データ分析と可視化

本編

愛媛大学　　松浦 真也

Ver. 2024.1.10

本資料について

本資料の著作権は放棄していませんが、教育機関において、教育目的で本資料をご活用いただくことは、歓迎致します。

- 生徒・学生の皆さんが宿題やレポート等に本資料を活用する際は、引用元の不記載など、引用の要件を満たさないコピペは厳に慎んで下さい。
- 記載内容の厳密性については、保証しかねます。非専門家向けに、平易で簡潔な説明を行うことを目的としていますので、厳密な説明は意図していません。
- 本資料内で例示しているチャート（データの図表・グラフ）は、プログラミング言語のPython（主としてPlotlyライブラリ）を用い、オリジナルに作成しました。あくまで例示が目的であり、チャートの中身が正確とは限りません。
- 本資料の利用に伴う損害や不利益については、一切の責任を負いかねます。
- 本資料の一部または全部を、予告なく削除したり、修正したりする可能性があります。

はじめに

資料作成時、他人のグラフをコピペしていませんか？

確かに

- 数値データは、客観的事実を単純に記録しただけなら、著作権の対象外
- 著作権対象外のデータを一般的で単純な手法により描画しただけのグラフも、著作権対象外と思われる（裁判所の判例あり）
- たとえ著作権が存在しても、一定の条件を満たせば、適切に引用することは可能

しかし

- 世の中は、間違いのあるグラフや、不適切・不完全なグラフであふれかえっている
- 分析の目的や資料の性質によって、適切な可視化手法が違うので、ある目的において完璧に描かれたグラフであっても、目的が変われば、不適切・不完全なグラフになる
- 著作権の基準を詳細に理解し、権利侵害の判断を正確に行うのは困難

グラフや図表は、自分で書くのが基本！！

チャート vs グラフ

チャート (chart)

- "a diagram, lists of figures, etc. that shows information" (Oxford Learner's Dictionaries)
- グラフ以外の図表も含む
  - チャートの方が、グラフより広い概念
- 例：円グラフ (pie chart)
  - 英語では、pie graph より pie chart と呼ぶ方が一般的

グラフ (Graph)

- "a diagram consisting of a line or lines, showing how two or more sets of numbers are related to each other" (Oxford Learner's Dictionaries)
- チャートの中でも、特に数値の関係性を線を用いて表したもの
- 例：折れ線グラフ (line graph)
  - line chart とも呼ぶ

チャートの種類

あなたは、何種類のチャート（グラフを含む）を
知っていますか？

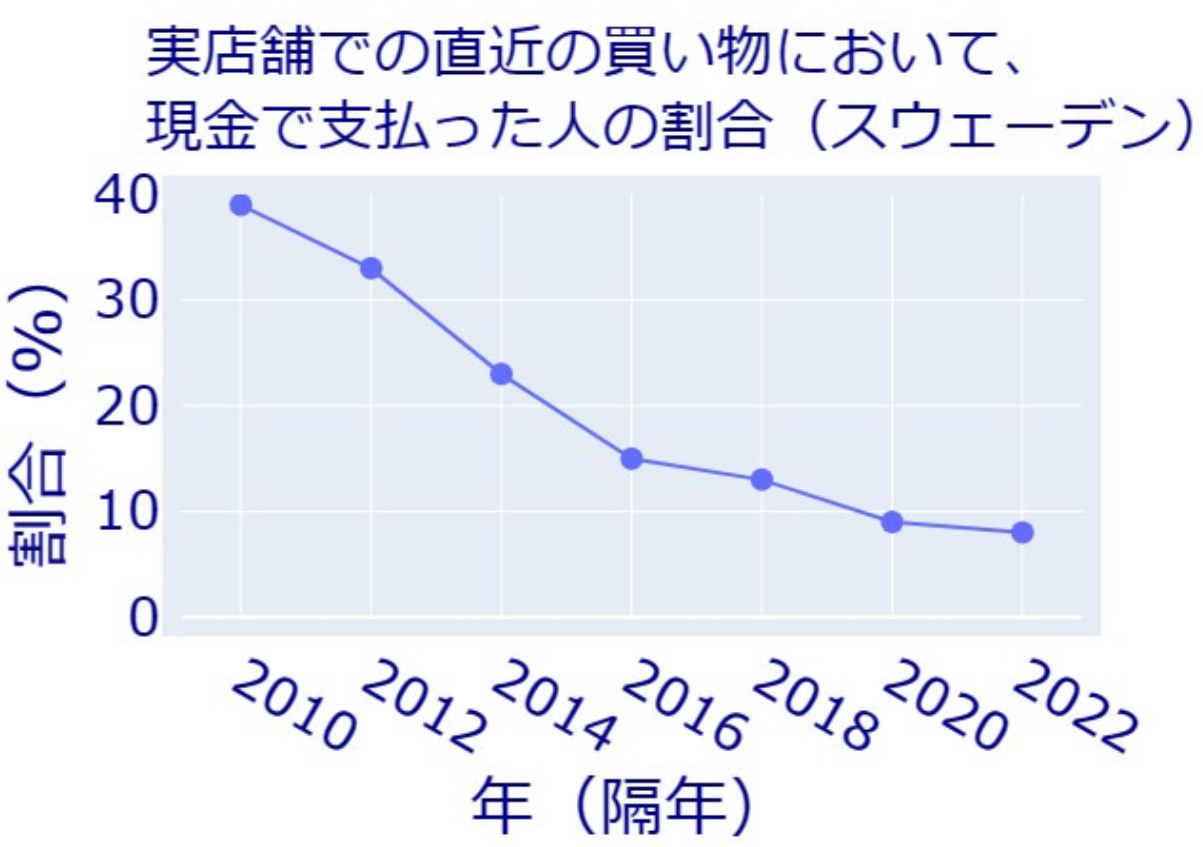
○○チャート、○○グラフという名称のものだけでなく、
○○図、○○マップなども、チャートの一種です。

チャートの種類

とりあえず、この資料では30種類を扱う（この他にも多数あり）

1. 折れ線グラフ
2. 面グラフ
3. 積み上げ面グラフ
4. 棒グラフ
5. 集合棒グラフ
6. 積み上げ棒グラフ
7. 滝グラフ
8. ファンネルチャート
9. 円グラフ
10. ドーナツグラフ
11. サンバーストグラフ
12. ツリーマップ
13. ワッフルチャート
14. 散布図
15. バブルチャート
16. 散布図行列
17. 3次元散布図
18. 三角図
19. ヒストグラム
20. 人口ピラミッド
21. 密度プロット
22. 箱ひげ図
23. バイオリン図
24. レーダーチャート
25. ポーラーチャート
26. デンドログラム
27. 平行座標プロット
28. ヒートマップ
29. 階級区分図
30. 比例シンボルマップ

1. 折れ線グラフ

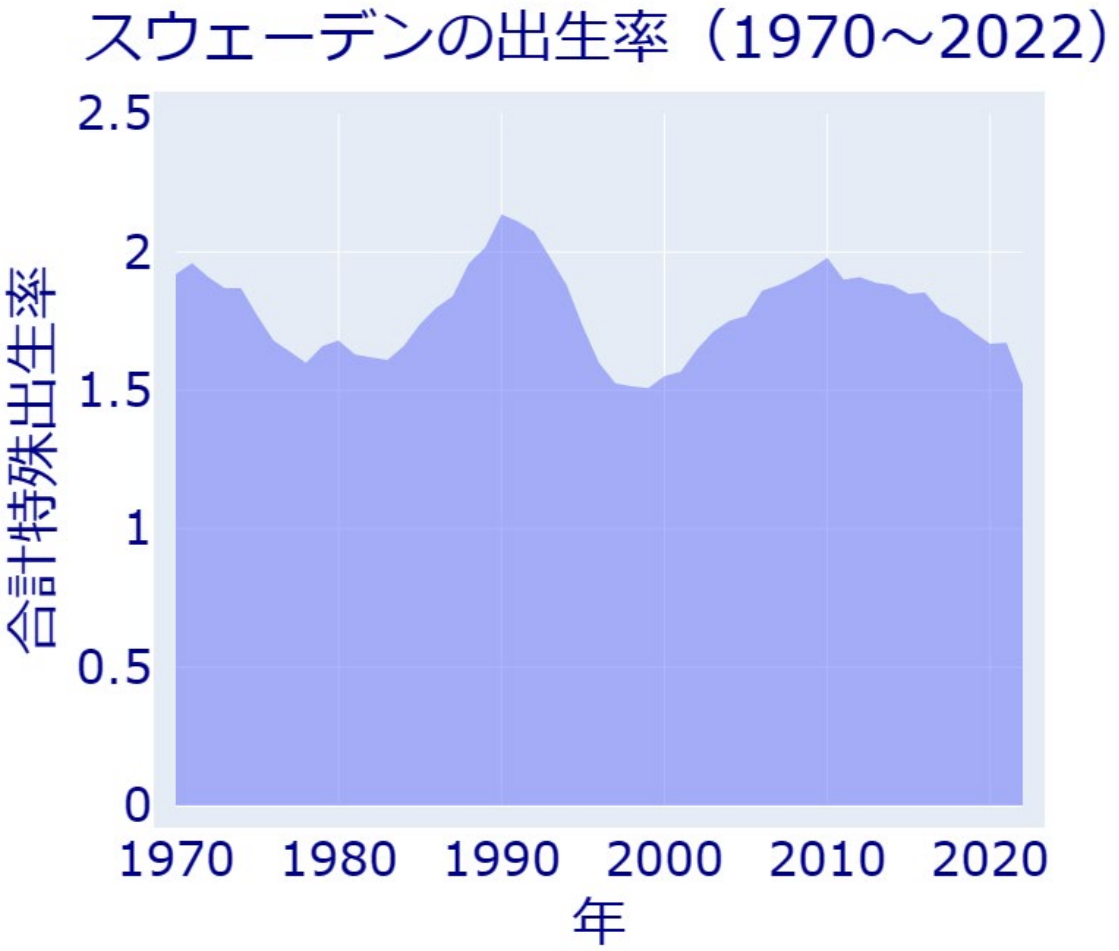


- 折れ線グラフは、主として、時間の経過に伴い値が変動するデータについて、その変動の様子を表現する際に用いる。
- 横軸は時間軸で、縦軸は各時刻におけるデータの値を示す。
- 通常、データは一定間隔で測定されており、グラフにおいて、隣接時刻間は線分で結ぶ。
- 横軸は、時間以外にも、空間的な距離などに対応させることも可能。

データの出典：
[1] Riksbanken（スウェーデン中央銀行），*Betalningsrapport 2022*（支払い報告書 2022），2022年12月15日発行

※ 2010～2022年の隔年のデータ。
※ スウェーデンではキャッシュレス化が進んでいる。

2. 面グラフ



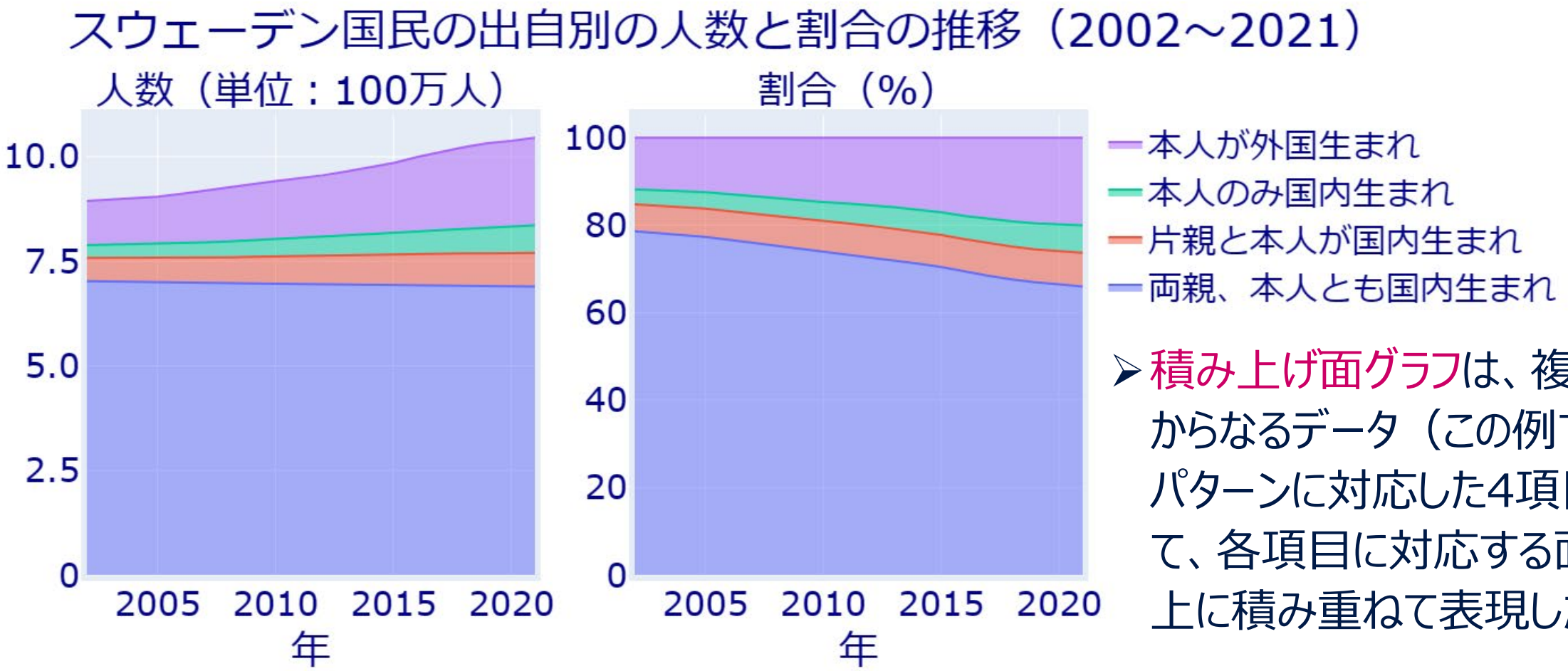
- 面グラフは、折れ線グラフの下側を塗りつぶしたもの。塗りつぶした面積が視覚的に重要な役割を果たすので、縦軸の目盛りは、0から始める必要がある。
- 左の面グラフは、スウェーデンの合計特殊出生率の推移（1970年～2022年）を表す。

※ 合計特殊出生率は、大雑把には、女性1人が生涯で生む子供の数の平均値

※ スウェーデンでは、ほとんどの年で、合計特殊出生率が2を下回っている。しかし、移民の影響で、人口は増加し続けている。

データの出典：[2] https://www.scb.se/ SCB（スウェーデン統計庁），
Antal barn per kvinna efter födelseland 1970–2022 samt framskrivning 2023–2070（出生国別の女性1人当たりの子供の数 1970-2022年の値と2023-2070年の予測），2024年1月7日閲覧

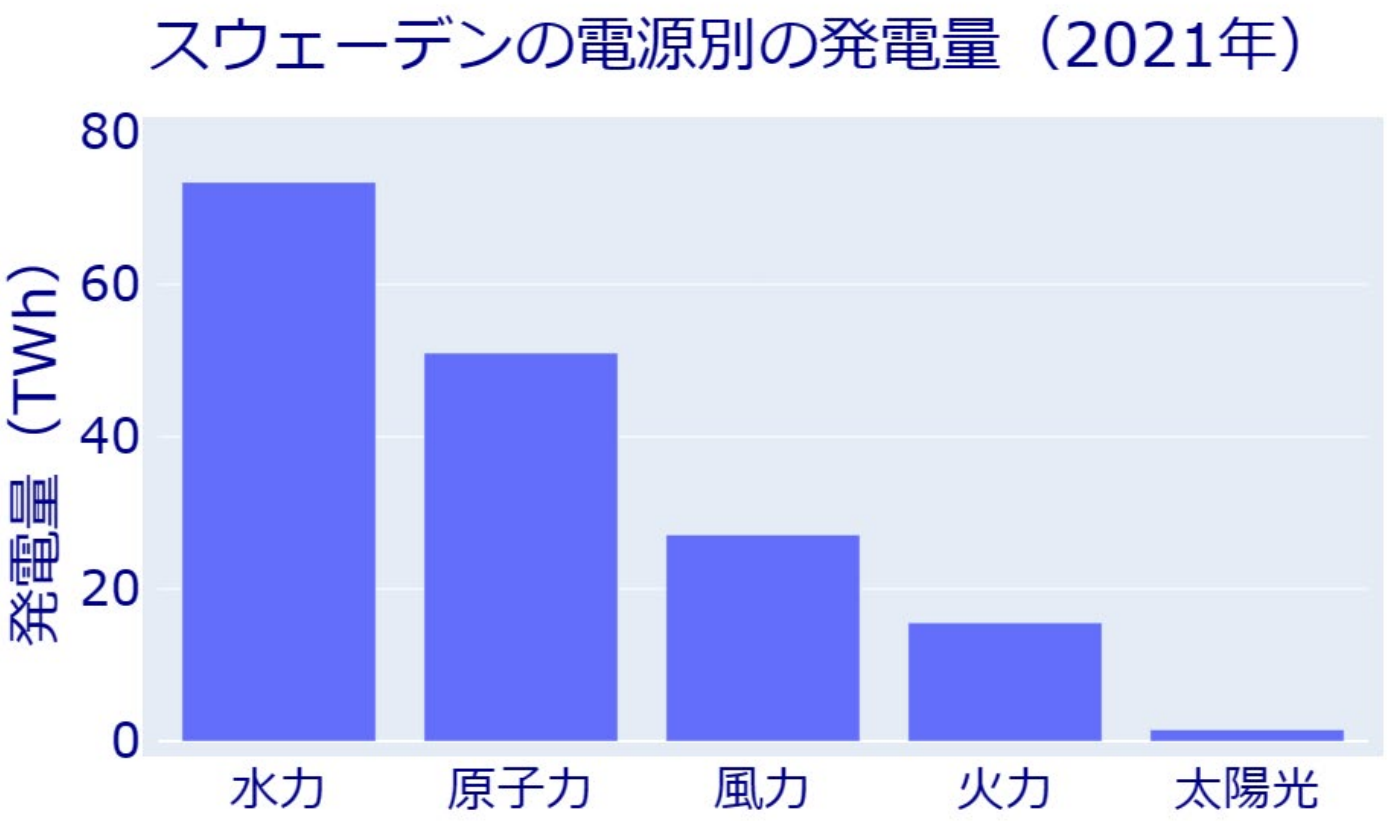
3. 積み上げ面グラフ



- 積み上げ面グラフは、複数の項目からなるデータ（この例では、出自パターンに対応した4項目）について、各項目に対応する面グラフを、上に積み重ねて表現したもの。

データの出典： [3] https://www.scb.se/ SCB（スウェーデン統計庁）, *Antal personer med utländsk eller svensk bakgrund (fin indelning) efter region, ålder och kön*（国外・国内に背景を持つ人々（詳細区分）の地域・年齢・性別ごとの人数）, 2023年1月10日閲覧

4. 棒グラフ



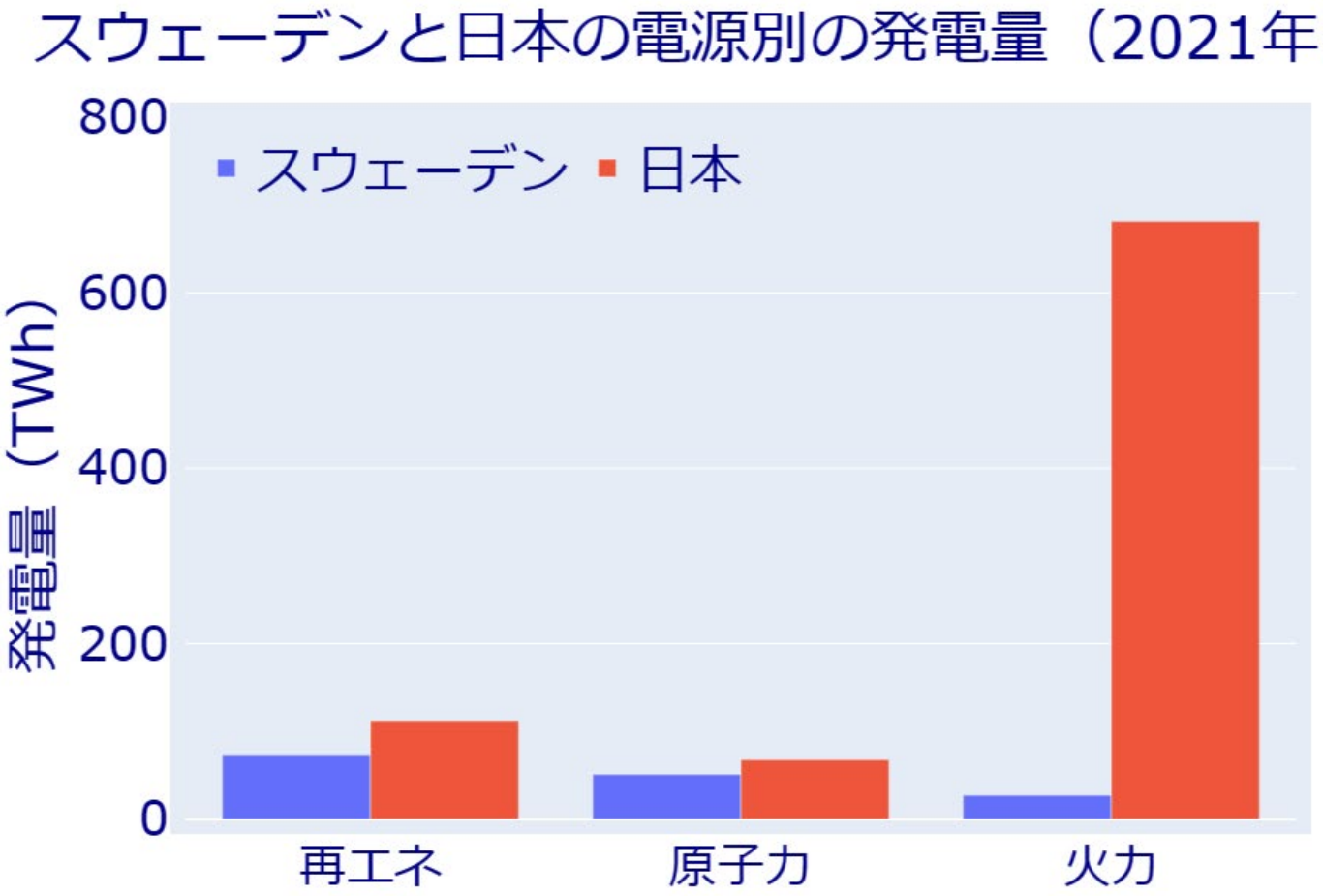
- 棒グラフは、各項目ごとに、データの値を棒の長さで表現したもの。棒の面積が視覚的に重要な役割を果たすので、縦軸の目盛りは、0から始める必要がある。
- 棒は、縦向き、横向きどちらに描くこともできる。縦横どちらが良いかは、一概には言えない。スペースやレイアウトも考慮に入れて選択すると良い。

※ スウェーデンの「火力」は、主に廃熱発電や地域熱供給等、いわゆるコジェネレーション

※ この他、他国からの輸入分もある。一方で、輸出もしている。

データの出典： [4] https://www.scb.se/ SCB（スウェーデン統計庁）, *Eltillförsel i Sverige efter produktionsslag*（スウェーデンの発電種別の電力供給量）, 2024年1月7日閲覧

5. 集合棒グラフ

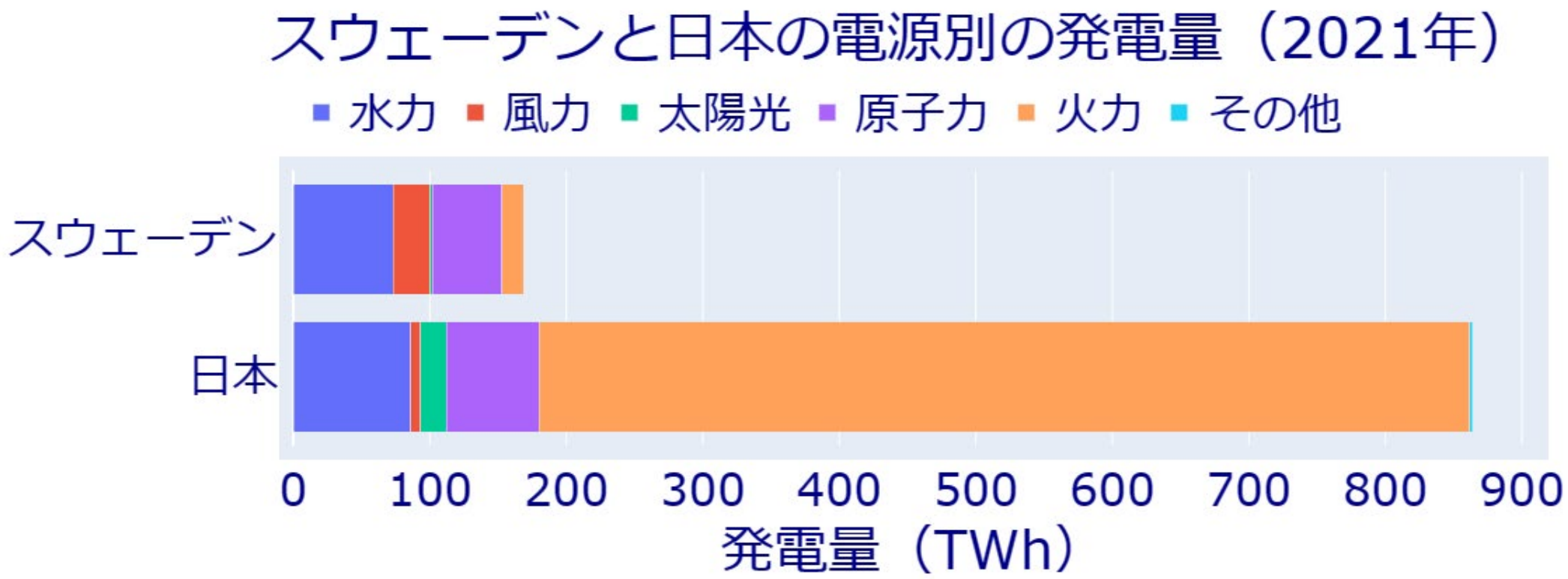


- 集合棒グラフでは、主項目（この例では、再エネ、原子力、火力）ごとに、いくつか（この例では、スウェーデンと日本に対応した2つ）の棒が描かれる。
- 集合棒グラフは、微妙な数値の差（例えば、「再エネ」のスウェーデンと日本の差）も正確に読み取れる。

※ 便宜上、水力と風力と太陽光をまとめて「再エネ」と表記した。

データの出典： 前ページの[4]及び[5] *https://www.enecho.meti.go.jp/* 経済産業省資源エネルギー庁, 2021年度電力調査統計表2-(1)発電実績, 2023年1月6日閲覧

6. 積み上げ棒グラフ

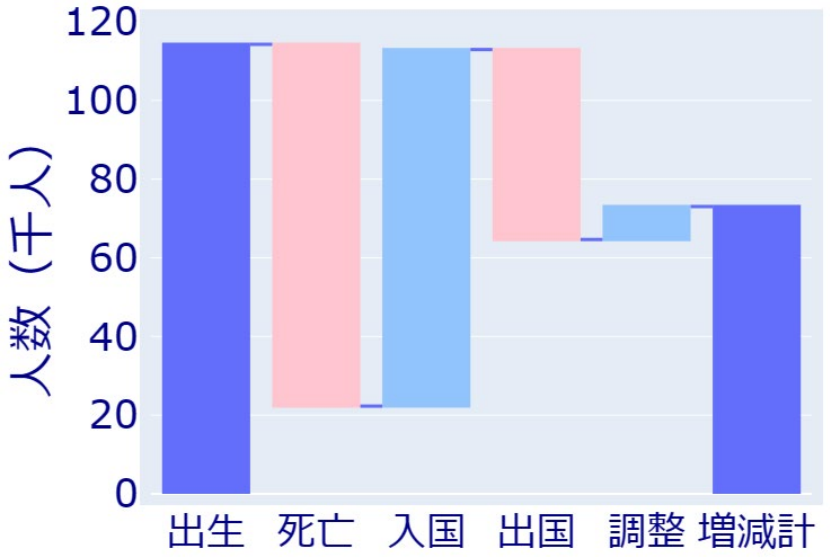


- 積み上げ棒グラフでは、各項目（この例では、水力、風力、太陽光、原子力、火力、その他）の棒グラフを積み上げ、一塊の棒にする。
- 文字通り、縦に積み上げても良いが、この例では、スペースの関係上、横に並べている。

データの出典： 前ページに同じ （SCBと経済産業省資源エネルギー庁）

7. 滝グラフ

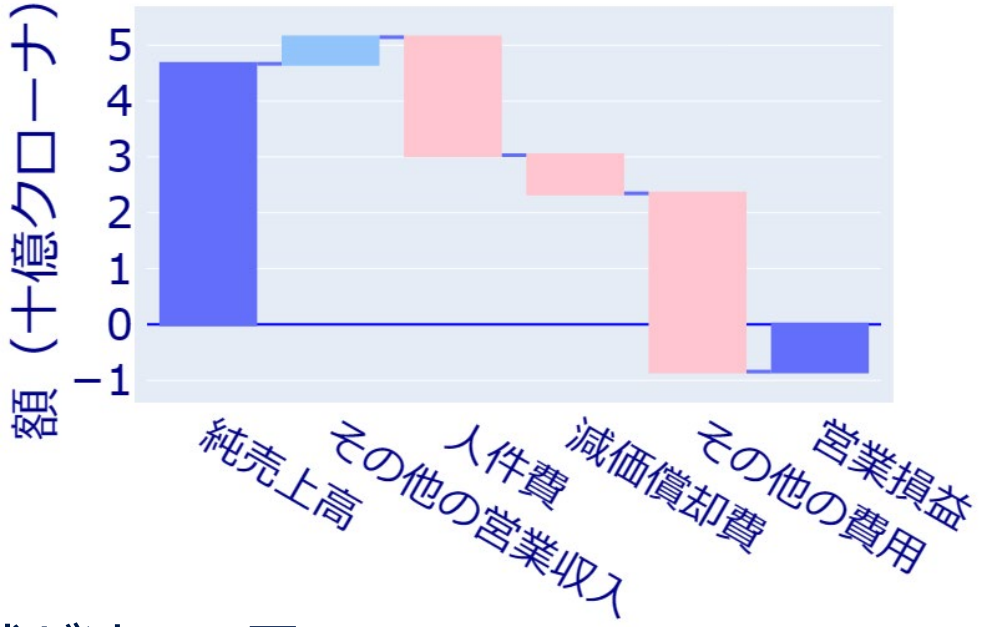
スウェーデンの人口の増減（2021年）



※ 2021年1年間で、スウェーデンの人口は「増減計」に記載の73,031人増加。

※ 「調整」は、2020年までの出来事（出生、死亡、移住）で、2021年に報告された分。

スウェーデン国鉄（親会社）の収支（2021年）

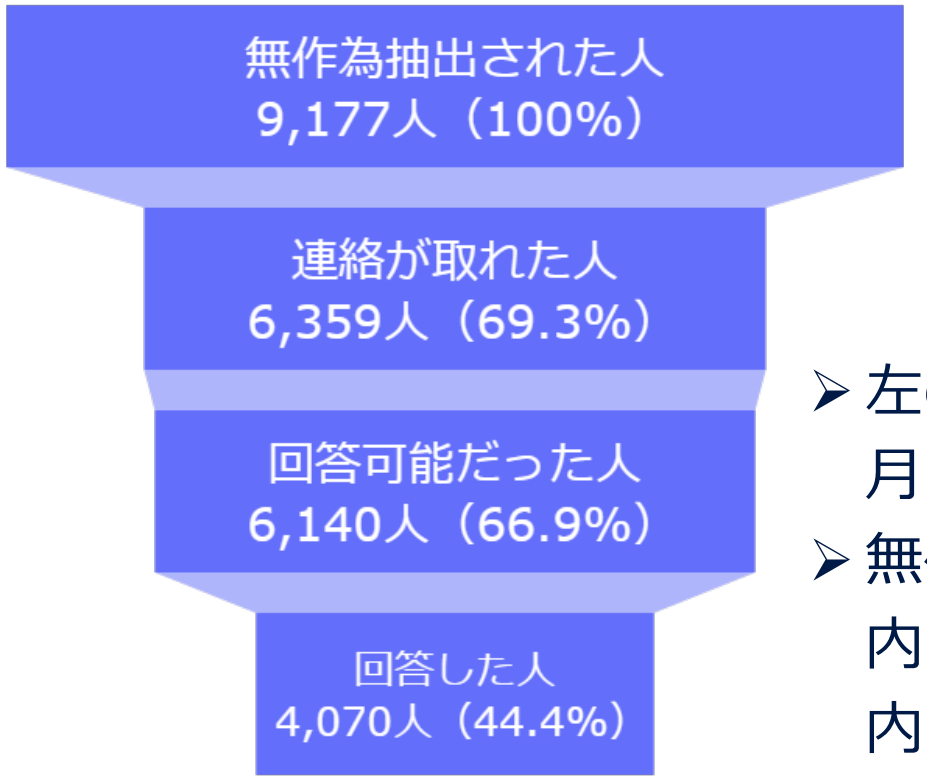


- 滝グラフは、種々の増加、減少要因の合算で全体の増減が決まる量について、左から右に向かい、増加は上方に、減少は下方に棒を伸ばして示したもの。

データの出典：[6] https://www.scb.se/ SCB（スウェーデン統計庁）, *Folkmängd i riket, län och kommuner 31 december och befolkningsförändringar 1 januari - 31 december 2021*（2021年12月31日現在の国、県、市の人口及び2021年1月1日～12月31日の人口変動）, 及び, [7] https://www.sj.se/ SJ（スウェーデン国有鉄道）, *SJs års- och hållbarhetsredovisning 2021*（SJ年報及び持続可能性レポート）, 2023年1月16日閲覧

8. ファンネルチャート

スウェーデン統計庁の世論調査の回答者数

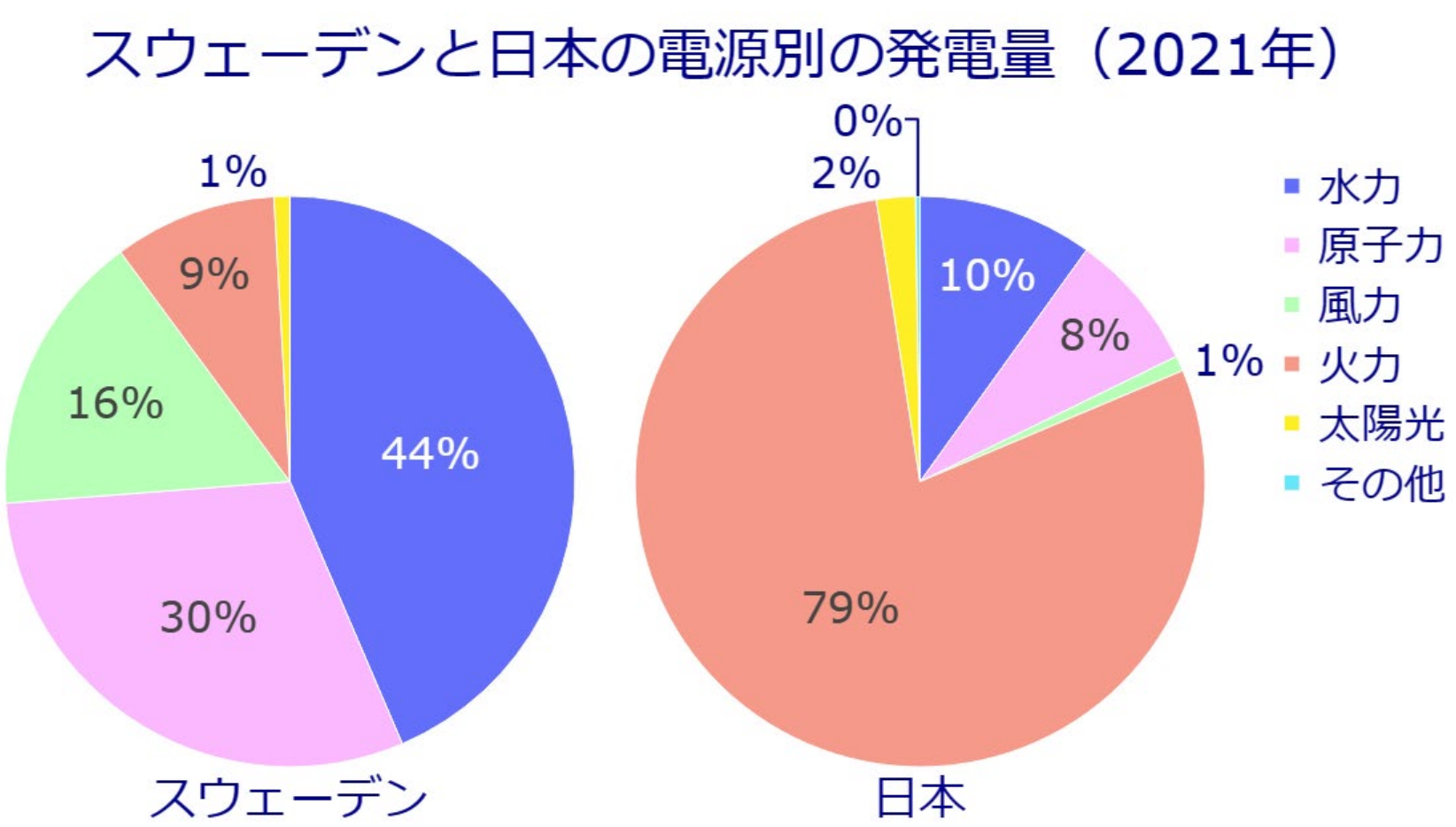


- ファンネルチャートは、値が順次減少していくような階層構造を持ったデータを表現するのに用いられる。上から下に向かって値が減っていくように描くことも、左から右に向かって値が減っていくように描くことも可能。

- 左のチャートは、スウェーデン統計庁が2022年10月27日から11月24日に実施した政党支持率の調査の回答者に関するデータ。
- 無作為（ランダム）に9,177人が、調査対象に選ばれた。その内、固定電話や携帯電話で連絡が取れた人が6,359人。その内、調査に協力できる状態になかった人を除くと、6,140人。その中で、実際に調査に協力し、回答してくれた人は4,070人だった。

データの出典：[8] https://www.scb.se/ SCB（スウェーデン統計庁）, *Partisympatiundersökningen november 2022*（政党支持率調査2022年11月）, 2023年1月18日閲覧

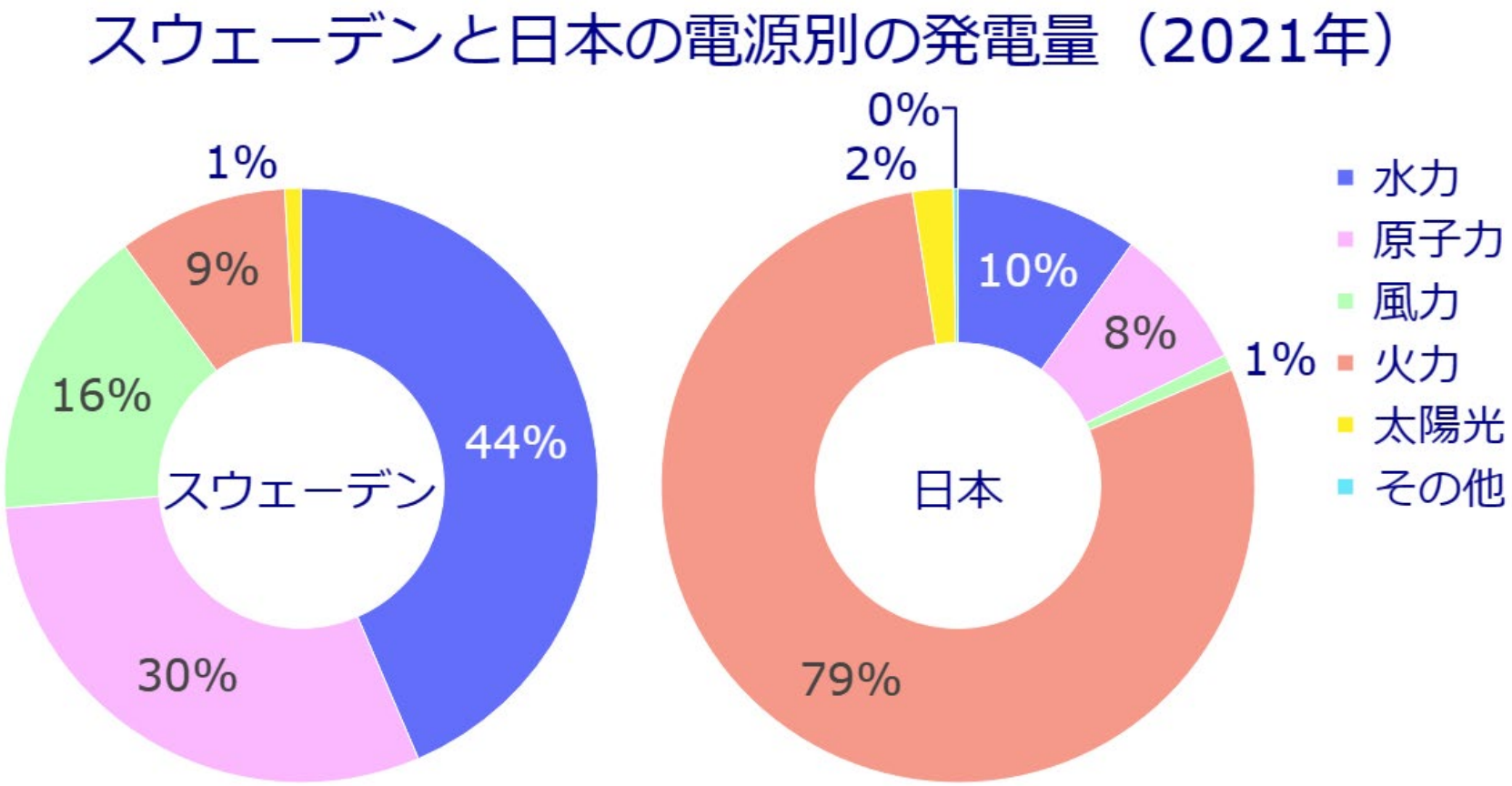
9. 円グラフ



- 円グラフは、各項目ごとに、データの値を扇型の角度で表現したもの。
- 角度の他、面積・弧長でも、各項目の割合を視覚的に把握できる。
- 円グラフだと、各項目間で、割合の大小を正確に比較するのは難しいので、安易に円グラフを多用しない（集合棒グラフの方が正確に比較可）。

データの出典：前述のSCB及び経済産業省資源エネルギー庁のデータ [4],[5] における電源別発電量をもとに、発電割合の概算値を独自に計算（正確な値とは限らない）

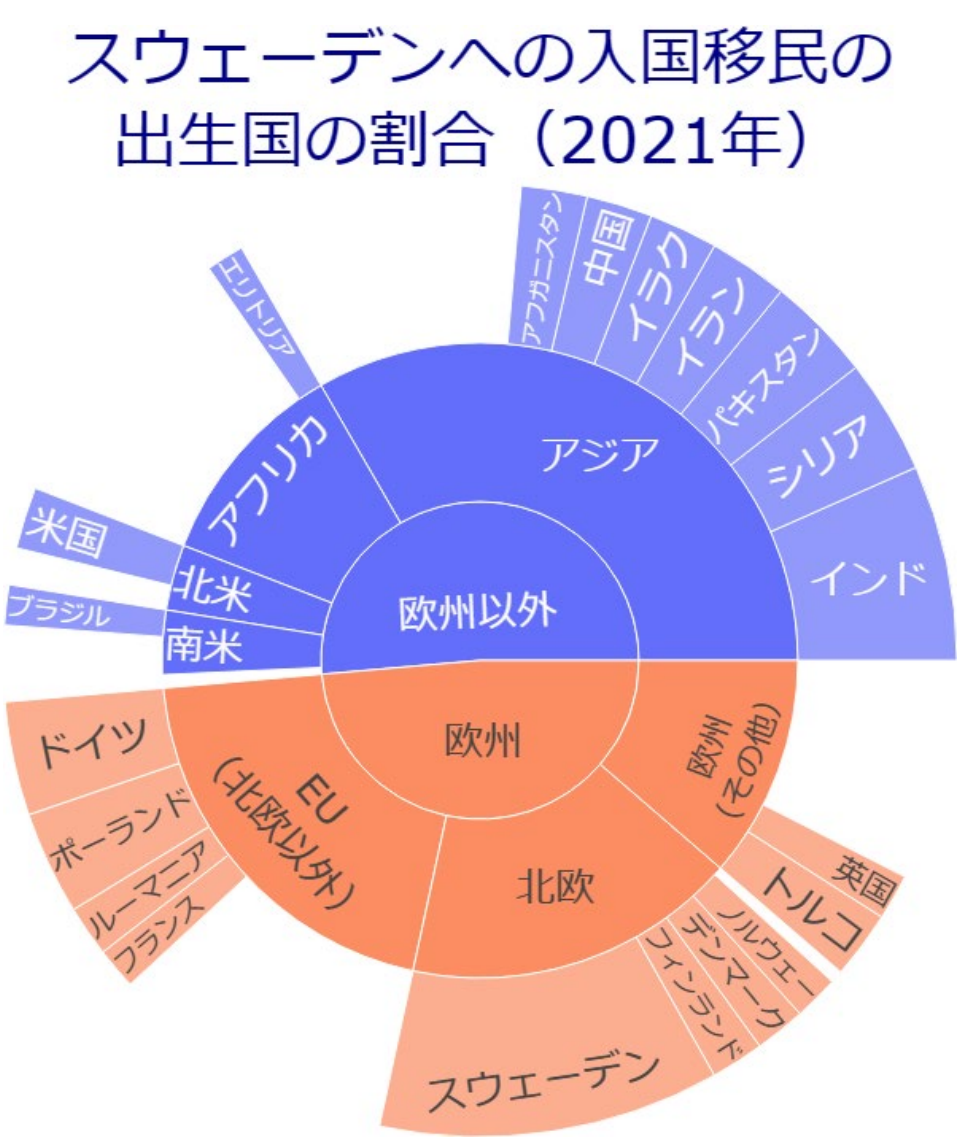
10. ドーナツグラフ



データの出典：前ページと同じ（SCB及び経済産業省資源エネルギー庁のデータをもとに、概算値を独自に算出）

- ドーナツグラフは、円グラフの中心部分をくりぬいたもの。
- 中心部分に説明を書けるので、スペースを有効活用できる。
- 中心の角度が見えなくなるので、円グラフと比べ、各項目の割合が、視覚的に把握しにくくなる。
- レイアウト上、止むを得ない場合を除き、ドーナツ形にしない（中心部分をくりぬかない）方が良い。

11. サンバーストグラフ



※2021年の入国移民は計90,631人

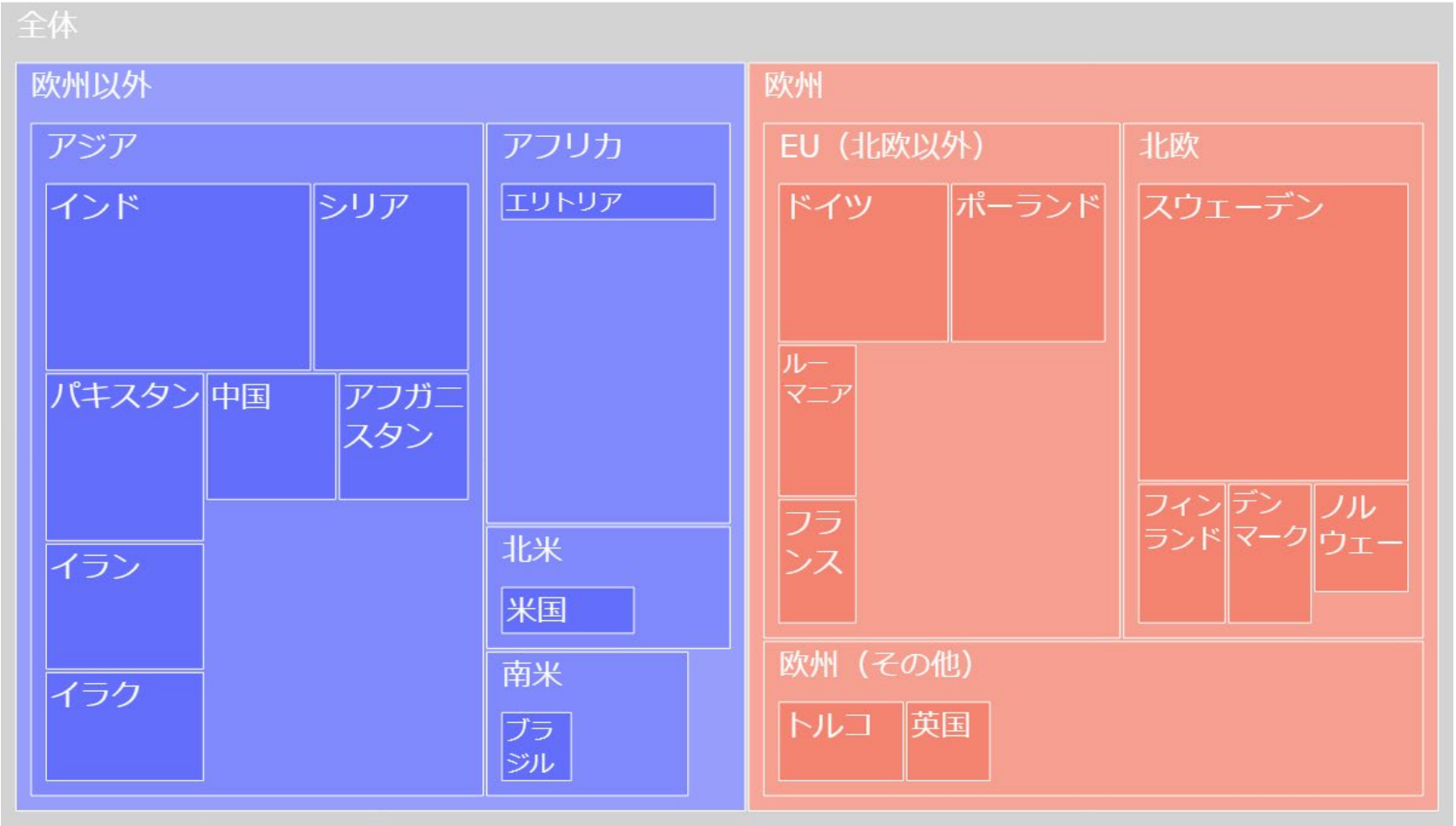
- サンバーストグラフは、円グラフに階層構造を持たせたもの。
- 左の例は、スウェーデンへの移民について、出生国の割合を示したもの。第1層では、欧州とそれ以外に分割。第2層では、欧州については、北欧、北欧以外のEU諸国、その他の欧州諸国に分割。第3層では、国ごとに分割（上位20か国のみ明示）。各層の空白部は、「その他」の意。
- 本来、各国の数値もグラフ上に記入した方が良いが、左の例では、スペースの関係上、省略。プログラミング言語Python等を用いてグラフを描画すれば、パソコン上で、マウスを当該国名に合せると、数値が表示されるようにできる。

データの出典：

[9] https://www.scb.se/SCB（スウェーデン統計庁），*Invandring 2021 efter län, kommun och födelseland*（県、市、出生国別入国移民2021）及び [10] *Invandringar och utvandringar efter födelseland och kön*（出生国別、性別ごとの入国移民），2023年1月13日閲覧

12. ツリーマップ

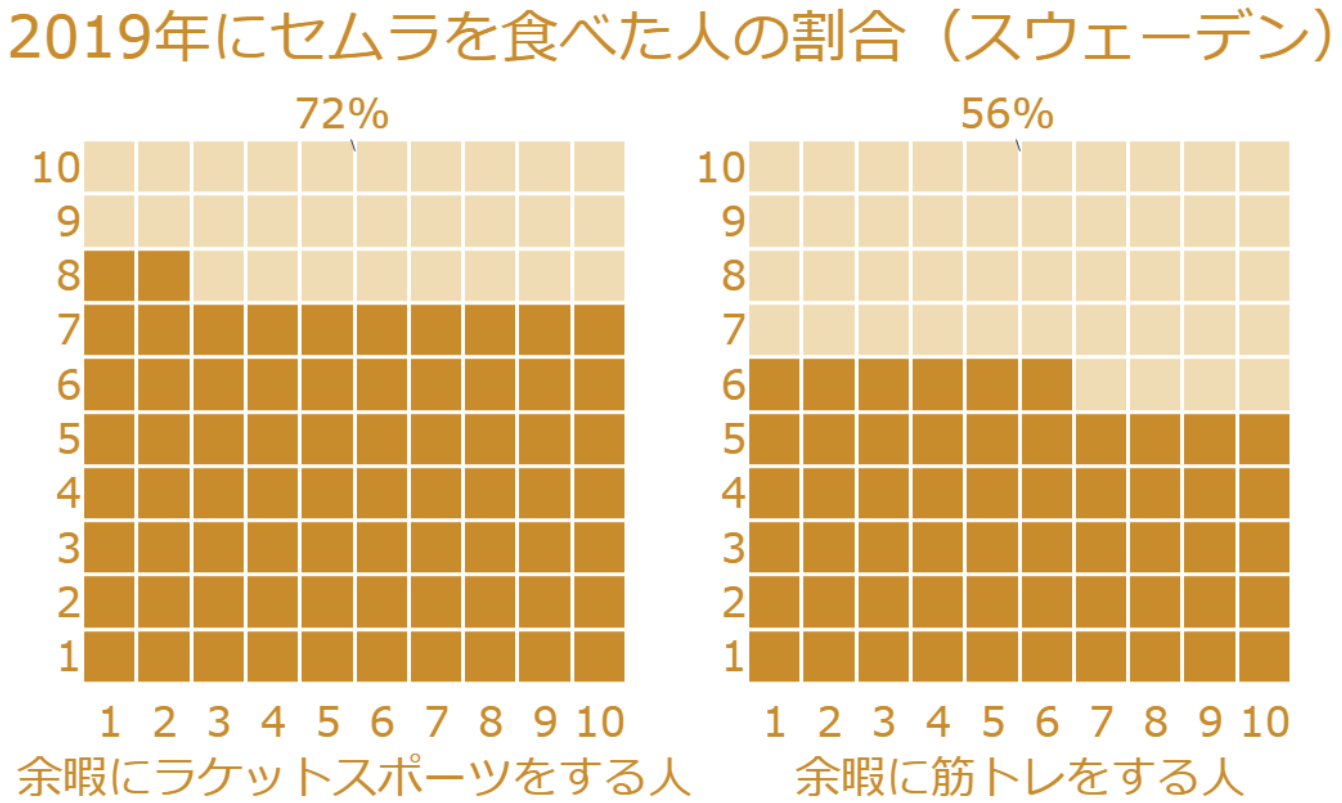
スウェーデンへの入国移民の出生国の割合（2021年）



データの出典：前ページと同じ（SCB）

- ツリーマップは、サンバーストグラフと同様の階層構造を、四角形の包含関係で表現。
- 数値の大きさは、四角形の面積で表現される。
- 面積だけから項目間の数値の大小を正確に比較するのは、必ずしも容易ではない。
- 具体的な数値も図に書き込むと良いが、スペースが足りないのに無理に書き込むと、文字が小さくなりすぎて、判読が難しくなる。

13. ワッフルチャート



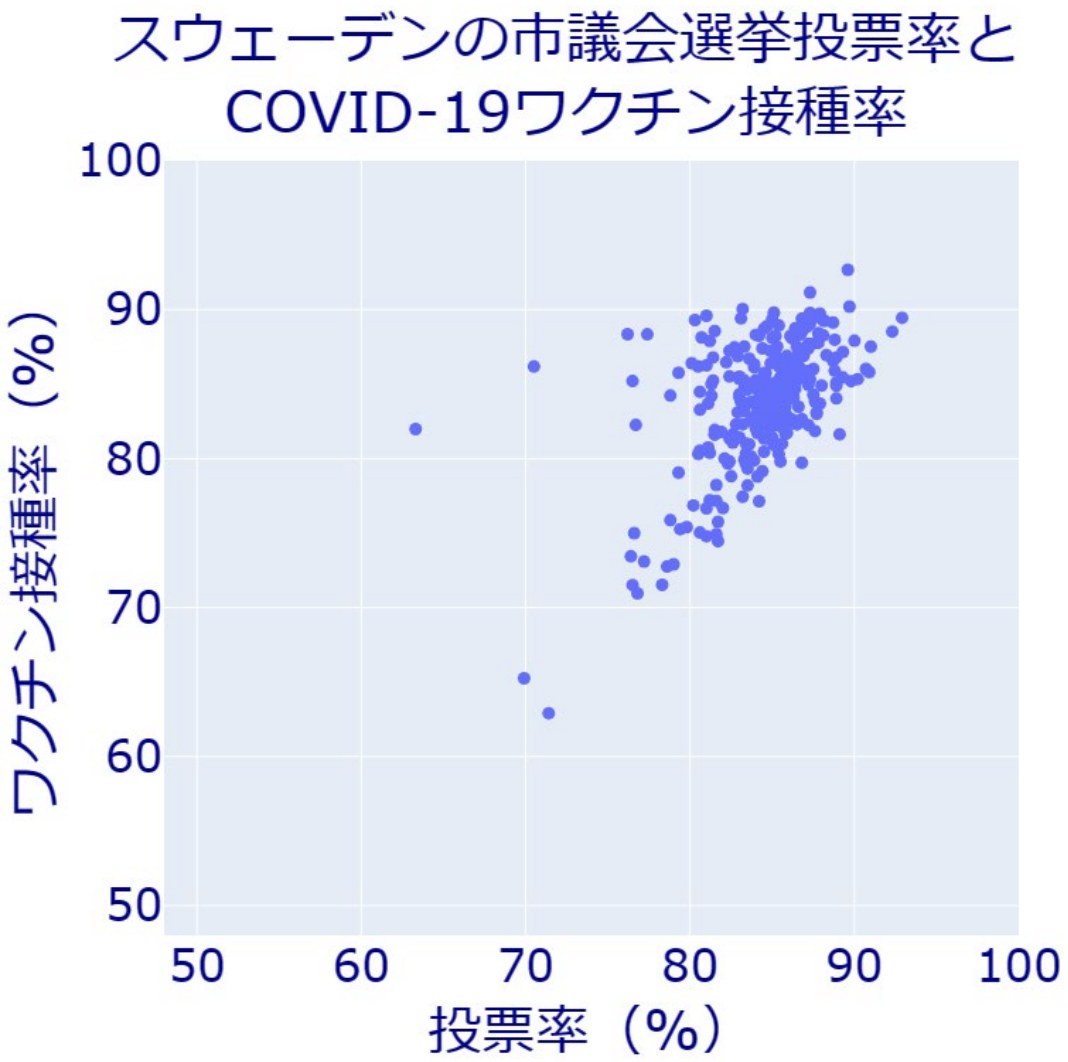
- ワッフルチャートでは、ワッフルの格子模様を塗りつぶすことで、全体に占める各項目の割合を表す。
- 円グラフ（Pie chart パイチャート）がパイを切り分けたピースで、割合を示すのと同様。

余暇にラケットスポーツをする人のうち、72%の人が2019年にセムラを食べた。
一方、余暇に筋トレをする人のうち、56%の人が、2019年にセムラを食べた。

セムラはスウェーデンの伝統的スイーツ（筆者撮影）

データの出典：[11] https://fettisdagenwww.bageri.se/press/pressmeddelanden/ Sveriges bagare & konditorer AB（スウェーデン・パン菓子職人株式会社）記者発表, *Idag äter vi närmre 6 miljoner semlor*（今日、600万個近くのセムラが食される）2020年2月23日発表（2023年1月14日閲覧）

14. 散布図

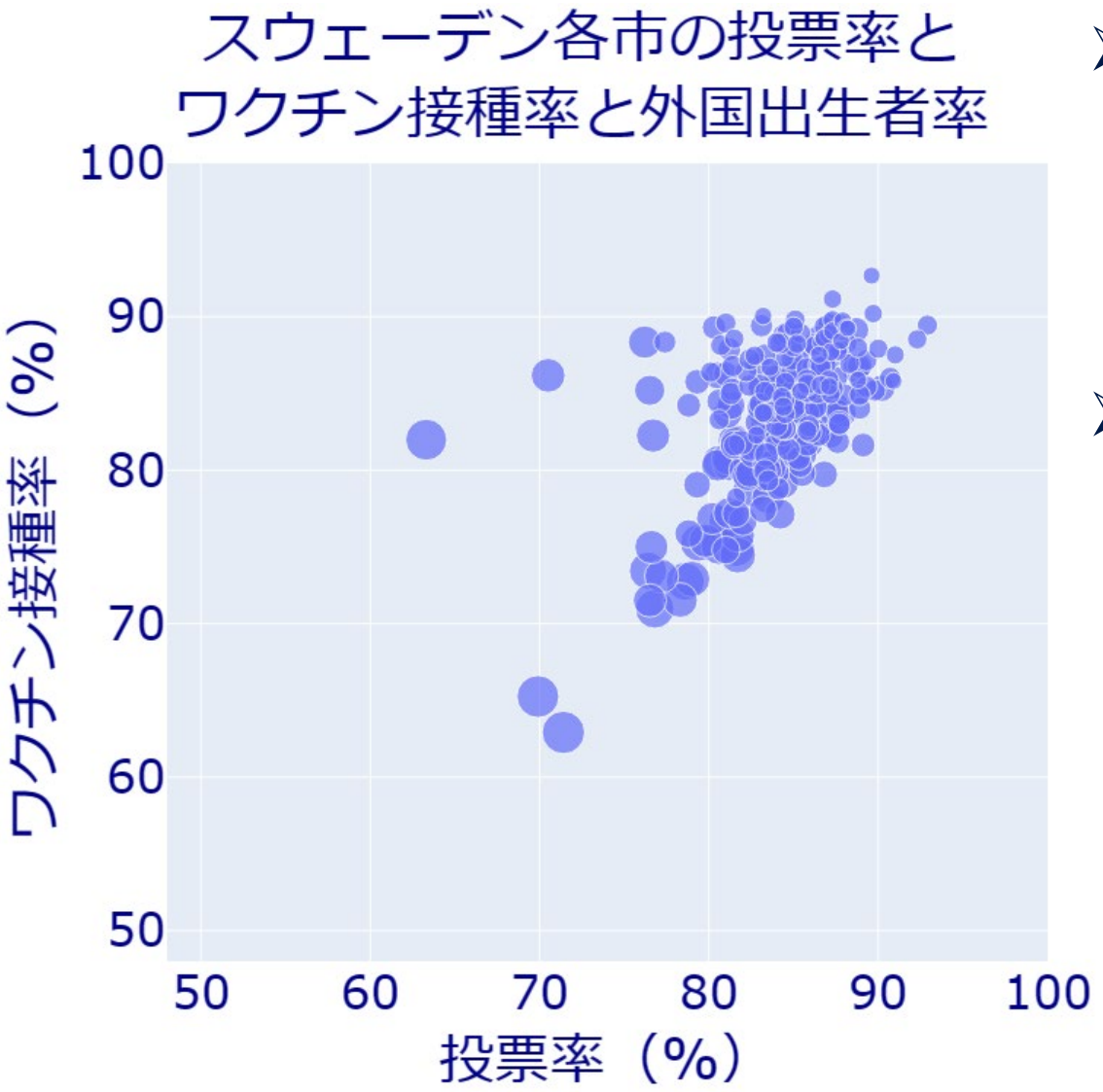


- 散布図は、データの2項目間の関係性を見るため、データに対応する点を平面上に配置した図。横軸と縦軸が、2項目それぞれの値に対応する。
- 左の図は、スウェーデン各市（全290市）の市議会議員選挙の投票率と、COVID-19ワクチン接種率の関係を示したもの。1つの点が、1つの市に対応する。
- 横軸は2018年に実施された市議会議員選挙の投票率、縦軸は当該市内の18歳以上の人のうち、COVID-19ワクチン接種を2回以上済ませた人の割合（2021年10月26日現在の値）。

※ スウェーデン統計庁の分析により、地域をさらに細分化すると、投票率と接種率の関係が、より鮮明になることが示されている。

データの出典：[12] https://www.scb.se/ SCB（スウェーデン統計庁）， *Vaccinationsgrad och valdeltagande i samtliga kommuner* （全市のワクチン接種率と投票率）, 2023年1月8日閲覧

15. バブルチャート

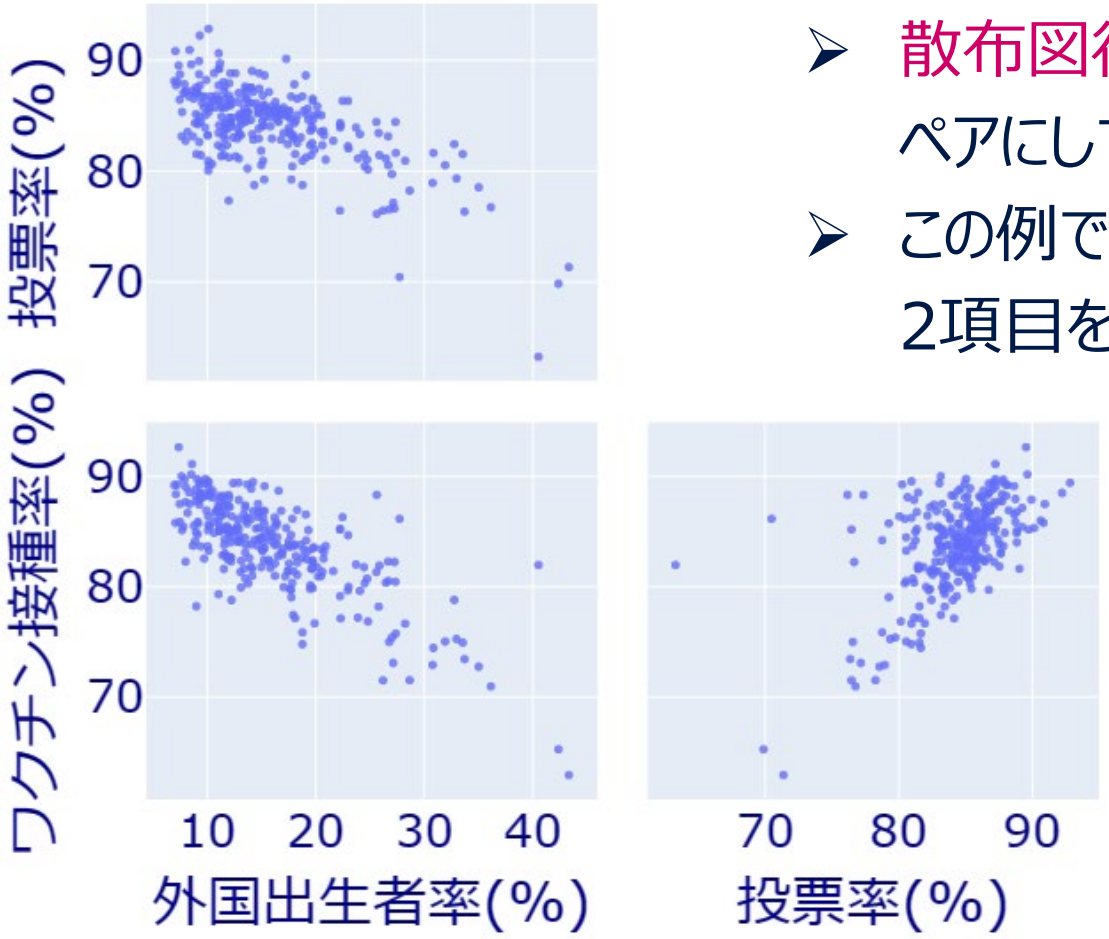


- バブルチャートは、3項目間の関係を同時に表す。横軸と縦軸は散布図と同様で、それに加え、3つ目の項目は、円の「大きさ」で表現する。「大きさ」を面積とするか半径とするかで見え方が変わるので注意。
- 左の図の横軸（投票率）と縦軸（ワクチン接種率）は、前ページの散布図と同じ。それに加え、各市の住民の内、外国生まれの人の割合の相対的な大きさを、円の面積で示している。

データの出典： SCB（スウェーデン統計庁），前ページのデータ [12] 及び前述の [3] *Antal personer med utländsk eller svensk bakgrund (fin indelning) efter region, ålder och kön*（国外・国内に背景を持つ人々（詳細区分）の地域・年齢・性別ごとの人数），2023年1月8日閲覧

16. 散布図行列

スウェーデン各市の投票率と ワクチン接種率と外国出生者率



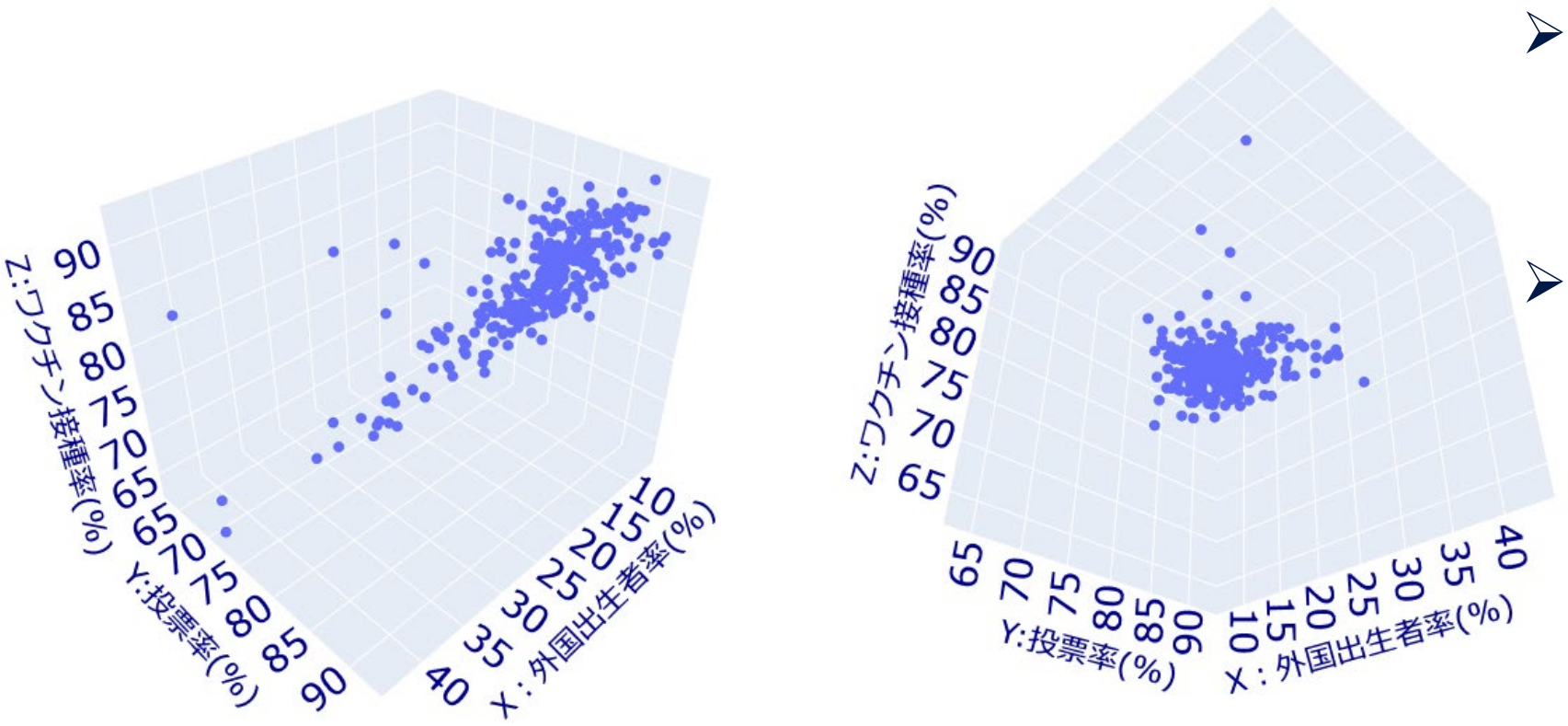
- 散布図行列では、3項目以上の相互の関係性を、2項目ずつペアにして散布図を描き、それを並べて表現する。
- この例では、外国出生者率、投票率、接種率の3項目から、2項目を選ぶことで、３組のペアができる。
- 対角線上に、ヒストグラムなど、各項目単独の分布に関する情報を記すこともできる。

※ 外国で生まれた人の割合（入国移民の割合）が高い市ほど、投票率とCOVID-19ワクチン接種率が、ともに低いという傾向が読み取れる。

データの出典：前ページと同じ（SCB）

17. 3次元散布図

スウェーデン各市の投票率とワクチン接種率と外国出生者率



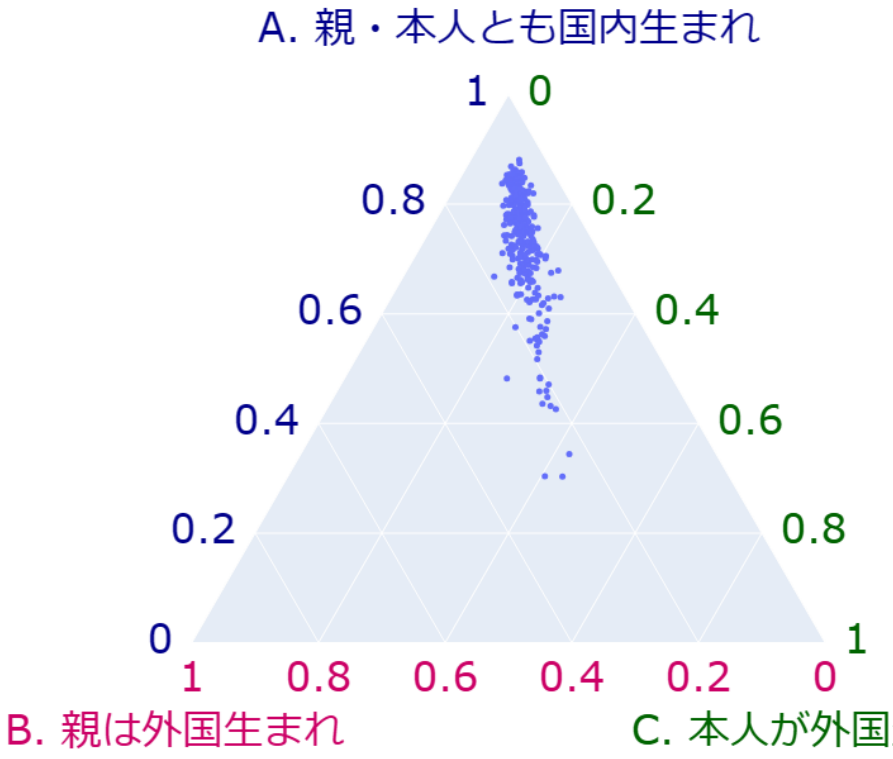
- 3次元散布図は、3項目間の関係性を、3次元空間の3つの軸に対応させて描画したもの。
- 一般に、点の分布の様子を正確に把握するのが難しい。また、視点の設定の仕方によっては、点が密集して見にくくなる。左の例では、左右で全く同じデータだが、見る角度を変えたために、印象が大きく異なる。

データの出典：前ページと同じ（SCB）

- 3次元空間がゆっくり回転するような動画にしたり、パソコンでマウスを使って、散布図を回転できるようにすると、幾分、把握しやすくなる。また、2次元平面に描くことができる別のチャート（バブルチャート、散布図行列等）で代替した方が分かりやすい場合も多い。

18. 三角図

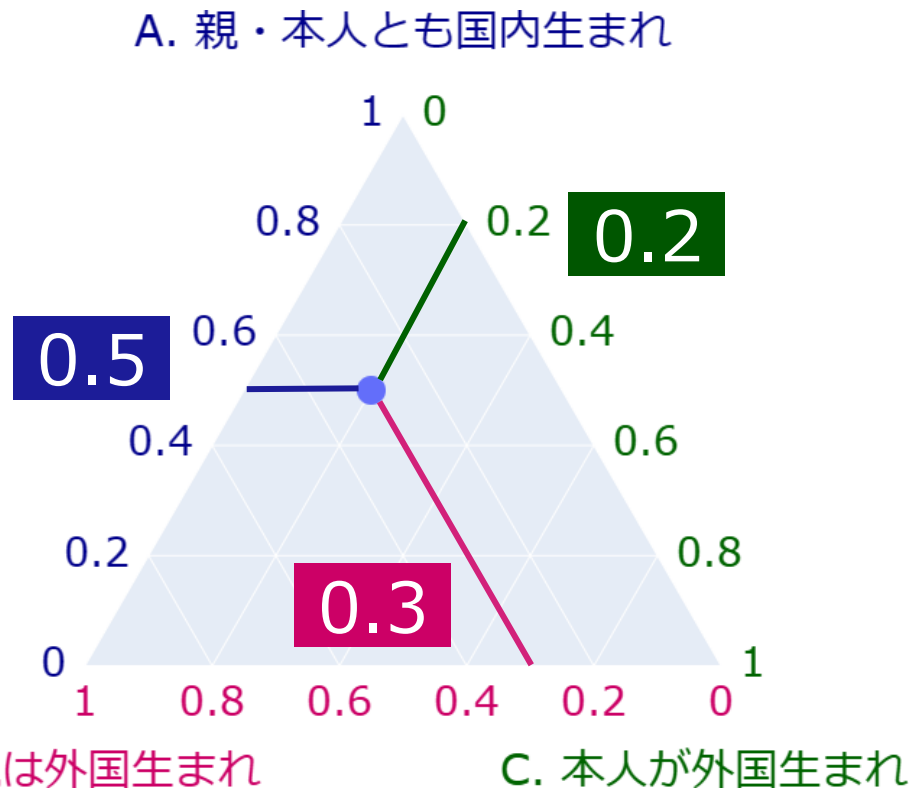
スウェーデン各市の住民の出自の割合



A.　親・本人とも国内生まれ
両親、本人ともスウェーデン生まれの人の割合

B. 親は外国生まれ
本人はスウェーデン生まれだが、両親の少なくとも一方が外国生まれの人の割合

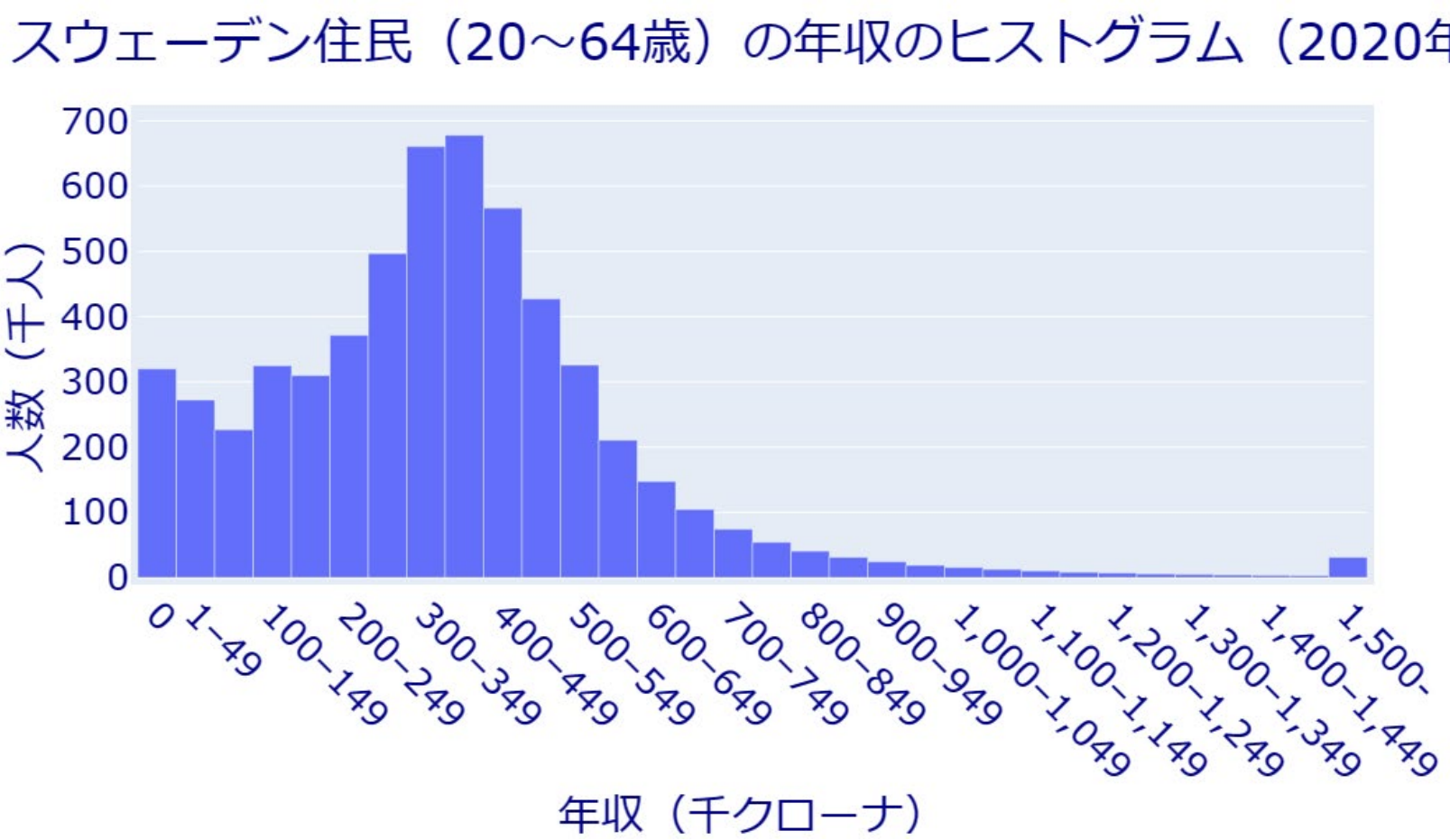
C. 本人が外国生まれ
本人が外国生まれの人の割合



データの出典：前述の [3] https://www.scb.se/ SCB（スウェーデン統計庁）, *Antal personer med utländsk eller svensk bakgrund (fin indelning) efter region, ålder och kön*（国外・国内に背景を持つ人々（詳細区分）の地域・年齢・性別ごとの人数）, 2023年1月12日閲覧

