PANDAS

OVERVIEW

Pandas

- Panel Data Module
- objects: series and dataframes
- series similar to a table column
- dataframe similar to a table
- designed to manage indexed data (like SQL)

Installing Pandas

- install via *pip*

```
>> pip install pandas
```

- in Spyder install via !pip

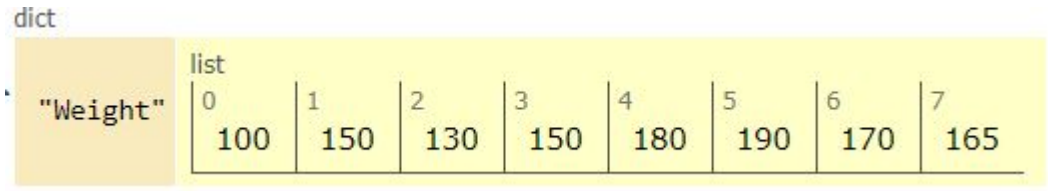
```
>> !pip install pandas
```

- import both Pandas and Numpy

```
import pandas as pd
import numpy as np
```

Pandas Series Object

- similar to an Excel column
- "similar" to dictionary:
  1. key: column name
  2. value: list of values (all of the same type)



- but values could have own index

Contructing Series

```
import pandas as pd
import numpy as np

index         = [ "x1","x2","x3","x4",
                  "x5","x6","x7","x8"]
Weight        = [100, 150, 130, 150,
                 180, 190, 170, 165]
weight_series = pd.Series(Weight, index)

>> weight_series
x1    100
x2    150
x3    130
x4    150
x5    180
x6    190
x7    170
x8    165
dtype: int64
```

Default Index

- can use default indexing

```
import pandas as pd
import numpy as np

Weight      = [100, 150, 130, 150,
               180, 190, 170, 165]
weight_series = pd.Series(Weight)
```

```
>> weight_series
0    100
1    150
2    130
3    150
4    180
5    190
6    170
7    165
dtype: int64
dtype: int64
```

Creating a Series from Dictionary

```
import pandas as pd
import numpy as np

new_dict = {"x1":100,"x2":150,"x3":130,"x4":150,
            "x5":180,"x6":190,"x7":170,"x8":165}

weight_series = pd.Series(new_dict)

>> weight_series
x1    100
x2    150
x3    130
x4    150
x5    180
x6    190
x7    170
x8    165
dtype: int64
```

Accessing Data in Series

```
import pandas as pd
import numpy as np

new_dict = {"x1":100,"x2":150,"x3":130,
            "x4":150,"x5":180,"x6":190,
            "x7":170,"x8":165}

weight_series = pd.Series(new_dict)
```

- use index

```
>> print(weight_series["x3"]
130
>> print(weight_series["x3"]
180
```

Operations on Series

```
import pandas as pd
import numpy as np

index  = [ "x1","x2","x3","x4",
           "x5","x6","x7","x8"]
Weight = [100, 150, 130, 150,
          180, 190, 170, 165]
Foot   = [6, 8, 7, 9, 13, 11, 12, 10]

weight_series = pd.Series(Weight, index)
foot_series = pd.Series(Foot, index)

weight_per_foot=weight_series/foot_series
```

- can do basic arithmetic (over index)

Operations on Series (cont'd)

```
>> weight_per_foot
x1    16.666667
x2    18.750000
x3    18.571429
x4    16.666667
x5    13.846154
x6    17.272727
x7    14.166667
x8    16.500000
dtype: float64
```

Broadcasting

- can perform element-wise broadcasting
- similar to numpy

```
>> weight_series * 2
x1    200
x2    300
x3    260
x4    300
x5    360
x6    380
x7    340
x8    330
dtype: int64
```

Pandas Dataframe

- *series* contains a list with index
- *dataframe* is a collection of series with (same) index
- can create in many ways:

1. series object
2. reading csv/Excel file
3. numpy array
4. dictionary

Dataframe from Dictionary

```
import pandas as pd
import numpy as np

df = pd.DataFrame(
        {"Weight": [100, 150, 130, 150,
                    180, 190, 170, 165],
        "Foot" : [6,8,7,9,13,11,12,10]})
```

```
>> df
   Foot  Weight
0     6     100
1     8     150
2     7     130
3     9     150
4    13     180
5    11     190
6    12     170
7    10     165
```

Custom Indexing

```
import pandas as pd
import numpy as np

df = pd.DataFrame(
     {"Weight": [100, 150, 130, 150,
                 180, 190, 170, 165],
      "Foot" : [6,8,7,9,13,11,12,10]},
     index=["x1","x2","x3","x4",
            "x5","x6","x7","x8"])
```

```
>> df
    Foot  Weight
x1     6     100
x2     8     150
x3     7     130
x4     9     150
x5    13     180
x6    11     190
x7    12     170
x8    10     165
```

Getting Column Values

```
>> df["Foot"]
x1     6
x2     8
x3     7
x4     9
x5    13
x6    11
x7    12
x8    10
Name: Foot, dtype: int64
```

- alternative method

```
>> df.Foot
```

Multiple Column Values

- pass a list of column names

```
>> df[ ["Foot", "Weight"]]
    Foot  Weight
x1     6     100
x2     8     150
x3     7     130
x4     9     150
x5    13     180
x6    11     190
x7    12     170
x8    10     165
```

Creating a New Column

- vectorized computation

```
>> df["weight_per_foot"] = df["Weight"]/
                           df["Foot"]
>> df
    Foot  Weight  weight_per_foot
x1     6     100         16.666667
x2     8     150         18.750000
x3     7     130         18.571429
x4     9     150         16.666667
x5    13     180         13.846154
x6    11     190         17.272727
x7    12     170         14.166667
x8    10     165         16.500000
```

Renaming Column(s)

- rename column(s) with new name(s)

```
>> df.rename(columns={"weight_per_foot":
          "density"}, inplace = True)
>> df
   Foot  Weight          density
x1     6     100         16.666667
x2     8     150         18.750000
x3     7     130         18.571429
x4     9     150         16.666667
x5    13     180         13.846154
x6    11     190         17.272727
x7    12     170         14.166667
x8    10     165         16.500000
```

Dropping Column(s)

- drop column(s) "in-place"

```
>> df.drop("density", axis=1,
           inplace=True)
>> df
    Foot  Weight
x1     6     100
x2     8     150
x3     7     130
x4     9     150
x5    13     180
x6    11     190
x7    12     170
x8    10     165
```

Simple Sorting

- sorting by one column
- can be done "in-place"

```
>> df_2 = df.sort_values(by=["Weight"],
          ascending = [False])
>> df_2
     Foot  Weight
x6     11     190
x5     13     180
x7     12     170
x8     10     165
x2      8     150
x4      9     150
x3      7     130
x1      6     100
```

Multi-Column Sorting

- sorting by multiple columns
- can be done "in-place"

```
>> df_3 = df.sort_values(
          by=["Weight","Foot"],
          ascending=[False, False])
>> df_3
     Foot  Weight
x6     11     190
x5     13     180
x7     12     170
x8     10     165
x4      9     150
x2      8     150
x3      7     130
x1      6     100
```

head() and *tail*()

- *head*(n) - first n rows
- *tail*(n) - last n rows

```
>> weight_frame.head(2)
    Foot  Weight
x1     6     100
x2     8     150
>>
>> weight_frame.tail(2)
    Foot  Weight
x7    12     170
x8    10     165
```

A Numerical Dataset

object x_i	Height (H)	Weight (W)	Foot (F)	Label (L)
x_1	5.00	100	6	green
x_2	5.50	150	8	green
x_3	5.33	130	7	green
x_4	5.75	150	9	green
x_5	6.00	180	13	red
x_6	5.92	190	11	red
x_7	5.58	170	12	red
x_8	5.92	165	10	red

- $N = 8$ items
- $M = 3$ (unscaled) attributes

Code for the Dataset

```
import pandas as pd
data = pd.DataFrame(
 {"id":[ 1,2,3,4,5,6,7,8],
  "Label":["green","green","green","green",
                    "red","red","red","red"],
  "Height":[5,5.5,5.33,5.75,6.00,5.92,5.58,5.92],
  "Weight":[100,150,130,150,180,190,170,165],
  "Foot":[6, 8, 7, 9, 13, 11, 12, 10]},
  columns=["id","Height","Weight",
                          "Foot","Label"])
```

