DataFrame Basic Functions and Groupby

Johnny Zhang

Changing Column Names

- For either development or reporting purposes you will often want to rename columns.

```
# Import data into a DataFrame.
path = "/Users/pm/Desktop/DayDocs/2019_2020/PythonForDataAnalytics/workingData/babysamp-98.txt"
df = pd.read_csv(path, skiprows=1, sep='\t’, names=('MomAge', 'DadAge', 'MomEduc', 'MomMarital', 'numlive’,
"dobmm", 'gestation', 'sex', 'weight', 'prenatalstart’, 'orig.id', 'preemie'))
# Rename the columns so they are more reader-friendly.
df = df.rename({'MomAge': 'Mom Age', 'DadAge':'Dad Age',
          'MomEduc':'Mom Edu', 'weight':'Weight'}, axis=1)  # new method
# Show all columns.
pd.set_option('display.max_columns', None)
# Increase number of columns that display on one line.
pd.set_option('display.width', 1000)
print(df.head())
```

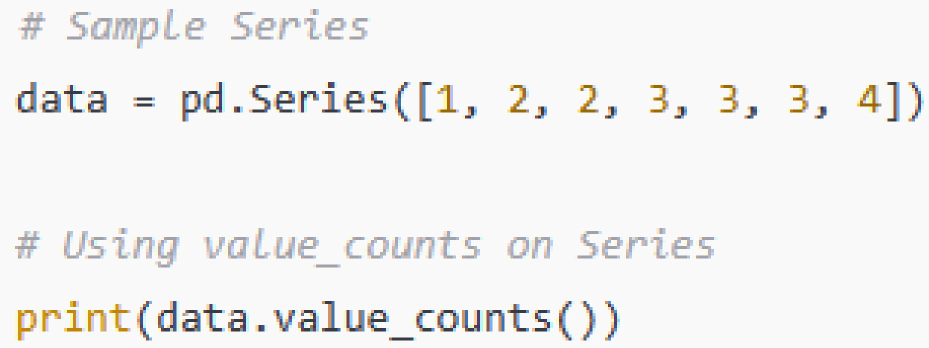
Exercise 1

Analyzing Frequencies (value_counts)

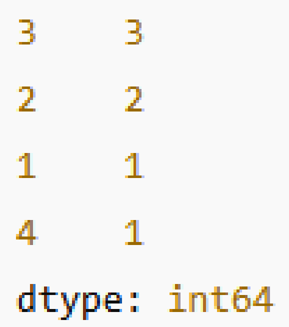
- With the value_counts function you can get a quick understanding of the column ranges and frequency for each value.
- count vs value_counts
  - value_counts: Category & count for each category
    - It is specifically for Series (single columns), but it can be used on a DataFrame by calling it on a specific column (df['column_name'].value_counts()).
  - count: total number of rows (It does not count NaN or None Value)

Series.value_counts(*normalize=False*, *sort=True*, *ascending=False*, *bins=None*, *dropna=True*)

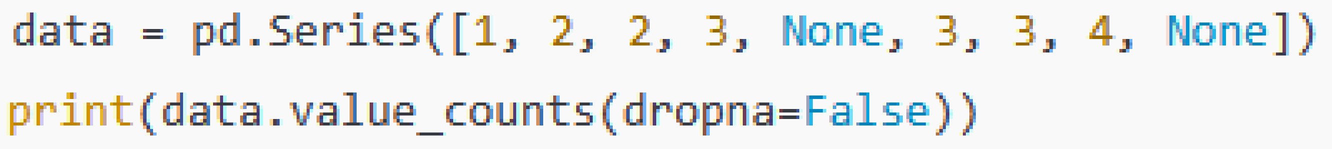
value_counts() vs. count()



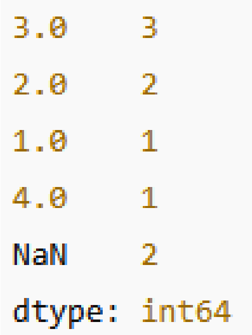
```
# Sample Series
data = pd.Series([1, 2, 2, 3, 3, 3, 4])

# Using value_counts on Series
print(data.value_counts())
```



```
3    3
2    2
1    1
4    1
dtype: int64
```

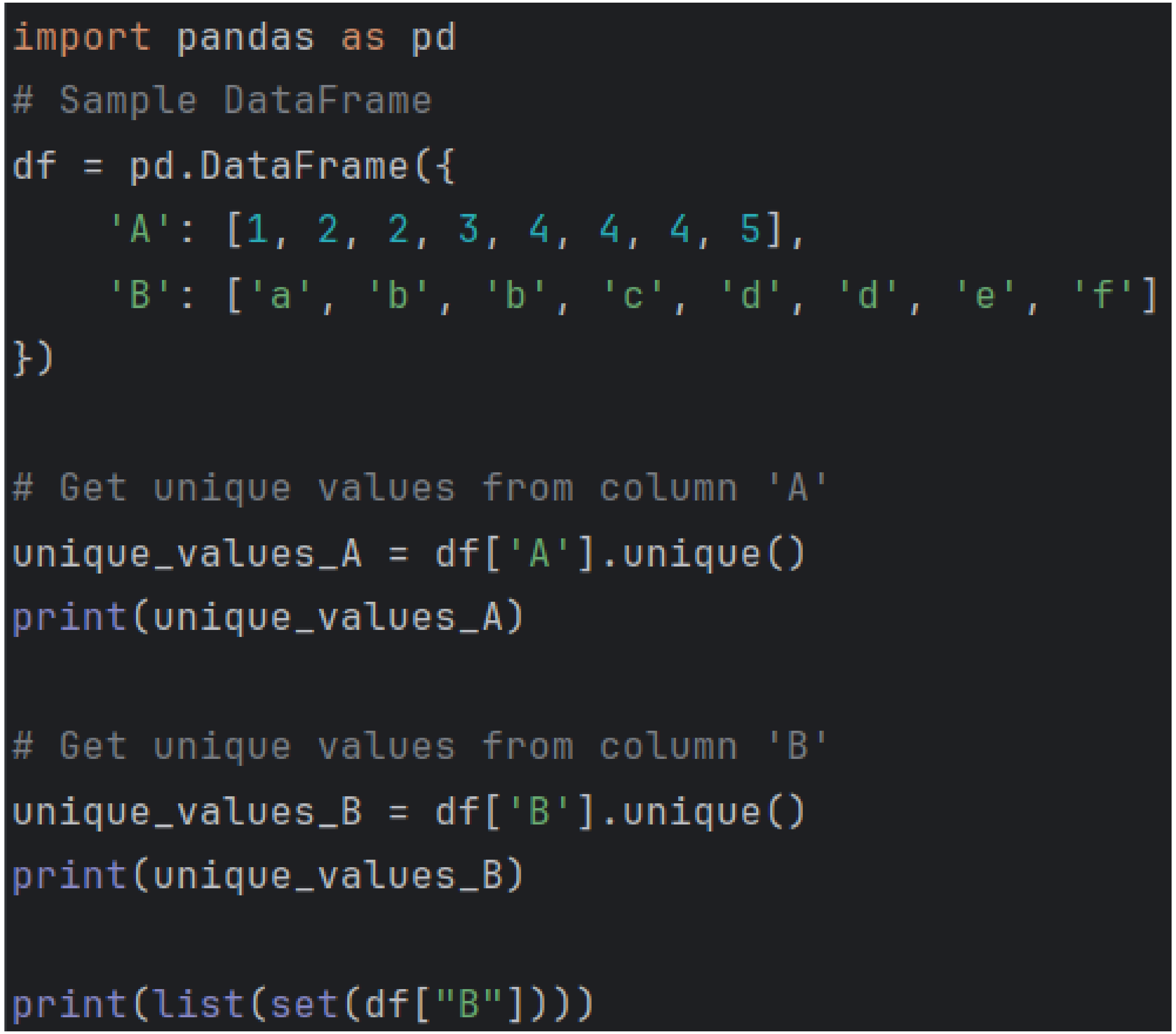


```
data = pd.Series([1, 2, 2, 3, None, 3, 3, 4, None])
print(data.value_counts(dropna=False))
```

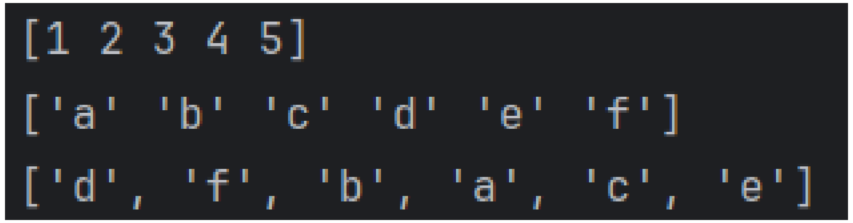


```
3.0    3
2.0    2
1.0    1
4.0    1
NaN    2
dtype: int64
```

It ignores NaN values by default but can include them if specified using the dropna=False parameter.



```
import pandas as pd
# Sample DataFrame
df = pd.DataFrame({
    'A': [1, 2, 2, 3, 4, 4, 4, 5],
    'B': ['a', 'b', 'b', 'c', 'd', 'd', 'e', 'f']
})

# Get unique values from column 'A'
unique_values_A = df['A'].unique()
print(unique_values_A)

# Get unique values from column 'B'
unique_values_B = df['B'].unique()
print(unique_values_B)

print(list(set(df["B"])))
```