- 三角図は、3項目間の相対的な割合を図示。
- 各項目の値は、同じ色の目盛りを読む。上の例では、Aが0.5、Bが0.3、Cが0.2（A、B、Cの和は、三角形内のどこの点でも一定）。

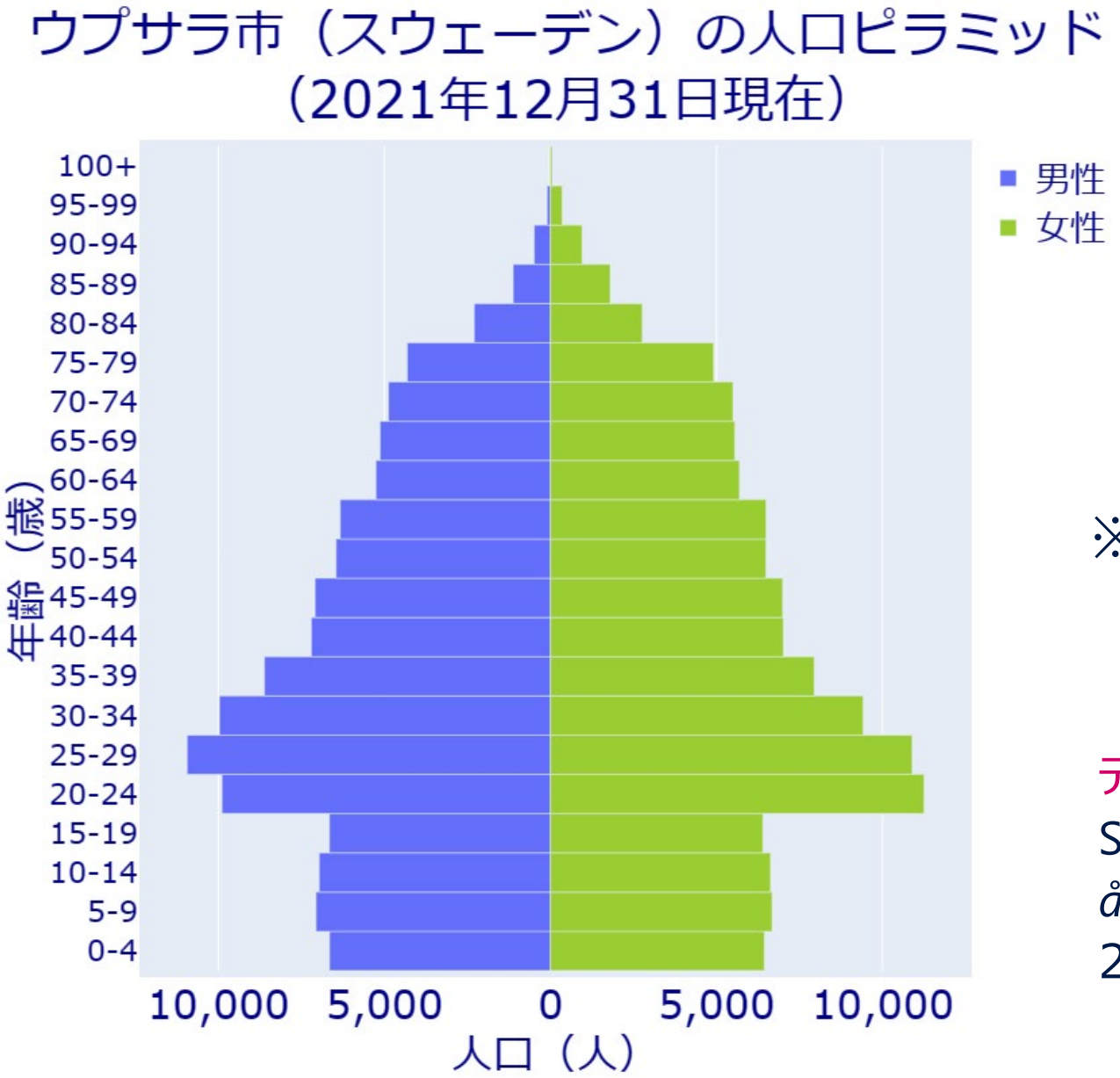
19. ヒストグラム



データの出典：[13] https://www.scb.se/ SCB（スウェーデン統計庁），Antal personer i olika intervall av sammanräknad förvärvsinkomst efter ålder 2020（年齢ごとの総稼得所得帯別人数, 2020年のデータ），2023年1月8日閲覧

- ヒストグラムでは、データの取り得る値を、いくつかの階級（値の幅）に分ける。図の横軸は階級を表し、縦軸は、その階級に当てはまる度数（頻度）を表す。
- 左の例では、年収が横軸に示された額の範囲に収まる人の人数を、縦軸の値で表している。
- ヒストグラムでは、階級の幅の取り方により、見た目の印象が大きく異なる場合がある。

20. 人口ピラミッド

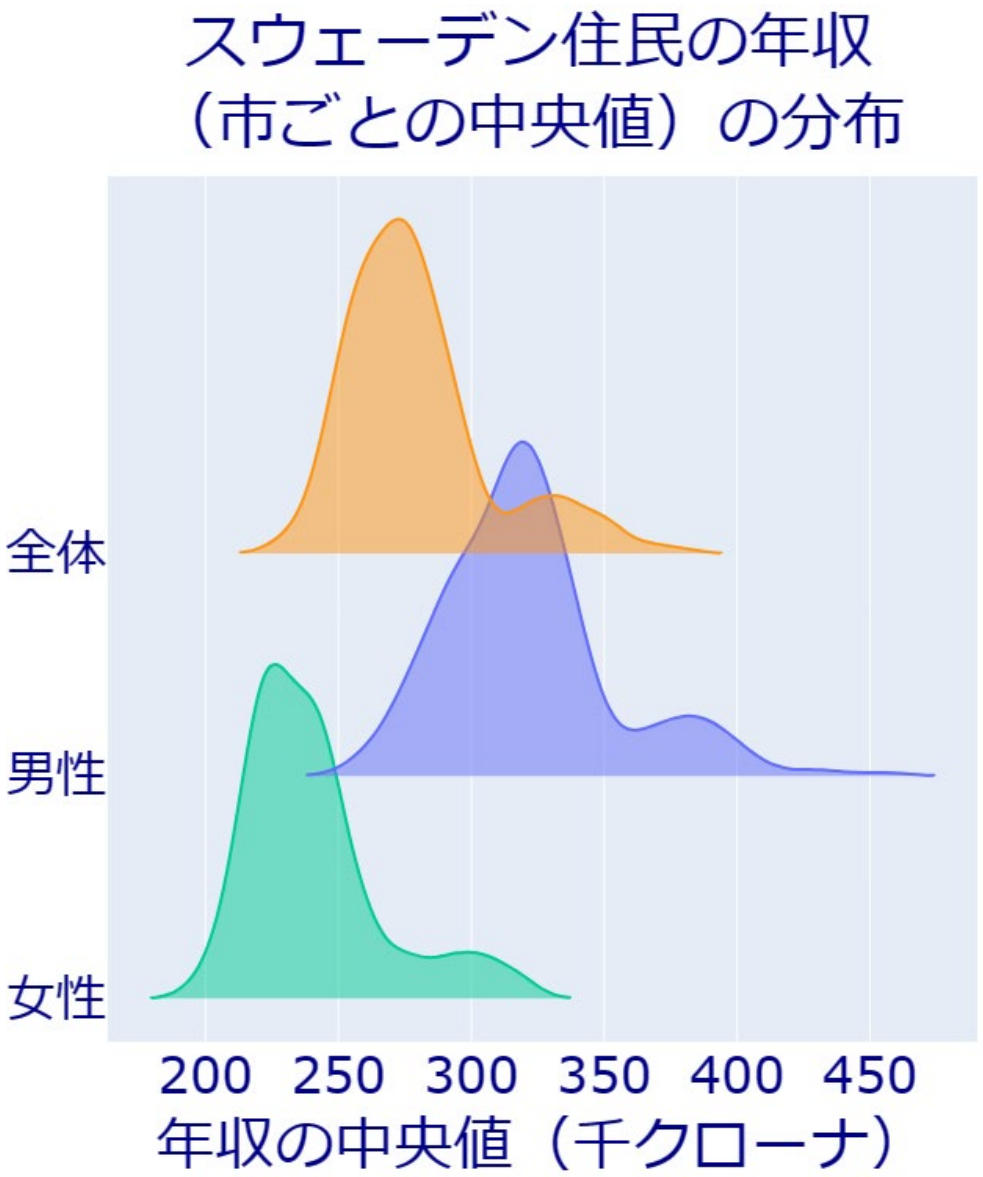


- 人口ピラミッドは、年齢層を階級とし、人口を度数とするヒストグラムである。
- 通常、性別ごとのヒストグラムを、左右に背中合わせの形で描く。

※ ウプサラ市は、ウプサラ大学を中心とする学園都市なので、20代、30代の人口が多い。

データの出典：[14] https://www.scb.se/
SCB（スウェーデン統計庁）, *Folkmängd efter region, ålder, kön och år*（地域、年齢、性別、年ごとの人口）, 2023年1月8日閲覧

21. 密度プロット

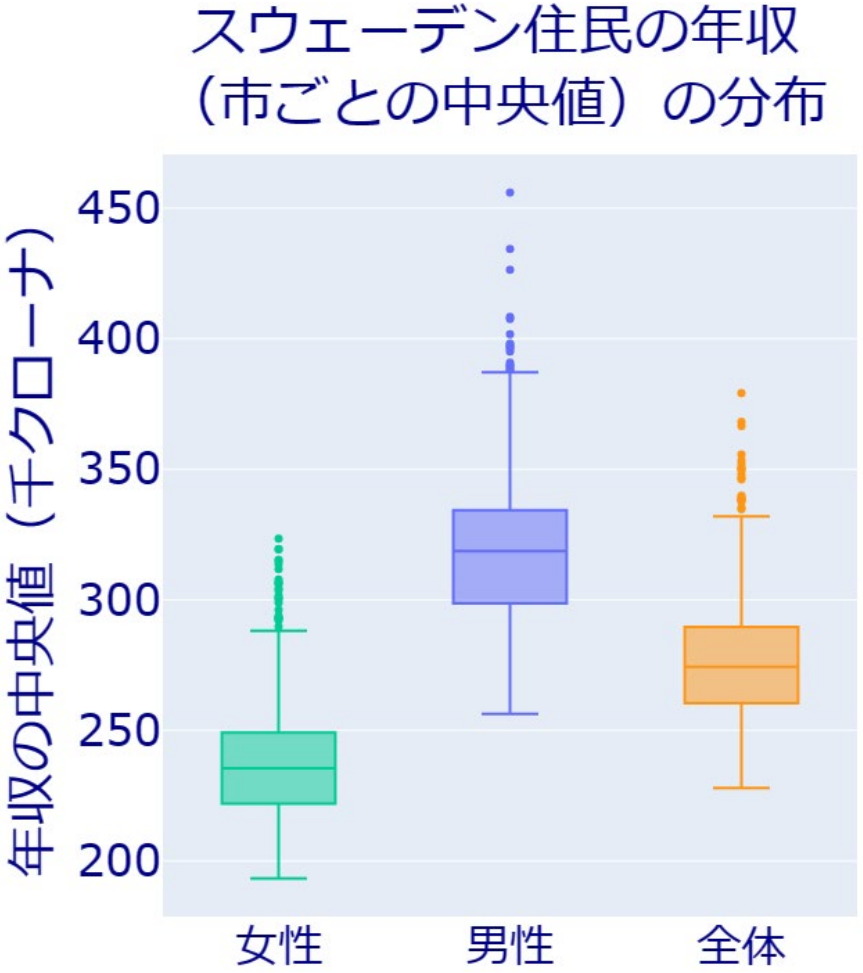


- 密度プロットは、大雑把に言えば、ヒストグラムを、滑らかな曲線に置き換えたもの。
- 実際に、曲線を算出する方法としては、「カーネル密度推定」と呼ばれる方法が広く用いられている。

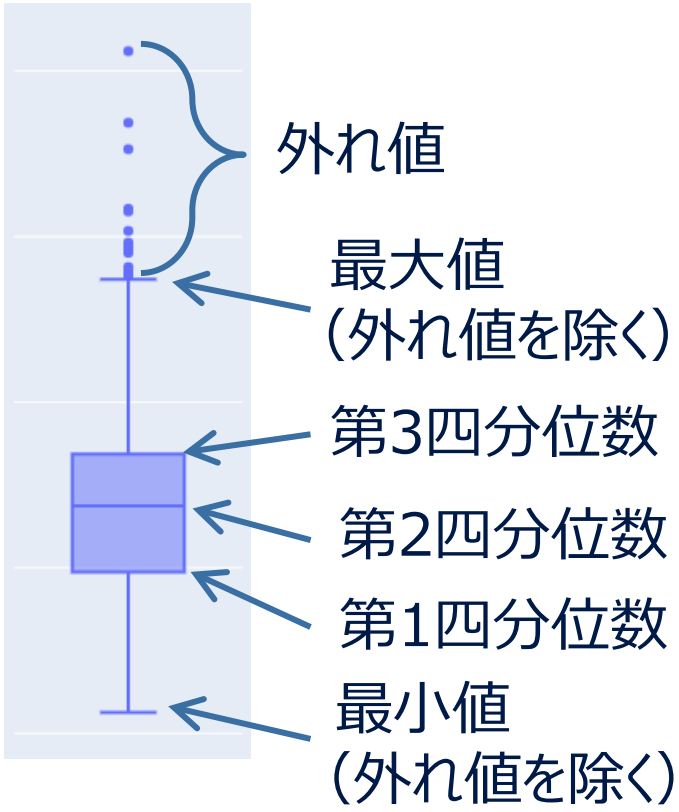
※元データは、スウェーデンの各市（全290市）について、16歳以上の全住民及び性別ごとの年収の中央値を算出したもの。この中央値の分布（290市間の差異の様子）を、密度プロットで表現した。

データの出典： [15] https://www.scb.se/ SCB（スウェーデン統計庁）， *Sammanräknad förvärvsinkomst för boende i Sverige den 31/12 resp år efter region, kön, ålder och inkomstklass*（年、地域、性別、年齢層、所得階級ごとの、12月31日現在のスウェーデン住民の総稼得所得），2023年1月9日閲覧

22. 箱ひげ図



- 箱ひげ図では、量的なデータの分布の様子を、最小値、第1～3四分位数、最大値を用いて表す。
- データを小さい順に並べ、全体を4等分する際の境目の値を、小さい方から順に、第1、2、3四分位数と呼ぶ。
- 第3四分位数から第1四分位数を引いた値（箱の高さ）を、四分位範囲（IQR）と呼ぶ。

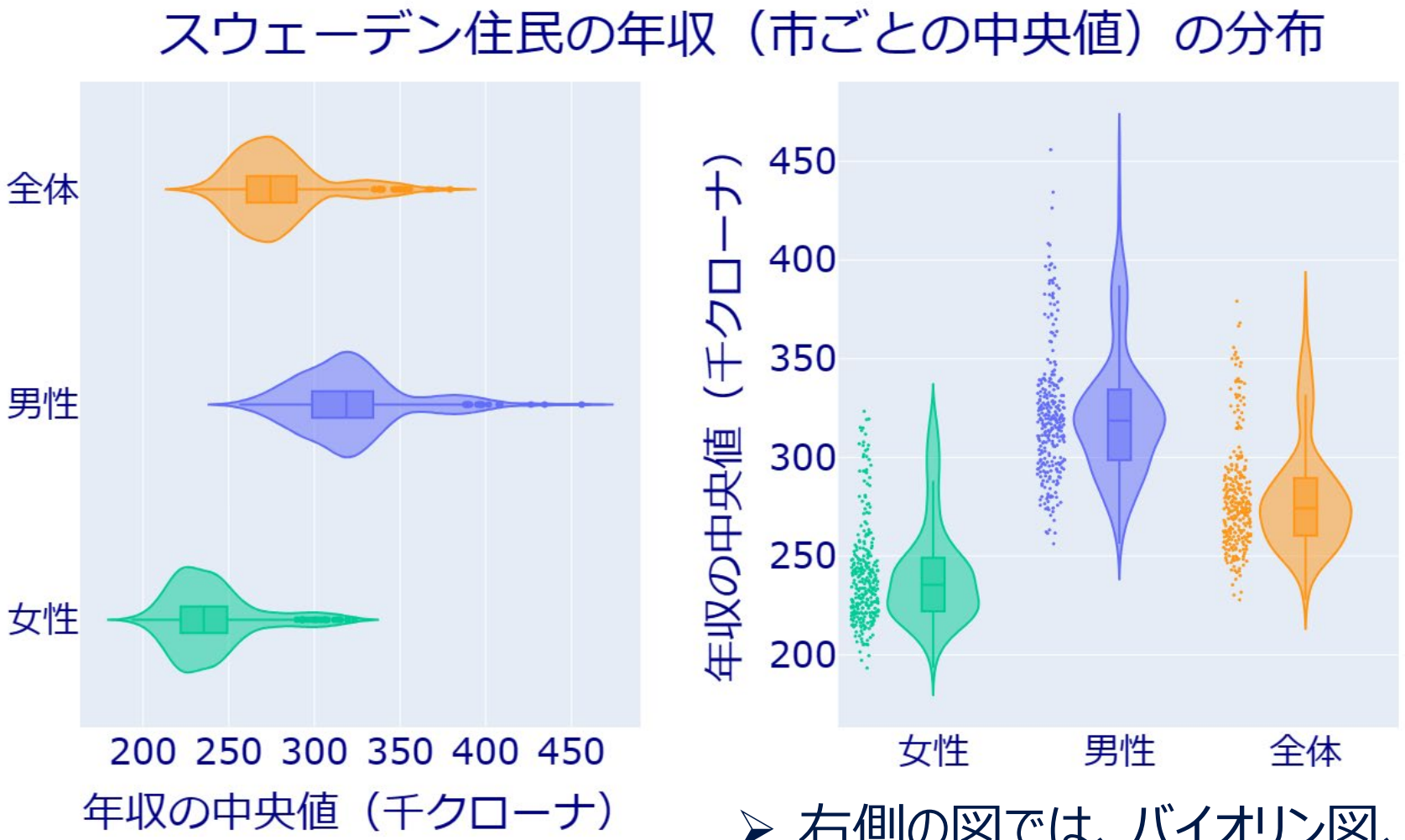


※ この例では、下側にはみ出た外れ値はなかった。

- 上の箱ひげ図では、第1四分位数より下方に、又は第3四分位数より上方に、四分位範囲の1.5倍以上ズレたデータを、外れ値として扱っている。

データの出典：
前ページと同じ（SCB）

23. バイオリン図

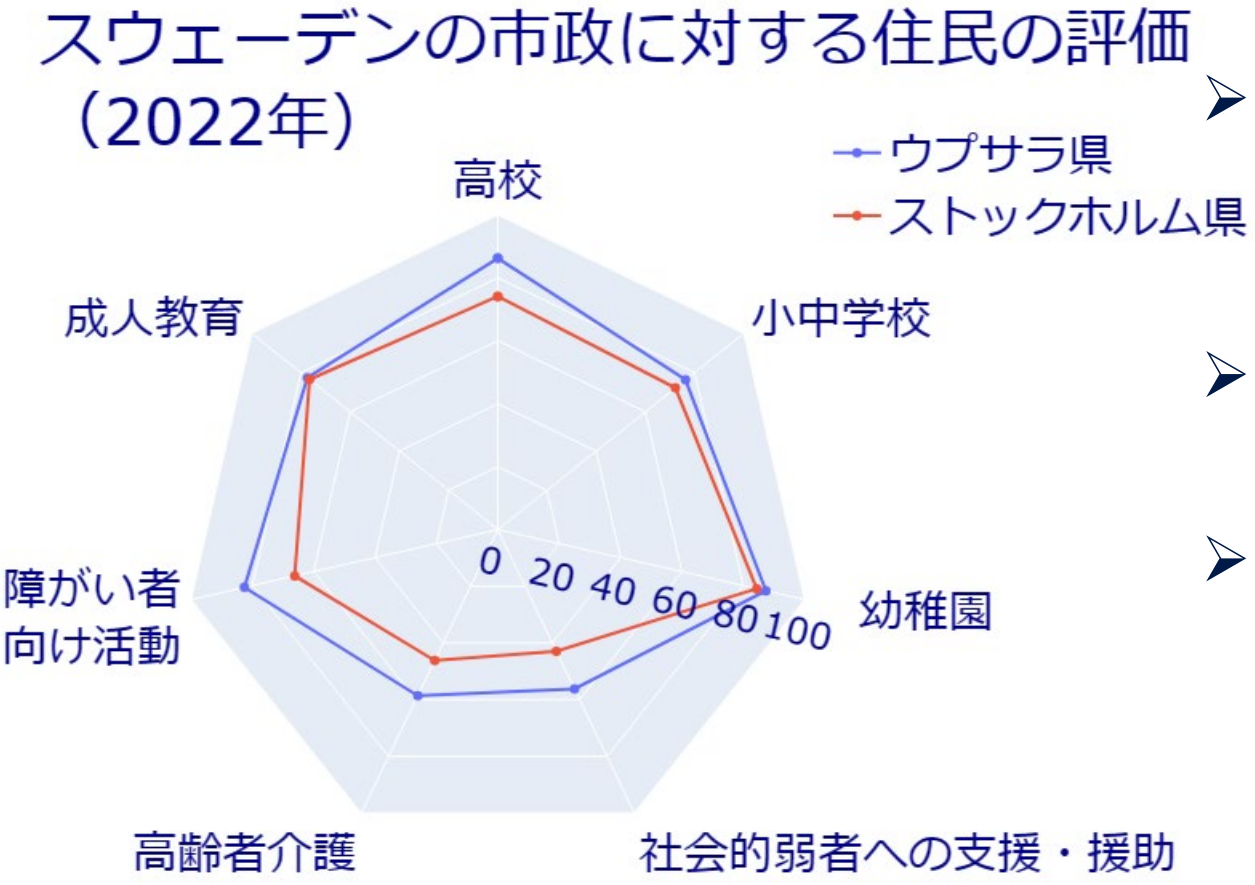


- バイオリン図は、密度プロットを上下対称に繋げたもの。
- 対称ではなく、上下で違うデータ（例えば、男性と女性）の分布を描く場合もある。
- 上下ではなく、左右に繋げても良い。

- 左側の図では、バイオリン図に加え、箱ひげ図も重ねて描いた。
- 右側の図では、バイオリン図、箱ひげ図に加え、もとのデータの分布の様子も記した。

データの出典：
前ページと同じ（SCB）

24. レーダーチャート

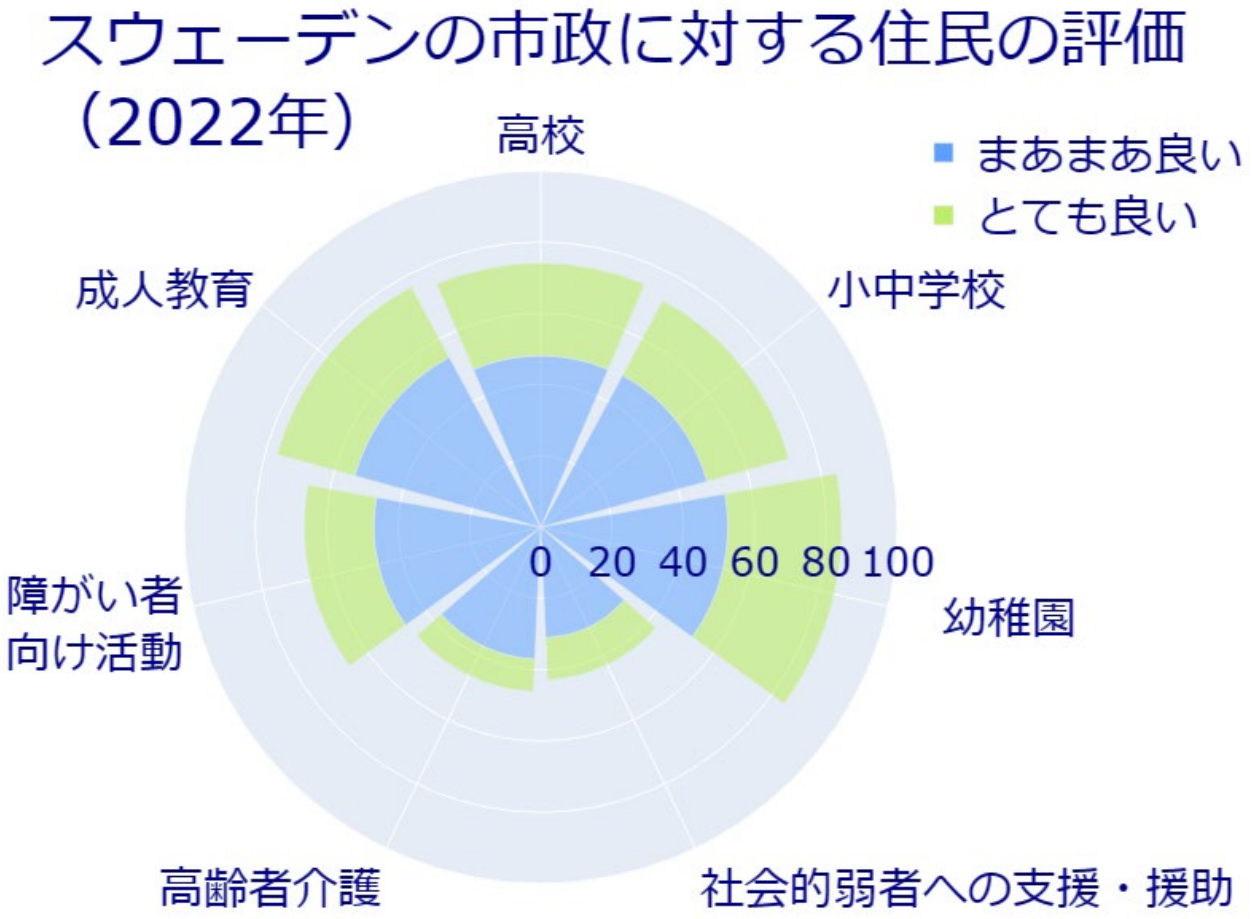


※ スウェーデンの一般市民対象のアンケート調査。教育・福祉の各項目に対し、市のサービスが、「とても良い」「まあまあ良い」と答えた人の割合の合計（%）を表示（ウプサラ、ストックホルム両県の分）。

- レーダーチャートでは、複数の項目について、それらの数値を、図の中心から放射状に伸びる軸に沿った距離で表現する。
- 各項目間を直線でつなぐことで、データの持つ特徴を、多角形の形状として捉えることができる。
- 左の例では、「幼稚園」「小中学校」～「社会的弱者への支援・援助」の7項目に対応する7角形を用い、ウプサラ、ストックホルム両県の住民による市のサービスに対する評価を示した。

データの出典： [16] https://www.scb.se/
SCB（スウェーデン統計庁）， *Medborgarnas syn på skola och omsorg efter region och bakgrund*（学校及び介護に関する地域・背景別の市民の見解），2023年1月17日閲覧

25. ポーラーチャート

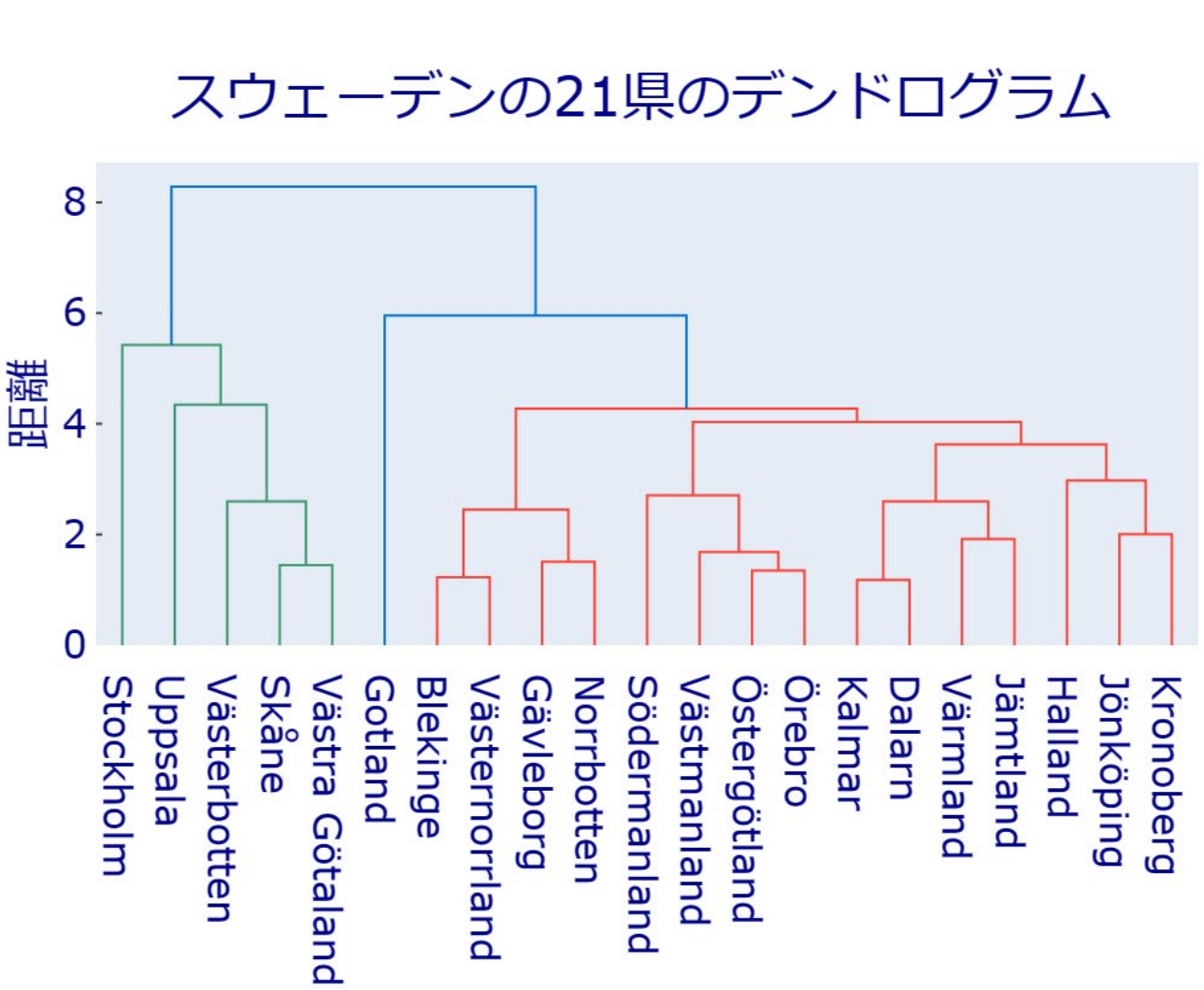


※ スウェーデンの一般市民対象のアンケート調査。教育・福祉の各項目に対し、市のサービスが、「とても良い」「まあまあ良い」と答えた人の割合（%）を表示（ストックホルム県の分）。

- ポーラーチャートでは、複数の項目について、それらの数値を、扇形で表現する。
- 各項目に対応する扇形は、中心角が固定されていて、数値の大きさは、半径に反映される。
- 積み上げ棒グラフと同様の要領で、各主項目（左の例では、「幼稚園」「小中学校」「高校」など）ごとに、複数の数値（左の例では「とても良い」と「まあまあ良い」の割合）を表現可能。
- レーダーチャート同様、全体の形のバランスにより、データの持つ特徴を把握できる。
- 円の中心から離れるほど、面積が大きくなり、見た目の印象が強くなるので留意が必要。

データの出典：前ページと同じ（SCB）

26. デンドログラム



- デンドログラムは、似たもの同士を結び付けていき、最終的に１つの集団になるまでの様子を、系統樹と同様な形で表現したもの。系統樹のように時間的要素が存在するとは限らない。
- 左の例は、スウェーデンの全国21県について、県民の「平均年齢」「家賃」「海外転入出」「国内転入出」「高等教育進学率」「週労働時間」「出生率」「持ち家率」の8項目の統計データを元に、「階層的クラスタリング」という手法により、デンドログラムを作成したもの。
- 項目（この例では県）同士の「距離」の定義に依存し、結果が大きく異なることがある。

データの出典： https://www.scb.se/ SCB（スウェーデン統計庁），各種統計データ（次ページに記載）

26. デンドログラム

「スウェーデンの21県のデンドログラム」に用いたデータ

データの出典： SCB（スウェーデン統計庁）， 2023年1月17日閲覧

[17] *Befolkningens medelålder efter region och kön*（地域、性別ごとの人々の平均年齢）

[18] *Hyra i hyreslägenheter efter län och kommun*（県・市ごとの賃貸アパートの家賃）

[19] *Flyttningsvariabler efter län och kön*（県及び性別ごとの転入出に関する変量）

[20] *Demografivariabler för samtliga efter län och kön*（全員が対象の県及び性別ごとの人口統計学的変量）

[21] *Medelarbetstid (överenskommen) per vecka för sysselsatta 15-74 år (AKU) efter region och kön*（地域、性別ごとの、労働力調査における15～74歳の被雇用者の（合意された）週平均労働時間）

[22] *Summerad fruktsamhet efter region och kön*（地域、性別ごとの、合計特殊出生率）

[23] *Antal och andel hushåll efter region, boendeform och bostadsarea (exklusive specialbostäder)*（地域、住居の形態、居住エリアごとの世帯数と割合（特殊な住居は除く））

26. デンドログラム

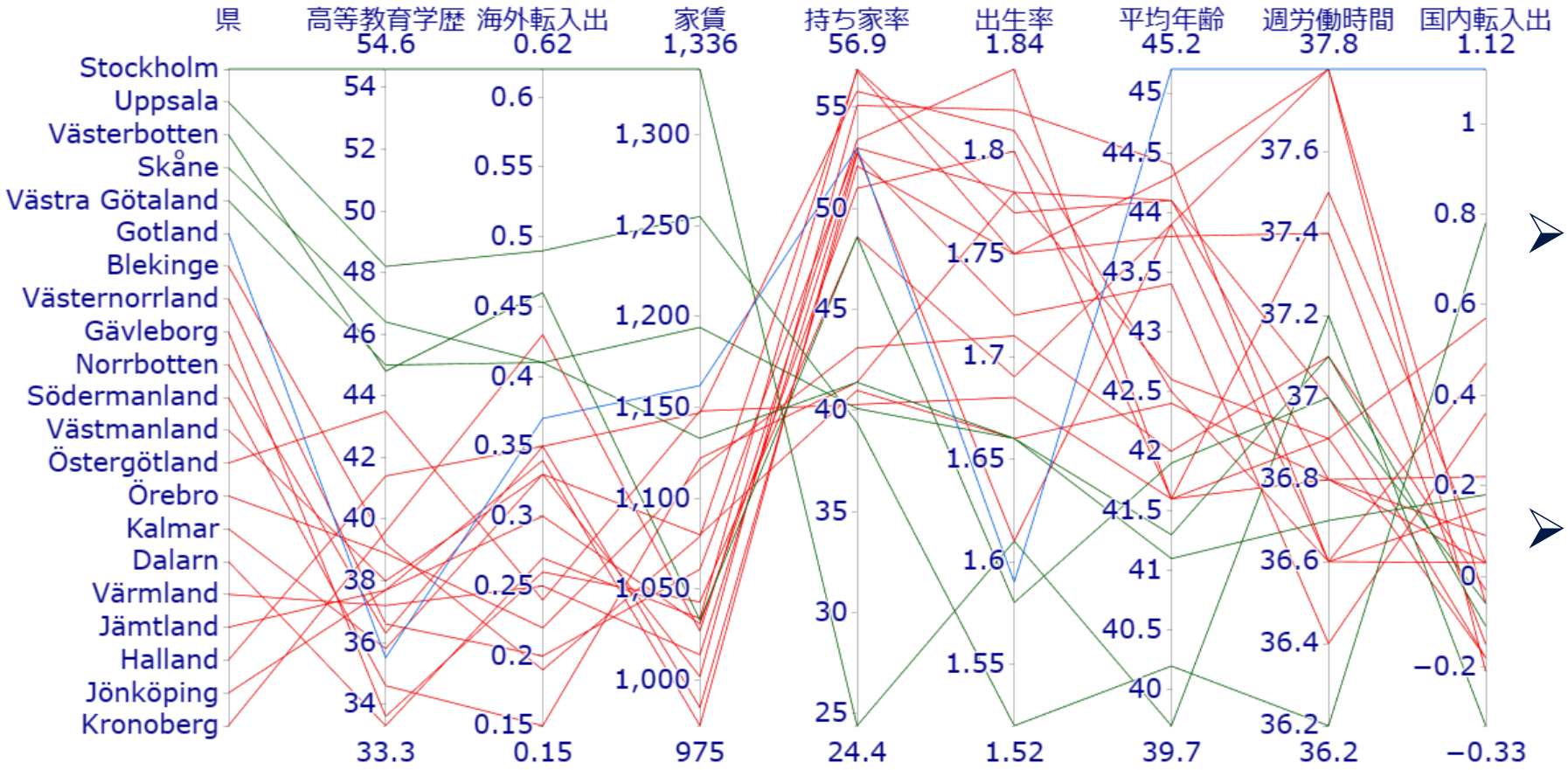
「スウェーデンの21県のデンドログラム」に用いたデータ

- 平均年齢：県民の平均年齢（歳）
- 家賃：県内の賃貸アパートの1平方メートル当たりの1年間分の家賃の中央値（クローナ）
- 海外転入出：海外からの移住者数から海外への移住者数を引いた数が県の人口に占める割合（%）
- 国内転入出：他県からの転入者数から他県への転出者数を引いた数が県の人口に占める割合（%）
- 高等教育進学率：スウェーデン生まれの県民の大学・職業学校等への進学割合（%）
- 週労働時間：県民の1週間の労働時間の平均値（時間）
- 出生率：女性県民の合計特殊出生率
- 持ち家率：所有権を有する戸建て住宅に住んでいる県民の割合（%）

※ いずれも、2021年のデータを使用した。

27. 平行座標プロット

スウェーデンの21県の平行座標プロット

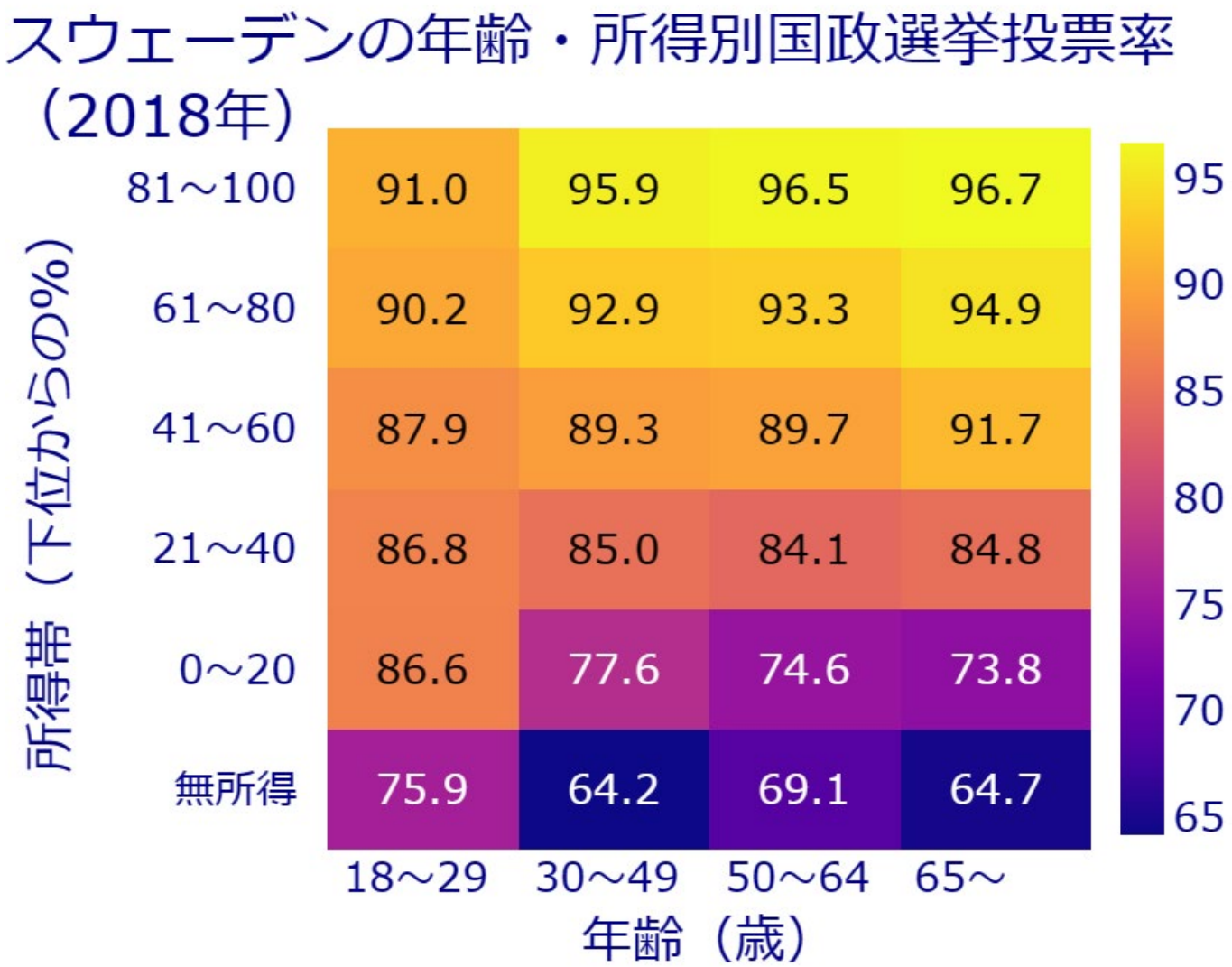


- 平行座標プロットでは、複数の項目の数値を、縦方向に平行に伸びる複数の軸で表す。
- 各項目間を直線でつなぐことで、データの持つ特徴を、折れ線の形状として捉えることができる。
- 左の例では、「高等教育学歴」から「国内転入出」の8項目の値を、8本の縦軸を用いて表している。

- 県の並び順、折れ線の色は、クラスタリングの結果（「26. デンドログラム」参照）に準じた。

データの出典： デンドログラムと同じ（SCB）

28. ヒートマップ

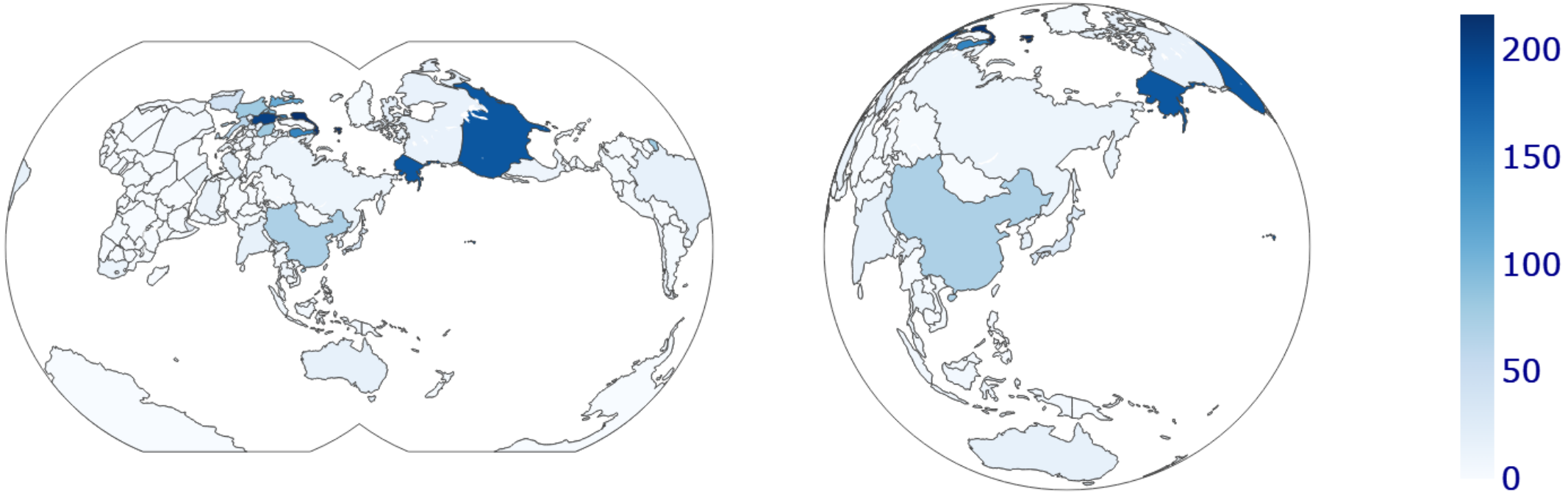


- ヒートマップは、横方向と縦方向それぞれに対応する2つの属性に応じて値が決まるデータを可視化したもの。データの値は、色で示される。
- 左の例では、スウェーデンの国会議員選挙について、有権者の年齢層（横方向）と所得帯（縦方向）の組み合わせごとに、投票率の値を色で表示している。紫が濃いほど投票率が低く、黄色に近いほど、投票率が高い。

データの出典：[24] https://www.scb.se/ SCB（スウェーデン統計庁），*Valdeltagande i riksdagsvalet 2018*（2018年国会議員選挙投票率），2023年1月16日閲覧

29. 階級区分図

スウェーデンから世界各国・地域への輸出額（2022年、単位：10億クローナ）

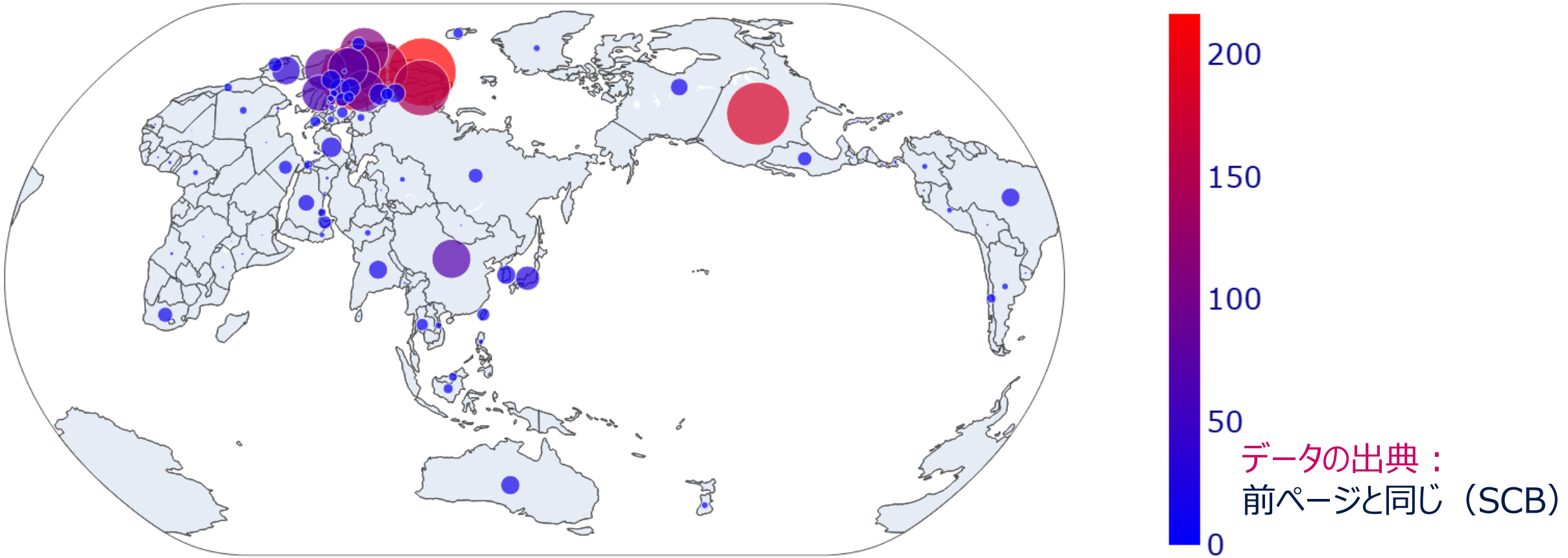


- 階級区分図では、データの値に応じ、該当箇所を色で塗り分ける。
- 国や県を塗り分けるなどすると、地理的な関係性が理解しやすくなる。
- 一方で、面積の大小等によって、誤った印象を与える可能性もあるので要注意。

データの出典： [25] https://www.scb.se/ SCB（スウェーデン統計庁）, *Varuimport och varuexport. Totala värden efter handelspartner, bortfallsjusterat*（貿易相手別の商品輸出入合計値、無回答調整済）, 2023年1月8日閲覧

30. 比例シンボルマップ

スウェーデンから世界各国・地域への輸出額（2022年、単位：10億クローナ）

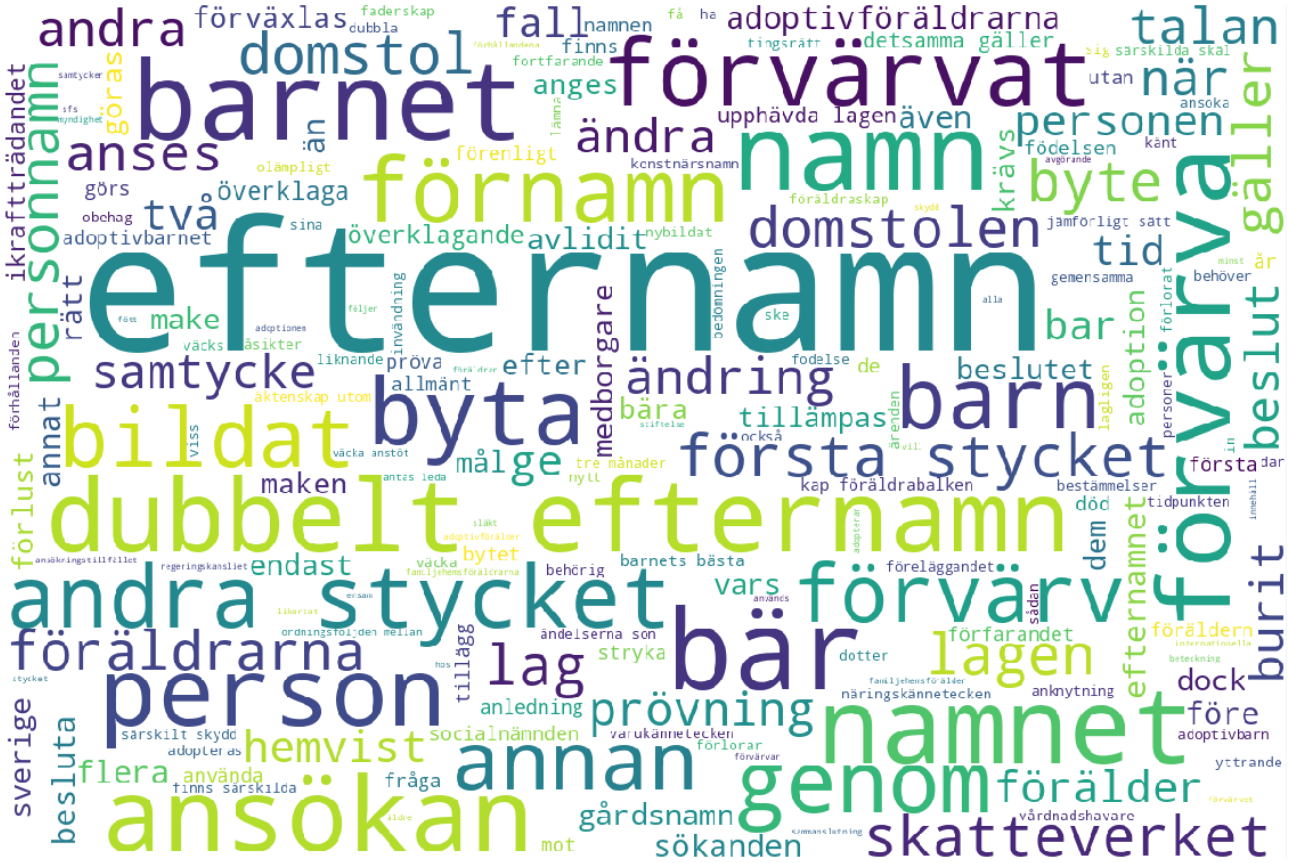


データの出典：
前ページと同じ（SCB）

- 比例シンボルマップでは、データの値に応じた大きさの円を、該当箇所に描く。
- 円の大きさだけからは、値が把握しにくいので、円の色も値に連動させると分かりやすい。
- 円の「大きさ」を面積とするか半径とするかで見え方が変わるの注意（上の例は面積）。

31. ワードクラウド（おまけ）

スウェーデンの人名に関する法律の条文を可視化



- ワードクラウドでは、テキストデータ（文章データ）に含まれる各単語が、その文書においてどの程度重要かを算出し、重要度の大きい単語ほど、目立つように表示。
- 重要度の決め方は様々だが、最も単純には、出現頻度の大きさを重要度とする。
- しかし、出現頻度のみに着目すると、「こと」など、どんな文章でも頻出する単語が、常に重要単語と判定されてしまう。
- 単語の重要度は、通常の文書におけるレア度と、解析対象の文書における頻度の両方を加味して判定すると良い。tf–idfと呼ばれる指標が有名。

データの出典： [26] https://www.riksdagen.se/
Sveriges riksdag（スウェーデン国会）,
Lag (2016:1013) om personnamn（人名に関する法律）, 2024年1月7日閲覧

データ分析と可視化

付録A

愛媛大学　　松浦 真也

Ver. 2024.1.10

とりあえず描画

Python & Plotly

- 以下では、Googleの無料サービスである Google Colaboratory 上で、Pythonの plotly ライブラリを用いてチャートを描画する方法を紹介する
- Pythonを用いたチャートの描画方法は種々あるが、plotly を用いると、マウスで動かせるインタラクティブで美しいチャートが、容易に描ける
- plotly には、手軽だが自由度が限定的な express もあるが、この資料では、主に、自由度が高い graph_objects を用いる
- Google Colaboratory には、 plotly が最初からインストール済みである
- Google Colaboratory 以外の環境で使用の場合、plotly が入っていなければ、自分でインストールする必要がある

とりあえず描画

まずは、細かいことは気にせず、描画してみよう！

- ここで紹介するスクリプト（簡易的なプログラム）は、すべて5行以内（多くは2～3行）
  例

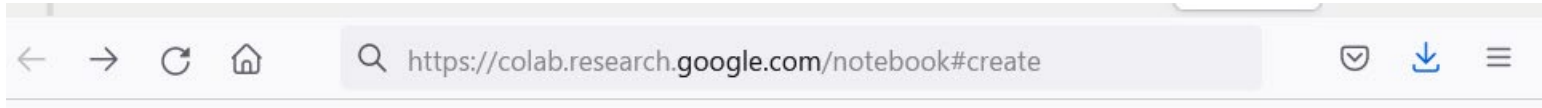
```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],fill='tozeroy')).show()
```

- とりあえず、必要最小限の記述で、図を表示させてみる
- 文字が小さかったり、図の縦横比が不釣り合いだったりするが、それは後で直すことにして、まずは気にしない

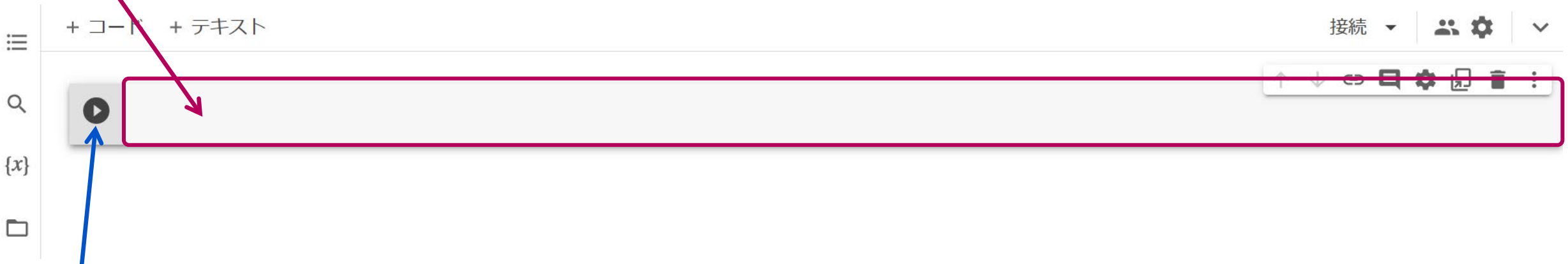
Google Colaboratory

Google Colaboratory 利用の場合の手順（Googleアカウントが必要）

1. Webブラウザ（Mozilla Firefox、Safari、Google Chrome、Microsoft Edgeなど）でGoogleにログイン
2. WebブラウザのURL欄に下記を入力
   https://colab.research.google.com/notebook#create=true



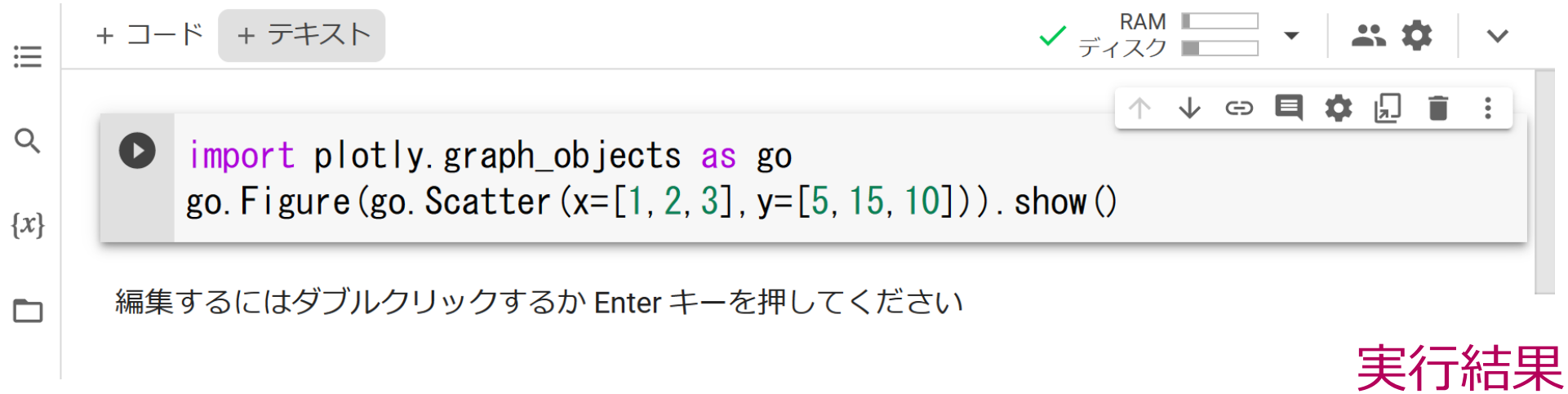
3. この欄に、次々ページ以降に記載のスクリプトを打ち込む



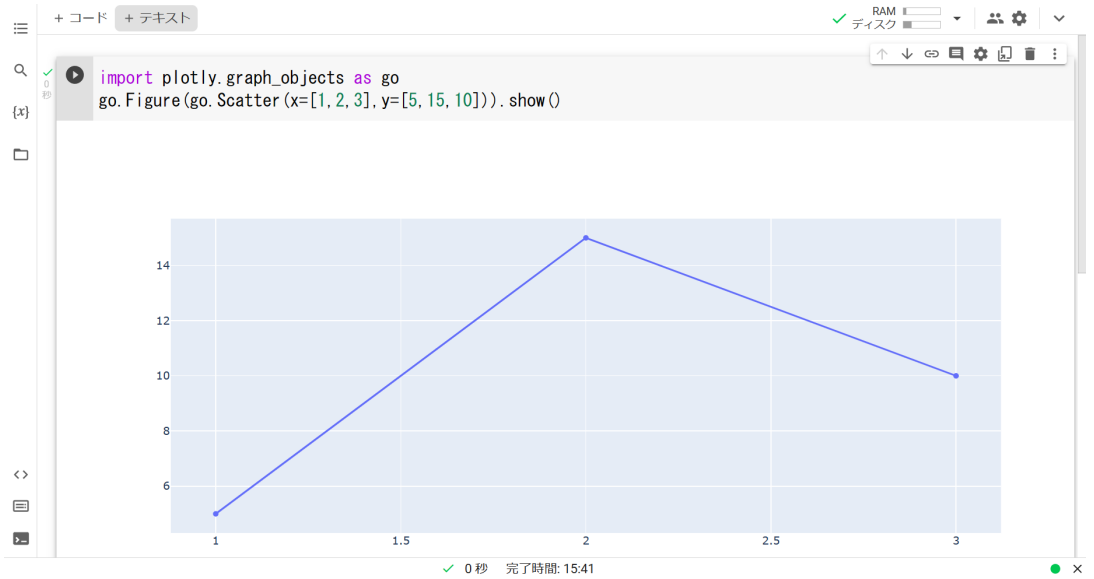
4. ここをクリックして、スクリプトを実行

Google Colaboratory

入力の様子



実行結果



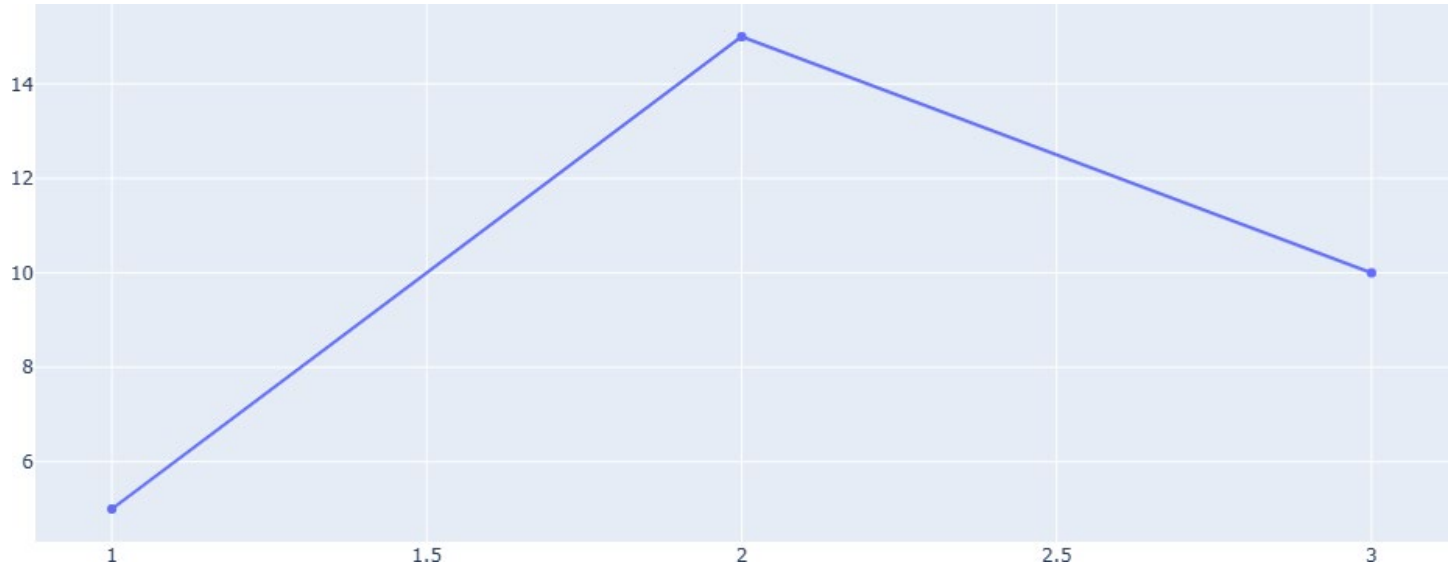
- 文字の色は、勝手に付く
- 全部入力し終わったら、左上の ▶ をクリックして、スクリプト（簡易的なプログラム）を実行
- 日本語を入力するとき以外は、半角文字を用いる（空白、カンマ、ピリオド、数字等、すべて半角）
- 行頭のインデント（字下げ）が、動作に影響することに注意

1. 折れ線グラフ

スクリプト1. 折れ線グラフ

```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10])).show()
```

実行結果



- importで始まる行は、描画に必要なライブラリを読み込んでいる
- 「as go」の部分では、読み込んだplotly.graph_objectsの略称として、goと名付けるという意味
- go.Scatterで散布図（scatter plot）を用意（ここで本当に描きたいのは折れ線グラフだが、散布図において、点を線分で結べば、折れ線グラフになる）
- 「go.Figure」で図を生成し、末尾の「.show()」で、生成した図を画面に表示

2. 面グラフ

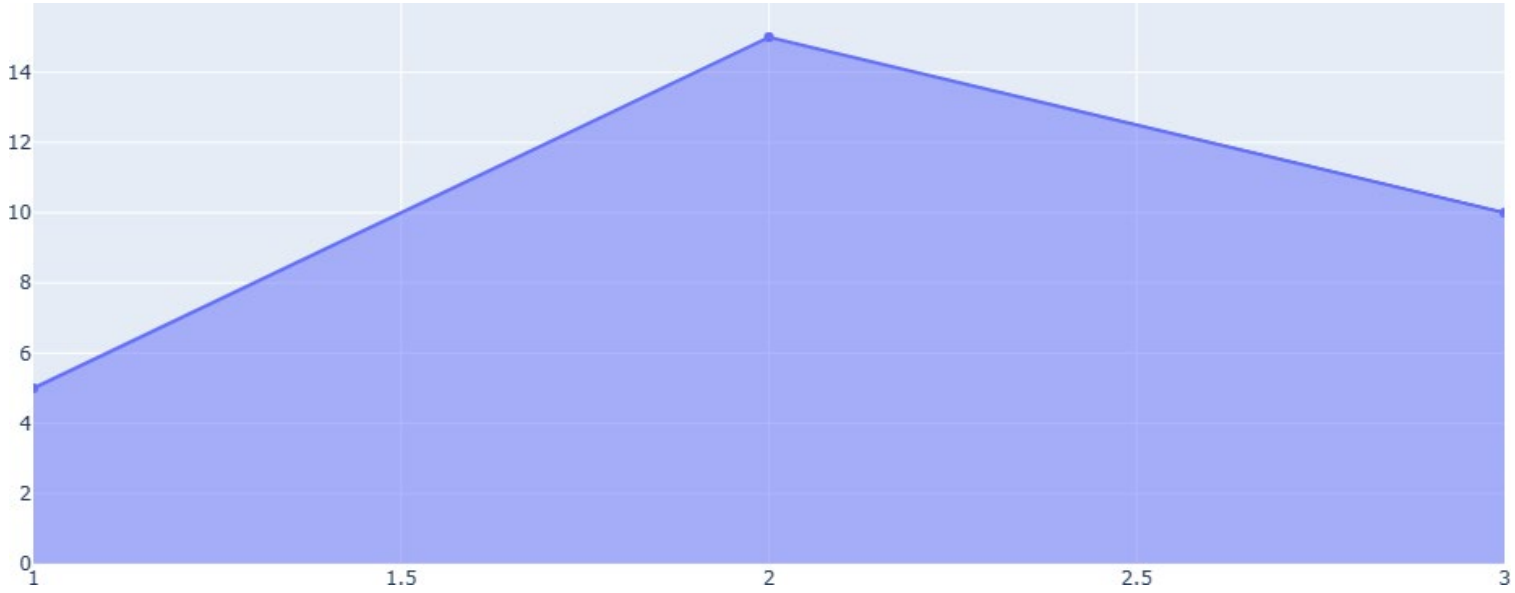
スクリプト2. 面グラフ

```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],fill='tozeroy')).show()
```

※以下のようにしても、同じグラフが描ける

```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],stackgroup=1)).show()
```

実行結果



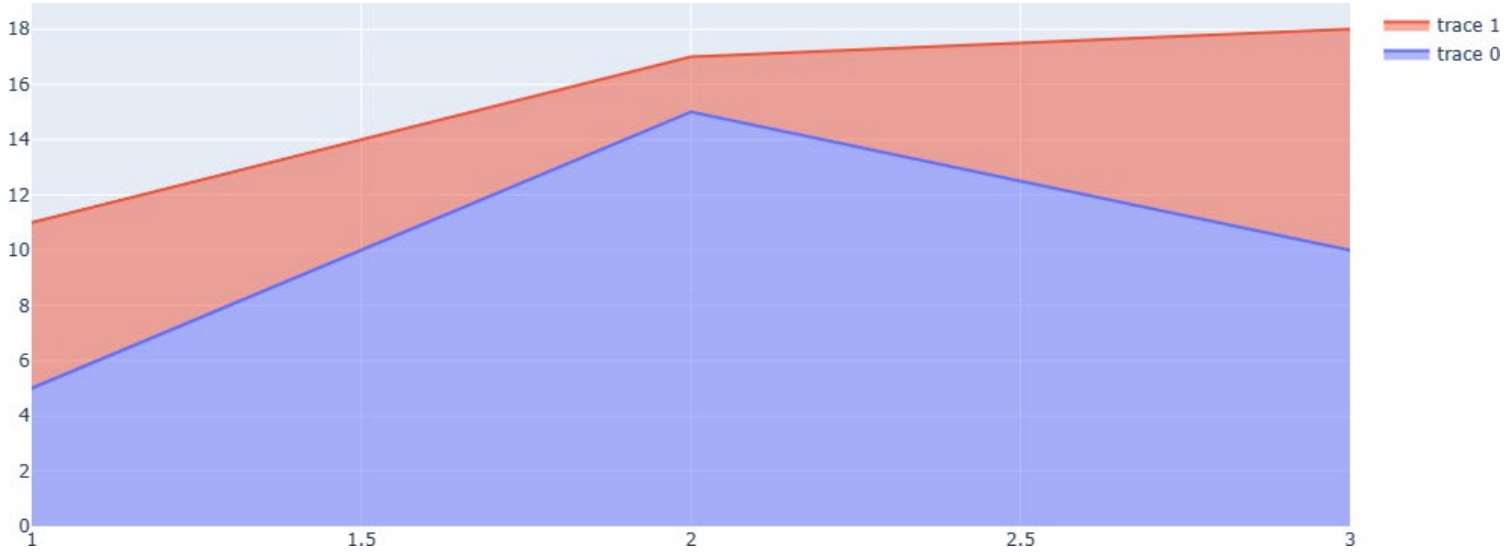
- 折れ線グラフを描くスクリプトにおいて、go.Scatter の部分で、fill='tozeroy' とオプションを指定すると、折れ線グラフの下側が塗りつぶされ、面グラフとなる
- stackgroup=1 とオプションを指定しても、効果は同じ

3. 積み上げ面グラフ

スクリプト3.　積み上げ面グラフ

```
import plotly.graph_objects as go
go.Figure([go.Scatter(x=[1,2,3],y=[5,15,10],stackgroup=1),
           go.Scatter(x=[1,2,3],y=[6,2,8],stackgroup=1)]).show()
```

実行結果



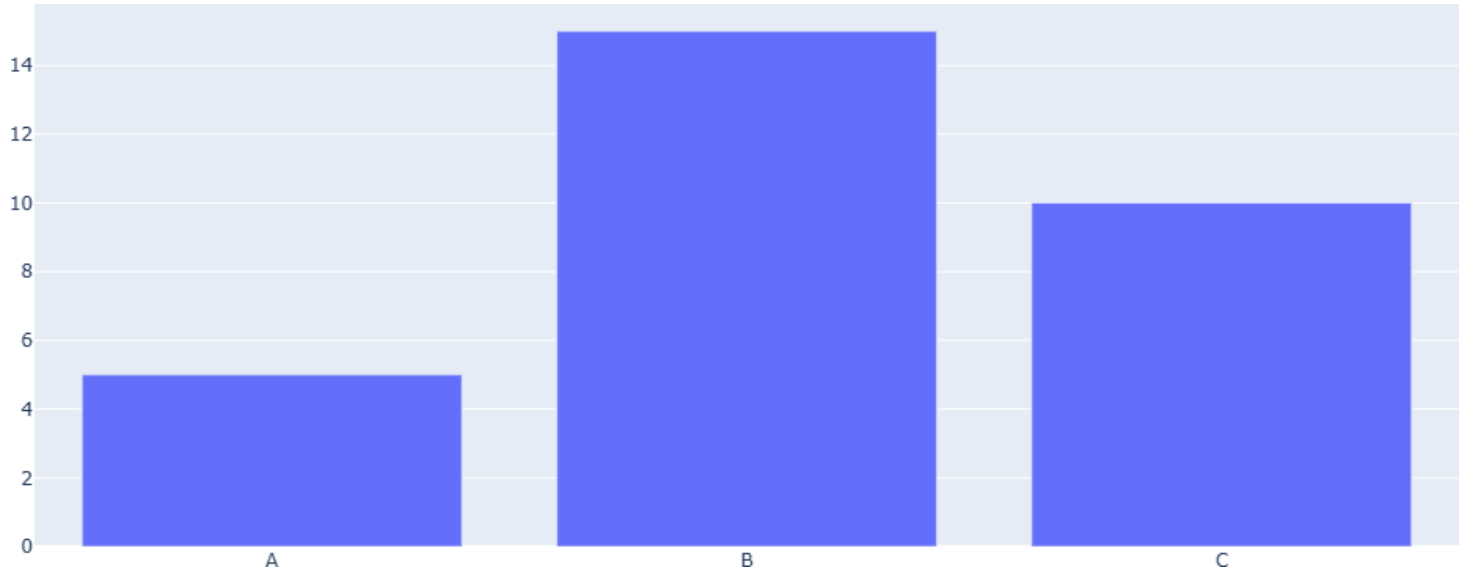
- 複数の項目の折れ線グラフを積み上げるには、積み上げるものを、カンマ（,）で区切って列挙し、四角括弧 [　] で囲えば良い
- この例では、面グラフを描くgo.Scatterを2つ、四角括弧の中に並べている

4. 棒グラフ

スクリプト4. 棒グラフ

```
import plotly.graph_objects as go
go.Figure(go.Bar(x=['A','B','C'],y=[5,15,10])).show()
```

実行結果



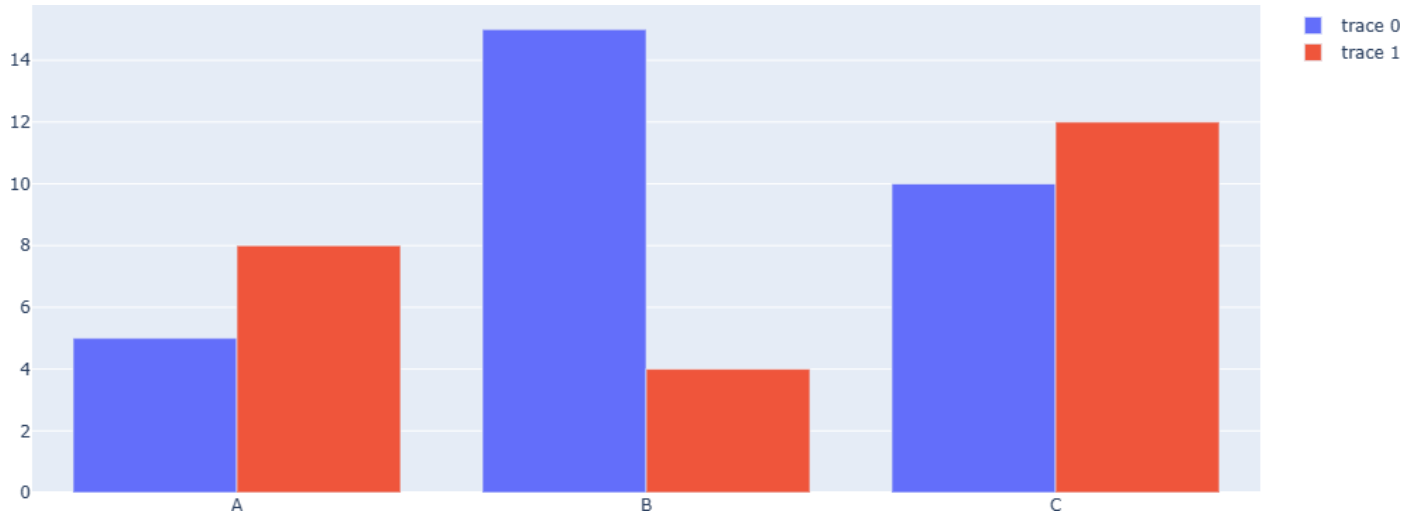
- 折れ線グラフのスクリプトでは、go.Scatter としていたが、棒グラフを描く際は、go.Bar を用いる
- 項目名を A,B,C とする際は、x=[A,B,C] ではなく、x=['A','B','C'] というように、文字をシングルクォーテーション「'」で囲む
- x=["A","B","C"] のように、文字をダブルクォーテーション「"」で囲んでも良い

5. 集合棒グラフ

スクリプト5.　集合棒グラフ

```
import plotly.graph_objects as go
go.Figure([go.Bar(x=['A','B','C'],y=[5,15,10]),
           go.Bar(x=['A','B','C'],y=[8,4,12])]).show()
```

実行結果



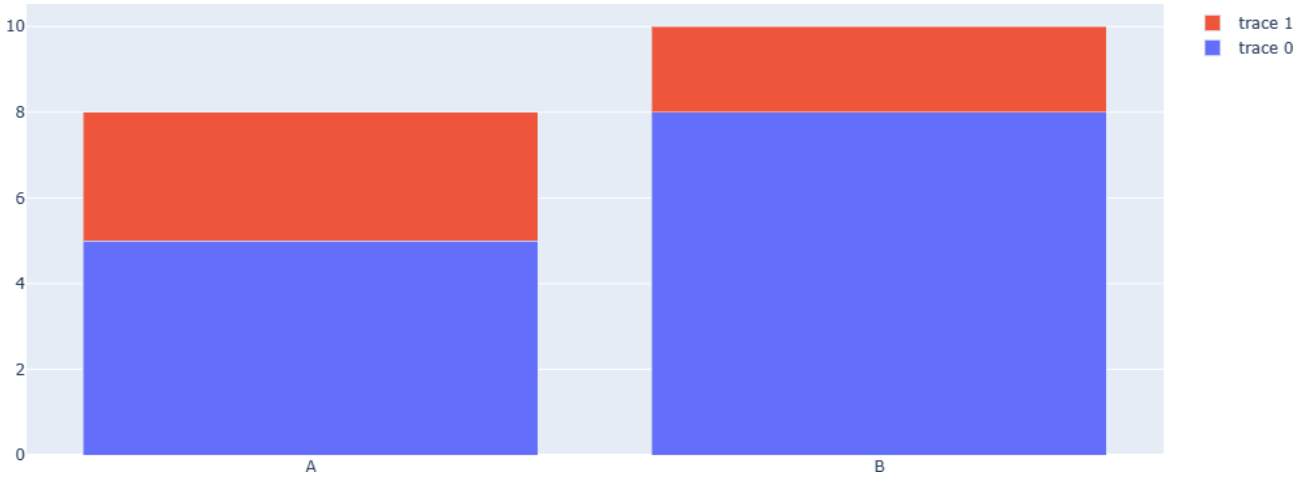
- 各項目に、複数の棒を並べて描きたい場合は、並べるものを、カンマ（,）で区切って列挙し、四角括弧[　]で囲えば良い
- この例では、青と赤の棒に対応して、2つの go.Bar を、四角括弧の中に並べている

6. 積み上げ棒グラフ

スクリプト6.　積み上げ棒グラフ

```
import plotly.graph_objects as go
go.Figure([go.Bar(x=['A','B'],y=[5,8]),go.Bar(x=['A','B'],y=[3,2])],
          layout=go.Layout(barmode='stack')).show()
```

実行結果



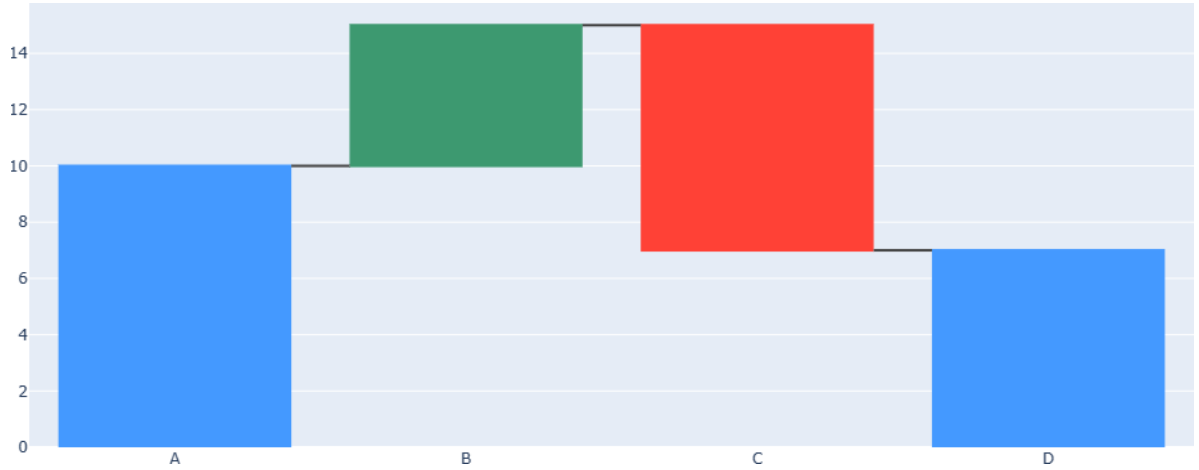
- 各項目に対応する複数の棒を、横に並べるのではなく、縦に積み上げる場合は、集合棒グラフと同様に記した上で、go.Figure の中で、layout=go.Layout(barmode='stack')
と指定すれば良い

7. 滝グラフ

スクリプト7. 滝グラフ

```
import plotly.graph_objects as go
go.Figure(go.Waterfall(x=['A','B','C','D'],y=[10,5,-8,7],
                       measure=['a','r','r','t'])).show()
```

実行結果



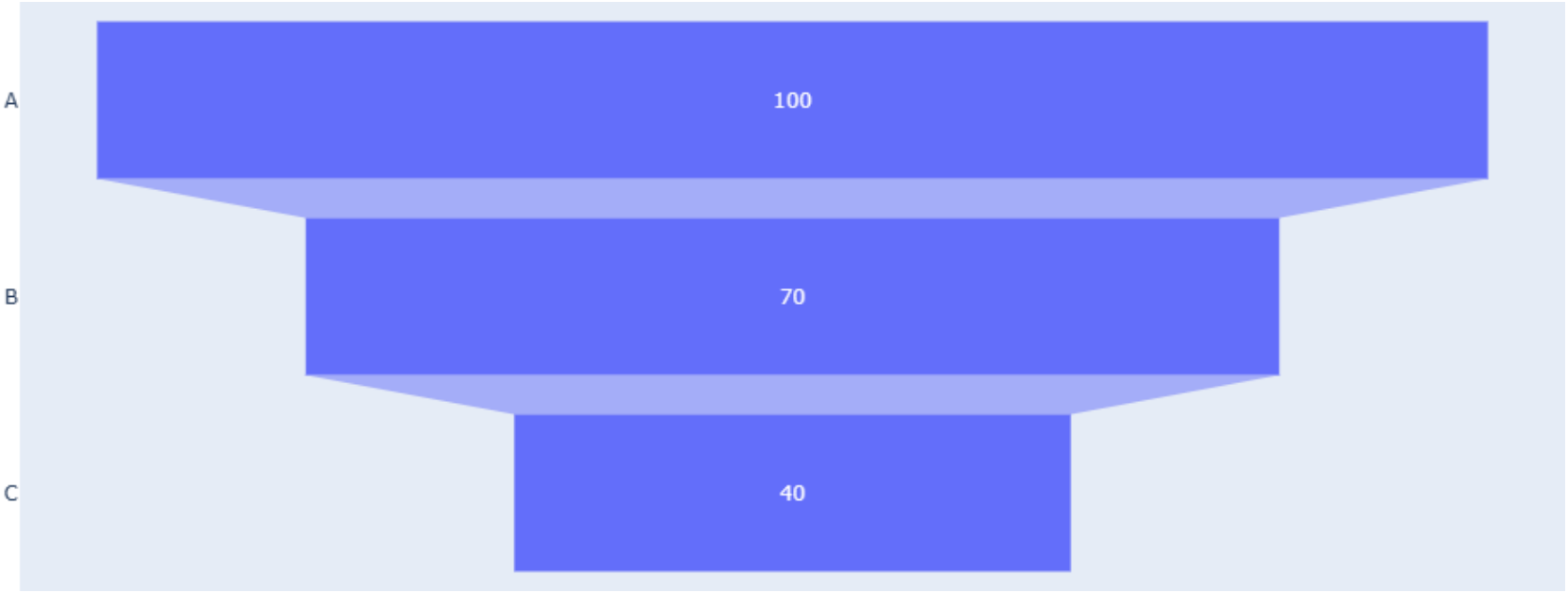
- 滝グラフを描くには go.Waterfall を用いる
- measure=['a','r','r','t'] の部分で、a は absolute（0を基準とした絶対的な値）、r は relative（直前からの増減）、t は total（総計）を意味する
- 丁寧に記すと、measure=['absolute','relative', 'relative','total'] となる

8. ファンネルチャート

スクリプト8. ファンネルチャート

```
import plotly.graph_objects as go
go.Figure(go.Funnel(x=[100,70,40],y=['A','B','C'])).show()
```

実行結果



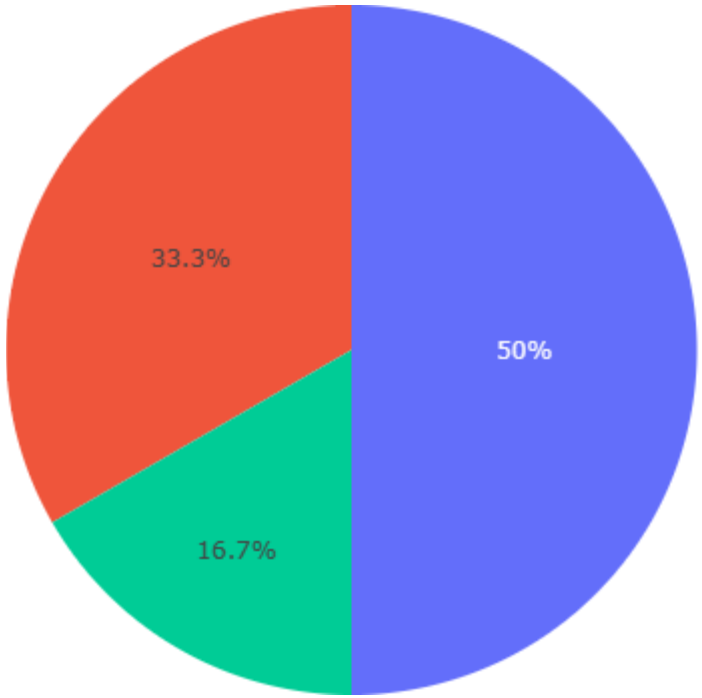
- ファンネルチャートを描くには go.Funnel を用いる
- ファンネルチャートでは、xの方に数値を記し、yの方に項目名を記す

9. 円グラフ

スクリプト9. 円グラフ

```
import plotly.graph_objects as go
go.Figure(go.Pie(labels=['A','B','C'],values=[5,15,10])).show()
```

実行結果



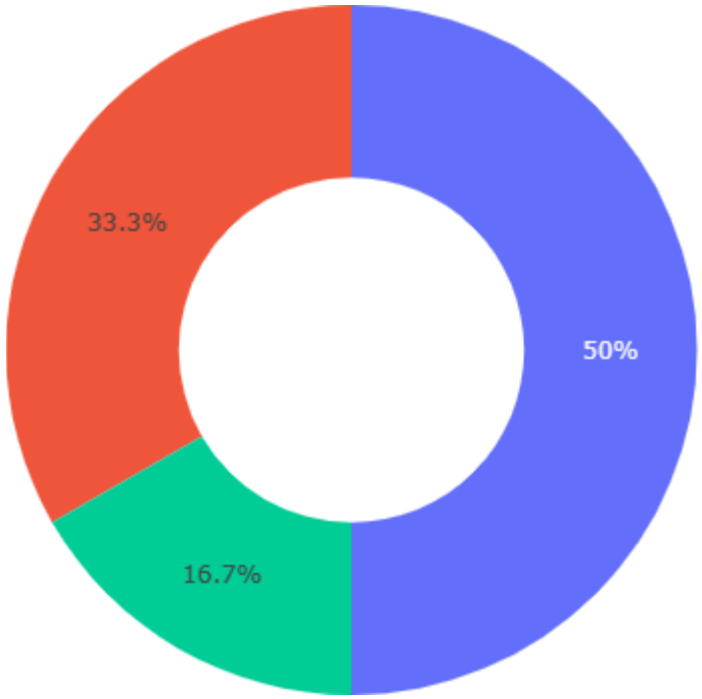
- 円グラフを描くには go.Pie を用いる
- 円グラフには、x軸（横軸）やy軸（縦軸）という概念がないので、項目名や数値を指定する際は、x= や y= と書くのではなく、labels= （項目名の指定）や values= （数値の指定）と書く

10. ドーナツグラフ

スクリプト10. ドーナツグラフ

```
import plotly.graph_objects as go
go.Figure(go.Pie(labels=['A','B','C'],values=[5,15,10],hole=0.5)).show()
```

実行結果



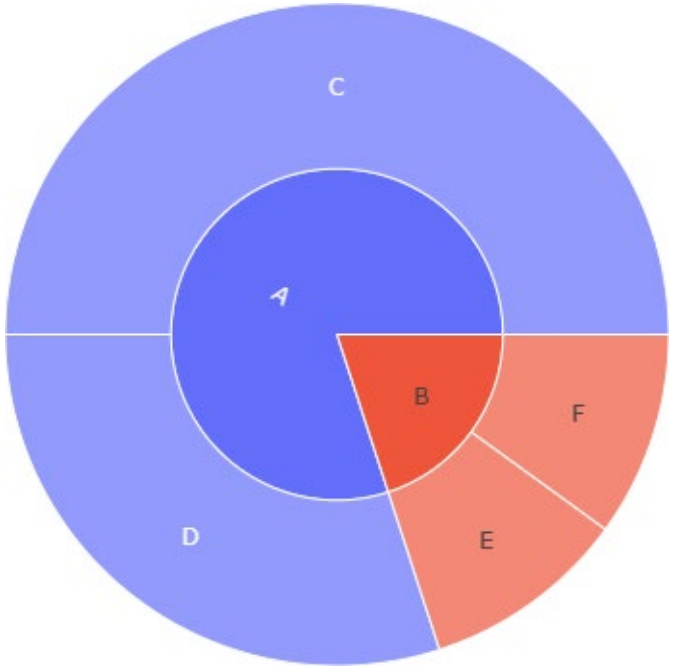
- 円グラフを描くスクリプトにおいて、 go.Pie の部分で、 hole=0.5 などとオプションを指定すると、円グラフの中央部分がくりぬかれて、ドーナツグラフになる
- hole= で指定する数値は、中央の穴の半径が、円全体の半径のどれだけの割合を占めるかを表す

11. サンバーストグラフ

スクリプト11. サンバーストグラフ

```
import plotly.graph_objects as go
go.Figure(go.Sunburst(labels=['A','B','C','D','E','F'],
                      parents=['','','A','A','B','B'],
                      values=[8,2,5,3,1,1],branchvalues='total')).show()
```

実行結果



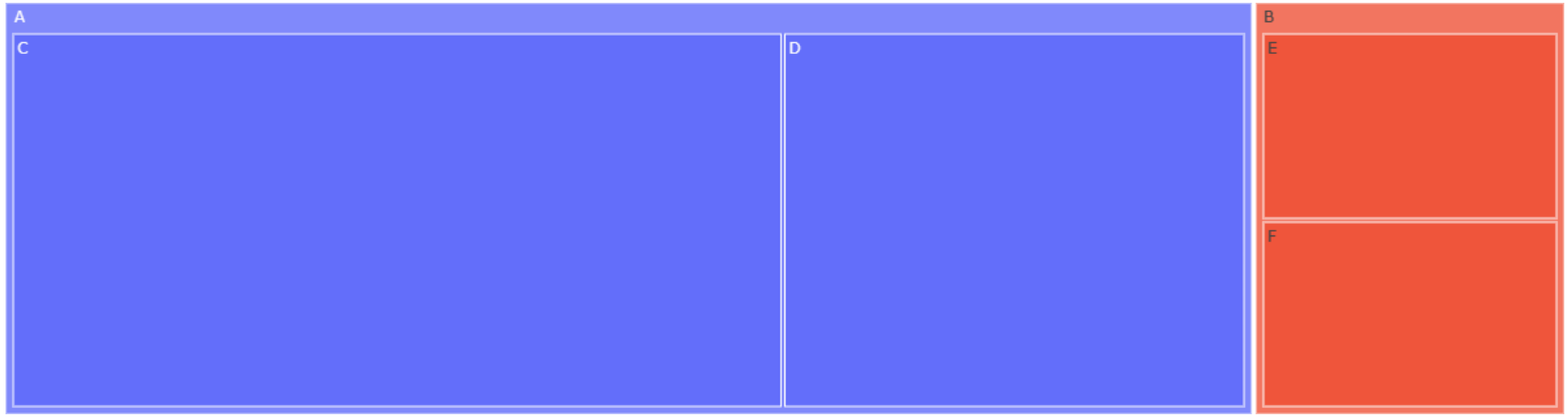
- サンバーストグラフを描くには go.Sunburst を用いる
- 各項目に対して、その親項目（上位の階層の項目）を parents= の欄に書き込む
- 一番上の階層にある項目については、親項目は空欄とする（クォーテーションマークのみ印す）
- この例では、AとBは一番上の階層にあり、CとDの親項目はAであり、EとFの親項目はBである
- branchvalues='total' という指定は、各項目の値が（親項目に占める割合ではなく）各項目の値そのものであることを示す

12. ツリーマップ

スクリプト12. ツリーマップ

```
import plotly.graph_objects as go
go.Figure(go.Treemap(labels=['A','B','C','D','E','F'],
                     parents=['','','A','A','B','B'],
                     values=[8,2,5,3,1,1],branchvalues='total')).show()
```

実行結果



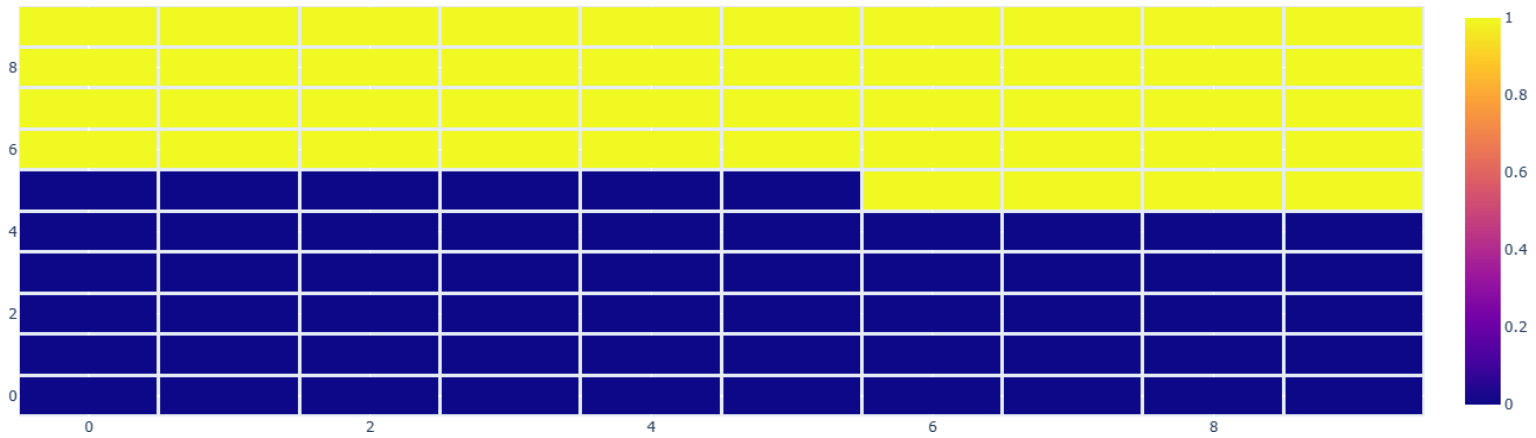
- ツリーマップを描くには go.Treemap を用いる
- 書式は、サンバーストグラフの場合と同様である

13. ワッフルチャート

スクリプト13. ワッフルチャート

```
import plotly.graph_objects as go
import numpy as np
p=np.zeros(100)
p[56:]=1
go.Figure(go.Heatmap(z=p.reshape(10,10),xgap=3,ygap=3)).show()
```

実行結果



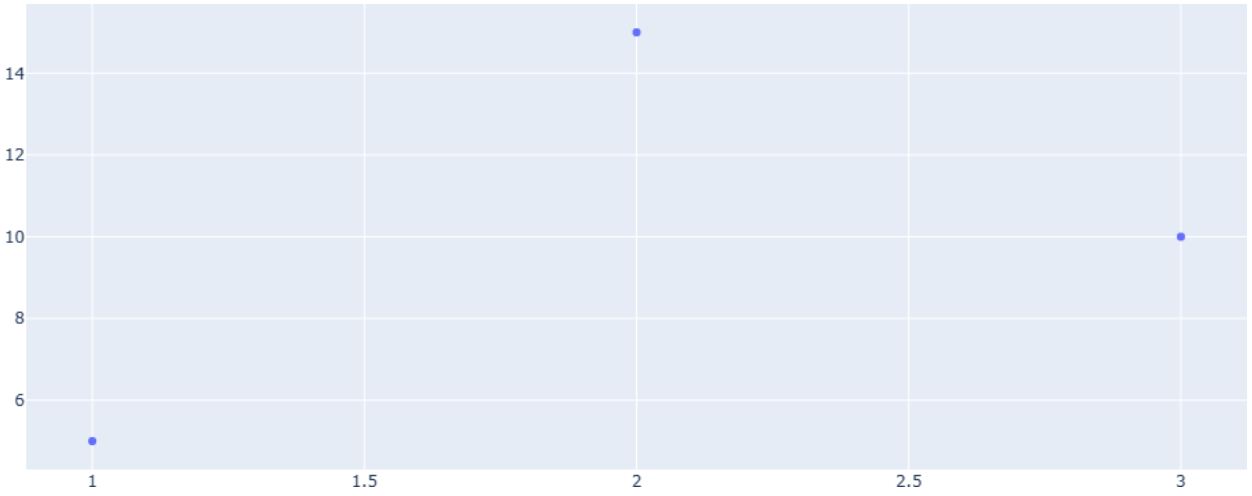
- 2023年3月現在、plotlyには、ワッフルチャート専用のツールは用意されていないので、ヒートマップを描くための go.Heatmap で代用する
- p[56:]=1 と指定すると、56%分（100個の四角のうち、56個）が青色で塗りつぶされる

14. 散布図

スクリプト14. 散布図

```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],mode='markers')).show()
```

実行結果



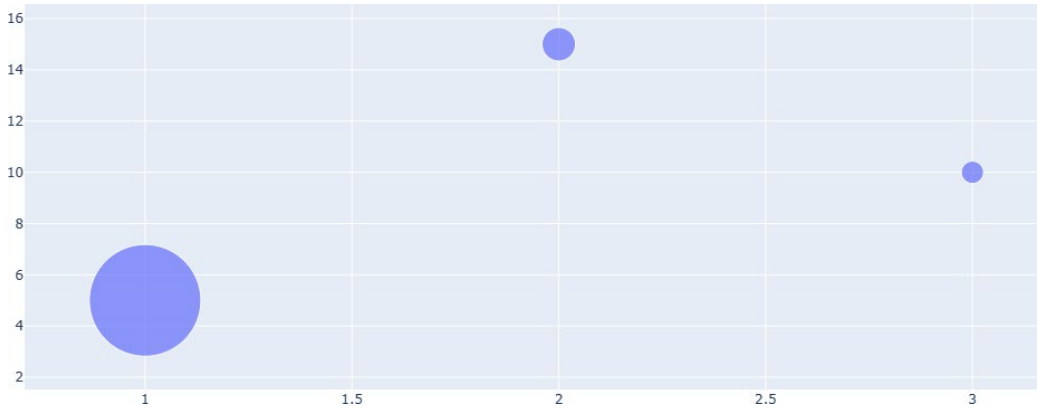
- 散布図を描くには go.Scatter を用いる
- 書式は、折れ線グラフの場合と同じだが、go.Scatter のオプションで、mode='markers' と指定する必要がある
- 上記のオプションを指定しないと、折れ線グラフが描かれる

15. バブルチャート

スクリプト15. バルブチャート

```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],mode='markers',
                     marker_size=[100,30,20])).show()
```

実行結果



- バブルチャートを描くには、散布図と同様、go.Scatterを用いる
- 書式も、散布図と同様だが、go.Scatter のオプションで、marker_size=[100,30,20] などと、円の大きさを追記する
- デフォルトでは、円の「大きさ」は「半径」に設定されている（marker_sizemode='area'で面積に変更可）

円の「大きさ」の設定を「面積」に変更

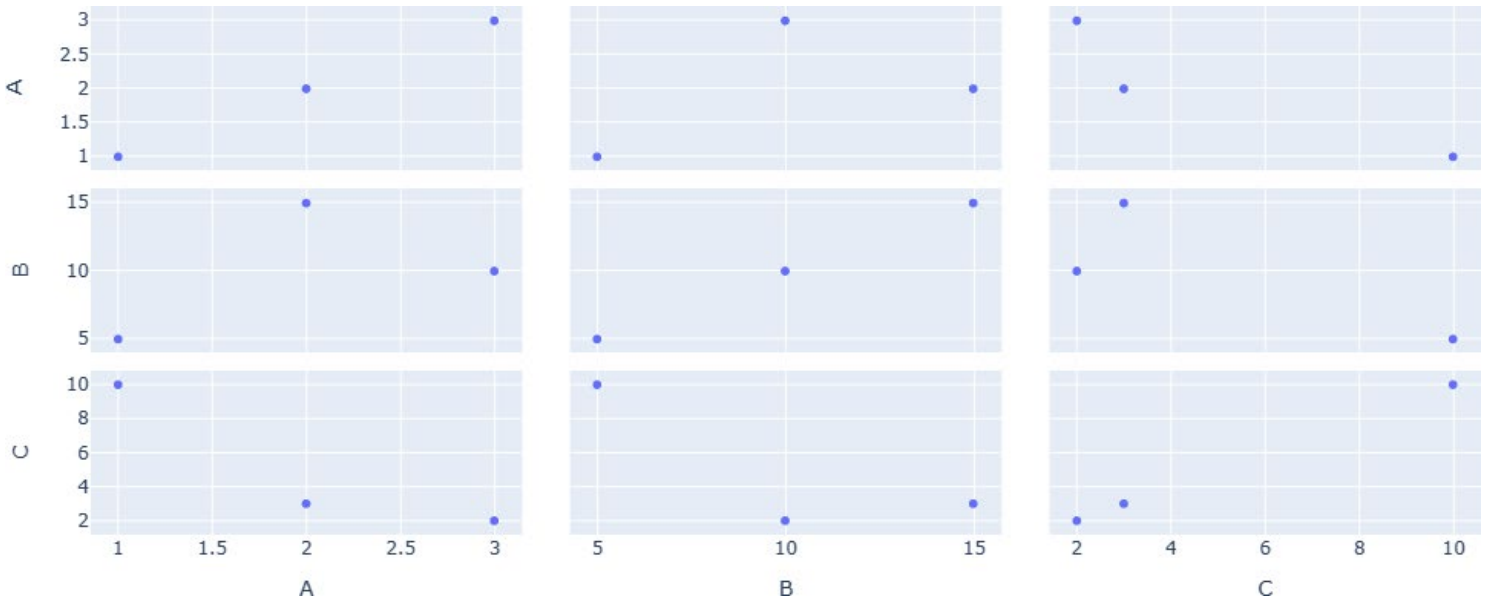
```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],mode='markers',
                     marker_size=[100,30,20],
                     marker_sizemode='area')).show()
```

16. 散布図行列

スクリプト16. 散布図行列

```
import plotly.graph_objects as go
go.Figure(go.Splom(dimensions=[dict(label='A',values=[1,2,3]),
                               dict(label='B',values=[5,15,10]),
                               dict(label='C',values=[10,3,2])])).show()
```

実行結果



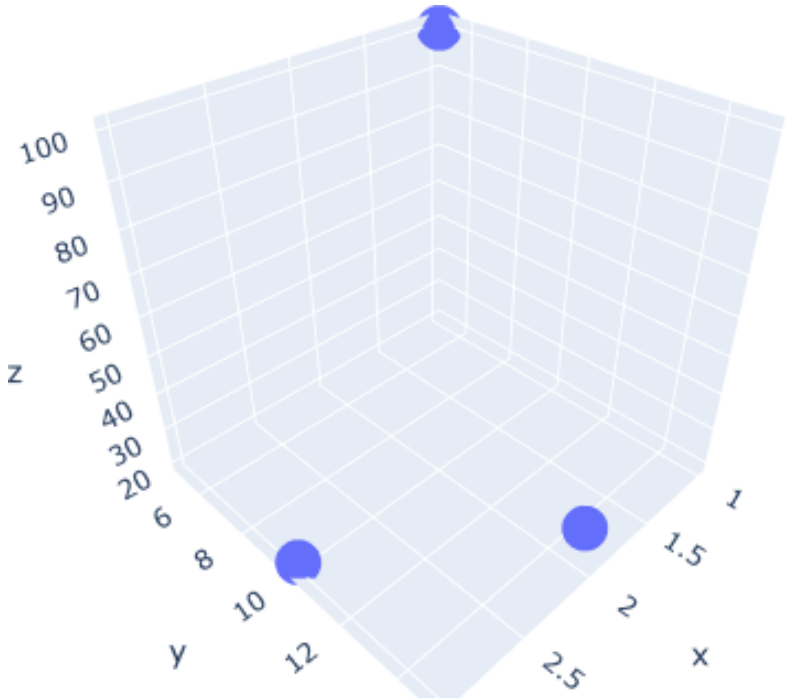
- 散布図行列を描くには、go.Splom を用いる（Splom は、Scatter plot matrix の略）
- go.Splom の中の label= の欄に項目名（この例では、A,B,C）を記し、values= の欄に数値を記す
- この例では、(A,B,C)=(1,5,10), (2,15,3), (3,10,2) という3つの点が図示される

17. 3次元散布図

スクリプト17． 3次元散布図

```
import plotly.graph_objects as go
go.Figure(go.Scatter3d(x=[1,2,3],y=[5,15,10],z=[100,30,20],mode='markers')).show()
```

実行結果



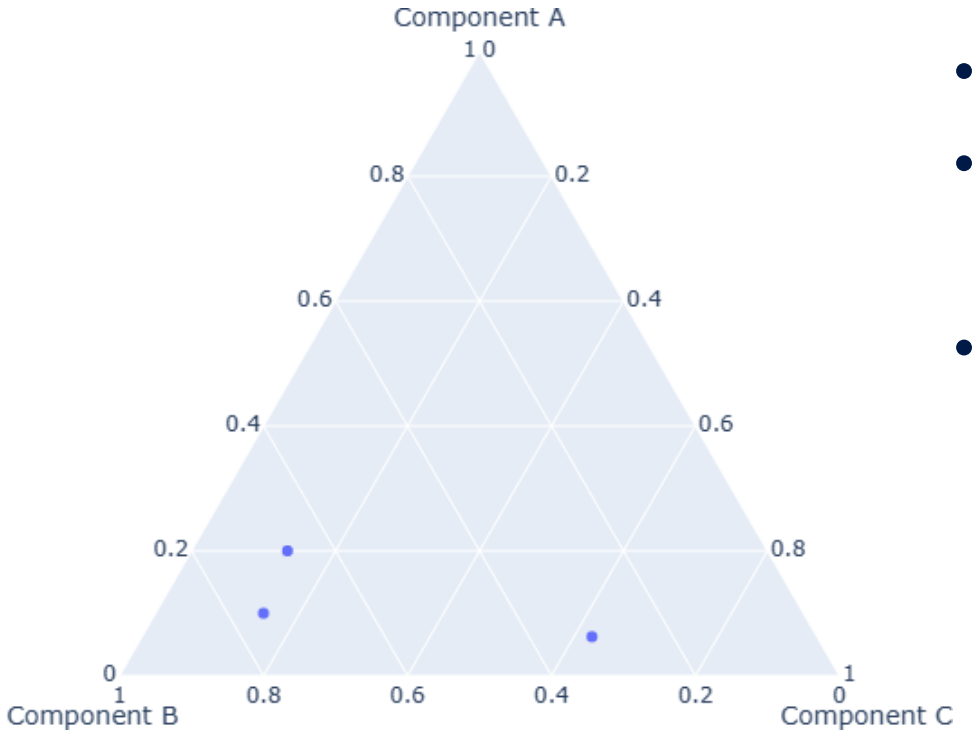
- 3次元散布図を描くには、go.Scatter3d を用いる
- 書式は、通常の（2次元の）散布図と同様だが、3次元目（z座標）の値を、z=[100,30,20] のように追記する
- 2次元の場合と同様、go.Scatter3d のオプションの mode='markers' の部分を削除すると、3次元の折れ線グラフになる

18. 三角図

スクリプト18. 三角図

```
import plotly.graph_objects as go
go.Figure(go.Scatterternary(a=[1,2,3],b=[5,15,10],c=[10,3,2],mode='markers')).show()
```

実行結果



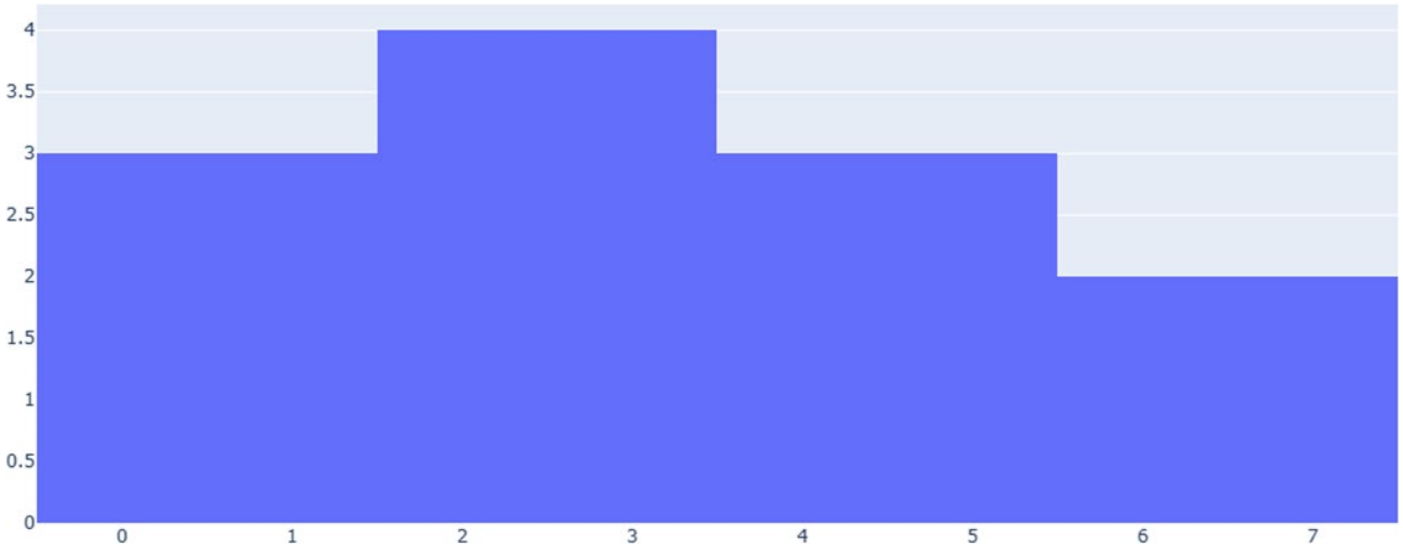
- 三角図を描くには、go.Scatterternary を用いる
- 書式は、3次元散布図と同様だが、座標の名称には、x、y、z ではなく、a、b、c を用いる
- 3次元散布図の場合と同様、go.Scatterternary のオプションの mode='markers' の部分を削除すると、三角図上の折れ線グラフ（点を直線で結んだ図）になる

19. ヒストグラム

スクリプト19. ヒストグラム

```
import plotly.graph_objects as go
go.Figure(go.Histogram(x=[1,4,3,7,4,3,1,0,3,2,5,6])).show()
```

実行結果



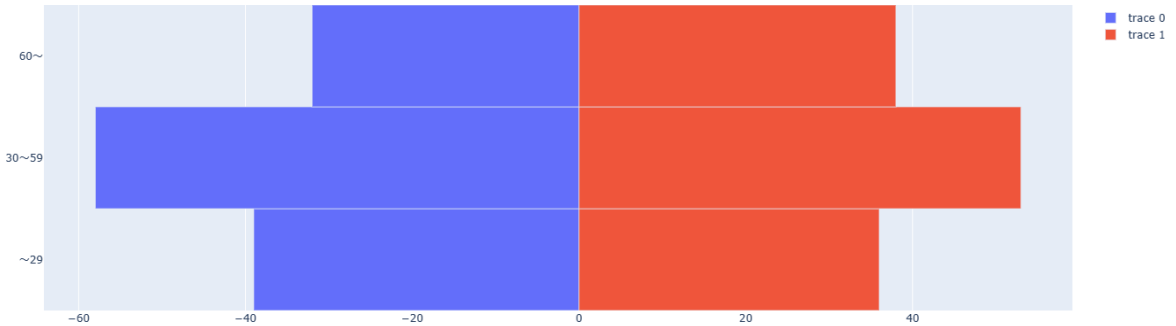
- ヒストグラムを描くには、go.Histogram を用いる
- データを go.Histogram の x= の部分に書き込む
- ヒストグラムの階級の幅は、明示的に指定しない場合、自動的に設定される
- この例では、−0.5以上1.5未満、1.5以上3.5未満、3.5以上5.5未満、5.5以上7.5未満という4つの階級が設定される

20. 人口ピラミッド

スクリプト20.　人口ピラミッド

```
import plotly.graph_objects as go
go.Figure([go.Bar(y=['～29','30～59','60～'],x=[-39,-58,-32],orientation = 'h'),
           go.Bar(y=['～29','30～59','60～'],x=[36,53,38],orientation = 'h')],
          layout=go.Layout(barmode = 'relative',bargap = 0.0)).show()
```

実行結果



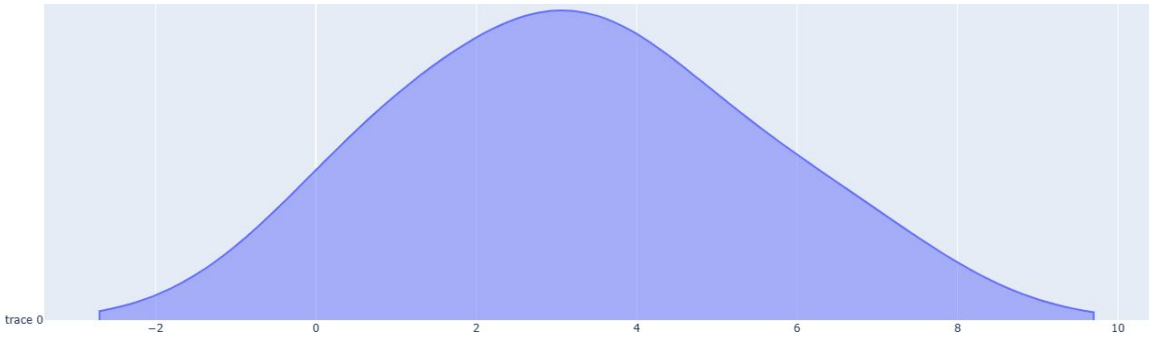
- 2023年3月現在、plotlyには、人口ピラミッド専用のツールは用意されていないので、棒グラフを描くための go.Bar で代用する
- go.Bar で、orientation = 'h' と指定することで、棒が横向きに描かれる
- 左側の青い棒を、左側に向かって描くために、x=[-39,-58,-32] というように、数値にマイナスを付けている（x=[39,58,32] だと、青い棒も、右側に向かって描かれてしまう）

21. 密度プロット

スクリプト21. 密度プロット

```
import plotly.graph_objects as go
go.Figure(go.Violin(x=[1,4,3,7,4,3,1,0,3,2,5,6],side='positive')).show()
```

実行結果



- 密度プロットを描くには、バイオリン図を描くための go.Violin を用いる
- 書式は、ヒストグラムを描く場合と同様だが、go.Violin のオプションとして、side='positive' を指定する
- 上記のオプションを指定しない場合、密度プロットを上下対称に繋げたバイオリン図になる
- 密度プロットを上方向ではなく、右方向に描きたい場合は、x= の部分を、y= に変えれば良い

22. 箱ひげ図

スクリプト22. 箱ひげ図

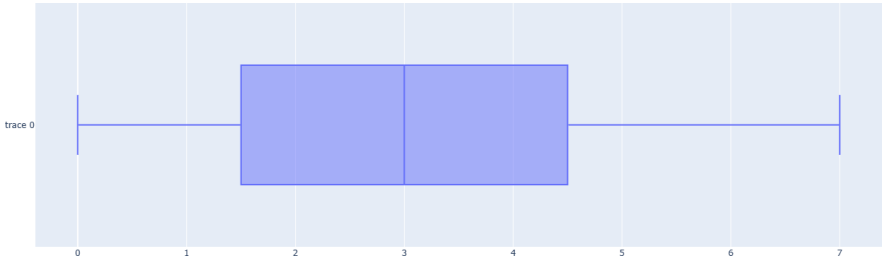
```
import plotly.graph_objects as go
go.Figure(go.Box(y=[1,4,3,7,4,3,1,0,3,2,5,6])).show()
```

実行結果

- 箱ひげ図を描くには、go.Box を用いる
- 書式は、ヒストグラムを描く場合と同様だが、箱を縦向き（y軸方向）に描きたい場合は、データを y= で記す

※データを x= で記すと、箱が横向き（x軸方向）に描かれる

```
import plotly.graph_objects as go
go.Figure(go.Box(x=[1,4,3,7,4,3,1,0,3,2,5,6])).show()
```



23. バイオリン図

スクリプト23. バイオリン図

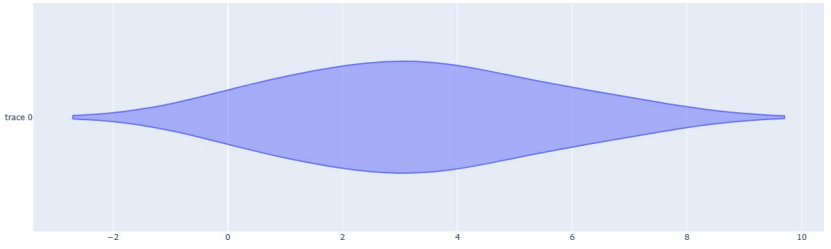
```
import plotly.graph_objects as go
go.Figure(go.Violin(y=[1,4,3,7,4,3,1,0,3,2,5,6])).show()
```

実行結果

- バイオリン図を描くには、go. Violin を用いる
- 書式は、ヒストグラムを描く場合と同様だが、バイオリンを縦向き（y軸方向）に描きたい場合は、データを y= で記す

※データを x= で記すと、バイオリンが横向き（x軸方向）に描かれる

```
import plotly.graph_objects as go
go.Figure(go.Violin(x=[1,4,3,7,4,3,1,0,3,2,5,6])).show()
```

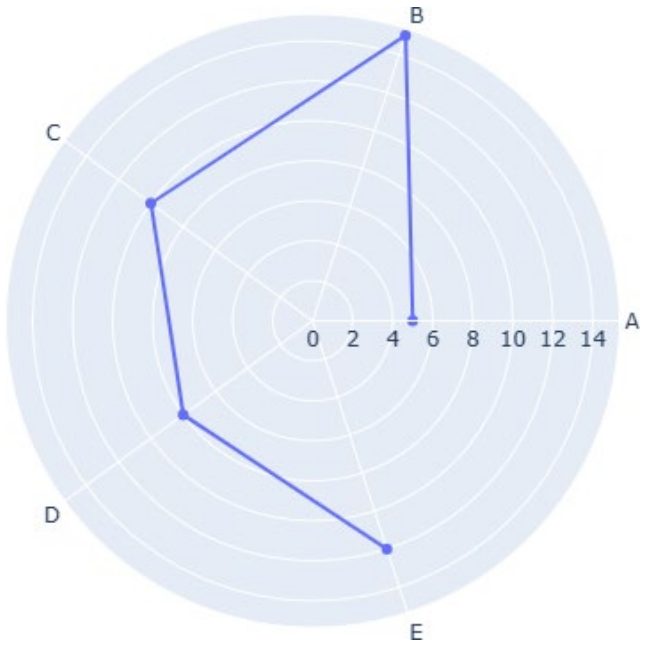


24. レーダーチャート

スクリプト24. レーダーチャート

```
import plotly.graph_objects as go
go.Figure(go.Scatterpolar(theta=['A','B','C','D','E'],r=[5,15,10,8,12])).show()
```

実行結果



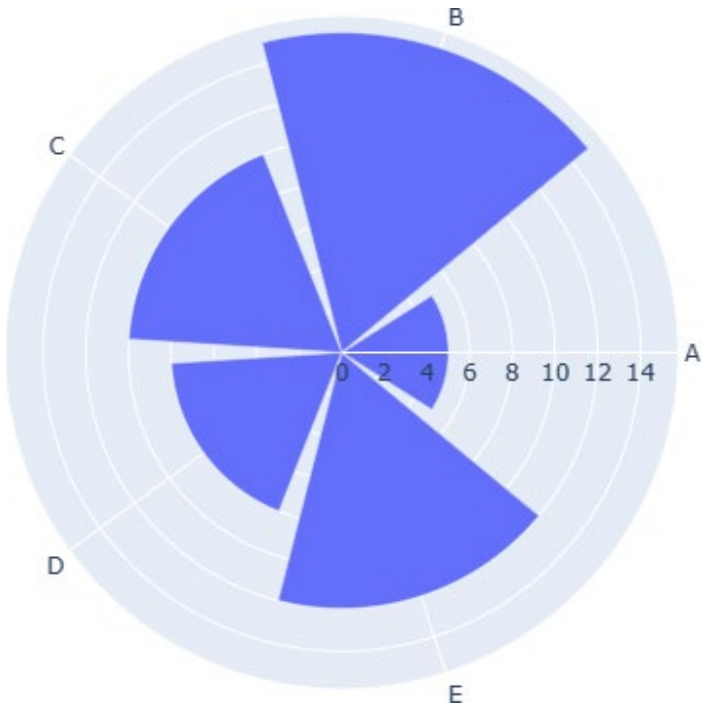
- レーダーチャートを描くには、go.Scatterpolar を用いる
- theta= で項目名を指定し、 r= に、各項目の数値を記す
- デフォルトでは、先頭の項目（この例では A）に対応する点と、末尾の項目（この例では E）に対応する点は、結ばれず、線が閉じない
- 線を閉じたい場合は、theta=['A','B','C','D','E','A'],r=[5,15,10,8,12,5] のように、先頭の項目を、末尾にもう一度記すと良い

25. ポーラーチャート

スクリプト25. ポーラーチャート

```
import plotly.graph_objects as go
go.Figure(go.Barpolar(theta=['A','B','C','D','E'],r=[5,15,10,8,12])).show()
```

実行結果



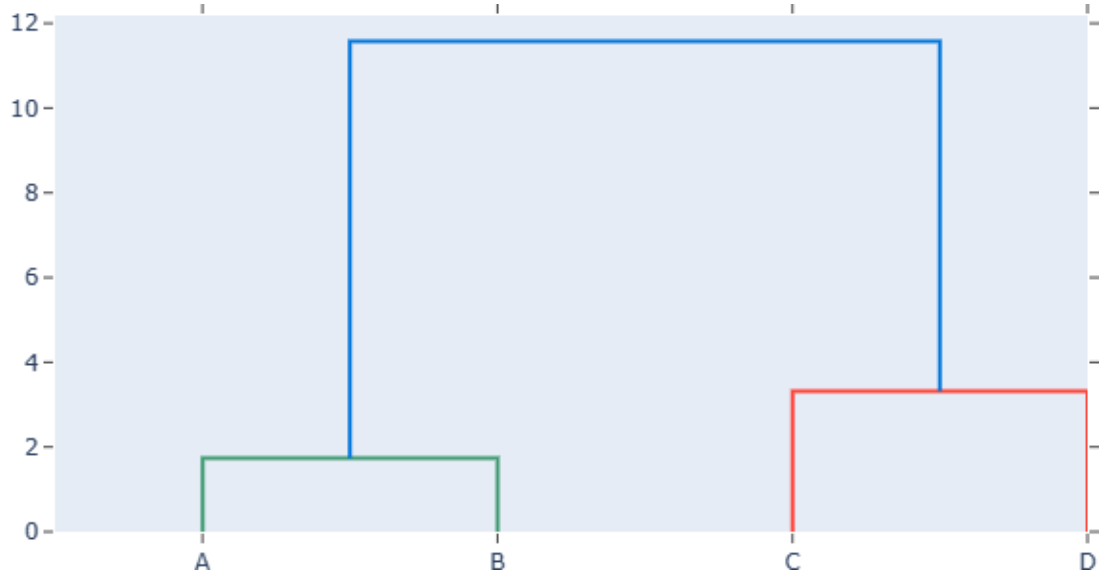
- ポーラーチャートを描くには、go.Barpolar を用いる
- 書式は、基本的にレーダーチャートのときと同様である
- theta= で項目名を指定し、 r= に、各項目の数値を記す
- theta は数学において、円の中心角をギリシャ文字のシータで表すことに由来する
- r はradius（半径）の頭文字であることから、数学では、中心（原点）からの距離を表す際などに用いられる

26. デンドログラム

スクリプト26. デンドログラム

```
import plotly.figure_factory as ff
import numpy as np
d=[[1,2,3],[2,3,2],[9,8,6],[8,9,9]]
ff.create_dendrogram(np.array(d),labels=['A','B','C','D']).show()
```

実行結果



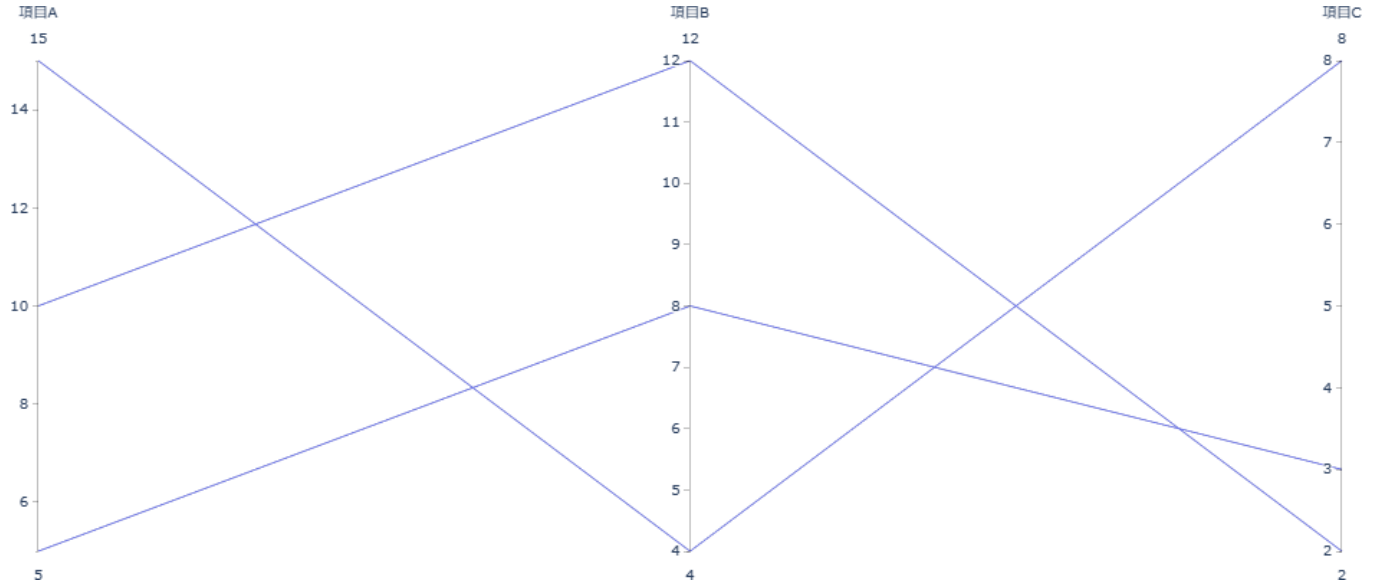
- デンドログラムを描くには、plotly.figure_factory （ff と命名）の ff.create_dendrogram を用いるのが手軽である
- この例では、A、B、C、D 4つの要素を、それらに対するデータ [1,2,3]、[2,3,2]、[9,8,6]、[8,9,9] に基づき、系統樹のように、階層的にグループ分けしている

27. 平行座標プロット

スクリプト27. 平行座標プロット

```
import plotly.graph_objects as go
go.Figure(go.Parcoords(dimensions=[dict(label='項目A',values=[5,15,10]),
                                   dict(label='項目B',values=[8,4,12]),
                                   dict(label='項目C',values=[3,8,2])])).show()
```

実行結果



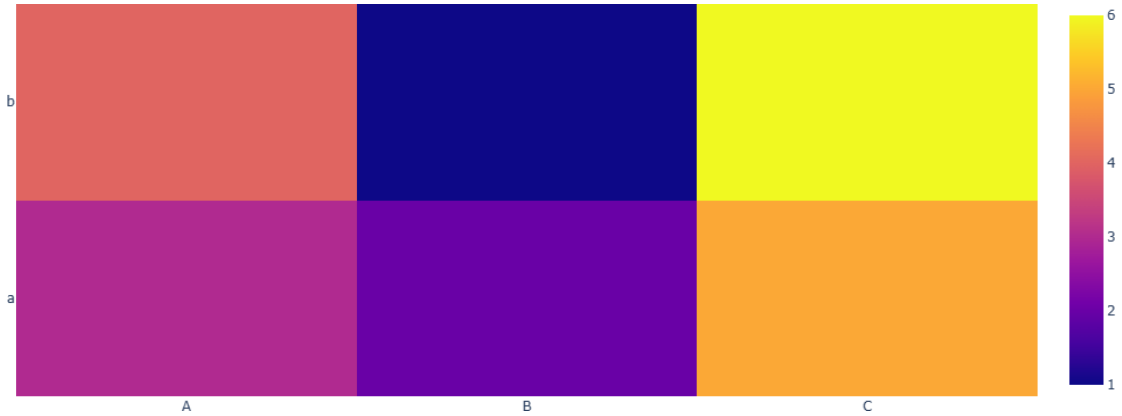
- 平行座標プロットを描くには、go.Parcoords を用いる
- 書式は散布図行列と同様である
- go.Parcoords の中の label= の欄に項目名（この例では、項目A,項目B,項目C）を記し、values= の欄に各項目の数値を記す

28. ヒートマップ

スクリプト28. ヒートマップ

```
import plotly.graph_objects as go
go.Figure(go.Heatmap(x=['A','B','C'],y=['a','b'],z=[[3,2,5],[4,1,6]])).show()
```

実行結果



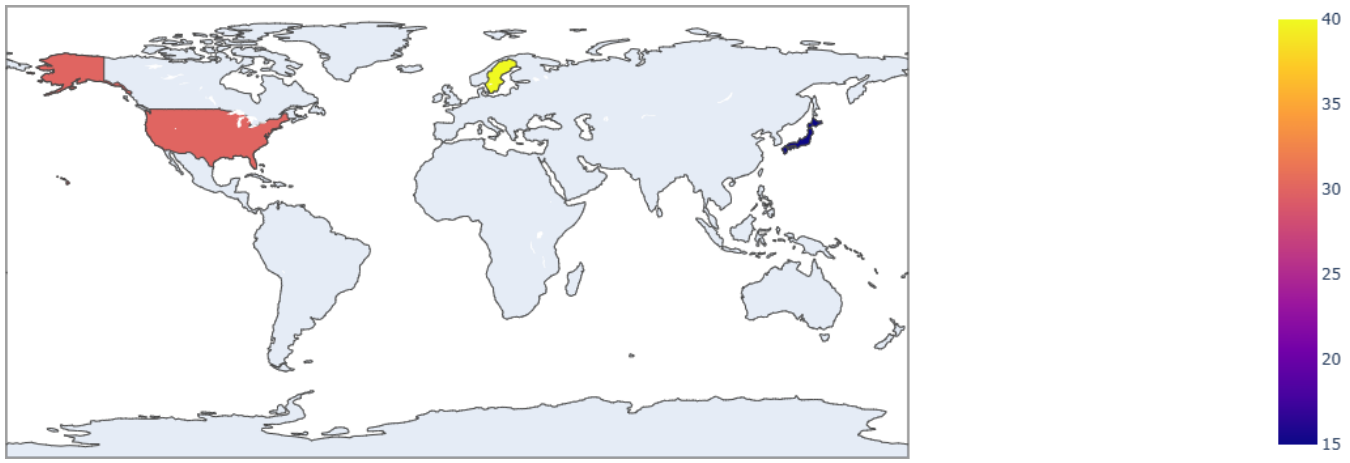
- ヒートマップを描くには、go.Heatmap を用いる
- go. Heatmap の中の x= の欄に横方向の項目名（この例では、A,B,C）、y= の欄に縦方向の項目名（この例では、a,b）、z= の欄に各マス目の数値を記す
- この例では、例えば (A,a)のマス目の値は3、(B,a) のマス目の値は2, (C,a) のマス目の値は5 、(A,b)のマス目の値は4、(B,b) のマス目の値は1, (C,b) のマス目の値は6 となっている

29. 階級区分図

スクリプト29. 階級区分図

```
import plotly.graph_objects as go
go.Figure(go.Choropleth(locations=['JPN','SWE','USA'],z=[15,40,30])).show()
```

実行結果



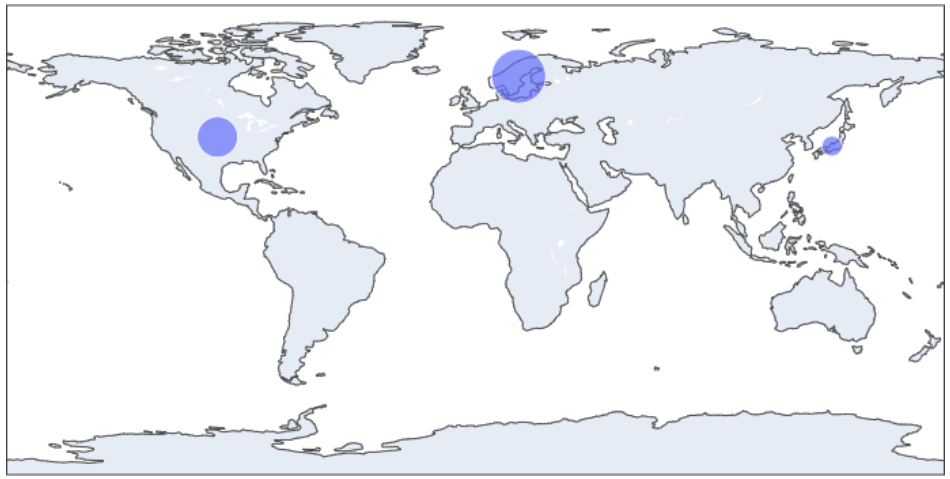
- 階級区分図を描くには、go.Choropleth を用いる
- go.Choropleth の中の locations= の欄に国名（この例では、日本、スウェーデン、米国）を表すコードを記し、z= の欄に各国の数値を記す
- デフォルトでは、国名にはISO 3166-1 alpha-3 と呼ばれる ISO（国際標準化機構）の規格を用いる

30. 比例シンボルマップ

スクリプト30. 比例シンボルマップ

```
import plotly.graph_objects as go
go.Figure(go.Scattergeo(locations=['JPN','SWE','USA'],marker_size=[15,40,30])).show()
```

実行結果



- 比例シンボルマップを描くには、go.Scattergeo を用いる
- go.Scattergeo の中の locations= の欄に国名（この例では、日本、スウェーデン、米国）を表すコードを記し、marker_size= の欄に各国の数値を記す
- 階級区分図と同様、デフォルトでは、国名にはISO 3166-1 alpha-3 と呼ばれる ISO（国際標準化機構）の規格を用いる
- デフォルトでは、円の「大きさ」は「半径」に設定されている（marker_sizemode='area'で「面積」に変更可）

円の「大きさ」の設定を「面積」に変更

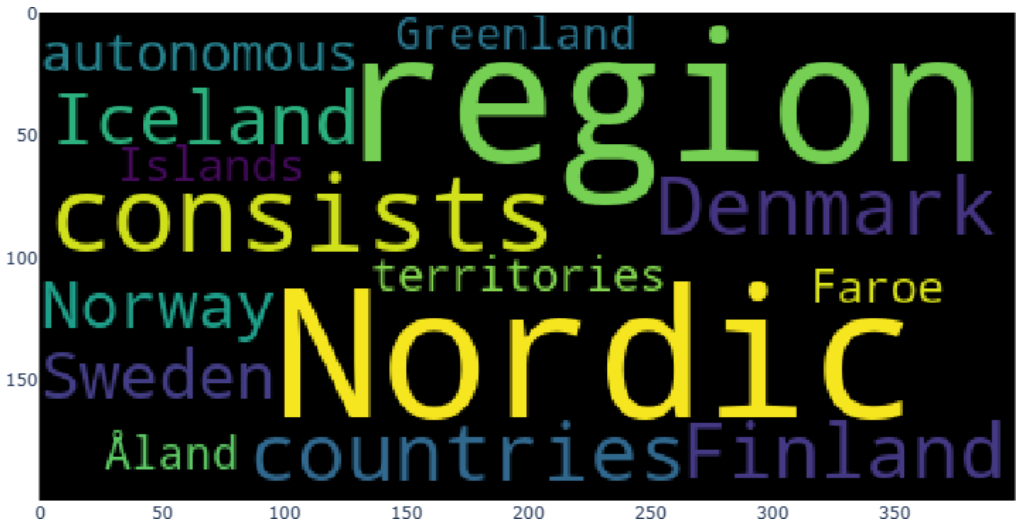
```
import plotly.graph_objects as go
go.Figure(go.Scattergeo(locations=['JPN','SWE','USA'],marker_size=[15,40,30],
                        marker_sizemode='area')).show()
```

31. ワードクラウド（おまけ）

スクリプト31. ワードクラウド

```
import plotly.graph_objects as go
from wordcloud import WordCloud
text='''Nordic region consists of 5 countries (Denmark, Finland, Iceland, Norway
and Sweden) and 3 autonomous territories (Faroe Islands, Greenland and Åland).'''
go.Figure(go.Image(z=WordCloud().generate(text))).show()
```

実行結果



- 2023年3月現在、plotlyには、ワードクラウドを描く専用のツールは用意されていないので、一旦、wordcloud パッケージを使って、自然言語（英語など）を処理し、それをplotlyの go.Image を用いて、go.Figureで表示可能な形式に変換する
- 日本語の文章からワードクラウドを生成する場合は、最初に文章を単語に切り分ける必要があるため、上記のスクリプトよりも、少し手間が増える

付録Bに続く

付録Bでは

- 文字の大きさ、線の色・太さなどを変更して、チャートを見やすくする
- ファイルからデータを読み込む
- チャートを html 形式（Web の標準形式）で保存する
- チャートを画像ファイルに保存する

などについて扱う

データ分析と可視化

付録B

愛媛大学　　松浦 真也

Ver. 2024.1.10

付録Aから続く

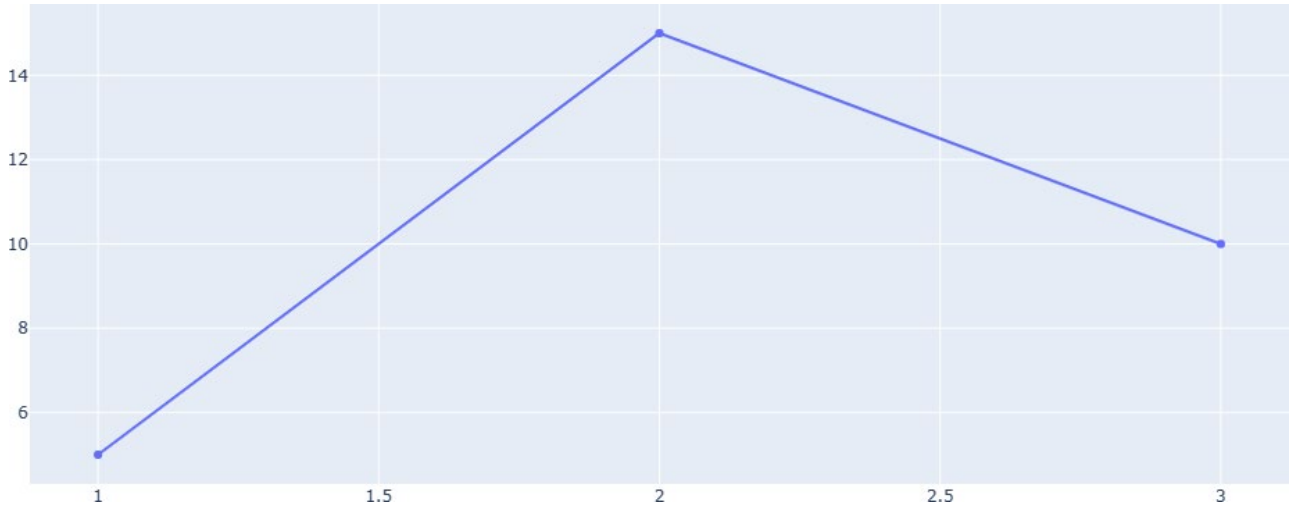
付録Aでは、Googleの無料サービスである Google Colaboratory 上で、Pythonの plotly ライブラリを用いてチャートを描画する方法の基本を紹介した。付録Bでは、以下を扱う。

- 文字の大きさ、線の色・太さなどを変更して、チャートを見やすくする
- ファイルからデータを読み込む
- チャートを html 形式（Web の標準形式）で保存する
- チャートを画像ファイルに保存する

細かい設定をするための準備

- 細かい設定をするために、スクリプトの書き方を少し変更する
- 折れ線グラフを例に説明する

実行結果



折れ線グラフ

```
import plotly.graph_objects as go
go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10])).show()
```

描画内容は同じ

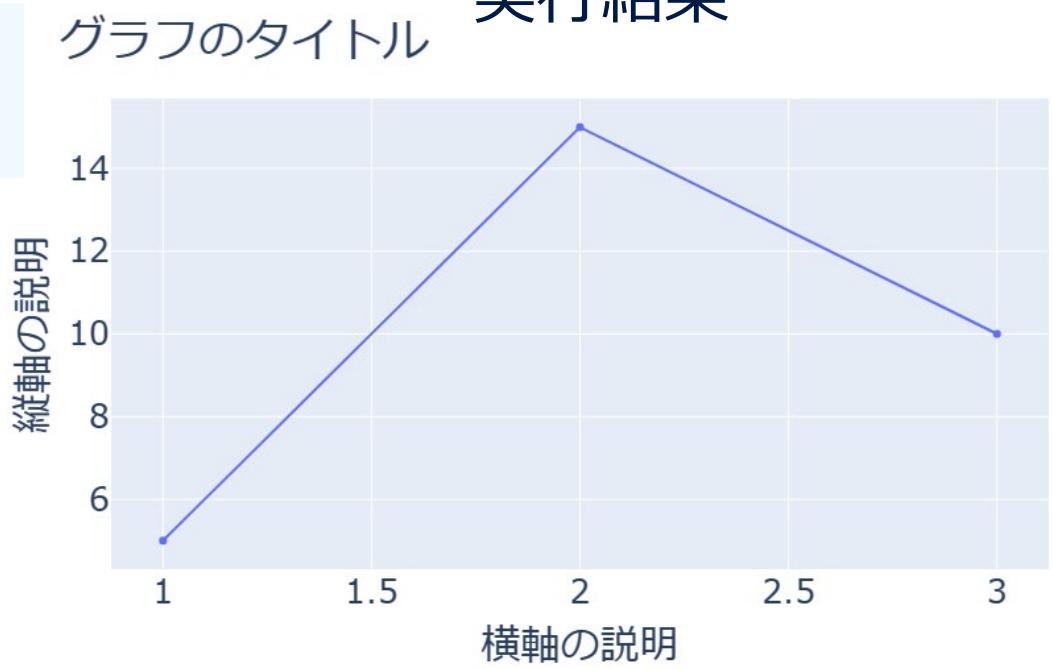
```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.show()
```

- 前編では、go.Figure で描いた図を、直ちに .show() を付けて表示していた
- 後編では、図に一旦、figと名前を付け、その後、fig.show()で図を表示する

基本的なレイアウトの調整

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.show()
```

実行結果



- 文字のサイズ（font_size）を指定
- グラフのタイトルを記入
- 横軸と縦軸の説明を記入
- 図の横幅（width）を指定
- 図の高さ（height）を指定
- 図の上下左右の余白（margin）を指定

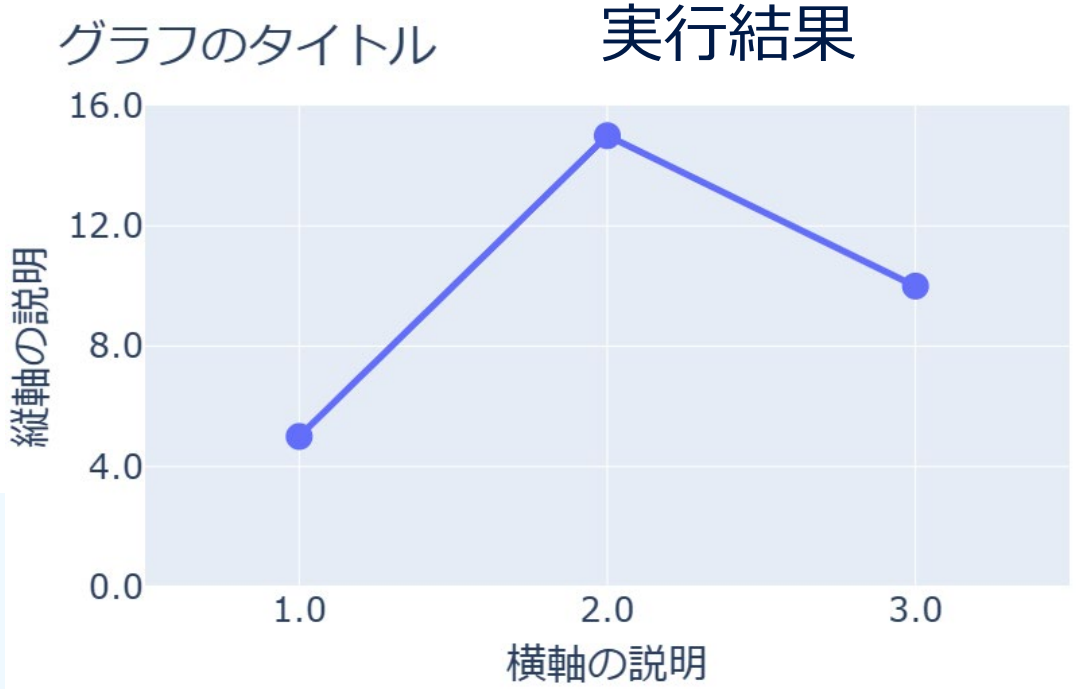
```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.update_layout(font_size=25,
                  title='グラフのタイトル',
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

t は上 (top)、l は左 (left)、r は右 (right)、b は下 (bottom) の余白

基本的なレイアウトの調整（続）

- 横軸 (xaxis) の描画範囲 (range) を指定
- 縦軸 (yaxis) の描画範囲 (range) を指定
- 縦軸と横軸の目盛りの間隔 (dtick) を設定
- 目盛りの書式 (tickformat) を設定
- 点 (marker) の大きさ (size) を指定
- 線 (line)　の太さ (width) を指定

実行結果




```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.update_layout(font_size=25,
                  title='グラフのタイトル',
                  xaxis=dict(title='横軸の説明',
                             range=(0.5,3.5),dtick=1,tickformat='.1f'),
                  yaxis=dict(title='縦軸の説明',
                             range=(0,16),dtick=4,tickformat='.1f'),
                  width=800,height=500,
                  margin = dict(t=65,l=10,r=10,b=10))
fig.update_traces(marker=dict(size=20),line=dict(width=5))
fig.show()
```

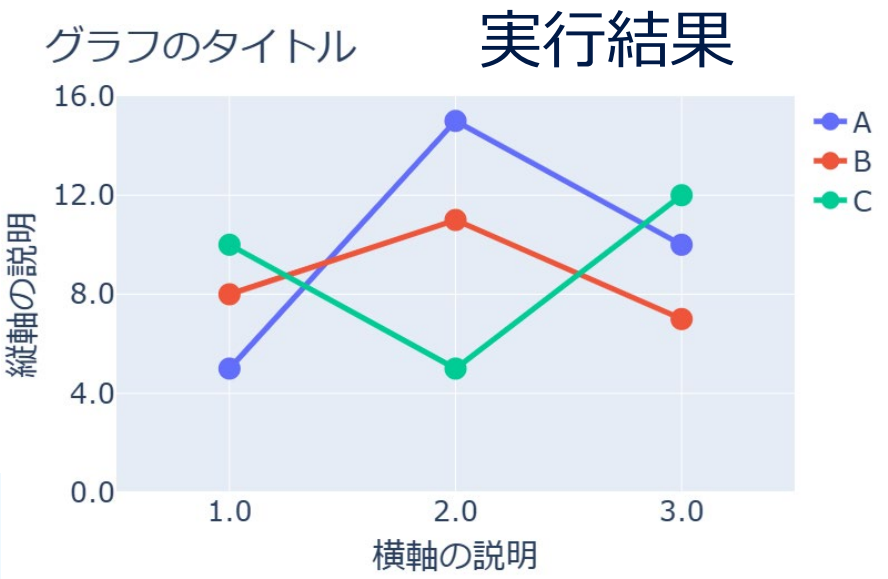
.1f で、小数点以下、1桁まで表示（.2f なら、小数点以下2桁まで）

複数のグラフを重ねる

- 折れ線を2本追加する
- 折れ線に名前 (name) を付ける（この例では、A、B、C）

※グラフを追加するときには、fig.add_trace を使う
※レイアウトの指定を変更・追加する際は、
fig.update_layout を使う

実行結果




```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],name='A'))
fig.add_trace(go.Scatter(x=[1,2,3],y=[8,11,7],name='B'))
fig.add_trace(go.Scatter(x=[1,2,3],y=[10,5,12],name='C'))
fig.update_layout(font_size=25,
                  title='グラフのタイトル',
                  xaxis=dict(title='横軸の説明',
                             range=(0.5,3.5),dtick=1,tickformat='.1f'),
                  yaxis=dict(title='縦軸の説明',
                             range=(0,16),dtick=4,tickformat='.1f'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.update_traces(marker=dict(size=20),line=dict(width=5))
fig.show()
```

自分好みのデザインに

- 文字、グラフ、点、線の色を変更（次ページ参照）
- タイトルをセンタリング（横方向の位置を x=0.5 と指定）

```
import plotly.graph_objects as go
c=['saddlebrown','red','blue','wheat']
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],name='A',
                         marker=dict(color=c[0]),line=dict(color=c[0])))
fig.add_trace(go.Scatter(x=[1,2,3],y=[8,11,7],name='B',
                         marker=dict(color=c[1]),line=dict(color=c[1])))
fig.add_trace(go.Scatter(x=[1,2,3],y=[10,5,12],name='C',
                         marker=dict(color=c[2]),line=dict(color=c[2])))
fig.update_layout(font_size=25,
                  title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明',
                             range=(0.5,3.5),dtick=1,tickformat='.1f'),
                  yaxis=dict(title='縦軸の説明',
                             range=(0,16),dtick=4,tickformat='.1f'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10),
                  plot_bgcolor=c[3])
fig.update_traces(marker=dict(size=20),line=dict(width=5))
fig.show()
```

自分好みのデザインに（結果）

下記のように書いて、グラフに使用する色を定義している

```
c=['saddlebrown','red','blue','wheat']
```

※このように書くと、c[0] で saddlebrown（サドルブラウン色）、c[1] で red（赤色）、c[2] で blue（青色）、c[3] で wheat（小麦色）を意味する

```
marker=dict(color=c[0])
```

※点 (marker) の色を、サドルブラウン色に指定

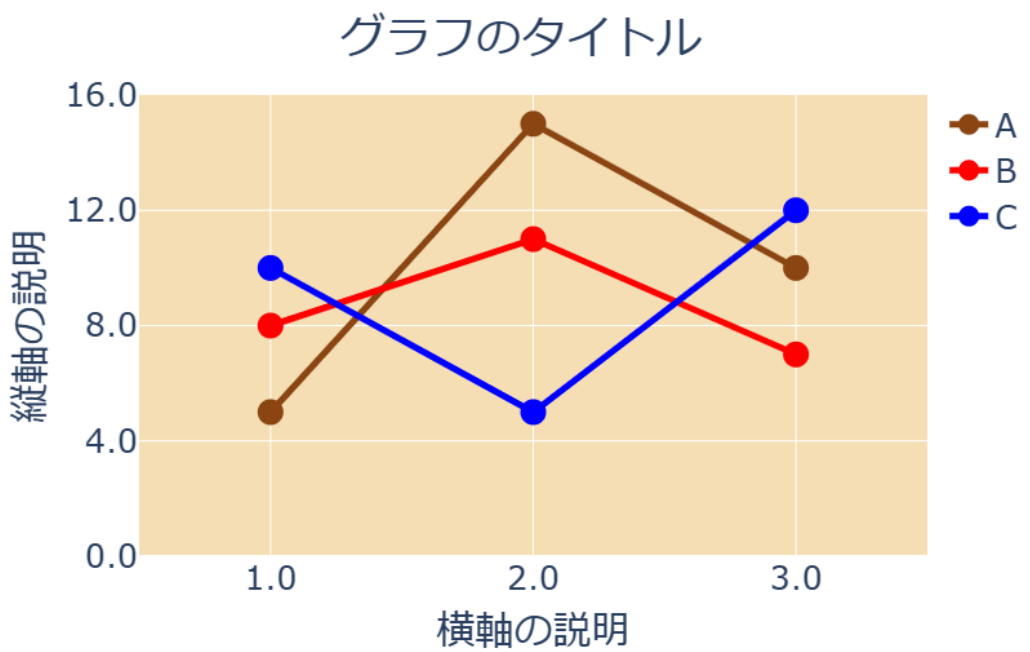
```
line=dict(color=c[0])
```

※線 (line) の色を、サドルブラウン色に指定

```
plot_bgcolor=c[3]
```

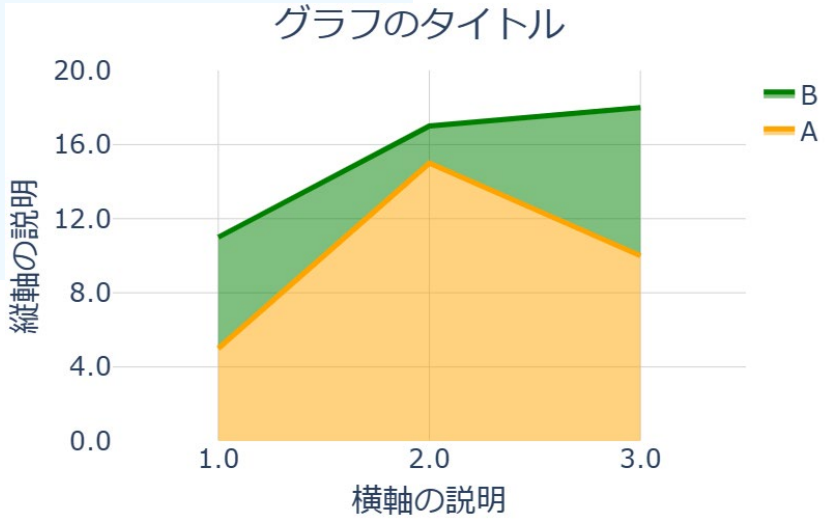
※背景の色 (bgcolor) を、小麦色に指定

実行結果



レイアウト調整（積み上げ面グラフ）

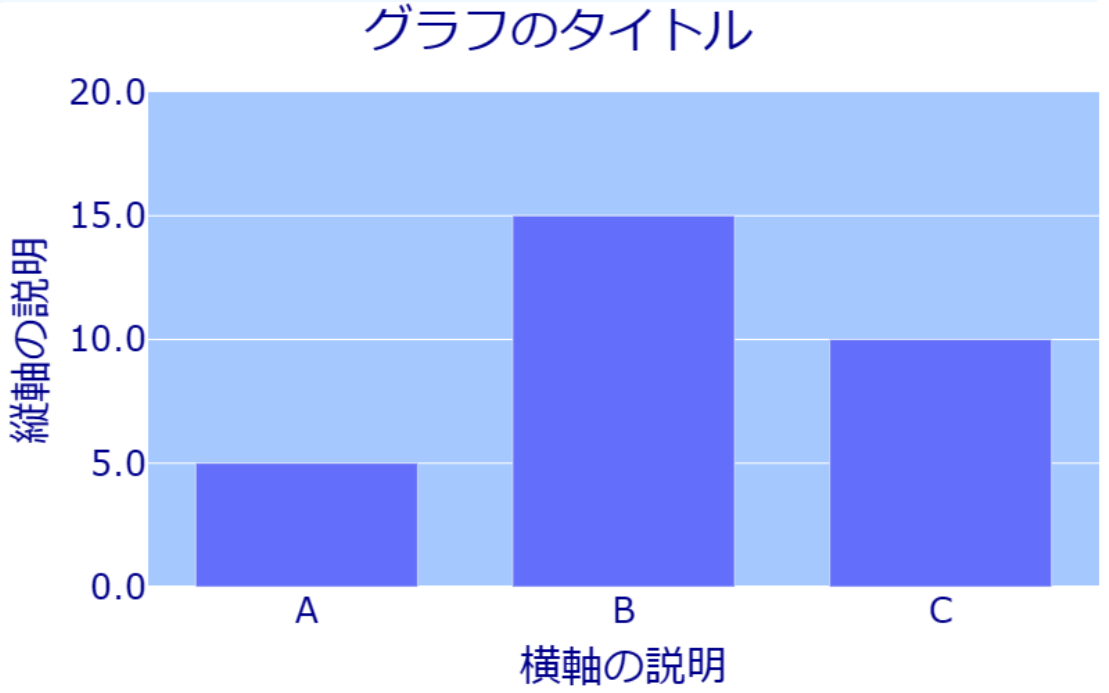
```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],stackgroup=1,
                         line=dict(color='orange'),name='A'))
fig.add_trace(go.Scatter(x=[1,2,3],y=[6,2,8],stackgroup=1,
                         line=dict(color='green'),name='B'))
fig.update_layout(font_size=25,
                  title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明',gridcolor='lightgrey',
                             range=(0.5,3.5),dtick=1,tickformat='.1f'),
                  yaxis=dict(title='縦軸の説明',gridcolor='lightgrey',
                             range=(0,20),dtick=4,tickformat='.1f'),
                  width=800,height=500,
                  margin=dict(t=65,l=10,r=10,b=10),
                  plot_bgcolor='white')
fig.update_traces(marker=dict(size=20),line=dict(width=5))
fig.show()
```



折れ線グラフの場合と同様にして、細かなレイアウトを調整可能
（以下、詳細な説明は省略し、スクリプトの例を列挙する）

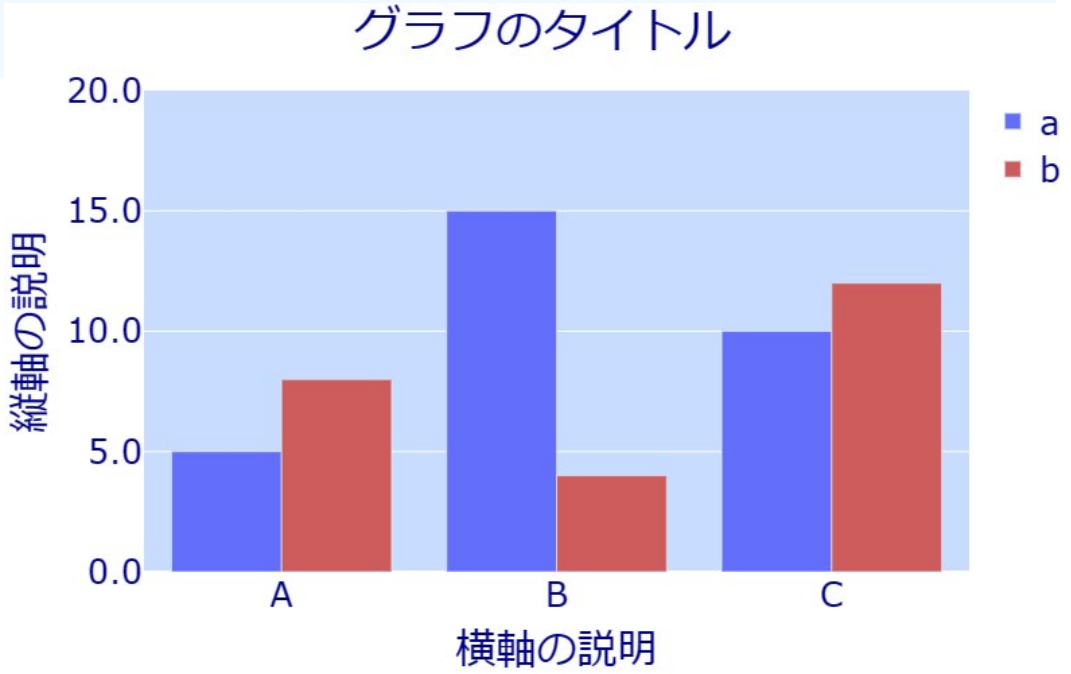
レイアウト調整（棒グラフ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Bar(x=['A','B','C'],y=[5,15,10],width=0.7))
fig.update_layout(font_size=25,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明',range=(0,20),dtick=5,tickformat=',.1f'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10),
                  plot_bgcolor='rgb(165,200,255)')
fig.show()
```



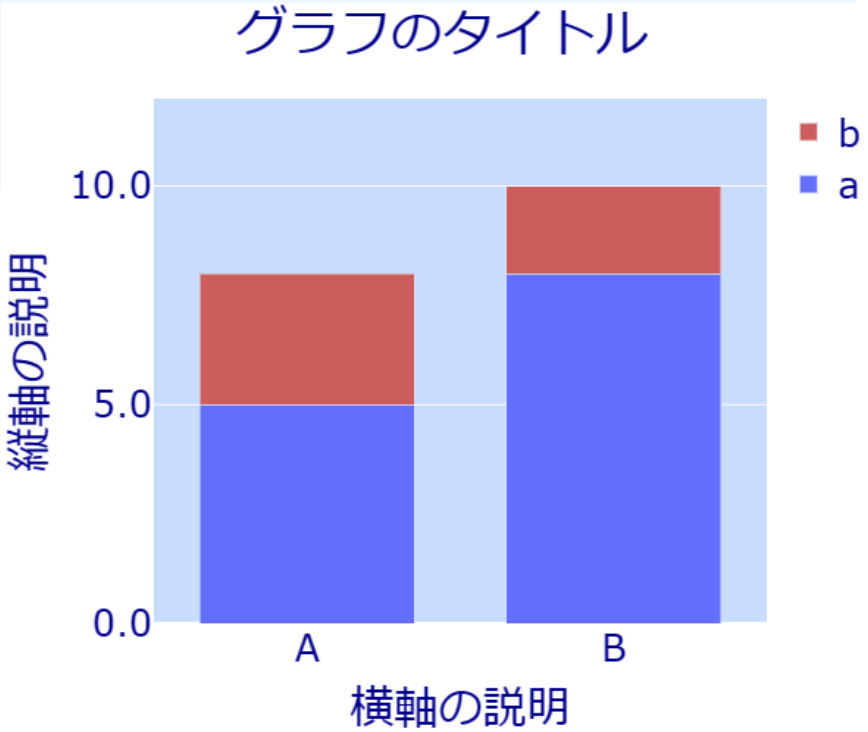
レイアウト調整（集合棒グラフ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Bar(x=['A','B','C'],y=[5,15,10],width=0.4,name='a'))
fig.add_trace(go.Bar(x=['A','B','C'],y=[8,4,12],width=0.4,name='b',
                     marker_color='indianred'))
fig.update_layout(font_size=25,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明',range=(0,20),dtick=5,tickformat=',.1f'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



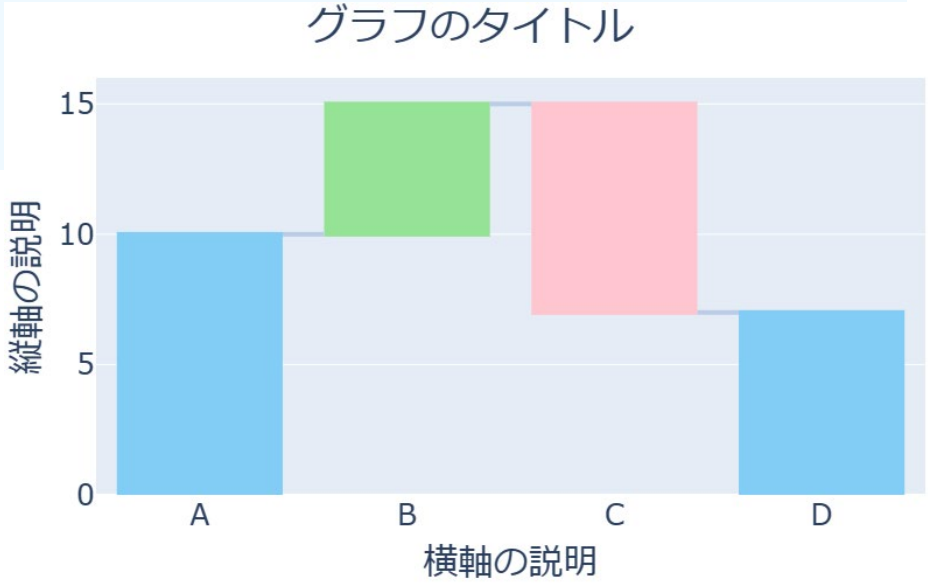
レイアウト調整（積み上げ棒グラフ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Bar(x=['A','B'],y=[5,8],width=0.7,name='a'))
fig.add_trace(go.Bar(x=['A','B'],y=[3,2],width=0.7,name='b',marker_color='indianred'))
fig.update_layout(barmode='stack',
                  font_size=25,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明',range=(0,12),dtick=5,tickformat=',.1f'),
                  width=600,height=500,
                  margin=dict(t=65,l=10,r=10,b=10),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



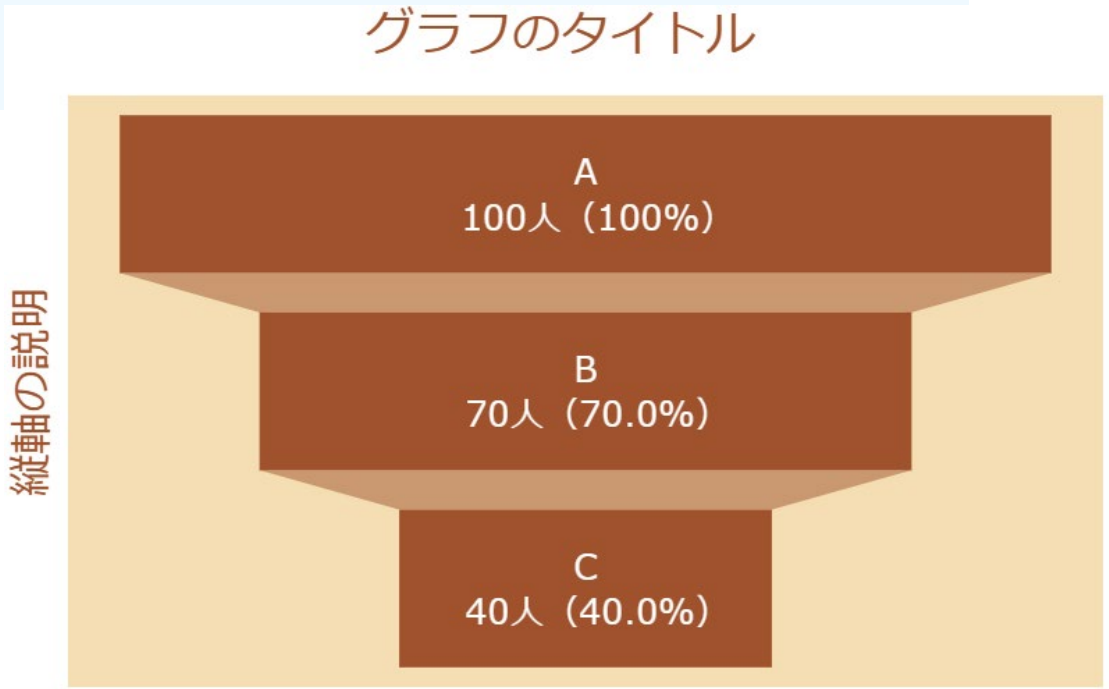
レイアウト調整（滝グラフ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Waterfall(x=['A','B','C','D'],y=[10,5,-8,7],
                           measure=['a','r','r','t'],
                           increasing=dict(marker=dict(color='#95E395')),
                           decreasing=dict(marker=dict(color='#FFC5CF')),
                           totals = dict(marker=dict(color='#82CDF6')),
                           connector = dict(line=dict(width=4, color='#BBCDE7'))))
fig.update_layout(font_size=25,
                  title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明',range=(0,16),dtick=5),
                  width=800,height=500,
                  margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```



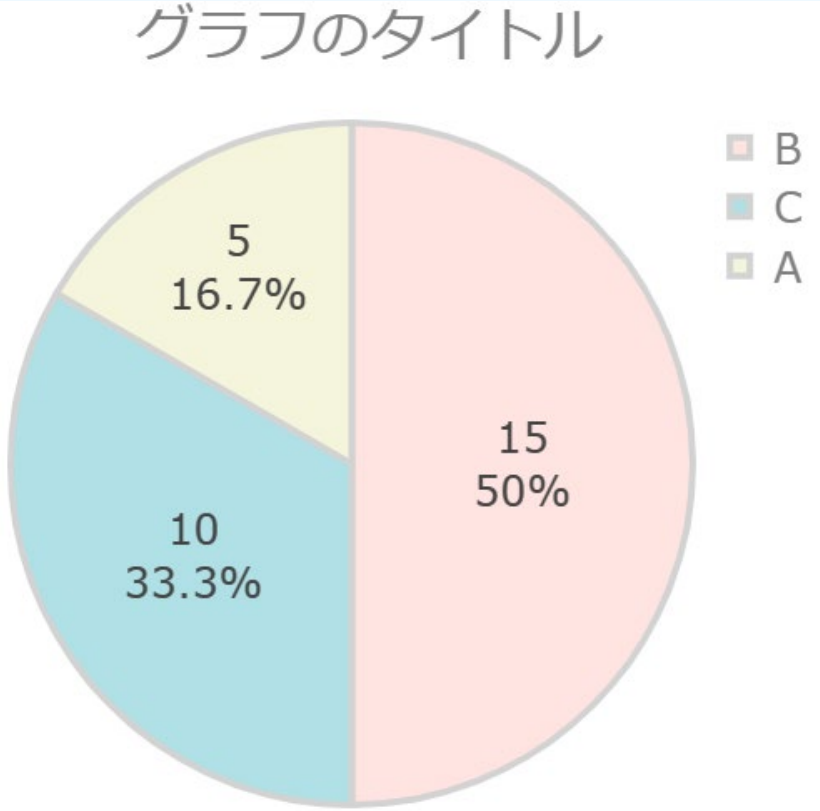
レイアウト調整（ファンネルチャート）

```
import plotly.graph_objects as go
fig=go.Figure(go.Funnel(x=[100,70,40],y=['A','B','C'],
                        marker=dict(color='sienna'),
                        texttemplate=
                        '%{label}<br>  %{value:,.0f}人(%{percentInitial:,.3p})'))
fig.update_layout(font_size=25,font_color='sienna',
                  title=dict(text='グラフのタイトル',x=0.5),
                  yaxis=dict(title='縦軸の説明',showticklabels=False),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10),
                  plot_bgcolor='wheat')
fig.show()
```



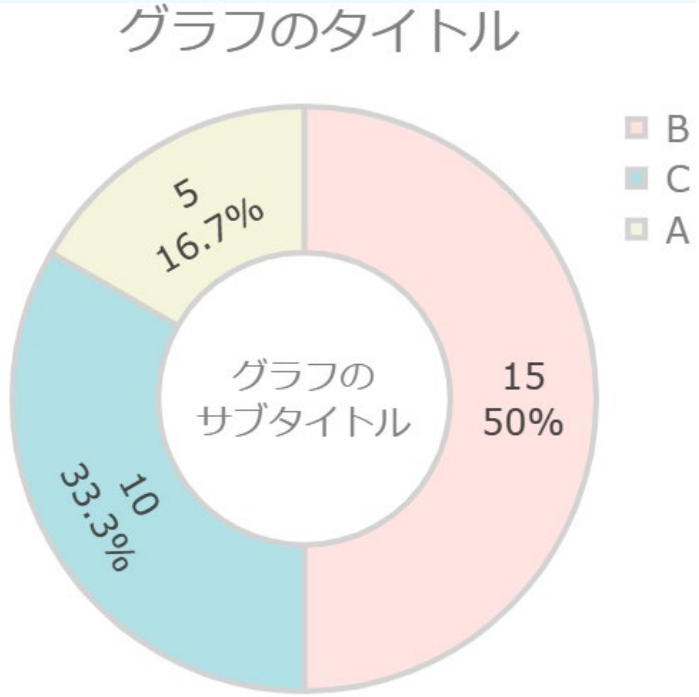
レイアウト調整（円グラフ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Pie(labels=['A','B','C'],values=[5,15,10],
                     direction='clockwise',
                     textinfo='value+percent',
                     marker=dict(colors=['beige','mistyrose','powderblue'],
                                 line=dict(color='lightgrey',width=4))))
fig.update_layout(font_size=25,font_color='grey',
                  title=dict(text='グラフのタイトル',x=0.45),
                  width=500,height=500,
                  margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```



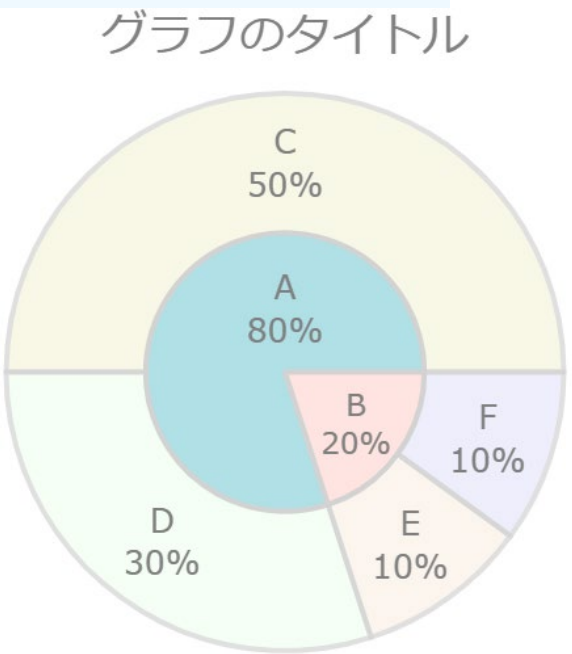
レイアウト調整（ドーナツグラフ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Pie(labels=['A','B','C'],values=[5,15,10],hole=0.5,
                     direction='clockwise',
                     textinfo='value+percent',
                     marker=dict(colors=['beige','mistyrose','powderblue'],
                                 line=dict(color='lightgrey',width=4))))
fig.update_layout(font_size=25,font_color='grey',
                  title=dict(text='グラフのタイトル',x=0.45),
                  annotations=[dict(text='グラフの<BR>サブタイトル',x=0.5,y=0.5,
                                    showarrow=False)],
                  width=500,height=500,
                  margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```



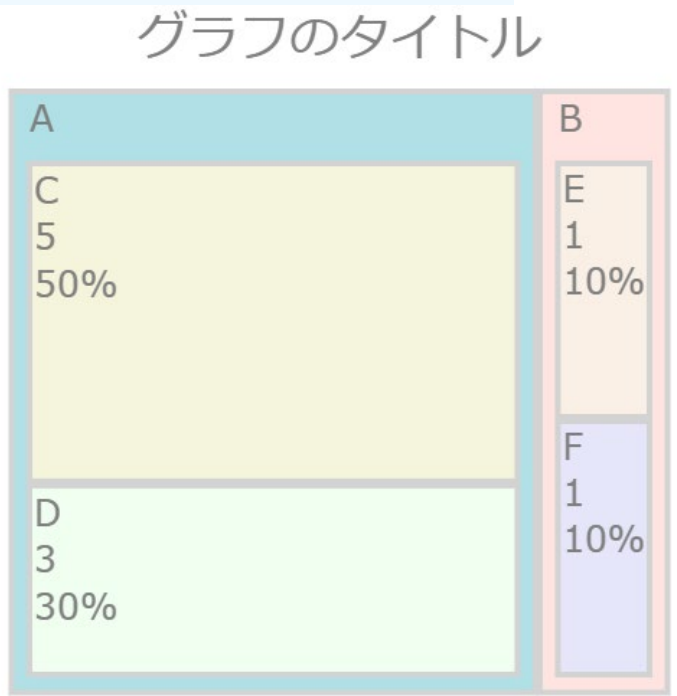
レイアウト調整（サンバーストグラフ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Sunburst(labels=['A','B','C','D','E','F'],
                          parents=['','','A','A','B','B'],
                          values=[8,2,5,3,1,1],branchvalues='total',
                          insidetextorientation='horizontal',
                          insidetextfont=dict(color='grey'),
                          textinfo='label+percent root',
                          marker=dict(colors=['powderblue','mistyrose','beige',
                                              'honeydew','linen','lavender'],
                                      line=dict(color='lightgrey',width=4))))
fig.update_layout(font_size=25,font_color='grey',
                  title=dict(text='グラフのタイトル',x=0.5),
                  width=500,height=500,
                  margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

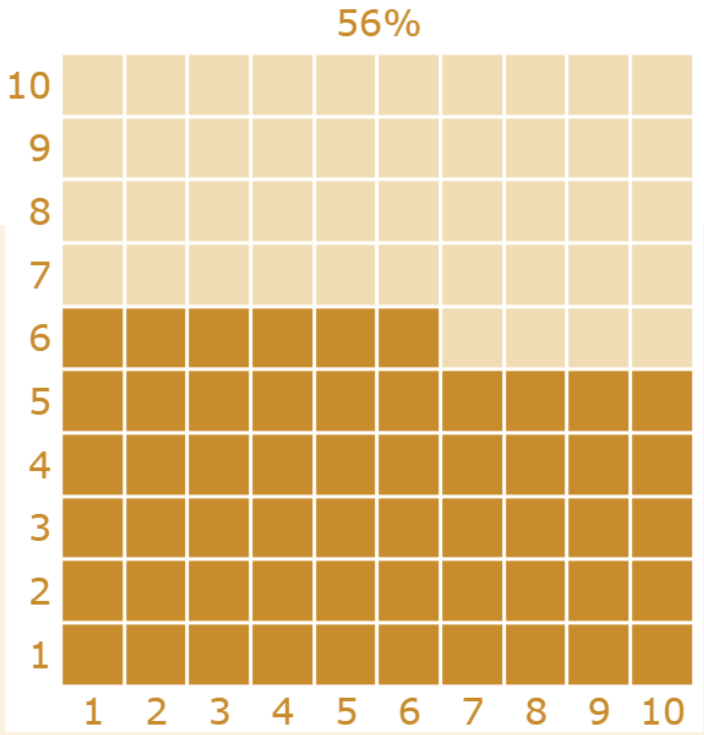


レイアウト調整（ツリーマップ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Treemap(labels=['A','B','C','D','E','F'],
                         parents=['','','A','A','B','B'],
                         values=[8,2,5,3,1,1],branchvalues='total',
                         insidetextfont=dict(color='grey'),
                         textinfo='label+value+percent root',
                         marker=dict(colors=['powderblue','mistyrose','beige',
                                             'honeydew','linen','lavender'],
                                     line=dict(color='lightgrey',width=4))))
fig.update_layout(font_size=25,font_color='grey',
                  title=dict(text='グラフのタイトル',x=0.5,y=0.98),
                  width=500,height=500,
                  margin=dict(t=10,l=10,r=10,b=10))
fig.show()
```



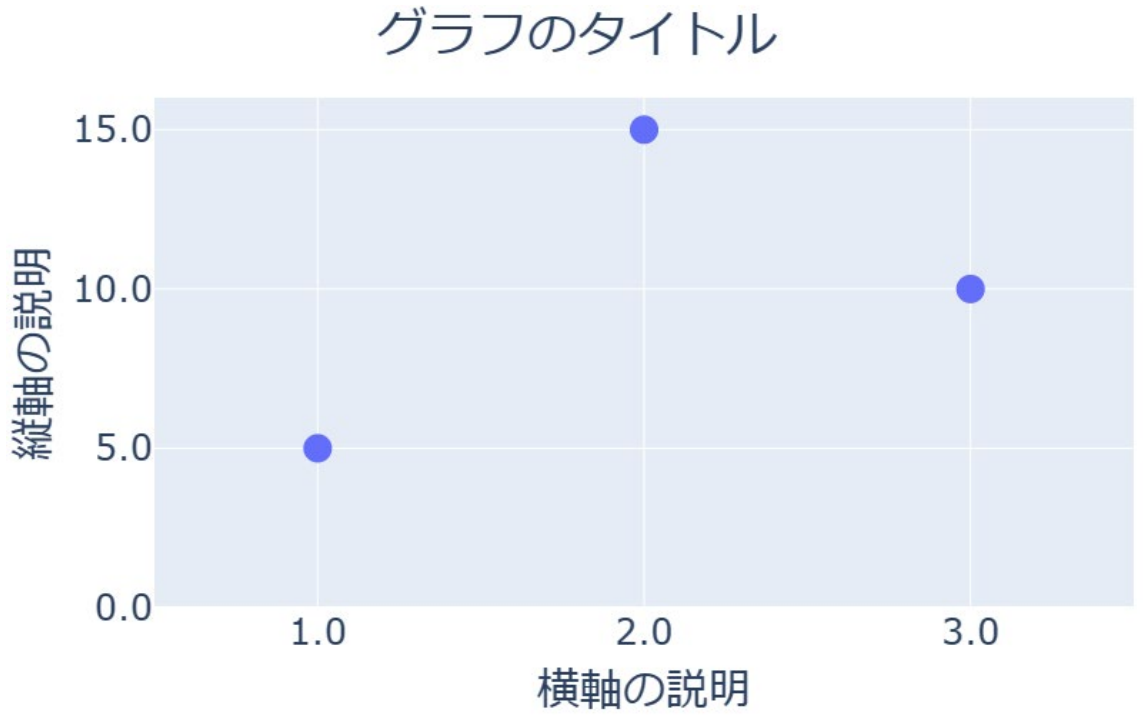
レイアウト調整（ワッフルチャート）




```
import plotly.graph_objects as go
import numpy as np
p=56
w=np.zeros(100)
w[:p]=1
n=[str(i) for i in range(1,11)]
c=['rgb(240,220,180)','rgb(200,140,45)']
if p>99: c[0]='rgb(200,140,45)'
fig=go.Figure(go.Heatmap(x=n,y=n,z=w.reshape(10,10),
                         colorscale=[[0,c[0]],[0.5,c[0]],[1,c[1]]],
                         xgap=3,ygap=3,showscale=False))
fig.update_layout(font_size=30,font_color=c[1],
                  width=570, height=600,margin=dict(t=0,l=0,r=0,b=0),
                  plot_bgcolor="white",
                  yaxis=dict(scaleanchor='x'),
                  annotations=[dict(text=str(p)+'%',x=4.5,y=10.0,showarrow=False)])
fig.show()
```

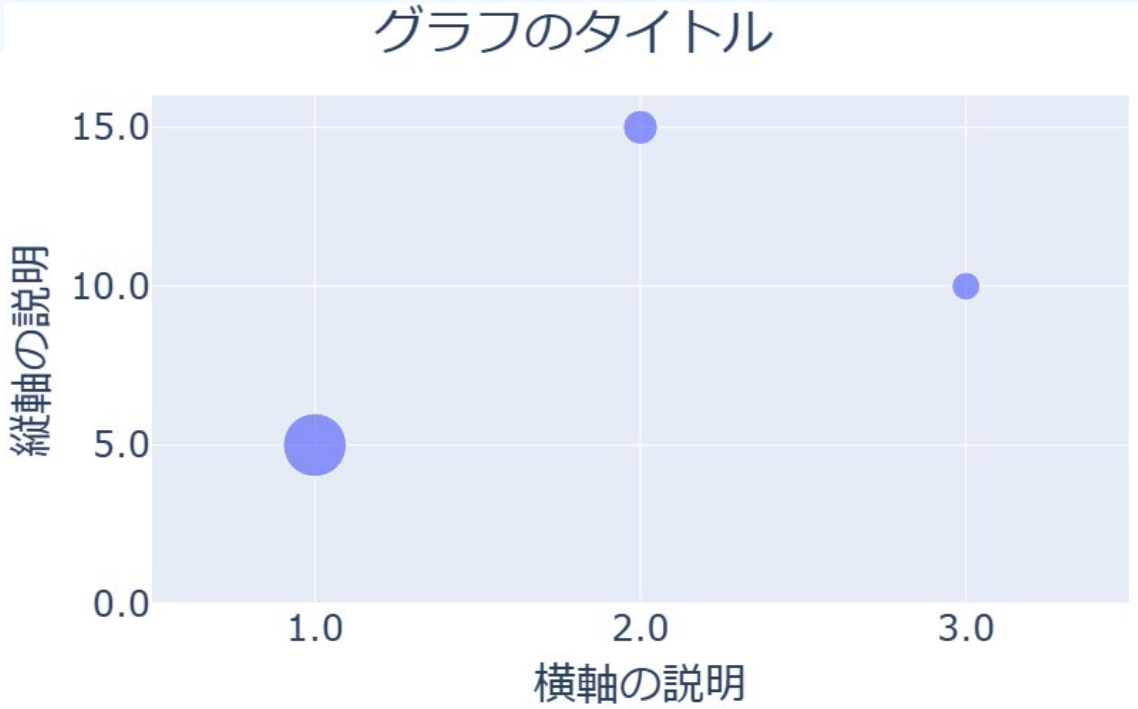
レイアウト調整（散布図）

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],mode='markers',marker_size=20))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明',
                             range=(0.5,3.5),dtick=1,tickformat=',.1f'),
                  yaxis=dict(title='縦軸の説明',
                             range=(0,16),dtick=5,tickformat=',.1f'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```



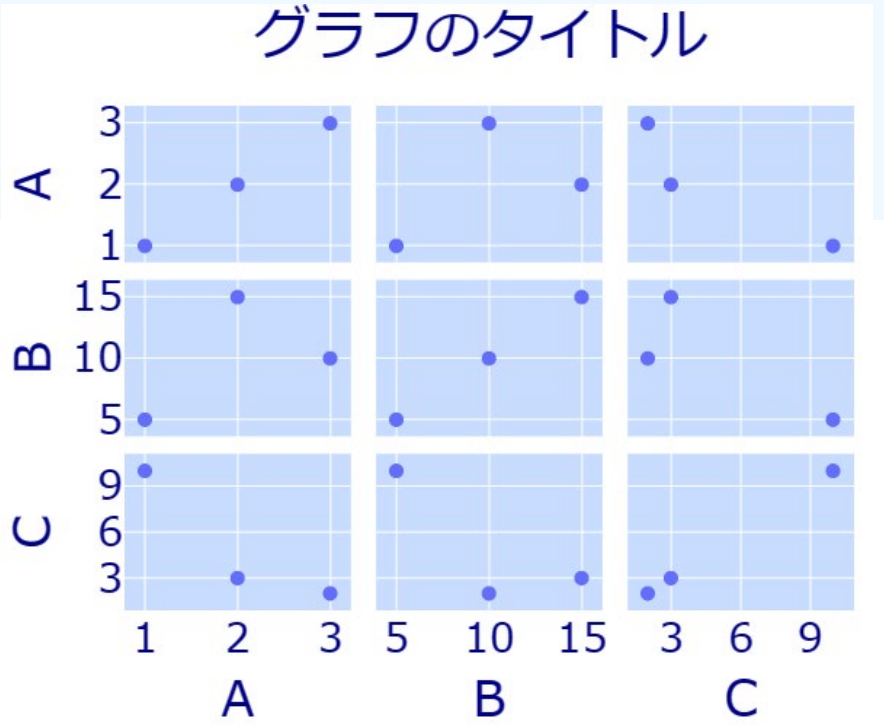
レイアウト調整（バブルチャート）

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10],mode='markers',
                         marker=dict(size=[100,30,20],sizemode='area',sizeref=0.1)))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明',
                             range=(0.5,3.5),dtick=1,tickformat=',.1f'),
                  yaxis=dict(title='縦軸の説明',
                             range=(0,16),dtick=5,tickformat=',.1f'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```



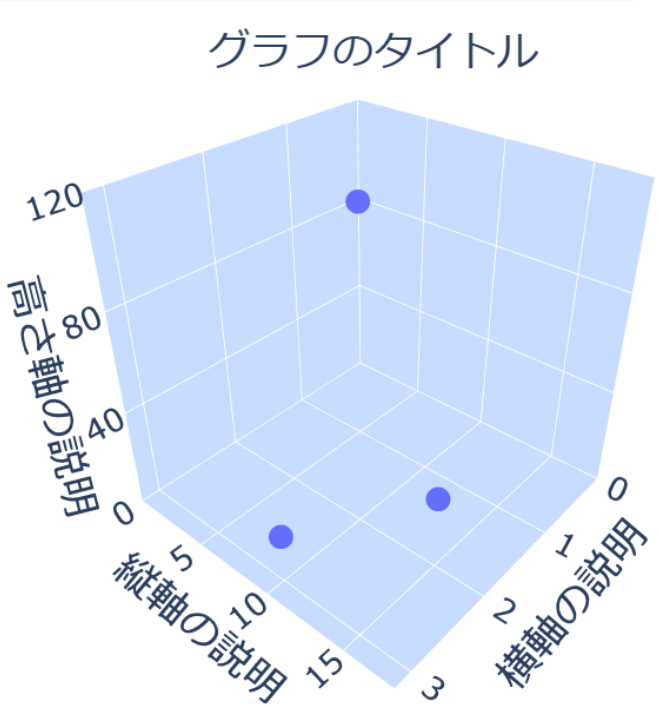
レイアウト調整（散布図行列）

```
import plotly.graph_objects as go
data=[dict(label='A',values=[1,2,3]),dict(label='B',values=[5,15,10]),
      dict(label='C',values=[10,3,2])]
fig=go.Figure(go.Splom(dimensions=data,marker=dict(size=10)))
fig.update_layout(xaxis1=dict(dtick=1),xaxis2=dict(dtick=5),xaxis3=dict(dtick=3),
                  yaxis1=dict(dtick=1),yaxis2=dict(dtick=5),yaxis3=dict(dtick=3),
                  font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.55),
                  width=600,height=500,
                  margin=dict(t=70,l=10,r=10,b=10),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



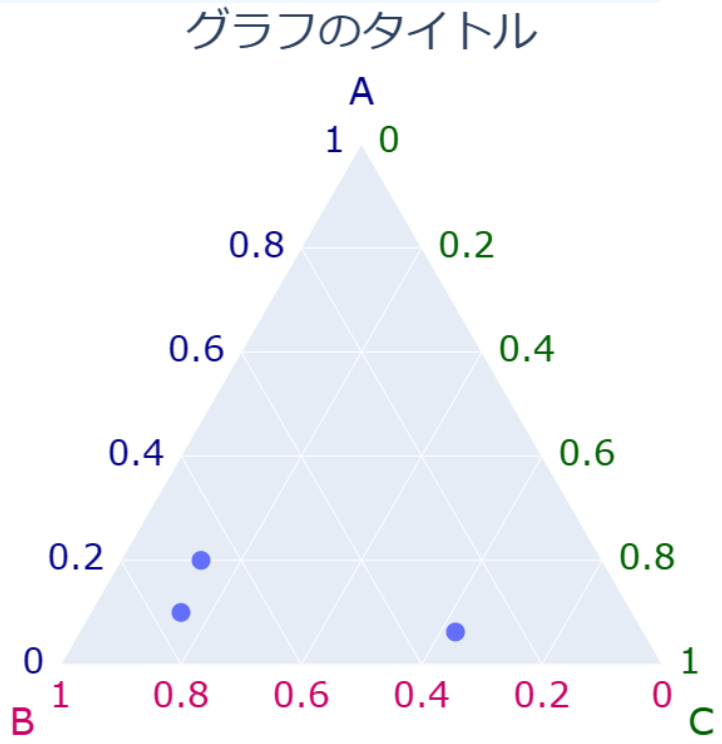
レイアウト調整（3次元散布図）

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter3d(x=[1,2,3],y=[5,15,10],z=[100,30,20],mode='markers'))
fig.update_layout(font_size=18,title=dict(text='グラフのタイトル',
                                          font=dict(size=35),x=0.5,y=0.95),
  scene=dict(xaxis=dict(title='横軸の説明',titlefont=dict(size=35),
                  range=[0,3.2],tickvals=[0,1,2,3],
                  backgroundcolor='rgb(200,220,255)'),
      yaxis=dict(title='縦軸の説明',titlefont=dict(size=35),
                  range=[0,18],tickvals=[0,5,10,15],
                  backgroundcolor='rgb(200,220,255)'),
      zaxis=dict(title='高さ軸の説明',titlefont=dict(size=35),
                  range=[0,120],tickvals=[0,40,80,120],
                  backgroundcolor='rgb(200,220,255)'),
      camera=dict(eye=dict(x=1.5, y=1.3, z=1.3))),
  width=650,height=600,margin=dict(t=0,l=0,r=0,b=0))
fig.show()
```



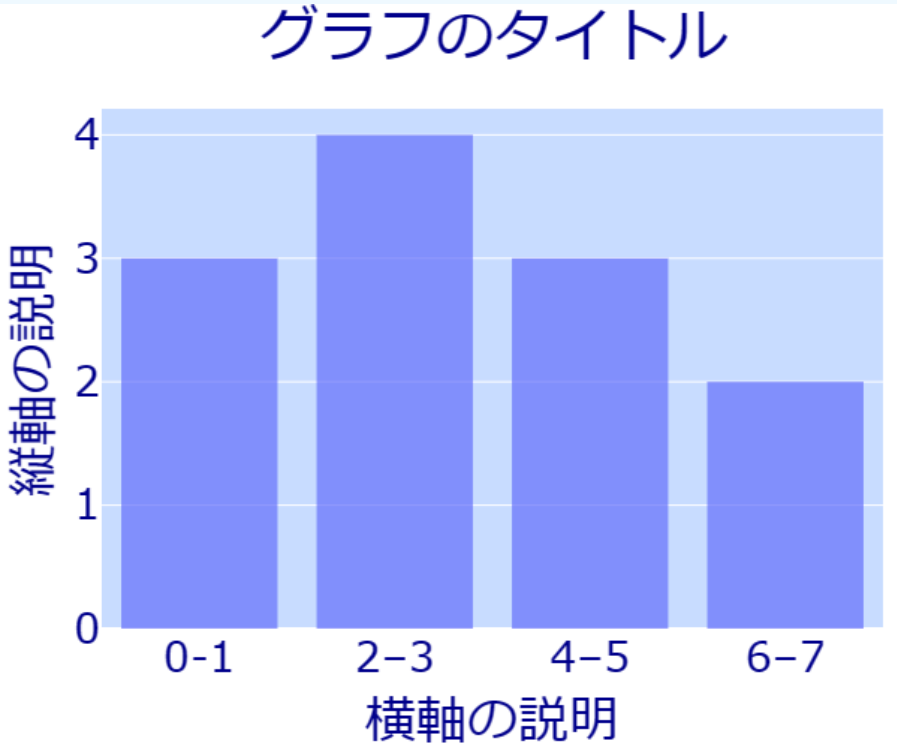
レイアウト調整（三角図）

```
import plotly.graph_objects as go
tv=[0,0.2, 0.4, 0.6,0.8,1]
tt=[' 0 ',' 0.2 ', ' 0.4 ',' 0.6 ',' 0.8 ',' 1 ']
c=['darkblue','#CC0066','darkgreen']
fig=go.Figure(go.Scatterternary(a=[1,2,3],b=[5,15,10],c=[10,3,2],
                                mode='markers',marker_size=15))
fig.update_layout(font_size=28,title=dict(text='グラフのタイトル',x=0.5),
  width=580,height=600,margin=dict(t=70,l=50,r=50,b=30),
  ternary=dict(aaxis=dict(tickfont=dict(color=c[0]),
                  title=dict(text='A',font=dict(color=c[0])),
                  tickvals=tv,ticktext=tt),
                baxis=dict(tickfont=dict(color=c[1]),
                  title=dict(text='B',font=dict(color=c[1])),
                  tickvals=tv,ticktext=tt),
                caxis=dict(tickfont=dict(color=c[2]),
                  title=dict(text='C',font=dict(color=c[2])),
                  tickvals=tv,ticktext=tt)))
fig.show()
```



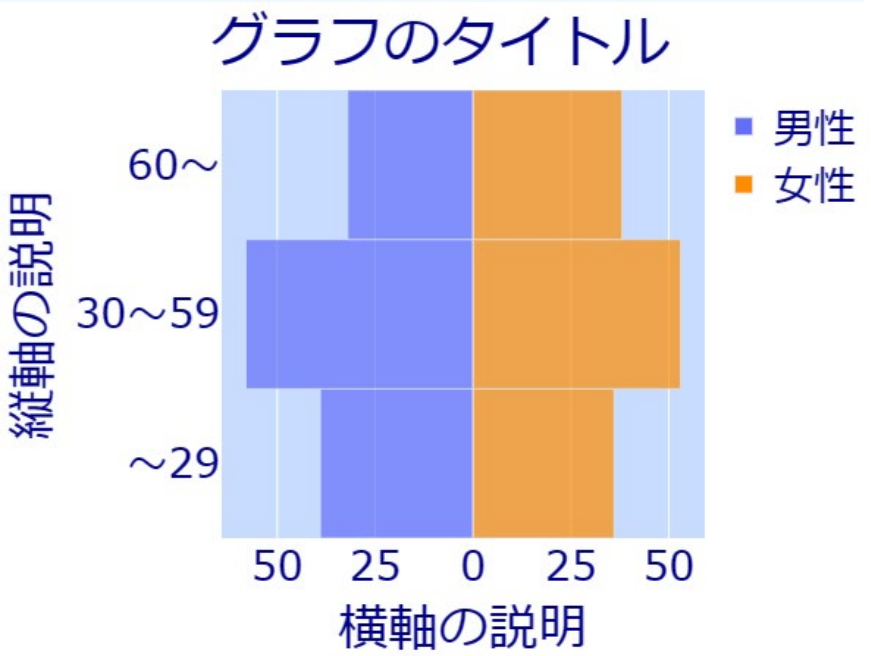
レイアウト調整（ヒストグラム）

```
import plotly.graph_objects as go
fig=go.Figure(go.Histogram(x=[1,4,3,7,4,3,1,0,3,2,5,6],
                           xbins=dict(start=0,end=7,size=2),marker=dict(opacity=0.7)))
fig.update_layout(font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.55),
                  xaxis = dict(title='横軸の説明',
                               tickvals=[1,3,5,7],
                               ticktext=['0-1','2-3','4-5','6-7']),
                  yaxis=dict(title='縦軸の説明',dtick=1),
                  bargap=0.2,
                  width=600,height=500,
                  margin=dict(t=70,l=10,r=10,b=10),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



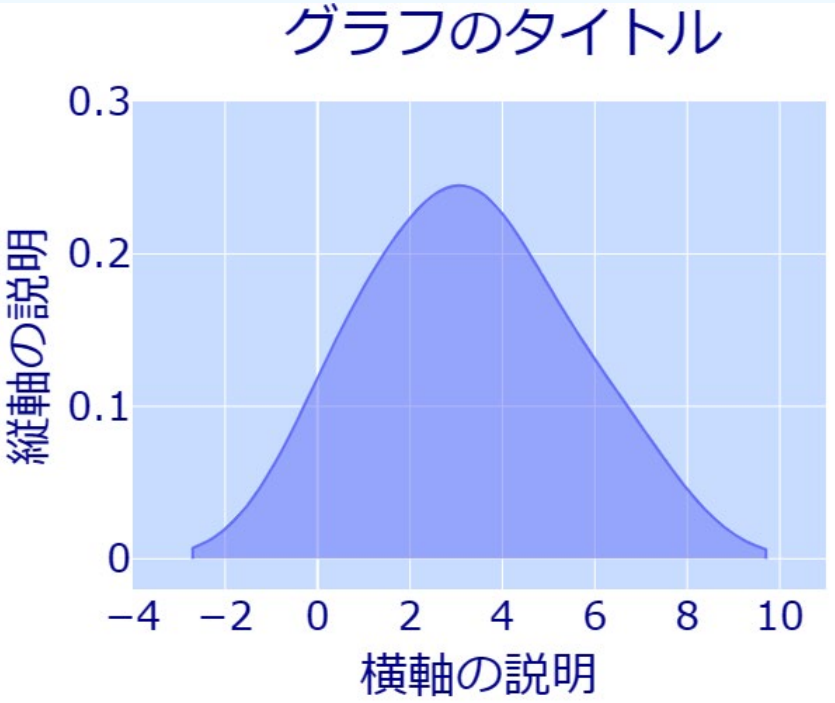
レイアウト調整（人口ピラミッド）

```
import plotly.graph_objects as go
fig=go.Figure([go.Bar(name='男性',y=['～29','30～59','60～'],
                      x=[-39,-58,-32],orientation='h',marker=dict(opacity=0.7)),
               go.Bar(name='女性',y=['～29','30～59','60～'],
                      x=[36,53,38],orientation='h',
                      marker=dict(color='darkorange',opacity=0.7))])
fig.update_layout(font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.5,y=0.97),
                  barmode='relative',bargap=0.0,
                  xaxis=dict(title='横軸の説明',
                             tickvals=[-50,-25,0,25,50],
                             ticktext=[50,25,0,25,50]),
                  yaxis=dict(title='縦軸の説明'),
                  width=600,height=450,
                  margin=dict(t=60,l=45,r=15,b=40),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



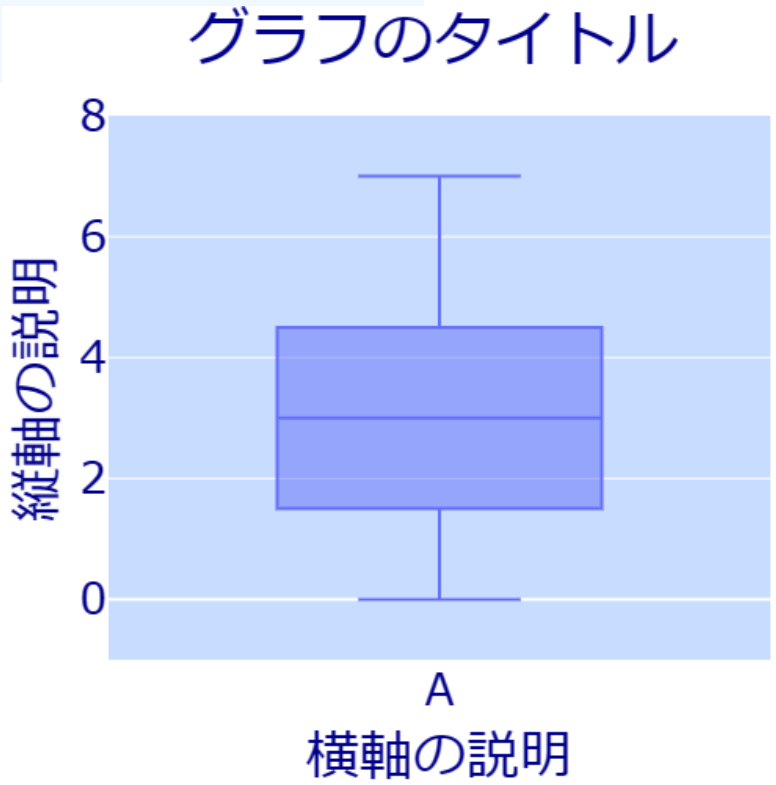
レイアウト調整（密度プロット）

```
import plotly.graph_objects as go
fig=go.Figure(go.Violin(x=[1,4,3,7,4,3,1,0,3,2,5,6],name='A',
                        orientation='h',side='positive'))
fig.update_layout(font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.6),
                  xaxis=dict(title='横軸の説明',dtick=2,range=(-4,11)),
                  yaxis=dict(title='縦軸の説明',range=(-0.02,0.3),showgrid=True,
                             tickvals=[0,0.1,0.2,0.3],ticktext=['0','0.1','0.2','0.3']),
                  width=600,height=500,margin=dict(t=70,l=10,r=10,b=10),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



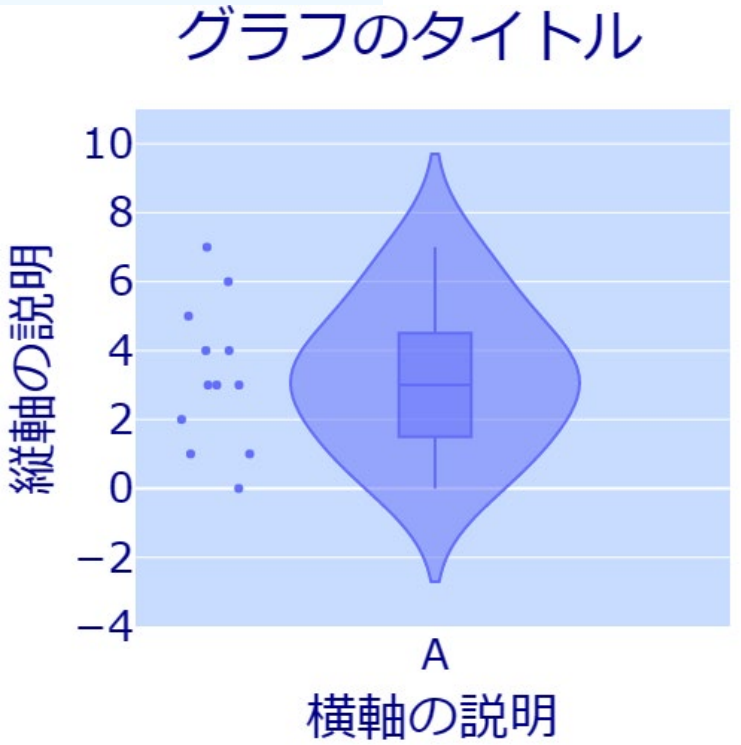
レイアウト調整（箱ひげ図）

```
import plotly.graph_objects as go
fig=go.Figure(go.Box(y=[1,4,3,7,4,3,1,0,3,2,5,6],name='A'))
fig.update_layout(font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.55),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明',dtick=2,range=(-1,8)),
                  width=500,height=500,margin=dict(t=70,l=10,r=10,b=10),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



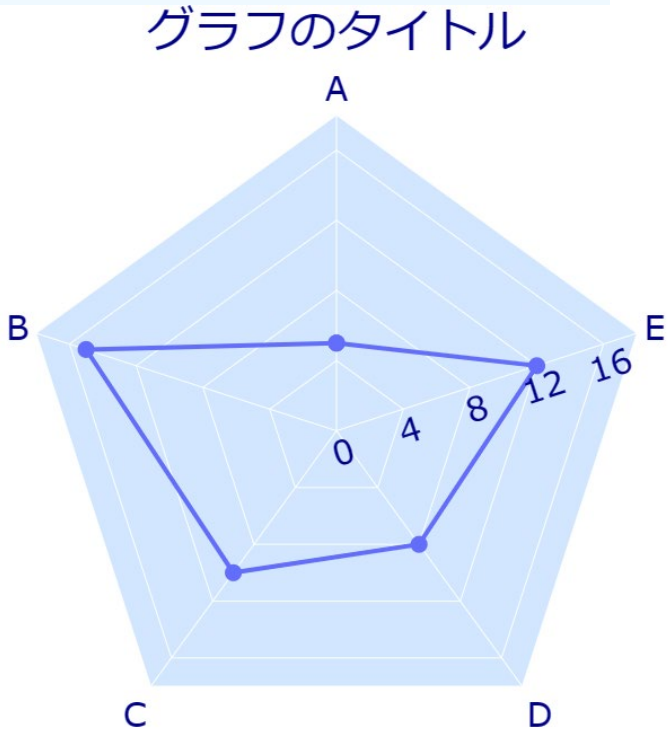
レイアウト調整（バイオリンプロット）

```
import plotly.graph_objects as go
fig=go.Figure(go.Violin(y=[1,4,3,7,4,3,1,0,3,2,5,6],name='A',
                        box_visible=True,points='all'))
fig.update_layout(font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.55),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明',dtick=2,range=(-4,11)),
                  width=500,height=500,margin=dict(t=70,l=10,r=10,b=10),
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



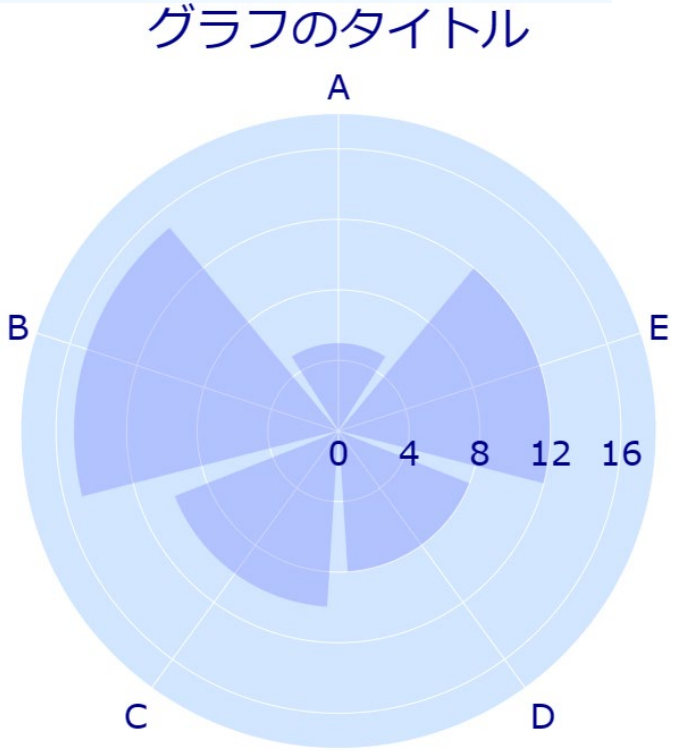
レイアウト調整（レーダーチャート）

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatterpolar(theta=['A','B','C','D','E','A'],r=[5,15,10,8,12,5],
                              line=dict(width=4),marker=dict(size=15)))
fig.update_layout(font_size=30,font_color='darkblue',
      polar=dict(radialaxis=dict(visible=True,range=[0,18],showline=False,
                                 tickvals=[0,4,8,12,16],
                                 tickfont=dict(size=30,color='darkblue')),
                 angularaxis=dict(tickfont = dict(size = 30,color='darkblue')),
                 bgcolor='rgb(210,230,255)',
                 angularaxis_rotation=90,
                 gridshape='linear'),
      title=dict(text='グラフのタイトル',x=0.5),
      width=600,height=680,margin=dict(t=75,l=20,r=20,b=0))
fig.show()
```



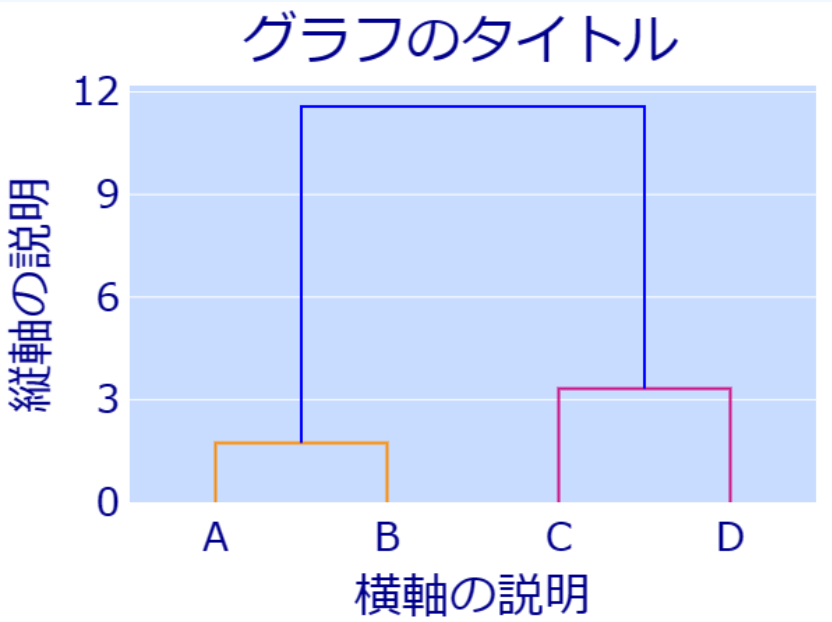
レイアウト調整（ポーラーチャート）

```
import plotly.graph_objects as go
fig=go.Figure(go.Barpolar(theta=['A','B','C','D','E'],r=[5,15,10,8,12],
                          marker=dict(opacity=0.3)))
fig.update_layout(font_size=30,font_color='darkblue',
      polar=dict(radialaxis=dict(visible=True,range=[0,18],showline=False,
                                 tickvals=[0,4,8,12,16],
                                 tickfont=dict(size=30,color='darkblue')),
                 angularaxis=dict(tickfont = dict(size = 30,color='darkblue')),
                 bgcolor='rgb(210,230,255)',
                 angularaxis_rotation=90),
      title=dict(text='グラフのタイトル',x=0.5),
      width=600,height=680,margin=dict(t=75,l=20,r=20,b=0))
fig.show()
```



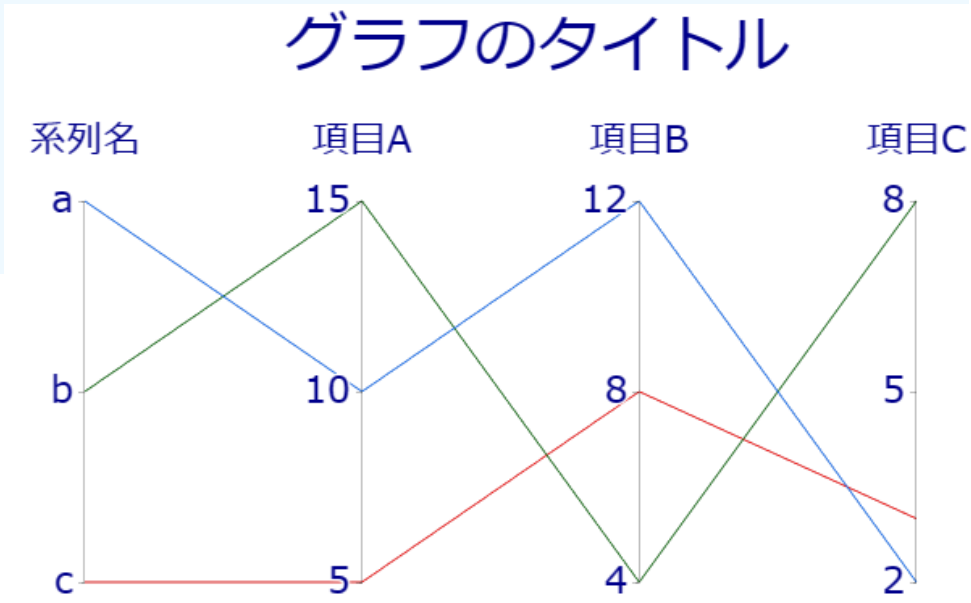
レイアウト調整（デンドログラム）

```
import plotly.figure_factory as ff
import numpy as np
d=[[1,2,3],[2,3,2],[9,8,6],[8,9,9]]
c=['blue','red','darkorange','darkturquoise''green',
  'darkgoldenrod','mediumvioletred','darkslateblue']
fig=ff.create_dendrogram(np.array(d),labels=['A','B','C','D'],colorscale=c)
fig.update_layout(font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.55,y=0.97),
                  margin=dict(t=60,l=45,r=15,b=40),
                  yaxis=dict(title='縦軸の説明',ticks='inside',ticklen=0,dtick=3,
                             showgrid=True),
                  xaxis=dict(title='横軸の説明',ticks='',
                             ticklen=0,range=(0,40)),
                  barmode='relative',bargap=0.0,
                  width=600,height=450,
                  plot_bgcolor='rgb(200,220,255)')
fig.show()
```



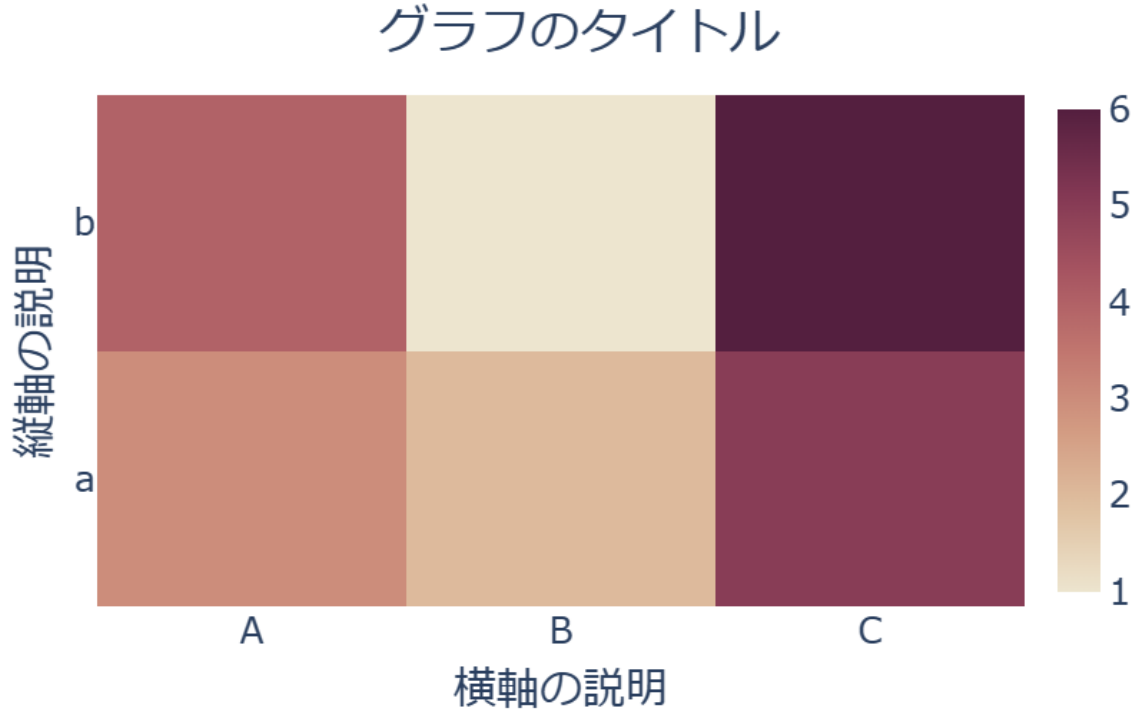
レイアウト調整（平行座標プロット）

```
import plotly.graph_objects as go
data=[dict(range=[1,3],tickvals=[3,2,1],ticktext=['a','b','c'],
           label='系列名',values=[1,2,3]),
      dict(range=[5,15],tickvals=[5,10,15],label='項目A',values=[5,15,10]),
      dict(range=[4,12],tickvals=[4,8,12],label='項目B',values=[8,4,12]),
      dict(range=[2,8],tickvals=[2,5,8],label='項目C',values=[3,8,2])]
fig=go.Figure(go.Parcoords(dimensions=data,line=dict(color=[1,2,3],
              colorscale=[[0,'red'],[0.5,'darkgreen'],[1,'#0066FF']])))
fig.update_layout(font_size=28,font_color='darkblue',
                  title=dict(text='グラフのタイトル',x=0.55,y=0.97),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明'),
                  width=600,height=400,
                  margin=dict(t=120,l=50,r=35,b=40))
fig.show()
```



レイアウト調整（ヒートマップ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Heatmap(x=['A','B','C'],y=['a','b'],z=[[3,2,5],[4,1,6]],
                         colorscale='Brwnyl'))
fig.update_layout(font_size=25,
                  title=dict(text='グラフのタイトル',x=0.5),
                  xaxis=dict(title='横軸の説明'),
                  yaxis=dict(title='縦軸の説明'),
                  width=800,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

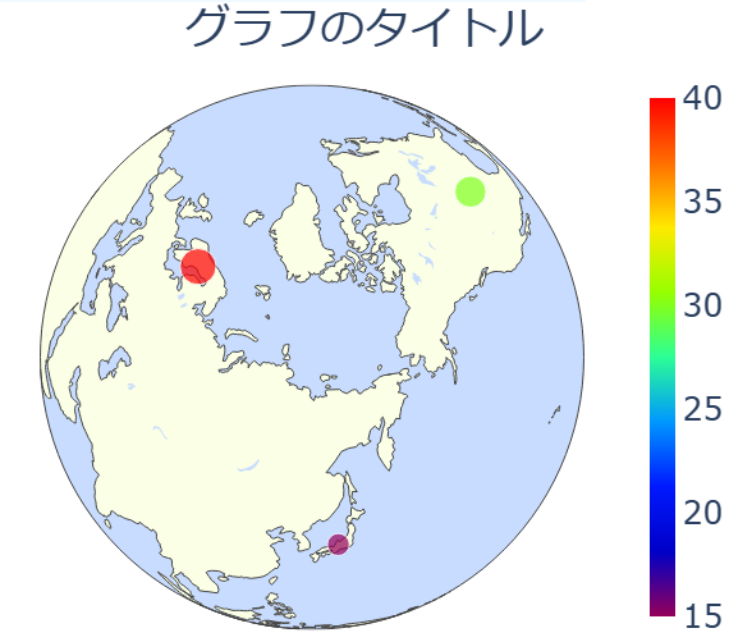


レイアウト調整（階級区分図）

```
import plotly.graph_objects as go
fig=go.Figure(go.Choropleth(locations=['JPN','SWE','USA'],z=[15,40,30],
                            colorscale = 'Pinkyl',colorbar_title = '色の説明'))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.4),
                  width=1000,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

レイアウト調整（比例シンボルマップ）

```
import plotly.graph_objects as go
fig=go.Figure(go.Scattergeo(locations=['JPN','SWE','USA'],
                            marker=dict(size=[15,40,30],color=[15,40,30],
                                        sizemode='area',sizeref=0.1,
                                        colorscale='Rainbow',
                                        colorbar=dict(thickness=20))))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.5),
                  width=570,height=500,margin=dict(t=65,l=0,r=10,b=10),
                  geo=dict(projection=dict(type='orthographic',
                                           rotation=dict(lat=80,lon=130)),
                           showland=True,
                           landcolor='rgb(250,255,230)',
                           showlakes=True,
                           lakecolor='rgb(200,220,255)',
                           showocean=True,
                           oceancolor='rgb(200,220,255)',
                           showcoastlines=True,
                           resolution=110))
fig.show()
```



データの読み込み

データをファイルから読み込もう！

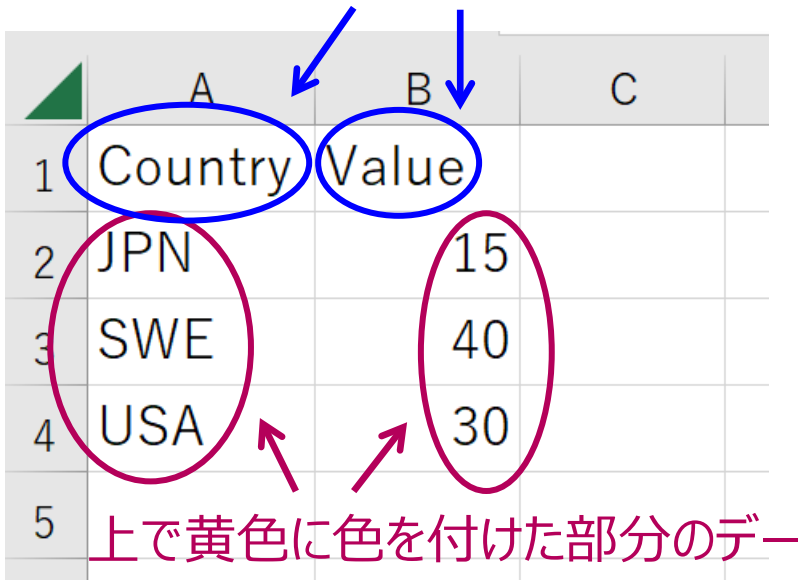
- データを毎回、Pythonのスクリプトの中に書き込むのは煩雑
- データのサイズが大きい場合、データをスクリプトの中に書き込むのは非現実的
- CSVファイルやExcelファイルなどからデータを読み込むのが効率的

読み込み手順（その1）

```
import plotly.graph_objects as go
fig=go.Figure(go.Choropleth(locations=['JPN','SWE','USA'],z=[15,40,30],
                            colorscale = 'Pinkyl',colorbar_title = '色の説明'))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.4),
                  width=1000,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

上で黄色に色を付けた部分のデータをExcelファイルから読み込むようにする

データの意味が分かるように、適当に名付ける（全角文字は避ける）



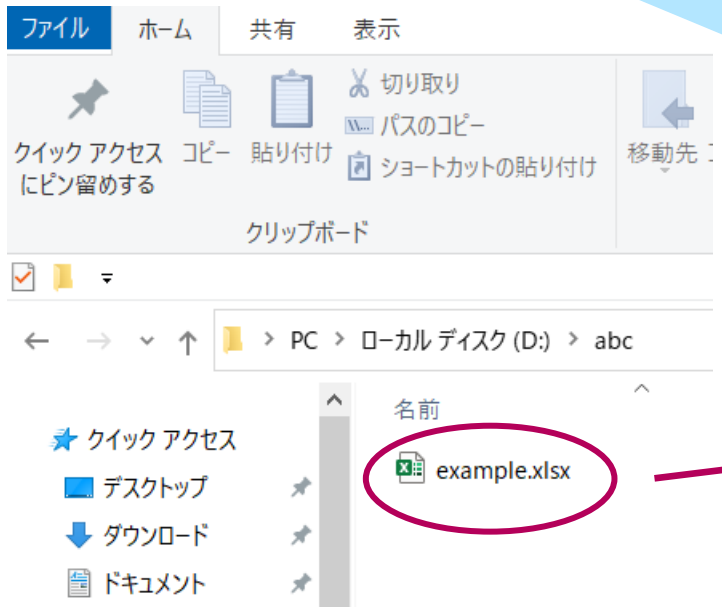
手順1 Excelでデータを入力し、適当なファイル名（日本語など全角文字は避ける）で保存。
例えば、example.xlsx と名付ける。

読み込み手順（その2）

Excelファイルからデータを読み込む手順の続き

手順2 Google Colabの左端のファイルのアイコンをクリックし、ファイルタブを開く

手順3 データを入力して保存したファイルを、Google Colab のファイルタブにドラッグ&ドロップ



マウスでドラッグ&ドロップ

読み込み手順（その3）

```
import plotly.graph_objects as go
fig=go.Figure(go.Choropleth(locations=['JPN','SWE','USA'],z=[15,40,30],
                            colorscale = 'Pinkyl',colorbar_title = '色の説明'))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.4),
                  width=1000,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

	A	B	C
1	Country	Value	
2	JPN	15	
3	SWE	40	
4	USA	30	
5			

手順4 データを直接、スクリプトに書き込んでいた部分を書き換え、ファイルから読み込むようにする

```
import plotly.graph_objects as go
import pandas as pd
df=pd.read_excel('example.xlsx')
fig=go.Figure(go.Choropleth(locations=df['Country'],z=df['Value'],
                            colorscale = 'Pinkyl',colorbar_title = '色の説明'))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.4),
                  width=1000,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

読み込み手順（補足）

```
import plotly.graph_objects as go
import pandas as pd  #データ解析用のライブラリを読み込む
df=pd.read_excel('example.xlsx') #example.xlsxからデータを読み込み、dfと名付ける
fig=go.Figure(go.Choropleth(locations=df['Country'],z=df['Value'],
#df['Country']で、df(example.xlsxから読み込んだデータ)のCountryの列をlocationsに読み込む。
#df['Vlue']で、df(example.xlsxから読み込んだデータ)のVlueの列をlocationsに読み込む。
                            colorscale='Pinkyl',colorbar_title='色の説明'))
fig.update_layout(font_size=25,title=dict(text='グラフのタイトル',x=0.4),
                  width=1000,height=500,margin=dict(t=65,l=10,r=10,b=10))
fig.show()
```

エクセルに複数のシートがある場合は、例えば下記のようにする（Sheet1という名前のシートを読み込む場合）

```
df=pd.read_excel('example.xlsx',sheet_name='Sheet1')
```

エクセルファイル（～.xlsx）ではなく、csvファイルからデータを読み込む場合は、例えば下記のようにする

```
df=pd.read_csv('example.csv')
```

	A	B	C
1	Country	Value	
2	JPN	15	
3	SWE	40	
4	USA	30	
5			

チャートの保存

描いたチャートをファイルに保存してみよう！

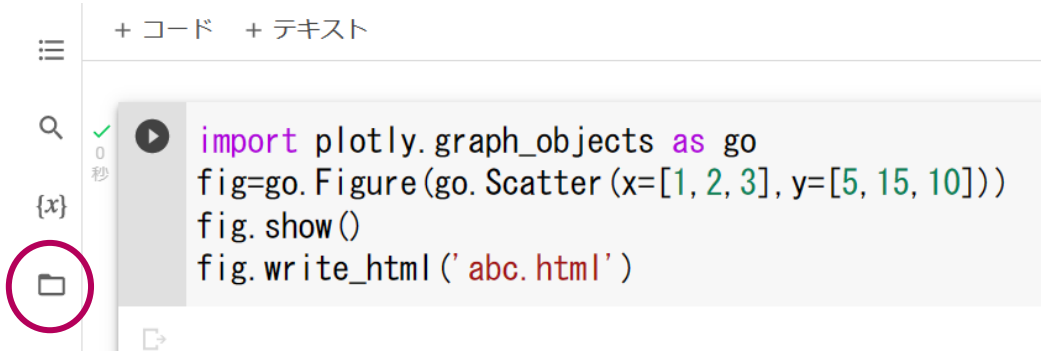
- Webサイトにおける標準形式であるhtml形式で保存すれば、Microsoft Edgeなどのブラウザで閲覧可能（マウスで画像を動かしたりできる）
- 画像ファイルやPDFファイルとして保存することも可能

チャートの保存（html形式）

描画結果をhtml形式（Webサイト用の標準的な形式）で保存

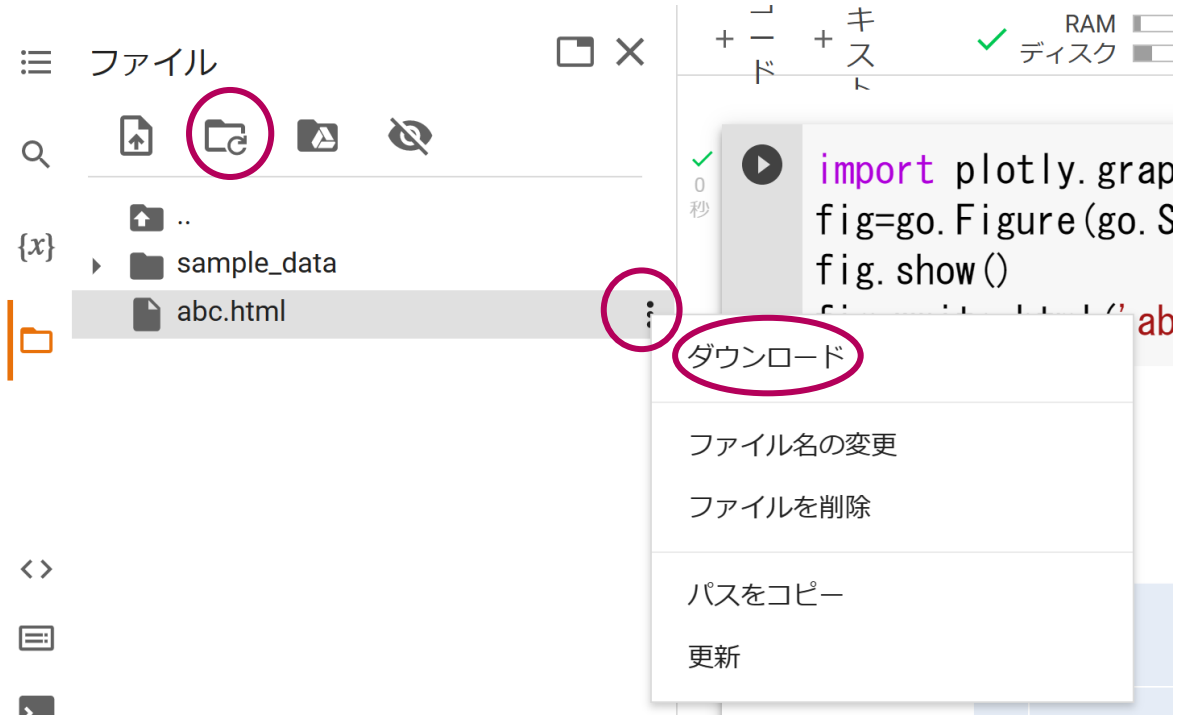
```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.show()
fig.write_html('abc.html')
```

末尾に、この一行を追加（ファイル名のabcは適宜変更）



1. ▶ をクリックして描画を実行
2. 📁 をクリックしてファイルタブを開く
3. ファイルタブにある「abc.html」というファイル名にマウスを合わせると、右側に ⋮ が現れるので、それをクリック
4. ⋮ をクリックして現れたメニューで「ダウンロード」を選択すると、画像ファイルをダウンロードできる

※ 上記の3で、ファイルが見つからない場合は、 🔄 をクリックして、情報を更新

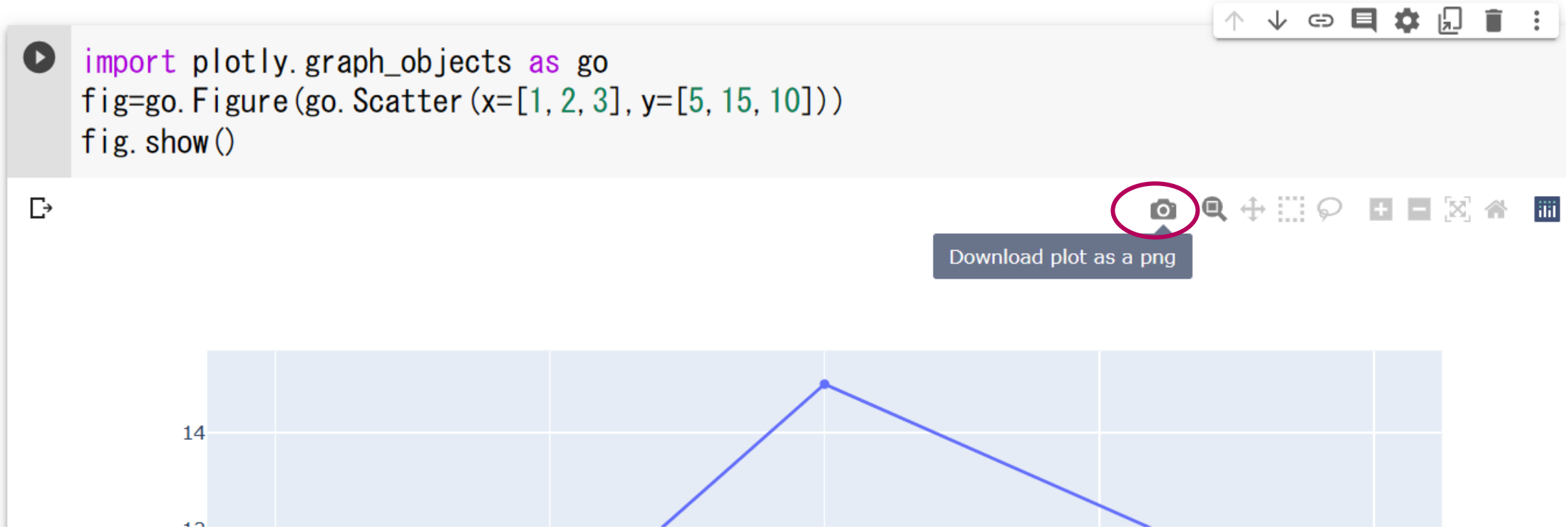


チャートの保存（画像形式 その1）

描画結果をマウス操作で画像ファイル（png形式）に保存する方法

チャートを描画後、チャート右上にマウスを持っていくと、アイコンが並んだメニューが表示される。
その中のカメラマーク  をクリックすると、チャートが画像として保存される。
保存先は、パソコンの環境によって違うが、Windowsの場合、一般的には「ダウンロード」フォルダ。

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.show()
```



チャートの保存（画像形式 その2）

描画結果を自動で画像ファイル（png, jpeg, pdf形式等）に保存する方法

※この方法は、2024年1月現在、Google Colaboratoryでは、そのままでは上手く機能しない模様（次ページで対処法を説明）

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.show()
fig.write_image('abc.png')
```

← 末尾に、この一行を追加（ファイル名のabcは適宜変更）

※ kaleidoという名前のライブラリがインストールされていない場合は、インストールが必要

※ ファイル名の拡張子を、png、jpg（またはjpeg）、webp、svg、pdfにすれば、それぞれの形式で保存可能

※ 出力画像のサイズは、下記のように指定可能

```
fig.write_image('abc.png',width=800)
```

チャートの保存（画像形式 その2）

描画結果を自動で画像ファイル（png, jpeg, pdf 形式等）に保存する方法

Google Colaboratoryでの対処法（2024年1月現在）

まず、下準備として、以下を実行

※ これは最初に1度行えば良い（チャートを描くたびに実施する必要はない）。
　ただし、時間が経つと初期化されるので、描画時にエラーが出たら、再度、実行する必要がある。

※ 下準備実行後、「ランタイム（プログラムの実行状態）の再起動」が必要。「ランタイムの再起動」は、
　Google Colaboratoryの上部メニューの「ランタイム」をクリックし、「セッションを再開する」を選択。

```
!pip install kaleido
!pip install -U plotly
```

上記の後、例えば以下を実行（あとは、html形式で保存した場合と同様にダウンロード）

```
import plotly.graph_objects as go
fig=go.Figure(go.Scatter(x=[1,2,3],y=[5,15,10]))
fig.show()
fig.write_image('abc.png')
```

データ出典URL一覧

[1] Riksbanken（スウェーデン中央銀行）, *Betalningsrapport 2022, Trender på betalningsmarknaden, Betalning i butik sker sällan med kontanter,* https://www.riksbank.se/sv/betalningar--kontanter/sa-betalar-svenskarna/betalningsrapport-2022/trender-pa-betalningsmarknaden-/betalning-i-butik-sker-sallan-med-kontanter/
[2] SCB（スウェーデン統計庁）, *Antal barn per kvinna efter födelseland 1970–2022 samt framskrivning 2023–2070,* https://www.scb.se/contentassets/998417ea486143f88e84757e5b57cf02/be0401_2023i70.xlsx
[3] SCB, *Antal personer med utländsk eller svensk bakgrund (fin indelning) efter region, ålder och kön,* https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__BE__BE0101__BE0101Q/UtlSvBakgFin/
[4] SCB, Eltillförsel i Sverige efter produktionsslag, https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__EN__EN0105__EN0105A/ElProdAr/

データ出典URL一覧

[5] 経済産業省資源エネルギー庁，2021年度電力調査統計表2-(1) 発電実績,
https://www.enecho.meti.go.jp/statistics/electric_power/ep002/results_archive.html
[6] SCB, *Folkmängd i riket, län och kommuner 31 december och befolkningsförändringar 1 januari - 31 december 2021,*
https://www.scb.se/contentassets/f3e2948f57cc433094b812995d98c1d0/be0101_tabhel2021.xlsx
[7] SJ（スウェーデン国有鉄道）, *SJs års- och hållbarhetsredovisning 2021,*
https://www.sj.se/content/dam/externt/dokument/finansiell-info/sj-ars-och-hallbarhetsredovisning-2021.pdf
[8] SCB, *Partisympatiundersökningen november 2022,*
https://www.scb.se/contentassets/ffed3272943f4989a70255bce8570f70/me0201_2022m11_br_me60br2202.pdf
[9] SCB, *Invandring 2021 efter län, kommun och födelseland*,
https://www.scb.se/hitta-statistik/statistik-efter-amne/befolkning/befolkningens-sammansattning/befolkningsstatistik

データ出典URL一覧

[10] SCB, *Invandringar och utvandringar efter födelseland och kön,* https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__BE__BE0101__BE0101J/ImmiEmiFod/
[11] Sveriges bagare & konditorer AB（スウェーデン・パン菓子職人株式会社）, *Idag äter vi närmre 6 miljoner semlor,* https://www.bageri.se/press/pressmeddelanden/2020/idag-ater-vi-narmre-6-miljoner-semlor-gif-fans-toppar-allsvenska-semmelligan-och-racketsportare-utklassar-soffliggare/
[12] SCB, *Vaccinationsgrad och valdeltagande i samtliga kommuner,* https://www.scb.se/contentassets/746bd147b5fe4c189a94081f4ae8f963/kommunvis-vaccinationstackning-v-46-2021-och-valdeltagande-kommunfullmaktige-2018.xlsx
[13] SCB, Antal personer i olika intervall av sammanräknad förvärvsinkomst efter ålder 2020, https://www.scb.se/contentassets/fa6271c19d604388adb28d26beec048f/csfvi---inkomstklasser-2020.xlsx
[14] SCB, Folkmängden efter region, civilstånd, ålder och kön, https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__BE__BE0101__BE0101A/BefolkningNy/

データ出典URL一覧

[15] SCB, Sammanräknad förvärvsinkomst för boende i Sverige den 31/12 resp år efter region, kön, ålder och inkomstklass, https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__HE__HE0110__HE0110A/SamForvInk2/
[16] SCB, *Medborgarnas syn på skola och omsorg efter region och bakgrund,* https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__ME__ME0003__ME0003A/MedborgSkolaOms/
[17] SCB, *Befolkningens medelålder efter region och kön,* https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__BE__BE0101__BE0101B/BefolkningMedelAlder/
[18] SCB, *Hyra i hyreslägenheter efter län och kommun,* https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__BO__BO0406__BO0406E/BO0406Tab01/
[19] SCB, *Flyttningsvariabler efter län och kön,* https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__AA__AA0003__AA0003C/IntGr2Lan/

データ出典URL一覧

[20] SCB, *Demografivariabler för samtliga efter län och kön,*
https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__AA__AA0003__AA0003E/IntGr3LanKONS/
[21] SCB, *Medelarbetstid (överenskommen) per vecka för sysselsatta 15-74 år (AKU) efter region och kön,*
https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__AM__AM0401__AM0401N/NAKUOkMedArbtidLAr/
[22] SCB, *Summerad fruktsamhet efter region och kön,*
https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__BE__BE0101__BE0101H/FruktsamhetSum
[23] SCB, *Antal och andel hushåll efter region, boendeform och bostadsarea (exklusive specialbostäder),*
https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__HE__HE0111__HE0111A/HushallT27/

データ出典URL一覧

[24] *SCB, Valdeltagande i riksdagsvalet 2018,* https://www.scb.se/hitta-statistik/statistik-efter-amne/demokrati/allmanna-val/allmanna-val-valdeltagandeundersokningen/pong/tabell-och-diagram/riksdagsval/valdeltagande-i-riksdagsvalet-2018/
[25] SCB, *Varuimport och varuexport. Totala värden efter handelspartner, bortfallsjusterat,* https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START__HA__HA0201__HA0201A/OImpExpLandTotAr/
[26] Sveriges riksdag（スウェーデン国会）, *Lag (2016:1013) om personnamn,* https://www.riksdagen.se/sv/dokument-och-lagar/dokument/svensk-forfattningssamling/lag-20161013-om-personnamn_sfs-2016-1013/