What is Pandas?

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

Installation of Pandas

In [1]:

```
# pip install pandas
```

Import Pandas as pd

Pandas is usually imported under the pd alias.

- alias: In Python alias are an alternate name for referring to the same thing.

In [3]:

```
import pandas as pd
# From pandas import *
# From pandas import pd
```

What is a Series?

- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.

In [10]:

```
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1.0]) #A Pandas Series is a one-dimensional array of indexed

data
```

Out[10]:

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

Labels

- If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

- This label can be used to access a specified value.

In [11]:
```
print( data[0])
```

```
0.25
```

Create Labels

- With the (index) argument, you can name your own labels.

In [12]:
```
data.index
```

Out[12]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [13]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0],index=['a', 'b', 'c', 'd'])
data
```

Out[13]:
```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

In [15]:
```
print( data['b'])
```

```
0.5
```

In [16]:
```
data.values
```

Out[16]:
```
array([0.25, 0.5 , 0.75, 1.  ])
```

In [17]:
```
data[:]
```

Out[17]:
```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

In [18]:
```
data[:2]
```

Out[18]:
```
a    0.25
b    0.50
dtype: float64
```

In [19]:
```
data[2:]
```

Out[19]:
```
c    0.75
d    1.00
dtype: float64
```

Series-as-dictionary

In [20]:
```
#Series-as-dictionary
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}

population = pd.Series(population_dict)
population
```

Out[20]:

```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

In [21]:

```
population['California']
```

Out[21]:

```
38332521
```

In [24]:

```
population[:'New York']
```

Out[24]:

```
California    38332521
Texas         26448193
New York      19651127
dtype: int64
```

DataFrames

DataFrame is an analog of a two-dimensional array with both flexible row indices and flexible column names

A DataFrame is a collection of Series objects, and a singlecolumn DataFrame can be constructed from a single Series

In [31]:

```
import pandas as pd

data = {"calories": [420, 380, 390],
        "duration": [50, 40, 45]}

#load data into a DataFrame object
df = pd.DataFrame(data)

print(df)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

In [32]:

```
df.index #index labels
```

Out[32]:

```
RangeIndex(start=0, stop=3, step=1)
```

In [34]:

```
df.columns #column labels
```

Out[34]:

```
Index(['calories', 'duration'], dtype='object')
```

In [35]:

```
df['calories']
```

Out[35]:

```
0    420
1    380
2    390
Name: calories, dtype: int64
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

In [38]:

```
# #refer to the row index:
print(df.loc[0])
```

```
calories    420
duration     50
Name: 0, dtype: int64
```

In [39]:

```
# Example : Return row 0 and 1:
#use a list of indexes:
print(df.loc[[0, 1]])
```

```
   calories  duration
0       420        50
1       380        40
```

In [40]:

```
# Named Indexes
import pandas as pd

data = {"calories": [420, 380, 390],
        "duration": [50, 40, 45]}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

In [41]:

```
print(df.loc["day2"])
```

```
calories    380
duration     40
Name: day2, dtype: int64
```

Extracting and transforming data (Load Files Into a DataFrame)

In [43]:

```
# Read file .CSV
df1 = pd.read_csv('F:\\Job\\FCDS\\Data Science Methodology\\Data sets\\Online Retail.csv')
df1
```

Out[43]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France

541909 rows × 8 columns

In [45]:

```
# Read in filename and set the index: election (index_col='column name')
df = pd.read_csv('F:\\Job\\FCDS\\Data Science Methodology\\Data sets\\Online Retail.csv',
                 index_col='InvoiceNo')
df
```

Out[45]:

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...
581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France
581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France
581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France
581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France
581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France

541909 rows × 7 columns

In [46]:

```
# Indexing using square brackets
# data name['column name']['row name']
df['Description']['536365']
```

Out[46]:

```
536365     WHITE HANGING HEART T-LIGHT HOLDER
536365                    WHITE METAL LANTERN
536365         CREAM CUPID HEARTS COAT HANGER
536365    KNITTED UNION FLAG HOT WATER BOTTLE
536365         RED WOOLLY HOTTIE WHITE HEART.
536365           SET 7 BABUSHKA NESTING BOXES
536365      GLASS STAR FROSTED T-LIGHT HOLDER
Name: Description, dtype: object
```

In [47]:

```
# Using column attribute and row label
df.Description['581587']
```

Out[47]:

```
581587           CIRCUS PARADE LUNCH BOX
581587     PLASTERS IN TIN CIRCUS PARADE
581587          PLASTERS IN TIN STRONGMAN
581587          ALARM CLOCK BAKELIKE PINK
581587          ALARM CLOCK BAKELIKE RED
581587         ALARM CLOCK BAKELIKE GREEN
581587         ALARM CLOCK BAKELIKE IVORY
581587    CHILDRENS APRON SPACEBOY DESIGN
581587                SPACEBOY LUNCH BOX
581587        CHILDRENS CUTLERY SPACEBOY
581587        PACK OF 20 SPACEBOY NAPKINS
581587       CHILDREN'S APRON DOLLY GIRL
581587      CHILDRENS CUTLERY DOLLY GIRL
581587    CHILDRENS CUTLERY CIRCUS PARADE
581587      BAKING SET 9 PIECE RETROSPOT
Name: Description, dtype: object
```

Pandas - Analyzing DataFrames

The head() method returns the headers and a specified number of rows, starting from the top.

In [61]:

```
# if the number of rows is not specified, the head() method will return the top 5 rows.
print(df.head())
```

```
          StockCode                          Description  Quantity  \
InvoiceNo
536365       85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
536365        71053                  WHITE METAL LANTERN         6
536365       84406B       CREAM CUPID HEARTS COAT HANGER         8
536365       84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
536365       84029E       RED WOOLLY HOTTIE WHITE HEART.         6

              InvoiceDate  UnitPrice  CustomerID         Country
InvoiceNo
536365     12/1/2010 8:26       2.55     17850.0  United Kingdom
536365     12/1/2010 8:26       3.39     17850.0  United Kingdom
536365     12/1/2010 8:26       2.75     17850.0  United Kingdom
536365     12/1/2010 8:26       3.39     17850.0  United Kingdom
536365     12/1/2010 8:26       3.39     17850.0  United Kingdom
```

In [62]:

```
print(df.head(10))
```

```
          StockCode                          Description  Quantity  \
InvoiceNo
536365       85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
536365        71053                  WHITE METAL LANTERN         6
536365       84406B       CREAM CUPID HEARTS COAT HANGER         8
536365       84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
536365       84029E       RED WOOLLY HOTTIE WHITE HEART.         6
536365        22752         SET 7 BABUSHKA NESTING BOXES         2
536365        21730    GLASS STAR FROSTED T-LIGHT HOLDER         6
536366        22633               HAND WARMER UNION JACK         6
```

```
536366        22632          HAND WARMER RED POLKA DOT         6
536367        84879       ASSORTED COLOUR BIRD ORNAMENT       32

                InvoiceDate  UnitPrice  CustomerID         Country
InvoiceNo
536365     12/1/2010 8:26       2.55     17850.0  United Kingdom
536365     12/1/2010 8:26       3.39     17850.0  United Kingdom
536365     12/1/2010 8:26       2.75     17850.0  United Kingdom
536365     12/1/2010 8:26       3.39     17850.0  United Kingdom
536365     12/1/2010 8:26       3.39     17850.0  United Kingdom
536365     12/1/2010 8:26       7.65     17850.0  United Kingdom
536365     12/1/2010 8:26       4.25     17850.0  United Kingdom
536366     12/1/2010 8:28       1.85     17850.0  United Kingdom
536366     12/1/2010 8:28       1.85     17850.0  United Kingdom
536367     12/1/2010 8:34       1.69     13047.0  United Kingdom
```

The tail() method returns the headers and a specified number of rows, starting from the bottom.

In [63]:

```
# Print the last 5 rows of the DataFrame
print(df.tail())
```

```
          StockCode                     Description  Quantity  \
InvoiceNo
581587        22613     PACK OF 20 SPACEBOY NAPKINS        12
581587        22899    CHILDREN'S APRON DOLLY GIRL          6
581587        23254   CHILDRENS CUTLERY DOLLY GIRL          4
581587        23255 CHILDRENS CUTLERY CIRCUS PARADE         4
581587        22138   BAKING SET 9 PIECE RETROSPOT          3

                InvoiceDate  UnitPrice  CustomerID Country
InvoiceNo
581587     12/9/2011 12:50       0.85     12680.0  France
581587     12/9/2011 12:50       2.10     12680.0  France
581587     12/9/2011 12:50       4.15     12680.0  France
581587     12/9/2011 12:50       4.15     12680.0  France
581587     12/9/2011 12:50       4.95     12680.0  France
```

The DataFrames object has a method called info(), that gives you more information about the data set.

In [64]:

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 541909 entries, 536365 to 581587
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   StockCode    541909 non-null  object
 1   Description  540455 non-null  object
 2   Quantity     541909 non-null  int64
 3   InvoiceDate  541909 non-null  object
 4   UnitPrice    541909 non-null  float64
 5   CustomerID   406829 non-null  float64
 6   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(4)
memory usage: 53.1+ MB
None
```

In [66]:

```
print(df.describe())
```

```
            Quantity      UnitPrice     CustomerID
count  541909.000000  541909.000000  406829.000000
mean        9.552250       4.611114   15287.690570
std       218.081158      96.759853    1713.600303
min    -80995.000000  -11062.060000   12346.000000
25%         1.000000       1.250000   13953.000000
50%         3.000000       2.080000   15152.000000
75%        10.000000       4.130000   16791.000000
max     80995.000000   38970.000000   18287.000000
```

merge datasets

In [116…

```
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
print(df1)
print(df2)
```

```
  employee        group
0      Bob   Accounting
1     Jake  Engineering
2     Lisa  Engineering
3      Sue           HR
  employee  hire_date
0     Lisa       2004
1      Bob       2008
2     Jake       2012
3      Sue       2014
```

In [117…

```
df3 = pd.merge(df1, df2)
df3
```

Out[117…

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

Accessors

A more efficient and more programmatically reusable method of accessing data in a DataFrame is by using accessors

- loc - accesses using lables
- iloc - accesses using index positions

Both accessors use left bracket, row specifier, comma, column specifier, right bracket as syntax

In [67]:

```
# Using the .loc accessor
df.loc['581587', 'Country']
```

Out[67]:

```
InvoiceNo
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
581587    France
Name: Country, dtype: object
```

In [68]:

```
# Using the .iloc accessor
```

```
df.iloc[4, 6]
```

Out[68]:

```
'United Kingdom'
```

Selecting only some columns

- When using bracket-indexing without the .loc or .iloc accessors, the result returned can be an individual value, Pandas Series, or Pandas DataFrame.
- To ensure the return value is a DataFrame, use a nested list within square brackets

In [69]:

```
df_new = df[['CustomerID','Country']]
df_new
```

Out[69]:

	CustomerID	Country
InvoiceNo		
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
...	...	...
581587	12680.0	France
581587	12680.0	France
581587	12680.0	France
581587	12680.0	France
581587	12680.0	France

541909 rows × 2 columns

In [70]:

```
df['Country']
```

Out[70]:

```
InvoiceNo
536365    United Kingdom
536365    United Kingdom
536365    United Kingdom
536365    United Kingdom
536365    United Kingdom
               ...
581587            France
581587            France
581587            France
581587            France
581587            France
Name: Country, Length: 541909, dtype: object
```

In [71]:

```
type(df.Country)
```

Out[71]:

```
pandas.core.series.Series
```

In [72]:

```
df['Country'][1:4] # Part of the Country column
```

Out[72]:
```
InvoiceNo
536365    United Kingdom
536365    United Kingdom
536365    United Kingdom
Name: Country, dtype: object
```

In [73]:
```
df['Country'][4] # The value associated with 536365
```

Out[73]:
```
'United Kingdom'
```

In [74]:
```
df.loc[:, 'CustomerID':'Country'] # All rows, some columns
```

Out[74]:

	CustomerID	Country
InvoiceNo		
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
536365	17850.0	United Kingdom
...	...	...
581587	12680.0	France
581587	12680.0	France
581587	12680.0	France
581587	12680.0	France
581587	12680.0	France

541909 rows × 2 columns

In [75]:
```
df.loc['536365':'581587',:] # Some rows, all columns
```

Out[75]:

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...
581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France
581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							
581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France
581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France
581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France

541909 rows × 7 columns

Filtering DataFrames

In [76]:

```
#Creating a Boolean Series
df.UnitPrice > 60
```

Out[76]:

```
InvoiceNo
536365    False
536365    False
536365    False
536365    False
536365    False
          ...
581587    False
581587    False
581587    False
581587    False
581587    False
Name: UnitPrice, Length: 541909, dtype: bool
```

In [77]:

```
df[df.UnitPrice > 60]
```

Out[77]:

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							
536392	22827	RUSTIC SEVENTEEN DRAWER SIDEBOARD	1	12/1/2010 10:29	165.00	13705.0	United Kingdom
536544	DOT	DOTCOM POSTAGE	1	12/1/2010 14:32	569.77	NaN	United Kingdom
536592	DOT	DOTCOM POSTAGE	1	12/1/2010 17:06	607.49	NaN	United Kingdom
536676	21769	VINTAGE POST OFFICE CABINET	1	12/2/2010 12:18	79.95	16752.0	United Kingdom
536835	22655	VINTAGE RED KITCHEN CABINET	1	12/2/2010 18:06	295.00	13145.0	United Kingdom
...	...	...	...	...	...	...	...
581238	DOT	DOTCOM POSTAGE	1	12/8/2011 10:53	1683.75	NaN	United Kingdom
581439	DOT	DOTCOM POSTAGE	1	12/8/2011 16:30	938.59	NaN	United Kingdom
581492	DOT	DOTCOM POSTAGE	1	12/9/2011 10:03	933.17	NaN	United Kingdom
581498	DOT	DOTCOM POSTAGE	1	12/9/2011 10:26	1714.17	NaN	United Kingdom

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							
C581499	M	Manual	-1	12/9/2011 10:28	224.69	15498.0	United Kingdom

1188 rows × 7 columns

In [78]:

```
x = df.UnitPrice > 60
df[x]
```

Out[78]:

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							
536392	22827	RUSTIC SEVENTEEN DRAWER SIDEBOARD	1	12/1/2010 10:29	165.00	13705.0	United Kingdom
536544	DOT	DOTCOM POSTAGE	1	12/1/2010 14:32	569.77	NaN	United Kingdom
536592	DOT	DOTCOM POSTAGE	1	12/1/2010 17:06	607.49	NaN	United Kingdom
536676	21769	VINTAGE POST OFFICE CABINET	1	12/2/2010 12:18	79.95	16752.0	United Kingdom
536835	22655	VINTAGE RED KITCHEN CABINET	1	12/2/2010 18:06	295.00	13145.0	United Kingdom
...	...	...	...	...	...	...	...
581238	DOT	DOTCOM POSTAGE	1	12/8/2011 10:53	1683.75	NaN	United Kingdom
581439	DOT	DOTCOM POSTAGE	1	12/8/2011 16:30	938.59	NaN	United Kingdom
581492	DOT	DOTCOM POSTAGE	1	12/9/2011 10:03	933.17	NaN	United Kingdom
581498	DOT	DOTCOM POSTAGE	1	12/9/2011 10:26	1714.17	NaN	United Kingdom
C581499	M	Manual	-1	12/9/2011 10:28	224.69	15498.0	United Kingdom

1188 rows × 7 columns

Combining filters

In [79]:

```
df[(df.UnitPrice >= 50) & (df.CustomerID < 200)] # Both conditions
```

Out[79]:

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							

In [80]:

```
df[(df.UnitPrice >= 50) | (df.CustomerID < 200)] # Either condition
```

Out[80]:

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							

	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
InvoiceNo							
536392	22827	RUSTIC SEVENTEEN DRAWER SIDEBOARD	1	12/1/2010 10:29	165.00	13705.0	United Kingdom
536540	C2	CARRIAGE	1	12/1/2010 14:05	50.00	14911.0	EIRE
536544	22769	CHALKBOARD KITCHEN ORGANISER	1	12/1/2010 14:32	51.02	NaN	United Kingdom
536544	DOT	DOTCOM POSTAGE	1	12/1/2010 14:32	569.77	NaN	United Kingdom
536592	22503	CABIN BAG VINTAGE PAISLEY	1	12/1/2010 17:06	59.53	NaN	United Kingdom
...	...	...	...	...	...	...	...
581238	DOT	DOTCOM POSTAGE	1	12/8/2011 10:53	1683.75	NaN	United Kingdom
581439	DOT	DOTCOM POSTAGE	1	12/8/2011 16:30	938.59	NaN	United Kingdom
581492	DOT	DOTCOM POSTAGE	1	12/9/2011 10:03	933.17	NaN	United Kingdom
581498	DOT	DOTCOM POSTAGE	1	12/9/2011 10:26	1714.17	NaN	United Kingdom
C581499	M	Manual	-1	12/9/2011 10:28	224.69	15498.0	United Kingdom

1417 rows × 7 columns

Creating a Series

A Pandas Series is a one-dimensional array of indexed data (one column)

In [81]:

```
data = { 'Name':pd.Series(['Huda','Mohamed','Hossam','Mina']),
        'Age': pd.Series([25,30,50,36])}
data
```

Out[81]:

```
{'Name': 0        Huda
 1    Mohamed
 2     Hossam
 3       Mina
 dtype: object,
 'Age': 0    25
 1    30
 2    50
 3    36
 dtype: int64}
```

In [82]:

```
prices = [10.70, 10.86, 10.74, 10.71, 10.79]
shares = pd.Series(prices)
shares
```

Out[82]:

```
0    10.70
1    10.86
2    10.74
3    10.71
4    10.79
dtype: float64
```

In [83]:

```
days = ['Mon', 'Tue', 'Wed', 'Thur', 'Fri']
shares = pd.Series(prices, index=days)
shares
```

Out[83]:

```
Mon     10.70
Tue     10.86
Wed     10.74
Thur    10.71
Fri     10.79
dtype: float64
```

Examining an index

In [84]:

```
print(shares.index)
print(shares.index[2])
print(shares.index[:2])
print(shares.index[-2:])
print(shares.index.name)
```

```
Index(['Mon', 'Tue', 'Wed', 'Thur', 'Fri'], dtype='object')
Wed
Index(['Mon', 'Tue'], dtype='object')
Index(['Thur', 'Fri'], dtype='object')
None
```

In [85]:

```
# Modifying index name
shares.index.name = 'weekday'
shares
```

Out[85]:

```
weekday
Mon     10.70
Tue     10.86
Wed     10.74
Thur    10.71
Fri     10.79
dtype: float64
```

In [86]:

```
# Modifying all index entries
shares.index = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
shares
```

Out[86]:

```
Monday       10.70
Tuesday      10.86
Wednesday    10.74
Thursday     10.71
Friday       10.79
dtype: float64
```

In [ ]:

In [87]:

```
stocks = pd.DataFrame([['2016-10-03', 31.50, 14070500, 'CSCO'],
                       ['2016-10-03', 112.52, 21701800, 'AAPL'],
                       ['2016-10-03', 57.42, 19189500, 'MSFT'],
                       ['2016-10-04', 113.00, 29736800, 'AAPL'],
                       ['2016-10-04', 57.24, 20085900, 'MSFT'],
                       ['2016-10-04', 31.35, 18460400, 'CSCO'],
                       ['2016-10-05', 57.64, 16726400, 'MSFT'],
                       ['2016-10-05', 31.59, 11808600, 'CSCO'],
                       ['2016-10-05', 113.05, 21453100, 'AAPL']],
                      columns=['Date', 'Close', 'Volume', 'Symbol'])
stocks
```

Out[87]:

	Date	Close	Volume	Symbol

	Date	Close	Volume	Symbol
0	2016-10-03	31.50	14070500	CSCO
1	2016-10-03	112.52	21701800	AAPL
2	2016-10-03	57.42	19189500	MSFT
3	2016-10-04	113.00	29736800	AAPL
4	2016-10-04	57.24	20085900	MSFT
5	2016-10-04	31.35	18460400	CSCO
6	2016-10-05	57.64	16726400	MSFT
7	2016-10-05	31.59	11808600	CSCO
8	2016-10-05	113.05	21453100	AAPL

In [88]:

```
# Setting index
stocks = stocks.set_index(['Symbol', 'Date'])
stocks
```

Out[88]:

		Close	Volume
Symbol	**Date**		
CSCO	**2016-10-03**	31.50	14070500
AAPL	**2016-10-03**	112.52	21701800
MSFT	**2016-10-03**	57.42	19189500
AAPL	**2016-10-04**	113.00	29736800
MSFT	**2016-10-04**	57.24	20085900
CSCO	**2016-10-04**	31.35	18460400
MSFT	**2016-10-05**	57.64	16726400
CSCO	**2016-10-05**	31.59	11808600
AAPL	**2016-10-05**	113.05	21453100

In [89]:

```
# MultiIndex on DataFrame
stocks.index
```

Out[89]:

```
MultiIndex([('CSCO', '2016-10-03'),
            ('AAPL', '2016-10-03'),
            ('MSFT', '2016-10-03'),
            ('AAPL', '2016-10-04'),
            ('MSFT', '2016-10-04'),
            ('CSCO', '2016-10-04'),
            ('MSFT', '2016-10-05'),
            ('CSCO', '2016-10-05'),
            ('AAPL', '2016-10-05')],
           names=['Symbol', 'Date'])
```

In [90]:

```
print(stocks.index.name)
```

```
None
```

In [91]:

```
print(stocks.index.names)
```

```
['Symbol', 'Date']
```

In [92]:

```
# Sorting index
stocks = stocks.sort_index()
stocks
```

Out[92]:

		Close	Volume
Symbol	**Date**		
AAPL	**2016-10-03**	112.52	21701800
	2016-10-04	113.00	29736800
	2016-10-05	113.05	21453100
CSCO	**2016-10-03**	31.50	14070500
	2016-10-04	31.35	18460400
	2016-10-05	31.59	11808600
MSFT	**2016-10-03**	57.42	19189500
	2016-10-04	57.24	20085900
	2016-10-05	57.64	16726400

In [93]:

```
# Indexing (individual row)
stocks.loc[('CSCO', '2016-10-04')]
```

Out[93]:

```
Close           31.35
Volume    18460400.00
Name: (CSCO, 2016-10-04), dtype: float64
```

In [94]:

```
stocks.loc[('CSCO', '2016-10-04'), 'Volume']
```

Out[94]:

```
18460400.0
```

In [95]:

```
stocks.loc['AAPL']
```

Out[95]:

	Close	Volume
Date		
2016-10-03	112.52	21701800
2016-10-04	113.00	29736800
2016-10-05	113.05	21453100

In [96]:

```
# Slicing (outermost index)
stocks.loc['CSCO':'MSFT']
```

Out[96]:

		Close	Volume
Symbol	**Date**		
CSCO	**2016-10-03**	31.50	14070500
	2016-10-04	31.35	18460400
	2016-10-05	31.59	11808600
MSFT	**2016-10-03**	57.42	19189500
	2016-10-04	57.24	20085900
	2016-10-05	57.64	16726400

In [ ]:

```

```

Stacking & unstacking DataFrames

In [97]:

```
trials = pd.DataFrame([[1, 'A', 'F', 5],
                       [2, 'A', 'M', 3],
                       [3, 'B', 'F', 8],
                       [4, 'B', 'M', 9]],
                      columns=['id', 'treatment', 'gender', 'response'])
trials
```

Out[97]:

```
   id  treatment  gender  response
0   1          A       F         5
1   2          A       M         3
2   3          B       F         8
3   4          B       M         9
```

In [98]:

```
trials = trials.set_index(['treatment', 'gender'])
trials
```

Out[98]:

```
                   id  response
treatment  gender
        A       F   1         5
                M   2         3
        B       F   3         8
                M   4         9
```

In [99]:

```
# Unstacking a multi-index
trials.unstack(level='gender')
```

Out[99]:

```
              id     response
   gender   F  M      F    M
treatment
        A   1  2      5    3
        B   3  4      8    9
```

In [100…

```
# Stacking DataFrames
trials_by_gender = trials.unstack(level='gender')
trials_by_gender
```

Out[100…

```
              id     response
   gender   F  M      F    M
treatment
        A   1  2      5    3
```

	id		response	
gender	**F**	**M**	**F**	**M**
treatment				
B	3	4	8	9

In [101…

```
trials_by_gender.stack(level='gender')
```

Out[101…

		id	response
treatment	**gender**		
A	**F**	1	5
	M	2	3
B	**F**	3	8
	M	4	9

In [102…

```
stacked = trials_by_gender.stack(level='gender')
stacked
```

Out[102…

		id	response
treatment	**gender**		
A	**F**	1	5
	M	2	3
B	**F**	3	8
	M	4	9

In [103…

```
# Swapping levels
swapped = stacked.swaplevel(0, 1)
swapped
```

Out[103…

		id	response
gender	**treatment**		
F	**A**	1	5
M	**A**	2	3
F	**B**	3	8
M	**B**	4	9

In [104…

```
# Sorting rows
sorted_trials = swapped.sort_index()
sorted_trials
```

Out[104…

		id	response
gender	**treatment**		
F	**A**	1	5
	B	3	8

		id	response
gender	**treatment**		
M	**A**	2	3
	B	4	9

Categorical and groupby

In [105…

```
sales = pd.DataFrame(
    {
        'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],
        'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],
        'bread': [139, 237, 326, 456],
        'butter': [20, 45, 70, 98]
    }
)
sales
```

Out[105…

	weekday	city	bread	butter
0	Sun	Austin	139	20
1	Sun	Dallas	237	45
2	Mon	Austin	326	70
3	Mon	Dallas	456	98

In [106…

```
# filter and count
sales.loc[sales['weekday'] == 'Sun'].count()
```

Out[106…

```
weekday    2
city       2
bread      2
butter     2
dtype: int64
```

In [107…

```
# Groupby and count
sales.groupby('weekday').count()
```

Out[107…

	city	bread	butter
weekday			
Mon	2	2	2
Sun	2	2	2

Aggregation/Reduction

Some reducing functions

- mean()
- std()
- sum()
- first(), last()
- min(), max()

In [108…

```
# Groupby and sum
sales.groupby('weekday')['bread'].sum()
```

Out[108…

```
weekday
Mon    782
Sun    376
Name: bread, dtype: int64
```

In [109…

```
# Groupby and sum: multiple columns
sales
```

Out[109…

	weekday	city	bread	butter
0	Sun	Austin	139	20
1	Sun	Dallas	237	45
2	Mon	Austin	326	70
3	Mon	Dallas	456	98

In [110…

```
sales.groupby('weekday')[['bread','butter']].sum()
```

Out[110…

	bread	butter
weekday		
Mon	782	168
Sun	376	65

Groupby and aggregation

The .agg() method can be used with a tuple or list of aggregations as input. When applying multiple aggregations on multiple columns, the aggregated DataFrame has a multi-level column index.

In [111…

```
sales
```

Out[111…

	weekday	city	bread	butter
0	Sun	Austin	139	20
1	Sun	Dallas	237	45
2	Mon	Austin	326	70
3	Mon	Dallas	456	98

In [112…

```
sales.groupby('city')[['bread','butter']].max()
```

Out[112…

	bread	butter
city		
Austin	326	70
Dallas	456	98

In [113…

```
sales.groupby('city')[['bread','butter']].agg(['max','sum'])
```

Out[113…

```
         bread      butter
         max  sum   max  sum
city
Austin   326  465    70   90
Dallas   456  693    98  143
```

Aggregation functions

string names

- 'sum'
- 'mean'
- 'count'

In [114…

```
# Custom aggregation
def data_range(series):
    return series.max() - series.min()
```

In [115…

```
sales.groupby('weekday')[['bread', 'butter']].agg(data_range)
```

Out[115…

```
          bread  butter
weekday
    Mon     130      28
    Sun      98      25
```

In [ ]: