目次

第5章

成長と変動

5.1 概要

この講義では以下のことを学ぶ。

- 片対数グラフ
- マクロ経済学における短期と長期
- 完全雇用，自然失業率，潜在 GDP
- 利用可能なマクロ・データ
- Pandas を使って取得する方法

理論パートの内容は，短期モデルと長期モデルを切り分けるときの考え方について説明する。次章以降に学ぶマクロ経済理論を学ぶための準備である。この章のプログラミングパートの後半部分は他の章と比べて陳腐化するのが早い。次のような理由からだ。

- Pandas のような進化の早いライブラリは使用法が比較的早く変化するため，ここで紹介したコードが標準的な方法ではなくなる，あるいは，まったく実行できなくなる可能性がある。
- データ取得は外部の Web API に依存している。API のバージョンアップによって古いコードが動かなくなる可能性がある。
- 新たなデータ提供者，外部 API のの出現により，より洗練されたデータ取得方

法が生み出される。

このような問題を事前に解決することはできないので，必要な修正と知識の更新は読み手に委ねることにする。気をつけて読んでください。

5.2　理論：数学

5.2.1　片対数グラフ

経済時系列を図示するときには，片対数グラフがよく使われるので簡単に説明しておこう。

ある変数 $y > 0$ の変化を次のように表現する。

$$y_t = (1 + g_t) y_{t-1}$$

ここで g は y の変化率（$1 + g_t > 0$）であり，時間変化する可能性がある。対数値を取ると，

$$\begin{aligned} \log y_t &= \log(1 + g_t) + \log y_{t-1} \\ &= \log(1 + g_t) + \log(1 + g_{t-1}) + \log y_{t-2} \\ &= \cdots \\ &= \sum_{k=1}^{t} \log(1 + g_k) + \log y_0 \end{aligned}$$

と書ける。ここで，$g_t \equiv g$ と定数であるとすれば，

$$\log y_t = t \log(1 + g) + \log y_0$$

横軸を t，縦軸を $\log y_t$ としてグラフを書くと，傾きが $\log(1 + g) \approx g$ の直線になる。実際にグラフを見てみよう。コードを短くするため，$y_0 = 1, g_t \equiv g = 0.02$ として，$y_t = (1 + 0.02)^t$ と解いてしまっている。図 5.1 のようになる。

```
import numpy as np
```

```
import matplotlib.pyplot as plt


t = np.arange(100)
y = (1 + 0.02) ** t
plt.plot(t, y)
plt.show()
```

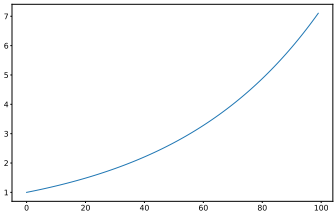

図 5.1: y_t のグラフ

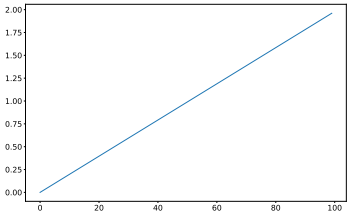

図 5.2: $\log y_t$ のグラフ

対数変換したグラフは図 5.2 のようになる。

```
plt.plot(t, np.log(y))
plt.show()
```

g が上昇すると傾きがきつくなる。では，g_t が一定ではなく上昇傾向にある場合はどうなるだろうか？ 変化の様子をみるために，$t < 50$ に対して $g_t = 0.02$,$t \geq 50$ に対して $g_t = 0.04$ であるとして作図してみよう（図 5.3）。$t = 50$ のあたりで変化率が変わっていることが分かるだろうか。

```
y2 = np.empty_like(y)
y2[:50] = (1 + 0.02) ** t[:50]
y2[50:] = y2[49] * (1 + 0.04) ** (t[50:] - 49)
plt.plot(t, y2)
plt.show()
```

対数変換したグラフ図 5.4 を見ると，変化率の変化は明白である。

```
plt.plot(t, np.log(y2))
plt.show()
```

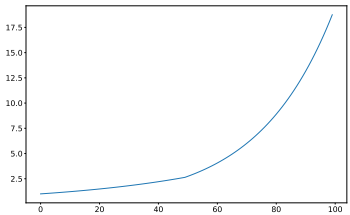

図 5.3: y_t のグラフ（$t \geq 50$ で成長率が 0.04 に上昇）

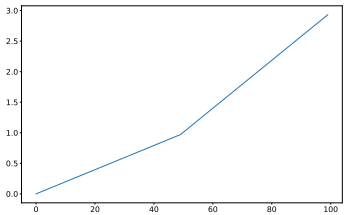

図 5.4: $\log y_t$ のグラフ（$t \geq 50$ で成長率が 0.04 に上昇）

上のコードでは，対数計算を明示的に行ったが，Matplotlib でプロットするときには明示的に対数計算をする代わりにメモリ（スケール）を変更するのが普通である。次のようにする（図 5.5）。縦軸のメモリに注目せよ。

```
plt.plot(t, y2)
plt.yscale('log')
plt.show()
```

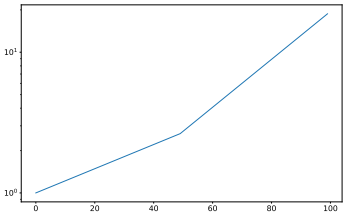

図 5.5: y_t の片対数グラフ（$t \geq 50$ で成長率が 0.04 に上昇）

片方の軸だけが対数メモリになっているので，このようなグラフを片対数グラフという。10 を底とする対数（常用対数）を用いることが多い。その場合，縦軸メモリを 1 つ上に上るとデータが 10 倍になるようなメモリになっている。

問題 5.1. 文房具店や大学生協に行き，片対数メモリのグラフ用紙を購入し，使ってみよう。

問題 5.2. 上のコードは途中で成長率が上昇するケースを可視化した。逆に，途中で成長率が低下する場合はどのようになるか。結果を予想し，コードを書き，グラフを確認せよ。

5.2.2　移動平均

非常に変動が激しいデータ x が手元にあるとしよう。このとき，隣り合う時点間の変化

$$\Delta x_{t+1} = x_{t+1} - x_t$$

は一時的かつ外的な要因に起因する変動まで拾ってしまって，変化の傾向を見失ってしまう可能性がある。例えば，$\bar{x}_t$ を観測できない本質的な変動要因であるとして（例：企業の生産性が上がるから株価が上がる），観測可能なデータ x_t はあまり本質的でない要因 ε_t（例：政治スキャンダルで株価が下がる）によって撹乱されてしまうとする。

$$x_t = \bar{x}_t + \varepsilon_t$$

ε_t と異なる時点の s の ε_s には独立性が仮定される。$\mathrm{Var}(\varepsilon_t) \equiv \sigma^2$ と仮定しよう。本来，

$$\bar{x}_{t+1} - \bar{x}_t$$

を測りたいのだけども，観測できないものは測れない。しぶしぶ，$x_{t+1} - x_t$ を使うのだが，

$$x_{t+1} - x_t = (\bar{x}_{t+1} - \bar{x}_t) + (\varepsilon_{t+1} - \varepsilon_t)$$

$$\Longrightarrow \mathrm{Var}\left((x_{t+1} - x_t) - (\bar{x}_{t+1} - \bar{x}_t)\right) = 2\sigma^2$$

となり，誤差の分散が拡大してしまう。

撹乱項の影響を除去して，時系列の変化の傾向（**トレンド**）を見るために使うのが**移**

動平均である。例えば，2 期間前方移動平均は次のように定義する[1]。

$$x_t^{(2)} = \frac{x_{t-1} + x_t}{2}$$

分散にどのように影響するかを見てみよう。

$$\begin{aligned} x_{t+1}^{(2)} - x_t^{(2)} &= \frac{x_t + x_{t+1}}{2} - \frac{x_{t-1} + x_t}{2} \\ &= \frac{x_{t+1} - x_{t-1}}{2} \\ &= \frac{\bar{x}_{t+1} - \bar{x}_{t-1}}{2} + \frac{\varepsilon_{t+1} - \varepsilon_{t-1}}{2} \end{aligned}$$

$$\Longrightarrow \mathrm{Var}\left(\left(x_{t+1}^{(2)} - x_t^{(2)}\right) - \left(\frac{\bar{x}_{t+1} - \bar{x}_{t-1}}{2}\right)\right) = \frac{\sigma^2}{2}$$

移動平均を計算することで，本質的な変動要因 $\bar{x}$ の推定に関する不確実性を逓減できる。平均を取る期間の長さ（ウインドウ）を長くすれば，変動要因の影響をさらに小さくすることができる。ただし，1 つ注意が必要である。移動平均が推定しているのはあくまでも $\frac{\bar{x}_{t+1} - \bar{x}_{t-1}}{2}$ であり，$x_{t+1} - x_t$ でない。トレンドが線形である場合にはこれらは一致するが，トレンドにも循環的な傾向がある場合にはその循環傾向さえも打ち消してしまう。

この性質をうまく利用すると，季節的な変動を除去することができる。例えば，消費の四半期データはボーナス月を含むか含まないかが実測値に影響しそうである。知りたいことが長期的な傾向である場合には，季節的な要因を除去するために 4 期の移動平均を計算するとよい。

問題 5.3. デパートの売上のデータを毎日取っているとする。どのような周期の循環成分が見られるだろうか。除去するためにはどのような移動平均を取ればよいか。

[1] 「前方」の意味，前方移動平均以外のオプションは以下のように整理できる。
- t 期の移動平均の値を t 期以前のデータのみで計算するとき，前方移動平均。
- t 期の移動平均の値を t 期以後のデータのみで計算するとき，後方移動平均。
- t 期の移動平均の値を t 期を中心としたデータで計算するとき，中心移動平均。

Answer

5.3　理論: 成長と変動

マクロ経済学で最も重視される経済変数は実質 GDP である。実質 GDP は，経済で生産された総付加価値額から物価上昇を除去したものとして定義される。人口規模で標準化した「一人あたり実質 GDP」は，その国の個人が享受する物質的豊かさの基本指標として特に注目される。実質 GDP や一人あたり実質 GDP にはもちろん様々な原因で変化が起こるのだが，マクロ経済学では時間変化を大まかに次の 2 つのタイプに分けて分析する。

1. 経済成長。資本の蓄積，教育や技術の進歩によって生産性が向上し，経済規模が拡大する。
2. 経済変動。予期しない出来事によって経済活動が落ち込んだり，逆に，活発になること。景気循環とも言う。

5.3.1　長期と短期

経済指標の変化を「成長」と「変動」という観点で切り分けるときには，注目している時間のスパンに意識を向けるようにしよう。通常，成長は長い時間を掛けて起こる現象なので，長期的な視野で経済を眺めることが有効である。一方，変動に注目するときには短い時間の中での変化に関心があることが多い。

短期は長期の一部だし，短期の積み重ねが長期ではないか，と考えた人もいるかもしれない。日常言語を基準に考えれば，まさにその通りなのだけど，多くの経済理論はそ

のようには作られていない。長期と短期は別々の枠組みを使って考えなければならない。マクロ経済モデルにはおおまかな分類として，

- 長期のモデル
- 短期のモデル

の 2 つのタイプのモデルが存在する[*2]。この 2 つのタイプを切り分けるのは，時間の長さというよりも，市場の機能や性質に由来する違いである。

- 長期のモデルでは，市場が完全であり，需給調整が円滑に機能する。
- 短期のモデルでは，市場は不完全であり，需給調整の機能が制限されている。

特に次のような分け方をすることが多い。

- 長期のモデルでは，物価が伸縮的（需給に応じて速やかに変化）である。
- 短期のモデルでは，物価が硬直的（需給に応じて変化しない）である。

なぜこのような切り分けに対して，長期・短期という言葉を使うかというと，

- 長い期間を通して見れば市場はうまく機能しているように見える，
- 短い期間だけを見ると，市場にはいろいろな不備が生じている，

ということだろう。

長期では，財・サービス市場，要素市場，金融市場などすべての市場が需給一致の状態にある。したがって，働きたいと考えている労働者はすべて雇用されている状態にあるし，遊休状態にある資本も存在しない。実際の経済はこのような状態にはないが，ベンチマークとなる時間変化を示すときに使われる便利な想定である。

短期では，市場は物価の変化によって需給を一致させることができない。したがって，一時的な不均衡状態が生じる可能性が高い。失業の拡大によって一時的な厚生の悪

[*2] もちろん，このような分類は不完全なものだ。「成長」という長期の動向を扱う分析であっても，工業化以前の経済が工業化に至る飛躍的な成長を分析するモデルと，先進国の比較的安定した成長を記述するモデルは異なっている。

化が生じるので，政府介入の余地が生まれる。望ましくない景気循環に対抗するための「反循環的経済政策」の理解が短期分析の 1 つの目標になっている。景気後退を引き起こす外的な撹乱要因（ショック）が起こったときに，市場が完全に機能していれば経済主体の間の取引によって解消できたであろう問題であっても，市場が不完全であるときには大きな厚生悪化につながる可能性がある。このようなショックが，経済全体にどのように波及するか，そして，その問題にどのように対処するかというのが短期分析のテーマになっている。

5.3.2　完全雇用 GDP・潜在 GDP

さて，ここではシンプルに「短期には失業が存在する」という一点だけに注目しよう。失業が存在するということは，その経済では使える生産要素を使い切っていないということだ。したがって，経済が生産できる最大の量を生産していない。経済が生産できる最大の量を**完全雇用 GDP** と呼ぼう。なお，後で説明する潜在 GDP のことを完全雇用 GDP と呼ぶことも多いが，ここでは区別して使っている。長期のモデルでは，失業が存在しないという大胆な仮定をおいて分析することが多い。すなわち，完全雇用 GDP の時間変化・成長を見るのが長期分析の目的である。

短期モデルでは，失業が中心的な役割を果たす。実際の経済においても，労働市場の価格（賃金）には色々な原因で硬直性が存在すると考えられていて，それが失業の 1 つの原因になっている。例えば，物価が下落したときに名目賃金が十分下がらなければ，実質賃金は上昇する。実質賃金に反応する労働需要と労働供給は，超過供給の状態になる。これは働きたい人が働けない状態，つまり失業，が生じる。名目賃金の硬直性について，大抵のマクロ経済学の教科書には次の仮説を提示して説明している。

- インサイダー・アウトサイダー理論
- 効率賃金仮説

いずれも，賃金が市場を清算させる水準よりも高い水準に留まってしまう原因に関する仮説である。各自，標準的なマクロ経済学の教科書を読んで概要を調べておいてほしい。

また，市場が清算する水準に賃金が設定されていたとしても労働市場には失業が存在するという考え方もある。労働市場で取引されている財（労働力）は，食品や工業製品などのように均質ではないので，適材適所な組み合わせ（マッチング）を見つけるためにどうしても時間や費用がかかってしまうからだ。したがって，市場均衡価格がついているにも関わらず失業が存在する。企業が労働者に望む能力や技能と，求職者が持っている能力との差によって生じる失業を**構造的失業**，職探しのためにかかる時間が原因で起こる失業を**摩擦的失業**と呼ぶ[3]。構造的失業と摩擦的失業は切り分けることが難しいので，摩擦的・構造的失業というようにセットで扱われることが多い。

問題 5.4. 失業者に対して支払われる失業給付金は摩擦的・構造的失業を意図せず高めてしまう可能性がある。理由を検討しなさい。

Answer

摩擦的・構造的失業だけが残っているときの失業率を**自然失業率**という。自然失業率を離職と就職が均衡している状態の失業率と解釈することもできて，長期的には自然失業率の水準で失業が推移すると考えられている。言うまでもなく，現実の経済の失業率は自然失業率より高いときもあれば低いときもある。この差，特に失業の上乗せ部分のことを**循環的失業**という。摩擦的・構造的失業は労働市場の制度，慣行や時代背景によって変わるものなので，短期的な政策目標とはなりにくい。一方，循環的失業は景

[3] 本書では学部の標準的なマクロ経済学の教科書として『マンキュー マクロ経済学 I, II』を推奨しているが，ここで用いている失業に関する用語法はマンキューと異なっていることに注意しておく。マンキューの教科書では「実質賃金の硬直性によって生じる需給の不一致が原因の失業」を構造的失業と定義し，マッチングの違いによって生じる失業を摩擦的失業と定義している。

気後退によって生じた失業なので，政策介入によって解消することができるかもしれない。

自然失業率を達成している経済で生産されている量は**潜在 GDP**（あるいは完全雇用 GDP）と呼ばれる。その定義から，経済のベンチマークとなる生産量としてふさわしいことが分かるだろう。もちろん，潜在 GDP を直接観測できる訳ではないので，各国の中央銀行や政府機関が推計している。図 5.6 はアメリカの四半期実質 GDP（Real GDP）と潜在 GDP（Potential GDP）の推計値をプロットしたものである。縦軸を対数スケールにしているので，経済成長率は徐々に低下しつつも比較的順調な成長経路を辿っていることが分かる。長期のモデルによって行う経済成長の分析は，このような経済の右肩上がりの拡大を対象としている[4]。

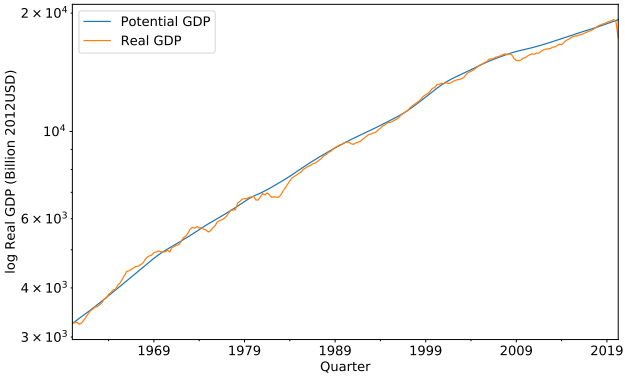

図 5.6: アメリカの実質 GDP と潜在 GDP（Source: FRED）

一方，実質 GDP と潜在 GDP の差に注目すると，いくつかの時期で大きな差が表れている。差を明確にするための変数を導入しよう。実質 GDP と潜在 GDP の差を **GDP ギャップ**という。下の図 5.7 は GDP ギャップを潜在 GDP で除した値（GDP ギャッ

[4] 完全雇用 GDP と潜在 GDP の違いを無視しているが，これらは非常に強く相関しているので大きな問題にはならないだろう。

プ率）

$$\text{GDP ギャップ率} = \frac{\text{実質 GDP} - \text{潜在 GDP}}{\text{潜在 GDP}}$$

をプロットしたグラフである。データは図 5.6 と同じものを使っている。正の値を取っている時期には経済のベンチマークとなるパフォーマンス（＝潜在 GDP）よりも大きな生産を行っているので，好景気（景気拡張期）を表している。逆に，負の値を取っている時期には経済は本来生産できる量よりも少ない量しか生産していない。したがって，この時期（色付き部分）には景気後退を経験している[5]。経済変動と経済変動に伴う厚生低下の是正に関する分析はマクロ経済の短期モデルの仕事である。

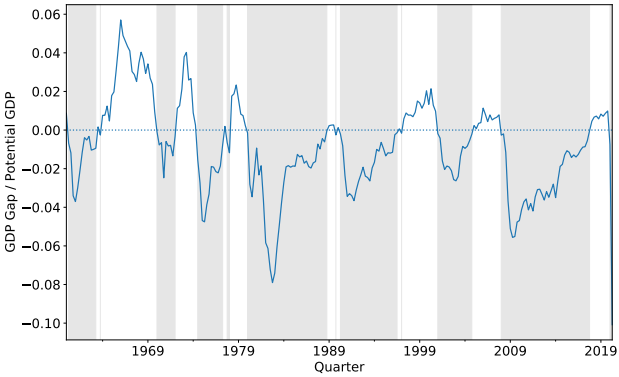

図 5.7: GDP ギャップ（Source: FRED）

失業に関するデータも見ておこう。図 5.8 は同じ時期の失業率と自然失業率（natural rate of unemployment) の推計値をプロットしたものである。色付き部分は図 5.7 と同様，GDP ギャップが負になる領域を表している。失業率が自然失業率の水準を超えて高くなっている時期と，GDP ギャップが負になる時期がおおよそ同じであることに

[5] ここでは，GDP ギャップの符号のみで景況判断をしているが，景気後退の終了前には「GDP ギャップは負だが景気はよい」という状況があり得る。したがって，実際の景況判断は実質 GDP の山と谷を見極めるという仕事になる。アメリカの景気の転換点は NBER（全米経済研究所）が発表している。

注意をしてほしい。もちろんこれは驚くようなことではなく，潜在 GDP の定義に由来する。

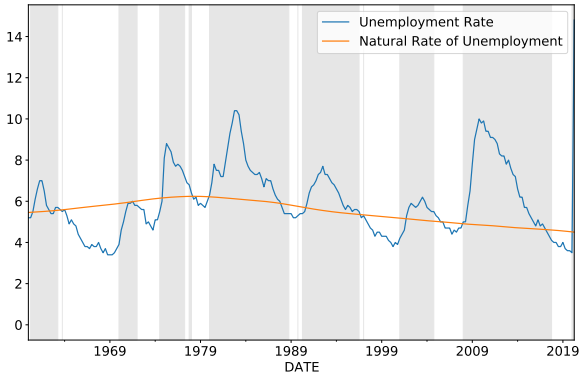

図 5.8: GDP ギャップ（Source: FRED）

5.3.3　フィリップス曲線

次にインフレーションと失業の関係を見てみよう。図 5.9 は PCE デフレーター（個人消費デフレーター）から計算したインフレ率の 12 期中心移動平均をプロットしたものである。色付き部分は図 5.7 と同様，GDP ギャップが負になる領域を表している。**色付き部分がインフレ率が減少している時期におおむね重なっている**ことに注意してほしい。つまり，

- GDP ギャップが正になるときには，インフレ率は上昇する傾向にある。
- GDP ギャップが負になるときには，インフレ率は低下する傾向にある。

つまり，

- GDP ギャップとインフレ率との間には正の相関がある。

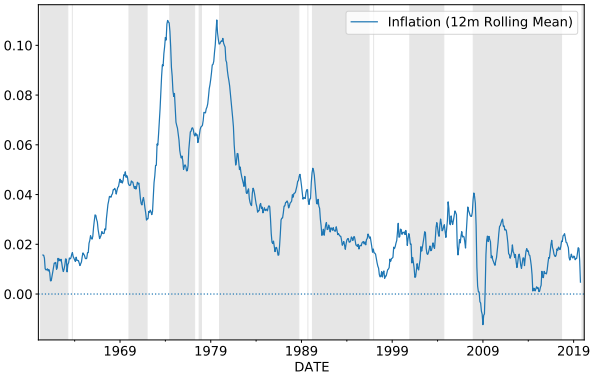

図 5.9: GDP ギャップとインフレーション（Source: FRED）

正の GDP ギャップを**インフレギャップ**，負の GDP ギャップを**デフレギャップ**と呼ぶ。GDP ギャップが正であるということは，その時期には基準となる水準より大きな生産活動が行われている。企業にとっては新しく雇用をして生産を拡大したい局面ではあるが，労働市場は逼迫しているから，賃金に上昇圧力が加わるだろう。消費者の所得も増えているので，上昇した雇用コストを生産物価格に転嫁したい企業は容易にそのようにできる。結果的にインフレ率は当初の予想水準よりも高めになる。逆に，GDP ギャップが負であるときには，企業は生産活動を抑えている。労働需要が小さく，賃金には下降圧力が加わる。所得は小さいので，生産物価格を上げにくい環境になっているし，企業はその必要性も感じていない。結果として，インフレ率は当初の予想水準よりも低くなる。このインフレ率に関する「当初の予想水準」は，**予想インフレ率**という。

長期的には市場は理想的な機能を果たし，情報の非対称性も解消される。長期的な経済の動向は次のようになる。

- 長期的には自然失業率が達成される。
- 長期的には潜在 GDP が生産される。
- 長期的には予想インフレ率は実際のインフレ率と一致する。

t 期のインフレ率を π_t,$(t-1)$ 期に形成された t 期のインフレ率に関する予想を $\mathbb{E}_{t-1}\pi_t$, 実質 GDP を Y_t, 潜在 GDP を $\bar{Y}_t$ とすると，前述の関係は次のように表現できる。

$$\pi_t - \mathbb{E}_{t-1}\pi_t = \alpha\left(Y_t - \bar{Y}_t\right), \qquad \alpha > 0$$

この関係式は（修正）**フィリップス曲線**である。賃金上昇率と所得の関係を表す。マクロ経済の物価水準と供給水準の関係を決定する方程式で，AS 曲線（Aggregate Supply）とも呼ばれている。

5.4　プログラミング: 準備

5.4.1　Conda による追加ライブラリのインストール

この章のプログラムでは，**pandas-datareader** というライブラリを用いる。インスールされていない可能性があるので，Anaconda プロンプトで次のコマンドを実行してインスールしておく。

```
conda install pandas-datareader
```

IPython からこのコマンドを実行するには次のようにする。

```
!conda install pandas-datareader
```

5.4.2　Python の基本型

文字列

すでに何度も使っているが「**文字列型**」について確認しておこう。文字列は文字の並びである。シングルクォーテーションかダブルクオーテーションのペアで囲むことで作成する。

```
s1 = "Hello."
s1
```

```
'Hello.'
```

日本語の文字も登録できる。

```
s2 = 'こんにちは'
s2
```

```
'こんにちは'
```

リストと同じ記法で文字列の一部を取り出すことができる。

```
s1[0]
```

```
'H'
```

```
s1[1:3]
```

```
'el'
```

```
s1[-1]
```

```
'.'
```

list() 関数を使うと明示的にリストに変換できる。

```
list(s2)
```

```
['こ', 'ん', 'に', 'ち', 'は']
```

このような変換が必要なときに自動的に行われていると考えてよい。文字列の中に含まれる各文字を走査する for ループを書くこともできる。

```
for i, s in enumerate(s1):
    print(s * (i + 1))
```

```
H
ee
lll
llll
ooooo
......
```

問題 5.5. enumerate() の使い方を調べて説明しなさい。

Answer

f-string という便利な記法がある。f-string を使えば文字列定義の中で式を使うことができる。次の使用例を見れば使い方は明らかだろう。クォーテーションマークの前の `f` と中の波括弧部分に注目する。

```
x, y = 10, 20
f"{x} + {y} = {x + y}"
```

```
'10 + 20 = 30'
```

文字列フォーマット用の記法を組み合わせると，小数点以下の表示桁数や，右寄せ左寄せなどのコントロールが可能になる[6]。

```
f"{x} / ({x} + {y}) ≒ {x / (x + y):.2f}"
```

```
'10 / (10 + 20) ≒ 0.33'
```

辞書

辞書（dictionary）というデータ構造が非常によく用いられるので覚えておこう。普通の辞書は「見出し語」と「定義」の多数の組合せから構成されている。Python の辞書も同じような構成になっている。「キー」と「値」のペアが辞書である。次の辞書 `d` には 3 つのキー,`1`,`'a'`,`'x'`, が登録されている。対応する値はそれぞれ，`100`,`200`,`[1, 2]` である。

```
d = {1: 100, 'a': 200, 'x': [1, 2]}
d
```

[6] 詳細は `https://docs.python.org/3/library/string.html#format-string-syntax` を参照。

```
{1: 100, 'a': 200, 'x': [1, 2]}
```

値はどのような型のオブジェクトでも割り当てることができる。一方，キーの方には制約があって，「不変（immutable）」なオブジェクトのみを割り当てることができる。例えば，整数，浮動小数点数，文字列やタプルは不変なオブジェクトである。リストは不変なオブジェクトではないので，キーにすることはできない。

問題 5.6. 「リスト」をキーとする辞書を定義しようとすると，どのようなエラーが発生するか。実験して結果を記録しなさい。

Answer

角括弧の中にキーを入れると値を読み出せる。キーは変数に格納していても問題ない。

```
d[1]
```

```
100
```

```
k = 'a'
d[k]
```

```
200
```

辞書をループするとすべての「キー」を走査する。この例では登録順に表示されているが，これは必ずしも保証されていない[7]。

```
for key in d:
    print(key)
```

```
1
```

[7] 順序が意味をもつ場合には OrderedDict 型を用いる。詳細は https://docs.python.org/3/library/collections.html#collections.OrderedDict

```
a
x
```

キーと値の組合せについてループしたいときには，items() メソッドを使うとよい。

```
for k, v in d.items():
    print(f"{k} : {v}")
```

```
1 : 100
a : 200
x : [1, 2]
```

5.4.3　Pandas 入門

Python で表形式のデータを扱うときは **pandas** というライブラリを用いるのが標準的である。**NumPy** の配列をベースに設計されている。配列との主要な違いは，

- 異なるデータ型を持つ列の混在が許される。
- 行・列に意味のあるラベル（インデックス）を付けることができる。
- その他データの加工・可視化を便利にする機能が追加されている。

4×3 行列をもとに **pandas** の DataFrame オブジェクトを作ってみよう。インデックスを付けずに作る場合は次のようになる。

```
import pandas as pd
x = np.arange(12.0).reshape(4, 3)
pd.DataFrame(x)
```

	0	1	2
0	0.0	1.0	2.0
1	3.0	4.0	5.0
2	6.0	7.0	8.0
3	9.0	10.0	11.0

多くの場合，次のようにインデックスが付けられている。

```
frame = pd.DataFrame(x, columns=['a', 'b', 'c'],
                     index=pd.PeriodIndex(range(2000, 2004), freq='A'))
frame
```

```
          a     b     c
2000   0.0   1.0   2.0
2001   3.0   4.0   5.0
2002   6.0   7.0   8.0
2003   9.0  10.0  11.0
```

DataFrame の定義にはリスト様のオブジェクトを値に持つ辞書を用いることもできる。

```
pd.DataFrame({'a': [0.0, 3.0, 6.0, 9.0],
              'b': [1.0, 4.0, 7.0, 10.0],
              'id': list('xyzw')},
             index=range(2000, 2004))
```

```
          a     b id
2000   0.0   1.0  x
2001   3.0   4.0  y
2002   6.0   7.0  z
2003   9.0  10.0  w
```

pandas の DataFrame オブジェクトには時系列データを扱うための便利なメソッドが定義されている。差分を計算する diff() や変化率を計算する pct_change() はその一例である。これらのメソッドを使うを 1 行目のデータが欠測値（NA）となるので，dropna() メソッドをつなげて削除することが多い。

```
frame.diff()
```

```
          a    b    c
2000   NaN  NaN  NaN
2001   3.0  3.0  3.0
2002   3.0  3.0  3.0
2003   3.0  3.0  3.0
```

```
frame.pct_change().dropna()
```

```
          a          b      c
2001   inf   3.000000  1.500
2002   1.0   0.750000  0.600
2003   0.5   0.428571  0.375
```

DataFrame オブジェクトの列を取得するのにはいくつかの方法があるが，列名を指定して呼び出す 2 つの方法を確実に覚えておこう。単一の列をけ呼び出す場合には表示の見た目が変わっていることに注意しよう。Series オブジェクトと呼ばれるオブジェクトになっている。要するに，DataFrame は Series を列方向に並べたものだ。これらの違いは，**NumPy** の 2 次元配列と 1 次元配列の違いと考えておけばよい。

```
frame.a
```

```
2000    0.0
2001    3.0
2002    6.0
2003    9.0
Freq: A-DEC, Name: a, dtype: float64
```

```
frame['b']
```

```
2000     1.0
2001     4.0
2002     7.0
2003    10.0
Freq: A-DEC, Name: b, dtype: float64
```

```
frame[['a', 'c']]
```

```
          a     c
2000  0.0   2.0
2001  3.0   5.0
2002  6.0   8.0
```

```
2003   9.0   11.0
```

行を取得するには次のようにインデックスのスライス記法を用いる。

```
frame.loc['2000':'2002', :]
```

```
          a     b     c
2000   0.0   1.0   2.0
2001   3.0   4.0   5.0
2002   6.0   7.0   8.0
```

pandas を使えば列を追加するのも簡単だ。

```
frame['d'] = frame.b * frame.c + 10
frame
```

```
          a      b      c       d
2000   0.0    1.0    2.0    12.0
2001   3.0    4.0    5.0    30.0
2002   6.0    7.0    8.0    66.0
2003   9.0   10.0   11.0   120.0
```

データを **pandas** の DataFrame にしておくと簡便な記法で可視化ができる。基本の plot() メソッドを使うと折れ線グラフが描ける（図 5.10）。

```
frame.plot(y = ['a', 'd'])
plt.show()
```

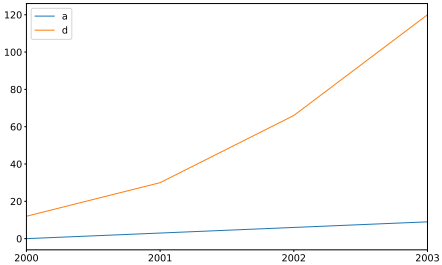

図 5.10: **pandas** のプロット

散布図を描くには `plot.scatter()` を使う。図 5.11 はマーカーのサイズをデータ依存にさせる，いわゆるバブルチャートである。

```
frame.plot.scatter(x='a', y='d', s=10 * frame.b ** 2)
plt.show()
```

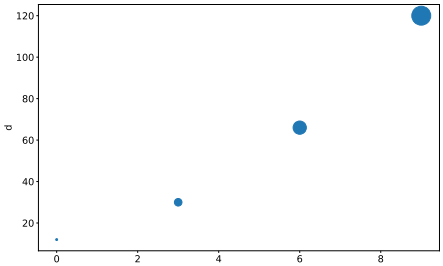

図 5.11: **pandas** の散布図

`plot.bar()`,`plot.barh()` を使うと棒グラフを描写できる。結果はそれぞれ図 5.12 と図 5.13 のようになる。`stacked=True` とすれば積み上げ棒グラフになる。水平方向にグラフを伸ばす方法も紹介しておこう。

```
frame.plot.bar()
plt.show()
```

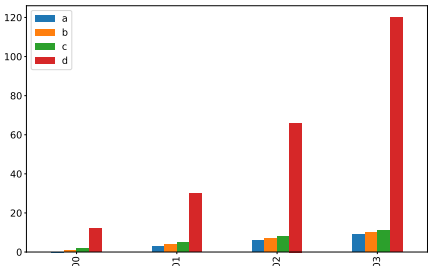

図 5.12: **pandas** の棒グラフ

```
ax = frame.plot.barh(stacked=True)
```

```
ax.invert_yaxis()
plt.show()
```

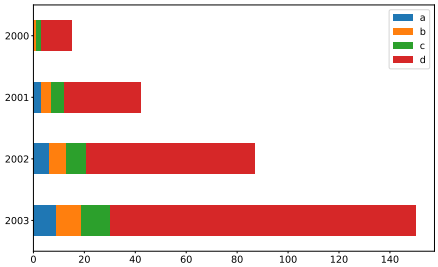

図 5.13: **pandas** の積み上げ棒グラフプロット

5.5　プログラミング: データを眺める

さて，本章プログラミングパートの本題に入ろう。

「オープンデータ」という標語のもとに多くの公的組織・私的組織が広く利用可能なデータをインターネット上で公開している。ウェブサイトに Excel ファイルや CSV ファイルを置いているだけのものから，機械可読が容易な形式で配布されているもの，プログラム上で取得ができるような特別な配慮がなされたものまで様々である。

5.5.1　API

一般にソフトウェア間で通信を行う方式のことを API (Application Programming Interface) と呼ぶ。データ提供者が API（Web API）を提供している場合，プログラマは API の規則に適合したコードを書くことで，プログラム的にデータを取得することができる。API には登録不要で利用できるものから，事前に利用者情報を登録した上で API キーや API トークンといった認証情報を取得しておく必要があるものもある。プログラマは必要な認証情報を取得した後，データ提供者のウェブサーバーに API リクエストを送信する。API リクエストは HTTP (HyperText Transfer Protocol) という方

式で行われる。大雑把に言えば，ウェブブラウザのアドレスバーにアドレスを打ち込むようなイメージである。ただし，ブラウザは使わない（使ってもいいけど）。API リクエスト用の「アドレス」には必要なデータは何かを説明するためのテキスト（「クエリ」）を付加する。データ提供者のサーバーがデータ要求を受け取ると，必要なデータをデータベースサーバーから取得する。API リクエストのクエリを使用しているデータベースに対するクエリに変換し，出力結果を適切にフォーマットしてプログラマに送り返す。プログラマは XML や JSON, CSV といった形式でフォーマットされたテキスト情報としてデータを受け取る[*8]。

すべてのデータ提供者が同じ形式でデータを提供していれば話は簡単なのだが，実際にはデータ提供者ごとに

- クエリ記法が違う，
- 出力データのフォーマット方法が違う

という問題がある。個別の API のドキュメントを読んで使用方法を調べるというのがデータ利用の最初の難関である。かなり早い段階で難関がやってくるので，ここで諦めてしまう人も多いだろう。

5.5.2　pandas-datareader

データ提供者ごとに API の規則が異なっていることがデータ利用を難しくしているのであれば，API の差異を吸収して共通の記法でデータを取得できるようにすれば，データ活用の利便性は格段に向上するだろう。**pandas-datareader** というライブラリはこのような目的で作られている。プログラマが直接 API リクエストを発行する代わりに，**pandas-datareader** に対して，標準化された記法でリクエストを送る。**pandas-datareader** はデータ提供者ごとのクエリ記法に変換し，データの取得を行う。

[*8] XML (eXtensible Markup Language) は HTML (HyperText Markup Language; ウェブページを書くときの記法）に似た記法でデータとメタデータを記録する方法。JSON (JavaScript Object Notation) は JavaScript (ウェブページの動的処理で用いられるプログラミング言語) のオブジェクト記法を用いたデータ記法。Python の辞書のような書き方をする。CSV (Comma Separated Values) は表形式のデータをコンマと改行で区切るだけのシンプルな表現である。

取得されたデータはデータ提供者が使用しているフォーマット方式から **pandas** のデータフレームに変換する。

pandas-datareader を使うとインターネット上で公開されているデータをダウンロードして **pandas** のデータフレーム形式に変換する作業が簡単になる。利用可能なデータは公式の安定ドキュメントで確認しておいてほしい。

- `https://pydata.github.io/pandas-datareader/remote_data.html`

インポートして実際に使ってみよう。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as pdr
```

FRED (St. Louis FED)

FRED はセントルイス連邦準備銀行が提供する経済時系列データのデータベースである。アメリカのデータだけでなく世界各国の情報も手に入れることができる。ウェブサイト `https://fred.stlouisfed.org/` にもデータのダウンロードや可視化のツールが一通り揃っているので，ぜひ試してほしい。

さて，ウェブサイトの検索窓で「GDP」と検索すると，1 つ目の結果は「Real Gross Domestic Product」となっている。実質 GDP のことである。これを開くと図 5.14 のような見出しと，グラフが表示されるはずだ。Python で処理するために必要な情報は図の赤枠で囲った部分の小さなコード「GDPC1」である。同じようにして潜在 GDP「Potential GDP」も検索してコードを確認しておこう。

次のコードが使用例である。`pdr.get_data_fred()` で FRED のデータを取得できる[9]。1 つ目の引数としてリスト `['GDPPOT', 'GDPC1']` を書いて，ダウンロード対象が GDPPOT（潜在 GDP）と GDPC1（実質 GDP）であることを指定する。結果に `gdp` という名前を付けた。これは **pandas** のデータフレームになっている。

[9] 多くのデータソースに対して `pdr.get_data_*()` という関数が用意されている。

図 5.14: FRED のウェブサイトでデータコードを確認する

```
gdp = pdr.get_data_fred(['GDPPOT', 'GDPC1'],
                        start='1960-01-01', end='2020-04-01')
gdp
```

```
                GDPPOT       GDPC1
DATE
1960-01-01   3247.590212   3275.757
1960-04-01   3280.322655   3258.088
1960-07-01   3313.239234   3274.029
1960-10-01   3345.865071   3232.009
1961-01-01   3378.938041   3253.826
...                  ...        ...
2019-04-01  18885.480000  19020.599
2019-07-01  18976.490000  19141.744
2019-10-01  19065.580000  19253.959
2020-01-01  19153.980000  19010.848
2020-04-01  19242.040000  17302.511

[242 rows x 2 columns]
```

データフレームに対して実行できる処理はすぐに使える。例えば，図を描くのも簡単だ。

```
gdp[['GDPPOT', 'GDPC1']].plot()
plt.show()
```

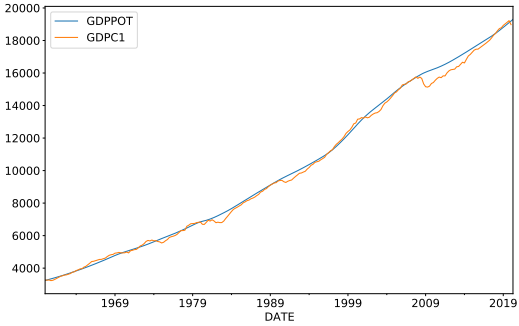

図 5.15: FRED で取得したデータ

問題 5.7. GDP ギャップと GDP ギャップ率を計算して，gdp データフレームに新しい列 (gap, rgap) として追加しなさい。結果は次のようになる。

```
                GDPPOT      GDPC1          gap      rgap
DATE
1960-01-01   3247.590212   3275.757    28.166788  0.008673
1960-04-01   3280.322655   3258.088   -22.234655 -0.006778
1960-07-01   3313.239234   3274.029   -39.210234 -0.011834
1960-10-01   3345.865071   3232.009  -113.856071 -0.034029
1961-01-01   3378.938041   3253.826  -125.112041 -0.037027
...                  ...        ...          ...       ...
2019-04-01  18885.480000  19020.599   135.119000  0.007155
2019-07-01  18976.490000  19141.744   165.254000  0.008708
2019-10-01  19065.580000  19253.959   188.379000  0.009881
2020-01-01  19153.980000  19010.848  -143.132000 -0.007473
```

```
2020-04-01  19242.040000  17302.511 -1939.529000 -0.100796

[242 rows x 4 columns]
```

問題 5.8. GDP ギャップ率をプロットしなさい。

次に，インフレ率を見るために Consumer Price Index for All Urban Consumers: All Items in U.S. City Average (CPIAUCSL) を取得してみよう。これは消費者物価指数の月次データである。

```
price = pdr.get_data_fred('CPIAUCSL',
                          start='1960-01-01', end='2020-04-01')
price
```

```
            CPIAUCSL
DATE
1960-01-01    29.370
1960-02-01    29.410
1960-03-01    29.410
1960-04-01    29.540
1960-05-01    29.570
...               ...
2019-12-01   258.203
2020-01-01   258.687
2020-02-01   258.824
2020-03-01   257.989
2020-04-01   256.192

[724 rows x 1 columns]
```

インフレ率を見るために pct_change() メソッドを用いる。月次のデータなので 12 倍して年率換算しておこう。

```
price['inflation'] = price.CPIAUCSL.pct_change() * 12
price.inflation.plot()
```

```
plt.show()
```

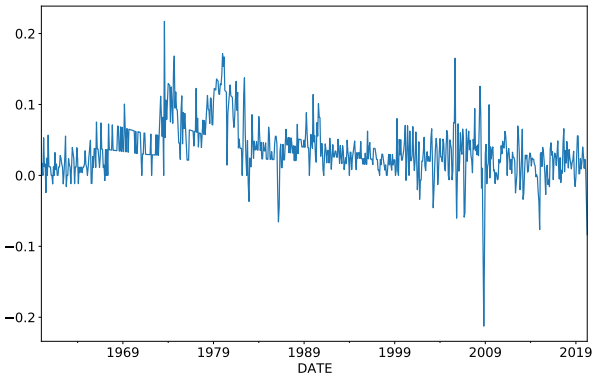

図 5.16: CPIAUCSL (Source: FRED)

変動が激しいので移動平均を計算してみよう。`rolling()` メソッドでグループ化して，グループごとに平均を取るとよい。

```
price['ma3'] = price.inflation.rolling(3, center=True).mean()
price['ma9'] = price.inflation.rolling(9, center=True).mean()
price
```

	CPIAUCSL	inflation	ma3	ma9
DATE				
1960-01-01	29.370	NaN	NaN	NaN
1960-02-01	29.410	0.016343	NaN	NaN
1960-03-01	29.410	0.000000	0.023129	NaN
1960-04-01	29.540	0.053043	0.021743	NaN
1960-05-01	29.570	0.012187	0.027154	NaN
...	...	...	...	...
2019-12-01	258.203	0.009954	0.020172	0.001446
2020-01-01	258.687	0.022494	0.012934	NaN
2020-02-01	258.824	0.006355	-0.003288	NaN

```
2020-03-01    257.989   -0.038714  -0.038648        NaN
2020-04-01    256.192   -0.083585         NaN        NaN

[724 rows x 4 columns]
```

最後にプロットしてみよう。移動平均を取る期間（ウインドウ）が長くなるほど，なめらかなグラフになっていることを確認してほしい。

```
ax = price.inflation.plot(alpha=0.2)
price[['ma3', 'ma9']].plot(ax = ax)
plt.show()
```

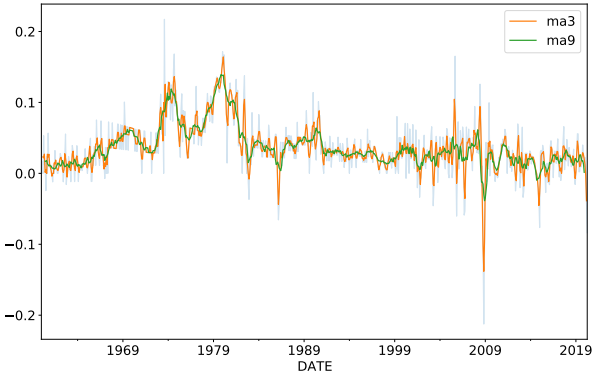

図 5.17: CPIAUCSL の移動平均 (Source: FRED)

World Bank

国際比較をしたいときには，世界銀行のパネルデータを用いることができる。1960 年から 2010 年までの一人あたりの名目 GDP のデータ（米ドル換算）をダウンロードするコードは以下のように書ける。FRED とはかなり使い方が異なっているので注意しよう。`from pandas_datareader import wb` というコードで世界銀行のデータベースにアクセスするための特別なモジュール `wb` を使えるようにしている。

```
from pandas_datareader import wb
gdp = wb.download(indicator='NY.GDP.PCAP.CD', country='all',
                  start=1960, end=2010)
gdp
```

```
                    NY.GDP.PCAP.CD
country     year
Arab World  2010      5926.712963
            2009      5180.581491
            2008      6149.511489
            2007      4963.224710
            2006      4360.335052
...                           ...
Zimbabwe    1964       281.558440
            1963       277.479715
            1962       276.688781
            1961       280.828951
            1960       278.813699

[13464 rows x 1 columns]
```

いわゆる「縦持ち」のデータになっているので国ごとに列を分けておこう。**pandas** を使ったデータ加工の詳細は McKinney(2018) などを参考にしてほしい。あとで可視化をするので，結果が見やすくなるように対数変換もしている。

```
gdp_pivot = gdp.reset_index()
gdp_pivot['NY.GDP.PCAP.CD'] = np.log(gdp_pivot['NY.GDP.PCAP.CD'])
gdp_pivot = gdp_pivot.pivot(index='year', columns='country',
                            values='NY.GDP.PCAP.CD')
gdp_pivot.index = gdp_pivot.index.astype('uint64')
```

出来上がった表は非常に大きいので出力は省略する。手元のコンピュータで確認してほしい。

```
gdp_pivot
```

4 年分のデータの密度プロットを表示する。この図はヒストグラムをなめらかにしたもので，横軸は 1 人あたり名目 GDP の対数値，縦軸はその GDP 水準に所属する国の頻度を表している。山が高いほどその水準の GDP を獲得する国が多い。

```
years = [1960, 1980, 2000, 2010]
fig, ax = plt.subplots(1)
for year in years:
    gdp_pivot.loc[year].dropna().plot.density(ax=ax)

ax.legend(years)
ax.set_xlabel('Log GDP per capita')
plt.show()
```

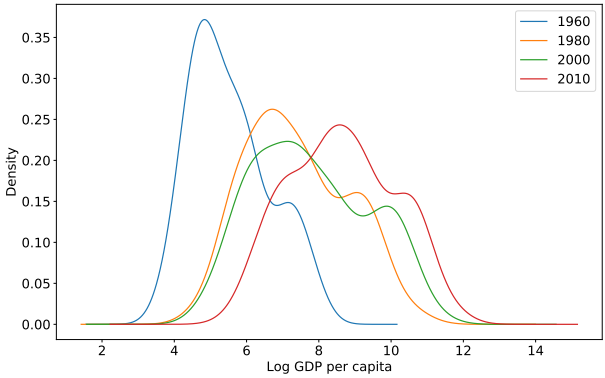

図 5.18: CPIAUCSL の移動平均 (Source: FRED)

グラフから次の傾向が読み取れることを見てほしい。

- 密度のグラフが右方向に移動している。つまり，平均的に見れば時代の経過とともに，多くの 1 人あたり GDP を生産するようになる。これは経済成長を意味

している。

- 密度のグラフが徐々に広がってきているように見える。これは，上位の国と下位の国の間での所得格差（国内の格差ではないことに注意）が広がっていることを意味している。ただし，この図だけでは格差についての正確な議論はできない。

問題 5.9. 1 人あたり名目 GDP をダウンロードするときに `NY.GDP.PCAP.CD` というコードを使った。World Bank Open Data`https://data.worldbank.org/` のウェブサイトを閲覧して，このデータを検索しなさい。ページのどの場所にコードが書かれているか。検索ボックスから他の指標についても調べてコードを探し，**pandas-datareader** でダウンロードしてみなさい。

Answer

参考文献

McKinney, Wes, 雅人 瀬戸山, 儀匡 小林, and 開資 滝口 (2018) *Python* によるデータ分析入門 第 2 版: オライリー・ジャパン.