

データ加工

《学修項目》

- ◎集計処理、四則演算処理
- ◎ソート処理、サンプリング処理
- ◎クレンジング処理（外れ値、異常値、欠損値）
- 結合処理（内部結合、外部結合）
- データ型変換処理
- データの標準化、ダミー変数

《キーワード》

量的データ、質的データ、単純集計、クロス集計、ソート処理とマージ処理、統合処理とクエリ、内部結合と外部結合、データクレンジング、データクリーニング、欠損値処理、外れ値処理、異常値処理、ノイズの除去、データの型変換、スケーリング（正規化、標準化）

《参考文献、参考書籍》

- [1] 東京大学MIセンター公開教材 「2-5 データ加工」 《利用条件CC BY-NC-SA》
- [2] データサイエンスのためのデータベース（データサイエンス入門シリーズ）（講談社）
- [3] 数理・データサイエンス・AI公開講座（放送大学）

1. データの集計

あらまし

- データベースにおける用語
- データの種類（量的データ、質的データ）
- 単純集計
- クロス集計
- ソート処理とマージ処理

1.1 データ加工（data processing）とは

データ加工 (data processing) とは

- 与えられたデータ集合を目的に適った便利な形に変換することです
- 与えられたデータ集合は、そのままでは扱いにくいことが多いです
 - 例えば、Webから表データを収集する際、HTML形式のWebページには、表中の値以外の記号が含まれていて、表データを解析しにくいです
- 単純な統計処理を含みます
 - 統計情報は「便利な形」の1つです

1.2 データベースにおける用語

- テーブル（表）、カラム（列）、レコード（行） RDBではタプル、フィールド値（セル値）※（カッコ）はExcelでの呼び方

Python pandasライブラリで行単位でデータを作成してみる

例) 9011から9016までの6名の学生の性別とクラス名、3科目のテストの点数（整数で100点満点）のテーブル

```
In [1]: ## Python pandasライブラリで列単位でデータを作成してカラム・レコード・フィールド値単位で見てみる
## 参考:https://pystyle.info/pandas-table-styling-cheatsheet/
import numpy as np
import pandas as pd
from IPython.display import display, HTML

# CSVデータなどから read_csv して dataへ読み込んでも良い。
data = [[["Male", "A", 84, 70, 96], ["Male", "B", 79, 79, 79], ["Female", "A", 99, 76, 98], ["Female", "C", 56, 60, 70], ["Female", "D", 65, 68, 72], ["Male", "E", 75, 70, 85], ["Female", "F", 60, 65, 70], ["Male", "G", 80, 75, 88], ["Female", "H", 62, 68, 75], ["Male", "I", 78, 72, 85], ["Female", "J", 64, 66, 72], ["Male", "K", 82, 78, 90], ["Female", "L", 66, 70, 78], ["Male", "M", 76, 72, 82], ["Female", "N", 68, 74, 80], ["Male", "O", 85, 80, 92], ["Female", "P", 70, 75, 85], ["Male", "Q", 72, 68, 80], ["Female", "R", 65, 70, 75], ["Male", "S", 88, 82, 95], ["Female", "T", 74, 78, 88], ["Male", "U", 80, 76, 84], ["Female", "V", 72, 74, 80], ["Male", "W", 86, 84, 93], ["Female", "X", 76, 80, 86], ["Male", "Y", 82, 78, 88], ["Female", "Z", 74, 76, 82]], [{"label": "Total", "value": 100}], [{"label": "Average", "value": 75}], [{"label": "Median", "value": 75}], [{"label": "Min", "value": 60}], [{"label": "Max", "value": 95}]]
```

```
In [2]: # English カラム(列) を緑色にする
css_bg_b = {"background-color": "green"}
display(df.style.set_properties(subset="English", **css_bg_b))
```

	gender	class	Math	English	Physics	Row_Total
9011	Male	A	84	70	96	250
9012	Male	B	79	79	79	237
9013	Female	A	89	92	86	267
9014	Male	A	99	76	98	273
9015	Female	C	56	60	70	186
9016	Female	C	46	44	47	137

```
In [3]: # id "9013"番のレコード(行)をオレンジ色にする
css_bg_o = {"background-color": "orange"}
display(df.style.set_properties(subset=pd.IndexSlice["9013", :], **css_bg_o))
```

	gender	class	Math	English	Physics	Row_Total
9011	Male	A	84	70	96	250
9012	Male	B	79	79	79	237
9013	Female	A	89	92	86	267
9014	Male	A	99	76	98	273
9015	Female	C	56	60	70	186
9016	Female	C	46	44	47	137

```
In [4]: # 数値に応じたカラースケール(ヒートマップ)で表示する
df.style.background_gradient()
```

	gender	class	Math	English	Physics	Row_Total
9011	Male	A	84	70	96	250
9012	Male	B	79	79	79	237
9013	Female	A	89	92	86	267
9014	Male	A	99	76	98	273
9015	Female	C	56	60	70	186
9016	Female	C	46	44	47	137

データの種類

- 量的データ：数字など（四則演算できる）例）各科目的「点数」（これの合計を求めることができた）
- 質的データ：文字列など（四則演算できない）例）学生番号、性別、クラス名（識別子として、別々のものとして認識することができる）

整然データ (tidy data)

- 1行が1つの観測（データ点）に整形された表形式のことです
 - 全ての行が同一の型（定数個の属性の組）となります
 - 整然データでないものを雑然データ（messy data）と呼びます
- 例：整然データ

地域	時間帯	気温°C
東京	午前	10
東京	午後	12
大阪	午前	13
大阪	午後	16

- 例：雑然データ

地域	気温°C (午前)	気温°C (午後)
東京	10	12
大阪	13	16

整然データの利点

- 一見すると雑然データの方がコンパクトで見やすく思えます
 - 実際、前頁の例において整然データの方がスペースを多く割いています
- 整然データはフィルタやソートと併用する際に扱いやすいです
- 前頁の例について言えば
 - 同一地域や同一時間帯の観測に限定した表がフィルタで手に入ります
 - 最も気温の高い（低い）地域と時間帯が単純なソートで手に入ります
 - 同一時間帯の平均気温と同一地域の平均気温が殆ど同じ式で計算できます
 - 時間帯を追加したり細分化する際に、表の形を変えなくて済みます
- 整然データは計算機で処理するための形式です

1.3 集計処理

- 単純集計
 - データ件数の数え上げ
- 算術演算、統計演算と組み合わせる
 - 四則演算や統計量
 - 条件付き集計
- クロス集計

- 複数項目の比較

ソート処理とマージ処理

- データの並び替え
- データの統合

1.4 集計処理の例（単純集計）

集計（カウント）

- 単純な集計処理
 - データの件数を計算：項目ごとでデータ集計
 - コンピュータでの実装：ExcelやPython(pandas)

⇒ 一つの項目（属性）の性質や、ばらつきを見るのに適している

例題：学生テーブルを作ってみる

- 入力：学生番号、性別、クラス名の一覧表
- 出力：「性別」「クラス」項目別で集計

```
In [5]: # 学生番号 : [性別, クラス名] の一覧表を得る(列名マッチで抽出)
df[["gender", "class"]]
```

```
Out[5]:      gender  class
9011     Male      A
9012     Male      B
9013   Female      A
9014     Male      A
9015   Female      C
9016   Female      C
```

```
In [6]: # [性別, クラス名] の一覧 から、性別毎の度数を得る(gender で groupby.count() を使う)
df[["gender", "class"]].groupby('gender').count()
```

```
Out[6]:      class
gender
Female      3
Male       3
```

```
In [7]: # [性別, クラス名] の一覧 から、クラス毎の度数を得る(class で groupby.count() を使う)
df[["gender", "class"]].groupby('class').count()
```

```
Out[7]: gender
```

class	
A	3
B	1
C	2

1.5 集計処理の例（演算処理）

- 算術演算
 - 四則演算などを組み合わせる
- 度数分布表
 - データを階層別に集計する
 - ヒストグラムを作成し可視化する
- 基本統計量の集計
 - 最大値、最小値、平均値などを求める
 - 箱ひげ図を作成し可視化する

例題：各科目の点数について、度数分布表を作ってみる

- 入力：各科目の点数
- 出力：Math, English, Physicsの度数分布

```
In [8]: # 学生番号 : [Math, English, Physics] の一覧表を得る(列名マッチで抽出)
df[["Math", "English", "Physics"]]
```

```
Out[8]: Math  English  Physics
```

9011	84	70	96
9012	79	79	79
9013	89	92	86
9014	99	76	98
9015	56	60	70
9016	46	44	47

```
In [9]: # 科目名 列名マッチで抽出後, [60, 100]を4区間に分割した度数分布を value_count で求める
df[["Math"]].value_counts(bins=np.linspace(60, 100, 5), sort=False)
## df[["English"]].value_counts(bins=np.linspace(60, 100, 5), sort=False)
## df[["Physics"]].value_counts(bins=np.linspace(60, 100, 5), sort=False)
```

```
Out[9]: (59.999, 70.0]      0
(70.0, 80.0]              1
(80.0, 90.0]              2
(90.0, 100.0]             1
Name: Math, dtype: int64
```

1.6 クロス集計

- 項目を2つ以上かけ合わせて集計する手法
 - 項目（属性）ごとの違いを見る
 - 分割表ともいう
 - 表側（目的変数）と表頭（説明変数）

例) アイスクリームの売上分析の場合、目的変数：アイスクリームの種類、説明変数：その日の気温や天気などとなる。

- 統計量を組み合わせる
 - 数学の平均をクロス集計する：データの精度が問題になる
 - 各科目の平均をクロス集計する：科目毎の値のばらつきが問題になる

例題：男女別、クラス別で各科目の平均点のクロス分析を行ってみる

- 入力：学生番号、性別、クラス名、各科目の平均点の一覧表
- 出力：男女別、クラス別の平均点（クロス分析）

```
In [10]: # 3科目の点数の平均値を新たな列 Row_Average として追加する
df.loc[:, 'Row_Average'] = df[['Math', "English", "Physics"]].sum(axis=1) / 3

# インタラクティブ・テーブル機能を使って、ソートなど簡単な分析は可能だが、クロス分析は難しい
df
```

```
Out[10]:   gender  class  Math  English  Physics  Row_Total  Row_Average
  9011    Male     A     84       70       96        250    83.333333
  9012    Male     B     79       79       79        237    79.000000
  9013  Female     A     89       92       86        267    89.000000
  9014    Male     A     99       76       98        273    91.000000
  9015  Female     C     56       60       70        186    62.000000
  9016  Female     C     46       44       47        137    45.666667
```

```
In [11]: # crosstabを使って、性別 vs クラス のクロス集計(人数)を行う
pd.crosstab(index=df['gender'], columns=df['class'])
```

```
Out[11]:   class  A  B  C
  gender
  Female  1  0  2
  Male    2  1  0
```

```
In [12]: # crosstabを使って、性別 vs クラス の 学生平均点のクロス集計(点数)を行う(欠損値はNaNである)
pd.crosstab(index=df['gender'], columns=df['class'], values=df['Row_Average'],
```

```
Out[12]:   class      A      B      C
  gender
  Female  89.000000  NaN  53.833333
  Male   87.166667  79.0  NaN
```

```
In [13]: # pivot_tableを使って、[性別、クラス]別の 学生平均点と合計のクロス集計(点数)を行う(欠損値の  
# デフォルト(引数aggfuncを省略した場合)はnumpy.mean()が指定される  
pd.pivot_table(df[["gender", "class", "Math", "English", "Physics", "Row_Tot
```

Out[13]:

gender	class	English	Math	Physics	Row_Total
Female	A	92	89.0	86.0	267.0
	C	52	51.0	58.5	161.5
Male	A	73	91.5	97.0	261.5
	B	79	79.0	79.0	237.0

1.7 データの種類

構造化データ

- 分析できる形に構造化されたデータ
- 表形式で整理されたデータセット

→量的データ、質的データに分類される

非構造化データ

- まだ構造化されていないデータ

→画像、音声、文章データなど *データ表現の章を参照

1.8 データのソート

- 順序付けの基準（オーダ）
 - 量的変数の場合：科目的得点など（間隔尺度（例：西暦年など）、数値尺度（例：重量など）の別がある）
 - 質的変数の場合：名義尺度（学生名など）、順序尺度（学籍番号など、順序が存在し、数量によるソートが可能）

昇順と逆順

- 数値の大小で順序付ける：質的変数も、順序尺度であれば可能
- 名義尺度も あいうえお順、ABC順などで順序付け (aab < ac < adbc など文字列にもオーダがつけられる)

1.9 ソート処理の例

大きさを指標として並び替える

- 数量データや順序データ
- 文字列のソート：漢字データは、ひらがな などを付与してオーダをつける（読み方の違いは漢字コード列の情報だけでは不充分）

1.10 名寄せ

名寄せ

- 概念や対象に対する表記を統一することです
 - 例えば「東京」と「東京都」は表記は別ですが同一地域です
 - また「學問のすゝめ」と「学問のすすめ」は同一著作物です
 - 一方、同姓同名の別人には適宜名前を付けて統一的に区別します
- 元々は、複数の銀行口座に渡って同一顧客を追跡して一元管理することを意味していましたが、今ではより広い意味で用いられます
- 適切に名寄せをしないと、加工結果から得る情報が不正確になります
- 表記の同一性に基づいて処理できないデータ形式は、機械的な処理に不向きなので、名寄せは実用上重要です
 - データベースにおける主キーの付与にはほぼ対応します
- 異なるデータセットを統合するデータマッピングにおいて、名寄せは中心的なタスクです
 - データマッピングツールは名寄せを支援します

日本語の氏名には、ふりがなが付いていないと、データ重複、名寄せ処理の妨げになる【重要】

例) 漢字標記で同一でも、同姓同名とは限らない。漢字表記が違っても、よみがなが同一のことがある（紛らわしい）

吉川五郎 ⇒ きっかわ/ごろう or よしかわ/ごろう
小山優子 or 尾山祐子 ⇒ おやま/ゆうこ

名寄せ処理時には、生年月日、現住所、電話番号などで同一か否かの判定をすることになるが、完全とは言えない。

2. データの統合処理

あらまし

- 統合処理とクエリ
- データの結合処理
- 内部結合と外部結合

2.1 データ統合とは？

統合処理とクエリ

- クエリ=データベース管理システムに対する問い合わせ、要求
 - SQLで実装する
- プログラムではデータを検索し、変換・整形して出力する。SQLクエリを内部の実装として含むこともある
 - 統合処理の条件指定などを行う必要がある

2.2 データの結合処理

- 複数のファイルやデータを（何らかの基準、キーで）結合する処理
- データベースでは、内部結合と外部結合の別がある
- ExcelやPython(pandas)で実装

2.3 データベースの結合例

- 複数のデータ集合の結合
 - 学籍簿と成績表の結合など
 - 2つのデータの集合を結ぶ「キー」が必要
- データベースの結合処理
 - 内部結合と外部結合の別、外部キーが必要
 - Union, Join といった処理が提供されている。
- 一般的なデータの結合処理
 - Excelの場合：データツール > 統合
 - Python pandasライブラリの場合：join

2.4 内部結合と外部結合

データベース（表）の内部結合

- 結合条件に指定している値が両方のテーブルに存在するデータを抽出する結合
- 2つのテーブル間で、共通（結合条件の項目の値が一致）するレコードのみを抽出

データベース（表）の外部結合

- 基準となるテーブルに存在すれば抽出する結合。どちらのテーブルを基準にするかを指定し、その基準となるテーブルに存在するデータを抽出、基準ではないテーブルからは抽出できるデータのみ取得する。
- 左外部結合（Left Outer Join）、右外部結合（Right Outer Join）の別がある
- 完全外部結合は、全ての要素を入れる

