実証会計・ファイナンスの勉強ノート

Soichi Matsuura

2026-01-01

Table of contents

第 1 章

はじめに

このノートは、大阪大学経済学研究科のファイナンス学者の笠原晃恭先生と会計学者の村宮克彦先生が書かれた「**実証会計・ファイナンス**」新世社に関する学習内容をまとめたものです。本書は、日経ストックリーグに参加する学生が、R を駆使して分析パートを作成できるように作られており、会計学とファイナンスの基礎知識を付けた後で、R の基本を学び、財務分析や投資理論を R で実装するための方法を学習できる、非常に有用な教科書です。ぜひ本書を手に取って、企業のデータ分析を自分の手で実践してみてください。

1.1 準備

各自の PC(Windows か Mac を想定) の好きな場所に `Kasahara_Muramiya` というフォルダを作成し，そこを作業ディレクトリにしていることを想定しています。この文章の意味が分からない人は，R の入門書を確認してください。作業ディレクトリの中に `data` というフォルダを作成し，そこにテキストで使うデータファイルを保存してある，という前提で進めます。

第 2 章

ファイナンス入門

```
pacman::p_load(
  tidyverse, # 便利なパッケージ群
  ggthemes, # ggplot2 のテーマ集
  plotly, # インタラクティブなグラフ作成
  patchwork # 複数のグラフをまとめて表示
)

mystyle <- list(
  # 1. ggthemes 自体の引数でフォントを指定します
  theme_economist_white(
    # gray_bg = FALSE,
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),

  # theme_calc(base_family = "HiraKakuProN-W3"),
  # scale_colour_calc(),

  # 3. その他の微調整（フォントサイズなど）
  theme(
    text = element_text(size = 12), # フォントファミリーは上で指定済みなの
    ↪  で省略可
    # 必要であれば個別の要素を調整
    # plot.title = element_text(hjust = 0.5),
    axis.title = element_text(size = 12)
  )
)
```

2.1　割引率

ファイナンス (finance) の大事な概念に**割引率** (discount rate) があります。これは、ある財の今の価値と将来の価値は異なっており、将来の価値を現在の価値 (これを**現在価値** (present value) という) に変換するための係数です。

たとえば，今の 100 万円が来年 110 万円になる世界において，

$$100 = f(110)$$

と表せる関数 f が存在するとします。このとき，110 万円を現在価値に変換するための係数が割引率です。具体的に，

$$f(110) = \frac{110}{1+r}$$

と表せるとき，r が割引率です。

$$100 = \frac{110}{1+r}$$
$$r = \frac{110}{100} - 1 = 0.1$$

この場合，割引率 r は 0.1(10%) となります。割引率 10% の世界において，今の 100 万円は 1 年後の 110 万円と同じ価値をもつ，ということを意味しています。

2.1.1　確実なキャッシュ・フローに対する割引率

- **現在価値**：将来に発生するキャッシュフローの現時点における価値
- **時間価値**：将来と現在の価値の違い
- **無リスク金利** (risk-free rate)：安全資産へと投資したときのリターンで，投資時点でリターンの額が確定します。無リスク金利は**確率変数ではなく定数** (parameter) です。

上の式を年を T という記号で表して一般的に表現します。T 年後に**確実に**獲得できるキャッシュフローを CF_T で表し、無リスク割引率を R_F で表します[*1]。このとき現在価値 PV は以下のように計算されます。

$$PV = \frac{CF_T}{(1+R_F)^T}$$

将来の確実なキャッシュ・フロー CF_T の現在価値は、割引率 R_F と将来受け取る時点である T に依存している。$CF_T = 100$ の場合、R_F と T の変化に応じて現在価値がどのように変化するのか確認する。

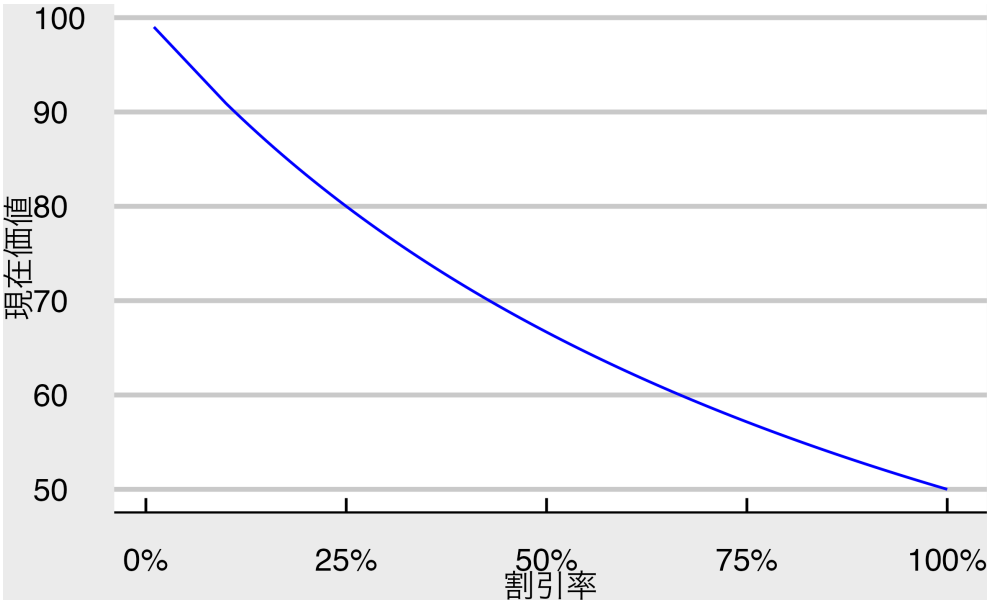
現時点で CF_0 を 1 年間貯金する。利息 r は 0.1 とする。1 年後に受け取れるキャッシュフロー CF_1 は、貯金した元本 CF_0 と利息 $CF_0 \times 0.1$ となる。つまり、$CF_0 + CF_0 \times 0.1 = (1+0.1)CF_0$ である。
逆に、来年 CF_1 受け取るためには、今いくら貯金するかを考える。必要な貯金額を X とすると、$X \times (1+0.1) = CF_1$ となる。つまり $X = CF_1/(1.1)$ となる。これが現在価値である。

[*1] 無リスク割引率は無リスク金利と同義で，全くリスクのない確実に手に入れられる利息のようなものです。

$T = 1$ として、横軸を割引率、縦軸を現在価値としたグラフが以下のものである。割引率が大きくなるにつれて現在価値が小さくなることが分かる。

Listing 2.1 現在価値と割引率

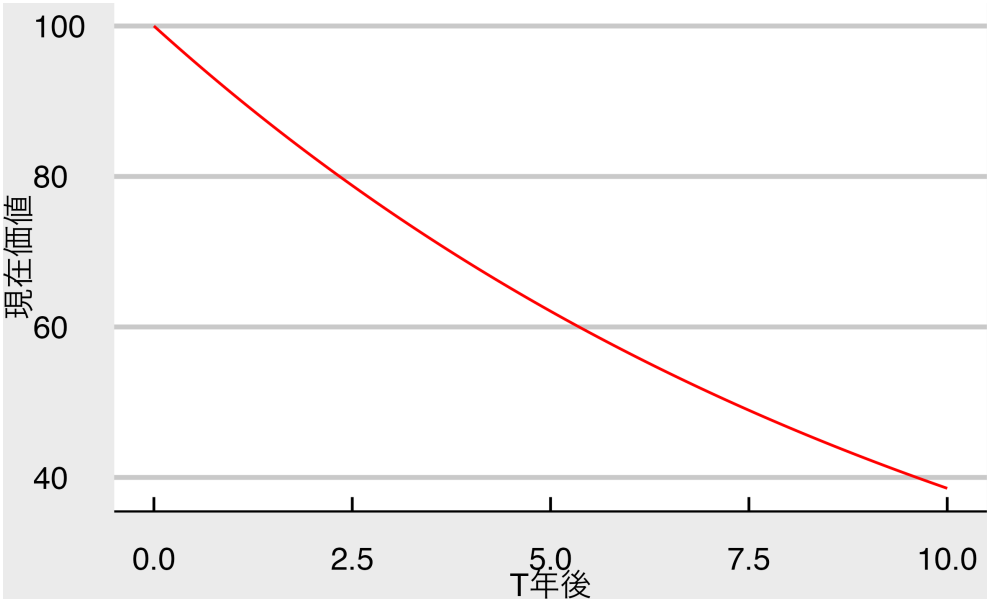
```
T_fixed <- 1 # 1 年後に固定

data.frame(
  R_F = c(0.01, seq(0.1, 1, by = 0.01))
  ) |>
  mutate(
    PV = 100 / (1 + R_F)^T_fixed
  ) |>
  ggplot() + aes(x = R_F, y = PV) +
  geom_line(color = "blue") +
  scale_x_continuous(labels = scales::percent) + # 軸を % 表示に
  labs(x = "割引率", y = "現在価値") +
  mystyle
```



次に、$R_F = 0.1$ に固定して、横軸を年 T、縦軸を現在価値 PV としてグラフが以下のものです。図を見ると，キャッシュ・フローを受け取る時点が遠くなるほど、現在価値が小さくなることがわかります。1 年後に受け取れる 100 万円と，10 年後に受け取れる 100 万円では，後者の方が価値が低い，というのは直観的ですよね。

Listing 2.2 現在価値と期間

```
R_F <- 0.1 # 無リスク利子率
T <- c(seq(0, 10, by = 0.1)) # 将来受け取る時点
PV <- 100 / (1 + R_F)^T # 現在価値の計算
df <- data.frame(T, PV) # データフレーム作成
# 作図
df |>
  ggplot() +
  aes(x = T, y = PV) +
  geom_line(color = "red") +
  xlab("T 年後") + ylab("現在価値") + mystyle
```



2.1.2　不確実なキャッシュ・フローに対する割引率

モデルに投資家の**リスクプレミアム** (risk premium; RP) を反映させることが割引率の 2 つ目の役割です。通常、将来キャッシュ・フローがいくらになるのか分からないのが普通であり、CF は様々な要因で変化する**確率変数** (random variable) であるため、現時点における**期待値** (expected value) で評価されます。

たとえば，1 年後に確実に 150 万円もらえる投資 A と，確率 50% で 200 万円，確率 50% で 100 万円もらえる投資 B があるとします。期待値で評価すると，投資 A の期待キャッシュフローは 150 万円，投資 B の期待キャッシュフローは $0.5 \times 200 + 0.5 \times 100 = 150$ 万円となり，同じ期待キャッシュフローをもつことになります。しかし，直観的に投資 B は 100 万円しか生み出さないリスクがある分，2 つの投資の価値を現在の価値で比較する

と，投資 A の方が価値が高いと思いませんか？

このリスクのある資産 B の割引率を $\tilde{R}$ と表すと，2 つの投資の現在価値は，次のようになります。

$$\frac{150}{1+R_F} \quad > \quad \frac{200\times 0.5+100\times 0.5}{1+\tilde{R}}$$

となるなら，

$$R_F < \tilde{R}$$

となります。この R_F と $\tilde{R}$ の差が**リスクプレミアム** (risk premium: RP) です。リスクプレミアムは、リスクのある資産に投資することに対する追加的な見返りを意味します。

$$RP \equiv \tilde{R} - R_F$$
$$R_F + RP = \tilde{R}$$

リスクプレミアムは割引率の調整で定量化されます。

$$PV_0 = \frac{\mathbb{E}_{\mathbb{0}}[CF_1]}{1+R_F+\underbrace{(\tilde{R}-R_F)}_{\text{RP}}} = \frac{\mathbb{E}_{\mathbb{0}}[CF_1]}{\underbrace{1+\tilde{R}}_{\text{リスク調整済み割引率}}}$$

割引率は時間価値やリスクプレミアムに関する定量的な情報を含むので、タイミングやリスクの異なるキャッシュフローを現在価値という同一の尺度で評価できます。

ここで、期待値をとる演算子 (operator) を $\mathbb{E}$ で表し、期待値をとる時点を添え字で表す。ここでは現時点 $t=0$ における期待値を $\mathbb{E}_0$ と表現している。たとえば，現時点を $t=0$ として，1 期先に起こりうる結果 X が 100 か 200 であることが分かっていて，それぞれの発生確率が 50% であったとする。この将来に起こりうる結果を現時点での情報を基に期待値をとる，ということは，

$$\mathbb{E}[X] = 0.5\times 100 + 0.5\times 200 = 150$$

となる。このように起こりうる結果と発生確率を掛けて足したものを**期待値**という。

2.1.3 NPV 法

先ほど学習した割引現在価値を判断の基準に，投資意思決定を行う方法を **NPV 法** (Net Present Value Method) といいます。いま，投資するかどうかを決めるタイミングを時点 0($t=0$) とします。キャッシュフローは CF_t で表し，マイナスなら支出，プラスなら収入を意味します。将来キャッシュフローは不確実であるため，現時点での期待値 $\mathbb{E}_{\mathbb{0}}[CF_t]$ で評価します。ちなみに CF_0 は時点 0 での投資なので，マイナスとなります。

投資時点から T 年間にわたって毎年 $\mathbb{E}[CF_t], \quad t=1,\ldots,T$ の期待キャッシュフローが生み出されるなら、それらを割引率 $1+\tilde{R}$ で現在価値に直して足し合わせた値 (つまり NPV) が、現時点で評価したプロジェクトの成果となります。

NPV はそのプロジェクトから発生するすべてのキャッシュフローの現在価値として解釈できます。コーポレートファイナンスでは，

- NPV がゼロ以上のプロジェクトは投資を実行
- NPV が負のプロジェクトは投資を見送る

ことを推奨しています。NPV 法を一般的に表現すると次のようになります (p.48)。

$$\begin{aligned} NPV_0 &= \frac{CF_0}{(1+\tilde{R})^0} + \frac{\mathbb{E}[CF_1]}{(1+\tilde{R})^1} + \cdots + \frac{\mathbb{E}[CF_T]}{(1+\tilde{R})^T} \\ &= CF_0 + \sum_{t=1}^{T} \frac{\mathbb{E}[CF_t]}{(1+\tilde{R})^t} \end{aligned}$$

$x^0 = 1$ であることに注意してください。

2.1.4　配当割引モデル

NPV 法の考え方を株式投資に適用することもできます。株式の保有期間を T 年とし、時点 t における株価を P_t で表します。株式から生み出される将来キャッシュ・フローは、

- 一株当たり配当と，
- 売却時点での売却損益

であり、t 時点の一株当たり配当を D_t，売却時点での一株当たり売却損益を $P_T - P_{T-1}$ と表します。将来配当や将来売却損益は確率変数であるため期待値で考え、それを割引率 $\tilde{R}$ で割り引くことで、一株当たりの株式価値を求めます。

$$\begin{aligned} P_0^* &= \frac{\mathbb{E}_0[D_1]}{1+\tilde{R}} + \frac{\mathbb{E}_0[D_2]}{(1+\tilde{R})^2} + \cdots + \frac{\mathbb{E}_0[D_\infty]}{(1+\tilde{R})^T} + \frac{\mathbb{E}_0[P_T] - P_0}{(1+\tilde{R})^T} \\ &= \sum_{t=1}^{T} \frac{\mathbb{E}_0[D_t]}{(1+\tilde{R})^t} + \frac{\mathbb{E}_0[P_T] - P_0}{(1+\tilde{R})^T} \end{aligned}$$

十分長い期間 $T \to \infty$ を考えると，売却損益の現在価値はゼロに近づくため，株式の理論価値 P_0^* は以下のように表せます。

$$P_0^* = \sum_{t=1}^{\infty} \frac{\mathbb{E}_0[D_t]}{(1+\tilde{R})^t}$$

これが**配当割引モデル** (Dividend Discount Model: DDM) です。将来配当の予測値データがあれば，現時点における理論上の株価を計算することができます。

2.1.5　ゴードン成長モデル

配当割引モデルは理論上は正しいのですが、将来の配当を無限に予測することは現実的ではありません。そこで、将来の配当が一定の割合で成長していくという仮定を置くことで、将来配当の予測を簡略化する方法があります。より具体的に，期待一株当たり配当が**一定の割合で成長**していくと仮定した割引配当モデルを**ゴードン成長モデル**という。

直近の実現した一株当たり配当を D_0 と置き、将来にわたってこの配当の期待値が一定割合 $G\%$ で成長していくと仮定する。つまり 1 時点先の配当額 D_1 が $(1+G)D_0$ となる、と仮定する。すると、t 時点先の配当額 $\mathbb{E}[D_t]$ は、成長率 G を用いた複利計算により、

$$D_t = (1+G)^t D_0$$

で表すことができます。一定割合で配当額が成長する株式の理論価値 P_0 を配当割引モデルで計算すると，

$$\begin{aligned} P_0 &= \frac{(1+G)D_0}{(1+\tilde{R})} + \frac{(1+G)^2 D_0}{(1+\tilde{R})^3} + \frac{(1+G)^3 D_0}{(1+\tilde{R})^3} + \cdots \\ &= \sum_{t=1}^{\infty} \frac{(1+G)^t D_0}{(1+\tilde{R})^t} \\ &= \frac{(1+G)D_0}{\tilde{R}-G} \end{aligned}$$

2 本目の式から 3 本目の式への計算で、**等比級数の和の公式**を利用している。

Tip

初項 a，公比 r の等比数列

$$a, ar, ar^2, ar^3, \ldots, ar^{n-1}, ar^n, \ldots$$

がある。この等比数列の和を S_n で表す。

$$S_n = a + ar + ar^2 + \cdots + ar^{n-1} + \cdots$$

両辺に r を乗じると，

$$rS_n = ar + ar^2 + \cdots + ar^{n-1} + ar^n + \cdots$$

となる。そして，$S_n - rS_n$ を計算すると，

$$\begin{aligned} S_n - rS_n &= a \\ (1-r)S_n &= a \\ S_n &= \frac{a}{1-r} \end{aligned}$$

上のゴードン成長モデルの初項は $(1+G)D_0/(1+\tilde{R})$，公比は $(1+G)/(1+\tilde{R})$ なので，

$$\begin{aligned} S_n &= \frac{\frac{(1+G)D_0}{(1+\tilde{R})}}{1-\frac{1+G}{1+\tilde{R}}} \\ &= \frac{\frac{(1+G)}{(1+\tilde{R})}D_0}{\frac{(1+\tilde{R})-(1+G)}{1+\tilde{R}}} \\ &= \frac{1+G}{\tilde{R}-G}D_0 \end{aligned}$$

ただし，$\tilde{R} \neq G$ の場合のみである。

2.1.6　割引率と期待リターンの関係

割引率 R は投資家が将来キャッシュフローを購入するにあたって最低限要求する期待リターン (**要求収益率**) とも解釈できます。これくらいリスクのある投資をするのだから，リスクのない資産への投資よりも高いリターンを要求する，という考えを反映しています。

2.2　平均分散アプローチ入門

個々の投資家にとって最適となる証券の組み合わせの比率を決めることを**最適ポートフォリオ選択**といいます。ポートフォリオの価値をリターンの期待値と分散で評価する考え方を**平均・分散アプローチ**といいます。

2.2.1　ポートフォリオのリスクとリターン

1. 投資対象が銘柄 A と銘柄 B の 2 銘柄のみが投資対象であるケース
2. 各銘柄への投資割合を w_A と w_B
3. $w_A + w_B = 1$

記号の定義は以下の通りです。

- 元本 X
- 投資銘柄 A のリターン $1 + R_A$
- 投資銘柄 B のリターン $1 + R_B$

> **Note**
>
> リターンの定義 $(P_t - P_{t-1})/P_{t-1} = P_t/P_{t-1} - 1$ は**ネット・リターン** (net return) と呼ばれるものである。これにたいして，元本も含めたリターン $1 + R_t = P_t/P_{t-1}$ は**グロス・リターン** (gross return) という。

元本 X のうち w_A 分だけ銘柄 A に投資すると、1 年後に期待値で $\mathbb{E}[X \times w_A \times (1+R_A)]$ になります。手元現金全額を銘柄 A と B に振り分けると、

$$\begin{aligned}(1 + R_A)w_A X + (1 + R_B)w_B X &= w_A X + w_A R_A X + w_B X + w_B R_B X \\ &= \left[\underbrace{(w_A + w_B)}_{\text{定義より}=1} + (w_A R_A + w_B R_B)\right] X \\ &= (1 + w_A R_A + w_B R_B)X\end{aligned}$$

となります。この **1 年後の価値と初期投資額の比**としてこのポートフォリオのリターン

$1 + R_P$ を計算します。

$$
\begin{aligned}
1 + R_P &= \frac{\overbrace{(1 + w_A R_A + w_B R_B)X}^{\text{将来時点の評価額}}}{\underbrace{X}_{\text{初期投資}}} \\
&= 1 + w_A R_A + w_B R_B \\
&= 1 + w_A R_A + w_B R_B \\
R_P &= w_A R_A + w_B R_B
\end{aligned}
$$

ポートフォリオ構築時には、各銘柄の実現リターンはわからないので (つまり確率変数)、かわりに期待値や分散を評価します。銘柄 A と銘柄 B のネット・リターンをそれぞれ R_A と R_B で表すと，期待値，分散，標準偏差は次のようになります。

- 期待値 $\mathbb{E}[R_A] = \mu_A$，$\mathbb{E}[R_B] = \mu_B$
- 分散 $\mathbb{V}[R_A] = \sigma_A^2$，$\mathbb{V}[R_B] = \sigma_B^2$
- 共分散 $\mathbb{C}\mathrm{o}\mathbb{V}[R_A, R_B] = \sigma_{AB}$
- 相関係数 $\rho = \frac{\sigma_{AB}}{\sigma_A \sigma_B}$

$$
\begin{aligned}
\rho &= \frac{\sigma_{AB}}{\sigma_A \times \sigma_B} \\
\sigma_{AB} &= \rho \sigma_A \sigma_B
\end{aligned}
$$

と表せます。

上の式より、ポートフォリオのリターンは $R_P = w_A R_A + w_B R_B$ なので、ポートフォリオのリターンの期待値 $\mathbb{E}[R_P] = \mu_P$ も各銘柄の期待リターンの加重平均となります。ここで投資割合 w_A と w_B は定数であり，R_A, R_B は確率変数です。

$$
\begin{aligned}
\mathbb{E}[R_p] = \mu_P &= \mathbb{E}[w_A R_A + w_B R_B] \\
&= w_A \mathbb{E}[R_A] + w_B \mathbb{E}[R_B] \\
&= w_A \mu_A + w_B \mu_B
\end{aligned}
$$

つぎに，ポートフォリオのリターン R_P の分散 $\mathbb{V}[R_P] = \sigma_P^2$ は各銘柄の分散及び相関係数を用いて計算できます。

$$
\begin{aligned}
\mathbb{V}[R_P] = \sigma_P^2 &= \mathbb{V}[w_A R_A + w_B R_B] \\
&= w_A^2 \mathbb{V}[R_A] + w_B^2 \mathbb{V}[R_B] + 2 w_A w_B \mathbb{C}\mathrm{o}\mathbb{V}[R_A, R_B] \\
&= w_A^2 \sigma_A^2 + w_B^2 \sigma_B^2 + 2 w_A w_B \sigma_{AB} \\
&= w_A^2 \sigma_A^2 + w_B^2 \sigma_B^2 + \underbrace{2 w_A w_B \rho \sigma_A \sigma_B}_{\text{ここの}\rho\text{の符号が重要}}
\end{aligned}
$$

確率変数の分散

確率変数 X と Y の線形結合 $aX+bY$ の分散は次のように計算できます。

$$\begin{aligned}\mathbb{V}(aX+bY) &= \mathbb{E}\left[\{(aX+bY)-\mathbb{E}[aX+bY]\}^2\right] \\ &= \mathbb{E}\left[\{(aX+bY)-(a\mu_X+b\mu_Y)\}^2\right] \\ &= \mathbb{E}\left[\{a(X-\mu_X)+b(Y-\mu_Y)\}^2\right] \\ &= \mathbb{E}\left[a^2(X-\mu_X)^2+2ab(X-\mu_X)(Y-\mu_Y)+b^2(Y-\mu_Y)^2\right] \\ &= a^2\mathbb{E}\left[(X-\mu_X)^2\right]+2ab\mathbb{E}\left[(X-\mu_X)(Y-\mu_Y)\right]+b^2\mathbb{E}\left[(Y-\mu_Y)^2\right] \\ &= a^2\mathbb{V}(X)+b^2\mathbb{V}(Y)+2ab\mathrm{Cov}(X,Y)\end{aligned}$$

ポートフォリオ P の分散 $\mathbb{V}(R_P)$ は，各銘柄の分散に投資割合の二乗を乗じたものに，各銘柄のリターンの相関関係部分を加えたものとなっていることが分かります。つまり，この 2 銘柄のリターンの相関係数 ρ に応じて，ポートフォリオ P の分散が大きくなるかどうかが決まる，ということです。

2.2.2　分散投資のメリット

保有比率 (w_A, w_B) を変化させたときのポートフォリオの μ_P と σ_P がどのように変化するかを確認しましょう。分散は非負の値をとることに注意してください。

$$\begin{aligned}\mathbb{V}[R_P] = \sigma_P^2 &= w_A^2\sigma_A^2+w_B^2\sigma_B^2+2\rho w_A w_B\sigma_A\sigma_B \\ &= (w_A\sigma_A+w_B\sigma_B)^2-2w_Aw_B\sigma_A\sigma_B+2\rho w_Aw_B\sigma_A\sigma_B \\ &= \underbrace{(w_A\sigma_A+w_B\sigma_B)^2}_{>0}-\underbrace{2(1-\rho)w_Aw_B\sigma_A\sigma_B}_{\text{ここ重要}}\end{aligned}$$

$0 \le w_A \le 1$ かつ $0 \le w_B \le 1$，$w_A+w_B=1$ のとき、

$$2(1-\rho)w_Aw_B\sigma_A\sigma_B \ge 0$$

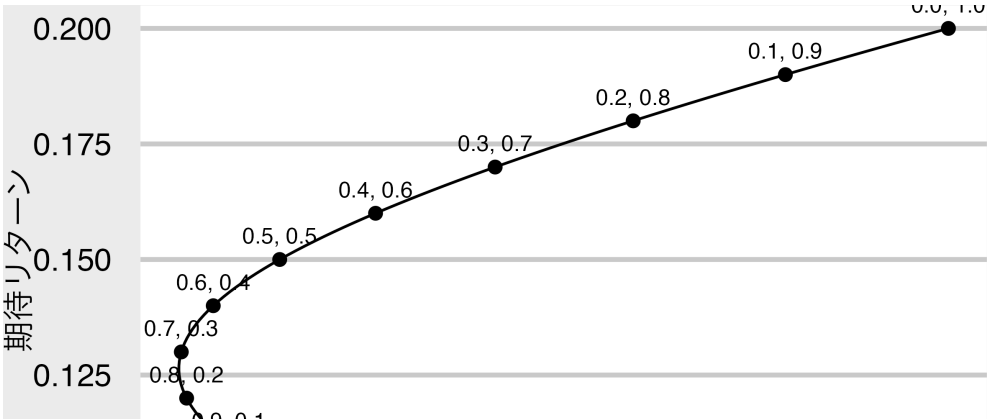
となります。$\rho=1$ のときのみ等号で成立します。したがって、$-1 \le \rho < 1$ のとき、$(1-\rho)w_Aw_B\sigma_A\sigma_B > 0$ となり，次の不等式が成立します。

$$\begin{aligned}&\sigma_P^2 = (w_A\sigma_A+w_B\sigma_B)^2-\underbrace{2(1-\rho)w_Aw_B\sigma_A\sigma_B}_{>0} \\ \iff &\sigma_P^2 \le (w_A\sigma_A+w_B\sigma_B)^2 \\ \iff &\sigma_P \le \underbrace{w_A\sigma_A+w_B\sigma_B}_{\text{リスクの加重平均}}\end{aligned}$$

となり，ポートフォリオのリスクを表す標準偏差 σ_P が銘柄 A と B の標準偏差の加重平均 $w_A\sigma_A+w_B\sigma_B$ 以下になることがわかります。これを**分散投資効果**という。

Listing 2.3 分散投資効果 p.65

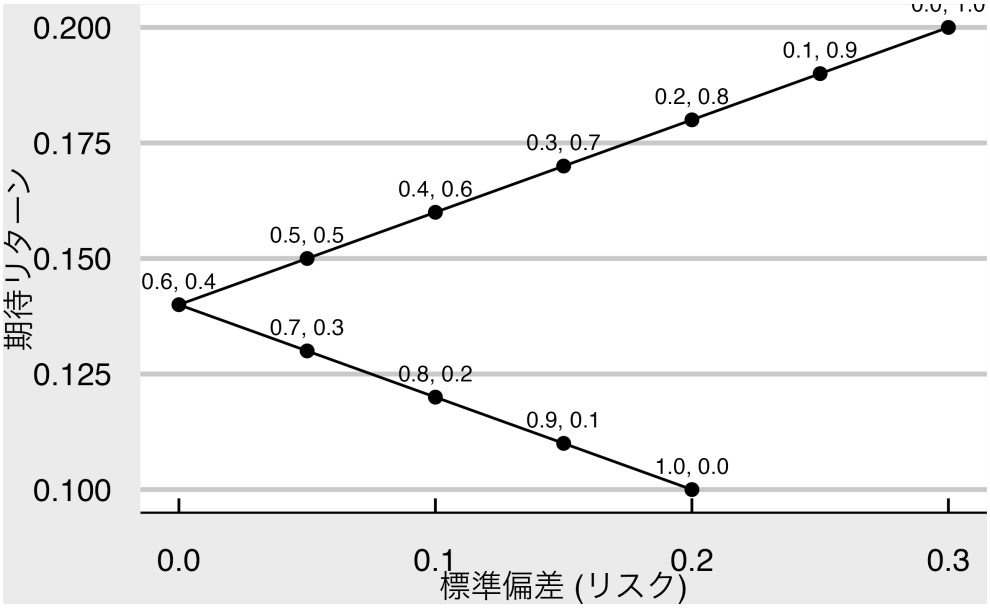
```
# パラメータ設定
params <- list(
  mu_A = 0.1, sigma_A = 0.2,
  mu_B = 0.2, sigma_B = 0.3,
  rho = 0.2
)

# データの生成
df_portfolio <- tibble(
  wa = seq(0, 1, by = 0.01),
  wb = 1 - wa
  ) |>
  mutate(
    mu_p = wa * params$mu_A + wb * params$mu_B,
    var_p = wa^2 * params$sigma_A^2 + wb^2 * params$sigma_B^2 +
      2 * params$rho * wa * wb * params$sigma_A * params$sigma_B,
    sigma_p = sqrt(var_p),
    # ラベル作成: 0.1 刻みの点のみラベルを付ける
    label = if_else(round(wa * 100) %% 10 == 0,
                    sprintf("%.1f, %.1f", wa, wb),
                    NA_character_)
  )

# 作図
ggplot(df_portfolio, aes(x = mu_p, y = sigma_p)) +
  geom_line() +
  geom_point(data = df_portfolio |> filter(!is.na(label)), size = 2) +
  geom_text(aes(label = label), na.rm = TRUE, vjust = -1, size = 3) +
  coord_flip() +
  labs(x = "期待リターン", y = "標準偏差（リスク）") +
  mystyle
```



完全な負の相関 ($\rho = -1$) であるの場合、$\sigma_P^2 = 0$ のポートフォリオを構築できる。確認のため、2 銘柄の価格が完全な負の相関 $\rho = -1$ をもつとき、ポートフォリオ P のリスク σ_P は

$$\begin{aligned}
\mathbb{V}[R_P] = \sigma_P^2 &= (w_A\sigma_A + w_B\sigma_B)^2 - 2(1-(-1))w_Aw_B\sigma_A\sigma_B \\
&= (w_A\sigma_A + w_B\sigma_B)^2 - 4w_Aw_B\sigma_A\sigma_B \\
&= w_A^2\sigma_A^2 + w_B^2\sigma_B^2 + 2w_Aw_B\sigma_A\sigma_B - 4w_Aw_B\sigma_A\sigma_B \\
&= w_A^2\sigma_A^2 - 2w_Aw_B\sigma_A\sigma_B + w_B^2\sigma_B^2 \\
&= (w_A\sigma_A - w_B\sigma_B)^2 \\
\sigma_P &= |w_A\sigma_A - w_B\sigma_B|
\end{aligned}$$

以下のように，$w_A\sigma_A = w_B\sigma_B$ となるように w_A と w_B を選べば、$\sigma_P = 0$ となるポートフォリオを作れる。



このケースでは、$\sigma_A = 0.2$、$\sigma_B = 0.3$ となっているため、$0.2w_A = 0.3w_B$ となる保有割合を考える。

$$\begin{aligned}
0.2w_A &= 0.3w_B \\
0.2w_A &= 0.3(1-w_A) \\
0.2w_A &= 0.3 - 0.3w_A \\
0.5w_A &= 0.3 \\
w_A &= 0.6
\end{aligned}$$

となるため、銘柄 A に 60 %、銘柄 B に 40% を投資することで、リスクゼロで期待リターン $0.6 \times 0.1 + 0.4 \times 0.2 = 0.14$ を獲得することができる。

2.2.3　空売りの効果

空売り (short sale) とは、

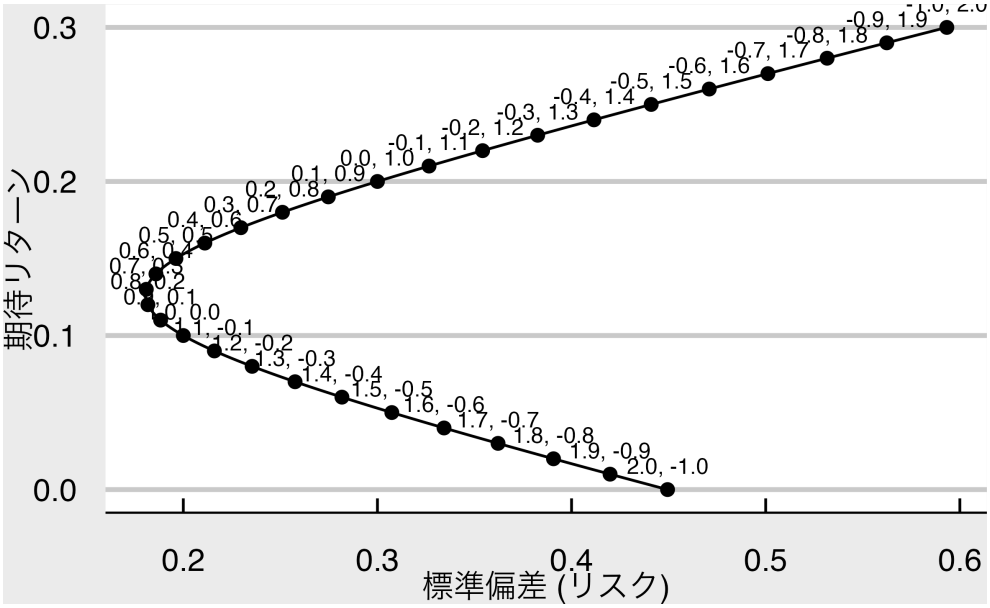
Listing 2.4 完全な負の相関の分散投資効果 p.65

```
# パラメータ設定
params <- list(
  mu_A = 0.1, sigma_A = 0.2,
  mu_B = 0.2, sigma_B = 0.3
)

# データ生成
df_neg_corr <- tibble(
  wa = seq(0, 1, by = 0.01),
  wb = 1 - wa
) |>
  mutate(
    mu_p = wa * params$mu_A + wb * params$mu_B,
    # 完全な負の相関の場合、標準偏差は加重平均の差の絶対値になる
    sigma_p = abs(wa * params$sigma_A - wb * params$sigma_B),
    # ラベル作成: 0.1 刻みの点のみラベルを付ける
    label = if_else(round(wa * 100) %% 10 == 0,
                    sprintf("%.1f, %.1f", wa, wb),
                    NA_character_)
  )

# 作図
ggplot(df_neg_corr, aes(x = mu_p, y = sigma_p)) +
  geom_line() +
  geom_point(data = df_neg_corr |> filter(!is.na(label)), size = 2) +
  geom_text(aes(label = label), na.rm = TRUE, vjust = -1, size = 3) +
  coord_flip() + # 縦軸をリターン、横軸をリスクにするため入れ替え
  labs(x = "期待リターン", y = "標準偏差（リスク）") +
  mystyle
```

1. 保有していない証券を誰か (普通は証券会社) から借りてきて売却し、
2. 一定期間後に買い戻して元の持ち主に返却する

取引をいい、**値下がりから利益を得る**。空売りを行う投資家をショートセラー (short seller) という。

いままでは、$0 \leq w_A, w_B \leq 1$ という制約を置いていたが、この制約をはずして $w_A + w_B = 1$ のみを課す。つまり $w_A < 0$ や $w_B < 0$ が空売りを表す。



先ほどの空売りなしのケースと比べると，曲線が大きく広がっていることがわかります。次に，安全資産を加えた 3 資産の場合を考えます。

2.2.4　安全資産の導入

安全資産の購入についても考える。

安全資産 F のリターンを R_F、ポートフォリオ P の保有比率を w_F，銘柄 A と銘柄 B と安全資産の投資割合をそれぞれ w_A, w_B, w_F で表します。

まず銘柄 A と安全資産の 2 資産からなるポートフォリオを考えます。各資産への投資割合を $w_A + w_F = 1, \quad w_B = 0$ とする場合を考える。この安全資産と銘柄 A からなるポートフォリオ P の (ネット) リターン $\tilde{R}_P$ は，

$$\tilde{R}_P = w_F R_F + w_A \tilde{R}_A$$

となり，このポートフォリオの期待値は，

$$\begin{aligned}
\mu_P = \mathbb{E}[R_P] &= \mathbb{E}[w_F R_F + w_A \tilde{R_A}] \\
&= w_F R_F + w_A \mathbb{E}[\tilde{R_A}] \\
&= w_F R_F + w_A \mu_A \\
&= (1 - w_A) R_F + w_A \mu_A \\
&= R_F - w_A R_F + w_A \mu_A \\
&= R_F + w_A (\underbrace{\mu_A - R_F}_{\text{リスクプレミアム}})
\end{aligned}$$

となります。もちろん安全資産のリターンは定数なので，期待値をとってもそのままです。$\mu_A - R_F$ はリスク資産である銘柄 A の**リスクプレミアム**を表しています。通常，リスクのある資産の期待リターンは安全資産のリターンより大きいため，リスクプレミアム

は正の値となります。したがって，リスク資産への投資割合 w_A を 1 単位増加させれば，$\mathbb{E}[R_P]$ はリスクプレミアム分増加する

つぎに，ポートフォリオのリスクを表す標準偏差 σ_P とリターンの関係は次式で表せます。まず安全資産のリスクはゼロであるため，リスク資産の銘柄Aを保有する分だけリスクが生じる。

$$\begin{aligned}\sigma_P^2 = \mathbb{V}[R_P] &= \mathbb{V}[w_F R_F + w_A R_A] \\ &= \mathbb{V}[w_A R_A] \\ &= w_A^2 \mathbb{V}[R_A] \\ &= w_A^2 \sigma_A^2 \\ \sigma_P &= |w_A| \sigma_A\end{aligned}$$

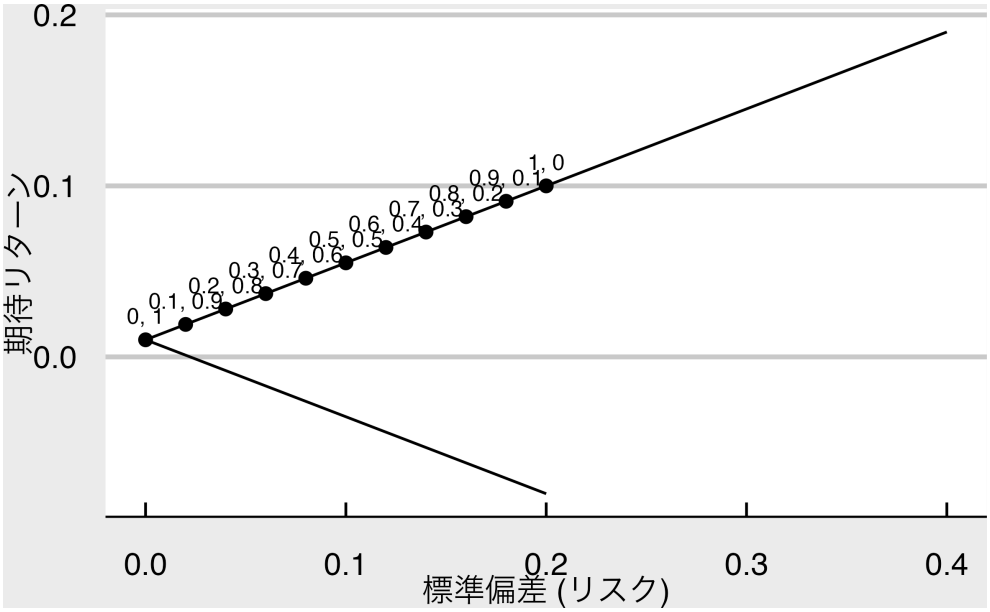
空売りを想定する場合 $w_A < 0$ となるため，標準偏差を求める際に絶対値をとっています。空売りはない状況（つまり，$w_A > 0$）を想定すると，

$$\begin{aligned}\sigma_P &= w_A \sigma_A \\ w_A &= \frac{\sigma_P}{\sigma_A}\end{aligned}$$

