Arbitrage Pricing Theory (APT)-yfinance

July 2, 2021

1 Import Pac

[4]:
```
!pip install yfinance
```

```
Requirement already satisfied: yfinance in c:\users\rluck\anaconda3\lib\site-
packages (0.1.59)
Requirement already satisfied: pandas>=0.24 in
c:\users\rluck\anaconda3\lib\site-packages (from yfinance) (1.2.4)
Requirement already satisfied: multitasking>=0.0.7 in
c:\users\rluck\anaconda3\lib\site-packages (from yfinance) (0.0.9)
Requirement already satisfied: lxml>=4.5.1 in c:\users\rluck\anaconda3\lib\site-
packages (from yfinance) (4.6.3)
Requirement already satisfied: requests>=2.20 in
c:\users\rluck\anaconda3\lib\site-packages (from yfinance) (2.25.1)
Requirement already satisfied: numpy>=1.15 in c:\users\rluck\anaconda3\lib\site-
packages (from yfinance) (1.19.5)
Requirement already satisfied: python-dateutil>=2.7.3 in
c:\users\rluck\anaconda3\lib\site-packages (from pandas>=0.24->yfinance) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in
c:\users\rluck\anaconda3\lib\site-packages (from pandas>=0.24->yfinance)
(2021.1)
Requirement already satisfied: six>=1.5 in c:\users\rluck\anaconda3\lib\site-
packages (from python-dateutil>=2.7.3->pandas>=0.24->yfinance) (1.15.0)
Requirement already satisfied: chardet<5,>=3.0.2 in
c:\users\rluck\anaconda3\lib\site-packages (from requests>=2.20->yfinance)
(4.0.0)
Requirement already satisfied: idna<3,>=2.5 in
c:\users\rluck\anaconda3\lib\site-packages (from requests>=2.20->yfinance)
(2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
c:\users\rluck\anaconda3\lib\site-packages (from requests>=2.20->yfinance)
(1.26.4)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\rluck\anaconda3\lib\site-packages (from requests>=2.20->yfinance)
(2020.12.5)
```

[6]:
```
import pandas as pd
import fix_yahoo_finance as yf
import pandas_datareader as dat
import numpy as np
import matplotlib
import statsmodel                    mf
import statsmodel
import yfinance a
```


2 Reading da finance

[18]:
```
#S&P500 =sp
sp = yf.Ticker("^GSPC").history(
                    start='2016-1-1',
                    end='2021-5-25')
#Stock (Nike)= st
st = yf.Ticker("NKE").history(
                    start='2016-1-1',
                    end='2021-5-25')
#Wilshire 5000 index
wls= yf.Ticker("^W5000").history(
                    start='2016-1-1',
                    end='2021-5-25')
#Russell 1000 value index
rlv = yf.Ticker("^RLV").history(
                    start='2016-1-1',
                    end='2021-5-25')
#Risk-free rate (Rf)
rf=sp = yf.Ticker("^IRX").history(
                    start='2016-1-1',
                    end='2021-5-25')
```


3 Computing Annualised Returns

$R = 365 * ln(p_t/p_{t-1})$

[23]:
```
#Stock returns
R =365*np.log(st['Close']/st['Close'].shift(1)).dropna()
#Market Index returns: S&P500
M =365*np.log(sp['Close']/sp['Close'].shift(1)).dropna()
#Size index: Wilshire 5000 index
S =365*np.log(wls['Close']/wls['Close'].shift(1)).dropna()
#Value index: Russell 1000 value index
V =365*np.log(rlv['Close']/rlv['Close'].shift(1)).dropna()
#Risk-free rate returns
Rf =(rf['Close']/100).dropna()
```

[24]:
```
#Determining the mean returns of NIKE, S&P500, Wilshire 5000 index, Russell␣
↪1000 value index
name= ['r_n','r_m','r_s','r_v','r_f']
mean=[R.mean(),M.mean(), S.mean(),V.mean(),Rf.mean()]
ret= (name,mean)
ret
```

[24]:
```
(['r_n', 'r_m', '
 [0.2209737717412
  -1.260160089123
  0.1961535758087
  0.1323768070689
  0.010230334309814671])
```

[25]:
```
# Determining the volatilities of NIKE stock, S&P500 index, Wilshire 5000 index␣
↪and Russell 1000 value index
name= ['s_n','s_m','s_s','s_v','s_f']
std=[R.var()**0.5,M.var()**0.5, S.var()**0.5,V.var()**0.5,Rf.var()**0.5]
std= (name,std)
std
```

[25]:
```
(['s_n', 's_m', 's_s', 's_v', 's_f'],
 [6.444606637495693,
  59.28057766874355,
  4.4475397913934795,
  4.459231818857248,
  0.008357364178046637])
```


4 Merging the columns into in one worksheet

[26]:
```
dt_M =pd.merge(M,Rf, on='Date', how='left').dropna()
dt =pd.merge(dt_M,R, on='Date', how='left').dropna()
dt_1= pd.merge(dt,S, on ='Date', how='left').dropna()
dta= pd.merge(dt_1,V, on='Date', how='left').dropna()
```

5 Renaming the Row Header

[27]:
```
dta_cols=['M','Rf','St','S','V']
dta.columns =dta_cols
dta
```

[27]:

Date	M	Rf	St	S	V
2016-01-04	16.867686	0.00155	-5.768505	-5.673750	-4.756967
2016-01-05	102.048469	0.00205	5.067068	0.674867	0.960959
2016-01-06	0.000000	0.00205	-5.245099	-5.043475	-5.916716

```
2016-01-07  -27.734857  0.00190 -9.867127 -9.041746 -8.455889
2016-01-08    0.000000  0.00190 -5.026039 -4.063790 -4.[illegible]
...                ...      ...       ...       ...       ...
2021-05-18  252.998721  0.00010 -2.281533 -2.558221 -3.413909
2021-05-19 -252.9[illegible]  [illegible]  -1.068576 -1.279263 -4.202521
2021-05-20 -186.4[illegible]  [illegible]   4.850007  4.053303  2.184694
2021-05-21    0.0[illegible]  [illegible]   0.674484 -0.180701  3.541917
2021-05-24    0.0[illegible]  [illegible]   3.831742  3.592793  1.392827

[1337 rows x 5 co[illegible]
```


6 OLS Regression to determine beta under APT (3-factor Model)

[28]:
```
#Factor Risk Premium
dta['Rp']= dta['M']-dta['Rf']
dta['Rs'] = dta['S']-dta['M']
dta['Rv']= dta['V']-dta['M']
#X & y Variables defined
X = dta [['Rp','Rs','Rv']]
X = sm.add_constant(X)
y= dta.St-dta.Rf
#OLS model
model = sm.OLS(y,X).fit()
predictions =model.predict(X)
Q = model.summary()
print(Q)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.432
Model:                            OLS   Adj. R-squared:                  0.430
Method:                 Least Squares   F-statistic:                     337.6
Date:                Fri, 02 Jul 2021   Prob (F-statistic):          4.74e-163
Time:                        20:51:30   Log-Likelihood:                -4007.6
No. Observations:                1337   AIC:                             8023.
Df Residuals:                    1333   BIC:                             8044.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0635      0.133      0.478      0.633      -0.197       0.324
Rp             0.9601      0.030     31.657      0.000       0.901       1.020
Rs             0.7754      0.084      9.180      0.000       0.610       0.941
Rv             0.1873      0.084      2.222      0.026       0.022       0.353
==============================================================================
Omnibus:                      458.212   Durbin-Watson:                   2.060
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8258.868
Skew:                           1.414   Prob(JB):                        0.00
Kurtosis:                      14.970   Cond. No.                         103.
==============================================================================

Notes:
[1] Standard Error[illegible] covariance matrix of the errors is correctly
specified.
```

[29]:
```
#Determining the [illegible] factor risk premiums of NIKE, S&P500,␣
↪Wilshire 5000 [illegible]1000 value index based on average.
f_m = M.mean()-Rf
f_s = S.mean()-M.mean()
f_v = V.mean()-M.mean()
r_f= Rf.mean()
```

[30]:
```
#Determining Expected Returns from APT given factor risk premiums
ER = r_f + model.params['Rp']*f_m+model.params['Rs']*f_s+model.params['Rv']*f_v
ER
```

[30]: 0.18060162154485404

[31]:
```
#Determining Alpha (or excess returns)
Alpha = R.mean()-ER
Alpha
```

[31]: 0.04037215019642912

[ ]:

[ ]:

[ ]: