PG07: データの前処理

データ（ここでは csvファイルを前提にしている）を読み込んでもすぐに使えるとは限らない。 そもそも、データが日本語を含んでいるときは、文字コードの不整合によって読み込めないこともある。 さらに、読み込んだデータには欠損値があったり、数字であっても「文字列」であれば、そのままではデータ解析や統計分析ができない。 複雑な事象に対しては、その対応も一筋縄で行かないのだが、ここでは単純な場合について最も基本的な対応について述べておく。

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

7.1. 日本語の文字コード (CodingError.csv)

日本語が混ざっている csv ファイルの読み込みは失敗しがちである。 オプションに

encoding='shift-jis'

を加えることで解決することが多いが、それでもだめなら

encoding='cp932'

を試してみよ。

pd.read_csv('F:2022_数理統計学共通問題集/Data/Example2016.csv', encoding='cp932')

のように記述する。

典型的なエラー(UnicodeDecodeError)の例

In [2]:

```
Sample = pd.read_csv('F:/2022_数理統計学概論/StatData/CodingError.csv')
Sample
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-2-0650a0966fdf> in <module>
----> 1 Sample = pd.read_csv('F:/2022_数理統計学概論/StatData/CodingError.csv')
      2 Sample

C:¥Anaconda3¥lib¥site-packages¥pandas¥io¥parsers.py in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map, float_precision)
    684     )
    685
--> 686     return _read(filepath_or_buffer, kwds)
    687
    688

C:¥Anaconda3¥lib¥site-packages¥pandas¥io¥parsers.py in _read(filepath_or_buffer, kwds)
    450
    451     # Create the parser.
--> 452     parser = TextFileReader(fp_or_buf, **kwds)
    453
    454     if chunksize or iterator:

C:¥Anaconda3¥lib¥site-packages¥pandas¥io¥parsers.py in __init__(self, f, engine, **kwds)
    944             self.options["has_index_names"] = kwds["has_index_names"]
    945
--> 946         self._make_engine(self.engine)
    947
    948     def close(self):

C:¥Anaconda3¥lib¥site-packages¥pandas¥io¥parsers.py in _make_engine(self, engine)
   1176     def _make_engine(self, engine="c"):
   1177         if engine == "c":
-> 1178             self._engine = CParserWrapper(self.f, **self.options)
   1179         else:
   1180             if engine == "python":

C:¥Anaconda3¥lib¥site-packages¥pandas¥io¥parsers.py in __init__(self, src, **kwds)
   2006         kwds["usecols"] = self.usecols
   2007
-> 2008         self._reader = parsers.TextReader(src, **kwds)
   2009         self.unnamed_cols = self._reader.unnamed_cols
   2010
```

```
pandas¥_libs¥parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas¥_libs¥parsers.pyx in pandas._libs.parsers.TextReader._get_header()

UnicodeDecodeError: 'utf-8' codec can't decode byte 0x93 in position 0: invalid st
art byte
```

In [3]:

```
Sample = pd.read_csv('F:/2022_数理統計学概論/StatData/CodingError.csv', encoding='shift-jis')
Sample
```

Out[3]:

	日本語文字コード(shift-jis)
色名	図形
青	正方形
緑	平行四辺形
黄	正四面体
赤	六角錐
紫	星形
黒	正五角形

In [4]:

```
Sample = pd.read_csv('F:/2022_数理統計学概論/StatData/CodingError.csv',
                     encoding = 'shift-jis',
                     skiprows=1)
Sample
```

Out[4]:

	色名	図形
0	青	正方形
1	緑	平行四辺形
2	黄	正四面体
3	赤	六角錐
4	紫	星形
5	黒	正五角形

In [5]:

```
Sample = pd.read_csv('F:/2022_数理統計学概論/StatData/CodingError.csv',
                     encoding='shift-jis',
                     skiprows=2,
                     names=['color', 'figure'])
Sample
```

Out[5]:

	color	figure
0	青	正方形
1	緑	平行四辺形
2	黄	正四面体
3	赤	六角錐
4	紫	星形
5	黒	正五角形

7.2. 欠損データの扱い (Cleansing.csv)

サンプルで用意した Cleansing.csv も文字コードのエラーが出るので、前節の方法でオプション encoding='shift-jis' を付けて読み込む。

In [6]:

```
Data = pd.read_csv('F:/2022_数理統計学概論/StatData/Cleansing.csv',
                   encoding='shift-jis',
                   skiprows=1,
                   names=['No', 'Mscore', 'Fscore'])
Data
```

Out[6]:

	No	Mscore	Fscore
0	101	20	58
1	102	欠席	53
2	103	59	71
3	104	29	51
4	105	4	38
...	...	...	...
59	160	31	68
60	161	15	54
61	162	10	57
62	163	14	65
63	164	40	70

