Python3を用いた
データ処理の基礎

pandas / NumPy / SciPy /
StatsModels / matplotlib

を用いて

第0.1.0版

中村勝則

2018年5月12日

目 次

1　はじめに

本書は Python 言語処理系と関連モジュールを用いた統計解析と機械学習の入門書である．特に統計学をこれから学ぶ学生を読者として想定しており，可能な限り基礎知識について解説し，Python による実習方法を紹介する．また，既に統計学について学んでいる人に対しては，本書は Python の関連ライブラリの導入のための資料として利用できる．

統計解析と機械学習の分野では，対象となる特定のデータが存在しており，それらデータに対して実際に処理を施すことで，基礎となる理論とそれらの応用方法についての理解が深まると筆者は信じている．本書を執筆している時点では，統計解析のためのソフトウェアとしては R **言語**（GNU R）とその関連ライブラリが主流であるが，Python 言語処理系には統計解析以外の幅広い分野のソフトウェア・ライブラリが多数提供されているだけでなく，更なる進化発展が予想されることから，Python 言語処理系を対象プラットフォームとして採用した．

1.1　Python 処理系と関連パッケージについて

本書が想定している Python の版は 3 である．また，統計解析のために必要となるパッケージとして，pandas，NumPy，SciPy，StatModels，skit-learn などを使用する．またグラフの描画には matplotlib を使用する．
Python の処理系と関連パッケージの導入や管理の方法に関する詳しいことは，別の情報源 [1] を参照のこと．

2　データ処理のための基礎事項

本書では，Python を中心にして複数のパッケージソフトウェアを使用することを前提とするが，基本的なデータ構造を提供するパッケージとしては NumPy，pandas がある．この内 Numpy は，数値の配列データ を取り扱うための基本的な機能を提供するものであり，pandas を始めとする他のパッケージの基礎部分として利用されている．pandas は統計解析のためのデータ構造とそれらを処理する機能を提供するものであり，データ処理の作業は基本的に，pandas のメソッドや関数を利用する形をとる．

Python の対話モードで表やグラフを簡便な形で扱うために JupyterLab （IPython ベースの対話環境）を使用する．またデータの入出力の対象としては CSV 形式のデータファイルの扱いを取り上げる．更に SQL でアクセスするデータベースに関しても，広く普及している実装を取り上げて若干の説明をする．

2.1　データファイルの形式

統計解析の対象となるデータは多種多様であるが，本書では表や列の形に整形されたデータの扱いを前提とする．特に表形式のデータは統計解析の際の基本的なデータ形式であり，複数の行を縦方向に蓄積したものである．表形式の各行はデータの**レコード**であり，ある**対象**の属性（**項目**）毎の**値**を横方向に並べたもの [2] として扱う．すなわち，1 つの行がデータ集合の単位であるということができる．このようなデータの表現形式は，データベースや RDF [3] の扱いにおける考え方の前提となっており重要である．

データをファイルとして記録媒体に保存する場合も表形式のデータとして扱うことが多く，特にテキストファイルとして表形式データを保存する際の代表的なファイル形式に CSV がある．CSV は RFC4180 として標準化されており，多くのソフトウェアにおいて利用できるデータフォーマットである．（ファイル名の拡張子は ‘*.csv’）
CSV データは，各項目をコンマ「,」で区切って並べるという素朴なものであり，テキストエディタで作成，編集することもできる．

---

[1] 多くの優良なる書籍やインターネットサイトが出版，公開されている．
（拙著を参考文献 [2][3] にご紹介させていただきます．無料で公開しています）
[2] 「**対象** の **属性** は **値** である」という表現の構造であり，この構造の連鎖としてデータを取り扱うのが **RDB**（リレーショナル・データベース）の基本的な考え方である．
[3] RDF: Resource Description Framework，Web コンテンツの表現のための論理構造．

2.2　表形式のデータを扱うパッケージとクラス

表形式のデータの扱いには pandas [4] パッケージを用いる．このパッケージは，表とデータ列を処理するための機能を提供するものであり，DataFrame クラスのオブジェクトとして表形式のデータを扱う．

pandas パッケージを利用するためには，それを Python 処理系に読み込む（インポートする）必要がある．

> **《pandas パッケージの利用開始》**（pandas パッケージのインポート）
>
> **書き方：**　import pandas
>
> これにより，'pandas' の接頭辞を付加して pandas の関数やメソッドを実行することができる．
> インポートする際に，
>
> import pandas as pd
>
> とすると，'pd' と接頭辞を短縮することができ，本書では主としてこの形でパッケージを読み込むこととする．

2.2.1　CSV 形式データの読込み

read_csv 関数を用いることで CSV 形式データを読み込んで DataFrame のインスタンスを生成することができる．CSV データ 'dat_kokugo.csv' を読み込んで DataFrame オブジェクト df を生成する例を次に示す．

例. CSV データの読込み

```
In [1]: import pandas as pd     # パッケージの読込み
In [2]: df = pd.read_csv('dat_kokugo.csv')     # データの読込み
In [3]: df.head(3)     # 先頭3行の表示
```

Out[3]:

	番号	氏名	得点
0	1	田原 慎太郎	34
1	2	矢吹 孝明	66
2	3	小平 英俊	69

参考 1） read_csv メソッドにキーワード引数 encoding=**文字コード** を与えると，CSV データの文字コード体系を指定することができる．

参考 2） CSV 形式データは，それぞれの列（項目）毎にデータ型が統一されていることが望ましいが，そうでない場合には read_csv メソッド実行時に DtypeWarning （データ型警告）が出ることがある．これはエラーではないが，read_csv メソッドの引数にキーワード引数 low_memory=False を与えておくことで警告メッセージを抑止できる．

DataFrame オブジェクトに対して head メソッドを使用すると先頭から指定した行数の部分を取り出して，それを別の DataFrame オブジェクトとして返す．head とは逆に，最終行から指定した行数を取り出すには tail メソッドを使用する．

[4] オープンソースのデータ処理パッケージ．（公式インターネットサイト：　http://pandas.pydata.org/）

例. 末尾からのレコードの抽出（先の例の続き）

```
In [4]: df.tail(3)     # 末尾3行の表示
```

Out[4]:

	番号	氏名	得点
2397	2398	三和 珠希	72
2398	2399	手崎 美衣奈	82
2399	2400	伊窪 けあき	98

2.2.1.1 CSV 読み込みの際の見出し行について

CSV データは，最初の行が**見出し行**となっていることが一般的であり，pandas パッケージで読み込む際も最初の行を見出し行とみなす．従って先頭行が見出し行となっていない CSV データを読み込む際は注意が必要である．例えば図 1 のような見出し行の無い CSV データの扱いについて考える．

```
1, 田原　慎太郎 , 67
2, 矢吹　孝明 , 49
3, 小平　英俊 , 60
4, 北林　正昭 , 100
5, 涌井　賢 , 70
6, 森元　邦雄 , 27
7, 草場　良幸 , 50
8, 波多江　哲治 , 79
9, 高屋　忠義 , 75
10, 新岡　一弥 , 51
```

図 1: 見出し行の無い CSV 形式データ ‘dat_kokugo_noheader.csv’ の例

これを先の例と同じ方法で読み込む例を次に示す．

例. CSV データ（見出し行無し）の読込み (1)

```
In [5]: df = pd.read_csv('dat_kokugo_noheader.csv')     # データの読込み
```

```
In [6]: df.head(3)     # 先頭3行の表示
```

Out[6]:

	1	田原 慎太郎	34
0	2	矢吹 孝明	66
1	3	小平 英俊	69
2	4	北林 正昭	88

この例からもわかるように，データの最初の行（レコード）が見出し行と見做されている．CSV データの最初のレコードもデータであると見做して読み込むには，次の例のように read_csv を実行する際にキーワード引数 names を指定する．

例. CSV データ（見出し行無し）の読込み (2)

```
In [7]: df = pd.read_csv('dat_kokugo_noheader.csv',
                         names=('No.','Name','Point'))    # データの読込み

In [8]: df.head(3)    # 先頭3行の表示
```

Out[8]:

	No.	Name	Point
0	1	田原 慎太郎	34
1	2	矢吹 孝明	66
2	3	小平 英俊	69

この例では read_csv を実行する際にキーワード引数 names=('No.','Name','Point') を指定している．この引数には付与する項目名（列の名前）のタプルを与える．

2.2.2 DataFrame に対する基本的な操作

2.2.2.1 行，列の抽出

DataFrame オブジェクトから特定の行や特定の列，あるいは行と列を指定して特定の値を取り出すことができる．基本的には Python 言語の**スライス**と呼ばれる**添字**の指定と同じ考え方で取り出すことができる．例えば次の例に示すような CSV データ 'dat_seiseki.csv' があるとする．

例. 成績データ 'dat_seiseki.csv' の読み込み

```
In [9]: df = pd.read_csv('dat_seiseki.csv')    # データの読込み

In [10]: df.head(3)    # 先頭3行の表示
```

Out[10]:

	番号	氏名	国語	物理
0	1	田原 慎太郎	85	3
1	2	矢吹 孝明	42	75
2	3	小平 英俊	45	49

head メソッドでは先頭から指定した行数のデータを取り出すが，次の例のようにスライスを指定して**行の抽出**ができる．

例. スライス指定による行の抽出（先の例の続き）

```
In [11]: df2 = df[1200:1203]    # インデックス1200以上,1203未満の抽出

In [12]: df2    # 内容確認
```

Out[12]:

	番号	氏名	国語	物理
1200	1201	前田 樹里	56	75
1201	1202	岡田 このみ	66	15
1202	1203	山越 麻理	50	92

この例では，インデックス 1200 以上,1203 未満の行（3 行分）を抽出して，それを別の DataFrame オブジェクト df2 として与えている．

スライスに列の名前（項目名）を与えると，指定した**列の抽出**ができる．(次の例参照)

例. 列の抽出（先の例の続き）

```
In [13]: sr = df['氏名']     # '氏名'の列を抽出

In [14]: sr.head(3)     # 先頭3件の表示

Out[14]: 0    田原 慎太郎
         1     矢吹 孝明
         2     小平 英俊
         Name: 氏名, dtype: object
```

この例では DataFrame オブジェクト df から ‘氏名’ の列だけを抽出して，それを sr に得ている．このようにして抽出した列のデータは Series クラスのオブジェクトであり，項目名（列の名前）を持たない.

以上に例示したように，インデックスの指定により行を抽出し，項目名の指定により列を抽出する．また，特定の行と列を絞り込んで抽出する場合はこの順序で取り出すと良い．あるいは loc メソッドを使用すると，行と列の絞り込みを一度に行って抽出することができる.（次の例参照）

例. loc メソッドによる DataFrame オブジェクトの部分抽出（先の例の続き）

```
In [15]: df.loc[1200:1203,['氏名','物理']]     # 部分を絞り込み
```

Out[15]:

	氏名	物理
1200	前田 樹里	75
1201	岡田 このみ	15
1202	山越 麻理	92
1203	小田島 るみ	17

この例では，インデックス 1200 以上 1203以下（!注意!）の範囲で，’ 氏名’ と’ 物理’ の列を抽出している.

2.2.2.2 管理情報の取得

DataFrame オブジェクトが持つインデックスに関しての情報は index プロパティが保持している.（次の例参照）

例. DataFrame の index プロパティ（先の例の続き）

```
In [16]: df.index     # indexプロパティの確認

Out[16]: RangeIndex(start=0, stop=2400, step=1)
```

この例では,「第 0 以上第 2400未満（!注意!）で 1 刻み」のインデックスが付いていることがわかる．項目名（列の名前）の情報は columns プロパティが保持している.（次の例参照）

例. DataFrame の columns プロパティ（先の例の続き）

```
In [17]: df.columns     # columnsプロパティの確認

Out[17]: Index(['番号', '氏名', '国語', '物理'], dtype='object')

In [18]: df.columns[0]     # columnsプロパティの0番目の確認

Out[18]: '番号'
```

項目名の情報が得られていることがわかる.

DataFrame オブジェクトに info メソッドを使用すると管理情報が表示される.（次の例参照）

例. info メソッドの実行（先の例の続き）

```
In [19]: df.info()    # infoメソッドの実行
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2400 entries, 0 to 2399
Data columns (total 4 columns):
番号    2400 non-null int64
氏名    2400 non-null object
国語    2400 non-null int64
物理    2400 non-null int64
dtypes: int64(3), object(1)
memory usage: 75.1+ KB
```

2.2.2.3 行の整列

DataFrame の行の並びを整列するには sort_values メソッドを使用する．このメソッドの第 1 引数に項目名（列の名前）を与える．（次の例参照）

例. DataFrame の行の降順の整列（先の例の続き）

```
In [20]: df2 = df.sort_values('物理', ascending=False) # 物理の点数で整列
         df2.head(7)    # 物理の点数のトップ7を表示
```

Out[20]:

	番号	氏名	国語	物理
132	133	知野 健也	90	100
2085	2086	東司 育巳	63	100
1543	1544	上條 亜喜	79	100
116	117	苺谷 国幸	50	100
549	550	西坂 靖雄	82	100
1283	1284	久多良木 万裕子	32	100
399	400	顔 法仁	40	99

これは「物理」の項目の値に従って**降順**に行を整列した例である．キーワード引数 ascending=False が降順の指定である．このメソッドは，処理の結果の DataFrame を返し，元の DataFrame の内容は変更しない．**昇順**に整列するにはキーワード引数 ascending=True を与える．（次の例参照）

例. DataFrame の行の昇順の整列（先の例の続き）

```
In [21]: df2 = df.sort_values('国語', ascending=True) # 国語の点数で整列
         df2.head(5)    # 国語の点数のワースト5を表示
```

Out[21]:

	番号	氏名	国語	物理
1683	1684	細矢 由利子	0	26
2349	2350	坂手 紗和	1	82
186	187	木寺 健司	1	14
956	957	上佐 勝嘉	2	3
2276	2277	小佐見 亜貴代	3	47

2.2.2.4　行の条件抽出

スライスに条件を与えて DataFrame の行を抽出することができる．

例. 「国語」が 2 点以下の行の抽出（先の例の続き）

```
In [22]: df[ df['国語']<=2 ]
```

Out[22]:

	番号	氏名	国語	物理
186	187	木寺 健司	1	14
956	957	上佐 勝嘉	2	3
1683	1684	細矢 由利子	0	26
2349	2350	坂手 紗和	1	82

条件に合う行を抽出したものが別の DataFrame オブジェクトとして返される．条件として使える演算子には <，>，<=，>=，== などがある．

抽出結果が DataFrame なので，各種のメソッドが使用できる．(次の例)

例. 「物理」が 100 点のレコードを抽出して「国語」の順に上位 3 件取得（先の例の続き）

```
In [23]: df[ df['物理']==100 ].sort_values('国語',ascending=False).head(3)
```

Out[23]:

	番号	氏名	国語	物理
132	133	知野 健也	90	100
549	550	西坂 靖雄	82	100
1543	1544	上條 亜喜	79	100

スライスの中に記述する条件式は括弧 '( )' で括って論理和，論理積の結合ができる．(次の例)

例. 「国語」「物理」が共に 10 点以下の行の抽出（先の例の続き）

```
In [24]: df[ (df['国語']<10) & (df['物理']<10) ]
```

Out[24]:

	番号	氏名	国語	物理
111	112	小坂部 久展	6	6
924	925	上柿 道則	8	0
956	957	上佐 勝嘉	2	3

例. 「国語」「物理」どちらかが 100 点の行を抽出して上位 3 件表示（先の例の続き）

```
In [25]: df[ (df['国語']==100) | (df['物理']==100) ].head(3)
```

Out[25]:

	番号	氏名	国語	物理
116	117	苺谷 国幸	50	100
132	133	知野 健也	90	100
261	262	中杉 光祐	100	32

'( )' で括った条件式の先頭に ' ~ ' を付加すると，その条件の否定を意味する．

2.2.2.5　新規作成

DataFrame オブジェクトは生成時にコンストラクタに列の情報を与える．

例. DataFrame オブジェクトの生成

```
In [1]: import pandas as pd    # パッケージの読込み
```

```
In [2]: df = pd.DataFrame(columns=['番号','氏名','英語','化学'])    # 新規作成
        df.info()    # 管理情報表示
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 4 columns):
番号    0 non-null object
氏名    0 non-null object
英語    0 non-null object
化学    0 non-null object
dtypes: object(4)
memory usage: 0.0+ bytes
```

内容が空の DataFrame が作成されていることがわかる．この DataFrame に対して列毎にデータを与える例を示す．

例. 列毎にデータを与える（先の例の続き）

```
In [3]: df['氏名'] = ['山田 太郎','田中 花子','中村 勝則']    # 氏名の列をセット
        df['番号'] = [1,2,3]        # 番号の列をセット
        df['英語'] = [52,61,89]     # 英語の列をセット
        df['化学'] = [81,72,64]     # 化学の列をセット
        df.info()    # 管理情報の表示
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 4 columns):
番号    3 non-null int64
氏名    3 non-null object
英語    3 non-null int64
化学    3 non-null int64
dtypes: int64(3), object(1)
memory usage: 176.0+ bytes
```

内容の確認を試みる．

例. 内容の確認（先の例の続き）

```
In [4]: df    # 内容確認
```

Out[4]:

	番号	氏名	英語	化学
0	1	山田 太郎	52	81
1	2	田中 花子	61	72
2	3	中村 勝則	89	64

2.2.2.6　行の追加

DataFrame オブジェクトに行（レコード）を追加するには，追加したい行のデータを Series クラスのオブジェクトとして生成し，append メソッドを使用してそれを対象の DataFrame オブジェクトに追加する．ただし，元の DataFrame オブジェクトは変更されず，追加拡張された新たな DataFrame オブジェクトが生成される．DataFrame に行を追加する作業の例を次に示す．

例. 行の追加（先の例の続き）

In [5]:
```
s = pd.Series(
    [4,'木村 美子',91,95],
    index=['番号','氏名','英語','化学'],
    name=3)          # Seriesオブジェクトsの生成
s    # 内容確認
```

Out[5]:
```
番号         4
氏名     木村 美子
英語        91
化学        95
Name: 3, dtype: object
```

In [6]:
```
df = df.append(s)    # DataFrameにsを追加
df    # 内容確認
```

Out[6]:

	番号	氏名	英語	化学
0	1	山田 太郎	52	81
1	2	田中 花子	61	72
2	3	中村 勝則	89	64
3	4	木村 美子	91	95

Series オブジェクトにはインデックスを与えることができるが，これは DataFrame のインデックスとは意味が異なる．Series オブジェクト生成時のコンストラクタには引数として

データリスト, index=インデックスリスト, name=名前

を与えて生成し，これを DataFrame オブジェクトに append メソッドを使用して追加する．Series 生成時に与える「名前」が対象 DataFrame のインデックス位置,「インデックスリスト」が「データリスト」の各要素をどの項目（列）データとして追加するかを決める．

2.2.2.7 列の追加

DataFrame オブジェクトに列（項目）を追加するには，追加したい列のデータを Series クラスのオブジェクトとして生成し，対象の DataFrame オブジェクトに列の名前のスライス（添字）を与えて代入する．DataFrame に列を追加する作業の例を次に示す．

例. 列の追加（先の例の続き）

In [7]:
```
s = pd.Series([45,82,56,97],index=[0,1,2,3]) # Seriesオブジェクトの生成
s    # 内容確認
```

Out[7]:
```
0    45
1    82
2    56
3    97
dtype: int64
```

In [8]:
```
df['地理'] = s     # DataFrameにsを追加
df    # 内容確認
```

Out[8]:

	番号	氏名	英語	化学	地理
0	1	山田 太郎	52	81	45
1	2	田中 花子	61	72	82
2	3	中村 勝則	89	64	56
3	4	木村 美子	91	95	97

’地理’ の列が追加されていることがわかる．この方法では元の DataFrame が変更される形で列が追加される．

2.2.2.8　値の参照と変更

DataFrame に対して at で位置指定して，その位置の値を参照，変更することができる．(次の例参照)

例. DataFrame 内の値の参照と変更（先の例の続き）

```
In [9]: df.at[0,'地理'] = 65  # インデックス0の'地理'の値の更新
        df    # 内容確認
```

Out[9]:

	番号	氏名	英語	化学	地理
0	1	山田 太郎	52	81	65
1	2	田中 花子	61	72	82
2	3	中村 勝則	89	64	56
3	4	木村 美子	91	95	97

この例のように　at[**インデックス, 列名**]　で DataFrame 内の特定の位置を指し示す．

2.2.2.9　行や列の削除

DataFrame オブジェクトの特定の行や列を削除するには drop メソッドを使用する．

例. 指定したインデックスの行を削除する（先の例の続き）

```
In [10]: df = df.drop(2)    # インデックスが2の行を削除
         df    # 内容確認
```

Out[10]:

	番号	氏名	英語	化学	地理
0	1	山田 太郎	52	81	65
1	2	田中 花子	61	72	82
3	4	木村 美子	91	95	97

この例ではインデックスが 2 の行を削除している．drop メソッドは元の DataFrame を変更せずに行を削除した形の DataFrame オブジェクトを生成する．

DataFrame の特定の列を削除するには Python の del 文を使用することができる．

例. 特定の列を削除する（先の例の続き）

```
In [11]: del df['地理']    # '地理'の列を削除
         df    # 内容確認
```

Out[11]:

	番号	氏名	英語	化学
0	1	山田 太郎	52	81
1	2	田中 花子	61	72
3	4	木村 美子	91	95

この例では　del df[' 地理']　とすることで' 地理' の列を削除している．

2.2.3　CSV 形式データの保存

DataFrame の内容を CSV 形式データとして保存するには to_csv メソッドを使用する．

例. DataFrame オブジェクトを CSV 形式データとして保存する（先の例の続き）

```
In [12]: df.to_csv('dat_seiseki_2.csv',index=False)    # CSVとして保存
```

この処理の結果，図 2 のような CSV 形式ファイル 'dat_seiseki_2.csv' ができる．

```
番号,氏名,英語,化学
1,山田 太郎,52,81
2,田中 花子,61,72
4,木村 美子,91,95
```

図 2: 保存した CSV 形式データ ‘dat_seiseki_2.csv’ の内容

DataFrame の各行にはインデックスが付与されているが，CSV データ保存時にこれを含めない場合は to_csv メソッドにキーワード引数 index=False を指定する．これを指定しない場合はインデックスの列を含んだ形で CSV データを作成する．

