Computing Maximum Drawdown

In this lab, we'll develop the code to compute the maximum drawdown of a return series, and we'll
which will act as a toolkit that we will add to during the course.

First, let's read the return series we processed in the previous lab:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os

DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/data'
print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")
```

```
import pandas as pd

me_m = pd.read_csv(DATAPATH + "/Portfolios_Formed_on_ME_monthly_EW.csv",
```

```
                   header=0, index_col=0, parse_dates=True, na_values=-99.99)
rets = me_m[['Lo 10', 'Hi 10']]
rets.columns = ['SmallCap', 'LargeCap']
rets = rets/100
rets.plot.line()
```

Timeseries - forcing the index to be a datetime

We asked Pandas to `parse_dates` in `read_csv()`. Let's check if it was able to do so with the inde

```
rets.index
```

The `dtype` is `int64` which suggests that it was not automatically converted to a date time index,
simplest way to force it to be a timeseries is by reformatting the index data to a `datetime` type as

```
rets.index = pd.to_datetime(rets.index, format="%Y%m")
rets.index
```

Now that the DataFrame has a datetime index, we can treat the entire dataframe as a timeseries, w
For instance, we can extract just the returns in 2008 as follows:

```
rets["2008"]
```

This looks good except that we know this is monthly data, and it's showing up with an index that is
the `to_period` method. We'll see several more examples of Pandas support for timeseries during

```
rets.index = rets.index.to_period('M')
rets.head()
```

```
rets.info()
```

```
rets.describe()
```

Computing Drawdowns

1. Convert the time series of returns to a time series that represents a wealth index
2. Compute a time series of the previous peaks
3. Compute the Drawdown as the difference between the previous peak and the current value

Let's do this for Large Cap stocks.

```
wealth_index = 1000*(1+rets["LargeCap"]).cumprod()
wealth_index.plot()
```

```
previous_peaks = wealth_index.cummax()
previous_peaks.plot()
```

```
drawdown = (wealth_index - previous_peaks)/previous_peaks
drawdown.plot()
```

```
drawdown.min()
```

```
drawdown["1975":].plot()
```

```
drawdown["1975":].min()
```

Creating a Drawdown Function

Redoing this analysis for SmallCap would be tedious, since we would need to re-enter all these cor create our first function that will form the first tool in our financial toolkit.

The function will take as input, a timeseries of returns, and return a timeseries as a DataFrame that previous peaks and the drawdowns as a percent.

```
def drawdown(return_series: pd.Series):
    """Takes a time series of asset returns.
       returns a DataFrame with columns for
```

```
        the wealth index,
        the previous peaks, and
        the percentage drawdown
    """
    wealth_index = 1000*(1+return_series).cumprod()
    previous_peaks = wealth_index.cummax()
    drawdowns = (wealth_index - previous_peaks)/previous_peaks
    return pd.DataFrame({"Wealth": wealth_index,
                         "Previous Peak": previous_peaks,
                         "Drawdown": drawdowns})

drawdown(rets["LargeCap"]).head()
```

```
drawdown(rets["LargeCap"]).min()
```

```
Period('1932-05', 'M')
```

```
drawdown(rets["SmallCap"]).min()
```

```
Period('1932-05', 'M')
```

```
drawdown(rets["LargeCap"])["Drawdown"].idxmin()
```

```
Period('2009-02', 'M')
```

```
drawdown(rets["SmallCap"])["Drawdown"].idxmin()
```

```
-0.6312068077252386
```

```python
drawdown(rets["LargeCap"]["1975":])["Drawdown"].idxmin()
```

```python
drawdown(rets["SmallCap"]["1975":])["Drawdown"].idxmin()
```

```python
drawdown(rets["SmallCap"]["1975":])["Drawdown"].min()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2231857d68>
```

```python

```

Computing Maximum Drawdown

In this lab, we'll develop the code to compute the maximum drawdown of a return series, and we'll start to develop our own module which will act as a toolkit that we will add to during the course.

First, let's read the return series we processed in the previous lab:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os

DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio
print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")
```

```
import pandas as pd

me_m = pd.read_csv(DATAPATH + "/Portfolios_Formed_on_ME_monthly_EW.csv"
```

```
                     header=0, index_col=0, parse_dates=True, na_values=-
rets = me_m[['Lo 10', 'Hi 10']]
rets.columns = ['SmallCap', 'LargeCap']
rets = rets/100
rets.plot.line()
```

Timeseries - forcing the index to be a datetime

We asked Pandas to `parse_dates` in `read_csv()`. Let's check if it was able to do so with the index:

```
rets.index
```

The `dtype` is `int64` which suggests that it was not automatically converted to a date time index, so let's do that now manually. The simplest way to force it to be a timeseries is by reformatting the index data to a `datetime` type as follows:

```
rets.index = pd.to_datetime(rets.index, format="%Y%m")
rets.index
```

Now that the DataFrame has a datetime index, we can treat the entire dataframe as a timeseries, which makes things very convenient. For instance, we can extract just the returns in 2008 as follows:

```
rets["2008"]
```

This looks good except that we know this is monthly data, and it's showing up with an index that is date stamped. We can fix this using the `to_period` method. We'll see several more examples of Pandas support for timeseries during the course.

```
rets.index = rets.index.to_period('M')
rets.head()
```

```
rets.info()
```

```
rets.describe()
```

Computing Drawdowns

1. Convert the time series of returns to a time series that represents a wealth index
2. Compute a time series of the previous peaks
3. Compute the Drawdown as the difference between the previous peak and the current value

Let's do this for Large Cap stocks.

```
wealth_index = 1000*(1+rets["LargeCap"]).cumprod()
wealth_index.plot()
```

```
previous_peaks = wealth_index.cummax()
previous_peaks.plot()
```

```
drawdown = (wealth_index - previous_peaks)/previous_peaks
drawdown.plot()
```

```
drawdown.min()
```

```
drawdown["1975":].plot()
```

```
drawdown["1975":].min()
```

Creating a Drawdown Function

Redoing this analysis for SmallCap would be tedious, since we would need to re-enter all these commands at the prompt. Instead, let's create our first function that will form the first tool in our financial toolkit.

The function will take as input, a timeseries of returns, and return a timeseries as a DataFrame that contains the wealth index, the previous peaks and the drawdowns as a percent.

```
def drawdown(return_series: pd.Series):
    """Takes a time series of asset returns.
       returns a DataFrame with columns for
```

```
        the wealth index,
        the previous peaks, and
        the percentage drawdown
    """
    wealth_index = 1000*(1+return_series).cumprod()
    previous_peaks = wealth_index.cummax()
    drawdowns = (wealth_index - previous_peaks)/previous_peaks
    return pd.DataFrame({"Wealth": wealth_index,
                         "Previous Peak": previous_peaks,
                         "Drawdown": drawdowns})

drawdown(rets["LargeCap"]).head()
```

```
drawdown(rets["LargeCap"]).min()
```

```
drawdown(rets["SmallCap"]).min()
```

```
drawdown(rets["LargeCap"])["Drawdown"].idxmin()
```

```
drawdown(rets["SmallCap"])["Drawdown"].idxmin()
```

```
drawdown(rets["LargeCap"]["1975":])["Drawdown"].idxmin()
```

```
drawdown(rets["SmallCap"]["1975":])["Drawdown"].idxmin()
```

```
drawdown(rets["SmallCap"]["1975":])["Drawdown"].min()
```