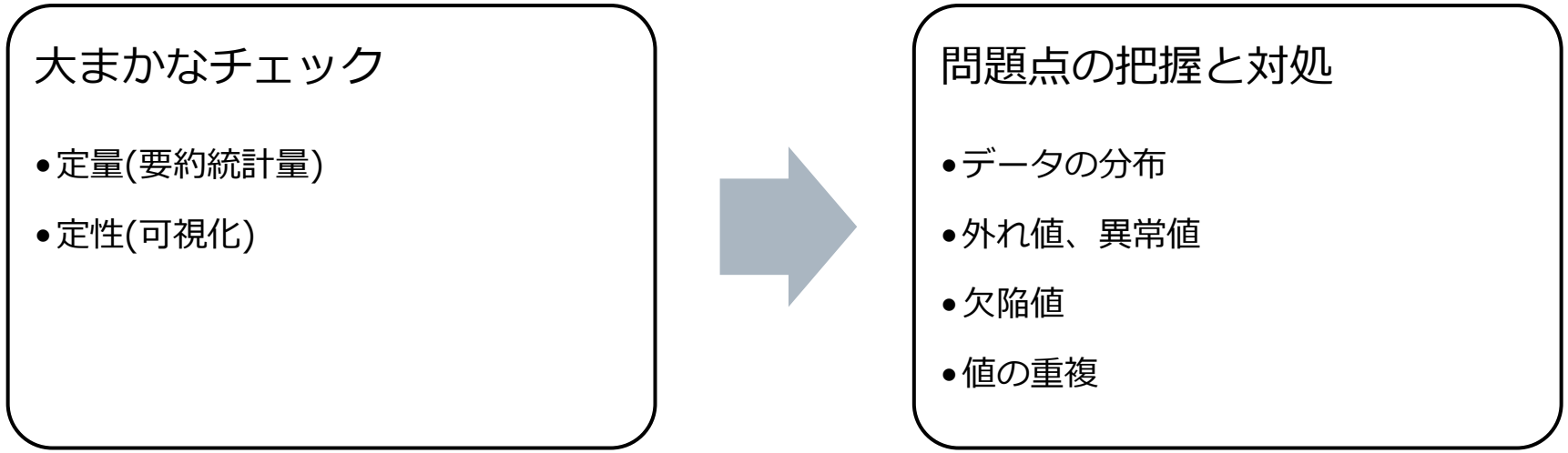
第5章 データの評価

データの評価方法を解説

データの評価

- データの加工と同時にデータの評価は重要である
- 加工するデータや分析するデータ、機械学習モデルに投入するデータの中身を把握することなく先に進むことはできない
- 表形式データの定量的および定性的なチェック方法と、外れ値や欠損値、値の重複への対処方法を解説していきます



データの評価のイメージ

使用するデータの紹介と読み込み

05-01-data.ipynb

使用するデータの紹介と読み込み

「南極のペンギンに関するデータ」の出典／データソース／ライセンス

南極半島に生息するペンギンの体格などを2007年から2009年にかけて調査したデータ

- 原出典
  - Gorman KB, Williams TD, Fraser WR (2014). Ecological Sexual Dimorphism and Environmental Variability within a Community of Antarctic Penguins（Genus Pygoscelis). PLoS ONE 9(3):e90081.
  - https://doi.org/10.1371/journal.pone.0090081
- データソース
  - Adélie penguins: Palmer Station Antarctica LTER and K. Gorman. 2020. Structural size measurements and isotopic signatures of foraging among adult male and female Adélie penguins (Pygoscelis adeliae) nesting along the Palmer Archipelago near Palmer Station, 2007-2009 ver 5. Environmental Data Initiative.
    - https://doi.org/10.6073/pasta/98b16d7d563f265cb52372c8ca99e60f
  - Gentoo penguins: Palmer Station Antarctica LTER and K. Gorman. 2020. Structural size measurements and isotopic signatures of foraging among adult male and female gentoo penguins (Pygoscelis papua) nesting along the Palmer Archipelago near Palmer Station, 2007-2009 ver 7. Environmental Data Initiative.
    - https://doi.org/10.6073/pasta/9fc8f9b5a2fa28bdca96516649b6599b
  - Chinstrap penguins: Palmer Station Antarctica LTER and K. Gorman. 2020. Structural size measurements and isotopic signatures of foraging among adult male and female Chinstrap penguins (Pygoscelis antarcticus) nesting along the Palmer Archipelago near Palmer Station, 2007-2009 ver 8. Environmental Data Initiative.
    - https://doi.org/10.6073/pasta/ce9b4713bb8c065a8fcfd7f50bf30dde
- ライセンス
  - CC-0（Palmer Station LTER Data PolicyおよびLTER Data Access Policy for Type I dataに基づく）

データ読み込み

- Adelie penguin（Pygoscelis adeliae）　　：アデリーペンギン
- Gentoo penguin（Pygoscelis papua）　　：ジェンツーペンギン
- Chinstrap penguin（Pygoscelis antarctica）　　：ヒゲペンギン

```
# https://portal.edirepository.org/nis/mapbrowse?packageid=knb-lter-pal.219.5 にあるコードを改変
import pandas as pd

uris = [
    # Adelie
    "https://pasta.lternet.edu/package/data/eml/knb-lter-pal/219/5/002f3893385f710df69eeebe893144ff",
    # Gentoo
    "https://pasta.lternet.edu/package/data/eml/knb-lter-pal/220/7/e03b43c924f226486f2f0ab6709d2381",
    # Chinstrap
    "https://pasta.lternet.edu/package/data/eml/knb-lter-pal/221/8/fe853aa8f7a59aa84cdd3197619ef462",
]
```

※ このデータが南極のパーマー基地周辺で収集されたことにちなんで、パーマーペンギン（Palmer Penguins）のデータと呼ばれることがある。ただし、そのようなペンギンの種は存在しない。

データ読み込み

```
def read_data(uri):
    return pd.read_csv(
        uri,
        sep=",",
        quotechar='"',
        usecols=[  # ・・①
            "Species",
            "Island",
            "Individual ID",
            "Date Egg",
            "Culmen Length (mm)",
            "Culmen Depth (mm)",
            "Flipper Length (mm)",
            "Body Mass (g)",
            "Sex",
            "Comments",
        ],
        na_values=".",  # 追加で欠損値として扱う文字列を指定
    ).rename(  # ・・②
        columns={
            "Individual ID": "Individual_ID",
            "Date Egg": "Date_Egg",
            "Culmen Length (mm)": "Culmen_Length",
            "Culmen Depth (mm)": "Culmen_Depth",
            "Flipper Length (mm)": "Flipper_Length",
            "Body Mass (g)": "Body_Mass",
        }
    )


df = pd.concat([read_data(uri) for uri in uris], ignore_index=True)  # ・・③
```

表5-1　ペンギンデータ

列名	説明
Species	ペンギンの種
Island	データを収集した島の名前
Individual_ID	個体識別番号（異種間では重複あり）
Date_Egg	調査年月日（巣に卵が確認された日）
Culmen_Length	くちばしの長さ（単位：mm）
Culmen_Depth	くちばしの太さ（単位：mm）
Flipper_Length	ひれの長さ（単位：mm）
Body_Mass	体重（単位：g）
Sex	性別
Comments	特記事項

読み込み結果の簡単なチェック

- 読み込んだデータが意図通りに読み込まれられているか、どういったデータなのかを確認する必要がある
- 小さいサイズのデータであればエディタで開き、大きいデータであればデータ提供元に確認するなどして把握しておく
- 部分的な表示だけでは完全にはチェックしきれないが、簡単に目視でチェックするだけでも異常を見つけられることがある

```
# 行数と列数を表示
df.shape
```

```
#データのタイプを表示
df.dtypes
```

```
# 先頭を表示
df.head()
```

```
#読み込んだデータの保存
df.to_parquet("data/penguins.parquet")
```

```
# 最後を表示
df.tail()
```

```
                    0
Species            object
Island             object
Individual_ID      object
Date_Egg           object
Culmen_Length     float64
Culmen_Depth      float64
Flipper_Length    float64
Body_Mass         float64
Sex                object
Comments           object

dtype: object
```

定量的評価（統計量）

05-02-quantitative.ipynb

要約統計量の確認

- DataFrameやSeriesからdescribe()メソッドを使うと一度に計算できる
- DataFrameでは、各列についての要約統計量を計算する

```
import pandas as pd

df = pd.read_parquet("data/penguins.parquet")
df.describe()
```

表5-2　主な要約統計量

種類	対応する pandas メソッド
件数	count()
最大値	max()
最小値	min()
平均値	mean()
中央値	median()
標準偏差	std()
ユニークな[※]値の数	nunique()
最頻値	mode()

※　同じ値（または値の組み合わせ）が1回だけ出現している状態のことです。「一意な」ともいいます。

	Date_Egg	Culmen_Length	Culmen_Depth	Flipper_Length	Body_Mass
count	344	342.000000	342.000000	342.000000	342.000000
mean	2008-11-27 03:46:02.790697728	43.921930	17.151170	200.915205	4201.754386
min	2007-11-09 00:00:00	32.100000	13.100000	172.000000	2700.000000
25%	2007-11-28 00:00:00	39.225000	15.600000	190.000000	3550.000000
50%	2008-11-09 00:00:00	44.450000	17.300000	197.000000	4050.000000
75%	2009-11-16 00:00:00	48.500000	18.700000	213.000000	4750.000000
max	2009-12-01 00:00:00	59.600000	21.500000	231.000000	6300.000000
std	NaN	5.459584	1.974793	14.061714	801.954536

定性的評価（可視化）

05-03-qualitative.ipynb

定性的評価（可視化）

- 定量的な評価だけでは把握しにくい分布の形状や値の変化、複数の変数の関係などを解説していく
- グラフを描くことでデータの内容を視覚的に把握することができる

データ可視化のためのライブラリ

- Python環境でデータを可視化するために使われる主なライブラリとして、pandas、Matplotlib、Plotlyなどがある

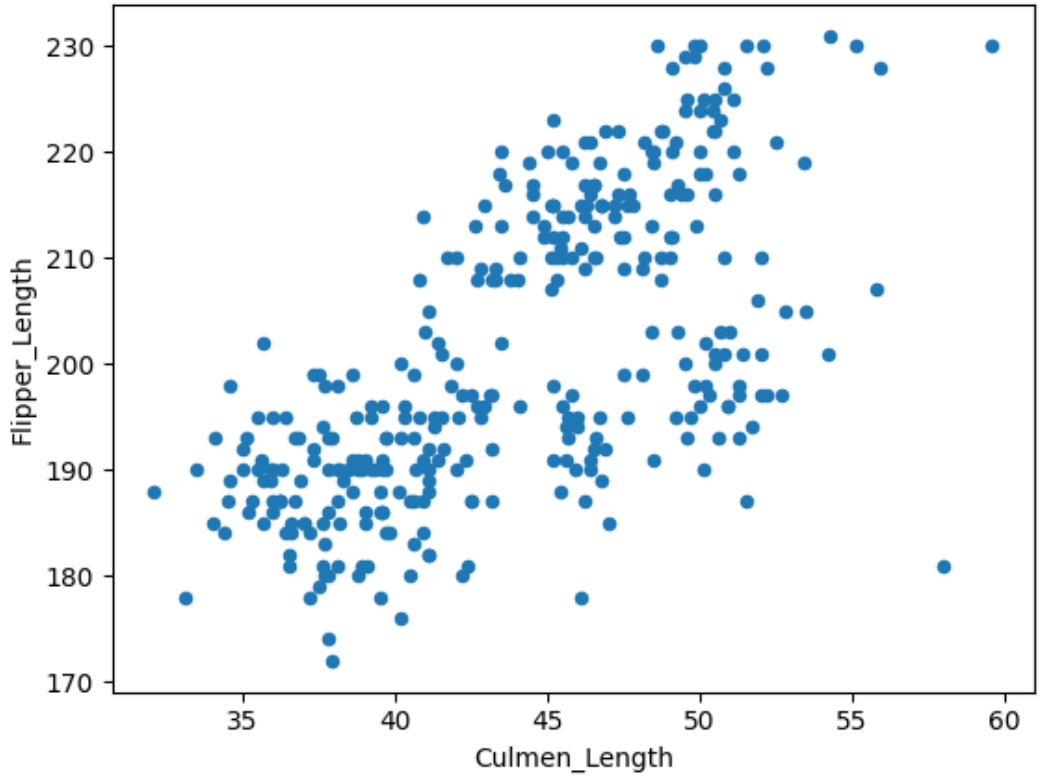
pandas.DataFrameのplot()

南極のペンギンに関するデータを使い、ペンギンのくちばしの長さ
Culmen_Lengthとひれの長さFlipper_Lengthの関係を散布図に描画する

```
import pandas as pd

df = pd.read_parquet("data/penguins.parquet")

# ペンギンのくちばしの長さ（横軸）とひれの長さ
（縦軸）の散布図
df.plot(
    x="Culmen_Length",
    y="Flipper_Length",
    kind="scatter",
)
```

表5-3　pandasの主な描画機能

種類	引数 kind の値
散布図	"scatter"
折れ線グラフ	"line"（デフォルト）
棒グラフ	"bar"
ヒストグラム	"hist"
箱ひげ図	"box"



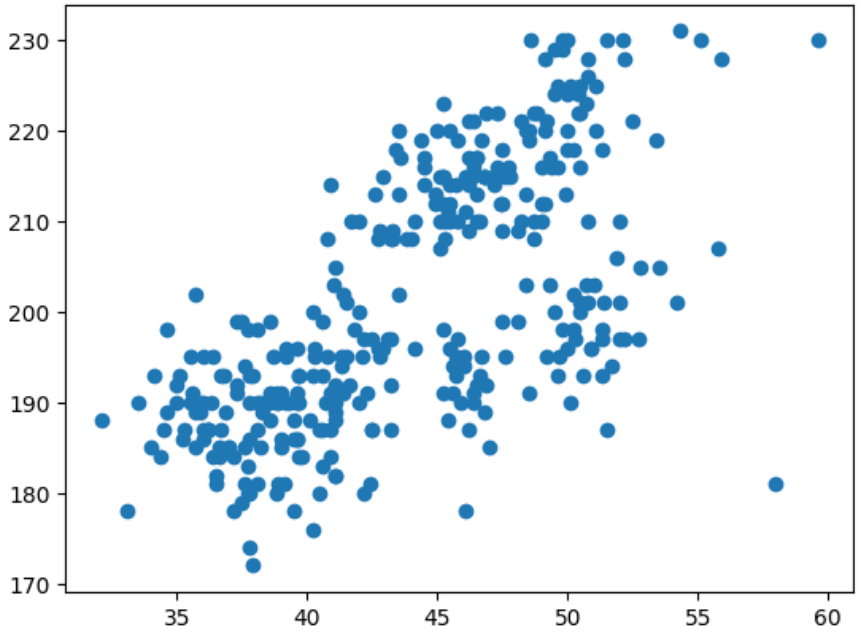
Matplotlibを使ったグラフ描画

- Matplotlibは、Python環境におけるデータ可視化の代表的なライブラリ
- Matplotlibを使ってDataFrameの値を描画するには、pyplotモジュールを使うことが一般的
- Axesインスタンスからグラフの種類に応じたメソッドを呼び出します。

```
import matplotlib.pyplot as plt

# 準備（subplotsコンストラクターを呼び出してAxesクラスのインスタンスを生成）
_, ax = plt.subplots()

# くちばしの長さ（横軸）とひれの長さ（縦軸）の散布図
ax.scatter(
    x=df.loc[:, "Culmen_Length"],
    y=df.loc[:, "Flipper_Length"],
)
plt.show()
```

表5-4　Matplotlibの主な描画メソッド

種類	メソッド
散布図	scatter()
折れ線グラフ	plot()
棒グラフ	bar()
ヒストグラム	hist()
箱ひげ図	boxplot()



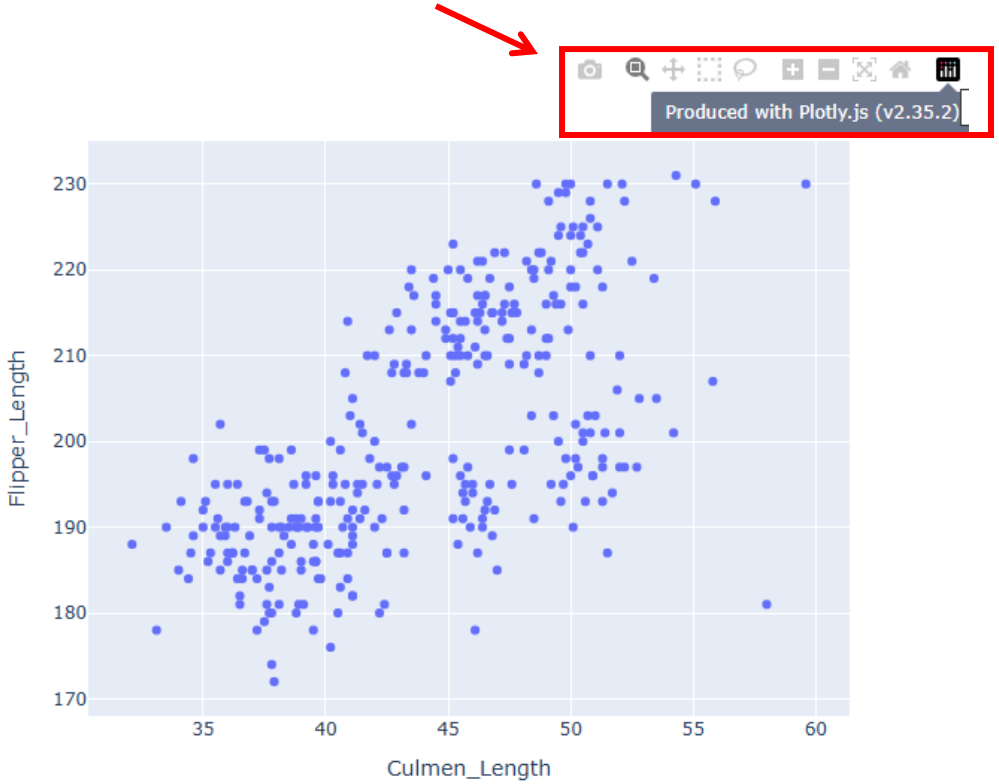
Plotlyを使ったグラフ描画

- Plotlyは、簡潔なコードでインタラクティブなデータ可視化を実現できるライブラリ
- 例えば、Plotly Expressというインタフェースを通してグラフの種類に応じた関数を使う

```
import plotly.express as px

# くちばしの長さ（横軸）とひれの長さ（縦軸）の散布図
px.scatter(
    df,
    x="Culmen_Length",
    y="Flipper_Length",
)
```

表5-5　Plotlyの主な描画関数

種類	関数
散布図	scatter()
折れ線グラフ	line()
棒グラフ	bar()
ヒストグラム	histogram()
箱ひげ図	box()



グラフの種類

データを可視化するは下記のグラフを使う

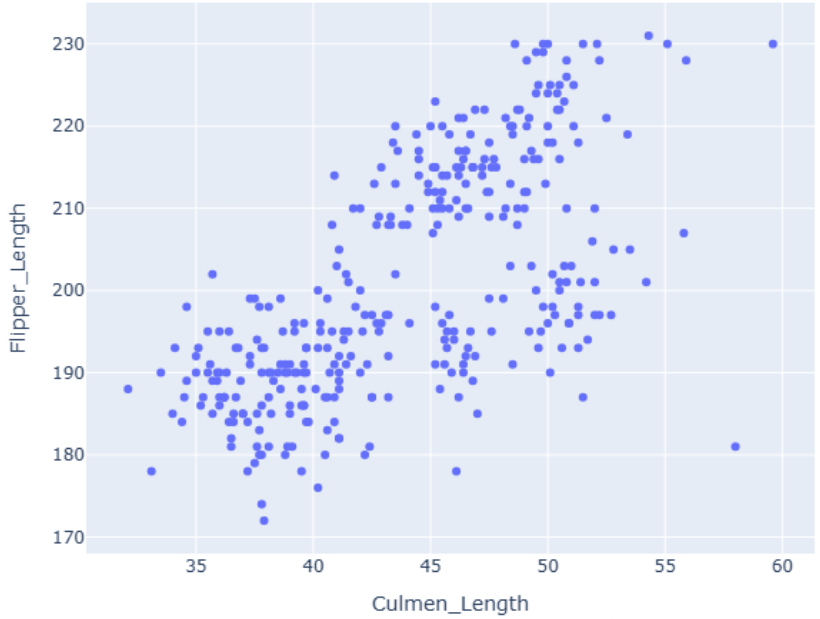
- 散布図
- 折れ線グラフ
- 棒グラフ
- ヒストグラム
- 箱ひげ図

データ可視化では、Plotly、pandas、Matplotlibを使用することが多い

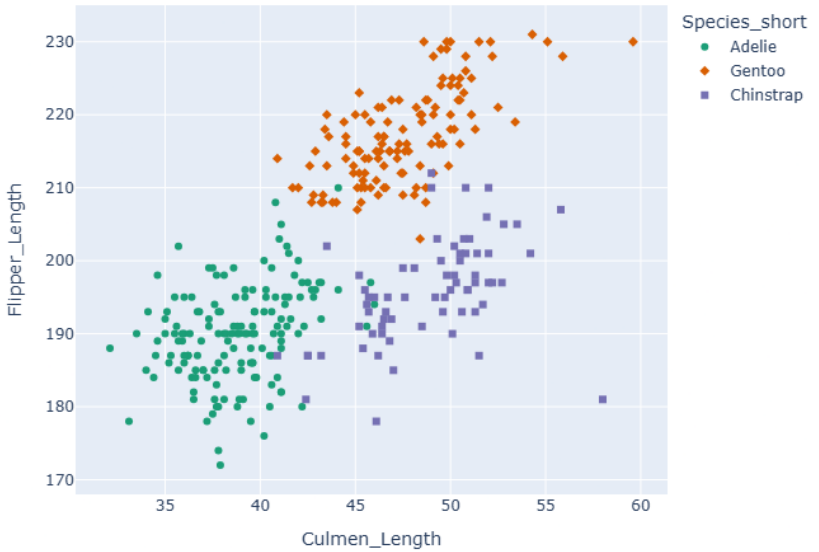
散布図

散布図は、2つの数値型の列どうしの関係を把握するために使う

```
# くちばしの長さ（横軸）とひれの長さ（縦軸）の散布図
px.scatter(
    df,
    x="Culmen_Length",
    y="Flipper_Length",
)
```



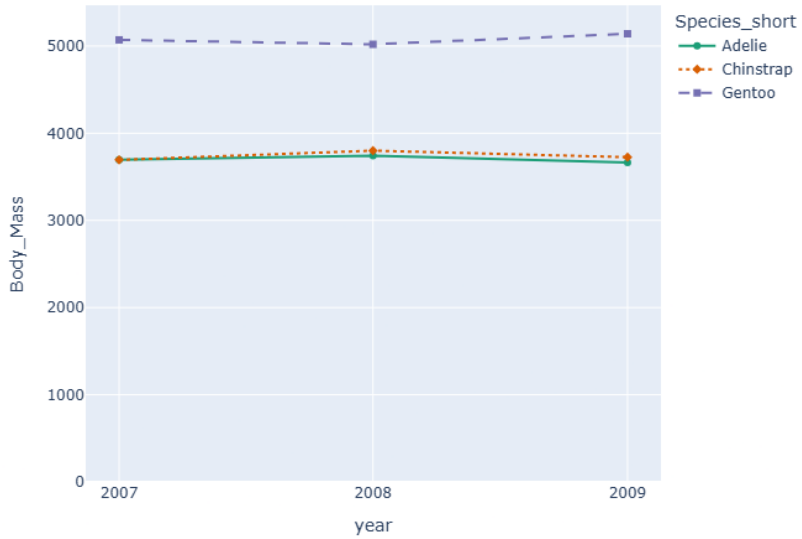

```
px.scatter(
    df,
    x="Culmen_Length",
    y="Flipper_Length",
    color="Species_short",
    color_discrete_sequence=px.colors.qualitative.Dark2,
    symbol="Species_short",
)
```



折れ線グラフ

折れ線グラフは、1つの数値型の列（縦軸）の値がもう1つの数値型もしくは日付型など時系列を表す列（横軸）の値に対して、どのように変化するかを把握するために使う

```
#平均体重の計算
df_avg_weight_year = (
    df.assign(year=df.loc[:,
"Date_Egg"].dt.to_period("Y"))
    .groupby(["Species_short", "year"],
as_index=False)["Body_Mass"]
    .mean()
)
```

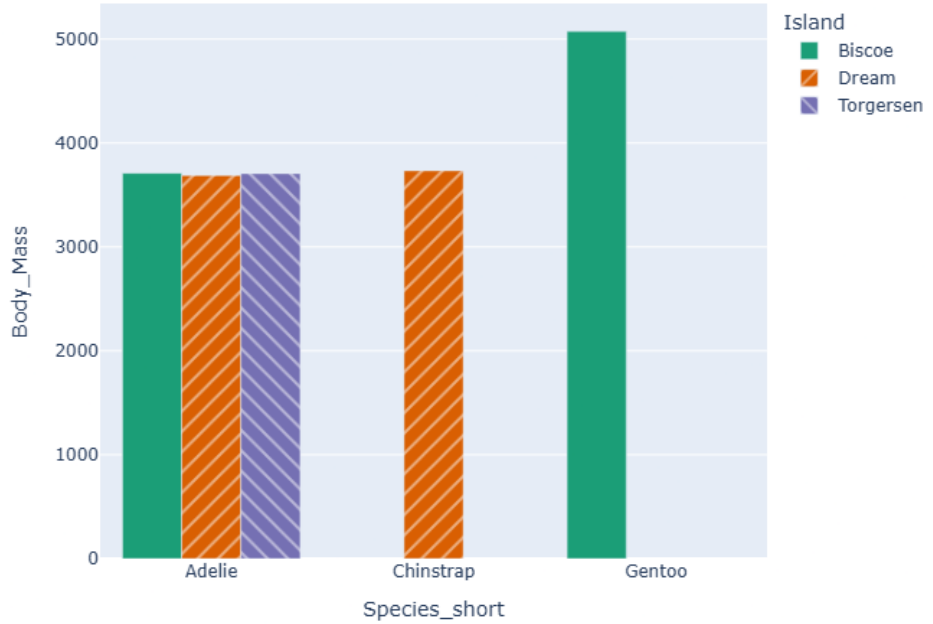



```
#折れ線グラフの描画
fig_line = px.line(
    df_avg_weight_year.assign(
        year=lambda x: x.loc[:,
"year"].dt.to_timestamp()
    ),
    x="year",
    y="Body_Mass",
    color="Species_short",
    color_discrete_sequence=px.colors.qualitative
.Dark2,
    line_dash="Species_short",
    symbol="Species_short",
)

fig_line.update_yaxes(rangemode="tozero")  #
縦軸の最小値を0にする
fig_line.update_xaxes(
    tickformat="%Y", dtick="M12"
)  # 目盛に年だけを表示し、間隔を12か月に設定
fig_line.show()
```

棒グラフ

棒グラフは、1つの数値型の列（縦軸）の値がもう1つのカテゴリー値の列（横軸）の値に応じて、どのように異なるかを比較するために使う。原則として、縦軸の最小値は0にする。

```
#平均体重の計算
df_avg_weight_island = df.groupby(
    ["Species_short", "Island"],
as_index=False
)["Body_Mass"].mean()
```

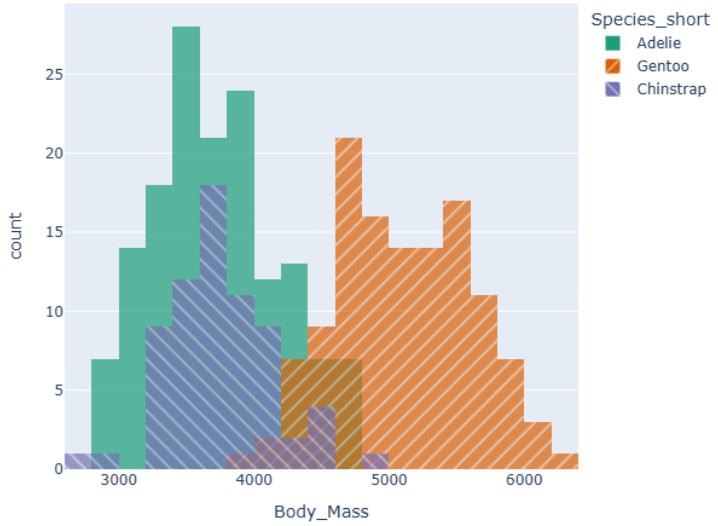



```
#棒グラフの描画
px.bar(
    df_avg_weight_island,
    x="Species_short",
    y="Body_Mass",
    color="Island",
    color_discrete_sequence=px.colors.qualitati
ve.Dark2,
    pattern_shape="Island",
    barmode="group",
)
```

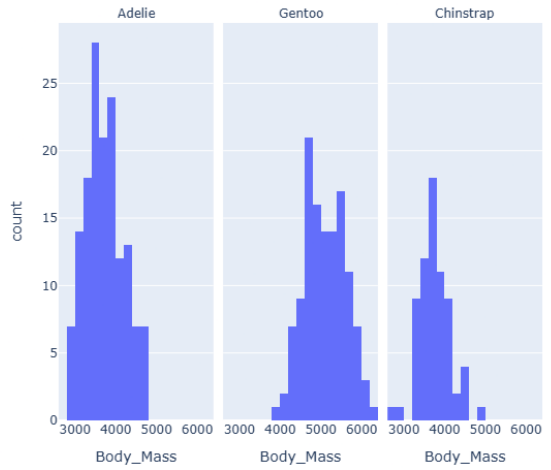
ヒストグラム

ヒストグラムは、1つの数値の列の値の度数分布を棒グラフとして表したもの

```
px.histogram(
    df,
    x="Body_Mass",
    color="Species_short",
    color_discrete_sequence=px.colors.qualitative.Dark2,
    pattern_shape="Species_short",
    opacity=0.7,
    barmode="overlay",
)
```



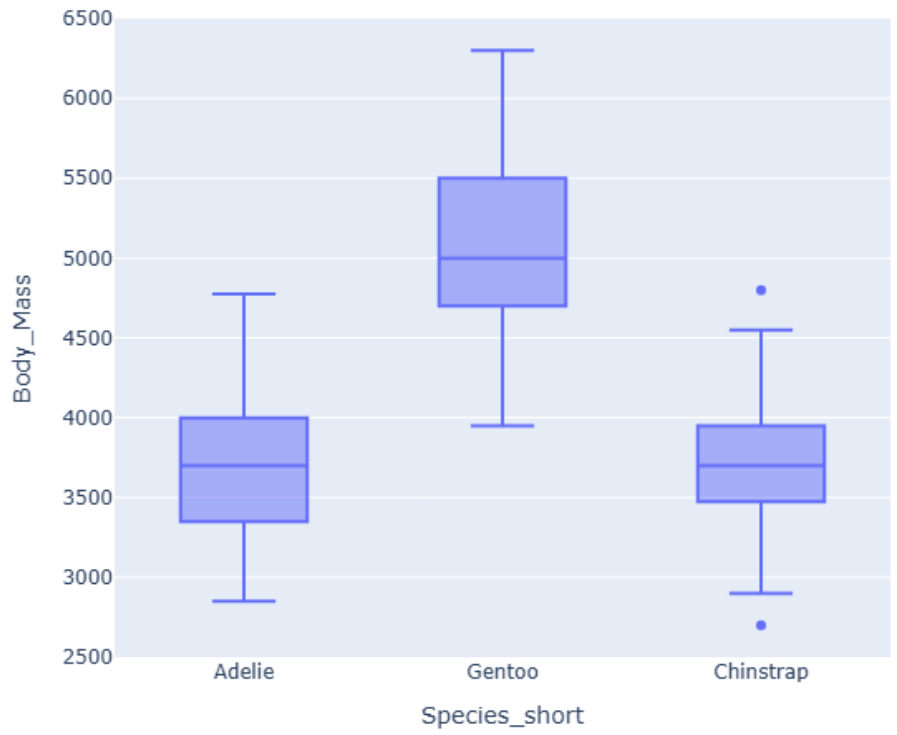

```
# 種ごとのヒストグラムを横に並べる
fig_col = px.histogram(
    df,
    x="Body_Mass",
    facet_col="Species_short",
)
import re

RE_TITLE = re.compile(
    r"(?<=Species_short=).+"
)  # 「Species_short=」に続く任意の文字列にマッチ
fig_col.for_each_annotation(
    lambda a: a.update(text=re.search(RE_TITLE, a.text)[0])
)
fig_col.show()
```



箱ひげ図

- 箱ひげ図は、ヒストグラムと同様に1つの数値列の値の分布を分位点について表したもの。
- ヒストグラムが分布の詳細（階級ごとの度数）を表すのに対し、箱ひげ図は第1四分位点（25%点）、中央値（第2四分位点、50%点）、第3四分位点（75%点）などを模式的に表す

```
px.box(df, x="Species_short", y="Body_Mass")
```



外れ値、異常値

05-05-outliers.ipynb

外れ値、異常値

- データの外れ値や異常値を、統計的、視覚的に把握する方法と対処方法について解説する
- 外れ値がデータ分析や機械学習モデル構築に影響を及ぼす要因をチェックするために、不可欠なステップ
- 外れ値とは、データ全体の分布から見て、ほかの値から大きく外れた値のこと
- 一方、異常値とは、データの測定時や記録時のエラーによって発生した特異な値のこと
- ただし、どちらも、ほかの値より離れた範囲に存在することが多いため、ここではまとめて「外れ値」と呼ぶ

外れ値の確認

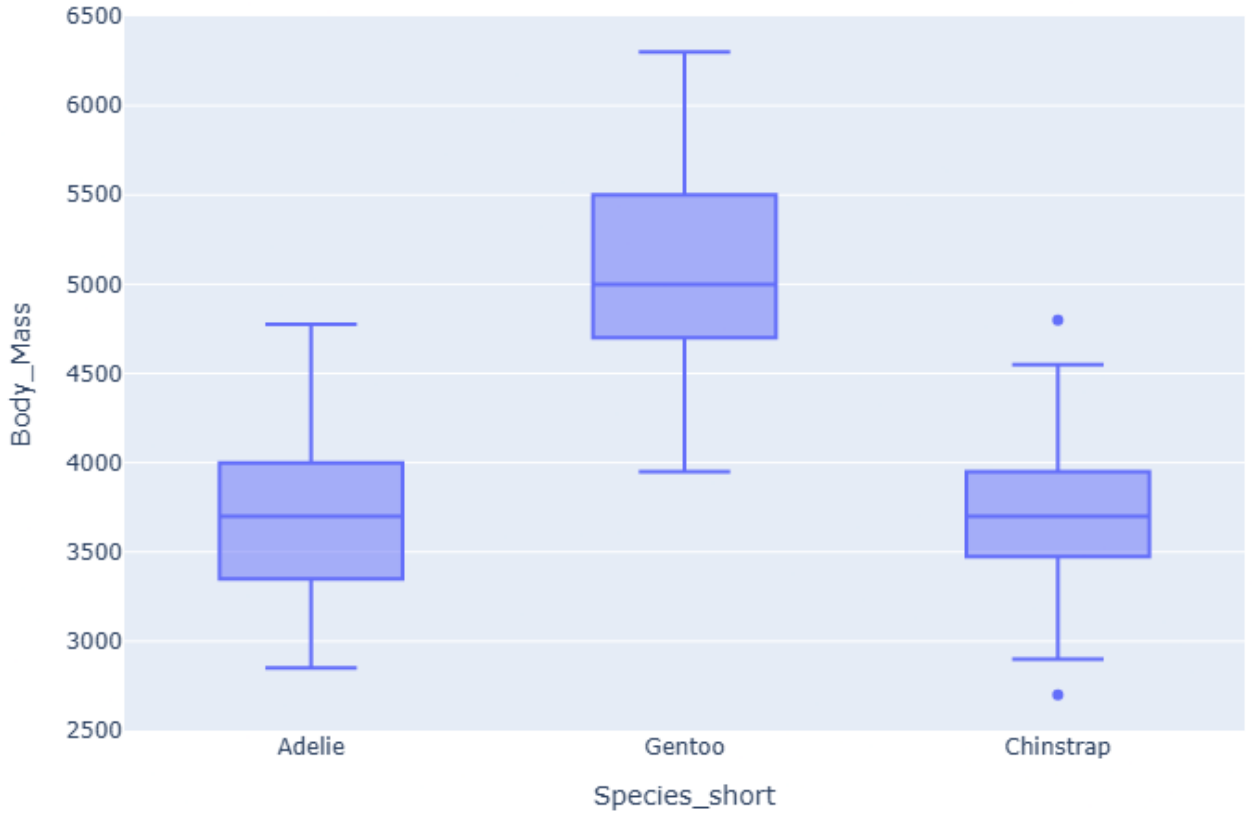
- 箱ひげ図やヒストグラムを描画することで定性的に外れ値の存在を把握できる
- 定量的に確認するには次の方法がある
  - 分位点
  - 正規分布と2σ範囲
    - σ(シグマ)は標準偏差を表す記号

箱ひげ図による外れ値の確認

- 箱ひげ図はその名前のとおり、「箱」と「ひげ」（箱から出る線）でデータの分布を簡単に表現した図
- 箱は、下記からなる
  - 第1四分位点（25%点）
  - 中央値（第2四分位点、50%点）、
  - 第3四分位点（75%点）
- ひげの長さは、よく使われる指標で、Plotlyやpandas、Matplotlibの箱ひげ図では箱の長さの最大1.5倍である
- 箱ひげ図では、ひげの外側にある値を外れ値とみなす
- 次の分位点による外れ値の確認で解説するquantile()メソッドを使って第1四分位点と第3四分位点の値を求めると、箱ひげ図のひげの長さが計算できる

箱ひげ図による外れ値の確認

```
import pandas as pd
import plotly.express as px

df = pd.read_parquet("data/penguins.parquet")
px.box(df, x="Species_short", y="Body_Mass")
```



分位点による外れ値の確認

- 外れ値を定量的に見つける代表的な手段として、分位点（%点）がある
- 分位点とは、データのサンプルサイズを100としたときに小さいほうから数えて何番目に当たるかを表したもの
- 最小値が0%点、中央値が50%点、最大値が100%点となる
- 100%を4等分した25%点を第1四分位点、50%点を第2四分位点、75%点を第3四分位点と呼ぶ
- pandasのSeriesはquantile()メソッドを使って分位点の値を計算できる。計算したい分位点を0から1の範囲の値としてリストで第1引数qに渡す。

分位点による外れ値の確認

- 例として、ペンギンの種ごとに、体重の0、1、5、95、99、100の各%点を計算する
- ここではデフォルトの線形補間を使い、引数interpolationは指定しない

```
df.groupby("Species_short")["Body_Mass"].quantile(
    q=[0, 0.01, 0.05, 0.95, 0.99, 1]
)
```

Species_short		Body_Mass
Adelie	**0.00**	2850.0
	0.01	2875.0
	0.05	3000.0
	0.95	4487.5
	0.99	4712.5
	1.00	4775.0
Chinstrap	**0.00**	2700.0
	0.01	2834.0
	0.05	3250.0
	0.95	4432.5
	0.99	4632.5
	1.00	4800.0
Gentoo	**0.00**	3950.0
	0.01	4111.0
	0.05	4300.0
	0.95	5850.0
	0.99	6039.0
	1.00	6300.0

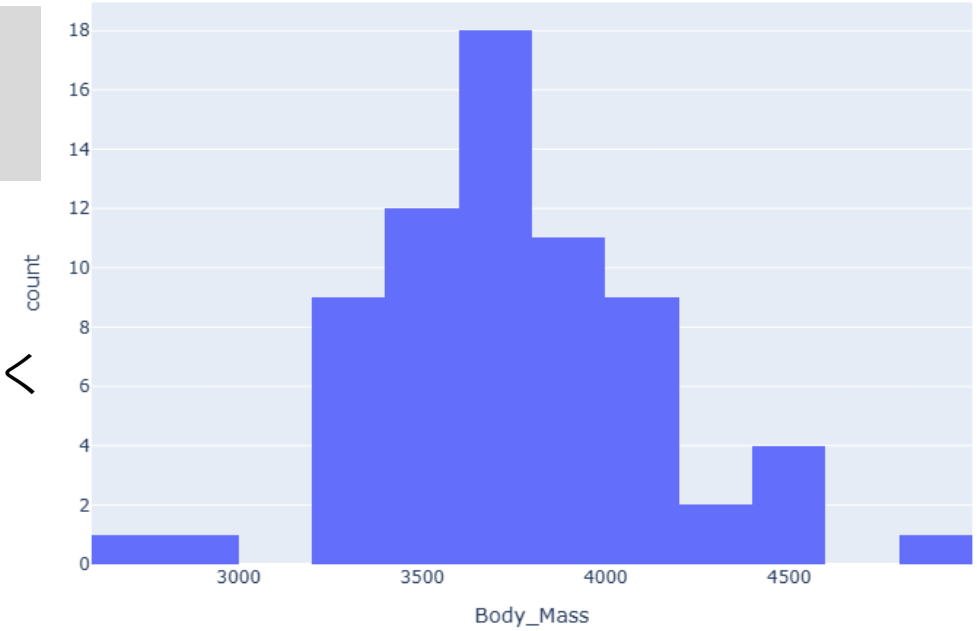
もし外れ値がないとすると、0%点（最小値）と1%点と5%点、95%点と99%点と100%点（最大値）はほぼ同じ値のはずです。しかし、この場合、ChinstrapとGentooの95%点、99%点、100%点の値をそれぞれ比べると、100%点の値が大きく異なっていて、外れ値である可能性がある

正規分布と2σ範囲による外れ値の確認

- 分布の正規性を仮定できる場合、標準偏差σ(シグマ)に基づいて外れ値を特定できる
- 正規分布において平均値から±2σの範囲には約95.5%のデータが含まれるので、この範囲外にある値は外れ値と考えて問題ない

```
df_chinstrap = df.loc[df.loc[:, "Species_short"] == "Chinstrap", :]

px.histogram(df_chinstrap, x="Body_Mass")
```

箱ひげ図との比較のために、Chinstrapの体重のヒストグラムを描く



外れ値への対処方法

外れ値が特定できたら、次のいずれかの方法で処理する

- データ（行）の削除
- 上限値と下限値の設定

データ（行）の削除

- 外れ値が何らかのエラーによるもの（異常値）と見なせる場合で、かつ、サンプルサイズが十分に大きい場合には、外れ値を含むデータ（行）を削除してもよい
- 表のように、非常に大きいか小さい身長の値は、「データ記録時に0を1つ多くまたは少なく入力してしまった」など何らかのエラーが発生したと考えられる
- 原因が明らかである場合には、値そのものを修正することも対処方法の1つ
- そもそもこのような外れ値が多い場合には、データそのものの品質が低く、データ分析や機械学習モデル構築に適していない可能性が高いので、あらためてデータの取得、入手から始めるほうが安全である

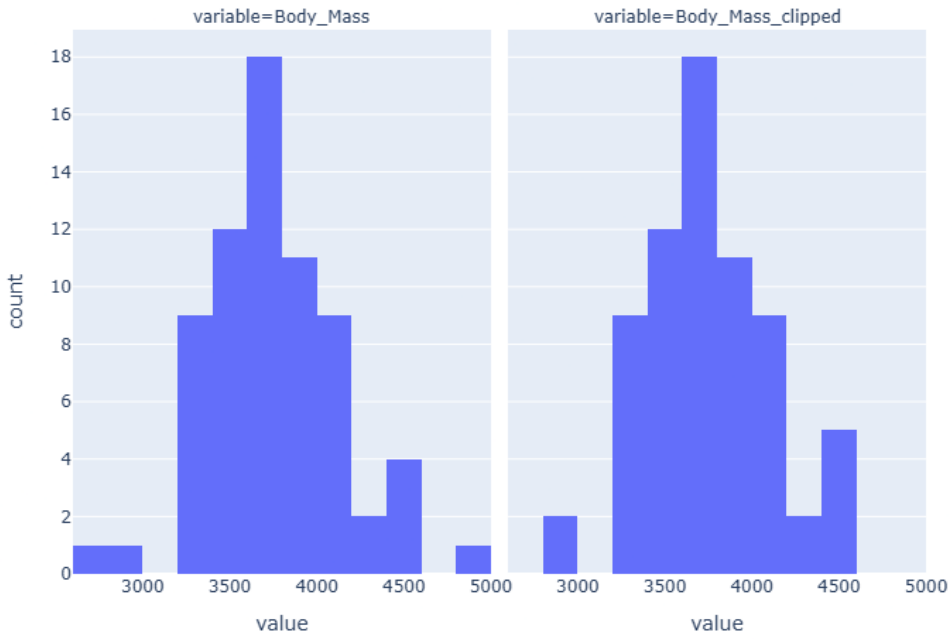
成人の身長(入力エラーが疑われる例)

id	身長
1	140
2	1700
3	180
4	19
…	…

上限値と下限値の設定

- 2σ範囲のしきい値か分位点より大きいか小さい値を外れ値と見なせる場合、その外れ値をしきい値か分位点の値で置き換えることも、外れ値に対する有効な対処方法である
- たとえば、Seriesのclip()メソッドを使って、先ほどのChinstrapの体重の外れ値を2σ範囲のしきい値で置き換える。引数lowerに下限値、引数upperに上限値を渡す

```
df_chinstrap = df_chinstrap.assign(
    Body_Mass_clipped=df_chinstrap.loc[:, "Body_Mass"].clip(
        lower=avg - 2 * sigma,
        upper=avg + 2 * sigma,
    )
)

px.histogram(
    df_chinstrap.melt(
        id_vars="Individual_ID",
        value_vars=["Body_Mass", "Body_Mass_clipped"],
    ),
    x="value",
    facet_col="variable",
)
```



上下限値設定前後のヒストグラム

欠損値

05-06-missing-values.ipynb

欠損値

- 実務上、何らか値の欠損があるデータを扱うことがある
- 欠損値のない入力データを前提としている機械学習モデルは多く、欠損値の取り扱いは最終的な分析結果やモデルに大きな影響を与える
- とりうる手段を理解し適切に処理することが重要である
- 欠損値のデータ型の性質と、データの欠損値を把握して対処する方法について解説する

欠損値の基礎知識

データ型

pandasとNumPyで扱う場合のデータ型に応じて欠損値として扱われるデータは下記のとおり

欠損値のデータ型

表記	データ型	クラス
NaN	float型	numpy.nan
NA	Int型	pandas.NA *
NaT	datetime型	pandas.NaT

※ pandas.NAクラスは、Experimentalという扱いであり、仕様が変わる可能性がある

欠損値の生成

```
#欠損値（NaN）を含むfloat型のSeriesを生成
import numpy as np
import pandas as pd

rng = np.random.default_rng(1)
float_ser = pd.Series(rng.random(4), index=range(0, 8, 2)).reindex(range(4))
float_ser
```

0	0.511822
1	NaN
2	0.950464
3	NaN

```
# int型のSeriesは、欠損値が含まれるとfloat型に型変換される
#欠損値（NaT）を含むdatetime型のSeriesを生成
dt_ser = pd.Series(
    pd.date_range("2023-01-01", periods=4),
    index=range(0, 8, 2),
).reindex(range(4))
dt_ser
```

0	2023-01-01
1	NaT
2	2023-01-02
3	NaT

```
#引数dtypeにpandas.Int64Dtypeを渡すことで、欠損値（NA）を含むint型のSeriesを生成
int_ser = pd.Series(
    rng.integers(0, 10, 4),
    index=range(0, 8, 2),
    dtype=pd.Int64Dtype(),
).reindex(range(4))
int_ser
```

0	2
1	<NA>
2	3
3	<NA>

データの型変換

pandasでは、int型およびbool型のデータに欠損値が含まれている場合は、自動的に型変換が行われる

欠損値が含まれたデータの型変換

データ型	変換後のデータ型
int型	float型
bool型	object型
float型	float型
object型	object型

欠損値を含むデータの評価

- NaNどうし、またはNaTどうしの比較演算は、Falseが返る
- NAどうしの比較演算は、NAが返る

```
np.nan == np.nan
```

False

```
np.nan > np.nan
```

False

```
pd.NaT == pd.NaT
```

False

```
pd.NA == pd.NA
```

<NA>

欠損値を含むデータの演算

- 欠損値であるかどうかを確認するには、isna()またはisnull()関数（またはメソッド）を使う

```
pd.isna(float_ser)
# or
pd.isnull(float_ser)
# or
float_ser.isna()
# or
float_ser.isnull()
```

0	False
1	True
2	False
3	True

```
dt_ser.isna()
```

0	False
1	True
2	False
3	True

```
int_ser.isna()
```

0	False
1	True
2	False
3	True

欠損値を含むデータの演算

欠損値を含むSeriesに対して、記述統計を行うメソッドを実行した場合やグループ化した場合は、次のような処理になる

- データを集計する場合、欠損値は0として扱われる
- すべてのデータが欠損値の場合は0となる
- 累積演算（cumsum()、cumprod()など）では欠損値を無視する

```
int_ser.sum()
```

```
5
```

```
int_ser.cumsum()
```

0	2
1	<NA>
2	5
3	<NA>

```
int_ser.sum(skipna=False)
```

```
<NA>
```

欠損値の発生パターン（メカニズム）と対処方法

実世界のデータに欠損値が生じるパターンと対処方法について解説する。
ここでは、欠損値の発生は、次の3パターンに分けられる

- MCAR（Missing Completely At Random）：完全にランダムに発生する場合
  - ある列の欠損値の発生がほかの列の値によらない場合
  - 欠損値が仮に欠損値ではなかった場合の分布がもともと欠損ではない値の分布と同じだと見なせる
  - つまり、この場合の欠損値は無視できる
- MAR（Missing At Random）：条件はあるがランダムに発生する場合
  - ある条件下ではMCARである場合
  - その条件についてのみ考える場合には欠損を無視できるが、そうでない場合には無視できない
- NMAR（Not Missing At Random）：ランダムには発生しない場合
  - 系統的に欠損値が発生する場合、欠損値を無視できない。
  - 実務上はMCARであることはあまりなく、MARまたはNMARであることが多い

欠損値の確認

例として、南極のペンギンに関するデータを使用する

```
df = pd.read_parquet("data/penguins.parquet")

print(df.isna().sum())
```

```
Species            0
Island             0
Individual_ID      0
Date_Egg           0
Culmen_Length      2
Culmen_Depth       2
Flipper_Length     2
Body_Mass          2
Sex               11
Comments         290
Species_short      0

dtype: int64
```

Culmen_Length、Culmen_Depth、
Flipper_Length、Body_Massは2個ずつ、
Sexは11個、Commentsは290個の
欠損値があることがわかる

欠損値の発生が完全にランダム（MCAR）な場合

MCARの場合には、欠損値を無視する、つまり欠損値を含む行を次の2つの方法で削除できる

- ペアワイズ：対象とする列のいずれかに欠損がある行を削除
  - DataFrameのdropna()メソッドの引数subsetに列名をリストとして渡す
- 完全セット（リストワイズ）：対象としないものも含めて、いずれかの列に欠損がある行を削除
  - DataFrameのdropna()メソッドの引数subsetを指定しない（Noneを渡す）

欠損値の発生が何らかの原因による場合

MARやNMARの場合、欠損の発生が何らかの原因によるものと考えられるので、欠損値を補完することを解説する

単変量補完と多変量補完

- 単変量補完：単純に「その列だけ」で埋める（平均・中央値など）
- 多変量補完：他の列の情報も利用して「より妥当な値」を埋める。

単変量補完と多変量補完

単変量補完（univariate imputation）

欠損値補完（missing value imputation）の一種で「その変数だけの情報を使って欠損を埋める方法

代表的な手法

- **平均値補完**：欠損をその列の平均値で埋める。
- **中央値補完**：外れ値の影響を避けたいときに中央値を使う。
- **最頻値補完**：カテゴリデータに多い方法で、一番よく出てくる値で埋める。
- **固定値補完**：0や"Unknown"など、任意の一定値で埋める。

単変量補完と多変量補完

多変量補完（multivariate imputation）

欠損している変数を他の変数との関係を利用して補完する方法。単変量補完と違い、対象の列だけでなく、データセット全体の相関構造を考慮する点が特徴

代表的な手法

- **回帰補完**
  欠損した列を目的変数とみなし、他の変数から回帰モデルを作って推定値を補完。（例：年収が欠損 → 年齢・学歴・職業から回帰して推定）
- **多重代入法（MICE: Multiple Imputation by Chained Equations）**
  複数の変数に欠損がある場合、連鎖的に1つずつ補完し、これを何回も繰り返して不確実性を反映させる。機械学習や統計解析でよく使われる。
- **機械学習を使った補完**
  ランダムフォレスト、k近傍法（kNN）、ベイズモデルなどを用いて欠損を推定。

行どうしに順序がある場合の補間

- 時系列データなど、欠損値のある行とその前後の行の値が何らかの関係性を持っていることを仮定できる場合、その関係性に基づいて欠損値を補間できる
- 欠損値を削除するよりも、合理的に多くのデータを分析やモデル構築に使える
- ただし、欠損区間が長い場合は、最初にその原因を調査する

- 前後いずれかの行の値による補間
  - 前の行の値による補間をforward fillingと呼ぶ
  - 後ろの行の値による補間をbackward fillingと呼ぶ
- 線形補間
  - 欠損区間において、値の変化が単調であることを前提とする
  - 欠損が1か所の場合は、前後の行の値の平均値と同等
  - 線形補間以外にも多項式補間など、さまざまな手法がある