64 rows × 3 columns

欠損値

点数欄に「欠席」があるので、浮動小数点数として扱われる欠損値 NaN で置き換える必要がある。

「欠席」以外の書き込みがあれば、それも同様に NaN で置き換える必要がある。 そのため、前もってデータを全部見ておきたければ、

pd.set_option('display.max_rows', None)

を実行して、データの表示行数の制限を解除する。

In [7]:

```
pd.set_option('display.max_rows', None)
```

In [8]:

```
Data
```

Out[8]:

	No	Mscore	Fscore
0	101	20	58
1	102	欠席	53
2	103	59	71
3	104	29	51
4	105	4	38
5	106	59	74
6	107	47	77
7	108	46	71
8	109	15	61
9	110	25	53
10	111	50	80
11	112	64	72
12	113	2	50
13	114	24	69
14	115	52	41
15	116	51	61
16	117	22	75
17	118	25	65
18	119	35	37
19	120	60	77
20	121	11	57
21	122	32	65
22	123	55	64
23	124	49	72
24	125	63	71
25	126	47	57
26	127	27	43
27	128	27	64
28	129	37	56
29	130	54	70
30	131	47	72
31	132	84	77
32	133	72	100
33	134	52	58

	No	Mscore	Fscore
34	135	40	87
35	136	11	67
36	137	17	50
37	138	46	70
38	139	55	75
39	140	74	87
40	141	21	45
41	142	52	77
42	143	54	75
43	144	21	72
44	145	44	78
45	146	24	77
46	147	17	59
47	148	74	84
48	149	52	35
49	150	7	44
50	151	32	54
51	152	27	85
52	153	0	15
53	154	24	57
54	155	35	70
55	156	10	43
56	157	欠席	欠席
57	158	37	56
58	159	35	52
59	160	31	68
60	161	15	54
61	162	10	57
62	163	14	65
63	164	40	70

「欠席」以外の欠損値はないので、「欠席」だけを np.nan に置き換えればよい。

In [9]:

```
Data = Data.replace('欠席', np.nan)
Data
```

Out[9]:

	No	Mscore	Fscore
0	101	20	58
1	102	NaN	53
2	103	59	71
3	104	29	51
4	105	4	38
5	106	59	74
6	107	47	77
7	108	46	71
8	109	15	61
9	110	25	53
10	111	50	80
11	112	64	72
12	113	2	50
13	114	24	69
14	115	52	41
15	116	51	61
16	117	22	75
17	118	25	65
18	119	35	37
19	120	60	77
20	121	11	57
21	122	32	65
22	123	55	64
23	124	49	72
24	125	63	71
25	126	47	57
26	127	27	43
27	128	27	64
28	129	37	56
29	130	54	70
30	131	47	72
31	132	84	77
32	133	72	100

	No	Mscore	Fscore
33	134	52	58
34	135	40	87
35	136	11	67
36	137	17	50
37	138	46	70
38	139	55	75
39	140	74	87
40	141	21	45
41	142	52	77
42	143	54	75
43	144	21	72
44	145	44	78
45	146	24	77
46	147	17	59
47	148	74	84
48	149	52	35
49	150	7	44
50	151	32	54
51	152	27	85
52	153	0	15
53	154	24	57
54	155	35	70
55	156	10	43
56	157	NaN	NaN
57	158	37	56
58	159	35	52
59	160	31	68
60	161	15	54
61	162	10	57
62	163	14	65
63	164	40	70

7.3: 数字に見える文字列

読み出した結果が数字であっても、「文字列 (str)」として認識されている場合がある。 文字列に対しては四則演算ができず、統計量の計算もできない。 「整数型(int)」または「浮動小数点数型(float)」に変換する必要がある。

Dataの0行2列のデータの型を調べよう。

In [10]:

```
Data.iat[0,2], type(Data.iat[0,2])
```

Out[10]:

```
('58', str)
```

In [11]:

```
Data.dtypes
```

Out[11]:

```
No         int64
Mscore    object
Fscore    object
dtype: object
```

NaN は浮動小数点数型をもつので、それに合わせて Mscore, Fscore ともに浮動小数点数型に変換する。

NaN が含まれなければ、astype(int) によって整数型に変換してもよい。

In [12]:

```
Data['Mscore'] = Data['Mscore'].astype(float)
Data['Fscore'] = Data['Fscore'].astype(float)
```

In [13]:

```
Data.dtypes
```

Out[13]:

```
No           int64
Mscore     float64
Fscore     float64
dtype: object
```

こうして、点数はすべて計算可能な浮動小数点数型に変換された。

NaN の扱いはあらかじめ決められている。たとえば、平均値の計算からは自動的に除外される。

In [14]:

```
np.mean(Data['Mscore']), np.mean(Data['Fscore'])
```

Out[14]:

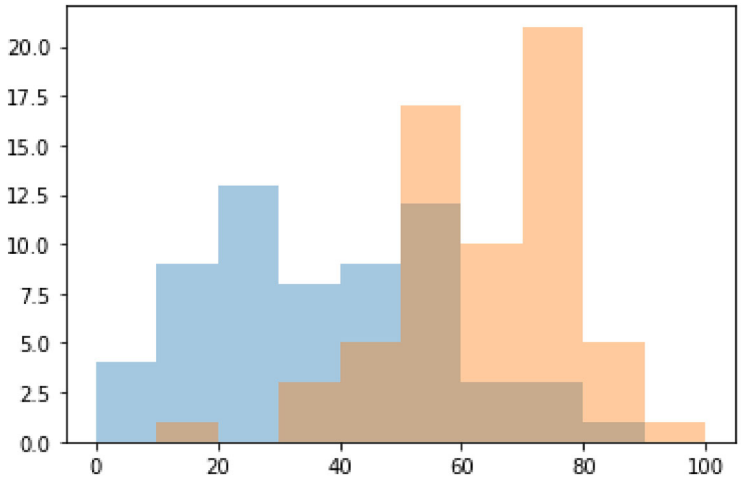
```
(36.53225806451613, 63.301587301587304)
```

In [15]:

```
plt.hist(Data['Mscore'], range=(0,100), bins=10, alpha=0.4)
plt.hist(Data['Fscore'], range=(0,100), bins=10, alpha=0.4)
```

Out[15]:

```
(array([ 0.,  1.,  0.,  3.,  5., 17., 10., 21.,  5.,  1.]),
 array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.]),
 <BarContainer object of 10 artists>)
```



NaN を含む行の削除

NaN を含む行は不要であるということであれば、一斉削除ができる。 このとき、NaN を1個以上含む行を削除するのか、すべてのデータが NaN であるような行だけを削除するのか、 目的によって決める。dropna() メソッドのデフォルトは前者、dropna(how='all')は後者。

なお、もとの Data を保持しておきたければ、NaN を削除したDataFrameには新しい名前を付ければよい。

In [16]:

```
Data_new = Data.dropna()
```

In [17]:

```
Data_new
```

Out[17]:

	No	Mscore	Fscore
0	101	20.0	58.0
2	103	59.0	71.0
3	104	29.0	51.0
4	105	4.0	38.0
5	106	59.0	74.0
6	107	47.0	77.0
7	108	46.0	71.0
8	109	15.0	61.0
9	110	25.0	53.0
10	111	50.0	80.0
11	112	64.0	72.0
12	113	2.0	50.0
13	114	24.0	69.0
14	115	52.0	41.0
15	116	51.0	61.0
16	117	22.0	75.0
17	118	25.0	65.0
18	119	35.0	37.0
19	120	60.0	77.0
20	121	11.0	57.0
21	122	32.0	65.0
22	123	55.0	64.0
23	124	49.0	72.0
24	125	63.0	71.0
25	126	47.0	57.0
26	127	27.0	43.0
27	128	27.0	64.0
28	129	37.0	56.0
29	130	54.0	70.0
30	131	47.0	72.0
31	132	84.0	77.0
32	133	72.0	100.0
33	134	52.0	58.0
34	135	40.0	87.0

	No	Mscore	Fscore
35	136	11.0	67.0
36	137	17.0	50.0
37	138	46.0	70.0
38	139	55.0	75.0
39	140	74.0	87.0
40	141	21.0	45.0
41	142	52.0	77.0
42	143	54.0	75.0
43	144	21.0	72.0
44	145	44.0	78.0
45	146	24.0	77.0
46	147	17.0	59.0
47	148	74.0	84.0
48	149	52.0	35.0
49	150	7.0	44.0
50	151	32.0	54.0
51	152	27.0	85.0
52	153	0.0	15.0
53	154	24.0	57.0
54	155	35.0	70.0
55	156	10.0	43.0
57	158	37.0	56.0
58	159	35.0	52.0
59	160	31.0	68.0
60	161	15.0	54.0
61	162	10.0	57.0
62	163	14.0	65.0
63	164	40.0	70.0

Data_new を使って平均値を確認しておこう。 先に計算した結果（No.56のデータの扱いが異なる）と比較しておくとよい。

In [18]:

```
np.mean(Data_new['Mscore']), np.mean(Data_new['Fscore'])
```

Out[18]:

```
(36.53225806451613, 63.46774193548387)
```

In [ ]: