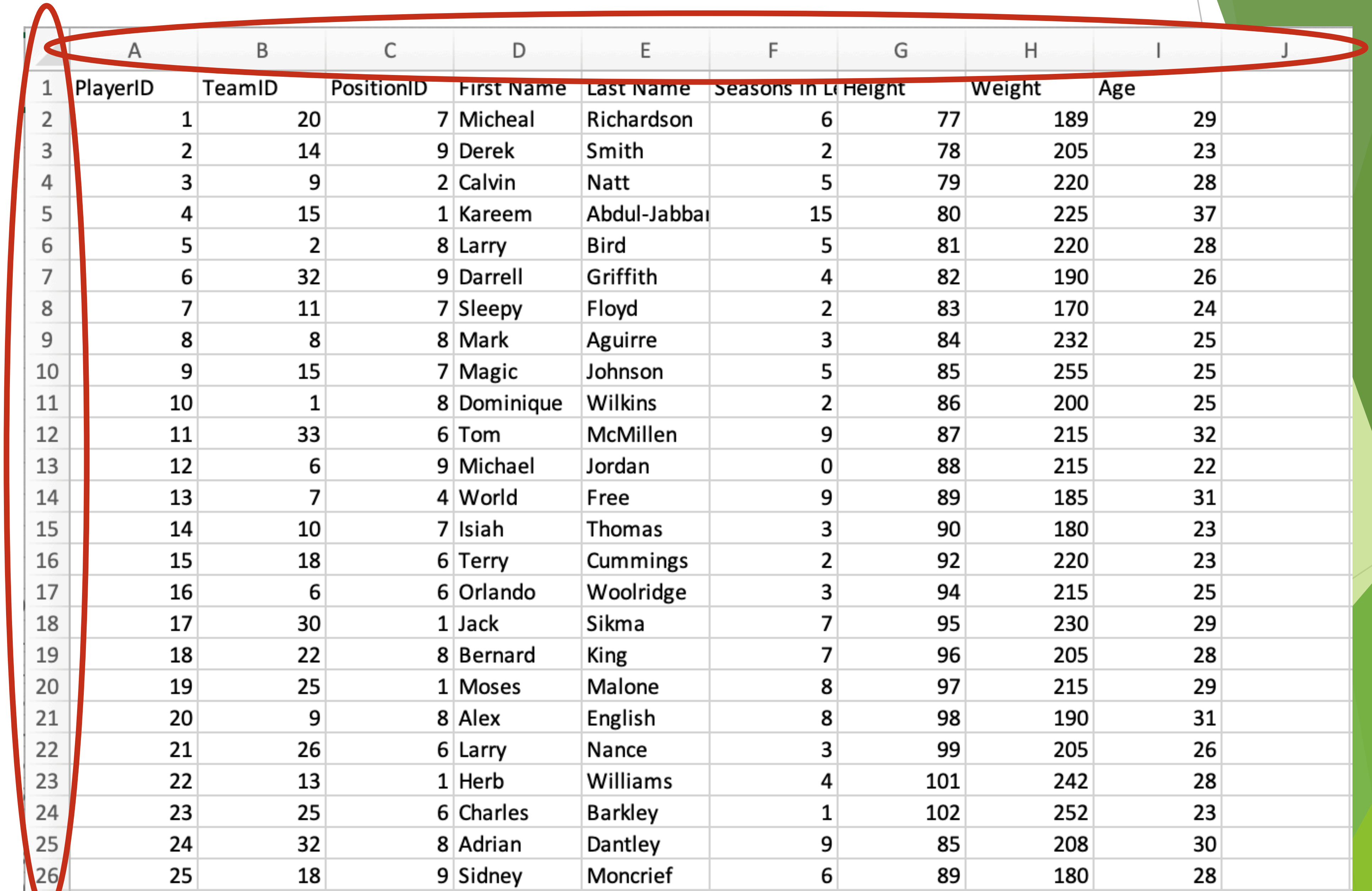
CS 217 Data Management and Information Processing

Data Table and Pandas

Comma Separated Values (CSV)

- CSV is a simple text format for storing tabular data (spreadsheets)
- Each row is represented on one line of text
- Columns are separated by commas
- Values can be enclosed in double quotes ("...") if necessary
  - For example, if value includes comma or newline characters
  - Double quotes within a text value must be “escaped” by using two double quotes
- Values can be empty by having nothing between the commas