参考） to_csv メソッドにキーワード引数 encoding=**文字コード** を与えると，保存する際の文字コード体系を指定することができる．

2.2.4 DataFrame から条件を満たす行を抽出する方法

次のような項目から成る住所に関する CSV 形式のデータ address_jp.csv の扱いを考える．

01) 住所 CD	**06)** 都道府県	**11)** 町域カナ
02) 都道府県 CD	**07)** 都道府県カナ	**12)** 京都通り名
03) 市区町村 CD	**08)** 市区町村	**13)** 字丁目
04) 町域 CD	**09)** 市区町村カナ	**14)** 字丁目カナ
05) 郵便番号	**10)** 町域	

例. address_jp.csv の読込みと管理情報の表示

```
In [1]: import pandas as pd    # パッケージの読込み
```

```
In [2]: df = pd.read_csv('address_jp.csv',low_memory=False)    # データの読込み
        df.info()    # 管理情報の表示
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149107 entries, 0 to 149106
Data columns (total 14 columns):
住所CD       149107 non-null int64
都道府県CD     149107 non-null int64
市区町村CD     149107 non-null int64
町域CD       149107 non-null int64
郵便番号       149107 non-null object
都道府県       149107 non-null object
都道府県カナ     149107 non-null object
市区町村       149107 non-null object
市区町村カナ     149107 non-null object
町域         146911 non-null object
町域カナ       148762 non-null object
京都通り名      1135 non-null object
字丁目        6858 non-null object
字丁目カナ      6858 non-null object
dtypes: int64(4), object(10)
memory usage: 15.9+ MB
```

149,107 件のレコード（行）を持つデータであることがわかる．これは次のようなデータである．

In [3]:
```
df.head(3)     # 先頭3行の表示
```

Out[3]:

	住所CD	都道府県CD	市区町村CD	町域CD	郵便番号	都道府県	都道府県カナ	市区町村	市区町村カナ	町域	町域カナ	京都通り名	字丁目	字丁目カナ
0	60000000	1	1101	11010000	060-0000	北海道	ホッカイドウ	札幌市中央区	サッポロシチュウオウク	NaN		NaN	NaN	NaN
1	60855200	1	1101	11010000	060-8552	北海道	ホッカイドウ	札幌市中央区	サッポロシチュウオウク	NaN		NaN	NaN	NaN
2	60872100	1	1101	11010000	060-8721	北海道	ホッカイドウ	札幌市中央区	サッポロシチュウオウク	NaN		NaN	NaN	NaN

このデータから，’町域’ が ’中村町’ であるレコードを抽出する．この場合は loc メソッドを使用する．

例. ’町域’ が ’中村町’ であるレコードの抽出

In [4]:
```
df2 = df.loc[ df['町域'] == '中村町' ]     # 町域が'中村町'であるレコードを抽出
```

In [5]:
```
df2.head(3)     # 内容確認
```

Out[5]:

	住所CD	都道府県CD	市区町村CD	町域CD	郵便番号	都道府県	都道府県カナ	市区町村	市区町村カナ	町域	町域カナ	京都通り名	字丁目	字丁目カナ
11995	38271100	2	2321	23210017	038-2711	青森県	アオモリケン	西津軽郡鰺ヶ沢町	ニシツガルグンアジガサワマチ	中村町	ナカムラマチ	NaN	NaN	NaN
36172	368005100	11	11207	112070043	368-0051	埼玉県	サイタマケン	秩父市	チチブシ	中村町	ナカムラマチ	NaN	NaN	NaN
52021	232003300	14	14105	141050032	232-0033	神奈川県	カナガワケン	横浜市南区	ヨコハマシミナミク	中村町	ナカムラチョウ	NaN	NaN	NaN

抽出したデータの中から更に ’都道府県’ が ’愛知県’ であるレコードを抽出する．

例. 先の処理で得られた表から更に ’都道府県’ が ’愛知県’ であるレコードを抽出

In [6]:
```
# 更に都道府県が'愛知県'であるレコードを抽出
df3 = df2.loc[ df2['都道府県'] == '愛知県' ]
```

In [7]:
```
df3     # 内容確認
```

Out[7]:

	住所CD	都道府県CD	市区町村CD	町域CD	郵便番号	都道府県	都道府県カナ	市区町村	市区町村カナ	町域	町域カナ	京都通り名	字丁目	字丁目カナ
81517	453005300	23	23105	231050073	453-0053	愛知県	アイチケン	名古屋市中村区	ナゴヤシナカムラク	中村町	ナカムラチョウ	NaN	NaN	NaN
84095	444021500	23	23202	232020180	444-0215	愛知県	アイチケン	岡崎市	オカザキシ	中村町	ナカムラチョウ	NaN	NaN	NaN
84947	475087300	23	23205	232050131	475-0873	愛知県	アイチケン	半田市	ハンダシ	中村町	ナカムラチョウ	NaN	NaN	NaN
84948	475858500	23	23205	232050131	475-8585	愛知県	アイチケン	半田市	ハンダシ	中村町	ナカムラチョウ	NaN	NaN	NaN

2.3 データベースの扱い

大量のデータを扱うには**データベース**の取り扱いが必須となる．データベースとしてデータを蓄積管理するためのシステムは DBMS [5]（データベース管理システム）と呼ばれる独立したシステムである．利用者側のアプリケーションシステムは，この DBMS からデータを抽出したり，DBMS に対してデータの登録や更新などを行う．この際の DBMS に対する処理の依頼を**クエリ**（query）と呼び，クエリのための言語として SQL [6] がある．本書では DBMS として主にパブリックドメインのフリーソフトウェアである SQLite の扱いを取り上げ，SQL を介したデータベースの使用方法について説明する．SQLite は充分な性能がある上に導入方法も簡単であるため，データベースの個人的な運用や学習に適している．

2.3.1 データベースの基礎事項

本書では**関係モデル** [7] に基づく**リレーショナル・データベース**（関係データベース，RDB）を取り扱う．RDB では，データを**対象**，**属性**，**値**の組として扱う．具体的には，これら組の 1 つを 1 行として扱い，複数の列（**項目**）の値を横方向に書き並べて 1 つの**レコード**とする．これは表計算ソフトウェアで表を扱う場合に似ている．すなわち，左端の列に対象物の名称を記述し，以降の右側には各種の項目の値を書き並べる形式である．関係データベースは，表を参照することによって対象の属性の値を返す，すなわち「…の〜は何？」という問い合わせ（クエリ）に対して，「△△△です」と値を返すシステムと見做すことができる．

RDB では，複数の**表**（table）をまとめて 1 つの**データベース**（database）とする．表計算ソフトウェアの場合では，1 つのデータファイルは複数の表（スプレッドシート）を束ねた形 [8] になっており，この意味でも，RDB と表計算ソフトウェアの類似点（表 1）が見られる．

表 1: RDB と表計算ソフトウェアの類似点

RDB	表計算ソフトウェア
table	シート
database	シートを束ねたデータファイル

RDB では特定の項目（列）を**主キー**（primary key）として定め，主キーによる高速なデータ検索を可能にしている．また，table の検索の結果として得られた値を**外部キー**（foreign key）として，別の table の主キーにすることで検索を連鎖することができ，複雑な情報検索を可能にする．

RDB では表の構成を設計する際に，どのような項目の集まりとするかに注意を払う必要があり，可能な限り，同じような項目や値が繰り返し現れることのないように配慮する必要がある．このような，無駄や矛盾が起こらないような配慮のために**正規化**と呼ばれる設計上の工夫が重要であるが，これに関しては本書では触れない．

2.3.1.1 データベースにおける基本的な操作

RDB に対する基本的な操作は**選択**，**射影**，**結合**の 3 種類である．データベースは多量のデータを保持するものであり，必要な行（レコード）や列（項目）を絞り込んで抽出することが重要であり，選択は「必要な行の抽出」，射影は「必要な列の抽出」を意味する．また，RDB では 1 つのデータベース（database）に複数の表（table）があり，それら複数の表を連鎖させて情報を抽出する際の基本的な操作が「結合」である．これらに関しては，後で事例を挙げて具体的に説明する．

2.3.2 データベースにアクセスするためのパッケージ

本書では主として pandas パッケージの DataFrame として表を扱うが，RDB から必要な情報を取得する際も，得たデータ集合を DataFrame として扱う．SQL のクエリを扱うための Python 用のパッケージとして SQLAlchemy が

[5] Microsoft 社の Access や SQL Server，オープンソースの PostgreSQL や MySQL，パブリックドメインの SQLite などがある．
[6] SQL: データベースにアクセスする際の標準的な言語であり，多くの DBMS がその名称に「SQL」という語を冠している．
[7] IBM のエドガー・F・コッドによって考案された現在もっとも広く用いられているデータモデル．
[8] Microsoft 社の Excel における標準的なデータファイル形式である「ブック形式」などがその例である．

ある．SQLAlchemy はオープンソースのソフトウェアであり，公式インターネットサイト

http://www.sqlalchemy.org/

から詳しい情報が得られる．

本書では，SQLAlchemy と pandas を併用することで RDB を扱う．

2.3.2.1 データベースへのアクセス例

データベースを新規に作成して，pandas の DataFrame をデータベースの table に登録する方法の最も簡単な例を示す．

例. pandas の DataFrame を SQLite の table に登録する

```
In [1]: import sqlalchemy as SQLA    # SQLAlchemyの読込み
        import pandas as pd          # pandasの読込み
```

```
In [2]: df = pd.DataFrame(columns=['番号','氏名','英語','化学'])  # 新規作成
        df['氏名'] = ['山田 太郎','田中 花子','中村 勝則']          # 氏名の列をセット
        df['番号'] = [1,2,3]        # 番号の列をセット
        df['英語'] = [52,61,89]     # 英語の列をセット
        df['化学'] = [81,72,64]     # 化学の列をセット
        df     # 内容確認
```

Out[2]:

	番号	氏名	英語	化学
0	1	山田 太郎	52	81
1	2	田中 花子	61	72
2	3	中村 勝則	89	64

```
In [3]: egn = SQLA.create_engine('sqlite:///testdb', echo=False)    # DB接続エンジン
        df.to_sql('tbl01',egn,index=False)     # DataFrameをDBに書き込み}
```

この例では，In[1] でパッケージの読込みを行っている．その際，SQLAlchemy を SQLA という接頭辞で表現するものとしている．In[2] では pd という DataFrame を生成して値を設定している．データベースへの接続とデータの登録は In[3] で行っている．

SQLAlchemy では，Engine というオブジェクトを介してデータベースにアクセスする．そのために，データベースの利用に先立って，このオブジェクトを生成する必要があり，そのためには create_engine というメソッドを使用する．このメソッドの第 1 引数に，アクセス対象のデータベースの URL を与え，正常に実行されると，Engine オブジェクトが返されて**データベースへの接続**が完了する．

■ SQL クエリの表示

データベースへの実際のアクセスは，SQLAlchemy が SQL クエリを発行することで実現されている．今回の例では create_engine にキーワード引数 echo=False を与えているが，echo=True を与えると次の例のように，発行された SQL クエリが表示される．

```
SELECT CAST('test plain returns' AS VARCHAR(60)) AS anon_1
SELECT CAST('test unicode returns' AS VARCHAR(60)) AS anon_1
PRAGMA table_info("tbl01")
CREATE TABLE tbl01 (
   "番号" BIGINT,
   "氏名" TEXT,
   "英語" BIGINT,
   "化学" BIGINT
)
COMMIT
BEGIN (implicit)
INSERT INTO tbl01 ("番号", "氏名", "英語", "化学") VALUES (?, ?, ?, ?)
((1, '山田 太郎', 52, 81), (2, '田中 花子', 61, 72), (3, '中村 勝則', 89, 64))
COMMIT
```

次に，データベースからデータを読み込む例を示す．

例. SQLite の table からデータを読み込んで DataFrame にする

```
In [1]: import sqlalchemy as SQLA    # SQLAlchemyの読込み
        import pandas as pd          # pandasの読込み
```

```
In [2]: egn = SQLA.create_engine('sqlite:///testdb', echo=False)    # DB接続エンジン
        q = 'SELECT * from tbl01'    # SQLクエリ
        df = pd.read_sql(q,egn)      # データベースから読込み
        df    # 内容確認
```

Out[2]:

	番号	氏名	英語	化学
0	1	山田 太郎	52	81
1	2	田中 花子	61	72
2	3	中村 勝則	89	64

この例では In[2] で，データベースへの接続とデータの読込みを行っている．この際，変数 q に読込み処理のための SQL 文を与え，pandas の read_sql メソッド（データベースからデータを読み込むメソッド）に与えている．

重要.

ここで示した方法は最も簡易な例であり，実際のデータベース運用のためには，表の項目のデータ型の設定や，キーの設定（主キー，外部キー）などが重要になる．SQLAlchemy に関する更に具体的な事柄（作業など）に関しては付録「C.2 SQLAlchemy によるデータベースの扱い」に記載する．

2.4 統計処理のためのデータの流れ

Python で統計処理を行う場合は複数のパッケージを併用するが，ここで各種パッケージと，そこで扱うオブジェクトの型，入出力の対象について整理する．各種データを保持するシステム資源としては CSV 形式のテキストファイルとデータベースが多く用いられる．すなわち，これらのシステム資源から必要なデータを抽出して Python 処理系に読み込み，各種パッケージで処理を施した後，処理の結果を必要に応じて CSV テキストファイルとして保存したり，データベースに追加登録（更新処理）する．また必要に応じてグラフとして可視化する．

使用するパッケージとしては pandas と NumPy/SciPy が主たるものである．pandas では表形式のデータを DataFrame オブジェクトとして扱い，NumPy/Scipy では配列データを ndarray オブジェクトとして扱う．DataFrame は，非数値のデータを含む一般的な表を扱うものであるが，数値ばかりの配列データを処理する際は ndarray の方が適しており，実際の統計処理においては，必要に応じて DataFrame と ndarray の間でデータ変換をした後，それぞれのパッケージが提供する関数やメソッドを実行することになる．

システム資源，パッケージ，データクラスの関連を図 3 に示す．

2.4.1 DataFrame と ndarray 相互の変換

DataFrame と ndarray 相互の変換は，実際の統計処理でしばしば必要となる．変換作業の例を以下に示す．

例. モジュールの読込み

```
In [1]: # パッケージの読込み
        import pandas as pd      # pandasパッケージ
        import numpy as np       # NumPyパッケージ
```

pandas パッケージを pd，NumPy パッケージを np として読み込んでいる．次に，NumPy の ndarray を生成して，それを pandas の DataFrame に変換する例を示す．まずは配列の生成を示す．

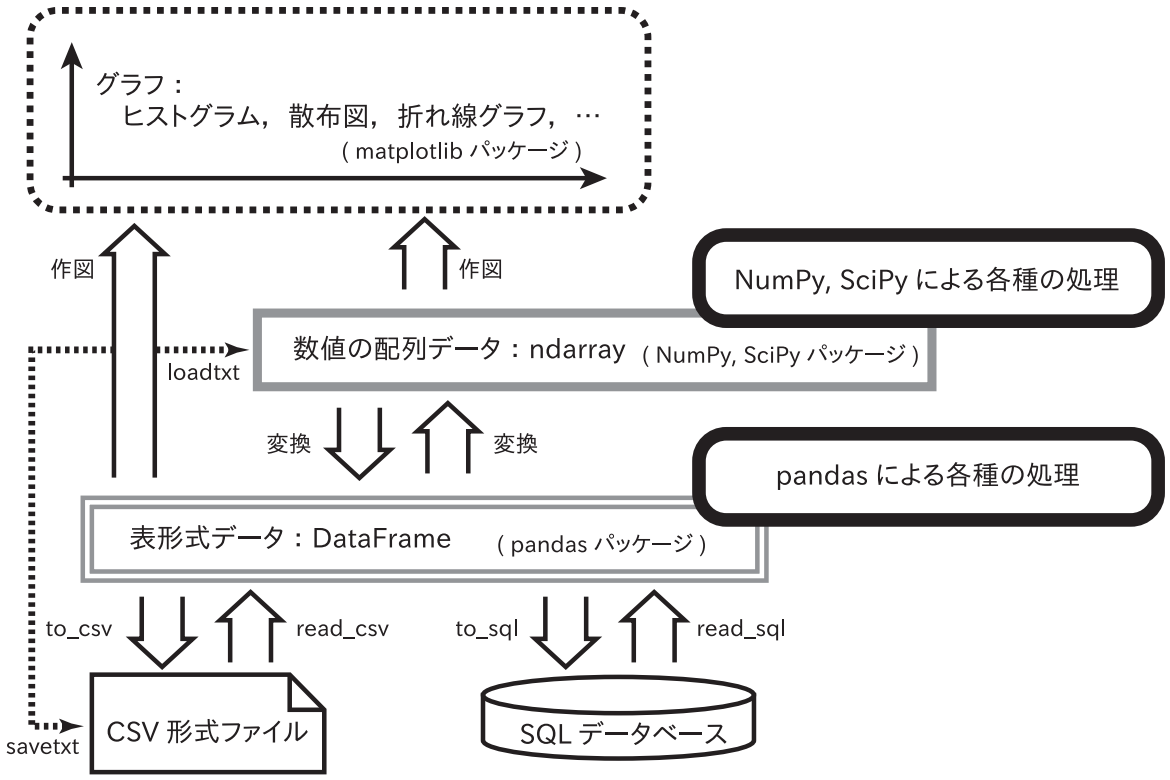


図 3: 統計処理のためのデータの流れ

例. ndarray オブジェクトの生成（続き）

In [2]:
```
# 配列の生成
x = np.arange( 0, 4, 1)
y1 = 2*x + 1
y2 = x**2
# 内容確認
print(x);   print(y1);  print(y2)
```

```
[0 1 2 3]
[1 3 5 7]
[0 1 4 9]
```

これは $0 \leq x < 4$ の整数を定義域として ndarray オブジェクト x として生成し，それに対する $y = 2x+1$，$y = x^2$ の値域をそれぞれ ndarray オブジェクト y1，y2 として生成している例である．これら x，y1，y2 を pandas の DataFrame オブジェクト df に与える例を次に示す．

例. ndarray を DataFrame に変換する例（続き）

In [3]:
```
# 配列をDataFrameに変換する (1)
df1 = pd.DataFrame(columns=['x','y1','y2'])
# 各列に配列を与える
df1['x'] = x;  df1['y1'] = y1;   df1['y2'] = y2
# 内容確認
df1
```

Out[3]:

	x	y1	y2
0	0	1	0
1	1	3	1
2	2	5	4
3	3	7	9

pd を生成した後で x，y1，y2 を与えているのがわかる．df の列の名前を 'x'，'y1'，'y2' としている．

次に，DataFrame の内容を ndarray に変換する例を示す．

例. DataFrame の列を ndarray に変換する例（続き）

In [4]:
```
# DataFrameから配列に変換 (1)
s = np.array( df1['x'] )
t = np.array( df1['y1'] )
u = np.array( df1['y2'] )
# 内容確認
print(s);   print(t);   print(u)
```

```
[0 1 2 3]
[1 3 5 7]
[0 1 4 9]
```

次は，DataFrame の内容を 2 次元の配列として 1 度に変換する例を示す．

例. DataFrame から 2 次元の ndarray への変換（続き）

In [5]:
```
# DataFrameから配列に変換 (2)
ar = np.array( df1 )
# 内容確認
print( ar )
```

```
[[0 1 0]
 [1 3 1]
 [2 5 4]
 [3 7 9]]
```

これとは逆に，2 次元の ndarray から DataFrame に一度に変換することも可能である．次にその例を示す．

例. 2 次元の ndarray から DataFrame への変換（続き）

In [6]:
```
# 配列をDataFrameに変換する (2)
df2 = pd.DataFrame( ar, columns=['x','y1','y2'] )
# 内容確認
df2
```

Out[6]:

	x	y1	y2
0	0	1	0
1	1	3	1
2	2	5	4
3	3	7	9

2.4.2 ndarray における表の構成

1 次元の ndarray の列は表における「行」を表す．すなわち表を構成するには「複数の行を書き並べる」という表現を用いる．

例. 1 次元の配列（続き）

```
In [7]: # 1次元の列（行）
        a1 = np.array( [10,2,3] )    # 1行目
        print( a1 )
```

```
[10  2  3]
```

ndarray オブジェクト a1 に 1 次元の配列が作成されている．これは「表」を構成するための 1 つの「行」であると見做す．更に複数の行を生成して，それらを用いて「表」を構成する例を次に示す．

例. 2 次元の配列（続き）

```
In [8]: # 更に2行目と3行目
        a2 = np.array( [30,4,6] )    # 2行目
        a3 = np.array( [50,6,9] )    # 3行目
        # 表を構成する
        ar1 = np.array( [a1,a2,a3] )
        print( ar1 )
```

```
[[10  2  3]
 [30  4  6]
 [50  6  9]]
```

1 次元配列を [a1, a2, a3] と並べる手法で 2 次元の配列を構成している．

行と列を転置（行と列の入れ替え）した形で 2 次元配列を作成するには次の例のように NumPy の column_stack メソッドを用いると良い．

例. 転置して 2 次元配列を生成（続き）

```
In [9]: # 転置して表を構成する
        ar2 = np.column_stack( [a1,a2,a3] )
        print( ar2 )
```

```
[[10 30 50]
 [ 2  4  6]
 [ 3  6  9]]
```

ndarray オブジェクトに対して T を作用する形でも表を転置することができる．

例. T による転置（続き）

```
In [10]: # 転置オペレーションでも同様
         print( ar1.T )
```

```
[[10 30 50]
 [ 2  4  6]
 [ 3  6  9]]
```

先に生成した ar1 を転置して ar2 と同じ配列を得ている．

2.4.2.1 2 次元配列の行，列へのアクセス

2 次元の ndarray オブジェクト `ar` があるとき，`n` 番目の行には `ar[n]` としてアクセスする．また `m` 番目の列には `ar[:,m]` あるいは `ar.T[m]` としてアクセスする．

2.4.3 ndarray の CSV ファイルに対する入出力

pandas の DataFrame を介することなく，NumPy の ndarray オブジェクトと CSV ファイルとの間でデータを入出力することができる．（数値のみの CSV に限る）

生成した ndarray オブジェクトを CSV ファイルに書き出す処理と，CSV ファイルから ndarray オブジェクトのデータを読み込む処理の例を以下に示す．

例．モジュールの読込みとデータの生成

```
In [1]: import numpy as np  # NumPyモジュール
        import matplotlib.pyplot as plt # matplotlibモジュール
```

```
In [2]: # 三角関数のデータ列
        x = np.arange(-6.28,6.28,0.02)
        y1 = np.sin(x); y2 = np.cos(x)
        ar = np.column_stack( [x,y1,y2] )   # 表（2次元配列）にする
```

この例では，x，$\sin(x)$，$\cos(x)$ の値の列を持つ配列 ar を生成している．これを CSV ファイルに出力する例を次に示す．

例． ndarray の内容を CSV ファイルに出力する（続き）

```
In [3]: # CSVファイルに保存
        np.savetxt( 'dat_triangle.csv', ar, delimiter=',' )
```

これで ar の内容が CSV ファイル dat_triangle.csv に出力される．

次に，CSV ファイル dat_triangle.csv の内容を ndarray オブジェクト tbl に読み込む例を示す．

例． CSV ファイルの読込み（続き）

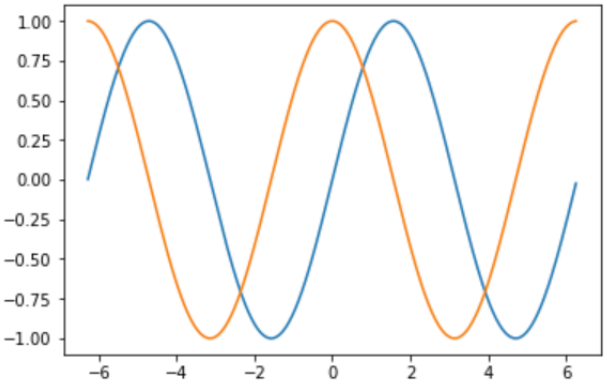
```
In [4]: # CSVファイルから読込み
        tbl = np.loadtxt( 'dat_triangle.csv', delimiter=',' )
```

読み込んだデータ tbl をグラフにプロットする例を次に示す．

例． tbl の内容のプロット（続き）

```
In [5]: # グラフ作成 （方法1）
        plt.plot(tbl[:,0],tbl[:,1])
        plt.plot(tbl[:,0],tbl[:,2])
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x13f00200b38>]
```



例． 上と同様の処理（続き）

```
In [6]: # グラフ作成 （方法2）
        plt.plot(tbl.T[0],tbl.T[1])
        plt.plot(tbl.T[0],tbl.T[2])
```

2.4.3.1 NumPy で CSV ファイルを扱う際の注意

NumPy の ndarray は数値の配列を扱うものであり，文字列データを含む CSV ファイルの読込みはできない．CSV ファイルの先頭が見出し行になっている場合は，それを読み飛ばす必要がある．loadtxt 実行時にキーワード引数 ‘skiprows=**行数**’ を与えることで，読み飛ばす行数を指定できる．

3　統計処理のための基礎事項

この章では，統計処理に必要となる作業のうち基本的なものについて説明する．また，各種の確率密度関数の算出や，それに従う乱数集合の生成について説明する．

3.1　基本的な値の算出

統計処理のためのデータ集合は pandas の DataFrame や NumPy の ndarray といったオブジェクトとして扱う．数値以外のデータを含んだ一般的な意味での表形式のデータは pandas の DataFrame で扱うが，数値のみの配列を処理する場合は，NumPy の ndarray として扱う方が処理時間の面で有利であることも多く，状況に応じて DataFrame と ndarray を使い分けるのが良い．DataFrame，ndarray 間でのデータ変換の方法も提供されている．

基本的な統計処理に関しては，pandas，NumPy に加え，SciPy パッケージも利用する．

3.1.1　パッケージの読込み

pandas，NumPy，SciPy パッケージを読み込む例を次に示す．

例. 各種パッケージの読込み

```
In [1]: import pandas as pd      # pandas
        import numpy as np       # NumPy
        from scipy import stats # SciPy統計モジュール
```

この例では，pandas を pd として，NumPy を np として読み込んでいる．また，SciPy からは stats モジュールを読み込んでいる．

3.1.2　データ集合の要約統計量の算出

データ集合から得られる**平均**，**分散**，**標準偏差**，**中央値**，**四分位点**，**最大値**，**最小値**，**最頻値**といった値を**要約統計量**という．要約統計量を**代表値**と呼ぶこともある．これから，要約統計量を算出する方法を例を挙げて説明する．

整数の乱数を 1,000,000 個保持する CSV データ dat_rand1M.csv を読み込んで，その平均値（算術平均）を求める例を次に示す．

例. 乱数のデータファイル dat_rand1M.csv の読込み（続き）

```
In [2]: df = pd.read_csv('dat_rand1M.csv')  # 1M個の整数値
        print( df.info() )  # 管理情報の表示
        df.head(3)          # 少し表示
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 1 columns):
data    1000000 non-null int64
dtypes: int64(1)
memory usage: 7.6 MB
None
```

Out[2]:

	data
0	1
1	18
2	86

この例では，読み取ったデータが DataFrame オブジェクト df として得られている．このオブジェクトは 'data' の名を持つ 1 つの列から成るもの（1 行目が 'data' で，2 行目以降が全て整数のテキスト）である．この 'data' の列の平均値を求めるには mean メソッドを使用する．このメソッドは，取り出した列の Series オブジェクトに対して使用する．すなわち，

```
df['data'].mean()
```

として実行する．[ ] の中の 'data' は DataFrame オブジェクト df の 'data' の列という意味である．今回は，平均値算出に要する時間を計測するために time モジュールを読み込んで使用する．

例. pandas による算術平均の算出（続き）

```
In [3]: import time      # 時間計測用モジュール

In [4]: # pandasによる平均値算出
        t1 = time.time()
        print( df['data'].mean() )  # 平均値の算出と表示
        t2 = time.time()
        print( t2 - t1 )            # 計算に要した時間の表示

        49.487526
        0.01399683952331543
```

この例では，mean メソッドの実行の前後で time モジュールの time メソッドを実行し，その差を取ることで実行に要した時間を計測している．算術平均は NumPy を用いても求めることができる．(次の例)

例. NumPy による算術平均の算出（続き）

```
In [5]: # NumPyによる平均値算出
        a = np.array(df['data'])    # NumPyの配列に変換
        t1 = time.time()
        print( np.mean(a) )         # 平均値の算出と表示
        t2 = time.time()
        print( t2 - t1 )            # 計算に要した時間の表示

        49.487526
        0.00499725341796875
```

これは pandas の Series オブジェクト df['data'] を，

```
a = np.array( df['data'] )
```

によって，NumPy の配列オブジェクト a に変換した後，NumPy の mean 関数を用いて

```
np.mean( a )
```

として算術平均を求める例である．
算術平均の値としては，どちらも 49.487526 という同じ値が得られているが，NumPy による算出の方が所要時間が短いことがわかる．従って，データの量や型などの状況に応じて，どのパッケージを使用するかを判断するのが良い．データ数が大きすぎない場合は，作業の能率の面から pandas のみの使用に絞っても良い．

■ 各パッケージの関連

数値の配列を扱うための最も基本的なパッケージが NumPy であり，配列を ndarray クラスのオブジェクトとして扱う．ndarray クラスに対する更に多くの関数やメソッドを SciPy が提供する関係になっている．pandas も各種機能の実装のためにパッケージ内部で NumPy を利用している．このような関連性があるため，統計処理はまず pandas で行うことを考え，状況に応じて NumPy や SciPy を利用するということになる．

生徒 2,400 人の国語と物理の成績のデータ dat_seiseki.csv を読み込んで各種の値を算出する例を示す．データは次のようにして読み込んで DataFrame オブジェクト df として生成する．

例. pandas による CSV データの読込み（続き）

```
In [6]: df = pd.read_csv('dat_seiseki.csv')  # 成績データの読込み
        df.head(3)           # 少し表示
```

Out[6]:

	番号	氏名	国語	物理
0	1	田原 慎太郎	85	3
1	2	矢吹 孝明	42	75
2	3	小平 英俊	45	49

データを読み込んで，先頭の数行を表示した例である．この DataFrame から ’国語’ の列（Series オブジェクト）を抽出したものに対して各種の値の取得を試みるが，この列を NumPy の ndarray オブジェクト a に変換したものも作成して，各種パッケージによる同様の計算作業を示す．

例. ’国語’ の列のデータ件数の表示（続き）

```
In [7]: # 要素数の表示
        print( 'pandas:\t', df['国語'].count() )
        a = np.array(df['国語'])     # NumPyの配列に変換
        print( 'len:\t', len( a ) )
```

```
pandas:  2400
len:     2400
```

Series オブジェクトに対して count メソッドを実行すると，そのデータ件数が得られる．また ndarray に関しては，Python の標準的な関数 len によって要素の個数が得られる．

3.1.2.1 四分位数の取得

DataFrame オブジェクトに対して describe メソッドを使用するとデータの個数と四分位数に関する情報が得られる．

例. 四分位数の取得（続き）

```
In [8]: df.describe()  # 情報の取得（戻り値：DataFrame）
```

Out[8]:

	番号	国語	物理
count	2400.000000	2400.000000	2400.000000
mean	1200.500000	66.751250	43.271667
std	692.964646	19.035918	24.107814
min	1.000000	0.000000	0.000000
25%	600.750000	55.000000	24.000000
50%	1200.500000	69.000000	42.000000
75%	1800.250000	81.000000	61.000000
max	2400.000000	100.000000	100.000000

戻り値は DataFrame 型である．

3.1.2.2　要約統計量を算出する機能

各種の要約統計量を算出する方法を，例を挙げて説明する．

■ 算術平均

pandas の mean メソッド，NumPy の mean 関数により得られる．

例. 算術平均（続き）

```
In [9]: # 平均値の算出と表示
        print( 'pandas:\t', df['国語'].mean() )
        print( 'NumPy:\t', np.mean(a) )
```

```
pandas:  66.75125
NumPy:   66.75125
```

■ 標準偏差

pandas の std メソッド，NumPy の std 関数により得られる．

例. 標準偏差（続き）

```
In [10]: # 標準偏差の算出と表示
         print( 'pandas:\t', df['国語'].std() )
         print( 'NumPy:\t', np.std(a) )
```

```
pandas:  19.035918488070173
NumPy:   19.031952258526537
```

pandas は不偏標準偏差を，NumPy は標本標準偏差を求める．（「A.3.1 標本分散と不偏分散」を参照のこと）．

■ 分散

pandas の var メソッド，NumPy の var 関数により得られる．

例. 分散（続き）

```
In [11]: # 分散の算出と表示
         print( 'pandas:\t', df['国語'].var() )
         print( 'NumPy:\t', np.var(a) )
```

```
pandas:  362.36619268445185
NumPy:   362.21520677083333
```

pandas は不偏分散を，NumPy は標本分散を求める．（「A.3.1 標本分散と不偏分散」を参照のこと）．

■ 最大値

pandas の max メソッド，NumPy の max 関数により得られる．

例. 最大値（続き）

```
In [12]: # 最大値の算出と表示
         print( 'pandas:\t', df['国語'].max() )
         print( 'NumPy:\t', np.max(a) )
```

```
pandas:  100
NumPy:   100
```

■ 最小値

pandas の min メソッド，NumPy の min 関数により得られる．

例. 最小値（続き）

```
In [13]: # 最小値の算出と表示
         print( 'pandas:\t', df['国語'].min() )
         print( 'NumPy:\t', np.min(a) )
```

```
pandas:  0
NumPy:   0
```

■ 歪度

pandas の skew メソッド，SciPy の skew 関数により得られる．

例. 歪度（続き）

```
In [14]: # 歪度の算出と表示
         print( 'pandas:\t', df['国語'].skew() )
         print( 'SciPy:\t', stats.skew(a) )
```

```
pandas:  -0.5797870458584444
SciPy:   -0.579424616025507
```

pandas の場合と SciPy の場合で値が異なることに注意．

■ 尖度

pandas の kurt メソッド，SciPy の kurtosis 関数により得られる．

例. 尖度（続き）

```
In [15]: # 尖度の算出と表示
         print( 'pandas:\t', df['国語'].kurt() )
         print( 'SciPy:\t', stats.kurtosis(a) )
```

```
pandas:  0.14350651168155526
SciPy:   0.14070875536343763
```

pandas の場合と SciPy の場合で値が異なることに注意．

■ 中央値

pandas の median メソッド，NumPy の median 関数により得られる．

例. 中央値（続き）

```
In [16]: # 中央値の算出と表示
         print( 'pandas:\t', df['国語'].median() )
         print( 'NumPy:\t', np.median(a) )
```

```
pandas:  69.0
NumPy:   69.0
```

■ 最頻値

pandas の mode メソッド，SciPy の mode 関数により得られる．最頻値は複数得られる可能性があるので，pandas では Series オブジェクトとして，SciPy では配列として得られる．

例. pandas による最頻値（続き）

```
In [17]: m = df['国語'].mode() # 最頻値の算出
         print(m)     # 最頻値の表示
         type(m)      # 型
```

```
0    72
dtype: int64
```

```
Out[17]: pandas.core.series.Series
```

処理の結果として Series オブジェクトが得られている．個別の最頻値を取り出すには，得られた結果にスライス [ ] を付けて，インデックスを指定すると良い．

例. SciPy による最頻値（続き）

```
In [18]: m = stats.mode(a)    # 最頻値の算出
         print(m.mode)    # 最頻値の表示
         print(m.count)   # 出現回数
         type(m)      # 型
```

```
[72]
[63]
```

```
Out[18]: scipy.stats.stats.ModeResult
```

SciPy の mode 関数は戻り値として ModeResult 型のオブジェクトを返す．これのプロパティ mode に最頻値の配列が，プロパティ count にそれぞれの最頻値の出現頻度の配列が保持されている．

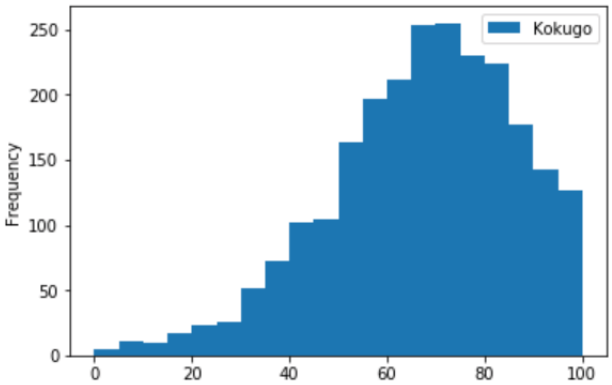
3.1.3 ヒストグラムの表示

DataFrame の列をヒストグラムとして表示するには plot メソッドを使用する．

例. plot メソッドによるヒストグラムの表示（続き）

```
In [19]: # グラフのインライン表示を設定
         %matplotlib inline

In [20]: ax = df.plot( kind='hist', y='国語',
                bins=20, label='Kokugo' )
```



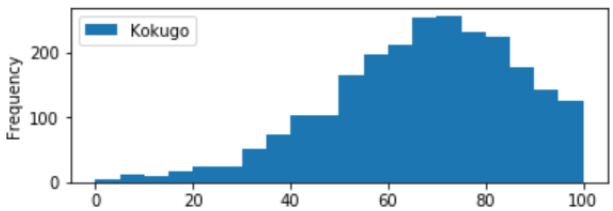
plot メソッドに与える基本的な引数（キーワード引数）は次の通りである．

キーワード引数	説明
kind=	グラフの種類を指定する．ヒストグラムの場合は 'hist' を与える．
y=	対象の列を指定する．
bins=	階級の数を指定する．
label=	グラフのラベルを指定する．
figsize=	グラフの縦横のサイズを指定する．

figzie でサイズを指定する例を次に示す．

例. ヒストグラムのサイズ指定（続き）

```
In [21]: ax = df.plot( kind='hist', y='国語',
                bins=20, label='Kokugo', figsize=(6,2) )
```



plot メソッドの結果，matplotlib パッケージの AxesSubplot オブジェクトが返される．このオブジェクトの中に作図されたグラフなどが保持されており，get_figure，savefig メソッドを組合せて，グラフをファイルとして出力することができる．(次の例)

例. 作図したグラフのファイルへの保存（続き）

```
In [22]: ax.get_figure().savefig('fig01.png', dpi=600)
```

この結果，600dpi の解像度の画像ファイル fig01.png が作成される．

3.2 テストデータ作成のための機能

統計処理のためのプログラミングでは，テストデータを与えて動作を検証することが必要になることがある．そのためには，各種の確率密度関数の値を求めたり，それに沿った乱数集合を生成するといった準備が必要である．それらに関することをここでは説明する．

■ 必要なパッケージの読込み

pandas，NumPy，SciPy に加え，基本的な数学関数を提供する math モジュールと，グラフの描画に必要となる matplotlib を次のようにして読み込む．

例. 各種パッケージの読込み

```
In [1]: import math              # mathモジュール
        import numpy as np       # NumPyモジュール
        import pandas as pd      # pandasモジュール
        from scipy import stats # SciPyのstatsモジュール
        import matplotlib.pyplot as plt # matplotlibモジュール
```

```
In [2]: # グラフのインライン表示を設定
        %matplotlib inline
```

グラフをノートブック内にインライン表示するために

%matplotlib inline

としている．

3.2.1 正規分布

最も基本的な確率密度関数が次に示す**正規分布**（normal distribution もしくは**ガウス分布** Gaussian distribution）である．（「A.1 中心極限定理と正規分布」参照）

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

正規分布 $f(x)$ における μ は**平均**（**期待値**），σ は**標準偏差**である．[9]

> **重要.** 正規分布以外の確率密度関数でも，μ と σ は定義のための基本的な要素である．

先の定義に従って正規分布を Python の関数 `normaldist` として記述したものを次に示す．

例. 正規分布の定義（続き）

```
In [3]: # 正規分布の定義を実装した関数
        def normaldist(x,m,s):
            return 1.0/(math.sqrt(2.0*math.pi)*s) * \
                        math.exp(-1.0*(x-m)**2/(2.0*s**2))
```

確率密度関数のうち，有名なものの多くは SciPy の stats パッケージが提供しており，正規分布関数も norm.pdf 関

[9] 正規分布は $N(\mu,\sigma^2)$ と表記されることが多く，特に $N(0,1)$ を**標準正規分布**という．

数[10] として提供されている．ただし，実用的な統計処理で求められる確率密度関数には既存のパッケージとして提供されていないものもあるので，今回の例のように，math パッケージの数学関数を用いて独自に定義を記述することが求められることもある．

ここで定義した normaldist 関数と，SciPy の norm.pdf 関数の両方で，$\mu = 0, \sigma = 1$ における $f(0)$ の値を求める例を次に示す．

例. $\mu = 0, \sigma = 1$ における正規分布関数 $f(0)$ の算出（続き）

```
In [4]: normaldist(0,0,1)      # μ=0,σ=1でx=0の値

Out[4]: 0.3989422804014327

In [5]: # SciPy提供の正規分布
        stats.norm.pdf(x=0, loc=0, scale=1)

Out[5]: 0.3989422804014327
```

norm.pdf 関数のキーワード引数 x= には定義域の値，loc= には μ の値，scale= には σ の値を与える．この例では，両方の関数で同じ値が算出されていることがわかる．

matplotlib で関数のプロットを描画するには，横軸と縦軸の値の列をそれぞれ作成する必要がある．その作業の例を次に示す．この場合も，normaldist と norm.pdf 関数の両方の値を生成してプロットしたものを比較する．

例. プロットのためのデータ列の生成（続き）

```
In [6]: # 正規分布のプロット用データ作成
        x = np.arange(-4,4,0.1)
        y1 = [normaldist(m,0,1) for m in x]
        y2 = [stats.norm.pdf(x=m,loc=0,scale=1) for m in x]
```

x に定義域のデータ列，y1 に normaldist 関数の値域，y2 に norm.pdf 関数の値域を生成している．

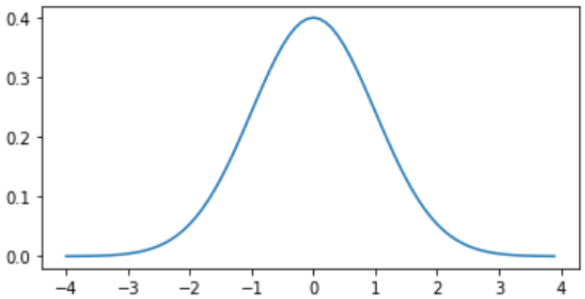
※ norm.pdf 関数のキーワード引数 'x=' には配列を与えることもでる．具体的には「3.3.1 確率密度関数：PDF」を参照のこと．

x を横軸，y1 を縦軸にしてグラフをプロットする例を次に示す．

例. normaldist のプロット（続き）

```
In [7]: plt.figure( figsize=(6,3) )
        plt.plot( x,y1 )

Out[7]: [<matplotlib.lines.Line2D at 0x1fd4bc9d2b0>]
```



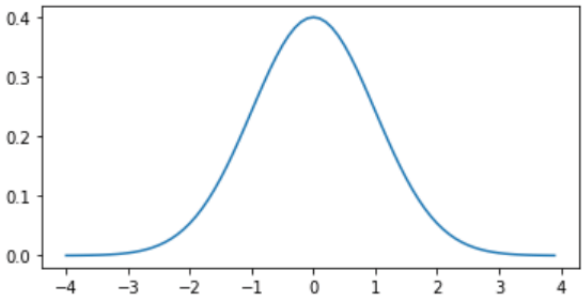
これが normaldist 関数の $-4 \leq x < 4$ におけるプロットである．同様に x を横軸，y2 を縦軸にしてグラフをプロットする例を次に示す．

[10] pdf は「確率密度関数」の英語 probability density function を省略したもの（acronym）である．

例. norm.pdf のプロット（続き）

```
In [8]: plt.figure( figsize=(6,3) )
        plt.plot(x,y2)
```

```
Out[8]: [<matplotlib.lines.Line2D at 0x1fd4bd2c748>]
```



これが norm.pdf 関数の $-4 \leq x < 4$ におけるプロットである．

3.2.1.1 正規分布に沿った乱数の生成

正規分布に沿った乱数集合を生成する方法について説明する．ScPy の stats モジュールは確率密度関数と同時に，それに沿った乱数発生のための関数も提供していることが多い．正規分布に沿った乱数発生関数は norm.rvs として提供されている．

次に示す例は，norm.rvs で 1,000,000 個生成した乱数データ（NumPy の ndarray 形式）を pandas の DataFrame に変換する例である．

例. 正規分布に沿った乱数発生（続き）

```
In [9]: # 正規分布に従う乱数を1,000,000個生成（μ=0,σ=1）
        a = stats.norm.rvs(loc=0, scale=1, size=1000000)
```

```
In [10]: df = pd.DataFrame( columns=['正規分布'] )
         df['正規分布'] = pd.Series( a )
```

norm.rvs 関数のキーワード引数 loc= には μ の値，scale= には σ の値，size= には生成する乱数の個数を与える．この例では生成した乱数集合を NumPy の ndarray オブジェクト a として生成し，それを DataFrame のオブジェクト df に変換している．列の名前は ’正規分布’ である．describe メソッドで df の情報を表示する例を次に示す．

例. describe メソッドによる情報表示（続き）

```
In [11]: df.describe()
```

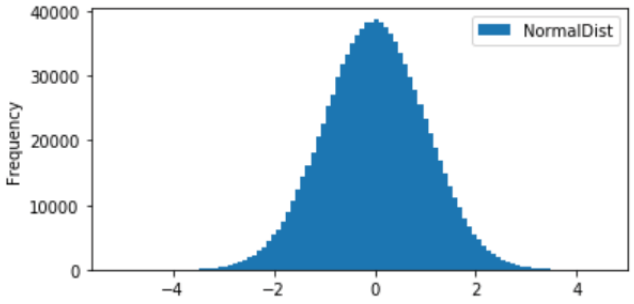
Out[11]:

	正規分布
count	1000000.000000
mean	-0.000910
std	1.000336
min	-5.138996
25%	-0.676364
50%	-0.001402
75%	0.674422
max	4.530882

この DataFrame をヒストグラムにして表示する例を次に示す．

例. ヒストグラムの表示（続き）

```
In [12]: ax = df.plot( kind='hist', y='正規分布',
             bins=100, figsize=(6,3), label='NormalDist' )
```



3.2.2 一様乱数

NumPy の random モジュールは一様乱数を生成する関数 rand，randint を提供している.（下記参照）

書き方	説明
np.random.rand()	0 以上 1 未満（小数点付き）の乱数を 1 つ生成して返す.
np.random.rand(**個数**)	0 以上 1 未満（小数点付き）の乱数を，指定した個数生成して返す.（ndarray 型）
np.random.randint(n1,n2)	n1 以上 n2 未満の整数の乱数を 1 つ生成して返す.
np.random.randint(n1,n2,size=**個数**)	n1 以上 n2 未満の整数の乱数を，指定した個数生成して返す.（ndarray 型）

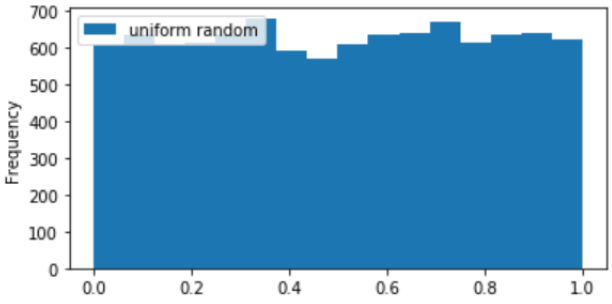
一様乱数をヒストグラムにする例を次に示す.

例. 0 以上 1 未満（浮動小数点数）の一様乱数のヒストグラムの表示（続き）

```
In [13]: # 一様乱数を10,000個生成（float）
         a = np.random.rand(10000)

In [14]: df = pd.DataFrame( columns=['一様乱数'] )
         df['一様乱数'] = pd.Series( a )

In [15]: ax = df.plot( kind='hist', y='一様乱数',
             bins=16, figsize=(6,3), label='uniform random' )
```

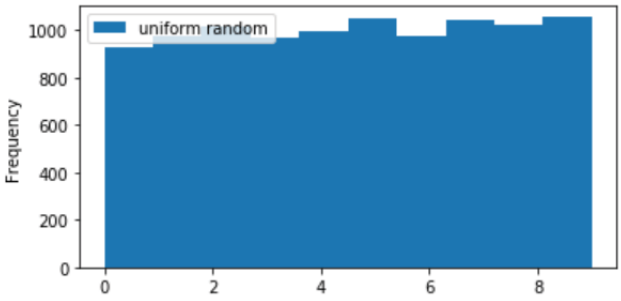


例. 0 以上 11 未満（整数）の一様乱数のヒストグラムの表示（続き）

```
In [16]: # 一様乱数を10,000個生成（int）
         a = np.random.randint(0,10,size=10000)

In [17]: df = pd.DataFrame( columns=['一様乱数'] )
         df['一様乱数'] = pd.Series( a )

In [18]: ax = df.plot( kind='hist', y='一様乱数',
             bins=10, figsize=(6,3), label='uniform random' )
```



3.2.3 階乗，順列，組合せ

統計学や確率論に関する計算をする際に，**階乗**，**順列**，**組合せ**は基本的な計算となる．

■ 階乗

階乗を求める関数は math モジュールと，SciPy の special モジュールの両方で提供されている．前者は整数型で計算結果を返し，後者は浮動小数点型で計算結果を返す．それぞれのモジュールで階乗を計算する例を次に示す．

例. 階乗の計算

```
In [1]: import math      # mathモジュール
        import scipy.special as ss  # SciPyのspecialモジュール

In [2]: # 各モジュールによる階乗の計算
        print( '10! =',math.factorial(10) ) # 10!
        print( '10! =',ss.factorial(10) )   # 10!
```

```
10! = 3628800
10! = 3628800.0
```

```
In [3]: # 複数の階乗を一度に求める
        print( ss.factorial([1,2,3,4,5,6]) )
```

```
[  1.   2.   6.  24. 120. 720.]
```

両方のモジュールとも，関数名は factorial であるが，この例が示すように SciPy の special モジュールが提供する関数は，リストの形で計算することができる．

■ 順列，組合せ

SciPy の special モジュールは，順列を求める関数 perm，組合せを求める関数 comb を提供する．それら関数を用いている例を次に示す．

例. 順列，組合せの計算（続き）

```
In [4]: # 順列： 5P2
        print( '5P2 =', ss.perm(5,2) )
        # 組合せ： 5C2
        print( '5C2 =',ss.comb(5,2) )
```

```
5P2 = 20.0
5C2 = 10.0
```

これら関数を用いることで，統計処理に必要な様々な関数を更に定義できる．その 1 つの例として，**二項分布**の関数定義を次に示す．

3.2.3.1　二項分布

各試行において成功か失敗かの 2 状態をとり，各試行において成功する確率が p であるという事象を考える．例えば，当たり／はずれのくじがあり，当たりが出る確率が p であるという状況を考える．このような状況で，n 回の試行[11] 中 x 回成功となる確率の分布 $f(x)$ は**二項分布**となる．具体的には次のような式になる．

$$f(x) = {}_nC_x \cdot p^x(1-p)^{n-x}$$

成功確率 p の試行を n 回行う場合の二項分布を $Bi(n,p)$ と書く．

【二項分布関数の実装例】

SciPy の special モジュールが提供する conb 関数を用いて二項分布の関数を定義し，それを実行する例を次に示す．

例. 二項分布関数 bi の定義と実行（続き）

In [5]:
```
# 二項分布関数の定義
def bi( n, p, x ):
    return ss.comb(n,x) * p**x * (1-p)**(n-x)
```

In [6]:
```
# p=0.5, n=5 で x = 0〜5
b1 = [bi(10,0.5,x) for x in range(11)]
print( b1 )
```

```
[0.0009765625, 0.009765625, 0.0439453125, 0.1171875, 0.2
05078125, 0.24609375, 0.205078125, 0.1171875, 0.04394531
25, 0.009765625, 0.0009765625]
```

これは，リストの内包表記を用いて複数の関数値を得ている例である．

二項分布の関数は SciPy の stats モジュールも提供（binom.pmf 関数[12] ）しており，実際の統計処理においてはそちらを利用する方がより簡便である．（次の例参照）

例. SciPy の stats モジュールが提供する二項分布関数 binom.pmf（続き）

In [7]:
```
# SciPy.statsの二項分布関数を用いて比較
from scipy import stats # SciPyのstatsモジュール
b2 = [stats.binom.pmf(x,10,0.5) for x in range(11)]
print( b2 )
```

```
[0.0009765625, 0.00976562500000001, 0.04394531249999999,
0.11718750000000014, 0.20507812500000022, 0.246093750000
00025, 0.20507812500000022, 0.11718750000000014, 0.04394
531249999999, 0.00976562500000001, 0.0009765625]
```

先に定義した関数 bi と比較すると，計算精度が若干異なることに注意すること．

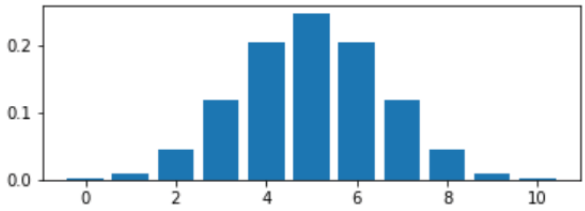
bi 関数によって算出された二項分布をヒストグラムにする例を次に示す．

[11] このような試行を**ベルヌーイ試行**という．
[12] pmf：probability mass function の略．「3.3.2 確率質量関数：PMF（Probability Mass Function）」のところでも解説している．

例. 二項分布のヒストグラム（続き）

```
In [8]: # ヒストグラムの作成
        import matplotlib.pyplot as plt # matplotlibモジュール
```

```
In [9]: plt.figure( figsize=(6,2) )
        fig = plt.bar( range(11), b1 )
```



この例のように，matplotlib の bar メソッドを使用することでヒストグラムを描画することができる．

3.3 統計処理に用いる様々な関数

SciPy には統計処理に利用できる関数が多数提供されており，scipy.stats モジュールを Python 処理系に読み込むことで使用可能となる．ここでは，統計処理に多く用いられるいくつかの関数の使用方法について例を挙げて説明する．

3.3.1 確率密度関数：PDF（Probability Density Function）

連続な変数に対する確率密度関数の使用例をいくつか挙げる．まず，scipy.stats モジュールと，関連パッケージ（NumPy，matplotlib）を読み込む．

例. scipy.stats，numpy，matplotlib の読込み

```
In [1]: import numpy as np       # NumPyモジュール
        from scipy import stats # SciPyのstatsモジュール
        import matplotlib.pyplot as plt # matplotlibモジュール
```

3.3.1.1 使用例：正規分布

正規分布の確率密度関数 [13] norm.pdf の使用例を次に示す．

例. 正規分布の確率密度関数 norm.pdf の実行（続き）

```
In [2]: # 正規分布の確率密度関数
        x1 = np.arange(-5,5,0.01)    # 定義域
        y08 = stats.norm.pdf(x=x1,loc=0,scale=0.8)  # 標準偏差:0.8
        y10 = stats.norm.pdf(x=x1,loc=0,scale=1)    # 標準偏差:1
        y14 = stats.norm.pdf(x=x1,loc=0,scale=1.4)  # 標準偏差:1.4
```

この例では，$-5 \leq x < 5$ の x に対する確率密度関数を，3 つの標準偏差 $\sigma = 0.8, \sigma = 1.0, \sigma = 1.4$ に関して算出している．norm.pdf 関数にキーワード引数を与えているが，x=**定義域の配列**，loc=**平均値（μ）**，scale=**標準偏差（σ）**である．

これをプロットする処理が次の例である．

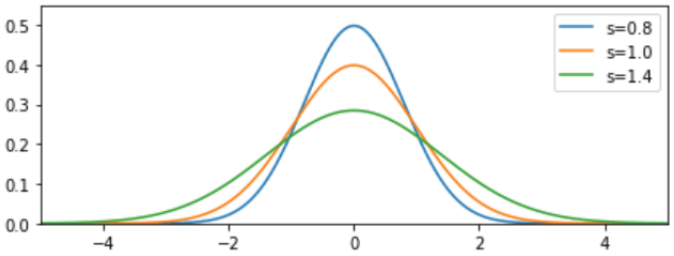
[13] 詳しくは「A.1 中心極限定理と正規分布」を参照のこと

例. 正規分布のプロット処理（続き）

```
In [3]: # プロット
        plt.figure( figsize=(7,2.5) )        # 描画サイズの設定
        plt.plot(x1,y08, label='s=0.8' )
        plt.plot(x1,y10, label='s=1.0' )
        plt.plot(x1,y14, label='s=1.4' )
        plt.ylim(0,0.55); plt.xlim(-5,5)     # 描画範囲の設定
        plt.legend()         # 凡例表示の設定
        fig1 = plt.show()    # 描画実行
```

この結果，次のようなグラフが表示される．

例. グラフの表示（続き）



3.3.1.2 使用例：t 分布

“スチューデントの t 分布”として知られる t 分布[14] の確率密度関数は t.pdf である．この関数の使用例を次に示す．

例. t 分布の確率密度関数 t.pdf の実行（続き）

```
In [4]: # t分布の確率密度関数
        x1  = np.arange(-5,5,0.01)       # 定義域
        y1  = stats.t.pdf(x=x1,df=1)     # 自由度:1
        y3  = stats.t.pdf(x=x1,df=3)     # 自由度:3
        y100 = stats.t.pdf(x=x1,df=100) # 自由度:100
```

この例では，$-5 \leq x < 5$ の x に対する確率密度関数を，3 つの自由度 $k = 1, k = 3, k = 100$ に関して算出している．t.pdf 関数にキーワード引数を与えているが，x=**定義域の配列**，df=**自由度（k）**である．

これをプロットする処理が次の例である．

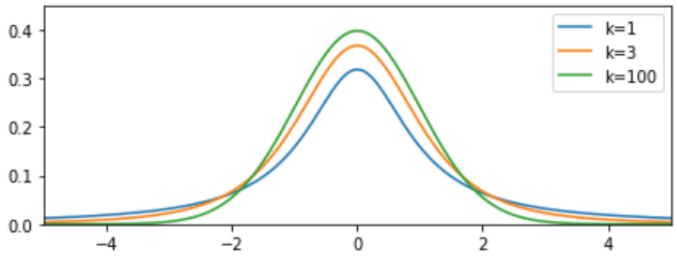
例. t 分布のプロット処理（続き）

```
In [5]: # プロット
        plt.figure( figsize=(7,2.5) )        # 描画サイズの設定
        plt.plot(x1,y1,   label='k=1' )
        plt.plot(x1,y3,   label='k=3' )
        plt.plot(x1,y100, label='k=100' )
        plt.ylim(0,0.45); plt.xlim(-5,5)     # 描画範囲の設定
        plt.legend()         # 凡例表示の設定
        fig1 = plt.show()    # 描画実行
```

この結果，次のようなグラフが表示される．

[14]詳しくは，「A.4.2 区間推定」で t 分布に関して説明している．

例. グラフの表示（続き）



3.3.1.3 使用例：χ^2 分布

χ^2 分布 [15] の確率密度関数は chi2.pdf である．この関数の使用例を次に示す．

例. χ^2 分布の確率密度関数 chi2.pdf の実行（続き）

```
In [6]: # カイ2乗分布の確率密度関数
        x1  = np.arange(0,10,0.01)       # 定義域
        y1  = stats.chi2.pdf(x=x1,df=1) # 自由度:1
        y4  = stats.chi2.pdf(x=x1,df=4) # 自由度:4
        y6  = stats.chi2.pdf(x=x1,df=6) # 自由度:6
```

この例では，$0 \leq x < 10$ の x に対する確率密度関数を，3 つの自由度 $k = 1, k = 4, k = 6$ に関して算出している．chi2.pdf 関数にキーワード引数を与えているが，x=**定義域の配列**，df=**自由度（k）**である．

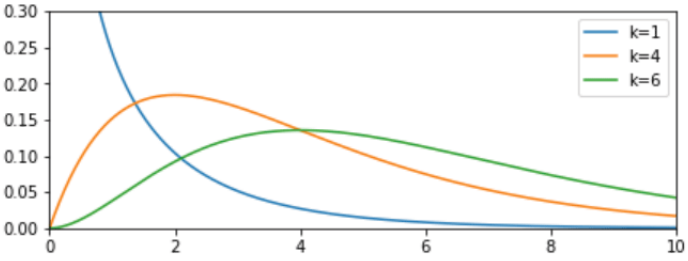
これをプロットする処理が次の例である．

例. χ^2 分布のプロット処理（続き）

```
In [7]: # プロット
        plt.figure( figsize=(7,2.5) )    # 描画サイズの設定
        plt.plot(x1,y1, label='k=1' )
        plt.plot(x1,y4, label='k=4' )
        plt.plot(x1,y6, label='k=6' )
        plt.ylim(0,0.3); plt.xlim(0,10) # 描画範囲の設定
        plt.legend()          # 凡例表示の設定
        fig1 = plt.show()     # 描画実行
```

この結果，次のようなグラフが表示される．

例. グラフの表示（続き）



3.3.1.4 使用例：対数正規分布

経済学や石油資源開発などの分野では**対数正規分布**がしばしば用いられる．この分布の定義は次の通りである．

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma\,x} \exp\left(-\frac{(\log x - \mu)^2}{2\sigma^2}\right) \qquad (x > 0)$$

scipy.stats モジュールは，対数正規分布の確率密度関数 lognorm.pdf を提供している．この関数を実行する例を示す．

[15] 詳しくは，「A.4.3 χ^2 分布」で χ^2 分布に関して説明している．

例. 対数正規分布の確率密度関数 lognorm.pdf の実行（続き）

```
In [8]: # 対数正規分布の確率密度関数
        x1 = np.arange(0.01,3,0.01)   # 定義域
        y250 = stats.lognorm.pdf(x=x1,s=2.5, loc=0)  # σ:2.5
        y150 = stats.lognorm.pdf(x=x1,s=1.5, loc=0)  # σ:1.5
        y100 = stats.lognorm.pdf(x=x1,s=1.0, loc=0)  # σ:1
        y050 = stats.lognorm.pdf(x=x1,s=0.5, loc=0)  # σ:0.5
        y025 = stats.lognorm.pdf(x=x1,s=0.25,loc=0)  # σ:0.25
        y015 = stats.lognorm.pdf(x=x1,s=0.15,loc=0)  # σ:0.15
```

この例では，$0 < x < 3$ の x に対する確率密度関数を，$\sigma = 2.5$, $\sigma = 1.5$, $\sigma = 1.0$, $\sigma = 0.5$, $\sigma = 0.25$, $\sigma = 0.15$ に関して算出している．

SciPy の lognorm.pdf 関数の定義は先に挙げた対数正規分布の定義をもう少し簡略化したものであり，次のような定義となっている．

$$lognorm.pdf(x,s) = \frac{1}{s \cdot x \cdot \sqrt{2\pi}} \cdot \exp\left\{-\frac{1}{2} \cdot \left(\frac{\log(x)}{s}\right)^2\right\}$$

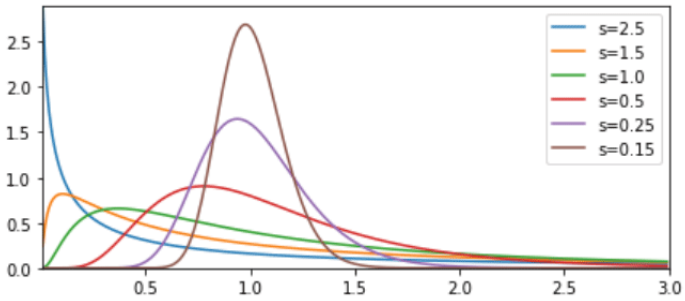
lognorm.pdf 関数にキーワード引数を与えているが，x=**定義域の配列**，s=**σ** である．更に loc=**オフセット**，scale=**スケール**というものを与えることができ，これらは処理結果を次のように変換する．

$$x_2 = \frac{x - loc}{scale} \quad \rightarrow \quad \text{計算結果：} \frac{lognorm.pdf(x_2,s)}{scale}$$

これをプロットする処理が次の例である．

例. 対数正規分布のプロット処理（続き）

```
In [9]: # プロット
        plt.figure( figsize=(7,3) )          # 描画サイズの設定
        plt.plot(x1,y250,label='s=2.5' );  plt.plot(x1,y150,label='s=1.5' )
        plt.plot(x1,y100,label='s=1.0' );  plt.plot(x1,y050,label='s=0.5' )
        plt.plot(x1,y025,label='s=0.25' ); plt.plot(x1,y015,label='s=0.15' )
        plt.ylim(0,2.9); plt.xlim(0.01,3)   # 描画範囲の設定
        plt.legend()         # 凡例表示の設定
        fig1 = plt.show()    # 描画実行
```



3.3.1.5　使用例：指数分布

単位時間内に平均して λ 回発生する確率事象があるとき，その事象が最後に発生してから次に生起するまでの時間を x とすると，時間 x 後にその事象が発生する確率の密度は**指数分布**となる．指数分布の定義は次の通りである．

$$f(x) = \begin{cases} x \geq 0 & \rightarrow \quad \lambda e^{-\lambda x} \\ x < 0 & \rightarrow \quad 0 \end{cases}$$

scipy.stats モジュールは，指数分布の確率密度関数 expon.pdf を提供している．この関数を実行する例を示す．

例. 指数分布の確率密度関数 expon.pdf の実行（続き）

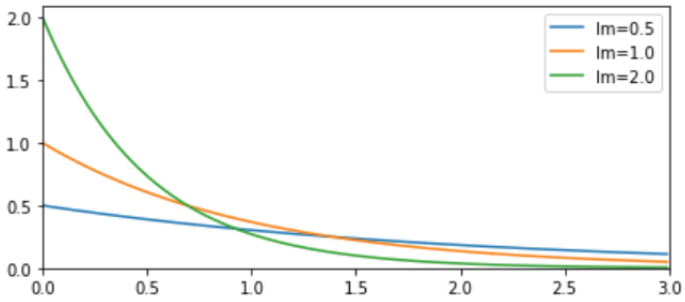
```
In [10]: # 指数分布の確率密度関数
         x1 = np.arange(0,3,0.01)   # 定義域
         y05 = stats.expon.pdf( x=x1, scale=2.0 )    # λ:1/2
         y10 = stats.expon.pdf( x=x1, scale=1.0 )    # λ:1
         y20 = stats.expon.pdf( x=x1, scale=0.5 )    # λ:1/0.5
```

この例では，$0 < x < 3$ の x に対する確率密度関数を，$\lambda = 0.5, \lambda = 1.0, \lambda = 2.0$ に関して算出している．λ の値はキーワード引数 scale=1/λ として与える．

これをプロットする処理が次の例である．

例. 指数分布のプロット処理（続き）

```
In [11]: # プロット
         plt.figure( figsize=(7,3) )     # 描画サイズの設定
         plt.plot(x1,y05,label='lm=0.5' )
         plt.plot(x1,y10,label='lm=1.0' )
         plt.plot(x1,y20,label='lm=2.0' )
         plt.ylim(0,2.1); plt.xlim(0,3)  # 描画範囲の設定
         plt.legend()         # 凡例表示の設定
         fig1 = plt.show()    # 描画実行
```



3.3.2 確率質量関数：PMF（Probability Mass Function）

確率質量関数[16] の使用例をいくつか挙げる．まず，scipy.stats モジュールと，関連パッケージ（NumPy，matplotlib）を読み込む．

例. scipy.stats，numpy，matplotlib の読込み

```
In [1]: import numpy as np       # NumPyモジュール
        from scipy import stats # SciPyのstatsモジュール
        import matplotlib.pyplot as plt # matplotlibモジュール
```

3.3.2.1 使用例：二項分布

二項分布の確率質量関数 binom.pmf の使用例を次に示す．

例. 二項分布の確率質量関数 binom.pmf の実行（続き）

```
In [2]: # 二項分布：pを0から0.9まで変えながら生成
        x1 = np.arange(0,101,1)   # 定義域
        y = [ stats.binom.pmf(x1,100,p)
             for p in np.arange(0.1,1.0,0.1) ]  # 10個の分布
```

この例では，100 回の試行に対する二項分布の確率質量関数を生成している．binom.pmf 関数の第 1 引数には 0〜100 の成功回数を表す整数の配列を与え，第 2 引数には試行回数，第 3 引数には確率を与えている．またこの例では，0.1〜0.9 までの確率毎に二項分布を 9 種類生成している．生成した 9 種類の分布をリスト y の要素として保持している．

リスト y の各要素についてプロットする例が次のものである．

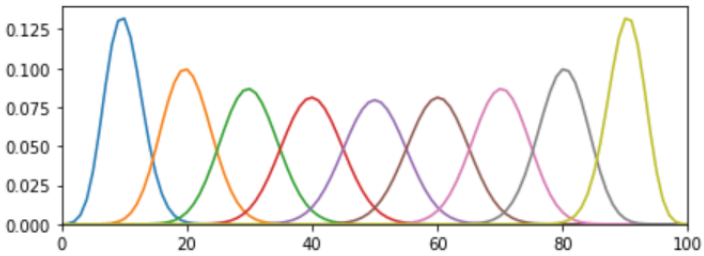
[16] 離散確率変数に対して確率を与える関数である．

例. 二項分布のプロット処理（続き）

```
In [3]: # プロット
        plt.figure( figsize=(7,2.5) )        # 描画サイズの設定
        for y1 in y:
            plt.plot(x1,y1)
        plt.ylim(0,0.14); plt.xlim(0,100)    # 描画範囲の設定
        fig1 = plt.show()    # 描画実行
```

この結果，次のようなグラフが表示される．

例. グラフの表示（続き）



3.3.2.2　使用例：幾何分布

成功確率 p のベルヌーイ試行を繰り返したとき，初めて成功するまでの回数 x の分布は**幾何分布**となる．幾何分布の定義は次の通り．

$$P(x) = p(1-p)^{x-1} \qquad \text{（定義 a）}$$

これとは別に，初めて成功するまでに失敗した回数の分布として幾何分布が定義されることもあり，その場合は

$$P(x) = p(1-p)^{x} \qquad \text{（定義 b）}$$

となる．統計解析に幾何分布を用いる際は，どちらの定義によるものかを示す必要がある．

SciPy は幾何分布の確率質量関数（先の定義 a）geom.pmf を提供しており，これの使用例を示す．

例. 幾何分布の確率質量関数 geom.pmf の実行（続き）

```
In [4]: # 幾何分布
        x1  = np.arange(1,26,1)          # 定義域
        y20 = stats.geom.pmf(x1,0.2)     # p=0.2
        y15 = stats.geom.pmf(x1,0.15)    # p=0.15
        y10 = stats.geom.pmf(x1,0.1)     # p=0.1
        y05 = stats.geom.pmf(x1,0.05)    # p=0.05
```

geom.pmf 関数の第 1 引数には，成功するまでの試行回数（配列）を与え，第 2 引数には成功確率 p を与える．この例では様々な p 毎に幾何分布を生成している．

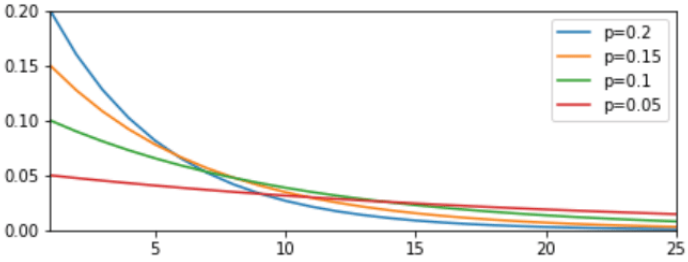
これをプロットする例を次に示す．

例. 幾何分布ののプロット処理（続き）

```
In [5]: # プロット
        plt.figure( figsize=(7,2.5) )    # 描画サイズの設定
        plt.plot(x1,y20,label='p=0.2'); plt.plot(x1,y15,label='p=0.15')
        plt.plot(x1,y10,label='p=0.1'); plt.plot(x1,y05,label='p=0.05')
        plt.ylim(0,0.2); plt.xlim(1,25) # 描画範囲の設定
        plt.legend()          # 凡例表示
        fig1 = plt.show()     # 描画実行
```

この結果，次のようなグラフが表示される．

例. グラフの表示（続き）



3.3.2.3　使用例：超幾何分布

非復元抽出による確率事象は**超幾何分布**に従うものが多い．サンプル抽出に伴い母集団の要素数は減少する（母集団が変化する）が，そのまま引き続いてサンプル抽出を行う形のものを非復元抽出と呼ぶ．非復元抽出による確率事象として現実的な事例としては「要素数が少なく，複数の当たりを含むくじ」である．例えば，抽選会などで使用される「当たり玉くじ」などが代表的な例である．この「当たり玉くじ」の状況をまとめると次のようになる．

「N 個の玉のくじがあり，K 個の当たりを含んでいる」

このようなくじから，n 個の玉を取り出した際に k 個の当たりが出る確率は超幾何分布となる．この分布の定義は次の通りである．

$$P(k) = \frac{{}_nC_k \cdot {}_{N-n}C_{K-k}}{{}_NC_K}$$

SciPy は超幾何分布の確率質量関数（先の定義 a）hypergeom.pmf を提供しており，これの使用例を示す．

例. 超幾何分布の確率質量関数 hypergeom.pmf の実行（続き）

In [6]:
```
# 超幾何分布
x1 = np.arange(0,501,1)          # 定義域
y1 = stats.hypergeom.pmf(x1,1000,100,500)
y2 = stats.hypergeom.pmf(x1,1000,200,500)
y3 = stats.hypergeom.pmf(x1,1000,300,500)
y4 = stats.hypergeom.pmf(x1,1000,400,500)
y5 = stats.hypergeom.pmf(x1,1000,500,500)
```

hypergeom.pmf 関数の第 1 引数には，くじの結果の当たりの玉の個数 k（配列）を与え，第 2 引数には玉の総数 N を，第 3 引数には取り出す玉の数 n を，第 4 引数には含まれている当たり玉の総数 K を与える．この例では様々な n 毎に超幾何分布を生成している．

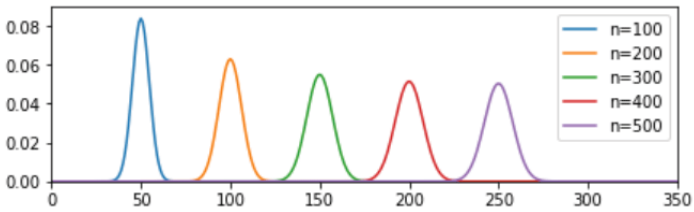
これをプロットする例を次に示す．

例. 超幾何分布のプロット処理（続き）

In [7]:
```
# プロット
plt.figure( figsize=(7,2) )    # 描画サイズの設定
plt.plot(x1,y1,label='n=100'); plt.plot(x1,y2,label='n=200')
plt.plot(x1,y3,label='n=300'); plt.plot(x1,y4,label='n=400')
plt.plot(x1,y5,label='n=500')
plt.ylim(0,0.09); plt.xlim(0,350) # 描画範囲の設定
plt.legend()          # 凡例表示
fig1 = plt.show()     # 描画実行
```

この結果，次のようなグラフが表示される．

例. グラフの表示（続き）



今回の例は 1000 個中 500 の当たりを含むくじであり，取り出す玉 n の半数のところに分布の頂点があることがわかる．

3.3.2.4　使用例：ポアソン分布

ある確率事象が定められた時間内に x 回生起する確率の分布は**ポアソン分布**となる．ポアソン分布の定義は次の通り．

$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

ポアソン分布に沿う現象の例としては，交通事故に遭う回数や，熟達したタイピストが起こすタイプミスの発生回数などが挙げられる．ポアソン分布に従う事象の特徴として，発生確率 p が小さく，十分に大きな試行回数 n の下で $np =$ **一定** となることがある．上記の関数定義において，$\lambda = np$ であり，確率変数 x は整数（発生回数）を取る．

SciPy はポアソン分布の確率質量関数 poisson.pmf を提供しており，これの使用例を示す．

例. ポアソン分布の確率質量関数 poisson.pmf の実行（続き）

```
In [8]: # ポアソン分布
        x1 = np.arange(0,111,1)          # 定義域
        y0 = stats.poisson.pmf(x1,1);  y1 = stats.poisson.pmf(x1,10)
        y2 = stats.poisson.pmf(x1,20); y4 = stats.poisson.pmf(x1,40)
        y6 = stats.poisson.pmf(x1,60); y8 = stats.poisson.pmf(x1,80)
```

poisson.pmf 関数の第 1 引数には，事象の発生回数 x（配列）を与え，第 2 引数には $\lambda = np$ を与える．この例では様々な λ 毎にポアソン分布を生成している．

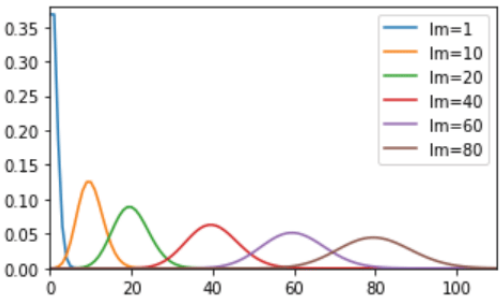
これをプロットする例を次に示す．

例. ポアソン分布のプロット処理（続き）

```
In [9]: # プロット
        plt.figure( figsize=(5,3) )    # 描画サイズの設定
        plt.plot(x1,y0,label='lm=1');  plt.plot(x1,y1,label='lm=10')
        plt.plot(x1,y2,label='lm=20'); plt.plot(x1,y4,label='lm=40')
        plt.plot(x1,y6,label='lm=60'); plt.plot(x1,y8,label='lm=80')
        plt.ylim(0,0.38); plt.xlim(0,110)  # 描画範囲の設定
        plt.legend()          # 凡例表示
        fig1 = plt.show()     # 描画実行
```

この結果，次のようなグラフが表示される．

例. グラフの表示（続き）



図の凡例 “ lm= ” は λ の値を表している．

3.3.3 累積分布関数：CDF（Cumulative Density Function）

確率密度関数 $f(x)$ を次のように積分した関数を**累積分布関数**（図 4）という．

$$cdf(x) = \int_{-\infty}^{x} f(u)\,du$$

累積分布関数は次のような性質を持つ．

$$\lim_{x\to\infty} cdf(x) = 1$$

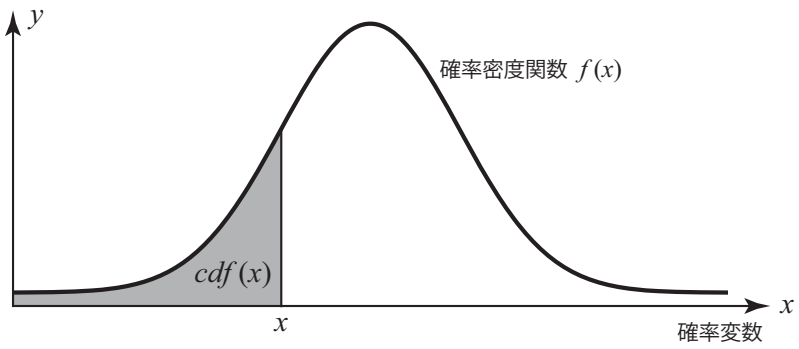


図 4: 累積分布関数

3.3.3.1 使用例：正規分布の累積分布関数

SciPy パッケージには各種の累積分布関数が提供されている．ここでは正規分布の累積分布関数を例に挙げて使用方法を紹介する．

まず，scipy.stats モジュールと，関連パッケージ（NumPy，matplotlib）を読み込む．

例. scipy.stats，numpy，matplotlib の読込み

```
In [1]: import numpy as np       # NumPyモジュール
        from scipy import stats # SciPyのstatsモジュール
        import matplotlib.pyplot as plt # matplotlibモジュール
```

正規分布の累積分布関数は norm.cdf として提供されている．この関数の引数は確率密度関数 norm.pdf と同様の形式で与える．

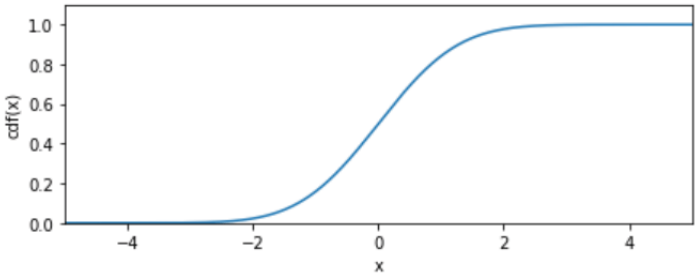
例. norm.cdf の実行（続き）

```
In [2]: # 正規分布の累積分布関数
        x1 = np.arange(-5,5,0.01)   # 定義域
        y = stats.norm.cdf(x=x1,loc=0,scale=1)
```

関数のグラフをプロットする例を示す．

例. 累積分布関数のプロット（続き）

```
In [3]: # プロット
        plt.figure( figsize=(7,2.5) )    # 描画サイズの設定
        plt.plot(x1,y)
        plt.ylim(0,1.1); plt.xlim(-5,5) # 描画範囲の設定
        plt.xlabel('x'); plt.ylabel('cdf(x)')    # 軸ラベルの設定
        fig1 = plt.show()    # 描画実行
```



norm.pdf のキーワード引数には単一の数値を与えることもできる.

例. norm.pdf の単一の値の算出（続き）

```
In [4]: stats.norm.cdf(x=0,loc=0,scale=1)

Out[4]: 0.5
```

$cdf(0)$ の値が得られている.

3.3.4 パーセント点関数：PPF（Percent Point Function）

累積分布関数の逆関数として**パーセント点関数**（図 5）がある．この関数は「累積確率が q となる確率変数 x の値」を求める場合に利用できる.

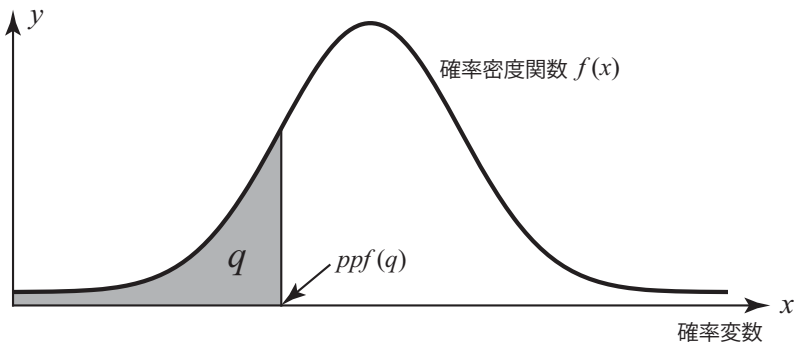


図 5: パーセント点関数

3.3.4.1 使用例：正規分布のパーセント点関数

SciPy は正規分布のパーセント点関数として norm.ppf を提供している.

例. norm.ppf の実行（続き）

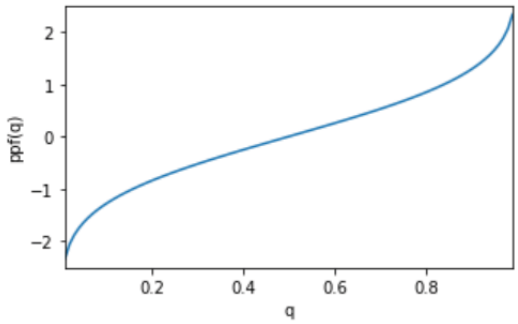
```
In [5]: # 正規分布のパーセント点関数
        q1 = np.arange(0.01,1,0.01)    # 定義域
        y = stats.norm.ppf(q=q1,loc=0,scale=1)
```

この関数にはキーワード引数 q=**累積確率** を与える．他は norm.pdf の場合と同様の形式で与える.

この関数のグラフをプロットする例を示す.

例. パーセント点関数のプロット（続き）

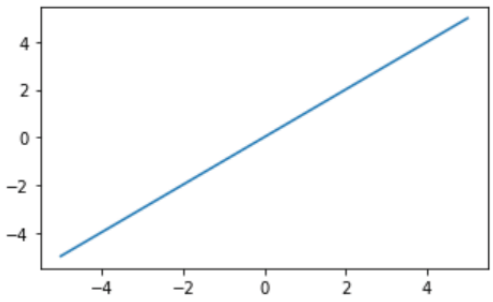
```
In [6]: # プロット
        plt.figure( figsize=(5,3) )      # 描画サイズの設定
        plt.plot(q1,y)
        plt.ylim(-2.5,2.5); plt.xlim(0.01,0.99) # 描画範囲の設定
        plt.xlabel('q'); plt.ylabel('ppf(q)')    # 軸ラベルの設定
        fig1 = plt.show()    # 描画実行
```



パーセント点関数が累積分布関数の逆関数になっていることを確かめる例を示す．次の例は，$ppf(cdf(x))$ が x に等しいことを確かめるものである．

例. $ppf(cdf(x)) = x$ の確認（続き）

```
In [7]: # 逆関数の検査
        x1 = np.arange(-5,5,0.01)    # 定義域
        y1 = stats.norm.cdf(x=x1,loc=0,scale=1) # 累積分布関数
        x2 = stats.norm.ppf(q=y1,loc=0,scale=1)   # パーセント点関数
        # プロット
        plt.figure( figsize=(5,3) ) # 描画サイズの設定
        plt.plot(x1,x2)
        fig1 = plt.show()    # 描画実行
```



データ列 x1 と x2 が同じものであることがわかる．

t 分布や χ^2 分布についても同様のパーセント点関数（.ppf）が提供されており，t 検定や χ^2 検定の際に利用できる．

3.3.5 乱数発生：RVS（Random Variates）

「3.2.1.1 正規分布に沿った乱数の生成」のところで乱数を生成する関数 rvs について説明した．rvs 関数は，正規分布以外の分布に関しても用意されている．例えば対数正規分布についても同様に rvs 関数が提供されており，これの使用例を示す．

今回も pandas パッケージを併用して，基本的な情報の表示とヒストグラムの表示の処理を行う形で例示する．

例. lognorm.rvs 関数の実行（続き）

In [8]:
```
# pandasパッケージ読込み
import pandas as pd
# 乱数生成（対数正規分布）
r = stats.lognorm.rvs(loc=0,s=1,size=1000000)
# データフレームにする
df = pd.DataFrame( columns=['対数正規分布'] )
df['対数正規分布'] = pd.Series( r )
# 情報表示
df.describe()
```

この例では，変数 r に生成した乱数を保持し，それを pandas の DataFrame オブジェクト df に変換している．この df に対して describe メソッドを使用して基本的な情報を表示（下記）している．

例. 基本的な情報の表示（続き）

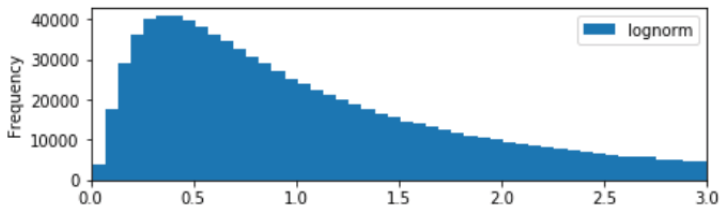
Out[8]:

	対数正規分布
count	1000000.000000
mean	1.647117
std	2.157946
min	0.007024
25%	0.509075
50%	0.998596
75%	1.961564
max	122.882772

df に対して plot メソッドを使用してヒストグラムを作成する例を次に示す．

例. ヒストグラムの表示（続き）

In [9]:
```
# ヒストグラム表示
ax = df.plot( kind='hist', y='対数正規分布',
        bins=2000, figsize=(7,2), label='lognorm',
        xlim=(0,3) )
```



本書で紹介したもの以外にも多くの関数が SciPy では利用できる．詳細についてはインターネットサイト

https://docs.scipy.org/

で公開されているドキュメントを参照のこと．

4　モデリング

4.1　多項式によるフィッティング（NumPy）

NumPy には，与えられたデータ列（1 次元配列）x, y の相関がどのような多項式に回帰するかを調べる関数 polyfit がある．ここでは例を挙げて polyfit 関数の基本的な使用方法について説明する．

まず，サンプルデータとして，1 次関数 $y = x$ に誤差（撹乱項）を加えたデータ列を作成する．

例. 1 次の多項式に回帰するサンプルデータの作成

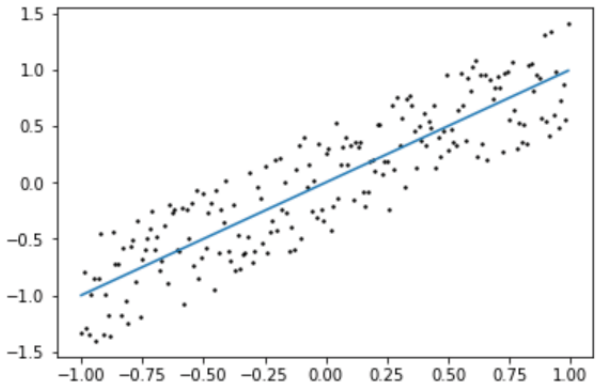
In [1]:
```
# モジュールの読込み
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
# テストデータ作成（一次）
x = np.arange(-1,1,0.01)
r = np.random.rand( len(x) )-0.5     # 撹乱項
y = x + r
```

この例では，-0.5 以上，0.5 未満の撹乱項（誤差，ノイズ）をデータ列 r として生成し，それを $-1 \leq x < 1$ の列に加えることでノイズを含んだデータ列 y を生成している．x, y の散布図を生成した例が次のものである．

例. 散布図のプロット（続き）

In [3]:
```
# 散布図の描画
fg1 = plt.scatter(x,y,s=2,color='black')
fg1 = plt.plot(x,x)
```



元の方程式（回帰方程式） $y = x$ を中心にデータが散らばっている様子がわかる．

次に polyfit によるフィッティングの例を示す．

例. polyfit によるフィッティング（続き）

In [4]:
```
# 一次式でフィッティング
p = np.polyfit(x,y,deg=1)
print( p )
```

```
[0.98517843 0.00601497]
```

polyfit 関数の第 1 引数には x 列を，第 2 引数には y 列を与え，キーワード引数 'deg=' にフィッティングする多項式の次数を与える．今回は 1 次関数にフィッティングするので deg=1 としている．polyfit では，フィッティング対象の多項式を

$$y = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_0$$

としており，実行結果として p_n〜p_0 の配列を返す．この例では $p_1 \approx 1$，$p_0 \approx 0$ となっていることがわかる．

次は 3 次の多項式へのフィッティングの例を示す．

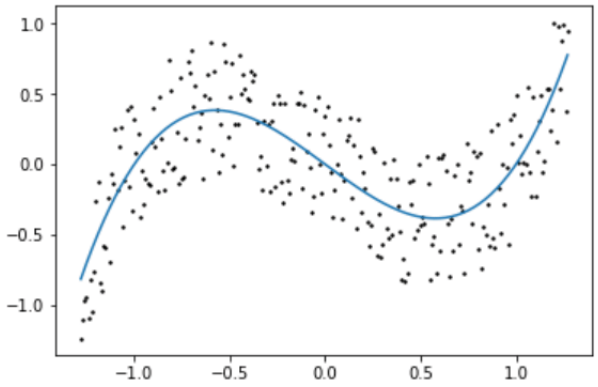
例. 3 次の多項式に回帰するサンプルデータの作成（続き）

```
In [5]: # テストデータ作成（三次）
        x = np.arange(-1.28,1.28,0.01)
        r = np.random.rand( len(x) )-0.5     # 撹乱項
        y0 = x*(x-1)*(x+1)                   # 規準の値
        y = y0 + r
```

これは，$y = x(x-1)(x+1) = x^3 - x$ に回帰するデータ列である．r は撹乱項（ノイズ）である．このデータの散布図を表示したものが次の例である．

例. 散布図のプロット（続き）

```
In [6]: # 散布図の描画
        fg3 = plt.scatter(x,y,s=2,color='black')
        fg3 = plt.plot(x,y0)
```



このデータ列 x, y を 3 次の多項式 $y = p_3x^3 + p_2x^2 + p_1x + p_0$ にフィッティングする例を次に示す．

例. polyfit によるフィッティング（続き）

```
In [7]: # 三次式でフィッティング
        p = np.polyfit(x,y,deg=3)
        print( p )
```

```
[ 1.06068549 -0.04782751 -1.03322682  0.01641095]
```

この例では $p_3 \approx 1$，$p_2 \approx 0$，$p_1 \approx -1$，$p_0 \approx 0$ となっていることがわかる．

4.2 StatsModels によるモデリング

StatsModels は統計データのモデリングのための Python 用パッケージであり，入手方法，使用方法に関する情報は，インターネットサイト https://www.statsmodels.org/ から入手できる．

4.2.1 最小二乗法による線形モデリング

データ列が，複数の既知の関数の重ね合わせとして表されるという前提で，それらの関数の係数を同定するモデリングである．

ここでは，ノイズを含んだ正弦波信号を同定する例を挙げて StatsModels の線形回帰モデリングの基本的な使用方法について説明する．

■ サンプルデータの作成

直流成分とノイズを含んだ正弦波信号を生成する例を次に示す．

例. テストデータの作成

In [1]:
```
import numpy as np                # NumPyの読込み
import matplotlib.pyplot as plt # matplotlibの読込み
import statsmodels.api as sm     # StatsModelsの読込み
```

In [2]:
```
# サンプルデータの作成
t = np.arange( 0, 5, 0.01)           # 時刻
v1 = np.sin(10*t)*3                  # 振動成分
v2 = np.repeat(2,len(t))             # 直流成分
e = (np.random.rand(len(t))-0.5)*3   # 撹乱項（ノイズ）
v3 = v1 + v2                         # 規準データ
v = v3 + e                           # 信号
```

この例では，最初に時刻のデータ列 t を生成し，それに対する振動成分を v1 に，直流成分（定数）の列を v2 に生成している．これは，

$$v_1 = 3\sin(10t),\quad v_2 = 2$$

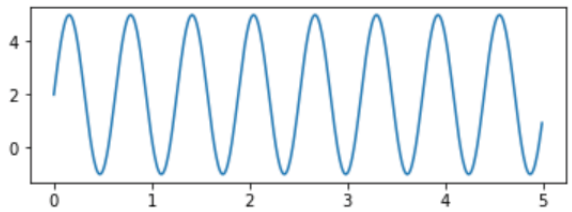
という 2 つの関数と見る．この 2 つの関数を合成したものが v3 である．すなわち，

$$v_3 = 3\sin(10t) + 2$$

である．これをプロットした例を次に示す．

例. $v_3 = 3\sin(10t) + 2$ のプロット（続き）

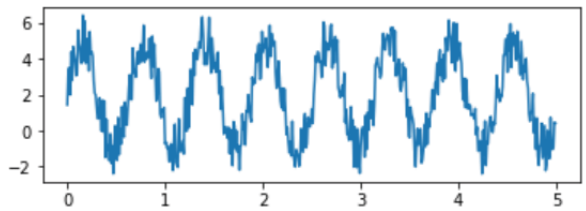
In [3]:
```
# 基準データ
plt.figure( figsize=(6,2) );    f1 = plt.plot(t,v3)
```



先に生成したデータ列 e は ±1.5 の振幅を持つホワイトノイズであり，v3 にこのノイズ成分を加えたものがデータ列 v である．これをプロットした例を次に示す．

例. ノイズを持つ信号（続き）

In [4]:
```
# 信号
plt.figure( figsize=(6,2) );    f2 = plt.plot(t,v)
```



■ 線形回帰モデリング

この v をサンプルデータとしてモデリングを行う．信号が直流成分 $m_1 = 1$ と振動成分 $m_2 = \sin(10t)$ を重ね合わせたもの $v = p_1 \times m_1 + p_2 \times m_2$ と見做してこれら p_1, p_2 を同定する．この処理のために StatsModels の OLS，GLS を用いる例を示す．

モデリングの準備として，まず信号の構成要素となる関数からなる配列を生成する必要がある．(次の例)

例. モデリング用の配列の生成（続き）

In [5]:
```
# モデリング行列の作成
m1 = np.repeat( 1, len(t) ) # 直流成分：1
m2 = np.sin( 10*t )         # 振動成分：sin(10*t)
ar = np.column_stack( [m1,m2] )
```

この例では，直流成分を m1，振動成分を m2 という配列（それぞれ 1 次元）に生成し，それらを束ねた 2 次元の配列 ar を作成している．すなわち，対象となる信号 v を ar でモデリングして係数を求める処理となる．

手順としては，**モデリング用のオブジェクト**を生成し，それに対して fit メソッドを実行する形となる．その例を次に示す．

例. モデリングの実行（続き）

In [6]:
```
# モデリング (1)
mdl1 = sm.OLS(v, ar)    # Ordinary Least Squares model
res1 = mdl1.fit()       # モデリング実行
```

この例では，モデリング用のオブジェクトとして mdl1 を生成している．この際に StatsModels の OLS 関数を使用する．この関数の第 1 引数にはサンプルデータを，第 2 引数にはモデリング用の配列を与える．fit メソッドはモデリング用オブジェクト mdl1 に対して実行する．

モデリング処理の結果が res1 に得られる．p_1, p_2 の値の配列は res1 の params プロパティに得られる．（次の例参照）

例. 同定結果の係数の表示（続き）

In [7]:
```
print( res1.params )    # 係数取り出し
```

```
[1.94994091 2.98336982]
```

この例から $p_1 \approx 2, p_2 \approx 3$ が得られていることがわかる．
更に，res1 オブジェクトに summary メソッドを実行するとモデリング処理の概要が得られる．（次の例参照）

例. モデリング処理の概要の表示（続き）

In [8]:
```
res1.summary()  # 要約情報
```

Out[8]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.860
Model:	OLS	**Adj. R-squared:**	0.860
Method:	Least Squares	**F-statistic:**	3061.
Date:	Wed, 04 Apr 2018	**Prob (F-statistic):**	8.22e-215
Time:	20:22:56	**Log-Likelihood:**	-629.96
No. Observations:	500	**AIC:**	1264.
Df Residuals:	498	**BIC:**	1272.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.9499	0.038	51.015	0.000	1.875	2.025
x1	2.9834	0.054	55.326	0.000	2.877	3.089

Omnibus:	266.221	Durbin-Watson:	2.028
Prob(Omnibus):	0.000	Jarque-Bera (JB):	29.549
Skew:	0.099	Prob(JB):	3.83e-07
Kurtosis:	1.826	Cond. No.	1.41

モデリング処理には，StatsModels の GLS 関数を使用することもできる．その例を次に示す．

例. GLS によるモデリング（続き）

In [10]:
```
# モデリング (2)
mdl2 = sm.GLS(v, ar)     # Generalized Least Squares model
res2 = mdl2.fit()        # モデリング実行
```

In [11]:
```
print( res2.params )     # 係数取り出し
```

```
[1.94994091 2.98336982]
```

In [12]:
```
res2.summary()  # 要約情報
```

Out[12]:

GLS Regression Results

Dep. Variable:	y	R-squared:	0.860
Model:	GLS	Adj. R-squared:	0.860
Method:	Least Squares	F-statistic:	3061.
Date:	Wed, 04 Apr 2018	Prob (F-statistic):	8.22e-215
Time:	20:22:56	Log-Likelihood:	-629.96
No. Observations:	500	AIC:	1264.
Df Residuals:	498	BIC:	1272.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.9499	0.038	51.015	0.000	1.875	2.025
x1	2.9834	0.054	55.326	0.000	2.877	3.089

Omnibus:	266.221	Durbin-Watson:	2.028
Prob(Omnibus):	0.000	Jarque-Bera (JB):	29.549
Skew:	0.099	Prob(JB):	3.83e-07
Kurtosis:	1.826	Cond. No.	1.41

5　免責事項

本書に掲載したプログラムリストは全て試作品であり，実用に向けた参考的資料である．また本書の使用に伴って発生した損害の一切の責任を筆者は負わない．

参考文献

[1] 松原 望, 縄田 和満, 中井検裕,
「統計学入門」東京大学教養学部統計学教室 編, 東京大学出版会, 1991

[2] 中村 勝則,
「Python3 入門」Kivy による GUI アプリケーション開発／サウンド入出力／ウェブスクレイピング, 2018
（執筆中の草稿を http://www.k-techlabo.org/www_python/python_main.pdf で公開中）

[3] 中村 勝則,
「Python3 モジュールブック」各種モジュールの基本的な使用方法, 2018
（執筆中の草稿を http://www.k-techlabo.org/www_python/python_modules.pdf で公開中）

付録

A　統計学の用語

統計解析の目的を素朴な形で表現すると,「与えられたデータの集合の特徴を調べる」ことであると言える．本書で扱ったの生徒の成績データ dat_seiseki.csv に対する作業例でも，データの特徴を Python と関連パッケージを使用して調べたことになる．ここで重要な用語として**母集団**[17]（population）と**標本**（sample）がある．母集団は調査対象の集団全体のことを指す．例えば，インターネット上に公開されている日本語の Web コンテンツの中に占める単語の出現頻度を調べる場合は,「インターネット上に公開されている日本語の Web コンテンツの全て」が母集団となる．実際の統計処理においては母集団を完全に調査することは困難であることが多く，部分的に標本を**抽出**（sampling）して調べることになる．このとき，調査の対象として取得した要素（とその属性）のことを標本と呼ぶ.

抽出された標本から算出される各種の値は，その標本のみに関するもの であり，母集団の統計的特徴とは差異がある．従って，母集団の特徴を調査するには，抽出する標本の要素の数や，抽出する頻度を多く取るといった様々な工夫をすることで**推定**（estimation）することになる．母集団の推定をする方法に関しても各種の手法がある.

実際の統計解析は，各種の**要約統計量**[18] を調べることからはじまる．主な要約統計量としては**平均値**[19]（mean），**最頻値**（mode），**中央値**,（median），**四分位数**（quartile points）があるが，これらに加えて**最大値**，**最小値**，**標準偏差**なども含める.

本書では，成績データ dat_seiseki.csv の取り扱いの例として,「成績の数値」と「成績毎の人数」の分布の調査の手順を示した．このように「ある**値**」（あるいは**階級**[20] ）に対する「その**度数**」（frequency）を調べたものが**度数分布**（frequency distribution）である．またそれを柱状のグラフにしたものを**ヒストグラム**（histogram）あるいは**度数分布図**という．度数分布の調査において，サンプルの総数に対する各階級値の出現頻度から，その階級値の出現確率 を算出することができる.

階級値毎の標本の出現確率の特徴を調べるには，それがどのような**確率分布関数**に沿っているかを考えることが基本的な作業となる.

■ 確率分布関数と確率密度関数

確率分布関数は，離散的な階級値に対する出現確率[21] を表現した関数である．これに対して，連続的な値の出現確率を考える場合には，その値がどのような**確率密度関数**に沿っているかを調べることが基本的な作業となる．確率密度関数の実用的な捉え方の 1 つとして,「ある区間において確率密度関数を定積分した値は，標本がその区間に出現する確率となる」というものがある.

重要な性質；

・ 全ての階級値（$\forall x \in X$）に対する確率分布関 $P(x)$ の値の総和は 1 になる.

$$\sum_{x} P(x) \;=\; 1$$

・ 定義域の全範囲（$-\infty$ から ∞）で確率密度関数 $f(x)$ を積分すると 1 になる.

$$\int_{-\infty}^{\infty} f(x)\,dx \;=\; 1$$

[17] 日本工業規格では,「考察の対象となる特性をもつ全てのものの集団」と定義している.
[18] **代表値**（measures of central tendency），基本統計量ともいう.
[19]「平均値」は通常**算術平均**を意味するが，他にも**調和平均**，**幾何平均**があり，それぞれ定義が異なる.
[20] 値の範囲．通常はその範囲の中央値を**階級値**（class value）とする.
[21] 厳密には**確率測度**として定義される値である.

A.1　中心極限定理と正規分布

ある母集団から無作為抽出で標本を取り出す場合，母集団の真の平均値（**母平均**）と，取り出した標本から得られた平均値の間には差異（**誤差**）がある．抽出する標本の数を多くするほど得られる平均値は母平均に近づくが，この際の誤差の分布は**正規分布**に近づく．これが**中心極限定理**である．（証明は他の文献に譲り割愛する）

正規分布の定義：

正規分布 $f(x)$ は確率密度関数であり，定義は次の通り．

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \qquad (x \text{ は実数})$$

ここで μ は $f(x)$ の**平均**（**期待値**），σ は**標準偏差**である．$f(x)$ のプロットを図 6 に示す．

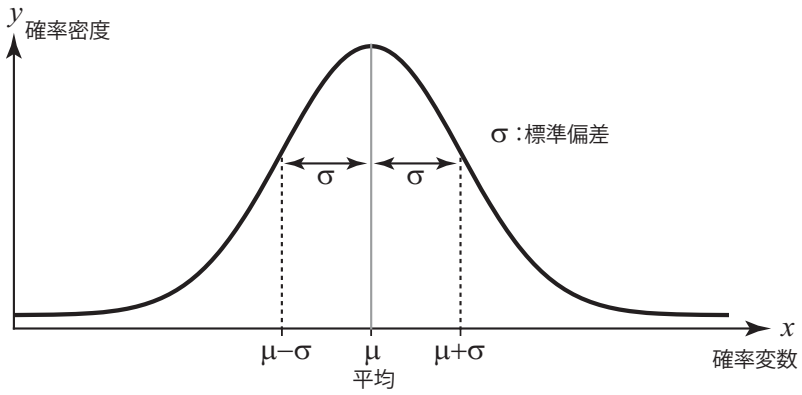


図 6: 正規分布

A.2　尖度，歪度

正規分布よりも「尖った」分布（図 7）を「**尖度**（kurtosis）が大きい」と表現する．

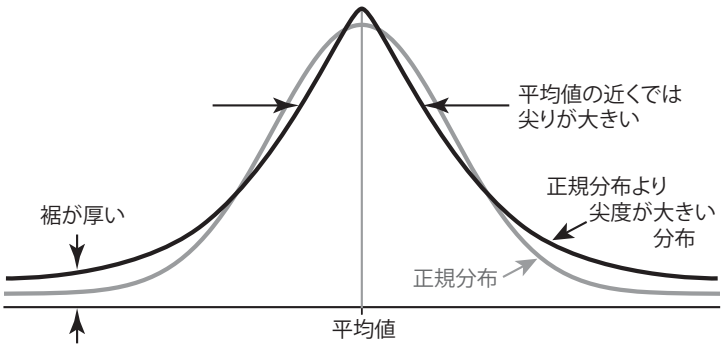


図 7: 尖度の大きい分布

尖度は正規分布を基準とし，正規分布の尖度を 0 [22] とする．尖度の大きい分布では，山の近くでは尖りが大きく，裾が厚い形となる．

確率変数の左右（大小）で非対称な分布は「**歪度**（skewness）が大きい」と表現する．左右対称な分布は歪度が 0 である．例えば図 8 に示す対数正規分布は歪度が大きい分布である．

対数正規分布：　$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma\,x} \exp\left(-\frac{(\log x - \mu)^2}{2\sigma^2}\right) \qquad (x > 0)$$

[22] 正規分布の尖度を 3 とする文献もあるので注意すること．

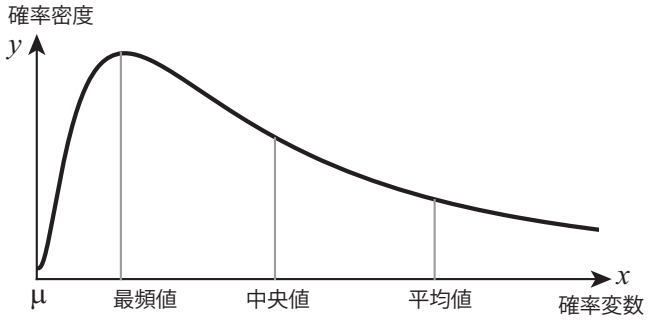


図 8: 対数正規分布
注）確率密度関数を構成する μ, σ と，結果としての
平均値（期待値），標準偏差は異なる．

対数正規分布では平均値，中央値，最頻値が全て異なる値となり，結果として μ，σ も「平均」（期待値）,「標準偏差」とは異なる意味を持つ．

【μ，σ が意味するもの】

統計調査で用いる確率分布関数（確率密度関数）の多くのものが，正規分布に変形，変換を施したものとして解釈できる．その意味では正規分布は確率分布関数のもっとも基本的なものであるということができる．実際に，多くの確率密度関数が μ，σ を用いて記述される．ただし，正規分布以外の確率密度関数においては，結果的な平均値と標準偏差は μ，σ とは必ずしも同じものとはならないことに注意すること．先の対数正規分布で μ と平均値が異なることが 1 つの例である．

注意：

実際の統計学の文献や調査報告において「μ」,「平均値」,「期待値」の用語の意味が統一されていないことがあるので注意すること．特に期待値 $E(X)$ は「標本（あるいは確率変数）の確率の重みを付けた合計」であり，確率変数 $\{x_1, x_2, \cdots, x_n\} \in X$ に対する確率（確率密度ではない）を $f(x_i)$ とすると次のように定義される．

$$E(X) = \sum_i x_i \cdot f(x_i) \qquad \text{（離散分布の場合）}$$

期待値を実質的な意味で「平均値」として取り扱う文献もあるので留意しておくこと．

尖度，歪度は「**モーメント**[23]（moment）」を用いて計算する．

■ α 周りの n 次のモーメント

基準となるある値 α を考え，平均値（期待値）の計算を E と書き，

$$E\,(X-\alpha)^n$$

を,「標本集合 X の α 周りの n 次のモーメント」と定義する．更に $E\,(X)^n$ を「原点周りの n 次のモーメント」[24] という．

平均値周りの n 次のモーメントを m_n と書くと，尖度は $\dfrac{m_4}{\sigma^4} - 3$

として定義される．(この定義では正規分布の尖度は 0 となる）また歪度は $\dfrac{m_3}{\sigma^3}$

として定義される．対数正規分布では歪度が正の値を取り，これは「右の裾が長い」ことを意味する．逆に歪度が負の値を取る分布は「左の裾が長い」形となる．

モーメントを用いると，平均値は原点周りの 1 次のモーメントとして，また分散は平均値周りの 2 次のモーメント m_2 として定義できる．

---

[23] **積率**ともいう．

[24] **参考）** $E\left(\dfrac{X-\mu}{\sigma}\right)^n$ を**標準化された**モーメントという．

A.3　標本の抽出

単純無作為抽出（単純ランダムサンプリング）で抽出された標本から統計量を算出する場合について考える．抽出した標本の集合を X と書き，標本の数を n とすると，**標本平均** $\bar{X}$ は次のような式で書く．

$$\bar{X} = \frac{x_1 + x_2 + \cdots + x_n}{n} \qquad (x_1, x_2, \cdots, x_n \in X)$$

標本平均は，**母平均**（母集団の平均）μ とは異なり，誤差を含んだ値である．標本平均の期待値 $E(\bar{X})$ は母平均 μ に等しい．例えば，同一の母集団に対して複数回の統計調査（ランダムサンプリング）を行い，各回の $\bar{X}$ の平均を取るとその値は母平均 μ に近づくという事実がこれに当たる．もちろんこの行為は n を母集団の数 N に近づけることに等しく，$E(\bar{X}) = \mu$ となることが理解できる．

A.3.1　標本分散と不偏分散

採取した標本から求めた分散は**標本分散**と呼ばれ，次の式で定義される．

$$S^2 = \frac{(x_1 - \bar{X})^2 + (x_2 - \bar{X})^2 + \cdots + (x_n - \bar{X})^2}{n}$$

標本分散は母分散（母集団の分散）σ^2 とは異なり，誤差を含んだ値である．次に，**不偏分散** s^2 というもの（次の式）を定義する．

$$s^2 = \frac{(x_1 - \bar{X})^2 + (x_2 - \bar{X})^2 + \cdots + (x_n - \bar{X})^2}{n-1}$$

分母を $n-1$ とする不偏分散を定義する理由としては， S^2 の期待値 $E(S^2)$ を算出すると，

$$E(S^2) = \frac{n-1}{n} \cdot \sigma^2$$

となるということがある．このことは，s^2 が S^2 に比べて**不偏性** [25] があり，σ^2 に近い形に補正されたものであることを意味する．実際の統計処理において，分散として s^2 が採用されることが多い．

S を**標本標準偏差**，s を**不偏標準偏差**と呼ぶ．

参考．

実際の統計処理では，計算の簡便性のために標本の値の 2 乗の平均を求めておくことがある．これにより，標本分散 S^2 を次の形で求めることができる．

$$S^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{X}^2$$

（証明）

$$S^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{X})^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^2 - 2x_i\bar{X} + \bar{X}^2\right) = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2\bar{X}\cdot\frac{1}{n}\sum_{i=1}^{n} x_i + \bar{X}^2 \cdot \frac{1}{n}\sum_{i=1}^{n} 1$$

$$= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2\bar{X}\cdot\bar{X} + \bar{X}^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2\bar{X}^2 + \bar{X}^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{X}^2$$

A.4　推定

統計調査が対称とする集団を**母集団**（population）という．実際の統計調査では，母集団から**標本**を**抽出**して，標本に関する各種統計量を算出する．

母集団の統計量に関する推定には次の 2 種類の場合がある．1 つは母集団の分布がどのような確率分布に沿っているかが予め分かっている場合である．この場合は，母集団の分布を決めている確率分布の基本的な値 μ, σ などを推定する．このような母集団が持つ統計量を**母数**（parameter）という．これには母集団の平均である**母平均** μ や，母集団の分散である**母分散** σ^2 などが含まれる．このような，母集団の分布が予め分かっているという前提での推定を

[25] **不偏性**と**一致性**に関する解説は他の文献を参照のこと．

パラメトリック（parametric）な推定という．
2 つ目は，母集団が従う確率分布が予め分からない場合である．このような場合は，得られた標本から平均，中央値，最頻値，尖度，歪度といった各種の代表値を算出して，統計処理を更なる段階へと進めるための判断材料とする．このような，母集団の分布が予め分かっていない前提での推定処理を**ノン・パラメトリック**（non-parametric）な推定という．

以上のように，母集団から抽出した標本を元にして母数を近似的に**推定**[26]（estimation）することになるが，抽出した標本から得られた統計量は，母数を推定するための**推定量**（estimator）であるという．これには，標本から算出した**標本平均**や**標本分散**などが含まれる．推定処理を行うために標本から各種の値を算出するが，それらをまとめて**統計量**（statistic）と呼ぶ．

A.4.1 点推定

統計調査において十分な数の標本が採取できる場合は**点推定**と呼ばれる方法で母数を推定することがある．点推定では，母数（母集団が持つ各種統計量）の各値を 1 通りに定める．点推定とは別に，母数の各値を「ある範囲にある」として推定する方法を**区間推定**といい，点推定とは区別する．（区間推定については後で説明する）

最も単純な点推定の方法としては，得られた標本の統計量をそのまま母数として採用するものがある．例えば n 個の標本の集合 X から算出した標本平均と不偏分散を，それぞれ母平均 μ の推定値 $\hat{\mu}$，母分散 σ^2 の推定値 $\hat{\sigma}^2$ とみなすものである．すなわち，

$$\hat{\mu} = \frac{x_1 + x_2 + \cdots + x_n}{n}, \quad \hat{\sigma}^2 = \frac{(x_1 - \bar{X})^2 + (x_2 - \bar{X})^2 + \cdots + (x_n - \bar{X})^2}{n - 1}$$

とする．このように，母数として推定された値（推定値）にはハット ‘ˆ’ を付けて表記する．

A.4.1.1 最尤法（Maximum likelihood estimation）

成功する（実現値が 1 である）確率が p，失敗する（実現値が 0 である）確率が $1 - p$ であるベルヌーイ試行を考え，この場合の母数 p を最尤法で推定する方法を例を挙げて説明する．この試行を 5 回行った結果，実現値（標本）が次のようになったとする．

$$1, \quad 1, \quad 1, \quad 0, \quad 1$$

すなわち，標本抽出の結果,「4 回の成功，1 回の失敗」という標本が得られたことになる．この事象が発生する確率は $p^4(1 - p)$ となる．以上のことを元にして最尤法で p を推定する．最尤法では「最も確率の高い事象が発生した」と仮定して母数を推定する．この仮定を**最尤原理**（principle of maximum likelihood）と呼ぶ．今回の例では，最尤なる p を探すために，事象の発生確率の関数 $L(p)$ を定義してこれが最大となる p を探す方法を取る．すなわち，

$$L(p) = p^4(1 - p)$$

が最大値を与える p を探す．この例では $0 \leq p \leq 1$ の範囲で母数 p を探すが，想定される母数の範囲（集合）を**母数空間**（parameter space）という． $L(p)$ のプロットを図 9 に示す．

今回の場合，$\frac{d}{dp}L(p) = p^3(4 - 5p)$ となり，これが 0 となりかつ $L(p)$ が極大となる点，すなわち $\hat{p} = 0.8$ が推定値となる．

（最尤法に関する今回の例は文献 [1] を参考にしている）

A.4.2 区間推定

実際の統計調査においては，母集団の要素を全て調査することは困難であることが多く，母集団よりも小さなサイズの標本を抽出して母数を推定する．ここでは，抽出した標本の統計量がどの程度の確からしさで母集団の母数を表

[26] これは統計学の用語である．統計調査の結果などを用いて各種の事象に関して推論することを**推測**（surmise）というが，「推定」という語とは異なる語であることを意識すること.「推定」とは母数を求めるという限定的な意味を持つ.

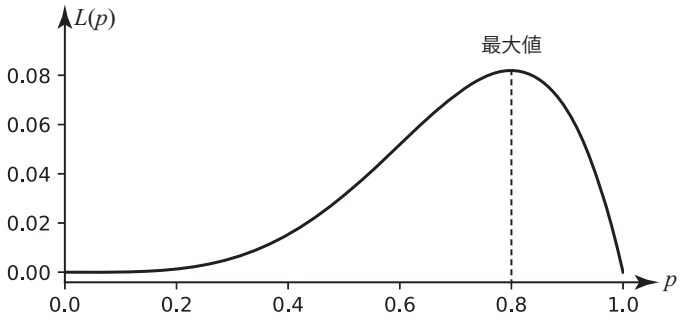


図 9: $L(p)$ のプロット

しているかについて考える．具体的には「ある母数がある範囲内に含まれる条件」を求め，抽出した標本がその条件を満たしているかを判定して，標本の統計量の信憑性について評価する．

A.4.2.1　信頼区間（confidence interval）

ある母数 θ がある区間 $[L, U]$ に $1-\alpha$ の確率（確からしさ／信頼性）で含まれるとする．例えば，抽出した標本の統計量が「〜%の信頼性で母数を表している」という評価を行う場合に $1-\alpha$ を満たす L と U を定める．これは，抽出した標本がどの程度の確からしさで母数を表しているかを判定する基準にもなり，信頼できる統計調査のための標本数の策定などに応用できる．この $[L, U]$ を「母数 θ の確率 $1-\alpha$ の**信頼区間**（confidence interval）」という．また，$1-\alpha$ を**信頼係数**（confidence coefficient）と呼び，L，U をそれぞれ**下側信頼限界**（lower confidence limit），**上側信頼限界**（upper confidence limit）と呼ぶ．

ここでは，正規分布に従う母集団（**正規母集団**）において信頼区間を求める方法について説明するが，そのための準備として，正規母集団からの標本抽出に関して基礎的な内容について説明しておく．

I. **標準正規分布**

平均が μ，分散が σ^2 の正規分布を $N(\mu, \sigma^2)$ と書き，平均が 0，分散が 1 の正規分布，すなわち $N(0,1)$ を**標準正規分布**という．$N(0,1)$ は確率変数 x の関数 $\phi(x)$ と書かれることがある．すなわち，

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

である．これは $N(\mu, \sigma^2)$ の正規分布

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

を，**標準化変数** $x' = \dfrac{x-\mu}{\sigma}$ を用いて書き換えたもの ($f(x')$) と見ることができる．

II. **誤差関数**（ガウスの誤差積分）

標準正規分布 $\phi(x)$ に従う事象において，ある z に対して $x \leq z$ の範囲の標本が得られる確率は，$\phi(x)$ の累積分布関数

$$\Phi(z) = \int_{-\infty}^{z} \phi(x)\, dx$$

を用いて表される．これを**誤差関数**（ガウスの誤差積分）と呼び，$erf(z)$ と記す．

III. **パーセント点**

$\phi(x)$ に従う事象において，ある x よりも大きな確率変数に対応する標本の発生確率が α であるとすると，その x の点を Z_α と書き，これを「発生確率が α の**パーセント点**」と呼ぶ．(図 10)

α と Z_α の間には $\alpha = 1 - \Phi(Z_\alpha)$ という関係があり，これを解くことでパーセント点を求めることができる．

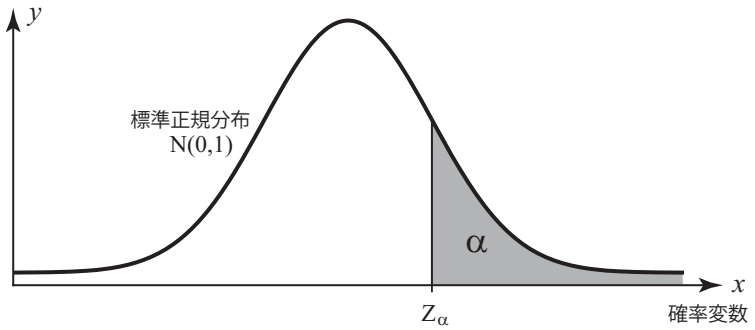


図 10: 発生確率が α のパーセント点 Z_α

【信頼区間の算出】

考え方 1.

パーセント点の考え方を応用して信頼区間を算出することができる．例えば正規母集団（簡単のため，標準正規分布に従うとする）から採取した標本が，平均値 0 を中心とする信頼係数 $1-\alpha$ の範囲内にある区間は図 11 における $\left[-Z_{\frac{\alpha}{2}}, Z_{\frac{\alpha}{2}}\right]$ である．

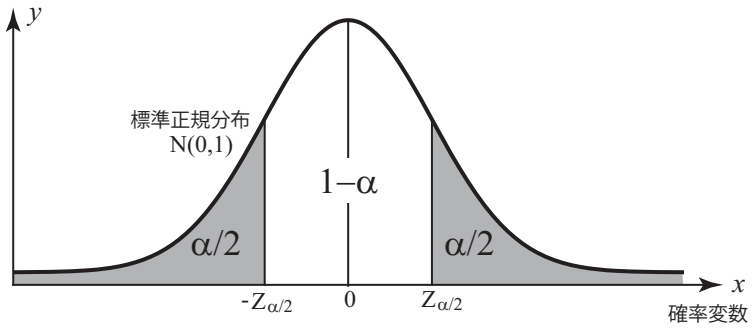


図 11: 信頼区間の作り方

考え方 2.

正規母集団 $N(\mu, \sigma^2)$ から n 個の標本を取り出したときの標本平均を $\bar{X}$ とする．この $\bar{X}$ は本当の平均値（母平均） μ からある誤差を持って離れており，$\bar{X}$ は別の正規分布 $N\left(\mu, \dfrac{\sigma^2}{n}\right)$ に従う．[27]　このことから，$\bar{X}$ が信頼係数 $1-\alpha$ で $N\left(\mu, \dfrac{\sigma^2}{n}\right)$ に重なる確率は（標準化された形で）

$$P\left(-Z_{\frac{\alpha}{2}} \leq \frac{\sqrt{n}(\bar{X}-\mu)}{\sigma} \leq Z_{\frac{\alpha}{2}}\right) = 1-\alpha$$

と表現される．更に μ が明にわかる形に変形すると，

$$P\left(\bar{X} - Z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{X} + Z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}}\right) = 1-\alpha$$

となり，信頼区間は，

$$\left[\bar{X} - Z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}}, \bar{X} + Z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}}\right]$$

となる．

考え方 3.

母集団が持つ真の分散（母分散）は未知であることが多く（従って母集団の標準偏差 σ も未知），上で示した信頼区間の式をそのまま用いることができる場合は少ない．そこで実際の統計調査では標本から得られた**不偏分散** s^2 を何らかの形で利用することになる．ただし，標本の不偏分散をそのまま母分散とするべきではなく，ここで t **統計量**を

[27]「A.1 中心極限定理と正規分布」を参照のこと．

導入して信頼区間を決める．

■ t 統計量，t 分布，標準誤差について

n 個の標本 X の分布が正規分布 $N(\mu,\sigma^2)$ に従うとする．このとき，

$$Z = \frac{\bar{X}-\mu}{\sigma/\sqrt{n}}$$

は標準正規分布 $N(0,1)$ に従う．この Z の σ を s で置き換えた t **統計量**

$$t = \frac{\bar{X}-\mu}{s/\sqrt{n}}$$

を導入する．t 統計量は t **分布**に従う．

● t 分布

$N(0,1)$ に従う Z と $\chi^2(k)$ に従う Y があり [28]，Z と Y は独立であるとするとき，次の t が従う分布を**自由度** k **の** t **分布** [29] といい，$t(k)$ と書く．

$$t = \frac{Z}{\sqrt{Y/k}}$$

先の t 統計量 $t = \dfrac{\bar{X}-\mu}{s/\sqrt{n}}$ は自由度 $n-1$ の t 分布 $t(n-1)$ に従う．

また，t 統計量の分母である $s/\sqrt{n}$（$\bar{X}$ の標準偏差）を「標本平均の**標準誤差**」という．

t 分布は標本数が小さい場合に「正規分布の代用品」として使用されることがある．すなわち，抽出した標本数が n の場合，標本平均 $\bar{X}$ は自由度 $n-1$ の t 分布 $t(n-1)$ に従うとする．

t 分布のパーセント点も正規分布のパーセント点と同様に定義され，$t_\alpha(k)$ と書く．実際の統計調査では，t 分布のパーセント点を用いて，母数の信頼区間を

$$\left[\bar{X}-t_{\frac{\alpha}{2}}(n-1)\cdot\frac{s}{\sqrt{n}}\,,\ \bar{X}+t_{\frac{\alpha}{2}}(n-1)\cdot\frac{s}{\sqrt{n}}\right]$$

とすることがある．

参考）実際の統計調査においては信頼係数は 99%（$1-\alpha=0.99$）や 95%（$1-\alpha=0.95$）といった値が採用されることが多い．

A.4.3 χ^2 分布

$N(0,1)$ に従う k 個の独立な確率変数 $Z_1, Z_2, \cdots, Z_k$ があるとき，

$$\chi^2 = Z_1^2 + Z_2^2 + \cdots + Z_k^2$$

が従う分布を χ^2 **分布**という．特に変数の個数 k によって「**自由度が** k **の** χ^2 **分布である**」という．
確率変数 x に対する χ^2 分布の確率密度関数 $f(x)$ は，ガンマ関数を用いて次のように定義される．

$$f(x) = \frac{x^{\frac{k}{2}-1}}{2^{\frac{k}{2}}\,\Gamma\left(\frac{k}{2}\right)}\exp\left(-\frac{x}{2}\right)$$

自由度 k の χ^2 分布を $\chi^2(k)$ と書く．

A.4.4 t 分布（スチューデントの t 分布）

自由度 k の t 分布の確率密度関数 $t(x)$ 定義は次の通り．

[28]「A.4.3 χ^2 分布」を参照のこと．
[29]「A.4.4 t 分布」を参照のこと．

$$t(x) = \frac{\Gamma\left(\frac{k+1}{2}\right)}{\sqrt{k\,\pi}\,\Gamma\left(\frac{k}{2}\right)} \left(1 + \frac{x^2}{k}\right)^{-\frac{k+1}{2}}$$

A.5 仮説検定

仮説検定[30]（hypothesis testing）とは，立てた仮説の真偽を統計的手法で評価することを意味する．ここでいう仮説検定は論理学的な証明とは異なり，“確からしさ”や “疑わしい”ということを統計調査を元にして，ある規準（**有意水準**）を設けて判定する行為を指す．

例. コインを 20 回投げる試行における表が出る回数に関する仮説検定[31]

コインを 20 回投げて，表側が 14 回出たとする．このとき「表と裏の出る確率は等しくない」という仮説を統計的に検定するために,「表と裏の出る確率は等しい」という仮説を設定し，この仮説を統計的データを用いて**棄却**（reject）するという方法（背理法）を取ることにする．このように，棄却したい仮説を**帰無仮説**（null hypothesis）と呼び，それに対する仮説を**対立仮説**（alternative hypothesis）と呼ぶ．帰無仮説，対立仮説という名称には特に意味はなく，検定作業をする状況に応じて設定される「互いに逆になっている命題」である．

今回の例では,

帰無仮説：「表と裏の出る確率は等しい」

対立仮説：「表と裏の出る確率は等しくない」

と設定する．また今回の試行は「確率 p の事象が n 回の試行で x 回発生する」と言い換えることができ，この x は二項分布 $Bi(n,p)$ に従う．この分布の確率密度とその累積値を表 2 に示す．これを元に検定の作業を進める．

表 2: $Bi(20, 1/2)$ の分布

x	値	累積値
0	9.54×10^{-7}	9.54×10^{-7}
1	1.91×10^{-5}	2.00×10^{-5}
2	0.000181198	0.000201225
3	0.001087189	0.001288414
4	0.004620552	0.005908966
5	0.014785767	0.020694733
6	0.036964417	0.057659149
7	0.073928833	0.131587982
8	0.120134354	0.251722336
9	0.160179138	0.411901474
10	0.176197052	0.588098526
11	0.160179138	0.748277664
12	0.120134354	0.868412018
13	0.073928833	0.942340851
14	0.036964417	0.979305267
15	0.014785767	0.994091034
16	0.004620552	0.998711586
17	0.001087189	0.999798775
18	0.000181198	0.999979973
19	1.91×10^{-5}	0.999999046
20	9.54×10^{-7}	1

$Bi(20, 1/2)$ の分布において，コインの表側が 14 回以上出る確率 $P(x \geq 14)$ を表 2 から求める．コインの表側が 13 回以下の回数で出る確率 $P(x \leq 13)$ は 0.942340851 であることから，$P(x \geq 14)$ は $1 - 0.942340851 = 0.057659149$ となる．これは発生確率としては低いと見做され，今回の試行のように，コインを 20 回投げて表側が 14 回出たことで,「表と裏の出る確率は等しい」（$p = 1/2$）という仮説は棄却され，逆の対立仮説が**採択**（accept）される．

[30] より厳密に**統計的仮説検定**ともいう．

[31] この例は参考文献 [1] からの引用である．

A.5.1　有意水準と誤りについて

棄却／採択のための規準を**有意水準**（significance level）といい，通常，記号 α で記される．先の例において $\alpha = 0.1$ と設定すると，0.057659149 という数値は α を下回っており，帰無仮説が棄却されるが，$\alpha = 0.01$ と設定すると帰無仮説は棄却されない．（帰無仮説のずれが有意ではないとする）

有意水準は統計調査の意向（厳格さなど）によって設定されるが，この設定が適切でないと，棄却／採択の判定に**誤り**が起こる．この場合の誤りには 2 種類のものがあり，「帰無仮説として設定された命題が正しいにもかかわらず棄却してしまう」誤りを**第一種の誤り**，「誤った帰無仮説を採択してしまう」誤りを**第二種の誤り**という．

A.5.2　母平均に関する検定（t 検定）

実際の統計調査では十分に大きな標本数が得られないことが多く，母数が従う分布として正規分布の替わりに t 分布を用いることがあると先に述べた．ここでは t 統計量を使用して母平均に関して検定する t **検定**について，例を挙げて説明する．

例. $N(\mu, \sigma^2)$ に従う正規母集団がある．ここから $n = 25$ の数の標本を抽出して $\bar{X} = 13.7$，$\sigma = 2.3$ を得た．この状況で次のような仮説を立てて検定する．

帰無仮説 H_0：「母平均は 15 である」（ $\mu = 15$ ）
対立仮説 H_1：「母平均は 15 でない」（ $\mu \neq 15$ ）

有意水準は $\alpha = 0.05$ とする．[32]

まず，$\mu = 15$ とする t 統計量を求めると，

$$t = \frac{13.7 - 15}{2.3/\sqrt{25}} = -2.826\cdots$$

となる．この例の対立仮説は $\mu < 15 \vee 15 < \mu$ と同値なので，$\alpha = 0.05$ となる $t(24)$ のパーセント点を求めると，

$$t_{\frac{0.05}{2}}(24) = 2.06389856\cdots$$

となり，先の t 統計量の絶対値はこの値を越えるので，帰無仮説は棄却される．

この検定では，求めた t 統計量が図 12 の t 分布における両端の部分（グレーの部分）に属していることから帰無仮説を棄却したということができる．この図のグレーの部分を**棄却域**（rejection region）といい，それ以外の部分を**採択域**（acceptance region）という．

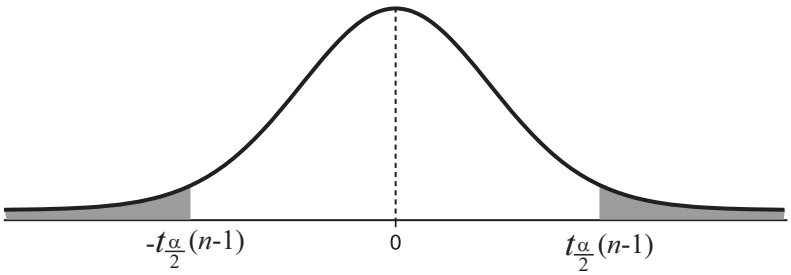


図 12: 両側検定の棄却域（t 検定）

両側検定

ここに示した t 検定では，帰無仮説の t 統計量が t 分布のパーセント点より遠い領域（0 と比べて）にあることで棄却する．また μ を別の値に設定する帰無仮説を立てた際，その t 統計量が t 分布のパーセント点より原点に近い場合は帰無仮説を採択することになる．このように，t 分布における両端の部分（グレーの部分）に属していることで帰無仮説を棄却する検定方法を**両側検定**（two-sided test）という．今回設定した対立仮説のように $\mu \neq 15$ という命題は，$\mu < 15$ という命題と $\mu > 15$ という命題の論理和であり，両側検定を適用するケースである．また今回のような

[32] この例は参考文献 [1] からの引用である．

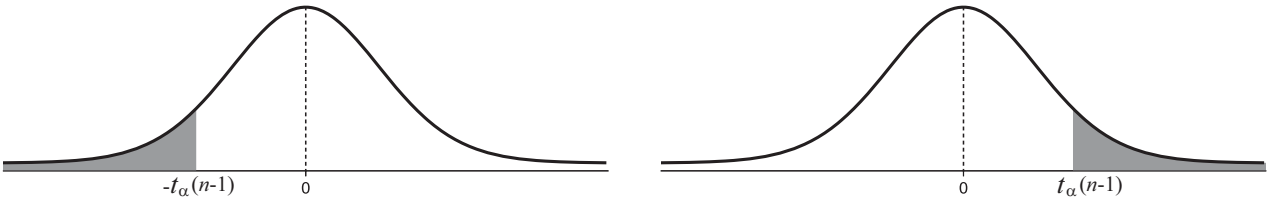
対立仮説を**両側対立仮説**（two-sided alternative hypothesis）という．

片側検定

帰無仮説 $\mu = 15$ に対して対立仮説 $\mu < 15$ を設定して検定すると，棄却域は図 13 の (a) の領域となる．この場合のパーセント点を求めると，

$$t_{0.05}(24) \;=\; 1.710882\cdots$$

となり，この値は未だ帰無仮説の t 統計量の絶対値より大きい，従ってこの設定でも帰無仮説は棄却される．



(a) 左側対立仮説における棄却域　　(b) 右側対立仮説における棄却域

図 13: 片側検定の棄却域

次に，対立仮説として $\mu > 15$ を設定して検定すると棄却域は図 13 の (b) の領域となる．この場合は帰無仮説の t 統計量は採択域に入り，帰無仮説は棄却されない．この結論は「$\mu > 15$ という仮説よりは $\mu = 15$ という仮説の方がより確からしい」と解釈される．

この例では，$\mu = 15$ に対して，$\mu < 15$ や $\mu > 15$ といった対立仮説を立てたが，これらをそれぞれ**片側対立仮説**（one-sided alternative hypothesis）といい，これによる検定を**片側検定**という．

片側検定においては，帰無仮説と対立仮説は論理的には互いに逆の関係になっていないことがあるので注意しなければならない．

A.5.3　母分散に関する検定（χ^2 検定）

母分散 σ^2 がある値 σ_0^2 と等しいかどうかを検定するには，帰無仮説 $\sigma^2 = \sigma_0^2$ に関する下記の統計量を用いる．

$$\chi^2 = \frac{(n-1)\,s^2}{\sigma_0^2}$$

すなわち，これが標本分散 s^2 によって自由度 $(n-1)$ の χ^2 分布に従うという性質を利用する．

【手順】

1. 有意水準 α を定める．

2. χ^2 分布のパーセント点を求めて棄却か採択かを判定する．

両側検定の場合は $\chi^2_{1-\frac{\alpha}{2}}(n-1)$ と $\chi^2_{\frac{\alpha}{2}}(n-1)$ の 2 つのパーセント点の値を求め，

$$\chi^2_{1-\frac{\alpha}{2}}(n-1) \;<\; \chi^2 \;<\; \chi^2_{\frac{\alpha}{2}}(n-1)$$

の場合は帰無仮説を採択し，そうでなければ棄却する．

対立仮説が $\sigma^2 > \sigma_0^2$ の場合は $\chi^2_\alpha(n-1) \le \chi^2$ による**右片側検定**で帰無仮説を棄却し，対立仮説が $\sigma^2 < \sigma_0^2$ の場合は $\chi^2 \le \chi^2_{1-\alpha}(n-1)$ による**左片側検定**で帰無仮説を棄却する．

このような検定方法を χ^2 **検定**という．

例. ある学力考査では，例年平均点は 50 点，分散 36 であった．本年度もこの学力考査を実施し，ランダムサンプリングで受験者 25 人の成績を抽出したところ，平均点は 53 点，分散 48 が得られた．本年度の受験者の学力の散らばりは例年よりも大きいと見るべきかどうかを

帰無仮説 $H_0 : \sigma^2 = 36$

帰無仮説 $H_1 : \sigma^2 \neq 36$

有意水準 $\alpha = 0.1$

として検定する.

χ^2 統計量を算出すると,

$$\chi^2 = \frac{(25-1)\cdot 48}{36} = 32$$

となる. 次に χ^2 分布のパーセント点を求めると,

$$\chi^2_{0.95}(24) = 13.8484\cdots$$
$$\chi^2_{0.05}(24) = 36.4150\cdots$$

となる. (図 14 参照)

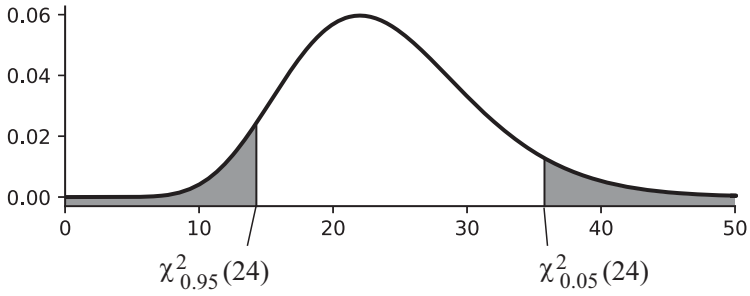


図 14: 両側検定の棄却域 (χ^2 検定)

従って, 帰無仮説の χ^2 統計量は採択域にあり, 棄却されない.

B 各種パッケージが提供する関数

B.1 scipy.stats

表 3: 各種の分布に関連する関数

分布の名称	クラス	確率密度／確率質量	説明
正規分布	norm	pdf(x=確率変数, loc=μ, scale=σ)	μ は平均，σ は標準偏差
t 分布	t	pdf(x=確率変数, df=自由度)	
χ^2 分布	chi2	pdf(x=確率変数, df=自由度)	
対数正規分布	lognorm	pdf(x=確率変数, s=標準偏差, loc=オフセット, scale=スケール)	
指数分布	expon	pdf(x=確率変数, scale=1/λ)	
二項分布	binom	pmf(確率変数, 試行回数, 成功確率)	
幾何分布	geom	pmf(確率変数, 成功確率)	
超幾何分布	hypergeom	pmf(確率変数, 要素の総数, 抽出数, 当たりの総数)	
ポアソン分布	poisson	pmf(確率変数, λ)	$\lambda = np$ （n=試行回数，p=確率）

各クラスには，次のような関数がある．

- 累積分布関数 cdf cumulative density function
- 生存関数 sf survival function
- 上記の逆関数 isf inverse survival function
- パーセント点関数 ppf percent point function キーワード引数：q=累積確率
- 乱数生成 rvs random variates キーワード引数：size=生成データ個数

C　SQL データベースについて

C.1　各種の DBMS

データベース管理システム（DBMS）はそれ自体で 1 つの独立したシステムであり，他のアプリケーションプログラムからアクセス（読み出し，更新）の要求（クエリ）を受けて処理を行うものである．商用の製品としては Microsoft 社の SQL Server，ORACLE 社の Oracle Database などが有名である．フリーソフトウェアでも機能，性能の両面で充実したものとして PostgreSQL [33]，MySQL [34] が有名である．これらの DBMS は複数のアプリケーションプログラムから同時にアクセスを受けることを前提に作られており，複数のユーザを管理し，高度な排他制御を行う機能が実装されている．

C.1.1　SQLite

単一のアプリケーションからのデータ管理を想定した DBMS としては SQLite [35] が有名であり，本書でもデータベースを用いる例についてはこれを利用する形で解説している．SQLite はパブリックドメインのソフトウェアであり，小規模ながら，個人で利用する範囲では充分な機能と性能を提供する．特に，データ管理機能を持つアプリケーションソフトウェアを開発する際に SQLite を利用するケースが多く見られる．

SQLite では，1 つのデータベースは 1 つのファイルとして管理されており，そのファイルの中に当該データベースの表が全て含まれる．SQLite のデータベースの内部を閲覧するための手軽なソフトウェアとして PupSQLite [36] というものがある．（使用例を後で示す）

C.2　SQLAlchemy によるデータベースの扱い

SQLAlchemy は**オブジェクト関係マッピング**（ORM：Object-Relational mapping）に基づく形で関係データベース（RDB）を扱う．これは，関係データベースの表の設計や，アクセスの方法を可能な限り**オブジェクト指向プログラミング**（OOP）の作法に則った形で実現しようとする方針である．従って SQLAlchemy は，ORM のためのクラスを提供する．（ただし，ORM に関する詳細は本書では扱わない）

本書では SQLAlchemy を読み込む際に

```
import sqlalchemy as SQLA
```

として，関数やメソッドを読み出す際の接頭辞を短縮する．これにより例えば，

```
egn = SQLA.create_engine('sqlite:///testdb', echo=True)
```

などと簡略化された形で文を記述することができる．

[33] PostgreSQL Global Development Group が開発，保守している．(https://www.postgresql.org/)
[34] ORACLE 社が管理している．
[35] 公式インターネットサイト：https://www.sqlite.org/
[36] 公式インターネットサイト：https://www.eonet.ne.jp/ pup/

C.2.1　データベース創成の例（CSV から RDB）

ここでは，SQLAlchemy と pandas を用いて，CSV データから RDB を創成する過程のサンプルを示す．扱うデータは次のようなものである．

● **CSV データ 1：**　都道府県別の行政組織の数　db01_gov_office.csv

項目 No.	項目名	型	項目 No.	項目名	型	項目 No.	項目名	型
1	番号	整数	4	政令市	整数	7	区役所	整数
2	都道府県	文字列	5	特別区	整数	8	町役場	整数
3	県庁	整数	6	市役所	整数	9	村役場	整数

● **CSV データ 2：**　都道府県別の面積の情報　db01_geographic.csv

項目 No.	項目名	型	項目 No.	項目名	型
1	都道府県	文字列	5	可住地比率	実数
2	総面積	整数	6	可住地人口密度	整数
3	総面積人口密度	整数	7	森林面積	整数
4	可住地面積	整数	8	森林比率	実数

備考： 面積の単位は km^2，人口密度は $1km^2$ 当たりの人数

● **創成するデータベース**

データベース名	：	DB01		
表の名称	：	主キーの項目	：	元になるデータ
「行政組織数」	：	「番号」	：	db01_gov_office.csv
「面積情報」	：	「都道府県」	：	db01_geographic.csv

ここで想定するデータベースは，表の間の関連を定義しない単純なものであり，外部キーによる他の表への連携も行わない．このデータベースを SQLite のデータとして創成するプログラムの例を gendB01.py に示す．

プログラム：gendB01.py

```
# coding: utf-8
#############################################################################
#    モジュールの読込み                                                     #
#############################################################################
#--- SQLAlchemy関連 ---
import sqlalchemy as SQLA
from sqlalchemy.ext.declarative import declarative_base
#--- Pandas関連 ---
import pandas as pd

#############################################################################
#    データベースの設計と表の作成                                           #
#############################################################################
#--- ベースクラスの生成 ---
decBase = declarative_base()

#--- 行政組織数テーブルの定義 ---
class 行政組織数(decBase):
    __tablename__ = '行政組織数'
    番号          = SQLA.Column(SQLA.Integer, primary_key=True)
    都道府県      = SQLA.Column(SQLA.String)
    県庁          = SQLA.Column(SQLA.Integer)
    政令市        = SQLA.Column(SQLA.Integer)
    特別区        = SQLA.Column(SQLA.Integer)
    市役所        = SQLA.Column(SQLA.Integer)
    区役所        = SQLA.Column(SQLA.Integer)
    町役場        = SQLA.Column(SQLA.Integer)
    村役場        = SQLA.Column(SQLA.Integer)

#--- 面積情報テーブルの定義 ---
class 面積情報(decBase):
    # テーブル名の定義
    __tablename__   = '面積情報'
    # 項目の定義
    都道府県          = SQLA.Column(SQLA.String, primary_key=True)
    総面積            = SQLA.Column(SQLA.Integer)
    総面積人口密度    = SQLA.Column(SQLA.Integer)
    可住地面積        = SQLA.Column(SQLA.Integer)
    可住地比率        = SQLA.Column(SQLA.Float)
    可住地人口密度    = SQLA.Column(SQLA.Integer)
    森林面積          = SQLA.Column(SQLA.Integer)
    森林比率          = SQLA.Column(SQLA.Float)

#--- データベースへの接続 ---
egn = SQLA.create_engine('sqlite:///DB01.db')

#--- テーブルの作成 ---
decBase.metadata.bind = egn
decBase.metadata.create_all(egn)

#############################################################################
#    データの登録                                                           #
#############################################################################
#--- 行政組織数のCSVデータ ---
df = pd.read_csv('db01_gov_office.csv')
df.to_sql( '行政組織数', egn, index=False, if_exists='replace' )

#--- 面積情報のCSVデータ --
df = pd.read_csv('db01_geographic.csv')
df.to_sql( '面積情報', egn, index=False, if_exists='replace' )
```

解説：

SQLAlchemy では，ORM に基いて RDB の表（table）を設計する．そのために必要な declarative_base というクラスを使用する．このクラスは sqlalchemy.ext.declarative に提供されており，使用に際して読み込んでおく必要がある．これを行っているのが 7 行目である．

表を定義するには，declarative_base を継承したクラスとして記述する．表のクラスの定義には，項目（列）を定義するための Column オブジェクトとその型を定義するためのオブジェクトが必要となる．15 行目では declarative_base オブジェクト decBase を生成し，それを使用して 18〜42 行目で表を定義している．

表を定義するクラスでは，__tablename__ プロパティに表の名称を与える．表の各項目は Column オブジェクトとして定義（生成）する．この際，Column の第 1 引数に項目の型を与える．この際，文字列は String，整数は Integer，浮動小数点数（実数）は Float として与える．また，キーワード引数 primary_key=True を与えるとその項目が主キーとなる．

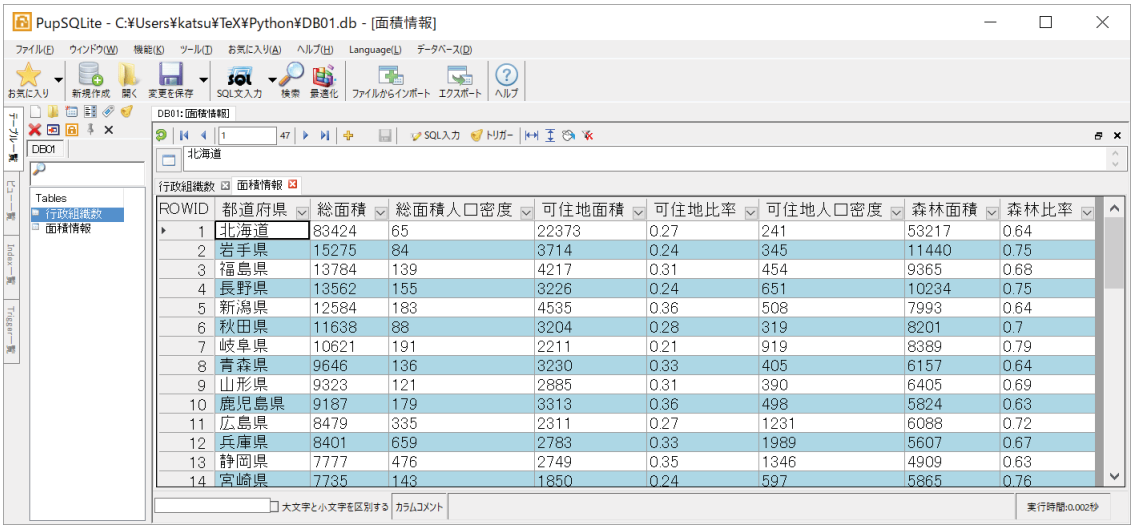
45 行目では，データベースに接続するための Engine オブジェクト egn を生成し，それに対して 48 行目で表の定義を与え，49 行目で create_all メソッドを用いて表を生成している．（この段階では表の内容は空である）

55 行目では CSV データ db01_gov_office.csv を読み込んで pandas の DataFrame オブジェクト df に与え，56 行目ではその内容をデータベースの表「行政組織数」に与えている．同様の方法により，59〜60 行目で「面積情報」に db01_geographic.csv の内容を与えている．

■ 実行例

2 つの CSV データ db01_gov_office.csv，db01_geographic.csv と同じディレクトリに Python のプログラム gendB01.py を配置して実行すると，データベースがファイル DB01.db として創成される．

データベース DB01.db の内容を PupSQLite で閲覧することができる．（図 15）



ROWID	都道府県	総面積	総面積人口密度	可住地面積	可住地比率	可住地人口密度	森林面積	森林比率
1	北海道	83424	65	22373	0.27	241	53217	0.64
2	岩手県	15275	84	3714	0.24	345	11440	0.75
3	福島県	13784	139	4217	0.31	454	9365	0.68
4	長野県	13562	155	3226	0.24	651	10234	0.75
5	新潟県	12584	183	4535	0.36	508	7993	0.64
6	秋田県	11638	88	3204	0.28	319	8201	0.7
7	岐阜県	10621	191	2211	0.21	919	8389	0.79
8	青森県	9646	136	3230	0.33	405	6157	0.64
9	山形県	9323	121	2885	0.31	390	6405	0.69
10	鹿児島県	9187	179	3313	0.36	498	5824	0.63
11	広島県	8479	335	2311	0.27	1231	6088	0.72
12	兵庫県	8401	659	2783	0.33	1989	5607	0.67
13	静岡県	7777	476	2749	0.35	1346	4909	0.63
14	宮崎県	7735	143	1850	0.24	597	5865	0.76

図 15: PupSQLite で DB01.db を開いた様子

C.2.2　選択・射影・結合の例

先に生成したデータベース DB01.db を用いて，**選択**，**射影結合**を行う例を示す．

pandas を用いてデータベースから表を読み込んで DataFrame にするには read_sql メソッド使用する．このとき，第 1 引数にクエリのための SQL 文を，第 2 引数に接続先の Engine オブジェクトを与える．データベースからの読込みには SQL の SELECT 文を記述する．

> **《SELECT 文の基本》**
> **書き方：**　SELECT **対象の列** FROM **表の名前** WHERE **読込みの条件**

「対象の列」としてアスタリスク '*' を記述すると全ての列（項目）が読み込まれる．まずは全ての行の全ての列を読み込む例を示す．

例. 表から全ての行と列のデータを読み込む

In [1]:
```
import sqlalchemy as SQLA    # SQLAlchemyの読込み
import pandas as pd          # pandasの読込み
```

In [2]:
```
# DB接続（エンジンの取得）
egn = SQLA.create_engine('sqlite:///DB01.db', echo=False)
```

In [3]:
```
q = 'SELECT * from 行政組織数'    # SQLクエリ
df = pd.read_sql(q,egn)          # データベースから読込み
df.head(4)     # 内容確認
```

Out[3]:

	番号	都道府県	県庁	政令市	特別区	市役所	区役所	町役場	村役場
0	1	北海道	1	1	0	34	10	129	15
1	2	青森県	1	0	0	10	0	22	8
2	3	岩手県	1	0	0	14	0	15	4
3	4	宮城県	1	1	0	13	5	21	1

これは「行政組織数」の表の内容を全て読み取ったものを DataFrame オブジェクト df にして先頭の 4 行を表示する例である．

指定した特定の列のみを読み込む際は SELECT 文の次に列の名前を記述する．複数の列を指定する場合はコンマで区切る．

例. 表から特定の列のみを読み込む

In [4]:
```
# 列の抽出（射影）
q = 'SELECT 都道府県,市役所 from 行政組織数'    # SQLクエリ
df = pd.read_sql(q,egn)          # データベースから読込み
df.head(4)     # 内容確認
```

Out[4]:

	都道府県	市役所
0	北海道	34
1	青森県	10
2	岩手県	14
3	宮城県	13

これは「行政組織数」の表から「都道府県」と「市役所」の列を読込み，先頭の 4 行を表示する例である．

DB01.db には 2 つの表があり，それぞれに共通の項目（列）として「都道府県」がある．この共通の項目で 2 つの表を**結合**して 1 つの表の様に読み込むことができる．その例を次に示す．

例. 表の結合

In [5]:
```
# 表の結合（全レコード）
# SQLクエリ
q = 'SELECT 番号,行政組織数.都道府県,市役所,総面積,総面積人口密度 \
        FROM 行政組織数,面積情報 \
        WHERE 行政組織数.都道府県=面積情報.都道府県'
df = pd.read_sql(q,egn)          # データベースから読込み
df.head(4)     # 内容確認
```

Out[5]:

	番号	都道府県	市役所	総面積	総面積人口密度
0	1	北海道	34	83424	65
1	2	青森県	10	9646	136
2	3	岩手県	14	15275	84
3	4	宮城県	13	7282	320

この例では，2 つの表を同時に読み込んで結合しているので「都道府県」の項目がどちらの表のものかを明示するためにドット「 .」を記述している．すなわち,「表の名称．項目の名称」という記述となる．

複数の表を同時に読み込んで結合する処理は柔軟なデータ処理のために必要となる．

大量のレコードを保持するデータベースから，必要とするレコードのみを**選択**する処理が重要である．選択の条件は WHERE の後に記述する．(次の例)

例． レコードの選択

In [6]:
```
# 表の結合（選択条件：可住地人口密度が1900以上のレコードを抽出）
# SQLクエリ
q = 'SELECT 番号,行政組織数.都道府県,政令市,可住地面積,可住地人口密度 \
        FROM 行政組織数,面積情報 \
        WHERE 行政組織数.都道府県=面積情報.都道府県 AND \
                可住地人口密度>=1900'
df = pd.read_sql(q,egn)         # データベースから読込み
# 可住地人口密度の降順で表示
df.sort_values('可住地人口密度',ascending=False)
```

Out[6]:

	番号	都道府県	政令市	可住地面積	可住地人口密度
5	13	東京都	0	1418	9529
6	27	大阪府	2	1331	6643
4	14	神奈川県	3	1471	6206
3	11	埼玉県	1	2585	2811
1	23	愛知県	1	2988	2505
2	26	京都府	1	1174	2224
0	28	兵庫県	1	2783	1989

これは 2 つの表を結合し,「可住地人口密度」が 1,900 以上であるレコードのみを選択して読み込んでいる例である．

索 引