第3章 表形式データの加工

表形式のデータの加工方法

データの連結／結合

データの連結／結合

- DataFrameのデータの連結／結合方法
- 複数のデータを1つにまとめる
- concat, join, merge関数を使う

表3-1　DataFrameの連結／結合

関数／メソッド	連結／結合	連結／結合方向	デフォルトの結合方法	キー
concat()関数	連結、結合	行方向、列方向	外部結合	インデックス
join()メソッド	結合	列方向	左外部結合	インデックス（デフォルト）、列
merge()関数	結合	列方向	内部結合	列（デフォルト）、インデックス

DataFrameの連結

- 「連結」とはデータを行（縦）方向または列（横）方向に接続する処理を指す
- 行方向の連結はDataFrameに行を追加
- 列方向の連結はDataFrameに列を追加

名前	科目	点数
寺田	国語	50
辻	国語	70
神沢	国語	90

名前	科目	点数
寺田	算数	60
辻	算数	80
神沢	算数	100

名前	科目	点数
寺田	国語	50
辻	国語	70
神沢	国語	90
寺田	算数	60
辻	算数	80
神沢	算数	100

図3-1　行（縦）方向に連結

名前	科目	点数
寺田	国語	50
寺田	算数	60
辻	国語	70
辻	算数	80
神沢	国語	90
神沢	算数	100

名前	科目	点数
寺田	国語	50
寺田	算数	60
辻	国語	70
辻	算数	80
神沢	国語	90
神沢	算数	100

図3-2　列（横）方向に連結

DataFrameの結合

- 「結合」とはキーを持つデータから特定のキーをもとに要素を取り出し、結合する処理を指す
- 結合処理はマージ処理とも呼ばれる
- 結合した結果、キーに該当する要素が存在しない場合は、欠損値として扱われる

出席番号	名前
1	寺田
2	辻
3	神沢

出席番号	点数
1	50
2	60
4	70

結合

内部結合

出席番号	名前	点数
1	寺田	50
2	辻	60

外部結合

出席番号	名前	点数
1	寺田	50
2	辻	60
3	神沢	NaN
4	NaN	70

左外部結合

出席番号	名前	点数
1	寺田	50
2	辻	60
3	神沢	NaN

右外部結合

出席番号	名前	点数
1	寺田	50
2	辻	60
4	NaN	70

図3-3　DataFrameの結合

「出席番号」の列をキーとして「名前」と「点数」の列を結合

concat()関数によるDataFrameの連結

- concat()関数は、第1引数に渡したリストに含まれたDataFrameを連結して返す
- **引数axisに0**を渡すと行方向（デフォルト）に連結する
- **引数axisに1**を渡すと列方向列方向に連結する
- ignore_index=Trueを指定すると元のDataFrameのインデックスを無視して0から始まる連番に振り直す
- ignore_index=Trueを指定しない場合は、結合されたDataFrameのインデックスは元のDataFrameのものが維持される

図3-4　concat()関数によるDataFrameの連結

concat()関数によるDataFrameの結合

- concat()関数を実行してDataFrameを列方向に結合する場合は、引数axisに1を渡す
- インデックスがキーになる
- デフォルトの結合方法は、外部結合（"outer"）
  - **inner：内部結合** 結合対象のキーがすべて存在する要素を結合
  - **outer：外部結合** 結合対象のすべての要素を結合
  - 内部結合する場合は、concat()関数の引数joinに“inner”を渡す

Index	名前	科目
0	寺田	国語
2	辻	国語
3	辻	算数
4	神沢	算数
5	神沢	算数

Index	点数
0	50
1	60
3	80
4	90
5	100

concat()　axis = 1

Index	名前	科目	点数
0	寺田	国語	50
1	NaN	NaN	60
2	辻	国語	NaN
3	辻	算数	80
4	神沢	国語	90
5	神沢	算数	100

図3-5　concat()関数によるDataFrameの結合

join()メソッドによるDataFrameの結合

- DataFrameのjoin()メソッドの引数に、結合対象のDataFrameを渡し、2つのDataFrameを結合して返す
- join()メソッドの結合方法はデフォルトで左外部結合（"left"）結合方法は引数howで指定
  - **left：左外部結合** 結合元の結合対象のキーが存在する要素を結合
  - **right：右外部結合** 結合先の結合対象のキーが存在する要素を結合
  - **inner：内部結合**
  - **outer：外部結合**

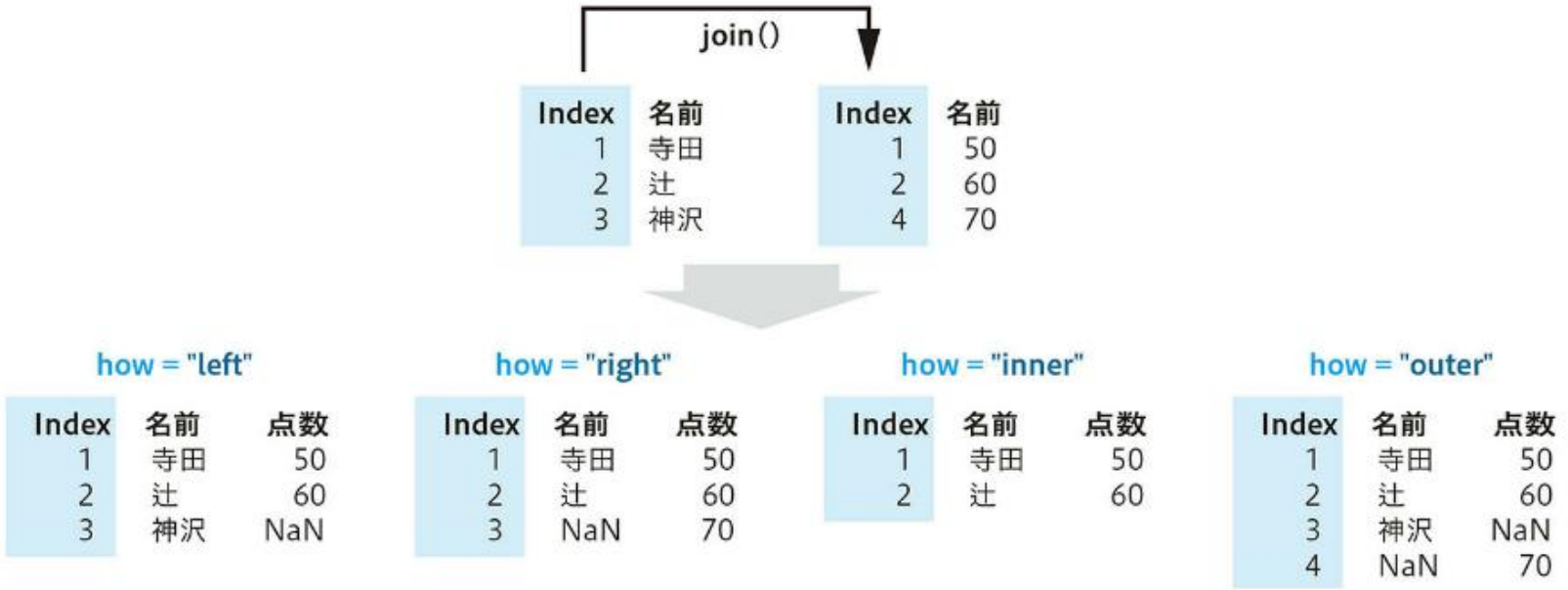


図3-6　join()メソッドによるDataFrameの結合

merge()関数によるDataFrameの結合

- merge()関数の第1引数（left）および第2引数（right）を結合して返す
- デフォルトで内部結合（“inner”）。結合方法は引数howで指定
  - left：左外部結合
  - right：右外部結合
  - inner：内部結合
  - outer：外部結合
- merge()関数は、デフォルトでは引数onに渡した列をキーとして結合する

出席番号	名前
1	寺田
2	辻
3	神沢

出席番号	点数
1	50
2	60
4	70

merge()

how = "inner"

出席番号	名前	点数
1	寺田	50
2	辻	60

how = "outer"

出席番号	名前	点数
1	寺田	50
2	辻	60
3	神沢	NaN
4	NaN	70

how = "left"

出席番号	名前	点数
1	寺田	50
2	辻	60
3	神沢	NaN

how = "right"

出席番号	名前	点数
1	寺田	50
2	辻	60
4	NaN	70

図3-7　merge()関数によるDataFrameの結合

重複したキーの検証

- merge()関数の引数validateに“one_to_one”を渡すと、キーの値が1：1であることを検証する
- 重複があるとMergeError例外が送出される
- merge()関数の引数validateに“one_to_many”を渡すと、キーの値が1：n（マージ元のキーが一意）であることを検証する

例）one_to_one

```
pd.merge(df1, df5, on="A", validate="one_to_one")
```

例）one_to_many

```
pd.merge(df1, df5, on=“A”,  validate="one_to_many" )
```

近い値での結合

- merge_asof()関数は、引数onで指定した列の値が近い要素を結合
- 引数toleranceに結合するための公差（基準値と許容される範囲の最大値と最小値の差）を指定

```
pd.merge_asof(
    ts_df1,
    ts_df2,
    on="timestamp",
    tolerance=pd.Timedelta("2s"),
)
```

「timestamp」列から2秒以内の値の要素を結合したDataFrameを生成している

DataFrame結合時の関数／メソッドの選び方

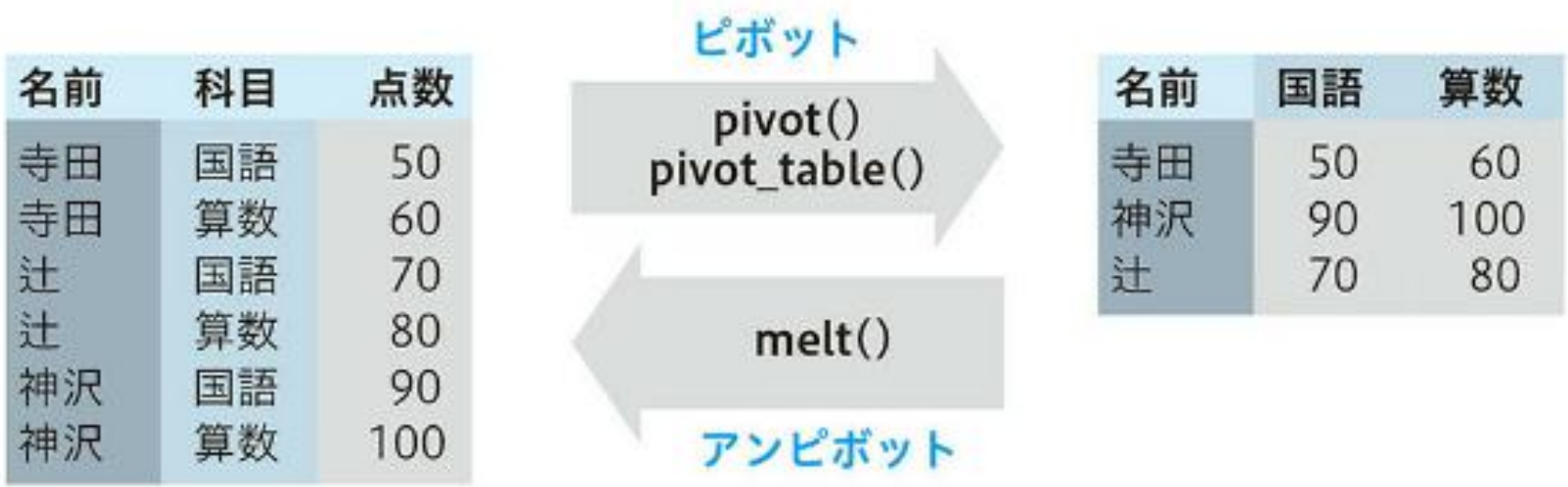
concat()関数、join()メソッド、merge()関数は、DataFrameの構造や結合条件に応じて使い分ける

- **concat()関数**
  - DataFrameのインデックスがキーとして使える場合は、concat()関数を検討する
  - インデックスをキーとすることで、高速な処理が期待できる
  - concat()関数に渡すリストには3つ以上のDataFrameを入れられるため、多数のデータをまとめて処理する場合にも使える
- **join()メソッド**
  - DataFrameのインデックスがキーとして使え、より複雑な条件で結合処理をする場合はjoin()メソッドを検討する
  - キーが列とインデックスの組み合わせの場合もjoin()メソッドが使える
- **merge()関数**
  - DataFrameの列をキーとして結合する場合は、merge()関数が使える

データの変形

ピボットとアンピボット

- ピボットは、表形式データの列の値に基づいてデータを再形成する処理
- アンピボットは、ピボットされたデータを解除する処理
- 「名前」を行にとり、「科目」を列にとって変形した処理が「ピボット」になり、その逆の処理が「アンピボット」になる



名前	科目	点数
寺田	国語	50
寺田	算数	60
辻	国語	70
辻	算数	80
神沢	国語	90
神沢	算数	100

名前	国語	算数
寺田	50	60
神沢	90	100
辻	70	80

図3-8　ピボットとアンピボット

ピボット

- ピボット処理は、pivot()メソッドやpivot_table()メソッドで行う
- crosstab()関数は、pivot_table()メソッドと同様の処理を行える
- 引数index、columns、valuesにarray-likeを渡すことで集計する
  - data（第一引数）: 元データのpandas.DataFrameオブジェクトを指定
  - index: 元データの列名を指定。結果の行見出しとなる
  - columns: 元データの列名を指定。結果の列見出しとなる
- 引数aggfuncを省略すると、平均値（numpy.mean）を算出する
- 引数aggfunc=“median”は中央値を算出する
- 引数aggfunc=“count”はデータ個数を数え上げる
- 引数marginsをTrueとすると、各カテゴリごとの結果（小計）および全体の結果（総計）が算出できる
- crosstab()関数を使うとクロス集計分析ができる
  - normalize=True を指定すると、クロス集計結果が全体の合計値が1になるように規格化（正規化）される

アンピボット

- アンピボットは、ピボットされたデータを解除する処理
- ピボットされたデータをアンピボットするには、DataFrameからmelt()メソッドで実行する

表3-2　melt()メソッドの引数

引数名	説明
id_vars	列として識別する列名
value_vars	ピボットを解除する列名、指定しない場合はすべての列を使用する
var_name	value_varsに付ける列名
value_name	値の列に付ける列名、省略すると"value"が付けられる

スタックとアンスタック

- スタックは、複数列から構成されるデータを1次元に積み上げる処理
- アンスタックは、階層化されたインデックスを列に展開する処理
- スタックしたデータはアンスタックすると、元のデータに戻る

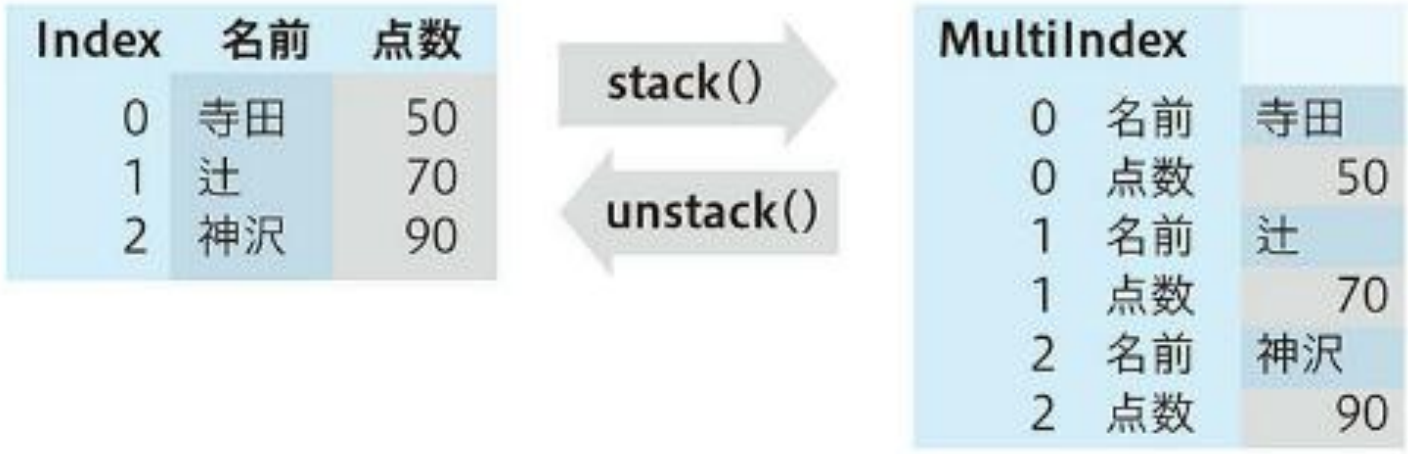


図3-9　スタックとアンスタック

スタックとアンスタック

スタック

- データをスタックするには、DataFrameのstack()メソッドを実行する
- 列名が各要素のインデックスになり、元のインデックスを親階層に持つMultiIndexになる

アンスタック

- データをアンスタックするには、DataFrameのunstack()メソッドを実行する

MultiIndexのスタック・アンスタック

- DataFrameをグループ化し、MultiIndexを持つDataFrameを生成

```
groupby_time = tips.groupby("time")[  # ①time列でグルーピング
    # ①グループ化したオブジェクトから列を選択
    ["total_bill", "tip"]
].agg(
    ("mean", "median")
)  # ①平均値と中央値を算出
groupby_time.columns.names = (
    "value",
    "agg",
)  # ②インデックスに名前を設定
groupby_time
```

value	total_bill		tip	
agg	mean	median	mean	median
time				
Dinner	20.797159	18.390	3.102670	3.00
Lunch	17.168676	15.965	2.728088	2.25

①の処理では、DataFrameを「total_bill」列と「tip」列でグループ化し、平均値と中央値を算出している
②の処理では、MultiIndexの各階層に名前を付けている。

ダミー変数

- ダミー変数とは、カテゴリーデータ（質的変数）をTrue（1）または False（0）の2値に変換した変数
- ダミー変数はカテゴリー数に応じて増加し、pandasから利用する場合は各変数が列に展開される
- 機械学習などのフレームワークやライブラリ（scikit-learnなど）によっては、カテゴリーデータをダミー変数に変換して入力する
- get_dummies()関数は、カテゴリーデータをダミー変数に変換する

表3-3　データの分類

分類		説明
質的変数	質的変数は離散的な値をとり、数値では表現できない変数	
	名義尺度	血液型や性別など、和や差、比率に意味がなく、比較ができない尺度
	順序尺度	震度や企業格付けなど、比較できるが、和や差、比率に意味がない尺度
量的変数	数や量で測れる変数	
	間隔尺度	気温や暦年など、和や差に意味があるが比率には意味がない尺度
	比例尺度	身長や重さなど、和や差、比率に意味がある尺度

	Fri	Sat	Sun	Thur
0	FALSE	FALSE	TRUE	FALSE
1	FALSE	FALSE	TRUE	FALSE
2	FALSE	FALSE	TRUE	FALSE
3	FALSE	FALSE	TRUE	FALSE
4	FALSE	FALSE	TRUE	FALSE
...	...	...	...	...
237	FALSE	TRUE	FALSE	FALSE
238	FALSE	TRUE	FALSE	FALSE
239	FALSE	TRUE	FALSE	FALSE
240	FALSE	TRUE	FALSE	FALSE
241	FALSE	TRUE	FALSE	FALSE
242	FALSE	TRUE	FALSE	FALSE
243	FALSE	FALSE	FALSE	TRUE

要素の展開

- explode()メソッドは、ネストされた（各要素がリストやタプルなどの複数要素を持つ）データを展開して1次元のデータに変換する

	名前	履修科目	得点
0	寺田	[国語, 英語, 数学]	[78, 65, 89]
1	辻	[英語, 物理]	[90, 82]

	名前	履修科目	得点
0	寺田	国語	78
0	寺田	英語	65
0	寺田	数学	89
1	辻	英語	90
1	辻	物理	82

カテゴリーデータの処理

カテゴリーデータの処理

質的変数をcategory型で処理すること

- カテゴリー上は存在するが、データには存在しない値の処理が行える
- 順序尺度の処理（比較、ソートなど）が行える
- グループ化処理が高速化する

尺度水準

データは、質的変数（カテゴリー変数）、量的変数（連続変数）に分類される

表3-3　データの分類

分類	説明	
質的変数	質的変数は離散的な値をとり、数値では表現できない変数	
	名義尺度	血液型や性別など、和や差、比率に意味がなく、比較ができない尺度
	順序尺度	震度や企業格付けなど、比較できるが、和や差、比率に意味がない尺度
量的変数	数や量で測れる変数	
	間隔尺度	気温や暦年など、和や差に意味があるが比率には意味がない尺度
	比例尺度	身長や重さなど、和や差、比率に意味がある尺度

カテゴリーデータの生成

SeriesおよびDataFrameで、引数dtypeに“category”を渡すことで、category型を生成できる

「非常に満足、満足、普通、不満、非常に不満」の5段階の満足度を示す、
category型のSeries「survey_ser」を生成

```
import pandas as pd

survey_data = [
    "満足",
    "普通",
    "普通",
    "非常に不満",
    "不満",
]
survey_ser = pd.Series(survey_data, dtype="category")
survey_ser
```

```
0        満足
1        普通
2        普通
3    非常に不満
4        不満

dtype: category
Categories (4, object): ['不満', '普通', '満足', '非常に不満']
```

カテゴリーデータへの型変換

Seriesをcategory型に変換するには、astype()メソッドの引数に“category”を渡して実行する

```
pd.Series(survey_data).astype("category").dtype
```

```
CategoricalDtype(categories=['不満', '普通', '満足', '非常に不満'], ordered=False)
```

カテゴリーデータの順序付け

survey_serをsort_values()とし、ソートすると、str型と同じ基準でソートされる

```
survey_ser.sort_values()
```

```
4       不満
1       普通
2       普通
0       満足
3    非常に不満
dtype: category
Categories (4, object): ['不満', '普通', '満足', '非常に不満']
```

value_counts()メソッドでカテゴリーごとの度数を集計

```
survey_ser.value_counts()
```

```
普通       2
不満       1
満足       1
非常に不満    1
Name: count, dtype: int64
```

カテゴリーデータの順序付け

- CategoricalDtypeクラスは、あらかじめカテゴリーを設定し、さらに順序付けできるcategory型を定義できる
- 引数categoriesにカテゴリー変数となる一意でかつNULL値を含まないシーケンス型を渡す
- 引数orderedにTrueを渡すことで、カテゴリーを引数categoriesの要素の順序で順序付けが可能となる

```
rating = pd.CategoricalDtype(
    categories=[
        "非常に不満",
        "不満",
        "普通",
        "満足",
        "非常に満足",
    ],
    ordered=True,
)
new_survey_ser = pd.Series(survey_data, dtype=rating)
new_survey_ser
```

```
0       満足
1       普通
2       普通
3    非常に不満
4       不満

dtype: category
Categories (5, object): ['非常に不満' < '不満' < '普通' < '満足' < '非常に満足']
```

カテゴリーデータの順序付け

.cat.ordered属性から、順番が設定されているかを確認できる

```
new_survey_ser.cat.ordered
```

```
True
```

sort_values()メソッドの実行結果から、カテゴリーを設定した順序でソートされたことが確認できる

```
new_survey_ser.sort_values()
```

```
3    非常に不満
4        不満
1        普通
2        普通
0        満足

dtype: category
Categories (5, object): ['非常に不満' < '不満' < '普通' < '満足' < '非常に満足']
```

カテゴリーデータの順序付け

- 順序付けされたカテゴリーデータは、min()メソッドでカテゴリーの先頭を取り出し、max()メソッドでカテゴリーの末尾を取り出せる
- value_counts()メソッドで集計した場合に、データが存在しないカテゴリーは度数が0となる
- describe()メソッドを実行すると、category型に適した記述統計量が算出される
- カテゴリーのインデックス（数値）を取得するには、.cat.codes属性にアクセスする
- 欠損値（NaN）が含まれたカテゴリーは、カテゴリーのインデックス値が-1になる

データの離散化によるカテゴリーデータの生成

- cut()関数、qcut()関数は量的変数を離散化し、質的変数（カテゴリーデータ）に変換する
- 第1引数xには、データとなる配列を渡し、引数binsにはビン（区間のしきい値）の数を定義する
- 返り値はcategory型になり、順序付けされる

```
rng = np.random.default_rng(1)
score = rng.integers(low=0, high=100, size=10)
satisfaction = pd.cut(
    score,
    bins=[0, 20, 40, 60, 80, 101],  # ①
    right=False,  # ②
    labels=[
        "非常に不満",
        "不満",
        "普通",
        "満足",
        "非常に満足",
    ],  # ③
)
survey_df = pd.DataFrame({"satisfaction": satisfaction, "score": score})
survey_df
```

コードでは、cut()関数を利用して、満足度を示す1から100までの数値（「score」列）と、これを5段階に分割したデータ（「satisfaction」列）を列としたDataFrame（survey_df）を生成する

	satisfaction	score
0	普通	47
1	普通	51
2	満足	75
3	非常に満足	95
4	非常に不満	3
5	非常に不満	14
6	非常に満足	82
7	非常に満足	94
8	不満	24
9	不満	31

pandas.qcut関数とpandas. cut関数

- 連続的なデータを離散的なビン（区間）に分割するための2つの重要な関数、qcut と cut がある
  - **qcut関数：**指定した数だけデータを等分する
  - **cut関数：**指定した領域ごとにデータを分割する

特徴	pandas.cut	pandas.qcut
分割基準	等間隔または指定された境界	各ビンに均等なデータ数が含まれるように
ビンのサイズ	各ビンの幅がほぼ等しい	各ビンのデータ数がほぼ等しい
ビンの境界	ユーザーが指定、または自動的に等間隔に決定	データの分布に基づいて自動的に決定
使いどころ	事前に決まったルールでデータを分類したい時	データの分布が偏っている時に均等なサイズのグループを作りたい時

- https://pandas.pydata.org/docs/reference/api/pandas.qcut.html
- https://pandas.pydata.org/docs/reference/api/pandas.cut.html

.cat アクセサ

- Seriesには、アクセサと呼ばれる、各要素に簡潔にアクセスできる仕組みがある。
- .catアクセサは、category型の要素にアクセスするアクセサ
- category型の順序付けを解除するには.cat.as_unordered()メソッドを実行する
- 順序を指定するには.cat.as_ordered()メソッドを実行する

```
unordered_satisfaction = survey_df.loc[:, "satisfaction"].cat.as_unordered()
```

```
unordered_satisfaction.cat.as_ordered()
```

CategoricalIndex

- CategoricalIndexは、CategoricalDtype型のインデックス
- インデックスに重複した要素を持つデータをカテゴリー化することで、重複したインデックスを効率よく処理できる
- 順序付けされたcategory型を利用することで、インデックスからソートした場合にカテゴリーの順序でソートできる

```
new_survey_df = survey_df.set_index("satisfaction")
```

satisfaction	score
普通	47
普通	51
満足	75
非常に満足	95
非常に不満	3
非常に不満	14
非常に満足	82
非常に満足	94
不満	24
不満	31

カテゴリーデータの結合

- 類似した複数のカテゴリーデータを統合する場合がある
- カテゴリーデータを結合するには、union_categoricals()関数を実行する
- 引数にカテゴリーデータを要素にしたリストを渡す
- 順序付けされたカテゴリーデータのすべての要素が合致しない場合は、TypeError例外を送出してエラーになる
- union_categoricals()関数の引数ignore_orderにTrueを渡し、順序付けをしないカテゴリーデータにすることで結合できる

```
ordered_cat1 = pd.Categorical(
    [
        "普通",
        "満足",
        "非常に満足",
    ],
    ordered=True,
)
ordered_cat2 = pd.Categorical(
    [
        "非常に不満",
        "不満",
        "普通",
    ],
    ordered=True,
)
union_categoricals([ordered_cat1, ordered_cat2], ignore_order=True,)
```

```
['普通', '満足', '非常に満足', '非常に不満', '不満', '普通']

Categories (5, object): ['普通', '満足', '非常に満足', '
不満', '非常に不満']
```

データのグループ化

データのグループ化

- データによっては、分類して集計する場合がある
- たとえば、学力テストの点数が学年や学級に分類されているケースが挙げられる
- データをカテゴリーごとにグループ化し、グループ化されたデータを処理する方法について解説する

GroupByオブジェクト

- SeriesおよびDataFrameのgroupby()メソッドを実行すると、SeriesGroupByオブジェクトおよびDataFrameGroupByオブジェクトを生成する

```
import pandas as pd

tips = pd.read_csv(

"https://raw.githubusercontent.com/plotly/datasets/master/tips.csv",
    dtype={
        "sex": "category",
        "smoker": "category",
        "day": "category",
        "time": "category",
    },
)
tips_groupby_time = tips.groupby("time")
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

データの集約

- GroupByオブジェクトから、記述統計量などを算出するメソッドを実行すると、グループごとの算出結果を返す
- 数値データの列だけを集約するには、引数numeric_onlyにTrueを渡す

表3-4　GroupByオブジェクトの代表的な集約メソッド

メソッド	機能概要
mean()	平均
sum()	合計
size()	行数（データ数）
count()	欠損値を除いた度数
nunique()	一意の要素の数
std()	標準偏差
var()	分散

メソッド	機能概要
sem()	標準誤差
describe()	要約統計量
first()	最初の値
last()	最後の値
nth()	n番目の値
min()	最小値
max()	最大値

GroupByオブジェクトのフィルタリング

- GroupByオブジェクトのfilter()メソッドの引数にbool型を返す関数を渡すことで、グループ化したデータに対してフィルタリングを行える

```
over3 = tips.groupby("day").filter(lambda x: x["tip"].mean() > 3)
```

「day」列でグループ化し、この中から「tip」列の平均値が3より大きいデータを抽出

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

GroupByオブジェクトのデータの可視化

- GroupByオブジェクトには、グループ化したデータを可視化するメソッドが用意されている

```
tips.groupby("time").boxplot();
```

tipsを「time」列でグループ化し、GroupByオブジェクトのboxplot()メソッドを実行して箱ひげ図を描画している