上海立信会计金融学院
SHANGHAI LIXIN UNIVERSITY OF ACCOUNTING AND FINANCE

# 《Python金融数据分析》


Hong Cheng（程宏）

School of Statistics and Mathematics

Shanghai LiXin University of Accounting and Finance


May 2022

Testing the significance of RM on MEDV

Estimating the impact of RM on MEDV

Checking the performance of our model

Are they valid claims?

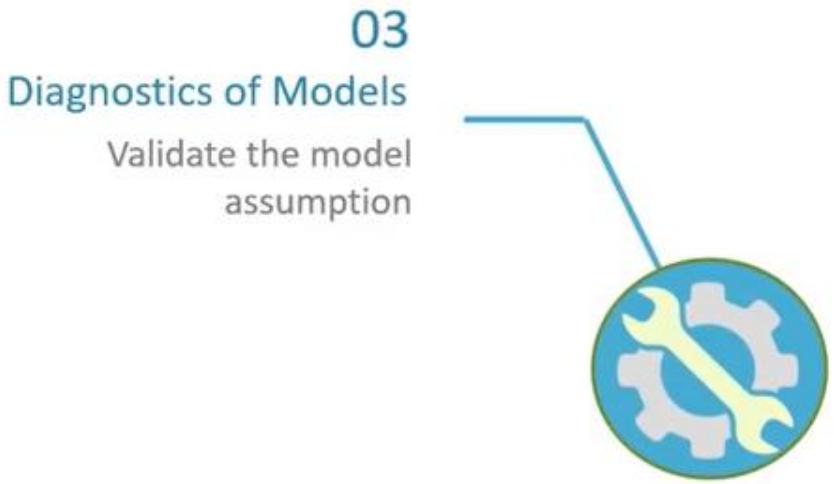
OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | MEDV | **R-squared:** | 0.484 |
| **Model:** | OLS | **Adj. R-squared:** | 0.483 |
| **Method:** | Least Squares | **F-statistic:** | 471.8 |
| **Date:** | Sun, 19 Aug 2018 | **Prob (F-statistic):** | 2.49e-74 |
| **Time:** | 20:05:30 | **Log-Likelihood:** | -1673.1 |
| **No. Observations:** | 506 | **AIC:** | 3350. |
| **Df Residuals:** | 504 | **BIC:** | 3359. |
| **Df Model:** | 1 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | -34.6706 | 2.650 | -13.084 | 0.000 | -39.877 | -29.465 |
| **RM** | 9.1021 | 0.419 | 21.722 | 0.000 | 8.279 | 9.925 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 102.585 | **Durbin-Watson:** | 0.684 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 612.449 |
| **Skew:** | 0.726 | **Prob(JB):** | 1.02e-133 |
| **Kurtosis:** | 8.190 | **Cond. No.** | 58.4 |

**In other words, do those assumptions which are necessary for inference hold true?**




**In this lecture, we will discuss how to diagnose models and validate assumptions of models.**

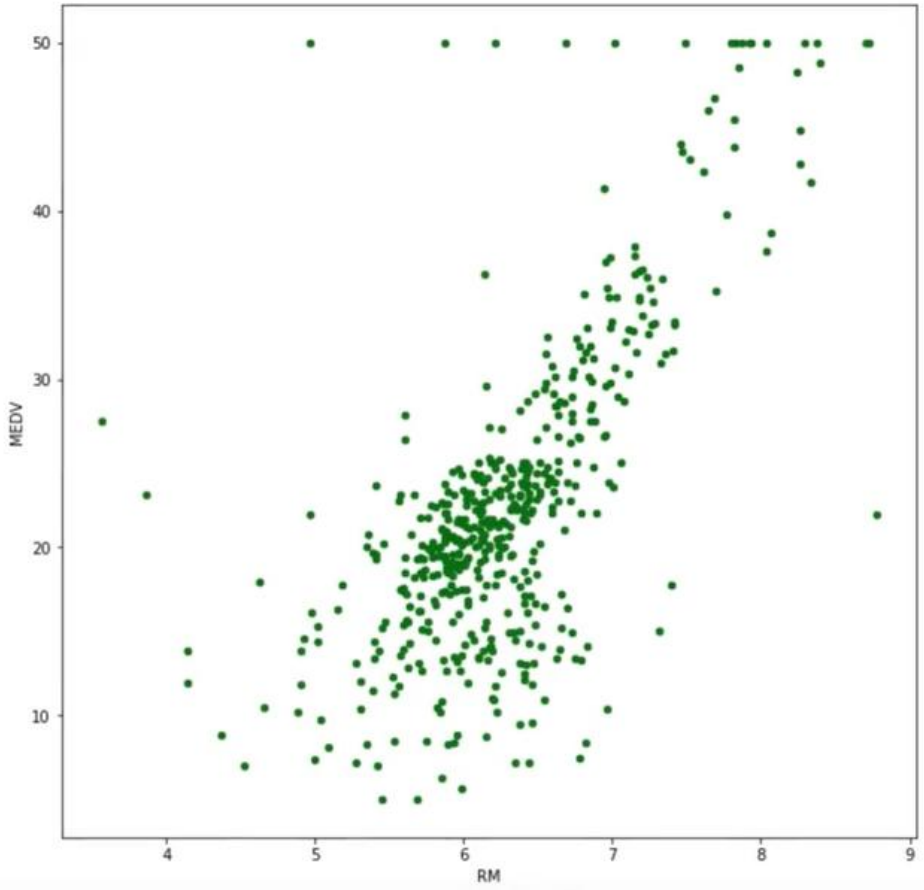
To validate four assumptions upon a population, we only need to **demonstrate that** our sample data is not against these assumptions.

Assumptions of Linear Regression Model

1.Linearity
2.Independence
3.Normality
4.Equal variance

Validation of **linearity** is straightforward, so a scatter plot of Y and X. In our case, **scatter plot** between medium price and number of rooms had linear pattern.

# Linearity



For **independence**, we just need to demonstrate the **observed error is independent mutually**. Alternatively speaking, there is no serial correlation in errors.

**First**, we can plot residual plot, which is the plot observed error.

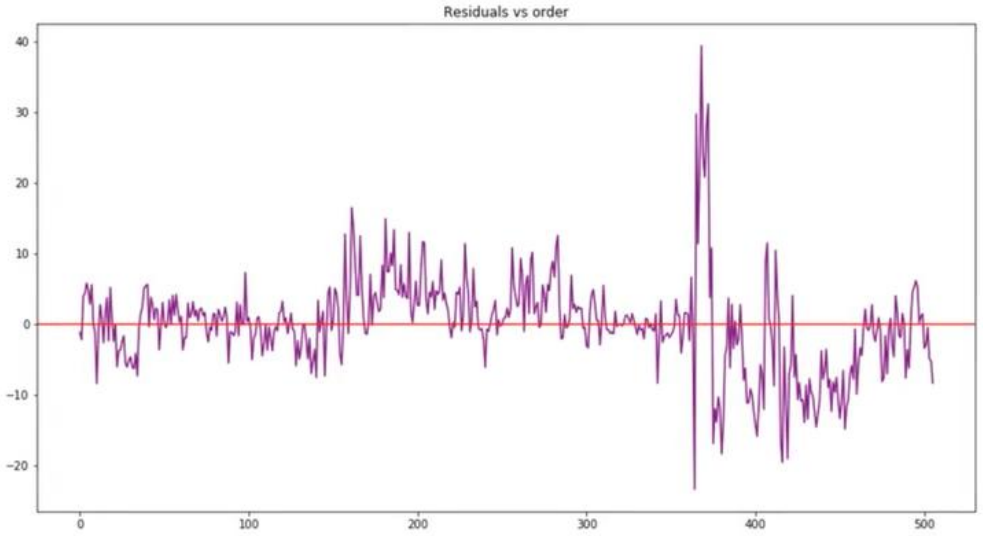
## Independence

In [1]:
```
housing['error'] = housing['MEDV'] - housing['BestResponse']
plt.figure(figsize=(15, 8))
plt.title('Residuals vs order')
plt.plot(housing.index, housing['error'], color='purple')
plt.axhline(y=0, color='red')
plt.show()
```

There's no obvious pattern in this plot. **For quantitative validation of independence**, we need a **Durbin-Watson test**.

## Independence between samples

$\epsilon_1\ \epsilon_2\ \epsilon_3\ \epsilon_4\ \epsilon_5\ \epsilon_6\ \cdots$ Independence of consecutive errors

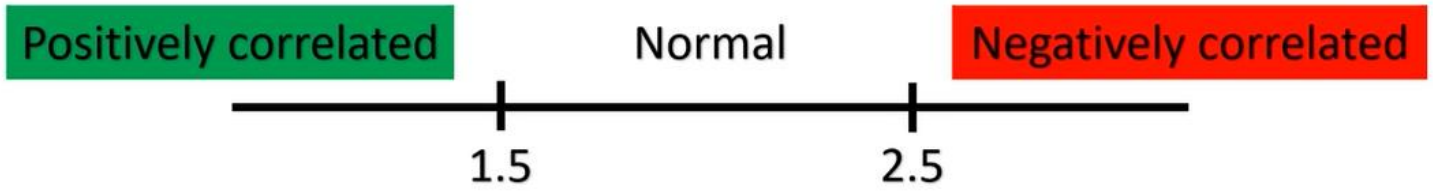


The Durbin Watson is that, there's no serial correlation in errors.

# Durbin Watson test for serial correlation

$$d = \frac{\sum_{t=2}^{T}(e_t - e_{t-1})^2}{\sum_{t=1}^{T} e_t^2}$$

| | | | |
|---|---|---|---|
| Omnibus: | 102.585 | Durbin-Watson: | 0.684 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 612.449 |
| Skew: | 0.726 | Prob(JB): | 1.02e-133 |
| Kurtosis: | 8.190 | Cond. No. | 58.4 |




**Hence an assumption of independence is violated.**

For **normality** validation, we can use **quantile - quantile plot** or **QQ plot**.

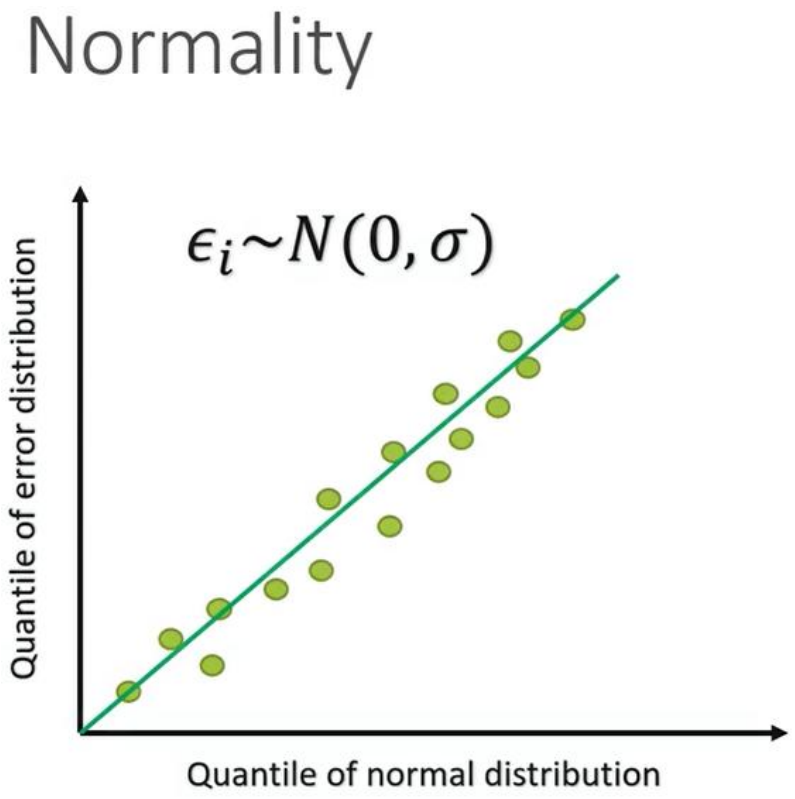
We use **stats.probplot** from **python package scipy.stats** to draw QQ plot.

