总结

关键：[row , col]

对于Dataframe，如果相应**操控列col**，直接df ['col' 名] ，如果要**操作行row**，则df.loc['row'名]

In [94]:
```
df3
```

Out[94]:

	a	b	c
one	1000	90000	1000
two	2	5	8
three	3	6	9

In [102]:
```
#row
df3.loc['one']
```

Out[102]:
```
a     1000
b    90000
c     1000
Name: one, dtype: int64
```

In [103]:
```
#col
df3['a']
```

Out[103]:
```
one      1000
two         2
three       3
Name: a, dtype: int64
```

iloc，则是对matrix的index做处理。

```
[58]: df
```

[58]:

	one	two	three
a	1	2	3
b	4	5	6
c	7	8	9

```
[54]: #index iloc
      #row
      df.iloc[0]
```

```
[54]: one      1
      two      2
      three    3
      Name: a, dtype: int32
```

```
[62]: #cols
      df.iloc[:,0]
```

```
[62]: a    1
      b    4
      c    7
      Name: one, dtype: int32
```

```
[56]: # row and col
      df.iloc[0,0]
```

```
[56]: 1
```

loc 与条件

还是关键 [row , col]

df.loc[df.column_name 条件]的意思是，选择column_name符合某些条件的row数据。

```
[64]: # loc与条件配对
      df
```

[64]:

	one	two	three
a	1	2	3
b	4	5	6
c	7	8	9

行

```
[72]: df.loc[df.two>=5]
```

[72]:

	one	two	three
b	4	5	6
c	7	8	9

行 列

```
[70]: df.loc[df.two>=5,'three']
```

```
[70]: b    6
      c    9
      Name: three, dtype: int32
```

还可以基于多个列条件来筛选row数据。

```
[150]: temp_2 = df.loc[(df['income']=='>50K') & (df['education'] == 'HS-grad')]

[152]: temp_2.head()
```

[152]:

	age	workclass	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_cou
7	52	Self-emp-not-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United-St
67	53	Private	HS-grad	9	Married-civ-spouse	Adm-clerical	Wife	White	Female	0	0	40	United-St
84	44	Private	HS-grad	9	Divorced	Craft-repair	Not-in-family	White	Female	14344	0	40	United-St
86	49	Local-gov	HS-grad	9	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-St
105	32	Self-emp-inc	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	White	Male	7688	0	40	United-St

```
[198]: plt.figure(figsize=(15,5))
```

Numpy的局限

不能处理混合数据类型的数据，这也是为什么我们要使用Pandas。

NumPy arrays are designed for numerical computation and **cannot easily handle collections of data that contain a mix of types** (e.g., strings, integers, and floats) in a single array.

A typical example of this limitation is a student attendance sheet. Each record might include a student's name (a string), their marks (an integer), and their student ID (which serves as a label). NumPy arrays aren't well-suited for such cases because they can't store multiple types of data in the same array or use labels for indexing.

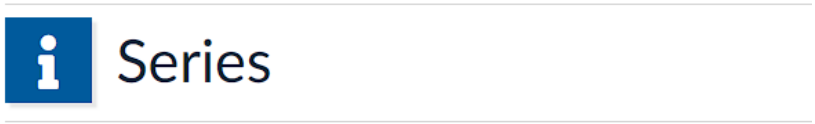
This is where **pandas** shines. pandas provides flexible and powerful tools to handle such **mixed-type datasets** and allows you to work with **labelled data**. It can store data in **DataFrames**, which are similar to tables in a database or Excel spreadsheet, making it ideal for data manipulation, analysis, and visualization tasks.

Series and Dataframe

pandas introduces two new data types: `Series` and `DataFrame`.

A pandas Series is a one-dimensional data structure that comprises of a key-value pair. It is similar to a Python dictionary, except it provides more freedom to manipulate and edit the data. In contrast, a pandas DataFrame is a two-dimensional data-structure that can be thought of as akin to a *spreadsheet*. A DataFrame can also be thought of as a *combination of two or more series*.

