Data Manipulation with Pandas - Part 2

IC152 Feb 2021

Indices can cause confusion

```
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
```

Explicit integer index

```
In [66]: data
Out[66]:
1    a
3    b
5    c
dtype: object
```

```
In [65]: data[1]
Out[65]: 'a'
```

Explicit index used here

```
In [67]: data[0:1]
Out[67]:
1    a
dtype: object
```

Implicit index while slicing

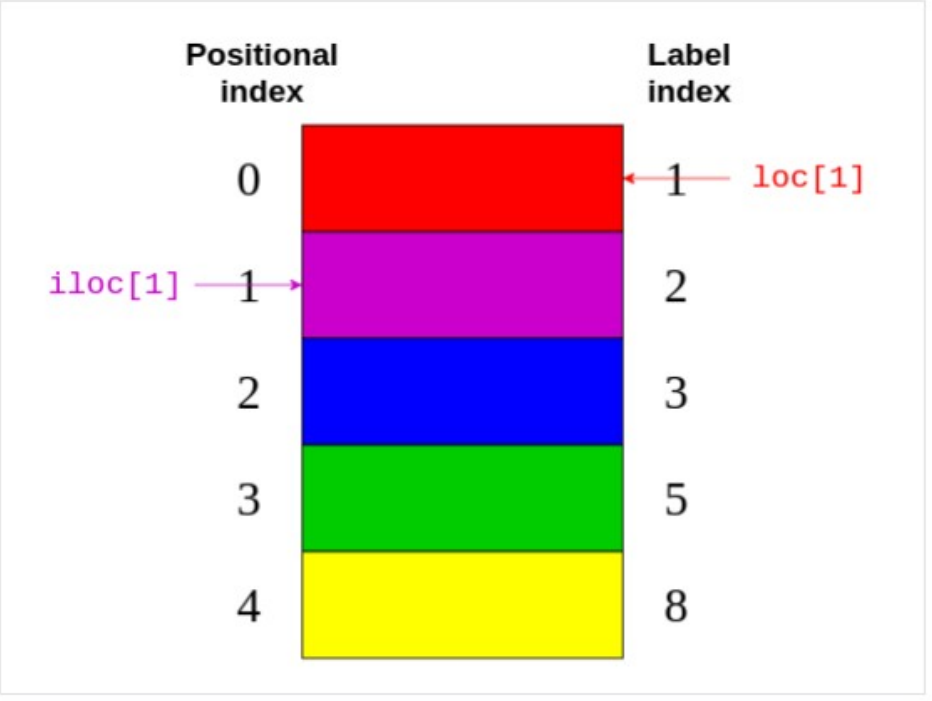
```
In [68]: data.loc[1]
Out[68]: 'a'

In [69]: data.iloc[1]
Out[69]: 'b'
```

Explicit index

Implicit index

Use `loc` and `iloc` indexers

```
13
14  colors = pd.Series(["red", "purple", "blue", "green", "yellow"],\
15  index=[1, 2, 3, 5, 8])
16
```



From: https://realpython.com/pandas-python-explore-dataset

DataFrame as a dictionary

```
50
51  area = pd.Series({'California': 423967, 'Texas': 695662,\
52  'New York': 141297, 'Florida': 170312, 'Illinois': 149995})
53
54  pop = pd.Series({'California': 38332521, 'Texas': 26448193,\
55  'New York': 19651127, 'Florida': 19552860,'Illinois': 12882135})
56
57  data = pd.DataFrame({'area':area, 'pop':pop})
58
```

```
In [71]: data
Out[71]:
             area       pop
California  423967  38332521
Texas       695662  26448193
New York    141297  19651127
Florida     170312  19552860
Illinois    149995  12882135
```

```
In [72]: data['density'] = data['pop'] / data['area']
```

Access using column names

```
In [73]: data
Out[73]:
             area       pop     density
California  423967  38332521   90.413926
Texas       695662  26448193   38.018740
New York    141297  19651127  139.076746
Florida     170312  19552860  114.806121
Illinois    149995  12882135   85.883763
```

Added new column

DF as a 2-D array

```
In [73]: data
Out[73]:
              area       pop     density
California  423967  38332521   90.413926
Texas       695662  26448193   38.018740
New York    141297  19651127  139.076746
Florida     170312  19552860  114.806121
Illinois    149995  12882135   85.883763
```

```
In [74]: data.values
Out[74]:
array([[4.23967000e+05, 3.83325210e+07, 9.04139261e+01],
       [6.95662000e+05, 2.64481930e+07, 3.80187404e+01],
       [1.41297000e+05, 1.96511270e+07, 1.39076746e+02],
       [1.70312000e+05, 1.95528600e+07, 1.14806121e+02],
       [1.49995000e+05, 1.28821350e+07, 8.58837628e+01]])
```

Returns ndarray

```
In [75]: data.values[0]
Out[75]: array([4.23967000e+05, 3.83325210e+07, 9.04139261e+01])
```

One row is returned

```
In [73]: data
Out[73]:
              area       pop     density
California  423967  38332521   90.413926
Texas       695662  26448193   38.018740
New York    141297  19651127  139.076746
Florida     170312  19552860  114.806121
Illinois    149995  12882135   85.883763
```

`loc` and `iloc` indexers

```
In [76]: data
Out[76]:
              area       pop     density
California  423967  38332521   90.413926
Texas       695662  26448193   38.018740
New York    141297  19651127  139.076746
Florida     170312  19552860  114.806121
Illinois    149995  12882135   85.883763

In [77]: data.iloc[:3, :2]
Out[77]:
              area       pop
California  423967  38332521
Texas       695662  26448193
New York    141297  19651127

In [78]: data.loc[:'Florida', :'pop']
Out[78]:
              area       pop
California  423967  38332521
Texas       695662  26448193
New York    141297  19651127
Florida     170312  19552860
```

```
58
59  rng = np.random.RandomState(42)
60  ser = pd.Series(rng.randint(0, 10, 4))
61
62  df = pd.DataFrame(rng.randint(0, 10, (3, 4)),\
63  columns=['A', 'B', 'C', 'D'])
64
```

```
In [82]: ser
Out[82]:
0    6
1    3
2    7
3    4
dtype: int64
```

```
In [84]: df
Out[84]:
   A  B  C  D
0  6  9  2  6
1  7  4  3  7
2  7  2  5  4
```

Indices and columns are preserved while applying Ufuncs

```
In [85]: np.exp(ser)
Out[85]:
0     403.428793
1      20.085537
2    1096.633158
3      54.598150
dtype: float64
```

```
In [88]: np.sin(df * np.pi / 4)
Out[88]:
          A             B         C             D
0 -1.000000  7.071068e-01  1.000000 -1.000000e+00
1 -0.707107  1.224647e-16  0.707107 -7.071068e-01
2 -0.707107  1.000000e+00 -0.707107  1.224647e-16
```

```
65
66  area = pd.Series({'Alaska': 1723337, 'Texas': 695662,\
67  'California': 423967}, name='area')
68  population = pd.Series({'California': 38332521, 'Texas': 26448193,\
69  'New York': 19651127}, name='population')
```

```
In [90]: area
Out[90]:
Alaska         1723337
Texas           695662
California      423967
Name: area, dtype: int64

In [91]: population
Out[91]:
California    38332521
Texas         26448193
New York      19651127
Name: population, dtype: int64

In [92]: population/area
Out[92]:
Alaska              NaN
California    90.413926
New York            NaN
Texas         38.018740
dtype: float64
```

NaN indicates missing values

```
70
71  A = pd.Series([2, 4, 6], index=[0, 1, 2])
72  B = pd.Series([1, 3, 5], index=[1, 2, 3])
73
```

```
In [94]: A+B
Out[94]:
0    NaN
1    5.0
2    9.0
3    NaN
dtype: float64
```

Index alignment for `Series`

```
74  A = pd.DataFrame(rng.randint(0, 20, (2, 2)),\
75  columns=list('AB'))
76
77  B = pd.DataFrame(rng.randint(0, 10, (3, 3)),\
78  columns=list('BAC'))
```

```
80  fill = A.stack().mean()
81  A.add(B, fill_value=fill)
82
```

```
In [96]: A
Out[96]:
   A   B
0  1  11
1  5   1

In [97]: B
Out[97]:
   B  A  C
0  4  0  9
1  5  8  0
2  9  2  6

In [98]: A+B
Out[98]:
      A     B   C
0   1.0  15.0 NaN
1  13.0   6.0 NaN
2   NaN   NaN NaN
```

Alignment in both columns and indices in DF

```
In [102]: A.add(B, fill_value=fill)
Out[102]:
      A     B     C
0   1.0  15.0  13.5
1  13.0   6.0   4.5
2   6.5  13.5  10.5
```

Fill values

```
In [78]: titanic = pd.read_csv("titanic.csv")

In [79]: type(titanic)
Out[79]: pandas.core.frame.DataFrame

In [80]: titanic
Out[80]:
     PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0              1         0       3  ...   7.2500   NaN         S
1              2         1       1  ...  71.2833   C85         C
2              3         1       3  ...   7.9250   NaN         S
3              4         1       1  ...  53.1000  C123         S
4              5         0       3  ...   8.0500   NaN         S
..           ...       ...     ...  ...      ...   ...       ...
886          887         0       2  ...  13.0000   NaN         S
887          888         1       1  ...  30.0000   B42         S
888          889         0       3  ...  23.4500   NaN         S
889          890         1       1  ...  30.0000  C148         C
890          891         0       3  ...   7.7500   NaN         Q

[891 rows x 12 columns]
```