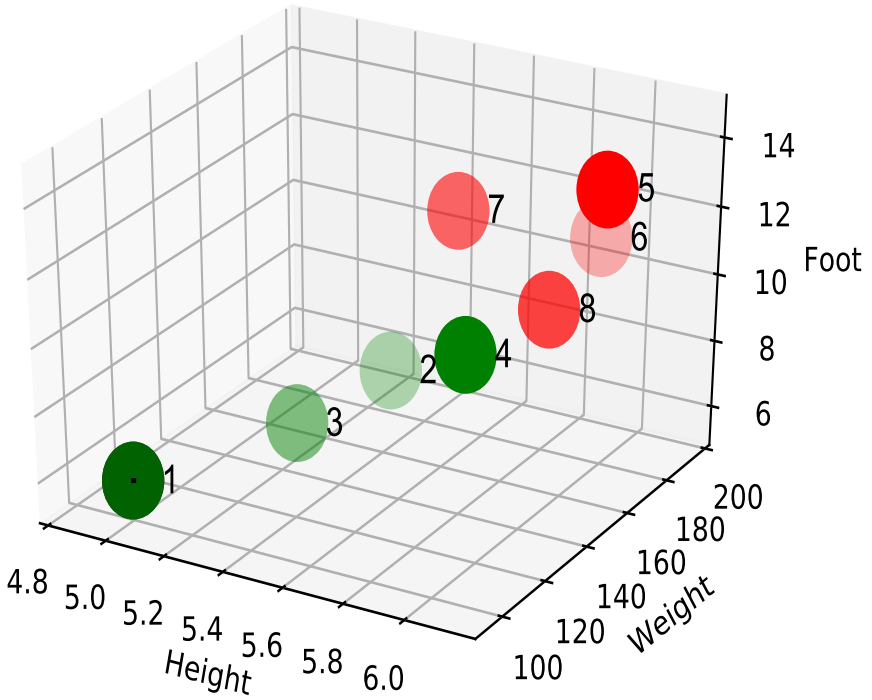
```
ipdb> data
  id Height Weight Foot Label
0  1   5.00    100    6  green
1  2   5.50    150    8  green
2  3   5.33    130    7  green
3  4   5.75    150    9  green
4  5   6.00    180   13    red
5  6   5.92    190   11    red
6  7   5.58    170   12    red
7  8   5.92    165   10    red
```

Alternative Approach

```
data = pd.DataFrame(
  data = [ [1, 5,    100, 6,  "green"],
           [2, 5.5,  150, 8,  "green"],
           [3, 5.33, 130, 7,  "green"],
           [4, 5.75, 150, 9,  "green"],
           [5, 6,    180, 13, "red"],
           [6, 5.92, 190, 11, "red"],
           [7, 5.58, 170, 12, "red"],
           [8, 5.92, 165, 10, "red"]],
  columns = ["id","Height","Weight",
                          "Foot","Label"] )
```

```
ipdb> data
  id Height Weight Foot Label
0  1   5.00    100    6  green
1  2   5.50    150    8  green
2  3   5.33    130    7  green
3  4   5.75    150    9  green
4  5   6.00    180   13    red
5  6   5.92    190   11    red
6  7   5.58    170   12    red
7  8   5.92    165   10    red
```

A Dataset Illustration



Some Observations

- data in different shapes (dictionary, lists)
- different data types
- columns have custom names
- index can be in different formats

Typical Operations

- index values

```
> data.index
RangeIndex(start=0, stop=8, step=1)
```

- column names

```
> data.columns
Index(['id','Height','Weight','Foot',
       'Label'], dtype='object')
```

Data Selection

- selection via index:

1. .loc by label
2. .iloc by position

```
> data.iloc[5]
id            6
Height     5.92
Weight      190
Foot         11
Label       red
Name: 5, dtype: object
```

Data Selection

- selection of multiple indices

```
> data.iloc[[5,7]]
   id  Height  Weight  Foot Label
5   6    5.92     190    11   red
7   8    5.92     165    10   red
```

- se;ection via index object

```
> data.iloc[data.index[1:7:2]]
   id  Height  Weight  Foot  Label
1   2    5.50     150     8  green
3   4    5.75     150     9  green
5   6    5.92     190    11    red
```

Statistical Functions

- apply statistical functions

```
> data[['Height','Weight',
            'Foot']].mean()
Height      5.625
Weight    154.375
Foot        9.500
dtype: float64
```

Lambda Functions

- apply lambda functions

```
> data[['Height',
      'Weight']].apply(lambda x: x**2)
    Height  Weight
0  25.0000   10000
1  30.2500   22500
2  28.4089   16900
3  33.0625   22500
4  36.0000   32400
5  35.0464   36100
6  31.1364   28900
7  35.0464   27225
```

Adding Column(s)

```
> data['n_col']=['a','b','c','d',
                 'e','f','g','h']
```

```
> data
```

	id	Height	Weight	Foot	Label	n_col
0	1	5.00	100	6	green	a
1	2	5.50	150	8	green	b
2	3	5.33	130	7	green	c
3	4	5.75	150	9	green	d
4	5	6.00	180	13	red	e
5	6	5.92	190	11	red	f
6	7	5.58	170	12	red	g
7	8	5.92	165	10	red	h