参考：https://ai-inter1.com/pandas-dataframe_join/

2.5 データベース（表）の内部結合の例

- 両方のテーブルに存在するデータを抽出して結合する
- 2つのテーブル間で、結合条件のフィールド値が一致するレコードのみを出力とする

Python pandasライブラリで2つのテーブルを作ってみる

- 学生テーブル：9011から9017までの7名の学生の性別とクラスIDの対応表（欠損値あり）
- クラス名テーブル：クラスIDとクラス名の対応表（当該クラスに学生が存在しないケースあり）

```
In [14]: # 学生テーブル:9011から9017までの7名の学生の性別とクラスIDの対応表("9017"はクラス対応が無い)
std_data = [[ "9011", "Male", "100"], [ "9012", "Male", "200"], [ "9013", "Female", "100"], [ "9014", "Male", "200"], [ "9015", "Female", "100"], [ "9016", "Female", "100"], [ "9017", "Male", np.nan]]
std_df = pd.DataFrame(std_data,
                      columns=["student_id", "gender", "class_id"])
std_df
```

```
Out[14]:   student_id  gender  class_id
0          9011    Male     100
1          9012    Male     200
2          9013  Female     100
3          9014    Male     200
4          9015  Female     100
5          9016  Female     100
6          9017    Male      NaN
```

```
In [15]: # クラス名テーブル:クラスIDとクラス名の対応表
class_data = [[ "100", "Science"], [ "200", "Humanities"], [ "300", "Comprehensive"]]
class_df = pd.DataFrame(class_data,
                        columns=["class_id", "class_name"])
class_df
```

```
Out[15]:   class_id    class_name
0          100      Science
1          200  Humanities
2          300  Comprehensive
```

```
In [16]: # 内部結合(Inner Join)
# merge, innerを用いて、2つのDataFrameを "class_id" をキーとして内部結合する(表示は学生)
pd.merge(std_df, class_df, how="inner", on="class_id").sort_values('student_id')

# 学生"9017"はクラス割当てが欠損しているので、内部結合結果のレコードとして出力されないように注意せ
```

```
Out[16]:   student_id  gender  class_id    class_name
0          9011    Male     100      Science
1          9012    Male     200  Humanities
2          9013  Female     100      Science
3          9014    Male     200  Humanities
4          9015  Female     100      Science
5          9016  Female     100      Science
```

2.6 データベース（表）の外部結合の例

- 一方のテーブルについて全レコードを抽出し、もう一方のテーブルについて結合条件のフィールド値と一致するデータのみ抽出
- 全抽出するテーブルの位置により、左外部結合（Left Outer Join）、右外部結合（Right Outer Join）が存在する
- 完全外部結合は、全ての要素を入れる

```
In [17]: # 左外部結合(Left Outer Join)
# merge, leftを用いて、2つのDataFrameを "class_id" をキーとして左外部結合する
pd.merge(std_df, class_df, how="left", on="class_id").sort_values('student_i
# 学生"9017"はクラス割当てが欠損しているが、学生テーブル(左側)にはデータが存在するので、結合結果
```

```
Out[17]:   student_id  gender  class_id  class_name
0         9011    Male      100    Science
1         9012    Male      200  Humanities
2         9013  Female      100    Science
3         9014    Male      200  Humanities
4         9015  Female      100    Science
5         9016  Female      100    Science
6         9017    Male       NaN       NaN
```

```
In [18]: # 右外部結合(Right Outer Join)
# merge, rightを用いて、2つのDataFrameを "class_id" をキーとして右外部結合する
pd.merge(std_df, class_df, how="right", on="class_id").sort_values('student_
# 学生テーブルのレコードとして300番のクラス割当した学生は存在しないが、クラス名テーブル(右側)には300
```

```
Out[18]:   student_id  gender  class_id  class_name
0         9011    Male      100    Science
1         9012    Male      200  Humanities
2         9013  Female      100    Science
3         9014    Male      200  Humanities
4         9015  Female      100    Science
5         9016  Female      100    Science
6          NaN     NaN      300  Comprehensive
```

```
In [19]: # 完全外部結合(Full Outer Join)
# merge, outerを用いて、2つのDataFrameを "class_id" をキーとして完全外部結合する
pd.merge(std_df, class_df, how="outer", on="class_id").sort_values('student_
# "9017" 学生のレコードのクラス値、300番クラスの学生値は欠損したままであるので注意
```

Out[19]:

	student_id	gender	class_id	class_name
0	9011	Male	100	Science
1	9012	Male	200	Humanities
2	9013	Female	100	Science
3	9014	Male	200	Humanities
4	9015	Female	100	Science
5	9016	Female	100	Science
6	9017	Male	NaN	NaN
7	NaN	NaN	300	Comprehensive

3. データの前処理とデータ変換

3.1 データの前処理

データクレンジング、データクリーニング

- 破損したデータ、不正確なデータ、無関係のデータなどの処理
- 住所や名簿などをデータ分析可能な状態にすること

主なデータ前処理

- 欠損値処理
- 外れ値処理、異常値処理
- ノイズの除去

データの変換

- 前処理としてのデータの変換
- 型変換やスケーリング
- 正規化、標準化

3.2 欠損値処理

データクレンジング、データクリーニング

- 欠損値の前処理：どんな値で埋めるのか？
 - 一般的には、数値データは平均値で置き換える。質的データは最頻値で置き換える（検討が必要）

ノイズの除去

- センサ値などでノイズが乗った形で出現する
- 正常値と置き換えるなどの処理が考えられる（検討が必要）
 - 例）売上の時系列データの場合、ノイズが生じた場合には過去の同じような時期のデータと比較するなど

データクレンジング

- 汚れたデータを特定して除去・訂正することです
 - 汚れの例：破損したデータ，不正なデータ，例外的なデータ
 - 統計処理の観点では，外れ値・異常値・欠測値への対処が主です
- ◆ 外れ値：他の測定値から大きく外れた値
 - 観測上の異常です
 - 一般に原因は分かりません
- ◆ 異常値：外れ値の中で異常さの原因が解明されているもの
 - 本質的な異常です
 - 誤りや故障によって，異常であるべくして異常になったものです
- ◆ 欠測値：測定データに含まれていない，本来測定されているべき値
 - 例えば，アンケートの「無回答」や記入漏れなどで生じます

3.3 欠損値の扱い

欠損値を処理する一般的な方法

- 削除：列や行ごと削除する
- 平均置換：平均値で置き換える（当該レコードの信頼性が毀損する）
- 最頻値で置き換え：質的データに有効
- ダミー置換：ダミー変数（Unknown, Zeroなど）で置き換える

完全データ：全ての値が観測されているデータ 不完全データ：欠損が含まれているデータ

頻繁に欠損値が出現するが，量的データのモデルが既知の場合には，欠損データを予測可能な回帰モデルを作成し，逐次代入する方法がとられる（統計的数値補完）

- 参考：note.nkmk.me | pandasで欠損値NaNを除外（削除）・置換（穴埋め）・抽出

欠測値への対処

- 欠測値は統計処理を不可能にします
 - 欠測値が入ると算術平均すら定義されません
→ 算術平均に基づく統計量が全て定義されません
 - 欠測値への典型的な対処法が2つあります
- ◆ 欠測値を含むケースを丸々除去する
 - つまり、欠測したものは存在しないものとして扱う
- ◆ 欠測値に非欠測ケースの平均値を代入する
 - つまり、欠測値は平均値に対して無害な値であると仮定する
- これらは統計処理を可能にしますが、統計量（統計モデル）に対してバイアスを生みます

3.4 外れ値処理、異常値処理

- 外れ値：通常とはかけ離れた値
 - 異常の大きい、小さいなど
- 外れ値処理
 - 一般的には「残す」

+異常値：存在し得ない値など（例：非常に小規模な図書館の入場者が100万人であることを示すデータ、など） *欠損値と同じ扱いとする *個別に吟味し、置き換えや削除を検討する

一般的に、データ件数、レコード数が非常に大きくなると、欠損値・外れ値・異常値を含む可能性が高まる

外れ値の検定

- 外れ値を特定する検定には様々な方法があります
- 単純かつ代表的な検定は次の2つです
 - ◆ 第1と第3四分位数の区間から外れる
 - つまり上位25%と下位25%を外れ値と見做します
 - ◆ 平均からの乖離が 2σ ～ 3σ に達する
 - σ は標準偏差を意味します

3.5 前処理：データ変換

データ変換処理の必要性

- 生データは「汚れている」「扱いにくい」
- データの型変換やスケーリングといった前処理が必要

データの型変換

- データ型：プログラミング言語における型変換、整数型、小数点型など
- データサイエンスにおけるデータ変換（スケーリング：正規化、標準化）

プログラミング言語によっては、変数データの型変換コマンドが実装されている

データ型変換

- 概念に対して特定の値を関連付けることがあります
 - 例えば、日時の表記法は様々ありますが、計算機ではUNIX時間で統一的に時刻を表現することが一般的です
 - UNIX時間：1970-01-01 00:00:00 からの経過秒数
- 文字列型のデータを所定のデータ型に変換する必要があります
 - UNIX時間の場合は符号なし整数への変換です
- 地理オブジェクト名に対して、地理座標（典型的には緯度経度）を付与するジオコーディングも、データ型変換の一種です
 - 文字列の名前から、地理座標を伴った文字列への変換です

3.6 データ変換処理

データサイエンスにおけるデータ変換

- 量的データの変換
- 質的データから量的データへの変換（オーダ、ラベリング）

量的データの変換

- スケーリング（尺度）の変更、対数化など
- 正規化、標準化

質的データの変換

- 質的データから量的データへの変換（順序尺度：例 成績が1番、2番、3番 であった場合、1番の方が3番より成績が良いことは分かるが、1番と3番を「足した」結果は意味をもたない）
- 量的データへの割当て（ラベリング、名義尺度：例 良い=1, 普通=2, 悪い=3）

ダミー変数

- 回帰分析では、独立変数Xと従属変数Yとの間にある定量的な関係を統計的に推定します
 - このときYは連続値です
- Xが離散的な値を取るとき、そのままではXに関する関数（例えば線形変換）としてYを定義できません
 - 例えば、Xが曜日を表す場合、数値として演算可能ではありません
- Xの定義域 $\text{dom}(X)$ が有限なら、 $x \in \text{dom}(X)$ は2値変数の組 x_1, \dots, x_k で表現できます
 - $x = i \Leftrightarrow x_i = 1 \wedge x_j = 0$ (ただし $j \neq i$ かつ $k = |\text{dom}(X)|$) と定義します
 - この2値変数 x_1, \dots, x_k を、それぞれ独立変数 X_1, \dots, X_k と見做すことで、関数の形でYが定義でき、重回帰分析が適用できるようになります
 - この X_1, \dots, X_k をダミー変数と呼びます
 - ダミー変数の導入は離散データに対する一種の正規化です

3.7 スケーリング：正規化、標準化

複数のデータを比較するためにデータの形をそろえる

- それぞれの指標に基づいてスケーリングする（尺度を統一する）
- 平均、標準偏差、最大、最小といった統計値を利用する

正規化：最大と最小を利用する

- Min-Max 正規化法
- 最小を0、最大を1でスケーリング

標準化：平均と標準偏差を利用する

- Z-score 標準化法
- 平均0、分散1でスケーリング

参考：[scikit-learn.preprocessing, StandardScalerで標準化してみる](#)

データの正規化

- 統計量を扱いやすくする幾つかの正規化があります
 - 分布の性質を保存しつつ、データ固有の大きさを捨象します
 - 以下では、 X' をデータセット X を正規化したものとします
- ◆ Min-max正規化：最小値が0で最大値が1になるよう変換する
 - min-max正規化は $X' = \left\{ \frac{x - \min(X)}{\max(X) - \min(X)} \mid \forall x \in X \right\}$ で構成できます
- ◆ 標準化：平均が0で分散が1になるよう変換する
 - 標準化は $X' = \left\{ \frac{x - \text{avg}(X)}{\sigma} \mid \forall x \in X \right\}$ で構成できます
 - ただし、 $\text{avg}(X)$ は X の平均値で、 σ は標準偏差です
- 異なるデータの統計量を合算する機械学習において特に有用です

3.8 データの標準化と正規化の例

正規化の例

- 最大値と最小値の差を求める
- 最小値からの差を求める
- 正規化値を求める（1.で割る）

標準化の例

- 点数を標準化
- 平均値と標準偏差を求める
- 平均からの差を標準偏差で割る

参考：[scikit-learn数値系特徴量の前処理まとめ\(Feature Scaling\)](#)

```
In [20]: ## 上で例にあげた 6名分の3科目の試験点数のデータを標準化,正規化してみる
import numpy as np
import pandas as pd

from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler # scikit-learn

## データを準備して data へ格納する
data = [[84, 70, 96], [79, 79, 79], [89, 92, 86],
[99, 76, 98], [56, 60, 70], [46, 44, 47]]
```

```
In [21]: ## 正規化ライブラリ MinMaxScaler を使って正規化する
scaler = MinMaxScaler()

scaler.fit(data) # 正規化処理を実行

print(scaler.data_max_) # 各科目的最大値
print(scaler.data_range_) # 各科目の範囲
```

```

print(scaler.transform(data)) # 各学生の各科目の点数の,正規化後の位置(最小0, 最大1の

print(scaler.transform([[75, 75, 75]])) # 点数が全て75点である別の学生がいた場合,i

[99. 92. 98.]
[53. 48. 51.]
[[0.71698113 0.54166667 0.96078431]
 [0.62264151 0.72916667 0.62745098]
 [0.81132075 1. 0.76470588]
 [1. 0.66666667 1. ]
 [0.18867925 0.33333333 0.45098039]
 [0. 0. 0. ]
 [[0.54716981 0.64583333 0.54901961]]
```

In [22]:

```

## 標準化ライブラリ StandardScaler を使って標準化する
scaler = StandardScaler()

scaler.fit(data) # 標準化処理を実行

print(scaler.mean_) # 各科目の平均値
print(np.sqrt(scaler.var_)) # 各科目の標準偏差値

print(scaler.transform(data)) # 各学生の各科目の点数の,標準化後の位置(平均0, 分散1の

print(scaler.transform([[75, 75, 75]])) # 点数が全て75点である別の学生がいた場合,i

[75.5 70.16666667 79.33333333]
[18.57193223 15.14834058 17.33653817]
[[ 0.4576799 -0.01100231 0.96136071]
 [ 0.18845643 0.58312218 -0.01922721]
 [ 0.72690336 1.44130198 0.38454428]
 [ 1.2653503 0.38508068 1.076724 ]
 [-1.04997152 -0.67114062 -0.538362 ]
 [-1.58841846 -1.72736192 -1.86503978]]
[[-0.02692235 0.31906685 -0.24995379]]
```

4. Python pandasによるデータクレンジングと分析の例

(重要：欠損値, 外れ値ありに加工した) 気象データを用いた
処理例 (Python pandas)

4.1 欠損値のあるレコードの検出, 欠損値の線形補間法による 穴埋め (使用する際は要注意)

In []:

```

# オリジナルのCSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする。
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DE/main/5/
```

```

# 欠損値,外れ値あり加工済CSVファイルもダウンロードする
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DE/main/5/
```

```

# wgetしなくても,Google colab.の左メニュー [ファイル] アイコンをクリックして,ブラウザへファイルをト
```

```

# ファイル (JMA_Nagano_2021_xxxxxxx.csv) がダウンロード・配置できたことを確認する
##!ls -al ./
```

```
# オリジナルと加工済ファイルの相違点を diff で確認しておく  
!diff JMA_Nagano_2021_original.csv JMA_Nagano_2021_error.csv
```

```
In [24]: # 欠損値,外れ値あり加工済CSVファイルをpandasで読み込んでデータフレームdfに格納  
import pandas as pd  
from IPython.display import display  
  
df = pd.read_csv('JMA_Nagano_2021_error.csv')  
df.columns = ['date', 'temp_ave', 'temp_high', 'temp_low', 'rain_total', 'sun_shi  
  
# インタラクティブ表示にして, [Filter] > [Search by all fields:] に NaN で検索すると,欠損  
# しかし,外れ値(異常値)がどこに潜んでいるかは,わかりませんね!  
display(df)
```

	date	temp_ave	temp_high	temp_low	rain_total	sun_shine	wind_ave	wind_m
0	2021/1/1	-1.9	0.9	-3.5	5.5	1.4	1.2	1.1
1	2021/1/2	-2.3	1.3	-4.5	0.0	0.1	1.7	1.1
2	2021/1/3	-2.5	0.7	-5.5	0.0	1.6	1.3	1.1
3	2021/1/4	-1.4	3.5	-5.3	0.0	4.4	1.1	1.1
4	2021/1/5	-0.6	2.7	-3.0	0.0	1.5	1.0	1.1
...
360	2021/12/27	-3.3	-0.3	-6.2	0.0	1.6	1.1	1.1
361	2021/12/28	-2.0	1.7	-5.3	2.0	5.9	2.3	1.1
362	2021/12/29	-1.3	5.8	-8.0	0.0	8.2	1.3	1.1
363	2021/12/30	0.0	2.0	-1.9	7.0	0.2	1.4	1.1
364	2021/12/31	-3.3	-0.5	-5.3	4.5	0.0	2.5	1.1

365 rows × 17 columns

```
In [25]: # 行・列ごとに欠損値の個数をカウント  
print(df.isnull().sum())
```

```
date          0
temp_ave      1
temp_high     1
temp_low      1
rain_total    1
sun_shine     1
wind_ave      0
wind_max      0
wind_dir      0
wind_peak     0
wind_dir2     0
wind_dir16    0
humid_ave     0
humid_max     0
humid_ave2    0
forecast_day   0
forecast_night 0
dtype: int64
```

```
In [26]: # 欠損値NaNが一つでも含まれる行を抽出
print(df[df.isnull().any(axis=1)])
```

	date	temp_ave	temp_high	temp_low	rain_total	sun_shine	\
44	2021/2/14	NaN	NaN	NaN	0.0	9.4	
73	2021/3/15	5.9	12.2	0.5	NaN	8.3	
357	2021/12/24	2.7	8.6	-2.7	0.0	NaN	
	wind_ave	wind_max	wind_dir	wind_peak	wind_dir2	wind_dir16	humid_ave
44	2.1	6.5	西南西	10.3	南西	西南西	
6.5							
73	2.5	7.8	東	11.5	東	東北東	
5.8							
357	1.3	2.8	南西	4.2	東北東	東	
5.6							
	humid_max	humid_ave2	forecast_day	forecast_night			
44	29	64	晴後一時曇	曇時々晴後一時雨			
73	44	64	晴一時曇	晴後時々曇			
357	49	77	晴時々曇	曇後雨			

```
In [27]: # 欠損値NaNを前後の値から補間するinterpolateを用いる例(欠損値1つのみ.線形補間法による.case-
df_inter = df.interpolate(method='linear', limit=1, limit_direction='both')

# さきほどメモした欠損値を含むレコードの場所で,補完されたデータの値を確かめよ.補完値は妥当か? 線形
df_inter
```

Out[27]:

	date	temp_ave	temp_high	temp_low	rain_total	sun_shine	wind_ave	wind_max
0	2021/1/1	-1.9	0.9	-3.5	5.5	1.4	1.2	1.2
1	2021/1/2	-2.3	1.3	-4.5	0.0	0.1	1.7	1.7
2	2021/1/3	-2.5	0.7	-5.5	0.0	1.6	1.3	1.3
3	2021/1/4	-1.4	3.5	-5.3	0.0	4.4	1.1	1.1
4	2021/1/5	-0.6	2.7	-3.0	0.0	1.5	1.0	1.0
...
360	2021/12/27	-3.3	-0.3	-6.2	0.0	1.6	1.1	1.1
361	2021/12/28	-2.0	1.7	-5.3	2.0	5.9	2.3	2.3
362	2021/12/29	-1.3	5.8	-8.0	0.0	8.2	1.3	1.3
363	2021/12/30	0.0	2.0	-1.9	7.0	0.2	1.4	1.4
364	2021/12/31	-3.3	-0.5	-5.3	4.5	0.0	2.5	2.5

365 rows × 17 columns

4.2 「統計ベース」による外れ値検知を試す

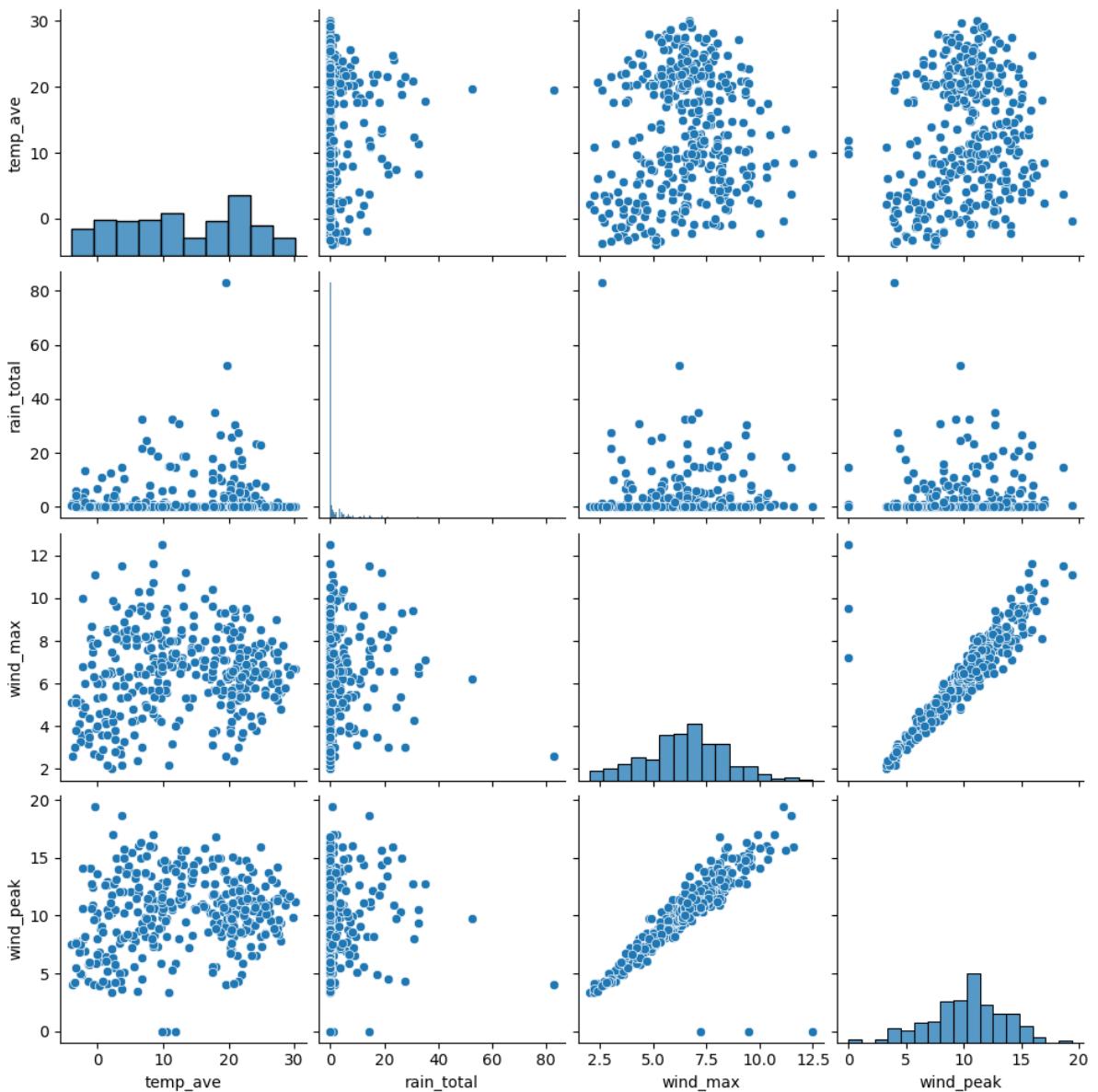
機械学習を取り入れたAIシステムの構築は、
 ①データセット作成（前処理）→②モデルの構築→③モデルの適用
 というプロセスで行っています。その際「データセット作成（前処理）」の段階では、正しくモデル構築できるよう、事前にデータを整備しておくことが求められます。その際に問題なることが多い「外れ値」とその対処方法について解説していきます。

参考:Qiita 外れ値検出

In []: !pip install seaborn # seabornライブラリをインストール

In [31]: # Python, pandas, seabornでペアプロット図(散布図行列)を作成してみる
`import pandas as pd`
`import seaborn as sns`
`%matplotlib inline`
`sns.pairplot(df_inter.loc[:, ['temp_ave','rain_total','wind_max','wind_peak']]`

Out[31]: <seaborn.axisgrid.PairGrid at 0x14599edd0>



wind_max vs wind_peakで、wind_peakがゼロに貼り付いている点が3箇所見つかったので、このペアで更に詳細に見てみる

参考: Seaborn で散布図・回帰モデルを可視化する

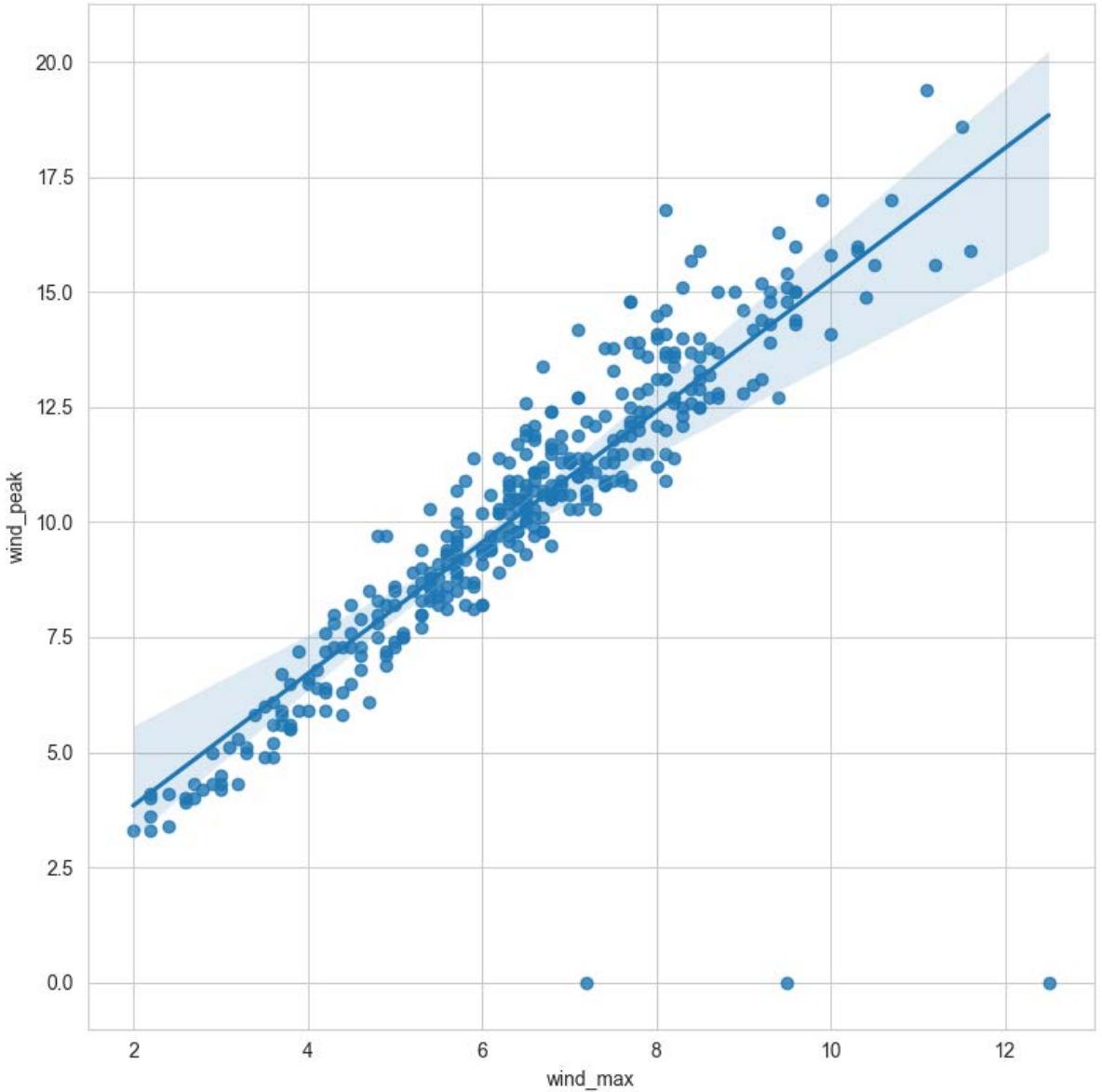
```
In [49]: ## seaborn regplotで線形回帰直線と信頼区間を描画してみる.
import warnings
warnings.simplefilter('ignore')

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

x = df_inter.loc[:, ['wind_max']] # 「最大風速」は10分間の平均風速の最大値
y = df_inter.loc[:, ['wind_peak']] # 「最大瞬間風速」は瞬間風速の最大値

sns.set_style("whitegrid")
plt.figure(figsize=(8, 8))

sns.regplot(x='wind_max', y='wind_peak', data=df_inter.loc[:, ['wind_max', 'wind_peak']])
plt.tight_layout()
plt.show()
```



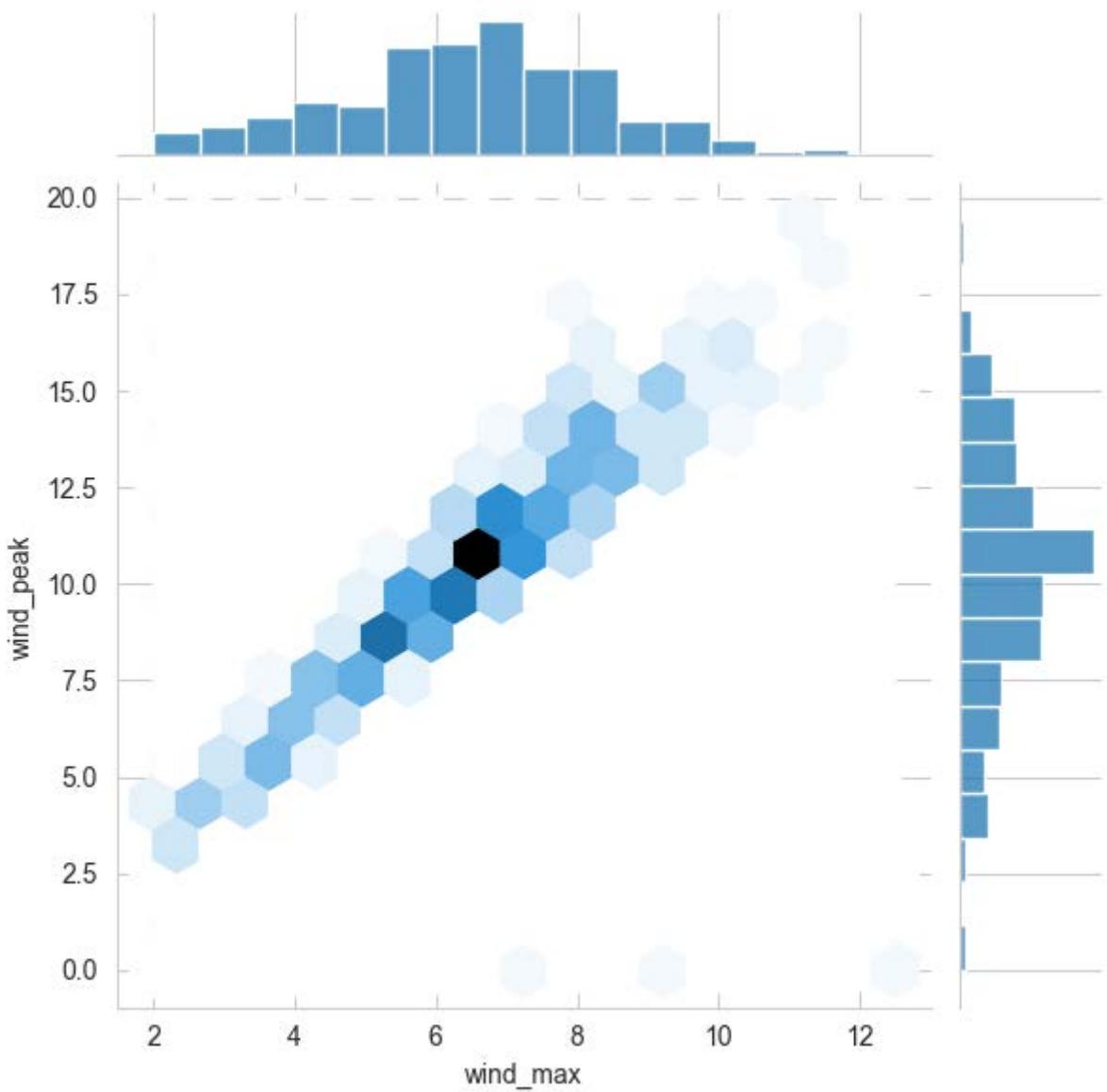
```
In [50]: ## joinlot, hexで見てみると、度数が大きく線形性のあるクラスターが見えてきて、かつ wind_peakのt
```

```
plt.figure(figsize=(16, 16))

sns.jointplot(x='wind_max', y='wind_peak', data=df_inter.loc[:, ['wind_max',
```

```
plt.show()
```

```
<Figure size 1600x1600 with 0 Axes>
```



```
In [51]: ## 確証を得るために,wind_max と wind_peakの記述統計量を比較する(両方とも風速なので自然ならい)
df_inter[['wind_max','wind_peak']].describe()
```

	wind_max	wind_peak
count	365.000000	365.000000
mean	6.484384	10.248767
std	1.907404	3.196122
min	2.000000	0.000000
25%	5.300000	8.300000
50%	6.500000	10.500000
75%	7.800000	12.400000
max	12.500000	19.400000

ここまでで明らかになったことは、

- 'temp_ave','rain_total','wind_max','wind_peak' の4つの説明変数の組み合わせのうち、 'wind_max','wind_peak'は線形性の関係にある（両方とも風速であるので）
- seaborn.regplot の図から読み取ると、 $wind_peak = wind_max * 1.56$ ぐらいと見える（原点を通ると仮定している）

- 'wind_max','wind_peak' の線形回帰直線からのばらつきは、ガウス分布であると仮定してよい。
- wind_peakの記述統計量として、最小値がゼロというのは異常と思われる (wind_maxの最小値が2.0にも関わらず、瞬間最大風速がゼロとは、どういうことなのか？？)

分位数を指定した、外れ値の除去を行う (`df.quantile()`)

今回は365日中で、3点異常に小さい値が見つかるということから、分位数0.8%の値を取得して、外れ値を除去していく。

```
In [52]: # 分位数を指定するdf.quantile()
q = df_inter['wind_peak'].quantile(0.008)
print(q)

df_inter_quan = df_inter.query('wind_peak < @q') # queryを使って外れ値となっている
df_inter_quan
## 確かに, wind_peak が 0.0 であって異常な値が入っているレコード(2021/5/1, 2, 3の3日間)か
3.0095999999999994
```

	date	temp_ave	temp_high	temp_low	rain_total	sun_shine	wind_ave	wind_max
120	2021/5/1	11.8	22.5	4.9	14.5	3.6	2.4	7.2
121	2021/5/2	10.6	17.3	5.0	1.0	6.0	3.7	9.5
122	2021/5/3	9.8	17.7	3.2	0.0	7.8	2.1	12.5

```
In [53]: # 外れ値となっているレコード3つを取り除いたデータセット(欠損値,異常値は無いので,各種分析に使える状態)
df_cleansed = df_inter.query('wind_peak >= @q') # queryを使って外れ値となっているレコードを削除
#df_cleansed
```

4.3 scikit-learn を用いた線形回帰の実行例: 単回帰分析

```
In [54]: # sklearn.linear_model.LinearRegression クラスを読み込み
from sklearn import linear_model
clf = linear_model.LinearRegression() # 線形回帰モデル(教師無し学習)を作る

X = df_cleansed.loc[:, ['wind_max']] # 説明変数に "wind_max (最大風速)" を利用
Y = df_cleansed.loc[:, ['wind_peak']] # 目的変数に "wind_peak" を利用
clf.fit(X, Y) # 線形予測モデルを作成

coef = float(clf.coef_); print(coef) # 回帰係数の表示
intercept = float(clf.intercept_); print(intercept) # 切片 (誤差) の表示
##print(clf.score(X, Y)) # 決定係数の表示

# 線形回帰直線による予測関数の定義
def wind_peak_regression(wm):
    return float(coef * wm + intercept)
```

1.5599976694562885
0.26008134773198677

これにより、予測値[window_peak] = 1.55999767 x [window_max] + 0.26008135 である

回帰式が求められた。

原点を通らず +0.26だけオフセットがあるのは、風速計の原理、機材の性質の関係かもしれない。他県のデータも取り寄せて比較するのも良いかもしない。

wind_peak に異常値が入っているレコード（2021/5/1, 2, 3の3日間）については、上でもとめた線形回帰式を使って、数値補完する方法が考えられる。

【異常値修正の実装まとめ】

1. df_inter DataFrameの各レコードのwind_peak値が 分位数0.8%の値（約3.009）を下回っているならば異常値と判断する。
2. この異常値を含むレコードを除去したクレンジング済の df_cleansed DataFrame に対して線形回帰予測を行う（説明変数 wind_max, 目的変数 wind_peak）。
3. 求めた回帰係数 coef と切片 intercept を用いて、wind_max 値からその日のwind_peak の予測値を線形回帰式で使って計算し、異常値であったフィールド値を書き換える。

```
In [55]: # pandas where関数を使って,cond論理式(第1引数)の結果がFalseのときに行う置換処理を第2引数に
# 注意:今回は異常値を上書きする形の処理にしている(inplace=True 第3引数)
df_inter['wind_peak'].where(df_inter['wind_peak'] >= q, coef * df_inter['wind_max'] + intercept, inplace=True)

# wind_peak が異常値であったレコード(2021/5/1, 2, 3の3日間) index 120,121,122 の wind_peak
##df_inter
```

```
In [56]: # 修正が終わったDataFrameを CSVファイルに書き出し
df_inter.to_csv('JMA_Nagano_2021_fixed.csv')
```

(参考) 教師なし学習(決定木)を用いた外れ値の自動検知 IsolationForest

参考:CUBE SUGAR CONTAINER | 教師なし学習で外れ値の検知に使える IsolationForest というアルゴリズムを試してみる

このアルゴリズムの興味深いところは、教師データの中にある程度外れ値が含まれていても構わないという点。つまり、アノテーションしていないデータをそのまま突っ込むことが許容されている。

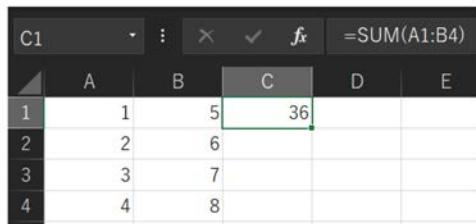
IsolationForest のアルゴリズムでは、決定木を使った分類しやすさにもとづいてデータが正常か外れ値かを判断する。外れ値は正常なデータに比べると数が少なく、特徴が大きく異なると仮定する。だとすると、外れ値は正常なデータに比べて分類するのに木の深さがより多く必要と考える。

(オプション) Excel Power Query によるデータ加工・分析[1]

データ加工のツール

- データ加工に使われるツールは多岐にわたります
- 最良の選択は、入力データの形式や、所望のデータモデル、そこから得たい情報、そのために必要な計算等に応じて変わります
 - しばしば複数のツールをプログラミング環境の下で組合せます
 - 万能のツールは有りません
- 表データに限定すればExcelは万能に近いツールです
 - Excelは表計算をするプログラミング環境です
 - 「方眼紙」としての使い方は本来の用途ではありません
 - HTML形式の表データであろうと、自動的に変換してくれます
 - HTMLファイルをExcelで開けば、ページ全体が変換されます
 - Webページ上でコピーして、Excel上で張り付けることもできます
- 以降では、Excelを利用した表データの加工を中心に説明します
 - Microsoft 365準拠

データ集計

- 総和を入れるセルに =SUM(と入力してから、総和を取るセルの範囲をドラッグで指定し、Enterで確定させます。
 - 例：
- SUMをAVERAGEに変えると算術平均が計算できます
- STDEV.Pに変えると標準偏差が計算できます

四則演算と反復

- セルを参照する計算式を記入できます

- 例：

C1	A	B	C	D	E
	x	✓	f _x	=A1*B1/2	
1	1	5	2.5		
2	2	6			
3	3	7			
4	4	8			

- 計算式があるセルの右下の角をドラッグすると反復できます

- 例：

C1	A	B	C	D	E
	x	✓	f _x	=A1*B1/2	
1	1	5	2.5		
2	2	6	6		
3	3	7	10.5		
4	4	8	16		
5					

=A4*B4/2

文字列処理

- 文字列を処理する演算子や関数も用意されています

- 計算式の中で文字列そのもの（リテラル）を書くときには"で囲みます

- &は文字列結合です

- 例：

C1	A	B	C	D	E
	x	✓	f _x	=A1&" "&B1	
1	東大	太郎	東大 太郎		
2	John	Doe	John Doe		

- SUBSTITUTE(文字列, 検索文字列, 置換文字列)は文字列置換です

- 例：

B1	A	B	C	D	E	F
	x	✓	f _x	=SUBSTITUTE(A1,"データ","data")		
1	データ加工	data加工				
2	データベース	dataベース				

フィルタとソート

- 表を選択して[データ]»[フィルター]を実行すると、その選択範囲をフィルタ可能になります
 - 選択範囲の1行目は属性名として解釈されます
 - 1行目がドロップダウンになります
- ドロップダウンから、行のフィルタやソートができます

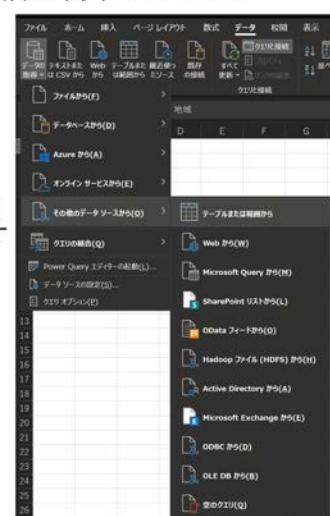
A	B
1 地域	気温°C
2 東京	10
3 大阪	13
4 福岡	15



Power Queryとテーブル

- Power Query機能を使うと表データに対する演算が簡単にできます
 - テーブルと呼ばれる単位で演算します
- [データ]»[データの取得]
 - [他のデータソースから]
 - [テーブルまたは範囲から]で選択範囲をテーブルに変換できます
- Power Queryエディターを[閉じて読み込む]と別のシートに変換後のテーブルが保存されます

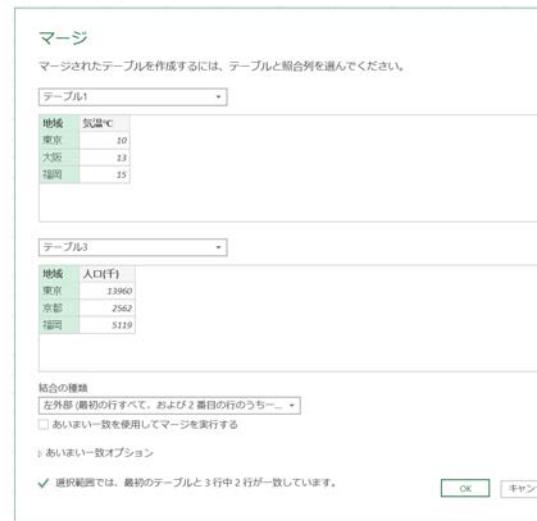
A	地域	B	気温°C
1	東京		10
2	大阪		13
3	福岡		15



- [クエリと接続]でテーブル一覧が見えます

テーブルの結合

- ・ [データの取得]»[クエリの結合]»[マージ]から異なるテーブルを結合して新しいテーブルを作れます
- ・ 結合には複数種類あります
 - ・ 内部：照合列が一致する行だけの結合
 - ・ 左外部：上のテーブル全てと下のテーブルから照合列が一致する行の結合
 - ・ 右外部：下のテーブル全てと上のテーブルから照合列が一致する行の結合
- ・ 外部結合における不一致行のセルは適当にnullで埋まります

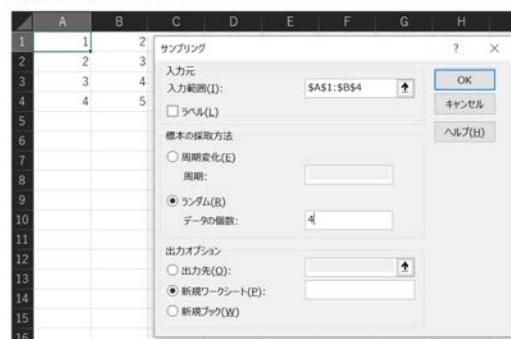


東京大学 数理・情報教育研究センター 佐藤重幸 2021 CC BY-NC-SA

11

分析ツールと統計処理

- ・ アドインの分析ツールを使うと統計処理が簡単にできます
 - ・ [ファイル]»[オプション]»[アドイン]»[Excelアドイン]の[管理]»[分析ツール]のチェックを入れて[OK]をすると、アドインを有効にできます
 - ・ Cf. [Excel で分析ツールPak を読み込む](#)
 - ・ [データ]»[データ分析]という項目が追加されます
- ・ 利用例：ランダムサンプリング
 1. [データ分析]»[サンプリング]
 2. 入力範囲を指定
 - ・ セルのドラッグで指定可能
 3. 採取するデータの個数を指定
 4. 出力先を指定
 - ・ デフォルトは新規シート



東京大学 数理・情報教育研究センター 佐藤重幸 2021 CC BY-NC-SA

12

(参考) ビッグデータ処理ツール

ビッグデータ処理

- ビッグデータとは、伝統的なデータ加工ツールで容易に扱えないほど大きいデータを指します
 - 伝統的なデータベース管理システムに載らない
 - 勿論Excelにも載らない
- 典型的にはビッグデータは1つの計算機のメモリに載りません
→ 分散データ処理
- ビッグデータ処理ツール
 - Apache Spark [Zaharia et al. '14] <https://spark.apache.org/>
 - 分散データセットを手軽に扱える多目的プログラミング環境
 - Apache Hadoop <https://hadoop.apache.org/>
 - 分散ファイルシステム上の大規模データ処理基盤の古典
 - Sparkへの代替が進んでいる
 - HillView [Budiu et al. '19] <https://github.com/vmware/hillview>
 - Excelのような表計算ツールの分散処理版

- [Apache Spark](#)
- [Apache Hadoop](#)
- [HillView](#)

memo