Most of this material is from pandas.pydata.org/docs

```
In [81]: titanic.head()
Out[81]:
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S

[5 rows x 12 columns]

In [82]: titanic.tail(7)
Out[82]:
     PassengerId  Survived  Pclass  ...    Fare Cabin  Embarked
884          885         0       3  ...   7.050   NaN         S
885          886         0       3  ...  29.125   NaN         Q
886          887         0       2  ...  13.000   NaN         S
887          888         1       1  ...  30.000   B42         S
888          889         0       3  ...  23.450   NaN         S
889          890         1       1  ...  30.000  C148         C
890          891         0       3  ...   7.750   NaN         Q

[7 rows x 12 columns]
```

```
In [83]: titanic.columns
Out[83]:
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')

In [84]: titanic.dtypes
Out[84]:
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Generate descriptive statistics

```
In [86]: titanic.describe()
Out[86]:
       PassengerId    Survived      Pclass  ...       SibSp       Parch        Fare
count   891.000000  891.000000  891.000000  ...  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642  ...    0.523008    0.381594   32.204208
std     257.353842    0.486592    0.836071  ...    1.102743    0.806057   49.693429
min       1.000000    0.000000    1.000000  ...    0.000000    0.000000    0.000000
25%     223.500000    0.000000    2.000000  ...    0.000000    0.000000    7.910400
50%     446.000000    0.000000    3.000000  ...    0.000000    0.000000   14.454200
75%     668.500000    1.000000    3.000000  ...    1.000000    0.000000   31.000000
max     891.000000    1.000000    3.000000  ...    8.000000    6.000000  512.329200

[8 rows x 7 columns]
```

Select specific columns

```
In [87]: df_1 = titanic[["Age", "Sex"]]

In [88]: df_1.shape
Out[88]: (891, 2)

In [89]: df_1
Out[89]:
      Age     Sex
0    22.0    male
1    38.0  female
2    26.0  female
3    35.0  female
4    35.0    male
..    ...     ...
886  27.0    male
887  19.0  female
888   NaN  female
889  26.0    male
890  32.0    male

[891 rows x 2 columns]
```

Select specific rows

```
25  # passengers above 35 years of age
26  above_35 = titanic[titanic["Age"] > 35]
27  above_35.head()
```

```
In [99]: above_35.shape
Out[99]: (217, 12)
```

Creates a `Series` of boolean values

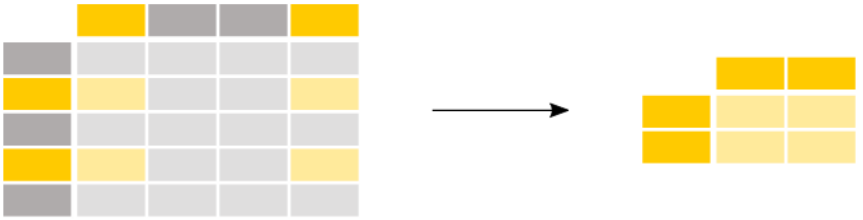
```
In [92]: titanic["Age"] > 35
Out[92]:
0      False
1       True
2      False
3      False
4      False
       ...
886    False
887    False
888    False
889    False
890    False
Name: Age, Length: 891, dtype: bool
```

```
29
30  # Titanic passengers from cabin class 2 and 3.
31  class_23 = titanic[titanic["Pclass"].isin([2, 3])]
32
33
34  # this does the same thing
35  class_23 = titanic[(titanic["Pclass"] == 2) | (titanic["Pclass"] == 3)]
```

```
In [95]: class_23.shape
Out[95]: (675, 12)
```

- Multiple conditional statements must be surrounded by ()
- Do not use And or Or operator, need to use | or &

Choosing specific rows and columns



```
41
42  # names of the passengers older than 35 years.
43  adult_names = titanic.loc[titanic["Age"] > 35, "Name"]
44
```

Selects the rows

Selects the columns

When boolean indexing is used, the `loc` indexer selects rows with True values