Series



You can create a series from a list or array. In th

Series	
Index	Value
0	-5
1	10
2	-3
3	-25
4	45

- **构造Series**

Constructing Series Objects

Let's construct a Series:

`pd.Series(data, index= index)`

- `data` can be an array-like, iterable, dict, or scalar (ie: a single value, like int, string, etc.) value.
- `index` is an optional parameter, by default it is an integer sequence starting from zero.

View examples

In [2]:

```
# Construct a series with a default index

pd.Series([5,10, 15, 20])
```

Out[2]:

```
0     5
1    10
2    15
3    20
dtype: int64
```

In [9]:

```
a=pd.Series([1,2,3,4],index=['a','b','c','d'])
a
```

Out[9]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

字典传入

In [10]:

```
#字典传入
b=pd.Series({'a':1,'b':2,'c':3,'d':4})
b
```

Out[10]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

```
In [15]: #字典传入
         #通过index取对应数值
         b=pd.Series({'a':1,'b':2,'c':3,'d':4},index=['a','c'])
         b

Out[15]: a    1
         c    3
         dtype: int64
```

pandas与Numpy的配合

```
In [17]: c=np.arange(10,100,10)
         cp=pd.Series(c)
         cp

Out[17]: 0    10
         1    20
         2    30
         3    40
         4    50
         5    60
         6    70
         7    80
         8    90
         dtype: int32
```

slicing

与Numpy一样

```
In [18]: cp

Out[18]: 0    10
         1    20
         2    30
         3    40
         4    50
         5    60
         6    70
         7    80
         8    90
         dtype: int32
```

```
In [21]: cp[0:8:2]

Out[21]: 0    10
         2    30
         4    50
         6    70
         dtype: int32
```

```
In [22]: cp[::-2]

Out[22]: 8    90
         6    70
         4    50
         2    30
         0    10
         dtype: int32
```

series filtering

```
array

Out[18]: array([ 7, 59, 42, 13, 67, 19, 26, 59, 99, 97, 77,  1, 36, 49, 10, 51, 41,
                73, 33, 79, 19, 34, 84, 11, 41, 75])
```

```
In [20]: series_6=array[array>50]
         series_6

Out[20]: array([59, 67, 59, 99, 97, 77, 51, 73, 79, 84, 75])
```

DataFrame

DataFrame

The pandas Series is a 1-dimensional labelled structure. DataFrame is a 2-dimensional labelled structure. As with a Series, it is built on top of NumPy arrays to take advantage of faster processing (compared to Python Lists and Dictionaries). DataFrames are 2-D arrays with attached row and columns labels, and generally with different data types across columns and/or missing data. A DataFrame is very similar to a spreadsheet.

In the table, you can see the structure of a Data Frame.

Data Frame				
	Columns			
index	city	state	pop_density	unemployed_rate
0	Sydney	New South Wales	4627345	4.3%
1	Melbourne	Victoria	4246375	4.9%
2	Brisbane	Queensland	2189878	NaN
3	Adelaide	South Australia	1225235	7.3%
4	Canberra	Australian Capital Territory	367752	3.5%
5	Perth	Western Australia	1896548	NaN
6	Hobart	Tasmania	NaN	6.4%
7	Darwin	Northern Territory	NaN	6.1%

构造

传入DataFrame的是一个array，或字典

In [24]:
```
#以Numpy array来构造
df1=pd.DataFrame(np.array([[1,2,3,4],['a','b','c','d']]))
df1
```

Out[24]:

	0	1	2	3
0	1	2	3	4
1	a	b	c	d

In [27]:
```
#以Series来构造
df2=pd.DataFrame([pd.Series(np.arange(4)),pd.Series(['a','b','c','d']),pd.Series([1.2,10.9,5.6,2.0])])
df2
```

Out[27]:

	0	1	2	3
0	0	1	2	3
1	a	b	c	d
2	1.2	10.9	5.6	2.0

传入字典，相当于定义col name。

In [36]:
```
a=pd.Series(np.array([1,2,3]))
b=pd.Series(np.array([3,4,5]))
c=pd.Series(np.array([5,6,7]))

df3=pd.DataFrame({'a':a,'b':b,'c':c})
df3
```

Out[36]:

	a	b	c
0	1	3	5
1	2	4	6
2	3	5	7

字典对应的value必须得是array-like object

In [37]:
```
#传入字典
df4=pd.DataFrame({'a':[2,4,5],'b':[4,5,7]})
df4
```

Out[37]:

	a	b
0	2	4
1	4	5
2	5	7

index

In [38]:
```
#传入字典
df4=pd.DataFrame({'a':[2,4,5],'b':[4,5,7]},index=['one','two','three'])
df4
```

Out[38]:

	a	b
one	2	4
two	4	5
three	5	7

df取得指定值

Out[12]:

	a	b	c
0	1	M	abc
1	2	S	cdb
2	3	M	aww

In [24]:
```
#通过[col][row]来取得数值
df1['b'][1:],df1['c'][2]
```

Out[24]:
```
(1    S
 2    M
 Name: b, dtype: object,
 'aww')
```

df 数据类型转换

传入字典，指定数据类型

In [36]:
```
df2=pd.DataFrame({'one':['a','b'],'two':[1,2]})
df2
```

Out[36]:

	one	two
0	a	1
1	b	2

In [37]:
```
df2.dtypes
```

Out[37]:
```
one     object
two      int64
dtype: object
```

In [40]:
```
df2=df2.astype({'two':'float'})
df2.dtypes
```

Out[40]:
```
one      object
two     float64
dtype: object
```

In [41]:
```
df2
```

Out[41]:

	one	two
0	a	1.0
1	b	2.0

常用的操作

- **更换index与赋值**

In [12]:
```python
# Write your solution here
az='abcdefghijklmnopqrstuvwxyz'
the_index=[]
for c in az:
    the_index.append(c)
# YOUR CODE HERE
series_4=series_3.copy()
series_4.index=the_index#更换index
series_4['a']=100 #通过index来更新数值。
series_4
```

Out[12]:
```
a    100
b     59
c     42
d     13
e     67
f     19
g     26
```

- **数据类型转换**

在 `pandas` 中，可以使用 `astype()` 方法来更改 `Series` 中数值的类型。将 `int` 转换为 `float` 也非常简单。

示例：将 `int` 转换为 `float`

python 複製程式碼

```python
import pandas as pd

# 创建一个包含整数的 Series
s = pd.Series([1, 2, 3, 4, 5])

# 将整数类型转换为浮点数类型
s_float = s.astype(float)
print(s_float)
```

In [16]:
```
series_5 = pd.Series(['1','2','3','4'])

# Write your solution here

# YOUR CODE HERE
series_5.name='Float data'
series_5.name
series_5=series_5.astype(float)
series_5
```

Out[16]:
```
0    1.0
1    2.0
2    3.0
3    4.0
Name: Float data, dtype: float64
```

Dataframe Operation

NanN Not a number

缺失数据，统一用np.nan

Pandas uses the `NaN` value to fill these spaces.

Note

Pandas allows you to explicitly define Not a Number (NaN) values and add them to a Series or a DataFrame.

Tip: Try to be consistent for all the data that you define as missing data. We suggest you use `np.nan` in all the cases.

原来可以直接放Series数据进series，并且缺失的数据，会自动以nan来填充。

not a number

In [3]:
```
a=pd.Series({'a':1,'b':2,'c':3,'d':np.nan})
a
```

Out[3]:
```
a    1.0
b    2.0
c    3.0
d    NaN
dtype: float64
```

In [4]:
```
b=pd.Series(a,index=['a','b','c','d','onee','twoo'])
b
```

Out[4]:
```
a       1.0
b       2.0
c       3.0
d       NaN
onee    NaN
twoo    NaN
dtype: float64
```

isnull()

若为nan，则为true.

Out[4]:
```
a        1.0
b        2.0
c        3.0
d        NaN
onee     NaN
twoo     NaN
dtype: float64
```

In [5]:
```
b.isnull()
```

Out[5]:
```
a       False
b       False
c       False
d        True
onee     True
twoo     True
dtype: bool
```

unique()

```
In [7]: b
Out[7]: a       1.0
        b       2.0
        c       3.0
        d       NaN
        onee    NaN
        twoo    NaN
        dtype: float64
```

```
In [6]: np.unique(b)
Out[6]: array([ 1.,  2.,  3., nan])
```

value_counts()

```
In [8]: b.value_counts()
Out[8]: 1.0    1
        2.0    1
        3.0    1
        Name: count, dtype: int64
```

isin()

倒过来读：**the feature_name is in df**，若有则true，若无，则False

```
df.isin(['feature_name'])
```

```
In [13]: b

Out[13]: a       1.0
         b       2.0
         c       3.0
         d       NaN
         onee    NaN
         twoo    NaN
         dtype: float64
```

```
In [14]: b.isin([1,3])

Out[14]: a        True
         b       False
         c        True
         d       False
         onee    False
         twoo    False
         dtype: bool
```

配合使用any，**只要有一个匹配为true，则返回true**

Question 08 (10 Points)

Write a `test_requirement_4` to ensure the following columns have been dropped (requirement 4):

```
'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'
```

```
In [ ]: DROPPED_COLS = ['SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']

def test_requirement_4 (function_under_test):

    # Write your solution here
    ...
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
In [51]: total_cols=df.columns
print(total_cols)
total_cols.isin(['SibSp']).any()#只要有一个就为true

Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')

Out[51]: True
```

用法

取特定数值的rows.

```
In [20]: b
```

```
Out[20]: a        1.0
         b        2.0
         c        3.0
         d        NaN
         onee     NaN
         twoo     NaN
         dtype: float64
```

```
In [17]: #只取特定数值
         c=b[b.isin([1,3])]
         c
```

```
Out[17]: a    1.0
         c    3.0
         dtype: float64
```

sum()

sum

```
In [23]: b

Out[23]: a       1.0
         b       2.0
         c       3.0
         d       NaN
         onee    NaN
         twoo    NaN
         dtype: float64
```

```
In [22]: b.isnull().sum()

Out[22]: 3
```

DataFrame Select/filtering data

.values ： 取值

.loc: 通过row与col选取数据

.iloc：通过index 来选取数据

1. **Accessing All Data**
   To retrieve all the data in your DataFrame, you can use the `.values` attribute. This will return the data as a NumPy array, without the row and column labels.
2. **Selecting Data by Index**
   You can select specific data points using the **row and column labels** (or indices) with `.loc[]` and `.iloc[]`:
   - `loc[]`: This method allows you to select data by row and column **labels** (names).
   - `iloc[]`: This method is used to select data by **index positions** (numerical indices).

loc

主要以选取row为主

记住一点，**loc [row , col]，loc[df['col'] < 某条件]**

Question 11 (5 Points)

Create a DataFrame, `dataframe_11`, that is a copy of `dataframe_10` but with the index:

`'one', 'two', 'three', 'four', 'five'`

Change the colour of the mug from `'white'` to `'black'`. `dataframe_10` should not be changed.

In [41]:
```
# Write your solution here

# YOUR CODE HERE
dataframe_11=dataframe_10.copy()
dataframe_11.index=['one', 'two', 'three', 'four', 'five']
# dataframe_11['five','mug']='black'
dataframe_11.loc['five','colour']='black' #通过行row与列col，选择一个值
dataframe_11
```

Out[41]:

	object	colour
one	ball	blue
two	pen	green
three	pencil	yellow
four	paper	red
five	mug	black

若单纯取列，则直接[ ]取即可。

Out[69]:

	a	b	c
one	1	4	7
two	2	5	8
three	3	6	9

In [76]:
```
#取列
df3['a']
```

Out[76]:
```
one      1
two      2
three    3
Name: a, dtype: int64
```

In [71]:
```
#选取多行
df3.loc[['one','three']]
```

Out[71]:

	a	b	c
one	1	4	7
three	3	6	9

loc选取并赋值

In [81]:
```
#选取元素，并赋值
df3.loc['one','c']=100
df3
```

Out[81]:

	a	b	c
one	1	4	100
two	2	5	8
three	3	6	9

iloc

通过行列的index来获取对应的值。

```
In [45]: dataframe_11.iloc[0,1]='rainbow'
         dataframe_11
```

