corresponding to HP Train
He_train_pred = linreg.predict(Le_train)

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_train, He_train)
plt.scatter(Le_train, He_train_pred, color = "r")
plt.show()
```



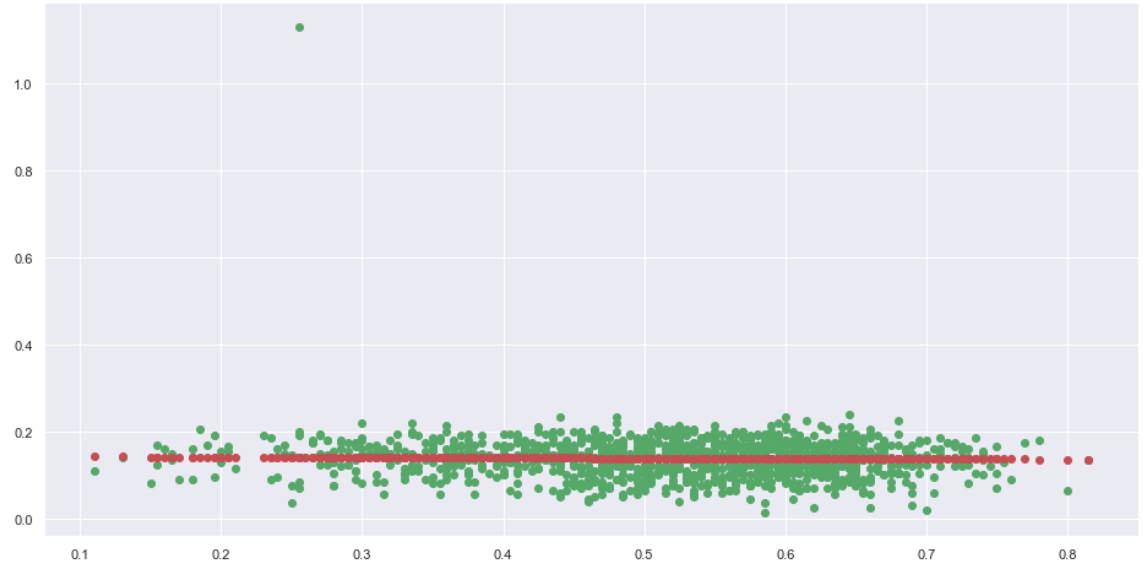
In [36]:

```
# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(Le_train, He_train))
r2_He1 = linreg.score(Le_train, He_train)

# Mean Squared Error (MSE)
def mean_sq_err(actual, predicted):
    '''Returns the Mean Squared Error of actual and predicted values'''
    return np.mean(np.square(np.array(actual) - np.array(predicted)))


mseG = mean_sq_err(He_train, He_train_pred)
print("Mean Squared Error (MSE) \t:", mseG)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mseG))
```

```
Explained Variance (R^2)        : 0.0007381830130536171
Mean Squared Error (MSE)        : 0.0015507997374430268
Root Mean Squared Error (RMSE)  : 0.03938019473597137
```

In [37]:

```
# Predict Total values corresponding to HP Train
He_test_pred = linreg.predict(Le_test)

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_test, He_test, color = 'g')
plt.scatter(Le_test, He_test_pred, color = "r")
plt.show()
```



In [38]:

```
# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(Le_test, He_test))
r2_He2 = linreg.score(Le_test, He_test)

# Mean Squared Error (MSE)
def mean_sq_err(actual, predicted):
    '''Returns the Mean Squared Error of actual and predicted values'''
    return np.mean(np.square(np.array(actual) - np.array(predicted)))


mseG = mean_sq_err(He_test, He_test_pred)
print("Mean Squared Error (MSE) \t:", mseG)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mseG))
```

```
Explained Variance (R^2)        : 0.002538161035543496
Mean Squared Error (MSE)        : 0.0022769036288325215
Root Mean Squared Error (RMSE)  : 0.047716911350510954
```

In [ ]:

```

```

In [ ]:

```
```

Le vs We

In [51]:

```
# Train the Linear Regression model
linreg.fit(Le_train, We_train)
```

Out[51]:

```
LinearRegression()
```

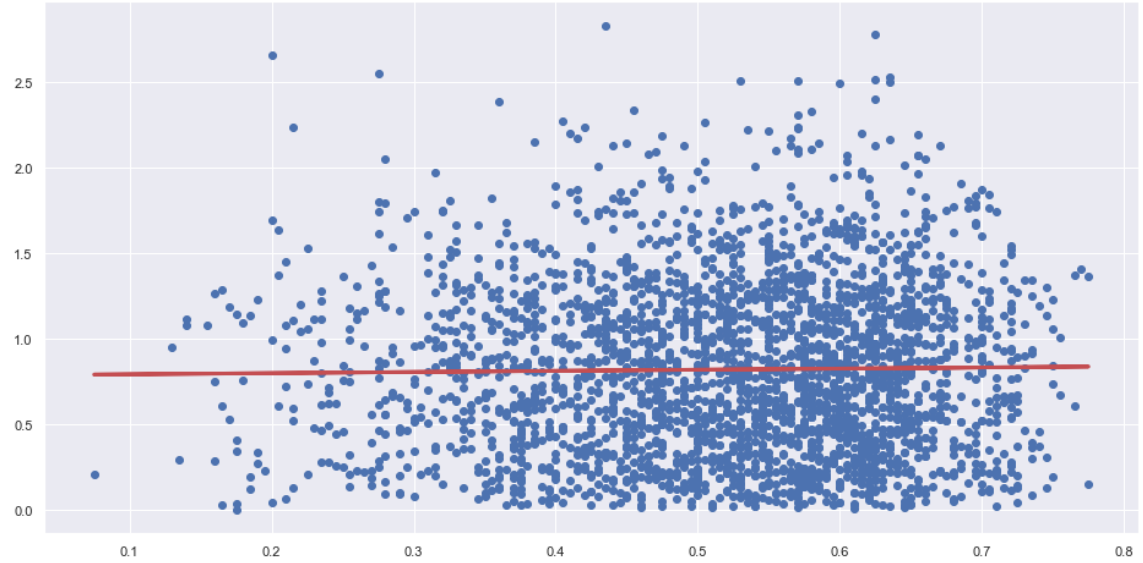
In [52]:

```
# Coefficients of the Linear Regression line
print('Intercept \t: b = ', linreg.intercept_)
print('Coefficients \t: a = ', linreg.coef_)
```

```
Intercept       : b =  [0.78386652]
Coefficients    : a =  [[0.06648293]]
```

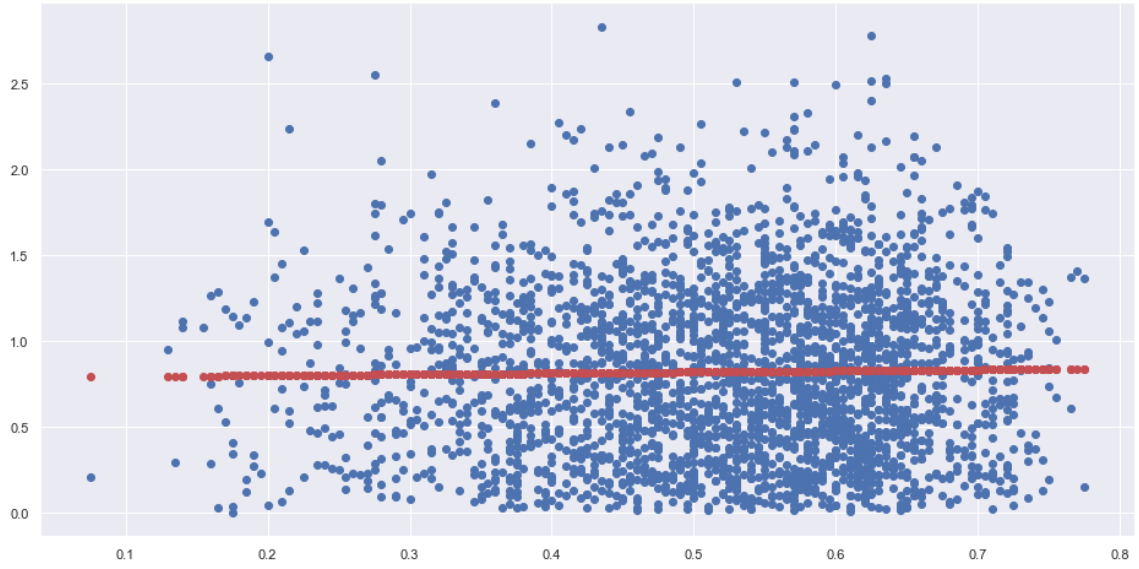
In [53]:

```
# Formula for the Regression line
regline_x = Le_train
regline_y = linreg.intercept_ + linreg.coef_ * Le_train

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_train, We_train)
plt.plot(regline_x, regline_y, 'r-', linewidth = 3)
plt.show()
```



In [54]:

```
# Predict Total values corresponding to HP Train
We_train_pred = linreg.predict(Le_train)

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_train, We_train)
plt.scatter(Le_train, We_train_pred, color = "r")
plt.show()
```



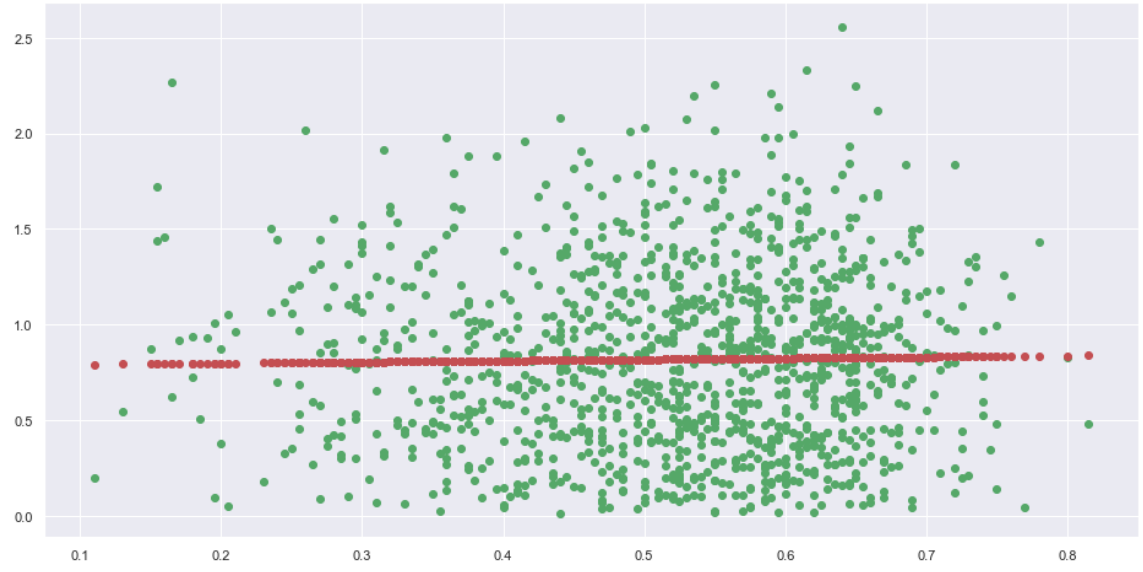
In [55]:

```
# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(Le_train, We_train))
r2_We1 = linreg.score(Le_train, We_train)

# Mean Squared Error (MSE)
def mean_sq_err(actual, predicted):
    '''Returns the Mean Squared Error of actual and predicted values'''
    return np.mean(np.square(np.array(actual) - np.array(predicted)))


mseG = mean_sq_err(We_train, We_train_pred)
print("Mean Squared Error (MSE) \t:", mseG)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mseG))
```

```
Explained Variance (R^2)        : 0.00026002828495230723
Mean Squared Error (MSE)        : 0.24583262366832778
Root Mean Squared Error (RMSE)  : 0.49581511036708814
```

In [56]:

```
# Predict Total values corresponding to HP Train
We_test_pred = linreg.predict(Le_test)

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_test, We_test, color = 'g')
plt.scatter(Le_test, We_test_pred, color = "r")
plt.show()
```



In [57]:

```
# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(Le_test, We_test))
r2_We2 = linreg.score(Le_test, We_test)

# Mean Squared Error (MSE)
def mean_sq_err(actual, predicted):
    '''Returns the Mean Squared Error of actual and predicted values'''
    return np.mean(np.square(np.array(actual) - np.array(predicted)))


mseG = mean_sq_err(We_test, We_test_pred)
print("Mean Squared Error (MSE) \t:", mseG)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mseG))
```

```
Explained Variance (R^2)        : -3.768345747956481e-05
Mean Squared Error (MSE)        : 0.22441677522699724
Root Mean Squared Error (RMSE)  : 0.4737264772281546
```

In [ ]:

```
```

Le vs Di

In [58]:

```
# Train the Linear Regression model
linreg.fit(Le_train, Di_train)
```

Out[58]:

```
LinearRegression()
```

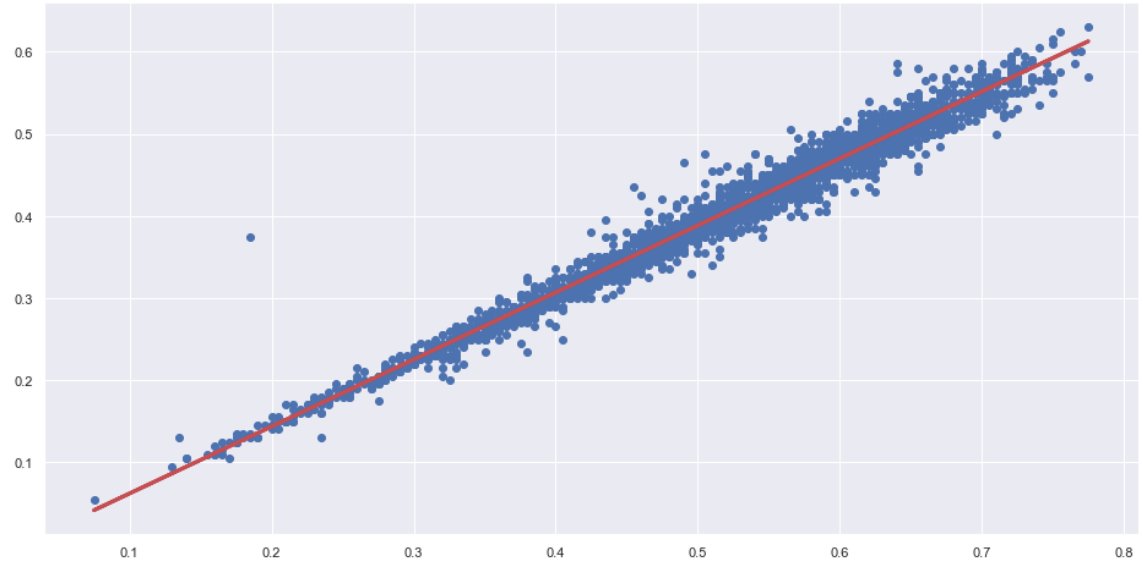
In [59]:

```
# Coefficients of the Linear Regression line
print('Intercept \t: b = ', linreg.intercept_)
print('Coefficients \t: a = ', linreg.coef_)
```

```
Intercept       : b =  [-0.01956451]
Coefficients    : a =  [[0.81592806]]
```

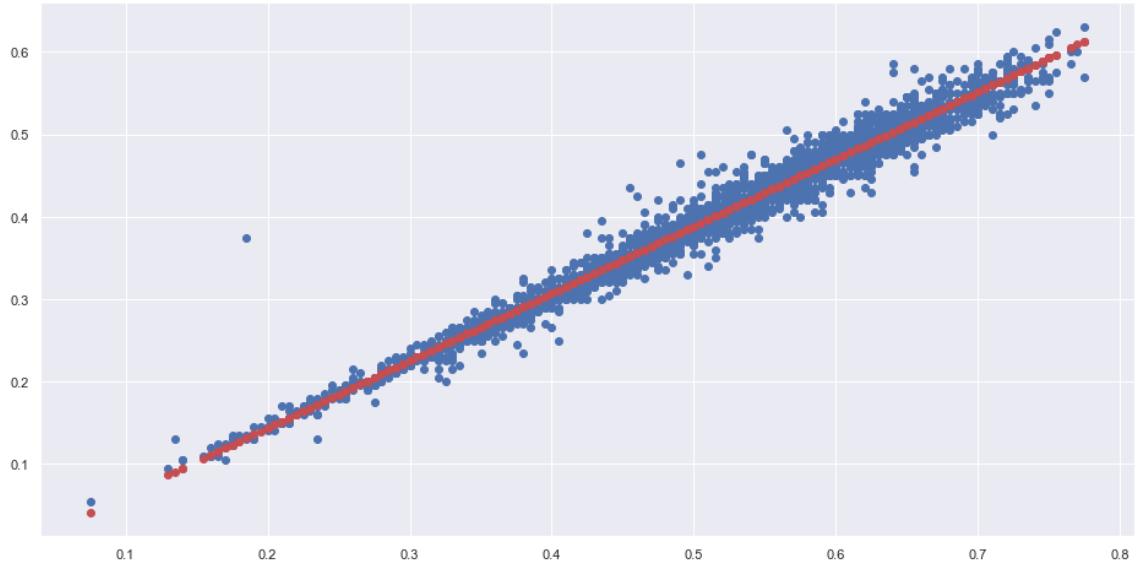
In [60]:

```
# Formula for the Regression line
regline_x = Le_train
regline_y = linreg.intercept_ + linreg.coef_ * Le_train

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_train, Di_train)
plt.plot(regline_x, regline_y, 'r-', linewidth = 3)
plt.show()
```



In [61]:

```
# Predict Total values corresponding to HP Train
Di_train_pred = linreg.predict(Le_train)

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_train, Di_train)
plt.scatter(Le_train, Di_train_pred, color = "r")
plt.show()
```



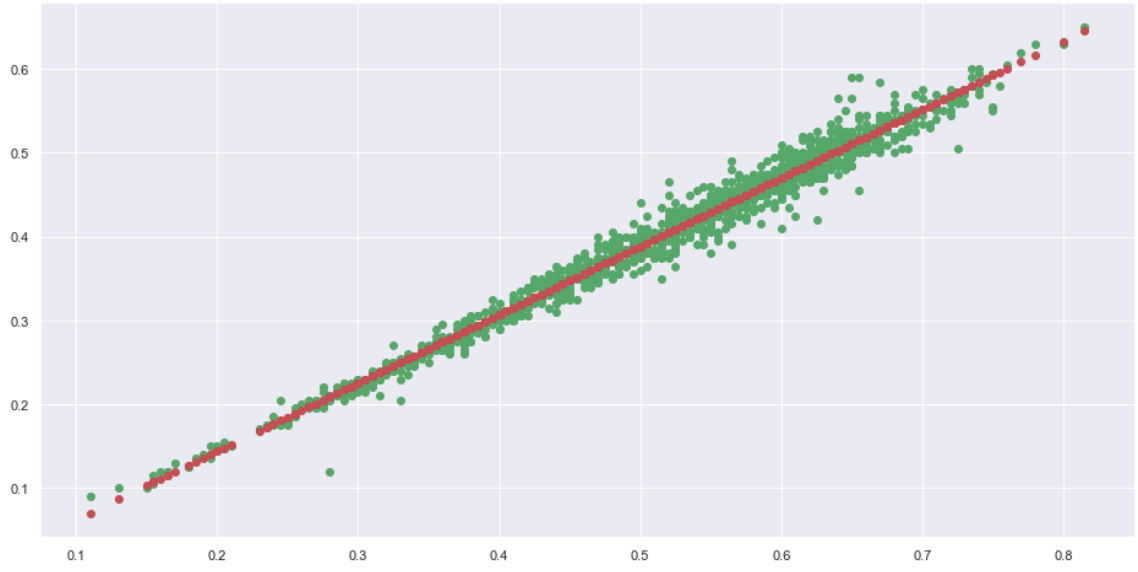
In [62]:

```
# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(Le_train, Di_train))
r2_Di1 = linreg.score(Le_train, Di_train)

# Mean Squared Error (MSE)
def mean_sq_err(actual, predicted):
    '''Returns the Mean Squared Error of actual and predicted values'''
    return np.mean(np.square(np.array(actual) - np.array(predicted)))


mseG = mean_sq_err(Di_train, Di_train_pred)
print("Mean Squared Error (MSE) \t:", mseG)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mseG))
```

```
Explained Variance (R^2)        : 0.9734443303616198
Mean Squared Error (MSE)        : 0.0002627257718160776
Root Mean Squared Error (RMSE)  : 0.016208817718022423
```

In [63]:

```
# Predict Total values corresponding to HP Train
Di_test_pred = linreg.predict(Le_test)

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.scatter(Le_test, Di_test, color = 'g')
plt.scatter(Le_test, Di_test_pred, color = "r")
plt.show()
```



In [64]:

```
# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(Le_test, Di_test))
r2_Di2 = linreg.score(Le_test, Di_test)

# Mean Squared Error (MSE)
def mean_sq_err(actual, predicted):
    '''Returns the Mean Squared Error of actual and predicted values'''
    return np.mean(np.square(np.array(actual) - np.array(predicted)))


mseG = mean_sq_err(Di_test, Di_test_pred)
print("Mean Squared Error (MSE) \t:", mseG)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mseG))
```

```
Explained Variance (R^2)        : 0.9761886972909731
Mean Squared Error (MSE)        : 0.00023858356447541194
Root Mean Squared Error (RMSE)  : 0.01544615047432246
```

In [ ]:

```
```

Problem 3

In [67]:

```
# Extract Response and Predictors
y = pd.DataFrame(prodata['Length'])
X = pd.DataFrame(prodata[['Weight','Height','Diameter']])

# Split the Dataset into random Train and Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

# Check the sample sizes
print("Train Set :", X_train.shape, y_train.shape)
print("Test Set  :", X_test.shape, y_test.shape)

# Create a Linear Regression object
linreg = LinearRegression()

# Train the Linear Regression model
linreg.fit(X_train, y_train)
```

```
Train Set : (2800, 3) (2800, 1)
Test Set  : (1200, 3) (1200, 1)
```

Out[67]:

```
LinearRegression()
```

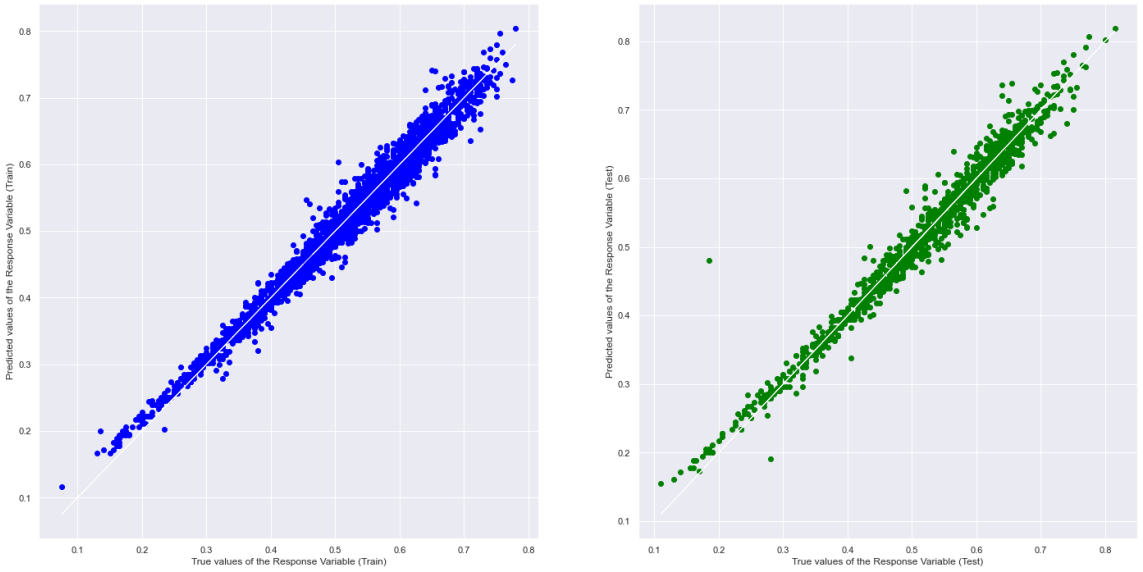
In [68]:

```
print('Intercept \t: b = ', linreg.intercept_)
print('Coefficients \t: a = ', linreg.coef_)
```

```
Intercept       : b =  [0.05508533]
Coefficients    : a =  [[ 1.85772909e-02 -6.12292047e-04  1.11
151946e+00]]
```

In [69]:

```
# Predict SalePrice values corresponding to Predictors
y_train_pred = linreg.predict(X_train)
y_test_pred = linreg.predict(X_test)

# Plot the Predictions vs the True values
f, axes = plt.subplots(1, 2, figsize=(24, 12))
axes[0].scatter(y_train, y_train_pred, color = "blue")
axes[0].plot(y_train, y_train, 'w-', linewidth = 1)
axes[0].set_xlabel("True values of the Response Variable (Train)")
axes[0].set_ylabel("Predicted values of the Response Variable (Train)")
axes[1].scatter(y_test, y_test_pred, color = "green")
axes[1].plot(y_test, y_test, 'w-', linewidth = 1)
axes[1].set_xlabel("True values of the Response Variable (Test)")
axes[1].set_ylabel("Predicted values of the Response Variable (Test)")
plt.show()
```



In [70]:

```
print("Explained Variance (R^2) on Train Set \t:", linreg.score(X_train, y_tr
print("Mean Squared Error (MSE) on Train Set \t:", mean_squared_error(y_train
print("Mean Squared Error (MSE) on Test Set \t:", mean_squared_error(y_test,
```

```
Explained Variance (R^2) on Train Set   : 0.9772255400075355
Mean Squared Error (MSE) on Train Set   : 0.000324591767423733
93
Mean Squared Error (MSE) on Test Set    : 0.000444499551448916
6
```

In [ ]:

```

```