```
In [124]: titanic.loc[10:15,['PassengerId','Pclass','Name']]
Out[124]:
    PassengerId  Pclass                                Name
10           11       3       Sandstrom, Miss. Marguerite Rut
11           12       1              Bonnell, Miss. Elizabeth
12           13       3        Saundercock, Mr. William Henry
13           14       3           Andersson, Mr. Anders Johan
14           15       3  Vestrom, Miss. Hulda Amanda Adolfina
15           16       2      Hewlett, Mrs. (Mary D Kingcome)
```

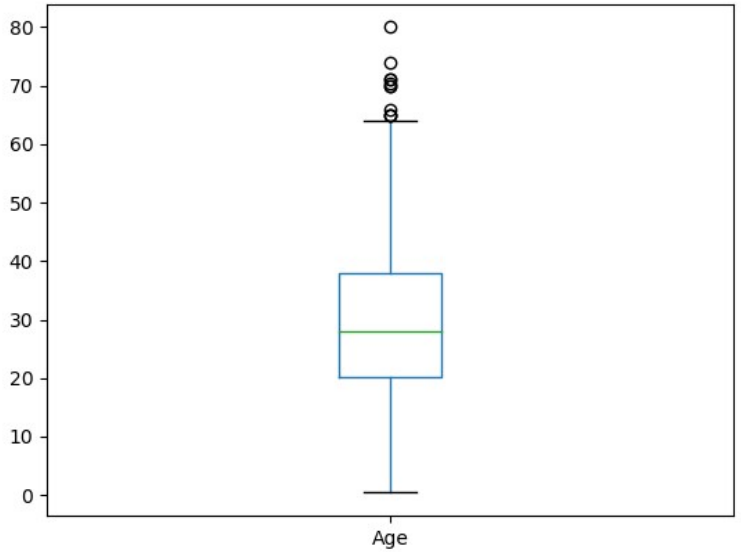
Label indexing with `loc`

```
In [135]: temp = titanic[titanic['Age']>35]

In [136]: temp['Name']
Out[136]:
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
6                                 McCarthy, Mr. Timothy J
11                               Bonnell, Miss. Elizabeth
13                            Andersson, Mr. Anders Johan
15                      Hewlett, Mrs. (Mary D Kingcome)
                               ...
865                              Bystrom, Mrs. (Karolina)
871     Beckwith, Mrs. Richard Leonard (Sallie Monypeny)
873                            Vander Cruyssen, Mr. Victor
879        Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)
885                   Rice, Mrs. William (Margaret Norton)
Name: Name, Length: 217, dtype: object
```

You can also do it like this

```
In [138]: t = titanic[titanic['Age']>35]['Name']
```

```
In [142]: titanic['Age'].plot.box()
Out[142]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb7444e01f0>
```



Pandas supports plotting, based on Matplotlib

Aggregation

```
In [149]: titanic['Age'].min()
Out[149]: 0.42

In [150]: titanic['Age'].max()
Out[150]: 80.0
```

```
In [167]: titanic['Age'].value_counts()
Out[167]:
24.00    30
22.00    27
18.00    26
19.00    25
30.00    25
         ..
55.50     1
70.50     1
66.00     1
23.50     1
0.42      1
Name: Age, Length: 88, dtype: int64
```

Other aggregations: grouping

```
In [159]: titanic.groupby('Pclass')['Age'].mean()
Out[159]:
Pclass
1    38.233441
2    29.877630
3    25.140620
Name: Age, dtype: float64
```

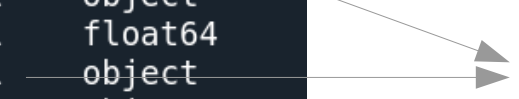
```
In [168]: titanic[titanic['Pclass']==1]['Age'].mean()
Out[168]: 38.233440860215055
```

```
In [169]: nba.columns
Out[169]:
Index(['gameorder', 'game_id', 'lg_id', '_iscopy', 'year_id',
'date_game',
       'seasongame', 'is_playoffs', 'team_id', 'fran_id', 'pts', 'elo_i',
       'elo_n', 'win_equiv', 'opp_id', 'opp_fran', 'opp_pts',
'opp_elo_i',
       'opp_elo_n', 'game_location', 'game_result', 'forecast', 'notes'],
      dtype='object')
```

```
 8  import numpy as np
 9  import pandas as pd
10
11  nba = pd.read_csv('nbaallelo.csv')
12
13  nba[(nba["fran_id"] == "Spurs") & (nba["year_id"] > 2010)].\
14  groupby(["year_id", "game_result"])["game_id"].count()
15
```

```
Out[171]:
year_id  game_result
2011     L              25
         W              63
2012     L              20
         W              60
2013     L              30
         W              73
2014     L              27
         W              78
2015     L              31
         W              58
Name: game_id, dtype: int64
```

```
In [172]: titanic.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```



Several missing values

`df.dropna()`	To drop a row having missing values
`df.dropna(axis=1)`	To drop a column having missing values

`df.fillna()`	Replace nan with some default value

```
In [173]: tit = titanic.copy()

In [174]: tit['Cabin'].fillna(value=0,inplace=True)

In [175]: tit.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        891 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Other issues:

Invalid values
Inconsistent values

Eg. IC152, IC-152, IC 152