Out[45]:

	object	colour
one	ball	rainbow
two	pen	green
three	pencil	yellow
four	paper	red
five	mug	black

选取特定的列 [[ ]]

```
In [36]: #多选几列
         df_pop[['cities','density']]
```

Out[36]:

	cities	density
0	Sydney	4627345
1	Melbourne	4246375
2	Brisbane	2189878
3	Perth	1896548
4	Adelaide	1225235

In [35]:
```
pricing = {'colour' : ['blue','green','yellow','red','white'],
        'object' : ['ball','pen','pencil','paper','mug'],
        'price' : [1.2,1.0,0.6,0.9,1.7]}

# Write your solution here

# YOUR CODE HERE
dataframe_10=pd.DataFrame(pricing)
dataframe_10=dataframe_10[['object','colour']]
dataframe_10
```

Out[35]:

	object	colour
0	ball	blue
1	pen	green
2	pencil	yellow
3	paper	red
4	mug	white

slicing

	cities	density	state	unemployed_rate
0	Sydney	4627345	New South Wales	4.3
1	Melbourne	4246375	Victoria	4.9
2	Brisbane	2189878	Queensland	NaN
3	Perth	1896548	Western Australia	NaN
4	Adelaide	1225235	South Australia	7.3
5	Gold Coast	591473	Queensland	6.4
6	Canberra	367752	Australian Capital Territory	3.5
7	Newcastle	308308	New South Wales	4.3
8	Wollongong	292190	New South Wales	4.3
9	Logan City	282673	Queensland	6.4

In [41]:
```
df_pop[0:5:2]
```

Out[41]:

	cities	density	state	unemployed_rate
0	Sydney	4627345	New South Wales	4.3
2	Brisbane	2189878	Queensland	NaN
4	Adelaide	1225235	South Australia	7.3

loc, slicing with condition

筛选出，某列符合某条件的row

slicing with condition

```
In [43]: df_pop.loc[df_pop.density>=3000000]
```

Out[43]:

	cities	density	state	unemployed_rate
0	Sydney	4627345	New South Wales	4.3
1	Melbourne	4246375	Victoria	4.9

筛选符合条件的特定的列

slicing with condition

```
In [44]: df_pop.loc[df_pop.density>=3000000,['cities','state']]
```

Out[44]:

	cities	state
0	Sydney	New South Wales
1	Melbourne	Victoria

例子2

Out[90]:

	a	b	c
one	1000	90000	1000
two	2	5	8
three	3	6	9

In [91]:
```
#condition
#选取，满足某列数据为某条件的行。
df3.loc[df3['b']<10]
```

Out[91]:

	a	b	c
two	2	5	8
three	3	6	9

In [92]:
```
df3.loc[df3['b']>10]
```

Out[92]:

	a	b	c
one	1000	90000	1000

筛选的例子

Using the data loaded in Question 12, create a pandas DataFrame called `dataframe_13` containing the maximum temperature and the rainfall only on those days when the maximum temperature for the day was greater than 34 degrees.

In [52]:
```
dataframe_12.columns
```

Out[52]:
```
Index(['Minimum temperature (C)', 'Maximum temperature (C)', 'Rainfall (mm)',
       'Direction of maximum wind gust', 'Speed of maximum wind gust (km/h)',
       'Time of maximum wind gust'],
      dtype='object')
```

In [54]:
```
# Write your solution here

# YOUR CODE HERE
dataframe_13=dataframe_12[['Maximum temperature (C)','Rainfall (mm)']]
dataframe_13=dataframe_13[dataframe_13['Maximum temperature (C)']>=34]
dataframe_13
```

Out[54]:

Date	Maximum temperature (C)	Rainfall (mm)
10/1/22	37.0	0.0
11/1/22	40.3	0.0
20/1/22	35.1	0.0
26/1/22	34.1	0.0
27/1/22	34.4	14.2
31/1/22	35.2	0.0

获得每一列的数据类型 dtype()

Question 09 (4 Points) ¶

In Question 8, we created a DataFrame with 6 columns. Create a series called `series_9` with values that reflect the **data type** of each column in `dataframe_8` and an index consisting of the column labels. Set the `name` attribute of this series to `'Column Types'`.