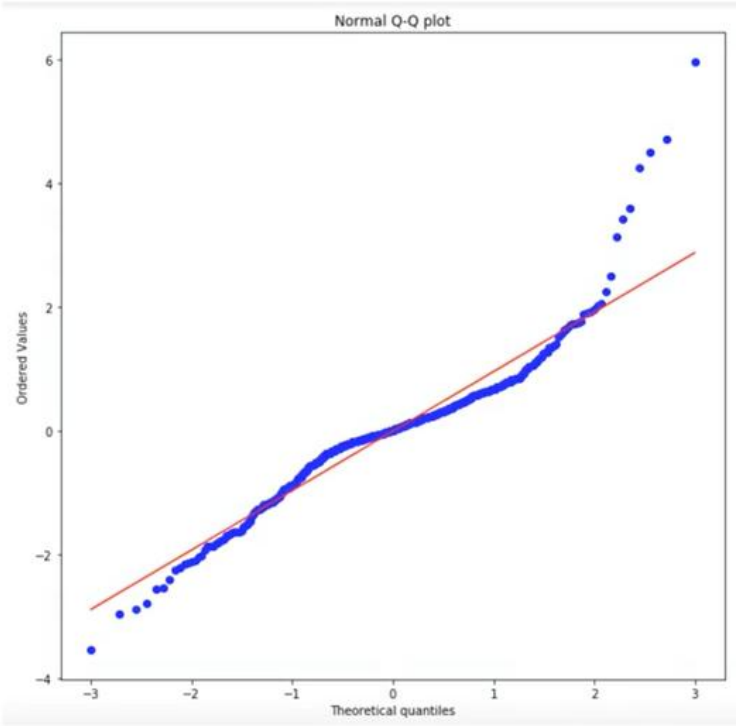
## Normality

In [3]:
```
import scipy.stats as stats
import matplotlib.pyplot as plt
z = (housing['error'] - housing['error'].mean())/housing['error'].std(ddof=1)

stats.probplot(z, dist='norm', plot=plt)
plt.title('Normal Q-Q plot')
plt.show()
```

**The first input of probplot** is **a standardized error. The second term** dist = 'norm' is to compare your distribution of standardized error with normal distribution.

If errors follow normal distribution, they will fall on the 45 degree line roughly. In our model, it deviates a bit in the right tail, but overall it satisfies the normality assumption.

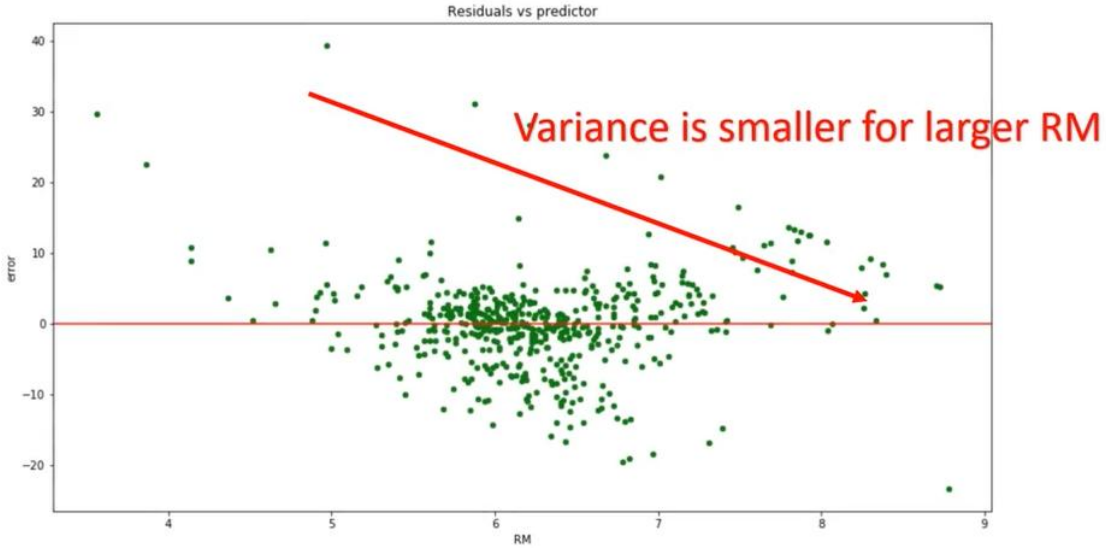


For **equal variance**, we can plot **observed error** versus **predictor**. If variance of noise is equal for different variance predictor, it should not have pattern.

# Equal variance

```
In [3] housing.plot(kind='scatter', x='RM', y='error', figsize=(15, 8), color='green')
       plt.title('Residuals vs predictor')
       plt.axhline(y=0, color='red')
       plt.show()
```




In our model, **noise variance is smaller for houses with big numbers of rooms.** Hence, assumption of equal variance is also violated.

# What have we done so far…

Cannot make statistical inference if assumptions are violated, but …

The accuracy and consistency of model does not rely on these four assumptions


**Return** on investment

In the next two lecture, we will **use multiple linear regression models**, to build a prediction model for price change of SPY, which is the ETF of S&P500, **using multiple global index as the predictors.** Finally, we will apply our model in paper trading and evaluate the performance of our models.

# Lab1： Diagnostic of linear regression model

**Instructions**

- Let's evaluate whether we can make statistical inference using the model between the two linearly related variables we identified.

# Diagnostic of models

In [1]:
```
import pandas as pd
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
% matplotlib inline
```

In [2]:
```
housing = pd.DataFrame.from_csv('../data/housing.csv')
housing.head()
```

Out[2]:

| | LSTAT | INDUS | NOX | RM | MEDV |
|---|---|---|---|---|---|
| **0** | 4.98 | 2.31 | 0.538 | 6.575 | 24.0 |
| **1** | 9.14 | 7.07 | 0.469 | 6.421 | 21.6 |
| **2** | 4.03 | 7.07 | 0.469 | 7.185 | 34.7 |
| **3** | 2.94 | 2.18 | 0.458 | 6.998 | 33.4 |
| **4** | 5.33 | 2.18 | 0.458 | 7.147 | 36.2 |

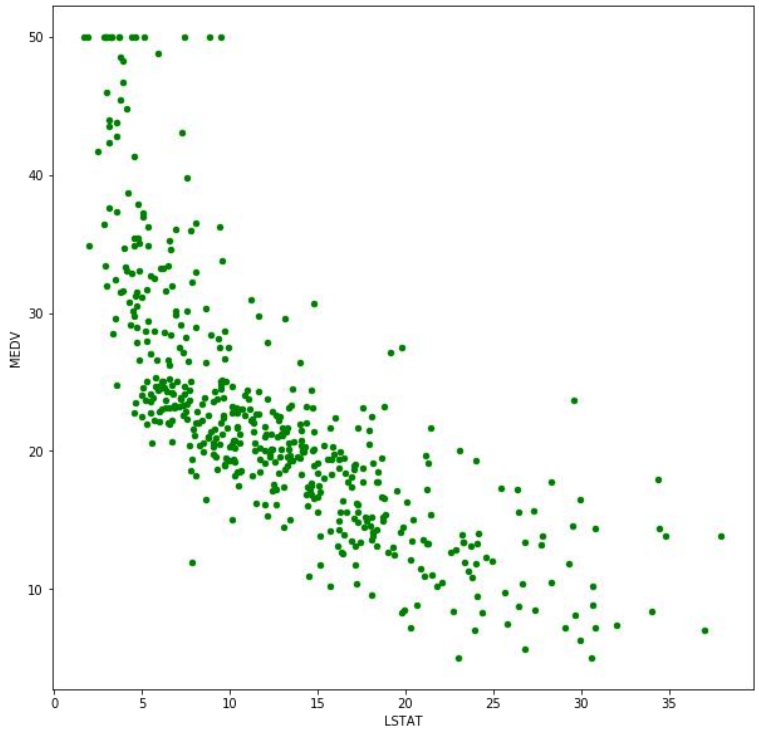
In [3]:
```
model = smf.ols(formula='MEDV~LSTAT', data=housing).fit()

# Here are estimated intercept and slope by least square estimation
b0_ols = model.params[0]
b1_ols = model.params[1]

housing['BestResponse'] = b0_ols + b1_ols*housing['LSTAT']
```

# Assumptions behind linear regression model

1. Linearity
2. independence
3. Normality
4. Equal Variance

## Linearity

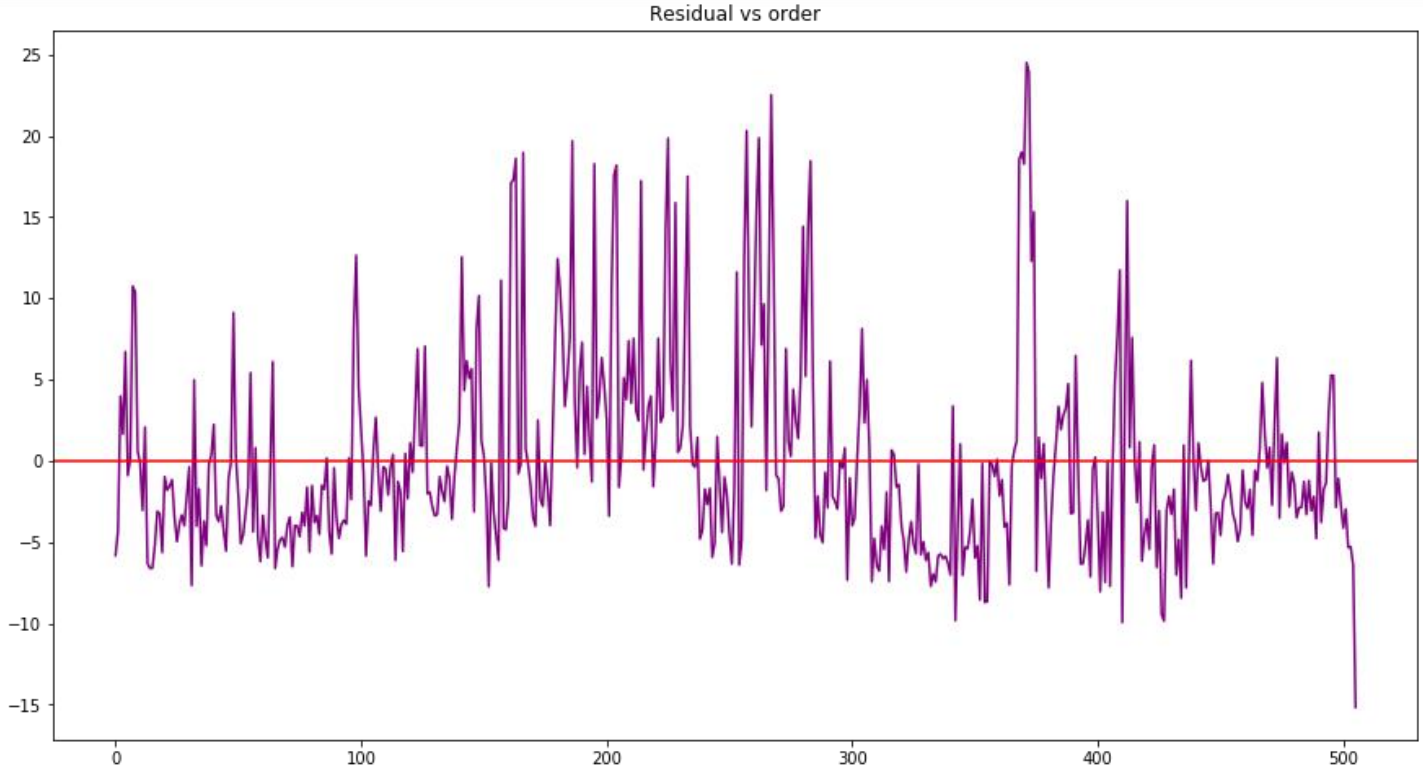
In [4]:
```
# you can check the scatter plot to have a fast check
housing.plot(kind='scatter', x='LSTAT', y='MEDV', figsize=(10, 10), color='g')
```



# Independence

In [5]:
```
# Get all errors (residuals)
housing['error'] = housing['MEDV'] - housing['BestResponse']
```

In [6]:
```
# Method 1: Residual vs order plot
# error vs order plot (Residual vs order) as a fast check
plt.figure(figsize=(15, 8))
plt.title('Residual vs order')
plt.plot(housing.index, housing['error'], color='purple')
plt.axhline(y=0, color='red')
plt.show()
```



In [7]:
```
# Method 2: Durbin Watson Test
# Check the Durbin Watson Statistic
# Rule of thumb: test statistic value in the range of 1.5 to 2.5 are relatively normal
model.summary()
```

Out[7]:

OLS Regression Results

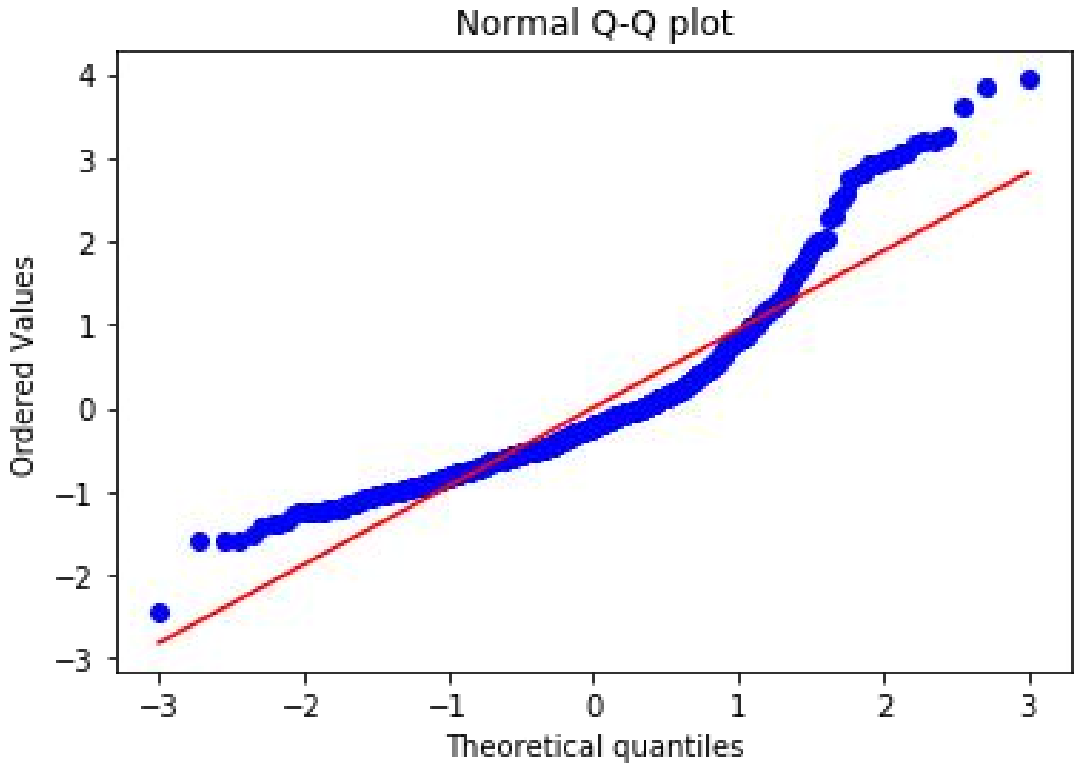
| | | | |
|---|---|---|---|
| **Dep. Variable:** | MEDV | **R-squared:** | 0.544 |
| **Model:** | OLS | **Adj. R-squared:** | 0.543 |
| **Method:** | Least Squares | **F-statistic:** | 601.6 |
| **Date:** | Sun, 15 Sep 2019 | **Prob (F-statistic):** | 5.08e-88 |
| **Time:** | 07:26:39 | **Log-Likelihood:** | -1641.5 |
| **No. Observations:** | 506 | **AIC:** | 3287. |
| **Df Residuals:** | 504 | **BIC:** | 3295. |
| **Df Model:** | 1 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 34.5538 | 0.563 | 61.415 | 0.000 | 33.448 | 35.659 |
| **LSTAT** | -0.9500 | 0.039 | -24.528 | 0.000 | -1.026 | -0.874 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 137.043 | **Durbin-Watson:** | 0.892 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 291.373 |
| **Skew:** | 1.453 | **Prob(JB):** | 5.36e-64 |
| **Kurtosis:** | 5.319 | **Cond. No.** | 29.7 |

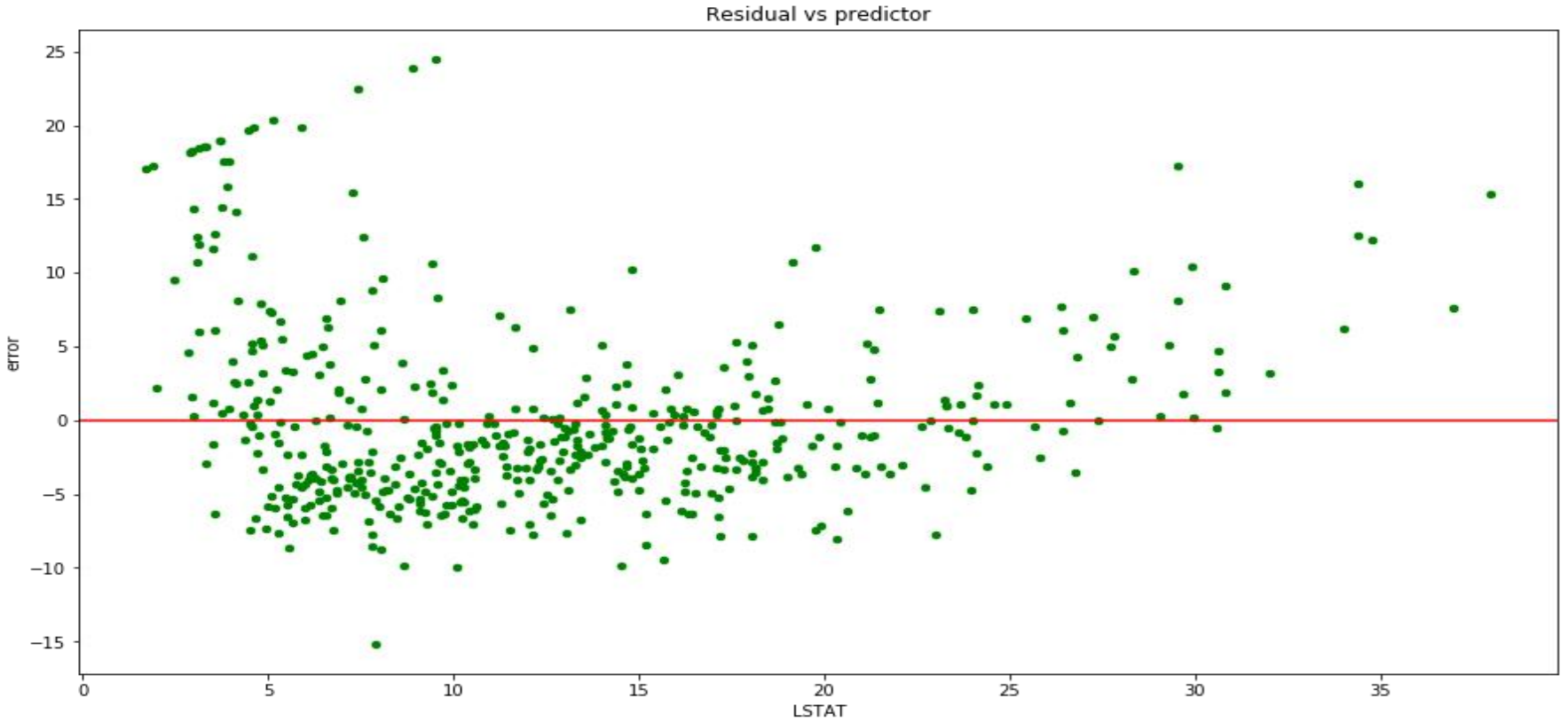
# Normality

In [8]:
```
import scipy.stats as stats
z = (housing['error'] - housing['error'].mean())/housing['error'].std(ddof=1)

stats.probplot(z, dist='norm', plot=plt)
plt.title('Normal Q-Q plot')
plt.show()
```



# Equal variance

In [9]:
```
# Residual vs predictor plot
housing.plot(kind='scatter', x='LSTAT', y='error', figsize=(15, 8), color='green')
plt.title('Residual vs predictor')
plt.axhline(y=0, color='red')
plt.show()
```




**We can see that the regression model (MEDV~LSTAT) violates all four assumptions. Therefore, we cannot make statistical inference using this model.**

# Diagnostic of linear regression model

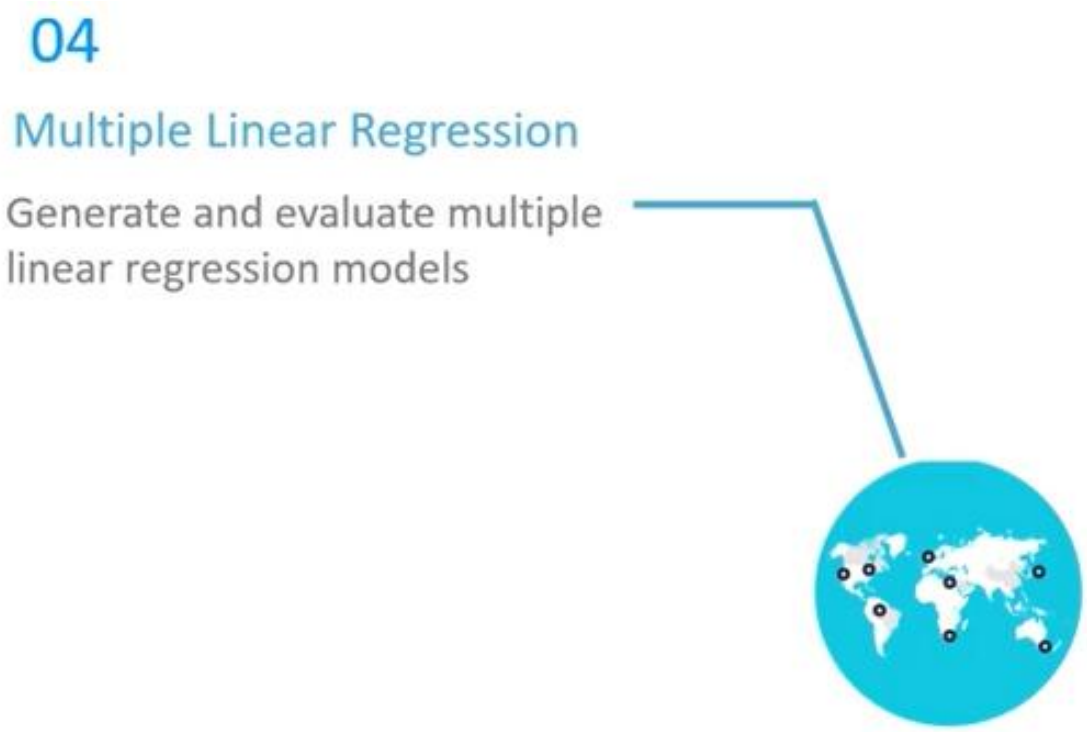
## .ipynb在Github中下载

https://github.com/cloudy-sfu/QUN-Data-Analysis-in-Finance/tree/main/Labs

## Jupyternote Book课堂练习

## 二十分钟

As we can see in previous model, it is a more natural to include more than one predictor in regression models. **The growth of response is determined by multifactors**. In this lecture, we will **apply Multiple Linear Regression model to generate a signal for the growth of SPY.** The exchange-traded fund, which tracks S&P 500. We finally come to the most interesting part of this course. **We will view the model using multiple indices from the global markets and predict the price change of SPY.**



The reason to choose SPY as a target to view the regression model is because it is very suitable for trading frequently.

# SPY

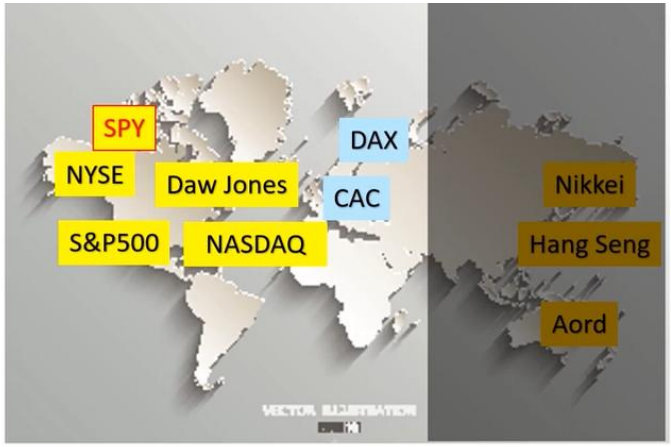
**SPY is an exchange-traded fund which tracks S&P500**

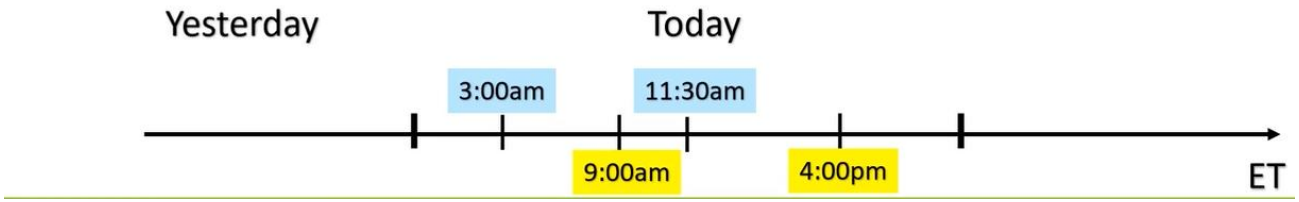
1. **Cheap:** The value of SPY is worth approximately 1/10 of S&P 500's index level
2. **Low fees:** 0.04%
3. **Volatility:** Frequently log double digits loss and gains


S&P500：标准普尔500指数

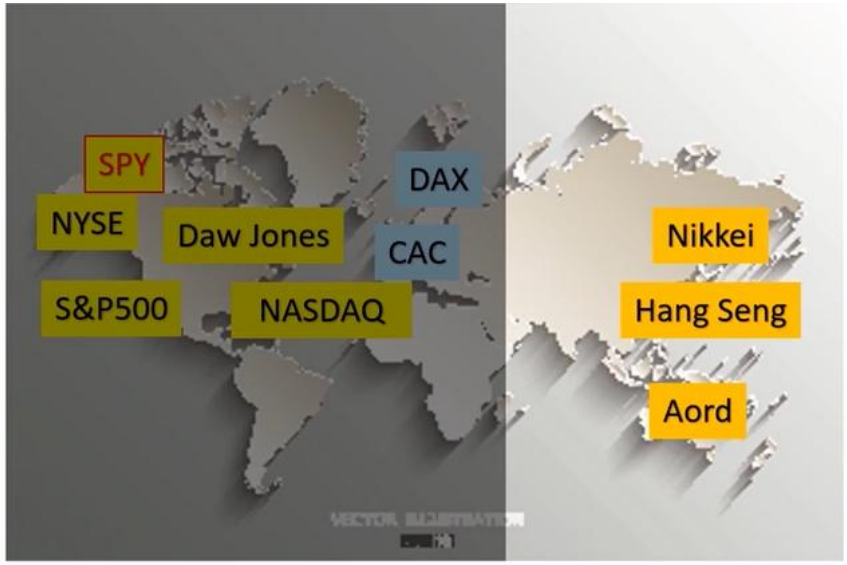
We will predict data price change of SPY when US market opens in the morning. **We know that different indices in different markets are highly correlated**. Using indices for other markets is good for prediction model of SPY.

US and EU Markets

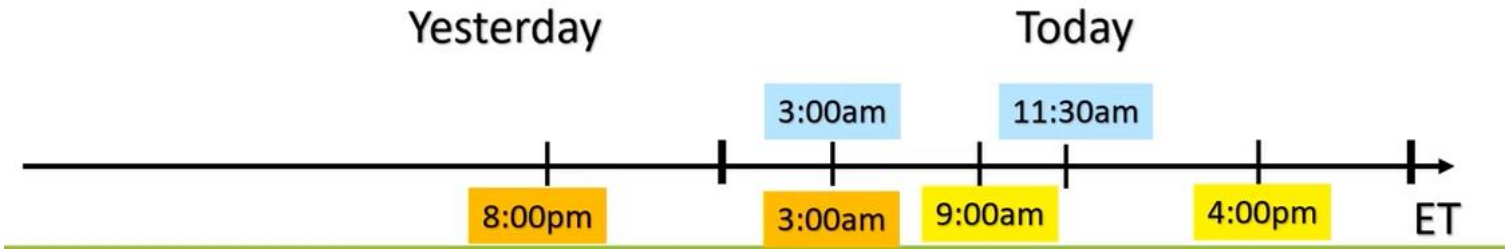



US market opens at 9:00 a.m and close at 4:00 p.m Eastern time today.
Europe markets open at 3:00 a.m and close at 11:30 a.m Eastern time on the same day.
**That is, when US market opens, update data of Europe market is available.**

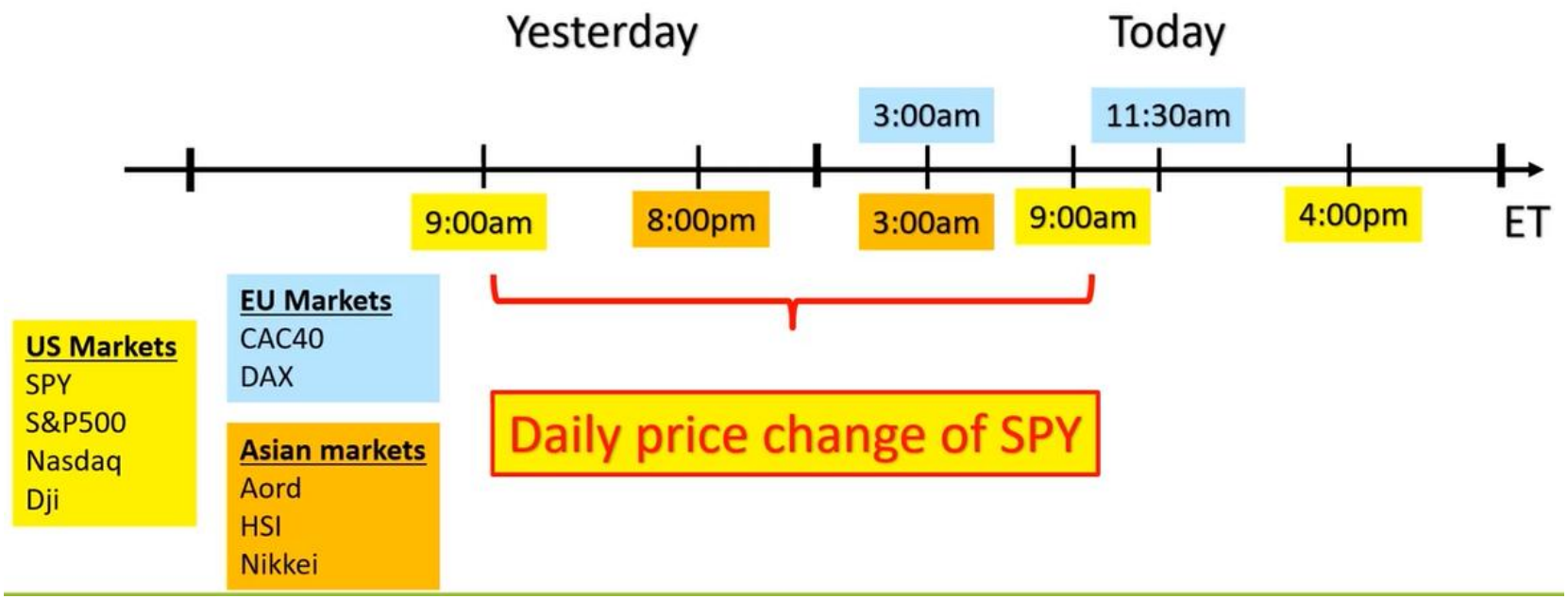



Asian Markets




Asian markets including Australia open at around 8:00 p.m Eastern time yesterday and close at 3:00 a.m Eastern time today. **That means the Asian market information is available for US market at its opening.**

In this lecture, we will use aud ordinaries from Australia, Nikkei from Japan, HSI from HK, DAX from Germany, CAC40 from France, S&P 500, Dji, and Nasdaq from US market, to predict the daily price change of SPY.



You use data frame from CSV to read assessing files of all indices and SPY.

# Data Loading

In [1]:
```
aord=pd.DataFrame.from_csv("data/ALLOrdinary.csv") # All ordinary

#Nikkei

nikkei=pd.DataFrame.from_csv("data/Nikkei225.csv").fillna(method='ffill')
hsi=pd.DataFrame.from_csv("data/HSI.csv"). # his


daxi=pd.DataFrame.from_csv("data/DAXI.csv") # dax performance index

cac40=pd.DataFrame.from_csv("data/CAC40.csv")  #cac40 index


sp500=pd.DataFrame.from_csv("data/SP500.csv") # s&p500

dji=pd.DataFrame.from_csv("data/DJI.csv") #dji:dow jones industry average index

nasdaq=pd.DataFrame.from_csv("data/nasdaq_composite.csv") # Nasdaq

spy=pd.DataFrame.from_csv("data/SPY.csv")
```

All data sets have six columns.

# Columns of Data

```
In [2]  spy.head()
```

Out [2]

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2008-02-01 | 146.529999 | 146.990005 | 143.880005 | 144.929993 | 116.593864 | 204935600 |
| 2008-03-01 | 144.910004 | 145.490005 | 144.070007 | 144.860001 | 116.537567 | 125133300 |
| 2008-04-01 | 143.339996 | 143.440002 | 140.910004 | 141.309998 | 113.681671 | 232330900 |
| 2008-07-01 | 141.809998 | 142.229996 | 140.100006 | 141.190002 | 113.585121 | 234991000 |
| 2008-08-01 | 142.080002 | 142.899994 | 138.440002 | 138.910004 | 111.750923 | 326365700 |

Different from Simple Linear Regression, **Multiple Linear Regression** will have multiple predictors.

# Multiple linear Regression Model

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + ..$$

**Predictors**

**Response**

US Market

SPY: Open – Open Last day
Sp500: Open – Open Last day
Nasdaq: Open – Open Last day
Dji : Open – Open Last day

SPY : Open Next day – Open

EU Markets

Cac40: Open – Open Last day
Daxi: Open – Open Last day

Asian Markets

Aord: Close –Open
HSI: Close –Open
Nikkei : Close – Open

we will mung the data to get all these predictors and the response.

# Data Munging

In [3]

```
indicepanel=pd.DataFrame(index=spy.index)
indicepanel['spy']=spy['Open'].shift(-1) -spy['Open']
indicepanel['spy_lag1']=indicepanel['spy'].shift(1)
indicepanel['sp500']=sp500["Open"] -sp500['Open'].shift(1)
indicepanel['nasdaq']=nasdaq['Open'] -nasdaq['Open'].shift(1)
indicepanel['dji']=dji['Open']-dji['Open'].shift(1)

indicepanel['cac40']=cac40['Open'] -cac40['Open'].shift(1)
indicepanel['daxi']=daxi['Open'] -daxi['Open'].shift(1)

indicepanel['aord']=aord['Close'] -aord['Open']
indicepanel['hsi']=hsi['Close'] -hsi['Open']
indicepanel['nikkei']=nikkei['Close'] -nikkei['Open']


indicepanel['Price']= spy['Open']
```

If we print the head of this table, we find **missing values**. This is due to **two reasons**. When we calculate price change, we may generate NaN value in the first row, one-day lag, and the last row, one-day in the future. In different markets, they may have different holidays in which the markets are closed. It can be shown by computing numbers of NaN values in each column. We find Australia markets seems to have more holidays.

## Data Munging

```
In [4]  indicepanel.head()
```

Out [4]

| Date | spy | spy_lag1 | sp500 | nasdaq | dji | |
|---|---|---|---|---|---|---|
| 2008-02-01 | -1.619995 | NaN | NaN | NaN | NaN | |
| 2008-03-01 | -1.570008 | -1.619995 | -20.419922 | -41.949951 | -217.70019 | - |
| 2008-04-01 | -1.529998 | -1.570008 | -3.540039 | -40.879883 | 2.43945 | |
| 2008-07-01 | 0.270004 | -1.529998 | -29.940064 | -56.930176 | -245.40918 | -1 |
| 2008-08-01 | -2.990006 | 0.270004 | 1.640015 | -7.179931 | 19.75000 | |

We need to handle NaN values first before we view the model.

**First**, we use fill-forward method to fill the holes of data frame by propagating last valid observation forward to next valid.

**Second**, we drop the first row by using dropna.

We can check if there is any NaN remaining by computing numbers NaN values.

From the printout, it is clear that we fill all holes.

Finally, we save our clean data into data file indicepanel.csv using method to_csv of DataFrame.

## Data Munging

```
In [6]  indicepanel=indicepanel.fillna(method='ffill')
        indicepanel=indicepanel.dropna()

In [7]  indicepanel.isnull().sum()

In [8]  indicepanel.to_csv("data/indicepanel.csv")
```

```
Out [7]
spy         0
spy_lag1    0
sp500       0
nasdaq      0
dji         0
cac40       0
daxi        0
aord        0
hsi         0
nikkei      0
Price       0
```

**Totally**, we have 2,677 days of data, one response variable, nine predictors, and the last column keeps a record of open price of SPY, which will be used in paper trading.

```
In [1]    indicepanel.shape

Out [1]   (2677, 11)
```

2677 days of data

1 Response

9 Predictors

1 Open of SPY

**To make sure that our model is consistent in future data**, we need to split data into two parts; **one is for building the model**, **the other part is for testing the model** to see if the model can still make reasonable prediction in this dataset.

# Data splitting

In [2]:
```
Train= indicepanel.iloc[-2000:-1000,:]
Test= indicepanel.iloc[-1000:,:]
print(Train.shape, Test.shape)
```

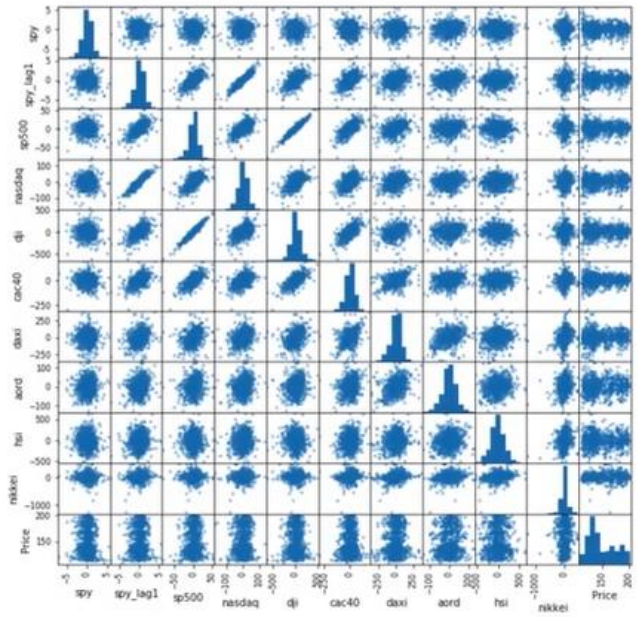
Out [2]:
```
(1000, 11) (1000, 11)
```

**Firstly**, we use a scatter matrix to get a pairwise scatterplot.

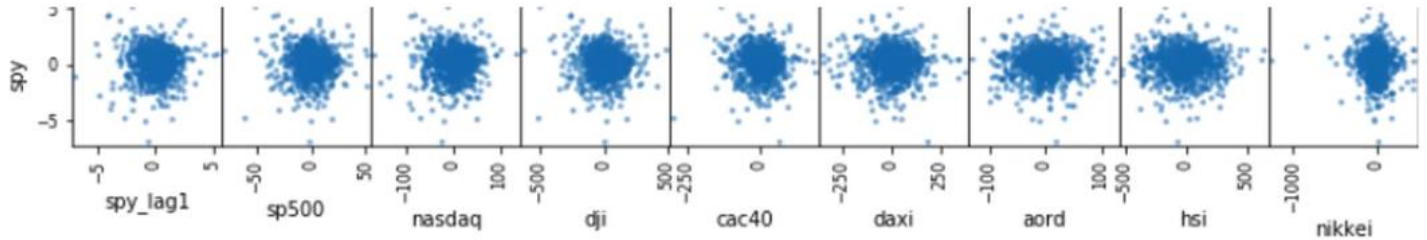
# Exploration on Train data

In [1]
```
from pandas.plotting import scatter_matrix
sm=scatter_matrix(Train,figsize=(10,10))
```

Out [1]



If you check the scatterplots, which will response SPY with other nine predictors. **You may find that**, **there is no explicit pattern, which is evidence of high noisy properties of stock markets**. We need to compute correlation in order to see the association clearly. From the output of correlation, **we find that** the predictors for Europe and Asian markets do have association with SPY, which have higher impacts than predictors of U.S. markets.




```
In [1]    Train.iloc[:,:-1].corr()['spy']
```

```
Out [1]   spy         1.000000
          spy_lag1   -0.011623
          sp500      -0.018632
          nasdaq      0.012333
          dji        -0.037097
          cac40      -0.055304
          daxi       -0.036091
          aord        0.075872
          hsi        -0.031038
          nikkei     -0.043934
          Name: spy, dtype: float64
```

We can use OLS method of Statsmodels to build multiple linear equation model.

# Regression model

In [1]
```
formula = 'spy~spy_lag1+sp500+nasdaq+dji+cac40+daxi+aord+nikkei+hsi'
lm= smf.ols(formula=formula, data=Train).fit()
lm.summary()
```

There are a couple things we need to pay attention; **the first thing is a p value for F-statistics.**

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | spy | R-squared: | 0.021 |
| Model: | OLS | Adj. R-squared: | 0.013 |
| Method: | Least Squares | F-statistic: | 2.407 |
| Date: | Thu, 23 Aug 2018 | Prob (F-statistic): | 0.0106 |
| Time: | 09:23:30 | Log-Likelihood: | -1641.8 |
| No. Observations: | 1000 | AIC: | 3304. |
| Df Residuals: | 990 | BIC: | 3353. |
| Df Model: | 9 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| Intercept | 0.0789 | 0.040 | 1.968 | 0.049 | 0.000 | 0.157 |
| spy_lag1 | -0.1101 | 0.093 | -1.184 | 0.237 | -0.293 | 0.072 |
| sp500 | 0.0183 | 0.014 | 1.304 | 0.193 | -0.009 | 0.046 |
| nasdaq | 0.0044 | 0.004 | 1.157 | 0.248 | -0.003 | 0.012 |
| dji | -0.0020 | 0.001 | -1.341 | 0.180 | -0.005 | 0.001 |
| cac40 | -0.0015 | 0.001 | -1.163 | 0.245 | -0.004 | 0.001 |
| daxi | -0.0005 | 0.001 | -0.906 | 0.365 | -0.002 | 0.001 |
| aord | 0.0040 | 0.001 | 3.250 | 0.001 | 0.002 | 0.006 |
| nikkei | -0.0005 | 0.000 | -1.287 | 0.198 | -0.001 | 0.000 |
| hsi | -0.0002 | 0.000 | -0.824 | 0.410 | -0.001 | 0.000 |

| | | | |
|---|---|---|---|
| Omnibus: | 96.133 | Durbin-Watson: | 1.997 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 270.514 |
| Skew: | -0.493 | Prob(JB): | 1.81e-59 |
| Kurtosis: | 5.350 | Cond. No. | 396. |

Significance of model

**F test is for overall significance of the multiple linear equation model.**

If we **reject**, we accept alternative and **it means that at least one of the predictors is useful**.

Our model is better fitted than intercept only model. **P-value equal to 0.0106** in our model, which is less than 0.05 and **it indicates that, our model includes useful predictors**.

## Significance of the model – F test

$$H_0: \beta_1 = \beta_2 = \beta_3 = \beta_4 = \cdots = 0$$

$$H_a: at\ least\ one\ of\ them\ is\ not\ zero$$

P value = 0.0106 <0.05, Reject $H_0$

Summary table also lists the p value for the test of significance of the individual predictors. From last lecture, we know that p value of test statistic used for this test. For our results, **we find that, most of the predictors are not significant, except the Aord.** That means all other predictors are useless information of SPY. **It may be because of multicollinearity.**

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | spy | **R-squared:** | 0.021 |
| **Model:** | OLS | **Adj. R-squared:** | 0.013 |
| **Method:** | Least Squares | **F-statistic:** | 2.407 |
| **Date:** | Thu, 23 Aug 2018 | **Prob (F-statistic):** | 0.0106 |
| **Time:** | 09:23:30 | **Log-Likelihood:** | -1641.8 |
| **No. Observations:** | 1000 | **AIC:** | 3304. |
| **Df Residuals:** | 990 | **BIC:** | 3353. |
| **Df Model:** | 9 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 0.0789 | 0.040 | 1.968 | 0.049 | 0.000 | 0.157 |
| **spy_lag1** | -0.1101 | 0.093 | -1.184 | 0.237 | -0.293 | 0.072 |
| **sp500** | 0.0183 | 0.014 | 1.304 | 0.193 | -0.009 | 0.046 |
| **nasdaq** | 0.0044 | 0.004 | 1.157 | 0.248 | -0.003 | 0.012 |
| **dji** | -0.0020 | 0.001 | -1.341 | 0.180 | -0.005 | 0.001 |
| **cac40** | -0.0015 | 0.001 | -1.163 | 0.245 | -0.004 | 0.001 |
| **daxi** | -0.0005 | 0.001 | -0.906 | 0.365 | -0.002 | 0.001 |
| **aord** | 0.0040 | 0.001 | 3.250 | 0.001 | 0.002 | 0.006 |
| **nikkei** | -0.0005 | 0.000 | -1.287 | 0.198 | -0.001 | 0.000 |
| **hsi** | -0.0002 | 0.000 | -0.824 | 0.410 | -0.001 | 0.000 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 96.133 | **Durbin-Watson:** | 1.997 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 270.514 |
| **Skew:** | -0.493 | **Prob(JB):** | 1.81e-59 |
| **Kurtosis:** | 5.350 | **Cond. No.** | 396. |

Significance of individual predictors

**Multicollinearity** refers to a situation in which two or more predictors in the multiple regression model are highly, linearly related.

One predictor can be predicted from the others with a substantial degree of accuracy and it is typical for our model since all indices of different markets are correlated.

Multicollinearity does not reduce predictive power. **In this situation, the coefficient estimates of the multiple equation may change erratically in response to small changes of data.**

## Muticollinarity

```
In [1]  Train.iloc[:, :-1].corr()
```

| | spy | spy_lag1 | sp500 | nasdaq | dji | cac40 | daxi | aord | hsi | nikkei |
|---|---|---|---|---|---|---|---|---|---|---|
| spy | 1.000000 | -0.011623 | -0.018632 | 0.012333 | -0.037097 | -0.055304 | -0.036091 | 0.075872 | -0.031038 | -0.043934 |
| spy_lag1 | -0.011623 | 1.000000 | 0.664272 | 0.932118 | 0.575321 | 0.522888 | 0.318190 | 0.220975 | 0.124295 | 0.125412 |
| sp500 | -0.018632 | 0.664272 | 1.000000 | 0.636528 | 0.960032 | 0.693578 | 0.327233 | 0.174477 | -0.011292 | 0.001759 |
| nasdaq | 0.012333 | 0.932118 | 0.636528 | 1.000000 | 0.517313 | 0.476918 | 0.291248 | 0.207667 | 0.099670 | 0.095777 |
| dji | -0.037097 | 0.575321 | 0.960032 | 0.517313 | 1.000000 | 0.689268 | 0.332618 | 0.154434 | -0.043488 | -0.007101 |
| cac40 | -0.055304 | 0.522888 | 0.693578 | 0.476918 | 0.689268 | 1.000000 | 0.489609 | 0.230942 | 0.147568 | 0.113297 |
| daxi | -0.036091 | 0.318190 | 0.327233 | 0.291248 | 0.332618 | 0.489609 | 1.000000 | 0.388659 | 0.165065 | 0.255790 |
| aord | 0.075872 | 0.220975 | 0.174477 | 0.207667 | 0.154434 | 0.230942 | 0.388659 | 1.000000 | 0.164765 | 0.207280 |
| hsi | -0.031038 | 0.124295 | -0.011292 | 0.099670 | -0.043488 | 0.147568 | 0.165065 | 0.164765 | 1.000000 | 0.241013 |
| nikkei | -0.043934 | 0.125412 | 0.001759 | 0.095777 | -0.007101 | 0.113297 | 0.255790 | 0.207280 | 0.241013 | 1.000000 |

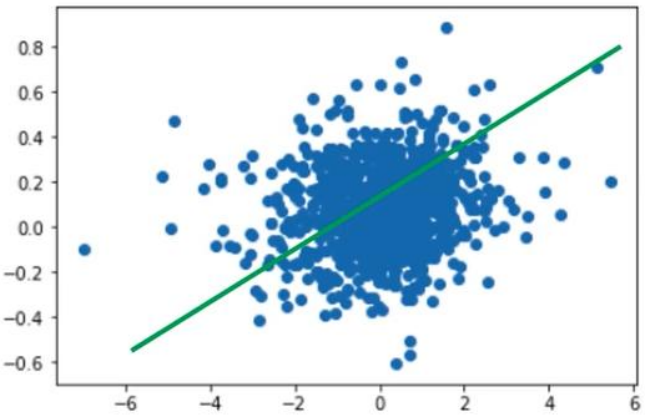
**Now**, we can predict daily change of SPY using message predict of our model LM. We predict SPY in both train and test. We can further ensure scatterplot between real daily change and predict daily change of SPY. It does have positive correlation although not very strong. Considering it is daily change, this result is not very bad.

# Making Prediction

In [1]
```
Train['PredictedY']= lm.predict(Train)
Test['PredictedY']= lm.predict(Test)
```

In [2]
```
plt.scatter(Train['spy'], Train['PredictedY'])
```

Out [2]



Next, we will evaluate our models by comparing two statistics in train and test.

- First statistic is **RMSE**, which is the square root of sum of squared errors averaged by degrees of freedom, where k is number of predictors. This statistic is to measure the prediction error. The reason to use the degrees of freedom is that, square of RMSE is unbiased estimator of variance of the noise.
- The second is **adjusted R-square**,which measures percentage of variation of a response that is explained by the model.

## Model Evaluation

RMSE

$$RMSE = \sqrt{\frac{SSE}{n - k - 1}}$$

Adjusted $R^2$

$$R^2_{adjusted} = 1 - \frac{(1-R^2)(n-1)}{n-k-1}$$

We will compute **Adjusted R-squared** and **RMSE** in many practice. In python, the compacted this course under a function name, so that we use this course repeatedly just by referring to the function next. The keyword **def** is a keyword to define a function and return its keyword to offload your result from the inside of function.

# Model Evaluation

In [3]:
```
def adjustedMetric(data, model,model_k, yname):
    data['yhat']=model.predict(data)
    SST=((data[yname] -data[yname].mean())**2).sum()
    SSR=((data['yhat'] -data[yname].mean())**2).sum()
    SSE=((data[yname]-data['yhat'])**2).sum()
    r2= SSR/SST
    adjustR2=1-(1-r2)*(data.shape[0]-1)/(data.shape[0]-model_k -1)
    RMSE=(SSE/(data.shape[0] -model_k -1))**0.5
    return adjustR2 ,RMSE
```

# Model Evaluation

In [3]
```
print("Adjusted R2 and RMSE om Train: ", adjustedMetric(Train,lm, 9,'spy'))
print("Adjusted R2 and RMSE om Test: ", adjustedMetric(Test,lm, 9,'spy'))
```

Out [3]
```
Adjusted R2 and RMSE om Train: (0.012515746612546508, 1.2559997558890261)
Adjusted R2 and RMSE om Test: (0.015884368214835742, 1.7385594344506785)
```

We give another function access table to generate output, which **compared to Adjusted R-square and RMSE between train and test** in order to check whether they are different dramatically.

# Model Evaluation

In [3]
```
def assessTable(test, train,model, model_k, yname):
    r2test,RMSEtest= adjustedMetric(test, model, model_k, yname)
    r2train,RMSEtrain= adjustedMetric(train, model, model_k, yname)
    assessment= pd.DataFrame(index=['R2','RMSE'],columns=['Train','Test'])
    assessment['Train']=[r2train, RMSEtrain]
    assessment['Test']=[r2test, RMSEtest]
    return assessment
```

In [4]
```
assessTable(Test, Train, lm, 9,'spy')
```

If so, this is called **overfitting**. Usually, for overfitting model, RMSE and adjusted R-square is much better in train than in test dataset.

That it implies that we cannot apply this model to real market in future.

## Overfitting?

If RMSE and adjusted $R^2$ in train and test differ dramatically

Overfitting

| | Train | Test |
|---|---|---|
| R2 | 0.012516 | 0.015884 |
| RMSE | 1.256000 | 1.738559 |

Overall, No Overfitting

# Lab2： Build the trading model by yourself!

**Instructions**

- Let's mimic the process of building our trading model of SPY, base on the historical data of different stock markets

# Multiple linear regression model

## Let's mimic the process of building our trading model of SPY, base on the historical data of different stock markets

In [44]:
```
import pandas as pd
import statsmodels.formula.api as smf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [45]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [46]:
```
# import all stock market data into DataFrame
aord = pd.DataFrame.from_csv('../data/indice/ALLOrdinary.csv')
nikkei = pd.DataFrame.from_csv('../data/indice/Nikkei225.csv')
hsi = pd.DataFrame.from_csv('../data/indice/HSI.csv')
daxi = pd.DataFrame.from_csv('../data/indice/DAXI.csv')
cac40 = pd.DataFrame.from_csv('../data/indice/CAC40.csv')
sp500 = pd.DataFrame.from_csv('../data/indice/SP500.csv')
dji = pd.DataFrame.from_csv('../data/indice/DJI.csv')
nasdaq = pd.DataFrame.from_csv('../data/indice/nasdaq_composite.csv')
spy = pd.DataFrame.from_csv('../data/indice/SPY.csv')
```

In [65]:
```
nasdaq.head()
```

Out[65]:

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2008-01-02 | 2653.909912 | 2661.500000 | 2597.810059 | 2609.629883 | 2609.629883 | 2076690000 |
| 2008-01-03 | 2611.959961 | 2624.270020 | 2592.179932 | 2602.679932 | 2602.679932 | 1970200000 |
| 2008-01-04 | 2571.080078 | 2571.080078 | 2502.679932 | 2504.649902 | 2504.649902 | 2516310000 |
| 2008-01-07 | 2514.149902 | 2521.620117 | 2471.229980 | 2499.459961 | 2499.459961 | 2600100000 |
| 2008-01-08 | 2506.969971 | 2527.419922 | 2440.510010 | 2440.510010 | 2440.510010 | 2566480000 |

## Step 1: Data Munging

In [49]:
```
# Due to the timezone issues, we extract and calculate appropriate stock market data for analysis
# Indicepanel is the DataFrame of our trading model
indicepanel=pd.DataFrame(index=spy.index)

indicepanel['spy']=spy['Open'].shift(-1)-spy['Open']
indicepanel['spy_lag1']=indicepanel['spy'].shift(1)
indicepanel['sp500']=sp500["Open"]-sp500['Open'].shift(1)
indicepanel['nasdaq']=nasdaq['Open']-nasdaq['Open'].shift(1)
indicepanel['dji']=dji['Open']-dji['Open'].shift(1)

indicepanel['cac40']=cac40['Open']-cac40['Open'].shift(1)
indicepanel['daxi']=daxi['Open']-daxi['Open'].shift(1)

indicepanel['aord']=aord['Close']-aord['Open']
indicepanel['hsi']=hsi['Close']-hsi['Open']
indicepanel['nikkei']=nikkei['Close']-nikkei['Open']
indicepanel['Price']=spy['Open']
```

In [50]:
```
indicepanel.head()
```

Out[50]:

| Date | spy | spy_lag1 | sp500 | nasdaq | dji | cac40 | daxi | aord | hsi | nikkei | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2008-01-02 | -1.619995 | NaN | NaN | NaN | NaN | NaN | NaN | -50.100097 | -71.679688 | NaN | 146.529999 |
| 2008-01-03 | -1.570008 | -1.619995 | -20.419922 | -41.949951 | -217.70019 | -71.779785 | -104.450195 | -2.300293 | -162.750000 | NaN | 144.910004 |
| 2008-01-04 | -1.529998 | -1.570008 | -3.540039 | -40.879883 | 2.43945 | 5.489746 | -27.990235 | NaN | 515.349609 | -464.320313 | 143.339996 |
| 2008-01-07 | 0.270004 | -1.529998 | -29.940064 | -56.930176 | -245.40918 | -111.689941 | -102.709961 | -27.500000 | 216.951171 | -48.830078 | 141.809998 |
| 2008-01-08 | -2.990006 | 0.270004 | 1.640015 | -7.179931 | 19.75000 | 44.509766 | 33.680176 | -33.899902 | -354.060547 | 99.370117 | 142.080002 |

In [51]:
```
# Lets check whether do we have NaN values in indicepanel
indicepanel.isnull().sum()
```

Out[51]:
```
spy          1
spy_lag1     1
sp500        1
nasdaq       1
dji          1
cac40       30
daxi        53
aord       319
hsi        121
nikkei     145
Price        0
dtype: int64
```

In [52]:
```
# We can use method 'fillna()' from dataframe to forward filling the Nan values
# Then we can drop the reminding Nan values
indicepanel = indicepanel.fillna(method='ffill')
indicepanel = indicepanel.dropna()
```

In [53]:
```
# Lets check whether do we have Nan values in indicepanel now
indicepanel.isnull().sum()
```

Out[53]:
```
spy         0
spy_lag1    0
sp500       0
nasdaq      0
dji         0
cac40       0
daxi        0
aord        0
hsi         0
nikkei      0
Price       0
dtype: int64
```

In [54]:
```
# save this indicepanel for part 4.5
path_save = '../data/indice/indicepanel.csv'
indicepanel.to_csv(path_save)
```

In [55]:
```
print(indicepanel.shape)
```

```
(2677, 11)
```

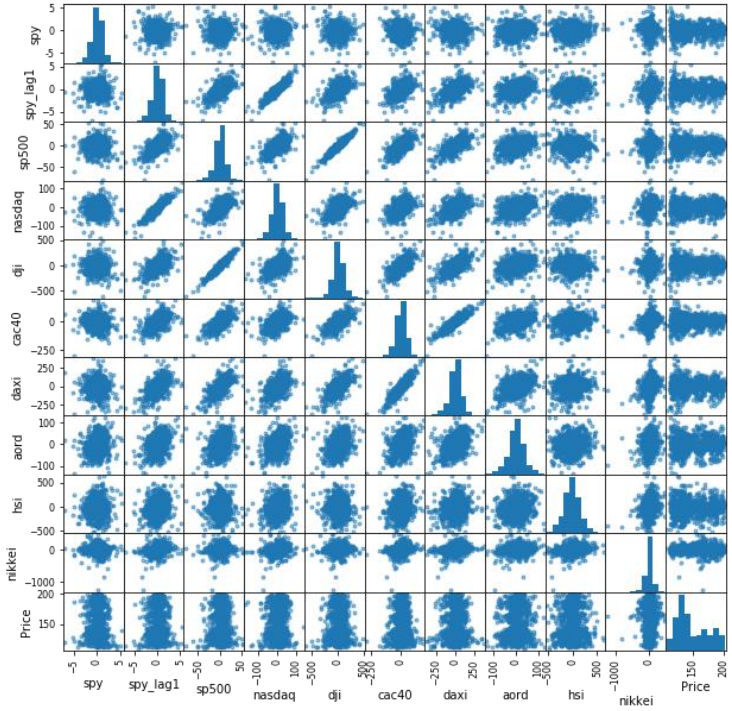
## Step 2: Data Spliting

In [56]:
```
#split the data into (1)train set and (2)test set

Train = indicepanel.iloc[-2000:-1000, :]
Test = indicepanel.iloc[-1000:, :]
print(Train.shape, Test.shape)
```

```
(1000, 11) (1000, 11)
```

## Step 3: Explore the train data set

In [57]:
```
# Generate scatter matrix among all stock markets (and the price of SPY) to observe the association

from pandas.tools.plotting import scatter_matrix
sm = scatter_matrix(Train, figsize=(10, 10))
```



## Step 4: Check the correlation of each index between spy

In [58]:
```
# Find the indice with largest correlation
corr_array = Train.iloc[:, :-1].corr()['spy']
print(corr_array)
```

```
spy          1.000000
spy_lag1    -0.011623
sp500       -0.018632
nasdaq       0.012333
dji         -0.037097
cac40       -0.055304
daxi        -0.069735
aord         0.179638
hsi          0.031400
nikkei      -0.035048
Name: spy, dtype: float64
```

In [59]:
```
formula = 'spy~spy_lag1+sp500+nasdaq+dji+cac40+aord+daxi+nikkei+hsi'
lm = smf.ols(formula=formula, data=Train).fit()
lm.summary()
```

Out[59]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | spy | **R-squared:** | 0.067 |
| **Model:** | OLS | **Adj. R-squared:** | 0.059 |
| **Method:** | Least Squares | **F-statistic:** | 7.962 |
| **Date:** | Sun, 27 Jan 2019 | **Prob (F-statistic):** | 1.97e-11 |
| **Time:** | 03:43:13 | **Log-Likelihood:** | -1617.7 |
| **No. Observations:** | 1000 | **AIC:** | 3255. |
| **Df Residuals:** | 990 | **BIC:** | 3305. |
| **Df Model:** | 9 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 0.0836 | 0.039 | 2.138 | 0.033 | 0.007 | 0.160 |
| **spy_lag1** | -0.1567 | 0.091 | -1.730 | 0.084 | -0.335 | 0.021 |
| **sp500** | 0.0221 | 0.014 | 1.621 | 0.105 | -0.005 | 0.049 |
| **nasdaq** | 0.0040 | 0.004 | 1.066 | 0.287 | -0.003 | 0.011 |
| **dji** | -0.0018 | 0.001 | -1.248 | 0.212 | -0.005 | 0.001 |
| **cac40** | -0.0003 | 0.002 | -0.153 | 0.879 | -0.004 | 0.004 |
| **aord** | 0.0093 | 0.001 | 7.492 | 0.000 | 0.007 | 0.012 |
| **daxi** | -0.0025 | 0.001 | -2.387 | 0.017 | -0.005 | -0.000 |
| **nikkei** | -0.0004 | 0.000 | -1.264 | 0.207 | -0.001 | 0.000 |
| **hsi** | 0.0003 | 0.000 | 1.222 | 0.222 | -0.000 | 0.001 |

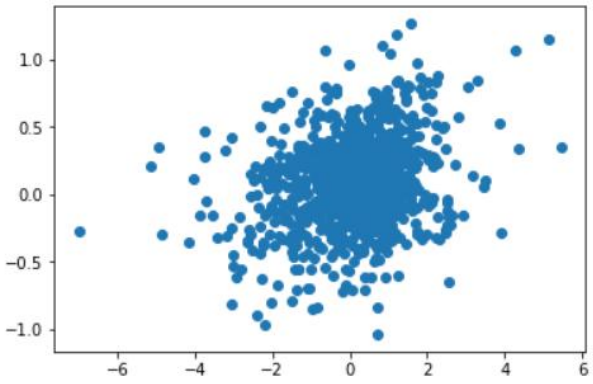
| | | | |
|---|---|---|---|
| **Omnibus:** | 91.018 | **Durbin-Watson:** | 2.015 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 267.687 |
| **Skew:** | -0.450 | **Prob(JB):** | 7.45e-59 |
| **Kurtosis:** | 5.369 | **Cond. No.** | 405. |

## Step 5: Make prediction

In [60]:
```
Train['PredictedY'] = lm.predict(Train)
Test['PredictedY'] = lm.predict(Test)
```

In [61]:
```
plt.scatter(Train['spy'], Train['PredictedY'])
```

Out[61]: <matplotlib.collections.PathCollection at 0x7fda3af1ce80>



## Step 6: Model evaluation - Statistical standard

We can measure the performance of our model using some statistical metrics - **RMSE**, **Adjusted** $R^2$

In [62]:
```
# RMSE - Root Mean Squared Error, Adjusted R^2
def adjustedMetric(data, model, model_k, yname):
    data['yhat'] = model.predict(data)
    SST = ((data[yname] - data[yname].mean())**2).sum()
    SSR = ((data['yhat'] - data[yname].mean())**2).sum()
    SSE = ((data[yname] - data['yhat'])**2).sum()
    r2 = SSR/SST
    adjustR2 = 1 - (1-r2)*(data.shape[0] - 1)/(data.shape[0] -model_k -1)
    RMSE = (SSE/(data.shape[0] -model_k -1))**0.5
    return adjustR2, RMSE
```

In [63]:
```
def assessTable(test, train, model, model_k, yname):
    r2test, RMSEtest = adjustedMetric(test, model, model_k, yname)
    r2train, RMSEtrain = adjustedMetric(train, model, model_k, yname)
    assessment = pd.DataFrame(index=['R2', 'RMSE'], columns=['Train', 'Test'])
    assessment['Train'] = [r2train, RMSEtrain]
    assessment['Test'] = [r2test, RMSEtest]
    return assessment
```

In [64]:
```
# Get the assement table fo our model
assessTable(Test, Train, lm, 9, 'spy')
```

Out[64]:

| | Train | Test |
|---|---|---|
| **R2** | 0.059020 | 0.067248 |
| **RMSE** | 1.226068 | 1.701291 |

# Multiple linear regression model.ipynb在Github中下载

https://github.com/cloudy-sfu/QUN-Data-Analysis-in-Finance/tree/main/Labs

# Jupyternote Book课堂练习三十分钟

In this part, we will use predict price change of SPY as a trading signal and then perform a simple strategy.

05

## Evaluating strategy built from Regression model

Build and evaluate the signal based trading strategy

**The price change of SPY as a trading signal, If the signal is positive, we were long. Otherwise, we were shot.**

# Profit of Signal-based strategy (Train data)

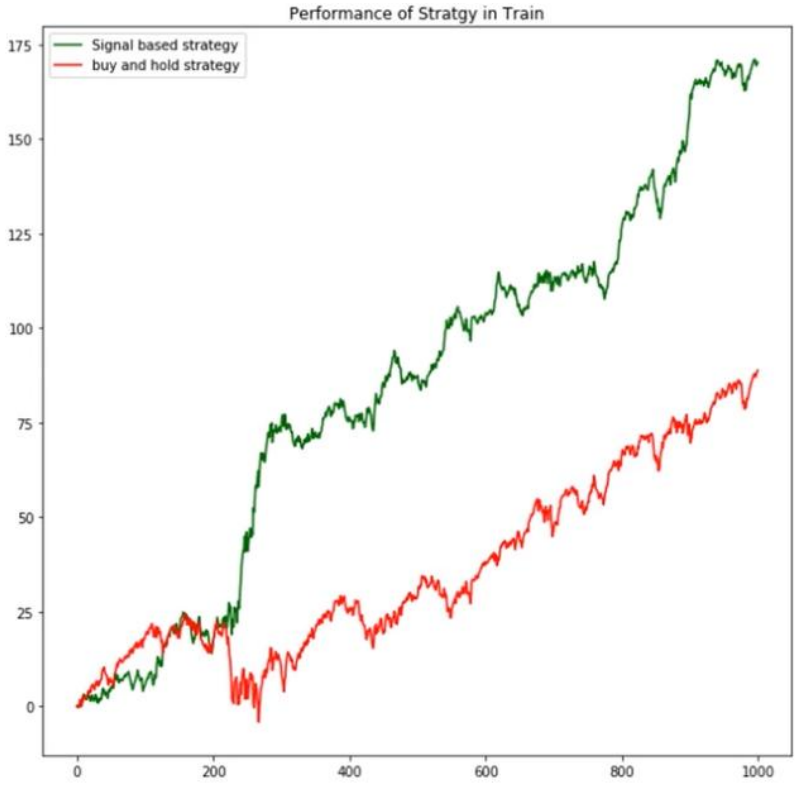
Predicted return of SPY

In [1]
```
Train['Order']=[1 if sig>0 else -1 for sig in Train['PredictedY'] ]
Train['Profit']=Train['spy']*Train['Order']

Train['Wealth']=Train['Profit'].cumsum()
print("Total profit made in train: ", Train['Profit'].sum())
```

Out [1]
```
Total profit made in train: 170.2201830000002
```

**Order equal to one** if **predicted value is positive** or our prediction for price change is positive for opening today to opening tomorrow.

Then, we can **compare the performance of this strategy**, which we call **the signal-based strategy**, with a passive strategy, which we call **buy and hold strategy**, which is to buy more shares of SPY initially and hold it for 1000 days.



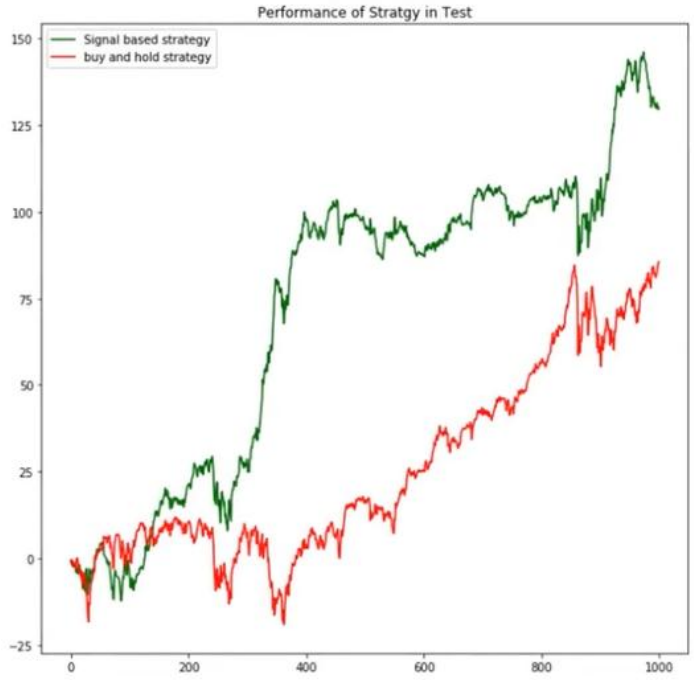

Signal-based

Buy and Hold

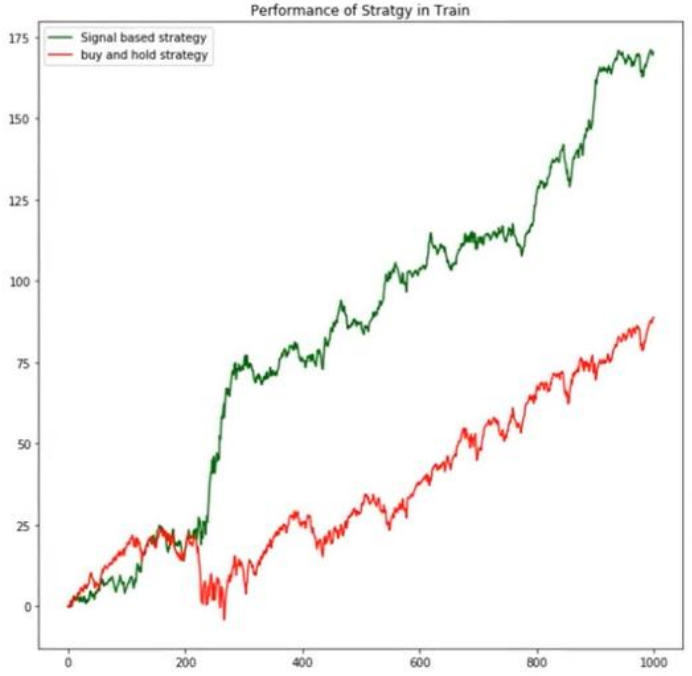
Similarly, we can view it as trading in test dataset, and the total profit is $130, a bit lower than that in train.

**Signal-based strategy also outperforms buy-and-hold strategy in the test data**. The consistency of performance is very important. Otherwise, it is too risky to apply it in the future.

Test




Train



Average daily return is a mirror we can make comparison in finance industry when they use a Sharpe ratio and maximum drawdown.

# Evaluation of model practical standard

1. Sharpe ratio
2. Maximum drawdown

# Sharpe ratio

Measures the excess return (or risk premium) per unit of deviation in an investment asset or a trading strategy, typically referred to as risk.

$$SR = \frac{E(R_a - R_b)}{\sqrt{Var(R_a - R_b)}}$$

Mean of excess return

SD of excess return

*Suppose the daily Sharpe ratio is* $SR_{Day}$. *Then yearly Sharpe ratio is*

$$SR_{Year} = \sqrt{252} SR_{Day}$$

To compute the Sharpe Ratio, we first need to revise wealth process by including initial investment which is the price of one share of SPY.

## Include the initial investment of SPY

```
In [2]  Train['Wealth']=Train['Wealth']+Train.loc[Train.index[0],'Price']
        Test['Wealth']=Test['Wealth']+Test.loc[Test.index[0],'Price']
```

## Sharpe ratio of Train Data

```
In [3]  Train['Return']=np.log(Train['Wealth'])-np.log(Train['Wealth'].shift(1))
        dailyr = Train['Return'].dropna()

        print("daily sharpe ratio is ", dailyr.mean()/dailyr.std(ddof=1))
        print("yearly sharpe ratio is ", (252**0.5)*dailyr.mean()/dailyr.std(ddof=1))

Out [3] daily sharpe ratio is 0.1281151647
        yearly sharpe ratio is 2.03376518983
```

## Sharpe ratio of Test Data

```
In [3]  Test['Return'] = np.log(Test['Wealth'])-np.log(Test['Wealth'].shift(1))
        dailyr = Test['Return'].dropna()
        print("daily sharpe ratio is ", dailyrmean()/dailyr.std(ddof=1))
        print("yearly sharpe ratio is", (252**0.5)*dailyr.mean()/dailyr.std(ddof=1))

Out [3] daily sharpe ratio is 0.0741272648442
        yearly sharpe ratio is 1.17673384888
```

# Maximum drawdown

The maximum percentage decline in the strategy from the historical peak profit at each point in time.

$$\text{drawdown} = \frac{\text{maximum} - \text{wealth}}{\text{maximum}}$$

We first compute drawdown, and then the maximum of all drawdowns in the trading period. **Maximum drawdown is that risk of mirror for extreme loss of a strategy.**

# 例子：

假设你的账户开户本金为5,000美元，在投资或者交易的过程中，你的账户升至了10,000美元，之后又跌至了4,000美元，然后又增长到了12,000美元，接着又跌至了3,000美元，当前又涨至13,000美元。

在这种情形下，资金经历了两次回撤：第一次回撤是10,000回落至4,000美元，回撤幅度为6,000美元，回撤率为6000/10000=60%；第二次回撤是12,000回落至3,000美元，回撤幅度为9,000美元，回撤率为9000/12000=75%。

因而，该账户在开户起至今，最大回撤为75%。

# Compute the peak of wealth process

In [3]
```
Train['Peak']=Train['Wealth'].cummax()
Train['Drawdown']=(Train['Peak']-Train['Wealth'])/Train['Peak']
```

In [4]
```
Test['Peak']=Test['Wealth'].cummax()
Test['Drawdown']=(Test['Peak']-Test['Wealth'])/Test['Peak']
```

In [5]
```
print("Maximum drawdown in train is", Train['Drawdown'].max())
print("Maximum drawdown in test is", Test['Drawdown'].max())
```

Out [5]
```
Maximum drawdown in train is 0.0797611248489
Maximum drawdown in test is 0.092685417175
```

From the mirror of a **Sharpe ratio** and **maximum drawdown**, we can tell that the performance of strategy is quite consistent in place of extreme loss. But the return per unit risk is not very consistent.

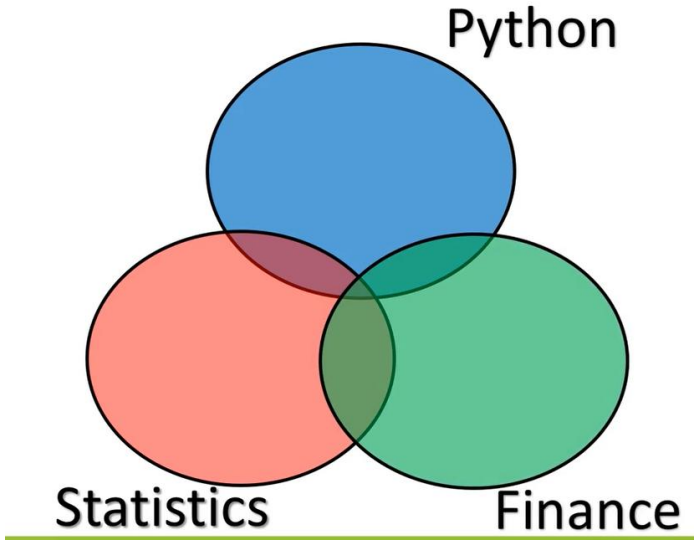
## Overall performance of the strategy

| | Train Data | Test Data |
|---|---|---|
| Sharpe Ratio | 2.03% | 1.17% |
| Maximum Drawdown | 7.98% | 9.27% |

Bid-ask spread
eating up profit ??


Not Overfitted

Consistent
Maximum Drawdown



# Lab3： Evaluating strategy built from Regression model

**Instructions**

- We can created a trading model, let's calculate the profit that would be made if we execute the trading model.
- In addition, we base on two indicators in evaluating the performance of the model - Sharpe ratio and Maximum Drawdown, which will also be illustrated in this Jupyter Notebook.

In [1]:
```
import statsmodels.formula.api as smf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
indicepanel = pd.DataFrame.from_csv('../data/indice/indicepanel.csv')
indicepanel.head()
```

Out[3]:

| Date | spy | spy_lag1 | sp500 | nasdaq | dji | cac40 | daxi | aord | hsi | nikkei | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2008-01-04 | -1.529998 | -1.570008 | -3.540039 | -40.879883 | 2.43945 | 5.489746 | -27.990235 | -2.300293 | 515.349609 | -464.320313 | 143.339996 |
| 2008-01-07 | 0.270004 | -1.529998 | -29.940064 | -56.930176 | -245.40918 | -111.689941 | -102.709961 | -27.500000 | 216.951171 | -48.830078 | 141.809998 |
| 2008-01-08 | -2.990006 | 0.270004 | 1.640015 | -7.179931 | 19.75000 | 44.509766 | 33.680176 | -33.899902 | -354.060547 | 99.370117 | 142.080002 |
| 2008-01-09 | 0.589997 | -2.990006 | -25.459961 | -63.119873 | -230.69043 | -17.109864 | -5.270020 | -12.900390 | 768.359375 | 234.450195 | 139.089996 |
| 2008-01-10 | 1.100006 | 0.589997 | 16.530029 | 8.270019 | 142.90039 | 9.140137 | -40.120117 | -100.200196 | -195.560547 | -158.209961 | 139.679993 |

In [4]:
```
Train = indicepanel.iloc[-2000:-1000, :]
Test = indicepanel.iloc[-1000:, :]
```

In [5]:
```
formula = 'spy~spy_lag1+sp500+nasdaq+dji+cac40+aord+daxi+nikkei+hsi'
lm = smf.ols(formula=formula, data=Train).fit()
```

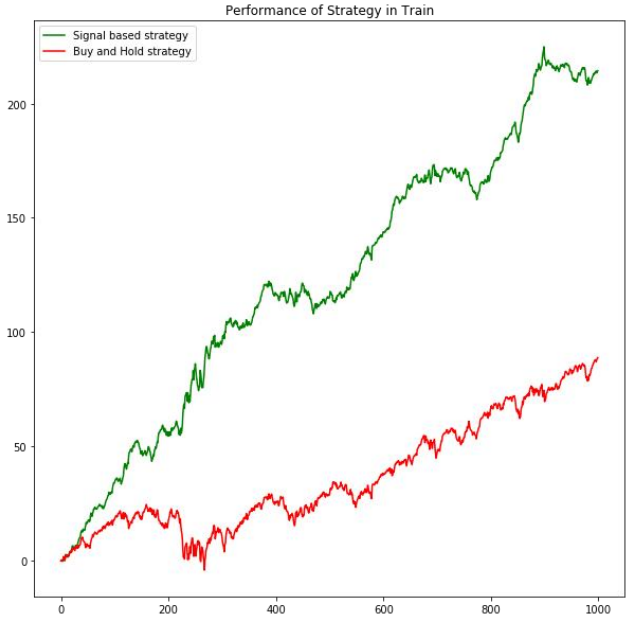
In [6]:
```
Train['PredictedY'] = lm.predict(Train)
Test['PredictedY'] = lm.predict(Test)
```

# Profit of Signal-based strategy

In [7]:
```
# Train
Train['Order'] = [1 if sig>0 else -1 for sig in Train['PredictedY']]
Train['Profit'] = Train['spy'] * Train['Order']

Train['Wealth'] = Train['Profit'].cumsum()
print('Total profit made in Train: ', Train['Profit'].sum())
```

```
Total profit made in Train:  214.340095
```

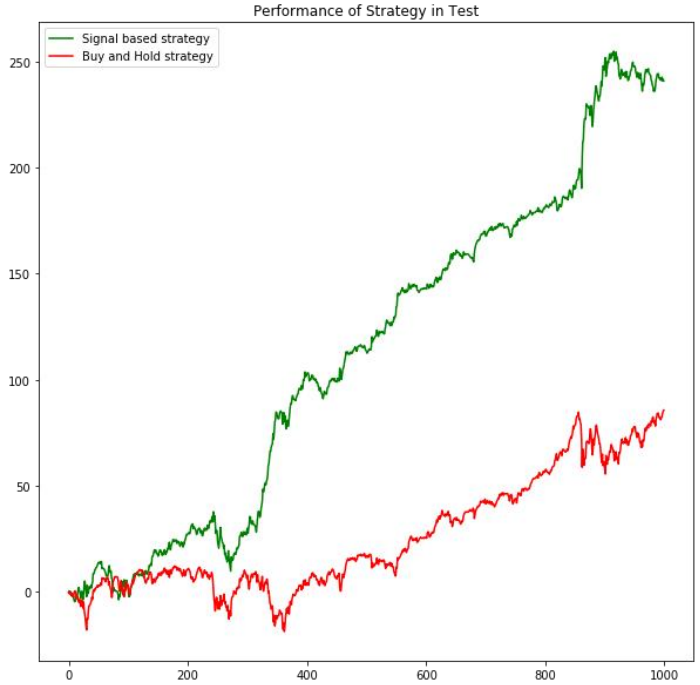
In [8]:
```
plt.figure(figsize=(10, 10))
plt.title('Performance of Strategy in Train')
plt.plot(Train['Wealth'].values, color='green', label='Signal based strategy')
plt.plot(Train['spy'].cumsum().values, color='red', label='Buy and Hold strategy')
plt.legend()
plt.show()
```



In [9]:
```
# Test
Test['Order'] = [1 if sig>0 else -1 for sig in Test['PredictedY']]
Test['Profit'] = Test['spy'] * Test['Order']

Test['Wealth'] = Test['Profit'].cumsum()
print('Total profit made in Test: ', Test['Profit'].sum())
```

```
Total profit made in Test:  241.030088
```

In [10]:
```
plt.figure(figsize=(10, 10))
plt.title('Performance of Strategy in Train')
plt.plot(Test['Wealth'].values, color='green', label='Signal based strategy')
plt.plot(Test['spy'].cumsum().values, color='red', label='Buy and Hold strategy')
plt.legend()
plt.show()
```



# Evaluation of model - Practical Standard

We introduce two common practical standards - **Sharpe Ratio**, **Maximum Drawdown** to evaluate our model performance

In [11]:
```
Train['Wealth'] = Train['Wealth'] + Train.loc[Train.index[0], 'Price']
Test['Wealth'] = Test['Wealth'] + Test.loc[Test.index[0], 'Price']
```

In [12]:
```
# Sharpe Ratio on Train data
Train['Return'] = np.log(Train['Wealth']) - np.log(Train['Wealth'].shift(1))
dailyr = Train['Return'].dropna()

print('Daily Sharpe Ratio is ', dailyr.mean()/dailyr.std(ddof=1))
print('Yearly Sharpe Ratio is ', (252**0.5)*dailyr.mean()/dailyr.std(ddof=1))
```

```
Daily Sharpe Ratio is  0.179650763033
Yearly Sharpe Ratio is  2.85186745096
```

In [13]:
```
# Sharpe Ratio in Test data
Test['Return'] = np.log(Test['Wealth']) - np.log(Test['Wealth'].shift(1))
dailyr = Test['Return'].dropna()

print('Daily Sharpe Ratio is ', dailyr.mean()/dailyr.std(ddof=1))
print('Yearly Sharpe Ratio is ', (252**0.5)*dailyr.mean()/dailyr.std(ddof=1))
```

```
Daily Sharpe Ratio is  0.130351262086
Yearly Sharpe Ratio is  2.06926213537
```

In [14]:
```
# Maximum Drawdown in Train data
Train['Peak'] = Train['Wealth'].cummax()
Train['Drawdown'] = (Train['Peak'] - Train['Wealth'])/Train['Peak']
print('Maximum Drawdown in Train is ', Train['Drawdown'].max())
```

```
Maximum Drawdown in Train is  0.0606901644364
```

In [15]:
```
# Maximum Drawdown in Test data
Test['Peak'] = Test['Wealth'].cummax()
Test['Drawdown'] = (Test['Peak'] - Test['Wealth'])/Test['Peak']
print('Maximum Drawdown in Test is ', Test['Drawdown'].max())
```

```
Maximum Drawdown in Test is  0.117198995246
```

# Evaluating strategy built from Regression model

.ipynb在Github中下载

https://github.com/cloudy-sfu/QUN-Data-Analysis-in-Finance/tree/main/Labs

Jupyternote Book课堂练习
三十分钟

# Thank You