TERA HW#1

§ Question 1

实线绘制 InnerCode 为 3 的股票的全样本收盘价。其中横轴为日期，纵轴为价格。

[50]:
```python
# ready
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
LOC = r'C:\Users\Zheng Xiangzhong\Desktop\GitHub Local\TERA_hw1\play_data.csv'
T = pd.read_csv(LOC)
T = T[T.InnerCode == 3]
T[T.columns[2:-1]].head()
```

[50]:

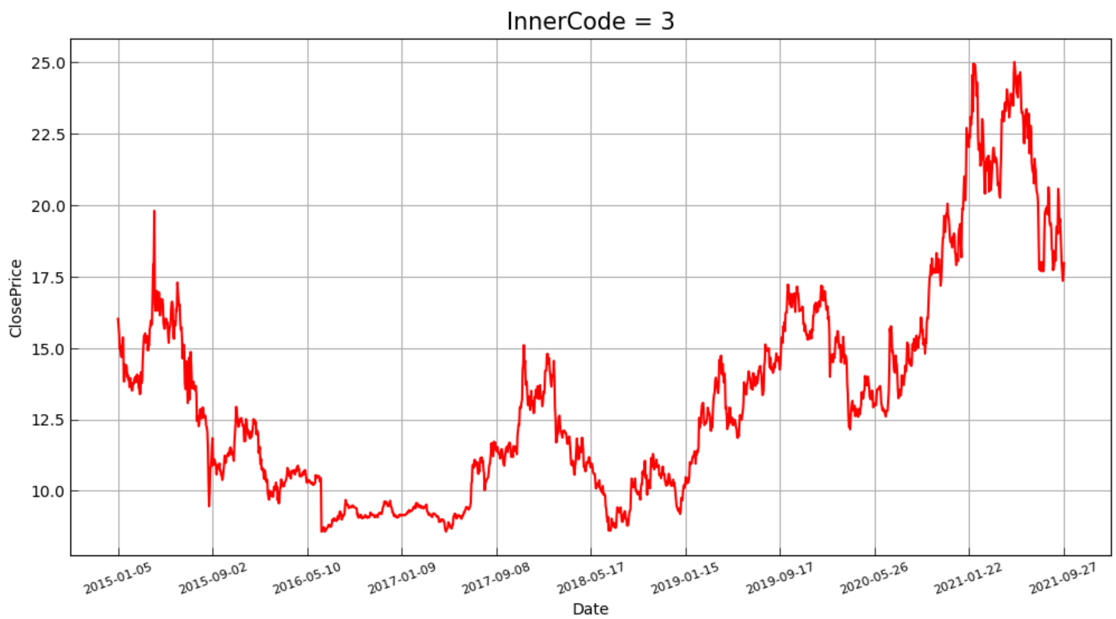
	TradingDay	OpenPrice	HighPrice	LowPrice	ClosePrice	TurnoverVolume \
0	2015-01-05	15.99	16.28	15.60	16.02	286043643.0
1	2015-01-06	15.85	16.39	15.55	15.78	216642140.0
2	2015-01-07	15.56	15.83	15.30	15.48	170012067.0
3	2015-01-08	15.50	15.57	14.90	14.96	140771421.0
4	2015-01-09	14.90	15.87	14.71	15.08	250850023.0

	TurnoverValue	AvgPrice	TotalMV	NegotiableMV
0	4.565388e+09	15.9605	1.830268e+11	1.575841e+11
1	3.453446e+09	15.9408	1.802848e+11	1.552233e+11
2	2.634796e+09	15.4977	1.768574e+11	1.522723e+11
3	2.128003e+09	15.1167	1.709164e+11	1.471572e+11
4	3.835378e+09	15.2895	1.722874e+11	1.483376e+11

[51]:
```python
# 01 basic plot
plt.figure(figsize=(12,6))
plt.plot(T.index,T.ClosePrice,
         linewidth = 1.5,
         linestyle = '-',
         marker = '',
         color = 'r')

# 02 basic elements
ax1 = plt.gca()
ls = np.arange(0,len(T.index),np.floor(len(T.index)/10))
_ = ax1.set_xticks(ls)
_ = ax1.set_xticklabels(T.TradingDay[ls],fontsize = '8')
_ = ax1.set_xlabel('Date')
_ = ax1.set_ylabel('ClosePrice')
_ = ax1.set_title('InnerCode = 3',fontsize = 15)

# 03 polish plot
_ = ax1.tick_params(axis = 'both',direction = 'in',color = 'k',length = 5,width␣
 ↪= 0.5)
_ = ax1.set(facecolor = "white")
_ = plt.xticks(rotation=20)
_ = plt.grid()
```



§ Question 2

使用任意线性模型对 InnerCode 为 3 的股票收盘价建模，解释变量自选，汇报系数估计值，相应的标准误，模型的 R^2 和 $adj - R^2$。

- 在这一节我们定义了一个函数teraOLS()用来执行OLS估计。通过传入DataFrame格式的X和y，可以自动输出相关变量的估计结果。并返回R^2和$adj - R^2$
- 我们使用开盘价(OpenPrice)和总市值(TotalMV)作为解释变量, 收盘价(ClosePrice)作为被解释变量来测试teraOLS()。所有的数据我们都取了对数。

[52]:
```
# OLS
import numpy as np
import pandas as pd
import scipy.stats as stats
def teraOLS(X,y):
    VN = pd.Series(X.columns)
    tmp = pd.Series(['Constant'])
    VN = pd.concat([tmp,VN],ignore_index=True)
    X.insert(0,'Constant',1)
    n,k = X.shape
```

```python
    beta = (np.linalg.inv(X.T.dot(X))).dot(X.T).dot(y).round(4)
    SSR = np.sum(np.square(y.values - (X.dot(beta).values)))
    SST = n * np.var(y).values
    R2 = 1  - SSR/SST
    adjR2 = 1 - (1 - R2)*(n-1)/(n-k-1)
    cov_matrix = (SSR/(n-k))*np.linalg.inv(X.T.dot(X))
    se = np.sqrt(np.diag(cov_matrix)).reshape(k,1)
    t = beta/se
    p_value = 2*(1 - stats.t.cdf(np.abs(t),n-k))

    R = pd.DataFrame({
        'VariableName':VN,
        'Coef':beta.reshape(VN.size),
        's.e.':se.reshape(VN.size).round(4),
        'p-value':p_value.reshape(VN.size).round(4)
    })
    return R,R2[0],adjR2[0]
```

[53]:
```python
# T.columns
# 选取合适的predictors。这里取了对数。
Xcols = T.columns[[3,10]]
Ycols = T.columns[[6]]
X = np.log(T[Xcols])
y = np.log(T[Ycols])
[R,R2,adjR2] = teraOLS(X,y)
print(R,f'\n\nR2:{R2.round(4)}\t',f'adj-R2:{adjR2.round(4)}')
```

```
  VariableName    Coef    s.e.  p-value
0     Constant -0.5821  0.0863      0.0
1    OpenPrice  0.9683  0.0046      0.0
2      TotalMV  0.0254  0.0037      0.0


R2:0.9951        adj-R2:0.9951
```

§ Question 3

针对 InnerCode 为 3 的股票，实现以下滚动窗口预测实验

* 滚动窗口长度为 2000

* 以 2000 为训练集预测下一期价格

* 持续滚动，持续训练，持续预测直到样本结束

* 要求使用以下算法进行训练和预测：线性模型（LM），LASSO , RIDGE , 回归树（RT）, 随机森林（RF）, 支持向量回归-线性核（SVR-L）, 支持向量回归-高斯核（SVR-G）

* 其中涉及到可调参数（Tunning Parameter）的算法，必须写清楚所选的参数的具体细节（以表格的形式进行展示）。

* 使用均方预测误差（MSFE）和平均绝对值预测误差（MAFE）对预测结果进行检验。并汇报最终结果。

Methods	Description
Linear Model	Linear model.
Ridge Regression	Ridge regression with $\lambda = 0.1$.
LASSO	LASSO with $\lambda = 0.1$.
Regression Tree	Regression Tree: the **minimum sample splits** (MSS) is set to n-1, the **minimum leaf size** (MLS) is set to 10.
Random Forest	Random Forest: MNS=n-1 and MSL=10.
SVR_L	Support vector regression with linear kernel.
SVR_G	Support vector regression with gaussian kernel.

- 首先，定义了一个teraPredict()函数，用于执行题目要求的所有算法。输入为训练数据和测试数据，返回值为预测误差。
- 其次，滚动窗口预测部分，窗口长度设定为1000。使用前1000个观测值作为训练数据，并使用训练后的模型来对下一期进行预测。随后训练数据和测试数据均向前前进一期后重复以上操作直至期末。最后使用MAFE和MSFE评估所有模型的预测效果。大部分的算法的参数使用其默认设置。最终的结果根据MSFE升序排列。
- 最后，通过对比最终的结果，OLS在当前样本的预测效果最佳，而且整体上线性模型的性能要优于非线性模型。

[54]:
```
from sklearn import linear_model
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

```python
from sklearn.svm import SVR

def teraPredict(Xt,yt,Xe,ye):
    IND = ['LASSO','Ridge','Regression Tree','Random Forest','SVRL','SVRG']
    res = []
    Xt = Xt.values
    yt = yt.values
    Xe = Xe.values
    ye = ye.values

    n,k = Xt.shape
    # 01. LASSO
    lasso = linear_model.Lasso(alpha=0.1,fit_intercept=True,max_iter=10000)
    lasso.fit(Xt,yt)
    yhat = lasso.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    # 02. RIDGE
    ridge = linear_model.Ridge(alpha=0.1,fit_intercept=True)
    ridge.fit(Xt,yt)
    yhat = ridge.predict(np.reshape(Xe,(1,-1)))[0][0]
    res.append(ye[0]-yhat)

    # 03. Regression Tree
    rt = DecisionTreeRegressor(min_samples_split=n-1,min_samples_leaf=10)
    rt.fit(Xt,yt)
    yhat = rt.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    # 04. Random Forest
    rf = 
 ↪RandomForestRegressor(min_samples_split=n-1,min_samples_leaf=10,max_features=0.
 ↪7)
    rf.fit(Xt,yt.reshape(n,))
    yhat = rf.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)
```

```
    # 05. SVR linear
    svrl = SVR(kernel='linear')
    svrl.fit(Xt,yt.reshape(n,))
    yhat = svrl.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    # 06. SVR gaussian
    svrg = SVR(kernel='rbf')
    svrg.fit(Xt,yt.reshape(n,))
    yhat = svrg.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    return res
```

[55]:
```
import time
L = 1000
st = time.time()
X = np.log(T[Xcols])
y = T[Ycols]
IND = ['LASSO','Ridge','Regression Tree','Random Forest','SVRL','SVRG','LM']
FINAL = pd.DataFrame(columns=IND)
for i in range(X.shape[0]-L):
    Xt = X.loc[i:L+i-1,:]
    yt = y.loc[i:L+i-1,:]
    Xe = X.loc[i+L]
    ye = y.loc[i+L]
    F = teraPredict(Xt,yt,Xe,ye)

    # teraOLS
    [R,tmp1,tmp2] = teraOLS(Xt,yt)
    yhatols = sum(R.Coef.values[1:] * Xe.values) + R.Coef.values[0]
    F.append(ye[0]-yhatols)
    FINAL.loc[i,:] = np.array(F)


ed = time.time()
```

```python
print(f'Time: {ed - st}')
```

Time: 120.37467646598816

[56]:
```python
SHOW = pd.DataFrame({
    'MSFE':np.square(FINAL).mean(axis=0),
    'MAFE':np.abs(FINAL).mean(axis=0)
})
SHOW.sort_values(['MSFE'])
```

[56]:

	MSFE	MAFE
LM	0.756730	0.598162
Ridge	0.769672	0.603835
SVRL	1.398101	0.789715
LASSO	2.582155	1.258166
Regression Tree	8.457337	2.244403
SVRG	21.676175	4.061538
Random Forest	29.154823	4.627971

§ Question 3.1 Parallel Version

[63]:
```python
def teraPredict_2(data):
    import statsmodels.api as sm
    from sklearn import linear_model
    from sklearn.linear_model import Lasso
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.svm import SVR
    import numpy as np
    import pandas as pd


    # IND = ['OLS','LASSO','Ridge','Regression Tree','Random␣
↪Forest','SVRL','SVRG']
    res = []
    Xt, yt, Xe, ye = data
    Xt = Xt.values
    yt = yt.values
```

```python
    Xe = Xe.values
    ye = ye.values

    n,k = Xt.shape
    # 00. OLS
    Xtols = sm.add_constant(Xt)
    # print(Xtols)
    Xeols  = Xe[:]
    Xeols = np.insert(Xeols,0,1)
    ols = sm.OLS(yt,Xtols).fit()
    yhat = ols.predict(np.reshape(Xeols,(1,-1)))
    res.append(ye[0] - yhat[0])

    # 01. LASSO
    lasso = linear_model.Lasso(alpha=0.1,fit_intercept=True,max_iter=10000)
    lasso.fit(Xt,yt)
    yhat = lasso.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    # 02. RIDGE
    ridge = linear_model.Ridge(alpha=0.1,fit_intercept=True)
    ridge.fit(Xt,yt)
    yhat = ridge.predict(np.reshape(Xe,(1,-1)))[0][0]
    res.append(ye[0]-yhat)

    # 03. Regression Tree
    rt = DecisionTreeRegressor(min_samples_split=n-1,min_samples_leaf=10)
    rt.fit(Xt,yt)
    yhat = rt.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    # 04. Random Forest
    rf =␣
↪RandomForestRegressor(min_samples_split=n-1,min_samples_leaf=10,max_features=0.
↪7)
    rf.fit(Xt,yt.reshape(n,))
```

```python
    yhat = rf.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    # 05. SVR linear
    svrl = SVR(kernel='linear')
    svrl.fit(Xt,yt.reshape(n,))
    yhat = svrl.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)

    # 06. SVR gaussian
    svrg = SVR(kernel='rbf')
    svrg.fit(Xt,yt.reshape(n,))
    yhat = svrg.predict(np.reshape(Xe,(1,-1)))[0]
    res.append(ye[0]-yhat)



    return res
```

[64]:
```python
%%time


import multiprocess as mp
if __name__ == '__main__':
    # start worker pool
    num_cores = mp.cpu_count()
    pool = mp.Pool(processes=num_cores)
    # Generate data partitions
    X = np.log(T[Xcols])
    y = T[Ycols]
    L = 1000
    datafull = []
    for i in range(X.shape[0]-L):
        Xt = X.loc[i:L+i-1,:]
        yt = y.loc[i:L+i-1,:]
        Xe = X.loc[i+L]
        ye = y.loc[i+L]
        datafull.append([Xt,yt,Xe,ye])
```

```python
    # parallel estimation
    results = pool.map(teraPredict_2, datafull)
    pool.close()
    pool.join()
```

```
CPU times: total: 922 ms
Wall time: 18.2 s
```

[65]:
```python
IND = ['OLS','LASSO','Ridge','Regression Tree','Random Forest','SVRL','SVRG']
SHOW = pd.DataFrame({
    'MSFE':np.square(np.array(results)).mean(axis=0),
    'MAFE':np.abs(np.array(results)).mean(axis=0)
},
index = IND)
SHOW.sort_values(['MSFE'])
```

[65]:

	MSFE	MAFE
OLS	0.756838	0.598189
Ridge	0.769672	0.603835
SVRL	1.398101	0.789715
LASSO	2.582155	1.258166
Regression Tree	8.457337	2.244403
SVRG	21.676175	4.061538
Random Forest	29.159337	4.627779

§ Question 4

使用一个和 3d 提及的不同的算法，在上述滚动窗口预测实验中，击败所有提及的算法。

- 在这里我们使用了XGBoost算法来进行滚动窗口预测。在默认设定下我们将MSFE和MAFE进一步降低。

[8]:
```python
from xgboost import XGBRegressor
import time
XGB= []
X = np.log(T[Xcols])
y = T[Ycols]
```

```python
start = time.time()
for i in range(X.shape[0]-L):
    Xt = X.loc[i:L+i-1,:].values
    yt = y.loc[i:L+i-1,:].values
    Xe = X.loc[i+L]
    ye = y.loc[i+L].values
    model = XGBRegressor()
    model.fit(Xt,yt)
    yhat = model.predict(Xe.values.reshape((1,2)))[0]
    XGB.append(ye[0] - yhat)
end = time.time()
print(f'Time: {end - start}')
```

```
Time: 44.62667775154114
```

[9]:
```python
FINAL['XGBoost'] = XGB
SHOW2 = pd.DataFrame({
    'MSFE':np.square(FINAL).mean(axis=0),
    'MAFE':np.abs(FINAL).mean(axis=0)
})
SHOW2.sort_values(['MSFE'])
```

[9]:

	MSFE	MAFE
XGBoost	0.059793	0.133228
LM	0.756730	0.598162
Ridge	0.769672	0.603835
SVRL	1.398101	0.789715
LASSO	2.582155	1.258166
Regression Tree	8.457337	2.244403
SVRG	21.676175	4.061538
Random Forest	29.156186	4.627596