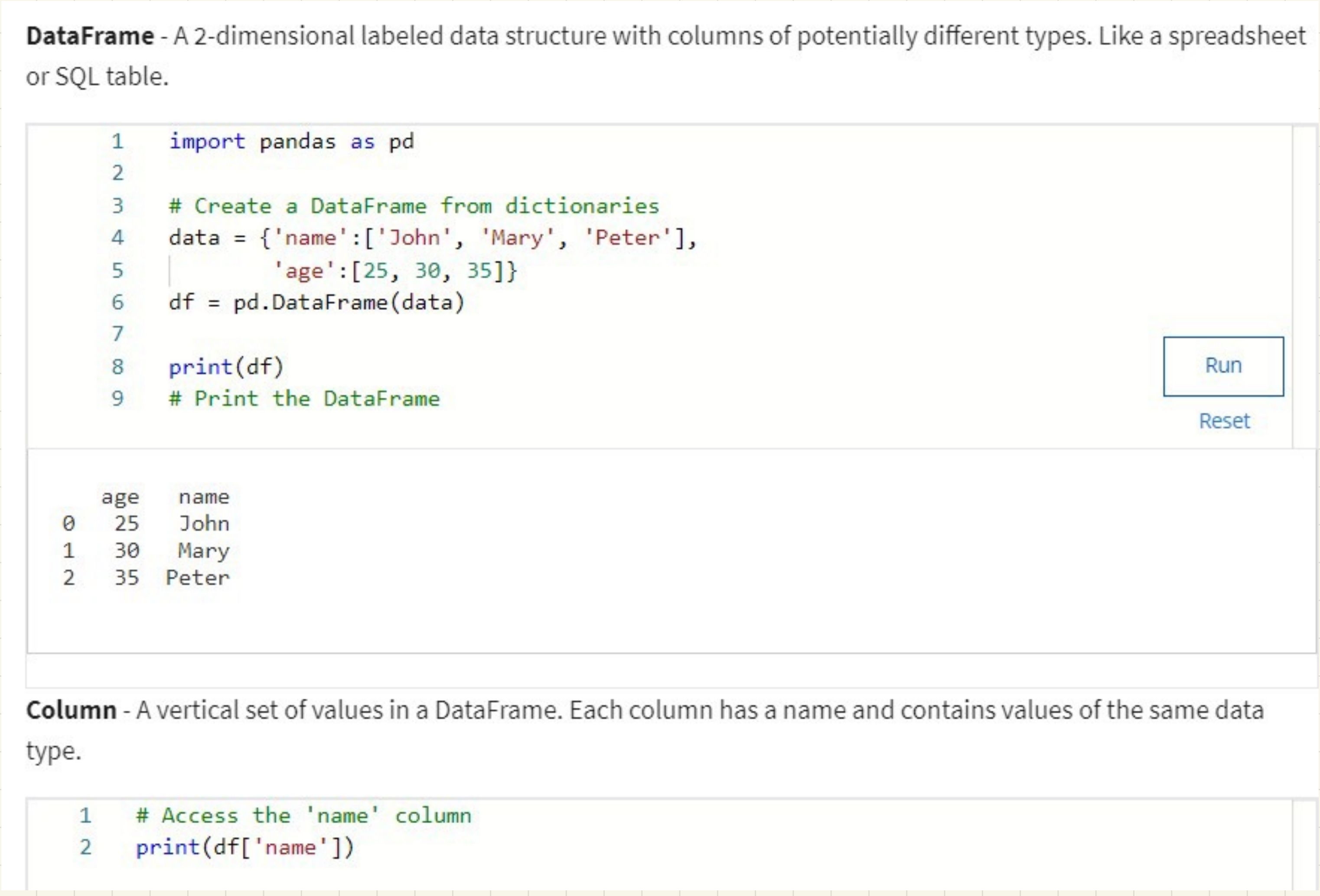
Key Term



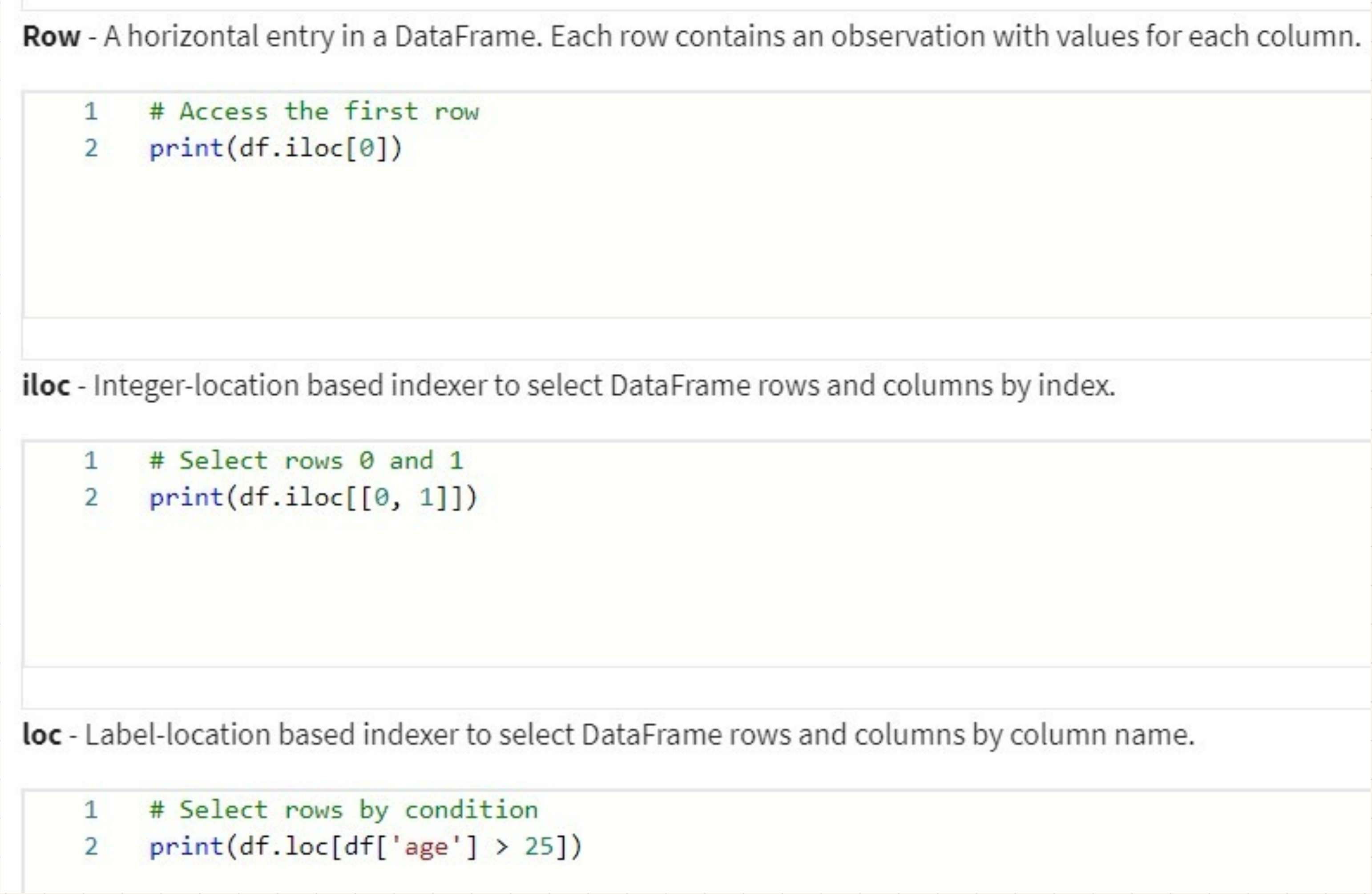
DataFrame - A 2-dimensional labeled data structure with columns of potentially different types. Like a spreadsheet or SQL table.

```
import pandas as pd

# Create a DataFrame from dictionaries
data = {'name':['John', 'Mary', 'Peter'],
        'age':[25, 30, 35]}
df = pd.DataFrame(data)

print(df)
# Print the DataFrame
```

```
   age   name
0   25   John
1   30   Mary
2   35  Peter
```

Column - A vertical set of values in a DataFrame. Each column has a name and contains values of the same data type.

```
# Access the 'name' column
print(df['name'])
```



Row - A horizontal entry in a DataFrame. Each row contains an observation with values for each column.

```
# Access the first row
print(df.iloc[0])
```

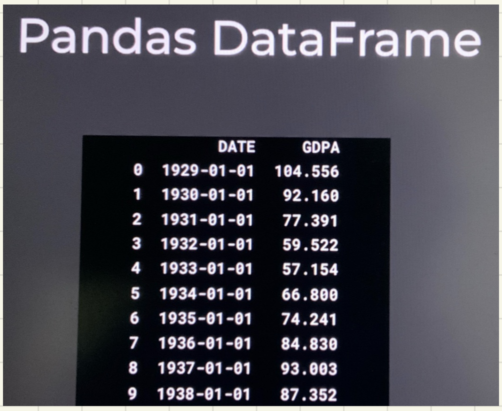
iloc - Integer-location based indexer to select DataFrame rows and columns by index.

```
# Select rows 0 and 1
print(df.iloc[[0, 1]])
```

loc - Label-location based indexer to select DataFrame rows and columns by column name.

```
# Select rows by condition
print(df.loc[df['age'] > 25])
```

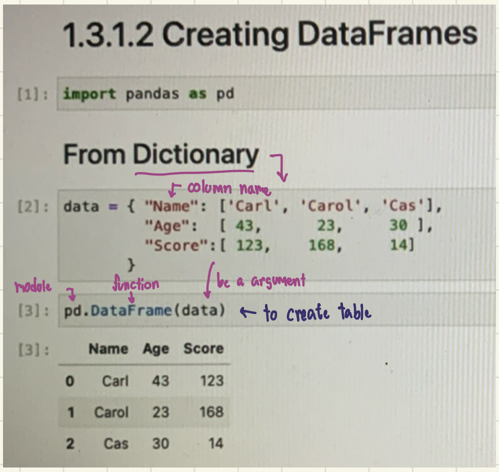
Creating Pandas DataFrame in Python



Pandas DataFrame

	DATE	GDPA
0	1929-01-01	104.556
1	1930-01-01	92.160
2	1931-01-01	77.391
3	1932-01-01	59.522
4	1933-01-01	57.154
5	1934-01-01	66.800
6	1935-01-01	74.241
7	1936-01-01	84.830
8	1937-01-01	93.003
9	1938-01-01	87.352

- hold data in two dimensional ,table-like structure
- The structure consists of columns and rows ,maybe labeled
- can be mixed types in DataFrames , each column is defined with a single type
- The data can be accessed by column or by row or both



1.3.1.2 Creating DataFrames

```
[1]: import pandas as pd
```

From Dictionary

column name

```
[2]: data = { "Name": ['Carl', 'Carol', 'Cas'],
             "Age":  [ 43,      23,      30 ],
             "Score":[ 123,     168,     14]
           }
```

module

function

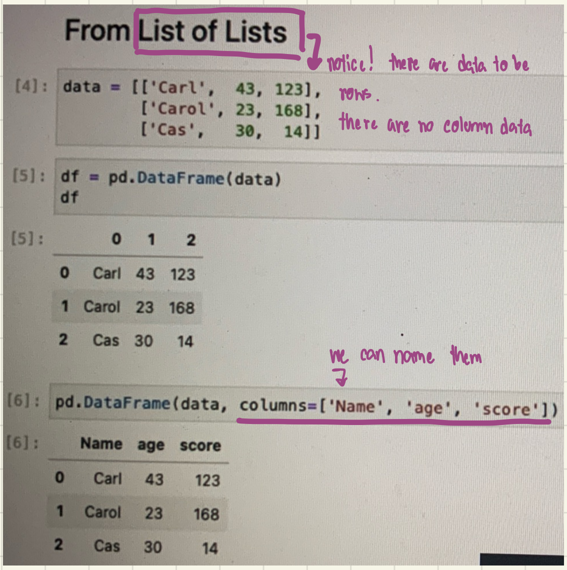
be a argument

```
[3]: pd.DataFrame(data)
```

← to create table

[3]:

	Name	Age	Score
0	Carl	43	123
1	Carol	23	168
2	Cas	30	14



From List of Lists

notice! there are data to be rows.
there are no column data

```
[4]: data = [['Carl',  43, 123],
             ['Carol', 23, 168],
             ['Cas',   30,  14]]
```

```
[5]: df = pd.DataFrame(data)
     df
```

[5]:

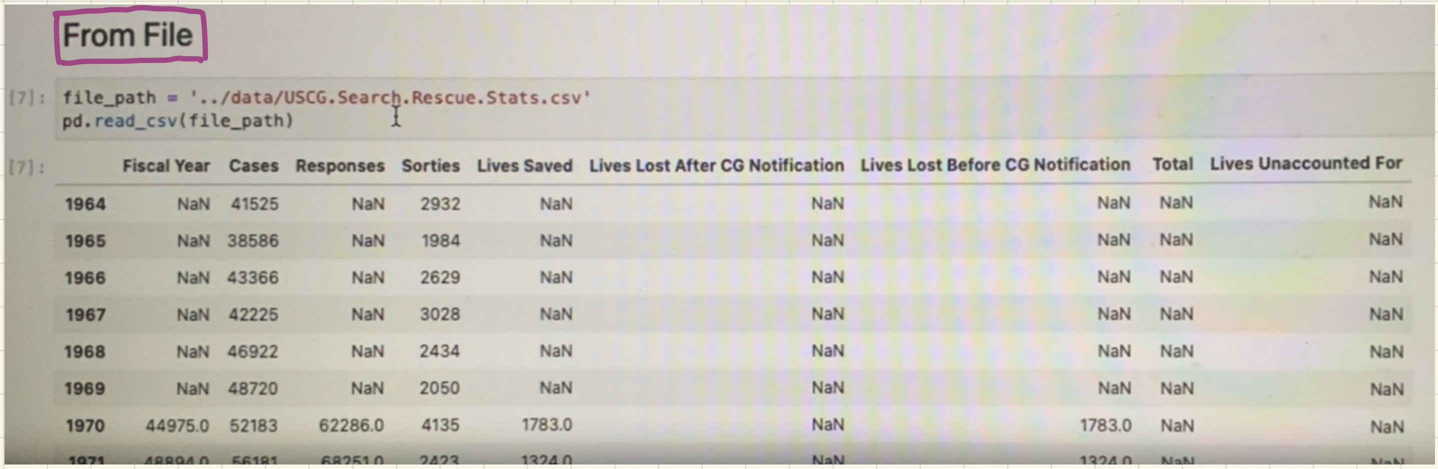
	0	1	2
0	Carl	43	123
1	Carol	23	168
2	Cas	30	14

we can name them

```
[6]: pd.DataFrame(data, columns=['Name', 'age', 'score'])
```

[6]:

	Name	age	score
0	Carl	43	123
1	Carol	23	168
2	Cas	30	14



From File

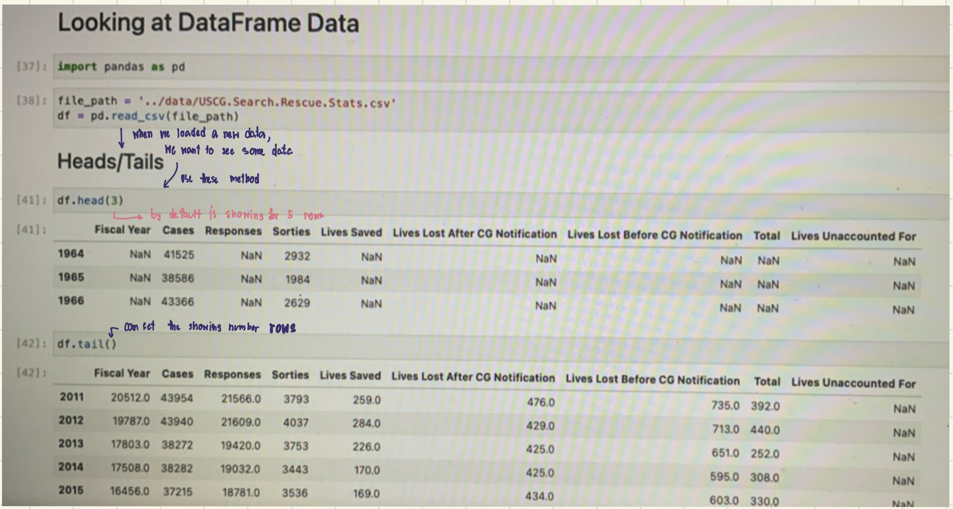
```
[7]: file_path = '../data/USCG.Search.Rescue.Stats.csv'
     pd.read_csv(file_path)
```

[7]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1964	NaN	41525	NaN	2932	NaN	NaN	NaN	NaN	NaN
1965	NaN	38586	NaN	1984	NaN	NaN	NaN	NaN	NaN
1966	NaN	43366	NaN	2629	NaN	NaN	NaN	NaN	NaN
1967	NaN	42225	NaN	3028	NaN	NaN	NaN	NaN	NaN
1968	NaN	46922	NaN	2434	NaN	NaN	NaN	NaN	NaN
1969	NaN	48720	NaN	2050	NaN	NaN	NaN	NaN	NaN
1970	44975.0	52183	62286.0	4135	1783.0	NaN	1783.0	NaN	NaN
1971	48894.0	56181	68251.0	2423	1324.0	NaN	1324.0	NaN	NaN

Investigating Data in a Pandas DataFrame

Looking:



Looking at DataFrame Data

```
[37]: import pandas as pd
```

```
[38]: file_path = '../data/USCG.Search.Rescue.Stats.csv'
      df = pd.read_csv(file_path)
```

when we loaded a new data,
we want to see some date

Heads/Tails

use these method

```
[41]: df.head(3)
```

by default is showing for 5 rows

[41]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1964	NaN	41525	NaN	2932	NaN	NaN	NaN	NaN	NaN
1965	NaN	38586	NaN	1984	NaN	NaN	NaN	NaN	NaN
1966	NaN	43366	NaN	2629	NaN	NaN	NaN	NaN	NaN

can set the showing number rows

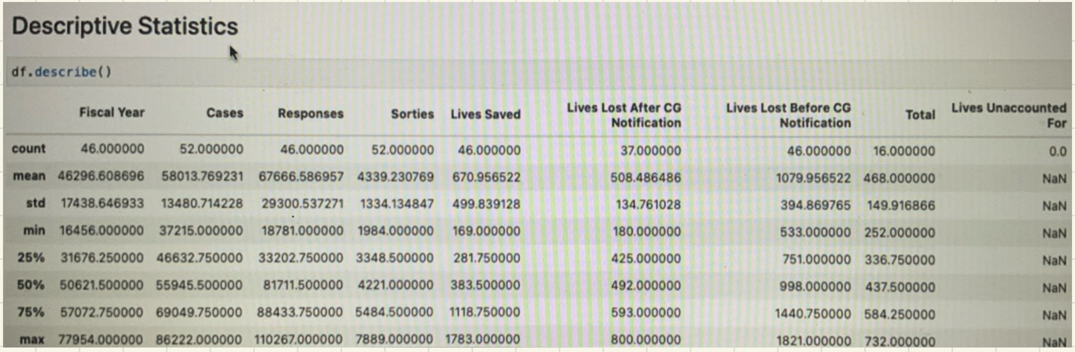
```
[42]: df.tail()
```

[42]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
2011	20512.0	43954	21566.0	3793	259.0	476.0	735.0	392.0	NaN
2012	19787.0	43940	21609.0	4037	284.0	429.0	713.0	440.0	NaN
2013	17803.0	38272	19420.0	3753	226.0	425.0	651.0	252.0	NaN
2014	17508.0	38282	19032.0	3443	170.0	425.0	595.0	308.0	NaN
2015	16456.0	37215	18781.0	3536	169.0	434.0	603.0	330.0	NaN

Describe:

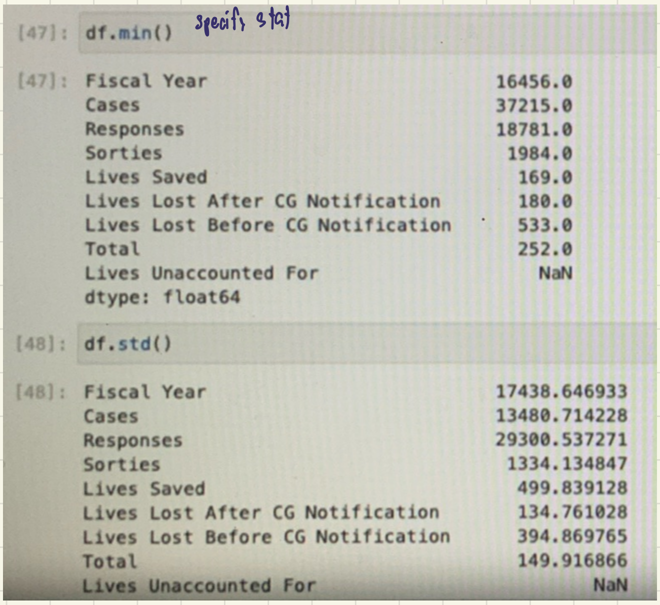
show all
statistic information



Descriptive Statistics

```
df.describe()
```

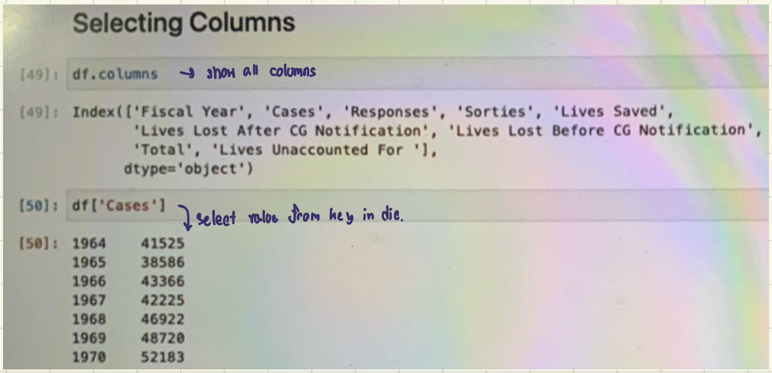
	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
count	46.000000	52.000000	46.000000	52.000000	46.000000	37.000000	46.000000	16.000000	0.0
mean	46296.608696	58013.769231	67666.586957	4339.230769	670.956522	508.486486	1079.956522	468.000000	NaN
std	17438.646933	13480.714228	29300.537271	1334.134847	499.839128	134.761028	394.869765	149.916866	NaN
min	16456.000000	37215.000000	18781.000000	1984.000000	169.000000	180.000000	533.000000	252.000000	NaN
25%	31676.250000	46632.750000	33202.750000	3348.500000	281.750000	425.000000	751.000000	336.750000	NaN
50%	50621.500000	55945.500000	81711.500000	4221.000000	383.500000	492.000000	998.000000	437.500000	NaN
75%	57072.750000	69049.750000	88433.750000	5484.500000	1118.750000	593.000000	1440.750000	584.250000	NaN
max	77954.000000	86222.000000	110267.000000	7889.000000	1783.000000	800.000000	1821.000000	732.000000	NaN



specify stat

```
[47]: df.min()

[47]: Fiscal Year                              16456.0
      Cases                                    37215.0
      Responses                                18781.0
      Sorties                                   1984.0
      Lives Saved                                169.0
      Lives Lost After CG Notification           180.0
      Lives Lost Before CG Notification          533.0
      Total                                      252.0
      Lives Unaccounted For                        NaN
      dtype: float64

[48]: df.std()

[48]: Fiscal Year                          17438.646933
      Cases                                13480.714228
      Responses                            29300.537271
      Sorties                               1334.134847
      Lives Saved                            499.839128
      Lives Lost After CG Notification       134.761028
      Lives Lost Before CG Notification      394.869765
      Total                                  149.916866
      Lives Unaccounted For                         NaN
```

select:



Selecting Columns

```
[49]: df.columns
```

→ show all columns

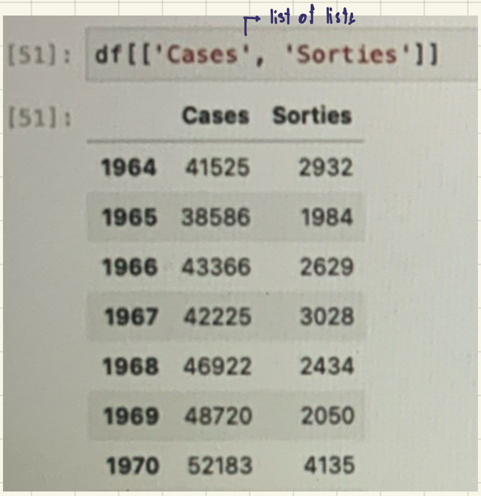
```
[49]: Index(['Fiscal Year', 'Cases', 'Responses', 'Sorties', 'Lives Saved',
             'Lives Lost After CG Notification', 'Lives Lost Before CG Notification',
             'Total', 'Lives Unaccounted For '],
            dtype='object')
```

```
[50]: df['Cases']
```

select value from key in dic.

```
[50]: 1964    41525
      1965    38586
      1966    43366
      1967    42225
      1968    46922
      1969    48720
      1970    52183
```

Selecting
multiple columns

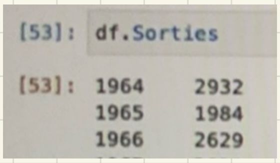


list of lists

```
[51]: df[['Cases', 'Sorties']]
```

[51]:

	Cases	Sorties
1964	41525	2932
1965	38586	1984
1966	43366	2629
1967	42225	3028
1968	46922	2434
1969	48720	2050
1970	52183	4135



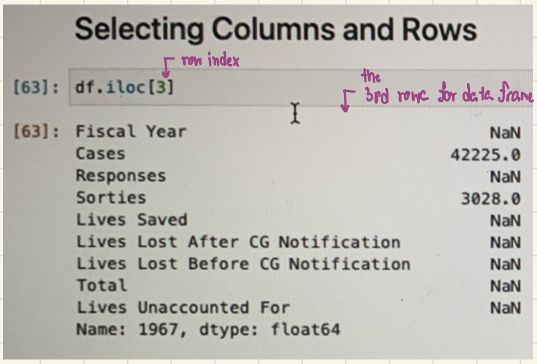
```
[53]: df.Sorties

[53]: 1964    2932
      1965    1984
      1966    2629
```

Selecting directly
by dot

Cont. Investigating Data in a Pandas DataFrame

iloc

- dealing with large amounts of data
- select based on indexes

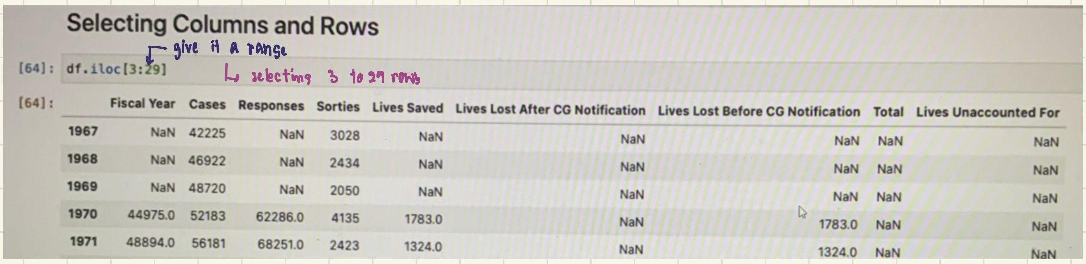


Selecting Columns and Rows

row index

the 3rd row for data frame

```
[63]: df.iloc[3]

[63]: Fiscal Year                              NaN
      Cases                                42225.0
      Responses                                NaN
      Sorties                               3028.0
      Lives Saved                              NaN
      Lives Lost After CG Notification         NaN
      Lives Lost Before CG Notification        NaN
      Total                                    NaN
      Lives Unaccounted For                    NaN
      Name: 1967, dtype: float64
```



Selecting Columns and Rows

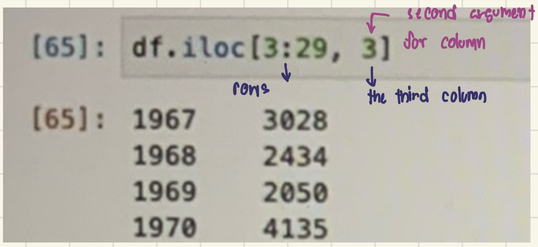
give it a range

```
[64]: df.iloc[3:29]
```

selecting 3 to 29 rows

[64]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1967	NaN	42225	NaN	3028	NaN	NaN	NaN	NaN	NaN
1968	NaN	46922	NaN	2434	NaN	NaN	NaN	NaN	NaN
1969	NaN	48720	NaN	2050	NaN	NaN	NaN	NaN	NaN
1970	44975.0	52183	62286.0	4135	1783.0	NaN	1783.0	NaN	NaN
1971	48894.0	56181	68251.0	2423	1324.0	NaN	1324.0	NaN	NaN

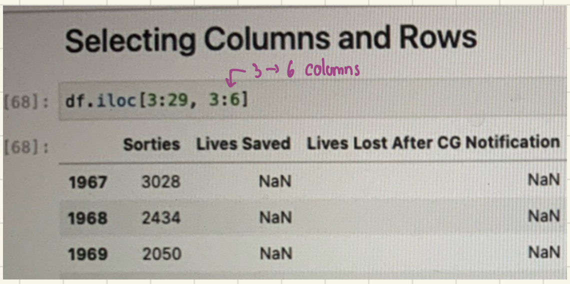


```
[65]: df.iloc[3:29, 3]

[65]: 1967     3028
      1968     2434
      1969     2050
      1970     4135
```

second argument for column

rows

the third column



Selecting Columns and Rows

3 → 6 columns

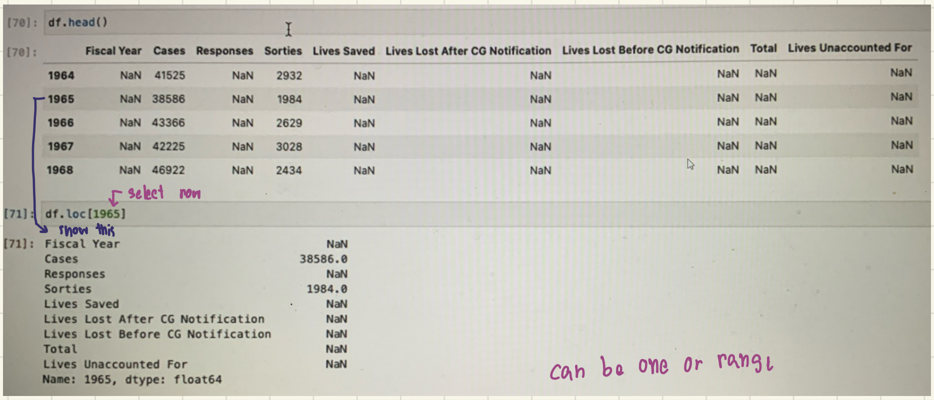
```
[68]: df.iloc[3:29, 3:6]
```

[68]:

	Sorties	Lives Saved	Lives Lost After CG Notification
1967	3028	NaN	NaN
1968	2434	NaN	NaN
1969	2050	NaN	NaN

loc

- uses labels names



```
[70]: df.head()
```

[70]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1964	NaN	41525	NaN	2932	NaN	NaN	NaN	NaN	NaN
1965	NaN	38586	NaN	1984	NaN	NaN	NaN	NaN	NaN
1966	NaN	43366	NaN	2629	NaN	NaN	NaN	NaN	NaN
1967	NaN	42225	NaN	3028	NaN	NaN	NaN	NaN	NaN
1968	NaN	46922	NaN	2434	NaN	NaN	NaN	NaN	NaN

select row

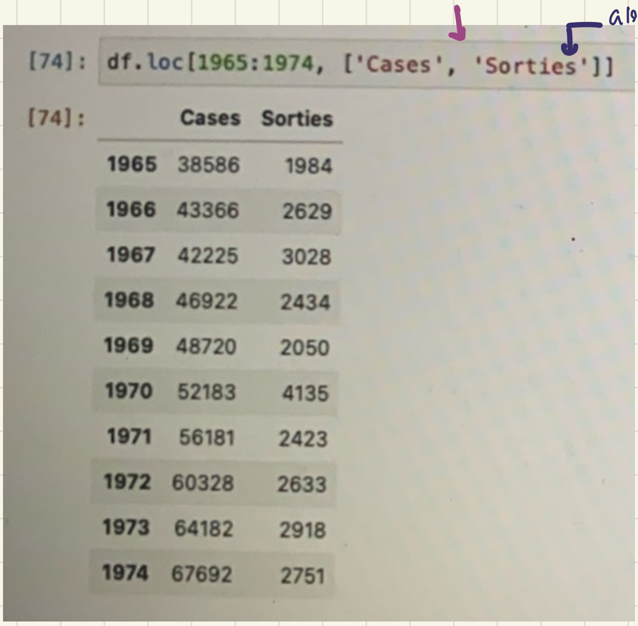
```
[71]: df.loc[1965]
```

show this

```
[71]: Fiscal Year                              NaN
      Cases                                38586.0
      Responses                                NaN
      Sorties                               1984.0
      Lives Saved                              NaN
      Lives Lost After CG Notification         NaN
      Lives Lost Before CG Notification        NaN
      Total                                    NaN
      Lives Unaccounted For                    NaN
      Name: 1965, dtype: float64
```

can be one or range

also like iloc
but used its label
not index,
use it for both rows and colomns

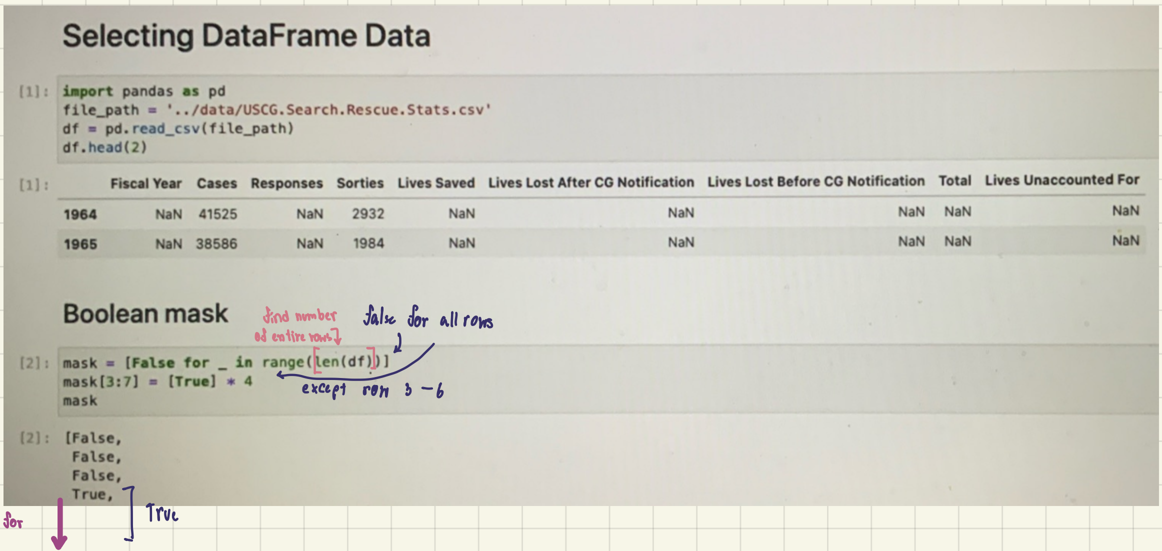


```
[74]: df.loc[1965:1974, ['Cases', 'Sorties']]
```

[74]:

	Cases	Sorties
1965	38586	1984
1966	43366	2629
1967	42225	3028
1968	46922	2434
1969	48720	2050
1970	52183	4135
1971	56181	2423
1972	60328	2633
1973	64182	2918
1974	67692	2751

Selecting Data in a Pandas DataFrame



Selecting DataFrame Data

```
[1]: import pandas as pd
     file_path = '../data/USCG.Search.Rescue.Stats.csv'
     df = pd.read_csv(file_path)
     df.head(2)
```

[1]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1964	NaN	41525	NaN	2932	NaN	NaN	NaN	NaN	NaN
1965	NaN	38586	NaN	1984	NaN	NaN	NaN	NaN	NaN

Boolean mask

find number of entire rows

false for all rows

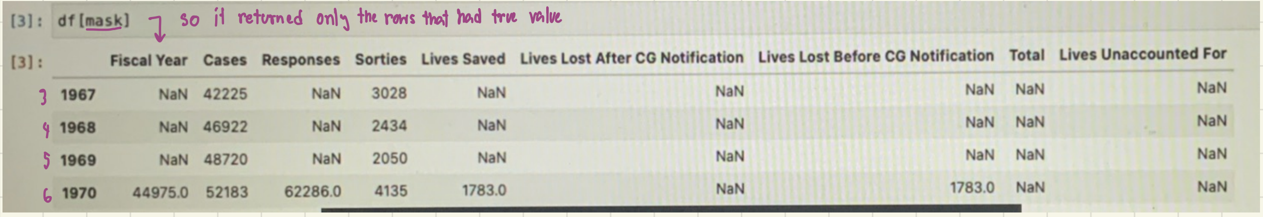
```
[2]: mask = [False for _ in range(len(df))]
     mask[3:7] = [True] * 4
     mask
```

except row 3 – 6

```
[2]: [False,
      False,
      False,
      True,
```

True

use this mark for an argument

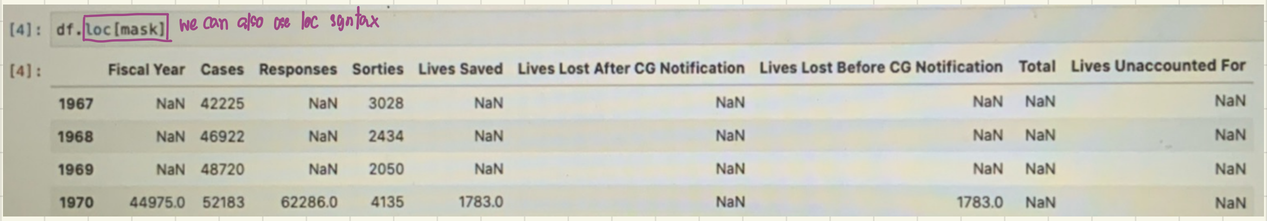


```
[3]: df[mask]
```

so it returned only the rows that had true value

[3]:

		Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
3	1967	NaN	42225	NaN	3028	NaN	NaN	NaN	NaN	NaN
4	1968	NaN	46922	NaN	2434	NaN	NaN	NaN	NaN	NaN
5	1969	NaN	48720	NaN	2050	NaN	NaN	NaN	NaN	NaN
6	1970	44975.0	52183	62286.0	4135	1783.0	NaN	1783.0	NaN	NaN

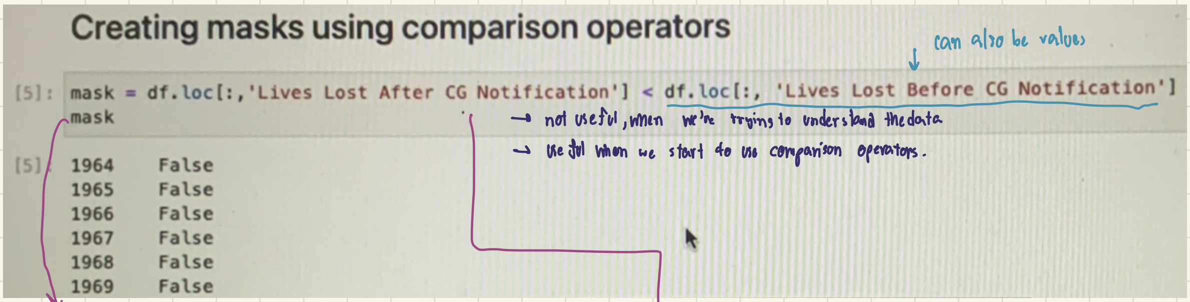


```
[4]: df.loc[mask]
```

we can also use loc syntax

[4]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1967	NaN	42225	NaN	3028	NaN	NaN	NaN	NaN	NaN
1968	NaN	46922	NaN	2434	NaN	NaN	NaN	NaN	NaN
1969	NaN	48720	NaN	2050	NaN	NaN	NaN	NaN	NaN
1970	44975.0	52183	62286.0	4135	1783.0	NaN	1783.0	NaN	NaN



Creating masks using comparison operators

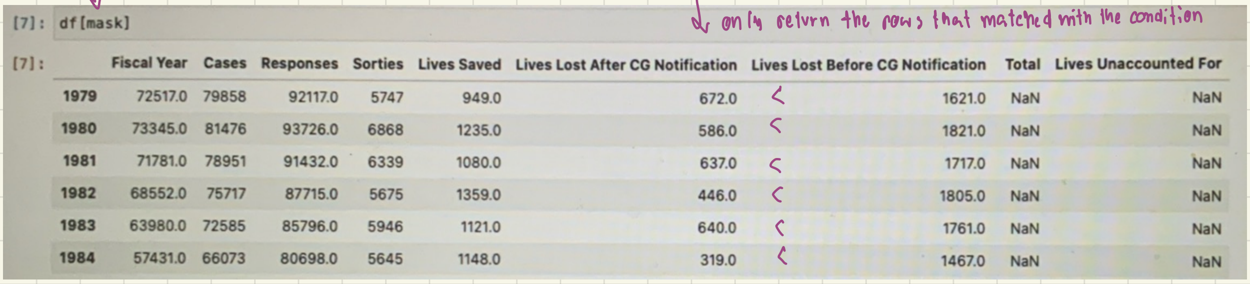
can also be values

```
[5]: mask = df.loc[:,'Lives Lost After CG Notification'] < df.loc[:, 'Lives Lost Before CG Notification']
     mask
```

→ not useful, when we're trying to understand the data

→ useful when we start to use comparison operators.

```
[5]: 1964    False
     1965    False
     1966    False
     1967    False
     1968    False
     1969    False
```



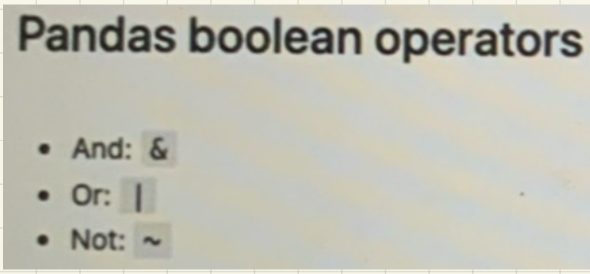
```
[7]: df[mask]
```

only return the rows that matched with the condition

[7]:

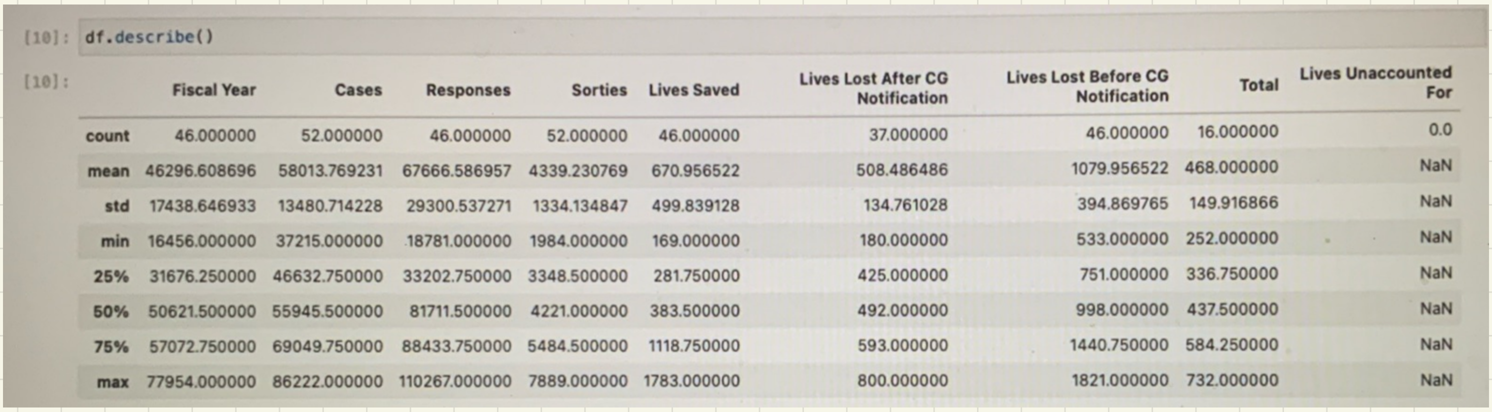
	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification		Lives Lost Before CG Notification	Total	Lives Unaccounted For
1979	72517.0	79858	92117.0	5747	949.0	672.0	<	1621.0	NaN	NaN
1980	73345.0	81476	93726.0	6868	1235.0	586.0	<	1821.0	NaN	NaN
1981	71781.0	78951	91432.0	6339	1080.0	637.0	<	1717.0	NaN	NaN
1982	68552.0	75717	87715.0	5675	1359.0	446.0	<	1805.0	NaN	NaN
1983	63980.0	72585	85796.0	5946	1121.0	640.0	<	1761.0	NaN	NaN
1984	57431.0	66073	80698.0	5645	1148.0	319.0	<	1467.0	NaN	NaN

Cont. Selecting Data in a Pandas DataFrame



Pandas boolean operators

- And: `&`
- Or: `|`
- Not: `~`

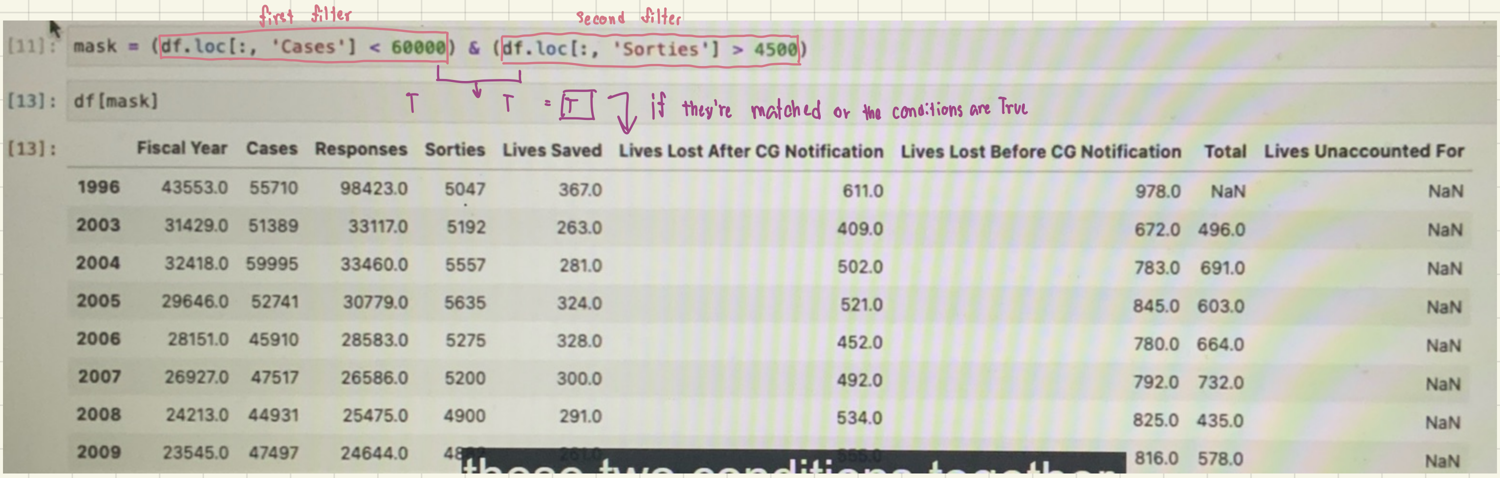


```
[10]: df.describe()
```

[10]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
count	46.000000	52.000000	46.000000	52.000000	46.000000	37.000000	46.000000	16.000000	0.0
mean	46296.608696	58013.769231	67666.586957	4339.230769	670.956522	508.486486	1079.956522	468.000000	NaN
std	17438.646933	13480.714228	29300.537271	1334.134847	499.839128	134.761028	394.869765	149.916866	NaN
min	16456.000000	37215.000000	18781.000000	1984.000000	169.000000	180.000000	533.000000	252.000000	NaN
25%	31676.250000	46632.750000	33202.750000	3348.500000	281.750000	425.000000	751.000000	336.750000	NaN
50%	50621.500000	55945.500000	81711.500000	4221.000000	383.500000	492.000000	998.000000	437.500000	NaN
75%	57072.750000	69049.750000	88433.750000	5484.500000	1118.750000	593.000000	1440.750000	584.250000	NaN
max	77954.000000	86222.000000	110267.000000	7889.000000	1783.000000	800.000000	1821.000000	732.000000	NaN

first filter　　　　second filter



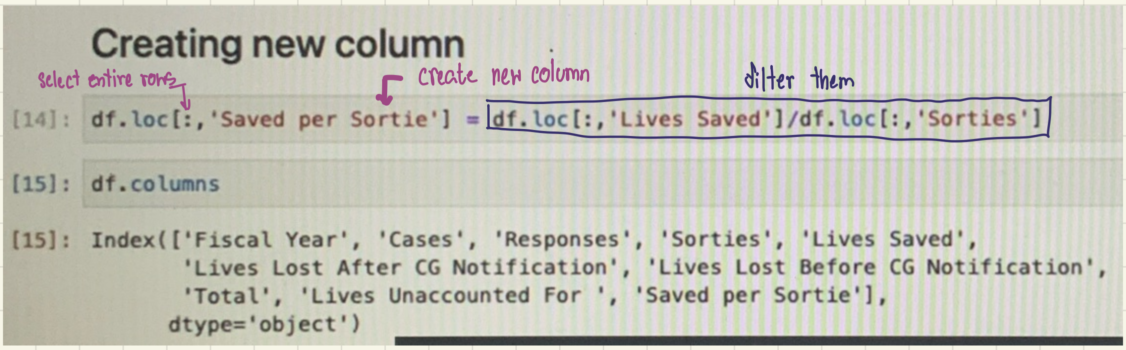
```
[11]: mask = (df.loc[:, 'Cases'] < 60000) & (df.loc[:, 'Sorties'] > 4500)

[13]: df[mask]
```

T ↓ T = $\boxed{T}$ ↓ if they're matched or the conditions are True

[13]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1996	43553.0	55710	98423.0	5047	367.0	611.0	978.0	NaN	NaN
2003	31429.0	51389	33117.0	5192	263.0	409.0	672.0	496.0	NaN
2004	32418.0	59995	33460.0	5557	281.0	502.0	783.0	691.0	NaN
2005	29646.0	52741	30779.0	5635	324.0	521.0	845.0	603.0	NaN
2006	28151.0	45910	28583.0	5275	328.0	452.0	780.0	664.0	NaN
2007	26927.0	47517	26586.0	5200	300.0	492.0	792.0	732.0	NaN
2008	24213.0	44931	25475.0	4900	291.0	534.0	825.0	435.0	NaN
2009	23545.0	47497	24644.0	[illegible]	[illegible]	[illegible]	816.0	578.0	NaN

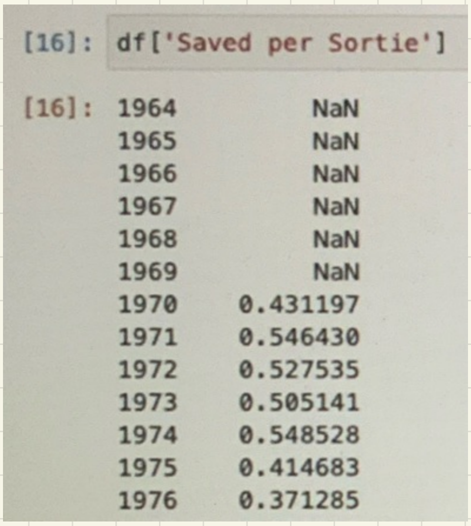
these two conditions together



Creating new column

select entire rows　　create new column　　filter them

```
[14]: df.loc[:,'Saved per Sortie'] = df.loc[:,'Lives Saved']/df.loc[:,'Sorties']

[15]: df.columns

[15]: Index(['Fiscal Year', 'Cases', 'Responses', 'Sorties', 'Lives Saved',
             'Lives Lost After CG Notification', 'Lives Lost Before CG Notification',
             'Total', 'Lives Unaccounted For ', 'Saved per Sortie'],
            dtype='object')
```



```
[16]: df['Saved per Sortie']

[16]: 1964         NaN
      1965         NaN
      1966         NaN
      1967         NaN
      1968         NaN
      1969         NaN
      1970    0.431197
      1971    0.546430
      1972    0.527535
      1973    0.505141
      1974    0.548528
      1975    0.414683
      1976    0.371285
```