The Mathematica® Journal

Foundations of Computational Finance

Ramesh Adhikari

The Wolfram Language has numerous knowledge-based built-in functions to support financial computations. This article introduces many built-in and other financial functions that are based on concepts and models covered in undergraduate-level finance courses. Examples are taken from a wide range of finance areas. They emphasize importing and visualization of data from many sources, valuation, capital budgeting, analysis of stock returns, portfolio optimization and analysis of bonds and stock options. We hope that all the functions selected in this article are very useful for analyzing real-world financial data. All examples provide a unique set of tools for users to engage with real-world financial data and solve practical problems. The feature of automatic data retrieval from online sources and its analysis makes all results reproducible without any modifications in the code. We hope this feature will attract new users from the finance community.

1. Introduction

Finance is computational in nature and often involves the analysis and visualization of complex data, optimization, simulation and use of data for risk management. Without the proper use of technology, it is almost impossible to analyze these functions of modern finance. Moreover, the field of finance has become far more driven by data and technology since the 1990s, which has made large-scale data analysis the norm. Data-driven decision-making and predictive modeling are now the heart of every strategic financial decision. Since the publication of Varian [1, 2], Shaw [3] and Stojanovic [4], there have been many updates and new functions, but no new articles or books have been written to cover wide areas of computational finance. This article provides a comprehensive overview of functions related to finance and introduces many functions that are useful for real-world financial data analysis using Mathematica 12.

We have provided all the custom functions in the text so that users can make changes as they learn how to program in the Wolfram Language. Furthermore, we minimize the explanation of any financial concepts in this article as our focus is on introducing financial application of the Wolfram Language.

We begin by defining some symbols that are frequently used as input arguments of custom functions in the article. The article uses `symbol` or `symbols`, `startDate`, `endDate` and `period` as arguments in many functions defined in this article. Most of these symbols are used as input in the built-in function `FinancialData`. All these arguments must be specified in the format acceptable in the `FinancialData` function. We use `symbol` or `symbols` to represent a company's or companies' stock ticker symbol or symbols. It could be a string or a list of strings. The format `startDate` represents the start date of the sample period specified and `endDate` represents the last date of the analysis period. Both must be specified as date objects in any date format supported by `DateObject`. Similarly, `period` represents data frequency. It may include `"Day"`, `"Week"`, `"Month"` or `"Year"`. In the subsequent functions defined in this article, we will not describe them when they are used as arguments.

The article is organized into 13 sections:

1. this Introduction
2. importing and visualizing data from different sources
3. capital budgeting and business valuation
4. functions for the analysis of security returns
5. rolling-window performance analysis
6. financial application of optimization
7. decomposing the risk of a portfolio into its components
8. importing factor data and running factor models
9. computing different types of portfolio performance measures
10. technical analysis of stock prices
11. bond analysis
12. analyzing derivative products
13. concluding remarks

2. Accessing Financial Data and Basic Data Visualization

The most commonly used built-in functions for retrieving company-specific financial data are `Company`, `CompanyData` and `Financial`. For example,

```
CompanyData[EntityValue[
  Interpreter["Company"]["FB"], "Company"], "Dataset"]
```

imports Facebook's financial statement data and

```
Interpreter["Financial"]["FB"]["Dataset"]
```

imports Facebook's price-related data.

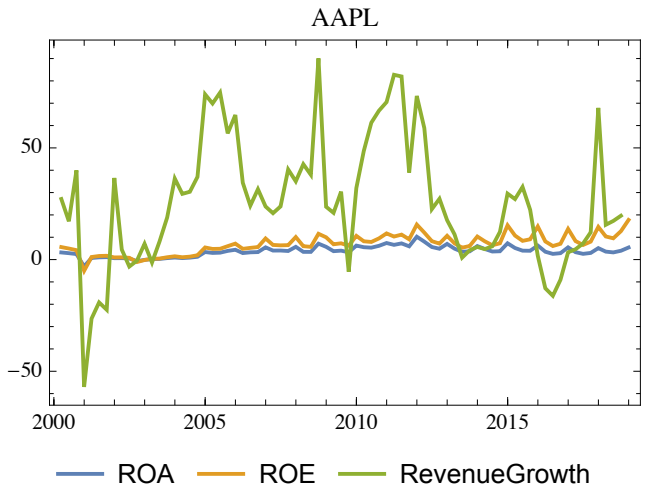
Similarly, the function `FinancialData` can be used to get data about stocks and other financial instruments. `DateListPlot` or `CandlestickChart` can be used to chart prices against time. `TradingChart` or `InteractiveTradingChart` can be used to make interactive plots with additional features of adding different technical indicators. Other functions such as `KagiChart`, `RenkoChart`, `PointFigureChart` or `LineBreakChart` can also be used to visualize financial data. In the remaining part of this section, we are going to show you how to import data from different sources and visualize it.

```
In[1]:= plotCompanyData[symbol_String, startDate_, endDate_,
          properties : {__String}] :=
        DateListPlot[
         EntityValue[
          EntityValue[Interpreter["Financial"][symbol],
           "Company"], Dated[properties,
           Interval[{DateObject[startDate],
             DateObject[endDate]}]]
         ],
         PlotLabel → symbol,
         PlotLegends → Placed[properties, Bottom],
         PlotRange → All
        ]
```

We download Apple's return on assets (ROA), return on equity (ROE) and revenue growth over the period January 1, 2001, to January 1, 2019, and plot them.

```
In[2]:= plotCompanyData["AAPL", {2000, 1, 1}, {2019, 1, 1},
         {"ROA", "ROE", "RevenueGrowth"}]
```

Out[2]=



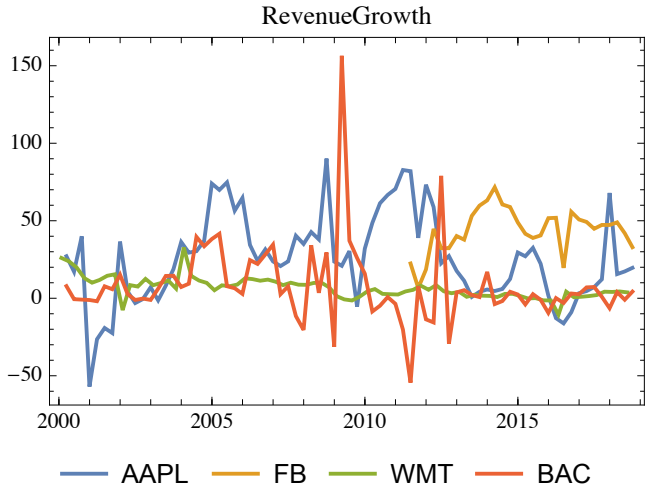
Similarly, we define the `plotCompaniesData` function to compare any specified property of different companies. The function takes a list of stock symbols, beginning period, end period and a property to consider as its arguments.

```
In[3]:= plotCompaniesData[symbols : {__String}, startDate_,
          endDate_, property_String] :=
        DateListPlot[
         EntityValue[
          EntityValue[Interpreter["Financial"][#], "Company"] & /@
           symbols,
          Dated[property,
           Interval[{DateObject[startDate], DateObject[endDate]}]]
         ],
         PlotLabel → property, PlotLegends → Placed[symbols, Bottom],
         PlotRange → Full, PlotRange → All
        ]
```

We plot the revenue growth of Apple, Facebook, Walmart and Bank of America over the period January 1, 2000, to January 1, 2019.

```
In[4]:= plotCompaniesData[{"AAPL", "FB", "WMT", "BAC"},
         {2000, 1, 1}, {2019, 1, 1}, "RevenueGrowth"]
```

Out[4]=



Second, we import and visualize data from the Federal Reserve Bank of St. Louis, as it is one of the most important data sources when it comes to economic data. The built-in function `ServiceExecute` can be used to request the data from the Federal Reserve Economic Data API. Its argument structure is:

```
ServiceExecute["FederalReserveEconomicData",
 "SeriesData", "ID" → {ids}]
```

where `ids` is a series ID or a list of IDs. It returns a time series containing data for the specified series. It is often of interest to plot the economic time series with the recession dates.

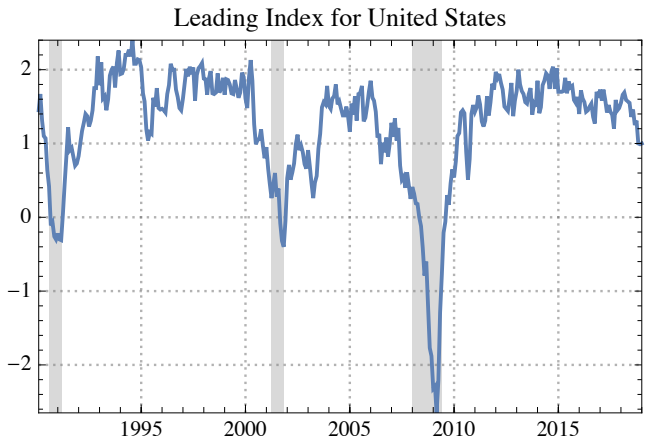
The `plotFedData` function downloads and plots the selected series along with the shaded recession period. The function takes series ID, start date, end date and title as inputs and returns a graph. It uses recession indicators based on USREC (US recession) data from the National Bureau of Economic Research (NBER) for the United States from the period following the peak through the trough to indicate the recession period. The Federal Reserve Bank of St. Louis may require the API key to download its data. The API key can be obtained freely by creating a user account at https://fred.stlouisfed.org (click "my account" and follow the instructions).

```
In[5]:= plotFedData[property_String, startDate_, endDate_,
          title_String] := Module[
          {conn, recessionIndicators, recessionDates, data},
          conn = ServiceConnect["FederalReserveEconomicData"];
          recessionIndicators =
           TimeSeriesWindow[conn["SeriesData", "ID" → {"USREC"}],
            {startDate, endDate}];
          recessionDates =
           Join[recessionIndicators["Dates"][[#]] & /@
             Position[
              Prepend[Differences[recessionIndicators["Values"]],
               0], 1.], recessionIndicators["Dates"][[#]] & /@
             Position[
              Append[Differences[recessionIndicators["Values"]], 0],
              -1.], 2];
          data = TimeSeriesWindow[
            conn["SeriesData", "ID" → {property}],
            {startDate, endDate}];
          Show[
           DateListPlot[
            Thread[{#, Min[data["Values"]]}] & /@ recessionDates,
            Joined → True, Filling → Top,
            FillingStyle → Opacity[0.3, Gray], PlotStyle → None,
            PlotRange → {{First[data["Dates"]], Last[data["Dates"]]},
              {Min[data["Values"]], Max[data["Values"]]}},
            PlotTheme → "Detailed", PlotLabel → title],
           DateListPlot[data, Joined → True, PlotRange → All]
          ]
         ]
```

Now we can download any series and plot it. For example, we download and plot the Leading Index for the United States (USSLIND) over the period January 30, 1990, to January 30, 2019. Please use the API key 207071a5f2e90e7816259d3c32c1ab81 if needed. The shaded regions indicate recession periods.

```
In[6]:= plotFedData["USSLIND", {1990, 01, 30}, {2019, 01, 30},
         "Leading Index for United States"]
```

Out[6]=

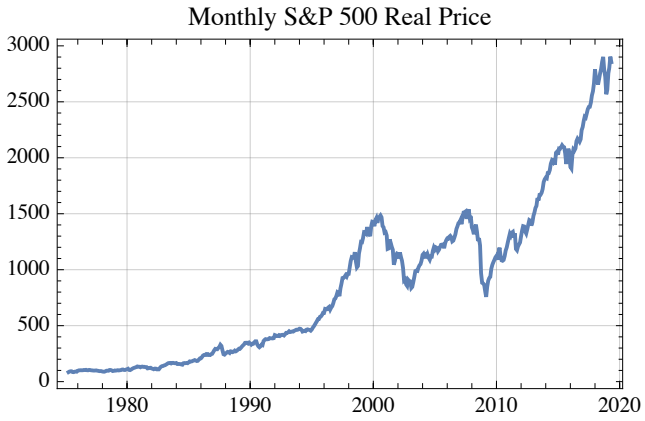



```
In[7]:= getQuandlData[dataID_String, startDate_, endDate_,
          APIKey_String] := Module[
          {data},
          data = Import[
            StringJoin[
             "https://www.quandl.com/api/v3/datasets/",
             dataID,
             "/data.csv?start_date=",
             startDate,
             "&end_date=",
             endDate,
             "&order=asc&api_key=",
             APIKey]
           ];
          TimeSeries[data[[2 ;;, 2]],
           {ToExpression[StringSplit[data[[2 ;;, 1]], "-"]]}]
        ]
```

We download and plot the historical real S&P 500 prices by month (MULTPL/SP500_ REAL_PRICE _MONTH) over the period March 31, 1975, to May 30, 2019.

```
In[8]:= DateListPlot[
         getQuandlData["MULTPL/SP500_REAL_PRICE_MONTH",
          "1975-03-31", "2019-05-30", "G_ydzsMB4zEWcWz3xNs4"],
         GridLines → Automatic,
         PlotLabel → "Monthly S&P 500 Real Price"
        ]
```

Out[8]=



Finally, we show how to create a dataset. The built-in function `Dataset` is very useful for organizing large or small sets of data.

The function `getCompanyData` can be used to get a company's fundamental data. After the data is stored, we can organize and analyze the data.

```
In[9]:= getCompanyData[symbol_String, startDate_, endDate_,
          properties : {__String}] := Module[
          {data, mergedData, dates},
          data = EntityValue[
            EntityValue[Interpreter["Financial"][symbol], "Company"],
            Dated[properties,
             Interval[{DateObject[startDate],
               DateObject[endDate]}]]
           ];
          mergedData = TimeSeriesThread[# &, data,
            ResamplingMethod → "NaN"];
          dates = DateObject[#] & /@
            mergedData["Dates"][[All, 1, {1, 2, 3}]];
          Dataset@Map[
            AssociationThread[Flatten[{"Date", properties}] → #] &,
            Transpose@
             Insert[Transpose[QuantityMagnitude[mergedData][
                 "Values"]], dates, 1]
           ]
         ]
```

For example, we download return on assets (`"ROA"`), return on equity (`"ROE"`) and revenue growth (`"RevenueGrowth"`) for Apple Inc. (AAPL) over the period January 1, 2000, to January 1, 2019, and make a dataset. After the dataset is constructed, we can pull data and do further analysis using a rich set of built-in Wolfram knowledge.

```
In[10]:= getCompanyData["AAPL", {2000, 1, 1}, {2019, 1, 1},
          {"ROA", "ROE", "RevenueGrowth"}]
```

Out[10]=

Date	ROA	ROE	RevenueGrowth
Fri 31 Mar 2000	3.1933	5.5949	27.1242
Fri 30 Jun 2000	2.8696	4.9438	17.1374
Sat 30 Sep 2000	2.4754	4.2199	39.9701
Sun 31 Dec 2000	−3.0495	−5.0388	−57.0209
Sat 31 Mar 2001	0.7098	1.1548	−26.4267
Sat 30 Jun 2001	0.9999	1.6061	−19.1781
Sun 30 Sep 2001	1.0916	1.6971	−22.4599
Mon 31 Dec 2001	0.6259	0.9637	36.5442
Sun 31 Mar 2002	0.6459	1.0025	4.4724
Sun 30 Jun 2002	0.5097	0.7921	−3.1186
Mon 30 Sep 2002	−0.7148	−1.1028	−0.4828
Tue 31 Dec 2002	−0.1273	−0.1949	7.0545
Mon 31 Mar 2003	0.2217	0.3392	−1.3378
Mon 30 Jun 2003	0.2968	0.4558	8.1176
Tue 30 Sep 2003	0.6639	1.045	18.8496
Wed 31 Dec 2003	0.914	1.4732	36.2772
Wed 31 Mar 2004	0.6712	1.0431	29.4237
Wed 30 Jun 2004	0.8741	1.3115	30.356
Thu 30 Sep 2004	1.2571	1.9417	37.0262
Fri 31 Dec 2004	3.3885	5.4298	73.9781

rows 1–20 of **76**

3. Common Financial Decision Making, Capital Budgeting and Business Valuation

Basic concepts used in common financial decision making are important for learning and under standing the finance discipline. Many functions such as `EffectiveInterest`, `CashFlow`, `Annuity`, `AnnuityDue` and `TimeValue` are directly related to finance. Other functions such as `Solve` and `FindRoot` can be used to find one of the unknowns when relevant information is given. All these functions are useful for solving time value of money, capital begetting and business valuation problems. The Mathematica documentation provides numerous examples of how to use these functions. In this section, we are going to focus on a few examples concerning loan amortization, capital budgeting and business valuation.

A loan amortization table is often used to visualize periodic payments of the loan, loan balance and payment breakdown into principle payment and interest payment. The function `amortizationTable` returns an amortization table given its input arguments. It takes four arguments:

`pv`: current value of loan amount

`years`: loan term in years

`APR`: annual percentage interest rate

`frequency`: frequency of loan payment per year: 12 for monthly payment, 1 for annual payment, and so on

`fv`: (an optional argument) future value of the loan amount; if no value for `fv` is provided, the future value of the loan is assumed to be zero

```
In[11]:= amortizationTable[pv_, years_, APR_, frequency_, fv_ : 0] :=
          Module[
           {periods, rate, payment, balance, interest, principal},
           periods = years frequency;
           rate = APR / frequency;
           payment = (pv + (pv + fv) / ((1 + rate)^periods - 1)) rate;
           balance = TimeValue[Annuity[payment, periods], rate, 0];
           Text@Grid[
             Prepend[
              Round@Table[
                 interest = balance rate;
                 principal = payment - balance rate;
                 balance = balance - principal;
                 {month, balance + principal, payment, interest,
                  principal, balance},
                 {month, periods}
               ],
              {"Period", "Beginning Balance", "Payment",
               "Interest Payment", "Principal Payment",
               "Ending Balance"}
             ], Dividers → All]]
```

Using this function, we compute an amortization table for a loan of \$40,000 with 1-year loan term, 5% APR paid monthly.

In[12]:=
```
amortizationTable[40000, 1, 0.05, 12]
```

Out[12]=

Period	Beginning Balance	Payment	Interest Payment	Principal Payment	Ending Balance
1	40000	3424	167	3258	36742
2	36742	3424	153	3271	33471
3	33471	3424	139	3285	30186
4	30186	3424	126	3299	26888
5	26888	3424	112	3312	23576
6	23576	3424	98	3326	20249
7	20249	3424	84	3340	16910
8	16910	3424	70	3354	13556
9	13556	3424	56	3368	10188
10	10188	3424	42	3382	6806
11	6806	3424	28	3396	3410
12	3410	3424	14	3410	0

The most commonly used decision tools in capital budgeting are net present value (NPV), internal rate of return (IRR), modified internal rate of return (MIRR) and profitability index (PI). These are defined in terms of the cash flows CF_i, $i = 0, 1, 2, \ldots, n$, the discount rate r and the reinvestment rate rr by:

$$\mathrm{NPV} = \frac{\mathrm{CF}_1}{(1+r)} + \frac{\mathrm{CF}_2}{(1+r)^2} + \ldots + \frac{\mathrm{CF}_n}{(1+r)^n} + \mathrm{CF}_0,$$

$$\frac{\mathrm{CF}_1}{(1+\mathrm{IRR})} + \frac{\mathrm{CF}_2}{(1+\mathrm{IRR})^2} + \ldots + \frac{\mathrm{CF}_n}{(1+\mathrm{IRR})^n} + \mathrm{CF}_0 = 0,$$

$$\left(\mathrm{CF}_1(1+\mathrm{rr})^{n-1} + \mathrm{CF}_2(1+\mathrm{rr})^{n-2} + \ldots + \mathrm{CF}_{n-1}(1+\mathrm{rr}) + \mathrm{CF}_n\right) \Big/ (1+\mathrm{MIRR})^n = \mathrm{CF}_0,$$

$$\mathrm{PI} = \frac{\frac{\mathrm{CF}_1}{(1+r)} + \frac{\mathrm{CF}_2}{(1+r)^2} + \ldots + \frac{\mathrm{CF}_n}{(1+r)^n}}{\mathrm{CF}_0}.$$

We use the built-in Mathematica functions `TimeValue`, `FindRoot` and `Cashflow` in the `capitalBudgetingTools` function to compute these measures. It takes cash flows `CF` (a list), discount rate `r` and reinvestment rate `rr` as its arguments.

```
In[13]:= capitalBudgetingTools[CF_List, r_, rr_] :=
          {
           "NPV" → TimeValue[Cashflow[CF], r],
           First@FindRoot[TimeValue[Cashflow[CF], IRR] == 0,
             {IRR, 0.005}],
           First@
            FindRoot[
             TimeValue[TimeValue[Cashflow[Flatten[{0, CF[[2 ;;]]}]],
                 rr, Length[Rest[CF]]], MIRR, -Length[Rest[CF]]] ==
              -First[CF], {MIRR, 0.005}],
           "PI" → TimeValue[Cashflow[Flatten[{0, CF[[2 ;;]]}]], r] /
             Abs[First[CF]]
          }
```

We illustrate the use of the `capitalBudgetingTools` function with an example. Say a project requires a \$50,000 initial investment and is expected to produce, after tax, a cash flow of \$15,000, \$8,000, \$10,000, \$12,000, \$14,000 and \$16,000 over the next six years. The discount rate is 10% and the reinvestment rate is 11%. We compute the project's NPV, IRR, MIRR and PI.

```
In[14]:= capitalBudgetingTools[
          {-50 000, 15 000, 8000, 10 000, 12 000, 14 000, 16 000},
          0.1, 0.11]
```

```
Out[14]= {NPV → 3681.72, IRR → 0.123835, MIRR → 0.117586, PI → 1.07363}
```

One of the most widely used business valuation models is the discounted cash flow model, in which the value of any asset is obtained by discounting the expected cash flows on that asset at a rate that reflects its riskiness. In its most general form, the value of a company is the present value of the expected free cash flows the company can generate in perpetuity. Because we cannot estimate free cash flows (FCF) in perpetuity, we generally allow for a period where FCF can grow at extraordinary rates, but we allow for closure in the model by assuming that the growth rate will decline to a stable rate that can be sustained forever at some point in the future. If we assume that the discount rate is the weighted average cost of capital (WACC), FCF grows at the rate of g per year and that the last year's free cash flow is $\mathrm{FCF}_0 > 0$, then the value of the firm can be defined as

$$\mathrm{Value}_0 = \sum_{t=1}^{\infty} \frac{\mathrm{FCF}_0\,(1+g)}{(1+\mathrm{WACC})^t}.$$

If we assume that FCF grows at the rate g_1 for the next n_1 years and at the rate g_2 thereafter, then the value of the firm can be written as

$$\text{Value}_0 = \sum_{t=1}^{n_1} \frac{\text{FCF}_0\,(1+g_1)^{\,t}}{(1+\text{WACC})^t} + \frac{1}{(1+\text{WACC})^{n_1}} \times \frac{\text{FCF}_{n_1}\,(1+g_2)}{(\text{WACC}-g_2)}. \tag{1}$$

We implement formula (1) with `twoStageDCFModel` that takes five arguments:

1. `FCF`, last year's free cash flows
2. `g1`, the annual growth rate of free cash flows in the first growth period
3. `n1`, the number of years in the first growth period
4. `g2`, the stable growth rate
5. `wacc`, the weighted average cost of capital

```
In[15]:= twoStageDCFModel[FCF_, g1_, n1_, g2_, wacc_] :=
          Sum[FCF × (1 + g1)^t / (1 + wacc)^t, {t, 1, n1}] + FCF × (1 + g1)^n1 × (1 + g2) / ((1 + wacc)^n1 × (wacc - g2))
```

Suppose that a company has \$100 as its past year's CFC, its FCF is expected to grow at 5% for the next five years and 2.5% thereafter and that its WACC is 9.5%. We compute its value.

```
In[16]:= twoStageDCFModel[100, 0.05, 5, 0.025, 0.095]

Out[16]= 1628.77
```

The most practical approach is to assume that a company goes through three phases: growth, transition and maturity. First, a company's growth increases. In the transition phase, the growth rate decreases. In the mature phase, a company grows at the same rate as that of the overall economy.

Assume that the FCF is positive and grew at the rate g_0 last year. Assume further that it grows at the higher rate g_1 each year for the next n_1 years and that after n_1 years, it declines at the rate g_2 each year for the next n_2 years. Finally, assume the company grows at the stable positive rate g_3 per year after $n_1 + n_2$ years and that the cost of capital is represented by WACC. Then the value of the firm can be written as

$$\begin{aligned}\text{Value}_0 = &\sum_{t=1}^{n_1} \frac{\text{FCF}_{t-1}\,(1+g_0+t\times g_1)^{\,t}}{(1+\text{WACC})^t} + \\ &\sum_{t=n_1+1}^{n_1+n_2} \frac{\text{FCF}_{t-1}\,(1+g_0+n_1\times g_1+(t-n_1)\times g_2)^{\,(t-n_1)}}{(1+\text{WACC})^t} + \\ &\frac{1}{(1+\text{WACC})^{n_1+n_2}} \times \frac{\text{FCF}_{n_1+n_2}\,(1+g_3)}{(\text{WACC}-g_3)}\end{aligned} \tag{2}$$

We implement formula (2) with `threeStageDCFModel` that takes eight arguments:

1. `FCF`, last year's free cash flows
2. `g0`, last year's FCF growth rate
3. `g1`, the incremental growth rate in the high growth period
4. `n1`, the number of years in the high growth period
5. `g2`, the declining growth rate in the translational growth period
6. `n2`, the number of years in the translational growth period
7. `g3`, the stable growth rate in the maturity growth period
8. `wacc`, the weighted average cost of capital

```
In[17]:= threeStageDCFModel[FCF_, g0_, g1_, n1_, g2_, n2_, g3_,
           wacc_] := N@Module[
            {growthrates, cashflows},
            growthrates =
             1 + Flatten[{g0 + Range[n1] g1,
                 Last[g0 + Range[n1] g1] + Range[n2] g2}];
            cashflows = FCF FoldList[Times, growthrates];
            TimeValue[Cashflow[Flatten[{0, cashflows}]], wacc, 0] +
             Last[cashflows] (1 + g3) / (wacc - g3) / (1 + wacc) ^ (n1 + n2)
           ]
```

To apply the function, consider a company in the early stage of its life cycle, assuming that the company experienced 10 percent growth in the past year. The company is expected to grow by 8% more each year for the next 7 years and its growth will start to decline by 5% each year after the seventh year for 5 years. After 12 years, the company is expected to grow at the same rate as that of the overall economy, which is 2.5% per year. Suppose the past year's CFC was $100 million and the weighted average cost of capital is 9.5%. We compute the value of the company.

```
In[18]:= threeStageDCFModel[100, 0.1, 0.08, 7, -0.05, 7, 0.025,
          0.095]

Out[18]= 82 433.5
```

When using `threeStageDCFModel`, we assume that the growth rate in the stable phase is never negative. Therefore, do not assume the declining growth rate to be too high. Otherwise, the growth rate of FCF in the maturity phase may be negative.

4. Analysis of Stock Returns

There are various methods in analyzing historical stock prices and returns data. We can use different kinds of charts and graphs as well as descriptive statistics. Similarly, it is also important to understand whether the stock returns distribution is normal. In the next two subsections, we explain some of the most common charts and descriptive measures.

4.1 Individual Stock Return Analysis

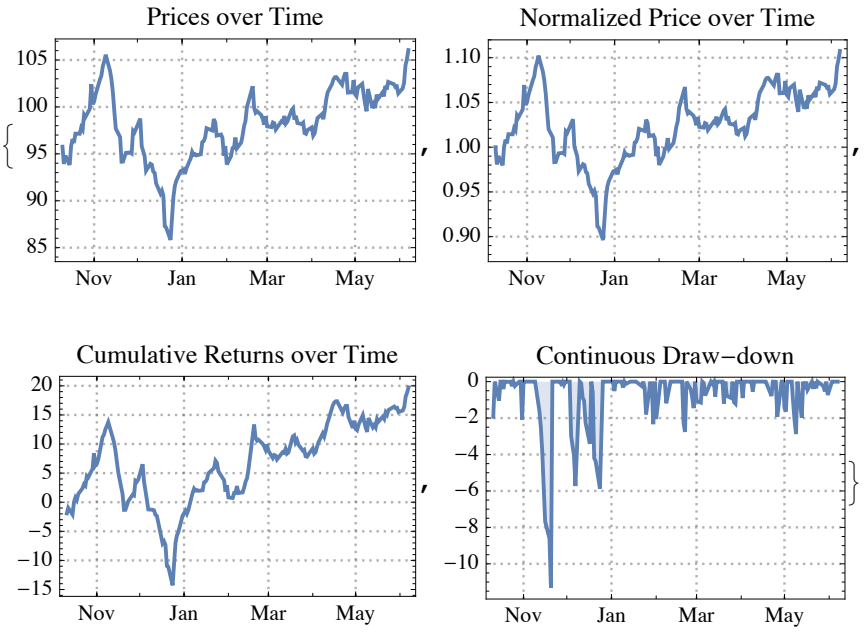
Commonly used charts for historical performance analysis are time series plot of prices, normalized prices (historical prices divided by the price at the beginning), continuous draw-downs (cumulative continuous returns) and cumulative returns. The function `performanceOfOneSymbol` takes four arguments as defined in Section 1 and returns four different plots: historical prices, the normalized price, continuous draw-downs and cumulative returns.

```
In[19]:= performanceOfOneSymbol[symbol_String, startDate_,
           endDate_, period_] := Module[
           {data, prices, dates, returns, timeseries, drawdowns,
            indices},
           data = FinancialData[symbol, "Close",
             {startDate, endDate, period}];
           prices = QuantityMagnitude[data["Values"]];
           dates = data["Dates"][[All, 1, {1, 2, 3}]];
           returns = 100 Differences[prices] / Most[prices];
           timeseries = TimeSeries[returns, {Rest[dates]}];
           drawdowns =
            TimeSeries[Flatten[Accumulate /@ SplitBy[returns, Sign]] /.
              _?Positive → 0, {dates[[2 ;;]]}];
           indices =
            TimeSeries[prices / Table[First[prices], Length[prices]],
             {dates}];
           {DateListPlot[data, PlotLabel → "Prices over Time",
             PlotTheme → "Detailed"],
            DateListPlot[indices,
             PlotLabel → "Normalized Price over Time",
             PlotTheme → "Detailed"],
            DateListPlot[Accumulate[timeseries],
             PlotLabel → "Cumulative Returns over Time",
             PlotTheme → "Detailed"],
            DateListPlot[drawdowns, PlotRange → All, Filling → Axis,
             PlotTheme → "Detailed",
             PlotLabel → "Continuous Draw-down"]}]
```

We can apply `performanceOfOneSymbol` to any symbol and period. For example, we plot the historical stock prices and returns of Walmart Inc. (WMT) over the period October 10, 2018, to June 7, 2019.

```
In[20]:= performanceOfOneSymbol["WMT", {2018, 10, 10}, {2019, 6, 7},
          "Day"]
```

Out[20]=



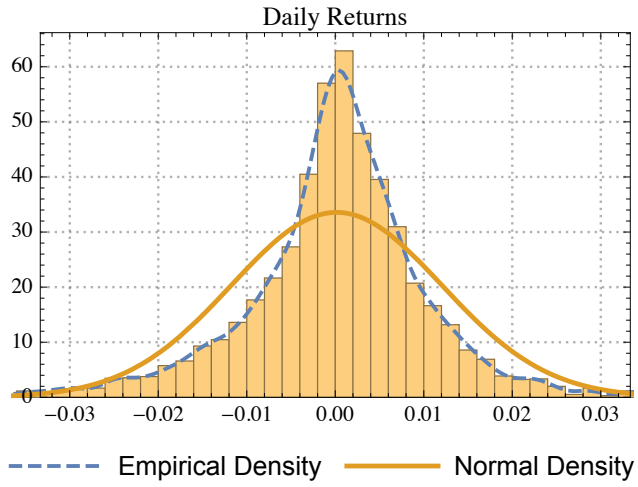
A histogram and an empirical plot of kernel density estimates are often used to describe the general shape of the data. The `densityPlot` function takes four arguments as defined in Section 1 and returns density and histogram plots of returns.

```
In[21]:= densityPlot[symbol_String, startDate_, endDate_,
          period_] := Module[
          {prices, returns, x},
          prices = FinancialData[symbol, "Close",
            {startDate, endDate, period}];
          returns =
           Differences[Log[QuantityMagnitude[prices]["Values"]]];
          Show[
           Histogram[returns, Automatic, "PDF",
            PlotLabel → "Daily Returns", PlotTheme → "Detailed"],
           Plot[{
             PDF[SmoothKernelDistribution[returns], x],
             PDF[NormalDistribution[Mean[returns],
               StandardDeviation[returns]], x]},
            {x, Min[returns], Max[returns]}, PlotRange → All,
            PlotLegends →
             Placed[{"Empirical Density", "Normal Density"}, Below],
            PlotStyle → {Dashed, Thick}]
          ]
         ]
```

Using the `densityPlot` function, we download the daily closing price of the S&P 500 index over the period October 1, 2000, to October 1, 2019, and plot the histogram, the empirical density function and the density function of a normal distribution with the same mean and variance.

```
In[22]:= densityPlot["^SPX", {2000, 10, 1}, {2019, 10, 1},
          "Day"]
```

Out[22]=



Many built-in functions can be used to compute different descriptive statistics. These descriptive statistics describe properties of distributions, such as location, dispersion and shape. The most common measures are computed by `performanceStatsOneSymbol`:

- holding period return
- average return
- geometric mean return
- cumulative returns
- standard deviation
- minimum return
- maximum return
- skewness
- kurtosis
- historical value at risk
- historical conditional value at risk

```
In[23]:= performanceStatsOneSymbol[symbol_String, startDate_,
           endDate_, period_] := Module[
           {data, prices, dates, returns, var, cvar},
           data = FinancialData[symbol, "Close",
             {startDate, endDate, period}];
           prices = QuantityMagnitude[data["Values"]];
           dates = data["Dates"][[All, 1, {1, 2, 3}]];
           returns = Differences[prices] / Most[prices];
           var = Sort[returns][[Floor[Length[returns] 0.05]]];
           cvar = Mean[Select[returns, # ≤ var &]];
           Apply[#1 → #2 &,
            Transpose[{
              {
               "Symbol",
               "Start Date",
               "End Date",
               "Holding Period Return",
               "Arithmetic Mean",
               "Geometric Mean",
               "Cumulative Return",
               "Stdandard Deviation",
               "Minimum Return",
               "Maximum Return",
               "Skewness",
               "Kurtosis",
               "Value at Risk (VaR)",
               "Conditional VaR (CVaR)"
              },
              {
               symbol,
               FromDigits[First[dates][[1 ;; 2]]],
               FromDigits[Last[dates][[1 ;; 2]]],
               100 (Last[prices] / First[prices] - 1),
               100 Mean[returns],
               100 (GeometricMean[1 + returns] - 1),
               100 Total[returns],
               100 StandardDeviation[returns],
               100 Min[returns],
               100 Max[returns],
               Skewness[returns],
               Kurtosis[returns],
               100 var,
               100 cvar
              }
             }], {1}]
          ]
```

For example, we compute the descriptive statistics for Walmart Inc. (WMT) using monthly returns over the period October 10, 2010, to June 6, 2019.

```
In[24]:= performanceStatsOneSymbol["WMT", {2010, 10, 10},
           {2019, 6, 6}, "Month"] // Column // Text
```

```
Out[24]= Symbol → WMT
         Start Date → 20 110
         End Date → 20 196
         Holding Period Return → 94.0373
         Arithmetic Mean → 0.757671
         Geometric Mean → 0.63942
         Cumulative Return → 78.7978
         Stdandard Deviation → 4.88944
         Minimum Return → −15.5628
         Maximum Return → 14.7765
         Skewness → −0.0693292
         Kurtosis → 3.98949
         Value at Risk (VaR) → −6.69229
         Conditional VaR (CVaR) → −10.2686
```

It is also informative to examine the historical performance of an individual stock. In such an analysis, we calculate monthly statistics using daily returns and report them on a monthly basis. We define the `monthlyOneStockStatistics` function to download historical stock prices, compute desired statistics and return a dataset. The function takes four arguments: stock ticker symbol, start date, end date and a statistical function such as `Mean`, `StandardDeviation` or `Total`, and returns a dataset. For the function to work, it requires more than two years of data.

```
In[25]:= monthlyOneStockStatitics[symbol_String, startData_,
           endDate_, statFunction_] := Module[
           {data, prices, returns, dates, years, months,
            timeseries, results, firstRow, annualReturns,
            midRows, lastRow, heads},
           data = FinancialData[symbol, {startData, endDate}];
           prices = QuantityMagnitude[data["Values"]];
           returns = 100 Differences[prices] / Most[prices];
           dates = data["Dates"][[All, 1, {1, 2, 3}]];
           years = DeleteDuplicates[
             DateString[#, "Year"] & /@ dates[[2 ;;]]];
           months = dates[[2 ;;]][[All, 2]];
           timeseries =
            Transpose@
             {DateString[#, "Year"] <> DateString[#, "Month"] & /@
                dates[[2 ;;]], returns};
           results = statFunction /@ GatherBy[timeseries, First];
           firstRow = Append[
             Join[
              Table["NA", First[months] - 1],
```

```
    results[[1 ;; 13 - First[months], 2]]
   ],
   statFunction[results[[1 ;; 13 - First[months], 2]]]
  ];
 annualReturns =
  statFunction /@
   Partition[
    results[[(13 - First[months] + 1) ;; -(Last[months] + 1),
     2]], 12];
 midRows = Transpose@Append[
    Transpose[
     Partition[results[[(13 - First[months] + 1) ;;
         -(Last[months] + 1), 2]], 12]],
    annualReturns
   ];
 lastRow = Append[
   Join[
    results[[-Last[months] ;;, 2]],
    Table["NA", 12 - Last[months]]
   ],
   statFunction[results[[-Last[months] ;;, 2]]]
  ];
 heads = {"Year", "January", "February", "March",
   "April", "May", "June", "July", "August", "September",
   "Octber", "November", "December", "Annual"};
 Dataset[
  AssociationThread[heads → #] & /@
   Transpose[
    Insert[Transpose@Append[Prepend[midRows, firstRow],
       lastRow], years, 1]]
 ]
]
```

For example, we compute the monthly cumulative returns for Walmart Inc. (WMT) using daily returns over the period October 1, 2010, to June 30, 2019.

```
In[26]:= monthlystatistics = monthlyOneStockStatitics["WMT",
    {2010, 10, 1}, {2019, 06, 30}, Total];
```

Once we compute the statistics, we can take specific columns by specifying their names.

In[27]:=
```
monthlystatistics[All,
 {"Year", "January", "February", "March", "Annual"}]
```

Out[27]=

Year	January	February	March	Annual
2010	NA	NA	NA	1.19228
2011	3.97911	−7.44722	0.176706	11.6715
2012	2.67958	−3.65275	3.56093	14.5859
2013	2.53261	1.26538	5.62873	15.0285
2014	−5.19207	0.1267	2.34163	9.62153
2015	−0.842104	−1.13443	−1.88485	−31.5297
2016	8.23019	0.132147	3.24908	13.8512
2017	−3.41632	6.21654	1.64689	37.3049
2018	7.75837	−16.0842	−0.983156	−2.9616
2019	2.91088	3.36999	−1.43949	17.5397

We are often interested in knowing whether returns data follows a normal distribution because understanding whether stock returns are normal or not is very important in investment management. One way to check whether returns are normally distributed or not is to compare the empirical quantiles of the data with normal distribution. The `QuantilePlot` function can be used to produce quantile-quantile plots. Many other built-in functions can help to assess whether returns are normally distributed.

The `DistributionFitTest` function can be used to test whether data is normally distributed and can also be used to assess the goodness of fit of data to any distribution.

4.2 Analysis of Multiple Stocks Returns

As a majority of financial data is multivariate, it is advantageous to perform comparative analysis of multiple security returns. In some cases, one has to compare one series with another. In other cases, many variables might have to be simultaneously measured to capture the complex nature of the relationship among variables. Comparing the complexities of these factors gives the analyst a more detailed account of the relationships between selected returns, thus allowing for a better interpretation of their values and behaviors. In this section, we first compare the performance of one asset with another using graphs, then compute descriptive statistics as well as correlation matrices.

Two most commonly used graphs for comparing historical performance of more than one stock/ETF are time series plots of normalized prices and cumulative returns. The next two functions take four arguments as defined in Section 1 and compute normalized prices and cumulative returns.

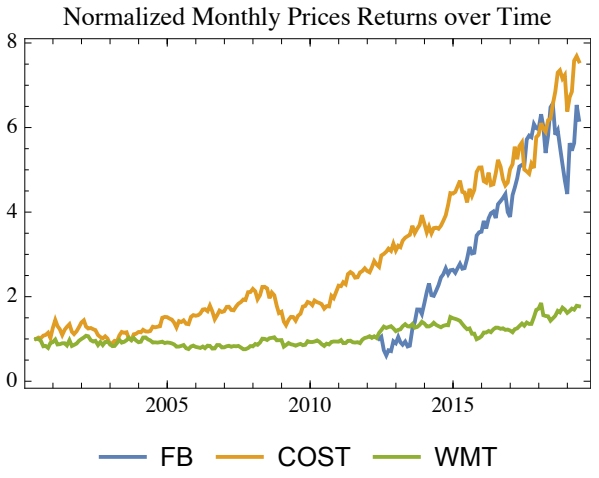
```
In[28]:= normalizedPrices[symbols : {__String}, startDate_,
           endDate_, period_] := Module[
           {data, mergedData, prices, dates, pricesTimeSeries,
            pruneprices, stock, index, result, series, newprices,
            indices, timeseries},
           data = FinancialData[symbols, "Close",
             {startDate, endDate, period}];
           mergedData = TimeSeriesThread[# &, data,
             ResamplingMethod → "NaN"];
           prices = QuantityMagnitude@mergedData["Values"];
           dates = mergedData["Dates"][[All, 1, {1, 2, 3}]];
           pricesTimeSeries = MapThread[Flatten[{{#1}, #2}, 1] &,
             {dates, prices}];
           pruneprices = Select[pricesTimeSeries[[All, {1, 2}]],
             ! MemberQ[#, "NaN"] &];
           stock = pruneprices[[All, 2]];
           index = stock / Table[First[stock], Length[stock]];
           result = MapThread[#1 → {#2} &,
             {pruneprices[[All, 1]], index}];
           Do[
            series = Select[pricesTimeSeries[[All, {1, i + 1}]],
              ! MemberQ[#, "NaN"] &];
            newprices = series[[All, 2]];
            indices =
             newprices / Table[First[newprices], Length[newprices]];
            timeseries = MapThread[#1 → {#2} &,
              {series[[All, 1]], indices}];
            result = Merge[KeyUnion[{result, timeseries}, Missing[] &],
              Identity];,
            {i, 2, Length[symbols]}
           ];
           TimeSeries[Flatten[#] & /@ Values[result], {Keys[result]}]
          ]
```

```
In[29]:= cumlativeReturns[symbols : {__String}, startDate_,
           endDate_, period_] := Module[
           {data, mergedData, prices, dates, pricesTimeSeries,
            pruneprices, stock, index, result, series, newprices,
            indices, timeseries},
           data = FinancialData[symbols, "Close",
             {startDate, endDate, period}];
           mergedData = TimeSeriesThread[# &, data,
             ResamplingMethod → "NaN"];
           prices = QuantityMagnitude@mergedData["Values"];
           dates = mergedData["Dates"][[All, 1, {1, 2, 3}]];
           pricesTimeSeries = MapThread[Flatten[{{#1}, #2}, 1] &,
             {dates, prices}];
           pruneprices = Select[pricesTimeSeries[[All, {1, 2}]],
             ! MemberQ[#, "NaN"] &];
           stock = pruneprices[[All, 2]];
           index = 100 Accumulate[Differences[stock] / Most[stock]];
           result = MapThread[#1 → {#2} &,
             {pruneprices[[2 ;;, 1]], index}];
           Do[
            series = Select[pricesTimeSeries[[All, {1, i + 1}]],
              ! MemberQ[#, "NaN"] &];
            newprices = series[[All, 2]];
            indices =
             100 Accumulate[Differences[newprices] /
                Most[newprices]];
            timeseries = MapThread[#1 → {#2} &,
              {series[[2 ;;, 1]], indices}];
            result = Merge[KeyUnion[{result, timeseries}, Missing[] &],
              Identity];,
            {i, 2, Length[symbols]}
           ];
           TimeSeries[Flatten[#] & /@ Values[result], {Keys[result]}]
          ]
```

We get normalized prices and cumulative returns and plot them for three stocks (Facebook, Inc. (FB), Costco Wholesale Corporation (COST) and Walmart Inc. (WMT)) over the period May 1, 2000, to May 30, 2019.

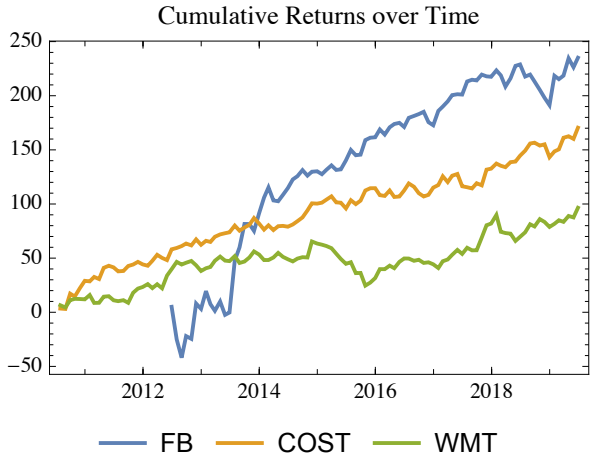
In[30]:=
```
DateListPlot[
 normalizedPrices[{"FB", "COST", "WMT"}, {2000, 5, 1},
  {2019, 5, 30}, "Month"],
 PlotLabel → "Normalized Monthly Prices Returns over Time",
 PlotRange → All,
 PlotLegends → Placed[{"FB", "COST", "WMT"}, Below]
]
```

Out[30]=



In[31]:=
```
DateListPlot[cumlativeReturns[{"FB", "COST", "WMT"},
  {2010, 06, 10}, {2019, 6, 30}, "Month"],
 PlotLabel → "Cumulative Returns over Time",
 PlotRange → All,
 PlotLegends → Placed[{"FB", "COST", "WMT"}, Below]
]
```

Out[31]=



Besides graphs, we can also compute descriptive statistics and compare their performance. We define a function `performanceStatisticsTable` for that purpose. It takes four arguments as defined in Section 1 and returns a table with different types of descriptive statistics.

```
In[32]:= performanceStatisticsTable[symbols : {__String},
           startDate_, endDate_, period_] := Module[
           {head, n, stats, data, prices, dates, returns, var, cvar},
           head = {"Start Date", "End Date", "Holding Period Return",
             "Arithmetic Mean", "Geometric Mean",
             "Cumulative Return", "Stdandard Deviation",
             "Minimum Return", "Maximum Return", "Skewness",
             "Kurtosis", "Value at Risk (VaR)",
             "Conditional VaR (CVaR)"};
           n = Length[symbols];
           stats = ConstantArray[Missing[], {n, Length[head]}];
           Do[
            data = FinancialData[symbols[[i]],
              {startDate, endDate, period}];
            prices = QuantityMagnitude@data["Values"];
            dates = data["Dates"][[All, 1, {1, 2, 3}]];
            returns = Differences[prices] / Most[prices];
            var = Sort[returns][[Floor[Length[returns] 0.05]]];
            cvar = Mean@Select[returns, # ≤ var &];
            stats[[i]] =
             {DateString[First[dates], "Year"] <>
               DateString[First[dates], "Month"],
              DateString[Last[dates], "Year"] <>
               DateString[Last[dates], "Month"],
              100 (Last[prices] / First[prices] - 1), 100 Mean[returns],
              100 (GeometricMean[1 + returns] - 1), 100 Total[returns],
              100 StandardDeviation[returns], 100 Min[returns],
              100 Max[returns], Skewness[returns], Kurtosis[returns],
              100 var, 100 cvar},
            {i, n}
           ];
           TableForm[Transpose[stats],
            TableHeadings → {head, symbols}]
          ]
```

For example, we download historical data and compute different descriptive statistics for three stocks (Walmart Inc. (WMT), Apple Inc. (AAPL) and Microsoft Corporation (MSFT)) using monthly data over the period January 1, 2010, to March 30, 2019.

```
In[33]:= performanceStatisticsTable[{"WMT", "AAPL", "MSFT"},
           {2010, 1, 1}, {2019, 3, 30}, "Month"] // Text
```

Out[33]=

	WMT	AAPL	MSFT
Start Date	201001	201001	201001
End Date	201903	201903	201903
Holding Period Return	82.5379	592.298	318.524
Arithmetic Mean	0.666611	2.03964	1.49796
Geometric Mean	0.548579	1.77451	1.30993
Cumulative Return	73.3272	224.36	164.776
Stdandard Deviation	4.88488	7.34558	6.20102
Minimum Return	−15.5628	−18.4045	−15.5068
Maximum Return	14.7765	19.6227	19.6409
Skewness	−0.0303605	−0.108899	0.0981689
Kurtosis	3.8595	3.03052	3.64059
Value at Risk (VaR)	−6.69229	−11.6698	−9.08562
Conditional VaR (CVaR)	−10.2686	−14.0612	−11.6272

Similarly to how we calculated an individual stock's monthly statistics in Section 4.1, we define the `monthlyStatisticsTable` function to compute monthly statistics for more than one stock given the arguments: stock ticker symbols, start date, end date and a statistical function such as `Mean`, `StandardDeviation` or `Total`.

```
In[34]:= statistics[symbol_String, startDate_, endDate_, function_] :=
          Module[
            {data, prices, returns, dates, timeseries},
            data = FinancialData[symbol, {startDate, endDate}];
            prices = QuantityMagnitude@data["Values"];
            returns = 100 Differences[prices] / Most[prices];
            dates = data["Dates"][[All, 1, {1, 2, 3}]];
            timeseries = Transpose[{
                DateString[#, "Year"] <> DateString[#, "Month"] & /@
                 dates[[2 ;;]], returns
               }];
            Transpose[{
              DeleteDuplicates[
               DateString[#, "Year"] <> DateString[#, "Month"] & /@
                dates[[2 ;;]]],
              (function /@ GatherBy[timeseries, First])[[All, 2]]
             }]
          ]
```

```
In[35]:= mergeData[a_List, b_List, pad_] := Module[
            {rules, keys},
            rules = Apply[# → {##2} &, {a, b}, {2}];
            keys = Union @@ Keys@rules;
            Join[List /@ keys, ##, 2] & @@
             (Lookup[#, keys, pad & /@ #[[1, 2]]] & /@ rules)
          ]
```

```
In[36]:= monthlyStatisticsTable[symbols : {__String}, startDate_,
           endDate_, function_] := Module[
           {table, heads = Flatten[{"Date", symbols}]},
           table = statistics[symbols[[1]], startDate, endDate,
             function];
           Do[
            table = mergeData[table,
               statistics[symbols[[i]], startDate, endDate,
                function], Missing[]];, {i, 2, Length[symbols]}
           ];
           Dataset[AssociationThread[heads → #] & /@ table]
          ]
```

For example, we compute the monthly cumulative returns for four stocks (Walmart Inc. (WMT), Apple Inc. (AAPL), Microsoft Corporation (MSFT) and Netflix, Inc. (NFLX)) over the period January 1, 2010, to June 30, 2019, and create a dataset. The first column represents year and month, the first four digits for the year and the last two digits for the month.

```
In[37]:= monthlyStatisticsTable[{"WMT", "AAPL", "MSFT", "NFLX"},
          {2010, 01, 1}, {2019, 06, 30}, Total]
```

Out[37]=

Date	WMT	AAPL	MSFT	NFLX
201001	−1.39345	−10.314	−9.14998	18.0942
201002	1.26554	6.52787	1.82623	6.17777
201003	2.85732	14.0356	2.20712	11.3768
201004	−3.57067	10.9046	4.25119	31.7034
201005	−5.7373	−0.830821	−16.3803	13.6443
201006	−4.89116	−1.70627	−11.0029	−1.38915
201007	6.39735	2.47334	11.6865	−3.75625
201008	−1.98091	−5.47658	−9.39433	21.4608
201009	6.58351	15.6809	4.50437	26.6364
201010	1.26006	6.13889	8.68883	8.15568
201011	−0.108804	3.53417	−5.31245	17.8957
201012	−0.257928	3.6372	10.1303	−15.1057
201101	3.97911	5.26971	−0.477127	21.1616
201102	−7.44722	4.185	−4.13941	−2.69776
201103	0.176706	−1.05154	−4.45235	14.9208
201104	5.51337	0.592624	2.21658	−1.39466
201105	0.469451	−0.520427	−3.48261	15.5764
201106	−3.7635	−3.33439	4.10312	−2.69276
201107	−0.764137	15.3551	5.43973	2.00245
201108	1.3814	−0.634705	−2.21054	−10.853

rows 1–20 of **114**

Similarly, box-and-whisker charts, paired histograms, paired smooth histograms and matrix scatterplots are often used to examine multivariate data. The `BoxWhiskerChart` function can be used to make a box plot that gives a glimpse of the distribution of the given dataset. You can see the statistical information by hovering over the boxes in the plot. The `PairedSmoothHistogram` and `SmoothKernelDistribution` functions are used to create paired histogram and smooth distribution plots. They can be used to compare how two datasets are distributed. The `PairwiseScatterPlot` function from the Statistical Plots package can be used to make scatter plots of multivariate data. It creates scatter plots comparing the data in each column against other columns. More complex analysis of multivariate data can be done using functions from the Multivariate Statistics package. The package contains functions to compute descriptive statistics for multivariate data and distributions derived from the multivariate normal distribution. All these functions are well explained in the official documentation.

5. Rolling-Window Performance Analysis

Rolling-window performance analysis is a simple technique to access variability of the statistical performance measures. For example, if we want to access the stability of mean or standard deviation of returns on a stock over time, we can choose rolling window (the number of consecutive observations per rolling window), estimate the mean or standard deviation and plot series of the estimates. A little fluctuation is normal but large fluctuations indicate a shift in the values of the estimate. Built-in functions such as `MovingAverage`, `MovingMap` and `TimeSeriesAggregate` are useful for rolling-window performance analysis. In this section, we are going to show a few examples of how to compute rolling-window-based performance statistics.

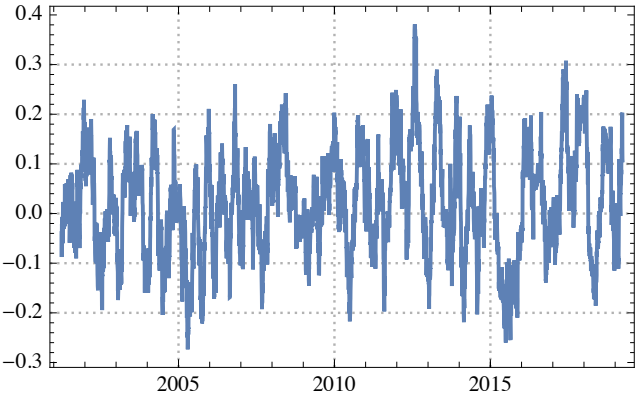
We define `plotRollingWindowStatistics` that can be used to plot rolling-window statistics given its five inputs: stock ticker symbol, start date, end date, size of window in days and function to apply, which can be any built-in or user-defined function.

```
In[38]:= plotRollingWindowStatistics[symbol_String, startDate_,
          endDate_, windowInDays_, function_] :=
         DateListPlot[
          MovingMap[
           function,
           FinancialData[symbol, "FractionalChange",
            {startDate, endDate}],
           Quantity[windowInDays, "Days"]
          ],
          PlotTheme → "Detailed"
         ]
```

For example, we compute and plot the 90-day rolling mean to standard deviation ratio on Walmart's daily stock returns over the period January 1, 2001, to March 30, 2019.

```
In[39]:= plotRollingWindowStatistics["WMT", {2001, 01, 01},
          {2019, 03, 30}, 90, Mean[#] / StandardDeviation[#] &]
```

Out[39]=



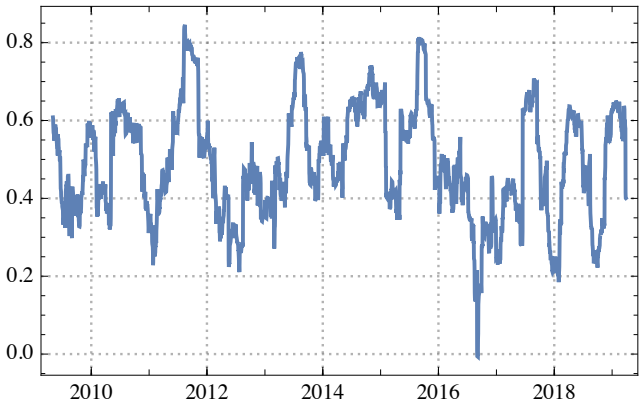
Similarly, we define the function `plotRollingCorrelation` to compute the rolling correlation of two series and apply it together with `MovingMap` to the desired data.

```
In[40]:= plotRollingCorrelation[{symbol1_String, symbol2_String},
           startDate_, endDate_, windowInDays_] := Module[
           {returns, correlationFunction},
           returns = TimeSeriesThread[# &,
             FinancialData[{symbol1, symbol2}, "FractionalChange",
              {startDate, endDate}], ResamplingMethod → Missing[]];
           correlationFunction =
            N[Apply[Correlation, Transpose[#]]] &;
           DateListPlot[MovingMap[correlationFunction, returns,
             Quantity[windowInDays, "Days"]], PlotRange → All,
            PlotTheme → "Detailed"]
         ]
```

We use the `plotRollingCorrelation` function to plot the 90-day rolling correlation of daily returns on two stocks, WMT and COST, for the period from March 30, 2009, to March 30, 2019.

```
In[41]:= plotRollingCorrelation[{"WMT", "COST"}, {2009, 01, 30},
          {2019, 03, 30}, 90]
```

Out[41]=



Sometimes, it is also useful to store these time-varying descriptive statistics as a dataset so that we can use them in the subsequent analysis. The function `rollingStatisticsTable`, given its input, computes the geometric mean, standard deviation and the ratio of the arithmetic mean to the standard deviation on a rolling-window basis.

```
In[42]:= rollingStatisticsTable[symbol_String, startDate_,
           endDate_, windowInDays_] := Module[
           {statisticsToCompute, statisticsComputed,
            statisticsValues, dates},
           statisticsToCompute =
            Flatten[{100 (GeometricMean[1 + #] - 1),
               StandardDeviation[#],
               Mean[#] / StandardDeviation[#]}] &;
           statisticsComputed = MovingMap[
             statisticsToCompute,
             FinancialData[symbol, "FractionalChange",
              {startDate, endDate}],
             Quantity[windowInDays, "Days"]
            ];
           statisticsValues = QuantityMagnitude[statisticsComputed][
             "Values"];
           dates = DateObject[#] & /@
             statisticsComputed["Dates"][[All, 1, {1, 2, 3}]];
           Dataset[
            AssociationThread[
               {"Date", "GM", "Std. Dev.", "AM / Std. Dev."} → #] & /@
             Transpose@Insert[Transpose[statisticsValues],
               dates, 1]]
         ]
```

For example, we compute the 90-day rolling-window geometric mean (GM), standard deviation (Std. Dev.) and arithmetic mean to standard deviation ratio (AM/Std. Dev.) of Walmart's daily stock returns over the period July 1, 2018, to October 30, 2019. You can scroll through the dataset.

In[43]:=
```
rollingStatisticsTable["WMT", {2018, 07, 01},
 {2019, 10, 30}, 90]
```

Out[43]=

Date	GM	Std. Dev.	AM / Std. Dev.
Mon 1 Oct 2018	0.185448	1.38612	0.140328
Tue 2 Oct 2018	0.189725	1.38758	0.143277
Wed 3 Oct 2018	0.168889	1.38658	0.128349
Thu 4 Oct 2018	0.168809	1.38658	0.128291
Fri 5 Oct 2018	0.154897	1.39349	0.117739
Mon 8 Oct 2018	0.151795	1.39024	0.115752
Tue 9 Oct 2018	0.167666	1.41178	0.125435
Wed 10 Oct 2018	0.158491	1.41986	0.118339
Thu 11 Oct 2018	0.128313	1.44356	0.0957225
Fri 12 Oct 2018	0.121876	1.43888	0.0915139
Mon 15 Oct 2018	0.106526	1.44619	0.0805091
Tue 16 Oct 2018	0.129574	1.46648	0.0953084
Wed 17 Oct 2018	0.143904	1.46823	0.104973
Thu 18 Oct 2018	0.143803	1.46827	0.104901
Fri 19 Oct 2018	0.153614	1.47198	0.11134
Mon 22 Oct 2018	0.161114	1.46983	0.116584
Tue 23 Oct 2018	0.165829	1.47099	0.119709
Wed 24 Oct 2018	0.163051	1.47162	0.117776
Thu 25 Oct 2018	0.182963	1.48314	0.1304
Fri 26 Oct 2018	0.180946	1.48365	0.129

rows 1–20 of **272**

6. Portfolio Optimization

Mean-variance analysis is one of the foundations of financial economics. Portfolio optimization is essential, whether it be in professional or personal financial planning. In this section, we are going to show how to implement the most commonly used optimization techniques in finance using historical returns. We want to point out that future returns on investment depend on expected returns and other conditioning information, not on the past returns. Past returns are used only for illustration and do not guarantee future returns.

Define the following variables:

- rf is the risk-free rate
- w_i is the proportion of wealth invested in security i
- μ_i is the average return on security i
- σ_i^2 is the variance of security i
- σ_{ij} is the covariance between securities i and j
- ρ_{ij} is the correlation between securities i and j

Then we define the vectors of mean returns and weights and the covariance matrix:

$$\mu = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_n \end{pmatrix}, \ \mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_2 \end{pmatrix} \text{and } \Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \dots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \dots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \dots & \sigma_{nn} \end{pmatrix}.$$

The formulas for the portfolio mean and variance are $\mu_p = \mu \cdot w = \sum_{i=1}^{n} \mu_i w_i$ and $\sigma_p^2 = w \cdot \Sigma \cdot w = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_{ij}$, respectively. The corresponding Mathematica code is `μ.w` and `w.Σ.w`.

In order to compute portfolio statistics, we need returns data. We can use the `getReturnsData` function to download historical returns data. It takes four arguments as defined in Section 1 and gives a matrix of returns. Most functions in this section use the `getReturnsData` function, so run it before you run other functions.

```
In[44]:= getReturnsData[symbols : {__String}, startDate_, endDate_,
           period_] := Module[
           {data, mergedData, prices, pruneprices},
           data = FinancialData[#, {startDate, endDate, period}] & /@
             symbols;
           mergedData = TimeSeriesThread[# &, data,
             ResamplingMethod → Missing[]];
           prices = QuantityMagnitude[mergedData["Values"]];
           pruneprices =
            Select[prices,
             ! MemberQ[#, QuantityMagnitude[Missing[]]] &];
           Differences[pruneprices] / Most[pruneprices]
          ]
```

To compute basic portfolio statistics such as portfolio mean, variance, standard deviation and Sharpe ratio, we can use `portfolioStatistics`, which takes six arguments. The first four arguments are as defined in Section 1 and the other two are a list of weights and the optional risk-free rate.

```
In[45]:= portfolioStatistics[symbols : {__String}, startDate_,
           endDate_, period_, weights_List, riskfreeRate_ : 0] :=
         Module[
          {returns = getReturnsData[symbols, startDate, endDate,
             period], portfolioMean, portfolioVariance},
          portfolioMean = weights.Mean@returns;
          portfolioVariance = weights.Covariance@returns.weights;
          TableForm[{
            {
             "Portfolio\nMean",
             "Portfolio\nVariance",
             "Portfolio\nStd. Dev.",
             "Sharpe\nRatio"
            },
            {
             portfolioMean,
             portfolioVariance,
             Sqrt[portfolioVariance],
             (portfolioMean - riskfreeRate) /
              Sqrt[portfolioVariance]
            }
           }, TableAlignments → Center]
         ]
```

For example, we compute the portfolio mean, variance, standard deviation and Sharpe ratio for the portfolio that consists of the stock returns of five companies: Apple (AAPL), Walmart (WMT), Boeing (BA), 3M and Exxon Mobil (XOM), using monthly returns over the period January 1, 2009, to May 30, 2019.

```
In[46]:= portfolioStatistics[{"AAPL", "WMT", "BA", "MMM", "XOM"},
           {2009, 1, 1}, {2019, 5, 30}, "Month",
           {0.2, 0.2, 0.2, 0.2, 0.2}, 0.0023] // Text
```

Out[46]=

Portfolio Mean	Portfolio Variance	Portfolio Std. Dev.	Sharpe Ratio
0.0125651	0.00166863	0.0408488	0.251295

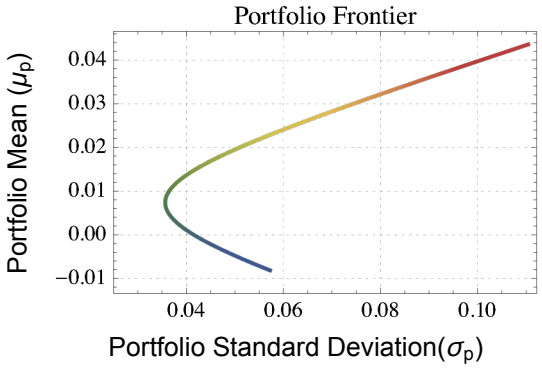
The `plotFrontier` function plots the Markowitz portfolio frontier; it takes a matrix of returns obtained from `getReturnsData` as its only argument. The function uses the concept that any two efficient portfolios are enough to establish the whole portfolio frontier, as first proved by Black [5]. It accepts any `ListPlot` option.

```
In[47]:= plotFrontier[returns_List,
           options : OptionsPattern[ListPlot]] := Module[
           {μ, σ, ones, wt1, wt2, w1, means, standardDeviations},
           μ = Mean[returns];
           σ = Covariance[returns];
           ones = ConstantArray[1, Length[μ]];
           wt1 = Inverse[σ].ones / ones.Inverse[σ].ones;
           wt2 = Inverse[σ].μ / ones.Inverse[σ].μ;
           means = w1 wt1.μ + (1 - w1) wt2.μ /.
             w1 → Range[-0.75, 1.75, 0.01];
           standardDeviations =
            Sqrt[w1^2 wt1.σ.wt1 + (1 - w1)^2 wt2.σ.wt2 +
                2 w1 (1 - w1) wt2.σ.wt1] /.
             w1 → Range[-0.75, 1.75, 0.01];
           Labeled[
            ListLinePlot[Transpose[{standardDeviations, means}],
             PlotTheme → "Detailed",
             AxesOrigin → {Min[standardDeviations] 0.75,
               Min[means] 1.5}, options],
            Text /@ {"Portfolio Standard Deviation(σp)",
              "Portfolio Mean (μp)"},
            {Bottom, Left}, RotateLabel → True
           ]
         ]
```

For example, we plot the portfolio frontier for the portfolio that consists of stock returns of five companies: Apple (AAPL), Walmart (WMT), Boeing (BA), 3M and Exxon Mobil (XOM), using monthly returns over the period January 1, 2009, to May 30, 2019.

```
In[48]:= plotFrontier[
          getReturnsData[{"AAPL", "WMT", "BA", "MMM", "XOM"},
           {2009, 1, 1}, {2019, 5, 30}, "Month"],
          PlotLabel → "Portfolio Frontier",
          ColorFunction → "DarkRainbow"]
```

Out[48]=



Next, we solve two kinds of portfolio problems: global minimum variance portfolio and tangency portfolio. In terms of the notation defined earlier in this section, the global minimum variance portfolio can be obtained by minimizing σ_p^2 subject to $\sum_{i=1}^{n} w_i = 1$ and solving for w_i. Its solution can be obtained with the built-in Mathematica function `NMinimize`.

The function `globalMininimumVariancePortfolioWeights` computes weights, returning the portfolio allocation on stocks considered for a global minimum variance portfolio.

```
In[49]:= globalMininimumVariancePortfolioWeights[symbols : {__String},
           startDate_, endDate_, period_] := Module[
           {returns, weights},
           returns = getReturnsData[symbols, startDate, endDate,
             period];
           weights = Subscript[w, #] & /@
             Range[Dimensions[returns][[2]]];
           weights /. NMinimize[
              {
               weights.(12 Covariance@returns).weights,
               Total[weights] == 1
              },
              weights][[2]
            ]
         ]
```

We compute the global minimum variance portfolio weights using the monthly stock returns of five companies: Apple (AAPL), Walmart (WMT), Boeing (BA), 3M and Exxon Mobil (XOM), over the period January 1, 2009, to May 30, 2019.

```
In[50]:= globalMininimumVariancePortfolioWeights[
          {"AAPL", "WMT", "BA", "MMM", "XOM"}, {2009, 1, 1},
          {2019, 5, 30}, "Month"]

Out[50]= {0.101357, 0.431649, 0.0305752, 0.081635, 0.354784}
```

Similarly, the tangency portfolio can be obtained by maximizing $(\mu_p - \text{rf})/\sigma_p$, where rf is the risk-free rate (a constant in this case), subject to $\sum_{i=1}^{n} w_i = 1$ and solving for w_i. The solution uses the built-in Mathematica function `NMaximize`. The `tangencyPortfolioWeights` function computes the tangency portfolio weights given its five inputs, four as defined in the Section 1 and the risk-free rate.

```
In[51]:= tangencyPortfolioWeights[symbols : {__String}, startDate_,
           endDate_, period_, rf_] := Module[
           {returns, weights},
           returns = getReturnsData[symbols, startDate, endDate,
             period];
           weights = Subscript[w, #] & /@
             Range[Dimensions[returns][[2]]];
           weights /. NMaximize[
              {
               (weights.(12 Mean@returns) - rf) /
                Sqrt[weights.(12 Covariance@returns).weights],
               Total[weights] == 1
              },
              weights][[2]]
         ]
```

Assuming a monthly risk-free rate of 0.1667 percent and using monthly data over the period January 1, 2009, to May 30, 2019, we calculate the tangency portfolio weights for our portfolio of five stocks: (Apple (AAPL), Walmart (WMT), Boeing (BA), 3M and Exxon Mobil (XOM)).

```
In[52]:= tangencyPortfolioWeights[{"AAPL", "WMT", "BA", "MMM", "XOM"},
          {2009, 1, 1}, {2019, 5, 30}, "Month", 0.001667]

Out[52]= {0.62145, 0.359178, 0.515035, 0.0782608, -0.573923}
```

Portfolio optimization using the Wolfram Language is very flexible. We can formulate any kind of portfolio and use built-in functions such as `NMinimize`, `NMaximize` or `QuadraticOptimization` to get numerical solutions to the portfolio problem.

7. Portfolio Risk Decomposition

In this section, we concentrate on how to decompose a measure of portfolio risk (portfolio standard deviation) into risk contribution from individual assets included in the portfolio. It helps to see how individual assets influence portfolio risk. When risk is measured by standard deviation, we can use Euler's theorem for decomposing risk into asset-specific risk contribution. Euler's theorem provides an additive decomposition of a homogeneous function. For reference, see Campolieti and Makarov [6]. Using Euler's theorem, we can define the percentage contribution to portfolio standard deviation of an asset as $\text{mc} \times \text{wt} / \sigma_p$, where mc is the marginal contribution of the asset, wt is the weight of the asset, and σ_p is the portfolio standard deviation.

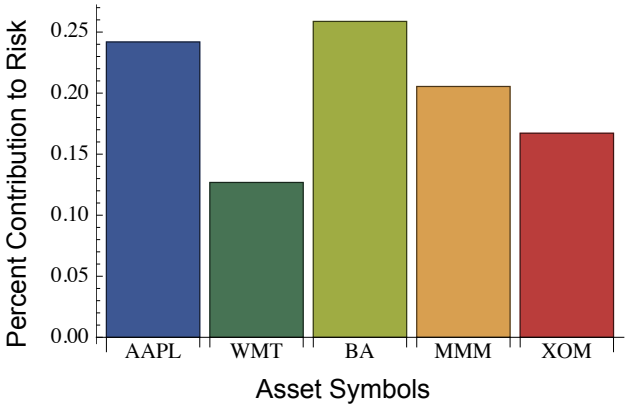
We define the function `decomposePortRisk` for portfolio risk decomposition. It takes five arguments, four arguments as defined in Section 1 and a list of portfolio weights; it returns a bar chart representing the individual asset's risk contribution to the portfolio standard deviation.

```
In[53]:= decomposePortRisk[symbols : {__String}, startDate_,
           endDate_, period_, wt_List] := Module[
           {data, mergedData, prices, pruneprices, returns, Σ,
            σ, mc},
           data = FinancialData[#, {startDate, endDate, period}] & /@
             symbols;
           mergedData = TimeSeriesThread[# &, data,
             ResamplingMethod → Missing[]];
           prices = QuantityMagnitude[mergedData["Values"]];
           pruneprices =
            Select[prices,
             ! MemberQ[#, QuantityMagnitude[Missing[]]] &];
           returns = Differences[pruneprices] / Most[pruneprices];
           Σ = 12 Covariance@returns;
           Labeled[BarChart[wt (wt.Σ) / (wt.Σ.wt),
             ChartLabels → symbols, ChartStyle → "DarkRainbow"],
            Text /@ {"Asset Symbols", "Percent Contribution to Risk"},
            {Bottom, Left}, RotateLabel → True]
         ]
```

We calculate the risk contribution of each asset in a portfolio that consists of five stocks using the historical monthly returns over the period January 30, 2010, to May 30, 2019 and make a bar chart.

```
In[54]:= decomposePortRisk[{"AAPL", "WMT", "BA", "MMM", "XOM"},
          {2010, 01, 30}, {2019, 05, 30}, "Month",
          {0.20, 0.20, 0.2, 0.2, 0.2}]
```

Out[54]=



8. Factor Models

Currently, factor models are widely accepted and used in finance to construct portfolios, to evaluate portfolio performance and for risk analysis. Factor models are regression models. We can use the built-in function `LinearModelFit` to estimate and evaluate the appropriateness of the regression models. In addition, we can download all factor data directly from Prof. Kenneth French's data library. Before we apply factor models to real-world data, we need data in the form

$$(I, D),$$

where

$$I = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1k} \\ f_{21} & f_{22} & \cdots & f_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nk} \end{pmatrix}$$

is a matrix of values of independent variables and

$$D = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

is a vector of values of a dependent variable.

Commonly used factor models are summarized in Table 1. We can find more about the factor models in Fama and French [7]. We use the following notation: $r - \text{rf}$ for excess return on the security or portfolio, MKT for excess return on the value-weighted market portfolio, SML for return on a diversified portfolio of small-capitalization stocks minus the return on a diversified portfolio of large-capitalization stocks, HML for difference in the returns on diversified portfolios of high-book-to-market stocks and low-book-to-market stocks, MOM for difference in returns on diversified portfolios of the prior year's winners and losers, RMW for difference between the returns on diversified portfolios of stocks with robust and weak profitability, CMA for difference between the returns on diversified portfolios of the stocks of low and high investment firms, α for risk-adjusted return on security or portfolio and $\beta_{\text{MKT}}, \beta_{\text{SMB}}, \beta_{\text{HML}}, \beta_{\text{MOM}}, \beta_{\text{RMW}}$ and β_{CMA} for betas or factor loadings.

Description	*Models for r–rf*
Capital asset pricing model (CAPM)	$\text{CAPM} = \alpha + \beta_{\text{MKT}}\,\text{MKT}$
Fama–French three-factor model	$\text{FF}_3 = \text{CAPM} + \beta_{\text{SMB}}\,\text{SMB} + \beta_{\text{HML}}\,\text{HML}$
Carhart four-factor model	$C_4 = \text{FF}_3 + \beta_{\text{RMW}}\,\text{RMW}$
Fama–French five-factor model	$\text{FF}_5 = C_4 + \beta_{\text{CMA}}\,\text{CMA}$

▲ **Table 1.** Overview of factor models.

Before we estimate these models using real data, we need factors data. We download five factors and the momentum factor data from Kenneth French's website (http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html) and define variables to store them.

```
In[55]:= FamaFrench5FactorData =
           First@
            Import[
             "http://mba.tuck.dartmouth.edu/pages/faculty/ken.french
               /ftp/F-F_Research_Data_5_Factors_2x3_CSV.zip",
             "*"];

In[56]:= FamaFrenchMomentumFactor =
           First[
            Import[
             "http://mba.tuck.dartmouth.edu/pages/faculty/ken.french
               /ftp/F-F_Momentum_Factor_CSV.zip",
             "*"]];
```

The function `factorsData` takes only two arguments, the start date and end date, and returns time series of all factors data. The start date and end date must be specified as date objects.

```
In[57]:= factorsData[startDate_, endDate_] := Module[
           {startDateFamaFrenchFactor, startDateMomentumFactor,
            lastDate, dateFactorAssociation, keys, mergedFactorData,
            dateRange, mergedFactorDataFinal},
           startDateFamaFrenchFactor = {1963, 07, 1};
           startDateMomentumFactor = {1927, 01, 1};
           lastDate = DatePlus[Today, {-4, "Month"}];
           dateFactorAssociation =
            Apply[# → {##2} &,
             {FamaFrench5FactorData[[
               5 ;;
                Length[DateRange[startDateFamaFrenchFactor,
                     lastDate, "Month"][[All, 1 ;; 2]]] + 5]],
              FamaFrenchMomentumFactor[[
               15 ;;
                Length[DateRange[startDateMomentumFactor,
                     lastDate, "Month"][[All, 1 ;; 2]]] + 15]]}, {2}];
           keys = Union @@ Keys@dateFactorAssociation;
           mergedFactorData = Join[List /@ keys, ##, 2] & @@
              (Lookup[#, keys, Missing[] & /@ #[[1, 2]]] & /@
                dateFactorAssociation);
           dateRange = DateRange[startDate, endDate, "Month"][[
             All, 1 ;; 3]];
           mergedFactorDataFinal =
            Select[mergedFactorData,
             (startDate[[1]] 100 + startDate[[2]]) ≤ #[[1]] ≤
               (endDate[[1]] 100 + endDate[[2]]) &];
           Prepend[
            Table[
             Flatten[{{dateRange[[i, 1 ;; 2]]},
               mergedFactorDataFinal[[i, {2, 3, 4, 5, 6, 8, 7}]]},
              1],
             {i, Length[mergedFactorDataFinal]}
            ],
            {"Date", "Mkt-RF", "SMB", "HML", "RMW", "CMA", "Mom",
             "RF"}
           ]
          ]
```

Similarly, the `monthlyStockReturns` function can be used to get a stock's monthly returns data. It takes three arguments, a ticker symbol of any publicly traded company, and the start and end dates for the analysis period.

```
In[58]:= monthlyStockReturns[symbol_String, startDate_, endDate_] :=
          Module[
           {data, prices, returns, dates},
           data = FinancialData[symbol, "Close",
             {startDate, endDate, "Month"}];
           prices = QuantityMagnitude[data["Values"]];
           returns = 100 Differences[prices] / Most[prices];
           dates = data["Dates"][[2 ;;, 1, {1, 2}]];
           Prepend[Transpose[{dates, returns}], {"Date", symbol}]
          ]
```

Using `factorsData` and `monthlyStockReturns`, we define `dataforNFactorModel` to combine factors and returns data. The `dataforNFactorModel` takes four arguments (the symbol of the stock for which we want to estimate the factor model, the start date, the end date and an integer that represents the number of factors) and returns a data matrix suitable for `LinearModelFit`.

```
In[59]:= dataforNFactorModel[symbol_String, startDate_, endDate_,
           nfac_ /; MemberQ[{1, 3, 4, 5}, nfac]] := Module[
           {returns, dateAssociationData, mergedData,
            mergedDatafinal, columnsToInclude},
           returns = monthlyStockReturns[symbol, startDate, endDate];
           dateAssociationData =
            Apply[# → {##2} &,
             {factorsData[startDate, endDate][[2 ;;]],
              returns[[2 ;;]]}, {2}];
           mergedData =
            Merge[KeyUnion[dateAssociationData, Missing[] &],
             Identity];
           mergedDatafinal = Select[Flatten[#] & /@ Values[mergedData],
             ! MemberQ[#, Missing[]] &];
           columnsToInclude = Switch[nfac, 1, {1}, 3, {1, 2, 3},
             4, {1, 2, 3, 6}, 5, {1, 2, 3, 4, 5}];
           Table[
            Flatten[{mergedDatafinal[[i, columnsToInclude]],
              mergedDatafinal[[i, 8]] - mergedDatafinal[[i, 7]]}],
            {i, Length[mergedDatafinal]}
           ]
          ]
```

Next, we estimate different factor models for Apple's stock using monthly data from October 1, 2008, to March 30, 2019. We estimate the capital asset pricing model (CAPM) with market factor (MKT).

```
In[60]:= capm = LinearModelFit[dataforNFactorModel["AAPL",
              {2008, 10, 1}, {2019, 3, 30}, 1], {MKT}, {MKT}];
         capm["ParameterTable"]
```

Out[61]=

	Estimate	Standard Error	t-Statistic	P-Value
1	1.21337	0.608441	1.99423	0.0483381
MKT	0.962268	0.143771	6.69308	6.89754×10^{-10}

```
In[62]:= Grid[
           Transpose[{#, capm[#]} &[
             {"AdjustedRSquared", "AIC", "BIC", "RSquared",
              "DurbinWatsonD"}]], Alignment → Left] // Text
```

Out[62]=

AdjustedRSquared	0.261013
AIC	829.044
BIC	837.529
RSquared	0.266973
DurbinWatsonD	1.80899

We estimate the Fama–French three-factor model with market, size and value factors (MKT, SMB, HML).

```
In[63]:= threeFactorModel =
           LinearModelFit[dataforNFactorModel["AAPL", {2008, 10, 1},
             {2019, 3, 30}, 3], {MKT, SMB, HML}, {MKT, SMB, HML}];
         threeFactorModel["ParameterTable"]
```

Out[64]=

	Estimate	Standard Error	t-Statistic	P-Value
1	0.843856	0.604833	1.39519	0.165514
MKT	1.16089	0.1602	7.2465	4.37019×10^{-11}
SMB	−0.219014	0.255563	−0.856986	0.393147
HML	−0.617145	0.230298	−2.67977	0.00839359

```
In[65]:= Grid[Transpose[{#, threeFactorModel[#]} &[
           {"AdjustedRSquared", "AIC", "BIC", "RSquared",
            "DurbinWatsonD"}]], Alignment → Left] // Text
```

Out[65]=

AdjustedRSquared	0.300336
AIC	824.209
BIC	838.351
RSquared	0.317263
DurbinWatsonD	1.76759

Similarly, we estimate the Carhart four-factor model with market, size, value and momentum factors (MKT, SMB, HML, MOM).

```
In[66]:= fourFactorModel =
           LinearModelFit[dataforNFactorModel["AAPL", {2008, 10, 1},
             {2019, 3, 30}, 4], {MKT, SMB, HML, MOM},
            {MKT, SMB, HML, MOM}];
         fourFactorModel["ParameterTable"]
```

Out[67]=

	Estimate	Standard Error	t-Statistic	P-Value
1	0.81764	0.60215	1.35787	0.177053
MKT	1.11445	0.162483	6.85887	3.22822×10^{-10}
SMB	−0.234738	0.254541	−0.922199	0.358275
HML	−0.738527	0.243434	−3.03378	0.00296109
MOM	−0.202644	0.137042	−1.4787	0.141842

```
In[68]:= Grid[Transpose[{#, fourFactorModel[#]} &[
           {"AdjustedRSquared", "AIC", "BIC", "RSquared",
            "DurbinWatsonD"}]], Alignment → Left] // Text
```

Out[68]=

AdjustedRSquared	0.30713
AIC	823.989
BIC	840.959
RSquared	0.329481
DurbinWatsonD	1.81842

Finally, the Fama–French five-factor model with market, size, value, profitability and investment factors (MKT, SMB, HML, RMW, CMA) is estimated as follows.

```
In[69]:= fiveFactorModel =
           LinearModelFit[dataforNFactorModel["AAPL", {2008, 10, 1},
             {2019, 3, 30}, 5], {MKT, SMB, HML, RMW, CMA},
            {MKT, SMB, HML, RMW, CMA}];
         fiveFactorModel["ParameterTable"]
```

Out[70]=

	Estimate	Standard Error	t-Statistic	P-Value
1	0.822316	0.586327	1.40249	0.163373
MKT	1.14355	0.158884	7.19737	5.972×10^{-11}
SMB	0.0127857	0.254256	0.0502868	0.959978
HML	−0.0600549	0.275185	−0.218235	0.82762
RMW	1.20712	0.402983	2.99546	0.00333491
CMA	−1.36628	0.465246	−2.93668	0.00398374

```
In[71]:= Grid[Transpose[{#, fiveFactorModel[#]} &[
           {"AdjustedRSquared", "AIC", "BIC", "RSquared",
            "DurbinWatsonD"}]], Alignment → Left] // Text
```

Out[71]=

AdjustedRSquared	0.368583
AIC	813.38
BIC	833.178
RSquared	0.394043
DurbinWatsonD	1.76778

Once the model is estimated, we must access different properties related to data and fitted models. To assess how well the model fits the data and how well the model meets the assumptions, there are many built-in functions.To learn more about obtaining diagnostic information, see the properties of `LinearModelFit`.

9. Stock Portfolio Performance Measures

There are various measures to evaluate the performance and risk of portfolios. Most of these measures are used to evaluate a portfolio of interest against a chosen benchmark, by taking a snapshot of the past or considering the entire historical picture. We will compute some common metrics often employed by investors while analyzing performance measures. All the computations are based on the formulas developed in Bacon [8]. Most common measures are summarized in the function `computePortfolioPerformanceTable`. The function takes four arguments:

- ticker symbols for the test assets and the benchmark asset
- periodic risk-free rate
- time period
- frequency of data

We repeat the definition of the `getReturnsData` function defined earlier to make this section self-contained.

```
In[72]:= getReturnsData[symbols_ : {__String}, startDate_,
           endDate_, period_] := Module[
           {data, mergedData, prices, pruneprices},
           data = FinancialData[#, {startDate, endDate, period}] & /@
             symbols;
           mergedData = TimeSeriesThread[# &, data,
             ResamplingMethod → Missing[]];
           prices = QuantityMagnitude[mergedData["Values"]];
           pruneprices =
            Select[prices,
             ! MemberQ[#, QuantityMagnitude[Missing[]]] &];
           Differences[pruneprices] / Most[pruneprices]
          ]

In[73]:= computePortfolioPerformanceTable[testAssets_ : {__String},
           benchmarkAsset_String, riskfreeRate_, startDate_,
           endDate_, period_] :=
          Module[{stockSymbols, returns, benchmarkReturns,
            totalReturns, n, mean, standardDeviation, skewness,
            kurtosis, SharpeRatio, adjustedSharpeRatio,
            trackingError, informationRatio, minReturns,
            SortinoRatio, historicalVAR, historicalCVAR,
            quantile, normalVAR, normalCVAR, mSquare,
            performanceMeasures, head},
           stockSymbols = Flatten[{benchmarkAsset, testAssets}];
           returns = getReturnsData[stockSymbols, startDate,
             endDate, period];
           benchmarkReturns = returns[[All, 1]];
           performanceMeasures =
            ConstantArray["NaN",
             {Length[stockSymbols[[2 ;;]]], 19}];
           Do[
            totalReturns = returns[[All, i]];
            n = Length[benchmarkReturns];
            mean = Mean[totalReturns];
            standardDeviation = StandardDeviation[totalReturns];
            skewness = n / ((n - 1) (n - 2))
              Total[(totalReturns - mean)^3] / standardDeviation^3;
            kurtosis =
             n (n + 1) / ((n - 1) (n - 2) (n - 3))
               Total[(totalReturns - mean)^4] / standardDeviation^4 -
              (3 (n - 1)^2) / ((n - 2) (n - 3));
            SharpeRatio = (mean - riskfreeRate) / standardDeviation;
            adjustedSharpeRatio =
             SharpeRatio
              (1 + skewness / 6 SharpeRatio -
                kurtosis / 24 SharpeRatio^2);
```

```
  trackingError = StandardDeviation[
      totalReturns - benchmarkReturns];
  informationRatio = (mean - Mean[benchmarkReturns]) /
      trackingError;
  minReturns = Min[# - riskfreeRate, 0] & /@ totalReturns;
  SortinoRatio = (mean - riskfreeRate) /
      Sqrt[Total[minReturns^2] / (Length[minReturns] - 1)];
  historicalVAR = Sort[totalReturns][[
      Floor[Length[totalReturns] 0.05]]];
  historicalCVAR =
    Mean[Select[totalReturns, # ≤ historicalVAR &]];
  quantile = Quantile[NormalDistribution[], 0.05];
  normalVAR = mean + standardDeviation quantile;
  normalCVAR =
    mean - standardDeviation
      (PDF[NormalDistribution[], quantile] /
        CDF[NormalDistribution[], quantile]);
  mSquare =
    ((mean - riskfreeRate) / standardDeviation -
       (Mean[benchmarkReturns] - riskfreeRate) /
        StandardDeviation[benchmarkReturns])
     StandardDeviation[benchmarkReturns];
  performanceMeasures[[i - 1]] = {
    mean,
    standardDeviation,
    100 (GeometricMean[1 + totalReturns / 100] - 1),
    Total[totalReturns],
    skewness,
    kurtosis,
    SharpeRatio,
    adjustedSharpeRatio,
    trackingError,
    informationRatio,
    informationRatio
     (1 + skewness / 6 informationRatio -
       kurtosis / 24 informationRatio^2),
    SortinoRatio,
    Min[Accumulate /@ SplitBy[totalReturns, Sign]],
    Sqrt[
     Total[(Boole@Thread[totalReturns < 0] totalReturns)^2] /
      n],
    historicalVAR,
    normalVAR,
    historicalCVAR,
    normalCVAR,
    mSquare
   },
  {i, 2, Length[stockSymbols]}
  ];
head = {
  "Arithmetic Mean",
  "Standard Deviation",
```

```
    "Geometric Mean",
    "Cumulative Returns",
    "Sample Skewness",
    "Sample Excess Kurtosis",
    "Sharpe Ratio",
    "Adjusted Sharpe Ratio",
    "Tracking Error",
    "Information Ratio",
    "Adjusted Information Ratio",
    "Sortino Ratio",
    "Continuous Drawdown",
    "Pure Downside Risk",
    "Historical 95% VaR",
    "Normal 95% VaR",
    "Historical 95% CVaR",
    "Normal 95% CVaR",
    "M-Square"
    };
  Prepend[
    Transpose@Prepend[performanceMeasures, head],
    Join[{"Measures"}, stockSymbols[[2 ;;]]]
   ] // TableForm]
```

The next example uses stocks, although these measures are used to evaluate the performance of portfolios, mutual funds and exchange-traded funds. We evaluate the performance of (Walmart Inc. (WMT), Apple Inc. (AAPL) and Microsoft Corporation (MSFT)) against the S&P 500 index (^SPX) using 0.0016 as a monthly risk-free rate and month as the data frequency over the period January 1, 1995, to March 30, 2019.

In[74]:=
```
computePortfolioPerformanceTable[{"WMT","AAPL","MSFT"},"^SPX",0.
{1995,01,01},{2019,03,30},"Month"]//Text
```

Out[74]=

Measures	WMT	AAPL	MSFT
Arithmetic Mean	0.00937693	0.0249761	0.0159834
Standard Deviation	0.0630267	0.124089	0.0905699
Geometric Mean	0.00935714	0.0248994	0.0159426
Cumulative Returns	2.71931	7.24307	4.63519
Sample Skewness	0.194635	−0.274029	0.479727
Sample Excess Kurtosis	1.27668	2.22451	2.45811
Sharpe Ratio	0.123391	0.188381	0.15881
Adjusted Sharpe Ratio	0.123785	0.186141	0.160416
Tracking Error	0.0612221	0.113465	0.0739334
Information Ratio	0.0369914	0.157439	0.119989
Adjusted Information Ratio	0.0370331	0.155945	0.120963
Sortino Ratio	0.197348	0.299788	0.270299
Continuous Drawdown	−0.315263	−1.07271	−0.494336
Pure Downside Risk	0.0385352	0.0771109	0.0523389
Historical 95% VaR	−0.0889564	−0.163909	−0.127342
Normal 95% VaR	−0.0942927	−0.179133	−0.132991
Historical 95% CVaR	−0.127939	−0.260497	−0.166878
Normal 95% CVaR	−0.120629	−0.230985	−0.170836
M−Square	−0.000309288	0.0024311	0.0011842

10. Interactive Graphics and Technical Analysis of Stock Prices

Short-term traders commonly use interactive graphics and technical indicators of stock prices to profit from stocks that may be overbought or oversold. Much is based on market sentiment, but also market timing. When a stock is oversold, the price is low and people want to buy. In comparison, when a stock is overbought, the price is between its normal range and higher, and people do not want to buy, or may want to short sell. Many technical indicators are used to determine a given stock's peak or bottom price and how to take advantage of that information. Three of the most useful functions for technical analysis of stocks are `TradingChart`, `InteractiveTradingChart` and `CandlestickChart`. The documentation provides a comprehensive set of examples on how to use them.

We show one example of how to use the `TradingChart` function and one example of how to use `InteractiveTradingChart`.

You can choose the chart type and from over 100 technical indicators, which are divided into eight groups:

- basic
- moving average
- market strength
- momentum
- trend
- volatility
- band
- statistical

`FinancialIndicator[All]` displays all the available technical indicators.

```
In[75]:= FinancialIndicator[All]

Out[75]= {AbsolutePriceOscillator, AccumulationDistributionLine,
          AccumulativeSwingIndex, Aroon, AroonOscillator,
          AverageDirectionalMovementIndex,
          AverageDirectionalMovementIndexRating, AverageTrueRange,
          ChaikinMoneyFlow, ChaikinOscillator, ChaikinVolatility,
          ChandeMomentumOscillator, CloseLocationValue,
          CloseLocationValueVolume, CommodityChannelIndex,
          CommoditySelectionIndex, DemandIndex,
          DetrendedPriceOscillator, DirectionalMovement,
          DynamicMomentumIndex, EaseOfMovement, FastStochastic,
          ForceIndex, ForecastOscillator, FullStochastic,
          Inertia, IntradayMomentumIndex, KlingerOscillator,
          LinearRegressionSlope, MarketFacilitation,
          MassIndex, MESAPhase, MESASineWave, Momentum,
          MoneyFlowIndex, MovingAverageConvergenceDivergence,
          MovingAverageConvergenceDivergenceHistogram,
          NegativeVolumeIndex, OnBalanceVolume,
```

```
PercentagePriceOscillator, PercentageVolumeOscillator,
Performance, PolarizedFractalEfficiency,
PositiveVolumeIndex, PriceVolumeTrend,
ProjectionOscillator, QStick, RandomWalkIndex,
RangeIndicator, RateOfChange, RelativeMomentumIndex,
RelativeStrengthIndex, RelativeVolatilityIndex,
RSquared, SlowStochastic, StandardDeviation,
StandardError, StochasticMomentumIndex, SwingIndex,
TimeSegmentedVolume, TRIX, TrueRange, UltimateOscillator,
VerticalHorizontalFilter, Volume, VolumeRateOfChange,
WilliamsAccumulationDistribution, WilliamsPercentR,
AccelerationBands, BollingerBands, SimpleMovingAverage,
HullMovingAverage, DoubleExponentialMovingAverage,
ExponentialMovingAverage, ParabolicStopAndReversal,
VolatilitySystem, WeightedMovingAverage,
LinearRegressionTrendlines, LinearRegressionIndicator,
MedianPrice, VariableMovingAverage,
TriangularMovingAverage, TripleExponentialMovingAverage,
TimeSeriesForecast, KeltnerChannels, TypicalPrice,
MovingAverageEnvelopes, PriceChannels, ProjectionBands,
RaffRegressionChannel, StandardDeviationChannels,
StandardErrorBands, WeightedClose, WildersMovingAverage,
Close, High, Low, Open, HighestHigh, LowestLow}
```

Here is the basic format:

```
TradingChart[{symbol,
   {startDate, endDate}, {indicator1, indicator2, ...}]
```

Alternatively, use:

```
TradingChart[timeseriesdata, {indicator1, indicator2, ...}]
```

The time series data must be of the form:

```
{
{date1, {open1, high1, low1, close1, volume1}},
{date2, {open2, high2, low2, close2, volume2}},
...
}
```

The historical open, high, low, close and volume data retrieved from `FinancialData` can also be used as data input. The `TradingChart` function has many options that can be used to enhance graphics. We produce a chart using historical prices of Apple's stock and volume over the period January 1, 2018, to March 30, 2019.

The top of the chart shows a plot of historical prices and 50-day and 200-day moving averages. The second part shows the historical volume. The last two parts show plots of two indicators, the commodity channel index (`CommodityChannelIndex`) and the relative strength index (`RelativeStrengthIndex`).

A good introduction to technical indicators can be found in standard references, including at Fidelity Learning Center [9].

```
In[76]:= TradingChart[{"AAPL", {{2018, 01, 01}, {2019, 03, 30}}},
          {"Volume", FinancialIndicator["SimpleMovingAverage", 50],
           FinancialIndicator["SimpleMovingAverage", 200],
           "CommodityChannelIndex", "RelativeStrengthIndex"}]
```

Out[76]=

The `InteractiveTradingChart` function provides a point-and-click interactive chart, with a similar setup:

```
InteractiveTradingChart[{symbol,
   {startDate, endDate}, {indicator1, indicator2, ...}]
```

Alternatively, use:

```
InteractiveTradingChart[
 timeseriesdata, {indicator1, indicator2, ...}]
```

For example, we make a chart showing prices, volume and indicators for historical data of Apple's stock over the period January 1, 2018, to March 30, 2019. The `InteractiveTradingChart` function provides a user-friendly environment where you can drag a slider to view different parts of the chart or you can choose different indicators with point-and-click.

In[77]:=
```
InteractiveTradingChart[
 {"AAPL", {{2018, 01, 01}, {2019, 03, 30}}},
 {"Volume", FinancialIndicator["SimpleMovingAverage", 50],
  FinancialIndicator["SimpleMovingAverage", 200],
  "CommodityChannelIndex", "RelativeStrengthIndex"}]
```

Out[77]=

11. Essentials of Bond Mathematics

A bond is a long-term debt instrument in which a borrower agrees to make payments of principal and interest, on specific dates, to the holders of the bond. When it comes to analysis and pricing of bonds and computing returns, convexity and duration are important concepts. When a bond is traded between coupon payment dates, its price has two parts: the quoted price and the accrued interest. The quoted price is net of accrued interest, and does not include accrued interest. Accrued interest is the proportion or share of the next coupon payment. The full price is the price of a bond including any interest that has accrued since issue of the most recent coupon payment. Similarly, yield to maturity is the rate of return earned on a bond if it is held to maturity. Duration is a measure of the average length of time for which money is invested in a coupon bond. Convexity estimates the change in the bond price given a change in the yield to maturity, assuming a nonlinear relationship between the two. The built-in functions `FinancialBond` and `FindRoot` can be used to compute various properties including value of the bond, accrued interest, yield, duration, modified duration, convexity, and so on. This section provides a few examples of how to use `FinancialBond` and how the concepts of bond convexity and duration can be used in bond portfolio management.

We discuss zero-coupon bonds first. The zero-coupon bond does not make coupon payments. The only cash payment is the face value of the bond on the maturity date. The yield to maturity (ytm) for a zero-coupon bond with n periods to maturity, current price P and face value FV can be obtained by solving $P = \text{FV} / (1 + \text{ytm})^n$.

For example, we compute the yield to maturity of a zero-coupon bond with a \$10,000 face value, time to maturity 4 years and current price \$9,662 using `NSolve` and `FindRoot`.

```
In[78]:= NSolve[9662 == 10 000/(1 + ytm)^4 && ytm ≥ 0, ytm, Reals]

Out[78]= {{ytm → 0.00863316}}
```

```
In[79]:= FindRoot[
          FinancialBond[
            {"FaceValue" → 10 000, "Coupon" → 0, "Maturity" → 4,
             "CouponInterval" → 1},
            {"InterestRate" → ytm, "Settlement" → 0}
           ] == 9662,
          {ytm, .1}
         ]

Out[79]= {ytm → 0.00863316}
```

Similarly, `FindRoot` can also be used to compute the yield to maturity of a nonzero coupon payment bond.

For example, we compute the yield to maturity of a $1,000 par value 10-year bond with 5% semiannual coupons issued on June 20, 2013, with maturity date of June 20, 2023, selling for $920 on September 15, 2018.

```
In[80]:= FindRoot[
          FinancialBond[
            {"FaceValue" → 1000, "Coupon" → 0.05,
             "Maturity" → {2023, 6, 20}, "CouponInterval" → 1/2},
            {"InterestRate" → y, "Settlement" → {2018, 9, 15}}
           ] == 920,
          {y, .1}
         ]

Out[80]= {y → 0.0700175}
```

`FinancialBond` can also be used to compute the price, duration, modified duration and convexity of a bond. For example, we compute those values for a bond with 8% yield, 8% annual coupons, 10-year maturity and $1,000 face value.

```
In[81]:= FinancialBond[
          {"FaceValue" → 1000, "Coupon" → 0.08, "Maturity" → 10,
           "CouponInterval" → 1},
          {"InterestRate" → .08, "Settlement" → 0},
          {"Value", "Duration", "ModifiedDuration", "Convexity"}
         ]

Out[81]= {1000., 7.24689, 6.71008, 60.5313}
```

There are different approaches to bond portfolio management. We concentrate here on a liability-driven portfolio strategy, in which the characteristics of the bonds that are held in the portfolio are coordinated with those of the liabilities the investor is obligated to pay. The matching techniques can range from an attempt to exactly match the levels and timing of the required cash payments to more general approaches that focus on other investment characteristics, such as setting the average duration or convexity of the bond portfolio equal to that of the underlying liabilities. One specific example would be to construct the portfolio so that the duration of the bond portfolio is equal to the duration of cash obligation and the total money invested in the bond portfolio today is equal to the present value of the future cash obligations.

To illustrate the concept of bond portfolio management, assume that we have an obligation to pay $1,000,000 in 10 years and there are two bonds available for investment. The first bond matures in 30 years with $100 face value and annual coupon payment of 6%. The second bond matures in 10 years with $100 face value and annual coupon payment of 5%. The yield to maturity is 9% on both bonds. We can decide on how much to invest in each bond so that the overall portfolio is immunized against changes in the interest rate.

We compute the duration of each bond using `FinancialBond`, which gives that the duration of bond 1 is 11.88 and that of bond 2 is 6.75. Assuming that the proportion of money invested in bonds 1 and 2 is w_1 and w_2, the immunized portfolio is found by solving the simultaneous equations:

$$\begin{cases} w_1 + w_2 = 1 \\ 11.88\, w_1 + 6.75\, w_2 = 10 \end{cases}$$

These two equations can be solved using `Solve`.

```
In[82]:= Solve[w1 + w2 == 1 && 11.88 w1 + 6.75 w2 == 10, {w1, w2}]

Out[82]= {{w1 → 0.633528, w2 → 0.366472}}
```

The result shows how much money should be allocated to each bond. More examples can be found in Benninga [10]. A more general approach to bond portfolio management can be solved by using linear programming. It is beyond the scope of this article to introduce linear programming.

12. Binomial and Black–Scholes–Morton Stock Option Pricing Models

The most popular options pricing models are the binomial model and the Black–Scholes–Morton option pricing formulas for European options. In the next two subsections, we discuss these models and their implementation.

12.1 Binomial Option Pricing Model

Following the notation from Hull [11], define:

S_0: the current stock price

K: the strike price

rf: the annual risk-free rate

σ: the annual standard deviation of the stock returns

T: the time to expiration of the option in years

n: the total number of up and down moves

j: the number of upward moves.

Then $n - j$ is the number of down moves on the tree.

In terms of these variables, we can define:

• time per period $\Delta t = T / n$

• up factor $u = e^{\sigma \sqrt{\Delta t}}$

• down factor $d = e^{-\sigma \sqrt{\Delta t}}$

• probability of an up move $p = \frac{e^{\text{rf}\,\Delta t} - d}{(u-d)\, e^{\text{rf}\,\Delta t}}$

• probability of a down move $q = 1 - p$

• stock price $S_0\, u^j\, d^{n-j}$ at node n

• payoff from a European call $\max[S_0\, u^j\, d^{n-j} - K, 0]$

• payoff from a European put $\max[K - S_0\, u^j\, d^{n-j}, 0]$

In the risk-neutral world, the price of the call and put using the n-period binomial options pricing model can be computed as:

$$\text{call} = e^{-\text{rf}\,T} \sum_{j=0}^{n} \binom{n}{j} p^j\, q^{n-j} \max\left[S_0\, u^j\, d^{n-j} - K, 0\right],$$

$$\text{put} = e^{-\text{rf}\,T} \sum_{j=0}^{n} \binom{n}{j} p^j\, q^{n-j} \max\left[K - S_0\, u^j\, d^{n-j}, 0\right].$$

The functions `binomialCallPrice` and `binomialPutPrice` calculate the prices of European call and put options; they output the option price. Each function takes six arguments as defined at the beginning of this subsection.

```
In[83]:= binomialCallPrice[S0_, K_, σ_, rf_, T_, n_] := Module[
           {u, d, r, p, q},
           u = N@Exp[σ Sqrt[T / n]];
           d = 1/u;
           r = Exp[rf T / n];
           p = (r - d)/((u - d) r);
           q = 1/r - p;
           Sum[Max[S0 u^j d^(n - j) - K, 0] Binomial[n, j] p^j q^(n - j),
            {j, 0, n}]
          ]
```

```
In[84]:= binomialPutPrice[S0_, K_, σ_, rf_, T_, n_] := Module[
           {u, d, r, p, q},
           u = N@Exp[σ Sqrt[T / n]];
           d = 1/u;
           r = Exp[rf T / n];
           p = (r - d)/((u - d) r);
           q = 1/r - p;
           Sum[Max[ K - S0 u^j d^(n - j), 0] Binomial[n, j] p^j q^(n - j),
            {j, 0, n}]
          ]
```

We use these functions to find the prices of call and put options when the current stock price is \$50, the strike price is \$45, the annual volatility is 40%, the risk-free rate is 10%, the time to maturity is half a year and the total number of up and down moves is 500.

```
In[85]:= binomialCallPrice[50, 45, 0.4, 0.1, 0.5, 500]

Out[85]= 9.57705

In[86]:= binomialPutPrice[50, 45, 0.4, 0.1, 0.5, 500]

Out[86]= 2.38237
```

12.2 Black–Scholes–Morton Options Pricing Model

Similar to the binomial option pricing formula defined in the last section, we follow Hull [11] to explain the Black–Scholes–Morton option pricing formulas. Define the variables:

c_0: the current call option value

p_0: the current put option value

S_0: the current stock price

K: the exercise price

T: the time in years until the option expires

rf: the annual risk-free interest rate

σ: the annual standard deviation of the rate of return of the underlying stock

δ: the annual dividend yield of the underlying stock

Furthermore, assume that $\phi(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$ is the standard normal density function (where $e \approx 2.71828\ldots$ the base of the natural logarithms) and let $N(x) = \int_{-\infty}^{\infty} \phi(x)\, dx$ be the standard normal cumulative distribution function, so that $N(d)$ denotes the probability that a random variable drawn from a standard normal distribution is less than d. Then the call and put values can be computed as

$$c_0 = S_0\, e^{-\delta T} N(d_1) - K\, e^{-\mathrm{rf}\, T} N(d_2),$$

$$p_0 = K\, e^{-\mathrm{rf}\, T}(1 - N(d_2)) - S_0\, e^{-\delta T}(1 - N(d_1)),$$

where

$$d_1 = \ln\left(\frac{S_0}{K}\right) + \left(\mathrm{rf} - \delta + \frac{\sigma^2}{2}\right) T,$$

$$d_2 = d_1 - \sigma\, \sqrt{T}.$$

Table 2 summarizes the price sensitivity measures of call and put options (denoted by Greek symbols) with respect to their major price determinants; here V stands for value of the option.

Greeks	*Evaluation*	*Interpretation*
		estimated change in value relative to change in:
Delta (Δ)	$\frac{\partial V}{\partial S}$	the underlying asset price
Vega (υ)	$\frac{\partial V}{\partial \sigma}$	the underlying asset volatility
Rho (ϱ)	$\frac{\partial V}{\partial r}$	the risk-free interest rate
Theta (θ)	$\frac{\partial V}{\partial T}$	the time to expiration (for a negative change)
Gamma (Γ)	$\frac{\partial^2 V}{\partial S^2}$	estimated change in delta relative to a change in the underlying asset price

▲ **Table 2.** Greeks as measures of sensitivity.

The built-in Mathematica function `FinancialDerivative` computes the values and other price sensitivity measures for common types of derivative contracts. The function can compute the value of an option, any of delta, gamma, theta and vega, as well as the implied volatility of the contract. The Mathematica documentation provides many examples of how to use `FinancialDerivative`. Here are the first 10 of a list of 101 available contracts.

```
In[87]:= Take[FinancialDerivative[], 10] // Column
```

```
Out[87]= Altiplano
         {American, Call}
         {American, Put}
         Annapurna
         {AsianArithmetic, European, Call}
         {AsianArithmetic, European, Put}
         {AsianGeometric, European, Call}
         {AsianGeometric, European, Put}
         Atlas
         {BarrierDownIn, American, Call}
```

For example, we compute the price and Greeks of the European-style put option with strike price \$50, expiration date 0.3846 years, interest rate 5%, annual volatility 20%, no annual dividend and current price \$49.

```
In[88]:= FinancialDerivative[{"European", "Call"},
          {"StrikePrice" → 50, "Expiration" → 0.3846},
          {"InterestRate" → 0.05, "Volatility" → 0.2,
           "CurrentPrice" → 49, "Dividend" → 0}, {"Value", "Greeks"}]
```

```
Out[88]= {2.40046, {Delta → 0.521602, Gamma → 0.0655455,
           Rho → 8.90646, Theta → -4.30541, Vega → 12.1051}}
```

Similarly, we compute the implied volatility of an American-style call option with the same values of the parameters.

```
In[89]:= FinancialDerivative[{"American", "Call"},
          {"StrikePrice" → 50, "Expiration" → 0.3846, "Value" → 45},
          {"InterestRate" → 0.05, "CurrentPrice" → 49, "Dividend" → 0},
          "ImpliedVolatility"]

Out[89]= 23.276
```

One interesting application of `FinancialDerivative` is to get real-world data and compute related measures related to options. We define `computeOptionParameters` that computes the theoretical value of options and their Greeks, which takes five arguments:

- ticker symbol (`symbol`)
- strike price (`K`)
- expiration date (`expirationDate`)
- risk-free rate (`rf`)
- either `"European"` or `"American"` (`type`)

```
In[90]:= computeOptionParameters[symbol_String, K_, expirationDate_,
           rf_, type_] := Module[
           {startDate, endDate, returns, σ, price, timeToExpiration,
            δ, callprice, putprice},
           startDate = DatePlus[Today, {-1, "Year"}];
           endDate = DatePlus[Today, {-1, "Day"}];
           returns = QuantityMagnitude[
             FinancialData[symbol, "FractionalChange",
               {startDate, endDate}]["Values"]];
           σ = StandardDeviation[returns] Sqrt[252] / 100;
           price = FinancialData[symbol, "Price"][[1]];
           timeToExpiration =
            DateDifference[Today, DateObject[expirationDate]][[1]] /
              360 // N;
           δ =
            Total@QuantityMagnitude[
               FinancialData[symbol, "Dividend",
                 {startDate, endDate}]["Values"]] / price;
           {callprice, putprice} =
            FinancialDerivative[{type, #},
               {"StrikePrice" → K, "Expiration" → timeToExpiration},
               {"InterestRate" → rf, "Volatility" → σ,
                "CurrentPrice" → price,
                "Dividend" → If[NumberQ@δ, δ, 0]},
               {"Value", "Greeks"}] & /@ {"Call", "Put"};
```

```
ArrayFlatten[
 {
  {{#} & /@ Flatten[{" ", "Price", Keys[putprice[[2]]]}],
   Prepend[
    Transpose[{
      Flatten[{callprice[[1]], Values@callprice[[2]] }],
      Flatten[{putprice[[1]], Values@putprice[[2]] }]
     }],
    {"Call", "Put"}]}
 }
]
]
```

We compute the European-style option parameters for Boeing (BA), assuming that the option expires on December 28, 2020, with exercise price \$145 and risk-free rate 0.0187. Make sure that the expiration date is later than the current date, since `computeOptionParameters` uses historical data.

In[91]:=
```
Grid[
  computeOptionParameters["BA", 320, {2020, 12, 28},
   0.0187, "European"],
  Frame → All] // Text
```

Out[91]=

	Call	Put
Price	7.06189	150.91
Delta	0.179132	−0.811994
Gamma	0.00288482	0.00288483
Rho	9.27225	−111.642
Theta	−30.6819	−28.814
Vega	28.2192	28.2188

We strongly encourage you to explore the built-in or online documentation for the powerful `FinancialDerivative` function.

13. Conclusion

The article provides a brief overview of built-in functions and introduces many functions especially designed for analysis of financial data. In particular, we have focused on functions that are more relevant to introductory computational financial concepts. We emphasize importing company fundamental data and its visualization, analysis of individual stocks and portfolio returns, factor models and the use of built-in functions for bond and financial derivative analysis. The functions we have provided are just a few examples. The Wolfram Language can do much more than what we have shown in this article. Interested readers can start exploring the Wolfram Language via Mathematica's extensive documentation.

References

[1] H. R. Varian, ed., *Economic and Financial Modeling with Mathematica*, New York: Springer-Verlag, 1993.

[2] H. R. Varian, ed., *Computational Economics and Finance Modeling and Analysis with Mathematica*, New York: Springer-Verlag, 1996.

[3] W. Shaw, *Modelling Financial Derivatives with Mathematica*, Cambridge, UK: Cambridge University Press, 1998.

[4] S. Stojanovic, *Computational Financial Mathematics using MATHEMATICA Optimal Trading in Stocks and Options*, Boston: Birkhäuser, 2003.

[5] F. Black, "Capital Market Equilibrium with Restricted Borrowing," *Journal of Business*, **4**(3), 1972 pp. 444–455. www.jstor.org/stable/2351499.

[6] G. Campolieti and R. Makarov, *Financial Mathematics: A Comprehensive Treatment*, London: Chapman and Hall/CRC Press, 2014.

[7] E. F. Fama and K. R. French, "A Five-Factor Asset Pricing Model," *Journal of Financial Economics*, **116**(1), 2015 pp.1–22. doi:10.1016/j.jfineco.2014.10.010.

[8] C. R. Bacon, *Practical Risk-Adjusted Performance Measurement*, 2nd ed., Hoboken: Wiley, 2013.

[9] Fidelity Learning Center. "Technical Indicator Guide." (Jul 29, 2020) www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/overview.

[10] S. Benninga, *Financial Modeling*, 4th ed., Massachusetts: The MIT Press, 2014.

[11] J.C. Hull, *Options, Futures and Other Derivatives*, 10th ed., New York: Pearson Education Limited, 2018.

R. Adhikari, "Foundations of Computational Finance," *The Mathematica Journal*, 2020.
https://doi.org/10.3888/tmj.22–2.

About the Author

Ramesh Adhikari is an assistant professor of finance at Humboldt State University. Prior to coming to HSU, he taught undergraduate and graduate students at the Tribhuvan University and worked at the Central Bank of Nepal. He was also a research fellow at the Osaka Sangyo University, Osaka, Japan. He earned a Ph.D. in Financial Economics from the University of New Orleans. He is interested in the areas of computational finance and high-dimensional statistics.

Ramesh Adhikari
School of Business, Humboldt State University
1 Harpst Street
Arcata, CA 95521
ramesh.adhikari@humboldt.edu