Dropping Column(s)

```
> data.drop(['n_col'],axis=1,inplace=True
> data
   id  Height  Weight  Foot  Label
0   1    5.00     100     6  green
1   2    5.50     150     8  green
2   3    5.33     130     7  green
3   4    5.75     150     9  green
4   5    6.00     180    13    red
5   6    5.92     190    11    red
6   7    5.58     170    12    red
7   8    5.92     165    10    red
```

- axis: 1-columns, 0 - rows

Dropping Duplicates

- can also drop "in-place"

```
>> data_2=data.drop_duplicates("Weight")
>> data_2
   id  Height  Weight  Foot  Label
0   1    5.00     100     6  green
1   2    5.50     150     8  green
2   3    5.33     130     7  green
4   5    6.00     180    13    red
5   6    5.92     190    11    red
6   7    5.58     170    12    red
7   8    5.92     165    10    red
```

Desribing the Dataset

```
import pandas as pd
data = pd.DataFrame(
 {"id":[ 1,2,3,4,5,6,7,8],
  "Label":["green","green","green","green",
                    "red","red","red","red"],
  "Height":[5,5.5,5.33,5.75,6.00,5.92,5.58,5.92],
  "Weight":[100,150,130,150,180,190,170,165],
  "Foot":[6, 8, 7, 9, 13, 11, 12, 10]},
  columns=["id","Height","Weight",
                          "Foot","Label"])
```

```
ipdb> data.describe()
             id    Height      Weight      Foot
count  8.00000  8.000000    8.000000   8.00000
mean   4.50000  5.625000  154.375000   9.50000
std    2.44949  0.343428   28.962722   2.44949
min    1.00000  5.000000  100.000000   6.00000
25%    2.75000  5.457500  145.000000   7.75000
50%    4.50000  5.665000  157.500000   9.50000
75%    6.25000  5.920000  172.500000  11.25000
max    8.00000  6.000000  190.000000  13.00000
```

Reversing Rows

```
> data_rev_rows = data.loc[::-1]
> data_rev_rows
```

	id	Height	Weight	Foot	Label
7	8	5.92	165	10	red
6	7	5.58	170	12	red
5	6	5.92	190	11	red
4	5	6.00	180	13	red
3	4	5.75	150	9	green
2	3	5.33	130	7	green
1	2	5.50	150	8	green
0	1	5.00	100	6	green

- similar to Python lists

Reversing Columns

```
> data_rev_cols = data.loc[:, ::-1]
> data_rev_cols
   Label  Foot  Weight  Height  id
0  green     6     100    5.00   1
1  green     8     150    5.50   2
2  green     7     130    5.33   3
3  green     9     150    5.75   4
4    red    13     180    6.00   5
5    red    11     190    5.92   6
6    red    12     170    5.58   7
7    red    10     165    5.92   8
```

- similar to Python lists

Filtering s DataFrame

```
> data_red = data[data["Label"]=="red"]
> data_red
    id  Height  Weight  Foot Label
4   5    6.00     180    13   red
5   6    5.92     190    11   red
6   7    5.58     170    12   red
7   8    5.92     165    10   red
> data_s = data[data["Foot"].isin([7,9])]
> data_s
   id  Height  Weight  Foot  Label
2   3    5.33     130     7  green
3   4    5.75     150     9  green
```

Filtering s DataFrame (cont'd)

```
> data_med = data[(data["Foot"] > 6)
                & (data["Weight"] < 160)]
> data_med
   id  Height  Weight  Foot  Label
1   2    5.50     150     8  green
2   3    5.33     130     7  green
3   4    5.75     150     9  green
```

- can use multiple criteria

Counting Values

```
> counts = data['Weight'].value_counts()
> counts
150    2
190    1
170    1
165    1
180    1
130    1
100    1
Name: Weight, dtype: int64
> counts.nlargest(1)
150    2
Name: Weight, dtype: int64
```

Aggregating

```
> data_m = data.groupby("Label")
                ["Weight"].mean()
> data_m
Label
green    132.50
red      176.25
Name: Weight, dtype: float64
> data_ms = data.groupby("Label")
     "Weight"].agg(["mean","std"])
          mean         std
Label
green  132.50  23.629078
red    176.25  11.086779
```

Concepts Check:

(a) *Series* object

(b) broadcasting

(c) Pandas *DataFrame*

(d) column creation, indexing, sort

(e) *head*() and *tail*() functions

(f) data selection (label, position)

(g) lambda functions

(h) filtering, counting, aggregation