のように，リスク資産である銘柄 A への投資割合 w_A が，ポートフォリオ P とリスク資産 A のリスクの割合で決定されることがわかります。これを，ポートフォリオの期待リターン μ_P に代入すると、

$$\begin{aligned}\mu_P &= R_F + w_A(\mu_A - R_F) \\ &= R_F + \frac{\sigma_P}{\sigma_A}(\mu_A - R_F) \\ &= R_F + \frac{\mu_A - R_F}{\sigma_A}\sigma_P\end{aligned}$$

となり，期待リターン μ_P は，切片が R_F，傾きが $(\mu_A - R_F)/\sigma_A$ とする σ_P の線形関数となる。



2.2.5　3 資産のポートフォリオ

リスク資産 A と B，安全資産 F の 3 資産に投資するポートフォリオを考える。ここで，$w_A + w_B > 0$ を仮定し，少なくとも少しはリスク資産を保有するケースを考える。当然だけれど，$w_A = w_B = 0$ のケースでは，安全資産のみを保有するケースとなり，リスクも無く，リターンも確定している。

3 資産 A,B,F への投資割合をそれぞれ w_A，w_B,w_F とすると，3 資産からなるポートフォリオの期待リターンは次のように計算できる。

$$\begin{aligned}\mathbb{E}[R_P] &= w_F R_F + w_A \mathbb{E}[\tilde{R}_A] + w_B \mathbb{E}[\tilde{R}_B] \\ &= w_F R_F + (w_A + w_B)\left(\frac{w_A}{w_A + w_B}\mathbb{E}[\tilde{R}_A] + \frac{w_B}{w_A + w_B}\mathbb{E}[\tilde{R}_B]\right)\end{aligned}$$

安全資産への投資割合 w_F とリスク資産への投資割合 $w_C = w_A + w_B$ とまとめて，式を変形させる。安全資産への投資以外の資金で構築したリスク資産 A と B からなるポートフォリオを P_C を考えると，P_C の期待リターン R_C は次のように計算できる。

$$R_C = \frac{w_A}{w_A + w_B}\mathbb{E}[\tilde{R}_A] + \frac{w_B}{w_A + w_B}\mathbb{E}[\tilde{R}_B]$$

安全資産 F とポートフォリオ C を保有した場合の期待リターンは次式となる。

$$\begin{aligned}\mu_P &= \mathbb{E}[w_F R_F + w_A R_A + w_B R_B] \\ &= w_F R_F + w_A \mathbb{E}[R_A] + w_B \mathbb{E}[R_B] \\ &= w_F R_F + (w_A + w_B) \underbrace{\left(\frac{w_A}{w_A + w_B}\mathbb{E}[R_A] + \frac{w_B}{w_A + w_B}\mathbb{E}[R_B]\right)}_{=w_C \text{とおく}} \\ &= w_F R_F + w_C \mu_C\end{aligned}$$

安全資産と 2 つのリスク資産からなるポートフォリオの期待リターンは，安全資産の期待リターンとリスク資産の期待リターンの和となる。

安全資産と 2 つのリスク資産に投資可能な場合，(μ_P, σ_P) の取りうる値を図示できる。テキストの数値例を用いて R で図示してみる。

- リスク資産 A の期待リターン 0.1，標準偏差 0.2
- リスク資産 B の期待リターン 0.2，標準偏差 0.3
- 安全資産の期待リターンを 0.01
- リスク資産 A と B の間の相関係数は 0.2
- 安全資産，銘柄 A，銘柄 B への投資割合を 0.2，0.3，0.5 とするポートフォリオを考える。このポートフォリオの期待リターン μ_P と標準偏差 σ_P は次のようになる。

2.3 最適ポートフォリオ問題

「どのようなポートフォリオが投資家にとって望ましいか」

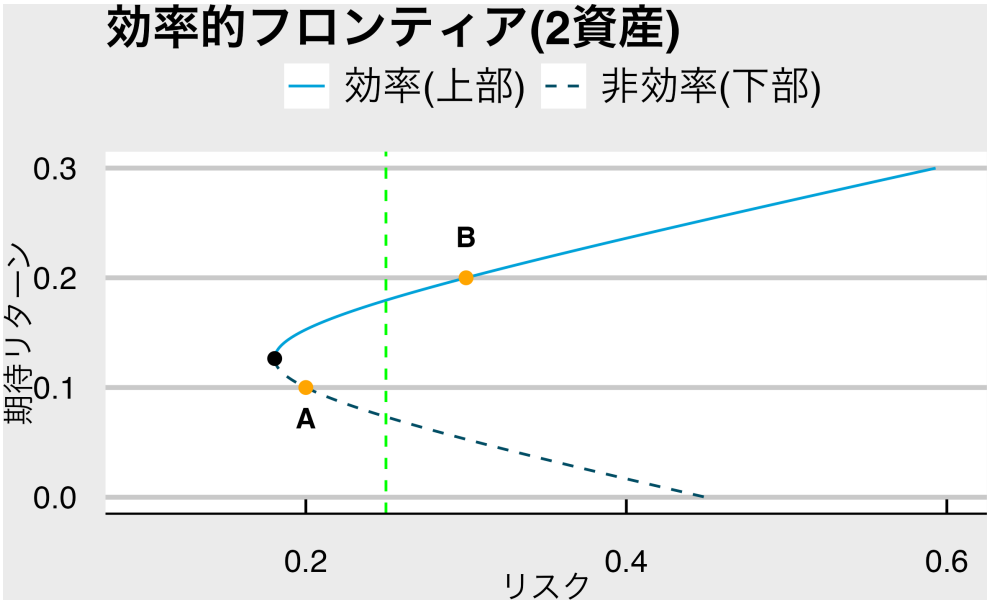
一般に、投資家はリスクが小さい一方でリターンが大きいポートフォリオを好む。ここでは，リターンをポートフォリオの期待リターン μ_P，リスクをポートフォリオの標準偏差 σ_P で表し，この 2 つの変数から構成される平面 (μ_P, σ_P) 上で最適ポートフォリオ問題を分析する。

2.3.1 効率的フロンティア

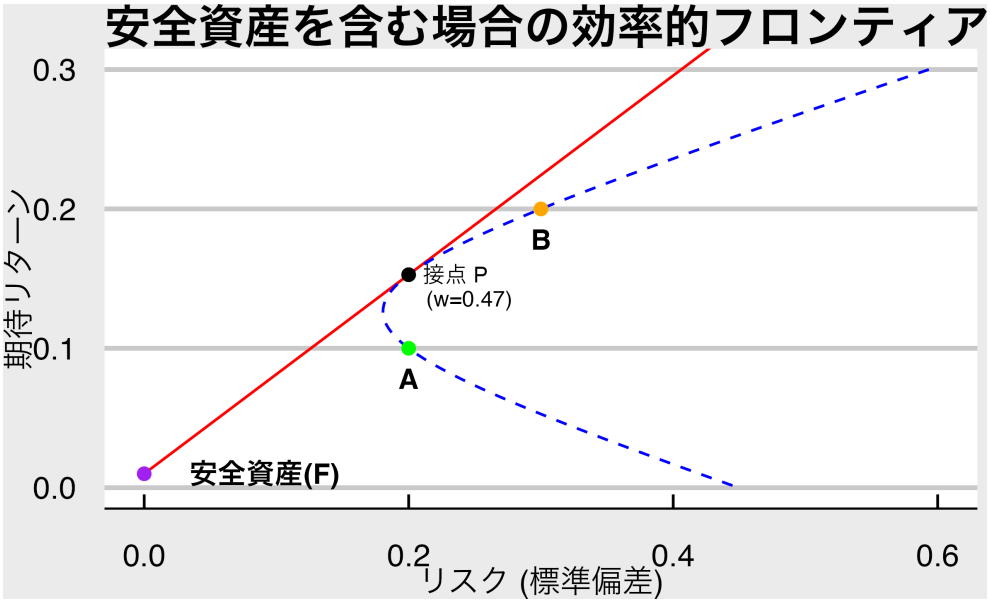
リスク資産 A と資産 B にのみ投資可能であり，それぞれに異なる割合で投資したポートフォリオ D と Eを比較する (以下の図)

- 標準偏差はともに 0.25 $\sigma_D = \sigma_E = 0.25$
- D の方が E よりも期待リターンが大きい，$\mu_D > \mu_E$

つまり，同じリスク（標準偏差）ならリターン（期待リターン）が高いポートフォリオに投資したほうがいいです。以下のグラフでいうと、同じリスク (緑のライン上) なら、リターンの高いポートフォリオが望ましい。そのため青い線が効率的フロンティアとなり、点線は選択されないポートフォリオになります。



リスク資産 A と B に加え、安全資産 F にも投資可能な場合、効率的フロンティアは**直線**になります。この効率的フロンティアは**資本市場線** (Capital Market Line; CML) とも呼ばれます。資本市場線の傾きは、この金融市場における**リスクとリターンのトレードオフ**を表しているといえます。



安全資産が利用可能な場合、投資家は「安全資産」と「任意のリスク資産ポートフォリオ」を組み合わせることができます。先ほどの式で確認した通り、安全資産とあるポートフォリオを組み合わせると、リスク・リターン平面上では直線が描かれます。では、投資家はどのリスク資産ポートフォリオと安全資産を組み合わせるべきでしょうか？

図の紫の点（安全資産 R_F）からスタートします。そこから、青い点線（リスク資産のみ

のフロンティア）上の任意の点に向けて直線を引くことができます。投資家は、同じリスクならより高いリターンを好むため、傾きが最も急になる直線を選ぼうとします。その直線とは、安全資産の点から伸び、リスク資産のフロンティアに接する直線（赤い実線）となります。この接点となるポートフォリオを**接点ポートフォリオ** (Tangency Portfolio) と呼びます。この赤い直線上の組み合わせは、青い曲線上のどの点よりも（接点を除いて）左上に位置するため、より効率的です。したがって、安全資産が存在する場合の効率的フロンティアはこの**直線（資本市場線）**となります。

【直感的理解】なぜ曲線ではなく直線が最強なのか？

この理屈は、**「ピンと定規」**を使うと直感的に理解できます。

1. **リスク資産のエリア（青い曲線）**これまで見てきた双曲線（青い点線）は、株式などのリスク資産だけで作れる限界ラインでした。投資家はこのライン上のどこかを選ぶしかありませんでした。
2. **安全資産（紫の点）**ここに「リスクゼロでお金が増える」安全資産（R_F）が登場します。これはグラフの左端（リスク 0）の軸上にピンを刺すようなものです。
3. **混ぜると「直線」になる**安全資産と、あるリスク資産を混ぜてポートフォリオを作ると、グラフ上ではその 2 点を結ぶ直線が描かれます。投資家は、安全資産のピン（R_F）を支点にして、定規をリスク資産のエリア（青い曲線）に向けて伸ばすことができます。
4. **定規を跳ね上げる（最適化）**投資家は「同じリスクなら、できるだけ高いリターン」を欲しがります。つまり、R_F に刺したピンを支点にして、定規の角度をできるだけ急にして、グイッと上に持ち上げたいと考えます。
   - 定規が曲線の内側を通るとき（割線）：もっと上にいけます。
   - 定規が曲線から離れてしまったとき：そのような商品は存在しないので買えません。

定規を限界まで上に持ち上げたとき、定規は青い曲線の一番出っ張った部分に「ピタリ」と接するはずです。この**「接した状態の直線」**こそが、物理的に可能な範囲で最もリターン効率が良いライン（効率的フロンティア）となります。そして、その接点にあるのが「接点ポートフォリオ」です。これ以外の青い曲線上の点は、すべてこの直線の下側（＝効率が悪い）に来てしまうため、選ばれなくなります。

2.3.2 投資家のリスク回避度と最適ポートフォリオ

効率的フロンティアのうち、どの点が投資家の最適ポートフォリオになるのか，について考える。そのためには投資家のリスク・リターンのトレードオフに関する**選好 (preference)** の特徴，つまり**リスクの回避度**の情報が必要となる。(μ_P, ρ_p) 平面上でそれを描く方法の一つが**無差別曲線** (indifference curve) である。**無差別曲線**とは，投資家

の効用 (utility) が**一定**となるリスクとリターンの組み合わせを描いた曲線をいう。つまり同じ効用水準を達成できるリスクとリターンの組み合わせを表現した曲線である。

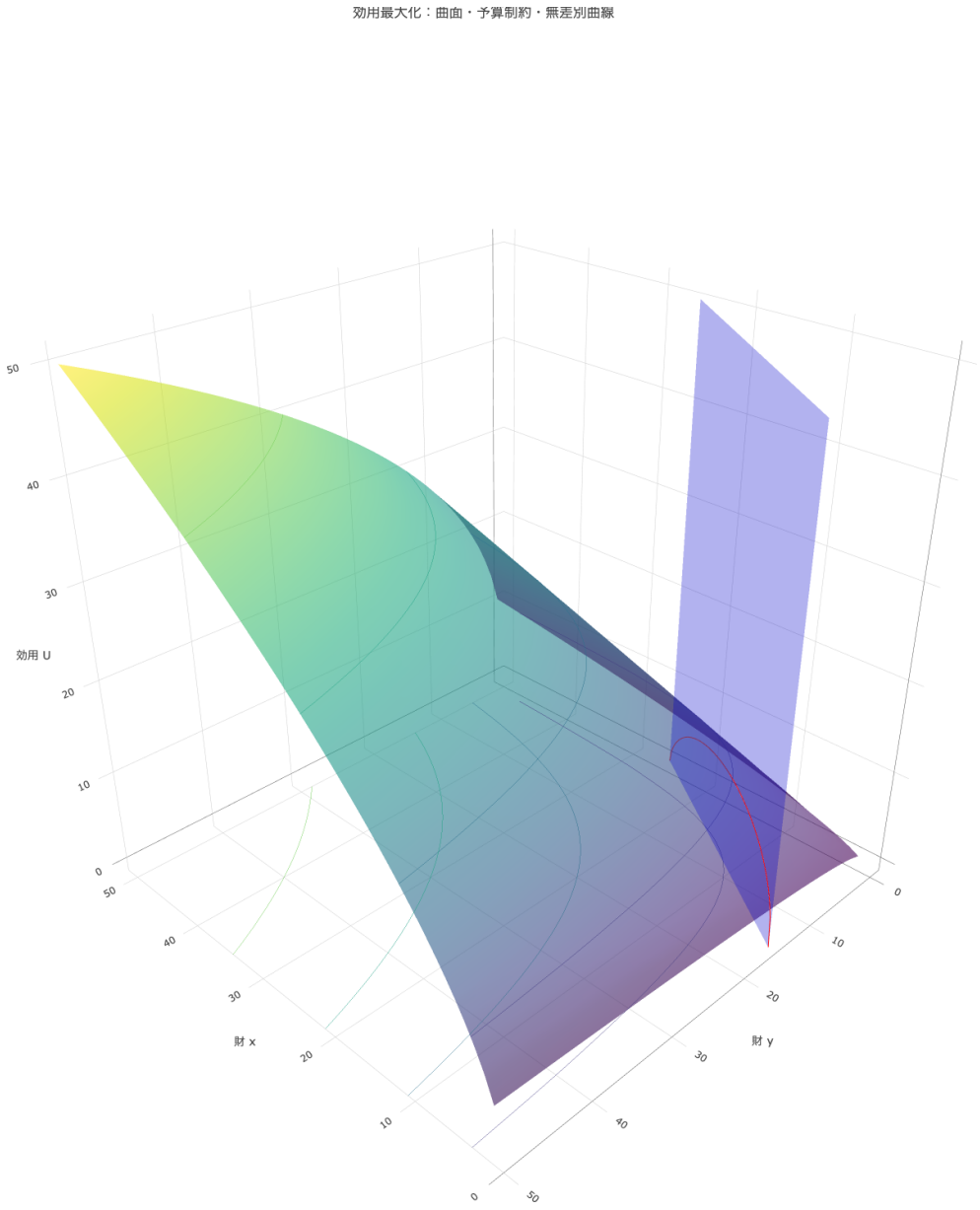
2.3.2.1　効用関数の例

以下では，財 x と y を消費したときの効用 U を図示しています。この消費者の効用関数は $U(x,y) = x^{\frac{2}{5}} \times y^{\frac{3}{5}}$ としている。

なぜこのような設定にしているのかというと，この効用関数は**限界代替率逓減** (diminishing marginal rate of substitution) を満たしており，たくさん消費すると効用が増えるが，その増え方がだんだん小さくなる，という現実をよく表している性質を持っているからです。

```
# 1. 効用関数のデータ作成
x <- 1:50
y <- 1:50
u_func <- function(x, y) { x^(2/5) * y^(3/5) }
U <- outer(x, y, u_func)

# 2. 予算制約面のデータ作成
bc_x_vec <- seq(0, 25, length.out = 50)
bc_z_vec <- seq(0, 50, length.out = 50)
bc_x_mat <- matrix(rep(bc_x_vec, times = 50), nrow = 50, ncol = 50)
bc_z_mat <- matrix(rep(bc_z_vec, each = 50), nrow = 50, ncol = 50)
bc_y_mat <- (100 - 4 * bc_x_mat) / 6

# 3. 交線のデータ作成
line_x <- seq(0, 25, length.out = 100)
line_y <- (100 - 4 * line_x) / 6
line_z <- u_func(line_x, line_y)

# 4. 描画
plot_ly() |>
  # (A) 効用関数の曲面 + 等高線
  add_surface(
    x = ~x, y = ~y, z = ~U,
    opacity = 0.6,
    colorscale = "Viridis",
    name = "効用関数",
    showscale = FALSE, #【追加】カラーバー（色の凡例）を消す
    contours = list(
      z = list(
        show = TRUE,
        usecolormap = TRUE,
```

```
      highlightcolor = "#ff0000",
      project = list(z = TRUE)
    )
  )
) |>
# (B) 予算制約面
add_surface(
  x = bc_x_mat,
  y = bc_y_mat,
  z = bc_z_mat,
  opacity = 0.3,
  colorscale = list(c(0, 1), c("blue", "blue")),
  showscale = FALSE, # 元からある設定（ここも消す）
  name = "予算制約"
) |>
# (C) 交線
add_paths(
  x = line_x, y = line_y, z = line_z,
  line = list(width = 2, color = "red"),
  name = "予算線上の効用"
) |>
layout(
  title = "効用最大化：曲面・予算制約・無差別曲線",
  showlegend = FALSE, #【追加】項目名（線や面の名前）の凡例を消す
  scene = list(
    xaxis = list(title = "財 x"),
    yaxis = list(title = "財 y"),
    zaxis = list(title = "効用 U"),
    camera = list(eye = list(x = -1.5, y = 1.5, z = 1.2))
  )
)
```



この山のような部分が効用関数の曲面を表しており，x と y をたくさん消費すれば効用水準が高くなっています。青い面は予算制約面を表しており，この面の上は予算を使い切って消費することを意味しています。つまり青い面上の点で，もっとも満足度が高い場所が，最適消費点となります。

分かりやすくするために，この三次元の図を上から見た図を考えます。青いラインは予算制約で，青い線の内側が**消費可能集合** (affordable set) となります。右上に行くほど効用が高くなるので，予算制約と無差別曲線の接点が最適消費点となります。

```
contour(x, y, U, method = "edge", labcex = 1, lwd = 2)
abline(a = 100/6, b = -4/6, lwd = 2, col = "blue") #予算制約線
```

```
points(x = 10, y = 10, lwd = 3, col = "darkblue", pch = 16) #最適消費点
```

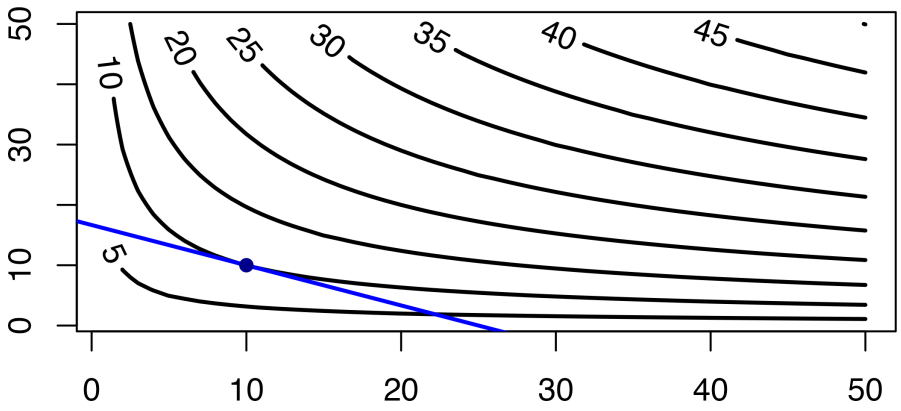


Figure2.1: 無差別曲線と消費可能集合

無リスク利子率が 10% で，リスクプレミアムが 5 %，β が 1.2 の場合の期待リターンは，$R_F + \beta \times (R_M - R_F) = 0.1 + 1.2 \times 0.05 = 0.16$ となります。このときの無差別曲線は $U = 0.16$ となるようなリスクとリターンの組み合わせを表します。

無差別曲線の**局所的な傾き**は、その投資家が追加的なリスクを引き受けるうえで要求するリスクプレミアム，つまり傾き＝我慢料の相場を表しています。したがって，リスク回避的な投資家ほど，リスクを 1 単位負担する際に，より大きなリスクプレミアムを要求するので、傾きは大きくなる，ということです。

無差別曲線と効率的フロンティアが接する点が、この投資家にとっての最適ポートフォリオとなります。投資可能なポートフォリオの範囲で、最も左上の無差別曲線を実現するのが接点となる。どの点が最適ポートフォリオとして選ばれるかは個々の投資家の無差別曲線の形状 (リスク回避度) に依存する。最適ポートフォリオにおいて、無差別曲線と効率的フロンティアの局所的な傾きは一致 (接線だから当然) するため、その投資家が要求するリスクプレミアムがちょうど実現されている。

上図の場合，この投資家の最適ポートフォリオは $(w_F, w_{tan}) \approx (0.29, 0.71)$ の比率で構成される。接点ポートフォリオは銘柄 A に 47 %、銘柄 B に 53 %投資するポートフォリオだったので、最適保有比率は、$(w_F, w_A, w_B) \approx (0.29, 0.33, 0.38)$ と書き換えられる。

2.3.3　トービンの分離定理

安全資産が投資可能な場合の最適ポートフォリオ問題を考える。

1. 接点ポートフォリオを求め、リスク資産同士の相対的な保有比率を求める。
2. 投資家ごとのリスク回避度に応じて安全資産と接点ポートフォリオの最適保有比率の決定

1 は各投資家で共通している。いったん接点ポートフォリオを求めてしまえば、他の投資家はその情報を用いて 2 を考えればよい。

最適ポートフォリオ問題を 2 段階に分離できるという命題は、**トービンの分離定理** (又は二基金分離定理) と呼ばれている。

2.4　CAPM

金融市場全体の均衡を考えるために、**資本資産価格モデル** (Capital Asset Pricing Model; CAPM) を学習します。

2.4.1　仮定の確認

- **選好**：ポートフォリオを**期待値と標準偏差の基準で評価**
- **取引コスト**：取引コストは存在せず、空売りも自由
- **流動性**：どれだけ売買しても証券の価格は不変
- **情報集合**：みんな同じ情報を共有

上記の仮定を満たす市場のことを、**完全資本市場** (完全市場: perfect market) と呼びます。空想上の市場ですが、理論構築の大事な出発点です。

2.4.2　CAPM の第一命題

先の仮定の下で、安全資産が投資可能なとき、全ての投資家の最適ポートフォリオ問題に対してトービンの分離定理を応用することができます。全ての投資家は「安全資産」と「接点ポートフォリオ」のみに投資し、危険資産に限定すれば全員が同じ構成比率のポートフォリオ（接点ポートフォリオ）を保有します。金融市場全体の均衡を議論するうえで、市場にその資産が供給されている以上、誰かがその最適ポートフォリオの一部として保有しているはずです。すなわち、需要と供給が一致している点がポイントです。

以上の議論をよりフォーマルに述べるために、市場ポートフォリオを導入します。

> **!** 市場ポートフォリオ
>
> **市場ポートフォリオ (market portfolio)** とは、市場に存在する全ての危険資産を時価総額比率で保有したポートフォリオをいう。厳密には、リスク資産には株式や債券に代表される金融資産の他、不動産や貴金属などの実物資産も含まれるが、実用上は TOPIX や S&P500 といった株価指数と同一視されることが多い。

> **!** CAPM の第一命題
>
> 市場ポートフォリオは接点ポートフォリオと一致し、効率的フロンティア（資本市場線）上に位置する。

投資家は市場ポートフォリオに投資するとき、σ_M のリスクを背負う見返りとして、R_F に加えて $\mu_M - R_F$ だけ追加的な報酬を期待します。この追加的な報酬を**市場リスクプレ**

ミアム (market risk premium) といいます。したがってこの命題の下では、資本市場線を市場リスクプレミアム ($\mu_M - R_F$) を利用して、以下のように表せます。

$$\mu_P = R_F + \frac{\mu_M - R_F}{\sigma_M}\sigma_P$$

前の章で用いたパラメータをそのまま考えます。接点ポートフォリオの保有比率は概ね 47% を銘柄 A に、53% を銘柄 B に投資する結果となりました。CAPM の第一命題によると、均衡した市場における銘柄 A と B の時価総額比率は約 0.47 対 0.53 になっていなければなりません。この命題に従えば、投資家は各銘柄の期待リターンや分散から接点ポートフォリオを計算する必要はなく、単に時価総額加重で市場ポートフォリオを保有すればよいことになります。

- **パッシブ運用：**幅広い銘柄に分散投資し、市場平均と同じようなパフォーマンスを目指す運用手法。
- **アクティブ運用：**市場平均を上回るパフォーマンスを目指し、投資銘柄を絞ったり、投資比率を工夫したりする運用方法。

任意のポートフォリオの収益性を測る指標として、**シャープ・レシオ**が提唱されています。シャープ・レシオは追加的なリスク・テイクによってどれだけリスクプレミアムを改善できるのかを表す指標です。CAPM の第一命題によると、市場ポートフォリオはシャープ・レシオを最大化するという意味で最も効率的なポートフォリオであり、資本市場線の傾き $\frac{\mu_M - R_F}{\sigma_M}$ は市場ポートフォリオのシャープ・レシオと一致します。

$$\text{Sharpe Ratio} = \frac{\mu_P - R_F}{\sigma_P}$$

2.4.3 CAPM の第二命題

第二命題は個々の資産のリスクとリターンのトレードオフを数式で表現したものです。ある証券に投資するときのリスクと、その証券に投資するときの期待リターンとの関係を知ることができるようになります。

各投資家が証券 i を追加的に保有する際、重要となるのは市場ポートフォリオとの相関です。分散が大きい資産であっても、市場ポートフォリオと負に相関していれば、その資産を追加的に保有することでポートフォリオ全体のリスクは低減されます。CAPM の第二命題は、この相関を以下の**マーケット・ベータ**として定量化します。(※ R_i は証券 i のリターン、R_M は市場ポートフォリオのリターン)

この β_i は市場ポートフォリオのリスクを 1 としてベンチマーク化し、その証券のリスクがベンチマークの 1 を上回るか下回るかを測るものです。β_i が大きいほど証券 i は投資家にとってリスク (システマティック・リスク) が大きいことを意味します。CAPM の世界では、証券 i のリスクはその証券のリターンの標準偏差ではなく、この β_i によって測られます。

$$\beta_i = \frac{\mathbb{C}\mathrm{ov}[R_i, R_M]}{\mathbb{V}\mathrm{ar}[R_M]}$$

金融市場全体が均衡しているには、リスクの高い証券はその分だけ期待リターンも高くなければなりません。もし β_i が低いにもかかわらず期待リターンが高い証券があるなら、投資家は市場ポートフォリオから離れてその証券をさらに買い増しするインセンティブを持ちます。その結果、市場価格が上がり、期待リターンが下がるため、最終的に β_i に応じた期待リターンが均衡で実現されます。

これまでは市場リスクプレミアムを $\mu_M - R_F$ と表記していましたが、以後ではより一般的な $\mathbb{E}[R_M] - R_F$ と表記します。

! CAPM の第二命題

各証券のリスクプレミアムは、その証券のマーケット・ベータに比例する。この式は、証券 i のリスクプレミアム $\mathbb{E}[R_i] - R_F$ を、β_i と市場リスクプレミアム $\mathbb{E}[R_M] - R_F$ に分解している。

$$\mathbb{E}[R_i] - R_F = \beta_i(\mathbb{E}[R_M] - R_F)$$
$$\text{ただし、} \beta_i = \frac{\mathbb{C}\mathrm{ov}[R_i, R_M]}{\mathbb{V}\mathrm{ar}[R_M]}$$

通常、市場リスクプレミアムは正の値をとるので、CAPM の第二命題によると、個々の証券のリスクプレミアムは β_i に関して線形に増加します。β_i はあくまで市場ポートフォリオとの相関でリスクを定量化しているのがポイントです。いくら個々の証券の分散（リスク）が大きくても、それが市場ポートフォリオと相関しない固有リスクであれば、リスクプレミアムには反映されません。期待値をとる前の R_i を分解して確認します。

$$R_i = R_F + \beta_i(R_M - R_F) + \varepsilon_i$$

ここで ε_i は期待値ゼロで R_M と相関しない誤差項です。

$$\mathbb{E}[\varepsilon_i] = 0, \qquad \mathbb{C}\mathrm{ov}[\varepsilon_i, R_M] = 0$$

分散を計算すると以下のようになります。

$$\begin{aligned}\mathbb{V}\mathrm{ar}[R_i] &= \mathbb{V}\mathrm{ar}[\beta_i R_M + \varepsilon_i] \\ &= \beta_i^2 \mathbb{V}\mathrm{ar}[R_M] + \mathbb{V}\mathrm{ar}[\varepsilon_i] + \underbrace{2\mathbb{C}\mathrm{ov}[\beta_i R_M, \varepsilon_i]}_{=0} \\ &= \underbrace{\beta_i^2 \mathbb{V}\mathrm{ar}[R_M]}_{\text{市場ポートフォリオとの相関による寄与分}} + \underbrace{\mathbb{V}\mathrm{ar}[\varepsilon_i]}_{\text{誤差項による寄与分}}\end{aligned}$$

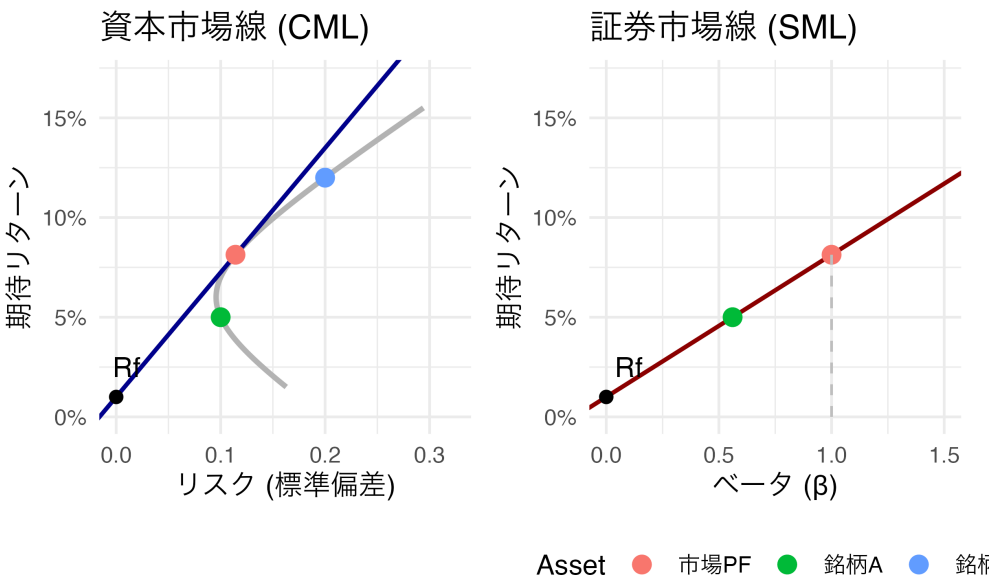
R_i の分散は、市場ポートフォリオとの相関による寄与分（システマティック・リスク）と誤差項による寄与分（固有リスク）に分解できます。誤差項の分散が大きければその分だけ R_i の分散も大きくなりますが、証券 i のリスクプレミアムは $\beta_i(\mathbb{E}[R_M] - R_F)$ のま

まで変化はありません。つまり、分散投資によって消去可能な固有リスクに対しては、市場は報酬（プレミアム）を与えないのです。

2.4.4　証券市場線

下左の図が示すように、安全資産と複数の危険資産が投資可能な場合、投資家の最適ポートフォリオは資本市場線上の 1 点 (オレンジの点) となります。一方、CAPM の第二命題が示唆するように、各証券のリスク（ベータ）とリターンとの関係は、右上がりの直線となります（図 2.14 右図)。縦軸に各証券の期待リターン、横軸に各証券のリスクを表すマーケット・ベータをとると、CAPM が完全に成立する世界では全ての資産が一直線上に並びます。この直線を**証券市場線** (Securities Market Line; SML) と呼びます。

定義通り β を計算すると銘柄 A は約 0.63、銘柄 B は約 1.33 となります。両者の期待リターンおよび β を図示すると証券市場線に乗っており、この仮想的な市場では CAPM が成立していることがわかります。



2.5　N 資産が投資可能な場合への拡張

ここまではリスク資産が銘柄 A と B の二つしかない場合を分析してきましたが，平均分散アプローチや CAPM は危険資産の数が任意の N 個であっても成立します。

Listing 2.5 空売りの効果 p.66

```
# パラメータ設定
params <- list(
  mu_A = 0.1, sigma_A = 0.2,
  mu_B = 0.2, sigma_B = 0.3,
  rho  = 0.2
)

# データ生成
df_short <- tibble(
  wa = seq(-1, 2, by = 0.01), # 空売りを考慮して範囲拡大
  wb = 1 - wa
) |>
  mutate(
    mu_p = wa * params$mu_A + wb * params$mu_B,
    sigma_p = sqrt(wa^2 * params$sigma_A^2 + wb^2 * params$sigma_B^2 +
                   2 * params$rho * wa * wb * params$sigma_A *
                    ↪  params$sigma_B),
    wa_int = round(wa * 100), # 100 倍して整数化

    # 条件:
    # 1. wa が -1 以上 2 以下 (wa_int が -100 ~ 200)
    # 2. wa が 0.1 刻み (wa_int が 10 で割り切れる)
    is_label_target = (wa_int >= -100 & wa_int <= 200) & (wa_int %% 10
     ↪  == 0),

    label = if_else(is_label_target,
                    sprintf("%.1f, %.1f", wa, wb),
                    NA_character_)
  )

# 作図
ggplot(df_short, aes(x = mu_p, y = sigma_p)) +
  geom_line() +
  # ラベルがある点 (0 <= wa <= 1) のみポイントとテキストを表示
  geom_point(data = df_short |> filter(!is.na(label)), size = 2) +
  geom_text(aes(label = label), na.rm = TRUE, vjust = -1, size = 3) +

  coord_flip() +
  labs(x = "期待リターン", y = "標準偏差 (リスク)") +
  mystyle
```

Listing 2.6 安全資産の導入 p.68

```
# パラメータ設定
params <- list(
  R_F     = 0.01, # 安全資産利子率
  mu_A    = 0.1,  # 銘柄 A 期待リターン
  sigma_A = 0.2   # 銘柄 A 標準偏差
)

# データ生成
df_safe <- tibble(
  wa = seq(-1, 2, by = 0.01) # 銘柄 A への投資比率
) |>
  mutate(
    wf = 1 - wa, # 安全資産への投資比率
    mu_p    = params$R_F + wa * (params$mu_A - params$R_F),
    sigma_p = abs(wa) * params$sigma_A, # 空売り時は絶対値

    # ラベルの作成
    wa_int = round(wa * 100),
    is_label_target = (wa_int >= 0 & wa_int <= 100) & (wa_int %% 10 ==
     ↪  0),
    label = if_else(is_label_target,
                    sprintf("%g, %g", wa, wf),
                    NA_character_)
  )

# 作図
ggplot(df_safe, aes(x = mu_p, y = sigma_p)) +
  geom_line() +
  geom_point(data = df_safe |> filter(!is.na(label)), size = 2) +
  geom_text(aes(label = label), na.rm = TRUE, vjust = -1, size = 3) +

  coord_flip() +

  labs(x = "期待リターン", y = "標準偏差（リスク）") +
  mystyle
```

Listing 2.7 効率的フロンティア

```
# --- 1. パラメータ設定 ---
params <- list(
  mu_A    = 0.1, sigma_A = 0.2,  # 株式 A の標準偏差
  mu_B    = 0.2, sigma_B = 0.3,  # 株式 B の標準偏差
  rho     = 0.2   # 相関係数
)

# --- 2. 最小分散ポートフォリオ (MVP) の算出 ---
# 分散・共分散
cov_ab <- params$rho * params$sigma_A * params$sigma_B
var_a  <- params$sigma_A^2
var_b  <- params$sigma_B^2

# MVP における資産 A のウェイト w_mvp
# 公式: w = (Var(B) - Cov(A,B)) / (Var(A) + Var(B) - 2Cov(A,B))
w_mvp <- (var_b - cov_ab) / (var_a + var_b - 2 * cov_ab)

# MVP のリターンとリスク（しきい値として使用）
mu_mvp    <- w_mvp * params$mu_A + (1 - w_mvp) * params$mu_B
sigma_mvp <- sqrt(w_mvp^2 * var_a + (1 - w_mvp)^2 * var_b + 2 * w_mvp
 ↪  * (1 - w_mvp) * cov_ab)

# --- 3. プロット用データの作成 ---
df_plot <- tibble(
  w = seq(-1, 2, by = 0.001)
) |>
  mutate(
    mu_p = w * params$mu_A + (1 - w) * params$mu_B,
    sigma_p = sqrt((w * params$sigma_A)^2 +
                   ((1 - w) * params$sigma_B)^2 +
                   2 * w * (1 - w) * cov_ab),
    frontier_type = if_else(mu_p >= mu_mvp, "効率 (上部)", "非効率
     ↪  (下部)")
  )

# --- 4. プロット作成 ---
ggplot(df_plot, aes(x = sigma_p, y = mu_p)) +
  geom_path(
    aes(color = frontier_type,
    linetype = frontier_type)
    ) +
  geom_vline(xintercept = 0.25, color = "green", linetype = "dashed")
```

Listing 2.8 3 資産の効率的フロンティア

```
# --- 1. パラメータ設定 ---
params <- list(
  mu_A    = 0.1,  sigma_A = 0.2,
  mu_B    = 0.2,  sigma_B = 0.3,
  rho     = 0.2,  R_F     = 0.01
)

# --- 2. データの作成 ---
# 2-1. リスク資産のみのフロンティア
df_frontier <- tibble(
  w = seq(-1, 2, by = 0.001)
) |>
  mutate(
    mu_p = w * params$mu_A + (1 - w) * params$mu_B,
    sigma_p = sqrt((w * params$sigma_A)^2 + ((1 - w) *
     ↪  params$sigma_B)^2 +
                   2 * w * (1 - w) * params$rho * params$sigma_A *
                    ↪  params$sigma_B),
    sharpe_ratio = (mu_p - params$R_F) / sigma_p
  )

# 2-2. 接点ポートフォリオ
tangency_port <- df_frontier |> slice_max(sharpe_ratio, n = 1)

# 2-3. 資本市場線 (CML)
df_cml <- tibble(
  sigma_p = c(0, 0.6),
  mu_p    = c(params$R_F, params$R_F + tangency_port$sharpe_ratio *
   ↪  0.6)
)

# --- 3. プロット作成 ---
ggplot() +
  # A. 資本市場線 (CML)
  geom_line(
    data = df_cml,
    aes(x = sigma_p, y = mu_p),
    color = "red"
  ) +

  # B. リスク資産のみのフロンティア
  geom_path(
```

Listing 2.9 証券市場線のプロット

```
# ------------------------------------------------------
# 1. パラメータ設定（本文の記述と整合するように調整した数値）
# ------------------------------------------------------
# ※ ユーザーの手元に前の章のデータがある場合はそれを読み込んでください。
# ここでは本文中の「A:47%, B:53%」「Beta A:0.63, Beta B:1.33」
# と概ね一致するパラメータを設定します。

mu_A <- 0.05       # 銘柄 A の期待リターン
sig_A <- 0.10      # 銘柄 A の標準偏差
mu_B <- 0.12       # 銘柄 B の期待リターン
sig_B <- 0.20      # 銘柄 B の標準偏差
rho <- 0.2         # 相関係数
Rf <- 0.01         # 安全利子率

# 共分散
cov_AB <- rho * sig_A * sig_B

# ------------------------------------------------------
# 2. 接点ポートフォリオ（市場ポートフォリオ）の計算
# ------------------------------------------------------
# 過剰リターンベクトル
R_excess <- c(mu_A - Rf, mu_B - Rf)
# 分散共分散行列
Sigma <- matrix(c(sig_A^2, cov_AB, cov_AB, sig_B^2), nrow = 2)

# 接点ポートフォリオのウェイト計算（解析解）
Sigma_inv <- solve(Sigma)
w_tangency_unscaled <- Sigma_inv %*% R_excess
w_M <- w_tangency_unscaled / sum(w_tangency_unscaled) # ウェイトの和を 1
 ↪ にする

# 市場ポートフォリオの期待リターンとリスク
mu_M <- as.numeric(t(w_M) %*% c(mu_A, mu_B))
sig_M <- as.numeric(sqrt(t(w_M) %*% Sigma %*% w_M))

# Beta の計算: Cov(Ri, RM) / Var(RM)
# Cov(Ra, Rm) = w_A*Var(A) + w_B*Cov(A,B)
cov_AM <- w_M[1] * sig_A^2 + w_M[2] * cov_AB
cov_BM <- w_M[1] * cov_AB + w_M[2] * sig_B^2

beta_A <- cov_AM / sig_M^2
beta_B <- cov_BM / sig_M^2
```

第 3 章

R 言語入門

Listing 3.1 いろいろな設定

```
pacman::p_load(
    tidyverse, # 便利なパッケージ群
    ggthemes, # ggplot2 のテーマ集
    microbenchmark, # ベンチマーク測定
    Rcpp # R と C++ の連携
    )
mystyle <- list(
  theme_economist_white(
    # gray_bg = FALSE,
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),
  theme(
    text = element_text(size = 12), # フォントファミリーは上で指定済みなの
    ↪ で省略可
    axis.title = element_text(size = 12)
  )
)

knitr::opts_chunk$set(
  class.source = "numberLines lineAnchors"
#   class.output = "numberLines lineAnchors chunkout"
  )
```

プログラミング言語にはいろんな種類があるけれど、今回学習する **R 言語**は、インタプリタ型とよばれるもので、コンパイルという作業の必要が無く、書いたらすぐ実行でき

る仕様となっています。たとえば、教科書にあるように

```
100 / (1 + 0.1)
```

```
[1] 90.90909
```

を実行すれば、結果がすぐ表示されます。Rstudio とか Visual Studio Code とか Antigravity を使って、上のような R ソースコードを一気に書いてまとめて実行するための**スクリプト・ファイル**を作成します。

ソースコードを書くにあたり注意する点が 4 つあります。

1. **大文字と小文字は区別される**ので、x と X は別の変数として扱われます。
2. **半角スペース**は、**区切り文字**として扱われるので、x <- 100 と x<-100 は同じ意味になります。
3. **改行**も、**区切り文字**として扱われます。長いソースコードは改行して読みやすくしましょう。
4. **コメント**は、#から行末までの文字列がコメントとして扱われ、実行されません。プログラムの内容を説明するためにたくさん書いて残しておきましょう。

3.1　R の基本的な機能

3.1.1　スカラー変数の定義

この学習を通じて**変数** (variable) とは、**数値や文字といったデータを格納するための箱**を表し、中に何が入っているのかにより、スカラー変数、ベクトル、行列、データフレームなどに分類されます。まずは、スカラー変数の定義を学びます。

スカラー (scalar) とは、大きさだけで決まる量のことで、つまり、**1 つの数値**を指します。R 言語ではスカラー変数を定義するには、<-を使います。たとえば、x <- 100 と書けば、x というスカラー変数に 100 という数値を格納できます。このとき、<-は代入演算子と呼ばれ、右辺の値を左辺の変数に代入するという意味です。また、x という変数を**左辺値** (left-hand side)、100 という数値を**右辺値** (right-hand side) と呼びます。

```
x <- 100 # 代入演算子<- の前後に半角スペースを入れるのがお作法
```

この中身を表示されるには、print() 関数を使います。

```
print(x) # x の中身を表示
```

```
[1] 100
```

あるいは

```
x
```

```
[1] 100
```

でも表示されます。

3.1.2 ベクトル変数の定義

R では#の後ろの文章はコメントとして扱われ、実行されません。コメントはプログラムの内容を説明するためにたくさん書いて残しておきましょう。

ベクトル (vector) とは、大きさと向きで決まる量のことで、つまり、**複数の数値**を指します。R 言語ではベクトル変数を定義するには、`c()` を使います。たとえば、`x <- c(1, 2, 3)` と書けば、x というベクトル変数に 1, 2, 3 という数値を格納できます。このとき、`c()` はベクトルを作る関数と呼ばれ、`1, 2, 3` という数値を引数として与えています。

```
x <- c(1, 5, 9) # x に 1 と 5 と 9 を要素とするベクトルを代入
print(x)
```

```
[1] 1 5 9
```

等差数列を作る関数に `seq()` 関数があります。`seq()` は 3 つの引数をとり、

- `from`：始点
- `to`：終点
- `by`：差分

を指定します。たとえば、2000 年から 2020 年を表す年度の変数を `year` として定義するには、

```
year <- seq(from = 2000, to = 2020, by = 1)
print(year)
```

```
 [1] 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
[16] 2015 2016 2017 2018 2019 2020
```

と書けば、2000 から 2020 までの公差 1 の等差数列を作ります。`seq()` 変数の引数には、`from` と `to` と `by` の 3 つの引数を指定することができますが、`from` と `to` のみを指定することもできます。このとき、`by` の値は 1 となります。次のように書いても、上と同じ結果を得ることができます。

```
seq(2000,2020)
```

```
 [1] 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
[16] 2015 2016 2017 2018 2019 2020
```

ベクトルの要素数を知るには、`length()` 関数を使います。

```
length(year) # year の要素数を表示
```

```
[1] 21
```

ベクトル変数 `year` の中には 21 個の要素があることがわかります。

3.1.2.1 ベクトルの要素の取り出し

複数の要素をもつベクトルから、一部の要素を取り出すには、`[]` を使います。たとえば、x の 2 番目の要素を取り出すには、`x[2]` と書きます。このとき、`[]` は添字演算子と

呼ばれ、2 という添字を引数として与えています。添字は 1 から始まります。

上の year から 2000 を取り出すには、year[1]、2020 を取り出すには year[20] と書きます。次のような書き方で、好きな要素を指定して取り出すことができます。

```
year[1] # 1 番目のデータを取り出す
```

```
[1] 2000
```

```
year[20] # 20 番目のデータを取り出す
```

```
[1] 2019
```

```
year[2:5] # 2 番目から 5 番目のデータを取り出す
```

```
[1] 2001 2002 2003 2004
```

```
year[c(5,10)] # 1 番目と 20 番目のデータを取り出す
```

```
[1] 2004 2009
```

```
year[6:length(year)] # 6 番目から最後のデータを取り出す
```

```
 [1] 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
[16] 2020
```

3.1.2.2　現在価値の計算

今の時点を $t = 0$ として、T 年後に確実に得られるキャッシュ・フロー CF_T の現在価値 PV_0 は、

$$PV_0 = \frac{CF_T}{(1+r)^T}$$

と書けます。たとえば 1 年後に確実に受け取れる 100 万円の現在価値 PV_0 を計算してみます。いま、無リスク利子率 r は 10% とします。

```
100 / (1 + 0.1)^1
```

```
[1] 90.90909
```

次に、この無リスク利子率 r が変化した場合の現在価値の計算を考えます。まず、無リスク利子率のベクトルを定義します。

```
# 下の 2 つは同じ結果
R <- seq(from = 0.1, to = 0.2, by = 0.01)  # 省略せずに書いた場合
R <- seq(0.1, 0.2, 0.01) # 略した場合
```

次に、無リスク利子率が変化した場合の現在価値を計算します。

```r
PV <- 100 / (1 + R)^1
print(PV)
```

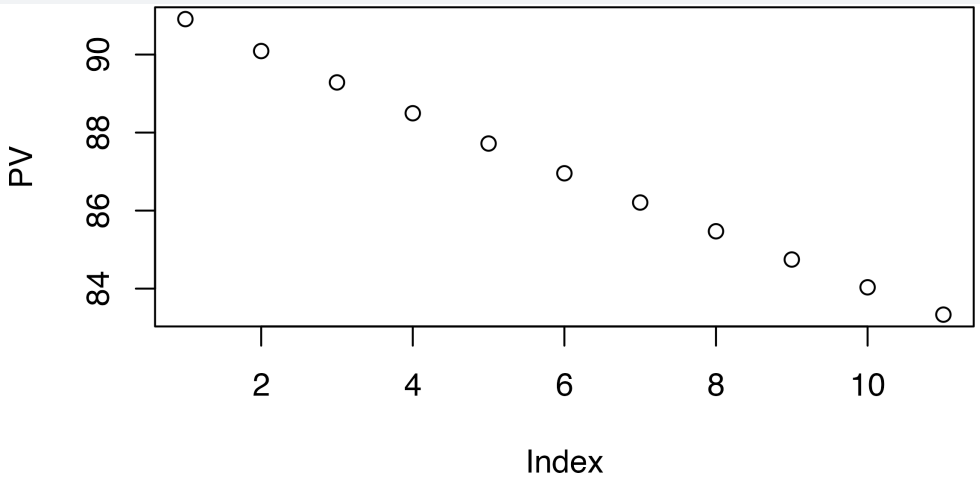
```
 [1] 90.90909 90.09009 89.28571 88.49558 87.71930 86.95652 86.20690 85.47009
 [9] 84.74576 84.03361 83.33333
```

無リスク利子率が 0.1 から 0.2 まで 0.01 ずつ変化した場合の現在価値が計算されました。この結果をグラフにしてみます。

3.1.3 基本パッケージ plot による作図

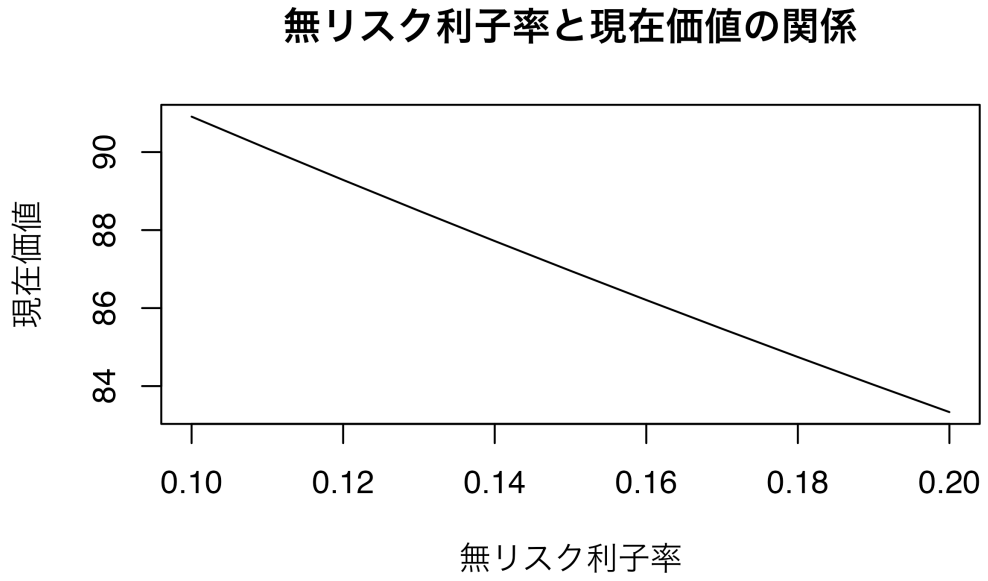
とりあえずサクッと作図してデータをチェックしたいとき、もとから R 言語に組み込まれている基本関数 plot() が便利です。先ほど作成したベクトル変数 PV をグラフにしてみます。

```r
plot(PV)
```



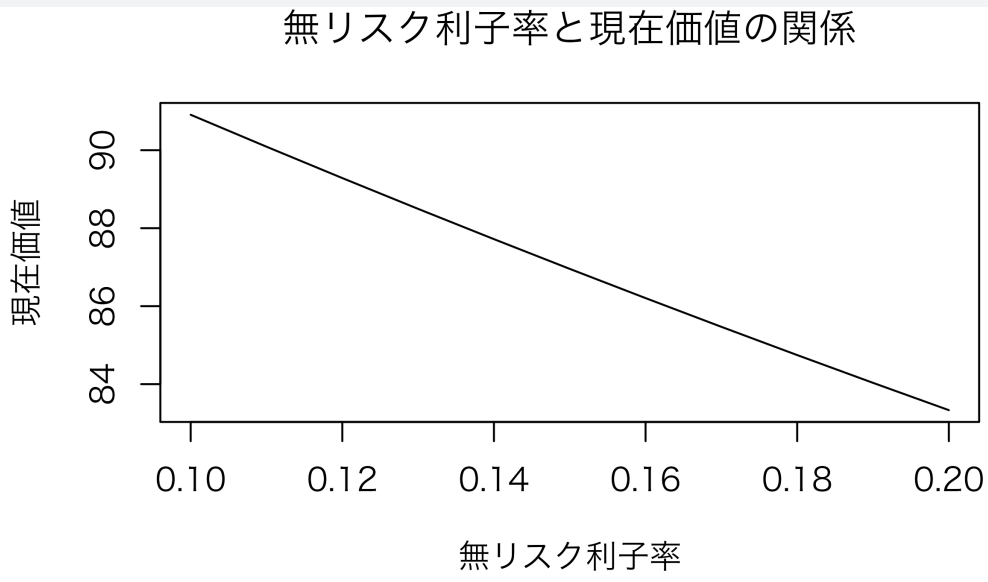
いま、PV は 11 個の要素をもつベクトル変数なので、データを左から順番に並べた散布図 (scatter diagram) が作成されています。これだと何のグラフか分かりづらいので、いろいろとオプションを指定してみます。

```r
plot(
    x = R, # x 軸のデータ
    y = PV, # y 軸のデータ
    xlab = "無リスク利子率",
    ylab = "現在価値",
    main = "無リスク利子率と現在価値の関係",
    type = "l" # 線グラフ
)
```



Mac だと文字化けしてしまいました。そこで文字コードを指定します。Windows だとこの作業は不要です。

```
par(family = "HiraKakuProN-W3") # Mac の場合のみ
plot(
    x = R, # x 軸のデータ
    y = PV, # y 軸のデータ
    xlab = "無リスク利子率", # x 軸のラベル
    ylab = "現在価値", # y 軸のラベル
    main = "無リスク利子率と現在価値の関係", # グラフのタイトル
    type = "l" # 折れ線グラフ
)
```



3.2 for 文の使い方

プログラミングの基本要素である

- 繰り返し
- 分岐
- 関数

の最初の要素である「繰り返し」を行うための文法が for 文です。for 文は、ある処理を繰り返し行うための文法です。たとえば、1 から 10 までの整数を順番に表示するには、次のように書きます。

```
for (i in 1:10) { # i は 1 から 10 まで
    print(i) # i を表示
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

この文の構造は、基本的には

```
for (好きな変数 in 繰り返す範囲) {
    繰り返したい処理
}
```

となっています。

たとえば、教科書のように、

- 初期投資 100 万円
- 1 年後に 50 万円のキャッシュ・フロー
- 2 年後に 50 万円のキャッシュ・フロー
- 3 年後に 50 万円のキャッシュ・フロー

という投資プロジェクトの現在価値を計算する場合、愚直に書くと次のようになります。

```
NPV1 <- -100 +
    50 / (1 + 0.1)^1 +
```

```
    50 / (1 + 0.1)^2 +
    50 / (1 + 0.1)^3
print(NPV1)
```

```
[1] 24.3426
```

この上のコードの 2 行目から 4 行目はほぼ同じ内容なので、数字が変化しているところに注目し、for 文を使って書き換えてみます。ここでは^1 のところが 1 ずつ大きくなってます。この部分を i という変数に置き換えてみます。ついでに、後で変化させることがあるかもしれない部分をすべて変数として定義しておきます。

```
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
CF <- 50 # キャッシュ・フロー

for (i in 1:3) { # i は 1 から 3 まで
    NPV <- NPV + CF / (1 + R)^i # 現在価値の計算
}
print(NPV)
```

```
[1] 24.3426
```

愚直に計算した場合の同じ結果となりました。これを 10 年間の現在価値を計算する場合だとすると、

```
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
NPV1 <- NPV +
    50 / (1 + R)^1 +
    50 / (1 + R)^2 +
    50 / (1 + R)^3 +
    50 / (1 + R)^4 +
    50 / (1 + R)^5 +
    50 / (1 + R)^6 +
    50 / (1 + R)^7 +
    50 / (1 + R)^8 +
    50 / (1 + R)^9 +
    50 / (1 + R)^10
print(NPV1)
```

```
[1] 207.2284
```

と面倒くさいことこの上ないですが、for 文を使えば、

```r
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
for (i in 1:10) { # i は 1 から 10 まで
    NPV <- NPV + 50 / (1 + R)^i
}
print(NPV)
```

```
[1] 207.2284
```

と短く書くことができます。使いこなせるように練習しておきましょう。次のように、print() 関数の位置を変えた場合、どうなるか考えてみてください。

```r
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
for (i in 1:3) { # i は 1 から 10 まで
    print(NPV)
    NPV <- NPV + 50 / (1 + R)^i
}
```

```
[1] -100
[1] -54.54545
[1] -13.22314
```

```r
print(NPV)
```

```
[1] 24.3426
```

この場合、最初に NPV の中を表示し、次に 1 期目の現在価値を計算し、またその結果を表示し、2 期目の現在価値を計算し・・・という順番で繰り返しが行われるので、計算の途中経過が表示されることになります。

3.2.1 if 文

次に、プログラミングの基本要素である

- 繰り返し
- **分岐**
- 関数

のうち分岐を行うための文法が if 文です。if 文は、ある条件を満たす場合にのみ処理を行うための文法です。たとえば、ある変数 x が 0 より大きい場合にのみ、その変数を表示するには、次のように書きます。

```r
x <- -1
if (x > 0) { # x が 0 より大きい場合
    print(x) # x を表示
```

```
}
```

この文の構造は、基本的には

```
if (条件) {
    条件を満たす場合に実行する処理
}
```

のようになっています。この if 文を使って、NPV が 0 より大きい場合にのみ、「プロジェクトを実行！」と表示されるようにしてみます。

```
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
for (i in 1:10) { # i は 1 から 10 まで
    NPV <- NPV + 50 / (1 + R)^i
}
if (NPV > 0) { # NPV が 0 より大きい場合
    print("プロジェクトを実行！") # 文字列を表示
}
```

```
[1] "プロジェクトを実行！"
```

ここでは NPV の値が 207.2284 となりプラスになっているので、「プロジェクトを実行！」と表示されます。

3.3　NPV と割引率の関係の可視化

無リスク利子率が 0.1 から 0.2 まで 0.01 ずつ変化した場合の現在価値 NPV の値を計算してみます。

```
R <- seq(0.1, 0.2, 0.01) # 無リスク利子率
N <- length(R) # 無リスク利子率の要素数 11 個
NPV <- rep(NA, N) # ベクトル変数に N 個の NA を代入

for (i in 1:N) { # i は 1 から N まで
    NPV[i] <- -100 # 初期投資
    for (j in 1:3) { # j は 1 から 3 まで
        NPV[i] <- NPV[i] + 50 / (1 + R[i])^j # 現在価値
    }
}
print(NPV) # 11 個の現在価値を表示
```

```
[1] 24.342600 22.185736 20.091563 18.057630 16.081601 14.161256 12.294477
```

```
[8] 10.479248  8.713646  6.995838  5.324074
```

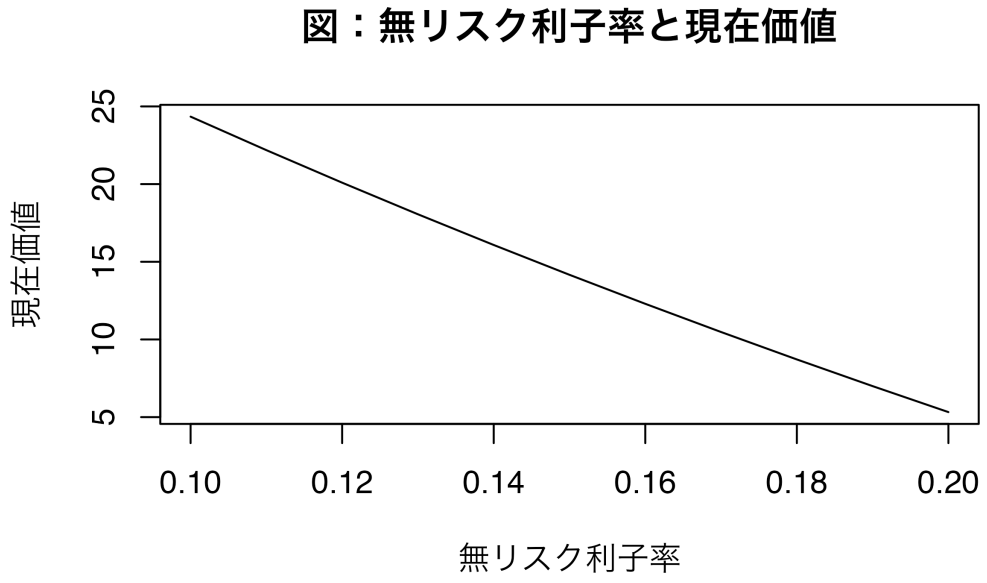
少し複雑な構造しているので、順番に説明します。

- 1 行目は、無リスク利子率のベクトル変数 R を定義しています。ここでは、0.1 から 0.2 まで 0.01 刻みのデータを作成しています。
- 2 行目は、ベクトル変数 R の要素数を N として定義しています。ここでは、N は 11 となります。
- 3 行目は、ベクトル変数 NPV に N 個の NA を代入しています。NA は Not Available の略で、欠損値を表します。NA を代入することで、空っぽの箱が 11 個入ったベクトル変数 NPV を用意します。
- 4 行目から 9 行目は、for 文を使って、NPV の中身を計算しています。for が 2 回出てきているので、二重に繰り返しの処理を行っています。これを**ネスト**と呼びます。1 つのめ for は i が 1 から N(ここでは 11) まで変化し、2 つめの for は j が 1 から 3 まで変化します。1 つめの for 文の i が 1 のとき、次の for 文の j が 1 から 3 までの処理を繰り返し、次に 1 つめの for 文の i が 2 のとき、次の for 文の j が 1 から 3 までの処理を繰り返し・・・という順番で処理が行われます。
- 10 行目は、NPV の中身を表示しています。

この結果をグラフにしてみます。

```
plot(
    x = R, # x 軸のデータ
    y = NPV, # y 軸のデータ
    xlab = "無リスク利子率", # x 軸のラベル
    ylab = "現在価値", # y 軸のラベル
    main = "図：無リスク利子率と現在価値", # グラフのタイトル
    type = "l" # 線グラフ
)
```

TAB キーを使って、インデントを行い、ソースコードのまとまりをわかりやすくしています。インデントは、プログラムの構造をわかりやすくするために行います。インデントを行うときは、半角スペース 2 つか 4 つを使います。どちらを使っても構いませんが、どちらかに統一することが大切です。



3.3.0.1　ベクトル化

上のコードは、for 文を使って、NPV の中身を計算しています。しかし、R 言語では、for 文を使わずに、ベクトルを使って、同じことを行うことができます。このように、for 文を使わずに、ベクトルを使って処理を行うことを**ベクトル化**と呼びます。ベクトル化を行うと、処理が高速化されることがあります。

```
R <- 0.1 # 無リスク利子率 10%
CF <- c(-100, 50, 50, 50) # キャッシュ・フローのベクトル
year <- 0:3 # 年度のベクトル
PV_CF <- CF / (1 + R)^year # 各期の現在価値を計算
NPV <- sum(PV_CF) # 現在価値の合計
print(NPV)
```

```
[1] 24.3426
```

3.4　独自関数の定義の仕方

プログラミングの基本要素である

- 繰り返し
- 分岐
- **関数**

の作り方について説明します。R では自分で関数を定義することができます。関数を定義することで、同じ処理を何度も書く必要がなくなり、プログラムの見通しがよくなります。例えば、足し算をする関数 my_add() を定義してみます。

```r
my_add <- function(x, y){
    x + y
}
```

この関数の構造は、

```r
好きな関数名 <- function(引数 1, 引数 2){
    処理内容
}
```

となっています。つまり、この独自関数 `my_add()` は、`x` と `y` という 2 つの引数 (ひきすう) を足し合わせる関数です。数学的に書くなら、

$$f(x, y) = x + y$$

となります。これは f という関数は 2 つの引数を足す関数であるという意味になっています。作成した独自関数 `my_add()` を使ってみます。

```r
my_add(1, 2)
```

```
[1] 3
```

3 が出力されました。

このように、独自関数を作成する場合には、

1. どのような引数を与えるのか？
2. それに対してどのような処理を行うのか？
3. 最終的にどの値を返す (出力させる) のか？

を考えておく必要があります。

では今までの流れで、現在価値を計算する関数を作成してみます。変化させたい値は、キャッシュフロー `CF` と無リスク利子率 `R` なので、その 2 つを引数とする独自関数を作成します。少し注意する必要がある点として、以下の計算例では `CF` の 1 番目の要素は初期投資額となることに注意しましょう。

```r
calc_PV <- function(CF, R) {
    PV <- CF[1] # 初期投資額なのでマイナスの値
    for (i in 2:length(CF)) { # i は 2 から CF の要素数まで
        PV <- PV + CF[i] / (1 + R)^(i - 1) # 現在価値
    }
    return(PV) # 現在価値を返す
}
```

この関数 `calc_PV()` を使って、現在価値を計算してみます。

```r
calc_PV(c(-100, 50, 50, 50), 0.1)
```

```
[1] 24.3426
```

ちゃんと計算されました。この関数を使って、無リスク利子率が 0.1 から 0.2 まで 0.01 ずつ変化した場合の現在価値を計算してみます。

```r
CF <- c(-100, 50, 50, 50) # キャッシュ・フローのベクトル
R <- seq(0.1, 0.2, 0.01) # 無リスク利子率のベクトル
calc_PV(CF,R)
```

```
 [1] 24.342600 22.185736 20.091563 18.057630 16.081601 14.161256 12.294477
 [8] 10.479248  8.713646  6.995838  5.324074
```

計算されました。関数の引数にデフォルトで値を設定することで、入力を楽にすることができます。例えば、無リスク利子率のデフォルト値を 0.1 に設定してみます。

```r
calc_PV <- function(CF, R = 0.1) {
    PV <- CF[1] # 初期投資額なのでマイナスの値
    for (i in 2:length(CF)) { # i は 2 から CF の要素数まで
        PV <- PV + CF[i] / (1 + R)^(i - 1) # 現在価値
    }
    return(PV) # 現在価値を返す
}
```

すると、無リスク利子率を指定しなくても、デフォルト値が使われるようになります。

```r
CF <- c(-100, 50, 50, 50) # キャッシュ・フローのベクトル
calc_PV(CF)
```

```
[1] 24.3426
```

ただ計算を間違えるもとにもなるので、なるべく省略せずに、しっかり書くことが大事です。

3.4.0.1　もっと凝った独自関数

繰り返し、分岐、関数というプログラミングの基本要素を勉強したので、もう少し複雑なプログラムを作成してみます。

まずは、引数に正の数字以外のもの、あるいは文字列を入力した場合にエラーを表示する関数を作成します。

```r
calc_PV_new <- function(CF, R) {
    if (R <= 0) {
        stop("無リスク利子率は正の値を入力してください。") # エラー処理
    }
```

```
    if ( !is.numeric(CF) ) {
        stop("キャッシュ・フローは数値を入力してください。")
    }
    if ( !is.numeric(R) ) {
        stop("無リスク利子率は数値を入力してください。")
    }

    PV <- CF[1]
    for (i in 2:length(CF)) {
        PV <- PV + CF[i] / (1 + R)^(i - 1)
    }
    return(PV)
}
```

できました。ついでに、NPV の計算結果とともに、NPV が 0 より大きい場合にのみ、「プロジェクトを実行！」と表示する機能も実装してみます。

```
calc_PV_new <- function(CF, R = 0.1) {
    if (R <= 0) {
        stop("無リスク利子率は正の値を入力してください。") # エラー処理
    }
    if ( !is.numeric(CF) ) {
        stop("キャッシュ・フローは数値を入力してください。")
    }
    if ( !is.numeric(R) ) {
        stop("無リスク利子率は数値を入力してください。")
    }

    PV <- CF[1]
    for (i in 2:length(CF)) {
        PV <- PV + CF[i] / (1 + R)^(i - 1)
    }

    if (PV >= 0) { # NPV が 0 より大きい場合
    paste0("NPV が", round(PV, digits = 2), "なので、プロジェクトを実
    ↪ 行！") # 文字列を表示
    } else {
    paste0("NPV が", round(PV, digits = 2), "なのでプロジェクト中止！") #
    ↪ 文字列を表示
```

```
    }
}
```

いろいろ駆使してより短く簡単に書くなら、

```
calc_PV <- function(CF, R = 0.1) {
  if (!is.numeric(CF) || !is.numeric(R) || R <= 0) {
    stop("キャッシュ・フローと無リスク利子率は数値を入力し、無リスク利子率は正
     ↪ の値を入力してください。")
  }
  PV <- sum(sapply(1:length(CF), function(i) CF[i] / (1 + R)^(i - 1)))

  if (PV >= 0) {
    paste0("NPV が", round(PV, digits = 2), "なので、プロジェクトを
     ↪ 実行！")
  } else {
    paste0("NPV が", round(PV, digits = 2), "なのでプロジェクト中止！")
  }
}
```

```
CF <- c(-100, 50, 50, 50)
calc_PV(CF)
```

```
[1] "NPV が 24.34 なので、プロジェクトを実行！"
```

うまくいきました。ちょっとキャッシュフローのベクトルを変化させて、初期投資を-200 にすると、

```
CF <- c(-200, 50, 50, 50)
calc_PV(CF)
```

```
[1] "NPV が-75.66 なのでプロジェクト中止！"
```

ちゃんと中止のメッセージが出ました。

このように、分岐、繰り返し、関数を駆使して、様々なプログラムを作成することができます。プログラミングの基本要素を使いこなせるように、練習を重ねてください。まずは教科書に書いてあるソースコードを自分の PC 上で実行してみてください。その際は、コピペせずに自分で入力するようにしてください。

3.4.0.2　付録：ベクトル化で早くなるのか？

どれほど高速化されるのかを確認するため、100 万年分の現在価値を計算してみます。最初に松浦の R 環境を確認してみます。コンピューターは Mac mini で、CPU は M4 Pro、メモリは 24GB、MacOS Tahoe 26.3 です。R やパッケージのバージョンは以下の

通りです。

```
sessionInfo()
```

```
R version 4.5.0 (2025-04-11)
Platform: aarch64-apple-darwin20
Running under: macOS 26.3

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version

locale:
[1] C.UTF-8/C.UTF-8/C.UTF-8/C/C.UTF-8/C.UTF-8

time zone: Asia/Tokyo
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] Rcpp_1.1.0           microbenchmark_1.5.0 ggthemes_5.1.0
 [4] lubridate_1.9.4      forcats_1.0.1        stringr_1.6.0
 [7] dplyr_1.1.4          purrr_1.2.0          readr_2.1.5
[10] tidyr_1.3.1          tibble_3.3.0         ggplot2_4.0.0
[13] tidyverse_2.0.0

loaded via a namespace (and not attached):
 [1] gtable_0.3.6       jsonlite_2.0.0     compiler_4.5.0     tinytex_0.57
 [5] tidyselect_1.2.1   magick_2.9.0       systemfonts_1.3.1  scales_1.4.0
 [9] textshaping_1.0.3  yaml_2.3.10        fastmap_1.2.0      R6_2.6.1
[13] generics_0.1.4     knitr_1.50         pillar_1.11.1      RColorBrewer_1.1-
3
[17] tzdb_0.5.0         rlang_1.1.6        stringi_1.8.7      xfun_0.53
[21] S7_0.2.0           timechange_0.3.0   cli_3.6.5          withr_3.0.2
[25] magrittr_2.0.4     digest_0.6.37      grid_4.5.0         hms_1.1.3
[29] lifecycle_1.0.4    vctrs_0.6.5        evaluate_1.0.5     glue_1.8.0
[33] farver_2.1.2       ragg_1.5.0         pacman_0.5.1       rmarkdown_2.29
[37] tools_4.5.0        pkgconfig_2.0.3    htmltools_0.5.8.1
```

```
R.version
```

```
               _
platform       aarch64-apple-darwin20
arch           aarch64
os             darwin20
system         aarch64, darwin20
status
major          4
minor          5.0
year           2025
month          04
day            11
svn rev        88135
language       R
version.string R version 4.5.0 (2025-04-11)
nickname       How About a Twenty-Six
```

今回比較した 4 つの手法について、それぞれの特徴と、どのような場面で使うべきかをまとめました。適切な手法を選択する際の参考にしてください。

1. `for` ループ：1 つずつ順番に計算する、最も基本的で直感的な方法です。プログラムの構造が分かりやすく、複雑な条件分岐を書きやすいです。しかし、R は繰り返し処理のたびに「解釈」を行うため、回数が増えると極端に時間がかかります。データ数が少ない場合や、計算ロジックが非常に複雑でベクトル化が難しい場合に使います。
2. **ベクトル化**：R の得意技で、データを「まとめて」一気に計算する方法です。`for` ループに比べて劇的に高速で、コードも短くシンプルになります。しかし、計算のために巨大な一時データをメモリ上に作成するため、メモリ消費量が大きいです。データが巨大すぎるとメモリ不足で停止することがありますが、R では通常はこの方法が推奨されます。
3. **分割計算** / チャンク化：データを一定のサイズ（例：100 万件ずつ）に区切って、少しずつベクトル化計算を行う方法です。ベクトル化の速さを活かしつつ、メモリ消費量を低く抑えることができますが、コードが少し複雑になります。また、区切る処理が入る分、純粋なベクトル化よりわずかに遅くなります。億単位のデータなど、一気に計算するとメモリ不足になるような大規模データを扱う場合に有効です。
4. `Rcpp` (C++ 連携)：R の中から、より高速なプログラミング言語である「C++」を呼び出して計算する方法です。圧倒的に最速です。また、メモリ管理も効率的なので、大規模データでも軽快に動作します。C++ の知識が必要で、コードを書く難易度が高いですが、生成 AI を活用すれば初心者でも比較的簡単に導入できま

す。非常に大規模なシミュレーションや、計算速度が最重要となる場合に適しています。

以下では、それぞれの手法で現在価値を計算する関数を定義します。

比較してみます。ランダムな将来キャッシュフローを 1 万年分用意します。

```
set.seed(121)
n_years <- 10^6 # 100 万年分
# 初期投資-100 とランダムなキャッシュフロー
CF <- c(-100, runif(n_years - 1, min = 0, max = 100))
```

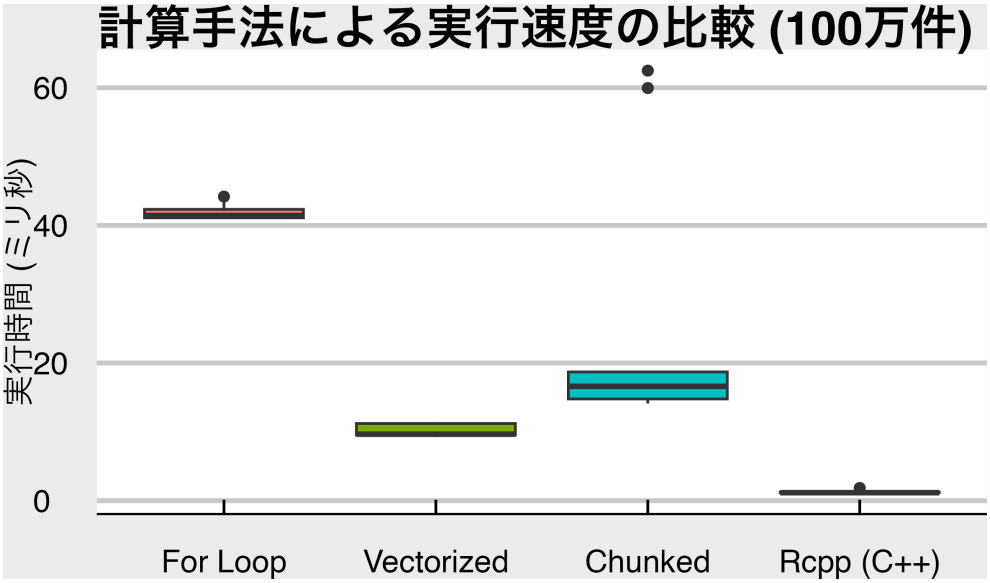
では、それぞれの方法で計算した速度を比較してみましょう。正確に比較するため、microbenchmark パッケージを使って、それぞれ 10 回ずつ計測し、その分布を確認します。

```
# 全ての手法をまとめて計測
res <- microbenchmark(
    "For Loop"    = calc_PV_for(CF),
    "Vectorized" = calc_PV_vec(CF),
    "Chunked"     = calc_PV_chunk(CF),
    "Rcpp (C++)" = calc_PV_cpp(CF, 0.1),
    times = 10, # 10 回繰り返して計測
    unit = "ms" # 単位をミリ秒に統一
)

# 結果の数値表示
print(res)
```

```
Unit: milliseconds
       expr       min        lq      mean    median        uq       max neval
   For Loop 41.019598 41.081098 41.903140 41.433144 42.515237 44.182379    10
 Vectorized  9.259112  9.514460 10.218545  9.664417 11.275246 11.373318    10
    Chunked 14.116833 14.772259 25.116395 16.609674 18.855695 62.485640    10
 Rcpp (C++)  1.127582  1.156036  1.252103  1.161940  1.288507  1.844631    10
```

```
# 結果の可視化（箱ひげ図）
ggplot(res, aes(x = expr, y = time / 1e6, fill = expr)) + # time を
 ↪ 1e6(100 万) で割ってミリ秒に変換
  geom_boxplot() + # 箱ひげ図を指定
  scale_y_continuous(labels = scales::comma) + # 軸の数値をカンマ区切りに
   ↪ (見やすくするため)
  labs(title = "計算手法による実行速度の比較 (100 万件)",
```

```
      y = "実行時間（ミリ秒）",
      x = "") + # x 軸ラベル（手法名）は自明なので空欄に
  theme_minimal() + # ベーステーマ（mystyle があればそちらに置き換えてくだ
   ↪ さい）
  mystyle +
  theme(legend.position = "none") # 色分けの凡例は不要なら消す
```



実行結果の解釈:

- `for` ループ: 最も遅くなります。R でのループ処理はオーバーヘッドが大きいためです。
- ベクトル化: for 文に比べて劇的に高速化します。**通常はこの方法が推奨**されます。
- Chunked: ベクトル化より少し遅くなりますが、メモリ消費を抑えられるメリットがあります。
- `Rcpp`: 圧倒的に高速です。C++ のレベルで最適化されるため、大規模なシミュレーションや反復計算では威力を発揮します。

これがベクトル化による実行速度の効率化です。とはいえ、演算に時間がかかるような大規模データや複雑なシミュレーションをするようになるまで、ベクトル化の恩恵はそれほど大きくないですし、巨大な中間ベクトルをメモリ上に生成するため、計算速度は速くなるがメモリ消費量は増えるます。ということなので今は気にせずに、読みやすく、確実に動くプログラムを書くことを心がけましょう。

3.5　演習

各自でやってみてください。

3.6 データフレーム入門

3.6.1 CSV ファイルの読み込み

R で CSV ファイルを読み込むには、readr パッケージの read_csv() 関数を使うのがよいでしょう。たとえば、data.csv というファイルを読み込むには、次のように書きます。

```
df <- readr::read_csv("data/ch03_daily_stock_return.csv")
```

この df というオブジェクトの型を見てみると,

```
class(df)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

spec_tbl_df, tbl_df, tbl, data.frame という 4 つの型が表示されます。data.frame という属性が含まれており，df が**データフレーム**であることを示しています。

あとは,

- nrow() 関数で行数を確認
- str() 関数で構造を確認
- mean() 関数で平均を計算
- sd() 関数で標準偏差を計算
- cor() 関数で相関係数を計算

などを使って，データの中身を確認してみましょう。

- which.min() 関数や which.max() 関数で最小値・最大値のインデックスを取得

例えば，最も日次リターンが高い日付を調べるには，次のようにします。

spec_tbl_df は，readr パッケージが読み込んだデータフレームに付与する属性で，データの仕様情報を保持しています。tbl_df は，tibble パッケージが提供するデータフレームの拡張型で，表示や操作がしやすくなっています。tbl は，dplyr パッケージが提供するデータフレームの拡張型で，データ操作が効率的に行えるようになっています。基本的には，data.frame として扱うことができます。

```
best_day_ID <- which.max(df$firm1)
best_day_ID
```

```
[1] 13
```

13 行目が最も日次リターンが高い日付なので，df からその行を取り出してみます。

```
df$date[best_day_ID]
```

```
[1] "2020-04-17"
```

3.7 ファクター型と日付型

3.7.1 ファクター型入門

ファクター型あるいは因子型 (factor) は，カテゴリカル変数を表現するためのデータ型で，分かりづらいものの，非常に便利かつ重要なデータの型なので，しっかり理解してお

きましょう。

数値型や文字列型を因子型に変換する方法には，

1. 基本関数 factor()
2. forcats パッケージの as_factor() 関数
3. forcats パッケージの fct_relevel() 関数

先ほど読み込んだデータの industry 変数は，文字列型ですが，これを因子型に変換してみます。

```
firm_ID <- c(1,2,3)
name <- c("Firm A", "Firm B", "Firm C")
industry <- c("Machinery", "Chemicals", "Machinery")

firm_data <- data.frame(
  firm_ID = firm_ID,
  name = name,
  industry = industry
)
```

この industry 変数はカテゴリー変数ですが，文字列型になっているので，因子型に変換してみます。

```
firm_data <- firm_data |>
    mutate(
        industry = forcats::fct_inorder(industry)
        )
class(firm_data$industry)
```

```
[1] "factor"
```

forcats パッケージには非常に便利な関数がたくさんあるので，ぜひドキュメントを参照してみてください。代表的なものに

- as_factor()：他の型から因子型に変換
- fct_inorder()：出現順にレベルを設定
- fct_order()：頻度順にレベルを設定
- fct_relevel()：レベルの順序を変更
- fct_recode()：レベルの名前を変更
- fct_collapse()：レベルをまとめる
- fct_lump()：頻度の低いレベルをまとめる

があります。

3.7.2 日付型入門

日付データとは、年月日や時間を表すデータです。たとえば、“2026-02-03” や”2025-01-01 00:00:01” のような形式で表されます。日付データは、文字列として記録されていることが多いので、日付型に変換する必要があります。日付データの操作には tidyverse の lubridate パッケージを使うと便利です。lubridate パッケージには、日付データを簡単に操作するための関数がたくさんあります。

- ymd()：年月日形式の文字列を日付型に変換
- mdy()：月日年形式の文字列を日付型に変換
- hms()：時分秒形式の文字列を時刻型に変換
- ymd_hms()：年月日時分秒形式の文字列を日付型に変換
- today()：今日の日付を取得
- now()：現在の日付と時刻を取得

たとえば，次のように日付っぽい値になっている文字列型なら、lubridate パッケージが判別して日付型に変換してくれます。今回は、2022-01-15 のように年月日の形式なので ymd() 関数を使います。

```
date <- c("2022-01-15", "2023/01/15")
class(date)
```

```
[1] "character"
```

文字列型の日付データを Date 型に変換したい場合は，次のようにします。

```
date <- lubridate::ymd(date) # 年月日
date
```

```
[1] "2022-01-15" "2023-01-15"
```

```
class(date)
```

```
[1] "Date"
```

3.8 外部パッケージ

pacman パッケージの p_load() 関数を使うと，CRAN に登録されているパッケージのうち，必要なパッケージを一括でインストール・読み込みできます。未インストールのパッケージなら自動でインストールして読み込み、インストール済みのパッケージならそのまま読み込みます。たとえば，tidyverse パッケージと skimr パッケージをインストール・読み込みするには，次のようにします。

```
pacman::p_load(tidyverse, skimr)
```

まれに，開発中のパッケージを GitHub からインストールしたい場合があります。その場合は，pacman パッケージの p_load_gh() 関数を使います。たとえば，username/repo という GitHub リポジトリからパッケージをインストール・読み込みするには，次のようにします。

```
pacman::p_load_gh("username/repo")
```

これで，GitHub からパッケージがインストールされ，読み込まれます。GitHub リポジトリにあるパッケージは CRAN に登録されていないため、自己責任で使用してください。

Listing 3.2 for 文とベクトル化の比較

```r
# for 文版
calc_PV_for <- function(CF, R = 0.1) {
  PV <- CF[1]
  n <- length(CF)
  if (n < 2) return(PV) # 要素数が 1 以下の場合は即リターン

  for (i in 2:n) {
    PV <- PV + CF[i] / (1 + R)^(i - 1)
  }
  return(PV)
}

# ベクトル版
calc_PV_vec <- function(CF, R = 0.1){
  year <- 0:(length(CF) - 1)

  PV_CF <- CF / (1 + R)^year
  NPV <- sum(PV_CF)
  return(NPV)
}

# ベクトル版一気に計算版
calc_PV_chunk <- function(CF, R = 0.1, chunk_size = 1e6) { # デフォルトで 100 万件ずつ処理
  n <- length(CF)
  total_npv <- 0

  # チャンクの開始位置を作成（例: 1, 1000001, 2000001...）
  starts <- seq(1, n, by = chunk_size)

  for (start in starts) {
    # 終了位置を計算（データの最後を超えないように min を使う）
    end <- min(start + chunk_size - 1, n)

    # 1. 必要な部分だけデータを切り出す（ここでメモリ消費を抑える）
    cf_part <- CF[start:end]

    # 2. 時間 t のベクトルを作成（全体の位置に合わせて調整）
    # CF[1] が t=0 に対応するため、インデックスから 1 を引く
    t_part <- (start:end) - 1

    # 3. 部分的な現在価値を計算して合計に加算
```

第 4 章

財務データの取得と可視化

```
pacman:: p_load(
    tidyverse,
    ggthemes
)
mystyle <- list(
  theme_economist_white(
    # gray_bg = FALSE,
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),
  theme(
    text = element_text(size = 12), # フォントファミリーは上で指定済みなの
    ↪  で省略可
    axis.title = element_text(size = 12)
  )
)
```

4.1　ディスクロージャー制度の概要とデータの入手先

4.1.1　法定開示と適時開示

	年次開示	四半期開示	重要事実
法定開示	有価証券報告書	四半期報告書	臨時報告書
適時開示	決算短信	四半期決算短信	適時開示

4.1.2　財務データの入手先

- **EDINET**：全上場企業の法定開示資料データベース，XBRL 形式も提供
- **TDnet**：上場企業の決算短信データベース, XBRL 形式も提供

XBRL(eXtensible Business Reporting Language) 形式で財務諸表などの主要情報を公開しています。XBRL 形式を学習しておくと，EDINET や TDnet から直接データを取得して R で分析することもできるので便利ですが，なかなか面倒なことが多いので，大学生なら大学が契約してくれているデータベースを活用しましょう。

4.2　R を利用した財務データの分析

4.2.1　tidyverse パッケージの概要

`tidyverse` とは，R 神 Wickham 氏が基本コンセプトを設定し，**整然データ** (tidy data) に対して一貫した記法でデータを扱えるパッケージ群です。インストールと読み込みは以下の通りです。

```
#install.packages("pacman") # まだなら 1 回だけ実行
pacman::p_load(tidyverse)
```

`tidyverse` パッケージを読み込むことで，次の代表的なパッケージが読み出されます。

- `dplyr` データハンドリング　めっちゃ使う
- `tidyr` tidy データにもっていく　使う
- `readr` データを読み込む めっちゃ使う
- `forcats` ファクター型変数の操作　めっちゃ使う
- `ggplot2` データの可視化　めっちゃ使う
- `purrr` 関数型プログラミングで使う　慣れてくると使う
- `tibble` data.frame ではなく tibble にする　あまり使わない
- `stringr` 文字列の加工・操作　ちょいちょい使う

4.2.2　財務データの読み込み

準備として，サポートサイトにある練習用のデータセット `ch04_financial_data.csv` をダウンロードして，作業ディレクトリに置いておきましょう。

`readr` パッケージの `read_csv()` 関数を使って，CSV ファイルを読み込みます。`read_csv()` 関数は，

作業ディレクトリの場所を確認するには `getwd()` を使います。作業ディレクトリを変更するときは，`setwd()` で作業ディレクトリを絶対パスで指定するとよいでしょう。

1. データの読み込みが高速かつ型の推論が柔軟
2. 基本の `data.frame` ではなく，その拡張版である `tibble` で返す
3. 列名を勝手に変換しない。
4. 文字列を勝手にファクター型にしない (`read.csv()` だと勝手にファクターに

なる)。

という利点があります。

Listing 4.1 データの読み込み

```
financial_data <- read_csv("data/ch04_financial_data.csv")
nrow(financial_data) # 行数
```

```
[1] 7920
```

```
ncol(financial_data) # 列数
```

```
[1] 11
```

```
head(financial_data, 5) # 最初の 5 行
```

```
# A tibble: 5 x 11
   year firm_ID industry_ID sales    OX   NFE     X     OA    FA    OL     FO
  <dbl>   <dbl>       <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>
1  2015       1           1 5261.  437.  NA    287. 13006. 3543. 4373.  2481.
2  2016       1           1 5949.  564.  50.7  513. 13866. 4642. 4534.  3960.
3  2017       1           1 6505.  691.  29.5  662. 13953. 7744. 5111.  6159.
4  2018       1           1 6846.  751.  86.5  665. 18818. 7285. 5137. 10124.
5  2019       1           1 7572.  959. 298.   660. 18190  9735. 5488. 11362.
```

この financial_data には、11 個の変数に観測値が 7920 個あることがわかります。

- year：年度
- firm_ID：企業 ID
- industry_ID：産業 ID
- sales：売上高
- OX：事業利益 (operating income)
- NFE：純金融費用 (net financial expenses)
- X：当期純利益 (net income)
- OA：事業資産 (operating assets)
- OL：事業負債 (operating liabilities)
- FE：金融資産 (financial assets)
- FO：金融負債 (financial obligations)

このデータフレームの構造を dplyr パッケージの glimpse() を使って確認します。

```
glimpse(financial_data)
```

```
Rows: 7,920
Columns: 11
$ year        <dbl> 2015, 2016, 2017, 2018, 2019, 2020, 2015, 2016, 2017, 2018~
```

```
$ firm_ID     <dbl> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4~
$ industry_ID <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ sales       <dbl> 5261.40, 5948.96, 6505.06, 6846.38, 7572.24, 7537.63, 3505~
$ OX          <dbl> 437.49, 564.14, 691.18, 751.29, 958.53, 778.37, 45.82, 51.~
$ NFE         <dbl> NA, 50.667498, 29.543157, 86.486500, 298.049774, -
65.45877~
$ X           <dbl> 286.64, 513.48, 661.64, 664.80, 660.48, 843.83, 40.07, 49.~
$ OA          <dbl> 13005.55, 13865.58, 13952.58, 18818.48, 18190.00, 20462.86~
$ FA          <dbl> 3543.43, 4642.16, 7743.99, 7284.72, 9735.13, 10274.25, 225~
$ OL          <dbl> 4372.96, 4534.22, 5111.22, 5137.28, 5487.96, 5371.38, 1840~
$ FO          <dbl> 2480.72, 3959.70, 6159.02, 10123.91, 11362.22, 13772.15, 2~
```

変数はすべて数値型 `double` になっていますが、`firm_ID` と `industry_ID` はカテゴリーを表す変数ですので、数値型ではなくファクター型に変換します。ついでに `year` は年度という時間を尺度なので、数値型ではなく `factor` 型に変換します。ここで重要なのは、`year` はただのファクター型ではなく、順序のあるファクター型とすることです。ここでは `forcats` パッケージの `as_factor()` を使います。

```
# firm_ID と industry_ID を factor 型に変換
financial_data <- financial_data |>
  mutate(
        year = as_factor(year),
        firm_ID = as_factor(firm_ID),
        industry_ID = as_factor(industry_ID)
  )
```

4.3　探索的データ分析

4.3.1　データセットの概要確認

データセットを操作するまえに，データの概要を大まかにつかむ必要があり，この作業を探索的データ分析 (exploratory data analysis) といいます。仮説などを持たず，とりあえず特徴や構造を理解するための方法です。データセットの概要を確認するために，`skimr` パッケージの `skim()` 関数を用います。

引数の型に応じて自動的に最適な結果を返す機能を**多態性** (polymorphism) といい，多態性をもつ関数を**総称関数** (generic function) という。

```
skimr::skim(financial_data)
```

Table4.2: Data summary

Name	financial_data
Number of rows	7920
Number of columns	11
Column type frequency:	
factor	3
numeric	8
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
year	0	1	FALSE	6	202: 1363, 201: 1356, 201: 1323, 201: 1319
firm_ID	0	1	FALSE	1515	1: 6, 2: 6, 3: 6, 4: 6
industry_ID	0	1	FALSE	10	3: 1760, 10: 1702, 7: 1334, 1: 1143

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
sales	0	1	166007.03	381980.29	205.34	16103.33	40430.74	118313.82	2496433.1	▇
OX	0	1	7968.91	25951.56	-353606.72	399.28	1602.88	5260.46	398034.5	▁ ▁ ▇ ▁ ▁

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
NFE	1	1	64.02	5941.34	-285383.86	-76.43	-1.19	41.36	331035.2	▁▁▇▁▁
X	0	1	7904.88	26910.18	-357624.83	383.27	1586.10	5204.60	572588.7	▁▇▁▁▁
OA	0	1	152272.76	453879.24	216.51	12559.93	30799.29	43469.20	7987936.	▇▁▁▁▁
FA	0	1	80185.35	422852.06	288.43	6835.07	19095.35	52117.92	2925061.	▇▁▁▁▁
OL	0	1	50260.88	148602.43	35.04	3964.84	10868.33	33110.89	2817974.	▇▁▁▁▁
FO	0	1	70680.61	290625.74	13.64	3757.44	11125.23	35446.03	7026923.	▇▁▁▁▁

さらに，データセットのある変数に含まれる固有な要素を抽出するには，`unique()` 関数を用います。

```
unique(financial_data$year) # financial_data の year 変数に含まれる固有要素
```

```
[1] 2015 2016 2017 2018 2019 2020
```

```
Levels: 2015 2016 2017 2018 2019 2020
```

```
# 2015, 2016, 2017, 2018, 2019, 2020
```

固有要素の数を確認するには，unique() 関数で取り出した要素の数を length() 関数で返します。企業-年の企業数と年度数を確認するには次のようにします。

Listing 4.2 企業数と産業数の確認

```
sort(unique(financial_data$year))
```

```
[1] 2015 2016 2017 2018 2019 2020
Levels: 2015 2016 2017 2018 2019 2020
```

```
financial_data$firm_ID |> n_distinct()
```

```
[1] 1515
```

```
financial_data$industry_ID |> n_distinct()
```

```
[1] 10
```

よってこのデータには 10 の産業、1515 の企業があることが分かります。

4.4 4.3.2 欠損データの処理

ほとんどのデータセットには，欠損値 (NA) が含まれているため，この欠損値の処理は非常に重要になります。欠損値の有無を確認するためには，complete.cases() 関数を用いるのが便利です。欠損値が含まれていると FALSE を返し，欠損値がないと TRUE を返します。

```
head(complete.cases(financial_data)) # 最初の６行の結果を表示
```

```
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

sum() 関数で，TRUE の個数を数え上げることもできます。

```
sum(complete.cases(financial_data)) # TRUE/FALSE を 1/0 に置き換えて合計
```

```
[1] 7919
```

欠損値の出現に何らかの傾向がある場合，欠損値の削除が**生存者バイアス** (survivorship bias) をもたらす可能性があります。たとえば，過去 20 年間にわたって連結財務諸表データに欠損値が含まれていない上場企業ばかりを分析すると，途中で倒産したり上場したりした企業は削除され，20 年間経営し続けている優良企業しかデータに残らない生存者バイアスが発生します。

このようなバイアスを考慮しなくても良いなら，欠損値をもつ個体 (unit) のデータ (行) を削除するのが単純な処理となります。このとき，tidyr パッケージに含まれる

drop_na() 関数を用いると簡単に欠損値を含む行を削除できます。基本関数の na.omit() でもよいですが，tidyr::drop_na() の方がオプションが豊富なのでおすすめです。

```
nrow(financial_data) # 欠損行を削除する前の行数
```

```
[1] 7920
```

```
nrow(drop_na(financial_data)) # 欠損行を削除した場合の行数
```

```
[1] 7919
```

```
financial_data <- drop_na(financial_data) # 欠損行を削除した上でデータを上
↪ 書き
# この作業には注意が必要である。オリジナルデータはそのまま残しておいたほうが良い
```

欠損値を含む行を削除するのではなく，欠損値に適切な推定値を代入することでサンプルサイズを減らさない方法も開発されていますが，欠損値の出現を説明する確率モデルを仮定し，その推定値を求める必要があります。

> **Note**
>
> 詳しくは髙橋・渡辺 (2019) 欠損データ処理：R による単一代入法と多重代入法を参照してください。
> さらに欠損値についての議論では，星野・岡田 (2016)「欠測データの統計科学―医学と社会科学への応用」岩波書店がめちゃめちゃ有用です。

4.5　データの抽出とヒストグラムによる可視化

4.5.1　条件にあうデータの抽出方法

教科書では複数の方法が紹介されているが，このメモでは tidyverse パッケージを用いた方法だけ取り上げます。具体的には，データベース操作のパッケージである dplyr の中の filter() 関数について説明します。さらに magrittr を用いたパイプ演算子 |> を用いたデータの受け渡しの記法を活用して，可読性の高いソースコードを書くことも紹介します。ここでは dplyr パッケージの filter() 関数であることを明示的に示すため、dplyr::filter() と書いていますが、dplyr パッケージを読み込んでいる場合は filter() と書いても同じです。

```
financial_data_2015 <- financial_data |>
    dplyr::filter(year == 2015) # year 変数が 2015 のデータを抽出
```

filter() で条件を満たすデータのみを取り出し，それを financial_data_2015 に代入している。

パイプ演算子|>は**左のオブジェクトを右の関数の第 1 引数に代入**する，という処理を行います。つまり，`x |> filter(year == 2015)` は，`filter(x, year == 2015)` と同じ意味になります。パイプ演算子を使うことで，データが次の処理に受け渡されていくプロセスが読みやすくなります。たとえば、

1. 欠損値を除去して，
2. 2015 年のデータを抽出し，
3. ROE を計算して，
4. 産業ごとに平均値を出す

というよく使いそうな処理を行いたい場合，tidyverse なら次のように書きます。

```
financial_data |>
    drop_na() |> # 欠損値を除去し，
    filter(year == 2015) |> # 2015 年のデータを抽出し，
    mutate(
        ROE = earnings / equity
        ) |> # ROE を計算し，
    summarise(
        mean_ROE = mean(ROE), # mean() で平均値を計算
        by = industry_ID # 産業ごとに
    )
```

基本関数の場合は、

```
financial_data <- na.omit(financial_data)
financial_data_2015 <- financial_data[financial_data$year == 2015, ]
financial_data_2015$ROE <- financial_data_2015$earnings /
 ↪  financial_data_2015$equity
mean_ROE_by_industry <- aggregate(financial_data_2015$ROE,
        by = list(financial_data_2015$industry),
        FUN = mean)
```

となりますので、上の方が読みやすいことがわかります。

4.6 4.4.2 ヒストグラムによる売上高の可視化

4.6.1 ヒストグラム

ヒストグラム (histogram) は，データの分布を可視化するためのグラフです。ヒストグラムは，連続データを区間に分けて，区間ごとのデータの個数を棒グラフで表現したものです。したがってヒストグラムの棒の高さは、その区間に含まれるデータの個数を表します。

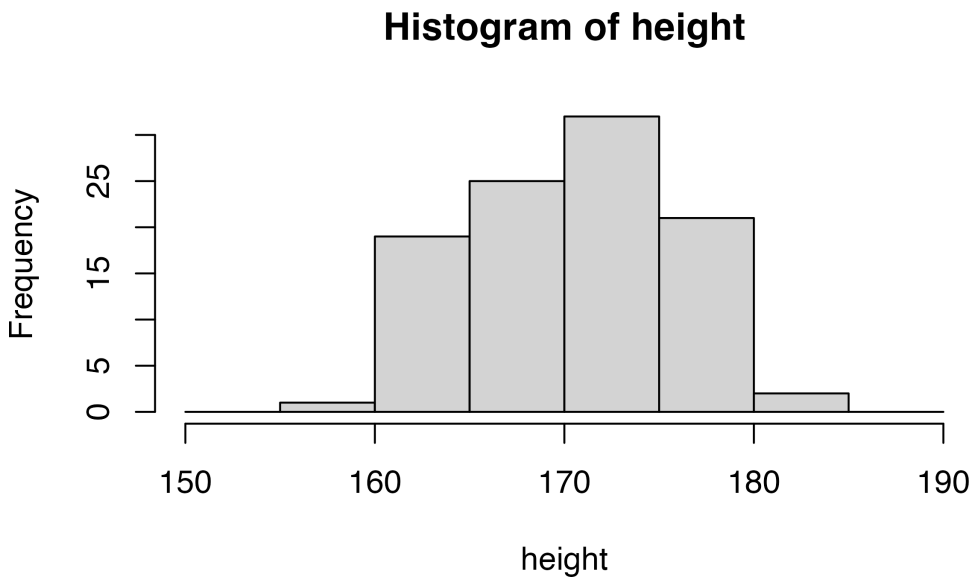
たとえば、例として生徒 100 人の身長データがあるとします。この身長データを R で生成するには，rnorm() 関数を使います。

```
# 平均 170cm，標準偏差 5cm の正規分布から 100 個のデータを生成
height <- rnorm(100, mean = 170, sd = 5)
print(height)
```

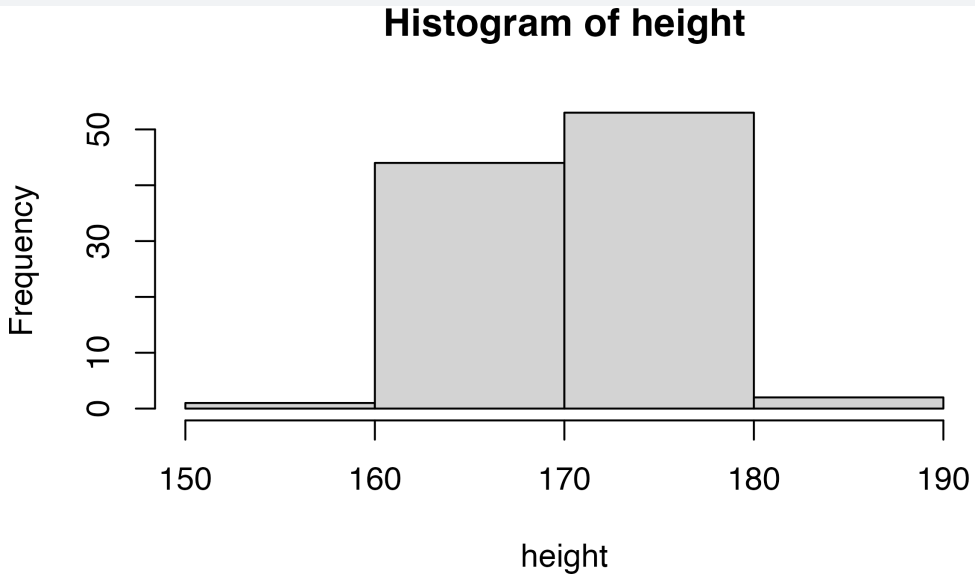
```
  [1] 164.9205 160.6358 177.1460 168.9079 164.9744 176.7954 161.9971 170.7038
  [9] 176.8454 182.0554 169.5937 175.8127 168.7761 181.7835 176.6639 173.4106
 [17] 170.5923 167.6094 167.9451 171.8052 170.7203 179.2915 163.3668 165.8677
 [25] 163.3148 164.1083 167.3397 173.7656 163.2541 176.5005 169.9076 160.1589
 [33] 177.4194 172.2858 164.7704 162.4569 171.2845 176.1653 167.8863 175.7839
 [41] 174.3949 169.1427 173.0604 175.3446 174.4560 168.2626 171.2081 171.2860
 [49] 166.0035 171.2585 178.2567 164.0956 172.6888 166.7440 169.5286 175.0822
 [57] 170.8167 174.3639 176.2128 165.5065 172.6126 170.2368 176.8795 172.2746
 [65] 170.0341 172.0154 163.1547 165.9379 173.2920 164.5706 172.7332 169.1814
 [73] 171.6336 168.9631 165.6040 162.3683 169.0026 177.1117 169.1371 159.5034
 [81] 163.1695 166.8916 176.1712 176.8173 171.9065 163.4690 171.7840 170.2291
 [89] 175.1851 171.4083 168.1763 164.0248 171.5008 171.8026 166.6785 162.2488
 [97] 169.1161 177.4755 175.8959 172.2913
```

100 個のデータを眺めていても、なかなか特徴をつかめませんよね。そこでこの身長という連続データを 5 センチごとの区間に分けます。たとえば、165cm 以上、170cm 未満の区間には何人の生徒がいるのか、170cm 以上、175cm 未満の区間には何人の生徒がいるのか、というように区間ごとのデータの個数を数えます。このとき、区間の幅を 5cm にするか、10cm にするか、20cm にするか、ということは、データの特徴をつかむ上で重要なことです。区間の幅を大きくすると、データの特徴がざっくりとしかつかめません。一方、区間の幅を小さくすると、データの特徴が細かくつかめますが、データの個数が少ない区間が多くなり、データの特徴をつかむのに時間がかかります。やってみましょう。

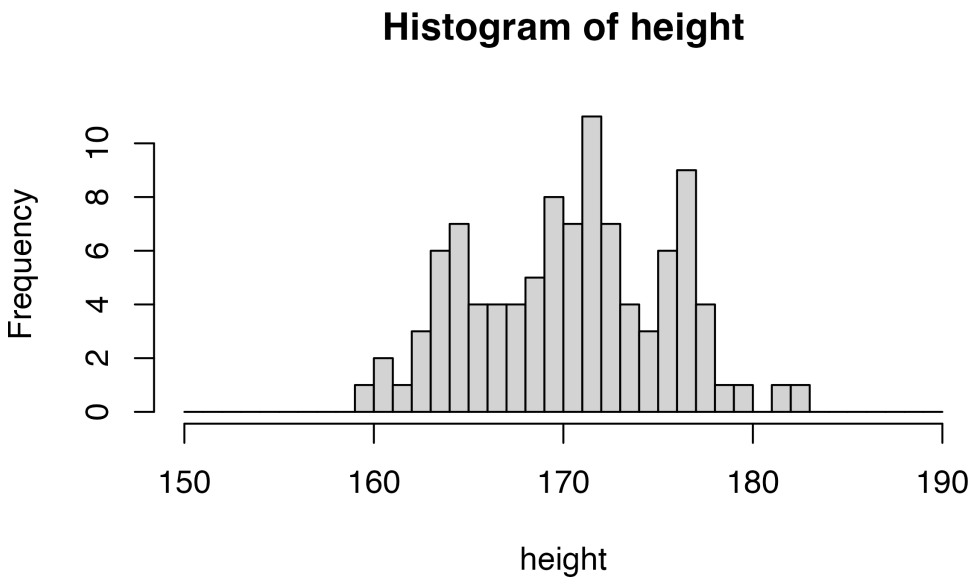
```
hist(height, breaks=seq(150,190,by=5)) # 区間の幅を 5cm にする
```




```
hist(height, breaks=seq(150,190,by=10)) # 区間の幅を 10cm にする
```




```
hist(height, breaks=seq(150,190,by=1)) # 区間の幅を 1cm にする
```



どのヒストグラムがデータの特徴を最も良く表しているのか、を考えて区間幅を設定しましょう。

4.6.2　ggplot でヒストグラム

ヒストグラムを書くためには，基本関数の `hist()` が最も簡単ですが，より高性能な `ggplot2` を用いたヒストグラムの書き方を説明します。ここでは、上で作成した 2015 年のデータ `financial_data_2015` を使って、売上高のヒストグラムを書きます。

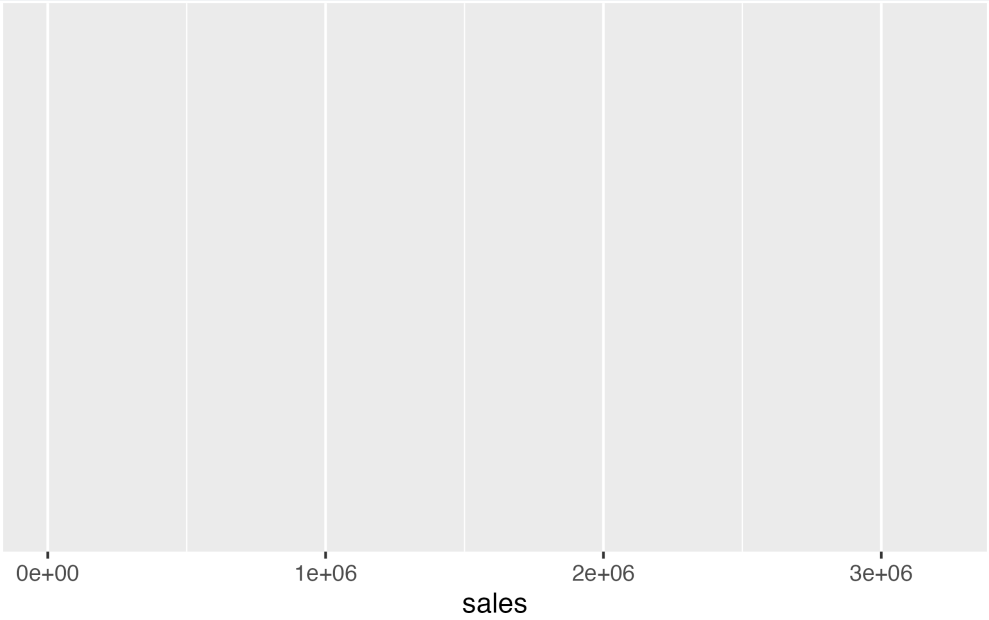
`ggplot2` の書き方は少し特殊ですが、慣れてくると非常に便利です。`ggplot2` ではレイヤー (階層) を上から重ねていくようにグラフを作っていきます。まず `ggplot()` 関数でグラフの土台を作ります。`ggplot()` に入れるデータの型は `data.frame` でなければならないので注意しましょう。

```
g <- ggplot(data = financial_data_2015)
print(g)
```

真っ白で何も出力されていませんが、financial_data_2015 というデータフレームを指定して、グラフの土台を作りました。

次に軸の設定をします。ヒストグラムは 1 変数のグラフなので x 軸のみを設定します。aes() 関数で変数を指定します。

```
g <- g + aes(x = sales)
print(g)
```

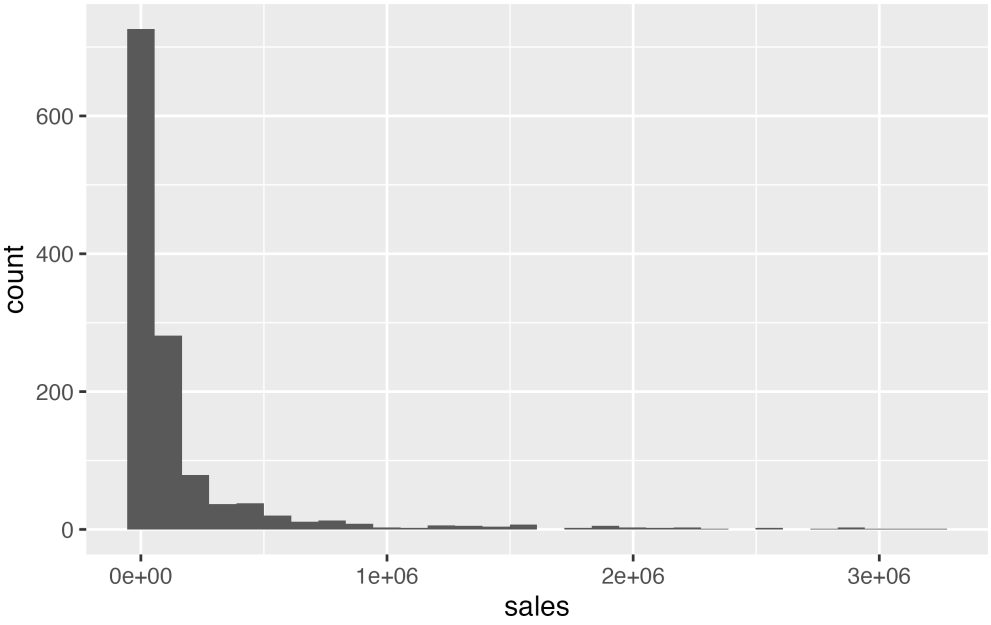


横軸が表示されました。この上に、ヒストグラムを書くために geom_histogram() 関数を追加します。ggplot2 パッケージでは、geom_***の形でグラフを指定します。例えば、

- geom_bar 棒グラフ
- geom_point 散布図
- geom_line 折れ線グラフ
- geom_boxplot 箱ひげ図
- geom_histogram ヒストグラム

あたりがよく使われるグラフです。

```
g <- g + geom_histogram() # グラフはヒストグラム
print(g)
```



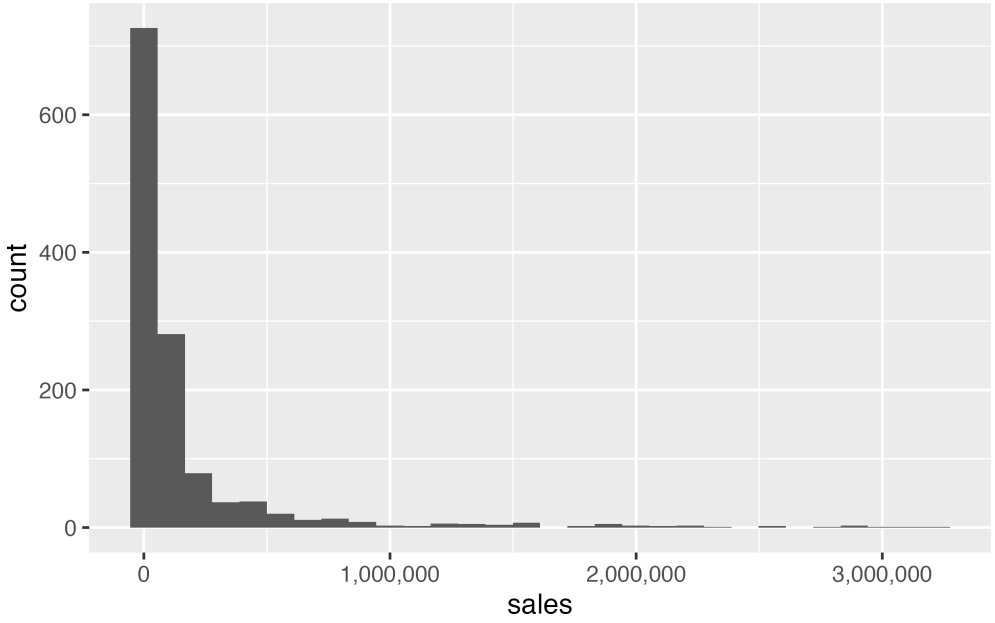
ここでコンソールに，

```
stat_bin() using bins = 30. Pick better value with binwidth.
```

というメッセージが出ますが、これは「何も指定されなかったので，ヒストグラムのビンの数を 30 にして作図したけど，オプションの stat_bin() で適切な区間幅を binwidth で設定してね」ということです。無視しても大丈夫です。

x 軸が指数表記となっていて見づらいので，scales() 関数を使って表記を変更します。

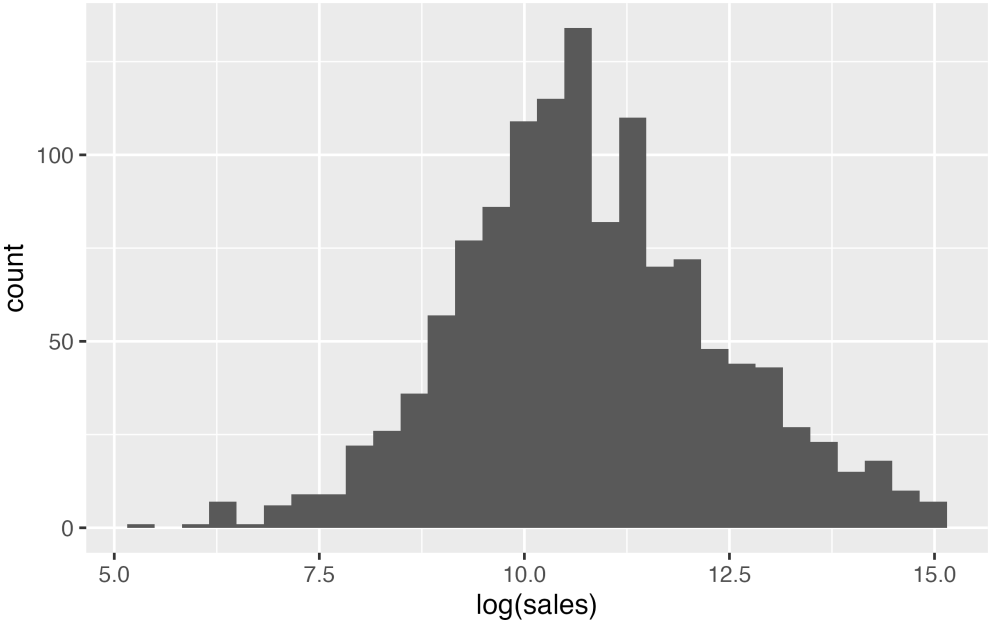
```
g <- g + scale_x_continuous(label = scales::label_comma()) # 3 桁ごとに
↪  コンマで区切った数値で表示
print(g)
```



横軸の数値が変化したことが分かります。

また、小数ながら非常に大きな売上高をもつ企業があるため，ヒストグラムの形が左側に集まるように歪んでいます。そこで売上高を自然対数に変換して，分布の歪みを修整したヒストグラムを書いてみます。データを変更するので、最初から全部書きます。

```
g <- ggplot(financial_data_2015) +
  aes(x = log(sales)) + # log() で自然対数変換
  geom_histogram()
print(g)
```



うまくいきました。

4.7　データの集計と折れ線グラフによる可視化

4.7.1　dplyr を用いた集計

もっとデータを加工して、データの特徴をつかむグラフを作成してみます。データを加工するために、非常に便利なパッケージである tidyverse の dplyr を用いたデータ加工を説明します。dplyr でよく使う関数に、

- summarise()
- mutate()
- filter()

があります。これらの関数を組み合わせることで、データの加工が非常に簡単にできます。特定の変数ごとにデータをグループ化するには，mutate() 関数や summarise() 関数の中で、.by = c(vars) という引数を用います。

たとえば、financial_data に含まれる売上高を年度ごとに集計してみましょう。

以前は、dplyr パッケージの group_by() 関数を用いてグループ化を行っていましたが、複数の変数でグループ化を行ったとき、ungroup() 関数でグループ化を解除する必要があり、また全てのグループ化を解除するにも手間でした。dplyr の新しいバージョンでは、group_by を用いずに、summarise() や mutate() 関数の中で .by 引数を用いることで、グループ化と集計を一度に行うことができ、また処理後はグループ化が完全に解除されるので、コードがシンプルになります。

```
N_firms_by_year <- financial_data |>
    summarize( # 以下の統計量を計算
        N_firms = n(), # データ個数 n()
        mean_sales = mean(sales), # 売上高の年度平均 mean()
        .by = year # year 変数ごとにグループ化
    )
```

これで N_firms_by_year というオブジェクトに、financial_data を年度ごとにグループ化して、年度ごとの企業数 N_firms と平均売上高 mean_sale を計算したデータが

入っています。2015 年から 2020 年の 6 年間のデータがあるので、6 行のデータが入っているはずです。中身を確認しておきましょう。

```
glimpse(N_firms_by_year)
```

```
Rows: 6
Columns: 3
$ year       <fct> 2016, 2017, 2018, 2019, 2020, 2015
$ N_firms    <int> 1293, 1319, 1323, 1356, 1363, 1265
$ mean_sales <dbl> 173359.5, 170010.9, 157995.4, 160928.2, 161043.7, 173614.9
```

以下の変数について 6 個のデータが入っていることがわかります。

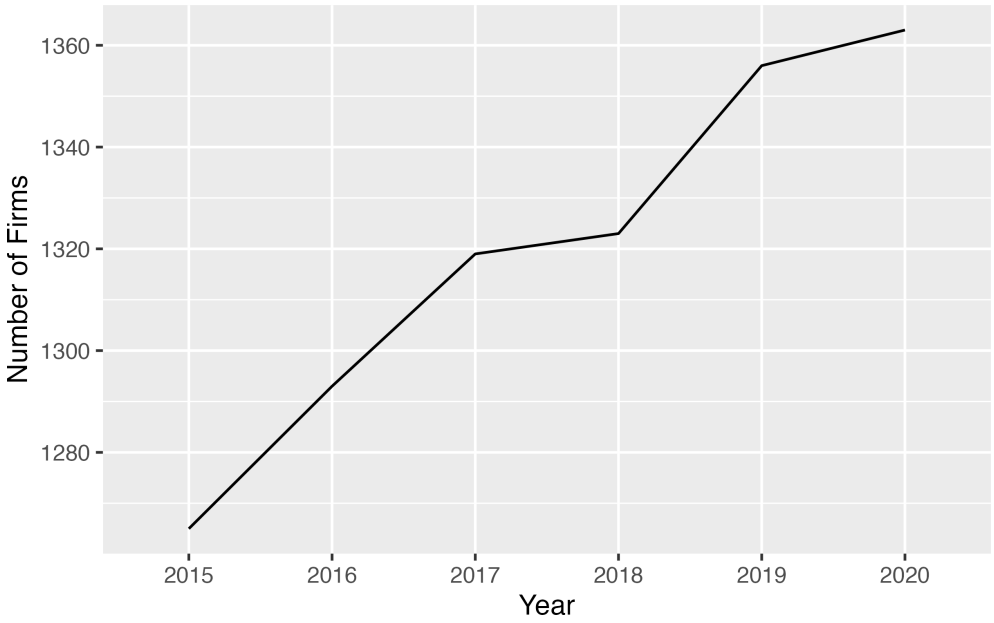
- year：年度 (ord)
- N_firms：企業数 (int)
- mean_sales：平均売上高 (dbl)

> **Note**
>
> 関数型プログラミング (functional programming) は，現代的なプログラミング・パラダイムの 1 種であり，定義された関数を用いて各データに対して行いたい処理を切り分ける。R では apply 系関数として，様々な関数が用意されている。tidyverse 群では，purrr がある。ちょっと難しいですが purrr 超便利

4.7.2　折れ線グラフによる上場企業数の可視化

データの成形が終わったので，折れ線グラフを作っていきます。ここでは x 軸 (横軸) を年度 year，y 軸 (縦軸) を上場企業数 N_firms とする折れ線グラフを作ってみます。折れ線グラフを作るには geom_line() 関数を使います。

```
g <- ggplot(N_firms_by_year) +
    aes(x = year, y = N_firms, group=1) +
    geom_line()
g <- g + labs(x = "Year", y = "Number of Firms") # 軸ラベル
print(g)
```



ここで突然現れた `group = 1` という `aes()` のオプションですが、これはすべてのデータが同じグループに属していることを指定しています。x 軸にファクター型を指定する場合、`group = 1` を指定しないと、x 軸の値ごとに別のグループとして認識されてしまい、折れ線グラフがうまく描けません。

4.8　変数の作成とヒストグラムによる可視化

`tidyverse` の `dplyr` パッケージの `mutate()` 関数を用いれば，パイプ演算子`|>`を用いて可読性の高いシンプルな書き方で、新しい変数を作成することができます。

4.9　キーボードショートカット

Mac なら `command + shift + m` でパイプ演算子が入力できます。Windows なら `ctrl + shift + m` です。

ここでは，ROE(Return on Equity) を計算してみます。ROE の定義は，

$$ROE_t = \frac{X_t}{BE_{t-1}}$$

となります。分子の X_t は t 期の当期純利益，分母の BE_{t-1} は t 期首の株主資本です。練習用データである `financial_date.csv` には，当期純利益は `X` という列名で収録されていますが、株主資本の列はありません。よってデータから株主資本は次のように計算します。

$$BE_t = \underbrace{(OA_t - OL_t)}_{NOA_t} - \underbrace{(FO_t - FA_t)}_{NFO_t}$$

この計算を行い，新しい変数 BE をデータフレームに加えるには，dplyr::mutate() を使います。

```
financial_data <- financial_data |>
    mutate(
        BE = (OA - OL) - (FO - FA) # 新たな BE 変数が加わる
    )
```

分母の株主資本は期首，つまり前期末の数値を用いる必要があります。1 期前の値を参照するには，lab() 関数を用います。ただ，クロスセクションのデータで普通に lag() 関数を用いると，次のように別の企業のデータを参照してしまいます。

```
financial_data <- financial_data |>
    mutate(
        lag_BE = lag(BE),
        ROE = X / lag_BE # これはダメ
    )

head(financial_data, 10)[,c("firm_ID", "year", "BE","lag_BE","ROE")]
```

```
# A tibble: 10 x 5
   firm_ID year      BE lag_BE      ROE
   <fct>   <fct>  <dbl>  <dbl>    <dbl>
 1 1       2016  10014.     NA  NA
 2 1       2017  10426. 10014.  0.0661
 3 1       2018  10842. 10426.  0.0638
 4 1       2019  11075. 10842.  0.0609
 5 1       2020  11594. 11075.  0.0762
 6 2       2015   1055. 11594.  0.00346
 7 2       2016   1082.  1055.  0.0468
 8 2       2017   1135.  1082.  0.0702
 9 2       2018   1184.  1135.  0.0763
10 2       2019   1237.  1184.  0.0770
```

結果の firm_ID が 2 の企業の 2015 年の ROE を計算するには，1 期前の企業 1 の 2014 年の株主資本を参照する必要があるけれど，2014 年のデータは存在しないため、企業 2 の 2015 年度の ROE は欠損値 NA になっている必要があるのに、lag() 関数が 1 つ前の企業 1 の 2020 年の株主資本を参照してしまっています。ROE を企業ごとに計算するために，mutate() 関数の引数として、.by = firm_ID を指定します。

```
financial_data <- financial_data |>
    mutate(
```

```
        lagged_BE = lag(BE), # lag 関数で前期の値を取り出す
        ROE = X / lagged_BE, # これは OK
        .by = firm_ID # firm_ID ごとにグループ化
    )
head(financial_data, 10)[,c("firm_ID", "year",
 ↪  "BE","lagged_BE","ROE")]
```

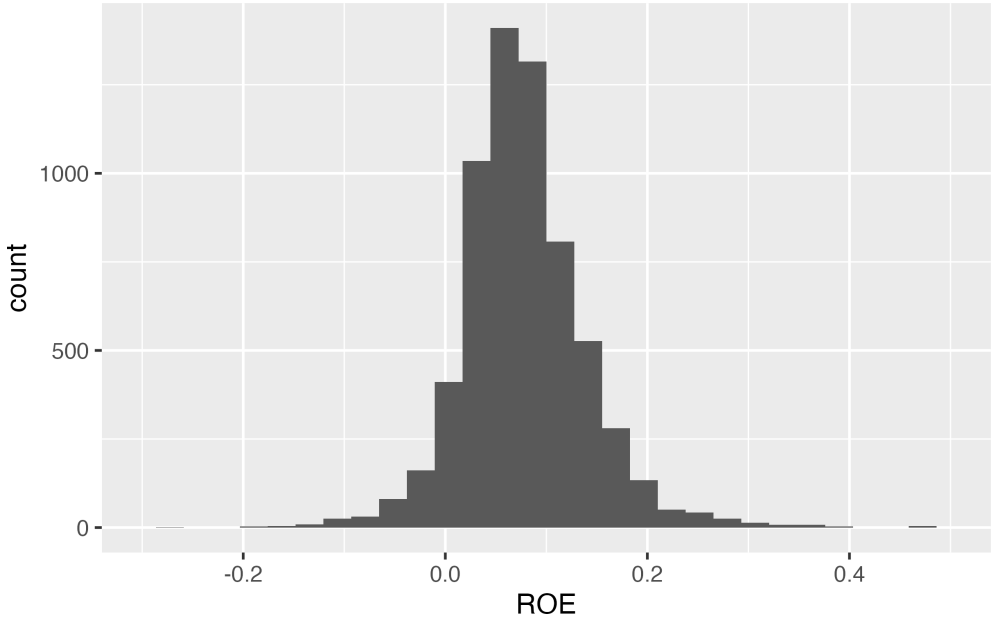
```
# A tibble: 10 x 5
   firm_ID year      BE lagged_BE     ROE
   <fct>   <fct>  <dbl>     <dbl>   <dbl>
 1 1       2016  10014.       NA  NA
 2 1       2017  10426.    10014.  0.0661
 3 1       2018  10842.    10426.  0.0638
 4 1       2019  11075.    10842.  0.0609
 5 1       2020  11594.    11075.  0.0762
 6 2       2015   1055.       NA  NA
 7 2       2016   1082.     1055.  0.0468
 8 2       2017   1135.     1082.  0.0702
 9 2       2018   1184.     1135.  0.0763
10 2       2019   1237.     1184.  0.0770
```

2015 年の ROE が欠損値になっており、正しい計算ができています。クロスセクション分析における `lag()` 関数の問題点を分かりやすくするために、上のように `lagged_BE` 変数と `ROE` 変数を別々に作成しましたが、通常は次のように書きます。

```
financial_data <- financial_data |>
    mutate(
        ROE = X / lag(BE), # これで一気に計算する方が OK
        .by = firm_ID # firm_ID ごとにグループ化
    )
```

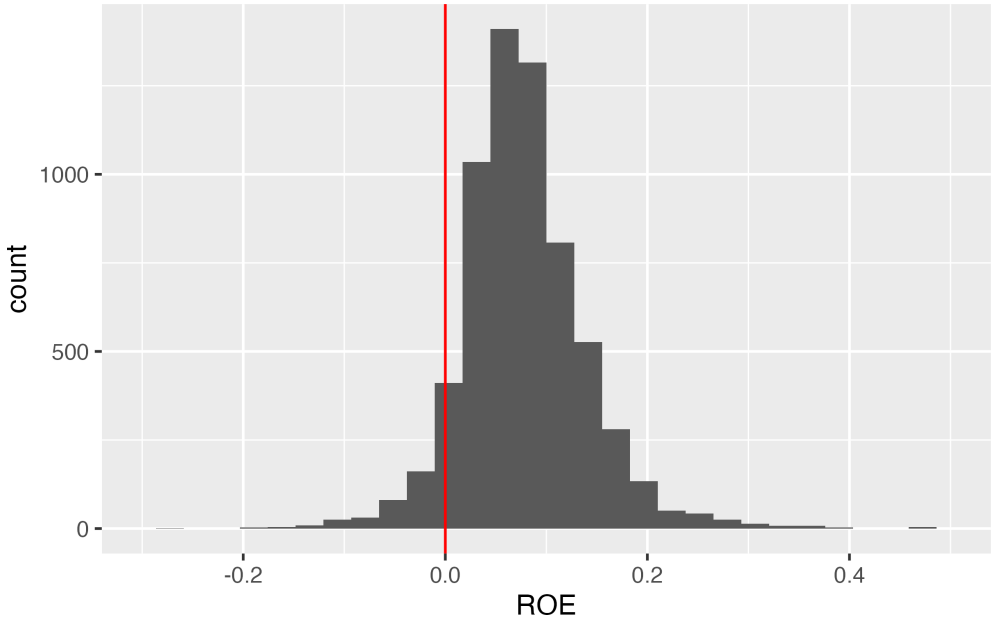
これで ROE の計算ができので、次に ROE のヒストグラムを作ってみます。

```
g <- ggplot(financial_data) + # データの選択
    aes(x = ROE) + geom_histogram()
g <- g + scale_x_continuous(limits = c(-0.3, 0.5)) # x 軸の範囲を調整
print(g)
```



ROE の分布が分かりました。赤字企業が分かりやすいように、ROE がゼロのところに縦線を引いてみます。縦線を引くには geom_vline() を使い、横線を引くには geom_hline() を使います。

```
g <- g + geom_vline(xintercept = 0, color = "red") # x 軸に縦線を引く
print(g)
```



4.10　グループごとの集計とランク付け

4.10.1　産業ごとの ROE 平均値と棒グラフによる可視化

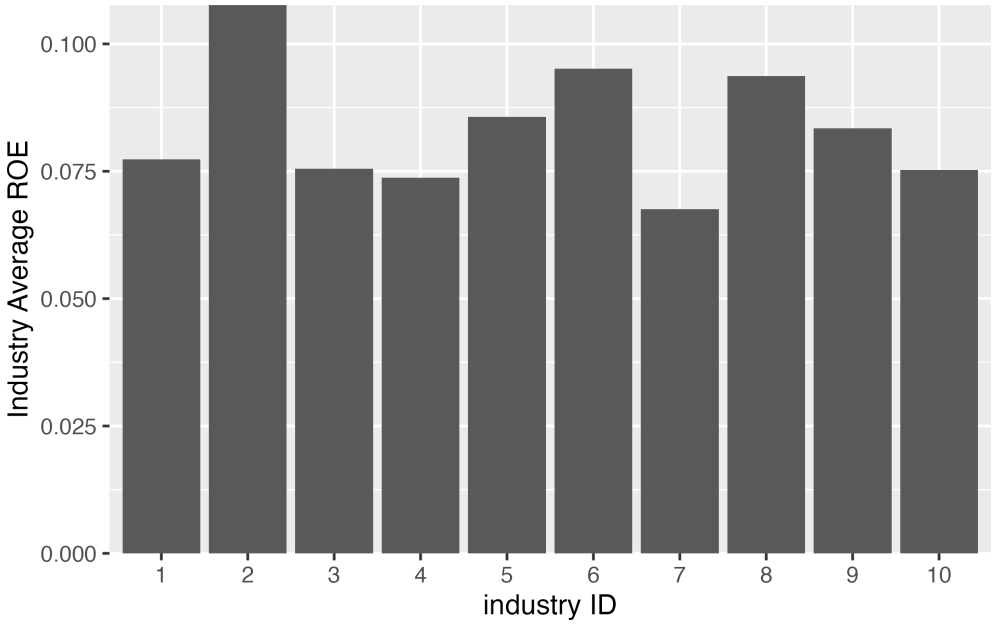
グループごとに平均値を出すといった処理は，dplyr::summarise() の引数で .by=var を指定すれば簡単です。ここでは、産業ごとに ROE の平均値と標準偏差を求めてみます。ROE には欠損値が含まれているため、mean() 関数を使うと NA が返ってきます。NA を無視して平均値を計算するには、mean() 関数のオプション na.rm = TRUE を指定します。

```
df_ind <- financial_data |>
    summarize(
        mean_ROE = mean(ROE, na.rm = TRUE), # 産業平均
        sd_ROE = sd(ROE, na.rm = TRUE),  # 産業標準偏差
        .by = industry_ID # 産業ごとにグループ化
    )
glimpse(df_ind)
```

```
Rows: 10
Columns: 3
$ industry_ID <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
$ mean_ROE    <dbl> 0.07731106, 0.10761577, 0.07548896, 0.07371925, 0.08570296~
$ sd_ROE      <dbl> 0.09264764, 0.09530125, 0.05569899, 0.04676826, 0.04859797~
```

10 の産業ごとに統計量を計算したので、10 行のデータが返ってきました。このデータを用いて産業ごとの ROE 平均の棒グラフを作成してみます。

```
# 作図
ggplot(df_ind) + # データフレームを指定
    aes(x = industry_ID, y = mean_ROE) + # 変数を 2 つ指定
    geom_col() + # 棒グラフ geom_bar() もあるけどこっち
    labs(x = "industry ID", y = "Industry Average ROE") + # ラベル設定
    scale_y_continuous(expand = c(0,0)) # グラフの原点 0,0 に設定
```



教科書では，パイプ処理で直接 ggplot() にデータフレームを渡していますが，個人的に可読性が低くなりオススメできないので，上の例ではデータ操作と作図を分けて書きました。

次に、2020 年度の産業別 ROE ランキングを作ってみます。ROE を大きい順にならべて、一番大きい企業に 1、2 番目の企業に 2、という風にランキングを表す変数を作成するには、`rank(desc())` を使います。`desc()` は降順に並べ替える関数です。

下のソースコードでは、前半のまとまりで、以下の処理を行ったデータを新しいデータフレーム `ROE_rank_data` に代入しています。

1. `financial_data` の中から 2020 年度のデータを抽出し、
2. 必要な変数として `firm_ID`、`industry_ID`、`ROE` の 3 つを選択し、
3. `industry_ID` ごとにグループ化して、
4. `mutate()` 関数で `ROE_rank` 変数を作成し，
5. `ungroup()` 関数でグループ化を解除しています。

後半のまとまりでは、上で作成した `ROE_rank_data` に対して、

1. 産業ごとの ROE ランキング第 1 位の企業を抽出し、
2. ROE が大きい順に並べ替えて、
3. それを `knitr::kable()` 関数で表として出力

という処理をしています。

```
# 2020 年度の産業内の ROE ランキングの変数を作成
ROE_rank_data <- financial_data |>
    filter(year == 2020) |>
    select(firm_ID, industry_ID, ROE) |>
    mutate(  # summarize を mutate に変更
```

```
        ROE_rank = rank(desc(ROE)),
        .by = industry_ID
    )

ROE_rank_data |>
    filter(ROE_rank == 1) |> # 各産業のランク 1 のものを抽出
    arrange(desc(ROE)) |>  # ROE が大きい順
    knitr::kable(booktabs = TRUE, # ここから下は表の装飾
        caption = "2020 年度産業別 ROE ランキング第 1 位企業",
        position = "h!" # 表示場所はここに
        )
```

Table4.5: 2020 年度産業別 ROE ランキング第 1 位企業

firm_ID	industry_ID	ROE	ROE_rank
929	7	0.5641813	1
475	3	0.4975356	1
8	1	0.3882552	1
242	2	0.3749986	1
661	5	0.2673141	1
1042	8	0.2559963	1
1380	10	0.2497929	1
1167	9	0.2346232	1
619	4	0.1491307	1
719	6	0.1422026	1

4.11　上級デュポン・モデルによる ROE の分析とその可視化

上級デュポン・モデルとは，次式で表される ROE の分解式です。

$$ROE_t := \frac{X_t}{BE_{t-1}} = \underbrace{\frac{OX_t}{NOA_{t-1}}}_{RNOA_t} + \underbrace{\frac{NFO_{t-1}}{BE_{t-1}}}_{FLEV_{t-1}} \times \left[\frac{OX_t}{NOA_{t-1}} - \frac{NFE_t}{NFO_{t-1}} \right]$$

各変数の意味は以下の通りです。- `X`：当期純利益 (net income) - `BE`：株主資本 (book of equity) - `OX`：事業利益 (operating income) - `NOA`：純事業資産 (net operating assets) - `NFO`：純金融負債 (net financial obligations) - `NFE`：純金融費用 (net financial expenses)

$RNOA_t$ は，ATO_t と PM_t とに分割できます。

$$\frac{OX_t}{\underbrace{NOA_{t-1}}_{RNOA_t}} = \frac{sales_t}{\underbrace{NOA_{t-1}}_{ATO_t}} \times \frac{OX_t}{\underbrace{sales_t}_{PM_t}}$$

いくつかの変数は，元のデータには含まれていないので，与えられたデータから計算する必要があります。`dplyr::mutate()` 関数を用いて新しい変数を作成し，データフレームに追加します。

```
financial_data <- financial_data |>
    mutate(
        NOA = OA - OL, # 純事業資産 = 事業資産 - 事業負債
        RNOA = OX / lag(NOA), # 会計上の事業リターン
        PM = OX / sales, # 利ざや profit margin
        ATO = sales / lag(NOA), # 純事業資産回転率
        NFO = FO - FA, # 純金融負債 = 金融負債 - 金融資産
        lagged_FLEV = lag(NFO) / lagged_BE, #期首財務レバレッジ
        NBC = NFE / lag(NFO), # 債権者のリターン net borrowing cost
        ROE_DuPont = RNOA + lagged_FLEV * (RNOA - NBC), # 上級デュポン・
        ↪ モデルによる ROE
        .by = firm_ID # 企業ごとにグループ化
    )
```

ROE の分解式が合っているかどうかを確認するため，`all.equal()` 関数を使って，第 1 引数と第 2 引数が等しいかどうかを判定してみます。普通に計算した `ROE` と上級デュポン・モデルの分解したものから計算した `ROE_DuPoint` との比較しています。

```
all.equal(financial_data$ROE, financial_data$ROE_DuPont)
```

```
[1] "Mean relative difference: 4.396878e-06"
```

となり，差の平均は 4.396878×10^{-6} となり，ほぼ 0 となっていることから，上級デュポン・モデルの分解式が正しいことが確認できます。完全にゼロにならない理由は計算の過程で生じる丸め誤差によるものです。

4.11.1 箱ひげ図による産業別比較

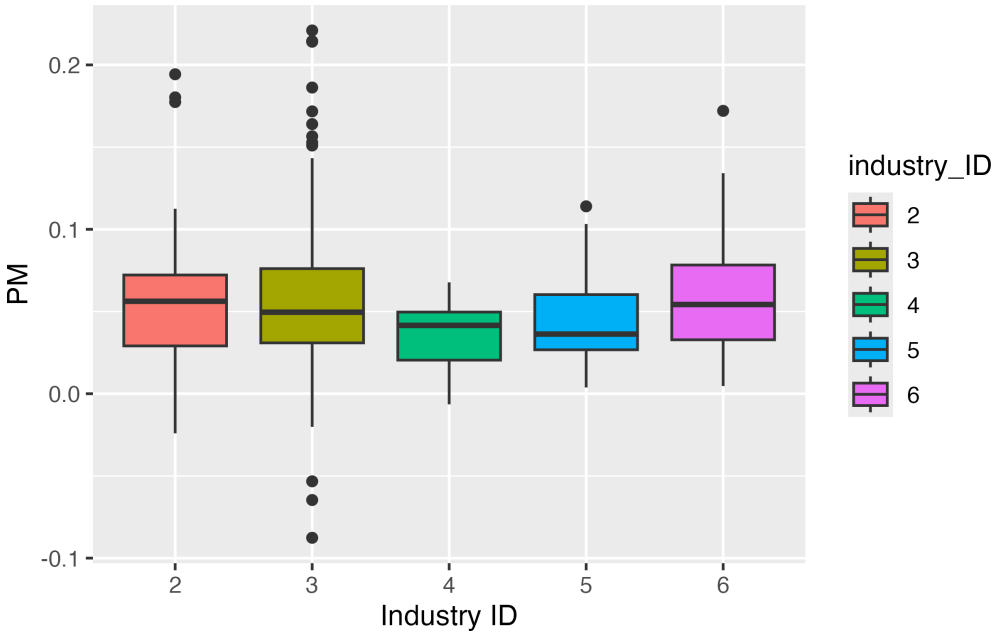
産業別で利ざや `PM` がどのように分布しているのかを調べるために，箱ひげ図 (box plot) を作ってみます。箱ひげ図は，データの分布を可視化するためのグラフで，第 1 四分位点，中央値，第 3 四分位点，(異常値をのぞく) 最大値，(異常値をのぞく) 最小値を表現できる，非常に情報量の多いグラフです。

`ggplot2` パッケージの `geom_boxplot()` 関数を用いることで，データフレームから箱ひげ図を作図できます。

先に作成したデータフレーム `financial_data` を用いて，`PM` の箱ひげ図を作成してみ

ましょう。あまり多くの箱ひげ図を作っても見づらくなるので，最終年度のデータで，産業 ID が 2〜6 までの企業に限定します。

```
df_2020 <- financial_data |>
    filter(
        year == 2020, # 最終年度
        industry_ID %in% 2:6 # 産業コードが 2 から 6
    )
g <- ggplot(df_2020) +
  aes(x = industry_ID, y = PM, fill = industry_ID) + geom_boxplot() #
   ↪ 箱ひげ図
g <- g + labs(x = "Industry ID")
print(g)
```



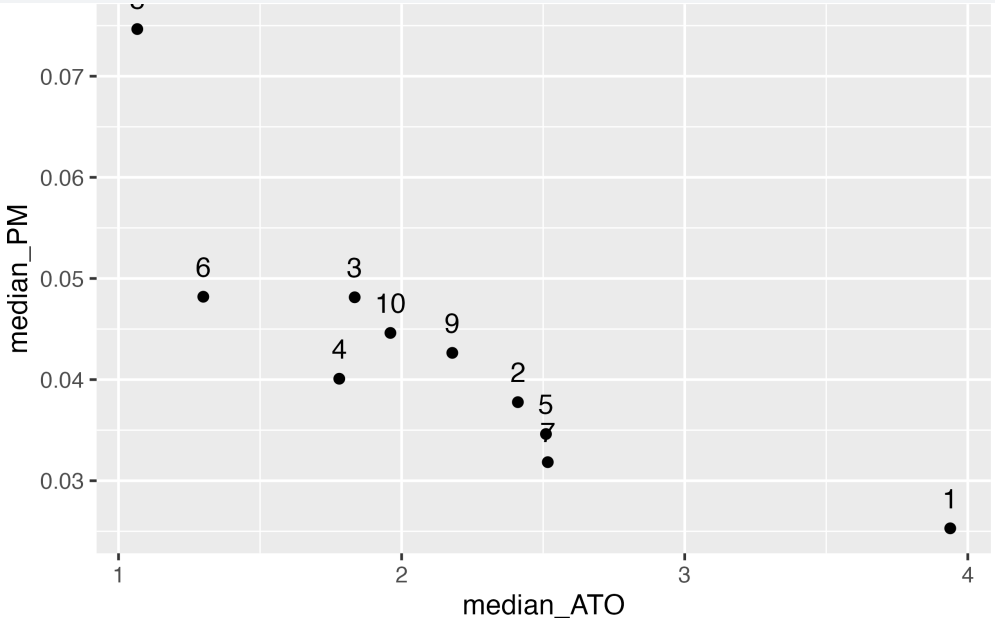
箱ひげ図から，産業ごとに利ざやの分布が異なることがわかります。とりわけ産業 3 は利ざやの散らばりが大きく，産業 4 は非常に散らばりが小さいことが分かります。

4.11.2　散布図による産業別比較

次に産業ごとに ATO(純事業資産回転率) と PM(売上高事業利益率) がどう分布しているか散布図を書いてみます。ここでは異常値の影響を受けにくい統計量である中央値 (median) を計算し，散布図を作成してみます。

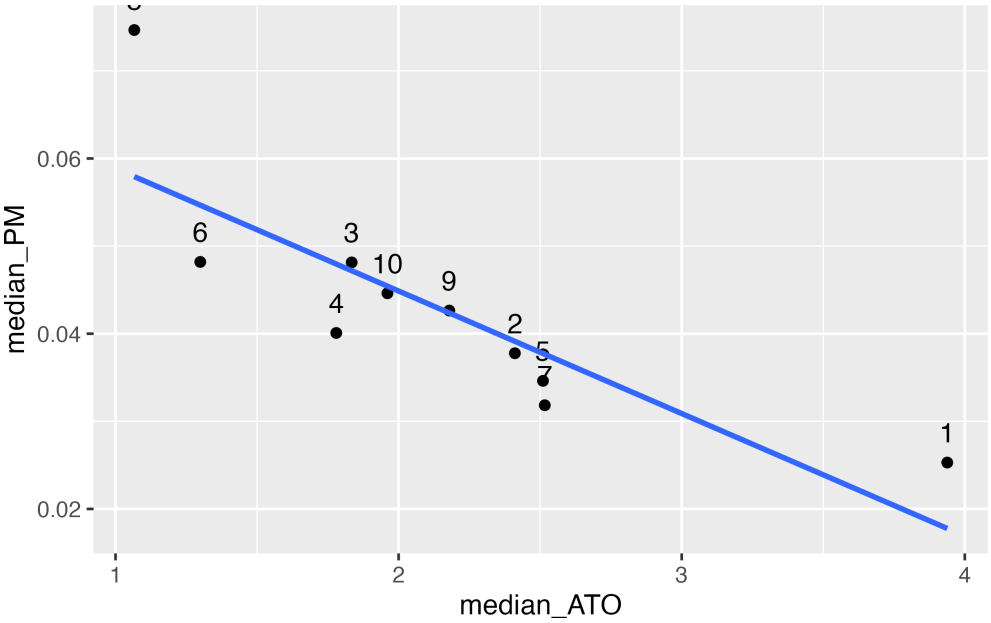
```
df_ind_median <- financial_data |>
    summarise(
```

```
        median_ATO = median(ATO, na.rm = TRUE), # ATO の中央値
        median_PM = median(PM, na.rm = TRUE), # PM の中央値
        .by = industry_ID # 産業ごとにグループ化
    )

g <- ggplot(df_ind_median) +
    aes(x = median_ATO, y = median_PM, label = industry_ID) + # 散布図
    geom_point() + # 散布図
    geom_text(vjust=-1) # ラベル
print(g)
```



この産業ごとに計算された中央値のデータを用いて，線形回帰直線を引いて，ATO と PM の関係を見てみます。

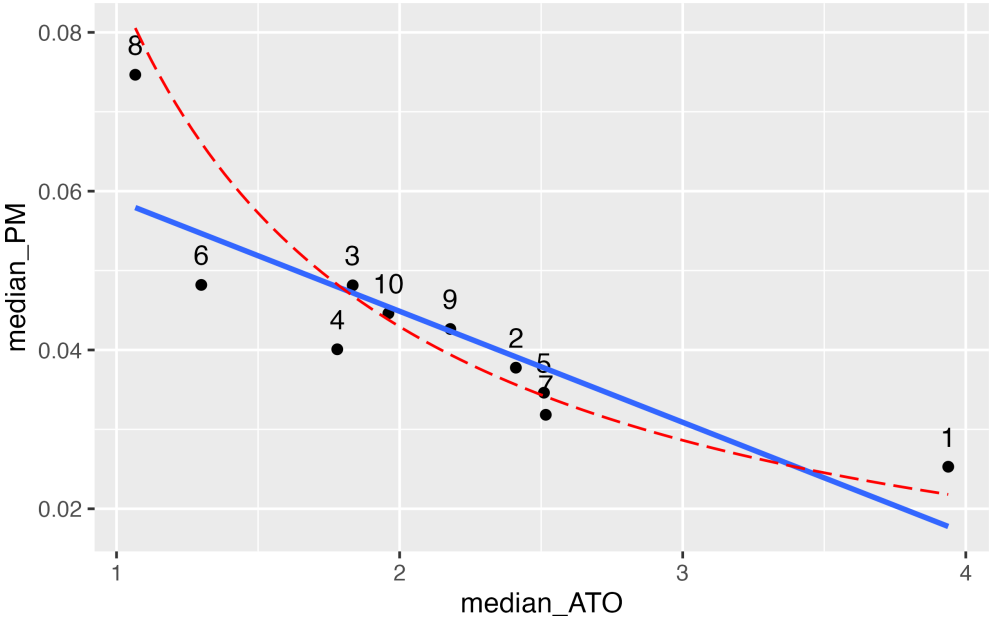
```
g <- g + geom_smooth(method = "lm", se = FALSE) # 線形回帰直線を追加
print(g)
```



いい感じですが，会計学入門 (1.3.4 節) で学習した $ATO \times RM = RNOA$ という関係のとおり，データからも ATO と PM との間にトレードオフの関係があることが予想されています。もし理論どおりの関係であればデータは $ATO = RNOA/PM$ といった反比例の関係になるはずです。これを示すため，RNOA を一定としたときの ATO と PM の関係，つまりを図に書き込んでみます。関数をグラフとして図に追加するために stat_function() 関数を用います。

stat_function() 関数の引数は，fun = function(x) x の関数系，linetype = “**スタイル**” とします。ここでは，function(x) で x の関数であることを指定し，median_RNOA / x としてます。

```
median_RNOA <- median(financial_data$RNOA, na.rm = TRUE) # 全データから
 ↪  計算した RNOA の中央値

g <- g + stat_function(
    fun = function(x) median_RNOA / x,
    linetype = "longdash",
    color = "red") # 反比例の関数を追加
print(g)
```



かなりあてはまりが良さそうな線が引けました。このように，データを可視化することで，理論とデータの整合性を確認することができます。

第 5 章

株式データの取得と可視化

Listing 5.1 パッケージの読み込みとテーマ

```
pacman::p_load(
  tidyverse, # 便利なパッケージ群
  ggthemes, # ggplot2 のテーマ集
  e1071, # 機械学習アルゴリズム
  broom, # モデルの整形
  scales # 軸のスケール調整
)

mystyle <- list(
  theme_economist_white(
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),
  theme(
    text = element_text(size = 10),
    axis.title = element_text(size = 10)
  )
)
```

前章では財務データを題材に、クロスセクションデータを扱うために必要なデータの取得と操作と可視化について学習しました。特に重要な操作として，`mutate()` や `summarise()` 関数で、`.by = var` を指定することでグループごとの変数の値を計算する方法を学びました。本章では株式データを題材に，株式データの理解に必要な株式の基礎知識とともに，リターンの定義や計算について学び，最後には統計的推論と線形回帰分析についても学習します。

5.0.1　株価データのダウンロードと読み込み

いままでと同じように、readr パッケージの read_csv() 関数で CSV ファイルを読み込みます。ここでは、ch05_stock_data.csv を使います。

Listing 5.2 株式データの読み込み

```
stock_data <- read_csv("data/ch05_stock_data.csv")
print(stock_data)
```

```
# A tibble: 95,040 x 9
    year month month_ID firm_ID stock_price   DPS shares_outstanding
   <dbl> <dbl>    <dbl>   <dbl>       <dbl> <dbl>              <dbl>
 1  2015     1        1       1         954     0            2422000
 2  2015     2        2       1         960     0            2422000
 3  2015     3        3       1        1113     0            2422000
 4  2015     4        4       1        1081     0            2422000
 5  2015     5        5       1        1317     0            2422000
 6  2015     6        6       1        1366    29            2422000
 7  2015     7        7       1        1353     0            2422000
 8  2015     8        8       1        1209     0            2422000
 9  2015     9        9       1        1291     0            2422000
10  2015    10       10       1        1407     0            2422000
# i 95,030 more rows
# i 2 more variables: adjustment_coefficient <dbl>, R_F <dbl>
```

9 変数，95,040 行という大規模なデータが読み込まれました。紙面の都合上、長い変数名を短く省略します。変数名を変えるには、dplyr::rename() を使います。

Listing 5.3 変数名の変更

```
stock_data <- stock_data |>
  rename(
    n_shares = shares_outstanding,
    adj_coef = adjustment_coefficient
  )
```

読み込まれたデータ stock_data の中には以下の変数が含まれています。

列名	データ型	説明
year	数値型 dbl	年度
month	数値型 dbl	月
month_ID	数値型 dbl	月 ID
firm_ID	数値型 dbl	企業 ID
stock_price	数値型 dbl	月末時点での終値
DPS	数値型 dbl	一株当たり配当額
n_shares	数値型 int	月末時点での発行済株式数
adj_coef	数値型 int	調整係数
R_F	数値型 dbl	月次無リスク金利

year〜firm_ID までは，時系列や企業の特定に必要なキーとなり，stock_price から R_F が分析する対象となる株価や配当，発行済株式数，調整係数，無リスク金利となります。7 列目の n_shares は発行済株式数です。発行済株式数は随時，新株発行や自社株買い，株式分割で変動するため，株式数の変化を調整するためのデータとして adj_coef があります。旧株式 1 に対して新株式 2 となる株式分割が行われた場合，この調整係数は 2 となります。

このような大規模データを前にした場合，とりあえずデータの構造を把握することから始めます。ここでは基本関数よりも多機能な tidyverse の dplyr パッケージに含まれる glimpse 関数を使って，データの構造を確認してみます。

```
glimpse(stock_data)
```

```
Rows: 95,040
Columns: 9
$ year        <dbl> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015~
$ month       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7~
$ month_ID    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
$ firm_ID     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ stock_price <dbl> 954, 960, 1113, 1081, 1317, 1366, 1353, 1209, 1291, 1407, ~
$ DPS         <dbl> 0, 0, 0, 0, 0, 29, 0, 0, 0, 0, 0, 29, 0, 0, 0, 0, 0, 40, 0~
$ n_shares    <dbl> 2422000, 2422000, 2422000, 2422000, 2422000, 2422000, 2422~
$ adj_coef    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ R_F         <dbl> 6.506826e-04, 5.834099e-04, 6.114423e-04, 6.848180e-
04, 7.~
```

データの型が dbl という初めて出てきた型になってますが，これは double の略で，数値型の一種です。double は 64 ビットの浮動小数点数を表し，小数点以下 15 桁までの精度を持ちます。

> **数値型の種類**
>
> 数値型 numeric は，double と int のどちらかになります。int は整数 (integer) 型で，32 ビットの整数を表し，小数点以下は切り捨てられます。dbl は int の上位互換で，小数点以下の桁数が多い場合に使われます。

たとえば，firm_ID が 74，month_ID が 29 から 32（つまり 2017 年 5 月から 8 月）のデータを抽出してみます。

Listing 5.4 データの抽出

```
stock_data |>
  # 条件を満たすデータを filter() で抽出
  filter(firm_ID == 74 , month_ID %in% 29:32) |>
  # 特定の変数を抽出
  select(
    year, firm_ID, month_ID,
    stock_price, n_shares, adj_coef
  )
```

```
# A tibble: 4 x 6
   year firm_ID month_ID stock_price n_shares adj_coef
  <dbl>   <dbl>    <dbl>       <dbl>    <dbl>    <dbl>
1  2017      74       29        2816  4960000        1
2  2017      74       30        1402  9920000        2
3  2017      74       31        1420  9920000        1
4  2017      74       32        1502  9920000        1
```

出力された結果の 2 行の adj_coef が 2 となっており、同時に n_shares が倍になっていることから、この会社は 1 株を 2 株に株式分割していることがわかります。

5.1　時価総額とリターンの計算

時価総額 (market capitalization) を計算するためには、株式数に株価を掛けて計算します。新しい変数を作成するときは dplyr パッケージの mutate() 関数を使います。mutate() で時価総額を表す新しい変数 ME を作成します。

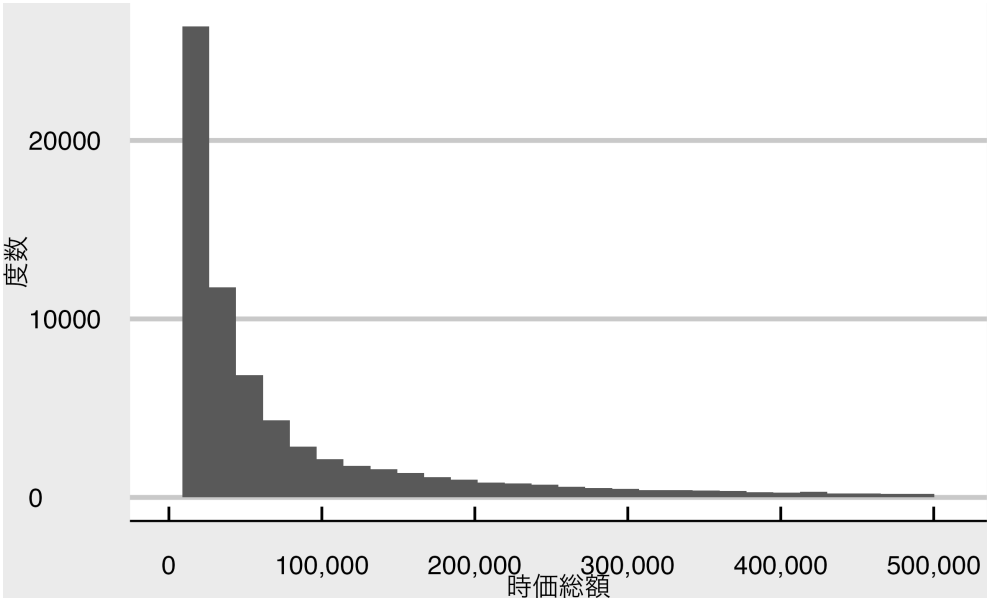
次に時価総額のヒストグラムを作成してみます。前節で学習した内容に加えて、いろいろ見た目の指定を増やしてみます。scale() 関数を使って軸の範囲や表記方法を細かく指定します。

Listing 5.5 時価総額の計算

```
stock_data <- stock_data |>
  mutate(
    ME = stock_price * n_shares # 時価総額
    )
```

Listing 5.6 時価総額のヒストグラム

```
# 日本語を含むグラフを作成するときは，dev = "quartz_pdf"を指定
g <- ggplot(stock_data) + aes(x = ME) +
  geom_histogram() + # ヒストグラムを描く
  scale_x_continuous( # 軸の範囲と表記方法の指定
    limits = c(0, quantile(stock_data$ME, 0.95)),
    labels = label_comma(scale = 1e-6)
  ) + xlab("時価総額") + ylab("度数") +  mystyle
print(g) # グラフを出力
```



5.1.1 トータル・リターンと超過リターンの計算

ある株式の t 期のトータル・リターン R_t は、次式で定義されます。

$$R_t = \frac{(\text{株価}_t + 1\text{ 株当り配当}_t) \times \text{調整係数}_t - \text{株価}_{t-1}}{\text{株価}_{t-1}}$$

たいていの場合，1 株当り配当 1 株当り配当$_t$ はゼロで，調整係数は 1 となるため，次

式のように簡略化できます。

$$R_t = \frac{\text{株価}_t - \text{株価}_{t-1}}{\text{株価}_{t-1}}$$

銘柄ごとにリターンを計算するので，mutate() と .by = firm_ID を使って計算します。ここでは，firm_ID ごとにトータルリターン R と，トータルリターンから無リスク利子率を除いた超過リターン Re を計算しています。

```
stock_data <- stock_data |>
  mutate( # 新しい変数を作成
    R = ((stock_price + DPS) * adj_coef - lag(stock_price)) /
     ↪ lag(stock_price),
    Re = R - R_F, # 月次超過リターン
    .by = firm_ID # firm_ID ごとにグループ
  )
```

ここまでの処理で stock_data に時価総額 ME，トータルリターン R，超過リターン Re が追加されました。以下では，このデータを使って探索的データ分析を行います。

5.1.2　株式データの探索的データ分析

探索的データ分析という名の，とりあえず何も考えずに目の前のデータから何が分かるのかを調べ倒してみる，という分析を行います。

とりあえず，summary() で基本統計量を確認します。

```
summary(stock_data)
```

```
      year          month          month_ID        firm_ID
 Min.   :2015   Min.   : 1.00   Min.   : 1.00   Min.   :   1.0
 1st Qu.:2016   1st Qu.: 3.75   1st Qu.:19.00   1st Qu.: 384.0
 Median :2018   Median : 6.50   Median :37.00   Median : 760.0
 Mean   :2018   Mean   : 6.50   Mean   :37.01   Mean   : 761.2
 3rd Qu.:2019   3rd Qu.: 9.25   3rd Qu.:55.00   3rd Qu.:1147.0
 Max.   :2020   Max.   :12.00   Max.   :72.00   Max.   :1515.0

  stock_price          DPS             n_shares           adj_coef
 Min.   :   112   Min.   :   0.000   Min.   :3.700e+04   Min.   :0.1000
 1st Qu.:  1417   1st Qu.:   0.000   1st Qu.:3.151e+06   1st Qu.:1.0000
 Median :  2445   Median :   0.000   Median :1.014e+07   Median :1.0000
 Mean   :  4685   Mean   :   6.802   Mean   :6.973e+07   Mean   :0.9999
 3rd Qu.:  4558   3rd Qu.:   0.000   3rd Qu.:3.564e+07   3rd Qu.:1.0000
```

```
 Max.   :622796   Max.   :1913.000   Max.   :2.111e+10   Max.   :2.0000

      R_F                   ME                   R                  Re
 Min.   :-2.329e-04   Min.   :1.459e+08   Min.   :-0.38028   Min.   :-
0.38020
 1st Qu.: 8.233e-06   1st Qu.:9.648e+09   1st Qu.:-0.04057   1st Qu.:-
0.04076
 Median : 8.203e-05   Median :2.563e+10   Median : 0.01033   Median : 0.01013
 Mean   : 2.041e-04   Mean   :1.365e+11   Mean   : 0.01586   Mean   : 0.01565
 3rd Qu.: 4.626e-04   3rd Qu.:7.987e+10   3rd Qu.: 0.06481   3rd Qu.: 0.06460
 Max.   : 7.368e-04   Max.   :3.293e+13   Max.   : 0.51498   Max.   : 0.51448
                                          NA's   :1515       NA's   :1515
```

さらに，分散と標準偏差も計算します。データには欠損値が含まれているため，`na.rm = TRUE` オプションをつけています。教科書とは違いますが，`dplyr::summarize()` 関数を使って一気に複数の統計量を計算します。ここでは，`stock_data <- stock_data` としていないため，`stock_data` には新しい変数は追加されず，結果を表示するだけです。

```
stock_data |>
  summarise(
    var_R = var(R, na.rm = TRUE), # 総リターンの分散
    sd_R  = sd(R, na.rm = TRUE), # 総リターンの標準偏差
    var_Re = var(Re, na.rm = TRUE), # 超過リターンの分散
    sd_Re = sd(Re, na.rm = TRUE) # 超過リターンの標準偏差
  )
```

```
# A tibble: 1 x 4
    var_R   sd_R  var_Re  sd_Re
    <dbl>  <dbl>   <dbl>  <dbl>
1 0.00830 0.0911 0.00830 0.0911
```

データの分布の形を表す統計量である，3 次のモーメントである歪度 (skewness) と 4 次のモーメントである尖度 (kurtosis) も計算してみます。たとえば確率変数 x の歪度は

$$\text{歪度} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\hat{\sigma}_x}\right)^3$$

で表され，正の値をとるとき分布が右に歪んでいる (裾が右に長い) ことを示している。尖度は，

$$\text{尖度} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\hat{\sigma}_x}\right)^4$$

で定義され，尖度が 3 のとき，正規分布と同じ尖度を持つことを示し，3 より小さいとき，正規分布よりとがった分布をしていることを示します。

歪度と尖度を計算するには，e1071 パッケージの skewness() 関数を使います。

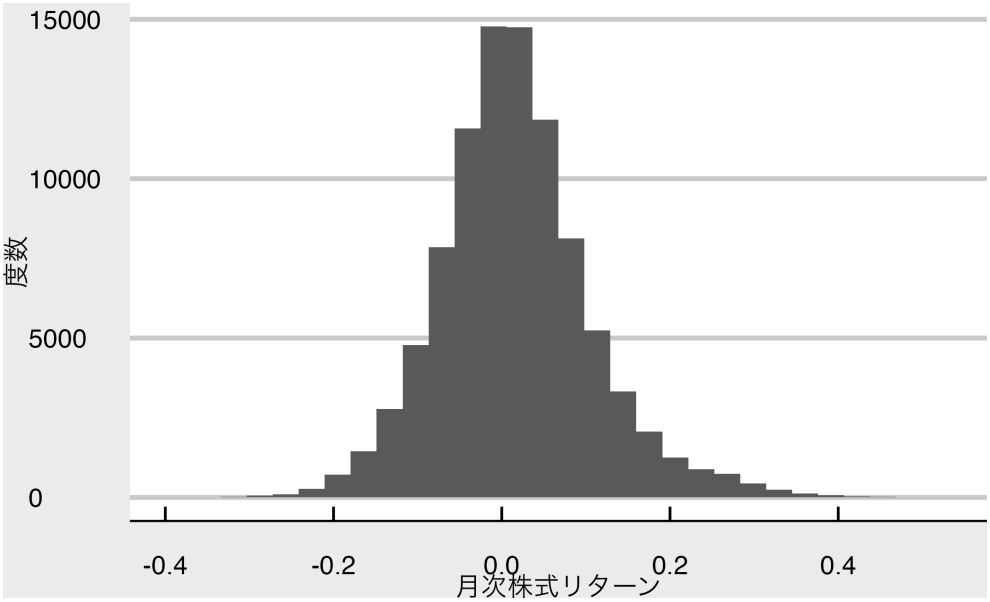
```
stock_data |>
  summarise(
    skewness_R = skewness(R, na.rm = TRUE), # 総リターンの歪度
    kurtosis_R = kurtosis(R, na.rm = TRUE), # 総リターンの尖度
    skewness_Re = skewness(Re, na.rm = TRUE), # 超過リターンの歪度
    kurtosis_Re = kurtosis(Re, na.rm = TRUE) # 超過リターンの尖度
  )
```

```
# A tibble: 1 x 4
  skewness_R kurtosis_R skewness_Re kurtosis_Re
       <dbl>      <dbl>       <dbl>       <dbl>
1      0.507       1.27       0.507        1.27
```

トータルリターンの歪度が $0.507 > 0$ なので，右に裾の広い分布となっており，尖度が $1.27 < 3$ なので正規分布よりもとがった形となっていることがわかります。

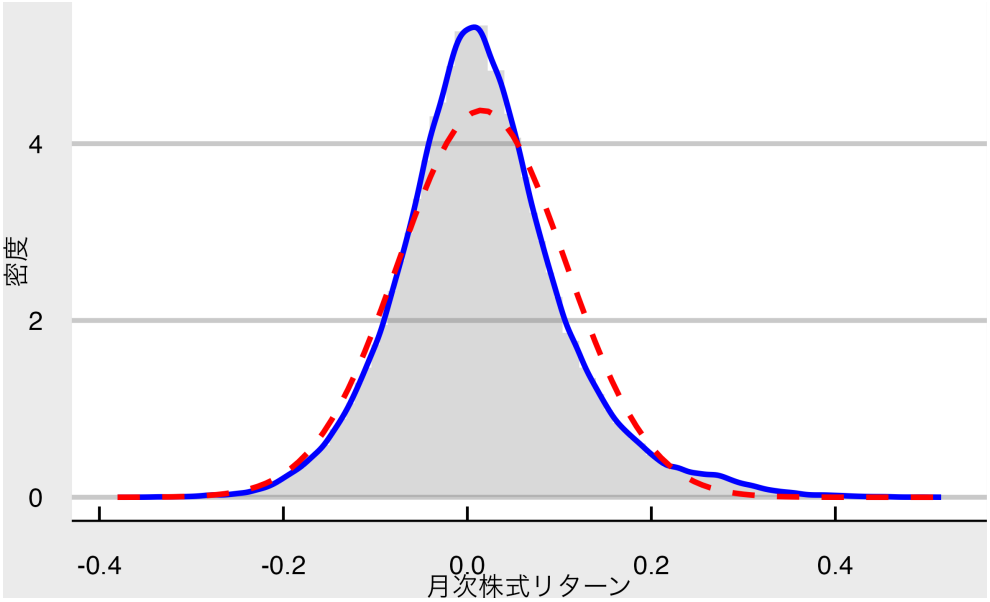
図でも確認するために，トータルリターン R のヒストグラムを書いてみます。

```
ggplot(stock_data) +
  aes(x = R) + geom_histogram() + # トータルリターンのヒストグラム
  xlab("月次株式リターン") + ylab("度数") + mystyle# 軸ラベル
```



トータルリターンのヒストグラムに正規分布のグラフを重ねてみると，次のようになります。

```
# トータルリターン
R <- stock_data$R
# 平均と標準偏差を計算
m <- mean(stock_data$R, na.rm = TRUE)
s <- sd(stock_data$R, na.rm = TRUE)

ggplot(stock_data) +
  aes(x = R) +
  # y 軸を密度 (density) に設定
  geom_histogram(aes(y = after_stat(density)), bins = 100, alpha =
  ↪  0.3, fill = "gray50") +
  geom_density(color = "blue", linewidth = 1) +  # 密度曲線を追加
  # 正規分布の曲線を追加
  stat_function(
    fun = dnorm, args = list(mean = m, sd = s),
    color = "red", linetype = "dashed", linewidth = 1) +
  xlab("月次株式リターン") + ylab("密度") + mystyle
```

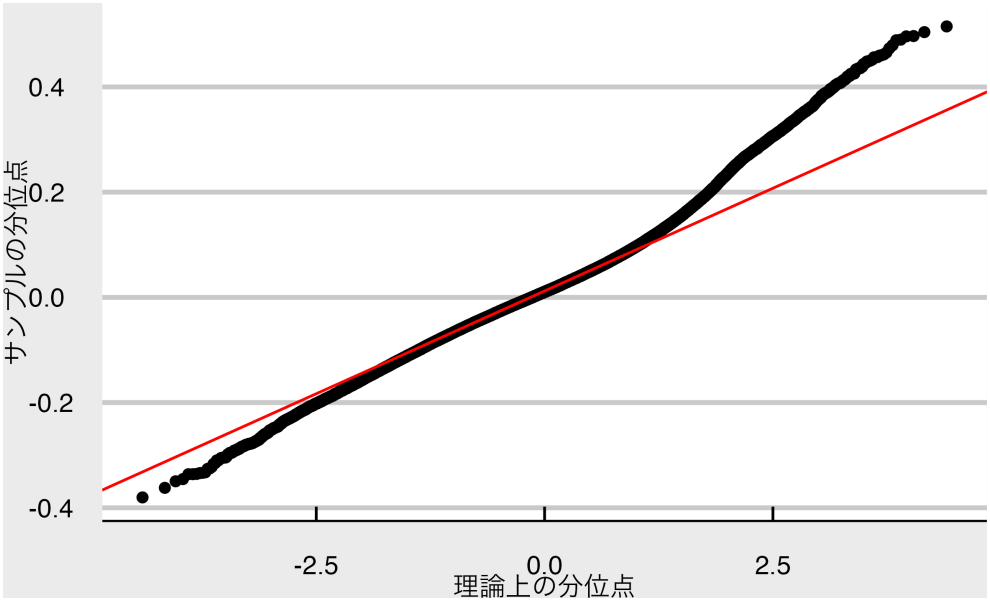


歪度と尖度の結果と整合的に、実際のトータル・リターンのデータの確率密度 (青い線) と正規分布 (赤い点線) を比べてみると、実際のデータは右に裾が長く，尖度も正規分布よりもとがった形となっていることが分かります。

より厳密にデータが正規分布にしたがっているかを確認するために，Q-Q プロットを描いてみます。

Listing 5.7 Q-Q プロット

```
ggplot(stock_data) +
  aes(sample = R) +
  stat_qq() +
  stat_qq_line(color = "red") +
  labs(x = "理論上の分位点", y = "サンプルの分位点") +
  mystyle
```



5.2　リターンの累積

5.2.1　バイ・アンド・ホールド・リターンの考え方

バイ・アンド・ホールド・リターン (buy-and-hold-return) は，ある時点で株式を購入し，そのまま保有し続けたときのリターンのことを指します。たとえば，12 月末に株式を購入し，3 月末に売却したときの、バイ・アンド・ホールド・リターンの累積は，次式で計算できます。ただし配当は無視します。

$$\left(\frac{W_{3\text{ 月}}}{W_{12\text{ 月}}}\right) = \underbrace{\left(\frac{W_{1\text{ 月}}}{W_{12\text{ 月}}}\right)}_{1+R_{1\text{ 月}}} \times \underbrace{\left(\frac{W_{2\text{ 月}}}{W_{1\text{ 月}}}\right)}_{1+R_{2\text{ 月}}} \times \underbrace{\left(\frac{W_{3\text{ 月}}}{W_{2\text{ 月}}}\right)}_{1+R_{3\text{ 月}}} \times$$

一般的に、月次で t 時点から T 時点までの年次リターンは、

$$\left(\frac{W_{\text{翌年 12 月末}}}{W_{12\text{ 月末}}}\right) = \prod_{t=1\text{ 月}}^{12\text{ 月}} (1 + R_t)$$

と計算できます。

5.3 年次リターンの計算

では、stock_data をもとに年次リターンを計算してみましょう。

```
annual_stock_data <- stock_data |>
  summarise(
    annual_R = prod(1 + R) - 1, # B&H 年次リターン
    annual_R_F = prod(1 + R_F) - 1, # 年次超過リターン
    .by = c(firm_ID, year) # 企業と年度でグループ化
  ) |>
  mutate(
    annual_Re = annual_R - annual_R_F # 年次超過リターン
    )
head(annual_stock_data)
```

```
# A tibble: 6 x 5
  firm_ID  year annual_R annual_R_F annual_Re
    <dbl> <dbl>    <dbl>      <dbl>     <dbl>
1       1  2015   NA       0.00743      NA
2       1  2016    0.997   0.000565      0.997
3       1  2017    0.688   0.0000488     0.688
4       1  2018   -0.214   0.00579      -0.219
5       1  2019    0.647  -0.000770      0.648
6       1  2020   -0.284   0.000380     -0.285
```

5.4 株式データと財務データを組み合わせた分析

ここでは，株式データと財務データを組み合わせて分析を行います。

5.4.1 データの結合

複数のデータフレームを結合する際に重要なところは、

1. データの頻度の違い
2. タイミングの一致

となる。今まで使ってきた財務データは年次データであるのに対して，株式データは月次データであるため，データの頻度が異なります。確認してみましょう。

```
financial_data <- readr::read_csv("data/ch04_output.csv")
nrow(financial_data) # 年次財務データの行数
```

```
[1] 7919
```

```
nrow(annual_stock_data) # 年次リターン・データの行数
```

```
[1] 7920
```

```
nrow(stock_data) # 月次リターン・データの行数
```

```
[1] 95040
```

月次リターンの行数が上の年次データとは大きく異なっていることが分かります。

> **先読みバイアス** (look-ahead bias) とは、ある時点での情報を使って、その時点よりも未来の情報を使っていることを指します。12 月末決算の会社のディスクロジャージャーは、最速で決算日後 45 日以内に出される決算短信か、3 ヶ月以内に出される有価証券報告書があります。このため、年次データを使って同時期の年次リターンを計算すると、年次データの発表後に出される有価証券報告書の情報を使っていることになります。これを先読みバイアスといいます。

```
annual_data <- annual_stock_data |>
  full_join(
    financial_data,
    by = c("firm_ID", "year")
  ) # firm_ID と year のペアをキーとして設定
```

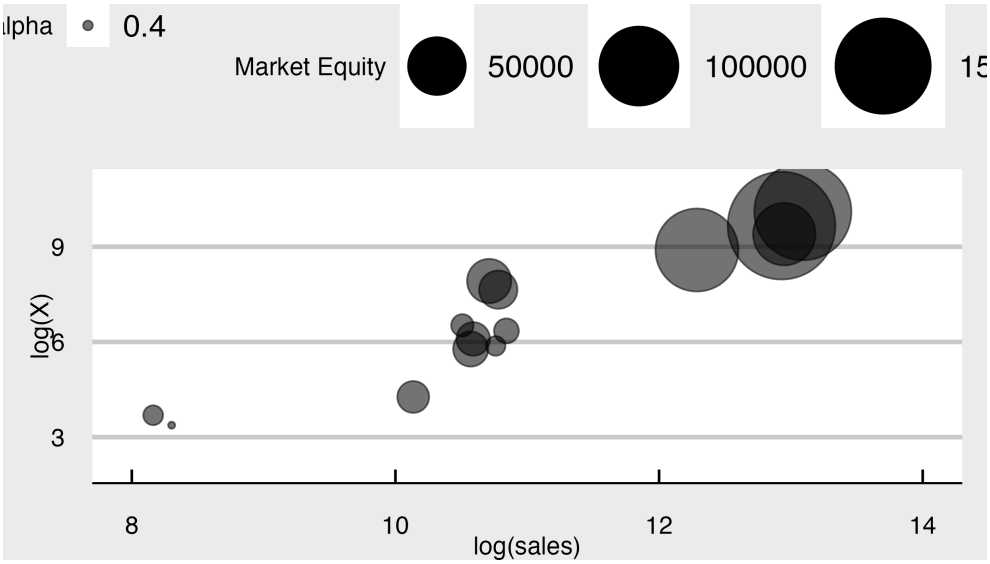
```
annual_data <- annual_stock_data |>
  full_join(financial_data) # キーを省略した場合，列名が同じ変数がキーになる
```

```
monthly_data <- stock_data |>
  full_join(financial_data, by = c("firm_ID", "year")) # firm_ID と
  ↪ year のペアをキーとして設定
```

5.5　バブルチャート

```
annual_data <- stock_data |>
  filter(month == 12) |> # 12 月のみ
  select(year, firm_ID, ME) |> # 変数を選択
  full_join(annual_data, ., by = c("year", "firm_ID")) |> # 年次データ
  ↪ と結合
  mutate(ME = ME / 1e6)
```

```
year2015 <- annual_data |>
   filter(
      year == 2015, # 2015 年のみ
      firm_ID %in% 2:20, # firm_ID が 2 から 20 のデータを抽出
      X > 0 # 対数を取るため当期純利益が正のデータのみ抽出
      )

ggplot(year2015) +
  aes(x = log(sales), y = log(X), size = ME, alpha = 0.4) +
  geom_point() + # バブルチャートを描くには size 引数を指定
  scale_size(range = c(1, 20), name = "Market Equity") + # range でバブ
  ↪ ルの最小最大面積を指定
  scale_x_continuous(limits = c(8, 14)) + # 両軸の範囲を指定
  scale_y_continuous(limits = c(2, 11)) + mystyle
```



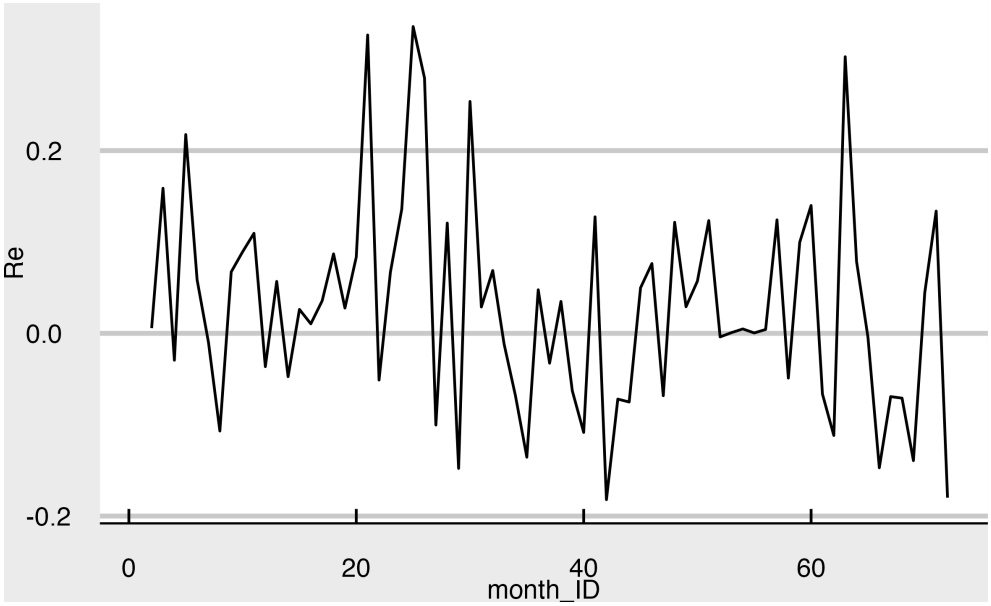
5.6 統計的推論入門

5.6.1 リターン・データに関する仮定

統計的推論の内容に入る前に、firm_ID が 1 の企業の超過リターン Re のデータを眺めてみます。

```
stock_data_1_month <- stock_data |>
  filter(firm_ID == 1) |>
  select(month_ID, Re, R)
```

```
ggplot(stock_data_1_month) +
  aes(x = month_ID, y = Re) + # 軸の設定
  geom_line() + mystyle # 折れ線グラフ
```



このようなデータは時系列データと呼ばれ、ある個体 (ここでは `firm_ID` が 1 の企業) の一定期間にわたって観測したデータのことを指します。

> **!** 仮定
>
> 月次超過リターンの時系列データは、何らかの確率分布 (モデル) から独立に生成されている。

1. 株価そのものでは無く変化率であるリターンをモデル化する理由

株価それ自体は株式分割などの要因でも変化するし、会社の成長に応じて上昇するため、その成長率をモデル化する方が、投資のリスクに対するリターンをより明確に評価することができるからである。

2. 月次リターンではなく月次超過リターンをモデルかする理由

投資リスクを取って得られる追加的なリターンであるリスクプレミアムを明確に評価するために、無リスク金利を差し引いた超過リターンをモデル化します。

3. 月次超過リターンが独立であるとする理由

半強度の効率的市場であれば、過去の情報はすでに株価に織り込まれているので、過去の超過リターンは将来の超過リターンと無関係である、とし、超過リターンに系列相関はない、と仮定します。しかし、アノマリーの存在などにより、現実には系列相関があると考えられますが、モデルが複雑になるので、ここでは独立の仮定をおいて、モデルを単純

化します。

4. 月次超過リターンが正規分布に従うとする理由

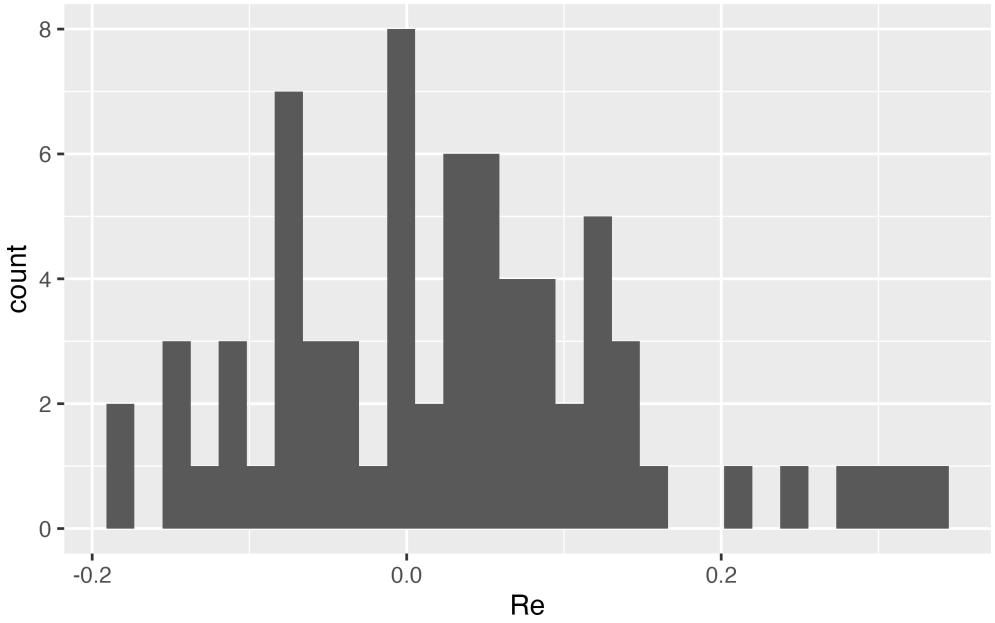
正規分布を仮定するといろいろ計算が楽になる、という理由とともに、株価の動きは多くの独立した事象の結果であると考えられ、中心極限定理により、超過リターンは正規分布に従うと考えられます。

これらの仮定を受け入れると、超過リターンの確率分布は次式で表されます。

$$Re_t \sim N(\mu, \sigma^2)$$

これにより、母集団分布に対して統計的推論が可能になります。`firm 1` の超過リターンのヒストグラムを書いてみます。

```
ggplot(stock_data_1_month) + aes(x = Re) + # データと変数
  geom_histogram()
```



サンプルサイズが小さいため凸凹しているけれど、もっとサンプルを増やせば、正規分布に近い形をとるはずです。

5.7 推定量と推定値の違い

推定量 (estimator) とは、母集団分布の母数 (パラメータ) を推定するために使われる統計量のことです。推定値 (estimates) とは、推定量に実際のデータを代入して計算した値のことです。たとえば母集団分布が正規分布 $N(\mu, \sigma^2)$ に従うと仮定すると、母数は μ と σ^2 の 2 つです。この母数を推定するために使われる統計量が、それぞれ標本平均 $\bar{x}$ と標本分散 s^2 です。このときの推定値とは、実際に観察された標本から計算された標本平均

$\bar{x}$ と標本分散 s^2 のことを指します。

次の問題を考えます。

> ❗ 問題
>
> firm_ID が 1 の銘柄の月次リターン $R^e_{i,t}$ は、期待値の意味で、ゼロより大きいのだろうか？

ここで「期待値の意味で」というのは平均的に、と言い換えても問題ないです。企業 1 の月次超過リターンの平均は 0 より大きい、ということは、企業 1 の月次リターンは平均的に無リスク利子率よりも大きいかどうか、を比べるということです。

この問題を解くために、まずは母集団分布の母数である期待値 μ を推定する必要があります。期待値 μ を推定するために使われる推定量は標本平均 $\bar{x}$ なので、ここで標本平均を計算します。

```
stock_data_1_month |>
  summarise(
    mean_Re = mean(Re, na.rm = TRUE)
  )
```

```
# A tibble: 1 x 1
  mean_Re
    <dbl>
1  0.0291
```

企業 1 の平均月次超過リターン mean_Re が 0.0291 となりました。これだけみるとゼロより大きい値ですが、これは母集団から抽出された 1 つのサンプルの平均ですので、他のサンプルの平均がゼロを超えるかどうかは分かりません。

5.7.1　大数の法則

母集団から無限個の標本 (sample) を抽出して、それぞれの標本平均 (sample mean) を計算すると、その標本平均の平均は母集団の期待値 μ に一致することが知られています。これを対数の法則 (law of large number) といいます。

数式よりも前にシミュレーションで確認してみましょう。まずは、母集団分布を平均が 10、標準偏差が 2 の正規分布 $N(10, 4)$ として、母集団から 100 のデータを抽出して標本を 1 つ作ります。そしてその標本の平均を計算します。

```
set.seed(123) # 乱数の種を固定
size <- 100 # 標本サイズ
# 母集団のパラメータの設定
mu <- 10 # 平均
```

```
sigma <- 4 # 標準偏差
population <- rnorm(size, mu, sigma) # 母集団からサンプルを抽出
mean(population) # 標本平均
```

```
[1] 10.36162
```

平均は 10.3616236 となりました。母集団の平均 10 とは異なる数値になっています。もう一度別の標本でやってみると、

```
population <- rnorm(size, mu, sigma) # 母集団からサンプルを抽出
mean(population) # 標本平均
```

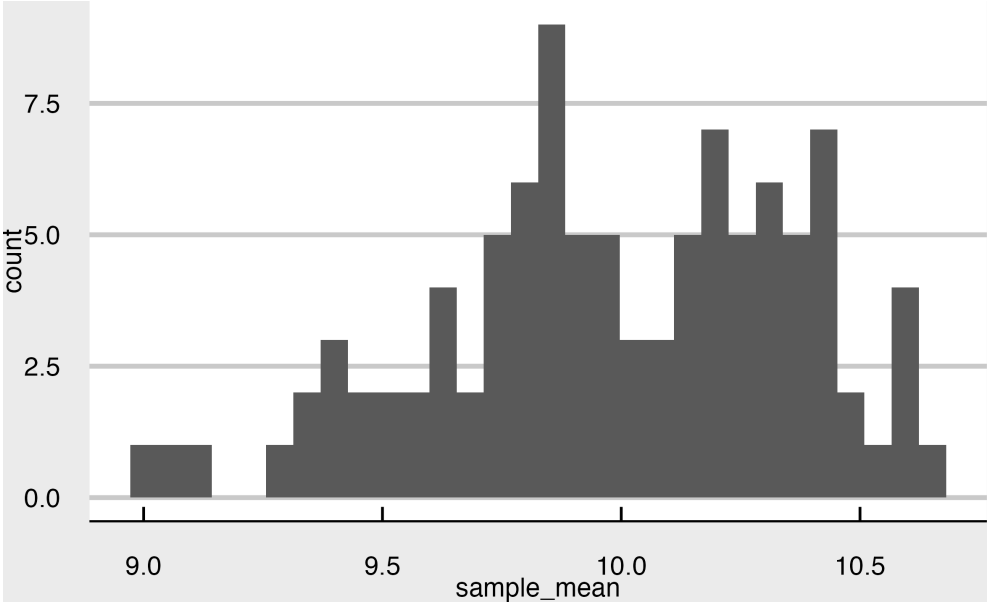
```
[1] 9.569813
```

また違う平均が計算されました。この作業を何度も何度も繰り返し、標本平均をたくさん計算します。ここでは、100 個のデータをもつ標本を 100 個作って、それぞれの標本平均を計算します。

```
n_sample <- 100 # サンプル数
sample_mean <- rep(NA, n_sample) # 標本平均を格納するベクトル
sample_sd <- rep(NA, n_sample) # 標本分散を格納するベクトル
for(i in 1:n_sample){
  population <- rnorm(size, mu, sigma) # 母集団からサンプルを抽出
  sample_mean[i] <- mean(population) # 標本平均を計算
  sample_sd[i] <- sd(population)
}
```

サンプルの平均が 100 個計算できたので、ヒストグラムを書いてみます。

```
ggplot() + aes(x = sample_mean) + geom_histogram() + mystyle
```



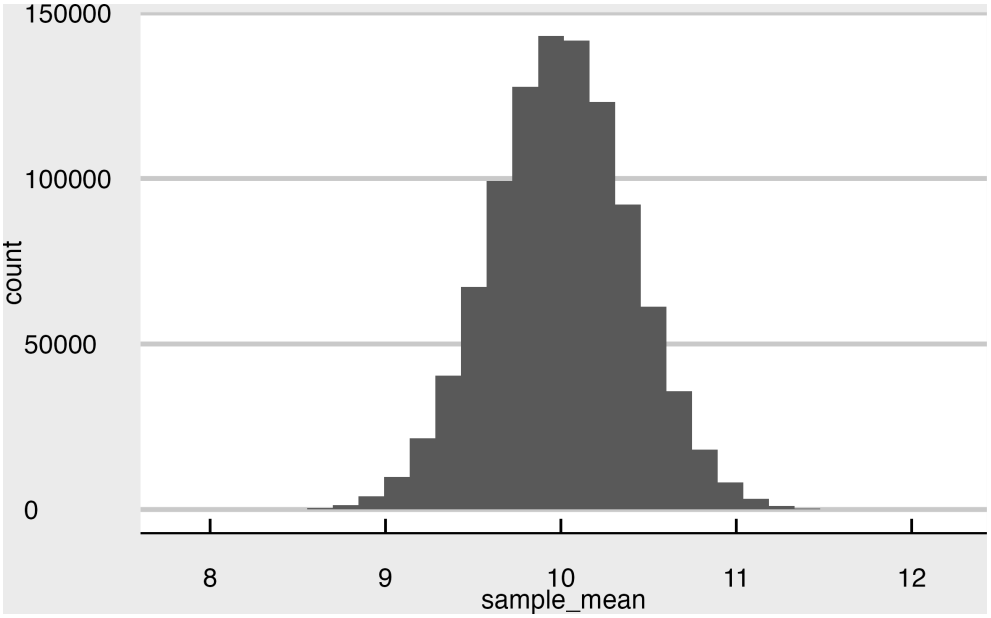
あまり正規分布のようには見えません。ではサンプルの平均の平均を計算してみます。

```
mean(sample_mean) # 標本平均の平均
```

```
[1] 9.99003
```

母集団の平均 10 に近い値になっていることが分かります。もっとサンプルの数を増やしてみます。データの数が 100 のサンプルを 1,000,000 個 (100 万個) 抽出して、それぞれのサンプルの平均値を計算します。

```
n_sample <- 10^6
# 標本平均のシミュレーション
sample_mean <- replicate(n_sample, mean(rnorm(size, mu, sigma)))
# 標本標準偏差のシミュレーション（必要な場合）
sample_sd <- replicate(n_sample, sd(rnorm(size, mu, sigma)))
# 作図
ggplot() + aes(x = sample_mean) + geom_histogram() + mystyle
```



ほぼ正規分布のような形をしていることが分かります。ではサンプルの平均の平均を計算してみます。

```
mean(sample_mean) # 標本平均の平均
```

```
[1] 10.00063
```

```
mean(sample_sd) # 標本分散の平均
```

```
[1] 3.989758
```

このように標本の数を無限に大きくしたとき、サンプルの平均の平均は母集団の平均 10 に一致するし，標本標準偏差は母集団の標準偏差 4 に一致する，というのが大数の法則です。

5.8 t 値の計算

先ほど計算した `firm_ID` が 1 の企業の超過リターンの平均値は NA でした。これがゼロより大きいかどうかはすぐ分かりますが，この値はたまたま今手元にある 1 つのサンプルから計算された平均値なので，他の標本ではどうなるか分かりません。このように推定量にばらつきがある場合には，その推定量の分布を考える必要があります。ここでは，その分布を **t 分布** (t distribution) と仮定して，t 値 (t-value) を計算してみます。t 値は次のように定義されます。

$$t = \frac{\bar{X} - \mu_0}{\sqrt{s^2/n}} \overset{d}{\approx} N(0,1)$$

ここで，$\bar{X}$ は標本平均，μ_0 は帰無仮説 (null hypothesis) の値で，ここでは $\mu_0 = 0$ とします。s^2 は標本分散，n は標本サイズです。分子に注目すると，標本平均と帰無仮説の

値の差となっており，もし標本平均が 0 に近いなら，t 値は 0 に近い値になります。

```
Re_firm_ID_1 <- stock_data |>
  filter(firm_ID == 1) |> # firm_ID が 1 の企業のデータを抽出
  select(Re) |> #超過リターンのみ選択
  drop_na() |> # 欠損値を除去
  unlist() # データフレームをベクトルに変換

mu0 <- 0 # 帰無仮説の値
n <- length(Re_firm_ID_1) # 標本サイズ

t_value <- (mean(Re_firm_ID_1) - mu0) / sqrt(var(Re_firm_ID_1) / n) #
 ↪  $t$値の計算
print(t_value)
```

```
[1] 2.121296
```

5.8.1　統計的検定の考え方

あなたがサイコロを投げるゲームをしていて、あるプレイヤーが非常に幸運だと主張しています。彼は 6 回サイコロを投げて、5 回も「6」が出たとします。これはただの偶然なのか、それとも何か他の要因（例えば、サイコロがいかさまであるとか）が関与しているのでしょうか？

この問いに答えるために、我々は**統計的検定** (statistical test) を用いることができます。まず**帰無仮説** (null hypothesis) を設定します。この例では、帰無仮説は「サイコロは公正であり、すべての出目が等確率（1/6）で出る」とすることが適切です。

次に、この帰無仮説が真 (true) である場合に、我々が観察した結果（5 回の「6」）がどれほどあり得ないかを計算します。これが p 値 (p value) です。この場合、6 回投げて 5 回「6」が出る確率を計算します。

これを計算すると、p 値は非常に小さいことが分かり（つまり、この結果は帰無仮説の下ではほぼあり得ない），帰無仮説が棄却されます。帰無仮説が棄却されるとは，**帰無仮説が正しいと仮定したときに，観測されたデータが得られる確率が小さいことを意味します**。したがって、我々はこの結果が偶然生じたとは考えにくく、その代わりにサイコロがいかさまである、または何か他の非ランダムな要因が作用している可能性を強く疑うことになります。これが p 値を用いて統計的検定の考え方です。

p 値が 0.05 より小さい場合，帰無仮説は棄却され，対立仮説が採択される，というケースが多いです。この場合，有意水準 5% で帰無仮説は棄却されます。有意水準は，帰無仮説が正しいと仮定したときに，観測されたデータが得られる確率が小さいと判断する基準値です。有意水準は，5% や 1% がよく使われます。

R では，`t.test()` 関数を使って，t 値と p 値を計算することができます。ここでは，

t.test() 関数を使って，firm_ID が 1 の企業の超過リターンがゼロなのかどうなのか，を検定するために，t 値と p 値を計算してみましょう。

```
t.test(Re_firm_ID_1)
```

```
    One Sample t-test

data:  Re_firm_ID_1
t = 2.1213, df = 70, p-value = 0.03744
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.001737894 0.056383256
sample estimates:
 mean of x
0.02906058
```

5.9 線形回帰入門

年次財務データ annual_data を使って線形回帰 (linear regression) について学習します。線形 (linear) とは，X と β の積が線形であることを意味します。回帰 (regression) とは，X と β の積が Y を説明することを意味します。回帰は，2 変数間の間で、一方の変数が他方の変数に対して影響を与えるという関係を想定できる場合に用いられます。しかし因果関係を直接調べているのではないことに注意しましょう。

影響を与える変数を説明変数 (explanatory variable) や独立変数 (independent variable) といい、影響を与えられる変数を目的変数 (response variable) や従属変数 (dependent variable) といいます。分野などでどっちを使うのかは異なりますが、ここでは説明変数と目的変数を使います。

線形関係を仮定した関係を式にすると，次のようになります。

$$Y = \alpha + \beta X$$

ここで，α は定数項 (constant term) と呼ばれ，切片 (intercept) とも呼ばれます。β は回帰係数 (regression coefficient) と呼ばれ，傾き (slope) とも呼ばれます。

実際のデータが上のモデルを満たす、つまりすべてのデータが直線上に並んでいるわけではないのです。実際のデータとモデルとの間には**ずれ**があることを考慮して、上のモデルを次のように書き換えます。

$$Y = \alpha + \beta X + u$$

ここで u は誤差項 (error term) とか観察不可能項 (unobservable term) と呼ばれます。

線形回帰の目的は，X と Y の関係を表す α と β を推定することです。推定するために，観測できるデータを使います。観測できるデータは X と Y のペアで (X_i, Y_i) と表記します。このペアを**観測値** (observed value) と呼びます。この観測値は，X と Y の関係を表す α と β を推定するために使われます。推定するために使われる α と β を**推定値** (estimated value) と呼びます。推定値は，$\hat{\alpha}$ と $\hat{\beta}$ と表記して，観察値と区別します。

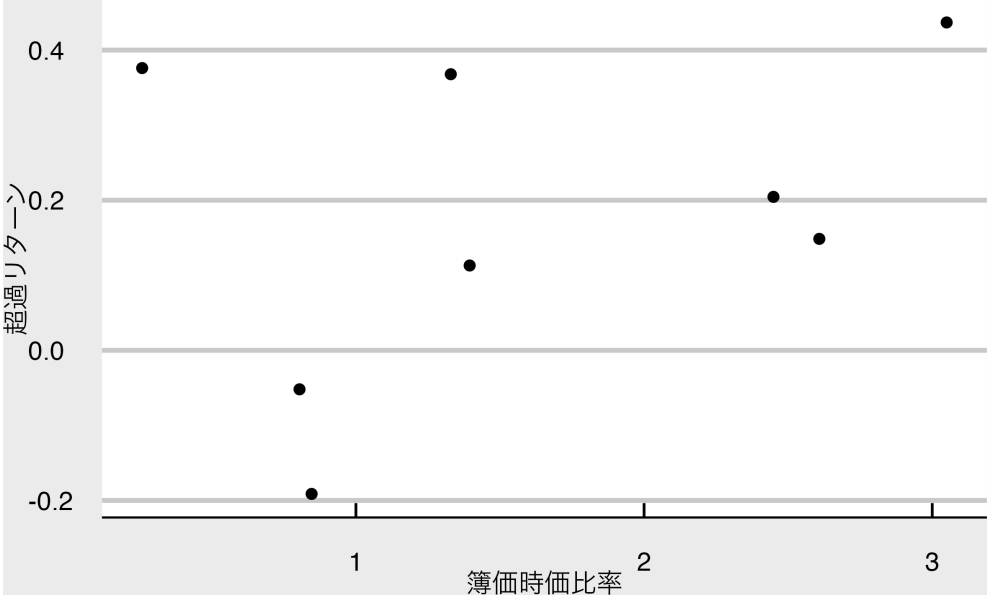
5.10　OLS 推定

推定方法として最も有名なものが**最小二乗法** (ordinary least squares; OLS) です。最小二乗法では，観測値と推定値の差の二乗の和が最小になるように α と β を推定します。このとき，観測値と推定値の差を**残差** (residual) と呼びます。最小二乗法では，残差の二乗の和である**残差平方和** (sum of squared residuals; SSR) が最小になるように α と β を推定します。

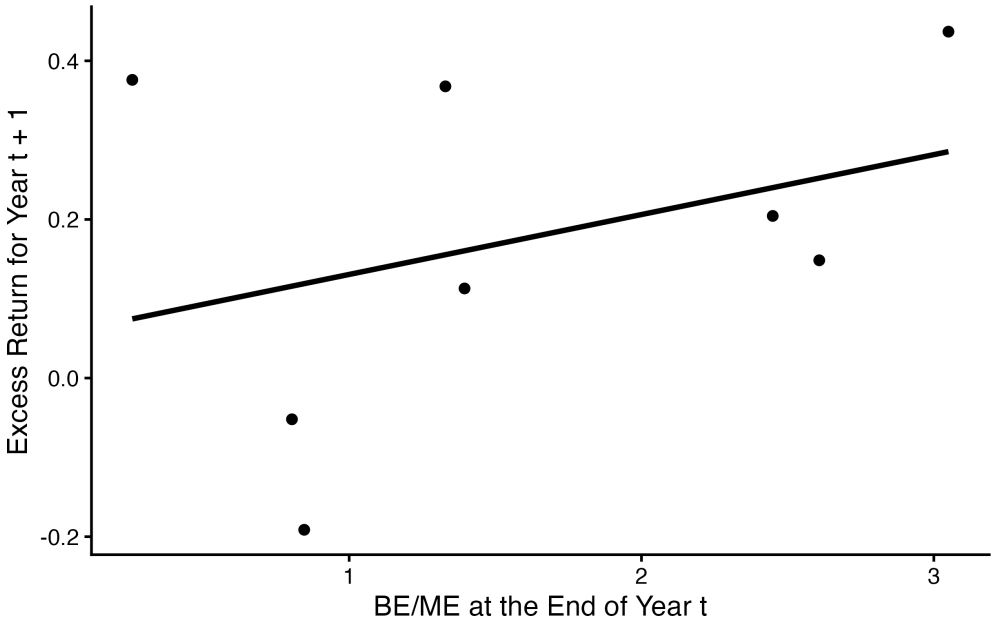
$$\min_{\alpha,\beta} \sum_{i=1}^{n} (Y_i - \alpha - \beta X_i)^2$$

```
# 線形回帰分析のための事前準備
# ch05_24 ですでに ME（時価総額）が結合されていることが前提

annual_data <- annual_data |>
  arrange(firm_ID, year) |> # 時系列順に並べ替え（lag 計算に必須）
  mutate(
    lag_ME = lag(ME),
    lagged_BEME = lagged_BE / lag_ME, # ここで BEME を計算
    .by = firm_ID
  )
```

```
lm_sample_data <- annual_data |>
  filter(year == 2016, firm_ID <= 10) |>
  # lagged_BEME は計算済みなので、select で選ぶだけにする
  select(firm_ID, year, Re = annual_Re, lagged_BEME) |>
  drop_na() # 欠損値を除去

ggplot(lm_sample_data) +
  geom_point(aes(x = lagged_BEME, y = Re)) + # 散布図
  xlab("簿価時価比率") + ylab("超過リターン") + mystyle
```




```
# ch05_32: 簿価時価比率と株式リターンの散布図に回帰直線を追加

ggplot(lm_sample_data) +
  geom_point(aes(x = lagged_BEME, y = Re)) +
  geom_smooth(aes(x = lagged_BEME, y = Re), method = "lm", se = FALSE,
   ↪  color = "black") + # 回帰直線を追加するには geom_smooth() 関数を用
   ↪  いる
  labs(x = "BE/ME at the End of Year t", y = "Excess Return for Year t
   ↪  + 1") +
  theme_classic()
```




```
# ch05_33: lm() 関数を用いた線形回帰

lm_results <- lm(Re ~ lagged_BEME, data = lm_sample_data) # 従属変数 ~
 ↪  独立変数
names(lm_results)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

```
print(lm_results$coefficients)
```

```
(Intercept) lagged_BEME
 0.05513385  0.07552921
```

```
# ch05_35: broom パッケージの tidy() 関数で係数の推定値に関する結果を確認
tidy(lm_results)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)   0.0551    0.156      0.354   0.736
2 lagged_BEME   0.0755    0.0844     0.895   0.405
```

```
glance(lm_results)
```

```
# A tibble: 1 x 12
```

```
  r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
      <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
1     0.118       -0.0294 0.223     0.800   0.405     1   1.81  2.37  2.61
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

5.11 対数回帰モデル

独立変数 X が変化したときの従属変数 Y への影響は一定 (つまり傾きが一定) と仮定してきましたが、実際には傾きが一定でない場合もあります。回帰式が非線形であることが想定される場合、対処法として

- 多項式回帰 (polynomial regression)：独立変数に X^2 とか X^3 を加える
- **対数回帰** (logarithmic regression)：独立変数や従属変数の対数をとる

たとえば、独立変数を対数変換した場合は、次のようなモデルになる。

$$Y_i = \beta_0 + \beta_1 \log(X_i) + \varepsilon_i$$

Listing 5.8 線形・対数モデルによる推定

```
tidy(lm(Re ~ log(lagged_BEME), data = lm_sample_data)) # 右辺のみ log()
 ↪  関数で自然対数を取る
```

```
# A tibble: 2 x 5
  term             estimate std.error statistic p.value
  <chr>               <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)         0.167     0.0868     1.93    0.102
2 log(lagged_BEME)    0.0355    0.109      0.326   0.756
```

作成したデータフレームを csv ファイルとして保存するには，`write_csv()` 関数を用います。前処理が終わった後や新しい変数を作った後に、データを保存しておくと便利です。6 章以降では、以下のデータを継続して使うので、csv ファイルとして保存しておきます。

Listing 5.9 データの保存

```
write_csv(monthly_data, "data/ch05_output1.csv")
write_csv(annual_data, "data/ch05_output2.csv")
```

第 6 章

ファクターモデルの導入

```
# パッケージの読み込み
pacman::p_load(
  tidyverse, # 便利なパッケージ群
  ggthemes, # ggplot2 のテーマ集
  broom # モデルの整形
)
# グラフのスタイル設定
mystyle <- list(
  theme_economist_white(
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),
  theme(
    text = element_text(size = 12),
    axis.title = element_text(size = 12)
  )
)
```

6.1 ファクター構築の準備

1. CAPM の実証的検証に必要な市場ポートフォリオの構築
2. ある特徴に基づいて各銘柄をランキングにし，ランキングに応じたポートフォリオの構築

6.2　市場ポートフォリオの構築

市場ポートフォリオ (market portfolio) とは，**市場に存在する全ての危険資産を時価総額比率で保有したポートフォリオ**をいいます。

入手可能な全銘柄の前年末時価総額 $\sum ME_{j,t-1}$ **と個別銘柄の時価総額** $ME_{i,t-1}$ **の比率**をウェイト $w_{i,t}^{M}$ として市場ポートフォリオを構築します。

厳密には，リスク資産には株式や債券に代表される金融資産の他、不動産や貴金属などの実物資産も含まれますが、実用上は TOPIX や S&P500 といった株価指数と同一視されることが多いです。

$$w_{i,t}^{M} = \frac{ME_{i,t-1}^{12月}}{\sum_{j=1}^{N} ME_{j,t-1}^{12月}}$$

株価は日々変動するため、時価総額も変動します。そのため、毎年 1 月に時価の変動で崩れた比率をリセットするために、市場ポートフォリオの中身を入れ替える**リバランス**を行います。

では練習用データで市場ポートフォリオを構築してみましょう。

```
monthly_data <- read_csv("data/ch05_output1.csv")
annual_data <- read_csv("data/ch05_output2.csv")
```

monthly_data は**月次の**株式データが収録されており、annual_data は**年次の**財務データが収録されています。データの構造を確認しておきます。

```
glimpse(monthly_data)
```

```
Rows: 95,040
Columns: 24
$ year        <dbl> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015~
$ month       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7~
$ month_ID    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
$ firm_ID     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ stock_price <dbl> 954, 960, 1113, 1081, 1317, 1366, 1353, 1209, 1291, 1407, ~
$ DPS         <dbl> 0, 0, 0, 0, 0, 29, 0, 0, 0, 0, 0, 29, 0, 0, 0, 0, 0, 40, 0~
$ n_shares    <dbl> 2422000, 2422000, 2422000, 2422000, 2422000, 2422000, 2422~
$ adj_coef    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ R_F         <dbl> 6.506826e-04, 5.834099e-04, 6.114423e-04, 6.848180e-
04, 7.~
$ ME          <dbl> 2310588000, 2325120000, 2695686000, 2618182000, 3189774000~
$ R           <dbl> NA, 0.006289308, 0.159375000, -0.028751123, 0.218316374, 0~
$ Re          <dbl> NA, 0.005705898, 0.158763558, -0.029435941, 0.217579602, 0~
$ industry_ID <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 1, 1, 1, 1~
$ sales       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 5948.96, 5~
$ OX          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 564.14, 56~
```

```
$ NFE          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 50.66750, ~
$ X            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 513.48, 51~
$ OA           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 13865.58, ~
$ FA           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 4642.16, 4~
$ OL           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 4534.22, 4~
$ FO           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 3959.70, 3~
$ BE           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 10013.82, ~
$ lagged_BE    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ ROE          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
```

```
glimpse(annual_data)
```

```
Rows: 7,920
Columns: 20
$ year         <dbl> 2015, 2016, 2017, 2018, 2019, 2020, 2015, 2016, 2017, 2018~
$ firm_ID      <dbl> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4~
$ ME           <dbl> 3577.294, 6883.324, 11376.990, 8694.752, 13957.518, 9708.9~
$ annual_R     <dbl> NA, 0.99727265, 0.68786382, -0.21361287, 0.64683576, -
0.28~
$ annual_R_F   <dbl> 7.432089e-03, 5.649890e-04, 4.884804e-05, 5.786109e-
03, -7~
$ annual_Re    <dbl> NA, 0.99670766, 0.68781497, -0.21939898, 0.64760569, -
0.28~
$ industry_ID <dbl> NA, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
$ sales        <dbl> NA, 5948.96, 6505.06, 6846.38, 7572.24, 7537.63, 3505.75, ~
$ OX           <dbl> NA, 564.14, 691.18, 751.29, 958.53, 778.37, 45.82, 51.25, ~
$ NFE          <dbl> NA, 50.667498, 29.543157, 86.486500, 298.049774, -
65.45877~
$ X            <dbl> NA, 513.48, 661.64, 664.80, 660.48, 843.83, 40.07, 49.37, ~
$ OA           <dbl> NA, 13865.58, 13952.58, 18818.48, 18190.00, 20462.86, 2977~
$ FA           <dbl> NA, 4642.16, 7743.99, 7284.72, 9735.13, 10274.25, 2258.33,~
$ OL           <dbl> NA, 4534.22, 5111.22, 5137.28, 5487.96, 5371.38, 1840.35, ~
$ FO           <dbl> NA, 3959.70, 6159.02, 10123.91, 11362.22, 13772.15, 2340.8~
$ BE           <dbl> NA, 10013.82, 10426.33, 10842.01, 11074.95, 11593.58, 1054~
$ lagged_BE    <dbl> NA, NA, 10013.82, 10426.33, 10842.01, 11074.95, NA, 1054.9~
$ ROE          <dbl> NA, NA, 0.06607269, 0.06376165, 0.06091859, 0.07619267, NA~
$ lag_ME       <dbl> NA, 3577.294, 6883.324, 11376.990, 8694.752, 13957.518, NA~
$ lagged_BEME <dbl> NA, NA, 1.4547942, 0.9164401, 1.2469602, 0.7934756, NA, 0.~
```

この銘柄ごとの保有比率を計算するために，前年度末の時価総額を計算し，lagged_ME に代入します。annual_data は 2015 年から 2020 年のデータが入っています。lag() で

前期末の時価総額を lagged_ME に代入しようとしても 2015 年の前年のデータは存在しないので，欠損値になることに注意しましょう。

```
annual_data <- annual_data |>
    arrange(firm_ID, year) |> # firm_ID と year で並び換え
    mutate(
      .by = firm_ID, # 企業ごと
      lagged_ME = lag(ME) # 前期末時価総額
      )
```

この処理の結果がおおよそこんな感じになっているはずです。

firm_ID	year	ME	lagged_ME
1	2015	3577.294	NA
1	2016	6883.324	3577.294
1	2017	11376.990	6883.324

この lagged_ME を使って保有比率を計算します。年度ごとに時価総額を合計し、ある企業の前期末時価総額を合計時価総額で割ることで保有比率 w_M を計算します。

```
annual_data <- annual_data |>
    mutate(
      .by = year, # 年度ごとに
      # ウェイト
      w_M = lagged_ME / sum(lagged_ME, na.rm = TRUE)
    )
```

2015 年の lagged_ME は欠損値なので，w_M も欠損値になっていますが，2015 年のデータはもう使わないので無視します。

次に，2016 年以降の欠損値の行を削除するのではなく，保有比率 w_M をゼロに置き換えることで投資しないことを表します。mutate() と replace() を用いて変数の置き換えをします。

```
annual_data <- annual_data |>
    mutate( # w_M の欠損値を 0 に置き換える
        w_M = replace(
          w_M,
          year >= 2016 & is.na(w_M),
          0
        )
    )
```

この処理は、

1. annual_data に annual_data を代入し直す
2. mutate() 関数で変数を変換する
3. replace() 関数で w_M の値を置き換える
    1. w_M に対して、replace() 関数を適用し、
    2. year >= 2016 かつ
    3. 欠損値かどうかを判定する関数 is.na() を使って判別した w_M が欠損値の場合に、
    4. w_M の値を 0 に置き換える

replace() 関数は，第 1 引数のデータに対して，第 2 引数の条件を満たす要素を，第 3 引数の値に置き換えます。ここでは，w_M に対して，year が 2016 以上で，かつ w_M が欠損値の場合の w_M を 0 に置き換えています。

作成した保有比率を表すウェイト w_M の合計が 1 になっているかどうかを確認します。

```
annual_data |>
    summarise(
        weight_sum = round(sum(w_M), digits = 2),
        .by = year
    )
```

```
# A tibble: 6 x 2
   year weight_sum
  <dbl>      <dbl>
1  2015         NA
2  2016          1
3  2017          1
4  2018          1
5  2019          1
6  2020          1
```

確認できました。これまでの操作で変数を追加した annual_data に monthly_data に結合します。**完全外部結合** (full outer join) を行います。完全外部結合とは，データベースを連結する操作の 1 つで、2 つのデータフレームからそれぞれ特定のキーとなる列を指定して，キーの値が一致する行同士は連結し、一致しない残りの行もそのまますべて抽出するものです。

では full_join() 関数を使って, annual_data と monthly_data を year と firm_ID の 2 つのキーで結合し，その結果を monthly_data に代入します。

できあがった拡大データセット monthly_data を確認します。

```
glimpse(monthly_data)
```

```
Rows: 95,040
```

Listing 6.1 月次データに保有比率のデータを追加

```
monthly_data <- annual_data |>
  select(year, firm_ID, w_M) |> # 必要な変数のみ
  full_join(monthly_data, by = c("year", "firm_ID")) |> # 完全外部結合
  select(-w_M, w_M) # w_M を最終列に移動
```

```
Columns: 25
$ year        <dbl> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015~
$ firm_ID     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ month       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7~
$ month_ID    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
$ stock_price <dbl> 954, 960, 1113, 1081, 1317, 1366, 1353, 1209, 1291, 1407, ~
$ DPS         <dbl> 0, 0, 0, 0, 0, 29, 0, 0, 0, 0, 0, 29, 0, 0, 0, 0, 0, 40, 0~
$ n_shares    <dbl> 2422000, 2422000, 2422000, 2422000, 2422000, 2422000, 2422~
$ adj_coef    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ R_F         <dbl> 6.506826e-04, 5.834099e-04, 6.114423e-04, 6.848180e-
04, 7.~
$ ME          <dbl> 2310588000, 2325120000, 2695686000, 2618182000, 3189774000~
$ R           <dbl> NA, 0.006289308, 0.159375000, -0.028751123, 0.218316374, 0~
$ Re          <dbl> NA, 0.005705898, 0.158763558, -0.029435941, 0.217579602, 0~
$ industry_ID <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 1, 1, 1, 1~
$ sales       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 5948.96, 5~
$ OX          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 564.14, 56~
$ NFE         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 50.66750, ~
$ X           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 513.48, 51~
$ OA          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 13865.58, ~
$ FA          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 4642.16, 4~
$ OL          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 4534.22, 4~
$ FO          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 3959.70, 3~
$ BE          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 10013.82, ~
$ lagged_BE   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ ROE         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ w_M         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 2.233661e-
~
```

準備が整ったので，市場ポートフォリオの月次リターンを計算します。t 時点における市場ポートフォリオのリターン $R_{M,t}$ は、個別銘柄のリターン $R_{i,t}$ とウェイト $w_{i,t}^{M}$ の積の合計で表されます。

$$R_{M,t} = \sum_{i=1}^{N} w_{i,t}^{M} R_{i,t}$$

これを R で実装します。monthly_data を month_ID でグループ化し，summarise() 関数を用いて，R_M を計算し，その後で mutate() 関数を用いて，R_Me を計算し，その結果を factor_data に代入します。

Listing 6.2 市場ポートフォリオの月次リターンを計算

```
factor_data <- monthly_data |>
  filter(month_ID >= 13) |> # 2016 以降のデータを抽出
  summarise(
    R_F = R_F[1], # 無リスク金利を抽出
    R_M = sum(w_M * R, na.rm = TRUE), # 月次リターンの加重平均
    .by = month_ID
  ) |>
  mutate(R_Me = R_M - R_F) # 月次超過リターン変数を作成
```

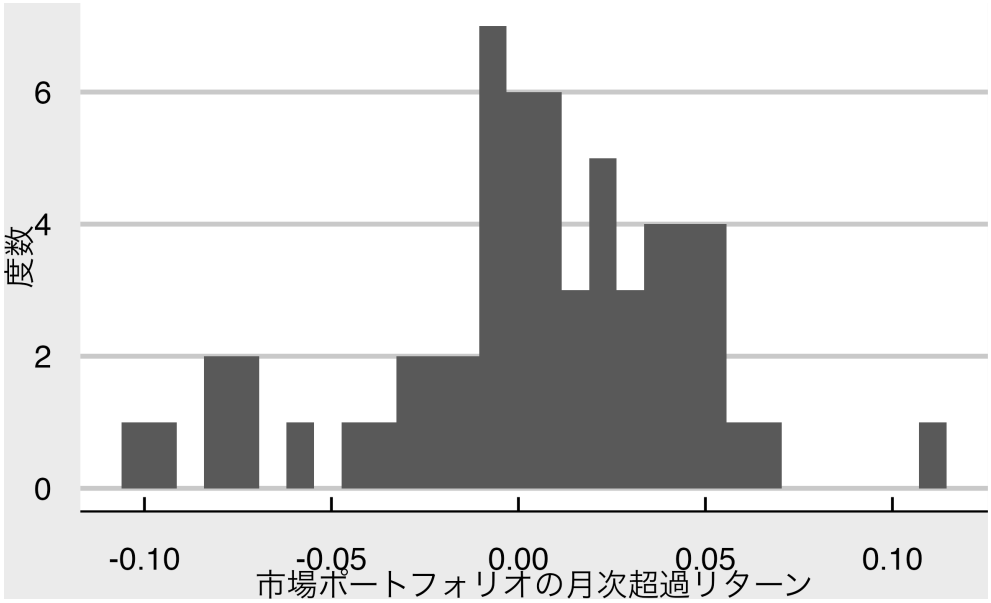
factor_data の中身を summary() で確認します。

```
summary(factor_data)
```

```
    month_ID          R_F                  R_M                 R_Me
 Min.   :13.00   Min.   :-2.329e-04   Min.   :-0.102438   Min.   :-
0.102438
 1st Qu.:27.75   1st Qu.:-4.107e-05   1st Qu.:-0.011056   1st Qu.:-
0.010890
 Median :42.50   Median : 3.870e-05   Median : 0.006081   Median : 0.006186
 Mean   :42.50   Mean   : 9.991e-05   Mean   : 0.004100   Mean   : 0.004000
 3rd Qu.:57.25   3rd Qu.: 1.323e-04   3rd Qu.: 0.031698   3rd Qu.: 0.031649
 Max.   :72.00   Max.   : 6.326e-04   Max.   : 0.111043   Max.   : 0.110819
```

作成した市場ポートフォリオの超過リターンをヒストグラムにして分布を確認します。

```
# 市場ポートフォリオの月次超過リターンをヒストグラムで可視化
ggplot(factor_data) + aes(x = R_Me) + geom_histogram() +
  labs(x = "市場ポートフォリオの月次超過リターン", y = "度数") + mystyle
```



次に、市場ポートフォリオの累積リターンを計算します。計算の仮定は以下の通りです。

1. month_ID が 13 の月初から運用スタートし、バイアンドホールドで運用すると仮定する。
2. 毎年 1 月にコストなしでリバランスし、リバランス前後で元本の変動はないと仮定する。

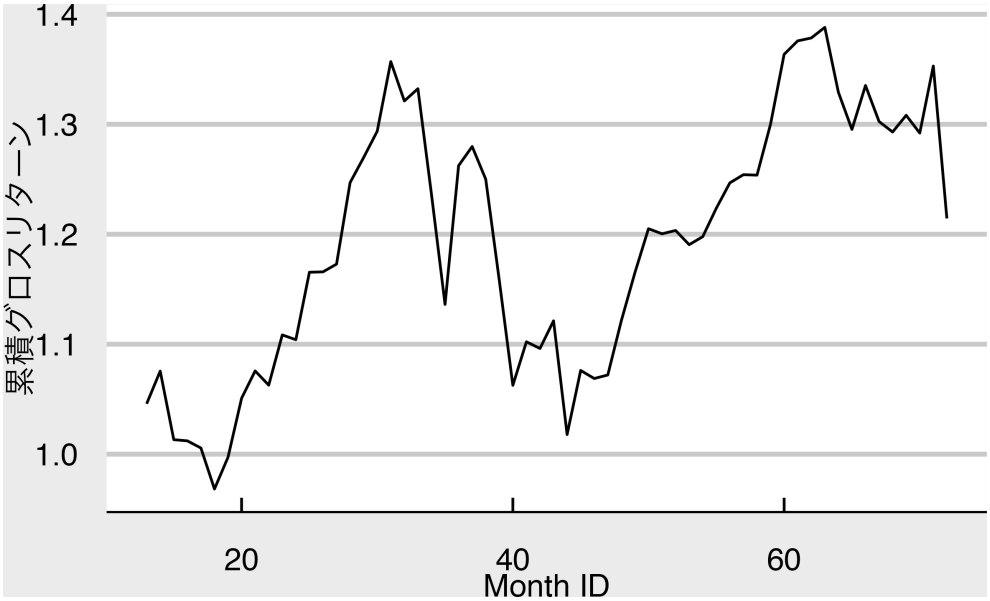
市場ポートフォリオの累積グロス・リターンを計算します。

Listing 6.3 市場ポートフォリオの累積リターンの可視化（1）

```
df_g <- factor_data |>
  mutate(
    gross_R_M = 1 + R_M, # r に 1 足してグロスリターン
    cumulative_gross_R_M = cumprod(gross_R_M) # 累積グロスリターン
    )
```

作成した累積グロス・リターンを折れ線グラフで可視化します。

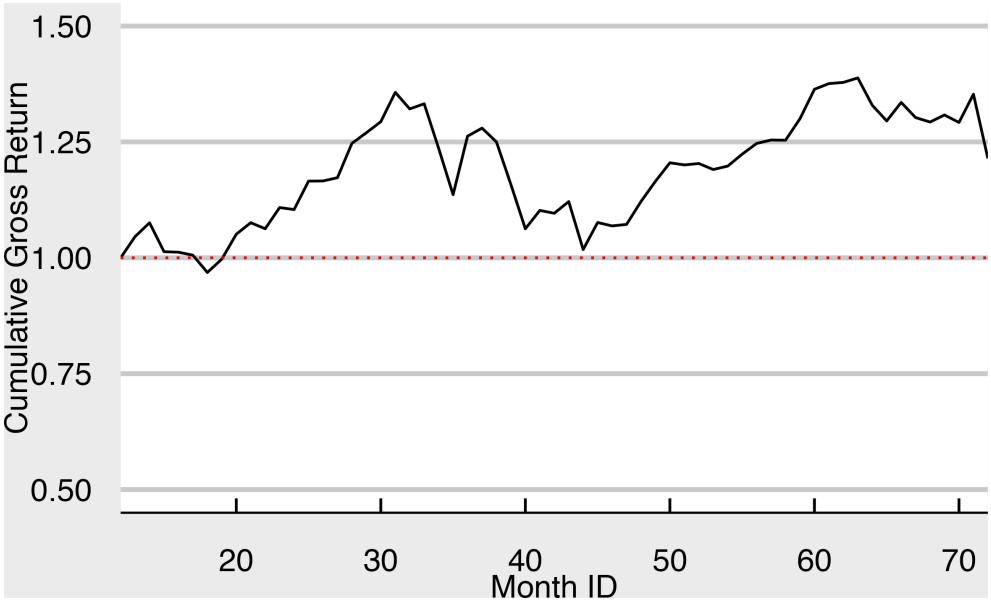
```
g <- ggplot(df_g) + aes(x = month_ID, y = cumulative_gross_R_M) +
 ↪  geom_line()
g <- g + labs(x = "Month ID", y = "累積グロスリターン") + mystyle
print(g)
```



累積リターンであることが一発で分かるように、始点を 1 として、折れ線グラフを描き直します。rbind() で始点となるデータを追加し、geom_hline() で始点の水準を点線で図示します。

Listing 6.4 市場ポートフォリオの累積リターンの可視化（2）

```
df_g <- factor_data |>
  mutate(
    gross_R_M = 1 + R_M,
    cumulative_gross_R_M = cumprod(gross_R_M)
  ) |>
  select(month_ID, cumulative_gross_R_M) |>
  add_row(month_ID = 12, cumulative_gross_R_M = 1, .before = 1)

# 折れ線グラフを作成
g <- ggplot(df_g) +
  geom_line(aes(x = month_ID, y = cumulative_gross_R_M)) +
  geom_hline(yintercept = 1, linetype = "dotted", color = "red") + #
    ↪ 元本の水準を点線で図示
  labs(x = "Month ID", y = "Cumulative Gross Return") +
  scale_x_continuous(expand = c(0, 0)) + ylim(0.5,1.5) +  mystyle
print(g)
```



6.3　ポートフォリオ・ソート

ある特性に基づいて株式銘柄をランキングにし、そのランキングに基づいてポートフォリオを構築することを**ポートフォリオ・ソート**と呼びます。ポートフォリオ・ソートは、ファクター・モデルの検証において重要な手法です。ここでは**前年度末の時価総額**に基づいて、企業を 10 個のグループに分類して、実現リターンの比較をしてみましょう。

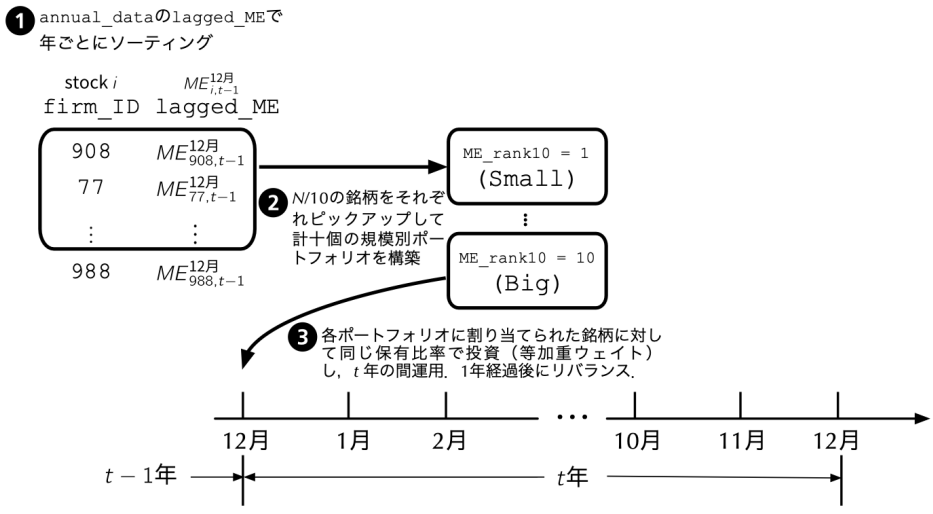


図 6.2　前年度末の時価総額に基づくポートフォリオ・ソートのイメージ

Figure6.1: 図 6.2 前年度末の時価総額に基づくポートフォリオ・ソート

R で時価総額ランキングを作成するには、ntile() 関数を用います。ntile() 関数は、データを指定した数のグループに分類します。以下のコードでは、mutate() 関数で ME_rank10 を新たに作成しています。ME_rank10 は、lagged_ME 変数を ntile() 関数で 10 個に分類し、as.factor() 関数で因子型に変換したものです。

```
# ch06_11: 前年度末の時価総額に基づくポートフォリオ・ソート (1)
annual_data <- annual_data |>
  mutate(
    ME_rank10 = as.factor(ntile(lagged_ME, 10)),
    .by = year
  ) # ntile() 関数を用いて十個のグループに分類
head(annual_data)
```

```
# A tibble: 6 x 23
   year firm_ID     ME annual_R annual_R_F annual_Re industry_ID sales    OX
  <dbl>   <dbl>  <dbl>    <dbl>      <dbl>     <dbl>       <dbl> <dbl> <dbl>
1  2015       1  3577.   NA      0.00743      NA              NA   NA    NA
2  2016       1  6883.    0.997  0.000565      0.997           1 5949.  564.
3  2017       1 11377.    0.688  0.0000488     0.688           1 6505.  691.
4  2018       1  8695.   -0.214  0.00579      -0.219           1 6846.  751.
5  2019       1 13958.    0.647 -0.000770     0.648           1 7572.  959.
6  2020       1  9709.   -0.284  0.000380     -0.285           1 7538.  778.
# i 14 more variables: NFE <dbl>, X <dbl>, OA <dbl>, FA <dbl>, OL <dbl>,
#   FO <dbl>, BE <dbl>, lagged_BE <dbl>, ROE <dbl>, lag_ME <dbl>,
#   lagged_BEME <dbl>, lagged_ME <dbl>, w_M <dbl>, ME_rank10 <fct>
```

ME_rank10 の値と、年・ランキングごとの会社数を確認してみましょう。

```
summary(annual_data$ME_rank10)
```

```
   1    2    3    4    5    6    7    8    9   10 NA's
 643  642  641  641  640  640  640  640  639  639 1515
```

```
table(annual_data$year,  annual_data$ME_rank10)
```

```
      1   2   3   4   5   6   7   8   9  10
2015  0   0   0   0   0   0   0   0   0   0
2016 125 124 124 124 124 124 124 124 124 124
2017 127 127 127 127 127 127 127 127 127 127
2018 127 127 127 127 127 127 127 127 126 126
2019 131 131 131 131 130 130 130 130 130 130
2020 133 133 132 132 132 132 132 132 132 132
```

ここでは、ME_rank10 の値が 10 の企業が時価総額ランキングの上位 10% に、1 の企業が時価総額ランキングの下位 10% に属することを意味します。

前回と同様に、full_join() 関数で monthly_data と annual_data を結合します。drop_na() 関数で欠損行を削除し、.by =month_ID と ME_rank10 に関してグループ化した上で、summarize() 関数で月次超過リターン Re の平均値を計算して、Re 変数としています。

Listing 6.5 前年度末の時価総額に基づくポートフォリオ・ソート

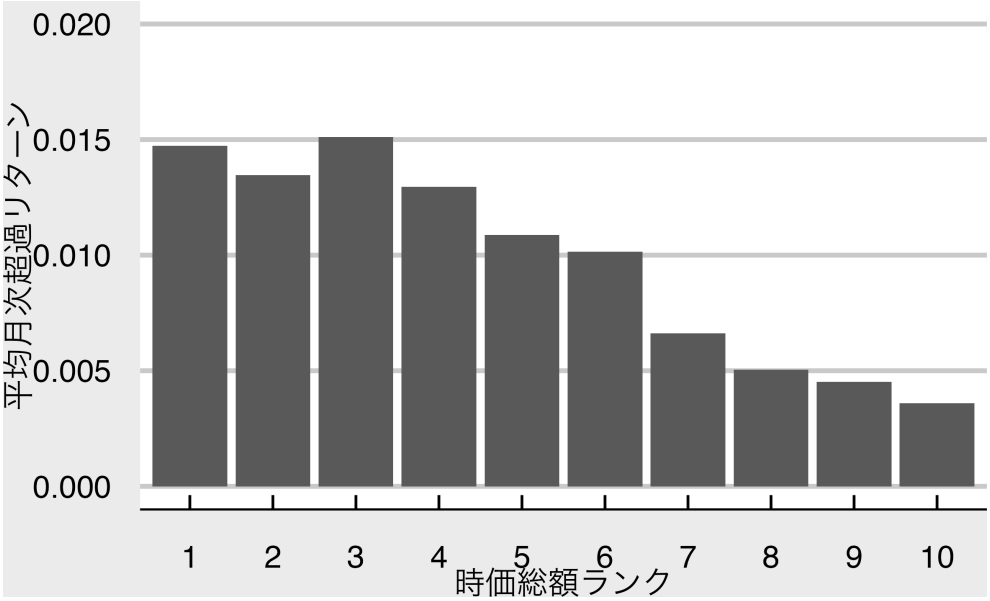
```
ME_sorted_portfolio <- annual_data |>
  select(year, firm_ID, ME_rank10) |> # 年次データから追加したい情報を抽出
  full_join(monthly_data, by = c("year", "firm_ID")) |> # year と
   ↪ firm_ID をキーに月次データと結合
  drop_na() |> # 欠損行を削除
  summarize(
    # 各グループで月次超過リターンの平均値を計算
    Re = mean(Re),
    .by = c(month_ID, ME_rank10)
    )
ME_sorted_portfolio
```

```
# A tibble: 600 x 3
   month_ID ME_rank10       Re
      <dbl> <fct>        <dbl>
 1       25 2          0.0976
 2       26 2          0.00392
 3       27 2          0.0172
 4       28 2          0.0773
 5       29 2          0.00948
 6       30 2          0.0473
 7       31 2          0.0313
 8       32 2          0.0136
 9       33 2          0.00911
10       34 2         -0.0962
# i 590 more rows
```

準備が出来たので、各ポートフォリオの平均超過リターンを可視化してみましょう。これにより、時価総額の大きい企業のポートフォリオが、時価総額の小さい企業のポートフォリオよりも高い、あるいは低いリターンを上げているかどうかを確認することができます。

Listing 6.6 各ポートフォリオの平均超過リターンを可視化

```
ME_cross_sectional_return <- ME_sorted_portfolio |>
  summarize(
    mean_Re = mean(Re),
    .by = ME_rank10
  ) # 月次超過リターンの平均値を計算

g <- ggplot(ME_cross_sectional_return) +
  aes(x = ME_rank10, y = mean_Re) +
  geom_col() + # 棒グラフ
  xlab("時価総額ランク") + ylab("平均月次超過リターン") +
  scale_y_continuous(expand = c(0, 0)) +
  ylim(0,0.02) + mystyle
print(g)
```



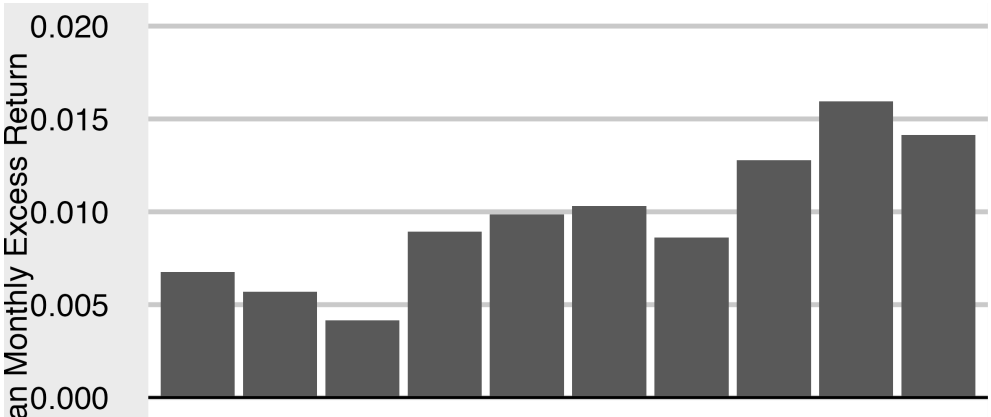
小型株ほど月次超過リターンの平均が高いことが分かりました。このように、時価総額の大きい企業のポートフォリオと小さい企業のポートフォリオのリターンの差を**サイズ・プレミアム**と呼びます。

先ほどは各ポートフォリオの区分を同じウェイトで保有した場合のリターンを計算しましたが，コラムでは，時価総額の大きさに応じてウェイトを変えた時価総額加重ポートフォリオを作成して，先ほどの結果を再現してみる。

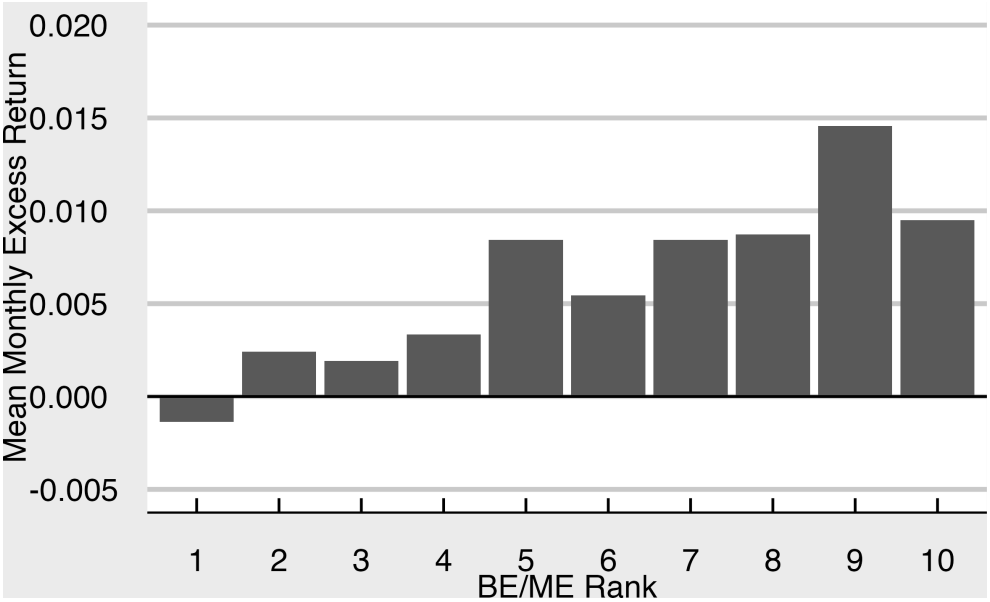
まずは等加重の場合のコードを確認する。

Listing 6.7 BPR に基づくポートフォリオ・ソート（等加重の場合）

```
annual_data <- annual_data |>
  mutate(
    lagged_BEME = lagged_BE / lagged_ME
    ) |>
  mutate(
    # 簿価時価比率に基づいて十個のグループに分類
    BEME_rank10 = as.factor(ntile(lagged_BEME, 10)),
    .by = year
    )

BEME_sorted_portfolio <- annual_data |>
  select(year, firm_ID, BEME_rank10, lagged_ME) |>
  full_join(monthly_data, by = c("year", "firm_ID")) |>
  drop_na() |>
  summarize(
    # 月次超過リターンの平均値を計算
    Re = mean(Re),
    .by = c(month_ID, BEME_rank10)
  )

# 作図
BEME_sorted_portfolio |>
  summarize(
    mean_Re = mean(Re),
    .by = BEME_rank10
  ) |>
  ggplot() +
  geom_col(aes(x = BEME_rank10, y = mean_Re)) +
  geom_hline(yintercept = 0) + # y = 0 の直線を追加
  labs(x = "BE/ME Rank", y = "Mean Monthly Excess Return") +
  scale_y_continuous(limits = c(-0.005, 0.02)) + mystyle
```



次に時価総額加重の場合のコードを確認します。



結果が異なっていることに注意しましょう。

次節では，この現象を，資産価格モデルの 1 つである CAPM(Capital Asset Pricing Model) が説明できるかどうかを検証します。

6.4 CAPM の実証的な検証

6.4.1 CAPM を検証する意義

まずは CAPM の復習から始めましょう。CAPM は，資産の期待リターンを，市場ポートフォリオの期待リターンと市場ポートフォリオとの共分散で説明するモデルです。

CAPM

- **第 1 命題**: 市場ポートフォリオは接点ポートフォリオと一致し，効率的フロンティア (資本市場線) 上に位置する.
- **第 2 命題**: 各証券のリスクプレミアムは，その証券のマーケット・ベータ に比例する.

$$\mathbb{E}[R_i] - R_F = \beta_i \left(\mathbb{E}[R_M] - R_F\right)$$

ただし，

$$\beta_i = \frac{\mathrm{COV}_{R_i, R_M}}{\mathrm{Var}_M}$$

この CAPM を回帰式で表現すると次のようになります。

$$R^e_{i,t} = \beta_i \times R^e_{M,t} + \varepsilon_{i,t}$$

ここで、$R^e_{i,t} = R_{i,t} - R_{F,t}$ である。つまり、t 時点における証券 i の実現超過リターン $R^e_{i,t}$ は、t 時点における市場ポートフォリオの実現超過リターン $R^e_{M,t}$ と、証券 i の市場ポートフォリオに対するベータ係数 β_i の積に、誤差項 $\varepsilon_{i,t}$ を加えたものとして表現されます。

また、誤差項 $\varepsilon_{i,t}$ に関して次の仮定を置きます。

- $varepsilon_{i,t}$ は独立同一分布 (i.i.d.) に従う
- $E[\varepsilon_{i,t}] = 0$
- $E[R^e_{M,t},\ \varepsilon_{i,t}] = 0$

こうすることで、CAPM 式を線形回帰モデルで表現できるので、β_i の推定が可能となります。

6.4.1.1　時系列回帰

CAPM 式は任意の i 証券で成立するモデルのため、ポートフォリオにも応用できます。つまりあるポートフォリオの超過リターンを、市場ポートフォリオの超過リターンと、そのポートフォリオに対するベータ係数の積で説明することができます。

$$R^e_{P,t} = \alpha_P + \beta_P R^e_{M,t} + \varepsilon_{P,t}$$

CAPM の式と比較すると、切片である α_P が追加されていることが分かります。もし証券市場に CAPM の関係が成立しているなら、α_P はゼロとなっているはずです。この $ $を調べることで、CAPM の検証が可能となります。

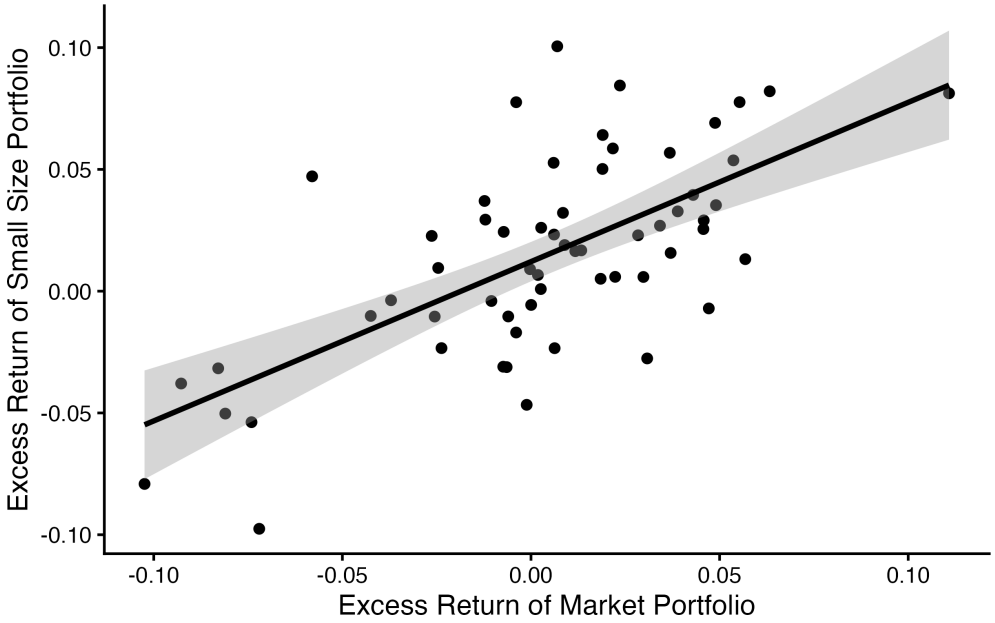
ここでは、**時系列回帰**を使って、市場ポートフォリオの超過リターンを説明変数として、各ポートフォリオの超過リターンを説明するモデルを推定します。

```
# ch06_16: 市場ポートフォリオの超過リターンを追加

ME_sorted_portfolio <- factor_data |>
  select(-R_F) |> # 無リスク金利は重複するので結合前に削除
  full_join(ME_sorted_portfolio, by = "month_ID") |> # month_ID を
  ↪ キーに
  select(-R_Me, R_Me) # R_Me を最終列へ移動
```

```
# ch06_17: 時系列回帰 (1)

ME_sorted_portfolio |>
  filter(ME_rank10 == 1) |> # 時価総額が最小のポートフォリオを抽出
  lm(Re ~ R_Me, data = _) |> # . を使って lm() 関数の第二引数にデータを代入
  tidy() # 線形回帰の結果を tidy() 関数でデータフレームに変換
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic       p.value
  <chr>          <dbl>     <dbl>     <dbl>         <dbl>
1 (Intercept)   0.0121   0.00404      3.00 0.00395
2 R_Me          0.654    0.0976       6.70 0.00000000937
```

```
# ch06_18: 時系列回帰 (2)

ME_sorted_portfolio |>
  filter(ME_rank10 == 1) |>
  ggplot(aes(x = R_Me, y = Re)) + # aes() 関数は ggplot() 関数の中にも代入
   ↪ 可能
  geom_point() +  # geom_point() 関数と次の geom_smooth() 関数で共通の
   ↪ aes() 関数を受け取る
  geom_smooth(method = "lm", color = "black") +
  labs(x = "Excess Return of Market Portfolio", y = "Excess Return of
   ↪ Small Size Portfolio") +
  theme_classic()
```

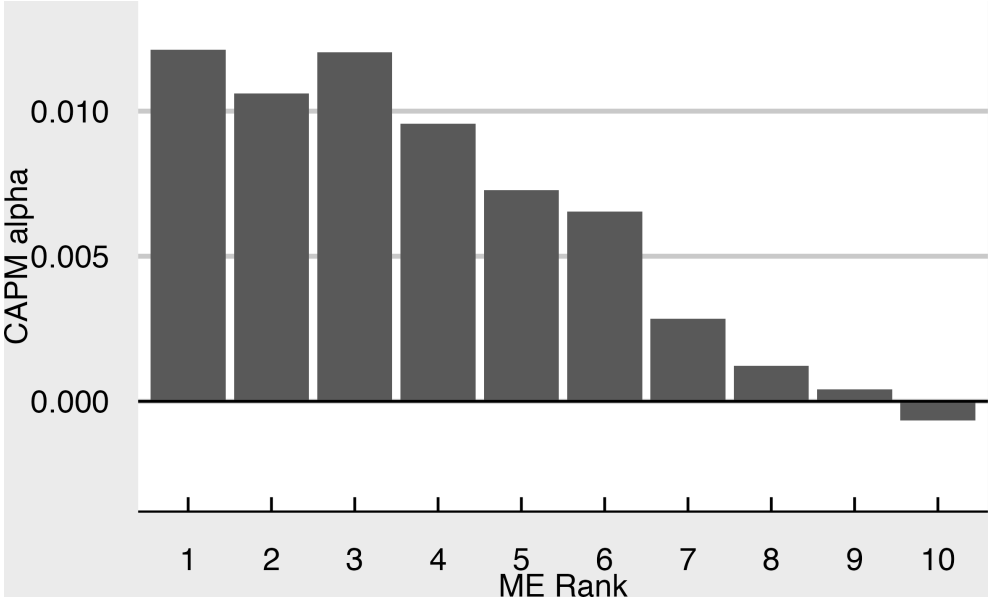


6.4.1.2 ポートフォリオごとの回帰

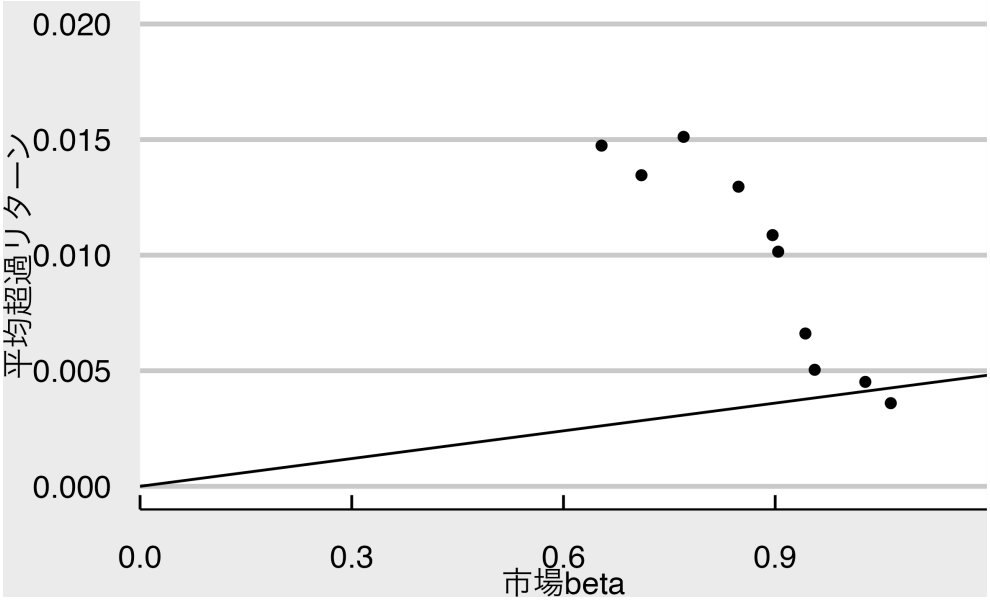
```
# A tibble: 20 x 6
   ME_rank10 term         estimate std.error statistic  p.value
```

	<fct>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	2	(Intercept)	0.0106	0.00375	2.83	6.44e- 3
2	2	R_Me	0.711	0.0908	7.83	1.19e-10
3	8	(Intercept)	0.00122	0.00168	0.723	4.73e- 1
4	8	R_Me	0.956	0.0407	23.5	2.59e-31
5	4	(Intercept)	0.00957	0.00289	3.31	1.62e- 3
6	4	R_Me	0.848	0.0699	12.1	1.54e-17
7	7	(Intercept)	0.00284	0.00173	1.64	1.06e- 1
8	7	R_Me	0.943	0.0418	22.6	2.16e-30
9	1	(Intercept)	0.0121	0.00404	3.00	3.95e- 3
10	1	R_Me	0.654	0.0976	6.70	9.37e- 9
11	9	(Intercept)	0.000406	0.00144	0.282	7.79e- 1
12	9	R_Me	1.03	0.0349	29.5	1.33e-36
13	3	(Intercept)	0.0120	0.00312	3.86	2.90e- 4
14	3	R_Me	0.770	0.0754	10.2	1.42e-14
15	6	(Intercept)	0.00653	0.00195	3.34	1.45e- 3
16	6	R_Me	0.904	0.0472	19.2	9.39e-27
17	5	(Intercept)	0.00728	0.00234	3.11	2.86e- 3
18	5	R_Me	0.896	0.0565	15.9	9.35e-23
19	10	(Intercept)	-0.000659	0.00113	-0.582	5.63e- 1
20	10	R_Me	1.06	0.0273	38.9	2.93e-43

6.4.1.3　CAPM アルファ



```
# A tibble: 10 x 4
   ME_rank10 CAPM_alpha  p_value significance
       <int>      <dbl>    <dbl> <fct>
 1         1   0.0121   0.00395  "***"
 2         2   0.0106   0.00644  "***"
 3         3   0.0120   0.000290 "***"
 4         4   0.00957  0.00162  "***"
 5         5   0.00728  0.00286  "***"
 6         6   0.00653  0.00145  "***"
 7         7   0.00284  0.106    ""
 8         8   0.00122  0.473    ""
 9         9   0.000406 0.779    ""
10        10  -0.000659 0.563    ""
```



6.4.2 Fama-French の 3 ファクター・モデル

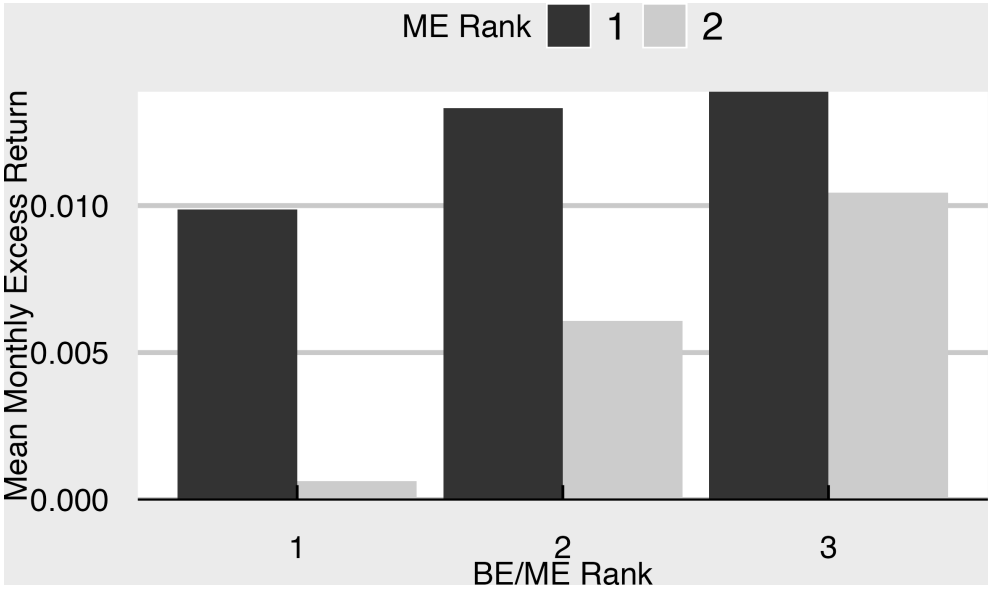
6.4.2.1 銘柄のランク付け

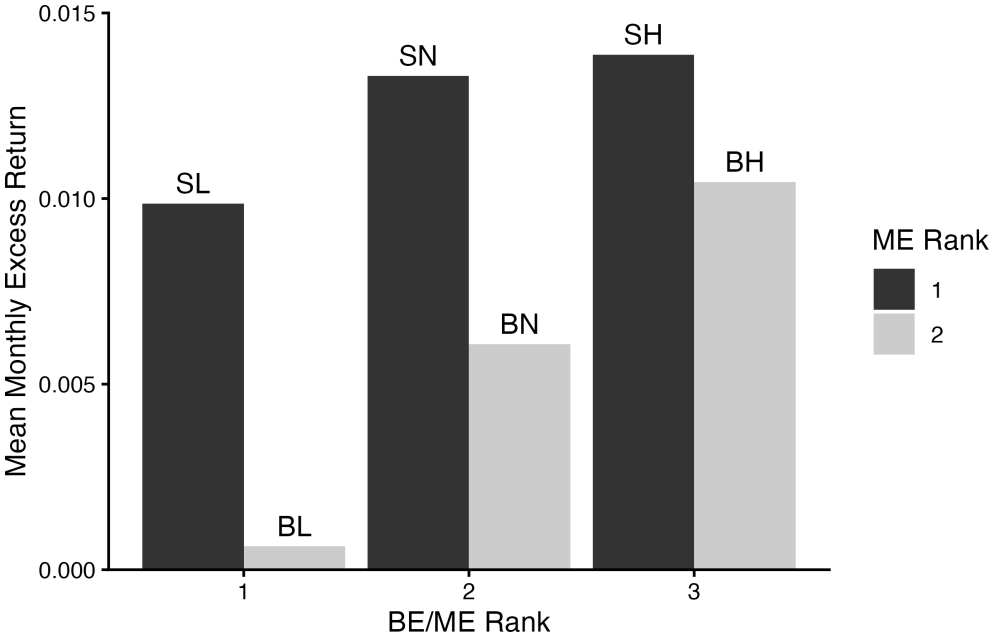
```
# A tibble: 7,920 x 26
    year firm_ID     ME annual_R annual_R_F annual_Re industry_ID sales    OX
   <dbl>   <dbl>  <dbl>    <dbl>      <dbl>     <dbl>       <dbl> <dbl> <dbl>
 1  2015       1  3577.   NA      0.00743      NA              NA   NA   NA
 2  2016       1  6883.    0.997  0.000565      0.997           1 5949. 564.
 3  2017       1 11377.    0.688  0.0000488     0.688           1 6505. 691.
 4  2018       1  8695.   -0.214  0.00579      -0.219           1 6846. 751.
```

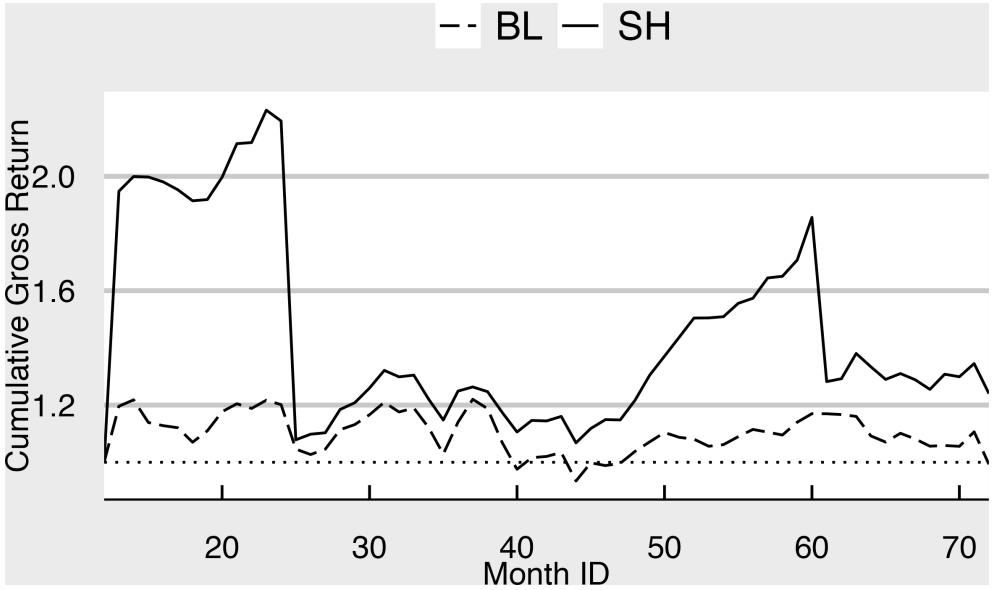
```
 5  2019       1 13958.    0.647 -0.000770      0.648           1 7572. 959.
 6  2020       1  9709.   -0.284  0.000380     -0.285           1 7538. 778.
 7  2015       2  4087.   NA      0.00579      NA               1 3506.  45.8
 8  2016       2  5593.    0.375 -0.000770      0.376           1 3491.  51.2
 9  2017       2  9153.    0.641  0.000380      0.640           1 3946.  83.4
10  2018       2  7104.   -0.220  0.00371      -0.223           1 4139.  93.4
# i 7,910 more rows
# i 17 more variables: NFE <dbl>, X <dbl>, OA <dbl>, FA <dbl>, OL <dbl>,
#   FO <dbl>, BE <dbl>, lagged_BE <dbl>, ROE <dbl>, lag_ME <dbl>,
#   lagged_BEME <dbl>, lagged_ME <dbl>, w_M <dbl>, ME_rank10 <fct>,
#   BEME_rank10 <fct>, ME_rank2 <fct>, BEME_percent_rank <dbl>
```

6.4.2.2　時価総額と BE/ME に基づくポートフォリオ・ソート

```
# A tibble: 6 x 6
  ME_rank2 BEME_rank3 FF_portfolio_type mean_BEME mean_ME mean_N_stocks
  <fct>    <fct>      <fct>                <dbl>   <dbl>         <dbl>
1 1        3          SH                   1.97   11023.          260.
2 1        2          SN                   0.973  11868.          226
3 1        1          SL                   0.416  11601.          155.
4 2        2          BN                   0.960 211793.          286
5 2        3          BH                   1.72  151135.          125.
6 2        1          BL                   0.468 414941.          230.
```

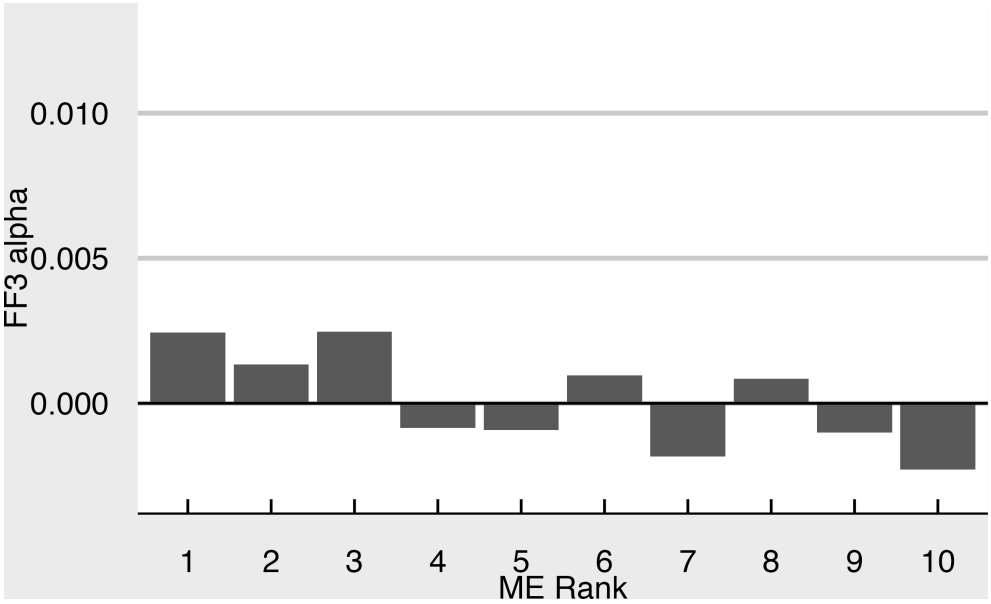






6.4.2.3 ファクター・リターンの計算

6.4.2.4 FF3 アルファ




```
# A tibble: 10 x 4
   ME_rank10 FF3_alpha p_value significance
       <int>     <dbl>   <dbl> <fct>
 1         1  0.00244   0.181  ""
 2         2  0.00134   0.412  ""
 3         3  0.00246   0.0893 "*"
 4         4 -0.000843  0.335  ""
 5         5 -0.000924  0.336  ""
 6         6  0.000957  0.456  ""
 7         7 -0.00183   0.119  ""
 8         8  0.000842  0.555  ""
 9         9 -0.00100   0.522  ""
10        10 -0.00228   0.0492 "**"
```

Listing 6.8 BPR に基づくポートフォリオ・ソート（時価総額加重の場合）

```
# 中盤で保有比率 w と月次超過リターン Re を計算している箇所を除けば,ch06_15a と
 ↪ 全く同じ
annual_data <- annual_data |>
  mutate(
    lagged_BEME = lagged_BE / lagged_ME
    ) |>
  mutate(
    BEME_rank10 = as.factor(ntile(lagged_BEME, 10)),
    .by = year
    )

BEME_sorted_portfolio <- annual_data |>
  select(year, firm_ID, BEME_rank10, lagged_ME) |>
  full_join(monthly_data, by = c("year", "firm_ID")) |>
  drop_na() |>
  mutate(
    # 各ポートフォリオで保有比率を計算
    w = lagged_ME / sum(lagged_ME),
    .by = c(month_ID, BEME_rank10)
  ) |>
  summarize(
    # 時価総額加重の月次超過リターンを計算
    Re = sum(w * Re),
    .by = c(month_ID, BEME_rank10)
  )

BEME_sorted_portfolio |>
  summarize(
    mean_Re = mean(Re),
    .by = BEME_rank10
  ) |>
  ggplot() +
  geom_col(aes(x = BEME_rank10, y = mean_Re)) +
  geom_hline(yintercept = 0) +
  labs(x = "BE/ME Rank", y = "Mean Monthly Excess Return") +
  scale_y_continuous(limits = c(-0.005, 0.02)) + mystyle
```

Listing 6.9 CAPM の実証的な検証（1）

```
# 推定結果を保存するために空のリストを準備
CAPM_results <- vector("list", 10)

for(i in 1:10){

  CAPM_results[[i]] <- ME_sorted_portfolio |>
    filter(ME_rank10 == i) |>
    lm(Re ~ R_Me, data = _) |>
    tidy() |>
    mutate(ME_rank10 = i) |> # 推定対象のポートフォリオ名を保存
    select(ME_rank10, everything()) # ME_rank10 を第一列に移動
}
```

Listing 6.10 CAPM の実証的な検証（2）

```
# リストを一つのデータフレームにまとめる
CAPM_results <- do.call(rbind, CAPM_results)
```

Listing 6.11 グループごとの線形回帰（1）lapply() 関数を使う場合

```
ME_sorted_portfolio_splitted <- split(ME_sorted_portfolio,
 ↪  ME_sorted_portfolio$ME_rank10) # 元データを ME_rank10 の値に応じて十個
 ↪  のデータフレームに分割

estimate_CAPM <- function(return_data) { # リターン・データを受け取り,
 ↪  CAPM の推定結果をデータフレームで返す関数を準備
  lm_results <- lm(Re ~ R_Me, data = return_data)
  tidied_lm_results <- tidy(lm_results)
}

CAPM_results_by_lapply <- lapply(ME_sorted_portfolio_splitted,
 ↪  estimate_CAPM)
# lapply() 関数は第一引数にリスト，第二引数に関数を取る
# lapply の返り値はリストなので，一つのデータフレームにまとめたい場合は
 ↪  do.call() 関数を用いる
```

Listing 6.12 グループごとの線形回帰（2）map() 関数を使う場合

```
ME_sorted_portfolio |>
  nest(.by = ME_rank10) |> #【修正 1】ここでグループ化を指定して畳み込む
  mutate(
    # map() 関数を用いて各グループを線形回帰
    CAPM_regression = map(data, ~lm(Re ~ R_Me, data = .x)),
    CAPM_summary = map(CAPM_regression, tidy)
    #【修正 2】ここは既にランクごとの行になっているので .by は不要
    ) |>
  select(-c(data, CAPM_regression)) |>
  unnest(cols = CAPM_summary)
```

Listing 6.13 CAPM アルファの可視化

```
CAPM_results |>
  filter(term == "(Intercept)") |> # 定数項に関する推定結果のみを抽出
  mutate(
    # ME_rank10 を整数型からファクター型に
    ME_rank10 = as.factor(ME_rank10)
  ) |>
  # 横軸を ME_rank10，縦軸を CAPM_alpha とする棒グラフ
  ggplot() + aes(x = ME_rank10, y = estimate) +
  geom_col() + # 棒グラフ
  geom_hline(yintercept = 0) + # y = 0 の直線を追加
  labs(x = "ME Rank", y = "CAPM alpha") +
  scale_y_continuous(limits = c(-0.003, 0.013)) +
  mystyle
```

Listing 6.14 CAPM アルファの統計的な有意性を評価

```
CAPM_results |>
  filter(term == "(Intercept)") |> # 定数項を抽出
  rename(
    # 列名を変更
    CAPM_alpha = estimate,
    p_value = p.value
  ) |>
  mutate(
    # 統計的に有意な結果を*で強調
    significance = cut(p_value,
                       breaks = c(0, 0.01, 0.05, 0.1, 1),
                       labels = c("***", "**", "*", ""),
                       include.lowest = TRUE)
  ) |>
  select(# 出力したい列を指定
    ME_rank10, CAPM_alpha, p_value, significance
  )
```

Listing 6.15 証券市場線の推定

```
ME_cross_sectional_return <- CAPM_results |>
  filter(term == "R_Me") |> # R_Me の係数を抽出
  rename(CAPM_beta = estimate) |> # 名称変更
  select(ME_rank10, CAPM_beta) |> # 変数を選択
  mutate( # ファクター型に変換
    ME_rank10 = as.factor(ME_rank10)
  ) |>
  # 超過リターンのデータと結合
  full_join(
    ME_cross_sectional_return, ., by = "ME_rank10"
  )

# 平均超過リターンと平均市場ポートフォリオ超過リターンを計算
mean_R_Me <- mean(factor_data$R_Me)

ggplot(ME_cross_sectional_return) +
  aes(x = CAPM_beta, y = mean_Re) +
  geom_point() + # 散布図
  geom_abline(intercept = 0, slope = mean_R_Me) + # 証券市場線
  labs(x = "市場 beta", y = "平均超過リターン") + #
  scale_x_continuous(limits = c(0, 1.2), expand = c(0, 0)) +
  scale_y_continuous(limits = c(0, 0.02)) + mystyle
```

Listing 6.16 前年度の時価総額に基づくランク付け

```
annual_data <- annual_data |>
  mutate(
    # lagged_BEME が欠損している場合は欠損扱いに
    lagged_ME = replace(lagged_ME, is.na(lagged_BEME), NA)
  ) |>
  mutate(
    .by = year,
    ME_rank2 = as.factor(ntile(lagged_ME, 2))
  )
```

Listing 6.17 簿価時価比率に基づくランク付け（1）

```
annual_data |>
  mutate(
    # 年度ごとに簿価時価比率のパーセンタイル順位を計算
    BEME_percent_rank = percent_rank(lagged_BEME),
    .by = year
  )
```

Listing 6.18 簿価時価比率に基づくランク付け（2）

```
annual_data <- annual_data |>
  mutate(
    # 年度ごとに簿価時価比率のパーセンタイル順位を計算
    BEME_percent_rank = percent_rank(lagged_BEME),
    .by = year
  ) |>
  mutate(
    # BEME_percent_rank の値に応じて 1 から 3 まで BEME_rank3 の値を定義
    BEME_rank3 = cut(BEME_percent_rank,
                     breaks = c(0, 0.3, 0.7, 1),
                     labels = c(1, 2, 3),
                     include.lowest = TRUE)
  )
```

Listing 6.19 Size-BE/ME ポートフォリオへの分類（1）

```
annual_data <- annual_data |>
  mutate(
    # ME_rank2 と BEME_rank3 の組合せで，ファクター型の変数
    ↪ FF_portfolio_type を定義
    FF_portfolio_type = interaction(ME_rank2, BEME_rank3)
    )
```

Listing 6.20 Size-BE/ME ポートフォリオへの分類（2）

```
annual_data <- annual_data |>
  mutate(
    # ファクター型の変数
    FF_portfolio_type = fct_recode(FF_portfolio_type,
                                   SL = "1.1",
                                   BL = "2.1",
                                   SN = "1.2",
                                   BN = "2.2",
                                   SH = "1.3",
                                   BH = "2.3")
  )
```

Listing 6.21 Size-BE/ME ポートフォリオへの分類（3）

```
annual_data |>
  summarize(FF_portfolio_type = FF_portfolio_type[1],
            mean_BEME = mean(lagged_BEME),
            mean_ME = mean(lagged_ME),
            mean_N_stocks = n() / length(unique(year)),
            .by = c(ME_rank2, BEME_rank3)
   ) |>
  drop_na() # 欠損データを削除
```

Listing 6.22 Size-BE/ME ポートフォリオの構築（1）

```
annual_data <- annual_data |>
  # year と FF_portfolio_type のペアでグループ化
  mutate(
    # 各ポートフォリオ内で時価総額加重の保有比率を計算
    w = lagged_ME  / sum(lagged_ME, na.rm = TRUE),
    .by = c(year, FF_portfolio_type)
  )
```

Listing 6.23 Size-BE/ME ポートフォリオの構築（2）

```
FF_portfolio <- annual_data |>
  select(
    year, firm_ID, FF_portfolio_type, ME_rank2, BEME_rank3, w
  ) |>
  full_join(
    monthly_data,
    by = c("year", "firm_ID")
  ) |> # 今までに準備したデータと月次データを結合
  summarize(
    ME_rank2 = ME_rank2[1],
    BEME_rank3 = BEME_rank3[1],
    R = sum(w * R, na.rm = TRUE), # 各ポートフォリオの月次リターンを計算
    R_F = R_F[1],
    .by = c(month_ID, FF_portfolio_type)
  ) |>
  drop_na() # 欠損データを削除
```

Listing 6.24 Size-BE/ME ポートフォリオのリターンの可視化（1）

```
FF_portfolio_mean_return <- FF_portfolio |>
  mutate(Re = R - R_F) |>
  summarize(
    ME_rank2 = ME_rank2[1],
    BEME_rank3 = BEME_rank3[1],
    mean_Re = mean(Re),
    .by = FF_portfolio_type
    ) # 各ポートフォリオの超過リターンの平均値を計算


ggplot(FF_portfolio_mean_return) +
  geom_col(aes(x = BEME_rank3, y = mean_Re, fill = ME_rank2), position
  ↪  = "dodge") + # x 軸を BEME_rank3，y 軸を mean_Re に，ME_rank2 のサブ
  ↪  グループで色分け
  scale_fill_grey() + # 棒グラフの色をモノトーンに
  labs(x = "BE/ME Rank", y = "Mean Monthly Excess Return", fill = "ME
  ↪  Rank") +
  scale_y_continuous(expand = c(0, 0)) + mystyle
```

Listing 6.25 Size-BE/ME ポートフォリオのリターンの可視化（2）

```
ggplot(FF_portfolio_mean_return) +
  geom_col(aes(x = BEME_rank3, y = mean_Re, fill = ME_rank2), position
   ↪ = "dodge") +
  scale_fill_grey() +
  geom_text(aes(x = BEME_rank3, y = mean_Re, group = ME_rank2, label =
   ↪ FF_portfolio_type), # (x, y) 座標を指定して各ポートフォリオの名前をグ
   ↪ ラフに挿入
            vjust = -0.5, # 棒グラフが重ならないよう文字ラベルを上にずらす
            position = position_dodge(width = 0.9)) + # ME_rank2 のサブ
             ↪ グループで文字ラベルが左右にずれるよう調整
  labs(x = "BE/ME Rank", y = "Mean Monthly Excess Return", fill = "ME
   ↪ Rank") +
  scale_y_continuous(expand = c(0, 0), limits = c(0, 0.015)) + # 文字ラ
   ↪ ベルがはみ出ないよう y 軸の範囲を指定
  theme_classic()
```

Listing 6.26 Size-BE/ME ポートフォリオのリターンの可視化（3）

```
initial_point <- tibble(
  month_ID = c(12, 12), # 累積リターンの起点を定義
  cumulative_gross_R = c(1, 1),
  FF_portfolio_type = c("BL", "SH")
  )

FF_portfolio_cumulative_return <- FF_portfolio |>
  mutate(# グロス・リターンを累積
    cumulative_gross_R = cumprod(1 + R),
    .by = FF_portfolio_type
  ) |>
  filter(FF_portfolio_type %in% c("BL", "SH")) |>
  select(month_ID, cumulative_gross_R, FF_portfolio_type) |>
  bind_rows(initial_point)

ggplot(FF_portfolio_cumulative_return) +
  geom_line(aes(x = month_ID, y = cumulative_gross_R, linetype =
  ↪  FF_portfolio_type)) +
  scale_linetype_manual(values = c("longdash", "solid")) +
  geom_hline(yintercept = 1, linetype = "dotted") +
  labs(x = "Month ID", y = "Cumulative Gross Return", linetype = "") +
  scale_x_continuous(expand = c(0, 0)) + mystyle
```

Listing 6.27 SMB と HML の構築（1）

```
# FF_portfolio_type の値に基づく列を作成し，縦長から横長のデータに変換
FF_portfolio <- FF_portfolio |>
  pivot_wider(
    id_cols = month_ID,
    names_from = FF_portfolio_type,
    values_from = R
    )
```

Listing 6.28 SMB と HML の構築（2）

```
factor_data <- FF_portfolio |>
  mutate(
    SMB = (SH + SN + SL) / 3 - (BH + BN + BL) / 3, # SMB と HML を計算
    HML = (SH + BH) / 2 - (SL + BL) / 2
    ) |>
  select(month_ID, SMB, HML) |>
  full_join(factor_data, by = "month_ID") |> # 3 ファクターの実現値を
   ↪ factor_data に集約
  select(-c("SMB", "HML"), c("SMB", "HML")) # SMB と HML を最後列に移動
```

Listing 6.29 FF3 モデルの推定

```
ME_sorted_portfolio <- ME_sorted_portfolio |>
  select(-c(R_Me, R_M)) |>
  # 3 ファクターの実現値を ME_sorted_portfolio に追加
  full_join(factor_data, by = "month_ID")

FF3_results <- list(NA)  # 推定結果を保存するために空のリストを準備

for(i in 1:10) {
  FF3_results[[i]] <- ME_sorted_portfolio |>
    filter(ME_rank10 == i) |>
    lm(Re ~ R_Me + SMB + HML, data = _) |> # 3 ファクターの実現値を独立変
     ↪ 数として重回帰
    tidy() |>
    mutate(ME_rank10 = i) |> # 推定対象のポートフォリオ名を保存
    select(ME_rank10, everything()) # ME_rank10 を第一列に移動
}

FF3_results <- do.call(rbind, FF3_results) # do.call() 関数を用いて複数の
 ↪ データフレームから構成されるリストを一つのデータフレームに統合
```

Listing 6.30 FF3 アルファの可視化

```
FF3_results |>
  filter(term == "(Intercept)") |> # 定数項に関する推定結果のみを抽出
  mutate(
    ME_rank10 = as.factor(ME_rank10)
  ) |> # ME_rank10 を整数型からファクター型に
  ggplot() +
  # 横軸を ME_rank10, 縦軸を FF3_alpha とする棒グラフ
  geom_col(aes(x = ME_rank10, y = estimate)) +
  geom_hline(yintercept = 0) +
  labs(x = "ME Rank", y = "FF3 alpha") +
  scale_y_continuous(limits = c(-0.003, 0.013)) + mystyle
```

Listing 6.31 FF3 アルファの統計的な有意性を評価

```
FF3_results |>
  filter(term == "(Intercept)") |> # 定数項に関する推定結果のみを抽出
  rename(
    FF3_alpha = estimate,
    p_value = p.value
  ) |> # 列名を変更
  mutate(
    significance = cut(p_value,
      breaks = c(0, 0.01, 0.05, 0.1, 1),
      labels = c("***", "**", "*", ""),
      include.lowest = TRUE)
  ) |> # 統計的に有意な結果を*で強調
  select(ME_rank10, FF3_alpha, p_value, significance)
```

Listing 6.32 データの保存

```
write_csv(factor_data, "data/ch06_output.csv")
```