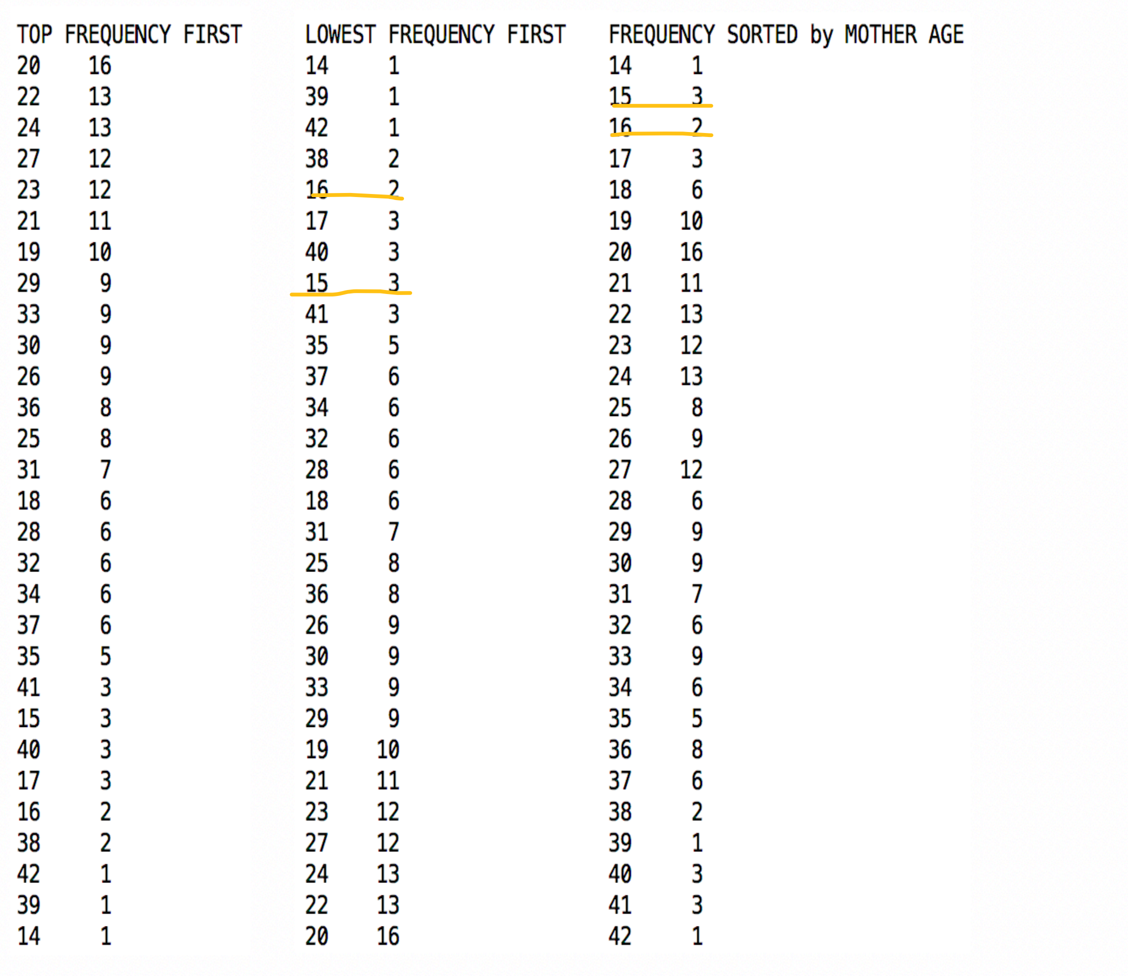
```
[1 2 3 4 5]
['a' 'b' 'c' 'd' 'e' 'f']
['d', 'f', 'b', 'a', 'c', 'e']
```

```
print("\nTOP FREQUENCY FIRST")
print(df['Mom Age'].value_counts())
```

```
print("\nLOWEST FREQUENCY FIRST")
print(df['Mom Age'].value_counts(ascending=True))
```

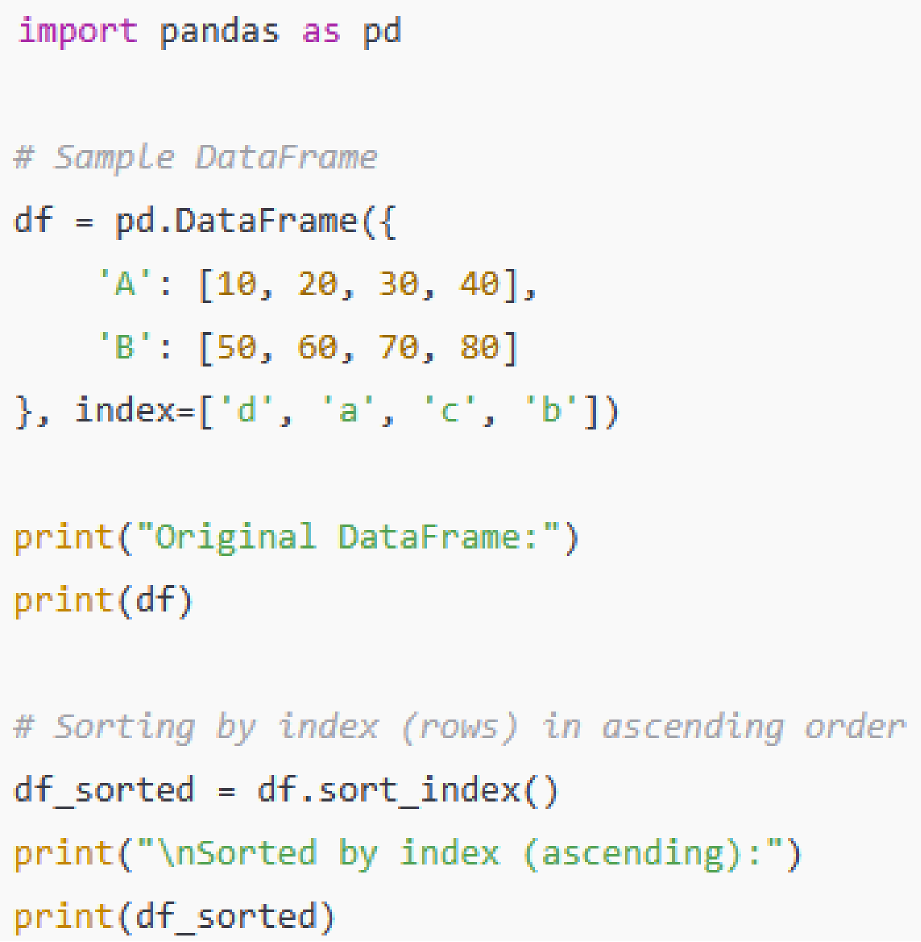
```
print("\nFREQUENCY SORTED by MOTHER AGE")
print(df['Mom Age'].value_counts().sort_index())
```

Exercise 2

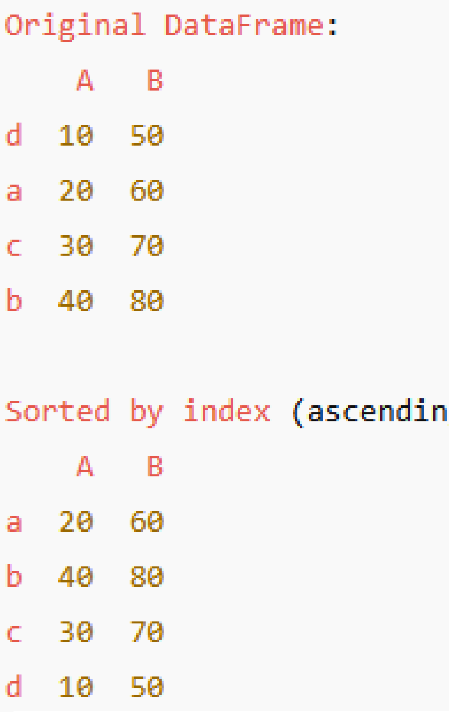


```
TOP FREQUENCY FIRST      LOWEST FREQUENCY FIRST   FREQUENCY SORTED by MOTHER AGE
20    16                 14     1                 14     1
22    13                 39     1                 15     3
24    13                 42     1                 16     2
27    12                 38     2                 17     3
23    12                 16     2                 18     6
21    11                 17     3                 19    10
19    10                 40     3                 20    16
29     9                 15     3                 21    11
33     9                 41     3                 22    13
30     9                 35     5                 23    12
26     9                 37     6                 24    13
36     8                 34     6                 25     8
25     8                 32     6                 26     9
31     7                 28     6                 27    12
18     6                 18     6                 28     6
28     6                 31     7                 29     9
32     6                 25     8                 30     9
34     6                 36     8                 31     7
37     6                 26     9                 32     6
35     5                 30     9                 33     9
41     3                 33     9                 34     6
15     3                 29     9                 35     5
40     3                 19    10                 36     8
17     3                 21    11                 37     6
16     2                 23    12                 38     2
38     2                 27    12                 39     1
42     1                 24    13                 40     3
39     1                 22    13                 41     3
14     1                 20    16                 42     1
```

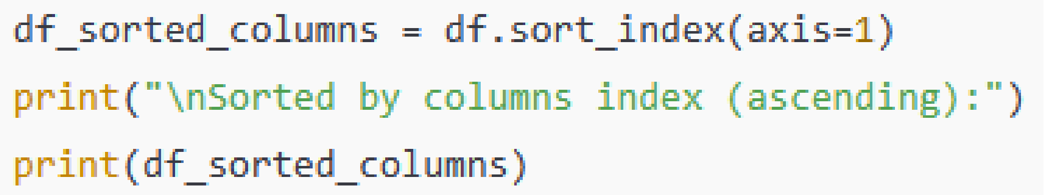
The sort_index() method in pandas is used to sort the rows or columns of a DataFrame or Series based on their index labels. You can sort the data in ascending or descending order.



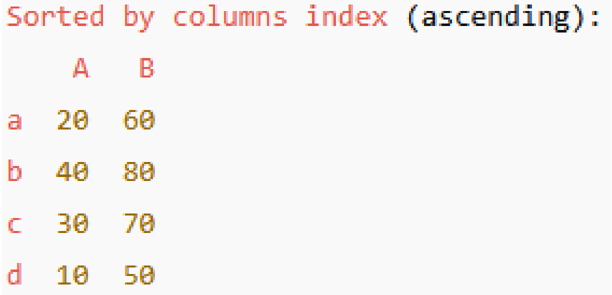
```
import pandas as pd

# Sample DataFrame
df = pd.DataFrame({
    'A': [10, 20, 30, 40],
    'B': [50, 60, 70, 80]
}, index=['d', 'a', 'c', 'b'])

print("Original DataFrame:")
print(df)

# Sorting by index (rows) in ascending order
df_sorted = df.sort_index()
print("\nSorted by index (ascending):")
print(df_sorted)
```



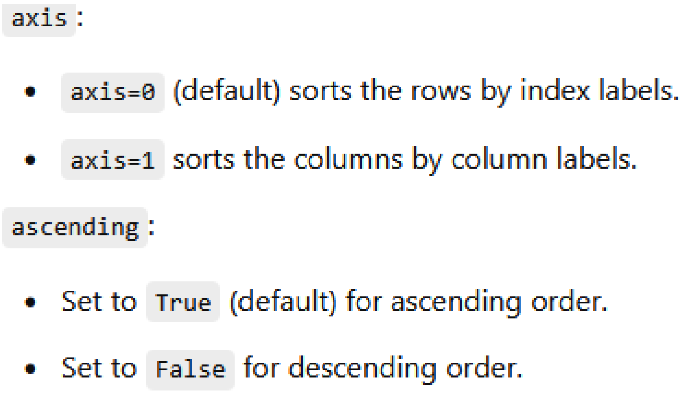
```
Original DataFrame:
    A   B
d  10  50
a  20  60
c  30  70
b  40  80

Sorted by index (ascending):
    A   B
a  20  60
b  40  80
c  30  70
d  10  50
```



```
df_sorted_columns = df.sort_index(axis=1)
print("\nSorted by columns index (ascending):")
print(df_sorted_columns)
```



```
Sorted by columns index (ascending):
    A   B
a  20  60
b  40  80
c  30  70
d  10  50
```



`axis`:

- `axis=0` (default) sorts the rows by index labels.
- `axis=1` sorts the columns by column labels.

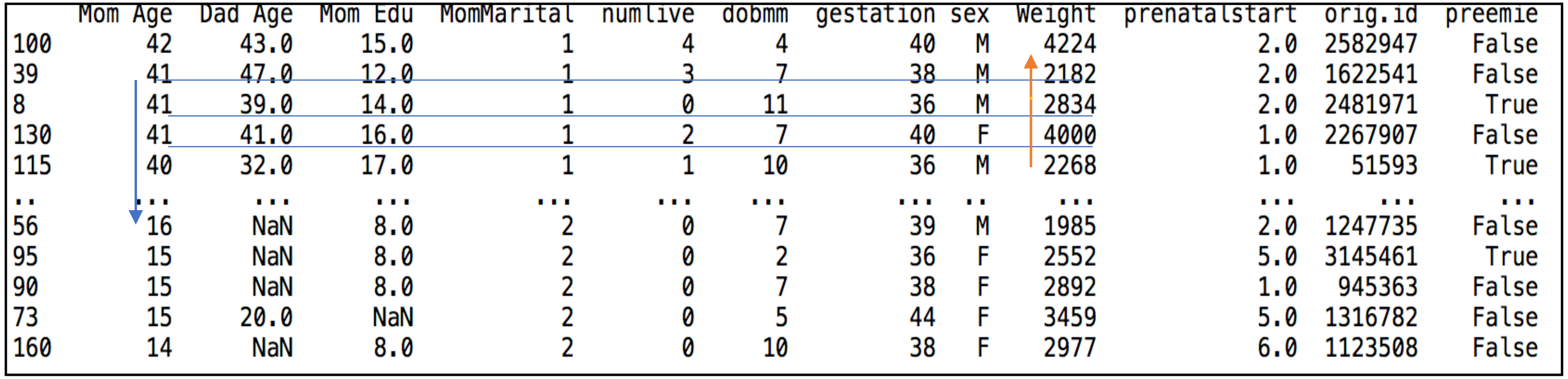
`ascending`:

- Set to `True` (default) for ascending order.
- Set to `False` for descending order.

Sorting a DataFrame (sort_values)

- The sort_values method allows us to sort DataFrame rows by multiple columns values in either ascending or descending order;

```
dfSorted = df.sort_values(['Mom Age', 'Weight'], ascending=[False, True])

print(dfSorted)
```



```
     Mom Age  Dad Age  Mom Edu  MomMarital  numlive  dobmm  gestation sex  Weight  prenatalstart  orig.id  preemie
100       42     43.0     15.0           1        4      4         40   M    4224            2.0  2582947    False
39        41     47.0     12.0           1        3      7         38   M    2182            2.0  1622541    False
8         41     39.0     14.0           1        0     11         36   M    2834            2.0  2481971     True
130       41     41.0     16.0           1        2      7         40   F    4000            1.0  2267907    False
115       40     32.0     17.0           1        1     10         36   M    2268            1.0    51593     True
..       ...      ...      ...         ...      ...    ...        ...  ..     ...            ...      ...      ...
56        16      NaN      8.0           2        0      7         39   M    1985            2.0  1247735    False
95        15      NaN      8.0           2        0      2         36   F    2552            5.0  3145461     True
90        15      NaN      8.0           2        0      7         38   F    2892            1.0   945363    False
73        15     20.0      NaN           2        0      5         44   F    3459            5.0  1316782    False
160       14      NaN      8.0           2        0     10         38   F    2977            6.0  1123508    False
```

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

# Import data into a DataFrame.
path = "/your/file/full/path/filename.file_extension"
df = pd.read_csv(path, skiprows=1,

                 sep='\t',
                 names=('MomAge', 'DadAge', 'MomEduc', 'MomMarital', 'numlive',
                    "dobmm", 'gestation', 'sex', 'weight', 'prenatalstart',
                    'orig.id', 'preemie'))

# Rename the columns so they are more reader-friendly.
df = df.rename({'MomAge': 'Mom Age', 'DadAge':'Dad Age',
             'MomEduc':'Mom Edu', 'weight':'Weight'}, axis=1)  # new method
# Show all columns.
pd.set_option('display.max_columns', None)

# Increase number of columns that display on one line.
pd.set_option('display.width', 1000)

# Sort by descending mother age and then by ascending weight.
dfSorted = df.sort_values(['Mom Age', 'Weight'], ascending=[False, True])
print(dfSorted)
```

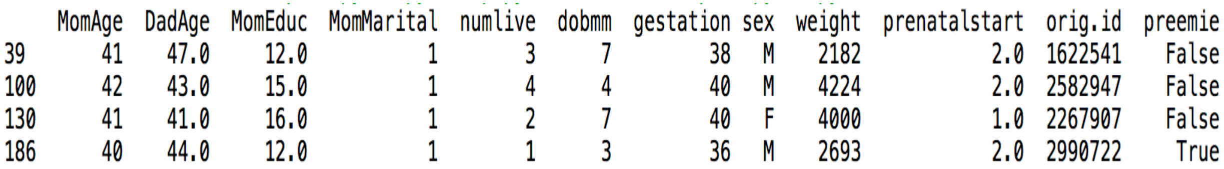
Exercise 3

Filtering a DataFrame

It is possible to query a DataFrame with conditional expressions and compound conditions:

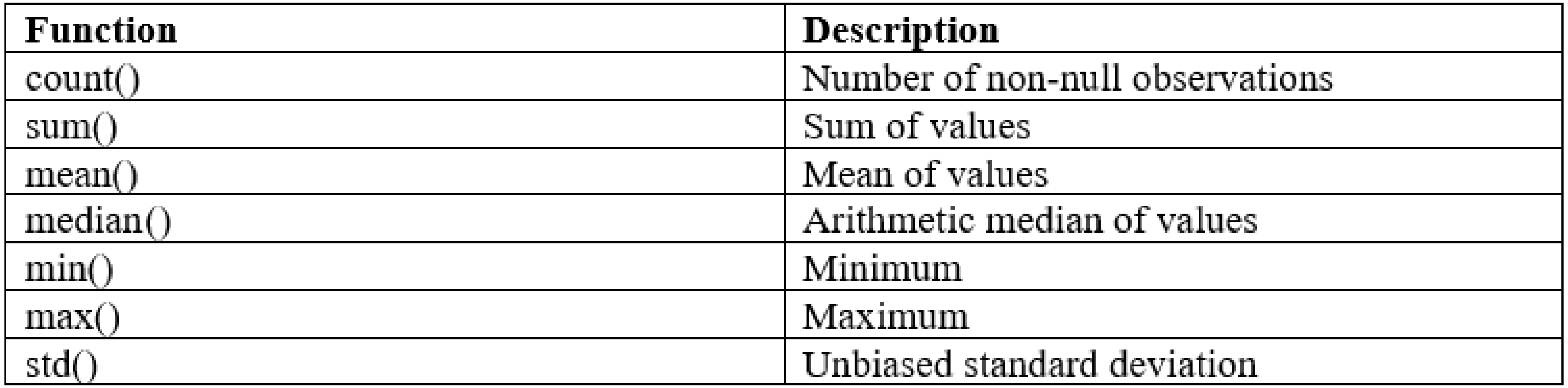
```
# Compound conditions require single '&' for 'AND'
# and single '|' for 'OR.
result = df[(df['DadAge']>40) & (df['MomAge'] > 30)]
result = df[(df['DadAge']>40) | (df['MomAge'] > 30)]
```

```
import matplotlib.pyplot as plt
import pandas as pd
# Import data into a DataFrame.
path = "/Users/pm/Desktop/DayDocs/2019_2020/PythonForDataAnalytics/workingData/babysamp-98.txt"
df = pd.read_csv(path, skiprows=1,
            sep='\t',
            names=('MomAge', 'DadAge', 'MomEduc', 'MomMarital', 'numlive',
               "dobmm", 'gestation', 'sex', 'weight', 'prenatalstart',
               'orig.id', 'preemie'))
# Show all columns.
pd.set_option('display.max_columns', None)
# Increase number of columns that display on one line.
pd.set_option('display.width', 1000)
# Compound conditions require single '&' for 'AND' and single '|' for 'OR.
resultDf = df[(df['DadAge']>=40) & (df['MomAge'] >= 40)]
print(resultDf)
```



```
     MomAge  DadAge  MomEduc  MomMarital  numlive  dobmm  gestation sex  weight  prenatalstart  orig.id  preemie
39       41    47.0     12.0           1        3      7         38   M    2182            2.0  1622541    False
100      42    43.0     15.0           1        4      4         40   M    4224            2.0  2582947    False
130      41    41.0     16.0           1        2      7         40   F    4000            1.0  2267907    False
186      40    44.0     12.0           1        1      3         36   M    2693            2.0  2990722     True
```

Numeric DataFrame Summaries

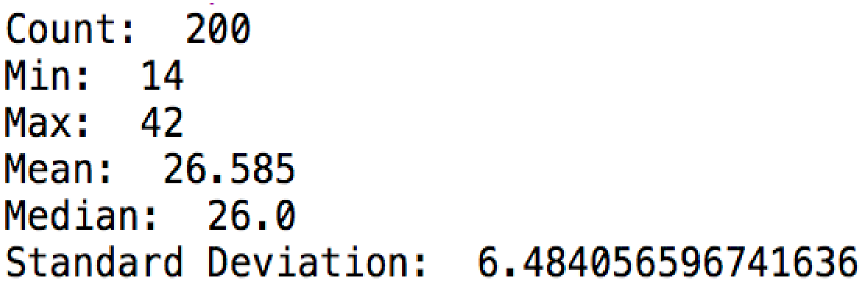


Function	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of values
median()	Arithmetic median of values
min()	Minimum
max()	Maximum
std()	Unbiased standard deviation

Syntax:

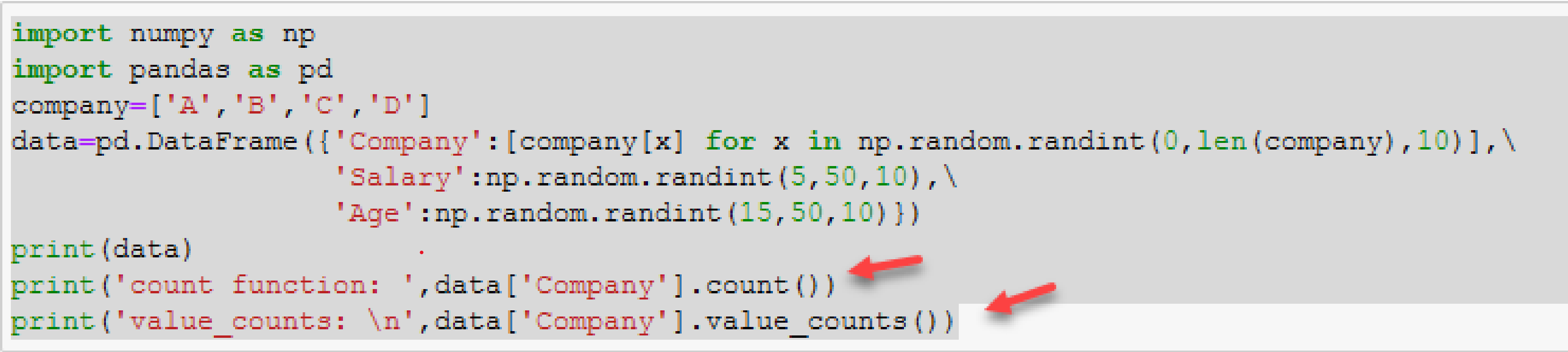
```
print(df['MomAge'].mean())
```

Exercise 4 (3 marks)

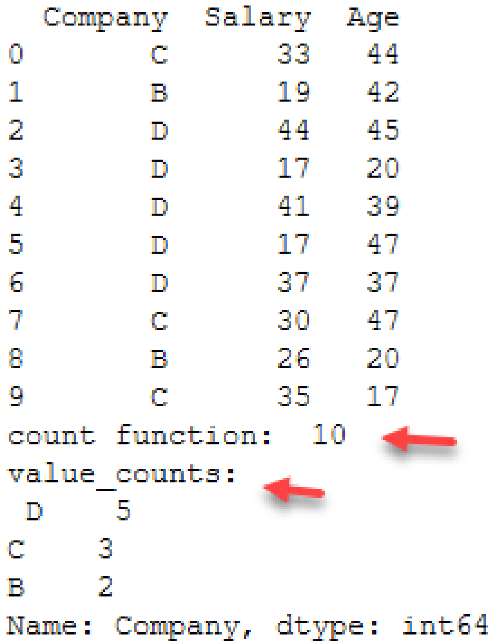


```
Count:  200
Min:  14
Max:  42
Mean:  26.585
Median:  26.0
Standard Deviation:  6.484056596741636
```

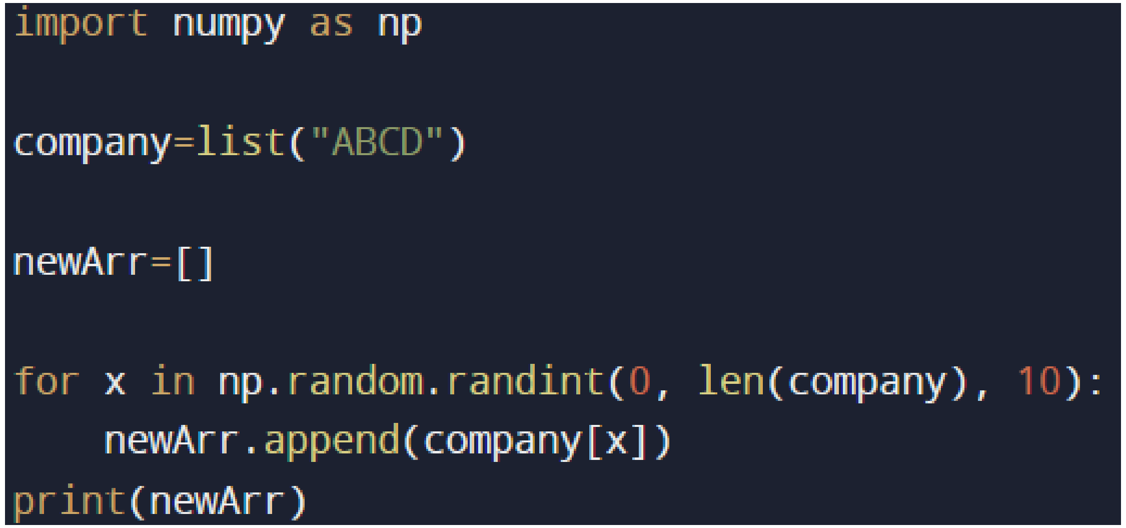
count() vs. value_counts()



```
import numpy as np
import pandas as pd
company=['A','B','C','D']
data=pd.DataFrame({'Company':[company[x] for x in np.random.randint(0,len(company),10)],\
                   'Salary':np.random.randint(5,50,10),\
                   'Age':np.random.randint(15,50,10)})
print(data)
print('count function: ',data['Company'].count())
print('value_counts: \n',data['Company'].value_counts())
```

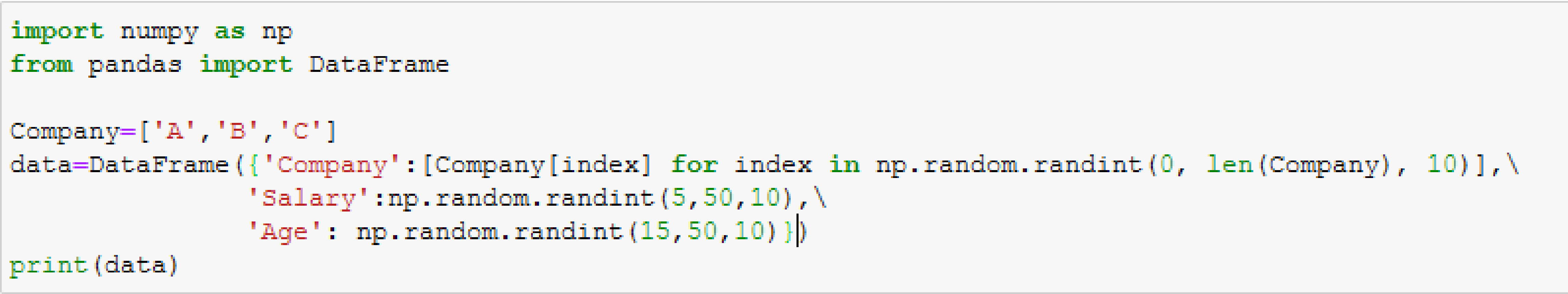


```
  Company  Salary  Age
0       C      33   44
1       B      19   42
2       D      44   45
3       D      17   20
4       D      41   39
5       D      17   47
6       D      37   37
7       C      30   47
8       B      26   20
9       C      35   17
count function:  10
value_counts: 
 D    5
C    3
B    2
Name: Company, dtype: int64
```

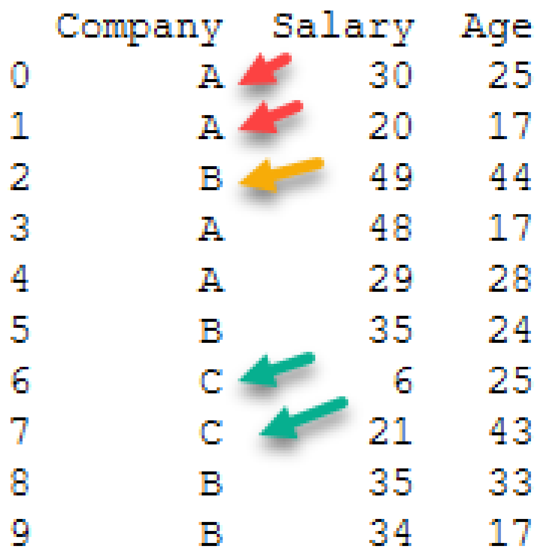


```
import numpy as np

company=list("ABCD")

newArr=[]

for x in np.random.randint(0, len(company), 10):
    newArr.append(company[x])
print(newArr)
```

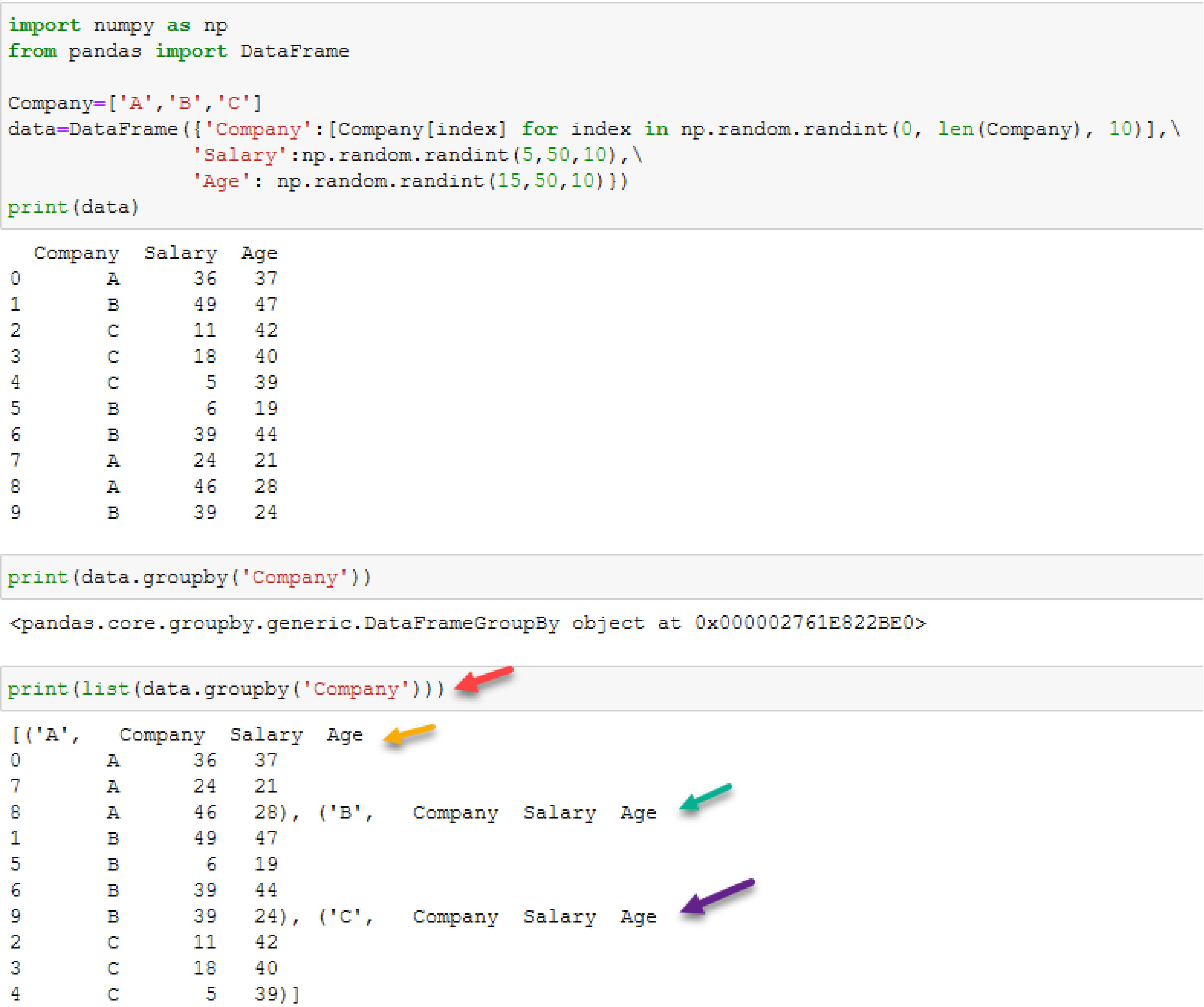
Grouping on Columns(groupby())



```
import numpy as np
from pandas import DataFrame

Company=['A','B','C']
data=DataFrame({'Company':[Company[index] for index in np.random.randint(0, len(Company), 10)],\
                'Salary':np.random.randint(5,50,10),\
                'Age': np.random.randint(15,50,10)})
print(data)
```



```
  Company  Salary  Age
0       A      30   25
1       A      20   17
2       B      49   44
3       A      48   17
4       A      29   28
5       B      35   24
6       C       6   25
7       C      21   43
8       B      35   33
9       B      34   17
```



```
import numpy as np
from pandas import DataFrame

Company=['A','B','C']
data=DataFrame({'Company':[Company[index] for index in np.random.randint(0, len(Company), 10)],\
               'Salary':np.random.randint(5,50,10),\
               'Age': np.random.randint(15,50,10)})
print(data)
```

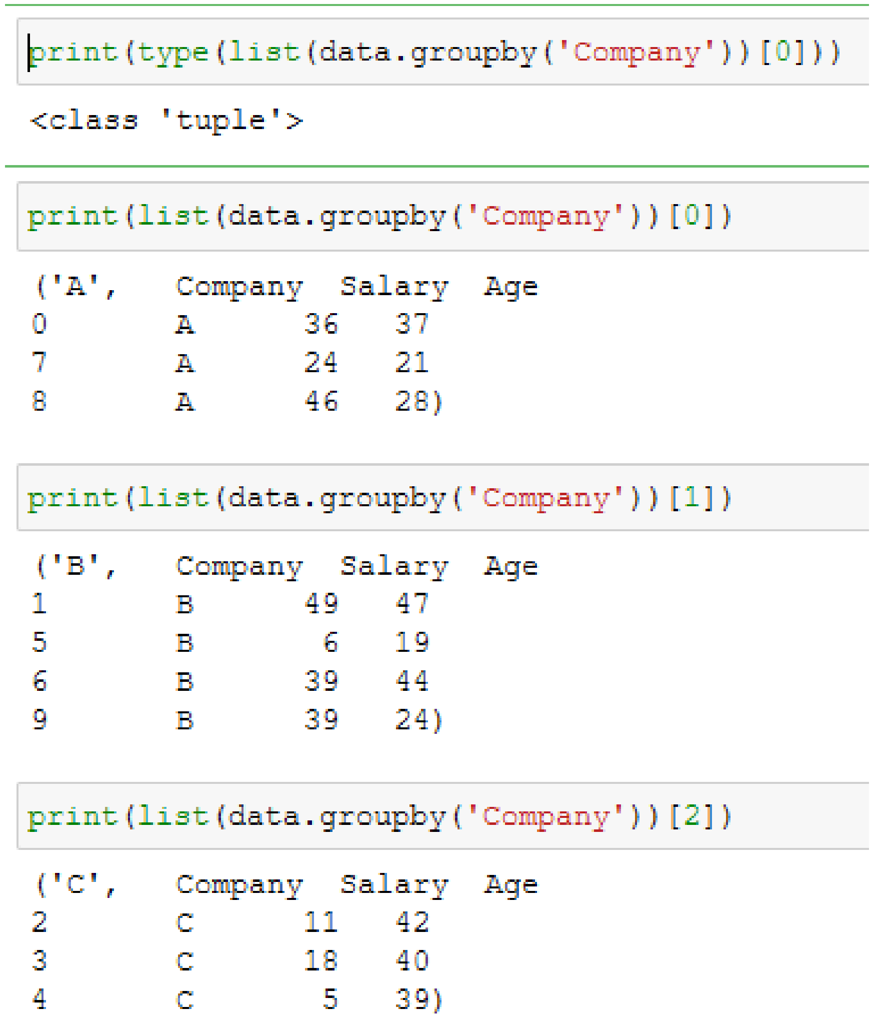
```
  Company  Salary  Age
0       A      36   37
1       B      49   47
2       C      11   42
3       C      18   40
4       C       5   39
5       B       6   19
6       B      39   44
7       A      24   21
8       A      46   28
9       B      39   24
```

```
print(data.groupby('Company'))
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002761E822BE0>
```

```
print(list(data.groupby('Company')))
```

```
[('A',   Company  Salary  Age
0       A      36   37
7       A      24   21
8       A      46   28), ('B',   Company  Salary  Age
1       B      49   47
5       B       6   19
6       B      39   44
9       B      39   24), ('C',   Company  Salary  Age
2       C      11   42
3       C      18   40
4       C       5   39)]
```



```
print(type(list(data.groupby('Company'))[0]))
```

```
<class 'tuple'>
```

```
print(list(data.groupby('Company'))[0])
```

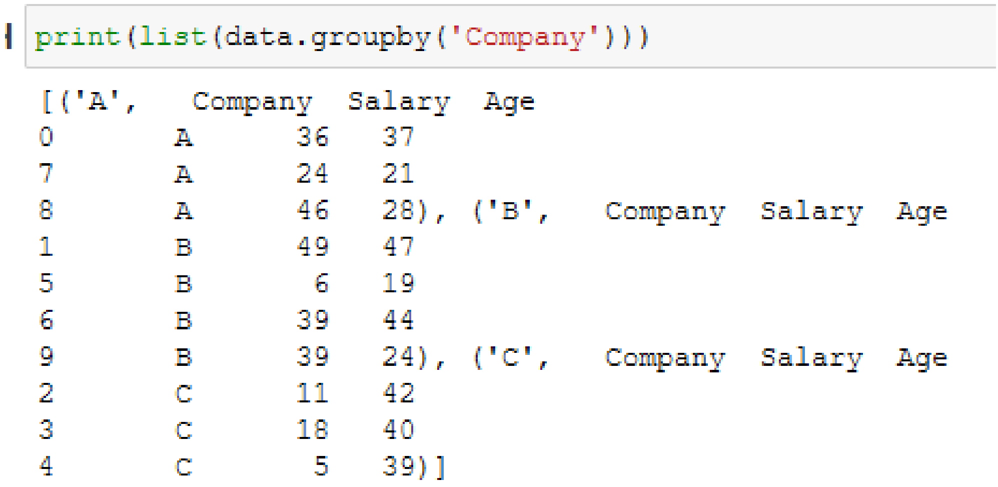
```
('A',    Company  Salary  Age
0        A      36   37
7        A      24   21
8        A      46   28)
```

```
print(list(data.groupby('Company'))[1])
```

```
('B',    Company  Salary  Age
1        B      49   47
5        B       6   19
6        B      39   44
9        B      39   24)
```

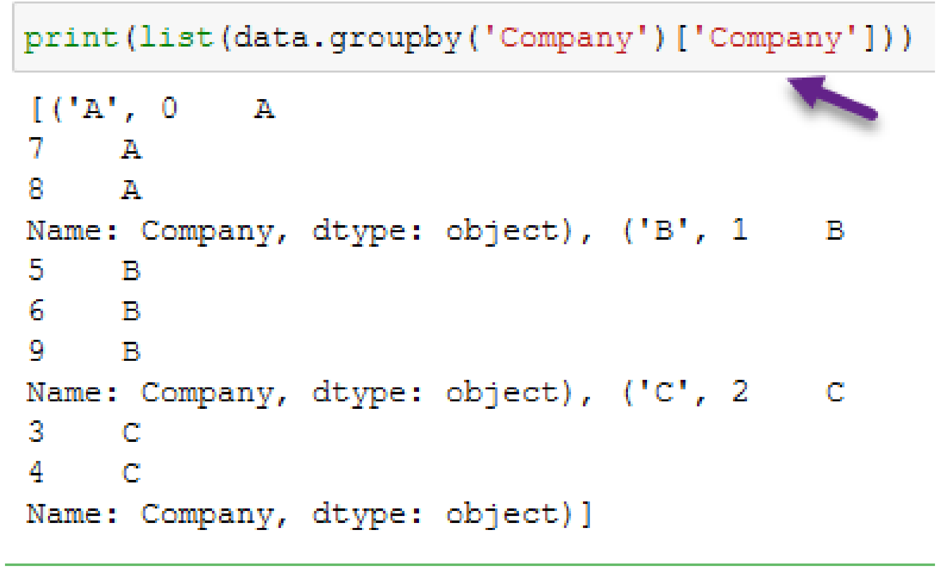
```
print(list(data.groupby('Company'))[2])
```

```
('C',    Company  Salary  Age
2        C      11   42
3        C      18   40
4        C       5   39)
```



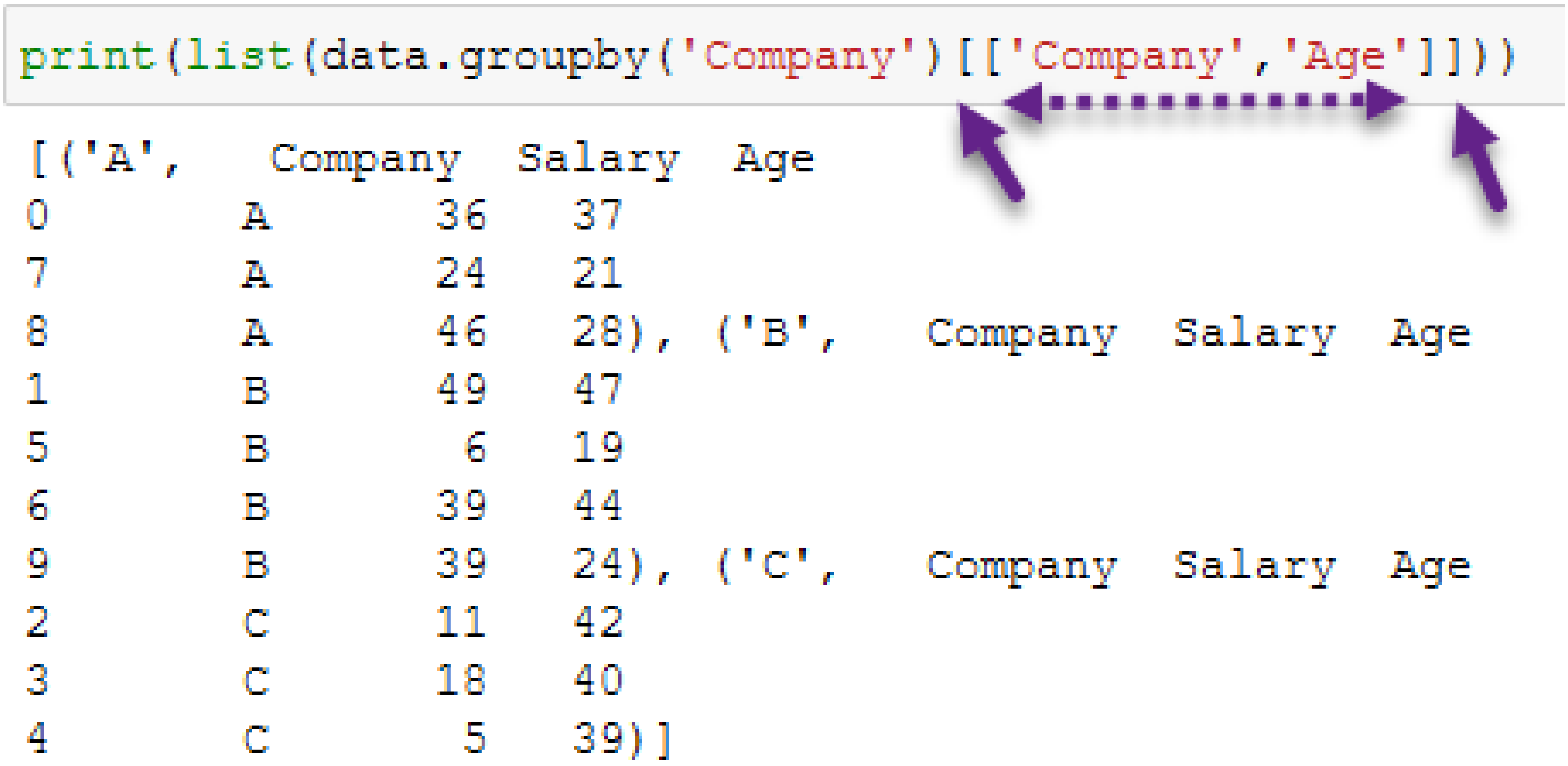
```
print(list(data.groupby('Company')))
```

```
[('A',    Company  Salary  Age
0        A      36   37
7        A      24   21
8        A      46   28), ('B',    Company  Salary  Age
1        B      49   47
5        B       6   19
6        B      39   44
9        B      39   24), ('C',    Company  Salary  Age
2        C      11   42
3        C      18   40
4        C       5   39)]
```



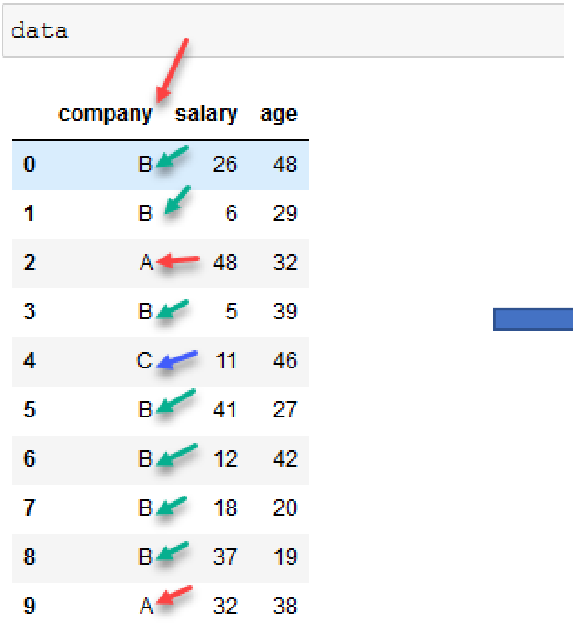
```
print(list(data.groupby('Company')['Company']))
```

```
[('A', 0    A
7    A
8    A
Name: Company, dtype: object), ('B', 1    B
5    B
6    B
9    B
Name: Company, dtype: object), ('C', 2    C
3    C
4    C
Name: Company, dtype: object)]
```



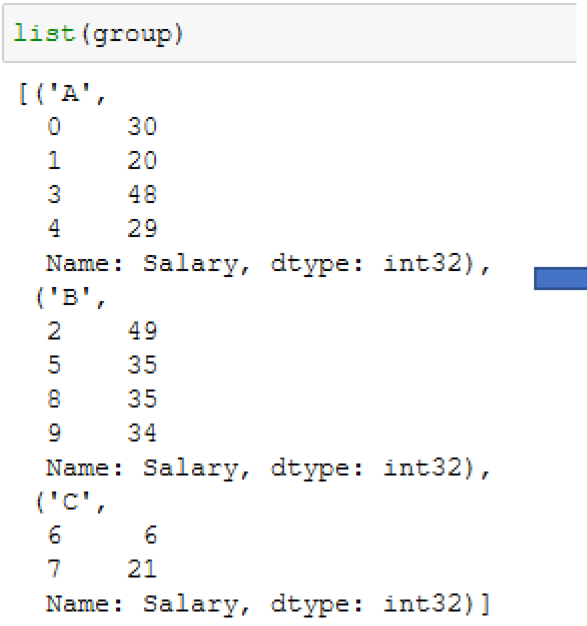
```
print(list(data.groupby('Company')[['Company','Age']]))
```

```
[('A',   Company  Salary  Age
0       A      36   37
7       A      24   21
8       A      46   28), ('B',   Company  Salary  Age
1       B      49   47
5       B       6   19
6       B      39   44
9       B      39   24), ('C',   Company  Salary  Age
2       C      11   42
3       C      18   40
4       C       5   39)]
```



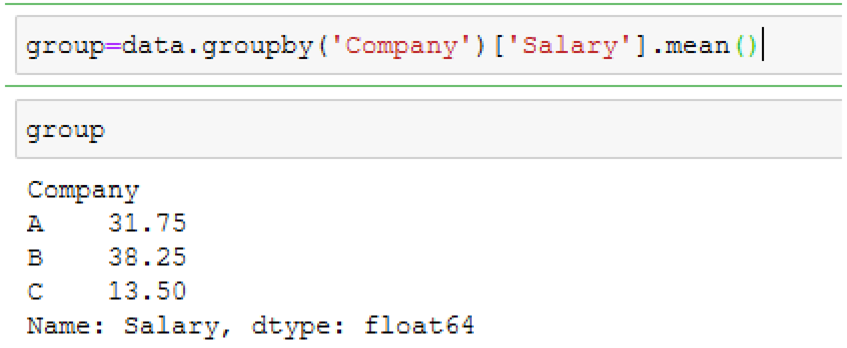
```
data
```

	company	salary	age
0	B	26	48
1	B	6	29
2	A	48	32
3	B	5	39
4	C	11	46
5	B	41	27
6	B	12	42
7	B	18	20
8	B	37	19
9	A	32	38



```
list(group)
```

```
[('A',
  0     30
  1     20
  3     48
  4     29
  Name: Salary, dtype: int32),
 ('B',
  2     49
  5     35
  8     35
  9     34
  Name: Salary, dtype: int32),
 ('C',
  6      6
  7     21
  Name: Salary, dtype: int32)]
```



```
group=data.groupby('Company')['Salary'].mean()
```

```
group
```

```
Company
A    31.75
B    38.25
C    13.50
Name: Salary, dtype: float64
```

Functions:

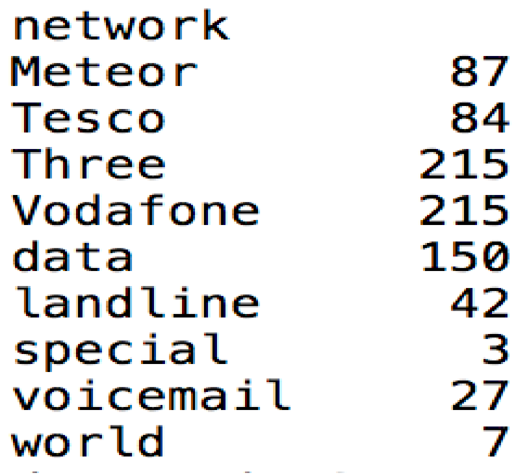
- count: Number of non-NA values in the group
- sum: Sum of non-NA values
- mean: Mean of non-NA values
- Median: Arithmetic median of non-NA values

Functions:

- Std, var: Unbiased standard deviation and variance
- min, max: Minimum and maximum of non-NA values
- prod: Product of non-NA values
- first, last: First and last non-NA values

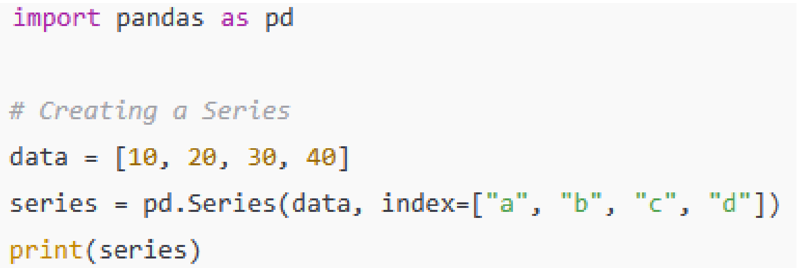
Output as Series

```
import pandas as pd

# The data file path and file name need to be configured.
PATH = "/Python/DataSets/"
CSV_DATA = "phone_data.csv"

# Note this has a comma separator.
df = pd.read_csv(PATH + CSV_DATA, skiprows=1, encoding="ISO-8859-1", sep=',',
                 names=('index', 'date', 'duration', 'item', 'month', 'network',
                        'network_type'))
dfStats=df.groupby('network')['index'].count()
print(dfStats)
```

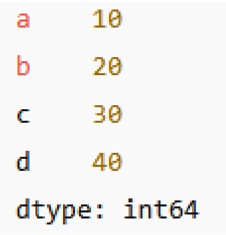


```
network
Meteor          87
Tesco           84
Three          215
Vodafone       215
data           150
landline        42
special          3
voicemail       27
world            7
```

- A Series is essentially a one-dimensional labeled array.
- It can hold data of any type (integers, floats, strings, etc.).
- It is similar to a list or a numpy array, but it comes with labels (called index) for each value.

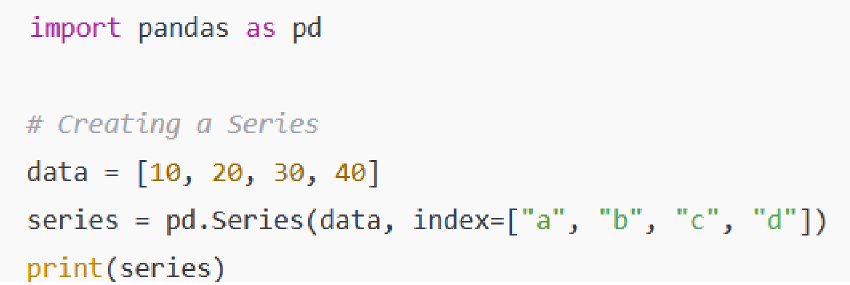


```
import pandas as pd

# Creating a Series
data = [10, 20, 30, 40]
series = pd.Series(data, index=["a", "b", "c", "d"])
print(series)
```

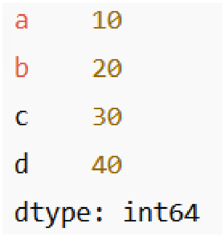


```
a    10
b    20
c    30
d    40
dtype: int64
```

Output as a DataFrame

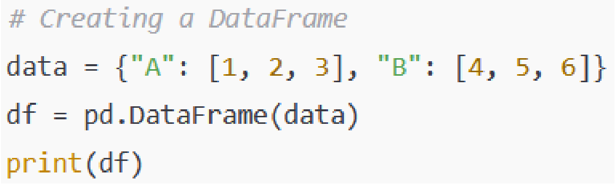


```
import pandas as pd

# Creating a Series
data = [10, 20, 30, 40]
series = pd.Series(data, index=["a", "b", "c", "d"])
print(series)
```

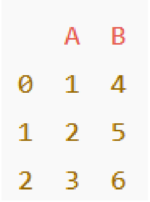


```
a     10
b     20
c     30
d     40
dtype: int64
```

dfStats=df.groupby('network')['index'].count()

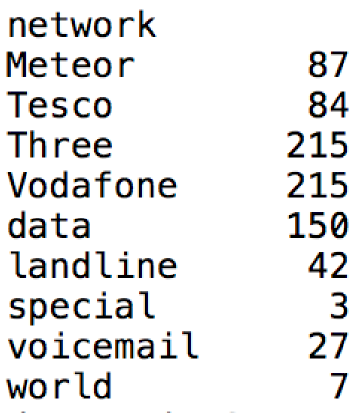


```
# Creating a DataFrame
data = {"A": [1, 2, 3], "B": [4, 5, 6]}
df = pd.DataFrame(data)
print(df)
```

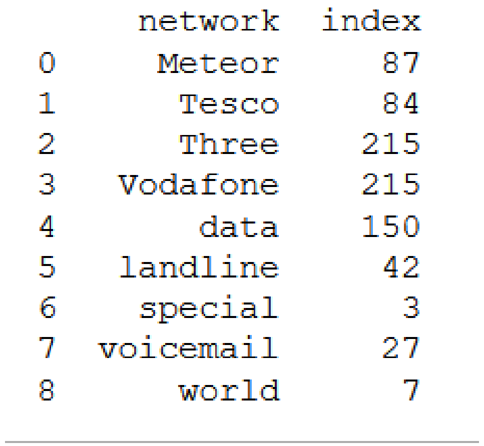


```
   A  B
0  1  4
1  2  5
2  3  6
```

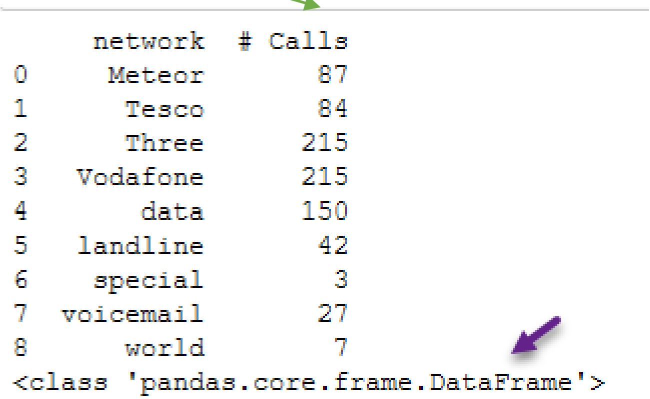
dfStats = df.groupby('network')['index'].count().reset_index().rename(columns={'index': '# Calls'})



```
network
Meteor        87
Tesco         84
Three        215
Vodafone     215
data         150
landline      42
special        3
voicemail     27
world          7
```

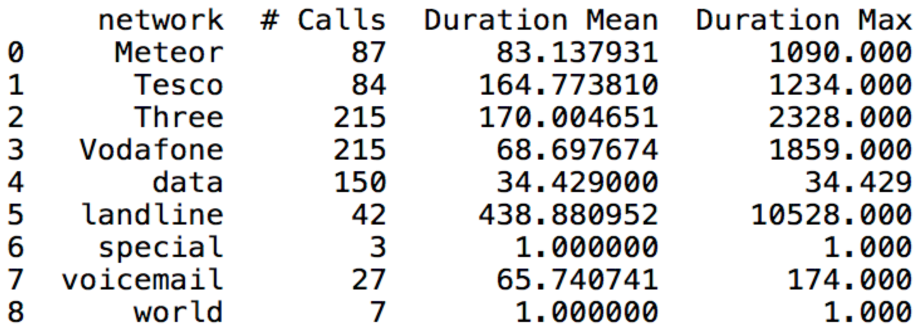


```
     network  index
0     Meteor     87
1      Tesco     84
2      Three    215
3   Vodafone    215
4       data    150
5   landline     42
6    special      3
7  voicemail     27
8      world      7
```



```
     network  # Calls
0     Meteor       87
1      Tesco       84
2      Three      215
3   Vodafone      215
4       data      150
5   landline       42
6    special        3
7  voicemail       27
8      world        7
<class 'pandas.core.frame.DataFrame'>
```

```
import pandas as pd

# The data file path and file name need to be configured.
PATH     = "/Users/pm/Desktop/DayDocs/2019_2020/PythonForDataAnalytics/workingData/"
CSV_DATA = "phone_data.csv"

# Note this has a comma separator.
df = pd.read_csv(PATH + CSV_DATA, skiprows=1,  encoding = "ISO-8859-1", sep=',',
                 names=('index', 'date', 'duration', 'item', 'month','network',
                        'network_type' ))
# Get count of items per month.
dfStats = df.groupby('network')['index']\
  .count().reset_index().rename(columns={'index': '# Calls'})

# Get duration mean for network groups and convert to DataFrame.
dfDurationMean = df.groupby('network')['duration']\
  .mean().reset_index().rename(columns={'duration': 'Duration Mean'})

# Get duration max for network groups and convert to DataFrame.
dfDurationMax = df.groupby('network')['duration']\
  .max().reset_index().rename(columns={'duration': 'Duration Max'})

# Append duration mean to stats matrix.
dfStats['Duration Mean'] = dfDurationMean['Duration Mean']

# Append duration max to stats matrix.
dfStats['Duration Max'] = dfDurationMax['Duration Max']
```

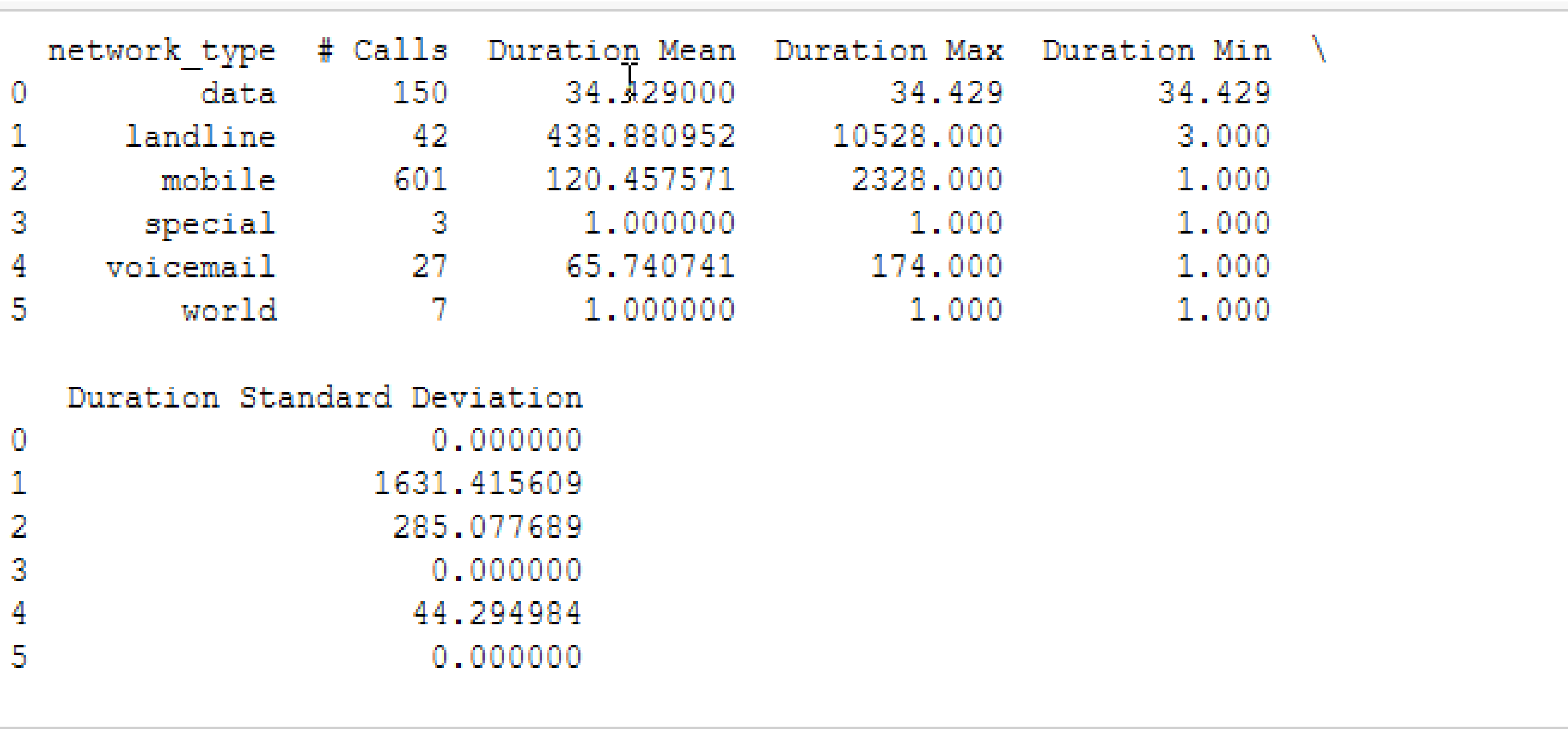


```
     network  # Calls  Duration Mean  Duration Max
0     Meteor       87      83.137931      1090.000
1      Tesco       84     164.773810      1234.000
2      Three      215     170.004651      2328.000
3   Vodafone      215      68.697674      1859.000
4       data      150      34.429000        34.429
5   landline       42     438.880952     10528.000
6    special        3       1.000000         1.000
7  voicemail       27      65.740741       174.000
8      world        7       1.000000         1.000
```

Exercise 5

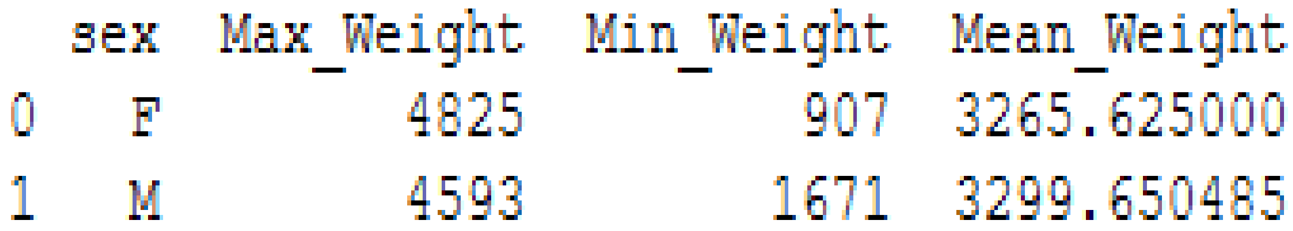
Exercise 6

Write a program to group phone calls by network type. Show network_type, total calls, duration mean, duration max, duration minimum and standard deviation in your output.



```
  network_type  # Calls  Duration Mean  Duration Max  Duration Min  \
0         data      150      34.429000        34.429        34.429   
1     landline       42     438.880952     10528.000         3.000   
2       mobile      601     120.457571      2328.000         1.000   
3      special        3       1.000000         1.000         1.000   
4    voicemail       27      65.740741       174.000         1.000   
5        world        7       1.000000         1.000         1.000   

   Duration Standard Deviation  
0                     0.000000  
1                  1631.415609  
2                   285.077689  
3                     0.000000  
4                    44.294984  
5                     0.000000  
```

Exercise 7 Using the baby sample, group on either male or female and show the maximum weight, minimum weight, and mean weight. Show your program here.



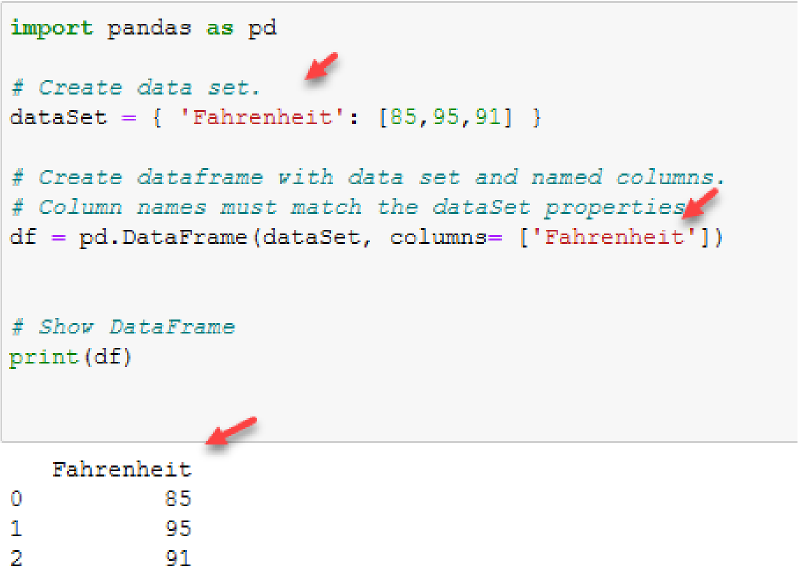
```
  sex  Max_Weight  Min_Weight  Mean_Weight
0   F        4825         907  3265.625000
1   M        4593        1671  3299.650485
```

Generating a Calculated Column

If the same calculation can be applied to each row of a column you can do it all at once.

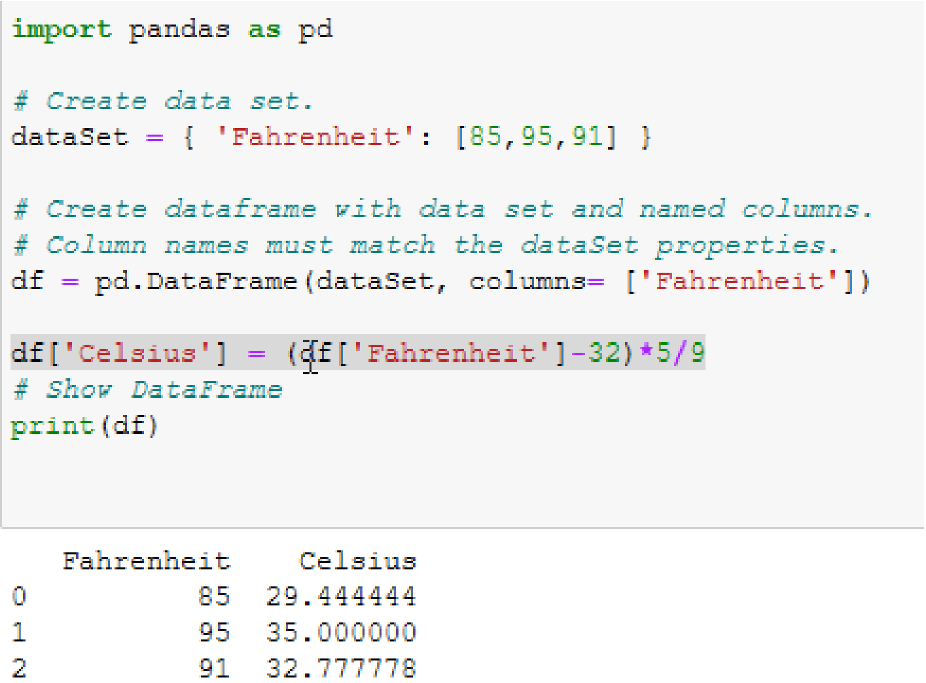
```
import pandas as pd

# Create data set.
dataSet = { 'Fahrenheit': [85,95,91] }

# Create dataframe with data set and named columns.
# Column names must match the dataSet properties.
df = pd.DataFrame(dataSet, columns= ['Fahrenheit'])

df['Celsius'] = (df['Fahrenheit']-32)*5/9
# Show DataFrame
print(df)
```

Exercise 8



```
import pandas as pd

# Create data set.
dataSet = { 'Fahrenheit': [85,95,91] }

# Create dataframe with data set and named columns.
# Column names must match the dataSet properties.
df = pd.DataFrame(dataSet, columns= ['Fahrenheit'])


# Show DataFrame
print(df)
```

```
   Fahrenheit
0          85
1          95
2          91
```



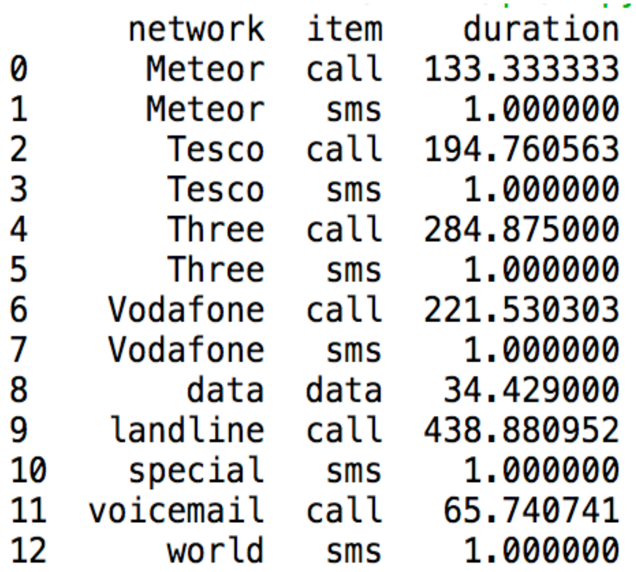
```
import pandas as pd

# Create data set.
dataSet = { 'Fahrenheit': [85,95,91] }

# Create dataframe with data set and named columns.
# Column names must match the dataSet properties.
df = pd.DataFrame(dataSet, columns= ['Fahrenheit'])

df['Celsius'] = (df['Fahrenheit']-32)*5/9
# Show DataFrame
print(df)
```

```
   Fahrenheit    Celsius
0          85  29.444444
1          95  35.000000
2          91  32.777778
```

Grouping on Multiple Columns

```
import pandas as pd

# The data file path and file name need to be configured.
PATH     = "/Users/pm/Desktop/DayDocs/2019_2020/PythonForDataAnalytics/workingData/"
CSV_DATA = "phone_data.csv"

# Note this has a comma separator.
df = pd.read_csv(PATH + CSV_DATA, skiprows=1,  encoding = "ISO-8859-1", sep=',',
              names=('index', 'date', 'duration', 'item', 'month','network',
                   'network_type' ))

df2 = df.groupby(['network','item'])['duration'].mean().reset_index()
print(df2)
```

List



```
       network  item    duration
0       Meteor  call  133.333333
1       Meteor   sms    1.000000
2        Tesco  call  194.760563
3        Tesco   sms    1.000000
4        Three  call  284.875000
5        Three   sms    1.000000
6     Vodafone  call  221.530303
7     Vodafone   sms    1.000000
8         data  data   34.429000
9     landline  call  438.880952
10     special   sms    1.000000
11   voicemail  call   65.740741
12       world   sms    1.000000
```

Exercise 9

Updating on DataFrame Cell at a Time

```
Data frame with original data in Celsius:
      City  Temperature
0    Mumbai         23.0
1   Beijing        -11.0
['City', 'Temperature']

Dataframe after changed to Fahrenheit:
      City  Temperature
0    Mumbai         73.4
1   Beijing         12.2
```

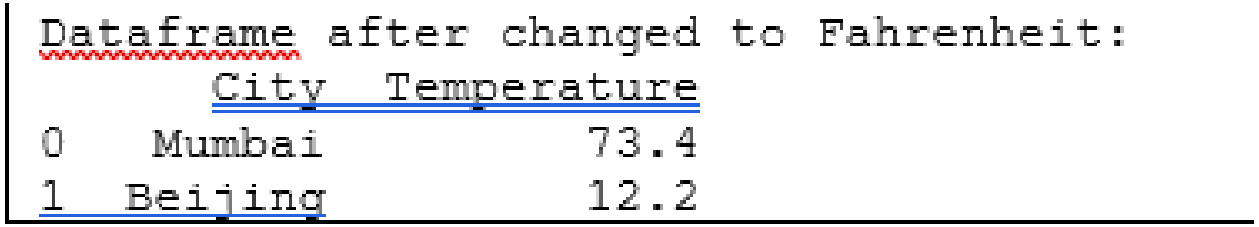
The line of code needed to update the cell uses the iat[] reference:

```
df.iat[i, tempColumnPosition] = celsius*9.0/5.0 + 32
```

```
    #Convert Columns to A list
    columnList = list(df.keys())

for i in range(0, len(columnList)):
    if(columnList[i]==columnName):
      columnPosition = i
      break # Exit the loop.
  return columnPosition
```

.iat vs .loc

Exercise 10



```
Dataframe after changed to Fahrenheit:
      City  Temperature
0   Mumbai         73.4
1  Beijing         12.2
```

Dynamically adjust Example 4 to change the city values, one cell at a time, so the updated data in the data frame becomes:

```
     City  Temperature
0   The city of Mumbai        73.4
1  The city of Beijing        12.2
```