# Programming For Business Intelligence Final Project

New York University

MG-GY 8401 (MOT)

Group 6

Parnika Bhosale, Tinghao Ruan, Huiyi Teng, Yanbin Zhang, Yichun Zhou, Zixin Zhou

# Volatility and Fractal Dimension

Volatility

- It measures the risk performance of assets such as stocks and currency over a period
- It represents the extent to which prices change in relation to the average
- It measures for the selected currencies provide an ideal system for assessing the risk levels and expected returns

Fractal Dimension

- The fractal dimension analysis for the selected currency pairs verifies the fractal market hypothesis assumptions
- The Fractal Dimension analysis tests the presence of fractal properties in the currency time series data

# Volatility and Fractal Dimension Computation

- Volatility
  - **Standard deviation divided** by the **mean**

```
price_list = price_data[i:i+100]
np.std(price_list)/np.mean(price_list)
```

- Fractal Dimension
  - Python **Hurst** Package, **Compute_Hc** function

```
price_list = price_data[i:i+100]
compute_Hc(price_list,kind='price',simplified=True)
```

# Predictive Model

$$Y = \theta_1 * X + \theta_0$$

- **Regularization regression**

  **L1:  LASSO**

  **L2: Ridge :**

# Logic

- Consider the Equation: $Y = aX + b$
- $Y$ = Volatility, $X$ = Fractal Dimension, a & b = coefficients
- After writing it in the intercept form:

$$Y = -(a/b)\ X + c$$

Now, we have to find out the values of:

1. Constant c
2. Coefficients a & b

# Assumption

- H = 2 - D
- Here H = Hurst Ratio and D can be treated as FD i.e. Fractal Dimension
- 0 < FD < 1
- 0< H + V < 2

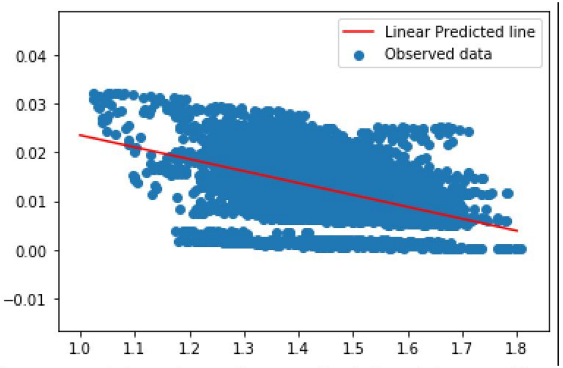
Therefore, average = 1

- Hence, we assume the constant ‘c’ to be 1
- We get: Y = -(a/b)X + 1/b
- Rewriting it as: 1 = aV + bFD
- Task is to find the values of a & b

# Data,Graphs And Accuracy

-----------------Linear Regression ----------------

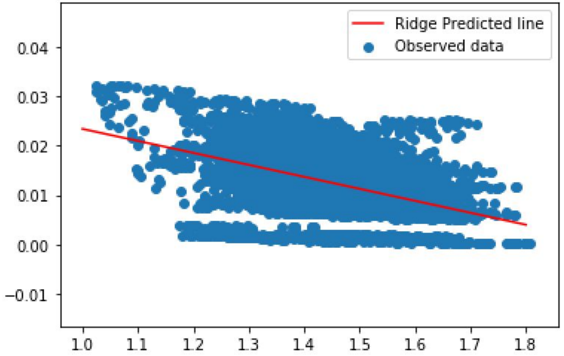
Linear Coef:  -0.024525304962776 35
Linear Intercept:  0.04803921559960726
Linear Regression R_square:  0.2283191674677999
Linear Regression MSE:  3.1377218657445904e-05




Conservation Law Generated by Linear Regression Model
0.5105267572890355 FD +  20.816326568166847 V = 1

-----------------Ridge Regression ----------------

Ridge Coef:  -0.02427044459967178
Ridge Intercept:  0.04766576792254487
Ridge Regression R_square:  0.2282945172253112
Ridge Regression MSE:  3.137822118164742e-05




Conservation Law Generated by Ridge Regression Model
0.5052215007413693 FD +  20.979416541132064 V = 1

# Data,Graphs And Accuracy

```
-----------------Lasso Regression ----------------


Lasso Coef:  -0.0
Lasso Intercept:  0.012102210503849496
Lasso Regression R_square:  0.0
Lasso Regression MSE:  4.066087601850162e-05
```

# Data,Graphs And Accuracy

After comparing the R-Squared and MSE from each model, we found out Linear Regression has the highest Accuracy.

# Accuracy:
# Different Ways to Calculate Hurst Exponent

```python
############################ hurst1 ############################
def FD(price_data):
    fd_list = []
    # hurst_list = []
    for i in range(len(price_data)):
        if len(price_data)-i >=100:
            eurusd_price_2 = price_data[i:i+100]

            lag1, lag2 = 2, 20
            lags = range(lag1, lag2)
            tau = [sqrt(std(subtract(eurusd_price_2[lag:], eurusd_price_2[:-
            lag]))) for lag in lags]
            m = polyfit(log(lags), log(tau), 1)
            if m[0] < 0 and m[1]<0:
                m[0] = 0
            elif m[0] < 0  and m[1]>0:
                m[0] = m[1]
            hurst = m[0] * 2
            fractal_d = 2 - hurst[0]
            fd_list.append(fractal_d)
    return fd_list
```

- This method takes about 5 minutes for the system to get all the FD.
- By using Hurst 1, the R-Squared for both Linear Regression and Ridge Regression are 5.3276% and 5.3270% respectively. This way decreased accuracy for both models.

# Accuracy:
# Different Ways to Calculate Hurst Exponent

```
############################ hurst2 ############################
def calcHurst2(ts):
    if not isinstance(ts, Iterable):
        print ('error')
    return
    n_min, n_max = 2, len(ts)//3
    RSlist = []
    for cut in range(n_min, n_max):
        children = len(ts) // cut
        children_list = [ts[i*children:(i+1)*children] for i in range(cut)]
        L = []
        for a_children in children_list:
            Ma = np.mean(a_children)
            Xta = Series(map(lambda x: x-Ma, a_children)).cumsum()
            Ra = max(Xta) - min(Xta)
            Sa = np.std(a_children)
            rs = Ra / Sa
            L.append(rs)
        RS = np.mean(L)
        RSlist.append(RS)
    return 2 - np.polyfit(np.log(range(2+len(RSlist),2,-1)), np.log(RSlist), 1)[0]

def FD_Other(price_data):
     fd_list_new = []
     # hurst_list = []
     for i in range(len(price_data)):
         if len(price_data)-i >=100:
             eurusd_price_2 = price_data[i:i+100]
             fd_list_new.append(calcHurst2(eurusd_price_2))
     return fd_list_new
FD_new = FD_Other(eurusd_price_serie)
plt.scatter(FD_new,eurusd_volatilitty)
```

- After seeing Hurst 1 was quite slow, and accuracy is also low. We tried to use Hurst 2 to compute FD.
- After running this method, we found out the speed is even slower, and accuracy remains low as well.

# Accuracy:
# Different Ways to Calculate Hurst Exponent

```
############################## hurst3 ##############################
fd_list = []
def FD(price_data):
    for i in range(len(price_data)):
        if len(price_data)-i >=100:
            eurusd_price_2 = price_data[i:i+100]
            hurst = compute_Hc(eurusd_price_2,kind='price',simplified=True)
            FD = 2 - hurst[0]
            fd_list.append(FD)
    return fd_list
```

- After failing the previous 2 methods, we tried hurst 3.
- Fortunately, hurst 3 increased the speed for getting all FDs.
- The accuracy is improved as well.
    - R-Square for linear regression model is 22.832% vs 5.3276%
    - R-Square for ridge regression model is 22.830% vs 5.3270%
    - R-Square for lasso Regression model is 0%

# Conclusion

## Best Model

Linear Regression Model

R-square: 0.22832

MSE: 3.1377

## Most Efficient Exponent for FD

Hurt 3 Exponent

***Thank you!***