# Regression Analysis Using Various Regressors in Scikit-learn

The regression analysis is to determine how locking in a fixed rate of COF for a TOB compared to leaving the funding cost floating at the 7-day rate based on the historical data, and to see if we can identify the driving factors if they exist, and use the factors to make statistic predictions for the differences in COF for th two funding strategies. For this analysis, we use a fixed term of 4y for the analysis, i.e., we'll compare the COF for fixing the COF at the market SIFMA swap rate, and the average COF of the floating COF realized during a 4y time window following the COF fixing date by calculating the difference between the two COF, which is labelled as'Realized_4yPremium' in the data file.

Since the fixed COF is a known number that remains unchanged for the remaining 4 years, the realized premium will be determined by the 7day SIFMA rate resets to be realized over the 4y following the COF fixing date. Basic macroeconomics suggusts that the resets will depend on macroeconomic data such as GDP, inflation and inflation expectations, etc. To build an econometric regression model, or a forecast model, the model inputs can only use data up to the COF fixed date, any data after that should not be used as model inputs.

One may be tempted to use the macroeconomic data such as those mentioned above as the independent variables. However, there is a serious problem that would limit this approach to be of any use. The first problem is that 7-day SIFMA resets should be correlated to the macro data on the reset date, our dependent variable, the Realized4yPremium is for the 4y time window following the COF fixing date, hence the the macroeconomic data to be used for the model should be the expected data for the same time window, which do not exist. Even the data exist, we know that the correlation bwteen the short rate reset and contemperaneous macroeconomic data is weak at the best.

Our hypothesis is that the futures market for the short term interest rates should reflect market participants' expection of the macroeconomic conditions in the future. The expectation is quantitatively measured in the market rates of the term swaps. Therefore we believe that the term swap rate quoted on the COF fixing date can be used as the independent variables, or features in ML terminology, for the model. Fortunately liquid markets exist for both the taxable and tax-exempt swap market, we use the swap rates traded in the tax-exempt market as the model inputs as TOBs are for tax-exempt bonds.

Historical SIFMA swap rates of maturities of 7days,1y, 2y,3y,4y, and 5y since 2007 are collected for this analysis.

In [1]:

```
#Import Libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
```

In [2]:

```
#Load Data
#Data consists of historical SIFMA swap rates of maturities of 7days,1y, 2y,3y,4y, and 5y.
Data = pd.read_csv('C:/Users/tingl/OneDrive/Documents/Python/Muni/Data4Realized4yPremium.csv')
```

In [3]:
```
#data summary info
Data.info()
#Data.count
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 591 entries, 0 to 590
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   D4yPrior_Date       591 non-null    object
 1   Realized_4yPremium  591 non-null    float64
 2   T4yPrior_SIFMA      591 non-null    float64
 3   T4yPrior_1ySwap     591 non-null    float64
 4   T4yPrior_2ySwap     591 non-null    float64
 5   T4yPrior_3ySwap     591 non-null    float64
 6   T4yPrior_4ySwap     591 non-null    float64
 7   T4yPrior_5ySwap     591 non-null    float64
dtypes: float64(7), object(1)
memory usage: 37.1+ KB
```

In [4]:
```
Data.head()
```

Out[4]:

| | D4yPrior_Date | Realized_4yPremium | T4yPrior_SIFMA | T4yPrior_1ySwap | T4yPrior_2ySwap | T4yPrior_3ySwap | T4yPrior_4ySwap | T4yPrior_5ySwap |
|---|---|---|---|---|---|---|---|---|
| **0** | 4/25/2018 | 1.27 | 1.75 | 1.665 | 1.862 | 1.983 | 2.06 | 2.12 |
| **1** | 4/18/2018 | 1.21 | 1.81 | 1.644 | 1.832 | 1.939 | 2.00 | 2.05 |
| **2** | 4/11/2018 | 1.13 | 1.72 | 1.589 | 1.772 | 1.869 | 1.93 | 1.98 |
| **3** | 4/4/2018 | 1.15 | 1.60 | 1.577 | 1.772 | 1.877 | 1.95 | 2.00 |
| **4** | 3/28/2018 | 1.11 | 1.58 | 1.556 | 1.747 | 1.847 | 1.92 | 1.98 |

In [5]:
```
Data.tail()
```

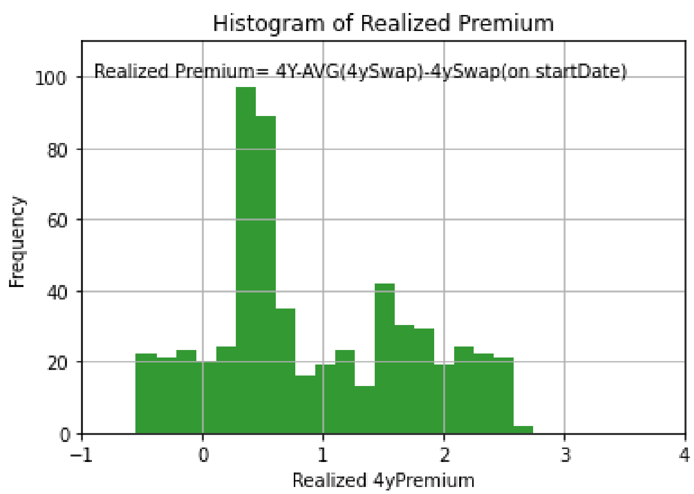
Out[5]:

| | D4yPrior_Date | Realized_4yPremium | T4yPrior_SIFMA | T4yPrior_1ySwap | T4yPrior_2ySwap | T4yPrior_3ySwap | T4yPrior_4ySwap | T4yPrior_5ySwap |
|---|---|---|---|---|---|---|---|---|
| **586** | 1/31/2007 | 1.99 | 3.50 | 3.602 | 3.562 | 3.557 | 3.56 | 3.61 |
| **587** | 1/24/2007 | 1.97 | 3.61 | 3.599 | 3.540 | 3.524 | 3.55 | 3.59 |
| **588** | 1/17/2007 | 1.94 | 3.62 | 3.591 | 3.535 | 3.522 | 3.54 | 3.59 |
| **589** | 1/10/2007 | 1.85 | 3.63 | 3.548 | 3.461 | 3.437 | 3.47 | 3.51 |
| **590** | 1/3/2007 | 1.83 | 3.45 | 3.524 | 3.466 | 3.412 | 3.45 | 3.48 |

In [6]:
```
#check the dependent variable Y
y=Data['Realized_4yPremium']
y.describe()
```

Out[6]:
```
count    591.000000
mean       0.900931
std        0.822446
min       -0.550000
25%        0.330000
50%        0.600000
75%        1.590000
max        2.750000
Name: Realized_4yPremium, dtype: float64
```

In [7]:
```
#show histogram of y
n, bins, patches = plt.hist(y, 20, density=0, facecolor='g', alpha=0.8)
plt.xlabel('Realized 4yPremium')
plt.ylabel('Frequency')
plt.title('Histogram of Realized Premium')
plt.text(-0.9, 100, r'Realized Premium= 4Y-AVG(4ySwap)-4ySwap(on startDate)')
plt.axis([-1, 4, 0, 110])
plt.grid(True)
plt.show()
```

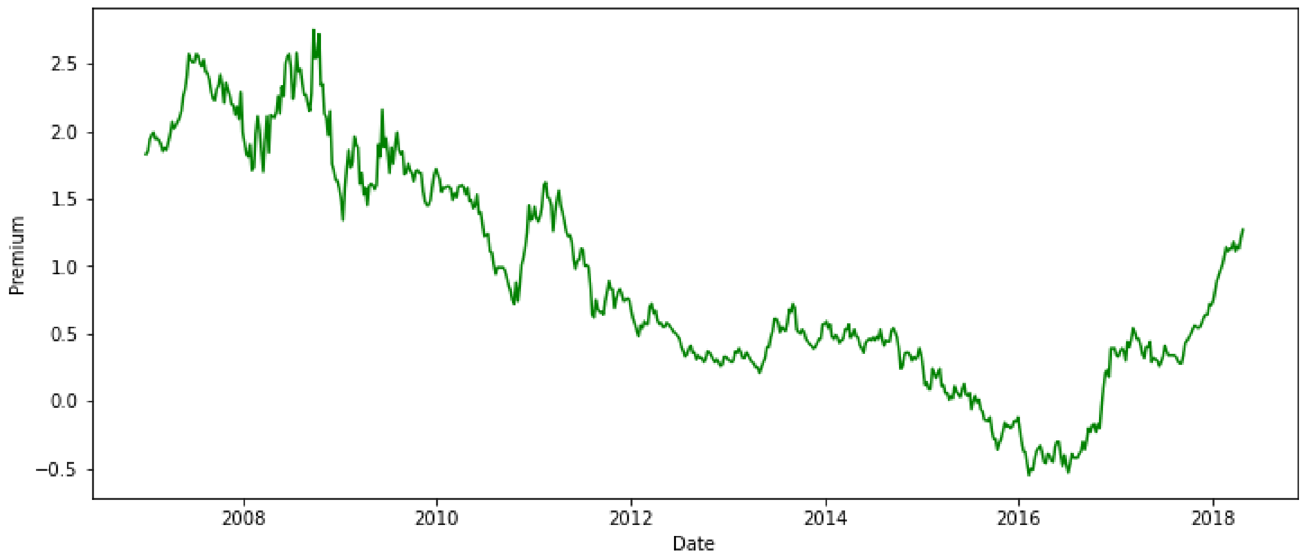



In [8]:
```
#n and bins returns values for counts and bin borders
bins
```

Out[8]:
```
array([-0.55 , -0.385, -0.22 , -0.055,  0.11 ,  0.275,  0.44 ,  0.605,
        0.77 ,  0.935,  1.1  ,  1.265,  1.43 ,  1.595,  1.76 ,  1.925,
        2.09 ,  2.255,  2.42 ,  2.585,  2.75 ])
```

In [9]:
```
plt.figure(figsize=(12,5))
DT=pd.to_datetime(Data['D4yPrior_Date'])
type(DT)
ax=plt.plot(DT, y, color='g')
plt.xlabel('Date')
plt.ylabel("Premium")
```

Out[9]: Text(0, 0.5, 'Premium')




In [10]:
```
#Before defining y and x variables, we can check the correlation matrix first.  We need to drop the Date col first
xy=Data.drop(['D4yPrior_Date'], axis=1)
xy.corr()
```

Out[10]:

| | Realized_4yPremium | T4yPrior_SIFMA | T4yPrior_1ySwap | T4yPrior_2ySwap | T4yPrior_3ySwap | T4yPrior_4ySwap | T4yPrior_5ySwap |
|---|---|---|---|---|---|---|---|
| **Realized_4yPremium** | 1.000000 | 0.669999 | 0.726648 | 0.781776 | 0.838441 | 0.882804 | 0.915235 |
| **T4yPrior_SIFMA** | 0.669999 | 1.000000 | 0.954004 | 0.931701 | 0.900182 | 0.865537 | 0.828883 |
| **T4yPrior_1ySwap** | 0.726648 | 0.954004 | 1.000000 | 0.990231 | 0.963548 | 0.931086 | 0.894675 |
| **T4yPrior_2ySwap** | 0.781776 | 0.931701 | 0.990231 | 1.000000 | 0.990296 | 0.968402 | 0.938875 |
| **T4yPrior_3ySwap** | 0.838441 | 0.900182 | 0.963548 | 0.990296 | 1.000000 | 0.992941 | 0.974956 |
| **T4yPrior_4ySwap** | 0.882804 | 0.865537 | 0.931086 | 0.968402 | 0.992941 | 1.000000 | 0.994179 |
| **T4yPrior_5ySwap** | 0.915235 | 0.828883 | 0.894675 | 0.938875 | 0.974956 | 0.994179 | 1.000000 |

We see that the Realized_4yPremium has high correlations to all of the swap rates, so it looks promissing that our hypothesis will be statistically validated. Because of the the swap rates are very correlated amongst themselves, it's likely that we'll not need all of the swap rates as the independent variables. Based on the correlation matrix, T4yPrior_4ySwap and T4yPrior_5ySwap have the highest correlation coefficients. We'll first run regression analysis using all input variables as the independent factors, then using only T4yPrior_4ySwap and T4yPrior_5ySwap as the independent factors as they are mostly correlcted to the dependent variable.

In [11]:

```
#Define x, and y, note that we need to use numpy array for the x and y to be used as input for sklearn regressors
y=Data['Realized_4yPremium'].values
#dropping the Date col and y col
x=Data.drop(['Realized_4yPremium','D4yPrior_Date'], axis=1).values
#check what x is
print(x)
```

```
[[1.75  1.665 1.862 1.983 2.06  2.12 ]
 [1.81  1.644 1.832 1.939 2.    2.05 ]
 [1.72  1.589 1.772 1.869 1.93  1.98 ]
 ...
 [3.62  3.591 3.535 3.522 3.54  3.59 ]
 [3.63  3.548 3.461 3.437 3.47  3.51 ]
 [3.45  3.524 3.466 3.412 3.45  3.48 ]]
```

In [12]:

```
#Split dataset into training set and test set, with 25% data randomly selected as the test data
x_train,  x_test, y_train, y_test =train_test_split(x,y,test_size=0.25, random_state=1)
```

In [13]:

```
#train linear regression model on the training data
LR=LinearRegression()
LR.fit(x_train,y_train)
print("R_Square:", LR.score(x_train,y_train))
```

```
R_Square: 0.9068694874789982
```

In [14]:

```
#prediction using test data
y_pred=LR.predict(x_test)
```

In [15]:

```
#Calc r-square for the test data
print("R_Square:", r2_score(y_pred, y_test))
```

```
R_Square: 0.9012004874948562
```

In [16]:

```
#Cacl mean-squared-error for the test data
print("MSE:", mean_squared_error(y_pred, y_test))
```

```
MSE: 0.059670867580460404
```

In [17]:
```
# Run regression analysis using only the 4y and 5y swap rates as the independent variable
x=Data.drop(['Realized_4yPremium','D4yPrior_Date','T4yPrior_SIFMA','T4yPrior_1ySwap','T4yPrior_2ySwap','T4yPrior_3ySwap'], axis=1).value
y=Data['Realized_4yPremium'].values
#Split dataset into training set and test set, with 25% data randomly selected as the test data
x_train,  x_test, y_train, y_test =train_test_split(x,y,test_size=0.25, random_state=1)
```

In [18]:
```
#train linear regression model on the training data
LR=LinearRegression()
LR.fit(x_train,y_train)
print("R_Square:", LR.score(x_train,y_train))
```

```
R_Square: 0.9010580365240635
```

In [19]:
```
#prediction using test data
y_pred=LR.predict(x_test)
#Calc r-square for the test data
print("R_Square:", r2_score(y_pred, y_test))
```

```
R_Square: 0.8965966880946858
```

In [20]:
```
#Cacl mean-squared-error for the test data
print("MSE:", mean_squared_error(y_pred, y_test))
```

```
MSE: 0.06176518979213677
```

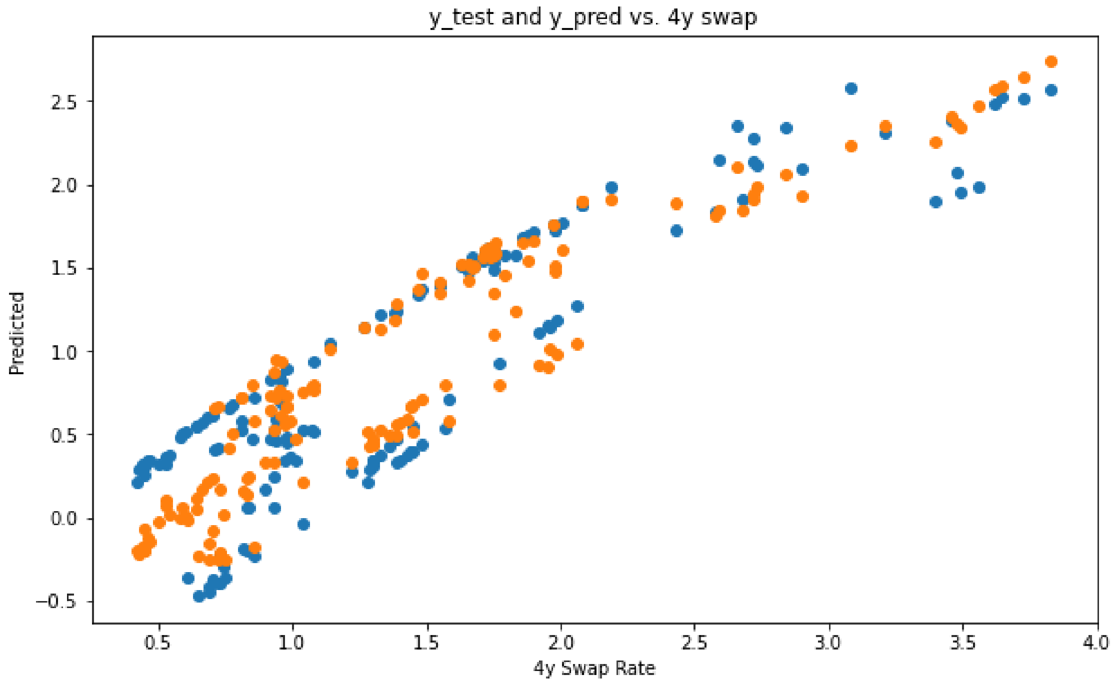
In [21]:
```
#print out regression coefficients
print("Coefficients: \n", LR.coef_)
print("Intercept: \n", LR.intercept_)
```

```
Coefficients:
 [-2.20941306  3.18214232]
Intercept:
 -1.147405456903586
```

In [22]:
```
#plot regression
x1=x_test[:,0]
x2=x_test[:,1]

plt.figure(figsize=(10,6))
plt.scatter(x1, y_test)
plt.scatter(x1, y_pred)
#note that x2 is not fixed for the y vs. x1 plot
plt.xlabel("4y Swap Rate")
plt.title("y_test and y_pred vs. 4y swap")
plt.ylabel("Predicted")
```

Out[22]: Text(0, 0.5, 'Predicted')




we see that results of using all swaps rates for the regression analysis are only marginally better than using only the 4y and 5y swap rates. Therefore we can use the results of using only the 4y and 5y swap rates for simplicity.

Given the regression results, we can conclude that we can predict, with ~90% confidence based on the 4y and 5y swap rates on the fixing date, the premium of fixing the OCF over the floating to be realized for the 4 years following the fixing date.

The regression analysis above can also be carried out in MSExcel, even though the data setup will not be as easy as in Python. In the following, we'll do more analysis not available in Excel. We'll show how to use various ML models available in sklearn, a Python library.

Before getting into using nonlinear models, we'll experiment with PCA to show how PCA works. PCA is a commonly used method for dimension reduction. With PCA, we use the so-called Principal Component, which is a linear combination of the original factors/features. The principal components are constructed to be orthogonal to each other, i.e. the correlation matrix for the principal components only have non-zero values along the diagnal of the matrix. We'll run linear regression using the principal components as the independent variables.

In [23]:

```
#import PCA from library
from sklearn.decomposition import PCA
```

In [24]:

```
x4pca=Data.drop(['Realized_4yPremium','D4yPrior_Date','T4yPrior_SIFMA'], axis=1).values
y=Data['Realized_4yPremium'].values
```

```
pca=PCA(n_components=2)
#we choose the first 2 Principal components so that we'll have the same number of independent variables for linear regression.
pca.fit(x4pca)
xpca=pca.transform(x4pca)
xpca.shape
```

Out[24]: (591, 2)

In [25]:

```
xpca
```

Out[25]:

```
array([[ 1.56084946, -0.18269575],
       [ 1.46262077, -0.22832726],
       [ 1.3181659 , -0.24943937],
       ...,
       [ 5.1909862 , -0.3475778 ],
       [ 5.03498285, -0.37972055],
       [ 4.99363474, -0.39304672]])
```

In [26]:

```
#to see the data in a better format
xdf=pd.DataFrame(xpca, columns=["PC1","PC2"])
xdf.head()
```

Out[26]:

| | PC1 | PC2 |
|---|---|---|
| **0** | 1.560849 | -0.182696 |
| **1** | 1.462621 | -0.228327 |
| **2** | 1.318166 | -0.249439 |
| **3** | 1.332798 | -0.221610 |
| **4** | 1.276534 | -0.226637 |

In [27]:

```
#Split dataset into training set and test set
xpca_train,  xpca_test, y_train, y_test =train_test_split(xpca,y,test_size=0.25, random_state=1)
```

In [28]:

```
#train linear regression model on training data
PCA_LR=LinearRegression()
PCA_LR.fit(xpca_train,y_train)
print("R_Square:", PCA_LR.score(xpca_train,y_train))
```

```
R_Square: 0.8672257729005755
```

In [29]:

```
#Cacl mean-squared-error for the test data
print("MSE:", mean_squared_error(y_pred, y_test))
```

```
MSE: 0.061765189792136777
```

In [30]:

```
#PCA contribution to variance
pca.explained_variance_ratio_
```

Out[30]:

```
array([0.97096734, 0.0276632 ])
```

we'll now use the following nonlinear regressiors: Decision Tree, Random Forrest, SVM and Neural Network

In [31]:

```
#import Decision Tree
from sklearn.tree import DecisionTreeRegressor
#create a model
DT_data =DecisionTreeRegressor().fit(x_train,y_train)
```

In [32]:

```
#print out the r2score
print("R_Square:", DT_data.score(x_train,y_train))
```

```
R_Square: 0.9998278368029373
```

In [33]:

```
#calculate the r2score using training data to see it matched the number above
DT_ypred=DT_data.predict(x_train)
print("R_Square:", r2_score(DT_ypred, y_train))
```

```
R_Square: 0.9998278071576671
```

In [34]:

```
##calculate the test score
DT_ytest=DT_data.predict(x_test)
print(r2_score(DT_ytest, y_test))
```

```
0.9057963249073541
```

In [35]:

```
#Cacl mean-squared-error for the test data
print("MSE:", mean_squared_error(DT_ytest, y_test))
```

```
MSE: 0.06207755255255256
```

In [36]:

```
#import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(n_estimators=10,random_state=12)
```

In [37]:

```
#Split dataset into training set and test set and train the model
x_train,  x_test, y_train, y_test =train_test_split(x,y,test_size=0.3, random_state=1)
```

```
rf_model=rf.fit(x_train,y_train)
```

In [38]:

```
#print out the R2score
print("R_Square:", rf_model.score(x_train,y_train))
```

```
R_Square: 0.9909112335707475
```

In [39]:

```
#run test data and calculate and print out the R2score for test
y_rfpred=rf_model.predict(x_test)
print("R_Square:", r2_score(y_rfpred, y_test))
```

```
R_Square: 0.9420339282121755
```

In [40]:

```
#import SVM model
from sklearn.svm import SVR
```

In [41]:

```
#create a SVM model and fit the training data
SVM_regression=SVR()
SVM_regression.fit(x_train,y_train)
```

Out[41]:

```
SVR()
```

In [42]:

```
print("R-Sqaure:", SVM_regression.score(x_train,y_train))
```

```
R-Sqaure: 0.9348425638424869
```

In [43]:

```
#again check if the above score is r2score
y_svmtrain=SVM_regression.predict(x_train)
print("R_Square:", r2_score(y_svmtrain, y_train))
```

```
R_Square: 0.9313591139847857
```

In [44]:

```
#generate prediction with test data, and print out test r2score
y_svmpred=SVM_regression.predict(x_test)
print("R_Square:", r2_score(y_svmpred, y_test))
```

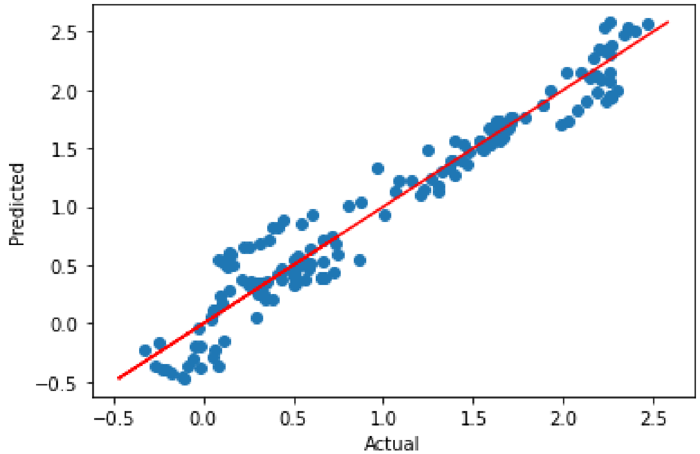
```
R_Square: 0.943040238733662
```

In [45]:

```
#plot prediction vs actual data
plt.scatter(y_svmpred, y_test)
plt.plot(y_test, y_test, "red")
```

```
plt.xlabel("Actual")
plt.ylabel("Predicted")
```

Out[45]: Text(0, 0.5, 'Predicted')




In [46]:
```
#import neural network MLPRegressor
from sklearn.neural_network import MLPClassifier, MLPRegressor
```

In [47]:
```
#Create a NN
NN=MLPRegressor(hidden_layer_sizes=(150,),max_iter=1000, activation="relu",random_state=12)
```

In [48]:
```
#train NN model with training data
NN.fit(x_train,y_train)
```

Out[48]: MLPRegressor(hidden_layer_sizes=(150,), max_iter=1000, random_state=12)

In [55]:
```
#print out training score
NN.score(x_train,y_train)
```

Out[55]: 0.9185387723757129

In [49]:
```
#generate predictions using test data
y_nnpred=NN.predict(x_test)
```

In [50]:
```
#print out r2score for test
```

```
r2_score(y_nnpred, y_test)
```

Out[50]: 0.9172634629796868

In [ ]: