Computing Performance Measures of SPY

We are going to use the ETF SPY to implement some performance measures. This ETF provides a convenient and inexpensive way to invest in the S&P500 index.

In this activity, we are going to work with a time-series. A time-series consists of observations, prices in our case, happening at specific points in time (days in our data). In this case, we say that prices are indexed by time, meaning that for a given time (day) you are going to have one observation. Usually indexes are distinct (i.e. you do not have multiple observations per day), but this is not a necessary condition.

Let's start by loading a CSV file into a Data Frame. To familiarize yourself with the data, you can open it in Excel. We are going to only keep two columns ('Date' and 'Adj Close') from this file.

Python also offers the possibility to work with dates, so we are going to transform the date from string into a datetime (that's the type of the variable that handles dates in Python)

In [2]:
```
import pandas as pd
#Load data from CSV file
data = pd.read_csv("SHV.csv")
#Keep Date and Adjusted Close
data = data[["Date","Adj Close"]]
data.rename(columns={"Adj Close": 'Price'}, inplace=True) #rename columns for convenience
#Build a column of type datetime
data["Date"] = pd.to_datetime(data.Date,format='%Y-%m-%d') #transform from string into an object of t
ype datetime
data.head()
```

Out[2]:

	Date	Price
0	2014-01-02	105.245224
1	2014-01-03	105.264328
2	2014-01-06	105.254799
3	2014-01-07	105.254799
4	2014-01-08	105.264328

Next, we need now to load auxiliary data about risk free rates and risk factors of interest. We are going to use data from https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html (https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html)

The file that we need from there is csv file "Fama/French 3 Factors".

> Note: daily return values are provided in percentage.

In [3]:
```
aux_data = pd.read_csv("F-F_Research_Data_Factors_daily.csv",skiprows=4,skipfooter=1,engine='python')
#skip the first rows and the last one as they contain descr. info
aux_data.rename(columns={"Unnamed: 0": "Date"}, inplace=True) #rename column
#Build a column of type datetime, then use it to index (label) prices
aux_data["Date"] = pd.to_datetime(aux_data["Date"],format='%Y%m%d') #transform from string into an ob
ject of type datetime
aux_data.head()
```

Out[3]:

	Date	Mkt-RF	SMB	HML	RF
0	1926-07-01	0.10	-0.24	-0.28	0.009
1	1926-07-02	0.45	-0.32	-0.08	0.009
2	1926-07-06	0.17	0.27	-0.35	0.009
3	1926-07-07	0.09	-0.59	0.03	0.009
4	1926-07-08	0.21	-0.36	0.15	0.009

Merging data sets

The problem we face now is that we have two data sets: one with prices for SPY and other with risk factors and risk-free rates. We are going to create a new data set out of these two, that is, we are going to **merge** the time-series of SPY with that of risk factors and risk free rates. When we merge, there has to be a common element in both data sets. In this case, we want that for a given date (a row), the values of SPY, the three factors, and the risk-free rate are stacked toghether in different columns.

Date	Price
2014-01-02	163.38

+

Date	Mkt-RF	SMB	HML	RF
2014-01-02	-0.88	-0.27	0.12	0.0

=

Date	Price	Mkt-RF	SMB	HML	RF
2014-01-02	163.38	-0.88	-0.27	0.12	0.0

Note that the time-series for risk factors is much longer than that of SPY. When merging both data sets, we are going to keep only observations that match the time period of SPY.

In [4]:
```
data_full = pd.merge(data, aux_data, left_on='Date',right_on='Date',how='left')
data_full.head()
```

Out[4]:

	Date	Price	Mkt-RF	SMB	HML	RF
0	2014-01-02	105.245224	-0.88	-0.27	0.12	0.0
1	2014-01-03	105.264328	0.03	0.36	0.05	0.0
2	2014-01-06	105.254799	-0.34	-0.57	0.26	0.0
3	2014-01-07	105.254799	0.68	0.40	-0.41	0.0
4	2014-01-08	105.264328	0.04	0.01	-0.11	0.0

You can take a look at the merged data sets in Excel:

In [5]:
```
# data_full.to_excel('merged_SPY_FFfile.xlsx')
```

Compute performance measures

To compute performance measures, we need returns and excess returns. Bear in mind that the risk-free rate we have is daily (in percentage).

In [6]:
```
import numpy as np
data_full["RF_d"] = data_full.RF/100 #compute daily risk-free rate in decimals
data_full["Return"] = data_full.Price.pct_change()
data_full["Ex_return"] = data_full.Return - data_full.RF_d #compute excess returns
#Compute arithmetic and geometric average returns
avg_return = data_full.Return.mean()
numPeriods = len(data_full) #obtain the number of periods in the data set
avg_Greturn = np.power(np.prod(1+data_full.Return),1/numPeriods)-1
volatility = data_full.Return.std()
#Annualized measures
annual_return = avg_return * 250
annual_vol = volatility * np.sqrt(250)
print("The arithmetic average return (in %%) for SPY is %5.3f, the gemoteric return is %5.3f, and the
volatility is %5.3f"\
      %(avg_return*100,avg_Greturn*100,volatility*100))
print("Annualized quantities correspond to %5.3f, %5.3f, and %5.3f, respectively."%(annual_return*100
,annual_Greturn*100,\
                                                                                    annual_vol*100))
#Compute Sharpe Ratio
avg_Exreturn = data_full.Ex_return.mean()
volatility_Exreturn = data_full.Ex_return.std()
SR = (avg_Exreturn/volatility_Exreturn)*np.sqrt(250)
print("The annualized average excess return is %5.3f and its volatility is %5.3f"%(avg_Exreturn*250*1
00,volatility_Exreturn*np.sqrt(250)*100))
print("The annualized Sharpe Ratio for the SPY is estmated at %4.3f" %(SR))
```

```
The arithmetic average return (in %) for SPY is 0.003, the gemoteric return is 0.003, and the volati
lity is 0.014
Annualized quantities correspond to 0.834, 0.837, and 0.218, respectively.
The annualized average excess return is 0.023 and its volatility is 0.209
The annualized Sharpe Ratio for the SPY is estmated at 0.108
```

Risk-Adjusted Performance

Now we are going to compute the market risk-adjusted performance of SPY. We need to estimate the following regression model:

$$R_t^e = \alpha + \beta R_t^{M,e} + \epsilon_t$$

To this end, we are going to use a module that performs this regression. When specifing the regression model, observe that we need a model with intercept (alpha in our case).

In [ ]:

In [91]:
```
import statsmodels.api as sm
data_full_n = data_full.dropna() #drop nan values
X = data_full_n["Mkt-RF"]
y = data_full_n["Ex_return"]*100 #Set excess returns in percentage
X = sm.add_constant(X) ## add an intercept (alpha) to the regression model

# create a model, then fit
model = sm.OLS(y, X).fit()
# Print out the statistics
#model.summary()
#Print out parameters of interests
alpha = model.params[0]
beta = model.params[1]
print("Alpha is %6.4f and beta %4.2f" %(alpha*250,beta))
```

```
Alpha is 0.0420 and beta -0.00
```

Once this model is estimated, we can also estimate the IR:

$$IR = \frac{\alpha}{\sigma(\epsilon)}$$

We have already computed alpha, so we need to compute the standard deviation of the idiosyncratic component (ϵ). These values correspond to the errors in the regression model.

In [92]:
```
#Compute IR
res = model.resid #get the regression residuals
std_res = np.std(res)
IR = alpha/std_res
print("The annualized IR in this case is %4.3f" %(IR*np.sqrt(250)))
```

```
The annualized IR in this case is 0.202
```

Statistics about Drawdowns

Next, we are going to look into High Water Marker and related statistics. The HWM can be seen as the maximum price observed up to a given point in time (how high the waters have risen)

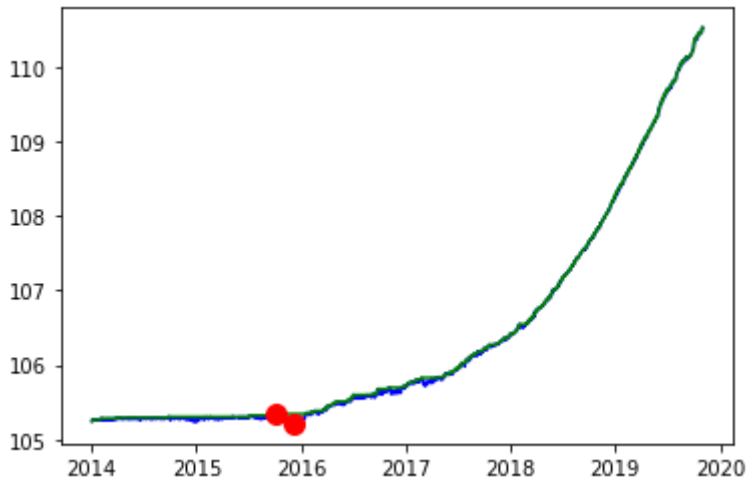
Once we have defined the highest point at a given time, we can look at the drawdown (how low we are with respect to the previous highest point) and the maximum drawdown (the worst case drop so far)

In [93]:
```
HWM = np.maximum.accumulate(data_full.Price) #Compute High Water Marker - the running maximum
DD = (HWM - data_full.Price)/HWM #Drawdown
MDD = np.maximum.accumulate(DD) #Maximum Drawdown


i = np.argmax(HWM - data_full.Price) # point (position in the list) at which the difference is the ma
ximum between HWM and price (largest drop)
j = np.argmax(data_full.Price[:i]) # point (position in the list) at which the maximum value is obser
ved before i

#The period [j,i] correspond to the duration of the maximum drawdown
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(data_full.Date, data_full.Price, color='blue')
plt.plot(data_full.Date, HWM,color = 'green')
start_date = data_full.Date[j] #date when the highest price was observed (before the largest drop)
end_date = data_full.Date[i] #date when the largest drop was observed
start_prc = data_full.Price[j] #obtain the price when the highest price observed (before the largest
 drop)
end_prc = data_full.Price[i] #obtain the price when the largest drop was observed
plt.plot([start_date, end_date], [start_prc, end_prc], 'o', color='Red', markersize=10)
```

Out[93]:
```
[<matplotlib.lines.Line2D at 0x12619afd0>]
```



In [94]:
```
print("The MDD started in %s when the price was %5.2f" %(start_date.strftime("%m/%d/%Y"),start_prc))
print("The MDD ended in %s when the price was %5.2f" %(end_date.strftime("%m/%d/%Y"),end_prc))
print("The MDD was %5.2f%%, in dollars $%5.2f" %(MDD[i]*100,MDD[i]*start_prc))
delta = end_date - start_date #compute the difference between two dates
print("The MDD lasted %d days" %(delta.days))
```

```
The MDD started in 10/02/2015 when the price was 105.34
The MDD ended in 12/08/2015 when the price was 105.21
The MDD was  0.12%, in dollars $ 0.13
The MDD lasted 67 days
```