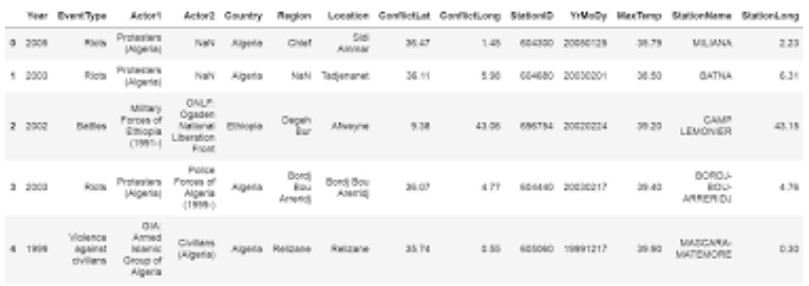
Dealing with data: Pandas

Numpy: Numerical computation of a large set of numbers

Scipy: Algorithms to solve problems.

Pandas: Manipulate data

Dataframe



	Year	EventType	Actor1	Actor2	Country	Region	Location	ConflictLat	ConflictLong	StationID	YrMoDy	MaxTemp	StationName	StationLong
0	2008	Riots	Protesters (Algeria)	NaN	Algeria	Chlef	Sidi Akkacar	36.47	1.48	604300	20080128	36.79	MILIANA	2.23
1	2003	Riots	Protesters (Algeria)	NaN	Algeria	NaN	Tadjenanet	36.11	5.90	604680	20030201	36.50	BATNA	6.31
2	2002	Battles	Military Forces of Ethiopia (1991-)	ONLF: Ogaden National Liberation Front	Ethiopia	Degeh Bur	Afweyne	9.38	43.96	696784	20020224	39.20	CAMP LEMONIER	43.15
3	2003	Riots	Protesters (Algeria)	Police Forces of Algeria (1999-)	Algeria	Bordj Bou Arreridj	Bordj Bou Arreridj	36.07	4.77	604440	20030217	39.40	BORDJ-BOU-ARRERIDJ	4.76
4	1999	Violence against civilians	GIA: Armed Islamic Group of Algeria	Civilians (Algeria)	Algeria	Relizane	Relizane	35.74	0.55	603060	19991217	39.90	MASCARA-MATEMORE	0.30

What is Pandas?

Numpy with even more Kool-aids!!!
…… which means… More syntactic sugars. ← Well… we need to learn even more that are specific to Pandas.

Tools to deal with the real world large dataset

Pandas is built on top of Numpy → We may encounter situations in which we need to use Numpy functions to manipulate parts of the dataframe.

What does that really mean?

Pandas' data structures are basically Numpy arrays with **labels**.

- For example, for a 2-dimensional array, the columns have names and rows have names.
- Column names (called 'columns') represent the kind of data, and the row names (called 'index') correspond to individual observations.

Note: Pandas data structure, dataframe, is not exactly the same as Numpy. Its memory structure is different and its behavior is also different. (We won't get into the technical details.)

Data types in Pandas

Series : 1-dimensional labeled array

Dataframe : multi-dimensional labeled array

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6, 4))

s = pd.Series([1, 3, 5, np.nan, 6, 8])
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

	0	1	2	3
0	1.512231	0.523552	0.867756	-1.992969
1	-1.346766	1.790476	-0.967559	1.085014
2	0.828109	-0.509792	2.551996	-0.161647
3	-1.586500	0.255919	0.089031	-0.220484
4	0.565549	-0.642467	-1.132395	0.892860
5	1.932905	0.825990	-0.655927	1.312352

Initialization of Dataframe

```
import pandas as pd
import numpy as np

dates = pd.date_range("20130101", periods=6)

df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
```

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

	A	B	C	D
2013-01-01	0.075437	-2.138378	0.220500	-2.622102
2013-01-02	-0.123794	-0.014208	1.028106	0.571388
2013-01-03	1.323161	-2.625521	0.385828	-1.700893
2013-01-04	-0.533021	-2.261734	-0.144942	1.625915
2013-01-05	-0.268247	0.066943	0.129547	1.406832
2013-01-06	0.633518	-1.297509	-1.086282	2.369941

Initialization of Dataframe

Using a dictionary

```
df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
        "F": "foo",
    }
)
```

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

```
df2.dtypes
```

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

Adding a new column

```
df2 = df.copy()
df2["E"] = ["one", "one", "two", "three", "four", "three"]
```

	A	B	C	D	E
2013-01-01	-0.961137	-1.190816	0.755875	-0.319302	one
2013-01-02	-0.702419	0.486172	-0.880247	1.420582	one
2013-01-03	1.106880	-0.236240	0.636290	1.761212	two
2013-01-04	-0.049483	-1.299836	0.070177	-1.729362	three
2013-01-05	-0.328242	-0.426409	0.202379	0.149186	four
2013-01-06	1.536557	-1.192452	-0.531820	0.852880	three

```
s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range("20130102", periods=6))
df["F"] = s1
```

	A	B	C	D	F
2013-01-01	-0.961137	-1.190816	0.755875	-0.319302	NaN
2013-01-02	-0.702419	0.486172	-0.880247	1.420582	1.0
2013-01-03	1.106880	-0.236240	0.636290	1.761212	2.0
2013-01-04	-0.049483	-1.299836	0.070177	-1.729362	3.0
2013-01-05	-0.328242	-0.426409	0.202379	0.149186	4.0
2013-01-06	1.536557	-1.192452	-0.531820	0.852880	5.0

`df.reindex`

Examining the dataframe

```
df.head()
df.tail(3)
len(df)
df.index
df.columns
df.describe()
df.T
df2["E"].isin(["two", "four"])

df.to_numpy()

df.sort_index(axis=1, ascending=False)
df.sort_index(axis=0, ascending=False)

df.sort_values(by="B")
df.sort_values(axis=1,by="2013-01-02")
```

Basic selection

5 selection methods

(1) Using Numpy method (Not recommended in Pandas)

ex) `df[0:3], df["A"], df.B, df.C.values,`
`df.D.values[:1]`

(2) .loc : index by label (Indexing is ***inclusive***!!!)

ex) `df.loc["20130101"], df.loc[:,["A","B"]],`
`df.loc["20130102":"20130104", ["A", "B"]]`

(3) .at : fast access to a single element my label

ex) `df.at["20130102", "A"]`

(4) .iloc : numeric indexing (similar to Numpy style)

ex) `df.iloc[3], df.iloc[3:5, 0:2],`
`df.iloc[[1, 2, 4], [0, 2]], df.iloc[:, 1:3]`

(5) .iat : fast access to a single element my numer

ex) `df.iat[1, 1]`

Boolean indexing

ex)

```
df[df["A"] > 0]
df[df["A"] > 0] = 0
df[df > 0]
df.loc[  df["A"]>0, ["B","D"]  ]

tmp = df[ (df["A"] < 0) & (df["C"]<0)    ]
tmp.iloc[0,0]=0   # Warning

tmp = df[ (df["A"] < 0) & (df["C"]<0)    ].copy()
tmp.iloc[0,0]=0   # No warning
```

Modifying the elements

ex)

```
df.at[dates[0], "A"] = 0
df.iat[0, 1] = 0

# similar to broadcasting
df.loc[:, "D"] = np.array([5] * len(df))

# “where” operation in Pandas:
# This is a special operation (i.e., special syntax)
df2 = df.copy()
df2[df2 > 0] = -df2
```

Dealing with missing data

Let's first make a fake dataframe with missing data

```
dates = pd.date_range("20130101", periods=6)
df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ["E"])
df1.loc[dates[0] : dates[1], "E"] = 1
```

ex)

```
pd.isna(df1)
```

```
df1.dropna(how="any")
```

```
df1[  pd.isna(df1)  ] = 1000
```

Merging

ex)

```
df = pd.DataFrame(np.random.randn(10, 4))
a, b, c = df[:3], df[3:7], df[7:]
pd.concat([a,b,c])
```

```
left = pd.DataFrame({"key": ["foo", "bar"], "lval": [1, 2]})
right = pd.DataFrame({"key": ["foo", "bar"], "rval": [4, 5]})
pd.merge(left, right, on="key")
```

```
left = pd.DataFrame({"key": ["foo", "foo"], "lval": [1, 2]})
right = pd.DataFrame({"key": ["foo", "foo"], "rval": [4, 5]})
pd.merge(left, right, on="key")
```

Some basic operations

ex)

```
df.mean()
df.mean(1)

df.describe()

df.sum(axis=1)
df['passengers'] = df.sum(axis=1)

s = pd.Series(np.random.randint(0, 7, size=10))
s.value_counts()
```

Grouping (operations are performed for each group)

```
df = pd.DataFrame(
        {
            "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
            "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
            "C": np.random.randn(8),
            "D": np.random.randn(8),
        }
    )
```

```
Ex)  df.groupby("A").sum()
     df.groupby(["A","B"]).sum()
```

Plotting

```
df = pd.DataFrame(
    np.random.randn(1000, 4),
    index=pd.date_range("1/1/2000", periods=1000),
    columns=["A", "B", "C", "D"]
)

Ex) df.cumsum().plot()
```

File I/O

In real situation, you will use these most of the time, rather than generating individual dataframes by yourselves. So, it is worth checking out other examples of them, especially "**`read_csv`**".

```
df.to_csv("foo.csv")
df3 = pd.read_csv("foo.csv")
```