NBA_player_of_the_week.csv viewed in Excel



	A	B	C	D	E	F	G	H	I	J
1	PlayerID	TeamID	PositionID	First Name	Last Name	Seasons in Le	Height	Weight	Age	
2	1	20	7	Micheal	Richardson	6	77	189	29	
3	2	14	9	Derek	Smith	2	78	205	23	
4	3	9	2	Calvin	Natt	5	79	220	28	
5	4	15	1	Kareem	Abdul-Jabbar	15	80	225	37	
6	5	2	8	Larry	Bird	5	81	220	28	
7	6	32	9	Darrell	Griffith	4	82	190	26	
8	7	11	7	Sleepy	Floyd	2	83	170	24	
9	8	8	8	Mark	Aguirre	3	84	232	25	
10	9	15	7	Magic	Johnson	5	85	255	25	
11	10	1	8	Dominique	Wilkins	2	86	200	25	
12	11	33	6	Tom	McMillen	9	87	215	32	
13	12	6	9	Michael	Jordan	0	88	215	22	
14	13	7	4	World	Free	9	89	185	31	
15	14	10	7	Isiah	Thomas	3	90	180	23	
16	15	18	6	Terry	Cummings	2	92	220	23	
17	16	6	6	Orlando	Woolridge	3	94	215	25	
18	17	30	1	Jack	Sikma	7	95	230	29	
19	18	22	8	Bernard	King	7	96	205	28	
20	19	25	1	Moses	Malone	8	97	215	29	
21	20	9	8	Alex	English	8	98	190	31	
22	21	26	6	Larry	Nance	3	99	205	26	
23	22	13	1	Herb	Williams	4	101	242	28	
24	23	25	6	Charles	Barkley	1	102	252	23	
25	24	32	8	Adrian	Dantley	9	85	208	30	
26	25	18	9	Sidney	Moncrief	6	89	180	28	

How to Process Tabular Data?

- Efficiency
  - The code should run quickly
- Easy to program
  - Should not take much effort to express our query
- Portable
  - The analysis can be quickly hooked up with other code blocks

Pandas

- One of the most popular library that data scientists use
- Created by Wes McKinney in 2008, now maintained by Jeff Reback and many others.
  - Author of one of the textbooks: Python for Data Analysis
- Powerful and productive Python data analysis and Management Library
- Its an open source product.
  - Free to use and free to modify

Overview

- Python Library to provide data analysis features similar to: R, MATLAB, SAS
- Rich data structures and functions to make working with data structure fast, easy and expressive.
- It is built on top of NumPy
- Key components provided by Pandas:
  - Series
  - DataFrame

Might be the most frequently used tool after taking this course!

Pandas: Essential Concepts

- A **Series** is a named Python list (one-entry dict with list as value):

```
{ 'grades': [50,90,100,45] }
```

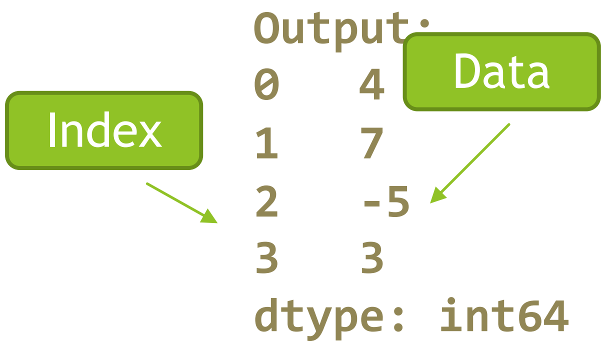
- A **DataFrame** is a collection of Series (dict-like container for series):

```
{'names': ['bob', 'ken', 'art', 'joe'],
  'grades': [50,90,100,45]
 }
```

Series

- One dimensional array-like object
- It contains array of data (of any NumPy data type) with associated indexes. (Indexes can be strings or integers or other data types.)
- By default , the series will get indexing from 0 to N where N = size -1

```
obj = Series([4, 7, -5, 3])
obj
```



```
Output:
0   4
1   7
2   -5
3   3
dtype: int64
```

Index

Data

Series: Create

```
from pandas import Series

data = [1,2,3,numpy.nan,5,6]   # nan == Not a Number
unindexed = Series(data)

indices = ['a', 'b', 'c', 'd', 'e', 'f']
indexed = Series(data, index=indices)

data_dict = {'a' : 1, 'b' : 2, 'c' : 3}
indexed = Series(data_dict)
```

Series: Accessing Elements

```
obj2 = Series([4, 7, -5, 3], \
              index=['d', 'b', 'a', 'c'])
obj2

Output:
d 4
b 7
a -5
c 3
dtype: int64
```

```
obj2.index
Output: Index(['d', 'b', 'a', 'c'],
dtype='object')
```

```
obj2.values
Output: array([ 4, 7, -5, 3], dtype=int64)
```

```
obj2['a']
Output: -5
```

```
obj2.a
Output: -5
```

```
obj2['d']=10
obj2[['d', 'c', 'a']]
Output:
d 10
c 3
a -5
dtype: int64
```

```
obj2[:2]
Output:
d 10
b 7
dtype: int64
```

Series – array/dict operations

- numpy array operations can also be applied, which will preserve the index-value link

```
obj2[obj2>0]
Output:
d 10
b 7
c 3
dtype: int64
```

```
obj2**2
Output:
d 100
b 49
a 25
c 9
dtype: int64
```

- Can be constructed from a dict directly.

```
obj3 = Series({'a': 10,
'b': 5, 'c': 30})
obj3

Output:
a 10
b 5
c 30
dtype: int64
```

DataFrame

- A DataFrame is a tabular data structure comprised of rows and columns, akin to a spreadsheet or database table.
- It can be treated as an order collection of columns
  - Each column can be a different data type
  - Have both row and column indices

```
data = {'state': ['Ohio', 'Ohio',
'Ohio', 'Nevada', 'Nevada’], 'year':
[2000, 2001, 2002, 2001, 2002], 'pop':
[1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)
frame
```

```
Output:
   pop state year
0 1.5 Ohio 2000
1 1.7 Ohio 2001
2 3.6 Ohio 2002
3 2.4 Nevada 2001
4 2.9 Nevada 2002
```

DataFrame: Create

```
from pandas import DataFrame

data_dict = {'col1' : [1, 2, 3, 4],
             'col2' : [10, 20, 30, 40]}
indices = ['a', 'b', 'c', 'd']
df = DataFrame(data_dict, index = indices)
```

```
df
Output:
      col1  col2
a      1     10
b      2     20
c     3      30
d     4      40
```

```
df2=DataFrame.from_items( [('col1', [1, 2, 3]),
                               ('col2', [4, 5, 6])])
```

```
df2
Output:
      col1  col2
0      1     4
1      2     5
2     3      6
```

DataFrame: Create

```
pop = {'Nevada': {2001: 2.9, 2002: 2.9}, 'Ohio':
{2002: 3.6, 2001: 1.7, 2000: 1.5}}
frame3 = DataFrame(pop)
frame3
```

Output:

```
        Nevada   Ohio
2000     NaN     1.5
2001     2.9     1.7
2002    2.9     3.6
```

DataFrame: index, columns, values

```
frame3.index
Output:
Int64Index([2000, 2001, 2002], dtype='int64')

frame3.columns
Output:
Index(['Nevada', 'Ohio'], dtype='object')
```

```
frame3.values
Output:
array([  [ nan, 1.5],
        [ 2.9, 1.7],
         [ 2.9, 3.6]])
```

```
frame3
Output:
        Nevada   Ohio
2000     NaN     1.5
2001     2.9     1.7
2002   2.9       3.6
```

```
frame3.index.name = 'year'
frame3.columns.name='state'
frame3

Output:
state Nevada Ohio
year
2000 NaN 1.5
2001 2.9 1.7
2002 2.9 3.6
```

DataFrame: Retrieving a Column

```
frame
Output:
   pop  state   year
0  1.5  Ohio    2000
1  1.7  Ohio    2001
2  3.6  Ohio    2002
3  2.4  Nevada  2001
4  2.9  Nevada  2002
```

- A column in a DataFrame can be retrieved as a Series by dict-like notation or as attribute
- Series index and name have been kept/set appropriately

```
frame['state']
Output:
0 Ohio
1 Ohio
2 Ohio
3 Nevada
4 Nevada
Name: state, dtype: object
```

```
frame.state
Output:
0 Ohio
1 Ohio
2 Ohio
3 Nevada
4 Nevada
Name: state, dtype: object
```

```
type(frame['state'])
Output: pandas.core.series.Series
```

DataFrame: Getting Rows

- loc for using indexes and iloc for using positions

```
frame2
Output:
    year state  pop  debt
A   2000 Ohio   1.5  NaN
B   2001 Ohio   1.7  NaN
C   2002 Ohio   3.6  NaN
D   2001 Nevada 2.4  NaN
E   2002 Nevada 2.9  NaN
```

```
frame2.loc['A']
Output:
year 2000
state Ohio
pop 1.5
debt NaN
Name: A, dtype:
object
```

```
frame2.loc[['A', 'B']]
Output:
    year  state pop debt
A   2000  Ohio  1.5 NaN
B   2001  Ohio  1.7 NaN
```

```
type(frame2.loc['A'])
Output:
pandas.core.series.Series
```

```
type(frame2.loc[['A', 'B']])
Output:
pandas.core.frame.DataFrame
```

More on DataFrame indexing

```
data
Output:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
```

```
frame = DataFrame(data,
index=['r1', 'r2', 'r3'],
columns=['c1', 'c2', 'c3'])
```

```
frame
Output:
   c1 c2 c3
r1 0  1  2
r2 3  4  5
r3 6  7  8
```

```
frame['c1']
Output:
r1 0
r2 3
r3 6
Name: c1, dtype: int64
```

```
frame.loc['r1']
Output:
c1 0
c2 1
c3 2
Name: r1, dtype:
int64
```

```
frame['c1']['r1']
Output: 0
```

```
frame[['c1', 'c3']]
Output:
   c1 c3
r1 0  2
r2 3  5
r3 6  8
```

```
frame.loc[['r1','r3']]
Output:
   c1 c2 c3
r1 0  1  2
r3 6  7  8
```

```
frame.iloc[:2]
Output:
   c1 c2 c3
r1 0  1  2
r2 3  4  5
```

More on DataFrame indexing - 2

```
frame
Output:
   c1 c2 c3
r1 0  1  2
r2 3  4  5
r3 6  7  8
```

```
frame[frame['c1']>0]
Output:
   c1 c2 c3
r2 3  4  5
r3 6  7  8
```

```
frame['c1']>0
Output:
r1 False
r2 True
r3 True
Name: c1, dtype: bool
```

```
frame < 3
Output:
    c1    c2    c3
r1 True  True  True
r2 False False False
r3 False False False
```

```
frame[frame<3] = 3
```

```
frame
Output:
   c1 c2 c3
r1 3  3  3
r2 3  4  5
r3 6  7  8
```

DataFrame – modifying columns

```
frame2
Output:
    year state  pop  debt
A   2000 Ohio   1.5  NaN
B   2001 Ohio   1.7  NaN
C   2002 Ohio   3.6  NaN
D   2001 Nevada 2.4  NaN
E   2002 Nevada 2.9  NaN
```

Rows or individual elements can be modified similarly. Using loc or iloc.

```
frame2['debt'] = 0

frame2
Output:
    year state  pop debt
A   2000 Ohio   1.5  0
B   2001 Ohio   1.7  0
C   2002 Ohio   3.6  0
D   2001 Nevada 2.4  0
E   2002 Nevada 2.9  0
```

```
frame2['debt'] = range(5)

frame2
Output:
    year state  pop debt
A   2000 Ohio   1.5  0
B   2001 Ohio   1.7  1
C   2002 Ohio   3.6  2
D   2001 Nevada 2.4  3
E   2002 Nevada 2.9  4
```

```
val = Series([10, 10, 10],
index = ['A', 'C', 'D'])

frame2['debt'] = val

frame2
Output:
   year state pop  debt
A  2000 Ohio   1.5 10.0
B  2001 Ohio   1.7 NaN
C  2002 Ohio   3.6 10.0
D  2001 Nevada 2.4 10.0
E  2002 Nevada 2.9 NaN
```

Removing rows/columns

```
frame
Output:
   c1 c2 c3
r1 0  1  2
r2 3  4  5
r3 6  7  8
```

```
frame.drop(['r1'])
Output:
        c1 c2 c3
     r2 3  4  5
     r3 6  7  8
```

```
frame.drop(['r1','r3'])
Output:
   c1 c2 c3
r2 3  4  5
```

```
frame.drop(['c1'], axis=1)
Output:
   c2 c3
r1 1  2
r2 4  5
r3 7  8
```

```
frame
Output:
   c1 c2 c3
r1 0  1  2
r2 3  4  5
r3 6  7  8
```

Does not change the old frame,
it returns a new frame

Function application and mapping

- DataFrame.applymap(f) applies f to every entry
- DataFrame.apply(f) applies f to every column or row

```
frame
Output:
   c1 c2 c3
r1 0  1  2
r2 3  4  5
r3 6  7  8
```

```
def max_minus_min(x): return max(x)-min(x)
frame.apply(max_minus_min)
Output:
c1 6
c2 6
c3 6
dtype: int64
```

```
def square(x): return x**2
frame.applymap(square)
Output:
   c1 c2 c3
r1 0  1  4
r2 9  16 25
r3 36 49 64
```

```
frame.apply(max_minus_min, axis=1)
Output:
r1 2
r2 2
r3 2
dtype: int64
```

Other DataFrame functions

- `head()  First few lines`
- `tail(5) Last 5 lines`
- mean()
  - Mean(axis=0, skipna=True)
- sum()
- describe(): return summary statistics of each column
  - for numeric data: mean, std, max, min, 25%, 50%, 75%, etc.
  - For non-numeric data: count, uniq, most-frequent item, etc.

DataFrame: I/O

```
df =  pandas.read_csv('data.csv')

df.to_csv('data.csv')
```

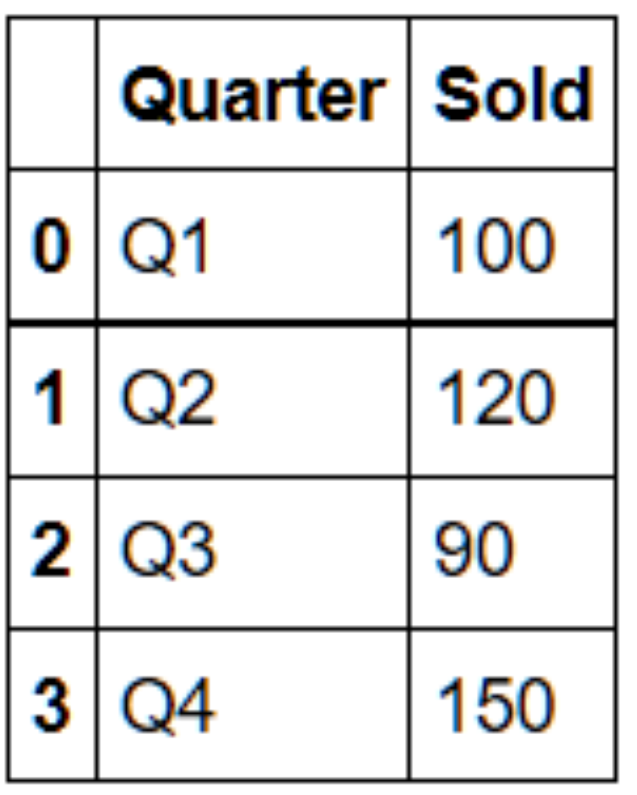
```
df = pandas.read_excel('data.xlsx', 'Sheet1', index_col=None,
na_values=['NA'])

df.to_excel('data.xlsx', sheet_name='Sheet1')
```

Quiz

What is the VALUE and TYPE of each of the following?

1. `df['Quarter']`
2. `df[ ['Quarter'] ]`
3. `df[df['Quarter']=='Q1']`
4. `df[ df['Sold'] < 110 ]`

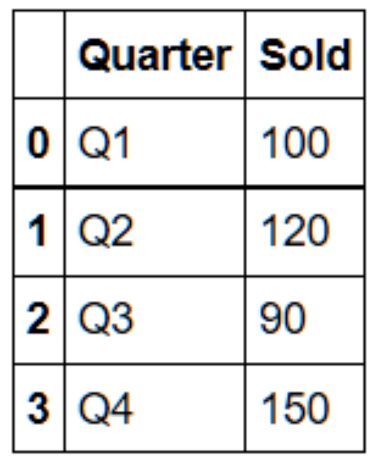
- df



	Quarter	Sold
0	Q1	100
1	Q2	120
2	Q3	90
3	Q4	150

Quiz Answer

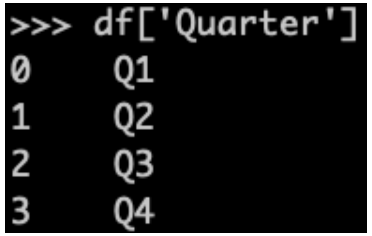
- df



	Quarter	Sold
0	Q1	100
1	Q2	120
2	Q3	90
3	Q4	150

What is the VALUE and TYPE of each of the following?

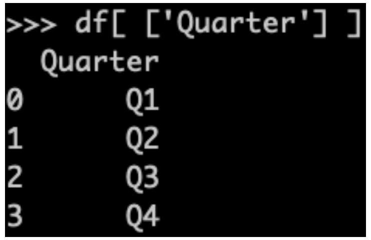
1. `df['Quarter']`



```
>>> df['Quarter']
0      Q1
1      Q2
2      Q3
3      Q4
```

Series

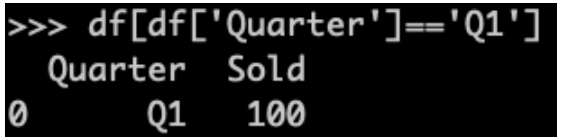
2. `df[ ['Quarter'] ]`



```
>>> df[ ['Quarter'] ]
  Quarter
0      Q1
1      Q2
2      Q3
3      Q4
```

Dataframe

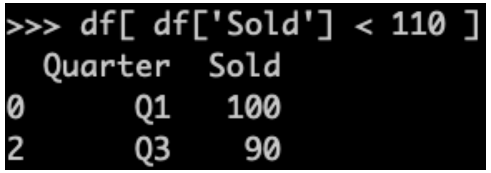
3. `df[df['Quarter']=='Q1']`



```
>>> df[df['Quarter']=='Q1']
  Quarter  Sold
0      Q1   100
```

Dataframe

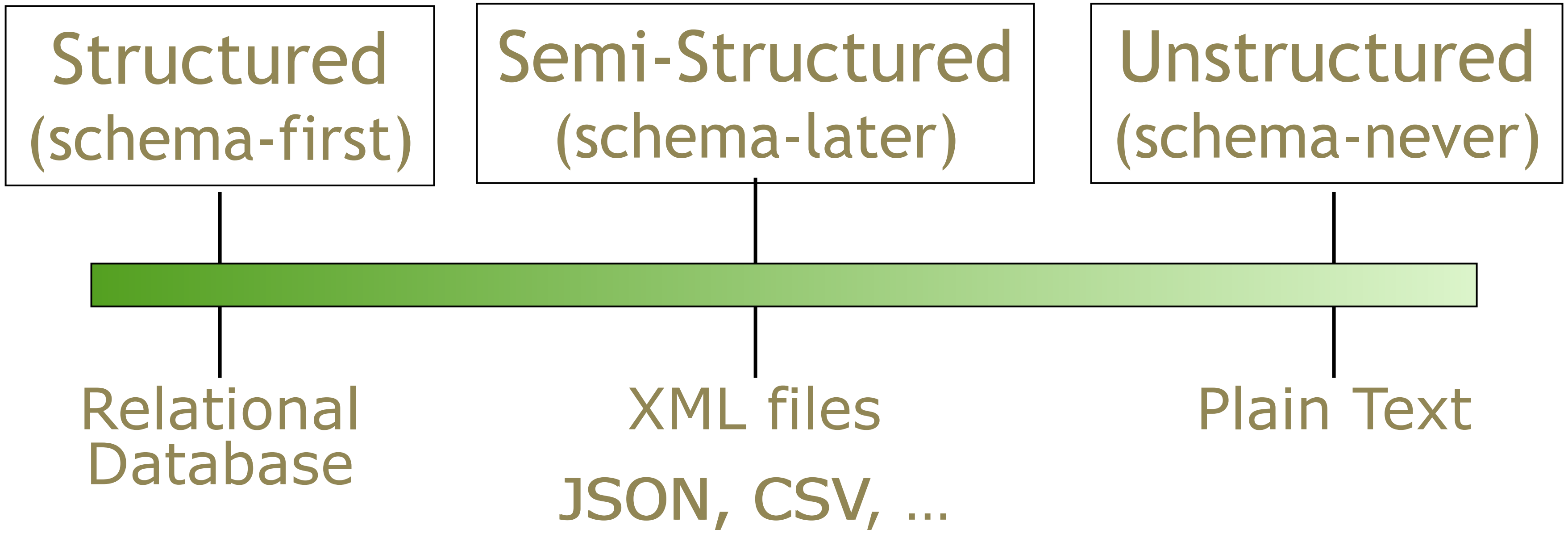
4. `df[ df['Sold'] < 110 ]`



```
>>> df[ df['Sold'] < 110 ]
  Quarter  Sold
0      Q1   100
2      Q3    90
```

Dataframe

Data Organization Spectrum



Structured (schema-first)

Semi-Structured (schema-later)

Unstructured (schema-never)

Relational Database

XML files

JSON, CSV, …

Plain Text

Starting from next lecture!