R + Python in R Markdown with `reticulate`

The `reticulate` package allows you to use R and Python simultaneously, seamlessly translating your data and functions between both languages. This allows you to combine R and Python in your work.

For example, you might do your data wrangling and modeling in Python, but then use R for visualization.

In this R Notebook, we'll show how you can use Python to get some stock data for 5 technology stocks, build a weighted portfolio and calculate returns, and then directly visualize the data in R with `ggplot2`.

First, let's load our raw price data. We can do this in Python using the `DataReader` function from `pandas_datareader`, and we can run our Python code inside of a code chunk within this R Markdown doc.

```
# First define our 5 tech stocks:
tickers = ['AAPL', 'AMZN', 'GOOG', 'NFLX', 'TSLA']

# Get data starting in 2010 and ending today
start_date = '2010-01-01'
end_date = date.today()

# User pandas_reader.data.DataReader to load the desired data from Yahoo Finance
stock_data = data.DataReader(tickers, 'yahoo', start_date, end_date)

# Get the Adjusted Close columns and store it in a new object, prices
prices = stock_data['Adj Close']

prices.head()
```

```
## Symbols           AAPL        AMZN        GOOG      NFLX  TSLA
## Date
## 2010-01-04  26.466835  133.899994  312.204773  7.640000   NaN
## 2010-01-05  26.512596  134.690002  310.829926  7.358572   NaN
## 2010-01-06  26.090879  132.250000  302.994293  7.617143   NaN
## 2010-01-07  26.042646  130.000000  295.940735  7.485714   NaN
## 2010-01-08  26.215786  133.520004  299.885956  7.614286   NaN
```

For this exercise, we've chosen 5 tech stocks (Apple, Amazon, Google, Netflix, and Tesla) and have obtained the adjusted closing price from Yahoo Finance starting January 2010. (Note Tesla does not have any price data until its IPO in June 2010).

Our next step is to calculate the daily returns for each stock.

```
# Calculate daily returns for each stock
returns = prices.pct_change(1)

# Fill missing values with 0
returns = returns.fillna(0)

returns.head()
```

```
## Symbols          AAPL      AMZN      GOOG      NFLX  TSLA
## Date
## 2010-01-04  0.000000  0.000000  0.000000  0.000000   0.0
```

```
## 2010-01-05  0.001729  0.005900 -0.004404 -0.036836   0.0
## 2010-01-06 -0.015906 -0.018116 -0.025209  0.035139   0.0
## 2010-01-07 -0.001849 -0.017013 -0.023280 -0.017254   0.0
## 2010-01-08  0.006648  0.027077  0.013331  0.017176   0.0
```

Now that we have our daily returns, we will choose our weights for our portfolio. For this exercise, we'll choose basic weights of 30% for `AAPL` and `AMZN`, 25% for `GOOG`, 10% for `NFLX`, and 5% for `TSLA`.

```
# Assign weights
weights = [0.3, 0.3, 0.25, 0.10, 0.05]

# Calculate individually weighted returns
weighted_returns = (weights * returns)

weighted_returns.head()
```

```
## Symbols          AAPL      AMZN      GOOG      NFLX  TSLA
## Date
## 2010-01-04  0.000000  0.000000  0.000000  0.000000   0.0
## 2010-01-05  0.000519  0.001770 -0.001101 -0.003684   0.0
## 2010-01-06 -0.004772 -0.005435 -0.006302  0.003514   0.0
## 2010-01-07 -0.000555 -0.005104 -0.005820 -0.001725   0.0
## 2010-01-08  0.001994  0.008123  0.003333  0.001718   0.0
```

Our final step is to sum our weighted returns for each stock across columns to calculate the daily returns for our portfolio.

```
# Sum across columns to calculate portfolio returns
portfolio_returns = weighted_returns.sum(axis=1).to_frame()

# Move Date index to column and rename returns column
portfolio_returns = portfolio_returns.reset_index()
portfolio_returns.rename(columns={0:'returns'}, inplace = True)

portfolio_returns.head()
```

```
##         Date   returns
## 0 2010-01-04  0.000000
## 1 2010-01-05 -0.002496
## 2 2010-01-06 -0.012995
## 3 2010-01-07 -0.013204
## 4 2010-01-08  0.015168
```

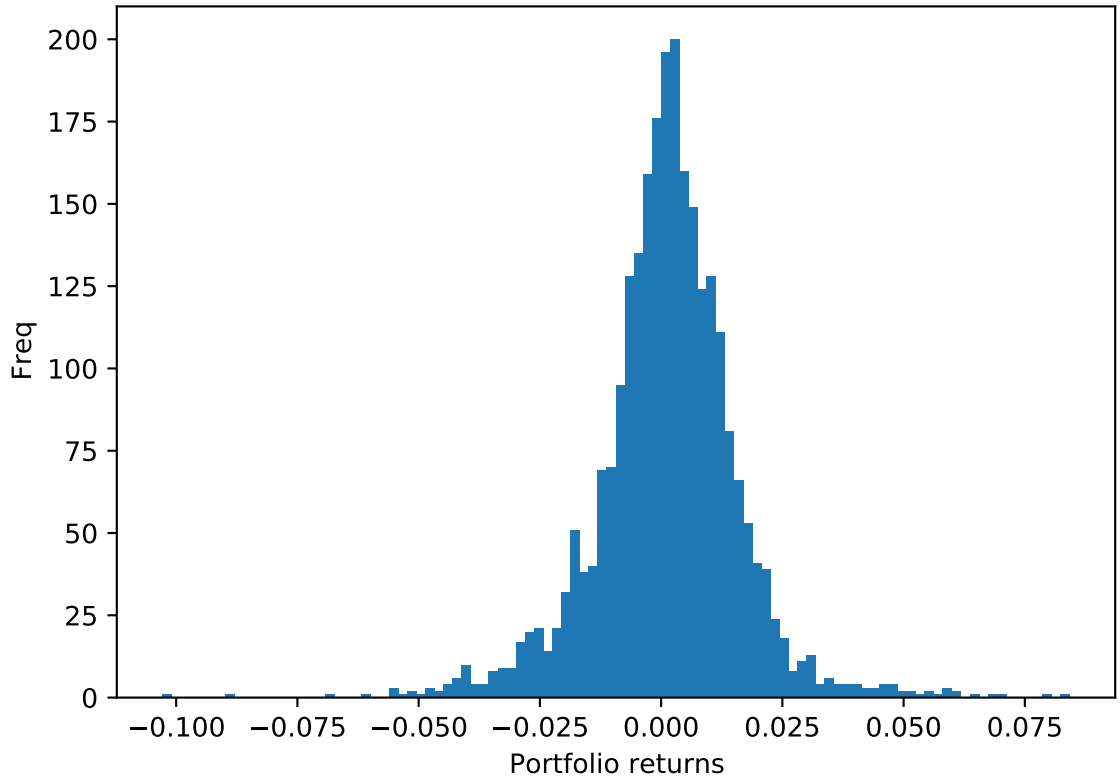
Great! Let's take a look at the distribution of our returns. RStudio and R Markdown / R Notebooks include support for common Python visualization libraries like `matplotlib` and `seaborn`.

```
fig = plt.figure()
ax1 = fig.add_axes([0.1,0.1,0.8,0.8])
ax1.hist(portfolio_returns['returns'], bins = 100)
```

```
## (array([  1.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,
##           0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,
##           1.,   0.,   0.,   3.,   1.,   2.,   1.,   3.,   2.,   4.,   6.,
##          10.,   4.,   4.,   8.,   9.,   9.,  17.,  20.,  21.,  14.,  21.,
##          32.,  51.,  38.,  40.,  69.,  70.,  95., 128., 135., 159., 176.,
##         196., 200., 160., 149., 124., 128., 111.,  81.,  66.,  53.,  41.,
##          39.,  24.,  18.,   8.,  11.,  13.,   4.,   6.,   4.,   4.,   4.,
##           3.,   3.,   4.,   4.,   2.,   2.,   1.,   2.,   1.,   3.,   2.,
```

```
##          0.,   1.,   0.,   1.,   1.,   0.,   0.,   0.,   0.,   1.,   0.,
##          1.]), array([-1.02883137e-01, -1.01011547e-01, -9.91399578e-02, -9.72683684e-02,
##        -9.53967790e-02, -9.35251896e-02, -9.16536002e-02, -8.97820107e-02,
##        -8.79104213e-02, -8.60388319e-02, -8.41672425e-02, -8.22956531e-02,
##        -8.04240637e-02, -7.85524743e-02, -7.66808849e-02, -7.48092955e-02,
##        -7.29377061e-02, -7.10661166e-02, -6.91945272e-02, -6.73229378e-02,
##        -6.54513484e-02, -6.35797590e-02, -6.17081696e-02, -5.98365802e-02,
##        -5.79649908e-02, -5.60934014e-02, -5.42218120e-02, -5.23502225e-02,
##        -5.04786331e-02, -4.86070437e-02, -4.67354543e-02, -4.48638649e-02,
##        -4.29922755e-02, -4.11206861e-02, -3.92490967e-02, -3.73775073e-02,
##        -3.55059179e-02, -3.36343285e-02, -3.17627390e-02, -2.98911496e-02,
##        -2.80195602e-02, -2.61479708e-02, -2.42763814e-02, -2.24047920e-02,
##        -2.05332026e-02, -1.86616132e-02, -1.67900238e-02, -1.49184344e-02,
##        -1.30468449e-02, -1.11752555e-02, -9.30366613e-03, -7.43207672e-03,
##        -5.56048731e-03, -3.68889790e-03, -1.81730849e-03,  5.42809225e-05,
##         1.92587033e-03,  3.79745974e-03,  5.66904915e-03,  7.54063856e-03,
##         9.41222797e-03,  1.12838174e-02,  1.31554068e-02,  1.50269962e-02,
##         1.68985856e-02,  1.87701750e-02,  2.06417644e-02,  2.25133538e-02,
##         2.43849432e-02,  2.62565327e-02,  2.81281221e-02,  2.99997115e-02,
##         3.18713009e-02,  3.37428903e-02,  3.56144797e-02,  3.74860691e-02,
##         3.93576585e-02,  4.12292479e-02,  4.31008373e-02,  4.49724268e-02,
##         4.68440162e-02,  4.87156056e-02,  5.05871950e-02,  5.24587844e-02,
##         5.43303738e-02,  5.62019632e-02,  5.80735526e-02,  5.99451420e-02,
##         6.18167314e-02,  6.36883209e-02,  6.55599103e-02,  6.74314997e-02,
##         6.93030891e-02,  7.11746785e-02,  7.30462679e-02,  7.49178573e-02,
##         7.67894467e-02,  7.86610361e-02,  8.05326255e-02,  8.24042149e-02,
##         8.42758044e-02]), <a list of 100 Patch objects>)
```

```
ax1.set_xlabel('Portfolio returns')
ax1.set_ylabel("Freq")
plt.show();
```



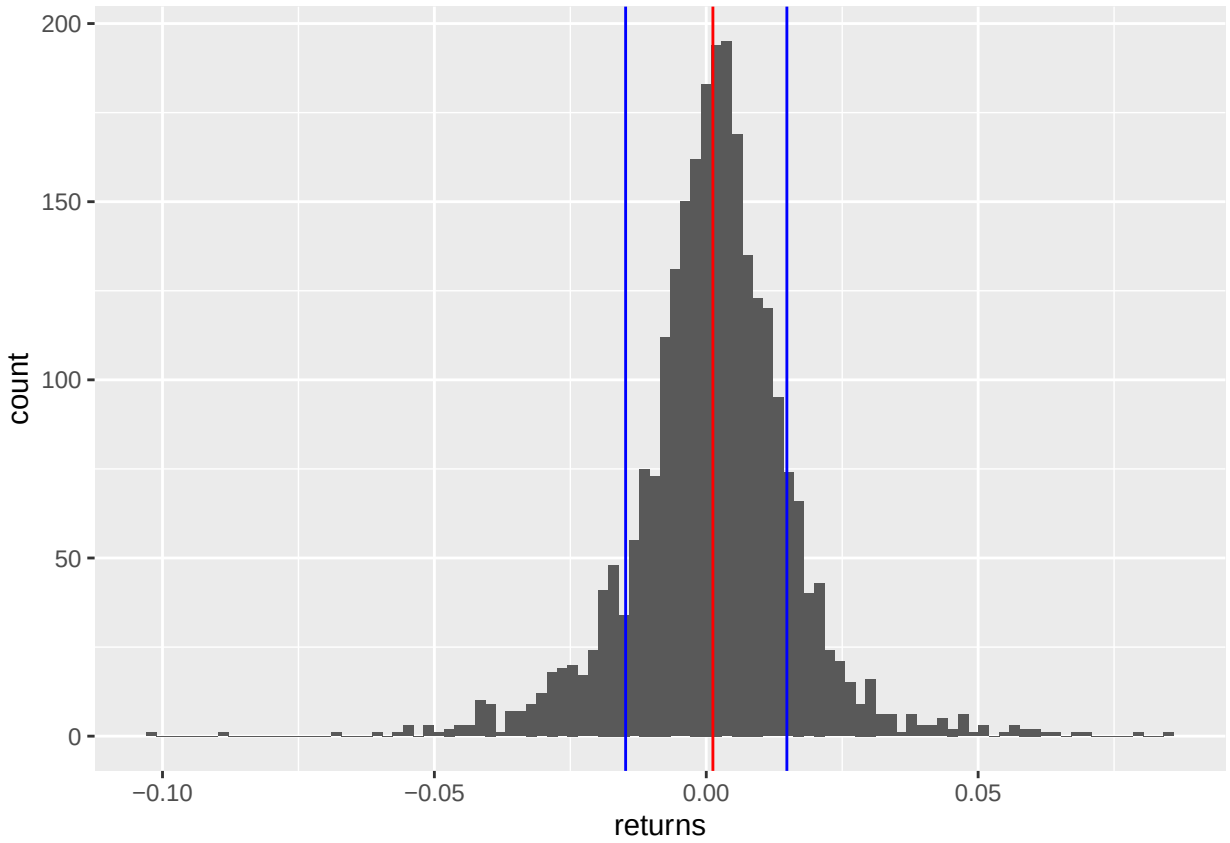
Combining R and Python

So far, we've learned how to get our stock data and built a simple portfolio in Python. Now let's visualize our data in R using `ggplot2`.

To access our pandas dataframes from our Python environment, all we have to do is add `py$` in front of whatever object we want to access from our Python environment.

We can easily build the same histogram we just made in Python with `matplotlib` but with R and `ggplot2`. We'll also add some vertical lines to indicate the mean and +/- 1 SD.

```
py$portfolio_returns %>%
  ggplot(aes(x = returns)) +
  geom_histogram(bins = 100) +
  geom_vline(xintercept = sd(py$portfolio_returns$returns), color = "blue") +
  geom_vline(xintercept = -sd(py$portfolio_returns$returns), color = "blue") +
  geom_vline(xintercept = mean(py$portfolio_returns$returns), color = "red")
```



Let's make one more plot. This time, we'll use R to calculate the cumulative returns for each stock and for our portfolio as a whole, then we'll plot those time series to see how our stocks and portfolio has performed over time.

```
# Create a function for calculating a 30-day rolling average, to be used on our portfolio returns
roll_avg <- slidify(.f = AVERAGE, .period = 30, .align = "center", .partial = TRUE)

# Add a column for cumulative returns to our portfolio data, and another column for a 30-day rolling av
port_returns <- py$portfolio_returns %>%
  mutate(cr = cumprod(1 + returns)) %>%
  mutate(cumulative_returns = cr - 1) %>%
  mutate(date = as.Date(Date)) %>%
  mutate(rolling_avg = roll_avg(cumulative_returns))

# Do the same for our stock returns, then plot both data sets using geom_line()
returns_plot <-
  py$returns %>%
  rownames_to_column("date") %>%
  pivot_longer(-date, "stock", values_to = "returns") %>%
  group_by(stock) %>%
  slice(-1) %>%
  mutate(cr = cumprod(1 + returns)) %>%
  mutate(cumulative_returns = cr - 1) %>%
  mutate(date = as.Date(date)) %>%
  ggplot() +
  geom_line(aes(x = date, y = cumulative_returns, group = stock, color = stock)) +
  scale_x_date(date_breaks = "year", date_labels = "%Y") +
```

```
  labs(y = "cumulative returns", caption = "Black line represents 30-day moving avg. of weighted portfol
  geom_line(data = port_returns, aes(x = date, y = rolling_avg))


returns_plot
```