In [31]:
```
# Write your solution here

# YOUR CODE HERE
series_9=dataframe_8.dtypes
series_9.name='Column Types'
series_9
```

Out[31]:
```
A          float64
B    datetime64[s]
C          float32
D            int32
E         category
F           object
Name: Column Types, dtype: object
```

计算某列中各项数据的比例

Question 19 (15 Points)

To end this practical, work out what percentage of female and male passengers survived. Present these two values in a variable called `survivors_19` that is a tuple `(female_percent, male_percent)` that expresses the percentage of survivors rounded to one decimal place like:

```
(72.1, 25.9)  # values are fictional but show the expected format of the result
```

To determine this value, explore the pandas API for the groupby function.

130]:
```
# Write your solution here

# YOUR CODE HERE
dataframe_18['Sex'].value_counts()
```

130]:
```
Sex
male      577
female    314
Name: count, dtype: int64
```

132]:
```
total=dataframe_18.shape[0]
total
```

132]:
```
891
```

137]:
```
female_percent=round(dataframe_18['Sex'].value_counts()['female']/total,2)*100
male_percent=round(dataframe_18['Sex'].value_counts()['male']/total,2)*100
survivors_19=(female_percent,male_percent)
survivors_19
```

137]:
```
(35.0, 65.0)
```

替换数据

需要配合使用numpy

888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	370376	7.7500	NaN	Q

891 rows × 10 columns

In [121]:
```
# Write your solution here

# YOUR CODE HERE
dataframe_18=dataframe_17.copy()
#根据列中特定数据的条件，进行替换。
#使用where
dataframe_18['Survived']=np.where(dataframe_18['Survived']==1,True,False)
dataframe_18
```

Out[121]:

	PassengerId	Survived	Passenger Class	Name	Sex	Age	Ticket	Fare	Cabin	Embarked
0	1	False	3	Braund, Mr. Owen Harris	male	22.0	A/5 21171	7.2500	NaN	S
1	2	True	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	PC 17599	71.2833	C85	C
2	3	True	3	Heikkinen, Miss. Laina	female	26.0	STON/O2. 3101282	7.9250	NaN	S
3	4	True	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	113803	53.1000	C123	S
4	5	False	3	Allen, Mr. William Henry	male	35.0	373450	8.0500	NaN	S

题目例子用法

In [37]:
```
df1.Survived=np.where(df1['Survived']==1,True,False)
```

In [50]:
```
df1.drop(['SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],axis=1)
```

Out[50]:

	PassengerId	Survived	Pclass	Name	Sex	Age
0	1	False	3	Braund, Mr. Owen Harris	male	22.0
1	2	True	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	3	True	3	Heikkinen, Miss. Laina	female	26.0
3	4	True	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
4	5	False	3	Allen, Mr. William Henry	male	35.0
...	...	...	...	...	...	...
886	887	False	2	Montvila, Rev. Juozas	male	27.0
887	888	True	1	Graham, Miss. Margaret Edith	female	19.0
888	889	False	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN
889	890	True	1	Behr, Mr. Karl Howell	male	26.0
890	891	False	3	Dooley, Mr. Patrick	male	32.0

891 rows × 6 columns

In [77]:
```
lucky=lucky_miss_andrews()
lucky=lucky[(lucky['Sex']=='female') & (lucky['Age']>=62)]
lucky
```

Out[77]:

	PassengerId	Survived	Pclass	Name	Sex	Age
275	276	True	1	Andrews, Miss. Kornelia Theodosia	female	63.0
483	484	True	3	Turkula, Mrs. (Hedwig)	female	63.0
829	830	True	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0