Accessing and Plotting FRED Data

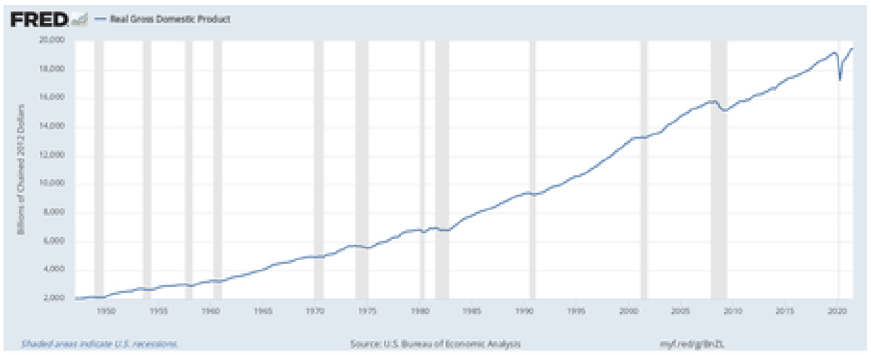
This section provides an introduction to accessing and plotting the time-series data in the Federal Reserve Economic Data (FRED) database. FRED provides an immense online repository of time-series data from more than 100 sources. As a rather arbitrary choice, the examples below all use the GDPC1 series for real GDP. This brief introduction shows how to import plots from FRED, how to download CSV data and create a dataset that is easy to plot, and how to import time-series data by means of the FRED API.

Method 1: Import a Chart from FRED

The FRED website automatically provides plots of the time-series data in the FRED database. For example, to examine the real GDP series GDPC1 on FRED, point a browser at https://fred.stlouis-fed.org/series/GDPC1. If you would like to include the displayed chart in a Mathematica notebook, get the “Image short URL” from the “Share Links” button. The **`Import`** command can import the chart with this URL as follows.

```
In[796]:= Import["https://fred.stlouisfed.org/graph/fredgraph.png?g=BnZL",
            ImageSize → Large]
```

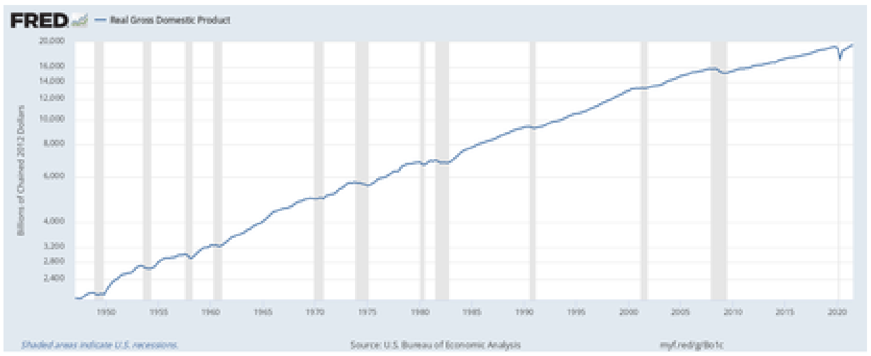
Out[796]=



The basic FRED plots are very professional, but often additional data manipulation is desirable. FRED provides facilities for common manipulations. For example, since the real GDP series grows over time, a ratio scale would facilitate visual comparisons between initial periods and final periods in the plot. To switch to a ratio scale, click the “Edit Graph” button and then the “FORMAT” tab, and then tick the “Log scale” box. The new sharable link provided by the resulting plot will access the new chart.

```
In[797]:= Import["https://fred.stlouisfed.org/graph/fredgraph.png?g=Bo1c",
            ImageSize → Large]
```

Out[797]=



The resulting charts are covered by the FRED Graphs License, which is quite liberal. Essentially, when you share the charts, you must display the entire chart and acknowledge the source as follows:

FRED® Graphs ©Federal Reserve Bank of St. Louis. 2021. All rights reserved. All FRED® Graphs appear courtesy of Federal Reserve Bank of St. Louis. https://fred.stlouisfed.org/

Method 2: Download CSV Data from FRED

The second approach has two steps: download the data from FRED in CSV format, and then import it as a dataset. To follow along, first establish a folder for downloaded FRED data. This section calls it **`fredFolder`**. We are going to use the “Download” button on the FRED chart to download the data in CSV format. Do not transform the chart before downloading the data, unless you wish to download transformed data. Download the GDPC1 data as CSV to **`fredFolder`**.

One disadvantage to this approach is that it involves downloading the data by hand. Currently it is possible to automate the download by means of an HTTP GET request, using the series identifier as in the following example. (Before attempting the following examples, assign a value to the **`fredFolder`** variable.) Note the use of FileNameJoin to construct the full path to the new file.

```
In[798]:= request = "https://fred.stlouisfed.org/graph/fredgraph.csv?id=GDPC1";
          filename = FileNameJoin[{fredFolder, "GDPC1.csv"}];
          test = URLDownloadSubmit[request, filename]
```

```
Out[800]= TaskObject[ URL  Task UUID: 3fad8e25-6622-43de-b6c8-a0aa43a1da5a
                           Task environment: External
                           Task type: Asynchronous                          ]
```

After constructing the path to the imported data, import the data in a notebook with **`SemanticImport`**. This produces a **`Dataset`**, as follows.

```
In[801]:= data = SemanticImport[filename] (* quarterly data *)
```

Out[801]=

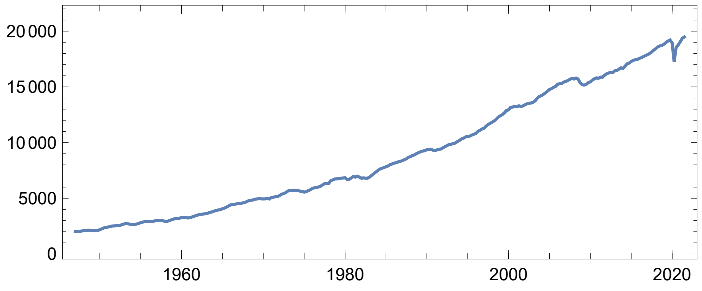
DATE	GDPC1
Wed 1 Jan 1947	2034.45
Tue 1 Apr 1947	2029.02
Tue 1 Jul 1947	2024.83
Wed 1 Oct 1947	2056.51
Thu 1 Jan 1948	2087.44

rows 1–5 of **299**

The **`SemanticImport`** recognizes that the first column contains dates, and it correspondingly creates a **`DateObject`** for each observation in the dataset. You can tell this by looking at the formatting when the dataset displays in a notebook. The **`DateListPlot`** command works with dates and values organized like this in a dataset.

```
In[802]:= DateListPlot[data,
            PlotTheme → "Default", AspectRatio → 2 / 5]
```

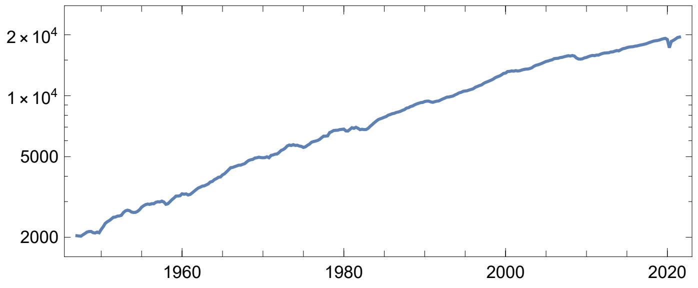
Out[802]=



To switch to a ratio scale, use **`DateListLogPlot`**.

```
In[803]:= DateListLogPlot[data,
            PlotTheme → "Default", AspectRatio → 2 / 5]
```

Out[803]=



There is a disadvantage to this approach: recessions are not marked for us, and adding them is extra work. (The NBER has the official business cycle dates.) However, there is an offsetting advantage: easy data manipulation.

Transforming a Dataset

Before exploring data manipulation, it may be useful to briefly review the basic syntax for querying a dataset. Use the **`Query`** command to retrieve data from a dataset. This command produces a function that can be applied to a dataset to retrieve data. As a simple example, consider a query that retrieves the the first record (i.e., row). To get the first record, apply **`Query[1]`** to the data. The expression **`Query[1][data]`** accomplishes this, but it is more idiomatic to use the equivalent expression **`Query[1]@data`**.

```
In[804]:= Query[1]@data
```

Out[804]=

DATE	Wed 1 Jan 1947
GDPC1	2034.45

In order to illustrate the power of dataset queries, consider the goal of examining the data type in each field (i.e., column) of this dataset. To display the data type of each field, construct a more complex query based on the following goals: fetch the first record, and for each field, produce the data type of the value. The **Head** command reports the data type, so apply apply **Head** to each field as follows. (As shown below, this same approach works for any function, not just **Head**.)

```
In[805]:= Query[1, All, Head]@data
```

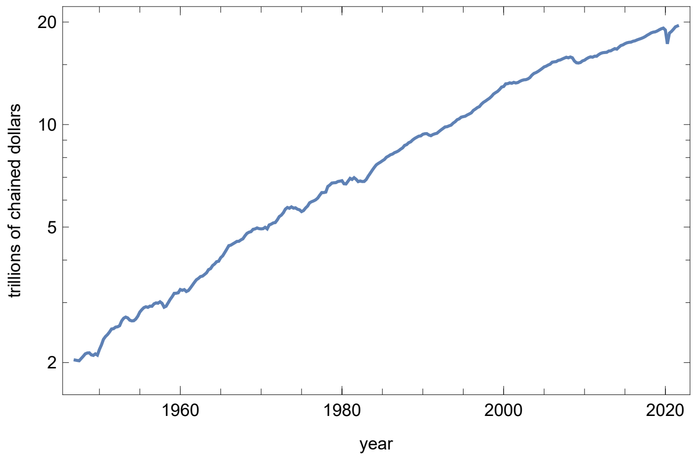
Out[805]=

DATE	DateObject
GDPC1	Real

As a convenience, datasets may be directly applied to the query arguments. For example, to get the first record, just use **`data[1]`**. This use of brackets with a dataset is supplementary syntax for a *query* of the dataset; here it produces the first record (i.e., row). Use the same logic as before: query data for the first record, all fields, and apply **Head** to each field. In other words, apply the dataset to the same query arguments. So **`data[1,All,Head]`** produces the same result.

Finally, use a query to create a usefully transformed dataset. As a particularly simple example, convert the GDP data from billions to trillions. The query **`Query[All,f]`** will apply the function *f* to each record in the dataset, thereby producing a new transformed dataset. A very nice feature of datasets that have strings as headers is that ordinary functions can refer directly to the headers. For example, for any record from our dataset, the function **`Function[{#DATE,#GDPC1/1000}]`** will produce a list from the DATE field and the GDPC1 field, appropriately deflating the latter. (Note the required octothorpe; see the documentation of **`Function`** for an explanation.)

In[806]:=
```
transform = Function[{#DATE, #GDPC1 / 1000}]; (* the desired transformation *)
query = Query[All, transform]; (* the associated query *)
DateListLogPlot[query@data,
 FrameLabel → {"year", "trillions of chained dollars"}]
```

Out[808]=


Method 3: Use the FRED API

As a web service, FRED provides internet data access via the FRED API. In order to use this service, you must first register for an account and then create an API key. You can then use the **`ServiceConnect`** command to connect to FRED. (It will asked for an API key.) To make use of the resulting ServiceObject, be sure to bind a variable name to it.

In[809]:=
```
fred = ServiceConnect["FederalReserveEconomicData"];
```

Although it is well documented, the syntax for querying the FRED database is not completely intuitive. Initially it is easiest to just copy examples. For example, the following code fetches the information about the GDPC1 series and stores it in a dataset with a single record, which can be queried by field name. The subsequent query determines the units of the GDPC1 series.

In[810]:=
```
info = fred["SeriesSearch", "Query" → "GDPC1"] ; (* returns only a description *)
info[1, "Units"]
```

Out[811]=
```
Billions of Chained 2012 Dollars
```

Fetch the series data as follows. Instead of a **`Dataset`**, this produces a **`TimeSeries`**. This is a special data type that provides a number of conveniences.

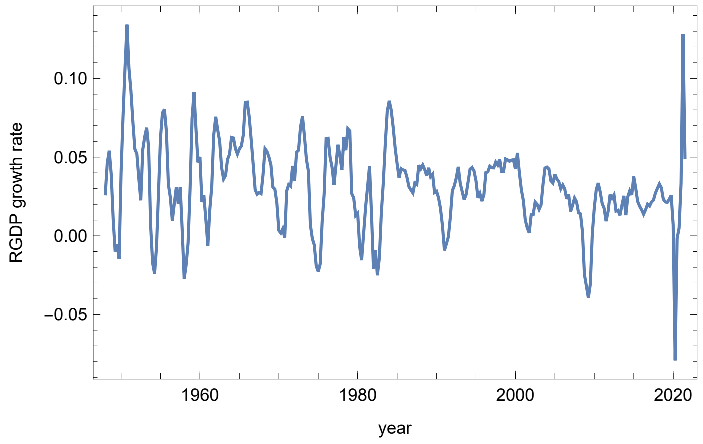
In[812]:=
```
ts = fred["SeriesData", "ID" → "GDPC1"] (* returns the data as a TimeSeries *)
```

Out[812]=
```
TimeSeries[ Time: 01 Jan 1947 to 01 Jul 2021
            Data points: 299 ]
```

A particularly nice feature of a **`TimeSeries`** is that algebraic transformations affect the values but leave the dates untouched. For example, to again change the units from billions to trillions, simply use the expression **`ts/1000`**. As an example that begins to expose the power of this feature, plot a one-year moving average of the annualized growth rates of the quarterly data.

```
In[813]:= gdpGrowthRates = MovingAverage[Ratios[ts]^4 - 1, 4];
          DateListPlot[gdpGrowthRates,
           FrameLabel → {"year", "RGDP growth rate"}]
```

Out[814]=



The **TimeSeries** data type is so convenient that we may occasionally wish to extract data from a **Dataset** in order to create a **TimeSeries**. As of version 12, the **TimeSeries** command requires a list of time-value pairs and will not accept a dataset, so we must convert the data into such a matrix. Use the **Values** command to discard the dataset headers, and then use the **Normal** command to convert the headerless dataset to a matrix. The following example uses the dataset above, which contains only the dates and real GDPs.

```
In[815]:= ts = TimeSeries[Normal@Values@data] (* convert Dataset to TimeSeries *)

Out[815]= TimeSeries[ Time: 01 Jan 1947 to 01 Jul 2021
                      Data points: 299 ]
```